Dynamizace datových struktur

Velmi obecně: Máme nějakou DS a chceme, aby se chovala "dynamičtěji"
(třeba úplně statické → dynamické s Insertem a Deletem)

Úplný rebuild (přestavba):
- jednou za čas přebudujeme celou strukturu
- tím řešíme omezenou kapacitu (pole, intervalové stromy, vEB, ...) nebo třeba změny parametrů (bloky velikosti ~ log n u indirekce) či "zanášení" struktury smazanými prvky
- strukturu velikosti n přebudujeme za $\Theta(n)$ operací
  → $T(n)/n$ na prvek amort.
  [předpokládáme ~~T(n)~~ $T(O(n)) = O(T(n))$]

Částečný rebuild ... příklad: vyhledávací stromy – dokonale vyvážený strom umíme postavit v $O(n)$ ze setříděné posl., ale nelze ho dynamicky udržovat

Df: BB-α strom ≡ BVS, v každém vrcholu v:

$$|T(\ell(v))| \leq \alpha \cdot |T(v)|$$
$$|T(p(v))| \leq \alpha \cdot |T(v)| \quad \text{pro } \tfrac{1}{2} < \alpha < 1$$

Vede na hloubku $\leq \log_{\frac{1}{1-\alpha}} n$ ... kontrola: udržuji počítadla $|T(v)|$ ve vrcholech

Strategie: Po Ins/Del přepočítám počítadla na cestě do kořene & kontroluji vyvážeňost

Pokud ∃ nevyvážený vrchol, vezmu nejvyšší, vše pod ním rozeberu a postavím znovu dokonale vyvážené.

Intuice: Rebuild stojí $\Theta(k)$ pro podstrom velikosti k, než nastane znovu, musí v podstromu nastat $\Omega(k)$ operací } $O(1)$/op.
... ∀ prvek si předplatí rebuild všech nadřazených stromů → $O(\log n)$/op.

Analýza:

$$\varphi(v) := \begin{cases} \big||T(\ell(v))| - |T(p(v))|\big| & \text{pokud vyjde} \geq 2 \\ 0 & \text{jinak} \end{cases}$$

} v dokonale vyváženém stromu všude 0

$$\Phi := \sum_v \varphi(v)$$

Ins/Del mění $\varphi(v)$ o $O(1)$ v $O(\log n)$ vrcholech ⇒ stojí $O(\log n)$

vyvážení podstromu $T(v)$ o k prvcích ... $\varphi(v) = \Omega(k)$ ⇒ Φ klesne o $\Omega(k)$

→ vyvážení zaplatíme z potenciálu

Věta: V BB-α stromu o n vrcholech trvá Find $O(\log n)$ w.c., Ins/Del $O(\log n)$ amort.

Dynamizace 2D intervalových stromů

- primární i sekundární stromy jsou BB-α
  - rebuild primárního stojí $\Theta(k \log k)$ → musím spořit $\Omega(\log k)$/prvek
  - rebuild sekundárního stojí $\Theta(k)$, → také $\Omega(\log k)$/prvek
    ale ∀ prvek leží v $O(\log n)$ takových

⇒ amort. složitost Ins/Del je $O(\log^2 n)$

pro d-dim. vyjde $O(\log^d n)$ amort. Ins/Del & Find zůstává

Obecná (semi)dynamizace

jen Insert

spíš $\binom{U_X}{<\omega}$ ... jen konečné

Df: Vyhledávací problém je $f: U_Q \times 2^{U_X} \to U_R$

universum dotazů, universum prvků, universum výsledků

Df: V.p. f je rozložitelný $\equiv$ $\exists \sqcup : U_R \times U_R \to U_R$ vyčíslitelná v čase $O(1)$ t.ž.

$$\forall A, B \subseteq U_X,\ A \cap B = \emptyset \quad \forall q \in U_Q \quad f(q, A \cup B) = f(q, A) \sqcup f(q, B)$$

Příklady: "$q \in X$" ... $U_X = \mathbb{Z}$, $U_Q = \mathbb{Z}$, $U_R = \{0,1\}$

nejbližší bod X ke q v $\mathbb{R}^d$ ... $U_X = U_Q = \mathbb{R}^2$, $U_R = \mathbb{R}^2$

} jsou rozložitelné

$q \in conv(X)$ ... $U_X = U_Q = \mathbb{R}^2$, $U_R = \{0,1\}$

} není

Mějme statickou strukturu pro f ... $B_S(n)$ – čas na build

$Q_S(n)$ – čas na dotaz

$S_S(n)$ – prostor

předpokládáme, že $Q_S(n)$, $\frac{B_S(n)}{n}$, $\frac{S_S(n)}{n}$ jsou $\boxed{\text{neklesající}}$

Chceme semidynamickou ... $Q_D(n)$, $S_D(n)$ ... čas na dotaz, prostor

$I_D(n)$, $D_D(n)$ ... amort. čas na Insert, Delete

Konstrukce: Množinu rozložíme na bloky $B_0, B_1, \dots$ takové, že $\forall i\ |B_i| \in \{0, 2^i\}$ } # bloků $\leq \log n$

pro $\forall$ blok mám statickou strukturu

$\Rightarrow$ velikost $n = |X|$ nám jednoznačně určuje, jaké bloky jsou neprázdné (bin. zápis)

Dotaz: Položím dotaz všem blokům, odpovědi zkombinuji

Trvá $\sum_{i \in I} Q_S(2^i) \in O(Q_S(n) \cdot \log n)$

indexy t.ž. $B_i \neq \emptyset$

Prostor: $\sum_{i \in I} S_S(2^i) = \sum_i \frac{S_S(2^i)}{2^i} \cdot 2^i \leq n \cdot \frac{S_S(n)}{n} = S_S(n)$.

$\frac{S_S(2^i)}{2^i} \leq \frac{S_S(n)}{n}$

Insert: Jako přičítání ve dvojkové soustavě ... nechť j je min. takové, že $B_j = \emptyset$

$\underbrace{\text{Rozeberu } B_0 - B_{j-1}, \text{ přidám nový prvek}}_{2^0 + \dots + 2^{j-1} + 1 = 2^j}$, vytvořím novou B_j

B_j přebudováváme jednou za 2^j kroků

$\Rightarrow$ amort. cena vyjde $\leq B_S(2^j) / 2^j \leq \frac{B_S(n)}{n}$

} $\forall$ prvek předplatí $O\left(\frac{B_S(n)}{n} \cdot \log n\right)$

Věta: ... existuje semidynamická struktura s parametry

$Q_D(n) = O(Q_S(n) \cdot \log n)$

$S_D(n) = O(S_S(n))$ ... hodí se udržovat kopie prvků (nemusíme rozebírat)

$I_D(n) = O\left(\frac{B_S(n)}{n} \cdot \log n\right)$

Příklady:

- binární vyhledávání → obecná množina
  - build $\Theta(n \log n)$, dotaz $\Theta(\log n)$, prostor $\Theta(n)$
  - → dotaz $\Theta(\log^2 n)$, prostor $\Theta(n)$, Insert $\Theta(\log^2 n)$ ← lze zrychlit: místo rebuildu Merge v $\Theta(n)$ ⇒ $O(\log n)$ amort.
- též není těsné, pokud $B_S(n) = O(n^\varepsilon)$ … tehdy log zmizí a $\bar{I}_D(n) = O(n^\varepsilon)$
- pro 2D intervalové stromy: build $\Theta(n \log n)$, dotaz $\Theta(\log^2 n)$, prostor $\Theta(n \log n)$ → Insert $\Theta(\log^2 n)$, dotaz $\Theta(\log^3 n)$, prostor $\Theta(\cancel{\log}\, n \log n)$

Worst-case semidynamizace

Myšlenka: rebuild rozložíme mezi více operací, pokaždé provedeme kousek
… ale mezitím musí původní struktury stále existovat a odpovídat na dotazy

Pro každý řád bloku povolíme až 4 bloky: $B_i^1, \dots, B_i^3$ hotové (odpovídají na dotazy)
B_i^* rozdělaná (sjednocení nějakých dvou B_{i-1}, B'_{i-1})

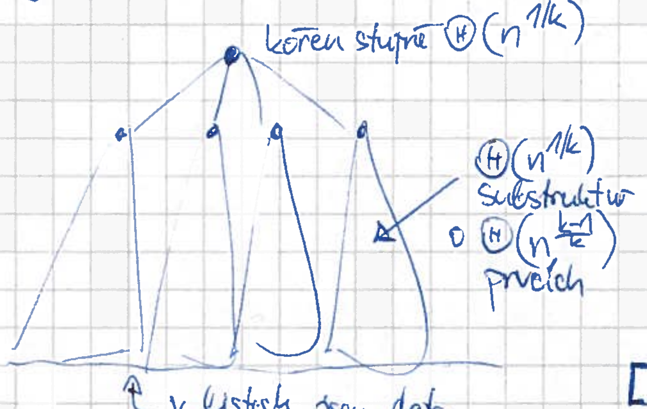
- jakmile se sejdou 2 hotové bloky téhož řádu i, spustíme build B^*_{i+1}, poběží 2^{i+1} kroků. Až doběhne, původní bloky zrušíme.
- nikdy nevzniknou víc než 3 bloky téhož řádu

⇒ Insert v $O(B_S(n)/n \cdot \log n)$ w.c.
dotaz ani paměť se asymptoticky nezhorší

Dále se umí:
- plná dynamizace – statická struktura musí umět "škrtat prvky"
  – jednou za čas úplný rebuild
- worst-case plná dynamizace

Exponenciální stromy

- technika vhodná k dynamizaci Fusion Trees a spol.
- mějme statickou strukturu pro hledání následníka s $B_S(n) \in O(n^{k-1})$ pro nějaké $k \geq 2$ a nějakým $Q_S(n)$
- vyrobíme násl. strom



+ spoj. seznam všech listů

- každý vrchol si pamatuje oddělovač

v_i má oddělovač s_i
v_{i+1} — s_{i+1}

listy v $X_i \subseteq [s_i, s_{i+1})$
a oddělovače

& oddělovač vrcholu (vnitř.) = oddělovač jeho levého syna
[pozor, v listech nemusí být oddělovač = prvek!]

- vrchol si pamatuje oddělovače synů ve statické struktuře

Dotaz: Řídíme se podle oddělovačů, procházíme shora dolů, v listu ±1 stejákem

$$Q_D(n) \le O(Q_S(O(n^{1/k}))) + Q_D\left(O\left(n^{\frac{k-1}{k}}\right)\right) \cdots$$

chceme se zbavit O-ček uvnitř → jednou dosadíme do sebe sama

$$\le \underbrace{O(Q_S(\underbrace{O(n^{1/k})}_{\le n \text{ pro dost velké } n}))}_{O(Q_S(n))} + \underbrace{O\left(Q_S\left(O\left(n^{\frac{k-1}{k}}\right)\right)\right)}_{\text{taktéž } O(Q_S(n))} + \underbrace{Q_D\left(O\left(n^{\left(\frac{k-1}{k}\right)^2}\right)\right)}_{\text{pro dost velké } n \text{ je } O\left(n^{\left(\frac{k-1}{k}\right)^2}\right) \le n^{\frac{k-1}{k}} \Rightarrow \le Q_D\left(n^{\frac{k-1}{k}}\right)} + 1$$

$$\le O(Q_S(n)) + Q_D\left(n^{\frac{k-1}{k}}\right)$$

Paměť: $S_D(n) \le O(\underbrace{(n^{1/k})^{k-1}}_{n^{\frac{k-1}{k}}}) + \sum_i S_D(n_i)$

$\sum_i n_i = n$, při $n_i \in \Theta\left(n^{\frac{k-1}{k}}\right)$

$\Rightarrow S_D(n) \in O(n)$ … na každé úrovni $\Theta\left(n^{\frac{k-1}{k}}\right)$, úrovní je jistě $O\left(n^{\frac{1}{k}}\right)$

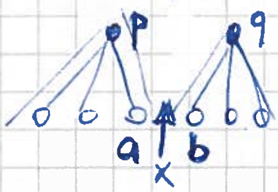
… čas na kompletní konstrukci vyjde stejně

Delete: prvek smažeme … problém:

mažeme 1. syna → museli bychom změnit oddělovač v otci … tak ho předáme 2. synovi

+ # synů může vylézt z povoleného rozsahu → později

Insert: najdeme předchůdce & následníka …



p, q mohou být stejné

- buď $x <$ oddělovač(b) ⇒ přidám pod p za a, $\text{odd}(x) \leftarrow x$
- nebo $x \ge \text{odd}(b)$ ⇒ přidám pod q před b, $\text{odd}(x) \leftarrow \text{odd}(b)$, $\text{odd}(b) \leftarrow b$

} nikdy neměníme oddělovač v otci

Rebalance: Velikost podstromu může opustit povolený rozsah Θ (coi)

→ řeším rozpůlením / přerozdělením / slitím (à la indirekce)

- v daném místě (struktura velikosti t, substruktury velikosti $\Theta\left(t^{\frac{k-1}{k}}\right)$)

  to nastane za $\Omega\left(t^{\frac{k-1}{k}}\right)$ operací Ins/Del

  a rebuild stojí $O\left(t^{\frac{k-1}{k}}\right)$ za podstrukturu + $\underbrace{O\left(\left(t^{\frac{1}{k}}\right)^{k-1}\right)}_{O\left(t^{\frac{k-1}{k}}\right)}$ za stat. strukturu v kořeni

  $= O\left(t^{\frac{k-1}{k}}\right)$

- ∀ Ins/Del si předplatí toto na všech úrovních

  ⇒ omezeno stejnou rekurencí jako $Q_D(n)$

Důsledek: Exponenciální strom zabere prostor $S_D(n) = \Theta(n)$, hledá v čase dle rekurence $Q_D(n) = \mathcal{O}(Q_S(n)) + Q_D(n^{\frac{k-1}{k}})$ w. c.
Vkládá/maže v $Q_D(n)$ amort.,
sestrojit jde v $\Theta(n)$.

umí se i worst-case verze, je (jako obvykle) výrazně složitější

Použití: • Pro Fusion Tree $B_S(n) = \mathcal{O}(n^4) \Rightarrow k = 5$
$Q_S(n) = \mathcal{O}\left(\frac{\log n}{\log W}\right)$

$$\Rightarrow Q_D(n) = \mathcal{O}\left(\frac{\log n}{\log W}\right) + Q_D(n^{\frac{4}{5}})$$

$$\Rightarrow Q_D(n) = \sum_{i \geq 0} \frac{\log(n^{(\frac{4}{5})^i})}{\log W} = \sum_{i \geq 0} \frac{(\frac{4}{5})^i \log n}{\log W} = \mathcal{O}\left(\frac{\log n}{\log W}\right)$$

• Pro vEB-like strukturu $k = 2$, $Q_S = \mathcal{O}(\log \log n) \Rightarrow Q_D(n) = \mathcal{O}(\log^2 \log n)$

Persistence

Možnosti:
- efemérní (pomíjivá) struktura – pokaždé nevratně mění stav

  $\wedge$
- semipersistentní – pamatuje si historii
  – update vytvoří novou verzi
  – dotaz pokládáme libovolné verzi

  $\wedge$
- plně persistentní – máme dovoleno upravovat historické verze
  → historie tvoří strom

  $\wedge$
- funkcionální (bez side-efektů) – co jsme zapsali, nikdy neupravujeme
  – to je šikovné i při paralelním programování

Příklady:
- srůstající seznamy s přidáváním na začátek → funkcionální; $\mathcal{O}(1)$ paměti na verzi
- vyhledávací stromy s kopírováním cesty → funkcionální
  - $\mathcal{O}(\log n)$ ~~na~~ času na dotaz i úpravu
  - $\mathcal{O}(\log n)$ prostoru na verzi (rozmyslet vyvažování)

    ↳ kopírujeme cestu do kořene

    ↳ rotace "blízko" cesty → navíc zkopírujeme $\mathcal{O}(\log n)$ vrcholů

Aplikace: Lokalizace bodu v rovině via zametání

Model: Pointerové datové struktury
- tvořené krabičkami, v každé $\mathcal{O}(1)$ dat + $\mathcal{O}(1)$ ukazatelů → orient. graf
- "držátka" – $\mathcal{O}(1)$ ukazatelů zvenku
- dotaz – začneme držátky, chodíme po pointerech do dalších krabiček
- update – jako dotaz + modifikace některých navštívených krabiček
  + zakládání nových krabiček

Pro semipersistenci: stačí verzovat jen to, co je třeba k hledání
… např. u AVL-stromů ukazatele a klíče, ale už ne znaménka } → strukturální změny

- ukážeme, jak 1 strukt. změnu uložit v čase a prostoru $O(1)$ amort., přičemž dotazy zpomalíme jen $O(1)$-krát
- Příklad: (2,4)-strom provede amort. $O(1)$ strukt. změn na update
  → persist. strom s časem $O(\log n)$ w.c. na dotaz, $O(\log n)$ amort. na update a prostorem $O(1)$ amort. na verzi

potenciál vrcholu =

klíčů	potenciál
0	2
1	1
2	0
3	2
4	4

} Ins/Del stojí přímo $O(1)$
štěpení: 4 klíče → 2 + 1 + 1 nahoru (Φ: $4 \to 0+1+\max. 2$)
sloučení: 0 klíčů + 1 klíč + 1 shora → 2 (Φ: $2+1 \to 0 + \max. 2$)
půjčení: tím skončíme, takže můžeme platit $O(1)$

„Tlusté" vrcholy

- každý vrchol si pamatuje všechny změny ukazatelů i dat
  → vyhledávací strom indexovaný verzí (to bude číslo)
- verze stojí $O(1)$ prostoru, ale operace jsme zpomalili $O(\log h)$-krát pro historii délky h
  ↳ můžeme zlepšit na $\log\log h$ pomocí VEB (alespoň randomizovaně)

Zkřížení tlustých vrcholů s kopírováním [Driscoll, Sarnak, Sleator, Tarjan 1986]

- funguje, pokud $\forall v \; \deg^{in}(v) \le p \in O(1)$
- vrchol (krabička) si pamatuje základní verzi (celou) + $e \ge p$ extra pointerů (slotů) (trojice (index, verze, kam))
- kdykoli měníme pointer a je volný extra slot, využijeme ho [pointer téže verze můžeme přepsat přímo]
- jinak (došlo místo nebo měníme data) zkopírujeme vrchol → update až p pointerů v předchůdcích (pozor, musíme je zjistit → zpětné pointery)

Amortizace: $\varphi(v) :=$ # využitých slotů [může být $e+1$, pokud vrchol právě přetekl]
$\Phi := \sum_v \varphi(v)$ přes v dosažitelné v aktuální verzi

- změna hodnoty dat či pointeru stojí $O(1)$
- kopírování při přetečení: $\varphi(v)$ byl $e+1 \ge p+1 \Rightarrow$ 1 spotřebuji, dalších $\le p$ dám předchůdcům + starý vrchol přestane být dosažitelný

Detaily:

- verze číslujeme přir. čísly
- krabička:
  - počáteční verze
  - počáteční stav dat i pointerů
  - pointer na nejbližší novější instanci
  - pointery na předchůdce (zpětné) v nejnovější verzi struktury – žádné spec. pořadí, p ks.
  - e slotů na novější verze pointerů
- pro $\forall$ verzi hodnoty „drátek" … $O(1)$ prostoru na verzi

všechny instance tvoří spoják setříděný dle verzí
- při update se nám může stát, že upravujeme pointer ve vrcholu, který už byl zkopírován [nikdy nekopírujeme tentýž vrchol 2× za update]

Invariant: Pointery verze v vedou na instance vrcholů, které obsahují verzi v. } dočasně narušeno

Důsledek: $O(1)$ amort. času i prostoru na verzi, $O(1)$ w.c. na 1 krok hledání

Stručně o plné persistenci

Problémy: ① musíme vyjít z worst-case struktury (jinak rozbijeme amortizaci)

② musíme verzovat vše … u Č-Č stromů barvy → zavedeme rozdílové kódování → $O(1)$ w.c. změn/op.

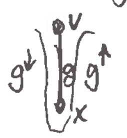
③ přestane existovat lin. uspoř. na verzích … místo toho strom verzí (částečné uspoř.)

→ "obejdeme strom po obvodu" – záznam průchodu DFS po hranách

- směrem dolů provedeme změnu
- směrem nahoru provedeme opačnou

nová verze sem

→ mezi $e^\uparrow$ a $g^\downarrow$ přidáme



potřebujeme reprezentovat seznam, umět ukládat a porovnávat polohy – List Order Problem

- obojí se umí v $O(1)$, dokonce w.c. (viz později)

Tlusté vrcholy

→ opět strom verzí, ale klíče jsou implicitní (via List Order)
stále $O(1)$ prostoru na verzi a zpomalení $O(\log h)$.

Struktura se sloty: [též Driscoll et al. 1986]

- předchůdce musíme verzovat
- pokud vrcholy mají $\le p$ předchůdců a $\le d$ následníků, volíme #slotů $e \ge 2d + 2p$
- $\varphi(v) = \min(0, \#\text{ využitých slotů} - e/2)$, $\Phi := \sum_v \varphi(v)$ tentokrát přes všechny vrcholy
- po přetečení vrchol dělíme na 2 části (volbou vhodné hraniční verze)
  - oba nové vrcholy mají $\varphi = 0$
  - předtím bylo $\varphi \ge 2d+2p+1 - e/2 \ge d+p+1$
    ↳ 1 spotřebujeme, $d+p$ předáme okolním vrcholům
- detaily značně trikové

⇒ opět $O(1)$ času a prostoru amort. na verzi, zpomalení $O(1)$ w.c.

→ umí se i w.c. verze se vším $O(1)$

Další persist. struktury:

to je optimální!

- pole s $O(\log\log h)$ času na operaci, $O(1)$ prostoru na verzi (vše amort., random.)
  - y-fast strom verzí v $\forall$ položce pole
  - semipersistentně snadné,
    s plnou persistencí se musí vyřešit přečíslování v List Orderu

[Viz disertace Milana Straky]