

Matematické Základy Informatiky (FI: IB000)

Prof. RNDr. Petr Hliněný, Ph.D.

`hlineny@fi.muni.cz`

31. srpna 2015



Obsažný a dobře přístupný úvod do nezbytných formálních matematických základů moderní informatiky, doplněný řadou interaktivních cvičení v IS MU.

Výukový text pro předmět IB000 na FI MU od roku 2012, navazující zčásti na původní výukové texty Úvodu do Informatiky z minulých let.

0.1 O tomto textu a jeho studiu

Vážení čtenáři,

dostává se vám do rukou výukový text Matematické Základy Informatiky, který je primárně určený pro studenty stejnojmenného předmětu na FI MU Brno. Seznamuje čtenáře s několika formálními matematickými oblastmi důležitými pro úspěšné studium moderní informatiky.

Náš text svým obsahem volně navazuje na původní slidy předmětu IB000 sepsané A. Kučerou do roku 2005 a rozšířené autorem v letech 2006–11 o volný text a komentáře. Od roku 2012 však byl obsah některých pasáží podstatně změněn či přímo vyměněn za nový – nejzřetelnější změnou je zahrnutí úvodu do teorie grafů. Mimo textu samotného (jak jej zde vidíte) jsou z téhož zdoje vytvářeny i přednáškové slidy předmětu, které najdete například v IS MU. Slidy však pochopitelně obsahují jen část textu a jsou jinak formátovány.

Učební text je psán strukturovaným matematickým přístupem, s velkým důrazem na přesnost a formalitu vyjadřování v nutných partiích. Na druhou stranu jsou strohá matematická vyjádření pokud možno doplněna obsáhlými neformálními komentáři, které mají studentům ulehčit pochopení látky. V žádném případě však čtenáři nemají zaměňovat neformální komentáře za matematické definice – v případě nejasností vždy platí to, co přesně říká formální definice.

Interaktivní osnova

<http://is.muni.cz/e1/1433/podzim2015/IB000/index.qwarp>

Nedílnou součástí celého výukového textu jsou interaktivní osnova předmětu IB000 v IS MU a z ní odkazované online odpovědníky sloužící k procvičování probrané látky na jednoduchých i složitějších příkladech. To je také důvod, proč tento text obsahuje jen poskrovnu řešených příkladů k probírané látce; od každého studenta očekáváme, že si látku *bude procvičovat online na zmíněných odpovědnících*, obsahujících až tisíce příkladů desítek typů. K praktickému pochopení přednesených znalostí i k jejich budoucím aplikacím je takové procvičení nezbytné(!). V návaznosti na tyto odpovědníky doporučujeme studentům diskutovat o probírané látce a svých (ne)úspěších s cvičícími předmětu i na předmětovém diskuzním fóru v IS.

Obsah

0.1	O tomto textu a jeho studiu	ii
1	Základní formalismy matematiky	1
1.1	Úvod do matematického dokazování	1
1.2	Význam matematických tvrzení	3
1.3	Tvoření matematických důkazů	4
1.4	Výroky a základ logiky	6
1.5	Střípky matematické logiky	7
2	Důkazové techniky, Indukce	14
2.1	Přehled základních důkazových technik	14
2.2	Věty typu „tehdy a jen tehdy“	16
2.3	Matematická indukce	17
2.4	Komentáře k matematické indukci	19
3	Množiny a množinové operace	22
3.1	Pojem množiny	22
3.2	Množinové operace	23
3.3	Porovnávání a určení množin	26
3.4	Posloupnosti a rekurentní vztahy	28
4	Relace a jejich použití	30
4.1	Relace a funkce nad množinami	30
4.2	Reprezentace konečných relací	32
4.3	Vlastnosti binárních relací	33
4.4	Inverzní relace a skládání relací	35
4.5	Skládání relací „v praxi“	36
5	Ekvivalence, Uspořádané množiny	39
5.1	Relace ekvivalence	39
5.2	Rozklady a jejich vztah k ekvivalencím	40
5.3	Uspořádání a uspořádané množiny	41
5.4	Další pojmy uspořádaných množin	44
5.5	Relace předuspořádání	46
6	Funkce a skládání, Induktivní definice	49
6.1	Vlastnosti funkcí	49
6.2	Skládání funkcí, permutace	51
6.3	Induktivní definice množin a funkcí	53
6.4	Uzávěry relací	57
7	Pojem grafu	60
7.1	Definice grafu	60
7.2	Podgrafy a Isomorfismus	63
7.3	Souvislost grafů, komponenty	67
7.4	Stromy – grafy bez kružnic	69
7.5	Použití a implementace grafů	70

8	Procházení grafu a odvozené úlohy	73
8.1	Jak obecně projít souvislý graf	73
8.2	Vzdálenost v grafu	76
8.3	Hledání nejkratší cesty	77
8.4	Problém minimální kostry	80
9	Orientované grafy, Toky v sítích	84
9.1	Základní pojmy orientovaných grafů	84
9.2	Definice sítě a toku	87
9.3	Nalezení maximálního toku	88
9.4	Zobecněné použití sítí	92
10	Formalizace a důkazy pro algoritmy	96
10.1	Formální popis algoritmu	96
10.2	O „správnosti“ a dokazování programů	98
10.3	Jednoduché indukční dokazování	99
10.4	Rekurzivní algoritmy	100
11	Pokročilé dokazování nad algoritmy	103
11.1	Dokazování konečnosti algoritmu a programu	103
11.2	Přehled technik důkazu indukcí	106
11.3	Zajímavé algoritmy aritmetiky	109
11.4	Dynamický algoritmus	111
12	Nekonečné množiny, Zastavení algoritmu	114
12.1	O nekonečných množinách a kardinalitě	114
12.2	„Naivní“ množinové paradoxy	116
12.3	Algoritmická neřešitelnost problému zastavení	118

1 Základní formalismy matematiky

Úvod

Začněme nejprve několika obecnými poznámkami ke smyslu a směřování celého našeho předmětu: Jak sami poznáte, studium informatiky neznamená jen „naučit se nějaký programovací jazyk“, nýbrž zahrnuje celý soubor dalších relevantních předmětů, mezi nimiž najdeme i matematicko–teoretické (formální) základy moderní informatiky. Právě odborný nadhled nad celou informatikou včetně nezbytné formální teorie nejspíše odliší „řadového programátora“, kterých jsou dnes spousty i bez VŠ vzdělání, od skutečného a mnohem lépe placeného iformatického experta.

A na tomto místě přichází náš odborný předmět Matematické Základy Informatiky, který vás právě na studium formálních základů moderní informatiky připraví. Výklad začínáme poněkud zostrá – zdůrazněním přesného matematického vyjadřování, výrokovou logikou a principy logických úsudků a matematických důkazů. Nelekejte se však hned a s chutí se dejte do studia, po překonání prvotního šoku to nebude až tak obtížné. . .

Cíle

Prvním cílem této lekce je rozebrat struktury matematických tvrzení (vět) a jejich formálních důkazů. V druhé řadě se naučíte chápat smysl matematických vyjádření formálním pohledem výrokové logiky a pracovat s touto formalizací, včetně základního pochopení významu kvantifikace.

1.1 Úvod do matematického dokazování

Matematika (a tudíž i teoretická informatika jako její součást) se vyznačuje velmi přísnými formálními požadavky na korektnost argumentace. Takto korektně postavená argumentace vede od přijatých předpokladů v elementárních krocích směrem k požadovanému závěru (a nikdy ne naopak!).

Definice 1.1. *Matematický důkaz* .

Uvažme matematickou větu (neboli tvrzení) tvaru

„Jestliže platí předpoklady, pak platí závěr“.

Důkaz této věty je konečná posloupnost tvrzení, kde

- každé tvrzení je buď
 - * předpoklad, nebo
 - * obecně přijatá „pravda“ – *axiom*, nebo
 - * plyne z předchozích a dříve dokázaných tvrzení podle nějakého „akceptovaného“ logického principu – *odvozovacího pravidla*;
- poslední tvrzení je *závěr*.

Komentář: O potřebné úrovni formality matematických důkazů a o běžných důkazových technikách se dozvíme dále v této a příští lekci. Pro úplný začátek si jen celou problematiku uvedeme názornými ukázkami. Čtěte si tyto ukázky pečlivě, i pokud jste se již s matematickým dokazováním setkali a (či nebo) vám následující příklady připadají zcela primitivní. Později, u složitějších důkazů, se právě k takovým jednoduchým ukázkám můžete vracet a lépe na nich pochopit, jak to správně v matematice „chodí“.

Příklad 1.2. Uvažujme následující matematické tvrzení (které jistě už znáte).

Věta. Jestliže x je součtem dvou lichých čísel, pak x je sudé.

Poznámka pro připomenutí:

- *Sudé* číslo je celé číslo dělitelné 2, tj. tvaru $2k$.
- *Liché* číslo je celé číslo nedělitelné 2, tj. tvaru $2k + 1$.

Důkaz postupuje v následujících *formálních krocích*:

tvrzení	zdůvodnění
1) $a = 2k + 1, k$ celé	předpoklad
2) $b = 2l + 1, l$ celé	předpoklad
3) $x = a + b = 2k + 2l + 1 + 1$	z 1,2) a komutativity sčítání (axiom)
4) $x = 2(k + l) + 2 \cdot 1$	ze 3) a distributivnosti násobení (axiom)
5) $x = 2(k + l + 1)$	ze 4) a opět distributivnosti násobení
6) $x = 2m, m$ celé	z 5) a $m = k + l + 1$ je celé číslo (axiom) □

Příklad 1.3. Dokažte následující tvrzení:

Věta. Jestliže x a y jsou racionální čísla pro která platí $x < y$, pak existuje racionální číslo z pro které platí $x < z < y$.

Důkaz po krocích (s již trochu méně formálním zápisem) zní:

- 1) Necht' $z = \frac{x+y}{2} = x + \frac{y-x}{2} = y - \frac{y-x}{2}$.
- 2) Číslo z je racionální, neboť x a y jsou racionální.
- 3) Platí $z > x$, neboť $\frac{y-x}{2} > 0$.
- 4) Dále platí $z < y$, neboť opět $\frac{y-x}{2} > 0$.
- 5) Celkem $x < z < y$.

Jak čtenář asi sám vidí, tento strohý formální zápis (i když matematicky kompletní) si zaslouží alespoň krátký vysvětlující komentář. Takový komentář, ať už se objeví před nebo hned po formálních argumentech, sice není součástí důkazu samotného, velmi však napomáhá správnému pochopení a má své nezastupitelné místo. Nezapomínejte na to ve svých vlastních budoucích matematických důkazech.

Konkrétně v tomto příkladě je velmi vhodné poznamenat, že klíčový krok (1) popisuje námi vymyšlenou (prostě uhodnutou) algebraickou konstrukci, která vede k požadovanému číslu z . Zbylé kroky (2–5) pak jen snadno zdůvodňují, že nalezené z má všechny požadované vlastnosti. □

Poznámka. Alternativní matematické formulace Věty z Příkladu 1.3 mohou znít:

- Pro každé $x, y \in \mathbb{Q}$, kde $x < y$, existuje $z \in \mathbb{Q}$ takové, že $x < z < y$.
- Množina racionálních čísel je *hustá*.

1.2 Význam matematických tvrzení

První krok formálního důkazu je uvědomit si, *co tvrdí věta*, která se má dokázat; tedy co je *předpoklad* a co *závěr* dokazovaného tvrzení. *Pravdivost* takového tvrzení pak je třeba chápat v následujícím významu:

Pro každou situaci, ve které jsou splněny všechny předpoklady, ()
je platný i závěr tvrzení.*

Komentář: Příklady běžné formulace matematických vět:

- * Konečná množina má konečně mnoho podmnožin.
- * $\sin^2(\alpha) + \cos^2(\alpha) = 1$.
- * Graf je rovinný, jestliže neobsahuje podrozdělení K_5 nebo $K_{3,3}$.

Co přesně nám uvedené matematické věty říkají? Všimněme si, že mnohdy jsou některé „samozřejmé“ předpoklady vět vynechány, například, že α ve druhém bodě je úhel, nebo se tvrzení implicitně odvolávají na obecně známé definice, například, co je to graf K_5 ve třetím bodě. Často nám k lepšímu pochopení toho, co je tvrzeno a je třeba dokázat, pomůže pouhé rozepsání definic pojmů, které se v dané větě vyskytují.

O pravdivosti implikace

Matematická tvrzení jsou často tzv. složená („platí ... a/nebo ...“) a v tom případě je třeba umět správně vyvodit celkovou platnost z platnosti jednotlivých elementů. Tomuto se budeme věnovat přesněji v Oddíle 1.5, ale již nyní je potřeba se zamyslet a správně pochopit, jaký je logický význam matematického tvrzení vysloveného typicky formou *implikace* („jestliže ..., pak ...“). Z předchozího (*) totiž vyplývá, že pokud předpoklady nejsou splněny nebo jsou sporné, tak celé tvrzení je platné bez ohledu na pravdivost závěru!

Příklad 1.4. *Je pravdivé následující matematické tvrzení?*

Věta. *Mějme dvě kuličky, červenou a modrou. Jestliže červená kulička je těžší než modrá a zároveň je modrá kulička těžší než ta červená, tak jsou obě kuličky ve skutečnosti zelené.*

„To přece nemůže být pravda, jak může být jedna kulička těžší než druhá a naopak zároveň? Jak mohou být nakonec obě zelené? To je celé nějaká blbost...“

Ano, výše uvedené jsou typické laické reakce na uvedenou větu. Přesto však tato věta pravdivá je! Stačí se vrátit o kousek výše ke kritériu – *Pro každou situaci, ve které jsou splněny všechny předpoklady, je platný i závěr tvrzení* – které je zjevně naplněno. Nenaleznete totiž situaci, ve které by byly splněny oba předpoklady zároveň, a tudíž ve všech takových neexistujících situacích si můžete říkat cokoliv, třeba že kuličky jsou zelené. □

Příklad 1.5. *Anna a Klára přišly na přednášku a usadily se do lavic. Proč je pravdivé toto matematické tvrzení?*

Věta. *Jestliže Anna sedí v první řadě lavic a zároveň Anna sedí v poslední řadě lavic, tak Klára nesedí ve druhé řadě lavic.*

Opět je třeba se hluboce zamyslet nad významem předpokladů a závěru, ale tentokrát není situace předpokladů tak triviálně sporná, jako v Příkladu 1.4. Kdy tedy nastávají oba předpoklady zároveň? Když první řada lavic je zároveň řadou poslední. Neboli posluchárna má jen (nejvýše) jednu řadu lavic a Klára tudíž v druhé řadě nemůže sedět. Důkaz je hotov. \square

1.3 Tvoření matematických důkazů

V předchozím jsme viděli několik různorodých ukázek matematických důkazů, od zcela formálního přístupu po velmi uvolněný text. Přirozenou otázkou teď je, jak „moc formální“ mají správné matematické důkazy vlastně být?

- Záleží, komu je důkaz určen — *konzument* musí být schopen „snadno“ ověřit korektnost každého tvrzení v důkazu a plně pochopit, z čeho vyplývá.
- Je tedy hlavně na vás zvolit tu správnou úroveň formálnosti zápisu vět i důkazů podle situace.
- Avšak vůbec *neplatí*, že čím více formálních matematických symbolů v důkaze použijete místo běžného jazyka, tím by byl důkaz přesnější – často bývá výsledek právě opačný.

A jak na ten správný matematický důkaz máme přijít?

- No... , nalézání matematických důkazů je tvůrčí činnost, která není vůbec snadná a jako taková vyžaduje tvůrčí (přímo „*umělecké*“) matematické vlohy. Přesto se jí alespoň trochu musíte přiučít.

Dokazovat či vyvracet tvrzení?

Představme si, že našim úkolem je *rozhodnout platnost* matematického tvrzení. Neboli nám není řečeno, že tvrzení platí, ale mohou nastat obě možnosti platí / neplatí. Jak pak matematicky správně zdůvodníme nalezenou odpověď?

- Záleží na odpovědi samotné... Pokud je to ANO (platí), prostě podáme důkaz tvrzení podle výše uvedených zvyklostí. Pokud je odpověď NE, tak naopak podáme důkaz *negace* daného tvrzení.

Poměrně častým případem je matematická věta T , která tvrdí nějaký závěr pro širokou oblast vstupních možností. Potom je postup následující:

- Pokud T platí, nezbyvá než podat *vyčerpávající důkaz* platnosti pro všechny vstupy.
- Avšak pokud T je nepravdivá, stačí *uhodnout* vhodný vstupní *protipříklad* a jen pro něj dokázat, že závěr tvrzení není platný.

Komentář: Již podruhé (po Příkladu 1.3) se dostáváme k fenoménu „*uhodnutí*“ některé součásti důkazu. Jak se k němu máme stavět (je to vůbec dovoleno udělat)? Ale ano, pokud váš důkaz bude matematicky správný, nikdo nebude namítat, že jste v něm něco jen šťastně uhodli. V takovém správném „*uhodnutí*“ věci právě spočívá ta zmíněná tvůrčí stránka vytváření matematických důkazů.

A nyní se znovu vraťte na začátek Oddílu 1.2 a srovnajte si výše uvedené se základními poznatky o významu matematických vět...

Příklad 1.6. Rozhodněte platnost následujícího tvrzení: Pro všechna reálná x platí

$$x^2 + 3x + 2 \geq 0.$$

Důkaz: Standardními algebraickými postupy si můžeme upravit vztah na $x^2 + 3x + 2 = (x + 1) \cdot (x + 2) \geq 0$. Co nám z něj vyplývá? Například to, že k porušení daného tvrzení stačí volit x tak, aby jedna ze závorek byla kladná a druhá záporná. To nastane třeba pro $x = -\frac{3}{2}$.

Pro vyvrácení tvrzení nám tedy stačí začít volbou protipříkladu $x = -\frac{3}{2}$ (není nutno zdůvodňovat, jak jsme jej „uhodli“!) a následně dokázat úpravou

$$x^2 + 3x + 2 = (x + 1) \cdot (x + 2) = \left(-\frac{3}{2} + 1\right) \cdot \left(-\frac{3}{2} + 2\right) = \left(-\frac{1}{2}\right) \cdot \left(+\frac{1}{2}\right) = -\frac{1}{4} < 0.$$

Dané tvrzení tudíž není platné. □

Konstruktivní a existenční důkazy

Z hlediska praktické využitelnosti je vhodné rozlišovat tyto dvě kategorie důkazů (třebaže z formálně-matematického pohledu mezi nimi kvalitativní rozdíl není).

- Důkaz z Příkladu 1.3 je *konstruktivní*. Dokázali jsme nejen, že číslo z existuje, ale podali jsme také návod, jak ho pro dané x a y sestrojít.
- *Existenční* důkaz je takový, kde se prokáže existence nějakého objektu *bez toho*, aby byl podán použitelný návod na jeho konstrukci. Třebaže to může vypadat divně, nějak se snažit dokazovat existenci něčeho, co nedokážeme sestrojít, ale takové situace skutečně v matematice nastávají a existenční důkazy jsou v nich užitečné.

Příklad 1.7. Čistě existenčního důkazu.

Věta. Existuje program, který vypíše na obrazovku čísla tažená ve 45. tahu sportky v roce 2015.

Důkaz: Existuje pouze konečně mnoho možných výsledků losování 45. tahu sportky v roce 2015. Pro každý možný výsledek existuje program, který tento daný výsledek vypíše na obrazovku. Mezi těmito programy je tedy jistě ten, který vypíše právě ten výsledek, který bude ve 45. tahu sportky v roce 2015 skutečně vylosován. □

Komentář: To je ale „podvod“, že? A přece *není*... Formálně správně to je prostě tak a tečka.

Příklad 1.8. Pravděpodobnostního existenčního důkazu.

Věta. Na dané množině bodů je zvoleno libovolně n^2 podmnožin, každá o n prvcích. Dokažte pro dostatečně velká n , že všechny body lze obarvit dvěma barvami tak, aby žádná množina nebyla jednobarevná.

Důkaz: U každého bodu si „hodíme mincí“ a podle výsledku zvolíme barvu červeně nebo modře. (Nezávislé volby s pravděpodobností $\frac{1}{2}$.) Pravděpodobnost, že zvolených n bodů vyjde jednobarevných, je jen $\frac{2}{2^n} = 2^{1-n}$.

Pro n^2 podmnožin tak je pravděpodobnost, že některá z nich vyjde jednobarevná, shora ohraničená součtem

$$\underbrace{2^{1-n} + \dots + 2^{1-n}}_{n^2} = \frac{n^2}{2^{n-1}} \rightarrow 0.$$

Jelikož je toto číslo (pravděpodobnost) pro $n \geq 7$ menší než 1, bude existovat obarvení bez jednobarevných zvolených podmnožin. \square

Komentář: Zkuste si sami v obou posledních příkladech přijít s konstruktivním důkazem, tj. takovým, který hledané řešení sestrojí. Pokud se vám to povede, budete skutečně dobří (a možná také bohatí)...

1.4 Výroky a základ logiky

Důležitým „pevným mostem“ mezi běžnou mluvou a přesným matematickým formalismem je následující pojem výroku. Jeho správné uchopení nám pomůže při chápání významu matematických výroků a v práci s nimi (mimo bezprostřední ukázky se třeba podívejte na mechanický postup negace složeného výroku v závěru lekce). To je úkolem tohoto oddílu.

Definice 1.9. *Výrok* v přirozené mluvě:

V běžné mluvě za *výrok* považujeme (každé) tvrzení, o kterém má smysl platně prohlásit, že je *buď* pravdivé *nebo* nepravdivé.

Komentář: Ukážeme si několik příkladů – které z nich jsou výroky?

- * Dnes už v Brně přšlo.
- * Předmět FI:IB000 se vyučuje v prvním ročníku.
- * Platí $2 + 3 = 6$.
- * To je bez problémů. (Co?)
- * Platí $x > 3$.
- * Pro každé celé číslo x platí, že $x > 3$.

Všimněte si, že pravdivost výroku by mělo být možné rozhodnout bez skrytých souvislostí (kontextu), a proto čtvrtý a pátý příklad za výroky nepovažujeme. Poslední příklad již zase výrokem je, neboť obor hodnot x je jednoznačně vymezen tzv. kvantifikací.

Z více jednoduchých výroků vytváříme výroky složitější pomocí tzv. *logických spojek*.

Komentář: Následuje několik dalších příkladů.

- * Množina $\{a, b\}$ má více než jeden prvek a není nekonečná.
- * Jestliže Karel váží přes 90 kg, nejedu s ním výtahem.
- * Jestliže má tato kráva 10 nohou, pak mají všechny domy modrou střechu.

Zastavme se na chvíli nad posledním výrokem. Co nám říká? Je pravdivý? Skutečně mají všechny domy modrou střechu a před námi stojí kráva s 10 nohama? (Ano, výrok je pravdivý, viz definice či pravdivostní tabulka implikace.)

Schopnost porozumět podobným větám je součástí lidského způsobu uvažování a z tohoto hlediska nemá přímou souvislost s matematikou (je to „*přirozená logika*“). *Formální (matematická) logika* pak v podobném duchu definuje jazyk matematiky a přitom odstraňuje nejednoznačnosti přirozeného jazyka.

Ukázka formální práce s výroky

Jako příklad užitečnosti matematické formalizace slovních výroků si rozebereme následující logickou „hádanku“, navazující na téma Příkladu 1.5.

Příklad 1.10. *Mějme trojici studentů X, Y, Z sedících v posluchárně na přednášce IB000 takových, že pro ně platí:*

- a) X sedí v první řadě,
- b) Y sedí v řadě hned za X ,
- c) Z sedí ve stejné řadě jako Y ,
- d) Z sedí ve druhé řadě.

Věta. *Jestliže zároveň platí (a),(b),(c),(d), pak student Z sedí ve druhé řadě.*

Tvrzení je samozřejmě triviální díky předpokladu (d), avšak úkolem je zjistit, jaké všechny minimální výběry z předpokladů (a),(b),(c),(d) ještě zajistí platnost uvedeného tvrzení.

Pro začátek osvětlíme (blíže viz Oddíl 5.4) význam slova *minimální*, kdy je myšlen výběr předpokladů takový, že odebráním libovolného z nich již platnost uvedeného tvrzení „ Z sedí ve druhé řadě“ bude porušena.

Okamžitou odpovědí je samotný předpoklad (d), který je zřejmě minimální (bez předpokladů si může Z sednout i do první řady) a dostačující. Je to však úplná odpověď, nebo lze vybrat i jiné minimální předpoklady? Ano, ještě lze vybírat jinak – po vynechání (d) stále postačuje kombinace předpokladů (a),(b),(c) k vyslovení závěru „ Z sedí ve druhé řadě“ (je to jasné?). Na druhou stranu, pokud kterýkoliv z (a),(b),(c) vynecháme, rozebráním těchto tří možností najdeme platná rozesazení, při kterých Z v druhé řadě nebude, takže se opět jedná o výběr minimální. Například pokud vynecháme (b), může Y sedět v první řadě spolu s X a stejně tak Z . Dořešte si sami zbylé dvě možnosti.

Pozor, ještě k úplnému vyřešení úlohy zbývá poslední důležitý krok – zdůvodnit, proč žádný jiný minimální výběr předpokladů úloze nevyhovuje. Zde je třeba rozebrat možnosti: Pokud náš hypotetický jiný výběr předpokladů obsahuje (d), pak nebude minimální, neboť lze odebrat cokoli jiného, pokud (d) zůstane. Naopak výběr neobsahující (d) musí podle výše uvedeného obsahovat všechny (a),(b),(c), neboť jinak není dostačující. A tím jsme hotovi; všechny minimální výběry řešící úlohu jsou dva, (a),(b),(c) a samotné (d). \square

1.5 Stríčky matematické logiky

Všimněte si, že podle Definice 1.9 každému výroku běžné mluvy prostě můžeme přiřadit logickou hodnotu 0 (*false*) nebo 1 (*true*) a dále se nestarat o jazykový význam... Proto jazykové výroky v matematice můžeme nahradit *výrokovými proměnnými*, které značíme velkými písmeny A, B, C, \dots a přiřadíme jim hodnotu 0 nebo 1.

Definice: *Výroková formule* (značíme $\varphi, \sigma, \psi, \dots$) vzniká z výrokových proměnných pomocí závorek a logických spojek \neg *negace* a \Rightarrow *implikace*. Zároveň používáme v zápise následujících zkratk

- * $\varphi \vee \psi$ (*disjunkce* / „nebo“) je jiný zápis formule $\neg\varphi \Rightarrow \psi$,
- * $\varphi \wedge \psi$ (*konjunkce* / „a“) je jiný zápis formule $\neg(\neg\varphi \vee \neg\psi)$,

* $\varphi \Leftrightarrow \psi$ (ekvivalence) je jiný zápis formule $(\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)$.

Poznámka: Uvedená definice je instancí tzv. *induktivní definice*, jíž se budeme blíže věnovat v Oddíle 6.3, a proto ji zde uvádíme jen v hodně zjednodušené podobě.

Komentář: Při zápise výrokových formulí je potřeba dávat pozor na správné závorkování, aby formule měla jednoznačný význam. Na intuitivní úrovni to ilustrujeme takto:

Správně $A, (A) \Rightarrow (B), A \Rightarrow B, \neg A \Rightarrow B, A \vee B \vee \neg C$

a nesprávně $A \Rightarrow B \Rightarrow C$ — znamená to $(A \Rightarrow B) \Rightarrow C$ nebo $A \Rightarrow (B \Rightarrow C)$?

Matematický význam logickým formulím pak dodává následující definice.

Definice 1.11. Sémantika (význam) výrokové logiky.

Nechť *valuace* (ohodnocení) je funkce $\nu : Prom \rightarrow \{0, 1\}$ na všech (dotčených) výrokových proměnných. Pro každou valuaci ν definujeme funkci $\mathcal{S}_\nu(\sigma)$, *vyhodnocení* formule σ , induktivně (tj. po krocích) takto:

* $\mathcal{S}_\nu(A) = \nu(A)$ pro každé $A \in Prom$.

* $\mathcal{S}_\nu(\neg\varphi) = \begin{cases} 1 & \text{jestliže } \mathcal{S}_\nu(\varphi) = 0; \\ 0 & \text{jinak.} \end{cases}$

* $\mathcal{S}_\nu(\varphi \Rightarrow \psi) = \begin{cases} 0 & \text{jestliže } \mathcal{S}_\nu(\varphi) = 1 \text{ a } \mathcal{S}_\nu(\psi) = 0; \\ 1 & \text{jinak.} \end{cases}$

Poznámka: Tento předpis (Def. 1.11) podává nejen *definici* funkce \mathcal{S}_ν , ale také návod na to, jak ji pro daný argument *vypočítat*.

Pro odvozené logické spojky disjunkce, konjunkce a ekvivalence dostaneme následující zcela přirozený výsledek (což potvrzuje, že jsme definici sémantiky zvolili správně).

Tvrzení 1.12. *Přímým důsledkem Definice 1.11 je následovné:*

* $\mathcal{S}_\nu(\varphi \vee \psi) = 1$ právě když $\mathcal{S}_\nu(\varphi) = 1$ nebo $\mathcal{S}_\nu(\psi) = 1$.

* $\mathcal{S}_\nu(\varphi \wedge \psi) = 1$ právě když $\mathcal{S}_\nu(\varphi) = 1$ a současně $\mathcal{S}_\nu(\psi) = 1$.

* $\mathcal{S}_\nu(\varphi \Leftrightarrow \psi) = 1$ právě když platí jedna z následujících podmínek

- $\mathcal{S}_\nu(\varphi) = 1$ a současně $\mathcal{S}_\nu(\psi) = 1$,
- $\mathcal{S}_\nu(\varphi) = 0$ a současně $\mathcal{S}_\nu(\psi) = 0$.

Pravdivostní tabulky

V praxi často vyhodnocení \mathcal{S}_ν logické výrokové formule zapisujeme do tzv. *pravdivostní tabulky*. Tato tabulka typicky má sloupce pro jednotlivé proměnné, případně „meziformule“ (pomůcka pro snazší vyplnění) a výslednou formuli. Řádků je 2^p (počet valuací), kde p je počet použitých proměnných.

Pro naše účely postačí uvést pravdivostní tabulku instruktážním příkladem. Čtenáři nechtě si vyplní podle Definice 1.11 vlastní pravdivostní tabulky dalších výrokových formulí, viz také příslušné cvičné odpovědníky v IS MU.

Příklad 1.13. Jaká je pravdivostní tabulka pro formuli $(A \Rightarrow B) \vee B \vee C$?

A	B	C	$A \Rightarrow B$	$(A \Rightarrow B) \vee B \vee C$
0	0	0	1	1
0	1	0	1	1
1	0	0	0	0
1	1	0	1	1
0	0	1	1	1
0	1	1	1	1
1	0	1	0	1
1	1	1	1	1

□

Splnitelnost formulí a tautologie

Definice: Formule $\varphi \in \Phi$ je *splnitelná*, pokud pro některou valuaci ν platí, že $\mathcal{S}_\nu(\varphi) = 1$. Formule je *nesplnitelná* (říká se také *kontradikce*), pokud není splnitelná. Formule $\varphi \in \Phi$ je vždy pravdivá, neboli výroková *tautologie*, psáno $\models \varphi$, pokud pro každou valuaci ν platí, že $\mathcal{S}_\nu(\varphi) = 1$.

Řekneme, že dvě formule $\varphi, \psi \in \Phi$ jsou *ekvivalentní*, právě když $\models \varphi \Leftrightarrow \psi$.

Tvrzení 1.14. Následující formule jsou tautologiemi:

- * $\models A \vee \neg A$
- * $\models \neg \neg A \Leftrightarrow A$
- * $\models (A \wedge (A \Rightarrow B)) \Rightarrow B$
- * $\models (\neg B \Rightarrow \neg A) \Rightarrow (A \Rightarrow B)$
- * $\models (\neg A \Rightarrow (B \wedge \neg B)) \Rightarrow A$

Komentář: Jak poznáme tautologii v pravdivostní tabulce? Snadno, v posledním sloupci jsou samé hodnoty 1.

Kvantifikace a predikátová logika

Výše popsaná výroková logika je velmi omezená faktem, že každý výrok musí být (tzv. absolutně) vyhodnocen jako pravda nebo nepravda. Co když však chceme zpracovat tvrzení typu „den D v Brně přšelo“? Jeho pravdivostní hodnota přece závisí na tom, co dosadíme za den D, a tudíž jej nelze považovat za výrok výrokové logiky.

- Predikátová logika pracuje s *predikáty*. Predikáty jsou „parametrizované výroky“, které jsou buď pravdivé nebo nepravdivé pro každou konkrétní volbu parametrů. Výrokové proměnné lze chápat jako predikáty bez parametrů.
- *Predikátová logika* je tak *obecnější než logika výroková*; každá formule výrokové logiky je i formulí predikátové logiky, ale ne obráceně.

Komentář: Pro neformální přiblížení si uvedeme několik ukázek predikátů:

- * $x > 3$ (parametrem je zde $x \in \mathbb{R}$),
- * čísla x a y jsou nesoudělná (parametry $x, y \in \mathbb{N}$),

* obecně jsou predikáty psány $P(x, y)$, kde x, y jsou libovolné parametry.

Definice: Z predikátů lze vytvářet *predikátové formule* pomocí už známých (viz Definice 1.11) výrokových spojek a následujících tzv. *kvantifikátorů*:

- $\forall x . \varphi$ „pro každou volbu parametru x platí formule φ “
nebo jinak řečeno „pro všechna / kterékoliv x platí formule φ “,
- $\exists x . \varphi$ „existuje alespoň jedna volba parametru x , pro kterou platí φ “
nebo jinak řečeno „pro nějaké x platí formule φ “.

Komentář: Pozorného čtenáře napadne, ze které množiny že „volíme parametr x “? To je v matematické logice obvykle implicitní (řečeno mimo samotnou formuli) a platí pro všechny kvantifikátory formule stejně. Může to být možná trochu v rozporu s tím, co jste se učili dříve, ale má to své dobré formální důvody.

Čtenář by měl v tomto místě vzít na vědomí, že naše definice vztahující se k predikátové logice jsou velmi velmi zjednodušené a nenahrazují kurz matematické logiky! Přesto poskytnou alespoň základní povědomí o této problematice (a hlavně o použití kvantifikace v tvrzeních) pro potřeby matematické formalizace v našem kurzu.

Jedním z nejdůležitějších poznatků k pochopení je, kdy lze určit pravdivostní hodnotu predikátové formule:

Fakt: Je-li *každá* proměnná – parametr predikátu – v dané formuli kvantifikovaná (tj. formule je *uzavřená*), pak je celá formule buď pravdivá nebo nepravdivá.

Příklad 1.15. Ukažme si vyjádření následujících slovních výroků uzavřenými formulami v predikátové logice:

- Každé prvočíslo větší než 2 je liché;

$$\forall n \in \mathbb{N} . [(Pr(n) \wedge n > 2) \Rightarrow Li(n)],$$

přičemž lze rozepsat $Li(n) \equiv \exists k \in \mathbb{N} . n = 2k + 1$. Predikát $Pr(n)$ vyjadřuje fakt, že n je prvočíslem.

- Každé celé číslo $n > 1$, které není prvočíslem, je dělitelné nějakým celým číslem y kde $n \neq y$ a $y > 1$;

$$\forall n \in \mathbb{Z} . (n > 1 \wedge \neg Pr(n)) \Rightarrow \exists y \in \mathbb{Z} (y | n \wedge n \neq y \wedge y > 1).$$

□

Poznámka: V tomto předmětu počítáme 0 za přirozené číslo.

Jak na kvantifikátory

Používání kvantifikátorů si neformálně přiblížíme pár příklady. Na jednu stranu je (asi) význam použití jednotlivého všeobecného či existenčního kvantifikátoru zřejmý z výše uvedeného popisu, na druhou stranu už tak jednoduché není kombinování více kvantifikátorů v jednom tvrzení. Podívejme se třeba na následující ukázkou:

Příklad 1.16. Proč na pořadí kvantifikátorů velmi záleží:

- Pro každého studenta A v posluchárně platí, že existuje student B v posluchárně takový, že A je kamarád B.
- Existuje student B v posluchárně, že pro každého studenta A v posluchárně platí, že A je kamarád B.

Vidíte ten propastný rozdíl ve významech uvedených dvou uzavřených predikátových formulí? Přitom jsme jen syntakticky zaměnili pořadí kvantifikace A a B. \square

Druhý, již složitější příklad, nám ukazuje, že kvantifikátory potkáme (skrytě) i v situacích, kde na první syntaktický pohled žádná kvantifikace není.

Příklad 1.17. Pepíček si přinesl ze školy domácí úkol z matematiky:

Pro dané $a = 3, b = 4, c = 5$ vypočtete hodnotu výrazu $(a + c - 5)(\bullet + 3) - b$.

Bohužel, jak vidíte, udělal do zadání kaňku \bullet a teď neví, zda pod kaňkou bylo písmeno a, b nebo c . „To je ale smůla“, řekl si Pepíček, „můj příklad teď nemohu jednoznačně vypočítat. Přitom by stačilo změnit v zadání jen jednu z hodnot a, b nebo c a ten příklad by už měl jednoznačné řešení i s tou mou nešťastnou kaňkou!“.

Vysvětlete a zdůvodněte, co tímto Pepíček myslel.

V příkladě jsou dvě roviny, ve kterých musíme argumentovat. Za prvé, že pro uvedené zadání domácího úkolu existují dvě (alespoň) přiřazení symbolu z a, b, c kaňce taková, že výsledné hodnoty budou různé. To je snadno vidět, třeba pro b na místě kaňky vypočítáme hodnotu 17, kdežto pro c na místě kaňky vypočítáme 20. Takže Pepíček opravdu nemohl zadaný úkol jednoznačně vyřešit.

Ve druhé rovině je našim úkolem najít a ověřit, že existuje změna hodnoty jednoho z a, b, c taková, že pro každé přiřazení symbolu z a, b, c kaňce v domácím úkolu vyjde výsledná hodnota výrazu stejně. Potom jednoznačné řešení existuje i s kaňkou a chytrý Pepíček jej najde. Takových vhodných změn je dokonce více, například „ $c = 5$ “ změníme na „ $c = 2$ “ a hodnota výrazu vyjde $0 \cdot (\bullet + 3) - 4 = -4$ bez ohledu na to, co se skrývá pod kaňkou.

Vidíte, jak důležité bylo v našem řešení pochopit a explicitně vyjádřit skryté kvantifikátory Pepíčkových úvah? Zbytek pak už šel docela snadno. \square

Mechanický postup negace výroků

Závěrem lekce se dostáváme k jednomu specifickému úskalí chápání přirozené logiky, totiž že přesný význam tvrzení se zanořenými negacemi je někdy skutečně obtížné pochopit. Co například následující?

„Není pravda, že nemohu neříct, že není pravda, že tě nemám nerad.“

Výrokové formule se proto pro lepší pochopení obvykle prezentují v tzv. normálním tvaru, ve kterém se negace vyskytují pouze u výrokových proměnných, formálně:

Definice: Formule $\varphi \in \Phi$ je v *normálním tvaru*, pokud se v ní operátor negace aplikuje pouze na výrokové proměnné.

Komentář: Například, pokud přijmeme pravidlo „dvojitá negace“ ($\models \neg\neg A \Leftrightarrow A$), tak výše napsanou větu si převedeme na lépe srozumitelný tvar:

„Nemusím říct, že tě mám nerad.“

Takovýto převod složitých tvrzení typu „není pravda, že ...“ na normální tvar je ale velmi obtížné dělat z hlavy, a proto je výhodné si ukázat čistě mechanický formální postup, který bez velké duševní námahy funguje vždy.

Tvrzení 1.18. Každou výrokovou formuli lze převést do normálního tvaru, pokud $k \Rightarrow$ povolíme i užívání odvozených spojek \wedge a \vee .

Komentář: Pro ilustraci $k \neg(A \Rightarrow B)$ je ekvivalentní $A \wedge \neg B$, $k \neg(C \wedge (\neg A \Rightarrow B))$ je ekvivalentní $\neg C \vee (\neg A \wedge \neg B)$ a $k \neg((A \Rightarrow B) \Rightarrow C)$ je ekvivalentní $(A \Rightarrow B) \wedge \neg C$.

Metoda 1.19. Převod formule φ do normálního tvaru $\mathcal{F}(\varphi)$.

Pro převod použijeme funktory \mathcal{F} a \mathcal{G} s neformálním významem $\mathcal{F}(X)$ jako „je pravda, že X “ a $\mathcal{G}(X)$ jako „není pravda, že X “. Tyto funktory definujeme induktivními předpisy následovně:

$$\begin{array}{ll} \mathcal{F}(A) & = A & \mathcal{G}(A) & = \neg A \\ \mathcal{F}(\neg\varphi) & = \mathcal{G}(\varphi) & \mathcal{G}(\neg\varphi) & = \mathcal{F}(\varphi) \\ \mathcal{F}(\varphi \Rightarrow \psi) & = \mathcal{F}(\varphi) \Rightarrow \mathcal{F}(\psi) & \mathcal{G}(\varphi \Rightarrow \psi) & = \mathcal{F}(\varphi) \wedge \mathcal{G}(\psi) \\ \mathcal{F}(\varphi \wedge \psi) & = \mathcal{F}(\varphi) \wedge \mathcal{F}(\psi) & \mathcal{G}(\varphi \wedge \psi) & = \mathcal{G}(\varphi) \vee \mathcal{G}(\psi) \\ \mathcal{F}(\varphi \vee \psi) & = \mathcal{F}(\varphi) \vee \mathcal{F}(\psi) & \mathcal{G}(\varphi \vee \psi) & = \mathcal{G}(\varphi) \wedge \mathcal{G}(\psi) \\ \mathcal{F}(\varphi \Leftrightarrow \psi) & = \mathcal{F}(\varphi) \Leftrightarrow \mathcal{F}(\psi) & \mathcal{G}(\varphi \Leftrightarrow \psi) & = (\mathcal{F}(\varphi) \wedge \mathcal{G}(\psi)) \vee (\mathcal{G}(\varphi) \wedge \mathcal{F}(\psi)) \end{array}$$

Pro predikátové formule toto rozšíříme ještě o pravidla:

$$\begin{array}{ll} \mathcal{F}(\forall x . \varphi) & = \forall x . \mathcal{F}(\varphi) & \mathcal{G}(\forall x . \varphi) & = \exists x . \mathcal{G}(\varphi) \\ \mathcal{F}(\exists x . \varphi) & = \exists x . \mathcal{F}(\varphi) & \mathcal{G}(\exists x . \varphi) & = \forall x . \mathcal{G}(\varphi) \end{array}$$

Komentář: Uvažme formuli $\neg(A \Rightarrow \neg(B \vee \neg(C \Rightarrow \neg A)))$. Užitím uvedeného postupu 1.19 získáme:

$$\begin{aligned} \mathcal{F}(\neg(A \Rightarrow \neg(B \vee \neg(C \Rightarrow \neg A)))) & = \mathcal{G}(A \Rightarrow \neg(B \vee \neg(C \Rightarrow \neg A))) = \\ \mathcal{F}(A) \wedge \mathcal{G}(\neg(B \vee \neg(C \Rightarrow \neg A))) & = A \wedge \mathcal{F}(B \vee \neg(C \Rightarrow \neg A)) = \\ A \wedge (\mathcal{F}(B) \vee \mathcal{F}(\neg(C \Rightarrow \neg A))) & = A \wedge (B \vee \mathcal{G}(C \Rightarrow \neg A)) = \\ A \wedge (B \vee (\mathcal{F}(C) \wedge \mathcal{G}(\neg A))) & = A \wedge (B \vee (C \wedge \mathcal{F}(A))) = \\ A \wedge (B \vee (C \wedge A)) & \end{aligned}$$

Formuli $A \wedge (B \vee (C \wedge A))$ lze dále zjednodušit na ekvivalentní formuli $A \wedge (B \vee C)$. To ale je již z našeho pohledu matematicky neformální (heuristický) postup.

Uvedené formální předpisy takto vyjadřují intuitivní postup převodu $\mathcal{F}(\varphi)$ a $\mathcal{G}(\varphi)$ do ekvivalentního „lidsky čitelného tvaru“.

Navazující studium

Látka této úvodní lekce má velmi široký záběr. S potřebou formálního zápisu tvrzení a důkazů se setkáte nejen v tomto, ale i ve všech dalších matematických předmětech, které zároveň studujete či budete studovat, a principy budou stále stejné. Proto se je snažte pochopit a zažít už na zde uvedených jednoduchých příkladech a vše si procvičte. A pokud

vám stále „jde hlava kolem“ z logického matematického vyjadřování, můžete se také zkusit podívat na hezkou úvodní (středoškolskou) knížku Antonína Sochora – *Logika pro všechny ochotné myslet*, Karolinum 2011.

Na druhou stranu je matematická logika v této lekci uvedena jen velmi omezeně na nejlépejší akceptovatelné intuitivní úrovni a její další rozvoj je ponechán samostatným kurzům. Na FI MU to bude navazující předmět IB101 Úvod do logiky. Zájemce o studium matematických hlubin logiky a třeba slavných Gödelových poznatků můžeme později odkázat na předmět MA007 Matematická logika.

2 Důkazové techniky, Indukce

Úvod

Náš hlubší úvod do matematických formalismů pro informatiku pokračujeme základním přehledem technik matematických důkazů. Pro vysvětlení, z Lekce 1 sice víme, že matematický důkaz má jednotnou formu Definice 1.1, ale nyní se budeme věnovat takřikajíc šablonám, podle nichž můžeme (snadněji) korektní důkaz sestavit.

Třebaže matematické dokazování a příslušné techniky mohou někomu připadat neprůhledné (a zpočátku možná zbytečné), jejich pochopení a zvládnutí není samoúčelné, neboť nám pomáhá si mnoho uvědomit o studovaných problémech samotných. Konečně, jak si můžeme být jisti svými poznatky, když bychom pro ně nebyli schopni poskytnout důkazy? Během studia tohoto předmětu poznáte (a ti méně šťastní až s překvapením u zkoušky), že podstata toho, k čemu naším výkladem směřujeme, se dá neformálně shrnout slovy „naučit se přesně vyjadřovat a být si svými tvrzeními naprosto jisti“ a analogicky později „naučit se navrhnout správné algoritmy a být si i svými programy naprosto jisti“.

Z důkazových postupů je pro informatiky asi nejdůležitější technika důkazů matematickou indukci, která je svou podstatou velmi blízká počítačovým programům (coby iterace cyklů). V dalším výkladu budeme indukci hojně využívat, především v Lkcích 10 a 11.

Cíle

Cílem této lekce je popsat základní techniky matematických důkazů, z nichž největší důraz klademe na matematickou indukci. Každý student by se měl alespoň naučit formálně psané důkazy číst a pochopit, nejlépe i sám dokázat jednoduché důkazy tvořit.

2.1 Přehled základních důkazových technik

V matematice je často používaných několik následujících způsobů – technik, jak k danému tvrzení sestavit korektní formální důkaz. (Uvědomme si, že jedno tvrzení mívá mnoho různých, stejně korektních důkazů; ty se však mohou výrazně lišit svou složitostí a čitelností. Uváděné techniky nám pomohou najít důkaz co nejjednodušší.) Tyto techniky si v bodech shrneme zde:

- *Přímé odvození*. To je způsob, o kterém jsme se dosud bavili.

Komentář: Postupujeme přímo od předpokladů k závěru, což vypadá nejpřirozeněji, ale sami poznáte, že taková „přímá“ cesta bývá obtížně k nalezení.

- *Kontrapozice* (také *obrácením* či *nepřímý důkaz*). Místo věty

„Jestliže platí *předpoklady*, pak platí *závěr*.“

budeme dokazovat ekvivalentní větu

„Jestliže *neplatí závěr*, pak *neplatí alespoň jeden z předpokladů*.“

Komentář: Na rozdíl od přímého odvození se obvykle lépe hledá cesta „pozpátku“, určité toto znáte například z hledání cest v dětských bludištích.

- *Důkaz sporem*. Místo věty

„Jestliže platí *předpoklady*, pak platí *závěr*.“

budeme dokazovat větu

„Jestliže platí *předpoklady* a *neplatí závěr*, pak *platí . . .*“

- nějaké zjevně nepravdivé tvrzení, nebo případně
- opak závěru či opak jednoho z předpokladů.

Komentář: Třebaže tato šablona vypadá trochu složitě, stará dobrá matematická moudrost praví: „Nevíte-li, jak nějakou větu dokázat, zkuste důkaz sporem. . .“

- *Matematická indukce.* Pokročilá technika, kterou popíšeme později. . .

Tyto techniky asi nejlépe ilustrujeme následujícími příklady důkazů.

Příklad důkazu kontrapozicí

Definice: *Prvočíslo* je celé číslo $p > 1$, které nemá jiné dělitele než 1 a p .

Příklad 2.1. *Na důkaz kontrapozicí (obrácením).*

Věta. *Jestliže p je prvočíslo větší než 2, pak p je liché.*

Důkaz obráceného tvrzení: Budeme místo uvedeného znění věty dokazovat, že je-li p sudé, pak p není větší než 2, nebo p není prvočíslo.

Připomínáme, že podle definice je p sudé, právě když lze psát $p = 2 \cdot k$, kde k je celé. Jsou jen dvě snadno řešitelné možnosti:

- $k \leq 1$. Pak $p = 2k$ není větší než 2.
- $k > 1$. Pak $p = 2 \cdot k$ není prvočíslo podle definice. □

Poznámka: Důkazy kontrapozicí pracují s *negací* (opakem) *předpokladů* a *závěru*. Je-li např. závěrem komplikované tvrzení tvaru

„z toho, že z A a B plyne C , vyplývá, že z A nebo C plyne A a B “,

není pouhou intuicí snadné zjistit, co je vlastně jeho negací. Proto se nebojte využívat snadného *mechanického postupu* Metody 1.19 ke správnému „znegování“ do normálního tvaru.

Příklady důkazu sporem

Příklad 2.2. *Jiný, kratší přístup k Důkazu 2.1.*

Věta. *Jestliže p je prvočíslo větší než 2, pak p je liché.*

Důkaz sporem: Nechť tedy p je prvočíslo větší než 2, které je sudé. Pak $p = 2 \cdot k$ pro nějaké $k > 1$, tedy p není prvočíslo, spor (s předpokladem, že p je prvočíslo). □

Důkaz sporem je natolik specifický a důležitý v matematice, že si zaslouží širší vysvětlení. Co je vlastně jeho podstatou? Je to (zcela přirozený) předpoklad, že v *konzistentní teorii* nelze zároveň odvodit tvrzení i jeho negaci. Jestliže tedy ve schématu

„Jestliže platí *předpoklady* a platí *opak závěru*, pak platí *opak jednoho z předpokladů*, nebo platí jiné *zjevně nepravdivé tvrzení*.“

odvodíme k některému předpokladu jeho opak, nebo případně jiné tvrzení, které odporuje všeobecně přijatým faktům (přesněji axiomům, například $0 = 1$), pak něco musí být „špatně“. Co však v našem tvrzení může (nezapomeňte předpoklad konzistence) být chybné? Původní předpoklady byly dány, takže zbývá jedině náš dodatečný předpoklad,

že platí *opak závěru*. Tudíž opak závěru nemůže nikdy platit a dvojnásobně negací odvodíme, že platí původní závěr.

Příklad 2.3. *Opět sporem.*

Věta. Číslo $\sqrt{2}$ není racionální.

Důkaz sporem: Nechť tedy $\sqrt{2}$ je racionální, tj. nechť existují nesoudělná celá kladná čísla r, s taková, že $\sqrt{2} = r/s$.

- Pak $2 = r^2/s^2$, tedy $r^2 = 2 \cdot s^2$, proto r^2 je dělitelné dvěma. Z toho plyne, že i r je dělitelné dvěma (proč? opět na to můžete jít sporem...).
- Jelikož r je dělitelné dvěma, je r^2 dělitelné dokonce čtyřmi, tedy $r^2 = 4 \cdot m$ pro nějaké m . Pak ale také $4 \cdot m = 2 \cdot s^2$, tedy $2 \cdot m = s^2$ a proto s^2 je dělitelné dvěma.
- Z toho plyne, že s je také dělitelné dvěma. Celkem dostáváme, že r i s jsou dělitelné dvěma, jsou tedy soudělná a to je spor (se základní vlastností racionálního čísla). \square

2.2 Věty typu „tehdy a jen tehdy“

Uvažujme nyní (v matematice poměrně hojně) věty tvaru

„Nechť platí předpoklady P . Pak tvrzení A platí *tehdy a jen tehdy*, platí-li tvrzení B .“

Příklady jiných jazykových formulací téže věty jsou:

- * Za předpokladů P je tvrzení B *nutnou a postačující* podmínkou pro platnost tvrzení A .
- * Za předpokladů P je tvrzení A *nutnou a postačující* podmínkou pro platnost tvrzení B .
- * Nechť platí předpoklady P . Pak tvrzení A platí *právě tehdy* když platí tvrzení B .

Fakt: *Plný důkaz* vět tohoto tvaru má vždy *dvě části(!)*. Je třeba dokázat:

- * Jestliže platí předpoklady P a tvrzení A , pak platí tvrzení B .
- * Jestliže platí předpoklady P a tvrzení B , pak platí tvrzení A .

Příklad 2.4. *Na důkaz typu „tehdy a jen tehdy“.*

Věta. Pro každá dvě celá čísla a, b platí, že $a < b$ právě tehdy, když $2^a < 2^b$.

Důkaz: Nezapomínáme, že našim úkolem je dokázat oba směry tvrzení (implikace zleva doprava a zprava doleva). Začneme s prvním: Předpoklad $a < b$ je definičně ekvivalentní tomu, že $b = a + k$ pro nějaké přirozené $k > 0$. Potom $2^b = 2^{a+k} = 2^a \cdot 2^k = \ell \cdot 2^a$ pro nějaké přirozené $\ell = 2^k \geq 2$, a tudíž $2^b - 2^a = (\ell - 1) \cdot 2^a \geq 1 \cdot 2^a > 0$. Tím je první část důkazu hotova.

V opačném směru postupujeme z předpokladu $2^a < 2^b$. Vydělením obou stran nerovnosti kladným číslem 2^a získáme $2^b/2^a = 2^{b-a} > 1$. Dále postupujeme sporem. Nechť tedy $a \geq b$, neboli $a - b = m \geq 0$. Potom máme $2^{a-b} = \underbrace{2 \cdot \dots \cdot 2}_{m \times} \geq 1$. Avšak celkově $1 = 2^0 = 2^{a-b} \cdot 2^{b-a} > 1 \cdot 1 = 1$ je sporné. Proto zbývá jen požadovaný závěr $a < b$. \square

Komentář: Je možno se někdy oddělenému zpracování obou směrů takového tvrzení vyhnout? Ano, stačí dělat pouze tzv. ekvivalentní úpravy, ale je to velmi nebezpečné. Co

například tvrzení „ $a = b$ právě když $ax = bx$ “? To přece dokážeme vynásobením obou stran rovnosti stejným číslem $x \dots$

Je to sice téměř pravda, ale(!) musíme si všimnout, že násobení obou stran rovnosti je ekvivalentní úpravou jen pro $x \neq 0$; jinak nazpět dělíme nulou (což na světě dokáže pouze Chuck Norris). Zapamatujte si, že nejlepší cestou, jak se takovým chybám a problémům vyhnout, je skutečně poctivě dokazovat každý směr ekvivalence zvlášť.

2.3 Matematická indukce

Pokud se souhrnně podíváme na důkazové techniky v matematice, všimneme si, že matematickou indukci lze považovat za „dvorní“ důkazovou technikou diskrétní matematiky. To proto, že umožňuje pohodlně dokazovat i složitá tvrzení po jednotlivých (malých a diskrétních) krocích od jednoduchých počátků.

Uvažme tedy větu ve tvaru:

„Pro každé přirozené (celé) $n \geq k_0$ platí $T(n)$.“

Zde k_0 je nějaké pevné přirozené číslo a $T(n)$ je tvrzení parametrizované číslem n . Příkladem je třeba tvrzení:

Pro každé $n \geq 0$ platí, že n prímek dělí rovinu nejvýše na $\frac{1}{2}n(n+1) + 1$ oblastí.

Definice 2.5. Princip matematické indukce říká, že k důkazu věty

„Pro každé přirozené (celé) $n \geq k_0$ platí $T(n)$.“

stačí ověřit platnost těchto dvou tvrzení:

- $T(k_0)$ (tzv. *báze* neboli základ indukce)
- Pro každé $n \geq k_0$; jestliže platí $T(n)$, pak platí také $T(n+1)$. (indukční předpoklad) (indukční krok)

Formálně řečeno, matematická indukce je axiomem aritmetiky přirozených čísel.

Opět jako v předešlém si tuto techniku ilustrujeme množstvím názorných příkladů.

Příklady důkazů indukcí

Příklad 2.6. Velmi jednoduchá a přímočará indukce.

Věta. Pro každé přirozené $n \geq 1$ je stejná pravděpodobnost, že při současném hodu n kostkami bude výsledný součet sudý, jako, že bude lichý.

Důkaz: *Základ indukce* je zde zřejmý: Na jedné kostce (poctivé!) jsou tři lichá a tři sudá čísla, takže obě skupiny padají se stejnou pravděpodobností.

Indukční krok pro $n \geq 1$: Nechtě p_n^s pravděpodobnost, že při hodu n kostkami bude výsledný součet sudý, a p_n^l je pravděpodobnost lichého. Podle indukčního předpokladu je $p_n^s = p_n^l = \frac{1}{2}$. Hoďme navíc $(n+1)$ -ní kostkou. Podle toho, zda na ní padne liché nebo sudé číslo, je pravděpodobnost celkového sudého součtu rovna

$$\frac{3}{6}p_n^l + \frac{3}{6}p_n^s = \frac{1}{2}$$

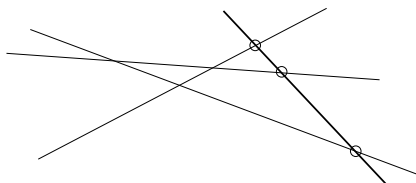
a stejně pro pravděpodobnost celkového lichého součtu. □

Příklad 2.7. Ukázka skutečné důkazové „síly“ principu matematické indukce.

Věta. Pro každé $n \geq 0$ platí, že n přímek dělí rovinu nejvýše na

$$\frac{1}{2}n(n+1) + 1$$

oblastí.



Důkaz: Pro bázi indukce stačí, že 0 přímek dělí rovinu na jednu část. (Všimněte si také, že 1 přímka dělí rovinu na dvě části, jen pro lepší pochopení důkazu.)

Mějme nyní rovinu rozdělenou n přímkami na nejvýše $\frac{1}{2}n(n+1) + 1$ částí. Další, $(n+1)$ -ná přímka je rozdělena průsečíky s předchozími přímkami na nejvýše $n+1$ úseků a každý z nich oddělí novou část roviny. Celkem tedy bude rovina rozdělena našimi přímkami na nejvýše tento počet oblastí:

$$\frac{1}{2}n(n+1) + 1 + (n+1) = \frac{1}{2}n(n+1) + \frac{1}{2} \cdot 2(n+1) + 1 = \frac{1}{2}(n+1)(n+2) + 1$$

□

Příklad 2.8. Další indukční důkaz rozepsaný v podrobných krocích.

Věta. Pro každé $n \geq 0$ platí $\sum_{j=0}^n j = \frac{n(n+1)}{2}$.

Důkaz indukcí vzhledem k n .

- **Báze:** Musíme dokázat tvrzení $T(0)$, což je v tomto případě rovnost $\sum_{j=0}^0 j = \frac{0(0+1)}{2}$. Tato rovnost (zjevně) platí.
- **Indukční krok:** Musíme dokázat následující tvrzení pro $n \geq 0$:

Jestliže platí $\sum_{j=0}^n j = \frac{n(n+1)}{2}$, pak platí $\sum_{j=0}^q j = \frac{q(q+1)}{2}$, kde $q = n+1$.

Předpokládejme tedy, že $\sum_{j=0}^n j = \frac{n(n+1)}{2}$ a pokusme se dokázat, že pak také

$$\sum_{j=0}^q j = \frac{q(q+1)}{2} = \frac{(n+1)(n+2)}{2}.$$

To už plyne úpravou:

$$\sum_{j=0}^q j = \sum_{j=0}^n j + (n+1) = \frac{n(n+1)}{2} + (n+1) = \frac{n(n+1) + 2(n+1)}{2} = \frac{(n+1)(n+2)}{2}$$

Podle principu matematické indukce je celý důkaz hotov. □

Poznámka: Výsledek Příkladu 2.8 je ukázkou tzv. sumačního vzorce pro posloupnost přirozených čísel. Jinou ukázkou je třeba vztah

$$1 + 3 + \dots + (2n-3) + (2n-1) = 1 + (2n-1) + 3 + (2n-3) + \dots = 2n \cdot \frac{n}{2} = n^2,$$

nebo $1^2 + 2^2 + \dots + n^2 = \frac{1}{6}n(n+1)(2n+1)$ a $1^3 + 2^3 + \dots + n^3 = (1+2+\dots+n)^2 = \frac{1}{4}n^2(n+1)^2$. Odvozování a práce s jednoduchými sumačními vzorci by také měla náležet do výbavy schopného informatika, princip matematické indukce je zde klíčový.

2.4 Komentáře k matematické indukci

Pro správné a úspěšné použití indukce v dokazování je vhodné si zapamatovat několik cenných rad:

- * Základní trik všech důkazů matematickou indukcí je vhodná *reformulace* tvrzení $T(n+1)$ tak, aby se „odvolávalo“ na tvrzení $T(n)$.
 - Dobře se vždy podívejte, v čem se liší tvrzení $T(n+1)$ od tvrzení $T(n)$. Tento „rozdíl“ budete muset v důkaze zdůvodnit.
- * Pozor, občas je potřeba „zesílit“ tvrzení $T(n)$, aby indukční krok správně „fungoval“.
 - Velkým problémem bohužel je, že není možno podat návod, jak vhodné „zesílení“ nalézt (ani kdy jej vůbec hledat). Jedná se vlastně o pokusy a „hádání z křišťálové koule“.
- * Často se chybuje v důkazu indukčního kroku, neboť ten bývá většinou výrazně obtížnější než báze, ale o to zrádnější jsou chyby v samotné zdánlivě snadné bázi!
 - Dejte si dobrý pozor, od které hodnoty $n \geq k_0$ je indukční krok univerzálně platný a jestli případně báze nezahrnuje více než jednu hodnotu...

Zesílení indukčního kroku

Příklad 2.9. *Když je nutno indukční krok zesílit...*

Věta. *Pro každé $n \geq 1$ platí*

$$s(n) = \frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{3 \cdot 4} + \cdots + \frac{1}{n(n+1)} < 1.$$

Důkaz: *Báze indukce* je zřejmá, neboť $\frac{1}{1 \cdot 2} < 1$.

Co však *indukční krok*? Předpoklad $s(n) < 1$ je sám o sobě „příliš slabý“ na to, aby bylo možno tvrdit $s(n+1) = s(n) + \frac{1}{(n+1)(n+2)} < 1$.

Neznamená to ještě, že by tvrzení nebylo platné, jen je potřeba náš indukční předpoklad *zesílit*. Budeme dokazovat

Tvrzení. *Pro každé přirozené $n \geq 1$ platí $s(n) \leq 1 - \frac{1}{n+1} < 1$.*

To platí pro $n = 1$ (nezapomeňte ověřit!) a dále už úpravou jen dokončíme zesílený indukční krok:

$$\begin{aligned} s(n+1) &= s(n) + \frac{1}{(n+1)(n+2)} \leq 1 - \frac{1}{n+1} + \frac{1}{(n+1)(n+2)} = \\ &= 1 + \frac{-(n+2) + 1}{(n+1)(n+2)} = 1 - \frac{1}{n+2} \quad \square \end{aligned}$$

Rozšíření báze a předpokladu

Mimo zesilování tvrzení indukčního kroku jsme někdy okolnostmi nuceni i k *rozšiřování* samotné báze indukce a s ní indukčního předpokladu na více než jednu hodnotu parametru n .

- Můžeme například předpokládat platnost (parametrizovaných) tvrzení $T(n)$ i $T(n+1)$ zároveň, a pak odvozovat platnost $T(n+2)$. Toto lze samozřejmě zobecnit na jakýkoliv počet předpokládaných parametrů.
- Můžeme dokonce předpokládat platnost tvrzení $T(j)$ pro všechna $j = k_0, k_0 + 1, \dots, n$ najednou a dokazovat $T(n+1)$. Toto typicky využijeme v případech, kdy indukční krok „rozdělí“ problém $T(n+1)$ na dvě menší části a z nich pak odvodí platnost $T(n+1)$.

Fakt: Obě prezentovaná „rozšíření“ jsou v konečném důsledku jen speciálními instancemi základní matematické indukce; použité rozšířené možnosti pouze zjednodušují formální zápis důkazu.

Příklad 2.10. *Když je nutno rozšířit bázi a indukční předpoklad. . .*

Věta. *Nechť funkce f pro každé $n \geq 0$ splňuje vztah*

$$f(n+2) = 2f(n+1) - f(n).$$

Pokud platí $f(0) = 1$ a zároveň $f(1) = 2$, tak platí $f(n) = n + 1$ pro všechna přirozená $n \geq 0$.

Důkaz: Už samotný pohled na daný vztah $f(n+2) = 2f(n+1) - f(n)$ naznačuje, že bychom měli rozšířit indukční předpoklad (a krok) zhruba takto:

Pro každé $n \geq 0$; jestliže platí $T(n)$; neboli $f(n) = n + 1$, a zároveň platí $T(n+1)$; $f(n+1) = n + 2$, pak platí také $T(n+2)$; $f(n+2) = n + 3$.

Báze indukce – pozor, zde už musíme ověřit dvě hodnoty

$$f(0) = 0 + 1 = 1, \quad f(1) = 1 + 1 = 2.$$

Náš *indukční krok* tak nyní může využít celého rozšířeného předpokladu, znalosti hodnot $f(n)$ i $f(n+1)$, pro ověření

$$f(n+2) = 2f(n+1) - f(n) = 2 \cdot (n+1+1) - (n+1) = n+3 = n+2+1. \quad \square$$

Komentář: Jak by tento důkaz měl být formulován v tradiční indukci? („Substitucí“ nového tvrzení.)

Závěrem malý „problém“

Příklad 2.11. *Aneb jak snadno lze v matematické indukci udělat chybu.*

Věta. („nevěta“)

V každém stádu o $n \geq 1$ koních mají všichni koně stejnou barvu.

Důkaz indukci vzhledem k n .

Báze: Ve stádu o jednom koni mají všichni koně stejnou barvu.

Indukční krok: Nechť $S = \{K_1, \dots, K_{n+1}\}$ je stádo o $n+1$ koních. Dokážeme, že všichni koně mají stejnou barvu. Uvažme dvě menší stáda:

- $S' = \{K_1, \underline{K_2}, \dots, K_n\}$
- $S'' = \{\underline{K_2}, \dots, K_n, K_{n+1}\}$

Podle indukčního předpokladu mají všichni koně ve stádu S' stejnou barvu B' . Podobně všichni koně ve stádu S'' mají podle indukčního předpokladu stejnou barvu B'' . Dokážeme, že $B' = B''$, tedy že všichni koně ve stádu S mají stejnou barvu. To ale plyne z toho, že koně K_2, \dots, K_n patří jak do stáda S' , tak i do stáda S'' . \square

Komentář: Ale to už je podvod! Vidíte, kde?

Navazující studium

Jak jsme již řekli, matematické důkazy a jejich chápání jsou samozřejmě nezbytné ke studiu vysokoškolských matematických předmětů. Bez schopnosti přesného vyjadřování a chápání definic a vět se ale informatik neobejde, ani pokud se zaměřuje čistě aplikovaným směrem; příkladem je třeba správné pochopení různých norem a specifikací.

Na druhou stranu umění “tvořit” nové matematické důkazy je dosti obtížné a nedá se jemu jen tak snadno naučit – vyžaduje to mnoho pokročilých matematických zkušeností. Jelikož je schopnost formálního matematického dokazování nezbytná (převážně jen) v teoretických informatických disciplínách, není tato část kritická v celém rozsahu našeho předmětu (a u zkoušek se objeví s menším důrazem), ale to neznamená, že byste se jí mohli zcela vyhýbat. Obzvláště techniku matematické indukce by měl každý informatik aspoň trochu ovládat, neboť s jejím použitím se zajisté ještě mnohokrát setkáte v budoucím studiu.

3 Množiny a množinové operace

Úvod

V přehledu matematických formalismů informatiky se v této a příští lekci zaměříme na základní datové typy matematiky, tj. na množiny, relace a funkce. Netradiční sousloví „datové typy“ matematiky zde volíme záměrně, abychom zdůraznili jejich fundamentálnost pro výstavbu navazující teorie v analogii k programování.

O množinách jste sice zřejmě slyšeli už na základní škole, ale podstatou našeho předmětu je uvést povětšinou neformálně známé pojmy na patřičnou formální úroveň nutnou pro teoretické základy informatiky. V případě konečné teorie množin si zde vystačíme s tzv. „naivním“ pohledem, který dostatečně matematicky přesně vystihuje všechny jejich konečné aspekty (o komplikacích nekonečných množin se krátce zmíníme až v Lekci 12).

Cíle

V této lekci si ukážeme první krok k seriózně vybudované matematické teorii množin – tzv. naivní teorii množin, která nám velmi dobře poslouží ve všech konečných případech. Na to navážeme zavedením základního množinového kalkulu a shrnutím běžných vlastností množin. Závěrem si uvedeme příklady rekurzivně definovaných posloupností.

3.1 Pojem množiny

Položme si hned na úvod tu nejdůležitější otázku: **Co je vlastně množina?**

Na tuto otázku bohužel není zcela jednoduchá odpověď... Abychom se vůbec někam v našem úvodu dostali, spokojíme se zatím jen s přirozeným „naivním pohledem“.

Definice naivní teorie množin: „Množina je soubor prvků a je svými prvky plně určena.“

Komentář: Příklady zápisu množin výčtem prvků

\emptyset , $\{a, b\}$, $\{b, a\}$, $\{a, b, a\}$, $\{\{a, b\}\}$, $\{\emptyset, \{\emptyset\}, \{\{\emptyset\}\}\}$, $\{x \mid x \text{ je liché přirozené číslo}\}$.

Co je ale pak prvek? Tady pozor, pojem prvku sám o sobě nemá matematický význam, svého významu totiž nabývá pouze ve spojení „být prvkem množiny“. Prvky množiny tak může být cokoli, mimo jiné i další množiny.

Komentář: Relativitu významu vztahu „prvek–množina“ si můžeme přiblížit třeba na vztahu „podřízený–nadřízený“ z běžného pracovního života. Tam také nemá smysl jen říkat, že je někdo podřízeným, aniž řekneme také jeho nadřízeného. Přitom i vedoucí je někomu ještě podřízený a naopak i ten poslední podřízený pracovník může být pánem třeba svého psa. Podobně je tomu s množinou jako „nadřízenou“ svých prvků.

Ale přece jenom... v dobře definovaném kontextu lze (omezeně) mluvit o prvcích jako samostatných entitách. Formálně se například jedná o prvky pevně dané nosné množiny.

Značení množin a jejich prvků:

- $x \in M$ „ x je prvkem množiny M “,
- \emptyset je prázdná množina $\{\}$.

Komentář: Některé vlastnosti vztahu „být prvkem“ jsou

* $a \in \{a, b\}$, $a \notin \{\{a, b\}\}$, $\{a, b\} \in \{\{a, b\}\}$, $a \notin \emptyset$, $\emptyset \in \{\emptyset\}$, $\emptyset \notin \emptyset$,

* rovnost množin dle svých prvků $\{a, b\} = \{b, a\} = \{a, b, a\}$, $\{a, b\} \neq \{\{a, b\}\}$.

Značení: Počet prvků (*mohutnost*) množiny A zapisujeme $|A|$.

Komentář: Například $|\emptyset| = 0$, $|\{\emptyset\}| = 1$, $|\{a, b, c\}| = 3$, $|\{\{a, b\}, c\}| = 2$.

Jednoduché srovnání množin

Vztah „být prvkem množiny“ nám přirozeně podává i způsob porovnávání množin mezi sebou. Jedná se o klíčovou část, čili hlavní nástroj teorie množin.

Definice: Množina A je *podmnožinou* množiny B , právě když každý prvek A je prvkem B . Píšeme $A \subseteq B$ nebo obráceně $B \supseteq A$. Říkáme také, že se jedná o *inkluzi*.

- * Platí $\{a\} \subseteq \{a\} \subseteq \{a, b\} \not\subseteq \{\{a, b\}\}$, $\emptyset \subseteq \{\emptyset\}$,
- * $A \subsetneq B$ právě když $A \subseteq B$ a $A \neq B$ (A je *vlastní* podmnožinou B).

Z naivní definice množiny pak přímo vyplývá následující:

Definice: Dvě množiny jsou si *rovné* $A = B$ právě když $A \subseteq B$ a $B \subseteq A$.

- * Podle naivní definice jsou totiž množiny A a B stejné, mají-li stejné prvky.
- * Důkaz rovnosti množin $A = B$ má obvykle *dvě části*: Odděleně se dokáží inkluze $A \subseteq B$ a $B \subseteq A$.

Ukázky nekonečných množin

Značení: Běžné číselné množiny v matematice jsou následující

- * $\mathbb{N} = \{0, 1, 2, 3, \dots\}$ je množina přirozených čísel,
- * $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, 3, \dots\}$ je množina celých čísel,
- * $\mathbb{Z}^+ = \{1, 2, 3, \dots\}$ je množina celých kladných čísel,
- * \mathbb{Q} je množina racionálních čísel (zlomků).
- * \mathbb{R} je množina reálných čísel.

Komentář: Tyto uvedené číselné množiny jsou vesměs *nekonečné*, na rozdíl od konečných množin uvažovaných v předchozím „naivním“ pohledu. Pojem nekonečné množiny se přímo v matematice objevil až teprve v 19. století a bylo s ním spojeno několik *paradoxů* ukazujících, že naivní pohled na teorii množin pro nekonečné množiny *nedostačuje*. My se k problematice nekonečných množin, Kantorově větě a Russelovu paradoxu vrátíme v závěru našeho předmětu v Lekci 12.

3.2 Množinové operace

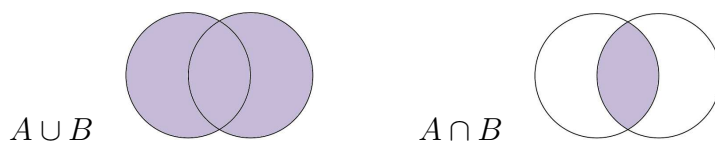
Začněme se základními operacemi množinového kalkulu, které zajisté na intuitivní úrovni již ovládáte ze školy. Pro formální úplnost podáme jejich přesné definice, na které se můžete odvolávat v důkazech.

Sjednocení a průnik

Definice 3.1. *Sjednocení* \cup a *průnik* \cap dvou množin A, B definujeme

$$\begin{aligned}A \cup B &= \{x \mid x \in A \text{ nebo } x \in B\}, \\A \cap B &= \{x \mid x \in A \text{ a současně } x \in B\}.\end{aligned}$$

Komentář: Pro neformální pochopení věci je velmi užitečné si množinové operace a vztahy zobrazovat tzv. Vennovými diagramy, které zobrazují jednotlivé množiny a v nich „oblasti“ prvků s daným vztahem k zobrazeným množinám. V našem případě to je takto:



- * Příklady $\{a, b, c\} \cup \{a, d\} = \{a, b, c, d\}$, $\{a, b, c\} \cap \{a, d\} = \{a\}$.
- * Vždy platí „distributivita“ $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ a $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
- * a také „asociativita“ $A \cap (B \cap C) = (A \cap B) \cap C$ (stejně pro \cup) a „komutativita“ $A \cap B = B \cap A$ (stejně pro \cup).

Komentář: Asociativita a komutativita operací sjednocení a průniku je na jednu stranu zcela přirozená, na druhou stranu však velmi důležitá například pro platnost následující definice.

Definice: Pro libovolný počet množin indexovaných pomocí I rozšířeně definujeme

$$\bigcup_{i \in I} A_i = \{x \mid x \in A_i \text{ pro nějaké } i \in I\},$$

$$\bigcap_{i \in I} A_i = \{x \mid x \in A_i \text{ pro každé } i \in I\}.$$

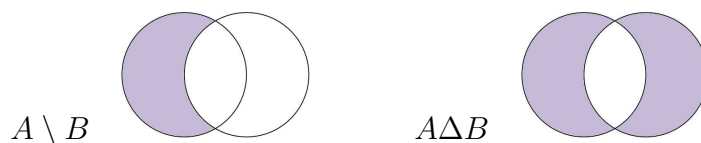
Komentář: Necht' $A_i = \{2 \cdot i\}$ pro každé $i \in \mathbb{N}$. Pak $\bigcup_{i \in \mathbb{N}} A_i$ je množina všech sudých přirozených čísel. Necht' $B_i = \{x \mid x \in \mathbb{N}, x \geq i\}$ pro každé $i \in \mathbb{N}$. Pak $\bigcap_{i \in \mathbb{N}} B_i = \emptyset$.

Množinový rozdíl

Definice 3.2. *Rozdíl* \ a *symetrický rozdíl* Δ dvou množin A, B definujeme

$$A \setminus B = \{x \mid x \in A \text{ a současně } x \notin B\},$$

$$A \Delta B = (A \setminus B) \cup (B \setminus A).$$



- * Příklady $\{a, b, c\} \setminus \{a, b, d\} = \{c\}$, $\{a, b, c\} \Delta \{a, b, d\} = \{c, d\}$.
- * Vždy platí například $A \setminus (B \cap C) = (A \setminus B) \cup (A \setminus C)$ apod.

Definice: Pro libovolný počet množin indexovaných pomocí *konečné* I rozšířeně definujeme

$$\bigtriangleup_{i \in I} A_i = \{x \mid x \in A_i \text{ pro lichý počet } i \in I\}.$$

Komentář: Povšimněte si, že operace rozdílu není asociativní ani komutativní (ostatně stejně je tomu u aritmetického rozdílu). Symetrický rozdíl už asociativní i komutativní je, není to však na první pohled vidět. Uměli byste toto dokázat?

Definice: Necht' $A \subseteq M$. *Doplňkem* A *vzhledem k* M je množina $\overline{A} = M \setminus A$.

Komentář: Jedná se o poněkud specifickou operaci, která musí být vztažena vzhledem k nosné množině M ! Je-li $M = \{a, b, c\}$, pak $\overline{\{a, b\}} = \{c\}$. Je-li $M = \{a, b\}$, pak $\overline{\{a, b\}} = \emptyset$.

* Vždy pro $A \subseteq M$ platí $\overline{\overline{A}} = A$ („dvojitý“ doplněk).

* Vždy pro $A, B \subseteq M$ platí $\overline{A \cup B} = \overline{A} \cap \overline{B}$ a $\overline{A \cap B} = \overline{A} \cup \overline{B}$. (Viz Věta 3.6.)

Uspořádané dvojice a kartézský součin

Zatímco prvky v množinách jsou zcela *neuspořádané*, jsou mnohé situace, kdy musíme pracovat se „seřazenými“ výčty prvků. V teorii množin lze takovéto seřazení definovat oklikou následovně:

Definice: *Uspořádaná dvojice* (a, b) je zadána množinou $\{\{a\}, \{a, b\}\}$.

Fakt: Platí $(a, b) = (c, d)$ právě když $a = c$ a současně $b = d$. Uměli byste toto sami dokázat přímo z podané definice?

* Co je podle definice (a, a) ? Je to $(a, a) = \{\{a\}, \{a, a\}\} = \{\{a\}, \{a\}\} = \{\{a\}\}$.

Definice 3.3. Kartézský součin dvou množin A, B definujeme jako množinu všech uspořádaných dvojic ze složek z A a B

$$A \times B = \{(a, b) \mid a \in A, b \in B\}.$$

* Příklady $\{a, b\} \times \{a\} = \{(a, a), (b, a)\}$, $\{c, d\} \times \{a, b\} = \{(c, a), (c, b), (d, a), (d, b)\}$.

* Platí $\emptyset \times X = \emptyset = X \times \emptyset$ pro každou množinu X .

* Jednoduchá mnemotechnická pomůcka říká $|A \times B| = |A| \cdot |B|$.

Definice: Pro každé $k \in \mathbb{N}$, $k > 0$ definujeme *uspořádanou k -tici* (a_1, \dots, a_k) induktivně

$$- (a_1) = a_1,$$

$$- (a_1, a_2, \dots, a_i, a_{i+1}) = ((a_1, a_2, \dots, a_i), a_{i+1}).$$

Všimněte si, že tato definice používá tzv. rekurentní vztah, blíže viz Oddíl 3.4.

Fakt: Platí $(a_1, \dots, a_k) = (b_1, \dots, b_k)$ právě když $a_i = b_i$ pro každé $i \in \mathbb{N}$ kde $1 \leq i \leq k$.

Definice kartézského součinu více množin: Pro každé $k \in \mathbb{N}$ definujeme

$$A_1 \times A_2 \times \dots \times A_k = \{(a_1, a_2, \dots, a_k) \mid a_i \in A_i \text{ pro každé } 1 \leq i \leq k\}.$$

Navíc pokud jsou množiny A_i indexovány přes libovolnou konečnou množinu $I \ni i$, tak jejich kartézský součin krátce píšeme jako $\prod_{i \in I} A_i$.

* Například $\mathbb{Z}^3 = \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} = \{(i, j, k) \mid i, j, k \in \mathbb{Z}\}$.

* Co je A^0 ? $\{\emptyset\}$, neboť jediná uspořádaná 0-tice je právě prázdná \emptyset .

Poznámka: Podle uvedené definice *není kartézský součin asociativní*, tj. obecně nemusí platit, že $A \times (B \times C) = (A \times B) \times C$. V matematické praxi je někdy výhodnější uvažovat upravenou definici, podle níž součin *asociativní* je. Pro účely této přednášky není podstatné, k jaké definici se přikloníme. Prezentované definice a věty „fungují“ pro obě varianty.

Potenční množina

Definice 3.4. *Potenční množina* množiny A , neboli množina všech podmnožin, je definovaná vztahem

$$2^A = \{B \mid B \subseteq A\}.$$

- * Platí například $2^{\{a,b\}} = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$,
- * $2^\emptyset = \{\emptyset\}$, $2^{\{\emptyset, \{\emptyset\}\}} = \{\emptyset, \{\emptyset\}, \{\{\emptyset\}\}, \{\emptyset, \{\emptyset\}\}\}$,
- * $2^{\{a\} \times \{a,b\}} = \{\emptyset, \{(a, a)\}, \{(a, b)\}, \{(a, a), (a, b)\}\}$.

Věta 3.5. *Počet prvků potenční množiny splňuje $|2^A| = 2^{|A|}$.*

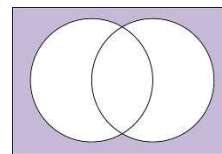
Důkaz: Stručně indukcí podle $|A|$: Pro $A = \emptyset$ platí $|2^A| = |\{\emptyset\}| = 1$. Pro každý další prvek $b \notin A$ rozdělíme všechny podmnožiny $A \cup \{b\}$ „na polovic“ na ty neobsahující b a na ty obsahující b , tudíž

$$|2^{A \cup \{b\}}| = 2 \cdot |2^A| = 2^{|A|+1} = 2^{|A \cup \{b\}|}. \quad \square$$

3.3 Porovnávání a určení množin

Pro obecnou ilustraci formálního množinového kalkulu si ukažme dva důkazy rovností mezi množinovými výrazy. Podobně lze (rozepsáním příslušných definic) rutinně dokazovat další množinové vztahy.

Věta 3.6. *Pro každé dvě množiny $A, B \subseteq M$ platí $\overline{A \cup B} = \overline{A} \cap \overline{B}$.*

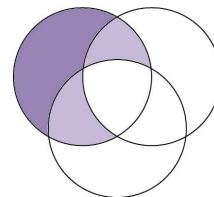


Důkaz v obou směrech rovnosti (viz ilustrační obrázek).

- $\overline{A \cup B} \subseteq \overline{A} \cap \overline{B}$: Pro $x \in M$ platí $x \in \overline{A \cup B}$, právě když $x \notin A \cup B$, neboli když zároveň $x \notin A$ a $x \notin B$. To znamená $x \in \overline{A}$ a zároveň $x \in \overline{B}$, z čehož vyplývá požadované $x \in \overline{A} \cap \overline{B}$.
- $\overline{A \cup B} \supseteq \overline{A} \cap \overline{B}$: Pro $x \in M$ platí $x \in \overline{A} \cap \overline{B}$, právě když $x \in \overline{A}$ a zároveň $x \in \overline{B}$, neboli když zároveň $x \notin A$ a $x \notin B$. To znamená $x \notin A \cup B$, z čehož vyplývá požadované $x \in \overline{A \cup B}$. □

Věta 3.7. *Pro každé tři množiny A, B, C platí*

$$A \setminus (B \cap C) = (A \setminus B) \cup (A \setminus C).$$



Důkaz (viz ilustrační obrázek).

- $A \setminus (B \cap C) \subseteq (A \setminus B) \cup (A \setminus C)$: Je-li $x \in A \setminus (B \cap C)$, pak $x \in A$ a zároveň $x \notin (B \cap C)$, neboli $x \notin B$ nebo $x \notin C$. Pro první možnost máme $x \in (A \setminus B)$, pro druhou $x \in (A \setminus C)$.

- Naopak $A \setminus (B \cap C) \supseteq (A \setminus B) \cup (A \setminus C)$: Je-li $x \in (A \setminus B) \cup (A \setminus C)$, pak $x \in (A \setminus B)$ nebo $x \in (A \setminus C)$. Pro první možnost máme $x \in A$ a zároveň $x \notin B$, z čehož plyne $x \in A$ a zároveň $x \notin (B \cap C)$, a tudíž $x \in A \setminus (B \cap C)$. Druhá možnost je analogická. \square

Charakteristický vektor (pod)množiny

V případech, kdy všechny uvažované množiny jsou podmnožinami nějaké konečné *nosné množiny* X , což není neobvyklé v programátorských aplikacích, s výhodou využijeme následující reprezentaci množin.

Definice: Mějme nosnou množinu $X = \{x_1, x_2, \dots, x_n\}$. Pro $A \subseteq X$ definujeme *charakteristický vektor* χ_A jako

$$\chi_A = (c_1, c_2, \dots, c_n), \text{ kde } c_i = 1 \text{ pro } x_i \in A \text{ a } c_i = 0 \text{ jinak.}$$

Užitečnost této reprezentace je ilustrována také následnými fakty.

- Platí $A = B$ právě když $\chi_A = \chi_B$.
- Množinové operace jsou realizovány „bitovými funkcemi“
sjednocení \sim OR, průnik \sim AND, symetrický rozdíl \sim XOR.

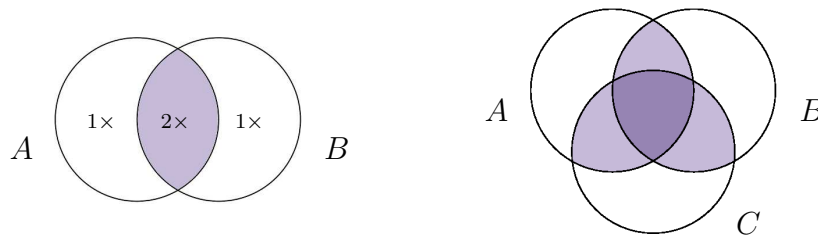
Princip inkluze a exkluze

Tento důležitý a zajímavý kombinatorický princip je někdy také nazýván „princip *zapojení a vypojení*“. Používá se ke zjišťování počtů prvků množin s neprázdnými průniky.

Věta 3.8. *Počet prvků ve sjednocení dvou či tří množin spočítáme:*

$$|A \cup B| = |A| + |B| - |A \cap B|$$

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$



Důkaz: Uvedeme si podrobný důkaz prvního vztahu $|A \cup B| = |A| + |B| - |A \cap B|$, přičemž druhá část je analogická (viz obrázková ilustrace). Nechť $x \in A \cup B$. Pokud $x \notin B$, tak je $x \in A$ a prvek x je započítán na pravé straně $|A| + |B| - |A \cap B|$ právě jednou v $|A|$. Obdobně je to v případě $x \notin A$. Nakonec pro $x \in A \cap B$ je ve vztahu $|A| + |B| - |A \cap B|$ tento prvek x započítán také $1 + 1 - 1 = 1$ krát. \square

Komentář: Všimněte si, že Větu 3.8 lze stejně tak využít k výpočtu počtu prvků v průniku množin. . .

Příklad 3.9. *Z 1000 televizí jich při první kontrole na výrobní lince má 5 vadnou obrazovku, 10 je poškrábaných a 12 má jinou vadu. Přitom 3 televize mají současně*

všechny tři vady a 4 jiné jsou poškrábané a mají jinou vadu. Kolik televizí je celkem vadných?

Řešení: Dosazením $|A| = 5$, $|B| = 10$, $|C| = 12$, $|A \cap B \cap C| = 3$, (zde pozor) $|A \cap B| = 3 + 0$, $|A \cap C| = 3 + 0$, $|B \cap C| = 3 + 4$ do Věty 3.8 zjistíme výsledek 17. \square

Poznámka. Jen stručně, bez důkazu a bližšího vysvětlení, si uvedeme *obecnou formu principu inkluze a exkluze*:

$$\left| \bigcup_{j=1}^n A_j \right| = \sum_{\emptyset \neq I \subseteq \{1, \dots, n\}} (-1)^{|I|-1} \cdot \left| \bigcap_{i \in I} A_i \right|$$

(Jeho znalost nebude v předmětu vyžadována.)

3.4 Posloupnosti a rekurentní vztahy

Uspořádané k -tice (Oddíl 3.2) jsou také nazývány *konečnými posloupnostmi* délky k , neboli seřazeními prvků z dané neprázdné množiny hodnot (opakování prvků je povoleno). Tento pohled lze přirozeně zobecnit na nekonečné posloupnosti, avšak musíme sáhnout k trochu složitějším pojmům:

Definice: *Nekonečná posloupnost* p je zobrazením z \mathbb{N} do svého oboru hodnot H , neboli $p: \mathbb{N} \rightarrow H$ (viz také Oddíl 4.1 o funkcích). Mimo „funkčního“ zápisu $p(n)$ často používáme „indexovou“ formu zápisu funkční hodnoty jako p_n .

Poznámka: Oborem hodnot posloupnosti obvykle bývá nějaká číselná množina, ale může to být i jakákoliv jiná množina. Také definiční obor posloupnosti (tj. indexy) může začínat od nuly nebo i od jedničky, jak je v aplikacích potřeba.

- Příklady posloupností:

- * $p_0 = 0, p_1 = 2, \dots, p_i = 2i, \dots$ je posloupnost sudých nezáporných čísel.

- * $3, 3.1, 3.14, 3.141, \dots$ je posloupnost postupných dekadických rozvoje π .

- * $1, -1, 1, -1, \dots$ je posloupnost určená vztahem $p_i = (-1)^i, i \geq 0$.

- * Pokud chceme stejnou posloupnost $1, -1, 1, -1, \dots$ zadat jako $q_i, i \geq 1$, tak ji určíme vzorcem $q_i = (-1)^{i-1}$.

- Posloupnost je *rostoucí* (či *klesající*), pokud $p_{n+1} > p_n$ ($p_{n+1} < p_n$) pro všechna n .

Rekurentní definice posloupnosti

Slovem *rekurentní* označujeme takové definice (či popisy), které se v jistých bodech odvolávají samy na sebe. (Už jste se setkali s „rekurzí“ při programování? A víte, co znamená?) Místo nepřehledných formálních definic si *rekurentní vztahy* uvedeme několika názornými ukázkami.

- Zadáme-li posloupnost p_n vztahy $p_0 = 1$ a $p_n = 2p_{n-1}$ pro $n > 0$, pak platí $p_n = 2^n$ pro všechna n .
- Obdobně můžeme zadat posloupnost q_n vztahy $q_1 = 1$ a $q_n = q_{n-1} + n$ pro $n > 1$. Potom platí $q_n = \frac{1}{2}n(n+1)$ pro všechna n . Uměli byste toto dokázat indukcí? Viz Příklad 2.8.

- Známá Fibonacciho posloupnost je zadána vztahy $f_1 = f_2 = 1$ a $f_n = f_{n-1} + f_{n-2}$ pro $n > 2$.

Příklad 3.10. Posloupnost f je zadána rekurentní definicí

$$f(0) = 3 \quad \text{a} \quad f(n+1) = 2 \cdot f(n) + 1$$

pro všechna přirozená n . Určete hodnotu $f(n)$ explicitním vzorcem v závislosti na n .

Řešení: V první fázi řešení takového příkladu musíme nějak „uhodnout“ hledaný vzorec pro $f(n)$. Jak? Zkusíme vypočítat několik prvních hodnot a uvidíme. . .

$$\begin{aligned} f(1) &= 2 \cdot f(0) + 1 = 2 \cdot 3 + 1 = 7 \\ f(2) &= 2 \cdot f(1) + 1 = 2 \cdot 7 + 1 = 15 \\ f(3) &= 2 \cdot f(2) + 1 = 2 \cdot 15 + 1 = 31 \\ f(4) &= 2 \cdot f(3) + 1 = 2 \cdot 31 + 1 = 63 \end{aligned}$$

Nepřipomínají nám tato čísla něco? Co třeba posloupnost $8 - 1, 16 - 1, 32 - 1, 64 - 1, \dots$? Bystrému čtenáři se již asi podařilo uhodnout, že půjde o mocniny dvou snižené o 1. Přesněji, $f(n) = 2^{n+2} - 1$ (proč je exponent $n + 2$? inu proto, aby správně vyšlo $f(0)$).

Ve druhé nesmíme ale zapomenout správnost našeho „věštění“ dokázat, nejlépe matematickou indukcí podle n .

- Báze: $f(0) = 2^{0+2} - 1 = 4 - 1 = 3$ platí.
- Indukční krok: za využití indukčního předpokladu

$$f(n+1) = 2 \cdot f(n) + 1 = 2 \cdot (2^{n+2} - 1) + 1 = 2 \cdot 2^{n+2} - 2 + 1 = 2^{(n+1)+2} - 1,$$

což také platí.

Podle principu matematické indukce je nyní dokázáno, že pro zadanou rekurentní posloupnost f platí $f(n) = 2^{n+2} - 1$ pro všechna přirozená n . \square

Navazující studium

I když jste se s „množinami“ setkávali už od základní školy, do formálně matematické teorie množin asi nahlédnete na VŠ poprvé. Nezálekněte se na úvod formality vyjadřování, která je bohužel nezbytná, a naučte se s množinami a relacemi dobře pracovat aspoň na zde ukázané naivní úrovni konečné teorie množin. Ostaně, jak již bylo psáno, množiny jsou základním stavebním kamenem celé matematiky. Mimo naší naivní teorie množin si lehký náhled na obecně nekonečné množiny a jejich zdánlivé paradoxy i informatické aplikace ukážeme později v Lekci 12. . .)

V dalším si porovnejte pojem posloupnosti s pojmem funkce v Oddílu 4.1 a také si později povšimněte blízké koncepční podobnosti rekurentních definic s induktivními definicemi množin a funkcí v Oddíle 6.3.

4 Relace a jejich použití

Úvod

V návaznosti na předchozí lekci si podrobně rozebereme matematické formalismy relací a funkcí coby jejich speciálního případu. Na rozdíl od množin, které se v jisté velmi naivní formě objevují už na základní škole, se relacím v jejich abstraktní podobě mnoho pozornosti v základní výuce nevěnuje. Stejně tak jste se již setkali na nižších stupních škol s pojmem funkce, ale povětšinou jen ve spojení s aritmetickými a analytickými funkcemi („vzorečky“) typu $x + 1$, $x^2 - y$, či $\sin x$, $1 + \cos x^2$, atd.

Jak uvidíte v této lekci, pojmy relace i funkce jsou zcela abstraktní a nemusí se k nim vázat žádný analytický vzorec výpočtu či podobné. Přitom na pojem (zcela obecné) relace velmi brzo narazí každý informatik při studiu dat a databází, které nad těmito pojmy staví. Není to však jen oblast relačních databází, ale i jiná místa informatiky, kde se obecné funkce a relace skrývají či přímo explicitně objevují. Nejčastěji se tak setkáte s binárními relacemi nad množinami, na něž se v textu hlouběji zaměříme.

Cíle

Úkolem této a příští lekce je vybudovat matematický aparát konečných relací, s primárním zaměřením na funkce a binární relace typu ekvivalence a uspořádání nad množinou. Ve vztahu k binárním relacím je zavedeno množství pojmů, které jsou později užitečné v různých oblastech matematiky i informatiky. V obecnosti jsou také vysvětleny inverze relací a jejich skládání. Látka pak plynule pokračuje Lekcí 5.

4.1 Relace a funkce nad množinami

Vedle množin jsou dalším důležitým základním „datovým typem“ matematiky relace, kterým vzhledem k jejich mnohotvárnému použití v informatice věnujeme významnou pozornost v této i příští lekci.

Definice 4.1. *Relace* mezi množinami A_1, A_2, \dots, A_k , pro $k \in \mathbb{N}$, je libovolná podmnožina kartézského součinu

$$R \subseteq A_1 \times A_2 \times \dots \times A_k.$$

Pokud $A_1 = A_2 = \dots = A_k = A$, hovoříme o k -ární relaci R na A . Speciálně tak mluvíme třeba o binární ($k = 2$), ternární ($k = 3$) nebo unární ($k = 1$) relaci.

Komentář: Příklady relací.

- * $\{(1, a), (2, a), (2, b)\}$ je relace mezi $\{1, 2, 3\}$ a $\{a, b\}$.
- * $\{(i, 2 \cdot i) \mid i \in \mathbb{N}\}$ je binární relace na \mathbb{N} .
- * $\{(i, j, i + j) \mid i, j \in \mathbb{N}\}$ je ternární relace na \mathbb{N} .
- * $\{3 \cdot i \mid i \in \mathbb{N}\}$ je unární relace na \mathbb{N} .
- * Jaký význam vlastně mají unární a nulární relace na A ?

Uvědomme si, jak obecně je relace definována – její definice umožňuje podchytit skutečně libovolné „vztahy“ mezi prvky téže i různých množin. V praxi se relace velmi široce využívají třeba v relačních databázích. . .

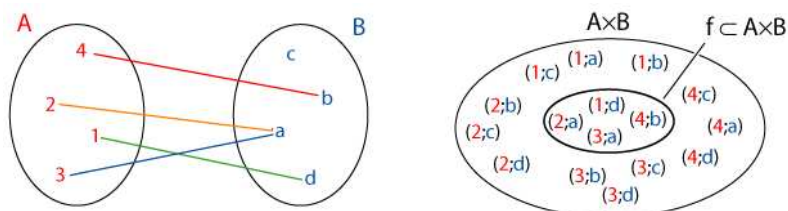
Funkce mezi množinami

První školní setkání s instancemi relací povětšinou proběhne u pojmu funkce. Ovšem tradiční školní pojetí *funkce* ji obvykle identifikuje s nějakým výpočetním (analytickým) předpisem či vzorcem. Pojem funkce je však daleko obecnější a zcela abstraktní, což si v tomto oddíle bohatě ilustrujeme a poslouží nám to i pro lepší pochopení obecnějšího pojmu relace.

Definice 4.2. (Totální) funkce z množiny A do množiny B

je relace f mezi A a B taková, že pro každé $x \in A$ existuje právě jedno $y \in B$ takové, že $(x, y) \in f$. Množina A se nazývá *definiční obor* a množina \overline{B} *obor hodnot* funkce f .

Komentář: Neformálně řečeno, ve funkci f je každé „vstupní“ hodnotě x přiřazena *jednoznačně* „výstupní“ hodnota y . (V obecné relaci počty „přiřazených“ dvojic neomezuje...)



Na druhou stranu si všimněte, že definiční obor i obor hodnot jsou součástí definice konkrétní funkce, neboli například v ukázce je oborem hodnot $B = \{a, b, c, d\}$, přestože samotné c není hodnotou pro žádný vstup.

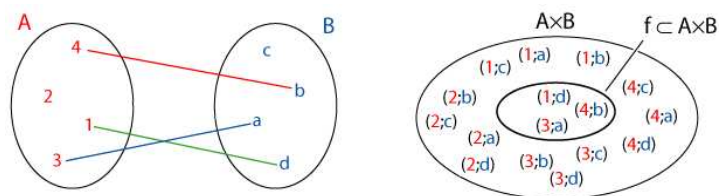
Značení: Místo $(x, y) \in f$ píšeme obvykle $f(x) = y$. Zápis $f : A \rightarrow B$ říká, že f je funkce s definičním oborem A a oborem hodnot B . Funkcím se také říká *zobrazení*.

Komentář: Příklady funkcí jsou třeba následující.

- * Definujeme funkci $f : \mathbb{N} \rightarrow \mathbb{N}$ předpisem $f(x) = x + 8$. Pak $f = \{(x, x + 8) \mid x \in \mathbb{N}\}$.
- * Definujeme funkci *plus* : $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ předpisem $plus(i, j) = i + j$.
Pak $plus = \{(i, j, i + j) \mid i, j \in \mathbb{N}\}$.

Definice: Pokud naši Definici 4.2 upravíme tak, že požadujeme pro každé $x \in A$ nejvýše jedno $y \in B$ takové, že $(x, y) \in f$, obdržíme definici *parciální funkce* z A do B .

Komentář:



V parciální funkci f nemusí být pro některé „vstupní“ hodnoty x funkční hodnota definována (viz například $f(2)$ v uvedeném obrázku).

Pro *nedefinovanou* hodnotu používáme znak \perp .

Komentář: Následuje několik příkladů parciálních funkcí.

* Definujeme parciální funkci $f : \mathbb{Z} \rightarrow \mathbb{N}$ předpisem

$$f(x) = \begin{cases} 3 + x & \text{jestliže } x \geq 0, \\ \perp & \text{jinak.} \end{cases}$$

Tj. $f = \{(x, 3 + x) \mid x \in \mathbb{N}\}$.

* Také funkce $f : \mathbb{R} \rightarrow \mathbb{R}$ daná běžným analytickým předpisem $f(x) = \log x$ je jen parciální – není definována pro $x \leq 0$.

* Co je relace, přiřazující lidem v ČR jejich (česká) rodná čísla?

4.2 Reprezentace konečných relací

Oblastí, kde informatici nejčastěji potkají obecné relace, je bezesporu ukládání dat. To proto, že shromažďovaná data, stejně jako relace, především sledují vztahy mezi objekty. Na druhou stranu je *relační databáze* zcela obecnou ukázkou reprezentace jakékoliv relace, kterou si ilustrujeme příkladem.

Příklad 4.3. *Tabulka relační databáze prezentuje obecnou relaci.*

Definujme následující množiny („elementární typy“)

* ZNAK = $\{a, \dots, z, A, \dots, Z, \text{mezera}\}$,

* CISLICE = $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Dále definujme tyto množiny („odvozené typy“)

* JMENO = ZNAK¹⁵, PRIJMENI = ZNAK²⁰, VEK = CISLICE³,

* ZAMESTNANEC „ \in “ JMENO \times PRIJMENI \times VEK.

Relaci „typu“ ZAMESTNANEC pak lze reprezentovat tabulkou:

JMENO	PRIJMENI	VEK
Jan	Novák	42
Petr	Vichr	28
Pavel	Zíma	26
Stanislav	Novotný	52

□

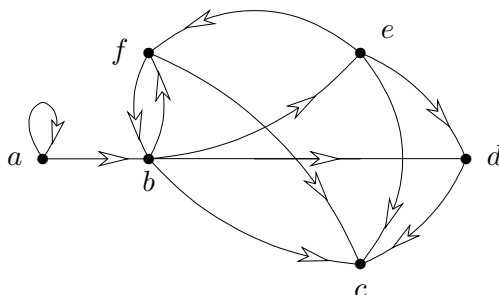
Reprezentace binárních relací na množině

Jistě čtenáři uznají, že zadání relace výčtem jejích složek není pro člověka (na rozdíl od počítače) tím nejpříjemnějším způsobem. Je tedy přirozené se ptát, jak co nejnázorněji takovou relaci, alespoň v její nejčastější binární podobě, ukázat.

Značení: Binární relaci $R \subseteq M \times M$ lze jednoznačně znázornit jejím *grafem*:

- Prvky M znázorníme jako body v rovině.
- Prvek $(a, b) \in R$ znázorníme jako *orientovanou hranu* („šipku“) z a do b . Je-li $a = b$, pak je touto hranou „smyčka“ na a .

Komentář: Například mějme $M = \{a, b, c, d, e, f\}$ a $R = \{(a, a), (a, b), (b, c), (b, d), (b, e), (b, f), (d, c), (e, c), (f, c), (e, d), (e, f), (f, b)\}$, pak:

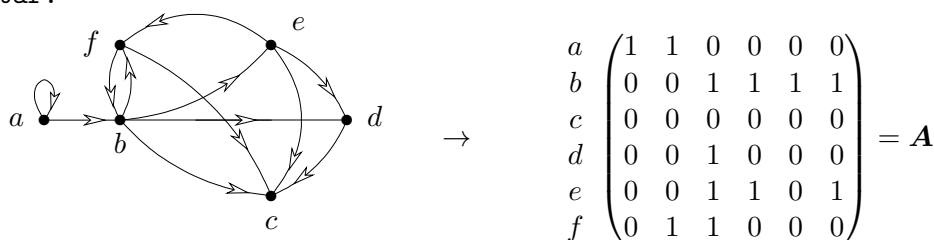


Pozor, nejedná se o „grafy funkcí“ známé třeba z matematické analýzy.

V případě, že M je nekonečná nebo „velká“, může být reprezentace R jejím grafem nepraktická (záleží také na míře „pravidelnosti“ R).

Značení: Binární relaci $R \subseteq M \times M$ lze jednoznačně zapsat také pomocí *matice* relace – matice A typu $M \times M$ s hodnotami z $\{0, 1\}$, kde $a_{i,j} = 1$ právě když $(i, j) \in R$.

Komentář:



4.3 Vlastnosti binárních relací

Pro začátek dalšího matematického výkladu relací si výčtem a doprovodnými schematickými obrázky uvedeme pět základních vlastností binárních relací na stejné množině, které nás typicky v matematické teorii zajímají. Je třeba si uvědomit, že se budeme zabývat velmi speciálním případem, neboť uvedené vlastnosti dávají dobrý smysl pouze pro uspořádané dvojice na téže množině, ale na druhou stranu se jedná o užitečný speciální případ.

Definice 4.4. Necht' $R \subseteq M \times M$. Binární relace R je

- *reflexivní*, právě když pro každé $a \in M$ platí $(a, a) \in R$;



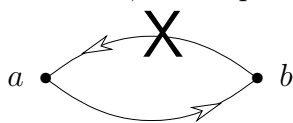
- *ireflexivní*, právě když pro každé $a \in M$ platí $(a, a) \notin R$;



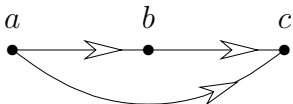
- *symetrická*, právě když pro každé $a, b \in M$ platí, že jestliže $(a, b) \in R$, pak také $(b, a) \in R$;



- *antisymetrická*, právě když pro každé $a, b \in M$ platí, že jestliže $(a, b), (b, a) \in R$, pak $a = b$;



- *tranzitivní*, právě když pro každé $a, b, c \in M$ platí, že jestliže $(a, b), (b, c) \in R$, pak také $(a, c) \in R$.



Následují dva základní typy binárních relací; kde R je

- relace *ekvivalence*, právě když je R reflexivní, symetrická a tranzitivní;
- *částečné uspořádání*, právě když je R reflexivní, antisymetrická a tranzitivní (často říkáme jen *uspořádání*).

Komentář: Pozor, může být relace *symetrická i antisymetrická zároveň*? Ano!

Vezměte si relaci $R = \{(x, x) \mid x \in M\}$, která obě jmenované vlastnosti splňuje. Proto pokud jste třeba dotázáni, zda relace je symetrická, nestačí se v odpovědi odvolávat na fakt, že je antisymetrická(!), neboť tyto vlastnosti se nevyklučují.



Příklad 4.5. Několik příkladů relací definovaných v přirozeném jazyce.

Nechť M je množina všech studentů 1. ročníku FI. Uvažme postupně relace $R \subseteq M \times M$ definované takto

- * $(x, y) \in R$ právě když x a y mají stejné rodné číslo;
- * $(x, y) \in R$ právě když x má stejnou výšku jako y (dejme tomu na celé mm);
- * $(x, y) \in R$ právě když výška x a y se neliší více jak o 2 mm;
- * $(x, y) \in R$ právě když x má alespoň takovou výšku jako y ;
- * $(x, y) \in R$ právě když x má jinou výšku než y (dejme tomu na celé mm);
- * $(x, y) \in R$ právě když x je zamilován(a) do y .

Zamyslete se podrobně, které z definovaných vlastností tyto jednotlivé relace mají. Které z nich tedy jsou ekvivalencí nebo uspořádáním? \square

Příklad 4.6. Jaké vlastnosti mají následující relace?

- * Bud' $R \subseteq \mathbb{N} \times \mathbb{N}$ definovaná takto $(x, y) \in R$ právě když x dělí y . (*Částečné uspořádání*, ale ne každá dvě čísla jsou porovnatelná.)
- * Bud' $R \subseteq \mathbb{N} \times \mathbb{N}$ definovaná takto $(x, y) \in R$ právě když x a y mají stejný zbytek po dělení číslem 5. (*Ekvivalence*.)
- * Nechť $F = \{f \mid f : \mathbb{N} \rightarrow \mathbb{N}\}$ je množina funkcí. Bud' $R \subseteq F \times F$ definovaná takto $(f, g) \in R$ právě když $f(x) < g(x)$ pro všechna x . (*Ireflexivní, antisymetrická a tranzitivní*, ale ne reflexivní – striktně řečeno není uspořádáním.) \square

Komentář: Co v případě, že naše relace některou z poptávaných vlastností nemá, ale nám by se ta vlastnost hodila? To řeší tzv. *uzávěry relací*:

- Reflexivní uzávěr jednoduše přidá do relace všechny dvojice (x, x) nad nosnou množinou.
- Symetrický uzávěr zahrne do relace všechny obrácené dvojice k existujícím dvojicím, neboli všechna chybějící (x, y) pokud $(y, x) \in R$.
- Tranzitivní uzávěr nelze popsat až tak jednoduše, ale stručně řečeno přidává do relace všechny ty dvojice (x, y) , pro které se lze (v grafu relace) dostat z x do y „po šipečkách“.

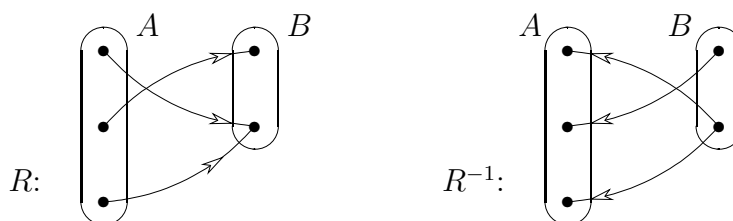
Formálně se této problematice uzávěrů relací bude věnovat Oddíl 6.4.

4.4 Inverzní relace a skládání relací

Ve výkladu se nyní vrátíme k obecnému pojetí relací mezi (třeba různými) množinami. K použitelné práci s relacemi v aplikacích se dostaneme, pokud budeme umět relace správně „převracet“ a „skládat“. To nám umožní následující definice.

Definice: Nechť $R \subseteq A \times B$ je binární relace mezi A a B . *Inverzní relace* k relaci R se značí R^{-1} a je definována takto:

$$R^{-1} = \{(b, a) \mid (a, b) \in R\}$$



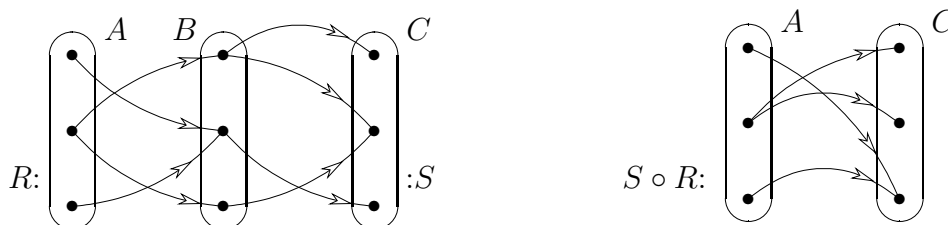
R^{-1} je tedy relace mezi B a A .

Následuje klíčová definice, pro jejíž bližší ilustraci odkazujeme také na Příklad 4.9.

Definice 4.7. Složení (kompozice) relací R a S .

Nechť $R \subseteq A \times B$ a $S \subseteq B \times C$ jsou binární relace. *Složení* relací R a S (v tomto pořadí!) je relace $S \circ R \subseteq A \times C$ definovaná takto:

$$S \circ R = \{(a, c) \mid \text{existuje } b \in B \text{ takové, že } (a, b) \in R, (b, c) \in S\}$$



Složení relací čteme „ R složeno s S “ nebo (pozor na pořadí!) „ S po R “.

Komentář: Několik matematických příkladů skládání relací následuje zde.

* Je-li

$$- A = \{a, b\}, \quad B = \{1, 2\}, \quad C = \{X, Y\},$$

$$- R = \{(a, 1), (b, 1), (b, 2)\}, \quad S = \{(1, X)\},$$

pak složením vznikne relace

$$- S \circ R = \{(a, X), (b, X)\}.$$

* Složením funkcí $h(x) = x^2$ a $f(x) = x + 1$ na \mathbb{R} vznikne funkce

$$(f \circ h)(x) = f(h(x)) = x^2 + 1.$$

* Složením těchto funkcí „naopak“ ale vznikne funkce $(h \circ f)(x) = h(f(x)) = (x + 1)^2$.

Poznámka: Nepříjemné je, že v některých oblastech matematiky (například v algebre při skládání zobrazení) se setkáme s právě opačným zápisem skládání, kdy se místo $S \circ R$ píše $R \cdot S$ nebo jen RS . Proto je si vždy dobré slovně ujasnit, které pořadí skládaných relací máme na mysli. My zde zásadně budeme používat pořadí $S \circ R$.

Tvrzení 4.8. *Nechť $R \subseteq A \times B$ a $S \subseteq B \times C$ jsou binární relace. Pak inverzí složené relace $S \circ R$ je relace $(S \circ R)^{-1} = R^{-1} \circ S^{-1}$.*

Důkaz: Zopakujme si, co nám říkají výše uvedené definice:

$$\begin{aligned} R^{-1} &= \{(b, a) \mid (a, b) \in R\} \\ S^{-1} &= \{(c, b) \mid (b, c) \in S\} \\ (S \circ R)^{-1} &= \{(c, a) \mid \text{existuje } b \in B \text{ takové, že } (a, b) \in R, (b, c) \in S\} \end{aligned}$$

Dosazením z prvních dvou vztahů do třetího vzniká

$$(S \circ R)^{-1} = \{(c, a) \mid \text{existuje } b \in B \text{ takové, že } (b, a) \in R^{-1}, (c, b) \in S^{-1}\}$$

a to již podle Definice 4.7 ihned implikuje požadovaný závěr

$$(S \circ R)^{-1} = R^{-1} \circ S^{-1}.$$

□

4.5 Skládání relací „v praxi“

Podívejme se nyní, jak se skládání relací přirozeně objevuje v práci s relačními databázemi. (Dá se zjednodušeně říci, že právě v operátoru skládání tabulkových relací tkví hlavní smysl relačních databází. . .)

Příklad 4.9. *Skládání v relační databázi studentů, jejich předmětů a fakult.*

Mějme dvě binární relace – jednu R přiřazující studentům MU kódy jejich zapsaných předmětů, druhou S přiřazující kódy předmětů jejich mateřským fakultám. Malý výsek z těchto relací může v tabulkové reprezentaci vypadat třeba následovně.

$R :$	student (učo)	předmět (kód)	$S :$	předmět (kód)	fakulta MU
	121334	MA010		MA010	FI
	133935	M4135		IB000	FI
	133935	IA102		IA102	FI
	155878	M1050		M1050	PřF
155878	IB000	M4135	PřF		

Jak z těchto „tabulkových“ relací zjistíme, kteří studenti mají zapsané předměty na kterých fakultách (třeba na FI)?

Jedná se jednoduše o složení relací $S \circ R$. V našem příkladě tabulkové reprezentace vyjde výsledek:

$$S \circ R :$$

student (učo)	fakulta MU
121334	FI
133935	FI
133935	PřF
155878	FI
155878	PřF

□

Zobecněné skládání relací

V praktických použitích relačních tabulek povětšinou nevystačíme jen s binárními relacemi, takže je přirozené se ptát, jestli lze podobně skládat i více-ární relace. Odpověď je snadná – lze to a ani nepotřebujeme novou definici, vystačíme s tou, kterou už máme výše uvedenou.

Definice: (skládání relací vyšší arity):

Mějme relace $T \subseteq K_1 \times K_2 \times \dots \times K_k$ a $U \subseteq L_1 \times L_2 \times \dots \times L_\ell$, přičemž pro nějaké $m < \min(k, \ell)$ platí $L_1 = K_{k-m+1}, L_2 = K_{k-m+2}, \dots, L_m = K_k$. Pak relaci T lze složit s relací U na zvolených m složkách L_1, \dots, L_m („překrytí“) s použitím Definice 4.7 takto:

- * Položme $A = K_1 \times \dots \times K_{k-m}$, $B = L_1 \times \dots \times L_m$ a $C = L_{m+1} \times \dots \times L_\ell$.
- * Příslušné relace pak jsou $R = \{(\vec{a}, \vec{b}) \in A \times B \mid (a_1, \dots, a_{k-m}, b_1, \dots, b_m) \in T\}$ a $S = \{(\vec{b}, \vec{c}) \in B \times C \mid (b_1, \dots, b_m, c_{m+1}, \dots, c_\ell) \in U\}$.
- * Nakonec přirozeně položme $U \circ_m T \simeq S \circ R$, takže vyjde

$$U \circ_m T = \{(\vec{a}, \vec{c}) \mid \text{ex. } \vec{b} \in B, \text{ že } (a_1, \dots, a_{k-m}, b_1, \dots, b_m) \in T \text{ a } (b_1, \dots, b_m, c_{m+1}, \dots, c_\ell) \in U\}.$$

Schematicky pro snadnější orientaci ve složkách našich relací:

$$\begin{aligned}
 T &\subseteq K_1 \times \dots \times K_{k-m} \times K_{k-m+1} \times \dots \times K_k \\
 U &\subseteq L_1 \times \dots \times L_m \times L_{m+1} \times \dots \times L_\ell \\
 U \circ_m T &\subseteq \underbrace{K_1 \times \dots \times K_{k-m}}_A \times \underbrace{L_1 \times \dots \times L_m}_B \times \underbrace{L_{m+1} \times \dots \times L_\ell}_C
 \end{aligned}$$

Opět je nejjednodušší si koncept skládání vícečetných relací ilustrovat příkladem.

Příklad 4.10. Skládání v relační databázi pasažérů a letů u leteckých společností.

Podívejme se na příklad hypotetické rezervace letů pro cestující, relace T . Jak známo (tzv. codeshare), letecké společnosti si mezi sebou „dělí“ místa v letadlech, takže různé lety (podle kódů) jsou ve skutečnosti realizovány stejným letadlem jedné ze společností. To zase ukazuje relace U .

pasažér	datum	let
Petr	5.11.	OK535
Pavel	6.11.	OK535
Jan	5.11.	AF2378
Josef	5.11.	DL5457
Alena	6.11.	AF2378

datum	let	letadlo
5.11.	OK535	ČSA
5.11.	AF2378	ČSA
5.11.	DL5457	ČSA
6.11.	OK535	AirFrance
6.11.	AF2378	AirFrance

Ptáme-li se nyní, setkají se Petr a Josef na palubě stejného letadla? Případně, čím letadlo to bude? Odpovědi nám dá složení relací $U \circ_2 T$, jak je popsáno výše.

$$U \circ_2 T :$$

pasážér	letadlo
Petr	ČSA
Josef	ČSA
Pavel	AirFrance
...	...

Zkuste se zamyslet, lze tyto dvě relace skládat ještě jinak? Co by pak bylo významem? □

Navazující studium

Mějte na paměti, že na pojmech množin, relací a funkcí jsou vystavěny prakticky všechny skutečné datové struktury používané v dnešní informatice. Explicitně toto můžete vidět na relačních databázích, ale i na mnoha jiných implicitních výskytech. Mnohem více si nejprve o relacích a poté speciálně o funkcích vysvětlíme v dalších dvou lekcích. Dalším důležitým matematickým datovým typem odvozeným z binárních relací jsou pak grafy probírané od Lekce 7, na nichž v různé míře staví většina základních algoritmů, které se budete učit.

5 Ekvivalence, Uspořádané množiny

Úvod

V této lekci pokračujeme rozebíráním relací na množinách jako nástrojů vyjadřujících vztahy mezi objekty. Zaměříme se přitom pouze na relace binární, které nějakým způsobem srovnávají objekty podle jejich vlastností (obvykle ve smyslu „stejný jako“ nebo „lepší/větší než“). Takto vágně opsané binární relace mívají jasné společné znaky, které se pak objevují ve zde uvedených formálních definicích relace ekvivalence a uspořádání.

Co se týče ekvivalencí, lze si je neformálně představit jako rozdělení prvků nosné množiny na „hromádky“, přičemž prvky každé jednotlivé hromádky jsou si navzájem v jistém smyslu stejné. Naopak uspořádání nám již podle svého názvu udává srovnání prvků nosné množiny, neboli které prvky si „stojí lépe“ než jiné.

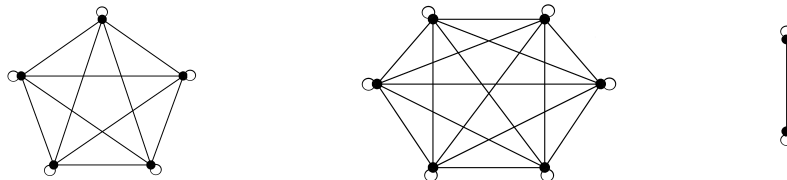
Cíle

Definujeme relace ekvivalence a uspořádání a mnohé další pojmy k nim se vztahující, například rozklady a Hasseovy diagramy. Studující by měli porozumět matematické problematice rozkladů množin a uspořádaných množin a naučit se je správně používat.

5.1 Relace ekvivalence

Nyní se hlouběji podívejme na první specifický typ binární relace zmíněný v Lekci 4: Podle Definice 4.4 je relace $R \subseteq M \times M$ ekvivalence právě když R je reflexivní, symetrická a tranzitivní. Tyto tři vlastnosti tedy musí být splněny a ověřeny k důkazu toho, že daná relace R je ekvivalencí.

Komentář: Jak vypadá graf relace ekvivalence? Poměrně příznačně, jak nám ukazuje následující obrázek (všimněte si absence šipek, která je dána symetrií relace).



Neformálně řečeno; ekvivalence je relace $R \subseteq M \times M$, taková, že $(x, y) \in R$ právě když x a y jsou v nějakém smyslu „stejné“ (leží na stejné hromádce).

Značení. V případě relace ekvivalence se poměrně často lze setkat s označením jako \sim či \approx místo R . Místo $(x, y) \in R$ se pak píše $x \approx y$.

Příklad 5.1. Necht' M je množina všech studentů 1. ročníku FI. Uvažme postupně relace $R \subseteq M \times M$ definované následovně a zkoumejme, zda se jedná o ekvivalence:

- * $(x, y) \in R$ právě když x má stejnou výšku jako y ;
- * $(x, y) \in R$ právě když x má stejnou barvu vlasů jako y ;
- * $(x, y) \in R$ právě když x, y mají stejnou výšku a stejnou barvu vlasů;
- * $(x, y) \in R$ právě když x, y mají stejnou výšku nebo stejnou barvu vlasů.

U kterého body se nejedná o relaci ekvivalence a proč? Je to poslední případ, kdy není splněna tranzitivita. \square

Uvedený příklad ukazuje na následující univerzální poznatek, který může platit (a taky platí) pouze pro průnik a nikoliv pro sjednocení.

Tvrzení 5.2. *Nechť R, S jsou dvě relace ekvivalence na stejné množině M . Pak jejich průnik $R \cap S$ je opět relací ekvivalence.*

Důkaz (náznak): Jelikož reflexivita a symetrie jsou zřejmé, stačí snadno ověřit platnost tranzitivity na $R \cap S$. Nechť $(a, b), (b, c) \in R \cap S$, pak podle tranzitivity každé jedné z R, S plyne $(a, c) \in R$ a zároveň $(a, c) \in S$. \square

Příklad 5.3. *Nechť $R \subseteq \mathbb{N} \times \mathbb{N}$ je binární relace definovaná takto: $(x, y) \in R$ právě když $|x - y|$ je dělitelné třemi. V jakém smyslu jsou zde x a y „stejné“?*

Platí $(x, y) \in R$ právě když x, y dávají stejný zbytek po dělení třemi. Je to opět relace ekvivalence. \square

Příklad 5.4. *Bud' R binární relace mezi všemi studenty na přednášce FI:IB000 definovaná takto: $(x, y) \in R$ právě když x i y sedí v první lavici.*

Už na první pohled jde o relaci symetrickou a tranzitivní. Proč se v tomto případě *nejedná* o relaci ekvivalence? Protože není reflexivní pro studenty sedící v dalších lavicích. (Takže si dávejte dobrý pozor na správné pochopení definic.) \square

5.2 Rozklady a jejich vztah k ekvivalencím

Náplní následující části výkladu je ukázat jiný přirozený pohled na ekvivalence. Tento nový pohled nám matematicky formalizuje již nastíněnou představu ekvivalence jako rozdělení prvků nosné množiny M na „hromádky“, přičemž prvky každé jednotlivé hromádky jsou si navzájem v jistém smyslu „stejné“.

Definice 5.5. Rozklad množiny. Nechť M je množina.

Rozklad (na) M je množina podmnožin $\mathcal{N} \subseteq 2^M$ splňující následující tři podmínky:

- $\emptyset \notin \mathcal{N}$ (tj. každý prvek \mathcal{N} je neprázdná podmnožina M);
- pokud $A, B \in \mathcal{N}$, pak buď $A = B$ nebo $A \cap B = \emptyset$;
- $\bigcup_{A \in \mathcal{N}} A = M$.

Prvkům \mathcal{N} se také říká *třídy rozkladu*.

Komentář:

* Bud' $M = \{a, b, c, d\}$. Pak $\mathcal{N} = \{\{a\}, \{b, c\}, \{d\}\}$ je rozklad na M .

* Nechť $A_0 = \{k \in \mathbb{N} \mid k \bmod 3 = 0\}$, $A_1 = \{k \in \mathbb{N} \mid k \bmod 3 = 1\}$, $A_2 = \{k \in \mathbb{N} \mid k \bmod 3 = 2\}$. Pak $\mathcal{N} = \{A_0, A_1, A_2\}$ je rozklad všech přirozených čísel \mathbb{N} podle zbytkových tříd.

Každý rozklad \mathcal{N} na M jednoznačně určuje jistou ekvivalenci $R_{\mathcal{N}}$ na M :

Věta 5.6. *Nechť M je množina a \mathcal{N} rozklad na M . Nechť $R_{\mathcal{N}} \subseteq M \times M$ je relace na M definovaná takto*

$(x, y) \in R_{\mathcal{N}}$ právě když existuje $A \in \mathcal{N}$ taková, že $x, y \in A$.

Pak $R_{\mathcal{N}}$ je ekvivalence na M .

Důkaz: Dokážeme, že $R_{\mathcal{N}}$ je reflexivní, symetrická a tranzitivní (Definice 4.4).

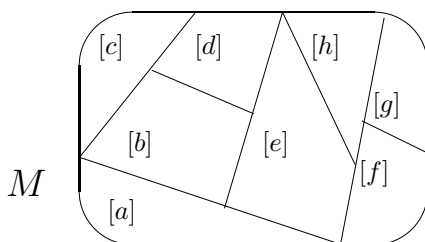
- Reflexivita: Buď $x \in M$ libovolné. Jelikož \mathcal{N} je rozklad na M , musí existovat $A \in \mathcal{N}$ takové, že $x \in A$ (jinak spor se třetí podmínkou z Definice 5.5). Proto $(x, x) \in R_{\mathcal{N}}$, tedy $R_{\mathcal{N}}$ je reflexivní.
- Symetrie: Necht' $(x, y) \in R_{\mathcal{N}}$. Podle definice $R_{\mathcal{N}}$ pak existuje $A \in \mathcal{N}$ taková, že $x, y \in A$. To ale znamená, že také $(y, x) \in R_{\mathcal{N}}$ podle definice $R_{\mathcal{N}}$, tedy $R_{\mathcal{N}}$ je symetrická.
- Tranzitivita: Necht' $(x, y), (y, z) \in R_{\mathcal{N}}$. Podle definice $R_{\mathcal{N}}$ existují $A, B \in \mathcal{N}$ takové, že $x, y \in A$ a $y, z \in B$. Jelikož $A \cap B \neq \emptyset$, podle druhé podmínky z Definice 5.5 platí $A = B$. Tedy $x, z \in A = B$, proto $(x, z) \in R_{\mathcal{N}}$ podle definice $R_{\mathcal{N}}$. \square

Každá ekvivalence R na M naopak jednoznačně určuje jistý rozklad M/R na M :

Věta 5.7. Necht' M je množina a R ekvivalence na M . Pro každé $x \in M$ definujeme množinu

$$[x] = \{y \in M \mid (x, y) \in R\}.$$

Pak $\{[x] \mid x \in M\}$ je rozklad na M , který značíme M/R a čteme „rozklad M podle R “.



Důkaz: Dokážeme, že M/R splňuje podmínky Definice 5.5.

- Pro každé $[x] \in M/R$ platí $[x] \neq \emptyset$, neboť $x \in [x]$.
- Necht' $[x], [y] \in M/R$. Ukážeme, že pokud $[x] \cap [y] \neq \emptyset$, pak $[x] = [y]$.

Jestliže $[x] \cap [y] \neq \emptyset$, existuje $z \in M$ takové, že $z \in [x]$ a $z \in [y]$. Podle definice $[x]$ a $[y]$ to znamená, že $(x, z), (y, z) \in R$. Jelikož R je symetrická a $(y, z) \in R$, platí $(z, y) \in R$. Jelikož $(x, z), (z, y) \in R$ a R je tranzitivní, platí $(x, y) \in R$. Proto také $(y, x) \in R$ (opět ze symetrie R). Nyní dokážeme, že $[y] = [x]$:

- * „ $[x] \subseteq [y]$ “: Necht' $v \in [x]$. Pak $(x, v) \in R$ podle definice $[x]$. Dále $(y, x) \in R$ (viz výše), tedy $(y, v) \in R$ neboť R je tranzitivní. To podle definice $[y]$ znamená, že $v \in [y]$.
- * „ $[y] \subseteq [x]$ “: Necht' $v \in [y]$. Pak $(y, v) \in R$ podle definice $[y]$. Dále $(x, y) \in R$ (viz výše), tedy $(x, v) \in R$ neboť R je tranzitivní. To podle definice $[x]$ znamená, že $v \in [x]$.

- Platí $\bigcup_{[x] \in M/R} [x] = M$, neboť $x \in [x]$ pro každé $x \in M$. \square

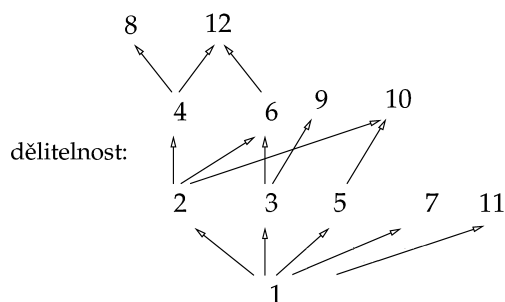
5.3 Uspořádání a uspořádané množiny

Přirozený vágní pojem porovnání/uspořádání objektů má také svou přesnou matematickou definici: Podle Definice 4.4 je relace $R \subseteq M \times M$ (částečné) uspořádání právě když

R je reflexivní, antisymetrická a tranzitivní. Tyto tři vlastnosti tedy musí být splněny a ověřeny k důkazu toho, že daná relace R je uspořádáním.

Komentář: Neformálně řečeno: uspořádání je taková relace $R \subseteq M \times M$, kde $(x, y) \in R$ právě když x je v nějakém smyslu „menší nebo rovno“ než y . Mohou ovšem existovat taková $x, y \in M$, kde neplatí $(x, y) \in R$ ani $(y, x) \in R$. (Pak říkáme, že x a y jsou *nesrovnatelné*.)

Jak *názorně* zobrazit (částečné) uspořádání? Příklad zjednodušeného zakreslení (jsou vynechány šipky vyplývající z reflexivity a tranzitivity, viz Oddíl 5.4) je zde:



Všimněte si, že je někdy zvykem „větší“ prvky kreslit nad ty „menší“.

Značení. Pro větší přehlednost se relace uspořádání často značí symboly jako \sqsubseteq či \preceq místo prostého R . I my tak často budeme psát třeba $x \preceq y$ místo $(x, y) \in R$.

Poznámka: Zajisté jste se již neformálně setkali s „neostrým“ uspořádáním čísel \leq a „ostrým“ uspořádáním $<$. Všimněte si dobře, že námi definované uspořádání je vždy neostré. Pokud byste naopak chtěli definovat *ostré* uspořádání, mělo by vlastnosti *ireflexivní*, antisymetrické a tranzitivní. (Příliš se však tato varianta nepoužívá.) Nadále budeme pracovat pouze s neostrým uspořádáním, ale smyčky vyplývající z reflexivity a případně i šipky z tranzitivity budeme pro větší přehlednost v obrázcích vynechávat.

Uspořádaná množina

Matematicky důležitějším pojmem než samotná relace uspořádání je uspořádaná množina, neboli soubor prvků s pevně zvoleným uspořádáním na nich.

Definice 5.8. *Uspořádaná množina* je dvojice (M, \preceq) , kde M je množina a \preceq je (částečné) uspořádání na M .

Definice: Uspořádání \preceq na M je *lineární* (nebo také *úplné*), pokud každé dva prvky M jsou v \preceq srovnatelné.

Příklady

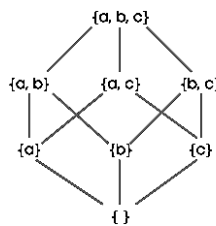
Příklad 5.9. *Nechť M je množina všech studentů 1. ročníku FI. Uvažme postupně relace uspořádání $R \subseteq M \times M$ definované následovně (jedná se vždy o uspořádání?):*

- * $(x, y) \in R$ právě když x má alespoň takovou výšku jako y ;
- * $(x, y) \in R$ právě když y má alespoň takovou výšku jako x ;
- * $(x, y) \in R$ právě když x a y mají stejné rodné číslo.

Ano, i v posledním bodě se jedná o uspořádání. Dobře si promyslete proč (které dvojice jsou vůbec porovnatelné?). \square

Příklad 5.10. Další ukázky uspořádaných množin následují zde:

- * (\mathbb{N}, \leq) je lineárně uspořádaná množina, kde \leq má „obvyklý“ význam.
- * $(\mathbb{N}, |)$, kde $a|b$ je relace dělitelnosti „ a dělí b “ na přirozených číslech, je uspořádaná množina. Toto uspořádání není lineární.
- * Bud' M množina. Pak $(2^M, \subseteq)$ je uspořádaná množina (říkáme *inkluzí*). Které dvojice jsou v uspořádání inkluzí *nesrovnatelné*?



\square

Komentář: Zamyslete se také, jak vypadá relace dělitelnosti na celých (tj. i záporných) číslech. Proč se už nejedná o uspořádání? Blíže viz konec Oddílu 5.4.

Multikriteriální uspořádání

Přirozenou ukázkou praktického problému, ve kterém potkáme problematiku netriviálně uspořádaných objektů, je výběr toho *nejlepšího* podle více nezávislých kritérií. Co třeba znamená požadavek na „nejlepší počítač“? I pokud se soustředíme jen na užitečné / výkonové charakteristiky počítače, můžeme porovnávat nejen podle rychlosti procesoru, ale také třeba podle výkonu grafiky, velikosti paměti a disku, nebo i počtu nakouslých jablek na víku. A jen málokdy se setkáme s dostupným počítačem, který by jasně dominoval ve všech kritériích najednou.

Takže jak lze objekty podle více kritérií seřadit?

- Pokud jsou všechna *kritéria numerická*, zkoumané objekty lze řadit podle jejich váženého průměru (či součtu). Avšak numerické hodnoty jsou ne vždy dostupné a především „průměrování“ kritérií často dává velmi nepřirozené výsledky.
- Objekty můžeme uspořádat podle průniku seřazení jednotlivých kritérií (říká se *po složkách*). Výhodou je, že pokud objekt (počítač) P je lepší než Q , tak P v žádném kritériu nemůže zaostávat za Q . Avšak výsledkem bude typicky existence několika „nejlepších“ objektů, které nebudeme umět vzájemně porovnat. (Podívejte se na pojem maximálního prvku v Definicí 5.13.)
- Nakonec si můžeme pomoci *slovníkovým uspořádáním*: Kritéria seřadíme od nejdůležitějšího a první odlišná hodnota kritéria rozhoduje. Například nás v první řadě zajímá rychlost procesoru, poté velikost paměti a až nakonec výkon grafiky či velikost disku... (Máme přece počítač na práci a ne na hraní her či sledování filmů.)

Předchozí vágní popisy uspořádání nyní formalizujeme matematicky. Formálně mějme několik kritérií označených indexovou množinou I (například čtyři kritéria jako výše s $I = \{1, 2, 3, 4\}$). Pro $i \in I$ nechť hodnoty nabývané i -tým kritériem tvoří uspořádanou

množinu (A_i, \leq_i) . Ve výsledku nás pak zajímá „složené“ uspořádání na kartézském součinu $\prod_{i \in I} A_i$, které definujeme takto:

Definice 5.11. Uspořádání podle více kritérií.

Mějme systém uspořádaných množin (A_i, \leq_i) kde $i \in I$. Na množině $M := \prod_{i \in I} A_i$ definujeme binární relace \sqsubseteq a \preceq takto; pro $\ell, m \in M$ kde $\ell = (\ell_i : i \in I)$ a $m = (m_i : i \in I)$:

* (po složkách)

$$\ell \sqsubseteq m \quad \text{právě když} \quad \ell_i \leq_i m_i \text{ pro všechna } i \in I,$$

* (lexikograficky)

$$\ell \preceq m \quad \text{právě když existuje } j \in I \text{ tak, že } \ell_j \leq_j m_j \\ \text{a pro všechna } i \in I, i < j \text{ platí } \ell_i = m_i.$$

Komentář: Lexikografické uspořádání z Definice 5.11 je staré známé uspořádání slov ve slovníku podle abecedy – kde I indexuje jednotlivá písmena slov zleva. Porovnejte si jej s detaily naší definice pro lepší porozumnění věci.

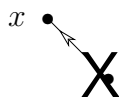
Tvrzení 5.12. *Nechť \sqsubseteq a \preceq jsou binární relace na množině M z Definice 5.11. Pak se v obou případech jedná o relace uspořádání. Navíc, pokud všechny (A_i, \leq_i) jsou lineárně uspořádané množiny, tak lexikografické uspořádání (M, \preceq) nad nimi je taktéž lineární.*

5.4 Další pojmy uspořádaných množin

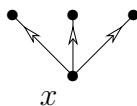
K tématu uspořádaných množin se vztahuje množství formálních pojmů, které potkáte v různých oblastech matematiky i informatiky.

Definice 5.13. Nechť (M, \preceq) je uspořádaná množina.

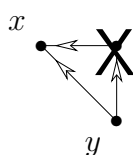
- $x \in M$ je *minimální* právě když pro každé $y \in M$ platí, že jestliže $y \preceq x$, pak $x \preceq y$. (Tj. x je minimální právě když neexistuje žádný prvek ostře menší než x .)



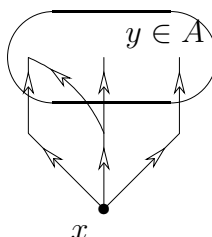
- $x \in M$ je *maximální* právě když pro každé $y \in M$ platí, že jestliže $x \preceq y$, pak $y \preceq x$. (Tj. x je maximální právě když neexistuje žádný prvek ostře větší než x .)
- $x \in M$ je *nejmenší* právě když pro každé $y \in M$ platí, že $x \preceq y$.



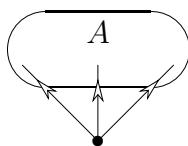
- $x \in M$ je *největší* právě když pro každé $y \in M$ platí, že $y \preceq x$.
- $x \in M$ *pokrývá* $y \in M$ právě když $x \neq y$, $y \preceq x$ a neexistuje žádné $z \in M$ takové, že $x \neq z \neq y$ a $y \preceq z \preceq x$.



- $x \in M$ je *dolní závora* (mez) množiny $A \subseteq M$ právě když $x \preceq y$ pro každé $y \in A$.



- $x \in M$ je *horní závora* (mez) množiny $A \subseteq M$ právě když $y \preceq x$ pro každé $y \in A$.
- $x \in M$ je *infimum* množiny $A \subseteq M$ právě když x je největší dolní závora (mez) množiny A .



- $x \in M$ je *supremum* množiny $A \subseteq M$, právě když x je nejmenší horní závora (mez) množiny A .
- $A \subseteq M$ je *řetězec* v uspořádání \preceq právě když (A, \preceq) je *lineárně* uspořádaná množina.



Komentář: Tuto dlouhou definici se sluší poněkud neformálně okomentovat. Za prvé, s pojmy nejmenšího a největšího prvku jste se už intuitivně setkali mnohokrát, ale (matematicky slabší) pojmy minimálního a maximálního působí někdy problémy. Zapamatujte si proto dobře, že minimálních prvků může mít množina několik, jsou to prostě všechny ty “vespod”, ale nejmenší prvek existuje nejvýše jeden a je to pouze ten *unikátní* minimální prvek množiny. Stejně pro maximální. . .

Další poznámka se vztahuje k infimu a supremu množiny. Jak jsme napsali (a asi totéž znáte z matematické analýzy), množina nemusí mít nejmenší ani největší prvek, ale v mnoha případech je lze “nahradit” po řadě infimem a supremem, které hrají v jistých ohledech podobnou roli. Avšak ani supremum a infimum nemusí vždy existovat.

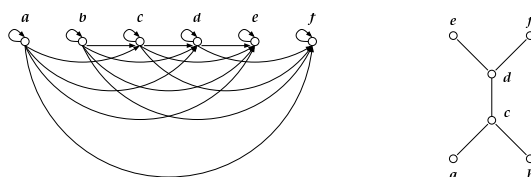
Nakonec si *dejte pozor*, že některé uvedené definice mají „ netriviální chování“ na nekonečných množinách. Proto je budeme obvykle uvažovat jen nad konečnými množinami.

Příklad 5.14. Proč má každá uspořádaná množina nejvýše jeden největší prvek?

Tvrzení dokážeme sporem: Necht' m i n jsou největší prvky uspořádané množiny (M, \preceq) . Pak podle Definice 5.13 platí $n \preceq m$ i $m \preceq n$ zároveň. Ovšem jelikož uspořádání musí být antisymetrické, pak platí $m = n$ a největší prvek je jen jeden. \square

Haseovské diagramy

Motivací zavedení tzv. Haseovských diagramů uspořádaných množin jsou přehlednější „obrázky“ než u grafů relací. Například si srovnajte následující ukázky:

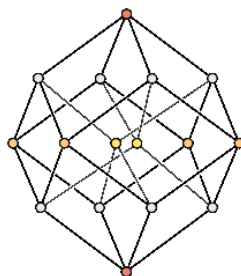


Zjednodušení a konvence přinesené následující definicí jsme ostatně měli možnost vidět už na některých z předchozích ilustračních obrázků uspořádání.

Definice: *Hasseovský diagram* konečné uspořádané množiny (M, \preceq) je jeho (jednoznačné) grafické znázornění získané takto:

- * Do první „horizontální vrstvy“ zakreslíme body odpovídající minimálním prvkům (M, \preceq) . (Tj. které *nepokrývají* nic. Pojem pokrývání prvku najdete v Definici 5.13.)
- * Máme-li již zakreslenou vrstvu i , pak do vrstvy $i + 1$ (která je „nad“ vrstvou i) zakreslíme všechny nezakreslené prvky, které *pokrývají pouze* prvky vrstev $\leq i$. Pokud prvek x vrstvy $i + 1$ pokrývá prvek y vrstvy $\leq i$, spojíme x a y neorientovanou hranou (tj. „čárou“).

Příklad 5.15. *Relaci inkluze na čtyřprvkové množině $\{a, b, c, d\}$ zakreslíme Hasseovským diagramem takto:*



□

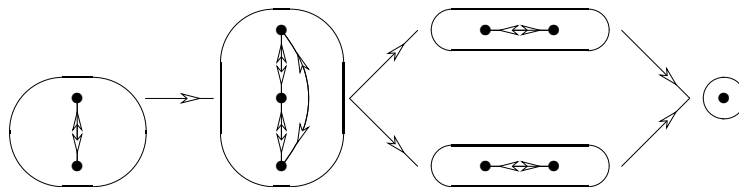
Komentář: Jak vidíme, v Hasseovském diagramu „vynecháváme“ ty hrany relace \preceq , které vyplývají z *reflexivity* či *tranzitivity*. To celý obrázek výrazně zpřehlední, a přitom nedochází ke ztrátě informace. Lze vynechat i šipky na hranách, neboť dle definice všechny míří „vzhůru“. Také pojem „vrstvy“ v definici je jen velmi neformální, důležité je, že větší (pokrývající) prvky jsou *nad menšími* (pokrývanými).

5.5 Relace předuspořádání

Mimo uspořádání chápaných *striktně antisymetricky* se v praxi často setkáme se „polouspořádáními“, ve kterých některé očividně různé prvky stojí na stejné úrovni naší preference. Co třeba dostaneme, když studenty na zkoušce „*uspořádáme*“ podle výsledné známky A–F? Bude to vždy uspořádání ve smyslu Definice 4.4? Samozřejmě ne, obvykle více než jeden student bude mít třeba známku E, což zjevně porušuje antisymetrii. Přesto cítíme, že studenty podle známky tak nějak uspořádané máme. Z toho důvodu je užitečné zavést i následující obecnější pojem:

Definice: Relace $R \subseteq M \times M$ je *předuspořádání* (také *kvaziuspořádání*, nebo *polouspořádání*) právě když R je *reflexivní* a *tranzitivní*.

Komentář: Rozdíl mezi uspořádáním a předuspořádáním je (jen neformálně řečeno!) v tom, že u předuspořádání srovnáváme prvky podle kritéria, které není pro prvek jedinečné. V předuspořádání takto mohou vznikat „balíky“ (třídy) se stejnou hodnotou kritéria, což schematicky ilustrujeme na následujícím obrázku.



Důležitým poznatkem je, že předuspořádání se „až tak moc“ neliší od dřívějšího uspořádání – přirozeně totiž odvodíme následující:

Věta 5.16. *Je-li \sqsubseteq předuspořádání na M , můžeme definovat relaci \sim na M předpisem*

$$x \sim y \quad \text{právě když} \quad x \sqsubseteq y \text{ a } y \sqsubseteq x.$$

Pak \sim je ekvivalence na M , která se nazývá jádro předuspořádání \sqsubseteq .

Na rozkladu M/\sim pak lze zavést relaci \preceq definovanou takto

$$[x] \preceq [y] \quad \text{právě když } x \sqsubseteq y.$$

Pak $(M/\sim, \preceq)$ je uspořádaná množina indukovaná \sqsubseteq .

Komentář: Pro ukázkou si vezměme relaci dělitelnosti na \mathbb{Z} . Pak třeba $-2 \sim 2$. Jádrem zde jsou dvojice čísel stejné absolutní hodnoty.

Důkaz (náznak): Tranzitivita a reflexivita relace \sim vyplývá z tranzitivity a reflexivity relace \sqsubseteq . Symetrie \sim pak je přímým důsledkem její definice. Tudíž \sim skutečně je relací ekvivalence a M/\sim je platný rozklad nosné množiny.

Tranzitivita a reflexivita relace \preceq se opět dědí z relace \sqsubseteq . Její antisymetrie vyplývá následující úvahou: Pokud $[x] \preceq [y]$ a $[y] \preceq [x]$, pak podle naší definice $x \sqsubseteq y$ a $y \sqsubseteq x$, neboli $x \sim y$ a $[x] = [y]$ podle definice tříd rozkladu. Pozor, nejdůležitější částí této věty důkazu je však ještě zdůvodnění, že naše podaná definice vztahu $[x] \preceq [y]$ je korektní, což znamená, že její platnost nezávisí na konkrétní volbě reprezentantů x z $[x]$ a y z $[y]$.

Poslední zmíněné tvrzení dokážeme sporem: Nechť $x, x' \in [x]$ a $y, y' \in [y]$ jsou (možná různí) reprezentanti dvou zkoumaných tříd rozkladu M/\sim , pro které však platí $x \sqsubseteq y$ a $x' \sqsupseteq y'$. Podle definice třídy rozkladu je $x \sim y$ a $x' \sim y'$ a z tranzitivity $x \sqsubseteq y \sqsubseteq y' \sqsubseteq x' \sqsubseteq x$ a tudíž $[x] = [y]$ a na volbě reprezentanta skutečně nezáleží. \square

Rozšiřující studium

S relacemi ekvivalence i jimi implicitně definovanými rozklady množin se lze setkat všude tam, kde nějaké objekty „rozdělujeme do přihrádek“ podle nějakých sdílených znaků nebo kritérií. Principy takových rozkladů vám zajisté byly intuitivně známy ještě dříve, než jste vůbec slyšeli o matematickém pojmu ekvivalence, v této lekci jsme si je jen zavedli na pevných formálních základech. Toto formální pojetí relací ekvivalence a rozkladů budete porůznu potkávat i v dalších infromatických předmětech, například u teorie automatů apod.

Druhá část lekce pojednává o látce, kterou opět určitě na intuitivní úrovni znáte (však schopnost si věci „uspořádat“ je jednou ze základních lidských dovedností). Přesto správné

matematické uchopení podstaty uspořádaných množin vyžaduje se prokousat několika nesnadnými formálními definicemi, které byly v této lekci vysvětleny a okomentovány. Všimněte si, že hlavní těžkosti konceptu uspořádání se vztahují k částečným, tedy ne-lineárním uspořádáním, kdy běžnému lidskému uvažování není zcela intuitivně jasná práce s nesrovnatelnými dvojicemi. S trochou příslušného matematického formalismu se správné nakládání s konečnými částečnými uspořádáními stane snadnou rutinou.

6 Funkce a skládání, Induktivní definice

Úvod

Na závěr učiva o relacích se vracíme k úvodu Lekce 4, k pojmu abstraktní funkce coby speciálního případu relace a ke skládání funkcí. Kde jste se již přirozeně se skládáním funkcí setkali – jak například spočítáte na kalkulačce výsledek složitějšího vzorce? K funkcím se váže několik dalších základních matematických pojmů, jako je injektivní, surjektivní a bijektivní funkce. Operace s funkcemi si osvětlíme na příkladu permutací a jejich skládání.

Mimo to se ještě zobecněně vrátíme k problematice „postupných“ (induktivních a rekurentních) definic a vztahů ze závěru Lekce 3. Jedná se vlastně o matematické analogie rekursivních programů a jejich správné formální pochopení oceníme třeba i při programování samotném.

Cíle

V této lekci definujeme základní vlastnosti funkcí a především popíšeme a podrobně rozebereme skládání funkcí jako relací (jako model nám poslouží permutace). Na závěr se stručně podíváme na problematiku induktivních definic funkcí, coby na rozšíření rekurentních vztahů z Oddílu 3.4.

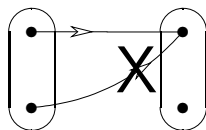
6.1 Vlastnosti funkcí

Funkce je intuitivně chápána jako předpis, který každé hodnotě (či hodnotám v případě více proměnných) levé strany přiřadí jednoznačně hodnotu pravé strany, tzv. *funkční hodnotu* neboli *výsledek*. Avšak v nejobecnější podobě pojem funkce nemusí být nijak svázaný s konkrétním předpisem jejího výpočtu; tento obecnější náhled je obzvláště potřebný v informatice, kdy se budete setkávat i s funkcemi, které *nejsou algoritmicky vyčíslitelné*.

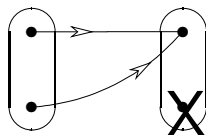
Připomeňme si, že funkce je podle Definice 4.2 speciálním případem relace, mající pro každou hodnotu levé strany jedinou (funkční) hodnotu pravé strany. V tomto oddíle si uvedeme několik vlastností specifických pro funkce.

Definice 6.1. *Funkce* (případně parciální funkce) $f : A \rightarrow B$ je

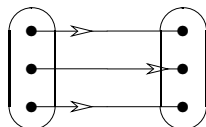
- * *injektivní* (nebo také *prostá*) právě když pro každé $x, y \in A$, $x \neq y$ platí $f(x) \neq f(y)$;



- * *surjektivní* (nebo také „na“) právě když pro každé $y \in B$ existuje $x \in A$ takové, že $f(x) = y$;



- * *bijektivní* (vzáj. jednoznačná) právě když je injektivní a současně surjektivní.



Komentář: Následují jednoduché ukázky vlastností funkcí.

* Funkce $plus : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ je surjektivní, ale není prostá.

* Funkce $g : \mathbb{Z} \rightarrow \mathbb{N}$ daná předpisem

$$g(x) = \begin{cases} -2x - 1 & \text{jestliže } x < 0, \\ 2x & \text{jinak} \end{cases}$$

je bijektivní.

* Funkce $\emptyset : \emptyset \rightarrow \emptyset$ je bijektivní.

* Funkce $\emptyset : \emptyset \rightarrow \{a, b\}$ je injektivní, ale není surjektivní.

Příklad 6.2. Dokázali byste nalézt jednoduše (tj. „hezky“) vypočitatelnou bijektivní funkci $\mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N}$?

V řešení si problém nejprve zkusíme představit vhodným geometrickým modelem (není to nutné, ale velmi dobré pro pochopení). Množina $\mathbb{N} \times \mathbb{N}$ nechť je představována všemi jednotkovými čtverečky v prvním kvadrantu roviny. Celý tento kvadrant postupně vyplňujeme po krocích většími čtverci z levého dolního rohu, tj. postupně o rozměrech 1×1 , 2×2 , 3×3 , atd. Co nám v každém kroku i přibude (postupujeme od $i = 0$)? Bude to pásek $2i + 1$ jednotkových čtverečků ve tvaru \sqcap . Pokud tyto pásy narovnáme a seřadíme za sebou, dostaneme jednostranně nekonečný pás jednotkových čtverečků, což znamená pro nás množinu \mathbb{N} . Kýžená bijekce je na světě!

Nyní zbývá formální stránka věci – zapsat tuto bijekci předpisem, čili vzorcem. K tomu si připomeneme, že $\mathbb{N} = \{0, 1, 2, 3, \dots\}$. Jak si sami můžete snadno ověřit, předpis výpočtu hodnoty $f(n)$, kde $f : \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N}$ je výše geometricky popsaná bijekce, je následující:

* Nalezneme největší k takové, že $k^2 \leq n$ ($k := \lfloor \sqrt{n} \rfloor$).

* Pro $n - k^2 \leq k$ máme $f(n) := (k, n - k^2)$,
kdežto pro $n - k^2 > k$ je $f(n) := (k^2 + 2k - n, k)$. □

Příklad 6.3. Pro jaké hodnoty parametrů a, b je funkce $f(x) = ax^3 + 3x^2 + bx$ z \mathbb{R} do \mathbb{R} injektivní či surjektivní?

Nejprve rozebereme případ $a = 0$, kdy platí $f(0) = f(-b/3) = 1$. Pro $b \neq 0$ je $-b/3 \neq 0$, kdežto pro $b = 0$ zase platí $f(1) = f(-1) = 4$. Tudíž pro $a = 0$ funkce f nemůže být nikdy injektivní. Zároveň pro $a = 0$ nemůže být ani surjektivní, neboť pro každé x platí, že $f(x) = 3x^2 + bx \geq -b^2/12$.

Nechť tedy $a > 0$. Potom $\lim_{x \rightarrow -\infty} f(x) = -\infty$ a $\lim_{x \rightarrow \infty} f(x) = \infty$ a vzhledem ke spojitosti funkce f musí nabývat všech reálných hodnot a je surjektivní. Totéž platí pro $a < 0$. Zodpovězení otázky, kdy je f injektivní, již vyžaduje některé základní poznatky matematické analýzy. Ze školní znalosti průběhu polynomické funkce plyne, že ta je prostá právě tehdy, když nemá žádný lokální extrém. V případě kubického polynomu to konkrétně znamená, že první derivace nesmí být nulová ve dvou různých reálných bodech. Vyjádřeno příslušnou kvadratickou rovnicí, $f'(x) = 3ax^2 + 6x + b = 0$, dostáváme z jejího determinantu podmínku $36 - 12ab \leq 0$.

Závěr: Funkce f je surjektivní, právě když $a \neq 0$. Funkce f je injektivní, právě když $a \neq 0$ a $36 - 12ab \leq 0$. □

Inverze funkce

Inverze funkce je dána definicí inverze relace z Oddílu 4.4.

Komentář: Příklady inverzí pro běžné funkce:

- * Inverzí bijektivní funkce $f(x) = x + 1$ na \mathbb{Z} je funkce $f^{-1}(x) = x - 1$.
- * Inverzí prosté funkce $f(x) = e^x$ na \mathbb{R} je parciální funkce $f^{-1}(x) = \ln x$.
- * Funkce $g(x) = x \bmod 3$ není prostá na \mathbb{N} , a proto její inverzí je „jen“ relace $g^{-1} = \{(a, b) \mid a = b \bmod 3\}$. Konkrétně $g^{-1} = \{(0, 0), (0, 3), (0, 6), \dots, (1, 1), (1, 4), \dots, (2, 2), (2, 5), \dots\}$.

K pojmu inverze funkce se vztahují následující základní a jednoduché poznatky:

Tvrzení 6.4. *Mějme funkci $f : A \rightarrow B$. Pak její inverzní relace f^{-1} je*

- parciální funkce právě když f je prostá,*
- funkce právě když f je bijektivní.*

Důkaz vyplývá přímo z definic funkce a inverze relace. □

Tvrzení 6.5. *Je-li inverzní relace f^{-1} funkce f taktéž parciální funkcí, tak platí $f^{-1}(f(x)) = x$ pro všechna x z definičního oboru.*

Důkaz vyplývá přímo z definic funkce a inverze relace. □

6.2 Skládání funkcí, permutace

Obdobně jako u inverze, i skládání funkcí je dáno definicí skládání relací z Oddílu 4.4. Jelikož však definice skládání relací je poměrně složitá, je vhodné si uvést alternativní jednoduchou definici skládání pouze pro funkce.

Fakt: Mějme zobrazení (funkce) $f : A \rightarrow B$ a $g : B \rightarrow C$. Pak jejich složením coby relací v tomto pořadí vznikne zobrazení $(g \circ f) : A \rightarrow C$ definované

$$(g \circ f)(x) := g(f(x)).$$

Komentář:

- * Jak například na běžné kalkulačce vypočteme hodnotu funkce $\sin^2 x$? Složíme (v tomto pořadí) „elementární“ funkce $f(x) = \sin x$ a $g(x) = x^2$.
- * Jak bychom na „elementární“ funkce rozložili aritmetický výraz $2 \log(x^2 + 1)$? Ve správném pořadí složíme funkce $f_1(x) = x^2$, $f_2(x) = x + 1$, $f_3(x) = \log x$ a $f_4(x) = 2x$.
- * A jak bychom obdobně vyjádřili složením funkcí aritmetický výraz $\sin x + \cos x$? Opět je odpověď přímočará, vezmeme „elementární“ funkce $g_1(x) = \sin x$ a $g_2(x) = \cos x$, a pak je „složíme“ další funkcí $h(x, y) = x + y$. Vidíme však, že takto pojaté „skládání“ už nezapadá hladce do našeho zjednodušeného formalismu skládání relací.

Pro nedostatek prostoru si skládání funkcí s více parametry nedefinujeme, ale sami vidíte, že obdobné skládání se v programátorské praxi vyskytuje doslova „na každém rohu“ a ani se nad tím nepozastavujeme.

Skládání permutací

Po zbytek tohoto oddílu se zaměříme na permutace coby speciální případ (bijektivních) zobrazení, neboť dobře ilustrují problematiku skládání bez zbytečných technických komplikací a zároveň se jedná o užitečnou oblast diskrétní matematiky.

Definice: Necht' permutace π množiny $\{1, 2, \dots, n\}$ je určena seřazením jejích prvků (p_1, \dots, p_n) . Pak π je zároveň bijektivním zobrazením $\{1, \dots, n\} \rightarrow \{1, \dots, n\}$ definovaným předpisem $\pi(i) = p_i$. Tudíž lze permutace skládat jako relace podle Definice 4.7.

Poznámka: Všechny permutace množiny $\{1, 2, \dots, n\}$ spolu s operací skládání tvoří grupu, zvanou symetrická grupa S_n . Permutační grupy (podgrupy symetrické grupy) jsou velice důležité v algebře, neboť každá konečná grupa je vlastně isomorfní některé permutační grupě.

Komentář: Příkladem permutace vyskytující se v programátorské praxi je třeba zobrazení $i \mapsto (i + 1) \bmod n$ ("inkrement"). Často se třeba lze setkat (aniž si to mnohdy uvědomujeme) s permutacemi při indexaci prvků polí.

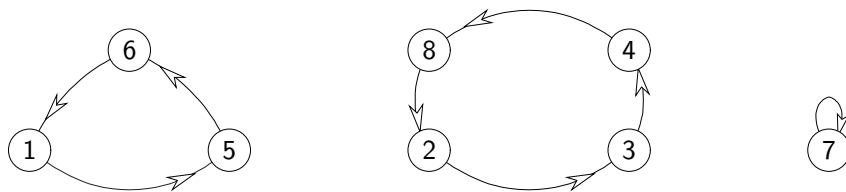
V kontextu pohledu na funkce a jejich skládání coby relací si zavedeme jiný, názornější, způsob zápisu permutací – pomocí jejich cyklů.

Definice: Necht' π je permutace na množině A . *Cyklem* v π rozumíme posloupnost $\langle a_1, a_2, \dots, a_k \rangle$ různých prvků A takovou, že $\pi(a_i) = a_{i+1}$ pro $i = 1, 2, \dots, k - 1$ a $\pi(a_k) = a_1$.

Jak název napovídá, v zápise cyklu $\langle a_1, a_2, \dots, a_k \rangle$ není důležité, kterým prvkem začneme, ale jen dodržení cyklického pořadí. Cyklus v permutaci může mít i jen jeden prvek (zobrazený na sebe).

Komentář: Nakreslete si (vámi zvolenou) permutaci π obrázkem, ve kterém vedete šipku vždy od prvku i k prvku $\pi(i)$. Pak uvidíte, že cykly dle naší definice jsou právě cykly tvořené šipkami ve vašem obrázku. S tímto grafickým zobrazením pro vás nebude problém pochopit následující látku.

Například permutaci $(5, 3, 4, 8, 6, 1, 7, 2)$ si lze obrázkem nakreslit takto:



Reprezentace permutací jejich cykly

Věta 6.6. Každou permutaci π na konečné množině A lze zapsat jako složení cyklů na disjunktních podmnožinách (rozkladu) A .

Důkaz: Vezmeme libovolný prvek $a_1 \in A$ a iterujeme zobrazení $a_2 = \pi(a_1)$, $a_3 = \pi(a_2)$, atd., až se dostaneme „zpět“ k $a_{k+1} = \pi(a_k) = a_1$. Proč tento proces skončí? Protože A je konečná a tudíž ke zopakování některého prvku a_{k+1} musí dojít. Nadto je π prostá, a proto nemůže nastat $\pi(a_k) = a_j$ pro $j > 1$. Takto získáme první cyklus $\langle a_1, \dots, a_k \rangle$.

Induktivně pokračujeme s hledáním dalších cyklů ve zbylé množině $A' = A \setminus \{a_1, \dots, a_k\}$, dokud nezůstane prázdná. V tomto indukčním kroku si musíme uvědomit, že π omezené na nosnou množinu A' je stále permutací podle definice (neboli žádná prvek z A' se nezobrazí do $\{a_1, \dots, a_k\}$). \square

Značení permutace jejími cykly: Necht' se permutace π podle Věty 6.6 skládá z cyklů $\langle a_1, \dots, a_k \rangle$, $\langle b_1, \dots, b_l \rangle$ až třeba $\langle z_1, \dots, z_m \rangle$. Pak zapíšeme

$$\pi = (\langle a_1, \dots, a_k \rangle \langle b_1, \dots, b_l \rangle \dots \langle z_1, \dots, z_m \rangle).$$

Komentář: Primitivní pseudonáhodné generátory v počítačích iterují z náhodného počátku permutací danou vztahem $i \mapsto (i + p) \bmod q$. Je pochopitelné, že tato permutace nesmí obsahovat krátké cykly, lépe řečeno, měla by se skládat z jediného (dlouhého) cyklu. (Pro úplnost, jedná se o permutaci množiny $\{0, 1, \dots, q - 1\}$).

Příklad 6.7. Ukázka skládání permutací daných svými cykly.

Vezmeme 7-prvkovou permutaci $\pi = (3, 4, 5, 6, 7, 1, 2)$. Ta se skládá z jediného cyklu $\langle 1, 3, 5, 7, 2, 4, 6 \rangle$. Jiná permutace $\sigma = (5, 3, 4, 2, 6, 1, 7)$ se rozkládá na tři cykly $\langle 1, 5, 6 \rangle$, $\langle 2, 3, 4 \rangle$ a $\langle 7 \rangle$. Nyní určíme složení $\sigma \circ \pi$ těchto dvou permutací (už přímo v zápisu cykly):

$$(\langle 1, 5, 6 \rangle \langle 2, 3, 4 \rangle \langle 7 \rangle) \circ (\langle 1, 3, 5, 7, 2, 4, 6 \rangle) = (\langle 1, 4 \rangle \langle 2 \rangle \langle 3, 6, 5, 7 \rangle)$$

(Nezapomínejme, že první se ve složení aplikuje pravá permutace!)

Postup skládání jsme použili následovný: 1 se zobrazí v permutaci vpravo na 3 a pak vlevo na 4. Následně 4 se zobrazí na 6 a pak na 1. Tím „uzavřeme“ první cyklus $\langle 1, 4 \rangle$. Dále se 2 zobrazí na 4 a pak hned zpět na 2, tj. má samostatný cyklus. Zbylý cyklus $\langle 3, 6, 5, 7 \rangle$ určíme analogicky. \square

6.3 Induktivní definice množin a funkcí

Dalším výkladem se vracíme k podstatě množin a funkcí a k jejich popisu. Vzpomeňme si na definici posloupnosti *rekurentním vztahem* z Oddílu 3.4. Příným zobecněním dřívějších rekurentních definic je následující koncept.

Definice 6.8. Induktivní definice množiny.

Jedná se obecně o popis (nějaké) množiny M v následujícím tvaru:

- Je dáno několik pevných (*bázických*) prvků $a_1, a_2, \dots, a_k \in M$.
- Je dán soubor *induktivních pravidel* typu

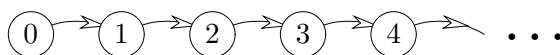
Jsou-li (libovolné prvky) $x_1, \dots, x_\ell \in M$, pak také $y \in M$.

V tomto případě je y typicky funkcí $y = f_i(x_1, \dots, x_\ell)$.

Pak naše *induktivně definovaná množina* M je určena jako nejmenší (inkluzí) množina vyhovující těmto pravidlům.

Komentář: Několik ukázek...

- Pro nejbližší příklad induktivní definice se obrátíme na množinu všech přirozených čísel, která je formálně zavedena následovně.
 - $0 \in \mathbb{N}$
 - Je-li $i \in \mathbb{N}$, pak také $i + 1 \in \mathbb{N}$.



- Pro každé $y \in \mathbb{N}$ můžeme definovat jinou množinu $M_y \subseteq \mathbb{N}$ induktivně takto:

- $y \in M_y$
- Jestliže $x \in M_y$ a $x + 1$ je liché, pak $x + 2 \in M_y$.

Pak například $M_3 = \{3\}$, nebo $M_4 = \{4 + 2i \mid i \in \mathbb{N}\}$.

- Dalším příkladem induktivní definice je už známé zavedení *výrokových formulí* z Oddílu 1.5. Uměli byste teď přesně říci, co tam byly *bázické prvky* a jaká byla *induktivní pravidla*? A jaká byla v definici formulí *role přítomných závorek*? K tomu se blíže vyjádříme v Definici 6.9.

Jednoznačnost induktivních definic

Definice: Řekneme, že daná induktivní definice množiny M je *jednoznačná*, právě když každý prvek M lze odvodit z *bázických prvků* pomocí *induktivních pravidel* právě *jedním způsobem*.

Komentář: Definujme například množinu $M \subseteq \mathbb{N}$ induktivně takto:

- $2, 3 \in M$
- Jestliže $x, y \in M$ a $x \leq y$, pak také $x^2 + y^2$ a $x \cdot y$ jsou prvky M .

Proč tato induktivní definice není jednoznačná? Například číslo $8 \in M$ lze odvodit způsobem $8 = 2 \cdot (2 \cdot 2)$, ale zároveň zcela jinak $8 = 2^2 + 2^2$.

V čem tedy spočívá důležitost jednoznačných induktivních definic množin? Je to především v dalším možném využití induktivní definice množiny jako „základny“ pro odvozené vyšší definice, viz následující Definice 6.9 a třeba doplňková Věta 6.12. Stručně a neformálně řečeno, hlavní role jednoznačnosti induktivní definice je v možnosti pak přiřadit prvkům této induktivní množiny nějaký „jednoznačný význam“.

Induktivně definovaná množina povětšinou nemá valný význam sama o sobě, avšak poskytuje *definiční obor* pro následnou induktivně definovanou funkci:

Definice 6.9. Induktivní definice funkce z induktivní množiny.

Nechť množina M je dána jednoznačnou induktivní definicí. Pak říkáme, že funkce $\mathcal{F} : M \rightarrow X$ je definována *induktivně* (vzhledem k induktivní definici M), pokud je řečeno:

- Pro každý z *bázických prvků* $a_1, a_2, \dots, a_k \in M$ je určeno $\mathcal{F}(a_i) = c_i$, kde c_i je konstanta.
- Pro každé induktivní pravidlo typu

“Jsou-li (libovolné prvky) $x_1, \dots, x_\ell \in M$, pak také $f(x_1, \dots, x_\ell) \in M$ ”

je definováno

$\mathcal{F}(f(x_1, \dots, x_\ell))$ na základě hodnot $\mathcal{F}(x_1), \dots, \mathcal{F}(x_\ell)$.

Komentář: Ilustrujme si induktivní definici funkce dětskou hrou na „tichou poštu“. Definičním oborem je řada sedících hráčů, kde ten první je *bázickým prvkem* a každý následující (mimo posledního) odvozuje hráče sedícího hned za ním jako další prvek hry. Hodnotou *bázického prvku* je první (vymyšlené) posílané slovo. Induktivní pravidlo pak následujícímu hráči přiřazuje slovo, které je odvozeno ze („zkomolením“) slova předchozího hráče. Výsledkem hry pak je hodnota–slovo posledního hráče.

Pro další příklad se podívejme třeba do manuálových stránek unixového příkazu `test EXPRESSION`:

EXPRESSION is true or false and sets exit status. It is one of:	
(EXPRESSION)	EXPRESSION is true
! EXPRESSION	EXPRESSION is false
EXPRESSION1 -a EXPRESSION2	both EXPRESSION1 and EXPRESSION2 are true
EXPRESSION1 -o EXPRESSION2	either EXPRESSION1 or EXPRESSION2 is true
[-n] STRING	the length of STRING is nonzero
STRING1 = STRING2	the strings are equal
.....	

Vidíte, jak tato ukázka koresponduje s Definicí 6.9? No, ne úplně, poněkud problematická je otázka jednoznačnosti této definice – jednoznačnost není vynucena (jen umožněna) syntaktickými pravidly, jinak je pak dána nepsanými konvencemi implementace příkazu. To je pochopitelně z matematického hlediska velmi špatně, ale přesto jde o pěknou ukázkou z praktického života informatika.

Induktivní definice se „strukturální“ indukcí

Závěrem ještě doplňkově zařazujeme malou ukázkou, která přirozeně zkombinovat indukční definice s „pokročilou formou matematické indukce“ v dokazování, s tzv. *strukturální indukci*.

Příklad 6.10. *Jednoduché aritmetické výrazy a jejich význam.*

Nechť je dána abeceda $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \odot, \oplus, (,)\}$. Definujme množinu *jednoduchých výrazů* $SExp \subseteq \Sigma^*$ induktivně takto:

- Dekadický zápis každého přirozeného čísla n je prvek $SExp$.
- Jestliže $x, y \in SExp$, pak také $(x) \odot (y)$ a $(x) \oplus (y)$ jsou prvky $SExp$.
- Jak vidíme, díky nucenému závorkování je tato indukční definice jednoduchých výrazů (nikoliv jejich „hodnot“) *jednoznačná*.

Tímto jsme aritmetickým výrazům přiřadili jejich „formu“, tedy *syntaxi*.

Pro přiřazení „významu“, tj. *sémantiky* aritmetického výrazu, následně definujme funkci $Val: SExp \rightarrow \mathbb{N}$ induktivně takto:

- Bázické prvky: $Val(\mathbf{n}) = n$, kde \mathbf{n} je dekadický zápis přirozeného čísla n .
- První indukční pravidlo: $Val((x) \oplus (y)) = Val(x) + Val(y)$.
- Druhé indukční pravidlo: $Val((x) \odot (y)) = Val(x) \cdot Val(y)$.

Co je tedy správným významem („hodnotou“) uvedených aritmetických výrazů? (Příklad 6.11) □

Příklad 6.11. *Důkaz správnosti přiřazeného „významu“ $Val: SExp \rightarrow \mathbb{N}$.*

Věta. *Pro každý výraz $s \in SExp$ je hodnota $Val(s)$ z Příkladu 6.10 číselně rovna výsledku vyhodnocení výrazu s podle běžných zvyklostí aritmetiky.*

Jelikož pojednáváme o indukčně definované funkci Val , je přirozené pro důkaz jejich vlastností aplikovat *matematickou indukci*. Na rozdíl od dříve probíraných příkladů zde nevidíme žádný celočíselný „parametr n “, a proto si jej budeme muset nejprve definovat. Naši indukci tedy povedeme podle „délky ℓ odvození výrazu s “ definované jako *počet aplikací indukčních pravidel* potřebných k odvození $s \in SExp$.

Důkaz: V *bázi indukce* ověříme vyhodnocení bázických prvků, kteréžto jsou zde dekadické zápisy přirozených čísel. Platí $Val(\mathbf{n}) = n$, což skutečně odpovídá zvyklostem aritmetiky.

V indukčním kroku se podíváme na vyhodnocení $Val((x) \oplus (y)) = Val(x) + Val(y)$. Podle běžných zvyklostí aritmetiky by hodnota $Val((x) \oplus (y))$ měla být rovna součtu vyhodnocení výrazu x , což je podle indukčního předpokladu rovno $Val(x)$ (x má zřejmě kratší délku odvození), a vyhodnocení výrazu y , což je podle indukčního předpokladu rovno $Val(y)$. Takže skutečně $Val((x) \oplus (y)) = Val(x) + Val(y)$.

Druhé pravidlo $Val((x) \odot (y))$ se dořeší analogicky. □

Dodatek: důkaz pro normální tvar formule

V Oddíle 1.5 jsme stručně zavedli výrokovou matematickou logiku a výrokové formule. Nyní si snadno můžeme všimnout, že jak definice syntaxe, tak i definice sémantiky výrokové logiky jsou induktivní ve smyslu (po řadě) Definic 6.8 a 6.9. Také Metoda 1.19 pro převod formule do normálního tvaru je induktivní, a proto je na místě následující ilustrativní ukáзка důkazu strukturální indukci:

Věta 6.12. *Pro libovolnou výrokovou formuli φ platí (viz Metoda 1.19), že*

- a) $\mathcal{F}(\varphi)$ je ekvivalentní formule k φ v normálním tvaru
- b) a $\mathcal{G}(\varphi)$ je formule v normálním tvaru ekvivalentní negaci $\neg\varphi$.

Důkaz povedeme indukci ke struktuře formule, neboli indukci povedeme podle „délky“ ℓ – počtu aplikací induktivních pravidel při sestavování formule φ .

- Báze indukce ($\ell = 0$): Pro všechny atomy, tj. výrokové proměnné, zřejmě platí, že $\mathcal{F}(A) = A$ je ekvivalentní A a $\mathcal{G}(A) = \neg A$ je ekvivalentní $\neg A$.
- V indukčním kroku předpokládejme, že a) i b) platí pro všechny formule φ délky nejvýše ℓ . Vezmeme si formuli ψ délky $\ell + 1$, která je utvořená jedním z následujících způsobů:

- * $\psi \equiv \neg\varphi$ (\equiv je „definiční rovnítko“ pro formule). Podle výše uvedeného induktivního předpisu je $\mathcal{F}(\psi) = \mathcal{F}(\neg\varphi) = \mathcal{G}(\varphi)$. Podle indukčního předpokladu pak je $\mathcal{G}(\varphi)$ formule v normálním tvaru ekvivalentní $\neg\varphi = \psi$. Obdobně pro funktor \mathcal{G} vyjádříme $\mathcal{G}(\psi) = \mathcal{G}(\neg\varphi) = \mathcal{F}(\varphi)$. Podle indukčního předpokladu pak je $\mathcal{F}(\varphi)$ formule v normálním tvaru ekvivalentní φ a to je dále ekvivalentní $\neg\neg\varphi = \neg\psi$ podle Tvzení 1.14.

- * $\psi \equiv (\varphi_1 \Rightarrow \varphi_2)$. Podle výše uvedeného induktivního předpisu je $\mathcal{F}(\psi) = \mathcal{F}(\varphi_1 \Rightarrow \varphi_2) = \mathcal{F}(\varphi_1) \Rightarrow \mathcal{F}(\varphi_2)$. Podle indukčního předpokladu jsou $\mathcal{F}(\varphi_1)$ i $\mathcal{F}(\varphi_2)$ formule v normálním tvaru ekvivalentní φ_1 a φ_2 . Potom i $\mathcal{F}(\varphi_1) \Rightarrow \mathcal{F}(\varphi_2)$ je v normálním tvaru dle definice a podle sémantiky \Rightarrow je ta ekvivalentní formuli $(\varphi_1 \Rightarrow \varphi_2) = \psi$.

Obdobně rozepíšeme $\mathcal{G}(\psi) = \mathcal{G}(\varphi_1 \Rightarrow \varphi_2) = \mathcal{F}(\varphi_1) \wedge \mathcal{G}(\varphi_2)$. Jelikož \wedge je pro nás jen zkratka, výraz dále rozepíšeme $\mathcal{G}(\psi) = \neg(\mathcal{F}(\varphi_1) \Rightarrow \neg\mathcal{G}(\varphi_2))$. Podle indukčního předpokladu (a dvojí negace) jsou $\mathcal{F}(\varphi_1)$ a $\neg\mathcal{G}(\varphi_2)$ po řadě ekvivalentní formulím φ_1 a φ_2 . Tudíž nakonec odvodíme, že $\mathcal{G}(\psi)$ je ekvivalentní negaci formule $\varphi_1 \Rightarrow \varphi_2$, což jsme zde měli dokázat.

- * $\psi \equiv (\varphi_1 \vee \varphi_2)$. Zde si musíme opět uvědomit, že spojka \vee je pro nás jen zkratka, a přepsat $\psi \equiv (\neg\neg\varphi_1 \Rightarrow \varphi_2)$. Potom podle předchozích dokázaných případů víme, že $\mathcal{F}(\psi) = \mathcal{F}(\neg\neg\varphi_1 \Rightarrow \varphi_2) = \mathcal{F}(\neg\neg\varphi_1) \Rightarrow \mathcal{F}(\varphi_2)$ je ekvivalentní formuli $(\neg\neg\varphi_1 \Rightarrow \varphi_2) = \psi$, což bylo třeba dokázat. Stejně tak $\mathcal{G}(\psi) = \mathcal{G}(\neg\neg\varphi_1 \Rightarrow \varphi_2) = \mathcal{F}(\neg\neg\varphi_1) \wedge \mathcal{G}(\varphi_2)$ je podle předchozích případů důkazu ekvivalentní $(\neg\neg\varphi_1 \wedge \neg\varphi_2) = \neg\psi$.

* $\psi \equiv (\varphi_1 \wedge \varphi_2)$ a $\psi \equiv (\varphi_1 \Leftrightarrow \varphi_2)$ už dokončíme analogicky. □

6.4 Uzávěry relací

Posledním bodem našeho výkladu je vysvětlení postupu, jak danou relaci můžeme „obohatit“ o zvolenou vlastnost (například proto, že naše data o relaci jsou neúplná a vlastnost je tak poškozena). Toto se pochopitelně týká pouze vlastností, které lze nějak ustanovit prostým přidáním dvojic do existující relace, jak říká následující definice.

Definice: Bud' V (nějaká) vlastnost binárních relací. Řekneme, že V je *uzavíratelná*, pokud splňuje následující podmínky:

- * Pro každou množinu M a každou relaci $R \subseteq M \times M$ existuje alespoň jedna relace $S \subseteq M \times M$, která má vlastnost V a pro kterou platí $R \subseteq S$.
- * Nechť I je množina a nechť $R_i \subseteq M \times M$ je relace mající vlastnost V pro každé $i \in I$. Pak relace $\bigcap_{i \in I} R_i$ má vlastnost V .

Fakt: Libovolná kombinace vlastností *reflexivita*, *symetrie*, *tranzitivita* je uzavíratelná vlastnost. Ireflexivita a antisymetrie nejsou uzavíratelné vlastnosti.

Věta 6.13. *Nechť V je uzavíratelná vlastnost binárních relací. Bud' M množina a R libovolná binární relace na M . Pak pro množinu všech relací $S \supseteq R$ na M majících vlastnost V existuje infimum R_V (vzhledem k množinové inkluzi), které samo má vlastnost V .*

Definice: Tuto infimální relaci R_V s vlastností V nazýváme *V -uzávěr* relace R .

Zmíněný V -uzávěr relace je vlastně daný induktivní definicí (viz Definice 6.8), kde báze je původní relace a induktivní pravidla odpovídají vlastnosti V .

Tvrzení 6.14. Nechť R je binární relace na M . Pak platí následující poznatky.

- * *Reflexivní uzávěr* R je přesně relace $R \cup \{(x, x) \mid x \in M\}$.
- * *Symetrický uzávěr* R je přesně relace $\overset{\leftrightarrow}{R} = \{(x, y) \mid (x, y) \in R \text{ nebo } (y, x) \in R\}$.
- * *Tranzitivní uzávěr* R je přesně relace $R^+ = \bigcup_{i=1}^{\infty} \mathcal{T}^i(R)$, kde \mathcal{T} je funkce, která pro každou binární relaci S vrátí relaci

$$\mathcal{T}(S) = S \cup \{(x, z) \mid \text{existuje } y \text{ takové, že } (x, y), (y, z) \in S\}$$

a $\mathcal{T}^i = \underbrace{\mathcal{T} \circ \dots \circ \mathcal{T}}_i$ je i -krát iterovaná aplikace funkce \mathcal{T} .

- * Reflexivní a tranzitivní uzávěr R je přesně relace $R^* = Q^+$, kde Q je reflexivní uzávěr R .
- * Reflexivní, symetrický a tranzitivní uzávěr R (tj. nejmenší ekvivalence obsahující R) je přesně relace $(\overset{\leftrightarrow}{Q})^+$, kde Q je reflexivní uzávěr R .

Poznámka: Na pořadí aplikování uzávěrů vlastností záleží! Zhruba řečeno, tranzitivní uzávěr aplikujeme coby poslední.

Komentář: Význam reflexivních a symetrických uzávěrů je z předchozího docela zřejmý. Význam tranzitivního uzávěru R^+ je následovný: Do R^+ přidáme všechny ty dvojice (x, z) takové, že v R se lze „dostat po šípkách“ z x do z . Nakreslete si to na papír pro nějakou jednoduchou relaci, abyste význam tranzitivního uzávěru lépe pochopili.



Například buď $R \subseteq \mathbb{N} \times \mathbb{N}$ definovaná takto: $R = \{(i, i+1) \mid i \in \mathbb{N}\}$. Pak tranzitivní uzávěr R^+ je běžné ostré lineární uspořádání $<$ přirozených čísel. Zájemce o jednoduchý algoritmus pro výpočet tranzitivního uzávěru odkazujeme na Oddíl 11.4.

A jak bylo dříve řečeno, antisymetrický uzávěr relace prostě nedává smysl.

Příklad 6.15. *Studenti předmětu IB000 sedí v lavicích v obdélníkové posluchárně (však to dobře znáte). Ne všechny sedačky jsou nutně obsazené. Definujeme relaci R tak, že dvojice studentů (a, b) náleží do R , právě když b sedí ve stejné řadě vlevo od a nebo b sedí ve stejném sloupci před a .*

a) *Dokažte, že tranzitivní uzávěr R je antisymetrická relace.*

b) *Kolik nejméně a nejvíce tříd ekvivalence může definovat reflexivní, symetrický a tranzitivní uzávěr relace R ?*

Část (a) dokážeme sporem; necht' tedy platí $(a, b), (b, a) \in R^+$ kde $a \neq b$. Proč by toto nemělo nastat? Pro důkaz si pomůžeme jednoduchým trikem: pro studenta x definujeme hodnotu $s(x) := i + j$, kde i je řada a j sloupec, ve kterých x sedí, číslováno zleva a zepředu. Pak podle popisu relace R zřejmě platí $s(x) > s(y)$ pokud $(x, y) \in R$. Podle definice tranzitivního uzávěru pak $s(a) > s(b)$ jakmile $(a, b) \in R^+$ a $a \neq b$, avšak také $s(a) < s(b)$ jelikož $(b, a) \in R^+$. To je zřejmě spor.

Nyní přejdeme k části (b). Označme S reflexivní, symetrický a tranzitivní uzávěr relace R . Nejméně tříd, jednu, získáme při plném obsazení posluchárny, ať už má jakékoliv rozměry. Vezměme kterékoliv dva různé studenty a, b . Podle definice reflexivního a symetrického uzávěru existuje student c sedící ve stejné řadě jako a a ve stejném sloupci jako b (může to být i jeden z a, b), přičemž platí $(a, c), (c, b) \in S$. Podle definice tranzitivního uzávěru pak $(a, b) \in S$, a proto a, b leží ve stejné jedné třídě podle Věty 5.7.

Naopak nejvíce tříd získáme, pokud žádní dva studenti nesedí ve stejné řadě nebo stejném sloupci (třeba sedí jen na jedné úhlopříčce). Počet tříd pak zřejmě je ℓ rovno minimu z počtu řad a sloupců posluchárny. Předpokládejme pro spor, že by mohlo existovat více než ℓ tříd ekvivalence uzávěru S . Zvolme z každé třídy jednoho studenta jako reprezentanta. Podle Dirichletova principu někteří dva a, b z vybraných studentů sedí ve stejné řadě či sloupci. Podle definice naší relace R a symetrického uzávěru platí $(a, b) \in S$. Avšak a, b pochází z různých tříd ekvivalence a to je ve sporu s Větou 5.7. \square

Příklad 6.16. *Proč při výpočtu tranzitivního uzávěru relace na konečné množině podle vzorce $R^+ = \bigcup_{i=1}^{\infty} \mathcal{T}^i(R)$ vždy stačí uvažovat konečně mnoho členů tohoto sjednocení?*

Pro odpověď si uvědomme zásadní fakt — pokud $\mathcal{T}^{i+1} = \mathcal{T}^i$, tak už platí $\mathcal{T}^{i+k} = \mathcal{T}^i$ pro všechna přirozená k . Neboli je potřeba sjednotit jen tolik prvních členů, dokud se ony „zvětšují“, což může nastat jen konečně krát nad konečnou množinou. Mimo jiné tak vidíme, že uvedený popis tranzitivního uzávěru je *konstruktivní*. \square

Rozšiřující studium

S formalizací pojmu funkce a jejími vlastnostmi se setkáváte především v matematice, avšak například na bijektivní funkce narazíte při ukládání dat při volbě klíče apod. Naším cílem bylo ukázat práci s funkcemi v jejich abstraktní podobě, tj. bez vazby na nějaký konkrétní analytické vzoreček.

Poslední částí látky o množinách a relacích je problematika induktivních definic, které ač ve formálním podání mohou nejprve vypadat nepochopitelně, jsou ve skutečnosti zcela přirozenou popisnou metodou v mnoha aplikačních sférách informatiky a jejich alespoň intuitivní chápání pro vás bude v dalším studiu nezbytné. Schválně, zkuste se podívat zrovna do vaší oblíbené oblasti informatiky, kde všude induktivní definice, tj. ty odvolávající se rekurzivně samy na sebe pro „menší případy“, najdete (třeba nepřímo).

7 Pojem grafu

Úvod

Po relacích se náš výklad upře směrem k další základní struktuře diskrétní matematiky. Třebaže grafy (pozor, nepletejte si s grafem funkce) jsou jen jedním z mnoha objektů zkoumání v matematice a vlastně pouze speciálním případem binárních relací, vydobily si svou užitečností a názorností (a to především ve vztahu k informatice) důležité místo na slunci. Dá se tak bez nadsázky říci, že teorie grafů je asi nejvýznamnější součástí soudobé diskrétní matematiky, a proto se jí budeme věnovat po tři následující lekce.

Neformálně řečeno, graf se skládá z vrcholů, které si představíme jako nakreslené „puntíky“, a z hran, které spojují dvojice vrcholů mezi sebou. Svě důležité místo si grafy získaly dobře vyváženou kombinací vlastností – snadno pochopitelným názorným nakreslením a zároveň jednoduchým zpracováním na počítačích. Díky těmto vlastnostem se grafy prosadily jako vhodný matematický model pro popis různých vztahů mezi daty a objekty.

Cíle

Definujeme, co je graf a jaké jsou nejzákladnější grafové pojmy (třeba hrany a stupně, podgrafy, souvislost). Klademe důraz na to, aby se čtenář naučil grafy uchopit a účelně s nimi pracovat a také aby správně viděl „stejnost“ (isomorfismus) grafů. Poté se věnujeme nejjednoduššímu druhu grafů, totiž stromům. Jedná se vesměs o pojmy, které se hojně vyskytují v informatických aplikacích grafů a především pak ve většině základních algoritmů, které se student informatiky učí.

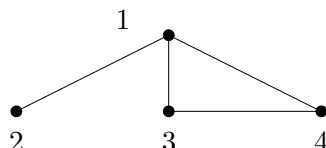
7.1 Definice grafu

Hned na úvod přistoupíme k formální definici grafu. Bude se jednat o definici tzv. *jednoduchého neorientovaného grafu*, který budeme považovat za základní, pokud neřekneme jinak. Svým způsobem navazujeme na reprezentace relací v Oddíle 4.2.

Definice 7.1. *Graf* je uspořádaná dvojice $G = (V, E)$, kde V je konečná množina *vrcholů* a E je množina *hran* – množina vybraných dvouprvkových podmnožin množiny vrcholů; tj. $E \subseteq \binom{V}{2}$.

Značení: Hranu mezi vrcholy u a v píšeme jako $\{u, v\}$, nebo zkráceně uv . Vrcholy spojené hranou jsou *sousední* a hrana uv *vychází* z vrcholů u a v . Na množinu vrcholů známého grafu G odkazujeme jako na $V(G)$, na množinu hran $E(G)$.

Komentář: Grafy se často zadávají přímo názorným obrázkem, jinak je lze formálně zadat výčtem vrcholů a výčtem hran. Například:



$$V = \{1, 2, 3, 4\}, \quad E = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{3, 4\}\}$$

Na graf se lze dívat také jako na symetrickou ireflexivní relaci, kde hrany tvoří právě dvojice prvků z této relace.

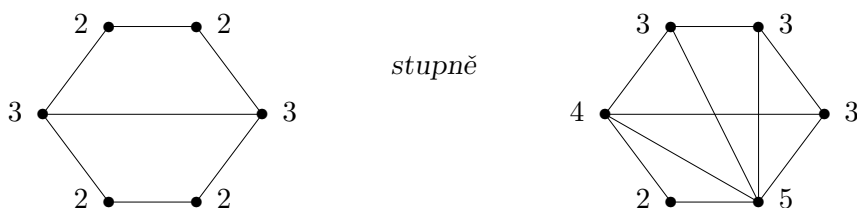
Stupně vrcholů v grafu

Máme-li graf, často nás zajímá, kolik z kterého vrcholu vychází hran-spojnic, neboli kolik má vrchol „sousedů“. Proto jedním z prvních definovaných pojmů bude stupeň vrcholu v grafu.

Definice 7.2. *Stupněm vrcholu* v v grafu G

rozumíme počet hran vycházejících z v . Stupeň v v grafu G značíme $d_G(v)$.

Komentář: Slovo „vycházející“ zde nenaznačuje žádný směr; je totiž obecnou konvencí u neorientovaných grafů říkat, že hrana vychází z obou svých konců zároveň.



Například v nakreslené ukázce jsou stupně přímo zapsány u vrcholů.

Definice: Graf je d -*regulární*, pokud všechny jeho vrcholy mají stejný stupeň d .

Značení: *Nejvyšší* stupeň v grafu G značíme $\Delta(G)$ a *nejnižší* $\delta(G)$.

Věta 7.3. *Součet stupňů v grafu je vždy sudý, roven dvojnásobku počtu hran.*

Důkaz. Při sčítání stupňů vrcholů v grafu započítáme každou hranu dvakrát – jednou za každý její konec. Proto také výsledek vyjde sudý. \square

Příklad 7.4. *Zodpovězme následující otázky:*

- Kolik hran má graf se 17 vrcholy stupňů 4?*
- Existují dva „různé“ grafy se 6 vrcholy stupňů 2?*

V otázce (a) je součet stupňů $17 \cdot 4 = 68$ a podle Věty 7.3 je tudíž počet hran $68/2 = 34$.

Ano, v (b) existují dva velmi odlišné grafy uvedené vlastnosti, snadno je nakreslíme takto:

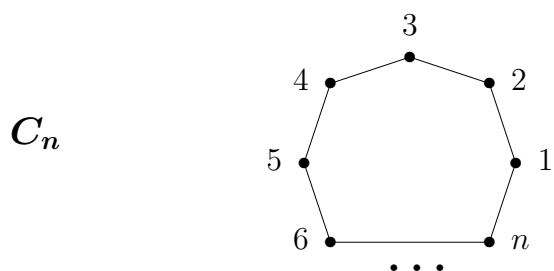


V dalším textu si zavedeme některé základní názvy, které nám umožní podobně jednoduché grafy snadno a stejně názorně popisovat slovy místo obrázků. \square

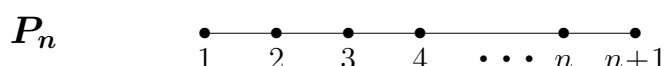
Běžné typy grafů

Pro snadnější vyjadřování je zvykem některé běžné typy grafů nazývat popisnými jmény. Jde čistě o věc konvence a autoři se mohou v některých názvech lišit (i přicházet s novými názvy), avšak následujících pět názvů patří k všeobecným základům teorie grafů.

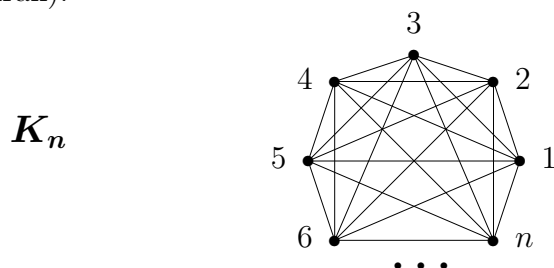
Kružnice délky n má $n \geq 3$ různých vrcholů spojených „do jednoho cyklu“ n hranami:



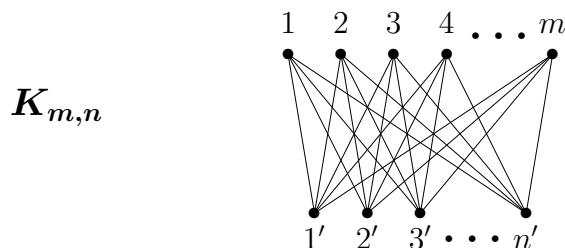
Cesta délky $n \geq 0$ má $n + 1$ různých vrcholů spojených „za sebou“ n hranami:



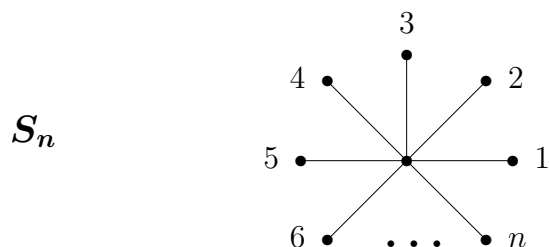
Úplný graf na $n \geq 1$ vrcholech má n různých vrcholů spojených po všech dvojicích (tj. celkem $\binom{n}{2}$ hran):



Úplný bipartitní graf na $m \geq 1$ a $n \geq 1$ vrcholech má $m+n$ vrcholů ve dvou skupinách (partitách), přičemž hranami jsou spojeny všechny $m \cdot n$ dvojice z různých skupin:



Hvězda s $n \geq 1$ rameny je zvláštní název pro úplný bipartitní graf $K_{1,n}$:



Definice: Formálně necht' *kružnice* délky $n \geq 3$ je graf C_n , kde $V(C_n) = \{1, 2, \dots, n\}$ a $E(C_n) = \{\{i, i+1\} : 1 \leq i \leq n\} \cup \{\{n, 1\}\}$. Necht' *cesta* délky $n \geq 0$ je graf P_n , kde $V(P_n) = \{1, 2, \dots, n+1\}$ a $E(P_n) = \{\{i, i+1\} : 1 \leq i \leq n+1\}$. Necht' *úplný graf* na $n \geq 1$ vrcholech je K_n , kde $V(K_n) = \{1, 2, \dots, n\}$ a $E(K_n) = \{\{i, j\} : 1 \leq i < j \leq n\}$. Necht'

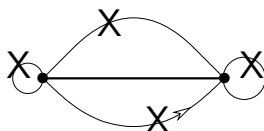
úplný bipartitní graf na $m \geq 1$ a $n \geq 1$ vrcholech je $K_{m,n}$, kde $V(K_{m,n}) = \{1, 2, \dots, m, m+1, \dots, m+n\}$ a $E(K_{m,n}) = \{\{i, j\} : 1 \leq i \leq m, m+1 \leq j \leq m+n\}$.

Příklad 7.5. *Zodpovězte si sami následující snadné otázky:*

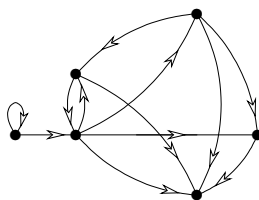
- * Pro jakou hodnotu n je úplný graf K_n zároveň cestou?
- * Pro jakou hodnotu n je úplný graf K_n zároveň kružnicí?
- * Pro jaké hodnoty $m, n > 0$ je úplný bipartitní graf $K_{m,n}$ zároveň kružnicí?
- * Kolik hran musíte přidat do kružnice délky 6, aby vznikl úplný graf na 6 vrcholech?
- * Pro jaké hodnoty $m, n > 0$ úplný bipartitní graf $K_{m,n}$ neobsahuje žádnou kružnici? \square

Zmínka o zobecněných grafech

Komentář: Všimněme si, že v definici grafu (Def. 7.1) vůbec neuvažujeme možnosti vícenásobných hran (mezi stejnou dvojicí vrcholů) a tzv. „smyček“ (hrana se stejným jedním koncem)—takovému zobecnění by se říkalo *multigraf*; ani zatím nepřisuzujeme hranám žádný směr.



V Lekci 9 si však ještě zavedeme *orientované grafy*, které každé hraně přiřazují jistý směr. Orientované grafy budou mít množinu *orientovaných hran* $A \subseteq V(G) \times V(G)$ a zobrazíme je třeba takto...



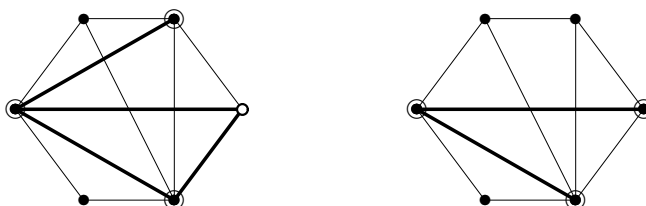
7.2 Podgrafy a Isomorfismus

Dva základní nástroje pro práci s grafy jsou následující; možnost popisovat „část grafu“ (podobně jako podmnožinu množiny, avšak je nutno se vyvarovat nekorektních situací) a poznávat „stejnost“ dvou grafů.

Definice: *Podgrafem* grafu G rozumíme libovolný graf H na podmnožině vrcholů $V(H) \subseteq V(G)$, který má za hrany libovolnou podmnožinu hran grafu G majících oba vrcholy ve $V(H)$.

Píšeme $H \subseteq G$, tj. stejně jako množinová inkluze (ale význam je trochu jiný).

Komentář: Na následujícím obrázku vidíme zvýrazněné podmnožiny vrcholů hran. Proč se vlevo nejedná o podgraf? Obrázek vpravo už podgrafem je.

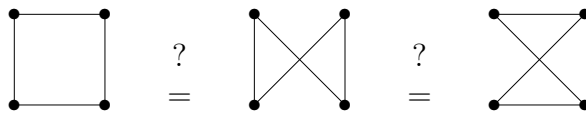


Definice: *Indukovaným podgrafem* je podgraf $H \subseteq G$ takový, který obsahuje všechny hrany grafu G mezi dvojicemi vrcholů z $V(H)$.

„Stejnost“ grafů

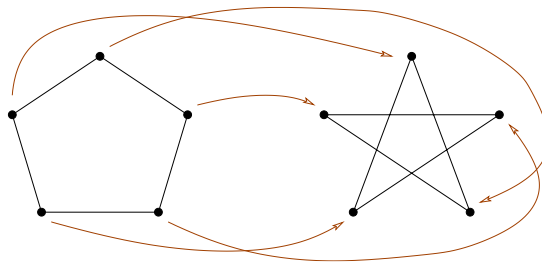
Pozorný čtenář si možná již při čtení předchozího oddílu položil otázku: Co když vezmeme jeden graf (třeba kružnici délky 4) a nakreslíme nebo zapíšeme jej jednou tak, podruhé zase jinak – je to stále tentýž graf nebo ne? Viz obrázky dole.

Přísně formálně řečeno, každé další nakreslení jistého grafu, třeba této kružnice C_4 , je jiným grafem, ale přitom bychom rádi řekli, že různá nakreslení téhož grafu jsou „stále stejná“. Pro tuto stejnost grafů se vžil pojem *isomorfní grafy*.



Definice 7.6. *Isomorfismus* \simeq grafů G a H

je bijektivní (*vzájemně jednoznačné*) zobrazení $f : V(G) \rightarrow V(H)$, pro které platí, že každá dvojice vrcholů $u, v \in V(G)$ je spojena hranou v G právě tehdy, když je dvojice $f(u), f(v)$ spojena hranou v H .

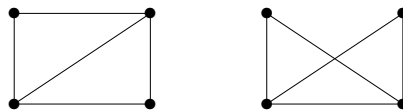


Grafy G a H jsou *isomorfní*, pokud mezi nimi existuje isomorfismus. Píšeme $G \simeq H$.

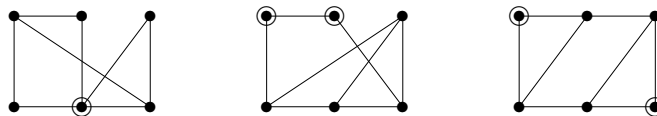
Fakt: Mějme isomorfismus f grafů G a H . Pak platí následující

- * G a H mají stejný počet hran,
- * f zobrazuje na sebe vrcholy stejných stupňů, tj. $d_G(v) = d_H(f(v))$.

Komentář:



U výše zakreslených dvou grafů objevíme isomorfismus velmi snadno – podíváme se, jak si odpovídají vrcholy stejných stupňů.



Naopak v této trojici grafů (se stejnými počty vrcholů i hran) žádné dva nejsou isomorfní. Proč? Ten vlevo má vrchol stupně 4, čímž se od obou zbylých liší. Prostřední graf pak má

jediné dva vrcholy stupně 2 spojené hranou, kdežto v pravém takové dva vrcholy spojené nejsou (isomorfismus by je však i s hranou musel zachovat).

Příklad 7.7. Jsou následující dva grafy isomorfní?



Pokud mezi nakreslenými dvěma grafy hledáme isomorfismus, nejprve se podíváme, zda mají stejný počet vrcholů a hran. Mají. Pak se podíváme na stupně vrcholů a zjistíme, že oba mají stejnou posloupnost stupňů 2, 2, 2, 2, 3, 3. Takže ani takto jsme mezi nimi nerozlišili a mohou (nemusejí!) být isomorfní. Dále tedy nezbývá, než zkusit všechny přípustné možnosti zobrazení isomorfismu z levého grafu do pravého.

Na levém grafu si pro ulehčení všimněme, že oba vrcholy stupně tři jsou si symetrické, proto si bez újmy na obecnosti můžeme vybrat, že nejlevější vrchol prvního grafu, označme jej 1, se zobrazí na nejlevější vrchol 1' v druhém grafu (taky stupně tři). Očíslujme zbylé vrcholy prvního grafu 2, ..., 6 v kladném smyslu, jak je ukázáno na následujícím obrázku. Druhý vrchol stupně tři, označený 4, se musí zobrazit na analogický vrchol druhého grafu (pravý spodní). Pak je již jasně vidět, že další sousedé 2, 6 vrcholu 1 se zobrazí na analogické sousedy 2', 6' vrcholu 1' v druhém grafu, a stejně je to i se zbylými vrcholy 3, 5. Výsledný isomorfismus vypadá v odpovídajícím značení vrcholů takto:



□

Abychom mohli s isomorfismem grafů přirozeně pracovat, je potřeba následující fakt:

Věta 7.8. Relace „být isomorfní“ \simeq na třídě všech grafů je ekvivalencí.

Důkaz. Relace \simeq je reflexivní, protože graf je isomorfní sám sobě identickým zobrazením. Relace je také symetrická, neboť bijektivní zobrazení lze jednoznačně obrátit. Tranzitivita \simeq se snadno dokáže skládáním zobrazení–isomorfismů. □

Důsledkem je, že všechny možné grafy se rozpadnou na *třídy isomorfismu*. V praxi pak, pokud mluvíme o *grafu*, myslíme tím obvykle jeho celou třídu isomorfismu, tj. nezáleží nám na konkrétní prezentaci grafu.

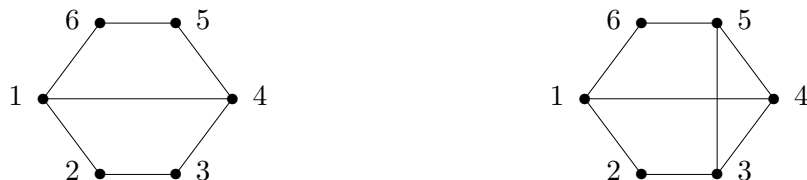
Komentář: Je uvedený přístup, tj. zaměňování konkrétního grafu za celou jeho třídu isomorfismu, v matematice neobvyklý? Ne, například už v geometrii jste říkali „čtverec o straně 2“ či „jednotkový kruh“ a podobně, aniž jste měli na mysli konkrétní obrázek, nýbrž celou třídu všech těchto shodných objektů.

Další grafové pojmy

Definice: Mějme libovolný graf G .

- * Podgrafu $H \subseteq G$, který je isomorfní nějaké kružnici, říkáme *kružnice* v G .
- * Speciálně říkáme *trojúhelník* kružnici délky 3.
- * Podgrafu $H \subseteq G$, který je isomorfní nějaké cestě, říkáme *cesta* v G .
- * Podgrafu $H \subseteq G$, který je isomorfní nějakému úplnému grafu, říkáme *klika* v G . (Někdy se za kliku považuje pouze takový úplný podgraf, který je maximální vzhledem k uspořádání inkluzí.)
- * Podmnožině vrcholů $X \subseteq V(G)$, mezi kterými nevedou v G vůbec žádné hrany, říkáme *nezávislá množina* X v G .
- * Indukovanému podgrafu $H \subseteq G$, který je isomorfní nějaké kružnici, říkáme *indukovaná kružnice* v G .

Komentář: Uvažujme následující ukázky grafů:



První z ukázaných grafů například neobsahuje žádný trojúhelník, ale obsahuje kružnici délky 4, dokonce indukovanou. Druhý graf trojúhelník obsahuje a kružnici délky 4 taktéž. První graf obsahuje cestu délky 4 na vrcholech 1, 2, 3, 4, 5, ale ta není indukovaná. Indukovaná cesta délky 4 v něm je třeba 2, 3, 4, 5, 6. Druhý graf tyto cesty také obsahuje, ale naopak žádná z nich není indukovaná. První graf má největší kliku velikosti 2 – jedinou hranu, kdežto druhý graf má větší kliku na vrcholech 3, 4, 5. Největší nezávislá množina u obou grafů má 3 vrcholy 2, 4, 6.

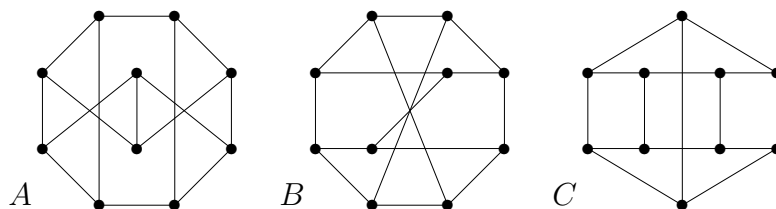
Fakt: Mějme isomorfismus f grafů G a H . Pokud G obsahuje podgraf F , pak H také musí obsahovat podgraf isomorfní F . Obecněji lze tvrdit, že počet podgrafů v grafu G isomorfních zvolenému F je vždy roven takovému počtu v grafu H .

Příklad 7.9. Jsou následující dva grafy isomorfní?



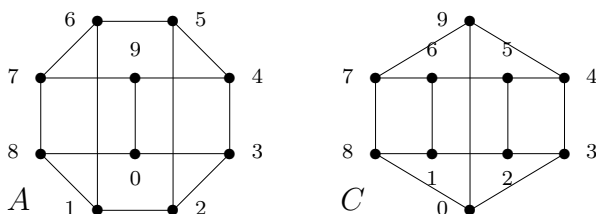
Postupovat budeme jako v Příkladě 7.7, nejprve ověříme, že oba grafy mají stejně mnoho vrcholů i stejnou posloupnost stupňů 2, 2, 2, 2, 3, 3. Pokud se však budeme snažit najít mezi nimi isomorfismus, něco stále nebude vycházet, že? Co nám tedy v nalezení isomorfismu brání? Podívejme se, že v druhém grafu oba vrcholy stupně tři mají svého společného souseda, tvoří s ním trojúhelník. V prvním grafu tomu tak není, první graf dokonce nemá žádný trojúhelník. Proto zadané dva grafy nejsou isomorfní. \square

Příklad 7.10. Najděte všechny isomorfní dvojice grafů v následujících obrázcích tří 10-vrcholových grafů. Isomorfní dvojice odpovídajícím způsobem očísľujte, u neisomorfních toto zdůvodněte.



Postupujme systematicky: Všechny tři grafy mají po 10 vrcholech a všechny vrcholy stupňů 3. Takto jsme je tedy nijak nerozlišili. Podívejme se třeba na trojúhelníky v grafech – opět si nepomůžeme, neboť žádný z nich trojúhelníky neobsahuje. Co se tedy podívat na obsažené kružnice délky 4? Graf C jich jasně obsahuje pět, graf A po chvíli zkoumání také, ale v grafu B najdeme i při vší snaze jen tři kružnice délky 4. (Obdobného rozdílu si můžeme všimnout, pokud se zaměříme na kružnice C_5 , zkuste si to sami.)

Takže co z dosavadního zkoumání plyne? Graf B nemůže být isomorfní žádnému z A, C . Nyní tedy zbývá najít (očekávaný) isomorfismus mezi grafy A a C . To se nám skutečně podaří poměrně snadno - stačí „prohozením“ prostředních dvou vrcholů u grafu A získat lepší obrázek



a odpovídající bijekce je na pohled zřejmá. Pro větší názornost jsme bijekci potvrzující isomorfismus zakreslených grafů explicitně vyznačili odpovídajícím číslováním vrcholů. \square

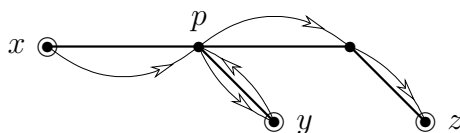
Poznámka: Výše uvedené příklady nám ukazují některé cesty, jak poznat (tj. najít nebo vyloučit) isomorfismus dvou grafů. Ty však ne vždy musí fungovat. Čtenář se může ptát, kde tedy najde nějaký univerzální postup pro nalezení isomorfismu? Bohužel vás musíme zklamat, žádný rozumný univerzální postup není znám a zatím platí, že jediná vždy fungující cesta pro nalezení či nenalezení isomorfismu mezi dvěma grafy je ve stylu *vyzkoušejte všechny možnosti* bijekcí mezi vrcholy těchto grafů. (Těch je, jak známo, až $n!$.) Avšak heuristické algoritmické přístupy pracující se stupni vrcholů samotných a jejich sousedů a případně s malými podgrafy jsou v praxi velmi efektivní pro rozhodování isomorfismu.

7.3 Souvislost grafů, komponenty

Důležitou globální vlastností grafů je *souvislost*, tedy možnost se v nich pohybovat odkudkoliv kamkoliv podél jeho hran, neboli po cestách v grafu. Tuto vlastnost si nyní upřesníme.

Tvrzení 7.11. Mějme relaci \sim na množině vrcholů $V(G)$ libovolného grafu G takovou, že pro dva vrcholy $x \sim y$ právě když existuje v G cesta začínající v x a končící v y . Pak \sim je relací ekvivalence.

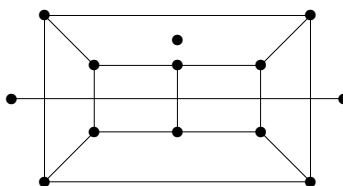
Důkaz. Relace \sim je reflexivní, neboť každý vrchol je spojený sám se sebou cestou délky 0. Symetrická je také, protože cestu z x do y snadno v neorientovaném grafu obrátíme na cestu z y do x . Důkaz tranzitivity však není takto triviální—pokud vezmeme cestu z x do y a cestu z y do z , tak se tyto dvě cesty mohou protínat i jinde než v y a nelze je prostě „navázat“ na sebe.



Pro důkaz tranzitivity si označme P cestu z x do y a Q cestu z y do z . Pokud označíme $P' \subseteq P$ tu část první cesty z x do prvního vrcholu p v průniku s Q (tj. $p \in V(P) \cap V(Q)$) a označíme $Q' \subseteq Q$ zbytek druhé cesty od p do z , tak $P' \cup Q'$ vždy je cestou z x do z . \square

Definice 7.12. Komponentami souvislosti grafu G nazveme třídy ekvivalence výše popsané (Tvrz. 7.11) relace \sim na $V(G)$. Jinak se také *komponentami souvislosti* myslí podgrafy indukované na těchto třídách ekvivalence.

Komentář: Podívejte se, kolik komponent souvislosti má tento graf:

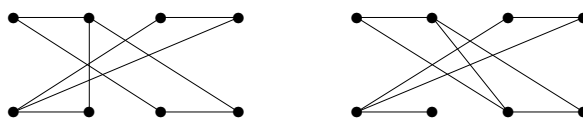


Vidíte v obrázku všechny tři komponenty? Jedna z nich je izolovaným vrcholem, druhá hranou (tj. grafem isomorfním K_2) a třetí je to zbývající.

Definice 7.13. Graf G je souvislý pokud je G tvořený nejvýše jednou komponentou souvislosti, tj. pokud každé dva vrcholy G jsou spojené cestou.

Poznámka: Prázdný graf je souvislý a má 0 komponent.

Komentář: Který z těchto dvou grafů je souvislý?



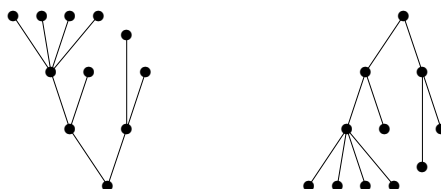
Příklad 7.14. Dokažme si, že každý souvislý jednoduchý 2-regulární graf G je kružnicí (tj. isomorfní některé kružnici C_n z Oddílu 7.1).

Nechť e je hranou mezi vrcholy u, v grafu G a vezmeme graf $G' = G - e$ vzniklý odebráním hrany e . Pokud by u a v náležely v G' různým komponentám souvislosti, tyto komponenty by každá měla lichý součet stupňů, což nelze podle Věty 7.3. Proto u a v jsou spojeny cestou v G' , nechť vrcholy této cesty jsou po řadě značeny $u_1 = u, u_2, \dots, u_k = v$. Toto značení nám nyní udává isomorfismus zobrazující vrchol u_i na vrchol i z definice kružnice C_k . Nyní už zbývá jen drobnost; dokázat, že jiné vrcholy než u_1, \dots, u_k v grafu G nejsou. Pokud bychom měli další vrchol x , pak x je spojen cestou Q do u a první vrchol z $V(Q) \cap \{u_1, \dots, u_k\}$ by měl stupeň větší než 2, spor. \square

7.4 Stromy – grafy bez kružnic

Podrobnější studium některých užitečných aspektů grafů začneme u toho nejjednoduššího typu grafu – na *stromech*, jež jsou mimo jiné základem mnoha datových typů používaných v informatice.

Komentář:



Charakteristickými znaky stromů je absence kružnic a souvislost...

Definice 7.15. *Strom* je jednoduchý souvislý graf T bez kružnic.

Komentář: Obecněji *les* je jednoduchý graf bez kružnic (nemusí být souvislý). Komponenty souvislosti lesa jsou stromy. Jeden vrchol bez hran a prázdný graf jsou také stromy.

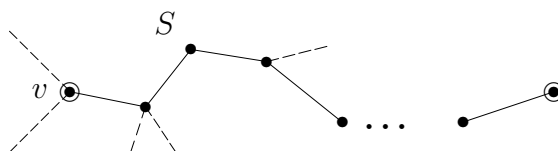
Grafy bez kružnic také obecně nazýváme *acyklické*.

Vlastnosti stromů

Přehled základních vlastností stromů je pro nás zároveň příležitostí si ukázat několik nových hezkých matematických důkazů a naučit se správně zdůvodňovat v oblasti grafů.

Tvrzení 7.16. *Strom s více než jedním vrcholem obsahuje vrchol stupně 1.*

Důkaz: Souvislý graf s více než jedním vrcholem nemůže mít vrchol stupně 0. Proto vezmeme libovolný strom T a v něm libovolný vrchol v . Sestrojíme nyní co nejdélší cestu S v T začínající ve v : S začne libovolnou hranou vycházející z v ; v každém dalším vrcholu u , do kterého se dostaneme a má stupeň větší než 1, lze pak pokračovat cestu S další novou hranou.



Pokud by se v S poprvé zopakoval některý vrchol, získali bychom *kružnici*, což ve stromě nelze. Proto cesta S musí jednou skončit v nějakém vrcholu stupně 1 v T . \square

Komentář: Zamyslete se, proč v každém stromě s více než jedním vrcholem jsou alespoň dva vrcholy stupně 1 (odpověď je skrytá už v předchozím důkaze). Zároveň si odpovězte, jestli lze tvrdit, že každý strom s více než jedním vrcholem obsahuje tři vrcholy stupně 1.

Věta 7.17. *Strom na n vrcholech má přesně $n - 1$ hran pro $n \geq 1$.*

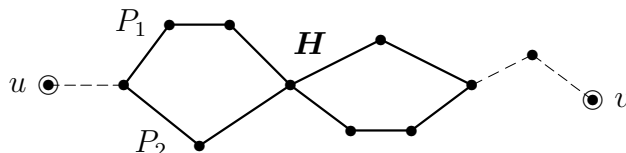
Důkaz: Toto tvrzení dokážeme indukcí podle n . Strom s jedním vrcholem má $n - 1 = 0$ hran. Nechť T je strom na $n > 1$ vrcholech. Podle Tvrzení 7.16 má T vrchol v stupně 1. Označme $T' = T - v$ graf vzniklý z T odebráním vrcholu v . Pak T' je také souvislý bez kružnic, tudíž strom na $n - 1$ vrcholech. Dle indukčního předpokladu T' má $n - 1 - 1$ hran, a proto T má $n - 1 - 1 + 1 = n - 1$ hran. \square

Příklad 7.18. *Les G má 2015 vrcholů a 4 souvislé komponenty. Kolik má hran?*

Pokud počty vrcholů jednotlivých komponent lesa G po řadě označíme n_1, n_2, n_3, n_4 , tak máme $n_1 + n_2 + n_3 + n_4 = 2015$. Každá komponenta G je strom podle definice lesa. Zároveň podle Věty 7.17 má i -tá komponenta přesně $n_i - 1$ hran. V součtu má G celkem $(n_1 - 1) + (n_2 - 1) + (n_3 - 1) + (n_4 - 1) = 2015 - 4 = 2011$ hran. \square

Věta 7.19. *Mezi každými dvěma vrcholy stromu vede právě jediná cesta.*

Důkaz:



Jelikož strom T je souvislý dle definice, mezi libovolnými dvěma vrcholy u, v vede nějaká cesta. Pokud by existovaly dvě různé cesty P_1, P_2 mezi u, v , tak bychom vzali jejich symetrický rozdíl, podgraf $H = P_1 \Delta P_2$ s neprázdnou množinou hran, kde H zřejmě má všechny stupně sudé. Na druhou stranu se však podgraf stromu musí opět skládat z komponent stromů, a tudíž obsahovat vrchol stupně 1 podle Tvrzení 7.16, což je spor. Proto cesta mezi u a v existuje jen jedna. \square

Důsledek 7.20. *Přidáním jedné nové hrany do stromu vznikne právě jedna kružnice.*

Důkaz: Nechť mezi vrcholy u, v ve stromu T není hrana. Přidáním hrany $e = uv$ vznikne právě jedna kružnice z e a jediné cesty mezi u, v v T podle Věty 7.19. \square

Alternativní charakterizace stromů

Z předchozích tvrzení vyplývá následující alternativní charakterizace stromů, která ukazuje důležitost jich samotných i jako tzv. *koster* obecných grafů (viz Oddíl 8.4).

Na dané množině vrcholů je (vzhledem k inkluzi množin hran) *strom*

- *minimální souvislý graf* (plyne z Věty 7.19)
- a zároveň *maximální acyklický graf* (plyne z Důsledku 7.20).

Jen tak mimochodem, kolik dokážete nalézt neisomorfních stromů na 4 nebo 5 vrcholech? Vidíte, že jich není mnoho? Nakreslete si je všechny.

7.5 Použití a implementace grafů

Závěrem si v našem studijním textu nastíníme některé základní motivace pro zavedení a použití grafů při popisu a řešení problémů například v informatice.

Příklad 7.21. *Ukázky některých problémů „ze života“ popsatelných grafy. Podotýkáme, že tyto ukázky jsou často velmi zjednodušené (pro jejich lepší přístupnost širokému okruhu čtenářů), ale to neubírá jejich motivačnímu potenciálu.*

- Vyjádření mezilidských vztahů – „mají se rádi“, „kamarádí se“, „nesnesou jeden druhého“, apod:

Zde jednotlivé osoby tvoří vrcholy grafu a vztahy jsou hranami (často neorientované, ale i orientace je přípustná). Všimněme si coby zajímavosti, jak tento model přirozeně preferuje „párový“ pohled na vztahy – hrany přece spojují jen dvojice vrcholů. Třebaže například vztah „kamarádí se“ může být obecně platný pro větší skupinky lidí než dvojice, stejně se obvykle vyjadřuje klikou v grafu (každí dva v našem družstvu jsou dobří kamarádi. . .). Tato obecně pojímaná tendence vyjadřovat i *složitější vztahy těmi párovými* je vodou na mlýn použití teorie grafů jako téměř univerzálního vyjadřovacího prostředku v podobných případech.

Na druhou stranu i teorie grafů disponuje pojmem tzv. *hypergrafu* umožňujícího použití hran libovolné arity (počtu koncových vrcholů), ale rozsah výskytu hypergrafů v teorii i aplikacích je oproti grafům vskutku zanedbatelný.

- Vyjádření závislostí mezi objekty nebo procesy:

Představme si situace, ve kterých jednotlivé entity (modelované jako vrcholy) závisí na výstupech jiných entit a naopak poskytují výstupy další entitám. Typickým příkladem mohou být závislosti jednotlivých kroků výrobního nebo rozhodovacího procesu. Ty pak vedou k definici *orientovaného grafu* na dané množině vrcholů/entit, tj. použití hran „se šipkami“. Všimněme si, že závislosti často bývají časového charakteru (přičemž směr závislosti je implicitně jasný) a pak je nezbytnou doplňkovou podmínkou *vyločení výskytu orientovaných cyklů* v modelovém grafu. Na druhou stranu existují i situace, kdy cyklické závislosti jsou dovoleny a mají svůj význam.

Pro ještě jednu ukázkou závislostí z běžného života informatika se podíváme na správu balíčků softwaru například v Linuxových distribucích. V tom případě jsou jednotlivé balíčky vrcholy grafu, jejich vyžadované závislosti popisují odchozí hrany a jejich poskytované vlastnosti jsou příchozími hranami grafu závislostí. Korektní instalace zvoleného balíčku pak řeší problém zahrnutí všech dalších vrcholů „dosažitelných“ ze zvoleného. Vše je navíc komplikováno správou verzí balíčků, ale to už je mimo rámec našeho úvodního slova.

- Modelování technických či dopravních sítí grafy:

V takových případech bývají vrcholy grafu jednotlivá technická zařízení jako třeba rozvodny, routery, křižovatky a podobně, kdežto hrany jsou tvořeny spojnicemi/vedením mezi vrcholy. Často se zde setkáváme s orientovanými grafy a obecně *multigrafy*. K této problematice se blíže vyjadřuje Lekce 9.

- Vizualizace vztahů a závislostí pro lidského pozorovatele:

Nejen při řešení cvičných příkladů v naší učebnici, ale i v mnoha reálných aplikacích využívajících grafy jako modely, je velmi potřebné tyto grafy *vizualizovat* (tj. hezky nakreslit) pro lidského pozorovatele. Jedná se obecně o poměrně obtížný úkol, který přesahuje hranice našeho textu. □

Zpracování grafu počítačem

Mějme jednoduchý graf G na n vrcholech a značme vrcholy jednoduše čísly $V(G) = \{0, 1, \dots, n-1\}$. Pro počítačovou implementaci grafu G se nabízejí dva základní způsoby, které budeme implicitně využívat i v některých algoritmech následujících lekcí.

- Implementace *maticí sousednosti* \mathbf{G} , tj. dvourozměrným polem, ve kterém hodnota $\mathbf{G}[i, j] = 1$ reprezentuje hranu mezi vrcholy i a j .
- Implementace *výčtem sousedů*, tj. systémem množin, který každému vrcholu i grafu udává seznam jeho sousedů. Všimněte si, že pořadí sousedů může být jakékoliv.

Poznámka: Dávejte si pozor na symetrii hran v implementaci! To znamená, že pokud uložíte hranu $\mathbf{G}[i, j] = 1$, tak musíte zároveň uložit i hranu $\mathbf{G}[j, i] = 1$, jinak se dočkáte v algoritmech nepřijemných překvapení. Totéž se týká i seznamů sousedů.

Komentář: Implementace maticí sousednosti je hezká svou jednoduchostí. Druhá možnost se však mnohem lépe hodí pro grafy s malým počtem hran, což nastává ve většině praktických aplikací. (Navíc je implementace výčtem sousedů vhodná i pro multigrafy.) Pochopitelně je pak vhodnější místo polí používat různé spojové seznamy. Ke grafům lze do zvláštních složek přidat také *ohodnocení vrcholů a hran* libovolnými čísly či značkami. . .

Rozšiřující studium

Grafy můžeme v informatice potkat doslova na každém kroku, mimo jiné hojně už v základních kurzech algoritmicke. Nejen že grafy (především stromy) jsou základem mnoha programátorských datových struktur, ale představují i vhodný model pro spoustu praktických problémů. Oněch pár lekcí teorie grafů v našem učebním textu je jen lehkým úvodem, přičemž na FI MU lze pokračovat v jejím cíleném studiu v předmětech MA010, MA015.

Rozsáhlý matematický úvod do teorie grafů je zahrnut ve skvělé knize *Kapitoly z diskrétní matematiky autorů Jiřího Matouška a Jaroslava Nešetřila*. Vřele ji doporučujeme jako doplňkový studijní zdroj všem, kteří chtějí lépe pochopit grafy z jejich matematické stránky.

8 Procházení grafu a odvozené úlohy

Úvod

Na rozdíl od předchozí lekce, která se zjednodušeně řečeno zabývala grafy z pohledu matematika, tedy podáním definice a obrázků, nyní se hlouběji podíváme na grafy z algoritmické či programátorské perspektivy. Proto se v první řadě podíváme na obecné schéma procházení grafu, které je základem mnoha užitečných algoritmů na grafech. Poté se hlouběji zaměříme na dvě specifické grafové úlohy – hledání nejkratší cesty a minimální kostry, hojně se vyskytující v mnoha praktických obměnách. Na základě obecného schématu si uvedeme jejich jednoduché a velmi efektivní algoritmy.

Cíle

Zjednodušeně podáme obecné schéma algoritmu procházení grafu. Definujeme poté dvě konkrétní grafové úlohy, hledání nejkratší cesty mezi dvěma vrcholy a hledání nejmenší kostry, obojí v grafu s hranami ohodnocenými reálnými délkami. Na základě předchozího pak vysvětlíme Dijkstrův algoritmus pro nejkratší cesty a Kruskalův a Jarníkův algoritmus pro minimální kostru.

8.1 Jak obecně projít souvislý graf

S úlohou procházení grafu se svým způsobem setkávají už děti, když hledají cestu z bludiště (jejich naivní postup se dá přirovnat k procházení „do hloubky“). Celý problém má ale mnohem širší záběr a s vhodnou implementací dodatečných lokálních funkcí lze pouhým prohledáním grafu podat odpověď na jiné zajímavé otázky, jako najít nejkratší cestu, minimální kostru, komponenty vyšší souvislosti, apod. Navíc se procházení grafu vyskytuje jako podúloha v jiných algoritmech.

Metoda 8.1. *Schéma algoritmu pro procházení grafem*

Pro vytvoření co nejobecnějšího schématu si pomůžeme následujícími datovými stavy a pomocnou strukturou:

- *Vrchol grafu:* má stavy ...
 - * *iniciační* – dostane na začátku,
 - * *nalezený* – implicitní stav poté, co jsme jej přes některou hranu našli (a odložili ke zpracování později),
 - * *zpracovaný* – poté, co jsme už probrali všechny hrany z něj vycházející,
 - * (případně ještě stav „post-zpracovaný“, po dokončení všech jeho následníků).
- *Úschovna:* je pomocná datová struktura (množina s dodatečnými atributy),
 - * udržuje odložené, tj. nalezené a ještě nezpracované vrcholy, spolu s dodatečnou specifickou informací.
- Způsob, kterým se vybírají vrcholy z úschovny ke zpracování, určuje variantu algoritmu procházení grafu.
- V prohledávaných vrcholech a hranách se *volitelně* provádějí *dodatečné programové akce pro prohledání a zpracování* našeho grafu.

Samotný algoritmus pak popíšeme v následujících obecných bodech. (V této souvislosti dodáváme, že podrobněji se budeme způsobu formálního matematického zápisu algoritmů věnovat v Lekci 10, ale zde si vystačíme s běžným jazykem.)

Algoritmus 8.2. *Generické procházení souvislé komponenty G grafu*

- *Vstup:* Souvislý graf G , daný seznamem vrcholů a seznamy vycházejících hran z každého vrcholu, plus případné ohodnocení vrcholů a hran.
- Vybereme libovolný počátek prohledávání $u \in V(G)$; úschovna $U \leftarrow \{u\}$.
- Dokud $U \neq \emptyset$, opakujeme:
 - * Zvolíme libovolně $v \in U$; odebereme $U \leftarrow U \setminus \{v\}$. (!)
 - * Pokud $stav(v) = zpracovaný$, jdeme zpět na start cyklu. (*)
 - * Případně provedeme libovolnou akci $ZPRACUJ(v)$.
 - * Pro všechny hrany $f \in E(G)$ vycházející z v provedeme:
 - Nechť w je druhý konec hrany $f = vw$;
 - pokud $stav(w) \neq zpracovaný$, odložíme $U \leftarrow U \cup \{w\}$. (**)
 - * $stav(v) \leftarrow zpracovaný$; na start cyklu.
- Souvislý G je celý prohledaný a zpracovaný.
- *Výstup:* Případné výsledky spočítané během prohledávání.

*Pozor, všimněte se, že v bodě (**)* obecně dochází k násobnému ukládání odložených vrcholů, což v praktické implementaci často obejdeme pouhou změnou „odloženého stavu“. *Doplňkový bod (*)* je nutný jen v případě, že násobně uložené kopie stejného vrcholu v zůstávají v úschovně.

Některé implementace procházení grafu

V Algoritmu 8.2 si dobře všimněte, jak je vlastně proveden krok (!); „zvolíme libovolně $v \in U$ “? Právě tato volba je klíčová pro výslednou podobu projití grafu G :

- *Procházení „do šířky“*, *BFS* – úschovna U je implementovaná jako *fronta*, neboli je voleno $v \in U$ od prvních vrcholů vložených do úschovny.
- *Procházení „do hloubky“*, *DFS* – úschovna U je implementovaná jako *zásobník*, neboli je voleno $v \in U$ od později vložených do úschovny. (Zde je extrémně důležité, že opakované vložení vrcholu v do U jej posune na vršek zásobníku.)

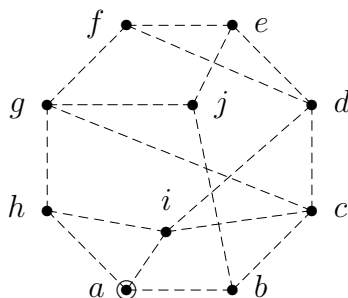
Dále zmíníme i tyto dva konkrétní, staré a dobře známé algoritmy přímo založené na prohledávání grafu:

- *Dijkstrův algoritmus* pro nejkratší cestu – z úschovny vybíráme vždy vrchol nejbližší (dosud určenou celkovou vzdáleností) k počátečnímu u .
- *Jarníkův algoritmus* pro minimální kostru – z úschovny vybíráme vždy vrchol nejbližší (délkou hrany) ke kterémukoliv již zpracovanému vrcholu.

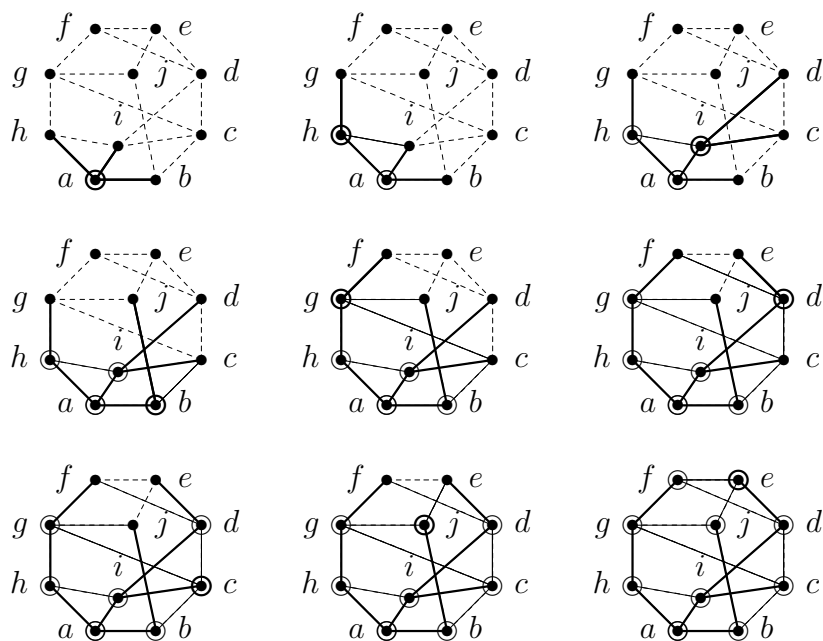
Poznámka: Jarníkův algoritmus se ve světové literatuře se obvykle připisuje Američanu Primovi, který jej však objevil a publikoval až skoro 30 let po Jarníkovi. A co více, původ problému minimální kostry je úzce svázán s Brnem, jak si řekneme za chvíli.

Konkrétní ukázky BFS a DFS

Příklad 8.3. Ukázka průchodu následujícím grafem do šířky z vrcholu a .

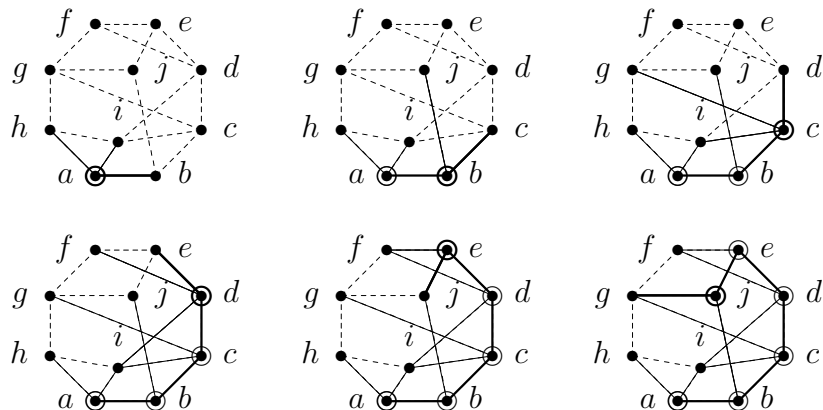


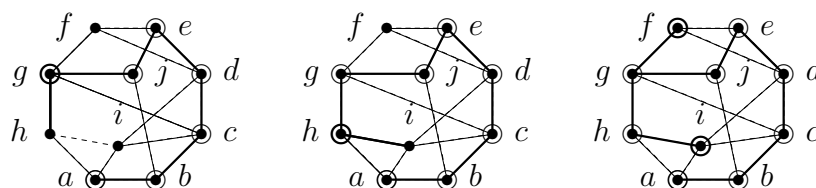
Značení v prohledávaném grafu: barevně aktuálně zpracovávaný vrchol a jeho hrany objevující nové vrcholy, kroužkem a plnou čarou již zpracované vrcholy a hrany, tlustě výsledný strom prohledávání.



□

Příklad 8.4. Ukázka průchodu předchozím grafem do hloubky z vrcholu a .





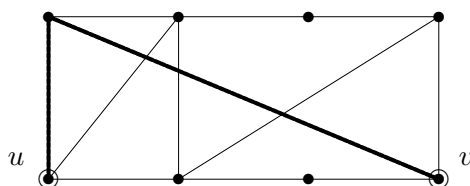
□

8.2 Vzdálenost v grafu

Mimo prohledávání celého souvislého grafu, čímž se zabýval předchozí oddíl, je častou úlohou se dostat z místa na místo nejkratší cestou. Ve zjednodušeném podání považujeme každou hranu grafu za jednotkově dlouhou a definujeme následující.

Definice 8.5. *Vzdálenost* $d_G(u, v)$ dvou vrcholů u, v v grafu G je dána délkou nejkratší cesty mezi u a v v G . Pokud cesta mezi u, v neexistuje, je vzdálenost definována $d_G(u, v) = \infty$.

Komentář: Neformálně řečeno, vzdálenost mezi u, v je rovna *nejmenšímu počtu hran*, které musíme překonat, pokud se chceme dostat z u do v . Speciálně vždy platí $d_G(u, u) = 0$.



Jaká je ve výše zakresleném grafu vzdálenost $d_G(u, v)$? Ano, snadno určíme hodnotu $d_G(u, v) = 2$, viz vyznačená cesta délky 2.

Najdete ale v témže grafu nějakou dvojici vrcholů se vzdáleností 3?

Fakt: V neorientovaném grafu je vzdálenost symetrická, tj. $d_G(u, v) = d_G(v, u)$.

Lema 8.6. *Vzdálenost v grafech splňuje trojúhelníkovou nerovnost:*

$$\forall u, v, w \in V(G) : d_G(u, v) + d_G(v, w) \geq d_G(u, w).$$

Důkaz. Postupujeme podobně jako v důkaze Tvzení 7.11 – pokud máme cesty P, P' mezi u, v a mezi v, w , tak existuje cesta $Q \subseteq P \cup P'$ mezi u, w , jež má zřejmě délku nejvýše $d_G(u, v) + d_G(v, w)$. Skutečná vzdálenost mezi u, w pak už může být jen menší. □

Příklad 8.7. *Proč každý 3-regulární graf na 12 vrcholech musí obsahovat dvojici vrcholů se vzdáleností 3?*

Dokažme požadované tvrzení sporem. Nechť v je libovolný vrchol našeho grafu G . Pak v má tři sousedy a, b, c (tj. ve vzdálenosti 1 od v). Podle definice vzdálenosti je každý vrchol ve vzdálenosti 2 od v sousedem některého z a, b, c . Avšak a má za jednoho souseda v a mimo něj má už jen dva další sousedy. Stejně tak b, c . Tudíž v G existuje nejvýše šest vrcholů mimo v , kteří jsou sousedé jednoho z a, b, c . Celkem tak napočítáme (od v) jen $1 + 3 + 6 = 10$ vrcholů a některý z 12 vrcholů grafu G musí být ve větší vzdálenosti než 2, spor. Zakreslete si sami tuto úvahu obrázkem. □

BFS a zjištění vzdálenosti

Jak nejsnadněji určíme vzdálenost v grafu? Stačí si povšimnout hezkých vlastností procházení grafu do šířky.

Věta 8.8. *Algoritmus procházení grafu do šířky lze použít pro výpočet grafové vzdálenosti z daného vrcholu u .*

Toto je poměrně jednoduchá aplikace, kdy počátečnímu vrcholu u přiřadíme vzdálenost 0, a pak vždy každému dalšímu nalezenému vrcholu v přiřadíme vzdálenost o 1 větší než byla vzdálenost vrcholu, ze kterého byl nalezen.

Důkaz se opírá o následující tvrzení:

- * Nechť u, v, w jsou vrcholy souvislého grafu G takové, že $d_G(u, v) < d_G(u, w)$. Pak při algoritmu procházení grafu G do šířky z vrcholu u je vrchol v nalezen dříve než vrchol w .

V důkaze postupujeme indukcí podle vzdálenosti $d_G(u, v)$: Pro $d_G(u, v) = 0$, tj. $u = v$ je tvrzení jasné – vrchol u jako počátek prohledávání byl nalezen první. Proto nechť $d_G(u, v) = d > 0$ a označme v' souseda vrcholu v bližšího k u , tedy $d_G(u, v') = d - 1$. Obdobně uvažme libovolného souseda w' vrcholu w . Pak

$$d_G(u, w') \geq d_G(u, w) - 1 > d_G(u, v) - 1 = d_G(u, v'),$$

a tudíž vrchol v' byl nalezen v prohledávání do šířky dříve než vrchol w' podle indukčního předpokladu. To znamená, že v' se dostal do fronty úschovny dříve než w' . Proto sousedé v' (mezi nimiž je v , ale ne w neboť $v'w$ není hranou) jsou při pokračujícím prohledávání také nalezeni dříve, než w coby soused w' . \square

8.3 Hledání nejkratší cesty

V dalším textu se již budeme věnovat grafům s „obecně dlouhými“ hranami. Zároveň předesíláme, že přednášená látka stejně dobře může stavět na orientovaných grafech.

Definice: Vážený graf je graf G spolu s ohodnocením w hran reálnými čísly $w : E(G) \rightarrow \mathbb{R}$. *Kladně vážený graf* (G, w) je takový, že $w(e) > 0$ pro všechny hrany e .

Definice 8.9. (vážená vzdálenost) Mějme (kladně) vážený graf (G, w) .

Váženou délkou cesty P je

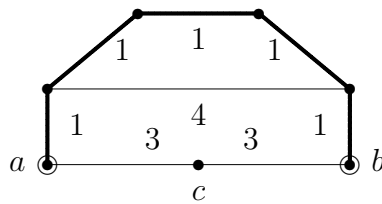
$$d_G^w(P) = \sum_{e \in E(P)} w(e).$$

Váženou vzdáleností v (G, w) mezi dvěma vrcholy u, v pak je

$$d_G^w(u, v) = \min\{d_G^w(P) : P \text{ je cesta s konci } u, v\}.$$

Lema 8.10. *Vážená vzdálenost v nezáporně vážených grafech (i orientovaných grafech) splňuje trojúhelníkovou nerovnost.*

Příklad 8.11. *Podívejme se na následující ohodnocený graf (čísla u hran udávají jejich váhy–déłky.)*



Vzdálenost mezi vrcholy a, c je 3, stejně tak mezi b, c . Co ale mezi a, b ? Je jejich vzdálenost 6? Kdepak, vzdálenost a, b je 5, její cesta vede po „horních“ vrcholech, jak je vyznačeno. Povšimněte si, že tento příklad zároveň ukazuje, že postup prohledávání do šířky není korektní pro hledání vzdáleností ve váženém grafu. \square

Problém nejkratší cesty

Problém nejkratší cesty mezi dvojicí vrcholů prostě hledá váženou vzdálenost vrcholů a příslušnou cestu. Jedná se patrně o nejznámější „grafový“ problém v praktických aplikacích, jenž nalezneme od vyhledávání dopravních spojení, GPS navigací, plánování pohybů robota, až po třeba rozhodovací systémy.

Tento problém se nejčastěji řeší implementací klasického *Dijkstrova algoritmu*, který je vhodnou úpravou procházení grafu do šířky – vybírá vždy vrchol s nejmenší vzdáleností mezi uschovanými vrcholy.

Dijkstrův algoritmus

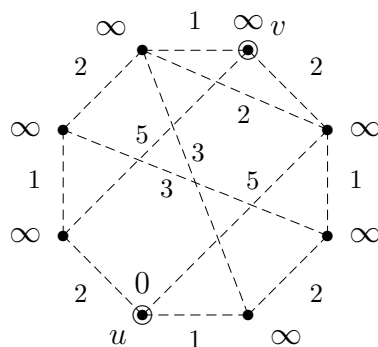
Algoritmus 8.12. Hledání nejkratší cesty mezi u a v v kladně váženém grafu. Tento algoritmus využívá proměnnou hodnotu (pole) $d(x)$ k ukládání nalezených vzdáleností do každého vrcholu x , vedle samotného vrcholu, přičemž tuto hodnotu postupně „vylepšuje“ až do finálního zpracování vrcholu.

- **Vstup:** Souvislý graf G , daný seznamem vrcholů a seznamy vycházejících hran z každého vrcholu, plus váhy w hran. Počáteční vrchol u a koncový v .
- Úschovna $U \leftarrow \{(u, d(u) = 0)\}$.
- Dokud $U \neq \emptyset$, opakujeme:
 - * Zvolíme $(x, d(x)) \in U$ takové, že $d(x)$ je *minimální*. Odebereme $U \leftarrow U \setminus \{(x, d(x))\}$.
 - * Pokud $x = v$, algoritmus může skončit.
 - * Pro všechny hrany $f \in E(G)$ vycházející z x provedeme:
 - Nechť y je druhý konec hrany $f = xy$; a nechť $d'(y) = d(x) + w(f)$ (nová cesta do y přes x).
 - Pokud $(y, d(y)) \notin U$, nebo $(y, d(y)) \in U$ pro $d(y) > d'(y)$, odložíme $U \leftarrow (U \setminus \{(y, d(y))\}) \cup \{(y, d'(y))\}$ (výměna za novou, lepší dočasnou vzdálenost do y).
- **Výstup:** $d(v)$ udává váženou vzdálenost z u do v .

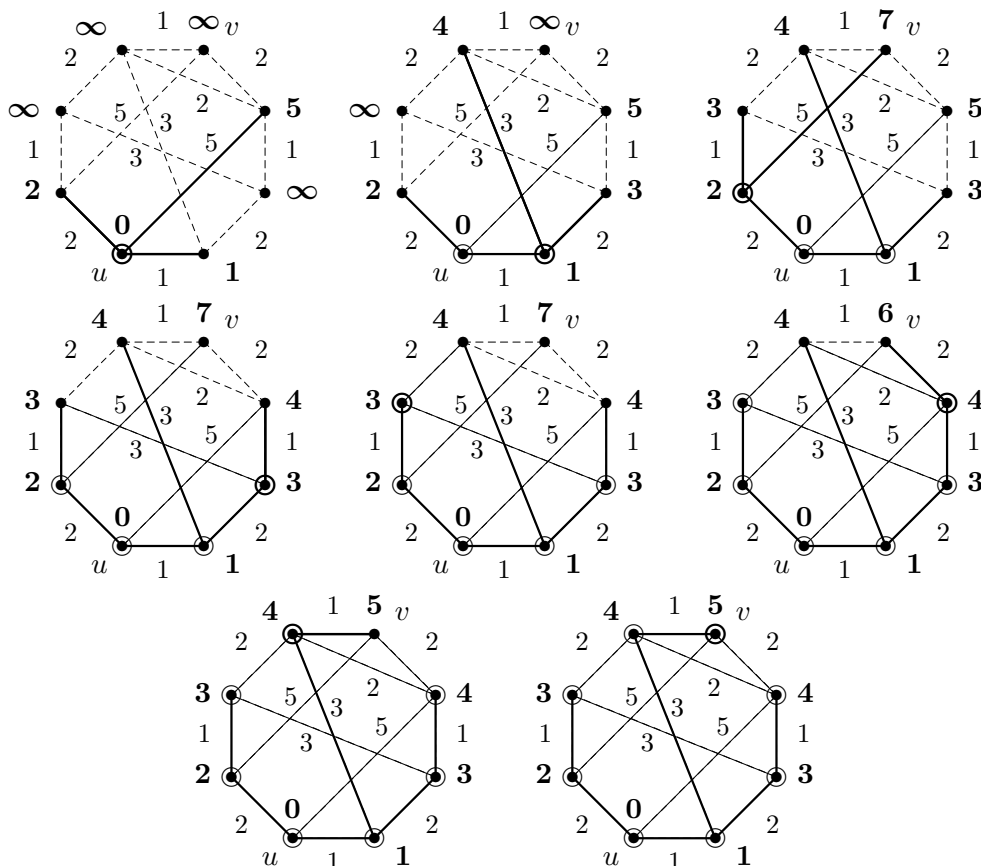
Poznámka: V konkrétní implementaci Dijkstrova algoritmu nejspíše využijeme nějakou datovou strukturu podporující rychlé vkládání prvků a vybírání podle minimální hodnoty klíče – nejlépe tedy *haldu*, a k tomu pole $d[x]$ přímo ukládající průběžně aktualizovanou dočasnou váženou vzdálenost z u do každého x .

Komentář: Klíčem k pochopení činnosti Dijkstrova algoritmu je „uvidět“, že v každé jeho fázi jsou správně nalezeny všechny nejkratší cesty z u vedoucí po zpracovaných vrcholech. Postupem prohledávání grafu se tak jednou dostaneme až k určení správné vzdálenosti cíle v .

Příklad 8.13. Ukázka běhu Dijkstrova Algoritmu 8.12 pro nalezení nejkratší cesty mezi vrcholy u, v v následujícím váženém grafu.



Jednotlivé iterace algoritmu jsou zakresleny níže. Čísla u vrcholů v obrázcích udávají průběžně aktualizovanou dočasnou vzdálenost z u .



□

Důkaz správnosti

Věta 8.14. *Dijkstrův Algoritmus 8.12 pro kladně vážený graf (G, w) vždy správně najde nejkratší cestu mezi danými vrcholy u, v .*

Důkaz vede přes následující zesílené tvrzení indukací:

- V každé iteraci Algoritmu 8.12 hodnota $d(x)$ udává nejkratší vzdálenost z vrcholu u do x při cestě pouze po už zpracovaných vnitřních vrcholech.

V bázi indukce dovoluujeme pouze cesty používající u a x , tj. jen hrany vycházející z u . Ty jsou v první iteraci algoritmu probrány a jejich délky uloženy do U . V každém dalším kroku je vybrán jako vrchol x ke zpracování ten, který má ze všech nezpracovaných vrcholů nejkratší nalezenou vzdálenost od počátku u . V tom okamžiku je $d(x)$ platnou vzdáleností do x , neboť jakákoliv cesta přes jiný nezpracovaný vrchol nemůže být kratší díky nezápornosti vah w . Z toho pak vyplývá, že zpracování vrcholu x správně upraví dočasné vzdálenosti odložené do U . Důkaz indukci je hotov. \square

8.4 Problém minimální kostry

Na závěr této lekce se zaměříme ještě na jednu tradiční úlohu, která se koncepčně podobá problému nejkratší cesty. V tomto případě nebudeme hledat nejkratší spojení mezi dvojicí vrcholů, ale mezi všemi vrcholy najednou – této úloze se říká *minimální kostra* neboli MST („minimum spanning tree“). V návaznosti na Oddíl 7.4 definujeme toto:

Definice: Podgraf $T \subseteq G$ souvislého grafu G se nazývá *kostrou*, pokud

* T je stromem a

* $V(T) = V(G)$, neboli T propojuje všechny vrcholy G .

Váhou (délkou) kostry $T \subseteq G$ váženého souvislého grafu (G, w) rozumíme

$$d_G^w(T) = \sum_{e \in E(T)} w(e).$$

Definice 8.15. *Problém minimální kostry (MST)* ve váženém souv. grafu (G, w) hledá kostru $T \subseteq G$ s nejmenší možnou vahou (přes všechny kostry grafu G).

Komentář: Problém minimální kostry je ve skutečnosti historicky úzce svázán s jižní Moravou a Brnem, konkrétně s elektrifikací jihomoravských vesnic ve dvacátých letech! Právě na základě tohoto praktického optimalizačního problému brněnský matematik Otakar Borůvka jako první v matematické literatuře zformuloval a podal řešení problému minimální kostry v roce 1926. První ne-českou publikací na toto téma je až Kruskalův hladový algoritmus z roku 1956.

Hladové řešení minimální kostry

Následující postup tzv. hladového nalezení minimální kostry pochází od Kruskala.

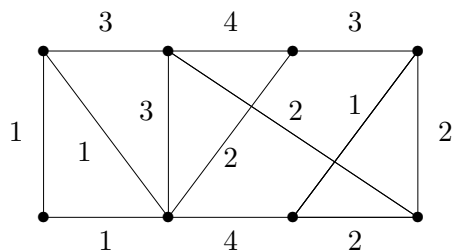
Metoda 8.16. *Hladový postup pro minimální kostru grafu (G, w) .*

Mějme dán souvislý vážený graf G s ohodnocením hran w .

- Seřadíme všechny hrany G jako $E(G) = (e_1, e_2, \dots, e_m)$ tak, že $w(e_1) \leq w(e_2) \leq \dots \leq w(e_m)$.

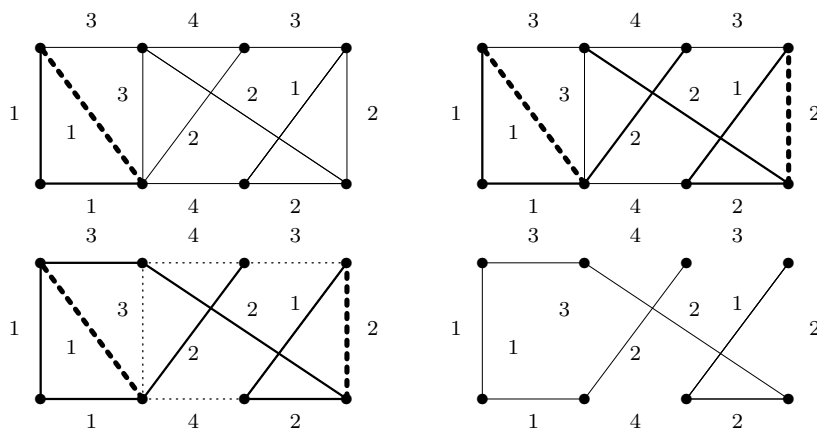
- Inicializujeme prázdnou kostru $T = (V(G), \emptyset)$.
- Po řadě pro $i = 1, 2, \dots, m$ provedeme následující:
 - * Pokud $T + e_i$ nevytváří kružnici, tak $E(T) \leftarrow E(T) \cup \{e_i\}$.
(Neboli pokud e_i spojuje různé komponenty souvislosti dosavadního T .)
- Na konci T obsahuje minimální kostru grafu G (případně jednu z několika takových).

Komentář: Pro ilustraci si ukážeme postup hladového algoritmu pro vyhledání kostry následujícího grafu:



Hrany si nejprve seřadíme podle jejich vah 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4.

V obrázku průběhu algoritmu používáme tlusté čáry pro vybrané hrany kostry a tečkované čáry pro zahozené hrany. Hrany teď postupně přidáváme do kostry / zahazujeme...



Získáme tak minimální kostru velikosti $1 + 2 + 2 + 3 + 1 + 1 + 2 = 12$, která je v tomto případě (náhodou) cestou, na posledním obrázku vpravo.

Poznamenáváme, že při jiném seřazení hran stejné váhy by kostra mohla vyjít jinak, ale vždy bude mít stejnou velikost 12.

Věta 8.17. *Hladový postup korektně spočítá minimální kostru váženého grafu (G, w) .*

Důkaz (náznak): Pro spor předpokládejme, že T_1 je kostra spočítaná Metodou 8.16 a T_2 nějaká minimální kostra, kde $d_G^w(T_2) < d_G^w(T_1)$ a rozdíl $|E(T_1) \Delta E(T_2)|$ je nejmenší. Nechť i je nejmenší index takový, že $e_i \in E(T_1) \Delta E(T_2)$. Pak nutně $e_i \in E(T_1) \setminus E(T_2)$ (proč?), a tudíž $T_2 + e_i$ podle Důsledku 7.20 obsahuje kružnici procházející také hranou e_j pro $j > i$. Potom však $T_3 = (T_2 + e_i) \setminus \{e_j\}$ je další kostrou mající váhu $d_G^w(T_2) + w(e_i) - w(e_j) \geq d_G^w(T_2)$, a proto $w(e_i) = w(e_j)$. Tudíž T_3 je minimální kostra „bližší“ T_1 ve smyslu symetrického rozdílu, což je spor s volbou T_2 . \square

Jarníkův (Primův) algoritmus

Ač koncepčně velmi jednoduchá, má Metoda 8.16 některé problematické implementační detaily, pro které je mnohem častěji používán následující algoritmus (často připisován Američanu Primovi, ale mnohem dříve publikován Vojtěchem Jarníkem v 1930), založený na běžném procházení grafu.

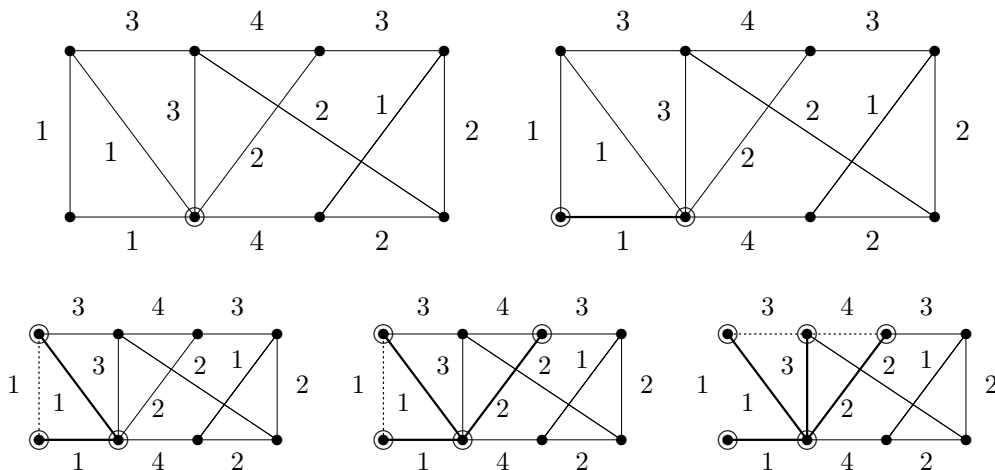
Algoritmus 8.18. Hledání minimální kostry ve váženém grafu (G, w) .

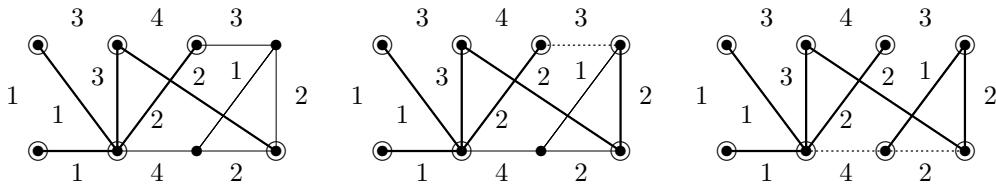
Opět mějme dán souvislý vážený graf G s ohodnocením hran w . Níže uvedená specifická implementace procházení grafu využívá úschovnu rozšířeným způsobem, kdy ukládá i přichozí hranu do vrcholu.

- *Vstup:* Souvislý graf G , daný seznamem vrcholů a seznamy vycházejících hran z každého vrcholu, plus váhy w hran.
- Vybereme libovolný počátek prohledávání $u \in V(G)$.
Úschovna $U \leftarrow \{(u, \emptyset)\}$ a počáteční kostra $T \leftarrow (V(G), \emptyset)$.
- Dokud $U \neq \emptyset$, opakujeme:
 - * Zvolíme $(x, e) \in U$ takové, že $w(e)$ je *minimální* (kde $w(\emptyset) = 0$).
Odebereme $U \leftarrow U \setminus \{(x, e)\}$.
 - * Přidáme $E(T) \leftarrow E(T) \cup \{e\}$ (nová hrana do budoucí kostry).
 - * Pro všechny hrany $f \in E(G)$ vycházející z x provedeme:
 - Nechť y je druhý konec hrany $f = xy$ a nechť f' je takové, že $(y, f') \in U$.
 - Pokud takové f' neexistuje nebo $w(f') > w(f)$, odložíme $U \leftarrow (U \setminus \{(y, f')\}) \cup \{(y, f)\}$ (nalezena kratší hrana f mezi zpracovanými a nezpracovanými vrcholy).
- *Výstup:* T udává výslednou minimální kostru.

Poznámka: V konkrétní implementaci postupujeme stejně jako u Dijkstrova Algoritmu 8.12, nejlépe za použití haldy ukládající dvojice (y, f) podle klíče $w(f)$.

Komentář: Následuje stručná ukázka průběhu Jarníkova algoritmu, kde kostru opět vyznačujeme tučnými čarami a zpracované vrcholy kroužky kolem.





Rozšiřující studium

S běžnými grafovými algoritmy, které zahrnují zde uvedené a mnohem více, se studenti informatiky seznámí hlouběji v každém kurzu návrhu algoritmů. (Navíc jsou běžné grafové algoritmy výborně popsány na Wikipedii. . .) Rozšíření postupu prohledávání grafu nám například umožní vyhledávat 2-souvislé nebo silně-souvislé komponenty. Pro hledání nejkratší cesty také existují sofistikovanější algoritmy, které jsou schopny pracovat i za přítomnosti záporných hran (ale bez záporných cyklů), jako třeba Johnsonův algoritmus. My si dále uvedeme jako příklad Floyd-Warshallův algoritmus s důkazem správnosti v Lekci 11.

Celkově je třeba říci, že v našem kurzu se na rozdíl od kurzů algoritmických nezabýváme vůbec otázkami programátorské implementace a efektivity, nýbrž studujeme matematické aspekty grafových úloh a metod jejich řešení. Oba pohledy by pak dobře vybavený informatik měl znát.

A pokud máte zájem se dozvědět více o zajímavé české historii problému minimální kostry, přečtěte si o ní v dříve zmíněné knize *Kapitoly z diskretní matematiky* autorů Jiřího Matouška a Jaroslava Nešetřila.

9 Orientované grafy, Toky v sítích

Úvod

Jak jsme viděli již v předchozí lekci, grafy se často používají k modelování různých druhů sítí z reálného světa. Na rozdíl třeba od cestních sítí se však nyní budeme zabývat typem síťových úloh, ve kterých není podstatná délka hran a spojení, nýbrž jejich propustnost. Typickými příklady jsou potrubní nebo počítačové sítě. Základní úlohou v této oblasti je problém nalezení maximálního toku v síti za podmínky respektování daných kapacit hran. I na tento problém si uvedeme jednoduchý klasický algoritmus a především si budeme všimnout mnohých důležitých matematických důsledků tohoto algoritmu, jmenovitě vázaných k tzv. dualitě maximálního toku a minimálního řezu, která přirozeně vysvětluje optimalitu řešení mnohých úloh. Následně si ukážeme široké využití toků v síti, které sahá od grafové souvislosti až třeba pro problém segmentace obrazu.

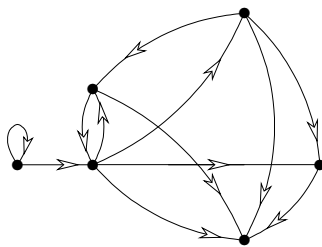
Jelikož pro zkoumaný typ síťových úloh je klíčová implicitní orientace každé hrany, v této lekci začneme samotnou definicí orientovaného grafu a základními vlastnostmi; obdobně úvodní látce vyložené o grafech v Lekci 7.

Cíle

Zadefinujeme orientované grafy a uvedeme si jejich základní vlastnosti, včetně orientované souvislosti. Definujeme pojem síť s kapacitami hran a ukážeme si základní algoritmus pro úlohu hledání maximálního toku. Probereme některé důležité teoretické důsledky postupu nalezení maximálního toku; především dualitu toku a řezu, vyšší úroveň souvislosti grafu, hledání bipartitního párování a systémů různých reprezentantů.

9.1 Základní pojmy orientovaných grafů

Ve většině aplikací grafů se v základu soustředíme na neorientované grafy (a případná orientovaná rozšíření úloh jsou přímočará), ale existují i některé problémy, které si orientaci hran grafu vynucují. Jedním z nich je právě náš problém toků v sítích. Požadavek explicitně vyjádřit směr hrany přirozeně vede na následující definici orientovaného grafu, ve kterém hrany jsou uspořádané dvojice vrcholů. (V obrázcích kreslíme orientované hrany se šipkami.)



Definice 9.1. *Orientovaný graf* je uspořádaná dvojice $D = (V, E)$, kde V je množina vrcholů a $E \subseteq V \times V$ je množina hran – tj. podmnožina vybraných uspořádaných dvojic vrcholů.

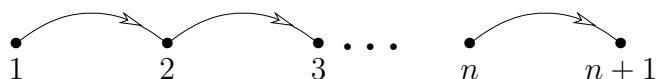
Pojmy *podgrafu* a *isomorfismu* z Oddílu 7.2 se přirozeně přenášejí na orientované grafy.

Značení: Hrana (u, v) (zvaná také *šipka*) v orientovaném grafu D začíná ve vrcholu u a končí ve (míří do) vrcholu v . Opačná hrana (v, u) je různá od (u, v) .

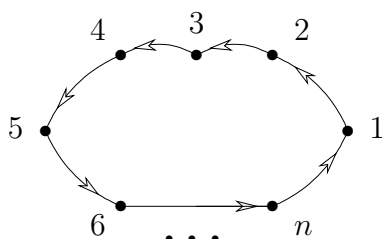
Speciálně hrana tvaru (u, u) se nazývá *orientovaná smyčka*.

Komentář: Orientované grafy odpovídají relacím, které nemusí být symetrické. Mezi stejnou dvojicí vrcholů mohou tudíž existovat dvě hrany v obou směrech, nebo jen kterákoliv jedna z nich nebo žádná. Všimněte si také, že orientované grafy implicitně odlišujeme od obyčejných použitím písmene D místo dřívějších G, H .

Například *orientovaná cesta* délky $n \geq 0$ je následujícím grafem na $n + 1$ vrcholech



a *orientovaná kružnice* (také *cyklus*) délky $n \geq 1$ vypadá takto:



Definice: Počet hran začínajících ve vrcholu u orientovaného grafu D nazveme *výstupním stupněm* $d_D^+(u)$ a počet hran končících v u nazveme *vstupním stupněm* $d_D^-(u)$.

Komentář: Součet všech výstupních stupňů je přirozeně roven součtu všech vstupních stupňů orientovaného grafu. Důkaz viz Věta 7.3.

Všimněte si, že některá literatura používá opačná znaménka $(-/+)$ při značení výstupních a vstupních stupňů, ale s tím nic nenaděláme a čtenář se prostě vždy musí zorientovat a zjistit aktuální situaci.

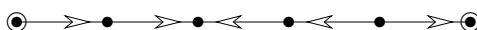
Definice: *Symetrizací* orientovaného grafu D rozumíme neorientovaný graf G vzniklý „zapomenutím směru hran“ v D , přesněji $V(G) = V(D)$ a $uv \in E(G)$ právě když $\{(u, v), (v, u)\} \cap E(D) \neq \emptyset$.

Souvislost na orientovaných grafech

Pojem orientované souvislosti grafu D je natolik fundamentálně odlišný od neorientovaného případu (což je dáno právě jeho „směrovostí“), že si zaslouží samostatnou diskusi i v našem zběžném pohledu na orientované grafy. Uvedeme si odstupňovaně tři základní pohledy na orientovanou souvislost:

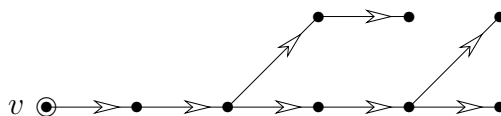
- *Slabá souvislost.* Jedná se o tradiční *souvislost na symetrizaci* grafu D

Komentář: Zjednodušeně a názorně se dá říci, že při cestování grafem „zapomeneme“ směr šipek. Na obrázku:

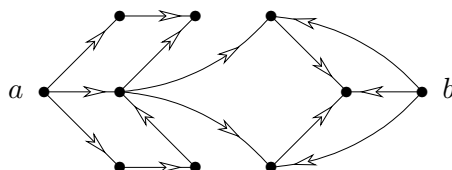


Tento přístup sice může vypadat poněkud nesmyslně – proč šipky zavádět a pak zapomínat jejich směr, avšak brzy u tzv. nenasycených cest uvedeme smysl představy, že cesta v orientovaném grafu se „tlačí“ proti směru hrany.

- *Dosažitelnost (směrem „ven“)*. Orientovaný graf D je *dosažitelný směrem ven*, pokud v něm existuje vrchol $v \in V(D)$ takový, že každý vrchol $x \in V(D)$ je dosažitelný orientovaným sledem z v .

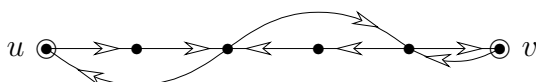


Komentář: Podrobným zkoumáním následujícího obrázku zjistíme, že jeho graf není dosažitelný směrem ven, neboť chybí možnost dosáhnout vrchol b úplně vpravo. Na druhou stranu po vypuštění b je zbylý graf dosažitelný ven z vrcholu a vlevo.



Všimněte si, že definici dosažitelnosti lze snadno „obrátit“, avšak touto další možností se v našem krátkém přehledu nebudeme zabývat.

- *Silná souvislost*. V nejsilnější verzi vyžadujeme současnou existenci spojení (cest) v obou směrech mezi dvojicí vrcholů.

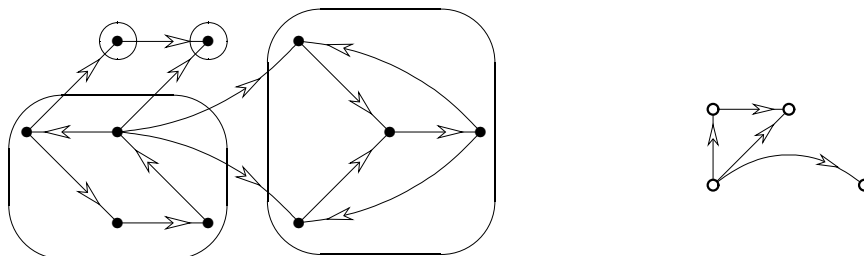


Tvrzení 9.2. *Nechť \approx je binární relace na vrcholové množině $V(D)$ orientovaného grafu D taková, že $u \approx v$ právě když existuje dvojice orientovaných cest – jedna z u do v a druhá z v do u v grafu D . Pak \approx je relace ekvivalence.*

Důkaz (náznak). Postupujeme podobně jako v důkaze Tvrzení 7.11. □

Definice 9.3. Silné komponenty orientovaného grafu D jsou třídy ekvivalence relace \approx uvedené v Tvrzení 9.2. Orientovaný graf D je *silně souvislý* pokud má nejvýše jednu silnou komponentu.

Komentář: Pro ilustraci si mírně upravíme dříve prezentovaný orientovaný graf tak, že bude dosažitelný z nejlevějšího vrcholu. Je výsledek silně souvislý?



Ne, na obrázku jsou vyznačené jeho 4 silné komponenty. Vpravo zároveň uvádíme pro ilustraci obrázek *kondenzace* silných komponent tohoto grafu, což je acyklický orientovaný graf s vrcholy reprezentujícími zmíněné silné komponenty a směry hran mezi nimi.

Pro zvědavé čtenáře poznamenáváme, že si mohou definici silné komponenty porovnat s jádrem předuspořádání v Oddíle 5.4. Kde vidíte drobný rozdíl?

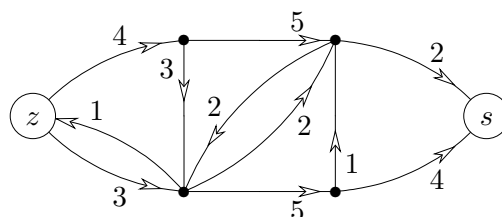
9.2 Definice sítě a toku

Základní strukturou pro reprezentaci sítí je *vážený orientovaný graf* (přičemž implicitní směr hran je v tomto kontextu nezbytný). Váhy hran v takovém modelu vyjadřují „kapacity“ jednotlivých spojů a nás zajímá, jak mnoho uvažované *substance* dokážeme (různými cestami) přenést z daného *zdroje do stoku*. Podstata substance není důležitá, ale ta musí být vhodně rozdělitelná.

Definice 9.4. *Síť* je čtveřice $S = (D, z, s, w)$, kde

- * D je orientovaný graf,
- * vrcholy $z \in V(D)$, $s \in V(D)$ jsou *zdroj* a *stok*,
- * $w : E(D) \rightarrow \mathbb{R}^+$ je kladné ohodnocení hran, zvané *kapacita hran*.

Komentář:



Na obrázku je zakreslena síť s vyznačeným zdrojem z a stokem s , jejíž kapacity hran jsou zapsány čísla u hran. Šipky udávají směr hran, tedy směr proudění uvažované substance po spojnicích. Pokud směr proudění není důležitý, vedeme mezi vrcholy dvojici opačně orientovaných hran se stejnou kapacitou. Kapacity hran pak omezují maximální množství přenášené substance.

Poznámka: V praxi může být zdrojů a stoků více, ale v definici stačí pouze jeden zdroj a stok, z něhož / do něž vedou hrany do ostatních zdrojů / stoků. (Dokonce pak různé zdroje a stoky mohou mít své kapacity.)

Velikost toku v síti

V souladu s praktickými aspekty nás u toku v síti zajímá především to, jaké „množství“ substance (velikost toku) se skutečně přenesou od zdroje ke stoku.

Značení: Pro jednoduchost píšeme ve výrazech značku $e \rightarrow v$ pro hranu e končící ve vrcholu v a $e \leftarrow v$ pro hranu e začínající z v .

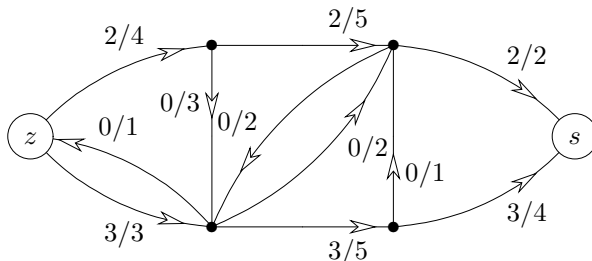
Definice 9.5. *Tok v síti* $S = (D, z, s, w)$ je funkce $f : E(D) \rightarrow \mathbb{R}_0^+$ splňující

- * $\forall e \in E(D) : 0 \leq f(e) \leq w(e)$,
- * $\forall v \in V(D), v \neq z, s : \sum_{e \rightarrow v} f(e) = \sum_{e \leftarrow v} f(e)$.

Velikost toku f je dána výrazem $\|f\| = \sum_{e \leftarrow z} f(e) - \sum_{e \rightarrow z} f(e)$.

Značení: Tok a kapacitu hran v obrázku sítě budeme zjednodušeně zapisovat ve formátu F/C , kde F je hodnota toku na hraně a C je její kapacita.

Komentář: Neformálně tok znamená, kolik substance je každou hranou zrovna přenášeno (ve směru této hrany, proto hrany musí být orientované). Tok je pochopitelně nezáporný a dosahuje nejvýše dané kapacity hrany. Navíc je nutno v každém vrcholu mimo z, s splnit podmínku „zachování substance“ $\sum_{e \rightarrow v} f(e) = \sum_{e \leftarrow v} f(e)$.



Ve vyobrazeném příkladě vede ze zdroje vlevo do stoku vpravo tok o celkové velikosti 5.

Poznámka: Obdobně se dá velikost toku definovat u stoku (či i na obecném řezu v síti, viz dále), neboť platí následující identita odvozená z vlastnosti zachování substance v definici toku

$$0 = \sum_{e \in E} (f(e) - f(e)) = \sum_{v \in V} \left(\sum_{e \leftarrow v} f(e) - \sum_{e \rightarrow v} f(e) \right) = \sum_{v=z,s} \left(\sum_{e \leftarrow v} f(e) - \sum_{e \rightarrow v} f(e) \right).$$

Proto velikost toku počítaná u zdroje je rovna opačné velikosti toku počítaného u stoku

$$\left(\sum_{e \leftarrow z} f(e) - \sum_{e \rightarrow z} f(e) \right) = \left(\sum_{e \rightarrow s} f(e) - \sum_{e \leftarrow s} f(e) \right).$$

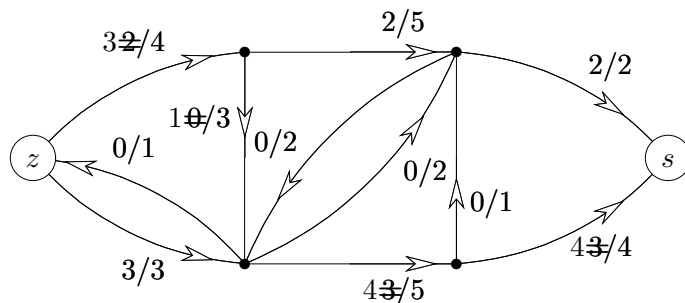
9.3 Nalezení maximálního toku

Naším úkolem je nalézt co největší přípustný tok v dané síti. Pro jeho nalezení existují jednoduché i velmi rychlé algoritmy. Navíc tyto algoritmy mají zajímavé teoretické souvislosti a důsledky uvedené později.

Definice 9.6. *Úloha hledání maximálního toku* v síti $S = (D, z, s, w)$.

Úkolem je v síti S najít tok f ze zdroje z do stoku s podle Definice 9.5 takový, který maximalizuje velikost $\|f\|$.

Komentář: Tok velikosti 5 uvedený v ukázce v předchozí části nebyl optimální, neboť v této síti najdeme i tok velikosti 6:



Jak však poznáme, že větší tok již v dané síti neexistuje? V této konkrétní ukázce to není obtížné, vidíme totiž, že obě dvě hrany přicházející do stoku mají součet kapacit $2 + 4 + 6$, takže více než 6 do stoku ani přitéct nemůže. V obecnosti lze použít obdobnou úvahu, kdy najdeme podmnožinu hran, které nelze tokem „obejít“ a které v součtu kapacit dají velikost našeho toku. Existuje však taková množina hran vždy? Odpověď nám dá následující definice a věta.

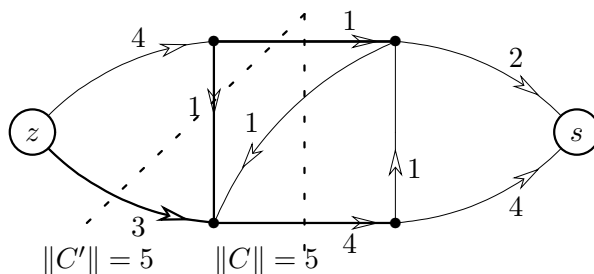
Pojem řezu v síti

Definice 9.7. Řez v síti $S = (D, z, s, w)$ je podmnožina hran $X \subseteq E(D)$ taková, že v podgrafu $D - X$ (tj. po odebrání hran X z D) nezbude žádná orientovaná cesta ze z do s . Velikostí řezu X rozumíme součet kapacit hran z X , tj. $\|X\| = \sum_{e \in X} w(e)$.

Zcela klíčovým poznatkem síťových úloh je následující velmi hezká charakterizace jejich optimálního řešení

Věta 9.8. Maximální velikost přípustného toku v síti je vždy rovna minimální velikosti řezu v ní.

Komentář: Na následujícím obrázku vidíme trochu jinou síť s ukázkou netriviálního minimálního řezu velikosti 5, naznačeného svislou čárkovanou čarou. Všimněte si dobře, že definice řezu mluví o přerušení všech orientovaných cest ze z do s , takže do řezu stačí započítat hrany jdoucí přes svislou čáru od z do s , ale ne hranu jdoucí zpět. Proto je velikost vyznačeného řezu $1 + 4 = 5$.



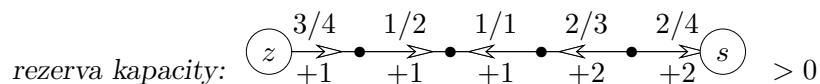
Důkaz Věty 9.8 bude proveden následujícím Algoritmem 9.10.

Nenasycené cesty v síti

Definice: Mějme síť S a v ní tok f . *Nenasycená cesta* P (v S vzhledem k f) je neorientovaná cesta v D mezi určenými vrcholy (obvykle ze z do s), tj. posloupnost navazujících libovolně orientovaných hran e_1, e_2, \dots, e_m , kde $f(e_i) < w(e_i)$ pro e_i ve směru ze z do s a $f(e_j) > 0$ pro e_j v opačném směru.

Hodnotě $w(e_i) - f(e_i) > 0$ pro hrany e_i ve směru z do s a hodnotě $f(e_j) > 0$ pro hrany e_j v opačném směru říkáme *rezerva kapacity* hran. Nenasycená cesta je tudíž cesta s kladnými rezervami kapacit všech hran.

Komentář: Zde vidíme příklad nenasycené cesty ze zdroje do stoku s minimální rezervou kapacity +1.

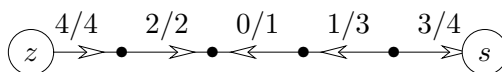


Všimněte si dobře, že cesta není orientovaná, takže hrany na ní jsou v obou směrech.

Metoda 9.9. Maximální tok vylepšováním nenasycených cest.

Základní myšlenkou této jednoduché metody hledání maximálního toku v dané síti je prostě opakovaně vylepšovat tok podél nalezených nenasycených cest.

Komentář: Ve výše zakresleném obrázku nenasyčené cesty byla minimální rezerva kapacity ve výši +1, a tudíž nasycením této cesty o velikost toku 1 vzniká následující fragment přípustného toku v síti:



Zajímavé je se podívat, co se stalo s prostředními zpětně orientovanými hranami – fakticky byl jejich zpětný tok o 1 snížen/zastaven, přičemž toto množství nyní „teče doprava“ (velmi neformálně řečeno).

Fakt: Je-li minimální rezerva kapacity hran nenasyčené cesty P ze z do s ve výši $r > 0$, tok ze z do s zvýšíme o hodnotu r následovně;

- * pro hrany $e_i \in E(P)$ ve směru ze z do s zvýšíme tok na $f'(e_i) = f(e_i) + r$,
- * pro hrany $e_j \in E(P)$ ve směru ze s do z snížíme tok na $f'(e_j) = f(e_j) - r$.

Výsledný tok f' pak bude opět přípustný.

Základní algoritmus nenasyčených cest

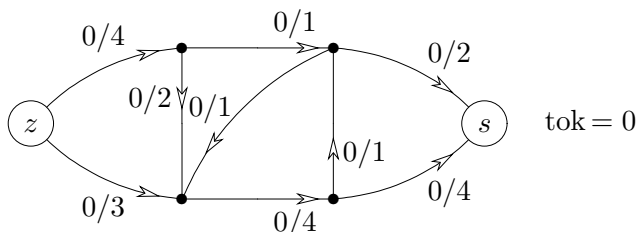
Rámcové implementační podrobnosti Metody 9.9 následují v popise konkrétního původního algoritmu pro náš problém.

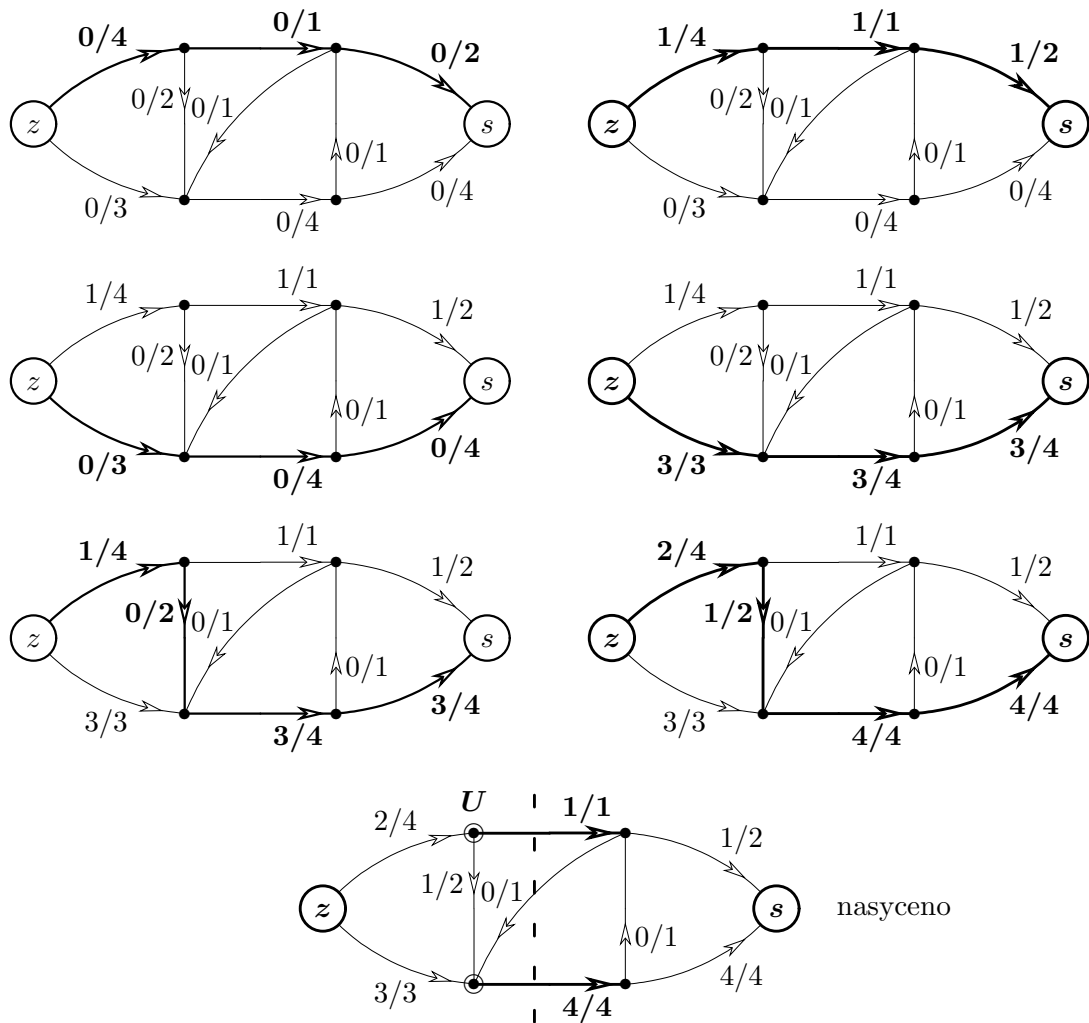
Algoritmus 9.10. Ford–Fulkersonův pro tok v síti.

- *Vstup:* Síť $S = (D, z, s, w)$ podle Definice 9.4.
- Tok $f \leftarrow (0, 0, \dots, 0)$.
- Dále opakujeme následující:
 - * Prohledáváním grafu najdeme množinu U vrcholů D , do kterých se dostaneme ze z po nenasyčených cestách.
 - * Pokud $s \in U$, nechť P značí nalezenou nenasyčenou cestu v S ze z do s .
 - Zvětšíme tok f o minimální rezervu kapacity hran v P .
- Opakujeme kroky výše, dokud nenastane $s \notin U$.
- *Výstup:* Vypíšeme maximální tok f a také minimální řez jako množinu všech hran vedoucích z U do $V(D) - U$.

Příklad 9.11. Ukázka průběhu Algoritmu 9.10

Následují obrázky jednotlivých kroků algoritmu, kde vždy levý obrázek zvýrazní nenasyčenou cestu a vpravo se nasycením zvýší celkový tok.



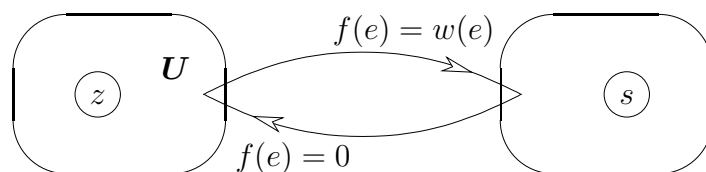


Závěrečný obrázek pak ukazuje výsledný tok velikosti 5 zároveň se zvýrazněným minimálním řezem stejné velikosti. \square

Důkaz a důsledky Ford–Fulkersonova algoritmu

Důkaz správnosti Algoritmu 9.10: Pro každý tok f a každý řez X v síti S platí $\|f\| \leq \|X\|$. Jestliže po zastavení algoritmu s tokem f nalezneme v síti S řez o stejné velikosti $\|X\| = \|f\|$, je jasné, že jsme našli maximální možný tok v síti S . (Pozor, zastavení algoritmu jsme zatím nezdůvodnili, to není tak jednoduché.)

Takže dokažme, že po zastavení algoritmu nastane rovnost $\|f\| = \|X\|$, kde X je vypsáný řez mezi U a zbytkem grafu D . Vezměme tok f v S bez nenasyčené cesty ze z do s . Pak množina U z algoritmu neobsahuje s . Schematicky vypadá situace takto:



Jelikož z U žádné nenasyčené cesty dále nevedou, má každá hrana $e \leftarrow U$ (odcházející

z U) plný tok $f(e) = w(e)$ a každá hrana $e \rightarrow U$ (přicházející do U) tok $f(e) = 0$, takže

$$\sum_{e \leftarrow U} f(e) - \sum_{e \rightarrow U} f(e) = \sum_{e \leftarrow U} f(e) = \sum_{e \in X} w(e) = \|X\| .$$

Na druhou stranu, když v sumě uvažujeme všechny hrany grafu indukované na naší množině U , tj. $e \in E(D \upharpoonright U)$, analogicky dostaneme požadované

$$\begin{aligned} 0 &= \sum_{e \in E(D \upharpoonright U)} (f(e) - f(e)) = \sum_{v \in U} \left(\sum_{e \leftarrow v} f(e) - \sum_{e \rightarrow v} f(e) \right) - \left(\sum_{e \leftarrow U} f(e) - \sum_{e \rightarrow U} f(e) \right) = \\ &= \left(\sum_{e \leftarrow z} f(e) - \sum_{e \rightarrow z} f(e) \right) - \left(\sum_{e \leftarrow U} f(e) - \sum_{e \rightarrow U} f(e) \right) = \|f\| - \|X\| , \quad \text{tj. } \|f\| = \|X\| . \end{aligned}$$

□

Z důkazu Algoritmu 9.10 odvodíme několik zajímavých faktů:

Fakt: Pokud zajistíme, že Algoritmus 9.10 vždy skončí, zároveň tím dokážeme i platnost Věty 9.8.

Fakt: Pro celočíselné kapacity hran sítě S Algoritmus 9.10 vždy skončí.

Fakt: Existují snadná vylepšení Algoritmu 9.10 (jako Edmonds–Karp či Dinitz), u kterých je zaručeno, že výpočet vždy skončí. Obecně k dosažení tohoto cíle postačí vždy uvažovat pouze nejkratší nenasycené cesty, což je ale látka mimo omezený rozsah našeho kurzu.

Pro další využití v teorii grafů je asi nejdůležitější tento poznatek:

Důsledek 9.12. *Pokud jsou kapacity hran sítě S celočíselné, optimální tok v Algoritmu 9.10 také vyjde celočíselně.*

Důkaz: Postupujeme jednoduchou indukcí podle iterací Algoritmu 9.10: Algoritmus začíná s celočíselným tokem 0. V každé další iteraci bude na počátku tok všemi hranami celočíselný, tudíž i rezervy kapacit hran vyjdou (rozdílem od celočíselné kapacity) celočíselně. Proto i hodnoty toku budou změněny jen celočíselně, což zakončuje indukční krok. □

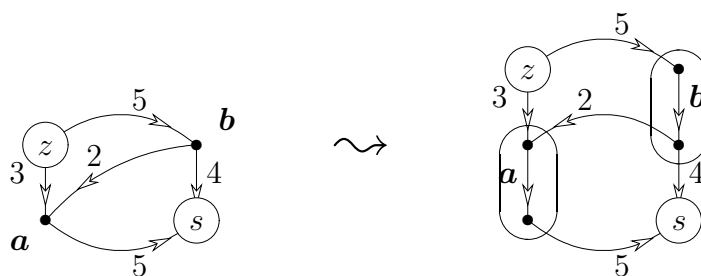
9.4 Zobecněné použití sítí

Pojmy sítě a toků v ní lze v kombinatorické optimalizaci zobecnit a využít několika směry. My si zde stručně uvedeme následující možnosti.

- **Sítě s kapacitami vrcholů:**

U sítě můžeme zadat *kapacity vrcholů*, neboli kapacitní váhová funkce je dána jako $w : E(D) \cup V(D) \rightarrow \mathbb{R}^+$. Takovou síť „zdvojením“ vrcholů snadno převedeme na běžnou síť, ve které kapacity původních vrcholů budou uvedeny u nových hran spojujících

zdvojené vrcholy. Viz neformální schéma:



• **Sítě s dolními kapacitami:**

Pro hrany sítě lze zadat také jejich *minimální kapacitu*, tedy dolní meze přípustného toku, jako váhovou funkci $\ell : E(D) \rightarrow \mathbb{R}_0^+$. Přípustný tok pak musí splňovat $\ell(e) \leq f(e) \leq w(e)$ pro všechny hrany e . V této modifikaci úlohy již přípustný tok *nemusí vůbec existovat*, takže postup řešení musí zavést počáteční fázi, ve které se *nějaký* tok splňující minimální kapacity nalezne vhodnou úpravou zadané sítě. Poté se pokračuje metodou nenasyčených cest.

Bipartitní párování

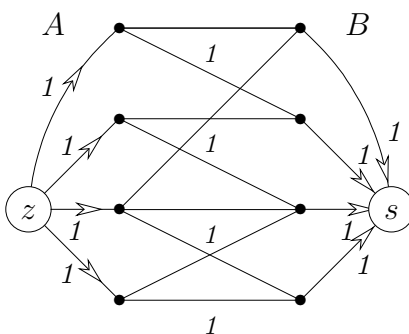
Bipartitní grafy jsou grafy, jejichž vrcholy lze rozdělit do dvou množin tak, že všechny hrany vedou jen mezi těmito množinami, neboli podgrafy úplných bipartitních grafů.

Definice: *Párování* v (nyní bipartitním) grafu G je podmnožina hran $M \subseteq E(G)$ taková, že žádné dvě hrany z M nesdílejí koncový vrchol.

Komentář: Úlohu nalézt v daném bipartitním grafu co největší párování lze poměrně snadno vyřešit pomocí toků ve vhodně definované síti. Uvedená metoda použití toků v síti na řešení problému párování přitom hezky ilustruje obecný přístup, jakým toky v sítích pomohou řešit i úlohy, které na první pohled se sítěmi nemají nic společného.

Metoda 9.13. Nalezení bipartitního párování

Pro daný bipartitní graf G s vrcholy rozdělenými do množin A, B sestojíme síť S následovně:



- Všechny hrany sítě S orientujeme od zdroje do stoku a přiřadíme jim kapacity 1.
- Nyní najdeme (celočíselný) maximální tok v S Algoritmem 9.10. Do párování vložíme ty hrany grafu G , které mají nenulový tok.

Důkaz správnosti Metody 9.13: Podle Důsledku 9.12 bude maximální tok celočíselný, a proto každou hranou poteče buď 0 nebo 1. Jelikož však do každého vrcholu v A může ze zdroje přitéct jen tok 1, bude z každého vrcholu A vybrána do párování nejvýše jedna hrana. Stejně tak odvodíme, že z každého vrcholu B bude vybrána nejvýše jedna hrana, a proto vybraná množina skutečně bude párováním. Zároveň to bude největší možné párování, protože z každého párování lze naopak vytvořit tok příslušné velikosti a větší než nalezený tok v S neexistuje. \square

Poznámka: Popsaná metoda je základem tzv. Maďarského algoritmu pro párování v bipartitních grafech. Úlohu nalezení maximálního párování lze definovat i pro obecné grafy a také ji efektivně algoritmicky vyřešit, ale příslušný algoritmus [Edmondsův] není jednoduchý.

Výběr různých reprezentantů

Definice: Nechtě M_1, M_2, \dots, M_k jsou neprázdné množiny. *Systémem různých reprezentantů* množin M_1, M_2, \dots, M_k nazýváme takovou posloupnost různých prvků (x_1, x_2, \dots, x_k) , že $x_i \in M_i$ pro $i = 1, 2, \dots, k$.

Důležitým a dobře známým výsledkem v této oblasti je Hallova věta plně popisující, kdy lze systém různých reprezentantů daných množin nalézt.

Věta 9.14. *Nechtě M_1, M_2, \dots, M_k jsou neprázdné množiny. Pro tyto množiny existuje systém různých reprezentantů, právě když platí*

$$\forall J \subseteq \{1, 2, \dots, k\} : \left| \bigcup_{j \in J} M_j \right| \geq |J|,$$

neboli pokud sjednocení libovolné skupiny z těchto množin má alespoň tolik prvků, kolik množin je sjednoceno.

Důkaz: Označme x_1, x_2, \dots, x_m po řadě všechny prvky ve sjednocení $M_1 \cup M_2 \cup \dots \cup M_k$. Definujeme si bipartitní graf G na množině vrcholů $\{1, 2, \dots, k\} \cup \{x_1, x_2, \dots, x_m\} \cup \{u, v\}$, ve kterém jsou hrany $\{u, i\}$ pro $i = 1, 2, \dots, k$, hrany $\{v, x_j\}$ pro $j = 1, 2, \dots, m$ a hrany $\{i, x_j\}$ pro všechny dvojice i, j , pro které $x_j \in M_i$.

Komentář: Konstrukce našeho grafu G je obdobná konstrukci sítě v Metodě 9.13: Vrcholy u a v odpovídají zdroji a stoku, ostatní hrany přicházející do vrcholu x_j znázorňují všechny z daných množin, které obsahují prvek x_j .

Cesta mezi u a v má tvar u, i, x_j, v , a tudíž ukazuje na reprezentanta $x_j \in M_i$. Systém různých reprezentantů tak odpovídá k disjunktním cestám mezi u a v . Nechtě X je nyní *libovolná minimální* množina vrcholů v G , neobsahující samotné u a v , po jejímž odebrání z grafu nezbude žádná cesta mezi u a v . Podle této úvahy mají naše množiny systém různých reprezentantů, právě když každá taková oddělující množina X má aspoň k prvků.

Položme $J = \{1, 2, \dots, k\} \setminus X$. Pak každá hrana z J (mimo u) vede do vrcholů z $X \cap \{x_1, \dots, x_m\}$ (aby nevznikla cesta mezi u, v), a proto

$$\left| \bigcup_{j \in J} M_j \right| = |X \cap \{x_1, \dots, x_m\}| = |X| - |X \cap \{1, \dots, k\}| = |X| - k + |J|.$$

První rovnost vyplývá z minimality množiny X . Vidíme tedy, že $|X| \geq k$ pro všechny (minimální) volby oddělující X , právě když $\left| \bigcup_{j \in J} M_j \right| \geq |J|$ pro všechny volby J , což je dokazovaná podmínka naší věty. \square

Poznámka: Tento důkaz nám také dává návod, jak systém různých reprezentantů pro dané množiny nalézt – stačí použít Algoritmus 9.10 na vhodně odvozenou síť.

Rozšiřující studium

Orientované grafy obvykle v učebnicích teorie grafů mnoho místa nezabírají a speciálně v češtině je málo zdrojů věnujících se přímo orientovaným grafům. Mnohé aspekty grafů (včetně některých grafových algoritmů) však zůstávají stejné i v orientovaném podání, jak můžeme vidět třeba v otázkách isomorfismu či hledání nejkratší cesty. Otázky síťových toků a silné souvislosti studované v této lekci však dostávají na orientovaných grafech „nový rozměr“ nemající v neorientovaných grafech rozumné obdoby - pro ně je směr hran podstatný. My se zde hlouběji orientovanými grafy nezabýváme, ale zájemce odkazujeme především na algoritmické zdroje a kurzy, neboť úloha síťových toků je jednou ze základních praktických úloh kombinatorické optimalizace.

10 Formalizace a důkazy pro algoritmy

Úvod

Po předchozí převážně matematické látce se náš výklad obrací bezprostředně k informatice. Mnozí z vás si asi již všimli, že umění programovat není zdaleka jen o tom naučit se syntaxi programovacího jazyka, ale především o schopnosti vytvářet a správně formálně zapisovat algoritmy. Přitom třeba situace, kdy programátorem zapsaný kód ve skutečnosti počítá něco trochu jiného, než si autor představuje, je snad nejčastější programátorskou chybou – o to zákeřnější, že ji žádný „chytrý“ překladač nemůže odhalit.

Proto již na počátku studia informatiky je žádoucí klást důraz na správné chápání zápisu algoritmů i na přesné důkazy jejich vlastností a správnosti. A to je téma, kterému se budeme věnovat následující dvě lekce.

Cíle

Bude zaveden způsob formálního zápisu algoritmů pro potřeby dalšího výkladu, nezávisle na konkrétních programovacích jazycích. Na tomto formalismu pak bude ukazováno správné chápání chování algoritmů a příklady důkazů na konkrétních „malých“ algoritmech. Nejdůležitější technikou důkazů bude matematická indukce. Na druhou stranu je třeba dodat, že uváděné algoritmy jsou pouze bezvýznamné ilustrativní ukázky pro cvičení důkazů a není úkolem tohoto textu učit čtenáře návrhu algoritmů.

10.1 Formální popis algoritmu

Před samotným závěrem našeho matematického kurzu si položíme klíčovou otázku, co je vlastně algoritmus? Když se na tím zamyslíte, asi zjistíte, že to není tak jednoduché přesně říci. Neformálně je algoritmus něčím jako kuchařským receptem, podle kterého ze surovin (vstupů) uvaříme chutné jídlo (očekávaný výstup/výsledek). Říci toto matematicky přesně je však natolik obtížný úkol, že si zde můžeme podat jen docela zjednodušenou (či naivní?) odpověď, přesto však dostatečnou pro zamýšlenou demonstraci matematických důkazů pro běžné algoritmy.

Poznámka: Za definici algoritmu je obecně přijímána tzv. *Church–Turingova teze* tvrdící, že všechny algoritmy lze „simulovat“ na Turingově stroji. Jedná se sice o přesnou, ale značně nepraktickou definici. Mimo Turingova stroje existují i jiné *matematické modely výpočtů*, jako třeba stroj RAM, který je abstrakcí skutečného strojového kódu, nebo také třeba tzv. *neprocedurální* (neimperativní) modely zahrnující funkcionální a logické programování.

Konvence 10.1. Zjednodušeně zde *algoritmem* rozumíme konečnou posloupnost elementárních výpočetních *kroků*, ve které každý další krok *vhodně*¹ využívá (neboli závisí na) vstupní údaje či hodnoty vypočtené v předchozích krocích. Tuto závislost přitom pojímáme zcela obecně nejen na operandy, ale i na vykonávané instrukce v krocích.

Pro zápis algoritmu a jeho zpřehlednění a zkrácení využíváme *řídící konstrukce* – podmíněná větvení a cykly. Při jejich použití však je třeba dát dobrý pozor, aby byla naplněna podmínka skončení algoritmu – blíže viz Oddíl 11.1.

Komentář: Vidíte, jak blízké si jsou konstruktivní matematické důkazy a algoritmy v našem pojetí? Jedná se nakonec o jeden ze záměrů našeho přístupu. . .

¹Zvídaví studenti si mohou na tomto místě uvědomit, že ve slůvku „vhodně“ se skrývá celá hloubka Church–Turingovy teze. V žádném případě tak nelze mechanicky bez zamýšlení obracet, že by každá posloupnost kroků atd . . . byla algoritmem ve smyslu této teze (viz také Lekce 12).

Příklad 10.2. Zázpis algoritmu pro výpočet průměru daného pole $a[]$ s n prvky.

Algoritmus.

- Inicializujeme $sum \leftarrow 0$;
- postupně pro $i=0,1,2,\dots,n-1$ provedeme
 - * $sum \leftarrow sum+a[i]$;
- vypíšeme podíl (sum/n) . □

Ve „vyšší úrovni“ formálnosti (s jasnějším vyznačením *elementárních kroků* a *řídících struktur* algoritmu) se totéž dá zapsat jako:

Algoritmus 10.3. Průměr z daného pole $a[]$ s n prvky.

```
input pole a[] délky n;
sum ← 0;
foreach i ← 0,1,2,...,n-1 do
    sum ← sum+a[i];
done
res ← sum/n;
output res.
```

Symbolický zápis algoritmů

Značení. Pro potřeby symbolického formálního zápisu algoritmů v předmětu FI:IB000 si zavedeme následující pravidla:

- * *Proměnné* nebudeme deklarovat ani typovat, pole odlišíme závorkami $p[]$.
- * *Přiřazení* hodnoty zapisujeme $a \leftarrow b$, případně $a := b$, ale nikdy ne $a=b$.
- * Jako elementární operace je možné použít jakékoliv *aritmetické výrazy* v běžném matematickém zápise. Rozsahem a přesností čísel se zde nezabýváme.
- * Podmíněné *větvení* uvedeme klíčovými slovy `if ... then ... else ... fi`, kde `else` větev lze vynechat (a někdy, na jednom řádku, i `fi`).
- * Pevný *cyklus* uvedeme klíčovými slovy `foreach ... do ... done`, kde část za `foreach` musí obsahovat *předem danou* množinu hodnot pro přiřazování do řídicí proměnné.
- * *Podmíněný cyklus* uvedeme klíčovými slovy `while ... do ... done`. Zde se může za `while` vyskytovat jakákoliv logická podmínka.
- * V zápise používáme jasné *odsazování* (zleva) podle úrovně zanoření řídicích struktur (což jsou `if`, `foreach`, `while`).
- * Pokud je to dostatečně jasné, elementární operace nebo podmínky můžeme i ve formálním zápise *popsat běžným jazykem*.

Poznámka: Známá inforatická poučka (a zároveň název klasické učebnice programování od N. Wirtha) praví „*Algoritmy + Datové struktury = Programy*“. Jelikož však náš text neaspiruje být učebnicí programování ani algoritmizace, druhý sčítanec uvedené poučky záměrně pomíjíme a aspekty použití vhodných datových struktur pro uváděné algoritmy ponecháváme pro jiné kurzy studia informatiky.

Vyzbrojení těmito pravidly symbolického zápisu a matematickým aparátem dokazování si nyní můžeme dovolit se podívat na mnohé základní (a jednoduché) algoritmy novým kritickým pohledem.

Co počítá následující algoritmus?

Příklad 10.4. Je dán následující symbolicky zapsaný algoritmus. Co je jeho výstupem v závislosti na vstupech a, b ?

Algoritmus 10.5.

```
input a, b;
res ← 7;
foreach i ← 1, 2, ..., b-1, b do
    res ← res + a + 2·b + 8;
done
output res .
```

Nejprve si zkusmo vypočítáme hodnoty výsledku res v počátečních iteracích cyklu:

$$\begin{aligned} b = 0: & \quad res = 7, \\ b = 1: & \quad res = 7 + a + 2b + 8, \\ b = 2: & \quad res = 7 + (a + 2b + 8) + (a + 2b + 8), \dots \end{aligned}$$

Co dále? Výčet hodnot naznačuje pravidelnost a závěr, že obecný výsledek po b iteracích cyklu bude mít hodnotu

$$res = 7 + b(a + 2b + 8) = ab + 2b^2 + 8b + 7.$$

Jak však toto dokážeme? Nejlépe asi indukcí podle vstupní hodnoty b , ale to není formálně úplně přesné, neboť s hodnotou b musíme pracovat jako se zcela obecným pevným parametrem a zároveň ji měnit(?) podle potřeb indukce. Pro tuto část odkazujeme do Oddílu 10.3. □

10.2 O „správnosti“ a dokazování programů

Jak se máme přesvědčit, že je daný program počítá „správně“?

- * Co třeba ladění programů? Jelikož počet možných vstupních hodnot je (v principu) neohrazený, *nelze otestovat* všechna možná vstupní data.
- * Situace je zvláště komplikovaná v případě paralelních, randomizovaných, interaktivních a nekončících programů (operační systémy, systémy řízení provozu apod.). Takové systémy mají *nedeterministické chování* a opakované experimenty vedou k různým výsledkům. (Nelze je tudíž ani rozumně ladit, respektive ladění poskytne jen velmi nedostatečnou záruku správného chování za jiných okolností. . .)
- * V některých případech je však třeba mít naprostou jistotu, že program funguje tak jak má, případně že splňuje základní bezpečnostní požadavky.
 - Pro „malé“ algoritmy je možné podat přesný matematický důkaz správnosti.
 - Narůstající složitosti programových systémů a požadavky na jejich bezpečnost si pak vynucují vývoj jiných „spolehlivých“ formálních *verifikačních metod*.

Komentář: Mimochodem, co to vlastně znamená „počítat správně“? Uvědomte si, že abychom toto rozumně zodpověděli, je třeba mít dopředu dány explicitní specifikace očekávaného chování programu.

Ukázka formálního důkazu algoritmu

Příklad 10.6. Je dán následující symbolicky zapsaný algoritmus. Dokažte, že jeho výsledkem je „výměna“ vstupních hodnot a, b .

Algoritmus 10.7.

```
input a, b;
a ← a+b;
b ← a-b;
a ← a-b;
output a, b.
```

Pro správný formální důkaz si musíme nejprve uvědomit, že je třeba symbolicky odlišit od sebe proměnné a, b od jejich daných vstupních hodnot, třeba h_a, h_b . Nyní v krocích algoritmu počítáme hodnoty proměnných:

```
* a = h_a, b = h_b,
* a ← a + b = h_a + h_b, b = h_b,
* a = h_a + h_b, b ← a - b = h_a + h_b - h_b = h_a,
* a ← a - b = h_a + h_b - h_a = h_b, b = h_a,
```

V jednotlivých krocích tak jasně vidíme, že pro *kteřekoliv* vstupní hodnoty h_a, h_b bude ve výsledku obsahem proměnné a hodnota h_b a obsahem proměnné b hodnota h_a . Tímto jsme s důkazem hotovi. \square

10.3 Jednoduché indukční dokazování

Příklad 10.6 je poněkud extrémní ukázkou ve smyslu toho, že v praxi jen málokdy zkoumaný algoritmus udělá tak málo kroků, že bychom každý krok mohli studovat zvlášť. Především počet kroků (cyklů či iterací) bývá proměnný a závislý na vstupu a pak musí nastoupit lepší důkazové prostředky, především *matematická indukce*, která je „jako stvořená“ pro formální uchopení opakovaných sekvencí v algoritmech.

Příklad 10.8. Dokažte, že následující algoritmus v závislosti na vstupech a, b navrátí výsledek $ab + 2b^2 + 8b + 7$ (viz Příklad 10.4).

Algoritmus 10.9.

```
input a, b;
res ← 7;
foreach i ← 1, 2, ..., b-1, b do
    res ← res + a + 2·b + 8;
done
output res.
```

V první řadě si z důvodu formální přesnosti přeznačíme mez cyklu v algoritmu na `foreach i ← 1, 2, ..., c do ..` (kde $c = b$). To děláme proto, že jinak by se nám pletla hodnota b vstupující do přiřazení do proměnné res s hodnotou počtu provedených iterací

cyklu. Poté postupujeme přirozeně indukcí podle počtu c iterací (už nezávisle na vstupní hodnotě b); dokazujeme, že výsledek výpočtu algoritmu bude

$$res = (a + 2b + 8)c + 7 = ac + 2bc + 8c + 7.$$

Pro $c = 0$ je výsledek správně $res = 0 + 7$. Pokud dále předpokládáme platnost vztahu $res = ac + 2bc + 8c + 7$ po nějakých c iteracích cyklu `foreach`, tak následující iterace pro $i \leftarrow c + 1$ (jejíž průběh na samotné hodnotě i nezáleží) změní hodnotu na

$$\begin{aligned} res &\leftarrow res + a + 2b + 8 = ac + 2bc + 8c + 7 + a + 2b + 8 = \\ &= a(c + 1) + 2b(c + 1) + 8(c + 1) + 7. \end{aligned}$$

Důkaz indukci je tím hotov. □

Příklad 10.10. Zjistěte, kolik znaků 'x' v závislosti na celočíselné hodnotě n vstupního parametru n vypíše následující algoritmus.

Algoritmus 10.11.

```
input n;
foreach i ← 1, 2, 3, ..., n-1, n do
    foreach j ← 1, 2, 3, ..., i-1, i do
        vytiskni 'x';
    done
done
```

Nejprve si uvědomíme, že druhý (vnořený) cyklus vždy vytiskne celkem i znaků 'x'. Proto iterací prvního cyklu (nejspíše) dostaneme postupně $1 + 2 + \dots + n$ znaků 'x' na výstupu, což již víme (Příklad 2.8), že je celkem $\frac{1}{2}n(n + 1)$. Budeme tedy dokazovat následující tvrzení:

Věta. Pro každé přirozené n Algoritmus 10.11 vypíše právě $\frac{1}{2}n(n + 1)$ znaků 'x'.

Důkaz: Postupujeme indukcí podle n . Báze pro $n = 0$ je zřejmá, neprovede se ani jedna iterace cyklu a tudíž bude vytištěno 0 znaků 'x', což je správně.

Nechť tedy tvrzení platí pro jakékoliv n_0 a položme $n = n_0 + 1$. Prvních n_0 iterací vnějšího cyklu podle indukčního předpokladu vypíše (ve vnitřním cyklu) celkem $\frac{1}{2}n_0(n_0 + 1)$ znaků 'x'. Pak již následuje jen jedna poslední iterace vnějšího cyklu s $i \leftarrow n = n_0 + 1$ a v ní se vnitřní cyklus $j \leftarrow 1, 2, \dots, i = n$ iteruje celkem $n = n_0 + 1$ -krát. Celkem tedy bude vytištěn tento počet znaků 'x':

$$\frac{1}{2}n_0(n_0 + 1) + n_0 + 1 = \frac{1}{2}(n_0 + 1 + 1)(n_0 + 1) = \frac{1}{2}n(n + 1)$$

Důkaz indukčního kroku je hotov. □

10.4 Rekurzivní algoritmy

Rekurentní vztahy posloupností, stručně uvedené v Oddíle 3.4, mají svou přirozenou obdobu v *rekurzivně zapsaných algoritmech*. Zjednodušeně řečeno to jsou algoritmy, které se v průběhu výpočtu odvolávají na výsledky sebe sama pro jiné (pokud možno striktně menší) vstupní hodnoty. U takových algoritmů je zvláště důležité kontrolovat

jejich správnost a také proveditelnost. My si tuto oblast algoritmů osvětlíme několika jednoduchými příklady.

Příklad 10.12. *Symbolický zápis jednoduchého rekurzivního algoritmu.*

Algoritmus .

```
function factorial(x):
    if x < 1 then t ← 1;
    else t ← x · factorial(x-1);
return t.
```

Co je výsledkem výpočtu? Jednoduše řečeno, výsledkem je *faktoriál* vstupní přirozené hodnoty x , tj. hodnota $x! = x \cdot (x-1) \cdot \dots \cdot 2 \cdot 1$. (Záporné a necelé vstupní hodnoty x pro zjednodušení ignorujeme, ale poctivě napsaný program by si samozřejmě měl přípustnost vstupu kontrolovat sám!) Všimněte si ještě, že algoritmus i pro vstup $x = 0$ správně vyhodnotí $0! = 1$. \square

Fibonacciho čísla

Pro další příklad rekurze se vrátíme k Oddílu 3.4, kde byla zmíněna známá Fibonacciho posloupnost 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ... Ve skutečnosti tuto posloupnost budeme uvažovat již od nultého členu, tj. jako 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

Algoritmus 10.13. Rekurzivní výpočet členů Fibonacciho posloupnosti.

Pro dané přirozené $x \geq 0$ vypočítáme x -té Fibonacciho číslo následovně:

```
function fibonacci(x):
    if x < 2 then t ← x;
    else t ← fibonacci(x-1)+fibonacci(x-2);
return t.
```

Komentář: Správnost Algoritmu 10.13 je víceméně zřejmá z jeho přímé podoby s rekurentním vzorcem v definici Fibonacciho čísel. Zamyslete se však, jak je to s praktickou „proveditelností“ takového algoritmu... Vidíte (případně si zkuste naprogramovat), že čas výpočtu roste velmi rychle? Třebaže hodnotu $fibonacci(30)$ tímto algoritmem spočítáte poměrně rychle, s výpočtem $fibonacci(40)$ už budete mít větší problémy a $fibonacci(50)$ asi bude mimo vaše možnosti. To skutečně není dobrý algoritmus! Podívejte se také na budoucí Příklad 11.5, který odhaduje, jak mnoho (exponenciálně) kroků výpočtu je potřeba.

Proto si v dalším Příkladu 10.14 uvedeme poněkud (ve skutečnosti velmi výrazně) lepší algoritmus výpočtu, podobající se přirozenému lidskému postupu psaní členů posloupnosti „do řádku na papír“. Doporučujeme si oba algoritmy zkusit implementovat a mezi sebou porovnat.

Příklad 10.14. *Nerekurzivní algoritmus pro Fibonacciho čísla.*

Dokažte, že následující algoritmus pro každé přirozené n počítá tutéž hodnotu jako rekurentní funkce $fibonacci(n)$ v Algoritmu 10.13 (ale mnohem mnohem rychleji).

Algoritmus .

```
input n;
b[0] ← 0; b[1] ← 1;
foreach i ← 2,3,...,n do
    b[i] ← b[i-1]+b[i-2];
```

done
output b[n] .

Důkaz: Indukcí podle řídicí proměnné i budeme dokazovat, že po i -té iteraci cyklu algoritmu bude vždy platit $b[i] = \text{fibonacci}(i)$: Co se týče báze indukce, toto vyplývá z úvodního přiřazení. Pro libovolné $i \geq 2$ předpokládáme platnost indukčního předpokladu $b[j] = \text{fibonacci}(j)$ pro $j \in \{i-1, i-2\}$. V i -té iteraci cyklu pak nastane $b[i] \leftarrow b[i-1] + b[i-2] = \text{fibonacci}(i-1) + \text{fibonacci}(i-2) = \text{fibonacci}(i)$ podle definice. Tím je důkaz hotov pro hodnotu $i = n$. \square

Rozšiřující studium

Náš výklad pohlíží na algoritmy a programování tzv. procedurálním neboli imperativním paradigmatickým. Vedle toho existují i jiné přístupy k programování, jako zmíněné funkcionální nebo logické. Na FI se s jinými přístupy studenti seznámí třeba v předmětu IB015 Neimperativní programování. Pro náš výklad není výběr výpočetního paradigmatu to nejdůležitější.

Smyslem této lekce bylo především ukázat, že u jednoduchých algoritmů lze (a je vhodné) je matematicky formálně zapisovat i dokazovat jejich správnost. Samozřejmě je iluzorní předpokládat, že obdobné důkazy správnosti podáme i pro velké softwarové projekty čítající až milióny řádků, ale postupy a techniky naučené při ověřování jednoduchých algoritmů s úspěchem využijete i při kontrole a ladění jednotlivých kousků velkých projektů.

O mnoha různých pokročilých technikách formální verifikace programů se v případě zájmu dozvíte v pokračujícím studiu. Tyto techniky jsou schopny na vhodném „modelu“ rutinně a matematicky přesně ověřovat (ne)existenci mnoha běžných programátorských chyb.

11 Pokročilé dokazování nad algoritmy

Úvod

Lekce navazuje na Lekci 10 a pokračuje v ukázkách matematického dokazování nad relativně krátkými a jednoduchými algoritmy. Výklad již bude trochu náročnější a mimo uměle přichystaných triviálních příkladů se budeme věnovat i několika obecně dobře známým a používaným algoritmům. Čtenář by se měl v ukázkách důkazů soustředit na pochopení, jak jednotlivé formální matematické kroky korespondují s průběhem algoritmů.

Cíle

Naší snahou v těchto dvou lekcích je čtenáři ukázat poměrně vyčerpávající přehled způsobů a triků, které lze využít k analýze a dokazování správnosti rozumně krátkých algoritmů. Tyto poznatky by měly základem toho, aby si čtenář jako programátor uměl po sobě své algoritmy „přečíst“ a ověřit jejich skutečnou správnost na lokální úrovni.

11.1 Dokazování konečnosti algoritmu a programu

Co bývá snad ještě horší, než chybný výsledek algoritmu/programu? Je to situace, kdy spuštěný program běží „do nekonečna“ a vůbec se nezastaví.

Minule jsme problém zastavení algoritmu jednoduše obešli Konvencí 10.1, která vyžaduje, aby posloupnost kroků algoritmu byla konečná, a dále se nad tím nezamýšleli. To je ostatně běžný informaticko–filozofický přístup. Co když však, v důsledku použití řídicích konstrukcí v zápise algoritmu, bude podmínka konečnosti porušena? V příkladech Lekce 10 to naštěstí nemůže nastat, neboť jsme dosud využívali jen `foreach` cykly – přitom podle naší konvence `foreach` cyklus obsahuje předem danou konečnou množinu hodnot dosazovaných do řídicí proměnné a tudíž musí skončit.

Avšak už v příštím algoritmu využijeme `while` cyklus, u kterého nemusí být obecně jasné *kdy a jestli skončí*, a tudíž pro formálně korektní návrh algoritmu bude potřebný i důkaz konečnosti. Předeseíláme, že další podnětná látka k tématu skončení algoritmu se nachází v Lekci 12, kde mimo jiné dokážeme, že otázka zastavení daného programu je algoritmicky neřešitelná.

Příklad 11.1. *Zastaví se vždy výpočet následujícího primitivního programu?*

Algoritmus 11.2.

```
input x;
while x>5 do
    x ← x+1;
done
output x .
```

Odpověď je samozřejmě NE, jak každý vidí pro jakýkoliv vstup x větší než 5. Jak však tuto negativní odpověď matematicky dokázat? To není zcela jednoduché, a proto si pomůžeme následujícím trikem:

- Předpokládejme pro spor, že Algoritmus 11.2 někdy skončí pro $x > 5$. Nechť přirozené $n > 5$ je zvoleno tak, že Algoritmus 11.2 skončí pro $x = n$ po nejmenším možném počtu ℓ průchodů cyklem `while`.

Pak jistě $\ell > 0$, neboť na začátku je podmínka `x>5` splněna z definice n . Po prvním průchodu pak $x = n + 1 > 5$, avšak nyní již Algoritmus 11.2 musí skončit po $\ell - 1 < \ell$

dalších průchodech cyklu. To je *spor* s volbou $x = n$ coby vstupní hodnoty s nejmenším možným počtem průchodů (pro $x = n + 1$ totiž do ukončení výpočtu proběhne o jeden průchod méně – o ten jeden vykonaný na začátku naší úvahy). \square

Komentář: Pokud vás teď napadá, proč tedy povolujeme `while` cyklus jako řídicí konstrukci, když s `foreach` cyklem bychom si ušetřili problémy, tak odpověď je krátká a jasná: Lze matematicky dokázat, že některé (samozřejmě korektní) algoritmy nelze zapsat programem bez použití podmíněných cyklů.

Když algoritmus vždy končí

Na rozdíl od Příkladu 11.1 se budeme nadále věnovat pouze algoritmům, které poslušně končí své výpočty v souladu s naší Konvencí 10.1. V první řadě se musíme zamyslet, jak konečnost algoritmů budeme dokazovat.

Metoda 11.3. *Jednoduchý důkaz konečnosti.*

Máme-li za úkol dokázat, že algoritmus skončí, lze doporučit následující postup:

- * Sledujeme zvolený celočíselný a zdola ohraničený *parametr algoritmu* (třeba přirozené číslo) a dokážeme, že se jeho hodnota v průběhu algoritmu neustále *ostře zmenšuje*.
- * Případně předchozí přístup rozšíříme na zvolenou *k-tici přirozených parametrů* a dokážeme, že se jejich hodnoty v průběhu algoritmu lexikograficky ostře zmenšují.

Jedná se zde vlastně o vhodné (a zjednodušené pro daný účel) využití principu matematické indukce. Pozor, naše „parametry“ vůbec nemusejí být proměnnými v programu, a přesto jsou s programem implicitně nerozlučně svázány.

Komentář: Například pro rekurzivní funkci `factorial(x)` z Příkladu 10.12 přímo využijeme parametr `x`, který se ostře zmenšuje s každým voláním.

Zamyslíme-li se nad Metodou 11.3 do hloubky (matematické), zjistíme, že stejně jako matematická indukce je založená na této vlastnosti: V každé podmnožině přirozených čísel existuje jedno nejmenší (tomu se říká *dobré uspořádání*). Proto garance zmenšování přirozené hodnoty parametru algoritmu prostě musí znamenat ukončenost jeho výpočtu, bez ohledu na to, co náš vymyšlený parametr znamená. Nejlépe je to ilustrovat dalšími názornými příklady.

Příklad 11.4. *Dokažte, že následující algoritmus vždy skončí pro jakýkoliv přirozený vstup x .*

```
Algoritmus .
input  x;
while  x<100 do
    y ← 0;  x ← x+1;
    while y<x do
        y ← y+1;
    done
done
```

Postupujme podle Metody 11.3 – jak však dosáhneme „zmenšování parametru“, když hodnoty proměnných `x,y` se zvětšují? To není až tolik obtížné, prostě budeme hodnoty `x,y` vhodně odečítat od velké konstanty. Vtip je v tom dobře zvolit vzorec parametru

(vlastním odhadem chování algoritmu doplněným o poznání, že v proměnných x, y se nevyskytnou nikdy hodnoty větší než 100):

$$p(x, y) = 101^2 - 101 \cdot x - y$$

Pak je již rutinou dokázat, že v průběhu algoritmu (tj. pokud se aspoň jednou vnoří do cyklu) je vždy $p(x, y) > 0$ a v každém průchodu vnějšího i vnitřního cyklu se $p(x, y)$ ostře zmenší. Čtenář nechť si toto ověří sám. \square

Příklad 11.5. *Dokažte, že následující algoritmus (viz dřívější 10.13) vždy skončí pro jakýkoliv přirozený vstup x .*

Algoritmus . Rekurzivní výpočet funkce $fibonacci(x)$ pro přirozené x .

```
function fibonacci(x):
    if x < 2 then t ← x;
    else t ← fibonacci(x-1)+fibonacci(x-2);
return t .
```

Nyní je na první pohled přirozené sledovat přímo parametr x . Indukcí podle něj pak snadno odvodíme, že výpočet $fibonacci(x)$ vždy skončí, neboť jsou rekurzivně volány jen menší hodnoty parametru. V čem je tedy možný nedostatek tohoto přímého přístupu? V zásadě pouze jediný; nezískáme z něj žádný rozumný horní odhad počtu kroků algoritmu. Pro vysvětlení, problém spočívá zhruba v tom, že v bodě $fibonacci(x-1)+fibonacci(x-2)$ se výpočet „rozvětjuje“ a my musíme vhodným způsobem umět sečíst počty kroků obou větví.

Na druhou stranu při trikové volbě parametru $p(x) = 2^x$ pro použití Metody 11.3 v každé iteraci rekurzivního volání $fibonacci(x)$ pro $x \geq 2$ v součtu nastane

$$p(x-1) + p(x-2) = 2^{x-1} + 2^{x-2} \leq 2 \cdot 2^{x-1} - 1 < 2^x.$$

Co přesně znamenají slova „v součtu“? To je, že všechny větve rekurzivního výpočtu *dohromady* na jedné úrovni sníží sledovaný parametr. Počet iterací výpočtu $fibonacci(n)$ tak bude vždy nejen konečný, ale podle uvedené úvahy přímo striktně menší než $p(n) = 2^n$.

Přesto, tento algoritmus je velmi pomalý a mnohem lepšího výpočetního času výpočtu Fibonacciho čísel dosáhneme třeba s alternativním algoritmem Příkladu 10.14. \square

Dodatek: Jak konstruovat cykly permutace

Pro mírně složitější ilustraci výše uvedeného konceptu dokazování konečnosti se podíváme na snadný postup nalezení rozkladu permutace na cykly, u kterého však vůbec není na první pohled jasné, zda má skončit.

Algoritmus 11.6. *Cykly permutace.*

Pro danou permutaci π na n -prvkové neprázdné množině $A = \{1, 2, \dots, n\}$ vypíšeme její cykly (viz Oddíl 6.2) takto:

```
input  $\pi$ ;
U ← {1, 2, ..., n} = A;
while U ≠ ∅ do
    x ← min(U);    (nejmenší prvek množiny)
```

```

output začínáme výpis cyklu '⟨';
while x∈U do
    output x ',';
    U ← U\{x};    x ← π(x);
done
output ukončíme výpis cyklu '⟩';
done

```

Jak dokážeme správnost a konečnost tohoto algoritmu? Platí, že přímá aplikace indukce podle n nepřinese nic podstatného. Důkaz si tentokrát rozdělíme na dvě části (podle dvou `while` cyklů).

Věta. Za předpokladu, že vnitřní `while` cyklus pro jakoukoliv počáteční volbu x skončí, vypíše cyklus permutace π obsahující x a odebere všechny prvky tohoto cyklu z množiny U , Algoritmus 11.6 vždy skončí se správným výsledkem.

Důkaz: Postupujeme indukcí podle počtu cyklů v permutaci π . Jediný cyklus v π (báze indukce) je vypsán dle předpokladu věty a množina U zůstane prázdná, tudíž vnější `while` cyklus skončí po první iteraci a výsledek je správný.

Podle Věty 6.6 se každá permutace dá zapsat jako složení disjunktích cyklů. Nechť π je tedy složena z $\ell > 1$ cyklů. Po první iteraci `while` cyklu zbude v restrikci permutace π na množinu U celkem $\ell - 1$ cyklů. Podle indukčního předpokladu pak tyto zbylé cykly budou správně vypsány a algoritmus skončí. \square

Komentář: Vidíte, že v tomto důkaze indukcí je indukční krok zcela triviální a důležitý je zde především základ indukce?

Věta. Pokud π je permutace, tak vnitřní `while` cyklus vždy skončí a nalezne v π cyklus obsahující libovolný počáteční prvek $x \in U$. Navíc všechny prvky nalezeného cyklu odebere z množiny U .

Důkaz: Zde přímo zopakujeme argument důkazu Věty 6.6: Vezmeme libovolný prvek $x = x_1 \in U$ a iterujeme zobrazení $x_{i+1} = \pi(x_i)$ pro $i = 1, 2, \dots$, až dojde ke zopakování prvku $x_k = x_j$ pro $k > j \geq 1$. (To musí nastat, neboť A je konečná.) Jelikož prvek x_j byl již odebrán z U , v kroku $x = x_k$ dojde k ukončení našeho `while` cyklu. Nadto je π prostá, a proto nemůže nastat $x_k = x_j = \pi(x_{j-1})$ pro $j > 1$. Takto byl nalezen a odebrán z U cyklus $\langle a_1, \dots, a_{k-1} \rangle$ a důkaz je hotov. \square

11.2 Přehled technik důkazu indukcí

Doposud v našem textu byla matematická indukce představována ve své přímočaré formě, kdy dokazované tvrzení obvykle přímo nabízelo celočíselný parametr, podle něž bylo potřebné indukci vést. Indukční krok pak prostě zpracoval přechod „ $n = i \rightarrow n = i + 1$ “. To však u dokazování správnosti algoritmů typicky neplatí a našim cílem zde je ukázat možné techniky, jak správně indukci na dokazování algoritmů aplikovat. Uvidíme, jak si z nabízejících se parametrů správně vybrat a jak je případně kombinovat.

Technika fixace parametru

První technika důkazu prostě dopředu za některé parametry dosazuje (obecně zvolené) konstanty. Tato technika je vhodná pro případy, kdy je sice v algoritmu více parametrů,

ale „zjevně“ dochází ke změně jen jednoho (nebo části) z nich a chování algoritmu ke zbylým „neměnným“ parametrům je dobře předvídatelné.

Příklad 11.7. *Mějme následující algoritmus. Co je jeho výsledkem výpočtu?*

```
Algoritmus .
input  x, y;
res ← 0;
while  x > 0 do
    res ← res + y;  x ← x - 1;
done
output res .
```

Sledováním algoritmu zjistíme, že hodnota proměnné `res` bude narůstat jako součet $y + \dots + y$, dokud se `x` nesníží na nulu. Poté odhadneme:

Věta. Pro každé $x, y \in \mathbb{N}$ Algoritmus 11.8 vypočítá hodnotu součinu $res = x \cdot y$.

Jaký je vhodný postup k důkazu tohoto tvrzení indukcí? Je snadno vidět, že na hodnotě vstupu `y` vlastně nijak podstatně nezáleží (lze `y` fixovat) a důležité je sledovat `x`. Tato úvaha nás dovede k následujícímu:

Důkaz: Budiž $h_y \in \mathbb{N}$ libovolné ale pro další úvahy *pevné*. Dokážeme, že pro každý vstup $x \in \mathbb{N}$ je výsledkem výpočtu hodnota $r_0 + x \cdot h_y$, kde h_y byla hodnota vstupu `y` a r_0 byla hodnota v pracovní proměnné `res` na začátku uvažovaného výpočtu (pro potřeby indukce, $r_0 = 0$ na úplném začátku). Podle principu matematické indukce uplatněné na parametr x dostáváme snadno:

- * *Báze* $x = 0$ znamená, že tělo cyklu ve výpočtu ani jednou neproběhne a výsledkem bude počáteční r_0 .
- * *Indukční krok.* Nechť je tvrzení známo pro $x = i$ a uvažujme nyní vstup $x = i + 1 > 0$. Prvním průchodem cyklem se uloží $res \leftarrow res + y = r_0 + h_y = r_1$ a $x \leftarrow x - 1 = i$. Počáteční hodnota pracovní proměnné `res` nyní (pro naše indukční úvahy) tudíž je $r_0 + h_y = r_1$ a podle indukčního předpokladu je pak výsledkem výpočtu hodnota

$$r_1 + i \cdot h_y = (r_0 + h_y) + i \cdot h_y = r_0 + (i + 1) \cdot h_y = r_0 + x \cdot h_y .$$

Důkaz matematickou indukcí je tímto ukončen. □

Komentář: Všimněte si, že techniku fixace parametru jsme mlčky použili již v Příkladě 10.8.

Indukce k součtu parametrů

Druhou techniku je vhodné použít především v případech, kdy se v průběhu algoritmu vždy některý parametr zmenšuje, ale pokaždé je to některý jiný parametr, takže v indukci se nelze zaměřit jen na jeden z nich. Jedná se spíše o situaci u rekurzivních algoritmů.

Příklad 11.8. *Co je výsledkem následujícího rekurzivního výpočtu?*

```
Algoritmus .
function kombinacni(m,n) :
    res ← 1;
    if m > 0 ∧ n > 0 then
```

```

    res ← kombinacni(m-1,n) + kombinacni(m,n-1);
  fi
return res .

```

Výše uvedený vzorec (a ostatně i název funkce) naznačuje, že funkce má co společného s kombinačními čísly a *Pascalovým trojúhelníkem*

$$\binom{a+1}{b+1} = \binom{a}{b+1} + \binom{a}{b},$$

je však třeba správně „nastavit“ význam parametrů a, b, \dots

Věta. Pro každé parametry $m, n \in \mathbb{N}$ je výsledkem výpočtu funkce $\text{kombinacni}(m, n)$ hodnota $res = \binom{m+n}{m}$ (kombinační číslo) – počet všech m -prvkových podmnožin $(m+n)$ -prvkové množiny.

Důkaz indukcí vzhledem k součtu parametrů $i = m + n$:

- * *Báze* $i = m + n = 0$ pro $m, n \in \mathbb{N}$ znamená, že $m = n = 0$. Zde však s výhodou využijeme tzv. „rozšíření báze“ na všechny hraniční případy $m = 0$ nebo $n = 0$ zvlášť. V obou rozšířených případech daná podmínka algoritmu není splněna, a proto výsledek výpočtu bude iniciální $res = 1$. Je toto platná odpověď?

Kolik je prázdných podmnožin ($m = 0$) jakékoliv množiny? Jedna, \emptyset . Kolik je m -prvkových podmnožin ($n = 0$) m -prvkové množiny? Zase jedna, ta množina samotná. Tím je důkaz rozšířené báze indukce dokončen.

- * *Indukční krok* přechází na součet $i + 1 = m + n$ pro $m, n > 0$. Nyní je podmínka algoritmu splněna a vykonají se rekurentní volání

$\text{kombinacni}(m-1, n) + \text{kombinacni}(m, n-1)$.

Rekurentní volání se vztahují k výběru podmnožin $m - 1 + n = m + n - 1 = i$ -prvkové množiny, například $M = \{1, 2, \dots, i\}$. Výsledkem tedy je, podle indukčního předpokladu pro součet i , počet $(m - 1)$ -prvkových plus m -prvkových podmnožin množiny M .

Kolik nyní je m -prvkových podmnožin $(i + 1)$ -prvkové množiny $M' = M \cup \{i + 1\}$? Pokud ze všech podmnožin odebereme prvek $i + 1$, dostaneme právě m -prvkové podmnožiny (z těch neobsahujících $i + 1$) plus $(m - 1)$ -prvkové podmnožiny (z těch původně obsahujících $i + 1$). A to je v součtu rovno $\text{kombinacni}(m-1, n) + \text{kombinacni}(m, n-1)$, jak jsme měli dokázat. \square

Zesílení dokazovaného tvrzení

Velmi častou situací při dokazování algoritmu je, že se zajímáme o hodnoty některých proměnných nebo „výstupy“ některé funkce, ale ke správnému matematickému důkazu musíme „postihnout“ i chování jiných funkcí a proměnných v algoritmu. Taková situace pak typicky vede na potřebu *zesílení požadovaného tvrzení* v matematické indukci.

Příklad 11.9. Zjistěte, kolik znaků 'z' v závislosti na celočíselné hodnotě n vstupního parametru n vypíše následující algoritmus.

Algoritmus 11.10.

```
st ← "z";
foreach i ← 1,2,3,...,n-1,n do
    vytiskni řetězec st;
    st ← st.st; (zřetězení dvou kopií st za sebou)
done
```

Zkusíme-li si výpočet simulovat pro $n = 0, 1, 2, 3, 4 \dots$, postupně dostaneme počty 'z' jako 0, 1, 3, 7, 15... Na základě toho již není obtížné „uhodnout“, že počet 'z' bude (asi) obecně určen vztahem $2^n - 1$. Toto je však třeba dokázat!

Komentář: Jak záhy zjistíme, matematická indukce na naše tvrzení přímo „nezabírá“, ale mnohem lépe se nám povede s následujícím přirozeným zesílením dokazovaného tvrzení:

Věta. Pro každé přirozené n Algoritmus 11.10 vypíše právě $2^n - 1$ znaků 'z' a proměnná st bude na konci výpočtu obsahovat řetězec 2^n znaků 'z'.

Důkaz: Postupujeme indukcí podle n . Báze pro $n = 0$ je zřejmá, neprovede se ani jedna iterace cyklu a tudíž bude vytištěno $0 = 2^0 - 1$ znaků 'z', což bylo třeba dokázat. Mimo to proměnná st iniciálně obsahuje $1 = 2^0$ znak 'z'.

Nechť tedy tvrzení platí pro jakékoliv n_0 a položme $n = n_0 + 1$. Podle indukčního předpokladu po prvních n_0 iteracích bude vytištěno $2^{n_0} - 1$ znaků 'z' a proměnná st bude obsahovat řetězec 2^{n_0} znaků 'z'. V poslední iteraci cyklu (pro $i \leftarrow n = n_0 + 1$) vytiskneme dalších 2^{n_0} znaků 'z' (z proměnné st) a dále řetězec st „zdvojnásobíme“. Proto po n iteracích bude vytištěno celkem $2^{n_0} - 1 + 2^{n_0} = 2^{n_0+1} - 1 = 2^n - 1$ znaků 'z' a v st bude uloženo $2 \cdot 2^{n_0} = 2^{n_0+1} = 2^n$ znaků 'z'. \square

11.3 Zajímavé algoritmy aritmetiky

Pro další ukázky důkazových technik pro algoritmy se podíváme na některé krátké algoritmy z oblasti aritmetiky. Například „zbytkové“ umocňování na velmi vysoké exponenty je podkladem známé RSA šifry:

Algoritmus 11.11. *Binární postup umocňování.*

Pro daná čísla a, b vypočteme jejich celočíselnou mocninu (omezenou na zbytkové třídy modulo m kvůli prevenci přetečení rozsahu celých čísel v počítači), tj. $a^b \bmod m$, následujícím postupem.

```
input a,b, m;
res ← 1;
while b > 0 do
    if b mod 2 > 0 then res ← (res·a) mod m;
    b ← [b/2]; a ← (a·a) mod m;
done
output res.
```

K důkazu správnosti algoritmu použijeme indukci podle délky ℓ binárního zápisu čísla b .

Věta. Algoritmus 11.11 skončí a vždy správně vypočte hodnotu $a^b \bmod m$.

Důkaz: Báze indukce je pro $\ell = 1$, kdy $b = 0$ nebo $b = 1$. Přitom pro $b = 0$ se cyklus vůbec nevykoná a výsledek je $res = 1$. Pro $b = 1$ se vykoná jen jedna iterace cyklu a výsledek je $res = a \bmod m$.

Nechť tvrzení platí pro $\ell_0 \geq 1$ a uvažme $\ell = \ell_0 + 1$. Pak zřejmě $b \geq 2$ a vykonají se alespoň dvě iterace cyklu. Po první iteraci budou hodnoty proměnných po řadě

$$a_1 = a^2, \quad b_1 = \lfloor b/2 \rfloor \quad \text{a} \quad res = r_1 = (a^{b \bmod 2}) \bmod m.$$

Tudíž délka binárního zápisu b_1 bude jen ℓ_0 a podle indukčního předpokladu zbylé iterace algoritmu skončí s výsledkem

$$res = r_1 \cdot a_1^{b_1} \bmod m = (a^{b \bmod 2} \cdot a^{2\lfloor b/2 \rfloor}) \bmod m = a^b \bmod m. \quad \square$$

Euklidův algoritmus

Již z dávných dob antiky pochází následující zajímavý a koneckonců velmi jednoduchý postup – algoritmus pro nalezení největšího společného dělitele dvou čísel.

Algoritmus 11.12. *Euklidův pro největšího společného dělitele.*

Pro zadaná přirozená čísla p, q počítá následovně:

```
input  p, q;
while  p>0 ^ q>0 do
    if  p>q then p ← p-q;
    else q ← q-p;
done
output p+q.
```

Věta. Pro každé $p, q \in \mathbb{N}$ na vstupu algoritmus vrátí hodnotu největšího společného dělitele čísel p a q , nebo 0 pro $p = q = 0$.

Důkaz opět povedeme indukcí podle součtu $i = p + q$ vstupních hodnot. (Jak jsme psali, je to *přirozená volba* v situaci, kdy každý průchod cyklem algoritmu sníží jedno z p, q , avšak není jasné, které z nich.)

- Báze indukce pro $i = p + q = 0$ je zřejmá; cyklus algoritmu neproběhne a výsledek ihned bude 0.
- Ve skutečnosti je zase výhodné uvažovat *rozšířenou bázi*, která zahrnuje i případy, kdy jen jedno z p, q je nulové (Proč? Jedná se o všechny případy, kdy průchod cyklem neproběhne, neboli touto indukcí sledujeme *počet průchodů cyklem*.) Pak výsledek $p + q$ bude roven tomu nenulovému z obou sčítanců, což je v tomto případě zároveň jejich největší společný dělitel.
- *Indukční krok.* Mějme nyní vstupní hodnoty $p = h_p > 0$ a $q = h_q > 0$ – tehdy dojde k prvnímu průchodu tělem cyklu našeho algoritmu, přičemž $h_p + h_q = i + 1$. Předpokládejme $h_p > h_q$; poté po prvním průchodu tělem cyklu budou hodnoty $p = h_p - h_q$ a $q = h_q$, přičemž nyní $p + q = h_p \leq h_p + h_q - 1 = i$.

Podle indukčního předpokladu (jelikož nyní $p + q \leq i$) tudíž výsledkem algoritmu pro vstupy $p = h_p - h_q$ a $q = h_q$ bude největší společný dělitel $NSD(h_p - h_q, h_q)$. Symetricky pro $h_p \leq h_q$ algoritmus vrátí $NSD(h_p, h_q - h_p)$. (Kde $NSD()$ krátce označuje největšího společného dělitele.)

Důkaz proto bude dokončen následujícím Lematem 11.13. □

Lema 11.13. Pro každá přirozená a, b platí $NSD(a, b) = NSD(a - b, b) = NSD(a, b - a)$.

Komentář: Všimněte si, že dělitelnost je dobře definována i na záporných celých číslech.

Důkaz: Ověříme, že $c = NSD(a - b, b)$ je také největší společný dělitel čísel a a b (druhá část je pak symetrická).

- * Jelikož číslo c dělí čísla $a - b$ a b , dělí i jejich součet $(a - b) + b = a$. Potom c je společným dělitelem a a b .
- * Naopak nechť d nějaký společný dělitel čísel a a b . Pak d dělí také rozdíl $a - b$. Tedy d je společný dělitel čísel $a - b$ a b . Jelikož c je největší společný dělitel těchto dvou čísel, nutně d dělí c a závěr platí. \square

Relativně rychlé odmocnění

Na závěr oddílu si ukážeme jeden netradiční krátký algoritmus a jeho analýzu a důkaz ponecháme zde otevřené. Dokážete popsat, na čem je algoritmus založen?

Algoritmus 11.14. *Celočíselná odmocnina.*

Pro dané přirozené číslo x vypočteme dolní celou část jeho odmocniny $\lfloor \sqrt{x} \rfloor$:

```
input x;
p ← x;  res ← 0;
while p > 0 do
    while (res + p)2 ≤ x do res ← res + p;
    p ← ⌊p/2⌋;
done
output res .
```

Poznámka: Zamysleli jste se, jaký mají algoritmy v tomto oddíle vlastně význam? Vždyť stejné úlohy jistě sami vyřešíte každý jednou jednoduchou `foreach` smyčkou. Podívejte se však (ale spouš velmi zhruba) na počet kroků, které zde uvedené algoritmy potřebují vykonat k získání výsledku, a srovnajte si to s počty kroků oněch „jednoduchých“ algoritmů.

Pro skutečně velká vstupní čísla zjistíte propastný rozdíl – s takovým „jednoduchým“ algoritmem, třeba `foreach i ← 1, . . . , b do res ← res · a mod m; done`, se pro obrovské hodnoty b výsledku nikdy nedočkáte, kdežto Algoritmus 11.11 stále poběží velmi rychle. (Spočítáte, jak rychle?) Obdobně je tomu u Algoritmu 11.14.

11.4 Dynamický algoritmus

Lekci zakončíme stručnou, ale velmi hezkou ukázkou tzv. *dynamického algoritmu*, který je znám pod jmény Floyd-Warshallův. Tento starý algoritmus je znám v mnoha podobách, my jej uvádíme jen v podobě výpočtu nejkratších cest v grafu, avšak prakticky tentýž algoritmus lze využít pro výpočet tranzitivního uzávěru relace i pro výpočet regulárního výrazu z konečného automatu a podobně.

Komentář: Klíčovou myšlenkou dynamických algoritmů je rozklad problému na podproblémy, jejichž řešení jsou postupně ukládána pro další možné použití. Metoda je obzvláště vhodná v případech, kdy rozložené podúlohy si jsou podobné a mohou se opakovat.

Metoda 11.15. *Dynamický výpočet všech nejkratších cest*

mezi vrcholy v grafu G na množině vrcholů $V(G) = \{v_0, v_1, \dots, v_{N-1}\}$.

- Na počátku nechť $d[i, j]$ udává 1 (případně váhu-délku hrany $\{v_i, v_j\}$), nebo ∞ pokud hrana mezi i, j není.
- Po kroku $t \geq 0$ nechť platí, že $d[i, j]$ udává délku nejkratší cesty mezi v_i, v_j , která užívá pouze vnitřní vrcholy z množiny $\{v_0, v_1, \dots, v_{t-1}\}$ (prázdné v $t = 0$).
- Při přechodu z kroku t na následující krok $t + 1$ upravujeme vzdálenost pro každou dvojici vrcholů v_i, v_j – jsou vždy pouhé dvě možnosti:
 - * Buď je cesta délky $d[i, j]$ z předchozího kroku t stále nejlepší (tj. nově povolený vrchol v_t nám nepomůže),
 - * nebo cestu vylepšíme spojením přes nově povolený vrchol v_t , čímž získáme menší vzdálenost $d[i, t] + d[t, j] \rightarrow d[i, j]$. (Nakreslete si obrázek.)
- Po N krocích úprav je výpočet hotov.

Výpočet všech nejkratších cest

Alternativně si zapíšeme postup této metody až překvapivě krátkým symbolickým algoritmem:

Algoritmus 11.16. Výpočet všech nejkratších cest; Floyd–Warshall

```

input  'Pole d[,] délek hran (nebo ∞) grafu G';
foreach t ← 0, 1, ..., N - 1 do
  foreach i ← 0, 1, ..., N - 1, j ← 0, 1, ..., N - 1 do
    d[i, j] ← min(d[i, j], d[i, t] + d[t, j]);
  done
done
output 'Matice vzdáleností dvojic d[,]'.

```

Poznámka: V praktické implementaci pro symbol ∞ použijeme velkou konstantu, třeba $\text{MAX_INT}/2$. (Nelze použít přímo MAX_INT , neboť by pak došlo k aritmetickému přetečení.)

Věta 11.17. Algoritmus 11.16 v poli $d[i, j]$ správně vypočte vzdálenost mezi každou dvojicí vrcholů v_i, v_j .

Důkaz povedeme matematickou indukcí podle řídicí proměnné t cyklu. Báze je snadná – na počátku kroku $t = 0$ udává $d[i, j]$ vzdálenost mezi v_i a v_j po cestách, které nemají vnitřní vrcholy (tj. pouze případně existující hranu).

Přejdeme-li na krok $t + 1$, musíme určit nejkratší cestu P mezi v_i a v_j takovou, že P používá (mimo v_i, v_j) pouze vrcholy $\{v_0, v_1, \dots, v_t\}$. Tuto nejkratší cestu P si hypoteticky představme: Pokud $v_t \notin V(P)$, pak $d[i, j]$ již udává správnou vzdálenost. Jinak $v_t \in V(P)$ a označíme P_1 podcestu v P od počátku v_i do v_t a obdobně P_2 podcestu od v_t do konce v_j . Podle indukčního předpokladu je pak délka P rovna $d[i, t] + d[t, j]$. Proto kratší z obou možností, tj. $\min(d[i, j], d[i, t] + d[t, j])$, je optimální hodnotou v kroku $t + 1$.

V kroku $t = N$ jsme hotovi se všemi vrcholy. □

Rozšiřující studium

V oblasti zjišťování a zdůvodňování správnosti algoritmů a počítačových programů existují (mimo heuristických metod jako ladění programů) rozsáhlé sofistikované přístupy

souhrnně nazývané souslovím formální verifikace. Pravděpodobně se s některými z nich setkáte při dalším teoretickém studiu, ale obecně leží daleko za hranicemi našeho studijního textu.

Na druhou stranu jsme se na začátku této lekce dotkli jiného zajímavého problému – zda algoritmus skončí svůj výpočet a kolik „kroků“ tento výpočet bude trvat. První podotázka je poměrně abstraktní a zabývá se jí disciplína vyčíslitelnosti, kdežto druhá podotázka počtu kroků výpočtu má velmi konkrétní dopady na praktickou použitelnost algoritmu: Však si sami vyzkoušejte, kolik kroků výpočtu bude trvat „hloupý“ rekurzivní výpočet Fibonacciho čísla! V obecnosti otázky počtu kroků nutných k dokončení výpočtu algoritmu studuje disciplína zvaná výpočetní složitost, která je jistým způsobem schopna předpovědět, zda na daný problém existuje rozumně rychlý přesný algoritmus nebo ne.

12 Nekonečné množiny, Zastavení algoritmu

Úvod

Hlubavého čtenáře může snadno napadnout kacírská myšlenka, proč se vlastně zabýváme dokazováním správnosti algoritmů a programů, když by to přece (snad?) mohl za nás dělat automaticky počítač samotný. Bohužel to však nejde a je hlavním cílem této lekce ukázat matematické důvody proč tomu tak je.

Konkrétně si dokážeme, že nelze algoritmicky rozhodnout, ani zda se daný algoritmus na svém vstupu zastaví nebo ne. Hlavními nástroji, které použijeme, budou nekonečné množiny a důkazová technika tzv. Cantorovy diagonály, která se ve velké míře používá právě v teoretické informatice. Touto lekcí se tak krátce vymaníme ze zajetí naivní teorie množin, která v této oblasti má své nepřekonatelné limity. (Pro zvědavé – obdobně, ale mnohem složitěji, lze dokázat že ani matematické důkazy nelze obecně algoritmicky konstruovat. . .)

Cíle

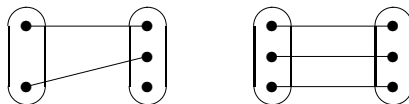
Zavedeme si ve zjednodušeném (a stále „poněkud naivním“) pohledu nekonečné množiny a na nich techniku důkazu Cantorovou diagonálou. Pak tuto klíčovou důkazovou techniku teoretické informatiky využijeme k důkazu algoritmické neřešitelnosti problému zastavení.

12.1 O nekonečných množinách a kardinalitě

Zatímco v naivní teorii jsme si velikost konečné množiny definovali jednoduše jako počet jejích prvků vyjádřený přirozeným číslem, pro nekonečné množiny je situace obtížná a mnohem méně intuitivní. Co nám třeba brání zavést pro velikost nekonečné množiny symbol ∞ ? Ponejvíce závažný fakt, že není „nekonečno“ jako „nekonečno“ (Věta 12.2)!

Právě proto si pro určení mohutnosti nekonečných množin musíme vypomoci následujícím příměrem: Velikosti dvou hromádek jablek dokážeme i bez počítání porovnat tak, že budeme z obou hromádek po řadě odebírat dvojice jablek (z každé jedno), až dokud první hromádka nezůstane prázdná – druhá hromádka pak je větší (nebo nejvýše rovna) té první.

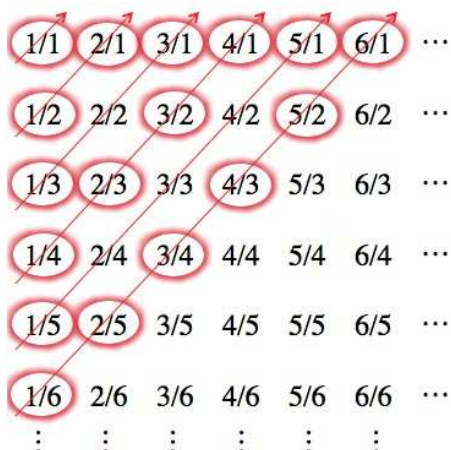
Definice: Množina A je „nejvýše tak velká“ jako množina B , právě když existuje injektivní funkce $f : A \rightarrow B$. Množiny A a B jsou „stejně velké“ právě když mezi nimi existuje bijekce. V případech nekonečných množin pak místo „velikosti“ mluvíme formálně o jejich *kardinalitě*.



Komentář: Uvedená definice kardinality množin funguje korektně i pro nekonečné množiny:

- * Například \mathbb{N} a \mathbb{Z} mají stejnou kardinalitu (jsou „stejně velké“, tzv. *spočetně nekonečné*).
- * Lze snadno ukázat, že i \mathbb{Q} je spočetně nekonečná, tj. existuje bijekce $f : \mathbb{N} \rightarrow \mathbb{Q}$, stejně

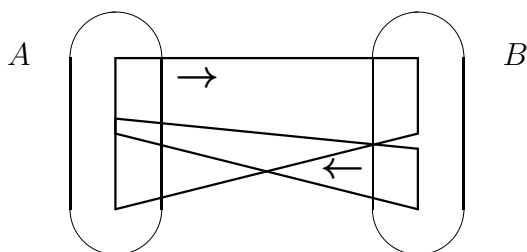
jako bijekce $h : \mathbb{N} \rightarrow \mathbb{N}^2$.



- * Existují ale i nekonečné množiny, které jsou „striktně větší“ než libovolná spočetná množina (příkladem je \mathbb{R} ve Větě 12.2).
- * Později dokážeme, že existuje nekonečná posloupnost nekonečných množin, z nichž každá je striktně větší než všechny předchozí.

Pro porovnávání velikostí množin někdy s výhodou využijeme následující přirozené, ale nelehké tvrzení (bez důkazu):

Věta 12.1. Pro libovolné dvě množiny A, B (i nekonečné) platí, že pokud existuje injekce $A \rightarrow B$ a zároveň i injekce $B \rightarrow A$, pak existuje bijekce mezi A a B .



Cantorova diagonála, aneb kolik je reálných čísel

Prvním klíčovým poznatkem ukazujícím na neintuitivní chování nekonečných množin je následující důkaz, který dal historicky vzniknout metodě tzv. *Cantorovy diagonály*.

Věta 12.2. Neexistuje žádné surjektivní zobrazení $g : \mathbb{N} \rightarrow \mathbb{R}$.

Důsledek 12.3. Neexistuje žádné injektivní (tudíž ani bijektivní) zobrazení $h : \mathbb{R} \rightarrow \mathbb{N}$. Neformálně řečeno, reálných čísel je striktně více než všech přirozených.

Důkaz (Věty 12.2 sporem): Nechť takové g existuje a pro zjednodušení se omezme jen na funkční hodnoty v intervalu $[0, 1)$ tj. ponechme z každé funkční hodnoty jen její nezápornou necelou část. Podle hodnot zobrazení g si takto můžeme „uspořádat“

dekadické zápisy všech reálných čísel v intervalu $[0, 1)$ po řádcích do tabulky:

$$\begin{array}{rcccccccccccc}
 g(0) = 0. & \mathbf{1} & 5 & 4 & 2 & 7 & 5 & 7 & 8 & 3 & 2 & 5 & \dots \\
 g(1) = 0. & & \mathbf{4} & & & & & & & & & & \dots \\
 g(2) = 0. & & & \mathbf{1} & & & & & & & & & \dots \\
 g(3) = 0. & & & & \mathbf{3} & & & & & & & & \dots \\
 g(4) = 0. & & & & & \mathbf{9} & & & & & & & \dots \\
 \vdots & \vdots & & & & & \ddots & & & & & & \dots
 \end{array}$$

Nyní sestrojíme číslo $\alpha \in [0, 1)$ následovně; jeho i -tá číslice za desetinnou čárkou bude 1, pokud v i -tém řádku tabulky na diagonále není 1, jinak to bude 2. V našem příkladě $\alpha = 0.21211\dots$

Kde se naše číslo α v tabulce nachází? (Nezapomeňme, g byla surjektivní, takže α někde musí být.) Konstrukce však ukazuje, že α se od každého čísla v tabulce liší na aspoň jednom desetinném místě, a to je spor. (Až na drobný technický detail s rozvojem $\dots 9$.) \square

12.2 „Naivní“ množinové paradoxy

Analogickým způsobem k Větě 12.2 lze dokázat následovné zobecnění vyjadřující se o jakékoliv množině a jí přiřazené striktně větší množině.

Věta 12.4. *Nechť M je libovolná množina. Pak existuje injektivní zobrazení $f : M \rightarrow 2^M$, ale neexistuje žádné bijektivní zobrazení $g : M \rightarrow 2^M$.*

Důkaz: Dokážeme nejprve existenci f . Stačí ale položit $f(x) = \{x\}$ pro každé $x \in M$. Pak $f : M \rightarrow 2^M$ je zjevně injektivní.

Neexistenci g dokážeme sporem. Předpokládejme tedy naopak, že existuje bijekce $g : M \rightarrow 2^M$. Uvažme množinu $K \subseteq M$ definovanou takto:

$$K = \{x \in M \mid x \notin g(x)\}.$$

Jelikož g je bijektivní a $K \in 2^M$, musí existovat $y \in M$ takové, že $g(y) = K$. Nyní rozlišíme dvě možnosti:

- $y \in g(y)$. Tj. $y \in K$. Pak ale $y \notin g(y)$ z definice K , což je spor.
- $y \notin g(y)$. To podle definice K znamená, že $y \in K$, tj. $y \in g(y)$, spor. \square

Komentář: Dvě navazující poznámky.

- Technika použitá v důkazech Vět 12.2 a 12.4 se nazývá *Cantorova diagonální metoda*, nebo také zkráceně *diagonalizace*.

Konstrukci množiny K lze znázornit pomocí následující tabulky:

	a	b	c	d	\dots
$g(a)$	\checkmark	$-$	$-$	\checkmark	\dots
$g(b)$	\checkmark	$-$	$-$	\checkmark	\dots
$g(c)$	$-$	\checkmark	$-$	\checkmark	\dots
$g(d)$	$-$	$-$	\checkmark	\checkmark	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\ddots

Symbol \surd resp. \dashv říká, že prvek uvedený v záhlaví sloupce patří resp. nepatří do množiny uvedené v záhlaví řádku. Tedy např. $d \in g(b)$ a $a \notin g(d)$. Množina K poté obsahuje ty diagonální prvky označené \dashv , tj. „převrací“ význam diagonály.

- Z toho, že nekonečna mohou být „různě velká“, lze lehce odvodit řadu dalších faktů. V jistém smyslu je např. množina všech problémů větší než množina všech algoritmů (obě množiny jsou nekonečné), a proto nutně existují problémy, které *nejsou algoritmicky řešitelné*.

Cantorův paradox

Naivní teorie množin, jak jsme si ji uvedli i v tomto předmětu, trpí mnoha neduhy a nepřesnostmi, které vyplynou na povrch především při „neopatrné manipulaci“ s nekonečnými množinami. Abychom se těmto „neopatrnostem“ vyhnuli bez přílišné formalizace, dva základní z těchto paradoxů si nyní ukážeme.

Příklad 12.5. *Uvážíme-li nyní nekonečnou posloupnost množin*

$$A_1, A_2, A_3, A_4, \dots$$

kde $A_1 = \mathbb{N}$ a $A_{i+1} = 2^{A_i}$ pro každé $i \in \mathbb{N}$, je vidět, že všechny množiny jsou nekonečné a každá je striktně větší než libovolná předchozí.

Kde však v tomto řazení kardinalit bude „množina všech množin“? Na tuto otázku, jak sami asi cítíte, nelze podat odpověď. Co to však znamená? \square

- * Takto se koncem 19. století objevil první *Cantorův paradox* nově vznikající teorie množin.
- * Dnešní moderní vysvětlení paradoxu je jednoduché, prostě „množinu všech množin“ nelze definovat, taková v matematice neexistuje.

Brzy se však ukázalo, že je ještě *mnohem hůř*...

Russelův paradox

Fakt: Není pravda, že *každý soubor prvků* lze považovat za množinu.

- * Necht' $X = \{M \mid M \text{ je množina taková, že } M \notin M\}$. Platí $X \in X$?
 - Ano. Tj. $X \in X$. Pak ale X splňuje $X \notin X$.
 - Ne. Pak X splňuje vlastnost $X \notin X$, tedy X je prvkem X , tj., $X \in X$.
- * Obě možné odpovědi vedou ke sporu. X tedy nelze prohlásit za množinu. Jak je ale něco takového vůbec možné?

Komentář: Vidíte u Russelova paradoxu podobnost přístupu s Cantorovou diagonalizací? Russelův paradox se objevil začátkem 20. století a jeho „jednoduchost“ zasahující úplně základy matematiky (teorie množin) si vynutila hledání seriózního rozřešení a rozvoj formální matematické logiky.

Pro ilustraci tohoto paradoxu, slyšeli jste už, že „v malém městečku žije holič, který holí právě ty muže městečka, kteří se sami neholí“? Zmíněné paradoxy naivní teorie množin zatím v tomto kurzu nerozřešíme, ale zapamatujeme si, že většina matematických a infor-matických disciplín vystačí s „*intuitivně bezpečnými*“ množinami.

12.3 Algoritmická neřešitelnost problému zastavení

Výše vysvětlené myšlenky diagonalizace a principů základních paradoxů naivní teorie množin sice vypadají „velmi matematicky“. Přesto je však téměř beze změny lze aplikovat i na bytostně inforatickou otázku, zda lze algoritmicky poznat, pro které vstupy se daný program vůbec zastaví. Negativní odpověď na tuto otázku je jedním z fundamentálních výsledků informatiky a přitom má překvapivě krátký a čistý důkaz diagonalizací.

Fakt: Uvědomme si (velmi neformálně) několik základních myšlenek.

- * Program v každém programovacím jazyce je konečná posloupnost složená z *konečně mnoha* symbolů (písmena, číslice, mezery, speciální znaky, apod.) Necht' Σ je množina všech těchto symbolů. Množina všech programů je tedy jistě podmnožinou množiny $\bigcup_{i \in \mathbb{N}} \Sigma^i$, která je spočetně nekonečná. Existuje tedy bijekce f mezi množinou \mathbb{N} a množinou všech programů. Pro každé $j \in \mathbb{N}$ označme symbolem P_j program $f(j)$. Pro každý program P tedy existuje $i \in \mathbb{N}$ takové, že $P = P_i$.
- * Každý možný vstup každého možného programu lze zapsat jako *konečnou posloupnost* symbolů z konečné množiny Γ . Množina všech možných vstupů je tedy spočetně nekonečná a existuje bijekce g mezi množinou \mathbb{N} a množinou všech vstupů. Pro každé $j \in \mathbb{N}$ označme symbolem V_j vstup $g(j)$.
- * Předpokládejme, že existuje program *Halt*, který pro dané $i, j \in \mathbb{N}$ zastaví s výstupem *ano/ne* podle toho, zda P_i pro vstup V_j zastaví, nebo ne.
- * Tento předpoklad dále dovedeme ke sporu dokazujícím, že problém zastavení nemá algoritmické řešení.

Věta 12.6. *Neexistuje program Halt, který by pro vstup (P_i, V_j) správně rozhodl, zda se program P_i zastaví na vstupu V_j .*

Důkaz: Sporem uvažme program *Diag* s následujícím kódem:

```
input k;
if Halt(k,k) = ano then while true do ; done
```

(Program *Diag*(k) má na rozdíl od *Halt* jen jeden vstup k , což bude důležité.)

Fungování programu *Diag* lze znázornit za pomoci následující tabulky:

	P_0	P_1	P_2	P_3	...
V_0	✓	–	–	✓	...
V_1	✓	–	–	✓	...
V_2	–	✓	–	✓	...
V_3	–	–	✓	✓	...
\vdots	\vdots	\vdots	\vdots	\vdots	\ddots

Symbol ✓ resp. – říká, že program uvedený v záhlaví sloupce zastaví resp. nezastaví pro vstup uvedený v záhlaví řádku. Program *Diag* „zneguje“ diagonálu této tabulky.

Podle našeho předpokladu (*Diag* je program a posloupnost P_i zahrnuje všechny programy) existuje $j \in \mathbb{N}$ takové, že $Diag = P_j$. Zastaví *Diag* pro vstup V_j ?

- *Ano*. Podle kódu *Diag* pak ale tento program vstoupí do nekonečné smyčky, tedy nezastaví.

– Ne. Podle kódu *Diag* pak ale *if* test neuspěje, a tento program tedy zastaví.
Předpoklad existence programu *Halt* tedy vede ke sporu. □

Komentář: Otázkami algoritmické (ne)řešitelnosti problémů se zabývá *teorie vyčíslitelnosti*. Metoda diagonalizace se také často využívá v *teorii složitosti* k důkazu toho, že dané dvě složitostní třídy jsou různé.

Rozšiřující studium

Látka této lekce zabrousila až do teoretických hlubin matematické logiky a teorie množin. Další studium v těchto oblastech se dá očekávat hlavně u studentů specificky zaměřených teoretickým směrem (a mířících spíše do akademické než aplikační sféry), zajímajících se o matematiku samotnou nebo o teorii vyčíslitelnosti. Proto také uvedené pokročilé poznatky Lekce 12 nebudou vyžadovány u zkoušky tohoto předmětu.

Závěrem

Gratulujeme všem, kteří se naším nelehkým učebním textem „prokousali“ až sem, a přejeme mnoho úspěchů v dalším studiu informatiky.

Konečně znovu připomínáme, že nedílnou součástí našeho studijního textu je Interaktivní osnova předmětu IB000 v IS MU a v ní přiložené online odpovědníky určené k procvičování přednesené látky.

<http://is.muni.cz/el/1433/podzim2015/IB000/index.qwarp>