

Syllabus

Definice algoritmu

- 1 • Výpočetní model RAM ✓
- 2 • Čas a prostor konkrétního výpočtu ✓
- 3 • Časová a prostorová složitost ✓
- 4 • Asymptotická notace: O , Ω , Θ ✓

Základní grafové algoritmy

- 5 • Prohledávání do šířky (BFS) ~
- 6 • Prohledávání do hloubky (DFS) ✓
- 7 • Klasifikace hran v DFS ✓
- 8 • Hledání mostů ✓

Algoritmy pro orientované grafy

- 9 • Detekce cyklů pomocí DFS ~
- 10 • Acyklický orientovaný graf (DAG), zdroj, stok ✓
- 11 • Topologické uspořádání DAGu ✓
- 12 • Konstrukce topologického uspořádání ~
- 13 • Princip indukce podle topologického uspořádání ~
- 14 • Počet cest mezi dvěma vrcholy v DAGu ✓
- 15 • Silná souvislost, její komponenty, graf komponent ~
- 16 • Rozklad grafu na komponenty silné souvislosti ✓

Nejkratší cesty

- 17 • Vzdálenost v grafu ✓
- 18 • Trojúhelníková nerovnost pro vzdálenost ✓
- 19 • Dijkstrův algoritmus ✓
- 20 • Implementace Dijkstrova algoritmu pomocí haldy
- 21 • Obecný relaxační algoritmus
- 22 • Bellmanův-Fordův algoritmus
- 23 • Floydův-Warshallův algoritmus

Minimální kostry

- 24 • Jarníkův algoritmus ✓
- 25 • Lemma o řezech ✓
- 26 • Jednoznačnost minimální kostry ~
- 27 • Borůvkův algoritmus ✓
- 28 • Kruskalův algoritmus ✓
- 29 • Datová struktura Union-Find ✓

Vyhledávací stromy

- 30 • Rozhraní slovníku, množiny a jejich uspořádaných verzí ✓
- 31 • Binární vyhledávací strom (BVS) ✓
- 32 • Operace Find, Insert a Delete v BVS ✓

- 33 • Dokonale vyvážený strom ✓
- 34 • Dolní odhad složitosti pro dokonale vyvážené stromy ~
- 35 • AVL strom ✓
- 36 • Odhad hloubky AVL stromu ✓
- 37 • Rotace hrany stromu ✓
- 38 • Operace Insert a Delete v AVL stromech ✓

(a,b)-stromy

- 39 • Vícecestný vyhledávací strom a (a,b)-strom ✓
- 40 • Odhad hloubky (a,b)-stromu ✓
- 41 • Operace Insert a Delete v (a,b)-stromech ✓
- 42 • Volba parametrů (a,b)-stromu ✓

Hešování

- 43 • Hešování s řetězcí v přihrádkách ✓
- 44 • Operace Find, Insert a Delete v hešování s řetězcí ~
- 45 • Dynamické rozšiřování tabulky ✓
- 46 • c-univerzální systém funkcí ✓
- 47 • Konstrukce 1-univerzálního systému pomocí skalárního součinu ✓
- 49 • Průměrná složitost operací při náhodné volbě hešovací funkce z c-univerzálního systému ~

Rozděl a panuj

- 49 • Třídění sléváním (Mergesort) ✓
- 50 • Násobení n-ciferných čísel v čase $O(n \log 23)$ – Karacubův algoritmus ✓
- 51 • Kuchařková věta (Master theorem) ✓
- 52 • Strassenův algoritmus na násobení matic (vzorce nezkouším) ✓
- 53 • Třídění a vyhledávání ~ ?
Quickselect – hledání k-tého nejmenšího prvku ✓
- 54 • Průměrná časová složitost Quickselectu při náhodné volbě pivota ✓
- 55 • k-tý nejmenší prvek v lineárním čase (algoritmus s pěticemi) ✓
- 56 • Quicksort ~ – musím dopsat z poznámek
- 57 • Průměrná časová složitost Quicksortu při náhodné volbě pivota ✓

Dynamické programování

- 59 • Nejdelší rostoucí podposloupnost ✓
- 60 • Editační vzdálenost řetězců ✓
- 66 • Konstrukce optimálního binárního vyhledávacího stromu
- 61 • Princip dynamického programování ✓
- 62 • Grafová interpretace dynamického programování ✓

POPIS SLOŽITOST ALGORITMU

①

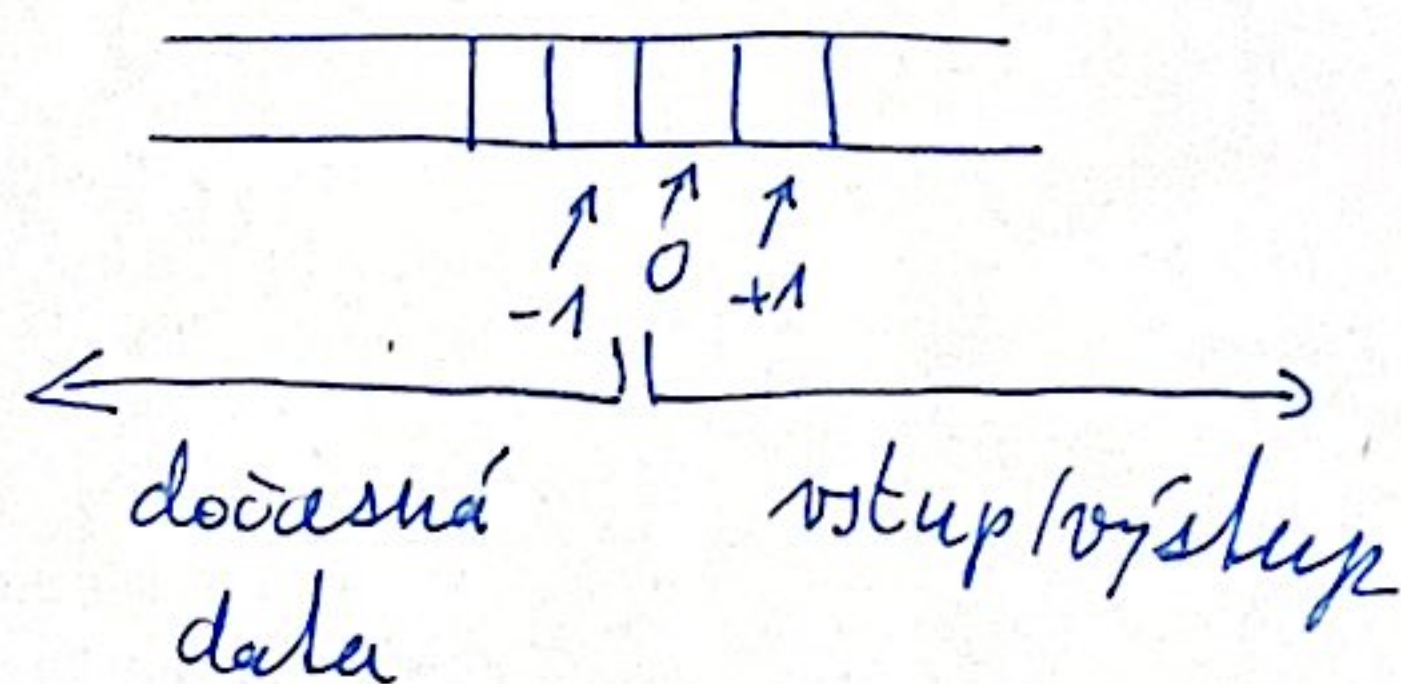
- algoritmus → je přesný a deterministický popis postupu

↳ stejný výsledek ↳ skončí

RAM (random access machine)

↳ počítač umožňující libovolný přístup v paměti

↳ paměť je pole celých čísel indexované celými čísly

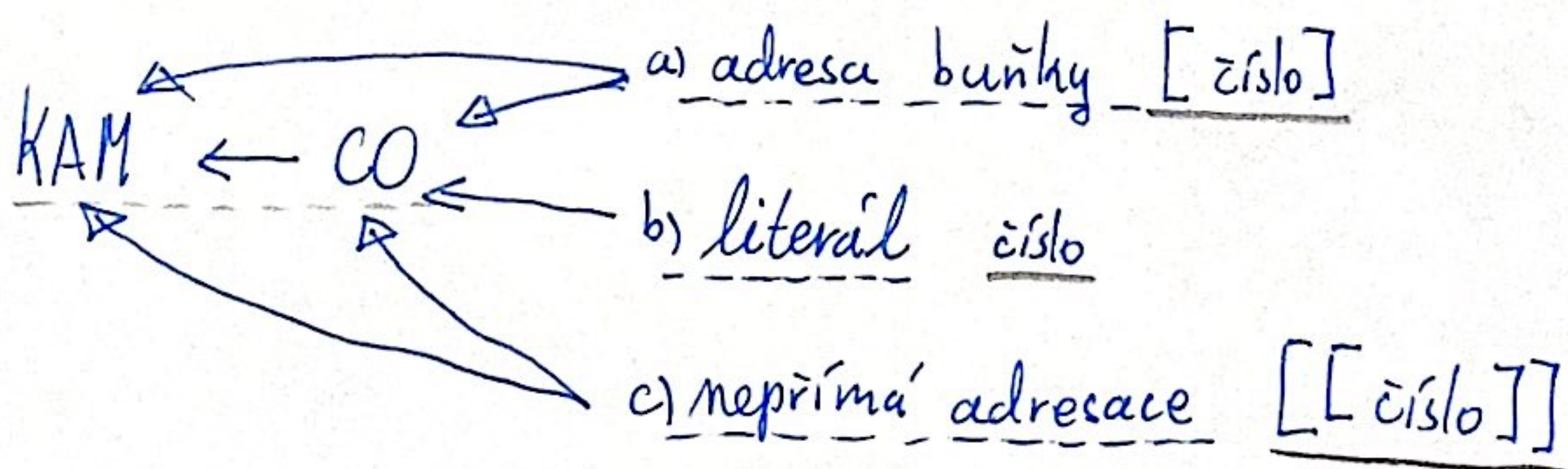


↳ program = posloupnost instrukcí

↳ Typy instrukcí

1) práce s pamětí

příklady: $[0] \leftarrow [1]$ $[3] \leftarrow 5$
 $[0] \leftarrow 0$ $[-2] \leftarrow [3]$



2) aritmetika

KAM $\leftarrow A \text{ op } B$

op = operace :

$+$, $-$, $*$, $/$, $\%$
(mod)

celočíslné

↳ pro kladná čísla,
jinak definované
pro záporné různé

$\&$ AND, $|$ OR, $\wedge$ XOR

$\ll$ SHL, $\gg$ SHR $\Rightarrow$ $x \ll k = x \cdot 2^k$
 $x \gg k = x / 2^k$

3) řízení toku / control flow

②

HALT → zastaví program

→ v každém programu je implicitní halt

GOTO kam → nepodmíněný skok

→ skáče podle návěští (signál / rozkaz) na číslo instrukce

IF A RELACE B GOTO kam → podmíněný skok

(=, ≠, >, <, ≥, ≤)

Program pro RAM

↳ 1. vstup → je uložený ve smluvených buňkách, nejčastěji v paměti od adresy 0

2. postupnost instrukcí

3. výstup → po HALTu je výstup také uložen ve smluvených buňkách

Úkklad programu pro RAM

• Bubble sort $O(n^2)$ časová a $O(1)$ prostorová

1. způsob - pseudokód:

VSTUP: N (celé číslo), P (pole velikosti N)

OPAKUJ

$Z \leftarrow NE$

PRO $I = 1, \dots, N-1$:

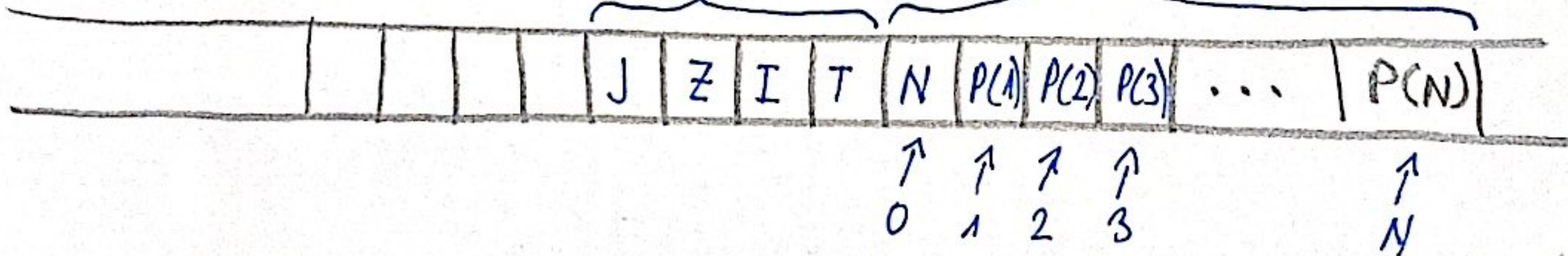
POKUD $P[I] > P[I+1]$:

$t \leftarrow P[I]; P[I] \leftarrow P[I+1], P[I+1] \leftarrow t$

$Z \leftarrow ANO$

DOKUD $Z = ANO$

2. způsob - instrukce počítače $J := [-4]i$ $T := [-1]$, $I := [-2]$, $Z := [-3]$, $N := [0]$
DOČASNĚ VSTUP $\rightarrow$ zkratkou



IF $N < 2$ GOTO KONEC

VNĚJŠÍ: $Z \leftarrow \emptyset$

$I \leftarrow 1$

OPAKUJ $J \leftarrow I+1$

IF $[I] \leq [J]$ GOTO OK

$T \leftarrow [I], [I] \leftarrow [J], [J] \leftarrow T, Z \leftarrow 1$

OK $I \leftarrow I+1$

IF $I < N$ GOTO OPAKUJ

IF $Z = 1$ GOTO VNĚJŠÍ

KONEC HALT

1. průchod
=
4 až 8 instrukcí
 $4n-4$
až
 $8n-8$

1 průchod # průchodů
 $4n-1$ 1
 $\vdots$ $\vdots$
 $8n-5$ n

$\Downarrow$
čas. složitost
 $4(n-1) + 2$
 $\dots$
 $8(n-1)(n-1) + 4(n-1) + 2$

$\nearrow$
nemí potřeba

CENA INSTRUKCE $\rightarrow$ k měření složitosti

(4)

1) jednotková

$\rightarrow$ všechny instrukce lze vykonat v konstantním čase nezávisle na velikosti (čísle) vstupu

$\rightarrow$ čas = $\sum$ instrukcí

2) omezená šířka slova

$\rightarrow$ omezení paměti |šířka| $< 2^w$

$\rightarrow$ pracuje se slovy maximálně w -bitů

$\rightarrow$ pro vstup velikosti n , tak musí být $w \geq \log(n)$

$\rightarrow := \log_2(n)$

$w = \log(\max(n, 2))$
přesněji

$|číslo| \leq n^c \quad \approx \quad w = c \cdot \log(n)$

3) logaritmická

$\rightarrow$ instrukce nemá konstantní cenu, ale závislou na velikosti operandů a adres

$\rightarrow$ cena = #bitů operandů + #bitů adresy

4) relativně logaritmická

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

$\rightarrow$ pro velká čísla $\rightarrow$ penalizuje
pro malá čísla $\rightarrow$ chová se jako konstantní

$\rightarrow$ nejčastěji se používá (4)

Def.: Doba běhu programu pro vstup x

(5)

• $t(x) :=$ součet cen provedených instrukcí

Def.: Časová složitost ("v nejhorším případě")

$$T(n) := \max \{ t(x) \mid x \text{ je vstup velikosti } n \}$$

Def.: Prostor běhu programu pro vstup x

$$\Delta(x) := \max. \text{ adresa} - \min. \text{ adresa} + 1$$

•  navštívené během běhu programu

Def.: Prostorová složitost

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

pracovní prostor → se vstupem se jenom čte a nepřepisuje se tam
části paměti.
→ do výstupu se jen zapisuje a nečte se odnikud

Def.: Asymptotická notace

$$f, g: \mathbb{N} \rightarrow \mathbb{R} : f \in O(g) : \exists c \in \mathbb{N} : \exists m_0 \in \mathbb{N} : \forall n \geq m_0 : 0 \leq f(n) \leq c \cdot g(n)$$

$$f \in O(g) = \exists c \forall^* n : f(n) \leq c \cdot g(n)$$

$$f \in \Omega(g) = \exists c \forall^* n : f(n) \geq c \cdot g(n)$$

$$f \in \Theta(g) = \exists c, c' : \forall^* n : c' g(n) \leq f(n) \leq c \cdot g(n) \rightarrow f \text{ roste stejně jako } g$$

$$f \in \Theta(g) \Leftrightarrow f \in O(g) \wedge f \in \Omega(g)$$

Grafové algoritmy

①

● BFS (do šířky) → nalezení nejkratší cesty

DFS (do hloubky) → Ariadnina nit

B.Ú.N.O. je graf G orientovaný



$n := |V(G)|$ počet vrcholů

$m := |E(G)|$ počet hran

● $uv :=$ hrana z u do v

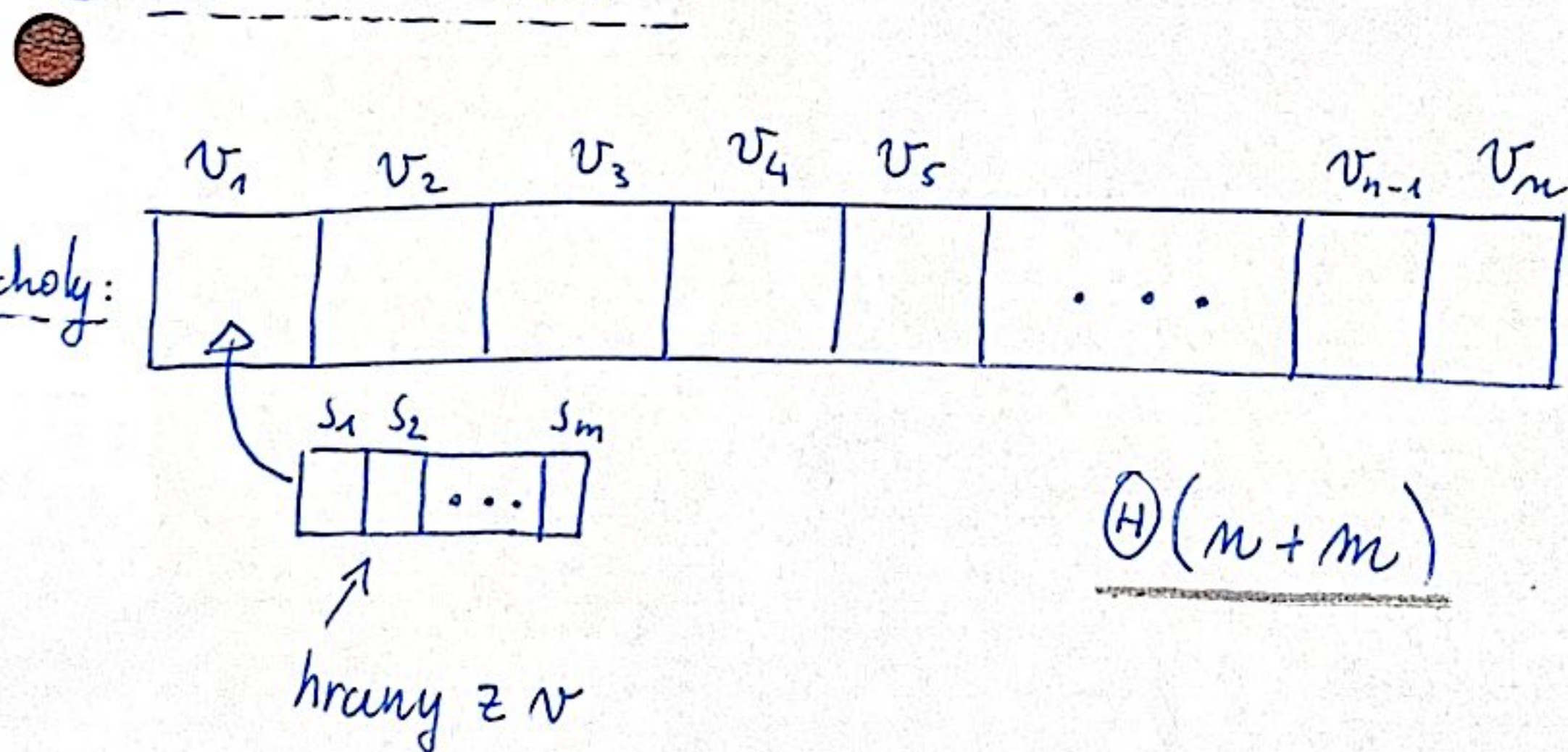
Reprezentace grafu

① matice sousednosti

$$(A)_{ij} = \begin{cases} 1 & \text{pokud } v_i v_j \in E(G) \\ 0 & \text{jinak} \end{cases}$$

= je velika $\Theta(n^2)$

② seznam sousedů



Prohledávání do hloubky (DFS)

②

stav vrcholu v : $stav(v) = \begin{cases} \text{neviděný} \\ \text{otevřený} \\ \text{uzavřený} \end{cases}$

inicializace : $\forall v \in V(G) : stav(v) \leftarrow \text{neviděný}$

rekurzivní DFS(v) : 1) $stav(v) \leftarrow \text{otevřený}$

] $\Theta(n)$ ← jednou pro každý vrchol

2) Pro $\forall vu \in E$:

3) Pokud $stav(u) = \text{neviděný}$:

4) DFS(u)

] $\Theta(\deg^{\text{OUT}}(v))$ ← stupeň ma počtu

5) $stav(v) \leftarrow \text{uzavřený}$

] $\Theta(1)$

① směny stavů

neviděný $\rightarrow$ otevřený $\rightarrow$ uzavřený

↳ po dobehnutí není žádný vrchol otevřený

② DFS neotevírá stejný vrchol vícekrát

↳ do vrcholu vstoupíme maximálně jednou

Lemma:

- Po dobehnutí DFS (v) je $\forall u \in V(G)$ stav(u) = $\begin{cases} \text{navštívený} \Leftrightarrow \exists \text{ cesta z } u \text{ do } v \\ \text{nevštívený} - \text{jinak} \end{cases}$

Dk.:

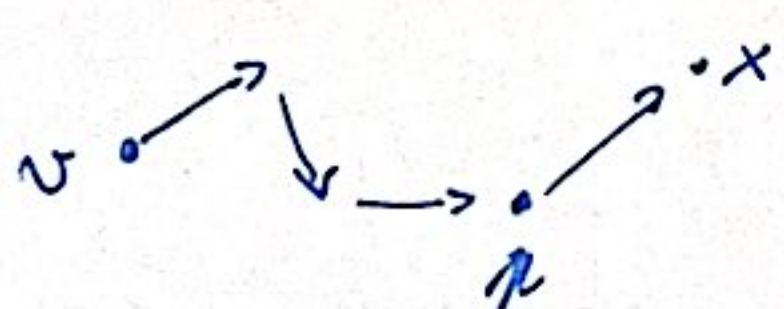
- 1) u je navštívený $\Rightarrow u$ je dosažitelný $\Leftrightarrow u$ je ovládnutý a v je navštívený $\Rightarrow u$ je dosažitelný

$\hookrightarrow$ indukci podle doby běhu programu

-  1) x ovládl $u \Rightarrow$ podle IP je x dosažitelný $\Rightarrow \exists x$ do u vede hrana $\Rightarrow \exists v$ do u vede sled / cesta

- 2) u je dosažitelný $\Rightarrow u$ je na konci navštívený

$\hookrightarrow$ nalezením minimálního protipříkladu

-  pro spor: $\exists x \in V(G) : x$ je dosažitelný $\Rightarrow x$ je nevštívený

$\Rightarrow$ zvolím x tak, aby cesta z v do x měla nejmenší hran

$\Rightarrow$ předposlední vrchol je p , který je vštívený, pak ovládnutý a ovládl vrchol x $\hookrightarrow$

Lemma:

DFS běží v čase $\Theta(n+m)$ $\hookrightarrow$ jen pro dosažitelnou část

Dk.: 2 způsoby

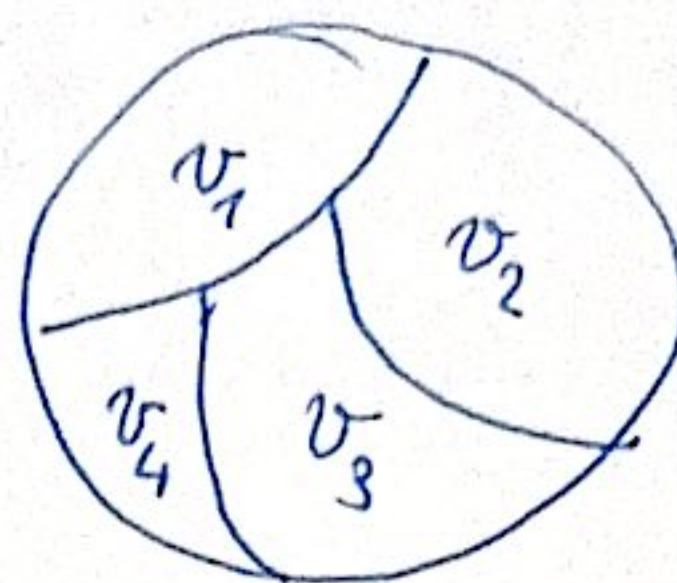
● 1) $\Theta(n) + \sum_v \Theta(\deg(v)) = \Theta(n+m)$

2) účtování času vrcholy a hranami $\left\{ \begin{array}{l} \text{vrcholy: } 1, 5 \rightarrow \Theta(n) \\ \text{hrany: } 2, 3, 4 \rightarrow \Theta(m) \end{array} \right\} \Theta(n+m)$

Opakované DFS

(4)

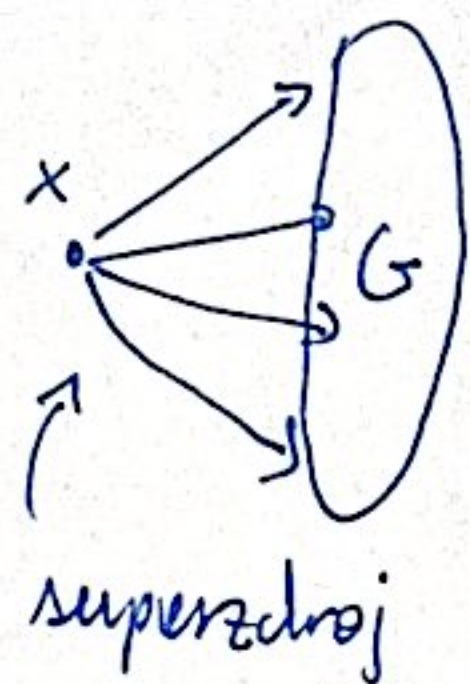
- 1 $\forall v \in V(G) : stav(v) \leftarrow nevíděn$
- 2 Pro $\forall v \in V(G)$:
- 3 Pokud $stav(v) = nevíděn$:
- 4 DFS(v)



↳ navštívení celého grafu $\rightarrow \Theta(n+m)$

1) pro V podgrafy to je $\Theta(n'+m')$ a
to se počítá na $\Theta(n+m)$

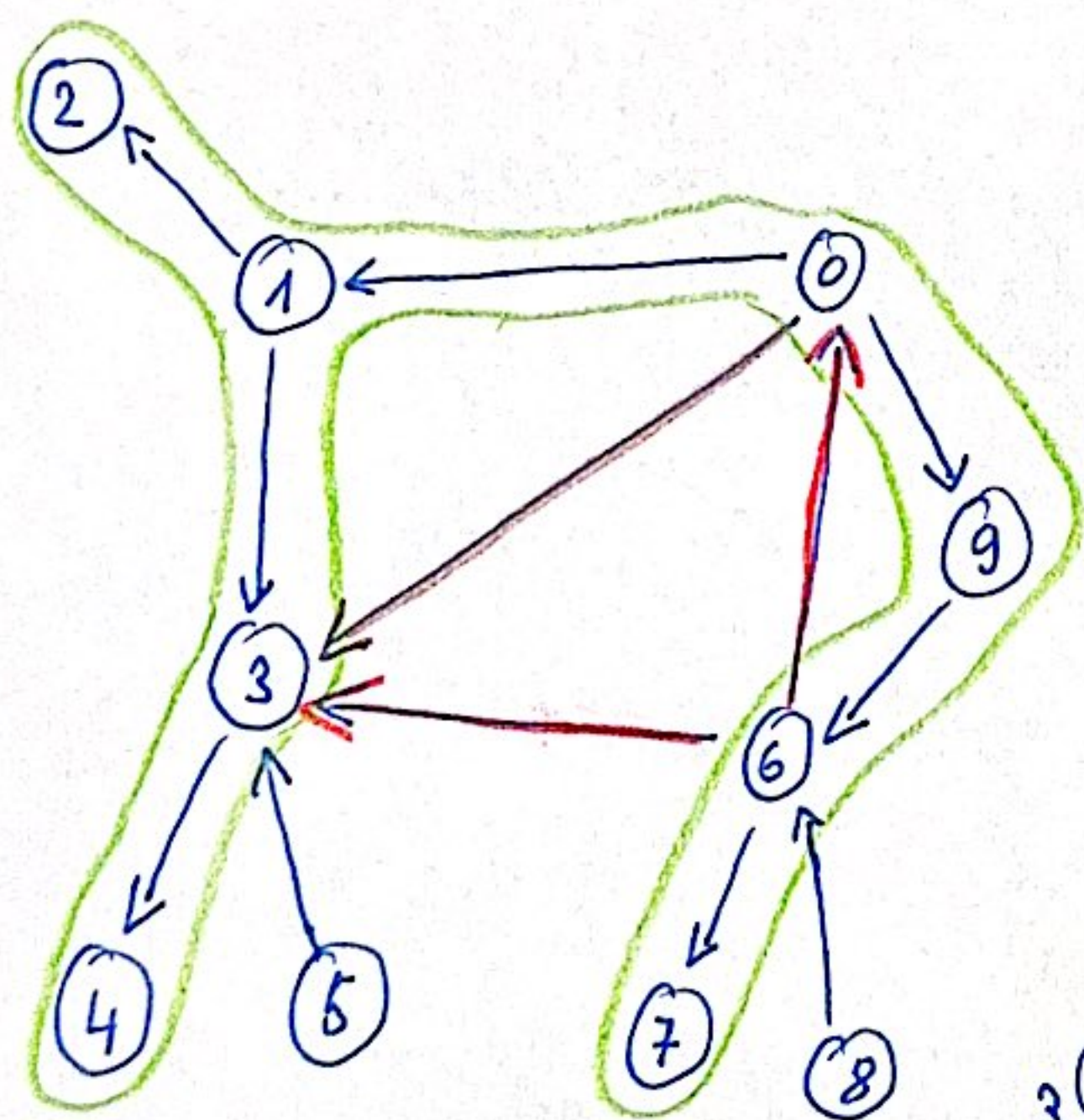
↳ alternativně:



DFS(x) $\Rightarrow \Theta(n+m)$ jde snadno odvodit

DFS strom grafu

↳ získáme kostru dosažitelné části grafu



DFS(0) $\rightarrow$ hrany jdoucí z vrcholu mají
prioritu procházení podle čísla cílového
vrcholu.

KLASIFIKACE HRAN DFS

1) STROMOVÉ (KOSTERNÍ)

↳

2) ZPĚTNÉ

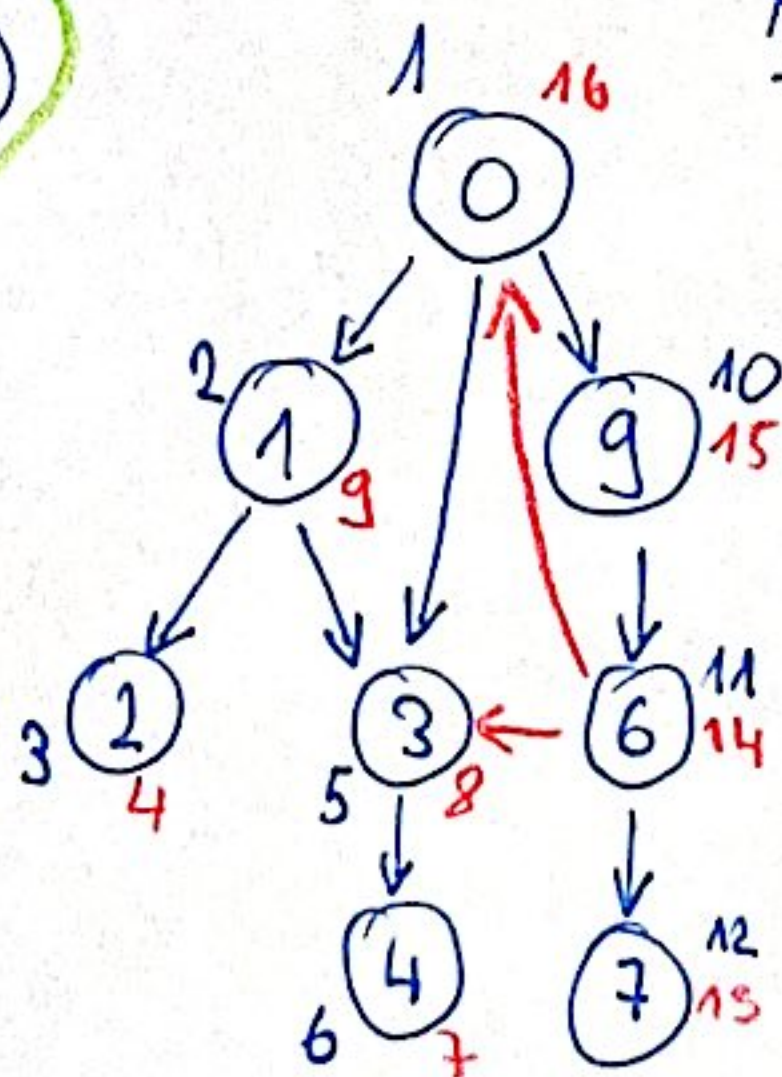
↳ hrany do předchůdce

3) DOPŘEDNÉ

↳ vedou do uzavřeného
vrcholu

4) PŘÍČNÉ

↳ vedou do navštíveného vrcholu,
nemají předek ani potomka



pořadí navštívení

inicializace

• $\forall v \in V(G) : \text{stav}(v) \leftarrow \text{neviděný} ; \text{in}(v), \text{out}(v) \leftarrow \emptyset$ (nedefinované)

$T \leftarrow 0$ čas (hodinky, co tikají při změně stavu)

DFS: 1 $\text{stav}(v) \leftarrow \text{otevřený}$

2 $T \leftarrow T+1, \text{in}(v) \leftarrow T$

3 Pro $vu \in E(G)$:

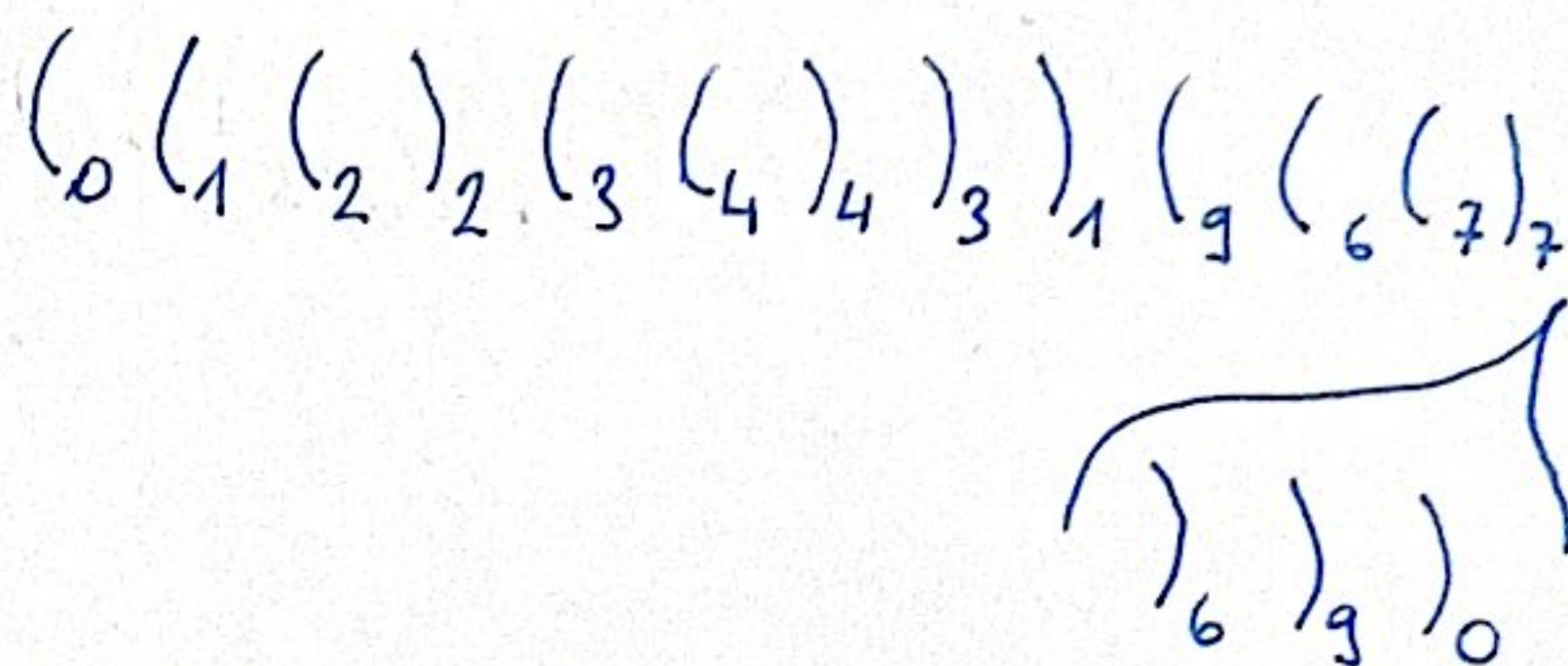
4 Pokud $\text{stav}(u)$ neviděný:

• 5 DFS(u)

6 $\text{stav}(v) \leftarrow \text{uzavřený}$

7 $T \leftarrow T+1, \text{out}(v) \leftarrow T$

otevírání a uzavírání =
začínání a končení



$\text{in}(v) = (v$

$\text{out}(v) =)v$

Klasifikace hran pomocí rávorů

↳ pro hranu $xy \in E(G)$

• 1) $(x)_x (y)_y \rightarrow \text{nenastane}$

2) $(y)_y (x)_x \rightarrow \text{příčná}$ $\text{stav}(y) = \text{uzavřený}$

3) $(x - (y)_y -)_x \rightarrow \text{dopředná}$ a) stromová $\rightarrow \text{stav}(y) = \text{neviděn}$
b) dopředná $\rightarrow \text{stav}(y) = \text{uzavřený}$

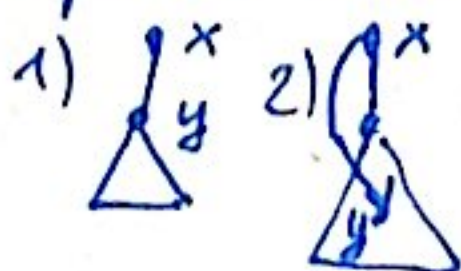
4) $(y (x)_x)_y \rightarrow \text{zpětná}$ $\text{stav}(y) = \text{otevřený}$ (při objevení)

↳ můžeme o jakékoli hraně zjistit v $O(1)$, kterého druhu je sáč

v neorientovaném grafu $\rightarrow$ podle toho, kdy byla hrana spárována

$xy \in E(G)$ poprvé
1) stromová
2) zpětná

podruhé
1) zpětná
2) dopředná



Věta: DFS běží v čase $\Theta(n+m)$ a prostoru $\Theta(n+m)$ a

najde dosažitelné vrcholy a klasifikaci hran.

↳ pomocí hledání mostů (v neorientovaných grafech)

Def.: Hrana e je most pokud $\#$ komponent v $G \setminus e$ stoupne

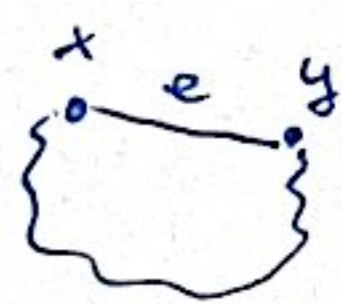
↳ e není most $\Leftrightarrow e$ leží na kružnici

Dk.: e leží na kružnici $\Rightarrow e$ není most



$e \in E(G)$

e není most $\Rightarrow e$ leží na kružnici

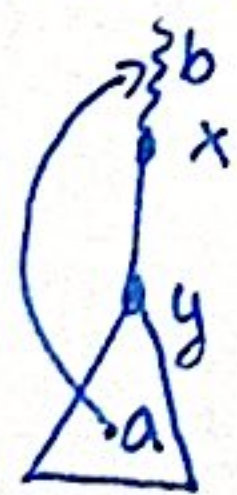


v $G \setminus e$ existuje cesta mezi x a $y \Rightarrow$ cesta + $e =$ kružnice

↳ zpětná hrana nemůže být most

Všechny mosty jsou stromové hrany

Stromová hrana xy není most (" xy leží na kružnici") $\Leftrightarrow \exists$ zpětná hrana ab ,
kde a leží v $T(y)$ a b leží outside $y \Leftrightarrow \underline{\text{low}}(y) < \text{in}(y)$

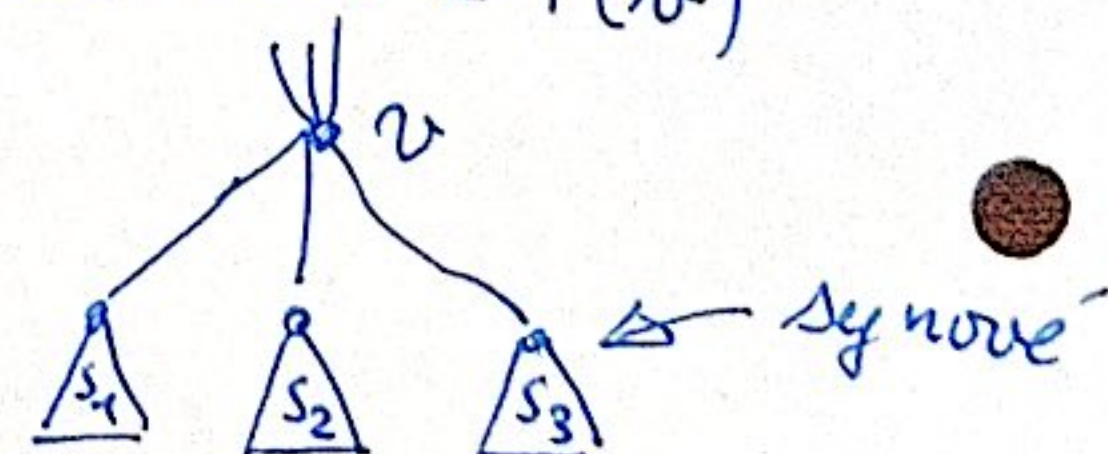


↳ strom pod y

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

↳ vlastně jak moc vysoko jsem subtree dosáhl z $T(v)$

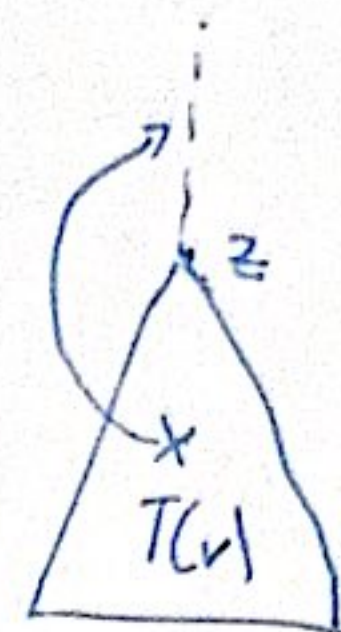
↳ spočítat $\text{low}(v)$ pro $\forall v \in V$ v lineárním čase



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

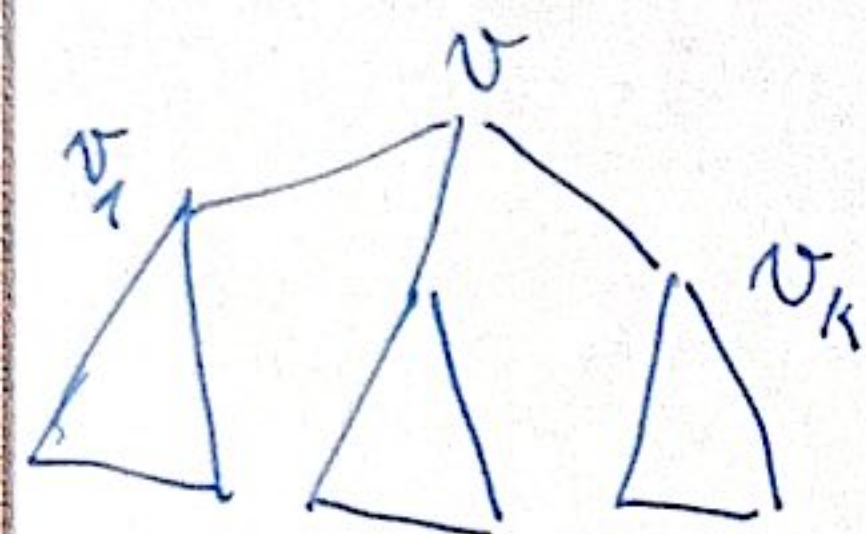
Opakování z minule

$$\text{low}(v) = \min \left\{ \text{in}(z) \mid \begin{array}{l} xz \in E \text{ zpětná} \\ x \in T(v) \end{array} \right\}$$



$$\text{low}(v) < \text{in}(v)$$

↳ pak není most

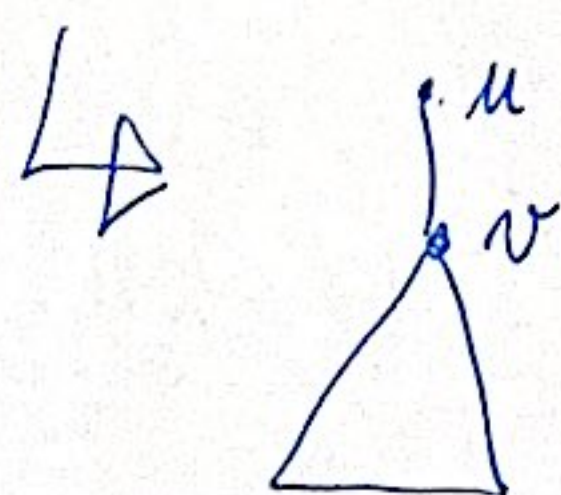


$$\text{low}(v) = \min \left(\{ \text{low}(v_1) \dots \text{low}(v_k) \} \cup \{ \text{in}(z) : vz \in E \text{ zpětná} \} \right)$$

lze najít v $\Theta(n+m)$

↓
pokud neexistuje další
vrchol spojený s v, co
dosáhne už mimo $T(v_1) \dots$
 $T(v_k)$

Stromová hrana není most $\Leftrightarrow \text{low}(v) \leq \text{in}(u)$



Acyklicky orientované grafy $\rightarrow$ DAG-y

(2)

Def.: Graf G je DAG („Directed acyclic graph“) $\Leftarrow$ nemá (orientované) kružnice

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

$\hookrightarrow$ nebo-li počet šipek směřujících do vrcholu je 0

Stok je vrchol s $\deg^{OUT}(v) = 0$

$\hookrightarrow$ nebo-li nevychází z něj žádná hrana



Lemma: Každý konečný DAG má alespoň 1 zdroj a stok

Dk.: Vyberu vrchol $v \rightarrow$ je stok $\checkmark$

$\rightarrow$ nemá stok $\Rightarrow$ vede z něj hrana do v_1

1) narazím na stok $\checkmark$

2) pokračuje do $\infty \rightarrow$ spor s graf je konečný

3) stok nemajdeme $\rightarrow$ spor s graf je bez kružnice

Pro zdroj: Pokud v je stok v $G \Rightarrow v$ je zdroj v G^T

$\nearrow$
otočím šipky

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

☺ G je DAG $\Leftrightarrow G^T$ je DAG (Transpozice cyklu je cyklus)

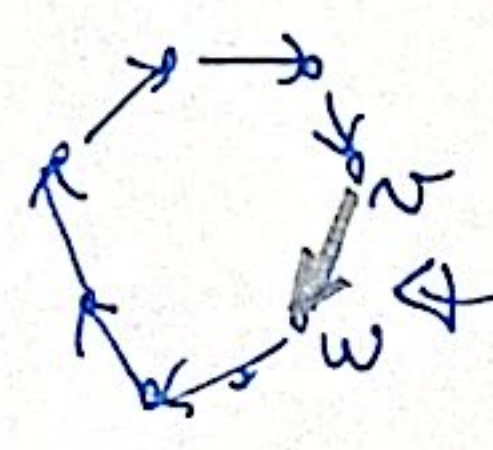
☺ v je zdroj v $G \Leftrightarrow v$ je sink v G^T

Algoritmus: (JE G DAG? $\Leftrightarrow G$ NEMÁ CYKLUS?) ? JE TO EKVIVALENTNÍ? ANO

↳ ano, pokud projdeme DFS a najdeme zpětnou hranu

☺ Lemma: Na každém cyklu leží alespoň 1 zpětná hrana

Ok.: Najdu vrchol s nejmenším $out(v)$



tedy $out(v) < out(w)$, což charakterizuje zpětnou hranu

↳ Věta: G JE DAG $\Leftrightarrow$ OPAKOVANÉ DFS NENAJDE ZPĚTNOU HRANU

Def.: Topologické uspořádání je uspořádání $<$ pokud pro každé $uv \in E$ platí $u < v$

↳ řadí se podle topologického číslování

Věta: G MÁ T.U. $\Leftrightarrow G$ JE DAG

Dk.: $\Rightarrow$ MÁ T.U. $\Rightarrow$ NEMÁ CYKLUS $\Rightarrow G$ JE DAG

$\Leftarrow$ POSTUPNÉ ODTRHÁVÁNÍ ZDROJŮ

$\hookrightarrow$ vznikne mi seřazené uspořádání $\overset{1}{\bullet} \overset{2}{\bullet} \overset{3}{\bullet}$

$\hookrightarrow$ běží v $\mathcal{O}(n \cdot m)$, ale ke lineárnímu pohledu si můžeme sestrojit

$\hookrightarrow$ G MÁ JEDNOZNAČNÉ T.U. $\Leftrightarrow$ TENHLE ALGORITMUS MÁ VŽDY JEN 1 ZDROJ NA VÝBĚR

Věta: Opakované DFS opouští vrcholy v opačném topologickém uspořádání. $\nwarrow$ RPO (reverse - post order)

Dk.: (topologická indukce)

$out(v)$ očíslová vrcholy a dostaneme topologické uspořádání

$\forall u, v \in E: u \preceq v \equiv out(u) > out(v) \Leftrightarrow$ na uv klesá out

$\Leftrightarrow uv$ není zpětná

$\hookrightarrow$ v DAG-u nejsou zpětné hrany a lze tohle provést pro každou hranu.

Využití T.V. : Linearizace číselného uspořádání

(5)

- Č.u. je DAG, kde vrcholy jsou prvky množiny a hrany jsou nerovnosti $\rightarrow$ tak se linearizují $\leq$

Algoritmus: Počet cest z u do v (v DAG u)

$\rightarrow C(x) :=$ počet cest z u do x

$\rightarrow$ indukci

- $\hookrightarrow$ předchůdc p leží před x v cestě a z indukčního předpokladu známe jejich počet cest $C(x) = \sum_{p \in E} C(p)$

④ $(n+m) \rightarrow$ konstantní čas na uzel i hranu

Algoritmus: Délka nejdelší / nejkratší cesty z u do v v DAG u

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

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

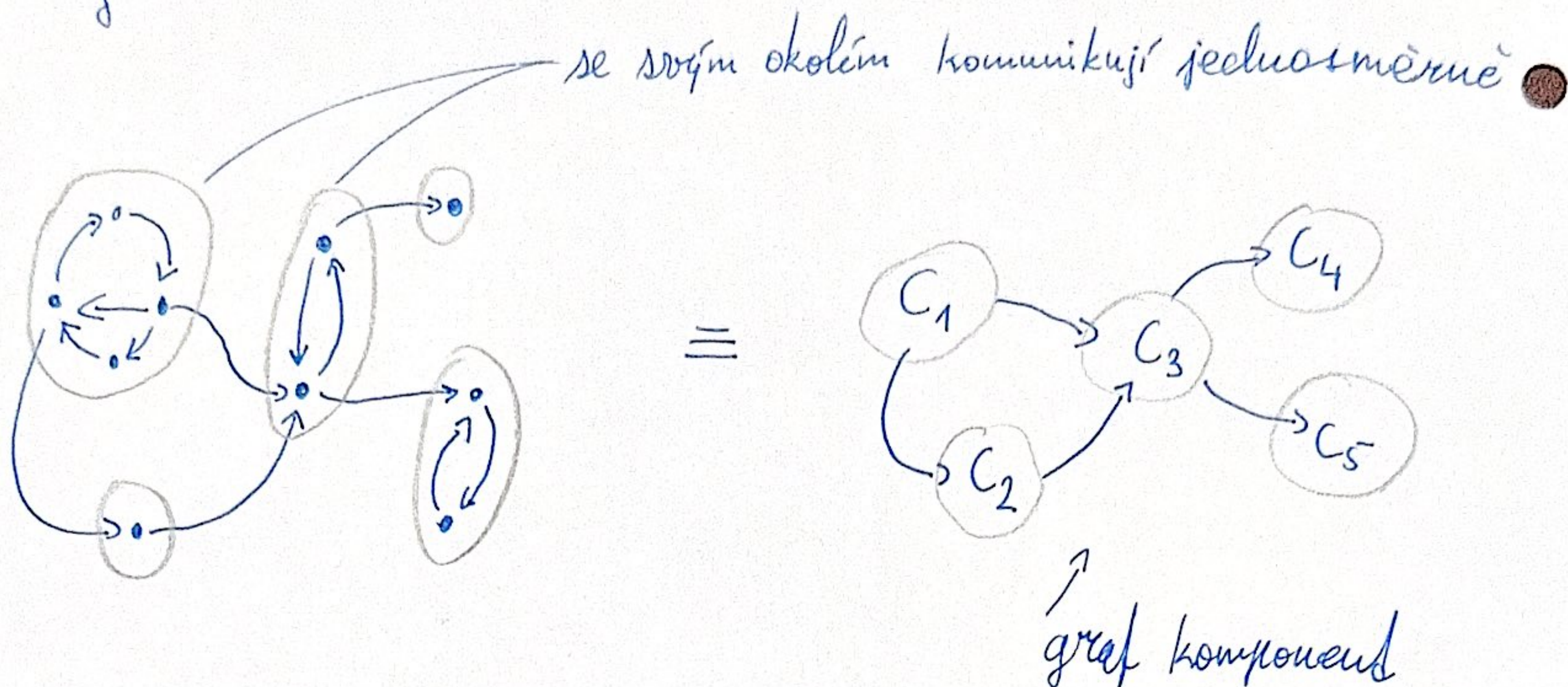
Komponenty silné souvislosti

Def. Necht' $G=(V,E)$ a relace $\rightarrow, \leftrightarrow$ na V :

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

Def.: Komponenty silné souvislosti jsou podgrafy indukované ekvivalenčním vztahem $\leftrightarrow$.

Příklady KSS:



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

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

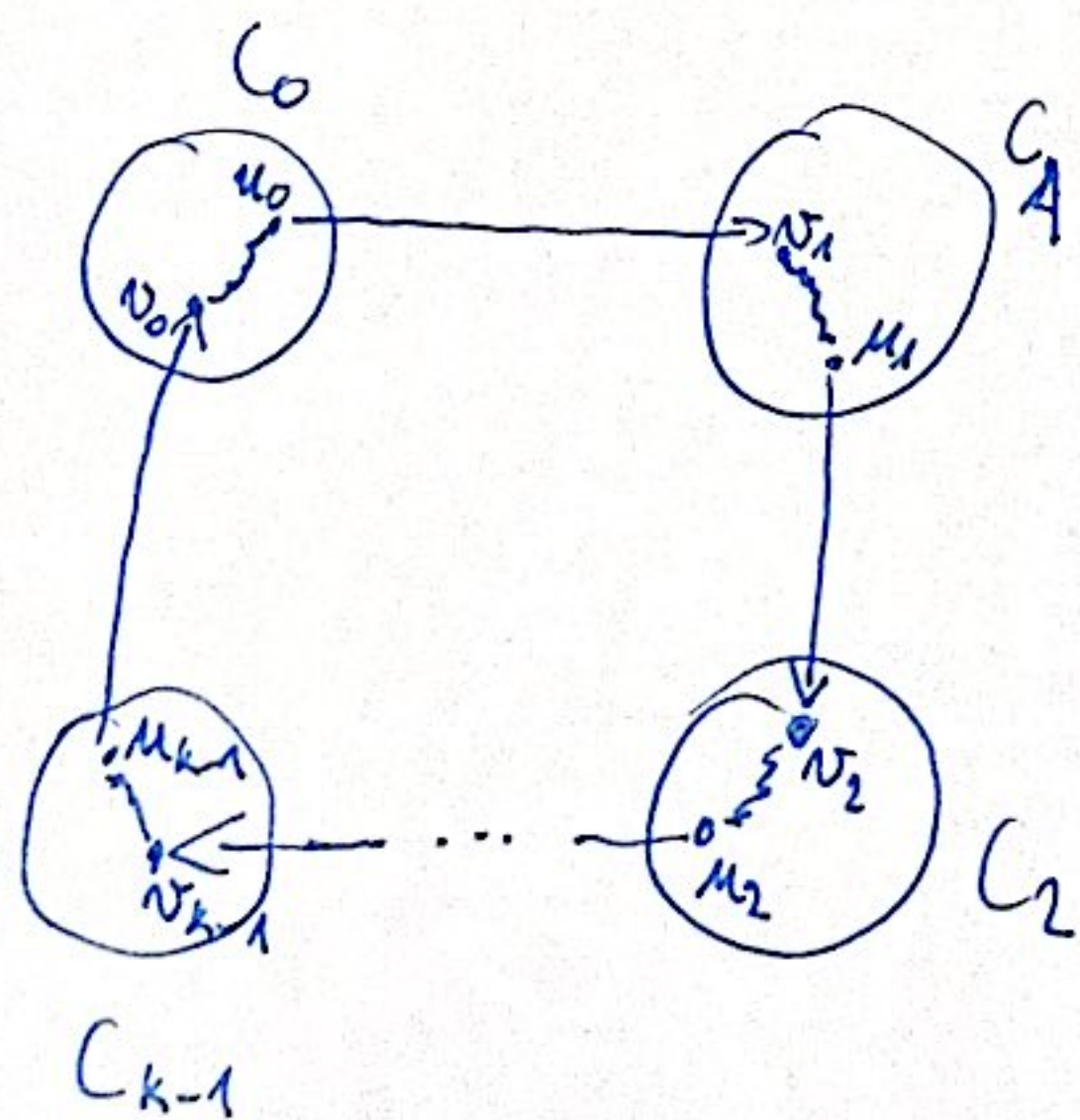
2) $C_1 C_2 \in E(K(G)) \iff \exists v_1 \in C_1, v_2 \in C_2 : v_1 v_2 \in E(G)$

↗ hrany kondenzace

↑ lze vytvořit z G postupnou kontrakcí hran mezi vrcholy ve stejné komponentě

Věta: Pro každý G je $K(G)$ DAG prostě nebude mít cyklus

Dk.: Kdyby v $K(G)$ byl cyklus $C_0 C_1 \dots C_{k-1} C_0$



z definice

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

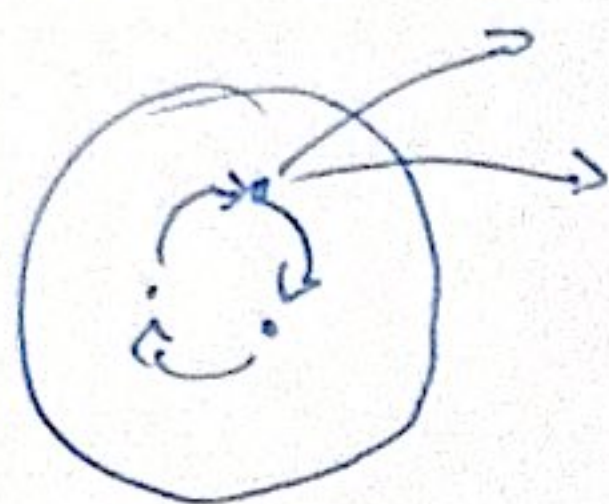
$\forall i \exists$ cesta z u_i do v_i uvnitř C_i

↳ pak u_i, v_i leží na cyklu a tedy

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

Úkol: Můžeme rozlišovat zdrojové a stokové komponenty.

zdrojová



stoková



Pozorování:

👁️ (1) Je-li C stoková komponent a $v \in C$, tak $\text{DFS}(v)$ projde max. všemi vrcholy v C .

👁️ (2) Vrchol s max. $\text{out}(v)$ v opakovaném DFS leží ve zdrojové komponentě. (Vrchol s min. $\text{out}(v)$ leží ve stokové)
 $\hookrightarrow$ v DAGu

👁️ (3) Pokud $K(G)$ obsahuje zdroj i stok $\Rightarrow G$ obsahuje zdrojovou i stokovou komponentu

👁️ (4) $K(G^T) = (K(G))^T$ (ZDROJOVÉ KOMPONENTY $\leftrightarrow$ STOKOVÉ KOMPONENTY)
 $\nearrow$
 mění se zdroj, ale zachová

chybí: algoritmus opakovaně opakovaného DFS na G^T nalezení stromové komponenty a jejich utřídění DFS na G

1) Zkonstruuj G^T $\leftarrow$ do zásobníku uložíme podle out sestupně

2) Opakované DFS na G^T , vrcholy ukládat na zásobník S

3) $\forall v \in V : \text{komp}(v) \leftarrow \text{nedefinováno}$

4) dokud $S \neq \emptyset$:

5) $v \leftarrow S$

6) dokud $\text{komp}(v) = \text{nedefinováno}$

7) DFS(v) a nastavit pro všechny nalezenné vrcholy stejnou komponentu

Nejkratší cesty

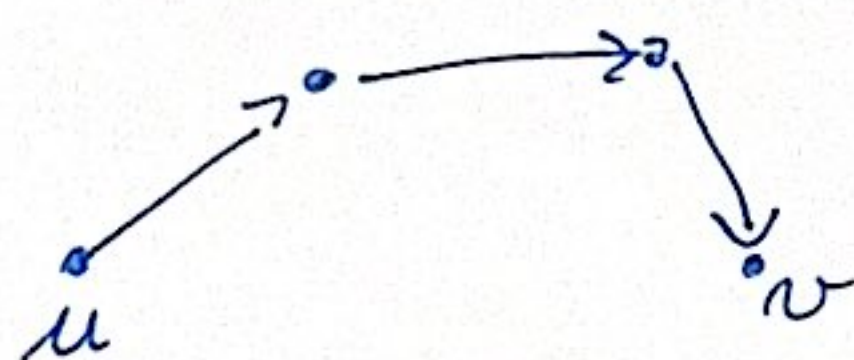
Ohodnocený graf

↳ $G = \text{orientovaný graf}$

↳ $l: E \rightarrow \mathbb{R}$ (délky hran) - funkce

↳ obvykle $\forall e: l(e) \geq 0$

sled s



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

Definice: Vzdálenost z u do v :

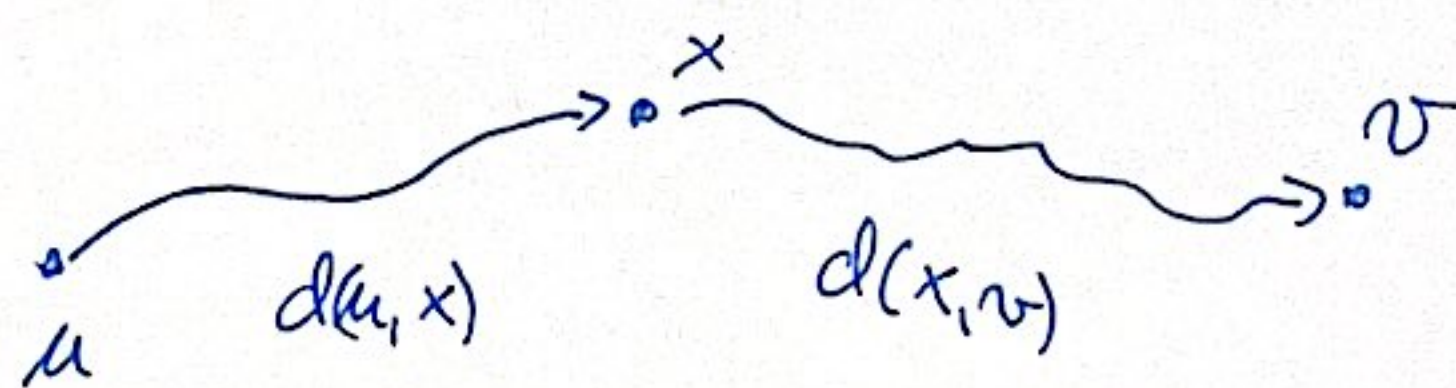
$$d(u, v) := \min_{\substack{P \text{ je cesta z } u \text{ do } v}} l(P) \quad \left. \vphantom{\min} \right\} \begin{array}{l} +\infty \text{ pokud z } u \text{ do} \\ v \text{ nevede žádná} \\ \text{cesta} \end{array}$$

☹ $d(u, u) = 0$

$d(u, v) = d(v, u)$ - nemusí platit v orientovaném grafu, platí jen v neorientovaném

$$d(u, v) \leq d(u, x) + d(x, v) \quad (\Delta\text{-nerovnost})$$

Důkaz:



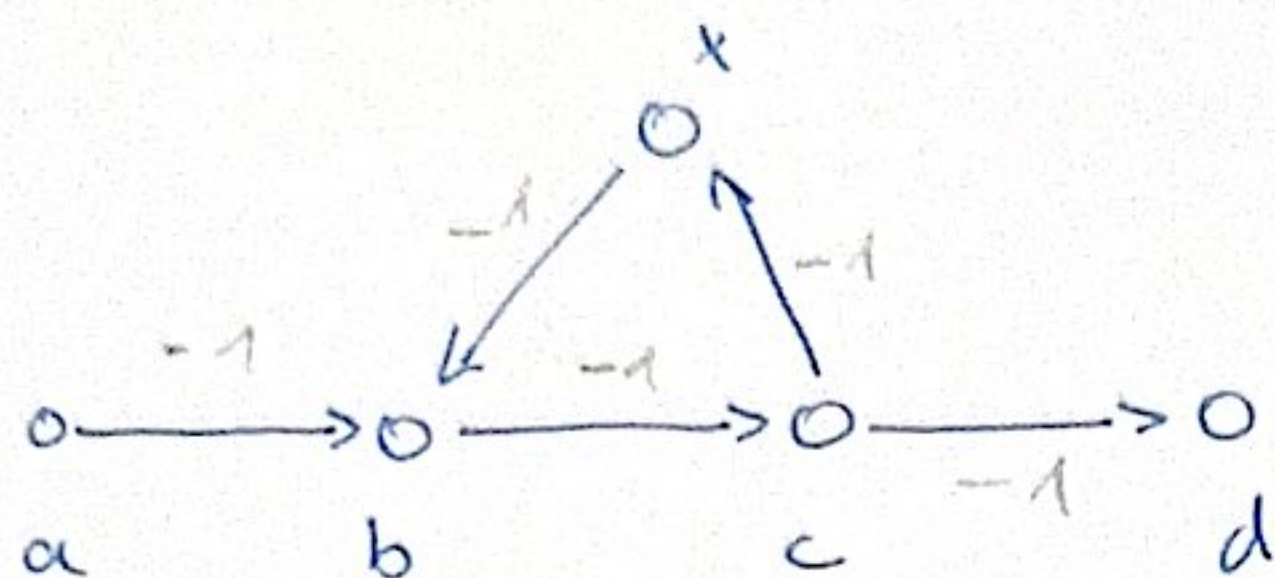
∃ uv cesta, co není delší.

Lemma: $\forall uv$ sled s existuje uv cesta P t.č. $l(P) \leq l(s)$

sled

↳ mohou se
opakovat
hrany i
vrcholy

Co když připustíme $l < 0$?



$d(a, d) = -3$ jako cesta

"jako sled
neexistuje"

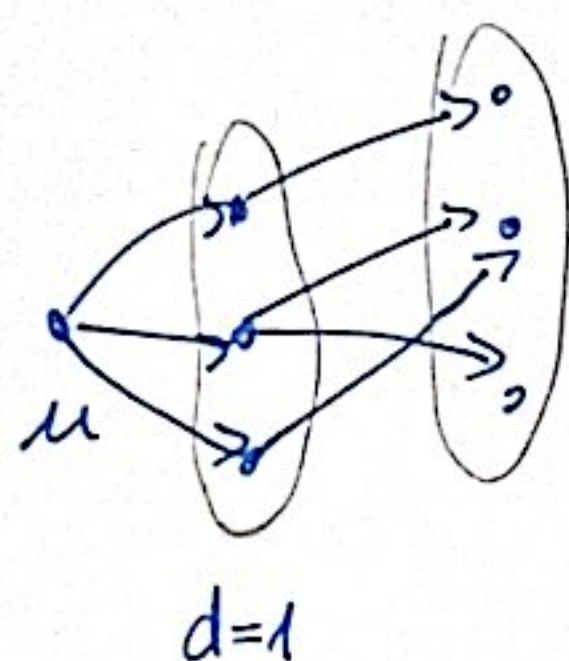
$d(a, d) \leq d(a, x) + d(x, d)$

$-3 \leq -3 + -3$

} rozbita se
 Δ -nerovností

↳ nemají se polynomiální algoritmus pro graf s Δ -nerovností
cykly

Pro $l=1$ je řešení BFS (prohledávání do šířky)



hrany: -stromové

-příčné

-zpětné

-dopředné příčné

↳ pomocí fronty v $\Theta(n+m)$

BFS s podrozdělováním hran

↳ hrany podrozdělím na jednostranné hrany a pak BFS

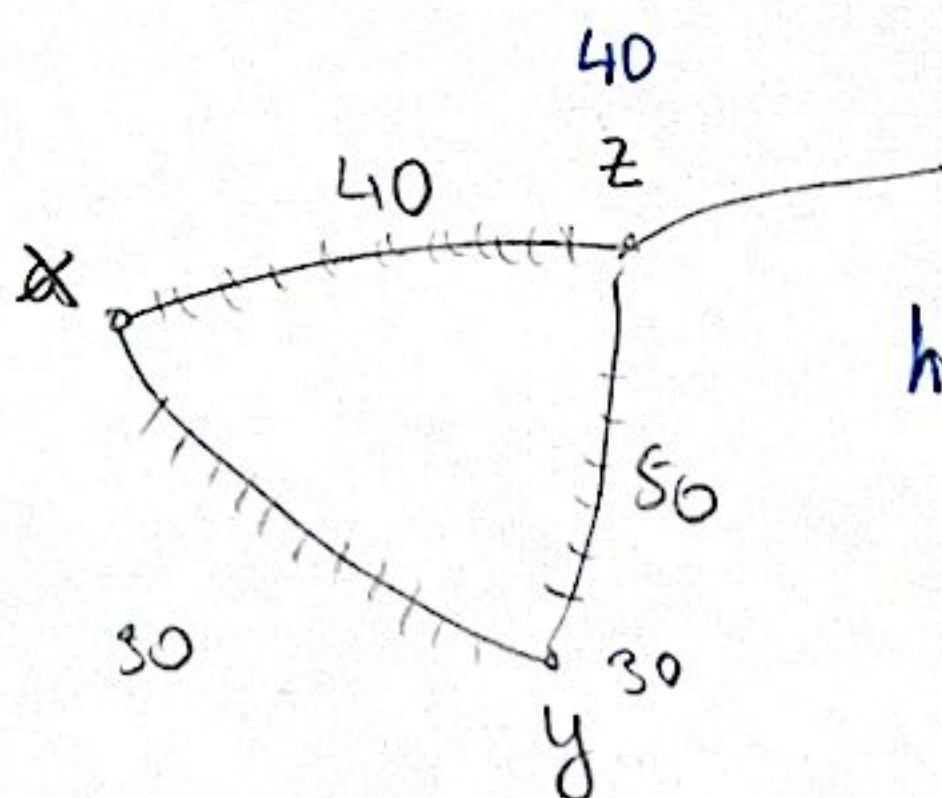
↳ pokud nejsou hrany malé, tak se z něj stane
obrovský graf

$1 \leq l(e) \leq L$

$O((n+m)L)$

pro pamatování cesty

↳ v každém vrcholu
si pamatují předchůdce



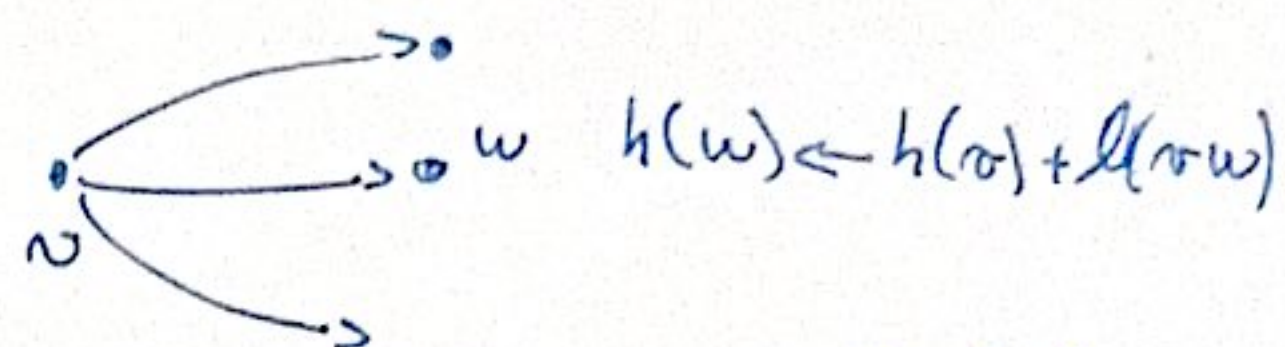
$h(w) :=$ "budík"

v jakém čase
do vrcholu w

(19)

Dijkstraův algoritmus

↳ podívám se na všechny následující vrcholy a zeptám se, zda nemám přenastavit budík



$stav(v) = \begin{cases} \text{neviděný} \\ \text{otevřený} \\ \text{zavřený} \end{cases}$ ← ty vrcholy, co mají nastavený budík

Algoritmus

Inicializace

1. $\forall v \in V: stav(v) \leftarrow \text{neviděný}, h(v) \leftarrow +\infty, P(v) \leftarrow \emptyset$] $\mathcal{O}(n)$
2. $stav(u) \leftarrow \text{otevřený}, h(u) \leftarrow 0$ Insert

3. Dokud $\exists v: stav(v) = \text{otevřený}$:

4. Extract Min
Vybereme $v: stav(v) = \text{otevřený}, h(v)$ minimální

5. Pro $\forall vw \in E$:

6. $\mathcal{O}(n)$ ≤ n přechodů $\mathcal{O}(n^2)$
Je-li $h(w) > h(v) + l(v, w)$:

7. $h(w) \leftarrow h(v) + l(v, w)$ Insert
 $P(w) \leftarrow v$ Decrease

8. $stav(w) \leftarrow \text{otevřený}$ =

9. $stav(v) \leftarrow \text{zavřený}$

je dobrý pro husté grafy,
ale ne pro řídké

Věta: Dijkstraův algoritmus se zastaví v case $\mathcal{O}(n^2)$, po

zastavení $\forall v: h(v) = d(u, v)$

předchůdci → $P(-)$ tvoří skom nejkratších cest

$\mathcal{O}(n + m + n \cdot T_{\min})$

↳ hodně času zabírá hledání minima

Datová struktura

Operace Insert (vloží prvek a řadí mu konkrétní hodnotu)

Extract Min (najde prvek s nejmenší hodnotou a vrátí ho)

Decrease (zmenší hodnotu prvků)

→ použijeme na množinu všech sledovaných vrcholů (max. n)

Dijkstra algoritmus má složitost

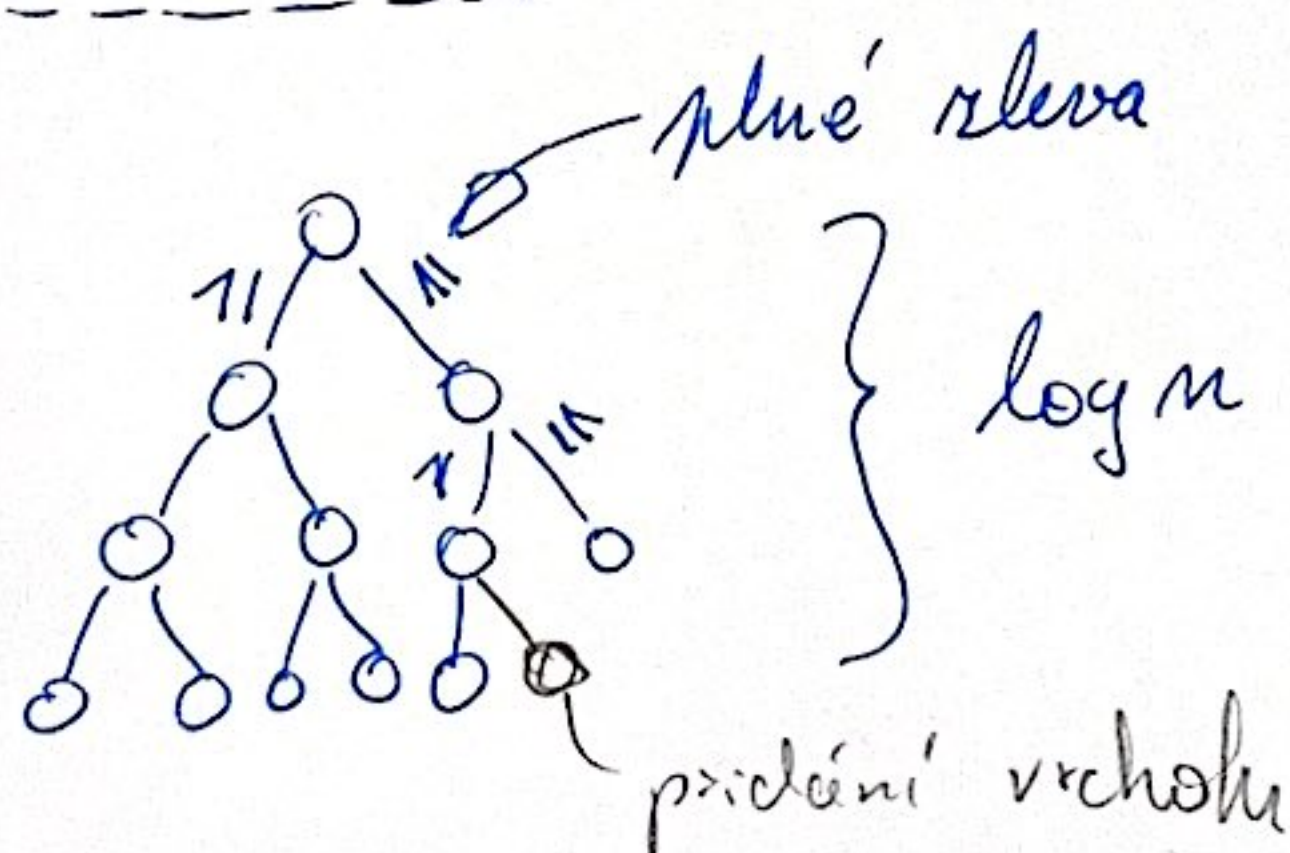
$$O(n \cdot T_{\text{ExtractMin}} + n \cdot T_{\text{Insert}} + n \cdot T_{\text{Decrease}})$$

↖ přes každou hranu jdeme nejvýše jednou

↗ $+ n + n$
inicializace,
asymptoticky
se vejde

	pole	set. pole	bin. haldá	Fibonacciho haldá	d-řez. haldá
Extract Min	n	n	$\log n$	$\log n$	
Insert	1	n	$\log n$	1	
Decrease	1	n	$\log n$	1	
○	n^2	$n \cdot n$	$(n+m) \cdot \log n$	$(n \cdot \log n) + m$	
řidké $m \in O(n)$					
husté $m \in O(n^2)$					

Binární haldá



kořen = min

Extract Min → prochodit kořen
a poslední prvek → bubláni
dolů

Insert → přidat nakonec +
bublání nahoru

Decrease → snížit hodnotu a
bublání nahoru

logaritmy

○ různých
základů

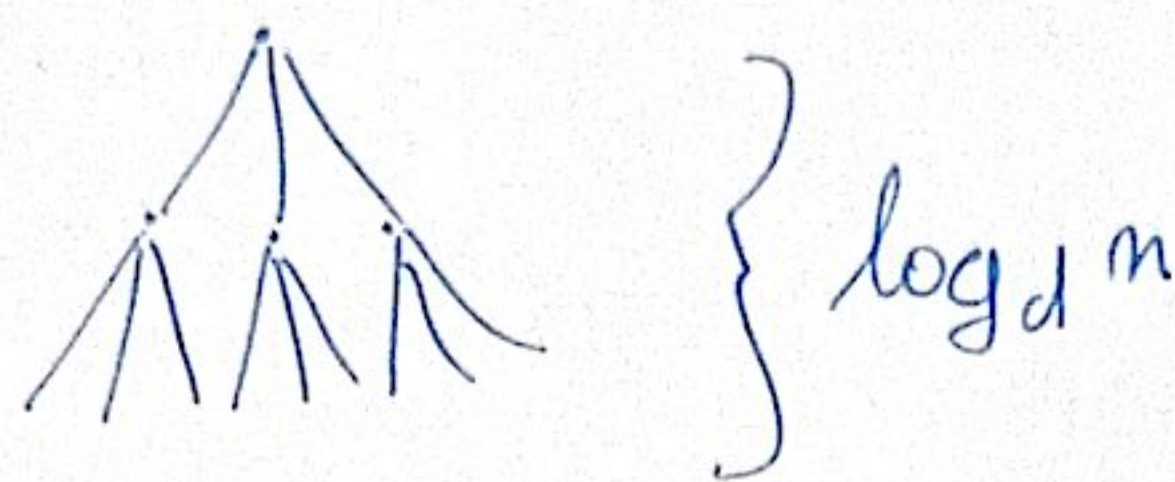
jsou
asymptoticky
stejně

→ procházím s
menší

pokud menší, tak
hodím nahoru

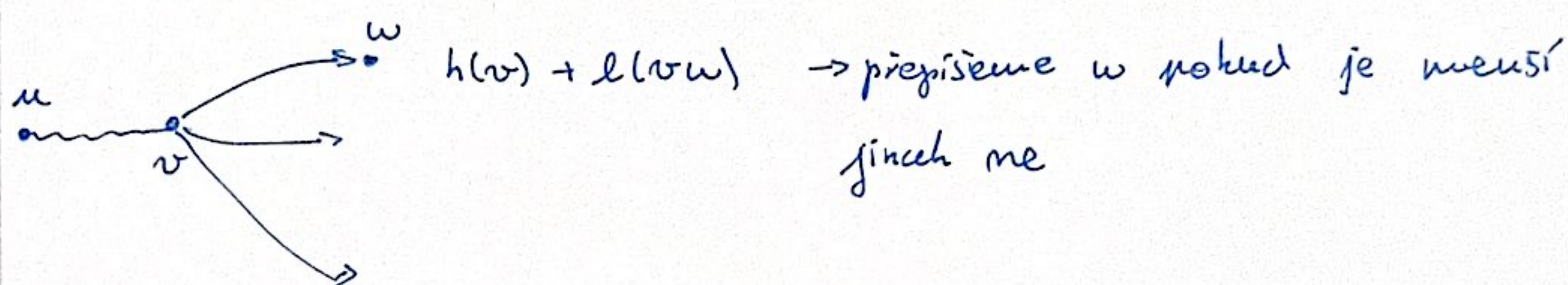
d-regularní haldy

nadobčítání



ohodnocení $h: V \rightarrow \mathbb{R}$

relaxace $\rightarrow$



Pro $vw \in E$: Pokud $h(w) > h(v) + l(v,w)$:

$h(w) \leftarrow h(v) + l(v,w)$
 $stav(w) \leftarrow \text{otevřený}$

$stav(v) = \begin{cases} \text{neviděný} \\ \text{otevřený} \\ \text{uzavřený} \end{cases}$

Obecný relaxační algoritmus

1. $\forall v \in V: h(v) \leftarrow +\infty, stav(v) \leftarrow \text{neviděný}$
2. $h(u) \leftarrow 0, stav(u) \leftarrow \text{otevřený}$
3. Pokud $\exists v: stav(v) \leftarrow \text{otevřený}$:
4. $stav(v) \leftarrow \text{uzavřený}$
5. Relaxujeme v

Invariant 0 (ohodnocení):

- ① $h(v)$ nikdy neroste
- ② Pokud $h(v)$ je konečné, pak existuje stezka délky $h(v)$ z u do v

zobecnění
Dijkstrava
algoritmu

$\hookrightarrow$ nemůžeme si
obecně nejmenší

Důkaz:

indukcí podle běhu

$\Rightarrow$ mám podle IP sled uv , dle uv

$u \xrightarrow{\quad} v \xrightarrow{\quad} w \quad h(w) = h(v) + l(wv)$

Lemma 0 (dosazitelnost)

$\triangle$ tak už nejsou žádné otevřené uzly

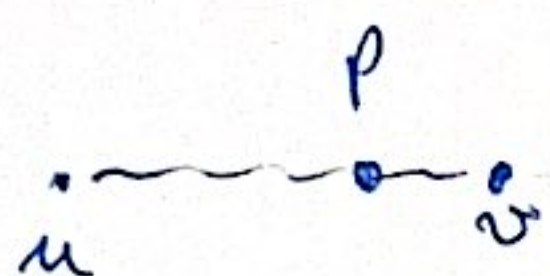
$\hookrightarrow$ pokud se algoritmus zastaví, pak

$\forall v: \text{slav}(v) = \text{uzavřený} \Leftrightarrow v$ je dosazitelný z u

$\Rightarrow$ Když existuje sled, tedy $h(v)$ má konečnou, tak je dosazitelný $\rightarrow$ z invarianty $\odot$

$\Leftarrow$ vezmu nejbližší špatný uzel v

$\hookrightarrow$ nejbližší $v \neq \text{hran}$



$\Leftarrow$ protože p je dobrý, tak jsme objevili v , tedy spou s tím, že v je špatně

Lemma V (vzdálenost)

Pokud se algoritmus zastaví, pak $\forall v \in V: h(v) = d(u, v)$

Důkaz

a) pro v nedosažitelné z u : $\infty = \infty$

b) pro v dosažitelné

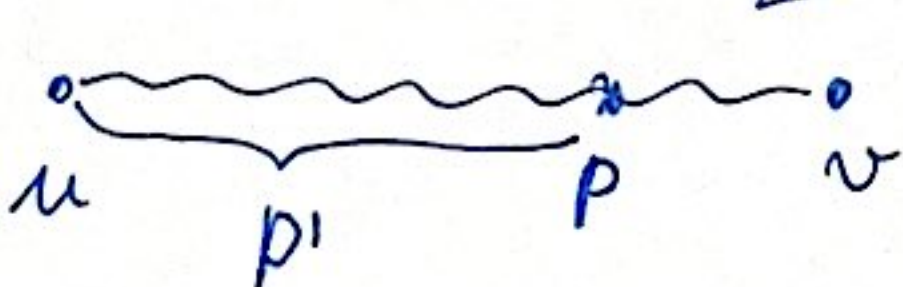
$\hookrightarrow$ kdyby $\exists v \in V: h(v) \neq d(u, v)$, tak pak určitě $h(v) > d(u, v)$

Vybereme v špatný t.j. nejkratší cesta z u do v má

nejmenší $\# \text{hran}$

$\triangle$ nejkratší cesta P , kde $l(P) = d(u, v)$

$\uparrow$ $h(v)$ neroste,
jen se snižuje



nejkratší z u do p

$h(p) = d(u, p) = l(P')$

$\Leftarrow$ pokud byl p relaxován a otevřen,

pak když alg. skončil, tak musel

být uzavřen, tedy musel najít v

prefix nejkratší
cesty je
pořadí
nejkratší
cesta

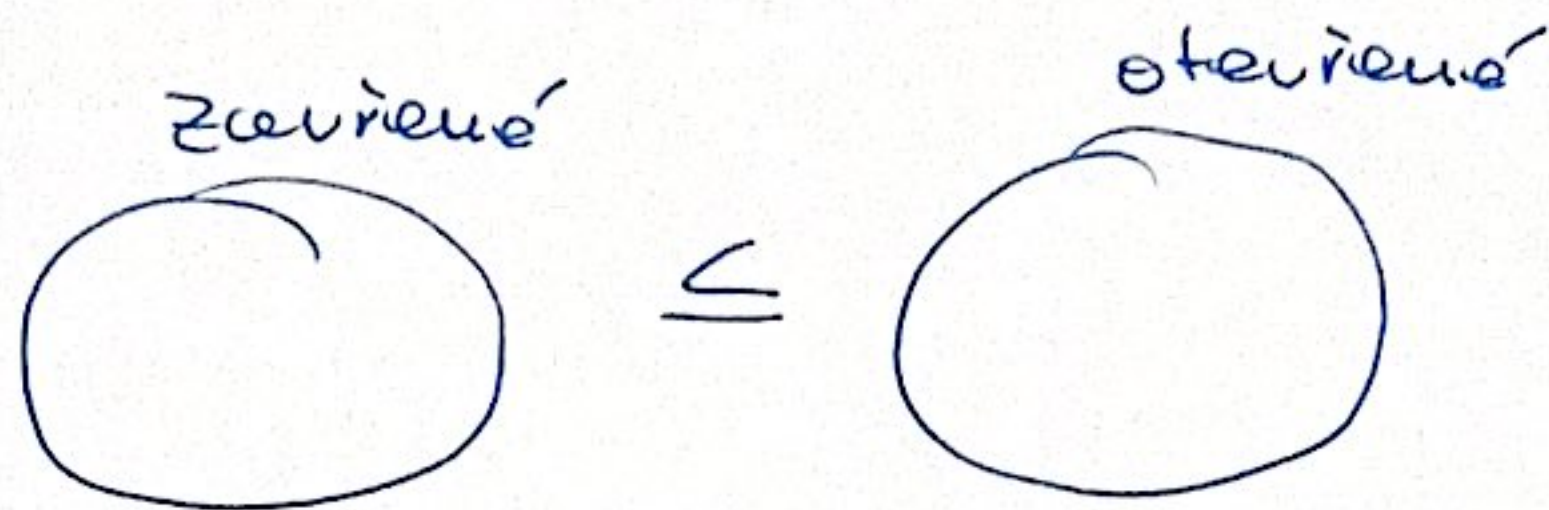
Dijkstra algoritmus (relaxaci s výběrem min $h(v)$)

bez
záporných
hran
 $\forall e: l(e) \geq 0$

Invariant M (o monotonii)

① Pro v otevřený a z uzavřený

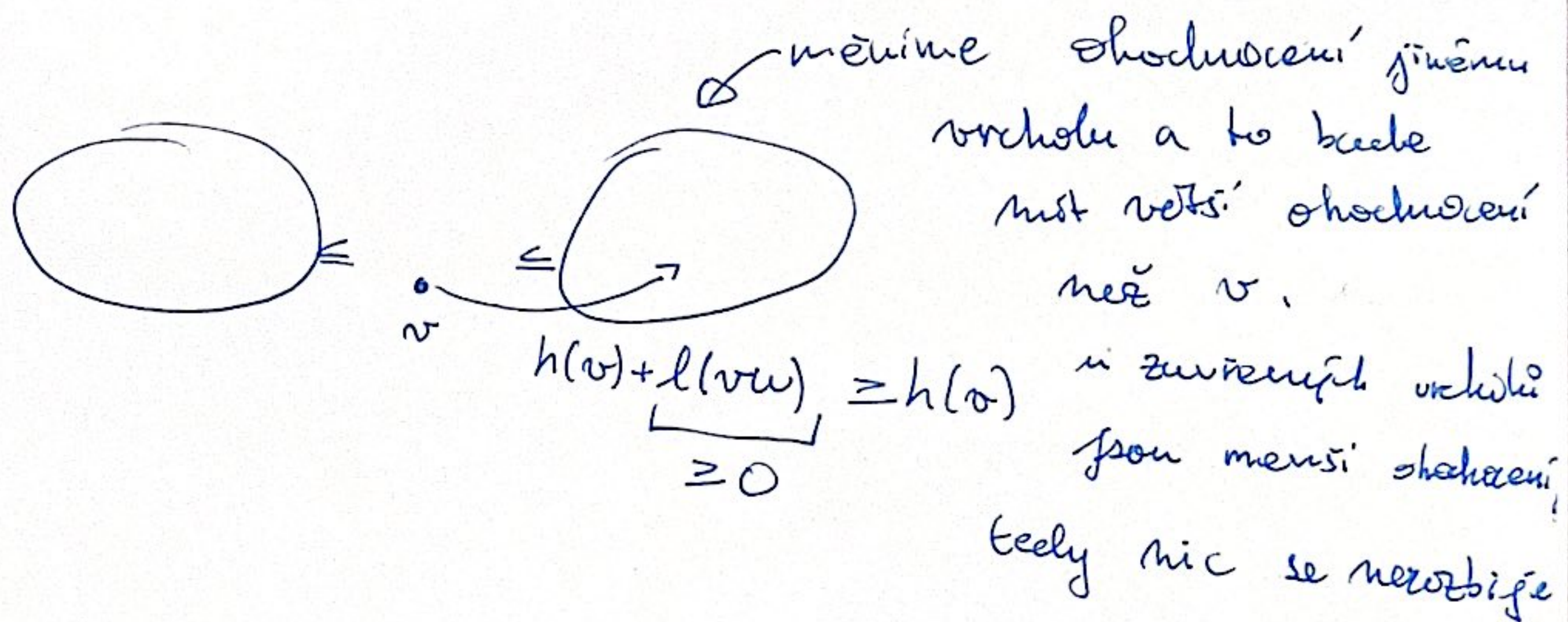
$$h(z) \leq h(v)$$



② ohodnocení uzavřených vrcholů se nikdy nemění

↳ každý vrchol uzavře max. jednou

Dk. (indukcí)



lema: Na grafu bez záporných hran D.a. uzavře každý vrchol maximálně jednou

V pořadí nerostoucích $d(u, _)$

V okamžiku uzavření v je $h(v) = d(u, v)$ a dále se nemění.

Bellmanov - Fordov

↖ uzavře se nejdříve otevřený vrchol

↳ relaxace + otevřené vrcholy ve frontě

Def. fáze běhu

$F_0, F_1, \dots$

$F_0 \rightarrow$ otevření u

$F_1 \rightarrow$ odeberu u z fronty a přidám jeho sousedství do fronty

$F_{i+1} \rightarrow$ uzavíráme vrcholy otevřené v F_i

Invariant F (o fázích)

$\forall i$:
↳ na konci F_i je $\forall v: h(v) \leq$ délka nejkratšího stezky z u

do v o maximálně i hranách

$\Downarrow$

$\# \text{ fází} \leq n \Rightarrow$ čas $O(nm)$

↳ na každé fázi max.
 m hran a m
vrcholů

proto na konci
 n -té fáze
je vše hotovo

Důkaz (indukcí podle i)

• $i=0$ ✓

• $i-1 \rightarrow i$

Vrchol v na konci F_i

$S :=$ nejkratší stezka z u do v o max. i hranách

Chceme $h(v) = l(S)$



↳ nejkratší stezka z u do p o max. $i-1$ hranách

① p na $i-1$ a p

Na konci F_{i-1} je $h(p) \leq l(S')$

② Nejpozději v F_{i-1} bylo uzavřeno $h(p) \Rightarrow p$ otevřeno

③ Nejpozději v F_i je p uzavřeno. V tu chvíli stále $h(p) \leq l(S')$

④ Po relaxaci p máme:

$$h(v) \leq \underbrace{h(p)}_{\leq l(s')} + l(pv) \leq l(s)$$

⑤ pokud $h(v)$ neroste, takhle na konci F_i je stále

$$h(v) \leq l(s')$$

┐ $d_i(u, v) := \text{min. délka sledu } u \rightarrow v \text{ o } i \text{ hraních}$

$$d_0(v, u) = 0$$

$$v \neq u \quad d_0(u, v) = +\infty$$

$$d_i(u, v) := \min_{p \in E} (d_{i-1}(u, p) + l(pv))$$

ne v testu →

tedy $O(n \cdot m)$

Floyd-Warshallův algoritmus

vrcholy $1, \dots, n$

matice
délky hran →

$$L_{ij} := \begin{cases} l(i, j) & \text{pokud } ij \in E \\ +\infty & \text{jinak} \\ 0 & \text{pro } i=j \end{cases}$$

omezují a postupně uvolňují

$$D_{ij}^k := \text{min. délka cesty z } i \text{ do } j \text{ přes } \{1, \dots, k\}$$

$$D_{ij}^0 = L_{ij}$$

$$D_{ij}^n = d(i, j)$$

$$D^{k-1} \rightarrow D^k$$



buď použit nebo nepoužit

$$D_{ij}^k = \min \begin{cases} D_{ij}^{k-1} \\ D_{ik}^{k-1} + D_{kj}^{k-1} \end{cases}$$

Pro $k = 0, \dots, n:$

Pro $i, j = 1, \dots, n:$ $\swarrow$ 2 cesty

$$D_{ij}^k \leftarrow \min (D_{ij}^{k-1}, D_{ik}^{k-1} + D_{kj}^{k-1})$$

$\swarrow$ bezpečné
delší hran

nemusí být cesta, ale sled, ale nic nevybereme chvilku

$\swarrow$ tedy sledy
nevyberem,
protože
bylo u delší

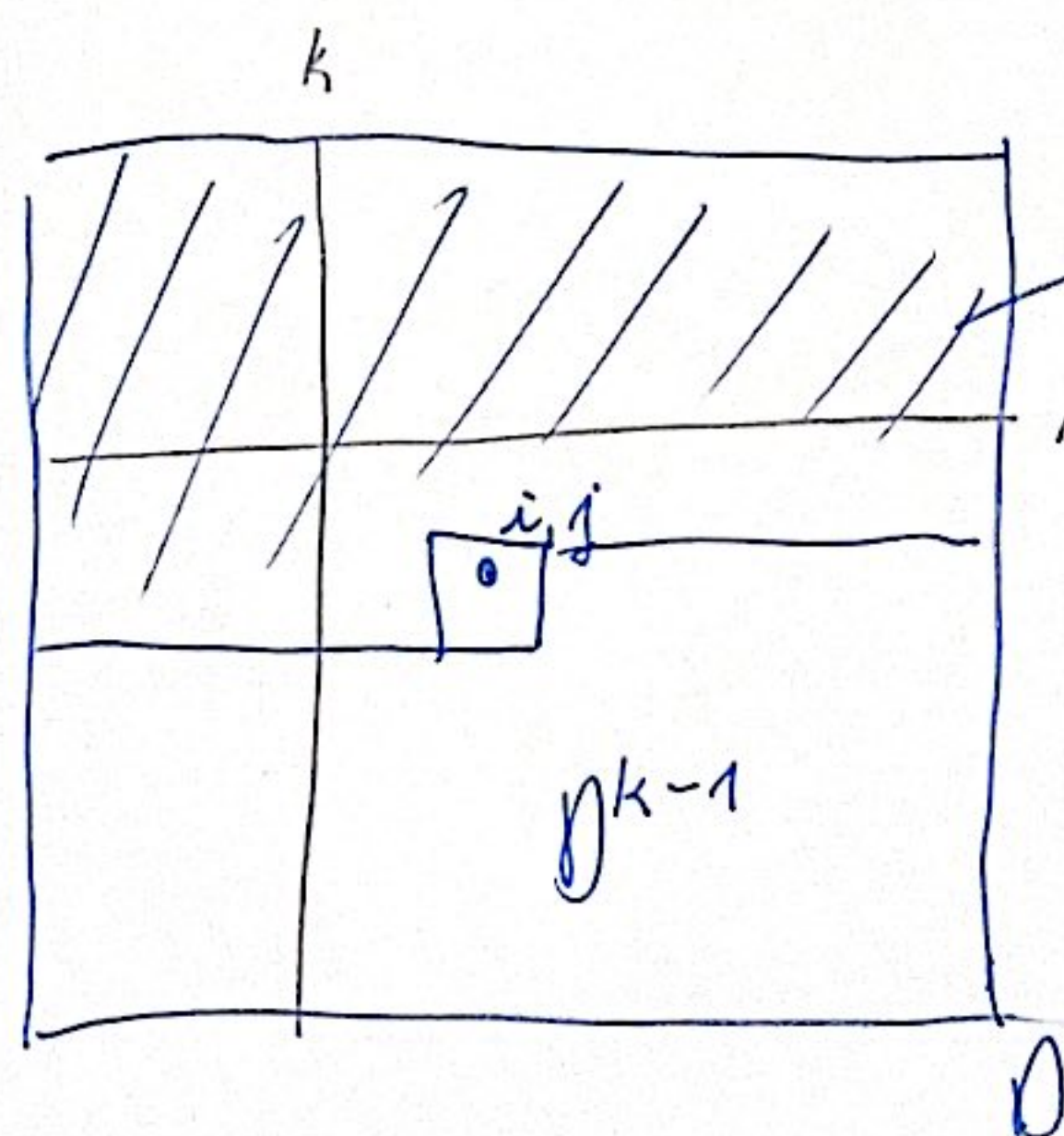
časová složitost $\Theta(n^3)$

prostorová složitost $\Theta(n^3)$ $\swarrow$ ne krychli

$\swarrow$ u záporných cyklů
neplatí

$\hookrightarrow$ ale stačí nám si pamatovat jen 2 matice $n \times n$, tedy
 $k-1$ a k matice

$\hookrightarrow$ stačí nám si přepisovat vše v jedné matici



D^k už prošli

$\swarrow$ tedy se to může pokazit, ale
 k použít stejně nemůžeme,
protože by to byl sled

Pro $k = 0, \dots, n:$

Pro $i, j = 1, \dots, n:$

$$D_{ij} \leftarrow \min (D_{ij}, D_{ik} + D_{kj})$$

7 Minimální kostry 7

Definice: Minimální kostra = "problém minimální kostry"

$G = (V, E)$ neorientovaný souvislý graf

$w: E \rightarrow \mathbb{R}$ váhová funkce, ohodnocení hran

$H \subseteq G$ podgraf H grafu G

$\hookrightarrow w(H) := \sum_{e \in E(H)} w(e)$ součet ohodnocení hran podgrafu H grafu G

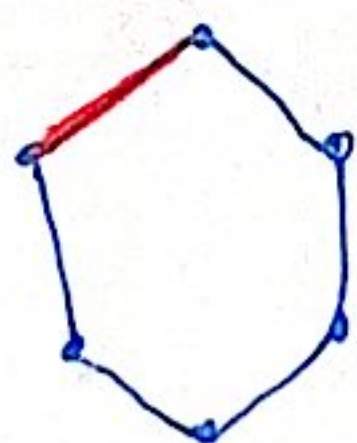
Problém (minimální kostry grafu):

$\hookrightarrow$ Hledáme $T \subseteq G$ s minimálním $w(T)$, tedy kostra T obsahuje všechny vrcholy a má mezi všemi kostřemi nejmenší váhu

! Kostra není jednoznačná

$\hookrightarrow$ Jednoznačná jen pro všechny váhy hran různé

např.:



$\rightarrow$ odebráním 1 hrany z šesticyklu lze získat 6 různých koster

např. K_n má n^{n-2} koster

předpoklad

$w(e) \neq w(e') \Leftrightarrow e \neq e'$

$\hookrightarrow$ Nic se nám neztratí, nalezneme jednu z minimálních koster po změně váh hran

(24) Jarníkův algoritmus = „hladový algoritmus“

neformálně: Vyberu si vrchol v_0 z grafu G . Počívám se na incidentní hrany v_0 a vyberu nejlehčí.

Algoritmus: $\approx O(mn)$, n iterací po m hranách

1. $T \leftarrow (\{v_0\}, \emptyset)$
2. Dokud existují vrcholy mimo T :
3. $uv \leftarrow$ hrana s min. $w(uv)$, kde $u \in V(T)$ a $v \notin V(T)$
4. $T \leftarrow T + uv$
5. Vratíme T .

vstup: Souvislý graf s různými váhami

výstup: minimální kostra T

Správnost

lemma: Algoritmus se po max. n krocích zastaví a vydá kostru grafu.

důkaz: v každém kroku přidáme list, po n krocích bude graf obsahovat všechny vrcholy

• (Indukcí)

V $k-1$ kroku je graf kostrou, tedy i v k -tém kroku po přidání listu je kostrou

z roku 1930

dnes:

Primův /
Dijkstrův

hladový algoritmus

↳ pokud si
vybere co nejlepší
pokračováním

↳ málokdy funguje

Řezové lemma

Def. Elementární řez



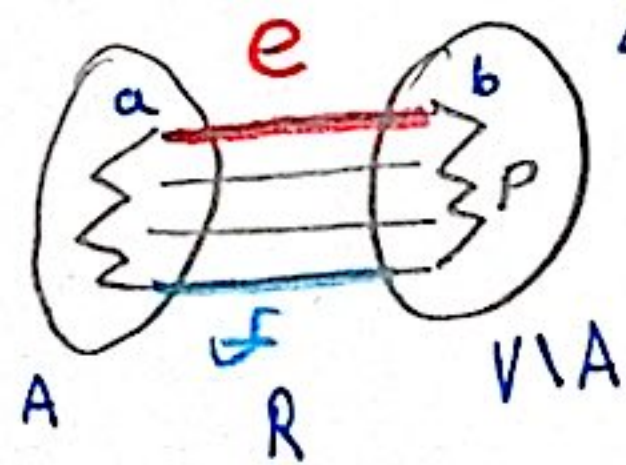
→ množina hran mezi dvěma množinami
vrcholů

Kdykoliv $A \subseteq V$, pak $\{ab \in E \mid a \in A \wedge b \notin A\}$ je elementární řez mezi A a $V \setminus A$

Řezové lemma

Lemma: Necht' R je elementární řez, $e \in R$ je nejlehčí hrana řezu, pak $e \in T$.

Důkaz:



→ Máme elementární řez R , kde $e \in R$ a je nejlehčí, ale $e \notin T$, kde T je min. kostra grafu. Pak existuje cesta mezi vrcholy a a b přes hranu f , ale odebráním hrany f získáme 2 komponenty a přidáním e nalezneme lehčí kostru grafu. Tedy spor s předpokladem, že T je min. kostra grafu. $\perp$

Necht' $e = ab$, $e \notin T$ a $P :=$ cesta mezi a, b v T , kde $e \notin P \Rightarrow$

$\Rightarrow \exists f \in P \cap R : w(f) > w(e) \Rightarrow T' := T - f + e$ je

kostra a $w(T') < w(T)$, což je spor s nejlehčí kostrou T

$$\swarrow$$

$$w(T') = w(T) - w(f) + w(e)$$

$\searrow$

Dokazuje správnost
f.a., protože vybírá
nejlehčí hrana z...
elementárního řezu,
tedy všechny hrany
patří do min. kostry
grafu. A na výstupu
je min. kostra grafu.

↓
je tedy rovnou
každé minimální
kostře.

Důsledky řezového lemmatu

① Lemma: Kostra T nalezená J.a. je minimální.

Důkaz: (Invariant): Hrany mezi T a zbytkem G tvoří elementární

řez $\Rightarrow uv$ je nejlehčí $\Rightarrow uv$ leží v každé min. kostře grafu

T' je lib. kostra grafu $T \Rightarrow T \subseteq T' \Rightarrow T = T'$ (díky stejnému počtu hran)

② Lemma: Existuje právě jedna minimální kostra grafu (pro různé hrany)

Důkaz: Protože $T \subseteq$ každá min. kostra

③ Lemma: Minimální kostra je určena porovnáváním hran, tedy nějakým lineárním uspořádáním.

Jarník ala Dijkstra

J.a. v.2.0.

1. $\forall v$: $stav(v) \leftarrow \text{mimo}$, $h(v) \leftarrow \emptyset$, $p(v) \leftarrow \emptyset$

$\leq n$ $stav(v_0) \leftarrow \text{soused}$, $T \leftarrow \emptyset$

2. Dokud $\exists v$: $stav(v) = \text{soused}$:

3. $v \leftarrow \text{soused}$ s min. $h(v)$

4. $stav(v) \leftarrow \text{uvnitř}$

5. Pokud $p(v) \neq \emptyset$, $T \leftarrow T + p(v)$

6. $\leq m$ Pro všechny $v, v' \in E$:

7. Je-li $stav(v') = \text{mimo}$, $stav(v) = \text{soused}$

$h(v') \leftarrow w(vv') + h(v)$

$p(v') \leftarrow vv'$

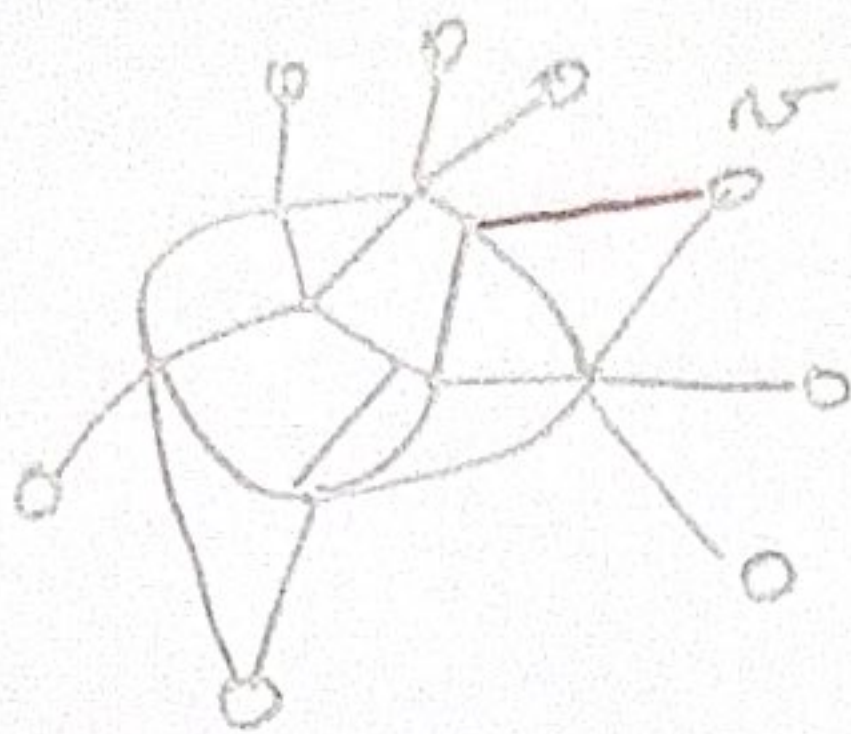
8. Je-li $stav(v') = \text{soused}$ & $h(v') > w(vv')$: $h(v') \leftarrow w(vv')$

$p(v') \leftarrow vv'$

2 smyčky $\rightarrow O(n^2)$

$\hookrightarrow$ vysvětlení na další stránce

aby to byl
Dijkstra alg.



pro každý vrchol si pamatujeme
sousedící hranu.

$p(v) :=$ nejlehčí hrana mezi v a T

$h(v) :=$ délka aktivní hrany

$\Delta_{\text{new}}(v) \begin{cases} \text{zavřený} & (\text{monitri } T) \\ \text{otevřený} & (\text{soused } T) \\ \text{navícený} & (\text{ostatní}) \end{cases}$

složitost

v poli $\rightarrow O(n^2)$

v bin. haldou $\Theta((m+n) \log n)$

v Fib. haldou $\Theta(m \log n + m)$

Boerwink's algorithmus (1926)

vstup: souvislý graf s unikátními váhami

výstup: minimální kostra T

algorithmus

1. $T \leftarrow (V, \emptyset)$

2. Dokud T není souvislý:

3. Rozložíme T na komponenty souvislosti $T_1, \dots, T_k$

4. Pro každý strom T_i najdeme nejlehčí hrana mezi
 T_i a zbytkem grafu

5. Přidáme do T hrany $\{e_1, \dots, e_k\}$

$\rightarrow$ důkaz správnosti na další straně

Inicializace: $\forall$ vrcholy jsou samostatný strom

1. fáze: z každého vrcholu vyberu nejlehčí hrany a tu přidám do kostry T

paralelní 1. f. a.

2. - k. fáze: opakuj - "-

☹ B.a. najde min. kostru $\rightarrow$ z rezového lemmatu

☹ 1 fáze trvá $O(n+m)$

• $O(m) \rightarrow$ pro všechny hrany najdu, mezi kterými stromy vede

• $O(n) \rightarrow$ mergejme stromy

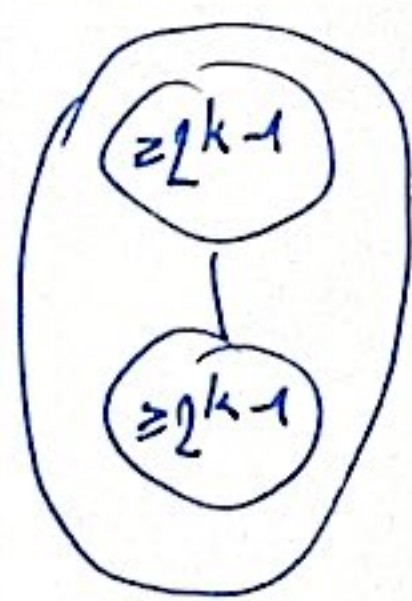
☹ alg. se zastaví

Lemma: Po k -té fázi mají $\forall$ stromy alespoň 2^k vrcholů

Důkaz: (Indukcí)

1) $k=0 \Rightarrow 2^0 \leq 1$

2) $k-1 \rightarrow k$

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

☹ # fází $\leq \log(n)$

☹ Borůvkův algoritmus najde min. kostru v čase $O((n+m) \log n)$

Kruskalův algoritmus

↳ vlastně hrany seřadíme podle velikosti a postupně přidáváme hrany a kontrolujeme, zda nevytvoří cyklus

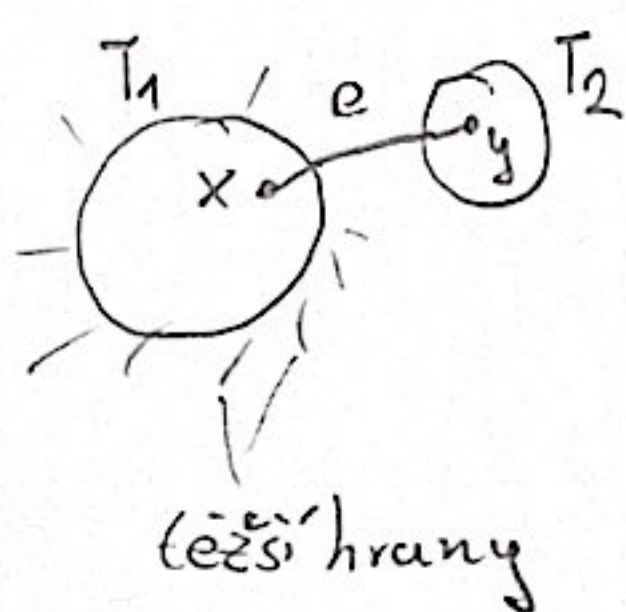
algoritmus

1. Seřadíme hrany podle vah
2. $T \leftarrow (V, \emptyset)$
3. Pro $e \in E$ vzestupně podle $w(e)$:
4. Pokud $T+e$ acyklický: $\quad] \text{ Find}(x, y)$, kde $e = xy$
5. $T \leftarrow T+e$ $\quad] \text{ Union}(x, y)$, kde $e = xy$
6. Vratíme T

korektnost

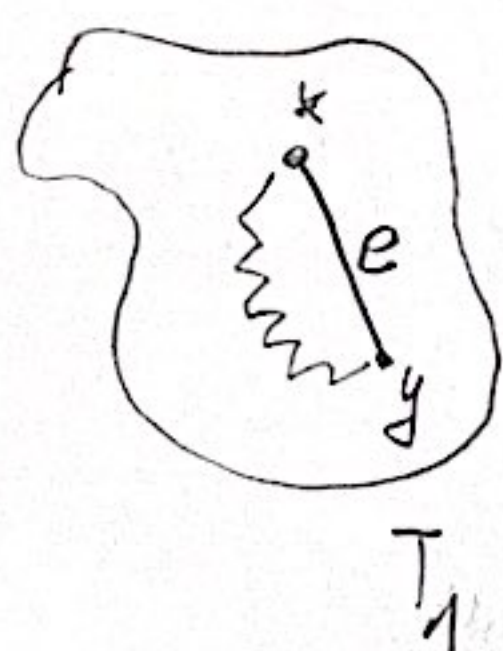
Lemma: K.a. vrátí minimální kostru (z řezového lemma)

1) přidáme e



podle řezového lemma, když oba konce e leží v různých komponentách, ale e je nejlehčí mezi T_1 a $G \setminus T_1$, pak e je v min. kostře

2) nepřidáme e



vrcholy x a y už leží ve stejné komponentě, přidáním hrany e vznikne cyklus, ale min. kostra je strom, tedy e neleží v min. kostře

složitost

$O(m \cdot \log n)$ - z rychlejšího Union-Find problému

Union Find

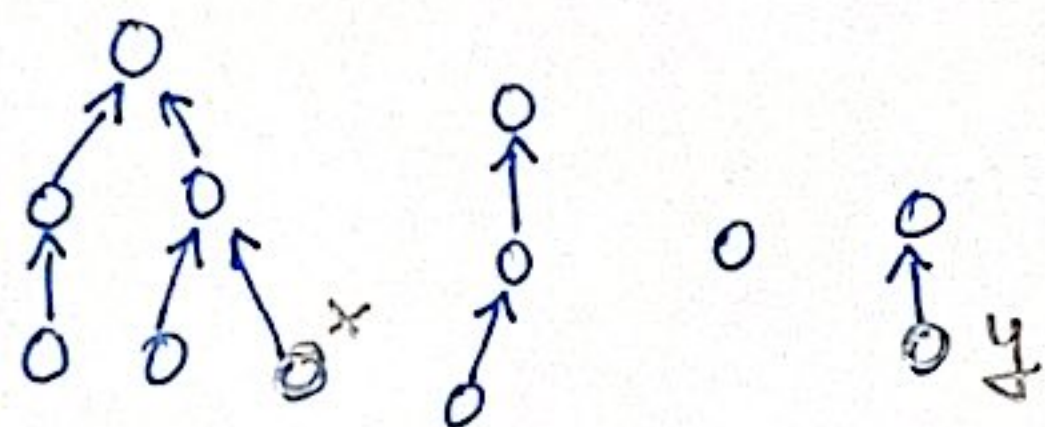
↳ datová struktura ke Kruskalovu algoritmu

Find (x, y): Jsou ve stejné komponentě?

Union (x, y): Přidá hranu x, y .

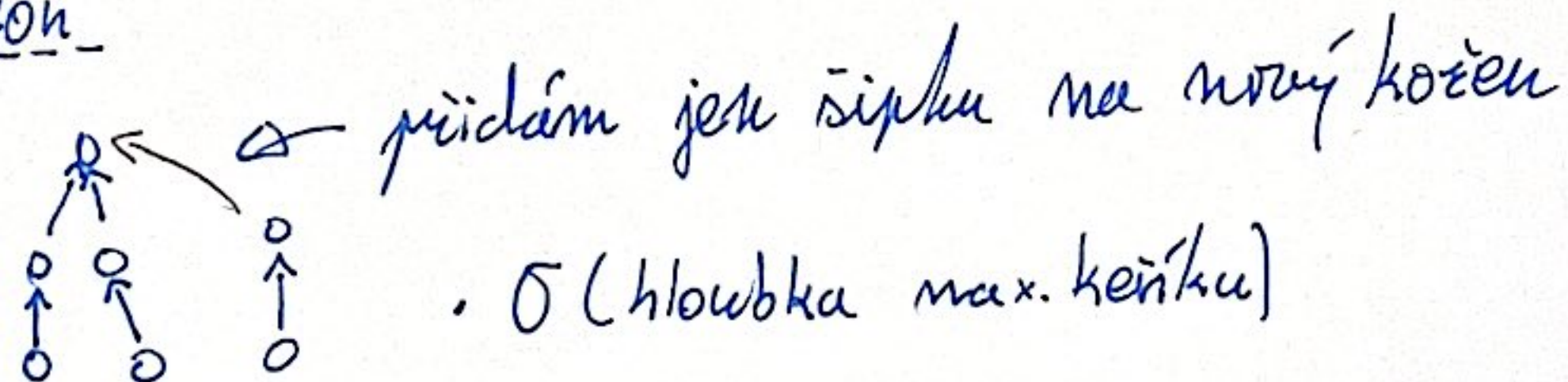
Find

- O (hloubka max. keřku)
- budu mít keřky udávající historii operací a vrchol si pamatují rodiče



- půjdu od x i od y do kořene a pokud je kořen stejný, pak x, y jsou ve stejné komponentě, jinak ne

Union



• O (hloubka max. keřku)

rychlení - kořeny si pamatují hloubku

$h(x)$ = hloubka keřku s kořenem x

- připojím menší pod hlubší, pak vzroste hloubka o max. 1

☹ Keřek hloubky h má alespoň 2^h vrcholů

↳ protože připojením menšího se hloubka vzroste o max. 1, ale počet vrcholů roste

- $h \leq \log n$ $\Rightarrow$ Find : $O(\log n)$
Union : $O(\log n)$

pomalejší:
vrchol si
pamatuje id
komponenty

Find : $O(1)$

Union : $O(n)$

Datové struktury

• abstrakce uložení dat

↳ statické = vytvoření, dotazy zda tam je atd

↳ dynamické = úprava prvků, přidání, odebrání

• rozhraní

↳ jaký máme přístup k datům, funkce

• implementace

↳ jak jsou datové struktury implementovány uvnitř

↳ pole, spojový seznam, bin. (k-regulární) halda,

• pole = obecně posloupnost

• fronta, zásobník

Fronta

	spojový seznam	pole - hloupě	pole - cyklicky
Enqueue	$O(1)$	$O(1)$	$O(1)$
Dequeue	$O(1)$	$O(n)$	$O(1)$

Množina

U = universum prvků

$X \subseteq U$ = podmnožina universa, tedy množina

$|X| := n$ = velikost množiny

předpoklad, že s prvky lze pracovat v $O(1)$

	seznam	pole	seřazené pole	vyhledávací strom	heš. tabulka
Find (Member)	$O(n)$	$O(n)$	$O(\log n)$	$O(\log n)$	$O(1)$
Insert	$O(n)$	$O(n)$	$O(n)$	$O(\log n)$	$O(1)$
Delete	$O(n)$	$O(n)$	$O(n)$	$O(\log n)$	$O(1)$
Build			$O(n \log n)$		$O(n)$

proto má pole $O(n)$ u insert a delete

pokud $a \in X$, tak nic

pokud $a \notin X$, tak nic

nemí pracovat o uspořádání

$O(n)$ u ins a del kvůli časování prvků

heš. tabulka
 $O(1)$
 $O(n)$
 $O(1)$
 $O(n)$
 $O(1)$
průměr

Uspořádaná množina

↳ na universu musí existovat lin. uspořádání $(u, \leq)$

↳ implementace pomocí vyhledávacího stromu nebo seřazeného pole

↳ operace navíc : $\left. \begin{array}{l} \bullet \text{Min/Max} \\ \bullet \text{Pred}(x)/\text{Succ}(x) \end{array} \right\} O(\log n)$ - díky vyhledávacím stromům

Glomik

• zjednodušení množiny

• množina dvojic $\{(k, h)\}$, kde $h \in U_h$ - univerza hodnot
 $k \in U_k$ - univerza klíčů

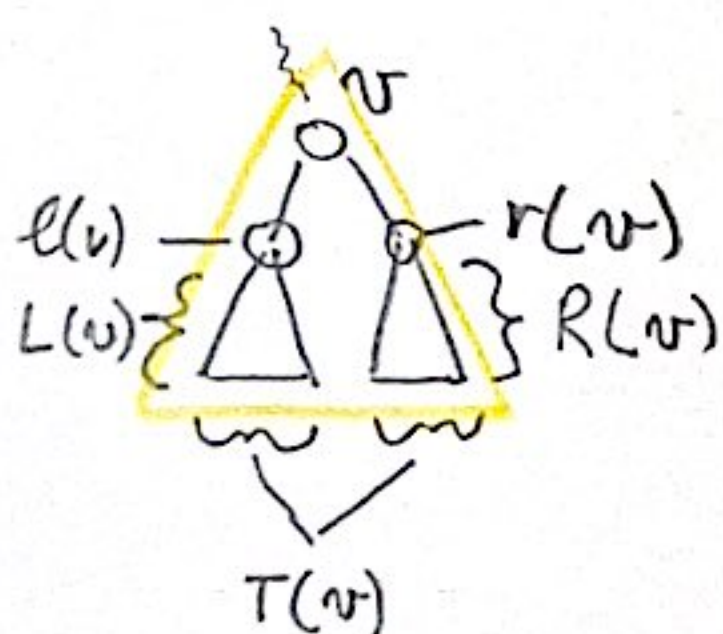
• Find (Member) (k) → vrátí podle klíče

• Insert (k, h) → přidá klíč a hodnotu

• Delete (k) → odstraní hodnotu u klíče

Binární strom

↳ strom, co má pevný kořen a rozlišuje levého a pravého syna



$T(v) :=$ podstrom v

$r(v) :=$ pravý vrchol

$l(v) :=$ levý vrchol

$\circ \Rightarrow h(v) = 0$

$L(v) := T(l(v))$ levý podstrom

$\circ \Rightarrow h(v) = 1$

$R(v) := T(r(v))$ pravý podstrom

$h(v) :=$ hloubka $T(v)$ v hranách

$h(\emptyset) := -1$ (hloubka prázdného podstromu)

list je definován tak, že nemá syny (jinak v Diskretech, že má stupeň 1)

jsou různé
binární stromy

Binární vyhledávací strom

na množině musí existovat lineární uspořádání
($U, \leq$). Triviálně splněno pro počítač.

definice BVS

Všechny uzly mají unikátní klíč $k(v) \in U$

$$\forall v: \forall a \in L(v): k(a) < k(v)$$

$$\forall v: \forall b \in R(v): k(b) > k(v)$$

inorder průchod

Show(v):

if $v = \emptyset$: return

Show($L(v)$)

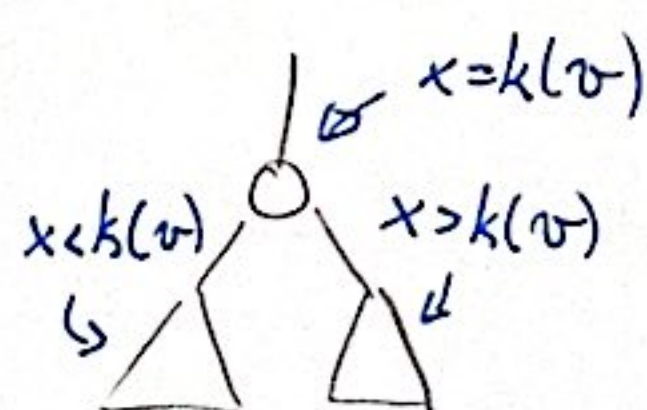
print($k(v)$)

Show($R(v)$)

vytiskne v po řadě, protože
první vytiskne všechny
menší, pak větší

Operace

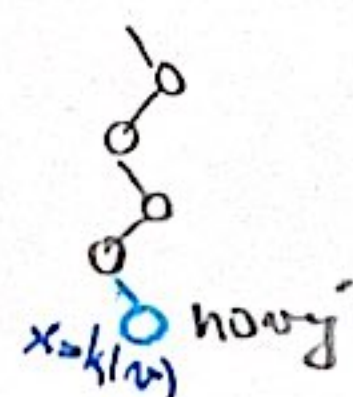
Member(x) / Find(x)



1) najdu x našel

2) najdu $\emptyset$ nenalel, vrchol není v stromě

Insert(x)



Find(x):

1) najdu x nic nedělat

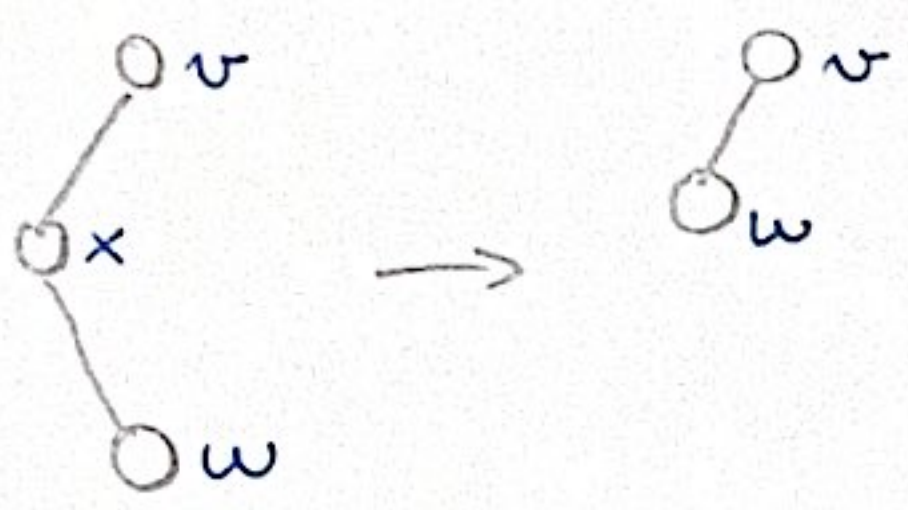
2) najdu $\emptyset$ přidám nový list s $x = k(v)$

• Delete (x)

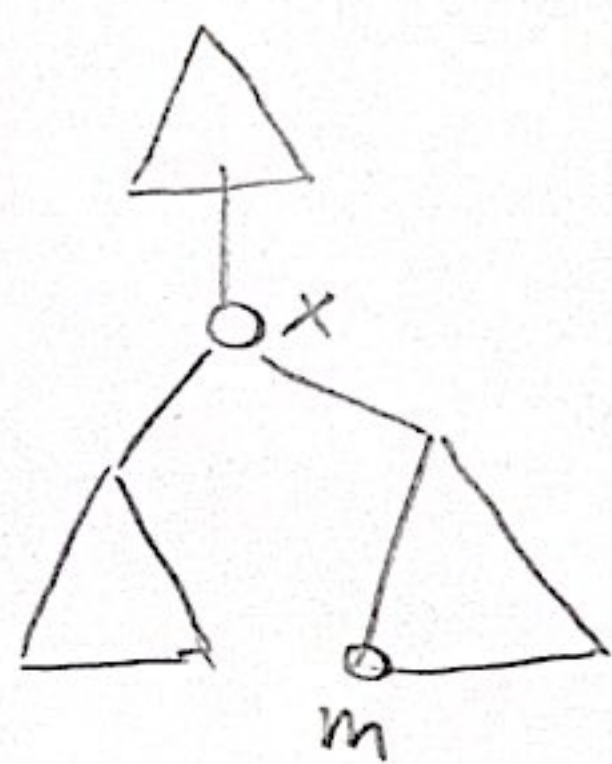
① x je list

 $\rightarrow$ smazem uzel po nalezení Find(x)

② x má 1 syna

 $\rightarrow$ smazem x a přepojím hranu do w (navěsím syna)

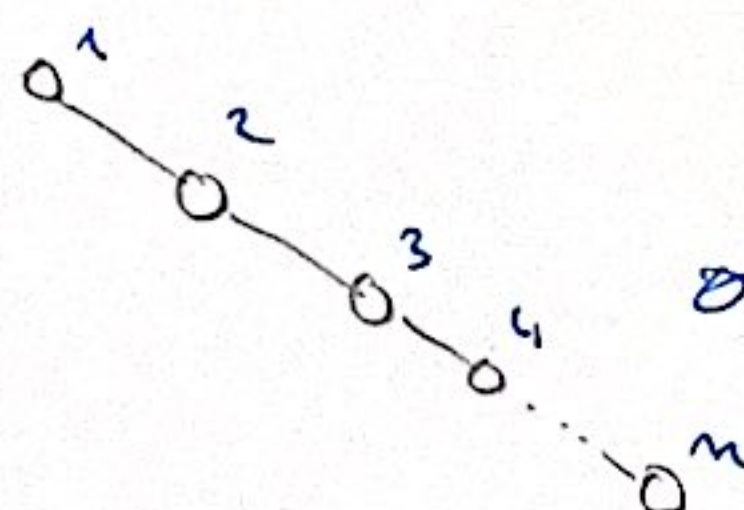
③ x má 2 syny



$\rightarrow$ najdu $m = \text{Min}(R(x))$ a $\rightarrow$ bude mít 0/1 synů a to lze vyřešit
na místo x. Tedy minimum z
praveho podstromu.

• díky BFS lze realizovat všechny operace s
množinami i uspořádanými množinami

• složitost všech operací je $O(h)$ (hloubka stromu)

 $\rightarrow$ degenerovaný strom, složitost stejná
jak se seznamem

• vyvážené stromy mají $h = \log(n)$

$\rightarrow$ pak mají všechny operace složitost $O(\log n)$

lze i maximum
z levého
podstromu

implementace:
nepřepisovat, ale
přepojit
kudli množství dat

kromě Show(v)

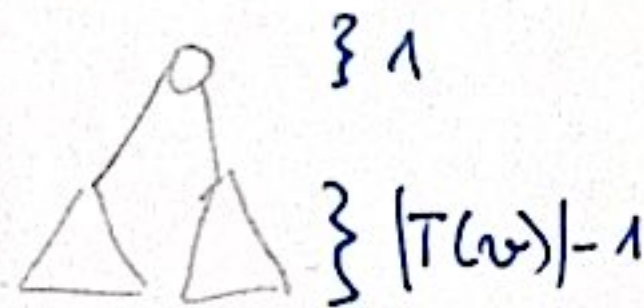
$\uparrow$
sen má
 $O(n)$ o
vel. množin

Dokonale vyvážené binární stromy

$$\forall v: ||L(v)| - |R(v)|| \leq 1$$

Lemma: Dokonale vyvážený strom s n vrcholy má
hloubku $h \leq \log n$

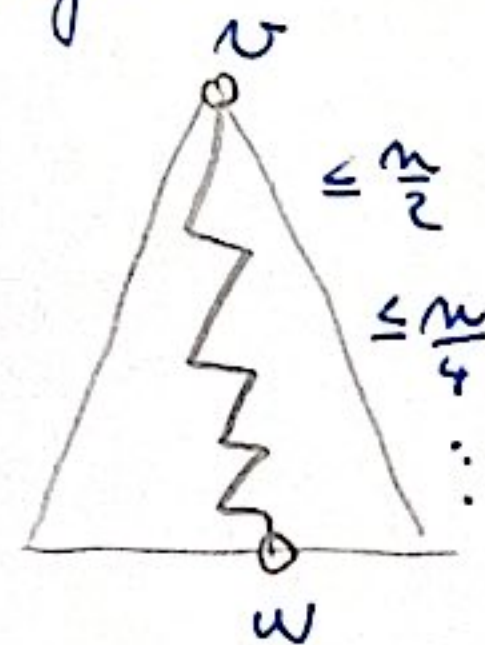
Důkaz:



← protože se oba podstromy liší
o max. 1 vrchol, tak jsou podobné:

- stejné $\rightarrow$ ten 1 vrchol je kořen
- různé $\rightarrow$ velký počet vrcholů $|T(v)| = n$

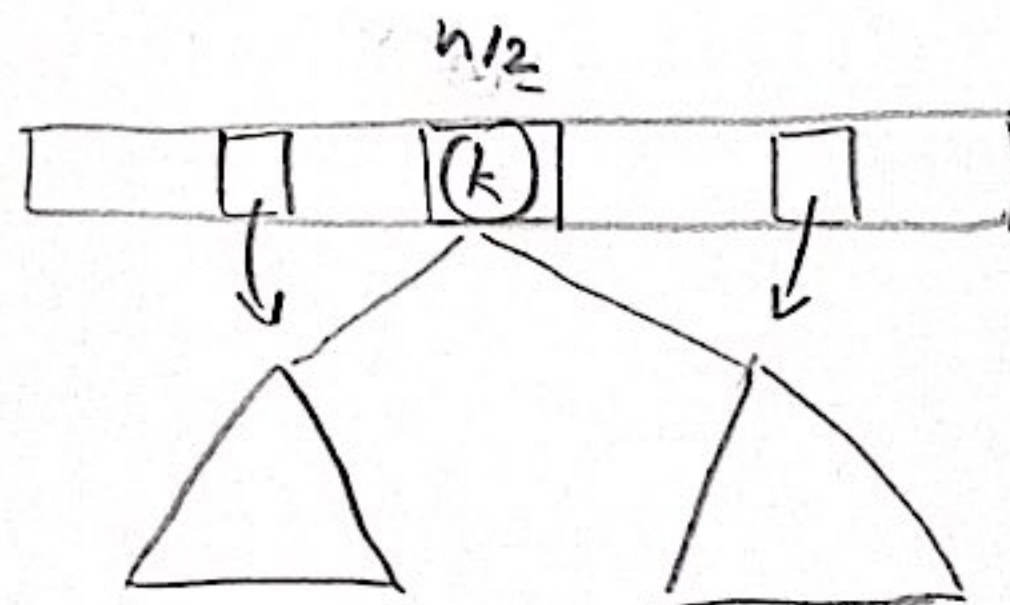
$\hookrightarrow$ platí pro každý vrchol, tedy najdu cestu od kořene k
nejhlubšímu listu



$\hookrightarrow$ bude logaritmicky zmenšovat, tedy
cesta $P(vw) = h \leq \log_2 n$

Konstrukce dokonale vyváženého BVS z seřazené posloupnosti

klíčů $x_1 < x_2 < \dots < x_n$.



$\hookrightarrow$ D.V. BVS vždy existuje a
lze vytvořit v $\Theta(n)$

Build(i, j):

1. střed $\rightarrow k$
2. $L(k) \leftarrow \text{Build}(i, k-1)$
3. $R(k) \leftarrow \text{Build}(k+1, j)$
4. return k

$\hookrightarrow$ Insert a Delete nejsou lépe než lineárně v D.V. BVS,
tedy dobrá statická, ale špatná dynamická datová struktura

D.V. BVS pro každý
počet prvků existuje

Věta: V každé implementaci D.V. BVS má Insert nebo Delete složitost $\Omega(n)$ pro nekonečně mnoho n .

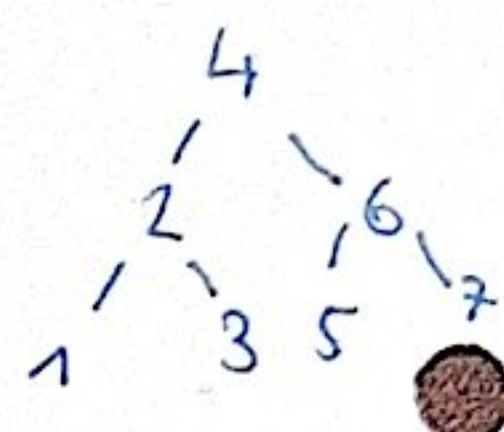
2 předchozí strany

Důkaz:

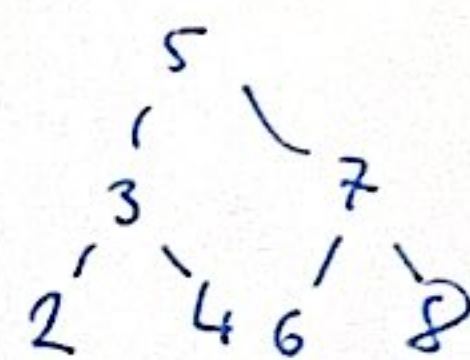
Pro $n = 2^k - 1$ je strom jednoznačně určen. Tedy podstromy jsou velké $\frac{n-1}{2} = \frac{2^k-2}{2} = 2^{k-1} - 1$. Kde klíče jsou $1 \dots n$

Pro operace Insert a Delete se n vzhledem změnilo, zla jsou listy. Změnilo se $\Omega(n)$ ukazatelů. Tedy časová složitost $\Omega(n)$.

strom $2^3 - 1$



Insert(8)
Delete(1)



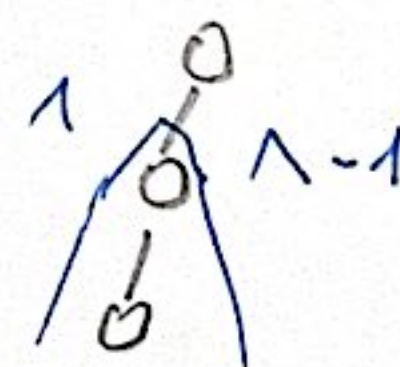
AVL stromy

Definice: Hlobkově vyvážený strom $\equiv \forall v : \underline{|h(l(v)) - h(r(v))| \leq 1}$

AVL $\rightarrow$ vyhledávací hlobkově vyvážený strom

Příklady:

není



Věta: AVL strom s n vrcholy má hloubku $\Theta(\log n)$

Důkaz:

$A_h :=$ minimální počet vrcholů pro hloubku h v AVL stromu
celého stromu

platí rekurence z definice AVL stromu:

$$\underline{A_h = 1 + A_{h-1} + A_{h-2}} \quad \text{nebo-li} \quad A_h \geq A_{h-1} + A_{h-2}$$

velikost roste alespoň exponenciálně $n \geq A_h \geq (\sqrt{2})^h$

Tvrzení: $n \geq A_h \geq (\sqrt{2})^h$

Důkaz: (Indukcí podle h)

$$h-1 \rightarrow h$$

$$A_h = 1 + A_{h-1} + A_{h-2} > \sqrt{2}^{h-1} + \sqrt{2}^{h-2} = \sqrt{2}^h \left(\frac{1}{\sqrt{2}} + \frac{1}{2} \right) > \sqrt{2}^h$$

Důsledek:

roste exponenciálně pro libovolné c $A_h \geq c^h$; $c = \sqrt{2}$

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

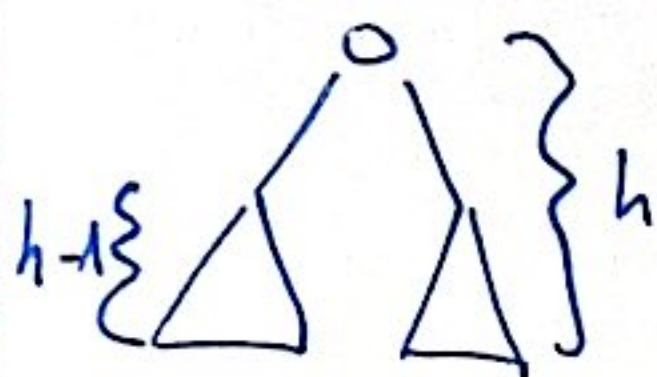
$B_k :=$ maximální počet vrcholů pro hloubku h

$$\left. \begin{array}{l} B_0 = 1 \\ B_1 = 3 \end{array} \right\} B_k = 2 \cdot B_{k-1} + 1 = 2^{k+1} - 1$$

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

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

$$\begin{array}{l} A_0 = 1 \\ A_1 = 2 \\ \vdots \end{array}$$



celý strom $\rightarrow h$
jeden podstrom $\rightarrow h-1$
druhý, může být
menší, ale
jen jeden podstrom $\rightarrow h-2$

dolní odhad
pomocí vzorce
pro dokonale vyv.
stromy

Vyvažování AVL stromu - Insert a Delete

musi' platit definice AVL stromu

$$\text{AVL strom} \Leftrightarrow \forall v: |h(l(v)) - h(r(v))| \leq 1$$

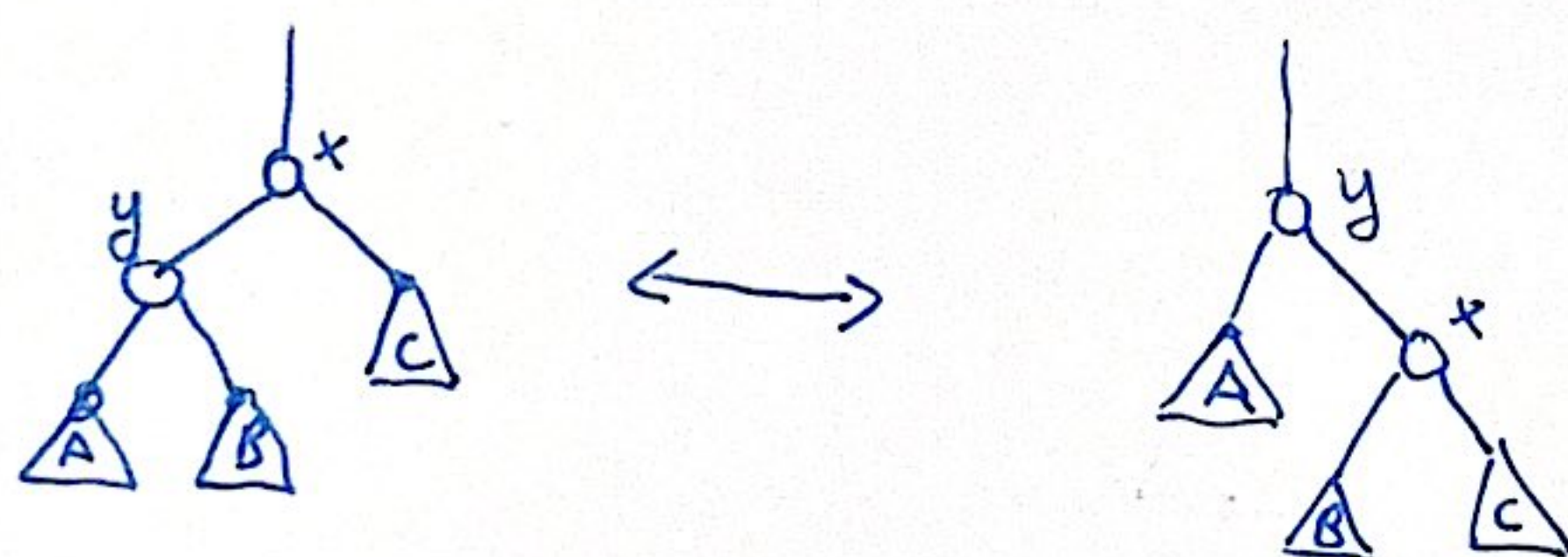
↳ saručuje hloubku $\Theta(\log n)$

Definice: znaménko je $S(v) := h(r(v)) - h(l(v))$

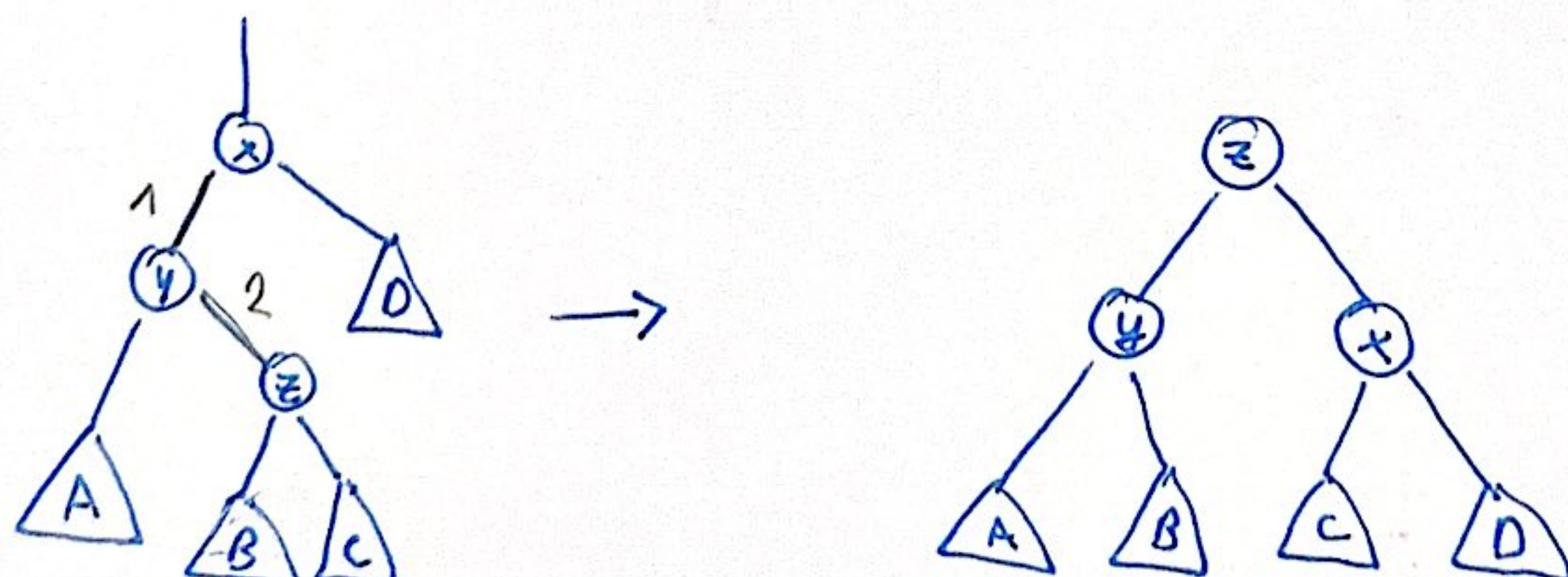
$$S(v) = \begin{cases} 0 & \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \triangle \quad \triangle \end{array} \rightarrow 0 \\ +1 & \begin{array}{c} \bullet \\ \diagup \\ \triangle \quad \triangle \end{array} \rightarrow + \\ -1 & \begin{array}{c} \bullet \\ \diagdown \\ \triangle \quad \triangle \end{array} \rightarrow - \end{cases}$$

Operace

1) Rotace hrany xy



2) Dvojitá rotace



Zachová
uspořádání

lze provést
v konstantním
čase-žněna
konst. počtu kaze

Zavěšení za z

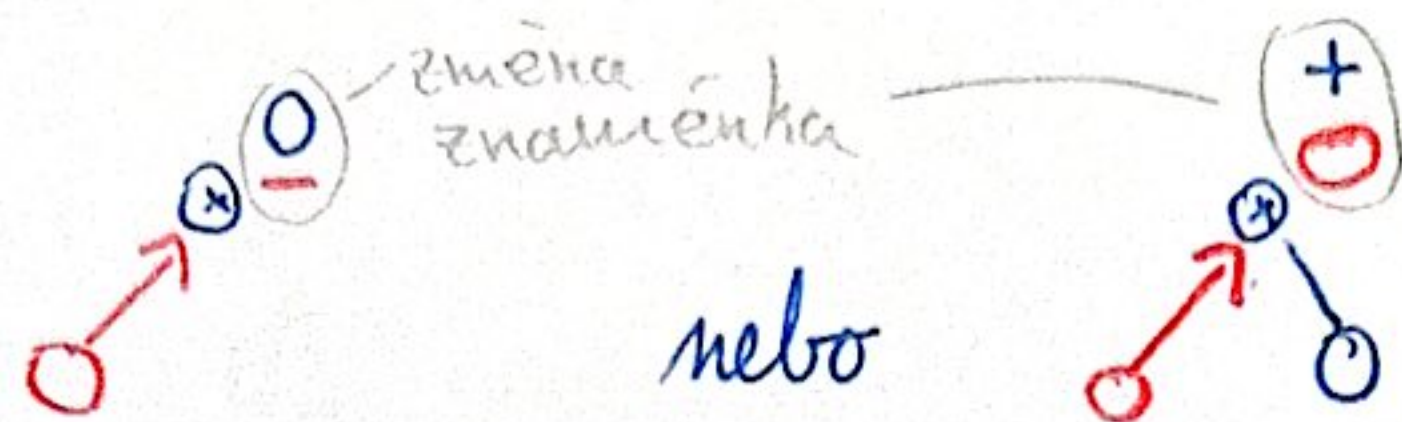
dá se složit z
rotace hran 2,
pak 1
doprava

nové •

Insert → tedy Find(x) a pak vyvážením

↳ přidám vrchol jako list namísto praveho podstromu, který zpropaguje nahoru, že se zvýšil o 1

Kdy se znaménko propaguje?
Pokud se zvýší hloubka



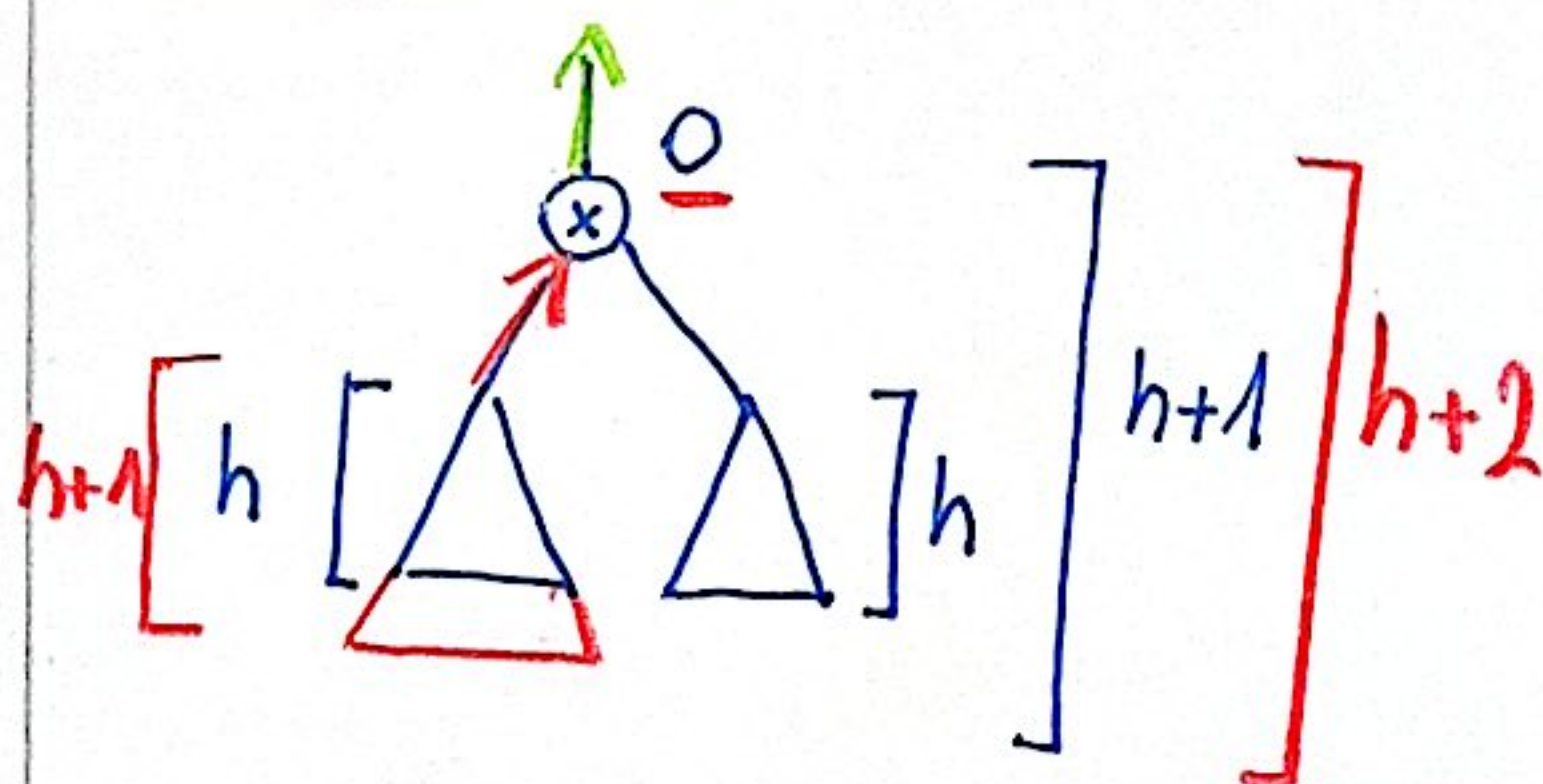
nebo

BÚNO přišla

zpráva zleva

Případy podle znaménka ve vrcholu $x \Leftrightarrow \delta(x)$

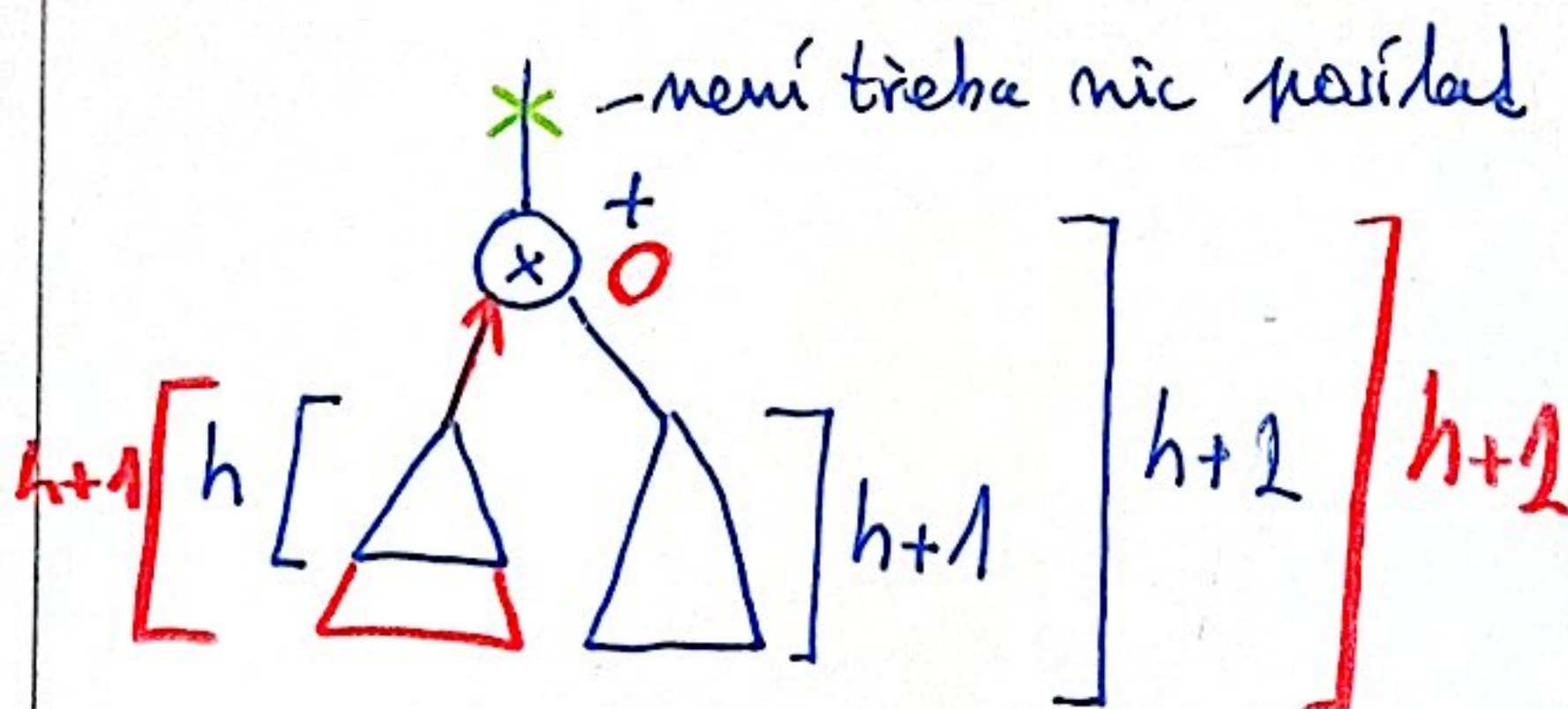
① $\delta(x) = 0$



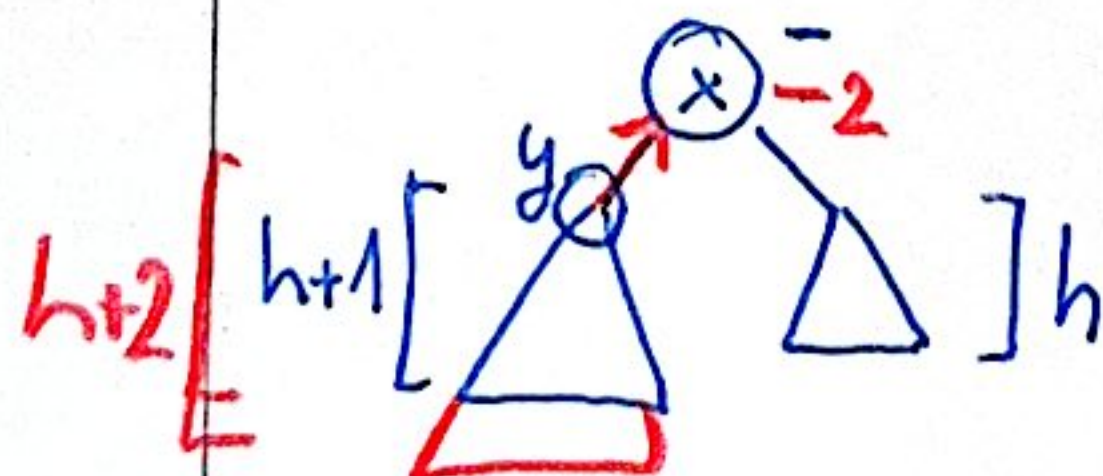
slučně:

- strom se prohloubil, posílám informaci nahoru
- ale není třeba nic přeorientoval, protože znaménko je validní

② $\delta(x) = +$

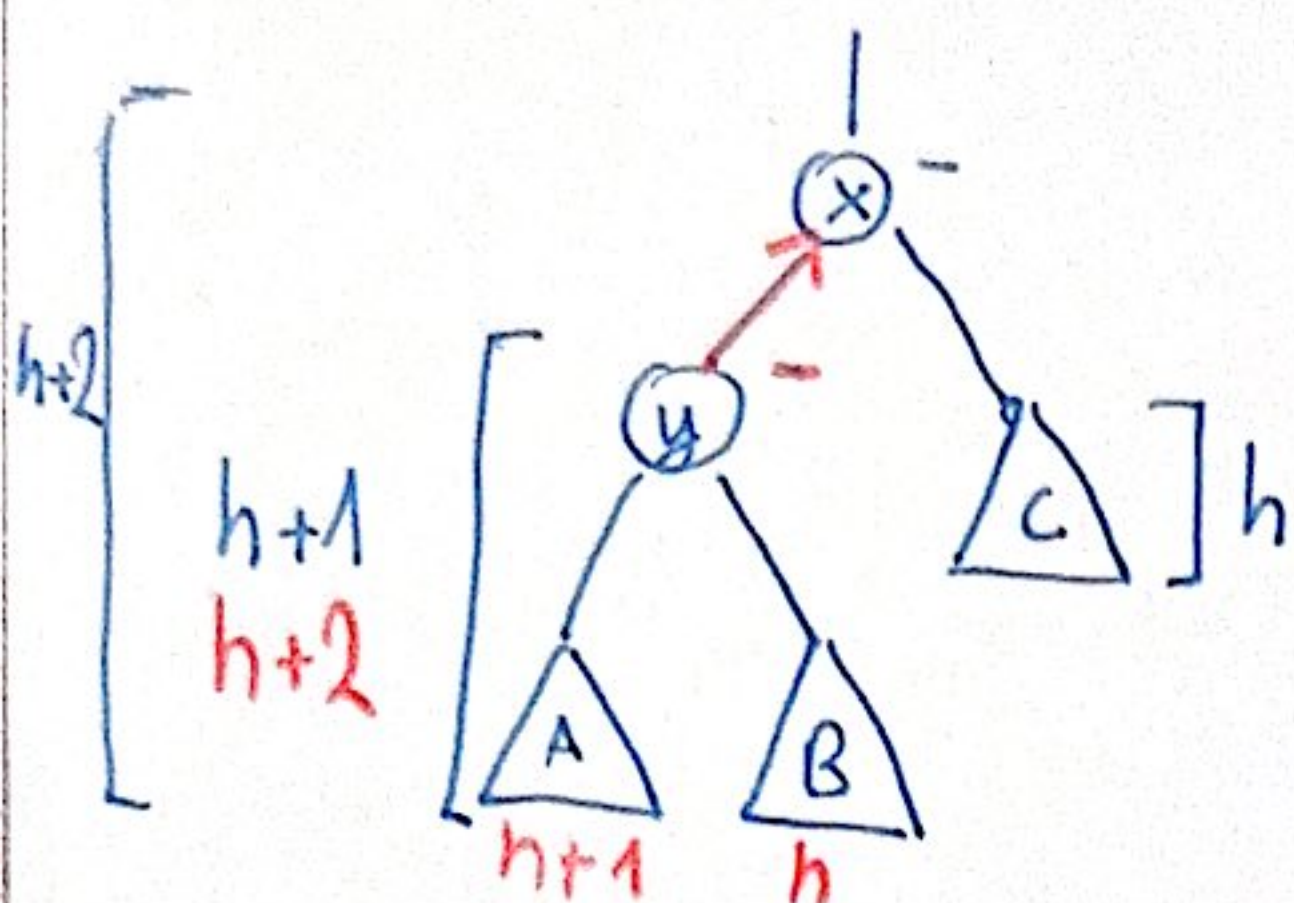
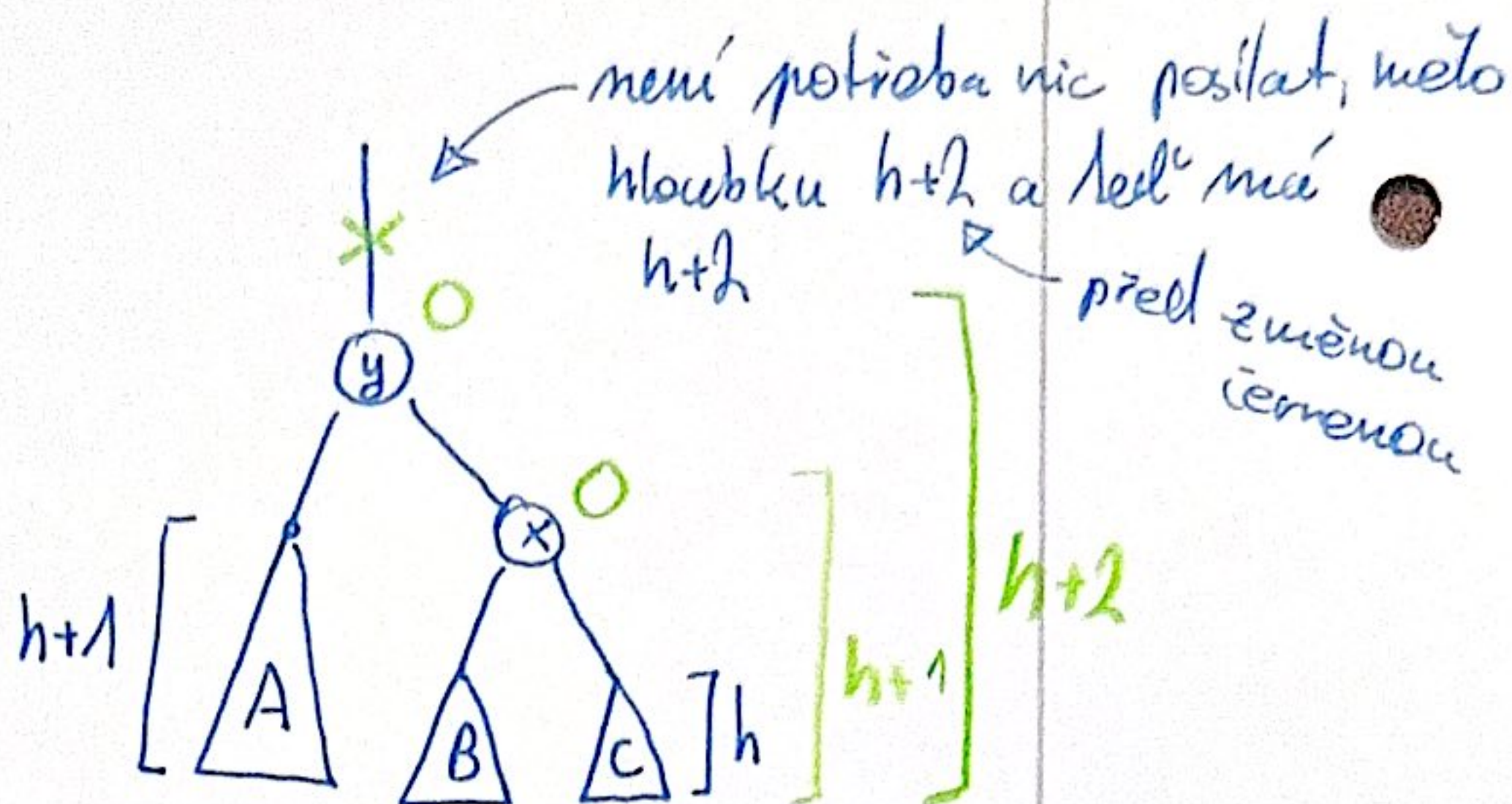


③ $\delta(x) = -$



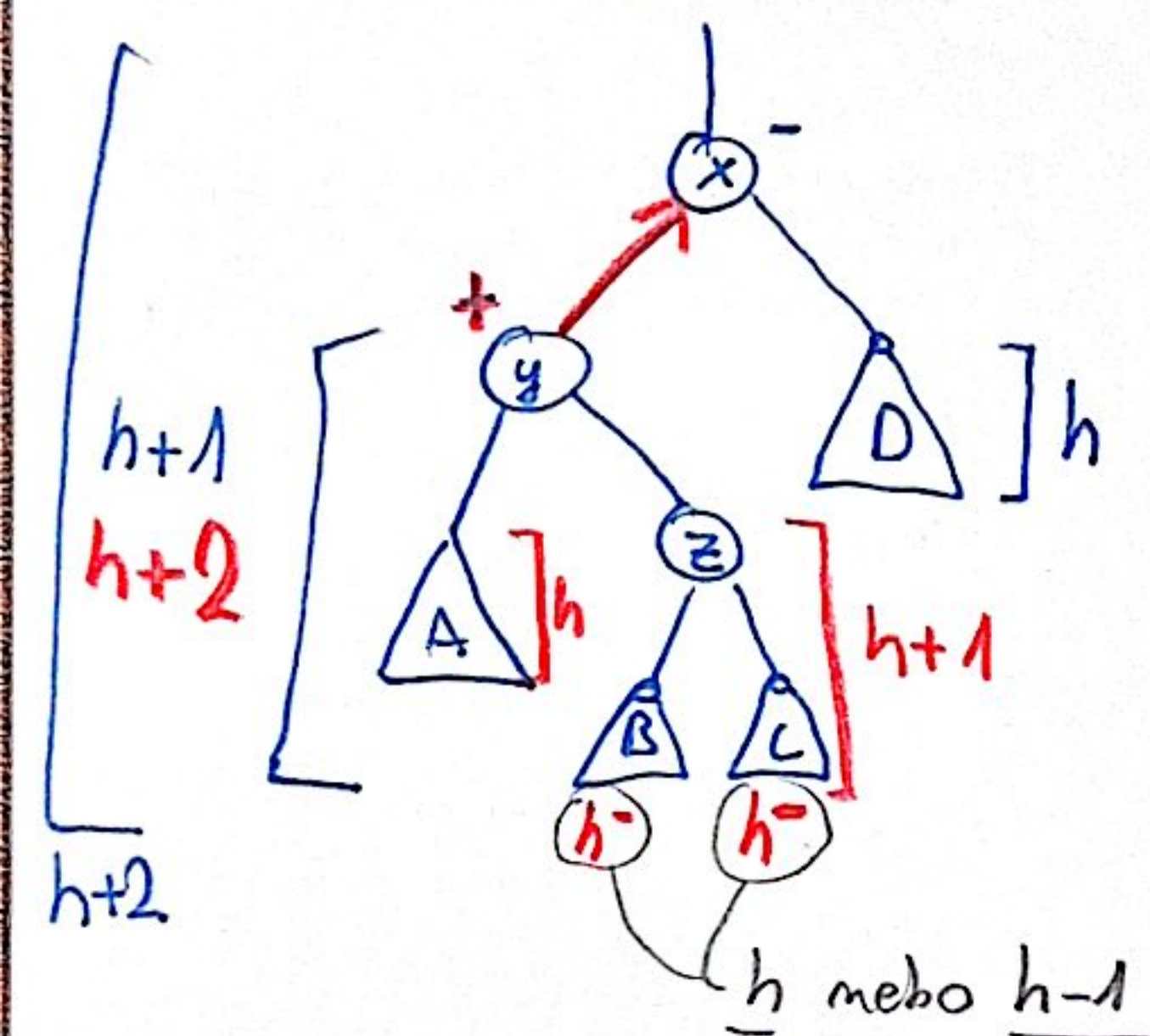
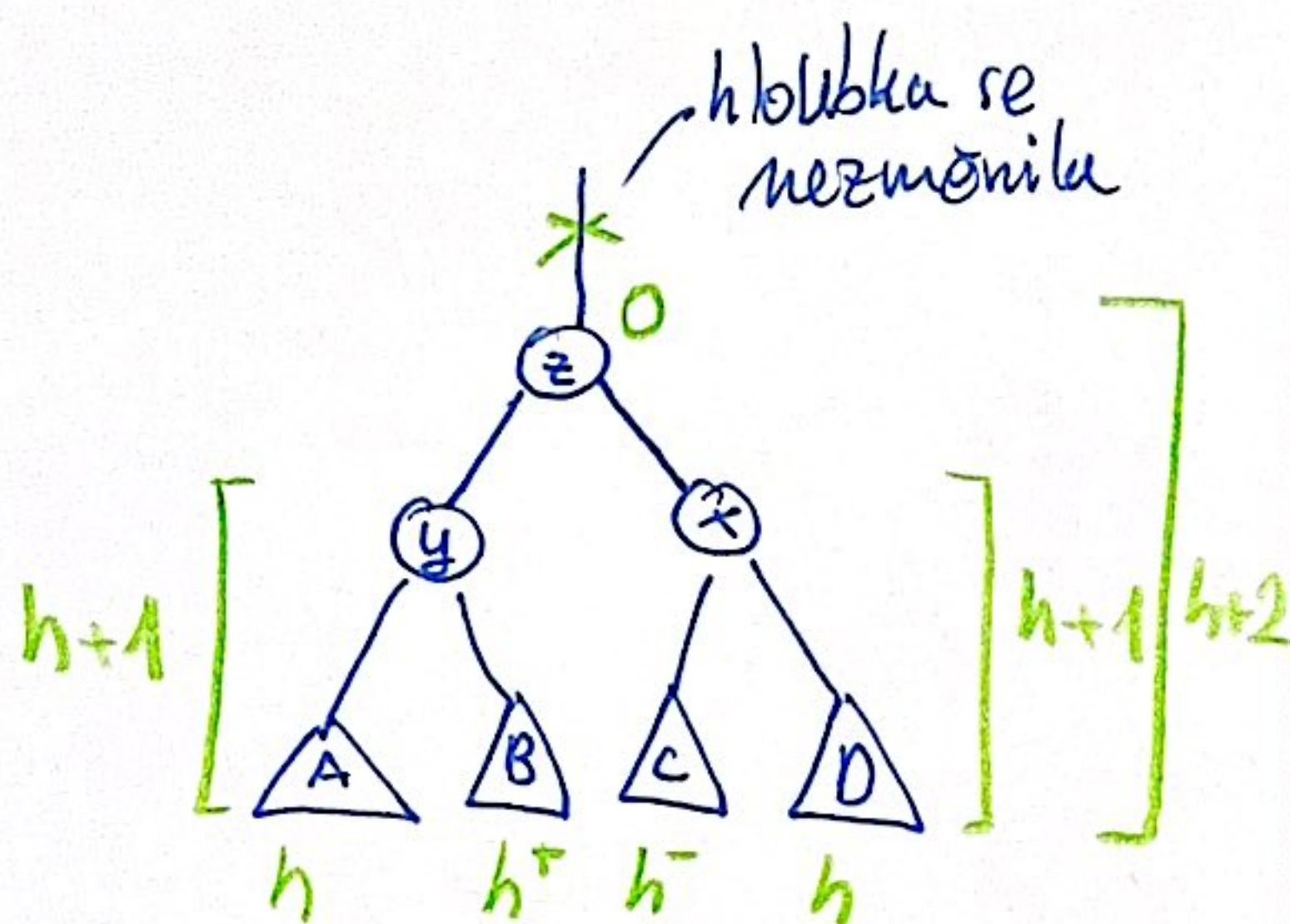
→ rozdělení případů na další strany

3a) $\delta(y) = -$



rotace
hrany xy

3b) $\delta(y) = +$



dvojitá
rotace

3c) $\delta(y) = 0$

→ pole v rodiči nastane 1/2

→ kromě počátečního případu listu, tak nikdy nenastane, protože z 0 se správa neposílá, nahoru se posílá jen z 1

Insert lze provést v čase $\Theta(\log n)$

• implementace rekursí

z existuje, protože vrchol s + musí mít pravé dítě

jedna rotace nestačí, protože levý syn x bude h+1 a y tedy bude mít levý h a pravý h+2

dvojitá rotace
→ rotace y z
doleva, pak
x z doprava

Jen tedy
pohyb zleva,
zredlování
obrážení
se znegují
všechna
znaménka

případy rozborů podobné

Delete

sim delete vždy končí

rozdělí
pošle zprávu,
že se snížila o 1

list

jedno
dítě

pošle na nové hraně, že se
hloubka snížila o 1

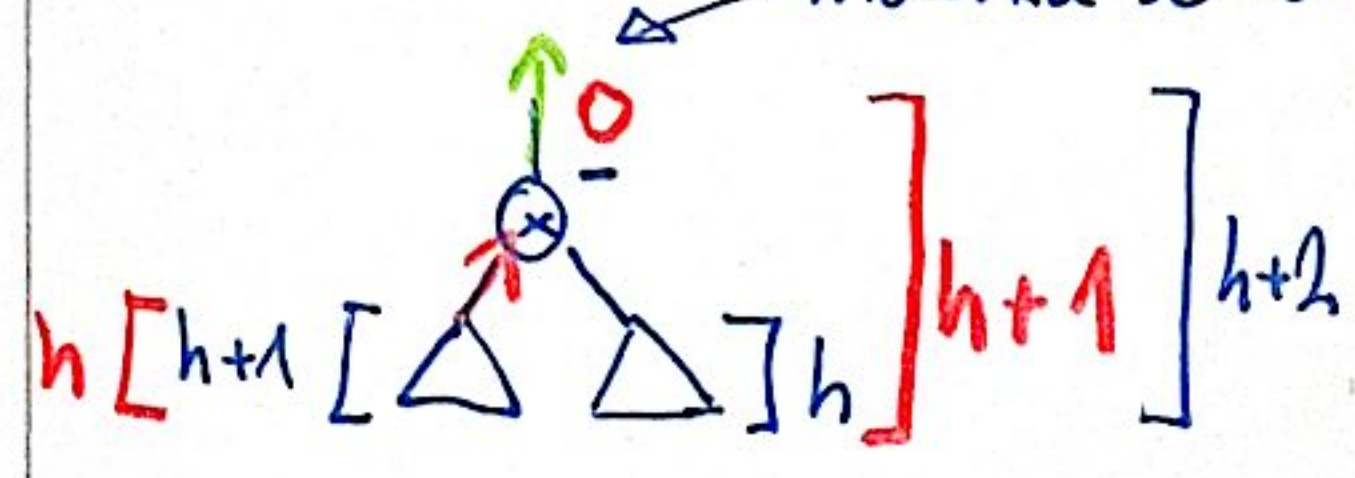
dvě
dětí } přivést na jeden z
předchozích případů

- obecně: do x přišla B.U.N.O. zleva zprava o snížení
hloubky podstromu o 1

přišla z
hlubšího

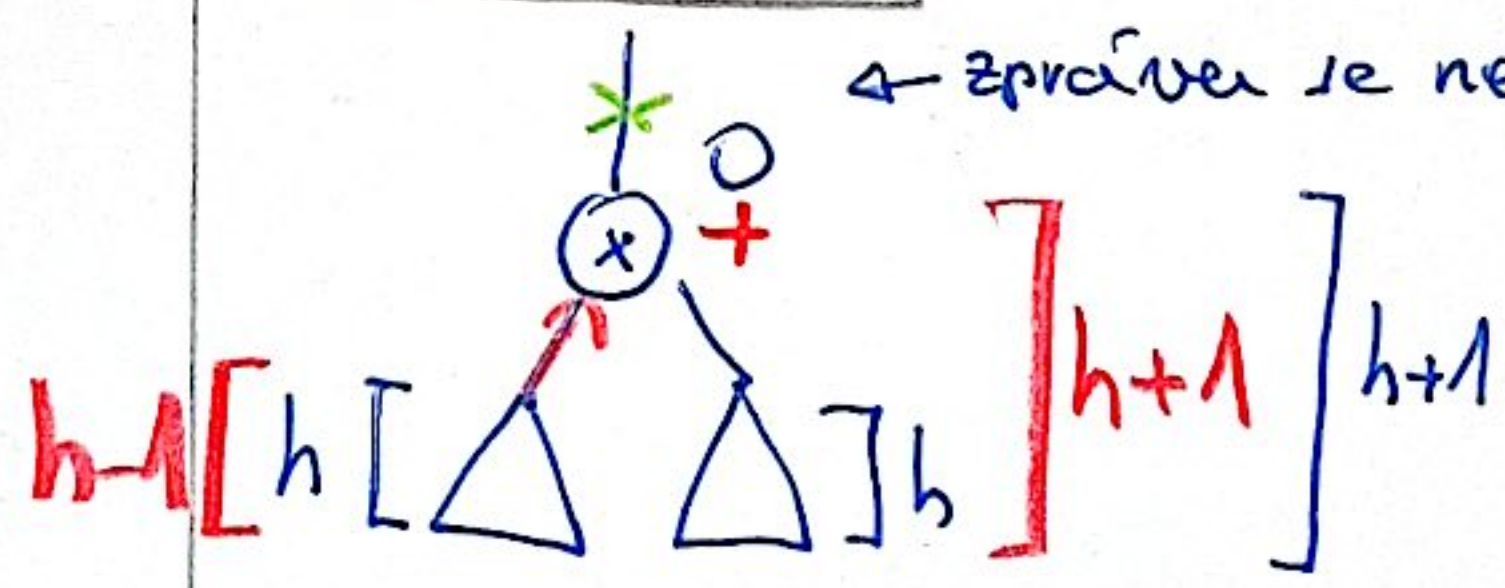
① $\delta(x) = -$

hloubka se zmenšila, tedy zpropagovat nahoru



② $\delta(x) = 0$

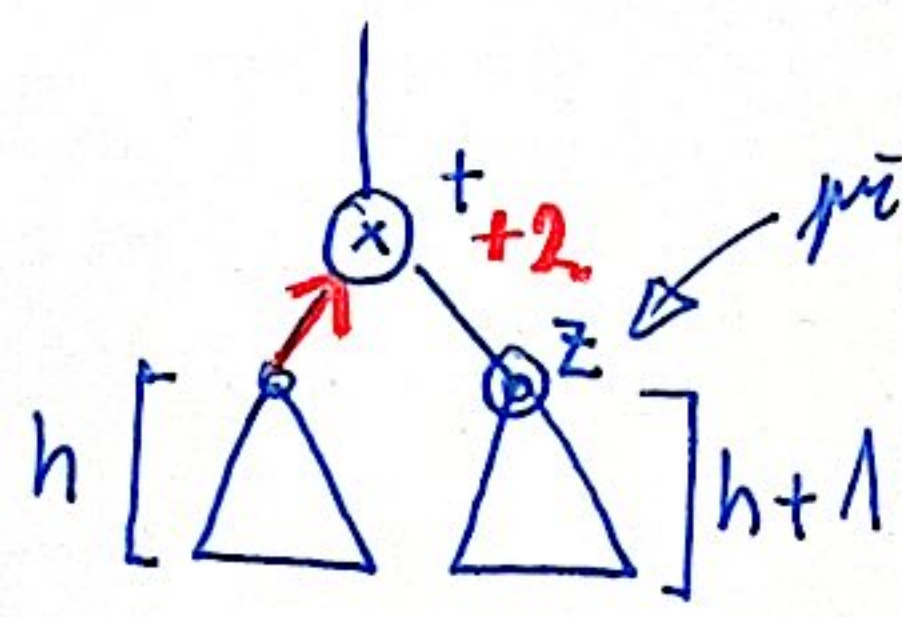
zpráva se neposílá



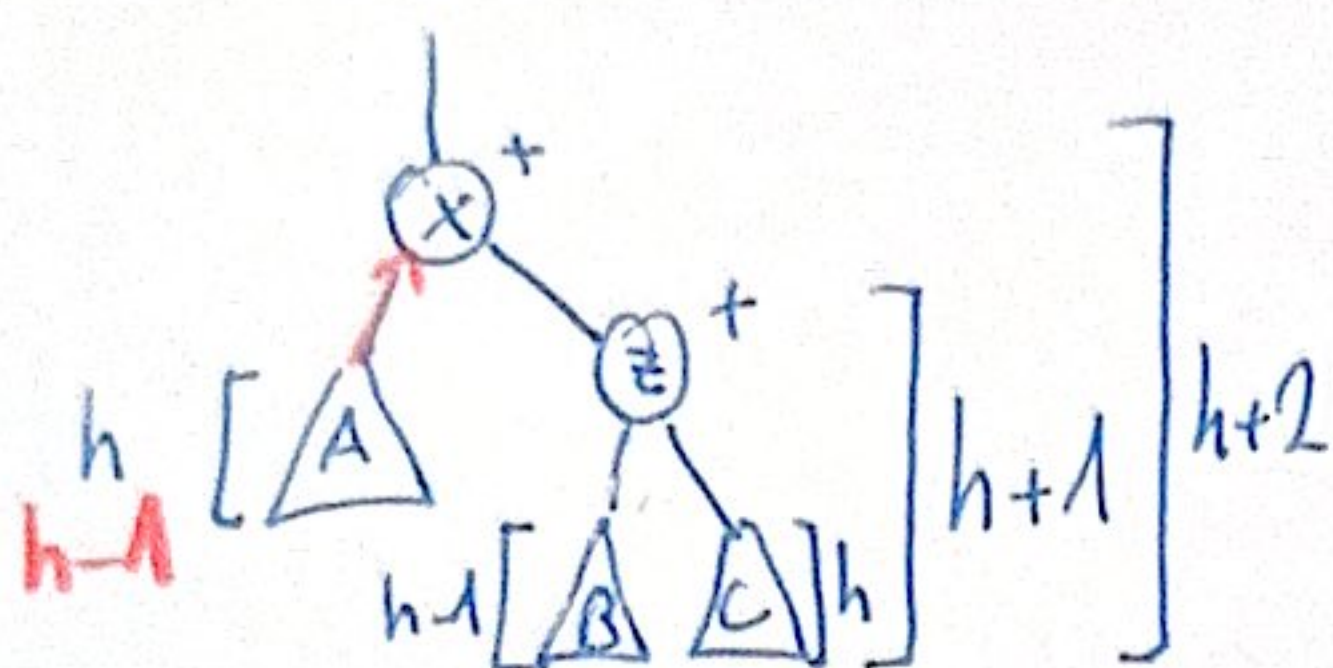
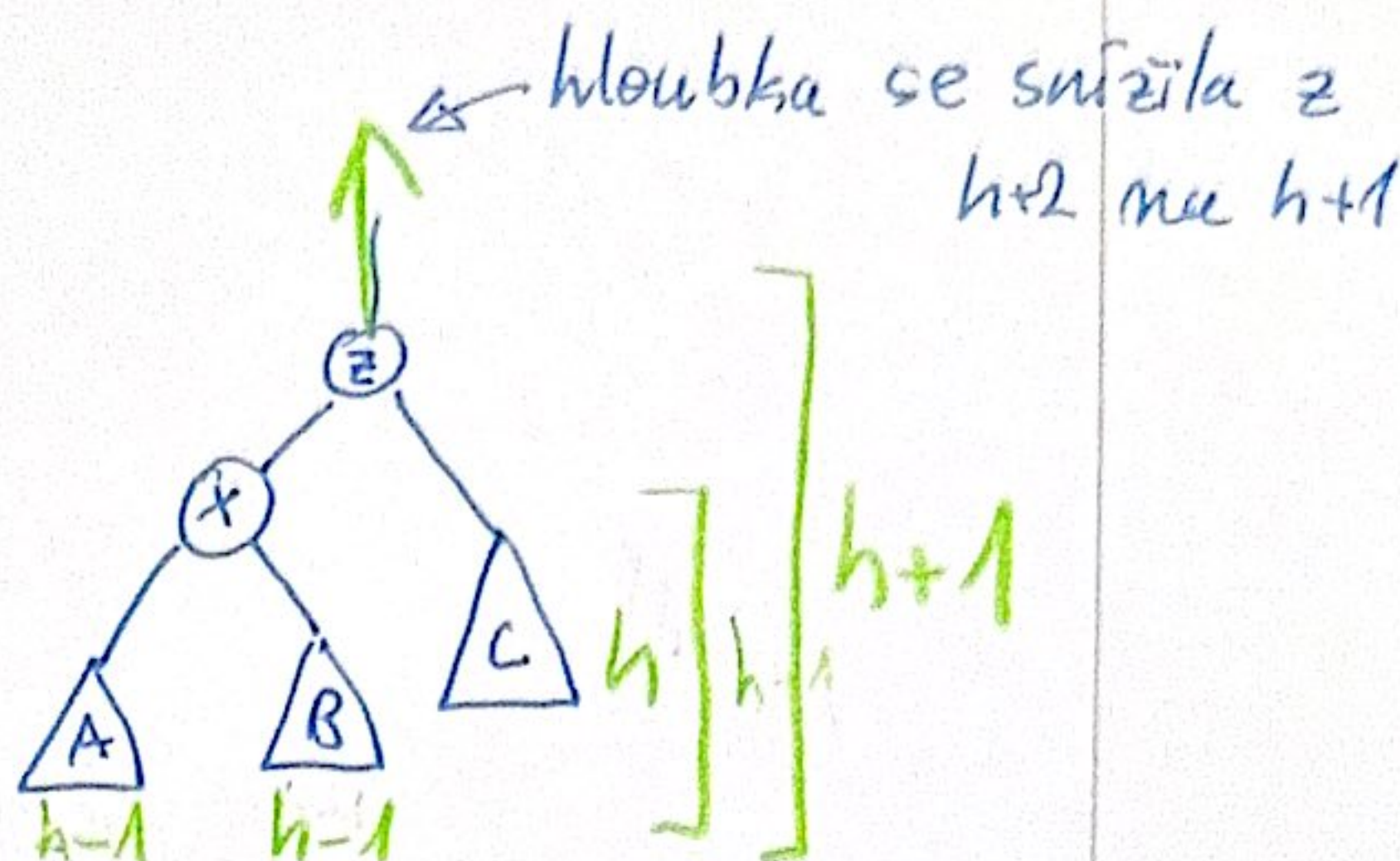
na levo bylo měčár
a ještě více se
zmečálo

③ $\delta(x) = +$

případy podle vrcholu z

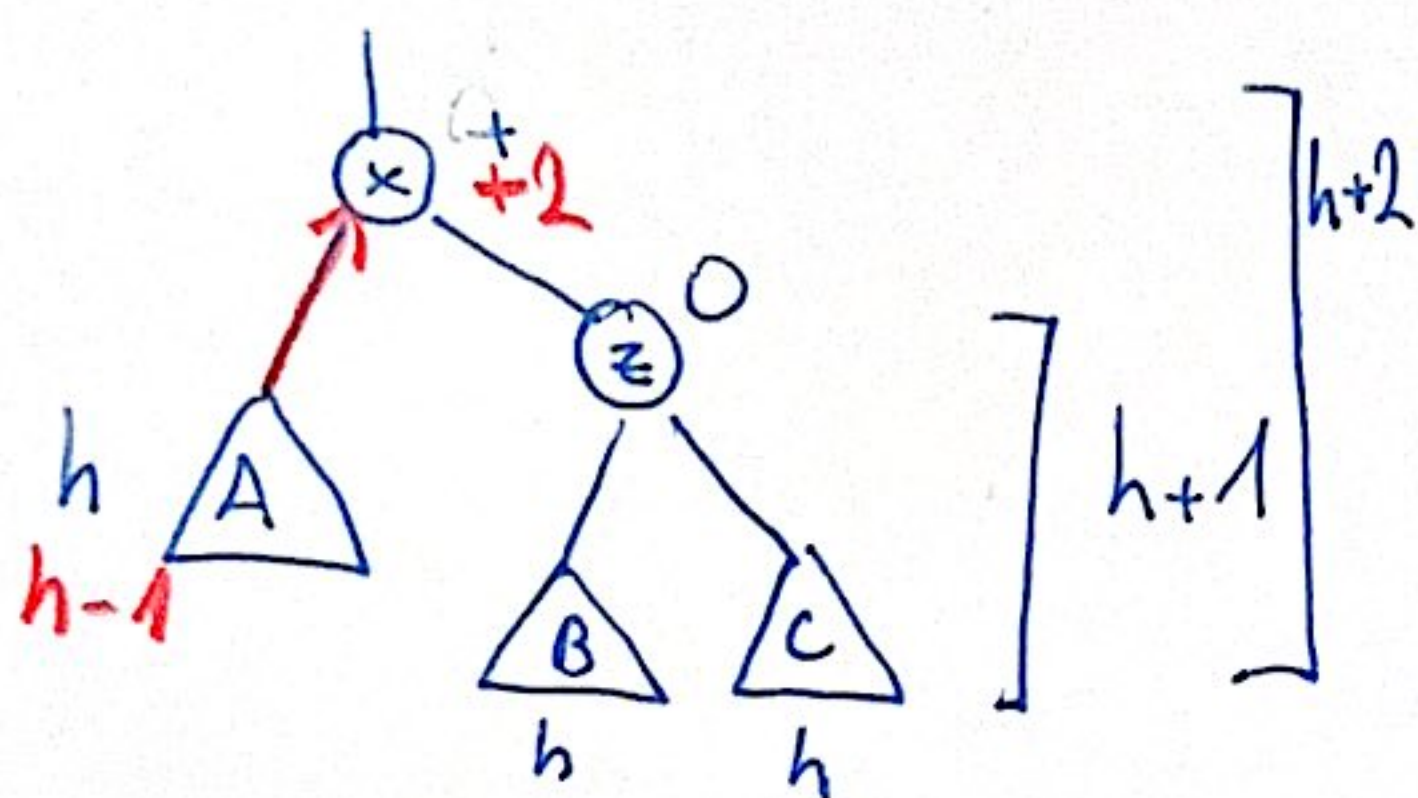
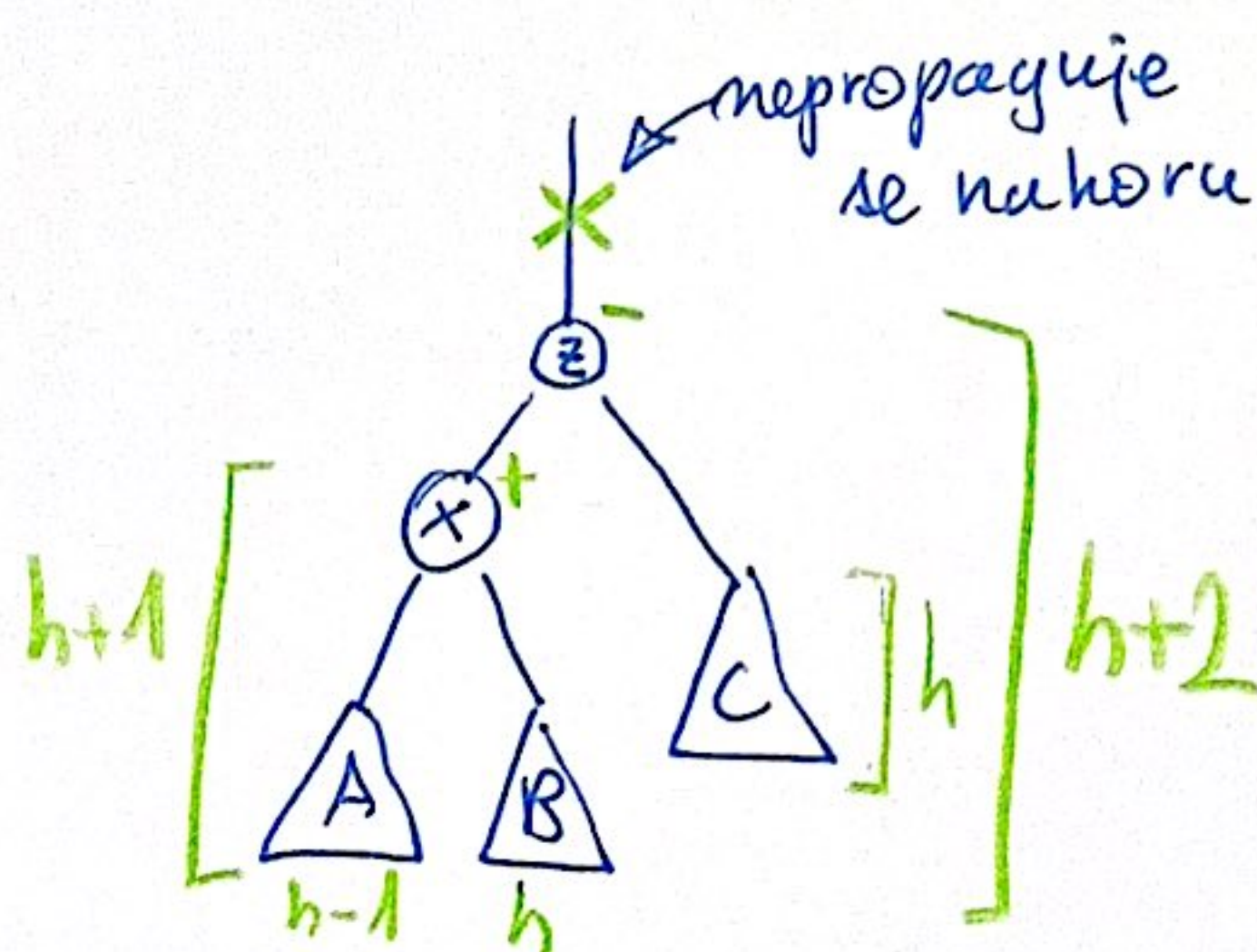


3a) $\delta(z) = +$



rotace xz

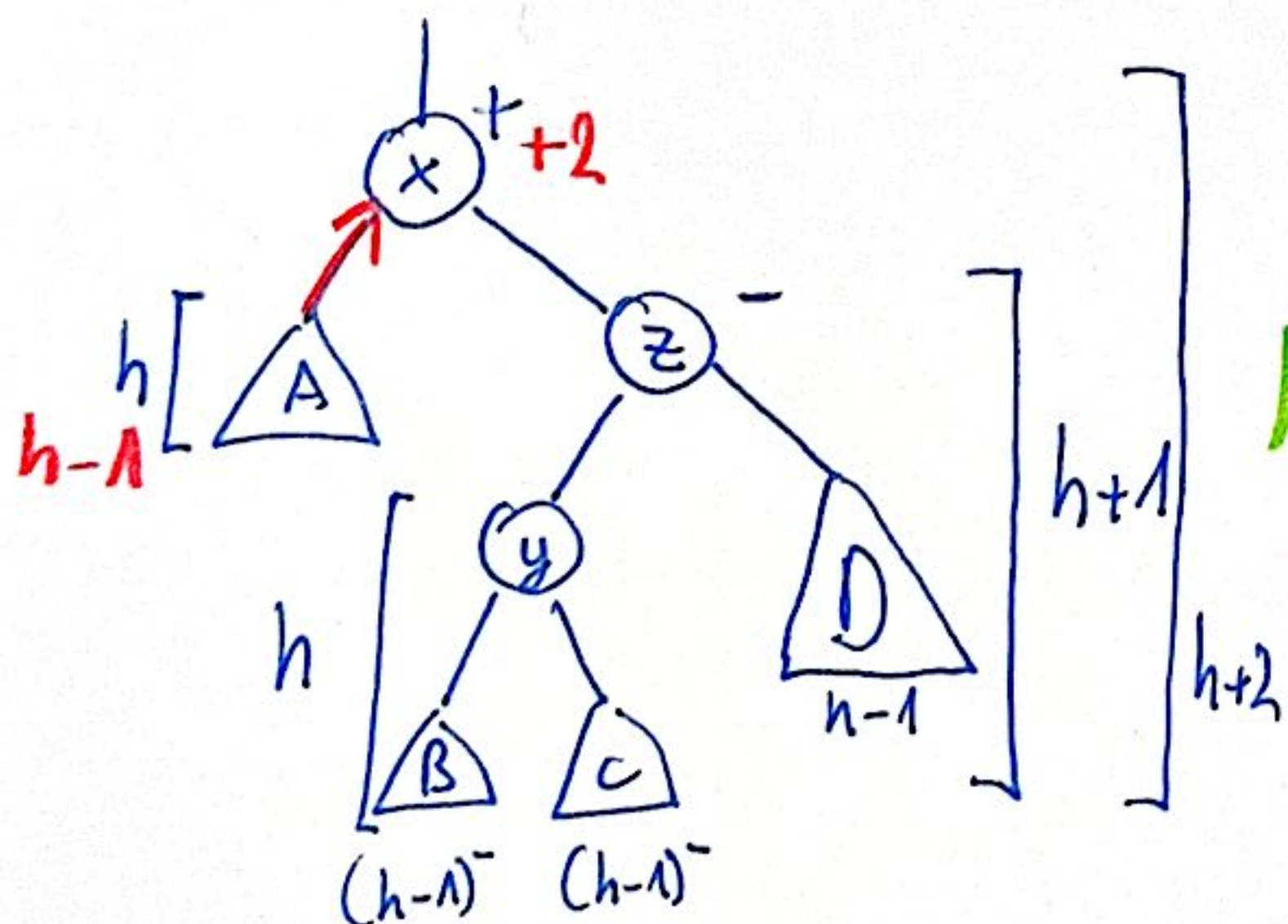
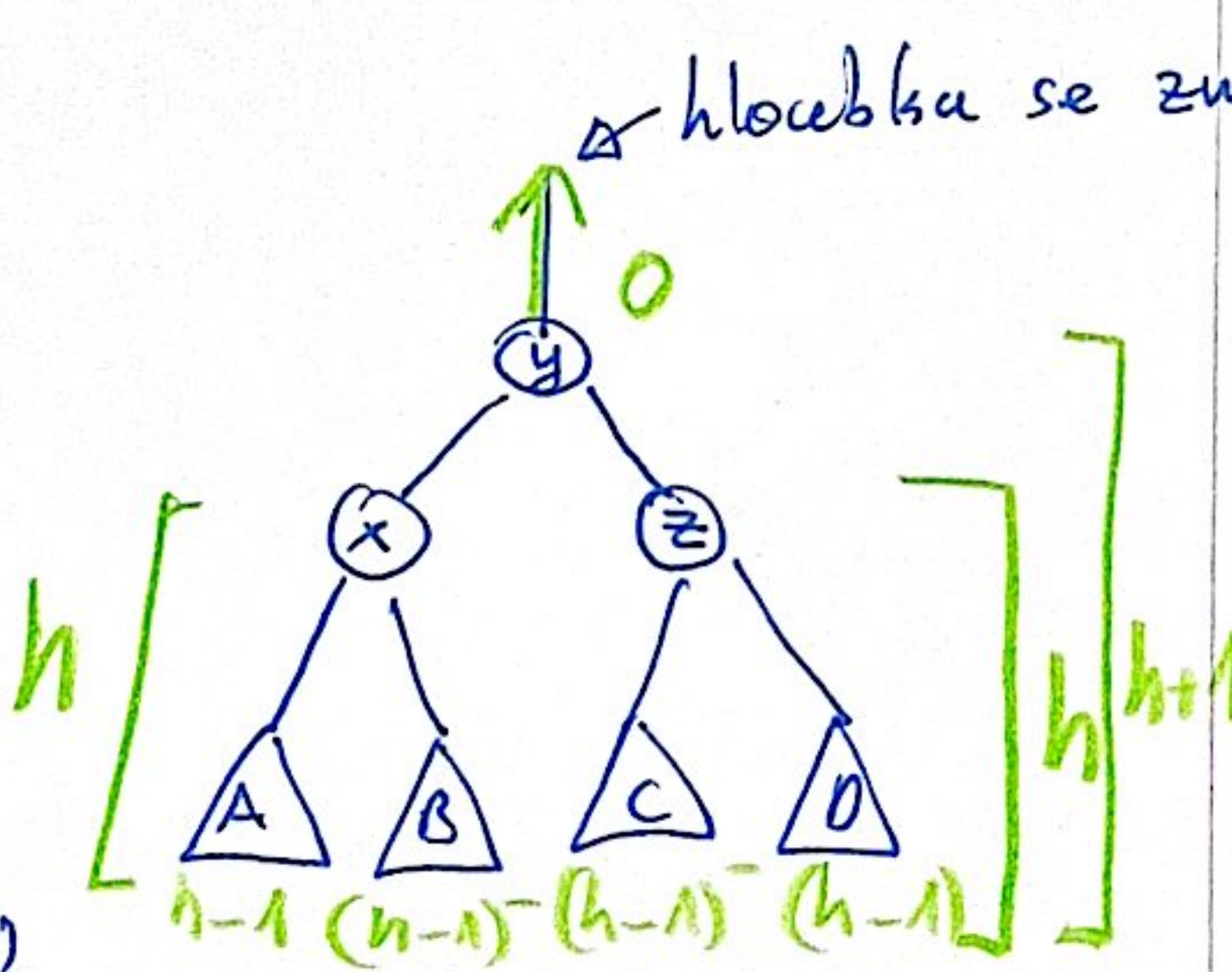
3b) $\delta(z) = 0$



rotace xz

3b skoro stejné co 3a

3c) $\delta(z) = -$



dvojitá rotace

hloubka se zmenšila
přeložením ze x y

složit z rotace yz, pak xy

u Delete se může rotovat vícerorát, v Insert ne, leu je jen 1 případ

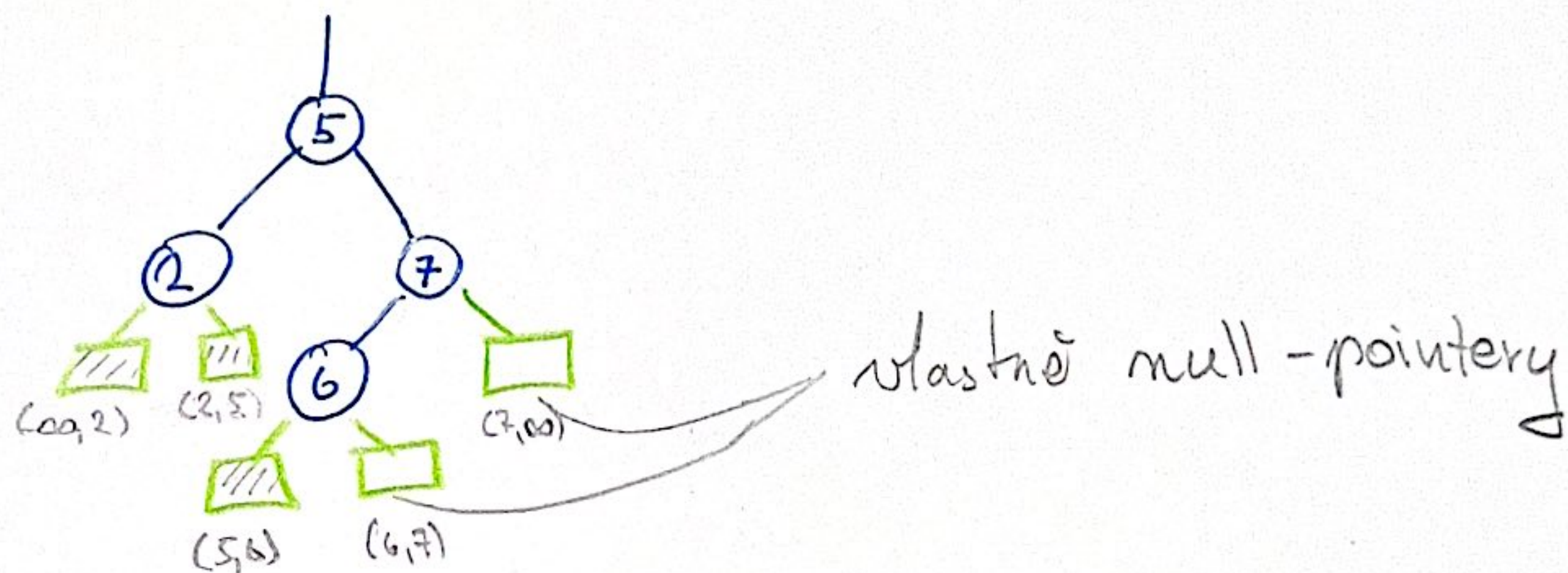
Číslo a "-" znamená to číslo nebo o 1 méně

Věta: Find, Insert, Delete v AVL stromě má časovou složitost $O(\log n)$

(a,b)-stromy

Externí vrcholy

↳ pokud vrchol nemá syny, tak dostane externí vrcholy



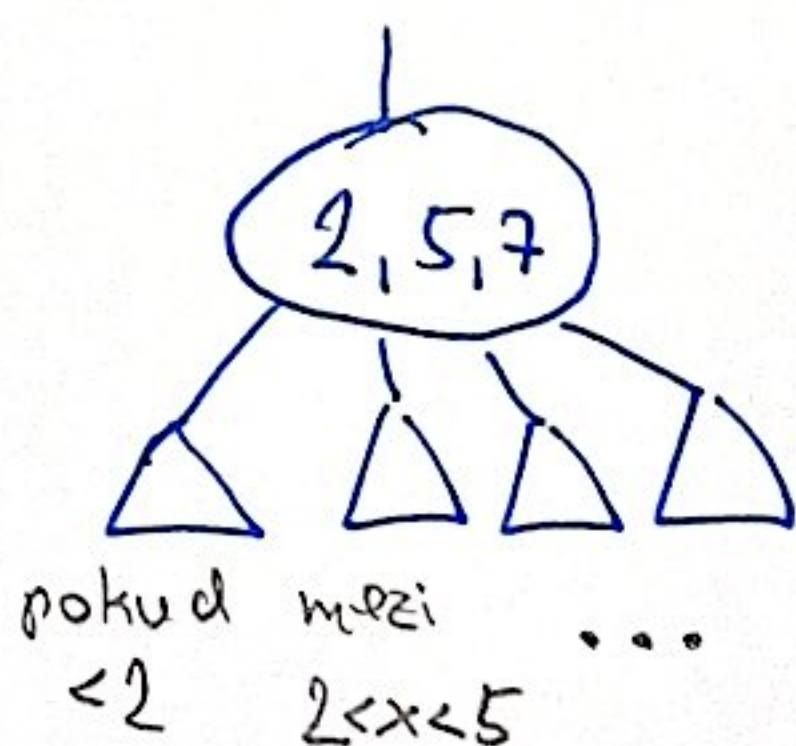
↳ jediné listy stromu jsou externí

↳ každý vrchol má právě 2 děti

↳ externí vrcholy odpovídají intervalům mezi klíči

↳ n vnitřních vrcholů $\Rightarrow n+1$ externích vrcholů

Vícecestné vyhledávací stromy



k klíčů $\Rightarrow k+1$ dětí



$$T_0 < x_1 < T_1 < x_2 < T_2 < \dots < x_k < T_k$$

dodefinovat $x_0 = -\infty$
 $x_k = +\infty$, pak $x_i < T_i < x_{i+1}$

Definice: (a, b) -strom pro $a \geq 2, b \geq 2a-1$

je obecný vyhledávací strom

s ext. vrcholy t.ž.: (platí axiomy)

① Počet dětí vnitřních vrcholů leží od a do b

↳ výjimka: kořen má počet dětí od 2 do b

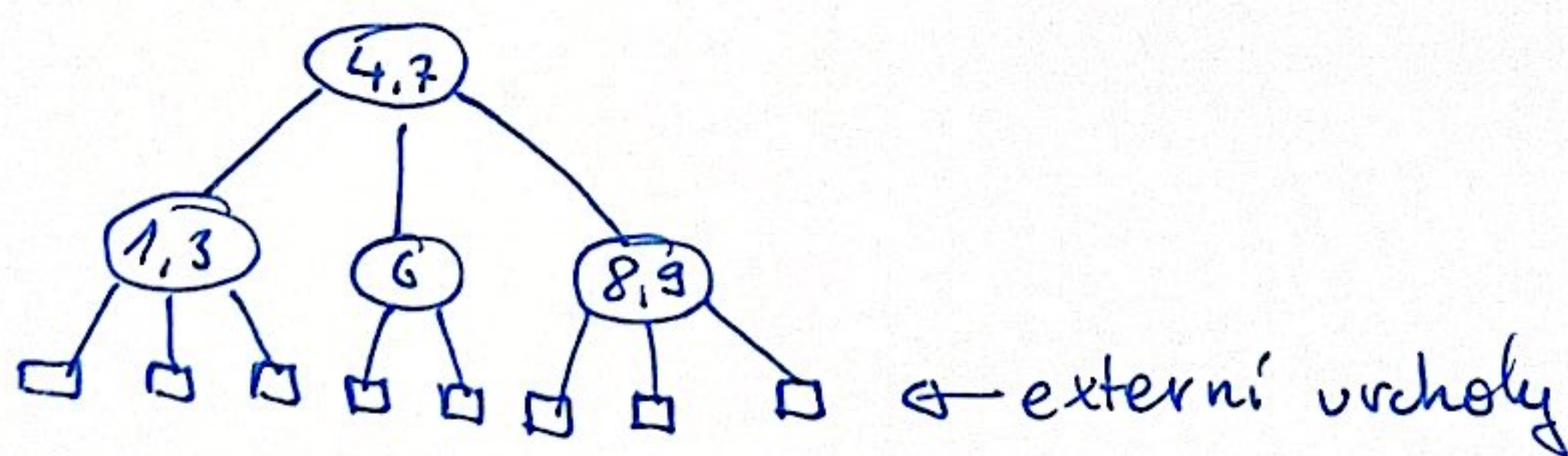
→ tedy a-1 až b-1 klíčů

→ tedy 1 až b-1 klíčů

② Všechny vnější vrcholy leží stejné hluboko

Příklad: $(2, 3)$ strom

$$\rightarrow a=2, \text{ pak } b=2 \cdot 2 - 1 = 3$$



rozmístění klíčů
do vrcholů může
být hodně

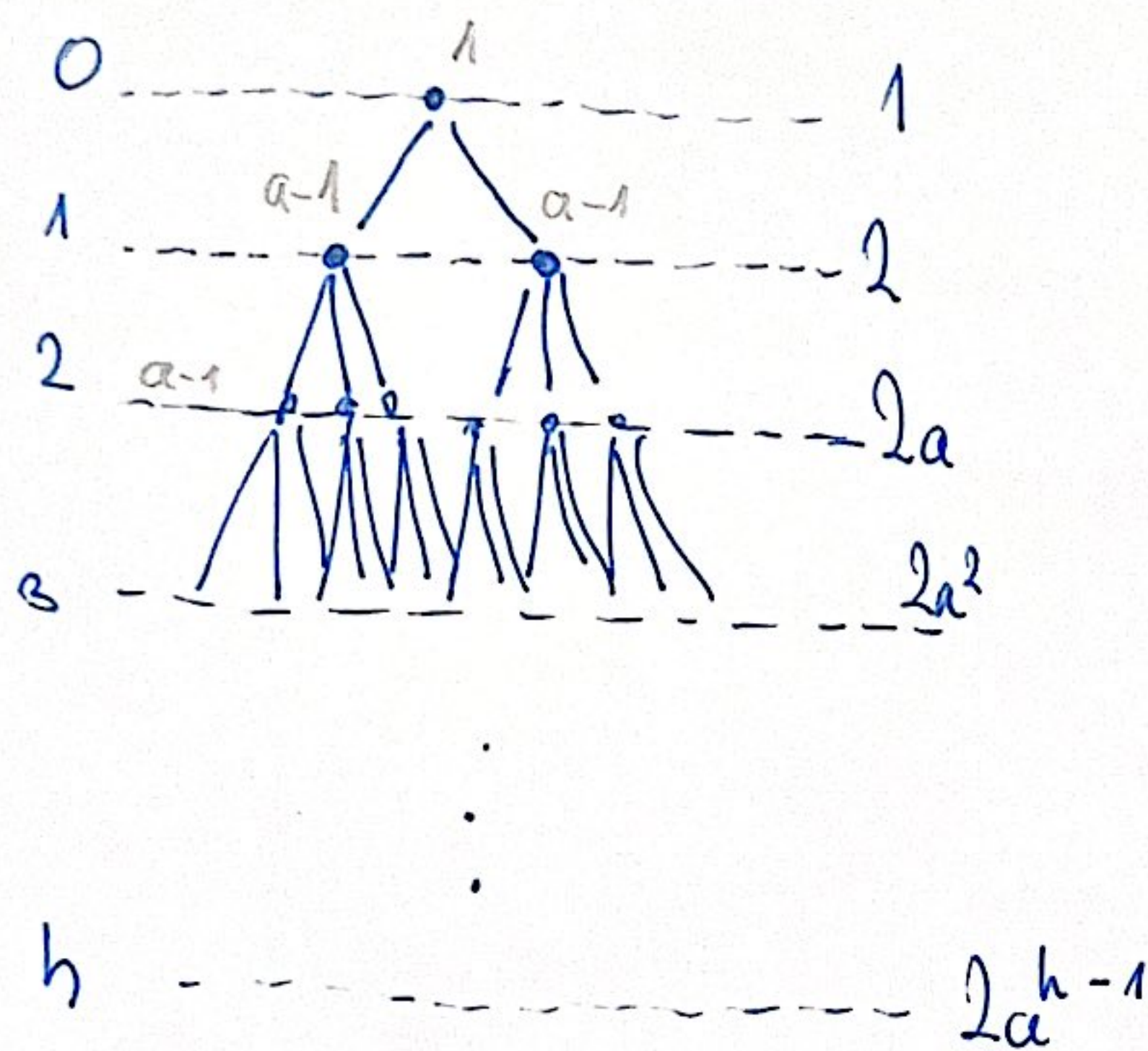
Lemma: (a,b) -strom s n klíči má $\Theta(\log n)$ hloubku.

Důkaz:

ukázat, že počet klíčů roste exponenciálně s hloubkou

m_h := minimální # klíčů v stromu hloubky h

uvažovat nejrychlý strom



$$m_h = 1 + \sum_{i=1}^{h-1} (a-1) 2a^{i-1} = \leftarrow \text{počet klíčů}$$

$$= 1 + (a-1) 2 \cdot \sum_{j=0}^{h-2} a^j =$$

$$= 1 + (a-1) 2 \cdot (a^{h-1} - 1) = 2 \cdot a^{h-1} - 1$$

$$\sum_{i=1}^{n-1} q^i = \frac{q^n - 1}{q - 1}$$

$$\sum_{j=0}^{h-2} a^j = \frac{a^{h-1} - 1}{a - 1}$$

$h \leq \log_a n$ - konstanta

odhad shora

M_h := max # klíčů ve stromu hloubky h

$$M_h = b^h$$

$$h \leq \log_b n \pm \text{konstanta}$$

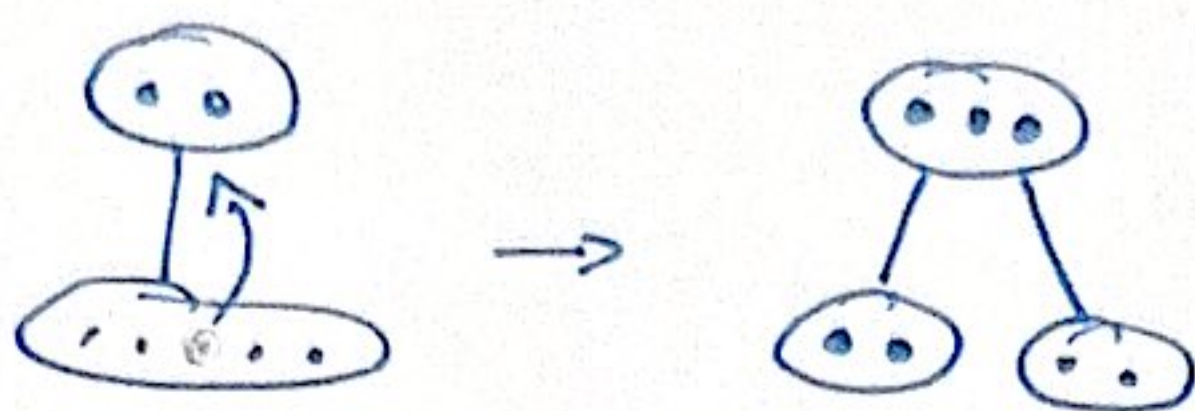
Find(x)

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

$\hookrightarrow$ ale hledat ve vrcholech b klíčů lze v $\log b$

pseudokód
v pravéci

Insert



↳ přidá se jako klíč do vrcholu

↳ pokud přeteče, rozdělí se a prostřední klíč se přesune do rodiče

↳ lze rozdělít i vrchol

na každé hladině konstantní čas $\Rightarrow O(\log n)$

Delete

① x není na poslední hladině



najde minimum $z R(x)$ a převede
na maximální pravoúhloho následníka
tím, že maximálním následníkem

$O(\log n)$

② x je na poslední hladině

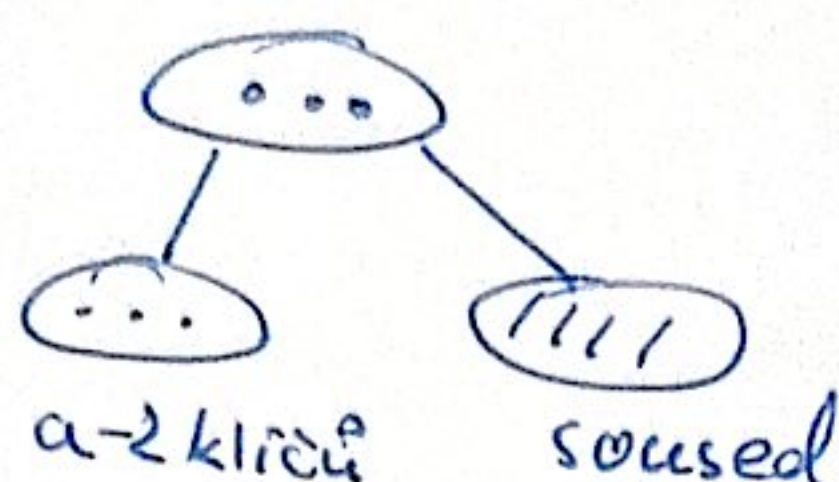


Pokud $\# \text{klíčů} \geq a-1 \Rightarrow \text{hotovo}$

Pokud $\# \text{klíčů} = a-2 \rightarrow \text{potřeba dále řešit}$

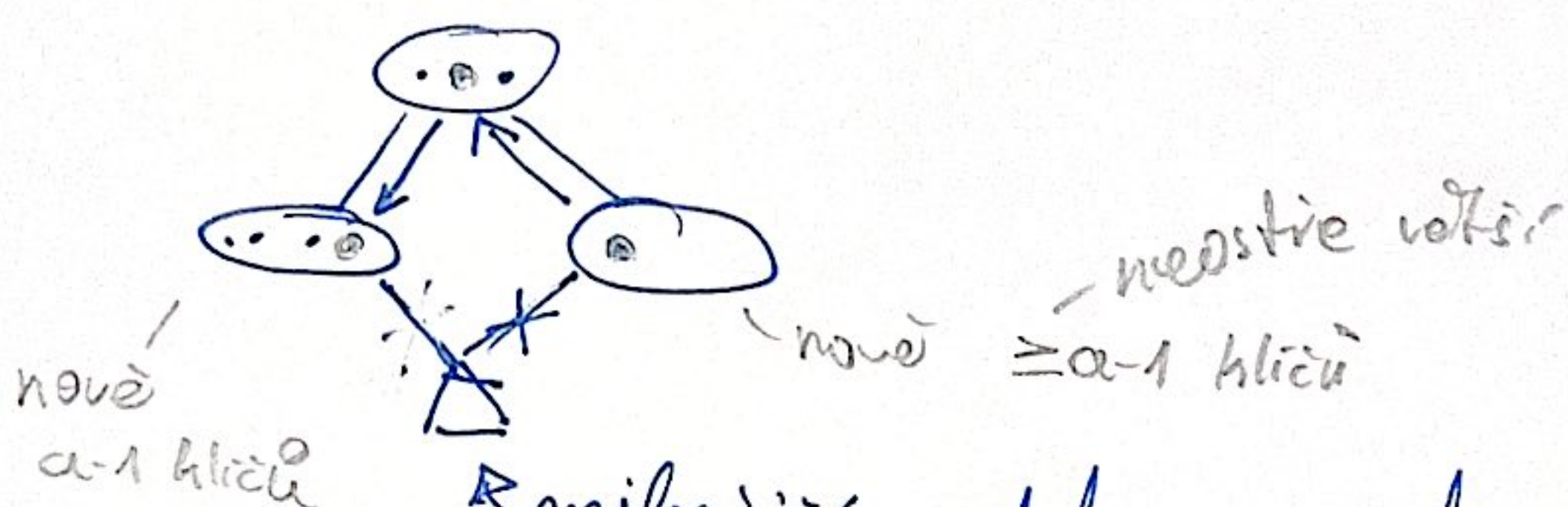
~~ŠPATNĚ STRANK, PŘESKOČ JEDNU!~~

využití suseda



(2a) sused má více než $a-1$ klíčů $> a-1$

↳ jeden klíč mu seberu



(2b) sused má $a-1$ klíčů

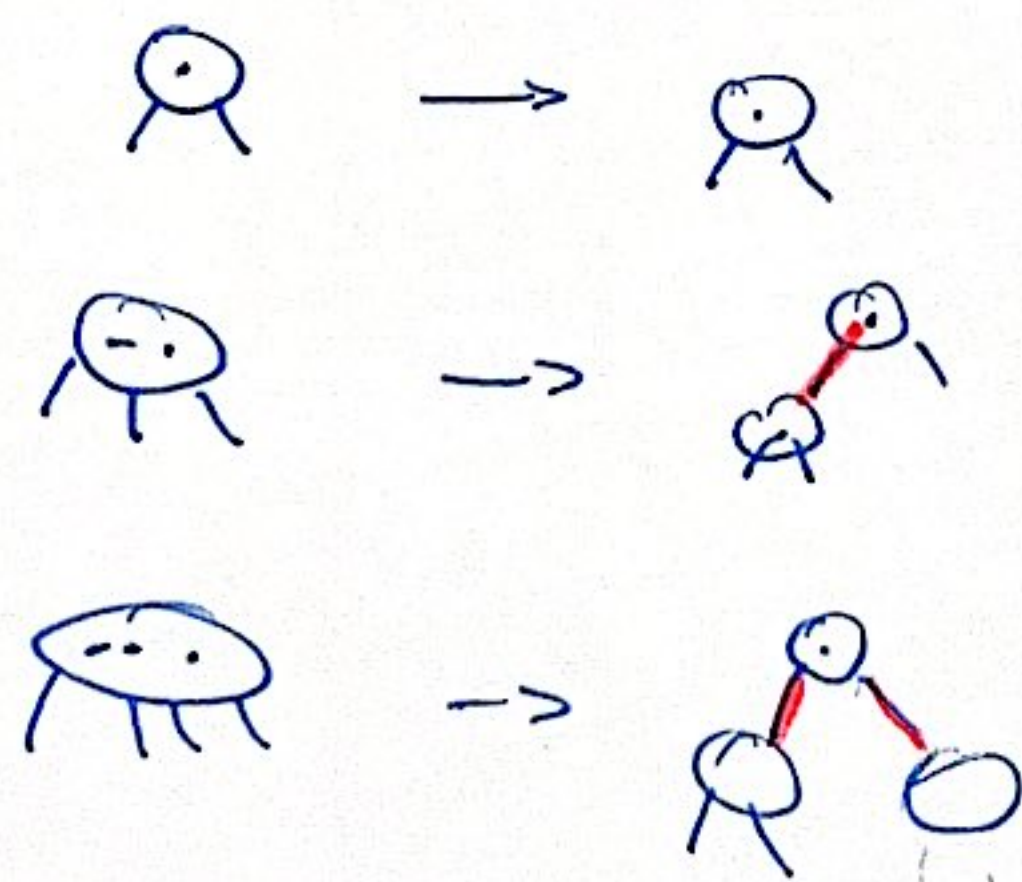
↳ seberu klíč rodiči

↳ pokud podleje otec, tak využije svého suseda, takže se může postupně snížit

Červeno-černé stromy -

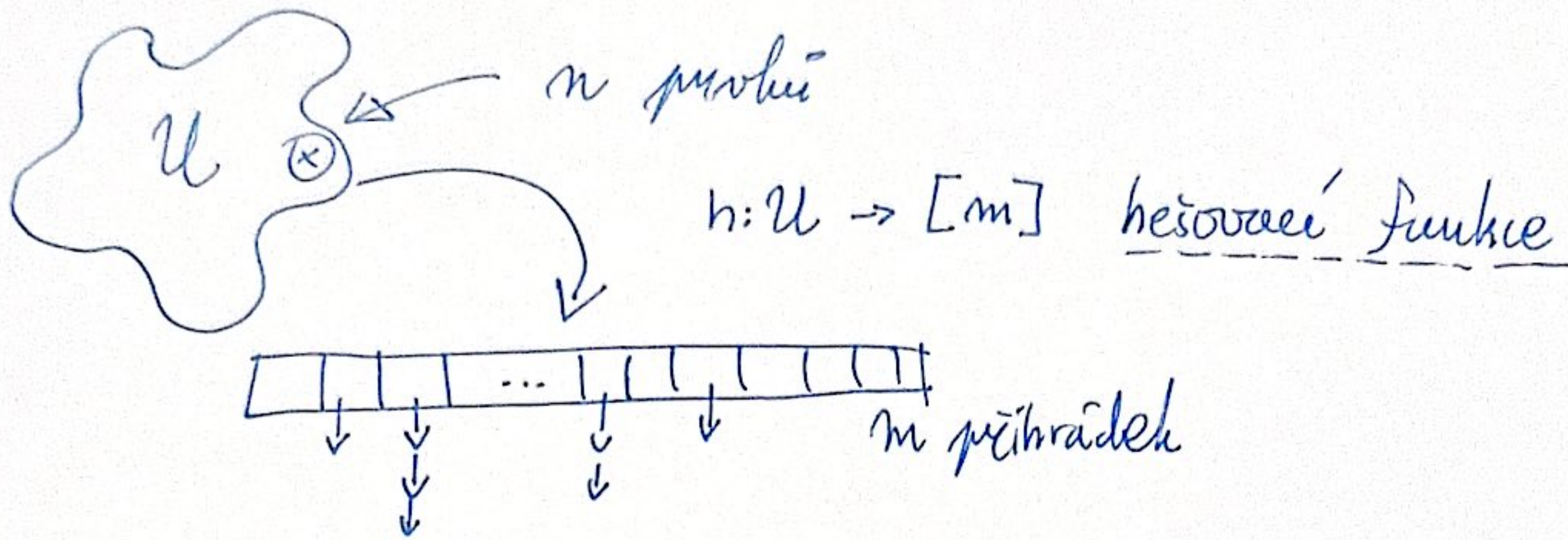
→ vlastně BVS, ale přeloží se operace z (4b) stromů BVS, aby to bylo jednodušší

↳ kódování (2,4) - stromu



Hešovací tabulka

švincl



prvků v příhrádce $\sim \frac{n}{m}$

- v každé příhrádce pamatují seznam prvků

$O(1)$ nebo-li $O(\text{délka řetězce})$ $\rightarrow$ pro všechny operace
 $\Delta \sim \frac{n}{m}$ Find, Insert, Delete

roztírací adresace

$\hookrightarrow$ jeden prvek v příhrádce



Find $\rightarrow$ neseke na příhrádku a přijde dolů, než narazím na prvek nebo přírůstek

Delete $\rightarrow$ neke jednoduše, proto pomníček nebo posun

Insert $\rightarrow$ na první prázdné

$\Downarrow$ pak přeorganizovat tabulku za čas

Chceme $m \in \Omega(n)$, tedy příhrádek např. $m \geq n$

bez řet i
 $m \geq L \cdot n$, ab
 vyple stejné

Dynamické rozšiřování tabulky

$m_0 = 2^0$
 $m_1 = 2^1$
 $m_2 = 2^2$
 $\vdots$
 $m_{k-1} = 2^{k-1}$
 $m_k = 2^k$

$\left. \begin{array}{l} \text{ } \end{array} \right\}$ kdy potřeba přehesovat

$\leftarrow n$ je větší tedy

$\rightarrow$ vytvořit novou tabulku, hešovací funkci a přičíst příhrádky

pek plati $\swarrow$ počet prvku je nekde mezi

$$2^{k-1} < n \leq 2^k = m \leq 2m$$

$$\Theta\left(\sum_{i=0}^{k-1} 2^i\right) = \Theta(2^k) = \underbrace{\Theta(m)}_{\Theta(n)}$$

$\nearrow$ potřebný čas na přehodnocení, tedy vkládání
n prvků má časovou složitost $\Theta(n)$

┐ - Vlastně něco jako ukládání peněz, např. 2 penízků
a pak zaplacení za přehodnocení 2 bankovky.

- je to tzv. amortizovaná časová složitost

Heisovací funkce

┐ - pokud $|U| = \infty$ a $|M| < \infty$, tak není problém
nalezt heisovací funkci, co všechny prvky hodí do
jedné přihrádky, tedy degraduje na lineární časovou
složitost

průměrná složitost

① přes vstup

② přes vlastní náhodná čísla

amortizovaná $\neq$

průměrná

(něco jako průměrná,
ale n průměrně je
potřeba vědět, přes co
se průměruje.)

Přklady heřovacích Junků

① lineární kongruence

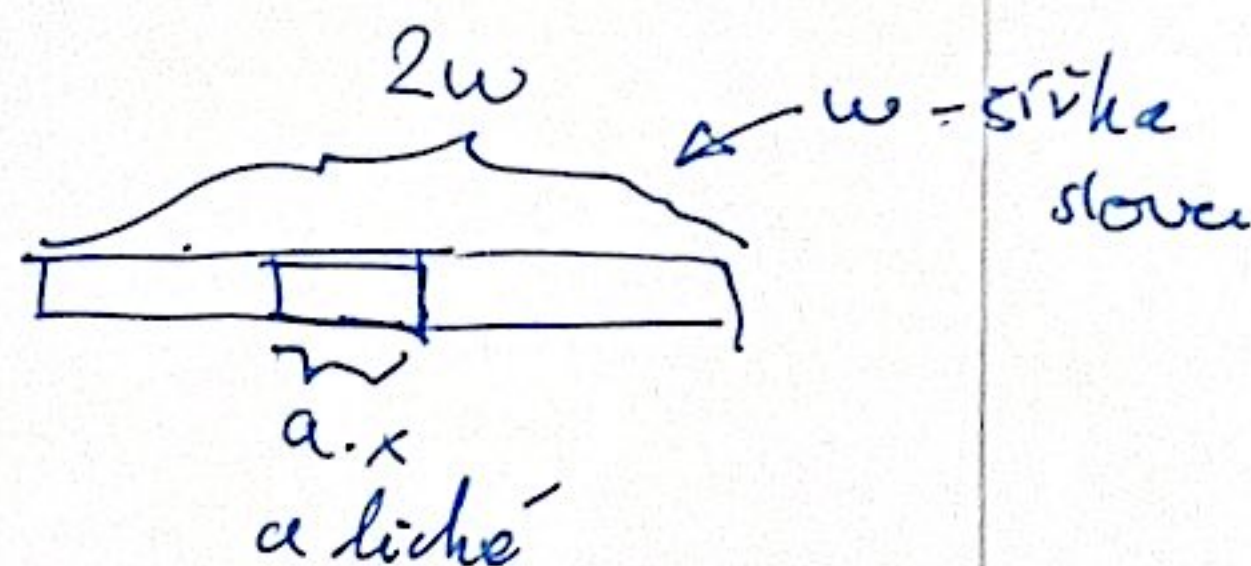
$$x \mapsto (ax+b) \bmod m$$

$$\gcd(a,m)=1 \quad ; m \text{ prvočíslo}$$

$$\text{BÚNO } b=0 \quad (\text{nezáleí na něm})$$

② multiply shift

$$h(x) = (ax \bmod 2^w) \Rightarrow (w-1)$$



$$\text{pro } (x_0, \dots, x_{d-1}) \in \mathbb{Z}_p^d \quad \leftarrow \text{vektory}$$

$$\text{③ } \vec{x} \rightarrow \vec{x} \cdot \vec{a} \quad \text{pro } \vec{a} \in \mathbb{Z}_p^d$$

$$\Leftrightarrow \left(\sum_{i=0}^{d-1} a_i x_i \right) \bmod p \quad \left. \vphantom{\sum_{i=0}^{d-1} a_i x_i} \right\} \text{skalární součin}$$

$$\text{④ } \text{polynomy} \quad \leftarrow \text{jednoduché, pomocí Hornerova schéma}$$

$$\left(\sum_{i=0}^{d-1} a^i x_i \right) \bmod p \quad a \in \mathbb{Z}_p$$

$\mathcal{H}$ systém funkcí z U do $[m]$

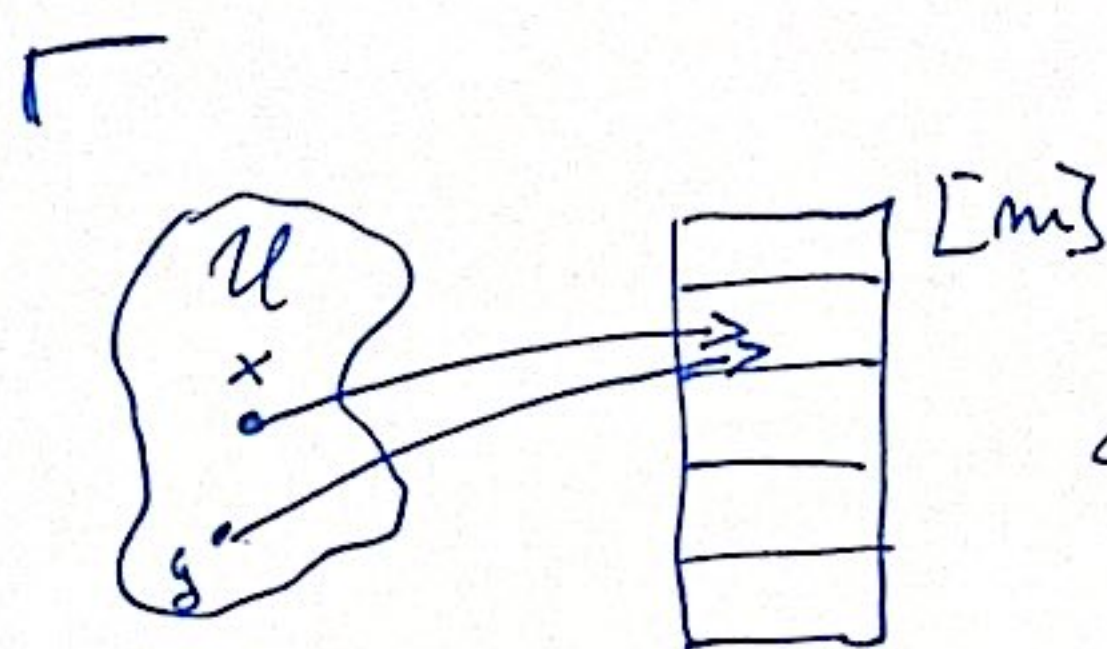
- efektivní výběr náhodně $h \in \mathcal{H}$
 - efektivní rozhodnutí h
- } $\mathcal{H}$ popsáno pomocí parametrů

Definice: Systém $\mathcal{H}$ funkcí z U do $[m]$ je

c -univerzální ($c \geq 1$) $\Leftrightarrow$

$$\forall x, y \in U : x \neq y : P_{h \in \mathcal{H}}(h(x) = h(y)) \leq \frac{c}{m}$$

ty příklady (4)
na straně 100
c-univerzální
systém funkcí



jaka je pravděpodobnost, že
2 prvky spadnou do ještě
přítvrdky

$\Rightarrow$ pro úplně náhodnou by to bylo $\frac{1}{m}$, takže
 c -univerzální je c-krát horší než pro úplně
náhodnou funkci

Lemma: Necht $\mathcal{H}$ je c -univerzální systém z $\mathcal{U}$ do $[m]$,

necht $x_1, \dots, x_n \in \mathcal{U}$ jsou navzájem různé

$y \in \mathcal{U}$

$$\mathbb{E}_{h \in \mathcal{H}} \underbrace{(\# i : h(x_i) = h(y))}_Z \leq \frac{c \cdot n}{m} + 1$$

vložit už
do tabulky

vždy v té příhradě, kde
hledám je aspoň ten 1
proch, tedy y

Dk.:

$$Z_i := \begin{cases} 0 \\ 1 \text{ pokud } h(x_i) = h(y) \end{cases}$$

$$\mathbb{E}(Z_i) = P(h(x_i) = h(y)) = \begin{cases} 1 \text{ pro } x_i = y \\ \leq \frac{c}{m} \text{ pro } x_i \neq y \end{cases}$$

$$\mathbb{E}(Z) = \sum_i \mathbb{E}(Z_i) \leq \frac{cn}{m} + 1$$

Definice: $\mathcal{S}$ systém funkcí z $\mathbb{Z}_p^d$ do $\mathbb{Z}_p$

$$\mathcal{S} := \{ h_{\vec{a}} \mid \vec{a} \in \mathbb{Z}_p^d \}$$

$$h_{\vec{a}}(\vec{x}) := \vec{a} \cdot \vec{x}$$

Věta: $\mathcal{S}$ je 1-univerzální

Důkaz: Pro $\vec{x}, \vec{y} \in \mathbb{Z}_p^d, \vec{x} \neq \vec{y}$

$$P_{h \in \mathcal{S}} (h(\vec{x}) = h(\vec{y})) = P_{\vec{a} \in \mathbb{Z}_p^d} (\underbrace{\vec{a} \cdot \vec{x} = \vec{a} \cdot \vec{y}}_{\vec{a}(\vec{x} - \vec{y}) = 0})$$

pravděpodobnost, že náhodný
vektor je kolmý na
první vektor

důkaz té
heuristické funkce
se sledováním
součinem

$$BUNO \quad z_{d-1} \neq 0$$

$$P_{\vec{a}} \left(\sum_{i=0}^{d-1} a_i z_i = 0 \right) = P_{\vec{a}} \left(\sum_{i=0}^{d-2} a_i z_i + a_{d-1} z_{d-1} = 0 \right)$$

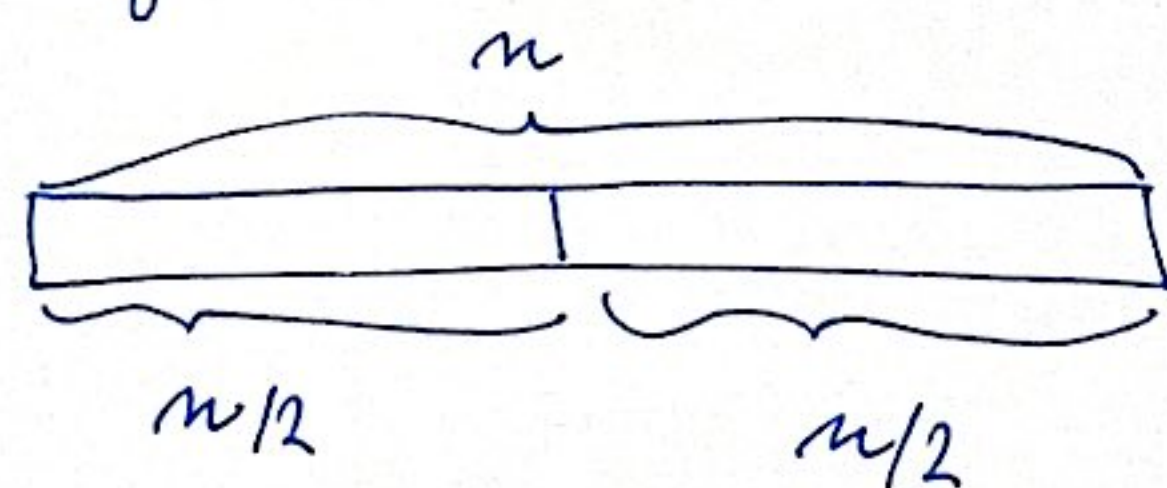
$$= P_{\vec{a}} \left(\underbrace{\sum_{i=0}^{d-2} a_i z_i}_{\text{něco}} = - \underbrace{a_{d-1} z_{d-1}}_{\neq 0} \right) = \frac{1}{p} \quad \text{p} = \text{počet prvků}$$

↑
prošli prvkem, že se dostaneme do
daného prvku a bude to splňovat rovnost
je $\frac{1}{p}$

Rozděli a panuj

↳ rozdělení na menší podproblémy a ty rekursivně
vyřešíme dalšími podproblémy, až je dokážeme řešit

Merge Sort



seřazené
výsledky

$\Theta(n)$
slévání
→

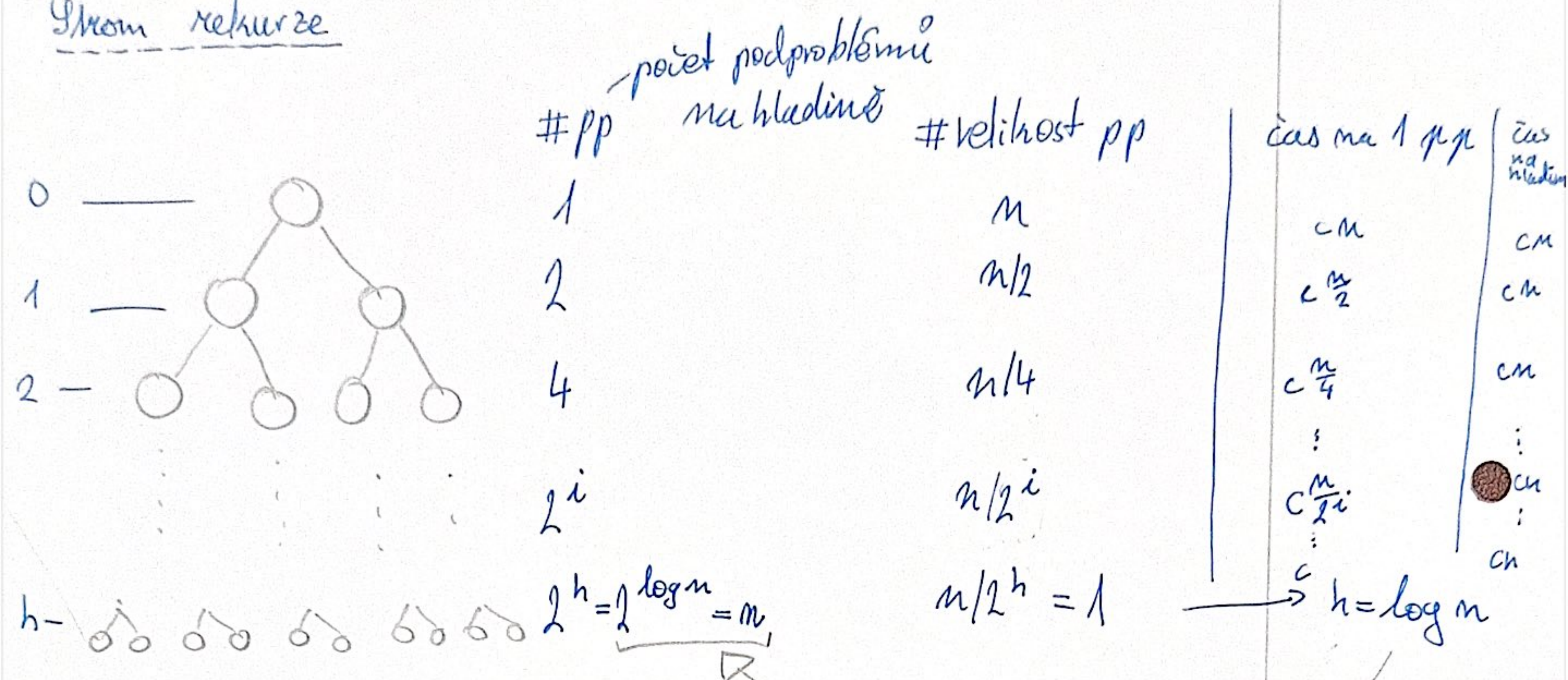
Merge Sort ($x_1, \dots, x_n$)

1. Pokud $n \leq 1$ vraťme $x_1, \dots, x_n$ (vstup) $\Theta(1)$
 2. $a_1, \dots, a_{\lfloor \frac{n}{2} \rfloor} \leftarrow \text{Merge Sort}(x_1, \dots, x_{\lfloor \frac{n}{2} \rfloor})$
 3. $b_1, \dots, b_{\lfloor \frac{n}{2} \rfloor} \leftarrow \text{Merge Sort}(x_{\lfloor \frac{n}{2} \rfloor + 1}, \dots, x_n)$
 4. $y_1, \dots, y_n \leftarrow \text{Merge}(a_1, \dots, a_{\lfloor \frac{n}{2} \rfloor}, b_1, \dots, b_{\lfloor \frac{n}{2} \rfloor})$
 5. return $y_1, \dots, y_n$
- $\Theta(n)$

$$T(n) = 2T(n/2) + \underbrace{\Theta(n)}_{c \cdot n}$$

$$T(1) = 1$$

Strom rekurze



na každé hladině trávíme stejný čas

↳ dokladujeme $c n \log(n) \in \underline{\Theta(n \log n)}$

Paměť

není "L.", protože se řeší vždy jen jeden podproblém

$$M(n) = M\left(\frac{n}{2}\right) + \Theta(1)$$

$$M(1)$$

$$M(n) = \Theta(1)$$

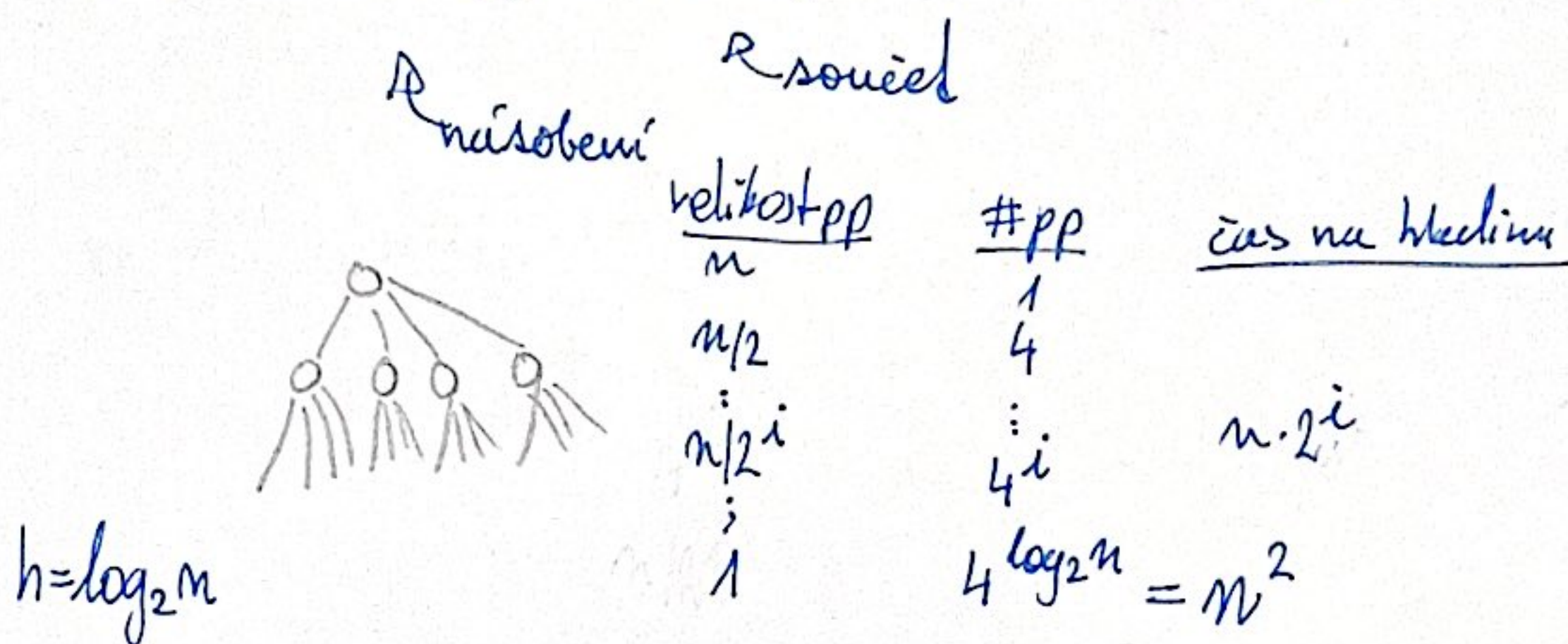
Násobení n-číselných čísel (Karacubův algoritmus)

$$X = \overbrace{[A \mid B]}^{n=2^k} \quad X = A \cdot 10^{\frac{n}{2}} + B$$

$$Y = [C \mid D] \quad Y = C \cdot 10^{\frac{n}{2}} + D$$

$$XY = AC \cdot 10^n + (AD + BC) \cdot 10^{\frac{n}{2}} + BD$$

$$T(n) = 4 \cdot T\left(\frac{n}{2}\right) + \Theta(n) \rightarrow T(n) \in \Omega(n^2)$$



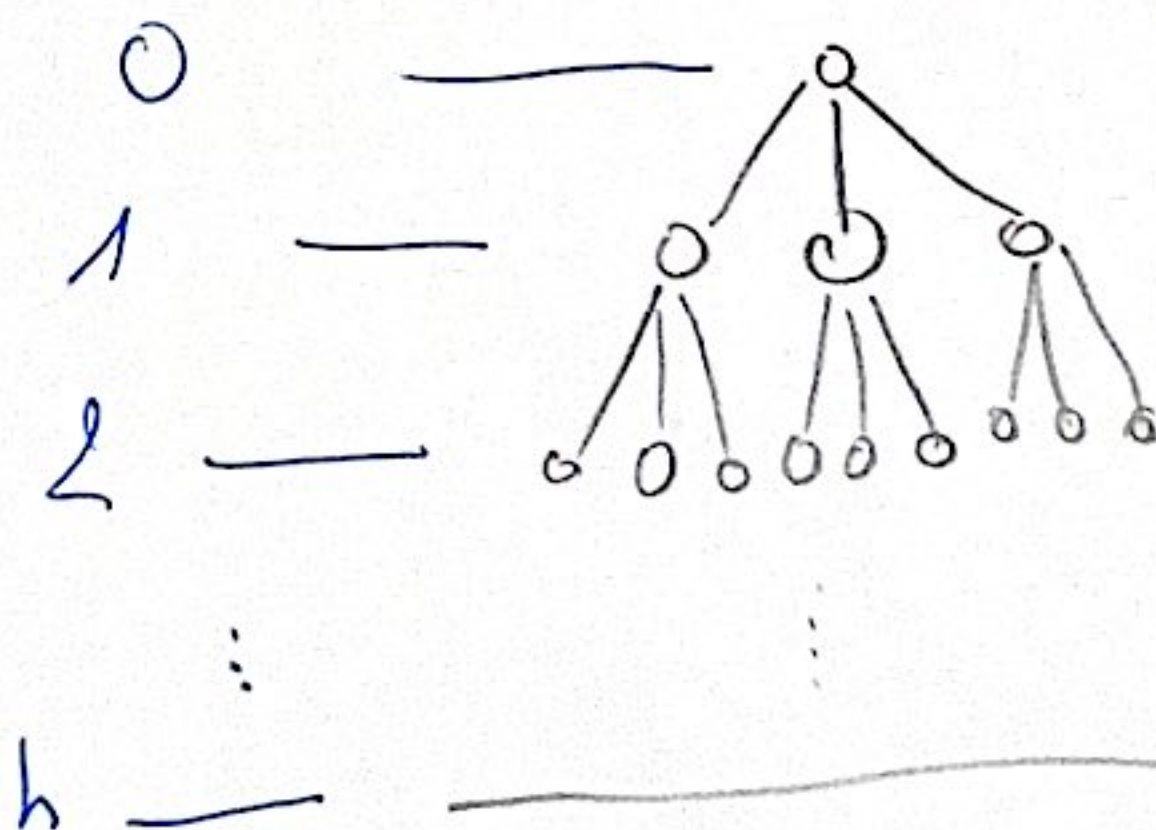
→ špatná složitost, protože normální násobení je n^2

Karacubův

$$(A+B)(C+D) = AC + AD + BC + BD$$

↳ Lze spočítat AC, BD a $(A+B) \cdot (C+D) - AC - BD$, což jsou jen 3 násobení, tedy:

$$T(n) = 3T(n/2) + \Theta(n) \quad \underline{\text{BÚNO}} \rightarrow n = 2^{\text{něco}}$$



$$\# \text{ podproblémů} = 3^i$$

$$\text{velikost podproblému} = \frac{n}{2^i}$$

↳ $h = \log_2 n \rightarrow$ na poslední hladině se rekurze zastaví

$$\text{čas/pp} \in \Theta\left(\frac{n}{2^i}\right)$$

$$\text{čas/hladinu} \in \Theta\left(n \cdot \frac{3^i}{2^i}\right)$$

$$T(n) = \sum_{i=0}^h \Theta\left(n \cdot \left(\frac{3}{2}\right)^i\right) = \Theta\left(n \cdot \sum_{i=0}^h \left(\frac{3}{2}\right)^i\right)$$

$$q^0 + q^1 + \dots + q^h = \frac{q^{h+1} - 1}{q - 1}$$

$$\frac{\left(\frac{3}{2}\right)^{h+1} - 1}{\frac{3}{2} - 1}$$

asymptoticky rovná se $\left(\frac{3}{2}\right)^h$

$$\approx n \cdot \left(\frac{3}{2}\right)^{\log_2 n} = \frac{n \cdot 3^{\log_2 n}}{2^{\log_2 n}} = 3^{\log_2 n} = \left(2^{\log_2 3}\right)^{\log_2 n} =$$

$$= \left(2^{\log_2 3}\right)^{\log_2 n} = n^{\log_2 3} \approx n^{1.59}$$

$$\underline{T(n) \in \Theta(n^{\log_2 3})}$$

Kuchařková věta

$$T(n) = a \cdot T(n/b) + \Theta(n^c)$$

BÚNO $n = b^{\text{něco}}$



$$\begin{array}{l} 1 \\ a \\ a^2 \\ \vdots \\ \frac{n}{b} \\ \frac{n}{b^2} \end{array}$$

i-tá hladina

$$\# \text{ pp} = a^i$$

$$\text{velikost pp} = \frac{n}{b^i} \quad \left. \vphantom{\frac{n}{b^i}} \right\} h = \log_b n$$

$$\text{čas / 1 pp} \in \Theta\left(\left(\frac{n}{b^i}\right)^c\right)$$

$$\text{čas / 1 hladinu} \in \Theta\left(a^i \cdot \left(\frac{n}{b^i}\right)^c\right)$$

$$n^c \left(\frac{a}{b^c}\right)^i$$

$$T(n) = \sum_{i=0}^h \Theta\left(n^c \cdot \left(\frac{a}{b^c}\right)^i\right) = \Theta\left(n^c \cdot \sum_{i=0}^h \underbrace{\left(\frac{a}{b^c}\right)^i}_q\right)$$

$$\textcircled{1} q < 1$$

$$\text{konst} \leq \sum_{i=0}^{\infty} q^i \leq \text{konst.} \quad \underline{T(n) \in \Theta(n^c)} \quad \& \text{ protože } \Theta(1) \text{ kóduje}$$

② $q=1$

$$\sum_{i=0}^h q^i = h+1 \in \Theta(\log n)$$

$$\underline{T(n) \in \Theta(n^c \cdot \log(n))}$$

③ $q>1$

$$\sum_{i=0}^h q^i = \frac{q^{h+1}-1}{q-1} \stackrel{\text{asymptoticky zanedbat}}{\approx} q^h = \left(\frac{a}{b^c}\right)^{\log_b n}$$

$$T(n) \in \Theta(n^c \cdot \underbrace{\frac{a^{\log_b n}}{b^{c \cdot \log_b n}}}_{n^c}) = \Theta(a^{\log_b n}) =$$

$$= \Theta(b^{\log_b a \cdot \log_b n}) \Rightarrow \underline{T(n) \in \Theta(n^{\log_b a})}$$

Jen shrnutí

Kuchařková věta na řešení rekurencí (Master Theorem)

Rekurence $T(n) = a \cdot T(n/b) + \Theta(n^c)$

$$T(1) = T(0) = 1$$

Pro $a \in \mathbb{N}^+$, $b > 1$, $c \geq 0$

Má řešení

① $T(n) \in \Theta(n^c)$

② $T(n) \in \Theta(n^c \log n)$

③ $T(n) \in \Theta(n^{\log_b a})$

pro $q < 1$

pro $q = 1$

pro $q > 1$

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

Důkaz:

Pro $n = b^{\text{něco}}$ (který) - na předchozích stránkách

def. x^-, x^+ ... jako zaokrouhlení na nejbližší mocninu b (nahoru (dole))

$$n^- \leq n \leq n^+$$

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

$\frac{n^+}{n^-} = b$ $\rightarrow$ takže asymptoticky se nemění
pokud vynásobíme řešení
konstantou

$$T(n^-) \approx T(n^+) \rightarrow \\ \Rightarrow T(n) \approx$$

Pro ně známé algoritmy

alg.	a <small>#problémů</small>	b <small>velikost pp</small>	c <small>rozdělení problému</small>	d <small>a skvělení</small>	$T(n)$
Mergesort	2	2	1	1	$n \log n$
Karacubův	3	2	1	1,5	$n^{\log_2 3}$
naivní násobení	4	2	1	2	$n^{\log_2 4} = n^2$
bin. vyhledávání	1	2	0	1	$\log n$
naivní násobení matic	8	2	2	2	$n^{\log_2 8} = n^3$
Strassen	7	2	2	$\frac{7}{4}$	$n^{\log_2 7} = n^{2,808}$ <small>$\leftarrow$ otevřený problém</small>

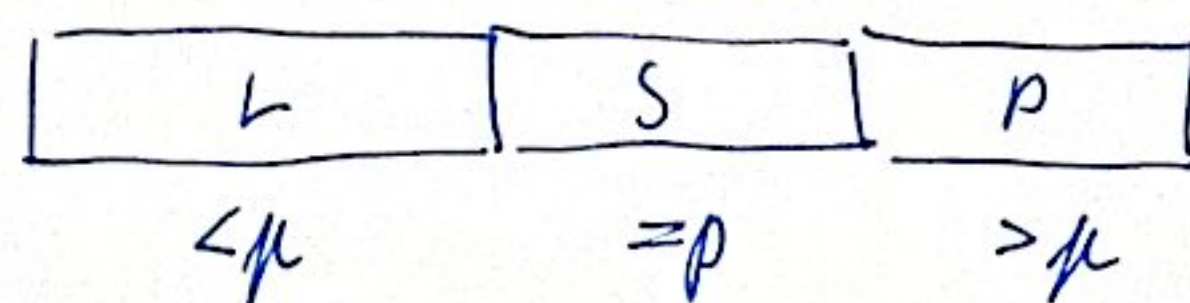
Strassenův algoritmus - maticové násobení

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

1 součin matic $n \times n$ na 8 součinů matic $\left(\frac{n}{2}\right) \times \left(\frac{n}{2}\right)$,
což nepomůže, ale lze ušetřit 7 součinů

k-tý nejmenší prvek = Quickselect

rozdělení na 3 části podle náhodného prvku



$$1 \leq k \leq n$$

Quick Select ($x_1, \dots, x_n; k$):

0. Pokud $n=1$: vraťme x_1

1. zvolíme prvok $p \in \{x_1, \dots, x_n\}$

2. $L \leftarrow \{x_i \mid x_i < p\}$

$S \leftarrow \{x_i \mid x_i = p\}$

$P \leftarrow \{x_i \mid x_i > p\}$

3. Pokud $k \leq |L|$:

vraťme $QS(L, k)$

jinak je-li $k \leq |L| + |S|$:

vraťme p

jinak:

vraťme $QS(P, k - |L| - |S|)$

ukončovací
podmínka
rekurze

postupnosti
multimnožiny
↑
opulsní množina

Příklady:

① P je medián

$$|L|, |P| \leq \frac{n}{2}$$

↗ rozdělování na skupiny

pak platí $T(n) = T\left(\frac{n}{2}\right) + \Theta(n)$

$$\left. \begin{array}{l} a=1 \\ b=2 \\ c=1 \\ \hline q = \frac{1}{2} < 1 \end{array} \right\} T(n) \in \Theta(n)$$

napi.

$\Theta(n^1)$, kde

$c=1$, tedy c je exponent
kolik stojí rozdělun

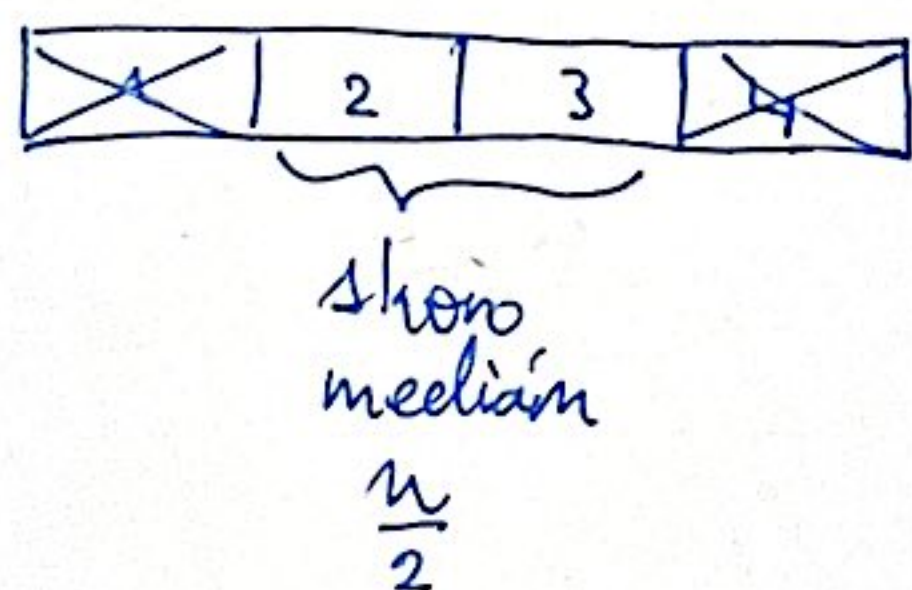
② P je minimum

$$|L|=0, |S|=1, |P|=n-1$$

$$T(n) = T(n-1) + \Theta(n)$$

↳ $T(n) \in \Theta(n^2)$ - v nejhorším případě

③ P je skoromedián



→ vždy alespoň $\frac{1}{4}$ zachováme

$$T(n) = T\left(\frac{3}{4}n\right) + \Theta(n)$$

↳ $T(n) \in \Theta(n)$

↳ proto náhodně zvolit, aby průměrná časová složitost byla $\Theta(n)$

Pravděpodobnostní algoritmus na hledání skoromediánu

- náhodně volíme prvky L a P a necháme skoromedián, což ověříme tím, že si vytvoříme L, S, P
- průchod singly trvá $\Theta(n)$, takže kolik v průměru je potřeba pokusů, než se dostaneme do skoromediánu
 $\hookrightarrow$ pravděpodobnost nalezení skoromediánu je $\frac{1}{2}$

Lemma (o dělení)

$$\mathbb{E}[\# \text{ pokusů do } \underbrace{1. \text{ úspěšného}}_{\text{nastane s pravděpodobností } p}] = \frac{1}{p}$$

$$\mathbb{E}[\# \text{ pokusů}] \leq 2$$

$$\mathbb{E}(\text{čas. složitosti}) \in \Theta(n)$$

Quickselect s náhodnou volbou pivotu

$$n = N_1, N_2 \mid N_3, N_4, \dots$$

~~~~~  
konec epochy, vybrali jsme skoro medián

$$\mathbb{E}(\# \text{ pokusů v epoše}) \leq 2$$

$$\left| \underbrace{N_i, N_{i+1}}_{\leq N_i}, N_{j-1} \right| N_j$$

$$\Rightarrow \mathbb{E}(\text{čas na epochu}) \in \Theta(N_i)$$



|         |         |         |
|---------|---------|---------|
| $L < p$ | $S = p$ | $P > p$ |
|---------|---------|---------|

pivot  $p$

pokud  $k \leq |L|$  : rekurzivně zavolat  $na(L, k)$

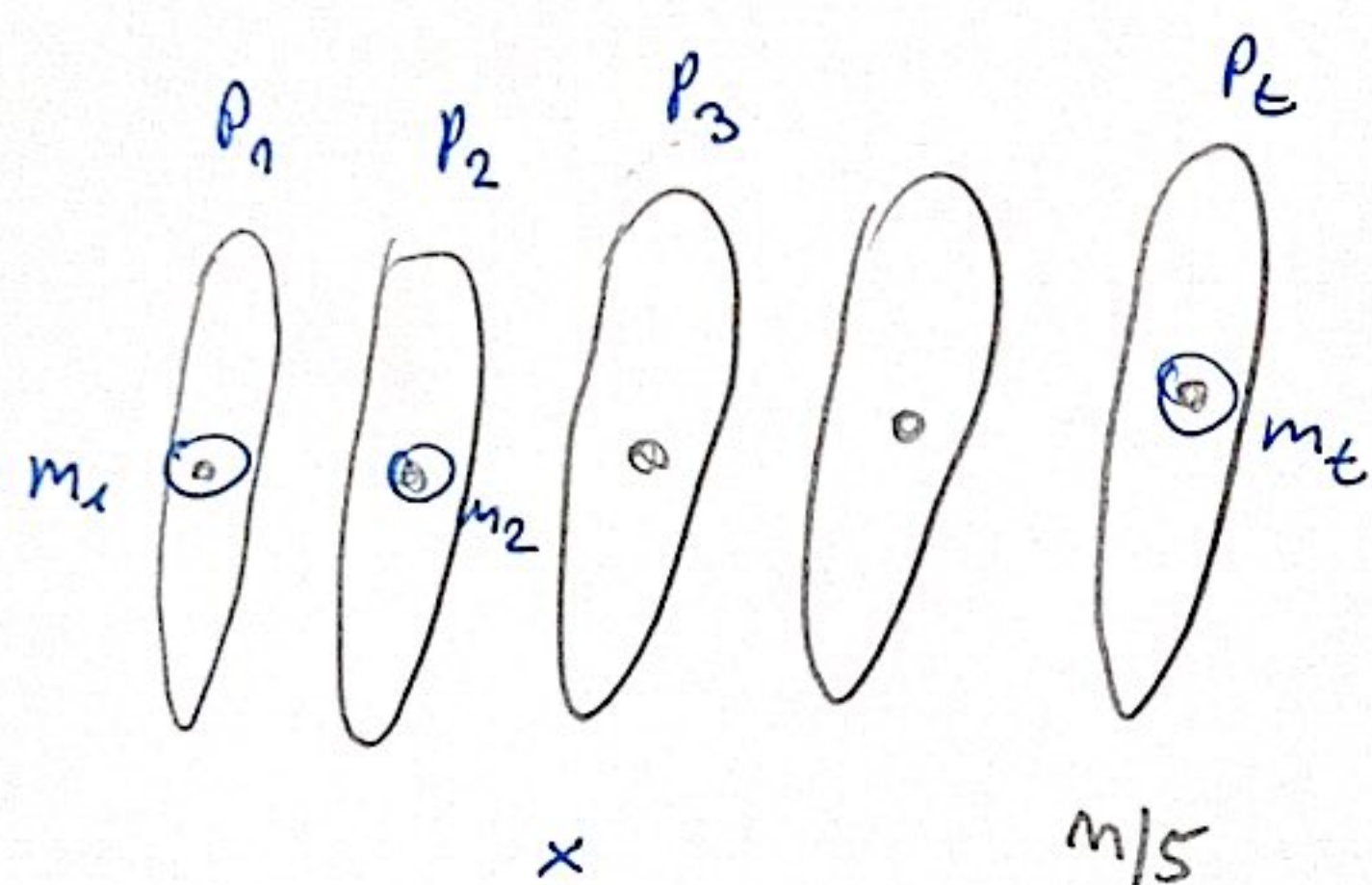
$|L| < L \leq |L| + |K|$  : vrátíme pivotu  $p$

$|L| + |S| < k$  : rekurzivně  $(P, k - (|L| + |S|))$

složitost : nejhorší  $\Theta(n^2)$

nejlépe  $\Theta(n)$

průměr  $\Theta(n)$  pro náhodnou volbu pivotu



Select( $x_1, \dots, x_n, k$ ):

1. Pokud  $n \leq 5$ , doručíme hrubou silou
2. Rozdělíme  $X$  na pětice  $P_1, \dots, P_t$
3. Pro  $i = 1, \dots, t$  :  $m_i \leftarrow \text{Medián}(P_i)$
4.  $p \leftarrow \text{Select}(m_1, \dots, m_t, \lceil t/2 \rceil)$
5. Rozdělíme  $X$  na  $L, S, P$  podle  $p$
6. Pokud  $k \leq |L|$  : vrátíme  $\text{Select}(L, k)$

jinak pro  $k \leq |L| + |S|$  vrátíme  $p$

jinak vrátíme  $\text{Select}(P, k - |L| - |S|)$

proste doplnim prvky,  
co neovlivni vystup

ma linearni  
složitost







QuickSort ( $x_1, \dots, x_n$ ) :

1. Pokud  $n \leq 1$  : vrat'  $x_1$

2.  $p \leftarrow \text{pivot}$

3. Rozdel na  $L, S, P$

4. vrat' quicksort( $L$ ) +  $S$  + quicksort( $P$ )



# Dynamické programování

## 58) Nejdelší rostoucí podsekvence

↳ posloupnost → celá čísla

↳ podposloupnost → po vyškrtání prvků, tak zbylé zachovají pořadí

Formulace  
problému

Vstup:  $x_1, \dots, x_n \in \mathbb{Z}$   
výstup:  $i_1 < i_2 < \dots < i_k : x_{i_1} < x_{i_2} < \dots < x_{i_k}$   
 $k$  maximální možné

Příklad

3 14 15 92 65 35 89 79 32 38 46 26 43

↑ hloubový algoritmus nefunguje (pokud nehlédáme zrovna minimální kóstru.)

rekurzivní řešení  
s časovou složitostí

$\Theta(2^n)$

↳ zkoušení všech  
možných podsekvencí

$NRP(i)$ :

Pokud  $T[i] \neq \emptyset$ : vrať  $T[i]$   
 $d \leftarrow 1$

pro  $j = i+1, \dots, n$ :

Pokud  $x_j > x_i$ :

$d \leftarrow \max(NRP(j) + 1, d)$

$T[i] \leftarrow d$

Vrať  $d$

→  $n$  volání a každý  
 $\Theta(n)$  času, tedy

$\Theta(n^2)$  času celkově

po cachování

obecný  
vzorec

→

aby bylo jasné, kde začít, tak přidám na začátek  $-\infty$

a zavolám  $NRP(0) - 1$



Iterativní řešení → vlastně postupují obruadu 2 for cykly

Pro  $i = n, n-1, \dots, 1$ :

$T[i] \leftarrow 1$

$P[i] \leftarrow \emptyset$

pro  $j = i+1, \dots, n$ :

Pokud  $(x_j > x_i)$  a  $(T[j] + 1 > T[i])$ :

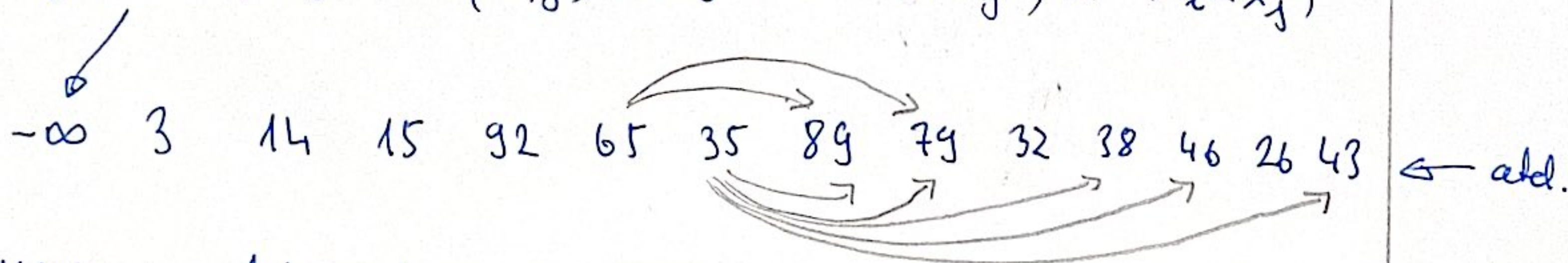
$T[i] \leftarrow T[j] + 1$

$P[i] \leftarrow j$

→ zase časová  
složitost  
 $\Theta(n^2)$ , protože  
2 for cykly,  
ale můžeme  
postoupnost  $i$   
napsat  
 $P[i]$  vždy obsahuje  
index následujícího pruhu

Grafový pohled

$V_G = \{0, \dots, n\}$  :  $(i, j) \in E_G \iff (i < j) \wedge (x_i < x_j)$



Hledání nejdelší cesty v grafu ( $-\infty$  leží na nejdelší cestě)

↳ v DAGu lze nalézt indukci podle topologického  
uspořádání

→ vlastně ten  
samý algoritmus  
co nahoře



## 59) Editční vzdálenost řetězců

koule  $\left\{ \begin{array}{l} \rightarrow \text{boule} = \text{změna } 1 \text{ znaku} \\ \rightarrow \text{koloule} = \text{vlození } 1 \text{ znaku} \\ \rightarrow \text{kule} = \text{smazání } 1 \text{ znaku} \end{array} \right\}$  (zv. editční operace v řetězci)

*jezdícím stiskem*

### Výpočet editční vzdálenosti

$\rightarrow$  kolik minimálně stačí přehlept, aby se z jednoho slova stalo jiné  $\leftarrow$  editčních operací

### Levenshteinova vzdálenost

Def: Editční vzdálenost řetězců  $x_1, \dots, x_n$  a

$y_1, \dots, y_m$  je

min. # editčních operací, které z  $x$  udělají  $y$

$\rightarrow$  proto se značí  $L(x, y)$

- chová se jako vzdálenost (operace navzájem dostanou původní řetězec), platí trojúhelníková nerovnost

Co se děje nalevo? mám 2 řetězce

$\boxed{x_1 \text{ --- } x_n}$   
 $\boxed{y_1 \text{ --- } y_m}$

1) nic : pro  $x_1 = y_1$   $L(x, y) = L(x[2:], y[2:])$

2) změna  $x_1$  na  $y_1$   $L(x, y) = 1 + L(x[2:], y[2:])$

3) vložení  $y_1$   $L(x, y) = 1 + L(x, y[2:])$

4) smazání  $x_1$   $L(x, y) = 1 + L(x[2:], y)$

metrika na množině všech řetězců  $\rightarrow$



Edit (i, j):  $\leftarrow$  lokle počíta  $L(x[i:], y[j:])$

Pokud  $i > n$  : vrátíme  $m - j + 1$

Pokud  $j > m$  : vrátíme  $n - i + 1$

Je-li  $L(i, j) \neq \emptyset$  : vrátíme  $L(i, j)$

$Z \leftarrow \text{Edit}(i+1, j+1) + (1 \text{ pokud } x_i \neq y_j)$

$V \leftarrow \text{Edit}(i, j+1) + 1$

$S \leftarrow \text{Edit}(i+1, j) + 1$

vrátíme  $\min(Z, S, V)$

$L[i, j]$  - přiřadím, než vrátím

počet možných  $\cdot i = n+1$   
 $\cdot j = m+1$  } počet různých volání  $O(n \cdot m)$

$\leftarrow$  vložit bývalý znak y

$\leftarrow$  smazat bývalý znak x

$\leftarrow$  změna/nic

$\leftarrow$  vložení

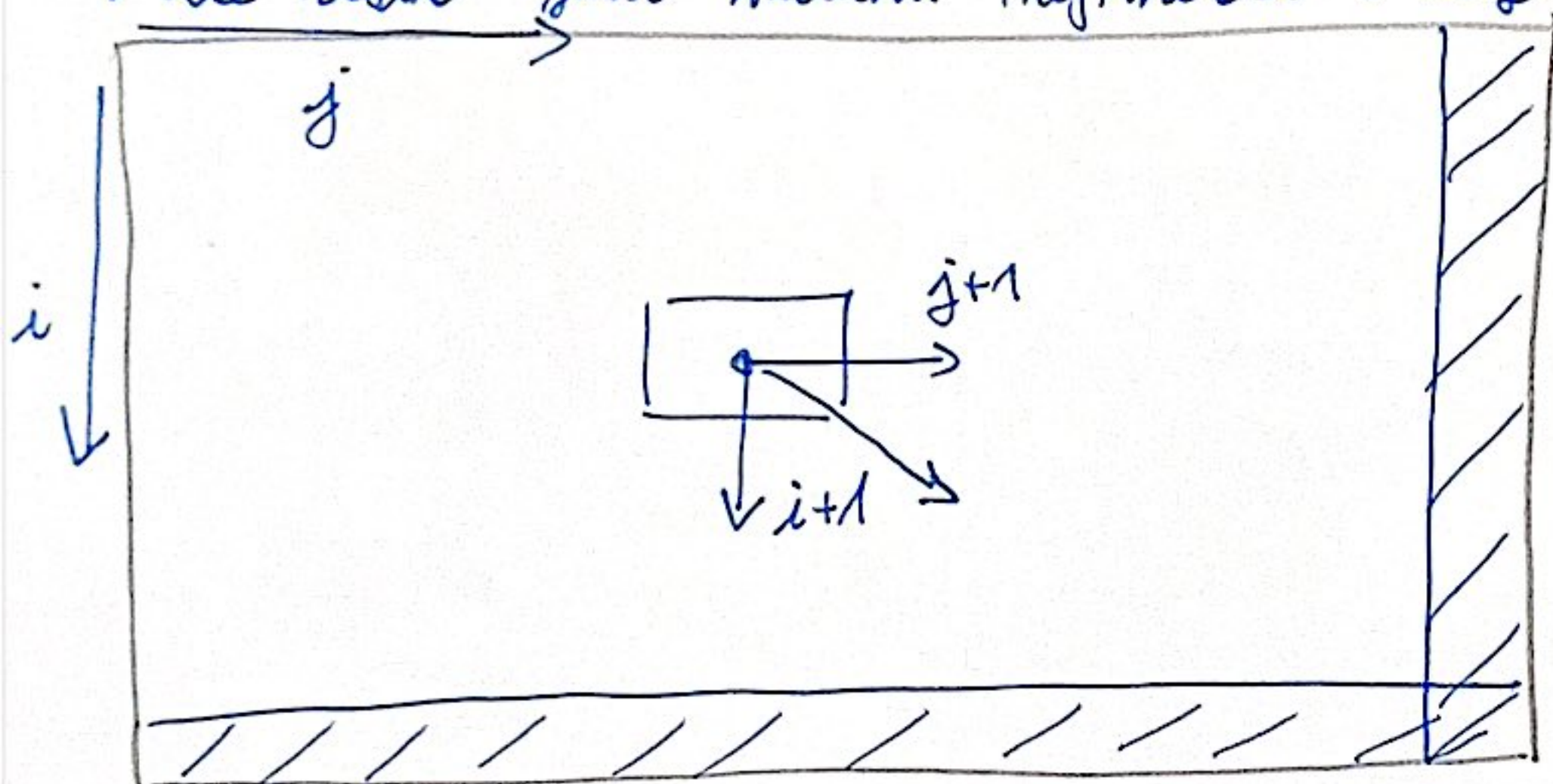
$\leftarrow$  smazání

$\rightarrow$  cachování

$\rightarrow$  proto se hečí cachování

$\rightarrow$  s cachováním složitost  $O(n \cdot m)$

$\rightarrow$  lze řešit jako hledání nejkratší cesty v DAG-u



vrcholy grafu reprezentují stav  $(i, j)$

od  $(1, 1)$  do  $(n+1, m+1)$  odpovídá posloupnosti editačních operací

$\rightarrow$  vložení z y  
 $\downarrow$  smazání z x  
 $\rightarrow$  : změna  $x_i$  a  $y_i$   
 $\rightarrow$  :  $x_i = x_j$

$\left. \begin{array}{l} \rightarrow \text{vložení z y} \\ \downarrow \text{smazání z x} \\ \rightarrow : \text{změna } x_i \text{ a } y_i \end{array} \right\} \text{ cena } 1$   
 $\rightarrow : x_i = x_j \quad \left. \vphantom{\begin{array}{l} \rightarrow \text{vložení z y} \\ \downarrow \text{smazání z x} \\ \rightarrow : \text{změna } x_i \text{ a } y_i \end{array}} \right\} \text{ cena } 0$

Pro  $i = n, n-1, \dots, 1$ :

Pro  $j = m, m-1, \dots, 1$ :

$L[i, j] \leftarrow \min(L[i+1, j+1] + \begin{cases} 0 & \text{jinak} \\ 1 & \text{pokud } x_i \neq x_j \end{cases},$

$L[i, j+1] + 1,$

$L[i+1, j] + 1)$



## 6.1) Princip dynamického programování

↳ rozdělení na menší podproblémy = stav

↳ pokud máme graf, co říká, který podproblém  
potřebuje výsledky kterých podproblémů

=> pokud graf bude DAG a lze řešit  
topologickým uspořádáním