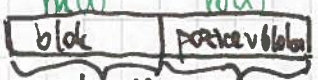
Celočíselné datové struktury ... vše na RAMu

Máme celočíselné universum $\mathcal{U} = \{0 \dots U\}$, $w = \log U$

- obyčejné množiny → hešování (třeba kukačka) – dotaz $O(1)$ w.c., update $O(1)$ průměrně amort.
- množiny s předchůdcem (Pred) a následníkem (Succ) → tato přednáška

van Emde-Boasovy stromy [1975] – zde v pozdější reformulaci

- universum rozdělíme na $\sqrt{U}$ bloků velikosti $\sqrt{U}$ — předpokládáme $w = 2^k$, tedy $U = 2^{2^k}$
  - binární pohled: 

  ... rozklad v $O(1)$ na RAMu
  značení: $pair(hi(x), lo(x)) = x$

Def: vEB(U) si pamatuje pro množinu $S \subseteq \mathcal{U}$:

- $\min S$ (zvlášť!) ... $S' := S \setminus \{\min S\}$ } pokud je strom prázdný, $min = max = \emptyset$
- $\max S$ (kopie)
- přihrádky $P_0, \dots, P_{\sqrt{U}-1}$ ... do nich rozdělíme S'
  - uvnitř P_i je $vEB(\sqrt{U})$ s dolními polovinami prvků $P_i := \{lo(x) \mid x \in S' \wedge hi(x) = i\}$
- sumární strom: $vEB(\sqrt{U})$ pro $\{i \mid P_i \neq \emptyset\}$

Hloubka vnoření je $O(\log \log U)$

Find(x): triviálně – zanořuji se do přihrádek, stojí $O(\log \log U)$

Succ(x):
1. Rozdělíme x na (i, j)
2. Pokud $x < min$, vrátíme min.
3. Pokud $x \geq max$, vrátíme $\emptyset$.
4. Pokud $P_i = \emptyset$ nebo $j \geq P_i.max$:
   $i' \leftarrow Sum.Succ(i)$
   Vrátíme $P_{i'}.min + i'\sqrt{U}$] $pair(i', P_{i'}.min)$
5. Jinak:
   Vrátíme $P_i.Succ(j) + i\sqrt{U}$] $pair(i, P_i.Succ(j))$

($P_i.min = \emptyset$; na každé úrovni $O(1)$ práce ⇒ celk. čas $O(\log \log U)$; ← jelikož $x < max$, i' určitě existuje)

Insert(x):
1. Pokud $min = \emptyset$, $min \leftarrow x$, $max \leftarrow x$ a skončíme.
2. Pokud $x < min$: $x \leftrightarrow min$
3. Pokud $x > max$: $max \leftarrow x$
4. Rozdělíme x na (i, j).
5. ~~Pokud $P_i \neq \emptyset$: P_i.Insert(j)~~
5. Pokud $P_i = \emptyset$:
   [Založíme prázdnou P_i.]
   Sum.Insert(i)
6. P_i.Insert(j)

(1'. Pokud $x = min$, skončíme [hodnota už ve stromu je]

Pokud proběhne 5. krok (zakládáme P_i), 6. krok je triviální ⇒ jen 1 netriv. rekurzivní volání ⇒ celkově $O(\log \log U)$)

Delete(x):
1. Pokud min=∅, skončíme
2. Pokud x=min:
   Je-li Sum.min=∅: min,max ← ∅ a skončíme
   Jinak min ← $P_{Sum.min}$.min, x ← min
3. Rozdělíme x na (i,j). + Sum.min$\sqrt{U}$
4. Pokud P_i=∅, skončíme.
5. P_i.Delete(j)
6. Pokud P_i=∅: Sum.Delete(i)
7. Pokud Sum.max≠∅: max ← $P_{Sum.max}$.max + Sum.max·$\sqrt{U}$
   Jinak max ← min.

na ∀ úrovni rekurze čas O(1),
opět je ≤1 větev rekurze netriviální
⇓
$O(\log\log U)$

- Dobré zprávy: $O(\log\log U)$ na operaci
- Špatné: struktura zabere paměť $O(U)$ a tu je potřeba na začátku vynulovat!

Trik: Pokud nás RAM dovoluje číst neinicializovanou buňku (byť nezaručuje nic o hodnotě), lze inicializaci simulovat.

- Myšlenka: Udržujeme si seznam buněk, do nichž jsme už zapsali.
  Při čtení se podíváme, je-li buňka na seznamu, jinak vrátíme 0.
- 1. pokus: M[0...] - paměť původního RAMu
  I[0...N] - seznam inicializovaných buněk
  N - # inic. buněk
  } poskládáno v paměti proloženě

  Read i Write stojí O(N). ... pomalé.
- Zrychlení: přidáme X[...] - index k poli I
  - pokud je i-tá buňka inicializovaná, pak I[X[i]]=i.
  - jinak X[i] neinicializované

  Read(i): x ← X[i]
  Pokud x<N a I[x]=i, vrátíme M[i].
  Jinak vrátíme 0.

  Write(i,y): M[i]=y
  x ← X[i]
  Pokud x<N a I[x]=i, skončíme.
  I[N] ← i
  X[i] ← N
  N++

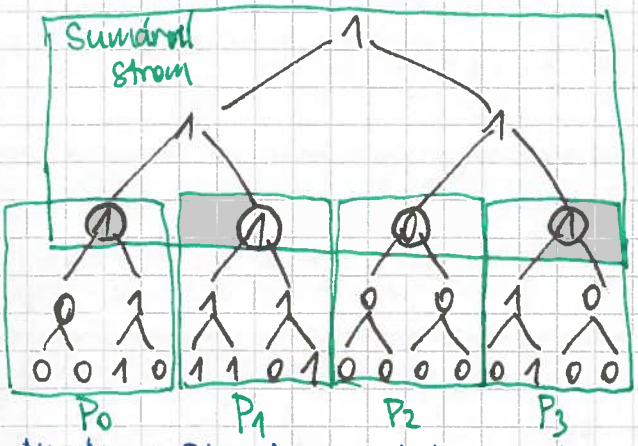
  } O(1)

Věta: Kterýkoli program se složitostí časovou T(n) a prostorovou S(n), který předpokládá paměť inicializovanou nulami, lze transformovat na program se složitostí O(T(n)), O(S(n)), který to nepředpokládá.

[potenciální chyták: výstup v neinicializované buňce] → vstup je potřeba na začátku konvertovat, výstup na konci taktéž → proto ⊗

(dá se tomu vyhnout za cenu složitější konstrukce, pokud program nenechává kusy výstupu neinicializované)

Stromová interpretace vEB (à la intervalový strom)



$O(\log U)$ hladin

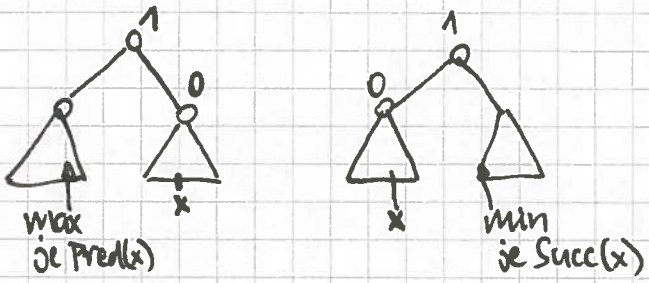
vnitřní vrcholy obsahují OR listů v podstromu

cesta kořen-list je monotónní (1...10...0)

← v listech indikátorový vektor množiny S

Min/Max: Jdu z kořene, pokud jsou pode mnou 2 jedničky, preferuji levou / pravou.

Pred/Succ: Na cestě kořen-list hledáme přechod 1-0 → binárně v $O(\log \log U)$



pro $x \notin S$ dostaneme snadno Pred xor Succ

↓

stačí si prvky S udržovat v seznamu

a ve vnitřních vrcholech mít min/max podstromu

Update trvá $O(\log U)$ ... klasický vEB z toho vybruslil ukládáním minima zvlášť

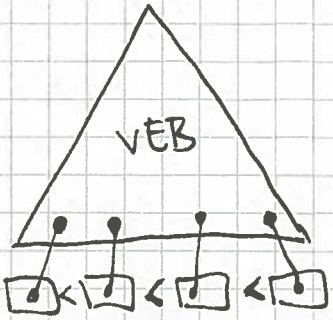
→ použijeme indirekci

Indirekce [Willard 1983]

Myšlenka: množinu rozdělíme na bloky velikosti $\sim B$, $\forall$ blok má reprezentanta v glob. stromu

(či spíše setříděnou posl. prvků)

B nastavíme na $\Theta(\log U)$



uvnitř bloků BVS ... operace v $O(\log B) = O(\log \log U)$

Find/Pred/Succ:

- globálním stromu mi řekne 2 bloky, kde se x může vyskytovat } $O(\log \log U)$
- ↓
- ≤2 dotazy na BVS } $O(\log \log U)$

- Update je založený na dělení / slučování bloků.

Invariant: Blok má hustoty $[\frac{1}{4}, 1]$ — tedy # prvků leží mezi $\frac{1}{4}B$ a B

Výjimka: ∃ jeden blok

Insert: pokud blok přeplním, rozdělím ho na 2 bloky o hustotě $\frac{1}{2} \pm \varepsilon$ ... čas $\Theta(B)$

změny reprezentantů → $O(1)$ updatů globálního stromu

Delete: Pokud hustota klesne pod $1/4$, podíváme se na souseda:

① soused má hustotu $[\frac{5}{8}, 1]$: půjčíme si od něj $\frac{1}{8}$ → náš blok má $\frac{3}{8}$, soused $[\frac{4}{8}, \frac{7}{8}]$

② soused má $[\frac{2}{8}, \frac{5}{8}]$: sloučíme se s ním → $[\frac{4}{8}, \frac{7}{8}]$

Opět $\Theta(B)$ času + $O(1)$ updatů glob. struktury.

! co když nemáme representanta? ... jen ho škrtneme a zůstane v bloku

Amortizace: Po rozdělení/sloučení je hustota alespoň o $1/8$ vzdálena od mezí
⇒ děje se to nejvýše 1× za $B/8$ operací ⇒ amortizovaně $O(1/B)$ krát

- tedy amort. $\Theta(1)$ času na op. + $\Theta(1/B)$ updatů glob. struktury.
- pro vEB volíme $B = \log u$ → čas $O(\log\log u)$ amort. na update i čtení
  ... ale prostor stále $\Theta(u)$!

[při velké změně $\log u$ přebudujeme celou strukturu]

x-fast stromy [Willard 1984]

- vyjdeme z "intervalového" vEB
- uložíme do hešovací tabulky pozice všech jedniček [to jsou prefixy bít. zápisů prvků z S]
  (kukačka či dyn. perf. hešování)
- prostor $O(n \cdot \log u)$ w.c.
- čtení místo stromu čte heš. tabulku → $O(\log\log u)$ w.c.
- zápisy updatují $O(\log u)$ položek heše → $O(\log u)$ průměrně amort.

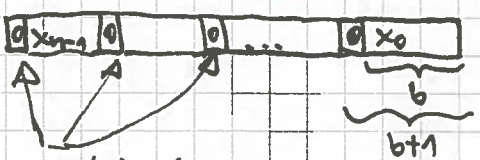
↓ indirekce

y-fast stromy

- čtení $O(\log\log u)$ w.c.
- zápisy $O(\log\log u)$ průměrně amort.
- prostor $O(n)$ w.c. [glob. strom obsahuje $\Theta(n/\log u)$ položek, každá stojí $\Theta(\log u)$ buněk]

RAM jako vektorový počítač:

Vektor $x_0 \dots x_{n-1} \in [2^b]$, kde $n \cdot b = O(w)$, můžeme reprezentovat číslem v $O(1)$ slovech:
(skaláry značíme řeckými písmeny)



nebo zavedeme složité neuniformní RAM: $O(1)$ konstant závislých jenom na w

výplňky (padding)

konstanty, o kterých umíme spočítat v $O(1)$, ale i kdyby ne, máváme dost času na předvýpočet

$\text{Read}(x,i) = \lfloor (x \gg 2^{(b+1)i}) \rfloor \,\&\, 1^b$

(1^b: bin. číslo s b jedničkami)

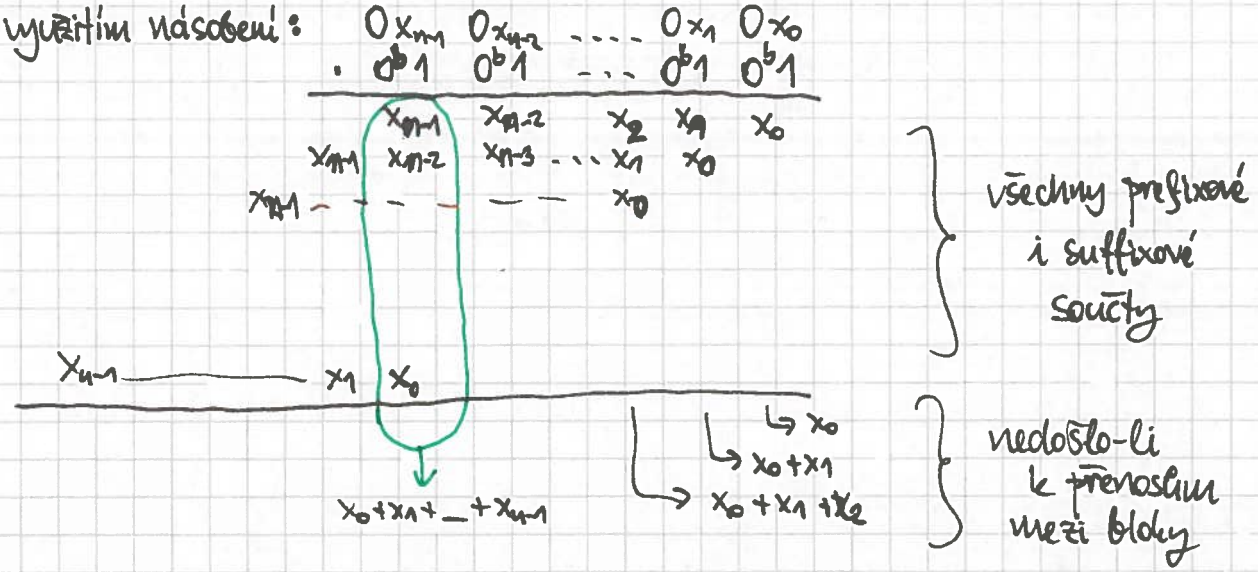
$\text{Write}(x,i,\alpha) = (x \,\&\, 1^{(b+1)(n-i-1)} 0^{b+1} 1^{(b+1)i}) + (\alpha \ll (b+1)i)$

Replicate(α) ... vytvoří vektor $(\alpha,\dots,\alpha)$ – stačí $\alpha\cdot(0^b1)^n$

Sum(x) ... sečte $x_0+\dots+x_{n-1}$, součet se musí vejít do skaláru

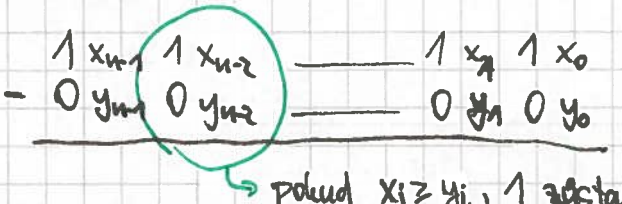
① "magické" řešení: $x \bmod \underbrace{\blacksquare^{b+1}}_{\text{tedy } 2^{bn}-1} \blacksquare \dots \sum_i 2^{(b+1)i}\cdot x_i \overset{\bmod 2^{bn}-1}{\equiv} \sum_i 1^i x_i \equiv \sum_i x_i.$

② využitím násobení:



Cmp(x,y) = z, $z_i = \begin{cases}1 & \text{pokud } x_i < y_i\\ 0 & \text{jinak}\end{cases}$

odečítáme:



tedy: $\left((x \mid \max\,(10^b)^n) - y\right) \gg b \;\&\; (0^b1)^n \oplus (0^b1)^n$

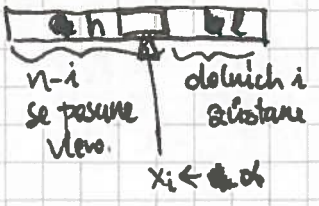
Min(x,y) = z, $z_i = \min(x_i, y_i)$ $\quad m \leftarrow Cmp(x,y)\cdot 1^b$ ... $m_i = \begin{cases}0^b & \text{pokud } x_i \geq y_i\\ 1^b & \text{pokud } x_i < y_i\end{cases}$

$z \leftarrow (x \,\&\, m) \mid (y \,\&\, \sim m)$

Rank(x_1 ■ α) = #i : $x_i < \alpha$ ... Sum(Cmp(x, Replicate(α)))

α: skalár

Insert(x,α) ... vkládání do setříděného vektoru



$i \leftarrow Rank(x,\alpha)$

$h \leftarrow (x \,\&\, 1^{(b+1)(n-i)} \blacksquare 0^{(b+1)i})$

$l \leftarrow (x \,\&\, 1^{(b+1)i})$

Vrátíme $(h \ll (b+1)) \mid (\blacksquare\alpha \ll (b+1)i) \mid l.$

Unpack(α) = x : x_i = i-tý bit čísla α : $\underbrace{\quad}_{\alpha[i]}$

$x \leftarrow \text{Replicate}(\alpha)$
$y \leftarrow (2^{b-1}, \dots, 2^0)$
$t \leftarrow x \& y$ ... $t_i = \begin{cases} 0 \\ y_i \text{ pokud } \alpha[i]=1 \end{cases}$
$z \leftarrow \text{Cmp}(y, t) \oplus (0^b 1)^n$

... $\text{Unpack}_\pi(\alpha)$ ... bity permutuje podle α: pouze se změní maska y: $y_i = 2^{\pi(i)}$

Pack(x) ... inverzní k Unpack

Špinavý trik: "přeformátujeme" vektor:

0	0	0	0	x_3	0	0	0	0	x_2	0	0	0	0	x_1	0	0	0	0	x_0
0	0	0	0	x_3	0	0	0	0	x_2	0	0	0	0	x_1	0	0	0	0	x_0

↓ Sum (pozor, nekorektně kódovaný vektor & trik s mod nefunguje, nadsobení ano.)

$x_3x_2x_1x_0$

Operace s čísly v kvadratické šířce slova ($w = \Omega(b^2)$)

Weight(α) ... Hammingova váha, tedy $\sum_i \alpha[i]$ ... stačí Sum(Unpack(α))

$\text{Permute}_\pi(\alpha) = \text{Pack}(\text{Unpack}_\pi(\alpha))$

LSB(α) ... $\min\{i \mid \alpha[i]=1\}$ ...

α	10000
α-1	01111
α⊕(α-1)	0000 1111

} stačí tedy Weight(α ⊕ (α-1)) − 1

MSB(α) ... $\max\{i \mid \alpha[i]=1\}$ ... třeba b-1-LSB($\text{Permute}_{\text{zrcadlení}}(\alpha)$)

Jiná možnost: $\#(\#i : 2^i \le \alpha)$ ... tedy Sum(Cmp($(2^{b-1}, \dots, 2^0)$, Replicate(α)))
→ tohle je Rank vzhledem k $(2^{b-1}, \dots, 2^0)$

MSB v prostoru O(w), [podobně LSB] [Brodnik 1993] ← ale asi to bylo známé i dříve

1. $b \leftarrow \lceil \sqrt{w} \rceil$, $\ell \leftarrow b$ ... ℓ bloků po b bitech + padding mezi bloky
2. $x \leftarrow (\alpha \,\&\, (01^b)^\ell) \mid ((\alpha \,\&\, (10^b)^\ell) \gg b)$ ... $x_i \neq 0 \Leftrightarrow$ i-tý blok byl neprázdný (proč tak složitě? Musíme dodržet padding)
3. $y \leftarrow \text{Cmp}(0, x)$ ... $y_i = \begin{cases} 0 \\ 1 \text{ pokud } i\text{-tý blok} \neq 0 \end{cases}$
4. $\beta \leftarrow \text{Pack}(y)$, $p \leftarrow \text{MSB}(\beta)$ ... p = číslo nejvyššího bloku s 1
5. $\gamma \leftarrow (\alpha \gg (b+1)p) \,\&\, 1^b$ ... obsah bloku
   $q \leftarrow \text{MSB}(\gamma)$ ... MSB uvnitř bloku
6. Vrátíme $(b+1)p + q$.

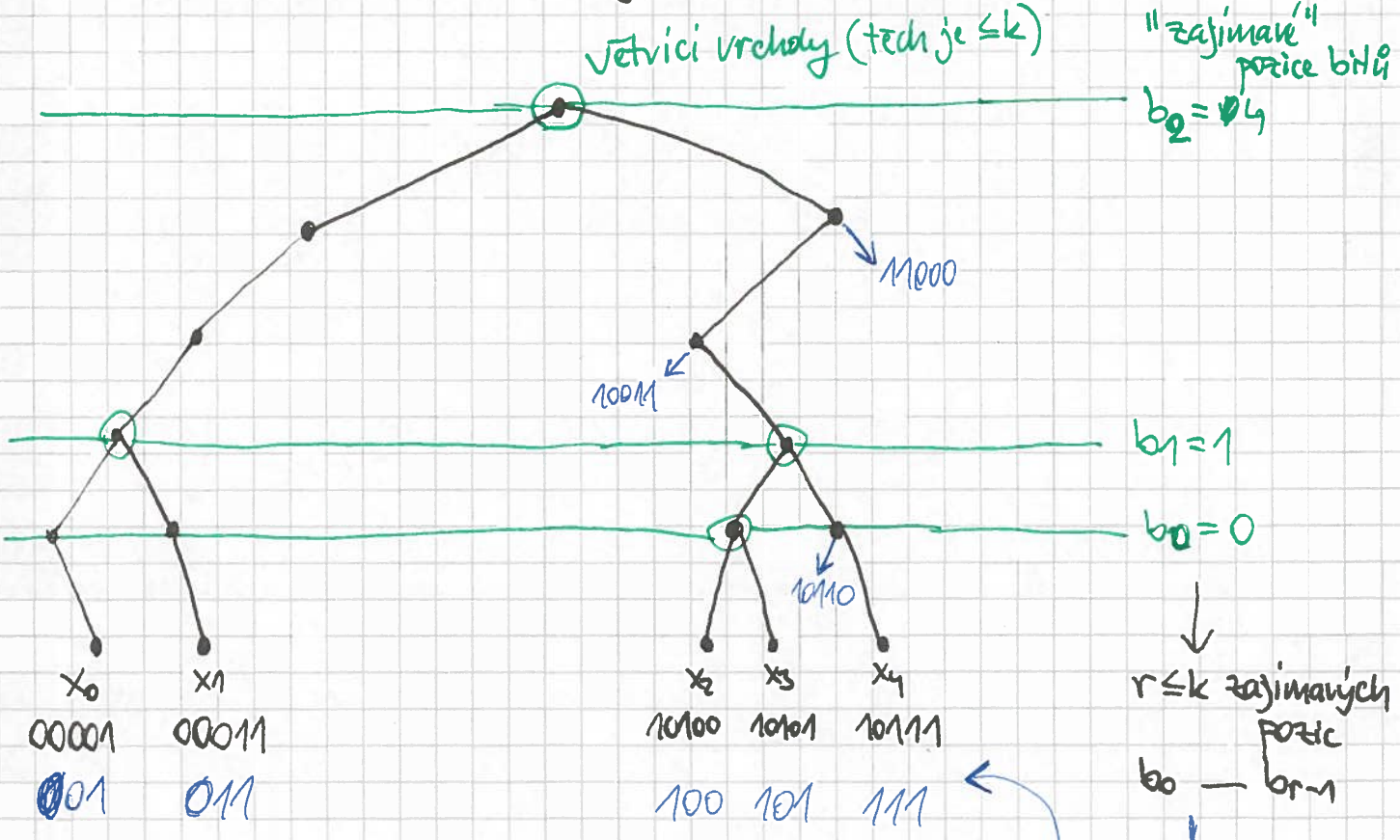
} čas O(1)
šířka slova O(w)

Fusion trees [Fredman & Willard 1990]

- rozhraní jako vEB stromy, fungují lépe pro širší slova (větší universum)
- předvedeme statickou verzi, později použijeme obecnou dynamizační techniku. (exponenciální stromy)
- **Fusion node** - pamatuje si k klíčů $x_0 < x_1 < \dots < x_{k-1}$, $k = O(w^{1/5})$ (šířka slova)
  - umí v konst. čase spočítat $\text{Rank}(q) = \#i : x_i < q$
    → z toho Pred, Succ v $O(1)$
  - konstrukce v čase $\text{Poly}(k)$
- **Fusion tree** - B-strom pro $B = \Theta(w^{1/5})$, ve vrcholech jsou Fusion nodes
  - hloubka $O(\log_w n) = O(\frac{\log n}{\log w})$ ... čím delší slovo, tím lepší
  - Rank počítá v $O(\frac{\log n}{\log w})$

Konstrukce Fusion nodes

- uvažujme trii nad binárními zápisy klíčů (má hloubku w)



- $s(x_0) < s(x_1) < \dots < s(x_{k-1})$
- ... pokud hledáme $q \in X$, stačí hledat $s(q)$ mezi $\{s(x_i)\}$
  - všechna $s(x_i)$ mají $r \cdot k \le k^2 = O(w^{2/5})$ bitů
    → vejdou se do $O(1)$ slov → stačí paralelní Cmp. ("fusion")

Co když hledáme $q \notin X$? $s(q)$ nás na 1. pohled zavede na scestí...

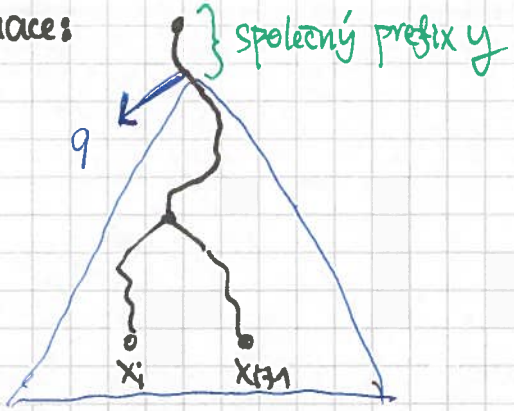
Např. pro $q = 11000$ je $s(q) = 100$, což nás zavede do x_2, ačkoli $Rank(q) = 5$

Přesto z toho něco odvodíme: Nechť $s(x_i) \leq s(q) < s(x_{i+1})$ ...

Spočítáme $MSB(q \oplus x_i)$, $MSB(q \oplus x_{i+1})$ a $MSB(x_i \oplus x_{i+1})$

↑ místo, kde cesta do q odbočí od cesty do x_i

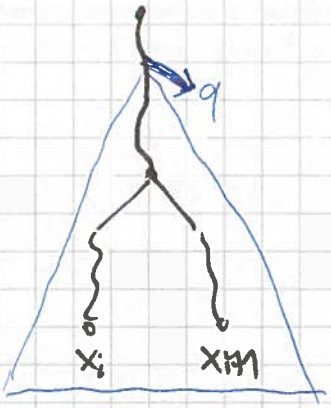
Možné situace:



Succ(q) je Min modrého podstromu

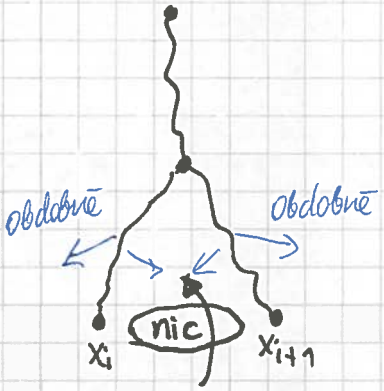
⇓

necháme se vést $s(y10\_\_0)$



K Pred(q) nás vede

$s(y01\_\_1)$



zde je q mezi x_i a x_{i+1}

Takže stačí umět počítat v $O(1)$ $s(x)$ a MSB. Z toho Rank v $O(1)$,

↓ ↓

tohle bohužel neumíme — víme vektorové trilky

Budeme počítat <u>přibližné sketche</u> $a(x)$ ... délky $O(r^4) \subseteq O(u^{4/5})$ } vektory stále mají $O(1)$ slov

... $s(x)$ "prostrkané nulami"

→ stále platí $a(x_i) < a(x_{i+1})$

<u>Princip:</u> $x' = x \,\&\, \sum_i 2^{b_i}$ ... vynulujeme nezajímavé bity

$$x' \cdot \underbrace{M}_{\text{pomocná maska}} = \left(\sum_{i=0}^{r-1} x[b_i] \cdot 2^{b_i}\right) \cdot \left(\sum_{j=0}^{r-1} 2^{m_j}\right) = \sum_{i=0}^{r-1} \sum_{j=0}^{r-1} x[b_i] \cdot 2^{b_i + m_j}$$

$$a(x) := \left((x' \cdot M) \,\&\, \left(\sum_i 2^{b_i + m_i}\right)\right) \gg (b_0 + m_0)$$

<u>Lemma:</u> Pro $b_0 < \_\_ < b_{r-1}$ existují $m_0, \_\_, m_{r-1}$ takové, že:

① všechna $b_i + m_j$ jsou navzájem různá (nevzniknou kolize)

② $b_0 + m_0 < \_\_\_ < b_{r-1} + m_{r-1}$ (zachováme pořadí)

③ $(b_{r-1} + m_{r-1}) - (b_0 + m_0) = O(r^4)$ (malé rozpětí)

Dk: (A) Najdeme $m_0 - m_{r-1} < r^3$ tak, aby $b_i + m'_j$ byly různé modulo r^3.

Indukcí: Máme-li $m_0 - m_{t-1}$ a hledáme m'_t

Chceme se vyhnout $m'_i + b_j \equiv m'_t + b_l$ pro všechna $i<t$, $j,l<r$

$m'_t \equiv m'_i + b_j - b_l$

t možností, r, r } $tr^2 < r^3$ možností

⇓

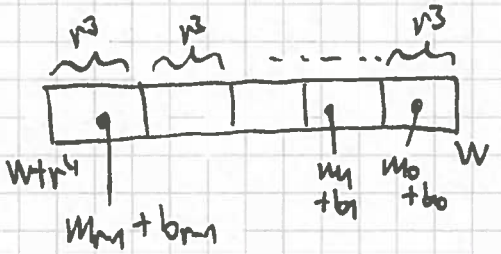
aspoň 1 volná

m'_t lze zvolit

(B) Posuneme m'_i o násobky r^3, aby $\forall i$ $m_i \in [i \cdot r^3, (i+1)r^3)$

m'_i (+b_i)

to by m_i mohla vyčnívat zdopoma, tak ke všem přičteme w



... $m_i + b_j$ jsou stále mod r^3 různá, takže bez mod také.

Shrnutí:

vEB odpovídá v čase $O(\log w)$ ... s w roste

FT v čase $O\left(\frac{\log n}{\log w}\right)$ ... s w klesá

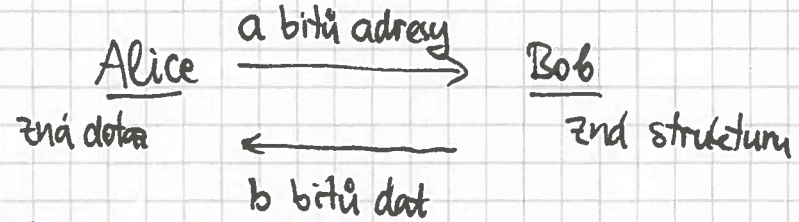
} vyrovná se pro $\log n \approx \log^2 w$

⇓

$O(\sqrt{\log n})$

Dolní odhad [Sen & Venkatesh 2006] pro Cell Probe

Princip: studujeme interaktivní protokol:



Věta: # cell probes = $\Omega\left(\min\left(\log_a w, \log_b n\right)\right)$.

Statická DS pro hledání předchůdce má

[bez důkazu]

Důsledek: pro $a = \Theta(\log n)$ [Poly prostor]

$b = w$

$\frac{\log w}{\log \log n}$ téměř vEB

$\frac{\log n}{\log w}$ Fusion Tree