

8 Dokazování vlastností algoritmů

Jak jste asi již poznali, umění programovat není zdaleka jen o tom naučit se syntaxi programovacího jazyka, ale především o schopnosti vytvářet a správně formálně zapisovat algoritmy. Přitom situace, kdy programátorem zapsaný algoritmus počítá něco trochu jiného, než si programátor představuje, je určitě nejčastější programátorskou chybou, o to zákeřnější, že ji žádný „chytrý“ překladač nemůže odhalit.



Proto již na počátku (seriózního) studia informatiky je dobré klást důraz na správné chápání zápisu algoritmů i na důkazy jejich vlastností a správnosti. □

Stručný přehled lekce

- * Jakými postupy ověřovat, že počítačový program „správně funguje“?
- * Použití matematické indukce k dokazování vlastností algoritmů.
- * Několik konkrétních algoritmů a jejich důkazů.

8.1 O „správnosti“ programů

Jak se máme přesvědčit, že je daný program „správný“? □

- Co třeba ladění programů? □
Jelikož počet možných vstupních hodnot je (v principu) neohraničený, **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 tudíž vedou k různým výsledkům. (Nelze je 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. □
Narůstající složitost programových systémů a zvýšené požadavky na jejich bezpečnost si vynucují vývoj „spolehlivých“ formálních verifikačních metod.

Připomenutí našeho formálního popisu algoritmů

- *Proměnné* nebudeme deklarovat ani typovat, pole odlišíme závorkami `p[]`.
- *Přiřazení* hodnoty zapisujeme `a ← b`, případně `a:=b`, ale nikdy **ne** `a=b`.
- Jako elem. 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** konečnou 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**.

8.2 Jednoduché indukční dokazování

Příklad 8.1. Zjistěte, kolik znaků 'x' v závislosti na celočíselné hodnotě n vstupního parametru n vypíše následující algoritmus.

Algoritmus 8.2.

```
foreach  $i \leftarrow 1, 2, 3, \dots, n-1, n$  do
    foreach  $j \leftarrow 1, 2, 3, \dots, i-1, i$  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 2.6), že je celkem $\frac{1}{2}n(n+1)$.

□ Budeme tedy dokazovat následující tvrzení:

Věta. Pro každé přir. n Algoritmus 8.2 vypíše právě $\frac{1}{2}n(n+1)$ znaků 'x'.

Algoritmus 8.2.

```
foreach  $i \leftarrow 1, 2, 3, \dots, n-1, n$  do
    foreach  $j \leftarrow 1, 2, 3, \dots, i-1, i$  do
        vytiskni 'x';
    done
done □
```

Věta. Pro každé přir. n Algoritmus 8.2 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 máme dokázat. □

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. □

Příklad 8.3. Zjistěte, kolik znaků 'z' v závislosti na celočíselné hodnotě n vstupního parametru n vypíše následující algoritmus.

Algoritmus 8.4.

```
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 $0, 1, 3, 7, 15, \dots$ □ Na základě toho již není obtížné „uhodnout“, že počet 'z' bude (asi) obecně určen vztahem $2^n - 1$. Toto je však třeba dokázat! □

Jak záhy zjistíme, matematická indukce na naše tvrzení přímo „nezabírá“, ale mnohem lépe se nám povede s následujícím přirozeným zesílením dokazovaného tvrzení:

Algoritmus 8.4.

```
st ← "z";  
foreach i ← 1,2,3,...,n-1,n do  
    vytiskni řetězec st;  
    st ← st+st;   (zřetězení dvou kopií st za sebou)  
done
```

Věta. Pro každé přirozené n Algoritmus 8.4 vypíše právě $2^n - 1$ znaků 'z' a proměnná st bude na konci 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 znaků 'z', což máme dokázat.

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 $2 \cdot 2^{n_0} = 2^n$ znaků 'z'. □

8.3 Algoritmy pro relace

Relace jsou velice vhodnou strukturou pro algoritmické zpracování.

Algoritmus 8.5. Symetrický uzávěr.

Pro danou relaci R na n -prvkové množině $A = \{a_1, a_2, \dots, a_n\}$ vytvoříme její symetrický uzávěr \overleftrightarrow{R} takto:

```
 $\overleftrightarrow{R} \leftarrow R;$   
foreach  $i \leftarrow 1, 2, \dots, n-1, n$  do  
  foreach  $j \leftarrow 1, 2, \dots, n-1, n$  do  
    if  $(a_i, a_j) \in R \wedge (a_j, a_i) \notin R$  then  $\overleftrightarrow{R} \leftarrow \overleftrightarrow{R} \cup \{(a_j, a_i)\};$   
  done  
done  $\square$ 
```

Důkaz: Zde není důkaz vůbec obtížný. Relace \overleftrightarrow{R} je zřejmě symetrická, neboť (vnitřní) tělo cyklu pro všechny dvojice $(a_i, a_j) \in R$ přidá i (a_j, a_i) . Z druhé strany všechny dvojice „přidané“ v $\overleftrightarrow{R} \setminus R$ musí být obsaženy podle definice symetrické relace, takže \overleftrightarrow{R} je skutečně symetrickým uzávěrem podle definice uzávěru relace. \square

Algoritmus 8.6. Tranzitivní uzávěr.

Pro danou relaci R na n -prvkové množině $A = \{a_1, a_2, \dots, a_n\}$ vytvoříme její tranzitivní uzávěr R^+ takto:

```
 $R^+ \leftarrow R;$   
foreach  $k \leftarrow 1, 2, \dots, n-1, n$  do  
  foreach  $i \leftarrow 1, 2, \dots, n-1, n; j \leftarrow 1, 2, \dots, n-1, n$  do  
    if  $(a_i, a_k) \in R^+ \wedge (a_k, a_j) \in R^+$  then  
      if  $(a_i, a_j) \notin R^+$  then  $R^+ \leftarrow R^+ \cup \{(a_i, a_j)\};$   
    fi  
  done  
done  $\square$ 
```

Jak by se dala dokázat správnost popsaného algoritmu? Přímá aplikace indukce podle n nevypadá přínosně. . . (Zkuste si sami!) \square

Nejkratší cesta k cíli vede použitím indukce (podle proměnné k vnějšího cyklu) na vhodně zesíleném tvrzení. Pro jeho formulaci si definujeme, že relace S na A je **k -částečně tranzitivní**, pokud pro libovolná i, j a pro $\ell \leq k$ platí, že z $(a_i, a_\ell), (a_\ell, a_j) \in S$ vyplývá $(a_i, a_j) \in S$.

Věta. Po každých $k \geq 0$ iteracích vnějšího cyklu Algoritmu 8.6 aktuální hodnota relace R^+ udává k -částečně tranzitivní uzávěr relace R na A . \square

Důkaz: Báze indukce pro $k = 0$ jasně platí, neboť věta v tom případě nic neříká. \square

Předpokládejme nyní, že tvrzení platí pro nějaké $k_0 \geq 0$ a dokažme jej i pro $k = k_0 + 1$. Zřejmě stačí uvažovat případ $k_0 < n$. Každá dvojice (a_i, a_j) přidaná do R^+ uvnitř cyklů musí náležet do k -částečně tranzitivního uzávěru podle definice. Zbývá zdůvodnit, proč každá dvojice (a_i, a_j) náležející do k -částečně tranzitivního uzávěru, ale ne do k_0 -částečně tranzitivního uzávěru, bude do R^+ v k -té iteraci přidána. \square

Není těžké ověřit, že (a_i, a_j) náleží do k -částečně tranzitivního uzávěru, právě když v relaci R nalezneme takovou cestu „po šipkách“ z a_i do a_j , která přechází pouze přes prvky a_ℓ kde $\ell \leq k$. V naší situaci vyplývá, že taková cesta musí použít i prvek a_k (jen jednou!), a proto (a_i, a_k) i (a_k, a_j) náleží do k_0 -částečně tranzitivního uzávěru R . V k -té iteraci tudíž bude příslušná `if` podmínka splněná a (a_i, a_j) bude přidána do R^+ . \square

Dokazování konečnosti algoritmu

Všimněte si, že jsme se zatím v důkazech vůbec nezamýšleli nad tím, zda náš algoritmus vůbec **skončí**. (To jistě není samozřejmé a důkaz konečnosti je nutno v obecnosti podávat!) □

Prozatím jsme však ukazovali algoritmy využívající jen **foreach** cykly, přitom podle naší konvence obsahuje **foreach** cyklus předem danou konečnou množinu hodnot pro řídící proměnnou, neboli náš **foreach** cyklus vždy musí skončit. Ale už v příštím algoritmu využijeme **while** **cyklus**, u kterého vůbec není jasné kdy a jestli skončí, a tudíž bude potřebný i důkaz konečnosti. □

Metoda 8.7. Důkaz konečnosti.

Máme-li za úkol dokázat, že algoritmus skončí, postupujeme nejlépe následovně:

- *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.*

Pozor, naše „parametry“ vůbec nemusejí být proměnnými v programu.

Algoritmus 8.8. Cykly permutace.

Pro danou permutaci π na n -prvkové neprázdné množině $A = \{1, 2, \dots, n\}$ vypíšeme její cykly (viz Oddíl 6.4) takto:

```
U ← {1, 2, ..., n};
while U ≠ ∅ do
    x ← min(U);    (nejmenší prvek množiny)
    začínáme výpis cyklu '⟨';
    while x ∈ U do
        vytiskneme x;
        U ← U \ {x};    x ← π(x);
    done
    ukončíme výpis cyklu '⟩';
done
```

Jak dokážeme správnost tohoto algoritmu? □

Opět platí, že přímá aplikace indukce podle n nepřinese nic podstatného. Důkaz si tentokrát rozdělíme na dvě části (podle dvou `while` cyklů). Všimněte se navíc, že tentokrát je nezbytnou součástí důkazu správnosti algoritmu i důkaz, že oba `while` cykly vždy skončí.

```
while  $U \neq \emptyset$  do
    .....
done
```

Věta. Za předp., že vnitřní **while** cyklus pro jakoukoliv poč. volbu x skončí, vypíše cyklus permutace π obsahující x a odebere všechny prvky tohoto cyklu z množiny U , Algoritmus 8.8 vždy skončí se správným výsledkem. \square

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. \square

Podle Věty 6.5 se každá permutace dá zapsat jako složení disjunktních 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

Vidíte, že v tomto důkaze indukcí je indukční krok zcela triviální a důležitý je zde především základ indukce?

```
while  $x \in U$  do
    vytiskneme  $x$ ;
     $U \leftarrow U \setminus \{x\}$ ;    $x \leftarrow \pi(x)$ ;
done
```

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 . \square

Důkaz: Zde přímo zopakujeme argument důkazu Věty 6.5: 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

8.4 Zajímavé algoritmy aritmetiky

Například umocňování na velmi vysoké exponenty je podkladem RSA šifry:

Algoritmus 8.9. 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. $c = a^b \bmod m$.

```
c ← 1;
while b > 0 do
    if b mod 2 > 0 then c ← (c·a) mod m;
    b ← ⌊b/2⌋;    a ← (a·a) mod m;
done
výsledek c; □
```

Zde použijeme k důkazu správnosti algoritmu indukci podle délky ℓ binárního zápisu čísla b .

Věta. Algoritmus 8.9 skončí a vždy správně vypočte hodnotu mocniny $c = a^b \bmod m$.

```

c ← 1;
while b > 0 do
    if b mod 2 > 0 then c ← (c·a) mod m;
    b ← ⌊b/2⌋;    a ← (a·a) mod m;
done
výsledek c;

```

Důkaz: Báze indukce je pro $\ell = 1$, kdy $b = 0$ nebo $b = 1$. Přitom pro $b = 0$ se cyklus vůbec nevykoná a výsledek je $c = 1$. Pro $b = 1$ se vykoná jen jedna iterace cyklu a výsledek je $c = a \bmod m$. \square

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 bude $a' = a^2$, $b' = \lfloor b/2 \rfloor$ a $c' = (a^{b \bmod 2}) \bmod m$. Tudíž délka binárního zápisu b' bude jen ℓ_0 a podle indukčního předpokladu zbylé iterace algoritmu skončí s výsledkem

$$c = c' \cdot a'^{b'} \bmod m = (a^{b \bmod 2} \cdot a^{2\lfloor b/2 \rfloor}) \bmod m = a^b \bmod m.$$

\square

Na závěr lekce si ukážeme jeden netradiční krátký algoritmus a jeho analýzu a důkaz ponecháme zde otevřené. Dokážete popsat, na čem je algoritmus založen?

Algoritmus 8.10. Celočíselná odmocnina.

Pro dané přirozené číslo x vypočteme dolní celou část jeho odmocniny $r = \lfloor \sqrt{x} \rfloor$.

```
p ← x;   r ← 0;
while p > 0 do
    while (r + p)2 ≤ x do r ← r + p;
    p ← ⌊p/2⌋;
done
výsledek r; □
```