# 11 Pokročilé dokazování nad algoritmy

Pokračujíce v látce předchozí Lekce 10 se budeme věnovat ukázkám důkazů správné funkce krátkých, ale ne zas tak jednoduchých, algoritmů.


## Stručný přehled lekce

* Zastaví se náš algoritmus a jak to dokázat?
* Důkazy indukcí se systematickými technikami.
* Ukázky netriviálních aritmetických algoritmů s důkazy.

## 11.1 Dokazování konečnosti algoritmu

- Co bývá snad ještě horší, než chybný výsledek algoritmu? Je to situace, kdy spuštěný algoritmus běží „do nekonečna“ a vůbec se nezastaví. □

Všimněte si, že jsme se zatím v důkazech Lekce 10 vůbec nezamýšleli nad tím, zda naše algoritmy vůbec skončí. (To 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. □

- Právě nad takovou situací a její možnou prevencí se v tomto oddíle zamyslíme. Předesíláme, že další podnětná látka k témuž tématu se nachází ještě v Lekci 12.

**Příklad 11.1.** *Zastaví se vždy výpočet následujícího primitivního algoritmu?*

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 $\ell$ 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. □

## Když algoritmus vždy končí

Na rozdíl od předchozí ukázky se budeme dále věnovat pozitivním případům, kdy algoritmy poslušně končí své výpočty.

**Metoda 11.3. Jednoduchý důkaz konečnosti.**
Máme-li za úkol dokázat, že algoritmus skončí, vhodný postup je následující:

- 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. □

Například pro rekurzivní funkci `factorial(x)` z Příkladu 10.12 přímo využijeme parametr `x`, který se ostře zmenšuje, pro snadný důkaz ukončenosti.

**Příklad 11.4.** *Dokažte, že násl. alg. 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):

$$p(x, y) \;=\; 101^2 - 101 \cdot x - y$$

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. □

**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 $x$.

```
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. . . □

V čem je tedy možný nedostatek tohoto přímého přístupu? V zásadě pouze jediný; nezískáme z něj žádný rozumný horní odhad počtu kroků algoritmu. □

Na druhou stranu při trikové volbě parametru $p(x) = 2^x$ pro Metodu 11.3 v každé iteraci rekurzivního volání `fibonacci(x)` pro $x \geq 2$ nastane

$$p(x-1) + p(x-2) = 2^{x-1} + 2^{x-2} \leq 2 \cdot 2^{x-1} - 1 < 2^x$$

a počet iterací výpočtu `fibonacci(n)` tak bude vždy nejen konečný, ale podle uvedené úvahy přímo striktně menší než $2^n$. □

## 11.2 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 „$n \to n + 1$“. □
- 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

**Příklad 11.6.** *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 narůstajícím součtem $y + \cdots + y$, dokud se `x` nesníží na nulu. Poté odhadneme:

**Věta**. Pro každé $x, y \in \mathbb{N}$ Algoritmus 11.6 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 už 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 $x \leftarrow x - 1 = i$.
  Začáteční hodnota vstupu `y` nyní (pro naše indukční úvahy) tudíž je $r_0 + h_y = r_1$ a podle indukčního předpokladu je 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. □

## Indukce k součtu parametrů

**Příklad 11.7.** *Je dán následující rekurentní algoritmus.*

Algoritmus . Funkce `kombinacni(m,n)` pro přirozená $m, n$.

```
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 $r = \binom{m+n}{m}$ (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
```

**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 $m-1+n = m+n-1 = i$-prvkové množiny, například $M = \{1, 2, \ldots, i\}$. 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 přesně rovno `kombinacni(m-1,n) + kombinacni(m,n-1)`, jak jsme měli dokázat.

  □

## Zesílení dokazovaného tvrzení

**Příklad 11.8.** *Zjistěte, kolik znaků* `'z'` *v závislosti na celočíselné hodnotě* $n$ *vstupního parametru* `n` *vypíše následující algoritmus.*

Algoritmus 11.9.

```
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 $n = 0, 1, 2, 3, 4 \ldots$, 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  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 11.9 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 $n_0$ 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'`. □

## 11.3 Zajímavé algoritmy aritmetiky

Například umocňování na velmi vysoké exponenty je podkladem RSA šifry:

**Algoritmus 11.10. 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. $a^b \mod m$, následujícím postupem.*

```
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 $\ell$ binárního zápisu čísla $b$.

**Věta.** Algoritmus 11.10 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 $\ell_0 \geq 1$ 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

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

Tudíž délka binárního zápisu $b_1$ bude jen $\ell_0$ a podle indukčního předpokladu zbylé iterace algoritmu skončí s výsledkem

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

## Euklidův algoritmus

**Algoritmus 11.11. 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** 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 $0$.

```
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. Mějme nyní vstupní hodnoty $p = h_p > 0$ a $q = h_q > 0$ – tehdy dojde k prvnímu průchodu tělem cyklu našeho algoritmu. □
  - Předp. $h_p > h_q$; poté po prvním průchodu tělem cyklu budou hodnoty $p = h_p - h_q$ a $q = h_q$, přičemž nyní $p + q = h_p < h_p + h_q = i$.
  - 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 11.12. □

## Největší společný dělitel

**Lema 11.12.** $NSD(a,b) = NSD(a-b,b) = NSD(a,b-a)$.

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.

□

## 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 11.13. Celočíselná odmocnina.**
*Pro dané přir. číslo $x$ vypočteme dolní celou část jeho odmocniny $\lfloor\sqrt{x}\rfloor$:*

```
p ← x;   res ← 0;
while  p > 0  do
    while  (res + p)^2 ≤ x  do res ← res + p;
    p ← ⌊p/2⌋;
done
output res;
```

## 11.4 Dynamický algoritmus

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 11.14. Dynamický výpočet všech nejkratších cest**
*mezi vrcholy v grafu $G$ na množině vrcholů $V(G) = \{v_0, v_1, \ldots, v_{N-1}\}$.*

- Na počátku nechť `d[i,j]` udává $1$ (případně *váhu–délku hrany* $\{v_i, v_j\}$), nebo $\infty$ pokud hrana mezi $i, j$ není. □
- Po kroku $t \geq 0$ nechť platí, že `d[i,j]` udává délku nejkratší cesty mezi $v_i, v_j$, který užívá pouze vnitřní vrcholy z množiny $\{v_0, v_1, \ldots, v_{t-1}\}$. □
- 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]`. □
- Po $N$ krocích úprav je výpočet hotov.

## Výpočet nejkratších cest

Alternativně si zapíšeme postup této metody až překvapivě krátkým symbolickým algoritmem:

**Algoritmus 11.15. 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 implementaci pro symbol $\infty$ použijeme velkou konst., třeba `MAX_INT/2`.

```
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
```

**Věta 11.16.** *Algoritmus 11.15 v poli* `d[i,j]` *správně vypočte vzdálenost mezi každou dvojicí vrcholů* $v_i, v_j$.

**Důkaz** povedeme matematickou indukcí podle řídící 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 $\{v_0, v_1, \ldots, v_t\}$. □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]`. □

V kroku $t = N$ jsme hotovi se všemi vrcholy. □