UDC: 001.102:519.713 Info M: str. 36-42

ANALIZA VREMENA IZVRŠAVANJA ALGORITAMA ZA PRETRAŽIVANJE I UREĐIVANJE KOLEKCIJA PODATAKA
ANALYSIS OF SEARCHING AND SORTING DATA COLLECTIONS ALGORITHMS' EXECUTION TIME

Selena Matijević, Vojska Srbije,
selenamatijevic2@gmail.com
Prof. dr Saša D. Lazarević, FON,
slazar@fon.bg.ac.rs

REZIME: Uređivanje (sortiranje) je sveprisutno, bilo u svetu računara ili u svakodnevnom životu. To je proces preuređivanja elemenata nekog skupa po određenom poretku i preduslov za efikasno pretraživanje takvog skupa. Pretraživanje je proces koji za cilj ima pronalaženje elemenata u nekoj kolekciji podataka. Najčešće je to traženje elemenata koji sadrži određeni podatak, tj ključ. Postoji više algoritama kako za uređivanje, tako i za pretraživanje, pa se postavlja pitanje koji algoritam je optimalan u zavisnosti od parametara date kolekcije podataka. Kroz rad se posmatraju različite kolekcije podataka, a razlike se ogledaju u više kriterijuma. Najpre prema mestu koje kolekcija zauzima u memoriji: nizovi i liste kao delovi unutrašnje memorije i datoteke kao delovi spoljašnje memorije. Ovakve kolekcije prema nivou uređenosti mogu biti: uređene, inverzno uređene, delimično uređene i nasumične, i mogu biti različite veličine (1000, 100000 i 10000000 elemenata). Kroz aplikacije u C programskom jeziku implementirano je uređivanje i pretraživanje nizova, lista i datoteka, koje poseduju različite kombinacije dve pomenute karakteristike. Koristeći rezultate merenja izvršavanja pomenutih algoritama nad svakim tipom kolekcije, izvršena je analiza, u cilju dobijanja odgovora na pitanje koji algoritam je najbolje koristiti u odnosu na prirodu kolekcije podataka.

KLJUČNE REČI: Algoritam, Kolekcija podataka, Pretraživanje, Uređivanje, Merenje vremena, Komparativna analiza

ABSTRACT: In word of computer programming, sorting is present to a large degree, just like in everyday life. It is process of arranging elements of any set by certain order and precondition for efficient search of that set. Searching is process which goal is to find certain element in some data collection. It usually refers to search of element that contain some information (key). There are several sorting and searching algorithms, and the question is which one is optimal to use, depending on parameters of data collection. This paper considers different data collections, and those differences compete in more criterions. Firstly, data collections are divided by place in memory that they held into arrays and lists, as parts of internal memory, and files as parts of external memory. Further, these data collections are divided by level of sort in assorted, inverse assorted, partially assorted and random collections, and can be of different size (1000, 100000 and 10000000 elements). Searching and sorting of arrays, lists and files that have different combinations of mentioned characteristics are implemented in applications, using C programming language. Analysis of algorithms' execution time is done by using the results of searching and sorting measurements, with goal to get answer to question which algorithm is best to use depending on characteristics of data collection.

KEY WORDS: Algorithm, Data Collection, Searching, Sorting, Time Measurement, Comparative Analysis

1. UVOD

U kompjuterskom programiranju, kao i u svakodnevnom životu, uređivanje (sortiranje) je veoma česta aktivnost, koja ima bitnu ulogu. Značajno je lakše i brže pronaći neku stavku u sortiranoj kolekciji. Bilo koja praktična aplikacija u računarstvu zahteva to da stvari budu složene određenim redosledom. Potreba da se elementi slože od najmanjeg do najvećeg ili od prvog do poslednjeg ne sme se potceniti. Uređivanje je proces preuređivanja elemenata nekog skupa po određenom poretku i preduslov za efikasno pretraživanje takvog skupa. Pretraživanje treba da da informaciju o tome da li neka kolekcija podataka sadrži određeni element. Postoji više načina pretraživanja (sekvencijalno, binarno, interpolaciono, itd), kao i više algoritama za uređivanje. Prilikom poređenja različitih algoritama postoji nekoliko stvari koje treba uzeti u obzir: vremensko trajanje, zauzimanje memorijskog prostora, kompleksnost i stabilnost algoritma. Od ove četiri bitne karakteristike u ovom radu akcenat će biti na dužini trajanja izvršavanja algoritma.

Efikasnost manipulacije podacima u velikoj meri zavisi od toga ŠTA se uređuje i pretražuje. Prilikom suočavanja sa velikim kolekcijama podataka, neki algoritmi mogu biti previše spori za praktičnu realizaciju, pa time i neefikasni. Prema mestu na kome se podaci nalaze, razlikuju se unutrašnje uređivanje (pretraživanje), kada se podaci nalaze u operativnoj memoriji, i spoljašnje uređivanje (pretraživanje), koje se koristi za uređivanje podataka koji se nalaze na spoljašnjoj memoriji, u vidu datoteka. Pod kolekcijom podataka koja se uređuje, odnosno pretražuje, podrazumevaju se niz i lista, kao kolekcije nad kojima se vrši unutrašnje uređivanje, i datoteka, kao deo spoljašnje memorije. Sve tri navedene vrste kolekcije podataka određene su sledećim karakteristikama: stepen uređenosti (uređena, inverzno uređena, delimično uređena i nasumična kolekcija), i veličina (1000, 100000 i 10000000 elemenata).

Pošto je i uređivanje i pretraživanje moguće implementirati različitim algoritmima, cilj rada je da odgovori na pitanje koji algoritam je optimalan u zavisnosti od prirode kolekcije podataka, posmatrano sa aspekta vremena neophodnog da se on izvrši. Vreme pronalaženja nekog podatka je veoma bitan faktor u svakodnevnom životu. U tome se ogleda značaj dolaska do zaključka kojom metodom će se neka kolekcija podataka najpre urediti, a kasnije pretražiti, da bi se pravovremeno dobila povratna informacija.

2. PRETRAŽIVANJE

Pretraživanje je veoma česta aktivnost koja za cilj ima pronalaženje elemenata u nekoj kolekciji podataka, koji zadovoljava unapred definisani kriterijum. Najčešće je to traženje elementa koji sadrži određeni podatak, tj ključ. Pretraživana kolekcija može biti uređena ili neuređena. Prema mestu gde se kolekcija koja se pretražuje nalazi, razlikuju se unutrašnje pretraživanje (kolekcije koje se nalaze u operativnoj memoriji) i spoljašnje pretraživanje (kolekcije koje se nalaze u sekundarnoj memoriji - datoteka). U zavisnosti od tipa kolekcije podataka koja se pretražuje, algoritmi mogu biti manje ili više kompleksni i efikasni. Postoji više tipova pretraživanja, od kojih su u radu implementirana sledeća tri: sekvencijalno, binarno i interpolaciono.

3. UREĐIVANJE

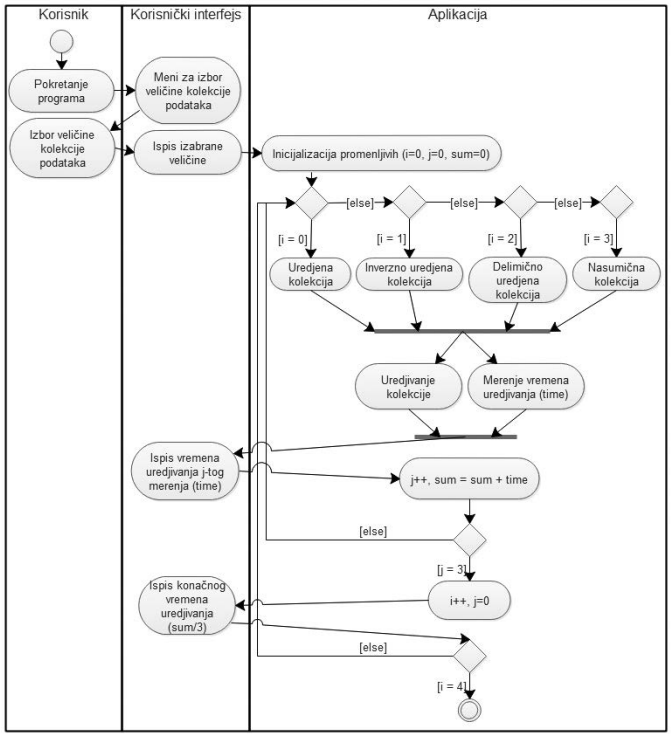
Uređivanje se može definisati kao postupak pri kojem se postojeći elementi kolekcije podataka ponovno raspoređuju prema željenom pravilu (najčešće po numeričkom ili abecednom redu). Efikasno uređivanje podataka je od bitne važnosti za optimizovanu upotrebu drugih algoritama (npr. pretraživanja), koji zahtevaju da ulazni podaci budu uređeni. Kroz aplikacije implementirano je sledećih osam algoritama za unutrašnje uređivanje: Bubble sort, Quick sort, Selection sort, Insertion sort, Radix sort, Shell sort, Heap sort, Merge sort, i algoritam za spoljašnje uređivanje: External Merge sort.

4. NIZOVI

Niz je kolekcija podataka sačinjena od fiksnog broja vrednosti istog tipa. Svaka ta vrednost se naziva elementom niza i ima svoj indeks, odnosno objekat preko kojeg mu se može pristupiti. C podržava samo proste nizove. Indeks prvog elementa je uvek 0, a poslednjeg n-1, za niz dimenzije n. Nizovi se u C-u implementiraju preko pokazivača. Ime niza je sinonim za adresu početnog (nulog) elementa niza. Nizovi se u funkciju uvek prenose preko adrese.

Aplikacija za uređivanje nizova

Aplikacija za uređivanje nizova izrađena je u C programskom jeziku, korišćenjem razvojnog alata Visual Studio 2013. Tehnologija izrade aplikacije je ista za sve naredne aplikacije, koje su sastavni deo ovog rada. Svaki od prethodno opisanih osam algoritama za uređivanje prolazi kroz deo algoritma sa slike 1 koji se tiče aplikacije.



Slika 1: *Algoritam za uređivanje*

Nakon korisnikovog izbora neke od ponuđenih veličina n, generišu se nizovi dimenzije n, četiri različite vrste (uređen, inverzno uređen, delimično uređen i nasumičan niz). Eksperimentalna pretpostavka je da je veličina elementa niza ceo broj (ključ), pa se može reći da se radi o uređivanju ključeva. Svaki izgenerisani niz se uređuje svakim od osam spomenutih algoritama, pri čemu se meri vreme koje protekne dok se uređivanje ne izvrši.

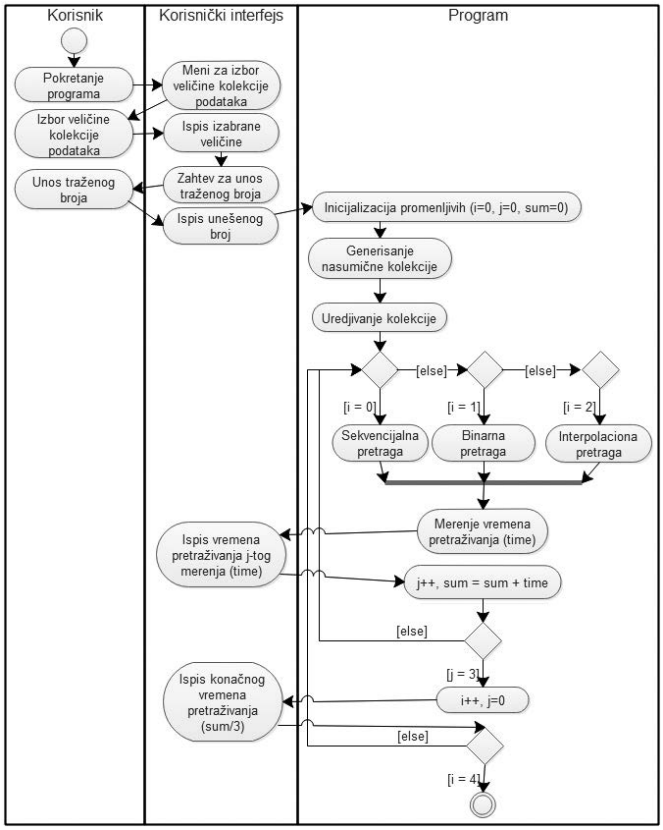
Merenje vremena uređivanja

Za meru vremenske efikasnosti nekog algoritma uzima se neposredno vreme izvršavanja za neke konkretne vrednosti. Operacije nad datumom i vremenom u C-u definisane su u standardnoj biblioteci, u zaglavlju time.h. Struktura clock_t je tip podoban za reprezentovanje vremena rada tekućeg procesa. Funkcija clock() je tipa clock_t i vraća vrednost procesorskog merača, koji startuje na početku programa, u jedinicama znatno manjim od sekunde (nekoliko milisekundi). Broj takvih jedinica u jednoj sekundi određen je konstantom CLOCKS_PER_SEC. U ovom slučaju, jedino ima smisla razmatrati razliku dva vremena dobijena ovom funkcijom. Razlika predstavlja broj vremenskih „tikova“, pa se broj sekundi između dva vremena dobija kao razlika dve vrednosti tipa clock_t podeljena konstantom CLOCKS_PER_SEC [7]. Na ovaj način se može izmeriti vreme koje je utrošio sam program, ili neki njegov deo. Precizniji rezultati se dobijaju ako se merenje izvršava više puta. Iz tog razloga, svako uređivanje određenim algoritmom je izvršeno tri puta, kao i svako merenje vremena izvršenja algoritma. Prosečna vrednost iz data tri merenja predstavlja konačno vreme izvršavanja algoritma. Rezultati svih merenja prikazuju se na korisničkom interfejsu, i čuvaju

u datoteci pod imenom result_[SatMinut]_[DanMesecGodina]. Isti postupak merenja i čuvanja rezultata merenja je primenjen u ostalim aplikacija.

Aplikacija za pretraživanje nizova

Na slici 2 nalazi se algoritam koji je upotrebljen za implementaciju pretraživanja kolekcija podataka. Prvi ulazni podatak aplikacije je broj članova niza n, koji korisnik definiše izborom iz menija koji se otvara po pokretanju programa. Zatim program formira nasumičan niz koji se potom uređuje optimalnom metodom. Dalje korisnik unosi broj za pretragu, i vrši se pretraživanje pomoću tri prethodno navedena algoritma. Istovremeno sa vršenjem pretraživanja meri se vreme neophodno da se pretraživanje izvrši. Postupak merenja je prethodno opisan.



Slika 2: *Algoritam pretraživanja*

5. LISTE

Lista je dinamička struktura podataka pogodna za predstavljanje nizova podataka. Elementi liste se nalaze na proizvoljnim mestima u memoriji i međusobno se povezuju pokazivačima. Lista omogućava efikasnu manipulaciju elementima (dinamičko stvaranje, brisanje i preuređivanje objekata u listi), ali je nedostatak taj što pokazivači zauzimaju dodatan memorijski prostor. Pristup elementima je sekvencijalan, a ne direktan, kao kod nizova. Poslednji element liste sadrži specijalan pokazivač koji ne ukazuje ni na jedan element. Glavna razlika lista u odnosu na nizove je ta što joj dužina nije unapred određena. Nizovi imaju prednost kada je potreban efikasan pristup proizvoljnom elementu, jer se pristup svakom elementu kod liste mora započeti od prvog elementa i sledeći njegovog sledbenika doći do drugog elementa i tako dalje, dok se ne stigne do željenog elementa.

Aplikacija za uređivanje lista

Logika za izradu aplikacija identična je kao i kod aplikacije sa nizovima. Veličina kolekcija podataka je ista, kao i njihova vrsta u odnosu na nivo sortiranosti. Korišćeni su isti algoritmi uređivanja implementirani u algoritam sa slike 1. Ono što je različito je vrsta kolekcije podataka i implementacija algoritama kroz funkcije, u skladu sa tim. Merenje vremena uređivanja algoritama se vrši na isti način kao što je prethodno opisano.

Aplikacija za pretraživanje lista

Aplikacija je konstruisana na osnovu algoritma sa slike 2, korišćenjem poznate tehnologije. Interakcija između korisnika i programa započinje se izborom broja članova liste n, na osnovu kojeg se formira i uređuje nasumična lista koja sadrži brojeve koji se ne ponavljaju. Zatim korisnik unosi broj za pretragu, i vrši se pretraživanje pomoću tri prethodno navedena algoritma (sekvencijalno, binarno i interpolaciono pretraživanje).

6. DATOTEKE

Datoteka je niz znakova (bajtova) kome je pridruženo ime, i koji se može interpretirati na razne načine. Na sadržaj datoteke tj. način na koji ga treba interpretirati, ukazuje ekstenzija uz ime datoteke. U programskom jeziku C datoteka je pre svega vezana za ulaz i izlaz podataka. Mogućnosti ulaza i izlaza nisu sastavni deo samog jezika C. Ovaj aspekt jezika podržan je funkcijama standardne biblioteke. Postoji više tipova organizacija datoteka, kao što su serijska, sekvencijalna, spregnuta, direktna, indeks-sekvencijalna, itd. To znači da postoji više načina za pristup sadržaju datoteke, a neki od tih su: sekvencijalni, direktan, indeksirani, itd. Tip datoteke razmatran u ovom radu je sekvencijalan, sa sekvencijalnim pristupom informacijama u njoj. To znači da se sadržaju pristupa redosledom kojim je on raspoređen unutar datoteke.

Aplikacija za uređivanje datoteka

Aplikacija za uređivanje datoteka nešto je jednostavnija od uređivanja nizova i lista, budući da je implementiran jedan algoritam uređivanja - External Merge sort. Upotrebljen je algoritam sa slike 1 za njeno kreiranje. Ulazni podatak je broj elemenata datoteke (n), koji korisnik bira iz ponuđene liste vrednosti u meniju (1000, 100000 ili 10000000). Zatim se generišu datoteke od n članova, četiri različite vrste nivoa uređenosti. Elementi datoteke su celobrojnog tipa. Merenje vremena izvršenja uređivanja datoteke External Merge sort algoritmom odigrava se na način već opisan.

Aplikacija za pretraživanje datoteka

Za pretraživanje datoteka upotrebljen je algoritam sekvencijalne pretrage. Kao što je prethodno napomenuto, u ovom radu je pretpostavka da je datoteka koja se pretražuje sekvencijalnog tipa, i da sadrži elemente ceolobrojnog tipa. Za implementaciju drugih tipova pretraživanja bilo bi noephodno kreiranje dopunskih struktura podataka ili korišćenje drugog tipa datoteke. Algoritmom predstavljenim na slici 2 opisan je proces konstrukcije ove aplikacije. Nisu implementirane sve funkcije prikazane na algoritmu, već samo onaj deo koji se tiče sekvencijalne pretrage. Nakon startovanja programa, korisnik najpre bira veličinu datoteke n, a zatim unosi broj koji želi da pronađe. Sledi formiranje datoteke koja sadrži nasumične brojeve i nosi naziv external_[n]. Ona se zatim uređuje, i tako uređena sekvencijalno pretražuje.

7. ANALIZA VREMENA IZVRŠAVANJA ALGORITAMA

Iz opisanih šest aplikacija u kojima su implementirani algoritmi za uređivanje i pretraživanje kolekcija podataka, dobijen je veliki broj izlaznih podataka u vidu vremena izvršavanja algoritama. Ti podaci su veoma bitni, jer se iz njih mogu izvući zaključci koji se tiču relacije između kolekcije podataka i algoritma koji bi trebalo nad njom upotrebiti, bilo za njeno uređivanje ili pretraživanje.

7.1. *Nizovi*

Uređivanje

Pod tim se podrazumeva uređivanje nizova koji, eksperimentalnom pretpostavkom veličine elementa niza, sadrže ključeve. U tabeli 1 su prikazana vremena uređivanja već uređenog niza od 1000, 100000 i 10000000 članova.

Veličina / Algoritam	1000	100000	10000000
Bubble sort	2	19078.66602	45764276
Quick sort	0	15.666667	1894
Selection sort	2	19079.66602	46809732
Insertion sort	0	0.666667	13
Radix sort	0	20.333334	3044.666748
Merge sort	1	87	9606.333008
Shell sort	0	8	1219.666626
Heap sort	0.666667	35.333332	5165

Tabela 1: *Vreme uređivanja uređenog niza (u ms)*

Primetno je da se niz manjih dimenzija (1000 elemenata) može bez problema urediti bilo kojim od algoritama, kao i da su razlike između pojedinih algoritama (Quick, Insertion, Radix i Shell sort) zanemarljive, jer su reda veličine manje od milisekunde. Kako raste dimenzija niza tako su razlike u performansama algoritama uočljivije. Iz tabele se vidi da se veći uređeni nizovi (100000 i 10000000 elemenata) najefikasnije mogu urediti Insertion sort algoritmom.

U tabeli 2 se vidi da inverzno uređen niz zahteva više vremena da bi se uredio. To je najgori mogući slučaj stanja niza koji želi da se uredi. Slično, niz manje dimenzije može se urediti korišćenjem više algoritama, dok Radix sort algoritam pokazuje najbolje osobine prilikom uređivanja većih inverznih nizova.

Veličina / Algoritam	1000	100000	10000000
Bubble sort	3.666667	33927.33203	79252584
Quick sort	0	24.666666	15618.33301
Selection sort	2	19138.33398	58490280
Insertion sort	2.333333	20053	72506944
Radix sort	0	17.666666	1748.333374
Merge sort	1	93.666664	10662
Shell sort	0.333333	45.333332	10262.33301
Heap sort	0.333333	38.333332	6262.666504

Tabela 2: *Vreme uređivanja inverzno uređenog niza (u ms)*

Delimično uređen niz od 1000 članova se efikasno može urediti svakim algoritmom, što se vidi u tabeli 3. Shell sort algoritam najbolje je upotrebiti za uređivanje ovakvih nizova većih dimenzija.

Veličina / Algoritam	1000	100000	10000000
Bubble sort	2	19135	54960088
Quick sort	0	17.666666	14996.33301
Selection sort	2	19128.66602	56885260
Insertion sort	0	2390	18182652
Radix sort	0	17.333334	1729.333374
Merge sort	1	86	9595.666992
Shell sort	0	8	657
Heap sort	2	19135	54960088

Tabela 3: *Vreme uređivanja delimično uređenog niza (u ms)*

Iz vremena uređivanja nasumičnog niza prikazanih u tabeli 4 može se očekivano zaključiti da se niz manjih dimenzija efikasno uređuje bilo kojim algoritmom. Nasumičan niz od 100000 ili 10000000 elemenata najefikasnije se uređuje Quick sort algoritmom.

Veličina / Algoritam	1000	100000	10000000
Bubble sort	2	19126	59845664
Quick sort	0	10	1083
Selection sort	2	19114	54175920
Insertion sort	0	4508	34295981.26
Radix sort	0	17	5682
Merge sort	1	85	9588
Shell sort	0	18	3173.666667
Heap sort	0	38.666668	4491.333496

Tabela 4: *Vreme uređivanja nasumičnog niza (u ms)*

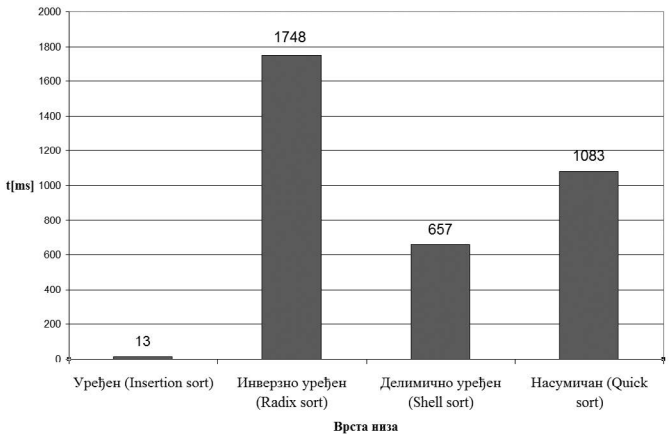
Generalno, razlike u vremenima uređivanja nizova manje veličine, bilo kog nivoa uređenosti, su zanemarljivo male, pa ne postoji jedan, već više optimalnih algoritama uređivanja. Povećanjem demenzije niza razlika u izmerenim vremenima uređivanja postaje izraženija. Vidno je da se pojedini algoritmi

ističu, pokazujući značajno bolje performanse od drugih, koje je u nekim situacijama besmisleno koristiti.

Praćenjem ponašanja algoritama u zavisnosti od veličine, kao i nivoa sortiranosti niza, dolazi se do zaključka koje algoritme je najbolje koristiti u određenoj situaciji, što je predstavljeno u tabeli 5.

Niz \ Veličina	1000	100000	10000000
Uređen	Quick, Insertion, Shell i Radix sort	Insertion sort	Insertion sort
Inverzno uređen	Quick i Radix sort	Radix sort	Radix sort
Delimično uređen	Quick, Insertion, Shell i Radix sort	Shell sort	Shell sort
Nasumičan	Quick, Insertion, Shell, Radix i Heap sort	Quick sort	Quick sort

Tabela 5: *Optimalni algoritmi uređivanja nizova*



Slika 3: *Vremena uređivanja niza dimenzije 10.000.000 optimalnim algoritmima*

Zanimljivo je primetiti ponašanje Insertion sort algoritma, koji pokazuje najbolje performanse prilikom uređivanja uređenog niza od 100000 (0.667ms) i 10000000 (13ms) elemenata, dok mu je za uređivanje inverzno uređenog niza dimenzije 10000000 potrebno nešto više od dvadeset sati.

Upoređivanje vremena uređivanja različitih vrsta nizova, optimalnim algorimom za datu vrstu niza, može se izvršiti na osnovu grafika sa slike 3. Na istom se vidi da je najviše vremena neophodno za uređivanje inverzno uređenog niza, zatim za uređivanje nasumičnog niza, potom delimično uređenog niza, i na kraju najefikasnije je „urediti" niz koji je već uređen. Da bi se optimalan algoritam upotrebio u pravom smislu, neophodno je doći do saznanja kakvog je tipa kolekcija podataka (niz) koji se želi urediti.

Pretraživanje

Pretraživanje nizova implementirano je kao pretraživanje uređenih ključeva, kroz tri algoritma pretraživanja: sekvencijalno, binarno i interpolaciono. U tabeli 6 prikazana su vremena pretraživanja svakim od algoritama za različite dimenzije niza.

Pretraga \ Veličina	1000	100000	10000000
Sekvencijalno	0	0.666667	31
Binarno	0	0	0
Interpolaciono	0	0	0

Tabela 6: *Vreme pretraživanja nizova (u ms)*

Pretraživanje se efikasno obavlja bilo kojim algoritmom. Pretpostavka je da su nizovi koji se pretražuju uređeni, pa bitnu ulogu ima pozicija na kojoj se traženi element nalazi; što je veći indeks traženog elementa, to je potrebno više vremena da se on i pronađe npr. sekvencijalnom pretragom. Eksperimentalno, pretraživani su udaljeni elementi niza, sa porastom dimenzije niza. Iako je moguće bilo kojim algoritmom pronaći traženi element niza u razumnom vremenu, sekvencijalno pretraživanje pokazuje slabije performanse od binarnog i interpolacionog pretraživanja.

7.2. LISTE

Uređivanje

U tabeli 7 su prikazana vremena uređivanja uređene liste svih dimenzija.

Algoritam \ Veličina	1000	100000	10000000
Bubble sort	0	2	110.666667
Quick sort	0	61	9983.666992
Selection sort	2.666667	68900.33594	11276688.68
Insertion sort	2.666667	67504	1667793.088
Radix sort	0	35	5430
Merge sort	0.666667	/	/

Tabela 7: *Vreme uređivanja uređene liste (u ms)*

Rekurzija Merge sort algoritma obuhvata kreiranje kopija podataka koje je potrebno urediti. To uzrokuje zauzimanje prevelikog memorijskog prostora, pa na testiranom računaru memorije 4GB nije bilo moguće izvršiti uređivanje lista od 100000 i 10000000 elemenata, bilo kog nivoa uređenosti (upisan znak / umesto rezultata). Uređene ulančane liste koje sadrže manje čvorova uspešno se mogu urediti sledećim algoritmima: Bubble, Quick i Radix sort, dok se Bubble sort pokazao kao najbolji algoritam za uređivanje većih ovakvih listi.

Vremena uređivanja inverzno uređene liste se mogu videti u tabeli 8. Za inverzno uređene liste veće veličine očigledno je da Insertion sort algoritam ima najbolje performanse.

Algoritam \ Veličina	1000	100000	10000000
Bubble sort	5.666667	93926	15150797.36
Quick sort	0	62.333333	10054.66699
Selection sort	2.333333	46423.66797	7488401.359
Insertion sort	0	3.333333	367
Radix sort	0	31	5302
Merge sort	0.666667	/	/

Tabela 8: *Vreme uređivanja inverzno uređene liste (u ms)*

Veličina / Algoritam	1000	100000	10000000
Bubble sort	2	36357	6139658.254
Quick sort	0.333333	93.333336	15761.33308
Selection sort	1	50692.66797	8560540.675
Insertion sort	2	34327.33203	5796903.85
Radix sort	0	38	5398
Merge sort	0.333333	/	/

Tabela 9: *Vreme uređivanja delimično uređene liste (u ms)*

Iz vremena uređivanja delimično uređene liste, prikazanih u tabeli 9, može se videti da Radix sort pokazuje najbolje rezultate za bilo koju veličinu ovakve liste. Ipak za liste ovog tipa koje sadrže manji broj elemenata se može zaključiti da se mogu urediti u razumnom vremenu i ostalim algoritmima, dok se to ne može reći za slučaj većeg broja elemenata liste.

Nasumična lista malih dimenzija se može urediti korišćenjem više algoritama (Bubble sort, Quick sort, Radix sort). Ako je ipak nasumična lista koja se želi urediti nešto većih dimenzija (100000, pa do 10000000 elemenata) onda se treba odlučiti za uređivanje Radix sort algoritmom.

Veličina / Algoritam	1000	100000	10000000
Bubble sort	0	83016	28430754.74
Quick sort	0	105.666664	36188
Selection sort	1	47391	16230150.86
Insertion sort	1	118525.6641	40591870.53
Radix sort	0	84	11915
Merge sort	0.333333	/	/

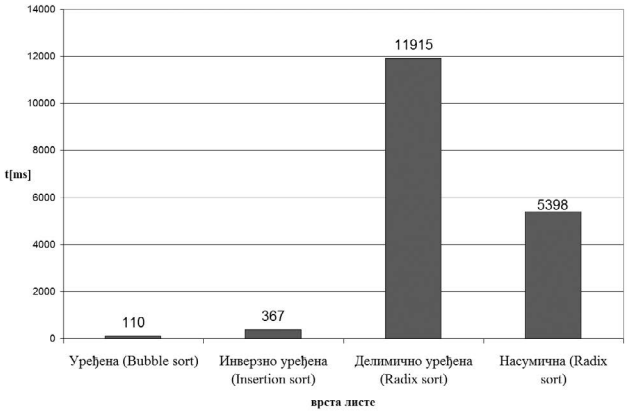
Tabela 10: *Vreme uređivanja nasumične liste (u ms)*

Iz prethodne četiri tabele, analizom vremena izvršenja algoritama može se zaključiti koji je algoritam u kojoj situaciji optimalan, što je prikazano u tabeli 11.

Veličina / Lista	1000	100000	10000000
Uređena	Bubble, Quick i Radix sort	Bubble sort	Bubble sort
Inverzno uređena	Quick i Radix sort	Insertion sort	Insertion sort
Delimično uređena	Radix sort	Radix sort	Radix sort
Nasumična	Quick, Radix i Bubble sort	Radix sort	Radix sort

Tabela 11: *Optimalni algoritmi uređivanja lista*

Treba uočiti karakteristike Bubble i Insertion sort algoritama. Naime, prvi od navedenih ima najbolje performanse od svih algoritama za slučaj uređivanja uređene liste, dok korišćenje istog nema puno smisla za liste ostalih nivoa sortiranosti. Slično je sa Insertion sort algoritmom koji je najefikasniji prilikom uređivanja inverzno uređenih lista. Za razliku od ova dva načina sortiranja, Radix sort pokazuje najbolje osobine za slučaj delimično uređenih i nasumičnih lista, ali vremena uređivanja ovog algoritma za ostale dve vrste kolekcija podataka ne odstupaju mnogo od vremena koja ga predstavljaju kao najefikasniji algoritam.



Slika 4: *Vremena uređivanja liste dimenzije 10.000.000 optimalnim algoritmima*

Analizirajući sliku 4, uočava se da je najgori slučaj uređivanja liste kada je ona delimično uređena. Gradacijski, prati je nasumična lista, zatim inverzno uređena i konačno najbolje karakteristike pokazuje uređena lista, za čije „uređivanje“ je potrebno izdvojiti najmanje vremena.

Upoređivanjem uređivanja nizova i lista istim algoritmima, primećuje se da se oni ne ponašaju isto za obe ove vrste kolekcije podataka. Kao primer se može uzeti Insertion sort algoritam, koji se pokazao kao optimalan prilikom uređivanja uređenih nizova, a gotovo najlošiji za uređivanje inverzno uređenih. Isti algoritam pokazuje totalno suprotno ponašanje prilikom uređivanja lista. Tada najbolje rezultate pokazuje upravo prilikom uređivanja inverzno uređenih lista.

Pretraživanje

U tabeli 12 prikazana su vremena pretraživanja svakim od pomenutih i opisanih algoritama pretraživanja, za različite dimenzije lista.

Veličina / Pretraga	1000	100000	10000000
Sekvencijalna	0	0	0
Binarna	0	1	99.666664
Interpolaciona	0	1	98.666664

Tabela 12: *Vreme pretraživanja liste (u ms)*

Primećuje se da će se do traženog podatka najbrže doći sekvencijalnim pretraživanjem, za razliku od pretraživanja nizova kod kojih ovaj algoritam pokazuje najslabije performanse.

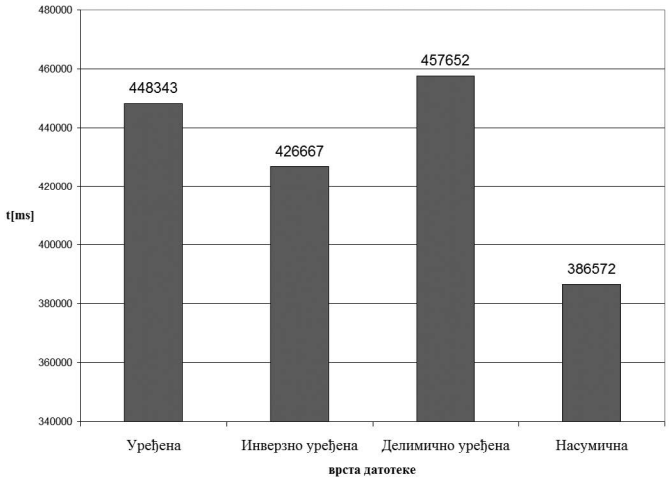
7.3. DATOTEKE

Uređivanje

Za uređivanje datoteka ima smisla napraviti analizu vremena uređenosti na osnovu prirode datoteke, obzirom da je implementiran jedan algoritam uređivanja External merge sort.

Veličina / Datoteka	1000	100000	10000000
Uređena	119	3314	448343
Inverzno uređena	121.664	2927.333	426667.344
Delimično uređena	120	3340	457652
Nasumična	120.664	3163.667	386572

Tabela 13: *Vreme uređivanja datoteka External merge algoritmom (u ms)*



Slika 5: Vremena uređivanja datoteka External merge algoritmom

Sa slike 5 se može videti da nema preterano velikih odstupanja u uređivanjima datoteka različitog nivoa sortiranosti. Isto se ne može reći ako se posmatra druga bitna karakteristika – veličina. Porastom broja elemenata datoteke znatno raste i vreme potrebno da se datoteka uredi.

Pretraživanje

Vremena sekvencijalnog pretraživanja datoteka nalaze se u tabeli 14, u kojoj se vidi da vreme pretraživanja datoteke značajno raste sa porastom broja elemenata datoteke.

Veličina / Pretraga	1000	100000	10000000
Sekvencijalna	3.333	158.333	18202.665

Tabela 14: *Vreme sekvencijalnog pretraživanja datoteke (u ms)*

8. PREGLED OBLASTI

Saznanja o optimalnim algoritmima, predstavljena u prethodnim tabelama, mogu se iskoristiti za optimizaciju uredjivanja i pretraživanja postojećih metoda u različitim programskim jezicima.

Programski jezik C# sadrži metodu Sort() za uredjivanje, koja implementira tri algoritma: Insertion Sort, Heap Sort i Quick Sort, u zavisnosti od broja elemenata kolekcije podataka. Kao što je prethodno u radu prikazano, veličina kolekcije nije jedini parametar koji se može uzeti u razmatranje, pa je moguće izvršiti optimizaciju ove metode, ako bi se u proces odabira algoritama za implementaciju dodao i podatak o nivou uredjenosti kolekcije. Slično, u java programskom jeziku implementiran je Merge Sort algoritam za uredjivanje lista, a Quick Sort za uredjivanje nizova. Već je dokazano da ovi algoritmi nisu uvek optimalni, tako da se može izvršiti poboljšanje performansi korišćenjem nekih drugih algoritama u konkretnim slučajevima. Očigledni su razlozi na osnovu kojih se pretpostavlja da će u narednim, naprednijim tehnologijama doći do izvesnih promena postojećih metode uredjivanja i pretraživanja, kako bi se iste poboljšale.

9. ZAKLJUČAK

Uređivanje i pretraživanje su veoma česte i neophodne radnje u radu sa podacima. Uobičajeno, oba ova procesa, ugrađena u biblioteke su dovoljno dobra za većinu primena. Da bi se bilo kakva radnja uređivanja izvršila neophodno je porediti i menjati mesta elementima kolekcije podataka, a sve to zauzima procesorsko vreme i memoriju.

U ovom radu pokazano je kako različiti tipovi algoritama mogu da naprave veliku razliku u performansama. Vreme potrebno za uređivanje ili pretraživanje značajno se menja kako broj elemenata neke kolekcije raste ili se menja nivo uređenosti kolekcije. Varijacije vremena izvršavanja algoritama ukazuju na značaj izbora pravog algoritma za manipulisanje podacima. Analizom vremena utvrđenih optimalnih algoritama uređivanja zaključuje se da postoje značajne razlike u njihovom korišćenju nad kolekcijama podataka različitih prema mestu na kome se podaci nalaze, pa je najefikasnije koristiti nizove, zatim liste i na kraju datoteke.

Veoma bitno je odrediti karakteristike kolekcije, i upotrebiti optimalan algoritam nad njom, kako bi se uštedelo vreme čekanja da se akcije nad njom izvrše.

LITERATURA

[1] Sedqewick, R., Wayne, K. (2009). Algorithms fourth edition, Massachusetts: Addison Wesley Publishing Company
[2] Niemann, T. (2008). Sorting and Searching Algorithms, Portland: epaperpress.com
[3] Sedgewick, R. (1990). Algorithms in C, Massachusetts: Addison Wesley Publishing Company
[4] Tomašević, M. (2004). Algoritmi i strukture podataka, Beograd: Akademska misao
[5] Weiss, M. A. (1992). Data Structures and Algorithm Analysis in C, Benjamin-Cummings Publishing Company
[6] Pavlović - Lažetić, G. (2014). Programiranje 2, Beograd: Matematički fakultet Univerziteta u Beogradu, (dostupno na: http://poincare.matf.bg.ac.rs/~gordana/programiranjeII/Finale.pdf)
[7] Rakić, G. (2005). Dinamičko alociranje memorije u programskom jeziku C, Beograd: Matematički fakultel Univerziteta u Beogradu
[8] Cormen, T., Leiserson, C., Rivest, R., Stein, C. (2009). Introduction to Algorithms third edition, Massachussetts: Massachussetts Institute of Technology
[9] Vitter, J. S. (2008). Algorithms and Data Structures for External Memory, now Publishers Inc
[10] Bourke, C. (2015) Searching & Sorting, Lincoln: Department of Computer Science & Engineering University of Nebraska

Selena Matijević, master inženjer organizacionih nauka, Vojska Srbije
Kontakt: selenamatijevic2@gmail.com
Oblasti interesovanja: algoritmi, programiranje, razvoj softvera

Dr Saša D. Lazarević, Fakultet organizacionih nauka, Univerzitet u Beogradu
Kontakt: sasa.lazarevic@fon.rs
Oblasti interesovanja: softversko inženjerstvo, informacioni sistemi, baze podataka, sistemi za upravljanje dokumentacijom, .NET platforma