# Přednášky ADS 2.

Jan Hric, KTIML MFF UK
e-mail: Jan.Hric@mff.cuni.cz
http://ktiml.mff.cuni.cz/~hric/vyuka/alg/ads2pr.pdf

22. prosince 2024

## Algoritmy a datové struktury 2.

Sylabus:

- Vyhledávání vzorků v textu: alg. Aho-Corasicková
- Toky v sítích
- Hradlové sítě: aritmetické, třídící
- Rychlá (diskrétní) Fourierova transformace
- NP-úplnost
- Aproximační algoritmy, pravděpodobnostní alg.
- Kryptografie, RSA
- Geometrické algoritmy: konvexní obal

... algoritmy v širším smyslu
22. prosince 2024

## Vyhledávání vzorků.
## Vyhledávání vzorků v textu: Algoritmus Aho-Corasick

Pojmy:

Pracujeme nad danou konečnou abecedou $\Sigma$ (tj. množina znaků).
$\Sigma^*$ je množina konečných slov nad $\Sigma$ (tj. posloupnosti znaků ze $\Sigma$).
Délka slova $w = x_1x_2...x_n \in \Sigma^*$ je $length(w) = n$, tj. počet znaků
Skládaní slov $u = u_1...u_m$ a $w = w_1...w_n$ vrací $uw = u_1...u_mw_1...w_n$, není komutativní
prázdné slovo $\epsilon$, má délku 0, pro $\forall w \in \Sigma^*$ platí $\epsilon w = w\epsilon = w$
$u \in \Sigma^*$ je předpona $w \in \Sigma^*$ pokud $\exists z \in \Sigma^* : uz = w$
$u \in \Sigma^*$ je přípona $w \in \Sigma^*$ pokud $\exists z \in \Sigma^* : zu = w$
pokud $z \neq \epsilon$, je přípona, resp. předpona vlastní, jinak nevlastní

Úloha:

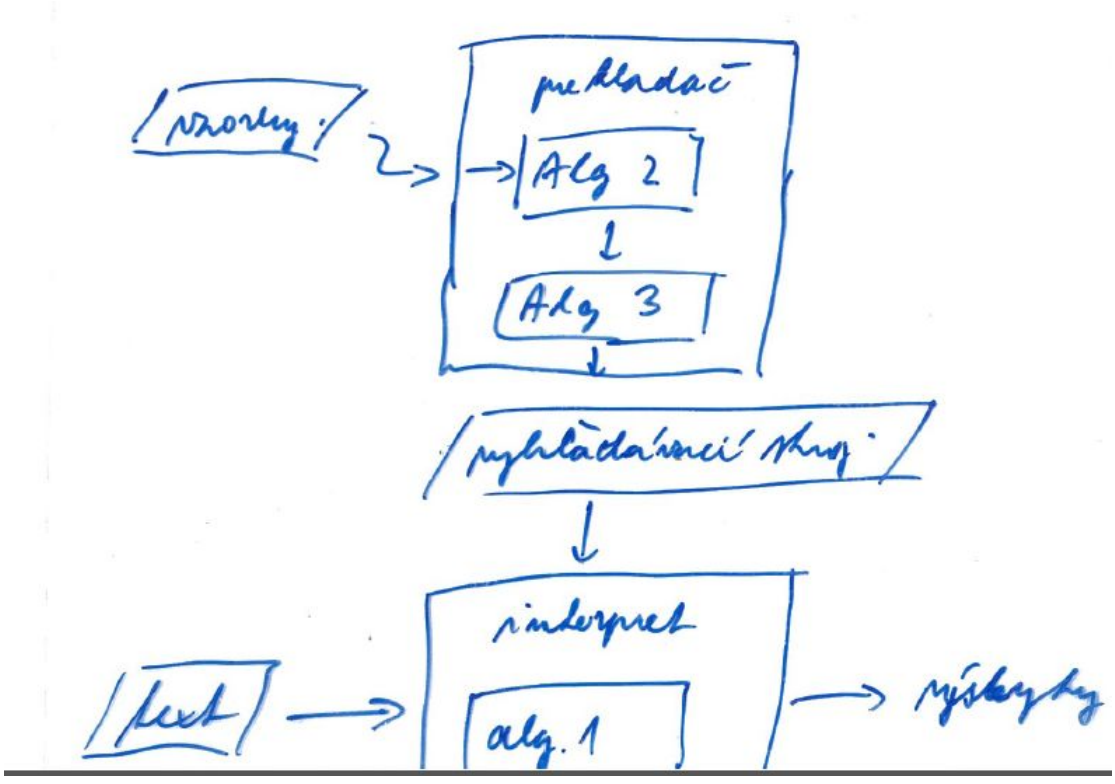
Vstup: abeceda $\Sigma$, prohledávané slovo $x = x_1x_2...x_n \in \Sigma^*$ a hledané vzorky $K = \{y_1, ..., y_k\}$, kde $y_p = y_{p,1}...y_{p,l(p)} \in \Sigma^*$, $p \in \{1...k\}$
Výstup: všechny výskyty vzorků z $K$ v $x$, tj. dvojice $\langle i, p\rangle$ tž. $y_p$ je příponou $x_1..x_i$
$l = |K| = \Sigma_{p=1}^{k}\, l(y_p)$ je velikost množiny vzorků

Naivní algoritmus.

1 for $p := 1$ to $k$ do
2  for $i := 1$ to $n - l(p) + 1$ do
3  $j := 0$
4  while $j < l(p)$ and $x_{i+j} = y_{p,j}$ do
5  $j := j + 1$
6  if $j = l(p)$ then Report($\langle i, p \rangle$)

Složitost: $O(n.l)$

Myšlenka AC: Vyrobíme speciální algoritmus ($\approx$ konečný automat) pro danou (pevnou) množinu vzorků $K$ v čase $O(l)$, který najde vzorky v textu $x$ v $O(n)$.




Obrázek 1: Celkové schéma algoritmu

Překladač: alg. 2 a 3 - ze vzorků do vyhledávacího stroje

Interpret vyhledávacího stroje: Algoritmus 1 - vstup je *popis* stroje a výstup jsou nalezené vzorky

Obecně: překladač/generátor, použití DSL - Doménově Specifický jazyk (Domain Specific Language)

Možnosti interpretace: vyhledávací stroj (tj. výsledek překladu) je

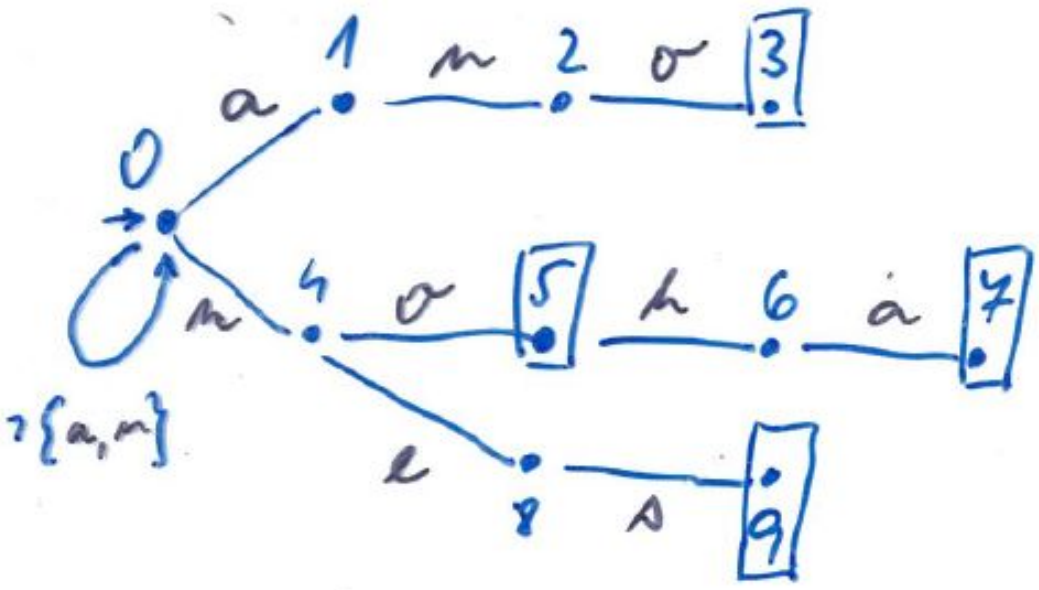
1. abstraktní stroj: a) datová struktura anebo b) (serializovaný) mezikód, který se interpretuje

2. a) zdroják anebo b) vykonatelný kód, typicky využívající runtime-knihovnu implementující specifické operace DSL (zde např. porovnání znaků, použití funkcí)

Příklad 1.

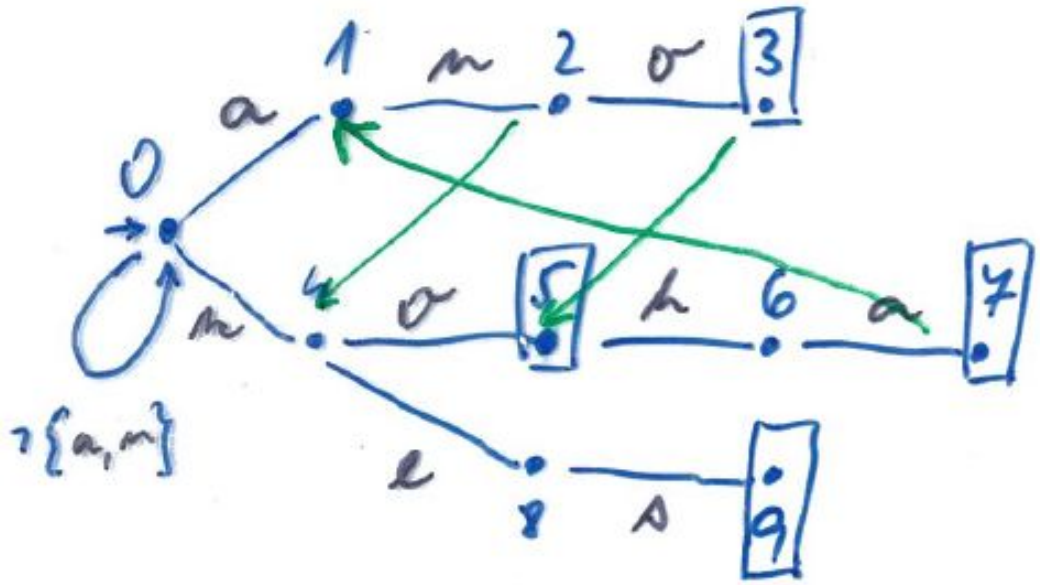
Vzorky: $K = \{ano, no, noha, nes\}$

Dopředná funkce $g$ (a pomocná funkce $o$), výstup alg. 2.



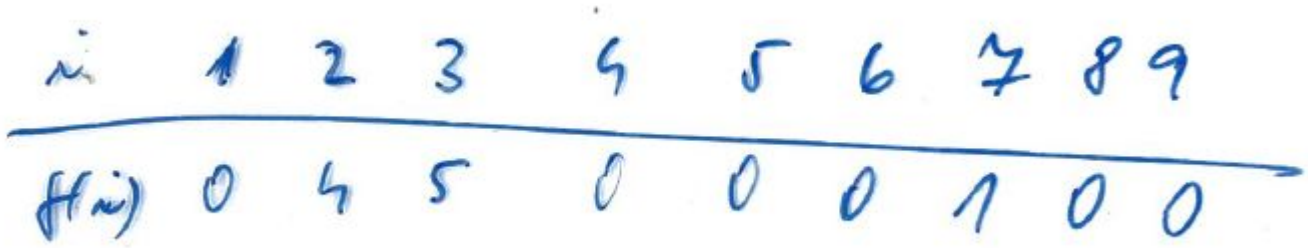

Obrázek 2: Dopředná funkce $g$

Zpětná funkce $f$, výstup alg. 3.



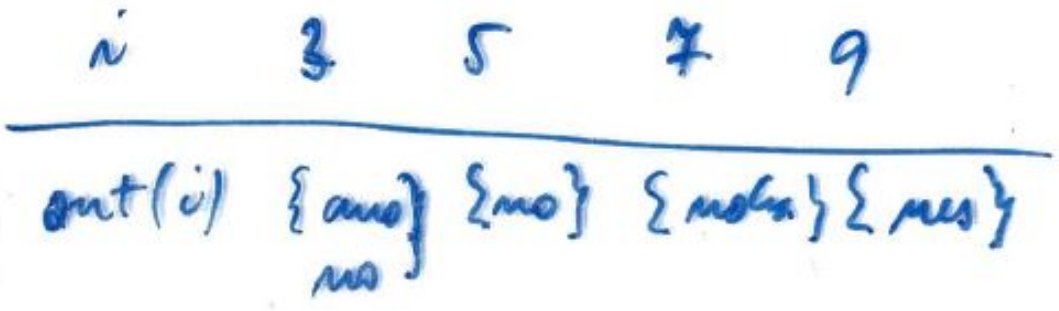

Obrázek 3: Přidána zpětná funkce $f$

Zpětná funkce $f$ tabulkou:




Obrázek 4: Zpětná funkce $f$ tabulkou

# Výstupní funkce *out* tabulkou




Obrázek 5: Výstupní funkce *out* tabulkou

Vyhledávací stroj je čtveřice $M = (Q, g, f, out)$, kde
$Q = \{0, ..., q\}$ je množina stavů
$g : Q \times \Sigma \to Q \cup \{\perp\}$ je (dopředná) přechodová funkce, pro kterou platí $\forall x \in \Sigma : g(0, x) \in Q$, (symbol $\perp$ znamená "nedefinováno", přechod ze stavu 0 je definován pro všechna písmena)
$f : Q \to Q$ je zpětná funkce, pro kterou platí $f(0) = 0$, (používá se, pokud $g$ vrátí $\perp$)
$out : Q \to P(K)$ je výstupní funkce, pro daný stav vydá podmnožinu vzorků

Alg. 1: Vyhledávací stroj
vstup: $x = x_1x_2...x_n \in \Sigma^*$, vzorky $K = \{y_1, ..., y_j\}$, $M = (Q, g, f, out)$
výstup: dvojice $\langle i, p \rangle$
1 $stav := 0$
2 for $i := 1$ to $n$ do // přes písmena textu
3  while $g(stav, x_i) = \perp$ do
4  $stav := f(stav)$
5  $stav := g(stav, x_i)$
6  forall $y \in out(stav)$ do
7  Report($\langle i, y \rangle$)

pozn:
- výstup vracíme na místě, kde vzorek končí
- pokud je nějaký vzorek příponou jiného, pak potřebujeme vracet několik vzorků - proto *out* vrací množinu vzorků; do stavu reprezentovaného kratším vzorkem se nemusíme dostat, proto vydáváme vzorek i když je vlastní příponou aktuálního stavu
- vzorky vracíme pouze ve stavech, kam jsme přišli pomocí $g$

(ř.6); vzorky vydávané po průchodu zpětnou funkcí $f$ by se opakovali
- podmínky na $g$ a $f$ ve stavu 0 jsou "zarážky"

Vlastnosti

1. přechodová funkce $g$: graf funkce $g$ je kromě smyčky ve stavu 0 ohodnocený strom, pro který
   - stav 0 je kořen stromu
   - každá cesta z kořene je ohodnocena nějakou předponou nějakého vzorku z $K$
   - každá předpona každého vzorku z $K$ popisuje cestu z kořene do nějakého (jediného) stavu $s$; říkáme, že předpona $u$ *reprezentuje* stav $s$, speciálně prázdné slovo $\epsilon$ reprezentuje stav 0

2. zpětná funkce $f$: pro každý stav $s$ reprezentovaný slovem $u$ je $f(s)$ reprezentován nejdelší vlastní příponou $u$, která je zároveň předponou nějakého vzorku v $K$

3. výstupní funkce: pro každý stav $s$ reprezentovaný $u$ a každý vzorek $y \in K$ platí: $y \in out(s)$ právě když $y$ je příponou $u$.

Správnost:
invariant: Algoritmus prochází stavy, které reprezentují nejdelší příponu přečtené části textu, která je zároveň předponou nějakého vzorku z $K$ a vydává všechny nalezené vzorky

Složitost.
Chceme algoritmus lineární v délce vstupu a délce výstupu. Reprezentace výstupu: odkazy na vzorky, případně. na množiny

vzorků.
Idea: Těžká část je počet použití $f$ (ř. 3, 4). Musíme je spočítat za celý výpočet dohromady (v nejhorším případě).
(Pokud odhadneme složitost pro všechna $i$ nejhorším případem $O(l)$, nedostaneme v obecnosti $O(n)$. Metapoznámka: Takto špatně zvoleným postupem jsme nedokázali, že alg. je pomalý - pouze takto použitý odhad je nepřesný)
Odhad složitosti pomocí potenciálu:
Hloubka aktuálního stavu je potenciál. Zpracování písmena pomocí $g$ zvyšuje potenciál, použití $f$ snižuje. Chceme ukázat, že celkový počet použití funkcí $f$ (a $g$) je $O(n)$.
(neformálně amortizovaná složitost: v celkovém průběhu je počet použití $f$ odhadnutý $O(n)$, tj. amortizovaně $O(1)$ na 1 znak vstupu)

Tvrzení: Počet (#) použití $f \leq n$
Dk: $n =$ # použití $g$ z alg. 1
$\geq$ celkový přírůstek potenciálu // $g$ zvyšuje hloubku nejvýš o 1, Ve stavu 0 někdy nezvyšuje
= celkový pokles potenciálu + koncová hodnota // poč. hodnota je 0
$\geq$ celkový pokles potenciálu // koncová hodnota $\geq 0$
$\geq$ # použití $f$ každé použití $f$ snižuje hloubku aspoň o 1.

Alg. 2: konstrukce vyhl. stroje (funkce $g$ a pomocná $o$)
vstup: vzorky $K$
výstup: stavy $Q = \{0, ..., q\}$,
přech. fce $g : Q \times \Sigma \to Q \cup \{\perp\}$,
pomocná výstupní $o : Q \to P(K)$

01 procedure Enter($y_1 ... y_m$) // připojení slova $y_p$ délky $m$
02 $stav := 0; j := 1$
03 while $j \leq m$ and $g(stav, y_j) \neq \perp$ do
04 $stav := g(stav, y_j)$ // již známá cesta
05 $j++$
06 for $p := j$ to $m$ do // nová větev
07 $q++, Q := Q \cup \{q\}$ //nový stav
08 forall $x \in \Sigma$ do $g(q, x) = \perp$ // přechody nedefinovány (implicitně)
09 $g(stav, y_p) = q$ // prodloužení větve
10 $stav := q$ // přesun do nového stavu
11 $o(stav) := \{y_p\}$
11.1 end Enter

12 $Q := \{0\}; q := 0$ // init počet stavů
13 forall $x \in \Sigma$ do $g(0, x) := \perp$
14 for $i := 1$ to $k$ do Enter($y_i$) //vložení slov
15 forall $x \in \Sigma$ do if $g(0, x) = \perp$
16 then $g(0, x) := 0$ // zarážka v 0

Alg. 3: konstrukce zpětné funkce $f$ a výstupní $out$
vstup: stavy $Q = \{0, ..., q\}$,
přech. fce $g : Q \times \Sigma \to Q \cup \{\perp\}$,
pomocná výstupní $o : Q \to P(K)$ // z alg. 2
výstup: $f : Q \to Q$
$out : Q \to P(K)$
01 queue := prázdná fronta // init
02 $f(0) := 0$, $out(0) := \emptyset$

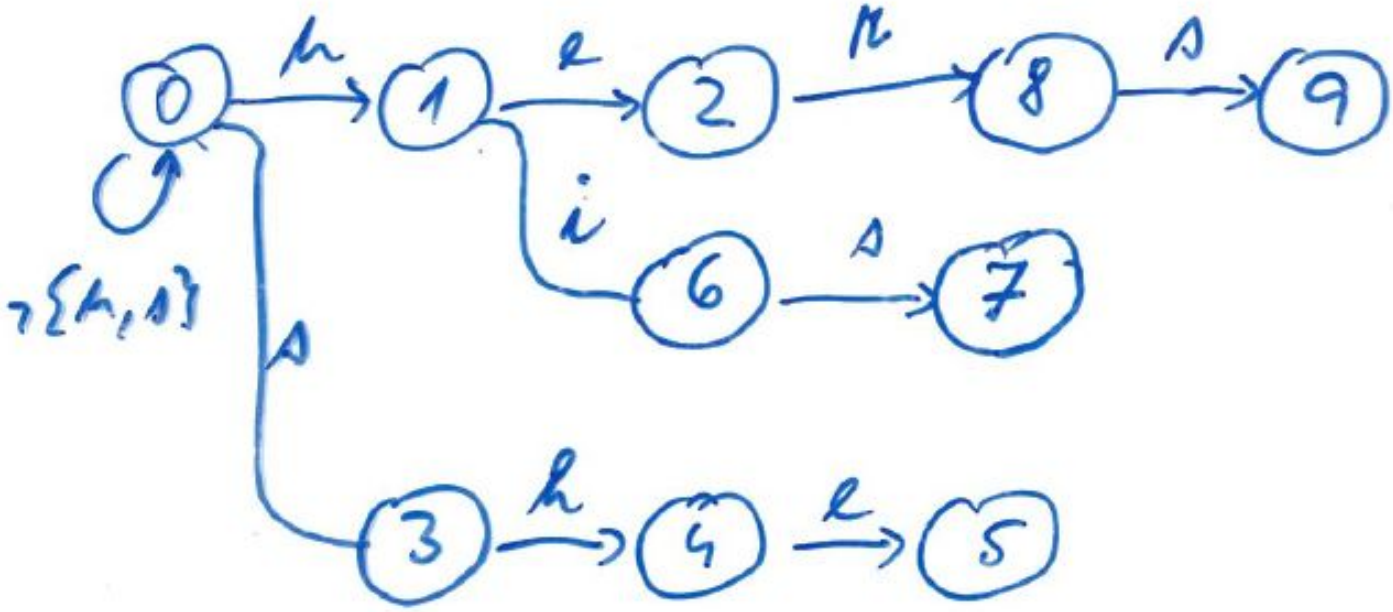
03 forall $x \in \Sigma$ do
04  if $(s := g(0,x)) \neq 0$ then //zpracuj potomky kořene
05  $f(s) := 0$; $out(s) := o(s)$
06  $queue := queue \cup \{s\}$ // na konec fronty
07 while $queue$ není prázdná do
08  $r :=$ první prvek z $queue$ (a vyřaď ho)
09  forall $x \in \Sigma$ do
10  if $(g(r,x)) \neq \perp$ then // zpracuje potomky $r$
11  $s := g(r,x)$; $t := f(r)$
12  while $g(t,x) = \perp$ do $t := f(t)$ // prohlíží přípony
13  $f(s) := g(t,x)$
14  $out(s) := o(s) \cup out(f(s))$
15  zařaď $s$ do $queue$

Tvrzení: Výstup alg. 3 vytvoří korektní vyhledávací stroj.

Složitost. Netriviální je opět počet přechodů $f$, ř. (12). Pro každý jednotlivý vzorek bude celkový počet přechodů při zpracovávání stavů reprezentovaných předponami vzorku odhadnut délkou vzorku, podobně jako při interpretaci. Celkový počet je odhadnut celkovou délkou vzorků, tj. $O(l)$, resp. pokud $|\Sigma|$ nepovažujeme za konstantu, tak $O(l \cdot |\Sigma|)$

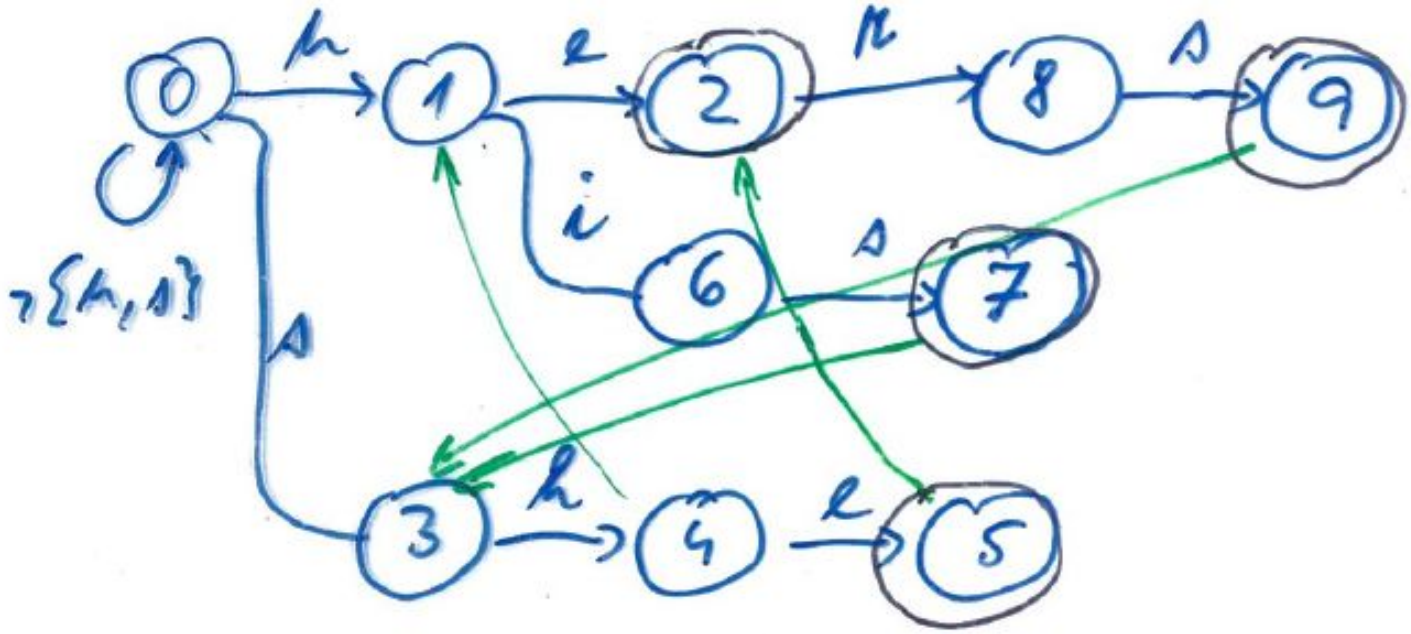
pozn:
- Implicitní (řídká) reprezentace $\perp$ (a $g(0,.) = 0$): hodnoty $\perp$ nejsou v paměti a teda je nemusíme inicializovat. Klasický konečný automat (v letním semestru) reprezentuje $g$ explicitně a používá $l \cdot |\Sigma|$ buněk. Řídká reprezentace $g$ potřebuje $O(l)$ buněk.
- strom funkce $g$ je vyhledávací datová struktura TRIE (bez optimalizací), tj. písmenkový strom

Příklad 2.
Vzorky: $K = \{he, she, his, hers\}$
Dopředná funkce $g$ (a pomocná funkce $o$), výstup alg. 2.




Obrázek 6: Dopředná funkce $g$

Stroj AC se zpětnou funkcí $f$ a výstupní *out*, výstup alg. 3.



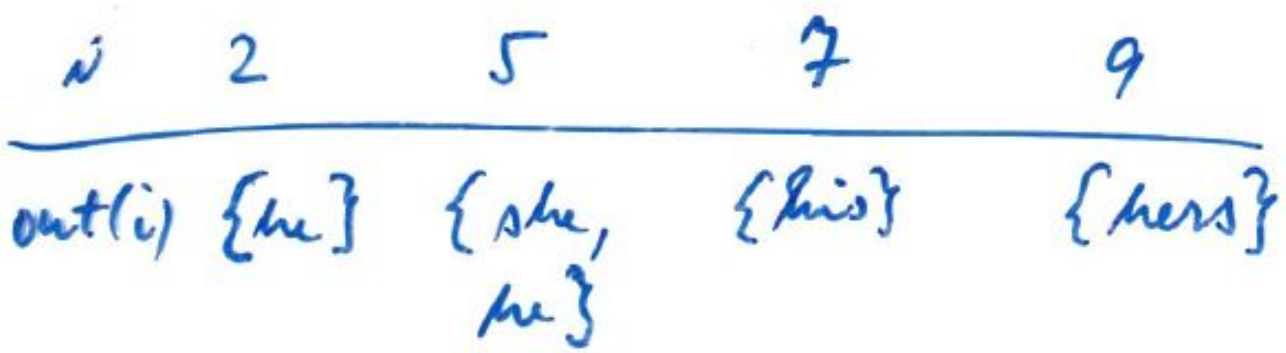

Obrázek 7: Přidána zpětná funkce $f$ a výstupní *out*

Zpětná funkce $f$ tabulkou:

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| $f(i)$ | 0 | 0 | 0 | 1 | 2 | 0 | 3 | 0 | 3 |

Obrázek 8: Zpětná funkce $f$ tabulkou

Výstupní funkce *out* tabulkou (informace není explicitně v obrázku)




Obrázek 9: Výstupní funkce *out* tabulkou

## Algoritmus Knuth-Morris-Pratt

- Vyhledávání 1 vzorku - zjednodušený alg. Aho-Corasicková

- (trochu) lepší asymp. složitost $\Theta(n + l)$ místo $\Theta(n + l \cdot |\Sigma|)$

- graf přechodové funkce $g$ není strom, ale řetězec. To umožňuje stavy reprezentovat počtem přečtených písmen (včetně 0) a používat $g$ pouze implicitně.

- zpětná funkce se zde nazývá prefixová funkce $\pi$, je *nezávislá* na zpracovávaném písmeně textu.

$\pi(s)$ je délka nejdelší vlastní přípony slova reprezentovaného stavem $s$, která je zároveň předponou vzorku.

$\pi : \{1 \dots l\} \to \{0 \dots l-1\}$

$\pi(s) = \max\{k | k < s \wedge \sigma_1 \dots \sigma_k$ je suffix $\sigma_1 \dots \sigma_s\}$

- výstupní funkce ve stavu $l$ hlásí výskyt vzorku, jinde nic

Algoritmus KMP_vyhledávání

vstup: $\sigma = \sigma_1 \dots \sigma_n$, $K = \{y\}$, prefixová funkce $\pi$

```
01 stav := 0
02 for i := 1 to n do
03    while stav > 0 ∧ y_{stav+1} ≠ σ_i
04       do stav := π(stav)
05    if y_{stav+1} = σ_i then stav:= stav+1
06    if stav = l then
07       print "našlo", i    // hlásíme pozici konce vzorku
08       stav := π(stav)
09 end
```

procedura Prefix
vstup: $K = \{y\}$ // jediný vzorek
výstup: funkce $\pi$ // prefixová funkce
01 $\pi(1) := 0$
02 $k := 0$
03 for stav := 2 to $l$ do
04 while $k > 0 \wedge y_{k+1} \neq y_{stav}$ do $k := \pi(k)$
05 if $y_{k+1} = y_{stav}$ then $k := k + 1$
06 $\pi(\text{stav}) = k$
07 return($\pi$)

**Algoritmus Rabin-Karp**

Idea: Považujeme vzor délky $l$ za $l$-ciferné číslo (znaky za cifry) v soustavě se základem $a = |\Sigma|$. Hodnoty (tj. signatury) vzoru (-ů) a stejně dlouhého úseku vstupu počítáme modulo zvolené (prvočíslo) $q \in N$. Pokud se hodnota $v$ charakteristiky vzoru a hodnota $t_i$ charakteristiky podřetězce na pozici $i$ nerovnají, vzor se na pozici $i$ určitě nenachází.

Výpočet $v$ a $t_1$ pomocí Hornerova schematu v čase $O(l)$.

$$v = ((..(\tau_1 \cdot a + \tau_2) \cdot a + ...) \cdot a + \tau_{l-1}) \cdot a + \tau_l$$

Výpočet $t_{i+1}$ z $t_i$, přesná hodnota.

$$t_{i+1} = a \cdot (t_i - a^{l-1} \cdot \sigma_i) + \sigma_{i+l}$$

Ve vzorci je $\sigma_i$ vypouštěná první cifra a $\sigma_{i+l}$ přidávaná poslední cifra.

- Pokud počítáme s přesnými čísly (bez modulo), tak jejich délka je $O(l)$ bitů (!)

Volba $q$: prvočíslo, kde $a \cdot q$ se vejde do registru
$\Rightarrow$ aritmetické operace v čase $O(1)$ místo $O(l)$.

$$t_{i+1} = (a \cdot (t_i - h \cdot \sigma_i) + \sigma_{i+l}) \bmod q$$

kde $h = a^{l-1} \bmod q$, předpočítané v čase $O(l)$
DC: Popište upravené rychlé umocňování pro výpočet $h$ v $O(\log l)$.

- Pozn. Charakteristika vzorku je vypočtena hašovací funkcí, kterou ale nepoužívame pro adresaci v tabulce.

Ale: Porovnávání hodnot modulo $q$ způsobuje *falešné hity*, když v algoritmu $v = t_i$, ale $\tau_1..\tau_l \neq \sigma_i..\sigma_{i+l-1}$

Algoritmus Rabin-Karp( $\sigma$ {text}, $\tau$ {vzor}, $a$, $q$)
01 $n :=$ délka textu $\sigma$
02 $l :=$ délka vzorku $\tau$
03 $h := a^{l-1} \bmod q$ // předvýpočet
04 $v := 0$, $t_1 := 0$
05 for $i := 1$ to $l$ do
06  $v := (a * v + \tau_i) \bmod q$
07  $t_1 := (a * t_1 + \sigma_i) \bmod q$
08 for $i := 1$ to $n - l + 1$ do
09  if $v = t_i$ then
10  if $\tau_1..\tau_l = \sigma_i..\sigma_{i+l-1}$ then
11  write "nalezeno na počáteční pozici", $i$
12  if $i < n - l + 1$ then
13  $t_{i+1} := (a * (t_i - \sigma_i * h) + \sigma_{i+l}) \bmod q$
14 end.

Analýza složitosti:
nejhorší případ: $\Theta((n-l+1)\cdot l)$
očekávaná složitost: $O(n)+O(l\cdot \#OK)+O(l\cdot \#FAL)$, kde
#OK je počet nalezených pozic (předpokládejme $O(1)$)
#FAL je počet falešných hitů: za předpokladu rovnoměrného rozložení $t_i$ je #FAL $= O(n/q)$
$\Rightarrow O(n)+O(l.(1+n/q))$

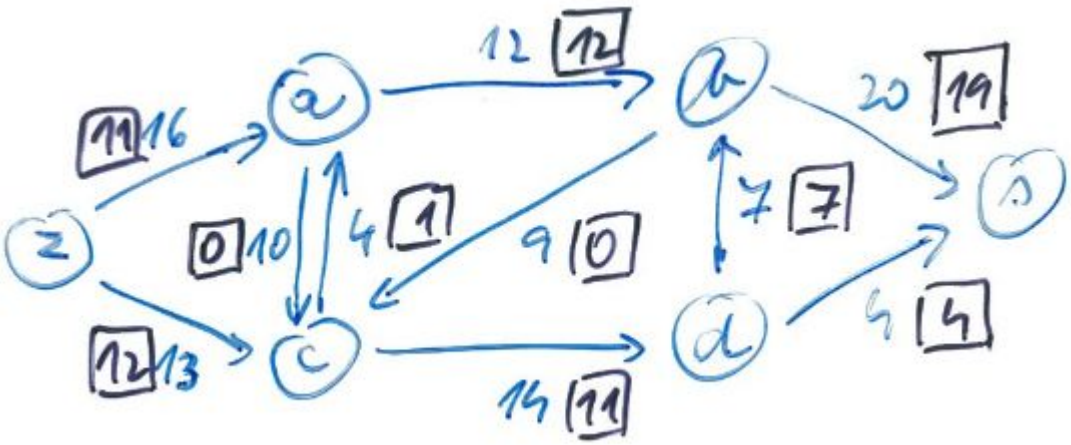
DC: upravte alg. pro víc vzorků, ...

## Toky v sítích

Def. *Síť* je $S = (G, c, z, s)$ kde
$G = (V, E)$ je orientovaný graf
$c : E \to R_0^+$ kapacity hran
$z \in V$ zdroj
$s \in V$, $s \neq z$ spotřebič (stok)
Kapacita je dodefinována i pro ostatní dvojice vrcholů: $c(u, v) = 0$, pokud $(u, v) \notin E$

Značení: $t(h) = t(u, v)$ pro $h = (u, v)$, $c(h) = c(u, v)$, $n = |V|$, $m = |E|$

Příklad sítě:




Obrázek 10: Síť: Modrou kapacity, černou tok velikosti 23

Def. *Tok* $t$ v síti $S = (G, c, z, s)$ je funkce $t : V \times V \to R$, tž.:
1) (symetrie) $t(u, v) = -t(v, u)$ pro $\forall u, v \in V$ a
2) (kapacita) $t(u, v) \leq c(u, v)$ pro $\forall u, v \in V$ a
3) (zachování toku) $\delta(t, u) = 0$ pro $\forall u \in V \setminus \{z, s\}$
kde $\delta(t, u) = \Sigma_{v \in V} t(u, v)$ je divergence funkce $t$ ve vrcholu $u$ (analogie Kirhoffova zákona pro el. proud)

Pokud $t(u,v) = c(u,v)$, tak říkáme, že hrana $(u,v)$ je saturovaná (nasycená).

Def. *Velikost* toku $t$ je $\delta(t,z)$ pro zdroj $z$, značíme $|t|$.

Problém: Nalézt v dané síti tok o maximální velikosti, tzv. maximální tok. (Značíme $t^*$, není určen jednoznačně.)

BÚNO,DC: Stačí 1 zdroj a 1 spotřebič. Idea: Zavedeme superzdroj a superspotřebič.

Def. *Řez:* v kontextu toků je řez disjunktní dvojice množin $(X,Y)$ taková, že $X \cup Y = V, z \in X, s \in Y$. Kapacita řezu $(X,Y)$ je součet kapacit hran jdoucích přes řez, tj. $c(X,Y) = \sum_{u \in X, v \in Y} c(u,v)$. Minimální řez je řez s minimální kapacitou. Tok přes řez je součet toků po hranách jdoucích přes řez, tj. $t(X,Y) = \sum_{u \in X, v \in Y} t(u,v)$.

Lemma 1. Pro každý tok $t$ a řez $(X,Y)$ platí, že tok přes řez $(X,Y)$ je roven velikosti toku, tj. $t(X,Y) = |t|$.

Dk/DC. Indukcí podle $|X|$ od $X = \{z\}$

Důsledek: Díky Lemma 1 a faktu, že $t(X,Y) \leq c(X,Y)$ pro každý řez $(X,Y)$ (důsledek kapacitního omezení) platí, že velikost maximálního toku je nejvýše rovna kapacitě minimálního řezu. Ukážeme, že platí rovnost.

*Reziduální kapacita toku* $t$ je funkce $r : V \times V \to R$ definovaná jako $r(u,v) = c(u,v) - t(u,v)$.

*Reziduální síť* $R$ k síti $S$ a toku $t$ je $R = (G_t, r, z, s)$, kde orientovaný graf $G_t = (V, E_t)$ obsahuje právě ty hrany, pro které $r(u,v) > 0$, tj. $E_t = \{\langle u,v \rangle \in V \times V | r(u,v) > 0\}$. Hodnota $r(u,v)$ je kapacita hrany v reziduálním grafu.

*Zlepšující cesta* pro $t$ je libovolná cesta $P$ ze $z$ do $s$ v $R$. *Reziduální kapacita cesty* $P$ je $r(P) = \min\{r(u,v)|(u,v) \in P\}$. Velikost toku můžeme zvýšit až o $r(P)$ zvýšením toku na všech hranách cesty $P$, přitom při zvýšení o $r(P)$ se aspoň jedna hrana cesty $P$ nasytí.

Věta (o max. toku a min. řezu). Následující podmínky jsou ekvivalentní.
1. tok $t$ je maximální tok
2. pro $t$ neexistuje zlepšující cesta
3. platí $|t| = c(X,Y)$ pro nějaký řez $(X,Y)$

Dk. 1. $\Rightarrow$ 2.: Předpokládejme, že $t$ je max. tok , ale že existuje zlepšující cesta $P$. Potom tok po zlepšení je ostře větší než $|t|$ - spor s předpokladem.

2. $\Rightarrow$ 3.: Předpokládejme, že v grafu $G$ není zlepšující cesta. Definujme $X = \{v \in V|$ existuje zlepšující cesta ze $z$ do $v\}$ a $Y = V \setminus X$. Rozdělení $(X,Y)$ je řez, protože $z \in X$ triviálně a $s \in Y$ z předpokladu neex. zlepšující cesty. Pro každé $u \in X$ a $v \in Y$ platí $t(u,v) = c(u,v)$, jinak hrana $(u,v)$ je v reziduálním grafu a $v$ patří do $X$. Podle Lemma 1 $|t| = t(X,Y) = c(X,Y)$.

3. $\Rightarrow$ 1.: Podle Důsledku platí $|t| \leq c(X,Y)$ pro všechny řezy $(X,Y)$. Podmínka $|t| = c(X,Y)$ proto implikuje, že $t$ je maximální tok.

**Metoda zlepšující cesty:** Ford a Fulkerson

1. Inicializuj tok $t$ na 0
2. while existuje zlepšující cesta $P$
3. do zlepši $t$ na hranách cesty $P$ o $r(P)$ od
4. return $t$

Vlastnosti:

1. ze znalosti maximálního toku můžeme v čase $O(m)$ zkonstruovat minimální řez (viz Věta o max. toku a min. řezu)

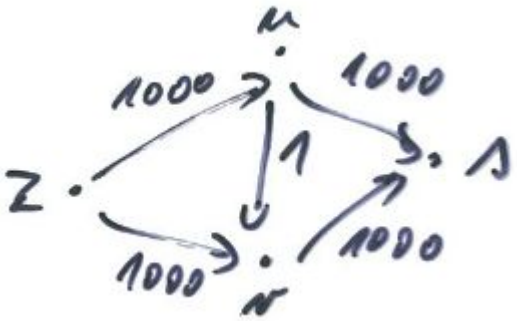
2. pokud jsou kapacity iracionální čísla, potom nemusí implementace metody zlepšující cesty skončit po konečném počtu kroků. Velikost toku konverguje, ale nemusí konvergovat k velikosti maximálního toku. (Neformálně: Použitá strategie výběru cesty není *spravedlivá*.)

3. pokud jsou kapacity racionální čísla, tak lze úlohu převést na ekvivalentní úlohu s celočíselnými kapacitami

4. pro celočíselné kapacity, každá zlepšující cesta zvýší velikost toku aspoň o 1. Proto metoda skončí nejvýše po $|t^*|$ krocích, navíc zkonstruovaný maximální tok je celočíselný (na každé hraně).

5. algoritmus je generický: zlepšující cestu lze vybírat libovolnou strategií, tj. hledat libovolným algoritmem na prohledávání orientovaných grafů (výhoda pro důkaz správnosti, nevýhoda pro odhad složitosti (a konečnost))

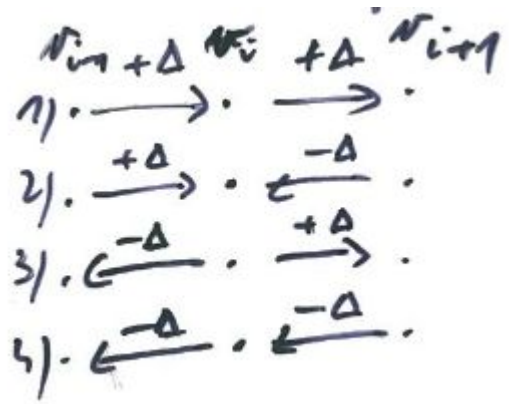
Příklad sítě s dlouhým výpočtem, obr. 11 k bodu 4. V lichých krocích volíme cestu $zuvs$, v sudých $zvus$. Kapacita cesty je vždy 1. Časová složitost algoritmu v nejhorším případě nezávisí na velikosti sítě, ale na velikosti kapacit v síti. (Problém částečně způsobuje použitá strategie výběru cesty. Cesta $zus$ není vybrána.)




Obrázek 11: Síť s dlouhým výpočtem

Tvrzení: Zkonstruovaná funkce $t$ je tok.

Dk. Indukcí podle počtu cyklů. a) Nulový tok je tok. b) Změna toku na *celé* zlepšující cestě $P$ změní divergenci jen v krajních vrcholech $z$ a $s$, ve vnitřních se nemění, viz obr. 12. Nový tok je přípustný, z volby $r(P)$.




Obrázek 12: Změny na hranách ve vnitřních vrcholech

Tvrzení: Pro celočíselné kapacity, časová složitost alg. Ford–Fulkerson je $O(|t^*|.m)$ (a alg. skončí). Pozn.: Nepolynomiální k velikosti bitového zápisu $c$.

Parciální (částečná) správnost alg. Pokud alg. skončí, nenašla se zlepšující cesta. Podle Věty je potom zkonstruovaný tok maximální. (+ konečnost $\Rightarrow$ úplná správnost)

Ideální verze zlepšující cesty: Vždy existuje posloupnost nejvýše $m$ zlepšujících cest, pomocí kterých lze skonstruovat max. tok.

Strategie volby cesty:

• Maximální zlepšení (Edmonds a Karp): Najdi zlepšující cestu s maximální reziduální kapacitou. (Použijeme upravený Dijkstrův alg.; DC) ; Konkrétní složitost nerozebíráme, další alg. jsou lepší.

• Implementace s nejkratším zlepšením (Edmonds a Karp): Najdi (a vyber) nejkratší zlepšující cestu, tj. cestu s minimálním počtem hran. (Použijeme prohledávání do šířky.)

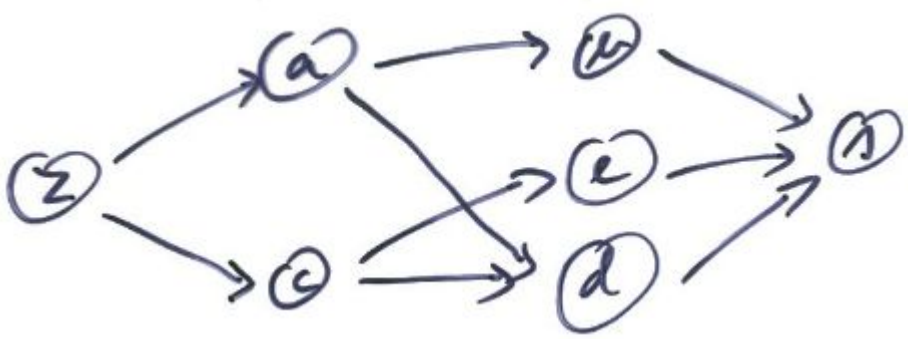
Věta: Počet zlepšujících kroků při implementaci s nejkratším zlepšením je $O(nm)$. Celkově metoda potřebuje čas $O(nm^2)$.

Idea rozboru: Při výběru podle délky zlepšující cesty se délky cest postupně prodlužují (dokážeme za chvíli u Dinicova alg.). Proto máme až $n$ fází podle délky cesty a v každé fázi potřebujeme nasytit až $m$ hran. Celkem $O(nm)$ hledání nejkratší zlepšující cesty. Jedno hledání prohledáváním do šířky v $O(n+m)$, odtud celkový čas $O(nm^2)$. (Pro husté grafy dostáváme $O(n^5)$.)

Další zlepšení dostaneme, pokud místo zlepšování toku samostatně po jedné cestě použijeme všechny nejkratší cesty "najednou". To využívá následující Dinicův alg. se složitostí $O(n^2m)$ (a $O(n^4)$), případně po další optimalizaci $O(n^3)$.

Def. Síť $S$ je *(pročištěná) vrstevnatá*, pokud existuje rozklad množiny jejich vrcholů na $X_0, X_1, ..., X_k$ tak, že
- $X_0 = \{z\}$
- $X_k = \{s\}$
- $(u, v) \in E(G), u \in X_i, v \in X_j \Rightarrow j = i + 1$
- z každého vrcholu kromě spotřebiče hrana vychází a do každého vrcholu kromě zdroje hrana vchází.

Pozorování: Ve vrstevnaté síti platí $\forall v : d(z, v) + d(v, s) = d(z, s)$ a $\forall (u, v) \in E(G) : d(z, u) + d(v, s) + 1 = d(z, s)$, tj. každý vrchol a hrana leží na nejkratší cestě ze $z$ do $s$. Síť splňující tuto podmínku nazveme *pročištěnou*.




Obrázek 13: Vrstevnatá (pročištěná) síť

Def. Hrana $(u, v)$ v síti $S$ s tokem $t$ je *nasycená*, pokud $t(u, v) = c(u, v)$. Tok v síti $S$ je *blokující*, pokud každá orientovaná cesta v $S$ ze zdroje do spotřebiče obsahuje nasycenou hranu.

Pozn. Ve vrstevnaté síti nebudeme hledat maximální tok, ale pouze blokující, který zablokuje všechny cesty nejkratší délky.

Značení: $d(u, v)$ je délka nejkratší orientované cesty z $u$ do $v$, měřená počtem hran.

Algoritmus Dinic(G,z,s)

1. forall $h \in E(G)$ do $t(h) := 0; r(h) := c(h)$ od
2. while $\exists$ cesta $P$ ze $z$ do $s$ v reziduální síti $S_t$
3. do {síť $S'_t$ vytvoříme z $S_t$ vynecháním všech hran, které neleží na (nějaké) nejkratší orientované cestě ze $z$ do $s$ }
4. $S'_t := S_t$
5. forall $v \in V$ do urči $d(z, v)$ // BFS ze $z$
6. forall $v \in V$ do urči $d(v, s)$ // BFS protisměrně z $s$
7. forall $(u, v) \in E(S'_t)$ do
8. if $d(z, u) + 1 + d(v, s) > d(z, s)$
9. then vynechej $(u, v)$ z $S'_t$ // zbylé hrany patří nějaké min. cestě
10. $q :=$ blokující tok v $S'_t$
11. forall $(u, v) \in E(S'_t)$ do $t(u, v) := t(u, v) + q(u, v)$ od
12. od

Lemma. Nech $S_{t_1}$ a $S_{t_2}$ jsou reziduální sítě vytvořené dvěma po sobě následujícími iteracemi Dinicova algoritmu. Potom $d_{S_{t_1}}(z, s) \leq d_{S_{t_2}}(z, s) - 1$.

Dk. Pro cesty obsahující hrany $S_{t_1}$ v $S_{t_2}$ platí, protože $q$ je blokující.

Pro cesty obsahující novou hranu přidanou do $S_{t_2}$: nová hrana $(v_i, v_{i-1})$ vzniká kvůli úpravě toku na hraně $(v_{i-1}, v_i)$ na nějaké minimální cestě $v_0, v_1 ... v_k$. Délka cesty s novou hranou je $d_{S_{t_1}}(z, v_i) + 1 + d_{S_{t_1}}(v_{i-1}, s) = i + 1 + (k - i + 1) = k + 2$. Přitom $d_{S_{t_1}}(z, v_i) \leq d_{S_{t_2}}(z, v_i)$ a $d_{S_{t_1}}(v_{i-1}, s) \leq d_{S_{t_2}}(v_{i-1}, s)$.

Důsledek: Časová složitost Dinicova algoritmu na síti $S$ s $n$ vrcholy a $m$ hranami je $O(n.f(n,m))$, kde $f(n,m)$ je čas potřebný k nalezení blokujícího toku ve vrstevnaté síti.

Dk. Maximální délka cesty je $n$, tj. počet konstruovaných vrstevnatých sítí je nejvýše $n$ a časová složitost zpracování jedné sítě je $f(n,m)$.

Určení blokujícího toku ve vrstevnaté síti
Vstup: vrstevnatá síť $S = (G, c, z, s)$
Výstup: blokující tok $q$ v síti $S$
1. forall $(u,v) \in E(G)$
2. do $q(u,v) := 0$ od
3. while $\exists$ cesta $P$ ze zdroje $z$ do $s$ v $S$
4. do $c(P) := min\{c(u,v) | (u,v) \in P\}$
5. forall $(u,v) \in P$ do $q(u,v) := q(u,v) + c(P)$ od
6. $S := S - \{$všechny nasycené hrany cesty P$\}$
7. pročistiSíť$(S)$ // uzavírání hran a vrcholů
8. od

Složitost
1) nalezení a update cesty trvá $O(n)$
výběr *libovolné* hrany při prodlužování cesty v pročistěné síti (nepotřebuju DFS a backtracking)
2) každý průchod cyklem (3) nasytí (a uzavře) aspoň 1 hranu $\Rightarrow O(m)$ průchodů
$\Rightarrow$ složitost nalezení blokujícího toku ve vrstevnaté síti je $O(n.m)$
$\Rightarrow$ složitost nalezení max. toku Dinicovým alg. je $O(n^2.m)$

Pozn/DC. Strategie: Výběr vrcholu s nejmenším přítokem a propagace změn od něho po vrstvách oběma směry: složitost

jedné fáze $O(n^2)$, celková složitost $O(n^3)$

Rychlá implementace PročistiSíť$(S)$:

Idea: kaskádové uzavírání hran a vrcholů neležících na cestě $P$ ze $z$ do $s$, tž. $c(P) > 0$.

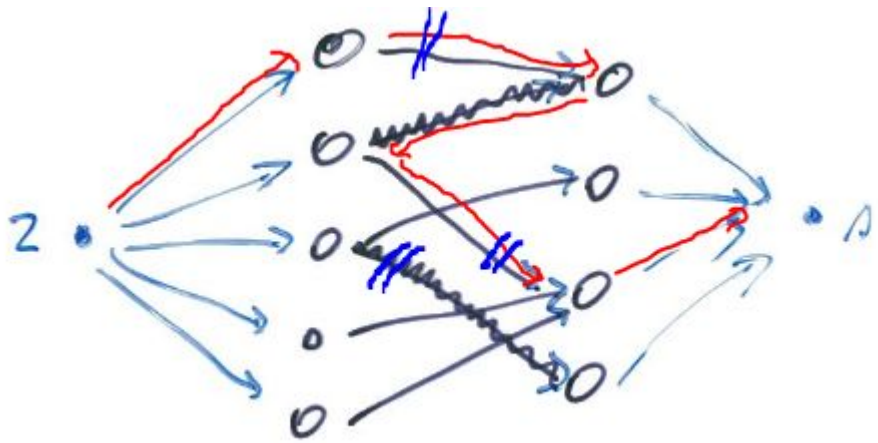
Na každém vrcholu a hraně chceme strávit v průběhu celého alg. pro 1 vrstevnatou síť pouze konstantní čas. Pamatujeme si vstupní $d_{in}(v)$ a výstupní $d_{out}(v)$ stupně vrcholů pro neuzavřené vrcholy. Hranu uzavírám, pokud se nasytí, a snížím stupně incidentních vrcholů. Vrchol uzavírám, pokud $d_{out}(v) = 0$ (nebo $d_{in}(v) = 0$) a dekrementuju stupeň sousedů (a započítám to hraně). Vrchol i hrana se (otvírá a) uzavírá $1\times$, v konstantním čase.

**Aplikace:** Určení maximálního párování v bipartitním grafu.

Def. *Párování* je množina hran $M \subseteq E(G)$ grafu $G$, tž. $\forall e_1, e_2 \in M : e_1 \cap e_2 = \emptyset$.

Maximální párování je párování s maximálním počtem hran.

Síť pro maximální párování:




Obrázek 14: Maximální párování velikosti 3 dvojitou modrou, zlepšující cesta červeně. Kapacity jsou 1

Hrany orientujeme z jedné partity do druhé. Zdroj spojíme s

každým vrcholem jedné partity, stok s každým vrcholem druhé partity. Kapacita všech hran je 1.

Maximální tok je nejvýš $n = |V|$, proto složitost Ford-Fulkersonova alg. je $O(nm)$

**Goldbergův algoritmus** (preflow-push)

Def. *Předtok* (preflow, vlna) je funkce $t : V \times V \to R$, splňující podmínku symetrie a kapacity na každé hraně, ale pro každý vrchol kromě zdroje dovolíme přebytek $excess : V \to R$.
$\forall v \in V - \{z\} : excess(v) \geq 0$, kde $excess(v) = \sum_{w \in V} t(w, v)$
Vrchol $v$ (kromě $z$ a $s$) s kladným $excess(v)$ se nazývá aktivní.

Def. *Výšková funkce*. Nech $t$ je předtok a $R$ reziduální graf pro $t$. Potom funkce $h : V \to N$ je výšková funkce pro $t$, pokud
1) $h(z) = |V|$
2) $h(s) = 0$
3) $\forall (u, v) \in E_R : h(u) \leq h(v) + 1$
Pokud v 3) platí rovnost hrana $(u, v)$ je přípustná.

Idea GA: konstruujeme předtok (ne tok po celých cestách), přesouváme přebytky vrcholu po hranách s rezervou (,kde hrana směřuje "dolů"a rozdíl výšek vrcholů je právě 1).

Goldbergův algoritmus (generický).

01 $h(z) := n$ $\quad // = |V|$
02 $h(v) := 0$ pro $v \in V - \{z\}$
03 tok hran ze zdroje je rovný kapacitě hrany
04 tok ostatních hran je 0
05 while $\exists$ vrchol s kladným přebytkem, kromě $s$ do
06 $\quad v :=$ vrchol s kladným přebytkem, $v \neq s$
07 $\quad$ if $\exists h = (v, w)$ s kladnou rezervou & $h(v) > h(w)$
08 $\quad$ then
09 $\quad\quad (v, w) :=$ hrana (z $v$) s kladnou rezervou
10 $\quad\quad\quad$ a splňující $h(v) > h(w)$
11 $\quad\quad \delta := min(excess(v), r(v, w))$
12 $\quad\quad$ převedeme přebytek z $v$ do $w$ velikosti $\delta$
13 $\quad$ else $h(v) := h(v) + 1$
14 $\quad$ endif
15 end.

Tři způsoby vykonání těla (záleží na pořadí):

1) nasycené převedení přebytku: $\delta$ je rovno rezervě hrany $(v, w) \Rightarrow$ nuluje rezervu hrany

2) nenasycené převedení přebytku: $\delta$ je menší než $r(v, w) \Rightarrow$ nuluje přebytek vrcholu $v$

3) zvýšení vrcholu $v$: příkaz $h(v) := h(v) + 1$

Pozn. Vrcholy mohou vystoupat nad $n = h(z)$, tj. výšku zdroje, a tak vrátit přebytek do zdroje.

Lemma 1. Po inicializaci vždycky platí, že neexistuje hrana $(v, w)$ taková, že $h(v) > h(w) + 1$, a rezerva hrany je nenulová.

Dk. Po inicializaci podmínka platí, protože každá hrana $(v, w)$ splňující $h(v) > h(w)$ vede ze zdroje a má nulovou rezervu (tj. je nasycená).

Vykonání těla while hranu s nenulovou rezervou porušující podmínku nevytvoří, protože

a) po zvednutí $v$: $v$ má přebytek (vybráním), hrana má nenulou rezervu (předpoklad), potom $v$ nezdviháme (spor), ale přesouváme přebytek. Předpoklad neplatí, hrany z $v$ mají nulovou rezervu.

b) po převedení přebytku opačnou hranou $(w, v)$, kde $r(v, w) = 0$, platí $h(w) > h(v)$.

Věta 1. Parciální správnost. Pokud Goldbergův alg. skončí, najde největší tok.

Dk. Pokud while cyklus skončí, potom $\forall v \in V, v \neq s$ má nulový přebytek, proto je zkonstruovaný předtok taky tok. Ešte chceme: tok je maximální $\Leftarrow$ neexistuje zlepšující cesta $\Leftarrow \forall$ cesta má nasycenou hranu s nulovou rezervou.

Mějme cestu ze zdroje do spotřebiče. Cesta začína ve výšce $n = h(z)$ a končí ve výšce $0 = h(s)$ a má nejvýše $n - 1$ hran. Proto existuje hrana klesající aspoň o 2. Podle Lemmy 1 má tato hrana nulovou rezervu.

Pozn. Strategie. Algoritmus umožňuje výběr vrcholu a hrany.
...
Strategie je parametr generického algoritmu. Matematicky: funkce: $(historie\times)stav \to V$, implementačne: podmínka nebo max/min, víc kritérií, random.
Čas výpočtu strategie započítáváme: Chci rychlou funkci výběru: lze udržovat datovou strukturu.

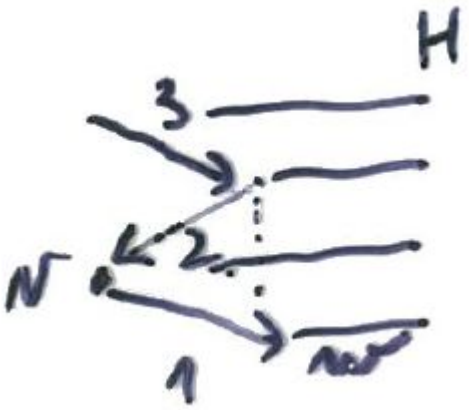
Postup pro důkaz složitosti: odhadneme postupně max. výšku vrcholu a teda počet zvednutí, počet nasycených a počet nenasycených převedení.

Lemma 2. Vždy od ukončení inicializace, pokud má vrchol $v$ kladný přebytek, potom z něho vede orientovaná cesta do zdroje, na které mají všechny hrany kladnou rezervu.
Dk. Nech má $v$ kladný přebytek. Označme $A$ množinu vrcholů, do kterých vede z $v$ orientovaná cesta skládající se z hran s kladnou rezervou.
Uvažujme hranu $(x, y), x \notin A, y \in A$. Pokud tok hranou $(x, y)$ je kladný, pak opačná hrana $(y, x)$ má kladnou rezervu a $x \in A$ kvůli cestě $v...y, x$; spor
$\Rightarrow$ tok po hranách z vrcholů mimo $A$ do $A$ je nulový
$\Rightarrow \sum_{v\in A} excess(v) \leq 0$. (1)
Víme: $v \in A$, přebytek $v$ je kladný
$\Rightarrow$ v $A$ je vrchol se záporným přebytkem, protože podle (1) je celkový přebytek $A$ záporný
$\Rightarrow$ zdroj je v $A$: $z$ je jediný vrchol se záporným přebytkem
$\Rightarrow$ z $v$ do zdroje vede cesta z hran s kladnou rezervou, z definice $A$. QED

Lemma 3. Výška vrcholu není nikdy větší než $2n$.
Dk. Sporem. Předpokládejme, že zdviháme vrchol nad $2n$. Potom je ve výšce $2n$ a má kladný přebytek. Podle lemmy 2 existuje cesta z $v$ do $z$ skládající se z hran s kladnou rezervou. Cesta obsahuje nejvýše $n-1$ hran, překonáva spád (aspoň) $n$, proto obsahuje hranu $(x, y)$, kde $h(x) > h(y)+1$ a podle lemmy 1 má tato hrana nulovou rezervu. Spor.

Lemma 4. Počet zvednutí v průběhu celého algoritmu je nejvíc $2n^2$
Dk. Máme $n$ vrcholů, každý zdviháme maximálně $2n$-krát.

Lemma 5. Počet nasycených převedení přebytku je za celý výpočet nejvíc $n.m$.
Dk. Nech $h = (v, w)$ je hrana. Součet výšek $v$ a $w$ je medzi 0 a $4n$. Pokud dojde k nasycenému převedení přebytku, potom rezerva hrany $(v, w)$ klesne na 0 a platí $h(v) = h(w) + 1$. (Stav 1)
Abychom mohli znova nasyceně převádět, musí se zvýšit rezerva na nenulovou hodnotu, obr. 15. To nastáva pouze v případě převádění přebytku opačnou hranou $(w, v)$. Proto $h(w)$ musíme zvýšit aspoň o 2 (Stav 2, dojde k převedení opačnou hranou) a následně zvýšit $h(v)$ aspoň o 2. (Stav 3) Mezi dvěma nasycenými převedeními hranou $h = (v, w)$ se zvýší $h(v) + h(w)$ aspoň o 4. Proto počet nasycených převedení hranou $h$ je nejvíc $n$ a celkově je $\# \leq n.m$. QED




Obrázek 15: Zvyšování při opakovaném nasyceném převedení na hraně (v,w)

Lemma 6. Počet nenasycených převedení za celý výpočet je nejvíc $2n^2 + 2n^2.m$.
Dk. (pomocí potenciálu) Označme $S$ součet výšek vrcholů, které mají kladný přebytek, kromě $z$ a $s$. Po inicializaci je $S = 0$, protože jedině zdroj má nenulovou výšku. Na konci výpočtu je $S = 0$, protože "vnitřní"vrcholy nemají přebytek.
Zvýšení vrcholu zvýší $S$ o 1. Nasycené převedení přebytku hranou $(u, v)$ zvýší $S$ nejvíc o $h(v) \leq 2n$ (pokud $v$ přebytek neměl a $u$ přebytek zůstane).
Celkové zvýšení $S$ je nejvíc $2n^2 + 2n.nm$. (1)
Nenasycené převedení přebytku hranou $(u, v)$ sníží $S$ aspoň o 1, protože výšky se nemění, sčítanec $h(u)$ vypadne a sčítanec $h(v)$ se případně přidá (pokud nebyl v $S$). Ale $h(u) = h(v)+1$, proto se $S$ sníží.
Celkový počet nenasycených převedení je $2n^2 + 2n^2m$ podle (1). QED

DC: Ve kterých případech zvýšení a snížení potenciálu nedosahuje krajních hodnot a jaká heuristika z toho plyne?

Věta 2. Časová složitost Goldbergova algoritmu je $O(n^2m)$.
Dk. Z lemmat 4, 5, 6.

Strategie výběru vrcholu: vždy nejvyšší vrchol s přebytkem $\Rightarrow$ # nenasycených převedení je $\leq 8n^2\sqrt{m}$

Na tomto základě je založen nejlepší známý alg. Goldberg, Tarjan 1996: $O(nm \log(n^2/m))$.

**Třídící sítě**

Třídící síť je obvod, který má $n$ vstupů s hodnotami z nějakého lineárně uspořádaného typu (tj. každé dvě hodnoty jsou porovnatelné) a $n$ výstupů, na kterých jsou vstupní hodnoty setříděné.

Tento obvod obsahuje jediný typ hradla - komparátor. To je hradlo se dvěma vstupy $x_1$ a $x_2$ a dvěma výstupy $y_1$ a $y_2$, pro které platí $y_1 = min(x_1, x_2)$ a $y_2 = max(x_1, x_2)$.

Formální definice sítě:
- $K = \{K_1, K_2, ..., K_s\}$ množina komparátorů, $s$ je velikost sítě
- $O = \{(k, i), 1 \leq k \leq s, 1 \leq i \leq 2\}$ je množina výstupů hradel ($k$ je číslo komparátoru a $i$ číslo výstupu)
- $I = \{(k, i), 1 \leq k \leq s, 1 \leq i \leq 2\}$ je množina vstupů
- $C = (K, f)$ je třídící síť, kde $f : O \to I$ je částečné prosté zobrazení (výstup lze použít opakovaně)

Podmínka acyklicity sítě:
Požadujeme, aby orientovaný graf $G = (K, E)$, kde $(K_u, K_v) \in E$, pokud existují $i$ a $j$ takové, že $f(u, i) = (v, j)$, byl acyklický.

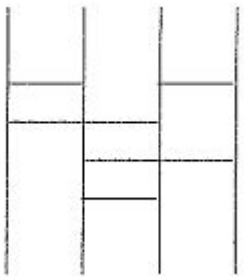
Rozdělení komparátorů do hladin:
Definujme $L_1 = \{K_i, K_i$ má v $G$ vstupní stupeň $0\}$ ($L_1$ je neprázdná díky acyklicitě)
Nech jsou definovány $L_1..L_n$ a $L = \bigcup_{i=1}^{n} L_i \subset K, L \neq K$. Potom definujeme $L_{n+1} = \{K_i, K_i$ má v $G \setminus L$ vstupní stupeň $0\}$.

Počet hladin se nazývá hloubka sítě a značí $d$.
Pozn. Síť spočítá své výstupy v $d$ (paralelních) krocích.

Jiná reprezentace sítě: (jen pro transpoziční sítě)
- "dráty"vedou ze vstupu $x_i$ do výstupu $y_i$ a jsou nakresleny jako přímky
- jednotlivé komparátory spojují příslušné (dva) "dráty"(vodiče)
- každá třídící síť jde takto překreslit
- počtu vstupů/výstupů (tj. #drátů) říkáme šířka sítě

Transpoziční síť $K$ můžeme zapsat jako množinu hradel. Popis hradla je $(j, p_1, p_2), 1 \leq j \leq d, 1 \leq p_1 < p_2 \leq m$, kde $d$ je hloubka a $m$ šířka sítě $K$.

Př: $S_4 = \{(1,1,2),(1,3,4),(2,1,3),(2,2,4),(3,2,3)\}$



Obrázek 16: Třídící síť šířky 4 (zkonstruovaná použitou metodou sudá–lichá). Data tečou shora dolů.

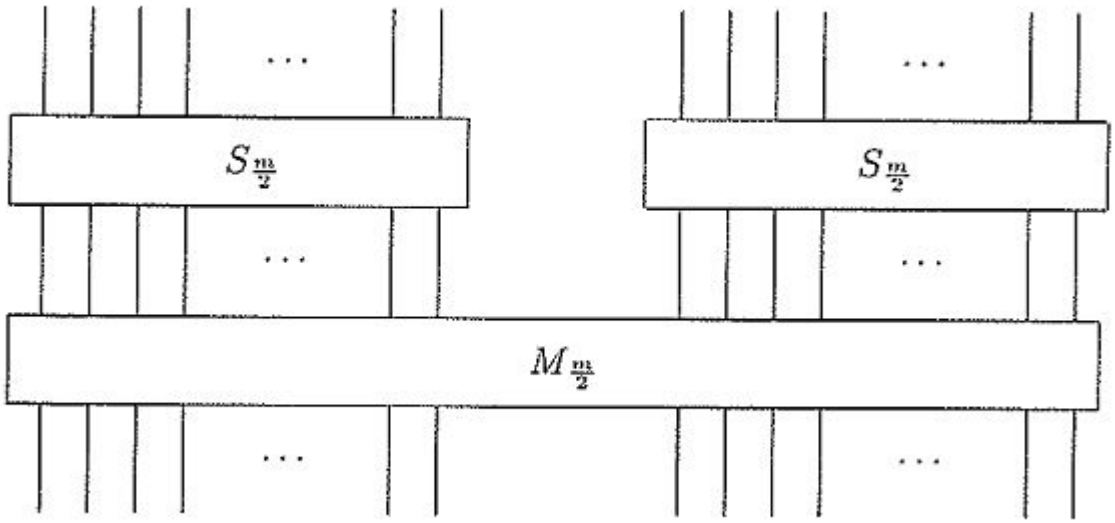
Souvislosti:
- Hradlové sítě jako hardwarová reprezentace algoritmů.
- Struktura sítě je daná: výpočty se liší v datech (ne v řízení).

Mergesort implementovaný třídící sítí.

Chceme setřídit $x_1..x_n$, kde $n = 2^l$.

Realizujeme to třídící sítí $S_n$, která je rekurzivně definována pomocí dvou třídících sítí $S_{n/2}$ a slučovací (slévací) sítě $M_{n/2}$ šířky $n$. Rekurze končí pro $n = 2$.
$S_1$ je prázdná síť.

$$
\begin{aligned}
S_n &= S_{n/2} \\
&\cup \{(j, \tfrac{n}{2} + p_1, \tfrac{n}{2} + p_2) | (j, p_1, p_2) \in S_{n/2}\} \\
&\cup \{(k + j, p_1, p_2) | (j, p_1, p_2) \in M_{n/2}\}, \text{ pro } k = d(S_{n/2})
\end{aligned}
$$

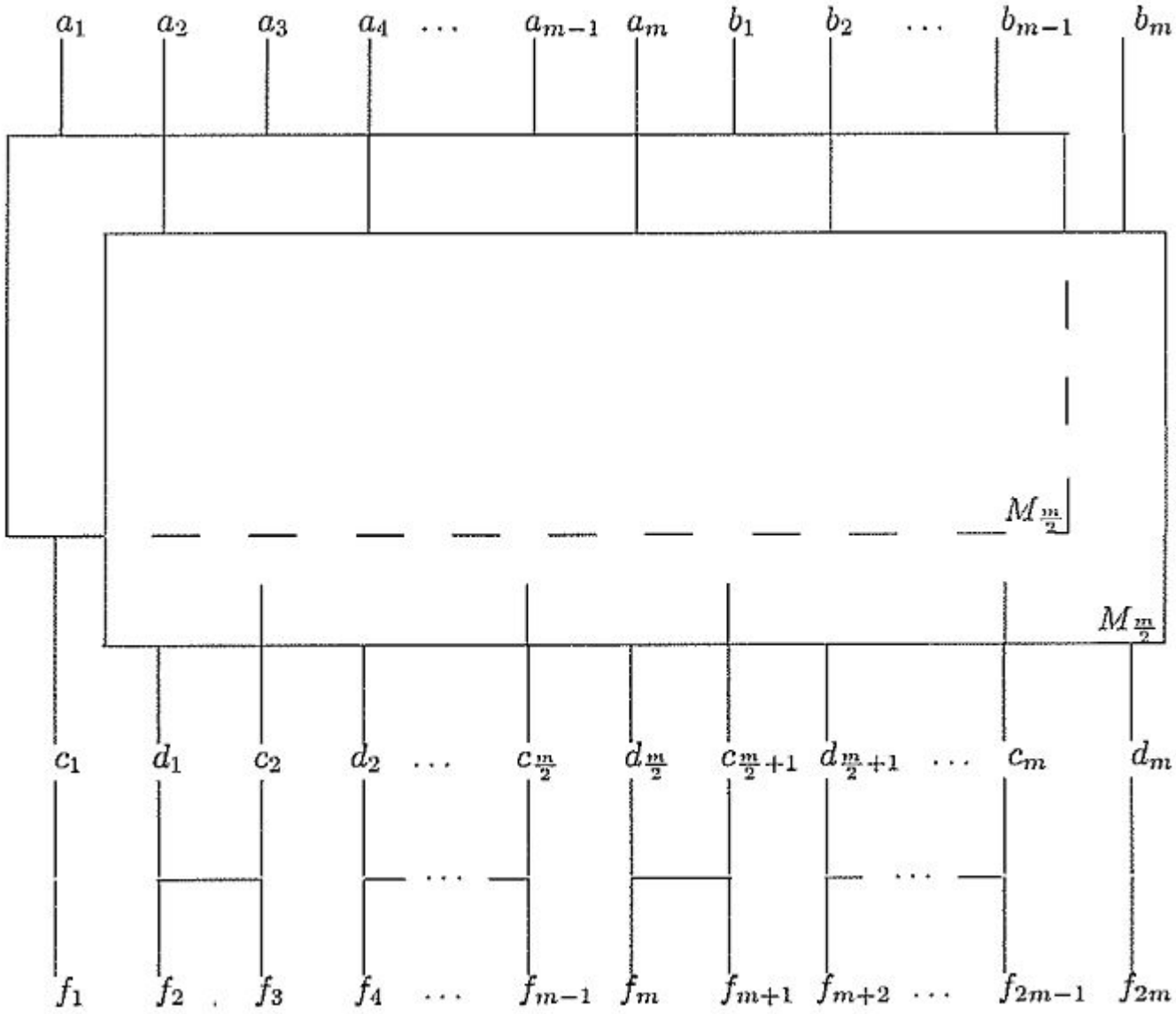



Obrázek 17: Třídící síť

Slučovací síť $M_n$ slučuje dvě uspořádané posloupnosti poloviční délky do jedné uspořádané posloupnosti. Konstruuje se také rekurzivně pro $n > 1$, rekurze končí pro $n = 1$.
$M_1$ je 1 hradlo: $\{(1, 1, 2)\}$.

$$
\begin{aligned}
M_n &= \{(j, 2p_1 - 1, 2p_2 - 1) | (j, p_1, p_2) \in M_{n/2}\} \\
&\cup \{(j, 2p_1, 2p_2) | (j, p_1, p_2) \in M_{n/2}\} \\
&\cup \{(k + 1, 2p, 2p + 1) | 1 \leq p \leq n - 1\}, \text{ pro } k = d(M_{n/2})
\end{aligned}
$$

Liché členy obou setříděných posloupností jsou vstupem jedné kopie $M_{n/2}$ s výstupy $c_i$ a sudé členy jsou vstupem druhé kopie $M_{n/2}$ s výstupy $d_i$. Na konci jsou výstupy obou sítí propojeny jednou hladinou komparátorů, $(d_i =)y_{2i}$ s $y_{2i+1}(= c_{i+1})$




Obrázek 18: Slučovací síť

Pro vstup platí: $a_1 \leq a_2 \leq ... \leq a_n$ a $b_1 \leq b_2 \leq ... \leq b_n$
Ind. předp.: $c_1 \leq c_2 \leq ... \leq c_n$ a $d_1 \leq d_2 \leq ... \leq d_n$
Dokážeme, že: $y_1 \leq y_2 \leq ... \leq y_{2n}$

Stačí dokázat pro neporovnané výstupy: $c_i \leq d_i$ a $d_i \leq c_{i+2}$ (DC), ostatní dva případy ($c_i \leq c_{i+1}$ a $d_i \leq d_{i+1}$) plynou z ind. předp.

Ukážeme $c_p \leq d_p$ pro lib. $p$, $1 \leq p \leq n$.
Nech $[c_1..c_p] = [a_1..a_{2i-1}, b_1..b_{2(p-i)-1}]$, (multimnožiny)

proto $c_p \in \{a_{2i-1}, b_{2(p-i)-1}\}$.
Podobne $[d_1..d_p] = [a_2..a_{2j}, b_2..b_{2(p-j)}]$,
proto $d_p \in \{a_{2j}, b_{2(p-j)}\}$.

Rozeberme $c_p = a_{2i-1}$.
(1) $i \leq j$: $\quad c_p = a_{2i-1} \leq a_{2i} \leq a_{2j} \leq d_p$.
(2) $i > j$ : $\quad p - i < p - j \Rightarrow p - i + 1 \leq p - j$
$b_{2(p-i)+1}$ není v (1) $\Rightarrow b_{2(p-i)+1} \geq c_p$
$c_p = a_{2i-1} \leq b_{2(p-i)+1} \leq b_{2(p-i+1)} \leq b_{2(p-j)} \leq d_p$.
V rozebraném případě tvrzení platí, ostatní případy analogicky.

Hloubka a velikost třídící sítě v závislosti na šířce $n = 2^l$.

Slučovací síť $M_n$ šířky $2n$:
hloubka z konstrukce: $d(M_n) = d(M_{n/2}) + 1, d(M_1) = 1$
hloubka explicitně: $d(M_n) = \log_2 n + 1$
velikost z konstrukce: $s(M_n) = 2s(M_{n/2}) + (n-1)$, $s(M_1) = 1$
velikost explicitně: $s(M_n) = n \log_2(n) + 1$

Třídící síť $S_n$ šířky $n$:
hloubka z konstrukce: $d(S_n) = d(S_{n/2}) + d(M_{n/2}), d(S_1) = 0$
hloubka explicitně: $d(S_n) = \frac{1}{2} \log_2 n(\log_2 n + 1)$
velikost z konstrukce: $s(S_n) = 2s(S_{n/2}) + s(M_{n/2})$
velikost explicitně: $s(S_n) = \frac{n}{4} \log_2 n(\log_2 n - 1) + (n-1)$

Dolní odhad složitosti třídění pomocí transpozičních sítí.
Transpoziční síť je síť složená pouze z komparátorů (libovolně umístěných)
$\Rightarrow$ Třídící síť je speciální případ transpoziční sítě.

Lemma 1: Každá transpoziční síť dává pro vstup $x_1, ..., x_n$ na výstupu nějakou permutaci $\pi$ vstupních hodnot, tj. na výstupu je $x_{\pi(1)}, ..x_{\pi(n)}$.

Dk: Zřejmý: indukcí podle počtu komparátorů. Komparátor prohodí nebo neprohodí své vstupy.

Df: Nech $C$ je transpoziční síť šířky $n$. Permutace $\pi : \{1, 2, .., n\} \to \{1, 2, .., n\}$ je dosažitelná pro $C$, pokud existuje vstupní posloupnost $x_1..x_n$ taková, že $C$ vydá $x_{\pi(1)}..x_{\pi(n)}$.

Lemma 2: Nech $C$ je třídící síť, potom pro $C$ je dosažitelných všech $n!$ permutací.

Dk: Zřejmý, jsme schopni předložit na vstup inverzní permutaci utříděné posloupnosti a *korektní* síť ji musí utřídit.

Lemma 3: Nech $C$ je transpoziční síť šířky $n$ a nech $p$ je počet dosažitelných permutací pro $C$. Potom $p \leq 2^{s(C)}$.

Důsledek. Pro třídící síť $C$ platí $n! \leq 2^{s(C)}$ a proto $C$ má velikost $s(C) \in \Omega(n \log n)$ a hloubku $d(C) \in \Omega(\log n)$.

**Aritmetické obvody / sítě**

Implementace aritmetických operací pomocí boolovských hradel ($And$, $Or$, $Not$, $Xor$, $Nand$, $Nor$).

Zde: Sčítání

(jednobitová) sčítačka: vstup $x, y, z$; výstup $s, c$ (suma, carry - tj. přenos)

$s = x \oplus y \oplus z$ (2x $Xor$)

$c = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) = majority(x, y, z)$

Úloha: sečíst dvě čísla $u = \sum_{i=0}^{n-1} u_i 2^i$ a $v = \sum_{i=0}^{n-1} v_i 2^i$, kde $u_i, v_i \in \{0, 1\}$, do výsledku $s = u + v = \sum_{i=0}^{n} s_i 2^i$, $s_i \in \{0, 1\}$

1. řešení: sčítání s přenosem:

$s_i = u_i \oplus v_i \oplus c_{i-1}$ pro $i = 0..n-1$

$s_n = c_{n-1}$

kde $c_{-1}, c_0, ...c_{n-1}$ jsou definovány

$c_{-1} = 0$

$c_i = majority(u_i, v_i, c_{i-1})$

Hloubka obvodu je $\Theta(n)$, t.j. paralelní čas

velikost obvodu $\Theta(n)$

Cíl: zlepšit hloubku obvodu: Carry-lookahead alg.

idea: vytvoříme stromovou strukturu místo lineární

! problém: nemáme včas vstupní data, konkrétně carry

řešení: (! programátorský i teoretický trik)

budeme počítat místo hodnot s funkcemi

*možný pohled: podmíněný/odložený výpočet, (implicitní) rozbor případů, např. ve funkcionálním programování*

Funkce pro počítání přenosu má 3 možné výstupy

1. generuje $c_i$ pokud $u_i \wedge v_i = 1$
2. přenáší $c_{i-1}$ pokud $u_i \oplus v_i = 1$
3. shazuje $c_i$ pokud $u_i = 0 \wedge v_i = 0$

DC: skládání funkcí pomocí tabulky $3 \times 3$

Formálně zavedeme pomocné proměnné

$g_i = u_i \wedge v_i$ // generuje

$p_i = u_i \oplus v_i$ // přenáší

Vyjádříme $c_i$ a $s_i$ pomocí $g_i$ a $p_i$:

$c_i = g_i \vee (p_i \wedge c_{i-1})$ pro $i = 0..n-1$

$s_0 = p_0$, $s_i = p_i \oplus c_{i-1}$ pro $i = 1..n-1$, $s_n = c_{n-1}$

Zavedeme operátor skládání $\circ$ : hodnotám $(g_1, p_1)$ a $(g_2, p_2)$ přiřazuje

$(g_1, p_1) \circ (g_2, p_2) = (g_1 \vee (p_1 \wedge g_2), p_1 \wedge p_2)$

Splněné $g_i$ znamená generování carry, splněné $p_i$ přenášení carry, nesplněné ani $g_i$, ani $p_i$ shození carry.

Lemma: funkce $\circ$ je asociativní, tj. $((g_1, p_1) \circ (g_2, p_2)) \circ (g_3, p_3) = (g_1, p_1) \circ ((g_2, p_2) \circ (g_3, p_3))$.

Dk:

$$\begin{aligned} LS &= ((g_1, p_1) \circ (g_2, p_2)) \circ (g_3, p_3) \\ &= (g_1 \vee (p_1 \wedge g_2), p_1 \wedge p_2) \circ (g_3, p_3) \\ &= (g_1 \vee (p_1 \wedge g_2) \vee ((p_1 \wedge p_2) \wedge g_3), (p_1 \wedge p_2) \wedge p_3) \\ &= (g_1 \vee (p_1 \wedge (g_2 \vee (p_2 \wedge g_3))), p_1 \wedge (p_2 \wedge p_3)) \\ &= (g_1, p_1) \circ (g_2 \vee (p_2 \wedge g_3), p_2 \wedge p_3) \\ &= (g_1, p_1) \circ ((g_2, p_2) \circ (g_3, p_3)) \quad QED. \end{aligned}$$

Pozn. Funkce $\circ$ není komutativní.

Lemma 2: Výpočet carry. Pokud zavedeme
$(G_0, P_0) = (g_0, p_0)$
$(G_i, P_i) = (g_i, p_i) \circ (G_{i-1}, P_{i-1})$ pro $i = 1..n-1$
potom $c_i = G_i$ pro $i = 0..n-1$

Dk: indukcí: $i = 0$
$c_0 = g_0 \vee (p_0 \wedge c_{-1}) = g_0 \vee (p_0 \wedge 0) = g_0 \vee 0 = g_0 = G_0$

pokud platí lemma pro $0..i-1$, potom pro $i$ máme

$$\begin{aligned}(G_i, P_i) &= (g_i, p_i) \circ (G_{i-1}, P_{i-1}) \\ &= (g_i, p_i) \circ (c_{i-1}, P_{i-1}) \\ &= (g_i \vee (p_i \wedge c_{i-1}), p_i \wedge P_{i-1}) \\ &= (c_i, p_i \wedge P_{i-1}) \quad QED.\end{aligned}$$

Počítáme ve dvou fázích. Výpočty operátorem $\circ$ uzávorkujeme do vyváženého stromu, konkrétně výpočet $(G_{n-1}, P_{n-1})$.

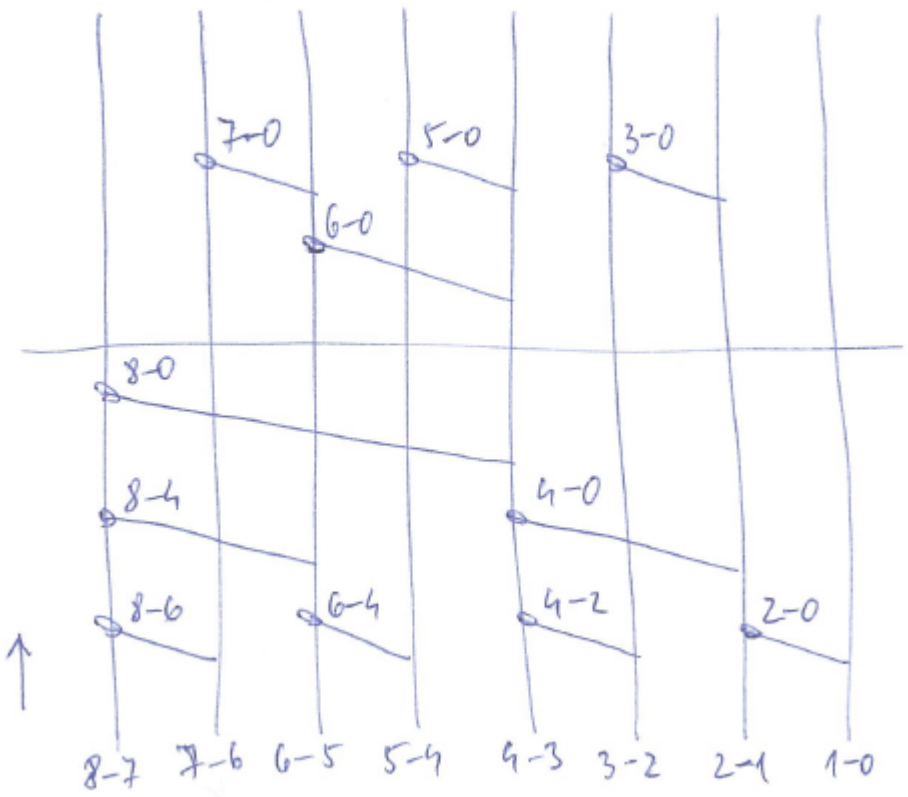
Levý a pravý operand každého výskytu $\circ$ můžeme počítat paralelně, protože na sobě nezávisí. Hloubka stromu, tj. počet úrovní je $\lceil \log_2 n \rceil$. Tím získáme mezivýsledky velikosti $2^i$, po $i$-té úrovni.

Ve druhé fázi spočítáme všechny $c_i$ z mezivýsledků[1], v počtu úrovní $\lceil \log_2 n \rceil$, a použijeme je pro určení $s_i$.

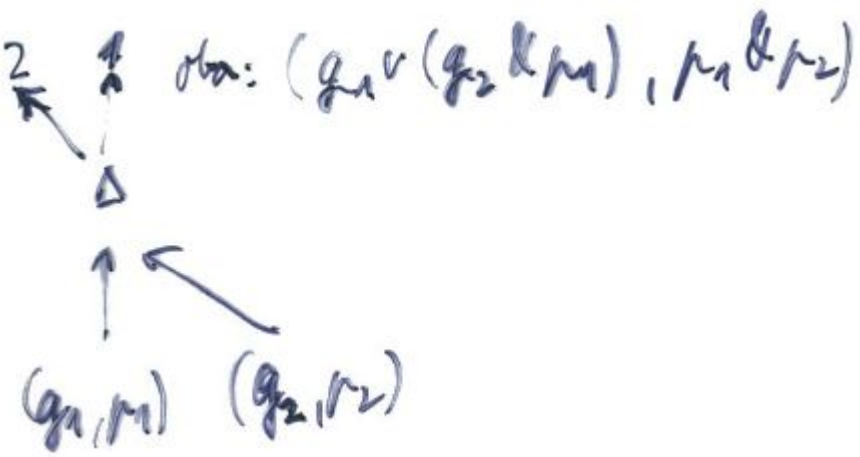
Pokud je $n = 2^l$, pak před $i$-tým krokem druhé fáze jsou spočítány $c_j$ pro $j = k \cdot 2^{l-i}$.

[1] analogie "kapitánskemu kroku"

• Sčítací síť. Data tečou zdola nahoru, nižší bity vpravo, vyšší vlevo. Dole mimo síť je preprocesing, který vytvoří funkce. Hradla pro funkci $\circ$ slučují vždy navazující hodnoty. Na konci mimo síť je postprocesing, který použije spočítané carry.




Obrázek 19: Sčítací síť šířky 8




Obrázek 20: Slučovací hradlo. Počítá $(g_1 \vee (g_2 \wedge p_1), p_1 \wedge p_2)$ z $(g_1, p_1)$ a $(g_2, p_2)$

## Rychlá (Diskrétní) Fourierova transformace

FFT - Fast Fourier Transform

Motivace: rychlé násobení polynomů.

$A(x) = \Sigma_{j=0}^{n-1} a_j x^j$

$B(x) = \sum_{j=0}^{n-1} b_j x^j$

$C(x) = A(x) \cdot B(x) = \Sigma_{j=0}^{2n-2} c_j x^j$, kde $c_j = \Sigma_{k=0}^{j} a_k b_{j-k}$

$$\begin{array}{ccc} A(x), B(x) & \rightarrow & C(x) = A(x) \cdot B(x) \\ \downarrow_{FFT} & & \uparrow_{FFT^{-1}} \\ \text{bodova repr.} & & \text{bodova repr.} \\ A(x), B(x) & \rightarrow & C(x) \end{array}$$

přímé násobení v horní části: $O(n^2)$

násobení po složkách (v bodech) v dolní části: $O(n)$

Df. Vektor koeficientů $c = (c_0, c_1, ..., c_{2n-2})$ je *konvoluce* vektorů $a = (a_0, a_1, ..., a_{n-1})$ a $b = (b_0, b_1, ..., b_{n-1})$.

Vyhodnocení polynomu v bodě $x_0$: Hornerovo schéma

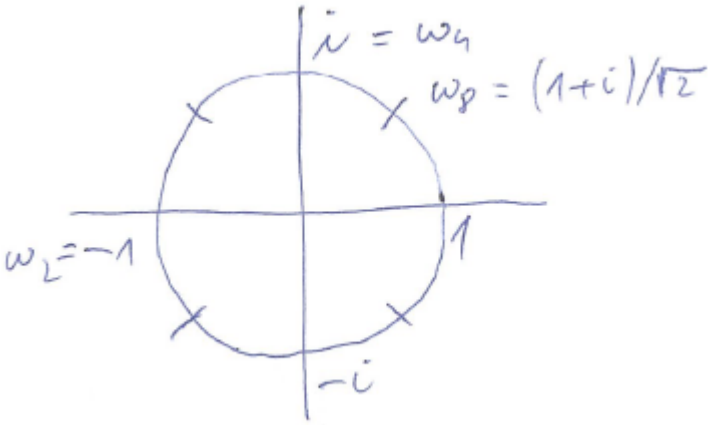
$A(x_0) = a_0 + x_0.(a_1 + x_0.(a_2 + ... + x_0.(a_{n-2} + x_0.a_{n-1})...))$

Časová složitost vyhodnocení $A(x_0)$: $O(n)$, pro $2n$ bodů celkem $O(n^2)$

(Násobení polynomů pomocí metody rozděl a panuj: $O(n^{\log_2 3})$ )

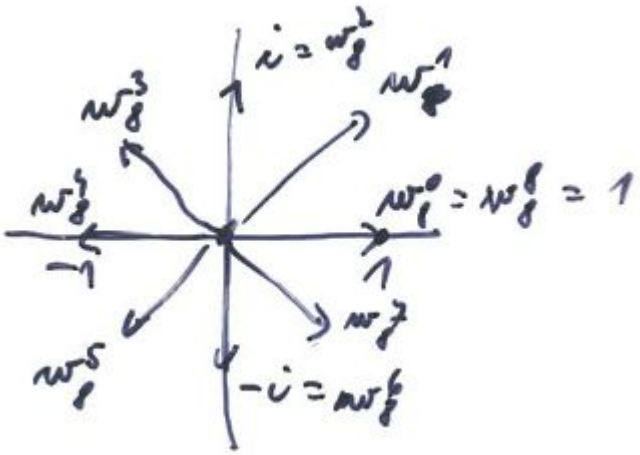
Ale: FFT (a $\text{FFT}^{-1}$) v $\Theta(n \log n)$

- vhodně vybereme body, ve kterých se budou polynomy vyhodnocovat: komplexní odmocniny 1
- využívame rozděl a panuj




Obrázek 21: Jednotková kružnice a odmocniny z 1

Komplexní odmocniny 1: $\omega_8 = \sqrt[8]{1}$, $\omega_8^8 = 1$




Obrázek 22: Odmocnina $\omega_8$ a její mocniny

komplexní $n$-té odmocniny z 1: kořeny polynomu $x^n - 1$
počet kořenů: $n$, hodnoty $e^{2\pi i \frac{k}{n}}$ pro $k = 0, \ldots, n-1$,
kde $e^{iu} = cos(u) + i \cdot sin(u)$

Hodnotu $\omega_n = e^{2\pi \frac{i}{n}}$ nazveme základní (primitivní) $n$-tý kořen 1, ostatní kořeny jsou jeho mocniny. Až $n$-tá mocnina primitivního kořene dá 1, nikoli nižší mocniny.

Pozn. Chceme $n = 2^l$, kvůli metodě rozděl a panuj.

Platí:
$\omega_{dn}^{dk} = \omega_n^k$ : LS $= (e^{2\pi \frac{i}{dn}})^{dk} = (e^{2\pi \frac{i}{n}})^k =$ PS
$\omega_n^{n/2} = \omega_2 = -1$
$(\omega_n^{k+\frac{n}{2}})^2 = (\omega_n^k)^2$ : LS $= \omega_n^{2k+n} = \omega_n^{2k} \cdot \omega_n^n = \omega_n^{2k} = (\omega_n^k)^2 =$ PS
$\Rightarrow$ druhé mocniny všech $n$ různých $n$-tých odmocnin 1 tvoří (pouze) $\frac{n}{2}$ různých $\frac{n}{2}$-tých odmocnin 1
$\Rightarrow$ rekurzivně volané podproblémy vyhodnocujme v polovičním počtě bodů
Pro $n \geq 1$ a $k \geq 0$, $n \not| k$ platí:
$\Sigma_{j=0}^{n-1} (\omega_n^k)^j = 0$, LS $= \frac{(\omega_n^k)^n - 1}{\omega_n^k - 1} = \frac{(\omega_n^n)^k - 1}{\omega_n^k - 1} = \frac{1^k - 1}{\omega_n^k - 1} = 0 = PS$
a pro $k$, kde $n|k$: $\Sigma_{j=0}^{n-1} (\omega_n^k)^j = \Sigma_{j=0}^{n-1} 1 = n$

- vyhodnocujeme polynom $A(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1}$ v bodech $\omega_n^0, \omega_n^1, \omega_n^2, \ldots, \omega_n^{n-1}$

- zápis pomocí Vandermondovy matice $F_n$: rozměry $n \times n$, na místě $(i, j)$ je $(\omega_n^i)^j$ (tj. $i$-tý kořen v $j$-té mocnině), $i, j = 0..n-1$ (v řádcích jsou body, tj. různé kořeny, ve sloupcích jsou mocniny $0..n-1$)

$$\begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega^1 & \omega^2 & \ldots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \ldots & \omega^{2n-2} \\ \vdots & & & & \vdots \\ 1 & \omega^{n-1} & \omega^{2n-2} & \ldots & \omega^{(n-1)^2} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} A(\omega^0) \\ A(\omega^1) \\ A(\omega^2) \\ \vdots \\ A(\omega^{n-1}) \end{pmatrix}$$

Df: Tato *lineární* transformace vektoru $(a_0, a_1 \ldots a_{n-1})$ na $(A(\omega^0), A(\omega^1) \ldots A(\omega^{n-1}))$ se nazývá Diskrétní Fourierova Transformace (DFT)

- matice $F_n$ má inverzi (uhádnutím, bez motivace, vhledu a odvození):
$(F_n^{-1})_{ij} = \frac{\omega^{-ij}}{n}$ , tj. inverzní matice má (až na koef. $1/n$) stejný tvar jako $F_n$ pro DFT, ale od základního kořene $\omega^{-1} = \omega^{n-1}$

T: $F_n$ a $F_n^{-1}$ jsou inverzní. Dk:

$$\begin{aligned} (F_n \cdot F_n^{-1})_{ij} &= \sum_{k=0}^{n-1} \omega^{ik} \cdot \frac{\omega^{-kj}}{n} \\ &= \frac{1}{n} \sum_{k=0}^{n-1} \omega^{k(i-j)} \\ &= \begin{cases} 1 & \text{pro } i = j \\ 0 & \text{jinak} \end{cases} \end{aligned}$$

Důsl: Inverzní DFT dokážeme spočítat ve stejném čase jako (dopřednou) DFT.

metoda FFT (Rychlá DFT): předpokládáme $n = 2^l$

k polynomu $A(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1}$ vytvoříme dva nové polynomy B(x) a C(x):

$B(x) = a_0 + a_2x + a_4x^2 + \ldots + a_{n-2}x^{n/2-1}$ (sudé koefs.)

$C(x) = a_1 + a_3x + a_5x^2 + \ldots + a_{n-1}x^{n/2-1}$ (liché koefs.)

Platí: $A(x) = B(x^2) + x \cdot C(x^2)$, (1), proto problém vyhodnocení $A(x)$ v bodech $\omega_n^0, \omega_n^1 \ldots \omega_n^{n-1}$ se redukuje na

1) vyhodnocení polynomů $B(x)$ a $C(x)$ stupně $n/2$ v bodech $(\omega_n^0)^2, (\omega_n^1)^2 \ldots (\omega_n^{n-1})^2$ - pouze $n/2$ různých bodů

2) výpočet hodnot polynomu $A(x)$ z mezivýsledků podle (1)

Algoritmus: rekurzivní_FFT(a)

01 $n$ :=length(a);
02 if $n = 1$ then return(a);
03 $\omega_n := e^{2\pi\frac{i}{n}}$; $\omega := 1$; – prim. kořen a akum.
05 $b := (a_0, a_2 \ldots a_{n-2})$; – vektor b
06 $c := (a_1, a_3 \ldots a_{n-1})$; – vektor c
07 $u$ := rekurzivní_FFT(b);
08 $v$ := rekurzivní_FFT(c);
09 for $k := 0$ to $n/2 - 1$ do
10 $y_k := u_k + \omega \cdot v_k$;
11 $y_{k+n/2} := u_k - \omega \cdot v_k$; – společné podvýrazy $u_k$ a $v_k$
12 $\omega := \omega \cdot \omega_n$; – $\omega$ je akt. kořen
13 od
14 return($y$);
15 end

Zdůvodnění, že vydané $y$ je DFT vstupu $a$:

- koncový případ: pro vektor délky 1: vracíme $y_0 = a_0$:

$y_0 = a_0 \cdot \omega_1^0 = a_0 \cdot 1 = a_0$.

- spočítaní hodnot v rekurzi: pro $k = 0, 1 \ldots n/2 - 1$:

z rekurze máme:

$u_k = B(\omega_{n/2}^k) = B(\omega_n^{2k})$ a

$v_k = C(\omega_{n/2}^k) = C(\omega_n^{2k})$

- odvodíme $y_k$ pro $k = 0, 1 \ldots n/2 - 1$:

$y_k = u_k + \omega_n^k v_k = B(\omega_n^{2k}) + \omega_n^k C(\omega_n^{2k}) = A(\omega_n^k)$.

- odvodíme $y_{k+n/2}$ pro $k = 0, 1 \ldots n/2 - 1$:

$y_{k+n/2} = u_k - \omega_n^k v_k \qquad - \omega_n^k = \omega_n^{k+n/2}$

$= u_k + \omega_n^{k+n/2} v_k$

$= B(\omega_n^{2k}) + \omega_n^{k+n/2} C(\omega_n^{2k}) \qquad \omega_n^n = 1$

$= A(\omega_n^{k+n/2})$, QED.

Složitost:

režie: $\Theta(n)$ v každém rek. volání (kde $n$ je akt. velikost dat)

rekurzivní vztah: $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = O(n \log n)$.

Příklad DFT a IDFT:

n=4, x=(1 0 3 2)

| | | |
|---|---|---|
| (1 0 | 3 2) | |
| (1 3) | (0 2) | |
| 1 3 | 0 2 | |
| 1 3 | 0 2 | |
| (4 -2) | (2 -2) | /·(1 i \| -1 -i) |
| (4+2 -2-2i | 4-2 -2+2i) | |
| (6 -2-2i | 2 -2+2i) | DFT |
| (6 2) | (-2-2i -2+2i) | zkouška s IDFT |
| 6 2 | -2-2i -2+2i | |
| (8 4) | (-4 -4i) | /·(1 -i \| -1 i) |
| (8-4 4+(-i)(-4i) | 8+4 4-(-i)(-4i)) | |
| (4 0 | 12 8) | /·1/4=1/n |
| (1 0 | 3 2) | OK |

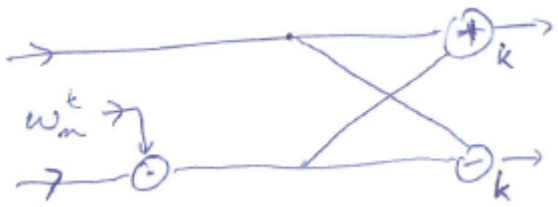
DC: FFT pro $n = 8, x = (abcdadcb)$ symbolicky, $a, b, c, d \in R$.

nápověda: $\omega_8 = (1+i)/\sqrt{2}$ a proto: $(1-i)\omega_8 = \sqrt{2}$ (obrázek)

Pozn. Řádky Vandermondovy matice kromě $i$-tého a $(n-i)$-tého jsou navzájem (jako vektory) nezávislé.

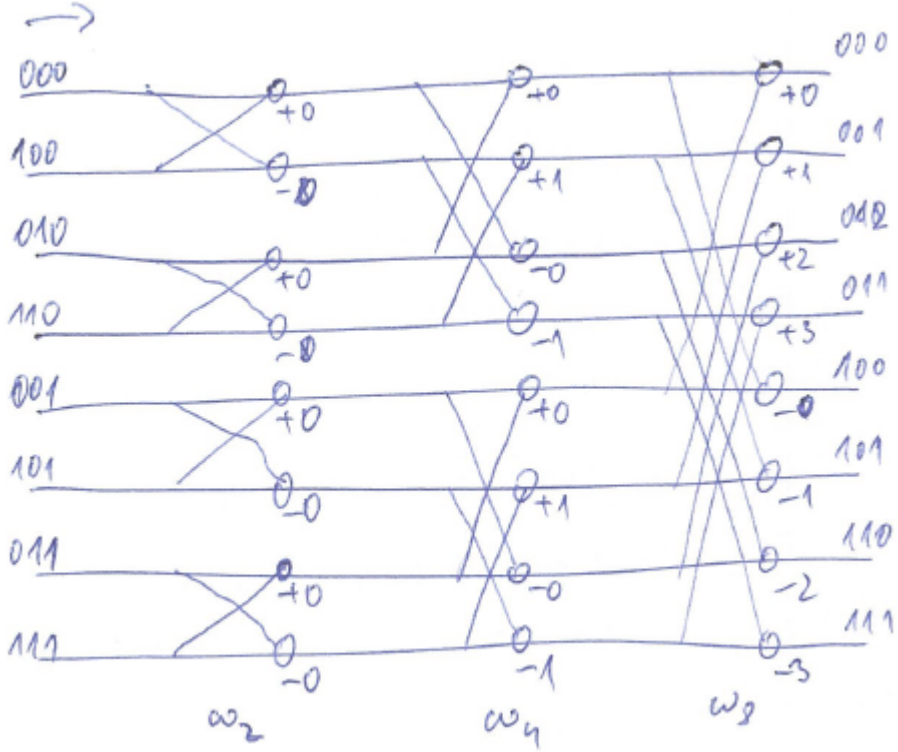
Existují a používají se i jiné transformace: kosínová (v $R^n$), waveletová ...

Butterfly operace. Data tečou zleva doprava. Každý vstup používá 2x. Druhý (dolní) vstup se násobí příslušnou mocninou kořene (a primitivní odmocniny jsou různé pro různé úrovně).



Obrázek 23: Butterfly operace.

Síť pro DFT šířky 8. Čísla určují mocninu primitivní odmocniny (příslušné úrovně) použitou v hradle.



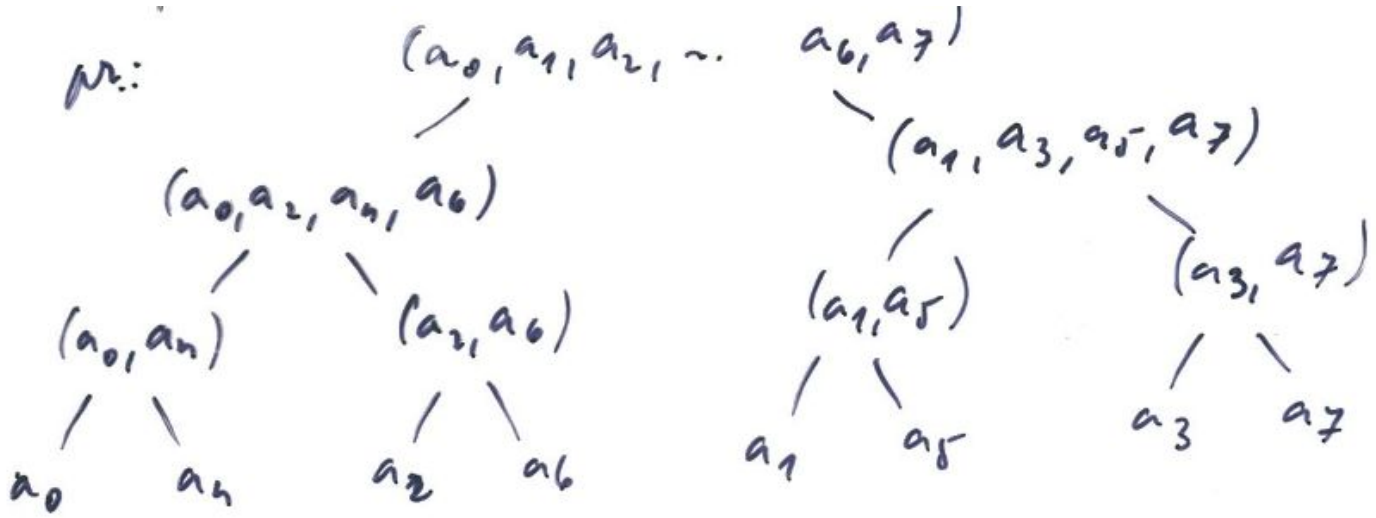

Obrázek 24: Síť pro DFT

Převod rekurze na iteraci. (”programátorský trik” z dyn. prog.):
+ menší režie
+ menší paměť (vzhledem k tabelaci z DP): použité hodnoty se

přepíšou
- složitější, tj. pracnější (delší) program

Idea: přeuspořádání vstupních koeficientů, podle reverzního bitového zápisu

$$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$$
$$(a_0, a_2, a_4, a_6) \quad (a_1, a_3, a_5, a_7)$$
$$(a_0, a_4)\ (a_2, a_6) \quad (a_1, a_5)\ (a_3, a_7)$$
$$a_0\ a_4 \quad a_2\ a_6 \quad a_1\ a_5 \quad a_3\ a_7$$



Obrázek 25: Přeuspořádání koeficientů pro nerekurzivní výpočet

alg: iterativní_FFT(a) – pseudokód

01 $n :=$ length(a);
02 for $k := 0$ to $n - 1$ do $A[rev_n(k)] := a_k$; – přeusporádání
03 for $s := 1$ to $\log n$ do – pro úrovně
04  for $k := 0$ to $n - 1$ step $2^s$ do
05  skombinuj dvě $2^{s-1}$-prvkové DFT
06  v $A[k..k + 2^{s-1} - 1]$ a $A[k + 2^{s-1}..k + 2^s - 1]$
07  do $2^s$-prvkové DFT v $A[k..k + 2^s - 1]$

Aplikace FFT
- konvoluce polynomů
- analýza signálů, spektrální
- zpracování obrazu a videa (pomocí kosínové transformace): analýza, komprese (JPEG, MPEG), syntéza (v 2D) (B)
- násobení dlouhých čísel (A)

ad (A): idea: (binární dlouhé) číslo rozdělíme na skupiny $k$ cifer a chápeme jako hodnotu polynomu v bodě $2^k$.
Součin čísel získáme jako hodnotu součinu odpovídajících polynomů v bodě $2^k$.
- hrubý odhad složitosti: $O(n \log^2 n)$, je lepší než $O(n^{\log_2 3})$ z "rozděl a panuj".

Počítání FFT v zbytkových okruzích $Z_n$: operace +, -, * počítané modulo $n$; pro vhodné $n$, obvykle prvočíslo. Výhoda: v $Z_n$ počítame beze ztráty přesnosti (a zaokrouhlovacích chyb)
např. $2^8 = 256 \equiv -1 \pmod{257}$
$\Rightarrow 2^{16} \equiv 1 \pmod{257}$
$\Rightarrow 2 = \sqrt[16]{1}$ v $Z_{257}$ , tj. 2 je primitivní $\omega_{16}$ v $Z_{257}$

ad (B): Kosínová transformace pro $n/2+1$ bodů: lze spočítat pomocí FFT na $n$ bodech.
Nechť $a = (a_0, a_1, ..., a_{n-1})$. Označme $c^*$ komplexně sdružené číslo: $re(c) - im(c)$.
Pokud $a_i = {a_{n-i}}^*$ a $a_0, a_{n/2} \in R$, potom DFT$(a) \in R^n$ (a naopak). (Př. $n = 8$: $(a, b, c, d, e, d^*, c^*, b^*)$)
Zdůvodnění: imaginární složky "združených" řádků $i$ a $n - i$ se při sčítání vyruší (ve stejných mocninách)

## Převoditelnost problémů, třídy P, NP a NPÚ

Motivace: Kromě složitosti konkrétních algoritmů nás zajímá *složitost problémů* vůči určité třídě algoritmů. (Např. rozlišujeme sekvenční výpočet a paralelní výpočet.)

Zde rozlišujeme: deterministický a nedeterministický algoritmus

Odhad složitosti problému:

horní odhad s.p. je složitost nejlepšího algoritmu (z dané třídy alg.), který problém řeší.

dolní odhad s.p. je odvozený z nějakých typických vlastností zadání problému. (viz. dolní odhad třídění $O(n \log n)$, př. hledání dosažitelných vrcholů: $O(n^2)$ - počet hran)

Rozhodovací problém: odpověď alg. je ANO/NE. Např.

1) Lze graf $G$ obarvit $k$ barvami?

2) Existuje v grafu $G$ klika velikosti aspoň $k$?

3) Existuje v $G$ řešení pro obchodního cestujícího menší/rovno než $p$ (práh). (Optimalizační problém "přeformulovaný"na rozhodovací)

4) (SAT) Je formule $\varphi$ výrokové logiky v CNF splnitelná? Tj. existuje ohodnocení výr. prom. z $\varphi$, při kterém je $\varphi$ pravdivá.

Def: *Instance* je konkrétní zadání problému.

Př: Instance pro 1) je nějaká dvojice $(G, k)$, pro 4) formule $\varphi$.

Nedeterministický algoritmus (pro rozhodovací problémy): Algoritmus může vykonat nedeterministické kroky. Pokud aspoň jedna z větví odpoví ANO, celý nedet. algoritmus odpoví ANO.

Př. ad 2) Alg. si nedeterministicky vybere $k$ různých vrcholů a ověří (v čase $O(k^2)$), že tvoří kliku. (Ale: Pro *pevné* $k$ je deterministický alg. polynomiální: $k$ vnořených for-cyklů.)

**Třídy problémů P, NP, NP-úplné**

Třída P (nebo PTIME): třída (rozhodovacích) problémů řešitelných sekvenčními deterministickými algoritmy v polynomiálním čase (tj. časová složitost je $O(n^k)$ - jsou efektivně řešitelné).

Třída NP (nebo NPTIME): třída (rozhodovacích) problémů řešitelných sekvenčními nedeterministickými algoritmy v polynomiálním čase.

Třída NP-úplných problémů (NPÚ, NP-complete): Problém A je NPÚ, pokud 1) je z NP a 2) je NP-těžký, tj. každý problém z NP je na A polynomiálně převeditelný.

Důsl.:

1) Problémy z NPÚ jsou nejtěžší problémy v NP (!),

2) NP-úplné problémy jsou navzájem polynomiálně převoditelné

3) NPÚ $\subseteq$ NP

4) $P \subseteq NP$, ale neví se, zda $P = NP$, tj. zda problémy z NP jsou polynomiálně deterministicky řešitelné. (Považuje se to za nepravděpodobné.)

! Ve třídě NP jsou problémy, které je možné *ověřit* v polynomiálním čase. Tj. pokud někdo tvrdí, že $x$ je řešení instance problému (tzv. *svědek*), jsme to schopni v polynomiálním čase ověřit (odpadá nedeterminizmus, protože testujeme jedno řešení).

Pozn.: Svědka máme pouze pro instance s odpovědí ANO, tj. situace je vzhledem k ANO a NE nesymetrická.

První NP-úplný problém získáme z definice (na konkrétním teoretickém modelu výpočtu: Turingův stroj, RAM stroj) - přednáška ze Složitosti.

Další NP-úplné problémy: polynomiálním převodem z již známých NPÚ problémů. (Pomocí $P_{znamy} \leq_p P_{novy}$ pro $P_{novy} \in NP$)

Pozn.: Sekvenční simulace nedeterministického výpočtu je možná (např. backtrackingem), ale větve/možnosti se procházejí sekvenčně a jejich čas zpracování se sčítá.

Polynomiální a nepolynomiální složitost. (opakování)

Pokud si koupíme 1000× rychlejší počítač, pak u algoritmu složitosti $O(n^3)$ můžeme ve stejném čase zpracovat data 10× větší ($\sqrt[3]{1000} = 10$) než předtím. V obecnosti několikanásobně větší.

U algoritmu exponenciální složitosti $O(2^n)$ můžeme zpracovat data o $\log_2 1000 \approx 10$ větší. V obecnosti k velikosti vstupních dat *přičítáme* konstantu.

Reklamní vložka: constraint programming - programování s omezujícími podmínkami: Zadám obory hodnot proměnných (klika: $k$ proměných pro vrcholy $G$) a podmínky na proměnné (klika: $k$ vybraných vrcholů tvoří úplný graf; a hrany grafu $G$ jako podmínky na hodnoty proměnných v relaci). Potom chytré a optimalizované[2] algoritmy hledají možné řešení (případně optimální řešení pro optimalizační úlohy).

[2]Efektivní, ve smyslu optimalizované, ne v dříve použitém smyslu polynomiální

**Převoditelnost, formálně.**

Problém $P$ je transformace vstupních dat (nad abecedou $\Sigma$) na výstupní data (nad $\Theta$).

$f_P : \Sigma^* \to \Theta^*$

Př. Násobení: $\Sigma = \{0, 1, *\}$, $\Theta = \{0, 1\}$

Def. Nech $f_1 : \Sigma_1^* \to \Theta_1^*$, resp. $f_2 : \Sigma_2^* \to \Theta_2^*$ je formalizace $P_1$, resp. $P_2$. Problém $P_1$ je *převoditelný* na problém $P_2$, píšeme $P_1 \leq P_2$, pokud existuje $g : \Sigma_1^* \to \Sigma_2^*$ a $h : \Theta_2^* \to \Theta_1^*$ (!) tak, že $\forall x \in \Sigma_1^* : f_1(x) = h(f_2(g(x)))$.

Pozn. Definice je obecná, na funkce $g$ a $h$ můžeme (a budeme) klást další podmínky.

Př. $P \leq P$. ($g$ a $h$ jsou identity)

Př. Problém hledání kostry je převeditelný na problém hledání minimální kostry.

Omezíme se na rozhodovací problémy: $P$ je formalizován funkcí $f$: $\Sigma^* \to \{ANO, NE\}$
Pokud $f$ považujeme za charakteristickou funkci $\chi_P$, potom problém $P \subseteq \Sigma^*$. (Pro syntakticky nesprávné vstupy je odpověď NE.)

Pro rozhodovací problémy: $P_1 \leq P_2$, pokud existuje $g(x)$, tž. $x$ je řešení $P_1$, právě když $g(x)$ je řešení $P_2$.

Odpověď pro $P_1$ na $x$ získáme složením $g$ a odpovědi $P_2$ na $g(x)$. Tj. $\chi_{P_1}(x) = \chi_{P_2}(g(x))$, a taky $x \in P_1 \equiv g(x) \in P_2$

Pozn. Funkce $g$ je z celé $\Sigma_1^*$ do části $\Sigma_2^*$. Pro druhý směr převodu - $P_2 \leq P_1$ - obvykle potřebujeme jinou funkci.

**Polynomiální převoditelnost**

$P_1 \leq_p P_2$ : $P_1$ je *polynomiálně převoditelný* na $P_2$, pokud $P_1$ je převoditelný na $P_2$ a $g(x)$ má polynomiální složitost.

Důsl.: Nech $P_1 \leq_p P_2$.

a) Pokud pro $P_1$ neexistuje polynomiální algoritmus, potom neexistuje ani pro $P_2$. (Dk: Sporem.)

b) Pokud pro $P_2$ existuje polynomiální algoritmus, potom existuje i pro $P_1$.

c) Relace $\leq_p$ je tranzitivní. Z toho:

d) Pokud $P_1$ je NPÚ a $P_2$ je z NP, pak $P_2$ je NPÚ.

Pozn.: Připomínám, že velikost vstupů měříme v bitech. Např. testování prvočíselnosti $n$ procházením do odmocniny je polynomiální vůči $n$, ale exponenciální vůči délce bitového zápisu $n$.

Ukážeme, že problém Splnitelnosti formulí výrokové logiky v konjunktivní normální formě (SAT) je polynomiálně převoditelný na problém Kliky (CLIQUE).

Problém SAT: SATisfiability - Splnitelnost.

Syntax (zápis): Formule výrokové logiky jsou:

1. $x_i$, tj. atomické formule
2. $(A).(B)$
3. $(A) + (B)$
4. $(\overline{A})$

pro formule $A$, $B$.

Konvence: Nadbytečné závorky můžeme vypustit.

Př. $(x1) + ((x2).(\overline{x3}))$ zkrátíme na $x1 + (x2.\overline{x3})$

Sémantika (význam):
Pravdivostní ohodnocení $v$: výrokové proměnné $\rightarrow$ {True, False}.
Pravdivostní funkce $\phi_v$: formule $\rightarrow$ {True, False}.

Funkce $\phi_v$ je definovaná rekurzivní indukcí podle struktury (zápisu) formule:

1. $\phi_v(x_i) = v(x_i)$
2. $\phi_v(A.B) = \phi_v(A) \wedge \phi_v(B)$
3. $\phi_v(A + B) = \phi_v(A) \vee \phi_v(B)$
4. $\phi_v(\overline{A}) = \neg\phi_v(A)$

Formule $A$ je *splnitelná*, pokud existuje $\phi_v$, tž. $\phi_v(A) = True$.

Př. $(x_1.\overline{x}_1)$ je nesplnitelná

Př. $(x_1 + \overline{x_2}).(x_2 + \overline{x_3}).(x_3 + \overline{x_1})$ je splnitelná, pro $v(x_1) = v(x_2) = v(x_3) = T$

Pozn.: ! Rozlišujte:
znaky . + $\overline{x}$ jsou formální zápisy
$\vee, \wedge, \neg$ jsou boolovské funkce

Testování splnitelnosti:
pro $m$ proměnných, $2^m$ pravdivostních ohodnocení.

Konjunktivní normální forma (KNF):
$A$ je zápis $F_1.F_2 \ldots F_p$, kde
$F_i$ je zápis $L_{i,1} + L_{i,2} + \ldots + L_{i,q_i}$ – klauzule
$L_{i,j}$ je ve tvaru $x_k$ anebo $\overline{x}_k$ – literál

Pro zápis KNF uvažujeme, že nejvyšší prioritu má negace, pak disjunkce a nejnižší konjunkce. Následně nepotřebujeme závorky. Proměnné číslujeme binárně: `x1011` .

Problém $P_1$ (SAT): {zápis($A$) | $A$ je splnitelná a $A$ je v KNF}.

Problém kliky. Graf $G = (V, E)$, $k$-klika je úplný podgraf $G$ na $k$ vrcholech.

Problém $P_2$ (KLIKA): {zápis($G$,$k$) | v grafu $G$ existuje $k$-klika}.

Tvrzení: $SAT \in NP$ a $KLIKA \in NP$.
Dk.: Přímočaře: Návrhem nějakého sekvenčního nedeterministického polynomiálního algoritmu.

**Převod SAT ($P_1$) na KLIKA ($P_2$)**

Konstrukce: Mějme formuli A ve tvaru:
$A$ je $F_1.F_2 \ldots F_p$, kde
$F_i$ je $L_{i,1} + L_{i,2} + \ldots + L_{i,q_i}$ – klauzule
$L_{i,j}$ je ve tvaru $x_k$ anebo $\overline{x}_k$ – literál

Vytvoříme vrcholy: $V = \{\langle i, j\rangle | 1 \leq i \leq p, 1 \leq j \leq q_i\}$.
Vrcholy odpovídají literálům $L_{i,j}$.

Hrany: $(\langle i_1, j_1\rangle, \langle i_2, j_2\rangle) \in E$ iff
$i_1 \neq i_2 \wedge L_{i_1,j_1}$ a $L_{i_2,j_2}$ nejsou vzájemnou negací (tj. mohou být současně pravdivé).

Problém $P_2$ je $((V, E), p)$, tj. velikost hledané kliky je $p$.

Chceme: 1) Důkaz, že odpověď ANO se zachovává. 2) Převod je polynomiální: lehké, např. $O(n^4)$.

Tvrzení: $A$ je splnitelná iff v grafu $(V, E)$ existuje $p$-klika.

Dk: "=>": Pro pravdivé ohodnocení, v každé klauzuli je pravdivý aspoň jeden literál $L_{i,j}$. Potom odpovídající vrchol $\langle i, j\rangle$ patří do $p$-kliky, protože dva takové vrcholy jsou spojeny hranou a vrcholů je celkem $p$.

"<=": $p$-klika určí ohodnocení některých literálů (tj. proměnných v nich). Ohodnocení je konzistentní, protože pokud se nějaké proměnné přiřazuje hodnota opakovaně, tak vždy stejná. Formule je při zkonstruovaném ohodnocení pravdivá, protože v každé klauzuli byl vybrán a je splněn aspoň jeden literál. Zbylé (neurčené) proměnné můžeme ohodnotit libovolně.

Důsl.: Pokud je SAT $\in$ NPÚ, potom KLIKA $\in$ NPÚ.

První NP-úplný problém potřebujeme získat přímo z definice. (Cookova věta: SAT je NP-úplný. Používá se i problém kachličkování.) Idea: Popíšeme podmínky (ve formě CNF pro SAT, nebo pro kachličkování) na správnou simulaci Turingova stroje a jejich splnění znamená úspěšný výpočet a teda odpověď ANO pro danou instanci. Podmínky zahrnují okrajové podmínky (na vstupu je daná instance a na výstupu je ANO) a podmínky na správnost simulace jednotlivých kroků, které jsou lokální.

**Další NPÚ problémy:**

Barvení: Pro daný graf $G = (V, E)$ a číslo $k \in N$, existuje obarvení $G$ pomocí (nejvýše) $k$ barev? (Obarvení je zobrazení $c : V \to C$, tž. $\forall (u, v) \in E$ je $c(u) \neq c(v)$, tj. sousední vrcholy jsou obarveny různě. Počet barev je $|C|$.) (Taky NPÚ: barvení grafu 3 barvami, tj. pro pevné $k = 3$.)

3-SAT: jako SAT, ale každá klauzule má nejvýše 3 literály. (Pro převod "z"problému chceme co nejomezenější zadání.)
Autotest: Platí SAT $\leq_p$ 3-SAT a 3-SAT $\leq_p$ SAT. Který převod je triviální?

HAM: Existuje v daném (neorientovaném neohodnoceném) grafu $G$ Hamiltonovská kružnice, tj. kružnice procházející všechny vrcholy?

TSP (Traveling Salesman Problem), Problém obchodního cestujícího (POC): Najít kružnici kratší než $k$ přes všechny vrcholy v ohodnoceném grafu $G$.
Pozn.: Obvykle se místo rozhodovacího problému TSP řeší optimalizační formulace, tj. najděte nejkratší hamiltonovskou kružnici v $G$.

Nezávislá množina: Pro daný graf $G$ a číslo $k$, obsahuje $G$ $k$ navzájem izolovaných vrcholů?
Autotest: Převeďte Nezávislou množinu z a na problém KLIKA.

Vrcholové pokrytí: Existuje pro dané $k$ a $G = (V, E)$ množina vrcholů $S \subseteq V$, $|S| \leq k$, tž. každá hrana má aspoň jeden vrchol v $S$ (tj. $(\forall E) S \cap E \neq \emptyset$)?

Problém batohu: Pro danou množinu $S$ předmětů, celočíselnou váhovou funkci $w : S \to N$, ziskovou funkci $b : S \to N$, limitní váhu $W \in N$ a požadovaný zisk $B \in N$ určete, zda existuje podmnožina $S' \subseteq S$, tž. $\sum_{x \in S} w(x) \leq W \wedge \sum_{x \in S} b(x) \geq B$. (Používají se i jiné formulace, i optimalizační.)
Pozn. Algoritmus pomocí dynamického programování je polynomiální k hodnotě $B$ (časově i paměťově, tj. k *unární reprezentaci* $B$), ne k počtu bitů $B$ (a $x_i$).

Suma podmnožiny: Pro danou množinu $S$ čísel z $N$ a číslo $B$, existuje podmnožina $S' \subseteq S$, tž. $\sum_{x \in S'} x = B$?

Rozdělení: Pro danou množinu $S$ čísel z $N$, existuje podmnožina $S' \subseteq S$, tž. $\sum_{x \in S'} x = \sum_{x \in S \setminus S'} x$?

Platí: Rozdělení $\leq_p$ Suma podmnožiny.
Idea: Volíme $B = \frac{1}{2} \sum_{x \in S} x$. (Instantní autotest: Co když $\sum_{x \in S}$ je liché?)

Platí: Suma podmnožiny $\leq_p$ Rozdělení.
Idea: Nech $M = \Sigma_{x \in S}\, x$. Přidáme prvky $Z - B$ a $Z - (M - B)$ pro velké $Z$, např. $Z = 2M$.
DC: Najděte jiný převod.

Cvičení:

DC 1: Pro 2-SAT navrhněte polynomiální algoritmus.

DC 2: Dokažte: SAT $\leq_p$ 3-SAT.

DC 3: Pro splnitelnost formule v Disjunktivní NF navrhněte polynomiální alg.

DC 4: Co můžete říct o převodu CNF na DNF, ve světle minulého příkladu a znalostí o SAT?

DC 5: Barvení grafu 2 barvami je polynomiální. Dokažte. (Tj. jiná formulace: Je graf bipartitní?)

DC 6: Převést HAM na SAT.

Závěrečné poznámky: Co s tím?

- převoditelnost se používá k dolnímu odhadu složitosti problémů (ale: pro polynomiální algoritmy je polynomiální převoditelnost "hrubá")

- složitost, se kterou pracujeme, je složitost v nejhorším případě. Očekávaná složitost (složitost v průměrném případě) algoritmu může být polynomiální.

- instance je "malá" a přímočarý prohledávací algoritmus najde řešení (na dostupném hardwaru)

- algoritmy pro speciální data (např. rovinné grafy vs.obecné)

- algoritmy pro přibližné řešení

- algoritmy pro pravděpodobností řešení

-- trade-off (u pravděp. alg.): rychlost za kvalitu/správnost

- heuristiky (obecné i doménové) a metastrategie (např. restart, lokální vylepšování, neúplné prohledávání, ...)

- anytime algoritmy (typicky pro optimalizaci): Lze kdykoli prerušit (po nějaké minimální době běhu); čím déle počítá, tím lepší výsledky poskytne. Např. prohledáváme stavový prostor řešení od nejslibnějších částí.

- aplikace (Faktorizace čísla): kryptografie (metoda RSA)

- úlohy z praxe jsou někdy strukturovány a tuto strukturu lze využít (vs. náhodná testovací data)

-- fázový přechod: u 3-SAT jsou (náhodné) problémy s málo klauzulemi anebo s mnoha klauzulemi obvykle lehce rozhodnutelné. Těžké případy jsou typicky okolo určitého poměru počtu proměnných k počtu klauzulí. (Zdůvodnění.)

- SAT solvery: univerzální reprezentace v CNF, použitelná pro

různé problémy. Následně: pokrok v technologii solverů je široce použitelný. (A "programování" v CNF je podobné jako: hradlové sítě, výroková logika.)
Př. reprezentace: $x_{a,h,t}$ pokud (doménová) proměnná $a$ má hodnotu $h$ v čase $t$.

## Aproximační algoritmy

Pro NP-úplné problémy chceme v *polynomiálním* čase "aspoň" přibližné řešení, ale dostatečně přesné.

Předpokládejme, že pracujeme na optimalizačních problémech, kde každé řešení má kladnou cenu a chceme najít skoro optimální řešení (maximum anebo minimum).

Př:
- najít maximální kliku
- najít obarvení s minimálním počtem barev
- najít nejkratší cestu pro obchodního cestujícího
- najít největší součet podmnožiny nepřevyšující dané $b$

Df. Poměrová chyba. Nech $C^*$ je (neznámá) cena optimálního řešení. Aproximační algoritmus pro problém má *poměrovou chybu* $\rho(n)$, pokud pro každý vstup velikosti $n$ cena $C$ řešení vydaného aproximačním algoritmem splňuje $max(\frac{C}{C^*}, \frac{C^*}{C}) \leq \rho(n)$

Df. Analogicky, algoritmus má *relativní chybu* $\epsilon(n)$, pokud $\frac{|C-C^*|}{|C^*|} \leq \epsilon(n)$

Platí: $\epsilon(n) \leq \rho(n) - 1$, pro maximalizační problémy $\epsilon(n) = (\rho(n) - 1)/\rho(n)$.

Pokud je chyba algoritmu pevná, tj. nezávislá na $n$, píšeme $\rho$ a $\epsilon$.

- Idea: Chceme aproximační algoritmy, které dosahují postupně menší poměrovou (anebo relativní) chybu při použití více času. (Uživatel si určí požadovanou přesnost vs. *anytime* algoritmy)

Df. *Aproximační schéma* pro optimalizační problém je aproximační algoritmus, který dostáva instanci problému a $\epsilon > 0$, a pro každé pevné $\epsilon$ algoritmus počítá s relativní chybou $\epsilon$. Aproximační schéma je *polynomiální*, pokud pro pevné $\epsilon > 0$ algoritmus beží v polynomiálním čase vzhledem k velikosti vstupu $n$.

Df. Aproximační schéma je *úplně polynomiální*, pokud je časová složitost polynomiální v $1/\epsilon$ a $n$, kde $n$ je velikost vstupu a $\epsilon$ je relativní chyba.

Ukážeme si:
- aproximační algoritmus pro vrcholové pokrytí s poměrovou chybou 2
- aproximační algoritmus pro problém obchodního cestujícího s trojúhelníkovou nerovností s poměrovou chybou 2 (opak.)
- neexistenci polynomiálního aproximačního algoritmu pro obecný problém obchodního cestujícího (bez trojúhelníkové nerovnosti)
- (úplné) polynomiální aproximační schéma pro problém součtu podmnožiny

Př. *Schéma* polynomiálních algoritmů, které pro pevné $k$ řeší problém $k$-kliky na daném grafu $G$.
- Generuj $k$ vložených cyklů přes vrcholy, ve vnitřním otestuj, zda jsou vrcholy různé a tvoří $k$-kliku. (neoptimalizované)
- Pro pevné $k$ je algoritmus polynomiální ($O(n^k)$), ale obecně pro parametrem dané $k$ je úloha NP-úplná.

Df. *Vrcholové pokrytí* grafu $G = (V, E)$ je podmnožina $V' \subseteq V$ taková, že pro každou hranu $(u, v) \in E$ platí $\{u, v\} \cap V' \neq \emptyset$ (tj. $u \in V'$ anebo $v \in V'$). Velikost vrcholového pokrytí je $|V'|$.

*Problém vrcholového pokrytí* je najít pro daný neorientovaný graf vrcholové pokrytí minimální velikosti. (!)

T. Problém vrcholového pokrytí je NP-úplný.

Aproximační algoritmus pro vrcholové pokrytí s poměrovou chybou 2.

vstup: $G = (V, E)$
1 $C := \emptyset$
2 $E' := E$
3 while $E' \neq \emptyset$ do
4  zvol lib. hranu $(u, v) \in E'$
5  $C := C \cup \{u, v\}$  ; přidáme oba vrcholy hrany
6  vypusť z $E'$ všechny hrany incidentní s $u$ anebo s $v$
7 od
8 return($C$)

- $C$ je vrcholové pokrytí, protože každá hrana $E$ je pokrytá
- chceme ukázat, že algoritmus má poměrovou chybu 2:
uvažujme všechny hrany zvolené na řádku 4, označíme jako $A \subseteq E$. Žádné dvě hrany $A$ nesdílejí vrchol, protože hrany incidentní s vybranou hranou jsou vypuštěny na ř. 6.
Z toho plyne, že $|C| = 2|A|$. Aby jsme pokryly hrany $|A|$, potom každé vrcholové pokrytí, včetně optimálního pokrytí $C^*$, musí zahrnout aspoň jeden vrchol každé hrany $A$. Protože hrany $A$ nesdílejí vrcholy, platí $|A| \leq |C^*|$ a odtud $|C| \leq 2|C^*|$ QED.

DC: Hladový algoritmus (jako heuristika) vybírající vrchol s největším stupněm (k nepokrytému zbytku grafu) nezaručí poměrovou chybu 2.

**Problém obchodního cestujícího (POC)**

Je daný neorientovaný graf $G = (V, E)$ a nezáporná celočíselná cena $c(u, v)$ pro každou hranu $(u, v) \in E$. Hledáme hamiltonovský cyklus s nejmenší cenou.

Trojúhelníková nerovnost: funkce $c$ splňuje trojúhelníkovou nerovnost, pokud pro každé vrcholy $u, v, w \in V$ platí $c(u, w) \leq c(u, v) + c(v, w)$.

Algoritmus přibližného řešení problému obchodního cestujícího s trojúhelníkovou nerovností v polynomiálním čase s poměrovou chybou 2:
1 zvolíme vrchol $r \in V$
2 najdeme minimální kostru v $G$ s cenou $C$
3 nech $L$ je seznam vrcholů v pořadí *preorder* při procházení kostry z vrcholu $r$
4 return(hamiltonovský cyklus $H$, který prochází vrcholy v pořadí daném $L$)

Idea dk: ... $|x|$ označuje cenu $x$
- nějaká kostra $K$ je obsažena v $H^*$, proto $|K^*| \leq |K| \leq |H^*|$
- úplná cesta se zaznamenáváním vrcholů v pre- i post-order je 2-krát delší jako "nosná" kostra: $|H_{up}| \leq 2|K^*|$
- vypouštěním vrcholů z úplné cesty cenu cesty zmenšujeme, protože platí trojúhelníková nerovnost: $|H| \leq |H_{up}|$
- souhrnem: $|H| \leq |H_{up}| \leq 2|K^*| \leq 2|H^*|$

Pozn. Získaná kružnice je horní odhad optima, který můžeme (heuristicky) vylepšovat lokálně: odstranění křížení, přeuspořádání $k$-tic vrcholů ...

Trojúhelníková nerovnost je podstatná:

V: Pokud $P \neq NP$ a $\rho \geq 1$, potom neexistuje polynomiální aproximační algoritmus pro problém obchodního cestujícího s poměrovou chybou $\rho$.

Dk: Sporem. Ukážeme, že pokud existuje algoritmus $A$ z věty, potom se dá použít na polynomiální řešení problému hamiltonovské kružnice, který je NPÚ.

Nech $G = (V, E)$ je instancí problému hamiltonovské kružnice. Transformujeme $G$ na instanci POC takto: $G' = (V, E')$ je úplný graf na $V$, tj. $E' = \{(u, v) | u, v \in V \wedge u \neq v\}$, kde $c(u, v) = \begin{cases} 1 & \text{pokud } (u, v) \in E \\ \rho \cdot |V| + 1 & \text{jinak} \end{cases}$

- vytvoření $G'$ a $c$ je polynomiální v $|V|$ a $|E|$.

Uvažujme o instanci POC $(G', c)$. Pokud původní $G$ má hamiltonovský cyklus $H$, potom všechny hrany $H$ mají cenu 1 a $(G', c)$ obsahuje cyklus ceny $|V|$. Pokud $G$ nemá hamiltonovský cyklus, potom každý cyklus v $G'$ obsahuje hranu mimo $E$ a cena cyklu bude aspoň $(\rho \cdot |V| + 1) + (|V| - 1) > \rho \cdot |V|$.

Protože hrany mimo $E$ jsou drahé, je veliký rozdíl mezi cenou hamiltonovského cyklu v $G$ (cena $|V|$) a libovolného jiného cyklu (cena aspoň $\rho \cdot |V|$)

Aproximační algoritmus $A$ musí vrátit (v polynomiálním čase) hamiltonovský cyklus, pokud v $G$ existuje, protože při požadované chybě $\rho$ nemá jinou možnost.

Pokud hamiltonovský cyklus neexistuje, algoritmus vráti cyklus ceny aspoň $\rho \cdot |V|$. Teda, pomocí $A$ jsme polynomiálně vyřešili problém hamiltonovské kružnice. Spor. QED.

**Úplné polynomiální aproximační schéma pro součet podmnožiny**

Problém: $(S, t)$, kde $S$ je $\{a_1, a_2..a_n\}$, množina kladných přirozených čísel a $t$ je kladné přirozené číslo.

Rozhodovací verze: ex. podmnožina $S' \subseteq S$, tž. $\Sigma_{a_i \in S'} a_i = t$.

Optimalizační verze: hledáme podmnožinu $S' \subseteq S$, které součet je co největší, ale nepřevyšující $t$.

Značení: $S. + .x = \{s + x, s \in S\}$ pro množinu $S$ i seznam $S$

Algoritmus: SoučetPodmnožinyPřesně($S$,$t$)

1 $n := |S|$;
2 $L_0 := \langle 0 \rangle$ // seznamy
3 for $i := 1$ to $n$ do
4 $L_i :=$ mergeList$(L_{i-1}, L_{i-1}. + .a_i)$
5 vypusť z $L_i$ všechny prvky větší než $t$
6 return(maximum z $L_n$)

Procedura mergeList sloučí uspořádané seznamy do uspořádaného seznamu

- délka $L_i$ je až $2^i$, tj. alg. je exponenciální (v obecnosti)

Aproximační schéma:

idea: každý seznam $L_i$ po vytvoření "zkrátíme". Používáme parametr $\delta, 0 < \delta < 1$. Zkrátit seznam $L$ znamená vypustit co nejvíc prvků z $L$ tak, že pro každý vypuštěný prvek $y$ zůstal v seznamu $L$ prvek $z \leq y$ takový, že $\frac{y-z}{y} \leq \delta$, tj. $(1-\delta)y \leq z \leq y$.

Prvek $z$ je reprezentant $y$ s "dostatečně malou chybou"a musí být, kvůli správnosti, menší než $y$.
Algoritmus: součetPodmnožinyAprox($S$,$t$, $\epsilon$)
1 $n := |S|$
2 $L_0 := \langle 0 \rangle$
3 for $i := 1$ to $n$ do
4 $\quad L_i :=$ mergeList$(L_{i-1}, L_{i-1}. + .a_i)$
5 $\quad L_i :=$ zkrať $(L_i, \epsilon/n)$
6 $\quad$ odstraň z $L_i$ všechny prvky větší než $t$
7 nech $z$ je největší hodnota v $L_n$
8 return $z$

- prvky $L_i$ jsou součty podmnožin
- chceme: $C^*(1-\epsilon) \leq C$ pro cenu $C$ nalezeného a $C^*$ optimální řešení.
- v každém kroku zavádíme chybu $\epsilon/n$, indukcí podle $i$ lze dokázat, že pro každý prvek $y^* \leq t$ z nezkrácené verze existuje $z \in L_i$, tž. $(1-\epsilon/n)^n y^* \leq z \leq y^*$, protože $1-\epsilon \leq (1-\epsilon/n)^n \Rightarrow (1-\epsilon)y^* \leq z$
- navíc, $z$ se nezahodí v kroku 6, protože $z \leq y^* \leq t$
- schéma je úplně polynomiální:
Idea: relativní chyba $\epsilon/n$ rozsah $1..t$ rozdělí na polynomiální počet úseků, v každém je $\leq 2$ reprezentantů.

**Pravděpodobnostní algoritmy. Test prvočíselnosti.**

Pravděpodobností alg. dělá (na rozdíl od deterministického alg.) náhodné kroky, typicky využívá náhodný anebo (kvůli opakovatelnosti) pseudonáhodný generátor.
Dva výpočty pravděpodobnostího alg. na *stejných* datech mají (obvykle) různý průběh.
Zde zmíníme pravd. alg. typu Las Vegas a typu Monte Carlo.

Alg. typu Las Vegas.
Vždy dávají správný výsledek, náhodnost ovlivňuje (pouze) dobu běhu.

Příklad: Randomizace Quicksortu (při výběru pivota).
Výhody:
1) dává dobrý průměrný čas (tj. $O(n \log n)$) pro všechny data - žádný vstup není apriori špatný (pro každý deterministický výběr pivota existují apriori špatné vstupy)
2) lze spustit paralelně v několika kopiích a výsledek vzít z kopie, která skončí nejdřív (pro deterministickou verzi nemá takový postup smysl)

Alg. typu Monte Carlo.
Náhodnost ovlivňuje dobu běhu i správnost výsledku: Alg. může udělat chybu, ale pouze jednostranně (u odpovědí ANO/NE) a s omezenou pravděpodobností.

Čím déle běží, tím přesnější výsledky dává.

Příklad: Rabin-Millerův algoritmus na testování prvočíselnosti.

Úloha: pro zadané přirozené číslo $n$ (rychle) rozhodnout, zda je $n$ prvočíslo.

Aplikace: např. pro RSA potřebujeme (nová, velká) prvočísla. (Hustota prvočísel, ...)

**Věta** (malá Fermatova): Pro prvočíslo $p$ a číslo $a$, $a < p$, které jsou vzájemně nesoudělné, platí $a^{p-1} \equiv 1 \pmod{p}$.

Použití F.V.: testování prvočíselnosti.
Idea: Pokud pro nějaké číslo $a$ není splněn závěr F.V., pak $p$ není prvočíslo (určitě!). Číslo $a$ je svědek složenosti $p$.

Tvrzení F.V. v opačném směru platí často, ale ne vždy. (Např. pro pevné $a = 2$, často platí: Pokud je

$$a^{(n-1)} \equiv 1 \pmod{n} \qquad (A)$$

, pak $n$ je prvočíslo. Nejmenší "falešně pozitivní" $n$ je 341.)

Problém 1: nemáme kvantitativní odhad, jak často test selže.
Problém 2: pro některá (složená) čísla $n$ (tzv. Carmichaelova čísla) platí (A) pro *všechna* $a < n$ nesoudělná s $n$.

Tvrzení: Test složenosti čísla $\in$ NP. (Svědek: rozklad)

Tvrzení: Test prvočíselnosti $\in$ NP. (Nebudeme potřebovat; *jiný* ověřovací alg.)

Od r. 2002: Test prvočíselnosti $\in$ P. (A test složenosti $\in$ P.)
Pozn. Situace, že máme nedet. alg. (tj. ověřování svědka) pro problém M i pro doplňkový problém k M, není obvyklá.

Ale: Pro nalezení rozkladu (faktorizace) čísla $n$ není znám polynomiální algoritmus. (Náročnost faktorizace se využívá v alg. RSA)

Označíme $T$ množinu všech dvojich přirozených čísel $(k, n)$, takových, že $k < n$ a platí jedna z podmínek:

a) $k^{n-1} \not\equiv 1 \pmod{n}$

b) existuje $i$ takové, že $m = \frac{n-1}{2^i}$ je celé číslo a platí $1 <$ nsd$(k^{m-1} - 1, n) < n$. ; nsd - největší společný dělitel

Vysvětlení: Ad a) $n$ porušuje podmínku malé Fermatovy věty pro prvočísla, ad b) analyzujeme opakovaně druhé odmocniny z $k^{n-1}$ pro zvyšující se $i$ dokud $k^m \equiv 1 \bmod n$; pokud $x^2 \equiv 1 \bmod n$, pak $x^2 - 1 \equiv (x+1)(x-1) \equiv 0 \bmod n$, pro $n$ prvočíslo nutně $x \equiv \pm 1 \bmod n$, pro $n$ složené můžeme dostat netriviální dělitele $n$ (a $k$ je svědek složenosti).

T.1: Číslo $n$ je složené právě tehdy, pokud existuje $k < n$ takové, že $(k, n) \in T$.

T.2: Nech je číslo $n$ složené. Pak existuje aspoň $\frac{n-1}{2}$ čísel $k < n$ takových, že $(k, n) \in T$.

Alg: Rabin-Millerův test.
Vstup: $n$ testované číslo, $m \in N$ lib.

```
begin
  for i := 1 to m do
    k[i] := Random(1,n-1);
    if T(k[i],n) then "n je složené"; konec fi
  od
  "n je prvočíslo"
end.
```

Pravděpodobnost chyby:
Pokud alg. tvrdí, že $n$ je složené, je to vždy správně (některé $k_i$

je "svědek").
Pokud alg. rozhodne, že $n$ je prvočíslo, pak se může jednat o chybu. V případě chyby všechny vybrané $k_i$ byly "ne-svědci"pro $n$, a to se může stát podle T.2 s pravděpodobností $P(chyba) \leq (1/2)^m$, pro nezávislé výběry čísel $k_i$.

Složitost alg.: Polynomiální k $\log n$, tj. počtu bitů $n$. (Důkaz složitosti testu $(k, n) \in T$ je netriviální a využívá znalosti z teorie čísel.)

Vlastnosti alg.:
Zvyšováním počtu iterací (tj. počtu testovaných $k_i$) lze dostat libovolně malou (předem zvolenou) pravděpodobnost chyby.
Jednotlivé testy (pro různé $k_i$) lze provádět paralelně.

Poznámka: podobný princip "svědků neshody"byl použit při vyhledávání vzorků – alg. Rabin-Karp.

Q: Jak by se choval alg. typu Las Vegas pro testovaní prvočíselnosti?
A: Odpovědi "ano"(určitě), "ne"(určitě), "nevím"(zřídka).

## Kryptografie

Komunikují účastníci Alice ($A$) a Bob ($B$). Každý vykonává svou část protokolu.

(Motivační) Příklad. Komutující šifry.
Používáme šifrovací funkci $e()$ - encryption, a dešifrovací funkci $d()$ - decryption.
$e() : \{0..K\} \to \{0..N\}$, $d() : \{0..N\} \to \{0..K\}$
$d()$ je zleva inverzní k $e() : \forall m : d(e(m)) = m$.

Alice má (své tajné) $e_A()$ a $d_A()$, Bob má $e_B()$ a $d_B()$.
Šifry jsou komutující: $e_A(e_B(m)) = e_B(e_A(m))$.

Protokol přenosu zprávy $m$:

1. Alice šifruje $m$ a posílá Bobovi $e_A(m)$.
2. Bob šifruje (svou šifrou) a posílá Alici $e_B(e_A(m))$.
3. Alice dešifruje $d_A(\underline{e_B(e_A}(m))) = \underline{d_A(e_A}(e_B(m))) = e_B(m)$.
4. Bob dešifruje $d_B(e_B(m)) = m$

Při každém poslání byla zpráva $m$ šifrována aspoň jedním klíčem.
Pozn.: Zpráva $m$ může být klíč pro další komunikaci (session).

**Kryptografie s veřejným klíčem (asymetrická šifra)**
- podporuje také digitální podpis

- Každý účastník má svůj veřejný klíč $P_X$ (public) a soukromý klíč $S_X$ (secret)
- $S_X$ zná pouze X, veřejný klíč $P_X$ je poskytnut každému, může být ve veřejném seznamu
- Klíče $P_X$ a $S_X$ specifikují funkce na množině $D$ všech zpráv (konečné posloupnosti bitů) - prosté a na, tj. permutace na $D$.
  Pozn.: Prakticky - Blokové použití, různé implementace $f$:
  $Cipher_i = f(Key, Plain_i, \{Cipher_{i-1}|Plain_{i-1}|i\})$
  Výhoda: stejný blok se na různých místech kóduje různě.
- Funkce $P_X()$ a $S_X()$ jsou efektivně vyčíslitelné, pokud známe příslušný klíč.
- Platí: $\forall M \in D : P_X(S_X(M)) = M \wedge S_X(P_X(M)) = M$, tj. funkce jsou vzájemně inverzní pro lib. $M$ (message).
- Bezpečnost šifry je zaručena, pokud nikdo kromě $X$ není schopen spočítat $S_X(M)$ pro lib. $M$ v rozumném čase. Proto požadujeme:
  1) soukromý klíč $S_X$ vlastní pouze $X$ (a chrání ho)
  2) funkce $S_X()$ nesmí být efektivně vyčíslitelná na základě znalosti $P_X$ (a funkce $P_X()$) - těžká část návrhu.

Ve skutečnosti jsou algoritmy funkcí známé, tj. veřejné, a pouze klíče se tají.

Poslání zprávy $M$ od $B$ (Bob) pro $A$ (Alice), odposlouchává $E$ (Eva, eavesdropper).

1. Bob získá veřejný klíč Alice $P_A$ (od Alice, z "webu"nebo od Certifikační Autority)
2. Bob spočíta zašifrovaný text $C = P_A(M)$ a pošle ho Alici
3. Alice použije na $C$ (od kohokoli) $S_A$ a získá $M = S_A(C)$.

Protože Eva nemá klíč $S_A$, neumí spočítat $M$ z $C$.
Pozn.: Bob potřebuje vědět, že klíč $P_A$ je Alice.
Pojmy: zpráva - plaintext, zašifrovaný text - ciphertext

Poslání autentizované a podepsané (nezašifrované) zprávy $M'$ od $A$ pro $B$.

1. Alice spočítá digitální podpis $s = S_A(M')$
2. Alice pošle Bobovi zprávu a podpis: $(M', s)$ - zpráva (zde) není šifrována
3. Bob získá $P_A$ a ověří $M' = P_A(s)$. (Zpráva obsahuje jméno Alice, aby Bob věděl, který klíč $P_A$ použít.)
4. Pokud zpráva souhlasí, Bob ví, že zpráva je od Alice a nebyla cestou změněna.

Praktické pozn.: V kroku 2 se zprávy taky šifrují, tj. metody se kombinují.

**Hybridní šifrování.**

Asymetrické šifry jsou pomalé, symetrické šifry jsou rychlejší. Symetrická šifra používá stejný klíč $K$ pro šifrování i dešifrování, např. AES (a prolomený DES).
Pro symetrickou šifru platí $C = K(M)$ a $M = K(C)$ pro lib. zprávu $M$ a klíč $K$. Klíč $K$ je krátký, stovky až tisíce bitů.

Místo (náročného) šifrování $C = P_A(M)$ Bob počítá $C' = K(M)$ a $K' = P_A(K)$ a posílá $(C', K')$. Alice dešifruje klíč $K = S_A(K') = S_A(P_A(K))$, a pak dešifruje $M = K(C)$ (a spočítá a skontroluje otisk).

Klíč $K$ se vygeneruje jednorázově pro poslání zprávy nebo pro session. Jsou i jiné protokoly pro předání klíče nebo zajištění shody na klíči (např. Diffie-Hellman-Merkle), které můžeme kombinovat s popsanými algoritmy.

**Hybridní autentizace.**

Pro dlouhou zprávu je náročné počítat digitální podpis $s = S_A(M')$. Místo celé zprávy $M'$ se podepisuje pouze otisk, získaný (veřejnou, jednocestnou - one-way) hašovací funkcí $h$ (např. MD5, SHA-1, SHA-2 atd.), která má následující vlastnosti:

1. pro dlouhou zprávu $M$ lze $h(M)$ spočítat rychle. Typicky je $h(M)$ krátký (128 - 512-bitový) otisk (fingerprint) zprávy $M$.

2. je výpočetně náročné najít dvě zprávy $M$ a $M'$, aby $h(M) = h(M')$, tzv. kolize. (A tedy pro danou $M$ najít $M'$.)

Protokol poslání podepsané a šifrované zprávy $M$ od $A$ k $B$ (Vše v jednom).

1. Alice získá Bobův klíč $P_B$.
2. Alice generuje symetrický klíč $K$, počítá $C = K(M)$ a zašifruje klíč pomocí $P_B(K)$.
3. Alice počítá otisk $h(M)$ a z něj digitální podpis $s = S_A(h(M))$.
4. Posíla Bobovi: $(od : A, C, P_B(K), s)$.
5. Bob přečte $od : A$, a získá $P_A$.
6. Bob spočítá klíč $K = S_B(P_B(K))$ a dešifruje $M = K(C)$. Dešifrovat $K$ dokáže pouze vlastník $S_B$.
7. Bob spočítá otisk $h(M)$ a porovná s dešifrovaným podpisem $A$: $P_A(s) = P_A(S_A(h(M)) = h(M)$. Podepsat $h(M)$ dokáže pouze vlastník $S_A$, zjistit $h(M)$ kdokoli (odposlechnutím a použitím veřejného $P_A$ na $s$).
8. Pokud dojde k neshodě spočítaného otisku a dešifrovaného podpisu, pak podpis není $A$ anebo došlo (úmyslně nebo neúmyslně) ke změně zprávy $M$. Je těžké podvrhnout zprávu $M'$, protože je těžké najít (vhodnou) zprávu se stejným otiskem jako $M$.

**Certifikační autority.**

Při získávání klíče potřebuje Bob vědět, že klíč $P_A$ patří Alici (a nejde o podvrh).

Jedno z řešení (základních, v principu používaných):

- Existuje certifikační autorita $Z$, jejíž veřejný klíč je známý (přišel s instalací, lze ho *ověřit* na webu).
- Alice získá (bezpečnou cestou) od autority $Z$ podepsaný *certifikát* pro zprávu $C$="Alicin veřejný klíč je $P_A$", tj. dvojici $(C, S_Z(C))$.
- Tuto dvojici připojí Alice ke každé podepisované zprávě, Bob (a každý vlastník $P_Z$) si může ověřit, že $C$ bylo vydáno autoritou $Z$ a klíč $P_A$ patří Alici.

**Def.**: Největší společný dělitel čísel $a$ a $b$ je nejmenší kladné číslo z množiny $\{ax + by | x, y \in \mathbf{Z}\}$, značíme nsd$(a, b)$.

Technika: Rozšířený Euklidův algoritmus
Vstup: $a \geq 0, b \geq 0$
Výstup: $d = \text{nsd}(a, b), x, y : d = ax + by$
Alg:

```
RozšířenýEuklid(a,b)
  if b=0
    then return (a,1,0)
  (d',x',y') := RozšířenýEuklid(b, a mod b)
  (d, x, y ) := (d', y', x' - (a div b)*y')
  return (d,x,y)
```

Správnost:
Pro výsledek rekurze platí: $d' = bx' + (a \bmod b)y'$
Dále platí: $d = \text{nsd}(a, b) = d'$
Chceme $x$ a $y$, tž. $d = ax + by$ (1).

Úpravou dostaneme:
$d = d' = bx' + (a \bmod b)y'$
$= bx' + (a - \lfloor a/b \rfloor \cdot b)y'$
$= ay' + b(x' - \lfloor a/b \rfloor y')$

Proto volba $x = y'$ a $y = x' - \lfloor a/b \rfloor y'$ zaručuje splnění (1).
Použití: pro počítání inverzních prvků v $Z_n$

**Tvrzení**: Pro $n$-bitová čísla potřebuje RozšířenýEuklid $O(n^3)$ bitových operací.
Idea dk.: Nejmenší čísla (tj. nejhorší případ) při daném počtu kroků jsou Fibonacciho čísla.

**Kongruence a okruhy zbytkových tříd.**

Pro pevné $m$ definujeme relaci kongruence modulo $m$ takto: $a \equiv b \pmod{m} =_{def} m|(a-b)$ .

Faktorová množina $Z_m = Z/\equiv \pmod{m}$ zbytkových tříd modulo $m$ s prvky $\langle a \rangle_m$ může být ztotožněna s množinou {0..m-1}. Operace sčítání a násobení jsou definovány $\langle a \rangle_m + \langle b \rangle_m = \langle a+b \rangle_m$ a $\langle a \rangle_m \cdot \langle b \rangle_m = \langle a \cdot b \rangle_m$ a $\langle 0 \rangle_m$ je nulou a $\langle 1 \rangle_m$ je jedničkou.

Multiplikativní inverzní prvek k $a$ v $Z_m$ je $x$, značený $\langle a \rangle_m^{-1}$, tž. $a \cdot x \equiv 1 \pmod{m}$, pokud $a$ a $m$ jsou nesoudělné.

Výpočet multiplikativního inverzního prvku $x = \langle a \rangle_n^{-1}$ pomocí `RozšířenýEuklid(a,n)` pro nesoudělné $a$, $n$ . Volání vrátí $(1, x, y)$, tž. $1 = a \cdot x + n \cdot y$, tj. $a \cdot x \equiv 1 \pmod{n}$.

**Def.**: Eulerova funkce $\phi(n)$ je pro $n > 1$ počet kladných celých čísel menších než $n$, nesoudělných s $n$.

**Věta**: Pokud $n$ je prvočíslo, pak $\phi(n) = n-1$. Pokud $n = pq$, kde $p$ a $q$ jsou různá prvočísla, pak $\phi(n) = (p-1)(q-1)$.

**Věta** (Eulerova): Pro $a$, $n$ nesoudělné, tj. nsd$(a, n) = 1$, platí $a^{\phi(n)} \equiv 1 \pmod{n}$. (bez dk.)

**Důsl.**: Pokud nsd$(a, n) = 1$, potom multiplikativní inverzní prvek $\langle a \rangle_n^{-1} = \langle a^{\phi(n)-1} \rangle_n$.

**Věta** (malá Fermatova): Pro prvočíslo $p$ a číslo $a$, $a < p$, které jsou vzájemně nesoudělné, platí $a^{p-1} \equiv 1 \pmod{p}$.
Důsledek Eulerovy věty, pro prvočíslo $p$ je $\phi(p) = p - 1$ .
Použití: testování prvočíselnosti.

## RSA šifra (Rivest, Shamir, Adelman)

1. Vyber dvě velká prvočísla $p$ a $q$ (každé má stovky bitů)
2. Spočítej $n = pq$. Spočítej $r = \phi(n) = (p-1)(q-1)$
3. Vyber malé liché číslo $e$, nesoudělné s $r$, tj. s $(p-1)(q-1)$.
4. Spočítej multiplikativní inverzní prvek $d$ k $e$ modulo $r$.
5. Zveřejni $(e, n)$ jako veřejný RSA klíč a uschovej $(d, n)$ jako soukromý RSA klíč.

Věta (korektnost RSA): Funkce $P(M) = M^e \pmod n$ a $S(M) = M^d \pmod n$ definují dvojici inverzních transformací na $Z_n = \{0, 1, \ldots, n-1\}$.

Důkaz: Pro všechny $M \in Z_n$ platí: $P(S(M)) = S(P(M)) = M^{ed} \pmod n$.

Protože $e$ a $d$ jsou inverzní prvky modulo $r$, můžeme upravovat (pro vhodné $c$)

$M^{ed} \pmod n \equiv M^{1+cr} \pmod n$

$\equiv M \cdot M^{c\cdot\phi(n)} \pmod n$

$\equiv M \cdot 1 \pmod n$

$\equiv M \pmod n$. Q.E.D

Pozn.: Nalezení velkých prvočísel pomocí pravděpodobnostního algoritmu je samostatně.

Při přípravě klíčů a v důkazu se používá i ( mod $n$) i ( mod $r$).

Pro RSA si klíče $P$ a $S$ můžete připravit sami a nechat si $P$ u Certifikační Autority pouze podepsat.

Zprávu dělíme na bloky dovolené velikosti (podle počtu bitů $n$) a použijeme blokový mod přenosu.

Příklad (ověřeno):
Volíme $p = 47$, $q = 71$.
Spočítáme $n = 3337$, $r = (p-1)(q-1) = 3220$.
Volíme $e = 79$, spočítáme $d = \langle 79 \rangle_{3220}^{-1} = 1019$.
Klíč $P$ je $(79, 3337)$.
Bob posílá $M = 688$.
$P(M) = \langle M^e \rangle_n = \langle 688^{79} \rangle_{3337} = 1570 = C$
My dešifrujeme:
$S(C) = \langle C^d \rangle_n = \langle 1570^{1019} \rangle_{3337} = 688 = M$

Proč je RSA bezpečná?
Na základě $(e, n)$ není (zatím) nikdo schopen rychle spočítat $d$, aniž by znal rozklad $n = p \cdot q$ a teda $\phi(n) = (p-1)(q-1)$. Faktorizace velkých čísel je výpočetně těžký problém.

Pozn.: Jsou i jiné (vhodné) těžké problémy, např. diskrétní logaritmus (v $Z_n$), na kterých jsou založené kryptografické algoritmy.

Jak je RSA rychlá?
Použijeme rychlé umocňování, `umocni(a,b,n)` počítá ($a^b$ mod $n$).

```
umocni(a,b,n) =
  if b==1
    then a
    else if sude(b)
      then umocni((a*a) mod n, b div 2,n)
      else (a*umocni(a,b-1,n)) mod n
```

Počet bitů předávaných čísel je $t = \lceil \log(n) \rceil$ a po násobení pokaždé operace modulo čísla zkrátí. Pro $t$ bitová čísla potřebujeme $O(t)$ aritmetických operací na (nejvýše) $2t$-bitových číslech, jednotlivé aritmetické operace potřebují $O(t^2)$ bitových operací, máme celkem $O(t^3)$ bitových operací.

**Konvexní obal, v rovině**

Df. Množina bodů $A \in R^n$ je *konvexní* iff (právě když) pro $\forall a, b \in A$ a $\forall t, 0 \leq t \leq 1$, platí $ta + (1-t)b \in A$.

*Konvexní obal* množiny $A$ je průnik všech konvexních množin v $R^n$, které obsahují $A$. (!Nekonstruktivní def.)

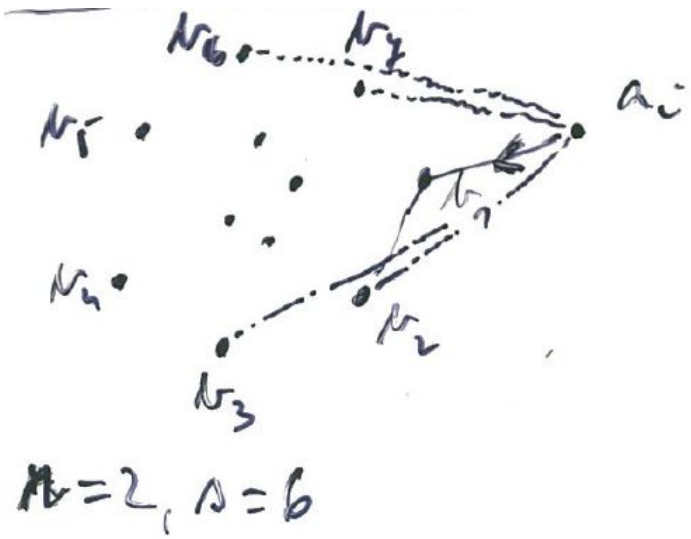
Pozn. Konvexní obal je dobře definovaný, protože průnik lib. systému konvexních množin je konvexní a celý prostor $R^n$ je konvexní.

Úloha: najít konvexní obal konečné množiny $A$ v $R^2$.
Vstup: $a_1, a_2 \ldots a_n$ jsou prvky množiny $A$, seřazené vzestupně podle $x$-ové souřadnice
Výstup: body na hranici konvexního obalu, procházené po směru hodinových ručiček (Posloupnost bodů určuje konvexní obal.)

Ukážeme si lineární algoritmus pro konstrukci konvexního obalu (za předpokladu uspořádaného vstupu).

Použitý obecný princip: zametání roviny přímkou.

Idea alg.: Tvoříme konvexní obaly množin $A_i = \{a_1 \ldots a_i\}$.
Pro $A_3$ máme trojúhelník $a_1, a_2, a_3$ anebo $a_1, a_3, a_2$.
Krok od $A_{i-1}$ k $A_i$: označme konv. obal $A_{i-1}$ jako $b_1, b_2, \ldots b_k$.
Bod $a_{i-1}$ leží na hranici $A_{i-1}$, protože má maximální $x$-ovou souřadnici $\Rightarrow$ BÚNO $a_{i-1} = b_1$



Obrázek 26: Přidávání $a_i$ ke konvexnímu obalu. Bod $b_1$ se vypouští.

Předpokládejme, že zkonstruujeme polopřímky $a_ib_1, a_ib_2, ..., a_ib_k$. Jejich směrnice (úhel s osou $x$) postupně klesá, roste, klesá. Hledáme $b_r$ a $b_s$ s nejmenší, resp. největší směrnicí. Konvexní obal $A_i$ je $a_i, b_r, b_{r+1}, ...b_{s-1}, b_s$.

Hledání $r$: konstruujeme polopřímky z $a_i$ do bodů $b_1, b_2, ...b_r, b_{r+1}$. Po zjištění, že směrnice $a_ib_{r+1}$ *stoupla* oproti $a_ib_r$, víme, že $a_ib_r$ má minimální směrnici.

Hledání $s$: z bodu $a_i$ do bodů $(b_1, )\ b_k, b_{k-1}...b_s, b_{s-1}$ konstruujeme polopřímky. Analogicky, až směrnice $a_ib_{s-1}$ *klesne* oproti $a_ib_s$, je $b_s$ hledaný bod s maximální směrnicí $a_ib_s$.

Složitost alg.: Lineární k $n$ (počtu bodů $A$), pokud neuvažujeme úvodní třídění podle $x$-ových souřadnic.
Bodu $a_i$ započítáme konstrukci polopřímek do $b_r, b_{r+1}, b_s, b_{s-1}$ a případně $a_ia_j$ pro $j > i$. Ostatní polopřímky z $a_i$ započítame příslušným bodům $b_l, 1 \leq l < r, s < l \leq k$. Body $b_l$ ale vypadnou z konvexního obalu $A_i$ při přechodu od $A_{i-1}$, teda se jim započítá "do"nich vedená přímka jedenkrát v průběhu celého

algoritmu. Celkem, maximálně 5 přímek na bod, QED.

Algoritmus má *včetně* počátečního třídění složitost $O(n \log n)$, která nejde zlepšit. Algoritmus hledání konvexního obalu (včetně cyklického uspořádání) je totiž možné použít pro třídění čísel.

Pro různá reálná čísla $r_1, r_2, ..r_n$ uvažujme konvexní obal množiny $\{(r_1, -r_1^2), (r_2, -r_2^2), ..., (r_n, -r_n^2)\}$. Funkce $y = -x^2$ je konkávní, proto všechny body leží na hranici konvexního obalu a při procházení ve směru hodinových ručiček body procházíme v pořadí rostoucích $x$-ových souřadnic, tj. uspořádaně.

**Dynamické programování** (přesunuto do ADS1)

DP je *metoda* pro řešení úloh. Podobně jako metoda rozděl a panuj anebo hladový algoritmus.

DP se používá (podobně jako rozděl a panuj) při úlohách, které můžeme rozdělit na podproblémy, ale ty jsou závislé - sdílejí podpodproblémy.

DP se používá typicky na optimalizační problémy. Řešení jsou ohodnocena a ze všech řešení chceme vybrat řešení s optimální (minimální nebo maximální) cenou.

Vývoj algoritmu založeného na DP se typicky skládá z:

- Charakterizace struktury optimálního řešení
- Rekurzivní definice ceny optimálního řešení
- Spočítání ceny optima (obvykle) zdola-nahoru
- Zrekonstruování optimálního řešení ze spočítaných informací, pokud potřebujeme i řešení (nestačí cena)

Přímé použití rekurze (bez zapamatování si mezivýsledků) nedává efektivní algoritmus.

- už znáte: (optimální) násobení obdélníkových matic, (optimální) dělení mnohoúhelníku na trojúhelníky, lineární počítání Fibonacciho čísel (jedinečná hodnota je optimální), Floyd-Warshallův alg. pro všechny min. cesty v grafu.

**Nutné vlastnosti úloh pro použití DP:**
- optimální podstruktura
- překrývající se podproblémy
Obecná charakterizace úloh pro DP není známá.

**Problém má optimální podstrukturu** , pokud optimální řešení problému obsahuje optimální řešení podproblémů. (Tuto vlastnost splňují i úlohy vhodné pro hladové algoritmy) (Bellmanův Princip optimality.)

Optimální podstruktura často "přirozeně" určí prostor podproblémů. Z řešení podproblémů budeme sestavovat řešení větších problému. Chceme co nejmenší prostor podproblémů.

Důsledek: pro využití v celkovém výsledku si stačí pamatovat hodnotu nejlepšího řešení podstruktury (a případně jedno řešení pro rekonstrukci)

**Překrývajíci se podproblémy.** Pokud chceme aplikovat DP, počet různých podproblémů musí být malý, typicky polynomiální. (Chceme si pamatovat spočítaná řešení.) Pokud rekurzivní řešení počítá stejné problémy opakovaně, potom optimizační problém má *překrývající se podproblémy*.

Pro porovnání, úlohy řešené pomocí metody rozděl a panuj typicky generují stále nové podproblémy (např. používají různé části vstupu).

Na rozdíl od hladových algoritmů musíme prohledávat, protože nelze zaručit, že lokálně nejlepší volba je součásti globálního optima (tj. že lok. opt. řešení lze doplnit na optimální). Opt. řešení nelze převést na jiné opt.ř. lokálními úpravami, protože mezi opt. řešeními jsou (v obecnosti) "bariéry".

Protipř: Nejdelší cesta v grafu. Pokud si pamatuju jen koncové body, potom neplatí princip optimality. Pokud si pamatuju i vnitřní body, mám nepolynomiálně mnoho podproblémů (a DP je nepraktické).

**Tabelace** (memoization). Použijeme rekurzivní algoritmus, ale pamatujeme si výsledky spočítaných podproblémů v tabulce. (Potřebujeme poznat všechny možné parametry podproblémů a určit jejich zobrazení do tabulky. Jinak lze použít hašování.) Tento přístup má výhodu, pokud nebudeme v rekurzi procházet celý prostor podproblémů a dokážeme ušetřit čas nebo paměť.

**Výpočet zdola nahoru.** Naplánujeme počítání hodnot v tabulce tak, že všechny potřebné podproblémy jsou vždy vyřešeny. Pokud každý podproblém budeme řešit aspoň jednou, výpočet zdola nahoru je obvykle efektivnější o multiplikativní konstantu, protože má menší režii.

Při některých problémech můžeme řešení průběžně zapomínat (pokud chceme znát pouze cenu), protože je už nebudeme potřebovat. Pokud si pamatujeme hodnoty řešení pro podstruktury, můžeme zrekonstruovat optimální řešení (trade-off čas/paměť) pomocí principu optimality.

Problém: **Nejdelší společná podposloupnost - NSP.**

Definice. Pokud máme posloupnost $X = \langle x_1, x_2, ..., x_n \rangle$, její *podposloupnost* vznikne vypuštěním některých prvků.

Posloupnost $Z$ je společná podposloupnost $X$ a $Y$, pokud je $Z$ podposloupnost $X$ a zároveň $Z$ je podposloupnost $Y$. $Z$ je *nejdelší společná podposloupnost* $X$ a $Y$, pokud je $Z$ společná podposloupnost $X$ a $Y$ a neexistuje žádna delší společná podpo-

sloupnost. (NSP $Z$ není určena jednoznačně, její délka je určena jednoznačně.)

Př.: $X$=ACBA, $Y$=BDAB, pak $Z^1$=AB, $Z^2$=BA, $k$=2.

**Problém nejdelší posloupnosti** je dán dvěma posloupnostmi $X = \langle x_1, x_2, ..., x_m \rangle$ a $Y = \langle y_1, y_2, ..., y_n \rangle$ a cílem je najít (jednu) nejdelší společnou podposloupnost $Z = \langle z_1, z_2, ..., z_k \rangle$ posloupností $X$ a $Y$.

Značení: $X_i$ je podposloupnost $X = \langle x_1, x_2, ..., x_i \rangle$ posloupnosti $X$.

Řešení hrubou silou: posloupnost délky $m$ má $2^m$ podposloupností.

Věta. Charakterizace struktury optimálního řešení.
1. Pokud $x_m = y_n$, potom $z_k = x_m = y_n$ a $Z_{k-1}$ je NSP $X_{m-1}$ a $Y_{n-1}$.
2. Pokud $x_m \neq y_n$, potom $z_k \neq x_m$ znamená, že $Z$ je NSP $X_{m-1}$ a $Y$.
3. Pokud $x_m \neq y_n$, potom $z_k \neq y_n$ znamená, že $Z$ je NSP $X$ a $Y_{n-1}$.

Dk.
1. Pokud $z_k \neq x_m$, potom $Z$ můžeme prodloužit, spor. Pokud $Z_{k-1}$ není nejdelší, můžeme ji v $Z$ nahradit delší, spor.
2. Pokud $z_k \neq x_m$, potom Z je společná podposloupnost $X_{m-1}$ a $Y$. Pokud existuje $W$ – delší společná podposloupnost $X_{m-1}$ a $Y$, je taky společnou podposloupností $X$ a $Y$ a je delší než $Z$, spor.
3. Symetricky k 2.

Důsl. Problém má optimální podstrukturu.

Jiný prostor podproblémů (větší :-( ). Podproblémy byly výše charakterizovány koncem $X_i$ a koncem $Y_j$, tj. dvojicí $(i, j)$. Nejdelší podposloupnost můžeme počítat a rekonstruovat, pokud známe nejdelší společné podposloupnosti $X_{i'..i}$ a $Y_{j'..j}$. Podprostor podproblémů je charakterizován čtveřicemi $(i', i, j', j)$.

Rekurzivní řešení.
Nechť c[i,j] je délka optimální podposloupnosti $X_i$ a $Y_j$. Potom
1. $c[i, j] = 0$, pokud $i = 0$ nebo $j = 0$.
2. $c[i, j] = c[i - 1, j - 1] + 1$, pokud $i, j > 0$ a $x_i = y_j$.
3. $c[i, j] = max(c[i, j - 1], c[i - 1, j]$, pokud $i, j > 0$ a $x_i \neq y_j$.

Je pouze $\Theta(mn)$ různých podproblémů, hodnoty můžeme počítat zdola nahoru (od menších problémů k větším), např. po řádcích c[]. Výpočet jedné hodnoty potřebuje čas $O(1)$.

Kromě hodnoty můžeme uschovávat (v pomocné tab. b[0..i,0..j]) směr, odkud jsme maximální hodnotu použili. "Diag"pro případ 2, "Sloupec", resp. "Řádek"pro případ 3 - první, resp. druhý člen v max.

Rekonstrukce řešení: Podle popisů v tabulce $b$[]. Pokud tabulku $b$ nemáme, rekonstruujeme řešení odzadu, podle směru, ve kterém platí shoda. Tento princip rekonstrukce je použitelný obecně.

V tomto problému: $c[i, j]$ závisí pouze na třech sousedních prvcích ve dvou řádcích. Pokud chceme pouze hodnotu, pri výpočtu zdola nahoru si stačí pamatovat dva řádky. (DC: ještě lépe, jeden kratší řádek a dva prvky.) Pro (rychlou) rekonstrukci posloupnosti nemáme dost informací.

**Jiné úlohy**

Klasické:

**Bitonická cesta v problému obchodního cestujícího** Průchod doprava a doleva. Neposkytuje globálně optimální řešení, ale optimum v určité třídě řešení.

**Vyrovnaný tisk** Optimalizujeme druhé mocniny chyb, kromě posledního řádku.

**Optimální vyhledávací strom** Prostor podproblémů je určen začátkem a koncem uspořádaného úseku. Pamatujeme si pozici kořene, tj. optimální rozdělení.

I méně klasické:

**Hry dvou hráčů s úplnou informací** při acyklickém grafu tahů. Určujeme, zda je pozice vyhraná nebo prohraná (tzv. minimax). Ve vyhrané pozici si pamatujeme vyhrávající (optimální) tah.

**Existence odvození v bezkontextových gramatikách** Prostor: Z neterminálu $N$ existuje odvození úseku vstupu od $i$ do $j$. Boolovské hodnoty: "Existuje"je lepší než "neexistuje".

**Součet podmnožiny** - přesný i co nejtěsnější. Existence řešení, není polynomiální.

**Hledání opt. strategie** rozhodování v diskrétní optimalizaci pro konečný (i nekonečný) počet kroků.

Pozn. Tabelace nemusí být úplná. V hrách se používají pro stavy (hašovací) transpoziční tabulky, které řeší konflikty přepsáním; podle nějaké strategie (např. zůstává lepší; větší/pracnější; novější).

Při optimalizačních problémech si můžeme pamatovat min. a max. odhad řešení a postupně upřesňovat. Např. $a_{ij} = min_k(a_{ik}+$

$a_{kj}$) (Lazy vyhodnocování bool. problémů je speciální případ. Např. $a_{ij} = \vee_k(a_{ik} \wedge a_{kj})$)