Implementacija Fibonacci Heap-a

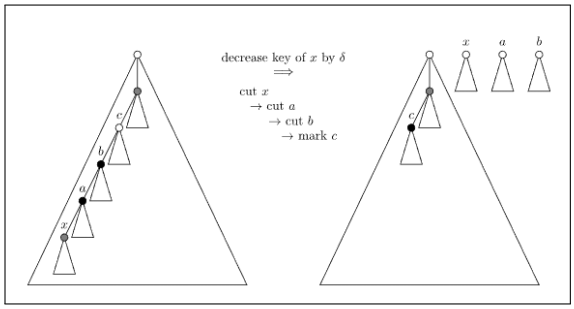
Irhad Fejzić, *TKN, Odsjek za matematiku, PMF Sarajevo*

***Abstract*—U radu je definisan i implementiran Fibonacci Heap koristeći standardne C++ biblioteke. Fibbonaci heap je stuktura podataka sastavljena od kolekcije *heap-ordered* stabala i korištenjem implementacije prioritetnog reda, zajedno sa Djikstrinim algoritmom, daje veoma efikasno i optimalno vrijeme kompleksnosti za glavne operacije unosa, brisanja i ubacivanja.**

***Ključne riječi*—Fibbonaci heap, C++, glavne operacije, prioritetni red**

I. Uvod

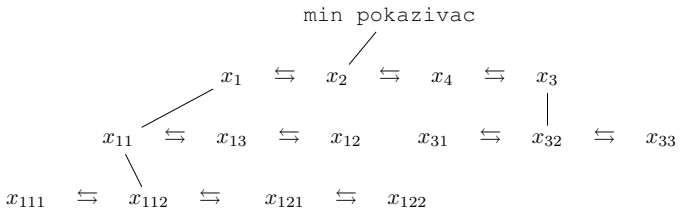
Fibonacci heap je klasična stuktura podataka koja podržava brisanje u logaritamskom vremenu i ostale heap operacije u *O(1)* amortiziranom vremenu. Heap je struktura podataka sačinjena od podataka sa ključevima gdje je ključ korijena najveća ili najmanja vrijednost od svih ostalih čvorova unutar stabla (ovisno da li se radi o min ili max heap-u). Fredman i Tarjan [1] su izumili Fibonacci heap, implementacija heap-a koja podržava brisanje-min (*Extract Min*) i brisanje n-tog elementa u vremenu *O(logn)*, kao i ostale operacije u konstantnom vremenu. Fibonacci heap se koristi u implementaciji bržeg Djikstra algoritma za pronalazak najkraćeg puta i brzih algoritama za neusmjerena i usmjerena minimalna razapinjuća stabla. Bitne osobine Fibonacci heap-a za svaki čvor ove strukture su (i) da svaki čvor sadrži polje roditelja i dijeteta, (ii) svako dijete nekog čvora x je povezan (sa dvostruko povezanom listom) između sebe radi bržeg brisanja i unosa (ovo je omogućeno sa varijablama lijevo i desno dijete) i (iii) svako dijete sadrži polje stepena i oznake. Oznaka označava da li je čvor x izgubio dijete od kad je postao dijete nekog drugog čvora. Standardna metoda u binarnom heap-u za popravljanje strukture heap-a je metoda *bubbling*, tj. poređenjem ključa sa njegovim roditeljem i ukoliko je roditelj veći dolazi do zamijene čvorova. Ovaj proces iako jednostavan može dovesti do izmjene čvorova od kraja do samog početka, tj. korijena stabla što može iznositi **O(logn)** vremena u najgorem slučaju. Dolaskom ideje da prilikom umanjenja ključa x algoritam može ukloniti čvor od roditelja i dodati uklonjeno podstablo u listu stabala. U tom trenutku heap se neće sastojati od binomnih stabala već od ostatka nakon uklanjanja čvorova. Sa informacijom stepena (brojem dijece nekog čvora) moguće je povezati stabla sa istim stepenom. Dodatno ubrzanje algoritma dolazi držanjem oznake za roditelja kod kojeg se eliminiše dijete. Ukoliko izbacimo 2 čvora tada se i roditelj izbacuje i to podstablo se dodaje u listu stabala. Na slici 1 prikazuje se proces rezanja crnih čvorova kod kojih se eliminiše dijete. Umanjivanjem ključa x dolazi do izbacivanja x što uz prethodnu tvrdnju dovodi i do rezanja čvorova a i b. Ovo dovodi do kaskadnog rezanja ukoliko su roditelji već označeni u prethodnim umanjivanjem ključeva. Brisanje min čvora iz Fibonacci heap-a dovodi do operacije koja zahtjeva logaritamsko vrijeme dok se korištenjem prethodno navede metode osigura da se ostale operacije mogu izvršiti u amortizovanom konstantnom vremenu.



Slika 1: Metoda umanjivanja ključa

II. Opis projekta

Projekat je urađen u C++ programskom jeziku sa standardnim bibliotekama koje nudi jezik. Kreiranje novog Fibonacci heap-a nastaje instanciranjem klase FibonacciHeap koja u tom momentu postavlja vrijednosti veličine stabla na 0 i min čvora na *NULL*. Operacije koje su dodane u implementaciji heap-a su unos, brisanje, povezivanje, izvlačivanje min čvora i print koje su u sljedećem dijelu sekcije detaljnije opisane. Operacije se izvršavaju u amortizovanom konstantnom vremenu osim operacije brisanja i operacije *pop*, tj. eliminacije najmanjeg ključa. Amortizovano vrijeme kompleksnosti označava da algoritam u najgorem slučaju, tj. rijetkim trenucima, izvršava operaciju sporije od navedenog, ali kroz prosječno vrijeme (uzimajući u obzir ogroman broj izvršenih operacija) vrijeme komplesnosti će zadovoljavati konstantno izvršenje [2]. Fibonacci heap sadrži čvorove koji se nalaze u dvostruko povezanoj listi, kao što su i dijeca čvorova dio dvostruko povezane liste. Ovo je moguće zamijeniti vektorom ili nizom, ali bitno je da ne dođe do usporenja i kršenja vremenske kompleksnosti heap-a. Na slici 2 prikazana je generalna struktura gdje pokazivač pokazuje na trenutni minimalni element u dvostruko povezanoj listi, dok su čvorovi heap-a unutar te liste ili sadrže svoju listu.



Slika 2: Struktura Fibonacci heap-a

A. Unos čvora

Metoda unosa čvora provjerava da li već postoji element unutar heap-a. Ako je Fibonacci heap prazan postavi novi čvor kao min čvor i vrati ga. U suprotnom, algoritam provjerava da li je novi čvor manji od trenutnog MIN i zamijeni ukoliko jeste. Pseudokod 1 objašnjava početnu inicijalizaciju čvora i postavljanje čvora unutar liste korijena ukoliko je MIN prisutan.

Algorithm 1 Unos čvora u Fibonacci heap

Require: n
$newNode->x \leftarrow n$
$newNode->degree \leftarrow 0$
$newNode->left \leftarrow newNode$
$newNode->right \leftarrow newNode$
$newNode->mark \leftarrow False$
$newNode->parent \leftarrow NULL$
$newNode->child \leftarrow NULL$
if MIN is null **then**
    $MIN \leftarrow newNode$
else
    Dodaj čvor *newNode* kao lijevo dijete MIN čvora i uradi update ako je manji
end if
$size \leftarrow size + 1$
return newNode

B. Unija dva Fibonacci heap-a

Neka su x i y korijeni dva stabla koji se trebaju povezati. Ukoliko je ključ (vrijednost) od x veća od y, tj $key(x) >= key(y)$ tada x postaje dijete od y u suprotnom se postavlja da y bude dijete od x. Operacija se izvodi u O(1) vremenu. U pseudokodu 2 dolazi do grupisanja liste korijena h1 i h2 u novu listu korijena H, postavljanje novog minimuma zavisno od veličine ključa i ažuriranja novog broja čvorova heap-a. Ove operacije se izvršavaju u konstantnom vremenu i ne dolazi do prilagođavanja stabala koji se dodaju u novi heap H.

Algorithm 2 Unija dva heap-a

Require: $h1, h2$
$H \leftarrow FibonacciHeap$
$min[H] \leftarrow min[h1]$
spoji listu korijena h2 sa listom korijena H
if $min[h1]$ is null
    OR ($min[h2]$ IS NULL AND $min[h2] < min[h1]$)
then
    $min[H] \leftarrow min[h2]$
end if
$size[H] \leftarrow size[h1] + size[h2]$
return H

C. Umanjivanje ključa

Pošto se radi o heap stablu jedna on najbitnijih metoda koje Fibonacci heap treba da izvrši efikasno je umanjenje ključa, tj. promijena vrijednosti ključa na neku manju vrijednost. Metoda se izvršava odabirom čvora X kojem se ključ mijenja i poređenjem njegove nove vrijednosti sa njegovim roditeljem. Ukoliko je vrijednost roditelja veća od vrijednosti X tada se pozivanju metode rezanja i kaskadnog rezanja stabla. Proces je sličan onome opisanom u sekciji uvoda gdje rezanje izbacuje čvor X sa trenutne pozicije i dodaje ga u listu korijena, tj. postaje povezan sa korijenom. Ukoliko je prethodno bio označen sada više nije. Metoda kaskadnog rezanja je metoda koja se izvršava ukoliko su roditelji označeni nakon izbacivanja čvora X. U ovom trenutku roditelj X se označava ukoliko prethodno nije bio i izbacuje se iz trenutne pozicije i dodaje u listu korijena kao i X. Ovo se izvršava i za njegovog roditelja i zaustavlja se kad dođe do neoznačenog roditelja. Nakon povezivanja sa korijenom provjerava se najmanji čvor koji je dodan u listu i označava se kao *MIN*. Pseudokod 3 objašnjava proces umanjivanja ključa pozivanjem metoda cut i cascade koje izvršavaju prethodno opisani proces. Razlog vršenja kaskadnog rezanja je zbog osiguranja da će stepen ukupne veličine heap-a biti dovoljno mala. Ukoliko se ostave roditelji bez njihovog rezanja tada stepen može postati linearan što dovodi do linearnog vremena brisanja minimalnog čvora u Fibonacci heap-u.

Algorithm 3 Smanjivanje ključa

Require: *Node x, vrijednost k*
if $k > x.key$ **then**
    return
end if
$x.key \leftarrow k$
$y \leftarrow x.parent$
if y != NULL and x.key < y.key **then**
    cut(x,y)
    cascade(y)
end if
if $x.key < min.key$ **then**
    $MIN \leftarrow x$
end if

Izbacivanje minimuma Jedna od najvažnijih operacija u Fibonacci heap strukturi podataka je izbacivanje trenutačnog minimalnog ključa i postavljanje novog minimuma. U ovoj operaciji nakon eliminacije čvora sa najmanjim ključem dolazi do raspoređivanja heap-a. Prvobitno dolazi do brisanja minimalnog čvora i prebacivanja pokazivača na sljedeći čvor u listi korijena. Pošto je potrebno prilagoditi stabla unutar heapa kreira se niz dužine maksimalnog stepena u stablu i kroz iteraciju vrši unija stabala koji su istog stepena. Na početku dolazi do mapiranja trenutnog korijena (min čvora) sa stepenom unutar niza i pređe na sljedeći čvor korijena. Ukoliko se tokom mapiranja naiđe na dva čvora sa istim stepenom dolazi do unije kako bi se zadržala osobina minimalnog heapa. Ovaj proces se ponavlja dok se ne grupišu svi korijeni istog stepena. Pseudokod 4 koristi metodu *consolidate* koja definiše proces unije i prilagođavanja stabala sa istim brojem stepena. Metoda prilagođavanja ili *consolidate* (opisana u pseudokodu 5 prolazi kroz trenutne čvorove u listi korijena i posmatra one

koji su istog stepena korištenjem novokreiranog niza čija se dužina dobija omjerom logaritma veličine heap-a i logaritma 2.

Algorithm 4 Izbacivanje najmanjeg ključa

Require: *Min MIN*
  if $x = NULL$ **then**
    return
  end if
  for `svako dijete X od MIN` **do**
    `dodaj X u listu korijena H`
    $array[X] \leftarrow NULL$
    `izbaci MIN iz liste korijena H`
    if $MIN = MIN->desniCvor$ **then**
      $min \leftarrow NULL$
    else
      $min \leftarrow MIN->desniCvor$
      $consolidate()$
    end if
    $state--$
  end for
  return MIN

Algorithm 5 Prilagođavanje (Consolidate)

Require: *Min MIN*
  for `i od najvećeg stepena(veličine[H])` **do**
    $A[i] \leftarrow NULL$
  end for
  for `svako dijete D u listi korijena` **do**
    $x \leftarrow D$
    $stepen \leftarrow stepen[x]$
    while A[D] is not NULL **do**
      $y \leftarrow A[D]$
      if $kljuc[x] > kljuc[y]$ **then**
        `zamijeni x i y`
      end if
      `FibonacciLink(y,x)`
      $A[D] \leftarrow NULL$
      $D \leftarrow D + 1$
    end while
    $A[D] \leftarrow x$
  end for
  $min \leftarrow$ Nađi novi minimum prolazom kroz liste korijena

III. Zaključak

U ovom projektu opisana je implementacija Fibonacci heap-a sa glavnim metodama unosa, unije, brisanja i umanjivanja ključa. Fibonacci heap je struktura podataka koja podržava iste operacije kao binarni heap sa boljim asimptotskim vremenom (konstantni unos čvora u poređenju sa *O(logn)*). Ovakve strukture podataka korisne su za implementaciju prioritetnog reda i algoritme koji koriste ove redove poput Djikstrinog algoritma (smanjuje se vrijeme kompleksnosti od *O((|E|+|V|)log|V|)* na *O(|E|+|V|log|V|)*), Primmovog algoritma i mnogih drugih kojim je operacija unosa ili umanjenja ključa potrebna. Bitnost Fibonacci heap-a je što drži minimalni element/ključ kao korijen jednog stabla unutar heap-a ali za razliku od binomnog heap-a koristi "lijeniju" verziju odrađivanja operacija poput prilagođavanja tek kad dolazi do brisanja minimalnog elementa iz stabla. Fibonacci heap ima svoje nedostatke, teže je programiranje ovakve strukture podataka da radi sa svakim algoritmima i teoretski njihovo poboljšano vrijeme kompleknosti nije uvijek u praksi najbolji od inferiornijih tipova heap-a. Specifično, imaju reputaciju da su veoma spori zbog velikog korištenja memorije po čvoru (pamćenjem stepena, oznake, vrijednosti, dijece i roditelja) gdje su se slabiji heap-ovi poput *array-based heap-a* i *pairing heap-a* pokazali efikasnijim u pojedinim slučajevima.

Reference

[1] M. Fredman and R. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *25th Annual Symposium onFoundations of Computer Science, 1984.*
[2] T. H. Cormen, *21*. MIT Press, 2022.