Dekartovo stablo - Treap

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

Vasilije Todorović

Jul 2024

Sažetak

Dekartovo stablo, poznato i kao trip, predstavlja nedeterminističku strukturu podataka koja može biti dobra zamena za neke složenije vrste balansiranih pretraživačkih stabala. Pored svoje jednostavne ideje i implementacije, uspeva da postigne rezultate koji su u prosečnom slučaju jednaki postojećim determinističkim strukturama za predstavljanje binarnih balansiranih pretraživačkih stabala.
Ključne reci: strukture podataka, randomizovani algoritmi, balansirana pretraživačka stabla

Sadržaj

1 Uvod

Dekartovo stablo, poznato i kao trip (*eng. treap*), je vrsta binarnog balansiranog pretraživačkog stabla koje kombinuje osobine binarnog pretraživačkog stabla i hipa ($Tree + Heap = Treap$). Elementi Dekartovog stabla, odnosno čvorovi, skladište parove vrednosti (X, Y), gde X predstavlja ključ, a Y prioritet datog čvora [2]. Zbog ovakvog načina skladištenja podataka, Dekartovo stablo nosi svoje ime jer se čvorovi mogu predstaviti tačkama, a grane dužima u Dekartovoj ravni.

Struktura je organizovana tako da je stablo binarno pretraživačko u odnosu na elemente X, a hip u odnosu na elemente Y − odnosno važe sledeći uslovi:

- Za svaki čvor v u tripu T važi da je vrednost njegovog ključa veća od svih vrednosti ključeva u njegovom levom podstablu i manja od vrednosti svih ključeva u njegovom desnom podstablu (uslov binarnog pretraživačkog stabla).
- Za svaki čvor v u tripu T važi da je njegov prioritet veći od svih prioriteta u oba podstabla (uslov max-hipa, analogno za min-hip)

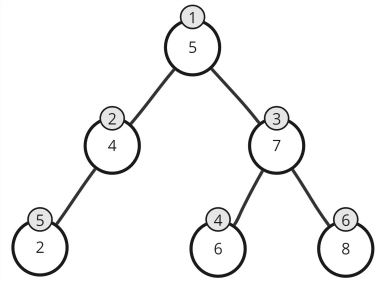
Na osnovu date definicije koja je rekurzivne prirode, može se zaključiti da je svako podstablo Dekartovog stabla takođe Dekartovo stablo. Osnovne operacije nad ovim stablom imaju logaritamsku vremensku složenost u odnosu na ukupan broj čvorova, što će u nastavku biti eksperimentalno pokazano.

2 Motivacija i ideja

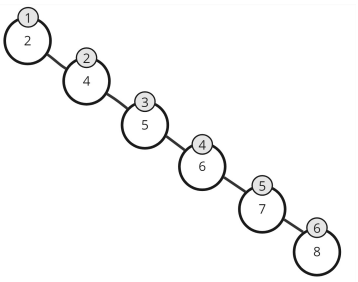
Razmotrimo motivaciju uvođenja Dekartovog stabla razmatranjem izazova sa kojima se suočavaju binarna pretraživačka stabla. Osnovne operacije poput pretraživanja, umetanja i brisanja čvorova u binarnom pretraživačkom stablu zahtevaju $O(k)$ vremena, gde k predstavlja dubinu stabla. Međutim, ukoliko se stablo konstruiše na određen način, ove operacije se mogu izvršiti u vremenu $O(\log n)$, gde je n ukupan broj čvorova u stablu.

Na primeru konstrukcije binarnog pretraživačkog stabla možemo videti da je redosled umetanja elemenata ključan u definisanju izgleda samog stabla. Ukoliko bismo želeli da konstruišemo binarno pretraživačko stablo od elemenata redom 5, 4, 7, 6, 2, 8, možemo videti na slici 1 da je stablo balansirano, što rezultira da je vreme izvršavanja osnovnih operacija $O(\log n)$. Ukoliko bismo ove elemente promešali, mogli bismo da dobijemo redosled elemenata u sortiranom poretku. Na slici 2 možemo videti da je takvo stablo nebalansirano i degenerisano u listu, a vreme izvršavanja osnovnih operacija je $O(n)$.

Pošto se unos u stablo izvršava redom, elementi koji se nalaze bliže početku imaju veći prioritet u odnosu na elemente koji im slede, odnosno biće ranije uneti u stablo. Prioritet elementa je na ovaj način definisan pozicijom u nizu − elementi sa nižim indeksom će se prvi uneti u stablo. Na slikama 1 i 2, koje prikazuju dva primera strukture binarnog pretraživačkog stabla, su pored vrednosti koje čuvaju čvorovi, prikazani i prioriteti svakog od čvorova. Ono što možemo primetiti jeste da je prioritet svakog čvora manji od prioriteta svih čvorova koji se nalaze u njegovim podstablima (uslov min-hipa, uz drugačiji izbor prioriteta moguć max-hip). Ukoliko posmatramo binarno pretraživačko stablo na ovaj način, možemo videti da su ispunjena oba uslova Dekartovog stabla.



Slika 1: Primer balansiranog binarnog pretraživačkog stabla



Slika 2: Primer nebalansiranog binarnog pretraživačkog stabla

Vidimo da izbor prioriteta igra ključnu ulogu u izgledu samog stabla. Jedna od strategija za izbor prioriteta koja se prirodno nameće kako bi se izbegao slučaj da konstruišemo stablo koje će biti degenerisano u listu jeste da početni niz na nasumičan način promešamo i potom elemente redom unesemo u stablo, tj. dodelimo im prioritete u odnosu na poziciju u novodobijenom nizu i od njega konstruišemo Dekartovo stablo (varijanta min-hip). Na ovaj način postojeća struktura postaje randomizovana, odnosno, ukoliko bismo više puta ponovili postupak konstrukcije Dekartovog stabla nad istim nizom, dobili bismo različite strukture stabala.

Pošto su podaci dinamičke prirode, često se menjaju i dodaju, a i često nisu svi podaci odmah poznati, problem koji nastaje u prethodnoj definiciji Dekartovog stabla jeste *Šta se događa sa naknadno dodatim elementima, koji se nisu nalazili u početnom nizu? Koji prioritet dodeliti takvom elementu, kada su prvih n prioriteta već dodeljeni elementima iz početnog niza?*. Ukoliko bismo takvom elementu dodelili najmanji prioritet (npr. $n + 1$), kako bi postao list u Dekartovom stablu, može se desiti da nakon unosa takvog elementa sledi pojedinačan unos redom sortiranih elemenata koji su veći od njega, koji će redom dobijati sve manji i manji prioritet čime se ponovo događa delimična degenerisanost, što dovodi do nebalansiranosti čitavog stabla.

Ponovo, rešenje koje je najjednostavnije i koje daje zadovoljavajuće rezultate leži u nasumičnosti. Dosadašnju definiciju Dekartovog stabla ćemo promeniti tako što ćemo prilikom konstruisanja stabla prioritete svakog čvora birati na nasumičan način iz dovoljno širokog opsega vrednosti, čime se podaci koji nisu bili odmah poznati tretiraju na isti način kao i podaci iz početnog niza, čime je izbegnut problem degenerisanja podstabla u listu.

3 Operacije nad Dekartovim stablom

Nad Dekartovim stablom moguće je izvršiti sledeće operacije:

- **Search** (T, k) - traži čvor sa ključem k u tripu T.
- **Insert** (T, k, p) - umeće novi čvor u trip T sa ključem k i prioritetom p. Prioritet p je generisan na slučajan način.
- **Delete** (T, k) - briše čvor sa ključem k iz tripa T.
- **Split** (T, a) - deli trip T na dva tripa T_1 i T_2 po uslovu a, tako da $T_1 = \{x \in T \mid x.key < a\}$ i $T_2 = \{x \in T \mid x.key > a\}$

- **Merge** (T_1,T_2) - spaja tripove T_1 i T_2 u jedan trip, pod uslovom da su vrednosti ključeva u T_1 manje od vrednosti ključeva u T_2.

Sve navedene operacije se u prosečnom slučaju izvršavaju u vremenu O(log$n$), gde je n ukupan broj čvorova u tripu. Razmotrimo kako izgledaju algoritmi koji implementiraju navedene operacije.

3.1 Pretraga čvora

Operacija pretraživanja Dekartovog stabla po zadatom ključu k se izvršava kao i kod binarnog pretraživačkog stabla. Ukoliko je početno stablo prazno, procedura vraća *null*, što znači da čvor nije pronađen. Ukoliko je ključ trenutnog čvora jednak zadatom ključu, procedura vraća taj čvor. U suprotnom, ako je zadati ključ veći od ključa trenutnog čvora, pretraga se nastavlja rekurzivno u desnom podstablu. Ako je zadati ključ manji, pretraga se nastavlja rekurzivno u levom podstablu. Na kraju, procedura vraća čvor sa traženim ključem ako je pronađen, ili *null* ako nije.

```
1  procedure Search(T: Treap, k: KeyType)
2    if T.key = null then
3        return null
4    if T.key = k then
5        return T
6    else if T.key < k then
7        return Search(T.right,k)
8    else
9        return Search(T.left,k)
```

3.2 Umetanje novog čvora

Operacijom umetanja se ubacuje novi čvor u Dekartovo stablo sa zadatim ključem k i prioritetom p. Ako je početno stablo prazno, kreira se i vraća novi čvor sa datim ključem i prioritetom. U suprotnom, procedura se rekurzivno poziva za desno podstablo ako je ključ novog čvora veći od trenutnog ključa, ili za levo podstablo ako je ključ manji. Nakon umetanja, ako prioritet desnog ili levog podstabla pređe prioritet trenutnog čvora, vrši se odgovarajuća rotacija [6] kako bi se očuvala svojstva hipa (u ovom slučaju je obrađen primer max-hipa). Ako je ključ već prisutan u stablu, čvor se ne ubacuje ponovo. Na kraju, vraća se koren stabla sa novim čvorom umetnutim na odgovarajuće mesto.

```
1  procedure Insert(T: Treap, k: KeyType, p: PriorType)
2    if T = null then
3        return new Treap(k,p)
4    else if T.key < k then
5        T.right = Insert(T.right,k,p)
6        if T.right.priority > T.priority then
7            RotateLeft(T)
8    else if T.key > k then
9        T.left = Insert(T.left,k,p)
10       if T.left.priority > T.priority then
11           RotateRight(T)
12   else
13       /* Cvor je vec u stablu*/
14   return T
```

3.3 Brisanje čvora

Operacija brisanja čvora sa zadatim ključem k počinje proverom da li je trenutno stablo prazno − ukoliko jeste vraća se *null* kao rezultat koji označava da čvor sa datim ključem ne postoji. Zatim se vrši provera da li je zadati ključ k manji ili veći od ključa u korenu stabla, što određuje da li će se brisanje nastaviti rekurzivno u levom ili desnom podstablu. Kada se nađe čvor v sa traženim ključem k:

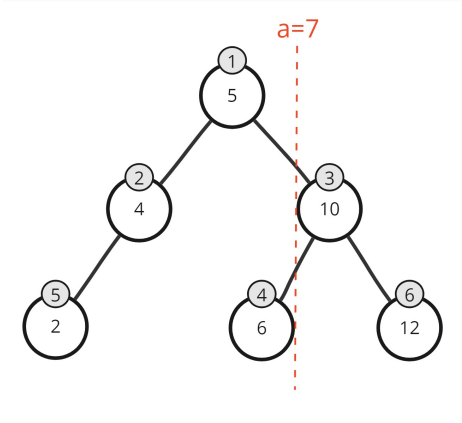
- Ukoliko čvor v nema levog potomka, zamenjuje se desnim potomkom i v se briše. Analogno u slučaju da nema desnog potomka.
- Ukoliko čvor v ima oba potomka, vrši se rotacija u skladu sa prioritetima kako bi se održala svojstva Dekartovog stabla, a zatim se rekurzivno vrši brisanje u odgovarajućem podstablu, čime se brisanje svodi na prethodni slučaj.

Na kraju se vraća ažurirani koren stabla T, sa obrisanim čvorom ili bez promena ako čvor sa ključem k nije pronađen.

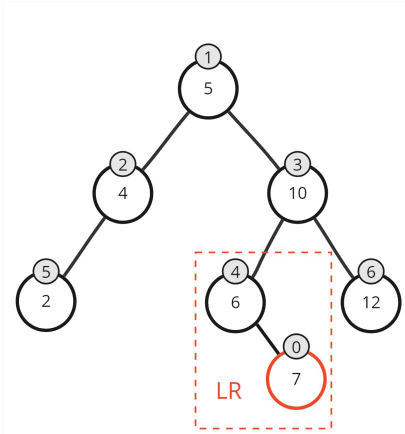
```
1  procedure Delete(T: Treap, k: KeyType)
2    if T = null then
3        return null
4    if k < T.key then
5        T.left = Delete(T.left, k)
6    else if k > T.key then
7        T.right = Delete(T.right, k)
8    else
9        if T.left = null then
10           r = T.right
11           delete T
12           T = r
13       else if T.right = null then
14           l = T.left
15           delete T
16           T = l
17       else
18           if T.left.priority > T.right.priority then
19               RotateRight(T)
20               T.right = Delete(T.right, k)
21           else
22               RotateLeft(T)
23               T.left = Delete(T.left, k)
24   return T
```

3.4 Podela Dekartovog stabla

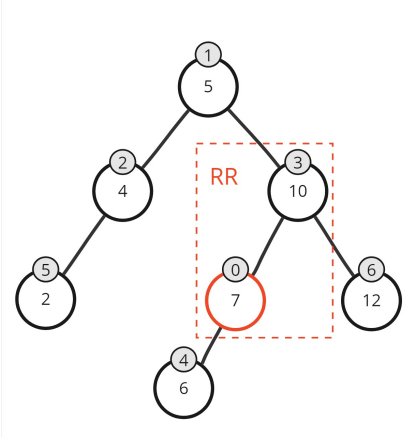
Operacija podele Dekartovog stabla po uslovu a deli stablo u dva Dekartova stabla, tako da se u prvom stablu nalaze čvorovi čije su vrednosti ključeva manje od a, a u drugom stablu čvorovi čije su vrednosti ključeva veće od a. Ova operacija može biti zgodna ukoliko želimo da izvršimo upite raspona. Implementacija ove operacije se svodi na korišćenje prethodnih osnovnih operacija. Naime, možemo veštački ubaciti novi čvor s ključem a koji ima najveći prioritet, čime će taj novi čvor postati koren Dekartovog stabla. U njegovom levom podstablu će se naći vrednosti ključeva koje su manje od a, a u desnom vrednosti koje su veće od a. Na narednim slikama možemo videti demonstraciju operacije podele stabla po uslovu $a = 7$.



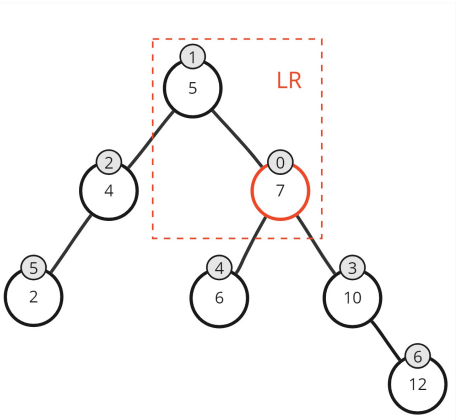
Slika 3: Prikaz granice razdvajanja



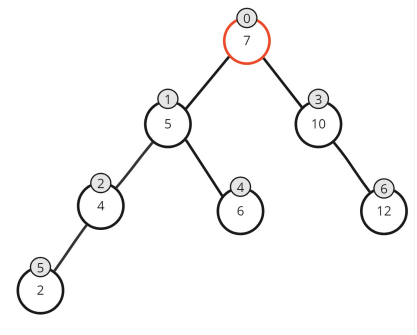
Slika 4: Dodavanje čvora sa ključem 7. Uslov min hipa nije zadovoljen - potrebno je izvršiti levu rotaciju.



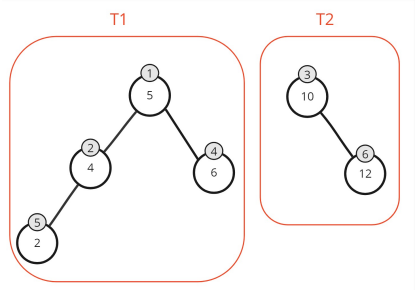
Slika 5: Nakon leve rotacije i dalje nije zadovoljen uslov min hipa - potrebno je izvršiti desnu rotaciju.



Slika 6: Nakon desne rotacije i dalje nije zadovoljen uslov min hipa - potrebno je izvšiti levu rotaciju.



Slika 7: Rezultujuće Dekartovo stablo sa novododatim čvorom.



Slika 8: Dekartova stabla T_1 i T_2 dobijena brisanjem korena.

```
procedure Split(T: Treap, a: KeyType)
  T = Insert(T,a,inf)
  [T1,T2] = T.left, T.right
  return [T1,T2]

procedure rotateRight(T:Treap)
  Treap l = T->left;
  T->left = l->right;
  l->right = T;
  T = l;

procedure rotateLeft(T: Treap)
  Treap r = T->right;
  T->right = r->left;
  r->left = T;
  T = r;
```

3.5 Spajanje Dekartovih stabala

Operacija spajanja dva Dekartova stabla T_1 i T_2, pri čemu T_1 ima ključeve čije su vrednosti manje od vrednosti ključeva iz T_2, odvija se tako što se dodaje pomoćni čvor i kao deca mu se dodeljuju T_1 i T_2. Nakon toga, potrebno je obrisati pomoćni čvor, odnosno koren i rezultujuće stablo će biti Dekartovo stablo koje sadrži početna dva Dekartova stabla. Pažljivim posmatranjem može se primetiti da su koraci pri spajanju dva Dekartova stabla obrnutog redosleda u odnosu na razdvajanje Dekartovog stabla na dva stabla po određenom uslovu.

```
procedure Merge(T1: Treap, T2: Treap)
  T = new Treap(x) /* x nije u skupu kljuceva*/
  T.left = T1
  T.right = T2
  T = Delete(T,x)
  return T
```

4 Procena vremenske složenosti operacija

Nakon što smo videli način implementacije operacija nad Dekartovim stablom, potrebno je proceniti koliko vremena je potrebno za njihovo izvršavanje. Posmatraćemo prosečno vreme izvršavanja ovih operacija i analizirati kako se ono menja u zavisnosti od broja čvorova u stablu.

Operacija pretrage čvora po zadatom ključu će se izvršiti u najgorem slučaju u vremenu $O(k)$, gde k predstavlja dubinu stabla.

Operacija umetanja čvora po zadatom ključu i prioritetu se sastoji iz operacije pretrage, pošto je potrebno pronaći mesto na koje se umeće novi čvor, a potom nizom rotacija dovesti ga na ispravno mesto tako da zadovoljava uslove hipa. Umajući to u vidu, ukupan broj operacija potrebnih za umetanje čvora je $O(k) + O(k) \cdot O(1) = O(k)$ operacija – u najgorem slučaju za pronalazak mesta novog čvora je potrebno $O(k)$ operacija a potom $O(k)$ rotacija, gde se jedna rotacija izvršava u vremenu $O(1)$ [6], za dubinu stabla k.

Za operaciju brisanja čvora sa zadatim ključem je potrebno prvo pronaći dati čvor, a potom, u najgorem slučaju, izvršiti $O(k)$ rotacija kako bi

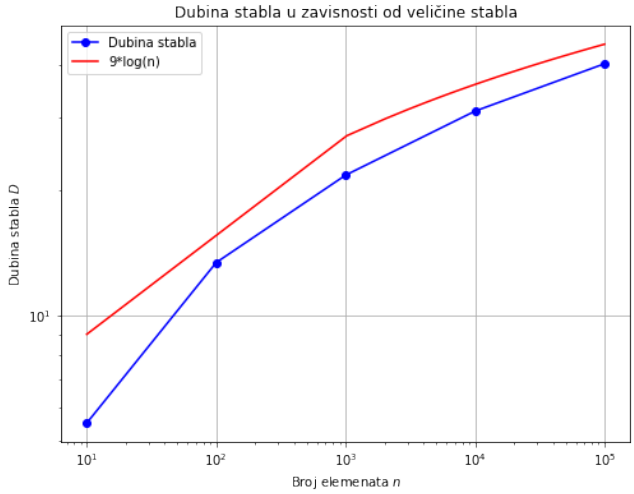
se čvor doveo u poziciju lista i jednostavno obrisao - čime je ukupan broj operacija takođe O(k), za dubinu stabla k.

Operacije podele i spajanja Dekartovog stabla se sastoje iz operacije brisanja i umetanja, tako da je ukupan operacije jednak O(k), za dubinu stabla k.

Postavlja se pitanje *Koju vrednost će imati dubina stabla?*. U najgorem slučaju dubina stabla može biti n, gde n predstavlja ukupan broj čvorova i u tom slučaju stablo je degenerisano u oblik liste. Na osnovu načina konstrukcije ove strukture podataka očekivano je da će se taj slučaj jako retko dešavati. Iz tog razloga ima smisla posmatrati kolika je dubina stabla u *prosečnom slučaju*. Da bismo to procenili, za niz od n različitih elemenata ćemo izgenerisati više različitih stabala sa nasumičnim prioritetima. Nakon generisanja stabala, izračunaćemo dubinu svakog stabla i te vrednosti uprosečiti. U tabeli 1 mogu se videti prosečne dubine stabala za različit broj elemenata n. Takođe, na slici 9 se mogu videti ove vrednosti prikazane na logaritamskoj skali. Na prikazanom grafiku se jasno vidi da počev od nekog n_0 postoji funkcija oblika $c \cdot log(x)$, koja se nalazi uvek iznad funkcije koja definiše dubinu stabla, što nam omogućava da zaključimo da je dubina stabla ograničena logaritamskom funkcijom, odnosno da raste kao $O(\log n)$ [7]. Takođe u tabeli 2 se mogu videti vremena izvršavanja operacija umetanja, brisanja i pretrage čvora, koja prate logaritamsku funkciju.

Tabela 1: Dubina stabla u zavisnosti od veličine stabla

Broj elemenata n:	10	100	1000	10000	100000
Dubina stabla k:	5.5	13.4	21.7667	31.05	40.38



Slika 9: Dubina stabla u zavisnosti od veličine stabla i logaritamska funkcija, prikazani na logaritamski skaliranim osama

Tabela 2: Vreme izvršavanja operacija u milisekundama

N	Insert (ms)	Delete (ms)	Search (ms)
10	1.989×10^{-4}	1.562×10^{-4}	1.617×10^{-4}
100	5.926×10^{-4}	2.010×10^{-4}	1.849×10^{-4}
1000	1.351×10^{-3}	5.189×10^{-4}	2.919×10^{-4}
10000	2.852×10^{-3}	1.472×10^{-3}	4.536×10^{-4}
100000	5.563×10^{-3}	2.968×10^{-3}	1.027×10^{-3}

5 Zaključak

U ovom radu smo analizirali Dekartovo stablo kao strukturu podataka i operacije koje je nad njim moguće izvršiti. U prosečnom slučaju, osnovne operacije se izvršavaju u vremenu $O(\log n)$, dok je za konstrukciju Dekartovog stabla od n elemenata potrebno $O(n \log n)$ vremena. Važno je napomenuti da postoje specifične konstrukcije koje omogućavaju konstrukciju Dekartovog stabla u vremenu $O(n)$, ukoliko je niz, nad kojim se formira stablo, unapred sortiran [5].

Dekartova stabla mogu imati primenu u različitim oblastima računarskih nauka. Kao i ostale vrste balansiranih pretraživačkih stabala i Dekartova stabla se mogu koristiti za efikasno indeksiranje baze podataka. Dekartova stabla se mogu koristiti kao zamena za matrice susedstva koje se koriste za skladištenje grafova. Pošto su društvene mreže predstavljene velikim i gusto povezanim grafovima, Dekartova stabla mogu dosta efikasnije skladištiti podatke o grafu za razliku od matrica susedstva a da pritom mogu podneti rast mreže bez ikakvih posledica [4]. U oblasti istraživanja podataka, u poređenju sa standardnim načinima za otkrivanje pravila pridruživanja među podacima, poput FP-rasta i Apriori algoritma, Dekartova stabla se pokazuju kao dosta efikasnije rešenje [1]. Dekartova stabla su pronašla svoju primenu i u veštačkoj inteligenciji. Naime, umesto tradicionalno korišćenih redova sa prioritetom, Dekartova stabla mogu doprineti efikasnoj implementaciji paralelizovanog algoritma A* [3].

U poređenju sa drugim strukturama binarnih balansiranih pretraživačkih stabala, Dekartovo stablo ima prednost u jednostavnosti. Naime, ne čuva dodatne informacije o balansiranosti, niti izračunava dodatne metapodatke kao što je to slučaj kod AVL stabala [6]. Zbog toga je mnogo jednostavnije konstruisati i održavati Dekartovo stablo, što ga može činiti efikasnijim rešenjem u mnogim primenama.

Literatura

[1] Vinodchandra S. S. Anand H. S. Association rule mining using treap. In *International Journal of Machine Learning and Cybernetics*, page 589–597, Berlin, Heidelberg, 2018. Springer Berlin Heidelberg.

[2] Raimud G. Seidel Cecilia R. Aragon. Randomized search trees. *Computer Science Division, University of California Berkeley*, 1989.

[3] Van Dat Cung and Bertrand Le Cun. An efficient implementation of parallel a*. In Michel Cosnard, Afonso Ferreira, and Joseph Peters, editors, *Parallel and Distributed Computing Theory and Practice*, pages 153–168, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[4] Shalini Batra Dharya Arora. Using treaps for optimization of graph storage. *International Journal of Computer Applications (0975 – 8887)*, 2012.

[5] Algorithms for Competitive Programming. Treap (cartesian tree), 2022.

[6] Vesna Marinković. Materijali sa predavanja - balansirano uredjeno binarno drvo., 2024.

[7] Filip Marić Predrag Janičić. *Programiranje 2 - Osnove programiranja kroz programski jezik C*. Matematički fakultet, Univerzitet u Beogradu, 2024.