Haldy

©kuko 2.10.2018

1 Prioritné fronty

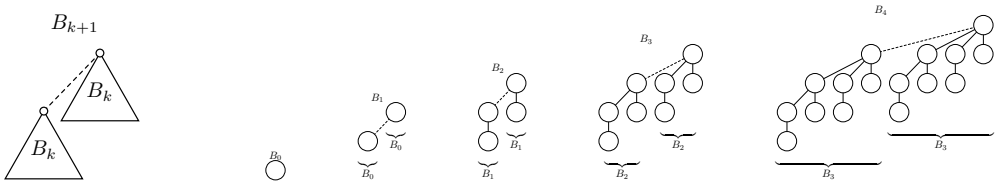
- chceme „čiernu krabičku“, ktorá dokáže operácie:
  - insert
  - delete min
  - decrease key
  - prípadne merge
- s takouto krabičkou vieme napr. triediť: $n\times$insert a následne $n\times$delete min
- tiež sa hodí na implementáciu Dijkstrovho algoritmu: vložíme začiatočný vrchol so vzdialenosťou 0 a zvyšné vrcholy so vzdialenosťou ∞; následne vždy vyberieme vrchol s najmenšou vzdialenosťou a všetkým susedom potenciálne znížime vzdialenosť (ak je to tadiaľto kratšie)
  - celkový čas: $n \times (\text{insert} + \text{delete min}) + m \times \text{decrease key}$
- všetci vedia, že prioritná fronta sa efektívne implementuje pomocou haldy, ale: dá sa lepšie? ak áno, ako? a ak nie, prečo nie? a dajú sa efektívne spojiť dve haldy?
- napr. existuje halda, ktorá spraví insert a delete min v $O(1)$, ale decrease key v $O(\log n)$?
  - NIE, pretože potom by sme vedeli triediť v $O(n)$ a to sa v porovnávacom modeli nedá
- existuje halda, ktorá spraví insert a decrease key v $O(1)$, ale delete v $O(\log n)$?
  - prekvapivo ÁNO a v tejto prednáške si ukážeme, ako sa to dá
  - pozn.: v stĺpci lenivá a Fibonacciho halda sú zložitosti amortizované, nie v najhoršom prípade; existujú aj DŠ dosahujúce rovnaké časy v najhoršom prípade, ale sú pomerne komplikované
  - pozn. 2: binomiálne a Fib. haldy sú v praxi pomerne pomalé (konštanty v Óčku sú pomerne veľké) a v praxi je lepšie použiť binárnu/d-árnu haldu; otvorený problém: existuje halda, ktorá má aj dobrú asymptotickú zložitosť (ako Fib. halda) a zároveň je efektívna v praxi?

	pole	binárna halda	d-árna halda	binomiálna halda	lenivá bin. halda	Fibonacciho halda
insert	$O(1)$	$O(\log n)$	$O(\log_d n)$	$O(\log n)$	$O(1)$	$O(1)$
delete min	$O(n)$	$O(\log n)$	$O(d \log_d n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
decrease key	$O(1)$	$O(\log n)$	$O(\log_d n)$	$O(\log n)$	$O(\log n)$	$O(1)$
merge	$O(n)$	$O(n)$	$O(n)$	$O(\log n)$	$O(1)$	$O(1)$
Dijkstrov alg.	$O(n^2)$	$O(m \log n)$	$O(m \log_{m/n} n)$			$O(m + n \log n)$

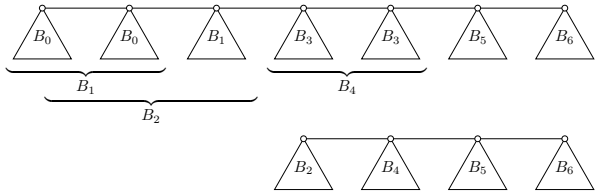
- pre Dijkstrov algoritmus:
  - obyčajné pole je najlepšia (a optimálna) možnosť, ak máme veľmi hustý graf ($m = \Theta(n^2)$)
  - v praxi sú grafy väčšinou riedke (planárne grafy, graf priateľstiev, kde max. stupeň je relatívne malý); $m \log n$ je preto oveľa lepšie ako n^2
  - d-árna halda má trade-off medzi insert, decrease key (rýchlejšie) a delete min (pomalšie); pre Dijkstrov alg. dostávame $O(m \log_d n + nd \log_d n)$, čo je najmenšie, ak vezmeme $d = m/n$ (resp. $\max(2, m/n)$); všimnite si, že pre $m = \Omega(n^{1+\varepsilon})$ je $\log_{m/n} n = \log_{n^\varepsilon} n = O(1/\varepsilon)$, takže pre grafy s aspoň $\Omega(n^\varepsilon)$ hranami je to optimálny algoritmus
  - s Fib. haldou dostávame $O(m + n \log n)$, čo je pre $m = \Omega(n \log n)$ optimálne
  - dá sa lepšie? dá sa nájsť najkratšia cesta v $O(m)$ aj pre riedke grafy? napr. $m = O(n)$?

2 Binomiálna halda

- B_k – binomiálny strom s rankom k definujeme rekurzívne:



- B_0 je jediný vrchol a B_{k+1} vznikne spojením dvoch B_k stromov
- zjavne $|B_k| = 2^k$ a koreň B_k má k synov – stromy $B_0, \ldots, B_{k-1}$
- v binomiálnej halde budeme požadovať, aby bol vždy otec menší ako všetci synovia, takže najmenší prvok je v koreni
- binomiálna halda je zoznam niekoľkých B_i + smerník na minimum
- navyše platí, že všetky ranky koreňov musia byť rôzne; z toho (a faktu $|B_k| = 2^k$) vyplýva, že tvar binomiálnej haldy s n vrcholmi zodpovedá binárnemu zápisu čísla n – halda obsahuje strom B_k, ak je k-ty bit 1
  - napr. halda so 41 vrcholmi ($41 = 32 + 8 + 1 = 2^5 + 2^3 + 2^0$) sa bude skladať z B_0, $B - 3$ a B_5
- operácia merge: jednoducho spojíme oba zoznamy a pospájame stromy s rovnakým rankom
  - funguje to rovnako ako sčítanie v dvojkovej sústave: tak ako $2^k + 2^k = 2^{k+1}$, $B_k + B_k \to B_{k+1}$
  - napr. spojenie 41-prvkovej haldy (B_0, B_3, B_5) a 75-prvkovej (B_0, B_1, B_3, B_6) dostaneme B_2, B_4, B_5, B_6:



- insert: vytvoríme novú 1-prvkovú haldu a primergujeme ju
- delete min: odstránime minimum, čo je jeden z koreňov; zo synov (stromy $B_0, B_1, B_2 \ldots$) vytvoríme novú haldu, ktorú pri-merge-ujeme k pôvodnej
- decrease key: znížime a vybubleme nahor
- keďže počet binomiálnych stromov a tiež maximálny rank a maximálna výška stromov je $O(\log n)$, všetky operácie trvajú $O(\log n)$ v najhoršom prípade

3 Lenivá binomiálna halda

- zrušíme podmienku, že všetky ranky musia byť rôzne
- merge bude lenivý: iba napojíme 2 spájané zoznamy (min je menší prvok z dvoch miním v $O(1)$ čase)
- „upraceme“ až počas operácie delete min: vtedy pospájame stromy s rovnakými rankami a nájdeme nové minimum
- oproti „obyčajnej“ binomiálnej halde zlepšíme čas merge a insert na $O(1)$ amortizovane
- delete min bude v najhoršom prípade až $O(n)$, ale amortizovane $O(\log n)$
- konkrétne tvrdím, že ak dostanem za každú operáciu insert: 2\$, merge: 1\$, delete min: $3 \log n + 1$\$, dokážem všetky kroky algoritmu zaplatiť
- budem dodržiavať nasledovný invariant: každý koreň stromu má nasporený 1\$

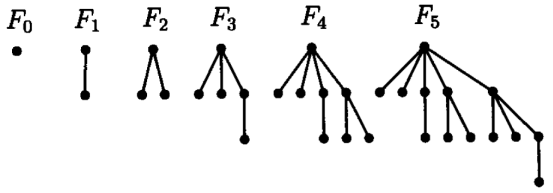
- merge: za 1$ spojíme dva zoznamy a vyberieme menšie z dvoch miním
- insert: za 1$ vytvoríme nový vrchol a pripojíme ho, zvyšný 1$ si vrchol odloží na neskôr, aby sme zachovali invariant
- delete min: vymazanie koreňa a pripojenie detí – 1$; nájdenie nového minima – $\log n$$; navyše každý syn je nový koreň a musíme mu dať 1$ na účet (spolu $\leq \log n$$)
- ostáva zaplatiť čas na „upratovanie“ – prechádzame zľava doprava, v pomocnom poli si pamätáme, aké ranky stromov máme a spájame stromy s rovnakým rankom
- toto upratovanie trvá až $O(t)$, kde t je počet stromov, avšak my ho dokážeme zaplatiť z už našetrených peňazí:
- vždy, keď nájdeme 2 stromy s rovnakým rankom, porovnanie a spájanie zaplatí ten strom, ktorý sa pripojí pod koreň s menším kľúčom („víťaznému“ koreňu ostane 1$, takže invariant ostane zachovaný)
- takto zaplatíme za všetky vrcholy, ktoré sa napoja pod iné; ostáva zaplatiť za prechod zvyšných vrcholov – tých, ktoré ostanú koreňmi aj na konci – ale tých je len $\leq \log n$

4 Fibonacciho halda

- motivácia: v Dijkstrovom algoritme len $n\times$ vkladáme/vyberáme minimum, ale potenciálne až $m\times$ znižujeme váhu – existuje halda, v ktorej je znižovanie váhy rýchle?
- nápad #1: ak chceme vrcholu x znížiť váhu, namiesto bublania x aj celý jeho podstrom odrežeme a pripojíme do zoznamu
- v zozname teraz už nebudeme mať nutne binomiálne stromy, ale akési osekané zvyšky
- rank vrcholu definujeme ako #synov a pri upratovaní spájame vrcholy s rovnakým rankom
- jednoduché, ale stačí to? ukazuje sa, že NIE
- pri lenivej binomiálnej halde bola veľkosť stromu exponenciálna od ranku; tým pádom bol rank (#synov) najviac $O(\log n)$ a pri vymazaní minima sme mohli pripojenie synov, nájdenie nového minima a vloženie 1$ na ich účet zaplatiť
- problém: ak by sme „bezhlavo“ odsekávali podstromy, už nemusí platiť, že rank je $O(\log n)$
- a nedá sa ten dôkaz zachrániť inak? nemohli by sme zmeniť analýzu/invarianty a šetriť inak? ukazuje sa, že NIE
- náznak dôkazu:
  - nech S_k je strom zložený z $k+1$ vrcholov – iba koreň a k synov; nech T_k je strom, ktorý sa skladá z koreňa a deti sú stromy $S_0, S_1, \ldots, S_{k-1}$;
  - T_k má $\Theta(k^2)$ vrcholov a dá sa vyrobiť pomocou $O(k^3)$ operácií
  - všimnite si, že ak napojíme S_k pod koreň T_k, dostaneme T_{k+1}
  - začneme s T_k, insert-neme kľúč väčší ako koreň, ale menší ako všetci jeho synovia – halda bude zložená z T_k a nového vrcholu – T_0
  - ak teraz spravíme delete min, najmenší prvok je koreň T_k; po ňom ostanú jeho synovia $S_0, S_1 \ldots, S_{k-1}$ a keď ich postupne pospájame s T_0, ostane nám T_k, takže sme sa dostali do pôvodneho stavu, ale alg. spravil $\Theta(k) = \Theta(\sqrt{n})$ operácií, čo je veľa

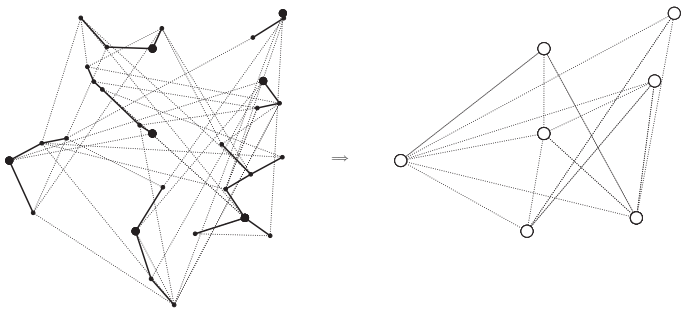
- potrebujeme teda, aby rank bol najviac $O(\log n)$, teda aby strom s rankom k mal exponenciálne veľa vrcholov od k
- nápad #2: ak vrcholu v usekneme syna, poznačíme si to; ak mu usekneme aj druhého syna, odrežeme aj samotné v a pripojíme ho ku koreňom a rekurzívne postupujeme vyššie: ak aj otec v týmto prišiel o druhého syna, odrežeme aj jeho, atď.
- najskôr si ukážme, že zložitosť decrease key bude stále $O(1)$ – presnejšie 4$ na operáciu
- invariant: každý označený vrchol, ktorý prišiel o syna, má nasporené 2$
- pri decrease key 1$ zaplatíme za $O(1)$ operácií (odrezanie, pripojenie do zoznamu koreňov, označenie otca); 1$ mu uložíme na účet (keďže je to nový koreň) a 2$ uložíme na účet otcovi
- ak mal otec predtým všetkých synov, teraz má ušetrené 2$ a všetko je v poriadku

- ak už mu 1 syn chýbal, má našetrené 2$ + teraz prišli ďalšie 2$ – spolu 4$ – z nich 1$ zaplatíme za $O(1)$ operácií, 1$ bude mať na účte ako nový koreň a 2$ pošle svojmu otcovi – atď, celá kaskáda urezávania označených vrcholov sa zaplatí z ušetrených peňazí
- ostáva dokázať, že pri takomto urezávaní bude mať vrchol s rankom k exponenciálne veľa vrcholov v závislosti od k
- lema: nech vrchol x má synov $y_1, \ldots, y_m$, ktorých sme označili v poradí, ako sa prilinkovali ku x; potom $\text{rank}(y_i) \geq i - 2$
- dôkaz: v čase, keď sme pripájali y_i pod x mal x aspoň $i - 1$ detí (a to $y_1, \ldots, y_{i-1}$; možno mal aj ďalšie, ktoré sa medzičasom usekli); my ale spájame iba stromy s rovnakým rankom, tzn. y_i malo vtedy $\geq i - 1$ detí a odvtedy max. 1 zmizlo
- nech F_n je najmenšia možná veľkosť stromu s rankom n; $F_0 = 1$, $F_1 = 2$ a podľa lemy $F_n = 2 + \sum_{i=2}^{n} F_{i-2}$; riešením tejto rekurencie sú (trochu posunuté) Fibonacciho čísla a preto $F_n = \Theta(\phi^n)$, kde $\phi \approx 1.618$



5 Hľadanie najlacnejšej kostry

- Kruskalov algoritmus – $O(m \log n)$ (kvôli triedeniu)
- Jarníkov-Primov algoritmus – $O(n^2)$, resp. $O(m + n \log n)$ s Fibonacciho haldou
- otvorený problém: existuje $O(m)$ algoritmus?
  - je známy randomizovaný $O(m)$ algoritmus
  - je známy deterministický $O(m\alpha(n))$ algoritmus
  - je známy optimálny algoritmus, ale nevie sa jeho zložitosť
- ukážeme si riešenie v $O(m \log^* n)$
- bottleneck pri Jarníkovom algoritme s Fibonacciho haldou je veľkosť haldy: ak do haldy nasúkame $\Omega(n)$ prvkov, výber minima bude trvať $O(\log n)$ a toto sa nedá veľmi zlepšiť
- myšlienka: budeme mať haldu s maximálnou veľkosťou k; začneme vykonávať Jarníkov algoritmus a budovať kostru z jedného vrcholu; akonáhle sa nám halda preplní, zastavíme a začneme budovať kostru z *iného* vrcholu (s prázdnou haldou)
- takto pokračujeme, až kým nie je každý vrchol v nejakom strome; čas: $O(m + n \log k)$
- následne prejdeme celý graf a každý strom (časť kostry) skontrahujeme do jedného vrcholu; ak medzi stromami vedie viacero hrán, necháme len tú najlacnejšiu (čas: $O(m)$)



- malé k znamená menšie $n \log k$, čiže rýchlejšiu 1 fázu, naopak väčšie k znamená, že stromy narastú viac, takže v nasledujúcej fáze bude menej vrcholov a celkovo budeme mať menej fáz
- vhodná voľba je $k = 2^{2m/n}$, pretože vtedy je $n \log k = \Theta(m)$
- na konci fázy má každý strom T aspoň k hrán začínajúcich v T; každá hrana má 2 konce, preto v nasledujúcej fáze bude $m' \geq k \cdot n'/2$, preto $k' = 2^{2m'/n'} \geq 2^k$
- teda $k_0 = 2^{2m/n} = \Omega(1)$, $k_1 \geq 2^{k_0}$, $k_2 \geq 2^{2^{k_0}}$, $k_3 \geq 2^{2^{2^{k_0}}}$, atď.
- celkovo teda bude $\leq \log^* n$ fáz a celkový čas je $O(m \log^* n)$