

Složitost

Složitost algoritmů

❖ Základní teoretický přístup vychází z Church-Turingovy teze:

Každý algoritmus je implementovatelný jistým TS.

❖ Zavedení TS nám umožňuje klasifikovat problémy (resp. funkce) do dvou tříd:

1. problémy, jež nejsou **algoritmicky ani částečně rozhodnutelné** (resp. funkce algoritmicky nevyčíslitelné) a
2. problémy **algoritmicky alespoň částečně rozhodnutelné** (resp. funkce algoritmicky vyčíslitelné).

❖ Nyní se budeme zabývat třídou algoritmicky (částečně) rozhodnutelných problémů (vyčíslitelných funkcí) v souvislosti s otázkou **složitosti** jejich rozhodování (vyčíslování).

❖ Analýzu složitosti algoritmu budeme chápat jako analýzu složitosti výpočtů příslušného TS, jejímž cílem je **vyjádřit (kvantifikovat) požadované zdroje (čas, prostor) jako funkci závisující na délce vstupního řetězce**.

Různé případy při analýze složitosti

❖ Můžeme rozlišit:

1. analýzu složitosti nejhoršího případu,
2. analýzu složitosti nejlepšího případu,
3. analýzu složitosti průměrného případu,
4. amortizovanou analýzu

❖ Průměrná složitost algoritmu je definována následovně:

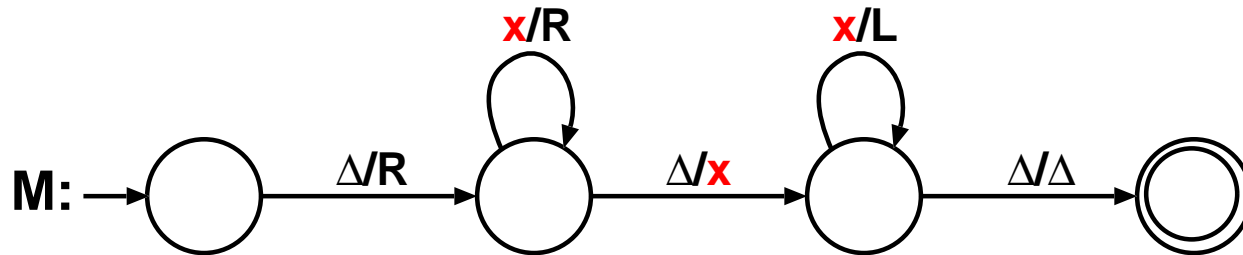
- Jestliže algoritmus (TS) vede k m různým výpočtům (případům) se složitostí c_1, c_2, \dots, c_m , jež nastávají s pravděpodobnostmi p_1, p_2, \dots, p_m , pak **průměrná složitost algoritmu** je dána jako $\sum_{i=1}^m p_i c_i$.

❖ **Amortizovaná analýza** studuje posloupnost operací jako celek. Tato technika umožňuje, na rozdíl od klasického přístupu mnohem přesnější určení časové složitosti algoritmu.

Složitost výpočtů TS pro daný vstup

- ❖ Časová složitost – počet kroků (přechodů) TS provedený od počátku do konce výpočtu.
- ❖ Prostorová (paměťová) složitost – počet „buněk“ pásky TS požadovaný pro daný výpočet.

Příklad 5.1 Uvažme následující TS M :



Pro vstup $\Delta xxx \Delta \Delta \dots$ je:

- časová složitost výpočtu M rovna 10,
- prostorová složitost výpočtu M rovna 5.

Lemma 5.1 Je-li časová složitost výpočtu prováděného TS rovna n , pak prostorová složitost tohoto výpočtu není větší než $n + 1$.

Klíčová definice: Složitost výpočtů TS

Definice 5.1 Řekneme, že k -páskový DTS (resp. NTS) M přijímá jazyk L nad abecedou Σ v čase $T_M : \mathbb{N} \rightarrow \mathbb{N}$, jestliže $L = L(M)$ a M přijme (resp. může přijmout) každé $w \in L$ v nanejvýš $T_M(|w|)$ krocích.

Definice 5.2 Řekneme, že k -páskový DTS (resp. NTS) M přijímá jazyk L nad abecedou Σ v prostoru $S_M : \mathbb{N} \rightarrow \mathbb{N}$, jestliže $L = L(M)$ a M přijme (resp. může přijmout) každé $w \in L$ při použití nanejvýš $S_M(|w|)$ buněk pásky – nepočítáme zde buňky pásky, na nichž je zapsán vstup, za předpokladu, že obsah těchto buněk není modifikován.

❖ Zcela analogicky můžeme definovat vyčíslování určité funkce daným TS v určitém čase, resp. prostoru.

Analýza složitosti mimo prostředí TS

- ❖ V jiném výpočetním prostředí, než jsou TS, nemusí mít každá primitivní operace stejnou cenu.
- ❖ Velmi často se užívá tzv. **uniformní cenové kritérium**, kdy každé operaci přiřadíme stejnou cenu.
- ❖ Používá se ale např. také tzv. **logaritmické cenové kritérium**, kdy operaci manipulující operand o velikosti i , $i > 0$, přiřadíme cenu $\lfloor \lg i \rfloor + 1$:
 - Zohledňujeme to, že s rostoucí velikostí operandů roste cena operací – logaritmus odráží růst velikosti s ohledem na binární kódování (v n bitech zakódujeme 2^n hodnot).
- ❖ Analýza složitosti za takových předpokladů není zcela přesná, důležité ale obvykle je to, aby se **zachovala informace o tom, jak rychle roste čas/prostor potřebný k výpočtu v závislosti na velikosti vstupu**^a.

^aAlgoritmus A_1 , jehož nároky rostou pomaleji než u jiného algoritmu A_2 , nemusí být výhodnější než A_2 pro řešení malých instancí.

❖ Význam srovnávání rychlosti růstu složitosti výpočtů si snad nejlépe ilustrujeme na příkladu:

Příklad 5.2 Srovnání polynomiální (přesněji kvadratické – n^2) a exponenciální (2^n) časové složitosti:

délka vstupu n	časová složitost n^2	časová složitost 2^n
10	0.0001 s	0.0001 s
20	0.0004 s	0.1024 s
30	0.0009 s	1.75 min
40	0.0016 s	1.24 dne
50	0.0025 s	3.48 roku
60	0.0036 s	35.68 století
70	0.0049 s	3.65 mil. roků

Složitost výpočtů na TS a v jiných prostředích

❖ Při použití vhodného cenového kritéria, je složitost pro výpočetní modely blízké běžným počítačům (RAM, RASP stroje) **polynomiálně vázaná** se složitostí výpočtu na TS. Tudiž složitost výpočtu na TS není „příliš“ rozdílná oproti výpočtům na běžných počítačích.

- **RAM stroje** mají paměť s náhodným přístupem (jedna buňka obsahuje libovolné přirozené číslo), instrukce typu LOAD, STORE, ADD, SUB, MULT, DIV, vstup/výstup, skok atd. U RAM stroje je program součástí řízení stroje (okamžitě dostupný), u **RASP** je uložen v paměti stejně jako operandy.
- Funkce $f_1(n), f_2(n) : \mathbb{N} \rightarrow \mathbb{N}$ jsou **polynomiálně vázané**, existují-li polynomy $p_1(x)$ a $p_2(x)$ takové, že pro všechny hodnoty n je $f_1(n) \leq p_1(f_2(n))$ a $f_2(n) \leq p_2(f_1(n))$.
- **Logaritmické cenové kritérium** se uplatní, uvažujeme-li možnost násobení dvou čísel. Jednoduchým cyklem typu $A_{i+1} = A_i * A_i$ ($A_0 = 2$) jsme schopni počítat 2^{2^n} , což TS s omezenou abecedou není schopen provést v polynomiálním čase (pro uložení/načtení potřebuje projít 2^n buněk při použití binárního kódování).

❖ Ilustrujme si nyní určení složitosti v prostředí mimo TS:

Příklad 5.3 Uvažme následující implementaci **porovnávání řetězců**.

```
int str_cmp (int n, string a, string b) {
    int i;

    i = 0;
    while (i<n) {
        if (a[i] != b[i]) break;
        i++;
    }

    return (i==n);
}
```

- Pro určení složitosti aplikujme **uniformní cenové kritérium** např. tak, že budeme považovat složitost každého řádku programu (mimo deklarace) za jednotku. (Předpokládáme, že žádný cyklus není zapsán na jediném řádku.)
- **Složitost nejhoršího případu**

❖ Ilustrujme si nyní určení složitosti v prostředí mimo TS:

Příklad 5.4 Uvažme následující implementaci **porovnávání řetězců**.

```
int str_cmp (int n, string a, string b) {
    int i;

    i = 0;
    while (i<n) {
        if (a[i] != b[i]) break;
        i++;
    }

    return (i==n);
}
```

- Pro určení složitosti aplikujme **uniformní cenové kritérium** např. tak, že budeme považovat složitost každého řádku programu (mimo deklarace) za jednotku. (Předpokládáme, že žádný cyklus není zapsán na jediném řádku.)
- **Složitost nejhoršího případu lze pak snadno určit jako $3n + 3$:**
 - cyklus má 3 kroky, provede se n krát, tj. $3n$ kroků,
 - tělo funkce má 3 kroky (včetně testu ukončujícího cyklus).

Příklad 5.5 Uvažme následující implementaci řazení metodou insert-sort.

```
void insertsort(int n, int a[]) {
    int i, j, value;
    for (i=0; i<n; i++) {
        value = a[i];
        j = i - 1;
        while ((j >= 0) && (a[j] > value)) {
            a[j+1] = a[j];
            j = j - 1;
        }
        a[j+1] = value;
    }
}
```

- Pro určení složitosti aplikujme **uniformní cenové kritérium** např. tak, že budeme považovat složitost každého řádku programu (mimo deklarace) za jednotku. (Předpokládáme, že žádný cyklus není zapsán na jediném řádku.)
- **Složitost nejhoršího případu**

Příklad 5.6 Uvažme následující implementaci řazení metodou insert-sort.

```
void insertsort(int n, int a[]) {
    int i, j, value;
    for (i=0; i<n; i++) {
        value = a[i];
        j = i - 1;
        while ((j >= 0) && (a[j] > value)) {
            a[j+1] = a[j];
            j = j - 1;
        }
        a[j+1] = value;
    }
}
```

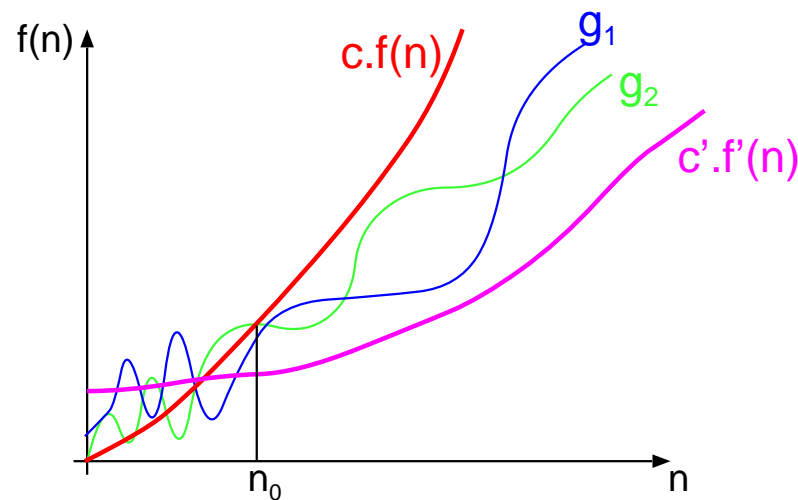
- Pro určení složitosti aplikujme **uniformní cenové kritérium** např. tak, že budeme považovat složitost každého řádku programu (mimo deklarace) za jednotku. (Předpokládáme, že žádný cyklus není zapsán na jediném řádku.)
- **Složitost lze pak určit jako $1.5n^2 + 3.5n + 1$:**
 - vnitřní cyklus má 3 kroky, provede se $0, 1, \dots, n - 1$ krát, tj.
 $3(0 + 1 + \dots + n - 1) = 3\frac{n}{2}(0 + n - 1) = 1.5n^2 - 1.5n$ kroků,
 - vnější cyklus má mimo vnitřní cyklus 5 kroků (včetně testu ukončujícího vnitřní cyklus) a provede se n krát, tj. $5n$ kroků,
 - jeden krok připadá na ukončení vnějšího cyklu.

Asymptotická omezení složitosti

- ❖ Při popisu složitosti algoritmů (výpočtů TS), chceme často **vyloučit vliv aditivních a multiplikativních konstant**:
 - různé aditivní a multiplikativní konstanty vzniknou velmi snadno „drobnými“ úpravami uvažovaných algoritmů,
 - např. srovnávání dvou řetězců je možné donekonečna zrychlovat tak, že budeme porovnávat současně 2, 3, 4, ... za sebou jdoucích znaků,
 - lineární komprese prostoru (rozšíření abecedy) a zrychlení výpočtu (před-vypočítání výsledků)
 - tyto úpravy ovšem nemají zásadní vliv na rychlost nárůstu časové složitosti (ve výše uvedeném případě nárůst zůstane kvadratický),
 - navíc při analýze složitosti mimo prostředí TS se dopouštíme jisté nepřesnosti již zavedením různých cenových kritérií.
- ❖ Proto se často složitost popisuje pomocí tzv. **asymptotických odhadů složitosti**

Definice 5.3 Necht' \mathcal{F} je množina funkcí $f : \mathbb{N} \rightarrow \mathbb{N}$. Pro danou funkci $f \in \mathcal{F}$ definujeme množiny funkcí $O(f(n))$, $\Omega(f(n))$ a $\Theta(f(n))$ takto:

- **Asymptotické horní omezení** funkce $f(n)$ je množina $O(f(n)) = \{g(n) \in \mathcal{F} \mid \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow 0 \leq g(n) \leq c \cdot f(n)\}$.
- **Asymptotické dolní omezení** funkce $f(n)$ je množina $\Omega(f(n)) = \{g(n) \in \mathcal{F} \mid \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow 0 \leq c \cdot f(n) \leq g(n)\}$.
- **Asymptotické oboustranné omezení** funkce $f(n)$ je množina $\Theta(f(n)) = \{g(n) \in \mathcal{F} \mid \exists c_1, c_2 \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow 0 \leq c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n)\}$.



Příklad 5.7 S využitím asymptotických odhadů složitosti můžeme říci, že složitost našeho srovnání řetězců patří do $O(n)$ a složitost insert-sort do $O(n^2)$.

Definice 5.4 Necht' \mathcal{F} je množina funkcí $f : \mathbb{N} \rightarrow \mathbb{N}$. Pro danou funkci $f \in \mathcal{F}$ definujeme množiny funkcí $O(f(n))$, $\Omega(f(n))$ a $\Theta(f(n))$ takto:

- **Asymptotické horní omezení** funkce $f(n)$ je množina $O(f(n)) = \{g(n) \in \mathcal{F} \mid \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow 0 \leq g(n) \leq c \cdot f(n)\}$.
- **Asymptotické dolní omezení** funkce $f(n)$ je množina $\Omega(f(n)) = \{g(n) \in \mathcal{F} \mid \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow 0 \leq c \cdot f(n) \leq g(n)\}$.
- **Asymptotické oboustranné omezení** funkce $f(n)$ je množina $\Theta(f(n)) = \{g(n) \in \mathcal{F} \mid \exists c_1, c_2 \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow 0 \leq c_1 \cdot f(n) \leq g(n) \leq c_2 \cdot f(n)\}$.

❖ Asymptotické horní omezení exponenciálních funkcí

$$2^{O(f(n))} = \{g(n) \in \mathcal{F} \mid \exists c \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow g(n) \leq 2^{c \cdot f(n)}\}$$

- relaxuje aditivní a multiplikatívni konstanty v exponentu
- dovoluje dále zjednodušit analýzu složitosti

Třídy složitosti

Složitost problémů

- ❖ Přejdeme nyní od složitosti konkrétních algoritmů (Turingových strojů) ke **složitosti problémů**.
- ❖ Třídy složitosti zavádíme jako **prostředek ke kategorizaci (vytvoření hierarchie) problémů dle jejich složitosti**, tedy dle toho, jak dalece efektivní algoritmy můžeme navrhnout pro jejich rozhodování (u funkcí můžeme analogicky mluvit o složitosti jejich vyčíslování).
- ❖ Podobně jako u určování typu jazyka se budeme snažit **zařadit problém do co nejnižší třídy složitosti** – tedy určit složitost problému jako složitost jeho nejlepšího možného řešení.
- ❖ Zařazení **různých problémů do téže třídy složitosti** může odhalit různé vnitřní podobnosti těchto problémů a může umožnit řešení jednoho převodem na druhý (a pragmatické využití např. různých již vyvinutých nástrojů).

Třídy složitosti

Definice 5.5 Mějme dány funkce $t, s : \mathbb{N} \rightarrow \mathbb{N}$ a necht' T_M , resp. S_M , značí časovou, resp. prostorovou, složitost TS M . Definujeme následující časové a prostorové třídy složitosti deterministických a nedeterministických TS:

- $DTime[t(n)] = \{L \mid \exists k\text{-páskový DTS } M : L = L(M) \text{ a } T_M \in O(t(n))\}$.
- $NTime[t(n)] = \{L \mid \exists k\text{-páskový NTS } M : L = L(M) \text{ a } T_M \in O(t(n))\}$.
- $DSpace[s(n)] = \{L \mid \exists k\text{-páskový DTS } M : L = L(M) \text{ a } S_M \in O(s(n))\}$.
- $NSpace[s(n)] = \{L \mid \exists k\text{-páskový NTS } M : L = L(M) \text{ a } S_M \in O(s(n))\}$.

❖ Definici tříd složitosti pak přímočaře zobecňujeme tak, aby mohly být založeny na množině funkcí, nejen na jedné konkrétní funkci.

❖ *Poznámka:* Dále ukážeme, že použití více pásek přináší jen polynomiální zrychlení. Na druhou stranu ukážeme, že zatímco nedeterminismus nepřináší nic podstatného z hlediska vyčíslitelnosti, může přinášet mnoho z hlediska složitosti.

Časově/prostorově zkonstruovatelné funkce

❖ Třídy složitosti obvykle budujeme nad tzv. časově/prostorově zkonstruovatelnými funkcemi:

- Důvodem zavedení časově/prostorově zkonstruovatelných funkcí je dosáhnout intuitivní hierarchické struktury tříd složitosti – např. odlišení tříd $f(n)$ a $2^{f(n)}$, což, jak uvidíme, pro třídy založené na obecných rekurzivních funkcích nelze.

Definice 5.6 Funkci $t : \mathbb{N} \rightarrow \mathbb{N}$ nazveme časově zkonstruovatelnou (time constructible), jestliže existuje vícepáskový TS, jenž pro libovolný vstup w zastaví po přesně $t(|w|)$ krocích.

Definice 5.7 Funkci $s : \mathbb{N} \rightarrow \mathbb{N}$ nazveme prostorově zkonstruovatelnou (space constructible), jestliže existuje vícepáskový TS, jenž pro libovolný vstup w zastaví s využitím přesně $s(|w|)$ buněk pásky.

❖ *Poznámka:* Pokud je jazyk L nad Σ přijímán strojem v čase/prostoru omezeném časově/prostorově zkonstruovatelnou funkcí, pak je také přijímán strojem, který pro každé $w \in \Sigma^*$ vždy zastaví:

- U časového omezení $t(n)$ si stačí předem spočítat, jaký je potřebný počet kroků a zastavit po jeho vyčerpání.
- U prostorového omezení spočteme z $s(n)$, $|Q|$, $|\Delta|$ maximální počet konfigurací, které můžeme vidět a z toho také plyne maximální počet kroků, které můžeme udělat, aniž bychom cyklili.

Nejběžněji užívané třídy složitosti

❖ Deterministický/nedeterministický polynomiální čas:

$$\mathbf{P} = \bigcup_{k=0}^{\infty} DTime(n^k)$$

$$\mathbf{NP} = \bigcup_{k=0}^{\infty} NTime(n^k)$$

❖ Deterministický/nedeterministický polynomiální prostor:

$$\mathbf{PSPACE} = \bigcup_{k=0}^{\infty} DSpace(n^k)$$

\equiv

$$\mathbf{NPSPACE} = \bigcup_{k=0}^{\infty} NSpace(n^k)$$

Třídy pod a nad polynomiální složitostí

❖ Deterministický/nedeterministický logaritmický prostor:

$$\mathbf{LOGSPACE} = \bigcup_{k=0}^{\infty} DSpace(k \lg n)$$

$$\mathbf{NLOGSPACE} = \bigcup_{k=0}^{\infty} NSpace(k \lg n)$$

❖ Deterministický/nedeterministický exponenciální čas:

$$\mathbf{EXP} = \bigcup_{k=0}^{\infty} DTime(2^{n^k})$$

$$\mathbf{NEXP} = \bigcup_{k=0}^{\infty} NTime(2^{n^k})$$

❖ Deterministický/nedeterministický exponenciální prostor:

$$\mathbf{EXPSPACE} = \bigcup_{k=0}^{\infty} DSpace(2^{n^k})$$

\equiv

$$\mathbf{NEXPSPACE} = \bigcup_{k=0}^{\infty} NSpace(2^{n^k})$$

Třídy nad exponenciální složitostí

❖ Det./nedet. k -exponenciální čas/prostor založený na věži exponenciál $2^{2^{\dots^2}}$ o výšce k :

$$\mathbf{k\text{-EXP}} = \bigcup_{l=0}^{\infty} DTime(2^{2^{\dots^{2^{n^l}}}})$$

$$\mathbf{k\text{-NEXP}} = \bigcup_{l=0}^{\infty} NTime(2^{2^{\dots^{2^{n^l}}}})$$

$$\mathbf{k\text{-EXPSPACE}} = \bigcup_{l=0}^{\infty} DSpace(2^{2^{\dots^{2^{n^l}}}}) \equiv \mathbf{k\text{-NEXPSPACE}} = \bigcup_{l=0}^{\infty} NSpace(2^{2^{\dots^{2^{n^l}}}})$$

$$\mathbf{ELEMENTARY} = \bigcup_{k=0}^{\infty} \mathbf{k\text{-EXP}}$$

Vrchol hierarchie tříd složitosti

❖ Na vrcholu hierarchie tříd složitosti se pak hovoří o obecných třídách jazyků (funkcí), se kterými jsme se již setkali:

- třída primitivně-rekurzivních funkcí **PR** (implementovatelných pomocí zanořených cyklů s pevným počtem opakování – `for i=... to ...`),
- třída μ -rekurzivních funkcí (rekurzivních jazyků) **R** (implementovatelných pomocí cyklů s předem neurčeným počtem opakování – `while ...`) a
- třída rekurzivně vyčíslitelných funkcí (rekurzivně vyčíslitelných jazyků) **RE**.

Příklad na analýzu složitosti I

Najděte co nejmenší k takové, aby platilo

a) $L \in DTIME[n^3] \Rightarrow \{w \in \Sigma^* \mid \exists w_1, w_2, w_3 \in L \text{ t.ž. } w = w_1 w_2 w_3\} \in DTIME[n^k]$

b) $L \in DTIME[n^3] \Rightarrow \{w \in \Sigma^* \mid \exists w_1, w_2, w_3 \in L \text{ t.ž. } w = w_1 w_2 w_3\} \in NTIME[n^k]$.

Uveďte hlavní myšlenku důkazu (zahrnující konstrukci požadovaného Turingova stroje a jeho časovou analýzu), že pro nalezené k implikace platí. Nedokazujte minimalitu k .

Příklad na analýzu složitosti I

Najděte co nejmenší k takové, aby platilo

$$\text{a) } L \in DTIME[n^3] \Rightarrow \{w \in \Sigma^* \mid \exists w_1, w_2, w_3 \in L \text{ t.ž. } w = w_1 w_2 w_3\} \in DTIME[n^k]$$

$$\text{b) } L \in DTIME[n^3] \Rightarrow \{w \in \Sigma^* \mid \exists w_1, w_2, w_3 \in L \text{ t.ž. } w = w_1 w_2 w_3\} \in NTIME[n^k].$$

Uveďte hlavní myšlenku důkazu (zahrnující konstrukci požadovaného Turingova stroje a jeho časovou analýzu), že pro nalezené k implikace platí. Nedokazujte minimalitu k .

Označíme $L' = \{w \in \Sigma^* \mid \exists w_1, w_2, w_3 \in L \text{ t.ž. } w = w_1 w_2 w_3\}$

a) Sestrojíme DTS M' akceptující M' . M' systematicky prochází všechny rozdělení vstupního slova w ($|w| = n$) na podslova w_1, w_2 a w_3 (tj. $w = w_1 w_2 w_3$). Těchto rozdělení je $O(n^2)$ (potřebujeme umístit 2 "zarážky"). Konstrukce každého rozdělení je v $O(n)$. Pro každé rozdělení zkontrolujeme, zda $w_1 \in L \wedge w_2 \in L \wedge w_3 \in L$, tj. 3-krát voláme DTS M , který akceptuje L v $O(n^3)$. Celkově složitost M' je: $O(n^2) \cdot (O(n) + 3 \cdot O(n^3)) = O(n^5)$. Tudíž $k = 5$.

b) Sestrojíme NTS M' akceptující M' . M' nedeterministicky rozdělí vstupní slovo w na podslova w_1, w_2 a w_3 (tj. $w = w_1 w_2 w_3$). Toto rozdělení zkonstruuje a ověří požadovanou vlastnost (tj. $w_1 \in L \wedge w_2 \in L \wedge w_3 \in L$) v $O(n) + 3 \cdot O(n^3)$ a tudíž celková složitost M' je v $O(n^3)$. Tudíž $k = 3$.

Příklad na analýzu složitosti II

Uvažme jazyk

$L_{NE} = \{(\Phi_1, \Phi_2) \mid \Phi_1, \Phi_2 \text{ jsou výrokové formule v konjunktivní normální formě, pro které existuje valuace proměnných } \bar{v} \text{ taková, že } \Phi_1(\bar{v}) \neq \Phi_2(\bar{v})\}.$

Poznámka: $\Phi_i(\bar{v}) \in \{\text{true}, \text{false}\}$ označuje, zda je formule Φ_i pravdivá při valuaci proměnných \bar{v} .

Příklad na analýzu složitosti II

Uvažme jazyk

$L_{NE} = \{(\Phi_1, \Phi_2) \mid \Phi_1, \Phi_2 \text{ jsou výrokové formule v konjunktivní normální formě, pro které existuje valuace proměnných } \bar{v} \text{ taková, že } \Phi_1(\bar{v}) \neq \Phi_2(\bar{v})\}.$

Poznámka: $\Phi_i(\bar{v}) \in \{\text{true}, \text{false}\}$ označuje, zda je formule Φ_i pravdivá při valuaci proměnných \bar{v} .

Nejdříve ukážeme, že $L_{NE} \in \mathbf{EXP}$.

- Sestrojíme DTS M , t.ž. $L(M) = L_{NE}$ a M pracuje v exponenciálním čase.
- M postupně generuje (např. v lexikografickém pořadí) jednotlivé valuace \bar{v} proměnných z formule Φ_1 a Φ_2 .
- Pro každou valuaci \bar{v} M vyhodnotí $\Phi_1(\bar{v})$ a $\Phi_2(\bar{v})$ (lineární průchod oběma formulemi).
- Pokud $\Phi_1(\bar{v}) \neq \Phi_2(\bar{v})$, tak M akceptuje. Pokud prošel všechny valuace a neakceptoval, tak zamítne.
- Pokud Φ_1 a Φ_2 obsahuje n proměnných, pak existuje 2^n různých valuací. Složitost M je tedy v $O(2^n \cdot n) \subseteq O(2^{2n}) \subseteq \mathbf{EXP}$.

Příklad na analýzu složitosti II

Uvažme jazyk

$L_{NE} = \{(\Phi_1, \Phi_2) \mid \Phi_1, \Phi_2 \text{ jsou výrokové formule v konjunktivní normální formě, pro které existuje valuace proměnných } \bar{v} \text{ taková, že } \Phi_1(\bar{v}) \neq \Phi_2(\bar{v})\}.$

Nyní ukážeme, že $L_{NE} \in \mathbf{NP}$.

- Sestrojíme NTS M , t.ž. $L(M) = L_{NE}$ a M pracuje v polynomiálním čase.
- M nedeterministicky zvolí (uhádne) valuaci \bar{v} proměnných z formule Φ_1 a Φ_2 .
- M vyhodnotí $\Phi_1(\bar{v})$ a $\Phi_2(\bar{v})$ (lineární průchod oběma formullemi).
- Pokud $\Phi_1(\bar{v}) \neq \Phi_2(\bar{v})$, tak M akceptuje. V opačném případě zamítne.
- Složitost M je tedy v $O(n)$. Ukázali jsme, že pro L_{NE} existuje polynomiální verifikátor.

Připomeňme, že $L_{NE} \in \mathbf{EXP}$ plyne taktéž přímo z $L_{NE} \in \mathbf{NP}$.

Těžkost a Úplnost

Jazyky \mathcal{C} -těžké a \mathcal{C} -úplné

❖ Až doposud jsme třídy používali jako horní omezení složitosti problémů. Všimněme si nyní **omezení dolního** – to zavedeme pomocí redukovatelnosti třídy problémů na daný problém.

Definice 5.8 Necht' \mathcal{R} je třída funkcí. Jazyk $L_1 \subseteq \Sigma_1^*$ je \mathcal{R} **redukovatelný** (přesněji \mathcal{R} many-to-one reducible) na jazyk $L_2 \subseteq \Sigma_2^*$, což zapisujeme $L_1 \leq_{\mathcal{R}}^m L_2$, jestliže existuje funkce f z \mathcal{R} taková, že $\forall w \in \Sigma_1^* : w \in L_1 \Leftrightarrow f(w) \in L_2$.

Definice 5.9 Necht' \mathcal{R} je třída funkcí a \mathcal{C} třída jazyků. Jazyk L_0 je **\mathcal{C} -těžký** (**\mathcal{C} -hard**) vzhledem k \mathcal{R} redukovatelnosti, jestliže $\forall L \in \mathcal{C} : L \leq_{\mathcal{R}}^m L_0$.

Definice 5.10 Necht' \mathcal{R} je třída funkcí a \mathcal{C} třída jazyků. Jazyk L_0 nazveme **\mathcal{C} -úplný** (**\mathcal{C} -complete**) vzhledem k \mathcal{R} redukovatelnosti, jestliže $L_0 \in \mathcal{C}$ a L_0 je \mathcal{C} -těžký (\mathcal{C} -hard) vzhledem k \mathcal{R} redukovatelnosti.

Nejběžnější typy úplnosti

❖ Uvedme nyní **nejběžněji používané typy úplnosti** – všimněme si, že je použita redukovatelnost dostatečně silná na to, aby bylo možné najít úplné problémy vůči ní a na druhou stranu nebyly příslušné třídy triviálně redukovány na jejich (možné) podtřídy:

- **NP, PSPACE, EXP** úplnost je definována vůči **polynomiální redukovatelnosti** (tj. redukovatelnosti pomocí DTS pracujících v polynomiálním čase),
- **P, NLOGSPACE** úplnost definujeme vůči **redukovatelnosti v deterministickém logaritmickém prostoru**,
- **NEXP** úplnost definujeme vůči **exponenciální redukovatelnosti** (tj. redukovatelnosti pomocí DTS pracujících v exponenciálním čase).

Příklady složitosti problémů

❖ Příklady **LOGSPACE** problémů:

- existence **cesty** mezi dvěma uzly v **neorientovaném grafu**.

❖ Příklady **NLOGSPACE-úplných** problémů:

- existence **cesty** mezi dvěma uzly v **orientovaném grafu**,
- **2-SAT** (*SATisfiability*), tj. splnitelnost výrokových formulí tvaru konjunkce disjunkcí dvou literálů (literál je výroková proměnná nebo její negace), např.
 $(x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_3)$.

❖ Příklady **P-úplných** problémů:

- **splnitelnost konjunkce Hornových klauzulí** $(p \wedge q \wedge \dots \wedge t) \Rightarrow u$, kde p, q, \dots jsou atomické formule výrokové logiky (výrokové proměnné),
 - nerozhodnutelné v predikátové podobě
- **náležitost řetězce** do jazyka **bezkontextové gramatiky**: algoritmus „CYK“ založený na dynamickém programování (Cocke, Younger, Kasami) – $O(n^3)$,
- **topologické uspořádání** – pořadí uzlů při průchodu grafem do hloubky (pro dané řazení přímých následníků).

❖ Příklady **NP**-úplných problémů:

- splnitelnost Booleovských formulí (**SAT**, **3SAT**)
- řada grafových a množinových problémů (viz výše)
- **problém obchodního cestujícího**,
- „**knapsack**“ – máme položky s váhou a hodnotou, maximalizujeme hodnotu tak, aby váha nepřekročila určitou mez.

❖ Příklady **co-NP**-úplných problémů:

- **ekvivalence regulárních výrazů bez iterace**.

❖ Příklady **PSPACE**-úplných problémů:

- ekvivalence regulárních výrazů,
- náležitost řetězce do jazyka kontextové gramatiky,
- inkluze jazyků nedeterministických konečných automatů,
- model checking formulí lineární temporální logiky (LTL – výroková logika doplněná o temporální operátory *until*, *always*, *eventually*, *next-time*) vůči velikosti formule.

❖ Příklady **EXP**-úplných problémů:

- inkluze jazyka bezkontextové gramatiky v jazyce NKA (opačná \subseteq nerozh.),
- inkluze nedeterministických konečných *stromových automatů*, zobecňujících přechody KA do podoby $p \xrightarrow{a} (q_1, \dots, q_n)$,
- inkluze pro tzv. *visibly push-down* jazyky (operace push/pop, které provádí přijímající automat, jsou součástí vstupního řetězce),
- nejlepší tah v šachu (zobecněno na šachovnici $n \times n$),
- model checking procesů s neomezeným zásobníkem (rekurzí) vůči zafixované formuli logiky větvícího se času (CTL) – tj. EXP ve velikosti procesu.

❖ Příklady **EXSPACE**-úplných problémů:

- ekvivalence regulárních výrazů doplněných o operaci kvadrát (tj. r^2).

❖ **k-EXP / k-EXPSPACE:**

- rozhodování splnitelnosti formulí **Presburgerovy aritmetiky** – tj. celočíselné aritmetiky se sčítáním, porovnáváním (ne násobením – to vede na tzv. Peanovu aritmetiku, která je již nerozhodnutelná) a kvantifikací prvního řádu (např. $\forall x, y : x \leq x + y$) je problém, který je v **3-EXP (2-EXPSPACE-úplný)**.

❖ Problémy mimo **ELEMENTARY:**

- ekvivalence regulárních výrazů doplněných o negaci,
- rozhodování splnitelnosti formulí logiky **WS1S** – celočíselná aritmetika s operací +1 a kvantifikací prvního a druhého řádu (tj. pro každou/existuje hodnota, resp. množina hodnot, taková, že ... – např. $\exists A : \forall x \in A : x + 1 \notin A$),
- verifikace dosažitelnosti v tzv. *Lossy Channel Systems* – procesy komunikující přes neomezenou, ale ztrátovou frontu (cokoliv se může kdykoliv ztratit).

Prvočíselný rozklad

- ❖ Problém rozkladu daného celého čísla na součin prvočísel.
- ❖ Velmi důležitý problém pro kryptografii.
- ❖ Rozhodovací problém: Rozhodni, jestli pro daná čísla n a k existuje číslo $1 < p < k$, které dělí číslo n .
- ❖ Ví se, že je v $\mathbf{NP} \cap \mathbf{co-NP}$.
- ❖ Neví se, zda je v \mathbf{P} , ani zda je \mathbf{NP} -úplný.
- ❖ Existuje polynomiální algoritmus pro test je dané číslo prvočíslo.
- ❖ Existuje polynomiální algoritmus pro kvantové počítače, který počítá rozklad.