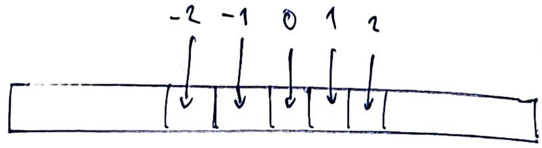
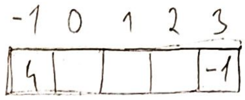
• **Random Access Machine** – RAM

- počítá s celými čísly — neomezeně velkých čísel
- paměť: pole čísel indexované celými čísly



- instrukce

① kam ← co
- literál ~ imm — literál
- obsah buňky [adresa]
- nepřímá adresace [[adresa]]



$[[3]] = [-1] = 4$

② kam ← a OP b   +, −, *, /, mod, dělení 0 je zakázané

celočíselné

XOR

&, |, ⊕, <<, >>

$x << k = x \cdot 2^k$

$x >> k = x / 2^k$

řídící operace

③ halt – zastavení programu

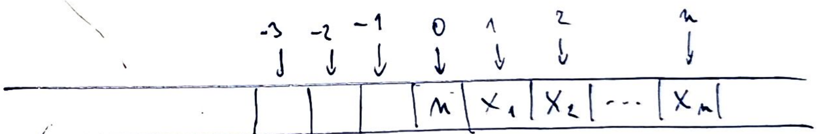
+ za programem je vždy implicitní halt

goto KAM – nepodmíněný skok

if a RELACE b goto KAM

= ≠ < > ≤ ≥

- vstup: je ve smluvených buňkách, jinde cokoli
- výstup: zase v nějakých smluvených buňkách ← po haltu



Def: Algoritmus je program RAMu.

Př: Bubble sort.

Opakujeme
- p ← NE ← 0
- Pro $i = 1, \dots n-1$:
  - Pokud $x_i > x_{i+1}$:
    - $x_i \leftrightarrow x_{i+1}$
    - p ← ANO ← 1
- dokud p = ANO

zkratky A := [-1], B := [-2], ... Z := [-26]

```
        if [0] ≤ 1 goto END
ZNOVU:  P ← 0
        I ← 1
DALŠÍ:  J ← I+1
        if [I] ≤ [J] goto OK
        P ← 1
        T ← [I]
        [I] ← [J]
        [J] ← T
OK:     I ← I+1
        if I < [0] goto DALŠÍ
        if P = 1 goto ZNOVU
END:    halt
```

3 instrukce navíc

2 navíc

1 průchod: 4 v 8 → $4n-4$ až $8n-8$

1 průchod: $4n-1$ až $8n-5$

průchodů: 1 až n

celkem: $4n+1$ až $8n^2-5n+2$

ve skutečnosti ale poslední průchod vždy jen $4n-1$ ⇒ $8n^2-9n+6$

• v nejlepším případě $4n+1$

• v nejhorším případě $8n^2-9n+6 \in \Theta(n^2)$

```
# inicializace
            if [0] ≤ 1 goto END
            N ← [0]
            B ← 1
            goto BUILDHEAP

# index maxima dvou potomků
MAXCHILD    J ← 2 * I
            K ← J + 1
            if K ≤ N goto 2SYNI
            goto 1SYN
2SYNI       if [K] ≤ [J] goto 1SYN
            J ← K
1SYN        goto MAXFOUND

# probublání haldou nahoru
BUBLANI     R ← I

ZNOVU       J ← 2 * I
            if J > N goto KONEC
            goto MAXCHILD      # J = max{potomci I}
MAXFOUND    if [I] ≥ [J] goto KONEC
            T ← [I]
            [I] ← [J]
            [J] ← T
            I ← J
            goto ZNOVU
KONEC       I ← R
            if B = 1 goto BUILDING
            goto SORTING

# build max heap
BUILDHEAP   I ← [0]
DALSI       goto BUBLANI
BUILDING    I ← I - 1
            if I ≥ 1 goto DALSI
            B ← 0

# samotné řazení
            H ← [0]
            I ← 1
DALSI#2     T ← [1]
            [1] ← [H]
            [H] ← T
            N ← H - 1
            goto BUBLANI
SORTING     H ← H - 1
            if H ≥ 2 goto DALSI #2

# konec
END         halt
```

• Cena 1 instrukce - více možností

1, jednotková ⇒ čas = Σ instrukcí - všechny lze zakódovat do 1 obřího čísla
→ všechno jde v konstantním čase !

2, omezená šířka slova na W b. ⇒ omezená max. abs. hodnota čísla na W
→ omezili jsme paměť na 2^{W+1} ← sgn
⇒ pro vstup délky n musí být $W \geq \log(n)$ → abychom ho přečetli
$\log(n) := \log_2(n)$
→ použijeme $W = c \cdot \log(n)$ ⇒ $|\text{čísla}| \leq n^c$
← konstanta
$\max(n, c')$ pro $n = 0, 1$
polynomiálně velká
⇒ ty kódovací algoritmy jsme tímhle zabili

3, logaritmická
- instrukce nemá konstantní cena - záleží na tom, s jak velkými č. počítá
cena = # bitů operandů včetně adres - i ta adresace něco stojí

4, relativní logaritmická

$$\text{cena} = \left\lceil \frac{\#\text{ bitů}}{\log(n)} \right\rceil$$

→ pro polynomiálně velká čísla ⇒ jednotková / konstantní
⇒ pro obří čísla ⇒ logaritmická

⇒ budeme používat 4, ale pro normální programy to je 1,

Def: Doba běhu algoritmu pro vstup X
$t(X) :=$ součet cen provedených instrukcí ← může být i nekonečná

Def: Časová složitost ← v nejhorším případě
$T(n) := \max \{ t(x) \mid x \text{ je vstup velikosti } n \}$

Def: Prostor běhu algoritmu pro vstup X
$s(X) :=$ max. adresa - min. adresa + 1
navštívená během běhu programu

nestačí # navštívených adres
↳ nějaké algoritmy to mohou zneužít

Def: Prostorová složitost
$S(n) := \max \{ s(x) \mid x \text{ je vstup velikosti } n \}$

→ pracovní prostor → z části paměti, kde byl vstup se jenom čte
↳ výstup smíme jenom zapisovat, ne číst

Def: Asymptotická notace $f, g: \mathbb{N} \to \mathbb{R}$ $\qquad \forall^* := \exists n_0 \in \mathbb{N}: \forall n \geq n_0$ – pro ∀ až na konečně mnoho výjimek

$$f \in O(g) \equiv \exists c \, \forall^* n: \; f(n) \leq c \cdot g(n)$$ – f roste nejvýše jako g

$$f \in \Omega(g) \equiv \exists c \, \forall^* n: \; f(n) \geq c \cdot g(n)$$ – f roste alespoň jako g

$$f \in \Theta(g) \equiv \exists c, c' \, \forall^* n: \; c' g(n) \leq f(n) \leq c\, g(n)$$ – f roste stejně jako g

$$\Theta(g) = O(g) \cap \Omega(g)$$

GRAFOVÉ ALGORITMY

- BFS ~ nalejeme do grafu vodu
- DFS ~ Ariadnina nit

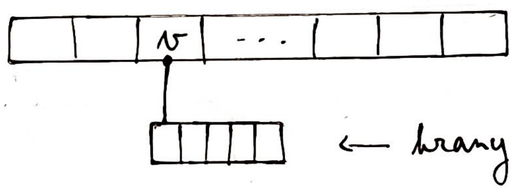
Def: Graf G BÚNO orientovaný, $n := |V(G)|$, $m := |E(G)|$.

- Reprezentace grafu

1. matice sousednosti $\quad A_{ij} := \begin{cases} 1, & v_i v_j \in E(G) \\ 0. \end{cases}$ $\quad \to$ velikost $\Theta(n^2)$

2. seznamy sousedů

vrcholy:



← hrany z v ⟵ vrchol, délka, ⋮

↳ prostor $\Theta(n+m)$

- Prohledávání do hloubky – DFS

$$\to \text{stav}(v) = \begin{cases} \text{nevidený} \\ \text{otevřený} \\ \text{zavřený} \end{cases}$$

$\forall v \in V$: stav(v) ← nevidený

DFS(v):
$O(1)$ [1. stav(v) ← otevřený
2. Pro ∀ vu ∈ E:
3. Pokud stav(u) = nevidený:
4. DSF(u)
$O(1)$ [5. stav(v) ← zavřený

inicializace smyčky

$\Theta(\deg^{OUT}(v))$

👁 změny stavů: N → O → Z

⇒ po doběhnutí není žádný vrchol otevřený

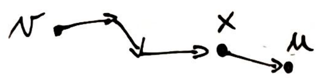
👁 DFS neotevře stejný vrchol vícekrát

⇒ do vrcholu vstoupíme max 1

Lemma: Po doběhnutí DFS(v) je $\forall u \in V(G)$ $\text{stav}(u) = \begin{cases} \text{zavřený} \iff \exists \text{ cesta z } v \text{ do } u \\ \text{neviděný, jinak.} \end{cases}$

Dk: 1) u zavřený ⇒ u dosažitelný   ⟺ u otevřený v zavřený ⇒ dosažitelný

→ Důkaz indukcí podle doby běhu programu – invariant



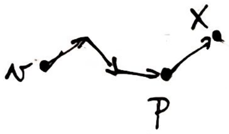
→ u otevřelo x ⇒ podle i.p. je x dosažitelné

⇓ z x vede do u hrana

⇒ z v vede do u sled/cesta ⇒ u dosažitelné

2) u dosažitelný ⇒ u na konci zavřený

→ Důkaz nalezením minimálního protipříkladu   špatný vrchol



pro spor: $\exists x \in V(G)$ t.ž. x je dosažitelný, ale neviděný

⇒ zvolíme x t.ž. cesta z v do x má nejméně hran

⇒ předposlední P je dobrý ⇒ viděný ⇒ otevřený ⇒ objevili jsme x ↯ ■

Lemma: DFS běží v čase $\Theta(n+m)$

Dk:

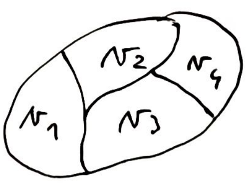
1) $\sum_v \Theta(1 + \deg^{out}(v)) \in \Theta(n+m)$

2) účtujeme vrcholům a hranám → vrcholy: 1, 5 → $\Theta(n)$; hrany: 2, 3, 4 → $\Theta(m)$ } $\Theta(n+m)$

• **Opakované DFS**

1. $\forall v \in V(G)$: stav(v) ← neviděn
2. Pro $\forall v \in V(G)$:
3. Pokud stav(v) = neviděn:
4. DFS(v)

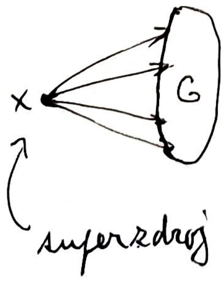
← navštívím celý graf + stále $\Theta(n+m)$



→ pro ∀ podgraf to je $\Theta(n'+m')$

⇒ posčítá se to na $\Theta(n+m)$

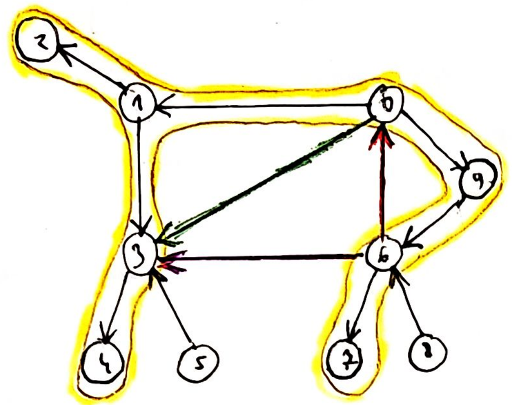
↳ nebo alternativně:



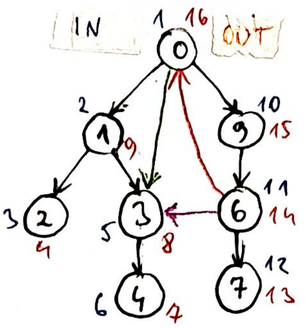
DFS(x) ⇒ $\Theta(n+m)$ snadno vidět

• **DFS strom grafu**

– získáme kostru dosažitelné části grafu



DFS(0) hrany jdoucí z jednoho vrcholu mají prioritu podle čísla cílového vrcholu



podle pořadí navštívení

• stromové hrany – v okamžiku jejich objevení
• zpětné – vede do předka
• dopředné – vede do zavřeného potomka
• příčné – vede do navštíveného vrcholu co není předek ani potomek

– alg DFS umět klasifikovat hrany, tak ho musíme upravit

∀ příčná hrana vede ← do minulosti

inicializace

$\forall v \in V(G)$: stav$(v) \leftarrow$ neviděný, in(v), out$(v) \leftarrow \emptyset$ – nedef.

$T \leftarrow 0$ ~~čas~~ – hodinky tiknou při změně stavu

DFS(v):
1. stav$(v) \leftarrow$ otevřený
2. $T \leftarrow T+1$, in$(v) \leftarrow T$
3. Pro $vu \in V(G)$:
4. Pokud stav(u) = neviděný:
5. DFS(u)
6. stav$(v) \leftarrow$ zavřený
7. $T \leftarrow T+1$, out$(v) \leftarrow T$

→ otevírání a zavírání ~ závorkování

$(_0 (_1 (_2)_2 (_3 (_4)_4)_3)_1 (_9 (_6 (_7)_7)_6)_9)_0$

→ in(v) = pozice $(_v$

out(v) = pozice $)_v$

• **Klasifikace hran pomocí závorek**

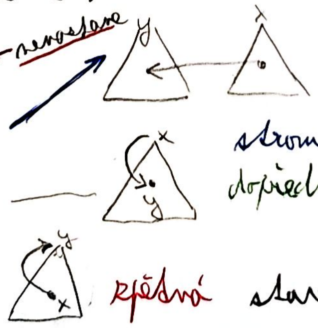
→ pro hranu $xy \in E(G)$:

1, $(_x)_x \; (_y)_y$ ← nenastane

2, $(_y)_y \; (_x)_x$

3, $(_x (_y)_y)_x$

4, $(_y (_x)_x)_y$



příčná stav(y) = zavřený

stromová stav(y) = neviděn

dopředná stav(y) = zavřený

zpětná stav(y) = otevřený při objevení

⇒ o jakékoli hraně umíme v $O(1)$ zjistit, kterého druhu je

→ v neorientovaném grafu

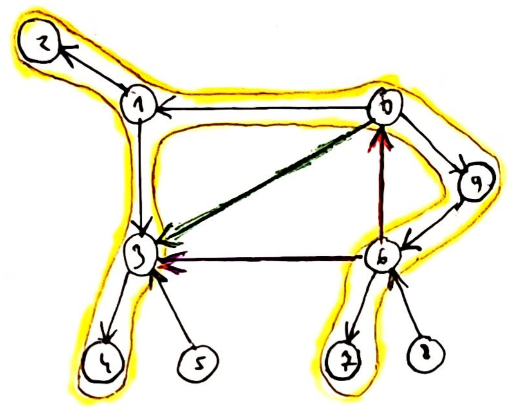
hrana xy spatřena

poprvé		podruhé
stromová		zpětná
zpětná		dopředná

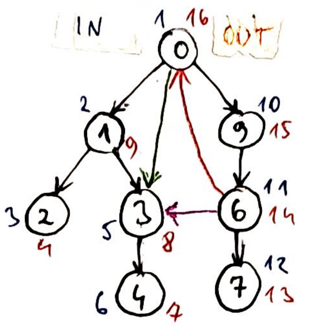
→ nejsou tu příčné hrany

• DFS strom grafu

- získáme kostru dosažitelné části grafu



DFS(0) hrany vedoucí z jednoho vrcholu mají prioritu podle čísla cílového vrcholu



→ podle pořadí navštívení

- • stromové hrany – v okamžiku jejich objevení
- • zpětné – vede do předka
- • dopředné – vede do zavřeného potomka
- • příčné – vede do navštíveného vrcholu co není předek ani potomek

- aby DFS uměl klasifikovat hrany, tak ho musíme upravit

∀ příčná hrana vede ← do minulosti

inicializace

$\forall v \in V(G)$: stav(v) ← neviděný, in(v), out$(v) \leftarrow \emptyset$ (nedef.)

$T \leftarrow 0$ ~~čas~~ – hodinky tiknou při změně stavu

DFS(v):
1. stav(v) ← otevřený
2. $T \leftarrow T+1$, in$(v) \leftarrow T$
3. Pro $vu \in V(G)$:
4. Pokud stav(u) = neviděný:
5. DFS(u)
6. stav(v) ← zavřený
7. $T \leftarrow T+1$, out$(v) \leftarrow T$

→ otevírání a zavírání ~ závorkování

$$(_0 (_1 (_2)_2 (_3 (_4)_4)_3)_1 (_9 (_6 (_7)_7)_6)_9)_0$$

→ in(v) = pozice $(_v$

out(v) = pozice $)_v$

• Klasifikace hran pomocí závorek

→ pro hranu $xy \in E(G)$:

1, $(_x)_x (_y)_y$ ← nevznikne  příčná stav(y) = zavřený

2, $(_y)_y (_x)_x$ stromová stav(y) = neviděn

3, $(_x (_y)_y)_x$  dopředná stav(y) = zavřený

4, $(_y (_x)_x)_y$  zpětná stav(y) = otevřený při objevení

→ o jakékoli hraně umíme v $O(1)$ zjistit, kterého druhu je

→ v neorientovaném grafu

hrana xy zpětná

poprvé		podruhé
stromová		zpětná
zpětná		dopředná

→ nejsou tu příčné hrany

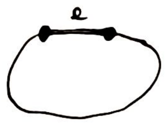
Věta: DFS(v) běží v čase $\Theta(n+m)$ a prostoru $\Theta(n+m)$ a najde dosažitelné vrcholy a klasifikaci hran mezi nimi.

• Hledání mostů v neorientovaných grafech

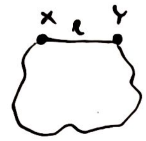
Def: Hrana e je **most** v $G \equiv G-e$ má více komp. souv. než G.

e není most $\Leftrightarrow$ e leží na kružnici

$\Leftarrow$



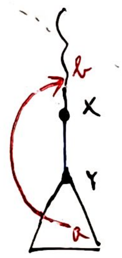
$\Rightarrow$



v $G-e$ $\exists$ cesta mezi x, y
$\to$ cesta + e = kružnice

Zpětná hrana nemůže být most.

$\Rightarrow$ Všechny mosty jsou stromové hrany $\to$ kdy leží s. hrana na kružnici?



stromová hrana xy není most

$\Updownarrow$

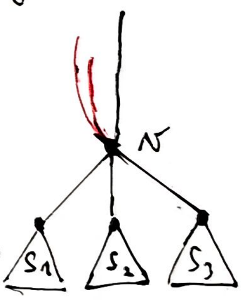
$\exists$ zpětná hrana ab, kde a leží v $T(y)$ a b leží ostře nad y $\Leftrightarrow$ $low(y) < in(y)$

strom pod y

$\to$ říká jak vysoko jsem schopen dosáhnout z $T(v)$

Def: $low(v) := \min\{in(b) \mid ab \text{ je zpětná hrana s } a \in T(v)\}$

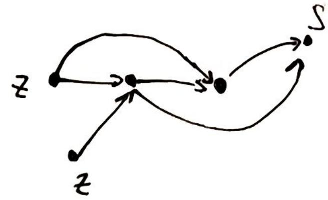
$\Rightarrow$ chci spočítat $low(v)$ pro $\forall v \in V$ v lineárním čase



$\Theta(n+m)$ $low(v) = \min\{low(s_1), \dots low(s_k), in(b) \mid vb \text{ zpětné hrany}\}$

• Acyklické orientované grafy – DAGy

Def: Zdroj je vrchol s $\deg^{IN}(-) = 0$.
Stok je vrchol s $\deg^{OUT}(-) = 0$.



Lemma: Každý konečný DAG má alespoň 1 zdroj a stok.

Dk: Pro stok: vyberu vrchol v → je stok ✓
↳ není stok ⇒ vede z něj hrana do v_1

1) narazíme na stok ✓
2) pokračujeme do ∞ → ↯ graf je konečný
3) stok nenajdeme ⇒ kružnice ↯

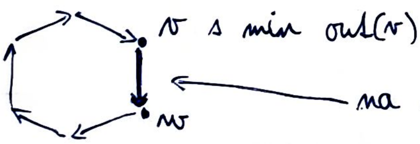
Pro zdroj ▨

Def: Transponovaný graf G^T := graf vzniklý z G otočením šipek.

👁 Transpozice cyklu je cyklus ⇒ G je DAG ⟺ G^T je DAG. } najdu stok v G^T ⇒ zdroj v G
👁 v je zdroj v G ⟺ v je stok v G^T

Alg: Je G DAG? ⟺ Má G cyklus?
👁 $\forall$ zpětná hrana leží na cyklu

Lemma: Na každém cyklu leží alespoň 1 zpětná hrana.
Dk: Na cyklu si vyberu vrchol s nejmenším $out(v)$



na vw roste out ↗ jediný typ hran, kde roste out jsou zpětné hrany ⇒ vw je zpětná ▨

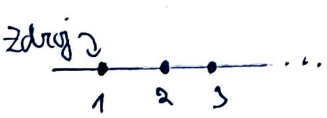
Věta: Graf je DAG ⟺ opakované DFS nenajde zpětnou hranu.

• Topologické uspořádání

↗ $\forall$ dva prvky jsou porovnatelné

Def: Topologické uspořádání grafu $G=(V,E)$ je lineární uspořádání $\preccurlyeq$ na V t.ž.
$$\forall uv \in E: u \preccurlyeq v.$$
Topologické očíslování = očíslování vrcholů podle t. usp.

Věta: Graf G má T.U. ⟺ G je DAG.
Dk: ⇒: má T.U. ⇒ nemá cyklus ⇒ je DAG
⇐: postupným odtrháváním zdrojů ↗ libovolný zdroj klidně může být první ⇒ utrhnu ho ⇒ další zdroj



Tohle běží $O(n \cdot m)$, ale jde to lineárně
↳ pamatuju si někde zdroje ↗ ▨

👁 G má jednoznačné T.U. ⟺ tenhle alg. má v $\forall$ kroku jen 1 zdroj na výběr.

Věta: Opakované DFS opouští vrcholy v opačném topologickém uspořádání.

Dk: out(v) očísluje vrcholy ⇒ lineární uspořádání

⇒ sestrojíme opačné a ukážeme, že je topologické

- Pro $\forall uv \in E$ chceme: $u \preccurlyeq v \equiv out(u) > out(v)$ ⟺ na uv klesá out
  ⟺ uv není zpětná

  ⇒ v DAGu zpětné hrany nejsou ⇒ tohle můžu udělat pro každou hranu ■
- opakované DFS najde T.U. v čase $\Theta(n+m)$

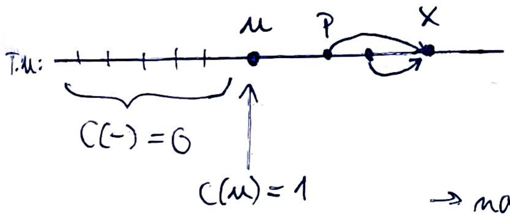
Využití T.U.: Linearizace částečného uspořádání.

↳ Č.U. ~ DAG kde vrcholy jsou prvky a hrany nerovnosti ⇒ doplníme to na lineární

Alg: Počet cest z u do v (v DAGu). ← $C(v)$

→ počítáme $C(x) :=$ # cest z u do x

→ indukcí podle T.U. → předchůdci x leží před x ⇒ z i. předpokladu už známe jejich $C(p)$

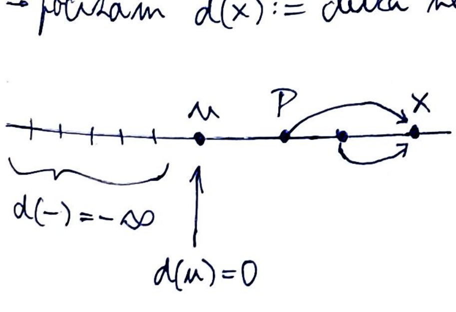


$$C(x) = \sum_{px \in E} C(p)$$

→ najdu T.U. + konstantní čas na vrchol i na hranu ⇒ $\Theta(n+m)$

Alg: Délka nejdelší (nejkratší) cesty z u do v (v DAGu).

→ počítám $d(x) :=$ délka nejdelší cesty z u do x



$$d(x) = 1 + \max\{d(p) \mid px \in E\}$$ → $\Theta(n+m)$

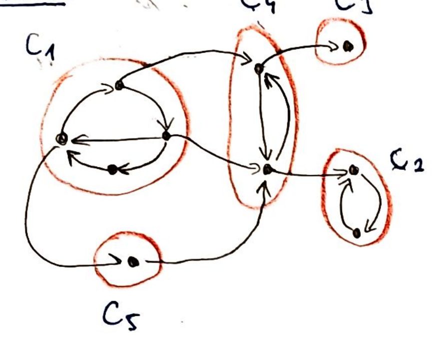
• Komponenty silné souvislosti

Def: Nechť $G=(V,E)$ je orientovaný graf. Definuji $\rightarrow, \leftrightarrow$ relace na V:

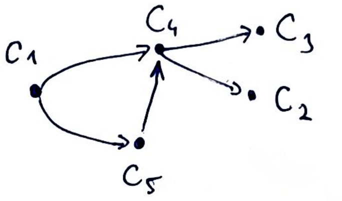
$$\left.\begin{aligned} u \rightarrow v &\equiv \exists \text{ cesta z } u \text{ do } v \\ u \leftrightarrow v &\equiv u \rightarrow v \ \& \ v \rightarrow u \end{aligned}\right\} \leftrightarrow \text{ je ekvivalence}$$

Def: Komponenty silné souvislosti jsou podgrafy indukované ekvivalenčními třídami $\leftrightarrow$.

Příklad



→ když cyklus komunikuje se svým okolím jen jednosměrně, tak tvoří komponentu



graf komponent

Def: Pro o. graf G definujeme graf komponent $K(G)$:

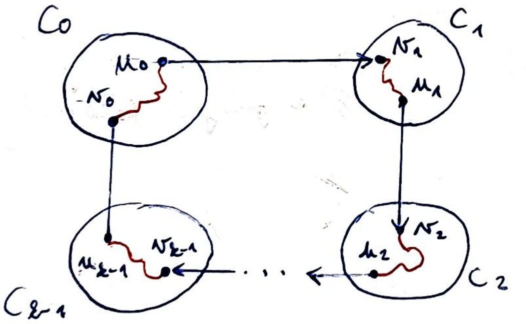
$V(K(G)) :=$ komponenty silné souvislosti G

$C_1C_2 \in E(K(G)) \equiv \exists v_1 \in C_1,\ v_2 \in C_2 : v_1v_2 \in E(G)$

} $K(G)$ lze z G vytvořit posloupností kontrakcí hran mezi vrcholy ve stejné komponentě

Věta: Pro každý G je $K(G)$ DAG.

Dk: Kdyby v $K(G)$ byl cyklus $C_0C_1 \dots C_{k-1}C_0$ ← indexy mod k



→ z definice $K(G)$

$\forall i\ \exists u_i \in C_i,\ v_{i+1} \in C_{i+1} : u_i v_{i+1} \in E(G)$

$\forall i\ \exists$ cesta z u_i do v_i uvnitř C_i

$\Rightarrow$ všechna $u_i v_i$ leží na cyklu v G

$\Rightarrow C_0, \dots C_{k-1}$ nejsou různé komponenty ↯ ■

Důsledek: Můžeme rozlišovat zdrojové a stokové komponenty.

Pozorování:

① Je-li C stoková komponenta a $v \in C$, pak DFS(v) projde přesně C. ← umím najít stokovou komponentu?

② Vrchol s min. outem leží ve stokové komponentě ← v DAGu!

③ Vrchol s max. outem leží ve zdrojové komponentě ← v obecném grafu } Pro DAGy z T.U.

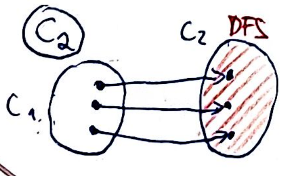
④ Transpozice otočí relaci $\rightarrow$, ale $\leftrightarrow$ neovlivní $\Rightarrow$ zachovává komp. s.s. $\Rightarrow$ v $K(G^T)$ se otočí šipky

③ $\Rightarrow K(G^T) \cong K(G)^T \Rightarrow$ najdu $v \in$ zdroj. k. v $G^T \Rightarrow v \in$ stok. k. v G

Lemma: Pokud $C_1C_2 \in E(K(G))$, pak $\max\limits_{u \in C_1}(\text{out}(u)) > \max\limits_{v \in C_2}(\text{out}(v))$.

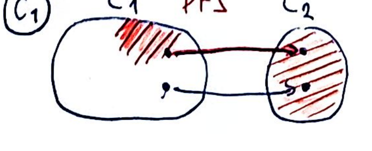
Dk: Opakované DFS vstoupí dřív do

(C_2)



$(_2\ )_2\ (_1\ )_1$

(C_1)



$(_1\ (_2\ )_2\ )_1$ ■

Při procházení vrcholů podle klesajících outů, tak C_1 potkám před C_2 $\Rightarrow$ komponenty objevujeme od zdrojových k stokovým

• **Algoritmus na komponenty silné souvislosti**

1. sestroj G^T] $\Theta(n+m)$
2. $Z \leftarrow$ prázdný zásobník
3. opakované DFS na G^T, při zavírání v jej přidáme do Z.] $\Theta(n+m)$
4. $\forall v$: $komp(v) \leftarrow \emptyset$, $k = 0$
5. Postupně odebíráme vrcholy v ze Z:
6. Pokud $komp(v) = \emptyset$:
7. $komp(v) \leftarrow k{+}{+}$
8. DFS(v) v G, nechodíme do vrcholů s komp $\neq \emptyset$ navštíveným vrcholům nastavujeme $komp(-) \leftarrow komp(v)$

(kroky 5.–8.: $\Theta(n+m)$)

v Z budu mít klesající outy na G^T $\Rightarrow$ zdrojové k. G^T $\Rightarrow$ stokové k. G ↳ pustíme na ně DFS

Věta: komponenty silné souvislosti lze najít v čase i prostoru $\Theta(n+m)$.

• **Nejkratší cesty**

→ orientovaný graf $G = (V, E)$ s délkami hran $l : E \to \mathbb{R}^+_0$

Def: Délka sledu s: $l(s) := \sum_{e \in E(s)} l(e)$

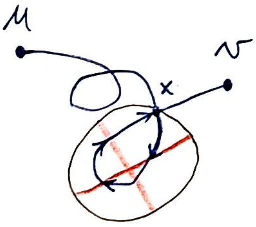
Def: Vzdálenost z u do v: $d(u,v) := \min\{l(p) \mid p$ je cesta (sled) z u do $v\}$, $d : V^2 \to \mathbb{R}^+_0 \cup \{\infty\}$

→ d není symetrická ⇒ není to metrika

Lemma: Pro $\forall$ uv sled S existuje uv cesta P t. ž. $l(P) \leq l(S)$.

→ min přes sledy je stejné jako přes cesty

Dk:

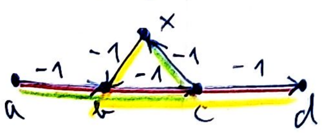


→ z S odstraníme cykly ⇒ # hran klesne ⇒ $l(S)$ nevzroste } nakonec dostanu cestu

Lemma (Δ-nerovnost): $\forall u, v, w$: $d(u,v) \leq d(u,w) + d(w,v)$

Dk: vezmu nejkratší uw a wv cesty ⇒ slepením uv sled délky ⇒ $\exists$ uv cesta, co není delší ☒

→ **Co když připustíme $l < 0$?**



⇒ rozbije se to Lemma ⇒ rozbije se Δ-nerovnost

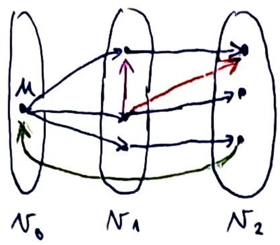
$$d(a,d) \leq d(a,x) + d(x,d)$$
$$-3 \leq -3 + -3$$

neexistuje nejkratší ad-sled
nejkratší cesty jsou divné
↳ neumíme je najít polynomiálně

→ problém jsou záporné cykly ⇒ pak ně neplatí to Lemma

• Prohledávání do šířky – BFS

- BFS(u)
- vrstvy:



hrany — stromové → vytváří strom nejkratších cest
↳ obsahuje nejkratší cesty z u do všech ostatních vrcholů
— příčné
— zpětné
— dopředné příčné

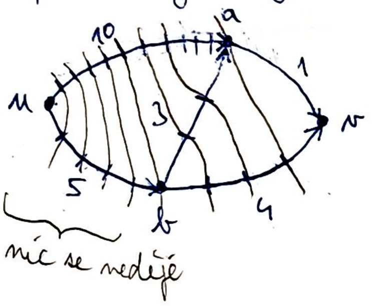
⚠ Neexistuje hrana, která by vedla o víc než 1 vrstvu dopředu. $\Theta(n+m)$

- fronta: | rozebírám v_i | vytvářím v_{i+1} |

- $d(v) :=$ číslo vrstvy nebo $\emptyset$ ⇒ ⚠ $d(v) = d(u,v)$ → pro hrany délky 1
- $p(v) :=$ předchůdce v
  ↳ z předchůdců lze sestrojit strom nejkratších cest

• BFS s podrozdělováním hran

→ pro hrany délky $l \in \mathbb{N}$ → hrany podrozdělíme na jednotkové hrany & BFS



$\Theta(n + mL)$
↳ $\max l(e)$

⇒ pomalé je to čekání na dlouhých hranách
⇒ přeskočíme to ⇒ Dijkstra pro $\mathbb{N}_0$ → $\mathbb{Q}_0^+$

• Dijkstrův algoritmus

stav(v) ... o, z, n
h(v) ... hodnota

Inicializace:

$\forall v$: stav(v) ← neviděn, $h(v) \leftarrow +\infty$, $p(v) \leftarrow \emptyset$
stav(u) ← otevřený, $h(u) \leftarrow 0$, Insert(u)] $\Theta(n)$

projdeme nejvýše n-krát

1. Dokud $\exists v$: stav(v) = otevřený & h(v) je minimální: ExtractMin] $n \cdot T_X$
2. Pro $\forall vw \in E$:
3. Pokud $h(w) > h(v) + l(vw)$:
4. $h(w) \leftarrow h(v) + l(vw)$ → neviděný Insert(w)
5. stav(w) ← otevřený → otevřený Decrease(w)
6. $p(w) \leftarrow v$
7. stav(v) ← zavřený

$n \cdot T_I + m \cdot T_D$

→ z BFS plyne, že pro kladné délky hran nikdy neotevřu zavřený vrchol

⇒ každý vrchol zavřu právě jednou ← pro $l \in \mathbb{R}_0^+$!

Věta: Dijkstrův algoritmus v poli spočte $d(u,*)$ v čase $\Theta(n^2)$.

- **Složitost Dijkstry**: $O(n \cdot T_I + n \cdot T_X + m \cdot T_D)$

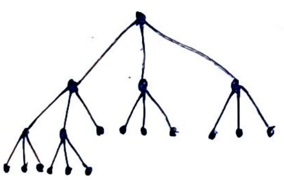
		pole	halda	fibbonacciho halda	d-reg. halda
Extract Min	T_X	n	$\log n$	$\log n$	$d \cdot \log_d(n)$
Insert	T_I	1	$\log n$	1	$\log_d(n)$
Decrease	T_D	1	$\log n$	1	$\log_d(n)$
O		n^2	$(n+m)\log n$	$m + n \log n$	$\frac{m \cdot \log(n)}{\log(m/n)} \in O(m \log n)$
řídké: $m \in O(n)$		n^2	$n \log n$	$n \log n$	$n \cdot \log n$
husté: $m \in O(n^2)$		n^2	$n^2 \log n$	n^2	$n^2 = m$

↳ rychlé pro řídké i husté

- **Binární halda** = úplný bin. strom
  - → haldové uspořádání: $\forall uv \in E: h(u) \leq h(v) \Rightarrow$ kořen = min
  - Extract Min → prohodit kořen a poslední prvek ⇒ bublání z kořene dolů
  - Insert → přidat na konec + bublání nahoru } Insert = Decrease
  - Decrease → snížit hodnotu + bublání nahoru } Extract Min = Increase

  ↳ musím si někde v poli pro každý vrchol pamatovat jeho index v haldě

- **d-reg. halda**



$\Big\}\ h = \log_d n$

decreasi uděláme nejvíc ⇒ chceme ji zrychlit

- bublání nahoru → $\log_d n = \frac{\log n}{\log d}$ → rychlejší
- bublání dolů → $d \cdot \log_d n = d \cdot \frac{\log n}{\log d}$ → pomalejší

DA: $O\left(n \cdot \frac{\log n}{\log d} + n \cdot \frac{d \cdot \log n}{\log d} + m \cdot \frac{\log n}{\log d}\right) = O\left(\log n \left(\frac{n \cdot d}{\log d} + \frac{m}{\log d}\right)\right)$

⇒ chceme minimalizovat $O\left(\underbrace{\frac{n \cdot d}{\log d}}_{\text{roste}} + \underbrace{\frac{m}{\log d}}_{\text{klesá}}\right) \Rightarrow n \cdot d = m \Rightarrow d = \frac{m}{n} \Rightarrow O\left(\frac{m}{\log m/n}\right)$

⇒ zvolím $d = \max\left(\left\lceil \frac{m}{n} \right\rceil, 2\right)$ $\Rightarrow O\left(\frac{m \log n}{\log(m/n)}\right) \in O(m \log n)$

⇒ něco mezi: $m \in O(n^{1+\varepsilon}) \Rightarrow O\left(\frac{m \cdot \log n}{\log(n^{\varepsilon})}\right) = O\left(\frac{m \log n}{\varepsilon \cdot \log(n)}\right) = O(m)$

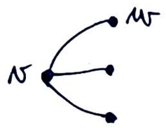
- **Správnost Dijkstry**
  - → ukázali jsme pro $l \in \mathbb{Q}_0^+$
  - ⇒ odvodíme správnost pro obecnější algoritmus s $l \in \mathbb{R}_0^+$

• Relaxační algoritmy

① udržujeme ohodnocení vrcholů $h: V \to \mathbb{R}^*$

- inicializace: $h(*) \leftarrow \infty$, $h(u) \leftarrow 0$
- cíl: $\forall v: h(v) = d(u,v)$

② relaxace v



$\forall vw \in E$: pokusíme se snížit $h(w)$ na $h(v) + \ell(vw)$

③ stavy vrcholů

- nevidění $\Rightarrow h(v) = +\infty$
- otevřený $\rightarrow$ od poslední změny $h(v)$ nebyl relaxován
- zavřený $\rightarrow$ od poslední relaxace nebyla $h(v)$ změněna

Inicializace:

$h(*) \leftarrow \infty$, stav$(*) \leftarrow$ neviděný
$h(u) \leftarrow 0$, stav$(u) \leftarrow$ otevřený

1. Dokud $\exists v$ otevřený:
2. relaxujeme v, při změně $h(w)$:
   stav$(w) \leftarrow$ otevřený
3. stav$(v) \leftarrow$ zavřený

Invariant O (ohodnocení):

1, $h(v)$ nikdy neroste ✓

2) Pokud $h(v) < \infty$, tak
$\exists$ uv-sled délky $h(v)$

Dk: indukcí podle doby běhu
$\Rightarrow$ už mám uv-sled, chci uw-sled

$h(w) = h(v) + \ell(vw)$ ∎

Lemma D (dosažitelnost): Když se alg. zastaví, pak $h(v) < \infty \iff v$ je dosažitelný z u.

Dk: $\Rightarrow$: z O. $\Leftarrow$: minimální protipříklad
↳ vezmu nejbližší co do počtu hran špatný vrchol x

P dobrý $\Rightarrow$ otevřen $\Rightarrow$ relaxován $\Rightarrow x$ objeveno ↯ ∎

Lemma V (vzdálenost): Po zastavení: $\forall v$: $h(v) = d(u,v)$.

→ [illegible] se $h(v) < d(\dots)$

Dk: BÚNO v dosažitelný (viz D). v je špatný $\equiv$ v je dosažitelný & $h(v) > d(u,v)$.

$\Rightarrow$ ze všech nejbližších špatných vrcholů si vyberu ten, který je nejbližší co do # hran $\to x$ ↯

👁 $x \neq u \Rightarrow \exists P$:

P dobrý $\Rightarrow$ otevřen $\Rightarrow$ relaxován $\Rightarrow$ pak $h(x) \leq \underbrace{h(P)}_{d(u,P)} + \ell(Px) = d(u,x)$

Věta: Pokud relaxační algoritmus doběhne, tak spočte správné vzdálenosti. Přičemž doběhne v grafech bez záporných cyklů.

- <u>Dijkstra je relaxační</u>

→ v kroku 1 vybíráme v s.t. $h(v)$ je minimální

<u>Invariant M (monotonie)</u>: Pro graf s <u>nezápornými</u> délkami hran platí:

① $\forall o$ otevřený, $\forall z$ zavřený: $h(z) \leq h(o)$ ⇒ zavřené $\leq h(v) \leq$ otevřené < neviděné

② $h(z)$ zavřeného z se nezmění ⇒ žádný vrchol neotevřeme dvakrát

<u>Dk</u>: indukcí podle # relaxací ~ podle doby běhu.

①, ② platí a relaxujeme v → hrana do w

② w zavřené: $h(w) \leq h(v)$ ⇒ $h(w)$ neměním

① jinak: předtím měním $h(w)$, tak $\underbrace{h(w)}_{\text{otevřený}} = h(v) + l(vw) \geq h(v) \geq$ zavřené ∎

⇒ pak v zavřu a přidám k zavřeným

Když jsou záporné hrany, tak Dijkstra je správný ale může běžet exponenciálně dlouho

záporné cykly ho rozbijí

<u>Důsledek</u>: $\forall v$ je zavřen nejvýše jednou ⇒ # relaxací $\leq n$

⇒ algoritmus se zastaví → podle lemma V je správně

<u>Věta</u>: Dijkstrův alg. uzavírá vrcholy v pořadí podle neklesající $d(u,-)$, v okamžiku zavření je $h(v) = d(u,v)$ a už se <u>nezmění</u>.

- <u>Bellman-Fordův algoritmus</u>

→ relaxační alg., kde zavíráme nejdéle otevřený vrchol ⇒ otevřené vrcholy jsou ve <u>frontě</u>

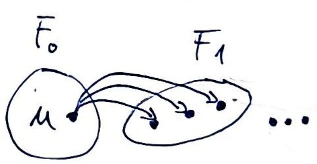
<u>Věta</u>: B.F. alg. spočítá $d(u,*)$ v čase $\underline{O(n \cdot (n+m))}$ pro <u>každý graf</u> bez záporných cyklů.

$O(n \cdot m)$

<u>Def</u>: <u>Fáze běhu</u> $F_0, F_1, \ldots$

- F_0 → otevření u
- F_{i+1} → zavření vrcholů otevřených v F_i

→ 1 vrchol může být ve více fázích



→ až najdu z, tak jej dám na konec

⇒ fronta se zarážkou

<u>Lemma</u>: 1 fáze trvá $O(n+m)$.

$\sum \deg^{out}(v) = m$

<u>Dk</u>: ve fázi je nejvýše n vrcholů ⇒ všechny zavře a relaxuje ⇒ sáhne max na m hran ∎

<u>Invariant</u>: Na konci F_i je $\forall v$ $h(v) \leq$ délky všech uv sledů o <u>nejvýše i hranách</u>.

<u>Dk</u>: indukcí podle i. $i-1 \to i$:

→ pro méně než i hran z I.P.

S = nejkratší uv sled s právě i hranami

- I.P. ⇒ na konci F_{i-1}: $h(p) \leq l(S')$
- nejpozději v F_{i-1} dostalo p tuto $h(p)$ ⇒ otevřeno p
- nejpozději v F_i bylo p zavřeno a relaxováno ← $h(p)$ se mezitím mohla změnit, ale nevzrostla

⇒ takže: $h(v) \leq h(p) + l(pv) \leq l(S') + l(pv) = l(S)$ ∎

<u>Důsledek</u>: Na konci F_n je $\forall v$ $h(v) = d(u,v)$ ⇒ alg. se zastaví po n fázích.

⇒ věta platí ∵ $O(n) \cdot O(n+m) \in O(n \cdot (n+m))$ ∎

• **Minimální kostry** – někdy také nejlehčí

→ Máme neorientovaný souvislý graf G, váhy $w: E \to \mathbb{R}$ BÚNO prosté

→ chceme kostru T grafu G t.ž.

$$w(T) := \sum_{e \in E(T)} w(e) \quad \text{je minimální.}$$

• **Jarníkův algoritmus**

→ hladový algoritmus

1. $T \leftarrow \{u\}$ pro $u \in V$ libovolný
2. Dokud $V(T) \mathrel{!}= V(G)$: ← n-krát
3. $\quad e \leftarrow$ nejlehčí hrana mezi T a zbytkem grafu $]\, m$ $\quad \Theta(n \cdot m)$
4. $\quad T \leftarrow T + e$

Korektnost:

Lemma: J.a. se zastaví a na konci je T kostra.

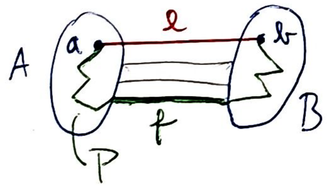
Dk: K T lepíme listy $\Rightarrow$ T je vždy strom + J.a. se zastaví když má všechny V. ■

Def: Elementární řez v G je $R \subseteq E(G) \equiv$

$\exists$ rozklad $E(G)$ na $\{A, B\}$ t.ž. $R = \{ab \in E(G) \mid a \in A, b \in B\} = E(A, B)$

Řezové lemma: Buď G graf s unikátními vahami, R elementární řez v G, e nejlehčí hrana R a T jakákoli min. kostra G. Potom $e \in T$.

Dk: Sporem... T min. kostra, $e \notin T$



$\exists$ cesta P v T mezi a, b

$\Rightarrow \exists f \in P \cap R$, $w(f) > w(e)$

$\Rightarrow T' := T - f + e$ je také kostra, ale $w(T') < w(T)$ ↯ ■

Důsledky:

① J.a. je korektní, protože hrany v každém kroku jsou el. řez $\Rightarrow$ min $\in T$

② Pro unikátní váhy je minimální kostra jediná → protože $T \subseteq$ každé min. kostry

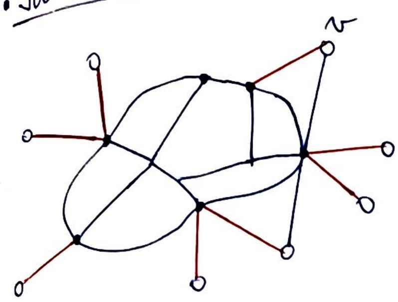
③ Minimální kostra je určena porovnáváním vah, hodnoty nepotřebujeme

$\Rightarrow$ stačí nám na hranách nadefinovat nějaké lineární uspořádání

$\Rightarrow$ když nemáme unikátní váhy hran, tak si hrany očíslujeme

(ale J.a. funguje i pro neunikátní váhy, jen už nefunguje ten důkaz)

Jarník ala Dijkstra



- Pro v sousedící s T si pamatuju **aktivní hranu**

  $a(v) :=$ nejlehčí hrana mezi v a T

  $h(v) := \ell(a(v))$ ... délka aktivní hrany

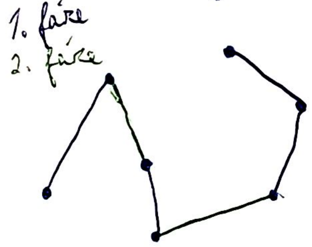
- $\text{stav}(v)$
  - zavřený → uvnitř T
  - otevřený → soused T
  - neviděn → ostatní

Inicializace...

1. Dokud $\exists\, v$ otevřený s min. $h(v)$:
2.   Pro $\forall vw \in E$, w otevřené nebo neviděné:
3.   Pokud $h(w) > \ell(vw)$:
4. $a(w) \leftarrow vw$
5. $\text{stav}(w) \leftarrow$ otevřený
6. $\text{stav}(v) \leftarrow$ zavřený

Složitost:
- pole $O(n^2)$
- halda $O(m \log n)$

Borůvkův algoritmus



Inicializace: $\forall$ vrchol je samostatný stromeček

1 fáze: $\forall$ strom si vybere nejlehčí hranu ven a tyto hrany všechny přidáme $\}$ paralelní J.a.

B.a. najde min. kostru (z řezového lemmatu)

1 fáze trvá $O(n+m) \in O(m)$ (G je souvislý)

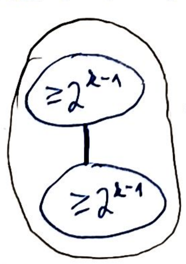
Implementace: očísluju si stromečky a pro všechny hrany se podívám, mezi kterými dvěma vede ⇒ aktualizuju jejich průběžné minimum → $O(m)$

→ na konci fáze musím nějak mergnout ty stromečky → $O(n)$

Lemma: Po k-té fázi má $\forall$ strom alespoň 2^k vrcholů.

Dk: indukcí podle k. $k=0$: $1 \geq 2^0$ ✓

$k-1 \to k$:



$\geq 2^{k-1} + 2^{k-1} = 2^k$ vrcholů ■

Důsledek: $\#$ fází $\leq \log_2(n)$ $\because$ po $\log_2(n)$-té fázi má n vrcholů

Věta: Borůvkův algoritmus najde minimální kostru v čase $O(m \cdot \log n)$.

• **Kruskalův algoritmus**

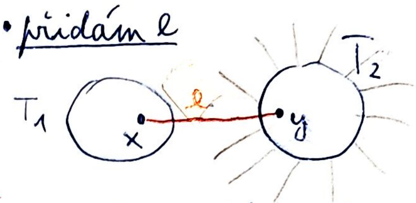
$m \in O(n^2)$

1. setřídíme brany podle vah] $O(m \log m)$ $\in O(m \log n)$
2. $T \leftarrow \{v \mid v \in V\}$ ... triv. les
3. Pro $e \in E$ od nejlehčí po nejtěžší:
4. Pokud $T+e$ je acyklický: Find] m-krát
5. $T \leftarrow T+e$ Union] n-krát

$e = xy \Rightarrow$ pokud x, y ve stejné komp., $T+e$ má cyklus

• **Korektnost**: z řezového lemmatu

• **přidám e**



protože brany mám seřazené

$\Rightarrow$ e je nejlehčí v řezu mezi T_2 a $G \setminus T_2$

$\Rightarrow e \in$ min. kostra

• **nepřidám e** $\rightarrow$ tvořilo by cyklus

$\hookrightarrow T \subseteq$ min. k. $\Rightarrow$ e nepotřebuju

• **Union-Find problem**

• Find(x, y) = jsou x, y v téže komponentě?
• Union(x, y) = přidej branu xy

• **časová složitost**:

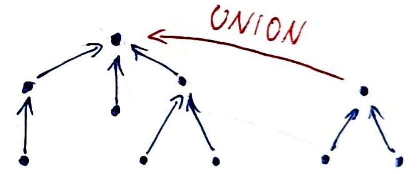
$O(m \cdot \log n + m \cdot T_F + n \cdot T_U)$

① **primitivně**

$\rightarrow \forall$ vrchol si pamatuje ID komponenty

Find $O(1)$, Union $O(n)$ } $O(m \log n + n^2)$

② **rychleji**



komponenta ~ strom orientovaný do kořene

$\rightarrow$ všechny vrcholy si pamatují svého rodiče

• Find: vystoupá do kořenů a porovná je
• Union: naváže kořen 1 na kořen 2

} oba v čase $O(\text{hloubka})$

$\rightarrow$ kořeny si pamatují hloubku

$\Rightarrow$ připojíme mělčí pod hlubší

$\Rightarrow$ hloubka vzroste nejvýše o 1

👁 strom hloubky h má alespoň 2^h vrcholů (indukcí)

$\Rightarrow$ hloubka $\leq \log_2 n$ $\Rightarrow$ U, F v čase $O(\log n)$

Věta: Kruskalův algoritmus najde minimální kostru v čase $O(m \cdot \log n)$.

• Datové struktury

• abstrakce uložení dat < statické – vytvoření, dotazy / dynamické – navíc i úpravy

• rozhraní – poskytuje operace nad daty

• implementace – jak jsou data uložena, jak provádíme operace

• Fronta

	Enqueue	Dequeue	
spojový seznam	$O(1)$	$O(1)$	
pole – hloupě	$O(1)$	$O(n)$	← read index
pole – cyklicky	$O(1)$	$O(1)$	← read, write index

} pro n prvků pole délky $\geq n+1$

• Množina

→ univerzum prvků $\mathcal{U}$ → BÚNO předpokládejme, že s prvky lze pracovat v $O(1)$ (copy, compare, …)

→ chceme reprezentovat $X \subseteq \mathcal{U}$, $n := |X|$

→ operace:

bin. vyhledávání, ale musíme posunout prvky

	Seznam	Pole	Setříděné pole	Vyhledávací strom
• Find (Member) ~ $a \in X$?	$\Theta(n)$	$\Theta(n)$	$\Theta(\log n)$	$\Theta(\log n)$
• Insert ~ pokud $a \in X$, tak nic	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(\log n)$
• Delete ~ pokud $a \notin X$, tak nic	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(\log n)$
• Build			$\Theta(n \log n)$	

↳ fajn statická struktura

• Uspořádaná množina – přes vyhl. strom / setříděné pole

→ operace navíc: Min, Max, Pred, Succ } $\Theta(\log n)$ → hash. tabulka v $\Theta(n)$

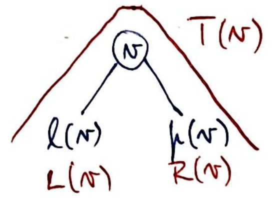
• Slovník

→ univerzum klíčů } slovník $:= \{(k, v)\}$
→ univerzum hodnot }

→ Find vrací hodnotu, Insert přidá klíč s hodnotou

→ množina se dá snadno předělat na slovník

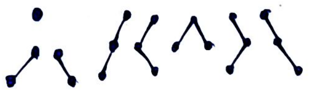
• **Binární strom** := zakořeněný strom, rozlišujeme levého a pravého syna



$T(v)$:= podstrom v a všech jeho tranzitivních potomků

$L(v) := T(l(v))$ & $R(v) := T(r(v))$

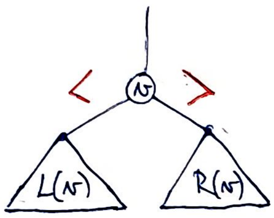
$h(v)$:= hloubka $T(v)$ v hranách



$\Rightarrow$ když syn chybí: $\left.\begin{array}{l} l(v) = \emptyset \Rightarrow L(v) = \emptyset \\ r(v) = \emptyset \Rightarrow R(v) = \emptyset \end{array}\right\} h(\emptyset) := -1$

• **Binární vyhledávací strom** – BVS

→ klíče $k(v) \in U$



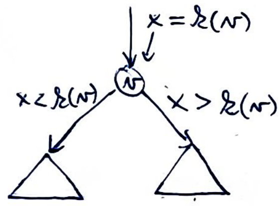
Podmínka stromu:

$\left.\begin{array}{l} \forall v: \forall a \in L(v): k(a) < k(v) \\ \quad\;\; \forall b \in R(v): k(b) > k(v) \end{array}\right\} \Rightarrow$ klíče jsou **unikátní**

→ inorder $(L(v), v, R(v))$ průchod vypíše klíče v rostoucím pořadí

Operace:

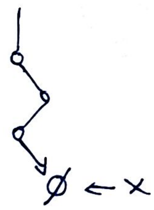
• **Find(x)**



1) najdu $x \Rightarrow v$

2) najdu $\emptyset \Rightarrow x \notin T(v)$

• **Insert(x)**



Find (x)

1) najdu $x \Rightarrow$ nic nedělám

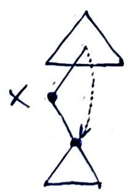
2) najdu $\emptyset \Rightarrow$ založím nový list

• **Delete(x)**

① x je list

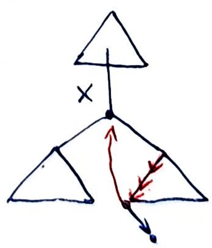
Find(x)
remove

② x má 1 syna



syna navěsíme

③ x má 2 syny



nebo max(L(x))

najdu min(P(x)) → jdu doleva

→ x nahradím tím $m := \min(P(x))$

→ Delete(m) → $r(m) = \emptyset \Rightarrow$ ①

↳ $r(m) \neq \emptyset \Rightarrow$ ②

• **složitost**

→ složitost všech operací je Θ(hloubka stromu)

$\Rightarrow$ chceme vyvážené stromy $h = \log(n)$

$\Rightarrow$ potom všechno $\Theta(\log n)$

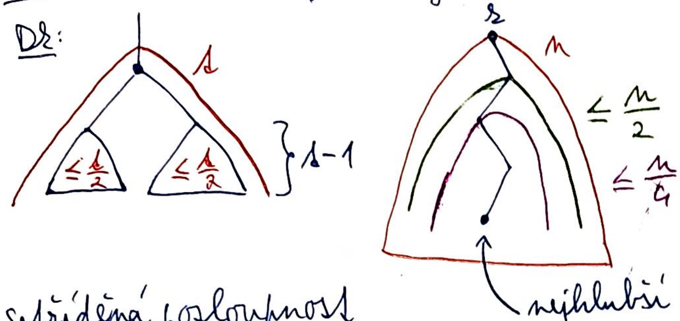
degenerovaný strom

• Dokonale vyvážený strom

Def: pro $\forall v$ $|L(v) - P(v)| \leq 1$.

Tvrzení: Dokonale vyvážený BVS na n vrcholech má hloubku $\leq \log_2(n)$.

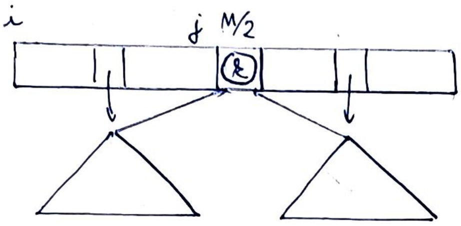
Dk:



délka cesty k nejhlubšímu listu je logaritmická

$\Rightarrow h \leq \log_2(n)$ ∎

setříděná posloupnost $\xrightarrow{O(n)}$ D.V. BVS



→ jako kořen zvolím prostřední prvek

⇒ rekurzivní fce Build(i, j):

1. najdi střed → k
2. $L(k) \leftarrow$ Build(i, k-1)
3. $R(k) \leftarrow$ Build(k+j, j)
4. return k

⇒ D.V. BVS lze vytvořit ⇒ vždy existuje

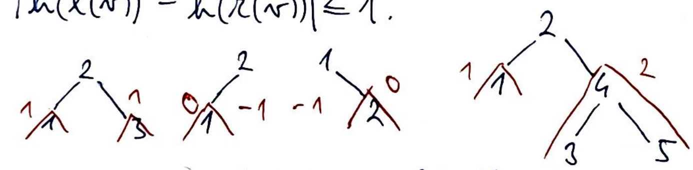
→ Insert a Delete nejdou líp než lineárně

⇒ je to dobrá statická, ale špatná dynamická datová struktura

• AVL strom

Def: pro $\forall v$: $|h(\ell(v)) - h(r(v))| \leq 1$.

Příklady:



Věta: AVL strom na n vrcholech má hloubku $\Theta(\log n)$.

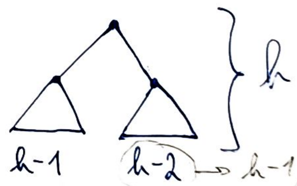
Dk: $A_h :=$ min. # vrcholů pro hloubku h.

$A_0 = 1$

$A_1 = 2$

$\vdots$

pro $h > 1$ uvážíme minimální strom (co do $|V(T)|$) s hloubkou h



$h-2$ → $h-1$/$h-2$, ale $A_{h-2} < A_{h-1}$

$\Rightarrow A_h = 1 + A_{h-1} + A_{h-2}$ ⇝ Fibonacci

Tvrzení: $A_h \geq 2^{h/2} = \sqrt{2}^h$

Dk: indukcí podle h: $h-1 \to h$: $A_h = 1 + A_{h-1} + A_{h-2} > \sqrt{2}^{h-1} + \sqrt{2}^{h-2} = \sqrt{2}^h \overbrace{\left(\frac{1}{\sqrt{2}} + \frac{1}{2}\right)}^{>1} > \sqrt{2}^h$ ∎

Důsledek: $A_h \geq c^h$, $c = \sqrt{2}$

$\Rightarrow$ hloubka $h \leq \log_c(A_h) \leq \log_c(n) \Rightarrow h \in O(\log n)$

$B_h :=$ max. # vrcholů pro hloubku h

$B_0 = 1$

$B_1 = 3$

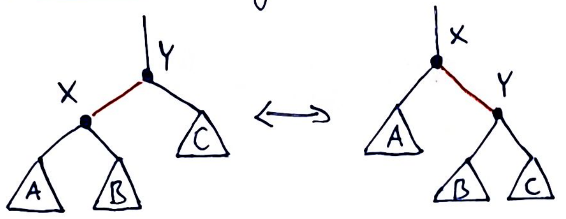
$B_h = 2 \cdot B_{h-1} + 1 = 2^{h+1} - 1$

$\Rightarrow \log(\underbrace{B_h + 1}_{\geq n}) - 1 = h \Rightarrow h > \log(n) - 1 \Rightarrow h \in \Omega(\log n)$

$\Rightarrow h \in \Theta(\log n)$ ∎

• Vyvažování AVL stromu

① rotace hrany

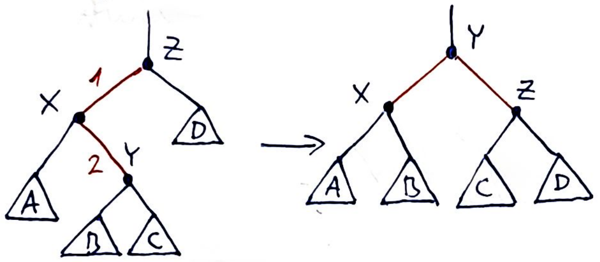


1. Prohoď X a Y
2. Přiřaď jim A, B, C tak, aby bylo zachováno uspoř.

⇒ jednoznačné převěšení podstromů

h(A) --
h(B)
h(C)++

② dvojitá rotace



1. Dej Y do kořene
2. Jednoznačně přiřaď A, B, C, D vrcholům X a Y

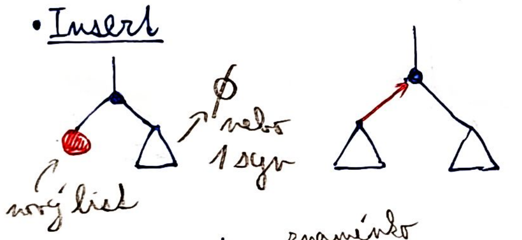
→ dá se složit z rotace hrany 1 a pak 2

⇒ tak se to programuje

• Vyvažování při operacích

Def: Znaménko vrcholu v je $\delta(v) := h(r(v)) - h(l(v))$. $\Rightarrow \delta(v) \in \{+1, 0, -1\}$

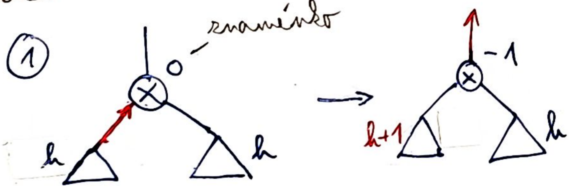
• Insert



nahoru posílám signál o prohloubení – BÚNO zleva

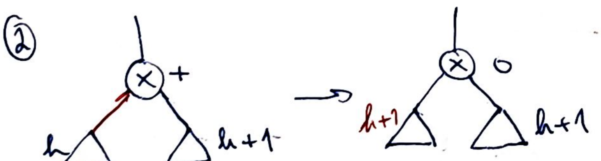
⇒ podle znaménka a signálu vyvažuji

①



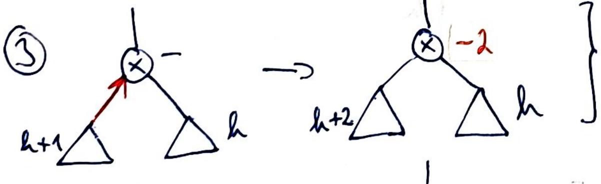
h+1 ↓ h+2 ⇒ prohloubení ⇒ další signál

②



h+2 ↓ h+2 ⇒ neprohloubilo se to ⇒ STOP

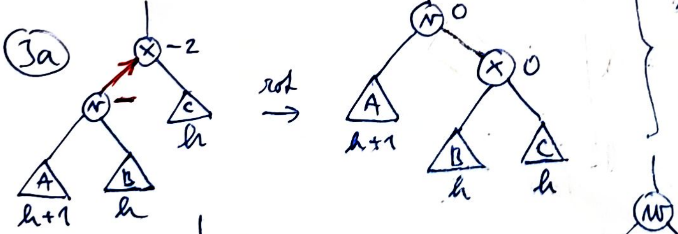
③



h+2 ↓ ?

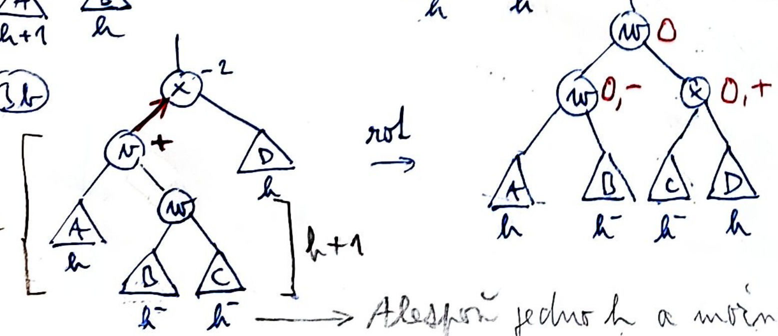
$\delta(x) = -2$ ⇒ rotace podle spodních pater

③a



h+2 ↓ h+2 ⇒ signál neposílám STOP

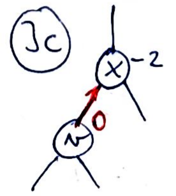
③b



h+1

Alespoň jedno h a možná vedle h−1

③c



nenastane

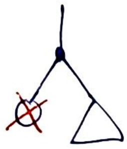
→ nikdy neposílám signál z vrcholu s $\delta(v) = 0$

⇒ indukcí nenastane

* ③b + posílám v ① zprava

- **Delete**

x je list $0 \to -1$



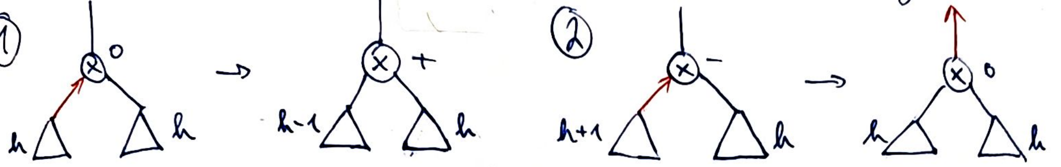
x má 1 syna $2 \to 1$

x má 2 syny → lze převést na jeden z předchozích případů

⇒ vždy se sníží hloubka nějakého podstromu

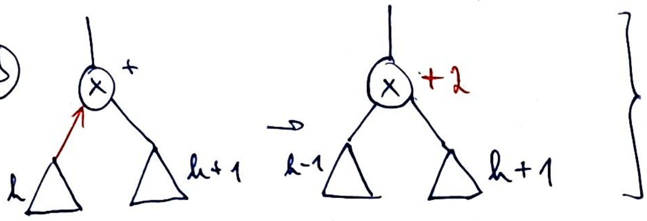
⇒ nahoru posílám signál o <u>snížení</u> hloubky – BÚNO zleva

① ②



$h+2 \downarrow h+1$ ⇒ signál dál

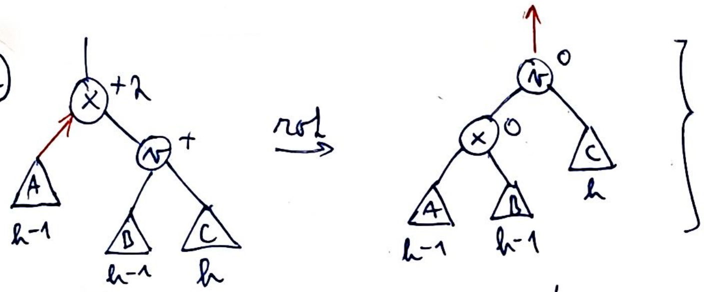
③



$h+2 \downarrow ?$

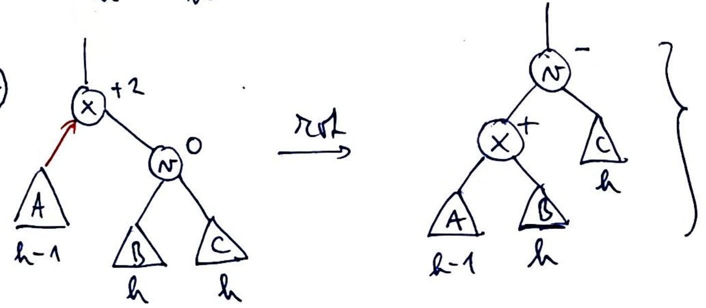
$\sigma(x) = +2$ ⇒ rozbor podpřípadů

3a



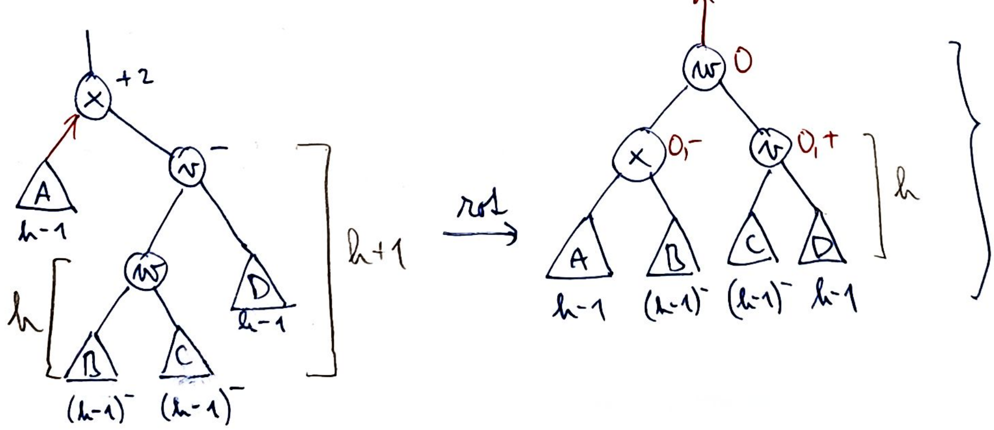
$h+2 \downarrow h+1$ ⇒ signál dál

3b



$h+2 \downarrow h+2$ ⇒ STOP

3c

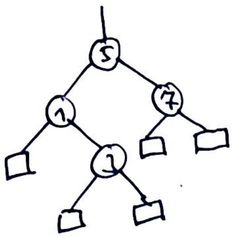


$h+2 \downarrow h+1$ ⇒ signál dál

⇒ na Insert dělám nejvýše 1 rotaci, na Delete můžu i víc, ale v obou případech strávím na všech hladinách konstantní dobu

<u>Věta:</u> Find, Insert, Delete v AVL stromu mají časovou složitost <u>$\Theta(\log n)$</u>.

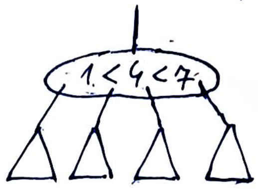
- (a,b)-stromy
  - Externí vrcholy



→ když vrcholu chybí syn, tak ho vyrobím

⇒ vlastní null-pointery jako v programu

- Vícecestní vyhledávací stromy



k klíčů ⇒ $k+1$ podstromů

⇒ klíče slouží jako oddělovače pro klíče podstromů

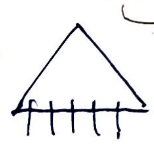
→ operace podobné jako na BVS, jen mám konstantně klíčů → musím najít podstrom

Def: (a,b)-strom pro $a \geq 2$, $b \geq 2a-1$ je vícecestný VS s externími vrcholy t.ž.

① # synů každého interního vrcholu $\in [a,b]$ ⇒ # klíčů $\in [a-1, b-1]$

\# synů kořene $\in [2,b]$ ⇒ # klíčů $\in [1, b-1]$ → proto výjimka

② všechny externí vrcholy leží na stejné hladině. ⇒



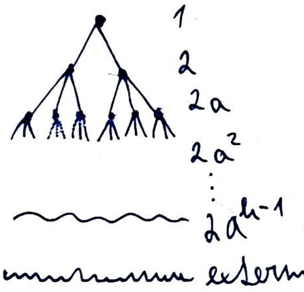
} interní

→ externí

Lemma: Hloubka (a,b)-stromu je $\Theta(\log n)$. → závisí na a,b

Dk: Stejný princip jako u AVL stromů → ukážu, že # klíčů roste exp. s hloubkou

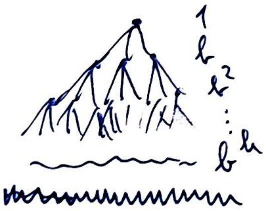
$A_h :=$ min # klíčů ve stromu hloubky h



$$n > A_h \geq 2a^{h-1} \in \Omega(a^h) \Rightarrow \log(n) > h \Rightarrow h \in O(\log n)$$

každý vrchol obsahuje alespoň a-1 klíčů ⇒ stačí 1

$B_h :=$ max # klíčů ve stromu hloubky h



$$n < B_h = \sum_{i=0}^{h} b^i < b^{h+1} \Rightarrow \log(n) < h+1 \Rightarrow h \in \Omega(\log n)$$

■

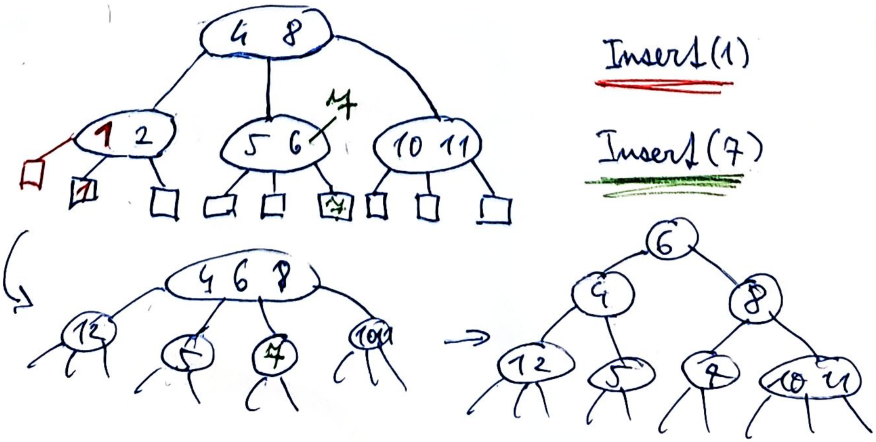
- Find

O(1) času na hladinu - konstantně klíčů ⇒ celkem $\Theta(\log n)$

Příklad Insertu

(2,3)-strom

\# klíčů 1 nebo 2

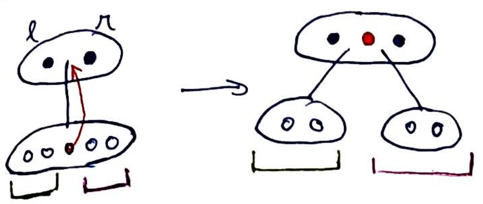


Insert(1)

Insert(7)

když klíče přetečou, tak prostřední vrchol vytáhnu nahoru

• **Insert**



→ v otci to zase mohlo přetéct
⇒ pokračuju nahoru ⇒ můžu rozdělit i kořen

↳ když klíče přetečou ⇒ b klíčů ⇒ 1 nahoru

⇒ může vzniknout vrchol s méně než $a-1$ klíči?   ↳ $\lfloor \frac{b-1}{2} \rfloor$ a $\lceil \frac{b-1}{2} \rceil$ $* \lfloor \frac{5}{2} \rfloor < 3$ $* \frac{5}{2} < 3$

⇒ kdyby se pokazilo, tak $\lfloor \frac{b-1}{2} \rfloor < a-1 \overset{*}{\Rightarrow} \frac{b-1}{2} < a-1 \Rightarrow b < 2a-1$ ↯

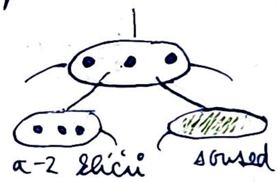
⇒ proto $b \geq 2a-1$

→ na každé hladině trávíme konstantní času ⇒ $\Theta(\log n)$

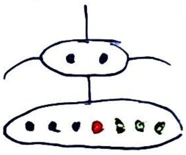
• **Delete**

① x není na poslední int. hladině
→ najdu minimum z $R(x)$
⇒ nahradím ⇒ převedu na ②

② x je na poslední hladině $a \geq 2$ ⇑
→ pokud vrchol nepodteče ✓
⇒ pokud podteče, tak využijeme souseda

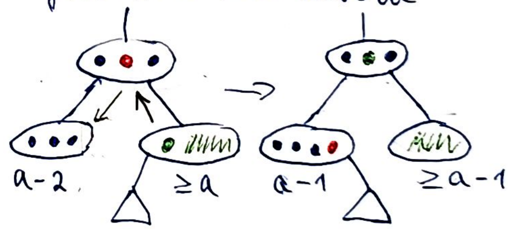


2a soused má a-1 klíčů
⇒ udělám merge + vezmu otci 1 vrchol
↳ protože bych měl 2 pointry vedle sebe



$a-2+1+a-1 = 2a-2 \leq b$

2b soused má více než a-1 klíčů
→ jeden klíč mu seberu



→ když podteče otec, tak zase využiju jeho souseda ⇒ znovu 2a nebo 2b
⇒ takhle se můžu dostat až do kořene
⇒ pokud kořenu seberu poslední klíč, tak se mi strom snížil
⇒ proto # synů kořene může být až 2

• **Červeno-černé stromy**
- další varianta vyhledávacího stromu
→ vlastně jde o kódování (2,4)-stromu do BVS

Hodí se $b = 2a$
Větší a, b:
→ 1 vrchol se má vejít do 1 bloku cache → seek je pomalý
→ 1 vrchol do 1 sektoru na disku
⇒ kořen v cache + velmi mělký

• **Řetězce**

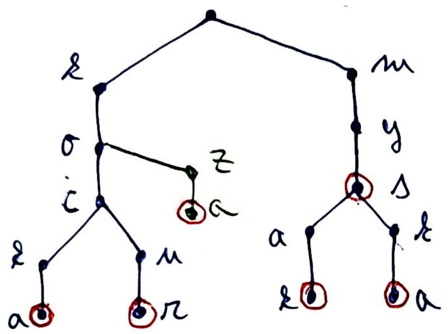
Řetězce nad abecedou Σ — $\{0,1\}$ / $\{A,\dots Z\}$ / Unicode … ~1M znaků

$x_1, \dots x_m$ řetězce délky L

$\Rightarrow$ BVS: operace v čase $\Theta(L \cdot \log m)$ → porovnání dvou řetězců je $\Theta(L)$

• **Písmenkové stromy** = Prefixové stromy = trie

{kočka, kocur, mys, mysak, myska}   Insert(koza)



→ když dojdu z kořene do vrcholu, tak jsem prošel po nějakém prefixu

$\Rightarrow$ vrcholy trie ≈ prefixy slov ze slovníku

$\Rightarrow$ # vrcholů $\in O$(součet délek slov)

$\Rightarrow$ vrcholy obsahují značky "konec slova" → rozhodování při Findu

+ pole ukazatelů na syny indexované abecedou $\Rightarrow \forall$ vrchol má $|\Sigma|$ synů

• **Find** $\Theta(L)$ → musím projít L hladin

• **Insert** → udělám Find

1, celé slovo je v trii $\Rightarrow$ přidám značku
2, v trii je jen prefix → prostě přidám vrcholy a nakonec značku } $\Theta(L)$

• **Delete** → přes Find najdu značku

$\Rightarrow$ rekurzivně: pokud jsem ve vrcholu co nemá ani značku ani syny, tak ho smažu $\Rightarrow$ jdu do otce } $\Theta(L)$

• **Závislost na velikosti abecedy**

Find → $\Theta(L)$
Insert → $\Theta(|\Sigma| \cdot L)$ → zakládá pole $|\Sigma|$ null-pointrů
Delete → $\Theta(|\Sigma| \cdot L)$ → maže pole pointrů + kontroluje všechny syny

• **Trie s BVS ve vrcholech**

→ ve vrcholu budu mít slovník ve tvaru {znak abecedy: pointer na vrchol}
$\Rightarrow$ řeší problém velké abecedy   ↳ až $|\Sigma|$ prvků
$\Rightarrow$ všechny operace jsou $\Theta(L \cdot \log|\Sigma|)$

• **Číslicový strom** = Radix trie

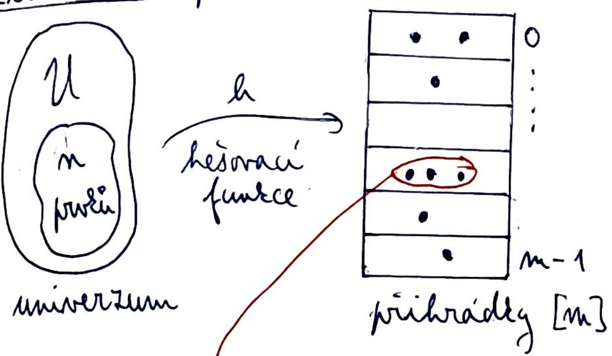
→ čísla ukládám jako řetězce do trie

• základ soustavy: Z

• číslo $\in [0, M]$ má nejvýše $\log_Z M + 1$ číslic

$\Rightarrow$ Find, Insert, Delete v čase $\Theta(\log_Z M)$ → asymptoticky si nepomůžem / → ale horší konstanty

• **Hešování s přihrádkami a řetězci**



chceme: h je rovnoměrná, ale deterministická

$\Rightarrow$ # prvků v přihrádce $\sim \frac{n}{m}$

$\Rightarrow$ pro $m \in \Theta(n)$ je to $O(1)$

$\Rightarrow$ Find, Ins, Del v čase $O(1)$?

kolize $\Rightarrow$ pro každou přihrádku si pamatuju seznam jejích prvků

$\Rightarrow$ Hešování s řetězci – tomu seznamu se říká řetězec

→ když $|U| = \infty$, tak $\exists$ přihrádka, do které se zobrazí ∞ prvků

$\Rightarrow$ umím udělat množinu, která se celá zobrazí do jediné přihrádky

Def: Systém $\mathcal{H}$ funkcí z U do $[m]$ je c-univerzální pro $c > 0 \equiv$

$$\forall x, y \in U, x \neq y: \Pr_{h \in \mathcal{H}}[h(x) = h(y)] \leq \frac{c}{m}.$$ ← náhodně volím $h \in \mathcal{H}$

→ kdybych měl úplně náhodnou fci, tak by to bylo $\frac{1}{m}$. (y má m možností)

Věta: Pro $x_1, \dots x_n, y \in U$ navzájem různé a $\mathcal{H}$ c-univerzální z U do $[m]$:

$$\mathbb{E}_{h \in \mathcal{H}}[\# i : h(x_i) = h(y)] \leq \frac{c \cdot n}{m}.$$ ← nevolím úplně náhodnou fci, ale náhodnou z toho mého univerzálního $\mathcal{H}$

Dk: Přes linearitu $\mathbb{E}$ a indikátory.

$$I_j := \begin{cases} 0 \\ 1 & \text{pokud } h(x_j) = h(y) \end{cases} \Rightarrow \mathbb{E}(I_j) = 0 + 1 \cdot \Pr[h(x_j) = h(y)] \leq \frac{c}{m}$$

$$\#\text{kolizí} = \sum_j I_j \Rightarrow \mathbb{E}[\#\text{kolizí}] = \mathbb{E}\left(\sum_j I_j\right) = \sum_j \mathbb{E}(I_j) \leq n \cdot \frac{c}{m} \quad \blacksquare$$

Princip: Nezajímá mě ošklivost vstupních dat, protože pro dobře zvolený systém hešovacích funkcí $\mathcal{H}$ a z něj náhodně vybranou funkci h bude amortizovaná složitost $O(c \frac{n}{m}) = O(c) = O(1)$

- **Konstrukce systému hešovacích funkcí**

- zvolíme nějaké konečné těleso $\mathbb{Z}_p$ ⇒ $m = p$ přihrádek $0, \dots p-1$

$\mathcal{U} = \mathbb{Z}_p^d$ → d-složkové vektory nad $\mathbb{Z}_p$ $(p \in \mathbb{P})$

⇒ hešovací fce bude $h_s(x) = s \cdot x$ – skalární součin s nějakým $s \in \mathbb{Z}_p^d$

Věta: Systém funkcí $\mathcal{H} = \{h_s \mid s \in \mathbb{Z}_p^d\}$ je 1-univerzální.

Dk: Chceme: $\forall x, y \in \mathbb{Z}_p^d : \Pr_{s \in \mathbb{Z}_p^d}[h_s(x) = h_s(y)] \le \frac{1}{m} = \frac{1}{p}$.

⇒ Máme dva různé vektory x, y, nechť k je souřadnice v níž $x_k \neq y_k$

⇒ skalární součin nezávisí na pořadí složek ⇒ BÚNO $k = d$.

$$\Pr_{s \in \mathbb{Z}_p^d}[x \cdot s \equiv y \cdot s] = \Pr[(x-y)s \equiv 0] = \Pr\left[\sum_{i=1}^{d}(x_i - y_i)s_i \equiv 0\right]$$

$$= \Pr\left[(x_d - y_d)s_d = \sum_{i=1}^{d-1}(x_i - y_i)s_i\right] = \Pr[a \cdot s_d = b]$$

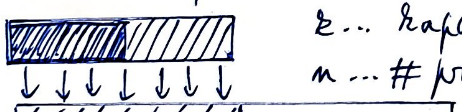
→ pokud už jsme náhodně zvolili $s_1, \dots s_{d-1}$ a teď náhodně volíme s_d, tak kolize nastane právě pro jednu volbu z $p = m$ možných ■

Příklad: Chci hešovat 32-bit čísla do cca 250 přihrádek.

$p = 257$. int32: [8b.|8b.|8b.|8b.] $2^8 = 256$ ⇒ int32 ~ $\mathbb{Z}_{257}^4$

$123\,456\,789 = 7 \cdot 2^{24} + 91 \cdot 2^{16} + 205 \cdot 2^8 + 21 \sim x = (7, 91, 205, 21)$

- **Nafukovací pole**



k… kapacita pole

n… # prvků

$k' = 2k$

když $n = k$, tak pole nafouknu na k'

Čas na vložení n prvků celkem?

před 2^l ← po 2^{l+1}

počáteční $k_0 = 1$
$k_1 = 2$
$k_3 = 4$
$\vdots$
$k_l = 2^l$
konečná $k_{l+1} = 2^{l+1}$

realokace: $1 + 2 + 2^2 + \dots + 2^l = 2^{l+1} - 1$

$$\underbrace{2^l < n \le 2^{l+1}}_{2^{l+1} < 2n} < 2n \Rightarrow 2^{l+1} \in \Theta(n)$$

⇒ n prvků … $\Theta(n)$ ⇒ amortizovaná pro 1 prvek je $O(1)$

- Zvětšování hešovací tabulky – stejný princip, jen to musím přehešovat

⇒ cena na zahešování je průměrně $O(1)$ ⇒ vložení n prvků je průměrně $\Theta(n)$

⇒ cena na vložení 1 prvku je pořád průměrně $O(1)$

- Rozděl a panuj

- Mergesort $(a_1, \dots a_n)$

1. Pokud $n \leq 1$ return vstup
2. Mergesort $(a_1, \dots a_{\lfloor n/2 \rfloor}) \to x_1, \dots x_{\lfloor n/2 \rfloor}$
3. Mergesort $(a_{\lceil n/2 \rceil}, \dots a_n) \to y_1, \dots y_{\lceil n/2 \rceil}$
4. return Merge (X, Y) $\leftarrow \Theta(n)$

Merge: $O(s+t)$

- analýza substitucí $T(n) :=$ čas pro vstup délky n $\leftarrow$ zatím předpokládám $n = 2^k$

$T(1) = 1$

$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + c \cdot n$

$T(n) = 2 \cdot \left(2 \cdot T\left(\frac{n}{4}\right) + c\left(\frac{n}{2}\right)\right) + c \cdot n = 4 \cdot T\left(\frac{n}{4}\right) + 2 \cdot c n$

$T(n) = 2^i \, T\left(\frac{n}{2^i}\right) + i \, c \cdot n$, chci $T\left(\frac{n}{2^i}\right) = 1 \Rightarrow i = \log_2 n$

$\Rightarrow T(n) = n \cdot T(1) + c \cdot n \log_2 n$, pro $i = \log_2 n$

$\Rightarrow T(n) = n + c \cdot n \log_2 n \in \Theta(n \log n)$

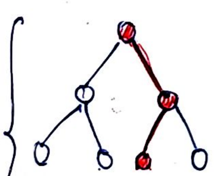
Paměť: slévám v $\Theta(n)$ + já a 1 z podproblémů si pamatujeme vstup

$S(n) = d \cdot n + S\left(\frac{n}{2}\right)$ $\to$ já + podproblém

$S(1) = 1$ vejde se tam i slévání

$\Rightarrow S(n) = dn + d \cdot \frac{n}{2} + S\left(\frac{n}{4}\right) = dn + d\frac{n}{2} + d\frac{n}{4} + \dots \leq 2dn \in \Theta(n)$

- analýza stromem rekurze



$\log_2 n$ hladin

vel. pp.	# p.p.		čas na 1 pp (merge)		čas na hladinu
n	1	$\cdot$	n	$=$	n
$n/2$	2	$\cdot$	$n/2$	$=$	n
$n/4$	4	$\cdot$	$n/4$	$=$	n
$\vdots$	$\vdots$		$\vdots$		$\vdots$
$n/2^i$	2^i	$\cdot$	$n/2^i$	$=$	n
listy: 1	n	$\cdot$	1	$=$	n

čas na 1 hladinu: n

hladin: $\log_2 n$

$\Rightarrow \Theta(n \log n)$

$\to$ paměť: když jsem v nějakém podproblému, tak si pamatuju všechny jeho předky

$\Rightarrow$ zásobník: $n + \frac{n}{2} + \frac{n}{4} + \dots \leq 2n \in \Theta(n)$

• **Násobení velkých čísel** – Karacubův algoritmus

$\overbrace{\quad}^{n=2^k}$

X	A	B
Y	C	D

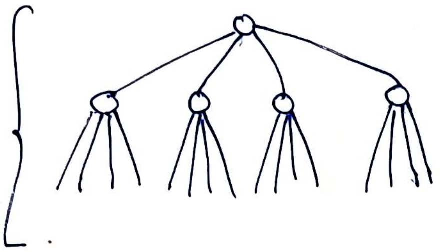
$$\left.\begin{aligned} X &= A\cdot 10^{n/2} + B \\ Y &= C\cdot 10^{n/2} + D \end{aligned}\right\} X\cdot Y = AC\cdot 10^{n} + AD\cdot 10^{n/2} + BC\cdot 10^{n/2} + BD$$

$\ll_{10} n$

↳ $\oplus$ a $\cdot 10^n$ je $\Theta(n)$

$$\Rightarrow T(n) = 4\cdot T\left(\frac{n}{2}\right) + c\cdot n$$

$\log_2 n$



vel. pp.	# pp.	čas na hladinu
n	1	
$n/2$	4	
$\vdots$	$\vdots$	
$n/2^i$	4^i	$n\cdot 2^i$
1	$4^{\log_2 n} = n^2$	

normální násobení je n^2 $\Rightarrow$ tohle je špatné

👁 stačí nám 3 násobení: $(A+B)(C+D) = AC + AD + BC + BD$

$$\begin{aligned} X\cdot Y &= AC\cdot 10^n + (AD+BC)\cdot 10^{n/2} + BD \\ &= AC\cdot 10^n + \left[(A+B)(C+D) - AC - BD\right]\cdot 10^{n/2} + BD \end{aligned}$$

$\Rightarrow$ čas na hladinu $\in \Theta\left(\frac{n}{2^i} 3^i\right) = \Theta\left(n \left(\frac{3}{2}\right)^i\right)$

👁 $\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1} \in \Theta(q^k)$

$$\Rightarrow T(n) \overset{\Theta}{=} \sum_{i=1}^{\log_2 n} n \left(\frac{3}{2}\right)^i \overset{\Theta}{=} n\cdot \left(\frac{3}{2}\right)^{\log_2 n} = 3^{\log_2 n} = 2^{\log_2 3\cdot \log_2 n} = n^{\log_2 3}$$

Věta: Karacubův algoritmus běží v čase $\Theta(n^{\log_2 3}) = \Theta(n^{1.58})$

• **Obecně**

$T(n) = a\cdot T\left(\frac{n}{b}\right) + \Theta(n^c)$ předpokládáme $n = b^k$

$$\left.\begin{aligned} \#\text{ hladin} &= \log_b n \\ \text{vel. pp.} &= n/b^i \\ \#\text{ pp.} &= a^i \end{aligned}\right\} \begin{aligned} \text{čas na pp.} &= \left(\frac{n}{b^i}\right)^c \\ \text{čas na hladinu} &= \left(\frac{n}{b^i}\right)^c a^i = n^c \left(\frac{a}{b^c}\right)^i \end{aligned}$$

$$\Rightarrow T(n) = \sum_{i=1}^{\log_b n} n^c \left(\frac{a}{b^c}\right)^i = n^c \sum_{i=1}^{\log_b n} q^i, \quad q = \frac{a}{b^c}$$

při posunu o hladinu dolů se práce změní q-krát

• $q<1$: $\sum \in O(1) \Rightarrow \Theta(n^c)$ = práce v kořeni

• $q=1$: $\sum = \log_b n \Rightarrow \Theta(n^c\cdot \log n)$ = práce na hl. · # hladin

$\log_b a > c$

listů

• $q>1$: $\sum \in \Theta\left(\left(\frac{a}{b^c}\right)^{\log_b n}\right) \overset{\Theta}{=} \frac{a^{\log_b n}}{n^c} = \frac{1}{n^c} b^{\log_b a \log_b n} = \frac{1}{n^c} n^{\log_b a} \Rightarrow \Theta(n^{\log_b a})$

→ co když $n \neq b^k$?

→ obecné n: $n^- \leq n \leq n^+$ ← nejbližší mocniny b

$T(n^-) \leq T(n) \leq T(n^+)$

⇒ indukcí od listů do kořene – pro listy $T(n^-) = T(n^+) = T(1) = 1$ ✓

$T(n) = a \cdot T\left(\frac{n}{b}\right) + \Theta(n^c)$ ← $T(n)$ je monotónní

⇒ pokud platí pro podproblémy, tak i pro rodičovský problém ∎

→ $\frac{n^+}{n^-} = b$ ⇒ $T(n^-) \in \Theta(T(n^+))$ ⇒ $T(n) \in \Theta(T(n^+))$

Věta (kuchařková): Rekurentní rovnice $T(n) = a \cdot T\left(\frac{n}{b}\right) + \Theta(n^c)$, $T(1) = 1$

pro $a \in \mathbb{N}$, $a \geq 1$ = počet podproblémů

$b \in \mathbb{R}^+$, $b > 1$ = kolikrát jsou podproblémy menší

$c \in \mathbb{R}^+_0$, $c \geq 0$ → n^c = práce v podproblému velikosti n

má řešení:

kořen (1) $\Theta(n^c)$ … $q < 1$

listy (2) $\Theta(n^{\log_b a})$ … $q > 1$

stejně (3) $\Theta(n^c \log n)$ … $q = 1$

pro $q = \frac{a}{b^c}$

$n^c \to a \cdot \left(\frac{n}{b}\right)^c = n^c \cdot \frac{a}{b^c}$

Příklady:

	a	b	c		
mergesort:	2	2	1	… $q = \frac{2}{2} = 1$	⇒ $n \cdot \log n$
násobení:	4	2	1	… $q = \frac{4}{2} = 2$	⇒ n^2
	3	2	1	… $q = \frac{3}{2}$	⇒ $n^{\log_2 3}$
bin-search:	1	2	0	… $q = \frac{1}{1} = 1$	⇒ $1 \cdot \log n$

- Násobení matic – Strassenův alg.

$n \times n$ → $n = 2^k$, jinak tam dáme padding z nul

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} P & Q \\ R & S \end{pmatrix} = \begin{pmatrix} AP + BR & AQ + BS \\ CP + DR & CQ + DS \end{pmatrix}$$

(bloky velikosti $n/2$)

→ 8 násobení $\left(\frac{n}{2}\right) \times \left(\frac{n}{2}\right)$
+ sčítání, sečtení, … $\in \Theta(n^2)$

	a	b	c		
násobení matic:	8	2	2	… $q = \frac{8}{4} = 2$	⇒ n^3 ~ jako z definice
strassen:	7	2	2	… $q = \frac{7}{4}$	⇒ $n^{\log_2 7} \approx n^{2.807}$

↳ Strassenovy vzorce → spočítá se 7 součinů ⇒ $\Theta(n^{\log_2 7})$

→ tohle je super, protože spoustu problémů lze převést na násobení matic

• **Quickselect**

→ alg. na k-tý nejmenší prvek ⇒ medián ~ $\lfloor \frac{n}{2} \rfloor$

Def: m je medián ≡ nejvýš $\lfloor \frac{n}{2} \rfloor$ prvků je menších & nejvýš $\lfloor \frac{n}{2} \rfloor$ je větších

$x_1 \ldots x_n$: | $L < p$ | $S = p$ | $P > p$ |

$p = x_i$ ... pivot

pokud $k \leq |L|$ → hledám v L

$|L| < k \leq |L| + |S|$ → pivot je výsledek

$|L| + |S| < k$ → hledám v P

Quickselect(X, k):

1. vybereme pivot $p \in X$
2. rozdělíme X na L, S, P podle p
3. Pokud $k \leq |L|$: return Quickselect(L, k)
   $|L| < k \leq |L| + |S|$: return p
   $|L| + |S| < k$: return Quickselect$(P, k - |L| - |S|)$

(1) Pivot = medián: $|L|, |P| \leq \frac{n}{2} \Rightarrow T(n) \in \Theta(n + \frac{n}{2} + \frac{n}{4} + \ldots) \in \Theta(n)$

(2) Pivot = maximum: $T(n) \in \Theta(n + n-1 + n-2 + \ldots) \in \Theta(n^2)$

(3) Pivot = skoromedián $\quad T(n) \in \Theta(n + \frac{3}{4}n + (\frac{3}{4})^2 n + \ldots) \in \Theta(n)$

| 1 | 2 | 3 | 4 |

mimo P, skorom., mimo L ⇒ vždy zahodím alespoň $n/4$ prvků

• **Randomizovaný RAM** – má instrukci pro random čísla

$x \leftarrow$ random(y) ... $x \in [y]$ rovnoměrně náhodné

→ různé výpočty ~ jevy

⇒ $T(n)$ je náhodná veličina ⇒ $\mathbb{E}[T(n)] = ?$ $\quad P[T(n) > 2^n] = ?$

• **Hledání skoromediánu**

→ opakovaně vybíráme $p \in X$, dokud to není skoromedián → kontrola $\Theta(n)$

Lemma (O džbánu): Nechť pokus uspěje s pstí $p > 0$, pak

$$\mathbb{E}[\# \text{opakování do 1. úspěchu}] = \frac{1}{p}$$

→ $P[p$ je skoromedián$] \geq \frac{1}{2}$ → $\mathbb{E}[\#$ pokusů$] \leq 2$

⇒ průměrně $2 \cdot \Theta(n) \in \Theta(n)$ $\Big\}$ $\mathbb{E}[T(n)] \in \Theta(n)$

(3) Pokud skoromedián hledám takhle ⇒ $\mathbb{E}[T(\text{Quickselect})] \in \Theta(n)$

(4) Pivot = náhodný prvek → prvky co nejsou skorom. nezahazuje na rozdíl od (3)

⇒ určitě není horší než (3) ⇒ $\mathbb{E}[T(n)] \in \Theta(n)$

④ analýza pomocí epoch – epocha končí, když p = skoromedián

$n \quad \frac{3}{4}n \quad \left(\frac{3}{4}\right)^i n$

$\left.\begin{array}{l} \mathbb{E}[\#\text{ průchodů v epoše}] \leq 2 \\ O(n) \text{ na 1 průchod} \end{array}\right\} \mathbb{E}[T(\text{epocha})] \in \Theta(n)$

$\Rightarrow T(n) = \sum_i T(\text{epoch}) = \sum_i \left(\Theta\left(\left(\frac{3}{4}\right)^i \cdot n\right)\right) \in \Theta(n) \leftarrow$ v průměru

• **Quicksort(X)**

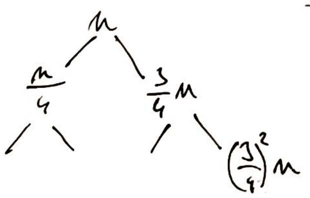
0. Pokud $|X| \leq 1$: return X
1. vybereme pivot $p \in X$
2. rozdělíme X na L, S, P podle p
3. $L' \leftarrow$ Quicksort(L)
   $P' \leftarrow$ Quicksort(P)
4. return $\boxed{L' \,|\, S \,|\, P'}$

↗ stejné jako mergesort

① p je medián $\Rightarrow \Theta(n \log n)$

② p je maximum $\Rightarrow \Theta(n^2)$

③ p je skoromedián $\Rightarrow \Theta(n \log n)$



$n = \left(\frac{4}{3}\right)^h \Rightarrow h = \log_{\frac{4}{3}} n$

$T(n) = n \cdot h = n \log n$

④ p je náhodný prvek

$T_i := \#$ porovnání, kterých se účastní x_i (když nebyl pivotem) ~ čas

$T(n) = \sum_i T_i \Rightarrow \mathbb{E}[T(n)] = \sum_i \mathbb{E}[T_i]$

→ T_i: v 1 podproblému 1 porovnání (s pivotem)

→ epochy jako u Quickselectu $\Rightarrow \mathbb{E}[\#\text{pp v epoše}] \leq 2$

↳ epocha končí skoromediánem $\Rightarrow \#\text{epoch} \leq \log_{\frac{4}{3}} n \in \Theta(\log n)$

$\Rightarrow \mathbb{E}[T_i] \in 2 \cdot \Theta(\log n) \in \Theta(\log n) \Rightarrow \mathbb{E}[T(n)] \in \Theta(n \log n)$

• **Linearselect**

→ Quickselect přes náhodné pivoty byl v nejhorším případě $\Theta(n^2)$ → jde to lineárně

→ rozložím prvky na pětice → najdu mediány pětic → p = medián těchto mediánů

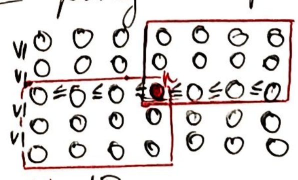
$O(n) \qquad O(n) \qquad T(\text{Linearselect})$

Linearselect(X, k)

0. Pokud $|X| \leq 5$: vyřeším triviálně
1. prvky rozdělíme na pětice $P_1, \dots P_{\lceil \frac{n}{5} \rceil}$
2. $m_i \leftarrow$ Linearselect$(P_i, 3)$
3. $p \leftarrow$ Linearselect$(\{m_1, \dots m_{\lceil \frac{n}{5} \rceil}\}, \lfloor \frac{n}{10} \rfloor)$
4. rozdělíme X na L, S, P podle p
5. Pokud $k \leq |L|$: return LS(L, k)
   $|L| < k \leq |L| + |S|$: return p
   $|L| + |S| < k$: return LS(P, k − |L| − |S|)

👁 vždy vypadne alespoň 3/10 prvků

Dk: prvky si uspořádám do pětic:



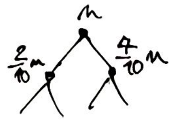
$\geq p \notin L$

← mediány pětic

$\leq p \notin P$

$\Rightarrow$ vypadne $\boxed{}$ $\frac{n}{10} \times 3$

$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + \Theta(n)$



$\rightarrow \frac{9}{10} n \Rightarrow$ exp. klesá $\Rightarrow$ kořen dominuje

$\Rightarrow T(n) \in \Theta(n)$

• **Dynamické programování**

• **Nejdelší rostoucí podposloupnost** – grafový pohled

$-\infty$ 3 14 15 92 65 35 89 71 32 38 36 18 43 $+\infty$

→ hrana vede z menšího čísla do většího

→ to určuje topologické uspořádání toho DAGu

→ chci nejdelší cestu z $-\infty$ do $+\infty$ ⇒ $O(n^2)$

• **Editační vzdálenost** = Levenshteinova vz.

$$\text{kopec} \begin{cases} \text{konec} \ldots \text{změna 1 znaku} \\ \text{skopec} \ldots \text{vložení 1 znaku} \\ \text{opec} \ldots \text{smazání 1 znaku} \end{cases} \Bigg\} \text{ editační operace}$$

$L(x_1 - x_n, y_1 - y_m) :=$ délka nejkratší posloupnosti edit. op., která $x_1 - x_n$ přetvoří na $y_1 - y_m$

≤ max(n, m) přepisování

L je metrika na množině všech řetězců

→ Odkud přišel 1. znak výsledného řetězce?

$$\left.\begin{array}{l} ① \ x_1 = y_1 : \ L(x_2 - x_n, y_2 - y_m) \\ ② \ \text{změna } x_1 \text{ na } y_1 : \ 1 + L(x_2 - x_n, y_2 - y_m) \\ ③ \ \text{vložení } y_1 : \ 1 + L(x_1 - x_n, y_2 - y_m) \\ ④ \ \text{smazání } x_1 : \ 1 + L(x_2 - x_n, y_1 - y_m) \end{array}\right\} \begin{array}{l} L = \min(①, ②, ③, ④) \\ L(\emptyset, \emptyset) = 0 \end{array}$$

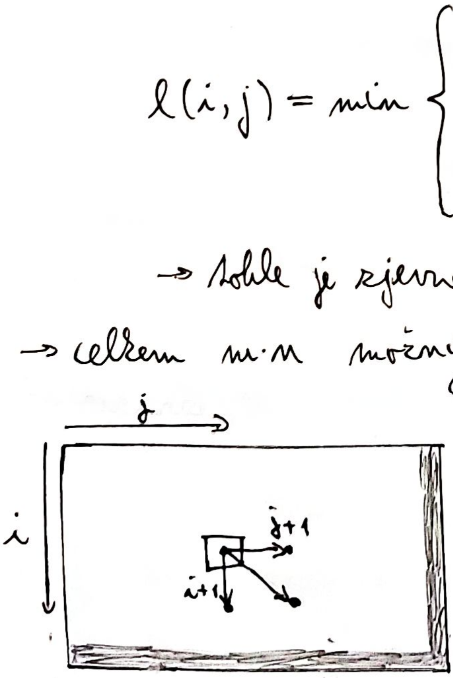
→ ty řetězce se nemění ⇒ stačí předávat indexy

$\ell(i,j) := L(x_i - x_n, y_j - y_m)$ kde $1 \le i \le n+1$, $1 \le j \le m+1$

$$\ell(i,j) = \min \left\{ \begin{array}{l} \ell(i+1, j+1) \ \ldots \ x_i = y_j \\ \ell(i+1, j+1) + 1 \\ \ell(i, j+1) + 1 \\ \ell(i+1, j) + 1 \end{array} \right\} \quad L = \ell(1,1)$$

→ tohle je zjevně exponenciální rekurze

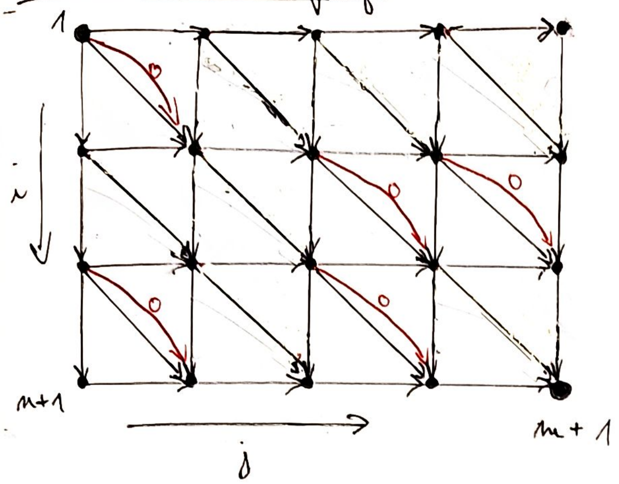
→ celkem $n \cdot m$ možných hodnot $\ell(i,j)$ ⇒ kešování ⇒ $O(n \cdot m)$



→ tabulku vyplňujeme zespoda po řádcích a v řádku vždy zprava doleva

⇒ nepotřebujeme rekurzi

- **Levenshteinův graf**



Editační operace ~ cesta po hraně

→ : vložení z y
↓ : smazání z x
↘ : změna x_i na y_j } cena 1
↘ : $x_i = x_j$ } cena 0

⇒ L = nejkratší cesta z (1,1) do (n+1, m+1)

⇒ je to DAG s $\Theta(n \cdot m)$ vrcholů i hran

→ cestu najdeme top. indukcí v $\Theta(n \cdot m)$

- **Floydův-Warshallův algoritmus**

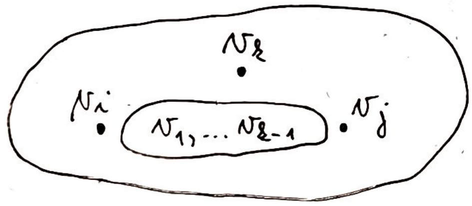
vstup: matice délek hran $L = \boxed{n \times n}$ $\quad L_{ij} = \begin{cases} \ell(v_i v_j) & v_i v_j \in E \\ +\infty & v_i v_j \notin E \end{cases}$

výstup: matice vzdáleností $D = \boxed{n \times n}$ $\quad D_{ij} = d(v_i, v_j)$ nebo $+\infty$

⇒ D^k_{ij} = délka nejkratší cesty z v_i do v_j přes $v_1, \dots v_k$
↳ vnitřní vrcholy cesty $\in \{v_1, \dots v_k\}$

$D^0 = L$
$D^n = D$

$D^{k-1} \to D^k$? $\quad \Theta(n^2)$

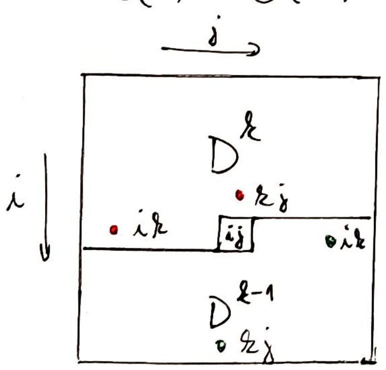


$$D^k_{ij} = \min \begin{cases} D^{k-1}_{ij} & \dots \text{ pokud } v_k \text{ nepoužijeme} \\ D^{k-1}_{ik} + D^{k-1}_{kj} & \dots \text{ jinak} \end{cases}$$

jen 2 možnosti ⇒ minimum

⇒ celkem $T(n) = \Theta(n^3)$

⇒ $S(n) = \Theta(n^2)$ – je možné to dělat na místě rovnou z L na D



→ kdy se to může pokazit?

1) D^{k-1}_{ij} ... to jsem ještě nepřepsal ✓

2) D^{k-1}_{ik}, D^{k-1}_{kj} ... to už může být v D^k, ale $D^{k-1}_{ik} = D^k_{ik}$, takže to nevadí

- **Princip dynamického programování**

→ máme nějaké podproblémy = stavy
→ a máme graf co říká, který podproblém potřebuje výsledky kterých podproblémů
⇒ tenhle graf je DAG ⇒ ty podproblémy můžeme řešit podle t. uspořádání