SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Bioinformatika

Izgradnja stabla valića (engl. multiary wavelet tree)

Studenti: Maja Buljan
Marin Grmača
Kristijan Ivančić
Jelena Konfic
Nikola Munđer
Marijana Spajić

Voditelj: Mirjana Domazet-Lošo

Zagreb, siječanj, 2015.

Sadržaj

1. Izgradnja stabla valića

Stablo valića (engl. *wavelet tree*) se koristi za spremanje velikih nizova podataka u podatkovne strukture u svrhu bržeg izvršavanja operacija *rank* i *select*. Operacija *rank* vraća koliko puta se traženi znak pojavljuje do određenog mjesta u nizu, dok operacija *select* vraća na kojem mjestu u nizu se nalazi *i*-ti traženi znak (pri čemu je *i* broj pojavljivanja znaka).

Najjednostavniji primjer je binarno stablo valića (engl. *binary wavelet tree*) pri čemu se znakovni niz kodira kao binarno stablo bit-vektora. Time je omogućeno izvršavanje operacija *rank* i *select* u vremenu *O(log(n) * log(k)),* gdje je *n* duljina znakovnog niza, a *k* broj znakova abecede niza. U slučajevima kada se abeceda niza sastoji od velikog broja znakova, *log(k)* će poprimiti veliku vrijednost čime se povećava vrijeme izvođenja operacija *rank* i *select*. Navedenu vrijednost, a i broj razina u stablu, moguće je smanjiti korištenjem stabla kratnosti veće od 2 – *multiary* stabla valića (engl. *multiary wavelet tree, MWT*). U tom slučaju se brzina izvođenja ubrzava za faktor *log(m),* gdje *m* predstavlja kratnost stabla. Također, broj razina u stablu je smanjen za isti faktor.

1.1. Postupak izgradnje MWT-a

Pretpostavimo da ulazni niz može sadržavati samo osnovni *ASCII* skup znakova. Stoga, abeceda ulaznog niza ima 7 bitova. Iz toga je moguće izračunati broj razina u stablu prema sljedećoj formuli.

$$brojRazina = \left\lceil \frac{7}{\log_2 m} \right\rceil$$

pri čemu je *m* kratnost stabla. Iz formule je vidljivo da je maksimalna kratnost stabla 128.

Neka je niz „*-BBDappa.HPDa -*“ ulazni niz znakova za koji je potrebno izgraditi stablo. Kratnost stabla je 4.

$$S = -BBDappa.HPDa -$$

$$m = 4$$

Prvo je potrebno odrediti znakove abecede niza *S*.

$$A = \{B, D, H, P, a, p, -, .\}$$

Nakon što su određeni znakovi abecede, svaki od znakova je potrebno prikazati kao jednu kodnu riječ koja se sastoji samo od brojeva. S obzirom na to da ulazni znak može biti samo jedan od *ASCII* znakova, znak se pretvara u njegovu *ASCII* vrijednost te se zatim taj broj pretvara u bazu koja je određena kratnošću stabla. Kodiranje abecede niza *S* je prikazano u sljedećoj tablici (Tablica 1).

Znak	ASCII vrijednost	Kodna riječ
B	66_{10}	1002_4
D	68_{10}	1010_4
H	72_{10}	1020_4
P	80_{10}	1100_4
a	97_{10}	1201_4
p	112_{10}	1300_4
-	45_{10}	0231_4
.	46_{10}	0232_4

Tablica 1 Kodiranje znakova abecede niza S

Niz znakova se zatim raspisuje na sljedeći način:

$$-= 0231_4$$
$$B = 1002_4$$
$$B = 1002_4$$
$$D = 1010_4$$
$$a = 1201_4$$
$$p = 1300_4$$
$$p = 1300_4$$
$$a = 1201_4$$
$$. = 0232_4$$
$$H = 1020_4$$
$$P = 1100_4$$
$$D = 1010_4$$
$$a = 1201_4$$
$$-= 0231_4$$

Nakon toga se znakovi raspisuju po slojevima na način da svi znakovi kodnih riječi koji se nalaze na prvom mjestu pripadaju *k*-toj razini pri čemu je *k* broj najviše razine. Svi znakovi koji se nalaze na drugom mjestu pripadaju *k-1* razini itd. Kao rezultat se dobiva sljedeće:

Razina	Znakovni niz
3	01111111011110
2	20002332201022
1	30010000320103
0	12201001200011

Tablica 2 Znakovi po razinama

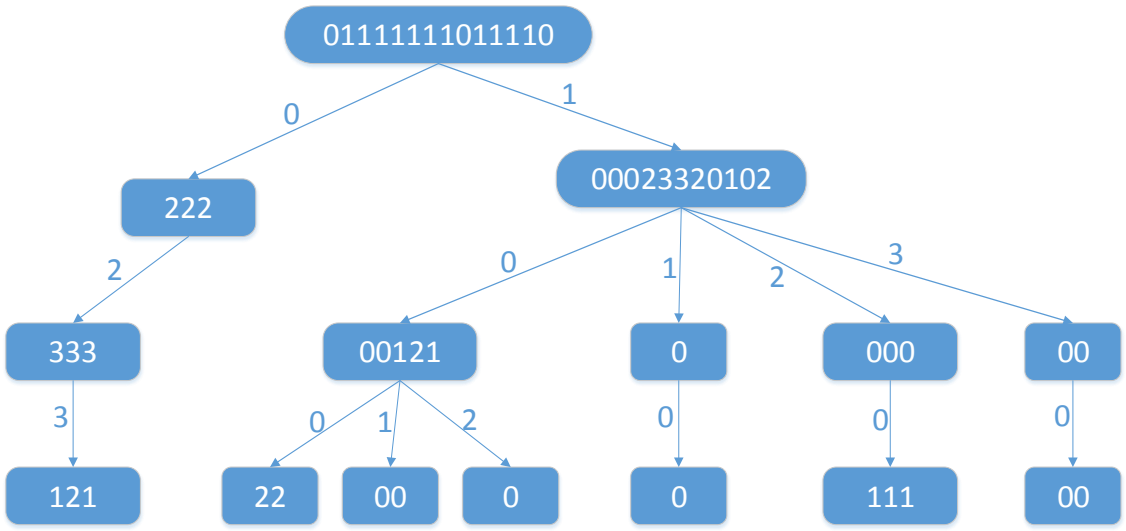
Koraci izgradnje stabla su sljedeći:

1) Znakovni niz *k*-te razine se dodijeljuje korijenskom čvoru.
2) Korijenski čvor može imati onoliko djece kolika je kratnost stabla. Svaki znak *k-1* razine se stavlja u onaj list koji ima kardinalost jednaku vrijednosti znaka koji se nalazi na istom mjestu na razini *k*.
3) Korak 2) se ponavlja dalje za sve razine, pri čemu se uvijek gleda jedna razina više od one za koju dodijeljujemo znakove listovima stabla. Također, znak se dodijeljuje uvijek onom listu do kojeg je moguće doći preko čvorova koji sadrže znakove koji se nalaze na istom mjestu na svim višim razinama.

Nadalje, ovo nije jedini način kreiranja stabla. Stablo je također moguće kreirati na sljedeći način:

1) Učita se jedan znak ulaznog niza
2) Za pročitani ulazni znak se dohvaća kodna riječ
3) Prva znamenka kodne riječi se sprema u korijenski čvor
4) Sljedeća znamenka kodne riječi se sprema u dijete koje se nalazi na grani koja ima težinsku vrijednost jednaku vrijednosti prethodne znamenke kodne riječi.
5) Korak 4) se ponavlja sve znamenke kodne riječi

Za oba navedena postupka dobije se stablo prikazano na sljedećoj slici (Slika 1). Na slici, radi preglednosti, nisu prikazani čvorovi u kojima nema podataka.



Slika 1 Konačni izgled stabla za niz S

1.2. Rank

Operacija *rank* računa koliko se puta traženi znak pojavljuje do određenog mjesta u nizu. Ako je na primjeru iz prethodnog poglavlja (Poglavlje 1.1) potrebno odrediti koliko se puta pojavljuje slovo „*B*“ do petog mjesta, postupak je sljedeći:

$$S = -BBDappa.HPDa -$$

$$rank\ (5, B) = ?$$

$$B = 1002_4$$

1) Prvo se računa koliko znakova „1“ ima u korijenskom čvoru do petog mjesta.

$$rank(5, 1) = 4$$

2) Rezultat dobiven u prethodnom koraku predstavlja novo mjesto do kojeg se računa *rank*, a sljedeća znamenka je znak koji se traži. *Rank* se računa za čvor na sljedećoj razini, i to onaj čvor koji se nalazi na grani koja ima težinsku vrijednost jednaku vrijednosti znamenke za koju se izračunao *rank* (u ovom primjeru je težinska vrijednost 1).

$$rank(4, 0) = 3$$

3) Ponavlja se korak 2) sve dok ima znamenaka u kodnoj riječi.

$$rank(3, 0) = 2$$

$$rank(2, 2) = 2$$

Rezultat zadnjeg računanja ranka odnosno za najniži list u stablu predstavlja konačan rezultat. U ovom primjeru je konačan rezultat 2, što znači da do petog mjesta u ulaznom nizu postoje dva znaka „*B*“. Ako se pogleda vrijednost ulaznog niza *S*, vidljivo je da je dobiven točan rezultat.

1.3. Select

Operacija select računa na kojem se mjestu nalazi *i*-ti traženi znak. Ako je na primjeru iz poglavlja 1.1. potrebno odrediti na kojem se mjestu nalazi drugi znak „*a*“, postupak je sljedeći:

$$S = -BBDappa.HPDa -$$

$$select(2, a) = ?$$

$$a = 1201_4$$

1) Prvo je potrebno pronaći list, na najnižoj razini stabla, koji sadrži zadnju znamenku kodne riječi slova „*a*“
2) U pronađenom listu se računa na kojem se mjestu nalazi druga po redu tražena znamenka (zadnja znamenka kodne riječi koja u ovom slučaju iznosi 1)

$$select(2, 1) = 1$$

3) Prelazi se na čvor više razine, odnosno roditelj čvora za koji se računao *select*. U tom čvoru se traži znamenka koja u kodnoj riječi prethodi znamenci za koju se računao *select*. Broj znamenaka koji se traži je rezultat izračunatog *selecta* uvećan za jedan.

$$select + 1 = 2$$

$$select(2, 0) = 1$$

4) Ponavlja se korak 3) sve do korijenskog čvora odnosno početka kodne riječi.

$$select(2, 2) = 6$$

$$select(7, 1) = 7$$

Konačan rezultat predstavlja vrijednost zadnjeg *selecta* odnosno *selecta* za korijenski čvor. U ovom slučaju je to 7, što se može potvrditi ako se pogleda ulazni niz S (potrebno je brojati mjesta od nule).

2. Testiranje

Ulazni nizovi spremljeni u datotekama *'input_i'*, pri čemu je i broj od 1 do 10, generirani su korištenjem Python skripte. Nizovi predstavljaju dijelove genoma bakterije *Escherichie coli*. Genom je preuzet na stranici [2]. Datoteke i skripta se nalaze u direktoriju *'generate_inputs'*.

2.1. Programski jezik C (Jelena Konfic)

2.1.1. Analiza točnosti

Uspoređujući rezultate koje vraćaju funkcije *rank* i *select*, zaključeno je kako je stablo ispravno izgrađeno. Osim rezultata navedenih funkcija, napravljena je i usporedba s tekstualnim uređivačem *Notepad++* te pomoću opcije pretrage u navedenom programu, provjereni su rezultati koje vraćaju funkcije *rank* i *select*.

2.1.2. Vrijeme izvođenja i količina zauzete memorije

Testni slučajevi pokrenuti su na *Bio-Linuxu 8* (64-bit). *Bio-Linux* je pokrenut na virtualnoj mašini kojoj je dodijeljeno 2048 MB RAM-a i 40 GB podatkovnog prostora.

U tablicama (Tablica 3 i 4) su prikazani rezultati testiranja. Testiranje se radilo za stablo kratnosti 3 i stablo kratnosti 4. Za računanje *ranka* je uvijek uzeta maksimalna vrijednost odnosno duljina niza ulazne datoteke. Za računanje *selecta* je uzeta vrijednost koju je vratio *rank* za taj primjer. Za obje operacije se tražio znak „*A*“ .Memorijsko zauzeće se gleda nakon što se kreira stablo.

Datoteka	Duljina niza	Kreiranje stabla (s)	Rank(max) (s)	Select(max) (s)	Ukupno memorijsko zauzeće (KB)
Input_1	120	0.00006	0.00001	0.00001	328
Input_2	540	0.00025	0.00002	0.00002	328
Input_3	1020	0.00042	0.00004	0.00004	328
Input_4	5040	0.00194	0.00017	0.00019	464
Input_5	10020	0.00371	0.00036	0.00034	604
Input_6	50040	0.02302	0.00159	0.00174	1628
Input_7	100020	0.04276	0.00322	0.00419	2912
Input_8	250020	0.08901	0.00493	0.00778	5848
Input_9	500040	0.15731	0.01706	0.01788	10472
Input_10	1000020	0.28663	0.03411	0.03345	20748

Tablica 3 Vrijeme izvođenja i memorijsko zauzeće za stablo kratnosti 3

Datoteka	Duljina niza	Kreiranje stabla (s)	Rank(max) (s)	Select(max) (s)	Ukupno memorijsko zauzeće (KB)
Input_1	120	0.00005	0.00001	0.00001	328
Input_2	540	0.00017	0.00002	0.00002	328
Input_3	1020	0.00031	0.00004	0.00004	328
Input_4	5040	0.00143	0.00019	0.00019	328
Input_5	10020	0.00274	0.00032	0.00034	464
Input_6	50040	0.01431	0.00167	0.00169	1304
Input_7	100020	0.02624	0.00312	0.00391	2028
Input_8	250020	0.05915	0.00844	0.00885	4504
Input_9	500040	0.11156	0.01741	0.01762	8524
Input_10	1000020	0.20681	0.03398	0.03487	16824

Tablica 4 Vrijeme izvođenja i memorijsko zauzeće za stablo kratnosti 4

Iz rezultata je vidljivo da je vrijeme izvođenja i memorijsko zauzeće veće što je veći ulazni niz znakova. Isto je očekivano s obzirom na broj znakova koje je potrebno dodati u stablo. Nadalje, kao što je već spomenuto u prvom poglavlju, veća kratnost stabla povećava brzinu izvođenja. Isto je potvrđeno ovim rezultatima. Vidljivo je da je vrijeme izvođenja za ulazni niz od 1000020 za kratnost 3 iznosilo 0.28663, a za kratnost 4 0.20681. Vrijeme izvođenja operacija *rank* i *select* je također kraće u tom slučaju te je memorijsko zauzeće manje. Stoga, u slučajevima kada je potrebno obraditi velike ulazne nizove, bolje je koristiti stablo s većom kratnošću. Za manje ulazne nizove isto nije potrebno jer neće biti velike razlike u brzini izvođenja niti u zauzeću memorije.

2.2. Python (Marin Grmača)

2.2.1. Analiza točnosti

Za kraće ulazne nizove za koje je moguće ručno izgraditi stablo, *rank* i *select* su vraćali ispravne rezultate. Za duže ulazne nizove, rezultati koje je program vratio provjereni su s programom *Notepad++* i podudarali su se. Osim toga, funkcije *rank* i *select* su međusobno inverzne stoga je dodatna provjera točnosti bila usporedba tih rezultata koji su bili komplementarni.

2.2.2. Vrijeme izvođenja i količina zauzete memorije

10 testnih slučajeva izvedeno je na 64-bitnom *Bio-linuxu 8* koji je pokrenut na virtualnoj mašini uz pomoć programa *VirtualBox*. Virtualnoj mašini dodijeljeno je 2048 MB RAM-a i 2 procesora te 40 GB podatkovnog prostora.

Datoteka	Duljina niza	Kreiranje stabla (s)	Rank(max) (s)	Select(max) (s)	Ukupno memorijsko zauzeće (KB)
Input_1	120	0.0012	0.0001	0.0001	7164
Input_2	540	0.0045	0.0003	0.0004	7164
Input_3	1020	0.0092	0.0005	0.0007	7164
Input_4	5040	0.0529	0.0022	0.0044	7340
Input_5	10020	0.1072	0.0059	0.0072	7348
Input_6	50040	0.4694	0.0336	0.0464	9440
Input_7	100020	0.9517	0.0763	0.0876	11456
Input_8	250020	2.1676	0.1783	0.1962	17100
Input_9	500040	4.4086	0.3212	0.3931	27192
Input_10	1000020	8.5194	0.6423	0.7675	47116

Tablica 5 Vrijeme izvođenja i memorijsko zauzeće za stablo kratnosti 3

Datoteka	Duljina niza	Kreiranje stabla (s)	Rank(max) (s)	Select(max) (s)	Ukupno memorijsko zauzeće (KB)
Input_1	120	0.0009	0.0001	0.0001	7164
Input_2	540	0.0042	0.0003	0.0004	7164
Input_3	1020	0.0073	0.0005	0.0007	7164
Input_4	5040	0.0376	0.0025	0.0034	7332
Input_5	10020	0.0764	0.0047	0.0069	7336
Input_6	50040	0.3457	0.0373	0.0431	8988
Input_7	100020	0.6872	0.0704	0.0753	10300
Input_8	250020	1.6683	0.1913	0.2142	15064
Input_9	500040	3.3357	0.2978	0.3709	23192
Input_10	1000020	6.7426	0.6349	0.7746	39412

Tablica 6 Vrijeme izvođenja i memorijsko zauzeće za stablo kratnosti 4

U tablicama 5. i 6. vidljiva su vremena izvođenja. Kratnost stabla je u prvoj tablici iznosila 3, a u drugoj 4. Vrijeme izgradnje stabla raste linearno s rastom duljine ulaznog niza. Iz tablica je vidljivo da je veća kratnost pridonijela manjem broju razina stabla i samim time poprilično ubrzala izgradnju stabla.

Rank i *select* operacije ispitane su za prvi simbol u ponuđenoj abecedi (A) te je u obzir uzeta maksimalna duljina niza. Vremena izvođenja *rank* i *select* operacija također su rasla linearno, a vrijeme *select* operacije je u većini slučajeva bilo duže od *rank* operacije jer se prilikom *select* operacije potrebno spustiti do najnižeg čvora u stablu i zatim ponovno vratiti do najvišeg čvora.

Kao što je vidljivo prema tablicama 5. i 6., ukupno memorijsko zauzeće se za najduži ulazni niz uz različitu kratnost uspjelo smanjiti za 7704 KB što ukazuje na još jednu korisnost veće kratosti stabla.

2.3. C++ (Kristijan Ivančić)

2.3.1. Analiza točnosti

Nad svim ulaznim podacima programska izvedba pokazuje ispravnu izgradnju stabla i točne rezultate za *rank()* i *select()* funkcije. Trenutno nisu implementirana ograničenja unosa te program za takve slučajeve daje netočan rezultat ili prekida izvođenje.

2.3.2. Vrijeme izvođenja i količina zauzete memorije

Testni slučajevi su izvedeni na operativnom sustavu *Bio-Linux* (x64) pokrenutom na virtualnoj mašini *VirtualBox*, sa dodijeljenih 2048MB RAM-a i 16GB prostora na tvrdom disku.

Dobiveni rezultati izračunati su uz kratnost 3 i 4 i prikazani u sljedećim tablicama (Tablica 7 i 8).

Datoteka	Duljina niza	Kreiranje stabla (s)	Rank(max) (s)	Select(max) (s)	Ukupno memorijsko zauzeće (KB)
Input_1	120	0.000504	0.000104	0.000075	1176
Input_2	540	0.00097	0.000118	0.000175	1176
Input_3	1020	0.001323	0.000242	0.00023	1176
Input_4	5040	0.01788	0.000275	0.000634	1700
Input_5	10020	0.020585	0.001106	0.001122	2476
Input_6	50040	0.056317	0.004011	0.00511	7528
Input_7	100020	0.117415	0.010237	0.007853	13828
Input_8	250020	0.255871	0.01825	0.01822	31936
Input_9	500040	0.498136	0.028502	0.029542	65088
Input_10	1000020	1.01339	0.045728	0.045701	126088

Tablica 7 Vrijeme izvođenja i memorijsko zauzeće za stablo kratnosti 3

Datoteka	Duljina niza	Kreiranje stabla (s)	Rank(max) (s)	Select(max) (s)	Ukupno memorijsko zauzeće (KB)
Input_1	120	0.000869	0.000073	0.000053	1176
Input_2	540	0.003909	0.000096	0.000229	1172
Input_3	1020	0.004156	0.000111	0.000237	1176
Input_4	5040	0.015157	0.000187	0.000656	1700
Input_5	10020	0.022102	0.001814	0.001173	2212
Input_6	50040	0.059519	0.006293	0.005361	6736
Input_7	100020	0.107305	0.008894	0.011491	12680
Input_8	250020	0.231459	0.020354	0.01951	29144
Input_9	500040	0.443381	0.028308	0.029236	57768

Input_10	1000020	0.889016	0.044191	0.048149	115316

Tablica 8 Vrijeme izvođenja i memorijsko zauzeće za stablo kratnosti 4

Vrijeme izgradnje stabla u ovoj implementaciji raste s veličinom ulaznog niza. Nakon što je stablo izgrađeno, isto se pretražuje vrlo brzo. Vrijeme izgradnje stabla pada kako kratnost raste, isto kao i memorijsko zauzeće.

2.4. Perl (Nikola Munđer)

2.4.1. Analiza točnosti

Programska izvedba je pokazala ispravnost nad svim ulaznim ispitnim podacima. I *select* i *rank* funkcija su pokazale točnost unutar odgovarajućeg vremena. Nažalost, u programskom kôdu ne postoje ograničenja za unos, tako da za nepostojeće unose i pogreške nema odgovarajućeg oporavka. Pri tome postoji mogućnost da funkcija nastavlja odrađivati posao, iako pravog rezultata nema.

2.4.2. Vrijeme izvođenja i količina zauzete memorije

Ispitni primjeri izvedeni su na virtualnoj mašini (*VirtualBox*) *Bio-Linux* (x64), sa dodijeljenih 1538MB RAM-*a*.

Datoteka	Duljina niza	Kreiranje stabla (s)	Rank(max) (s)	Select(max) (s)	Ukupno memorijsko zauzeće (KB)
Input_1	120	0.012	0.001	0.001	2384
Input_2	540	0.014	0.001	0.001	2516
Input_3	1020	0.016	0.010	0.004	2648
Input_4	5040	0.067	0.005	0.016	4020
Input_5	10020	0.089	0.009	0.012	5588
Input_6	50040	0.289	0.130	0.045	17928
Input_7	100020	0.536	0.075	0.222	33572
Input_8	250020	1.388	0.136	0.264	80528
Input_9	500040	2.645	0.420	0.613	158484
Input_10	1000020	5.202	1.221	1.149	315684

Tablica 9 Vrijeme izvođenja i memorijsko zauzeće za stablo kratnosti 3

Datoteka	Duljina niza	Kreiranje stabla (s)	Rank(max) (s)	Select(max) (s)	Ukupno memorijsko zauzeće (KB)
Input_1	120	0.012	0.001	0.001	2384
Input_2	540	0.014	0.001	0.001	2512

Input_3	1020	0.025	0.002	0.004	2648
Input_4	5040	0.068	0.030	0.054	3756
Input_5	10020	0.084	0.010	0.011	5284
Input_6	50040	0.258	0.167	0.042	16388
Input_7	100020	0.477	0.177	0.072	30460
Input_8	250020	1.211	0.193	0.255	72572
Input_9	500040	2.276	0.463	0.548	142340
Input_10	1000020	4.525	0.804	0.954	283212

Tablica 10 Vrijeme izvođenja i memorijsko zauzeće za stablo kratnosti 4

Vremena za navedene testne primjere su zadovoljavajuća. Razlog tome je korištenje hash tablica za stablo i korištenje skriptnog jezika za obradu. Za testirane kratnosti (3 i 4), vidi se mala razlika u brzini. Primjeri izvedeni sa kratnosti 4 pokazuju bržu izvedbu i izgradnje stabla i obrade podataka. Potrošnja memorije se znatno povećava sa povećanjem broja ulaznih znakova. Primjeri sa kratnosti 3 troše više memorije od primjera sa kratnosti 4. Uzrok razlikama memorije i brzine je razlika u izgrađenim stablima.

2.5. Ruby (Maja Buljan)

2.5.1. Analiza točnosti

Programska izvedba izgradnje stabla i izvođenja funkcija *rank* i *select* vratila je ispravne rezultate za sve ulazne ispitne podatke. Nažalost, programski kôd nema ograničenja za unos, tako ne postoji odgovarajući oporavak za pogrešne i nepostojeće unose. Stoga je moguće da se funkcija nastavlja izvoditi i u slučaju da nije moguće dobiti rezultat.

2.5.2. Vrijeme izvođenja i količina zauzete memorije

Ispitni primjeri izvedeni su na virtualnoj mašini (*VirtualBox*) *Bio-Linux* (x32), sa dodijeljenih 512MB RAMa.

Datoteka	**Duljina niza**	**Kreiranje stabla (s)**	**Rank(max) (s)**	**Select(max) (s)**	**Ukupno memorijsko zauzeće (KB)**
Input_1	120	0.007	0.001	0.001	9184
Input_2	540	0.025	0.012	0.010	9184
Input_3	1020	0.046	0.016	0.020	9184
Input_4	5040	0.245	0.064	0.075	9336
Input_5	10020	0.478	0.122	0.149	9908
Input_6	50040	2.611	0.479	0.687	14684
Input_7	100020	6.082	1.344	1.551	20660
Input_8	250020	14.652	2.367	3.222	41796

Input_9	500040	26.362	5.822	6.828	70172
Input_10	1000020	53.581	14.562	15.783	18712

Tablica 11 Vrijeme izvođenja i memorijsko zauzeće za stablo kratnosti 3

Datoteka	Duljina niza	Kreiranje stabla (s)	Rank(max) (s)	Select(max) (s)	Ukupno memorijsko zauzeće (KB)
Input_1	120	0.005	0.001	0.002	9184
Input_2	540	0.019	0.016	0.008	9184
Input_3	1020	0.035	0.033	0.028	9184
Input_4	5040	0.180	0.101	0.100	9336
Input_5	10020	0.358	0.186	0.189	9900
Input_6	50040	1.884	1.005	1.014	16020
Input_7	100020	3.915	1.785	1.803	21960
Input_8	250020	9.587	5.849	5.803	40260
Input_9	500040	19.264	8.490	10.610	73536
Input_10	1000020	37.461	17.908	18.078	44672

Tablica 12 Vrijeme izvođenja i memorijsko zauzeće za stablo kratnosti 4

Vremena za navedene testne primjere su odgovarajuća s obzirom na implementaciju u skriptnom jeziku, korištenjem hash tablica i polja.

2.6. Java (Marijana Spajić)

2.6.1. Analiza točnosti

Na temelju rezultata dobivenih izvođenjem funkcija *rank* i *select* zaključujemo da program daje točne rezultate. Da li je program 100% točan ne možemo sa sigurnošću odgovoriti, međutim za testirane nizove napravljena je i usporedba s tekstualnim uređivačem *Notepad++*, čime se pokazalo da su rezultati točni.

2.6.2. Vrijeme izvođenja i količina zauzete memorije

Testni slučajevi su pokrenuti na *Bio-Linuxu 8* (64-bit) koji je pokrenut na virtualnoj mašini kojoj je dodijeljeno 8 GB RAM-a i 20 GB podatkovnog prostora.

Testiranja su se provodila na 10 tekstualnih datoteka različitih duljina nizova. U oba slučaja se uzimala funkcija *rank* s parametrima maksimalnom duljinom niza i slovom A, dok se za funkciju *select* uzimala vrijednost koju je dala funkcija *rank* i slovo A. U prvom slučaju se uzimala kratnost 3, dok se u dugom slučaju uzimala kratnost 4. Memorijsko zauzeće se računa nakon što se kreira stablo kao razlika ukupne i slobodne memorije nakon kreiranja stabla.

U tablici (Tablica 13) su prikazani rezultati testiranja za kratnost 3.

Datoteka	Duljina niza	Kreiranje stabla (s)	Rank(max) (s)	Select(max) (s)	Ukupno memorijsko zauzeće (KB)
Input_1	120	0.049	0	0	682
Input_2	540	0.066	0.003	0.001	1366
Input_3	1020	0.094	0.002	0.004	1366
Input_4	5040	0.304	0.008	0.004	2046
Input_5	10020	0.388	0.011	0.016	3435
Input_6	50040	0.477	0.032	0.029	12603
Input_7	100020	0.443	0.027	0.065	24398
Input_8	250020	0.574	0.065	0.034	30199
Input_9	500040	0.838	0.037	0.029	34648
Input_10	1000020	1.310	0.060	0.082	72218

Tablica 13 Vrijeme izvođenja i memorijsko zauzeće za stablo kratnosti 3

Iz rezultata je vidljivo da vrijeme izgradnje stabla i ukupno memorijsko zauzeće rastu što je veća duljina niza, što je i očekivano. Vremena izvođenja operacija *rank* i *select* variraju, odnosno ne možemo reći da ovise o duljini niza, što je i očekivano s obzirom na njihov zadatak. Uspoređujući funkcije *rank* i *select*, na temelju rezultata, ne možemo zaključiti koja se od njih brže izvodi.

U tablici (Tablica 14) su prikazani rezultati testiranja za kratnost 4.

Datoteka	Duljina niza	Kreiranje stabla (s)	Rank(max) (s)	Select(max) (s)	Ukupno memorijsko zauzeće (KB)
Input_1	120	0.04	0	0	682
Input_2	540	0.058	0.002	0.003	682
Input_3	1020	0.082	0.003	0.005	1364
Input_4	5040	0.219	0.008	0.003	2046
Input_5	10020	0.266	0.011	0.011	2748
Input_6	50040	0.428	0.012	0.049	10209
Input_7	100020	0.439	0.016	0.055	18843
Input_8	250020	0.568	0.022	0.030	18048
Input_9	500040	0.964	0.034	0.039	32053
Input_10	1000020	1.159	0.041	0.051	49500

Tablica 14 Vrijeme izvođenja i memorijsko zauzeće za stablo kratnosti 4

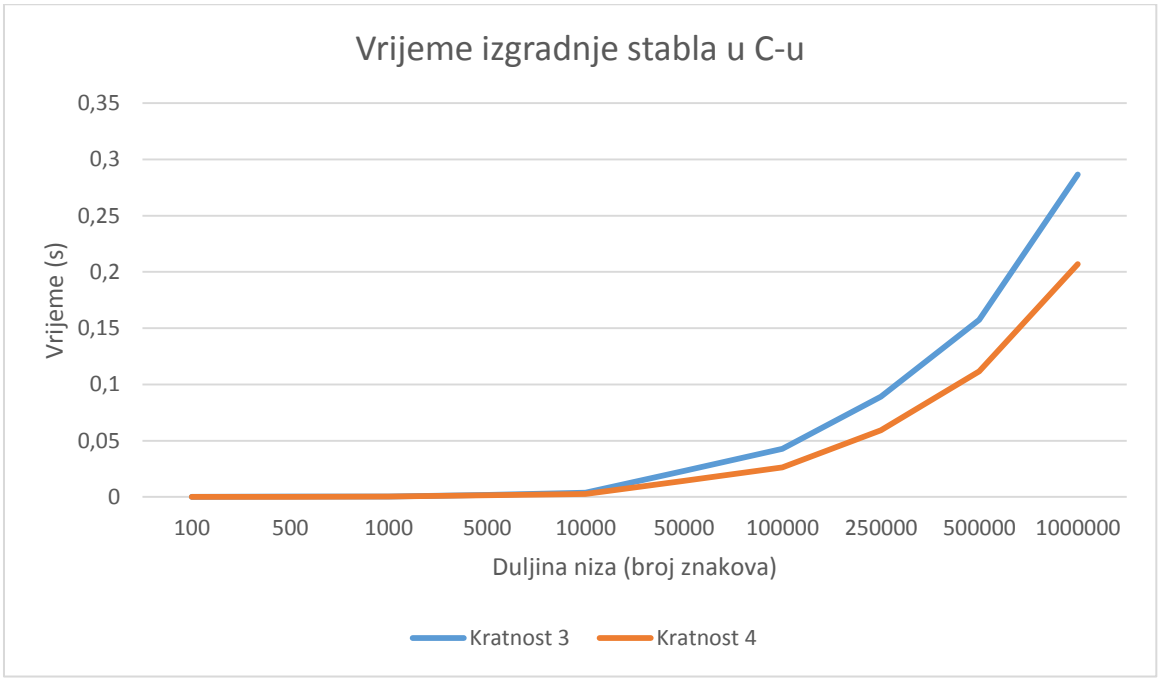
Kao i u prethodnom slučaju i u ovom vrijedi da vrijeme izgradnje stabla i ukupno memorijsko zauzeće rastu što je veća duljina niza. Vremena izvođenja operacija *rank* i *select* variraju, odnosno ne možemo reći da ovise o duljini niza. Uspoređujući funkcije vidimo da je vrijeme izvođenja funkcije *select* u većini slučajeva veće od vremena izvođenja funkcije *rank*. Takvi rezultati su očekivani jer funkcija *rank* prolazi kroz cijelo stablo dok ne dođe do zadnjeg čvora, a zatim se vraća nazad do početnog čvora (*root*).

Uspoređujući rezultate za kratnost 3 i kratnost 4 vidimo da je za kratnost 4 vrijeme izgradnje stabla kraće, a ukupno memorijsko zauzeće manje. U pravilu su i funkcije *rank* i *select* brže kada se gradi stablo s kratnosti 4. Zaključujemo da što je veća kratnost to je vrijeme izgradnje stabla kraće, a memorijsko zauzeće manje. Također, funkcije *rank* i *select* se izvode kraće uz veću kratnost.

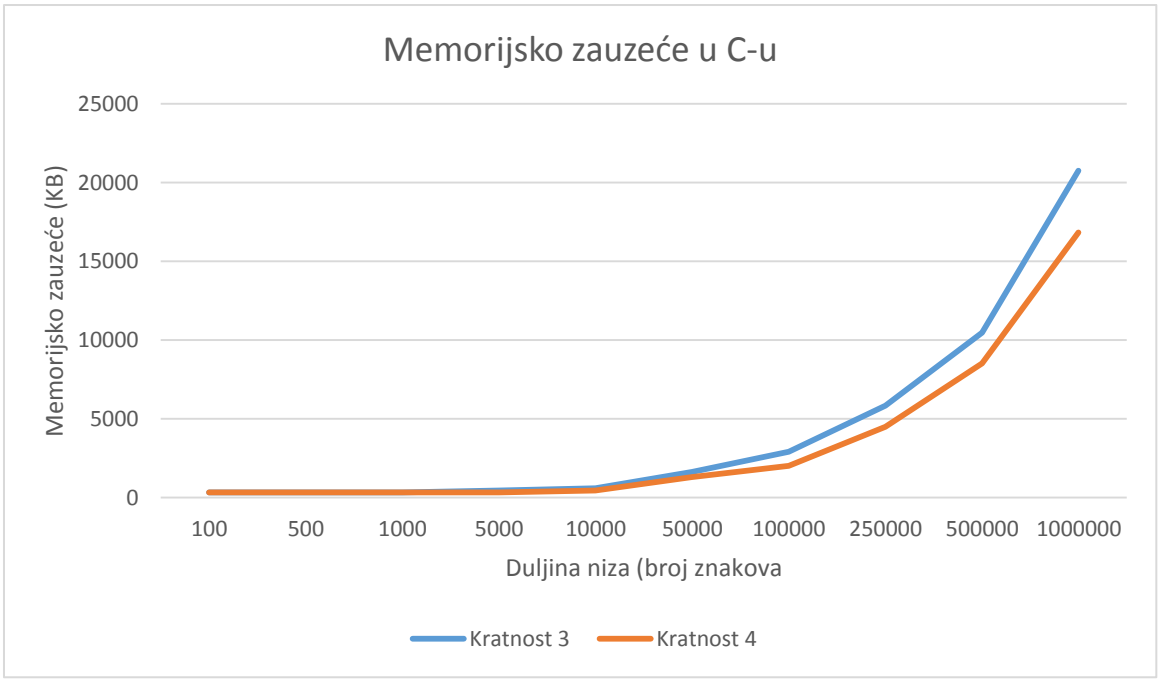
2.7. Usporedba rezultata testiranja

Kako bi se mogla napraviti usporedba rezultata testiranja, testiranje pojedinih implementacija je uvijek rađeno za iste ulazne nizove podataka (dostupne u direktoriju „*generate_inputs*“). Mjerila se brzina izgradnje stabla, brzina izvođenja operacija *rank* i *select* te memorijsko zauzeće nakon izgradnje stabla. Navedena mjerenja su se vršila za stabla kratnosti 3 i 4 kako bi se mogao odrediti utjecaj kratnosti stabla na brzinu izvođenja i memorijsko zauzeće. Za računanje *ranka* je uvijek uzeta maksimalna vrijednost odnosno duljina niza ulazne datoteke. Za računanje *selecta* je uzeta vrijednost koju je vratio *rank* za taj primjer. Za obje operacije se tražio znak „*A*“.

Na sljedećim grafovima je prikazan utjecaj kratnosti stabla na vrijeme izgradnje stabla i memorijsku potrošnju. Plavom crtom su prikazani rezultati za kratnost 3, a narančastom za kratnost 4.

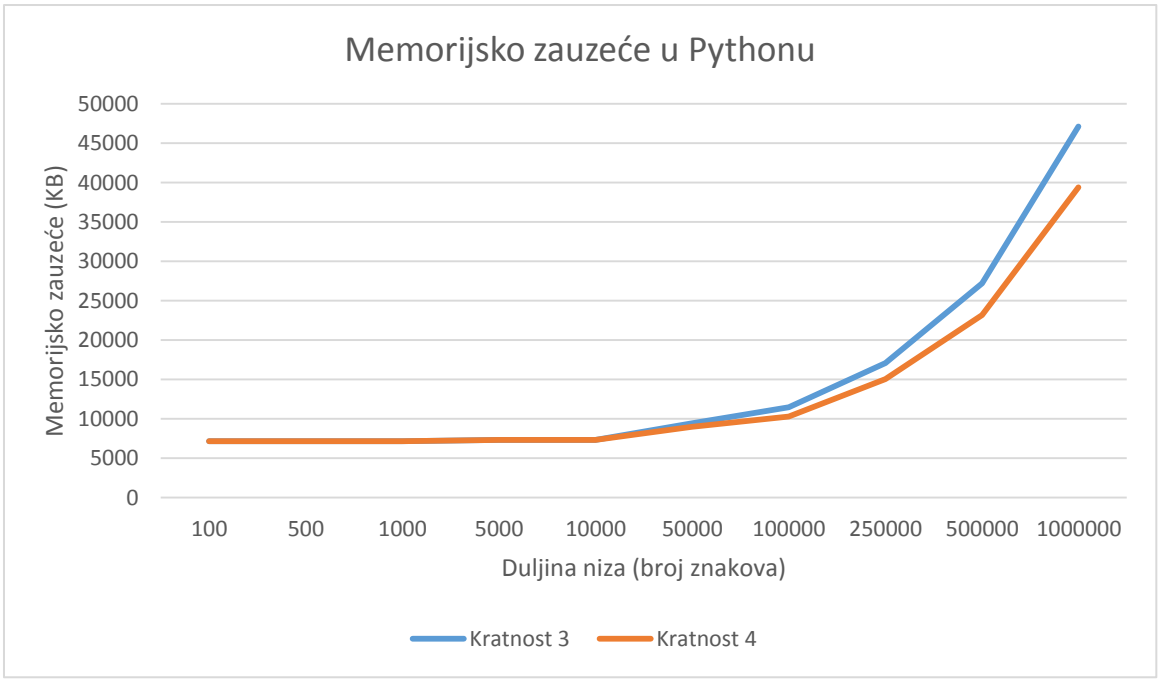


Graf 1 Vrijeme izgradnje stabla u C-u

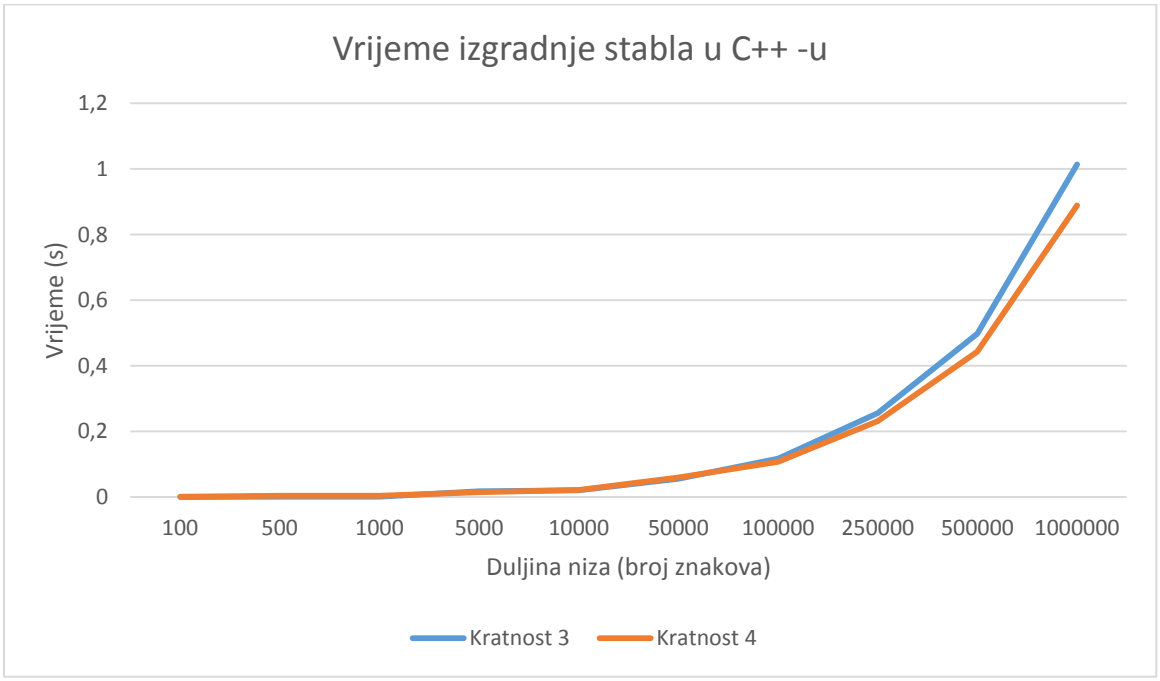


Graf 2 Memorijsko zauzeće u C-u

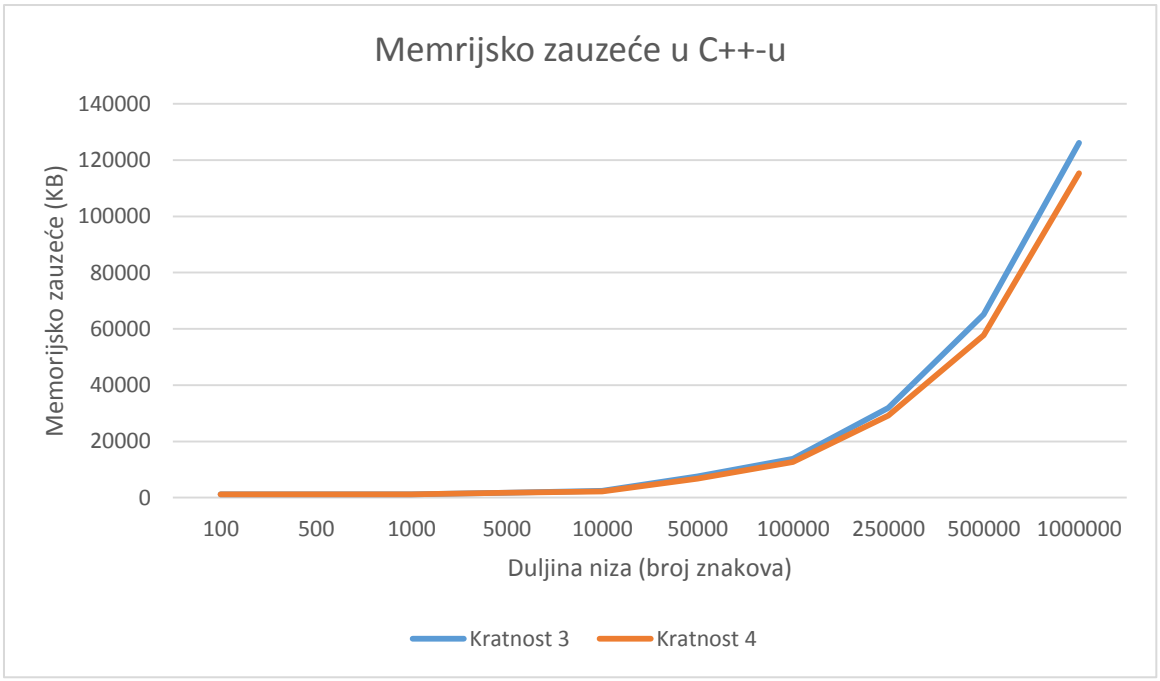
Graf 3 Vrijeme izgradnje stabla u Pythonu



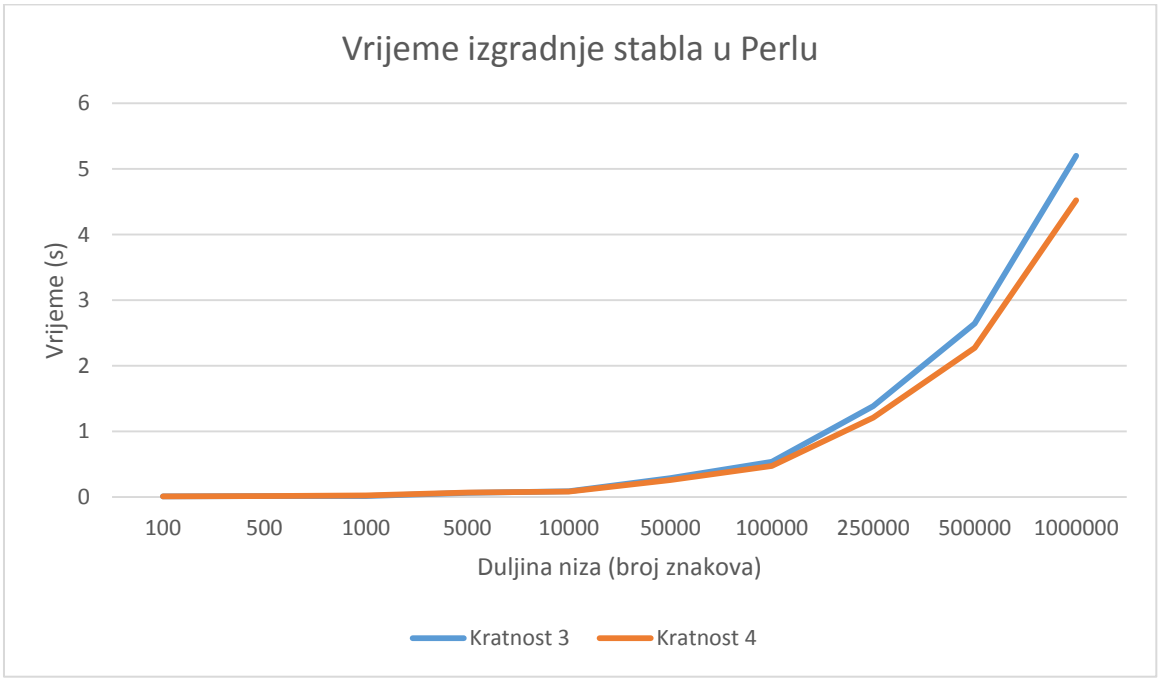
Graf 4 Memorijsko zauzeće u Pythonu



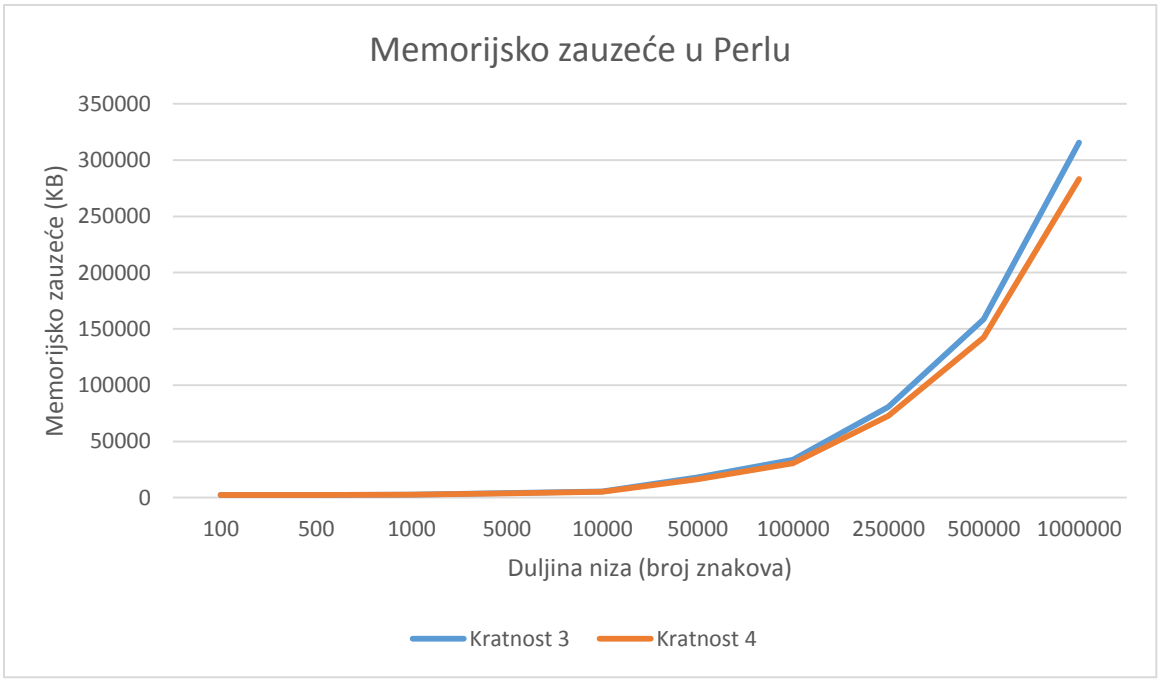
Graf 5 Vrijeme izgradnje stabla u C++-u



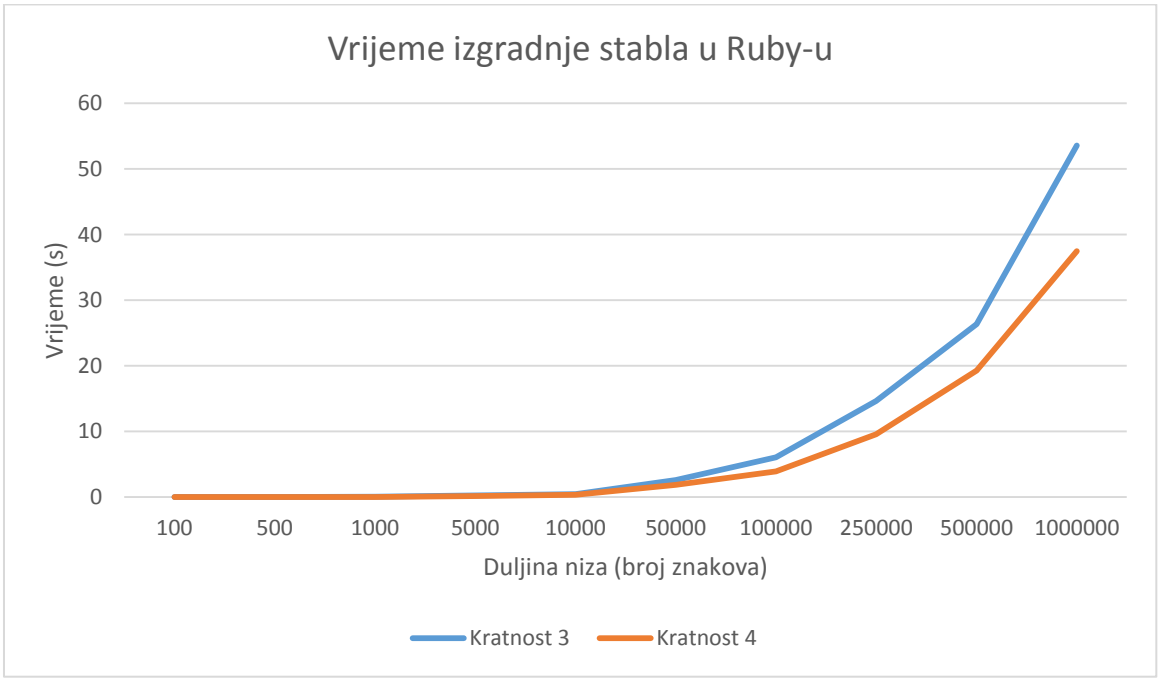
Graf 6 Memorijsko zauzeće u C++-u



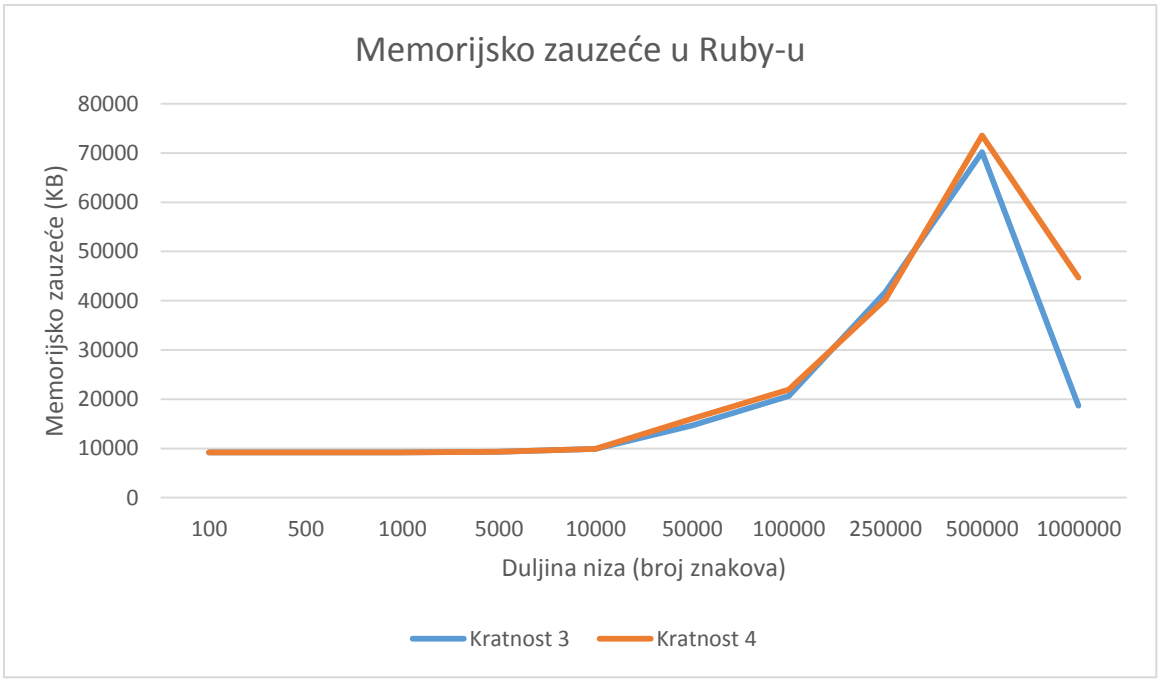
Graf 7 Vrijeme izgradnje stabla u Perlu



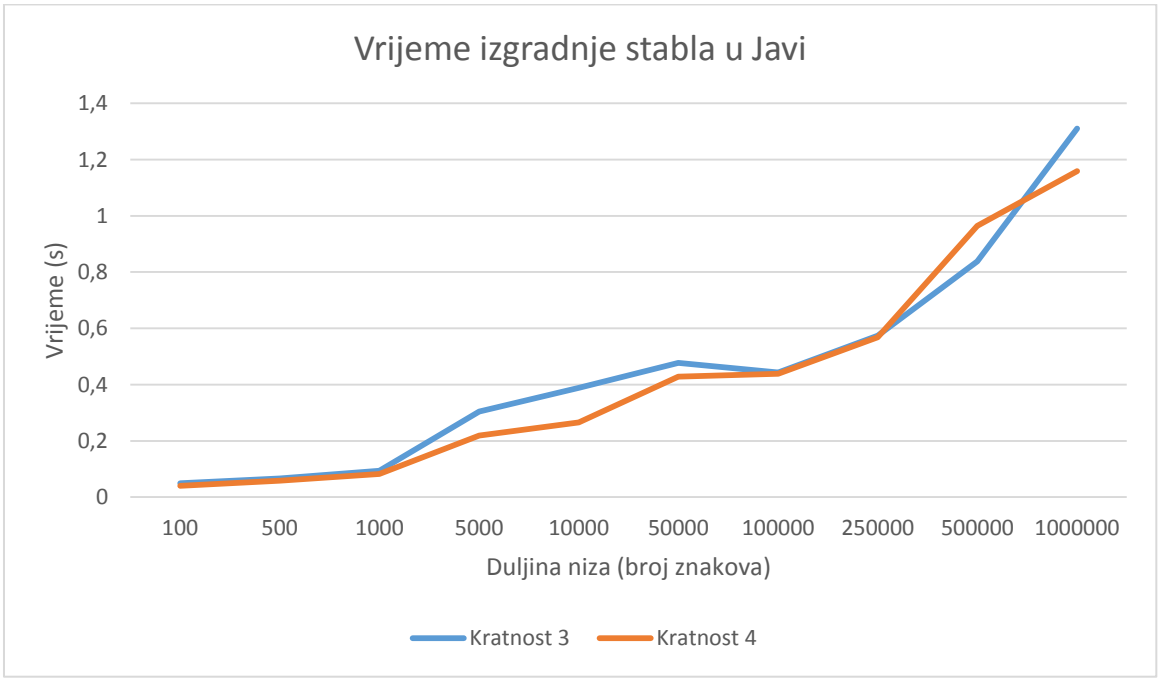
Graf 8 Memorijsko zauzeće u Perlu



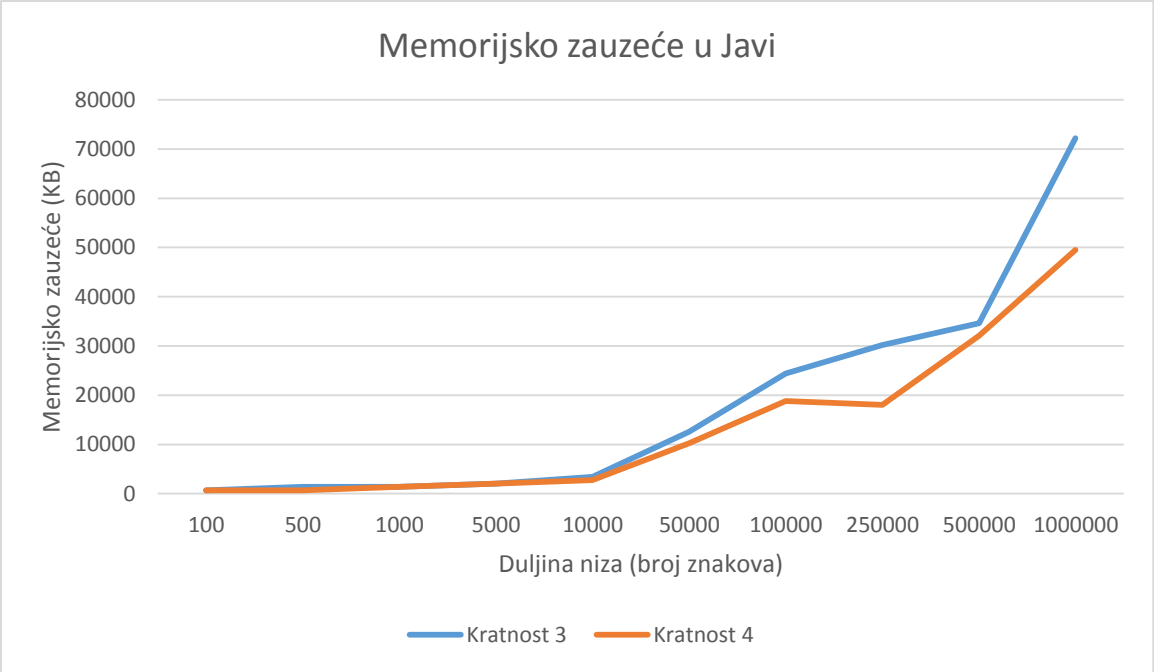
Graf 9 Vrijeme izgradnje stabla u Ruby-u



Graf 10 Memorijsko zauzeće u Ruby-u



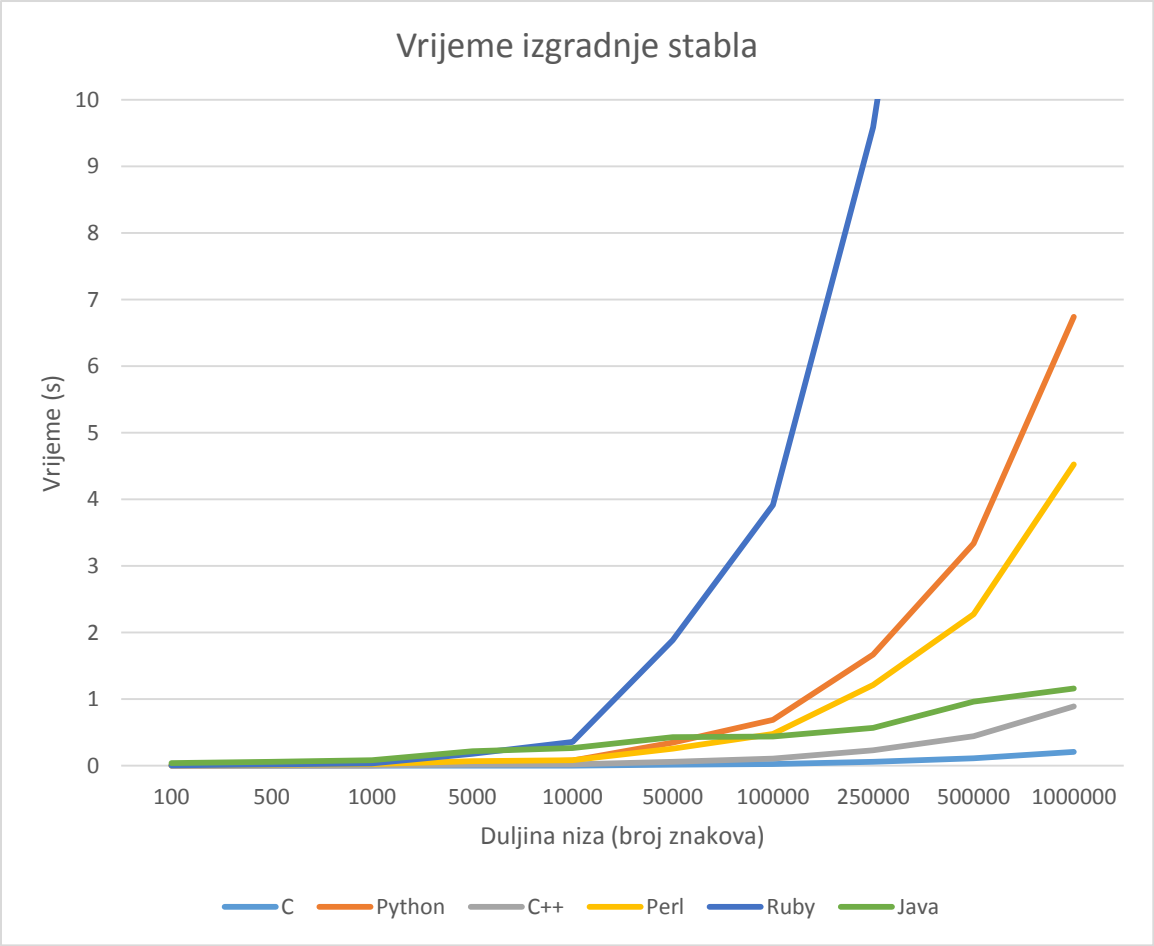
Graf 11 Vrijeme izgradnje stabla u Javi



Graf 12 Memorijsko zauzeće u Javi

Iz prethodnih grafova je vidljivo da veća kratnost skraćuje vrijeme potrebno za izgradnju stabla te smanjuje memorijsku potrošnju. Također je vidljivo da duljinom niza eksponencijalno raste i vrijeme izgradnje stabla i memorijsko zauzeće. U svim programskim jezicima je to potvrđeno, osim za programski jezik Java gdje dolazi do jednog odstupanja za niz duljine 500000 znakova. U tom primjeru je vrijeme izvođenja bilo veće za kratnost 4. Također, u programskom jeziku Ruby je u nekim slučajevima memorijska potrošnja veća za veću kratnost te je za niz duljine 1000000 znakova memorijska potrošnja manja nego za niz duljine 500000. Do navedenih odstupanja je vjerojatno došlo zbog načina implementacije.

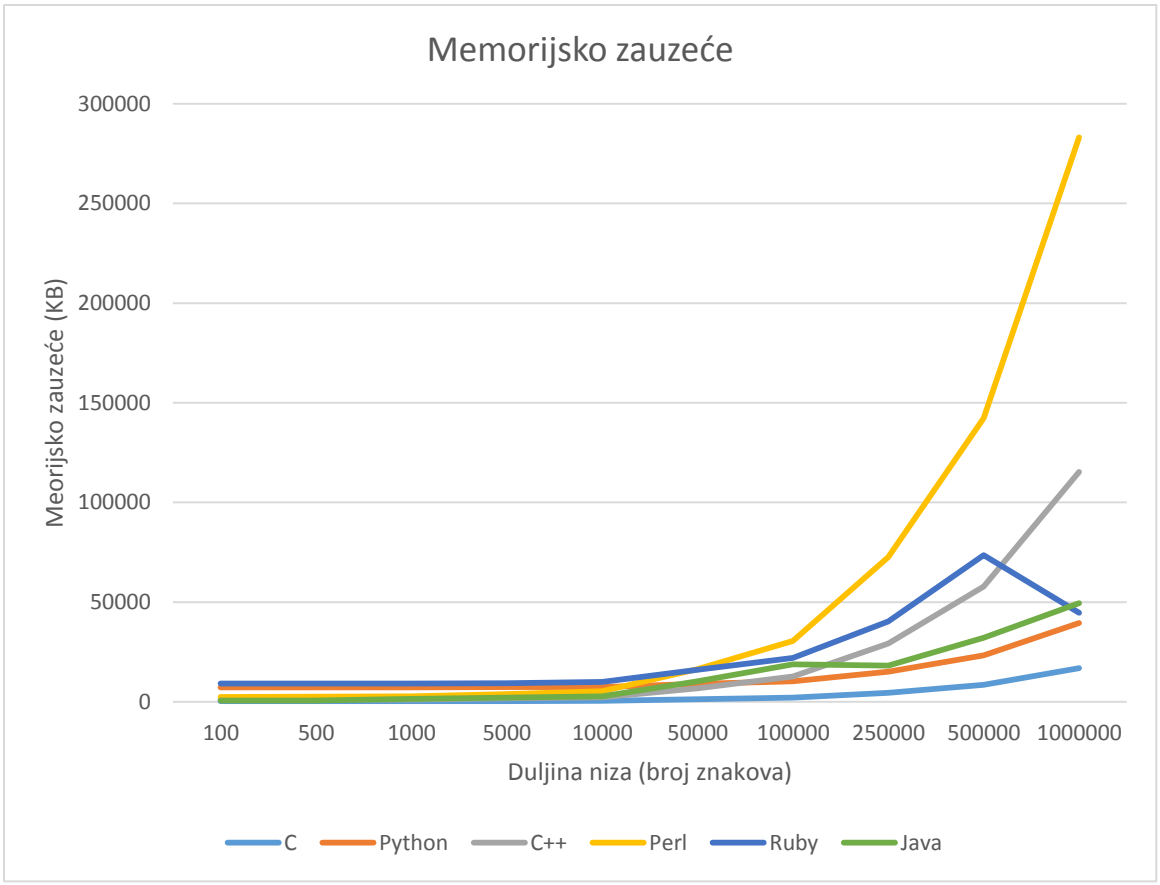
Na sljedećem grafu (Graf 13) su prikazana vremena izgradnje stabla za sve programske jezike u slučaju kada je kratnost stabla 4.



Graf 13 Vrijeme izgradnje stabla u različitim programskim jezicima

Iz grafa (Graf 13) je vidljivo da vrijeme izgradnje stabla raste s duljinom niza za sve programske jezike. Očekivano, stablo se najbrže izgradi u programskom jeziku C, nakon čega slijede C++ i Java. U skriptnim jezicima Python i Perl vrijeme izgradnje stabla je nešto duže za veće duljine niza nego što je to u C-u, C++-u i Javi. U programskom jeziku Ruby je vrijeme izgradnje stabla najduže te za ulazni niz od 1000000 znakova iznosi 37,461 s. Zbog preglednosti grafa je na Y osi prikazano samo vrijeme do 10 sekundi.

Na sljedećem grafu (Graf 14) je prikazana memorijska potrošnja za sve programske jezike u slučaju kada je kratnost stabla 4.



Graf 14 Memorijsko zauzeće za različite programske jezike

Iz grafa (Graf 14) je vidljivo da memorijska potrošnja raste s duljinom ulaznog niza, osim za jezik Ruby u kojem dolazi do odstupanja za najdulji niz. Memorijska potrošnja je najmanja u programskom jeziku C, nakon čega slijedi Python pa Java. Do veće memorijske potrošnje dolazi u programskim jezicima C++, Ruby i Perl. U programskom jeziku Perl je potrošnja čak preko 300 MB čemu je jedan od razloga način implementacije.

3. Zaključak

Brzina izgradnje višekratnog stabla valića uvelike ovisi o jeziku, ali i o načinu implementacije stabla. Najbrža izgradnja stabla te najkraće izvođenje operacija *rank* i *select* se, očekivano, izvršava u programskom jeziku C. S obzirom na to da programski jezik C ne podržava objektno orijentiranu paradigmu koja je jako prikladna za izgradnju višekratnog stabla valića, pisanje programa je bilo nešto teže od pisanja programa u jezicima koji podržavaju objektni način programiranja.

Ako se za način implementacije stabla odabere nekakva vrsta brzog pristupa memoriji (pr. pokazivači u C-u), takva implementacija će za velike ulazne nizove rezultirati brzim vremenom izvođenja. Kako bi se dodatno povećala brzina izgradnje stabla u svim programskim jezicima, povećava se kratnost stabla . Što je veća kratnost stabla to je kraće i vrijeme izgradnje stabla te je memorijska potrošnja manja. Utjecaj kratnosti je vidljiviji za duže ulazne nizove.

Literatura

[1] Multiary Wavelet Trees, https://bitbucket.org/vsmirnov/memoria/wiki/Multiary%20Wavelet%20Tree , s Interneta, 08. siječanj, 2015.

[2] Ensembl Bacteria, http://bacteria.ensembl.org/index.html , s Interneta, 10. siječanj, 2015.