Fibonačijev hip

Seminarski rad u okviru kursa
Konstrukcija i analiza algoritama 2
Matematički fakultet

Ivan Ristović
Milana Kovacević

januar 2019.

Sažetak

Fibonačijev hip je struktura podataka osmišljena sa ciljem da poboljša vreme potrebno za operacije nad hipovima. Pružaju bolje amortizovano vreme izvršavanja nego većina drugih prioritetnih redova, uključujući binarni i binomni hip. Fibonačijev hip je osmišljen 1984. godine i publikovan 1987. Ime je dobio po Fibonačijevim brojevima, koji se koriste u analizi složenosti operacija. Koristeći Fibonačijev hip, moguće je unaprediti vremena izvršavanja velikog broja poznatih algoritama kao što je Dijsktrin algoritam. Pružamo implementaciju Fibonačijevog hipa u programskom jeziku *Python*, sa interfejsom jednostavnim za upotrebu i testiranje. Takođe u ovom radu testiramo vreme izvršavanja operacije *decrease-key* kako bismo eksperimentalno pokazali konstantno amortizovano vreme izvršavanja ove operacije.

Sadržaj

1 Uvod

Binarni hip (eng. *Binary heap*) [1] je binarno stablo koje zadovoljava uslov da svaki čvor u stablu ima vrednost ključa veću (tj. manju) od oba svoja sina. Takav hip se često naziva *max-hip* (tj. *min-hip*). Jasno je da će se u hipu maksimum (tj. minimum) nalaziti u korenu stabla, što garantuje da je operacija pronalaženja elementa sa maksimalnom (minimalnom) vrednošću konstantne vremenske složenosti. Svaki hip podržava sledeće operacije [1]:

- `find_min` - vraća vrednost minimalnog elementa iz hipa.
- `extract_min` - uklanja minimalni element iz hipa.
- `insert(v)` - unosi novi čvor sa vrednošću ključa v.
- `decrease_key(k,v)` - spušta vrednost ključa k na vrednost v.
- `merge(h)` - unija sa novim hipom h.

Fibonačijev hip [2] je osmišljen 1984. od strane Fredman-a i Tarjan-a sa ciljem da se poboljša vreme izvršavanja Dijkstrinog algoritma za pronalaženje najkraćih puteva. Originalni Dijsktrin algoritam koji koristi binarni hip radi u vremenskoj složenosti $O(|E| \log |V|)$. Korišćenjem Fibonačijevog hipa umesto binarnog hipa, vremensku složenost Dijsktrinog algoritma je moguće poboljšati do $O(|E|+|V| \log |V|)$. Poređenje vremena izvršavanja u odnosu na binarni hip se može videti na sledećoj tabeli:

Operacija	Binarni hip	Fibonačijev hip
`find_min`	$O(1)$	$O(1)$
`extract_min`	$O(\log n)$	$O(\log n)$
`insert(v)`	$O(\log n)$	$O(1)$
`decrease_key(k,v)`	$O(\log n)$	$O(1)$ (amortizovano)
`merge(h)`	$O(n)$	$O(1)$

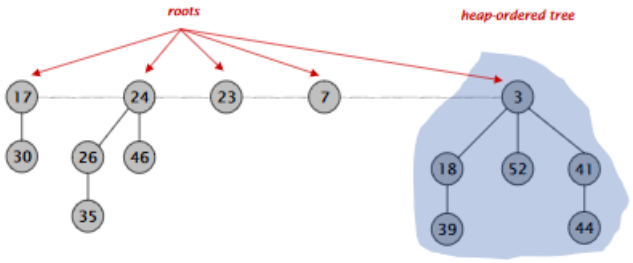
Tabela 1: Poređenje vremena izvršavanja operacija između Fibonačijevog i binarnog hipa.

U nastavku će biti detaljnije opisana struktura hipa (odeljak 2) i operacije nad njim (odeljak 3). U daljem tekstu ćemo pretpostaviti da se radi o min-hipu, analogno važi i za max-hip.

2 Opis strukture

Fibonačijev hip je skup stabala uređenih po uslovu hipa (videti sliku 2.1). Koreni tih hipova se čuvaju u dvostruko povezanoj listi radi lakšeg ubacivanja i izbacivanja. Takođe se čuva pokazivač na najmanji element, kako bi se upit mogao izvršiti u konstantnom vremenu.

[1] Pretpostavlja se da se radi o min-hipu, analogno važi i za max-hip.



Slika 2.1: Vizuelizacija strukture Fibonačijevog hipa.

Dodatno, svaki čvor u sebi sadrži podatak da li je označen ili ne (razlog zašto će biti objašnjen kasnije). Pošto hipovi koji čine skup ne moraju biti binarni, potreban je neki mehanizam *ispravljanja* hipova [3].

Uvodimo sledeće pojmove:

- n - ukupan broj čvorova u Fibonačijevom hipu.
- $rank(x)$ - broj potomaka čvora x.
- $rank(H)$ - maksimalni rank bilo kog čvora u Fibonačijevom hipu H.
- $trees(H)$ - broj hipova u Fibonačijevom hipu H.
- $marks(H)$ - broj označenih čvorova u Fibonačijevom hipu H.

Kako bismo analizirali složenosti operacija, potrebno je da prvo definišemo *potencijal* Fibonačijevog hipa:

Definicija 2.1. *Potencijal* Fibonačijevog hipa H, u oznaci $\Phi(H)$ se definiše kao:

$$\Phi(H) = trees(H) + 2 \cdot marks(H)$$

3 Opis operacija

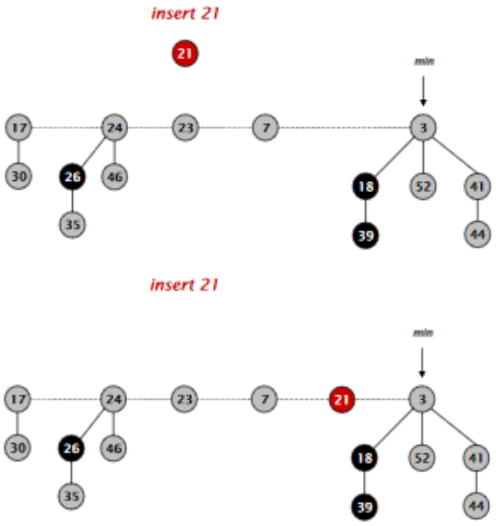
3.1 `find_min`

Pronalaženje vrednosti minimuma se trivijalno odvija jer imamo pokazivač na čvor koji sadrži minimalni element. Složenost ove operacije je, kao i kod običnog binarnog hipa, $O(1)$.

3.2 `insert`

Operacija umetanja novog čvora se odvija u dva dela. Neka je x čvor koji želimo da ubacimo u hip. Prvo se kreira novo stablo sa x kao korenom. To stablo (koje se sastoji samo od čvora x) se ubacuje u listu korena Fibonačijevog hipa. Eventualno je potrebno ažurirati pokazivač na najmanji element ukoliko je ključ čvora x manji od minimuma hipa. Vizuelni prikaz operacije umetanja se može videti na slici 3.1.

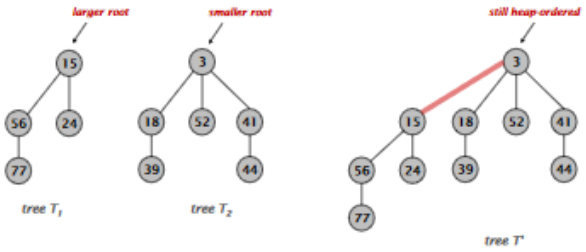
Složenost operacije umetanja je stoga $O(1)$, dok je promena u potencijalu $+1$.



Slika 3.1: Vizuelizacija operacije umetanja.

3.3 `extract_min`

Da bismo implementirali operaciju brisanja minumuma, potrebno je prvo da definišemo operaciju *spajanja* (eng. *linking*) dva hipa. Naime, porede se koreni dva hipa i kao sin manjeg se dodaje veći (videti sliku 3.2).



Slika 3.2: Vizuelizacija spajanja hipova.

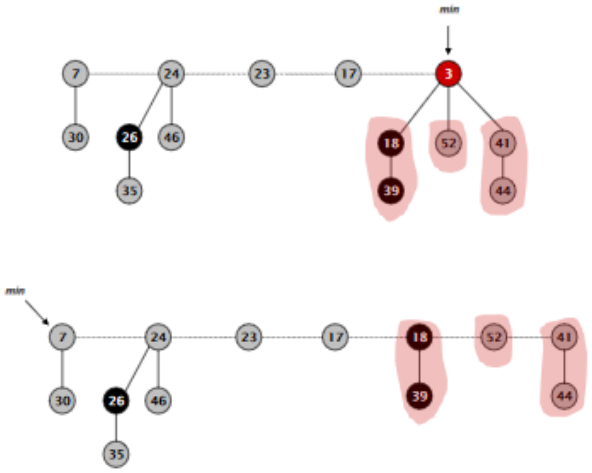
Brisanje se odvija u dva koraka. Prvo se čvor koji je minimalan izbaci iz liste korenova a njegova deca se ubacuju u listu korenova (videti sliku 3.3) i ažurira se minimum hipa. Zatim se radi *konsolidacija* hipa. Naime, potrebno je proći kroz listu korenova i postarati se da nijedna dva korena nemaju isti rank (videti sliku 3.4). Ukoliko dva korena imaju isti rank, njihova stabla se spajaju operacijom spajanja.

Složenost operacije umetanja se analizira tako što se analiziraju operacije od kojih se sačinjena:

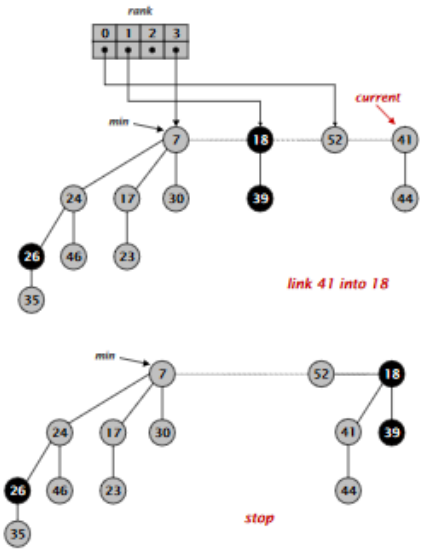
- Ubacivanje sinova minimalnog čvora u listu korena je složenosti $O(rank(H))$.
- Ažuriranje vrednosti minimuma je složenosti $O(rank(H))+O(trees(H))$.
- Konsolidacija hipa je složenosti $O(rank(H)) + O(trees(H))$.

Sve ukupno, $O(rank(H)) + O(trees(H))$.

Promena u potencijalu (označena sa $\delta\Phi(H)$)) je $O(rank(H) - trees(H))$. Ovo proizilazi iz činjenice da je $trees(H') = trees(H) + 1$ jer nijedna dva čvora nemaju isti rank posle konsolidacije. Stoga je $\delta\Phi(H) = rank(H) + 1 - trees(H)$. Iz svega ovoga zaključujemo da je amortizovana cena operacije brisanja minimuma $O(rank(H))$.



Slika 3.3: Vizuelizacija prvog koraka operacije brisanja minimuma.



Slika 3.4: Vizuelizacija drugog koraka operacije brisanja minimuma.

3.4 `decrease_key`

Operacija spuštanja ključa čvora x do vrednosti v (eng. *decrease key*) kao rezultat menja vrednost ključa čvora x u v (pretpostavlja se da je v manje od vrednosti ključa čvora x). S obzirom da ova operacija može da naruši uslov hipa, potrebno je preurediti hip kako bi se uslov održao.

Operacija spuštanja se radi u više koraka. Intuitivno, ukoliko je uslov hipa ispunjen posle spustanja vrednosti ključa čvora x, onda je posao gotov. U protivnom je potrebno *isećи* podstablo čiji je koren čvor x i ubaciti ga u skup hipova. Dodatno, ukoliko jednom čvoru isečemo i drugog sina, sečemo i njega i ubacujemo u korenu listu. Ovo radimo rekurizivno dokle god dolazimo do markiranog čvora ili do korena. Preciznije (videti sliku 3.5):

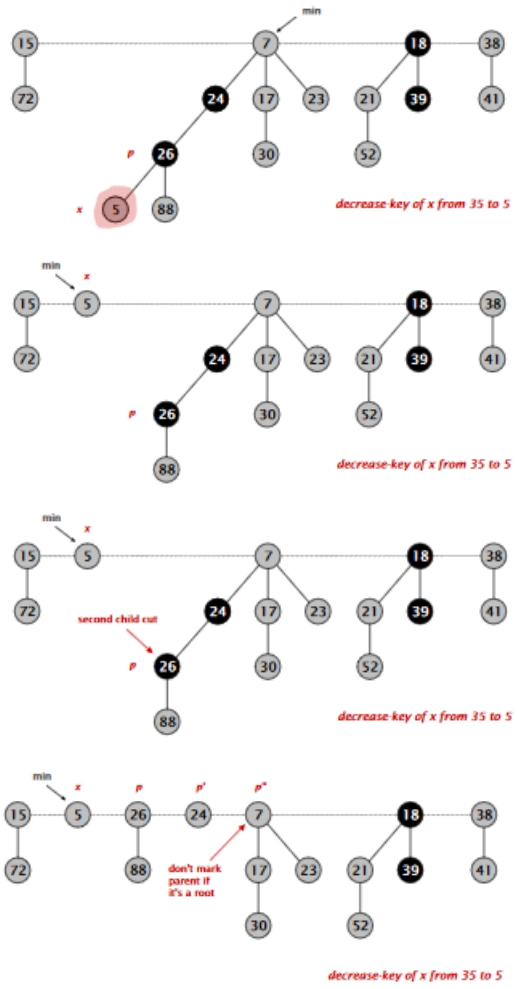
- Spustiti vrednost ključa x.
- Iseći drvo čiji je x koren, dodati u listu korenova i skloniti oznaku (ukoliko je bio označen).
- Ako je roditelj p čvora x neoznačen (nije do sada bio isečen), označiti ga.
- U protivnom, iseći p, dodati ga u listu korenova i skloniti oznaku.
- Rekurzivno ponoviti prethoda dva koraka za sve roditelje kojima su dvaput sečeni sinovi.

Kako bi se osigurala kompleksnost operacije `extract_min` i `decrease_key`, potrebno je balansirati vrednost funkcije potencijala. Ovo se postiže vođenjem računa o markiranju i brisanju čvorova, čime se održava *ispravljenost* hipova. Složenost operacije spuštanja ključa je $O(c)$, naime potrebno je $O(1)$ vreme za izmenu vrednosti ključa i $O(1)$ vreme za svako od c sečenja, uz dodavanje u listu korenova. Promena potencijala je $O(1) - c$, pa je amortizovana složenost $O(1)$.

Operacija brisanja čvora x se može implementirati koristći već opisane operacije: prvo `decrease_key`$(x, -\infty)$ (kako bi se garantovalo da će on onda postati minimum hipa) zatim `extract_min` (kako bi se taj novonastali minimum uklonio).

3.5 `merge`

Unija dva Fibonačijeva hipa se trivijalno izvršava spajanjem njihovih dvostruko povezanih listi korenova. Minimum rezultata je manji od dva minimuma hipova koji se spajaju.



Slika 3.5: Vizuelizacija operacije spuštanja ključa.

4 Veza sa Fibonačijevim brojevima

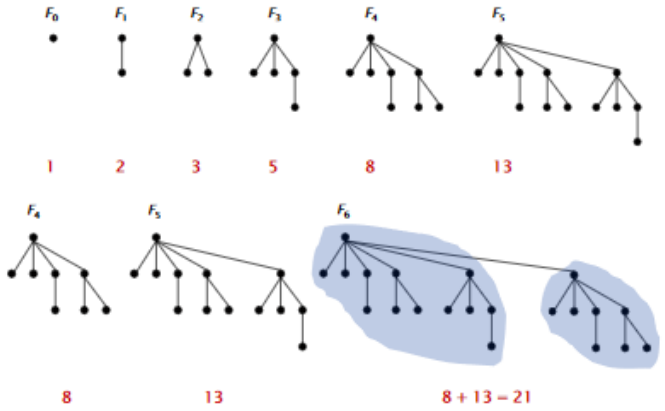
Lema 4.1. *Neka je x čvor i neka su $y_1, \dots y_k$ njegova deca po redosledu dodavanja u hip. Tada je:*

$$rank(y_i) \geq \begin{cases} 0, & i = 1 \\ i - 2, & i \geq 1 \end{cases}$$

A tree of order n has exactly n children. Trees of order n are formed by taking two trees of order n - 1 and making one the child of another. If a tree loses two children, that tree is cut away from its parent.

Označimo sa F_k najmanje moguće stablo ranka k koje zadovoljava uslov iznad. Ovakvo stablo je moguće konstruisati ranije opisanim operacijama Fubonačijevog hipa. Tokom izvršavanja operacije `extract_min`, stablo ranka n dobijamo povezivanjem dva stabla ranka $n - 1$. U suprotnom, u operaciji `decrease_min`, ukoliko jedan čvor izgubi dva deteta, i ono

samo je odsečeno od svog roditelja. Ovim osiguravamo da broj čvorova odgovara Fibonačijevim brojevima. Postoji veza - broj čvorova u F_k je baš k-ti Fibonačijev broj:



S obzirom da je $F_k \geq \phi^k$, gde je $\phi = (1 + \sqrt{5})/2$, dobijamo da je:

$$rank(H) \leq \log_\phi n$$

Iz ovoga zaključujemo da je kompleksnost operacije `extract_min` $O(\log n)$.

5 Poređenje efikasnosti

Kako bismo ispitali efikasnost naše implementacije Fibonačijevog hipa, implementirali smo i standardan binarni hip. U nastavku su prikazani rezultati poređenja efikasnosti ova dva hipa.

Poređenje efikasnosti operacija `decrease_key` i `extract_min` vršimo na sledeći način:

- Kreiramo Fibonačijev i binarni hip.
- U oba hipa ubacimo n random generisanih vrednosti.
- N puta izvršimo operaciju na oba hipa i merimo njihovo ukupno vreme izvršavanja.

n	Binarni hip	Fibonačijev hip
100	0.003965616226196289s	0.02899909019470215s
1000	0.05807352066040039s	0.2644772529602051s
10000	0.6527743339538574s	3.014829635620117s

Tabela 2: `extract_min` benchmark

n	Binarni hip	Fibonačijev hip
100	0.0010502338409423828s	0.0010502338409423828s
1000	0.01744818687438965s	0.02191758155822754s
10000	0.2323460578918457s	0.293259859085083s

Tabela 3: `decrease_key` benchmark

Dobijenim rezultatima nismo uspeli da dokazemo efikasnost ovih operacija nad Fibonačijevim hipom. Međutim, smatramo da su razlog za to neefikasne strukture podataka korišćenje pri implementaciji u programskom jeziku *Python*. Nastavak ovog rada bi mogao da obuhvata i implementaciju u nekom drugom programskom jeziku kao i analizu novodobijenih rezultata.

Literatura

[1] Victor Adamchik. Heaps. on-line at: https://www.cs.cmu.edu/~adamchik/15-121/lectures/Binary%20Heaps/heaps.html.

[2] Michael L. Fredman, Robert Sedgewick, Daniel D. Sleator, and Robert E. Tarjan. The Pairing Heap: A new form of self-adjusting heap, 1986. on-line at: http://www.cs.cmu.edu/~sleator/papers/pairing-heaps.pdf.

[3] Princeton. Fibonacci Heaps. on-line at: https://www.cs.princeton.edu/~wayne/teaching/fibonacci-heap.pdf.