Lenivý (scapegoat) strom

Existuje mnoho druhov „snaživých“ vyvažovaných stromov, ktoré si pre každý vrchol pamätajú nejakú dodatočnú informáciu – napríklad výšku, čiernu/červenú farbu alebo počet uzlov v podstrome – a pomocou rotácií sa neustále snažia udržať strom vyvážený. Naopak, v tejto kapitole si však ukážeme úplne iný prístup: Scapegoat stromy sú úplne lenivé a nerobia vôbec nič, kým nemusia.

Je to podobné, ako keď máme doma neporiadok: Pokiaľ je ešte relatívne malý, vieme potrebné veci nájsť pomerne rýchlo. Ale keď nám už prerastie cez hlavu, musíme si upratať. Zároveň nás môže hriať dobrý pocit, ako sme odďaľovaním upratovania ušetrili kopu času.[1]

Možno si v tomto momente kladiete (uprávnenú) otázku:

> Prečo sa vôbec zaoberať ďalším typom vyhľadávacieho stromu? To nám nestačia existujúce riešenia ako AVL alebo červeno-čierne stromy?

Uvediem tri dôvody, prečo sú scapegoat stromy zaujímavé:

#1. Lenivá metóda. Pre mnohé druhy stromovitých štruktúr sú základnou metódou vyvažovania rotácie. V neskorších kapitolách si však ukážeme dátové štruktúry, kde nie je možné efektívne rotovať vrcholy. Otázka potom znie: Má niekto plán B? My ho mať budeme! Elegantné riešenie: nič nerobiť, až kým strom nie je príliš nevyvážený a vtedy od základov prebudovať časť štruktúry.

#2. Výška stromu. Perfektne vyvážený binárny strom má výšku $\lceil \lg n \rceil$. Ako sú na tom iné vyvažované stromy?

- Červeno-čierne stromy garantujú maximálnu výšku $2 \lg n$.
- AVL stromy dosahujú maximálnu výšku okolo $1.44 \lg n$.
- Priemerná výška náhodného stromu (treapu) je okolo $2.988 \lg n$

Otázka znie: Ako veľmi sa vieme priblížiť k ideálnej výške $\lg n$?

Scapegoat stromy umožňujú dosiahnuť výšku $(1+\varepsilon) \lg n$ pre ľubovoľne malé $\varepsilon > 0$, pričom insert a delete bude stále trvať $O(\log n)$. Samozrejme, je to za určitú cenu: čím viac sa snažíme priblížiť k optimálnej výške, tým častejšie budeme musieť prebudovávať podstromy a konštanta skrytá v O bude väčšia. Scapegoat stromy nám však poskytujú možnosť zobchodovať efektívnosť aktualizácií za rýchlosť vyhľadávania.

#3. Dodatočná pamäť. Väčšina vyvažovaných stromov si musí pre každý uzol pamätať nejakú dodatočnú informáciu:

- AVL stromy si udržiavajú rozdiely výšok ľavého a pravého podstromu,
- červeno-čierne stromy si pamätajú farbu každého vrcholu,

[1] Disclaimer: Toto nie je odporúčanie do reálneho života. Chýbajú implementačné detaily.

- treapy si ukladajú náhodné priority vrcholov.

Otázka znie: Dá sa vyvažovať aj bez toho?

Prekvapujúca odpoveď znie ÁNO, hoci my si kvôli jednoduchosti ukážeme verziu, ktorá využíva dodatočnú pamäť.

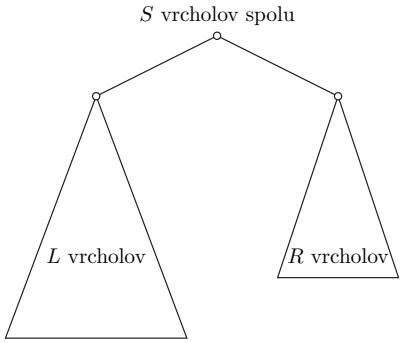
Ako fungujú scapegoat stromy

Základná myšlienka je veľmi jednoduchá: necháme strom rásť, nerobíme žiadne rotácie ani vyvažovanie, pokiaľ to nie je nevyhnutné. Keď sa strom stane príliš nevyváženým, nájdeme „vinníka“ – vrchol, ktorý je zodpovedný za nevyváženosť – a celý jeho podstrom prebudujeme na perfektne vyvážený binárny vyhľadávací strom.[2]

Prebudovanie podstromu s k vrcholmi síce trvá lineárny čas $O(k)$, ale ukážeme si, že takéto prebudovania nastávajú len zriedka a že celkový čas potrebný na všetky prebudovania je v amortizovanom zmysle stále $O(\log n)$ na operáciu. Dôležité je, že ak je nevyvážená iba malá časť stromu, prebudujeme len túto lokálnu časť a zvyšok stromu zostáva nedotknutý.

Kritérium vyváženosti

Vezmime si ľubovoľný vrchol v v strome a jeho podstrom, ktorý obsahuje celkovo S vrcholov (vrátane samotného v). Nech L je počet vrcholov v ľavom podstrome a R počet vrcholov v pravom podstrome v ako na obrázku:



Ak by bol vrchol v perfektne vyvážený, ľavý aj pravý podstrom by mali mať zhruba rovnakú veľkosť (polovicu celého podstromu)

$$L \approx R \approx \frac{1}{2}S.$$

[2] Odtiaľ pochádza aj anglický názov *scapegoat tree* – *scapegoat* znamená „obetný baránok“, teda niekto, na koho zvalíme (možno i neprávom) všetku vinu. Scapegoat strom môže byť mierne nevyvážený na viacerých miestach, no my si vždy vyberieme práve jeden vrchol, na ktorý to zvalíme.

To je pomerne ťažko dosiahnuteľné, takže sa v scapegoat stromoch uspokojíme so slabším kritériom: Budeme hovoriť, že *vrchol v je v rovnováhe,* ak platí

$$\begin{aligned} & L \le \frac{2}{3}S \quad \text{a zároveň} \quad R \le \frac{2}{3}S \\ \text{ekvivalentne:} \quad & R \ge \frac{1}{3}S - 1 \quad \text{a zároveň} \quad L \ge \frac{1}{3}S - 1 \\ \text{ekvivalentne:} \quad & L \le 2 \times R + 2 \quad \text{a zároveň} \quad R \le 2 \times L + 2 \end{aligned}$$

Budeme hovoriť, že *strom je vyvážený, ak je každý jeho vrchol v rovnováhe.* Formálne: pre každý vrchol v a každého jeho syna s platí:

$$\text{size}(s) \le 2/3 \times \text{size}(v),$$

teda žiadny syn nie je väčší ako dve tretiny celého podstromu svojho rodiča.

Z kritéria rovnováhy by malo byť zjavné, že ak je strom vyvážený, jeho výška bude logaritmická vzhľadom na počet vrcholov N. Prečo?

- Koreň má pod sebou všetkých N vrcholov.
- Každý jeho syn má pod sebou najviac $\frac{2}{3}N$ vrcholov.
- Každá ďalšia úroveň znižuje počet vrcholov v podstrome o faktor aspoň 2/3, teda
- každý vnuk má najviac $\frac{2}{3} \times \frac{2}{3} \times N = (\frac{2}{3})^2 \times N$ vrcholov.
- každý pravnuk má najviac $\frac{2}{3} \times \frac{2}{3} \times \frac{2}{3} \times N = (\frac{2}{3})^3 \times N$ vrcholov.
- Všeobecne: podstrom v hĺbke h obsahuje najviac $(\frac{2}{3})^h \times N$ vrcholov.

Otázka znie: aká je maximálna hĺbka stromu? Hľadáme najväčšie h, pre ktoré ešte existuje aspoň jeden vrchol v podstrome, teda

$$\begin{aligned} \left(\frac{2}{3}\right)^h \times N &\ge 1 \\ N &\ge \left(\frac{3}{2}\right)^h \\ \log_{3/2} N &\ge h \end{aligned}$$

Maximálna výška vyváženého stromu je teda: $h \le \log_{3/2} N$, čo je približne $1.7095 \lg N$. Rozmyslite si, ako by sa maximálna výška zmenila, ak by sme namiesto dvoch tretín zvolili inú konštantu $\alpha \in (\frac{1}{2}, 1)$.

Vkladanie

Algoritmus vkladania do Scapegoat stromu začína rovnako ako v obyčajnom binárnom vyhľadávacom strome: prechádzame od koreňa smerom nadol, nájdeme správne miesto podľa usporiadania kľúčov a vložíme nový vrchol ako list.

Rozmyslite si, že jediné veľkosti podstromov, ktoré sa pri vkladaní nového vrcholu zmenia, sú tie na ceste od koreňa k novo vloženému vrcholu. Preto po vložení prejdeme cestu späť do koreňa, prepočítame veľkosti podstromov a skontrolujeme, či všetky vrcholy na tejto ceste stále spĺňajú podmienku rovnováhy. (Vo väčšine prípadov budú a môžeme tu skončiť.)

Ak však nájdeme vrchol, ktorý je nevyvážený, prípadne viac takých vrcholov, vyberieme ten najvyšší a celý jeho podstrom prebudujeme na perfektne vyvážený binárny vyhľadávací strom.

Rozmyslite si, ako presne takúto rekonštrukciu efektívne vykonať. Cieľom je, aby prebudovanie podstromu s k vrcholmi prebehlo v čase $O(k)$.

Celková zložitosť jednej operácie bude $O(\log n)$ a „raz za čas“ ešte plus $O(k)$ navyše, ak prebudujeme podstrom veľkosti k. Formálnu analýzu vykonáme čoskoro, ale už teraz si môžeme všimnúť intuitívny dôvod, prečo bude rekonštrukcia v amortizovanom zmysle takmer „zadarmo“: Po prebudovaní je konkrétny vrchol perfektne vyvážený, jeho podstromy majú zhruba $\frac{1}{2}k$ vrcholov. Aby sa takýto vrchol opäť stal nevyváženým, musíme pridať *lineárne veľa* vrcholov (napríklad $\approx \frac{1}{2}k$ vrcholov len na jednu stranu). Tým pádom však môžeme nákladné prebudovanie rozpočítať na lineárne veľa predchádzajúcich operácií. Ukážeme, že ak každá operácia vkladania uloží dopredu nejaké „peniaze do fondu opráv“, tak kým sa nejaký podstrom stane opäť nevyvážený, stihne si nasporiť dostatok zdrojov na zaplatenie celej rekonštrukcie. Tým bude zaručené, že amortizovaný čas každej operácie zostane $O(\log n)$.

Vymazávanie

Vymazávanie v scapegoat stromoch je ešte lenivejšie ako vkladanie. Pri vymazávaní vrcholu x ho fyzicky neodstránime zo stromu, len ho *označíme* ako vymazaný. Pozor: Operáciu find musíme potom upraviť tak, aby ignorovala vrcholy označené ako vymazané. Ak počet živých (nevymazaných) vrcholov klesne pod polovicu všetkých uzlov, prebudujeme celý strom.

Intuitívne: Prebudovanie celého stromu síce trvá $O(n)$, ale predtým muselo nastať aspoň $n/2$ operácií `delete`, takže amortizovaný čas jednej operácie `delete` zostáva $O(\log n)$.

Analýza časovej zložitosti

Pre každý vrchol v si zadefinujeme Δ_v ako rozdiel veľkostí medzi ľavým a pravým podstromom, v absolútnej hodnote.

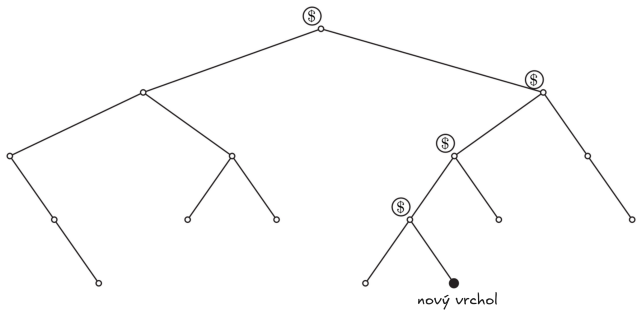
Budeme dodržiavať nasledovný invariant:

> Každý vrchol v má vždy našetrených *aspoň* $\Delta_v - 1$ dolárov.

Keď vrchol pridáme ako list, nemusí mať našetrené nič (pretože $\Delta_v = 0$). Rovnako, po prebudovaní perfektne vyváženého podstromu nemusí mať žiadny vrchol v ňom našetrené nič, lebo rozdiel veľkostí podstromov je najviac 1. Avšak čím je vrchol viac nevyvážený, tým musí mať aj viac našetrené.

Ukážeme, že ak za každú operáciu `insert` dostaneme $2\log_{3/2} N$\$, vystačí nám to:

- $\log_{3/2} N$ dolárov minieme hneď:
  - na nájdenie správneho miesta,
  - pripojenie nového listu,
  - návrat späť ku koreňu,
  - prepočítavanie veľkostí podstromov a kontrolu rovnováhy na ceste.
- $\log_{3/2} N$ dolárov si odložíme na neskôr:
  - každý vrchol na ceste od nového listu po koreň dostane 1 dolár.



Pri vkladaní sa zmení Δ_v iba pre vrcholy na ceste k novému vrcholu – a to najviac o 1. Zároveň do každého takéhoto vrcholu uložíme práve 1 dolár, takže invariant ostáva zachovaný. Ak strom po vložení zostáva vyvážený, operácia týmto končí.

Ukážme teraz, že ak dôjde k prebudovaniu nevyváženého podstromu, dokážeme túto operáciu zaplatiť z už našetrených mincí.

Povedzme, že prebudovaný podstrom má veľkosť k vrcholov. Keďže došlo k porušeniu rovnováhy, jeden z podstromov mal *viac ako* $\frac{2}{3}k$ vrcholov a druhý *menej ako* $\frac{1}{3}k - 1$ vrcholov. To znamená, že nevyvážený vrchol mal pred rekonštrukciou $\Delta_v \geq \frac{1}{3}k + 1$ a teda podľa nášho invariantu musel mať našetrených aspoň $\frac{1}{3}k$ dolárov.

Po prebudovaní budeme mať perfektne vyvážený podstrom a vrcholy v ňom nemusia mať nič našetrené. takže môžeme všetky našetrené doláre použiť na zaplatenie práce. A keďže prebudovanie podstromu veľkosti k trvá $O(k)$ času, máme dostatok financií na pokrytie celej rekonštrukcie.

A čo operácia delete?

Na jednu operáciu delete nám stačí $\log_{3/2} N + 1$ dolárov:

- $\log_{3/2} N$ dolárov zaplatíme ihneď za nájdenie a označenie vrcholu ako vymazaného,
- zvyšný jeden dolár si odložíme do prasiatka na neskôr.

Dodržíme tak invariant, že

> Strom, v ktorom je vymazaných D vrcholov má našetrených práve D dolárov.

Ak vymažeme polovicu vrcholov, strom má našetrených $N/2$ dolárov – a tie stačia na zaplatenie prebudovania celého stromu, ktoré trvá $O(N)$ času.

Pre korektnosť ešte musíme dodať, že N tu označuje počet všetkých vrcholov – vrátane tých, ktoré sme označili ako vymazané, ale zatiaľ fyzicky neodstránili. V analýze dátových štruktúr však vždy posudzujeme zložitosť v závislosti od skutočného počtu prvkov v strome $n = N - D$. Avšak keďže vymazaných vrcholov môže byť najviac polovica, $N \leq 2n$ a teda výška stromu je najviac $\log_{3/2}(2n) < \log_{3/2} n + 2$, takže všetky naše odhady ostávajú logaritmické.

Zhrnutie

Operácia	*Čas v najhoršom prípade*	*Amortizovaný čas*
`find`	$O(\log n)$	(scapegoat strom garantuje logaritmickú hĺbku vždy)
`insert`	$O(n)$	$O(\log n)$
`delete`	$O(n)$	$O(\log n)$

Úlohy

- Ako sa zmení maximálna výška stromu, ak namiesto konštanty $2/3$ zvolíme inú konštantu $\alpha \in (\frac{1}{2}, 1)$? Aké kritérium rovnováhy máme zvoliť, ak chceme strom s výškou $1.5 \lg n$ alebo $1.1 \lg n$?
- Ak si povieme, že pri prebudovaní podstromu veľkosti k vykonáme $c \times k$ inštrukcií, potom predpokladáme, že 1\$ zaplatí $3c$ inštrukcií. Ak namiesto konštanty $2/3$ zvolíme inú konštantu $\alpha \in (\frac{1}{2}, 1)$, koľkokrát viac (alebo menej) dolárov potrebujeme ušetriť pri každom `insert`e, aby sme vedeli prebudovanie podstromu stále zaplatiť?
- Rozmyslite si, ako algoritmus vkladania upraviť tak, aby sme si nemuseli v strome pamätať žiadnu informáciu navyše.

  Hint: Ak hĺbka nového uzla presiahne: $\log_{3/2} n$, kde n je aktuálny počet vrcholov v strome, vieme, že strom je príliš vysoký a musíme zasiahnuť. Ako nájdeme vinníka, keď nemáme predpočítané veľkosti podstromov? Ak je vinníkov viac, budeme musieť vybrať toho najnižšieho – rozmyslite si, či to stačí.