Co je to algoritmus?

Algoritmus

Algoritmus je mechanický postup skládající se z nějakých jednoduchých elementárních kroků, který pro nějaký zadaný **vstup** vyprodukuje nějaký **výstup**.

Algoritmus může být zadán:

- slovním popisem v přirozeném jazyce
- pseudokódem
- jako počítačový program v nějakém programovacím jazyce
- jako hardwarový obvod
- ...

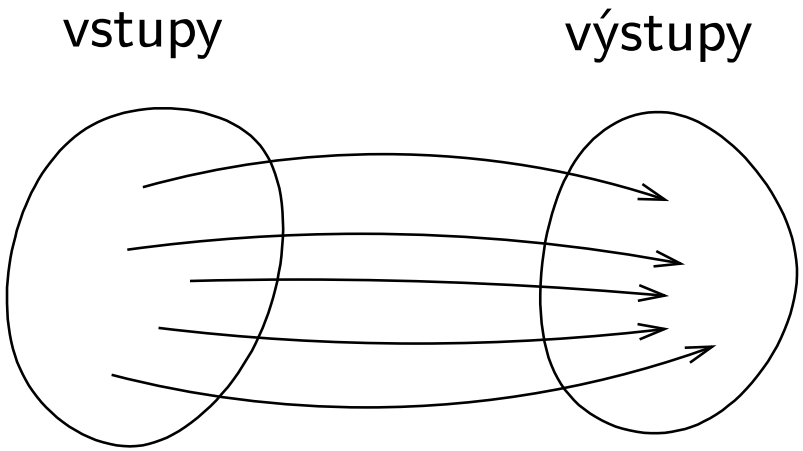
Algoritmy slouží k řešení různých **problémů**.

Problémy

Problém

V zadání **problému** musí být určeno:

- co je množinou možných vstupů
- co je množinou možných výstupů
- jaký je vztah mezi vstupy a výstupy



Příklady problémů

Problém „Třídění“

Vstup: Sekvence prvků $a_1, a_2, \ldots, a_n$.

Výstup: Prvky sekvence $a_1, a_2, \ldots, a_n$ seřazené od nejmenšího po největší.

Příklad:

- Vstup: $8, 13, 3, 10, 1, 4$
- Výstup: $1, 3, 4, 8, 10, 13$

Poznámka: Konkrétní vstup nějakého problému se nazývá **instance** problému.

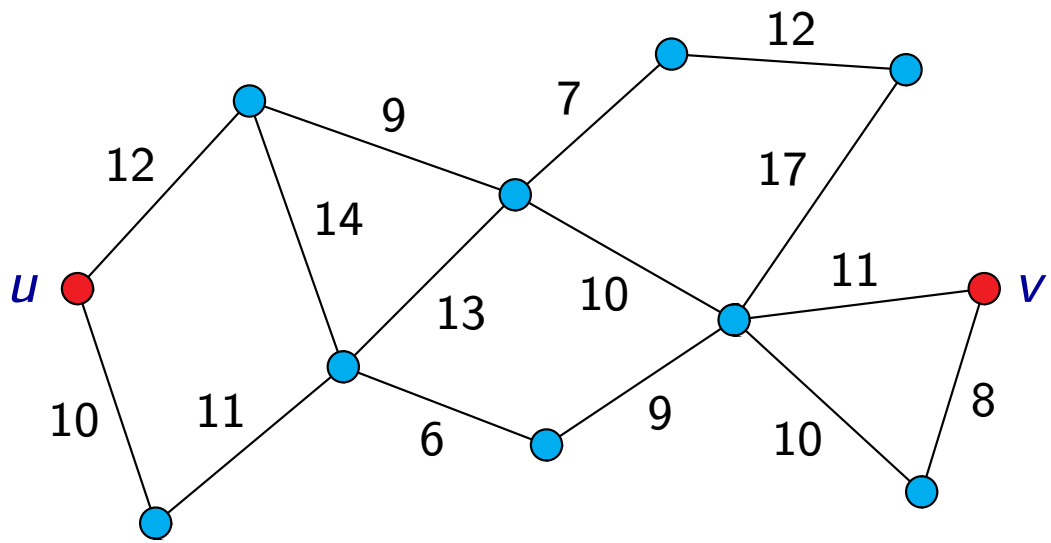
Příklady problému

Problém „Hledání nejkratší cesty v (neorientovaném) grafu“

Vstup: Neorientovaný graf $G = (V, E)$ s ohodnocením hran, a dvojice vrcholů $u, v \in V$.

Výstup: Nejkratší cesta z vrcholu u do vrcholu v.

Příklad:



Další příklady problémů

Problém „Prvočíselnost“

Vstup: Přirozené číslo n.

Výstup: ANO pokud je n prvočíslo, NE v opačném případě.

Poznámka: Přirozené číslo n je **prvočíslo**, pokud je větší než 1 a je dělitelné beze zbytku pouze čísly 1 a n.

Prvních několik prvočísel: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, ...

Rozhodovací problémy

Situace, kdy množina výstupů je $\{\text{Ano}, \text{Ne}\}$ je poměrně častá. Takovým problémům se říká **rozhodovací problémy**.

Rozhodovací problémy většinou specifikujeme tak, že místo popisu toho, co je výstupem, uvedeme otázku.

Příklad:

Problém „Prvočíselnost“

Vstup: Přirozené číslo n.

Otázka: Je n prvočíslo?

Optimalizační problémy

Dalším speciálním případem jsou tzv. optimalizační problémy.

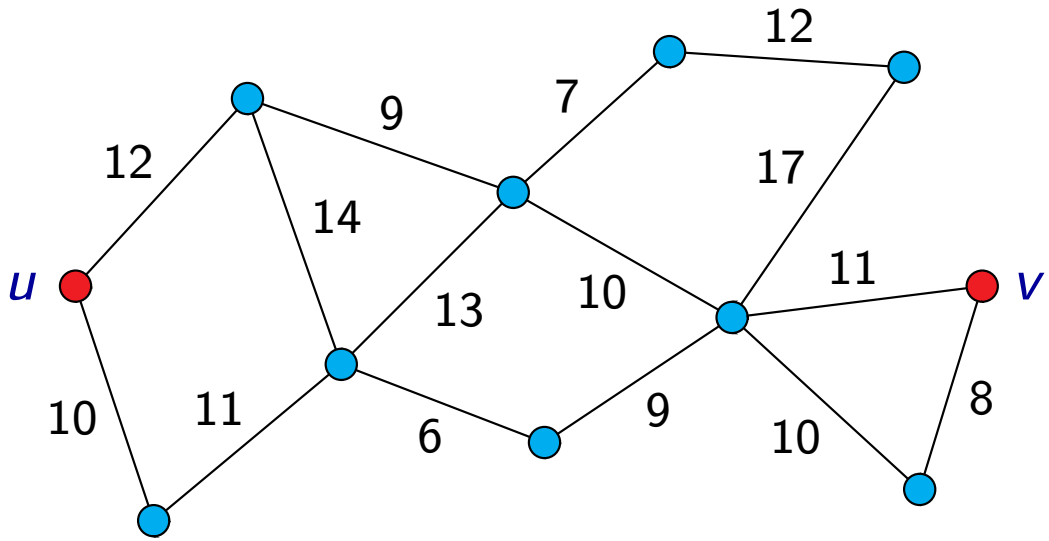
Optimalizační problém je problém, kde je úkolem vybrat z nějaké množiny přípustných řešení takové řešení, které je v nějakém ohledu optimální.

Optimalizační problémy

Dalším speciálním případem jsou tzv. optimalizační problémy.

Optimalizační problém je problém, kde je úkolem vybrat z nějaké množiny přípustných řešení takové řešení, které je v nějakém ohledu optimální.

Příklad: V problémy „Hledání nejkratší cesty v grafu“ je množina všech přípustných řešení tvořena všemi cestami z vrcholu u do vrcholu v. Kritériem, podle kterého cesty hodnotíme, je délka cesty.



Řešení problémů

Řešení problému

Algoritmus **korektně řeší** daný problém, když:

1. Se pro libovolný vstup daného problému (libovolnou vstupní instanci) po konečném počtu kroků zastaví.
2. Vyprodukuje výstup z množiny možných výstupů, který vyhovuje podmínkám uvedeným v zadání problému.

Pro jeden problém může existovat celá řada algoritmů, které jej korektně řeší.

Poznámka: Množina vstupních instancí bývá typicky nekonečná.

Kódování vstupu a výstupu

Většinou předpokládáme, že vstupy i výstupy jsou kódovány jako slova v nějaké abecedě Σ.

Příklad: Například u problému „Třídění“ bychom mohli zvolit jako abecedu $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, , \}$.

Vstupem by pak mohlo být například slovo

$$826,13,3901,101,128,562$$

a výstupem slovo

$$13,101,128,562,826,3901$$

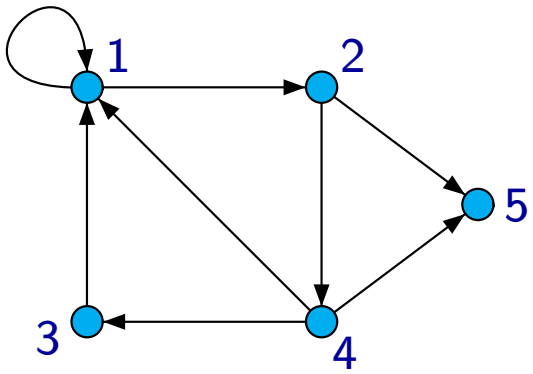
Kódování vstupu a výstupu

Poznámka: Ne každé slovo ze Σ^* musí reprezentovat nějaký vstup. Kódování bychom ale měli zvolit tak, abychom byli schopni snadno poznat ta slova, která nějaký vstup reprezentují.

Kódování vstupu a výstupu

Příklad: Pokud je vstupem nějakého problému například graf, můžeme ho reprezentovat jako seznam vrcholů a hran:

Například následující graf



můžeme reprezentovat jako slovo

$$(1,2,3,4,5),((1,2),(2,4),(4,3),(3,1),(1,1),(2,5),(4,5),(4,1))$$

v abecedě $\Sigma = \{0,1,2,3,4,5,6,7,8,9,,,(,)\}$.

Kódování vstupu a výstupu

Můžeme se omezit na případ, kdy jsou vstupy i výstupy kódovány jako slova v abecedě $\{0, 1\}$ (tj. jako sekvence bitů).

Symboly jakékoli jiné abecedy lze reprezentovat jako sekvence bitů.

Příklad: Abeceda $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}\}$

$$
\begin{aligned}
\mathtt{a} &\leftrightarrow 001 \\
\mathtt{b} &\leftrightarrow 010 \\
\mathtt{c} &\leftrightarrow 011 \\
\mathtt{d} &\leftrightarrow 100 \\
\mathtt{e} &\leftrightarrow 101 \\
\mathtt{f} &\leftrightarrow 110 \\
\mathtt{g} &\leftrightarrow 111
\end{aligned}
$$

Slovo '$\mathtt{defb}$' můžeme reprezentovat jako '100101110010'.

Funkce realizovaná algoritmem

Můžeme tedy říct, že každý algoritmus realizuje výpočet hodnot nějaké funkce

$$f : \Sigma^* \to \Sigma^*$$

kde $\Sigma = \{0, 1\}$.

Alternativně se také na algoritmy můžeme dívat tak, že realizují výpočet nějaké funkce

$$f : \mathbb{N} \to \mathbb{N}$$

kde $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ je množina přirozených čísel, neboť na každou sekvenci bitů se můžeme dívat jako na zápis čísla v binární soustavě.

Funkce realizovaná algoritmem

Funkce

$$f : \Sigma^* \to \Sigma^* \qquad \text{resp.} \qquad f : \mathbb{N} \to \mathbb{N}$$

realizovaná nějakým algoritmem nemusí být nutně totální, může být částečná. Hodnota $f(x)$ není definovaná například v případě, že se daný algoritmus pro vstup x nikdy nezastaví.

Poznámka: Funkce $f : A \to B$ je **totální**, jestliže hodnota $f(x)$ je definovaná pro libovolné $x \in A$, a **částečná**, jestliže pro některá $x \in A$ nemusí být hodnota $f(x)$ definovaná.

Algoritmy a problémy

Stejně jako na algoritmy i na problémy se můžeme dívat jako na funkce typu:

$$f : \Sigma^* \to \Sigma^* \qquad \text{resp.} \qquad f : \mathbb{N} \to \mathbb{N}$$

Jestliže algoritmus realizuje nějakou funkci f', řekneme, že tento algoritmus **řeší** problém popsaný funkcí f, jestliže se pro libovolný vstup x zastaví s tím, že

$$f(x) = f'(x)$$

Rozhodovací problémy a jazyky

Na rozhodovací problémy můžeme pohlížet jako na jazyky.

Jazyk odpovídající danému rozhodovacímu problému je množina těch slov ze Σ^*, která reprezentují ty vstupy, pro něž je odpověď ANO.

Příklad: Jazyk L tvořený těmi slovy ze $\{0,1\}^*$, která jsou binárním zápisem nějakého prvočísla.

Například $101 \in L$, ale $110 \notin L$.

Rozhodovací problémy

Často se při zkoumání algoritmů a problému omezujeme jen na rozhodovací problémy.

Není to však na úkor obecnosti, neboť libovolný obecný problém možné vhodným způsobem přeformulovat jako rozhodovací problém, s tím, že když najdeme algoritmus, který by řešil tento rozhodovací problém, tak bychom snadno sestrojili algoritmus, který by řešil původní problém, a naopak.

Rozhodovací problémy

Pokud například máme problém P, kde:

- vstupy jsou prvky z nějaké množiny X
- výstupy jsou slova z $\{0,1\}^*$

můžeme tento problém přeformulovat jako následující rozhodovací problém:

Problém

Vstup: Prvek $x \in X$ a číslo k.

Otázka: Když z je výstup, který odpovídá vstupu x problému P, má k-tý bit slova z hodnotu 1?

Algoritmicky řešitelné problémy

Předpokládejme, že máme dán nějaký problém P.

Jestliže existuje nějaký algoritmus, který řeší problém P, pak říkáme, že problém P je **algoritmicky řešitelný**.

Jestliže P je rozhodovací problém a jestliže existuje nějaký algoritmus, který problém P řeší, pak říkáme, že problém P je **rozhodnutelný**.

Když chceme ukázat, že problém P je algoritmicky řešitelný, stačí ukázat nějaký algoritmus, který ho řeší (a případně ukázat, že daný algoritmus problém P skutečně řeší).

Algoritmicky řešitelné problémy

U mnohých problémů je na první pohled zřejmé, že jsou algoritmicky řešitelné, jako třeba:

- Třídění
- Hledání nejkratší cesty v grafu
- Prvočíselnost

kde stačí probrat všechny možnosti (kterých je ve všech těchto případech konečně mnoho), i když takový triviální algoritmus založený na řešení **hrubou silou** nemusí být zrovna efektivní.

Na druhou stranu existuje celá řada problémů, u kterých to tak jasné není.

- Najít algoritmus a dokázat, že řeší daný problém, může být velmi netriviální úkol.
- Algoritmus, který by řešil daný problém, nemusí vůbec existovat.

Nutnost upřesnění pojmu „algoritmus“

Dosavadní definice pojmu algoritmus byla poněkud vágní.

Pokud bychom pro nějaký problém chtěli ukázat, že neexistuje algoritmus, který by daný problém řešil, tak by to s takovouto neurčitou definicí pojmu algoritmus asi nešlo.

Intuitivně chápeme, co by měl mít algoritmus za vlastnosti:

- Měl by se skládat z jednoduchých kroků, které je možno vykonávat „mechanicky“, bez porozumění problému.
- Objekty, se kterými algoritmus pracuje, i prováděné operace by měly být konečné.

Algoritmus realizovaný počítačem

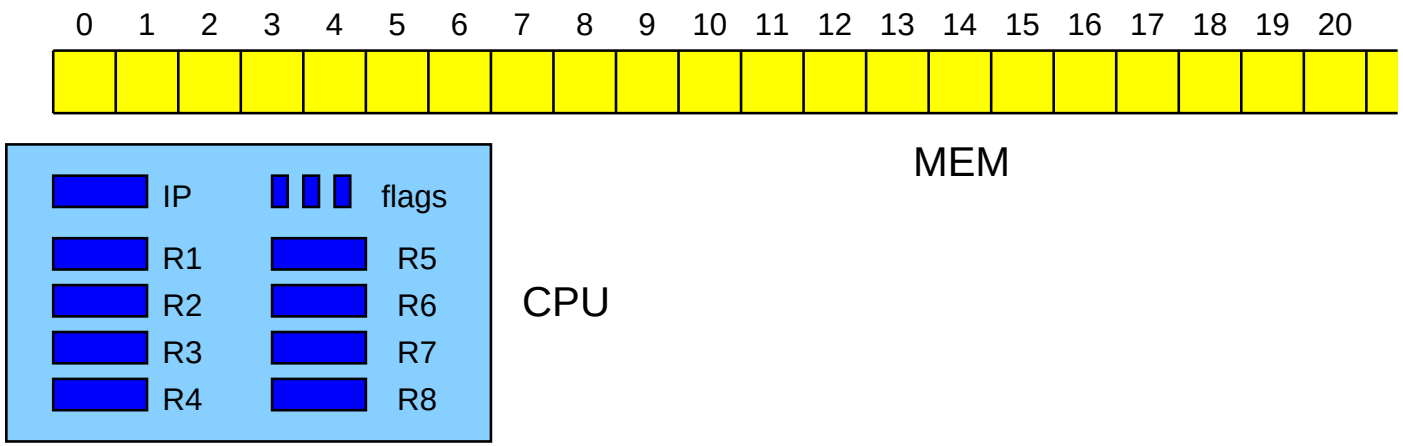
Například program v libovolném programovacím jazyce dané vlastnosti zcela jistě má.

Ať už je program napsán v jakémkoliv programovacím jazyce, jsou jeho instrukce nakonec prováděny na hardwaru nějakého konkrétního počítače na úrovni instrukcí procesoru.

Je tedy jasné, že **každý** program v **každém** programovacím jazyce bychom mohli zapsat jako program tvořený pouze instrukcemi strojového kódu nějakého procesoru.

Činnost počítače

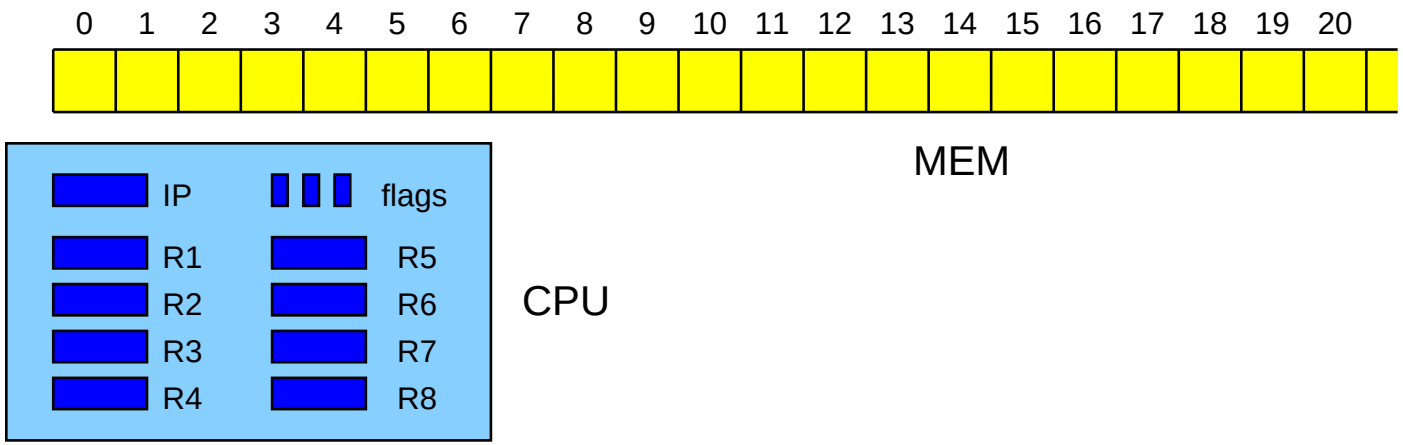
Typická architektura naprosté většiny počítačů vypadá následovně:



- Počítač má **paměť** skládající se z velkého množství paměťových buněk.
- Každá buňka může obsahovat číslo určité velikosti, typicky 1 byte (8 bitů), tj. číslo v rozsahu $0 \ldots 255$.
- Buňky jsou očíslovány. Číslo buňky se nazývá její **adresa**.

Činnost počítače

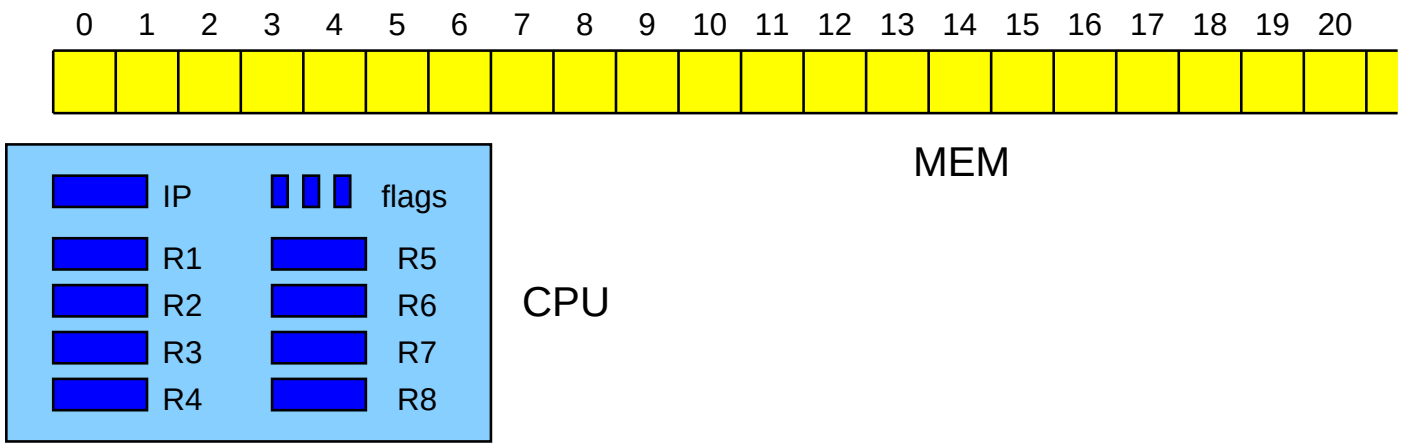
Typická architektura naprosté většiny počítačů vypadá následovně:



- Instrukce jsou uloženy v paměti (každá instrukce má svůj číselný **kód**) a jsou sekvenčně vykonávány **procesorem**.
- Procesor udržuje tzv. **čítač instrukcí** IP, který obsahuje adresu aktuálně prováděné instrukce.
- Procesor načte instrukci z adresy určené IP, zvětší IP o délku načtené instrukce a provede danou instrukci.

Činnost počítače

Typická architektura naprosté většiny počítačů vypadá následovně:



- Procesor obsahuje několik **registrů** pevné délky (např. 32 nebo 64 bitů).
- Většina operací je prováděna na registrech.
- Procesor obsahuje **příznaky (flags)**, které umožňují testovat výsledek poslední operace (např. přetečení, jestli je výsledek nula apod.)

Činnost počítače

Typické instrukce:

- Načtení obsahu paměťové buňky (resp. několika po sobě jdoucích buněk) do některého registru (LOAD).
- Uložení obsahu registru do některé paměťové buňky (resp. několika po sobě jdoucích buněk) (STORE).

  Poznámka: Adresa buňky je buď přímá (tj. je přímo součástí instrukce) nebo nepřímá (uložená v některém registru, případně spočítaná z obsahu jednoho nebo několika registrů).
- Načtení obsahu jednoho registru do jiného registru (MOV).
- Aritmetické instrukce (ADD, SUB, MUL, DIV, NEG, CMP, INC, DEC, ...).
- Logické instrukce (AND, OR, XOR, NOT, ...).
- Bitové posuny a rotace (SHL, SHR, ...)

Činnost počítače

Typické instrukce (pokračování):

- Nepodmíněný skok (JMP).

  Poznámka: Adresa skoku může být přímá nebo nepřímá.

- Podmíněné skoky (JZERO, JGTZ, ...).
- Volání podprogramů (CALL, RET).
- Různé speciální instrukce – práce se vstupem a výstupem, obsluha přerušení, mody činnosti procesoru, řízení přístupu do paměti (stránkování) apod.

Činnost počítače

Příklad: Instrukce zapsaná ve vyšším programovacím jazyce jako

$$x = y + 2$$

může být realizována následující sekvencí instrukcí určitého (hypotetického) procesoru:

```
LOAD    0x001b7c42,R5
ADD     $2,R5
STORE   R5,0x001b7c38
```

Poznámka: Předpokládáme, že proměnná x je uložena na adrese 0x001b7c38 a proměnná y na adrese 0x001b7c42.

Stroj RAM

Stroj RAM (Random Access Machine) je idealizovaný model počítače.

Rozdíly oproti skutečnému počítači:

- Velikost paměti není omezena (adresa může být libovolné přirozené číslo).
- Velikost obsahu jednotlivých buněk není omezena (buňka může obsahovat libovolné celé číslo).
- Čte data sekvenčně ze vstupu, který je tvořen sekvencí celých čísel. Ze vstupu lze pouze číst.
- Zapisuje data sekvenčně na výstup, který je tvořen sekvencí celých čísel. Na výstup je možné pouze zapisovat.

Stroj RAM a Turingův stroj

Každý program v každém jazyce by mohl být realizován jako program stroje RAM.

Není složité (i když je to trochu pracné) si rozmyslet, že libovolný algoritmus prováděný strojem RAM je možné realizovat také Turingovým stojem.

Turingův stroj je schopen realizovat libovolný algoritmus, který by bylo možné zapsat jako program v nějakém programovacím jazyce.

Poznámka: Turingův stroj pracuje se slovy nad nějakou abecedou, zatímco stroj RAM s čísly. Čísla ale můžeme zapisovat jako sekvence symbolů a naopak symboly nějaké abecedy můžeme zapisovat jako čísla.

Churchova-Turingova teze

Churchova-Turingova teze

Každý algoritmus je možné realizovat nějakým Turingovým strojem.

Není to věta, kterou by bylo možno dokázat v matematickém smyslu – není formálně definováno, co je to algoritmus.

Tezi formulovali nezávisle na sobě v polovině 30. let 20. století Alan Turing a Alonzo Church.

Churchova-Turingova teze

Ve stejné době bylo navrženo několik různých formalismů zachycujících pojem algoritmus:

- Turingovy stroje (Alan Turing)
- lambda kalkulus (Alonzo Church)
- rekurzivní funkce (Stephan Kleene)
- produkční systémy (Emil Post)
- ...

Dále můžeme uvést:

- Libovolný (obecný) programovací jazyk.

Všechny tyto modely jsou ekvivalentní z hlediska algoritmů, které jsou schopny realizovat.

Algoritmicky neřešitelné problémy

Problém, který není algoritmicky řešitelný je **algoritmicky neřešitelný**.

Rozhodovací problém, který není rozhodnutelný je **nerozhodnutelný**.

Příklad: Následující problém zvaný **Problém zastavení (Halting problem)** je nerozhodnutelný:

Halting problem

Vstup: Popis Turingova stroje M a slovo w.

Otázka: Zastaví se stroj M po nějakém konečném počtu kroků, pokud dostane jako svůj vstup slovo w?

Halting problem

Alternativně bychom ho mohli formulovat třeba takto:

Halting problem

Vstup: Zdrojový kód programu P v jazyce C, vstupní data x.

Otázka: Zastaví se program P po nějakém konečném počtu kroků, pokud dostane jako vstup data x?

Halting problem

Předpokládejme, že by existoval nějaký program, který by rozhodoval Halting problem.

Mohli bychom tedy vytvořit podprogram H, deklarovaný jako

boolean H(String kod, String vstup)

kde $H(P, x)$ vrátí:

- true pokud se program P zastaví pro vstup x,
- false pokud se program P nezastaví pro vstup x.

Poznámka: Řekněme, že podprogram $H(P, x)$ by vracel false v případě, že P není syntakticky správný kód programu.

Halting problem

S použitím podprogramu H bychom vytvořili program D, který bude provádět následující kroky:

- Načte svůj vstup do proměnné x typu String.
- Zavolá podprogram $H(x, x)$.
- Pokud podprogram H vrátil true, skočí do nekonečné smyčky

loop: goto loop

V případě, že H vrátil false, program D se ukončí.

Co udělá program D, pokud mu předložíme jako vstup jeho vlastní kód?

Halting problem

Pokud D dostane jako vstup svůj vlastní kód, tak se buď zastaví nebo nezastaví.

- Pokud se D zastaví, tak $H(D, D)$ vrátí true a D skočí do nekonečné smyčky. Spor!
- Pokud se D nezastaví, tak $H(D, D)$ vrátí false a D se zastaví. Spor!

V obou případech dospějeme ke sporu a další možnost není. Nemůže tedy platit předpoklad, že H řeší Halting problem.

Bonusový příklad

V předchozím příkladě jsme měli případ, kdy program zpracovával svůj vlastní kód. To není nic neobvyklého, například překladač může překládat svůj vlastní kód.

Uvažujme ale opačný případ: Chceme vytvořit program, který naopak svůj vlastní kód vydá jako výstup.

Úkol: Napište program ve vašem oblíbeném programovacím jazyce, který vypíše (například na standardní výstup) svůj vlastní zdrojový kód, přičemž ale:

- Nesmí číst data ze žádného externího zdroje (disku, klávesnice apod.).
- Nesmí být prázdný.

Další nerozhodnutelné problémy

S jedním příkladem nerozhodnutelného problému už jsme se setkali:

Problém

Vstup: Bezkontextové gramatiky G_1 a G_2.

Otázka: Je $L(G_1) = L(G_2)$?

případně

Problém

Vstup: Bezkontextová gramatika G generující jazyk nad abecedou Σ.

Otázka: Je $L(G) = \Sigma^*$?

Redukce mezi problémy

Pokud máme o nějakém (rozhodovacím) problému dokázáno, že je nerozhodnutelný, můžeme ukázat nerozhodnutelnost dalších problémů pomocí tzv. redukcí.

Řekněme, že A a B jsou rozhodovací problémy.
Redukce problému A na problém B je algoritmus P, který:

- Dostane jako vstup instanci problému A (označme ji x).
- Jako svůj výstup (označme jej $P(x)$) vyprodukuje instanci problému B.
- Platí

$$x \in A \qquad \Leftrightarrow \qquad P(x) \in B$$

tj. pro vstup x je v problému A odpověď ANO právě tehdy, když pro vstup $P(x)$ je v problému B odpověď ANO.

Redukce mezi problémy

Řekněme, že existuje redukce P problému A na problém B.

Pokud by problém B byl rozhodnutelný, pak i problém A je rozhodnutelný.

Řešení problému A pro vstup x:

- Zavoláme P se vstupem x, vrátí nám hodnotu $P(x)$.
- Zavoláme algoritmus řešící problém B se vstupem $P(x)$. Hodnotu, kterou nám vrátí vypíšeme jako výsledek.

Je zřejmé, že pokud A je nerozhodnutelný, tak B nemůže být rozhodnutelný.

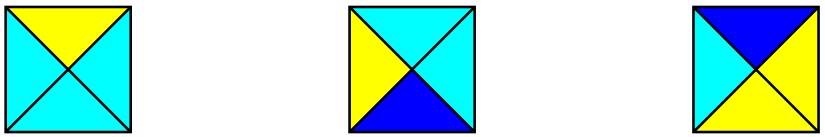
Další nerozhodnutelné problémy

Redukcí z Halting problému se dá ukázat nerozhodnutelnost celé řady problémů, které se týkají ověřování chování programů:

- Vydá daný program pro nějaký vstup odpověď ANO?
- Zastaví se daný program pro libovolný vstup?
- Dávají dva dané programy pro stejné vstupy stejný výstup?
- ...

Další nerozhodnutelné problémy

Vstupem je množina typů kachliček, jako třeba:

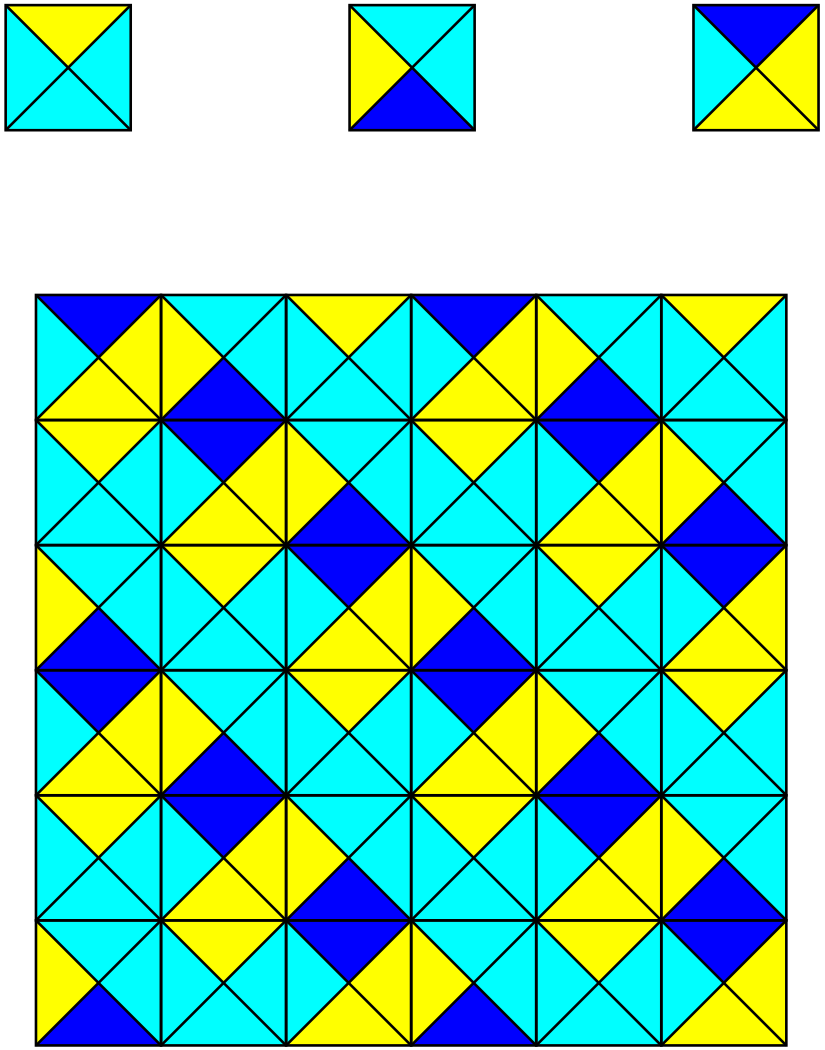


Otázka je, zda je možné použitím daných typů kachliček pokrýt každou libovolně velkou konečnou plochu tak, aby všechny kachličky spolu sousedily stejnými barvami.

Poznámka: Můžeme předpokládat, že máme v zásobě neomezené množství kachliček všech typů.

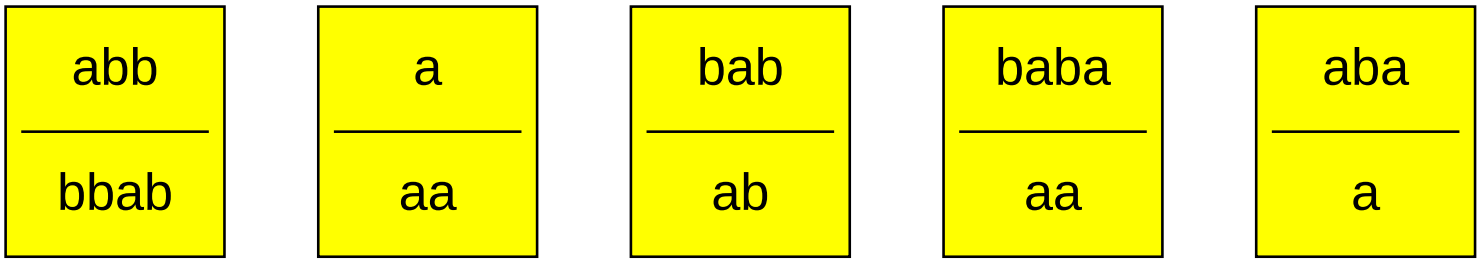
Kachličky není dovoleno otáčet.

Další nerozhodnutelné problémy

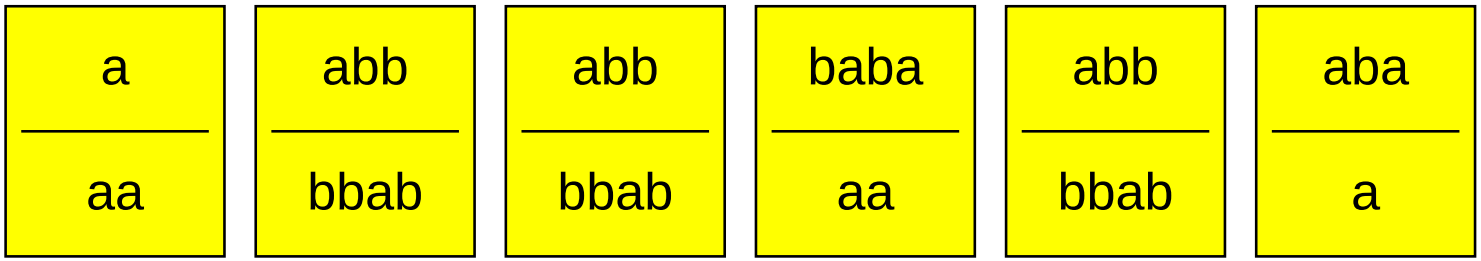


Další nerozhodnutelné problémy

Vstupem je množina typů kartiček, jako třeba:



Otázka je, zda je možné z těchto typů kartiček vytvořit neprázdnou konečnou posloupnost, kde zřetězením slov nahoře i dole vznikne totéž slovo. Každý typ kartičky je možné používat opakovaně.



Nahoře i dole vznikne slovo aabbabbbabaabbaba.

Další nerozhodnutelné problémy

Redukcí z předchozího problému se dá snadno ukázat nerozhodnutelnost některých dalších problémů z oblasti bezkontextových gramatik:

Problém

Vstup: Bezkontextové gramatiky G_1 a G_2.

Otázka: Je $L(G_1) \cap L(G_2) = \emptyset$?

Problém

Vstup: Bezkontextová gramatika G.

Otázka: Je G nejednoznačná?

Další nerozhodnutelné problémy

Problém

Vstup: Uzavřená aritmetická formule vytvořená ze symbolů $+$, $\times$, $=$, $\wedge$, $\vee$, $\neg$, $\forall$, $\exists$, $($, $)$, proměnných a celočíselných konstant.

Otázka: Je tato formule pravdivá v oboru přirozených čísel?

Příklad vstupu:

$$\forall x \exists y \forall z((x \times y = z) \wedge (y + 5 = x))$$

Poznámka: Úzce souvisí s Gödelovou větou o neúplnosti.

Rozhodnutelný problém

Následující na první pohled velice podobný problém je rozhodnutelný:

Problém

Vstup: Uzavřená aritmetická formule vytvořená ze symbolů $+$, $=$, $\wedge$, $\vee$, $\neg$, $\forall$, $\exists$, $($, $)$, proměnných a celočíselných konstant.

Otázka: Je tato formule pravdivá v oboru přirozených čísel?

Příklad vstupu:

$$\forall x \exists y \forall z((x + y = z) \wedge (y + 5 = x))$$