- kdo je kdo
- konzultační hody: 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 všechny příklady.

celkem 5

DÚ 1 – do 23.10., nejpozději do 30.10.

- úvod – o čem přednáška bude, o čem nebude

plán – viz sylabus

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

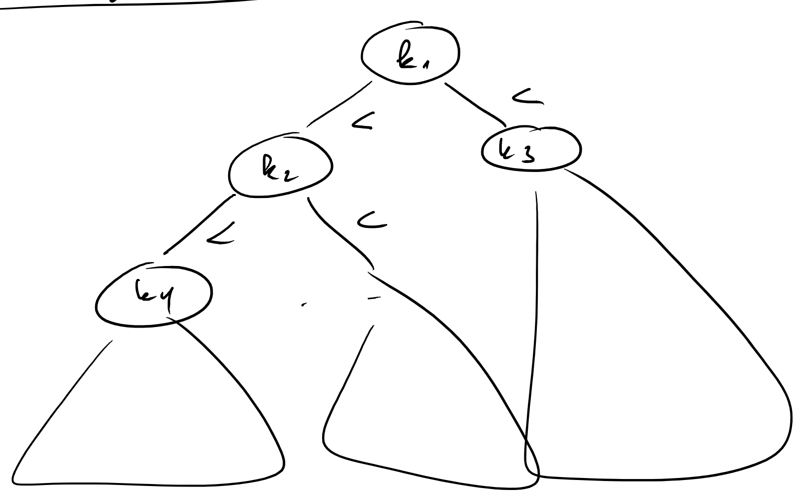
$\in U$ ... uspořádaná množina

chceme:
- insert (klíč, hodnota)
- delete (klíč)
- find (klíč) $\to$ hodnota

Zajímá nás <u>čas</u> na jednotlivé operace

$\hookrightarrow$ počet elementárních operací jako aritmetické operace, porovnání, skok, následování pointeru, ...

<u>binární vyhledávací strom</u>



- doba vyhledání ~ hloubka stromu

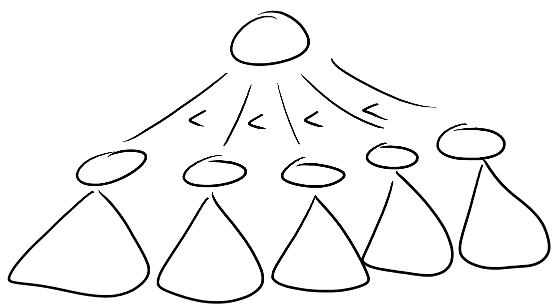
→ minimalizace hloubky – vyvažování

- 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

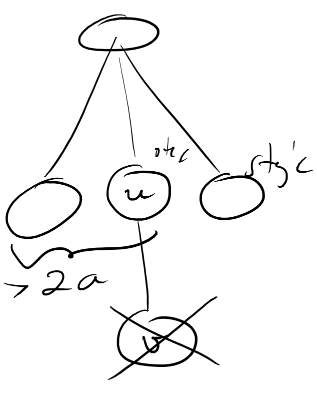
→ B-stromy $a, b \approx$ velikost bloku na disku

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

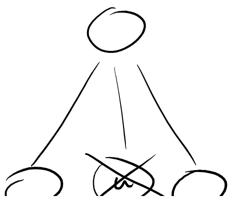
- find (k)
  - projdi strom od kořene, jdi vždy do podstromu, kde by k mohlo být.

- 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 najíž $\frac{b+1}{2}$ synů (→ vytvoř nový vrchol v', který dostane $\frac{b+1}{2}$ menších synů v)
    → rekurzivně ulož v' do otce v.

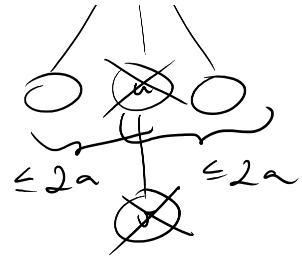
- delete (k)
  - (rekurzivně od listu v obsahující klíč k)
  - pokud otec u mazaného vrcholu v obsahuje > a synů, vymaž z u informaci o v → done



  - 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 více než 2a synů, přesuň syny u do jednoho ze strýčků a



nemá více než 2a synů, přesuň syny u do jednoho ze strýců a rekurzivně smaž u. (zaktualizuj informaci o strýcích v otci u)

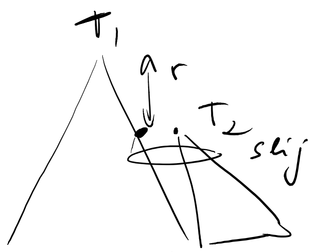
- Find, Insert, delete ... čas na operaci $O(\log n)$
  (za předpokladu, že slití a rozštěpení vrcholu trvá čas $O(1)$)

Operace Join, Split

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

maximální hodnota v T

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

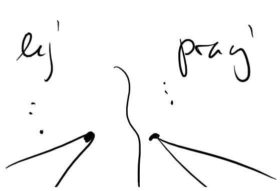


$r = d(T_1) - d(T_2)$

syny kořene T_2 přidej najednou k synům posledního vrcholu na hladině r ve stromu T_2. 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

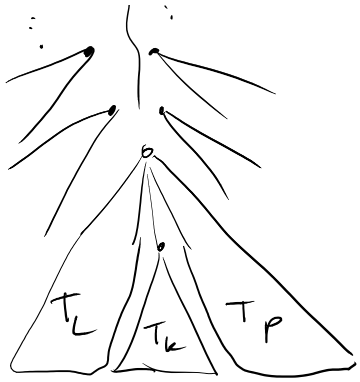
– pokud výška (T_2) > výška (T_1) postupuj analogicky jako v předchozím případě.

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

  list přidáme na levý zásobník

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

- 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

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 ... $O(m + l + \log n)$

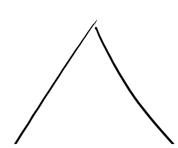
  předpoklad $b \geq 2a$.

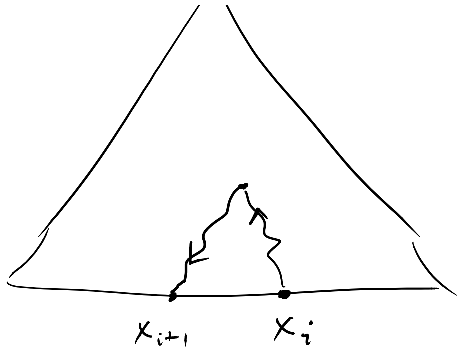
  → <u>amortizovaně</u> $O(1)$ štěpení / slučování za operaci

- 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 slitím se sourozencem.

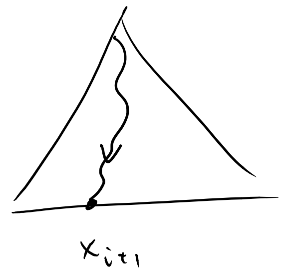
- <u>A-sort</u> setřídí posloupnost klíčů tím, že je vloží do (a,b)-stromu a pak je vypíše během průchodu do hloubky
  - při vkládání hledáme pozici pro další klíč ne od kořene, ale od listu, kam jsme vložili naposled (strom s "prstem" – ukazatelem na poslední použitý list)

<u>vstupní posloupnost</u>: $x_1, x_2, \ldots x_i, x_{i+1}, \ldots$



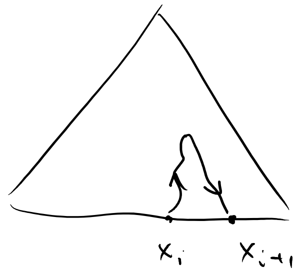


místo



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

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



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

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

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

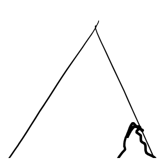
z konkávnosti fce log

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

$F = \sum_{i=1}^{n} |\{j \le i;\ x_i < x_j\}|$ ← 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

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

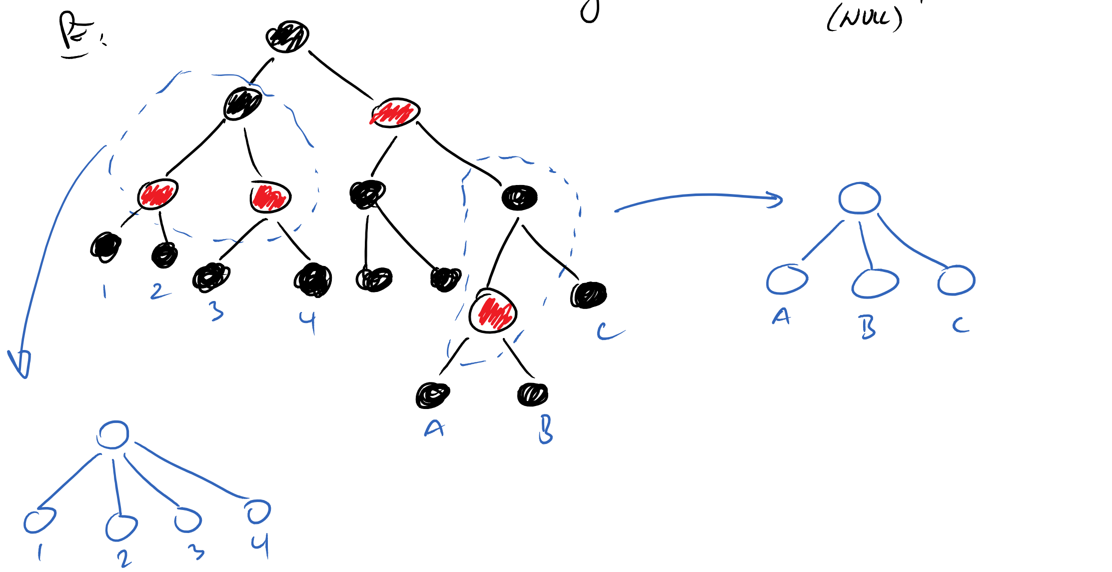
Červeno-černé stromy

- 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 pointrem)
(NULL)

Př.:

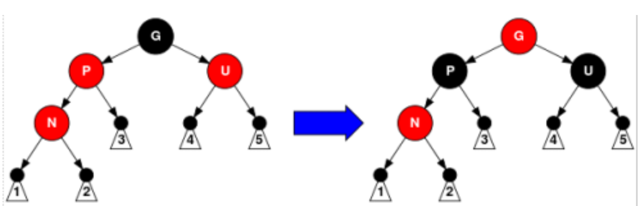


$\Rightarrow$ (*)+(**) Červeno-černé stromy jsou ekvivalentní (2,4)-stromům

Vyvažování při vložení (INSERT)

- vložen červený uzel N

1)

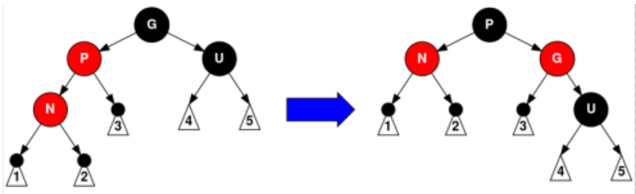


→ problém s příliš mnoha červenými vrcholy se přenese k otci

2.

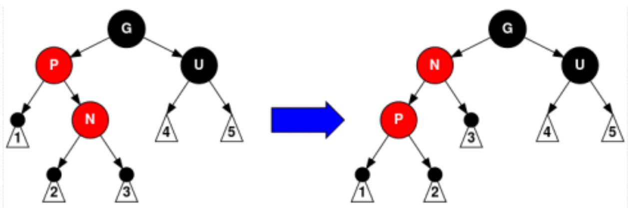
k otci

2)



Špatné rozložení černé vrcholy se přerotuje

3)



[obrázky z wikipedie]

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

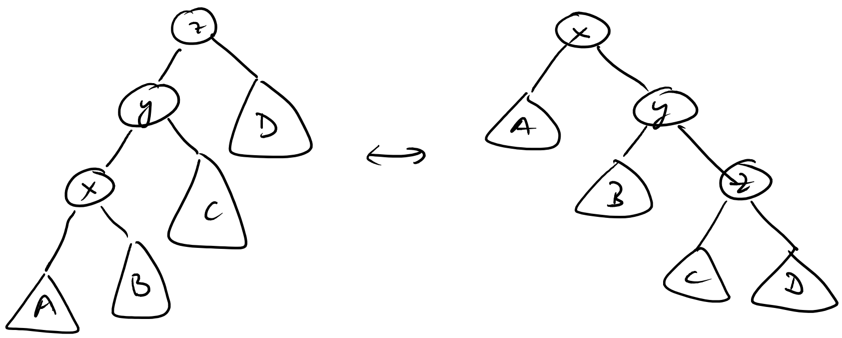
Splay stromy

- 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

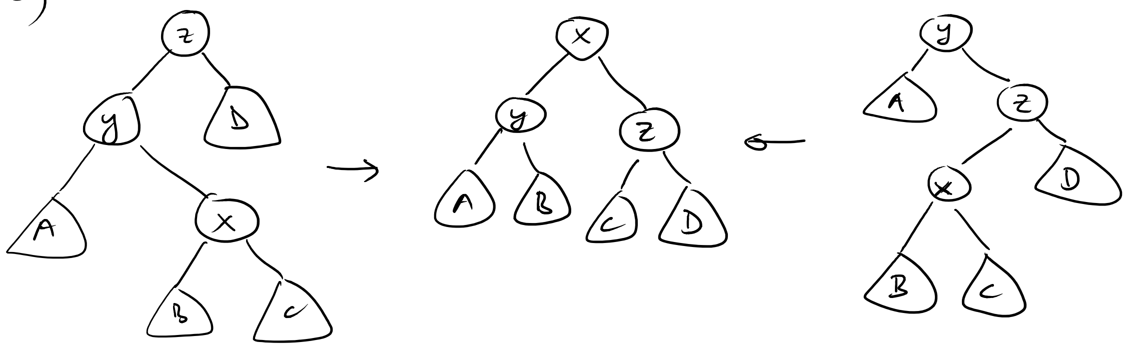
- operace splay(x), přesune prvek x do kořene pomocí rotací
- po každé operaci find, insert, ... aplikujeme operaci splay.

- dvojité rotace

a)

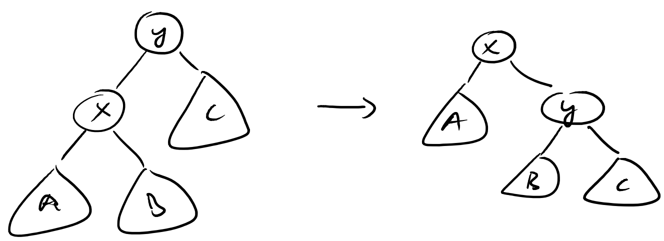


b)



• rotace u kořene

c)



amortizovaný čas operace t:

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

skutečný čas

potenciál po operaci

potenciál před operací

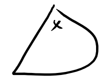
$\Phi(T)$ .. potenciál stromu T

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

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

v T

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



- Čas na $\underline{m}$ operací:

strom po i-té operaci

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

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

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

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

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

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

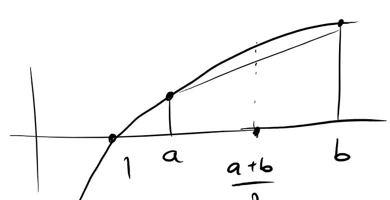
$$= 2 + r'(x) - 2r(x) + r'(z) = (*)$$

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

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

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

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



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

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

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

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

$$t \le (*) \le 3r'(x) - 3r(x) \qquad \blacksquare$$

<u>Dk b):</u>

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

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

<u>Dk:</u> stejný jako výše $\blacksquare$

$$(*) \le 2r'(x) - 2r(x) = 2(r'(x) - r(x)) \qquad \blacksquare$$

<u>Dk c):</u>

$$t \le 1 + r'(x) - r(x) + r'(y) - r(y)$$
$$\le 1 + r'(x) - r(x)$$
$$\le 1 + 3(r'(x) - r(x)) \qquad \blacksquare$$

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

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

(v:) původní kořen stromu T

<u>Dk.</u>

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

a: amort. čas; t: skutečný čas; t_i: i-tá rotace; T_i: po i-té rotaci; T_{i-1}: před i-tou rotací

$$\le \sum_i a_i$$

a_i: amort. čas i-té rotace

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

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

rank i po i-té rotaci

za poslední jednoduchou rotaci typu c)

$$= 3(r'(x) - r(x)) + 1$$

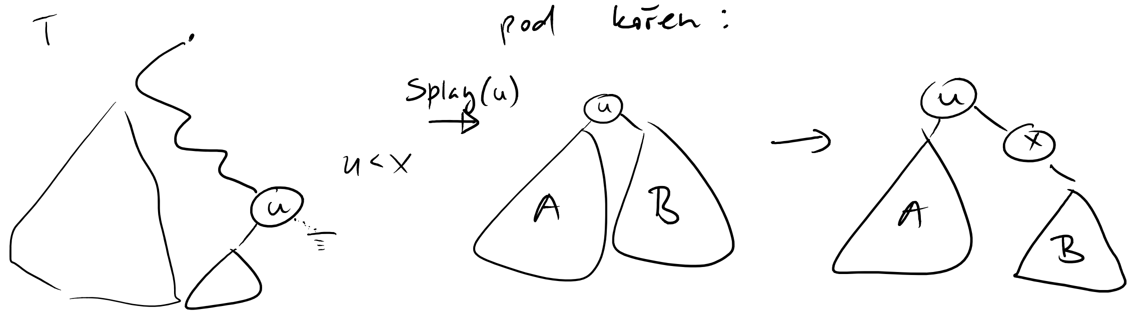
konečný rank x = rank kořene

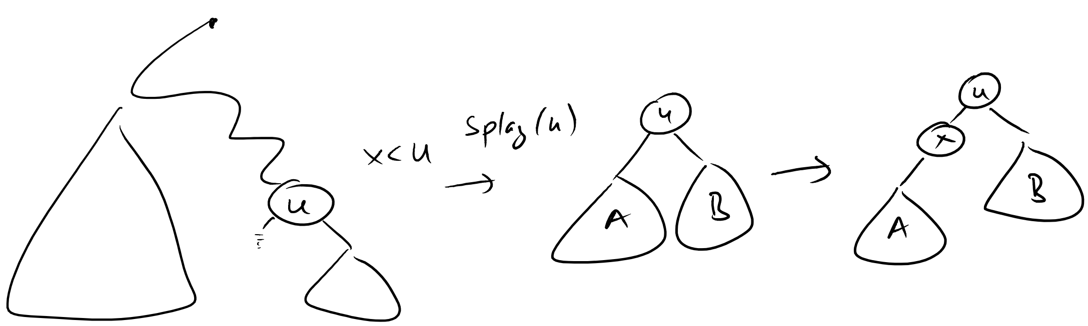
původní rank x

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

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

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





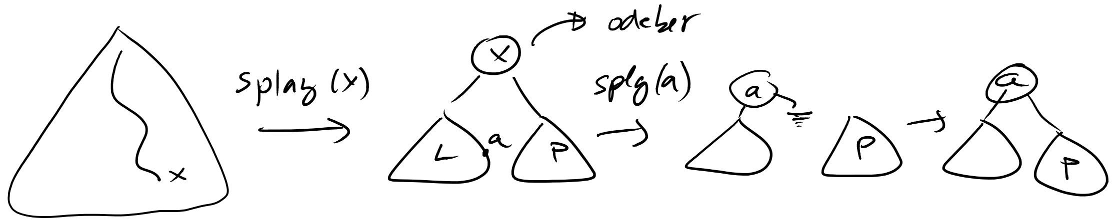
Pozorování: nechť $a, b \in T$ jsou takové, že $\forall c \in T$
$a < b$
$c \notin (a, b)$
$x \in (a, b)$

pak u je buď $\underline{a}$ nebo $\underline{b}$.

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

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

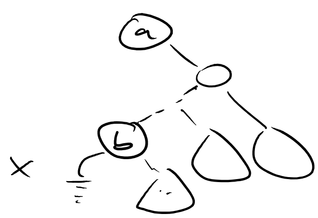
- $\underline{\text{delete}}(x)$: najdi $\underline{x}$, splay(x), odeber x, pokud získám dva neprázdné stromy L a P, najdi nejvyšší prvek $\underline{a}$ v L, splay(a), připoj P pod $\underline{a}$.



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

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

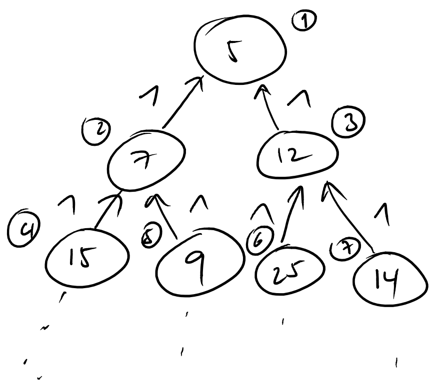
$\underline{\text{Pozn}}$: Operace z hodiny (a,b) t.ž. $a < x < b$ a $\nexists c \in T$ $a < c < b$ může být též



Haldy

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

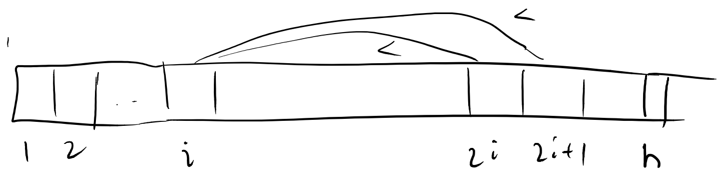
i-tý vrchol má za syny



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

→ lze jednoduše uložit do pole

... regulární halda



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 dolu k listům, dokud otec je větší než syn.

čas na operace:

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

Chceme rychlejší operaci Insert → binomiální haldy

binomiální halda – soubor haldově uspořádaných stromů velikosti 2^k

. pamatujeme si, který z těchto stromů obsahuje minimální prvek

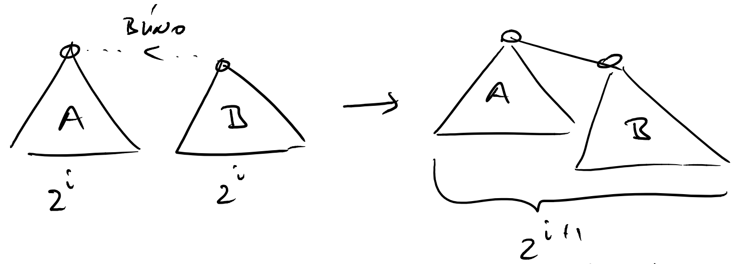
zbrklá varianta – nejvýše jeden strom dané velikosti

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:

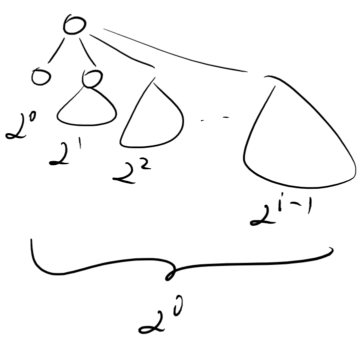


- průběžně aktualizujeme ukazatel na strom s minimálním prvkem

min ... vrátí hodnotu kořene stromu s minimálním prvkem

$\underbrace{\text{hodnotu kořene}}_{=\text{minimální prvek}}$

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 souboru 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.

sliji stromy
velikosti 2^i probíhne
$\frac{n}{2^i}$ - krát $\Rightarrow$
celkem $\sum_{i=0}^{\log n} \frac{n}{2^i} \le n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$

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

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

Min ... vrátí nejmenší prvek

Delete-min ... stejně jako ve zbrklé variantě,
odřízneme kořen s nejmenším prvkem,
podstromy přidáme do seznamu
a sléváme 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.)

Čas na operaci
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

amortizovaně: Insert $\le C + \underbrace{\Phi(T') - \Phi(T)}$

C ... vhodná konstanta

$= C$ ✓

$= O(1)$

Min $= O(1)$

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

$\leq C' \cdot \log n$

$\leq C_1 \log n = O(\log n)$

Fibonacci halda

nové:

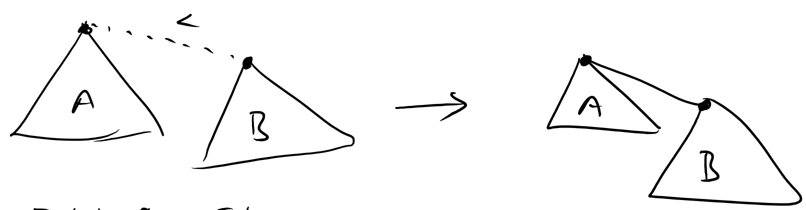
- Decrease-key (x, Δ) ... sníží hodnotu prvku x o Δ.

  → amortizovaně $O(1)$

- podobně jako líná binomiální halda, stromy nemusí mít velikost mocniny 2, tj. 2^i.

  → řád stromu = počet synů kořene

  → slučujeme 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, Δ)
  - snížíme hodnotu x o Δ, 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 ukládaných stromů odznačuj.)

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

• kořen může přijít o libovolně mnoho synů

implementace: vrchol si musí pamatovat počet synů potomků, kteří jsou uspořádaní ve spojovém seznamu, a musí si pamatovat zda již přišel o nějakého potomka.

Struktura stromů:

Fibonacciho čísla $F_1 = 1$ $\quad F_0 = 0$

$F_2 = 1 \qquad F_n = F_{n-1} + F_{n-2}$

$F_3 = 2$

$F_4 = 3$

$F_5 = 5$

$\vdots$

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

Dk: indukce na n. $n = 1, 2$ triv.

$n \to n+1$

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

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

<u>Dk</u>: indukce na n. $n = 3$ ✓

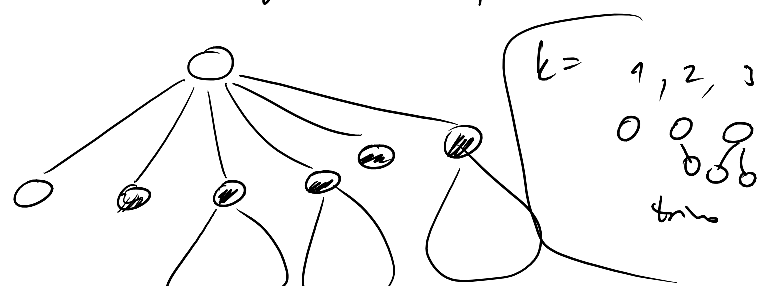
$n \to n+1$

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

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

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

nejhorší případ:



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

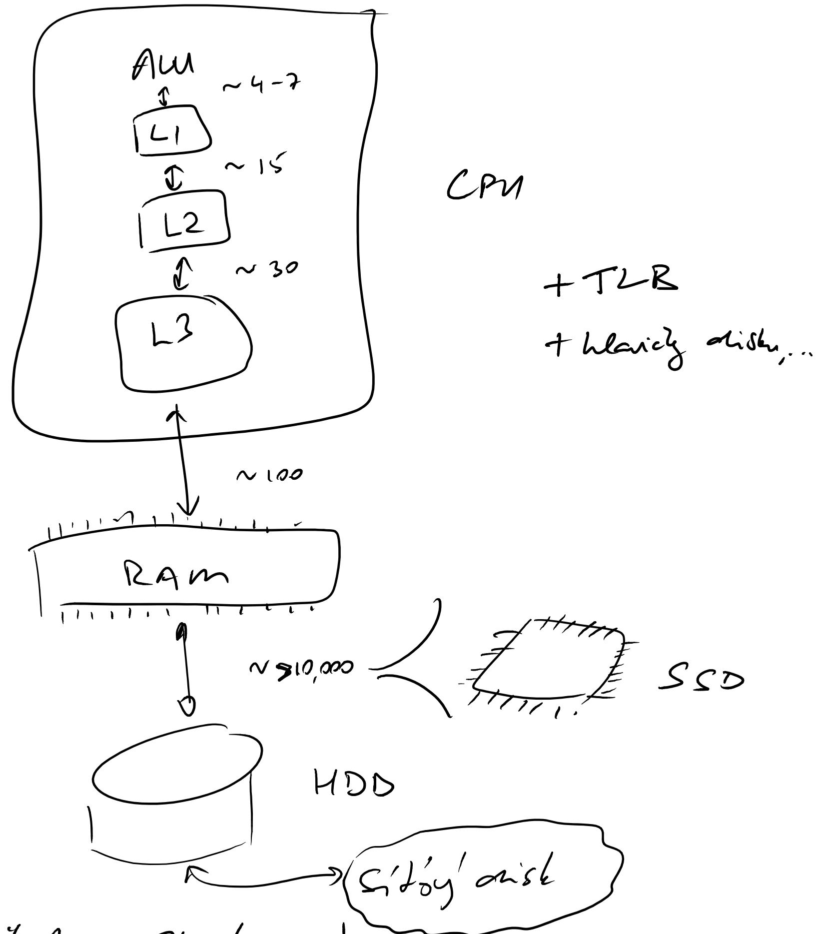
- velikosti podstromů to splňují $\Rightarrow$ splňuje i otec:

$$\geq \underset{\text{kořen}}{1} + \underset{\text{nejlevější syn}}{F_1} + \sum_{i=1}^{k-1} F_i = 1 + 1 + F_{k+2} - 1 = F_{k+1} + 1$$

<u>amortizovaná analýza</u>:

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

Paměťové hierarchie

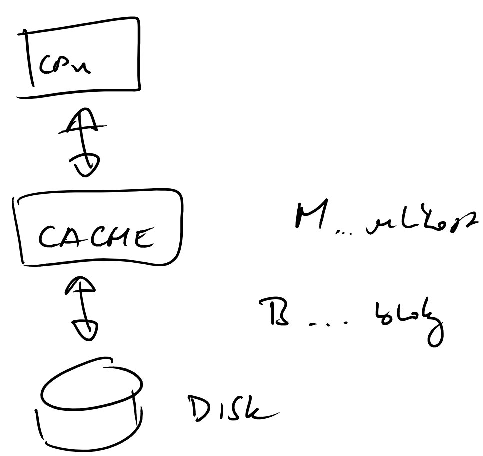


M ... velikost paměti (cache)

B ... velikost přenášených bloků

→ M/B počet bloků v paměti

model externí paměti



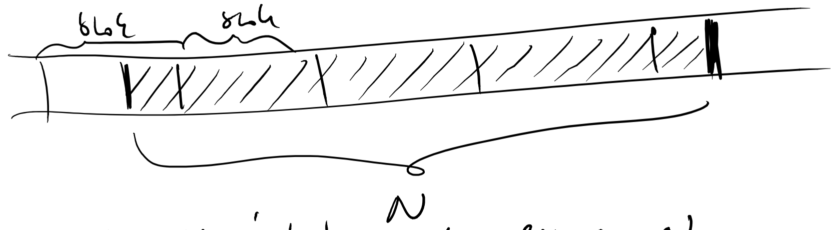
- u algoritmů měříme počet přenesených bloků
- soustředíme se na jednu úroveň a zbylé ignorujeme
  → algoritmus optimalizovaný pro konkrétní úroveň

Př.: • Průchod souvislého kusu dat délky N.

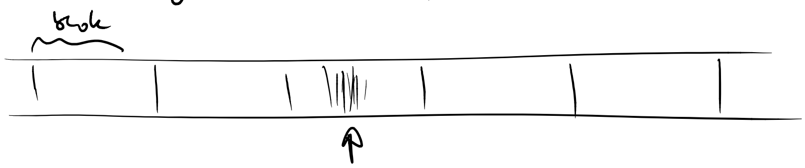
- $\lceil M/B \rceil + 1$ přenos

  bloky   zač.

• $\lceil M/B \rceil + 1$ přenos

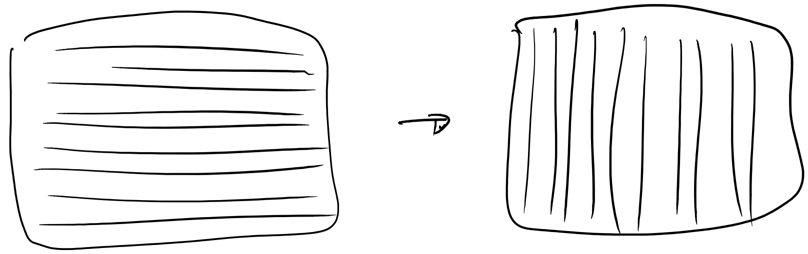


• binární vyhledávání v poli velikosti N



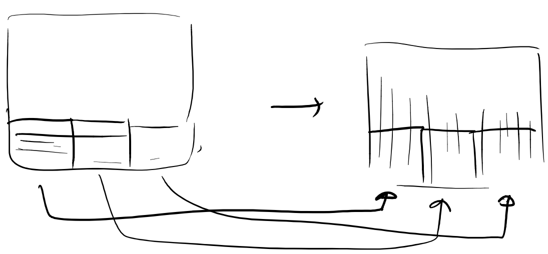
• $\log N - \log B$ přenosů

• transposice 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 nevyskytuje.

⇒ 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:

$$\begin{array}{c} 1/2 \\ 1/2 \end{array} \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \longrightarrow \begin{pmatrix} A_{11}^T & A_{21}^T \\ A_{12}^T & A_{22}^T \end{pmatrix}$$

$\quad 1/2 \quad 1/2$

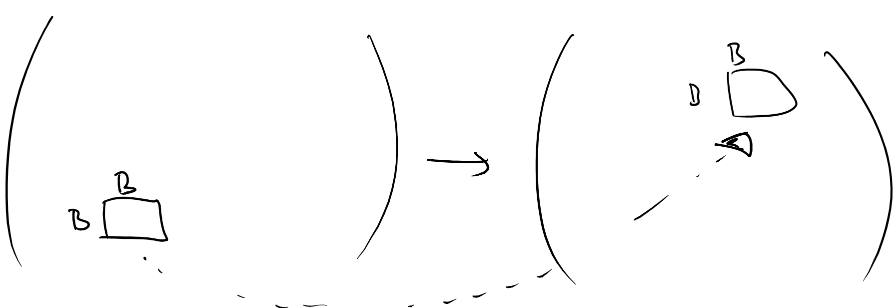
rekurzivně opakuj

\# operací $O(n^2)$

\# IO $O(n^2/B)$

předpoklad $M \geq B^2$ ("tall cache assumption")

Dk.:

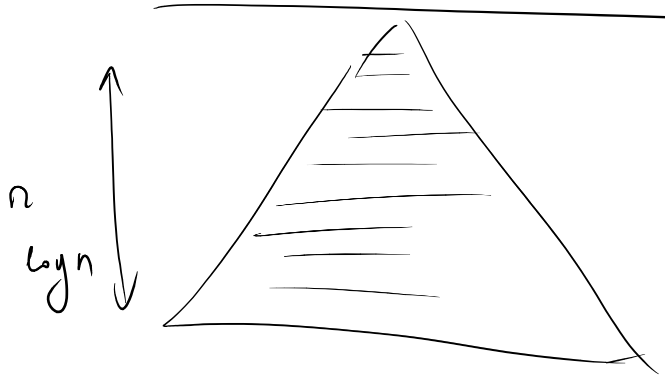


$O(B)$ IO operací, jakmile se rekurze dostane na velikost matice $\ell \times \ell$, $\ell = O(B)$

$\frac{n^2}{B^2}$ takových podmatic

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

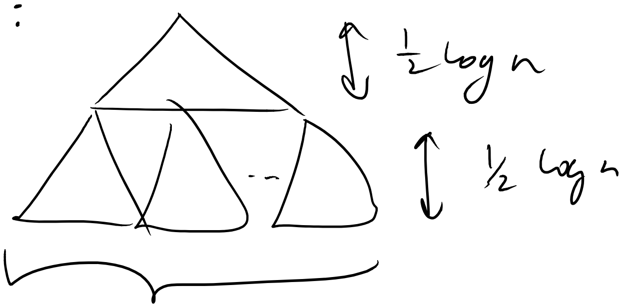
• van Emde Boas rozložení stromů (statických)



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.

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

→ $O(1)$ 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, ... ✓

• <u>optimální správa cache</u>:

problémy: 1) neznámá budoucnost
2) asociativita cache

1) není problém:

<u>Sleator - Tarjan</u> (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

<u>Tvrz</u>: $\#\text{výpadků LRU} \le \frac{n_{LRU}}{n_{LRU} - n_{OPT}} \cdot \#\text{výpadků OPT} + n_{LRU}$

<u>Pozorování</u>: 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. $|\{s_i,\ t_1 < i < t_2\}| \ge n_{LRU}$)

<u>Dk</u>: 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 $t_1 \Rightarrow$ tvrzení ▨

<u>Dk</u>: 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

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

⇒ Pokud $n_{LRU} = 2\, n_{OPT}$ pak LRU se chová optimálně 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 $\le n_{LRU}$ výpadků
OPT má v každém úseku $\ge n_{LRU} - n_{OPT}$ 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 $\underline{m}$.

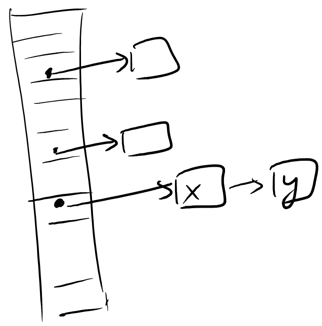
$$h: U \to \{1, \dots, m\} \quad \dots \text{ hašovací fce}$$

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

- **způsoby řešení**:

- separované řetězce

m

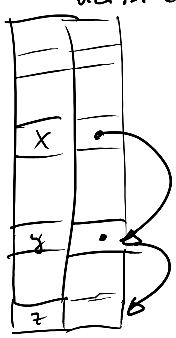


$h(x), h(y)$

← seznam / vyhledávací strom

- srůstající řetězce

ukazatel



$h(x), h(y)$

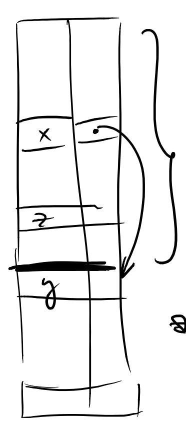
$h(z)$

LICH ... nový prvek přidán na konec seznamu

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

(seznam není obousměrně propojený)

- s pomocným polem



$h(x), h(y)$

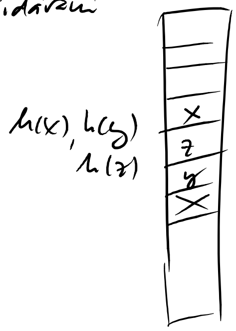
$h(z)$

$h(\cdot)$

← 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 – podobně jako lineární přidávání, ale
zkouším pozice

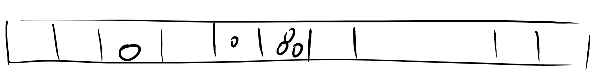
$$h_1(x) + i\, h_2(x) \quad , i = 0, 1, 2, \dots$$

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
$h_2(x) \in \{1, \dots, m-1\}$

<u>DELETE</u> – obvykle problematický
- označování smazaných prvků a jejich
znovupoužití při INSERT
→ pokud příliš mnoho označených prvků
→ přehašuj vše

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



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

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

$k \geq 1$

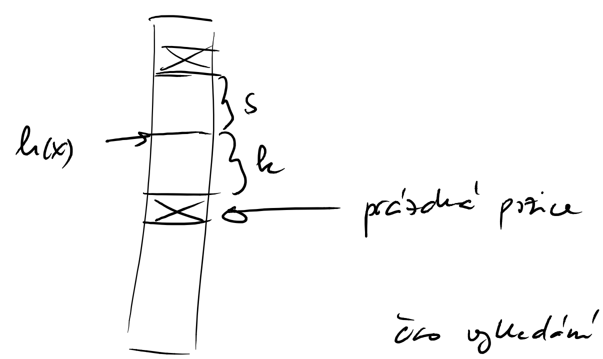
$$\underset{k \ll n}{\approx} \frac{n^k}{k!} \cdot \frac{1}{n^k} \cdot \frac{1}{e} = \frac{1}{e \cdot k!}$$

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

– odpovídá to situaci, kdy bych při hašování bral 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 muselo namapovat do bloku velikosti $k+s$ z celkového počtu m pozic.

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

z určeného pole 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 $m \geq 3n$ $\qquad e = 2.718\ldots$

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

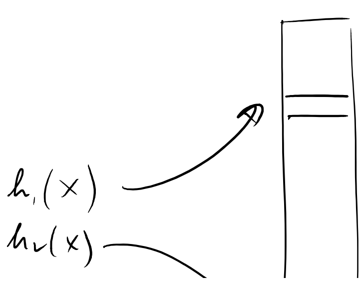
očekávaná doba vyhledávání $\leq \sum_{k \geq 1} k \cdot \Pr[\text{vyhledání trvá čas } k]$

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

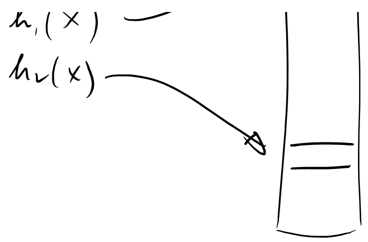
- Balls & Bins s volbou – n míčků, n košů, pro každý míček zvolím náhodně dva koše a hodím ho do toho prázdnějšího
- očekávané maximální zaplnění koše $O(\log \log n)$.

→ kukaččí hašování

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

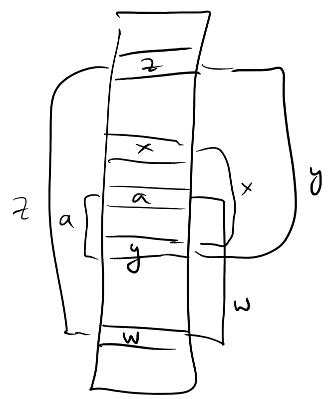


- x je buď na pozici $h_1(x)$ nebo $h_2(x)$, ale



$h_1(x)$ 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 $m \approx 2.3\,n$

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

$\{h_1(x), h_2(x)\}$ ... hrany

$x \in S$

$m = 6n$

m vrcholů, n hran

<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:

$i \quad i_1 \quad i_2 \quad i_3 \quad \dots \quad i_{l-1} \quad j$

někdo z S — x_1 ; někdo z $S \setminus \{x_1\}$ — x_2 ; někdo z $S \setminus \{x_1, \dots x_{l-1}\}$ — x_l

$$\Pr[d(i,j) = l] \le \Pr[\exists x_1 \dots x_l, \exists i_1 \dots i_{l-1} : h(x_1) = \{i, i_1\} \,\&\, \dots \, h(x_l) = \{i_{l-1}, j\}]$$

$$\le \frac{2^l \cdot n \cdot (n-1) \cdot \dots \cdot (n-l+1) \cdot m^{l-1} \cdot m^{2(n-l)}}{m^{2l}}$$

2^l: $h_1(x_s) = i_s$ nebo $h_2(x_s) = i_s$

$n \cdot (n-1) \cdots (n-l+1)$: výběr $x_1, \dots x_l$

m^{l-1}: výběr $i_1, \dots i_{l-1}$

$m^{2(n-l)}$: výběr hašovacích hodnot pro $S \setminus \{x_1, \dots x_l\}$

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

Dk Tvrzení 1:

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

dle lemmatu cesta z i do i délky l.

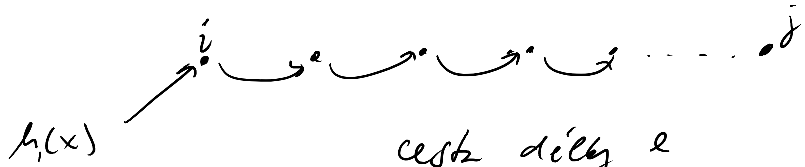
$$\Pr[i \text{ je obsažen v cyklu}] \le \sum_{l \ge 1} \frac{1}{3^l} \cdot \frac{1}{m} \le \frac{1}{m} \cdot \frac{1}{2}$$

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

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

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

Doba na operaci Insert



cesta délky ℓ

očekávaná doba $\le \sum_{\ell \ge 1} \ell \cdot \Pr[$z i vede cesta délky $\ell]$ (*)

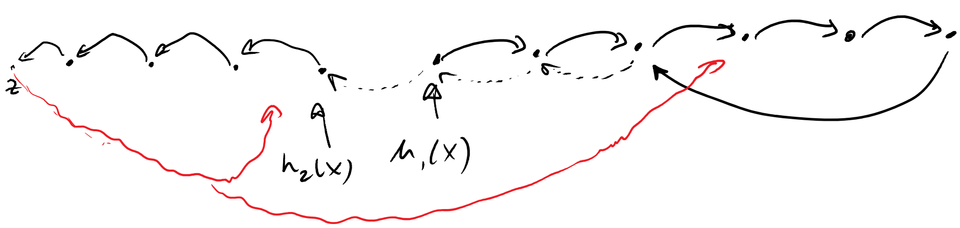
$$\le \sum_{\ell \ge 1} \ell \cdot \left(\frac{1}{3^\ell} \cdot \frac{1}{m}\right) \cdot m = \sum_{\ell \ge 1} \frac{\ell}{3^\ell}$$

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

(*) za předpokladu, že kukaččí graf neobsahuje cyklus. Pokud obsahuje cykly, musíme vynásobit 4.

- smyčku při operaci Insert lze zaříznout 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 posouvají o jednu pozici

[prvky v cyklu se posouvají 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 (± 1) pro $\forall a \in \{1,\dots,m\}$

ALE:

- data jsou málokdy distribuována uniformně náhodně
  → nutno brát náhodné hašovací fce.

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

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

všechny prvky z S se hašují na a.

⇒ pro každou pevně zvolenou hašovací funkci existuje špatná množina. (→ 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 h potřebuje $|U| \cdot \log m$ bitů na popis → neřeší to náš původní problém, protože bychom mohli S uložit triviálním způsobem.

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

... "h" požadavek na $\mathcal{H}$

snadno rozeznán .

- minimální požadavek na $\mathcal{H}$

pro ∀ pevné $x, y \in \mathcal{U}$, $x \neq y$

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

tedy pravděpodobnost kolize dvou prvků taková jako u $\underbrace{\text{náhodné}}_{\text{zuba}}$ fce .

$\mathcal{H}$, která splňuje (*), je <u>univerzální hašovací systém</u>

- <u>silnější požadavek</u>:

pro ∀ pevné $x, y \in \mathcal{U}$, $x \neq y$ a pro ∀ pevné $a, b \in \{1, \ldots, m\}$

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

$\mathcal{H}$, která splňuje (**), je <u>2-univerzální hašovací systém</u>.

(nebo tzv. <u>po dvou nezávislý</u> haš. fce)

- <u>obecněji</u>

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

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

... <u>po k nezávislý hašovací systém</u>

<u>Př</u>: <u>2-univerzální hašovací systémy</u>

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

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

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

- vybrat náhodné $h \in \mathcal{H}$, znamená vybrat náhodně <u>a</u> a <u>b</u> z $\{0, \ldots, m-1\}$

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

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

$$\mathcal{H} = \left\{ h_{A,b} : \{0,1\}^w \to \{0,1\}^k, \quad \begin{matrix} A \in \{0,1\}^{k \times w} \\ b \in \{0,1\}^k \end{matrix} \right\}$$

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

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

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

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

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

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

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

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

nejvyšší bity $1, \dots, k$

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

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

$\times, \cdot$ nad celými čísly

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

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

---

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 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 $p \ge |U|$

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

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

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

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

Dk: dva různé polynomy stupně $\le d-1$ se mohou shodovat v nejvýše d bodech ∎

Př: $p = 2^{89}-1$ ← Mersennovo prvočíslo $\qquad d \le 2^{57}$ → pst. kolize ≤ 1

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

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

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

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

- 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 $T_0, T_1, \dots T_{d-1} : \{0, \dots, m-1\} \to \{0,1\}^\ell$

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

XOR po bitech

$\to$ 2-univerzální hašování

- speciálně 5-univerzální

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

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

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

XOR po bitech

k-univerzální hašování

$x \in \{0, \dots, p-1\}$ $\quad p$ ... 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 $\mathcal{U} \to \{1, \dots, m\}$

$S \subseteq \mathcal{U} \qquad |\mathcal{U}| = n$

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

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