Efikasne implementacije prioritetnog reda

Žanić, Mislav

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO–MATEMATIČKI FAKULTET
MATEMATIČKI ODSJEK

Mislav Žanić

EFIKASNE IMPLEMENTACIJE PRIORITETNOG REDA

Diplomski rad

Voditelj rada:
doc. dr. sc. Goranka Nogo

Zagreb, lipanj, 2022.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Sadržaj **iii**

Uvod **1**

1 Uvodne definicije **2**

2 Fibonaccijeva Hrpa **4**
2.1 Osnovna svojstva 4
2.2 Operacije prioritetnog reda 5
2.3 Analiza složenosti 8

3 Brodalov Red **14**
3.1 Osnovna svojstva 14
3.2 Vodilja 17
3.3 Operacije u stablu 19
3.4 Operacije prioritetnog reda 23
3.5 Analiza složenosti 25
3.6 Detalji implementacije 26

4 Analiza performansi **28**
4.1 Sortiranje 28

Bibliografija **31**

Uvod

Prioritetni red apstraktna je struktura podataka korištena u implementaciji mnogih algoritama, npr. u algoritmima sortiranja i algoritmima pronalaska najkraćeg puta. Osim implementacija, zanimljivo je promatrati asimptotski efikasne prioritetne redove. U ovom radu prezentirat ćemo dvije asimptotski efikasne implementacije prioritetnog reda, Fibonaccijevu hrpu i Brodalov red.

U prvom poglavlju uvodimo definiciju prioritetnog reda i oznake potrebne u ostalim poglavljima.

U drugom poglavlju opisujemo Fibonaccijevu hrpu. Prvo dajemo intuitivan opis strukture i implementacije operacija prioritetnog reda. Za kompleksnije operacije dajemo pseudokod uz opis. Nakon toga opisujemo metodu amortizacijske analize složenosti i dajemo motivaciju za takvu analizu složenosti. Također, dokazujemo neke tvrdnje potrebne za analizu složenosti operacija Fibonaccijeve hrpe. Poglavlje završavamo komentarom na performanse Fibonaccijeve hrpe.

U trećem poglavlju opisujemo drugu implementaciju prioritetnog reda koju je dao Gerth Stølting Brodal 1996. godine u [1]. Prvo dajemo intuitivan opis strukture. Uz intuitivan opis, dajemo i formalan opis strukture pomoću invarijanti i dokazujemo dvije tvrdnje vezane uz čvorove Brodalovog reda. Zatim opisujemo pomoćnu strukturu podataka *vodilja*, kojom održavamo neke od invarijanti Brodalovog reda. Nakon toga opisujemo jednostavne operacije Brodalovog reda pomoću kojih gradimo implementaciju kompleksnih operacija i održavamo strogu strukturu strukture podataka. Na kraju dajemo detaljan opis implementacije operacija prioritetnog reda iz kojeg direktno slijede pripadne složenosti. Uz opis implementacije dajemo i pseudokod kompleksnih operacija kao i neke napomene vezane uz implementaciju.

Za kraj, radimo jednostavnu usporedbu performansi Fibonaccijeve hrpe i Brodalovog reda pomoću sortiranja n nasumice generiranih brojeva i pronalaska najkraćeg puta u matrici.

Poglavlje 1

Uvodne definicije

U ovom poglavlju uvodimo definicije i oznake korištene kroz ovaj rad.

Definicija 1.0.1. ***Stablo*** *je apstraktna struktura podataka čiji podatci čine hijerarhiju. Elemente stabla nazivamo* ***čvorovima****. Čvor može imati vrijednost obično reprezentiranu elementom nekog totalno uređenog skupa, te roditelja, djecu i braću koji su također čvorovi. Čvor bez roditelja nazivamo* ***korijen****. Čvor bez djece nazivamo* ***list****. Svako je stablo reprezentirano pokazivačem na neki korijen.*

U ovom ćemo radu poistovjećivati čvorove s njihovim vrijednostima. Dakle za čvorove x i y, s $x < y$ zapravo kažemo da x ima manju vrijednost od y.

Definicija 1.0.2. *Neka je T stablo s $n \in \mathbb{N}$ čvorova i neka je x jedan čvor stabla T. Definiramo atribute čvora x.*

- ***Stupanj*** *čvora x definiramo kao broj djece čvora x, a označavamo ga s $d(x)$. S $D(n)$ označavamo najveći stupanj u stablu T.*
- ***Rang*** *čvora x definiramo kao $r(x) = \max\{r(y) \mid y \text{ je dijete čvora } x\}$.*
- *S $n_i(x)$ označavamo broj djece čvora x ranga $i = 0, \ldots, r(x) - 1$.*
- *S $p(x)$ označavamo roditelja čvora x.*

Definicija 1.0.3. *Neka je T stablo. Kažemo da T poštuje* ***svojstvo hrpe*** *ako za sve čvorove x vrijedi*

$$p(x) < x. \tag{1.1}$$

Definicija 1.0.4. ***Prioritetan red*** *je apstraktna struktura podataka kojom se održava skup podataka, od kojih svaki ima pridruženu vrijednost koju nazivamo prioritet. Prioritetan red podržava sljedeće operacije:*

- INSERT(S, x) *dodaje element x skupu S.*
- MIN(S, x) *vraća element najmanjeg prioriteta iz S.*
- EXTRACTMIN(S, x) *briše element najmanjeg prioriteta iz S.*
- DECREASEKEY(S, x, k) *smanjuje vrijednost prioriteta x na k, pretpostavlja se da je k manji od trenutnog prioriteta elementa x.*

Poglavlje 2

Fibonaccijeva Hrpa

2.1 Osnovna svojstva

U onom poglavlju opisujemo Fibonaccijevu hrpu i njena osnovna svojstva, te dajemo intuitivan opis implementacije.

Definicija 2.1.1. ***Fibonaccijeva hrpa*** *je kolekcija stabla koja poštuju svojstvo hrpe.*

Stabla Fibonaccijeve hrpe povezana su cirkularnom dvostruko vezanom listom koju nazivamo *lista korijena*. Cirkularne dvostruko vezane liste pridonose na dva načina ovoj strukturi podataka:

1. Dodavanje novog čvora u listu kao i micanje čvora iz liste imaju vremensku složenost $O(1)$.

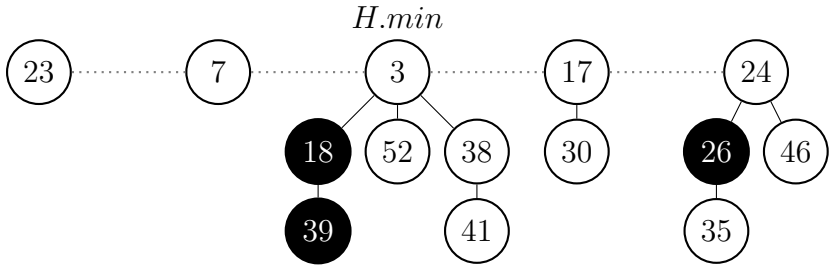
2. Spajanje dvije liste ima vremensku složenost $O(1)$.

Fibonaccijeva hrpa održava pokazivač na najmanji korijen u listi korijena.
Za čvorove Fibonaccijeve hrpe definiramo sljedeći atribut.

Definicija 2.1.2. *Neka je x čvor neke Fibonaccijeve hrpe.* ***Oznaka*** *čvora je atribut kojim označavamo da je čvor izgubio dijete.*

Možemo primijetiti da definicija Fibonaccijeve hrpe ne uvodi nikakva ograničenja na strukturu. Vidjet ćemo da struktura Fibonaccijeve hrpe proizlazi implicitno iz označenih čvorova, te da nalikuje strukturi **Binomne hrpe**.

Slika 2.1: Primjer Fibonaccijeve hrpe.



2.2 Operacije prioritetnog reda

U ovom poglavlju dajemo opis operacija Fibonaccijeve hrpe.

Posebno promatramo DeleteMin i DecreaseKey zato što te operacije implicitno održavaju strukturu hrpe (DeleteMin smanji listu korijena i učini stabla više nalik na stabla u binomnoj hrpi, a DecreaseKey povećava listu korijena što ima utjecaja na amortizacijsku analizu koju provodimo u 2.3). Za ostale operacije dajemo kratak opis zato što njihova implementacija jasno slijedi iz konstrukcije.

2.2.1 DeleteMin

Operacija DeleteMin najkompliciranija je operacija u Fibonaccijevoj hrpi. Možemo ju podijeliti u 3 koraka:

1. Svu djecu čvora $H.min$ prebacujemo u listu korijena. Nakon toga brišemo $H.min$ iz liste.

2. *Konsolidiramo* listu korijena. Svaka dva stabla istog stupnja spajamo u stablo većeg stupnja tako da veći od dva čvora dodamo kao dijete drugog čvora. Ovo ponavljamo dokle god postoje dva korijena istog stupnja.

3. Tražimo novi minimum po listi korijena.

Na prvi nam se pogled može učiniti da je konsolidiranje vremenske složenosti $O(|W|^2)$, gdje je $|W|$ duljina liste korijena. Vidjet ćemo da je konsolidiranje zapravo linearne složenosti.

Algorithm 1 DeleteMin(H)

```
z ← H.min
if z ≠ NULL then
  for x ∈ z.children do
    H.korijeni.append(x)
    x.p ← NULL
  H.korijeni.remove(z)
  if z = z.right then
    H.min ← NULL
  else
    Consolidate(H)
  H.n ← H.n − 1
```

Algorithm 2 Consolidate(H)

```
A[0, ..., D(H.n)] novi niz
for i = 0 ... D(H.n) do
  A[i] = NULL
for w ∈ H.korijeni do
  x ← w
  d ← x.degree
  while A[d] ≠ NULL do
    y ← A[d]
    if x.val > y.val then
      swap(x, y)
    Link(H,y,x)
    A[d] ← NULL
    d ← d + 1
  A[d] ← x
H.min ← NULL
for i = 0 ... D(H.n) do
  if A[i] ≠ NULL then
    if H.min = NULL then
      H.korijeni.init(A[i])
      H.min = A[i]
    else
      H.korijeni.append(A[i])
      if A[i].val < H.min.val then
        H.min ← A[i]
```

2.2.2 DecreaseKey

Operacija DecreaseKey je naizgled kompliciranа, ali je ustvari sasvim jednostavna. Možemo je podijeliti u dva dijela.

1. Smanjujemo vrijednost čvora x. Ako nova vrijednost ne krši svojstvo hrpe, gotovi smo.
2. Ako čvor x krši svojstvo hrpe, režemo ga i dodajemo u listu korijena i brišemo mu oznaku. Ako je roditelj čvora x bio označen, onda postupak rezanja ponavljamo. To ponavljanje zovemo *kaskadno rezanje*. Rezanje zaustavljamo ako smo došli do čvora koji nije bio označen ili do korijena.

Algorithm 3 DecreaseKey(H)

```
if k > x.val then
  ERROR
x.val ← k
y ← x.p
if y ≠ NULL ∧ x.val < y.val then
  Cut(H,x,y)
  CascadingCut(H,y)
if x.val < H.min.val then
  H.min ← x
```

Algorithm 4 Cut(H,x,y)

```
y.djeca.REMOVE(x)
H.korijeni.APPEND(x)
x.p ← NULL
x.mark ← FALSE
```

Algorithm 5 CascadingCut(H,y)

```
z ← y.p
if z ≠ NULL then
  if y.mark = FALSE then
    y.mark ← TRUE
  else
    Cut(H,y,z)
    CascadingCut(H,z)
```

2.2.3 Ostale operacije

Ove operacije navodimo zajedno zato što je implementacija trivijalna.

- FindMin vraća čvor najmanje vrijednosti. Ovo je trivijalna operacija zato što Fibonaccijeva hrpa održava pokazivač na minimum.
- Meld spaja dvije cirkularne dvosturko vezane liste i ažurira minimum. Ovo je trivijalna operacija zbog cirkularnih dvostruko vezanih listi.
- Insert je poseban slučaj operacije Meld.
- Delete implementiramo pomoću DecreaseKey i DeleteMin.

2.3 Analiza složenosti

U ovom se poglavlju bavimo analizom složenosti operacija Fibonaccijeve hrpe, dokazujemo tvrdnje potrebne za analizu i objašnjavamo metodu kojom vršimo analizu složenosti.

2.3.1 Amortizacijska analiza

Amortizacijska analiza jedna je od metoda analiziranja složenosti algoritama koja promatra prosječno vrijeme izvršavanja algoritama. Ovu metodu analize koristimo kada je analiza složenosti u najgorem slučaju prepesimistična. Potrebu za amortizacijskom analizom možemo vidjeti na primjeru tzv. *dinamičnih nizova*.

Primjer 2.3.1. *Dinamični niz je struktura podataka nalik na listu koja dopušta dodavanje proizvoljno velikog broja elemenata. Neka je A neki dinamični niz duljine $n \in \mathbb{N}$ s n elemenata. Dodavanje novog elementa u dinamički niz A izaziva kopiranje cijelog niza u novi niz duljine $2 * |A|$.*

Složenost dodavanja novog elementa u dinamični niz duljine $n \in \mathbb{N}$ u najgorem slučaju iznosi $O(n)$, dok prosječno vrijeme dodavanja novog elementa u niz iznosi $O(1)$. Razlika u složenostima dolazi iz činjenice da rijetko moramo kopirati cijeli niz.

Dalje opisujemo tehniku koju ćemo koristiti u amortizacijskoj analizi.

2.3.2 Metoda potencijala

Postoji više tehnika koje se koriste u amortizacijskoj analizi složenosti. Mi ćemo koristiti tzv. *metodu potencijala*.

Neka je D_0 nekakva struktura podataka na koju ćemo primijeniti n operacija. Neka je c_i trošak izvršavanja i-te operacije na strukturi podataka D_{i-1} i neka je D_i struktura podataka dobivena nakon izvršavanja operacije. *Funkcija potencijala* Φ pridružuje strukturi podataka D_i realni broj $\Phi(D_i)$ koji zovemo *potencijal* strukture podataka D_i. *Amortizirani trošak* $\overline{c_i}$ i-te operacije definiramo kao

$$\overline{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1}). \tag{2.1}$$

Primijetimo da iz (2.1) slijedi

$$\sum_{i=1}^{n} \overline{c_i} = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^{n} c_i + \Phi D_n - \Phi(D_0). \tag{2.2}$$

Ako definiramo potencijalnu funkciju Φ tako da vrijedi $\Phi(D_n) \geq \Phi(D_0)$, onda nam suma $\sum_{i=1}^{n} \overline{c_i}$ daje gornju ogradu za pravi trošak $\sum_{i=1}^{n} c_i$. Zbog toga zahtijevamo da vrijedi $\Phi(D_i) \geq \Phi(D_0)$ za sve i zato što ne znamo unaprijed koliko operacija ćemo morati izvesti.
Intuiciju dajemo primjerom s dinamičkim nizovima.

Primjer 2.3.2. *Neka je A neki dinamički niz, n broj elemenata u tom nizu i m broj alociranih mjesta. Trošak ubacivanja elementa u niz je $n+1$ ako $n = m$. Inače je 1. Definiramo funkciju potencijala Φ:*

$$\Phi(D_n) = \frac{2n^2}{m}. \tag{2.3}$$

Pretpostavimo da vrijedi $m \neq n$. Tada vrijedi

$$\overline{c_n} = c_n + \frac{2}{m}(n^2 - (n-1)^2) = 1 + \frac{2(n-1)}{m} \in O(1).$$

Pretpostavimo sada da vrijedi $m = n$. Tada

$$\overline{c_n} = c_n + m - 2m = m - m \in O(1).$$

2.3.3 Stupanj stabla

U ovom potpoglavlju pokazujemo nekoliko tvrdnji o gornjoj ogradi za najveći stupanj u Fibonaccijevoj hrpi H. Gornja nam je ograda potrebna kako bismo pokazali amortiziranu složenost operacije DeleteMin. U narednim dokazima koristiti ćemo konstantu $\phi = \frac{1+\sqrt{5}}{2}$ (tzv. *zlatni rez*). Inače, Fibonaccijeva je hrpa dobila naziv koji ima zbog korištenja zlatnog reza i Fibonaccijevih brojeva u analizi složenosti.

Lema 2.3.3. *Neka je x čvor u Fibonaccijevoj hrpi, neka je k stupanj od x. Neka su $y_0, \ldots, y_{k-1}$ djeca od x poredana po redoslijedu ubacivanja pod x. Tada vrijedi $d(y_i) \geq \max\{0, i-1\}$, za $i = 0, \ldots, k-1$.*

Dokaz. $d(y_0) \geq 0$ vrijedi trivijalno. Za $i \geq 1$, možemo primijetiti da su $y_0, \ldots, y_{i-1}$ bili djeca čvora x prije nego što je y_i postao dijete čvora x. Zbog toga je sigurno vrijedilo $d(x) \geq i$.

Također, primijetimo da spajamo isključivo čvorove istog stupnja, pa je onda sigurno vrijedilo i $y_i \geq i$. Znamo da je y_i izgubio najviše jedno dijete otkako je ono dijete čvora x (operacija CascadingCut bi odrezala y_i da je izgubio dvoje ili više djece).
Slijedi $d(y_i) \geq i - 1$. □

Lema 2.3.4. $\forall k \in \mathbb{N}, F_{k+2} = 1 + \sum_{i=0}^{k} F_i$.

Dokaz. Tvrdnju dokazujemo indukcijom.

Baza: $k = 0$. Vrijedi $F_2 = 1 = 1 + 0 = 1 + F_0$.
Pretpostavka: $\exists k \in \mathbb{N}$ takav da $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$.
Korak: $F_{k+3} = F_{k+2} + F_{k+1} = 1 + \sum_{i=0}^{k} F_i + F_{k+1} = 1 + \sum_{i=0}^{k+1} F_i$.

□

Lema 2.3.5. $\forall k \in \mathbb{N}, F_{k+2} \geq \phi^k$.

Dokaz. Tvrdnju dokazujemo indukcijom.

Baza: $k = 0, k = 1$.
Za $k = 0$ imamo $F_{k+2} = 1$, a za $k = 1$ imamo $F_3 = 2 = \frac{1+\sqrt{9}}{2} > \phi$.
Pretpostavka: Neka je $k \in \mathbb{N}, k \geq 1$. Za $i = 0, 1, \ldots, k$ vrijedi $F_{i+2} \geq \phi^i$.
Korak: Sjetimo se da je ϕ korijen jednadžbe $x^2 - x - 1 = 0$. Slijedi:

$$F_{k+3} = F_{k+2} + F_{k+1} \geq \phi^{k+2} + \phi^{k+1} = \phi^{k+1}(\phi + 1) = \phi^{k+1}\phi^2 = \phi^{k+3}.$$

□

Teorem 2.3.6. *Neka je x proizvoljan čvor u Fibonaccijevoj hrpi H. Neka je $k = d(x)$. Vrijedi*

$$s(x) \geq F_{k+2} \geq \phi^k.$$

Dokaz. Neka je s_k najmanja veličina koju čvor stupnja k može imati. Trivijalno vrijedi $s_0 = 1$ i $s_1 = 2$. Također, za proizvoljan čvor x vrijedi $size(x) \geq s_k$ da s_k raste monotono po k. Neka je z čvor u nekoj Fibonaccijevoj hrpi takav da vrijedi $d(z) = k$

i $s(z) = s_k$. Dovoljno nam je pokazati $s_k \geq F_{k+2}$. Neka su $y_0, \ldots, y_{k-1}$ djeca čvora z poredana po redu ubacivanja pod z. Vrijedi

$$\begin{aligned} s(x) \geq s_k &\geq 2 + \sum_{i=2}^{k} s_{d(y_i)} \\ &\geq 2 + \sum_{i=2}^{k} s_{i-2}, \end{aligned} \tag{2.4}$$

gdje zadnja nejednakost slijedi iz leme 2.3.3 i monotonosti s_k.

Dalje indukcijom pokazujemo $s_k \geq F_{k+2}$.
Baza: $k = 0$ i $k = 1$. Trivijalno vrijedi $s_0 = 1 = F_2$, $s_1 = 2 = F_3$.
Pretpostavka: za $k \in \mathbb{N}, i = 0, \ldots, k$ vrijedi $s_i \geq F_{i+2}$.
Korak:

$$\begin{aligned} s_{k+1} &\geq 2 + \sum_{i=2}^{k} s_{i-2} \\ &\geq 2 + \sum_{i=2}^{k} F_i \\ &=^{2.3.4} F_{k+2} \\ &\geq^{2.3.5} \phi^k. \end{aligned} \tag{2.5}$$

□

Korolar 2.3.7. *Neka je H Fibonaccijeva hrpa s n čvorova. Vrijedi*

$$D(n) \leq \lfloor \log_\phi n \rfloor.$$

Dokaz. Neka je x proizvoljan čvor stupnja k u Fibonaccijevoj hrpi s n čvorova. Teorem 2.3.6 povlači $n \geq s(x) \geq \phi^k$. To je ekvivalentno s $k \leq \lfloor \log_\phi n \rfloor$. □

2.3.4 Složenost operacija

Kao što smo već rekli, koristit ćemo metodu potencijala u analizi složenosti operacija. Neka je H Fibonaccijeva hrpa. Označimo s $t(H)$ broj stabla u listi korijena Fibonaccijeve hrpe H, a s $m(H)$ broj označenih čvorova Fibonaccijeve hrpe H. Definiramo funkciju potencijala:

$$\Phi(H) = t(H) + 2m(H). \tag{2.6}$$

Jednadžba (2.6) naglašava međusobnu ovisnost broja čvorova u listi stabla i oznaka čvorova. Jednadžba (2.6) je ovako konstruirana kako bi mogli "platiti" izvršavanje operacija DecreaseKey i DeleteMin. Pretpostavimo da postoji neka Fibonaccijeva hrpa H s $t(H)$ čvorova u listi stabla i $m(H)$ označenih čvorova. Pretpostavimo da smo upravo izvršili DeleteMin. Ta je operacija složenosti $O(t(H) + D(n))$ i ostavlja nas s $O(\log n)$ stabla. Vidjet ćemo u (2.7) da stvarni trošak operacije DeleteMin podmirujemo potencijalom. Trebamo još vidjeti kako obnavljamo potrošeni potencijal. Potencijal obnavljamo operacijom DecreaseKey. Pretpostavimo da je opracija DecreaseKey odrezala $k > 0$ čvorova. Trošak smo operacije DecreaseKey platili smanjivanjem $m(H)$ za $k-1$, ali, zbog faktora 2, povećali smo i potencijal (dodali smo $k - 1$ čvorova u listu stabla).

Insert

Neka je $\overline{H}$ Fibonaccijeva hrpa dobivena dodavanjem čvora u Fibonaccijevu hrpu H. Vrijedi

$$\Phi(\overline{H}) = t(\overline{H}) + 2m(\overline{H}) = t(H) + 1 + m(H).$$

Pošto je stvarni trošak ubacivanja novog čvora $O(1)$, tada je i amortizirani trošak $1 + O(1) = O(1)$.

DeleteMin

Stvarni trošak operacije DeleteMin je $O(t(H) + D(n))$. To lako možemo utvrditi promatrajući pseudokod iz 2.2.1. Naime, korijen s najmanjim elementom Fibbonacijeve hrpe ima najviše $O(D(n))$ djece. Zbog toga je petlja u funckiji DeleteMin složenosti $O(D(n))$.

U funkciji Consolidate spajamo najviše $t(H)+D(n)-1$ stabla. Svako spajanje stabla ima vremensku složenost $O(1)$.

Sada lako vidimo da je amortizirani trošak jednak

$$\begin{aligned} O(t(H) + D(n)) + D(n) + 1 + 2m(H) - t(H) - 2m(H) & \\ = O(t(H)) + O(D(n)) - t(H) & \\ = O(D(n)). & \end{aligned} \tag{2.7}$$

$O(t(H))$ možemo dominirati tako da korigiramo konstante koje množe $t(H)$ u funkciji potencijala. Potencijal nakon izvršavanja DeleteMin je *najviše* $D(n) + 1 + 2m(H)$ zato što nova hrpa ima najviš $D(n) + 1$ stabla.

DecreaseKey

Dalje pokazujemo da je amortizirani trošak operacije DecreaseKey samo $O(1)$. Vidimo da je stvarni trošak operacije DecreaseKey $O(1)$ plus broj poziva funkcije CascadingCut napravimo. Pretpostavimo da smo napravili m poziva funkcije CascadingCut. Bez rekurzivnog poziva, CascadingCut je vremenske složenosti $O(1)$. Dakle, stvarna vremenska složenost DecreaseKey je $O(c)$.

Izračunajmo promjenu potencijala. DecreaseKey može stvorit najviše c novih stabla i maknuti najviše $c-2$ oznake (CascadingCut dodaje najviše jednu oznaku, a miče najviše $c-1$). Vidimo da vrijedi:

$$t(H) + c + 2(m(H) - c + 2) - t(H) - 2m(H) = 4 - c.$$

Dakle, amortizirani trošak operacije DecreaseKey je najviše

$$O(c) + 4 - c = O(1),$$

zato što možemo namjestiti konstante u funkciji potencijala tako da dominiraju konstantu u $O(c)$.

Meld

Neka je Fibonaccijeva hrpa H nastala spajanjem hrpa H_1 i H_2. Izračunajmo promjenu potencijala. Očito vrijedi $t(H) = t(H_1) + t(H_2)$ i $m(H) = m(H_1) + m(H_2)$.

$$\begin{aligned} &\Phi(H) - (\Phi(H_1) + \Phi(H_2)) \\ &\qquad = (t(H) + 2m(H)) - (t(H_1) + 2m(H_1) + t(H_2) + 2m(H_2)) \\ &\qquad = 0. \end{aligned} \tag{2.8}$$

Pošto je složenost spajanja dvije cirkularne dvosturko vezane liste jednaka $O(1)$, slijedi da je amortizacijska složenost operacije Meld jednaka $O(1)$.

Delete

Delete očito ima amortizacijsku složenost $O(\log n)$ zato što znamo da je amortizacijska složenost DecreaseKey $O(1)$ i DeleteMin $O(\log n)$.

Poglavlje završavamo komentarom na amortizacijsku složenost Fibonaccijeve hrpe. U praksi, Fibonaccijeva je hrpa poprilično dobra u baratanju s donekle normalnim brojem elemenata ($\approx 10^9$). Vidjet ćemo da strukture podataka poput Brodalovog reda, tj. strukture podataka s boljim asimptotskim granicama, nisu nužno bolje u praksi. Konkretno, Brodalov red kojeg analiziramo u idućem poglavlju ima lošije vrijeme izvršavanja za manji broj elemenata (testirano do 10^8).

Poglavlje 3

Brodalov Red

Apstraktna struktura podataka prezentirana u ovom poglavlju zadovoljava asimptotske granice $O(1)$ za FindMin, Meld, Insert i DecreaseKey, te $O(\log n)$ za DeleteMin.

3.1 Osnovna svojstva

U ovom potpoglavlju navodimo invarijante koje struktura podataka treba zadovoljavati, a potom dajemo intuitivan opis invarijanti i dokaze nekih potrebnih tvrdnji. Vidjet ćemo da Brodalov red ima strogo definiranu strukturu. Prvo navodimo definiciju Brodalovog reda i nekih atributa elemenata Brodalovog reda.

Definicija 3.1.1. Brodalov red *je skup dva stabla T_1 i T_2 koja zadovoljavaju invarijante iz 3.1.2, 3.1.3 i 3.1.4. Korijen stabla T_i označavamo s t_i, $i = 1, 2$.*

Djecu čvora spremamo u dvostruko vezanu listu. Svaki čvor ima pokazivač na roditelja i na krajnje lijevo dijete.

Intuitivno, t_1 će uvijek biti najmanji element. Stablo T_2 je pomoćno stablo potrebno za održavanje pravila koja navodimo kasnije.

Stabla Brodalovo reda donekle poštuju svojstvo hrpe, odnosno, u konkretnum slučajevima dopuštamo kršenje tog svojstva. Čvorove koji krše pravilo hrpe nazivamo ***lošim čvorovima***.

Kako je svaki loš čvor, uz djecu korijena t_1 i t_2 i sam korijen t_2, kandidat za drugi najmanji element, trebamo način za praćenje takvih čvorova. To postižemo sa skupovima V i W. Svakom čvoru x Brodalovog reda pridružujemo skupove $W(x)$ i $V(x)$.

Intuicija iza V i W je sljedeća. Svaki čvor koji dodajemo u red, a da on krši svojstvo hrpe, dodajemo u $V(t_1)$ ili $W(t_1)$. Zahtijevamo da je $V(x) \cap W(x) = \emptyset$. Skup V je za čvorove "većeg" ranga, a W za čvorove "manjeg" ranga.

Skupove V i W također implementiramo pomoću dvostruko vezanih listi. $V(x)$ i $W(x)$ implementiramo tako da svakom čvoru x dodajemo 4 pokazivača, dva na prve čvorove u $V(x)$ i $W(x)$ i dva na prethodni i sljedeći element u skupu V ili skupu W u kojoem se on sam nalazi (ako se, naravno, nalazi u jednom). Nove čvorove skupa V dodajemo na početak liste, a nove čvorove skupa W dodajemo tako da čvorovi istih rangova budu grupirani.
Dakle, ako prilikom dodavanja novog čvora u W već postoji čvor istog ranga u W, novi čvor dodamo iza njega. Inače, novi čvor ubacimo na početak ili kraj skupa. To je potrebno kako bi smo mogli održavati jednu od invarijanti opisanu u 3.1.3. Ako čvor nekog ranga ne postoji u W, stavljamo NULL na to mjesto u nizu.

Dalje navodimo 3 vrste invarijanti. Prva opisuje pridjeljivanje ranga čvorovima i tako utvrđuje strukturu stabla T_i. Druga opisuje strukturu skupova V i W. Treća dodaje dodatna pravila na čvorove t_1 i t_2.

Definicija 3.1.2 (Invarijante čvorova)**.** *Neka je x čvor Brodalovog reda. Tada za čvor x trebaju vrijediti sljedeće invarijante:*

S1: Ako je x list, onda je $r(x) = 0$.
S2: $r(x) < r(p(x))$.
S3: Ako $r(x) > 0$, onda $n_{r(x)-1}(x) \geq 2$.
S4: $n_i(x) \in \{0, 2, \ldots, 7\}$.
S5: $T_2 = \emptyset$ ili $r(t_1) \leq r(t_2)$.

S1 i S2 kažu da je rang lista jednak 0 i da rang čvora x strogo raste prema korijenu. S3 kaže da čvor ranga $k > 0$ mora imati barem dvoje djece ranga $k - 1$. S5 kaže da je T_2 prazno stablo, ili ima rang veći od T_1. S4 kaže da je broj djece ranga i čvora x konstantan i ne smije biti 1. Ne dopuštamo da broj čvorova stabla bude 1 kako bi i dalje vrijedila invarijanta S3 nakon što odrežemo djecu najvećeg ranga nekog čvora. $n_i(x) \leq 7$ je posljedica konstrukcije koju kasnije objašnjavamo.

Definicija 3.1.3 (Invarijante strukture)**.** *Neka je $w_i(x)$ broj čvorova skupa $W(x)$ ranga i. Definiramo sljedeće invarijante:*

O1: $t_1 = \min T_1 \cup T_2$.
O2: ako $y \in V(x) \cup W(x)$, onda $y \geq x$.
O3: ako $y \leq p(y)$, onda $\exists x \neq y$ tako da $y \in W(x) \cup V(x)$.
O4: $w_i(x) \leq 6$.
O5: ako $V(x) = (y_{|V(x)|-1}, \ldots, y_1, y_0)$, onda $r(y_i) \geq \lfloor \frac{i}{\alpha} \rfloor$, za $i = 0, 1, \ldots, |V(x)|-1$, gdje je α konstanta.

O1 garantira da je t_1 minimalni element u redu. O2 kaže da čvorovi poštuju svojstvo *min heap* s obzirom na skupove V i W kojima pripadaju. O3 kaže da svi loši čvorovi pripadaju u skup V ili W. O4 i O5 kažu da su veličine V i W $O(\log n)$. Izbacivanjem čvorova iz V ili W ne kršimo invarijante.

Definicija 3.1.4 (Invarijante korijena)**.** *Neka je x čvor Brodalovog reda. Tada za čvor x trebaju vrijediti sljedeće invarijante:*

R1: $n_i(t_j) \in \{2, 3, \ldots, 7\}$, za $i = 0, 1, \ldots, r(t_j) - 1$.
R2: $|V(t_1)| \leq \alpha r(t_1)$.
R3: ako $y \in W(t_1)$, onda $r(y) < r(t_1)$.

R1 garantira da korijeni t_1 i t_2 imaju djecu svakog ranga. R2 veže $|V(t_1)|$ i $r(t_1)$, povećanjem $r(t_1)$ dobivamo novih α mjesta u $V(t_1)$. R3 kaže da loši čvorovi iz $W(t_1)$ moraju imati rang manji od $r(t_1)$.
Dalje dokazujemo neke od tvrdnji koje proizlaze iz gore navedenih pravila. Iz S1 i S3 slijedi sljedeća lema. Ona pokazuje vezu između ranga čvora i veličine stabla.

Lema 3.1.5. *Neka je x čvor nekog stabla T koje zadovoljava S1 i S3. Podstablo s korijenom x, ranga $r(x)$ ima barem $2^{r(x)+1} - 1$ čvorova.*

Dokaz. Dokaz indukcijom po rangu čvora x.
Baza: $r(x) = 0$.
Iz S1 slijedi da je x list, pa podstablo s korijenom u x ima $1 = 2^{0+1} - 1$ čvorova.
Pretpostavka: za čvor x ranga $r(x) = n \in \mathbb{N}$ vrijedi tvrdnja.
Korak:
Neka je x čvor ranga $r(x) = n + 1$. Tada, po S3, slijedi da x ima barem dvoje djece ranga n. Tada podstablo s korijenom u čvoru x ima barem $2(2^{k+1} - 1)$ djece iz čega slijedi tvrdnja. □

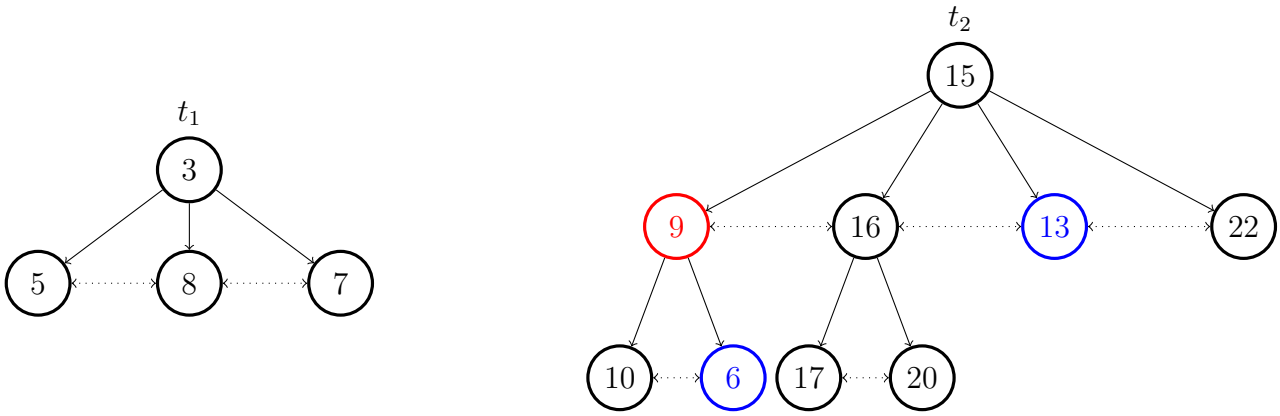
Lema 3.1.6. *Neka je x čvor ranga $r(x)$ nekog stabla T koje zadovoljava S1-S5 i neka je $n \in \mathbb{N}$ broj čvorova u T. Tada su rang i stupanj čvora x $O(\log n)$.*

Dokaz. Dovoljno je tvrdnju pokazati za korijen stabla T. Označimo ga s t. Iz S1-S4 slijedi da n možemo ograničiti odozgo s $7^{r(t)} + 1$ i da se ta granica može postići (svi čvorovi osim listova imaju po 7 djece svakog ranga manjeg od svog ranga). Iz toga slijedi $r(t) \in O(\log n)$, a iz toga i da je stupanj t iz $O(\log n)$. □

Argumentirajmo još zašto O5 povlači $|V(x)| \in O(\log n)$ za proizvoljan čvor Brodalovog reda. To slijedi iz činjenice da je najveći rang u Brodalovom redu iz $O(\log n)$. Neka je $k \in \mathbb{N}$ najveći rang nekog Brodalovog reda. Tada vrijedi $k \geq \lfloor \frac{|V(x)|-1}{\alpha} \rfloor$.

Općenito, održavanje invarijanti O4 i R1 nije trivijalno ako želimo imati $O(1)$ ubacivanje čvorova u stablo i izbacivanje čvorova iz stabla. Zbog toga uvodimo pomoćnu strukturu podataka *vodilju* (eng. *guide*) koju opisujemo u 3.2.

Slika 3.1: Primjer Brodalovog reda. Čvorovi obojani plavo pripadaju W skupu, a čvorovi obojani crveno V skupu.



3.2 Vodilja

U ovom dijelu opisujemo pomoćnu apstraktnu strukturu podataka *vodilju*. Vodilje će nam pomagati u održavanju invarijanti O4 iz 3.1.3 i R1 iz 3.1.4.

Problem koji riješavaju vodilje može se opisati ovako. Pretpostavimo da imamo niz cijelih brojeva $x_n, \ldots, x_1$ za koji želimo da vrijedi $x_i \leq T, i \in \{1, \ldots, n\}$ za neku cjelobrojnu granicu T. Jedina operacija koju na nizu $x_n, \ldots, x_1$ možemo izvoditi je operacija REDUCE(i) koja smanjuje x_i za najmanje 2 i povećava x_{i+1} za najviše 1. Svaki se x_i može "prisilno" povećati, odnosno smanjiti za 1, ali za svaku takvu promjenu u nizu smijemo napraviti $O(1)$ operacija REDUCE kako bi održali $x_i \leq T$. Vodiljin je posao reći na kojim indeksima i izvesti operaciju REDUCE.

Dalje opisujemo implementaciju vodilje.
Vodilji pridružujemo niz cijelih brojeva $\overline{x_n}, \ldots, \overline{x_1}$ takav da vrijedi $x_i \leq \overline{x_i} \in \{T-2, T-1, T\}$. Dokle god vrijedi $x_i \leq \overline{x_i}$, ne trebamo pomoć vodilje. Pomoć vodilje trebamo kada povećavamo $x_i = \overline{x_i}$. Npr. ako $\overline{x_i} = 6, x_i = 4$, tada x_i možemo dva puta povećati bez da trebamo pomoć vodilje.
Primijetite da x_i možemo proizvoljno smanjivati, tj. ako vrijedi $x_i \leq \overline{x_i}$, onda sigurno vrijedi $x_i - 1 \leq \overline{x_i}$.
U daljnjem opisivanju implementacije vodilje BSOMP da vrijedi $T = 2$, tj. $\overline{x_i} \in \{0, 1, 2\}$ i da REDUCE(i) smanjuje $\overline{x_i}$ za 2 i povećava $\overline{x_{i+1}}$ za 1.

Niz $\overline{x_n}, \ldots, \overline{x_1}$ vodilje dijelimo u *blokove* oblika $21^\beta 0$, $\beta \in \mathbb{N}$. Vodilja održava invarijantu koja kaže da broj 2 mora biti dio bloka, dok 1 i 0 ne moraju. Broj 2 nazivamo vođom bloka.

Primjer 3.2.1. *Primjer nekog niza s blokovima koji zadovoljava prethodno opisane uvjete*

$$1, \underline{2, 1, 1, 0}, 1, 1, \underline{2, 0}, \underline{2, 0}, 1, 0, \underline{2, 1, 0}, 0.$$

Za održavanje blokova koristimo dodatan niz pokazivača na pokazivače. Elementi niza koji ne pripadaju niti jednom bloku imaju pridružen NULL pokazivač kojeg ćemo označavati s $\perp$. Elementi niza koji jesu u bloku pokazuju na pokazivač vođe tog bloka. Razlog tome je dvojak:

1. Za bilo koji element niza možemo u vremenu $O(1)$ odrediti nalazi li se u bloku i, ako se nalazi u bloku, vođu bloka u kojem se nalazi.

2. U $O(1)$ vremenu uništiti proizvoljan blok. Jednostavno pridružimo $\perp$ bloku.

Dalje opisujemo operaciju REDUCE(i).
Razlikujemo nekoliko slučajeva:

1. $d_{i+1} = 0$ i d_{i+1} nije dio bloka. $d_{i+1} \leftarrow 1$ i uništimo blok s vođom d_i.

2. $d_{i+1} = 0$ i d_{i+1} je dio bloka. $d_{i+1} \leftarrow 1$ i uništimo blok s vođom d_i i dodamo d_i bloku u kojem je i d_{i+1}.

3. $d_{i+1} = 1$. Uništimo blok s vođom d_i, napravimo novi blok kojem je vođa d_{i+1} i dodamo mu d_i.

Nakon opisane operacije REDUCE(i), možemo opisati operaciju FORCEINCREASE(I) kojom se izvršava "prisilno" povećavanje $\overline{x_i}$. Razlikujemo nekoliko slučajeva:

1. $d_i = 0$ i d_i nije dio bloka. $d_i \leftarrow 1$.

2. $d_i = 0$ i d_i je dio bloka. Napravi REDUCE na vođi i $d_i \leftarrow 1$.

3. $d_i = 1$ i d_i nije dio bloka. $d_i \leftarrow 2$ i napravi REDUCE(i).

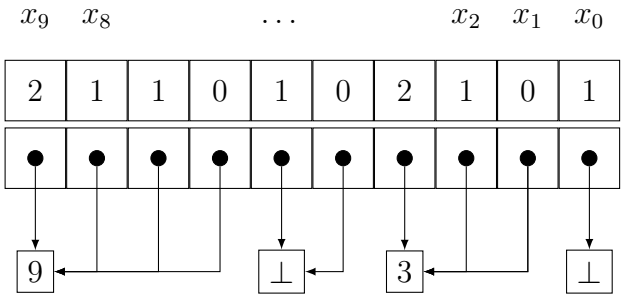
4. $d_i = 1$ i d_i je dio bloka. Napravi REDUCE na vođi tog bloka, $d_i \leftarrow 2$ i napravi REDUCE(i).

Iz gore navedenog je jasno da *vodilja* u $O(1)$ u najgorem slučaju može odlučiti koje REDUCE operacije izvesti.

Primjer 3.2.2. *Jedan slučaj operacije* FORCEINCREASE.

$\underline{2,1,1,0},\underline{2,1,1,1,0}$
$\underline{2,1,1,0},\underline{2,2,1,1,0}$ FORCEINCREASE(3),
$\underline{2,1,1,1},\underline{0,2,1,1,0}$ REDUCE(4),
$\underline{2,1,1,1},\underline{1,0,1,1,0}$ REDUCE(3),
$\underline{2,1,1,1,1,0},1,1,0$ *ponovno uspostavi blokove.*

Slika 3.2: Primjer vodilje.



3.3 Operacije u stablu

Nakon opisa strukture Brodalovog reda i vodilje možemo krenuti opisivati operacije u stablima T_j potrebne kako bi implementirali funkcije prioritetnog reda.

3.3.1 Link i delink

Dvije najosnovnije operacije u stablima su tzv. *link* i *delink* stabla. Pretpostavimo da imamo 3 čvora x_1, x_2, x_3 jednakog ranga, i da niti jedan od njih nije t_i. BSOMP da je x_1 najmanji po vrijednosti od njih, minimum možemo saznati radeći najviše 2 usporedbe. x_2 i x_3 mogu postati krajnje lijeva djeca x_1 i tako povećati rang od x_1 za 1. Time su zadovoljene invarijante S1-S5 i O1-O5.

Primijetite da, iako smo rekli da niti jedan od 3 čvora u operaciji link ne smije biti t_i, operacija link daje nam odgovor na pitanje "Kako dodati novi čvor u Brodalov red sa samo 2 čvora, a da sve invarijante vrijede?"

Operacija delink malo je kompliciranija. Pretpostavimo da imamo neko stablo kojemu je korijen x. Razlikujemo dva slučaja:

1. x ima dvoje ili troje djece ranga $r(x) - 1$. To dvoje/troje djece odrežemo i pridružimo x-u rang krajnje lijevog djeteta plus 1. S4 povlači da S3 i dalje vrijedi (definicija 3.1.2).

2. x ima više od troje djece ranga $r(x) - 1$. Dvoje djece odrežemo. x i dalje ima rang $r(x)$. Sve invarijante i dalje vrijede.

Delink stabla ranga k uvijek rezultira s dva ili 3 stabla ranga $k - 1$ i jednim stablom ranga manjeg ili jednakog k.

3.3.2 Operacije na djeci korijena

Dalje opisujemo kako dodavati i izbacivati djecu korijena, a da i dalje vrijedi R1 (definicija 3.1.4). Za to su nam potrebne 4 vodilje, dvije za t_1 i dvije za t_2. Dalje opisujemo postupak za t_1, konstrukcije za t_2 idu analogno.

Kako bismo mogli izvršiti operacije link i delink u $O(1)$ na djeci korijena proizvoljnog ranga, potreban nam je niz pokazivača na dijete svakog ranga korijena t_1. Zbog R1 (definicija 3.1.4) znamo da takva djeca postoje.

Spomenuli smo da t_1 ima dvije vodilje. Jedna vodilja održava $n_i(t_1) \leq 7$, a druga $n_i(t_1) \geq 2$ za $i = 0, \ldots, r(t_1) - 3$. Djecu ranga $r(t_1) - 1$ i $r(t_1) - 2$ izdvajamo i ručno održavamo zbog međusobne ovisnosti vodilje za gornju granicu i vodilje za donju granicu. Ta ovisnost dolazi do izražaja prilikom malog broja čvorova u stablu ili povećavanja ranga stabla.
Vodilji koja održava gornju granicu dajemo ogradu $T = 7$, a vodilji za donju granicu $T = -2$.

Dodavanje novog djeteta ranga $k < r(t_1) - 2$ ekvivalentno je operaciji FORCEINCREASE(K) na vodilji za gornju granicu. Vodilja nam vrati koje REDUCE operacije trebamo napraviti kako bismo obnovili invarijante. Operacija REDUCE(i) tu odgovara operaciji link na čvorovima ranga i, odnosno poveća $n_{i+1}(t_1)$ za 1 i smanji $n_i(t_1)$ za 3.
Operaciju link radimo samo kada je $n_i(t_1) = 7$ kako ne bismo utjecali na vodilju za donju granicu (ipak smanjujemo za 3). To implicira promjene u vodilji koju ćemo opisati u 3.6.2.
Ako REDUCE poveća broj djece ranga $r(t_1) - 2$ ili $r(t_1) - 1$, napravimo link operaciju na njima i možda povećamo rang od t_1.

Micanje djece je slično, operacija REDUCE odgovara operaciji delink (pošto promatramo vodilju s negativnom granicom T, povećati $\overline{x_i}$ za k znači upravo odrezati mu k djece). Dodatno stablo nastalo kao rezultat operacije delink s rangom $l \in 0, \ldots, k$ dodaje se zasebno, nakon operacije delink, kao dijete t_1 postupkom opisanim gore.

Situacija s korijenom t_2 jest slična. Razlika je ta što je t_1 čvor najmanje vrijednosti, dok t_2 može imati proizvoljnu vrijednost. Zbog te proizvoljne vrijednosti, gore opisane operacije mogu stvoriti nova kršenja svojstva *min heap*. Linking nikad ne stvara loše čvorove, a delinking može stvoriti najviše 3 nova loša čvora. Loši su čvorovi ranga

većeg od t_1 dodani u $V(t_1)$. Kako bi O5 i R2 ostali zadovoljeni, trebamo garantirati da će se rang t_1 povećati i da je α dovoljno velik.

3.3.3 Operacije koje smanjuju broj loših čvorova

Dalje opisujemo operaciju s kojom možemo smanjiti broj *potencijalno loših čvorova* iz $\bigcup_{y \in T_1 \cup T_2} V(y) \cup W(y)$ za najmanje 1.

Pretpostavimo da postoje dva potencijalno loša čvora x_1 i x_2 jednakog ranga $k < r(t_1)$ i da x_1 i x_2 nisu djeca korijena. Prvo trebamo provjeriti jesu li x_1 i x_2 loši. Ako jedan od njih nije, izbacimo ga iz skupa loših čvorova kojem pripada. Ako su oba loša, radimo sljedeće.

BSOMP da su x_1 i x_2 braća. S4 nam garantira da x_1 i x_2 oba imaju barem jednog brata. Ako x_1 i x_2 slučajno nisu braća, uz pretpostavku $p(x_1) \leq p(x_2)$, možemo zamijeniti podstablo kojem je korijen x_1 s podstablom kojem je korijen brat od x_2. Tom zamjenom potencijalno smanjujemo broj loših čvorova za 1 (slijedi iz pretpostavke $p(x_1) \leq p(x_2)$ i činjenice da je x_1 loš).

Neka je y roditelj od x_1 i x_2. Razlikujemo nekoliko slučajeva:

1. y ima više od dvoje djece ranga k. x_1 možemo odrezati i dodati pod t_1 postupkom opisanim u 3.3.2.

2. y ima dvoje djece ranga k i $r(y) > k+1$. Odrežemo x_1 i x_2 i učinimo ih djecom t_1.

3. y ima dvoje djece ranga k i $r(y) = k+1$. Odrežemo x_1, x_2 i y. Zamijenimo y s dijetetom od t_1 ranga $k+1$ kojeg možemo odrezati postupkom opisanim u 3.3.2. Ako je y dijete od t_1, odrežemo samo y. Ako zamjena za y postane loš čvor, dodamo ga u $W(t_1)$. x_1, x_2 i y učinimo djecom t_1 postupkom opisanim u 3.3.2.

3.3.4 Održavanje O4 i R2

Dalje opisujemo kako očuvati invarijante O4 i R2.

Jedini skupovi u koje dodajemo loše čvorove su $V(t_1)$ i $W(t_1)$ (čvorove ne mičemo iz njih prilikom mijenjanja t_1). Loše čvorove ranga većeg od $r(t_1)$ dodajemo u $V(t_1)$, a inače ih dodajemo u $W(t_1)$. Čvorove iz $W(t_1)$ kontrolira vodilja (vodilja održava O4), a čvorove iz $V(t_1)$ kontroliramo ručno.

Nakon dodavanja čvora u $W(t_1)$ radimo operacije koje nam vodilja kaže da napravimo. Operacija REDUCE(i) ovdje odgovara operaciji opisanoj u 3.3.3. Operaciju REDUCE(i) izvršavamo ako je broj loših čvorova u $W(t_1)$ ranga i jednak 6 i barem 2 ta čvora nisu direktna djeca od t_2. Ako ih je više od 4, odrežemo 2 čvora iz $W(t_1)$

i učinimo ih dobrom djecom od t_1. Ne moramo se brinuti o vodilji za donju granicu zato što postoje još 4 čvora ranga i koji su djeca od t_2 (znamo zato što su u $W(t_1)$).

R2 održavamo povećavanjem ranga t_1 nakon svake operacije prioritetnog reda koju napravimo. To postižemo prebacivanjem konstantnog broja djece t_2 pod t_1 Ako je $T_2 = \emptyset$, pravilo $R2$ trivijalno je zadovoljeno zato što vrijedi $V(t_1) = \emptyset$. Pretpostavimo $T_2 \neq \emptyset$. Razlikujemo dva slučaja:

1. $r(t_2) \leq r(t_1)+1$. Odrežemo djecu t_2 najvećeg ranga i dodamo ih pod t_1. Nakon toga dodamo t_2 pod t_1.

2. $r(t_2) > r(t_1) + 1$. Odrežemo dijete t_2 ranga $r(t_1) + 1$. Nakon toga napravimo operaciju delink nad tim čvorom i rezultirajuća stabla dodamo pod t_1.

Ovim se postupkom $r(t_1)$ poveća za najmanje 1.

3.4 Operacije prioritetnog reda

3.4.1 DeleteMin

Algorithm 6 DeleteMin(Q)

```
min ← Q.min
if Q.T2 ≠ NULL then
  if Q.T2.rank = 0 then
    Q.T1 ← Q.T2
    Q.T2 ← NULL
  else
    preseli djecu stabla Q.T2 u Q.T1
    UpdateMin(Q)
else
  if Q.T1.rank = 0 then
    Q.T1 ← NULL
```

Ovoj je operaciji dopušteno da traje $O(\log n)$ vremena. Prvo ispraznimo T_2 tako da prebacimo svu djecu stabla T_2 pod T_1 (korijen t_2 postaje dijete korijena t_1 ranga 0). Nakon toga brišemo t_1. Time dobivamo $O(\log n)$ nezavisnih stabla čiji su korijeni kandidati za novi minimum. Ostali se kandidati za minimum nalaze u skupovima $W(t_1)$ i $V(t_1)$.

Intuitivno, ovo ima smisla, ali nije preočito da se minimum možda ne nalazi u nekom drugom skupu V ili W. Neka je novi minimum označen sa x. Pretpostavimo

Algorithm 7 UPDATEMIN(Q)

```
newMin ← nađi novi minimum u {Q.T1.W, Q.T1.V, Q.T1.children}
if newMin ∉ Q.T1.children then
  zamijeni newMin sa dijetetom Q.T1.children istog ranga
nodes ← odreži svu djecu Q.T1
Q.T1.root ← newMin
for node ∈ nodes do
  if node.rank > Q.T1.rank then
    ubaci čvorove dobivene smanjivanjem ranga node pod Q.T1
  Q.T1.INSERT(node)
pobrini se za loše čvorove iz Q.T1.W ∪ Q.T1.V
```

da x nije jedan od korijena u nizu nezavisnih stabla koja smo dobili izbacivanjem t_1. Pošto je x novi minimum, BSOMP da $x < p(x)$. x je mogao biti dodan u Brodalov red prije ili nakon dodavanja t_1 (prethodnog minimuma). Ako je bio dodan prije, to znači da je x bio minimum. Tada je x morao postati dijete korijena t_2 ili sam korijen t_2 nakon dodavanja novog minimuma u Brodalov red. To znači da je x morao biti dodan u $V(t_1) \cup W(t_1)$ ubacivanjem novog minimuma, ili nekad kasnije prilikom operacije MELD. Ako je x bio dodan nakon prethodnog minimuma, onda je (po pretpostavci $x < p(x)$) morao biti dodan pod stablo T_2. Ponovo, tim dodavanjem x je morao biti dodan u $V(t_1) \cup W(t_1)$. Dakle, korektno je reći da je x u $V(t_1) \cup W(t_1)$, ili je jedan od korijena u listi nezavisnih stabla.

Novi minimum možemo pronaći u vremenu $O(\log n)$ u najgorem slučaju. Ako novi minimum ima roditelja, zamijenimo podstablo kojem je korijen novi minimum s nekom podstablom istog ranga iz niza nezavisnih stabla. Time se stvara potencijalno loš čvor kojeg kasnije zbrinjujemo. Nakon odabira novog minimuma, sva podstabla (kojima po potrebi smanjujemo rang operacijama opisanim u 3.3.1) dodajemo pod novi minimum. Skup $V(t_1) \cup W(t_1)$ dodajemo u skup W novog minimuma, kao i potencijalno loš čvor nastao pronalaskom minimuma. Sve čvorove skupa W pretvaramo u dobre postupkom opisanim u 3.3.3.

3.4.2 Meld

Algorithm 8 MELD(Q_1, Q_2)

Require: $Q_1 \neq \emptyset \wedge Q_2 \neq \emptyset$
  $NQ \leftarrow$ MAKEQUEUE()
  $NQ.T_1 \leftarrow \min\{Q_1.T_1, Q_2.T_1\}$
  $MT \leftarrow$ MAXRANK($Q_1.T_1, Q_1.T_2, Q_2.T_1, Q_2.T_2$)
  if $MT = NQ.T_1$ **then**
    if $NQ.T_1 = Q_1.T_1$ **then**
      INSERT($NQ.T_1$, $Q_2.T_1$)
      INSERT($NQ.T_1$, $Q_2.T_2$)
    else
      INSERT($NQ.T_1$, $Q_1.T_1$)
      INSERT($NQ.T_1$, $Q_1.T_2$)
  else
    $NQ.T_2 \leftarrow MT$
    $arr \leftarrow$ stabla koja nisu $NQ.T_1$ ili $NQ.T_2$
    while LEN(arr) > 0 **do**
      $y \leftarrow$ POP(arr)
      if $y.rank \leftarrow NQ.T_2.rank$ **then**
        $derankArray \leftarrow$ DERANK(y)
        INSERTALL($NQ.T_2$, $derankArray$)
      INSERT($NQ.T_2$, y)
  return NQ

Operacija MELD uključuje najviše četri stabla. Stablo s najmanjim čvorom (T_1 stablo od Q_1 ili Q_2) postaje stablo T_1 novog reda. Ako je novo stablo T_1 stablo najvećeg ranga, ostala stabla dodamo pod T_1 postupkom opisanim u 3.3.2.
Inače (ako postoji stablo ranga *strogo* većeg), stablo najvećeg ranga postaje stablo T_2 novog reda. Tada dodajemo ostala stabla pod stablo T_2 novog reda. Stablo koje dodajemo pod T_2 može biti jednakog ranga kao i T_2. U tom slučaju smanjujemo rang drugog stabla operacijom delink opisanim u 3.3.1. Dodavanje stabla pod T_2 stvara potencijalno loše čvorove. Njih zbrinjujemo postupkom opisanim u 3.3.4. Vodilje i dodatne nizove korijena stabla koje smo dodavali pod stabla T_1 ili T_2 odbacujemo.

3.4.3 Ostale operacije

Ostale se operacije mogu dobiti jednostavnom kombinacijom prethodnih operacija.

- FindMin(Q) je trivijalan. Jednostavno vratimo vrijednost korijena T_1.
- Insert(Q, e) je poseban slučaj operacije Meld.
- DecreaseKey(Q, e, f) zamijeni e sa f. Ako je f manji od t_1, zamijenimo vrijednost ta dva čvora. Ako smo stvorili loš čvor, izvršavamo operaciju opisanu u 3.3.4.
- Delete(Q, e) implementiramo tako da pozovemo DecreaseKey(Q, e, $-\infty$) i DeleteMin(Q).

3.5 Analiza složenosti

3.5.1 Meld

Pošto radimo s najviše 4 stabla, odredit stablo najvećeg ranga i stablo s korijenom najmanje vrijednosti možemo odrediti u konstantnom vremenu.

Dodajemo najviše 16 novh stabla (dva su stabla koja dodajemo možda jednakog ranga kao i stablo najvećeg ranga, a oba mogu imati najviše sedmero djece) pod stablo najvećeg ranga, a lako vidimo iz konstrukcije opisane u 3.3.2 da je dodavanje čvora pod stablo operacija konstantne vremenske složenosti.

Slijedi da vremenska složenost unije dva Brodalova reda mora biti $O(1)$.

3.5.2 DeleteMin

Pretpostavimo da imamo Brodalov red Q s n čvorova čija stabla T_1 i T_2 imaju netrivijalan rang. Prvo što radimo je prebacivanje djece stabla T_2 u stablo T_1. Vremenska složenost toga je $O(\log n)$ zato što T_2 ima najviše $O(\log n)$ djece, a dodavanje novog čvora pod t_1 je složenosti $O(1)$.

Dalje tražimo novi čvor najmanje vrijednosti. Pošto iz definicije 3.1.3 slijedi da $|V|, |W| \in O(\log n)$, traženje novog čvora najmanje vrijednosti vremenske je složenosti $O(\log n)$. Zamjena dva stabla je složenosti $O(1)$. Rezanje djece stabla T_1 je operacija složenosti $O(\log n)$, kao što je i dodavanje pod novo T_1 stablo. Još se moramo pobrinuti za loše čvorove. U ovoj operaciji smo stvorili jedan potencijalno loš čvor. Brisanje čvora iz skupa V je trivijalno i složenosti $O(1)$ (samo dodamo čvor pod T_1 ako već nije tamo).

Brisanje čvora iz W smo opisali u 3.3.3. Iz konstrukcije vidimo da je i ovo brisanje složenosti $O(1)$. Dakle, zbrinjavanje loših čvorova je složenosti $O(\log n)$.

Iz svega ovoga vidimo da je složenost operacije DeleteMin $O(\log n)$.

3.5.3 Insert

Očito je složenosti $O(1)$ zato što je Insert poseban slučaj Meld.

3.5.4 DecreaseKey

Mijenjanje vrijednosti dvaju čvorova je operacija složenosti $O(1)$. Za potencijalno loš čvor pobrinemo se postupkom opisanim u 3.3.4 koji je složenosti $O(1)$.

3.5.5 Delete

Pošto je ovo kombinacija DeleteMin i DecreaseKey, ova je operacija složenosti $O(\log n)$.

3.6 Detalji implementacije

U ovom dijelu detaljnije opisujemo implementaciju i napominjemo neke detalje koje smo ispustili, a bitni su.

3.6.1 Opis implementacije struktura podataka

Svaki čvor sadrži sljedeće elemente:

- vrijednost asociranu s čvorom
- rang
- pokazivač na prethodni i sljedeći čvor
- pokazivač na roditelja
- pokazivač na krajnje lijevo dijete
- pokazivače na prve čvorove u W i V skupovima tog čvora
- pokazivač na prethodni i sljedeći čvor u V ili W listi u kojoj se čvor nalazi.

Korijeni sadrže sljedeće elemente.

- Čvor koji je sam korijen.
- Vodilje za gornju i donju granicu.
- Niz pokazivača na djecu ranga $i = 0, \ldots, r(t_j) - 1$.

Korijen stabla T_1 sadrži još i niz pointera na loše čvorove u $W(t_1)$ ranga $i = 0, \ldots, r(t_1) - 1$ i vodilju za broj čvorova u $W(t_1)$.

3.6.2 Izmjene vodilje

Prilikom opisa ASP-a *vodilja* i njene uloge u balansiranju stabla Brodalovog reda, nismo se osvrnuli na jedan sitan, ali ključan detalj. Detalj opisujemo sljedećim pitanjem.

Što bi se trebalo dogoditi ako odrežemo sedmo dijete ranga k korijena t_i?
Odgovor je, naravno, ništa. Odrežemo to dijete, pitamo vodilju kako izbalansireti stablo nakon rezanja i vodilja nam kaže da ga ne treba balansirat zato što korijen t_i ima još 6 djece ranga k. Zašto se uopče zanemarujemo ovime? Zato što nismo nigdje rekli vodilji gornje granice da na k-tom mjestu niše nema 7, nego ima 6, a u 3.3.2 smo rekli da operaciju REDUCE radimo samo kada djece ranga k ima točno 7. O ovome se trebamo brinuti samo ako se pozove REDUCE(k). U tom slučaju, operacija REDUCE samo uništi blok kojem je voditelj bio na k-tom indeksu.

Poglavlje 4

Analiza performansi

U ovom poglavlju uspoređujemo performanse Fibonaccijeve hrpe i Brodalovog reda. Performanse ćemo uspoređivati sortiranjem niza nasumičnih brojeva duljine n. Za generiranje niza pseudoslučajnih brojeva koristili smo `math/rand` modul programskog jezika Go.

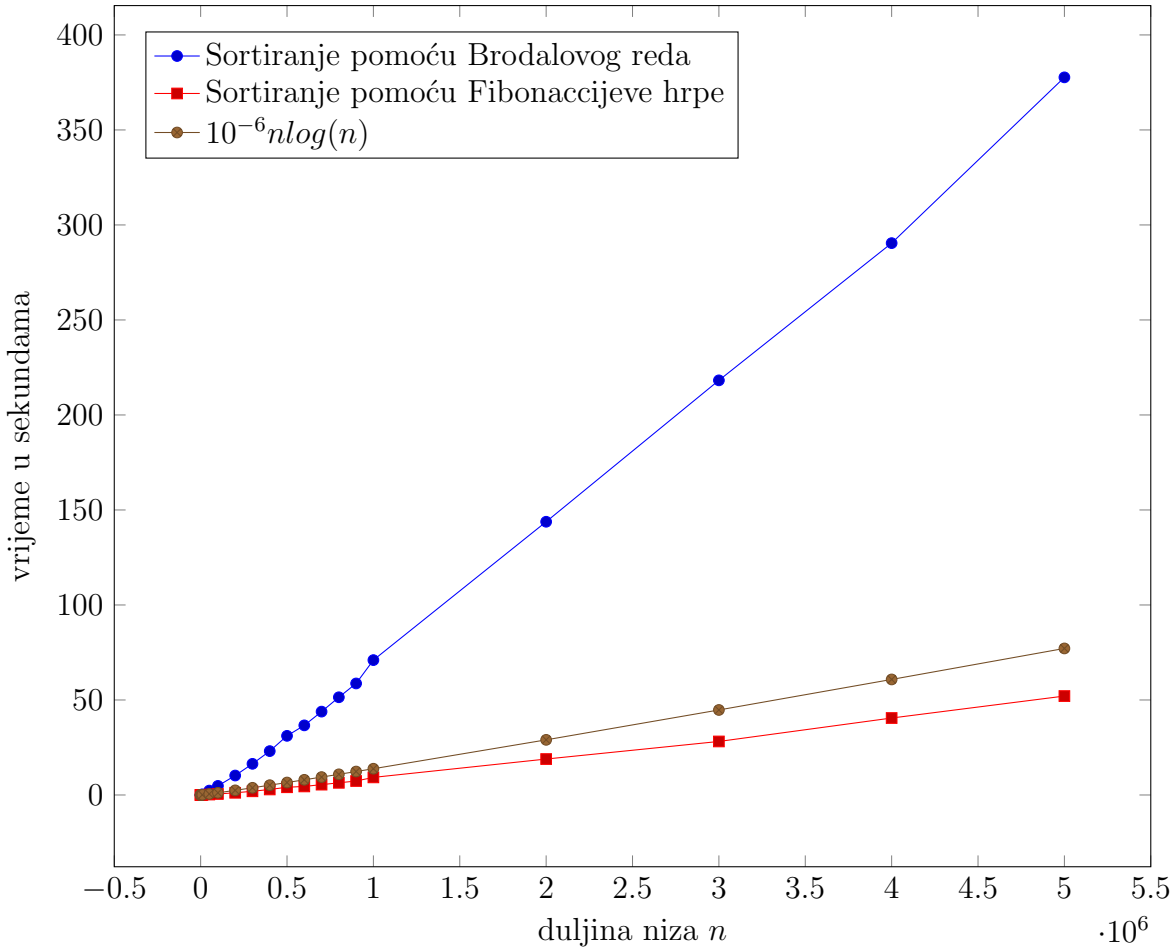
Testiranje provodimo na laptopu Acer Swift 314-54, s procesorom Intel Core i5-8250U i 8GB RAM-a. Implementacija Brodalovog reda i Fibonaccijeve hrpe pisana je u programskom jeziku Go.

4.1 Sortiranje

Sortiranje provodimo na nizu elemenata tipa `float64`, duljine 10^m i $5 \cdot 10^m$, $m \in 1, \ldots, 6$.

	Brodalov red	Fibonaccijeva hrpa
$n = 10$	0.000056 s	0.000012 s
$n = 50$	0.000548 s	0.000077 s
$n = 10^2$	0.001482 s	0.000162 s
$n = 5 \cdot 10^2$	0.012276 s	0.001530 s
$n = 10^3$	0.022344 s	0.002159 s
$n = 5 \cdot 10^3$	0.160780 s	0.014188 s
$n = 10^4$	0.324206 s	0.026189 s
$n = 5 \cdot 10^4$	2.310226 s	0.233117 s
$n = 10^5$	4.780583 s	0.537613 s
$n = 5 \cdot 10^5$	31.125398 s	4.065474 s
$n = 10^6$	70.986380 s	9.258329 s
$n = 5 \cdot 10^6$	6 min 17.6736 s	52.062594 s

Slika 4.1: Usporedba vremena sortiranja niza nasumično generiranih vrijednosti duljine n



Iako pomalo neočekivani, ovi rezultati imaju smisla.

Objasnimo prvo način na koji sortiramo niz. Sve elemente niza dodamo u prioritetni red. Nakon toga, sve elemente prioritetnog reda izbacujemo van koristeći operaciju DeleteMin. To nam daje algoritam sortiranja složenosti $n(O(f(n))+O(g(n)))$, gdje je operacija Insert složenosti $O(f(n))$, a operacija DeleteMin složenosti $O(g(n))$.

Lako vidimo da, iako je operacija DeleteMin Brodalovog reda bolje složenosti u najgorem slučaju, ona je puno "teža" od operacije DeleteMin Fibonaccijeve hrpe.

Također, lako provjerimo da iterativno izvođenje operacije DeleteMin Fibonaccijeve hrpe je složenosti $O(\log n)$ u najgorem slučaju (lista korijena je duljine $O(\log n)$ svaki put nakon prvog DeleteMin). Pošto u operaciji DeleteMin Fibonaccijeve hrpe, iteriramo samo dva puta po listi duljine $O(\log n)$, jasno je da će ona imat bolje vrijeme izvođenja.

Vidimo da je primjer analize složenosti sortiranja niza pomoću Fibonaccijeve hrpe sličan primjeru 2.3.1.

Bibliografija

[1] G. S. Brodal, *Worst-case Efficient Priority Queues*, (1996), `https://cs.au.dk/~gerth/papers/soda96.pdf`.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest i C. Stein, *Introduction to Algorithms Third Edition*, 2009.

[3] J. R. Driscoll, H. N. Gabow, Shrairman R. i Tarjan R. E., *An alternative for fibonacci heaps with applications to parallel computation*, (1988).

[4] A. Elmasry, C. Jensen i J. Katajainen, *On the Power of Structural Violations in Priority Queues*, (2007), 45–53, `http://crpit.scem.westernsydney.edu.au/abstracts/CRPITV65Elmasry.html`.

[5] M. L. Fredman i R. E. Trajan, *Fibonacci heaps and their uses in inproved network optimization algorithms*, (1984).

[6] R. Manger, *Strukture podataka i algoritmi*, 2014.

Sažetak

U ovom smo radu prezentirali dvije asimptotski efikasne implementacije prioritetnog reda.

U prvom smo poglavlju dali definiciju prioritetnog reda. Također, naveli smo i neke dodatne definicije koje smo koristili kroz ovaj rad.

U drugom poglavlju bavili smo se analizom i opisom implementacije Fibonaccijeve hrpe. Dali smo intuitivan opis implementacije operacija prioritetnog reda i dokazali smo tvrdnje potrebne za analizu složenosti te opisali metodu kojom analiziramo složenost. Također smo dali pseudokodove kompleksnijih operacija. Na kraju smo poglavlja analizirali složenost metodom opisanom u istom poglavlju.

U trećemo smo poglavlju opisali i analizirali Brodalov red. Prvo smo opisali strogu strukturu Brodalovog reda. Potom smo dali opis pomoćne strukture podataka *vodilje*. Nakon toga opisali smo osnovne operacije nad stablima Brodalovog reda i implementaciju operacija prioirtetnog reda, kao i pseudokod. Na kraju smo napravili analizu složenosti operacija.

U zadnjem smo poglavlju uspoređivali performanse Fibonaccijeve hrpe i Brodalovog reda pomoću sortiranja niza nasumice generiranih brojeva.

Summary

We have presented two asymptotically efficient implementation of a priority queue in this thesis.

In the first chapter we have given the definition of priority queue. We have, also, given some additional definitions which we used throughout this thesis.

In the second chapter we have analysed and described the implementation of Fibonacci heap. We gave a intuitive description of the implementation of priority queue operations and proved claims needed for the complexity analysis. We, also, briefly describe the complexity analysis method and gave pseudocodes of some operations. We ended the chapter by analysing complexity using the method described in this chapter.

In the third chapter we have described and analysed Brodal queue. Firstly, we have described the strict structure of Brodal queue. Then we described a helper data structure called *guide*. After that, we have described simple tree operations *link* and *delink* and implementation of priority queue operations. Finally, we analysed the worst case complexity of implemented operations.

In the last chapter, we have compared performance of Fibonacci heap and Brodal queue. We did this by sorting an array of randomly generated numbers using priority queues.

Životopis

Rođen sam 28. 11. 1998. godine u Zagrebu. Osnovno sam obrazovanje završio u Osnovnoj školi Pantovčak.

2013. sam godine upisao opći smjer Gimnazije Lucijana Vranjanina koji sam završio 2017. godine.

2017. sam godine upisao preddiplomski sveučilišni studij Matematika na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta u Zagrebu. Preddiplomski sam studij završio 2020. godine te sam odmah upisao diplomski sveučilišni studij Računarstvo i matematika na istom fakultetu. Od listopada 2020. godine do svibnja 2021. godine sam radio kao backend developer u startup tvrtci Atomic Intelligence. Od svibnja 2021. godine radim u tvrtci Photomath kao Junior DevOps inženjer.