- 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 dostatečný počet úloh.

  celkem 5

  DÚ 1 – do 23. 10., měkký do 30. 10.

- úvod – o čem přednáška bude, o čem nebude

  plán – viz sylabus

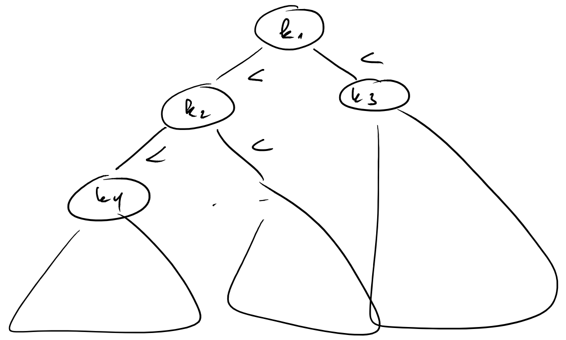
- <u>slovníkový problém</u>: ⟨klíč⟩ ⟨hodnota⟩

  ∈ 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.

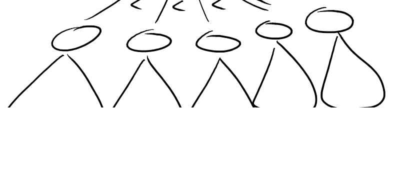
"dynamicky"
splay stromy

(a,b) - stromy

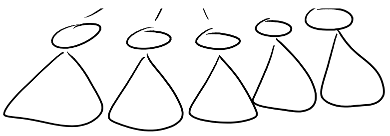
<u>(a,b) - stromy</u>

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

- strom, kde každý vnitřní uzel má alespoň <u>a</u> a nejvýše <u>b</u> synů



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



- 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 d má alespoň a^{d-1} a nanejvýš b^d listů.

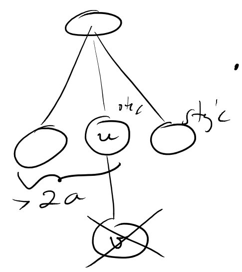
⟹ strom obsahující n hodnot (listů) má hloubku d, kde $\log_b n \le d \le 1 + \log_a n$.

– a,b v závislosti na použití, např. (2,3)-stromy
→ B-stromy a,b ≈ velikost bloku na disku

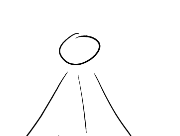
- find (k)
  - projdi strom od kořene, jdi vždy do podstromu, kde by k mohlo být.

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

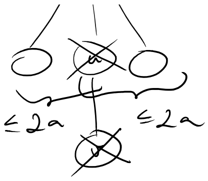
- delete (k)
  - (rekurzivně od listu v obsahující klíč k)
  - pokud otec u mazaného vrcholu v obsahuje $> a$ synů, vymaž z u informaci o v → done



  - pokud otec u společně buď se svým 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 svým levým ani se svým pravým sourozencem nemá více než $2a$ synů, přesuň syny u do jednoho ze strýčků a … (zaktualizuj



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

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

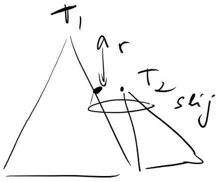
  (za předpokladu, že slití a rozštěpení vrcholu trvá čas $O(1)$)

Operace Join, Split

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

↑ maximální hodnota v T

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

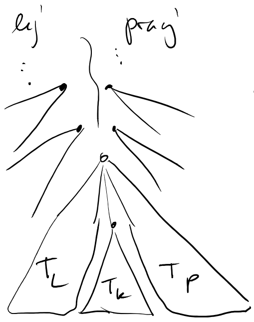


$r = d(T_1) - d(T_2)$

syny kořene T_2 přidej najednou k synům posledního vrcholu na hladině r ve stromě 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 $\geq k$.



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. obsahující 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 v levém zásobníku pospojujeme

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

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

každý přidáme za lichý 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 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 m insertech a l deletech … $O(m + l + \log n)$

předpoklad $b \geq 2a$.

$\longrightarrow$ amortizovaně $O(1)$ štěpení / slučování za operaci

• paralelní verze: $b \geq 2a$

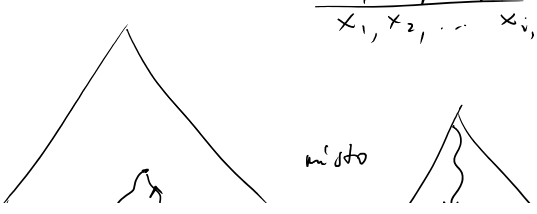
• při insertu při hledání od kořene, rozštěp každý vrchol s b syny (preventivní štěpení)

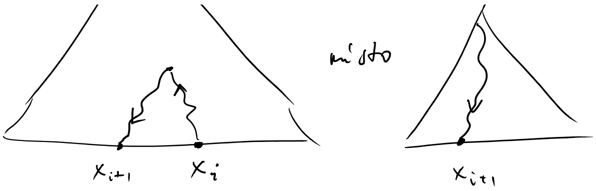
• při deletu při hledání od kořene uprav každý vrchol s a syny buď přidáním syna ze sourozence, nebo slitím se sourozencem.

• 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 naposled (strom s "prstem" – ukazatelem na poslední použitý list)

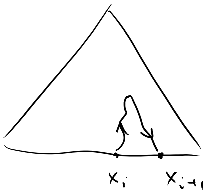
vstupní posloupnost: $x_1, x_2, \ldots x_i, x_{i+1} \ldots$





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

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



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

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

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

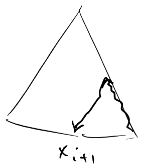
z konkávnosti fce log

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

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

$0 \le F \le n^2$

<u>Alternativně</u> - vždy hledám od nejpravějšího listu



→ cesta nahoru

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

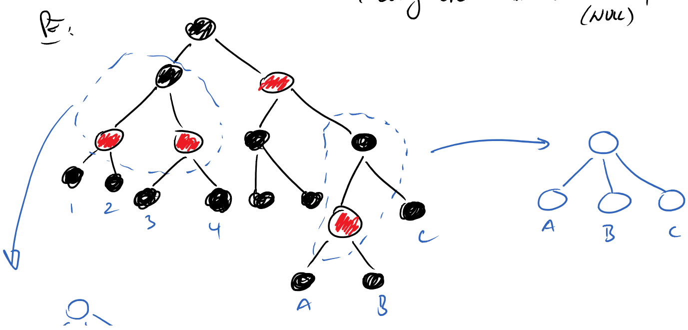
Červeno-černé stromy

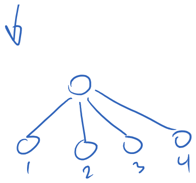
- binární vyhledávací strom, každý vnitřní uzel má stupeň 2.
- každý uzel je buď obarven černě nebo červeně.
- červené uzly mohou být syny pouze černých uzlů (*)
- na každé cestě do listu stejný počet černých vrcholů (**)

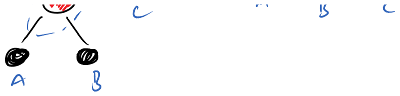
→ hodnoty uloženy ve vnitřních vrcholech

(listy lze nahradit NIL (NULL) pointerem)

Př:



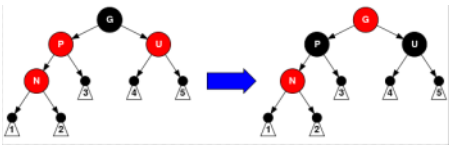




(*)+(**) $\Rightarrow$ Červeno-černé stromy jsou ekvivalentní (2,4)-stromům

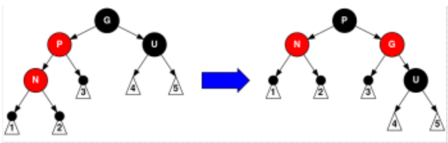
Vyvažování při vložení (INSERT) - vložen červený uzel N

1)



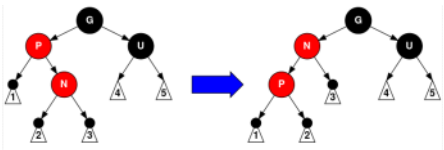
→ problém s příliš mnoha červenými uzly se přenese k otci

2)



Špatné rozložení černé uzly se přerotují

3)



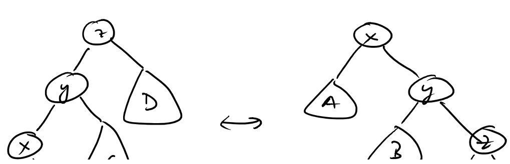
{Obrázky z wikipedie}

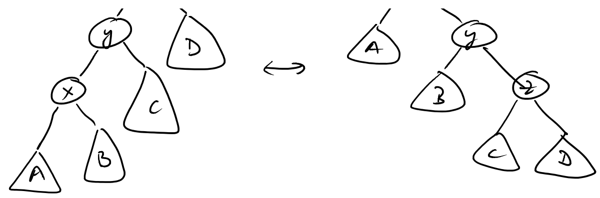
- DELETE - dodatek
- používají se v knihovnách, např. STL v C++

Splay stromy

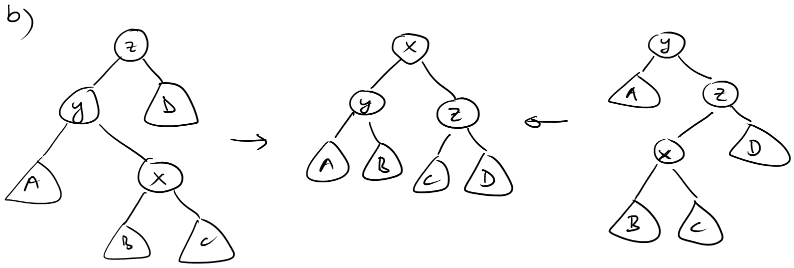
- binární vyhledávací stromy, samoupravující
  - cena za operaci až $O(n)$, ale cena za m operací nejvýše $O(m \log n + n \log n)$ kde n je počet vložených prvků
- operace splay(x), přesune prvek x do kořene pomocí rotací
- po každé operaci: find, insert, ... aplikujeme operaci splay.
- dvojité rotace

a)



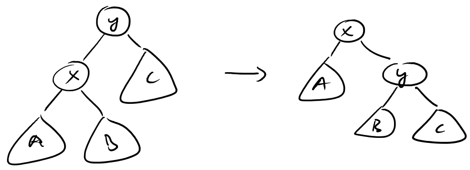


b)



• 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

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

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

• Čas na 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 m operací $m \cdot O(\log n) + O(n \cdot \log n)$.

• amortizovaný čas rotace a), b) $\leq 3(r'(x) - r(x))$

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

r' ... rank po operaci

r ... rank před operací

Dk a): pouze vrcholy x, y, z mění svůj rank

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

$$r'(z) - r(z)$$

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

$$\left.\begin{array}{l} r'(y) \le r'(x) \\ r(y) \ge r(x) \end{array}\right\} \implies \le 2 - r(x) + r'(x) - r(x) + r'(z)$$

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

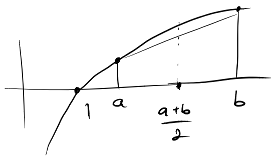
Pozorování: $2 \le 2r'(x) - r(x) - r'(z)$

$\forall\, a+b \le 1$
$\Rightarrow 4ab \le (a+b)^2$
$-(a+b)^2 \le 1$
$\Rightarrow \log ab \le \log 1/4$
$\Rightarrow \log a + \log b \le -2$

$$r'(x) = \log_2 \underbrace{|A| + |B|}_{a} + |C| + |D| + 3$$

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

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



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

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

$$\Rightarrow 2r'(x) - r(x) - r'(z) \ge 2$$

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

Dk b): $t \le 2 + r'(x) - r(x) + r'(y) - r(y) + r'(z) - r(z)$

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

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

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

Dk: stejný jako výše

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

Dk c): $t \le 1 + r'(x) - r(x) + r'(y) - r(y)$

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

$$\le 1 + r'(x) - r(x)$$

$$\le 1 + 3(r'(x) - r(x))$$

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

$r(v)$: původní kořen stromu T

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

a: amort. čas; t: skutečný čas; t_i: i-té rotace; $\Phi(T_i)$: po i-té rotaci; $\Phi(T_{i-1})$: před i-tou rotací

$$\le \sum_i a_i$$

a_i: amort. čas i-té rotace

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

r_i: rank i po i-té rotaci; $+1$: za poslední jednoduchou rotaci typu c)

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

$r'(x)$: konečný; $r(x)$: původní rank x

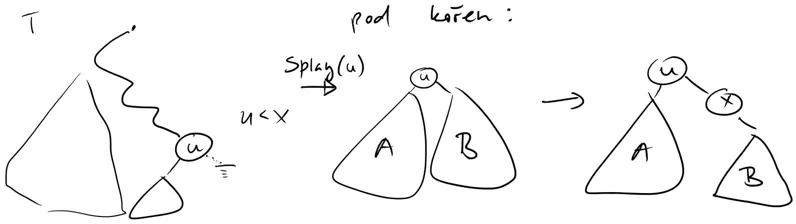
$= 3(r'(x) - r(x)) + 1$

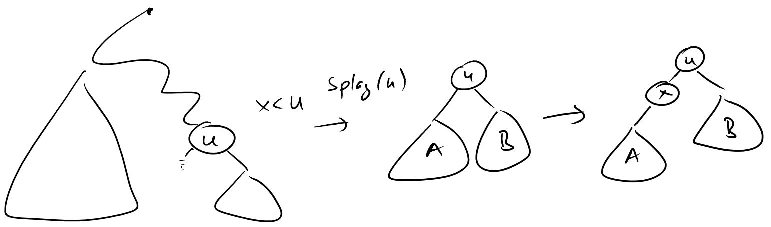
$r'(x)$ = p ... konečný rank x = rank kořene; $r(x)$ = q ... původní rank x ∎

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

$\Rightarrow$ amortizovaný čas na <u>splay</u> $\le 1 + 3\log_2 n$.

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



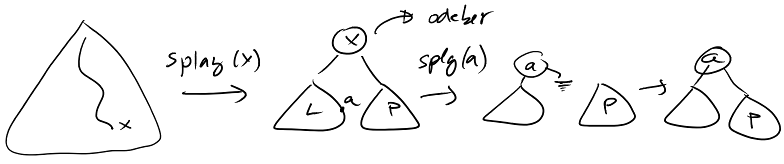


<u>Pozorování</u>: nechť $a, b \in T$, $a < b$ jsou takové, že $\forall c \in T$: $c \notin (a,b)$, $x \in (a,b)$.

pak u je buď <u>a</u> nebo <u>b</u>.

<u>Dk</u>: <u>a</u> i <u>b</u> musí být navštíveny po cestě k <u>x</u>, jinak by jejich vyhledávání selhalo, protože <u>a</u> i <u>b</u> následují stejnou cestu jako <u>x</u>. ∎

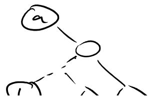
- <u>delete</u>(x) : najdi <u>x</u>, splay(x), odeber x, pokud získám dva neprázdné stromy L a P, najdi nejvyšší prvek <u>a</u> v L, splay(a), připoj P pod <u>a</u>.

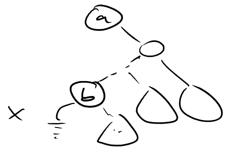


a ... "nejpravější" vrchol v <u>L</u>.
$= \max(L)$

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

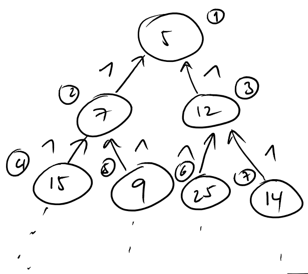
<u>Pozn</u>: Operace z hodiny (a,b) t.ž. $a < x < b$ a $\nexists c \in T$ $a < c < b$ může být též





Haldy

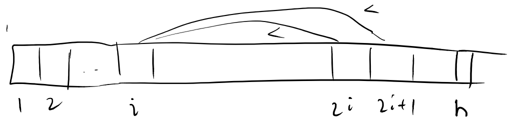
operace
- Insert (x) … vloží x do haldy
- Min … vrátí nejmenší prvek haldy
- Delete-min … odebere nejmenší prvek z haldy



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 dolů 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

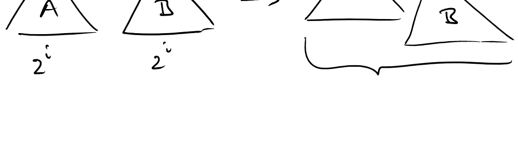
- pamatujeme si, který z těchto stromů obsahuje minimální prvek

zbrklá varianta – nejvýše jeden strom dané velikosti

líná varianta – bez omezení na počet stromů stejné velikosti

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

Dokud existují dva stromy A, B stejné velikosti, spoj je do jednoho:

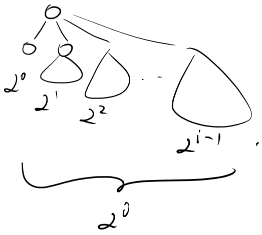


2^{i+1}

- průběžně aktualizujeme ukazatel na strom s minimálním prvkem

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

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 neprovádíme Delete-min.

Slití stromů velikosti 2^i proběhne $\frac{n}{2^i}$-krát ⇒

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, zaktualizuje odkaz na nejmenší

Min ... vrátí nejmenší prvek

Delete-min ... stejně jako ve striktní variantě, odřízneme kořen s nejmenším prvkem, podstromy přidáme do souboru 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

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

C ... vhodná konstanta

$= O(1)$

Min $= O(1)$

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

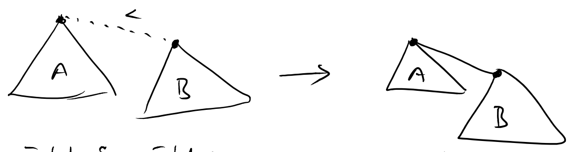
$\leq C \cdot \log n$

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

Fibonacci halda

nové:

- Decrease-key (x, 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 – spojitý 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 vkládaných stromů odznačuj.)

⇒ 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 synů potomků, kteří jsou uspořádaní ve spojovém seznamu, a musí si pamatovat zda již ztratil nějakého potomka.

<u>Struktura stromů:</u>

Fibonacciho čísla
$F_1 = 1$
$F_2 = 1$
$F_3 = 2$
$F_4 = 3$
$F_5 = 5$
$\vdots$

$F_0 = 0$

$F_n = F_{n-1} + F_{n-2}$

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

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

$n \to n+1$

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

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

<u>Dk</u>: indukce na n. $n = 3$ ✓

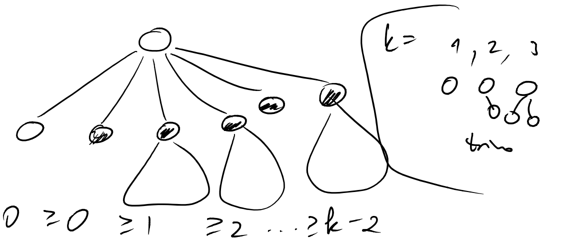
$n \to n+1$

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

• 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:



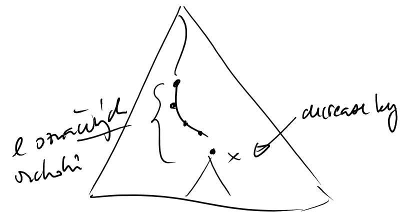
řád $\geq 0 \quad \geq 0 \quad \geq 1 \quad \geq 2 \quad \ldots \geq k-2$

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

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

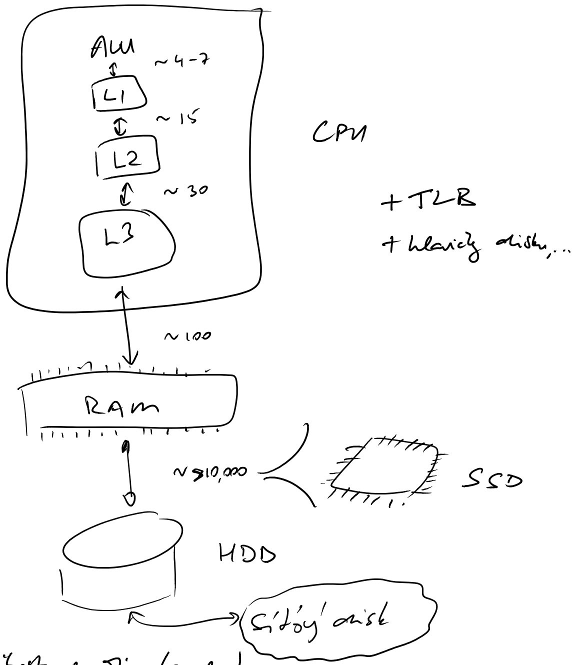
<u>amortizovaná analýza</u>:

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



$$a_{dec.key} = c\cdot \ell + \Phi(H') - \Phi(H)$$
$$= c\cdot \ell + \underbrace{c\cdot \ell - 2\cdot c\cdot \ell}_{\Delta\Phi}$$
$+O(1)$

Paměťová hierarchie

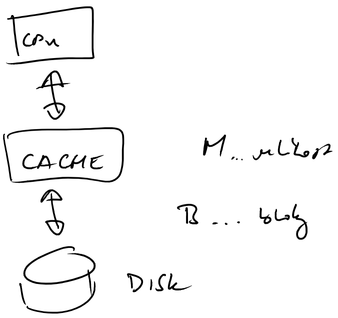


M... velikost paměti (cache)

B... velikost přenášených bloků

→ M/B počet bloků v paměti

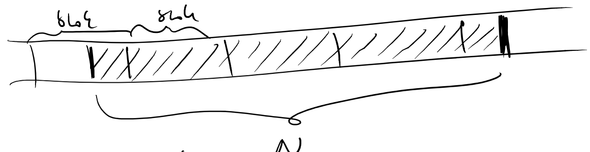
Model externí paměti



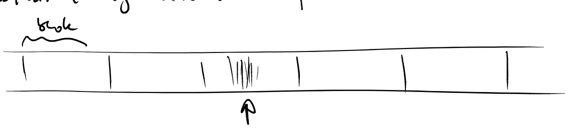
- u algoritmů měříme počet přenesených bloků
- soustředíme se na jednu úroveň a ostatní ignorujeme
  - → algoritmus optimalizovaný pro konkrétní úroveň

Př: • Průchod souvislého kusu dat délky N.

- $\lceil M/B \rceil + 1$ přenos

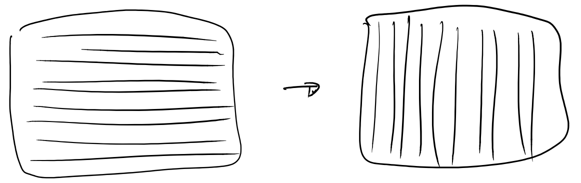


- binární vyhledávání v poli velikosti N



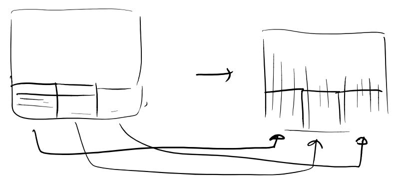
- $\log N - \log B$ přenosů

• transpozice matic



⇒ 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 $\lceil M/B \rceil + 1$ přenosů

– lépe to nejde

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

transpozice matic:

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

$\tfrac{1}{2}$ $\tfrac{1}{2}$

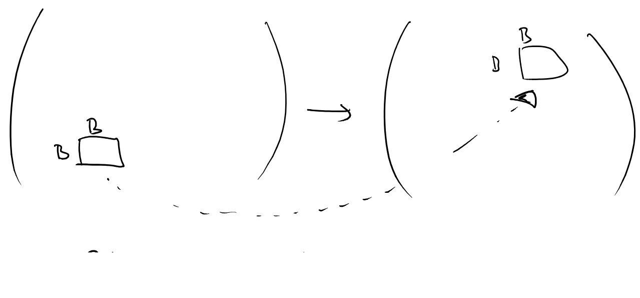
rekurzivně opakuj

\# operací $O(n^2)$

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

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

Dk.:



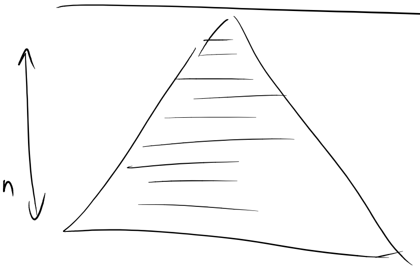
$O(B)$ IO operací, jakmile se rekurze dostane na velikost matice $l \times l$

$l = O(B)$

$\frac{n^2}{B^2}$ 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)

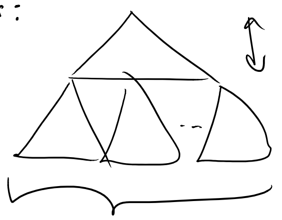


úplný binární strom

$\log n$

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

• van Emde Boas:



$\frac{1}{2} \log n$

$\frac{1}{2} \log n$

$\sqrt{n}$ 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

Tvrz: $\# \text{výpadků LRU} \le \dfrac{n_{LRU}}{n_{LRU} - n_{OPT}} \cdot \# \text{výpadků OPT} + n_{LRU}$

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: $|\{s_i,\ t_1 < i < t_2\}| \ge n_{LRU}$)

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 $\Rightarrow$ tvrzení ■

Dk: rozsekejme $s_1, s_2, \ldots$ 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.
  $\Uparrow$
  buď jsou všechny výpadky různé nebo se užije předchozí tvrzení

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

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

$\Rightarrow$ Pokud $n_{LRU} = 2 n_{OPT}$ pak LRU se dostává optimálně až na konstantu 2!

Alternativní důkaz:

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

$\Rightarrow$ LRU má v každém úseku $\le n_{LRU}$ výpadků
OPT má v každém úseku $\ge n_{LRU} - n_{OPT}$ výpadků ■

ANKETA

Hašování

Slovníkový problém ... universum U

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

chceme reprezentovat S

Operace - Find (MEMBER)

- Insert
- (DELETE)

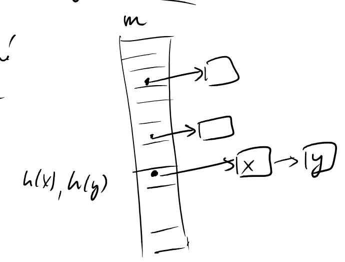
- triviálně ... pole velikosti U, mapování 1-1
- lépe ... hašování do pole velikosti m.

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

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

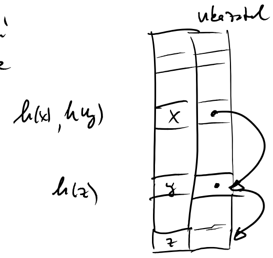
- způsoby řešení:

- separované řetězce



seznam / vyhledávací strom

- srůstající řetězce

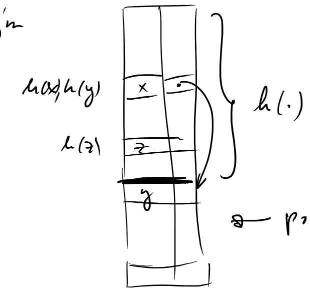


LISH ... nový prvek přidán na konec seznamu

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

(seznam není obousměrně propojený)

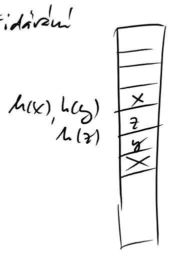
- s pomocným polem



pomocná část, když dojde pomocná část, začnu ukládat přetékající prvky do hlavního pole

→ srůstání

- lineární přidávání



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

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

- dvojité hašování - podobně jako lineární přidávání, ale zkoušíme pozice

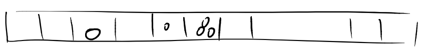
$$h_1(x) + i\, h_2(x), \quad i = 0, 1, 2, \dots$$

h_1, h_2 jsou různé hašovací funkce

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

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

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



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

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

$k \geq 1$

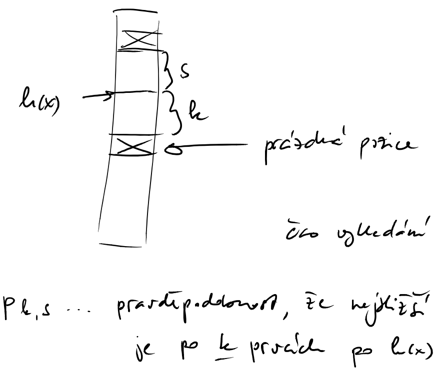
$$\underset{k\ll n}{\approx} \frac{n^k}{k!}\cdot\frac{1}{n^k}\cdot\frac{1}{e} = \frac{1}{e\cdot k!}$$

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

– odpovídá to situaci, kdy bych při hashování bral náhodně zvolenou množinu funkcí

• lineární přidávání

analýza: hledám prvek x, předpokládám h distribuuje prvky zcela náhodně



čas vyhledání $\leq k$

$p_{k,s}$ ... pravděpodobnost, že nejbližší volné 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 napasovat do bloku velikosti $k+s$ z celkového počtu m pozic.

. ...

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

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 $m \ge 3n$ $\qquad e = 2.718\ldots$

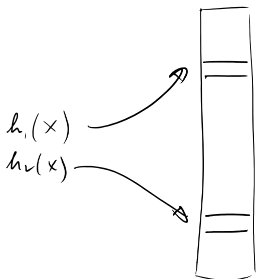
$$\le \left(\frac{e}{3}\right)^{k+s}\cdot\frac{1}{\sqrt{(k+s)\,2\pi}}$$

očekávaná doba vyhledávání $\le \sum_{k\ge1} k\cdot \Pr[\text{vyhledání trvá čas } k]$

$$\le \underbrace{\sum_{k\ge1} k\cdot \underbrace{\sum_{s\ge0}\left(\frac{e}{3}\right)^{k+s}\cdot\frac{1}{\sqrt{(k+s)\,2\pi}}}_{\le O\left(\left(\frac{e}{3}\right)^k\right)}}_{O(1)} = O(1) \qquad \blacksquare$$

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

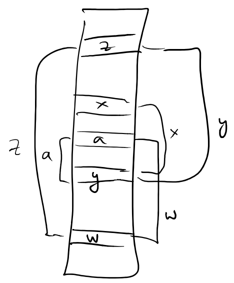
→ kukaččí hašování



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

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

Analýza pro $m = 6n$, kukačkové hašovací funkce
dobře i pro $m \approx 2.3n$

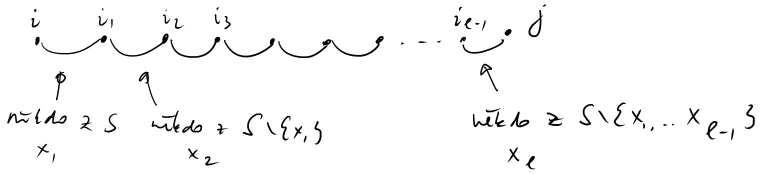
<u>Kukačkový graf</u>: pozice v tabulce ... vrcholy
$\{h_1(x), h_2(x)\}$ ... hrany
$x \in S$

$m = 6n$
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 spojené cestou délky ℓ v kukačkovém grafu $\leq \left(\frac{2n}{m}\right)^{\ell} \cdot \frac{1}{m} \leq \frac{1}{3^{\ell}} \cdot \frac{1}{m}$.

<u>Dk</u>:



$$\Pr[d(i,j) = \ell] \leq \Pr\left[\exists x_1, \dots x_\ell, \exists i_1, \dots i_{\ell-1}: h(x_1) = \{i, i_1\} \,\&\, \dots\, h(x_\ell) = \{i_{\ell-1}, j\}\right]$$

$$\leq \frac{2^{\ell} \cdot n \cdot (n-1) \cdot \ldots \cdot (n-\ell+1) \cdot m^{\ell-1} \cdot m^{2(n-\ell)}}{m^{2n}}$$

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

výběr $x_1, \dots x_\ell$

výběr $i_1, \dots i_{\ell-1}$

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

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

<u>Dk Tvrzení 1</u>:

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

dle lemmatu cesta z i do i délky ℓ.

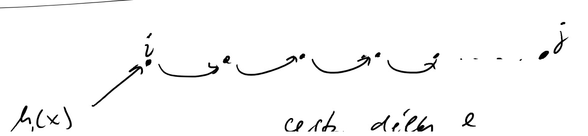
$$\Pr[i \text{ je obsažená v cyklu}] \leq \sum_{\ell \geq 1} \frac{1}{3^{\ell}} \cdot \frac{1}{m} \leq \frac{1}{m} \cdot \frac{1}{2}$$

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

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

⇒ Očekávaný počet přehašování ≤ 1 během <u>n</u> operací insert.

<u>Doba na operaci Insert</u>



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

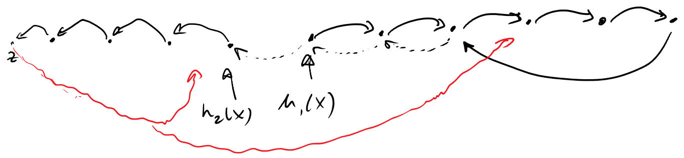
$$\le \sum_{l\ge 1} l \cdot \left(\frac{1}{3^l} \cdot \frac{1}{m}\right) \cdot m = \sum_{l \ge 1} \frac{l}{3^l}$$

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

(*) za předpokladu, že kukačkový graf neobsahuje cyklus. Pokud obsahuje 1 cyklus, musíme vynásobit 4.

- smyčku při operaci Insert lze zařídit 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 tam a zpět]

Výběr hašovací fce

- pokud jsou data distribuovaná 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 distribuovaná uniformně náhodně
  → musíme brát náhodné hašovací fce.

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

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

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, t.j. $\forall x \in U$; $h(x)$ 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 uchovat triviálním způsobem.

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

- minimální požadavek na $\mathcal{H}$

pro ∀ pevné $x, y \in U$, $x \neq y$

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

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

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

- silnější požadavek:

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

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

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

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

- obecněji:

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

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

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

Pro správnou funkci vyžaduje:

- lineární přidávání – 5-univerzální $\mathcal{H}$
- kukačkové hašování – $\lg n$-univerzální $\mathcal{H}$ nebo tabulkové hašování (viz níže)

<u>Perfektní hašování</u>

$\mathcal{H}$ .. univerzální hašovací systém $U \to \{0, \dots, m-1\}$

pevná $S \subseteq U$, $|S| = n$

$h \in \mathcal{H}$ hašuje S <u>perfektně</u> pokud
$\forall (x,y) \in S^2,\ x \neq y \quad h(x) = h(y)$

• Pokud $m \geq kn^2$, pro $k>1$, pak

$\Pr_{h\in\mathcal{H}}[h$ hašuje S perfektně$] \geq 1-\frac{1}{k}$

Dk: $C_h = |\{(x,y)\in S^2,\ x\neq y,\ h(x)=h(y)\}|$

... počet kolizí

$$\mathop{\mathbb{E}}_{h\in\mathcal{H}}[C_h] = \sum_{\substack{(x,y)\in S^2\\ x\neq y}} \underbrace{\Pr_{h\in\mathcal{H}}[h(x)=h(y)]}_{\text{pro univerzální } \mathcal{H}\ \leq \frac{1}{m}} \leq \frac{n^2}{m}$$

linearita střední hodnoty

průměrný počet kolizí

$\mathbb{E}_{h\in\mathcal{H}}[C_h] \leq \frac{n^2}{m} \leq \frac{1}{k}$ tj. průměrný počet kolizí je $\frac{1}{k} < 1$

$\Rightarrow \Pr_{h\in\mathcal{H}}[$alespoň jedna kolize tj. $C_h \geq 1] \leq \frac{1}{k}$

(jinak by průměrný počet kolizí byl větší než $1/k$)

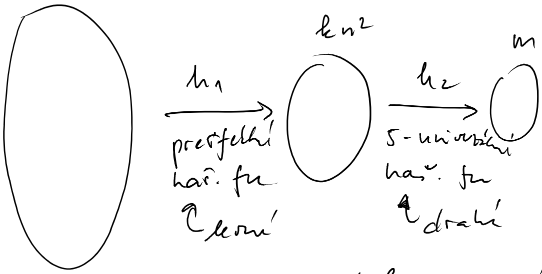
= "Markovova nerovnost"

$\Rightarrow \Pr_{h\in\mathcal{H}}[h$ je perfektní, tj. $C_h = 0] \geq 1-\frac{1}{k}$

$\boxed{\text{B}}$

• Perfektní hašování lze použít jako mezikrok pro hašování s lepšími vlastnostmi

U ... obrovská velikost, např. řetězce



$h_1 \in \mathcal{H}_1$ ... univerzální haš. systém

$h_2 \in \mathcal{H}_2$ 5-univerzální haš. systém

$h: U \to \{0,\dots,m-1\}$ získám

jako $h(x) = h_2(h_1(x))$

[Pokud h_1 je perfektní, vlastnosti určuje h_2]

↳ což je s velkou pravděpodobností

• Zcela náhodná hašovací funkce vyžaduje

$|U| \log m$ prostor na uložení

Kompromis nabízí tabulkové hašování

Tabulkové hašování

• obecně $x_1, x_2 \dots x_d \in \{0, \dots, W-1\}$

náhodné tabulky $T_1, \dots T_d : \{0, \dots, m-1\} \to \{0,1\}^k$

$$T_1[x_1] \oplus T_2[x_2] \oplus T_3[x_3] \dots \oplus T_d[x_{d-1}]$$

XOR po bitech

$\longrightarrow$ 2-univerzální hašování

Př: $\mathcal{U} = \{0,1\}^{32}$ $x \in \mathcal{U}$ interpretuji jako $x_1 \dots x_4 \in \{0,1\}^8$

$$\underbrace{|\overset{8}{x_1}|\overset{8}{x_2}|\overset{8}{x_3}|\overset{8}{x_4}|}_{x}$$ $k = 16$

$$h(x) = T_1[x_1] \oplus T_2[x_2] \oplus T_3[x_3] \oplus T_4[x_4]$$

místo $16 \cdot 2^{32}$ bitů pro zcela náhodné

$8GB$ hašování stačí $4 \cdot 16 \cdot 2^8 \approx 2KB$.

speciální případ – 5-univerzální:

$|U| = 2^w$

$$\underbrace{|x_1|x_2|}_{x}$$

$x \in \{0, \dots |\mathcal{U}| - 1\}$

$x_1 = x / \sqrt{|\mathcal{U}|}$, $x_2 = x \bmod \sqrt{|\mathcal{U}|}$

$T_1, T_2 : \{0, \dots, \sqrt{|\mathcal{U}|} - 1\} \to \{0,1\}^k$

$T_3 : \{0, \dots, 2\sqrt{\mathcal{U}}\} \to \{0,1\}^k$

$$h(x) = T_1[x_1] \oplus T_2[x_2] \oplus T_3[x_1 + x_2]$$

Př: 2-univerzální hašovací systémy

1) p ... prvočíslo $\mathcal{U} \subseteq \{0, \dots, p-1\}$

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

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

· vybrat náhodné $h \in \mathcal{H}$, znamená vybrat náhodně $\underline{a}$ a $\underline{b}$ z $\{0, \dots, p-1\}$

$\to$ potřebuji $2 \log p$ bitů na zapamatování h.

• chci $h : \mathcal{U} \to \{0, \dots, m-1\}$ $m \leq |\mathcal{U}| < p$ prvočíslo

$$h_{a,b}(x) = ((ax+b) \bmod p) \bmod m$$

kde $a, b \in \{0, \dots, p-1\}$ jsou zvoleny náhodně

Platí: $\forall x \neq y \in \mathcal{U}$; $\forall s, r \in \{0, \dots, m-1\}$

$$\frac{1}{4m^2} \leq \Pr_{a,b}[h_{a,b}(x) = r \; \& \; h_{a,b}(x) = s] \leq \frac{4}{m^2}$$

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

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

kde $h_{A,b}(x) = \underset{\smile}{A}x + b$

násobení matic nad $GF[2]$

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

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

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

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

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

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

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

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

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

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

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

$\times, \cdot$ nad celými čísly

2), 3) nepraktické

4) rychlé praktické, nepotřebuje dělení, pouze jedno násobení

1) často používané, potřebuje dělení

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

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

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

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

- pro d sudé lze též použít:

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

$\to$ 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 $p \geq |U|$

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

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

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

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

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

Př: $p = 2^{89}-1$ ← Mersennovo prvočíslo $\qquad d \leq 2^{57} \to$ prst. kolize $\leq \frac{1}{2^{32}}$

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

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

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

• Pokud $d < \frac{p}{m}$ pak je prst. kolize $\leq \frac{2}{m}$.

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

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

$$\to \quad y = (y \,\&\, p) + (y \gg a) \pmod p$$

---

k-univerzální hašování

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

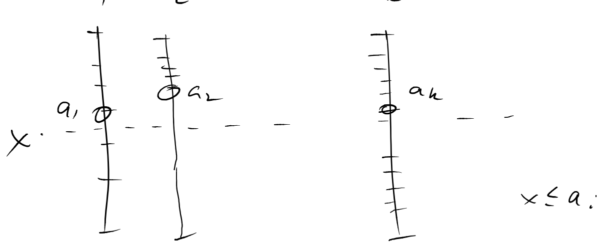
$a_0, \dots a_{k-1} \in \{0, \dots, p-1\}$ náhodné

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

$\to$ k-univerzální

Problém: Množiny $L_1, L_2, \dots L_k \subseteq \mathbb{N}$, $|L_i| \leq n$. Chci d.s., která na dotaz $x \in \mathbb{N}$, vrátí nejbližší větší nebo rovno číslo z každé množiny L_i.

$\to$ ideálně - čas $O(\log(n) + k)$
prostor $O(\sum |L_i|)$



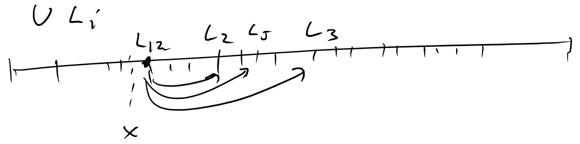
jednoduché řešení

1) každou množinu L_i reprezentuji setříděným polem $\to$ čas $O(k \cdot \log n)$
prostor $O(\sum |L_i|)$

2) množiny sjednotím, setřídím a pro každý prvek ve sjednocení si pamatuji ukazatel

na nejbližší výskyt prvku z každé množiny, tj. k ukazatelů na prvek

$\cup L_i$



čas $O(\log n + k)$

prostor $O(k \cdot \sum_i |L_i|)$

Řešení – kaskádování pokližk ("fractional cascading")

definujme: $M_k = L_k$

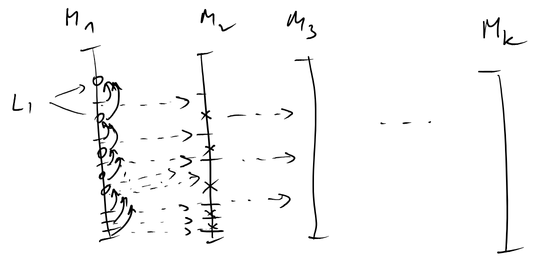
a pro $i < k$, $M_i = L_i \cup \{$každý druhý prvek z $L_{i+1}\}$

- $|M_i| \leq \sum_{j \geq i}^{k} \frac{|L_j|}{2^{j-i}}$ Dk: indukce na i.

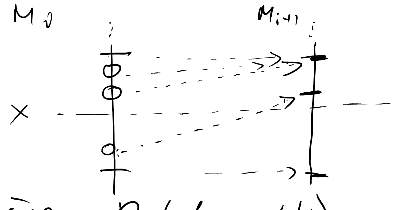
$$\Rightarrow \sum_{i=1}^{k} |M_i| \leq \sum_{i=1}^{k} \sum_{j \geq i}^{k} \frac{|L_j|}{2^{j-i}} \leq \sum_{i=1}^{k} \sum_{j=1}^{k} \frac{|L_i|}{2^j}$$

$$\leq 2 \sum_{i=1}^{k} |L_i|$$

- místo L_i reprezentuji setříděným polem M_i
  + každý prvek z M_i má ukazatel na nejbližší větší prvek z L_i & na nejbližší větší nebo rovný prvek v reprezentaci M_{i+1}



- pomocí binárního vyhledávání / bin. vyhledávacího stromu, nalezneme mezi kterými prvky M_1 leží x. To mi dovoluje nalézt nejbližší prvek k x v L_1, přeskočím do M_2 a nahlédnu interval pro x pomocí jednoho porovnání



→ čas $O(\log n + k)$

bin. hledání v M

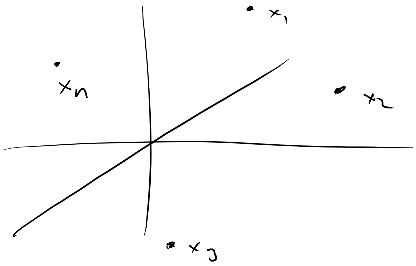
pokračuji pro další M_i až do $i = k$

průchod přes k polí M_i

prostor $O(\sum_i |M_i|) = O(\sum |K_i|)$. ✓

Vícedimenzionální data

bod v $\mathbb{R}^d$

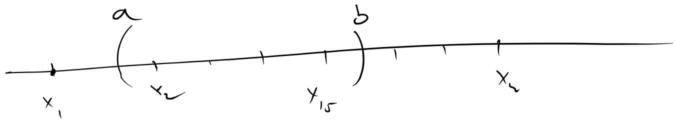


- vyhledávání, přibližné vyhledávání
- intervalové dotazy – který bod je v obdélníku $(x_1,x_2)\times(y_1,y_2)\times \dots (z_1,z_2)$

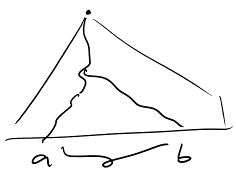
– databáze, výpočetní geometrie…

→ kd-stromy

- $d=1$ dotaz (a,b)



bin. vyhl. strom



body mezi a a b

- u vyváženého stromu $O(\log n + k)$ čas na dotaz

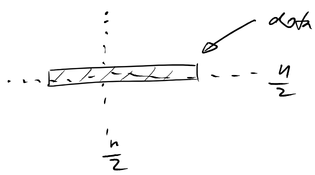
počet bodů výstupu

- počítací dotaz (zajímá nás pouze počet bodů v daném intervalu)

čas $O(\log n)$, pokud si ve vnitřních vrcholech udržuji počet listů v podstromu

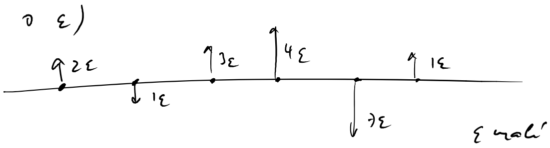
- $d=2$ … 2-dim

Př.:



- BÚNO: žádné dva body nemají totožnou žádnou souřadnici (každý bod můžeme posunout o ε.)

BÚNO: žádné dva body nemají [illegible] souřadnici (každý bod můžeme posunout o ε)

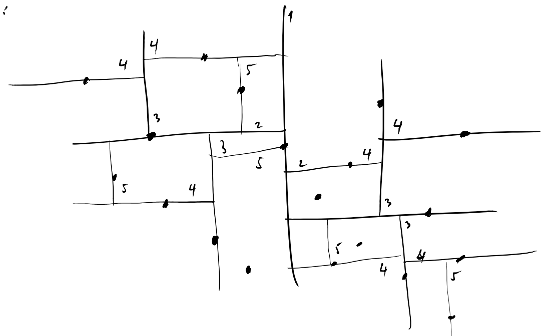


ε malé

2-dim kd-strom

- v lichých vrstvách se rozhodujeme podle x-ové souřadnice, v sudých podle y-ové.

Př:



- výška stromu je $O(\log n)$ (přesně $\lceil \log n \rceil + 1$)

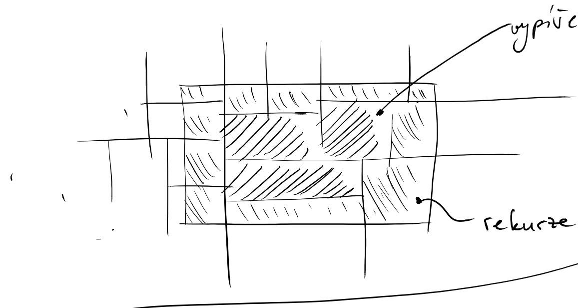
- každý uzel odpovídá určité oblasti v rovině (obdélník / nekonečný obdélník) tu lze spočítat při průchodu od kořene

- Find (vrchol v, interval R)

Vstup - v ... vrchol podstromu kd-stromu, R ... interval zájmu
Výstup - vypíše všechny body v intervalu R, které jsou v podstromu v

1) Pokud je v list, vypiš příslušný bod pokud leží v R
2) Pokud oblast levého podstromu v_L je celá obsažena v R, vypiš všechny její body
   jinak pokud tato oblast protíná R, $\to$ Find(v_L, R)
3) Pokud oblast pravého podstromu v_R leží celá v R, vypiš všechny její body
   jinak pokud tato oblast protíná R $\to$ Find(v_R, R).
4) End.

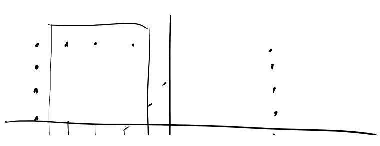
Časová složitost:



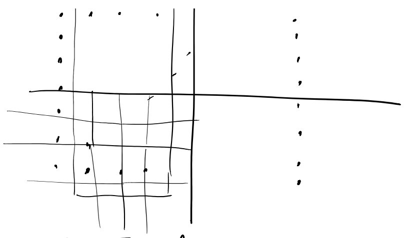
vypíšeme celé $O(k)$ čas

rekurze $O(\sqrt{n})$ čas $+ O(k)$

Proč: Př:



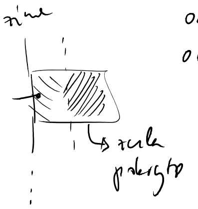
v rovině: ...

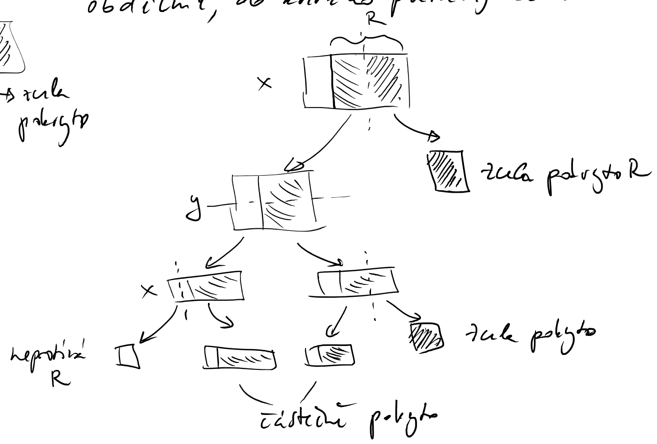


mřížka $\sqrt{n}\times\sqrt{n}$, každý bod posunut o zanedbatelné ε.

- čas $\Omega(\sqrt{n})$ i když obdélník může být prázdný ($k=0$) (hubený obdélník )

- uvažujme uzel hranicí a neúplně pokrytý obdélník R k ní přilehlý. Rozhodování podle os x odštípí obdélník, který je zcela pokryt a jeden obdélník, do kterého pokračujeme v rekurzi. → nebo disjunkt s R





→ v každé druhé vrstvě se větvíme na 2, v ostatních vrstvách se nevětvíme

→ $\frac{1}{2}\log n$ větvení → velikost větvícího podstromu $\approx 2^{\frac{1}{2}\log n} = \sqrt{n}$.

→ čas $O(\sqrt{n})$ na větvení + $O(k)$ na výpis bodů ve zcela pokrytých obdélnících.

$d>3$ střídám větvení dle jednotlivých dimenzí

čas $O(n^{1-\frac{1}{d}}+k)$ na dotaz, prostor $O(n)$.

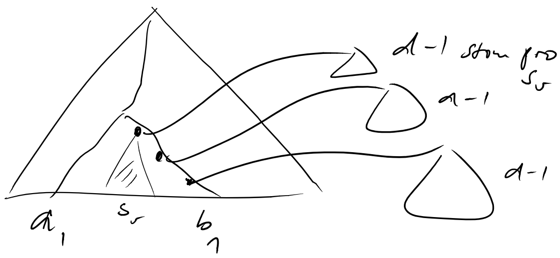
("stejná" analýza – až na každou d-tou vrstvu větvení na 2)

Intervalové stromy ("range trees")

$d=1$ ... stejné jako výše.

$d>1$

binární vyhledávací strom podle první souřadnice, každý uzel ukazuje na intervalový strom dimenze $d-1$, kam jsou uloženy všechny prvky odpovídajícího podstromu podle první souřadnice.

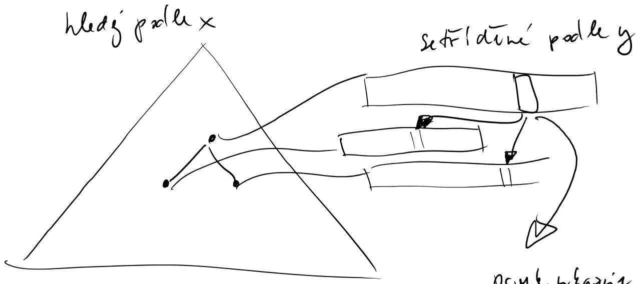


vyhledávání $O(\log^d n)$

prostor $O(n \log^{d-1} n)$

hledání pro $d=2$ lze zlepšit na $O(\log n)$ pomocí techniky kaskádování

→ pro obecné d čas $O(\log^{d-1} n)$ na intervalový dotaz



prvek ukazuje na nejbližší prvek v pomocné struktuře (setříděné podle dalšího y) pro každého ze synů

→ při průchodu hlavním stromem podle x se zároveň navigujeme v pomocném poli podle y.