Základy složitosti a vyčíslitelnosti

NTIN090

Petr Kučera

2017/18

Úvod

Sylabus

1. Turingovy stroje a jejich varianty. Churchova-Turingova teze
2. Halting problém.
3. RAM a jeho ekvivalence s Turingovými stroji. Algoritmicky vyčíslitelné funkce.
4. Rekurzivní a rekurzivně spočetné jazyky a množiny a jejich vlastnosti.
5. 1-převoditelnost a m-převoditelnost, 1-úplné a m-úplné množiny.
6. Riceova věta.
7. Nedeterministické Turingovy stroje, základní třídy složitosti, třídy P, NP, PSPACE, $\mathrm{EXPTIME}$.
8. Savičova věta — ekvivalence tříd PSPACE a $\mathrm{NPSPACE}$.
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 $\mathrm{co}\text{-}\mathrm{NP}$ a $\#\mathrm{P}$.

Literatura

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/~kucerap/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

(I) Co je to algoritmus?
(II) Co všechno lze pomocí algoritmů spočítat?
(III) Dokáží algoritmy vyřešit všechny úlohy a problémy?
(IV) Jak poznat, že pro řešení zadané úlohy nelze sestrojit žádným algoritmus?
(V) Jaké algoritmy jsou „rychlé“ a jaké problémy jimi můžeme řešit?
(VI) Jaký je rozdíl mezi časem a prostorem?
(VII) Které problémy jsou lehké a které těžké? A jak je poznat?
(VIII) Je lépe zkoušet, nebo být zkoušený?
(IX) Jak řešit problémy, pro které neznáme žádný „rychlý“ algoritmus?

Vyčíslitelnost

Lehký úvod do te-orie 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 funkčností 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

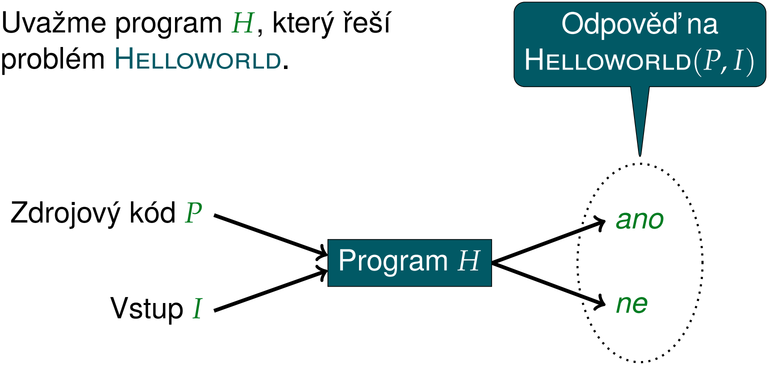
HELLOWORLD	
Instance	Zdrojový kód programu P v jazyce C a jeho vstup I.
Otázka	Je pravda, že prvních 12 znaků, které daný program vypíše, je *Hello, world*? (Nevyžadujeme zastavení.)

Lze napsat program v jazyce C, který o vstupu 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?

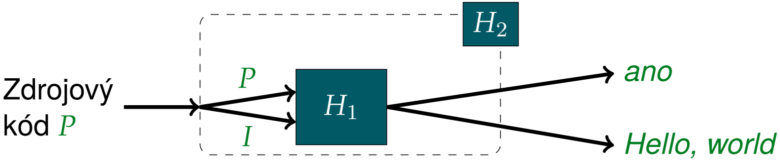
Co je program H_1 schopen říci sám o sobě?

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.

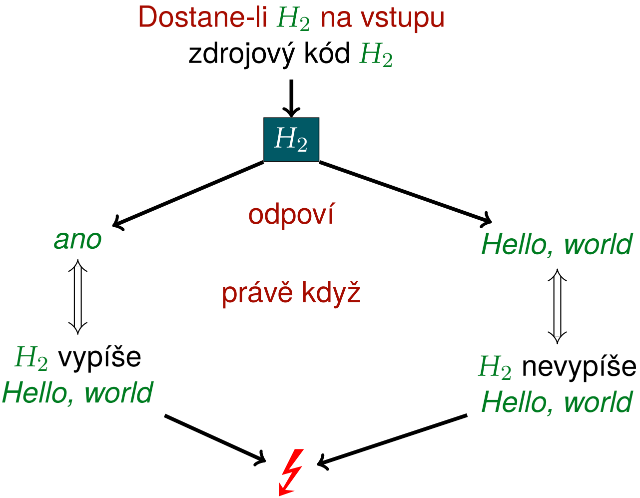
Dva vstupy v jednom

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?

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

Volání funkce `foo`

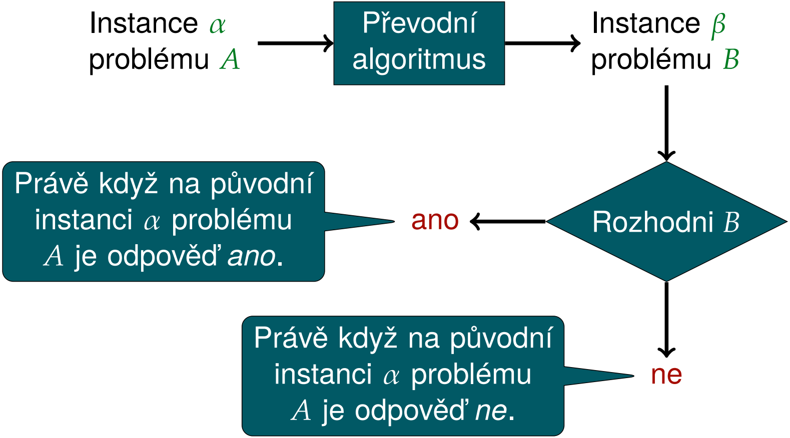
Uvažme další problém:

Volání funkce `foo`	
Instance	Zdrojový kód programu Q v jazyce C a jeho 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

- Popíšeme, jak převést problém HELLOWORLD na problém VOLÁNÍ FUNKCE **foo**.
- Musíme tedy popsat, jak převést instanci problému HELLOWORLD (tedy dvojici programu P a vstupu I) na instanci problému VOLÁNÍ FUNKCE **foo** (tedy dvojici programu Q a vstupu 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`.

- Pokud se nám to podaří, bude to znamenat, že i problém VOLÁNÍ FUNKCE **foo** je 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 pokud použije příkaz pro výstup, zkontroluje pole A, je-li v něm alespoň dvanáct znaků a na začátku obsahuje *Hello, world*. Pokud ano, zavolá funkci **`foo`** (tím dostaneme 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:

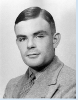
Efektivně vyčíslitelné funkce jsou právě ty, které jsou definované v λ-kalkulu.

Tuto tezi později (1936) upravil na

Efektivně vyčíslitelné funkce jsou právě částečně rekurzivní funkce.

Turingova teze

V roce 1936 publikoval Alan Turing následující tezi

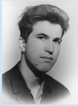


Ke každému algoritmu v intuitivním smyslu existuje ekvivalentní Turingův stroj.

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

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

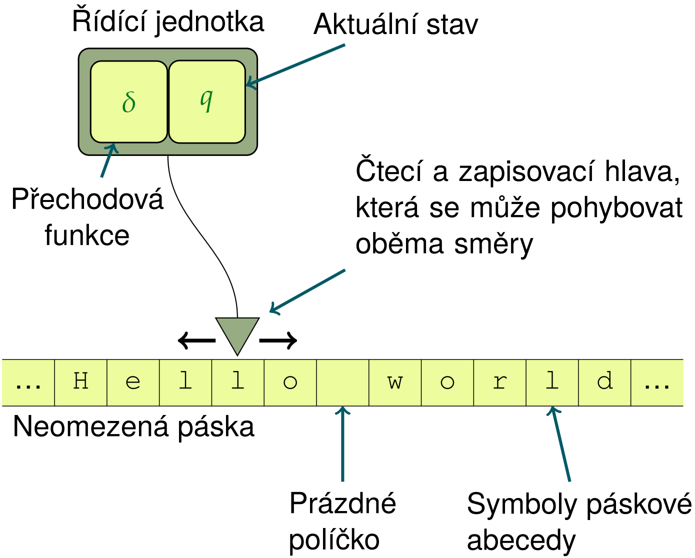
Podle Churchovy-Turingovy teze je algoritmus ekvivalentní ...

- 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

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
  - řídící 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 řídící 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.
- Doplněk jazyka L označíme pomocí $\overline{L} = \Sigma^* \setminus L$.
- 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 přijímající.
- 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!

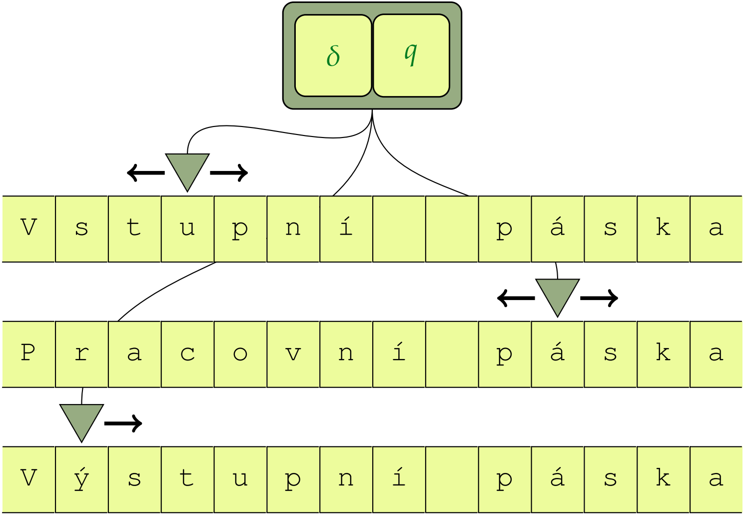
Varianty Turingových strojů

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

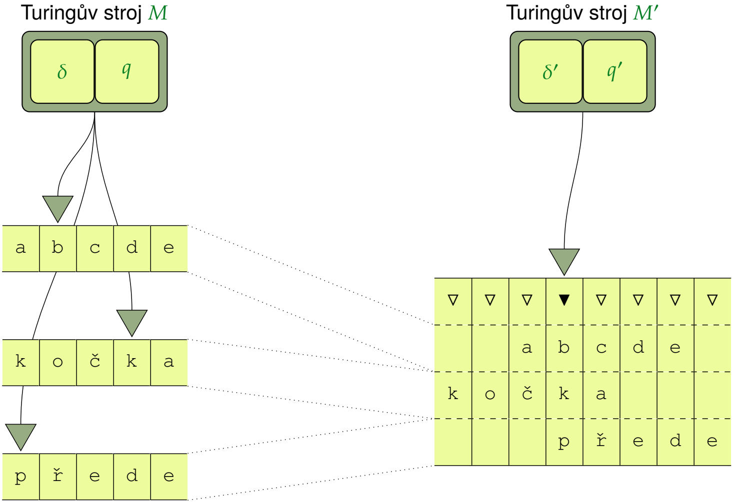
k-páskový Turingův stroj se od jednopáskového Turingova stroje líší následujícím způsobem:

- Má k pásek, na každé je zvláštní hlava.
  - Vstupní páska na počátku obsahuje vstupní řetězec. Často je určena jen pro čtení.
  - Pracovní pásky jsou určeny pro čtení i zápis.
  - Výstupní páska na konci obsahuje výstupní řetězec. Často je určena 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

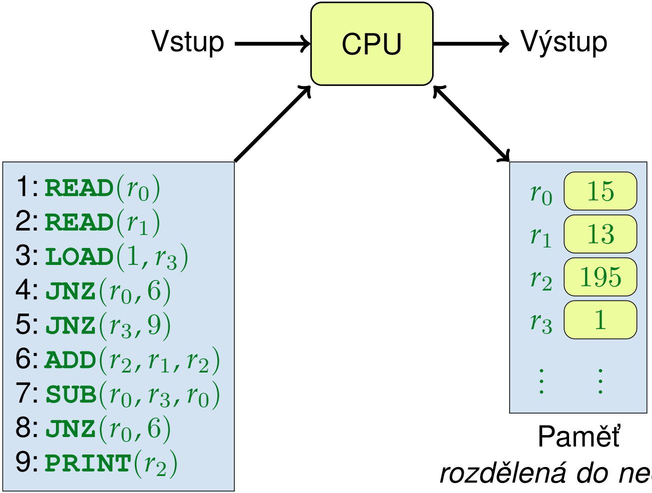
Ke každému k-páskovému Turingovu stroji M existuje jednopáskový Turingův stroj M', který simuluje práci M, přijímá týž jazyk jako M a počítá touž funkci jako M.

Reprezentace k pásek na jedné pásce



Random Access Machine

Random Access Machine (RAM)



Program

Paměť
rozdělená do neomezeného počtu registrů

Random Access Machine (definice)

- Random Access Machine (RAM, stroj s náhodným přístupem do paměti) se skládá z
    - řídící 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 $P = I_0, I_1, \ldots, I_\ell$.
- Instrukce jsou vykonávané v pořadí daném programem.

Možné instrukce RAM

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

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

Jazyky rozhodnutelné RAMem

- Uvažme abecedu $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$.
- Slovo $w = \sigma_{i_1}\sigma_{i_2}\ldots\sigma_{i_n}$ předáme RAMu R jako posloupnost čísel $i_1, \ldots, i_n$.
- 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 $(x_1, \ldots, x_n)$, platí následující:

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

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 $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$, pokud platí:

- Vstupní řetězec $w = \sigma_{i_1}\sigma_{i_2} \ldots \sigma_{i_n}$ je předaný jako posloupnost čísel $i_1, \ldots, i_n$.
- Konec slova pozná R díky tomu, že **READ** načte 0, není-li už k dispozici vstup.
- Pokud je $f(w)\!\downarrow = \sigma_{j_1}\sigma_{j_2} \ldots \sigma_{j_m}$, pak $R(w)\!\downarrow$ a na výstup je zapsaná posloupnost čísel $j_1, j_2, \ldots, j_m, 0$.
- 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 $A_1, \ldots, A_p$ a skalární proměnné $x_0, \ldots, x_s$.

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

Turingův stroj $\longrightarrow$ RAM

Věta 2

Ke každému Turingovu stroji M existuje ekvivalentní RAM R.

- Obsah pásky 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 následujícím způsobem:

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

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

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. Nejprve si ukážeme, jak zapsat Turingův stroj pomocí řetězce 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 na Turingovy stroje, které

(i) mají jediný přijímající stav a

(ii) mají pouze binární vstupní abecedu $\Sigma_{in} = \{0, 1\}$.

- Omezení vstupní abecedy znamená, že řetězce, které budeme předávat uvažovaným Turingovým strojům na vstup 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

- Uvažme TS $M = (Q, \Sigma, \delta, q_0, F)$ s jediným přijímajícím stavem a binární vstupní abecedou.
- K zakódování M stačí zakódovat přechodovou funkci.
- Prvním krokem bude zápis přechodové funkce pomocí řetězce v abecedě

$$\Gamma = \{0, 1, L, N, R, |, \#, ;\} \ .$$

- Každý znak abecedy Γ pak zapíšeme pomocí tří znaků z binární abecedy $\{0, 1\}$.
- Tím vznikne binární kód TS M.

Zápis v abecedě Γ

- Předpokládejme, že
  - $Q = \{q_0, q_1, \ldots, q_r\}$ pro nějaké $r \geq 1$, kde q_0 je počáteční stav a q_1 je jediný přijímající stav.
  - $\Sigma = \{X_0, X_1, X_2, \ldots, X_s\}$ 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

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

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

$$C_1\#C_2\#\ldots\#C_n \quad .$$

Převod do binární abecedy

λ

$\delta(q_3, X_7) = (q_5, X_2, R)$

$11|111|101|10|R$

Γ	kód
0	000
1	001
L	010
N	011
R	100
|	101
#	110
;	111

Znaky abecedy Γ zakódujeme pomocí této tabulky

001001101001001001101001000001101001000101100

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.

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

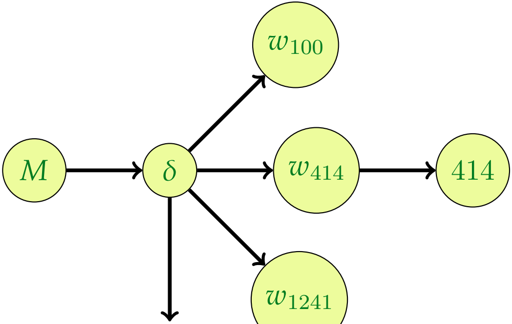
- K tomu přidáme konvenci, že 0 odpovídá prázdnému řetězci (tj. $w_0 = w_1 = \varepsilon$).

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.
- Jazyk přijímaný Turingovým strojem M_e označíme $L_e = L(M_e)$.
- Pokud řetězec w_e není syntakticky správným kódem Turingova stroje, pak M_e je prázdným Turingovým strojem, který každý vstup okamžitě odmítne a $L_e = \emptyset$.
- 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)

- Každý objekt (např. číslo, řetězec, Turingův stroj, RAM, graf nebo formuli) můžeme zakódovat do binárního řetězce.
- Podobně můžeme zakódovat i n-tice objektů.

Definice 4

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

- 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 $|$.

$\ldots \, | \, 010 \, | \, 001 \, | \, 100 \, | \, 000 \, | \, 010 \, | \, 011 \, | \, \ldots$

3. **páska** obsahuje číslo aktuálního stavu q_i stroje M.

$10011 \, (= (i)_B)$

Algoritmicky (ne)rozhodnutelné jazyky

Definice

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. $M(x)\downarrow$).*
- *Pomocí L_e označíme částečně rozhodnutelný jazyk přijímaný Turingovým strojem M_e.*

- 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 $\overline{L}$ jsou částečně rozhodnutelné jazyky.

(I) Jsou všechny jazyky nad konečnou abecedou částečně rozhodnutelné?

(II) 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.

- Turingovy stroje lze očíslovat, protože můžeme každému Turingovu stroji přiřadit Gödelovo číslo.
- Částečně rozhodnutelné jazyky lze očíslovat, protože každému můžeme přiřadit nějaký Turingův stroj.

Částečně rozhodnutelných jazyků je spočetně mnoho.

Jsou všechny jazyky rozhodnutelné?

- Uvážíme-li třeba $\Sigma = \{0,1\}$, pak jazyk $L \subseteq \Sigma^*$ odpovídá množině přirozených čísel $A = \{i-1 \mid i \in \mathbb{N} \setminus \{0\} \wedge w_i \in L\}$.
- Z toho plyne, že jazyků nad abecedou $\Sigma = \{0,1\}$ není spočetně mnoho.

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

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

Diagonalizační jazyk

Jako příklad jazyka, který není částečně rozhodnutelný nám může posloužit diagonalizační jazyk:

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

Věta 9

Jazyk DIAG není částečně rozhodnutelný (jinými slovy není rekurzivně spočetný).

- $\overline{\mathrm{DIAG}}$ je částečně rozhodnutelný, protože máme k dispozici univerzální Turingův stroj $\mathcal{U}$ a $\langle M\rangle \in \overline{\mathrm{DIAG}}$, právě když $\langle M, \langle M\rangle\rangle \in L(\mathcal{U})$.
- $\overline{\mathrm{DIAG}}$ není ovšem rozhodnutelný na základě Postovy věty.

Univerzální jazyk

Rozhodnutí, zde 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 10

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 11

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

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

- Oborem hodnot funkce f je množina

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

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

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

Algoritmicky vyčíslitelné funkce

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

Definice 12

- *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 $\Longrightarrow$ ne všechny funkce jsou vyčíslitelné.

Univerzální funkce

Věta 13

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

Částečně rozhodnutelné jazyky

Věta 14

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

(i) *L je částečně rozhodnutelný.*

(ii) *Existuje Turingův stroj M splňující*

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

(iii) *Existuje algoritmicky vyčíslitelná funkce f splňující*

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

(iv) *Existuje rozhodnutelný jazyk B splňující*

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

Rozhodnutelné jazyky

Věta 15

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.

Uspořádání řetězců

Definice 16 (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

(i) *je u kratší (tj. $|u| < |v|$), nebo*

(ii) *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 17

(i) *Jazyk L je částečně rozhodnutelný, právě když pro něj existuje enumerátor E.*

(ii) *Jazyk L je rozhodnutelný, právě když pro něj existuje enumerátor E, který navíc vypisuje prvky L v lexikografickém pořadí.*

Enumerace jazyků a funkce

Věta 18

Nechť L je nekonečný jazyk, potom jazyk L je …

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

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

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

Převoditelnost a úplnost

Definice 19

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.

Úplné jazyky

Problém zastavení a jeho diagonálu můžeme formalizovat jako

$$\begin{aligned}\mathrm{HALT} &= \{\langle M, x\rangle \mid M(x)\!\downarrow\}\\ K &= \{\langle M\rangle \mid M(\langle M\rangle)\!\downarrow\}\end{aligned}$$

Věta 20

Jazyky L_u, K a HALT jsou m-úplné. Zvláště pak jde o jazyky částečně rozhodnutelné, které nejsou rozhodnutelné.

Postův korespondenční problém

Postův korespondenční problém	
Instance	Množina „dominových kostek“ P: $$P = \left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \ldots, \left[\frac{t_k}{b_k}\right] \right\}$$ kde $t_1, \ldots, t_k, b_1, \ldots, b_k \in \Sigma^*$ jsou řetězce.
Otázka	Existuje párovací posloupnost $i_1, i_2, \ldots, i_l$, kde $l \geq 1$ a $t_{i_1} t_{i_2} \ldots t_{i_l} = b_{i_1} b_{i_2} \ldots b_{i_l}$?

Theorem 21

Postův korespondenční problém je nerozhodnutelný.

Riceova věta

Věta 22 (Riceova věta (jazyky))

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

Věta 23 (Riceova věta (funkce))

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

Riceova věta (důsledky)

Z Riceovy věty plyne, že následující jazyky nejsou rozhodnutelné:

$$
\begin{aligned}
\mathrm{NONEMPTY} &= \{\langle M\rangle \mid L(M) \neq \emptyset\} \\
\mathrm{Fin} &= \{\langle M\rangle \mid L(M) \text{ je konečný jazyk}\} \\
\mathrm{Cof} &= \{\langle M\rangle \mid \overline{L(M)} \text{ je konečný jazyk}\} \\
\mathrm{Inf} &= \{\langle M\rangle \mid L(M) \text{ je nekonečný jazyk}\} \\
\mathrm{Dec} &= \{\langle M\rangle \mid L(M) \text{ je rozhodnutelný jazyk}\} \\
\mathrm{Tot} &= \{\langle M\rangle \mid L(M) = \Sigma^*\} \\
\mathrm{Reg} &= \{\langle M\rangle \mid L(M) \text{ je regulární jazyk}\}
\end{aligned}
$$

S-m-n věta

Věta 24 (s-m-n)

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

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

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:
  - Otázka, zda je daný graf souvislý.
  - Otázka, zda lze danou logickou formuli splnit nějakým ohodnocením proměnných.
  - Otázka, zda daný lineární program má přípustné řešení.
  - Otázka, zda je dané číslo prvočíslem či číslem složeným.

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.

Časová a prostorová složitost Turingova stroje

Definice 25

Nechť M je (deterministický) Turingův stroj, který se zastaví na každém vstupu a nechť $f : \mathbb{N} \mapsto \mathbb{N}$ je funkce.

- *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 $|x| = n$ využije nejvýš $f(n)$ buněk pracovní pásky.*

Základní deterministické třídy složitosti

Definice 26

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

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

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

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

Význačné deterministické třídy složitosti

Definice 27

Třída problémů řešitelných v polynomiálním čase:

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

Třída problémů řešitelných v polynomiálním prostoru:

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

Třída problémů řešitelných v exponenciálním čase:

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

Proč polynomy?

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 (*pokud lze tento simulovat na Turingovu stroji s polynomiálním zpomalením*).
- P zhruba odpovídá třídě problémů, které lze řešit na počítači v rozumném čase.

Verifikátor čili ověřovatel

Definice 28

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 zveme 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.*

Třída NP

Definice 29

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 nedetermistických 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 týž 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 $C_0, C_1, C_2, \ldots$, 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 $L(M)$.

Časová a prostorová složitost NTS

Definice 30

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* $|x| = n$ *využije nejvýše* $f(n)$ *buněk pracovní pásky.*

Základní nedeterministické třídy složitosti

Definice 31

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

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

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

Věta 32

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

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

NP=nedeterministicky polynomiální

Věta 33 (Alternativní definice třídy NP)

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

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

Model TS s menším než lineárním prostorem

Pro prostor menší než lineární uvažujeme vícepáskový TS:

Vstupní páska je jen pro čtení

Pracovní pásky 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.

Další prostorové třídy

Definice 34

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

Vztahy mezi třídami

Věta 35

Nechť $f(n)$ *je funkce, pro kterou platí* $f(n) \geq \log_2 n$. *Pro každý jazyk* $L \in \mathrm{NSPACE}(f(n))$ *platí, že* $L \in \mathrm{TIME}(2^{c_L f(n)})$, *kde* c_L *je konstanta závislá na jazyku* L.

Věta 36

Platí následující inkluze

$$\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{NPSPACE} \subseteq \mathrm{EXPTIME}.$$

Savičova věta

Savičova věta

Věta 37 (Savičova věta)

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

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

Důsledek 38

$$\mathrm{PSPACE} = \mathrm{NPSPACE}$$

Věty o hierarchii

Věta o deterministické prostorové hierarchii

Definice 39

Funkci $f : \mathbb{N} \mapsto \mathbb{N}$*, 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 40 (Věta o deterministické prostorové hierarchii)

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

Deterministická prostorová hierarchie

Důsledek 41

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

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

(ii) *Pro každá dvě reálná čísla* $0 \leq \epsilon_1 < \epsilon_2$ *platí, že*

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

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

Věta o deterministické časové hierarchii

Definice 42

Funkci $f : \mathbb{N} \mapsto \mathbb{N}$, *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 43 (Věta o deterministické časové hierarchii)

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

Důsledek 44

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

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

(ii) *Pro každá dvě reálná čísla* $0 \leq \epsilon_1 < \epsilon_2$*,*

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

(iii) $P \subsetneq EXPTIME$*.*

Polynomiální převoditelnost a NP-úplnost

Polynomiální převoditelnost

Definice 45

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

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

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

NP-úplnost

Definice 46

- *Jazyk B je NP-**těžký**, pokud je na něj převoditelný kterýkoli problém $A \in \mathrm{NP}$.*
- *NP-těžký jazyk B, který navíc patří do NP zveme NP-**úplným**.*

- Pokud chceme ukázat, že nějaký problém B je NP-úplný, pak stačí
  1. ukázat $B \in \mathrm{NP}$ a
  2. najít jiný NP-úplný problém A a převést jej na B (tj. $A \leq_m^P B$).

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

NP-úplný problém

KACHLÍKOVÁNÍ (TILING)	
Instance	Množina barev B, přirozené číslo s, čtvercová mřížka 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 47

KACHLÍKOVÁNÍ je NP-úplný problém.

Splnitelnost

Literál – proměnná (např. x) nebo její negace (např. $\overline{x}$).

Klauzule – disjunkce literálů.

Konjunktivně 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 48 (Cookova-Levinova věta)

Pokud by byla Splnitelnost řešitelná v polynomiálním čase, pak by se $\mathrm{P} = \mathrm{NP}$. 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 49

3-Splnitelnost je NP-úplný problém.

- 2-Splnitelnost jej již polynomiálně řešitelná.

Vrcholové pokrytí

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 50

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 51 (Bez důkazu)

HAMILTONOVSKÁ KRUŽNICE je NP*-úplný problém.*

Obchodní cestující

Obchodní cestující (OC, Traveling salespersion)	
Instance	Množina měst $C = \{c_1, \ldots, c_n\}$, hodnoty $d(c_i, c_j) \in \mathbb{N}$ přiřazující každé dvojici měst vzdálenost a přirozené číslo D.
Otázka	Existuje permutace měst $c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(n)}$, pro kterou platí, že $$\left(\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)})\right) + d(c_{\pi(n)}, c_{\pi(1)}) \leq D \text{ ?}$$

Věta 52

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 53

Trojrozměrné párování je NP*-úplný problém.*

Loupežníci

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 54

Loupežníci je NP*-úplný problém.*

Batoh

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

Věta 55

BATOH je NP-úplný problém.

- NP-těžkost lze ukázat snadným převodem z LOUPEŽNÍKŮ.

Rozvrhování

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 56

Rozvrhování je NP-úplný problém.

- NP-těžkost lze ukázat snadným převodem z Loupežníků.

Aproximační algoritmy

Optimalizační úloha

Definice 57

- *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* $\mathrm{opt}(I)$*.*

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 $U_1, \ldots, U_m$, pro které platí, že $$(\forall i \in \{1, \ldots, m\}) \left[\sum_{u \in U_i} s(u) \leq 1 \right].$$
Cíl	Minimalizovat počet košů m.

Rozhodovací verze Bin Packing je shodná s Rozvrhováním.

Aproximační algoritmus

Definice 58

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

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 59

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

Lepší algoritmus pro Bin Packing

Algoritmus 2 First Fit Decreasing (FFD)

1: Setřiď 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 60 (Bez důkazu)

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

Obchodní cestující (optimalizační verze)

Obchodní cestující (OC, Traveling salespersion)

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

Těžkost aproximace

Věta 61

Pokud $\mathrm{P} \neq \mathrm{NP}$, 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.

Pseudopolynomiální algoritmy a silná $\mathbb{NP}$-úplnost

Batoh (optimalizační verze)

Batoh (Knapsack)	
Instance	Množina předmětů A, s každým předmětem $a \in A$ asociovaná velikost $s(a) \in \mathbb{N}$ a velikost $v(a) \in \mathbb{N}$, 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 velikosti 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 $\{1,\ldots,j\}$ 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)

```
T[0,0] ← ∅, S[0,0] ← 0
for c ← 1 to V do
    T[0,c] ← ∅, S[0,c] ← B + 1
end for
for j ← 1 to n do
    T[j,0] ← ∅, S[j,0] ← 0
    for c ← 1 to V do
        T[j,c] ← T[j−1,c], S[j,c] ← S[j−1,c]
        if v[j] ≤ c and S[j,c] > S[j−1,c−v[j]] + s[j] then
            T[j,c] ← T[j−1,c−v[j]] ∪ {j}
            S[j,c] ← S[j−1,c−v[j]] + s[j]
        end if
    end for
end for
c ← max{c′ | S[n,c′] ≤ B}
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.

Číselné problémy

Definice 62

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

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

$\max(I)$ *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* $\max(I) > p(\mathrm{len}(I))$*.*

- Například Batoh nebo Loupežníci jsou číselné problémy.
- Problémy Splnitelnost nebo Kachlíkování číselné nejsou.

Pseudopolynomiální algoritmus

Definice 63

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

- Obvykle měříme časovou složitost jen vzhledem k $\operatorname{len}(I)$.
- Pokud by existoval polynom p, pro který by platilo, že $\max(I) \le p(\operatorname{len}(I))$ (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.

Silná NP-úplnost

Definice 64

- *Nechť A je rozhodovací problém a p je polynom. Pomocí $A(p)$ označíme restrikci problému A na instance I, pro něž platí $\max(I) \leq p(\mathrm{len}(I))$.*
- *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 $\mathrm{P} = \mathrm{NP}$.

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 $C = \{c_1, \ldots, c_n\}$, hodnoty $d(c_i, c_j) \in \mathbb{N}$ přiřazující každé dvojici měst vzdálenost a přirozené číslo D.
Otázka	Existuje permutace měst $c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(n)}$, pro kterou platí, že $$\left(\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)})\right) + d(c_{\pi(n)}, c_{\pi(1)}) \le D$$?

Věta 65

Problém Obchodního cestujícího je silně NP-úplný.

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 velikosti nejvýš B a s celkovou cenou alespoň $\frac{1}{1+\varepsilon}\mathrm{opt}(I)$.

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

Aproximační schéma pro Batoh – vlastnosti

Věta 66

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

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

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

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

Úplně polynomiální aproximační schéma

Definice 67

- *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* $\mathrm{len}(I)$*, pak jde o polynomiální aproximační schéma.*
- *Pokud* ALG *pracuje v polynomiálním čase vzhledem k* $\mathrm{len}(I)$ *a* $\frac{1}{\varepsilon}$*, jedná se o úplně polynomiální aproximační schéma (ÚPAS).*

- BAPX je úplně polynomiální aproximační schéma pro úlohu Batohu.

Aproximační schémata a silná NP-úplnost

Věta 68

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

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

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

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

Třídy co-NP a #P.

Nesplnitelnost

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

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

Třída co-NP

Definice 69

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

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

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

co-NP-úplnost

Definice 70

Problém A je co-NP-úplný, pokud

(i) *A patří do třídy co-NP a*

(ii) *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 $\text{NP} = \text{co-NP}$.

Třída #P

Definice 71

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

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

- S každým problémem $A \in \mathrm{NP}$ můžeme asociovat funkci $\#A$ v $\#\mathrm{P}$ (asociovanou s „přirozeným“ polynomiálním verifikátorem pro A).
- Přirozeným verifikátorem 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 $\#\mathrm{SAT}(\varphi) = |\{v \mid \varphi(v) = 1\}|$.

Třída #P (vlastnosti)

Uvažme funkci $f \in \#\mathrm{P}$ a problém:

Nenulová hodnota f	
Instance	$x \in \Sigma^*$.
Otázka	$f(x) > 0$?

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

Převod funkce na funkci

Definice 72

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*

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

- 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 73

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.

#P-úplnost

Definice 74

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

(i) $f \in \#\mathrm{P}$ *a*

(ii) *každá funkce* $g \in \#\mathrm{P}$ *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.

#DNF-SAT

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 $\vert A\vert = \vert B\vert$.
Otázka	Existuje v G párování velikosti $\vert A\vert = \vert B\vert$?

Věta 75 (Bez důkazu)

Funkce #BPM je #$\mathbb{P}$-úplná.

Permanent matice

Definice 76

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

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

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

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

Věta 77 (Bez důkazu)

Funkce perm je $\#\mathrm{P}$-úplná.

Reklama

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.

Vyčíslitelnost (NTIN064) — sylabus

(I) 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

(II) 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) — sylabus

1. Turingovy stroje s orákulem.
2. Polynomiální hierarchie (definice pomocí orákulí a pomocí alternujicích 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 Szelepcsenyi-Immermana a $\mathrm{NL} = \text{co-NL}$.
7. Neuniformní výpočetní modely — radící funkce, booleovské obvody, třídy NC a $\mathrm{P/poly}$, funkce s maximální velikostí obvodu.
8. Pravděpodobnostní algoritmy — třídy RP, co-RP, ZPP a BPP.
9. Redukce chyby pro BPP, BPP je v $\mathrm{P/poly}$, BPP je v Σ_2.
10. NP-úplnost UNIQUE-SAT (pravděpodobnostní redukce)
11. PCP věta (bez důkazu) a její využití pro