Semestrálny projekt z MI-PAL, LS 2016/17

Implementácia dátovej štruktúry

Implicit Treap

Peter Mitura (miturpet@fit.cvut.cz)

11. mája 2017

Treap

Treap (názov zložený zo slov *tree* a *heap*) je stromová dátová štruktúra, ktorá je kombináciou binárneho vyhľadávacieho stromu (ďalej *BVS*) a haldy. Základy tejto štruktúry vychádzajú z *kartézskych stromov* [1] – binárnych stromov postavených nad poľom, ktoré spĺňajú haldovú podmienku a sú usporiadané tak, že ich in-order priechod vypíše originálne pole.

Treapy sú rozšírením tejto myšlienky. Tiež sú to binárne stromy, v každom svojom vrchole ale ukladajú dve hodnoty: *kľúč* a *prioritu*, pri čom simultánne sa udržujú zoradené ako BVS podľa kľúča a ako minimová halda podľa priority. Nakoľko sa najčastejšie používajú ako implementácia abstraktného dátového typu *množina* (tzn. efektívna implementácia funkcií *Insert*, *Find* a *Delete*), kľúče vrcholov je logické využiť na uloženie prvkov v množine. Priority potom slúžia jedine k vyvažovaniu stromu, pri čom ďalej ukážeme, že ak ich budeme voliť náhodne z rovnomerného rozloženia nad dostatočne veľkým rozsahom, strom získa z hľadiska efektivity operácií nad kľúčmi zaujímavé vlastnosti.

Treap v tejto podobe bol po prvý krát popísaný Aragonom a Seidlom v roku 1989 [2], s určitými rozdielmi v názvosloví. [1] Základné operácie, podporované touto štruktúrou zodpovedajú, ako už bolo zmienené, abstraktnému dátovému typu množina:

- $Insert(x)$ – vloží prvok x do stromu
- $Find(x)$ – zistí, či sa prvok x nachádza v strome
- $Delete(x)$ – zmaže prvok x zo stromu

[1] Aragon a Seidel pri pojme *Treap* nešpecifikovali spôsob voľby priority, náhodnú hodnotu jej dávali až v štruktúre ktorú pomenovali *Randomized Search Tree* – tento pojem sa ale neskôr začal využívať pre zložitejšiu dátovú štruktúru a označenie Treap sa zaužíval pre verziu s náhodnou voľbou priority.

Podobne ako u iných BVS, aj tu od univerza vkladaných prvkov požadujeme, aby tvorilo úplne usporiadanú množinu a reláciu usporiadania je nutné definovať pri inicializácií štruktúry (štandardne v implementácií využívame funkciu `std::less`, ak je dostupná). Základnou myšlienkou potom je, že všetky operácie implementujeme analogicky ako v bežnom BVS, s jediným rozdielom že v nich chceme zachovať haldové usporiadanie podľa priority bez zvýšenia asymptotickej časovej zložitosti.

Ak by sa nám to podarilo (konkrétnu implementáciu popíšeme nižšie), zložitosť operácií nad stromom T by dosiahla $\mathcal{O}(h(T))$, kde $h(T)$ je hĺbka daného stromu. Tá by v nevyvažovanom BVS mohla dosiahnúť až $\mathcal{O}(n)$, kde n je počet prvkov v strome, čo pre nás rozhodne nie je priaznivé. Kľúčové pre celú štruktúru je teda nasledujúce tvrdenie:

Veta 1. Ak je priorita každého vrchola nezávislá, rovnomerne rozložená spojitá náhodná veličina, stredná hodnota hĺbky *ľubovoľného* vrchola v Treape je $\mathcal{O}(\log n)$, kde n je počet vrcholov v Treape.

Pri dôkaze tejto vety sa inšpirujeme postupom z kurzu na Univerzite v Illinois [3], ktorý je do istej miery prehľadnejší než dôkaz v originálnom článku.

Počet vrcholov v strome budeme naďalej označovať ako n. Ďalej si označme vrchol s k-tým najmenším kľúčom ako x_k a počet vrcholov. Zároveň si zavedieme sadu premenných A_k^i, ktorá bude indikovať či je vrchol x_i predkom vrchola x_k (predka zavádzame tak, že vrchol nie je predkom sám sebe):

$$A_k^i = \begin{cases} 1 & \text{ak je } x_i \text{ predkom } x_k \\ 0 & \text{inak} \end{cases}$$

Pomocou indikátora A dokážeme jednoducho vyjadriť hĺbku vrchola $h(x_k)$:

$$h(x_k) = \sum_{i=1}^{n} A_k^i \tag{1}$$

Takže stredná hodnotu hĺbky vrchola x_k bude rovná sume pravdepodobností, že x_i je predkom x_k nad všetkými x_i.

$$\mathbb{E}(h(x_k)) = \sum_{i=1}^{n} P(A_k^i = 1) \tag{2}$$

Veta 1 o tejto hodnote tvrdí, že je rovná $\mathcal{O}(\log n)$. Pre overenie tejto rovnosti si zavedieme ešte jedno značenie, $X(i,k)$ bude označovať množinu vrcholov $\{x_i, x_{i+1}, \ldots, x_k\}$ alebo $\{x_k, x_{k-1}, \ldots, x_i\}$ podľa toho či $i < k$ alebo $i > k$.

Lemma 1. Pre každé $i \neq k$ platí, že x_i je predkom x_k vtedy a len vtedy, keď má x_i najnižšiu prioritu spomedzi vrcholov v $X(i,k)$.

Dôkaz. V prípadoch kedy je x_i alebo x_k koreňom stromu je rovno vidieť, že lemma platí (x_i resp. x_k má vtedy najnižšiu prioritu z celého stromu), obmedzíme sa teda na prípad kedy je koreňom x_j pre nejaké $i \neq j \neq k$. Potom môžeme rozlíšiť dve možnosti: x_i a x_k sú buď v rovnakom alebo rozdielnom podstrome x_j.

Ak sú v rôznom podstrome, tak x_j určite patrí do $X(i,k)$ (jedná sa o BVS, takže jedna hodnota z dvojice i, k musí byť menšia než j, a jedna väčšia než j). Priorita x_j je

ale najnižšia z celého stromu, takže aj z $X(i,k)$, čo by podľa lemmy malo znamenať že x_i nie je predkom x_k (a nakoľko sú v rôznych podstromoch, je to aj pravda).

Pre x_i a x_k v jednom podstrome môžeme uvažovať len daný podstrom a v ňom postupovať analogicky ako v prípadoch vyššie (rekurzívne). □

Pravdepodobnosť že vrchol x_i je predkom x_k je teda zhodná s pravdepodobnosťou že x_i má minimálnu prioritu z $X(i,k)$ a nakoľko je táto šanca pre všetky vrcholy z daného rozsahu rovnaká, môžeme ju jednoducho vyjadriť ako 1/počet vrcholov v rozsahu:

$$P(A_k^i = 1) = \begin{cases} \dfrac{1}{k-i+1} & \text{ak } i < k \\ \dfrac{1}{i-k+1} & \text{ak } i > k \\ 0 & \text{ak } i = k \end{cases}$$

Zostáva už len vyjadriť hodnotu strednej hodnoty $\mathbb{E}(h(x_k))$. Z 2 máme jej vzťah k $P(A_k^i)$, ich sumu môžeme rozdeliť na 2 časti podľa vzťahu medzi i a k:

$$\sum_{i=1}^{n} P(A_k^i = 1) = \sum_{i=1}^{k-1} \frac{1}{k-i+1} + \sum_{i=k+1}^{n} \frac{1}{i-k+1} \tag{3}$$

(prvá suma je prípad s $i < k$, druhá $i > k$)

Tieto sumy ďalej môžeme previesť na harmonické rady, pre ktoré máme známe odhady:

$$\sum_{i=1}^{k-1} \frac{1}{k-i+1} + \sum_{i=k+1}^{n} \frac{1}{i-k+1} = \sum_{j=2}^{k} \frac{1}{j} + \sum_{j=2}^{n-k} \frac{1}{j} = H_k - 1 + H_{n-k} - 1 \tag{4}$$

$$H_k - 1 + H_{n-k} - 1 < \ln k + \ln(n-k) - 2 < 2\ln n - 2 \tag{5}$$

čo asymptoticky dáva

$$\mathbb{E}(h(x_k)) < 2\ln n - 2 = \mathcal{O}(\log n) \tag{6}$$

čím je veta 1 dokázaná.

Implementácia operácií

Ako sme už uviedli vyššie, vďaka predpokladu logaritmickej hĺbky stromu v priemernom prípade nám stačí implementovať operácie *Insert*, *Delete* a *Find* v zložitosti zodpovedajúcej hĺbke stromu.

Pre zjednodušenie ich implementácie (ktorá by inak musela byť riešená komplikovanými rotáciami) sa bežne zavádza dvojica primitívnych operácií *Split* a *Merge*:

- $Split(x,t)$ – rozdelí Treap s koreňom vo vrchole t na dva Treapy: ľavý Treap l, v ktorom sú kľúče všetkých vrcholov menšie rovné hodnote x a pravý Treap r, v ktorom sú kľúče všetkých vrcholov väčšie než x. Originálny Treap v tomto procese zanikne.
- $Merge(l,r)$ – spojí dva Treapy do jedného, pod podmienkou že maximálny kľúč z l je menší ako minimálny kľúč z r.

Realizácia operácie $Split$

Pre potreby práce so stromom označíme ľavého syna vrchola v ako $v.left$ a pravého ako $v.right$. Operácia $Split(x, t)$ môže potom naraziť na 3 prípady:

- Ak je x menšie ako kľúč v t, tak ako r vrátime vrchol t s jeho pôvodným pravým podstromom, a ako ľavý podstrom mu priradíme pravý strom z výsledku operácie $Split(x, t.left)$. Takto zvolené r bude určite spĺňať haldovú podmienku (pôvodný strom bol Treap, takže všetky prvky z ľavého podstromu t musia mať väčšiu prioritu ako r) a zároveň sú všetky kľúče v ňom väčšie ako x (pre vrchol t a jeho pravý podstrom to platí nakoľko sa jedná o BVS, pre pravý strom zo splitu ľavého podstromu t to platí z definície operácie $Split$).

  Ako Treap l ďalej vrátime ľavý strom získaný operáciou $Split(x, t.left)$ (ktorú sme už zavolali pre zísakanie Treapu r). Z definície $Split$-u sa musí jednať o Treap so všetkými kľúčmi menšími než x, takže spĺňa všetky podmienky.

- Ak je x väčšie alebo rovné kľúču v t, potom vykonáme reverznú verziu minulého prípadu – ako Treap l vrátime vrchol t s jeho originálnym ľavým podstromom a jeho pravý podstrom nahradíme ľavým Treapom z operácie $Split(x, t.right)$. Ako pravý Treap vrátime pravý strom získaný tou istou funkciou.

- Ak je t prázdny Treap (napr. v situácií kedy sa zanoríme do neexistujúceho syna), vrátime ako l aj r prázdny Treap.

Je zjavné, že operácia $Split$ sa v každej instancií môže rekurzívne zanoriť maximálne jeden krát do vrcholu o jednu úroveň nižšie. Jej časová zložitosť v najhoršom prípade je teda lineárna v hĺbke stromu, čo v priemere podľa vety 1 dáva asymptoticky $O(\log n)$.

Realizácia operácie $Merge$

Literatúra

[1] J. Vuillemin, „A unifying look at data structures“, Communications of the ACM, vol. 23, no. 4. Association for Computing Machinery (ACM), pp. 229–239, 1. apríla 1980.

[2] C. R. Aragon a R. G. Seidel, „Randomized search trees“, 30th Annual Symposium on Foundations of Computer Science. IEEE, 1989.

[3] J. Erickson, „Randomized Binary Search Trees“, 2013. [Online]. Dostupné z: `http://jeffe.cs.illinois.edu/teaching/algorithms/notes/10-treaps.pdf`. [prístup 9. mája 2017].