

Základy složitosti a vyčísitelnosti

NTIN090

Petr Kučera

2018/19

Úvod

Sylabus

- 1 Turingovy stroje a jejich varianty. Churchova-Turingova teze
- 2 Halting problém a další nerozhodnutelné problémy
- 3 RAM a jeho ekvivalence s Turingovými stroji. Algoritmicky vyčíslitelné funkce
- 4 Rozhodnutelné a částečně rozhodnutelné jazyky a jejich vlastnosti
- 5 m -převoditelnost a m -úplné jazyky
- 6 Riceova věta
- 7 Nedeterministické Turingovy stroje, základní třídy složitosti, třídy **P**, **NP**, **PSPACE**, **EXPTIME**
- 8 Savičova věta
- 9 Věty o deterministické prostorové a časové hierarchii
- 10 Polynomiální převoditelnost problémů, pojmy **NP**-těžkosti a **NP**-úplnosti
- 11 Cook-Levinova věta, příklady **NP**-úplných problémů, důkazy **NP**-úplnosti
- 12 Pseudopolynomiální algoritmy a silná **NP**-úplnost
- 13 Aproximace **NP**-těžkých optimalizačních úloh, aproximační algoritmy a schémata
- 14 Třídy **co-NP** a **#P**

Obojí

- Sipser, M. *Introduction to the Theory of Computation*. Vol. 2. Boston: Thomson Course Technology, 2006.
- Mé poznámky na stránce k předmětu (<http://ktiml.mff.cuni.cz/~kucera/NTIN090/>)

Vyčíslitelnost

- Demuth O., Kryl R., Kučera A.: *Teorie algoritmů I, II*. SPN, 1984, 1989
- Soare R.I.: *Recursively enumerable sets and degrees*. Springer-Verlag, 1987
- Odifreddi P.: *Classical recursion theory*, North-Holland, 1989

Složitost

- Garey, Johnson: *Computers and intractability — a guide to the theory of NP-completeness*, W.H. Freeman 1978
- Arora S., Barak B.: *Computational Complexity: A Modern Approach*. Cambridge University Press 2009.

Motivační otázky

- 1 Co je to algoritmus?
- 2 Co všechno lze pomocí algoritmů spočítat?
- 3 Dokáží algoritmy vyřešit všechny úlohy a problémy?
- 4 Jak poznat, že pro řešení zadané úlohy nelze sestavit žádným algoritmus?
- 5 Jaké algoritmy jsou „rychlé“ a jaké problémy jimi můžeme řešit?
- 6 Jaký je rozdíl mezi časem a prostorem?
- 7 Které problémy jsou lehké a které těžké? A jak je poznat?
- 8 Je lépe zkoušet, nebo být zkoušený?
- 9 Jak řešit problémy, pro které neznáme žádný „rychlý“ algoritmus?

Vyčíslitelnost

Lehký úvod do teorie algoritmů

První program: Hello, world!

Jak se patří na přednášku o programování, i my začneme programem „Hello world“ (například v jazyce C).

```
helloworld.c
```

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Hello, world\n");
    return 0;
}
```

- Na první pohled vidíme, že program vždy skončí a prvních dvanáct znaků, které vypíše jsou *Hello, world*.
- Program s podobnou funkcí můžeme však napsat i jiným způsobem...

Program Hello, world! (2. verze)

helloworld2.c

```
#include <stdio.h>

int exp(int i, int n)
/* Vrátí n-tou mocninu i */
{
    int moc, j;
    moc=1;
    for (j=1; j<=n; ++j) moc *= i;
    return moc;
}
```

Program Hello, world! (2. verze)

```
int main(int argc, char *argv[]) {
    int n, total, x, y, z;
    scanf("%d", &n);
    total=3;
    while (1) {
        for (x=1; x<=total-2; ++x) {
            for (y=1; y<=total-x-1; ++y) {
                z=total-x-y;
                if (exp(x,n)+exp(y,n)==exp(z,n)) {
                    printf("Hello, world\n");
                    return 0;
                }
            }
        }
        ++total;
    }
}
```



Za jakých podmínek vypíše program `helloworld2` jako prvních dvanáct znaků na výstup *Hello, world* a zastaví se?

Program `helloworld2` skončí a vypíše jako prvních 12 znaků *Hello, world*, právě když `scanf` načte číslo $n \leq 2$. Pro $n > 2$ program `helloworld2` neskončí.

K důkazu tohoto faktu potřebujeme velkou Fermatovu větu!



Problém HELLOWORLD

Helloworld

Instance: Zdrojový kód programu P v jazyce C a vstup I .

Otázka: Je pravda, že prvních 12 znaků, které daný program vypíše, je *Hello, world*? (Nevyžadujeme zastavení.)



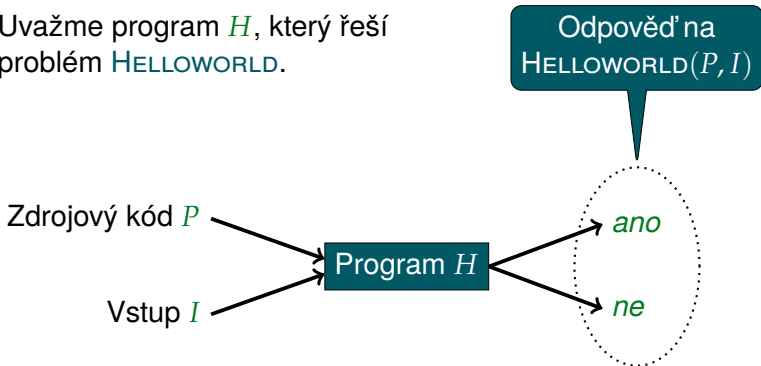
Lze napsat program v jazyce C, který pro dvojici P , I zodpoví otázku kladenou v problému HELLOWORLD?



Ukážeme si, že nikoli.

Nerozhodnutelnost HELLOWORLD

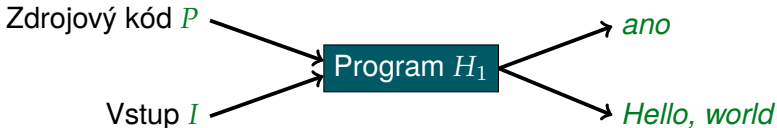
Uvažme program H , který řeší problém HELLOWORLD.



- Předpokládáme, že vstup je předáván programu P i H na standardní vstup a je čten výhradně funkcí `scanf`.
- Předpokládáme, že výstup je programy P i H zapisován na standardní výstup, a to výhradně voláním funkce `printf`.

Pozdrav místo odmítnutí

Upravíme si program H (na H_1) tak, aby místo *ne* psal *Hello, world*.



Program H_1 získáme následující úpravou programu H :

Vypíše-li H jako první znak n , víme, že nakonec vypíše *ne*, můžeme tedy upravit odpovídající `printf` tak, aby rovnou vypsalo *Hello, world*, další `printf` už nic nevypisují.

Co řekne H_1 o sobě?



Co je program H_1 schopen říci sám o sobě?

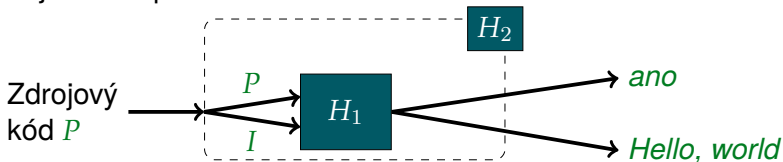


H_1 očekává na vstupu kódy programů s jedním vstupním souborem, ale H_1 sám očekává dva vstupní soubory.

H_1 musíme ještě upravit tak, aby očekával jen jeden vstupní soubor. Ten si vyloží jednak jako kód programu P k simulaci, jednak jako vstup I simulovaného programu.

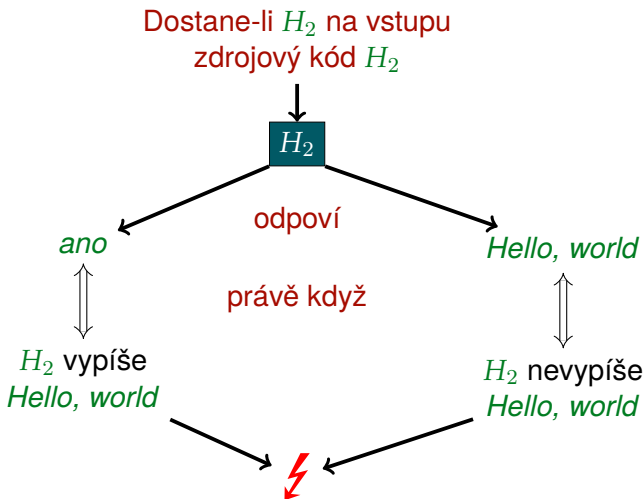
Dva vstupy v jednom

Program H_2 očekává jeden vstupní soubor, který předloží programu H_1 jako oba vstupní soubory, tedy jako zdrojový kód P i jako vstup I .



- 1 Program H_2 nejprve načte celý vstup a uloží jej v poli A , které alokuje v paměti (např. pomocí `malloc`).
- 2 Poté program H_2 simuluje práci H_1 , přičemž:
 - a Ve chvíli, kdy H_1 čte vstup (pomocí `scanf`), H_2 místo čtení přistoupí do pole A (tj. nahradí `scanf` pomocí čtení z A).
 - b Pomocí dvou ukazatelů do pole A si H_2 pamatuje, kolik z P a I program H_1 přečetl (`scanf` čte popořadě).

Pokud se H_2 zamyslí sám nad sebou



Co z toho vyplývá?

- ⇒ Program H_2 nemůže existovat.
- ⇒ Tedy ani program H_1 nemůže existovat.
- ⇒ Tedy ani program H nemůže existovat.
- ⇒ Problém **HELLOWORLD** nelze vyřešit žádným programem v jazyku C (a je tedy algoritmicky neřešitelný).

Volání funkce `foo`

Uvažme další problém:

Volání funkce `foo`

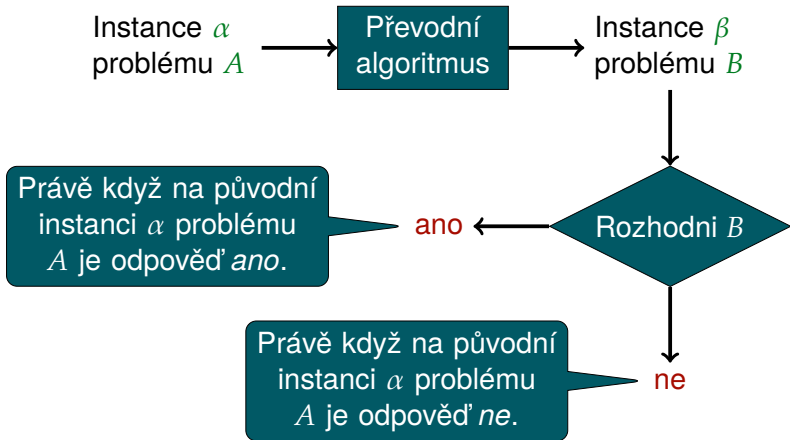
Instance: Zdrojový kód programu `Q` v jazyce C a vstup `V`.

Otázka: Zavolá program `Q` při běhu nad vstupem `V` funkci jménem `foo`?

- Chceme ukázat, že problém **VOLÁNÍ FUNKCE `foo`** je algoritmicky nerozhodnutelný.
- Ukážeme, že kdybychom uměli rozhodnout problém **VOLÁNÍ FUNKCE `foo`**, uměli bychom rozhodnout i problém **HELLOWORLD**.

Lehký úvod do převoditelnosti

Jsme-li pomocí problému B schopni vyřešit problém A , říkáme, že A je **převoditelný** na B .



Pozdrav voláním

- Převédeme problém **HELLOWORLD** na problém **VOLÁNÍ FUNKCE foo**.
- Popíšeme, jak převést instanci problému **HELLOWORLD** (program P a vstup I) na instanci problému **VOLÁNÍ FUNKCE foo** (program Q a vstup V).
- Musíme přitom zabezpečit, aby platilo, že

program P se vstupem I jako prvních dvanáct znaků svého výstupu vypíše *Hello, world*,

právě když

program Q se vstupem V zavolá funkci jménem `foo`.

- Problém **VOLÁNÍ FUNKCE foo** je tedy algoritmicky nerozhodnutelný.

Jak převést pozdrav na volání

Vstupem převodního algoritmu popsaného níže je program P a vstupní soubor I .

- 1 Je-li v P funkce `foo`, přejmenujeme ji i všechna její volání na dosud nepoužité jméno (refaktoring, výsledný program nazveme P_1).
- 2 K programu P_1 přidáme funkci `foo`, funkce nic nedělá a není volána ($\rightarrow P_2$).
- 3 Upravíme P_2 tak, aby si pamatoval prvních dvanáct znaků, které vypíše a uložil je v poli A ($\rightarrow P_3$).
- 4 Upravíme P_3 tak, že použije-li příkaz pro výstup, zkontroluje pole A , je-li prvních dvanáct znaků rovno *Hello, world*. Pokud ano, zavolá funkci `foo`.
- 5 Tím jsme zkonstruovali výsledný program Q , vstup $V = I$.

Nevýhody jazyka C pro teorii algoritmů

- Jazyk C je příliš komplikovaný.
- Museli bychom definovat výpočetní model (tj. zobecněný počítač), který bude programy v jazyce C interpretovat.
- V době vzniku teorie nebyly procedurální jazyky k dispozici, proto je teorie v literatuře obvykle popisovaná tradičnějšími prostředky.
- Potřebujeme výpočetní model dostatečně jednoduchý, aby jej bylo lze snadno popsat, současně dostatečně silný, aby byl schopen zachytit to, co intuitivně chápeme pod pojmem algoritmus.

Trocha historie ...

10. Hilbertův problém

V roce 1900 zformuloval David Hilbert 23 problémů, desátý z nich lze zformulovat takto:



Existuje postup, který by po konečném počtu operací zjistil, zda polynom více proměnných s celočíselnými koeficienty má celočíselný kořen?

Aby bylo možné zodpovědět tuto otázku, je potřeba mít formální definici pojmu algoritmu a efektivní vyčíslitelnosti.

Intuitivně: Algoritmus je konečná posloupnost jednoduchých instrukcí, která vede k řešení zadané úlohy.

Churchova teze

V roce 1934 navrhl Alonzo Church následující tezi:



Efektivně vyčíslitelné funkce jsou právě ty, které jsou definované v λ -kalkulu.

Tuto tezi později (1936) upravil na



Efektivně vyčíslitelné funkce jsou právě částečně rekurzivní funkce.

Turingova teze

V roce 1936 publikoval Alan Turing následující tezi



Ke každému algoritmu v intuitivním smyslu existuje ekvivalentní Turingův stroj.

- Zmíněné výpočetní modely (λ -kalkulus, částečně rekurzivní funkce, Turingovy stroje) jsou navzájem ekvivalentní co do výpočetní síly.
- Obvykle se této tezi říká **Churchova-Turingova**.

10. Hilbertův problém



Existuje postup, který by po konečném počtu operací zjistil, zda polynom více proměnných s celočíselnými koeficienty má celočíselný kořen?

V roce 1970 dal na tuto otázku Yuri Matijasevič negativní odpověď.



Neexistuje algoritmus, který by zjistil, zda daný polynom více proměnných s celočíselnými koeficienty má celočíselný kořen.

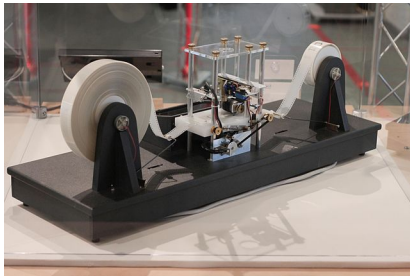
Ekvivalentní modely

Podle Churchovy-Turingovy teze je algoritmus ekvivalentní ...

- popisu Turingova stroje,
- programu pro RAM,
- odvození částečně rekurzivní funkce,
- odvození funkce v λ -kalkulu,
- programu ve vyšším programovacím jazyce, jako je C, Pascal, Java, Basic apod.,
- programu ve funkcionálním jazyce jako je Lisp, Haskell apod.

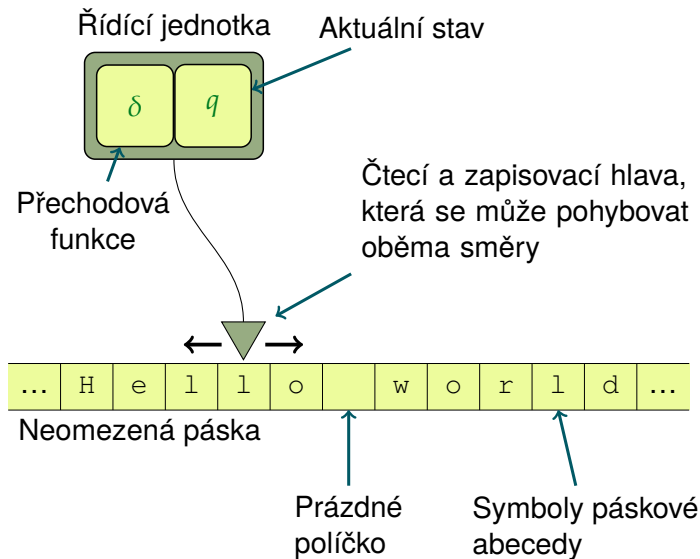
Ve všech těchto modelech jsme schopni počítat tytéž funkce, řešit tytéž problémy a úlohy.

Turingovy stroje



By Rocky Acosta — Own work, CC BY 3.0

Turingův stroj



Turingův stroj (definice)

(Jednopáskový deterministický) Turingův stroj (TS) M je pětice

$$M = (Q, \Sigma, \delta, q_0, F)$$

- Q je konečná množina stavů.
- Σ je konečná pásková abeceda, která obsahuje znak λ pro prázdné políčko.
 - Často budeme rozlišovat páskovou (vnitřní) a vstupní (vnější) abecedu.
- $\delta : Q \times \Sigma \mapsto Q \times \Sigma \times \{R, N, L\} \cup \{\perp\}$ je přechodová funkce, kde \perp označuje nedefinovaný přechod.
- $q_0 \in Q$ je počáteční stav.
- $F \subseteq Q$ je množina přijímajících stavů.

Konfigurace a displej Turingova stroje

- Turingův stroj sestává z
 - řídicí jednotky,
 - pásky, která je potenciálně nekonečná v obou směrech a
 - hlavy pro čtení a zápis, která se pohybuje oběma směry.
- **Displej** je dvojice (q, a) , kde $q \in Q$ je aktuální stav Turingova stroje a $a \in \Sigma$ je symbol pod hlavou.
 - Na základě displeje TS rozhoduje, jaký další krok má vykonat.
- **Konfigurace** zachycuje stav výpočtu Turingova stroje a skládá se ze
 - stavu řídicí jednotky,
 - slova na pásce (od nejlevějšího do nejpravějšího neprázdného políčka) a
 - pozice hlavy na pásce (v rámci slova na této pásce).

Výpočet Turingova stroje

- **Výpočet** zahajuje TS M v **počáteční konfiguraci**, tedy
 - v počátečním stavu,
 - se vstupním slovem zapsaným na pásce a
 - *Vstupní slovo nesmí obsahovat prázdné políčko.*
 - hlavou nad nejlevějším symbolem vstupního slova.
- Pokud se M nachází ve stavu $q \in Q$ a pod hlavou je symbol $a \in \Sigma$ a
- je-li $\delta(q, a) = \perp$, pak výpočet M končí,
- je-li $\delta(q, a) = (q', a', Z)$, kde $q' \in Q$, $a' \in \Sigma$ a $Z \in \{L, N, R\}$, pak M
 - přejde do stavu q' ,
 - zapíše na pozici hlavy symbol a' a
 - pohne hlavou doleva (pokud $Z = L$), doprava ($Z = R$), nebo hlava zůstane na místě ($Z = N$).

Slova a jazyky

- **Slovo** nad abecedou Σ je posloupnost znaků $w = a_1 a_2 \dots a_k$, kde $a_1, a_2, \dots, a_k \in \Sigma$.
- **Délku řetězce** $w = a_1 a_2 \dots a_k$ označujeme pomocí $|w| = k$.
- Množinu všech slov nad abecedou Σ označujeme pomocí Σ^* .
- **Prázdné slovo** označujeme pomocí ε .
- **Konkatenaci** slov w_1 a w_2 zapíšeme jako $w_1 w_2$.
- **Jazyk** $L \subseteq \Sigma^*$ je množina slov nad abecedou Σ .
- **Doplňěk jazyka** L označíme pomocí $\bar{L} = \Sigma^* \setminus L$.
- **Konkatenací** dvou jazyků L_1 a L_2 vznikne jazyk $L_1 \cdot L_2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$.
- **Kleeneho uzávěrem** jazyka L je jazyk $L^* = \{w \mid (\exists k \in \mathbb{N})(\exists w_1, \dots, w_k \in L)[w = w_1 w_2 \dots w_k]\}$.

Rozhodovací problém formalizujeme jako otázku, zda daná instance patří do jazyka kladných instancí.

Turingovsky rozhodnutelné jazyky

- TS M přijímá slovo w , pokud výpočet M se vstupem w skončí v přijímajícím stavu.
- TS M odmítá slovo w , pokud výpočet M se vstupem w skončí ve stavu, který není přijímajícím.
- Jazyk slov přijímaných TS M označíme pomocí $L(M)$.
- Fakt, že výpočet TS M nad vstupem w skončí, označíme pomocí $M(w) \downarrow$, budeme také říkat že výpočet konverguje.
- Fakt, že výpočet TS M nad vstupem w neskončí, označíme pomocí $M(w) \uparrow$, budeme také říkat že výpočet diverguje.
- Řekneme, že jazyk L je částečně (Turingovsky) rozhodnutelný (též rekurzivně spočetný), pokud existuje Turingův stroj M , pro který $L = L(M)$.
- Řekneme, že jazyk L je (Turingovsky) rozhodnutelný (též rekurzivní), pokud existuje Turingův stroj M , který se vždy zastaví a $L = L(M)$.

Turingovsky vyčíslitelné funkce

- Turingův stroj M s páskovou abecedou Σ počítá nějakou částečnou funkci $f_M : \Sigma^* \mapsto \Sigma^*$.
- Pokud $M(w) \downarrow$ pro daný vstup $w \in \Sigma^*$, je hodnota funkce $f_M(w)$ definovaná, což označíme pomocí $f_M(w) \downarrow$.
- Hodnotou funkce $f_M(w)$ je potom slovo na (výstupní) pásce M po ukončení výpočtu nad w .
- Pokud $M(w) \uparrow$, pak je hodnota $f_M(w)$ nedefinovaná, což označíme pomocí $f_M(w) \uparrow$.
- Funkce $f : \Sigma^* \mapsto \Sigma^*$ je **turingovsky vyčíslitelná**, pokud existuje Turingův stroj M , který ji počítá.



Každá turingovsky vyčíslitelná funkce má nekonečně mnoho různých Turingových strojů, které ji počítají!

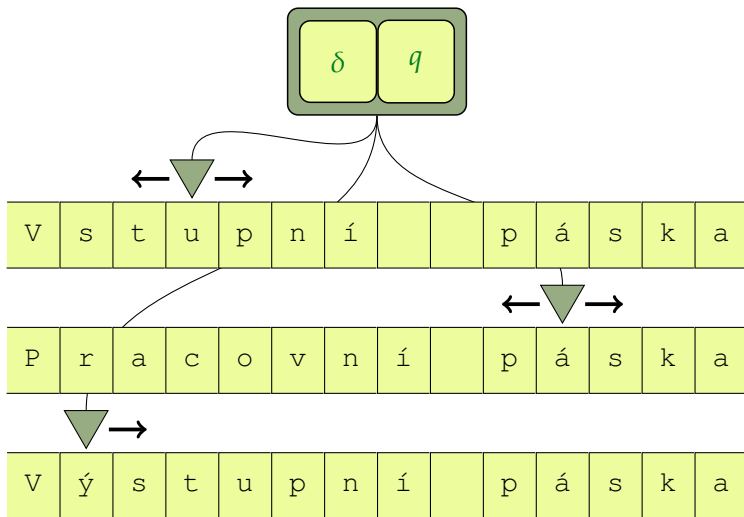
Varianty Turingových strojů

Turingovy stroje mají řadu variant, například

- TS s jednosměrně nekonečnou páskou.
- TS s více páskami (vstupní/výstupní/pracovní).
- TS s více hlavami na páskách,
- TS s pouze binární abecedou,
- nedeterministické TS.

Zmíněné varianty jsou ekvivalentní „našemu“ modelu v tom smyslu, že všechny přijímají touž třídu jazyků a vyčíslují touž třídu funkcí.

Struktura 3-páskového Turingova stroje



Vícepáskový Turingův stroj

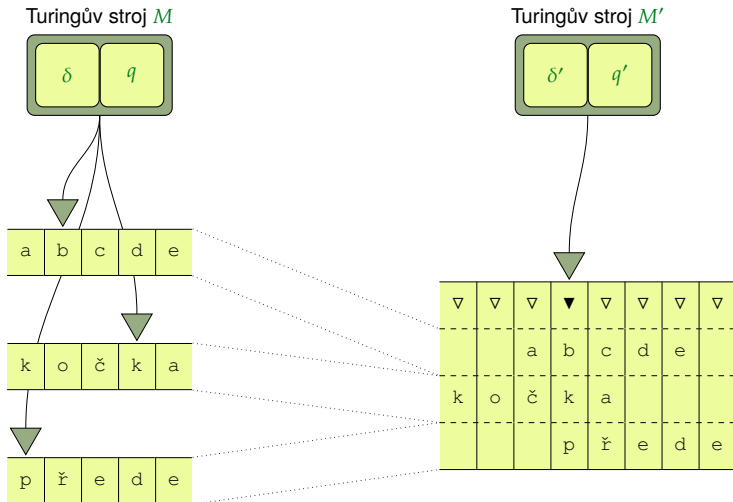
k -páskový Turingův stroj se od jednopáskového Turingova stroje liší následujícím způsobem:

- Má k pásek, na každé je zvláštní hlava.
Vstupní páska na počátku obsahuje vstupní řetězec
Často jen pro čtení.
Pracovní páska jsou určeny pro čtení i zápis.
Výstupní páska na konci obsahuje výstupní řetězec.
Často jen pro zápis s pohybem hlavy jen vpravo.
- Hlavy na páskách se pohybují nezávisle na sobě.
- Přechodová funkce je typu
$$\delta : Q \times \Sigma^k \mapsto Q \times \Sigma^k \times \{R, N, L\}^k \cup \{\perp\}.$$

Věta 1

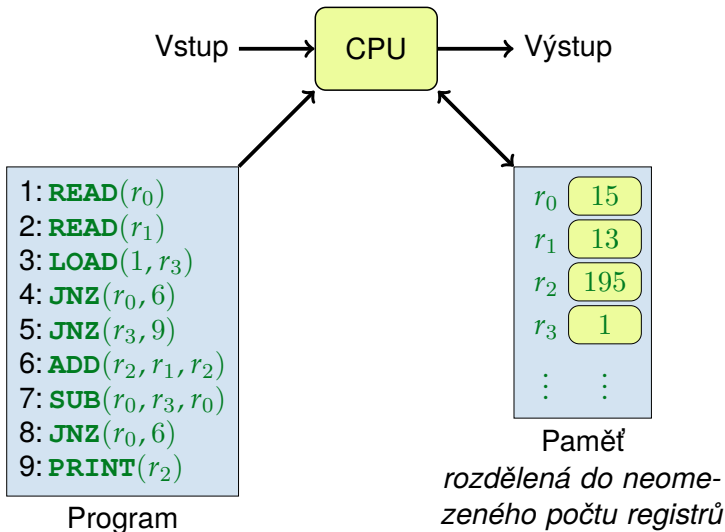
Ke každému k -páskovému Turingovu stroji M existuje jednopáskový Turingův stroj M' , který simuluje práci M , přijímá též jazyk jako M a počítá touž funkci jako M .

Reprezentace k pásek na jedné pásce



Random Access Machine

Random Access Machine (RAM)



Random Access Machine (definice)

- **Random Access Machine (RAM)**, stroj s náhodným přístupem do paměti) se skládá z
 - řídicí jednotky (procesoru, CPU) a
 - neomezené paměti.
- Paměť RAMu je rozdělená do **registrů**, které budeme označovat r_i , $i \in \mathbb{N}$.
- V každém registru může být libovolné přirozené číslo (na začátku je to 0).
- Obsah registru r_i označíme pomocí $[r_i]$.
- **Nepřímá adresace**: $[[r_i]] = [r_{[r_i]}]$.
- **Programem pro RAM** je konečná posloupnost instrukcí $P = I_0, I_1, \dots, I_\ell$.
- Instrukce jsou vykonávány v pořadí daném programem.

Možné instrukce RAM

Instrukce	Efekt
LOAD (C, r_i)	$r_i \leftarrow C$
ADD (r_i, r_j, r_k)	$r_k \leftarrow [r_i] + [r_j]$
SUB (r_i, r_j, r_k)	$r_k \leftarrow [r_i] \dot{-} [r_j]$
COPY ($[r_p], r_d$)	$r_d \leftarrow [[r_p]]$
COPY ($r_s, [r_d]$)	$r_{[r_d]} \leftarrow [r_s]$
JNZ (r_i, I_z)	if $[r_i] > 0$ then goto z
READ (r_i)	$r_i \leftarrow \text{input}$
PRINT (r_i)	$\text{output} \leftarrow [r_i]$

$$x \dot{-} y = \begin{cases} x - y & x > y \\ 0 & \text{jinak} \end{cases}$$

Jazyky rozhodnutelné RAMem

- Uvažme abecedu $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_k\}$.
- Slovo $w = \sigma_{i_1} \sigma_{i_2} \dots \sigma_{i_n}$ předáme RAMu R jako posloupnost čísel i_1, \dots, i_n .
- Konec slova pozná R díky tomu, že **READ** načte 0, není-li už k dispozici vstup.
- RAM R **přijme slovo** w , pokud $R(w) \downarrow$ a první číslo, které R zapíše na výstup je 1.
- RAM R **odmítne slovo** w , pokud $R(w) \downarrow$ a R buď na výstup nezapíše nic, nebo první zapsané číslo je jiné než 1.
- **Jazyk slov přijímaných RAMem** R označíme pomocí $L(R)$.
- Pokud pro jazyk L platí, že $L = L(R)$ pro nějaký RAM, pak řekneme, že je **částečně rozhodnutelný (RAMem)**.
- Pokud se navíc výpočet R nad každým vstupem zastaví, řekneme, že $L = L(R)$ je **rozhodnutelný (RAMem)**.

Funkce vyčíslitelné na RAMu

O RAMu R řekneme, že počítá částečnou aritmetickou funkci $f : \mathbb{N}^n \mapsto \mathbb{N}$, $n \geq 0$, pokud za předpokladu, že R dostane na vstup n -tici (x_1, \dots, x_n) , platí následující:

- Je-li $f(x_1, \dots, x_n) \downarrow$, pak $R(x_1, \dots, x_n) \downarrow$ a R vypíše na výstup hodnotu $f(x_1, \dots, x_n)$.
- Je-li $f(x_1, \dots, x_n) \uparrow$, pak $R(x_1, \dots, x_n) \uparrow$.

O funkci f , pro niž existuje RAM, který ji počítá, řekneme, že je **vyčíslitelná na RAMu**.

Řetězcové funkce vyčíslitelné na RAMu

RAM R počítá částečnou funkci $f : \Sigma^* \mapsto \Sigma^*$, kde $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_k\}$, pokud platí:

- Vstupní řetězec $w = \sigma_{i_1} \sigma_{i_2} \dots \sigma_{i_n}$ je předaný jako posloupnost čísel i_1, \dots, i_n .
- Konec slova pozná R díky tomu, že **READ** načte 0, není-li už k dispozici vstup.
- Pokud je $f(w) \downarrow = \sigma_{j_1} \sigma_{j_2} \dots \sigma_{j_m}$, pak $R(w) \downarrow$ a na výstup je zapsaná posloupnost čísel $j_1, j_2, \dots, j_m, 0$.
- Pokud $f(w) \uparrow$, pak $R(w) \uparrow$.

O funkci f , pro kterou existuje RAM R , který ji počítá, říkáme, že je **vyčíslitelná na RAMu**.

Programování na RAMu

Programy pro RAM odpovídají procedurálnímu jazyku:

- Máme k dispozici **proměnné** (**skalární** i **neomezená pole**).
- Cykly (**for** i **while**) — s pomocí podmíněného skoku, případně čítače v proměnné.
- Nepodmíněný skok (**goto**) — s použitím pomocného registru, kam uložíme **1** a použijeme podmíněný skok.
- **Podmíněný příkaz** — s pomocí podmíněného skoku.
- **Funkce** a **procedury** — do místa použití funkce rovnou v programu napíšeme tělo funkce (*inline*).
- **Nemáme rekurzivní volání funkcí** — Ta se však dají vždy nahradit pomocí cyklu **while** a **zásobníku**.

Proměnné v programu pro RAM

Předpokládejme, že v programu používáme pole A_1, \dots, A_p a skalární proměnné x_0, \dots, x_s .

- Pole indexujeme přirozenými čísly, tedy od 0.
- Prvek $A_i[j]$, kde $i \in \{1, \dots, p\}$, $j \in \mathbb{N}$, umístíme do registru $r_{i+j*(p+1)}$.
- Prvky pole A_i , $i = 1, \dots, p$ jsou tedy v registrech $r_i, r_{i+p+1}, r_{i+2(p+1)}, \dots$
- Proměnnou x_i , kde $i \in \{0, \dots, s\}$ umístíme do registru $r_{i*(p+1)}$.
- Skalární proměnné jsou tedy postupně v registrech $r_0, r_{p+1}, r_{2(p+1)}, \dots$

Turingův stroj \longrightarrow RAM

Věta 2

Ke každému Turingovu stroji M existuje ekvivalentní RAM R .

- Obsah pásky je uložen ve dvou polích:
 - T_r obsahuje pravou část pásky a
 - T_l obsahuje levou část pásky.
- Poloha hlavy — pamatujeme si index v proměnné h a stranu pásky (pravá/levá) v proměnné s .
- Stav — v proměnné q .
- Výběr instrukce — podmíněný příkaz podle h , s a q .

RAM \longrightarrow Turingův stroj

Věta 3

Ke každému RAMu R existuje ekvivalentní Turingův stroj M .

Obsah paměti R reprezentujeme na pásce M takto:

Jsou-li aktuálně využití registry $r_{i_1}, r_{i_2}, \dots, r_{i_m}$, kde $i_1 < i_2 < \dots < i_m$, pak je na pásce reprezentující paměť RAM R řetězec:

$$(i_1)_B | ([r_{i_1}])_B \# (i_2)_B | ([r_{i_2}])_B \# \dots \# (i_m)_B | ([r_{i_m}])_B$$

RAM \longrightarrow Turingův stroj (struktura TS)

K RAMu R sestrojíme TS M jako 4-páskový.

Vstupní páska posloupnost čísel, která má dostat R na vstup.
Jsou zakódovaná binárně a oddělená znakem $\#$.
Z této pásky M jen čte.

Výstupní páska sem zapisuje M čísla, která R zapisuje na výstup. Jsou zakódovaná binárně a oddělená znakem $\#$. Na tuto pásku M jen zapisuje.

Paměť RAM obsah paměti stroje R .

Pomocná páska pro výpočty součtu, rozdílu, nepřímých adres, posunu části paměťové pásky a podobně.

Číslování Turingových strojů

Jak očíslovat Turingovy stroje

Naším cílem je každému Turingovu stroji přiřadit číslo.

- 1 Turingův stroj popíšeme řetězcem nad malou abecedou.
- 2 Řetězec nad touto abecedou převedeme do binární abecedy.
- 3 Každému binárnímu řetězci přiřadíme číslo.
- 4 Ve výsledku takto každému Turingovu stroji přiřadíme číslo — tzv. **Gödelovo číslo**.

Pár technických omezení

Omezíme se na Turingovy stroje, které

- 1 mají **jediný přijímající stav** a
 - 2 mají pouze **binární vstupní abecedu** $\Sigma_{in} = \{0, 1\}$.
- Vstupní řetězce budou zapsány jen pomocí znaků 0 a 1.
 - Pracovní abecedu nijak neomezujeme — během výpočtu si Turingův stroj může na pásku zapisovat libovolné symboly.
 - Jakoukoli konečnou abecedu lze zakódovat do binární abecedy.
 - Každý TS M lze upravit tak, aby splňoval obě omezení.

Zakódování přechodové funkce

$$M = (Q, \Sigma, \delta, q_0, F = \{q_1\})$$



δ zapíšeme řetězcem v_M v abecedě
 $\Gamma = \{0, 1, L, N, R, |, \#, ;\}$



v_M

Každý znak v_M zapíšeme pomocí 3 bitů



$\langle M \rangle$

Binární řetězec reprezentující M

Zápis přechodové funkce v abecedě Γ

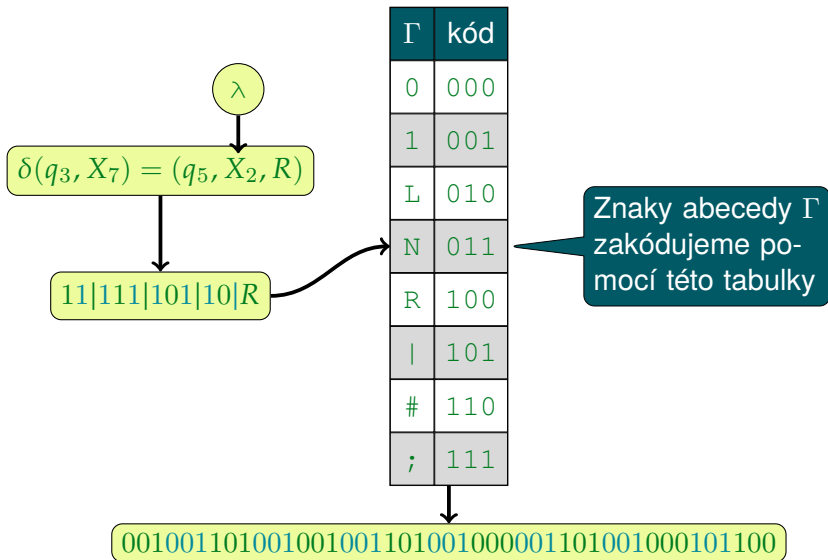
- Předpokládejme, že
 - $Q = \{q_0, q_1, \dots, q_r\}$ pro nějaké $r \geq 1$, kde q_0 je počáteční stav a q_1 je jediný přijímající stav.
 - $\Sigma = \{X_0, X_1, X_2, \dots, X_s\}$ pro nějaké $s \geq 2$, kde X_0 označuje znak 0, X_1 znak 1 a X_2 znak prázdného políčka λ .
- Instrukci $\delta(q_i, X_j) = (q_k, X_l, Z)$, kde $Z \in \{L, N, R\}$ zakódujeme řetězcem

$$(i)_B|(j)_B|(k)_B|(l)_B|Z.$$

- Jsou-li C_1, \dots, C_n kódy instrukcí TS M , pak přechodovou funkci δ zakódujeme řetězcem

$$C_1\#C_2\#\dots\#C_n.$$

Převod do binární abecedy



Číslování binárních řetězců

- Binárnímu řetězci $w \in \{0,1\}^*$ přiřadíme číslo i , jehož binární zápis je $1w$, tedy $(i)_B = 1w$.
- Řetězec s číslem i označíme pomocí w_i (tj. $(i)_B = 1w_i$).
- Tím dostaneme vzájemně jednoznačné zobrazení (tj. bijekci) mezi $\{0,1\}^*$ a kladnými přirozenými čísly.
- K tomu přidáme konvenci, že 0 odpovídá prázdnému řetězci (tj. $w_0 = w_1 = \varepsilon$).

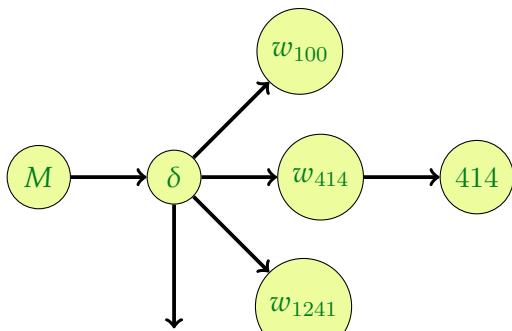
w_i	$1w_i$	i
ε	1	1
0	10	2
1	11	3
00	100	4
\vdots	\vdots	\vdots
001011	1001011	75
\vdots	\vdots	\vdots

Gödelovo číslo

- Každému Turingovu stroji M můžeme přiřadit Gödelovo číslo e , pro které platí, že řetězec w_e je kódem Turingova stroje M .
- Turingův stroj s Gödelovým číslem e označíme pomocí M_e .
- Pokud řetězec w_e není syntakticky správným kódem Turingova stroje, pak M_e je prázdným Turingovým strojem, který každý vstup okamžitě odmítne a $L(M_e) = \emptyset$.
- Z toho plyne, že ke každému číslu e jsme naopak schopni přiřadit nějaký Turingův stroj M_e .

Nejednoznačnost kódu TS

- Kód TS není jednoznačný, protože nezáleží na
 - pořadí instrukcí,
 - na očíslování stavů kromě počátečního a přijímajícího,
 - znaků páskové abecedy kromě 0, 1, λ , a
 - binární zápis čísla stavu nebo znaku může být uvozen libovolným počtem 0.
- Každý TS má **nekonečně mnoho různých kódů** a potažmo **nekonečně mnoho Gödelových čísel**.



Jedno z
Gödelových
čísel M

Kódování objektů (značení)

- Konečné objekty (např. číslo, řetězec, Turingův stroj, RAM, graf nebo formulí) můžeme kódovat do binárními řetězci.
- Podobně můžeme zakódovat i n -tice objektů.

Definice 4

- $\langle X \rangle$ označuje *kód objektu X pomocí binárního řetězce*.
- $\langle X_1, \dots, X_n \rangle$ označuje *kód n -tice objektů X_1, \dots, X_n* .

Například:

- Je-li M Turingův stroj, pak $\langle M \rangle$ označuje binární řetězec, který ho kóduje.
- Jsou-li M Turingův stroj a x je řetězec, pak $\langle M, x \rangle$ označuje kód dvojice M a x .

Univerzální Turingův stroj

Univerzální Turingův stroj

- Vstupem univerzálního Turingova stroje \mathcal{U} je kód dvojice $\langle M, x \rangle$, kde M je Turingův stroj a x je řetězec.
- \mathcal{U} simuluje práci stroje M nad vstupem x .
- Výsledek práce $\mathcal{U}(\langle M, x \rangle)$ (tj. zastavení/přijetí/zamítnutí vstupu a obsah výstupní pásky) je dán výsledkem $M(x)$.
- \mathcal{U} popíšeme jako 3-páskový, protože je to technicky jednodušší než konstrukce jednopáskového UTS.
- Převodem 3-páskového UTS na jednopáskový získáme jednopáskový UTS.
- Jazyku univerzálního Turingova stroje \mathcal{U} budeme říkat **univerzální jazyk** a budeme jej značit L_u , tedy

$$L_u = L(\mathcal{U}) = \{\langle M, x \rangle \mid x \in L(M)\}.$$

Struktura univerzálního Turingova stroje

1. páska obsahuje vstup \mathcal{U} , tedy kód $\langle M, x \rangle$.

$\langle M, x \rangle$

Na 2. pásce je uložen obsah pracovní pásky M . Symboly X_i jsou zapsány jako $(i)_B$ v blocích téže délky oddělených $|$.

... | 010 | 001 | 100 | 000 | 010 | 011 | ...

3. páska obsahuje číslo aktuálního stavu q_i stroje M .

10011 ($= (i)_B$)

Algoritmicky (ne)roz- hodnutelné jazyky

Definice 5

- Jazyk L je **částečně rozhodnutelný**, pokud existuje Turingův stroj M , který jej přijímá (tj. $L = L(M)$).
 - Jazyk L je **rozhodnutelný**, pokud existuje Turingův stroj M , který jej přijímá (tj. $L = L(M)$) a navíc se výpočet M zastaví s každým vstupem x (tj. $M(x) \downarrow$).
-
- Částečně rozhodnutelný jazyk = **rekurzivně spočetný jazyk**.
 - Rozhodnutelný jazyk = **rekurzivní jazyk**.

Základní vlastnosti rozhodnutelných jazyků

Věta 6

Jsou-li L_1 a L_2 (částečně) rozhodnutelné jazyky, pak $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 \cdot L_2$, L_1^* jsou (částečně) rozhodnutelné jazyky.

Věta 7 (Postova věta)

Jazyk L je rozhodnutelný, právě když L i \bar{L} jsou částečně rozhodnutelné jazyky.



- 1 Jsou všechny jazyky nad konečnou abecedou částečně rozhodnutelné?
- 2 Jsou všechny částečně rozhodnutelné jazyky rozhodnutelné?

Kolik je částečně rozhodnutelných jazyků?

Definice 8

Množina A je **spočetná**, pokud existuje prostá funkce $f : A \mapsto \mathbb{N}$, tj. pokud lze prvky A očíslovat.

- Je jen spočetně mnoho Turingových strojů — každý má Gödelovo číslo.
- Každý částečně rozhodnutelný jazyk je přijímán nějakým Turingovým strojem.

Lemma 9

Částečně rozhodnutelných jazyků je spočetně mnoho.

Jsou všechny jazyky rozhodnutelné?

Jazyk $L \subseteq \{0,1\}^*$ odpovídá množině přirozených čísel

$$A = \{i - 1 \mid i \in \mathbb{N} \setminus \{0\} \wedge w_i \in L\}.$$

- $\mathcal{P}(\mathbb{N})$ je nespočetná množina.
- Jazyků nad abecedou $\{0,1\}$ tedy **není spočetně mnoho**.



Musí proto existovat jazyky nad abecedou $\{0,1\}$, které nejsou ani částečně rozhodnutelné!

Dokonce by se dalo říct, že většina jazyků není ani částečně rozhodnutelná.

Diagonální jazyk

Diagonální jazyk definujeme takto:

$$\text{DIAG} = \{\langle M \rangle \mid \langle M \rangle \notin L(M)\}$$

Jeho doplněk je definován jako

$$\overline{\text{DIAG}} = \{\langle M \rangle \mid \langle M \rangle \in L(M)\}$$

Věta 10

- 1 Jazyk **DIAG** není částečně rozhodnutelný (jinými slovy není rekurzivně spočetný).
- 2 Jazyk $\overline{\text{DIAG}}$ je nerozhodnutelný, ale je částečně rozhodnutelný.

Univerzální jazyk

Rozhodnutí, zda dané slovo y patří do univerzálního jazyka L_u je formalizací **UNIVERZÁLNÍHO PROBLÉMU**:

Univerzální problém

Instance: Kód Turingova stroje M a vstup x .

Otázka: Je $x \in L(M)$? Tj. přijme Turingův stroj M vstup x ?

Věta 11

*Univerzální jazyk (tedy i **UNIVERZÁLNÍ PROBLÉM**) je částečně rozhodnutelný, ale není rozhodnutelný.*

Problém zastavení

- Klasickou ukázkou algoritmicky nerozhodnutelného problému je ovšem **PROBLÉM ZASTAVENÍ**.

Problém zastavení (Halting problem)

Instance: Kód Turingova stroje M a vstup x .

Otázka: Je $M(x) \downarrow$? Tj. zastaví se výpočet Turingova stroje M nad vstupem x ?

Věta 12

***PROBLÉM ZASTAVENÍ** je částečně rozhodnutelný, ale není rozhodnutelný.*

Algoritmicky vy- číslitelné funkce

Funkce — značení

Jsou-li $f, g : \Sigma^* \mapsto \Sigma^*$ dvě částečné funkce, pak

- **Doménou** funkce f je množina

$$\text{dom } f = \{x \in \Sigma^* \mid f(x) \downarrow\}$$

- **Oborem hodnot** funkce f je množina

$$\text{rng } f = \{y \in \Sigma^* \mid (\exists x \in \Sigma^*) [f(x) \downarrow = y]\}$$

- f a g jsou si **podmíněně rovny** ($f \simeq g$) pokud

$$f \simeq g \iff [\text{dom } f = \text{dom } g \text{ a } (\forall x \in \text{dom } f) [f(x) = g(x)]]$$

Algoritmicky vyčíslitelné funkce

Intuitivně: (Algoritmicky) vyčíslitelná funkce je funkce, jejíž hodnotu lze spočítat nějakým algoritmem.

Definice 13

- Částečná funkce $f : \Sigma^* \mapsto \Sigma^*$ je (**algoritmicky**) **vyčíslitelná** pokud je turingovsky vyčíslitelná.
- φ_e označuje funkci počítanou Turingovým strojem M_e .
- Vyčíslitelné funkce = **částečně rekurzivní funkce**.
- Totální vyčíslitelné funkce = **obecně rekurzivní funkce**.
- Uvažujeme i aritmetické funkce a funkce více parametrů, například funkce $f(x, y) = x^2 + y^2$ je realizována řetězcovou funkcí $f'(\langle x, y \rangle) = \langle x^2 + y^2 \rangle$.
- Vyčíslitelných funkcí je jen spočetně mnoho \implies ne všechny funkce jsou vyčíslitelné.

Věta 14

Univerzální funkce Ψ pro vyčíslitelné funkce je definována jako

$$\Psi(\langle e, x \rangle) \simeq \varphi_e(\langle x \rangle).$$

Tato funkce je algoritmicky vyčíslitelná.

...protože máme k dispozici univerzální Turingův stroj.

Vlastnosti (částečně) rozhodnutelných jazyků

Věta 15

Pro jazyk $L \subseteq \Sigma^*$ jsou následující tvrzení ekvivalentní:

- 1 L je částečně rozhodnutelný.
- 2 Existuje Turingův stroj M splňující

$$L = \{x \in \Sigma^* \mid M(x) \downarrow\}.$$

- 3 Existuje algoritmicky vyčíslitelná funkce f splňující

$$L = \text{dom } f = \{x \in \Sigma^* \mid f(x) \downarrow\}$$

- 4 Existuje rozhodnutelný jazyk B splňující

$$L = \{x \in \Sigma^* \mid (\exists y \in \Sigma^*)[\langle x, y \rangle \in B]\}$$

Věta 16

Jazyk $L \subseteq \Sigma^*$ je rozhodnutelný, právě když jeho *charakteristická funkce*

$$\chi_L(x) = \begin{cases} 1 & x \in L \\ 0 & x \notin L \end{cases}$$

je algoritmicky vyčíslitelná.

Definice 17 (Lexikografické uspořádání)

Nechť Σ je abeceda, předpokládejme, že $<$ je ostré uspořádání na znacích. Nechť $u, v \in \Sigma^*$ jsou dva různé řetězce. Řekneme, že u je *lexikograficky menší než* v , pokud

- 1 je u kratší (tj. $|u| < |v|$), nebo
- 2 mají oba řetězce touž délku (tj. $|u| = |v|$) a je-li i první index s $u[i] \neq v[i]$, pak $u[i] < v[i]$.

Tento fakt označíme pomocí $u < v$. Obvyklým způsobem rozšiřujeme značení i na $u \leq v$, $u > v$ a $u \geq v$.

Výčet prvků jazyka

Enumerátorem pro jazyk L je Turingův stroj E , který

- ignoruje svůj vstup,
- během výpočtu vypisuje řetězce $w \in L$ (např. oddělené znakem '#') na vyhrazenou výstupní pásku a
- každý řetězec $w \in L$ je někdy vypsán TS E .
- Je-li L nekonečný, E svou činnost nikdy neskončí.

Věta 18

- 1 Jazyk L je částečně rozhodnutelný, právě když pro něj existuje enumerátor E .
- 2 Jazyk L je rozhodnutelný, právě když pro něj existuje enumerátor E , který navíc vypisuje prvky L v lexikografickém pořadí.

Věta 19

Nechť L je nekonečný jazyk, potom jazyk L je ...

① ...*částečně rozhodnutelný, právě když je oborem hodnot nějaké totální algoritmicky vyčíslitelné funkce f (tj. $L = \text{rng } f$).*

② ...*rozhodnutelný, právě když je oborem hodnot nějaké rostoucí totální algoritmicky vyčíslitelné funkce f (tj. $L = \text{rng } f$).*

- Funkce $f : \Sigma^* \rightarrow \Sigma^*$ je **rostoucí**, pokud platí, že $u < v$ implikuje $f(u) < f(v)$ pro každé dva řetězce $u, v \in \Sigma^*$, kde $f(u) \downarrow$ a $f(v) \downarrow$.

Převoditelnost a úplnost

Definice 20

Jazyk A je m -**převoditelný** na jazyk B (což označíme pomocí $A \leq_m B$), pokud existuje totální vyčíslitelná funkce f splňující

$$(\forall x \in \Sigma^*) [x \in A \Leftrightarrow f(x) \in B]$$

Jazyk A je m -**úplný**, pokud je A částečně rozhodnutelný a každý částečně rozhodnutelný jazyk B je na něj m -převoditelný.

- **1-převoditelnost** a **1-úplnost** — navíc chceme, aby funkce f byla prostá.
- \leq_m je reflexivní a tranzitivní relace (**kvaziuspořádání**).
- Pokud $A \leq_m B$ a B je (částečně) rozhodnutelný jazyk, pak totéž lze říct o A .
- Pokud $A \leq_m B$, B je částečně rozhodnutelný jazyk a A je m -úplný jazyk, pak B je též m -úplný.

Problém zastavení můžeme formalizovat jako jazyk

$$\text{HALT} = \{ \langle M, x \rangle \mid M(x) \downarrow \}$$

Věta 21

Jazyky L_u , DIAG a HALT jsou m -úplné. Jde tedy o jazyky částečně rozhodnutelné, které nejsou rozhodnutelné.

Věta 22 (Riceova věta (jazyky))

Nechť C je třída částečně rozhodnutelných jazyků a položme $L_C = \{\langle M \rangle \mid L(M) \in C\}$. Potom je jazyk L_C rozhodnutelný, právě když je třída C buď prázdná nebo obsahuje všechny částečně rozhodnutelné jazyky.

Věta 23 (Riceova věta (funkce))

Nechť C je třída vyčíslitelných funkcí a položme $A_C = \{w_e \mid \varphi_e \in C\}$. Potom je jazyk A_C rozhodnutelný, právě když je třída C buď prázdná nebo obsahuje všechny vyčíslitelné funkce.

Riceova věta (důsledky)

Z Riceovy věty plyne, že následující jazyky nejsou rozhodnutelné:

$$\text{NONEMPTY} = \{\langle M \rangle \mid L(M) \neq \emptyset\}$$

$$\text{Fin} = \{\langle M \rangle \mid L(M) \text{ je konečný jazyk}\}$$

$$\text{Cof} = \{\langle M \rangle \mid \overline{L(M)} \text{ je konečný jazyk}\}$$

$$\text{Inf} = \{\langle M \rangle \mid L(M) \text{ je nekonečný jazyk}\}$$

$$\text{Dec} = \{\langle M \rangle \mid L(M) \text{ je rozhodnutelný jazyk}\}$$

$$\text{Tot} = \{\langle M \rangle \mid L(M) = \Sigma^*\}$$

$$\text{Reg} = \{\langle M \rangle \mid L(M) \text{ je regulární jazyk}\}$$

Postův korespondenční problém

Postův korespondenční problém

Instance: Množina „dominových kostek“ P :

$$P = \left\{ \left[\frac{t_1}{b_1} \right], \left[\frac{t_2}{b_2} \right], \dots, \left[\frac{t_k}{b_k} \right] \right\}$$

kde $t_1, \dots, t_k, b_1, \dots, b_k \in \Sigma^*$ jsou řetězce.

Otázka: Existuje **párovací posloupnost** i_1, i_2, \dots, i_l , kde $l \geq 1$ a $t_{i_1} t_{i_2} \dots t_{i_l} = b_{i_1} b_{i_2} \dots b_{i_l}$?

Theorem 24

POSTŮV KORESPONDENČNÍ PROBLÉM je nerozhodnutelný.

Věta 25 (s-m-n)

Pro každá dva přirozená čísla $m, n \geq 1$ existuje prostá totální vyčíslitelná funkce $s_n^m : \mathbb{N}^{m+1} \rightarrow \mathbb{N}$ taková, že pro každé $x, y_1, y_2, \dots, y_m, z_1, \dots, z_n \in \Sigma^*$ platí:

$$\varphi_{s_n^m(x, y_1, y_2, \dots, y_m)}^{(n)}(z_1, \dots, z_n) \simeq \varphi_x^{(m+n)}(y_1, \dots, y_m, z_1, \dots, z_n)$$

Složitost

Základní třídy složitosti

Rozhodovací problémy

- V rozhodovacím problému se ptáme, zda daná instance x splňuje danou podmínku.
- Odpověď je typu *ano/ne*.
- Rozhodovací problém formalizujeme jako jazyk $L \in \Sigma^*$ kladných instancí a otázku, zda $x \in L$.
- Příklady rozhodovacích problémů:
 - Je daný graf souvislý?
 - Má daná logická formule model?
 - Má daný lineární program přípustné řešení.
 - Je dané číslo prvočíslem?

Úlohy a optimalizační úlohy

- V **úloze** pro danou **instanci** x hledáme y , které splňuje určitou podmínku.
- Odpovědí je zde y nebo informace o tom, že žádné vhodné y neexistuje.
- Úlohu formalizujeme jako relaci $R \subseteq \Sigma^* \times \Sigma^*$.
- Příklady úloh:
 - Nalezení silně souvislých komponent orientovaného grafu.
 - Nalezení splňujícího ohodnocení logické formule.
 - Nalezení přípustného řešení lineárního programu.
- V **optimalizační úloze** navíc požadujeme, aby hodnota y byla maximální nebo minimální vzhledem k nějaké míře.
- Příklady optimalizačních úloh:
 - Nalezení maximálního toku v síti.
 - Nalezení nejkratší cesty v grafu.
 - Nalezení optimálního řešení lineárního programu.

Definice 26

Nechť M je (deterministický) Turingův stroj a nechť $f : \mathbb{N} \mapsto \mathbb{N}$ je funkce, která je definovaná pro každý vstup.

- Řekneme, že M **pracuje v čase** $f(n)$, pokud výpočet M nad libovolným vstupem x délky $|x| = n$ skončí po provedení nejvýše $f(n)$ kroků.
- Řekneme, že M **pracuje v prostoru** $f(n)$, pokud výpočet M nad libovolným vstupem x délky $|x| = n$ skončí a využije nejvýš $f(n)$ buněk pracovní pásky.

Základní deterministické třídy složitosti

Definice 27

Nechť $f : \mathbb{N} \mapsto \mathbb{N}$ je funkce, potom definujeme třídy:

$\text{TIME}(f(n))$ *třída jazyků přijímaných Turingovými stroji, které pracují v čase $O(f(n))$.*

$\text{SPACE}(f(n))$ *třída jazyků přijímaných Turingovými stroji, které pracují v prostoru $O(f(n))$.*

- Triviálně platí, že $\text{TIME}(f(n)) \subseteq \text{SPACE}(f(n))$ pro každou funkci $f : \mathbb{N} \mapsto \mathbb{N}$.

Význačné deterministické třídy složitosti

Definice 28

Třída *problémů řešitelných v polynomiálním čase*:

$$P = \bigcup_{k \in \mathbb{N}} \text{TIME}(n^k)$$

Třída *problémů řešitelných v polynomiálním prostoru*:

$$\text{PSPACE} = \bigcup_{k \in \mathbb{N}} \text{SPACE}(n^k).$$

Třída *problémů řešitelných v exponenciálním čase*:

$$\text{EXPTIME} = \bigcup_{k \in \mathbb{N}} \text{TIME}(2^{n^k}).$$

Proč polynomy?

Teze 29 (Silnější verze Churchovy-Turingovy teze)

Reálné výpočetní modely lze simulovat na Turingovu stroji s polynomiálním zpomalením/nárůstem prostoru.

- Polynomy jsou uzavřeny na skládání.
- Polynomy (obvykle) nerostou příliš rychle.
- Definice třídy P nezávisí na zvoleném výpočetním modelu.

Teze 30 (Cobhamova-Edmondsova teze, 1965)

P odpovídá třídě prakticky řešitelných problémů na počítači.

Definice 31

Verifikátorem pro jazyk A je algoritmus V , pro který platí, že

$$A = \{x \mid (\exists y)[V \text{ přijme } \langle x, y \rangle]\}.$$

- Řetězec y zoveme také *certifikátem* x .
- Časovou složitost verifikátoru měříme vzhledem k $|x|$.
- *Polynomiální verifikátor* je takový, který pracuje v polynomiálním čase vzhledem k $|x|$.
- Pokud polynomiální verifikátor V přijímá $\langle x, y \rangle$, pak y má nutně délku polynomiální vzhledem k x .
- Řetězec y je pak zván *polynomiálním certifikátem* x .

Definice 32

NP je třídou *jazyků, které mají polynomiální verifikátory*.

- Odpovídá třídě úloh, u nichž jsme schopni v polynomiálním čase ověřit, že daný řetězec y je řešením, i když jej nejsme nutně schopni v polynomiálním čase najít.
- Jazyky v třídě **NP** lze také charakterizovat pomocí nedeterministických Turingových strojů, jež pracují v polynomiálním čase.
- Nedeterminismus zde odpovídá „hádání“ správného certifikátu y vstupu x .

Nedeterministický Turingův stroj

Nedeterministický Turingův stroj (NTS) je pětice

$M = (Q, \Sigma, \delta, q_0, F)$, kde

- Q, Σ, q_0, F mají též význam jako u „obyčejného“ deterministického Turingova stroje (DTS).
- Rozdíl oproti DTS je v přechodové funkci, nyní

$$\delta : Q \times \Sigma \mapsto \mathcal{P}(Q \times \Sigma \times \{L, N, R\}).$$

- Možné představy
 - NTS M v každém kroku „uhodne“ nebo „vybere“ správnou instrukci.
 - NTS M vykonává všechny možné instrukce současně a nachází se během výpočtu ve více konfiguracích současně.
- Nedeterministický Turingův stroj není reálný výpočetní model ve smyslu silnější Churchovy-Turingovy teze.

Jazyk přijímaný NTS

- Výpočet NTS M nad slovem x je posloupnost konfigurací C_0, C_1, C_2, \dots , kde
 - C_0 je počáteční konfigurace a
 - z C_i do C_{i+1} lze přejít pomocí přechodové funkce δ .
- Výpočet je přijímající, pokud je konečný a v poslední konfiguraci výpočtu se M nachází v přijímajícím stavu.
- Slovo x je přijato NTS M pokud existuje přijímající výpočet M nad x .
- Jazyk slov přijímaných NTS M označíme pomocí $L(M)$.

Definice 33

Nechť M je nedeterministický Turingův stroj a nechť $f : \mathbb{N} \mapsto \mathbb{N}$ je funkce.

- Řekneme, že M **pracuje v čase** $f(n)$, pokud **každý výpočet** M nad libovolným vstupem x délky $|x| = n$ skončí po provedení nejvýše $f(n)$ kroků.
- Řekneme, že M **pracuje v prostoru** $f(n)$, pokud **každý výpočet** M nad libovolným vstupem x délky $|x| = n$ skončí a využije nejvýše $f(n)$ buněk pracovní pásky.

Základní nedeterministické třídy složitosti

Definice 34

Nechť $f : \mathbb{N} \mapsto \mathbb{N}$ je funkce, potom definujeme třídy:

$\text{NTIME}(f(n))$ *třída jazyků přijímaných nedeterministickými TS, které pracují v čase $O(f(n))$.*

$\text{NSPACE}(f(n))$ *třída jazyků přijímaných nedeterministickými TS, které pracují v prostoru $O(f(n))$.*

Věta 35

Pro každou funkci $f : \mathbb{N} \mapsto \mathbb{N}$ platí

$$\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$$

NP=nedeterministicky polynomiální

Věta 36 (Alternativní definice třídy NP)

Třída NP je třída jazyků přijímaných nedeterministickými Turingovými stroji v polynomiálním čase, tj.

$$\text{NP} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k).$$

Model TS s menším než lineárním prostorem

Pro prostor menší než lineární uvažujeme vícepáskový TS:

Vstupní páska je **jen pro čtení**

Pracovní pásy jsou pro **čtení i zápis**

Výstupní páska je **jen pro zápis** a hlava se hýbe jen vpravo

- Do prostoru se počítá jen obsah pracovních pásek.
- Součástí konfigurace je
 - stav,
 - poloha hlavy na vstupní pásce,
 - polohy hlav na pracovních páskách a
 - obsah pracovních pásek.
- Konfigurace neobsahuje vstupní slovo.

Definice 37

$$\begin{aligned}L &= \text{SPACE}(\log_2 n) \\NL &= \text{NSPACE}(\log_2 n) \\ \text{NPSPACE} &= \bigcup_{k \in \mathbb{N}} \text{NSPACE}(n^k)\end{aligned}$$

Vztah mezi prostorem a časem

Věta 38

Nechť $f(n)$ je funkce, pro kterou platí $f(n) \geq \log_2 n$. Pro každý jazyk L platí, že

$$L \in \text{NSPACE}(f(n)) \Rightarrow (\exists c_L \in \mathbb{N}) [L \in \text{TIME}(2^{c_L f(n)})] .$$

Důsledek 39

Je-li $f(n)$ funkce, pro kterou platí $f(n) \geq \log_2 n$ a je-li $g(n)$ funkce, pro kterou platí $f(n) = o(g(n))$, pak

$$\text{NSPACE}(f(n)) \subseteq \text{TIME}(2^{g(n)}) .$$

Věta 40

Platí následující inkluze

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq NPSPACE \subseteq EXPTIME.$$

Savičova věta

Věta 41 (Savičova věta)

Pro každou funkci $f(n) \geq \log_2 n$ platí, že

$$\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$$

Důsledek 42

$$\text{PSPACE} = \text{NPSPACE}$$

Věty o hierarchii

Věta o deterministické prostorové hierarchii

Definice 43

Funkci $f : \mathbb{N} \rightarrow \mathbb{N}$, kde $f(n) \geq \log n$, nazveme **prostorově konstruovatelnou**, je-li funkce, která zobrazuje 1^n na binární reprezentaci $f(n)$ vyčíslitelná v prostoru $O(f(n))$.

Věta 44 (Věta o deterministické prostorové hierarchii)

Pro každou prostorově konstruovatelnou funkci $f : \mathbb{N} \rightarrow \mathbb{N}$ existuje jazyk A , který je rozhodnutelný v prostoru $O(f(n))$, nikoli však v prostoru $o(f(n))$.

Důsledek 45

- ① Jsou-li $f_1, f_2 : \mathbb{N} \rightarrow \mathbb{N}$ funkce, pro které platí, že $f_1(n) \in o(f_2(n))$ a f_2 je prostorově konstruovatelná, potom

$$\text{SPACE}(f_1(n)) \subsetneq \text{SPACE}(f_2(n)).$$

- ② Pro každá dvě reálná čísla $0 \leq \epsilon_1 < \epsilon_2$ platí, že

$$\text{SPACE}(n^{\epsilon_1}) \subsetneq \text{SPACE}(n^{\epsilon_2}).$$

- ③ $\text{NL} \subsetneq \text{PSPACE} \subsetneq \text{EXPSPACE} = \bigcup_{k \in \mathbb{N}} \text{SPACE}(2^{n^k})$.

Věta o deterministické časové hierarchii

Definice 46

Funkci $f : \mathbb{N} \rightarrow \mathbb{N}$, kde $f(n) \in \Omega(n \log n)$, nazveme **časově konstruovatelnou**, je-li funkce, která zobrazuje 1^n na binární reprezentaci $f(n)$ vyčíslitelná v čase $O(f(n))$.

Věta 47 (Věta o deterministické časové hierarchii)

Pro každou časově konstruovatelnou funkci $f : \mathbb{N} \rightarrow \mathbb{N}$ existuje jazyk A , který je rozhodnutelný v čase $O(f(n))$, nikoli však v čase $o(f(n)/\log f(n))$.

Důsledek 48

- ① Jsou-li $f_1, f_2 : \mathbb{N} \rightarrow \mathbb{N}$ funkce, pro které platí, že $f_1(n) \in o(f_2(n)/\log f_2(n))$ a f_2 je časově konstruovatelná, potom

$$\text{TIME}(f_1(n)) \subsetneq \text{TIME}(f_2(n)).$$

- ② Pro každá dvě reálná čísla $1 \leq \epsilon_1 < \epsilon_2$,

$$\text{TIME}(n^{\epsilon_1}) \subsetneq \text{TIME}(n^{\epsilon_2}).$$

- ③ $P \subsetneq \text{EXPTIME}$.

Polynomiální převoditelnost a NP-úplnost

Definice 49

Jazyk A je *převoditelný v polynomiálním čase* (*polynomiálně převoditelný*) na jazyk B , psáno $A \leq_m^P B$, pokud existuje funkce $f : \Sigma^* \mapsto \Sigma^*$ vyčíslitelná v polynomiálním čase, pro kterou platí

$$(\forall w \in \Sigma^*) [w \in A \iff f(w) \in B].$$

- \leq_m^P je reflexivní a tranzitivní relace (*kvaziuspořádání*).
- Pokud $A \leq_m^P B$ a $B \in P$, pak $A \in P$.
- Pokud $A \leq_m^P B$ a $B \in NP$, pak $A \in NP$.

Definice 50

- Jazyk B je **NP-těžký**, pokud je na něj polynomiálně převoditelný každý problém $A \in \text{NP}$.
- **NP-těžký** jazyk B , který navíc patří do **NP** zvine **NP-úplným**.
- Pokud chceme ukázat, že nějaký problém B je **NP-úplný**, pak stačí
 - 1 ukázat $B \in \text{NP}$ a
 - 2 najít jiný **NP-úplný** problém A a převést jej na B (tj. $A \leq_m^P B$).



Za předpokladu $P \neq \text{NP}$ platí, že pokud B je **NP-úplný** problém, pak $B \notin P$.

Kachlíkování (Tiling)

Instance: Množina barev B , přirozené číslo s , čtvercová mřížka S o rozměrech $s \times s$, v níž jsou vnější hrany krajních buněk obarveny barvami z B , množina typů kachlíků K , každý má tvar čtverce s okraji obarvenými barvami z B .

Otázka: Je možné buňkám S přiřadit typy kachlíků z K (bez otáčení) tak, aby sousední kachlíky měly shodnou barvu na sdílené hraně a aby kachlíky v krajních buňkách měly odpovídající okrajovou barvu?

Věta 51

KACHLÍKOVÁNÍ je NP-úplný problém.

Splnitelnost

Literál proměnná (např. x) nebo její negace (např. \bar{x}).

Klauzule disjunkce literálů.

Konjunktivní normální forma (KNF) Formule je v KNF, pokud jde o konjunkci klauzulí.

Splnitelnost (SAT)

Instance: Formule φ v KNF

Otázka: Existuje ohodnocení proměnných v , pro které je $\varphi(v)$ splněno?

Věta 52 (Cookova-Levinova věta)

*Pokud by byla **SPLNITELNOST** řešitelná v polynomiálním čase, pak by se $P = NP$. Přesněji, **SPLNITELNOST** je **NP**-úplný problém.*

3-Splnitelnost

- Formule φ je v **3-KNF**, pokud se skládá z klauzulí, z nichž každá obsahuje právě tři literály.

3-Splnitelnost (3SAT)

Instance: Formule φ v 3-KNF.

Otázka: Existuje ohodnocení proměnných v , pro které je $\varphi(v)$ splněno?

Věta 53

3-SPLNITELNOST je NP-úplný problém.

- 2-SPLNITELNOST** je již polynomiálně řešitelná.

Vrcholové pokrytí (VP, Vertex Cover)

Instance: Neorientovaný graf $G = (V, E)$, přirozené číslo k .

Otázka: Existuje množina vrcholů S , která má neprázdný průnik s každou hranou grafu G a která má velikost nejvýš k ? *Množina vrcholů S tedy „pokrývá“ všechny hrany.*

Věta 54 (Bez důkazu)

VRCHOLOVÉ POKRYTÍ je NP-úplný problém.

Vrcholové pokrytí (souvislosti)

- NP-úplné problémy související s VRCHOLOVÝM POKRYTÍM:
 - KLIKA (CLIQUE): Obsahuje G jako podgraf kliku, tj. úplný graf, na k vrcholech?
 - NEZÁVISLÁ MNOŽINA (INDEPENDENT SET): Obsahuje G nezávislou množinu velikosti k ? (Množina vrcholů S je nezávislá, pokud mezi žádnými dvěma vrcholy z S nevede hrana.)
- Analogický problém HRANOVÉHO POKRYTÍ (EDGE COVER), kde hledáme co nejmenší množinu hran, jež pokrývá všechny vrcholy, je polynomiálně řešitelný.

Hamiltonovská kružnice

Hamiltonovská kružnice (HK, Hamiltonian cycle)

Instance: Neorientovaný graf $G = (V, E)$.

Otázka: Existuje v grafu G cyklus vedoucí přes všechny vrcholy?

Věta 55 (Bez důkazu)

HAMILTONOVSKÁ KRUŽNICE je NP-úplný problém.

Obchodní cestující

Obchodní cestující (OC, Traveling salesperson)

Instance: Množina měst $C = \{c_1, \dots, c_n\}$, hodnoty $d(c_i, c_j) \in \mathbb{N}$ přiřazující každé dvojici měst vzdálenost a přirozené číslo D .

Otázka: Existuje permutace měst $c_{\pi(1)}, c_{\pi(2)}, \dots, c_{\pi(n)}$, pro kterou platí, že

$$\left(\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) \right) + d(c_{\pi(n)}, c_{\pi(1)}) \leq D?$$

Věta 56

OBCHODNÍ CESTUJÍCÍ je NP-úplný problém.

Trojrozměrné párování

Trojrozměrné párování (3DM, 3D Matching)

Instance: Množina $M \subseteq W \times X \times Y$, kde W , X a Y jsou množiny velikosti q .

Otázka: Má M perfektní párování? Tj. existuje množina velikosti q , která neobsahuje dvojici trojic, jež by se shodovaly v nějaké souřadnici?

Věta 57 (Bez důkazu)

TROJROZMĚRNÉ PÁROVÁNÍ je NP-úplný problém.

Loupežníci (Partition)

Instance: Množina předmětů A , s každým předmětem $a \in A$ asociované přirozené číslo $s(a)$ (váha, cena, velikost).

Otázka: Existuje $A' \subseteq A$, pro kterou platí, že

$$\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)?$$

Věta 58

LOUPEŽNÍCI je NP-úplný problém.

Batoh (Knapsack)

Instance: Množina předmětů A , s každým předmětem $a \in A$ asociovaná velikost $s(a) \in \mathbb{N}$ a cena $v(a) \in \mathbb{N}$, velikost batohu $B \in \mathbb{N}$ a limit na cenu $K \in \mathbb{N}$.

Otázka: Lze vybrat množinu předmětů $A' \subseteq A$ tak, aby platilo

$$\sum_{a \in A'} s(a) \leq B \text{ a } \sum_{a \in A'} v(a) \geq K?$$

Věta 59

BATOH je NP-úplný problém.

- NP-těžkost lze ukázat snadným převodem z LOUPEŽNÍKŮ.

Rozvrhování (Scheduling)

Instance: Množina úloh U , s každou úlohou $u \in U$ asociovaná doba zpracování $d(u) \in \mathbb{N}$, počet procesorů m , limit $D \in \mathbb{N}$.

Otázka: Lze úlohy U rozdělit na m procesorů tak, aby byly všechny úlohy zpracované v časovém limitu D ?

Věta 60

ROZVRHOVÁNÍ je NP-úplný problém.

- NP-těžkost lze ukázat snadným převodem z LOUPEŽNÍKŮ.

Pseudopolynomiální algoritmy a silná NP-úplnost

Batoh (optimalizační verze)

Batoh (Knapsack)

Instance: Množina předmětů A , s každým předmětem $a \in A$ asociovaná velikost $s(a) \in \mathbb{N}$ a cena $v(a) \in \mathbb{N}$, velikost batohu $B \in \mathbb{N}$.

Přípustné řešení: Množina předmětů $A' \subseteq A$, pro kterou platí

$$\sum_{a \in A'} s(a) \leq B$$

Cíl: Maximalizovat celkovou cenu předmětů v A' , tedy $\sum_{a \in A'} v(a)$.

Pseudopolynomiální algoritmus pro Batoh (1)

Vstup: Velikost batohu B , počet předmětů n . Pole velikostí s a pole cen v (obě délky n). Předpokládáme, že $(\forall i)[0 \leq s(i) \leq B]$.

Výstup: Množina předmětů A' s celkovou velikostí nejvýš B a s maximální cenou.

- 1: $V \leftarrow \sum_{i=1}^n v[i]$
- 2: T je nová matice typu $(n + 1) \times (V + 1)$, kde $T[j, c]$ bude na konci obsahovat množinu prvků z $\{1, \dots, j\}$ s cenou rovnou c a minimální celkovou velikostí předmětů.
- 3: S je nová matice typu $(n + 1) \times (V + 1)$, kde $S[j, c]$ bude na konci obsahovat součet velikostí předmětů v $T[j, c]$ nebo $B + 1$, pokud v $T[j, c]$ není žádná množina.

Pseudopolynomiální algoritmus pro Batoh (2)

```
4:  $T[0, 0] \leftarrow \emptyset, S[0, 0] \leftarrow 0$ 
5: for  $c \leftarrow 1$  to  $V$  do
6:    $T[0, c] \leftarrow \emptyset, S[0, c] \leftarrow B + 1$ 
7: end for
8: for  $j \leftarrow 1$  to  $n$  do
9:    $T[j, 0] \leftarrow \emptyset, S[j, 0] \leftarrow 0$ 
10:  for  $c \leftarrow 1$  to  $V$  do
11:     $T[j, c] \leftarrow T[j - 1, c], S[j, c] \leftarrow S[j - 1, c]$ 
12:    if  $v[j] \leq c$  and  $S[j, c] > S[j - 1, c - v[j]] + s[j]$  then
13:       $T[j, c] \leftarrow T[j - 1, c - v[j]] \cup \{j\}$ 
14:       $S[j, c] \leftarrow S[j - 1, c - v[j]] + s[j]$ 
15:    end if
16:  end for
17: end for
18:  $c \leftarrow \max\{c' \mid S[n, c'] \leq B\}$ 
19: return  $T[n, c]$ 
```

Pseudopolynomiální algoritmus pro Batoh (3)

- Popsaný algoritmus pracuje v čase $\Theta(nV)$ (počítáme-li aritmetické operace jako konstantní).
- Algoritmus obecně nepracuje v polynomiálním čase, neboť velikost vstupu je $O(n \log_2(B + V))$.
- Algoritmu tohoto typu budeme říkat **pseudopolynomiální**.

Definice 61

Nechť A je libovolný rozhodovací problém a I nechť je instance tohoto problému. Potom

$\text{len}(I)$ označuje délku zakódování instance I při standardním binárním kódování.

$\text{max}(I)$ označuje hodnotu největšího číselného parametru, který se vyskytuje v I .

Řekneme, že A je **číselný problém**, pokud pro každý polynom p existuje instance I tohoto problému taková, že $\text{max}(I) > p(\text{len}(I))$.

- Například **BATOH** nebo **LOUPEŽNÍCI** jsou číselné problémy.
- Problémy **SPLNITELNOST** nebo **KACHLÍKOVÁNÍ** číselné nejsou.

Pseudopolynomiální algoritmus

Definice 62

Řekneme, že algoritmus, který řeší problém A je *pseudopolynomiální*, pokud je jeho časová složitost omezena polynomem dvou proměnných $\text{len}(I)$ a $\text{max}(I)$.

- Obvykle měříme časovou složitost jen vzhledem k $\text{len}(I)$.
- Pokud by existoval polynom p , pro který by platilo, že $\text{max}(I) \leq p(\text{len}(I))$ (pro každou instanci), stal by se pseudopolynomiální algoritmus polynomiálním.
- Jiný pohled je te, že pseudopolynomiální algoritmus by byl polynomiální, pokud bychom předali vstup zakódovaný unárně.

Příklady pseudopolynomiálních algoritmů

- Eratosthenovo síto
- Naivní algoritmus pro faktORIZaci
- Counting sort

Definice 63

- Nechť A je rozhodovací problém a p je polynom. Pomocí $A(p)$ označíme restrikci problému A na instance I , pro něž platí $\max(I) \leq p(\text{len}(I))$.
- Řekneme, že problém A je **silně NP-úplný**, pokud existuje polynom p , pro který $A(p)$ je NP-úplný problém.
- Každý nečíselný NP-úplný problém je silně NP-úplný.
- Pokud by existoval silně NP-úplný problém, který lze vyřešit pseudopolynomiálním algoritmem, znamenalo by to, že $P = NP$.

Binární vs. unární kódování

- Pseudopolynomiální=polynomiální při unárním kódování.
- Silně NP-úplný=NP-úplný i při unárním kódování.

Binární kódování	Unární kódování
P	Řešitelné pseudopolynomiálním algoritmem.
NP-úplné	Silně NP-úplné.

Silná NP-úplnost Obchodního cestujícího

Obchodní cestující (OC, Traveling salesperson)

Instance: Množina měst $C = \{c_1, \dots, c_n\}$, hodnoty $d(c_i, c_j) \in \mathbb{N}$ přiřazující každé dvojici měst vzdálenost a přirozené číslo D .

Otázka: Existuje permutace měst $c_{\pi(1)}, c_{\pi(2)}, \dots, c_{\pi(n)}$, pro kterou platí, že

$$\left(\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) \right) + d(c_{\pi(n)}, c_{\pi(1)}) \leq D?$$

Věta 64

Problém OBCHODNÍHO CESTUJÍCÍHO je silně NP-úplný.

Aproximační algoritmy

Definice 65

- *Optimalizační úlohu* definujeme jako trojici $A = (D_A, S_A, \mu_A)$, kde
 - $D_A \subseteq \Sigma^*$ je množina instancí,
 - $S_A(I)$ přiřazuje instanci $I \in D_A$ množinu přípustných řešení,
 - $\mu_A(I, \sigma)$ přiřazuje instanci $I \in D_A$ a přípustnému řešení $\sigma \in S_A(I)$ kladné racionální číslo (hodnotu řešení).
- Je-li A maximalizační úloha, pak optimálním řešením instance I je to přípustné řešení $\sigma \in S_A(I)$, jež má maximální hodnotu $\mu_A(I, \sigma)$.
- Je-li A minimalizační úloha, pak optimálním řešením instance I je to přípustné řešení $\sigma \in S_A(I)$, jež má minimální hodnotu $\mu_A(I, \sigma)$.
- Hodnotu optimálního řešení označíme pomocí $\text{opt}(I)$.

Bin Packing

Bin Packing (BP)

Instance: Množina předmětů U , s každým předmětem $u \in U$ asociovaná velikost $s(u)$, což je racionální číslo z intervalu $\langle 0, 1 \rangle$.

Přípustné řešení: Rozdělení předmětů do po dvou disjunktních množin U_1, \dots, U_m , pro které platí, že

$$(\forall i \in \{1, \dots, m\}) \left[\sum_{u \in U_i} s(u) \leq 1 \right].$$

Cíl: Minimalizovat počet košů m .

Rozhodovací verze **BIN PACKING** je shodná s **ROZVRHOVÁNÍM**.

Definice 66

Algoritmus R nazveme **aproximačním algoritmem** pro optimalizační úlohu A , pokud pro každou instanci $I \in D_A$ je výstupem $R(I)$ přípustné řešení $\sigma \in S_A(I)$ (pokud nějaké existuje).

- Je-li A maximalizační úloha, pak $\varepsilon \geq 1$ je **aproximačním poměrem** algoritmu R , pokud pro každou instanci $I \in D_A$ platí, že $\text{opt}(I) \leq \varepsilon \cdot \mu_A(I, R(I))$.
- Je-li A minimalizační úloha, pak $\varepsilon \geq 1$ je **aproximačním poměrem** algoritmu R , pokud pro každou instanci $I \in D_A$ platí, že $\mu_A(I, R(I)) \leq \varepsilon \cdot \text{opt}(I)$.

Aproximační algoritmus pro Bin Packing

Algoritmus 1 First Fit (FF)

- 1: Ber předměty jeden po druhém a pro každý najdi první množinu, do níž se vejde.
 - 2: Pokud taková množina neexistuje, přidej novou množinu, obsahující jen tento předmět.
-

Věta 67

- Je-li I instance *BIN PACKING* a je-li m počet košů, které vytvoří algoritmus FF pro instanci I , pak $m < 2 \cdot \text{opt}(I)$.
- Pro každé m existuje instance I , pro niž je $\text{opt}(I) \geq m$ a FF vytvoří pro instanci I alespoň $\frac{5}{3}\text{opt}(I)$ košů.

Lepší algoritmus pro Bin Packing

Algoritmus 2 First Fit Decreasing (FFD)

- 1: Seříd' předměty vzestupně podle velikosti.
 - 2: Ber předměty od největšího po nejmenší a pro každý najdi první množinu, do níž se vejde.
 - 3: Pokud taková množina neexistuje, přidej novou množinu, obsahující jen tento předmět.
-

Věta 68 (Bez důkazu)

- Je-li I instance *BIN PACKING* a je-li m počet košů, které vytvoří algoritmus FFD pro instanci I , pak $m \leq \frac{11}{9} \cdot \text{opt}(I) + 4$.
- Pro každé m existuje instance I , pro niž je $\text{opt}(I) \geq m$ a FFD vytvoří pro instanci I alespoň $\frac{11}{9}\text{opt}(I)$ košů.

Množinové pokrytí

Množinové pokrytí (Set Cover, SC)

Instance: Množina prvků $U = \{u_1, \dots, u_n\}$, systém množin $\mathcal{S} = \{S_i \mid S_i \subseteq U, i = 1, \dots, m\}$.

Přípustné řešení: Množina indexů $I \subseteq \{1, \dots, m\}$ množin pokrývajících všechny prvky v U , tj. $\bigcup_{i \in I} S_i = U$.

Cíl: Minimalizovat počet množin v pokrytí $|I|$.

- Rozhodovací verze je NP-úplná.
- VRCHOLOVÉ POKRYTÍ je zvláštním případem množinového pokrytí.
- Ekvivalentní problému HITTING SET.

Hladový algoritmus pro Množinové pokrytí

Algoritmus 3 Hladový algoritmus pro Množinové pokrytí

```
1:  $X \leftarrow U$ 
2:  $I \leftarrow \emptyset$ 
3: while  $X \neq \emptyset$  do
4:   Vyber  $i \in \{1, \dots, m\}$ , pro něž  $|S_i \cap X| = \max_{j=1}^m |S_j \cap X|$ .
5:    $I \leftarrow I \cup \{i\}$ 
6:    $X \leftarrow X \setminus S_i$ 
7: end while
8: return  $I$ 
```

Theorem 69 (Bez důkazu)

Hladový algoritmus pro MNOŽINOVÉ POKRYTÍ je polynomiální aproximační algoritmus s aproximačním poměrem $\ln n + 1$.

Obchodní cestující (optimalizační verze)

Obchodní cestující (OC, Traveling salesperson)

Instance: Množina měst $C = \{c_1, \dots, c_n\}$, hodnoty $d(c_i, c_j) \in \mathbb{N}$ přiřazující každé dvojici měst vzdálenost.

Přípustné řešení: Permutace měst $c_{\pi(1)}, c_{\pi(2)}, \dots, c_{\pi(n)}$

Cíl: Minimalizovat

$$\left(\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) \right) + d(c_{\pi(n)}, c_{\pi(1)}).$$

Věta 70

Pokud $P \neq NP$, neexistuje polynomiální aproximační algoritmus s konstantním aproximačním poměrem pro úlohu OBCHODNÍHO CESTUJÍCÍHO.

- Existuje $\frac{3}{2}$ -aproximační algoritmus pro úlohu OC s trojúhelníkovou nerovností.
- Existuje polynomiální aproximační schéma pro OC v eukleidovské rovině.
- Pokud $P \neq NP$, neexistuje polynomiální aproximační algoritmus pro MNOŽINOVÉ POKRYTÍ s aproximačním poměrem $(1 - o(1)) \ln n$.

Aproximační schémata

Aproximační schéma pro Batoh

Vstup: Velikost batohu B , počet předmětů n . Pole velikostí s a pole cen v (obě délky n). Předpokládáme, že $(\forall i)[0 \leq s(i) \leq B]$. Racionální číslo $\varepsilon > 0$.

Výstup: Množina předmětů A' s celkovou velikostí nejvýš B a s celkovou cenou alespoň $\frac{1}{1+\varepsilon} \text{opt}(I)$.

```
1: function BAPX( $I = (B, n, s, v), \varepsilon$ )
2:    $m \leftarrow \arg \max_{1 \leq i \leq n} v[i]$ 
3:   if  $\varepsilon \geq n - 1$  then return  $\{m\}$  end if
4:    $t \leftarrow \left\lfloor \log_2 \left( \frac{\varepsilon \cdot v[m]}{n} \right) \right\rfloor - 1$ 
5:    $c$  je nové pole délky  $n$ 
6:   for  $i \leftarrow 1$  to  $n$  do
7:      $c[i] \leftarrow \left\lfloor \frac{v[i]}{2^t} \right\rfloor$ 
8:   end for
9:   Pseudopolynomiálním algoritmem pro Batoh najdi optimální
   řešení instance  $B, s, c$  a vrať nalezené řešení.
10: end function
```

Věta 71

Nechť I je instance problému *BATOCHU* a nechť $\varepsilon > 0$ je racionální číslo.

- Nechť $\text{bapx}(I, \varepsilon)$ je hodnota řešení vráceného algoritmem BAPX pro danou instanci I a danou hodnotu $\varepsilon > 0$, potom

$$\text{opt}(I) \leq (1 + \varepsilon) \cdot \text{bapx}(I, \varepsilon).$$

- Algoritmus BAPX pracuje v čase $O(\frac{1}{\varepsilon}n^3)$ (počítáme-li aritmetické operace jako konstantní).

Úplně polynomiální aproximační schéma

Definice 72

- Algoritmus **ALG** je **aproximačním schématem** pro optimalizační úlohu **A**, pokud na vstupu očekává instanci $I \in D_A$ a racionální číslo $\varepsilon > 0$ a na výstupu vydá řešení $\sigma \in S_A(I)$ s aproximačním poměrem $1 + \varepsilon$.
 - Pokud **ALG** pracuje v polynomiálním čase vzhledem k $\text{len}(I)$, pak jde o **polynomiální aproximační schéma**.
 - Pokud **ALG** pracuje v polynomiálním čase vzhledem k $\text{len}(I)$ a $\frac{1}{\varepsilon}$, jedná se o **úplně polynomiální aproximační schéma (ÚPAS)**.
-
- BAPX je úplně polynomiální aproximační schéma pro úlohu **BATOHU**.

Věta 73

Nechť A je optimalizační úloha, jejíž přípustná řešení mají nezápornou celočíselnou hodnotu a nechť existuje polynom q dvou proměnných takový, že pro každou instanci I úlohy A platí, že

$$\text{opt}(I) < q(\text{len}(I), \max(I)).$$

Pokud existuje úplně polynomiální aproximační schéma pro A , pak existuje i pseudopolynomiální algoritmus pro A .

- Pokud tedy $P \neq NP$, neexistuje ÚPAS pro žádnou silně NP-úplnou úlohu, která splňuje požadavky této věty.

Třídy co-NP a #P.

Nesplnitelnost (UNSAT)

Instance: Formule φ v KNF

Otázka: Platí, že pro každé ohodnocení proměnných v je $\varphi(v) = 0$ (nesplněno)?

- Neumíme popsat polynomiální verifikátor pro problém **UNSAT**, tento problém nejspíš nepatří do třídy **NP**.
- Jazyk **UNSAT** je (v podstatě) doplňkem jazyka **SAT**, neboť pro každou formuli φ v KNF platí

$$\varphi \in \text{UNSAT} \iff \varphi \notin \text{SAT}$$

Definice 74

Jazyk A patří do třídy co-NP, právě když jeho doplněk \overline{A} patří do třídy NP.

- Například UNSAT patří do co-NP (poznat řetězce, které nekódují formule, je snadné).
- Jazyk L patří do co-NP, právě když existuje polynomiální verifikátor V , pro který platí, že

$$L = \{x \mid (\forall y) [V(x, y) \text{ odmítne}]\}.$$

- Platí, že $P \subseteq NP \cap \text{co-NP}$.

Definice 75

Problém A je co-NP-úplný, pokud

- ❶ *A patří do třídy co-NP a*
- ❷ *každý problém $B \in \text{co-NP}$ je na A polynomiálně převoditelný.*

- Jazyk A je co-NP-úplný, právě když jeho doplněk \overline{A} je NP-úplný.
- Například UNSAT je co-NP-úplný problém.
- Pokud by existoval NP-úplný jazyk A , který by patřil do co-NP, platilo by $\text{NP} = \text{co-NP}$.

Definice 76

Funkce $f : \Sigma^* \mapsto \mathbb{N}$ patří do třídy #P, pokud existuje polynom p a polynomiální verifikátor V takové, že pro každé $x \in \Sigma^*$

$$f(x) = |\{y \mid |y| \leq p(|x|) \text{ a } V(x, y) \text{ přijme}\}|.$$

- S každým problémem $A \in \text{NP}$ můžeme asociovat funkci $\#A$ v #P (asociovanou s „přirozeným“ polynomiálním verifikátorem pro A).
- **Přirozený verifikátor** myslíme verifikátor, který ověřuje, zda y je řešením odpovídající úlohy.
- Například přirozený verifikátor pro SAT přijme dvojici φ, v , pokud φ je KNF a v je splňující ohodnocení φ .
- Potom $\#\text{SAT}(\varphi) = |\{v \mid \varphi(v) = 1\}|$.

Třída #P (vlastnosti)

Uvažme funkci $f \in \#P$ a problém:

Nenulová hodnota f

Instance: $x \in \Sigma^*$.

Otázka: $f(x) > 0$?

- Problém **NENULOVÁ HODNOTA** f patří do **NP**.
- Hodnotu $f \in \#P$ lze získat pomocí polynomiálně mnoha dotazů na náležení prvku do množiny $\{(x, N) \mid f(x) \geq N\}$.
- Hodnotu $f \in \#P$ lze spočítat v polynomiálním prostoru.

Převod funkce na funkci

Definice 77

Funkce $f : \Sigma^* \mapsto \mathbb{N}$ je **polynomiálně převoditelná** na funkci $g : \Sigma^* \mapsto \mathbb{N}$ ($f \leq_P g$) pokud existují funkce $\alpha : \Sigma^* \times \mathbb{N} \mapsto \mathbb{N}$ a $\beta : \Sigma^* \mapsto \Sigma^*$, jejichž hodnotu lze spočítat v polynomiálním čase a

$$(\forall x \in \Sigma^*) [f(x) = \alpha(x, g(\beta(x)))]$$

- To odpovídá tomu, že hodnotu f můžeme spočítat v polynomiálním čase s jedním voláním funkce g (pokud bereme toto volání jako konstatní operaci).

Převod se zachováním počtu řešení

Definice 78

Řekneme, že problém $A \in \Sigma^*$ je převoditelný na problém $B \in \Sigma^*$ v *polynomiálním čase se zachováním počtu řešení* ($A \leq_c^P B$), pokud existuje funkce $f : \Sigma^* \mapsto \Sigma^*$ vyčíslitelná v polynomiálním čase, pro kterou platí, že

$$|\{y \mid V_A(x, y) \text{ přijme}\}| = |\{y \mid V_B(f(x), y) \text{ přijme}\}|,$$

kde V_A a V_B jsou přirozené verifikátory pro A a B .

- Pokud $A \leq_c^P B$, pak $\#A \leq_P \#B$.
- Převody, které jsme si ukazovali, lze provést tak, aby zachovávaly počty řešení.

Definice 79

Řekneme, že funkce $f : \Sigma^* \mapsto \mathbb{N}$ je #P-úplná, pokud

- 1 $f \in \#P$ a
- 2 každá funkce $g \in \#P$ je polynomiálně převoditelná na f .

- Například #SAT, #VRCHOLOVÉ POKRYTÍ a další početní verze NP-úplných problémů, jsou #P-úplné.
- A to pomocí převoditelnosti se zachováním počtu řešení.
- Existují problémy z P, jejichž početní verze jsou #P-úplné.

Term je konjunkcí literálů.

Disjunktivní normální forma (DNF) je disjunkcí termů.

DNF-Splnitelnost (DNF-SAT)

Instance: Formule φ v DNF

Otázka: Existuje ohodnocení proměnných v , pro které je $\varphi(v)$ splněno?

- **DNF-SAT** je polynomiálně řešitelný.
- Funkce **#DNF-SAT** je **#P**-úplná.

Počet perfektních párování v bipartitním grafu

Perfektní párování v bipartitním grafu (BPM)

Instance: Bipartitní graf $G = (V = A \cup B, E \subseteq A \times B)$, kde $|A| = |B|$.

Otázka: Existuje v G párování velikosti $|A| = |B|$?

Věta 80 (Bez důkazu)

Funkce #BPM je #P-úplná.

Permanent matice

Definice 81

Je-li A matice typu $n \times n$ definujeme permanent A jako

$$\text{perm}(A) = \sum_{\pi \in S(n)} \prod_{i=1}^n a_{i,\pi(i)},$$

kde $S(n)$ je množina permutací množiny $\{1, \dots, n\}$.

- „Determinant“, kde neuvažujeme znaménko permutace.
- Je-li A matice sousednosti bipartitního grafu G , pak $\text{perm}(A)$ určuje počet perfektních párování G .

Věta 82 (Bez důkazu)

Funkce perm je #P-úplná.

Pro ty, kdo chtějí vědět víc, doporučuji navazující přednášky v letním semestru:

Vyčíslitelnost (NTIN064)

Přednáší doc. RNDr. Antonín Kučera, CSc.

Složitost (NTIN063)

Přednáší doc. RNDr. Ondřej Čepek, Ph.D.

1 Základy vyčíslitelnosti

- a Algoritmicky vyčíslitelné funkce, numerace, s-m-n věta
- b Základní vlastnosti rekurzivních a rekurzivně spočetných množin — shrnutí
- c Věty o rekurzi a jejich aplikace
- d Produktivní a kreativní množiny a jejich vlastnosti
- e Efektivně neoddělitelné dvojice množin, Gödelovy věty o neúplnosti
- f Relativní vyčíslitelnost

2 Relativní vyčíslitelnost, částečně rekurzivní funkcionály, Turingovská převeditelnost

- a Stupně nerozhodnutelnosti, operace skoku, relativizovaný halting problém
- b Limitní vyčíslitelnost
- c Aritmetická hierarchie, věta o hierarchii
- d Aplikace teorie vyčíslitelnosti

Složitost (NTIN063) — syllabus

- 1 Turingovy stroje s orákulem.
- 2 Polynomiální hierarchie (definice pomocí orákulí a pomocí alternujících kvantifikátorů, důkaz ekvivalence).
- 3 Kvantifikované booleovské formule QBF a jejich úplnost pro $PSPACE$ a Σ_i .
- 4 Nedeterministická hierarchie.
- 5 Log-space převoditelnost, P -úplnost a její důsledky.
- 6 Věta Szelepcsényi-Immermana a $NL = co-NL$.
- 7 Neuniformní výpočetní modely — radící funkce, booleovské obvody, třídy NC a $P/poly$, funkce s maximální velikostí obvodu.
- 8 Pravděpodobnostní algoritmy — třídy RP , $co-RP$, ZPP a BPP .
- 9 Redukce chyby pro BPP , BPP je v $P/poly$, BPP je v Σ_2 .
- 10 NP -úplnost $UNIQUE-SAT$ (pravděpodobnostní redukce)
- 11 PCP věta (bez důkazu) a její využití pro neaproximovatelnost.