Usporedba algoritma za sortiranje i pretraživanje

Ivančić, Andrijana

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

ANDRIJANA IVANČIĆ

USPOREDBA ALGORITAMA ZA SORTIRANJE I PRETRAŽIVANJE

Završni rad

Pula, veljača, 2018. godine

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

ANDRIJANA IVANČIĆ

USPOREDBA ALGORITAMA ZA SORTIRANJE I PRETRAŽIVANJE

Završni rad

JMBAG: 0081141352, izvanredan student
Studijski smjer: Informatika

Predmet: Strukture podataka i algoritmi
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: Doc. dr. sc. Tihomir Orehovački

Pula, veljača, 2018. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisana Andrijana Ivančić, kandidat za prvostupnika Informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, _____, _____. godine.

IZJAVA
o korištenju autorskog djela

Ja, Andrijana Ivančić dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom Usporedba algoritama za sortiranje i pretraživanje koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.
Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, _____.

Potpis

_____

SADRŽAJ

1. UVOD

Kada se nađemo pred nekim problemom, u privatnom ili poslovnom svijetu, tražimo način kako ga riješiti, a često težimo da to bude u što kraćem vremenskom roku. Kako bi pristupili samom rješavanju problema potrebno je uočiti ili definirati problem te ga što detaljnije rastaviti na manje cjeline. Sljedeći korak je traženje najboljeg rješenja našeg problema. Algoritmi rade na sličan princip. „Algoritam mora biti sastavljen od konačnog broja koraka koji ukazuju na slijed operacija koje treba obaviti nad početnim objektima kako bi se dobili završni objekti ili rezultati. Svaki korak se opisuje instrukcijom. Obavljanje algoritma naziva se algoritamskim procesom“ (Čičak, n.d., p. 3). Kako bi se pojedina instrukcija algoritma shvatila mora biti napisana precizno, jasno i u odgovarajućem jeziku. Algoritmi se mogu prikazati različito te tako oni mogu biti opisni, prikazani rečenicama, grafički ili pseudokodom.

U ovom radu algoritmi su prikazani u programskom jeziku C++. „Algoritam za sortiranje je bilo koji algoritam koji rješava problem sortiranja. Rješenje problema uzlaznog sortiranja dobiva se permutacijom te se od liste n s početnim parametrima $(a_1, a_2, a_3...an)$ dobiva lista za koju vrijedi $a_1 \leq a_2 \leq a_3 \leq ... \leq an$“ (Manger, 2014, p. 141). Sortirati možemo uzlazno, od manjeg prema većem, te silazno od većeg prema manjem. Nemoguće je predvidjeti točno ponašanje pojedinog algoritma. Nakon sortiranja važno nam je i pretraživanje, odnosno način na koji ćemo doći do određene datoteke, podatka ili stvari koje tražimo. Osnovne strategije pretraživanja su slijepo i usmjereno pretraživanje. Neki od najjednostavnijih algoritama za pretraživanje su linearno i binarno pretraživanje. „Binarno pretraživanje je tehnika pronalaska lokacije određenog elementa u sortiranoj listi“ (Harris & Ross, 2006, p. 199). Svaki algoritam ima i svoju vremensku složenost: najbolju, najgoru i prosječnu. Algoritmi sortiranja i pretraživanja objašnjeni su u nastavku.

Cilj ovoga rada je obuhvatiti i objasniti algoritme pretraživanja i sortiranja te dati uvid u usporedbu ovih algoritama.

Ovaj rad obuhvaća proučavanje algoritama, način na koji dijelimo algoritme te kako ih međusobno procjenjujemo. Algoritmi obuhvaćeni u ovom radu su algoritmi za sortiranje i pretraživanje.

Poglavlje 2 pruža uvid u značenje pojma algoritma te u kratku povijest njegova nastanka i utvrđuje koji je algoritam zapravo dobar algoritam.

Poglavlje 3 objašnjava vremensku složenost algoritama koja se može iskazati u najgorem, prosječnom i najboljem slučaju.

Podjela i objašnjavanje algoritama za sortiranje je prikazano u poglavlju 4. Podjela obuhvaća sljedeće algoritme: algoritme sortiranja zamjenom elemenata, sortiranje umetanjem, rekurzivne algoritme za sortiranje i algoritam sortiranja s pomoću binarnih stabla među kojima je najpoznatiji algoritam sortiranja s pomoću hrpe (engl. heap sort). Algoritam sortiranja zamjenom elemenata obuhvaća sortiranje izborom najmanjeg elementa (engl. selection sort) i sortiranje zamjenom susjednih elemenata (engl. bubble sort). Sortiranje umetanjem se dijeli na jednostavno sortiranje umetanjem (engl. insertion sort) i višestruko sortiranje umetanjem (engl. shell sort). Rekurzivni algoritmi obuhvaćaju algoritam brzog sortiranja (engl. quick sort) i algoritam sortiranja spajanjem (engl. merge sort).

U poglavlju 5 prikazana je usporedba algoritama za sortiranje.

Poglavlje 6 bavi se algoritmima za pretraživanje koji se dijele na slijepe i usmjerene strategije za pretraživanje. Strategija slijepog pretraživanja obuhvaća sljedeće algoritme: algoritam linearnog pretraživanja (engl. linear search), algoritam binarnog pretraživanja (engl. binary search), pretraživanje najprije u širinu (engl. breadth-first search), pretraživanje najprije u dubinu (engl. depth-first search) i pretraživanje s jednolikom cijenom (engl. uninformed-cost search). Strategija usmjerenog pretraživanja obuhvaća: algoritam pretraživanja najboljeg prvog (engl. best-first search) i A* (čita se A star) algoritam.

Ovaj rad završava prikazivanjem vremenske složenosti algoritama za pretraživanje i njihovom međusobnom usporedbom.

2. ALGORITMI I STRUKTURE PODATAKA

Kako bi razumjeli rad pojedinih algoritama u nastavku će biti objašnjeni osnovni pojmovi kao što su algoritam, struktura podataka i dijelovi strukture podataka.

„Riječ „algoritam“ potječe od perzijskog matematičara, astronoma i geografa iz devetog stoljeća al-Khwarizmija (puno ime u engleskoj transkripciji glasi Muhammad ibn Musa al-Khwarizmi). Napisao je knjigu u kojoj je opisao postupke za računanje u indijskom brojevnom sustavu. Original na arapskom jeziku nije sačuvan, a latinski prijevod proširio se Europom pod naslovom „Algoritmi de numero Indorum“ („Al-Khwarizmi o indijskim brojevima“). Prema tom naslovu postupke za sustavno rješavanje problema danas nazivamo algoritmima“ (Krčadinac, 2016/2017, p. 4).

Jedna od definicija koja opisuje što je to algoritam (Manger, 2014, p. 2):

- „algoritam - konačan niz naredbi od kojih svaka ima jasno značenje i izvršava se u konačnom vremenu. Izvršavanjem tih naredbi zadani ulazni podatci pretvaraju se u izlazne podatke (rezultate)“.

To je postupak koji se može izvršiti na Turingovom stroju. Algoritam ima sljedeće karakteristike (Basu, 2013):

- ima konačni slijed uputa,
- zahtijeva konačan unos,
- proizvodi konačan izlaz,
- prestaje nakon konačnog vremena.

Svaki algoritam ima i sljedeća svojstva:

- konačnost,
- definiranost,
- ulaz,
- izlaz,
- efikasnost.

„Algoritmi predstavljaju dinamički aspekt (ono što se radi) dok strukture podataka predstavljaju statički aspekt (to je ono s čime se radi)“ (Manger, 2014, p. 1).

„Struktura podataka - skupina varijabli u nekom programu zajedno s vezama između tih varijabli. Stvorena je s namjerom da omogući pohranjivanje određenih podataka te efikasno izvršavanje određenih operacija s tim podatcima (upisivanje, promjena, čitanje, traženje po nekom kriteriju...)“ (Manger, 2014, p. 2).

Sljedeće strukture podataka i operacije na njima su osnovni elementi brojnih algoritama. U ovom radu obuhvaćene su sljedeće strukture podataka (Pavlica, 2014):

- „polje (engl. array) - sekvencijalni niz podataka istog tipa koje imaju zajedničko ime,
- lista, povezana lista (eng. linked list) - niz elemenata koji sadrže podatke i pokazivače na sljedeći element,
- stog (engl. stack) - niz elementa u kojem je dodavanje i brisanje moguće obavljati samo na jednom kraju niza,
- red (engl. queue) - niz elemenata u koje je dodavanje moguće samo na jednom kraju a brisanje samo na drugom kraju,
- stablo, binarno stablo (engl. binary tree) – hijerarhijska struktura u kojoj svaki element može imati samo jednog prethodnika,
- graf (engl. graph) – općenita struktura u kojoj svaki element može biti povezan s više drugih elemenata“.

3. VREMENSKA SLOŽENOST ALGORITAMA

„Ne ovisi o veličini inače bi krava ulovila zeca“ (Manber, 1989., p. 37).

Na rad pojedinog algoritma utječu različiti parametri. Što se tiče složenosti algoritma, razlikujemo vremensku i prostornu složenost. Vremenska složenost je vrijeme koje je potrebno da se algoritam izvrši. Kako bi saznali točno vrijeme izvršavanja algoritma potrebno je pobrojati sve instrukcije koje su bile aktivne tijekom samog procesa i pomnožiti s vremenom koje je potrebno da računalo procesuira tu instrukciju. Stoga razlikujemo sljedeće slučajeve (Baumgartner, 2013/2014):

- najgori slučaj trajanja algoritma –vrijeme izvršavanja algoritma je jako sporo
- prosječni slučaj trajanja algoritma – vrijeme izvršavanja algoritma je prosječno
- najbolji slučaj trajanja algoritma – vrijeme izvršavanja algoritma je brzo

S vremenom izvršavanja pojedinog algoritma možemo poistovjetiti broj osnovnih operacija koje odgovarajući program treba obaviti, usporedbe, pridruživanja vrijednosti, aritmetičke operacije i dr.. „Vrijeme izražavamo kao funkciju T(n) gdje je n neka pogodna mjera za veličinu skupa ulaznih podataka. Ako promatramo algoritam koji sortira niz brojeva tada njegovo vrijeme izražavamo kao T(n) gdje je n duljina toga niza brojeva“ (Manger, 2014, p. 13).

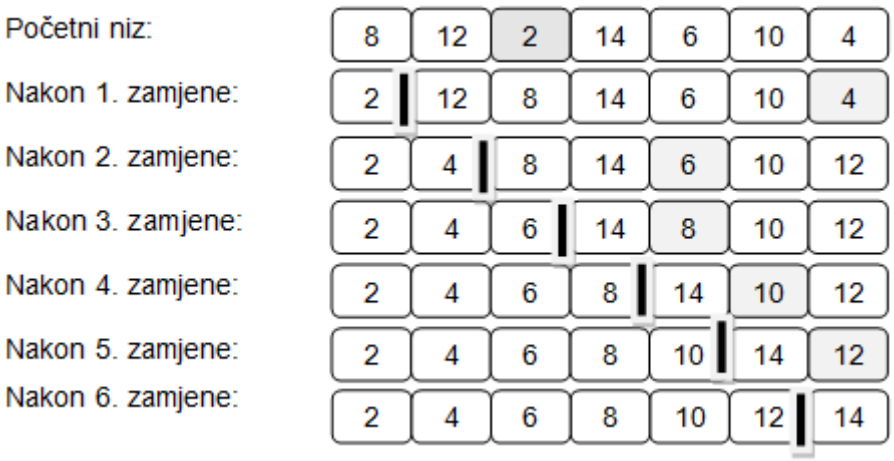
4. ALGORITMI ZA SORTIRANJE

Za pojedini problem postoji više algoritama pomoću kojih se određeni problem može riješiti. Najjednostavniji i najsporiji algoritmi za sortiranje su sortiranje izborom najmanjeg elementa i sortiranje zamjenom susjednih elemenata, a među najbrže se ubraja brzo sortiranje. Osnovna podjela algoritama za sortiranje je (Manger, 2014):

- sortiranje zamjenom elemenata – obuhvaća sortiranje izborom najmanjeg elementa (engl. selection sort) i sortiranje zamjenom susjednih elemenata (engl. bubble sort).
- sortiranje umetanjem – postoji jednostavno sortiranje umetanjem (engl. insertion sort) i višestruko sortiranjem umetanjem (engl. shell sort).
- rekurzivni algoritmi za sortiranje – najbrži rekurzivni algoritam je algoritam brzog sortiranja (engl. quick sort) i algoritam sortiranja s pomoću hrpe (engl. heap sort).
- sortiranje s pomoću binarnih stabla – algoritam sortiranja s pomoću hrpe (engl. heap sort).

4.1 Sortiranje izborom najmanjeg elementa

Jedan od najjednostavnijih algoritama za sortiranje započinje pretraživanjem svih elemenata u nizu dok se ne pronađe najmanji element. Sortiranje izborom najmanjeg elementa uzima najmanji broj i zatim ga pozicionira na mjesto početnog indeksa ako se utvrdi da ima manju vrijednost od početnog indeksa. Zatim se pronalazi sljedeći najmanji broj i pozicionira ga se na mjesto sljedećeg indeksa. Zamjena se nastavlja dalje bez početnog elementa u polju, tj. bez početnog indeksa, sve dok cijeli niz ne bude sortiran. Iako algoritam prolazi kroz sve elemente, dovoljno je napraviti jednu zamjenu kako bi broj bio na svojoj odgovarajućoj poziciji indeksa. Ovaj postupak se ponavlja kako bi se sortirali svi brojevi.



Slika 1. Prikaz rada algoritma sortiranja izborom najmanjeg elementa

Izvor: modificirano prema (Manger, 2014, p. 142)

```
void SelectionSort (int skup[], int n) {
    for (int i=0; i<n-1; i++) {
        int min=i;
        for (int j=i+1; j<n; j++) {
            if (skup[j] < skup[min]) {
                min=j;
            }
        }
        zamjena (skup[min], skup[i]);
    }
}
```

Slika 2. Implementacija algoritma sortiranja izborom najmanjeg elementa u C++

Izvor: modificirano prema (Manger, 2014, p. 143)

U ovoj implementaciji koristimo funkciju izbora najmanjeg elementa koja prima polje i broj njegove duljine. Pomoću naredbe zamjene mijenjamo poredak elemenata prema njihovoj vrijednosti, manjoj ili većoj, te ih stavljamo na mjesto odgovarajućeg indeksa.

Vremenska složenost algoritma sortiranja izborom najmanjeg elementa iznosi $O(n^2)$. U prvom prolazu broj usporedbi je *n-1*. U drugom prolazu funkcija prođe *n-2* usporedbe te sa svakom sljedećom iteracijom broj usporedbi se smanjuje za jedan.

Izračun iznosi:

$$(n-1)+(n-2)+(n-3)+.....+2+1=n(n-1)/2$$

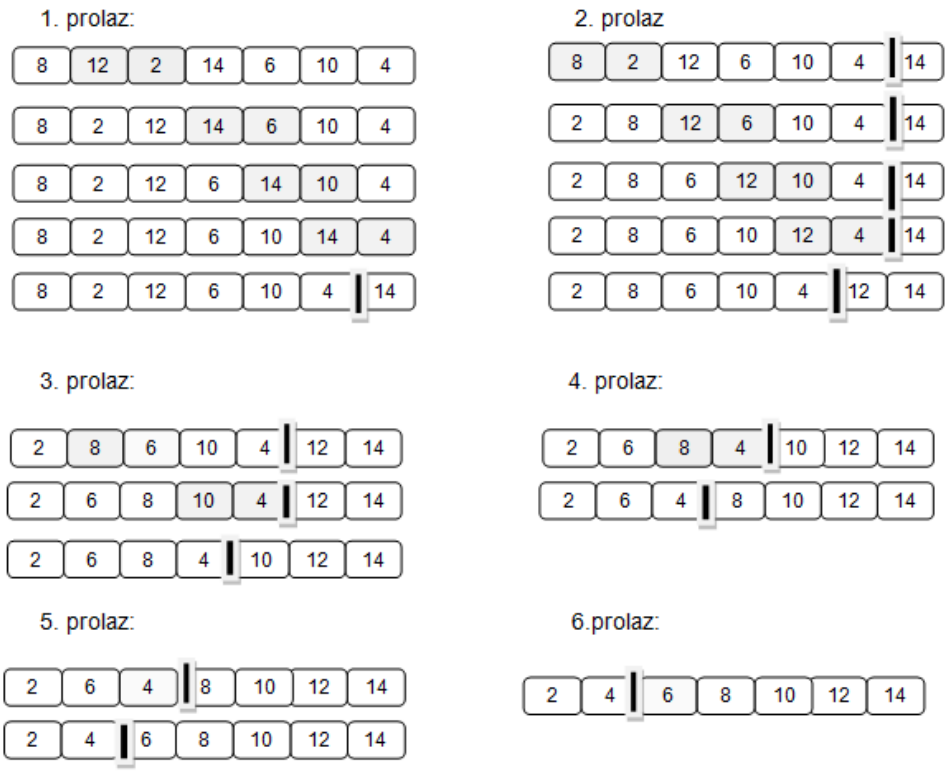
Ukupan broj usporedbi je $n(n-1)/2$. Kroz svaki prolazak vrši se zamjena elemenata. Ukupan broj operacija iznosi $3(n-1)$ tj. iznosi: $n(n-1)/2+3(n-1)=O(n^2)$ (Manger, 2014).

Najgori slučaj vremenske složenosti: $O(n^2)$.
Najbolji slučaj vremenske složenosti: $O(n^2)$.
Prosječna vremenska složenost: $O(n^2)$.

Najbolji slučaj vremenske složenosti je kada je niz već sortiran uzlazno, a najgori kada je niz sortiran silazno. Vremenska složenost ostaje ista ($O(n^2)$) za svaki slučaj jer funkcija algoritma uvijek prolazi kroz sve elemente niza onoliko puta koliko ima elemenata u nizu.

4.2 Sortiranje zamjenom susjednih elemenata

Cilj algoritma za sortiranje zamjenom susjednih elemenata je dovesti najmanji element na početnu poziciju tj. na početni indeks niza. Sortiranje zamjenom susjednih elemenata ili mjehurićasto sortiranje (engl. bubble sort) je jednostavan algoritam sortiranja koji počevši od početnog indeksa uspoređuje dva susjedna elementa i radi zamjenu vrijednosti elemenata na pojedinim indeksima polja ako je potrebno. Ponavljamo postupak za prve $n-1$ članove niza, bez posljednjeg elementa niza, sve dok cijeli niz ne bude sortiran. Posljednji element niza ne ulazi u postupak jer je sortiran na svoje pravo mjesto. Tijekom sortiranja niz se dijeli na sortirani i nesortirani niz.



Slika 3. Prikaz rada algoritma sortiranja zamjenom susjednih elemenata

Izvor: modificirano prema (Manger, 2014, p. 145)

```
void BubbleSort (int skup[], int n) {
    int i, j;
    for (i=0; i < n-1; i++) {
        for (j=0; j < n-1-i; j++) {
            if (skup[j+1] < skup[j])
                swap (skup[j], skup [j+1]);
        }
    }
}
```

Slika 4. Implementacija algoritma sortiranja zamjenom susjednih elemenata u C++

Izvor: modificirano prema (Manger, 2014, p. 145)

Algoritam sortiranja zamjenom susjednih elemenata koristi se za sortiranje manjih skupova podataka. Najbolji slučaj vremenske složenosti je O(n) tj. kada je niz već sortiran te ima samo jednu iteraciju kroz petlju sa *n-1* usporedbom i 0 zamjena. Položaj elemenata u nizu imaju značajnu ulogu u određivanju vremenske složenosti.

Ako se najmanji elementi nalaze na kraju niza, jako sporo se pomiču na početak. Najgori slučaj vremenske složenosti je kada je niz naopako sortiran.

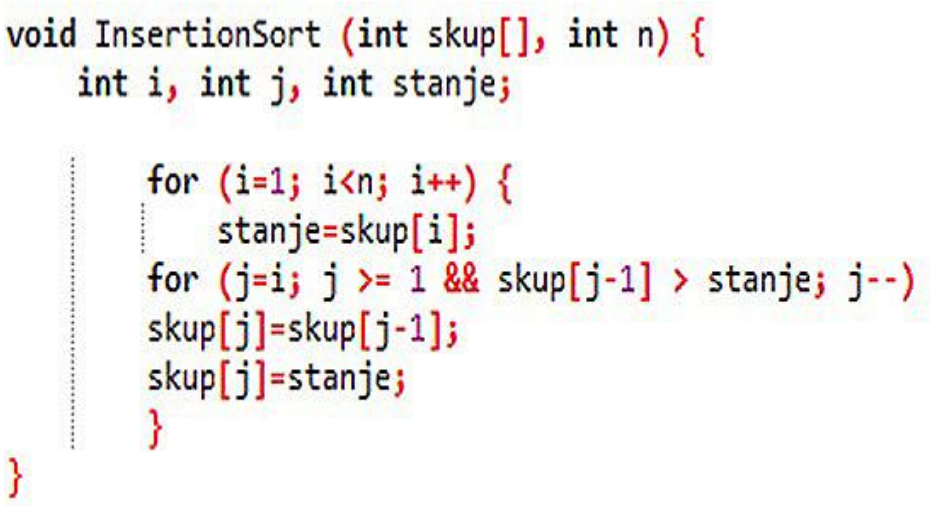
„U prvom prolazu imamo u najgorem slučaju *n-1* usporedbi i *n-1* zamjena elemenata. U drugom prolazu imamo u najgorem slučaju *n-2* usporedbi i *n-2* zamjena elemenata. Stoga najgori slučaj vremenske složenosti iznosi O(n²)“ (Manger, 2014, p. 146). Prosječni slučaj vremenske složenosti također iznosi O(n²).

4.3 Jednostavno sortiranje umetanjem

Tijekom jednostavnog sortiranja umetanjem imamo dvije skupine: sortiranu i nesortiranu skupinu. Ovdje se sortiraju vrijednosti na pojedinim indeksima polja. U početku sortirana skupina je prazna na poziciji početnog indeksa. Smatra se da je vrijednost početnog indeksa sortirana. U nesortiranoj skupini nalaze se svi elementi niza. Uzima se jedan po jedan element iz nesortirane skupine i postavlja se na odgovarajuću poziciju indeksa u sortiranoj grupi. Na kraju sortiranja svi elementi se nalaze sortirani u sortiranoj skupini (Harris & Ross, 2006). Ako se utvrdi da je promatrani element iz nesortirane skupine manji od prethodnih elemenata u sortiranoj skupini, elementi koji su veći pomiču se za jedno mjesto udesno.

Slika 5. Prikaz rada algoritma jednostavnog sortiranja umetanjem

Izvor: modificirano prema (Manger, 2014, p. 147)



Slika 6. Implementacija algoritma jednostavnog sortiranja umetanjem u C++

Izvor: modificirano prema (Manger, 2014, p. 147)

Jednostavno sortiranje umetanjem ima i nekoliko pozitivnih karakteristika (Oljica, 2014):

- jedna je od najjednostavnijih primjena,
- djelotvoran je za manje skupove podataka, ali neučinkovit za velike skupine podataka,
- prilagodljiv je tj. smanjuje broj ukupnih koraka ako je dodijeljen djelomično već sortiran niz, a to povećava njegovu učinkovitost,
- bolji je od algoritma sortiranja izborom i mjehurićastog sortiranja,
- kao i kod mjehurićastog sortiranja zahtijeva konstantan memorijski prostor O(1).

Ako je niz sortiran, vrijeme izvođenja je O(n). U tom slučaju imamo samo jednu usporedbu po iteraciji kroz petlju koja iznosi *n-1* usporedbi. Najgori slučaj je kada ima najveći broj usporedbi za svaku *n-1* iteraciju kroz petlju.

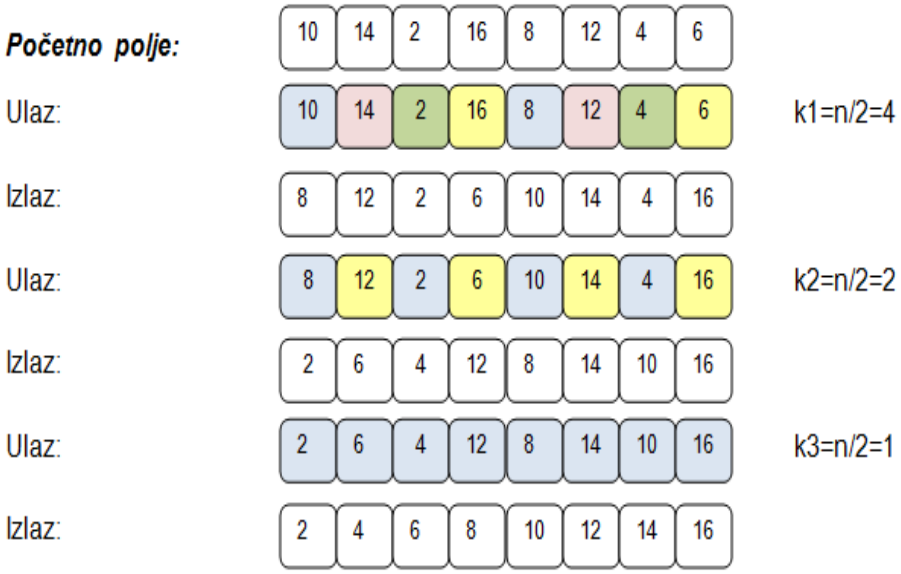
Prosječni slučaj jednak je i najgorem slučaju, a to je $O(n^2)$. „Za analizu prosječnog slučaja prvo pronalazimo prosječan broj usporedbi koje zahtijevaju umetanje *i*-tog elementa u niz. Prije samog umetanja *i*-tog elementa u niz postoji već *i* elemenata u tom nizu te nakon umetanja postoji *i+1* elemenata“ (Scheffler, 2003, p. 1).

4.4 Algoritam višestrukog sortiranja umetanjem

Algoritam višestrukog sortiranja umetanjem nazvan je po autoru Donaldu Shellu. Ideja ovog algoritma je sljedeća:

- organizirati slijed podataka u dvodimenzionalnom nizu,
- sortiranje stupaca niza.

Algoritam uspoređuje vrijednosti elemenata na određenim pozicijama indeksa koji su međusobno jednako udaljeni. Udaljenost elemenata u nizu, tj. *k,* izračunava se tako da se veličina niza podijeli s brojem 2. Pri svakom sljedećem koraku opet se dijeli s brojem 2. „Algoritam višestrukog sortiranja umetanjem koristi sekvencu padajućeg niza *k1* > *k2* > ... > *km=1* da obavi sortiranje *k1*-potpolja, zatim *k2*-potpolja, ... na kraju *km*-potpolja“ (Manger, 2014, p. 148). Posljednje potpolje sortira algoritam jednostavnog sortiranja umetanjem. Primjer sortiranja višestrukim umetanjem možemo vidjeti na sljedećoj slici (Slika 7.). Sortira se niz od 8 brojeva tj. *n* iznosi 8. Kada se veličina niza podijeli s brojem 2 dobijemo 4 te je to naš *k*. Zatim se, počevši od nultog indeksa, uspoređuju elementi sa svakim četvrtim udaljenim indeksom. Nakon što se izvrše određene potrebne zamjene elemenata po vrijednostima na određenim indeksima pomičemo se na sljedeću poziciju.
Sada se kreće od pozicije prvog indeksa i uspoređuju se elementi koji su udaljeni točno za 4 mjesta. Postupak se ponavlja sve dok ne dobijemo sortirani niz. Nakon završetka sortiranja sa *k1* nastavljamo dalje sa *k2* i tako dalje sve dok se ne dobije vrijednost *kn=1*. Pojedina potpolja prikazana su u bojama.



Slika 7. Prikaz rada algoritma višestrukog umetanja

Izvor: modificirano prema (Manger, 2014, p. 149)

```
void Shell sort (int skup [], int n)
{
  int move, i, j, temp;

  for (move=n/2; move>0; move/=2)
    {

  for (i=move; i<n; i+=1)
  {
      temp=skup[i];
  for (j=i; j>=move && skup[j-move]>temp; j-=move)
       skup[j]=skup[j-move];

      skup[j]=temp;
    }

   }
}
```

Slika 8. Implementacija algoritma višestrukog umetanja u C++

Izvor: modificirano prema (Manger, 2014, p. 149)

U ovoj implementaciji koristimo funkciju sortiranja višestrukog umetanja koja sadrži 3 for petlje pomoću kojih odabiremo naš *k* (potpolje) te stavlja određeni element na njegovu odgovarajuću poziciju indeksa.

Algoritam višestrukog sortiranja umetanjem predstavlja nepoznanicu za istraživače. Naime, za taj algoritam još uvijek ne postoje ocjene vremena izvršavanja. „Ako je padajući niz *k1 > k2 > ... > km = 1* oblika $2^{t}-1 > 2^{t-1}-1 > .... > 7 > 3 > 1$ tada se može dokazati da je vrijeme u najgorem slučaju O(n 3/2)“ (Manger, 2014, p. 150). Niz vrijednosti koje se koriste kod algoritma višestrukog sortiranja umetanjem naziva se niz inkrementa. Mogući su različiti nizovi inkrementa. Vrijeme izvršavanja algoritma ovisi o odabiru niza inkrementa (Sinapova, 2004):

- Shell-ov inkrement – {1,.....(N/2)/2, N/2} = > O(n²) –najgori slučaj odabira niza inkrementa.
- Hibbard-ov inkrement – {1,3,7,----2^k-1} = >
  O(N3/2) – u najgorem slučaju,
  O(N5/4) – u prosjeku, ali je to utvrđeno simulacijom, a ne dokazom.
- Sedgwick-ov inkrement – {1,5,19,41,109,....}, u kojem se vrijednosti generiraju po dvije formule: 9*4-9*2+1 i 4-3*2+1,
- O(N4/3) – u najgorem slučaju,
- O(N7/6) – u prosjeku.

4.5 Rekurzivni algoritmi sortiranja

Algoritam brzog sortiranja jednak je algoritmu sortiranja spajanjem po tome što oba algoritma dijele zadano polje na dva manja polja i sortiraju ta dva manja polja, tj. njihove vrijednosti na određenim pozicijama. Na kraju ih povezuju u veće polje. Rekurzivni algoritam je algoritam koji poziva sam sebe. U ovom radu će biti objašnjen algoritam brzog sortiranja.

4.5.1 Algoritam brzog sortiranja

„Metoda podijeli-pa-vladaj (engl. divide-and-conquer) sastoji se u tome da zadani primjerak problema razbijemo u nekoliko manjih (istovrsnih) primjeraka i to tako da se rješenje polaznog primjeraka može konstruirati iz rješenja manjih primjeraka“ (Manger, 2014, pp. 169-170). Ova metoda koristi se za rekurzivne algoritme među kojima je i algoritam brzog sortiranja. Osnovna ideja algoritma brzog sortiranja je:

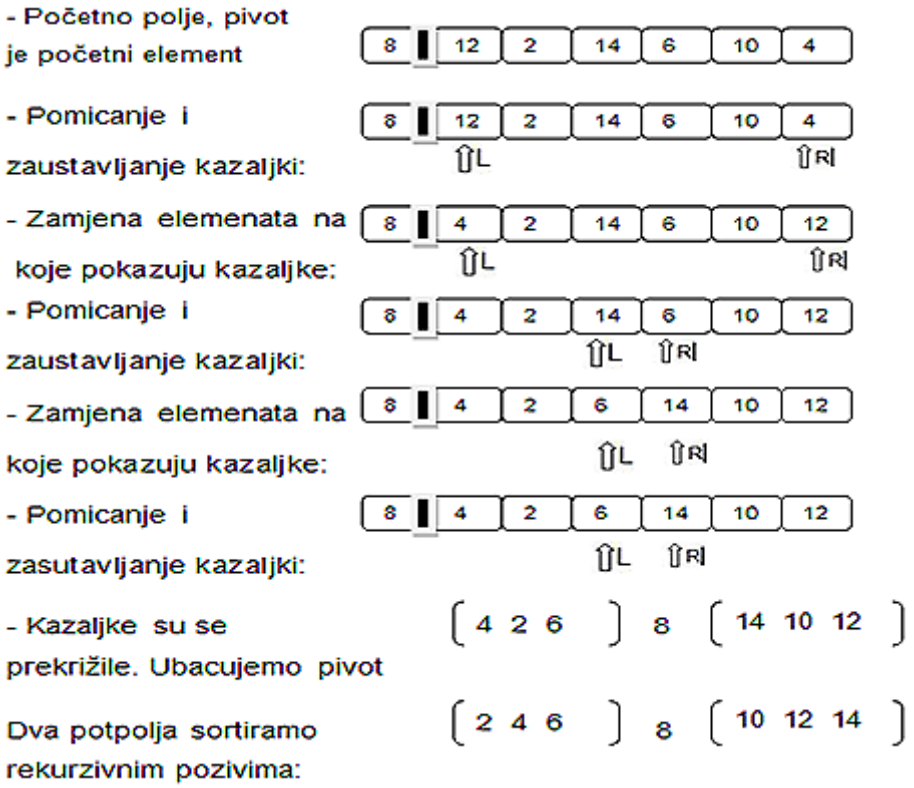
- odabere se jedan element kojeg nazivamo pivot,

- ulazi (elementi) koji su manji od pivota slažu se na lijevoj strani od pivota,
- ulazi (elementi) koji su veći od pivota slažu se na desnoj strani od pivota.

Koriste se dvije kazaljke, *i* i *j (ili L i R)* kako bi se usporedili elementi od lijeve strane prema desnoj te od desne prema lijevoj strani (Sinapova, 2004):

- „sve dok je *i* lijevo od *j* pomičemo *i* udesno preskačući sve elemente manje od pivota. Ako je za neki element utvrđeno da je veći od pivota *i* se zaustavlja.
- sve dok je *j* desno od *i* pomičemo *j* ulijevo preskačući sve elemente koji su veći od pivota. Ako je za neki element utvrđeno da je manji od pivota *j* se zaustavlja.
- kada su oboje *i* i *j* zaustavljeni elementi se zamjenjuju.
- kada se *i* i *j* prekriže nije potrebno mijenjati elemente. Zaustave se prolasci kroz niz i element na koji pokazuje *i* zamijeni se s pivotom te se zatim ponovno postavi“.

Algoritam brzog sortiranja je najbrži algoritam rekurzivnog sortiranja.



Slika 9. Prikaz rada algoritma brzog sortiranja

Izvor: modificirano prema (Manger, 2014, p. 144)

```
void quickSort (int arr [], int left, int right) {
    int i = left, j = right;
    int tmp;
    int pivot = arr [ (left + right) / 2];


    while (i <= j) {
        while (arr[i] < pivot)
        i++;
        while (arr[j] < pivot)
        j--;
  if (i<=j) {
    tmp = arr [i];
    arr[i] = arr[j];
    arr[j] = tmp;
    i++;
    j--;
    }
};
```

Slika 10. Implementacija algoritma brzog sortiranja u C++

Izvor: modificirano prema (Algolist, n.d.)

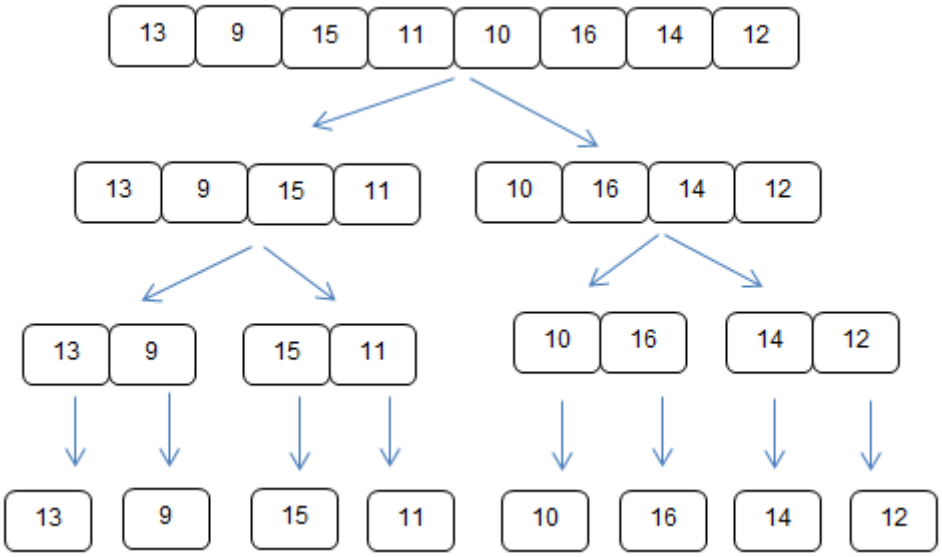
Najgori slučaj vremenske složenosti je kada je pivot prvi element u nizu. Vremenska složenost iznosi $O(n^2)$, no ovaj algoritam je bolji od algoritma mjehurićastog sortiranja. Ovaj slučaj ponavlja se samo kada svaki pivot pridonosi lošoj podjeli, ali ako je pivot odabran slučajno mala je vjerojatnost da će se to dogoditi.

Najbolji slučaj se javlja kada pivot razdvoji niz na dva jednaka dijela. U najboljem slučaju biti će *log n* podjela s kojom dobivamo više tzv. razina. Na jednoj razini imamo jedan niz sa *n* elemenata. Svaka razina sa svojom podjelom ima ukupno *n* prolaza kroz niz koji iznosi *n log n* prolaska. Prosječna vremenska složenost algoritma brzog sortiranja iznosi O(n log n) (Sinapova, 2004).

Jedni od popularnijih algoritama za sortiranje koje je potrebno spomenuti su: algoritam sortiranja spajanjem (engl. merge sort) i algoritam sortiranja s pomoću hrpe (engl. heap sort) koji su u nastavku ukratko objašnjeni.

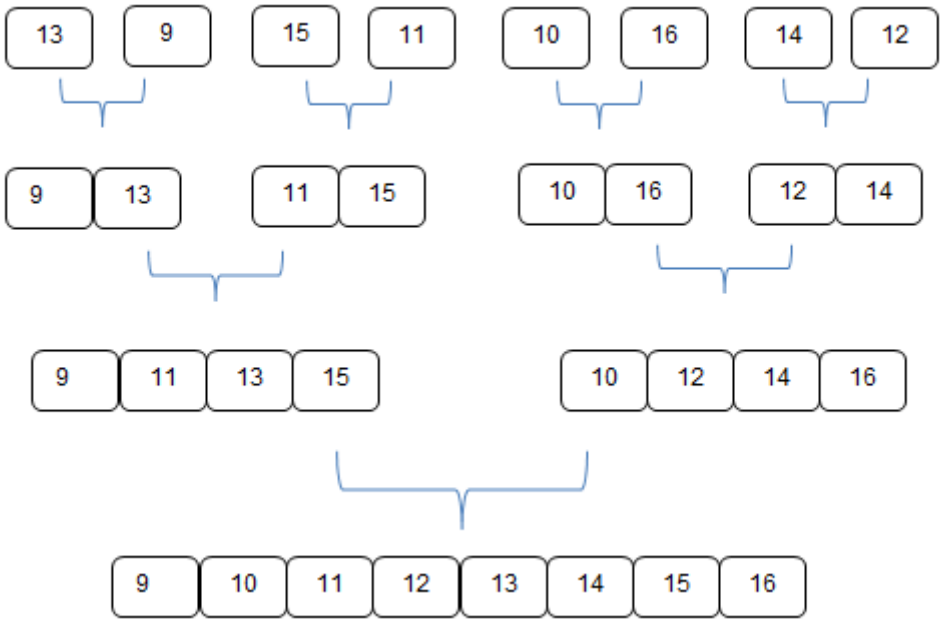
4.5.2 Algoritam sortiranja spajanjem

Algoritam sortiranja spajanjem kao i algoritam brzog sortiranja dijeli niz na manja potpolja. Bitna razlika između ova dva algoritma je u tome što algoritam sortiranja spajanjem dijeli niz na manja potpolja sve dok se ne dobije jedan element. Zatim ih sortira i vraća ih natrag, tj. spaja ih, sve do konačnog sortiranog niza. Vremenska složenost ovog algoritma iznosi O(nlogn).



Slika 11. Implementacija algoritma sortiranja spajanjem – dijeljenje niza na podliste

Izvor: modificrano prema (Joshi, 2017.)



Slika 12. Implementacija algoritma sortiranja spajanjem

Izvor: modificrano prema (Joshi, 2017.)

4.6 Sortiranje s pomoću binarnih stabla

Algoritam uzima elemente polja i ubacuje ih u binarno stablo, a zatim ih iz njega vadi i slaže u sortiranom redoslijedu (Manger, 2014).

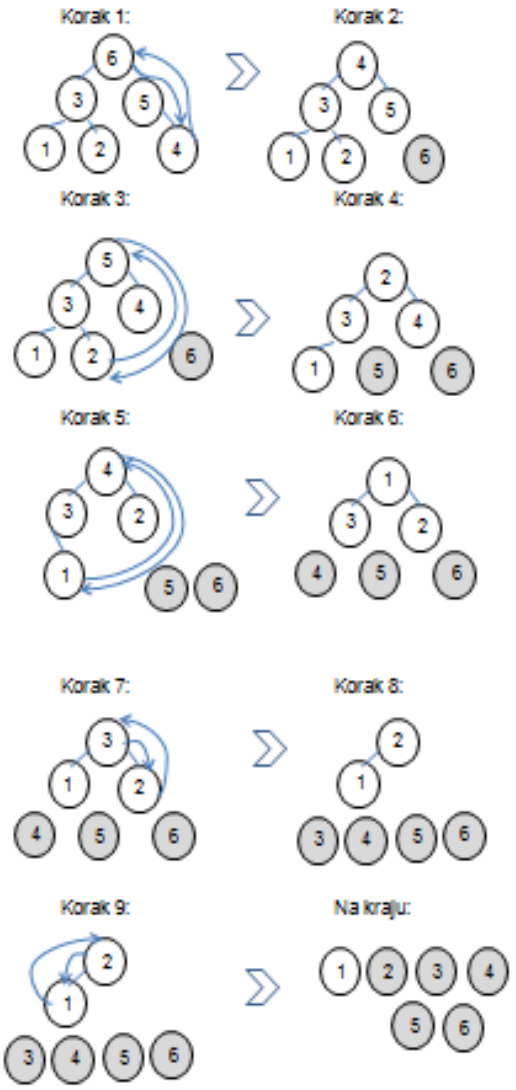
4.6.1 Sortiranje s pomoću hrpe

Svaki dan ima svoje brige, obaveze i poslove koje je potrebno odraditi. Pokušavamo svrstati naše obaveze prema nekom prioritetu. Sortiranje s pomoću hrpe (engl. heap sort) razvrstava elemente nekog niza po važnosti. Kod ovog algoritma služimo se kompletnim binarnim stablom i poljem kao strukturom podataka za elemente koje treba sortirati. Kompletno binarno stablo (engl. complete binary tree) predstavlja stablo u kojem su sve razine skroz popunjene, i to redom s lijeva na desno, osim možda posljednje razine te svi čvorovi su „gurnuti“ u lijevu stranu.

Kako bi napravili kompletno binarno stablo iz nesortiranog niza uzima se jedan po jedan element i pozicionira na odgovarajuću poziciju. Prvi element iz niza se postavlja kao korijen stabla. Sljedeća dva elementa se postavljaju kao roditelji, jedan element kao lijevi, a drugi element kao desni roditelj. Opet uzimamo dva elementa ali ih ovaj put postavljamo kao dijete lijevog roditelja. Za desnog roditelja dodamo dva elementa. Postupak se ponavlja dok se ne dođe do posljednjeg elementa. „Ako je indeks bilo kojeg elementa u polju *i*, element u indeksu *2i + 1* postat će lijevo dijete i element u indeksu *2i + 2* postat će desno dijete. Također, roditelj bilo kojeg elementa indeksa zadan je donjom granicom od *(i-1) / 2*“ (Joshi, 2017.).

Bitno je napomenuti da kod izgradnje kompletnog binarnog stabla moramo odrediti želimo li da naš niz bude sortiran u rastućem ili u padajućem nizu. Ako se odabere da niz bude u padajućem nizu onda slijedi da korijen ima manju vrijednost od svoje djece, a djeca korijena imaju još manju vrijednost od njihove djece i obrnuto.

Sami algoritam radi na principu da stavlja sve elemente niza redom na hrpu. Zatim se podaci brišu sa hrpe i ponovno dodaju u niz ali ovaj put u sortiranom redoslijedu. Vremenska složenost ovog algoritma je O(nlogn).



Slika 13. Implementacija algoritma sortiranja s pomoću hrpe

Izvor: modificirano prema (Gupta & Prakash, 2018.)

5. USPOREDBA ALGORITAMA ZA SORTIRANJE

Kako je već prethodno spomenuto, najjednostavniji te ujedno i najsporiji algoritmi za sortiranje su mjehurićasto sortiranje, sortiranje izborom najmanjeg elementa i sortiranje umetanjem elemenata. Najbrži algoritam je algoritam brzog sortiranja. Svaki od njih razlikuje se po svojoj vremenskoj složenosti i principu rada. Da bi se algoritmi usporedili, moramo promatrati tri slučaja vremenske složenosti: najbolji, najgori i prosječni slučaj. Međutim, broj usporedbi koje je potrebno izvršiti kod algoritma mjehurićastog sortiranja i algoritma izborom najmanjeg elementa ne ovisi o ulaznom stanju te međusobno uvijek imaju približno isti broj usporedbi. Za razliku od ova dva algoritma, algoritam sortiranja umetanjem uvelike ovisi o početnom stanju. Algoritam sortiranja umetanjem izvršava nekoliko naredbi manje u odnosu na broj elemenata u nizu te je sortiranje umetanjem brže od sortiranja izborom najmanjeg elementa. Algoritam mjehurićastog sortiranja više se koristi kod manjih skupina datoteka jer radi provjeru nekoliko puta dok ne sortira prvi element. Stoga kod mjehurićastog sortiranja prvo prolazimo kroz usporedbu vrijednosti prva dva susjedna elemenata, zatim sljedeća dva susjedna elemenata i tako dalje dok se ne sortira cijeli skup elemenata odnosno vrijednosti na pojedinim pozicijama indeksa. Ovaj algoritam radi brže ako je lista već sortirana ili barem djelomično sortirana. U suprotnom, ako se najmanji elementi nalaze na kraju liste, sortiranje ove liste će biti jako sporo jer se mali elementi jako sporo pomiču na početak liste. Učinkovitost najbržeg algoritma tj. algoritma brzog sortiranja ovisi o rasporedu ulaznih podataka te ovisi o točnom izračunu srednje vrijednosti koja prilikom sortiranja dijeli jedan niz na podnizove tj. ovisi o odabiru pivota.

Da bi razumjeli vremensku složenost prvo će biti prikazana tablica s objašnjenim vremenskim izrazima (Tablica 1.). U sljedećoj tablici (Tablica 2.) dan je prikaz samih algoritama s njihovim vremenskim izrazima u najboljem, prosječnom i najgorem slučaju.

Oznaka	Složenost
1	Konstanta
O(n)	Linearna
On^2	Kvadratna
On^3	Kubična
log n	Logaritamska
n log n	Loglinearna

Tablica 14. Prikaz funkcije vremena izvršavanja algoritama

Izvor: modificirano prema (Živković, 2010)

Algoritmi	Vremenska složenost		
	Najbolji slučaj	Prosječni slučaj	Najgori slučaj
Sortiranje izborom najmanjeg elemenata	$O(n^2)$	$O(n^2)$	$O(n^2)$
Sortiranje zamjenom susjednih elemenata	O(n)	$O(n^2)$	$O(n^2)$
Jednostavno sortiranje umetanjem	O(n)	$O(n^2)$	$O(n^2)$
Višestruko sortiranje umetanjem	O(n log n)	Hibbard-ov inkrement $O(n^{5/4})$ $O(n^{3/2})$	
Algoritam brzog sortiranja	O(n log n)	O(n log n)	$O(n^2)$
Algoritam sortiranja spajanjem	O(n log n)	O(n log n)	O(n log n)
Algoritam sortiranja s pomoću hrpe	O(n log n)	O(n log n)	O(n log n)

Tablica 15. Prikaz algoritama za sortiranje i njihove vremenske složenosti

Izvor: modificirano prema (Geeksforgeeks, n.d.)

6. ALGORITMI ZA PRETRAŽIVANJE

Pomoću pretraživanja pronalazimo telefonski broj prijatelja, određeni recept ili najkraći put do traženog odredišta. „Algoritam pretraživanja je algoritam koji daje rješenje nekog problema nakon evaluacije skupa mogućih rješenja. Skup svih mogućih rješenja problema naziva se „prostor pretraživanja““ (Pavlica, 2014).

„Pretraživanje je proces koji za cilj ima pronalaženje elemenata niza, lista, stabla ili neke druge strukture podataka“ (Stojanović, 2016, p. 2).
Dvije su osnovne strategije pretraživanja:

1. slijepo (linearno) pretraživanje (engl. blind, uninformed search)
2. usmjereno (binarno) pretraživanje (engl. directed, informed, heuristic search)

Svojstva algoritama za pretraživanje su (Algoritam, 2011):

- potpunost (engl. completeness) – algoritam je potpun ako pronalazi rješenje uvijek kada ono postoji,
- optimalnost (engl. optimality, admissibility) – algoritam je optimalan ako pronalazi optimalno rješenje (ono s najmanjom ocjenom),
- prostorna složenost – ukupni broj čvorova u memoriji,
- vremenska složenost → broj proširenih čvorova.

Vremenska i prostorna složenost mjere se u smislu (Uninformed search, n.d.):

- b – maksimalno grananje faktora stabla pretraživanja
- d – dubina najmanjeg rješenja
- m – maksimalna dubina (može biti beskonačna)

Kao i kod algoritama za sortiranje i ovdje gledamo vremensku složenost.
Kod strategije slijepog pretraživanja (engl. „uninformed search“) nemamo informaciju o problemu kojeg pokušavamo riješiti. Kod strategije usmjerenog pretraživanja (engl. informed search) koristimo se heurističkim podacima tj. imamo informacije o udaljenosti između početka i ciljanog rješenja do kojeg želimo doći. Na temelju tih informacija i međusobnoj udaljenosti odabiremo naše sljedeće rješenje.

Osnovne metode strategije slijepog pretraživanja prostora stanja su: linearno pretraživanje (engl. linear search), binarno pretraživanje (engl. binary search), pretraživanje najprije u širinu (engl. breadth-first search), pretraživanje najprije u dubinu (engl. depth-first search), pretraživanje do određene dubine (engl. depth limited), algoritam pretraživanja s jednolikom cijenom (engl. uninformed-cost search), iterativno pretraživanje najprije u dubinu (engl. depth-first iterative deepening search) i algoritam dvosmjernog pretraživanja (engl. bidirectional search).
Strategija usmjerenog pretraživanja obuhvaća sljedeće metode: pretraživanje usponom na vrh (engl. hill climbing), pretraživanje najboljeg prvog (engl. best-first search), ograničeno pretraživanje po širini (engl. beam search) A i A* algoritme (Atwood & Spolsky, 2017). Metode koje su objašnjene u ovom radu su: linearno pretraživanje, binarno pretraživanje, pretraživanje najprije u širinu, pretraživanje najprije u dubinu, pretraživanje s jednolikom cijenom te pretraživanje najboljeg prvog i A* (čita se A star).

6.1 Strategija slijepog pretraživanja

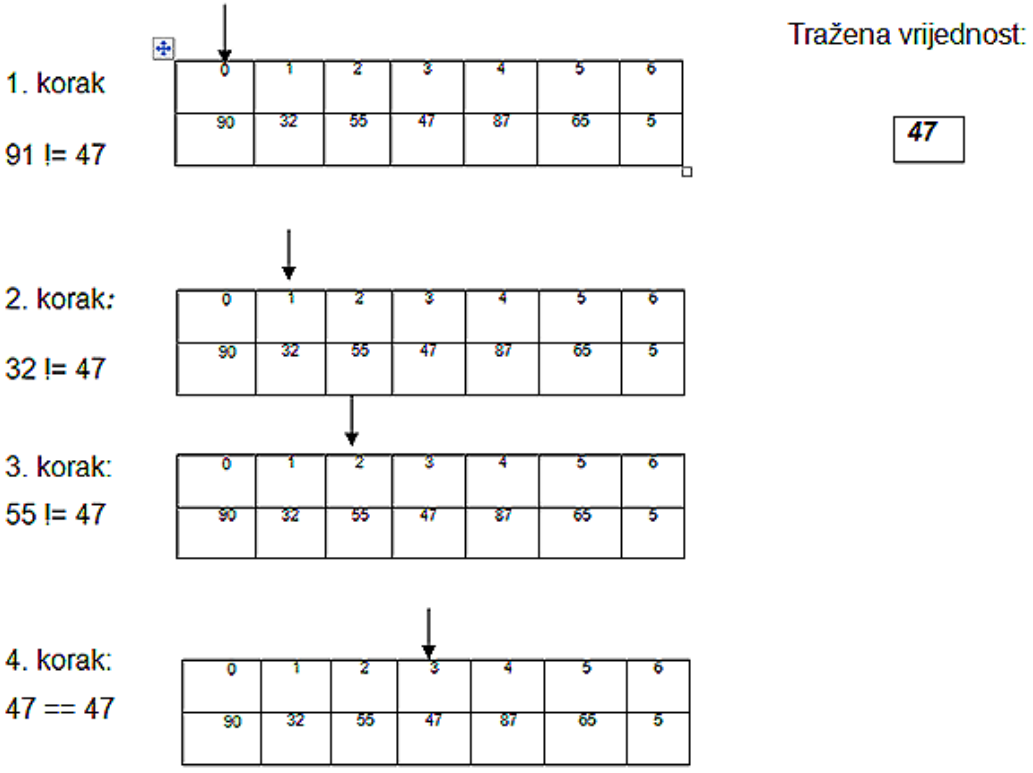
Pretraživanje liste je jedan od najjednostavnijih algoritama za pretraživanje. Pretraživanje liste možemo izvršiti pomoću linearnog ili binarnog pretraživanja liste. „Strategija slijepog pretraživanja (engl. „blind search“) ne uzima u obzir specifičnu prirodu problema. Na taj način „slijepi“ algoritmi pretraživanja mogu se poopćiti, odnosno na isti način mogu se primijeniti na široko područje problema“ (Bojčetić, et al., 2005, p. 24).

6.1.1 Linearno pretraživanje

Linearno pretraživanje obavlja se prema određenom ključu koji sadrži informaciju što tražimo. Linearno pretraživanje (engl. linear search) je najjednostavnije i najčešće korišteno pretraživanje. Ovo pretraživanje je i najsporije jer prolazi kroz svaki element liste od početka do kraja sve dok se ne nađe traženi ključ (element).

Kada se traženi ključ (element) pronađe pretraživanje se prekida te je njegov rezultat broj njegovog indeksa u listi. Inače se vraća vrijednost *-1* koja ne može biti indeks

(Miškovic, 2016/2017). Prednost ovog algoritma je u njegovoj primjeni na nesortiranu listu.



Slika 16. Prikaz rada linearnog pretraživanja liste

Izvor: modificirano prema (Carić, 2013)

```
int trazi (int a[], int n, int t) {
 int i;
   for (i=0; i<n; i++)
     if (a[i]==t) return (i);

     return (-1);
}
```

Slika 17. Implementacija algoritma linearnog pretraživanja u C++

Izvor: modificirano prema (Stojanović, 2016, p. 1)

Najbolji slučaj je taj u kojem prva usporedba vraća traženo, tj. zahtijeva jednu usporedbu te iznosi O(1) operacija. U najgorem slučaju, ako pretražujemo niz od *n* elemenata, potrebno je O(n) operacija.

Prosječna vremenska složenost ovisi o mogućnosti da se traženi ključ nađe u nizu kojeg pretražujemo (Morris, 1998).

6.1.2 Binarno pretraživanje

Binarno pretraživanje započinje na sredini liste. Srednja vrijednost se dobije zbrajanjem prvog indeksa sa zadnjim indeksom te dijeljenjem s brojem 2. Ako je broj elemenata u podnizu paran, rezultat srednje vrijednosti se zaokružuje na donju vrijednost. Dobiveni središnji element se zatim uspoređuje s traženim elementom. Ako se utvrdi da su vrijednosti jednake pretraživanje se prekida.
Ako je dobiveni element manji od traženog elementa, pretraživanje se nastavlja u prvom dijelu liste, a ako je dobiveni element veći od traženog elementa, pretraživanje se nastavlja na drugom dijelu niza (Miškovic, 2016/2017).

„Binarno pretraživanje je algoritmima isto što i kotač mehaničarima“ (Manber, 1989.). Binarno pretraživanje započinje na polovici liste i nastavlja se stalnim rastavljanjem intervala. Ako pretražujemo listu duljine *n*, onda ukupni broj usporedbi koje se moraju izvršiti iznose $log_2 n-1$. Za binarno pretraživanje lista mora biti sortirana i mora biti omogućen direktan pristup svim elementima liste. Najgori slučaj vremenske složenosti kod ovog pretraživanja je O(n). To je slučaj kada se traženi element ne nalazi u listi (Bojčetić, et al., 2005, p. 24).

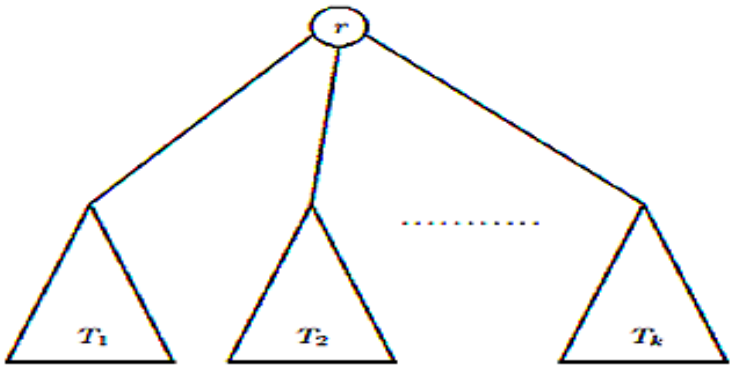
```
int BinarySearch (int arr [], int value, int left, int right) {
    while (left <= right) {
    int middle = (left + right) / 2;
    if (arr[middle] == value)
            return middle;
    else if (arr[middle] > value)
            right = middle -1;
    else
            left = middle + 1;
}

return -1;
}
```

Slika 18. Implementacija algoritma binarnog pretraživanja

Izvor: modificirano prema (Algolist, n.d.)

6.1.3 Pretraživanje stabla

Da bi razumjeli algoritme pretraživanja stabla prvo se moraju shvatiti osnove binarnog stabla kao i samo stablo. Svaki put kada pretražujemo stablo pretražujemo njegove čvorove. Svako stablo ima svoj korijen. „Korijen (engl. root) je najviši čvor u stablu. U stablu svaki čvor ima jednog roditelja. Pojam dijete i roditelj se određuje za svaki čvor pojedinačno. Potomci su djeca i djeca djece, a preci su roditelji roditelja. Brat nekom čvoru je čvor s kojim dijeli roditelja. Podstablo je dio stabla. Svaki čvor zajedno sa svim svojim potomcima čini podstablo u kojemu je on korijen“ (Kusalić, 2010, p. 281). Stablo može biti uređeno, binarno i k-stablo. U ovom radu je objašnjeno binarno stablo i pretraživanje binarnog stabla. Binarno stablo (engl. binary tree) je konačan skup podataka istog tipa koji ima istaknuti čvor *r* koji se zove korijen od T, a ostali čvorovi grade uređeni par (T_L, T_R) manjih binarnih stabala (Manger, 2014, p. 68).



Slika 19. Urednost stabla i njegova podstabla

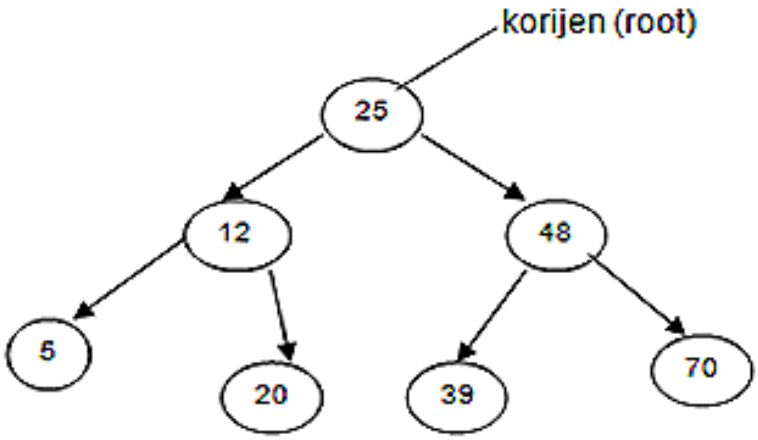
Izvor: modificirano prema (Manger, 2014, p. 48)

„Binarno stablo pretraživanja (engl. binary search tree) jedna je od najvažnijih struktura podataka u računalnoj znanosti. Za svaki čvor ove strukture vrijedi (Bojčetić, et al., 2005, p. 15):

- vrijednost u čvoru je veća od svih vrijednosti u lijevom podstablu i manja od svih vrijednosti u desnom podstablu“.

Bitno je napomenuti da navedeni algoritmi se ne odnose samo na polja, redove i slične strukture podataka, već služe kako bi se različiti problemi umjetne inteligencije prilagodili računalnom procesu kroz pojednostavljene korake. Prostor pretraživanja prostora stanja svodi se na pretraživanje usmjerenog grafa gdje vrhovi grafa predstavljaju stanja, a lukovi prijelaze između stanja.

Pretraživanjem usmjerenog grafa izgrađujemo stablo pretraživanja. Stablo gradimo tako da pojedine čvorove proširujemo. Postoje otvoreni i zatvoreni čvorovi. Otvoreni čvorovi su prošireni, a zatvoreni čvorovi još nisu prošireni. Redoslijed kojim proširujemo čvorove određujemo strategiju pretraživanja (Bašić, Dalbelo & Šnajder, 2010/2011).



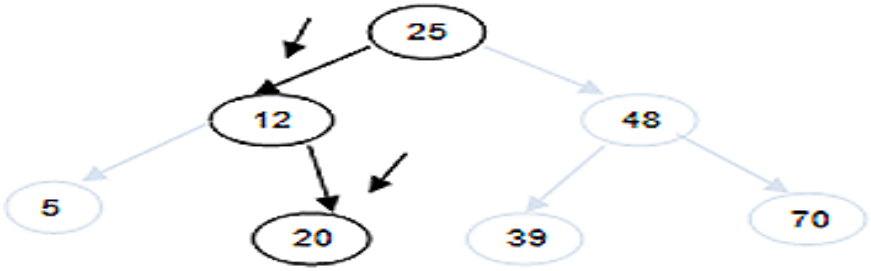
Slika 20. Binarno stablo pretraživanja

Izvor: modificirano prema (Bojčetić, et al., 2005, p. 15)

Na primjer, recimo da se želi pronaći broj 20 u binarnom stablu pretraživanja. Započinje se sa čvorom (korijenom) te ga se uspoređuje s traženim brojem. U ovom slučaju traženi broj je 20, a on je manji od čvora (korijena) koji iznosi 25. Stoga se gleda lijeva strana stabla i njegova podstabla.
Kao što je već prethodno spomenuto, lijeva strana stabla predstavlja brojeve manje od čvora (korijena), a desna strana predstavlja brojeve veće od čvora (korijena).

Ako je stablo balansirano, kažemo da razlika između njih nije veća od 1. Također, ako je stablo balansirano počinjemo pretraživanje sa *n* čvorova i kada odbacimo jedan čvor dobijemo *n/2* čvora.

Stoga prostor pretraživanja biti će reduciran na *n/2*, u sljedećem koraku bit će *n/4* i tako dalje sve dok se ne nađe traženi element, odnosno sve dok se prostor pretraživanja ne reducira na samo jedan čvor. Stablo može biti i nestabilno no to nije u ovom slučaju. Zatim pogledamo lijevog roditelja od čvora 25, a to je 12. Utvrdili smo da je 12 < 20, ali isto tako i da je 48 > 20 te se traženi element ne može pronaći na desnom podstablu. Smanjili smo područje pretraživanja na 3 čvora s vrijednostima lijevog podstabla: 12, 5 i 20. Sada znamo da trebamo krenuti pregledavati desnu stranu čvora roditelja jer je 12 < 20. Gledamo desnu stranu čvora te uspoređujemo je li broj 20 jednak traženom broju i potvrđujemo. Pretraživanje se zaustavlja nakon što se pronađe traženi element.



Slika 21. Prikaz rada pretraživanje binarnog stabla

Izvor: Modificirano prema (Stojanović, 2016, p. 5)

Kod pretraživanja stabla „posjećujemo“ čvor i pritom čitamo podatke. Možemo ići u širinu ili u dubinu pretraživanja. Svaki od ovih algoritama koristi određenu strukturu podataka te kako bismo mogli raditi s navedenim algoritmima moramo prvo razumjeti te uvidjeti razliku između reda (engl. queue) i stoga (engl. stack).

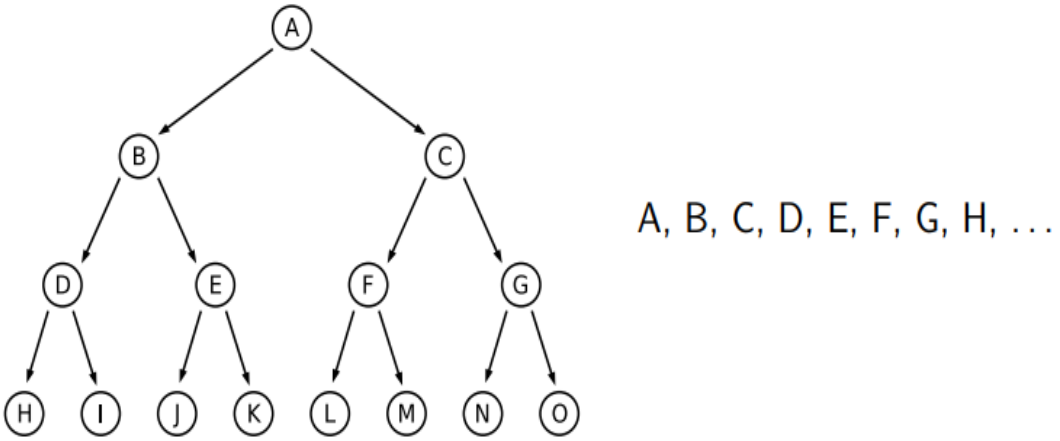
„Linearna struktura u kojoj se elementi mogu dodavati ili oduzimati samo na jednom kraju zove se stog (engl. stack)“ (Pavlica, 2014). Nije moguće izvaditi ili umetnuti bilo koji element iz sredine prije nego se maknu svi elementi iza njega.
Iz toga slijedi da je zadnji dodani element ujedno i onaj koji će se prvi izvaditi. Uobičajeni naziv za takvu strukturu podataka kod koje se zadnji pohranjeni element prvi uzima u obradu je „LIFO“ (engl. last in first out).

„Red (engl. queue) je linearna struktura u kojoj se elementi dodaju isključivo na jednom kraju, a oduzimaju isključivo na drugom kraju. Drugi uobičajeni naziv za ovakvu strukturu podataka je „FIFO“ (engl. first in - first out)“ (Pavlica, 2014).

6.1.4 Pretraživanje najprije u širinu

U stablu pretraživanja pretražujemo čvor po čvor. Algoritam pretraživanja najprije u širinu će prvo potražiti rješenja na prvoj razini pa se zatim pomiče na drugu razinu i tako dalje, a pritom se koristi redom kao strukturom podataka. Pretraživanje započinje od početnog (korijenskog) čvora kojeg sami odaberemo. „Početni čvor se stavlja na početak reda, ispituje se, a zatim se sve čvorove koji su povezani s njim (djeca) dodaju u red. Svi čvorovi koji se trenutno nalaze u redu ispituju se prije svoje

djece tj. nasljednika. Općenito, svi čvorovi na dubini *d* u pretraživanju stabla su prošireni prije bilo kojeg čvorova na *d+1*, tj. pretražujemo po razinama“ (Splithorizont, 2013). U nastavku prikazan je princip rada algoritma pretraživanja najprije u širinu.



Slika 22. Prikaz rada algoritma pretraživanja najprije u širinu

Izvor: modificirano prema (Bašić, Dalbelo & Šnajder, 2010/2011)

Vremenska složenost iznosi:

$$b + b^2 + b^3 + \ldots + b^d \quad \rightarrow \quad O(b^{d+1})$$

```
static void BFS(Graph G, int start) {
    Queue<integer> Q = newAQueue<Integer> (G, n());
    Q.enqueue(start);
    G.setMarko (start, VISITED);
    while (Q-lenght() > 0) {
        int v = Q.dequeue ();
        PreVisit (G,V);
        for (int w= G.first(v); w < G.n (); w = G.next (v, m))
          if (G.getMark(w) == UNVISITED) {
            G.setMark (w, VISTED);
            Q.enqueue(w);
          }
          PostVisit (G,v);
        }
}
```

Slika 23. Implementacija algoritma pretraživanja najprije u širinu

Izvor: modificirano prema (Shaffer , 2008)

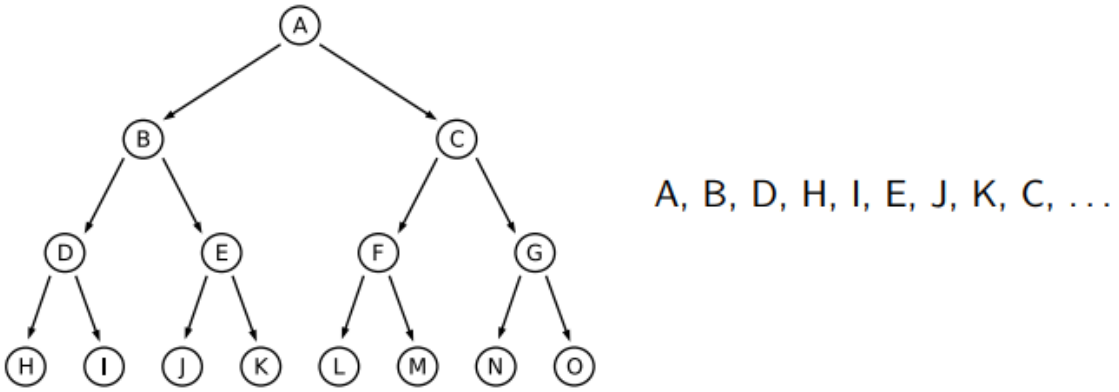
6.1.5 Pretraživanje najprije u dubinu

Pretraživanjem najprije u dubinu pretražujemo stablo što je brže moguće tako da uvijek generiramo djecu zadnjeg proširenog čvora dok ne dođe do rješenja (ili dok ne prođe zadani limit dubine) (Bojčetić, et al., 2005). Pretraživanje najprije u dubinu uvijek prvo proširuje najdublji čvor u stablu pretraživanja. Tek kada dosegne stanja koja nemaju sljedbenika, vraća se na pliće razine. Proširene čvorove moramo dodavati na početak otvorene liste (Bašić, Dalbelo & Šnajder, 2010/2011).

Kod obilaska stabla u dubinu imamo tri strategije:

- preorder obilazak – prvo polazimo od korijena stabla, zatim na lijevo podstablo i zatim na desno podstablo,
- inorder obilazak– prvo polazimo od lijevog podstabla, pa do korijena pa zatim na desno podstablo,
- postorder obilazak – prvo polazimo od lijevog podstabla na desno podstablo pa zatim do korijena.

Što se tiče vremenske i prostorne složenosti ona iznosi O(bm).



Slika 24. Prikaz rada algoritma pretraživanja najprije u dubinu

Izvor: modificirano prema (Bašić, Dalbelo & Šnajder, 2010/2011)

```
static void DFS (Graph, G, int v) {
    PreVisit (G, v);
    G.setMark (v, VISITED);
    for (int w = G.first(v); w < G.n(); w = G.next(v, w))
      if (G.getMark (w) == UNVISITED)
      DFS (G, w);
      PostVisit (G, v);
}
```

Slika 25. Implementacija algoritma pretraživanja najprije u dubinu

Izvor: modificirano prema (Shaffer , 2008, p. 400)

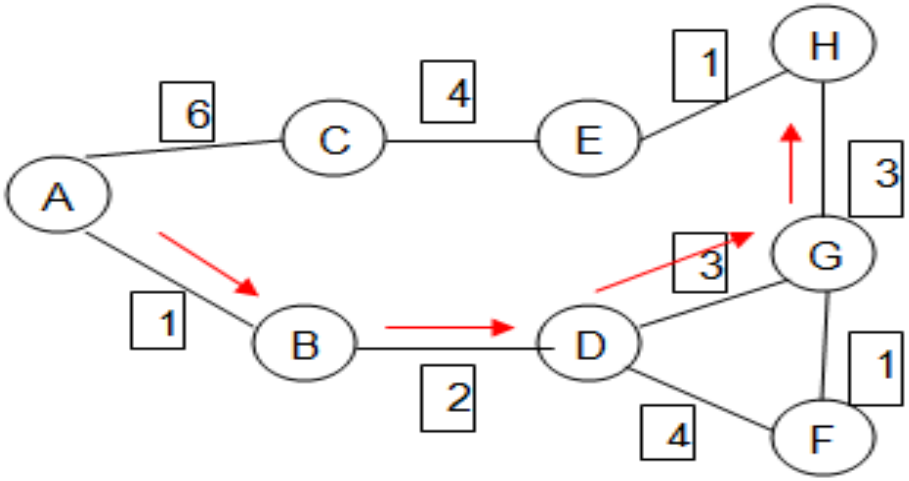
6.1.6 Algoritam pretraživanja s jednolikom cijenom

Algoritam pretraživanja s jednolikom cijenom, koji proširuje čvor s najmanjim putnim troškom, prikazuje se funkcijom g(n). Funkcija g(n) prikazuje trošak puta pretraživanje stabla od čvora do čvora *n* korijena. Svaki čvor ima svoj trošak kojim upravljamo smjerom pretraživanja. Algoritam pretraživanja s jednolikom cijenom uvijek proširuje čvor s najmanjim troškom od početnog čvora do trenutnog čvora. "Prostorna i vremenska složenost ovog algoritma je $b[^{1+C*/e}]$“ (Bašić, Dalbelo & Šnajder, 2010/2011).

```
      Vratiti rješenje ili neuspjeh

      Q je prioritetni red koji se
      razvrstava po trenutnom
      trošku od početka do cilja

      Korak 1. Dodati početno stanje
      (ili korijen) u red čekanja.
      Korak 2. Sve dok cilj nije postignut ili red nije prazan
Radi:
      Korak 2.1. Ukloniti prvi put iz reda čekanja.
      Korak 2.2. Izradite novi put proširivanjem
     prvog puta do svih susjednih čvorova.
      Korak 2.3. Ukloniti sve nove putove iz petlje.
      Korak 2.4. Dodati preostale nove putove,
     ako ih ima, u red čekanja.
      Korak 2.5. Sortirati cijeli red tako da
      putevi sa najnižim troškovima
     budu naprijed.
Kraj.
```

Slika 26. Implementacija algoritma pretraživanja s jednolikom cijenom

Izvor: modificirano prema (Abraham & Grosan, 2011, p. 32)



Slika 27. Prikaz rada algoritma pretraživanja s jednolikom cijenom

Izvor: modificirano prema (Artificial intelligence, n.d.)

6.2 Strategija usmjerenog pretraživanja

Heuristika (prema grčki εὑρίσκειν: nalaziti, otkrivati) postupak koji vodi prema otkriću ili ga potiče (Leksikografski zavod Miroslav Krleža, 2011).

Usmjerene (engl. informed, heuristic) strategije pretraživanja koriste evaluacijsku funkciju h za usmjeravanje pretraživanja stabla. Funkcija *h(x)* mjeri ili procjenjuje trošak najkraćeg puta između čvora *x* i ciljnog čvora.

Jedni od najboljih algoritama usmjerenog pretraživanja su:

- algoritam pretraživanja najboljeg prvog (engl. best first search) koji obuhvaća metode pohlepnog pretraživanja (engl. greedy search) i A* (čita se A star) pretraživanje.

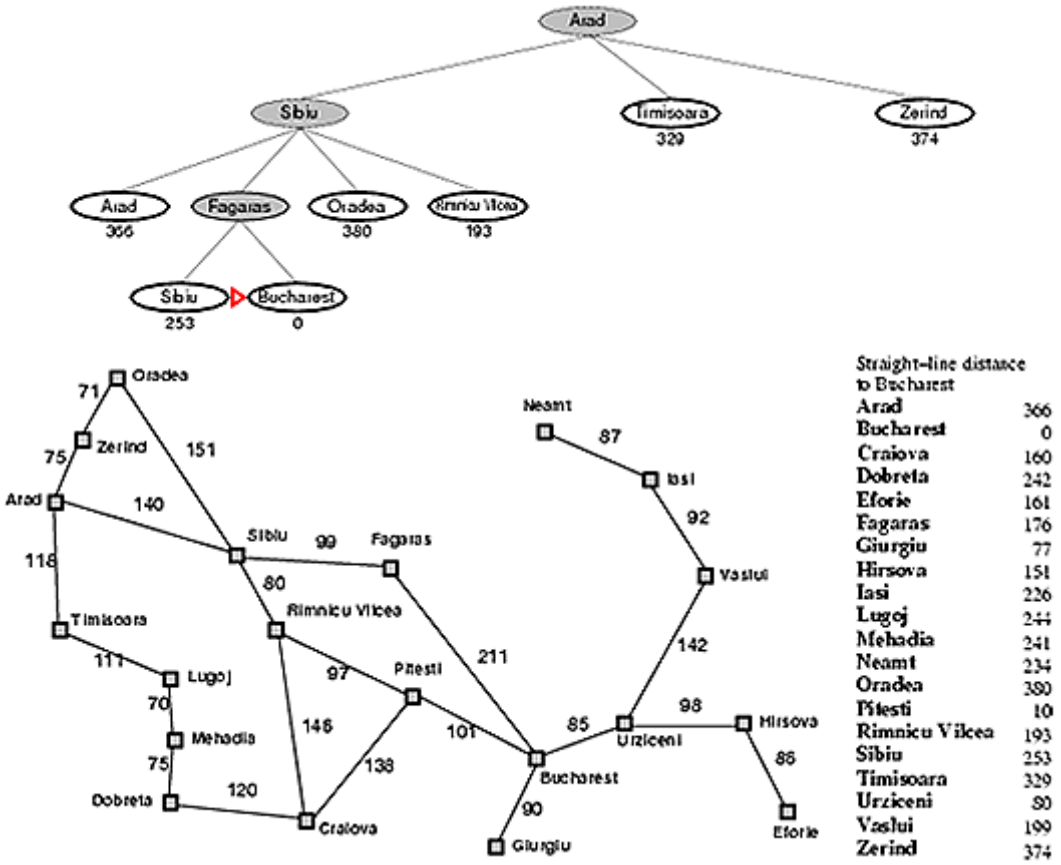
„Obje metode procjenjuju najmanju udaljenost između čvora *x* i ciljnog čvora koristeći pritom evaluacijsku funkciju h(x). Ako je čvor *x* traženi čvor onda je h(x)=0“ (Rothlauf, 2011, p. 64).

Potrebno je razlikovati sljedeće definicije:

- g(n) funkcija troška puta→ trošak puta od korijena do čvora n koji je trenutno pronađen,
- h(n) heuristička funkcija →procjenjuje trošak puta od čvora n do „najbližeg“ čvora,
- f(n) evaluacijska funkcija→mjeri kolika je vjerojatnost da je čvor n dio rješenja
  →postoji samo jedna mogućnost: f(n) = g(n) + h(n).

6.2.1 Algoritam pretraživanja najboljeg prvog

U algoritmu pretraživanja najboljeg prvog koristimo evaluacijsku funkciju pomoću koje odabiremo i proširujemo čvor s najmanjim troškom puta. Međutim, pretraživanje najboljeg prvog zahtijeva puno vremena jer ovaj algoritam prolazi kroz skoro sve moguće putove. Kao i kod pretraživanja u širinu, algoritam pretraživanja najboljeg prvog koristi red (engl. queue), ali umjesto da uzme prvi čvor iz reda uzima najbolji čvor. Prošireni čvorovi, nakon što im je dodijeljena vrijednost, dodani su u listu. Funkcija troška f(n) je primijenjena na svaki čvor.
Čvorovi s manjom vrijednošću su prošireni kasnije (Abraham & Grosan, 2011).
Ovaj algoritam se zove i pohlepno pretraživanje (engl. greedy search).
Algoritam pohlepnog pretraživanja najboljeg prvog proširuje čvor za koji se smatra da ima najmanji trošak čvora prema traženom cilju. Vremenska složenost ovog algoritma iznosi O(b^m).



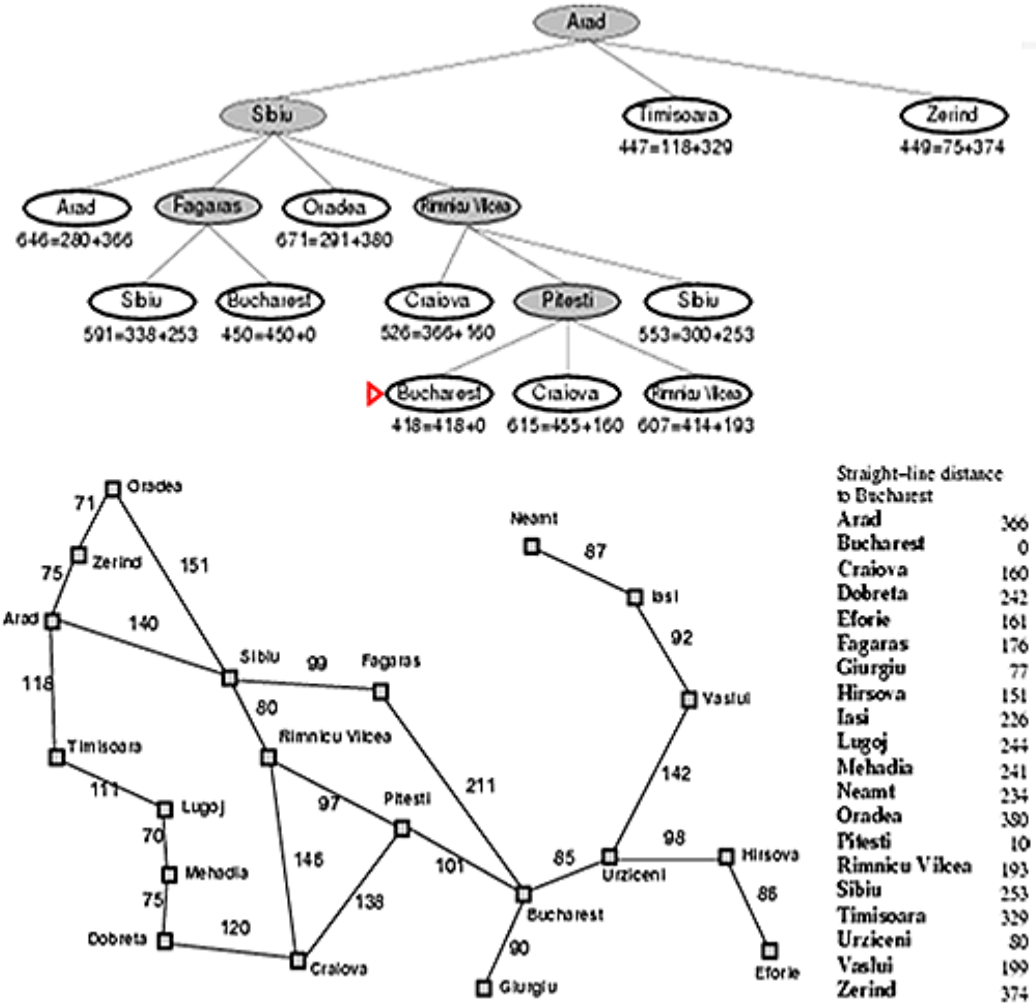
Slika 28. Prikaz rada algoritma pretraživanja najboljeg prvog

Izvor: modificirano prema (Informed search algorithms, n.d.)

6.2.2 A* pretraživanje

U A* pretraživanju funkcija evaluacije f(n) predstavlja ukupni trošak. U funkciji *f(n)* = *g(n)* + *h(n),* g(n) je trošak dosegnut do *n*, dok je h(n) procijenjeni trošak od *n* do ciljanog stanja. A* algoritam koristimo za pronalazak najkraćeg puta. (Joshi & Kulkarni, 2015).

„A* algoritam kombinira algoritam pretraživanja neinformiranog troška zajedno s algoritmom pretraživanja najboljeg prvog. Trošak f(x) od čvora *x* je izračunat kao f(x)=g(x)+h(x) gdje je g(x) udaljenost između čvora korijena i *x* i h(x) procjenjuje najmanju udaljenost između *x* čvora i traženog čvora. A* algoritam uvijek proširuje čvor *x* s minimalnim f(x) i zaustavlja se nakon što je traženi čvor pronađen“ (Rothlauf, 2011, p. 64). Što se tiče vremenske i prostorne složenosti ona iznosi $O(b^d)$.



Slika 29. Prikaz rada A algoritma u rješavanju problema udaljenosti između gradova*

Izvor: modificirano prema (Informed search algorithms, n.d.)

7. USPOREDBA ALGORITAMA ZA PRETRAŽIVANJE

Algoritam radi brže ako ima neke informacije. Stoga u brže algoritme spadaju algoritmi usmjerenog pretraživanja. Na sljedećoj tablici (Tablica 3.) prikazani su algoritmi pretraživanja i njihova vremenska i prostorna složenost.

Vremenska složenost	Najbolji slučaj	Prosječni slučaj	Najgori slučaj		
Linearno pretraživanje	O(1)	O(N)	O(N)		
Binarno pretraživanje	O(1)	O(log N)	O(log N)		
	Pretraživanje najprije u širinu	Pretraživanje najprije u dubinu	Pretraživanje s jednolikom cijenom	Pretraživanje najboljeg prvog	A*
Vrijeme	O(bd+1)	$O(b^m)$	$b[^{1+C^*/e}]$	O(bm)	O(bd)
Prostor	O(bd+1)	$O(b^m)$	$b[^{1+C^*/e}]$	O(bm)	O(bd)

Tablica 30. Prikaz algoritama za pretraživanje i njihove vremenske složenosti

Izvor: modificirano prema (Phyton-textbok, n.d.) i (Podravec, n.d.)

Najpoznatiji algoritmi za pretraživanje su linearno i binarno pretraživanje. Binarno pretraživanje je bolje od linearnog pretraživanja jer se temelji na već sortiranoj listi te stoga brže pronalazi traženi element. Najgori slučaj je kada se traženi element nalazi na kraju liste. Linearno pretraživanje za razliku od binarnog pretraživanja ne zahtijeva da lista bude sortirana. Uspoređujući algoritme slijepog i usmjerenog pretraživanja, točnije algoritme pretraživanja neinformiranog troška i pretraživanje najboljeg prvog, uviđamo da je algoritam pretraživanja najboljeg prvog brži. Bitna razlika između ova dva algoritama je u evaluacijskoj funkciji.

Kod algoritma pretraživanja neinformiranog troška nema nikakve informacije o veličini troška te algoritam proširuje najmanji čvor čineći to u svim smjerovima jer se ne zna koji je smjer put s najmanjim troškom i najbržim dolaskom do cilja.

Međutim, algoritam pretraživanja najboljeg prvog koristi se također evaluacijskom funkcijom. No, u ovom slučaju postoje neke konkretne informacije o trošku. Oba algoritma proširuju čvorove s najmanjom veličinom troška. Ipak, algoritam pretraživanja najboljeg prvog teži da ih ima što manje. Algoritam pretraživanja najboljeg prvog pokušava zapravo odrediti za svaki smjer koji je čvor potencijalno najbliži određenom cilju.

Postoji još jedna bitna razlika u usmjerenim algoritmima pretraživanja, a to je ona između algoritma pretraživanja najboljeg prvog i A* algoritma. Oba algoritma koriste evaluacijsku funkciju. Algoritam pretraživanja najboljeg prvog, prilikom traženja najkraćeg puta, ne gleda koliki je trošak sveukupno od početka do sljedećeg čvora, već uspoređuje vrijednosti čvorova i utvrđuje koji čvor u tom trenutku ima najmanju vrijednost. Za razliku od ovog algoritma, A* algoritam temelji se na funkciji koja u obzir uzima sveukupni trošak uzimajući u obzir trošak od početnog čvora do dolaska do ciljanog čvora tj. koristi funkciju $f(n) = g(n) + h(n)$. Brzina algoritama za pretraživanje ovisi o heurističkoj funkciji.

8. ZAKLJUČAK

Promatranje pojma pretraživanja nas asocira na promišljanje o problematici: gdje se nalazimo, gdje želimo stići ili što želimo pronaći. Ovisno o tome kamo se želi doći, što se želi pronaći ili što se želi napraviti pronalazimo algoritam koji rješava naš specifičan problem. Kako bi algoritam radio, mora imati određene korake koje treba izvršiti. Kada imamo ili spoznamo određenu informaciju želimo ju i pohraniti i iskoristiti. Danas postoji nebrojeno puno problema s kojima se suočavamo i pokušavamo ih riješiti, a u većini slučajeva to nije jednostavno.

Postoji više načina kako možemo sortirati i pretraživati i neke od načina koristimo u svakodnevnom životu. Na svakom koraku nailazimo na skup instrukcija kako bi u konačnici došli do bržeg rješenja. Prilikom rješavanja problema bitno je imati u vidu način njegova rješavanja, vremensku složenost pojedinog algoritma (najbolju, prosječnu i najgoru) kao i veličinu skupa koji se promatra. Na temelju veličine podataka razlikujemo učinkovitost algoritama za sortiranje. Najsporiji algoritmi za sortiranje, koji su ujedno dobri za manje skupove podataka, su mjehurićasto sortiranje i sortiranje izborom najmanjeg elementa. Algoritam mjehurićastog sortiranja sadrži problem zečeva i kornjača. Ako se najmanji elementi nalaze na kraju liste jako sporo se pomiču prema početku. Za brzo pretraživanje najbolje je odabrati binarno pretraživanje jer se koristi sortiranom listom. Kod algoritama za pretraživanje više se okrećemo prema funkciji koju koristimo prilikom pretraživanja.

Postoje tri bitne funkcije koje je potrebno poznavati kako bi se shvatio pojedini algoritam pretraživanja, a to su funkcija troška puta, heuristička funkcija i evaluacijska funkcija. Algoritam A* smatra se najboljim algoritmom jer kombinira algoritam pretraživanja neinformiranog troška i algoritam pretraživanja najboljeg prvog tj. promatra sveukupni trošak puta od početka do traženog cilja. Ostaje pitanje koji je algoritam najbolje odabrati za pojedini problem. Stoga moramo prvenstveno utvrditi problem koji želimo riješiti te na temelju karakteristika poput veličine, tipa podataka i željenog cilja odrediti algoritam koji bi bio najprihvatljiviji.

9. LITERATURA

Abraham, A. & Grosan, C., 2011. *Intelligent Systems: A Modern Approach.* Berlin: Springer Science & Business Media.

Algolist, n.d. *Algorithms and Data Structures with implementation in Java and C++.* [Mrežno]
Dostupno na: http://www.algolist.net/Algorithms/Sorting/Quicksort,
pristupano: 07.12.2017.

Algoritam, 2011. *Serijsko (sekvencijalno - linearno) pretraživanje.* [Mrežno]
Dostupno na: http://razno.sveznadar.info/02-2razred/5-Algoritam/4-SerijskaPretraga.htm,
pristupano: 08.12.2017.

Artificial intelligence, n.d. *Artificial intelligence.* [Mrežno]
Dostupno na: https://www.slideshare.net/QAUWorrier/lec2-44360794,
pristupano: 08.12.2017.

Atwood, J. & Spolsky, J., 2017. *Stack overflow.* [Mrežno]
Dostupno na: https://stackoverflow.com/questions/39760905/what-is-the-main-difference-between-informed-search-and-uninformed-search-algori/39777979,
pristupano: 08.12.2017.

Basu, S. K., 2013. *Design methods and analysis of algorithm.* 2nd ur. Delhi: PHI Learning Private Limited.

Bašić, Dalbelo, B. & Šnajder, J., 2010/2011. *Rješavanje problema pretraživanjem prostora stanja.* [Mrežno]
Dostupno na:
http://www.ieee.hr/_download/repository/UI_2_pretrazivanje_prostora_stanja.pdf,
pristupano: 08.12.2017.

Baumgartner, A., 2013/2014. *Programerska metodologija.* [Mrežno]
Dostupno na: http://www.mathos.unios.hr/spa/Files/materijali/SPA_skripta_ch01.pdf,
pristupano: 05.12.2017.

Bojčetić, N., Marjanović, D. & Pavković, N., 2005. *Programiranje i algoritmi, Skripta, drugi dio.* [Mrežno]
Dostupno na:
http://ttl.masfak.ni.ac.rs/SUK/Programiranje%20i%20algoritmi%202.pdf,
pristupano: 09.12.2017.

Carić, 2013. [Mrežno]
Dostupno na: https://www.weboteka.net/fpz/Algoritmi%20i%20programiranje/e-student/008_2013-04-29%20P08%20ALGPRO%20-%20sortiranje%20i%20pretra%C5%BEivanje.pdf,
pristupano: 08.12.2017.

Čičak, M., n.d. *Algoritmi.* [Mrežno]
Dostupno na: https://element.hr/artikli/file/1135,
pristupano: 05.12.2017.

Geeksforgeeks, n.d. *Time Complexities of all Sorting Algorithms,.* [Mrežno]
Dostupno na: https://www.geeksforgeeks.org/time-complexities-of-all-sorting-algorithms/,
pristupano: 03.01.2018.

Gupta, S. & Prakash, V., 2018. *hackerearth.* [Mrežno]
Dostupno na: https://www.hackerearth.com/practice/algorithms/sorting/heap-sort/tutorial/,
pristupano: 19.03.2018.

Harris, S. & Ross, J., 2006. *Beginning Algorithms.* Canada: Wiley Publishing, Inc, Indianapolis, Indiana.

Informed search algorithms, n.d. *chapter 4.* [Mrežno]
Dostupno na:
https://www.ics.uci.edu/~welling/teaching/271fall09/InfSearch271f09.pdf,
pristupano: 04.01.2018.

Informed search algorithms, n.d. *Informed search algorithms.* [Mrežno]
Dostupno na:
https://www.google.hr/url?sa=t&rct=j&q=&esrc=s&source=web&cd=9&ved=0ahUKEwiHurnluMbYAhUHLlAKHS5qDGwQFghqMAg&url=https%3A%2F%2Fwww.ics.uci.edu%2F~welling%2Fteaching%2FICS171Fall05%2FSearch061005.ppt&usg=AOvVaw3WmWFcoU6v9Lnx_xEBwrJO,
pristupano: 09.12.2017.

Joshi, P. & Kulkarni, P., 2015. *ARTIFICIAL INTELLIGENCE: Building Intelligent Systems.* Delhi: PHI Learning Pvt. Ltd.

Joshi, V., 2017.. *basecs.* [Mrežno]
Dostupno na: https://medium.com/basecs/heapify-all-the-things-with-heap-sort-55ee1c93af82,
pristupano: 24.02.2018.

Krčadinac, V., 2016/2017. *Osnove algoritama.* [Mrežno]
Dostupno na: https://web.math.pmf.unizg.hr/nastava/oa/oa-skripta.pdf,
pristupano: 05.12.2017.

Kusalić, D., 2010. *Napredno programiranje i algoritmi u C-u i C++-u.* 1st ur. Zagreb: ELEMENT d.o.o..

Leksikografski zavod Miroslav Krleža, H. e., 2011. *Hrvatska enciklopedija.* [Mrežno]
Dostupno na: http://www.enciklopedija.hr/Natuknica.aspx?ID=25317,
pristupano: 09.12.2017.

Manber, U., 1989.. *Introduction to algorithms, A creative approach.* Canada: Addison WEISLEY Publishing company INC.

Manger, R., 2014. *Strukture podataka i algoritmi.* 1st ur. Zagreb: ELEMENT d.o.o.

Miškovic, V., 2016/2017. *Analiza algoritama, pretraživanje i sortiranje u jeziku Python.* [Mrežno]
Dostupno na:
http://predmet.singidunum.ac.rs/pluginfile.php/13995/mod_folder/content/0/OP%2011%20Analiza%20algoritama%2C%20pretrazivanje%20i%20sortiranje.pdf?forcedownload=1,
pristupano: 08.12.2017.

Morris, J., 1998. *Data Structures and Algorithms, 4. Searching.* [Mrežno]
Dostupno na: https://www.cs.auckland.ac.nz/software/AlgAnim/searching.html,
pristupano: 08.12.2017.

Oljica, M., 2014. *Vizualizacija osnovnih algoritama za soritranje.* [Mrežno]
Dostupno na:
http://mapmf.pmfst.unist.hr/~ani/radovi/zavrsni/Oljica_Mate_zavrsni.pdf,
pristupano: 06.12.2017.

Pavlica, D., 2014. *Škola koda.* [Mrežno]
Dostupno na: https://skolakoda.org/strukture-podataka,
pristupano: 05.12.2017.

Pavlica, D., 2014. *Škola koda, Algoritmi pretrage.* [Mrežno]
Dostupno na: https://skolakoda.org/algoritmi-pretrazivanja,
pristupano: 07.12.2017.

Phyton-textbok, n.d. *Sorting and searching algorithms.* [Mrežno]
Dostupno na: http://python-textbok.readthedocs.io/en/1.0/Sorting_and_Searching_Algorithms.htm,
pristupano: 04.01.2018.

Podravec, n.d. *FER.* [Mrežno]
Dostupno na:
http://www.zemris.fer.hr/predmeti/is/zadaci/AkGod2001_2002/Podravec/index.html#sirinu,
pristupano: 04.01.2018.

Rothlauf, F., 2011. *Design of Modern Heuristics Principles and Application.* Berlin: Springer Science & Business Media.

Scheffler, K., 2003. *Algorithms and Complexity 2003, Analysis of InsertionSort and BubbleSort.* [Mrežno]
Dostupno na:
http://delab.csd.auth.gr/courses/c_ds/insertion_bubblesort_analysis.pdf,
pristupano: 06.12.2017.

Shaffer , C., 2008. [Mrežno]
Dostupno na: http://courses.cs.vt.edu/cs3114/Spring09/book.pdf,
pristupano: 08.12.2017.

Sinapova, L., 2004. *Sorting algorithms: Quick sort.* [Mrežno]
Dostupno na:
http://faculty.simpson.edu/lydia.sinapova/www/cmsc250/LN250_Tremblay/L06-QuickSort.htm,
pristupano: 07.12.2017.

Sinapova, L., 2004. *Sorting algorithms: Contents.* [Mrežno]
Dostupno na:
http://faculty.simpson.edu/lydia.sinapova/www/cmsc250/LN250_Tremblay/Contents.htm,
pristupano: 07.12.2017.

Sinapova, L., 2004. *Sorting algorithms: shell sort.* [Mrežno]
Dostupno na:
http://faculty.simpson.edu/lydia.sinapova/www/cmsc250/LN250_Weiss/L12-ShellSort.pdf,
pristupano: 06.12.2017.

Splithorizont, 2013. *METODE PRETRAŽIVANJA WEB-A.* [Mrežno]
Dostupno na: https://www.splithorizont.com/pdf/metode_pretrazivanja_weba.pdf,
pristupano: 09.12.2017.

Stojanović, B., 2016. *Strukture podataka i algoritmi 2.* [Mrežno]
Dostupno na: https://imi.pmf.kg.ac.rs/index-old.php?option=com_docman&task=doc_view&gid=16&Itemid=198,
pristupano: 07.12.2017.

Uninformed search, n.d. *Uninformed (also called blind) search algorithms.* [Mrežno]
Dostupno na: https://www.ics.uci.edu/~rickl/courses/cs-171/cs171-lecture-slides/cs-171-03-UninformedSearch.pdf,
pristupano: 08.12.2017.

Živković, D., 2010. *Uvod u algoritme i strukture podataka.* 1st ur. Beograd: Univerzitet Singidunum.

POPIS SLIKA

SAŽETAK

Usporedba algoritama za sortiranje i pretraživanje

Algoritmi danas imaju sve veću važnost u razvoju ljudskog potencijala. Algoritam nudi rješenje ne samo za računalni već i za društveni problem. Kako bi bolje razumjeli pojedini algoritam potrebno je prvenstveno ući u srž algoritma, njegovo značenje i princip rada, a još je važnije utvrditi njegovu vremensku složenost.

Ovaj završni rad obuhvaća algoritme za sortiranje i pretraživanje te prikazuje njihove vremenske složenosti prema kojima su izrađene međusobne usporedbe. Može se reći da smo i mi ljudi na neki način algoritmi. Za pojedini, naš vlastiti ili računalni problem, utvrđujemo korake rješavanja problema i upravo to je odlika dobrog algoritma. Na temelju veličine problema, podataka i traženog cilja kojeg želimo postići odabiremo algoritam za rješavanje našeg problema.

Ključne riječi: Algoritam, vremenska složenost, usporedba algoritama za sortiranje i pretraživanje

ABSTRACT

A Comparison of Sorting and Searching Algorithms

Algorithms today have a growing importance in the development of human potential. The algorithm provides a solution not only for the computer but also for the social problem. To better understand the individual algorithm it is necessary primarily to get into the core of the algorithm, its significance and principle of operation and it is even more important to determine its time complexity.

This bachelor's thesis includes algorithms for sorting and searching, and displays their time complexity according to which are made of mutual comparison. It can be said that we are people in some way algorithms. For some of our own, or computer problem, we determine steps to resolve the problem and that is the quality of a good algorithm. Based on the size of the problem, the data and the goal that we want to achieve we select the algorithm for solving our problems.

Key words: Algorithm, time complexity, A Comparison of Sorting and Searching Algorithms