- kdo je kdo
- konzultační hodiny: email
- zkouška – znalosti v rozsahu přednášky
- DÚ – dobrovolné, lze se na ně zeptat u zkoušky za účelem vylepšení známky až o jeden stupeň, v takovém případě nutno vyřešit dostatečně velký počet úloh.

  celkem 5

  DÚ 1 – do 23.10., měkčí do 30.10.

- úvod – o čem přednáška bude, o čem nebude

  plán – viz sylabus

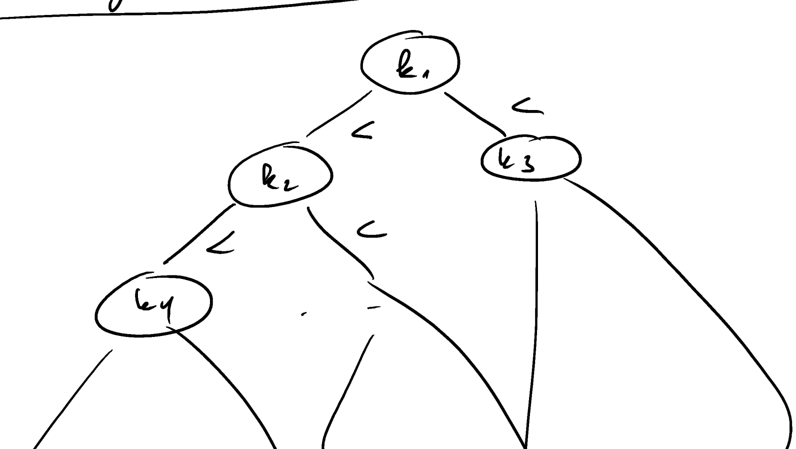
- <u>slovníkový problém</u>: ⟨klíč⟩ ⟨hodnota⟩

  ∈ U ... uspořádaná množina

chceme:
- insert (klíč, hodnota)
- delete (klíč)
- find (klíč) → hodnota

Zajímá nás <u>čas</u> na jednotlivé operace
→ počet elementárních operací jako aritmetické operace, porovnání, skok, následování pointeru, ...

<u>binární vyhledávací stromy</u>



- doba vyhledání ~ hloubka stromu

→ minimalizace hloubky - vyvažování

↙ systematicky AVL stromy apod.

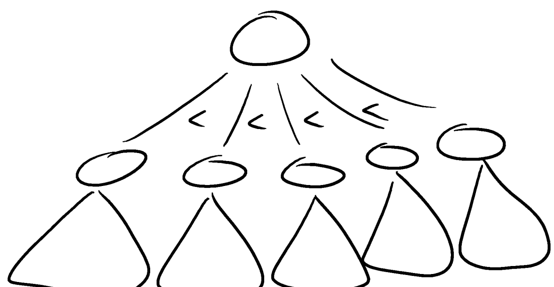
↓ (a,b)-stromy

↘ "dynamicky" Splay stromy

(a,b)-stromy

$a, b \in \mathbb{Z}$ , $a \geq 2$ , $b \geq 2a - 1$

- strom, kde každý vnitřní uzel má alespoň $\underline{a}$ a nejvýše $\underline{b}$ synů




- hodnoty jsou v listech
- vnitřní uzly obsahují maxima svých podstromů

(neplatí pro kořen a pro listy)

- hodnoty jsou v listech
- vnitřní uzly obsahují maxima svých podstromů → pole klíčů / spojový seznam.

⟹ (a,b)-strom hloubky $\underline{d}$ má alespoň $a^{d-1}$ a nanejvýš $b^d$ listů.

⟹ strom obsahující $\underline{n}$ hodnot (listů) má hloubku $d$, kde $\log_b n \leq d \leq 1 + \log_a n$.

- a,b v závislosti na použití, např. (2,3)-stromy

→ B-stromy   a,b ≈ velikost bloku na disku

→ B-stromy   $a, b \approx$ velikost bloku na disku

• **find (k)**

• projdi strom od kořene, jdi vždy do podstromu, kde by $k$ mohlo být.

• **insert (k, val)**

• najdi vrchol $v$, pod který patří nový list

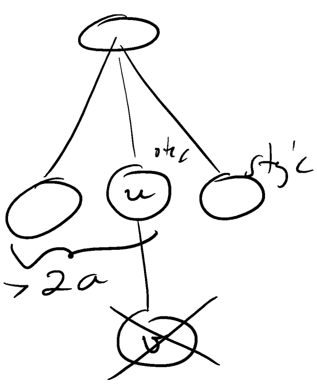
• pokud $v$ má $< b$ synů, přidej nový list $(k, v)$ → done

• pokud $v$ má $b$ synů, rozštěp $v$ na dva nové vrcholy, každý má $\frac{b+1}{2}$ synů (→ vytvoř nový vrchol $v'$, který dostane $\frac{b+1}{2}$ menších synů $v$)
→ rekurzivně vlož $v'$ do otce $v$.

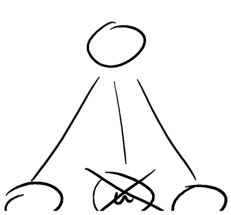
• **delete (k)**

• (rekurzivně od listu $v$ obsahující klíč $k$)

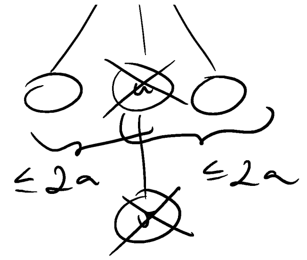
• pokud otec $u$ mazaného vrcholu $v$ obsahuje $> a$ synů, vymaž z $u$ informaci o $v$ → done




• pokud otec $u$ společně buď se synem levým nebo pravým sourozencem (strýčky $v$) obsahuje $> 2a$ synů, přesuň jednoho syna (bratrance) k $u$ a smaž $v$. Zaktualizuj informace v otci $u$ o $u$ a sourozenci (dotčených strýčcích)



• pokud otec $u$ společně ani se synem levým ani se synem pravým sourozencem nemá víc než $2a$ synů, přesuň syny $u$ do jednoho ze strýčků a




nemá víc než 2a synů, přesuň syny $\underline{u}$ do jednoho ze strýčků a rekurzivně smaž $\underline{u}$. (zaktualizuj informaci o strýčcích v otci $\underline{u}$)

- Find, Insert, delete ... čas na operaci $O(\log n)$

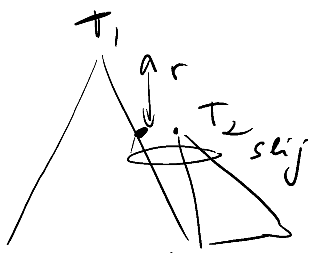
  (za předpokladu, že slití a rozštěpení vrcholu trvá čas $O(1)$)

## Operace Join, Split

Join $(T_1, T_2)$ – spojí stromy $T_1$ a $T_2$ za předpokladu, že $\max T_1 < \min T_2$

maximální hodnota v $T$

alg: pokud $\underbrace{\text{výška } T_1}_{d(T_1)} > \underbrace{\text{výška } T_2}_{d(T_2)}$ pak:

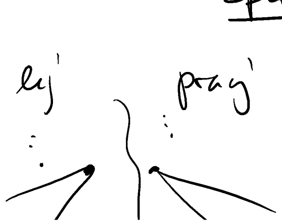



$r = d(T_1) - d(T_2)$

syny kořene $T_2$ přidej najednou k synům posledního vrcholu na hladině $\underline{r}$ ve stromě $T_1$. Pokud tento vrchol bude mít po slití více než b synů, rozštěp ho na dva a rekurzivně přidej nově vzniklý vrchol jako při operaci Insert

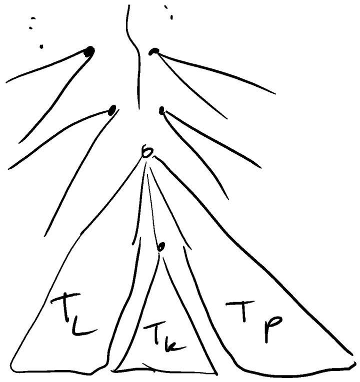
– pokud výška $(T_2)$ > výška $(T_1)$ postupuj analogicky jako v předchozím případě.

Split $(T, k)$: rozdělí strom $T$ na dva stromy, jeden obsahující klíče $< k$ a druhý obsahující klíče $\geq k$.




alg:

... pro levý




alg.:

- máme dva zásobníky, jeden pro levé stromy, druhý pro pravé stromy.

Postupujeme jako při find(k). Při průchodu vrcholem, vrchol rozštěpíme na tři podstromy: $T_L$, $T_k$, $T_p$, kde $T_L$ obsahuje podstromy vrcholu s hodnotami menšími než $k$, $T_k$ je podstrom vrcholu, kam pokračujeme při hledání $k$, a $T_p$ sestává ze zbylých podstromů vrcholu, tj., obsahují hodnoty $> k$.

- $T_L$ dáme na levý zásobník, $T_p$ na pravý a pokračujeme v hledání v podstromu $k$.

- až dojdeme do listu, z vrcholů na levém zásobníku pospojujeme pomocí operace Join výsledný strom s vrcholy $< k$, a stejně tak pospojováním pravého zásobníku dostaneme výsledný strom s vrcholy $\geq k$.

  (Nutno spojovat stromy odshora zásobníku, abychom dosáhli optimální časové složitosti.)

list přidáme na levý zásobník

- časová složitost Join$(T_1, T_2)$ je úměrná rozdílu výšek $T_1$ & $T_2$

$\Rightarrow$ • časová složitost Split$(T, k)$ je úměrná výšce $T$ $\rightarrow O(\log n)$

Operace Ord(i): vrátí v pořadí i-tý prvek

, ...   σ /

Operace Ord(i): vrátí v pořadí i-tý prvek ve stromě.

– pokud v každém vrcholu udržuji aktuální počet listů v daném podstromě, lze operaci Ord(k) provést v čase $O(\log n)$

- počet štěpení a slučování vrcholů při <u>m</u> insertech a <u>l</u> deletech ... $O(m + l + \log n)$

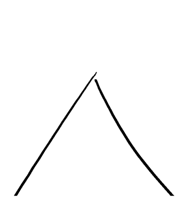
  předpoklad $b \geq 2a$.

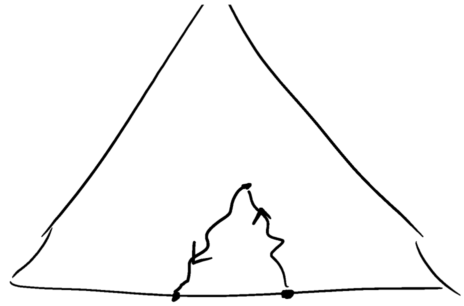
  → amortizovaně $O(1)$ štěpení / slučování za operaci

- paralelní verze: $b \geq 2a$
  - při insertu při hledání od kořene, rozštěp každý vrchol s <u>b</u> syny (preventivní štěpení)
  - při deletu při hledání od kořene uprav každý vrchol s <u>a</u> syny buď přidáním syna ze sourozence, nebo sloučím se sourozencem.

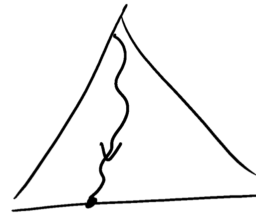
- A-sort: setřídí posloupnost tak, že je vloží do (a,b)-stromu a pak je vypíše během průchodu do hloubky
  - při vkládání hledáme pozici pro další prvek ne od kořene, ale od listu, kam jsme vložili naposledy (strom s "prstem" – ukazatelem na poslední použitý list)

vstupní posloupnost: $x_1, x_2, \dots x_i, x_{i+1}, \dots$





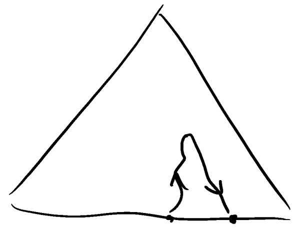
místo



$x_{i+1}$ $x_i$ $x_{i+1}$

$x_i > x_{i+1}$ ... cesta nahoru maximálně

$$\log_a |\{j \le i;\ x_{i+1} < x_j\}|$$



$x_i$ $x_{i+1}$

$x_i < x_{i+1}$ ... cesta nahoru maximálně

$$\log_a |\{j \le i,\ x_i < x_j\}|$$

$$\Rightarrow \text{celkový čas} \le \underbrace{2 \cdot \sum_{i=1}^{n} \log_a |\{j \le i,\ x_i < x_j\}|}_{\text{hledání ve stromě}}$$

$+ O(n)$ ← výpis posloupnosti

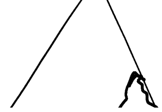
z konkávnosti fce log

$$\le 2 \cdot n \cdot \log_a \frac{\sum_{i=1}^{n} |\{j \le i,\ x_i < x_j\}|}{n} + n = O\left(n \log \frac{F}{n}\right).$$

$$F = \sum_{i=1}^{n} |\{j \le i;\ x_i < x_j\}|$$ ← celkový počet inverzí v původní posloupnosti.

$$0 \le F \le n^2$$

Alternativně - vždy hledám od nejpravějšího listu



→ cesta nahoru

$$\le 1 + \log_a \{j < i+1,\ x_j > x_i\}$$


→ cesta nahoru

$\leq 1 + \log_a \{ j < i+1,\ x_j > x_i \}$

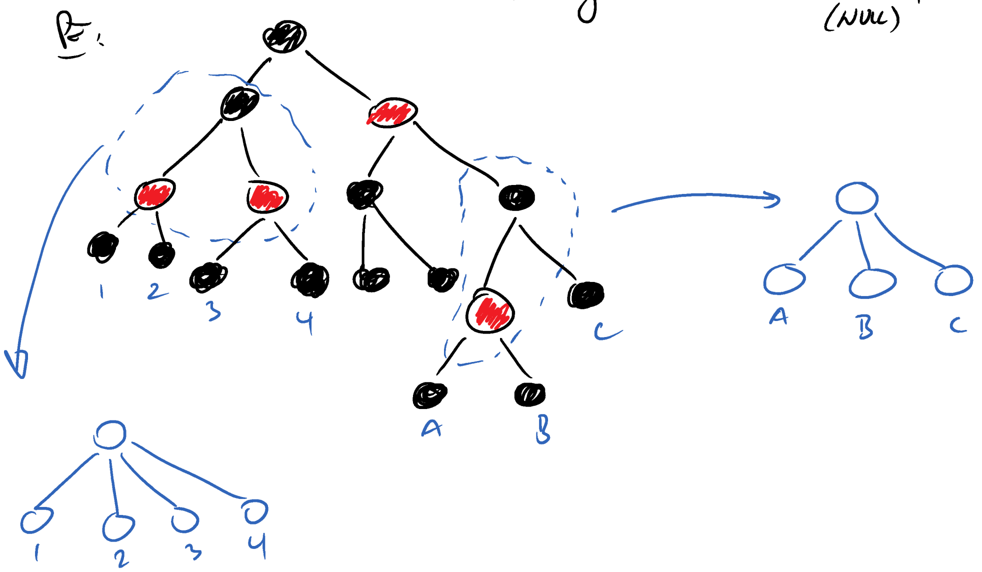
## Červeno-černé stromy

- binární vyhledávací strom
- každý uzel je buď obarven černě nebo červeně.
- červené uzly mohou být syny pouze černých uzlů (*)
- na každé cestě do listu stejný počet černých vrcholů (**)

→ hodnoty uloženy ve vnitřních vrcholech

(listy lze nahradit NIL pointerem)
(NULL)

Př.:

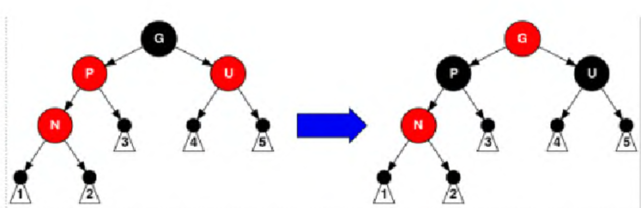



(*)+(**) ⇒ Červeno-černé stromy jsou ekvivalentní (2,4)-stromům

## Vyvažování při vložení (INSERT)

- vložen červený uzel N

1)

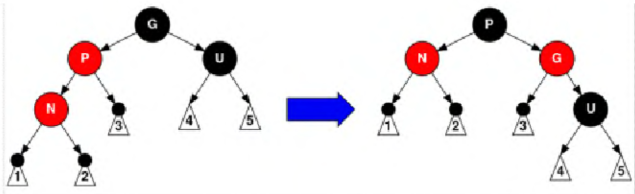



→ problém s příliš mnoha červenými vrcholy se přenese ke otci

2.

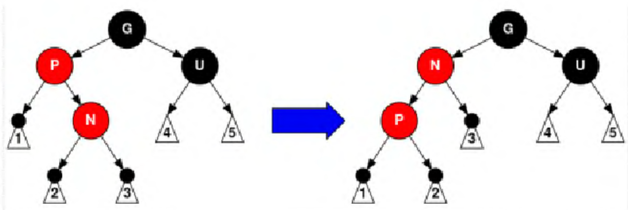

k otci

2)




Špatné rozložení červené vrcholy se přerotují!

3)




{obrázky z wikipedie}

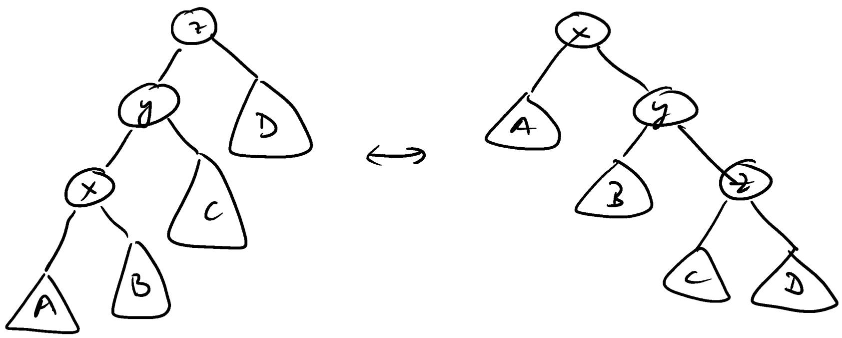
- DELETE – podobný
- používají se v knihovnách, např. STL v C++

---

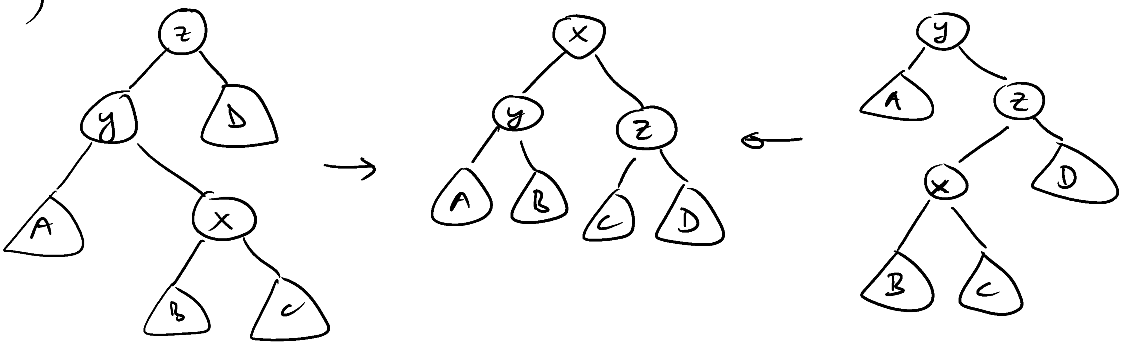
## Splay stromy

- binární vyhledávací stromy, samoupravující
  - cena za operaci až $O(n)$, ale cena za $m$ operací nejvýše $O(m \log n + n \log n)$ kde $n$ je počet vložených prvků
- operace splay(x), přesune prvek x do kořene pomocí rotací
- po každé operaci: find, insert, ... aplikujeme operaci splay.
- dvojité rotace

a)

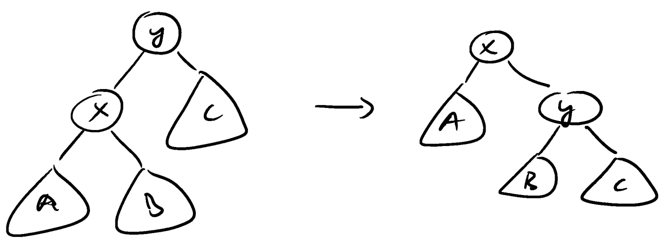


b)



• rotace u kořene

c)



amortizovaný čas operace $t$:

$$t = a + \Phi(T') - \Phi(T)$$

skutečný čas

potenciál před operací

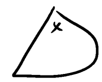
potenciál po operaci

$\Phi(T)$ .. potenciál stromu $T$

$$\Phi(T) = \sum_{\substack{x \text{ vrchol} \\ v\ T}} r(x)$$

kde $r(x) = \log$ (počet vrcholů v podstromu $x$)

tedy $r(x) = \log_2$ (počet vrcholů v T v podstromu $x$) ... "rank"




- Čas na $\underline{m}$ operací:

strom po i-té operaci

$$\sum_{i=1}^{m} t_i = \sum_{i=1}^{m} a_i + \Phi(T_i) - \Phi(T_{i-1}) =$$

$$= \left(\sum_{i=1}^{m} a_i\right) + \Phi(T_m) - \Phi(T_0)$$

$$\Rightarrow \quad \sum_{i=1}^{m} a_i = \sum_{i=1}^{m} t_i + \Phi(T_0) - \Phi(T_m)$$

$\Rightarrow$ Čas na $\underline{m}$ operací $m \cdot O(\log n) + O(n \cdot \log n)$.

- <u>amortizovaný čas rotace</u> a), b) $\leq 3(r'(x) - r(x))$

  c) $\leq 3(r'(x) - r(x)) + 1$

$r'$ ... rank po operaci

$r$ ... rank před operací

<u>Dk a)</u>: pouze vrcholy $x, y, z$ mění svůj rank

$\Rightarrow$ amortizovaný čas $t \leq 2 + r'(x) - r(x) + r'(y) - r(y) + r'(z) - r(z)$

$$= 2 - r(x) + r'(y) - r(y) + r'(z)$$

$$\left.\begin{array}{l} r'(y) \leq r'(x) \\ r(y) \geq r(x) \end{array}\right\} \Rightarrow \leq 2 - r(x) + r'(x) - r(x) + r'(z)$$

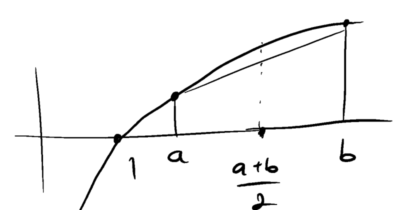
$$= 2 + r'(x) - 2r(x) + r'(z) = (*)$$

<u>Pozorování</u>: $2 \leq 2r'(x) - r(x) - r'(z)$

$$r'(x) = \log_2 |A| + |B| + |C| + |D| + 3$$

$$r(x) = \log_2 \overbrace{|A| + |B| + 1}^{a}$$

$$r'(z) = \log_2 \underbrace{|C| + |D| + 1}_{b}$$




$$\log_2 a + \log_2 b \leq 2 \log \frac{a+b}{2}$$

$$\log_2 a + \log_2 b \leq 2 \log \frac{a+b}{2}$$

$$r'(x) = \log(a+b+1) \geq \left(\log \frac{a+b}{2}\right) + 1$$

$$\Rightarrow \quad 2r'(x) - r(x) - r'(z) \geq 2$$

$$t \leq (*) \leq 3r'(x) - 3r(x)$$

Dk b):

$$t \leq 2 + r'(x) - r(x) + r'(y) - r(y) + r'(z) - r(z)$$
$$= 2 - r(x) + r'(y) - r(y) + r'(z)$$
$$\leq 2 - 2r(x) + r'(y) + r'(z) = (*)$$

Pozorování: $2 \leq 2r'(x) - r'(y) - r'(z)$

Dk: stejný jako výše

$$(*) \leq 2r'(x) - 2r(x) = 2(r'(x) - r(x))$$

Dk c):

$$t \leq 1 + r'(x) - r(x) + r'(y) - r(y)$$
$$\leq 1 + r'(x) - r(x)$$
$$\leq 1 + 3(r'(x) - r(x))$$

- $r'(y) < r(y)$
- $r'(x) - r(x) > 0$

$\Rightarrow$ součet amortizovaných časů pro rotace, které přenesou $\underline{x}$ do kořene je $\leq 1 + 3(r(v) - r(x))$

původní kořen stromu T

Dk. $$a = t + \Phi(T') - \Phi(T) = \sum_i t_i + \Phi(T_i) - \Phi(T_{i-1})$$

amort. čas; skutečný čas; i-tá rotace; po i-té rotaci; před i-tou rotací

$$\leq \sum_i a_i$$

amort. čas i-té rotace

$$\leq \sum_i 3(r_i(x) - r_{i-1}(x)) + 1$$

$$\le \sum_i 3\left(r_i(x) - r_{i-1}(x)\right) + 1$$

rank i po i-té rotaci

za poslední jednoduchou rotaci typu c)

$$= 3\left(r'(x) - r(x)\right) + 1$$

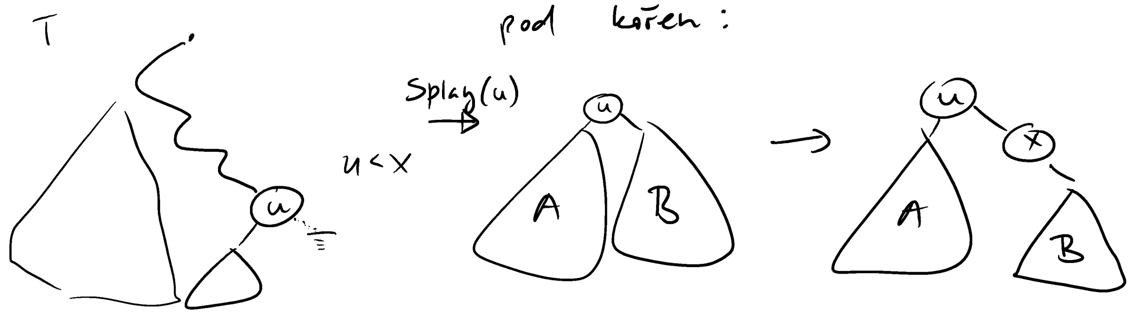
konečný rank x = rank kořene

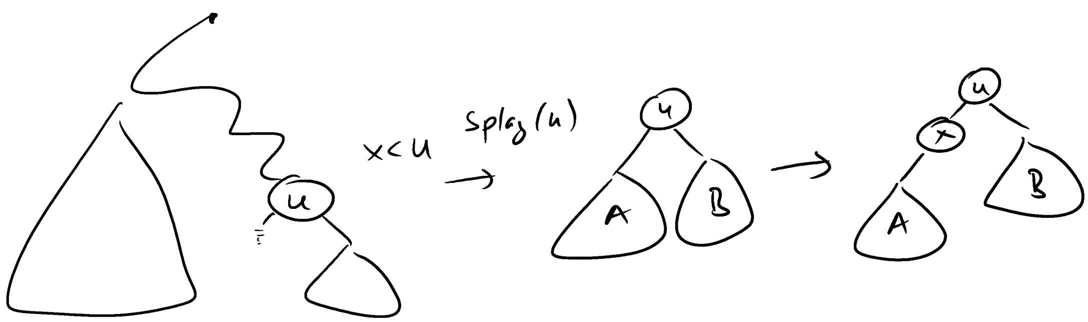
původní rank x

$\forall u \in T,\ r(u) \le \log_2 n$

$\Rightarrow$ amortizovaný čas na splay $\le 1 + 3\log_2 n$.

- find (x) : najdi x, proveď splay (x)
- Insert (x) : najdi x; nechť u je poslední vrchol podél cesty k chybějícímu x. Proveď splay (u), vlož x doleva nebo doprava pod kořen:



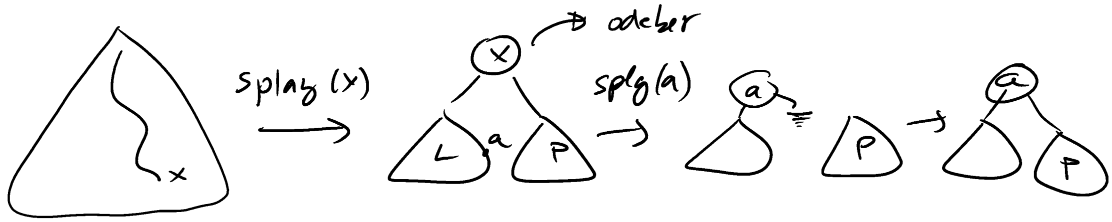

Pozorování: nechť $a, b \in T$, $a < b$ jsou takové, že $\forall c \in T$ $c \notin (a,b)$, $x \in (a,b)$

pak u je buď a nebo b.

Dk: a i b musí být navštíveny po cestě k x, pak $u$ je buď $\underline{a}$ nebo $\underline{b}$.

Dk: $\underline{a}$ i $\underline{b}$ musí být nastřeny po cestě k $\underline{x}$, jinak by jejich vyhledávání selhalo, protože $\underline{a}$ i $\underline{b}$ následují stejnou cestu jako $\underline{x}$. ∎

- delete$(x)$: najdi $\underline{x}$, splay$(x)$, odeber $x$, pak získám dva nepropojené stromy $L$ a $P$, najdi nejvyšší prvek $\underline{a}$ v $L$, splay$(a)$, připoj $P$ pod $\underline{a}$.

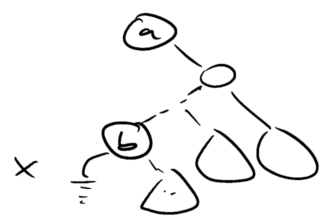



$a$ ... "nejpravější" vrchol v $\underline{L}$.
$\parallel$
$\max(L)$

→ všechny operace mají amortizovanou složitost $O(\log n)$.

---

Pozn: Oprava z hodiny $(a,b)$ t.ž. $a<x<b$ a $\nexists c \in T$ $a<c<b$ může být tíž

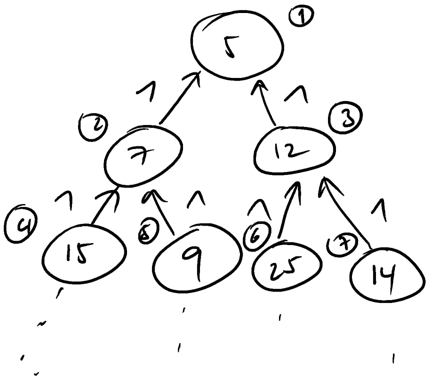



---

## Haldy

operace
- Insert$(x)$ ... vloží $\underline{x}$ do haldy
- Min ... vrátí nejmenší prvek haldy
- Delete-min ... odebere nejmenší prvek z haldy

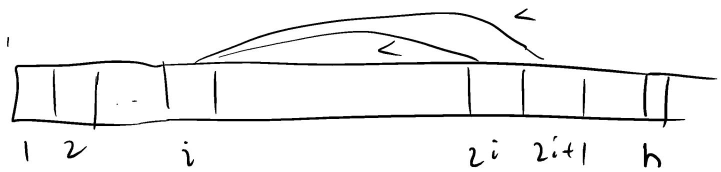

$i$-tý vrchol má za syny




i-tý vrchol má za syny vrcholy $2i$ a $2i+1$

→ lze jednoduše uložit do pole

... <u>regulární halda</u>




Insert(x) ... přidám na konec pole jako $n+1$ prvek a bublám ho směrem ke kořeni, dokud je porušena podmínka, že otec je menší než syn

Min ... vrátí první prvek v poli

Delete-min ... poslední prvek pole přesune do prvního a bublá s menším ze synů směrem dolů k listům, dokud otec je větší než syn.

<u>čas na operace:</u>

| | |
|---|---|
| Insert | $O(\log n)$ |
| Min | $O(1)$ |
| Delete-min | $O(\log n)$ |

chceme rychlejší operaci Insert → <u>binomiální haldy</u>

binomiální halda – soubor haldově uspořádaných stromů velikosti $2^k$

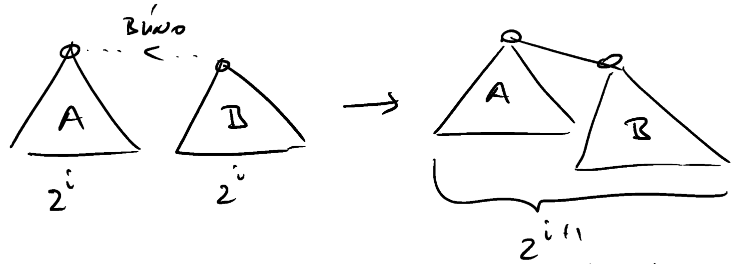
. pamatujeme si, který z těchto stromů obsahuje minimální prvek

zbrklá varianta – nejvýše jeden strom dané velikosti

líná varianta – bez omezení na počet stromů stejné velikosti

→ Insert(x) – přidá nový haldový strom velikosti 1

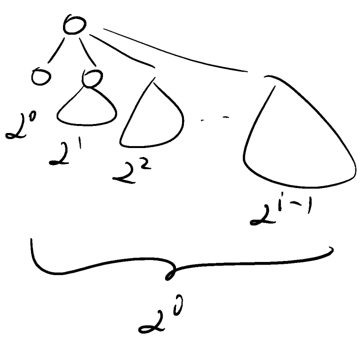
↳ Insert(x) – přidá nový haldový strom velikosti 1 obsahující x.
Dokud existují dva stromy A, B stejné velikosti, spoj je do jednoho:




• průběžně aktualizujeme ukazatel na strom s minimálním prvkem

Min ... vrátí $\underbrace{\text{hodnotu}}$ kořene stromu s minimálním prvkem
= minimální prvek

Delete-min ... odřízneme kořen stromu s minimálním prvkem. Pokud tento strom měl $2^i$ vrcholů, rozpadne se na $i$ stromů velikosti $2^j$, $j = 0, \dots, i-1$.




• Tyto stromy přidáme do seznamu a podobně jako při Insert, slučujeme stromy stejné velikosti, dokud máme dva stromy stejné velikosti.

• Čas na operaci: Insert $O(\log n)$
Min $O(1)$
Delete-min $O(\log n)$

→ stejné jako předtím, ale $n$ vložení do prázdné haldy trvá pouze $O(n)$
→ amortizovaně $O(1)$ na Insert, pokud nepoužíváme Delete-min.

pokud neprovádíme Delete-min.

Slijí stromy
velikosti $2^i$ proběhne
$\frac{n}{2^i}$ - krát $\Rightarrow$
celkem $\sum_{i=0}^{\log n} \frac{n}{2^i} \leq n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$

<u>líná varianta</u>: spojový seznam stromů velikostí $2^k$,
stejná velikost se může opakovat.
+ ukazatel na nejmenší prvek

Insert (x) ... vloží nový strom velikosti 1 s $x$,
zaktualizuje odkaz na nejmenší

Min ... vrátí nejmenší prvek

Delete-min ... stejně jako ve striktní variantě,
odřízneme kořen s nejmenším prvkem,
podstromy přidáme do seznamu
a slévame stromy stejné velikosti,
dokud to lze.
(Nejvýše $\log_2 n$ různých velikostí,
vytvořím si pole, kde i-tá položka
ukazuje na strom velikosti $2^i$,
a pomocí tohoto pole stromy slévám.)

<u>Čas na operaci</u> Insert (x) ... $O(1)$
Min ... $O(1)$
Delete-min $O(n)$

amortizovaně, ale Delete-min $O(\log n)$ čas

potenciál $\Phi(T) = C \cdot$ počet stromů v haldě

<u>amortizovaně</u>: Insert $\leq C + \underbrace{\Phi(T') - \Phi(T)}$

$C$ ... vhodná konstanta

$= C$ ✓

$= O(1)$

min $= O(1)$

Delete-min $\leq C \cdot \#\text{stromů} + \Phi(T') - \Phi(T)$

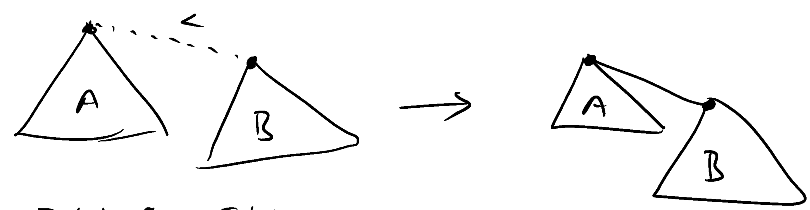
$\leq C' \cdot \log n$

$\leq C \log n = O(\log n)$

## Fibonacci halda

nové:

- Decrease-key$(x, \Delta)$ ... sníží hodnotu prvku $x$ o $\Delta$.
  - → amortizovaně $O(1)$
- podobná jako líná binomiální halda, stromy nemusí mít velikost mocniny 2, tj. $2^i$.
  - → řád stromu = počet synů kořene
  - → slíváme stromy stejného řádu




řád $i$ řád $i$ → řád $i+1$

→ Fibonacciho halda - spojový seznam haldových stromů

- Insert, min, Delete-min jako u líných binom. hald
- Decrease-key$(x, \Delta)$
  - Snížíme hodnotu $x$ o $\Delta$, pokud hodnota $x$ je pořád vyšší než u otce → hotovo
  - pokud hodnota $x$ klesne pod hodnotu otce

→ odtrhni podstrom x a zařaď ho do seznamu stromů. Pokud otec x již takto přišel o jeden podstrom (otec je označený) a otec x není kořen haldového stromu, rekurzivně odtrhni též otce x.

(Kořeny vkládaných stromů odznačíme.)

⇒ uvnitř stromu může vrchol přijít pouze o jednoho syna, pak je sám odtržen

- kořen může přijít o libovolně mnoho synů

<u>implementace</u>: vrchol si musí pamatovat počet svých potomků, kteří jsou uspořádáni ve spojovém seznamu, a musí si pamatovat zda již nepřišel o nějakého potomka.

## Struktura stromů:

Fibonacciho čísla

$$F_0 = 0$$
$$F_1 = 1$$
$$F_2 = 1 \qquad F_n = F_{n-1} + F_{n-2}$$
$$F_3 = 2$$
$$F_4 = 3$$
$$F_5 = 5$$
$$\vdots$$

<u>Pozorování</u>: $\sum_{i=1}^{n} F_i = F_{n+2} - 1$

<u>Dk</u>: indukce na $n$. $n=1,2$ triv.

$n \to n+1$

$$\sum_{i=1}^{n} F_i + F_{n+1} = F_{n+2} - 1 + F_{n+1} = F_{n+3} - 1 \quad \blacksquare$$

Pozorování: $\forall n \geq 3 \quad F_n \geq \left(\frac{5}{4}\right)^n$

Dk: indukce na n. $n=3$ ✓

$n \to n+1$

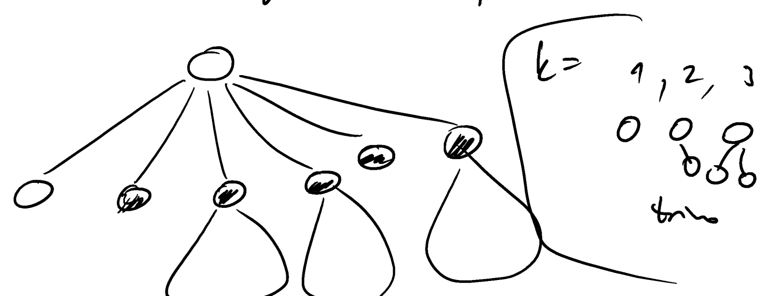
$$F_{n+1} = F_n + F_{n-1} \geq \left(\frac{5}{4}\right)^n + \left(\frac{5}{4}\right)^{n-1} = \left(\frac{5}{4}\right)^n + \frac{4}{5}\cdot\left(\frac{5}{4}\right)^n = 1.75\cdot\left(\frac{5}{4}\right)^n \geq \left(\frac{5}{4}\right)^{n+1}$$

- lze vylepšit na $F_n \geq \left(\frac{1+\sqrt{5}}{2}\right)^n \approx 1.68...^n$

"zlatý řez"

- strom řádu k má alespoň $F_{k+1}$ prvků.

nejhorší případ:




řád $\geq 0 \quad \geq 0 \quad \geq 1 \quad \geq 2 \quad \dots \geq k-2$

- velikosti podstromů to splňují $\Rightarrow$ splňuje i otec:

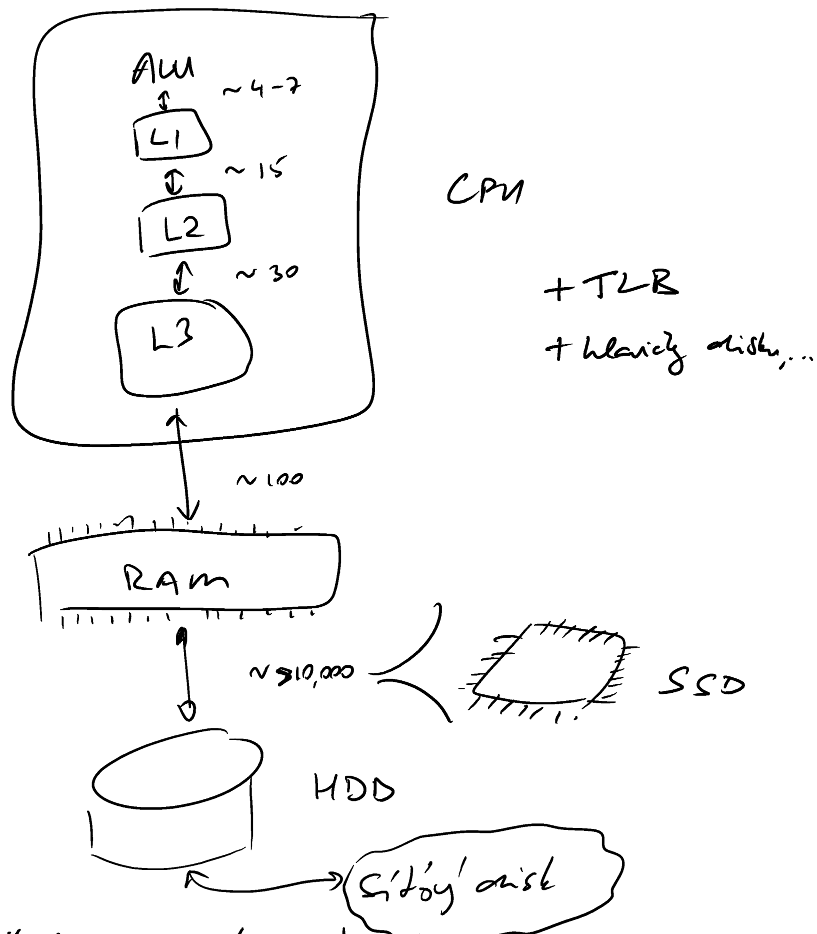
$$\geq 1 + F_1 + \sum_{i=1}^{k-1} F_i = 1 + 1 + F_{k+2} - 1 = F_{k+1} + 1$$

kořen, nejlevější syn

amortizovaná analýza:

potenciál $\Phi(H) = \#\text{stromů} + 2\cdot\#\text{označených vrcholů}$

Paměťové hierarchie

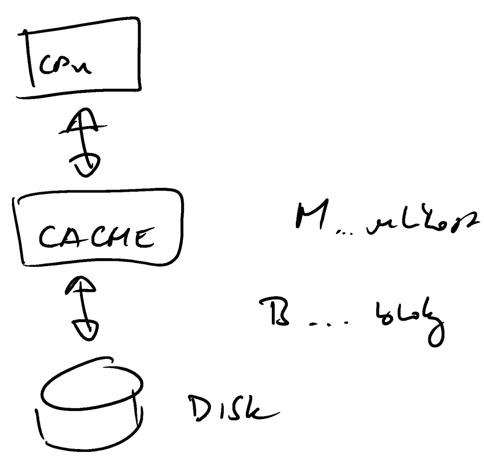



M ... velikost paměti (cache)

B ... velikost přenášených bloků

→ M/B počet bloků v paměti

<u>model externí paměti</u>

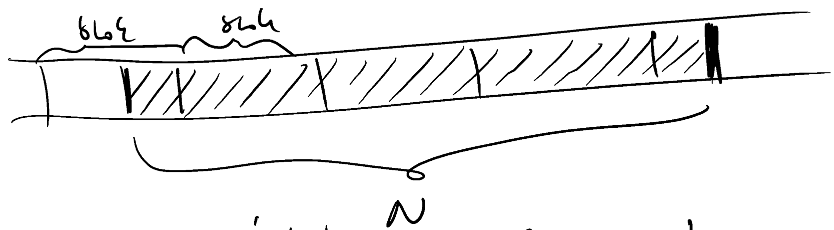



- u algoritmů měříme počet přenesených bloků
- soustředíme se na jednu úroveň a zbylé ignorujeme
  - → algoritmus optimalizovaný pro konkrétní úroveň

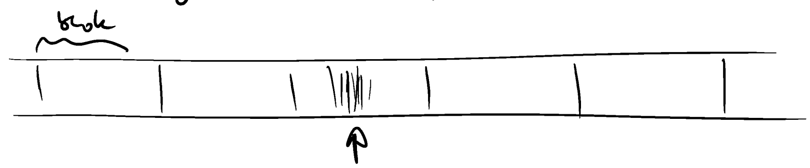
Př.: • Přečtení souvislého kusu dat délky $\underline{N}$.

- $\lceil M/B \rceil + 1$ přenos

• $\lceil M/B \rceil + 1$ přenos

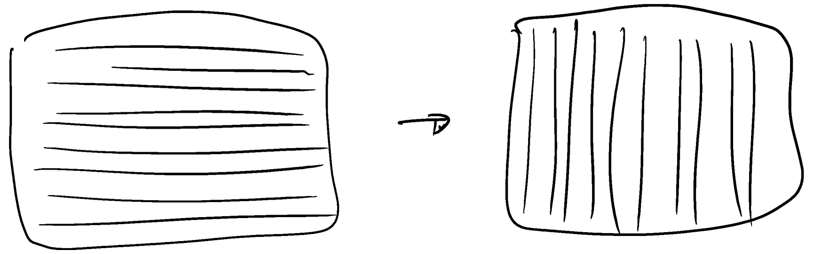



• binární vyhledávání v poli velikosti $N$



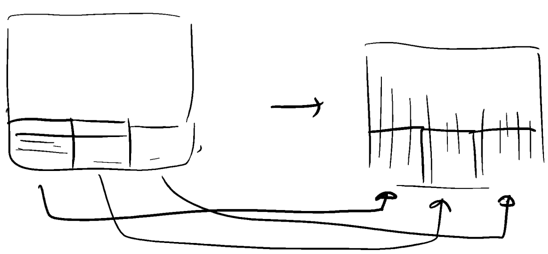

• $\log N - \log B$ přenosů

• transpozice matice



→ naivní algoritmus - $N^2$ přenosů (pokud $\frac{M}{B} < N$)

• pokud $M > B^2$ lze udělat algoritmus s $N^2/B$ přenosy (optimální)



– transponuj po podmaticích velikosti $B \times B$. $A_2$

## Cache Oblivious Analýza ("Analýza ignorující cache")

• analyzujeme v neznámých $M$ a $B$
chceme alg., který se bude chovat optimálně pro každou volbu $M$ a $B$,
ale algoritmus nezávisí na $M$ a $B$, tj.
$M$ a $B$ se v algoritmu nevyskytují.

⇒ na každé úrovni paměťové hierarchie
optimální počet přenosů

Př.: • přečtení souvislého kusu dat (viz výše)
- algoritmus nezávisí na $M$ a $B$
- na každé úrovni $\lceil N/B \rceil + 1$ přenosů
- lépe to nejde

• transpozice matice – algoritmus (A2)
závisí na $B$ → není dobrý

transpozice matice:

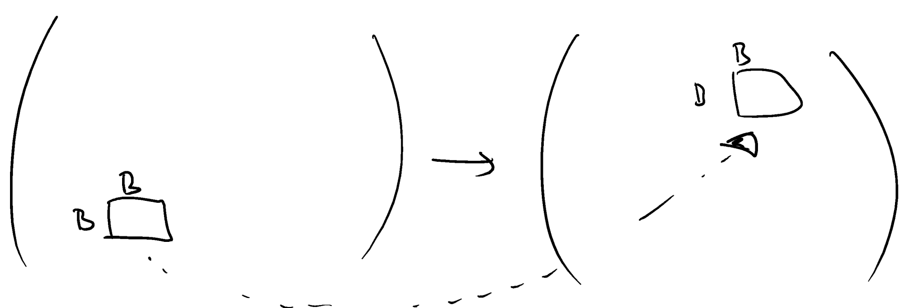
$$\begin{array}{c c} \frac{1}{2} & \multirow{2}{*}{$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right)$} \\ \frac{1}{2} & \\ & \frac{1}{2} \quad \frac{1}{2} \end{array} \longrightarrow \left(\begin{array}{c|c} A_{11}^T & A_{21}^T \\ \hline A_{12}^T & A_{22}^T \end{array}\right)$$

rekurzivně opakuj

\# operací $O(n^2)$

\# IO $O(n^2/B)$ předpoklad $M \geq B^2$ („tall cache assumption")

Dk.:

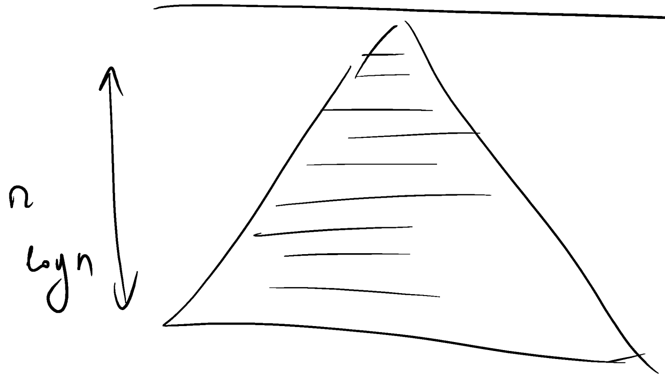



$O(B)$ IO operací, jakmile se rekurze dostane na velikost matice $\ell \times \ell$
$\ell = O(B)$

$\frac{n^2}{B^2}$ takových podmatic

$\longrightarrow O\left(\frac{n^2}{B^2} \cdot B\right) = O\left(\frac{n^2}{B}\right)$ IO operací ∎

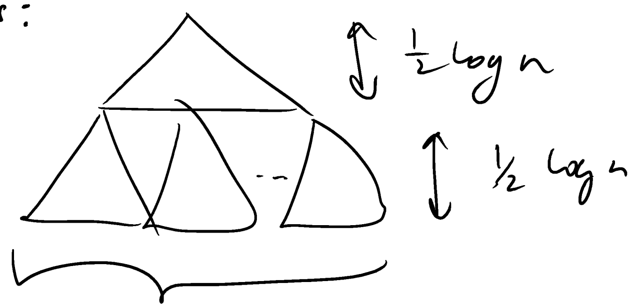
• van Emde Boas rozložení stromu (statický)




úplný binární strom

• při klasickém rozložení jako u regulární haldy v poli $\log n - \log B$ IO operací

• van Emde Boas:




$\sqrt{n}$ stromů, každý velikosti $\sqrt{n}$

Strom velikosti $< B$ se zpracuje za cenu $O(1)$ IO operací.

$\longrightarrow$ velikost stromu mezi $\sqrt{B}$ a $B$.

$\rightarrow O(1)$ IO operací

\# operací $O(\log n)$

\# IO operací $\frac{\log n}{\log \sqrt{B}} = 2 \log_B n$

• třídění: lze

\# operací $O(n \cdot \log n)$

\# IO operací $O\left(\frac{n}{B} \cdot \log_{\frac{M}{B}} \frac{n}{B}\right)$

za předpokladu $M \geq B^2$

• násobení matic, FFT, ... ✓

• optimální správa cache:

problémy: 1) neznámá budoucnost
2) asociativita cache

1) není problém:

Sleator – Tarjan (1985)

• LRU strategie vs OPT strategie

posloupnost přístupů $s_1, s_2, \dots s_N$

LRU má k dispozici $n_{LRU}$ stránek v cache
OPT má k dispozici $n_{OPT}$ stránek v cache

Věta: $\#$výpadků LRU $\leq \dfrac{n_{LRU}}{n_{LRU} - n_{OPT}} \cdot \#$výpadků OPT $+ n_{LRU}$

Pozorování: pokud v čase $t_1$ a $t_2$, $t_1 < t_2$, LRU má výpadek na té samé stránce, $s_{t_1} = s_{t_2}$, pak mezi $t_1$ a $t_2$, posloupnost $s$ přistupuje k $n_{LRU}$ různým stránkám (tj. $|\{s_i,\ t_1 < i < t_2\}| \geq n_{LRU}$)

Dk: v čase $t_1+1$ je $s_{t_1}$ v cache u LRU, aby z ní vypadla, musí se přistoupit k $n_{LRU}$ různým stránkám po $t_1$ ⇒ tvrzení ∎

Dk: rozsekejme $s_1, s_2, \dots$ na kusy, kde v každém kusu nastane v LRU $n_{LRU}$ výpadků. až na poslední

• v každém kusu se přistupuje k alespoň $n_{LRU}$ různým stránkám.

⇑

buď jsou všechny výpadky různé nebo se užije předchozí tvrzení

buď "jsou všechny výpadky různé"
nebo se užije předchozí tvrzení

⇒ OPT musí v daném úseku mít alespoň $n_{LRU} - n_{OPT}$ výpadků, neboť na začátku úseku má OPT v cache nejvýše $n_{OPT}$ stránek

$$\Rightarrow \quad \frac{\#\text{výpadků LRU}}{n_{LRU}} \leq \left\lceil \frac{\#\text{výpadků OPT}}{n_{LRU} - n_{OPT}} \right\rceil \blacksquare$$

⇒ Pokud $n_{LRU} = 2\, n_{OPT}$ pak LRU se dostane optimálního až na konstantu 2!

Alternativní důkaz:

$s_1, \dots s_N$ rozdělíme na kusy, kde v každém kuse je právě $n_{LRU}$ různých stránek.

⇒ LRU má v každém úseku $\leq n_{LRU}$ výpadků
OPT má v každém úseku $\geq n_{LRU} - n_{OPT}$ výpadků $\blacksquare$

---

ANKETA

## Hašování

Slovníkový problém ... universum $U$

$S \subseteq U$ $\qquad |S| = n$

chceme reprezentovat $S$

Operace - Find (MEMBER)
- Insert
- (DELETE)

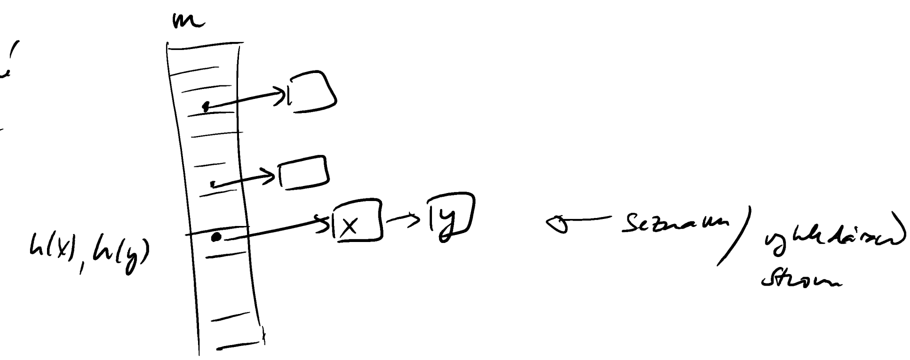
- triviální ... pole velikosti $U$, mapování 1-1
- lépe ... hašování do pole velikosti $\underline{m}$.

$$h: U \to \{1, \dots, m\} \quad \dots \text{ hašovací fce}$$

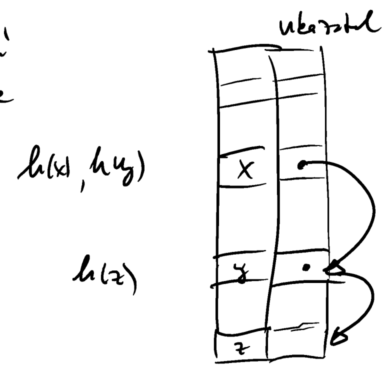
- prvek $x$ uložím na pozici $h(x)$
- může nastat <u>kolize</u>: $x, y \in S \quad x \neq y$, $h(x) = h(y)$

- <u>způsoby řešení</u>:

- separované řetězce




- srůstající řetězce

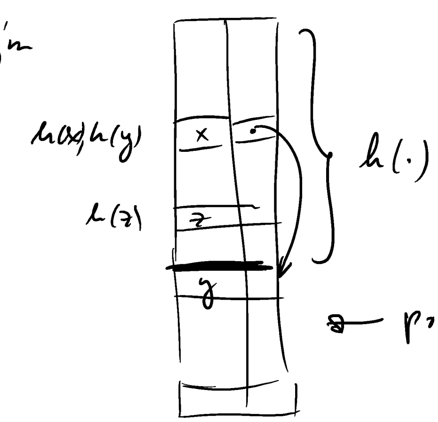



LICH ... nový prvek přidán na konec seznamu

EICH ... nový prvek přidán hned za první prvek seznamu

(seznam není obousměrně propojený)

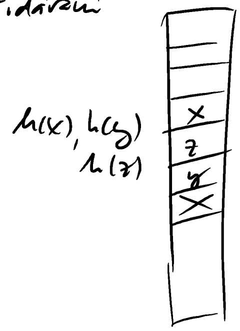
- s pomocným polem




← pomocná část, když dojde pomocná část, začnu ukládat přetékající prvky.

ukládat přetékající prvky do hlavního pole
→ srůstání

- lineární přidávání




najdu nejbližší volnou pozici k $h(x)$ a tam dám ukládaný prvek $\underline{x}$.

→ $h(x), h(x)+1, h(x)+2, h(x)+3, \dots$

- dvojité hašování – podobné jako lineární přidávání, ale zkoumám pozice

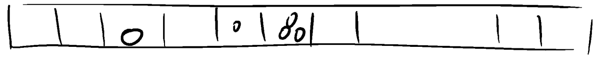
$$h_1(x) + i\, h_2(x) \quad , i = 0, 1, 2, \dots$$

$h_1, h_2$ jsou různé hašovací funkce

- je potřeba, aby $h_2(x)$ bylo nesoudělné s $\underline{m}$, což je pravda například když je $\underline{m}$ prvočíslo a $h_2(x) \in \{1, \dots, m-1\}$

DELETE – většinou problematický
- označování smazaných prvků a jejich znovupoužití při INSERT
→ pokud příliš mnoho označených prvků
→ přehašuj vše

- Balls & Bins – n míčků, n košů, každý míček hodím do náhodně zvoleného koše



$$Pr[\text{daný koš je prázdný}] = \left(1 - \frac{1}{n}\right)^n \underset{n \to \infty}{\approx} \frac{1}{e}$$

$$\Pr[\text{daný koš obsahuje } k \text{ míčků}] = \binom{n}{k} \frac{1}{n^k}\left(1-\frac{1}{n}\right)^{n-k}$$

$k \geq 1$

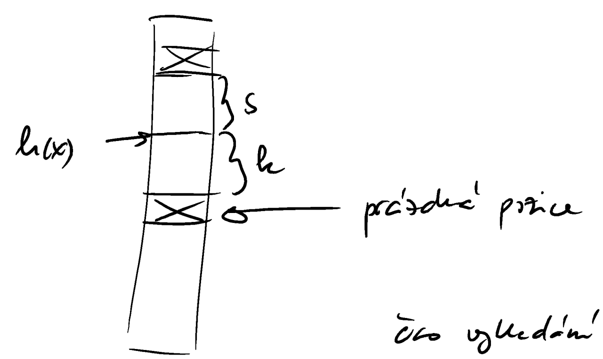
$$\underset{k \ll n}{\approx} \frac{n^k}{k!} \cdot \frac{1}{n^k} \cdot \frac{1}{e} = \frac{1}{e \cdot k!}$$

$\Rightarrow$ s velkou pravděpodobností, maximum v libovolném koši je $\Theta\left(\frac{\log n}{\log\log n}\right)$.

– odpovídá to situaci, kdy bychom při hašování brali náhodně zvolenou hašovací funkci

• <u>lineární přidávání</u>

<u>analýza</u> hledám prvek $x$, předpokládám $h$ distribuuje prvky zcela náhodně




čas vyhledání $\leq k$

$p_{k,s}$ ... pravděpodobnost, že nejbližší volná pozice je po $k$ prvcích po $h(x)$ a před $h(x)$ je alespoň $s$ prvků

$$p_{k,s} = \binom{n}{k+s} \cdot \left(\frac{k+s}{m}\right)^{k+s} = (*)$$

$k+s$ prvků z $n$, se musí namapovat do bloku velikosti $k+s$ z celkového počtu $m$ pozic.

$(*) \leq \; n^{k+s} \quad (k+s)^{k+s} \quad n^{k+s} \quad (1 \ldots)^{k+s}$

t určeno pevně $m$ pozic.

$$(*) \leq \frac{n^{k+s}}{(k+s)!} \cdot \frac{(k+s)^{k+s}}{m^{k+s}} = \frac{n^{k+s}}{m^{k+s}} \cdot \frac{(k+s)^{k+s}}{(k+s)!} = (**)$$

Stirlingova aproximace: $a! \approx \sqrt{2\pi a}\left(\frac{a}{e}\right)^a$

$$(**) \approx \left(\frac{n}{m}\right)^{k+s} \cdot \frac{1}{\sqrt{k+s}} \cdot e^{(k+s)} \cdot \frac{1}{\sqrt{2\pi}}$$

nechť $m \geq 3n$ $\qquad\qquad e = 2.718\ldots$

$$\leq \left(\frac{e}{3}\right)^{k+s} \cdot \frac{1}{\sqrt{(k+s)2\pi}}$$

očekávaná doba vyhledání $\leq \sum_{k \geq 1} k \cdot \Pr[\text{vyhledání trvá čas } k]$

$$\leq \underbrace{\sum_{k \geq 1} k \cdot \underbrace{\sum_{s \geq 0} \left(\frac{e}{3}\right)^{k+s} \cdot \frac{1}{\sqrt{(k+s)2\pi}}}_{\leq O\left(\left(\frac{e}{3}\right)^{k}\right)}}_{O(1)} = O(1)$$

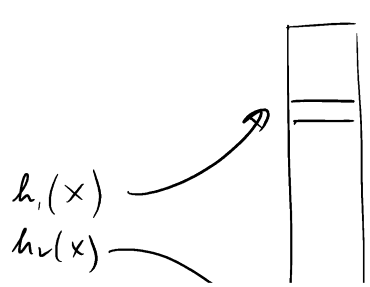
■

• <u>Balls & Bins</u> s volbou – $n$ míčků, $n$ košů, pro každý míček zvolím náhodně dva koše a hodím ho do toho prázdnějšího

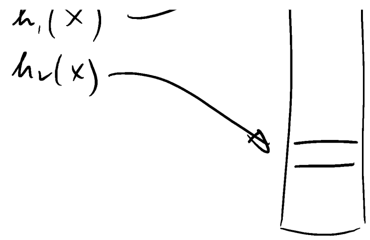
• očekávané maximální zaplnění koše $O(\log \log n)$.

→ <u>kukaččí hašování</u>

$h_1, h_2 : U \to \{1, \ldots, m\}$

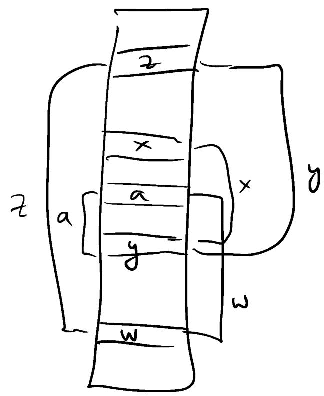



• $x$ je buď na pozici $h_1(x)$ nebo $h_2(x)$, ale




$h_1(x)$ nebo $h_2(x)$, ale nikde jinde není!

```
procedure insert(x)                              [R. Pagh, 2006]
   if T[h1(x)] = x or T[h2(x)] = x then return;
   pos:=h1(x);
   loop n times {
      if T[pos] = NULL then { T[pos]:=x; return };
      swap x and T[pos];
      if pos = h1(x) then pos:=h2(x) else pos:=h1(x);
   }
   rehash(); insert(x);
end
```




- Find
- Delete

} $O(1)$ v nejhorším případě

- Insert $O(1)$ v očekávaném případě

→ pouze dvě možné pozice pro $\underline{x}$.

Analýza pro $m = 6n$, kukačkové hašování funguje dobře i pro $m \approx 2.3n$

<u>kukačkový graf</u>: pozice v tabulce ... vrcholy

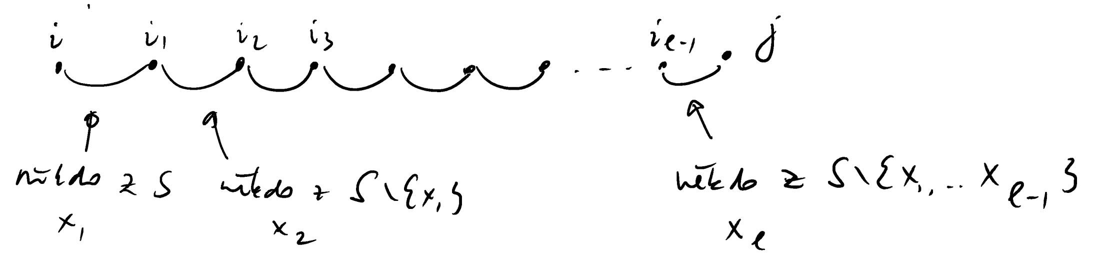
$m = 6n$ $\{h_1(x), h_2(x)\}$ ... hrany

$m$ vrcholů, $n$ hran $x \in S$

<u>Tvrzení 1</u>: Nechť $S \subseteq U$ t.ž. $|S| = n$. Pravděpodobnost, že pro zcela náhodně zvolené hašovací fce, kukačkový graf obsahuje cyklus $\leq \frac{1}{2}$.

<u>Lemma</u>: Nechť $S \subseteq U$ t.ž. $|S| = n$. Pravděpodobnost, že pro zcela náhodně zvolené hašovací fce, pozice $i$ a $j$ jsou spojeny cestou délky $l$ v kukačkovém grafu $\leq \left(\frac{2n}{m}\right)^l \cdot \frac{1}{m} \leq \frac{1}{3^l} \cdot \frac{1}{m}$.

Dk:




$$\Pr[d(i,j) = \ell] \le \Pr\left[\begin{matrix}\exists x_1 \dots x_\ell, \exists i_1, \dots i_{\ell-1} \\ h(x_1) = \{i, i_1\} \,\&\, \dots\; h(x_\ell) = \{i_{\ell-1}, j\}\end{matrix}\right]$$

$$\le \frac{2^\ell \cdot \underbrace{n \cdot (n-1) \cdot \dots (n-\ell+1)}_{\text{výběr } x_1 \dots x_\ell} \cdot m^{\ell-1} \cdot m^{2(n-\ell)}}{m^{2\ell}}$$

$h_1(x_s) = i_s$ nebo $h_2(x_s) = i_s$

výběr $i_1 \dots i_{\ell-1}$

výběr hašovacích hodnot pro $S \setminus \{x_1 .. x_\ell\}$

$$\le \frac{2^\ell}{m^{2\ell}} \cdot n^\ell \cdot m^{\ell-1} = \left(\frac{2n}{m}\right)^\ell \cdot \frac{1}{m} \quad \blacksquare$$

## Dk Tvrzení 1:

$$\Pr[\, i \text{ je obsažený v cyklu délky } \ell\,] \le \frac{1}{3^\ell} \cdot \frac{1}{m}$$

dle lemmatu cesta z $i$ do $i$ délky $\ell$.

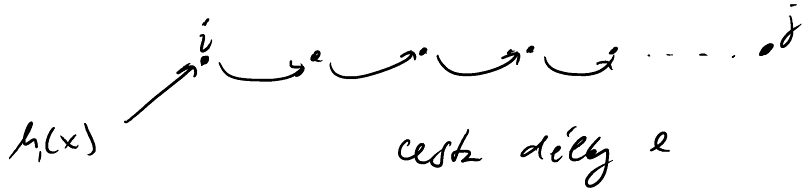
$$\Pr[\, i \text{ je obsažený v cyklu}\,] \le \sum_{\ell \ge 1} \frac{1}{3^\ell} \cdot \frac{1}{m} \le \frac{1}{m} \cdot \frac{1}{2}$$

$$\Pr[\,\text{někdo je obsažen v cyklu}\,] \le m \cdot \frac{1}{m} \cdot \frac{1}{2} = \frac{1}{2} \quad \blacksquare$$

→ pokud kukačkový graf neobsahuje cyklus, všechny operace Insert uspějí. → pst. že uspějí všechny $\ge \frac{1}{2}$. Pokud některá neuspěje, přehašujeme.

⇒ Očekávaný počet přehašování $\le 1$ během $\underline{n}$ operací insert.

## Doba na operaci Insert




cesta délky $\ell$

očekávaná doba (∗) $\leq \sum_{\ell \geq 1} \ell \cdot \Pr[$z $i$ vede cesta délky $\ell]$

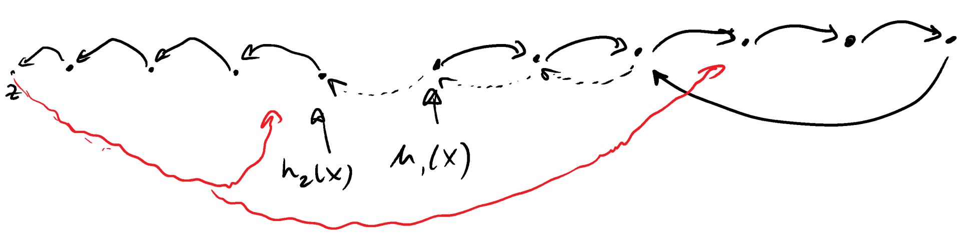
$$\leq \sum_{\ell \geq 1} \ell \cdot \left(\frac{1}{3^\ell} \cdot \frac{1}{m}\right) \cdot m = \sum_{\ell \geq 1} \frac{\ell}{3^\ell}$$

$$\leq O(1). \blacksquare$$

(∗) za předpokladu, že kukaččí graf neobsahuje cyklus. Pokud obsahuje cyklus, nutno vynásobit 4.

- smyčku při operaci Insert lze zařízout již po třeba $2 \log n$ operacích, neboť po $2 \log n$ krocích jsme téměř jistě zacyklení (cesta této délky je nepravděpodobná)

## Detailnější pohled na Insert:




při neúspěšném Insertu je pozice $z$ obsazená a prvek na této pozici se mapuje na již prošlou pozici
→ nekonečný cyklus

[prvky v cyklu se posunují o jednu pozici

[prvky v cyklu se posunují o jednu pozici
tam a zpět ]

## Výběr hašovací fce

- pokud jsou data distribuována zcela náhodně, pak libovolná rozumná fce $h: U \to \{1, \dots, m\}$ bude fungovat dobře

uniformně a nezávisle z $U$

$h^{-1}(a)$ je stejně velká $(\pm 1)$ pro $\forall a \in \{1, \dots, m\}$

ALE:

- data jsou málokdy distribuována uniformně náhodně
  $\to$ nutno brát náhodné hašovací fce.

Př: $h: U \to \{1, \dots, m\}$ $\quad |U| \geq m \cdot n$

$\exists a;\ |h^{-1}(a)| \geq n \to S \subseteq h^{-1}(a)$

všechny prvky z $\underline{S}$ se hašují na $\underline{a}$.

$\Rightarrow$ pro každou pevně zvolenou hašovací funkci existuje špatná množina. ($\to$ DDOS attack)

- ideálně: $h: U \to \{1, \dots, m\}$ vybráno zcela náhodně, tj. $\forall x \in U;\ h(x)$ je zvoleno nezávisle uniformně náhodně z $\{1, \dots, m\}$.

problém: taková $\underline{h}$ potřebuje $|U| \cdot \log m$ bitů na popis $\to$ neřeší to náš původní problém, protože bychom mohli $\underline{S}$ uložit triviálním způsobem.

$\to$ chceme podmnožinu $\mathcal{H}$ všech hašovacích fcí, t.ž. náhodně zvolená $h \in \mathcal{H}$ se bude chovat dobře z hlediska použití při hašování a $\mathcal{H}$ bude relativně malá.

... 'h' požadavek na $\mathcal{H}$

[illegible] .

• minimální požadavek na $\mathcal{H}$

pro ∀ páry $x, y \in U$, $x \neq y$

(*) $$\Pr_{h \in \mathcal{H}} [\, h(x) = h(y) \,] = \frac{1}{m}$$

tedy pravděpodobnost kolize dvou prvků taková jako u zcela náhodné fce.

$\mathcal{H}$, který splňuje (*), je univerzální hašovací systém

• silnější požadavek:

pro ∀ páry $x, y \in U$, $x \neq y$ a pro ∀ páry $a, b \in \{1, \dots, m\}$

(**) $$\Pr_{h \in \mathcal{H}} [\, h(x) = a \ \ \&\ \ h(y) = b \,] = \frac{1}{m^2}$$

$\mathcal{H}$, který splňuje (**), je 2-univerzální hašovací systém.

(nebo tzv. po dvou nezávislé haš. fce)

• obecněji

∀ různé $x_1, x_2, \dots x_k \in U$ a ∀ $a_1, \dots, a_k \in \{1, \dots m\}$

(**) $$\Pr_{h \in \mathcal{H}} [\, h(x_1) = a_1 \ \&\ h(x_2) = a_2 \ \&\ \dots \ \&\ h(x_k) = a_k \,] = \frac{1}{m^k}$$

... po k nezávislý hašovací systém

Př: 2-univerzální hašovací systémy

1) $m$ ... prvočíslo $\quad U \subseteq \mathbb{N}$

$$\mathcal{H} = \{\, h_{a,b} : U \to \{0, \dots, m-1\};\ \ a, b \in \{0, \dots, m-1\} \,\}$$

kde $h_{a,b}(x) = a x + b \mod m$

• vybrat náhodně $h \in \mathcal{H}$, znamená vybrat náhodně $\underline{a}$ a $\underline{b}$ z $\{0, \dots, m-1\}$ .

→ potřebuji: $2 \log m$ bitů na zapamatování $h$.

2) $w, k$ celá čísla $\quad h: \{0,1\}^w \to \{0,1\}^k$

$$\mathcal{H} = \{ h_{A,b} : \{0,1\}^w \to \{0,1\}^k, \ A \in \{0,1\}^{k \times w}, \ b \in \{0,1\}^k \}$$

kde $h_{A,b}(x) = \underbrace{Ax}_{\text{násobení matic nad } GF[2]} + b$

potřebuji: $k \cdot w + k$ bitů na popis $h$.

3) (konvoluce) $\quad w, k$ celá čísla $\quad h: \{0,1\}^w \to \{0,1\}^k$

$$\mathcal{H} = \{ h_{a,b} ; \ a \in \{0,1\}^{w+k-1}, b \in \{0,1\}^k \}$$

$$(h_{a,b}(x))_j = b_j + \sum_{i=1}^{w} a_{i+j-1} x_i \qquad j = 1, \dots, k$$

4) (multiply-shift) $\quad w, k$ celá čísla $\quad h: \{0,1\}^w \to \{0,1\}^k$

$$\mathcal{H} = \{ h_{a,b} ; \ a, b \in \{0,1\}^{w+k-1} \}$$

$$h_{a,b}(x) = \left[ (ax + b) >> (w-1) \right]_{1..k}$$

nejnižší bity $1, \dots, k$

x, · nad celými čísly

- obecněji: $\quad w' \geq w + k - 1$ $\qquad$ např. $w = 32$, $k = 15$, $w' = 64$

$a, b \in \{0,1\}^{w'}$

$$h_{a,b}(x) = [ax + b]_{w'-k+1, \dots, w'}$$

$$= \left[ (ax + b) >> (w' - k) \right]_{1..k}$$

---

2), 3) nepraktické

4) rychlé praktické, nepotřebuje dělení, pouze jedno násobení

1) často používané, potřebuje dělení

---

5) vektory $\qquad h: \{0,1\}^{w \times d} \to \{0,1\}^k \qquad w' \geq w + k - 1$

$$\mathcal{H} = \{ h_{a_0, a_1, \dots a_{d-1}, b} : \ a_0, a_1, \dots a_{d-1}, b \in \{0,1\}^{w'} \}$$

$$\mathcal{H} = \{ h_{a_0,a_1,\dots a_{d-1},b} \;:\; a_0,a_1,\dots a_{d-1}, b \in \{0,1\}^w \}$$

$$h_{a_0,\dots a_{d-1},b}(\underbrace{x_0,\dots x_{d-1}}_{\in \{0,1\}^w}) = \left[\left(\sum_{i\in\{0,\dots d-1\}} a_i x_i\right) + b\right]_{w'-k+1,\dots,w'}$$

$a_0, a_1, \dots a_{d-1}, b$ zvoleny náhodně

• pro $\underline{d}$ sudé lze též použít:

$$h_{a_0,\dots a_{d-1},b}(x_0,\dots x_{d-1}) = \left[\left(\sum_{i\in\{0,\dots,d/2-1\}} (a_{2i} + x_{2i+1})(a_{2i+1}+x_{2i})\right) + b\right]_{w'-k+1,\dots,w'}$$

→ ušetří se polovina násobení!

• pokud chceme hašovat vektory proměnné délky $d' < d$,
kde $d'$ je sudé

$$h_{a_0,\dots a_d}(x_0,\dots x_{d'-1}) = \left[\left(\sum_{i\in\{0,\dots \frac{d'}{2}-1\}} (a_{2i} + x_{2i+1})(a_{2i+1}+x_{2i})\right) + a_{d'}\right]_{w'-k+1,\dots,w'}$$

## Hašování řetězců

$x_0,\dots x_{d-1} \in U$

prvočíslo $p \geq |U|$

$a \in \{0,\dots p-1\}$

$$h_a(x_0,x_1,\dots x_{d-1}) = \sum_{i=0}^{d-1} x_i \cdot a^i \bmod p$$

• $\forall\; x_0,\dots x_{d-1},\; y_0,\dots y_{d-1} \in U \qquad \bar{x} \neq \bar{y}$

$$Pr_a\left[h_a(x_0,x_1,\dots x_{d-1}) = h_a(y_0,\dots y_{d-1})\right] \leq \frac{d}{p}$$

<u>Dk</u>: dva různé polynomy stupně $\leq d-1$ se mohou shodovat v nejvýše $\underline{d}$ bodech ∎

<u>Př</u>: $p = 2^{89}-1$ ← Mersennovo prvočíslo $\qquad d \leq 2^{57}$ → pst. kolize $\leq \frac{1}{\;}$

Př: $p = 2^{89}-1$ — Mersennovo prvočíslo  $d \le 2^{57} \to$ pst. kolize $\le \frac{1}{2^{32}}$

$h_a(\cdot)$ lze složit s hašovací fcí $\{0,\dots p-1\} \to \{0,\dots m-1\}$:

$a, b, c \in \{0,\dots p-1\}$

$\to h_{a,b,c}(x_0,\dots x_{d-1}) = \left(\left(a\left(\sum_{i=0}^{d-1} x_i c^i\right) + b\right) \bmod p\right) \bmod m$

- Pokud $d < \frac{p}{m}$ pak je pst. kolize $\le \frac{2}{m}$.

## Tabulkové hašování

- obecně $x_0, x_1, x_2 \dots x_{d-1} \in \{0,\dots,m-1\}$
  náhodné tabulky $T_0, T_1, \dots T_{d-1} : \{0,\dots,m-1\} \to \{0,1\}^l$

  $$T_0[x_0] \oplus T_1[x_1] \oplus T_2[x_2] \dots \oplus T_{d-1}[x_{d-1}]$$

  XOR po bitech

  $\to$ 2-univerzální hašování

- speciálně 5-univerzální

  $x_0, x_1 \in \{0,\dots,m-1\}$

  náhodné tabulky $T_0, T_1 : \{0,\dots m-1\} \to \{0,1\}^l$
  $T_2 : \{0,\dots,2m-1\} \to \{0,1\}^l$

  $$T_0[x_0] \oplus T_1[x_1] \oplus T_2[x_1 + x_2]$$ $\leftarrow$ 5-univerzální

  XOR po bitech

## k-univerzální hašování

$x \in \{0,\dots,p-1\}$  $p$ … prvočíslo

$a_0,\dots a_{k-1} \in \{0,\dots,p-1\}$ náhodné

$$h_{a_0,\dots a_{k-1}}(x) = \sum_{i=0}^{k-1} a_i x^i \bmod p$$

$\to$ k-univerzální

Mersenne prvočísla : $2^{31}-1,\ 2^{61}-1,\ 2^{89}-1,\ 2^{107}-1$

$p = 2^a - 1$ ... Mersennovo prvočíslo

$\rightarrow \quad y = (y \,\&\, p) + (y >> a) \pmod p$

Perfektní hašování

$\mathcal{H}$ ... 2-univerzální hašovací systém $U \to \{1, .., m\}$

$S \subseteq U \qquad |U| = n$

$x_1 \neq x_2 \in S \qquad \Pr_{h \in \mathcal{H}} [h(x_1) = h(x_2)] = \frac{1}{m}$

očekávaný počet dvojic $(x_1, x_2) \in S^2$, které kolidují:

$$\leq n^2 \cdot \frac{1}{m}$$

Pokud $m \geq 2n^2$ pak $\Pr_{h \in \mathcal{H}} [h \text{ je perfektní pro } S] \geq \frac{1}{2}$