Externé DŠ

©kuko 15.5.2018

1 I/O model

- I/O model (známy tiež ako external memory / disk access model / cache aware model):
  - máme neobmedzenú externú pamäť + cache veľkosti M
  - počítať môžeme iba s údajmi v cache
  - pamäť je rozdelená do blokov veľkosti B a vždy, keď čítame/zapisujeme údaje z/do externej pamäte, číta/zapisuje sa celý blok
  - okrem časovej a pamäťovej zložitosti algoritmov nás bude zaujímať najmä IO zložitosť – počet prístupov (čítaní + zápisov blokov) do pamäte
- zjavne IO zložitosť $\leq$ časová zložitosť (v najhoršom prípade spraví každá operácia cache miss)
- naopak, IO zložitosť $\geq$ min #prístupov k jednotlivým slovám / B

1.1 vyhľadávanie:

- $O(\log(N/B)) = O(\log N - \log B)$ binárne vyhľadávanie v utriedenom poli
- $O(\log(N/B)) = O(\log N - \log B)$ v úplnom binárnom strome, ktorý uložíme po úrovniach (BFS usporiadanie)
- $O(\log_B N) = O(\log N / \log B)$ – B-strom
- v porovnávacom modeli to lepšie nejde:
  - treba aspoň $\Omega(\log N)$ porovnaní – intuitívne: na zakódovanie pozície potrebujeme $\log N$ bitov, každým porovnaním získame len 1 bit informácie ($<$ alebo $>$; ak predpokladáme, že všetky kľúče sú rôzne a hľadaný prvok sa v nich nenachádza, čo je najhorší prípad)
  - $\Omega(\log_B N)$ prístupov do pamäte (po blokoch veľkosti B) – každý blok nám dá $\log B$ bitov informácie
- ako vyhľadávať, ak nepoznáme B?

1.2 triedenie

- quicksort/heapsort/mergesort – $O(N \log N)$ čas; IO?
- lepšie: utriedime najskôr úseky dĺžky M, potom mergesort – $O(\frac{N}{B} \log \frac{N}{M})$
- ešte lepšie: nebudeme robiť merge binárne, po dvoch, ale čo najviac ako sa dá: $M/B - 1$ postupností naraz
- do pamäte vždy načítame 1 blok z každej postupnosti a merge-ujeme; zvyšný 1 blok použijeme na výstup
- vždy, keď sa výstupný blok zaplní, zapíšeme ho späť do externej pamäte, vždy, keď sa vstupný blok minie, načítame ďalší blok postupnosti
- výsledná zložitosť $O(\frac{N}{B} \log_{M/B} \frac{N}{M}) = O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ (ak $N/M = (M/B)^k$, tak $N/B = (M/B)^{k-1}$)
- akú časovú zložitosť má tento algoritmus?
- ako triediť, ak nepoznáme M, B?

2 Cache oblivious model

- cache oblivious (CO) model – ako I/O model, ale:
  - algoritmus nepozná M ani B (hoci v analýze zložitosti sa vyskytnú)
  - cache je spravovaná automaticky (predpokladáme optimálny algoritmus, ktorý dopredu vie, ktoré dáta bude kedy treba; optimálne je vždy vyhodiť ten blok, ktorý bude treba čo najneskôr)

- v praxi potrebujeme cache spravovať online; dá sa však dokázať, že FIFO / LRU cache nie sú ďaleko od optima
- presnejšie: FIFO aj LRU cache s pamäťou M spraví len $O(1)\times$viac prístupov do pamäte ako optimálny offline algoritmus s pamäťou $M/2$
- predpokladáme tiež, že cache je plne asociatívna, tzn. každý blok pamäte sa môže uložiť hocikde do cache (v praxi niektoré cache sú iba k-asociatívne pre malé k, povedzme 8 – tzn., že každý blok pamäte sa môže uložiť iba na 1 z k možných pozícií – to zmenšuje efektívnu veľkosť cache, pretože zopár blokov sa môže navzájom vytláčať)
- plne asociatívna cache sa dá simulovať napr. pomocou hešovania

- výhoda: keďže algoritmus nepozná M, B, funguje dobre pre všetky M, B
- prekvapenie: vo veľa prípadoch vieme dosiahnuť rovnakú zložitosť ako v I/O modeli

3 Van Emde Boasove usporiadanie

- ukážeme, ako sa dá vyhľadávať v statickej množine v $O(\log_B N)$ (v I/O modeli B-strom, ale teraz nepoznáme B)
- použijeme úplný binárny strom, pričom do pamäte ho uložíme podľa tzv. van Emde Boasovho (vEB) usporiadania
- strom výšky h rozdelíme v strede – na horný strom výšky $h/2$ a spodné stromy výšky $h/2$
- hore máme 1 strom s $\approx \sqrt{n}$ prvkami, dole zhruba $\sqrt{n}$ stromov s $\approx \sqrt{n}$ stromami (ak $n = 2^h$, potom $2^{h/2} = n^{1/2} = \sqrt{n}$)

```
              1
      2               3
-----------------------------
  4  |   7  |   10  |   13
5  6 | 8  9 | 11 12 | 14 15
```

- rekurzívne usporiadame každý podstrom a výsledné postupnosti zreťazíme; dôležité je, že každý podstrom v poli zaberá 1 súvislý úsek

- tvrdíme, že na vyhľadávanie v takto uloženom strome stačí $O(\log_B N)$ IO
- „zoomnime“ si do vEB-usporiadania na úroveň rekurzie, keď sa podstromy celé zmestia do jedného bloku cache ($\leq B$)
- tzn. veľkosť s týchto podstromov je $\sqrt{B} < s \leq B$; ich výška h je $\frac{1}{2}\log B < h \leq \log B$
- pri vyhľadávaní v strome prejdeme cestu od koreňa dĺžky $\leq \log N$, pričom navštívime $\leq \log N/h$ stromov výšky h, t.j. $O(\log N/\log B)$
- každý podstrom je v pamäti uložený ako súvislý úsek $\leq B$ prvkov, takže potrebujeme načítať ≤ 2 bloky

4 Usporiadaný súbor

- chceme udržovať zoznam prvkov (v danom poradí); operácie:
  - vlož nový prvok medzi dané 2 prvky
  - vymaž daný prvok
- toto je jednoduché pomocou obojsmerného spájaného zoznamu v čase $O(1)$
- ak je zoznam utriedený, môžeme použiť vyvážený BST a podporovať navyše vyhľadávanie v $O(\log N)$
- ťažšia úloha: ako takýto zoznam reprezentovať v poli? bez pointrov?

- ak zoznam uložíme ako súvislý úsek v poli, potrebujeme $O(N)$ na posunutie všetkých prvkov

- ak by sme mali medzi prvkami medzery veľkosti k, dokážeme spraviť $O(\log k)$ vložení bez toho, aby sme čokoľvek posunuli
- táto úloha vyzerá dosť beznádejne; prekvapenie: dokážeme udržiavať N prvkov v poli veľkosti $O(N)$, pričom medzi každými dvoma prvkami je medzera najviac $O(1)$ dĺžky a vkladanie a vymazávanie trvá len $O(\log^2 N)$ amortizovane
- dá sa ukázať, že lepšie ako $\Omega(\log^2 N)$ sa za takýchto podmienok nedá
- bonus: algoritmus iba upraví súvislý úsek poľa pomocou $O(1)$ sekvenčných prechodov, takže sa dá jednoducho implementovať v CO modeli s $O(\log^2 N/B)$ IO amortizovane

- riešenie:
- rozdeľme prvky do blokov dĺžky $O(\log N)$
- nad týmito úsekmi vybudujme implicitný binárny strom (tento strom používame len pri vysvetlení a analýze algoritmu – nie je súčasťou DŠ)
- každý vrchol zodpovedá intervalu v pôvodnom poli
- definujme hustotu v ako #prvkov/dĺžka intervalu; nech d je hĺbka vrcholu a $h = O(\log N)$ výška stromu mínus 1
- budeme sa snažiť, aby hustota v bola $\geq \frac{1}{2} - \frac{1}{4} \cdot \frac{d}{h}$ a $\leq \frac{3}{4} + \frac{1}{4} \cdot \frac{d}{h}$
- tzn., pre koreň je $d = 0$ a hustota by mala byť medzi $\frac{1}{2}$ a $\frac{3}{4}$ – koreň by teda mal byť zaplnený na 50–75%
- pre listy je $d = h$ a hustota by mala byť medzi $\frac{1}{4}$ a 1 – t.j. listy budú zaplnené na 25–100%
- pre vrcholy na rôznych úrovniach bude dolná hranica niekde medzi 25–50% a horná niekde medzi 75–100%, pričom čím je vrchol vyššie, tým je naše obmedzenie striktnejšie, čím nižšie, tým sú hranice voľnejšie
- navyše rozsahy 25–50% a 75–100% interpolujeme lineárne; keďže hĺbka je $O(\log N)$, platí, že vrchol na úrovni o 1 nižšie má rozsah hustoty o $O(1/h) = O(1/\log N)$ voľnejší

- vkladanie: prvok vložíme na danú pozíciu, pričom posunieme prvky v bloku, ak treba; ak sa blok preplní, ideme v strome nahor, až kým nenájdeme taký vrchol, že aj po vložení bude mať hustotu v svojich medziach – celý interval, ktorý prislúcha tomuto vrcholu prepíšeme tak, že všetky prvky v ňom rovnomerne rozdistribuujeme
- vymazávanie: podobne ako vkladanie – prvok vymažeme, nájdeme najnižší vrchol s hustotou v medziach a prvky v ňom rozdistribuujeme
- pripomeňme, že binárny strom je iba koncepčný, v skutočnosti iba skenujeme pole doľava alebo doprava (podľa toho, či bol daný vrchol pravý syna alebo ľavý syn), pričom si priebežne počítame dĺžku intervalu a #prvkov v ňom (a teda hustotu) a hĺbku, v ktorej sme (a povolený rozsah hustoty)
- poznámka: listy budú vždy v medziach, ale ostatné vrcholy nemusia nutne byť a napravíme ich až neskôr (napríklad ak by sme do takejto DŠ vkladali prvky zľava doprava, môžeme zaplniť všetky listy na 100% – zjavne koreň ani iné vnúrné vrcholy nebudú v medziach, ale napravíme ich až vtedy, keď prvý list pretečie)
- opačná implikácia však platí: ak je vrchol v hĺbke d v medziach hustoty, potom sú určite v medziach aj všetky vrcholy pod ním
- navyše, tesne po redistribuovaní platí, že vrcholy nižšie nielenže sú v medziach, ale sú oveľa viac „v pohode“ – majú voľnejšie hranice a ich hustota je ďalej od hraníc, konkrétne o $O(1/h) = O(1/\log N)$ ďalej od hraníc
- keďže listy zodpovedajú blokom dĺžky $O(\log n)$, pre vhodne zvolené konštanty hustota $O(1/\log N)$ zodpovedá aspoň 1 prvku
- analýza zložitosti (pre vkladanie):
  - ak sme sa dostali do vrcholu v a redistribuovali interval dĺžky k, potom reálna zložitosť je $O(k)$ – potrebujeme ukázať, ako hospodáriť tak, aby sme si medzičasom nasporili $O(k)$\$
  - po redistribúcii je vrchol v a všetky vrcholy pod ním v medziach; najbližšie tu skončíme, až keď sa jeden zo synov preplní
  - synovia vrcholu v majú vzdialenosť od hranice aspoň $O(1/\log N)$, čo zodpovedá $O(k/\log N)$ prvkom

- ak teda každý nový prvok vloží na účet v $O(\log N)$\$, potom po vložení $O(k/\log N)$ prvkov, keď sa syn v preplní, bude mať v ušetrené $O(k)$\$ a teda bude vedieť zaplatiť prebudovanie celého intervalu dĺžky k
- $O(\log N)$\$ treba vložiť na účet každému vrcholu na ceste od listu ku koreňu (pretože vloženie 1 prvku mení hustotu v každom z týchto vrcholov), preto $O(\log^2 N)$ amortizovane

5 Cache-oblivious B-strom

- prvky uložíme utriedené do DŠ pre usporiadaný súbor; nad ním zostrojíme úplný binárny strom, ktorý uložíme vo vEB-usporiadaní
- listy v strome zodpovedajú políčkam v usporiadanom súbore (aj prázdnym) a pomocou uložených pointrov vieme skákať medzi políčkom a listom (oboma smermi)
- vo vnutorných vrcholoch si udržiavame maximum podstromu – vďaka tomu vieme v strome vyhľadávať
- hľadanie: $O(\log_B N)$ vďaka vEB-usporiadaniu
- vkladanie a vymazávanie: nájdeme daný prvok/miesto nový prvok a spravíme príslušnú operáciu v usporiadanom súbore; vo všeobecnosti sa pri tom zmení interval dĺžky k – príslušný interval + všetky vrcholy nad ním upravíme v jednom post-order prechode
- ukážeme, že vďaka vEB-usporiadaniu trvá prepísanie intervalu dĺžky k + vrcholov nad ním $O(k/B + \log_B N)$, z čoho dostaneme $O((\log^2 N)/B + \log_B N)$ amortizovane

- „zoomnime“ si do vEB-usporiadania na úroveň rekurzie, keď sa podstromy celé zmestia do jedného bloku cache ($\leq B$)
- analýzu rozdelíme na 3 časti:
- 1.) vrcholy v spodných 2 úrovniach vEB podstromov
  - každý podstrom tvorí súvislý úsek v pamäti, t.j. stačí načítať ≤ 2 bloky
  - v post-orderi striedavo prechádzame medzi týmito dvoma úrovňami, pričom prejdeme 1 celý podstrom na spodnej úrovni, potom pár vrcholov na 2. úrovni, potom zase 1 celý podstrom na spodnej úrovni, potom pár vrcholov na 2. úrovni... až kým neprejdeme celý podstrom na 2. úrovni
  - ak teda máme cache s aspoň 4 blokmi ($M \geq 4B$), zmestia sa doňho 2 podstromy na posledných 2-och úrovniach – dôležité je, že podstrom na 2. úrovni nevytlačíme z cache až kým neprejdeme všetky vrcholy v danom podstrome – takto $O(k)$ vrcholov v spodných 2 úrovniach prejdeme na $O(k/B)$ IO
- 2.) všetky vrcholy nad nimi, až po LCA
  - malé podstromy, ktoré sa zmestia do 1 bloku majú veľkosť aspoň $\sqrt{B}$ – preto ak vezmeme 2 úrovne, dostaneme stromy veľkosti aspoň B
  - tým pádom #ich koreňov je najviac $\lceil k/B \rceil$ a teda zložitosť bude $O(1+k/B)$, aj keby každý vrchol znamenal 1 cache miss
- 3.) cesta od LCA po koreň – cesta je dĺžky $\leq \log N$ a vďaka vEB stačí $O(\log_B N)$ IO

- či je väčšie $O(\log^2 N/B)$ alebo $O(\log_B N) = O(\log N/\log B)$ závisí od B – pre dostatočne veľkú dĺžku bloku $B = \Omega(\log N \log\log N)$ dominuje $O(\log_B N)$
- vedeli by sme sa zbaviť $O(\log^2 N/B)$? áno:
  - rozdeľme prvky do $\Theta(N/\log N)$ skupín veľkosti $\Theta(\log N)$
  - každá skupina je obyčajné pole zaplnené na 25–100%; ľubovoľná operácia na ňom trvá $O(\log N/B) = O(\log_B N)$
  - v B-strome popísanom vyššie budeme udržiavať maximá jednotlivých skupín
  - vkladanie a mazanie v B-strome treba spraviť iba vtedy, keď sa niektorá skupina preplní/podtečie, čo je raz za $\Omega(\log N)$ operácií
  - tzn., ak každá operácia zaplatí $\log N/B$\$, po $\Omega(\log N)$ operáciach budeme mať nasporené $\Omega(\log^2 N/B)$, takže túto časť vieme zaplatiť
- dostávame tak vkladanie/vymazávanie/hľadanie v $O(\log_B N)$ – čo je optimálne

- nevýhoda: s usporiadaným poľom vieme robiť range query a prejsť interval k prvkov v $O(k/B)$ $(+O(\log_B N)$ na nájdenie začiatku a konca) – v poslednej DŠ treba na prechod $O(k/B + k/\log N)$, pretože po prejdení každej skupiny $\Theta(\log N)$ prvkov máme cache miss