• **Hledání podřetězců v textu**

Značení: Σ ... abeceda

→ x, y, z ... znaky abecedy

→ α, β, γ ... slova / řetězce → $|\alpha| = $ délka

→ ε ... prázdný řetězec, $|\varepsilon| = 0$

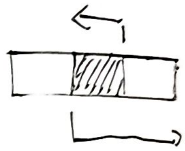
→ $\alpha\beta$ ... konkatenace

→ $\alpha[i]$ ... i-tý znak

$\alpha[i:j]$ ... $\alpha[i]\alpha[i+1] \dots \alpha[j-1]$ → $|\alpha[i:j]| = j-i$

$\left.\begin{array}{l} \alpha[:j] = \alpha[0:j] \text{ ... prefix} \\ \alpha[i:] = \alpha[i:|\alpha|] \text{ ... suffix} \end{array}\right\} \alpha[:] = \alpha$

👁 podřetězce jsou prefixy suffixů resp. suffixy prefixů



Problém

jehla ι, $J := |\iota|$ .... $\iota = $ ana

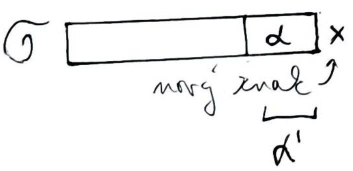
seno σ, $S := |\sigma|$ .... $\sigma = $ bananas

1, Lineární průchod → když najdu 1. písmenko tak zkontroluju ⇒ $\Theta(J \cdot S)$

↳ pokud by v jehle bylo 1. písmenko unikátní, tak $\Theta(S)$

2, Inkrementální algoritmus → postupně přidáváme znaky z sena

Def: Stav algoritmu := nejdelší prefix jehly, který je suffixem sena.



→ nový stav je buď prázdný nebo nějaké $\alpha' x$

$\left.\begin{array}{l} \Rightarrow \alpha' x \text{ je suf. } \sigma x \Rightarrow \alpha' \text{ je suf. } \sigma \\ \Rightarrow \alpha' x \text{ je pref. } \iota \Rightarrow \alpha' \text{ je pref. } \iota \end{array}\right\}$ α je nej. prefix ι ⇒ α' je suf. α

Def: Zpětná funkce $z(\alpha)$ přiřadí stavu α jeho nejdelší vlastní sufix, který je prefixem jehly. (vlastní: různý od α)

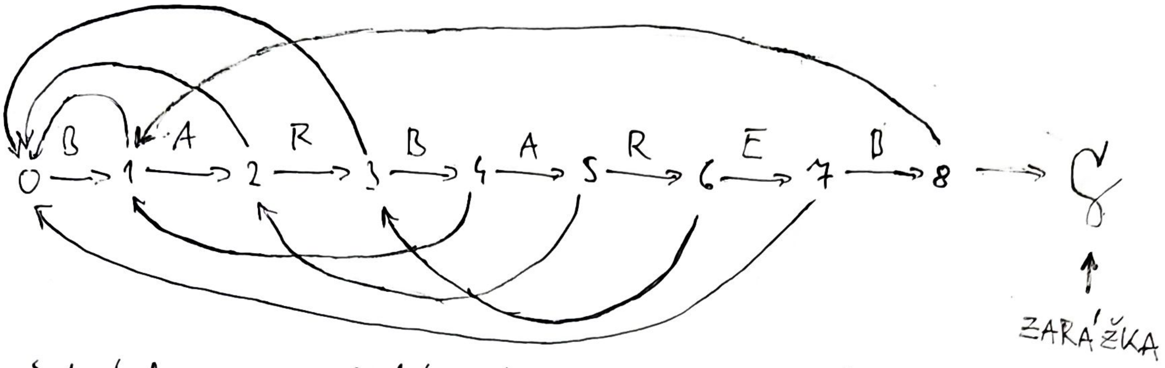
→ $\iota = $ kokos ⇒ $z(\text{koko}) = \text{ko}$

⇒ Tou zpětnou fcí budeme zkracovat α zleva dokud $\alpha' x$ nebude pref. jehly

- **KMP (Knuth, Morris, Pratt)**

→ vyhledávací automat ... stavy = délky prefixů jehly ; ι = BARBAREB

stavy:



→ dopředné hrany ~ přidávání písmenek ⇒ pamatujeme v jehle

→ zpětné hrany ~ aplikace zpětné funkce ⇒ pole $Z[|stav|] = Z(stav)$

stav, nový znak

Krok(s, x):

$s=4,\ x=R$ $\quad s=1$ $\quad s=0$

BARBR → BR → R

1. Dokud $s > 0$ & $\iota[s] \neq x$:
2. $\quad s \leftarrow Z[s]$
3. Pokud $\iota[s] = x$: return $s+1$
4. Jinak : return 0

KMP Hledej(σ)

1. $s \leftarrow 0$
2. Pro $\forall$ znaky $x \in \sigma$:
3. $\quad s \leftarrow$ Krok(s, x)
4. $\quad$ Pokud $s = J$:
5. $\quad\quad$ ohlásíme výskyt

→ pak se zavolá Krok(J, x)

⇒ indexace $\iota[J]$ → musí být zarážka

ς = znak co nikde jinde není

↳ v C by to fungovalo → string končí \0

Invariant: Stav je max. délka prefixu jehly, který je suffixem zpracovaného sena.

σ  ι ⇒ KMP funguje. ↳ díky zpětné fci.

Složitost:

\# dopředných kroků $\leq S$

$\geq$

\# zpětných kroků

\# kroků po hranách $\leq 2S$

⇒ KMP Hledej běží v čase $\Theta(S)$

⇒ KMP celkem v $\Theta(S+J)$

KMP Konstrukce(ι)

1. $J \leftarrow |\iota|$
2. $Z[0] \leftarrow \emptyset$, $Z[1] \leftarrow 0$ vždy
3. $s \leftarrow 0$ $\qquad \Theta(J)$
4. Pro $i = 2, 3, \dots J$:
5. $\quad s \leftarrow$ Krok$(s, \iota[i-1])$
6. $\quad Z[i] \leftarrow s$

→ automat postavím pomocí automatu

BARBA → chci BA

⇒ vyhledám BARBAREB v ~~B~~ARBA

BARBAR → chci BAR

⇒ vyhledám BARBAREB v ~~B~~ARBAR ↗ nový znak

⇒ stačí vyhledat BARBAREB v ARBAREB ⇒ mezivýsledky

• Aho - Corasick

σ ... seno, $S := |\sigma|$

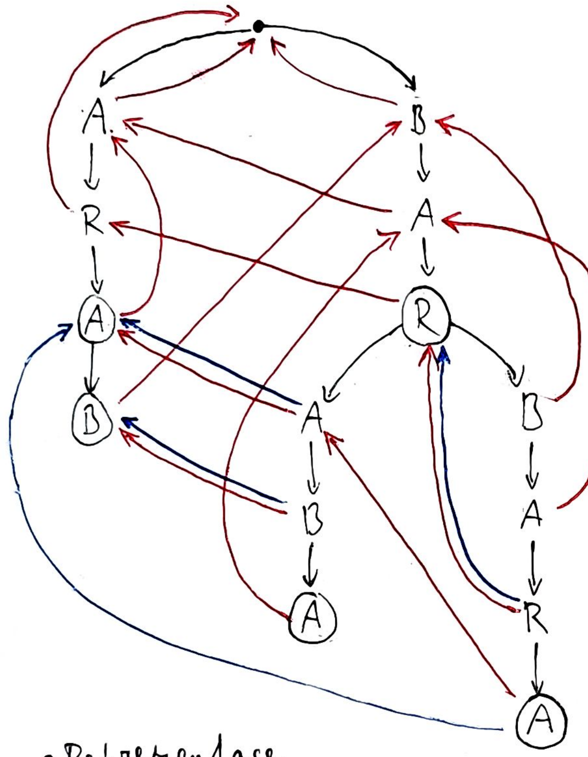
$\iota_1, \dots \iota_n$ ... jehly, $J := \sum_i |\iota_i|$

V ... # výskytů jehel v seně

• vyhledávací automat - obdobné jako u KMP

→ stavy = prefixy jehel
→ dopředné hrany : $\alpha \mapsto \alpha x$ } trie

ARA, ARAB, BAR, BARABA, BARBARA



• Zpětné hrany - stejné jako KMP

$\alpha \mapsto$ nejdelší vlastní sx α, co je stavem

→ nové písmeno → je možné ho přidat?

pokud ne, tak skáču zpětnými hranami, dokud to nepůjde nebo nebudu v kořeni

• Inv: Aktuální stav je nejdelší sx sena, který je prefixem nějaké jehly

• Zkratkové hrany - $\alpha \mapsto$ nejbližší stav dosažitelný po z. hranách, kde končí jehla

⇒ za celý běh alg. po nich projdu $\Theta(V)$

• Reprezentace

- stavy očíslujeme ⇒ kořen = 0, zbytek libovolně

HRANY
- Dopředu (s, x)
- Zpět (s) → číslo stavu nebo $\emptyset$
- Zkratka (s)

- Slovo (s) → slovo co končí v s nebo $\emptyset$

• AcKrok(s, x)

1. Dokud $s \neq$ kořen & Dopředu$(s,x) = \emptyset$:
2. $s \leftarrow$ Zpět(s)
3. Pokud Dopředu$(s, x) = \emptyset$: return kořen
4. Jinak: return Dopředu(s,x)

• AcHledej(σ)

1. $s \leftarrow$ kořen
2. Pro každý znak $x \in \sigma$:
3. $s \leftarrow$ AcKrok(s,x)
4. $q \leftarrow s$
5. Dokud $q \neq \emptyset$:
6. Pokud Slovo$(q) \neq \emptyset$:
7. hlásíme Slovo(q)
8. $q \leftarrow$ Zkratka(q)

(kroky 2–4: $\Theta(S)$; kroky 5–8: $S + \Theta(V)$)

• **Složitost** – když máme automat

👁: $S \geq$ # dopředných $\geq$ # zpětných
$V \geq$ # zkratkových } celkem $\Theta(S+V)$

• **Konstrukce automatu**

→ po hladinách ⇒ paralelní KMP pro všechny jehly

AcKonstrukce($\iota_1, \dots \iota_n$)

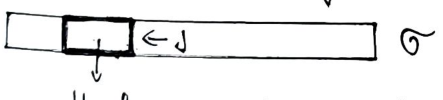
1. Založíme trii s kořenem r
2. Vložíme do trie všechny jehly < → dopředné h., → Slovo(_)
3. Zpět(r) ← $\emptyset$, Zkratka(r) ← $\emptyset$
4. F ← fronta se syny kořene
5. Synům kořene nastavíme Zpět(_) ← r, Zkratka(_) ← $\emptyset$] $\Theta(J)$

6. Dokud F není prázdná:
7. i ← F.Dequeue()
8. Pro syny s vrcholu i: (AcKrok by šel po dopředné h.)
9. z ← AcKrok(Zpět(i), znak na hraně z i do s)
10. Zpět(s) ← z
11. Pokud Slovo(z) ≠ $\emptyset$: Zkratka(s) ← z
12. Jinak: Zkratka(s) ← Zkratka(z)
13. F.Enqueue(s)

] Tady vlastně hledám ve všech jehlách ⇒ lineární s délkou sena ⇒ $O(J)$

Věta: Algoritmus A-C najde všechny výskyty jehel v čase $\Theta(J+S+V)$.

• **Rabinův-Karpův algoritmus**



⇒ porovnávám hash jehly s hashy obdélníčků

⇒ chceme h. fci co se dá přepočítat v $O(1)$ když posuneme okénko.

$$h(x_1, \dots x_J) := (x_1 p^{J-1} + x_2 p^{J-2} + \dots + x_J p^0) \bmod M$$

$$h(x_2, \dots x_{J+1}) = p \cdot h(x_1, \dots x_J) - x_1 p^J + x_{J+1}$$ ⇒ předpočítat $p^J \bmod M$

Alg:

1. $j \leftarrow h(\iota)$
2. $\sigma \leftarrow h(\sigma[:J])$] $\Theta(J)$ – hornerovo schéma
3. Pro $i = 0, \dots, S-J$:
4. Pokud $j = \sigma$ & $\sigma[i:i+J] = \iota$:] $J \cdot (V + ?)$ ← # false p.
5. Ohlásíme výskyt
6. $\sigma \leftarrow (p \cdot \sigma - \sigma[i] p^J + \sigma[i+J]) \bmod M$] S

Časová složitost → pro dokonale náhodnou $h(x)$

$\Theta(J + S + J \cdot V + J \cdot \frac{S}{M})$ → # false pos

⇒ chceme $M \in \Omega(J \cdot S)$

→ p, M nesoudělné, M prvočíslo

$\mathbb{E}$[# false pos. kolizí] $= \frac{SJ}{M}$ → reálně horší

• Toky v sítích

Def: **Síť** je struktura obsahující

- orientovaný graf (V,E), BÚNO symetrický: $uv \in E \Rightarrow vu \in E$
- **zdroj** $z \in V$ a **spotřebič** (stok) $s \in V$
- **kapacity** $c: E \to \mathbb{R}^+_0$

Def: **Tok** je funkce $f: E \to \mathbb{R}^+_0$ splňující

1) $\forall e \in E: f(e) \leq c(e)$ → hranou neteče víc než její kapacita
2) $\forall v \in V,\ v \neq z, s: f^\Delta(v) = 0$ → **Kirchhoffův zákon**

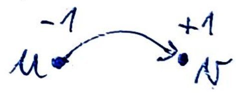
Def: Pro $v \in V$: $f^+(v) := \sum_{uv \in E} f(uv)$ ... **přítok** $\quad f[In(v)]$

$f^-(v) := \sum_{vw} f(vw)$ ... **odtok** $\quad f[Out(v)]$

$f^\Delta(v) := f^+(v) - f^-(v)$ ... **přebytek**

Def: **Velikost toku** definujeme jako $|f| := f^\Delta(s)$.

: $\underline{f^\Delta(s) = -f^\Delta(z)}$

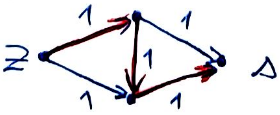
$\sum_{v \in V} f^\Delta(v) = f^\Delta(s) + f^\Delta(z)$ $\qquad \forall uv \in E$:  $\Rightarrow 0$

$\sum_{v \in V} f^\Delta(v) = 0$ ∵ ta Σ je lin. komb. toků na hranách ■

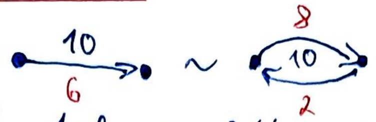
• Zvyšování toku

→ začneme s nulovým tokem a postupně ho budeme zvětšovat
→ vybereme náhodnou cestu P ze z do s

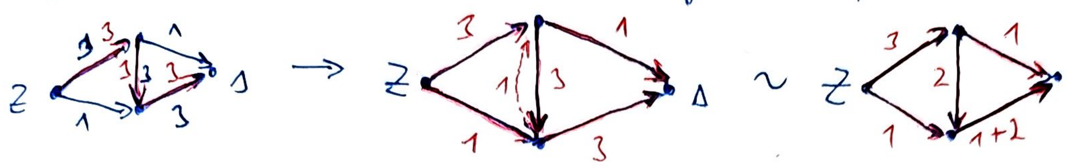
$f'(e) := \begin{cases} f(e) + \varepsilon, & e \in P \\ f(x), & e \notin P \end{cases} \qquad \varepsilon := \min_{e \in P} (c(e) - f(e))$

→ tohle se zastaví, protože vždy nasytíme alespoň 1 hranu
→ ale nenajde to max. tok  $|f| = 1$, ale max $= 2$

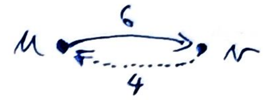
• Fordův – Fulkersonův algoritmus

→ využívá zpětné hrany  ← uvědomuji si, že tohle je ekvivalentní

→ když pustím hranou tok s, tak vyrobím zpětnou hranu stejné kapacity



$C = 10$

Def: Rezerva hrany $r(uv) := c(uv) - (f(uv) - f(vu))$ 

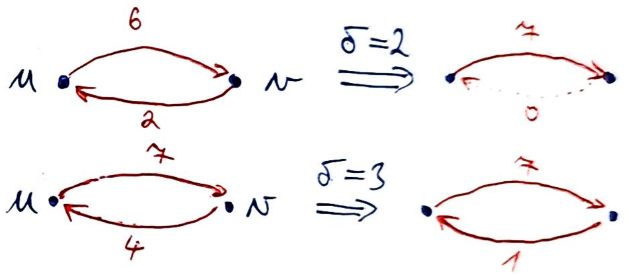
Def: Hrana $e \in E$ je nenasycená $\equiv r(e) > 0$ 

Cesta P je nenasycená $\equiv \forall e \in P: r(e) > 0$ $\quad r = 10 - (6 - 4)$

👁: Hrana uv je nasycená $\iff f(uv) = 10$ & $f(vu) = 0$.

• Algoritmus

1. $f \leftarrow$ všude nulový tok $\qquad C = 10, \; \varepsilon = 3$
2. Dokud $\exists P$ nenasycená cesta:
3. $\qquad \varepsilon \leftarrow \min_{e \in P}(r(e))$
4. $\qquad$ Pro $\forall uv \in P$:
5. $\qquad\qquad \delta \leftarrow \min(\varepsilon, f(vu))$
6. $\qquad\qquad f(vu) \leftarrow f(vu) - \delta$
7. $\qquad\qquad f(uv) \leftarrow f(uv) + \varepsilon - \delta$



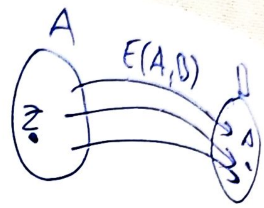
$$\left.\begin{array}{l} 6. \\ 7. \end{array}\right\} r' = c - (f(uv) + \varepsilon - \delta - f(vu) + \delta) = r - \varepsilon$$

⟹ rezerva všech hran na P klesne o ε, přičemž se vždy nejprve snažíme odečíst co nejvíc od zpětné hrany, než zvyšujeme dopřednou

• Konečnost

- pro $c \in \mathbb{N}$ ano – algoritmus zachovává celočíselnost – jen sčítá a odčítá
  ⟹ $|f|$ vždy stoupne alespoň o 1 ⟹ skončí
- pro $c \in \mathbb{Q}$ ano – invariance vůči změně měřítka – jako Dijkstra
  ⟹ vynásobím kapacity NSN jejich jmenovatelů ⟹ $\mathbb{N}$
- pro $c \in \mathbb{R}$ jak kdy, ale max. tok existuje

Def: Pro $A, B \subseteq V$: $E(A,B) := E \cap (A \times B)$



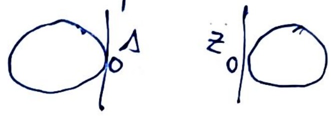
Def: Elementární řez je $E(A,B)$ pro $A \subseteq V$, $B = V \setminus A$, $z \in A$, $s \in B$

Def: $f(A,B) := \sum_{e \in E(A,B)} f(e) = f[Out(A)]$ ... tok z A do B

$C(A,B) := \sum_{e \in E(A,B)} c(e)$ ... kapacita řezu

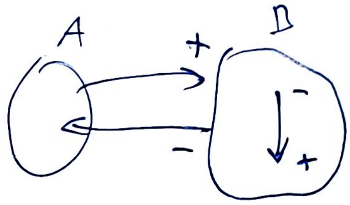
$f^{\Delta}(A,B) := f(A,B) - f(B,A)$ ... kolik toho reálně teče ke spotřebiči

Pokud je f tok a $E(A,B)$ řez, pak $f^{\Delta}(A,B) = |f|$.



$f(A,B) - f(B,A) = \sum_{v \in B} f^{\Delta}(v) = f^{\Delta}(s) = |f|$

↳ ty hrany co jsou uvnitř B jednou přičtou ⊕, podruhé ⊖

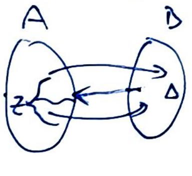


Důsledek: $|f| = f(A,B) - f(B,A) \leq C(A,B)$ ... pro $\forall$ tok a řez

Důsledek: Pokud $|f| = C(A,B)$ pak f je max. tok a $E(A,B)$ je min. řez

Lemma: Pokud se F.F. alg. zastaví, pak f je max. tok.

Dk: $A := \{v \in V \mid \exists$ nenasycená cesta $z \to v\}$



$B := V \setminus A$, $z \in A$, $s \notin A$ $\Rightarrow$ $E(A,B)$ je řez

hrany $A \to B$: $f = c$ } jinak rezerva > 0 $\Rightarrow$ nenasycená hrana ↯
hrany $B \to A$: $f = 0$ }

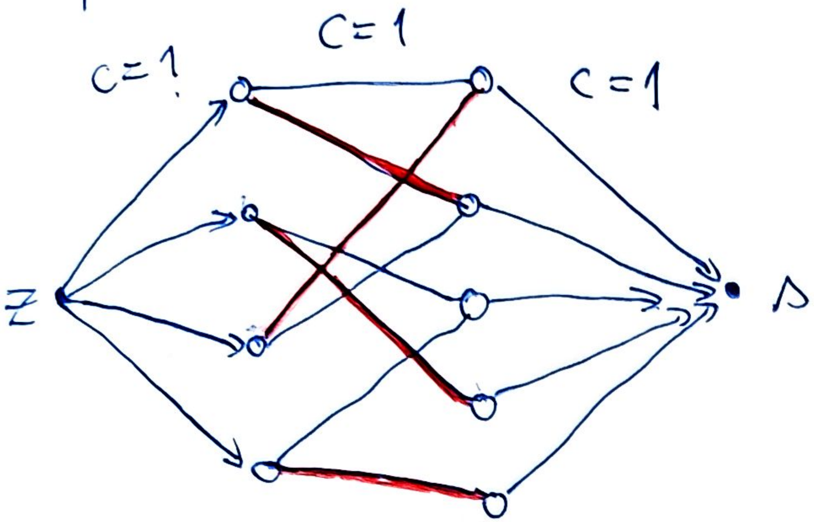
$\Rightarrow |f| = f(A,B) - f(B,A) = C(A,B) - 0$ $\Rightarrow$ f je max. a $E(A,B)$ je min. ■

Shrnutí

1. F.F. alg pro $c \in \mathbb{Q}$ skončí a najde max. tok (a min. řez)
2. $\forall f$ max. tok $\exists E(A,B)$ min. řez t.ž. $|f| = C(A,B)$
3. V síti s $c \in \mathbb{N}$ je alespoň 1 max. tok celočíselný a F.F. alg. ho najde
   ↳ protože zachovává celočíselnost

• Největší párování v bipartitním grafu

↳ párování je $F \subseteq E$ t.ž. $\forall e, f \in F : e \cap f = \emptyset$ ... neor. graf



→ najdeme maximální tok

⇒ bude 0-1 ... po ∀ hraně teče 0 nebo 1

z párování umím vyrobit 0-1 tok, jehož velikost = # hran párování

z 0-1 toku umím vyrobit párování

→ prostě vemu hrany kde něco teče

⇒ ∃ bijekce mezi 0-1 toky a párováními, navíc $|f|$ = # hran párování

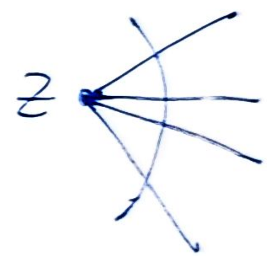
⇒ max 0-1 tok ~ největší párování

• Složitost F.F. alg. pro $c=1$

1) 1 průchod trvá $O(m)$ ... BFS
2) ∀ průchod zvětší $|f|$ alespoň o 1
3) # průchodů $\leq n$

$\Big\}\ O(n \cdot m)$

3)



řez s

$c \leq n$

• Dinitzův algoritmus

Def: Toku f přiřadíme průtok $f^*: E \to \mathbb{R}$, $f^*(uv) := f(uv) - f(vu)$.

1) $f^*(uv) = -f^*(vu)$
2) $f^*(uv) \le c(uv)$

$\Big\} \; -c(vu) \le f^*(uv)$

Kirchhoffův zákon

3) $\forall v \ne z,s$: $\underbrace{\sum_{uv \in E} f^*(uv) = \sum_{uv \in E} (f(uv) - f(vu)) = In - Out = f^\Delta(v) = 0}$

4) $r(uv) = c(uv) - f(uv) + f(vu) = c(uv) - f^*(uv)$

Lemma: Pokud fce $f^*: E \to \mathbb{R}$ splňuje 1) & 2) & 3), pak $\exists f$ jehož průtok je f^*.

Dk: $\forall uv, vu \in E$: BÚNO $f^*(uv) \ge 0$ ... z 1)

$f(uv) := f^*(uv)$ } tohle je tok ... je to nezáporné a omezené kapacitami ze 2)
$f(vu) := 0$ } Kirch. z. platí ze 3)

Def: Pro síť $S = (V,E,z,s,c)$ a tok f v ní je síť rezerv $R(S,f) := (V,E,z,s,\overset{r = c - f^*}{r})$.

Lemma: Pro $\forall$ tok f v S a $\forall$ tok g v $R(S,f)$
$\exists$ tok h v S t.ž. $|h| = |f| + |g|$ a navíc lze h najít v čase $O(m)$

Dk: $h^* := f^* + g^*$ a z h^* umíme sestrojit h v $O(m)$

↳ chceme ukázat, že h^* je průtok a sečtou se velikosti

→ h^* zjevně splňuje 1
→ $h^*(uv) = f^*(uv) + g^*(uv) \le f^*(uv) + \overset{c - f^*}{\overset{\parallel}{r(uv)}} = c(uv)$ ... 2
→ 3 taky protože pro $\forall v \ne z,s$ máme $0 + 0 = 0$ ✓

} h^* je průtok

→ $|h| = |f| + |g|$ protože $|f| =$ přebytek spotřebiče a ty přebytky se sčítají

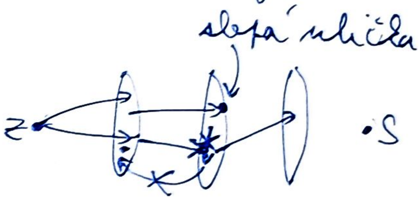
Def: Tok f je blokující $\equiv \forall P$ cestu $z \to s$ $\exists e \in P: f(e) = c(e)$

Def: Síť S je vrstevnatá (pročištěná)
$\equiv \forall$ vrchol i hrana leží na nějaké nejkratší cestě $z \to s$.

• <u>Dinitzův alg.</u> $O(n^2 m)$

1. $f \leftarrow$ všude nulový tok (nebo klidně i nějaký lepší)
2. Opakujeme: $\# \text{fází} \leq n$
3. $R \leftarrow$ síť rezerv vzhledem k S $O(m)$
4. Smažeme z R nulové hrany $O(m)$
5. Pročistíme R = necháme tam jenom vrcholy a hrany na nejkratších cestách $O(m)$
6. Pokud je R prázdná, tak končíme
7. $g \leftarrow$ blokující tok v R $O(nm)$
8. Vylepšíme f pomocí g $O(m)$

• <u>Korektnost</u>: Pokud se zastaví tak $\nexists$ cesta $z \to s$ v $R \Rightarrow \nexists$ nenasycená v S $\Rightarrow$ z korektnosti F.F. alg. je korektní i tohle



• <u>Čištění sítě</u> $O(m)$

1. Pomocí DFS(z) rozdělíme síť na vrstvy $O(m)$
2. Smažeme vrstvy za s
3. Smažeme všechny hrany co nevedou o vrstvu dopředu
4. Pořídíme si frontu na slepé uličky a všechny je smažeme dokud v F něco je

• <u>Hledání blokujícího toku ve vrstevnaté síti</u> $O(m \cdot n)$

1. $g \leftarrow 0$ ↳ tohle jde i algoritmem 3 indů $O(n^2)$
2. Dokud $\exists P$ cesta $z \to s$ v R:
3. $\varepsilon \leftarrow \min_{e \in P}(r(e) - g(e))$ ... $r(e)$ je f-rezerva = g-kapacita
4. $\forall e \in P$: $g(e) \leftarrow g(e) + \varepsilon$ a pokud $g(e) = r(e)$, hrana e smažu z R
5. Dočistíme nově vzniklé slepé uličky - mohou vzniknout při mazání e

- 1 iterace kromě (5) trvá $O(n) = O(\#\text{vrstev})$ & $\#$ iterací $\leq m$
  ↳ tu cestu hledáme metodou rovnou za nosem
- (5) celkově poběží $O(m+n)$ = smazání všech vrcholů a hran } $O(m \cdot n)$

<u>Lemma</u>: Mezi dvěma sousedními fázemi vzroste $\#$vrstev R alespoň o 1.

<u>Dk</u>: Odstraním všechny nejkratší cesty délky l, takže tam teď nejkratší bude $l+1$. Ale mohla mi tam vzniknout zpětná hrana. To ale nevadí $\because$ vede dozadu $\Rightarrow \forall$ cesta co jí použije má délku alespoň $l+2$ ■

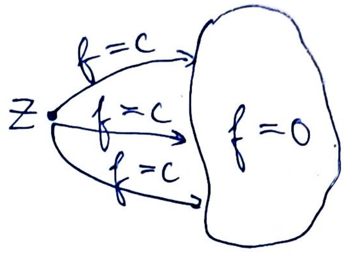
<u>Důsledek</u>: $\#$ fází $\leq n$ $\Rightarrow$ Dinitzův algoritmus běží v čase $O(n^2 m)$

↳ reálně to je $\Theta(n^2 m)$

• Goldbergův algoritmus

Def: $f: E \to \mathbb{R}_0^+$ je vlna $\equiv$ $\forall e \in E: f(e) \leq c(e)$

$\forall v \neq z, s: f^\Delta(v) \geq 0$



→ tohle je vlna

• Převedení přebytku

$u \to v$

$f^\Delta(u) > 0$, $r(uv) > 0$ $\Rightarrow$ f' zůstane vlnou

$\delta := \min(f^\Delta(u), r(uv))$ $\quad f^\Delta(u) -= \delta$, $r(uv) -= \delta$

$f'(uv) := f(uv) + \delta$ $\quad f^\Delta(v) += \delta$, $r(vu) += \delta$

Def: Výška $h: V \to \mathbb{N}$ ... a převádíme jen z kopce

Algoritmus

1. $f(*) \leftarrow 0$, $\forall zv \in E: f(zv) \leftarrow c(zv)$
2. $h(*) \leftarrow 0$, $h(z) \leftarrow n$

] $O(n)$

3. Dokud $\exists u \neq z, s: f^\Delta(u) > 0$:
4.   Pokud $\exists uv \in E: r(uv) > 0$ & $h(u) > h(v)$:
           převedeme po uv
5.   Jinak $h(u) \leftarrow h(u) + 1$

• Analýza algoritmu

Inv A (základní)

1) f je vlna $\Leftarrow$ 4)
2) $\forall v$: $h(v)$ neklesá
3) $h(z) = n$, $h(s) = 0$ ... v tom cyklu vylučujeme z a s
4) $\forall v \neq z$: $f^\Delta(v) \geq 0$ ... přebytky se mění převedením

Inv S (o spádu) → vždy převádíme přesně o 1 patro níž

Def: Spád hrany uv je $h(u) - h(v)$ ... o kolik dolů ta hrana vede

Inv: $\nexists uv \in E: r(uv) > 0$ & $h(u) > h(v) + 1$

Dk: Po inicializaci platí – ty hrany už mají + rezervu ale vedou do kopce.

a, nenasycená hrana se spádem 1 a zvedl jsem ji ↯ 4

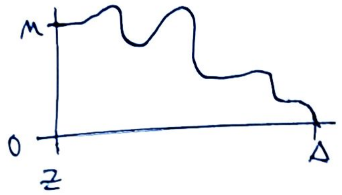
b, nasycená se spádem > 1 a stala se nenasycenou ⇒ vzrostla jí rezerva

↳ rezerva roste při převádění té zpětné hraně, ale ta je do kopce ↯

Lemma K (Korektnost): Pokud se algoritmus zastaví, tak f je max. tok.

Dk: ① f je tok $\because$ se zastavil ten cyklus $\Rightarrow \forall v: f^\Delta(v)=0$

② f je max $\because$ kdyby ne, tok podle F. F. $\exists P$ nenasycená cesta $z \to s$



$\Rightarrow$ ta cesta překonává spád n & má nejvýše $n-1$ hran

$\Rightarrow \exists e \in P$: spád$(e) \geq 2$ & $r(e) > 0$ ↯ Inv S

Inv C (cesta do zdroje): $\forall v: f^\Delta(v) > 0$ $\exists P$ nenasycená cesta $v \to z$.

$\Rightarrow$ **Inv V (o výšce):** $\forall v: h(v) \leq 2n$.

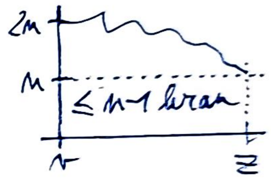
$\Rightarrow$ **Lemma Z (o zvednutí):** # zvednutí $\leq 2n^2$.

Dk: n vrcholů, $\forall$ zvednu max. $2n$-krát ∎

Dk: Uvažme první porušení … to muselo nastat zvednutím

$\Rightarrow$ zvedáme $v \in V$ z výšky $2n$, tehdy $f^\Delta(v) > 0$

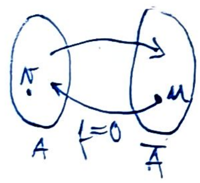
$\Rightarrow$ podle Inv C $\exists P$ nenasycená cesta $v \to z$ ↯ Inv S



Dk: $A := \{u \in V \mid \exists$ nenasycená cesta $v \to u\}$, ukážeme $z \in A$.

$$\sum_{a \in A} f^\Delta(a) = \sum_{a \in A}\left(f^{In}(a) - f^{Out}(a)\right) = \underbrace{f[In(A)]}_{0} - f\underbrace{[Out(A)]}_{\geq 0} \leq 0$$

↳ ty hrany uvnitř přispějí jednou ⊕ a ⊖ ↳ jinak zpětná hrana má rezervu $> 0 \Rightarrow u \in A$ ↯



$\Rightarrow$ ta Σ obsahuje $f^\Delta(v) > 0$, ale je nekladná

$\Rightarrow$ obsahuje záporný člen, jediný takový je $z \Rightarrow z \in A$. ∎

• Kolik bude převedení?

Def: Převedení je nasycené $\equiv$ vynuluje rezervu hrany.

Nenasycené převedení po hraně uv vynuluje $f^\Delta(u)$.

Lemma S (sytá převedení): # nasycených převedení $\leq n \cdot m$.

Dk: Uvažme hranu uv … po sytém převedení $r(uv)=0$. Abychom mohli sytě převést znovu, tak potřebujeme $r(uv) > 0$. Na to musíme převést po hraně vu, ale uv má nyní spád 1, takže v musím zvednout o 2. uv má nyní spád -1, abych mohl převádět, tak musím u zvednout o 2.

$\Rightarrow$ podle Inv V tohle nejvýše n-krát & mám m hran $\Rightarrow n \cdot m$ ∎

Lemma H (hladová převedení) ... nenasytí hranu

Def: Potenciál $\phi := \sum_{\substack{v \neq z,s \\ f^\Delta(v)>0}} h(v)$

① $\phi \geq 0$

② Na počátku $\phi = 0$

③ Zvednutí ~ $\phi += 1$

④ SP... možná $+h(v) - h(u)$ ... $\phi +=$ max. $2n$ (může se i snížit)

⑤ HP... určitě $-h(u)$, možná $+h(v)$ ... $\phi -=$ alespoň 1 $(h(v)-h(u)=1)$

④ 

výšky se nemění

možná $f^\Delta(u)=0 \Rightarrow -h(u)$

možná předtím $f^\Delta(v)=0 \Rightarrow +h(v)$

Důsledek: $\phi \leq 2n^2\cdot 1 + n\cdot m\cdot 2n = 2n^2(m+1) \Rightarrow \# HP \leq 2n^2(m+1) \in O(n^2 m)$

$\Rightarrow$ Goldberg je konečný

Časová složitost Goldberga

→ zvedání a převádění umíme rychle

→ chceme umět rychle najít ten vrchol s přebytkem

① $S :=$ seznam vrcholů s přebytkem

↳ údržba při změně $f^\Delta(v)$ a výběr v kroku ③ : $O(1)$

② Pro $\forall u \in V$: $K(u) := \{uv \mid r(uv) > 0 \ \&\ h(u) > h(v)\}$

↳ výběr ve kroku ④ : $O(1)$

→ údržba při převedení: snížení $r(uv)$ a zvýšení $r(vu)$ $O(1)$

↳ ale vu vede do kopce → není v žádném $K(*)$

→ údržba při zvednutí: musím probrat všechny hrany uv a vu ... $O(n)$

↳ ale zvednutí je jen $O(n^2)$

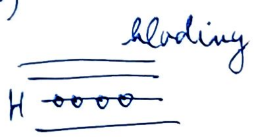
$$\Rightarrow \text{Celkem } O(\underbrace{2n^2\cdot n}_{\text{zvednutí}} + \underbrace{nm\cdot 1}_{\text{SP}} + \underbrace{n^2 m}_{\text{HP}}) = O(n^2 m)$$

Vylepšení Goldberga

→ nejvíce mě brzdí HP, chci jich míň

Lemma H*: Když vybíráme nejvyšší vrchol s $f^\Delta(v) > 0$: $\# HP \in O(n^3)$

Dk: $H := \max\{h(v) \mid f^\Delta(v) > 0,\ v \neq z\}$



→ běh algoritmu rozdělíme na fáze → fáze končí změnou H

1) konec fáze: zvýšení H ... $\leq 2n^2$-krát, vždy právě o 1

2) H klesne aspoň o 1 ... ale $H \geq 0 \Rightarrow \leq 2n^2$-krát

} # fází $\in O(n^2)$

Během 1 fáze děláme z $\forall$ vrcholu nejvýše 1 HP

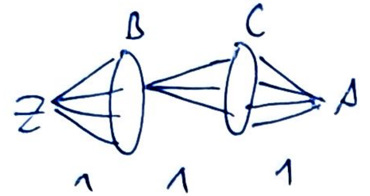
↳ HP $\Rightarrow f^\Delta(v) = 0$, aby další HP, tak jsme do něj museli něco převést, ale H je max. ↯

$\Rightarrow$ # HP ve fázi $\leq n \Rightarrow$ # HP celkem $\in O(n^3) \Rightarrow$ Goldberg $O(n^3)$

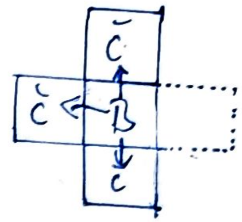
→ ∃ lepší potenciál co dává $\Theta(n^2\sqrt{m})$

Aplikace toků v sítích

1) Děravá šachovnice, chceme ji pokrýt dominem.



→ hrany vedou ze všech bílých políček do jejich černých sousedů

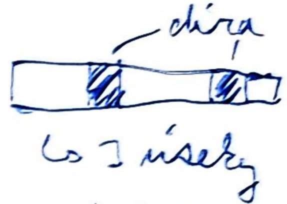


→ najdu největší párování

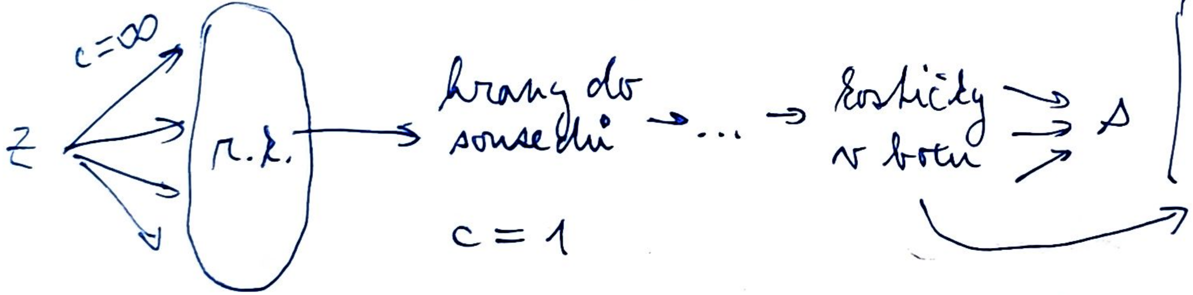
2) Děravá šachovnice, chceme rozmístit co nejvíc věží, aby se navzájem neohrožovaly

↳ věže nemohou přes díry

⇒ najdeme největší párování úseků sloupců a řádků



3) Radioaktivní kostičky v minecraftu. Chceme do světa rozmístit destičky, co uzavřou tzv. r.k. ⇒ ∄ cesta r.k. → ∞ co neprojde destičkou.



všechny tzv. r.k. lze ohraničit nějakým čtvercem

↳ s ~ nekonečno

⇒ max. tok ⇒ min. řez - hrany toho řezu ~ umístění destiček.

- Fourierova transformace

- Polynomy $P(x) := \sum_{k=0}^{n-1} p_k x^k$, $\vec{p} = (p_0, \dots p_{n-1})$, n = velikost polynomu

↳ normalizace: $p_{n-1} \neq 0$ nebo $n = 0$ $\quad (1,0,3,0) \to (1,0,3)$

$\quad$ ↳ stupeň polynomu $\quad$ ↳ stupeň 2

$\Rightarrow$ stupeň (1) je 0, stupeň $()$ je -1

- Násobení polynomů ... BÚNO P, Q stejně velké

$R = P \cdot Q$

$$R(x) = \sum_{j=0}^{n-1} p_j x^j \sum_{k=0}^{n-1} q_k x^k = \sum_{j,k=0}^{n-1} p_j q_k x^{j+k} = \sum_{s=0}^{2n-2} \left(\sum_{j=0}^{s} p_j q_{s-j} \right) x^s \Rightarrow \Theta(n^2)$$

- Rovnost polynomů

1, identita $P \equiv Q$: stejné vektory po normalizaci

2, rovnost fcí : $\forall x: P(x) = Q(x)$

$1 \Downarrow 2$ $\quad$ $1 \Uparrow 2$ dokážeme

Věta: Nechť P, Q jsou polynomy stupně max. d a $P(x_j) = Q(x_j)$ pro navzájem různá čísla $x_0, \dots x_d$. Potom $P \equiv Q$.

Lemma: Pro polynom P stupně $d \geq 0$: $(\# x : P(x) = 0) \leq d$.

Dk: $R := P - Q$

$\forall j: R(x_j) = P(x_j) - Q(x_j) = 0 \Rightarrow d+1$ kořenů & stupeň $R \leq d$

$\Rightarrow$ lemma neplatí $\Rightarrow$ nesplnili jsme jeho předpoklad $\Rightarrow R \equiv 0 \Rightarrow P \equiv Q$. ■

Def: Graf polynomu P velikosti n pro pevně zvolená $x_0, \dots x_{n-1}$ je $(P(x_0), \dots P(x_{n-1}))$.

👁 Polynom je grafem jednoznačně určen.

👁 Pokud $R = P \cdot Q$, potom $R(x) = P(x) \cdot Q(x)$

Plán: Zvolíme nějak $x_0 \dots x_{n-1}$

$P \longrightarrow$ graf(P)

$Q \longrightarrow$ graf(Q)

součin po složkách = graf(R)

$R \longleftarrow$

Problém: stupeň R může být $> n$

$\Rightarrow$ BÚNO horních $n/2$ koef P a Q jsou nuly

Problém: Převod $P \to$ graf(P) a zpět neumíme rychle

• **Pokus o vyhodnocení polynomu metodou rozděl a panuj**

P velikosti $n = 2^k$ ⇒ vyhodnocujeme v $x_0, \dots x_{n-1}$, párování $x_{\frac{n}{2}+j} = -x_j$

$Q(x) = x^{2m} = Q(-x)$ ⇒ když znám $Q(x)$, mám i $Q(-x)$

☼ $Q(x) = x^{2m+1} = -Q(-x)$

$$P(x) = p_0 x^0 + p_1 x^1 + p_2 x^2 + \dots + p_{n-1} x^{n-1}$$
$$= (p_0 x^0 + p_2 x^2 + \dots p_{n-2} x^{n-2}) = S(x^2)$$
$$+ (p_1 x + p_3 x^3 + \dots p_{n-1} x^{n-1}) = x\,L(x^2)$$

$$\left.\begin{aligned} P(x) &= S(x^2) + x\,L(x^2) \\ P(-x) &= S(x^2) - x\,L(x^2) \end{aligned}\right\}$$

Vyhodnocení velikosti n v n bodech → 2x velikost $n/2$ v $n/2$ bodech

$$T(n) = 2 \cdot T(n/2) + \Theta(n) = \Theta(n \log n)$$

→ ale potřebujeme aby to párování vydrželo i v tom podproblému

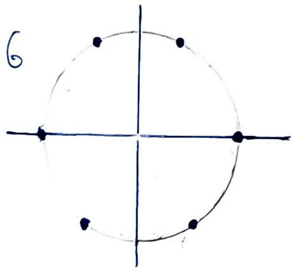
⇒ buď počítáme v nějakém chytrém tělese nebo komplexní čísla

• **Komplexní odmocniny**

$x = \sqrt[n]{1} \Rightarrow |x| = 1 \Rightarrow x = e^{i\varphi}$ pro nějaké φ

$x^n = e^{i\varphi n} = \cos(\varphi n) + i \sin(\varphi n) = 1 \Rightarrow \varphi n = k \cdot 2\pi \Rightarrow \varphi = \frac{2k\pi}{n}$, $k = 0, \dots n-1$

$n = 6$



→ n-úhelník s vrcholem v 1 ⇒ n různých odmocnin

pro $|x| = 1$: $x^{-1} = e^{-i\varphi} = \overline{x}$

Def: $\omega \in \mathbb{C}$ je primitivní n-tá odmocnina z 1 $\equiv \omega^n = 1$ a $\omega^1, \dots \omega^{n-1} \neq 1$.

☼ $\omega^j \neq \omega^k$ pro $0 \leq j < k < n$.

Kdyby $\omega^j = \omega^k$, tak $\frac{\omega^k}{\omega^j} = 1 \Rightarrow \omega^{k-j} = 1$, ale $1 \leq k-j < n$ ↯

☼ pro sudé n: $\omega^{n/2} = -1$, protože $\omega^{n/2} = \sqrt{\omega^n}$ & $\omega^{n/2} \neq 1$.

⇒ $\omega^{n/2+k} = -\omega^k$

ω	0	1	...	$n/2-1$	$n/2$	...	$n-1$
	$=1$	$=\omega$		$=\omega^{n/2-1}$	$=-1$	$=-\omega$	$=-\omega^{n/2-1}$

druhá půlka = − první půlka

⇒ je to spárované

☼ pro $\forall n$ $\exists$ primitivní n-tá odmocnina 1

→ např. $\omega = e^{\frac{2\pi}{n} i}$, $e^{-\frac{2\pi}{n} i}$

→ podposloupnost

☼ ω^2 je primitivní $n/2$-tá odmocnina z 1 … ∵ $\omega^2, \omega^4, \dots \omega^{n-2} \neq 1$

⇒ i $\omega^0, \omega^2, \omega^4, \dots \omega^{n-2}$ je dobře spárovaná posloupnost

• **Algoritmus FFT** → převod polynomu na graf

Vstup: $n = 2^k$, ω = primitivní n-tá odmocnina z 1

$(p_0, \dots p_{n-1})$ koeficienty polynomu P.

Výstup: $(y_0, \dots y_{n-1})$ graf(P) v bodech $(1, \omega, \omega^2, \dots \omega^{n-1})$ $\Theta(n \log n)$

0. Pokud $n = 1$: $y_0 = p_0$ a končíme

1. $(s_0, \dots s_{n/2-1}) \leftarrow FFT(n/2, \omega^2, (p_0, p_2, \dots p_{n-2}))$
2. $(l_0, \dots l_{n/2-1}) \leftarrow FFT(n/2, \omega^2, (p_1, p_3, \dots p_{n-1}))$

} 2× rekurze

$\Theta(n)$ 3. Pro $j = 0 \dots n/2 - 1$:

$$y_j \leftarrow s_j + \omega^j l_j \quad = S(x_j^2) + x_j L(x_j^2) = P(x_j) \quad \text{kde } x_j = \omega^j$$

$$y_{n/2+j} \leftarrow s_j - \omega^j l_j = S(x_j^2) - x_j L(x_j^2) = P(-x_j) \quad x_j^2 = (\omega^2)^j$$

Def: Diskrétní Fourierova transformace (DFT) je lineární zobrazení

$$\mathcal{F}: \mathbb{C}^n \to \mathbb{C}^n, \ \vec{y} = \mathcal{F}(\vec{x}) \equiv \forall j \ y_j = \sum_{k=0}^{n-1} x_k \omega^{jk} = X(\omega^j) \dots$$ vyhodnocení polynomu X

$\hookrightarrow \omega = \text{pr. } \sqrt[n]{1}$

$\Rightarrow$ FFT počítá DFT ale fast

👁 $\mathcal{F}$ je l.z. $\Rightarrow \vec{y} = \Omega \vec{x}$, kde $\Omega \in \mathbb{C}^{n \times n}$, $\Omega_{jk} = \omega^{jk}$.

👁 Když chci z $\vec{x}$ zpátky $\vec{y}$, tak potřebuju $\vec{x} = \Omega^{-1} \vec{y}$

Tvrzení: $\Omega \cdot \overline{\Omega} = n I_n \Rightarrow \Omega^{-1} = \frac{1}{n} \overline{\Omega}$.

Dk: $(\Omega \overline{\Omega})_{jk} = \sum_l \Omega_{jl} \cdot \overline{\Omega}_{lk} = \sum_l \omega^{jl} \cdot \overline{\omega}^{lk} = \sum_l \omega^{jl} \cdot \omega^{-lk} = \sum_{l=0}^{n-1} (\omega^{j-k})^l$

1) $j \neq k$: $\omega^{j-k} \neq 1 \Rightarrow (\Omega \overline{\Omega})_{jk} = \dfrac{(\omega^{j-k})^n - 1}{\omega^{j-k} - 1} = \dfrac{0}{\neq 0} = 0$.

2) $j = k$: $\omega^{j-k} = 1 \Rightarrow (\Omega \overline{\Omega})_{jk} = n \cdot 1 = n$ ■

Důsledek: Inverzní DFT můžu spočítat pomocí FFT, kde za ω zvolím $\overline{\omega}$.
Pak to ale ještě musím vynásobit $1/n$.

Důsledek: Polynomy lze násobit pomocí FFT v čase $\Theta(n \log n)$.
$\hookrightarrow$ graf → polynom pomocí inverzní FFT

Důsledek linearity DFT:

$$\mathcal{F}(\vec{x} + \vec{y}) = \Omega(\vec{x} + \vec{y}) = \mathcal{F}(\vec{x}) + \mathcal{F}(\vec{y})$$

$$\mathcal{F}(k \cdot \vec{x}) = \Omega(k \cdot \vec{x}) = k \cdot \mathcal{F}(\vec{x})$$

Jiný pohled na DFT

Problém: Máme nějakou funkci F a chceme ji vyjádřit pomocí sinů a kosinů.

Stačí nám to v nějakém intervalu ⇒ navzorkujeme si ji v $n=2^k$ bodech.

⇒ chceme najít $\alpha_1, \dots \alpha_{n/2-1}$, $\beta_1, \dots \beta_{n/2}$ a γ tž.

$$F(x) = \sum_{k=1}^{n/2-1} \alpha_k \cdot \sin(2\pi k x) + \sum_{k=1}^{n/2} \beta_k \cdot \cos(2\pi k x) + \gamma \quad ; \quad x = 0, \tfrac{1}{n}, \tfrac{2}{n}, \dots \tfrac{n-1}{n}$$

pro $n=4$: $F(x) = \alpha_1 \sin(2\pi x) + \beta_1 \cos(2\pi x) + \beta_2 \cos(4\pi x) + \gamma$

💡 $k = \frac{n}{2}$: $\sin(2\pi \frac{n}{2} \cdot \frac{l}{n}) = \sin(\pi l) = 0$ ⇒ místo $\alpha_{n/2} \cdot \sin(\dots)$ máme konstantu γ

⇒ teď ten vektor $(x_0, \dots x_{n-1})$ transformuju pomocí DFT → $(y_0, \dots y_{n-1})$

↳ reálné a imaginární složky těch y_i mi dávají ty koeficienty α a β

⇒ zjistíme DFT navzorkovaného sinu a kosinu o různých frekvencích

⇒ díky linearitě DFT víme jak bude vypadat DFT nějakého složení těchto sinů a kosinů

⇒ také víme, že každý reálný vektor lze zapsat jako → z lingebry
součet sinů a kosinů

⇒ každý reálný vektor můžu zfurierovat a koukat se na to, co mi vyjde jako na nějakou l.k. těch zfurierovaných sinů a kosinů

⇒ to mi dá, z jakých sinů a kosinů se skládal ten původní vektor

Věta: Nechť $\vec{x} \in \mathbb{R}^n$ a $\vec{y} = \mathcal{F}(\vec{x})$. Potom $\forall j$: $y_j = \overline{y_{n-j}}$.

Dk:

$$y_j = \sum_{k=0}^{n-1} x_k \omega^{jk}$$

$$y_{n-j} = \sum_{k=0}^{n-1} x_k \omega^{(n-j)k} = \sum_{k=0}^{n-1} x_k \cdot \overset{1}{\overset{\parallel}{\omega^{n}}} \cdot \omega^{-jk} = \sum_{k=0}^{n-1} \overline{x_k} \cdot \overline{\omega^{jk}} = \overline{y_j} \quad \blacksquare$$

Tvrzení: Každý reálný vektor lze zapsat jako l.k. navzorkovaných sinů a kosinů.

$Re(y_j)$ = koef. u $\cos(jx)$ pro $j = 1, \dots n/2$

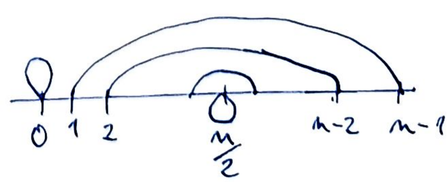
$Im(y_j)$ = koef u $\sin(jx)$ pro $j = 1, \dots n/2 - 1$

$Re(y_0)$ = aditivní konstanta $(\cos(0x) = 1)$

$Im(y_0) = 0$ $(\sin(0x) = 0)$

$Im(y_{n/2}) = 0$

Symetrie:



• <u>Příklad</u>

$P(x) = x^4 + x^2 - 3x + 1 \quad \dots \quad \vec{p} = (1,-3,1,0,1,0,0,0)$

$Q(x) = x^3 + 2x^2 - 1 \quad \dots \quad \vec{q} = (-1,0,2,1,0,0,0,0)$

$R(x) = P(x)Q(x) = x^7 + 2x^6 + x^5 - 2x^4 - 5x^3 + x^2 + 3x - 1 \quad \dots \quad \vec{r} = (-1,3,1,-5,-2,1,2,1)$

stupeň R je 7 $\Rightarrow$ stačí mi $n = 8$ bodů, $\omega = e^{\frac{2\pi}{8}i} = e^{\frac{\pi}{4}i}$ 

$$\ast \qquad \Omega = (\omega^{ij}) = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}\begin{array}{c} 0 \\ \left[\begin{array}{cccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 & -1 & -\omega & -\omega^2 & -\omega^3 \\ 1 & \omega^2 & -1 & -\omega^2 & 1 & \omega^2 & -1 & -\omega^2 \\ 1 & \omega^3 & -\omega^2 & \omega & -1 & -\omega^3 & \omega^2 & -\omega \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -\omega & \omega^2 & -\omega^3 & -1 & \omega & -\omega^2 & \omega^3 \\ 1 & -\omega^2 & -1 & \omega^2 & 1 & -\omega^2 & -1 & \omega^2 \\ 1 & -\omega^3 & -\omega^2 & -\omega & -1 & \omega^3 & \omega^2 & \omega \end{array}\right]\end{array} \cdot \begin{bmatrix} 1 & -1 \\ -3 & 0 \\ 1 & 2 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2 \\ -3\omega + \omega^2 & -1 + 2\omega^2 + \omega^3 \\ 1 - 3\omega^2 & -3 - \omega^2 \\ -3\omega^3 - \omega^2 & -1 - 2\omega^2 + \omega \\ 6 & 0 \\ 3\omega + \omega^2 & -1 + 2\omega^2 - \omega^3 \\ 1 + 3\omega^2 & -3 + \omega^2 \\ 3\omega^3 - \omega^2 & -1 - 2\omega^2 - \omega \end{bmatrix}$$

$$\Omega = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \dots & \omega^7 \\ 1 & \omega^2 & \omega^4 & & \dots & \omega^{14} \\ \vdots \\ 1 & \omega^7 & \omega^{14} & & \dots & \omega^{49} \end{bmatrix} \quad \& \quad \begin{array}{l} \omega^{4+k} = \omega^4 \cdot \omega^k = -\omega^k \\ \omega^8 = 1, \ \overline{\omega} = \omega^{-1} \end{array} \quad \Bigg| \quad \Omega \cdot \vec{p} = (P(1), P(\omega), \dots P(\omega^7))$$

$\Rightarrow$ teď mám graf(P) a graf(Q) $\Rightarrow$ vynásobím je pro graf(R)

↳ dělat tohle nad $\mathbb{C}$ je nechutný $\Rightarrow$ $\omega = 2 \bmod 17$ je primitivní $\sqrt[8]{1}$

$$\Omega \cdot [\vec{p}\ \vec{q}] = \begin{bmatrix} 0 & 2 \\ -2 & -2 \\ 6 & -7 \\ 6 & -7 \\ 6 & 0 \\ 10 & -1 \\ 13 & 1 \\ 3 & 6 \end{bmatrix} \Rightarrow \text{graf}(R) = \Omega\vec{p} \cdot \Omega\vec{q} = \begin{bmatrix} 0 \\ 4 \\ 9 \\ 9 \\ 0 \\ 7 \\ 13 \\ 1 \end{bmatrix} \qquad \begin{array}{l} \overline{\omega} = \omega^{-1} \\ \omega^{-1} = -\omega^3 \\ \omega^{-2} = -\omega^2 \\ \omega^{-3} = -\omega \end{array}$$

$$\overline{\Omega} = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -\omega^3 & -\omega^2 & -\omega & -1 & \omega^3 & \omega^2 & \omega \\ 1 & -\omega^2 & -1 & \omega^2 & 1 & -\omega^2 & -1 & \omega^2 \\ 1 & -\omega & \omega^2 & -\omega^3 & -1 & \omega & -\omega^2 & \omega^3 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & \omega^3 & -\omega^2 & \omega & -1 & -\omega^3 & \omega^2 & -\omega \\ 1 & \omega^2 & -1 & -\omega^2 & 1 & \omega^2 & -1 & -\omega^2 \\ 1 & \omega & \omega^2 & \omega^3 & -1 & -\omega & -\omega^2 & -\omega^3 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 4 \\ 9 \\ 9 \\ 0 \\ 7 \\ -4 \\ 1 \end{bmatrix} = \begin{bmatrix} 4+1+7-3 \\ 3\omega^3 - 13\omega^2 - 8\omega \\ -\omega^2 - 5 \\ 3\omega + 13\omega^2 - 8\omega^3 \\ -4-7-5 \\ -3\omega^3 - 13\omega^2 + 8\omega \\ \omega^2 - 5 \\ -3\omega + 13\omega^2 + 8\omega^3 \end{bmatrix} = \begin{bmatrix} 9 \\ 7+16-16 \\ -4-5 \\ 6-16+4 \\ 1 \\ -7+16+16 \\ 4-5 \\ -6-16-4 \end{bmatrix} = \begin{bmatrix} -8 \\ 7 \\ 8 \\ -6 \\ -16 \\ 8 \\ 16 \\ 8 \end{bmatrix}$$

$\vec{r} = \frac{1}{n}\overline{\Omega} = (-1, 3, 1, -5, -2, 1, 2, 1)$ ✓ $\quad \Rightarrow$ potřeboval jsem nějakou dostatečně velkou bázi toho tělesa aby to nepřeteklo

↳ $\frac{1}{n} = 8^{-1} = 15 = -2$

- **Nyní pomocí FFT** — zase $\omega = 2 \bmod 17$

sudé:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^2 & -1 & -\omega^2 \\ 1 & -1 & 1 & -1 \\ 1 & -\omega^2 & -1 & \omega^2 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 1 & 2 \\ 1 & 0 \\ 0 & 0 \end{bmatrix}$$

S: $\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 & -1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ 0 & -1 \end{bmatrix}$

L: $\begin{bmatrix} 1 & 1 \\ \omega^2 & -\omega^2 \end{bmatrix}\begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & \\ & \omega^2 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & \\ & \omega^2 \end{bmatrix}\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 4 & 8 \end{bmatrix}$

↳ 1. půlka: [ ] + [ ]
2. půlka: [ ] − [ ]

$$\Rightarrow \begin{bmatrix} 2+1 & -1+2 \\ 0+4 & -1+8 \\ 2-1 & -1-2 \\ 0-4 & -1-8 \end{bmatrix} = \begin{bmatrix} 3 & 1 \\ 4 & 7 \\ 1 & -3 \\ -4 & 8 \end{bmatrix}$$

liché:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^2 & -1 & -\omega^2 \\ 1 & -1 & 1 & -1 \\ 1 & -\omega^2 & -1 & \omega^2 \end{bmatrix} \begin{bmatrix} -3 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} -3 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} -3 & 0 \\ -3 & 0 \end{bmatrix}$

$\begin{bmatrix} 1 & 1 \\ \omega^2 & -\omega^2 \end{bmatrix}\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 4 \end{bmatrix}$

$$\Rightarrow \begin{bmatrix} 1 & & & \\ & \omega & & \\ & & \omega^2 & \\ & & & \omega^3 \end{bmatrix}\begin{bmatrix} -3 & 1 \\ -3 & 4 \\ -3 & -1 \\ -3 & -4 \end{bmatrix} = \begin{bmatrix} -3 & 1 \\ -6 & 8 \\ 5 & -4 \\ -7 & 2 \end{bmatrix}$$

→ 1. půlka [ ] + [ ]
2. půlka [ ] − [ ]

$$\Rightarrow \begin{bmatrix} 3-3 & 1+1 \\ 4-6 & 7+8 \\ 1+5 & -3-4 \\ -4-7 & 8+2 \\ 3+3 & 1-1 \\ 4+6 & 7-8 \\ 1-5 & -3+4 \\ -4+7 & 8-2 \end{bmatrix} = \begin{bmatrix} 0 & 2 \\ -2 & -2 \\ 6 & -7 \\ 6 & 10 \\ 6 & 0 \\ 10 & -1 \\ -4 & 1 \\ 3 & 6 \end{bmatrix} \rightarrow \text{graf}(R) = \begin{bmatrix} 0 \\ 4 \\ 9 \\ 9 \\ 0 \\ 7 \\ -4 \\ 1 \end{bmatrix} \checkmark$$

- **Motýlkový algoritmus** = nerekurzivní FFT

→ indexy se seřadí podle binárky pozpátku

000	1	1	ω^0	2		$2+1 = 3$		$3-3 = 0$
001	-3	1		0	1	$0+\omega^2 = 4$		$4-3\omega = -2$
010	1	1	ω^0	1		$2-1 = 1$		$1-3\cdot\omega^2 = 6$
011	0	0		1	ω^2	$0-\omega^2 = -4$		$-4-3\omega^3 = 6$
100	1	-3	ω^0	-3		$-3+0 = -3$	1	$3+3 = 6$
101	0	0		-3		$-3+0 = -3$	ω	$4+3\omega = 10$
110	0	0	ω^0	0	1	$-3-0 = -3$	ω^2	$1+3\cdot\omega^2 = 13$
111	0	0		0	ω^2	$-3-0 = -3$	ω^3	$-4+3\cdot\omega^3 = 3$

→ motýlek nazpátek: $\omega^{-1}\cdot\omega = 1 \Rightarrow 9\cdot 2 = 1 \Rightarrow \omega^{-1} = 9$ … odteď $\omega = 9$, $\omega^2 = -4$, $\omega^3 = -2$

000	0	0	$0+0 = 0$		$0+5 = 5$		$5+4 = 9$	$\cdot\frac{1}{n}$	-1
001	4	0	$0-0 = 0$		$0-4\omega^2 = -1$		$-1-\omega = 7$		3
010	9	9	$9-4 = 5$	1	$0-5 = -5$		$-5+\omega^2 = -9$		1
011	9	-4	$9+4 = -4$	ω^2	$0+4\omega^2 = 1$		$1-5\omega^3 = 11$		-5
100	0	4	$4+7 = -6$		$-6-7 = 4$		$5-4 = 1$	→	-2
101	7	7	$4-7 = -3$		$-3+8\omega^2 = -1$	ω	$-1+\omega = 8$		1
110	-4	9	$9+1 = -7$	1	$-6+7 = 1$	ω^2	$-5-\omega^2 = -1$		2
111	1	1	$9-1 = 8$	ω^2	$-3-8\omega^2 = -5$	ω^3	$1+5\omega^3 = -9$	$\cdot\frac{1}{n}$	1

• **Příklad** $\vec{r} = (-2, 5, -4, 1)$

$R(x) = (x^2 - 2x + 1)(x - 2) = x^3 - 4x^2 + 5x - 2$ pomocí $\omega = 4 \bmod 17$

$[1\ -2\ 1\ 0][-2\ 1\ 0\ 0]$ $1\ \omega\ \omega^2\ \omega^3$

$[1\ 1][-2\ 0][-2\ 0][1\ 0]$ $1\ \omega^2$ } zde to chci vyhodnotit

$[1][1]\ [-2][0]\ [-2][0]\ [1][0]$ 1

$[1][1]\ [-2][0]\ [-2][0]\ [1][0]$ $* [2\ 0] = [1 + 1\cdot 1 \quad 1 + 1\cdot\omega^2]$

$[1] \quad [0] \quad [0] \quad [0] \cdot [1]$ → tady $\omega = \omega^2$

$\overset{*}{[2\ 0]}\ [-2\ -2][-2\ -2][1\ 1]$

$[-2\ -8] \quad [1\ 4] \cdot [1\ \omega]$ → tady $\omega = \omega$

$[0\ -8\ 4\ 8][-1\ 2\ -3\ -6]$ → graf P, graf Q

⇒ $[0\ 1\ 5\ 3]$ ↙ vynásobit po složkách

$[0\ 5][1\ 3]$ $1\ \overline{\omega}^2$ … $\overline{\omega}\cdot\omega = 1 \Rightarrow \overline{\omega} = 13$, $13\cdot 4 = 2\cdot 26 = 2\cdot 9 = 18 = 1$

$\overline{\omega}^2 = -1$, $\overline{\omega} = -4$

$[5\ -5][4\ -2]$

$[4\ 8]\cdot[1\ -4] = [1\ \overline{\omega}]$ $-4 = \overline{\omega}$

$[9\ 5\ 1\ -13] \longrightarrow [-8\ 3\ -16\ 4]\cdot\frac{1}{4} = [-2\ 5\ -4\ 1] \Rightarrow R(x) = -2 + 5x - 4x^2 + x^3$

• **Příklad**

Pomocí FFT vynásob $59 \cdot 24 = 1416$

→ $59 = 9 + 10\cdot 5 + 0 + 0$ → $(9, 5, 0, 0)$

→ $24 = 4 + 10\cdot 2 + 0 + 0$ → $(4, 2, 0, 0)$

→ můžou vznikat velká čísla → až $2\cdot 9^2$ ⇒ potřeboval bych velké těleso ⇒ $\mathbb{C}$

$[9\ 5\ 0\ 0][4\ 2\ 0\ 0]$ $\omega = i$

$[9\ 0][5\ 0][4\ 0][2\ 0]$ v $1, \omega^2 \Rightarrow 1, -1$

$[9\ 9][5\ 5][4\ 4][2\ 2]$

$[5\ 5i] \quad [2\ 2i]\cdot[1\ i]$

$[14\ \ 9+5i\ \ 4\ \ 9-5i][6\ \ 4+2i\ \ 2\ \ 4-2i]$ … $y_0 \in \mathbb{R}$, $y_{n/2} = y_2 \in \mathbb{R}$, $y_1 = \overline{y_3}$

$\overset{*}{\Rightarrow} [84\ \ 26+38i\ \ 8\ \ 26-38i]$ $\overline{\omega} = -i$

$[84\ 8][26+38i\ \ 26-38i]$ v $1, \overline{\omega}^2 \Rightarrow 1, -1$

$[92\ 76][52\ \ 76i]$

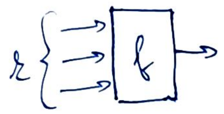
$[52\ 76] \quad \cdot[1\ -i]$

$[144\ 152\ 40\ 0] \rightarrow \cdot\frac{1}{4} \rightarrow [36\ 38\ 10\ 0] \Rightarrow$

$$\begin{array}{r} 1000 \\ 380 \\ 36 \\ \hline 59\cdot 24 = 1416 \end{array}$$

• Hradlové sítě

Def: Hradlo arity k počítá $f: \Sigma^k \to \Sigma$, kde Σ je konečná abeceda.



typicky $\Sigma = \{0, 1\}$

• Booleovská hradla … nad booleovskou abecedou

- binární (arita 2): AND, OR, XOR, $\leq$ (Implikace)
- unární (arita 1): NOT, Identita
- nulární (arita 0): 1, 0 … konstanty

Fakt: Z logiky víme, že AND, OR a NOT jsou univerzální.
Takže každé hradlo je možné vyrobit skládáním těchto funkcí.

Lemma: NAND a NOR jsou jediná dvě univerzální binární hradla.

Dk: 1, Jedině NAND a NOR mohou být univerzální

	0	1
0	1	0/1
1	0/1	0

- abych dokázal udělat negaci, tak potřebuju $(0,0) \to 1$ a $(1,1) \to 0$
- buď $(0,1) \to 0$ & $(1,0) \to 0$ nebo $(0,1) \to 1$ & $(1,0) \to 1$,
  jinak to je fce jedné proměnné

$\Rightarrow$ $(0,1) \to 0$ je NOR $(0,1) \to 1$ je NAND

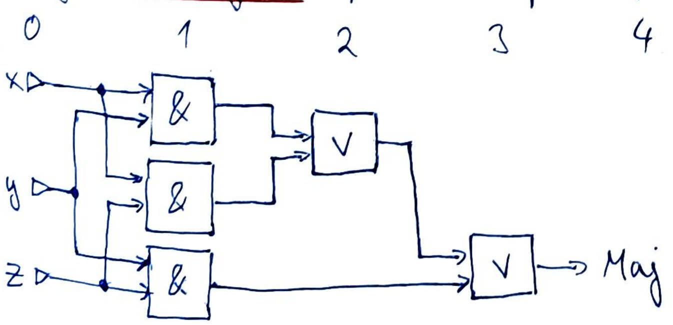
2, NAND a NOR jsou univerzální

$NOT(x) = NOR(x, x) = NAND(x, y)$

$OR(x, y) = NOT(NOR(x, y)) = NAND(NOT(x), NOT(y))$ … $x \vee y = \neg(\neg x \wedge \neg y)$

$AND(x, y) = NOT(NAND(x, y)) = NOR(NOT(x), NOT(y))$ … $x \wedge y = \neg(\neg x \vee \neg y)$ ■

• Majorita(x, y, z): pokud alespoň 2 jedničky 1, jinak 0



ekvivalentně

$Maj(x, y, z) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$

Každá logická formule lze zapsat jako nějaké hradlo

Ale hradla jsou mnohem mocnější, protože umožňují použít spočítaný mezivýsledek vícekrát

Def: Hradlová síť sestává z:

- hradla
- vstupní porty (X, Y, Z)
- výstupní porty Maj
- acyklické propojení vstupů a výstupů
- do $\forall$ vstupu $\forall$ hradla něco vede
- od $\forall$ výstupu $\forall$ hradla něco vede
- do vstupních portů sítě nic nevede a z výstupních nic nevede

- Výpočet probíhá v taktech

→ vždy chceme aby vydaly výsledky ty věci, co už mají všechny vstupy

0. takt: ohodnotíme vstupní porty sítě a konstanty

i+1. takt: ohodnotíme hradla a porty, jejichž vstupy byly ohodnoceny nejpozději v i-tém taktu.

↳ rozklad sítě na vrstvy

⇒ v i-tém taktu paralelně počítá všechno co je v i-té vrstvě

čas = # vrstev

prostor = # hradel

maximální arita hradel = 2

Potřebujeme omezit aritu hradel, jinak je vše spočitatelné v $O(1)$

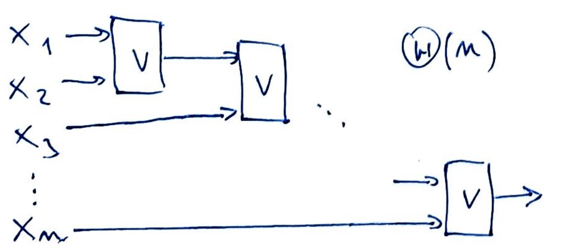
Poznámka: Kombinační obvody – obecná Σ

Booleovské obvody – $\Sigma = \{0,1\}$

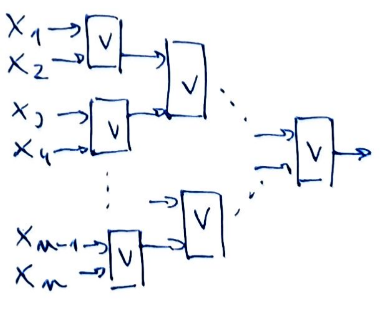
Požadavek: Hradlová síť umí počítat pouze se vstupy dané velikosti ⇒ není to alg.

⇒ Budeme chtít, aby $\exists$ program, který umí efektivně (polynomiálně) generovat hradlové sítě pro danou velikost vstupu.

- n-bit $OR(X_1, \dots X_n)$

1) 

$\Theta(n)$

2) 

$\Theta(\log n)$ vrstev

$\Theta(n)$ hradel

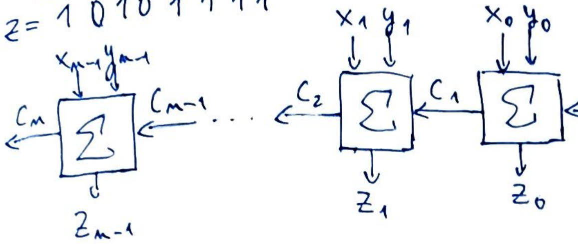
• Binární sčítání n-bit čísel

→ XOR = sčítání mod 2

$x = 01110100$
$y = 00111011$ $\quad c_0 = 0$
$c: 011100000$
$z = 10101111$

$z_i = x_i \oplus y_i \oplus c_i$ $\qquad c_0 = 0$

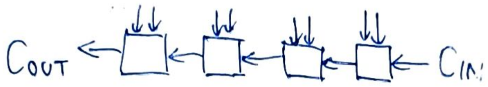
$c_{i+1} = Maj(x_i, y_i, c_i)$ $\qquad c_n = 1$ pokud to přeteklo



$\Theta(n)$ vrstev

$\Theta(n)$ hradel

1-bit sčítačky



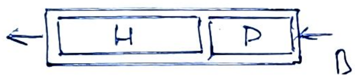
Chování bloku

↳ zafixujeme ty vstupy bl ⇒ závislost C_{OUT} na C_{IN}

• blok šířky 1:

x_i	0	0	1	1
y_i	0	1	0	1
$C_{OUT} =$	0	C_{in}	C_{in}	1
		<	<	

• blok šířky k:



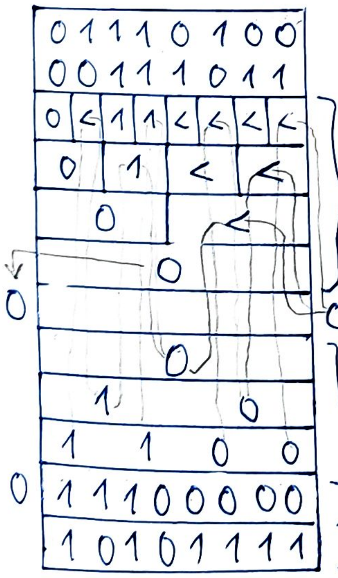
blok B rozseknutý na podbloky H a D

* (H \ D)	0	1	<
0	0	0	0
1	1	1	1
<	0	1	<

} horní blok je konstanta ⇒ je jedno co je D

} H kopíruje ⇒ řídí se podle D

⇒ budeme chtít popsat chování kanonických bloků = celý vstup, poloviny, čtvrtiny...



chování kanonických bloků $\Theta(\log n)$ vrstev

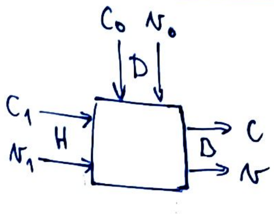
počítání carry $\Theta(\log n)$ vrstev

finální XORy $O(1)$

celkem $\Theta(\log n)$

• Jak zakódovat tu tabulku do hradla

→ chceme nějaké hradlo, co bude popisovat chování té tabulky *



C = constanta (1 = true)

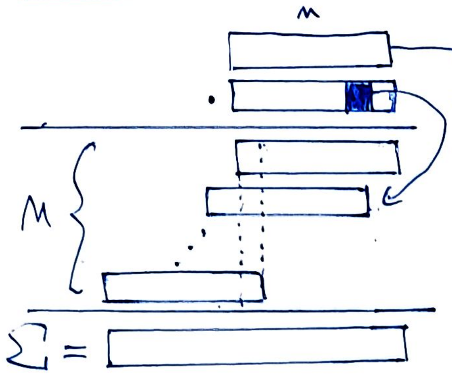
N = value

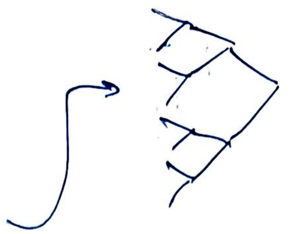
$C = C_0 \vee C_1$ aspoň 1 konstanta ⇒ konstanta

if C_1 : N_1
else : N_0 → Selektor

$N = (C_1 = 1)\ ?\ N_1 : N_0 = N_{C_1} = (C_1 \,\&\, N_1) \vee (\neg C_1 \,\&\, N_0)$

• Násobení čísel



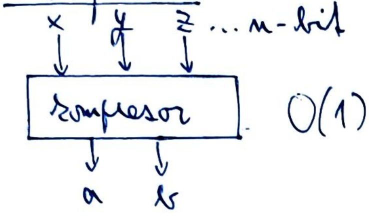


takto by bylo log(n) sčítání

⇒ celkem $\Theta(\log^2(n))$

paměť: $\Theta(n^2)$ ... mezivýsledky pro sčítání

• Kompresor



$O(1)$

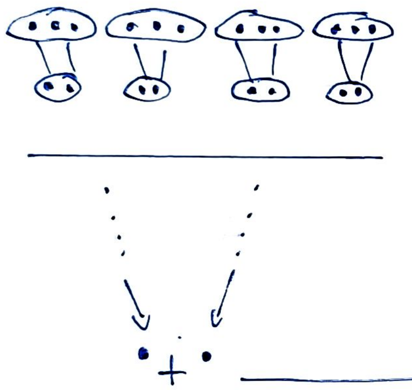
$a + b = x + y + z$

$$\begin{array}{cccc} x_3 & x_2 & x_1 & x_0 \\ y_3 & y_2 & y_1 & y_0 \\ z_3 & z_2 & z_1 & z_0 \\ \hline a_3 & a_2 & a_1 & a_0 \\ b_4 & b_3 & b_2 & b_1 \; 0 \end{array}$$

$x_i + y_i + z_i = b_{i+1} a_i$ → 2-ciferné číslo

Násobení $\Theta(\log n)$

• Obvod pro sečtení n čísel v log(n)



n čísel

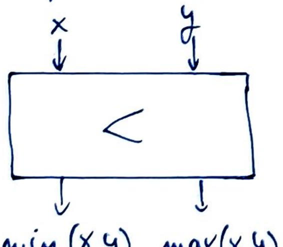
$\left(\frac{2}{3}\right) n$ čísel

$\left(\frac{2}{3}\right)^2 n$ čísel

$\Theta(\log n)$ hladin

$\Theta(\log n)$ finální sečtení

• Komparátorová síť



komparátor ... implementace pomocí hradla na porovnávání dvou čísel (2-doplněk, odečteme) a potom výsledky zadrátovat přes selektory (if-else)

↳ hloubka log(#bitů)

→ komparátorová síť se skládá z komparátorů, není nějaká pevná velikost čísel, to je důležité až pro implementaci

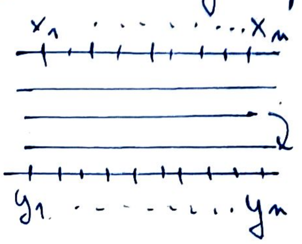
→ cíl je nějaké řazení čísel

BÚNO výstupy komparátorů se nikdy nevětví

↳ komparátor bere 2 a vrací 2

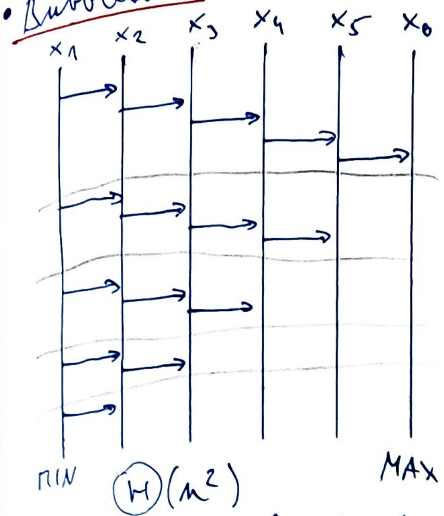
& čísel na vstupu je stejně jako na výstupu

→ to větvení by se stejně zobrazilo



→ na každé vrstvě nějaká permutace vstupu

• Bubblesort



$\Theta(n^2)$

příklad řazení 6 čísel

1 drát = 1 číslo

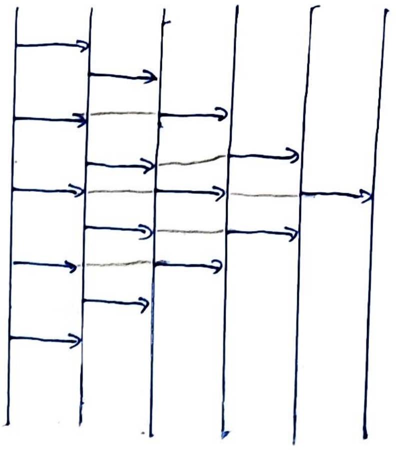
šipka ~ komparátor

↳ pošle maximum doprava

← klasická verze

paralelizace →

~ 2n hladin

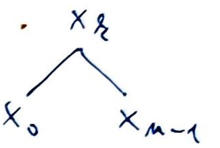


$\Theta(n)$

• Mergesort $n=2^k$ (předpokládáme, že čísla jsou navzájem různá)

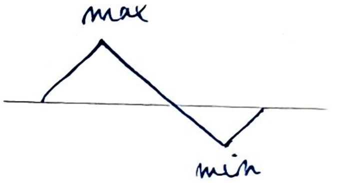
Def: Posloupnost $x_0,\dots x_{n-1}$ je

① čistě bitonická $\equiv \exists k: x_0 < x_1 < \dots < x_k > x_{k+1} > \dots > x_{n-1}$

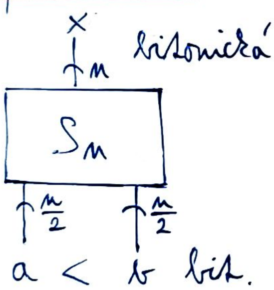


② bitonická $\equiv$ má čistě bitonickou rotaci

tedy $\exists l: x_l, \underbrace{x_{l+1}, \dots}_{\text{mod } n} \underbrace{x_{l+n-1}}_{\text{mod } n}$ je čistě b.



• Separátor S_n



Vstup: $x_0,\dots x_{n-1}$ bitonická

Výstup: $a_0 \dots a_{n/2-1}$ bitonická

$b_0 \dots b_{n/2-1}$ bitonická

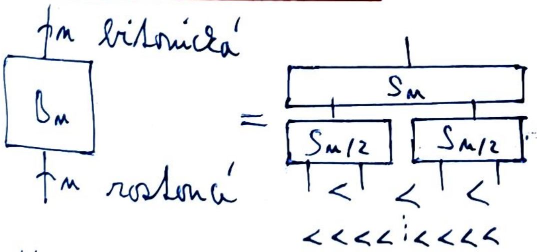
} podposloupnosti $\forall i: a_i \in X, b_i \in X$

↳ $\forall i,j: a_i < b_j$

čas $O(1)$ → ještě nevíme jak

prostor $O(n)$

• Bitonická třídička B_n



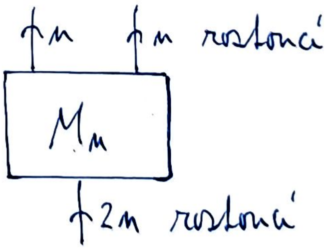
čas $\Theta(\log n)$

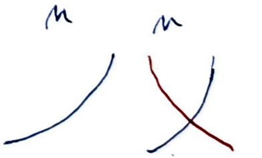
prostor $\Theta(n \log n)$

Vstup: $x_0,\dots x_{n-1}$ bitonická

Výstup: $y_0,\dots y_{n-1}$ rostoucí

• Slévačka M_n

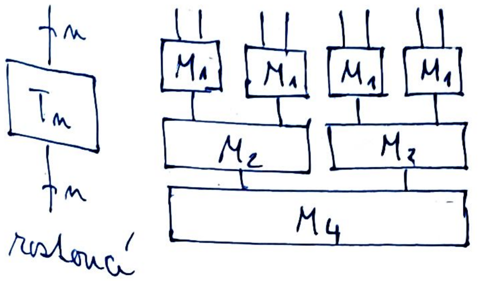




⇒ otočím tu druhou

⇒ b.p. ⇒ B_{2n}

• Třídička T_n



čas $\Theta(\log^2 n)$

prostor $\Theta(n \log^2 n)$

} čas $\cdot\, n$ = prostor

👁 na $\forall$ hladině (čas) je nejvýš n hradel

↳ n drátů ⇒ max. $n/2$ komparátorů

- **Je mergesort dobrý?**

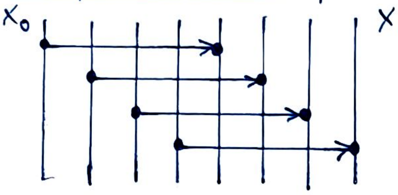
dolní odhad složitosti řazení porovnáváním $\Omega(n \log n)$ $\Rightarrow$ hloubka každé komparátorové třídicí sítě je $\Omega(\log n)$

↳ viz poslední pozorování … 1 hladina ~ n komp.

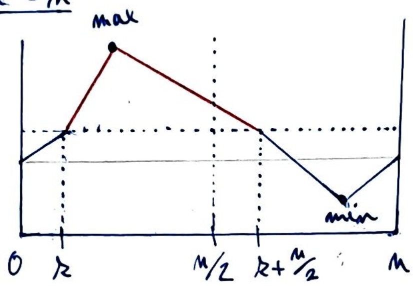
umí se $O(\log n)$ s obří konstantou

mergesort $\Theta(\log^2 n)$ je taky dost dobrý

- **Implementace separátoru S_n**



komp. $x_i \to x_{n/2+i}$
pro $i = 0, \dots n/2 - 1$



HORA: $n/2$ největších prvků
Souvislá: $x_k \dots x_{k+n/2-1}$
ÚDOLÍ: $n/2$ nejmenších prvků
Souvislé: $x_{k+n/2}, \dots x_{k-1}$

BÚNO: $k < \frac{n}{2}$ … jinak udělám rotaci o $\frac{n}{2}$ $\Rightarrow$ komparátory pořád na stejné prvky

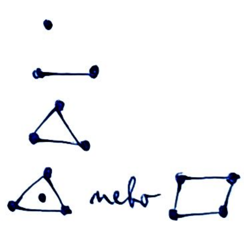
→ pro $i < k$: údolí → hora … neprohazují
pro $i \geq k$: hora → údolí … prohazují

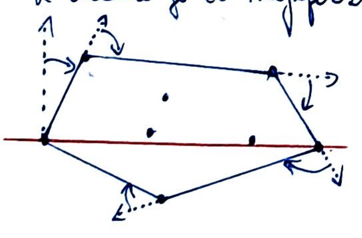
$\Rightarrow$ levá půlka: rotace údolí
pravá půlka: rotace hory } a rotace b. p. je b. p. ✓

čas $O(1)$ … komp. nezávislé
prostor $\Theta(n)$ … $n/2$ komp.

- **Geometrické algoritmy v rovině**
  - **Konvexní obal** $\Theta(n \log n)$

→ máme množinu bodů $x_1, \dots x_n \in \mathbb{R}^2$
a chceme je co nejefektivněji oplotit

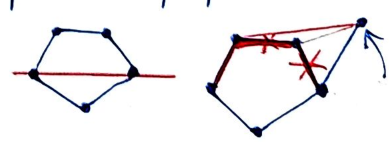




- nejlevější a nejpravější bod tam určitě patří
- ↳ inkrementální alg.: má obal pro nějaké body a objevíme nový bod napravo $\Rightarrow$ přepočítá se to

$\Theta(n \log n)$
1. Setřídíme body zleva doprava a pokud $x_1 = x_2$ zdola nahoru

Horní obálka
Dolní obálka



$\Theta(n)$
2. $H \leftarrow$ (levý bod), $D \leftarrow H$
3. Pro zbylé body b:
4. Dokud $|H| \geq 2$ & $H[-2]\, H[-1]\, b$ zatáčí doleva:
   Odebereme poslední bod z H
   Přidáme b na konec H
5. Podobně pro D ale zatáčíme doprava
6. Vrátíme D slepené s (H pozpátku)

Kontrola pravo / levo - točivosti

$B - A$
$C - B$

determinant

$\begin{vmatrix} B-A \\ C-B \end{vmatrix} > 0 \iff$

$\begin{vmatrix} B-A \\ C-B \end{vmatrix} < 0 \iff$

• Obecná poloha bodů

→ ve všech geometrických úlohách chceme, aby body byly v obecné poloze

= aby nenastal nějaký nepříjemný edge-case

↳ například u konvexního obalu nechceme aby bylo více bodů nad sebou 

→ pak je nemůžeme jednoznačně setřídit zleva doprava

⇒ řešení: otočíme rovinu o nějaké malé ε, což rozbije tyhle neobecné polohy

↳ musí být dost malé, aby nevytvořilo žádnou novou

⇒  → už je lze seřadit zleva doprava

~ předtím zleva doprava a zdola nahoru

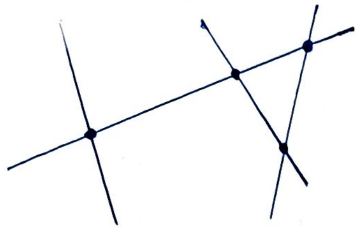
→ ten konvexní obal bude vypadat pořád stejně

⇒ původní alg funguje i pro neobecnou polohu, ale řadíme → a ↑

→ obecná poloha jsou ty hezké případy

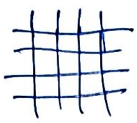
☼ Pravděpodobnost o.p. při náhodném rozmístění bodů je 1.

- Průsečíky úseček



zkontrolovat průsečík trvá $O(1)$ ... lingebra

→ zkusit každou s každou $\Theta(n^2)$

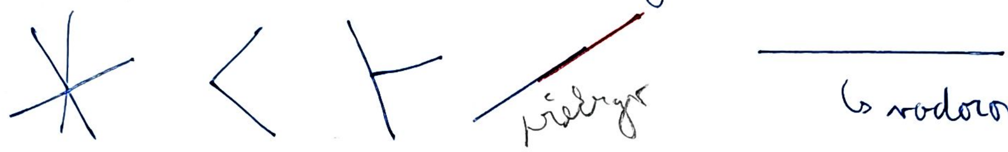


→ pro tohle je to optimální

p := # průsečíků

→ chceme něco jako O(nějaká funkce $(n+p)$)

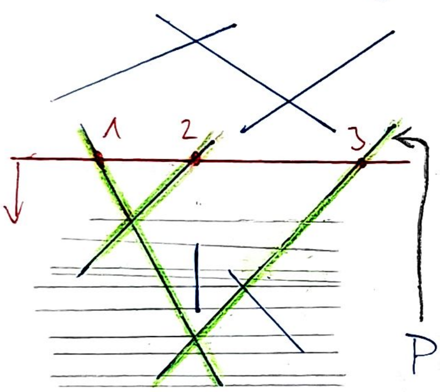
- obecná poloha → nechceme následující



↳ vodorovná ⇒ || se zametací p.

→ budeme předpokládat, že to nenastane, v implementaci by se to mělo ošetřit

- Zametání roviny → shora dolů



- děláme diskrétní simulaci spojitého posouvání přímky
→ koukáme se na přímku jen když se děje něco zajímavého

Události: Začátky úseček = horní bod
Konce úseček = spodní bod
Průsečíky úseček

P = Průřez := posloupnost úseček protínajících zametací p. seřazená zleva doprava podle x průsečíků

K = Kalendář událostí ... prioritní fronta

👁 Těsně před zametením průsečíku sousedí protínající se přímky v P

⇒ když spolu úsečky začnou sousedit → naplánujeme průsečík * → ale zabráníme odplánování

↳ když se mezi ně něco vmezeří, tak ho odplánujeme,
ale naplánujeme ty dva nové → ten starý se pak ověří znovu

→ když ohlásíme průsečík, tak se pořadí úseček v P prohodí

→ začátky a konce úseček můžeme všechny naplánovat na začátku

→ události jsou seřazeny podle jejich y souřadnice shora dolů

↳ odpovídá to pořadí ve kterém je přímka zametá

(pro 2 stejné y řadíme podle x zleva doprava ~ otočení roviny o ε)

* začnou spolu sousedit ⇒ podíváme se jestli se protínají

↳ pokud se protínají pod zametací přímkou, tak naplánujeme průsečík

Algoritmus

1. $P \leftarrow \emptyset$ → zametací přímka v $+\infty$
2. $K \leftarrow$ {začátky a konce úseček} → seřazení podle výšky
3. Dokud není K prázdný, odebíráme nejvyšší událost

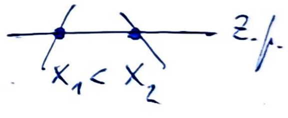
4. Začátek ⇒ přidáme úsečku do P → ovlivní nejvýš 3 sousednosti ↳ 1 maže, 2 přidám
5. SC Konec ⇒ odebereme úsečku z P → 3 sousednosti
6. Průsečík ⇒ nahlásíme na výstup, prohodíme úsečky v P → jednu smažu a zase vložím ↳ nejvýš 6 sousedností

$3+3$

7. Pro ∀ změnu sousednosti v P přeplánujeme dotčené průsečíky v K

Reprezentace

- **Kalendář** – prioritní fronta ⇒ halda nebo BVS
  ↳ max. $3n$ událostí najednou (začátky, konce, průsečíky v průřezu)
  ⇒ doba 1 operace je $O(\log n)$, paměť $O(n)$
- **Průřez** – chci aby uměl říct co je nalevo a napravo od dané úsečky → nepamatuju si celé LU ale umím to spočítat
  ⇒ BVS s úsečkami jako klíči → mají lineární uspořádání
  $a < b \equiv (a \cap \text{zam. p.})$ je vlevo od $(b \cap \text{zam. p.})$



  → y souř. zam. p. = y souř. aktuální události
  ⇒ max. n úseček ⇒ $O(\log n)$ čas/operace, paměť $O(n)$

Složitost

#událostí $= 2n + p$ → na 1 událost $O(1)$ operací s P i K

⇒ #operací $\in O(n + 2n + p) = O(n+p)$ ⇒ čas $O((n+p)\cdot\log n)$
inicializace K (u n)
prostor $O(n)$

pro $p = n^2$ horší než hrubá síla

umí se $O(n \log n + p)$

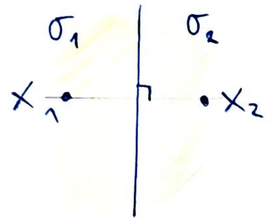
• Voroného diagram

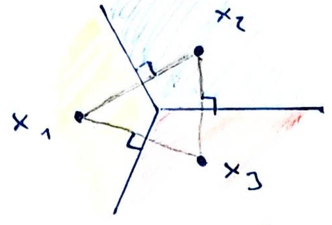
místa $X = \{x_1, \dots x_n\} \subseteq \mathbb{R}^2$

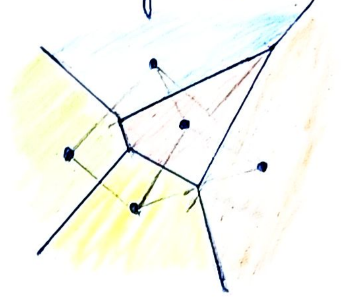
→ Eukl. metrika

oblasti $\sigma_1, \dots \sigma_n \in \mathbb{R}^2$, $\sigma_i := \{y \in \mathbb{R}^2 \mid \forall j : d(y, x_i) \leq d(y, x_j)\}$

↳ z σ_i je je to k x_i blíž než k jakémukoli jinému místu







↳ hranice patří do obou oblastí

👁 oblasti jsou určeny polorovinami danými přímkami oddělujícími místa

⇒ průnik polorovin je nějaký zobecněný konvexní mnohoúhelník

↳ mohou být otevřené do nekonečna

→ oblasti se umí najít v čase $O(n \log n)$

• Lokalizace bodu

→ máme V.d., dostaneme bod → ? do které oblasti patří

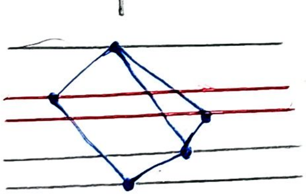
DS na rozklad roviny na mnohoúhelníky (ne nutně konvexní)

→ to že mnohoúhelníky mohou být otevřené nevadí

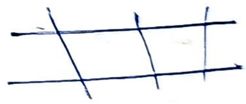
↳ vždy se dají uzavřít do nějaké krabičky a mimo k. bin. vyhl. mezi paprsky

Dotaz: Do jaké oblasti patří zadaný bod?

↳ pro hranici odpoví kteroukoli oblast



Rovinu rozsekáme na pásy → v pásu



→ pro ∀ úsečku si pamatuju jaká oblast je nalevo a napravo *

⇒ binárně vyhledám pás a pak oblast v něm ⇒ čas $O(\log n)$

→ pro ∀ pás si musím pamatovat úsečky v něm ⇒ paměť $O(n^2)$

⇒ chceme nějak zlepšit paměť

👁 pásy jsou vlastně průřezy po zametení v tom alg. pro průsečíky úseček

↳ bodu co definuje ten pás

* je to rovinný graf, takže # úseček $\in O(n)$

• **semi-perzistentní DS**

→ pamatuje si svoji historii ~ Git

• zápis ⇒ vytvoření nové verze, umíme vyhledat v historii podle ID

• Dotaz ⇒ dostaneme ID verze

→ průřez ~ pás

⇒ budeme si pamatovat všechny průřezy co vznikají při zametání roviny

⇒ uděláme semi-perz. BVS co má $O(\log n)$ čas / operace, $O(\log n)$ paměti / verze

⇒ **Konstrukce stromu**

- zametáme rovinu: # průsečíků = # bodů = n ⇒ $O(n)$ operací s průřezem

⇒ čas konstrukce $O(n \log n)$

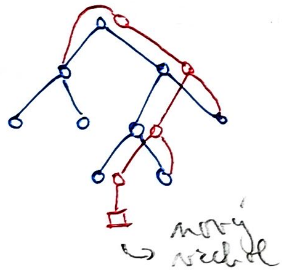
⇒ $O(n)$ verzí ⇒ prostor $O(n \log n)$ → umí se i prostor $O(n)$

⇒ dotaz v čase $O(\log n)$

↳ bin. v. přes pásy → získám ID průřezu ⇒ konkrétní BVS → dotaz $O(\log n)$

• **Semiperzistentní BVS**

- pravidlo: nesmí se změnit to, co už je zapsáno



→ perzistence kopírováním cest

→ ∀ operace si zkopíruje cestu k dotčenému vrcholu a vyrobí nový kořen (ID verze)

↳ pointry na podstromy jiných verzí – ale ty se už nezmění ✓

→ pro rozumnou hloubku vyrobíme $O(\log n)$ nových vrcholů

⇒ vyvažování: AVL strom mění pouze vrcholy na cestě a v jejím nejbližším okolí

→ pořád pouze $O(\log n)$ nových vrcholů na verzi

⇒ pro ∀ pás si v nějakém poli pamatuji pointer na kořen jeho stromu

- ## Třídy složitosti algoritmů

Def: <u>Rozhodovací problém</u> je funkce $r: \{0,1\}^* \to \{0,1\}$. → konečná posloupnost 0 a 1 ↳ ANO / NE

👁 Cokoliv lze zakódovat do posloupnosti bitů.

Příklad: Ex. p. graf a $k \in \mathbb{N}$

$$\underbrace{000\ldots0}_{k}\,|\,1\,|\,\underbrace{n}_{k}\,|\,\underbrace{k}_{k}\,|\,\underbrace{\text{matice sousednosti}}_{n^2}$$ → jednoznačné kódování

→ součástí problému je odpovědět NE na vstup ve špatném tvaru

Def: Problém A je <u>převoditelný</u> na problém B, píšeme $A \to B$ nebo $A \leq_p B$
$\equiv \exists f: \{0,1\}^* \to \{0,1\}^*$ tž.

1. $\forall \alpha \in \{0,1\}^*: A(\alpha) = B(f(\alpha))$ ... řešíme A pomocí B
2. $f(\alpha)$ lze spočítat polynomiálně vůči $|\alpha|$
   ↳ převod neboli redukce

B je alespoň tak těžká jako A

Příklad: ∃ párování velikosti alespoň $k \in \mathbb{N}$ v G? ⟶ Existuje tok velikosti alespoň $k \in \mathbb{N}$ v síti S?

Lemma: Nechť $A \to B$, B řešitelné v poly. čase. Potom A je také řešitelné v poly. čase.

Dk: vstup pro A $\xrightarrow{f}$ vstup pro B $\xrightarrow{\text{poly alg. pro B}}$ výstup pro A

α $\quad$ $f(\alpha)$ $\quad$ $A(\alpha) = B(f(\alpha))$

$\exists c: f(\alpha)$ lze spočítat v $O(n^c)$ .. $n = |\alpha|$

$\exists d: B(f(\alpha))$ lze spočítat v $O(m^d)$ ... $m = |f(\alpha)| \in O(n^c)$

alg co běží v čase T(n) nemůže vyrobit výstup delší než T(n)

$\Rightarrow A(\alpha)$ lze spočítat v $O(n^c + m^d) = O(n^c + n^{c \cdot d}) = O(n^{cd})$ → polynom ■

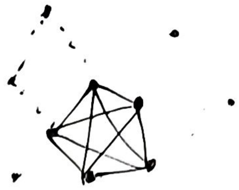
- ## Vlastnosti relace převoditelnosti:

1. $A \to A$ ... reflexivní
2. $A \xrightarrow[f]{} B \;\&\; B \xrightarrow[g]{} C \Rightarrow A \xrightarrow[f \circ g]{} C$ ... tranzitivní
3. $\exists A, B: A \to B \;\&\; B \to A$ ... např. A: má vstup lichou délku, B: má vstup sudou délku } není to uspořádání
4. $\exists A, B: A \not\to B \;\&\; B \not\to A$ ... např. $\forall \alpha: A(\alpha) = 0$, $B(\alpha) = 1$ } kdyby to bylo usp., tak ne lineární

👁 → je <u>částečné kvaziuspořádání</u>

- některé prob. jsou mezi sebou ekvivalentní $A \leftrightarrow B$, což nám dává třídy ekvivalence → a mezi těmito třídami už to je částečné uspořádání

• **Klika velikosti k**



Vstup: neorientovaný graf G, $k \in \mathbb{N}$

Výstup: $\exists A \subseteq V: |A| \geq k \ \& \ \forall u,v \in A: uv \in E$

• **Nezávislá množina velikosti k**

Vstup: neorientovaný graf G, $k \in \mathbb{N}$

Výstup: $\exists A \subseteq V: |A| \geq k \ \& \ \forall u,v \in A: uv \notin E$

• **NzMna $\Leftrightarrow$ klika**

→ stačí prohodit vlastnost být hranou

→ převodní funkce je hranový doplněk grafu } stejná převodní fce oběma směry

• **SAT splnitelnost**

Vstup: booleovská formule v CNF = konjunkce klauzulí ← disjunkce literálů (p nebo $\neg p$)

Výstup: $\exists$ ohodnocení proměnných, co tu formuli splňuje? = splňující ohodnocení

• **3-SAT** ... navíc $\forall$ klauzule obsahuje nejvýše 3 literály

• **3-SAT → SAT** ⇒ zjevné

→ jako převodní funkce nestačí obyčejná identita, protože

3-SAT($p_1 \vee p_2 \vee p_3 \vee p_4$) = NE ... nevalidní vstup } musí se provést kontrola validity vstupu

SAT($p_1 \vee p_2 \vee p_3 \vee p_4$) = ANO

Platí: (speciální případ problému A) $\xrightarrow{\text{"identita"}}$ A

• **SAT → 3-SAT**

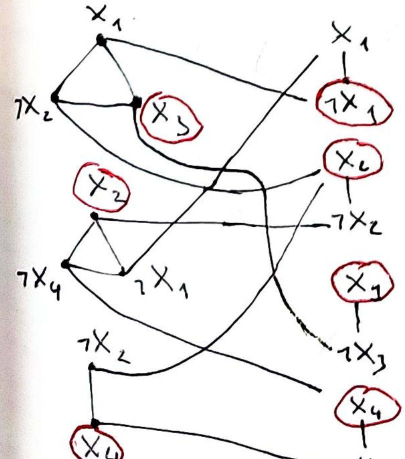
→ nová proměnná

$$\overbrace{(\underset{\substack{\uparrow \\ 2}}{\alpha} \vee \underset{\substack{\uparrow \\ k-2}}{\beta})}^{k} \iff \underset{\substack{\uparrow \\ 3}}{(\alpha \vee x)} \wedge \underset{\substack{\uparrow \\ k-1}}{(\beta \vee \neg x)}$$ ... ekvisplnitelné

⇒ # literálů v klauzuli klesl o 1 ⇒ štípeme dál, celkově poly. čas

• **3-SAT → NzMna** ← Takovému kódování se říká gadgety

klauzule literály



klauzule → kliky o velikosti dané klauzule

literály ↗

navíc vedeme hrany ($\ell \in$ klauzule) — ($\bar{\ell} \in$ literál)

k := # klauzulí, l := # literálů

formule je splnitelná ⇔ v grafu $\exists$ NzMna velikosti $\geq k+l$ (reálně)

⇐: $\forall$ klauzule je splněna vybraným literálem v $\forall$ literálu jsem zvolil jeho ohodnocení

⇒: vezmu si nějaké splňující ohodnocení a ohodnotím podle něho literály ⇒ v $\forall$ klauzuli $\exists$ alespoň 1 kladný literál, tak jej vyberu (nenastane $x_3 - \neg x_3$)

Tady vlastně nebylo třeba, aby to byl 3-SAT.

• **NzMna → SAT**

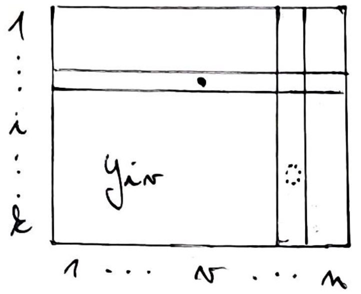
DÚNO $V(G)=\{1,2,\dots n\}$, NzMna velikosti k

• $x_1,\dots x_n \rightarrow x_u=1 \Leftrightarrow u \in$ NzMna

• y_{iv} pro $1\le i\le k$, $1\le v\le n$ → pomocné proměnné

1) $\forall uv\in E$: $\underbrace{(\neg x_u \vee \neg x_v)}_{}$ ... nenastane $x_u=x_v=1$ ⇒ je to nezávislé   ($\neg(x_u\wedge x_v)$ $\|$)

2) ještě potřebujeme velikost k → vrcholy v ní si seřadíme



$y_{iv}=1 \Leftrightarrow$ vrchol v je i-tý v NzMně

• v $\forall$ řádku je právě jedna 1

• v $\forall$ sloupci je je nejvýše jedna 1

$\forall i,j,v$: $\neg(y_{iv}\wedge y_{jv}) \sim (\neg y_{iv} \vee \neg y_{jv})$

$\forall i,u,v$: $\neg(y_{iu}\wedge y_{iv}) \sim (\neg y_{iu} \vee \neg y_{iv})$ ... nejvýše 1

$\forall i$: $(y_{i1}\vee y_{i2}\vee\dots\vee y_{in})$ ... alespoň 1 } právě 1

→ ∃ množina k vrcholů

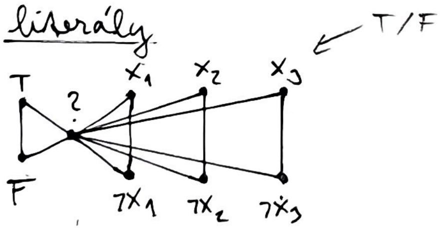
⇒ $\forall i,v$: $y_{iv} \Rightarrow x_v \sim (\neg y_{iv} \vee x_v)$ → pokud je něco v té množině, tak to je nezávislé

• **3-SAT → 3-Obarvitelnost**

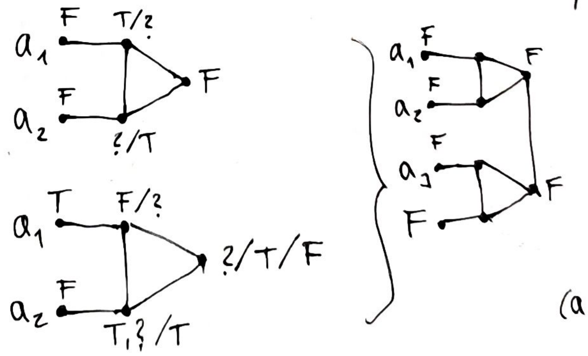
barvy: T true, F false, ? filler barva

☼ trojúhelník △ obsahuje všechny 3 barvy

literály



klauzule: $(a_1\vee a_2\vee a_3)$ nelze splnit $\Leftrightarrow \overbrace{a_1=a_2=a_3}^{F}$



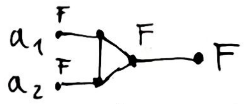
nelze obarvit
⇕
$a_1=a_2=a_3=F$
⇕
$(a_1\vee a_2\vee a_3)$ nelze splnit

→ klauzule nekreslíme stranou, ale zadrátují se do toho grafu literálů

→ $a_1, a_2, a_3 \sim x_1/\neg x_1, x_2/\neg x_2, x_3/\neg x_3$

→ jako F si vezmu F z toho $\triangleright$ vlevo (T, F)

$(a_1\vee a_2)$ nelze $\Leftrightarrow a_1=a_2=F$



(a_1) nelze splnit $\Leftrightarrow a_1=F$

→ v opačném směru to taky platí

- 3,3-SAT ... navíc ∀ proměnná je v nejvýše 3 klauzulích
- 3,3-SAT → 3-SAT pomocí identity s kontrolou validity vstupu
- 3-SAT → 3,3-SAT

→ nechť X je proměnná s $k > 3$ výskyty

⇒ pro ∀ výskyt nová proměnná $X_1, \dots X_k$ a i-tý výskyt X nahradíme X_i
navíc přidáme nové klauzule $X_1 \Rightarrow X_2,\ X_2 \Rightarrow X_3, \dots X_k \Rightarrow X_1$,
což nám zajistí, že $v(X_1) = v(X_2) = \dots = v(X_k)$

$$\left.\begin{array}{l} X_3 \Rightarrow X_4 \sim \neg X_3 \vee X_4 \\ X_4 \Rightarrow X_5 \sim \neg X_4 \vee X_5 \end{array}\right\}$$ ∀ nová proměnná má 3 výskyty a alespoň 1 je ⊕ a jeden ⊖

- 3,3-SAT* ... navíc ∀ literál nejvýše 2-krát
- 3,3-SAT → 3,3-SAT*

→ pokud mám nějaký literál více než 2-krát, tak pro odpovídající proměnnou udělám stejnou tranformaci jako předtím

- 3D-párování

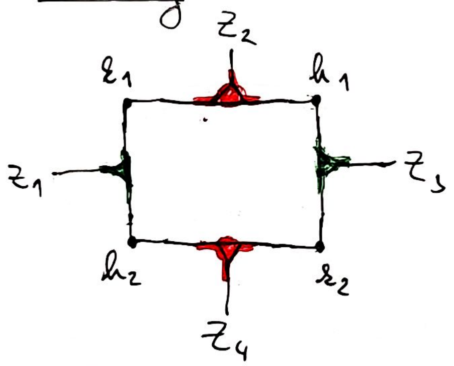
HOŠI DÍVKY KOČKY PREFERENCE

můžeme všechny uspokojit?

Vstup: Konečné množiny K, H, Z + množina $T \subseteq K \times H \times Z$

Výstup: $\exists\, T' \subseteq T$ t.ž. ∀ prvek K, H, Z je v právě 1 trojici z T!

literály

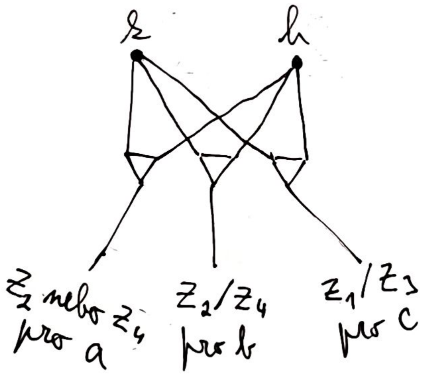


● 0... volné Z_1, Z_3

● 1... volné Z_2, Z_4

klauzule

$a \vee b \vee \neg c$



3D párování ⇔ splnitelné

⇒: proměnné ohodnotím podle gadgetů pro literály
↳ to mi dává 2 volná zvířátka
↳ každá klauzule je splněna některým z těchto zvířátek
↳ zvířátka jsou jen 2 ⇒ 3,3-SAT*

⇐: g. pro literály spárujeme podle ohodnocení proměnných
→ ∀ klauzule je splněná nějakým literálem ⇒ alespoň 2 volná zvířátka
→ ∀ literál použitý nejvýše 2-krát ⇒ 2 zvířátka nám stačí

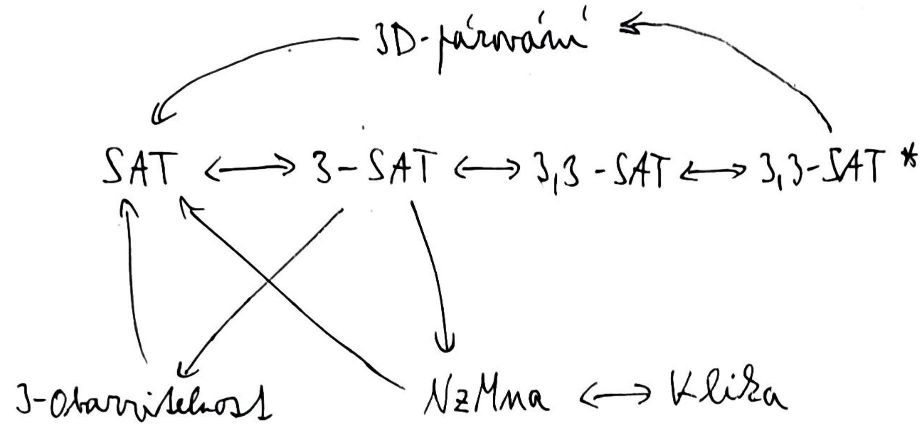
! Přebývají nám zvířátka **!** ... ∀ klauzule adoptuje jen 1 zvířátko

→ (2·# proměnných − # klauzulí) volných zvířátek

⇒ přidám tolik párů univerzálních milovníků zvířátek

↳ každému páru se líbí všechna zvířátka

- Co už umíme převádět?



- Třídy problémů

$L \in P \equiv \exists$ polynomiální algoritmus

Def: Třída problémů P. Problém $L \in P \equiv$

$\exists A$ algoritmus, $\exists p$ polynom: $\forall x: A(x) = L(x)$ & $A(x)$ doběhne do $p(|x|)$ kroků.

Def: Třída problémů NP. Problém $L \in NP \equiv$

$\exists V \in P$, verifikátor

$\exists g$ polynom ← omezení délky certifikátu

$\forall x: L(x) = 1 \iff \exists y$ certifikát

$\underbrace{|y| \leq g(|x|)}$ & $V(x,y) = 1$.
důkaz

certifikát není příliš dlouhý

⇒ $L \in NP$, pokud je možné rychle zkontrolovat jeho údajné řešení

→ například najít kliku je těžké, ale zkontrolovat, jestli je daná množina klikou je jednoduché. Certifikát by v tomto případě byla ta klika. A verifikátor by byl alg. co o množině rozhodne, zda to je klika.

👁 $P \subseteq NP$, $SAT \in NP$. $?P = NP?$

Def: Problém L je NP-těžký $\equiv \forall K \in NP: K \to L$.

Def: Problém L je NP-úplný $\equiv$ je NP-těžký & $L \in NP$.

Lemma: Pokud $L \in P$ je NP-těžký, potom $P = NP$.

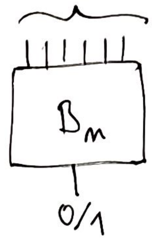
Dk: Nechť $K \in NP$. $K \to L$ & $L \in P \Rightarrow K \in P$. ■

Lemma: Nechť $K, L \in NP$, $K \to L$, K je NP-úplný, pak L je NP-úplný.

Dk: Nechť $M \in NP$. $M \to K \to L \Rightarrow M \to L$. ■

Věta (Cook, Levin): SAT je NP-úplný. — Bn je vždy poly. velká n

Lemma: $\forall L \in P$ $\exists A$ polynomiální alg. t.ž.
$\forall n$: $A(n)$ je hradlová síť B_n s n vstupy a 1 výstupem řešící L pro vstupy délky n.

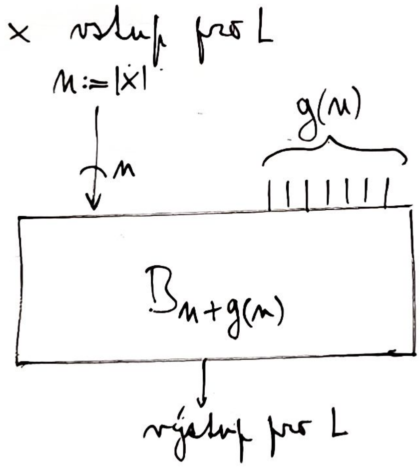


Dk: Neformálně: Pro L existuje nějaký polynomiální alg. který lze spustit na počítači. Počítač je uvnitř vlastně velká hradlová síť, co se mění v čase (jsou tam nějaké obvody co si pamatují stav do dalšího taktu)
→ Prostě tu síť okopírujeme v každém taktu výstupy těchle měnících se obvodů zadrátujeme vždy do té další sítě.
⇒ Takto získáme polynomiálně mnoho polynomiálně velkých, propojených sítí
⇒ to je zase nějaká polynomiálně velká síť. ∎

Věta: Obvodový SAT je NP-úplný.

Dk: Obvodový SAT je problém, kde dostaneme hradlovou síť →
Výstup: ∃ ohodnocení vstupních portů, aby byl výstup 1?
⇒ Nechť $L \in NP$, V verifikátor, g omezení délky certifikátu dle definice NP.
Určo chceme certifikát délky právě $g(|\text{vstup}|)$.
↳ kratšímu certifikátu lze dát padding $\boxed{0000\ldots01 \mid \text{Actual certifikát}}$

Plán: chceme ukázat $L \to$ Obvodový SAT. ... pomocí Lemmatu vyrobíme obvod



Předchozí lemma nám garantuje polynomiálně velkou síť, co umíme vyrobit v poly. čase.
⇒ ta síť simuluje V a víme $V \in P$.

- zadrátujeme vstup $x_1, \ldots x_n$

Obvodový SAT vydá 1
⇕
∃ ohodnocení těch vstupních $g(n)$ portů t.ž. výstup té sítě $= V(\text{vstup}, \text{ohodnocení}) = 1$
⇕
∃ řešení problému L pro tento vstup ∎

ohodnocení = certifikát

Lemma: Obvodový SAT → SAT v CNF

Dk: ① převedeme obvod aby byl celý z NORů.... máme jen unární a binární hradla a konstanty → určitě polynomiálně jde
② zavedeme proměnné pro vstupní porty
③ zavedeme proměnné pro výstupy hradel

$\neg x \wedge \neg y \Rightarrow z \sim (x \vee y \vee z)$
$x \Rightarrow \neg z \sim (\neg x \vee \neg z)$
$y \Rightarrow \neg z \sim (\neg y \vee \neg z)$

→ přidáme tyhle klauzule pro ∀ hradlo
& přidáme 1-prvkovou klauzuli pro výstupní port sítě
↳ to je nějaká proměnná buď ze vstupu nebo výstupu hradel ∎

Důsledek: SAT je NP-úplný. ∎

Důsledek: Když jsme SAT omezili na CNF, tak jsme si to nezlehčili.

Dk: Přímý převod sice může nabobtnat až exponenciálně, ale

SAT co není v CNF → Obvodový SAT → SAT v CNF ■

NP-úplné

logické
- SAT
- 3-SAT
- 3,3-SAT
- Obvodový SAT

grafové
- NzMna
- klika
- k-obarvitelnost ($k \geq 3$)
- Hamiltonovská cesta
- 3D-párování

číselné
- Součet podmnožiny - aby byl n
- batoh - omezená kapacita, chci ukrást co nejvíc
- 2 loupežníci - rozdělit lup na půl
- 0-1 řešení soustavy lin. rovnic

• **Jak řešit NP-úplné problémy?**

① využít malých vstupů, dělat nějaké lokální optimalizace

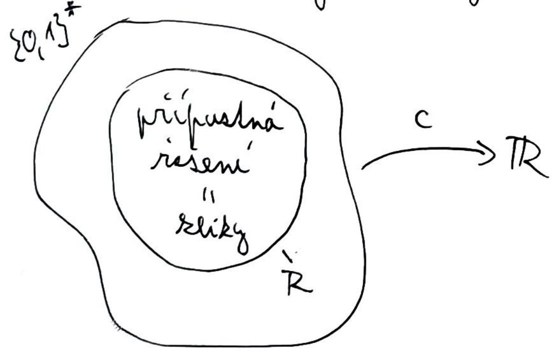
② speciální případy mohou být polynomiálně řešitelné ... 2-SAT jde lineárně

③ udělat nějaký polynomiální aproximační algoritmus

④ použít heuristiku nebo evoluční algoritmus

• **Optimalizační problémy**

→ Například najít co největší kliku.



$R := \{x \in \{0,1\}^* \mid x \text{ je přípustné řešení}\}$

$C: R \Rightarrow \mathbb{R}$ ... cena / ohodnocení

chci: $x^* \in R$ t.ž. $C^* = C(x^*)$ je min. / max.

optimální řešení

• **NzMna ve stromech**

Lemma: Nechť T je strom a l jeho list. Pak alespoň 1 max. NzMna v T obsahuje l.

Dk: Nechť M je nějaká max. NzMna. Potom buď

1, $l \in M$ ✓

2, $l \notin M$ <
- soused$(l) \notin M \Rightarrow M$ nebyla max ↯
- soused$(l) \in M \Rightarrow M' := M -$ soused $+ l$ je pořád max. a navíc $l \in M'$. ■

Algoritmus

dokud $T \neq \emptyset$:
- Najdeme list l (nebo izolovaný vrchol)
- Přidáme l do NzMny
- Smažeme l, souseda l

$O(n)$ → **Implementace pomocí DFS**

→ z rekurze vracím, zda jsem byl přidán do NzMny
- alespoň 1 syn byl přidán ⇒ já ne
- žádný syn nebyl přidán ⇒ já ano

• **Rozdělování přednášek do poslucháren**

Vstup: uzavřené intervaly $\underbrace{\langle x_1, y_1\rangle, \langle x_2, y_2\rangle, \dots \langle x_n, y_n\rangle}_{\text{vrcholy}}$ — má to netriviální průnik

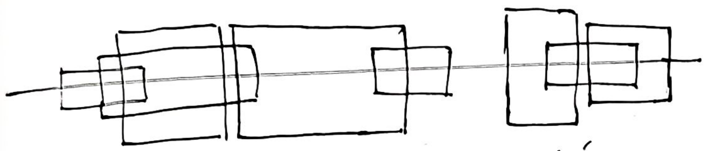
⇒ Intervalový graf: $\{I, J\} \in E \equiv |I \cap J| \geq 1$

Rozdělování přednášek do poslucháren ~ obarvení tohoto grafu co nejméně barvami

- barvy ~ posluchárny
- 2 přednášky se nemůžou překrývat v 1 posluchárně ~ konce hran různě barevné

⇒ obarvování je obecně NP-úplné

⇒ ale pro intervalové grafy to jde rychle



⇒ uděláme zametání roviny přímkou

⇒ 1D, takže zametání přímky bodem

⇒ máme seznam volných poslucháren a #vyrobených poslucháren celkem

⇒ když přijde přednáška, tak jí dáme libovolnou volnou posluchárnu

↳ pokud není, tak ji nejprve vyrobíme

⇒ na začátku musíme intervaly setřídit — $O(n \log n)$ třídění + $O(n)$ zametání

na konci #vyrobených poslucháren = min #potřebných barev — $c(K_n) = n$

Dk: novou posluch. vyrábím když jsou všechny ostatní zabrané

⇒ ty přednášky se všechny překrývají ⇒ v tom grafu je klika dané velikosti

• **Problém batohu**

n předmětů: hmotnosti $h_1, \dots h_n \geq 0$; ceny $c_1, \dots c_n \geq 0$ — $h(P) := \sum_{i \in P} h_i$, $c(P) := \sum_{i \in P} c_i$, $[k] := \{1, 2, \dots k\}$

H ... nosnost batohu

Výstup: $P \subseteq [n]$ t.ž. $h(P) \leq H$ a $c(P)$ je max. ... chceme ukrást co nejvíc

⇒ pro $c_1, \dots c_n \in \mathbb{R}$ to je NP-úplné, ale pro $\mathbb{N}$ lze dynamické programování

Dynam. prog.: vyplňujeme tabulku

$A_{k,c} := \min\{h(P) \mid P \subseteq [k] \wedge c(P) = c\}$ ... řádek i ~ můžu jen prvních i předmětů

$\min(\emptyset) = \infty$ — v tabulce máme min. hmotnosti ↳ klidně $> H$

	0	1	2	...	$C = \sum_i c_i$
0	0	$+\infty$	$+\infty$	...	$+\infty$
1	0				
2	0	Vše máme			
⋮					
k		povolíme k. předmět			
⋮					
n	0				

pokud $c_k > c$: $A_{k,c} = A_{k-1,c}$

jinak $A_{k,c} = \min \begin{cases} A_{k-1,c} & \dots \text{pokud } k \notin \min \\ h_k + A_{k-1,c-c_k} & \dots \text{pokud } k \in \min \end{cases}$

⇒ Tabulku vyplním za $\Theta(n \cdot C)$, kde $C = \sum_i c_i$ a $C^* = \max\{c \mid A_{n,c} \leq H\}$

⇒ chci tu optimální množinu — $x_1 = B_{k,c^*}$ ... poslední v optimální m.

$B_{k,c}$ = poslední přidaný předmět ⇒ $x_2 = B_{x_1, c^* - c(x_1)}$ ... předposlední

⋮

Je ta složitost $\Theta(n \cdot C)$ dobrá?

... $n \cdot C$ vypadá jako polynom

ale pokud délka vstupu je ten řetězec $\in \{0,1\}^*$, tak
ta čísla jsou zapsaná ve 2-soustavě

$\Rightarrow$ C jé exponenciální vůči délce vstupu

$\Rightarrow$ jé to pseudopolynomiální algoritmus

↳ složitost závisí na interpretaci vstupu

• Aproximační alg. → relativní chyba $\frac{C^*-C'}{C^*} \le 1-\alpha$

- Maximalizační problém: chceme $C(\text{výstup}) \ge \alpha \cdot C^*$, $\alpha \in (0,1]$ $\quad \frac{C}{C^*} \ge \alpha \Rightarrow 1-\frac{C}{C^*} \le 1-\alpha$
- Minimalizační problém: chceme $C(\text{výstup}) \le \alpha \cdot C^*$, $\alpha \ge 1$

↳ α-aproximace ↳ relativní chyba $\frac{C'-C^*}{C^*} \le \alpha - 1$

• Problém obchodního cestujícího

Vstup: hranově ohodnocený neorientovaný graf

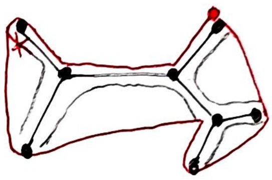
Výstup: nejkratší hamiltonovská kružnice ... obsahuje ∀ vrcholy

⇒ Speciální případ: Úplný graf s △ nerovností. ← konečný metrický prostor

2-aproximace:

① $T \leftarrow$ min. kostra

② pomocí DFS obcházíme okolo $T \Rightarrow$ h.k.



ALG := $\sum$ délek hran této h. kružnice

$|T|$:= délka min. kostry

$ALG \le 2|T|$

$|T| \le |\text{optim. h.k. bez nějaké hrany}| \le OPT$ } $ALG \le 2 \cdot OPT$

↳ h.k. bez libovolné hrany je kostra & T je min. kostra

→ zjevně, kdyby ∃ nějaká aproximace obecného obch. cest., tak P = NP, protože ta aprox. by našla nějakou hamilt. k., což je NP-úplné.

Věta: Kdyby existoval polynomiální t-apx. alg. pro p.o.c. v úplných grafech, tedy △ nerovnost nemusí platit, tak P = NP.

Dk: Ukážeme, že pak lze rozhodnout, zda graf má h.k. polynomiálně.

$G \longrightarrow G'$ zúplnění G < původní hrany délka 1; nové hrany délka $L > 1$

chci zda G má HK

G má HK ⟺ opt. v G' má délka n

G nemá HK ⟺ opt. v $G' \ge n-1+L$ ... alespoň 1 nová hrana

→ pro 1 aprox. je jasné kdy G má HK

→ pro t-aprox. chci $\underbrace{t \cdot n}_{\text{má}} < \underbrace{n-1+L}_{\text{nemá}}$ ⇒ zvolíme $L > tn - n + 1$

⇒ pokud mi aprox. řekne výstup $\le t \cdot n$ ⇒ má

jinak nemá ■

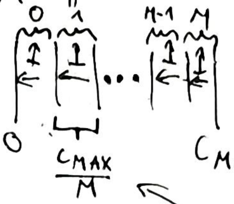
Aproximace batohu kvantováním cen

- pokud jsou všechny ceny násobky nějaké konstanty, tak je tou konstantou můžu vydělit, vyřešit tenhle zredukovaný problém a výsledek vynásobit tou konstantou

$\Rightarrow$ ceny prostě vydělíme a nepřesnosti zaokrouhlíme ... to bude fungovat i pro $C \in \mathbb{R}$

chci $\langle 0, C_{max} \rangle \to \{0, \dots M\}$

$$\hat{c}_i := \left\lfloor \frac{c_i}{C_{max}/M} \right\rfloor = \left\lfloor \frac{c_i}{C_{MAX}} \cdot M \right\rfloor \quad \dots \quad \frac{C_{max}}{M} \text{ je ta škálovací konstanta}$$



rozdělím to na M intervalů délky C_{MAX}/M

$\to$ vyřeším pro ceny $\hat{C}$

$\to$ vrátím ty původní předměty

$\Rightarrow$ vše v něm $\mapsto$ i $\frac{C_{MAX}}{M}$

$\Rightarrow$ chyba ceny 1 položky $< \frac{C_{MAX}}{M}$ = délka toho intervalu

$\Rightarrow$ chyba ceny odpovězené množiny $< n \cdot \frac{C_{MAX}}{M} \overset{?}{\leq} \varepsilon \cdot C^*$ (*)

(*) musíme nejprve vypustit předměty $i : h(i) > H$ ... mohou mít obří ceny ale jsou nepoužitelné

$\Rightarrow$ potom $C_{MAX} \leq C^*$

$\Rightarrow$ pro $M \geq \frac{n}{\varepsilon} \Rightarrow \frac{n}{M} \leq \varepsilon$ máme chyba $< \frac{n}{M} C_{MAX} \leq \varepsilon C_{MAX} \leq \varepsilon \cdot C^*$

$\Rightarrow$ relativní chyba $= \frac{\varepsilon \cdot C^*}{C^*} = \varepsilon = 1 - \alpha \Rightarrow \alpha = 1 - \varepsilon$

$\uparrow$ aproximační poměr

Algoritmus

1. Odstraníme položky těžší než H
2. $C_{MAX} \leftarrow \max_i \{c_i\}$
3. $M \leftarrow \left\lceil \frac{n}{\varepsilon} \right\rceil$
4. $\forall i : \hat{c}_i \leftarrow \left\lfloor \frac{c_i}{C_{max}} \cdot M \right\rfloor$
5. Vyřešíme úlohu s $h, \hat{C}$
6. vrátíme ty předměty, které nám to řeklo

ta dolní celá část je tady důležitá! ↳ velikost té chyby by se správně měla ukázat přes nějaké nerovnosti

Složitost: $T(n) = O(n \cdot \hat{C}) = O\left(\frac{n^3}{\varepsilon}\right) \quad \dots \quad \hat{C} \leq n \cdot \hat{c}_{MAX} \leq n \cdot M = n \cdot \left\lceil \frac{n}{\varepsilon} \right\rceil$

$\Rightarrow$ vyrobili jsme algoritmus, který pro $\forall \varepsilon > 0$ nalezne $(1-\varepsilon)$-aproximaci problému batohu v čase $O(n^3/\varepsilon)$. $\to$ aproximační schéma

Def: Polynomiální apx. schéma (PTAS) max. problému je alg., který pro $\forall \varepsilon > 0$ nalezne $(1-\varepsilon)$-aproximaci v čase $O(\text{polynom}(n))$ $\leftarrow$ nějaká závislost na ε $\to$ klidně $n^{\frac{1}{\varepsilon}}$

... je plně polynomiální (FPTAS) $\equiv$ čas $\in O\left(\text{polynom}\left(n, \frac{1}{\varepsilon}\right)\right)$