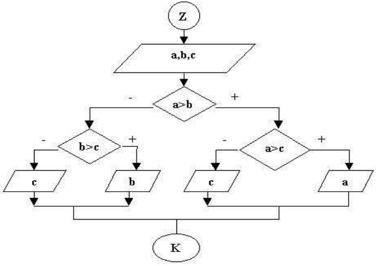
11 Formalizace a důkazy pro algoritmy

Je faktem, že 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.



Stručný přehled lekce

* Jednoduchá formalizace pojmu algoritmus.
* Jak dokazovat vlastnosti a správnost algoritmů.
* Indukce při dokazování algoritmů.

11.1 Formální popis algoritmu

Před samotným závěrem kurzu si položme otázku, co je vlastně algoritmus?

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 neprocedurální modely. □

Konvence 11.1. Zjednodušeně zde *algoritmem* rozumíme konečnou posloupnost elementárních výpočetních *kroků*, ve které každý další krok vhodně 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. □

Vidíte, jak blízké si jsou konstruktivní matematické důkazy a algoritmy v našem pojetí? Jedná se nakonec o jeden ze záměrů našeho přístupu. . .

Ukázka algoritmického zápisu

Příklad 11.2. *Zápis algoritmu pro výpočet průměru daného pole* `a[]` *s* `n` *prvky.*

- Inicializujeme `sum` $\leftarrow$ `0` ;
- postupně pro `i=0,1,2,...,n-1` provedeme
  - $\ast$ `sum` $\leftarrow$ `sum+a[i]` ;
- vypíšeme podíl `(sum/n)` . □

□

Ve „vyšší úrovni" formálnosti se totéž dá zapsat jako:

Algoritmus 11.3. Průměr z daného pole `a[]` s `n` prvky.

```
input  pole a[]  délky n ≥ 1;
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 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 množinu hodnot pro přiřazování do řídící 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 řídících struktur (což jsou `if`, `foreach`, `while`).
- Pokud je to dostatečně jasné, elementární operace nebo podmínky můžeme i ve formálním zápise popsat běžným jazykem.

Co počítá následující algoritmus?

Příklad 11.4. *Je dán následující symbolicky zapsaný algoritmus. Co je jeho výstupem v závislosti na vstupech a, b?*

Algoritmus 11.5.

```
input a, b;
res ← 7;
foreach  i ← 1,2,...,b-1,b  do
    res ← res+a+2·b+8;
done
output res .
```

□

Vypočítáme hodnoty výsledku res v počátečních iteracích cyklu:

$$\begin{aligned} b=0: &\quad res = 7, \\ b=1: &\quad res = 7 + a+2b+8, \\ b=2: &\quad res = 7 + (a+2b+8) + (a+2b+8), \ldots \end{aligned}$$ □

Co dále? Výčet hodnot naznačuje pravidelnost a závěr, že obecný výsledek po b iteracích cyklu bude mít hodnotu

$$res = 7 + b(a+2b+8) = ab + 2b^2 + 8b + 7 .$$ □

11.2 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) 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 vedou k různým výsledkům.
  Nelze je tudíž ani rozumně ladit...

- 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.
  * Narůstající složitosti programových systémů si pak vynucují vývoj jiných „spolehlivých“ formálních verifikačních metod.

Mimochodem, co to vlastně znamená „počítat správně“?

Ukázka formálního důkazu algoritmu

Příklad 11.6. *Je dán následující symbolicky zapsaný algoritmus. Dokažte, že jeho výsledkem je „výměna" vstupních hodnot a, b.*

Algoritmus 11.7.

```
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:

- $a = h_a,\ b = h_b$,
- $a \leftarrow a + b = h_a + h_b, \quad b = h_b$, □
- $a = h_a + h_b, \quad b \leftarrow a - b = h_a + h_b - h_b = \underline{h_a}$, □
- $a \leftarrow a - b = h_a + h_b - h_a = \underline{h_b}, \quad b = h_a$,

Tímto jsme s důkazem hotovi. □

Jednoduché indukční dokazování

Pro dokazování algoritmů se jeví nejvhodněji matematická indukce, která je „jako stvořená“ pro formální uchopení opakovaných sekvencí v algoritmech. □

Příklad 11.8. *Dokažte, že násl. algoritmus navrátí výsledek* $ab + 2b^2 + 8b + 7$.

Algoritmus 11.9.

```
input a, b;
res ← 7;
foreach  i ← 1,2,...,b-1,b  do
    res  ←  res+a+2·b+8;
done
output res .
```

□

V prvé řadě si z důvodu formální přesnosti přeznačíme mez cyklu v algoritmu na `foreach i ← 1,2,...,c do ..` (kde $c = b$). Poté postupujeme přirozeně indukcí podle počtu c iterací cyklu (už nezávisle na vstupní hodnotě b); dokazujeme, že výsledek výpočtu algoritmu bude

$$res = (a + 2b + 8)c + 7 = ac + 2bc + 8c + 7\,.$$

Algoritmus .

```
input a, b;
res ← 7;
foreach  i ← 1,2,...,c-1,c  do
    res  ←  res+a+2·b+8;
done
output res .
```

Tvrzení: $res = ac + 2bc + 8c + 7$. □

Důkaz indukcí podle c (= počtu iterací cyklu „`foreach i`“):

∗ 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

$$res \leftarrow res + a + 2b + 8 = ac + 2bc + 8c + 7 \; + \; a + 2b + 8 =$$
$$= a(c+1) + 2b(c+1) + 8(c+1) + 7\,.$$

Důkaz indukcí je tím hotov. □

11.3 Rekurzivní algoritmy

* Rekurentní vztahy posloupností, stručně uvedené v Oddíle 5.1, 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 (menší) vstupní hodnoty. □
* U takových algoritmů je zvláště důležité kontrolovat jejich správnost a také praktickou proveditelnost (časovou i paměťovou). □

Příklad 11.10. *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 $x! = x \cdot (x-1) \cdot \dots \cdot 2 \cdot 1$. □

Fibonacciho čísla

Pro jiný příklad rekurze se vrátíme k Oddílu 5.1, kde byla zmíněna známá Fibonacciho posloupnost $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$. Ve skutečnosti tuto posloupnost budeme uvažovat již od nultého členu, tj. jako $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$. □

Algoritmus 11.11. 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.
```
□

Správnost Algoritmu 11.11 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...

Co třeba $fibonacci(40)$ nebo $fibonacci(50)$?

Příklad 11.12. *Nerekurzivní algoritmus pro Fibonacciho čísla.*

Dokažte, že následující algoritmus pro každé přirozené n počítá tutéž hodnotu jako rekurentní funkce `fibonacci(n)` v Algoritmu 11.11.

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] .
```

Indukcí budeme dokazovat, že po i-té iteraci cyklu algoritmu bude vždy platit $b[i] = fibonacci(i)$: Co se týče báze indukce, toto vyplývá z úvodního přiřazení.

* Pro libovolné $i \geq 1$ předpokládáme platnost indukčního předpokladu $b[j] = fibonacci(j)$ pro $j \in \{i, i-1\}$.
* V $(i+1)$-ní iteraci cyklu nastane

  $$b[i+1] \leftarrow b[i]+b[i-1] = fibonacci(i)+fibonacci(i-1) = fibonacci(i+1),$$

  přesně podle definice. □

11.4 Přehled technik důkazu indukcí

- Doposud zde byla matematická indukce představována ve své přímočaré formě, kdy dokazované tvrzení obvykle přímo nabízelo celočíselný parametr, podle nějž bylo potřebné indukci vést. □
- Indukční krok pak prostě zpracoval přechod „$n = i \leadsto n = i + 1$“. □
- To však u dokazování správnosti algoritmů typicky neplatí a naším 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

Příklad 11.13. *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 $y + \cdots + y$, dokud se x nesníží na nulu. Poté odhadneme:

Věta. Pro každé $x, y \in \mathbb{N}$ Algoritmus 11.13 vypočítá hodnotu $res = x \cdot y$.

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:

```
while  x>0  do
    res ← res+y;   x ← x-1;
done
```

Důkaz: Budiž $h_y \in \mathbb{N}$ libovolné ale pro další úvahy pevné.

Dokážeme, že pro každý vst. $x \in \mathbb{N}$ je výsledkem výpočtu hodnota $r_0 + x \cdot h_y$, kde h_y byla hodnota vstupu `y` a r_0 byla hodnota v proměnné `res` na začátku uvažovaného výpočtu (pro potřeby indukce, $r_0 = 0$ na úplném začátku). □

- 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 $x = i \in \mathbb{N}$ 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 `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 (a ještě položíme $r_0 = 0$). □

Indukce k součtu parametrů

Příklad 11.14. *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 $a, b \ldots$ □

Věta. Pro každé parametry $m, n \in \mathbb{N}$ je výsledkem výpočtu funkce `kombinacni(m,n)` hodnota $res = \binom{m+n}{m}$ (kombinační číslo) – počet všech m-prvkových podmnožin $(m+n)$-prvkové množiny.

```
res ← 1;
if m>0 ∧ n>0 then
    res ← kombinacni(m-1,n)+kombinacni(m,n-1);
fi
```

$$res \stackrel{?}{=} \binom{m+n}{m}$$

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.

```
res ← 1;
if m>0 ∧ n>0 then
    res ← kombinacni(m-1,n) + kombinacni(m,n-1);
fi
```

- Indukční krok přechází na součet $i+1 = m+n$ pro $m, n > 0$.
  Nyní je podmínka algoritmu splněna a vykonají se rekurentní volání

  `kombinacni(m-1,n) + kombinacni(m,n-1).` □

  Rekurentní volání se vztahují k výběru podmnožin nosné množiny, která má $m-1+n = m+n-1 = i$ prvků, například $M = \{1, 2, \ldots, i\}$. Výsledkem tedy je, podle indukčního předpokladu pro součet i, počet všech $(m-1)$-prvkových plus m-prvkových podmnožin množiny M. □

  Kolik je m-prvkových podmn. $(i+1)$-prvkové množiny $M' = M \cup \{i+1\}$?
  Pokud ze všech těchto podmnožin odebereme prvek $i+1$, dostaneme právě
  - m-prvkové podmnožiny (z těch neobsahujících prvek $i+1$)

  plus
  - $(m-1)$-prvkové podmnožiny (z těch původně obsahujících $i+1$). □

  A to je v součtu rovno `kombinacni(m-1,n) + kombinacni(m,n-1)`, jak jsme měli dokázat. □

Zesílení dokazovaného tvrzení

Příklad 11.15. *Zjistěte, kolik znaků* `'z'` *v závislosti na celočíselné hodnotě* n *vstupního parametru* `n` *vypíše následující algoritmus.*

Algoritmus 11.16.

```
input  n;
st ←"z";
foreach  k ←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 $n = 0, 1, 2, 3, 4 \ldots$, postupně dostaneme počty `'z'` jako $0, 1, 3, 7, 15 \ldots$. □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 .

```
st ← "z";
foreach k ← 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 11.16 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 $0 = 2^0 - 1$ znaků `'z'`, což bylo třeba dokázat. Mimo to proměnná `st` iniciálně obsahuje $1 = 2^0$ znak `'z'`. □

Nechť tedy tvrzení platí pro jakékoliv $n = i \geq 0$ a uvažme vstup $n := i+1$. Podle indukčního předpokladu po prvních i iteracích bude vytištěno $2^i - 1$ znaků `'z'` a proměnná `st` bude obsahovat řetězec 2^i znaků `'z'`. V poslední iteraci cyklu (pro $\mathtt{k} \leftarrow \mathtt{n} = i + 1$) vytiskneme z proměnné `st` dalších 2^i znaků `'z'` a poté délku řetězce `st` „zdvojnásobíme“. □

Proto po n iteracích bude vytištěno celkem $2^i - 1 + 2^i = 2^{i+1} - 1 = 2^n - 1$ znaků `'z'` a v proměnné `st` bude uloženo $2 \cdot 2^i = 2^{i+1} = 2^n$ znaků `'z'`. □

11.5 Dodatek: Zajímavé algoritmy aritmetiky

Euklidův algoritmus

Algoritmus 11.17. Euklidův pro největšího společného dělitele.

```
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 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.

```
while  p>0 ∧ q>0  do
    if  p>q  then  p ← p-q;
    else  q ← q-p;
done
```

- Ve skutečnosti je zase výhodné uvažovat rozšířenou bázi, která zahrnuje i případy, kdy jen jedno z p, q je nulové (což je ukončovací podmínka cyklu). Pak výsledek $p+q$ bude roven tomu nenulovému z obou sčítanců, což je v tomto případě zároveň jejich největší společný dělitel.

- Indukční krok. Uvažme vstupy $p := h_p$ a $q := h_q$, přičemž $h_p + h_q = i + 1$ a $h_p > 0$ a $h_q > 0$ – tehdy dojde k prvnímu průchodu tělem cyklu.
  * Předp. $h_p > h_q$; poté po prvním průchodu tělem cyklu budou hodnoty $p = h_p - h_q$ a $q = h_q$, což znamená $p + q = h_p \leq h_p + h_q - 1 = i$.
  * Podle indukčního předpokladu 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 $NSD(h_p, h_q - h_p)$.

Důkaz proto bude dokončen následujícím Lematem 11.18. □

Největší společný dělitel

Lema 11.18. $NSD(a,b) = NSD(a-b,b) = NSD(a,b-a)$.

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. □

Modulární umocňování

Dále například umocňování na velmi vysoké exponenty je podkladem RSA šifry.

Algoritmus 11.19. Binární postup umocňování.
Pro daná čísla a, b vypočteme jejich celočíselnou mocninu (omezenou na zbytkové třídy modulo m pro prevenci přetečení rozsahu celých čísel v počítači), tj. hodnotu $a^b \mod m$, 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 použijeme indukci podle délky ℓ binárního zápisu čísla b.

Věta. Algoritmus 11.19 skončí a správně vypočte hodnotu $a^b \mod m$.

```
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.
```

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 $res = a \mod m$. □

Nechť tvrzení platí pro všechny vstupy b délky $\ell = i \geq 1$ a uvažme vstup délky $\ell := i + 1$. Pak zřejmě $b \geq 2$ a vykonají se alespoň dvě iterace cyklu. □

Po první iteraci budou hodnoty proměnných po řadě

$$a_1 = a^2, \quad b_1 = \lfloor b/2 \rfloor \text{ a } \quad res = r_1 = (a^{b \mod 2}) \mod m\,.$$

Tudíž délka binárního zápisu b_1 bude jen $\ell - 1 = i$ a podle indukčního předpokladu zbylé iterace algoritmu skončí s výsledkem

$$res = r_1 \cdot a_1^{b_1} \mod m = \left(a^{b \mod 2} \cdot a^{2\lfloor b/2 \rfloor}\right) \mod m = a^b \mod m\,.$$

□

Dodatek II

Relativně rychlé odmocnění

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

Algoritmus 11.20. 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.
```