# Dynamizace datových struktur

Velmi obecně: Máme nějakou DS a chceme, aby se chovala "dynamicky" (třeba úplně statická → dynamická s Insertem a Deletem)

**Úplný rebuild** (přestavba):
- jednou za čas přebudujeme celou strukturu
- tím řešíme omezenou kapacitu (pole, intervalové stromy, $v \in \mathbb{B}$, ...) nebo třeba změny parametrů (bloky velikosti $\sim \log n$ u indirekce) či "zarovnání" struktury snažící prvky
- strukturu velikosti $n$ přebudujeme za $\mathcal{H}(n)$ operací
  → $T(n)/n$ na prvek amort., kde $T(n)$ je složitost přebudování.
  [předpokládáme $T(O(n)) = O(T(n))$]

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

Def: BB-$\alpha$ strom ≡ BVS, v každém vrcholu $v$:

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

Vede na hloubku $\le \log_{\frac{1}{1-\alpha}} n$ ... kontrola: udržujeme počtadla $|T(v)|$ ve vrcholech

Strategie: Po Ins/Del přepočítám počtadla na cestě do kořene & kontroluji vyváženost

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

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

Analýza: $\varphi(v) := \begin{cases} \big||T(\ell(v))| - |T(p(v))|\big| & \text{pokud vyjde} \ge 2 \\ 0 & \text{jinak} \end{cases}$ } 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 ... $\varphi(v) \ge \Omega(k)$ ⇒ $\Phi$ klesne o $\Omega(k)$
→ vyvážení zaplatíme z potenciálu

**Věta:** V BB-$\alpha$ 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-$\alpha$
  - rebuild primárního stojí $\mathcal{H}(k \log k)$ → musím spořit $\Omega(\log k)$/prvek
  - rebuild sekundárního stojí $\mathcal{H}(k)$, ale ∀ prvek leží v $O(\log n)$ takových → také $\Omega(\log k)$/prvek

⇒ 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

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

- $U_Q$ ... universum dotazů
- $2^{U_X}$ ... universum prvků
- $U_R$ ... universum výsledků

**Def:** V.p. $f$ je rozložitelný $\equiv \exists \sqcup: U_R \times U_R \to U_R$ vyčíslitelná v čase $O(1)$ tož.

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

**Příklady:** "$q \in X$" } jsou rozložitelné
nejbližší bod $X$ ke $q$ v $\mathbb{R}^d$ } jsou rozložitelné
$q \in \text{conv}(X)$ } není

... $U_X = \mathbb{Z}$, $U_Q = \mathbb{Z}$, $U_R = \{0,1\}$
... $U_X = U_Q = \mathbb{R}^2$, $U_R = \mathbb{R}^2$
... $U_X = U_Q = \mathbb{R}^2$, $U_R = \{0,1\}$

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 rozložitelné

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 $\le \log n$
← 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:** $\sum_{i \in I} S_S(2^i) = \sum_i \frac{S_S(2^i)}{2^i} \cdot 2^i \le n \cdot \frac{S_S(n)}{n} = S_S(n)$.
$\le \frac{S_S(n)}{n}$

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

Rozeberu $B_0 \dots B_{j-1}$, přidám nový prvek, vytvořím novou $B_j$

$2^0 + \dots + 2^{j-1} + 1 = 2^j$

$B_j$ přebudováváme nejvýše jednou za $2^j$ kroků

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

} $\forall$ 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 $\mathcal{O}(n \log n)$
  - dotaz $\mathcal{O}(\log n)$
  - prostor $\mathcal{O}(n)$
  - dotaz $\mathcal{H}(\log^2 n)$
  - prostor $\mathcal{H}(n)$
  - Insert $\mathcal{H}(\log^2 n)$ ← lze zrychlit: místo rebuildu Merge v $\mathcal{W}(n)$ ⇒ $O(\log n)$ amort.
- též není těsné, pokud $B_S(n) = O(n^{\varepsilon})$ ... tehdy log zmizí a $T_D(n) = O(n^{\varepsilon})$
- pro 2D intervalové stromy:
  - build $\mathcal{H}(n \log n)$
  - dotaz $\mathcal{W}(\log^2 n)$
  - prostor $\mathcal{H}(n \log n)$
  - → Insert $\mathcal{H}(\log^2 n)$
  - dotaz $\mathcal{W}(\log^3 n)$
  - prostor $\mathcal{H}(\log n \log n)$

## 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 použijeme až 4 bloky: $B_i^1, \dots, B_i^3$ hotové (odpovídají na dotazy)
$B_i^*$ rozdělaná (sjednocení nějakých dvou $B_{i-1}^1, B_{i-1}^2$)

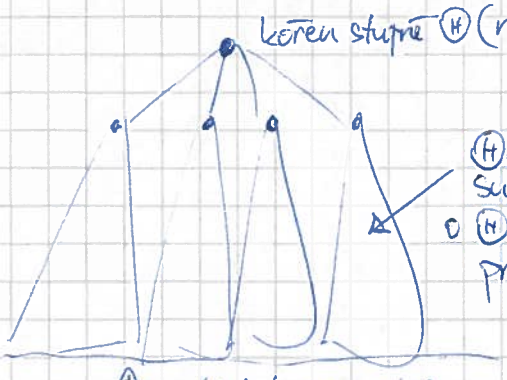
- jakmile se sejdou 2 hotové bloky téhož řádu $i$, spustíme build $B_{i+1}^*$, poběží $2^{i+1}$ kroků. Až dobehne, původní bloky zrušíme.
- nikdy nevzniknou víc než 3 bloky téhož řádu

⇒ 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 "zkrátit prvky"
  - jednou za čas úplný rebuild
  - pomocná struktura, která si pamatuje prvek → blok (třeba BBS)
- worst-case plná dynamizace

## Exponenciální stromy

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




↑ v listech jsou data
+ 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 = oddělovač jeho levého syna (vnitř.)
[pozor, v listech nemusí být oddělovač = prvek!]

- vrchol si pamatuje oddělovače synů ke statické struktuře

**Dotaz:** Řídíme se podle oddělovačů, procházíme shora dolů, v listu $\neq 1$ spojkem

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

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

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

$$\underbrace{O(Q_S(n))}_{O(Q_S(n))} \qquad O(Q_S(n))$$

pro dost velké $n$ je $O(n^{\left(\frac{k-1}{k}\right)^2}) \leq n^{\frac{k-1}{k}}$

$$\Rightarrow \leq Q_D(n^{\frac{k-1}{k}})$$

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

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

$\sum_i n_i = n$, $\forall i\ n_i \in \Theta(n^{\frac{k-1}{k}})$

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

... čas na kompletní konstrukci vyjde stejně

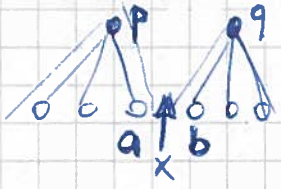
↳ tohle není tak triviální, ale čas na hladinu směrem dolů exponenciálně roste ⇒ dominují listy a těch je $O(n)$.

**Delete:** prvek smažeme ... problém:

↳ mažeme 1. syna → museli bychom změnit oddělovač v otci ... tak ho předáme ze synovi

\+ # synů může vyletět z povoleného rozsahu → později

**Insert:** najdeme předchůdce & následníka ...




p, q mohou být stejné

- buď $x <$ oddělovač(b) ⇒ přidám pod p za a, odd(x) ← x
- nebo $x \geq$ odd(b) ⇒ přidám pod q před b (ale $x < b$), odd(x) ← odd(b), odd(b) ← b

} nikdy nemění oddělovač v otci

přebudují stat. strukturu v otci, ale ta je tedy nad listy konstantně velká

**Rebalance:** Velikost podstromu může opustit povolený rozsah $\Theta(\text{cor}_i)$

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

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

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

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

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

- ∀ 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 $Q_D(n) = O(Q_S(n)) + Q_D(n^{\frac{k-1}{k}})$ u.c.o
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 $B_S(n) = O(n^{\frac{1}{k}}) \Rightarrow k = 5$
$Q_S(n) = O(\frac{\log n}{\log w})$

$\Rightarrow Q_D(n) = O(\frac{\log n}{\log w}) + Q_D(n^{\frac{4}{5}})$

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

pro setříděné pole s bin. searchem vyjde $O(\log n)$ ☺

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

---

# Persistence

**Možnosti:** • efemérní (používá) struktura – pokaždé nevratně mění stav
∧
• semipersistentní – pamatuje si historii
– update vytvoří novou verzi
∧ – dotaz pokládáme libovolné verzi
• plně persistentní – máme dovoleno upravovat historické verze
→ historie tvoří strom
∧
• funkcionální (bez side-efektů) – co jsme zapsali, nikdy nepřepíšeme
– to jesikové i při paralelním programování

**Příklady:**

• srůstající seznamy s přidáváním na začátek → funkcionální, $O(1)$ paměti na verzi
• vyhledávací stromy s kopírováním cesty → funkcionální
– $O(\log n)$ času na dotaz i úpravu
– $O(\log n)$ prostoru na verzi (rozmyslet vyvažování)
↳ kopírujeme cestu do kořene ↳ rotace "blízko" cesty → navíc zkopírujeme $O(\log n)$ vrcholů

**Aplikace:** Lokalizace bodu v rovině via zametání

**Model:** Pointerové datové struktury

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

**Pro semiperzistenci:** • stačí uchovávat 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 =
- 0 klíčů → 2
- 1 → 1
- 2 → 0
- 3 → 2
- 4 → 4

Ins/Del stojí přímo $O(1)$
štěpení: 4 klíče → 2 + 1 + 1 nahoru ($\Phi$: 4 → 0+1+max.2)
sloučení: 0 klíčů + 1 klíč + 1 shora → 2 ($\Phi$: 2+1 → 0 + max.2)
půjčení: tím skončíme, takže určitě 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 $h$
  ↳ určitě můžeme zlepšit na $\log\log h$ pomocí vět (alespoň randomizované)

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

- funguje, pokud $\forall v: \deg^m(v) \le p \in O(1)$
- vrchol si pamatuje základní verzi (celou) + $e \ge p$ extra pointerů (slotů) (krabička)
  (trojice (index, verze, kam))
- kdykoli měníme pointer a je volný extra slot, využijeme ho [pointer téže verze určitě 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 $e+1 \ge p+1$ ⇒ 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 (v nejnovější verzi struktury)
  - pointery na předchůdce (zpětné) – žádné spec. pořadí, $p$ ks.
  - $e$ slotů na novější verze pointerů
- pro $\forall$ verzi hodnoty "držátek" ... $O(1)$ prostoru na verzi

všechny instance tvoří spoják setříděný dle verzí
- při update se nám určitě stačí zaopatřit 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

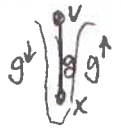
**Důsledek:** $O(1)$ amort. času i prostoru na verzi, $O(1)$ w.c. na 1 krok hledání

# Stručně o plné persistenci

Problémy: ① musíme vyjít z worst-case struktury (jinak rozbijeme amortizaci)
② musíme verzovat vše ... u C-C stromů bary → zavedeme rozdílové ladování → $O(1)$ u.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


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




potřebujeme reprezentovat seznam, umět vkládat a porovnávat polohy – List Order Problem
- obojí se umí v $O(1)$, dokonce u.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 $\leq p$ předchůdců a $\leq d$ následníků, volíme # slotů $e \geq 2d + 2p$
- $\varphi(v) = \max(0, \#\text{využitých slotů} - e/2)$, $\Phi(\mathcal{B}) = \sum_v \varphi(v)$ (tentokrát přes všechny vrcholy)
- po přetečení vrchol dělíme na 2 části (volbou vhodné hraniční verze)
  - oba nové vrcholy mají $\varphi = 0$
  - předtím bylo $\varphi \geq e, +1 - e/2 \geq d+p+1$
    ↳ 1 spotřebujeme, $d+p$ předáme okolním vrcholům
- detaily značně trikové

→ opět $O(1)$ času a prostoru amort. na verzi, zpomalení $O(1)$ u.c.
→ umí se i u.c. verze se vším $O(1)$

**Další persist. struktury:**

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

[Viz disertace Milana Straky]