UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Anja M. Ivanišević

REŠAVANJE PROBLEMA ISPITIVANJA PRIPADNOSTI INTERVALIMA

master rad

Beograd, 2021.

Mentor:

dr Vesna Marinković, docent
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Miodrag Živković, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Predrag Janičić, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Zahvaljujem se mentoru doc. dr Vesni Marinković na svim savetima, kao i na velikom strpljenju tokom izrade rada. Veliku zahvalnost dugujem porodici i prijateljima za podršku tokom studija.

Naslov master rada: Rešavanje problema ispitivanja pripadnosti intervalima

Rezime: Računarska geometrija je podoblast algoritmike koja se bavi konstrukcijom i analizom algoritama za rešavanje geometrijskih problema. Jedan od problema koji se izučava u okviru računarske geometrije je *problem ispitivanja pripadnosti skupu hiperpravougaonika* (eng. *stabbing queries problem*) koji glasi: za dati skup S hiperpravougaonika čije su stranice paralelne koordinatnim osama i za datu tačku q pronaći sve elemente skupa S koji sadrže tačku q. U ovom radu bavićemo se jednodimenzionom varijantom ovog problema, odnosno *problemom ispitivanja pripadnosti intervalima*.
Problem pripadnosti intervalima nalazi primenu u raznim oblastima: u sistemima za upravljanje bazama podataka, u razvoju algoritama i struktura podataka za eksternu memoriju, u geografskim informacionim sistemima, u rešavanju problema prepoznavanja šablona, itd. U radu će biti predstavljene tri različite strukture podataka za rešavanje ovog problema: intervalno stablo, segmentno stablo i intervalna skip lista. Pored algoritama i implementacije, biće predstavljeno i poređenje efikasnosti ovih pristupa.

Ključne reči: računarstvo, geometrija, algoritmi

Sadržaj

Glava 1

Uvod

Računarska geometrija je podoblast algoritmike koja se bavi konstrukcijom i analizom efikasnih algoritama za rešavanje geometrijskih problema. Geometrijski algoritmi su počeli da se izdvajaju kao zasebna podoblast od ostatka algoritmike krajem 1970-ih godina. Vremenom, ova oblast dobijala je sve više na značaju, delom zbog lepote matematičkih problema koje izučava, a, takođe zbog brojnih primena koje rešenja tih problema nalaze. Geometrijski algoritmi predstavljaju osnov za računarsku grafiku, imaju primene u geografskim informacionim sistemima, robotici, računarskoj viziji, bazama podataka, itd.

Jedan od problema koji se izučava u sklopu računarske geometrije je *problem ispitivanja pripadnosti skupu hiperpravougaonika* (eng. *stabbing queries problem*) koji glasi: za dati skup S hiperpravougaonika [1] čije su stranice paralelne koordinatnim osama i za datu tačku upita q potrebno je pronaći sve elemente skupa S koji sadrže tačku q [12]. U ovom radu bavićemo se jednodimenzionom varijantom ovog problema, odnosno *problemom ispitivanja pripadnosti intervalima*. U nastavku rada biće prikazano zašto je ovaj problem značajan, kao i neka od efikasnih rešenja ovog problema.

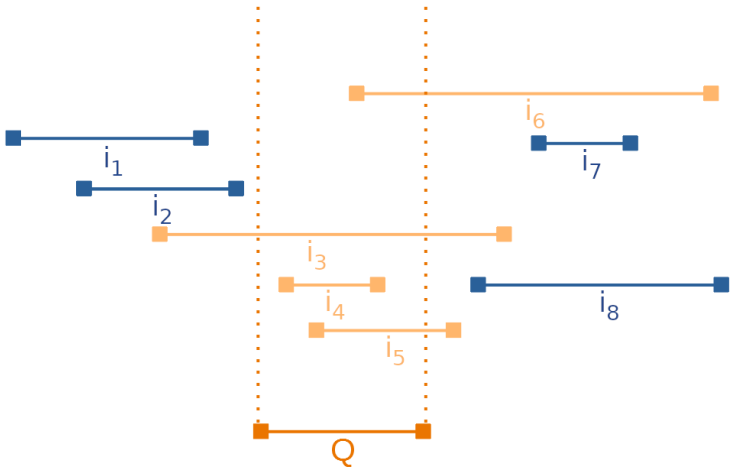
1.1 Motivacija

Problem ispitivanja pripadnosti intervalima je dugo poznat problem koji privlači pažnju naučnika iz oblasti računarske geometrije zbog svojih brojnih primena. Neke od najznačajnijih primena su u sistemima za upravljanje bazama podataka, u geo-

[1] Hiperpravougaonik je generalizacija pravougaonika na više dimenzija. Formalno je definisan kao Dekartov proizvod ortogonalnih intervala.

grafskim informacionim sistemima (GIS), u razvoju algoritama i struktura podataka za eksternu memoriju, u rešavanju problema prepoznavanja šablona (eng. *pattern matching*), itd.

Zbog svoje jednostavne reprezentacije, intervali se često koriste u sistemima za upravljanje bazama podataka. Uz pomoć njih, na primer, možemo predstaviti ograničenja atributa u bazama podataka ili period validnosti atributa u privremenim bazama podataka. Zbog toga se u ovim sistemima javlja potreba za strukturom podataka za čuvanje intervala sa efikasnim operacijama pretrage, umetanja, ažuriranja i brisanja intervala. Ovaj problem pripada oblasti računarske geometrije, a naziva se *problem upravljanja intervalima* (eng. *interval management problem*). Njegova definicija glasi: za dati skup intervala S i dati interval upita $Q = [q_1, q_2]$, odrediti sve intervale skupa S koji presecaju interval Q [4]. Problem upravljanja intervalima ilustrovan je na slici 1.1. Neka je dat skup intervala $i_1, \ldots, i_8$ i interval upita Q. Intervali koji se seku sa Q su i_3, i_4, i_5 i i_6. Možemo primetiti da ovaj problem veoma liči na problem ispitivanja pripadnosti intervalima, jedino što upite ne postavljamo jednom tačkom već čitavim intervalom. Na osnovu sličnosti ovih problema možemo pretpostaviti da oni pripadaju istoj grupi problema, te da su i algoritmi i strukture podataka koji se koriste za njihovo rešavanje slični. Ovaj problem ima široku primenu u sistemima za upravljanje bazama podataka, uključujući prostorne, privremene i objektno orijentisane baze podataka [4].



Slika 1.1: Ilustracija problema upravljanja intervalima

Često je količina podataka sa kojom je potrebno manipulisati u aplikacijama

prevelika da bi bila smeštena u glavnu memoriju, te je neophodno da podaci budu smešteni na disku. Primer ovakvih aplikacija su geografski informacioni sistemi, koji barataju velikom količinom prostornih podataka. Ti podaci mogu biti tačke, duži ili poligoni, a neki od primera upita koji bi se izvršavali nad takvom bazom podataka su pronalaženje svih regiona koji presecaju dati region, pronalaženje svih objekata koji sadrže zadatu tačku, itd. Zbog velikih razlika u performansama između interne i eksterne memorije i potrebe za čestom komunikacijom između njih, često dolazi do zagušenja ili uskog grla. Kako procesori postaju sve brži, naročito sa paralelizacijom procesa, problem zagušenja i spore komunikacije između interne i eksterne memorije postaje sve primetniji. Problem ispitivanja pripadnosti intervalima igra veliku ulogu u pronalaženju efikasnih algoritama i struktura podataka za rad sa eksternom memorijom [13].

Još jedna od primena problema ispitivanja pripadnosti intervalima je u rešavanju problema prepoznavanja šablona. Problem prepoznavanja šablona se sastoji od pronalaženja jednog ili većeg broja pojavljivanja šablona P dužine m u tekstu T dužine n, gde važi da je $m \leq n$. Ispostavlja se da se problem prepoznavanja šablona može svesti na jednodimenzioni (1D) ili dvodimenzioni (2D) problem ispitivanja pripadnosti skupu geometrijskih objekata. Ideja je da se simulira algoritam za pretraživanje stringova poznat kao *AC automatizacija* (eng. *Aho-Corasick automation algorithm*). Ovaj algoritam na osnovu konačnog skupa stringova formira strukturu podataka nalik prefiksnom stablu, na osnovu koga se posle formira *konačni automat* (eng. *finite automaton*). Problem prepoznavanja šablona možemo rešiti tako što ćemo u svakoj iteraciji pročitati prvo određen broj karaktera, naći sva poklapanja šablona koji se završavaju jednim od tih karaktera, i na kraju skočiti na stanje u koje bismo došli AC automatizacijom. Pronalaženje svih poklapanja šablona se može svesti na problem 2D ispitivanja pripadnosti skupu geometrijskih objekata, dok se skok na naredno stanje AC automatizacije može svesti na problem 1D ispitivanja pripadnosti skupu geometrijskih objekata, odnosno problem ispitivanja pripadnosti intervalima [5].

Glava 2

Algoritmi za rešavanje problema ispitivanja pripadnosti intervalima

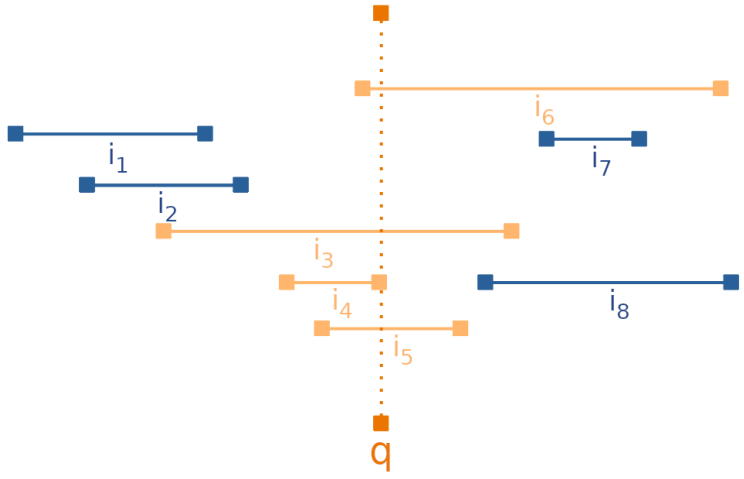
Problem ispitivanja pripadnosti intervalima je problem pronalaženja svih intervala iz datog skupa intervala koji sadrže zadatu tačku. Formulacija problema za dimenziju jedan glasi:[1]

Neka je dat skup intervala tačaka $S = \{I_1, I_2, \ldots, I_n\}$, $I_i = (l_i, r_i)$, $l_i \leq r_i$ *za* $i \in [1, n]$. *Za datu tačku upita* q, *pronaći podskup skupa intervala* S *koji sadrže tačku* q. *[9].*

Intervali će biti predstavljeni kao parovi tačaka. Biće korišćene različite vrste zagrada da bismo razlikovali da li interval sadrži krajnje tačke intervala. Ako interval sadrži krajnju tačku u zapisu intervala biće korišćene uglaste zagrade [], u suprotnom će biti korišćene oble zagrade (). U zapisu intervala koji kao jednu svoju granicu imaju pozitivnu ili negativnu beskonačnost, beskonačne granice intervala biće označene sa $+\infty$ i $-\infty$. Primeri intervala su $(1, 13), [8, +\infty), (-17, 7]$.

Na slici 2.1 prikazana je ilustracija problema ispitivanja pripadnosti intervalima. Skup intervala iz primera je $i_1, \ldots, i_8$, a tačka upita je q. Intervali koji sadrže tačku q su i_3, i_4, i_5 i i_6.

[1] U reprezentaciji intervala I_i korišćene su oble zagrade, međutim, intervali mogu biti i otvoreni i zatvoreni.



Slika 2.1: Ilustracija problema ispitivanja pripadnosti intervalima

2.1 Pregled postojećih rešenja

Više različitih struktura podataka može biti iskorišćeno za rešavanje problema pripadnosti intervalima. Najjednostavnije rešenje je korišćenjem metode grube sile, koju možemo realizovati korišćenjem dinamičkog niza. Najpre je potrebno smestiti n intervala u niz. Dalje je, jednim prolazom kroz niz, za svaki interval potrebno proveriti da li on sadrži zadatu tačku upita q. Ukoliko sadrži, interval se smešta u rezultujući skup. Vremenska i prostorna složenost ovog algoritma odgovaraju broju elemenata niza, odnosno broju intervala i iznose $O(n)$. U nastavku je prikazan pseudokod ovog algoritma.

```
# R - dinamicki niz intervala, q - tacka pretrage
# rezultat - skup markera intervala koji sadrze q
def pretraziDinamickiNiz(R, q, rezultat):
    for i in range(0, len(R), 1):
        if intervalSadrziTacku(R[i], q):
            rezultat.add(R[i].marker)
```

Ovim smo utvrdili gornju granicu vremenske složenosti algoritama za rešavanje ovog problema, i od narednih naprednijih algoritama očekujemo da imaju bolju vremensku složenost od $O(n)$.

Za rešavanje problema ispitivanja pripadnosti intervalima mogu se iskoristiti različita binarna stabla pretrage. Neka od njih su intervalna stabla i segmentna stabla. Pored struktura podataka koje su zasnovane na binarnim stablima pretrage,

postoji i algoritam koji koristi strukturu skip liste koja je posebno implementirana za rešavanje ovog problema. Takva skip lista se naziva intervalna skip lista.

Od početnog skupa intervala očekujemo da će se oni u određenoj meri preklapati. Jedna ideja koja se javlja jeste da intervale početnog skupa izdelimo u elementarne intervale. Elementarni intervali su intervali nastali presecanjem brojevne prave na mestima krajnjih granica intervala početnog skupa. Ovako konstruisan skup elementarnih intervala može biti smešten u strukturu podataka, a rešavanje problema ispitivanja pripadnosti intervalima bi se svelo na traženje elementarnog intervala koji sadrži zadatu tačku pretrage. Ova ideja uprošćava pronalaženje podskupa intervala koji sadrže tačku pretrage i biće korišćena u dve strukture podataka koje će biti prikazane u radu, a to su segmentno stablo i intervalna skip lista.

2.2 Intervalna stabla

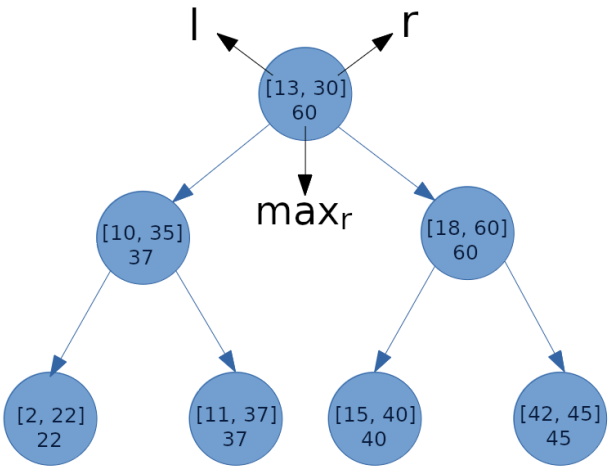
Problem ispitivanja pripadnosti intervalima je pre svega problem pretrage. Pitanje koje se prirodno nameće je da li bismo mogli da napravimo nekakvo uređenje u datom skupu intervala i da za rešavanje problema iskoristimo balansirano binarno stablo pretrage, s obzirom da znamo da je u slučaju balansiranog binarnog stabla složenost pretrage $O(\log n)$. Pošto su intervali određeni svojom levom i desnom granicom, sortiranjem samo po levim ili samo po desnim granicama intervala, nećemo moći da odsecamo grane prilikom pretrage, pa složenost pretrage neće biti manja od $O(n)$. Ako bismo pak želeli da napravimo dva stabla, jedno sortirano po levim granicama, a drugo sortirano po desnim granicama, mogli bismo da odvojeno pretražujemo oba stabla. Međutim, na kraju bi bilo neophodno spojiti rezultate te dve pretrage, što bi na kraju zahtevalo vremensku složenost $O(n)$ u najgorem slučaju, tako da ni taj pristup ne bi dao bolje rešenje od naivnog pristupa.

Struktura podataka koja je nastala na bazi ove ideje, a prevazilazi pomenute probleme je *intervalno stablo* (eng. *interval tree*). Intervalno stablo je vrsta balansiranog binarnog stabla pretrage kod koga svaki čvor sadrži sledeće informacije:

- interval koji je predstavljen kao par tačaka $I = (l, r)$, gde je l leva granica intervala, a r desna granica intervala;
- maksimalnu vrednost svih desnih granica intervala koji se nalaze u podstablu sa korenom u posmatranom čvoru - max_r [6, 12].

Pošto se radi o vrsti balansiranog binarnog stabla pretrage, potrebno je održavati uređenje intervala. Intervali će biti sortirani po njihovim levim granicama, i važiće naredni uslov: za svaki čvor N koji sadrži interval $I_n = (l_n, r_n)$, vrednost njegove leve granice l_n biće veća od svih vrednosti levih granica intervala čvorova levog podstabla stabla sa korenom u čvoru N, a manja ili jednaka od svih vrednosti levih granica intervala čvorova desnog podstabla stabla sa korenom u čvoru N.

Primer 1 *Na slici 2.2 dat je primer intervalnog stabla. Na primeru sa slike možemo primetiti da se čvor koji sadrži interval* $[10, 35]$ *nalazi u levom podstablu u odnosu na koren, jer je njegova leva granica manja od leve granice čvora koji odgovara korenu, odnosno važi* $10 < 13$. *Takođe, možemo primetiti da je* max_r *vrednost levog deteta korena jednaka* 37, *što odgovara maksimumu desnih granica intervala čvorova u podstablu sa korenom u tom čvoru, odnosno važi da je maksimum brojeva* 22, 35 *i* 37 *jednak broju* 37.



Slika 2.2: Struktura intervalnog stabla. l = leva granica intervala, r = desna granica intervala, max_r = maksimalna vrednost desnih granica intervala koji se nalaze u podstablu stabla sa korenom u posmatranom čvoru

Pretraga intervalnog stabla

Razmotrimo kako za opisanu strukturu intervalnog stabla na osnovu zadate tačke upita q možemo pretražiti intervalno stablo i vratiti sve intervale koji se nalaze

u stablu i sadrže tačku q. Algoritam pretrage se sastoji iz tri dela: najpre vršimo proveru za interval u čvoru u kom se trenutno nalazimo, pa onda rekurzivno, ako ima potrebe, pokrećemo pretragu za levo i desno podstablo. Na osnovu vrednosti promenljive max_r za dati čvor možemo zaključiti ima li potrebe da pozivamo funkciju za levo podstablo datog čvora. Naime, u slučaju da je koordinata tačke q veća od vrednosti max_r levog deteta čvora, to znači da je ona veća od desnih granica svih intervala u levom podstablu. Kako leve granice intervala moraju biti manje ili jednake desnim granicama intervala, koordinata tačke q mora biti veća i od svih levih granica intervala u levom podstablu. Samim tim možemo zaključiti da nijedan interval iz levog podstabla ne može da sadrži tačku q, te da nema potrebe da pretražujemo levo podstablo. Na ovaj način, korišćenjem promenljive max_r, izbegli smo potrebu da stablo bude sortirano po desnim granicama intervala. Pretraga se pokreće za desno podstablo samo ako je koordinata tačke q veća ili jednaka levoj granici intervala čvora u kom se trenutno nalazimo. Naime, ako je koordinata tačke q manja od leve granice posmatranog intervala to znači da je ona manja od svih levih granica intervala u desnom podstablu, jer je stablo sortirano po levim granicama intervala. Ako ovo važi, znamo da nijedan interval u desnom podstablu ne može sadržati tačku q pa nema potrebe da ga pretražujemo. Narednim pseudokodom opisan je mehanizam pretrage intervalnog stabla.

```
# R - koren intervalnog stabla, q - tacka pretrage
# rezultat - skup markera intervala koji sadrze q
def pretraziIntervalnoStablo(R, q, rezultat):
    I = R.interval
    if intervalSadrziTacku(I, q):
        rezultat.add(I)
    if R.levoDete.max_r >= q:
        pretraziIntervalnoStablo(R.levoDete, q, rezultat)
    if q >= I.levaGranica:
        pretraziIntervalnoStablo(R.desnoDete, q, rezultat)
```

Pošto je intervalno stablo balansirano, njegova visina je reda $O(\log n)$, gde je n broj intervala. Vremenska složenost najgoreg slučaja pronalaženja jednog intervala koji sadrži tačku upita q je $O(\log n)$ [12]. Za rešavanje problema ispitivanja pripadnosti intervalima, potrebno je da kao rezultat vratimo sve intervale koji sadrže tačku upita q. Pošto je intervalno stablo sortirano po levim granicama intervala i pošto se pored toga čuva informacija o maksimumu desnih granica intervala podstabla svakog čvora, ove informacije koristimo da sečemo pretragu ukoliko smo sigurni da se u podstablu ne nalazi nijedan interval koji sadrži tačku q. Na ovaj način osiguravamo

optimalnu vremensku složenost i dolazimo do konačne vremenske složenosti pretrage najgoreg slučaja koja iznosi $O(\log n + k)$, gde je k broj intervala koji sadrži tačku q [12]. U najgorem slučaju k će biti jednako n, te će složenost pretrage biti jednaka $O(n)$, međutim, u praksi je ovakav slučaj malo verovatan.

Umetanje elementa u intervalno stablo

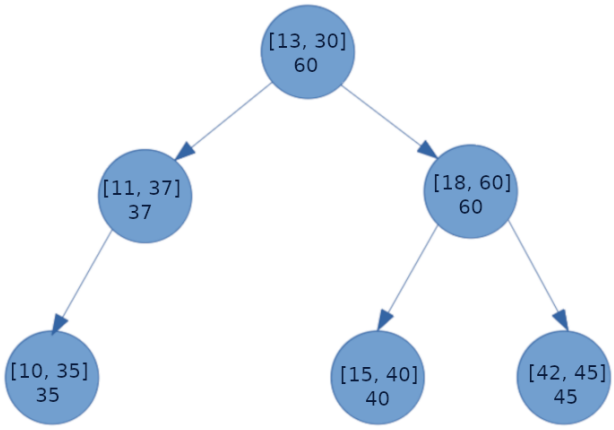
Operacija umetanja intervala u intervalno stablo nije ništa drugo nego operacija umetanja u balansirano binarno stablo koje je sortirano po levim granicama intervala: novi element se uvek umeće kao list na odgovarajuće mesto u stablu. Jedina razlika je u tome što je dodatno neophodno ažurirati i vrednost pomoćne promenljive max_r čvorovima u stablu. To se postiže jednostavno, proverom da li je desna granica intervala koji umećemo veća od vrednosti promenljive max_r čvora. Ako jeste, potrebno je ažurirati vrednost max_r, i ovu proveru treba izvesti za sve čvorove kroz koje ćemo proći prilikom umetanja. U nastavku je prikazan pseudokod algoritma za umetanje novog intervala u intervalno stablo.

```
# R - koren intervalnog stabla
# I - interval koji zelimo da umetnemo u intervalno stablo
def umetniNoviInterval(R, I):
    if R is None:
        R = Cvor(I)
        return R
    if I.levaGranica < R.interval.levaGranica:
        R.levoDete = umetniNoviInterval(R.levoDete, I)
    else:
        R.desnoDete = umetniNoviInterval(R.desnoDete, I)
    if I.desnaGranica > R.max_r:
        R.max_r = I.desnaGranica
```

Ako nakon umetanja novog intervala u intervalno stablo dođemo u situaciju da stablo više nije balansirano, potrebno ga je balansirati. Realizacija operacije balansiranja zavisi od tipa balansiranog binarnog stabla na osnovu kog je konstruisano intervalno stablo, pa i složenost ove operacije može varirati. Detaljnije ćemo opisati proces balansiranja intervalnog stabla koje je konstruisano kao *AVL stablo* (eng. *AVL tree*). AVL stablo je vrsta balansiranog binarnog stabla pretrage kod kojeg za svaki čvor važi da apsolutna vrednost razlike visina levog i desnog podstabla nije veća od 1 [10]. Balansiranje se u slučaju AVL stabla vrši tako što se pronalazi *kritični čvor*. Kritični čvor je čvor najmanje visine, odnosno najniži čvor, za koji važi da se visina levog i desnog podstabla stabla sa korenom u tom čvoru razlikuju za više od 1. Nakon toga se nad kritičnim čvorom vrši neka od četiri tipa rotacija: LL (leva-leva),

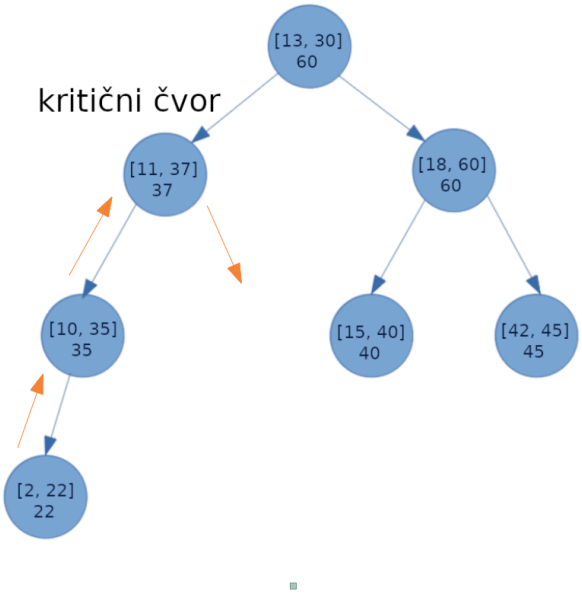
RR (desna-desna), LR (leva-desna) ili RL (desna-leva rotacija), u zavisnosti od toga u koje podstablo stabla sa korenom u kritičnom čvoru smo umetnuli novi element [10].

Primer 2 *Na narednim slikama možemo videti primer umetanja novog intervala u intervalno stablo. Slika 2.3 predstavlja početno stanje intervalnog stabla. Nakon umetanja intervala* $[2, 22]$ *dobijamo strukturu intervalnog stabla sa slike 2.4. Stablo više nije balansirano i kritični čvor stabla je čvor označen intervalom* $[11, 37]$*. Potrebno je sprovesti LL rotaciju i izbalansirati stablo. Nakon LL rotacije dobijamo ponovo balansirano stablo i ono izgleda kao na slici 2.5. Posle balansiranja stabla potrebno je ažurirati vrednost* max_r *za čvorove koji su promenili pozicije u stablu. Na primeru sa slike 2.5 ti čvorovi su* $[2, 22]$*,* $[10, 35]$ *i* $[11, 37]$ *i možemo primetiti da je čvor* $[10, 35]$ *promenio vrednost* max_r *sa* 35 *na* 37*.*

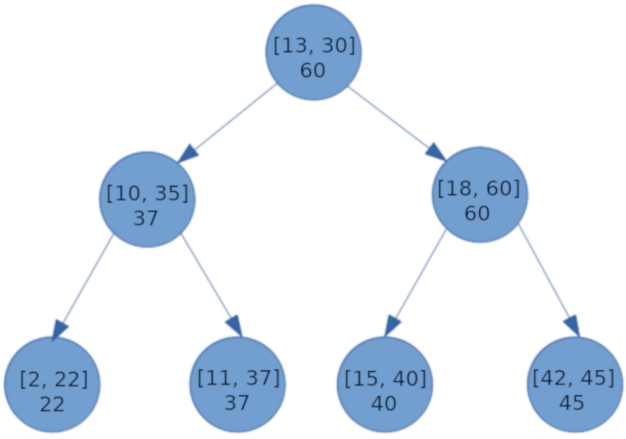


Slika 2.3: Intervalno stablo pre umetanja novog intervala

Vremenska složenost najgoreg slučaja za svaki od ovih tipova rotacija je konstantna i iznosi $O(1)$. U slučaju intervalnog stabla koje je konstruisano na osnovu AVL stabla, potrebno je nakon svake rotacije ažurirati i vrednosti pomoćnih promenljivih max_r. Ovaj korak zahteva ažuriranje samo onih čvorova koje smo rotacijama pomerali, te vremenska složenost rotacija ostaje ista i iznosi $O(1)$. Dakle, ukupna



Slika 2.4: Intervalno stablo nakon umetanja novog intervala $[2, 22]$



Slika 2.5: Intervalno stablo nakon LL rotacije

vremenska složenost operacije umetanja elementa u intervalno stablo koje sadrži n elemenata iznosi $O(\log n)$.

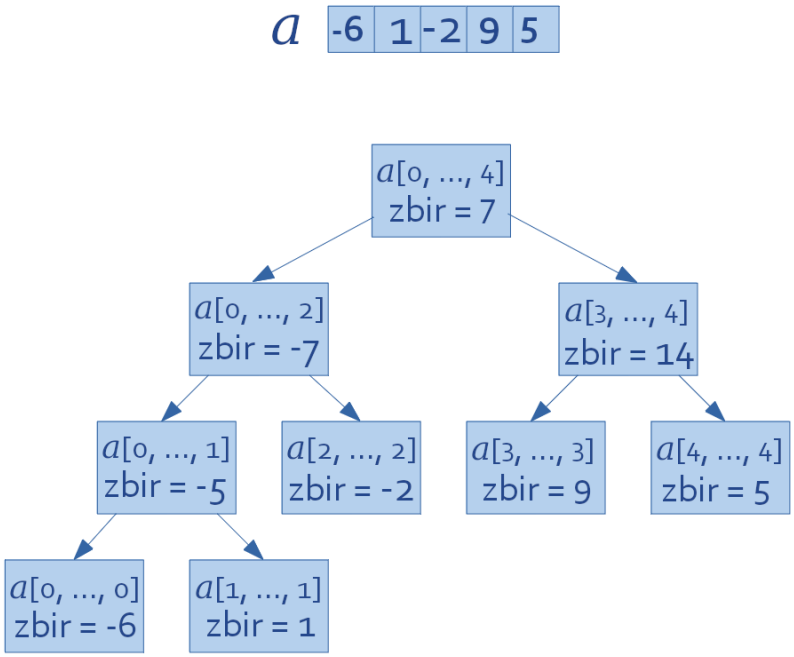
2.3 Segmentna stabla

Segmentno stablo (eng. *segment tree*) je još jedna struktura podataka zasnovana na binarnim stablima kojom je moguće rešiti problem ispitivanja pripadnosti intervalima. Osnovna svrha segmentnih stabala je efikasno izvršavanje upita raspona, poput pronalaženja zbira uzastopnih elemenata datog niza brojeva, pronalaženja najmanjeg broja u podnizu uzastopnih elemenata datog niza brojeva, itd [1].

Objasnimo strukturu segmentnog stabla na primeru problema efikasnog određivanja zbira uzastopnih elemenata datog niza brojeva. Neka je zadat niz a koji sadrži n brojeva. Konstruisaćemo binarno stablo u čijem će korenu biti smeštena vrednost zbira svih elemenata niza $a[0, \ldots, n-1]$. U sledećem koraku podelićemo niz a na dva podniza sa jednakim brojem elemenata do na jedan element $a[0, \ldots, n/2]$ i $a[n/2+1, \ldots, n-1]$. Zbirovi ova dva podniza elemenata biće smešteni u stablo kao levo i desno dete korena stabla. Postupak nastavljamo rekurzivno: za svaki unutrašnji čvor delimo njegov niz elemenata na dva jednaka dela i smeštamo zbirove dobijenih podnizova kao decu tog čvora segmentnog stabla, sve dok ne dođemo do nizova sačinjenih od pojedinačnih elemenata. Takvi elementi biće smešteni kao listovi segmentnog stabla.

Primer 3 *Na slici 2.6 prikazan je primer segmentnog stabla za efikasno izvršavanje zbira uzastopnih elemenata niza brojeva. Niz a je dužine* 5 *i sadrži elemente* $-6, 1, -2, 9, 5$. *Koren segmentnog stabla odgovara celom nizu a i sadrži vrednost jednaku zbiru svih brojeva iz niza. Slično, svi čvorovi na slici su označeni odgovarajućim podnizom niza a i u sebi čuvaju zbir tog podniza. Ilustrujmo kako bismo izračunali zbir podniza* $a[2, \ldots, 4]$ *pomoću ovako konstruisanog segmentnog stabla. Krećemo od korena stabla i proveravamo da li treba da posetimo njegovo levo i desno dete. Levo dete sadrži podniz* $a[0, \ldots, 2]$. *Pošto ovaj podniz ima presek sa traženim podnizom* $a[2, \ldots, 4]$ *(element* $a[2]$*) rekurzivno pozivamo funkciju za taj čvor i proveravamo njegovu decu. Levo dete sadrži podniz* $a[0, \ldots, 1]$ *koji nema preklapanja sa podnizom čiji zbir računamo, dok desno dete sadrži podniz* $a[2, \ldots, 2]$, *odnosno samo element* $a[2]$, *pa funkciju pozivamo samo za desno dete. Došli smo do lista stabla i njegovu vrednost* -2 *dodajemo u rezultujući zbir. Ispitujemo desno dete korena stabla koje sadrži podniz* $a[3, \ldots, 4]$. *Ovaj podniz u celosti pripada traženom podnizu* $a[2, \ldots, 4]$, *pa nema potrebe da ispitujemo dalje i u rezultujući zbir dodajemo njegovu vrednost* 14. *Konačni zbir iznosi* $-2 + 14 = 12$.

Vremenska složenost najgoreg slučaja za izvršavanje ovakvih upita nad segmentnim stablom iznosi $O(\log n)$. Ova složenost sledi na osnovu toga što imamo $O(\log n)$ nivoa u segmentnom stablu, a u svakom nivou ćemo posetiti najviše četiri čvora. Sa izvršavanjem upita krećemo od korena segmentnog stabla i u narednom nivou posetićemo najviše dva čvora, jer koren ima najviše dva deteta, pa je tvrđenje u ovom slučaju ispunjeno. U narednom nivou iz dva čvora možemo otići u najviše četiri čvora pa će i u tom slučaju tvrđenje biti ispunjeno. Pretpostavimo da smo u jednom nivou posetili tri ili četiri čvora. Pošto želimo da izračunamo zbir uzastopnih elemenata niza, to znači da će središnji čvorovi biti već pokriveni prethodnim nivoom i neće biti potrebno rekurzivno pozivati funkciju za te čvorove. Jedino će krajnji čvorovi nivoa, odnosno krajnji levi i krajnji desni čvor, potencijalno zahtevati rekurzivne pozive. Dakle, dolazimo do zaključka da iz ta dva čvora možemo imati najviše četiri rekurzivna poziva i time smo pokazali da prethodno tvrđenje važi. Zaključujemo da je vremenska složenost ovog algoritma u najgorem slučaju jednaka $O(4 \log n)$, što dalje iznosi $O(\log n)$ [12]. U nastavku teksta razmotrićemo na koji način segmentno stablo može biti prilagođeno za rešavanje problema ispitivanja pripadnosti intervalima.



Slika 2.6: Grafički prikaz segmentnog stabla za izvršavanje upita zbira uzastopnih elemenata niza brojeva

Neka je dat skup intervala $S = \{(l_1, r_1), (l_2, r_2), \ldots, (l_n, r_n)\}$. Za dati skup S neka je $p_1, \ldots, p_m$ sortirani niz levih i desnih granica intervala skupa S bez ponavljanja tačaka. Zamislimo da brojevnu pravu presečemo tako dobijenim tačkama $p_1, \ldots, p_m$. Na taj način dobićemo nove intervale koje ćemo nazvati *elementarni intervali*:

$$(-\infty, p_1), [p_1, p_1], (p_1, p_2), [p_2, p_2], \ldots, [p_m, p_m], (p_m, +\infty)$$

Skup elementarnih intervala čine otvoreni intervali od tačke p_i do p_{i+1} i zatvoreni intervali pojedinačnih tačaka p_i. Važi da $i \in [0, m+1]$, da je $p_0 = -\infty$ i $p_{m+1} = +\infty$. Ovakvom konstrukcijom elementarnih intervala želimo da postignemo fleksibilnost da podržimo i otvorene i zatvorene intervale. Na osnovu skupa elementarnih intervala možemo konstruisati binarno stablo.

Primer 4 *Na slici 2.7 ilustrovana je struktura segmentnog stabla na primeru skupa intervala* $S = \{(-\infty, 1], [1, 3], [3, 7], [6, 11], [6, 22], [15, +\infty)\}$. *Na osnovu levih i desnih granica intervala iz* S *konstruišemo skup elementarnih intervala* $P = \{(-\infty, 1), [1, 1], (1, 3), [3, 3], (3, 6), [6, 6], (6, 7), [7, 7], (7, 11), [11, 11], (11, 15), [15, 15], (15, 22), [22, 22], (22, +\infty)\}$. *Sortirani skup* P *elementarnih intervala biće smešten u listovima segmentnog stabla, dok će intervali u unutrašnjim čvorovima segmentnog stabla predstavljati uniju intervala levog i desnog deteta.*

Potrebno je uspostaviti vezu između elementarnih intervala i početnog skupa intervala S. Jasno je da proizvoljni interval početnog skupa intervala, recimo (l_i, r_i), može da se sastoji od više elementarnih intervala koje možemo označiti sa $P_j, \ldots, P_k$. Ako bismo oznake intervala skupa S smeštali u listove segmentnog stabla, oznaka intervala (l_i, r_i) iz S bi bila smeštena u $k - j + 1$ listova, što bi u najgorem slučaju bilo $O(m)$ oznaka za jedan interval, a $O(n \cdot m)$ oznaka za sve intervale skupa S, čime bismo povećali prostornu složenost ove strukture podataka. Ispostavlja se da možemo izbeći ovakav skok u prostornoj složenosti time što informacije o intervalima nećemo nužno čuvati u listovima stabla, nego u najvišim čvorovima stabla čije podstablo pokriva taj interval [12]. Segmentno stablo konstruisano za potrebe rešavanja problema ispitivanja pripadnosti intervalima je statička struktura podataka, što znači da ne podržava promene nakon što je inicijalno konstruisana.

Primer 5 *Razmotrimo primer segmentnog stabla sa slike 2.7 i interval* $S_3 = [3, 7]$. *Oznake intervala* S_3 *čuvaju u listu koji pokriva interval* $[3, 3]$ *i unutrašnjem čvoru*

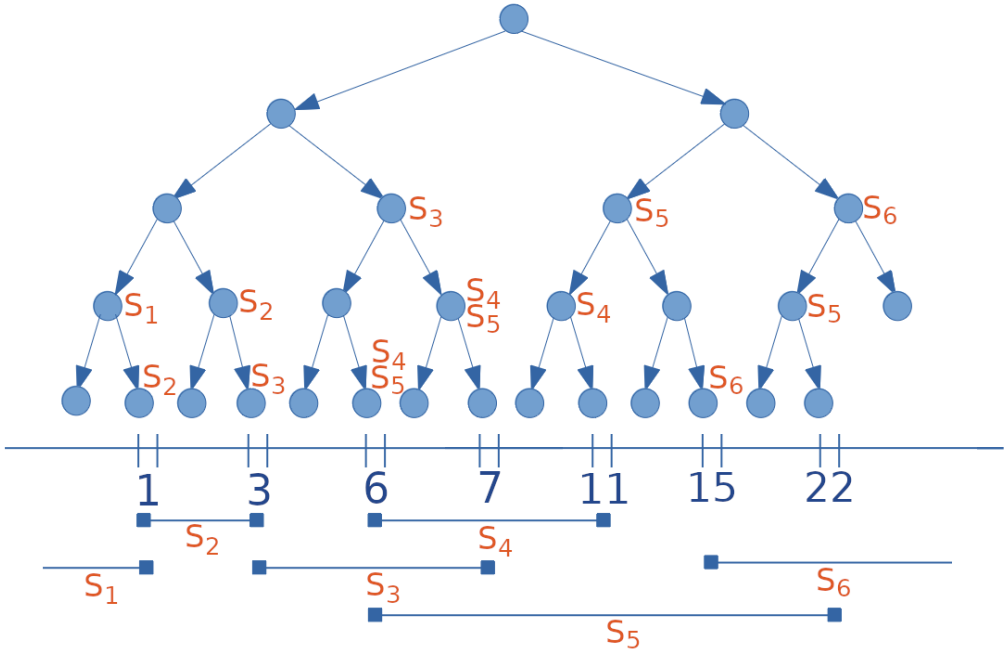
koji pokriva interval $(3, 7]$. *Možemo primetiti da je oznaka intervala* S_3 *smeštena u čvor sa intervalom* $(3, 7]$, *a ne u neki čvor u tom podstablu jer je ovo najviši čvor koji pokriva odgovarajući interval.*

S obzirom na to da je segmentno stablo u osnovi balansirano binarno stablo, možemo zaključiti da njegova visina iznosi $O(\log n)$. Važi sledeće tvrđenje: za svaki interval iz S postojaće najviše dve oznake po nivou segmentnog stabla. Dokažimo ovo tvrđenje. Pretpostavimo da se u segmentnom stablu na istom nivou redom nalaze tri čvora v_1, v_2 i v_3. Neka se interval $[x, y]$ proteže od leve granice intervala čvora v_1 sve do desne granice intervala čvora v_3. Pošto se čvor v_2 nalazi između v_1 i v_3, roditelj čvora v_2 mora takođe biti unutar intervala $[x, y]$. Pošto se markeri intervala čuvaju u najvišim čvorovima čije podstablo pokriva odgovarajući interval, to znači da se oznaka intervala $[x, y]$ ne može nalaziti na čvoru v_2. Dolazimo do zaključka da svaki interval može imati najviše dve oznake po nivou segmentnog stabla. Pošto segmentno stablo ima $O(\log n)$ nivoa, ukupna prostorna složenost ovako osmišljene strukture podataka iznosiće $O(n \log n)$ [12].

Pretraga segmentnog stabla

Prema konstrukciji, u listovima segmentnog stabla nalaze se sortirani elementarni intervali. Pored toga, segmentno stablo u određenim čvorovima sadrži informacije o intervalima iz početnog skupa S. Da bismo pretražili segmentno stablo i pronašli sve intervale iz skupa S koji sadrže zadatu tačku q, dovoljno je da prođemo kroz segmentno stablo do elementarnog intervala koji sadrži tačku q. Tokom pretrage prolazićemo od korena do lista, kroz unutrašnje čvorove stabla i pamtićemo markere intervala koje ti čvorovi čuvaju. Naime, na osnovu konstrukcije segmentnog stabla znamo da će svi čvorovi na putu ka elementarnom intervalu koji sadrži tačku q, sadržati tačku q. Takođe, važi da ako interval koji pokriva neki čvor ne sadrži tačku q, neće je sadržati nijedan interval njegovog podstabla.

Primer 6 *Razmotrimo još jednom primer segmentnog stabla sa slike 2.7 i kako bi izgledala pretraga ovog segmentnog stabla ako je data tačka upita* $q = 7$. *Krećemo od korena i pošto njemu odgovarajući interval pokriva celu brojevnu pravu pozivamo funkciju za oba deteta. Interval levog deteta korena odgovara intervalu* $(-\infty, 7]$, *pa pošto on sadrži tačku* 7 *pozivamo pretragu za levo podstablo. Interval desnog deteta*



Slika 2.7: Grafički prikaz segmentnog stabla za rešavanje problema ispitivanja pripadnosti intervalima

korena odgovara intervalu $(7, +\infty)$ *i pošto on ne sadrži tačku* 7 *nećemo pozivati pretragu za desno podstablo. Iz levog podstabla korena po istom principu idemo desno i dolazimo do unutrašnjeg čvora koji sadrži marker intervala* S_3*, koji dodajemo u rezultujući skup. Pozivamo funkciju za levo i desno dete posmatranog čvora. Levo dete odgovara intervalu* $(3, 6]$*, a desno intervalu* $(6, 7]$*. Nastavljamo pretragu samo za desno dete jer važi da* $7 \in (6, 7]$*. Ponovo dodajemo markere intervala koji se nalaze u trenutnom čvoru, a to su* S_4 *i* S_5*. Na kraju pretrage stižemo do listova segmentnog stabla i nalazimo elementarni interval koji sadrži tačku* 7*, a to je interval* $[7, 7]$*. Rezultat pretrage je skup intervala* $\{S_3, S_4, S_5\}$*, odnosno skup* $\{[3, 7], [6, 11], [6, 22]\}$*.*

Vremenska složenost određivanja svih intervala koji sadrže tačku upita q iznosi $O(\log n + k)$, gde je k broj intervala u rezultatu upita [12]. U nastavku je prikazan pseudokod pretrage segmentnog stabla koje sadrži informacije o datom skupu intervala.

```
1 # R - koren segmentnog stabla, q - tacka pretrage
2 # rezultat - skup markera intervala koji sadrze q
```

```
def pretraziSegmentnoStablo(R, q, rezultat):
    if R is None:
        return
    if intervalSadrziTacku(R.interval, q):
        rezultat.addAll(R.markeri)
        pretraziSegmentnoStablo(R.levoDete, q, rezultat)
        pretraziSegmentnoStablo(R.desnoDete, q, rezultat)
```

Konstrukcija segmentnog stabla

Algoritam konstrukcije segmentnog stabla možemo podeliti u tri celine:

- konstrukcija skupa elementarnih intervala na osnovu početnog skupa S;
- konstrukcija segmentnog stabla na osnovu skupa elementarnih intervala;
- dodavanje markera intervala skupa S u segmentno stablo.

Za prvi od navedenih koraka potrebno je napraviti sortiran skup krajnjih tačaka intervala i nakon toga na osnovu njega konstruisati redom elementarne intervale. Na osnovu tako dobijenog skupa elementarnih intervala treba konstruisati segmentno stablo, u čijim će se listovima nalaziti elementarni intervali. Kao poslednji korak potrebno je čvorovima konstruisanog segmentnog stabla dodati oznake intervala početnog skupa S. Za svaki interval I iz skupa S prolazićemo kroz segmentno stablo od korena ka listovima i za svaki čvor V na tom putu proveravaćemo da li interval I sadrži interval koji pokriva čvor V. Ako je ovaj uslov ispunjen to je znak da možemo da zaustavimo pretragu i dodamo marker intervala I u skup markera čvora V. U suprotnom vršimo istu proveru za levo i desno dete čvora V [12]. U nastavku je dat pseudokod algoritma konstrukcije segmentnog stabla.

```
# S - pocetni skup intervala
def konstruisiSegmentnoStablo(S):
    E = napraviElementarneIntervale(S)
    l = 1
    d = velicina(E)
    R = napraviStablo(E, l, d)
    for I in S:
        dodajMarkereIntervala(R, I)
    return R
```

```
# E - sortirani niz elementarnih intervala,
# l, d - brojevi koji oznacavaju levu i desnu granicu niza E
def napraviStablo(E, l, d):
```

```
    if l > d:
        return None
    if l == d:
        return Cvor(E[l])
    i = l + (d - l) // 2
    noviInterval = Interval(E[l].levoDete, E[d - 1].desnoDete)
    R = Cvor(noviInterval)
    R.levoDete = napraviStablo(E, l, i)
    R.desnoDete = napraviStablo(E, i, d)
    return R
```

```
# R - koren segmentnog stabla, I - interval
def dodajMarkereIntervala(R, I):
    if R is None:
        return
    if intervalSadrziDrugiInterval(I, R.interval):
        R.markeri.add(I.marker)
        return
    if R.levoDete is not None
    and intervaliImajuPresek(R.levoDete.interval, I):
        dodajMarkereIntervala(R.levoDete, I)
    if R.desnoDete is not None
    and intervaliImajuPresek(R.desnoDete.interval, I):
        dodajMarkereIntervala(R.desnoDete, I)
```

Vremensku složenost konstrukcije segmentnog stabla izračunaćemo kao zbir vremenske složenosti svakog od pomenuta tri koraka. Generisanje elementarnih intervala na osnovu početnog skupa intervala, zbog sortiranja tačaka, ima u najgorem slučaju vremensku složenost $O(n \log n)$. Vremenska složenost konstrukcije segmentnog stabla na osnovu skupa elementarnih intervala iznosi $O(n)$. Poslednji korak je dodavanje markera intervala iz početnog skupa. Za svaki interval potrebno je proći kroz stablo i dodati markere na odgovarajuće čvorove. Pošto će svaki interval imati najviše dva markera intervala po nivou stabla, odnosno $O(\log n)$ markera, vremenska složenost trećeg koraka konstrukcije segmentnog stabla iznosiće $O(n \log n)$ u najgorem slučaju. Dakle ukupna vremenska složenost konstrukcije segmentnog stabla iznosi $O(n \log n)$.

2.4 Intervalna skip lista

U prethodnim poglavljima razmotrili smo dve strukture podataka koje se mogu upotrebiti za efikasno rešavanje problema ispitivanja pripadnosti intervalima. Obe strukture podataka, i intervalna i segmentna stabla, zasnovane su na binarnim stablima pretrage. U ovom poglavlju videćemo da za rešavanje ovog problema, pored

stabala, mogu biti iskorišćene i *skip liste* (eng. *skip list*). Struktura podataka koja u svojoj osnovi predstavlja skip listu, a prilagođena je problemu ispitivanja pripadnosti intervalima, naziva se *intervalna skip lista* (eng. *interval skip list*).

Osnovna varijanta skip liste

Pre nego što razmotrimo intervalnu skip listu, razmotrićemo karakteristike osnovne varijante skip liste. Ako bismo uporedili balansirana binarna stabla pretrage, npr. *AVL stabla* (eng. *AVL tree*), i jednostruko povezanu listu u odnosu na operacije ažuriranja i pretrage, uočili bismo da obe strukture podataka imaju svoje prednosti i mane. Operacija pretrage u slučaju AVL stabala je vremenske složenosti $O(\log n)$ u najgorem slučaju. Međutim, mana AVL stabala jeste to što je prilikom ažuriranja neophodno rotacijama održavati balansiranost stabla. Operacije rotacije, iako konstantne vremenske složenosti, su nešto što ipak utiče na celokupno vreme izvršavanja i što bismo zbog složenosti algoritma želeli da izbegnemo. S druge strane, jednostruko povezana lista ima lošiju složenost pretrage, koja iznosi $O(n)$ u najgorem slučaju, ali kad jednom pronađemo mesto gde želimo da dodamo novi ili obrišemo postojeći element, sama operacija ažuriranja je jednostavna i ima konstantnu složenost $O(1)$. Idealno bi bilo kada bismo mogli da imamo jednostavnu operaciju ažuriranja kao kod jednostruko povezane liste sa logaritamskom operacijom pretrage kao kod AVL stabala. Upravo to je ono što želimo da postignemo strukturom koja se naziva skip lista.

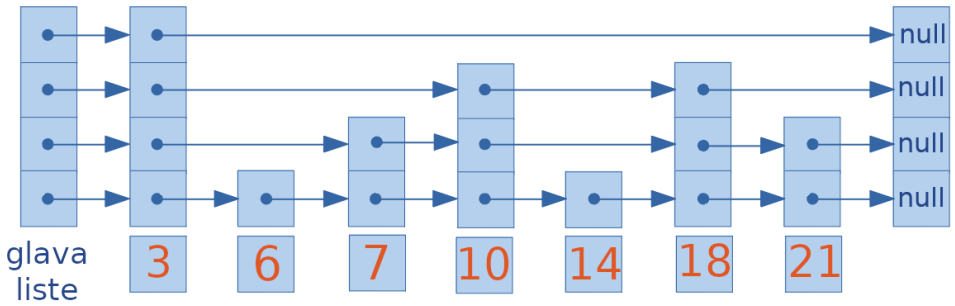
Skip lista [11] u svojoj osnovi predstavlja jednostruko povezanu listu sortiranih elemenata, sa dodatkom da svaki čvor umesto na jedan može da pokazuje na više narednih čvorova. Svaki čvor skip liste ima svoj *nivo*, koji određuje koliko će taj čvor imati pokazivača na naredne elemente. Broj nivoa ograničen je konstantom $MaxNivo$. Glava skip liste je početni čvor i ima $MaxNivo$ pokazivača, po jedan za svaki nivo. Za određivanje nivoa čvora koriste se slučajni brojevi, a verovatnoća da novi čvor bude nivoa k iznosi:

$$P(x = k) = \begin{cases} 0, & \text{za } k < 1 \\ (1-p) \cdot p^{k-1}, & \text{za } k \geq 1 \end{cases}$$

gde važi da $p \in (0, 1)$[9]. Ako je vrednost parametra p jednaka $1/2$, u tom slučaju će skip lista imati, sa najvećom verovatnoćom, polovinu čvorova nivoa jedan (čvorova koji imaju jedan pokazivač na naredni element), četvrtinu nivoa dva, osminu nivoa tri, itd. Ono što se postiže većim brojem pokazivača na naredne elemente i činjenicom

da su elementi u skip listi sortirani je mogućnost bržeg kretanja kroz skip listu, tako što "preskačemo" elemente. Otuda i naziv strukture skip lista.

Primer 7 *Na slici 2.8 dat je grafički prikaz skip liste. Elementi skip liste nivoa jedan su* 6 *i* 14*, nivoa dva* 7 *i* 21*, nivoa tri* 10 *i* 18*, a nivoa četiri je samo element sa brojem* 3*. Razmotrimo kako funkcioniše pretraga elementa sa vrednošću* 14 *na primeru ove skip liste. Pretragu započinjemo od glave skip liste i idemo po najvišem nivou dokle god ne pređemo vrednost* 14*. Prvi čvor najvišeg nivoa je* 3 *i njegova vrednost nije veća od* 14*. Iz čvora* 3 *ne možemo više da se krećemo po nivou četiri jer nema više čvorova na tom nivou pa se spuštamo nivo niže. Sledeću proveru vršimo za čvor* 10 *(nivoa tri), i pošto važi da je* 10 *manje od* 14 *ulazimo u taj čvor. Nastavljamo da se krećemo po nivou tri i nailazimo na sledeći čvor čija je vrednost* 18*. Pošto je* 18 *veće od* 14 *nećemo ući u ovaj čvor već se spuštamo nivo niže. Ista provera će biti i na nivou dva, te se opet spuštamo nivo niže na nivo jedan. Sledeći čvor na nivou jedan je čvor sa brojem* 14 *i to je broj koji smo tražili. Da smo, na primer, tražili broj* 13 *koji se ne nalazi u skip listi, pretragu bismo završili onda kad se spustimo na nivo jedan i krećući se po tom nivou dođemo do elementa koji je veći od traženog elementa. U ovom slučaju došli bismo do elementa sa vrednošću* 14 *i zaključili da se broj* 13 *ne nalazi u skip listi.*



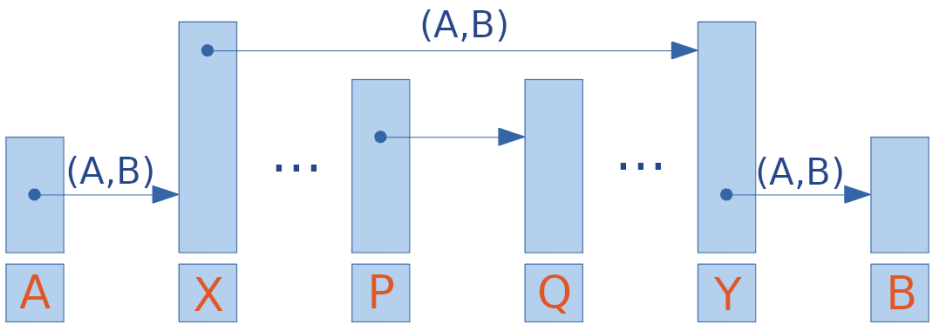
Slika 2.8: Osnovna varijanta skip liste

Intervalna skip lista

Da bismo iskoristili skip listu za rešavanje problema ispitivanja pripadnosti intervalima, potrebno je u skip listu smestiti informacije o skupu intervala. Ideja je da se u čvorovima skip liste nalaze redom sortirane leve i desne granice intervala. Biće, takođe, potrebno čuvati markere intervala iz skupa S na čvorovima i granama [2] skip liste. Potrebno je održavati narednu invarijantu:

Neka je dat interval $I = (A, B)$, i neka se tačke A i B nalaze u skip listi S. Neka postoji grana u skip listi S koja vodi od čvora koji odgovara tački X do čvora koji odgovara tački Y koju ćemo označiti sa (X, Y). Marker intervala I biće dodat u skup markera grane (X, Y) ako i samo ako su ispunjeni uslovi:

- *uslov sadržanosti:* interval I sadrži interval (X, Y);
- *uslov maksimalnosti:* ne postoji nijedna druga grana sa oznakom (X', Y'), za koju važi da se interval (X', Y') nalazi unutar I i sadrži interval (X, Y) [9].



Slika 2.9: Ilustracija uslova sadržanosti i uslova maksimalnosti za postavljanje markera intervala

Na slici 2.9 dat je primer koji ilustruje uslove sadržanosti i maksimalnosti. Pošto ne postoji nijedna grana iznad grane (X, Y) koja se nalazi između krajnjih granica intervala (A, B), i pošto su čvorovi X i Y između granica A i B, granu (X, Y) označavamo markerom intervala (A, B). Granu (P, Q) nećemo označiti markerom

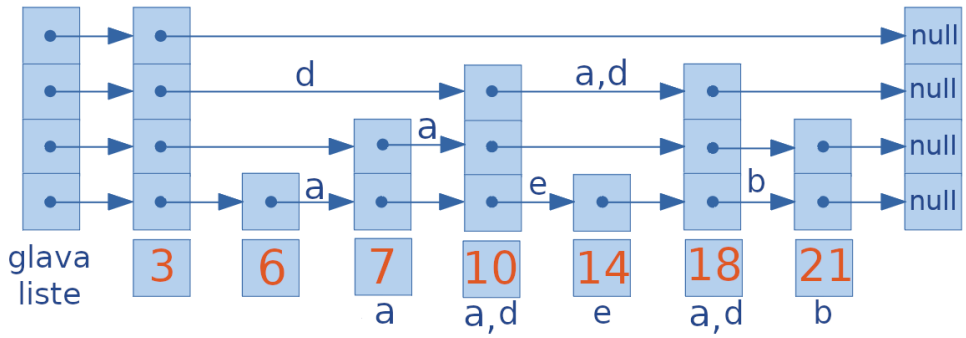
[2] Pod granom skip liste podrazumevaćemo pokazivač sa jednog čvora liste na neki naredni čvor.

intervala (A, B), jer taj interval već pokriva grana (X, Y). Ovim uslovom obezbeđujemo minimalan, a dovoljan broj markera intervala u intervalnoj skip listi.

Svaki čvor V intervalne skip liste sadrži:

- *ključ* - vrednost koja se čuva u čvoru;
- *naredni* - niz pokazivača na naredne elemente skip liste, veličine nivoa čvora;
- *markeri* - niz skupova markera, veličine nivoa čvora;
- *granicniMarkeri* - skup markera intervala kojima su označene grane koje se završavaju u ovom čvoru.

Prva dva atributa, *ključ* i *naredni*, su nasleđena iz strukture originalne skip liste. Atributi *markeri* i *granicniMarkeri* nam služe da bismo označili intervale. U strukturi *markeri* za svaku granu koja polazi iz čvora V ka nekom narednom čvoru čuvamo skup oznaka intervala koji se nalaze na toj grani. Dakle, takvih skupova za jedan čvor imamo onoliko koliko imamo pokazivača na naredne elemente, odnosno koliki je nivo čvora. Atribut *granicniMarkeri* za dati čvor V sadrži skup oznaka intervala na granama koje ulaze u čvor V. Na slici 2.10 prikazano je kako bi izgledala struktura intervalne skip liste za skup intervala $S = \{a = [6, 18], b = [18, 21], c = [7, 7], d = [3, 18], e = [10, 14]\}$. Skupovi pokazivača *markeri* označeni su na granama intervalne skip liste, dok su *granicniMarkeri* označeni ispod čvorova skip liste.



Slika 2.10: Primer intervalne skip liste. Skupovi pokazivača *markeri* označeni su na granama liste, dok su *granicniMarkeri* označeni ispod čvorova liste.

Pretraga intervalne skip liste

Algoritam za pretragu intervalne skip liste kojim se pronalaze svi intervali koji sadrže zadatu tačku q liči na osnovni algoritam pretrage skip liste. Krećemo se od najvišeg nivoa početnog čvora liste i idemo po istom nivou dokle god je ključ tekućeg čvora manji od koordinate tačke q. Kada naiđemo na čvor koji ne zadovoljava ovaj uslov spuštamo se jedan nivo niže. Prilikom svakog prelaska na niži nivo dodajemo u rezultujući skup oznake intervala koje su se nalazile na poslednjoj pretraženoj grani tog nivoa, odnosno grani za koju nije važio uslov da je vrednost ključa narednog elementa manji od q. Takva grana označava interval koji kreće od elementa čija je vrednost ključa manja od q a završava u elementu čija je vrednost ključa veća od q. Dakle, svi intervali skupa S čiji se markeri nalaze na ovoj grani sadržaće zadatu tačku q, te treba da budu dodati u rezultujući skup intervala [9].

Na kraju pretrage, ukoliko se tačka q nalazi u intervalnoj skip listi, naći ćemo njen čvor i tada je potrebno dodati granične markere tog čvora u rezultujući skup intervala. Ukoliko se tačka q ne nalazi u intervalnoj skip listi, odgovarajući čvor nećemo naći, pa je potrebno dodati oznake markera na poslednjoj pretraženoj grani nivoa jedan, iz istog razloga koji je važio i za sve više nivoe.

Primer 8 *Pokažimo na primeru intervalne skip liste sa slike 2.10 kako bi izgledala pretraga za tačku upita* 14. *Krećemo od najvišeg nivoa, nivoa četiri, redom dokle god vrednost narednog čvora ne premaši vrednost* 14. *Prvi element koji ispitujemo ima vrednost* 3. *Pošto je* $3 < 14$ *nastavljamo pretragu. Na nivou četiri nema više elemenata pa se spuštamo nivo niže. Prvi element na nivou tri ima vrednost* 10, *a njegov naredni element ima vrednost* 18. *Važi da je* $10 < 14$, *ali ne važi da je* $18 < 14$, *te se spuštamo nivo niže i dodajemo markere sa grane* $(10, 18)$ *u rezultujući skup. Na kraju dolazimo do nivoa jedan gde je sledeći element broj* 14, *odnosno broj koji smo tražili. Konačno, dodajemo granične markere elementa sa vrednošću* 14 *i rezultujući skup intervala biće jednak* $\{a = [6, 18], d = [3, 18], e = [10, 14]\}$.

Vremenska složenost pretrage intervalne skip liste odgovara vremenskoj složenosti pretrage osnovne verzije skip liste. Vremenska složenost pretrage jednog nivoa intervalne skip liste je $O(1)$, pa pošto imamo $O(\log n)$ nivoa, dolazimo do konačne vremenske složenosti koja iznosi $O(\log n)$ u najgorem slučaju [9].

```
1 # R - glava intervalne skip liste, q - tacka pretrage
2 def pretraziIntervalnuSkipListu(R, q):
```

```
3       trenutniCvor = R.glavaListe
4       rezultat = set()
5
6       for i in range(R.nivo, 0, -1):
7           while trenutniCvor.naredni[i] and trenutniCvor.naredni[i].kljuc < q
        :
8               trenutniCvor = trenutniCvor.naredni[i]
9           rezultat.addAll(trenutniCvor.markeri[i])
10
11      while trenutniCvor.naredni[0] and trenutniCvor.naredni[0].kljuc < q:
12          trenutniCvor = trenutniCvor.naredni[0]
13
14      if trenutniCvor.naredni[0] is None or trenutniCvor.naredni[0].kljuc !=
        q:
15          rezultat.addAll(trenutniCvor.markeri[0])
16      else:
17          rezultat.addAll(trenutniCvor.naredni[0].granicniMarkeri)
18
19      return rezultat
```

Umetanje elementa u intervalnu skip listu

Videli smo da je pretraga intervalne skip liste prilično jednostavna i da se ne razlikuje mnogo od pretrage osnovne verzije skip liste. Značajno složeniji je algoritam kreiranja intervalne skip liste. Prilikom umetanja novog intervala u intervalnu skip listu potrebno je primeniti nekoliko koraka da bi njena struktura ostala ispravna. Neka je interval koji ubacujemo $I = (X, Y)$. Da bismo ga ispravno umetnuli u intervalnu skip listu potrebno je sprovesti sledeće korake:

- umetnuti čvorove sa ključevima X i Y u skip listu, ukoliko se oni već ne nalaze u listi;
- prilikom svakog umetanja novog čvora popraviti postojeće markere intervala tako da je zadovoljena invarijanta intervalne skip liste;
- proći kroz listu od čvora X do čvora Y i označiti odgovarajuće grane i čvorove na tom putu oznakom intervala I, tako da zadovoljavaju invarijantu intervalne skip liste.

Algoritam umetanja novih čvorova X i Y u intervalnu skip listu isti je kao algoritam umetanja novih elemenata u osnovnu skip listu. Da bismo umetnuli čvor v u intervalnu skip listu, krećemo se kroz listu na isti način kao da vršimo pretragu liste sa zadatom tačkom v. Tokom pretrage krećemo se od najvišeg nivoa liste, i ostajemo

na istom nivou sve dok je ispunjen uslov da je vrednost narednog elementa na tom nivou manja od v. Kada uslov više nije ispunjen spuštamo se jedan nivo niže i u niz *azurirani* dodajemo poslednji ispitani čvor za svaki nivo. Na kraju, kada pronađemo mesto u skip listi na koje treba umetnuti novi čvor v, na slučajan način biramo nivo novog čvora i takav čvor dodajemo u listu. Potrebno je prevezati pokazivače koji ulaze i koji izlaze iz novododatog čvora i za to nam služi niz *azurirani*.

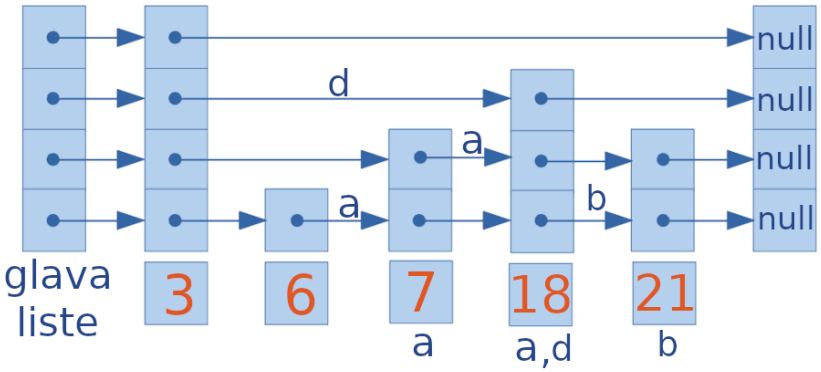
Nakon što umetnemo novi čvor X u intervalnu skip listu, može se narušiti invarijanta time što smo promenili pokazivače liste. Potrebno je proveriti sve grane koje ulaze u novododati čvor X i sve grane koje izlaze iz tog čvora i potencijalno ažurirati markere tih grana ukoliko je to potrebno, odnosno ako je narušena invarijanta intervalne skip liste. Ono što olakšava ovaj algoritam jeste činjenica da prilikom umetanja novog čvora, postojeći markeri intervala mogu samo ili da ostanu na istom nivou ili da se pomere na nivo iznad, odnosno ne mogu da se pomere na niži nivo [9].

Nakon umetanja novih čvorova X i Y i ažuriranja postojećih markera intervala u listi, dolazimo do poslednjeg koraka, a to je dodavanje markera za novi interval $I = (X, Y)$. Potrebno je naći čvor X i kretati se kroz intervalnu skip listu do čvora Y. Na tom putu treba dodati markere za novi interval I, tako da invarijanta intervalne skip liste bude zadovoljena. U opštem slučaju biće potrebno da se od čvora X pomerimo nekoliko nivoa naviše dokle god važi da interval I sadrži interval kojeg formiraju vrednost posmatranog čvora i vrednost narednog čvora. Kada ovaj uslov više nije ispunjen spuštamo se nekoliko nivoa naniže do čvora Y, koristeći algoritam pretrage intervalne skip liste.

Primer 9 *Pokažimo na primeru kako izgleda algoritam umetanja novog intervala u intervalnu skip listu. Neka je početni skup intervala* $\{a = [6, 18], b = [18, 21], c = [7, 7], d = [3, 18]\}$ *i neka je novi interval koji želimo da umetnemo u listu* $e = [10, 14]$. *Na slici 2.11 je prikazano početno stanje intervalne skip liste. Oznake na granama intervalne skip liste predstavljaju atribut* markeri*, dok oznake ispod čvorova skip liste predstavljaju atribut* granicniMarkeri*. Potrebno je umetnuti krajnje granice intervala* e *u intervalnu skip listu, odnosno čvorove sa vrednošću* 10 *i* 14*, pa nakon toga ažurirati postojeće markere intervala. Najpre pravimo novi čvor sa vrednošću* 10*, pošto u intervalnoj skip listi ne postoji čvor sa tom vrednošću. Na slučajan način biramo nivo čvora sa vrednošću* 10*. Pretpostavimo da je nivo novog čvora jednak* 3*. Algoritmom pretrage intervalne skip liste dolazimo do mesta gde je potrebno umetnuti čvor nivoa tri sa vrednošću* 10*. Nakon što smo umetnuli novi čvor i prevezali*

grane potrebno je ažurirati sve markere intervala koji se nalaze na granama koje ulaze ili izlaze iz novog čvora. Nakon ove operacije dobićemo intervalnu skip listu koja izgleda kao na slici 2.12. Isto je potrebno uraditi za element sa vrednošću 14*, a struktura intervalne skip liste nakon umetanja ovog elementa može se videti na slici 2.13. Poslednji korak algoritma umetanja novog intervala u intervalnu skip listu je dodavanje markera za novi interval. Krećemo od elementa sa vrednošću* 10 *ka elementu sa vrednošću* 14 *i dodajemo markere intervala* e *na tom putu. Na slici 2.14 možemo videti konačan rezultat nakon umetanja novog intervala* $e = [10, 14]$.

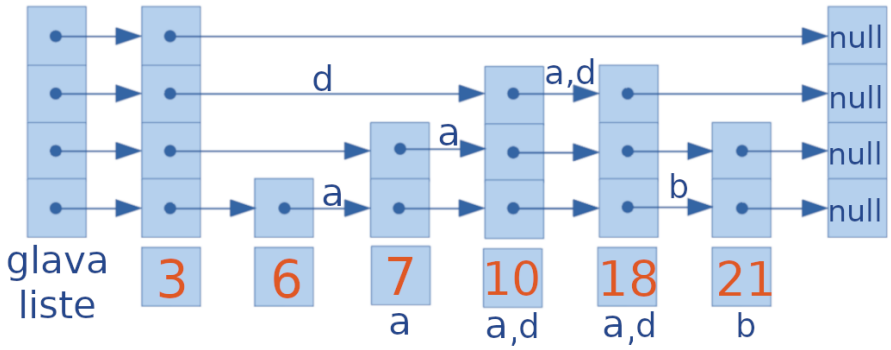
U nastavku teksta dat je pseudokod algoritma umetanja novog intervala u intervalnu skip listu.



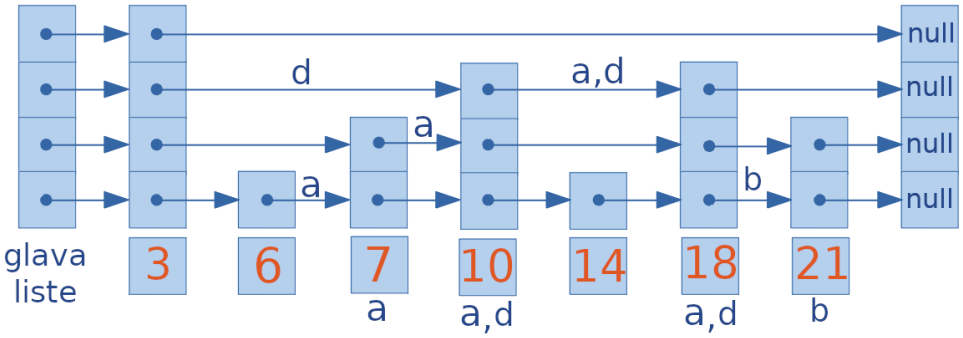
Slika 2.11: Primer intervalne skip liste pre umetanja novog intervala

```
# X - novododati cvor
# azurirani - niz azuriranih cvorova prilikom umetanja X
def azuriranjeMarkeraNakonDodavanja(X, azurirani):
    # Faza 1: popravi markere na granama koje izlaze iz X

    promovisani = set()
    noviPromovisani = set()
    for i in range(0, X.nivo, 1):
        for m in azurirani[i].markeri[i]:
            if m.sadrziInterval(X.kljuc, X.sledeci[i + 1].kljuc):
                # promovisi m
                # obrisi markere m u intervalu pomocniInterval na nivou i
                pomocniInterval = Interval(X, X.sledeci[i + 1])
                obrisiMarkere(pomocniInterval, m, i)
                noviPromovisani.add(m)
```

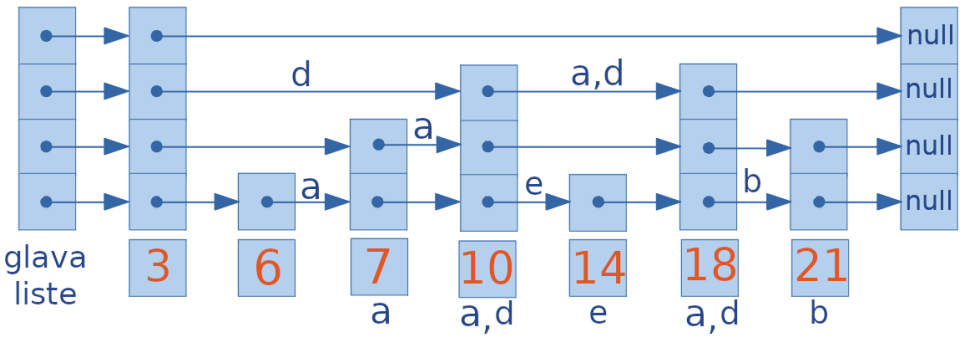


Slika 2.12: Primer intervalne skip liste nakon umetanja čvora sa vrednošću 10



Slika 2.13: Primer intervalne skip liste nakon umetanja čvora sa vrednošću 14

```
16              else:
17                  X.markeri[i].add(m)
18                  X.sledeci[i].granicniMarkeri.add(m)
19          for m in promovisani:
20              if not m.sadrziInterval(X.kljuc, X.sledeci[i + 1].kljuc):
21                  # m ne treba vise da se promovise na vise nivoe
22                  X.markeri[i].add(m)
23                  X.sledeci[i].granicniMarkeri.add(m)
24                  promovisani.discard(m)
25              else:
26                  # nastavi da promovises m
27                  pomocniInterval = Interval(X, X.sledeci[i + 1])
28                  obrisiMarkere(pomocniInterval, m, i)
```



Slika 2.14: Primer intervalne skip liste nakon dodavanja markera za novi interval e

```
29          promovisani.update(noviPromovisani)
30          noviPromovisani = set()
31
32      nivoX = X.novo
33      X.markeri[nivoX] = promovisani
34      X.markeri[nivoX].update(azurirani[nivoX].markeri[nivoX])
35      X.sledeci[nivoX].granicniMarkeri.update(X.markeri[nivoX])
36
37      # Faza 2: popravi markere na granama koje ulaze u X
38
39      promovisani = set()
40      noviPromovisani = set()
41      for i in range(0, X.nivo, 1):
42          for m in azurirani[i].markeri[i]:
43              if m.sadrziInterval(azurirani[i + 1].kljuc, X.kljuc):
44                  # promovisi m
45                  pomocniInterval = Interval(azurirani[i + 1], X)
46                  obrisiMarkere(pomocniInterval, m, i)
47                  noviPromovisani.add(m)
48              else:
49                  X.granicniMarkeri.add(m)
50          for m in promovisani:
51              if m.sadrziInterval(azurirani[i].kljuc, X.kljuc)
52              and not m.sadrziInterval(azurirani[i + 1].kljuc, X.kljuc):
53                  # m ne treba vise da se promovise na vise nivoe
54                  azurirani[i].markeri[i].add(m)
55                  X.granicniMarkeri.add(m)
56                  promovisani.discard(m)
57              else:
58                  # nastavi da promovises m
59                  pomocniInterval = Interval(azurirani[i + 1], X)
```

```
                obrisiMarkere(pomocniInterval, m, i)
        promovisani.update(noviPromovisani)
        noviPromovisani = set()

    nivoX = X.nivo
    azurirani[nivoX].markeri[nivoX].update(promovisani)
    X.granicniMarkeri.update(azurirani[nivoX].markeri[nivoX])
```

```
# I - novododati interval
def dodajMarkereZaNoviInterval(I):
    x = nadjiKljuc(I.levaGranica)
    if I.sadrziTacku(x.kljuc):
        x.granicniMarkeri.add(I.marker)
    i = 0
    while I.sadrziInterval(x.kljuc, x.sledeci[i].kljuc):
        while i <= x.nivo - 1
        and I.sadrziInterval(x.kljuc, x.sledeci[i + 1].kljuc):
            i = i + 1
        x.markeri[i].add(I.marker)
        x = x.sledeci[i]
        if I.sadrziTacku(x.kljuc):
            x.granicniMarkeri.add(I.marker)
    while x.kljuc < I.desnaGranica:
        while i > 0 and not I.sadrziInterval(x.kljuc, x.sledeci[i].kljuc):
            i = i - 1
        x.markeri[i].add(I.marker)
        x = x.sledeci[i]
        if I.sadrziTacku(x.kljuc):
            x.granicniMarkeri.add(I.marker)
```

Vremenska složenost operacije umetanja intervala u intervalnu skip listu odgovara zbiru vremenskih složenosti svakog od tri koraka ovog algoritma. Umetanje novih čvorova u intervalnu skip listu odgovara vremenskoj složenosti istog algoritma za osnovnu skip listu i jednaka je $O(\log n)$ u najgorem slučaju. Pretpostavimo da su markeri intervala na granama čuvani u strukturi podataka čije je vreme izvršavanja operacija umetanja i brisanja $O(\log n)$, na primer u balansiranom binarnom stablu. Vremenska složenost ažuriranja postojećih markera u intervalnoj skip listi nakon umetanja novog čvora iznosi $O(\log^2 n)$. Pošto intervalna skip lista ima $O(\log n)$ nivoa, a potrebno je konstantnom broju grana po nivou dodati markere novog intervala, gde je vremenska složenost dodavanja markera $O(\log n)$, dolazimo do ukupne vremenske složenosti dodavanja markera za novi interval i ona iznosi $O(\log^2 n)$. Konačna vremenska složenost algoritma umetanja novog intervala u intervalnu skip listu predstavlja zbir ove tri složenosti i iznosi $O(\log^2 n)$ u najgorem slučaju [9].

2.5 Drugi pristupi rešavanju problema pripadnosti intervalima

U ovom delu rada predstavićemo neke složenije strukture podataka koje mogu biti iskorišćene za rešavanje problema ispitivanja pripadnosti intervalima. Jedna takva struktura je *R-stablo* (eng. *R-tree*). R-stablo je struktura podataka koja je specijalno dizajnirana za smeštanje prostornih, odnosno višedimenzionih podataka. Ova struktura podataka predstavlja vrstu balansiranih stabala pretrage. Glavna ideja je da se grupišu objekti koji su blizu i da takve grupe objekata predstavimo njihovim minimalnim ograničavajućim pravougaonikom [8]. Otud i ime R-stablo, gde je R skraćeno od engleske reči za pravouganik (eng. *rectangle*). U svakom listu R-stabla nalaze se pojedinačni objekti, dok se u unutrašnjim čvorovima nalaze grupe susednih objekata. U slučaju rešavanja problema ispitivanja pripadnosti intervalima, u svakom listu stabla bi se nalazio po jedan interval iz početnog skupa intervala, predstavljen parom brojeva x i y, koji odgovaraju krajnjim granicama intervala. U svim ostalim čvorovima čuvali bismo pokazivače na levo i desno dete čvora, kao i minimalni ograničavajući pravouganik intervala njegove dece. Za skup intervala, minimalni ograničavajući pravougaonici bi predstavljali nove intervale koji sadrže intervale oba deteta čvora, a informacija o njima bi se takođe čuvala kao par brojeva. Bitan deo algoritma za konstrukciju R-stabla je heuristika koja se koristi za odlučivanje na koji način će se pravougaonici podeliti u podstabla. Cilj je naći takvu heuristiku kojom ćemo postići da R-stablo bude balansirano, a da sa druge strane minimalni ograničavajući pravougaonici zauzimaju što manje praznog prostora. Različite heuristike mogu biti iskorišćene za smeštanje intervala u R-stablo, a jedna od njih je *heuristika najmanje regije* (eng. *minimum area*). Ova heuristika smešta novododati čvor u podstablo čiji je minimalni ograničavajući pravougaonik najmanje površine. Zbog toga što je potrebno pronaći heuristiku koja će se efikasno pokazati za smeštanje intervala u R-stablo, ovaj algoritam je teži za implementaciju, što je ujedno i jedna njegova mana. Nasuprot tome, velika prednost ovog algoritma je njegova prostorna složenost koja za razliku od ostalih stabala iznosi $O(n)$. Ova struktura podataka pokazala je dobre performanse kada pravougaonici imaju mali stepen preklapanja [9]. Iako ne garantuje dobru vremensku složenost najgoreg slučaja, u praksi ova struktura podataka pokazuje dobre rezultate.

Još jedna struktura podataka koja omogućava dinamičko menjanje strukture i pritom efikasno rešava problem ispitivanja pripadnosti intervalima je *stablo pretrage*

sa prioritetom (eng. *priority search tree*). Ova struktura podataka nastala je kao kombinacija *reda sa prioritetom* (eng. *priority queue*) i binarnog stabla pretrage. U stablo pretrage sa prioritetom smeštaju se dvodimenzione tačke. Svaka tačka odgovara tačno jednom čvoru stabla i važe sledeće osobine:

- za svaki čvor v koji nije koren važi da njemu odgovarajuća tačka ima manju y koordinatu od y koordinate tačke u čvoru roditelja v;
- za svaki unutrašnji čvor stabla v važi da sve tačke u levom podstablu čvora v imaju x koordinatu manju od svih x koordinata tačaka u desnom podstablu čvora v.

Prva od navedenih osobina nam govori da je stablo pretrage sa prioritetom formirano kao *max-hip* (eng. *max heap*) na osnovu y koordinata tačaka. Druga osobina nam pak govori da je stablo pretrage sa prioritetom binarno stablo pretrage sa uređenjem po x koordinatama tačaka, sa izuzetkom da ne postoji relacija između x koordinate tačke čuvane u čvoru v i u deci čvora v [7]. U slučaju rešavanja problema ispitivanja pripadnosti intervalima, čvor u stablu pretrage sa prioritetom odgovarao bi jednom intervalu, tako što bi x koordinata tačke koja se čuva u čvoru predstavljala levu granicu, a y koordinata desnu granicu intervala. Kao i R-stablo, stablo pretrage sa prioritetom ima dobru prostornu složenost - $O(n)$. Jedan veliki nedostatak rešenja baziranog na ovoj strukturi podataka jeste što nije jednostavan za implementaciju, naročito u slučaju kada možemo očekivati više intervala koji imaju istu levu granicu intervala. Pošto je stablo uređeno po x koordinati, odnosno po levim granicama intervala, u tom slučaju neophodno je prvo transformisati ulazni skup intervala u skup intervala koji ima jedinstvene leve granice. Ova tranformacija utiče na to da se poveća vremenska složenost konstrukcije stabla [9].

Glava 3

Implementacija i evaluacija

U radu su predstavljene tri različite strukture podataka uz pomoć kojih je moguće rešavati problem ispitivanja pripadnosti intervalima. Sve tri strukture podataka kao i funkcije za njihovu pretragu i konstrukciju su implementirane u programskom jeziku *Python* [3]. Za njihovu implementaciju korišćene su postojeće pomoćne strukture podataka kao što su skup, dinamički niz i rečnik. Implementacija sve tri strukture podataka i njihova evaluacija napisana je u oko 700 linija koda. Za potrebe evaluacije predstavljenih algoritama i pravljenje grafika vremena izvršavanja korišćena je biblioteka *Matplotlib* [2]. Implementacije predstavljenih algoritama mogu se naći na javnom repozitorijumu `https://github.com/anja-ivanisevic/interval-stabbing-queries-problem`.

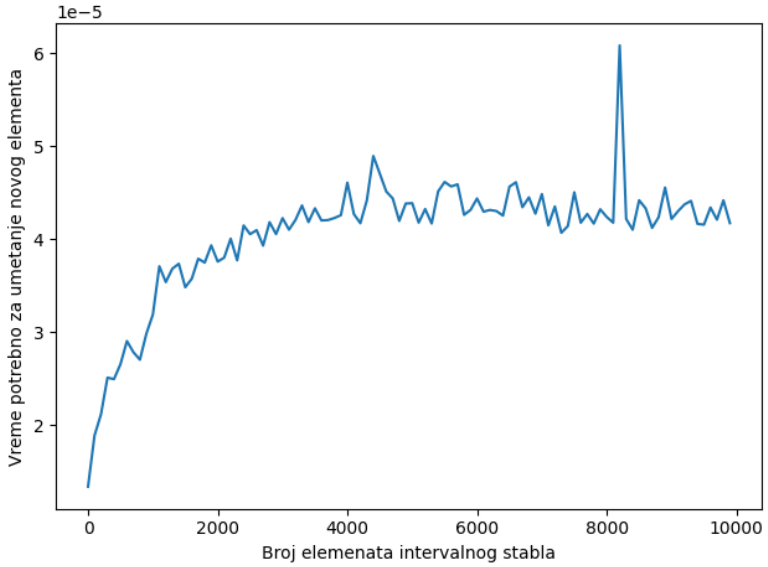
Evaluacija

U ovom poglavlju biće predstavljeni rezultati merenja vremena izvršavanja algoritama za ispitivanje pripadnosti intervalima zasnovanih na pomenute tri strukture podataka. Merenja algoritama su izvršavana na računaru sa sledećom konfiguracijom:

- Procesor: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz, 1992 Mhz, 4 Core(s), 8 Logical Processor(s)
- RAM: 8GB LPDDR3
- OS: Microsoft Windows 10 Pro, Version 10.0.19042 Build 19042

Intervalna stabla

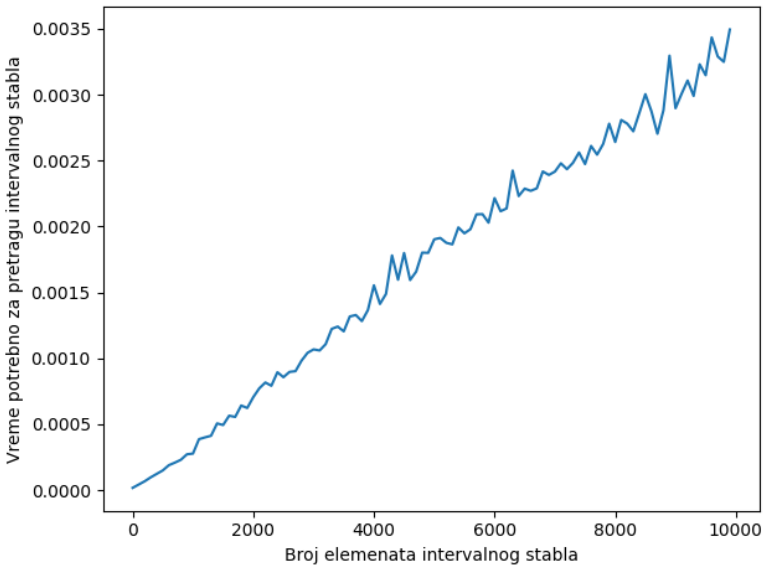
Da bi se procenila efikasnost implementacije zasnovane na intervalnim stablima, razmatrana su vremena izvršavanja dve ključne operacije. Prva operacija je umetanje novog elementa u intervalno stablo, koja je potrebna za konstrukciju intervalnog stabla na osnovu početnog skupa intervala. Na grafiku 3.1 prikazano je kako se kreće vreme izvršavanja operacije umetanja novog elementa u intervalno stablo u zavisnosti od broja postojećih elemenata u strukturi. Grafik je konstruisan tako što je u svakoj iteraciji konstruisano intervalno stablo sačinjeno od n elemenata, gde se broj n kretao od 100 do 10000. Intervali su konstruisani korišćenjem generatora slučajnih brojeva. Leva granica intervala je generisana kao slučajan broj u intervalu $[0, n]$, dok je desna granica generisana kao slučajan broj u intervalu $[levaGranica, n]$. Nakon svake iteracije povećavana je prethodna vrednost za n za 100. U ovako konstruisano intervalno stablo umetan je novi element i mereno je vreme izvršavanja operacije umetanja. Svaka od iteracija izvršena je 100 puta i za svako n prikazano je srednje vreme izvršavanja operacije umetanja elementa. Može se primetiti da vreme izvršavanja operacije umetanja novog elementa u intervalno stablo logaritamski raste sa porastom broja elemenata u intervalnom stablu, te da grafik odgovara vremenskoj složenosti koju smo očekivali za ovu operaciju, koja iznosi $O(\log n)$.



Slika 3.1: Vreme izvršavanja operacije umetanja novog elementa u intervalno stablo u odnosu na broj elemenata stabla izraženo u sekundama

Druga operacija od interesa za problem koji rešavamo jeste pretraga intervalnog

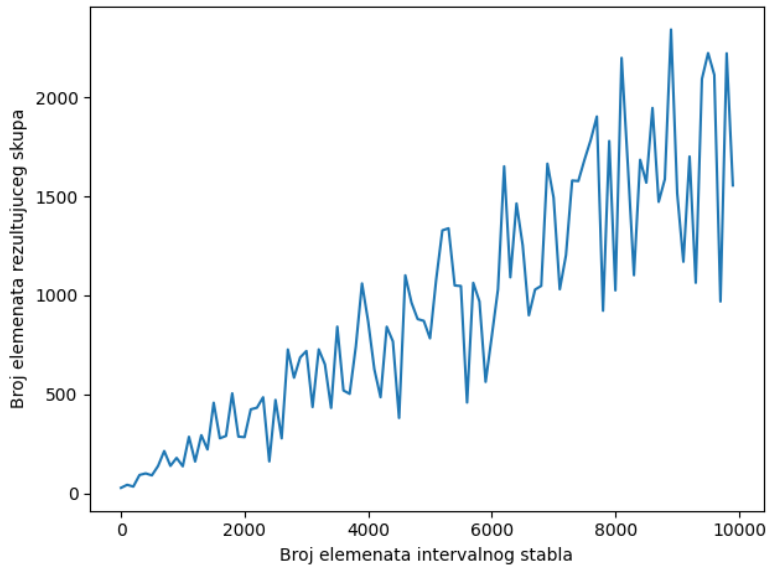
stabla na osnovu zadate tačke q koja vraća skup elemenata intervalnog stabla koji sadrže tačku q. Slično kao i za prethodni grafik, da bi se izmerilo vreme izvršavanja operacije pretrage u odnosu na broj elemenata intervalnog stabla, sprovedeno je n iteracija, gde se broj n kretao od 100 do 10000. U svakoj iteraciji vršena je pretraga intervalnog stabla koji sadrži n elemenata na osnovu zadate tačke q i povećavana je prethodna vrednost za n za 100. Svaka od iteracija izvršena je 100 puta i konačni rezultat predstavlja srednje vreme izvršavanja operacije pretrage intervalnog stabla. Tačka q je izabrana na slučajan način iz skupa svih krajnjih granica intervala koji se nalaze u intervalnom stablu, da bi se osiguralo da rezultujući skup intervala bude neprazan. Na grafiku 3.2 prikazan je odnos vremena izvršavanja operacije pretrage intervalnog stabla u zavisnosti od broja elemenata koji se nalaze u intervalnom stablu.



Slika 3.2: Vreme izvršavanja operacije pretrage intervalnog stabla u odnosu na broj elemenata stabla izraženo u sekundama

Može se primetiti da vreme izvršavanja operacije pretrage intervalnog stabla raste brže od vremena izvršavanja operacije umetanja intervala u intervalno stablo. Očekivana vremenska složenost operacije pretrage iznosi $O(\log n + k)$, gde je k broj intervala rezultujućeg skupa. Potrebno je proveriti koliko vrednost k utiče na vremensku složenost pretrage i stoga je konstruisan grafik na slici 3.3, gde se može videti kako raste broj elemenata rezultujućeg skupa k pretrage u odnosu na broj elemenata intervalnog stabla. Može se primetiti da grafik odgovara lineranoj funkci-

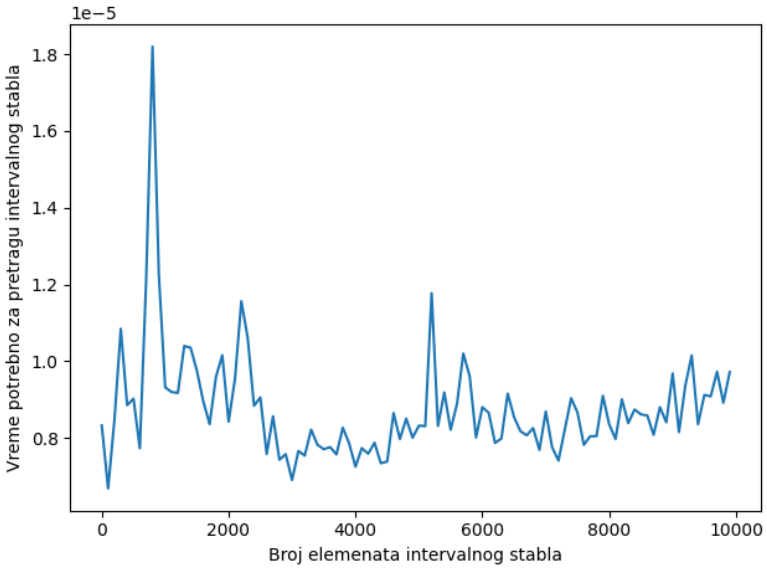
ji. Takođe izmereno je vreme izvršavanja operacije pretrage intervalnog stabla koja vraća samo jedan element. Rezultati merenja mogu se videti na slici 3.4 i grafik funkcije odgovara logaritamskoj funkciji. Zaključujemo da je vrednost k dosta uticala na vremensku složenost pretrage intervalnog stabla, što ujedno objašnjava rast grafika vremena izvršavanja operacije pretrage intervalnog stabla.



Slika 3.3: Grafik rasta broja elemenata rezultujućeg skupa k pretrage intervalnog stabla u odnosu na broj elemenata intervalnog stabla

Segmentna stabla

Za razliku od intervalnog stabla koje podržava dinamičko umetanje novih elemenata, segmentno stablo je statička struktura podataka. To znači da segmentno stablo ne podržava promene nakon što je jednom konstruisano. Vreme umetanja jednog elementa u slučaju segmentnog stabla nije moguće izmeriti jer se elementi ne dodaju jedan po jedan već se segmentno stablo konstruiše na osnovu celog skupa intervala. Ostaje da se grafički prikaže kako se kreće vreme pretrage segmentnog stabla u zavisnosti od broja elemenata stabla. Konstruisano je segmentno stablo od n elemenata, gde se broj n kretao od 100 do 10000 i u svakoj iteraciji je povećavana prethodna vrednost za n za 100. Elementi segmentnog stabla su konstruisani korišćenjem generatora slučajnih brojeva, na isti način kao što je to urađeno za intervalno stablo. U svakoj iteraciji mereno je vreme izvršavanja operacije pretrage segmentnog stabla, gde je za tačku upita q uzimana nasumično izabrana krajnja granica

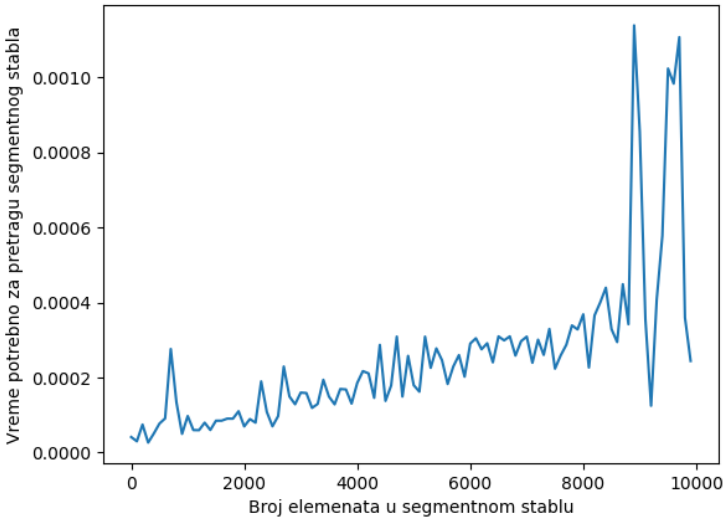


Slika 3.4: Vreme izvršavanja operacije pretrage intervalnog stabla koje vraća samo jedan element u odnosu na broj elemenata stabla izraženo u sekundama

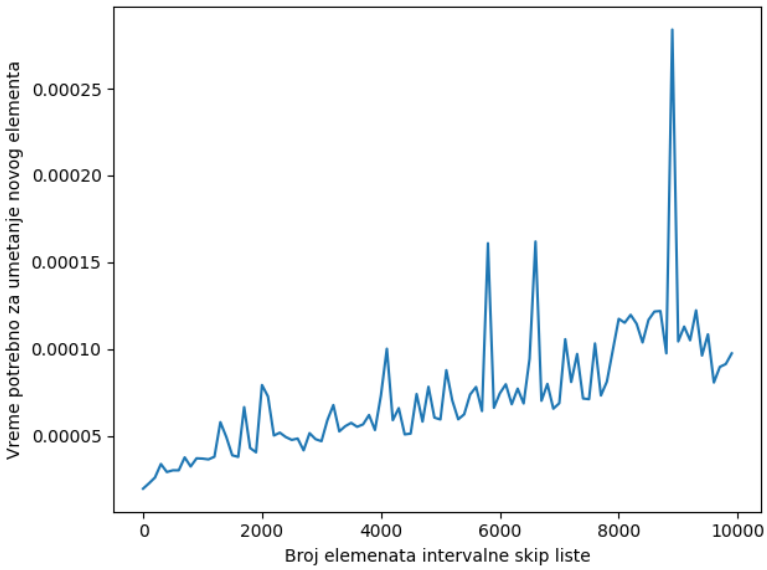
intervala iz skupa intervala. Svaka iteracija je izvršavana po 100 puta i prikazano je srednje vreme izvršavanja pretrage segmentnog stabla. Na grafiku 3.5 prikazan je odnos vremena izvršavanja operacije pretrage segmentnog stabla u zavisnosti od broja elemenata koji se nalaze u segmentnom stablu. Grafik odgovara očekivanoj vremenskoj složenosti od $O(\log n + k)$, gde je k broj intervala rezultujućeg skupa.

Intervalna skip lista

U slučaju intervalne skip liste, konstruisan je grafik vremenske složenosti operacija umetanja i pretrage na isti način kao što je to urađeno za intervalno stablo. Na grafiku 3.6 može se videti odnos vremena izvršavanja operacije umetanja novog elementa u intervalnu skip listu u zavisnosti od broja elemenata skip liste, dok grafik 3.7 prikazuje odnos vremena izvršavanja operacije pretrage intervalne skip liste u zavisnosti od broja elemenata. Dobijen je sličan rezultat kao u slučaju intervalnog stabla. Oba grafika, i za operaciju umetanja i za operaciju pretrage intervalne skip liste, pokazuju da vreme tih operacija logaritamski raste sa porastom broja elemenata intervalne skip liste. Očekivana vremenska složenost operacije umetanja novog elementa je $O(\log^2 n)$, dok je složenost operacije pretrage jednaka $O(\log n)$.



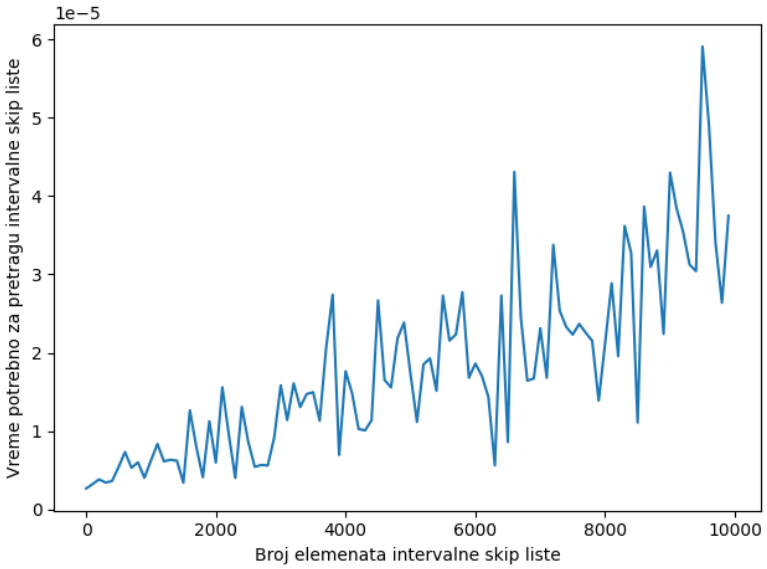
Slika 3.5: Vreme izvršavanja operacije pretrage segmentnog stabla u odnosu na broj elemenata stabla izraženo u sekundama



Slika 3.6: Vreme izvršavanja operacije umetanja novog elementa u intervalnu skip listu u odnosu na broj elemenata liste izraženo u sekundama

Poređenje rezultata

Na osnovu dobijenih grafika vremenske složenosti operacija nad prikazanim strukturama podataka, može se izvršiti analiza koja se od struktura podataka pokazala

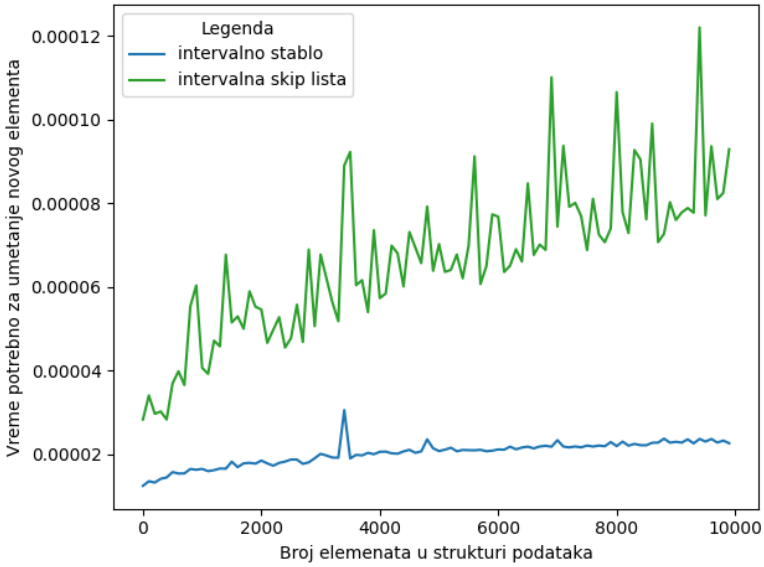


Slika 3.7: Vreme izvršavanja operacije pretrage intervalne skip liste u odnosu na broj elemenata liste izraženo u sekundama

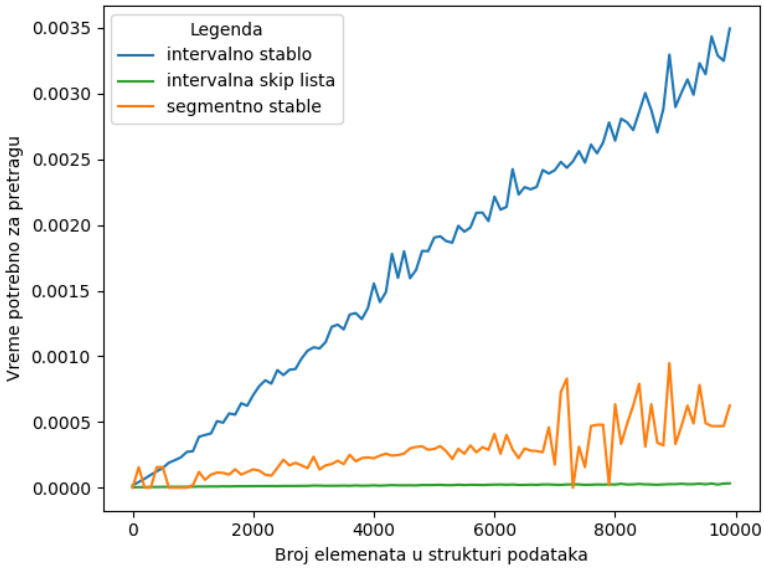
najefikasnijom. U slučaju operacije umetanja novog elementa u strukturu podataka, može se uporediti samo intervalno stablo i intervalnu skip lista. Naime, kako smo već napomenuli, segmentno stablo ne podržava dinamičko umetanje novih elemenata, te se može reći da ono, zbog ovakvog ograničenja, u startu predstavlja lošije rešenje problema ispitivanja pripadnosti intervalima. Ukoliko je neophodno da se elementi umeću u realnom vremenu, rešenje zasnovano na segmentnom stablu nije prihvatljivo. Međutim, ukoliko se unapred zna da neće biti dinamičkog umetanja novih elemenata, onda se segmentno stablo može razmatrati kao potencijalno rešenje ovog problema.

Na osnovu grafika 3.8 može se videti da je, za prikazane vrednosti n do 10000 intervalno stablo pokazalo bolje rezultate od intervalne skip liste. Mogu se uporediti i vremena izvršavanja operacije pretrage. Na osnovu grafika 3.9 može se primetiti da je intervalno stablo pokazalo najlošije rezultate. Iznenađujuće dobre rezultate pokazalo je segmentno stablo. Iako segmentno stablo ima ograničenje po pitanju naknadnog ažuriranja i umetanja novih intervala, ispostavlja se da ono daje dobre rezultate pretrage. Dakle, ukoliko nije neophodno da se struktura podataka naknadno menja, segmentno stablo predstavlja dobar izbor algoritma za rešavanje problema pripadnosti intervalima. Najbolje rezultate dala je intervalna skip lista koja ujedno predstavlja i najefikasnije od tri predstavljena rešenja za problem ispitivanja

pripadnosti intervalima.



Slika 3.8: Poređenje vremena izvršavanja operacije umetanja novog intervala izraženo u sekundama



Slika 3.9: Poređenje vremena izvršavanja operacije pretrage izraženo u sekundama

Glava 4

Zaključak

Problem ispitivanja pripadnosti intervalima je problem koji spada u oblast računarske geometrije. Neke od značajnijih primena ovog problema su u sistemima za upravljanje bazama podataka, u geografskim informacionim sistemima, u razvoju algoritama i struktura podataka za eksternu memoriju i u rešavanju problema prepoznavanja šablona. U radu su predstavljena neka rešenja ovog problema, sa fokusom na tri strukture podataka: intervalna stabla, segmentna stabla i intervalne skip liste. Sve tri strukture podataka su detaljno prikazane sa pseudokodom osnovnih operacija, analizom njihove složenosti i evaluacijom dobijenih rešenja.

Analizirane su dve ključne operacije za rešavanje problema ispitivanja pripadnosti intervalima, a to su umetanje novog intervala u strukturu podataka i pretraga strukture podataka po zadatoj tački. Prva operacija, umetanje novog elementa, evaluirana je za intervalno stablo i intervalnu skip listu. Intervalno stablo je pokazalo u proseku tri puta kraće vreme izvršavanja operacije umetanja novog intervala od intervalne skip liste. Pošto je segmentno stablo statička struktura podataka, nije bilo moguće razmatrati ovu operaciju. Druga operacija, operacija pretrage, pokazala je da intervalna skip lista predstavlja optimalno rešenje problema ispitivanja pripadnosti intervalima. Iznenađujuće dobre rezultate pokazalo je segmentno stablo, dok se kao najlošije pokazalo intervalno stablo. Zaključujemo da u slučaju problema ispitivanja pripadnosti intervalima intervalna skip lista ima velikih prednosti u poređenju sa ostala dva algoritma, i da od prikazanih algoritama predstavlja najefikasnije rešenje problema ispitivanja pripadnositi intervalima.

Neke od ideja za nastavak istraživanja rešavanja problema ispitivanja pripadnosti intervalima bi bile da se detaljnije analiziraju i implementiraju druge strukture podataka koje se mogu upotrebiti za rešavanja ovog problema, kao što su već po-

menuto R-stablo i stablo pretrage sa prioritetom, kao i rešavanje problema u više dimenzija.

Bibliografija

[1] CP Algorithms, Segment Tree. on-line at `https://cp-algorithms.com/data_structures/segment_tree.html`.

[2] Matplotlib. on-line at `https://matplotlib.org/`.

[3] Python. on-line at `https://www.python.org/`.

[4] Pankaj Agarwal, Lars Arge, and Ke Yi. An optimal dynamic interval stabbing-max data structure? Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 803-812, 2005.

[5] Djamal Belazzougui. Worst-case efficient single and multiple string matching on packed texts in the word-ram model. Journal of Discrete Algorithms, vol 14, pp. 91-106, 2012.

[6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* Massachusetts Institute of Technology, 2009.

[7] Minati De, Anil Maheshwari, Subhas Nandy, and Michiel Smid. An in-place priority search tree. Proceedings of the 23rd Annual Canadian Conference on Computational Geometry, Toronto, Ontario, Canada, August 10-12, 2011.

[8] Antonin Guttman. R trees: A dynamic index structure for spatial searching. Sigmod Record, vol 14, pp. 47-57, 1984.

[9] Eric N. Hanson and Theodore Johnson. The Interval Skip List: A Data Structure for Finding All Intervals That Overlap a Point. Algorithms and Data Structures. Lecture Notes in Computer Science, vol 519, pp. 153-164, 1992.

[10] Miodrag Živković. *Algoritmi.* Matematički fakultet, Univerzitet u Beogradu, Beograd, 2000.

[11] Vesna Marinković and Miodrag Živković. Skripta iz Konstrukcije i analize algoritama 2. on-line at `http://poincare.matf.bg.ac.rs/~vesnap/kaa2/kaa2.pdf`.

[12] Marc van Kreveld Mark de Berg, Otfried Cheong and Mark Overmars. *Computational Geometry.* Springer, Berlin, 2008.

[13] Jeffrey Scott Vitter. *Algorithms and Data Structures for External Memory.* now Publishers Inc., USA, 2006.