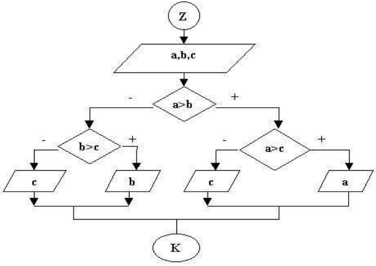
11 Formalizace a důkazy pro algoritmy

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
  - `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. □

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 11.8. *Symbolický zápis jednoduchého rekurzivního algoritmu.*

Algoritmus .

```
function faktorial(x):
    if x ≤ 1 then  t ← 1;
    else  t ← x · faktorial(x-1);
return t .
```

Co je výsledkem výpočtu? □

Jednoduše řečeno, výsledkem je faktoriál vstupní přirozené hodnoty x, tj. hodnota $x! = x \cdot (x-1) \cdot \cdots \cdot 2 \cdot 1$. □

Fibonacciho čísla

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

Algoritmus 11.9. 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 11.9 je víceméně zřejmá z jeho přímé podoby s rekurentním vzorcem v definici Fibonacciho čísel. Algoritmus 11.9 tudíž lze prohlásit za definiční, čili referenční implementaci výpočtu Fibonacciho čísel. Zamyslete se však, jak je to s praktickou „proveditelností" takového algoritmu…

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

Příklad 11.10. *Nerekurzivní algoritmus pro Fibonacciho čísla.*

Dokažte, že následující algoritmus pro každé přirozené n počítá tutéž hodnotu jako rekurentní funkce `fibonacci(n)` v Algoritmu 11.9.

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, tato vyplývá z úvodního přiřazení.

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

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

  což je přesně podle rekurentní definice z Algoritmu 11.9. □

11.4 Techniky důkazu indukcí

- Doposud zde byla matematická indukce obvykle představována ve své přímočaré formě, kdy dokazované tvrzení ihned 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 často neplatí a našim cílem zde je ukázat různé 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 11.11. *Mějme následující algoritmus. Co je jeho výsledkem výpočtu?*

Algoritmus .

```
function soucin(x,y):
    if x ≤ 0 then  t ← 0;
    else  t ← soucin(x-1,y)+y;
return t . □
```

Sledováním průběhu rekurze v algoritmu zjistíme, že hloubka zanoření volání funkce `soucin` bude rovna nezáporné (horní celé) části `x`. Jelikož každé zanoření poté k výsledku přičte hodnotu `y`, odhadneme:

Věta. Pro každé $x, y \in \mathbb{N}$ Algoritmus 11.11 vrátí hodnotu $t = x \cdot y$.

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:

```
function soucin(x,y):
    if x ≤ 0 then  t ← 0;
    else  t ← soucin(x-1,y)+y;
```

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

Dokazujeme tady, že pro každý vstup $x := i \in \mathbb{N}$ volání funkce `soucin(x,y)` navrátí hodnotu $i \cdot y$. □

- V bázi pro $x = i = 0$ provedeme první větev podmínky, tj. $t \leftarrow 0 = 0 \cdot y$.□
- Indukční krok. Nechť je tvrzení známo pro $x = i \in \mathbb{N}$ a uvažujme další vstup $x := i + 1 > 0$. V tom případě se vykonává druhá větev podmínky, čili $t \leftarrow \mathtt{soucin}(x - 1, y) + y$. Všimněte si, že platí $x - 1 = i$. □
  Podle indukčního předpokladu již víme, že návratovou hodnotu volání funkce $\mathtt{soucin}(i, y)$ je $i \cdot y$. □Zbývá jen uvedené poznatky správně zřetězit

$$t \leftarrow \mathtt{soucin}(x-1, y) + y = \mathtt{soucin}(i, y) + y = i \cdot y + y = (i+1)y = x \cdot y\,.$$

Důkaz matematickou indukcí je tímto zdárně ukončen. □

Indukce k součtu parametrů

Příklad 11.12. *Co je výsledkem následujícího rekurzivního výpočtu?*

Algoritmus .

```
function zkombinovat(m,n):
    if m=0 ∨ n=0 then  res ← 1;
    else
        res ← zkombinovat(m-1,n)+zkombinovat(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 výsledek výpočtu funkce `zkombinovat(m,n)` roven počtu všech m-prvkových podmnožin $(m+n)$-prvkové množiny, tedy kombinačnímu číslu $res = \binom{m+n}{m}$.

```
if m=0 ∨ n=0 then  res ← 1;
else
    res ← zkombinovat(m-1,n)+zkombinovat(m,n-1);
fi
```

$res \stackrel{?}{=}$ počet m-prvkových podmn. $(m+n)$-prvk. množ., tj. $\binom{m+n}{m}$

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 počáteční `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.

```
    if m = 0 ∨ n = 0 then  res ← 1;
    else
        res ← zkombinovat(m-1,n) + zkombinovat(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

  `zkombinovat(m-1,n) + zkombinovat(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 `zkombinovat(m-1,n) + zkombinovat(m,n-1)`, jak jsme měli v indukčním kroku dokázat. □

Zesílení dokazovaného tvrzení

Příklad 11.13. *Zjistěte, jaká je návratová hodnota volání následující funkce* `tajemne`$(x, 1)$ *v závislosti na celočíselné vstupní hodnotě* x.

Algoritmus 11.14.

```
function tajemne(x,y):
    if x ≤ 0 then  t←0;
    else  t ← tajemne(x-1,2·y)+y;
return t .
```
□

Zkusíme-li si výpočet simulovat pro $x = 0, 1, 2, 3, 4\dots$, postupně získáme pro `tajemne`$(x, 1)$ návratové hodnoty $0, 1, 3, 7, 15\dots$. □Na základě toho již není obtížné uhodnout, že návratová hodnota asi bude obecně určena vztahem $2^x - 1$.

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, které zobecňuje výsledek na proměnné hodnoty parametru y (a které také musíme uhodnout):

Algoritmus .

```
function tajemne(x,y):
    if x ≤ 0 then  t ← 0;
    else  t ← tajemne(x-1,2·y)+y;
```

Věta. Pro každá přirozená x, y volání funkce $\texttt{tajemne}(x, y)$ Algoritmu 11.14 vrátí hodnotu $(2^x - 1) \cdot y$. □

Důkaz: Postupujeme indukcí podle $x := i \in \mathbb{N}$. □

- V bázi, pro vstup $x = i = 0$ je navrácena hodnota $0 = (2^0 - 1) \cdot y$.□
- Indukční krok. Nechť je tvrzení známo pro $x = i \in \mathbb{N}$ a uvažujme další vstup $x := i + 1 > 0$. V tom případě se vykonává druhá větev podmínky a výsledek bude roven hodnotě $\texttt{tajemne}(x-1, 2y)+y = \texttt{tajemne}(i, 2y)+y$.□ Podle indukčního předpokladu již víme, že volání funkce $\texttt{tajemne}(i, 2y)$ navrátí $(2^i - 1) \cdot 2y$, a celkem tak platí: □

$$\texttt{tajemne}(i, 2y) + y = (2^i - 1) \cdot 2y + y = 2^{i+1}y - 2y + y = (2^{i+1} - 1) \cdot y$$

Poslední výraz je roven požadovanému $(2^x - 1) \cdot y$. □

11.5 Dodatek: Zajímavé algoritmy aritmetiky

Euklidův algoritmus

Algoritmus 11.15. **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 11.16. □

Největší společný dělitel

Lema 11.16. $NSD(a,b) = NSD(a-b,b) = NSD(a,b-a)$.

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 11.17. 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 11.17 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 11.18. 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.
```