

Auditorne vježbe



Pravila polaganja

Lv – 5 termina

- ulazni kolokvij (+5% na izlaznom u slučaju pada).
- izvještaj za svaku vježbu
- izlazni kolokvij (min 50% + padovi na ulaznim)

Av – 8 termina

- nije obavezno

Složenost

- Za provjeru kvalitete programa (algoritma) služi nam analiza složenosti.
- Svrha analize složenosti je optimizacija trenutnog algoritma kako bi napravili što bolji algoritam ako je to moguće.
- Vrste analiza:
 - analiza vremena izvođenja programa neovisno o programskom jeziku, prevoditelju, računalu...
 - stvarno testiranje programa, pokazuje stvarne podatke o vremenu koje je potrebno računalu da izvrši program. Ovisi o količini podataka koji se unose, o brzini procesora, kvaliteti kompjajlera i kvaliteti samog programa.
- Big O notacija koristi se u računarstvu kako bi opisala performanse ili kompleksnost algoritma. Big O posebno opisuje worst-case slučaj, i može se koristiti za opisivanje potrebnog vremena izvršavanja ili zauzeća prostora (memorije ili diska) od strane algoritma.
Ima apriornu složenost (ne uzima u obzir računalo na kojem se algoritam izvodi).

Složenost

Za O notaciju vrijedi:

O(1) - konstantni algoritmi - vrijeme izvođenja ovisi o konstanti. Označava konstantan broj operacija, neovisno o veličini niza ulaznih podataka. Opisuje algoritam koji će se uvijek izvoditi u jednakom vremenu ili prostoru bez obzira na veličinu ili ulazni skup podataka.

O(n) - linearni algoritmi – npr. petlja unutar for petlje. Složenost je proporcionalna broju ulaznih podataka kod algoritama u kojima se određena količina posla mora napraviti nad svakim elementom ulaznog niza. Opisuje algoritam čije će se performanse mijenjati linearno i proporcionalno veličini ulaznih podataka.

O(n^m) - stupanjski algoritmi - npr. ugniježđene petlje. Algoritam efikasan samo za umjerene vrijednosti broja n. Dvostruko veći niz ulaznih podataka zahtjeva četverostruko više vremena za rješavanje.

O(log n) - logaritamska složenost - kod svakog ponavljanja smanjuje se za red veličine baze. Kod algoritama koji problem mogu riješiti tako da ga transformiraju u jednostavniji smanjujući broj elemenata za fiksani omjer.

Složenost - vježba

Primjer 1:

```
void funkcija(int[] polje){  
    if(polje[0] == 1){  
        return true;  
    }  
    return false;  
}
```

Primjer 2:

```
void funkcija(int[] polje){  
    int n = sizeof(polje)/sizeof(polje[0]);  
    for(i=0;i<n;i++){  
        if(polje[i]%2==0) {  
            flag=1;  
            break;  
        }  
    }  
    if(flag){}  
}
```

Složenost - vježba

Primjer 5:

```
void funkcija(int[][] matrica){  
    for(int i=0;i<n;i++){  
        for(int j=0;j<m;j++){  
            //;  
        }  
    }  
    for(i=0;i<2*n;i++){ //}  
    for(int i=0;i<n*n;i++){  
        for(int j=0;j<n;j++){  
            //  
        }  
    }  
}
```

Složenost - vježba

Primjer 3:

```
void funkcija(int[][] matrica){  
    for(int i=0;i<n;i++){  
        for(int j=0;j<n;j++){  
            //;  
        }  
    }  
}
```

Primjer 4:

```
void funkcija(int[][] matrica){  
    int n=10;  
    int m=n*4;  
    for(i=0;i<n;i++){  
        for(j=0;j<m;j++){  
            //;  
        }  
    }  
}
```

Dijagrami toka – osnovni simboli

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectangle represents a process
	Decision	A diamond indicates a decision

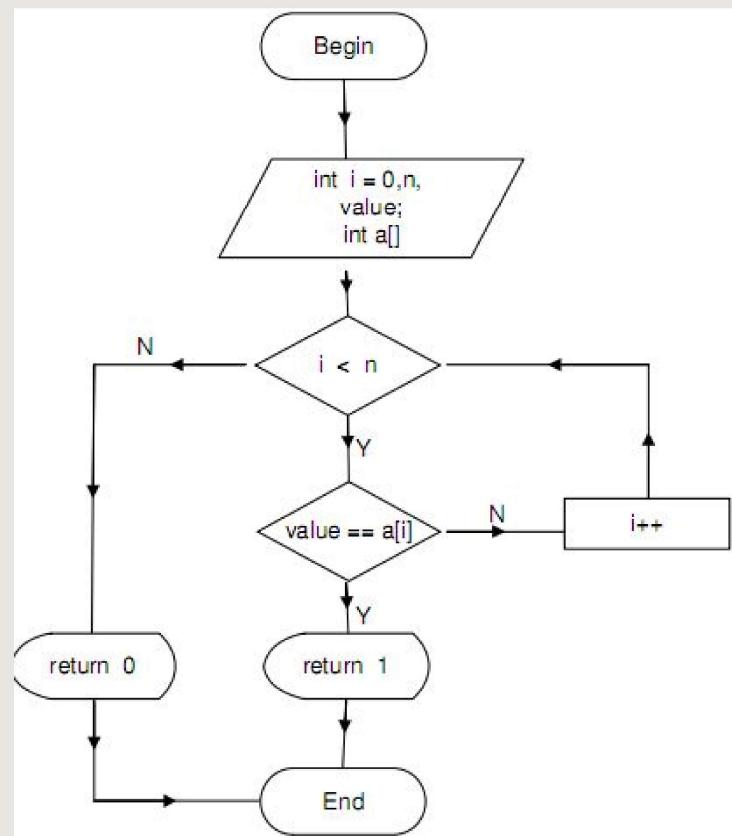
Pretraživanje - Sekvencijalno

```
int main(){
    int polje[10], trazeni, status, n;
    //broj elemenata
    scanf("%d", &n);

    for (status = 0; status < n; status++)
        scanf("%d", &array[c]);
    //trazenii broj
    scanf("%d", &trazeni);

    for (status = 0; status < n; status++){
        if (array[c] == search){
            //broj pronadjen
            break;
        }
    }
    if (status == n) //broj nije pronadjen
        return 0;
}
```

Pretraživanje – Sekvencijalno



Pretraživanje - Binarno

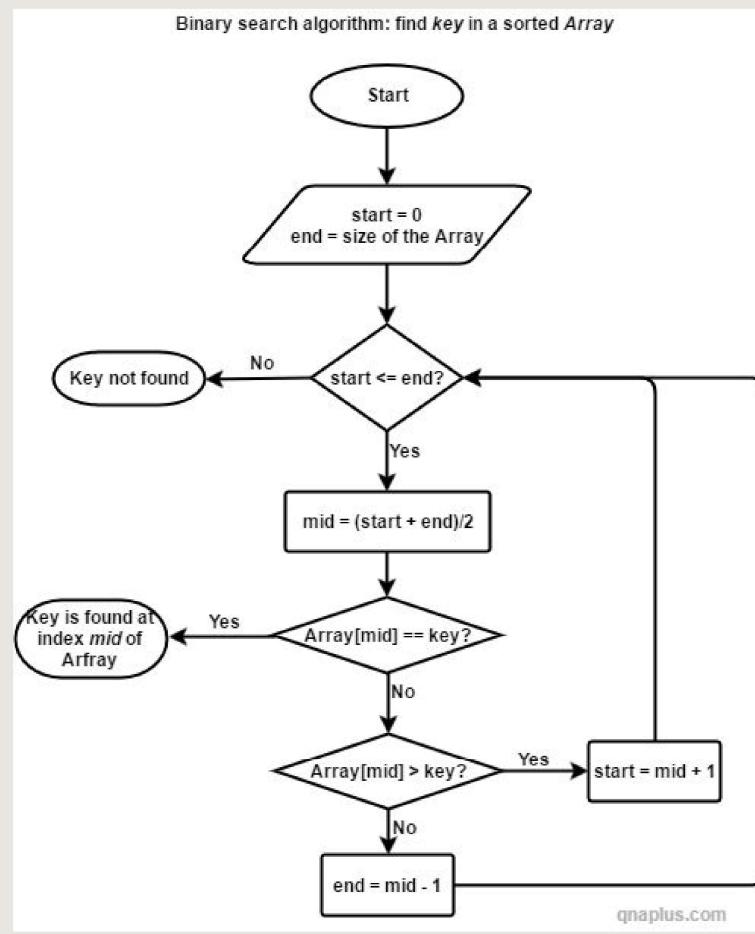
```
int main(){
    int polje[] = {2,8,18,22,100,110,222};
    int donjaG=0;
    int gornjaG=7-1;
    int sredina = 0;
    int trazeniBr = 300;

    while(donjaG<=gornjaG){
        sredina =(donjaG+gornjaG)/2;

        if(polje[sredina]>trazeniBr){
            gornjaG=sredina-1;
        }
        if(polje[sredina] < trazeniBr){
            donjaG = sredina + 1;
        }
        else{
            //broj pronadjen
            break;
        }
    }
    if(donjaG > gornjaG){ //broj nije pronadjen

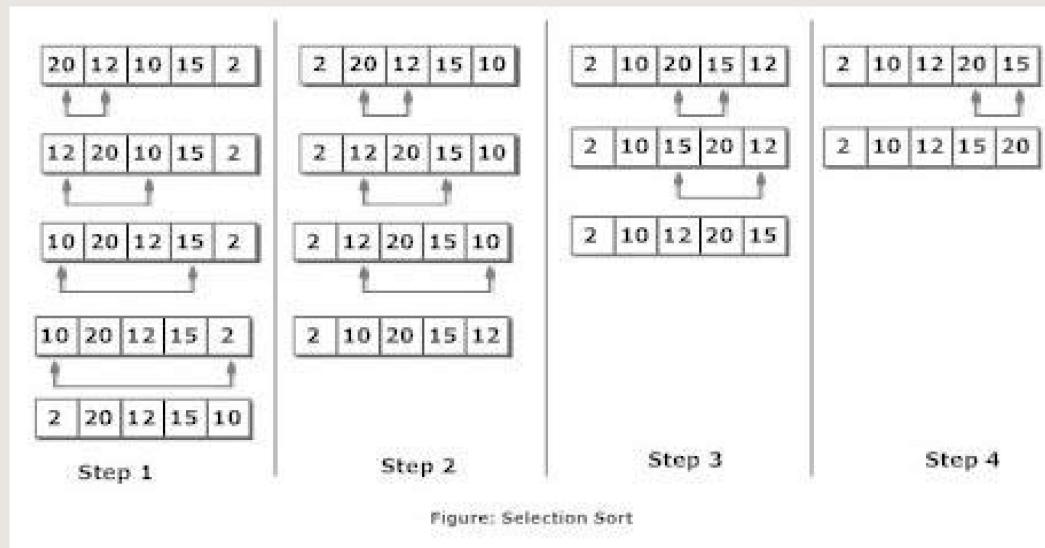
        return 0;
    }
```

Pretraživanje - Binarno



Sortiranje – Selection sort

- Algoritam „dijeli“ polje na dva dijela (sortiran i ne sortiran dio). Prilikom svake iteracije pronalazi se minimalni Element u nesortiranom dijelu i stavlja se u sortirani dio.



Sortiranje – Selection sort

```
int main(){
    ... //unos elemenata

    for (int i = 0; i < (n - 1); i++){
        index = i;

        for (int j = i + 1; j < n; j++){
            if (polje[index] > polje[j])
                index = j;
        }
        if (index != i){
            int tmp = polje[i];
            polje[i] = polje[index];
            polje[index] = tmp;
        }
    }
    return 0;
}
```

Sortiranje – Bubble sort

- Algoritam zasnovan na zamjeni susjednih elemenata opetovanim prolascima kroz niz koji se sortira, usporedbom susjednih članova i zamjene mjesta ako su pogrešno poredani.

Prvi prolaz:	19	33	5	45	13	26	10
	19	33	5	45	13	26	10
	19	5	33	45	13	26	10
	19	5	33	45	13	26	10
	19	5	33	13	45	26	10
	19	5	33	13	26	45	10
	19	5	33	13	26	10	45
Drugi prolaz:	19	5	33	13	26	10	45
	5	19	33	13	26	10	45
	5	19	33	13	26	10	45
	5	19	13	33	26	10	45
	5	19	13	26	33	10	45
	5	19	13	26	10	33	45
Treći prolaz:	5	19	13	26	10	33	45
	5	19	13	26	10	33	45
	5	13	19	26	10	33	45
	5	13	19	26	10	33	45
	5	13	19	10	26	33	45
Četvrti prolaz:	5	13	19	10	26	33	45
	5	13	19	10	26	33	45
	5	13	19	10	26	33	45
	5	13	10	19	26	33	45

Peti prolaz:	5	13	10	19	26	33	45
	5	13	10	19	26	33	45
	5	10	13	19	26	33	45
Šesti prolaz:	5	10	13	19	26	33	45
	5	10	13	19	26	33	45

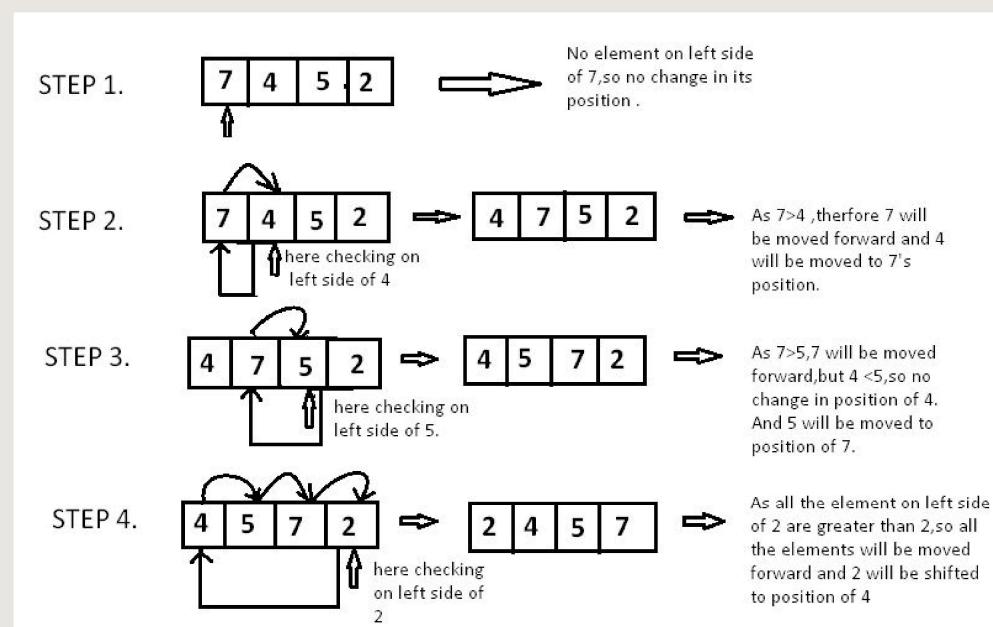
Sortiranje – Bubble sort

```
int main(){
    ...
    for(int i = 0 ; i < n - 1; i++){
        for(int j = 0 ; j < n-i-1; j++){
            if(polje[j] > polje[j+1]) {
                swap=polje[j];
                polje[j]=polje[j+1];
                polje[j+1]=swap;
            }
        }
    }
    return 0;
}
```

Insertion sort

Temelji se na ubacivanju elemenata (jedan po jedan) na odgovarajuće mjesto unutar sortiranog dijela polja.
Primjer:

1. Uzima se drugi element polja. Ako je manje od prvog napravi se zamjena mjesta.
2. Uzima se treći element polja. Ako je manji od drugog radi se zamjena. Nakon toga provjerava se da li je manji od prvog (ako je radi se zamjena).
3. ...



Insertion sort

```
void insertionSort(int polje[], int n) {  
    int i, tmp, j;  
    for (i = 1; i < n; i++) {  
        tmp = polje[i];  
        j = i - 1;  
        while (j >= 0 && polje[j] > tmp){  
            polje[j + 1] = polje[j];  
            j = j - 1;  
        }  
        polje[j + 1] = tmp;  
    }  
}
```

[12,11,13,5,6]
Tmp = polje[1];j=0
[12,12,13,5,6]
[11,12,13,5,6]
Tmp=polje[2];j=1
Tmp=polje[3];j=2
[11,12,13,13,6]
[11,12,12,13,6]
[11,11,12,13,6]
[5,11,12,13,6]
Tmp=polje[4];j=3
[5,11,12,13,13]
[5,11,12,12,13]
[5,11,11,12,13]
[5,6,11,12,13]

Zadaci

1. Zadana je cjelobrojna matrica A od M redaka i N stupaca. Vrijednosti matrice su popunjene nulama i jedinicama. Potrebno je dati algoritam koji će ispisati koliko u toj matrici ima podmatrica dimenzije 2×2 koje su u cijelosti popunjene jedinicama. Dopušteno je i preklapanje! Napisati koja je složenost vašeg algoritma.
2. Zadan je cjelobrojni niz V od N elemenata. Svi brojevi u nizu su pozitivni. Zadan je i pozitivan broj C. Potrebno je dati algoritam koji će ispitati da li postoje 2 elementa niza V čija suma iznosi točno C. Ako takvi elementi postoje, potrebno je ispisati na kojim mjestima se u nizu oni nalaze. Ako ne postoje takva 2 broja onda ispisati odgovarajuću poruku. Napisati također koja je složenost vašeg algoritma.
3. Zadan je niz $V = \{1, 1, 2, 3, 4, 4, 4, 4, 4, 5, 6, 7, 11, 11, 11, 14, 15, 16, 18, 19, 19, 19, 19, 19, 20, 21, 22, 22, 23, 26, 27, 29, 30, 32\}$. Opisati izvođenje algoritma BINARNO PRETRAŽIVANJE za podatke:

- a) $X = 31$
- b) $X = 30$

Napisati tablicu u kojoj će se pratiti vrijednosti sljedećih varijabli: sredina, dg, gg. Za svako ponavljanje petlje u algoritmu napisati jedan redak tablice. Iza tablice opisati na koji način je završio algoritam, te koliko ponavljanja je algoritam napravio. Napisati također koliko najviše ponavljanja ovaj algoritam radi i zašto.

R1

```
int main(){
    int matrica[4][4] = {{0,1,1,1},{0,1,1,1},{1,1,0,0},{0,0,1,1}};
    int m=4,n=4;
    int brojac=0;

    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++){
            if(i!=m-1 && j!=n-1){
                if(matrica[i][j]==1 && matrica[i][j+1]==1 && matrica[i+1][j]==1 && matrica[i+1][j+1]==1){
                    brojac++;
                }
            }
        }
    }
    printf("Ima ukupno %d",brojac);
}
```

R2

```
int main(){
    int polje[]={2,4,8,8,4,2,12};
    int c = 16;
    for(int i=0;i<7;i++){
        for(int j=0;j<7;j++){
            if(i!=j){
                if(polje[i]+polje[j]==c){
                    printf("%d (%d) + %d (%d) = %d\n",polje[i],i,polje[j],j,c);
                }
            }
        }
    }
}
```

R3

dg	gg	s
0	30	15
16	30	23
24	30	27
28	30	29
30	30	30

dg	gg	s
0	30	15
16	30	23
24	30	27
28	30	29

Zadaci

4. Za zadana dva vektora V1 i V2 provjeriti postojanje V1 u V2.
5. Za zadanu kvadratnu matricu A usporediti zbrojeve elemenata iznad i ispod sporedne dijagonale (ne računajući elemente na dijagonali).
6. Za zadanu matricu A ispisati sadržaj u spiralnom načinu.
7. Za zadanu matricu A ispisati dijagonalne elemente (pozitivan nagib)

Zadatak 4

```
void provjeri(int *p1,int n1,int *p2,int n2) {
    int flag=0;
    for(int i=0;i<n2;i++){
        if(*(p2+i)==*(p1)){
            flag=1;
            for(int j=i+1,k=1;j<n1,k<n1;j++,k++){
                if(*(p2+j)!=*(p1+k)){
                    flag=0;
                    break;
                }
            }
        }
        if(flag){
            printf("Postoji!");
            break;
        }
    }
}
```

Zadatak 5

```
void zbroj(int **matrica, int n) {  
    int sumalznad=0;  
    int sumalspod=0;  
  
    for(int i=0;i<n;i++){  
        for(int j=0;j<n-(i+1);j++){  
            printf("%d %d\n",i,j);  
            sumalznad+=matrica[i][j];  
        }  
    }  
    printf("\n\n");
```

```
    for(int i=1;i<n;i++){  
        for(int j=n-i;j<n;j++){  
            printf("%d %d\n",i,j);  
            sumalspod+=matrica[i][j];  
        }  
    }  
    printf("Suma iznad %d, suma ispod %d  
\n",sumalznad, sumalspod);  
}
```

Zadatak 6

```
void ispisSpiralno(int** A, int n){  
    int gore = 0, dolje = n-1, lijevo = 0, desno = n-1;  
    while(1){  
        if(lijevo>desno) break;  
        //ispis gornjeg reda  
        for (int i=lijevo; i<=desno; i++) printf("%d ",A[gore][i]);  
        gore++;  
  
        if(gore>dolje) break;  
        //ispis desne kolone  
        for(int i=gore; i<=dolje; i++) printf("%d ",A[i][desno]);  
        desno--;
```

```
        if(lijevo>desno) break;  
        //ispis donjeg reda  
        for (int i = desno; i >= lijevo; i--) printf("%d ",A[dolje][i]);  
        dolje--;  
  
        if(gore>dolje) break;  
        //ispis lijeve kolone  
        for (int i = dolje; i >= gore; i--) printf("%d ",A[i][lijevo]);  
        lijevo++;  
    }  
}
```

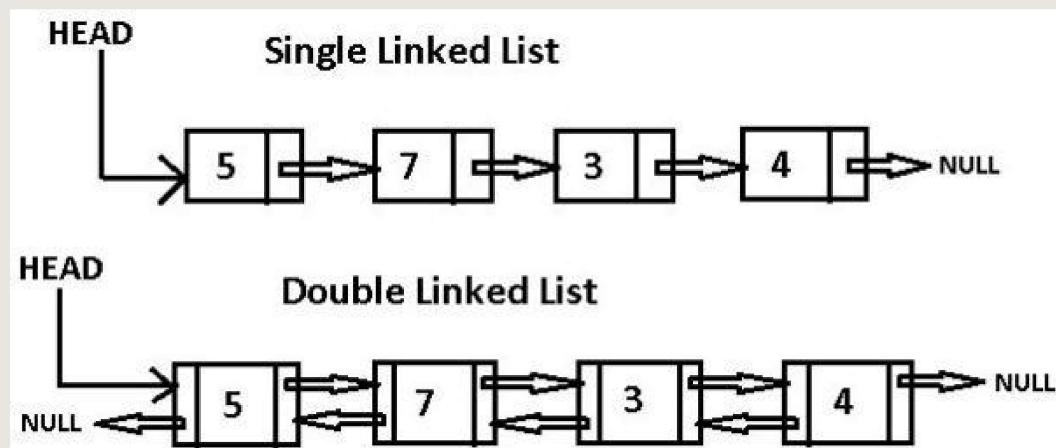
Zadatak 7

```
...
for(int i=0;i<n;i++){
    for(int j=i,k=0;j>=0,k<=i;j--,k++){
        printf("%d%d ",j,k);
    }
    printf("\n");
}
for(int i=1;i<n;i++){
    for(int j=n-1,k=i;j>0,k<n;j--,k++){
        printf("%d%d ",j,k);
    }
    printf("\n");
}
...

```

Povezane liste

- Povezani popis – dinamička struktura.
- Elementi se stavljaju u slobodni dio memorije, a kada pojedini zapis ili cijeli popis više nije potreban, zauzeti dio memorije se oslobađa. – Elementi su međusobno povezani adresnim podacima (pokazivačima).



Povezane liste

Osnovni element povezanog popisa sastoji se iz 2 dijela:

- x - dio s podacima (bilo kakvi elementarni tipovi).
- sljedeci - memorijska adresa koja pokazuje idući osnovni element u memoriji.

```
struct element{  
    int broj;  
    element * next;  
}
```

Povezane liste - stvaranje

```
element * stvori_listu(){
    int k,n;
    element *p,*glava;

    cout<<"Koliko zelite elemenata liste?";
    cin>>n;
    cout<<"\n\n";

    for(k=0;k<n;k++){
        if(k==0){
            //glava=(element*)malloc(sizeof(element));
            glava = new element;
            p=glava;
        }
        else{
            p->next=(element*)malloc(sizeof(element));
            p=p->next;
        }
        cout<<"\n Unesi p->broj: ";
        cin>>p->broj;
    }
    p->next=NULL;
    return(glava);
}
```

Povezane liste – ispis / aritmeticka

```
void ispis(element *glava){  
  
    Element *p = glava;  
    while(p!=NULL){  
        cout<<p->broj<<endl;  
        p=p->next;  
    }  
}
```

```
float aritmeticka(element*pok){  
    Element * pokazivac;  
    int brojac=0,suma=0;  
    float aritmeticka=0;  
  
    pokazivac=pok;  
  
    while(pokazivac!=NULL){  
        brojac++;  
        suma+=pokazivac->broj;  
        pokazivac=pokazivac->next;  
    }  
    aritmeticka=suma/brojac;  
    return aritmeticka;  
}
```

Povezane liste – dodavanje na kraj

```
void dodaj_na_kraj(element *glava){  
    element *temp1;  
    temp1 = glava;  
    while(temp1->next!=NULL){  
        temp1 = temp1->next;  
    }  
    element *temp;  
    temp = new element;  
    cout<<"Unesi broj: ";  
    cin>>temp->broj;  
    temp->next=NULL;  
    temp1->next = temp;  
}
```

Povezane liste – brisanje zadnjeg

```
void obrisi_kraj(element *glava){
    //kreiramo privremeni cvor
    element *temp1;
    temp1 = new element;
    temp1 = glava;
    element *old_temp;
    old_temp = new element;
    while(temp1->next!=NULL){

        old_temp = temp1;
        temp1 = temp1->next;
    }
    old_temp->next = NULL;
    free(temp1);
}
```

Povezane liste - ubacivanje

```
void ubaci(element * glava, int x, int index){  
    element* novi = new element;  
    element* tmp = glava;  
    int brojac=1;  
    novi->X=x;  
    if(index==1){  
        novi->next = glava->next;  
        glava->next = novi;  
    }else{  
        while(tmp->next!=NULL){  
            if(brojac==index){  
                novi->next=tmp->next;  
                tmp->next=novi;  
                break;  
            }else{  
                tmp = tmp->next;  
                brojac++;  
            }  
        }  
    }  
}
```

Zadaci

8. Neka su zadana dva povezana popisa P1 i P2 koji se sastoje od osnovnog elementa koji ima podatak X i pokazivača na sljedeći takav element NEXT. Također postoji i pokazivač FIRST na prvi element povezanog popisa. Svi elementi liste su zadani, zadnji element u listi ima vrijednost sljedećeg (NEXT) jednaku NULL.

Dati algoritam koji će izračunati i ispisati:

- a) Koliko ima elemenata povezanog popisa.
- b) Koja je srednja vrijednost povezanog popisa.
- c) Koliko ima negativnih elemenata u povezanom popisu.
- d) Koliko ima pozitivnih elemenata većih od srednje vrijednosti u povezanom popisu.
- e) Svi elementi pozitivni elementi koji su veći od srednje vrijednosti u P1 dodati na kraj P2. Ukoliko nema takvih elemenata dodati element s vrijednosti X = 0 na početak liste P2.

Dodatno zagrijavanje: Neka je zadan povezani popis kao u 1. zadatku. Skicirajte sve elemente povezanog popisa nakon sljedećeg koda:

```
PRVI = NULL
Za svaki i=1 do 6
    t = novi Osn. Element
    t.X = i*2+i
    t.NEXT = PRVI
    PRVI = t
```

Zadaci

9. Neka su zadana dva jednostruko povezana popisa s više podataka. Potrebno je napisati implementaciju funkcije u C jeziku koja spaja ta dva povezana popisa na način da na kraj prvog doda drugi, s tim da početak dodavanja drugog mora početi parnim brojem.

Popis 1: 2, 4, 6, 8

Popis 2: 3, 7, 2, 10, 12

Rezultat: 2, 4, 6, 8, 2, 10, 12

10. Neka je zadan jednostruko povezani popis s više podataka. Potrebno je napisati implementaciju funkcije u C jeziku koja provjerava jesu li podaci u tom povezanom popisu poredani od najmanjeg do najvećeg (sortirani) ili nisu. Koja je složenost algoritma?

Zadaci

11. Zadana je uzlazno poredana povezana lista. Napisati funkciju koja briše sve elemente duplike te ispisuje koliko lista na kraju ima elemenata.
12. Napisati funkciju koja će iz poredane liste izbaciti sve elemente koji imaju vrijednost manju ili jednaku prosječnoj vrijednosti svih prethodnih elemenata.
13. Zadana je povezana lista s N elemenata. Dati algoritam koji će pronaći polovicu zadane povezane liste i promijeniti taj povezani popis na način da druga polovica postane prva polovica. Ukoliko povezana lista ima neparni broj elemenata, element koji je višak treba pridružiti drugoj polovici.

Ulaz: A→B→C→D→E

Izlaz: C→D→E→A→B

Zadatak 11

```
void duplikati(element *glava) {  
    if(!sortiranost(glava)){  
        return;  
    }  
    element *p = glava;  
    element *tmp = NULL;  
    while (p->next) {  
        if (p->X == p->next->X) {  
            tmp = p->next->next;  
            free(p->next);  
            p->next = tmp;  
        }else {  
            p = p->next;  
        }  
    }  
}
```

Zadatak 12

```
element* brisi(element* glava) {  
    element* p=glava;  
    element* pret=p;  
    int sum=0,br=0;  
    sum+=p->x;  
    br++;  
    p=p->next;
```

```
    while(p!=NULL) {  
        br++;  
        sum+=p->x;  
        if(p->x <= ((double)sum;br)) {  
            pret->next=p->next;  
            free(p);  
            p=pret->next;  
        } else {  
            pret=p;  
            p=p->next;  
        }  
    }  
    return glava;  
}
```

Zadatak 13

```
element* izmjeni(element* glava) {
    element* p=glava;
    //racunam duljinu i sredinu
    int br=0;
    while(p->next!=NULL) {
        br++;
        p=p->next;
    }
    br+=1;
    p->next=glava;
    int sredina = br/2;
```

```
int stat=0;
while(p!=NULL){
    if(stat==sredina){
        glava=p->next;
        p->next=NULL;
        break;
    }
    stat++;
    p=p->next;
}
return glava;
```

Dvostruko povezane liste

```
struct cvor{  
    int x;  
    struct cvor * prev;  
    struct cvor *next;  
};
```

Dvostruko PL – dodavanje na pocetak

```
struct cvor* dodaj_na_pocetak(struct cvor* glava, int X){  
    if(glava!=NULL){  
        struct cvor* tmp = (struct cvor*)malloc(sizeof(struct cvor));  
        tmp->x = X;  
        tmp->prev = NULL;  
        tmp->next = glava;  
        return tmp;  
    }  
}
```

Dvostruko PL – dodavanje na kraj

```
void dodaj_na_kraj(struct cvor* glava, int X){  
    if(glava!=NULL){  
        struct cvor* tmp1 = glava;  
        while(tmp1->next!=NULL){  
            tmp1 = tmp1->next;  
        }  
        struct cvor* tmp2 = (struct cvor*)malloc(sizeof(struct cvor));  
        tmp2->x = X;  
        tmp2->prev = tmp1;  
        tmp2->next = NULL;  
        tmp1->next=tmp2;  
    }  
}
```

Dvostruko PL – dodaj nakon

```
void dodajNakon(struct cvor* glava, int X, int index){  
    if(glava!=NULL){  
        struct cvor* tmp = glava;  
        int brojac=1;  
        if(index>1){  
            while(tmp->next!=NULL){  
                if(brojac==index){  
                    struct cvor* novi = (struct cvor*)malloc(sizeof(struct cvor));  
                    novi->x=X;  
                    novi->next=tmp->next;  
                    novi->next->prev=novi;  
                    novi->prev = tmp;  
                    tmp->next=novi;  
                    break;  
                }else{  
                    tmp = tmp->next;  
                    brojac++;  
                }  
            }  
        }  
    }  
}
```

Dvostruko PL – ispis

```
void ispis(struct cvor* glava){
    struct cvor* p = glava;
    if(p!=NULL){
        printf("\n Ispis \n");
        while(p!=NULL){
            printf("%d ",p->x);
            p=p->next;
        }
    }
}
int main(){

    int polje[] = {2,1,8,12,4};
    struct cvor * glava = NULL;//(struct cvor*)malloc(sizeof(struct cvor));
    glava = stvori_listu(glava,polje);
    ispis(glava);
    glava = dodaj_na_pocetak(glava,3);
    ispis(glava);
    dodaj_na_kraj(glava,100);
    ispis(glava);
    dodajNakon(glava,222,1);
    ispis(glava);
}
```



Auditorne vježbe

Rekurzije

Rekurzije

- Funkcije koje pozivaju same sebe.
- Način implementacije različitih algoritama.
- Potrebno je definirati uvjet zaustavljanja.
- Rekursivne funkcije treba pisati na način da je svakim novim pozivom posao koji obavlja sve bliže kraju. Na taj način, rekursivna funkcija će sigurno završiti posao u konačno mnogo koraka.
- Da bi napisali rekursivnu funkciju potrebno je vidjeti da li se naš posao može izvršavati rekursivno.

Rekurzije

- Izračun faktorijela

$$n! = n * (n-1) * (n-2) * \dots * 3 * 2 * 1$$

Rekurvna funkcija za izračun faktorijela

```
int fact(int n){  
    if (n == 1) return 1;  
    return n * fact(n-1);  
}
```

Rekurzije

Simulacija rekurzivne funkcije za izračun faktorijela

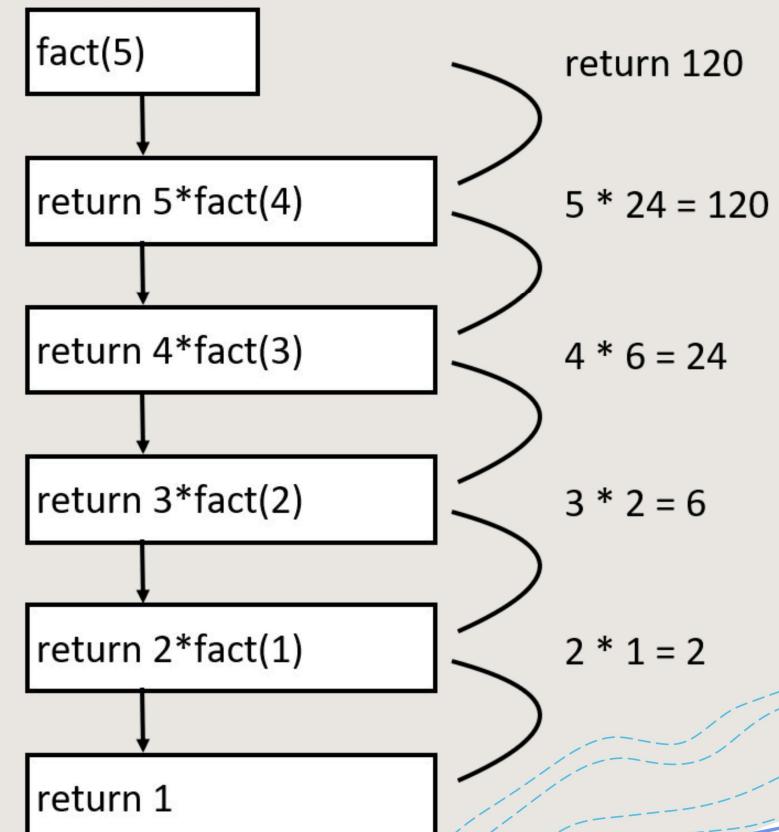
```
#include<iostream>
using namespace std;
int funkcija1(int);
int funkcija2(int);
int funkcija3(int);
int funkcija4(int);
int funkcija5(int);

int main(){
    cout<<funkcija1(5);
    return 0;
}
int funkcija1(int broj){
    return broj*funkcija2(broj-1);
}
int funkcija2(int broj){
    return broj*funkcija3(broj-1);
}
int funkcija3(int broj){
    return broj*funkcija4(broj-1);
}
int funkcija4(int broj){
    return broj*funkcija5(broj-1);
}
int funkcija5(int broj){
    if(broj==1) return 1;
    return 0;
}
```

Rekurzije

Rekurzivna funkcija za izračun faktorijela

```
int fact(int n){  
    if (n == 1) return 1;  
    return n * fact(n-1);  
}
```



Rekurzije

$$fib(0) = 1$$

$$fib(1) = 1$$

$$fib(n) = fib(n-1) + fib(n-2)$$

1, 1, 2, 3, 5, 8, 13, ...

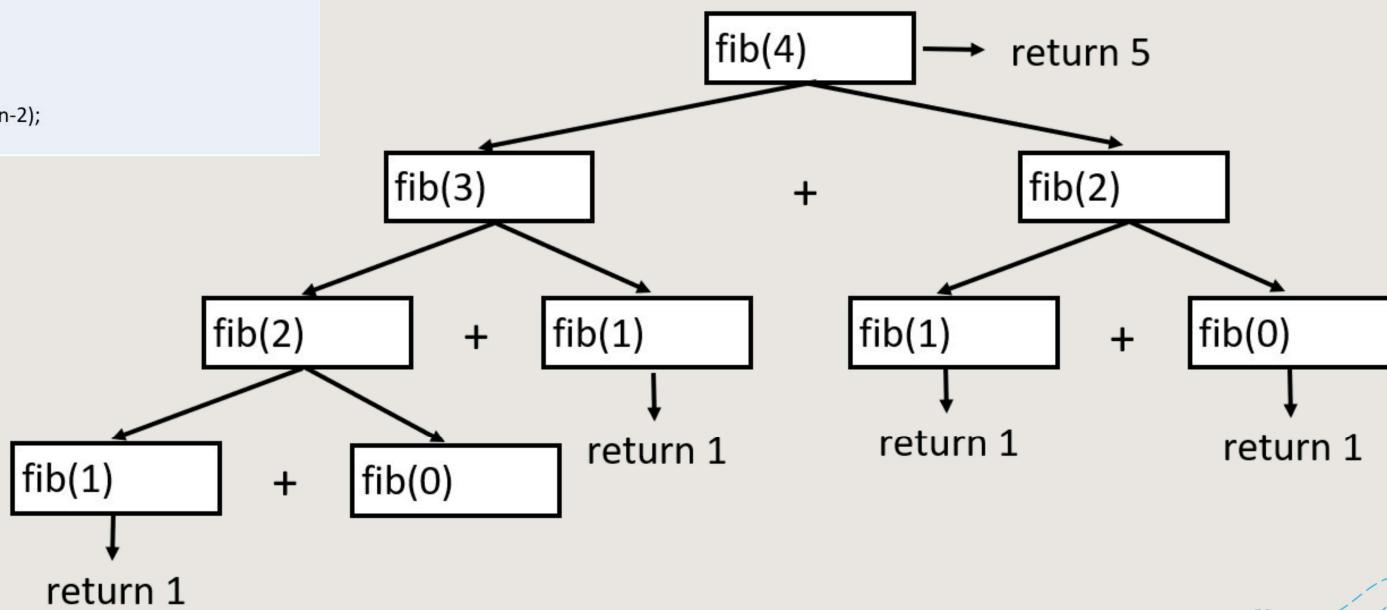
Rekurzivna funkcija za izračun fibonaciјevog broja

```
int fibonacci(int n){  
    if(n==0){  
        return 1;  
    }  
    if(n==1){  
        return 1;  
    }  
    return fibonacci(n-1)+fibonacci(n-2);  
}
```

Rekurzije

Rekursivna funkcija za izračun fibonaciјevog broja

```
int fibonacci(int n){  
    if(n==0){  
        return 1;  
    }  
    if(n==1){  
        return 1;  
    }  
    return fibonacci(n-1)+fibonacci(n-2);  
}
```



Zadaci

14. Napisati rekurzivnu funkciju koja određuje duljinu povezane liste.
15. Napisati rekurzivnu funkciju koja za rezultat vraća zbroj znamenki N znamenkastog broja.
16. Napisati rekurzivnu funkciju koja će izračunati sumu elemenata polja (opcionalno: na parnim i neparnim mjestima u polju).
17. Zadana je povezana lista s N elemenata. Napisati rekurzivnu funkciju koja će obrnuti redoslijed liste. (Domaća zadaća)
18. Napisati rekurzivne funkcije za ispis sljedećih oblika:

* ****
** ***
*** **
**** *
***** **
***** *

**
*

**
*

Zadatak 14

```
/*Napisati rekurzivnu funkciju koja određuje duljinu povezane liste.*/
int pobroji(element * glava){
    if(glava==NULL){
        return 0;
    }
    return 1+pobroji(glava->next);
}
```

Zadatak 15

```
/*Napisati rekurzivnu funkciju koja za rezultat vraća zbroj znamenki N znamenkastog broja.*/
int zbroji(int n){
    if(n==0){
        return 0;
    }
    return (n%10)+funk(n/10);
}
```

Zadatak 16

/*Napisati rekurzivnu funkciju koja će izračunati sumu elemenata polja (opcionalno: na parnim i neparnim mjestima u polju).*/

```
int suma(int polje[], int n){  
    if(n<=0){  
        return 0;  
    }  
    //return polje[0]+suma(polje+2,n-2);  
    return polje[1]+suma(polje+2,n-2);  
}
```

Zadatak 17

Zadana je povezana lista s N elemenata. Napisati rekurzivnu funkciju koja će obrnuti redoslijed liste.
(Domaća zadaća)

Zadatak 18

//Napisati rekurzivne funkcije za ispis sljedećih oblika:

```
void iscrtaj(int n){  
    if (n==1){  
        printf("*");  
        printf("\n");  
    }else if(n>1){  
        iscrtaj(n-1);  
        for (int i=0;i<n;i++) printf("*");  
        printf("\n");  
    }  
}
```

*
**

Zadatak 18

//Napisati rekurzivne funkcije za ispis sljedećih oblika:

```
void ispis(int n){  
    if (n==1){  
        printf("*");  
        printf("\n");  
    }else if(n>1){  
        for (int i=0;i<n;i++) printf("*");  
        printf("\n");  
        ispis(n-1);  
    }  
}
```

**

*

Zadatak 18

//Napisati rekurzivne funkcije za ispis sljedećih oblika:

```
void funkcija(int X){  
    if (X == 0) return;  
    else{  
        for (int i = 1; i <= X; i++)printf("*");  
        printf("\n");  
        funkcija(X - 1);  
        for (int i = 1; i <= X; i++)printf("*");  
        printf("\n");  
    }  
}
```


**
*
*
**

Zadatak 18

//Napisati rekurzivne funkcije za ispis sljedećih oblika:

```
void iscrtajTrokut(int visina) {  
    static int z=1;  
    if (visina == 0) return;  
    // Ispisi razmak  
    for (int i = 0; i < visina; i++) printf(" ");  
    // Ispisi zvjezdice  
    for (int i = 0; i < 2 * z - 1; i++) printf("*");  
    z++;  
    printf("\n");  
    iscrtajTrokut(visina - 1);  
}
```

*

Zadaci

19. Napisati rekurzivnu funkciju koja će provjeriti je li zadani niz znakova palindrom.
20. Napisati rekurzivnu funkciju koja će pretvoriti decimalni u binarni broj.
21. Napisati rekurzivnu funkciju koja će za zadani cijeli broj N izbaciti sve parne znamenke.

Zadatak 19

//Napisati rekurzivnu funkciju koja će provjeriti je li zadani niz znakova palindrom.

```
int jePalindrom(char* niz, int pocetak, int kraj){  
    if (pocetak >= kraj) return 1;  
    if (niz[pocetak] == niz[kraj])  
        return jePalindrom(niz, pocetak + 1, kraj - 1);  
    return 0;  
}
```

Zadatak 20

//Napisati rekurzivnu funkciju koja će pretvoriti decimalni u binarni broj.

```
int bin(int n){  
    if (n == 0)  
        return 0;  
    else  
        return (n % 2 + 10 * bin(n / 2));  
}
```

Zadatak 21

//Napisati rekurzivnu funkciju koja će za zadani cijeli broj N izbaciti sve parne znamenke.

```
long samoNeparni(long n){  
    long tmp;  
    int zn;  
    if (n<10){  
        tmp=0;  
        if (n % 2==1) tmp=n;  
        return tmp;  
    }  
    zn=n % 10;  
    tmp=samoNeparni(n/10);  
    if(zn % 2==1) tmp=tmp*10+zn;  
    return tmp;  
}
```

Zadaci

22. Napisati rekurzivnu funkciju koja će pomoći Popaju da pronađe Olivu. Pozicije Popaja i Olive kao i mogući putevi prikazani su matricom:

```
char polje[5][10] = {{"#", ., ., ., ., ., ., ., "#", .},  
                      {"#, ., ., "#", "#", "#", "#", ., "#", "O"},  
                      {"#" "P", ., ., ., "#", ., ., "#", .},  
                      {"#" "#", ., ., ., "#", ., ., "#", .},  
                      {"#" "#", "#", "#", "#", ., ., ., ., .}};
```

Zadatak 22

//Napisati rekurzivnu funkciju koja će za zadani cijeli broj N izbaciti sve parne znamenke.

```
void pronadji(int i, int j){  
    if(!granica(i,j) || posjeceni[i][j]==1){  
        return;  
    }  
    posjeceni[i][j]=1;  
    if(polje[i][j]=='O'){  
        printf("Pronadjen!\n");  
        return;  
    }else if(polje[i][j]=='. || polje[i][j]=='P'){  
        pronadji(i,j+1);  
        pronadji(i-1,j);  
        pronadji(i+1,j);  
        pronadji(i,j-1);  
    }  
}
```

```
int granica(int i, int j){  
    if(i<5 && j<10 && j>=0 && i >=0){  
        return 1;  
    }  
    return 0;  
}
```

Zadatak 22

```
void pronadji(int,int);
int granica(int,int);
char polje[5][10] = {{'#','.',',','.',',','.',',','.',',','.'},
                      {'#','.',',','.',',','.',',','.',',','O'},
                      {'#','P','.',',','.',',','.',',','.'},
                      {'#','.',',','.',',','.',',','.',',','.'},
                      {'#','.',',','.',',','.',',','.',',','.'}};
int posjeceni[5][10];
int main(){
    for(int i=0;i<5;i++){
        for(int j=0;j<10;j++){
            posjeceni[i][j]=0;
        }
    }
    pronadji(2,1);
    return 0;
}
```

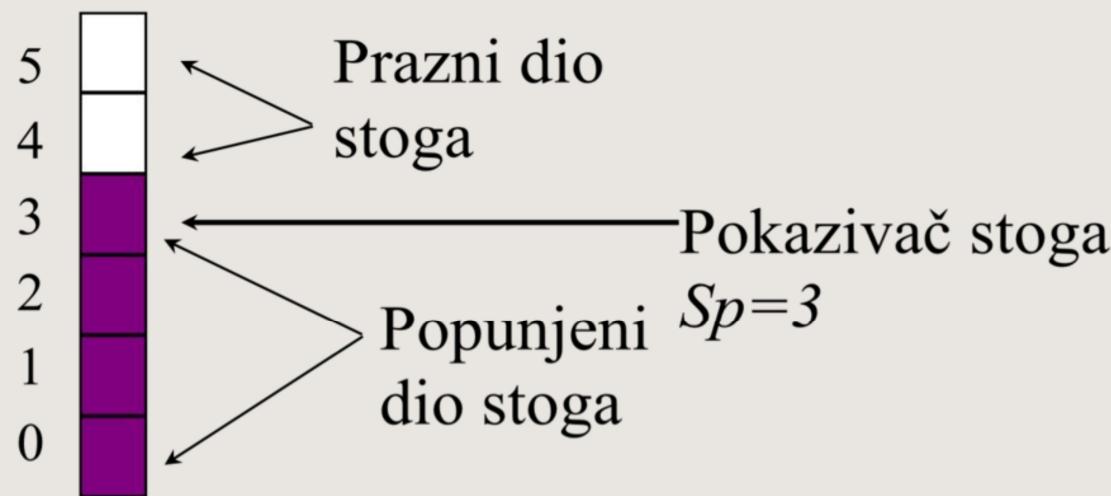


Auditorne vježbe

Stogovi/Redovi

Stog

- Linearna struktura.
- Pokazivač stoga ima vrijednost indeksa zadnjeg popunjeno elementa.



Stog

```
void push(int x){  
    if(sp==MAX_SIZE-1){  
        printf("Stack full\n");  
        return;  
    }  
    sp+=1;  
    stack[sp]=x;  
}
```

```
int pop(){  
    if(sp==-1){  
        printf("Stack empty\n");  
        return -1;  
    }  
    int tmp = stack[sp];  
    sp-=1;  
    return tmp;  
}
```

Stog/druga implementacija

```
typedef struct stog stog;

void push(float,stog* );
float pop(stog* );
int is_empty(stog* );

struct stog{
    int sp;
    int max;
    float *p;
    void (*Push)(float x,stog* s);
    float (*Pop)(stog* s);
    int (*IsEmpty)(stog* s);
};
```

Stog/druga implementacija

```
stog* init(){
    stog * s = (stog*)malloc(sizeof(stog));
    s->max=10;
    s->sp=-1;
    s->p=(float*) malloc (10*sizeof(float));
    s->Push=push;
    s->Pop=pop;
    s->IsEmpty=is_empty;
    return s;
}
```

Stog/druga implementacija

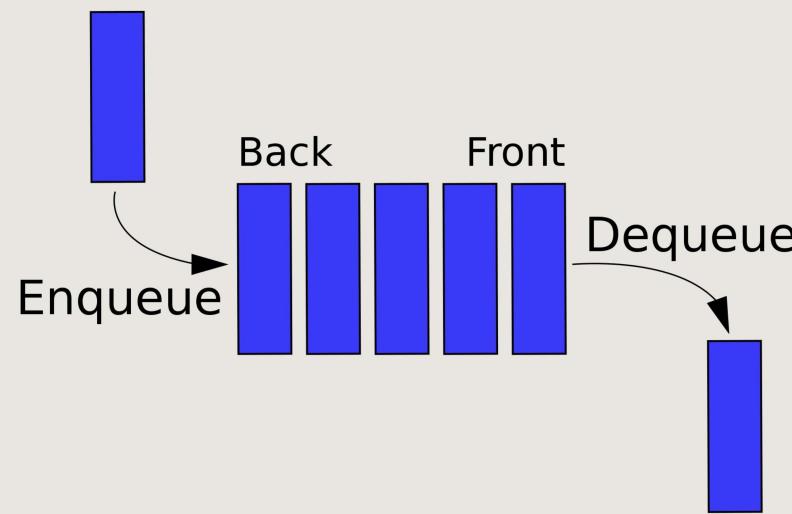
```
void push(float x, stog* s){  
    if(s->sp==s->max){  
        s->max+=10;  
        s->p=realloc(s->p,s->max*sizeof(float));  
    }  
    s->sp=s->sp+1;  
    s->p[s->sp]=x;  
}  
float pop(stog* s){  
    if(s->sp==-1){  
        printf("Stog je prazan!\n");  
        return -1;  
    }else{  
        float rez = s->p[s->sp];  
        s->sp=s->sp-1;  
        return rez;  
    }  
}
```

Stog/druga implementacija

```
int is_empty(stog* s){  
    if(s->sp == -1){  
        return 0;  
    }else{  
        return 1;  
    }  
}
```

Red

- Linearna struktura.
- FIFO struktura podataka



Red

```
void enqueue(int element) {  
    if (rear == MAX_SIZE - 1) {  
        printf("Queue is full");  
        return;  
    }  
    if (front == -1) {  
        front = 0;  
    }  
    rear++;  
    queue[rear] = element;  
}  
  
int dequeue() {  
    if (front == -1 || front > rear) {  
        printf("Queue is empty");  
        return -1;  
    }  
    int element = queue[front];  
    front++;  
    return element;  
}
```

Zadaci

23. Što će se nalaziti u redu nakon izvršavanja sljedećeg programskog odsječka (na početku je red prazan):

```
push(5);
for(j=1;j<=6;j++)
    push(j*pop()+j);
```

24. Što će se nalaziti u redu nakon izvršavanja sljedećeg programskog odsječka (na početku je red prazan):

```
int x = 100;
for(;;){
    push( x );
    if (x%2==0) pop();
    x = x/2;
    if (x==0) break;
}
```

Zadaci

25. Neka je zadan sljedeći algoritamski zapis koji koristi stog S:

```
S.clear()  
S.push(5)  
Za svaki i=1 do 4  
    t = S.pop()  
    Za svaki j=1 do t-1  
        S.push(j)
```

Analizirati ovaj algoritam i odgovoriti koliko će se elemenata na kraju nalaziti na stogu i također zapisati sve elemente stoga.

Zadaci

26. Napisati funkciju koristeći stogove koja će pretvoriti decimalni broj u binarni.
27. Napisati funkciju koristeći stogove koja će provjeriti je li zadani niz znakova palindrom.

Zadatak 26

```
void decUbin(int n) {  
    stog* s1 = init();  
    int i = 0;  
    while (n > 0) {  
        s1->Push(n % 2, s1);  
        n = n / 2;  
        i++;  
    }  
    while(s1->isEmpty(s1)){  
        printf("%d\n",s1->Pop(s1));  
    }  
}
```

Zadatak 27

```
int isPalindrome(char* string){  
    stog* s1 = init();  
    char c;  
    for (int i=0; i<strlen(string); i++) {  
        if (i < (strlen(string)/2)) {  
            s1->Push(string[i],s1);  
        }  
        else if (i == (strlen(string)/2) && strlen(string) % 2 == 1) {  
            // ništa  
        }  
        else {  
            c=s1->Pop(s1);  
            if (c != string[i])  
                return 0;  
        }  
    }  
    return 1;  
}
```

Zadaci

28. Napisati implementaciju stog push i pop funkcija koristeći povezane liste.
29. Potrebno je napisati funkciju "funk" čiji je prototip:
$$\text{funk(stog* s, double d);}$$

Funkcija funk iz stoga s treba ukloniti sve elemente koji su od aritmetičke sredine elemenata stoga s udaljeni barem za d . Uklonjeni elementi se stavljaju na stog s2 (redoslijed nije bitan). Na stogu s, poredak elemenata koji nisu uklonjeni treba ostati isti. (Dozvoljeno je korištenje pomoćnih stogova).

Primjer: Stog s (dno) 4-5-6-4-1 (vrh). Aritmetička sredina elemanta na stogu je 4. Ako je funkcija pozvana za $d=2$, na izlazu iz funkcije stog s treba biti 4-5-4, a s2 1-6. Ako je funkcija pozvana za $d=4$, na izlazu iz funkcije stog s se ne mijenja, a stog s2 je prazan.

30. Napisati algoritam u C/C++ jeziku koji za zadani znakovni niz provjerava ispravnost postavljenih zagrada. Npr. `char polje[] = { '(', '[', ']', ')' };`. Napomena: koristiti stogove.

Zadatak 28

```
typedef struct element element;
#define MAX 10
int sp=-1;
struct element {
    int x;
    element* next;
}*l;
```

```
void push(int x) {
    if(sp==MAX) {
        printf("Stack Overflow\n");
    }else {
        if(sp==-1) {
            l=(element*)malloc(sizeof(element));
            l->x=x;
            l->next=NULL;
            sp++;
        }else {
            element* n = (element*)malloc(sizeof(element));
            n->x=x;
            n->next=l;
            l=n;
            sp++;
        }
    }
}
```

Zadatak 28

```
int pop() {
    if(sp== -1) {
        printf("Stack Underflow\n");
        return -1;
    }else {
        int rez = l->x;
        element*tmp = l->next;
        free(l);
        l=tmp;
        sp--;
        return rez;
    }
}
```

Zadatak 29

```
void funk(stog* s1, double d) {  
    stog* s2 = init();  
    stog* s3 = init();  
    double suma=0;  
    int brojac=0;  
    while(s1->IsEmpty(s1)){  
        float p = s1->Pop(s1);  
        s2->Push(p,s2);  
        suma+=p;  
        brojac++;  
    }  
    double srednja = suma/brojac;  
    while(s2->IsEmpty(s2)){  
        float p = s2->Pop(s2);  
        if(p>(srednja-d) && p<(srednja+d)) {  
            s1->Push(p,s1);  
        }else {  
            s3->Push(p,s3);  
        }  
    }  
}
```

```
while(s3->IsEmpty(s3)){  
    s2->Push(s3->Pop(s3),s2);  
}  
  
printf("\n\n Ispis s1\n");  
while(s1->IsEmpty(s1)){  
    printf("%.2f\n",s1->Pop(s1));  
}  
printf("\n\n Ispis s2\n");  
while(s2->IsEmpty(s2)==1){  
    printf("%.2f\n",s2->Pop(s2));  
}  
}// gotova funkcija funk
```

Zadatak 30

```
char polje[] = { '(', ')', '[', ']', '{', '}' };
int f=0;
for(int i=0;i<strlen(polje);i++){
    if(polje[i]== '(' || polje[i]== '[' || polje[i]== '{'){
        push(polje[i]);
    }else{
        if(polje[i]== ')'){
            char z = pop();
            if(z!= '('){
                printf("Neispravno\n");
                f=1;
                break;
            }
        }
    }
}
```

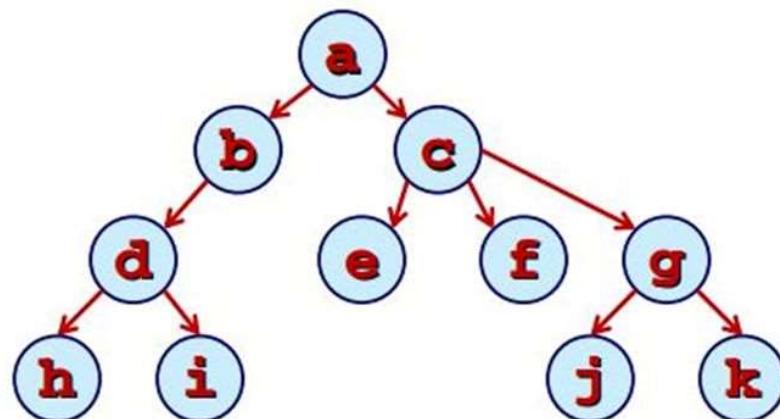
```
if(polje[i]== ']'){
    char z = pop();
    if(z!= '['){
        printf("Neispravno\n");
        f=1;
        break;
    }
}
if(polje[i]== '}'){
    char z = pop();
    if(z!= '{'){
        printf("Neispravno\n");
        f=1;
        break;
    }
}
}
if(f==0){ printf("Ispravno");}
```

Auditorne vježbe

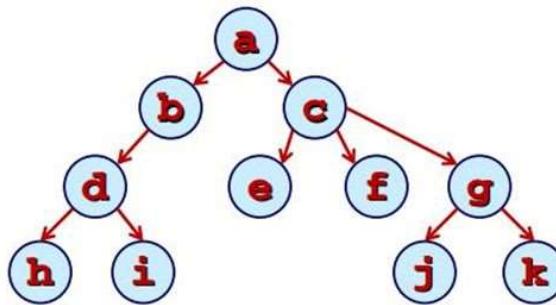
Stabla

Stabla – svojstva

- Stablo predstavlja konačan skup čvorova (elemenata) gdje postoji jedan “glavni” čvor koji se naziva još i korijen stabla.
- Hijerarhijska struktura stabla sastoji se od čvorova koji se označavaju točkama ili kružićima. Čvorovi su spojeni dužinama koje se nazivaju grane stabla.



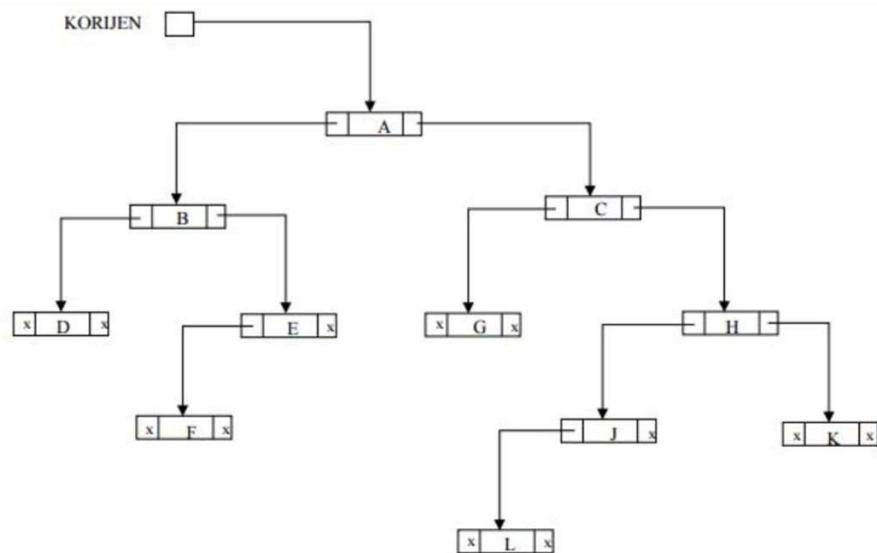
Stabla – svojstva



Parametri stabla:

- Stupanj čvora predstavlja broj podstabala nekog čvora. (npr. stupanj čvora A je 2).
- Stupanj stabla jednak je maksimalnom stupnju svih čvorova tog stabla (npr. stupanj stabla prikazanog na slici je 3).
- Razina čvora - za računanje razine čvora polazi se od toga da je korijen na razini jedan, a da su razine djece nekog čvora razine n jednaki n+1.
- Dubina stabla jednaka je maksimalnoj razini nekog čvora u stablu.

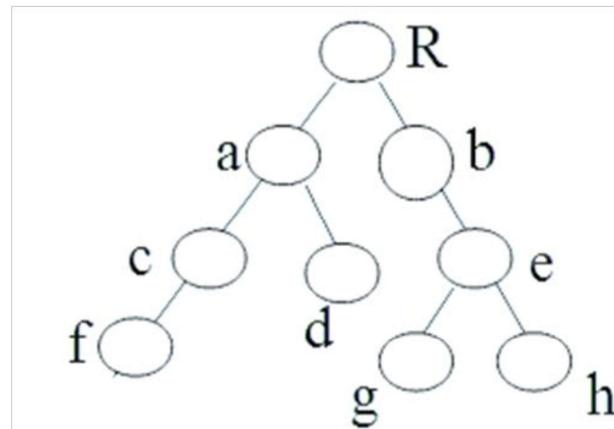
Stabla - načini prikaza



	X	LIJEVI	DESNI
1	A	7	2
2	C	3	5
3	G	0	0
4	K	0	0
5	H	10	4
6	L	0	0
7	B	11	9
8	F	0	0
9	E	8	0
10	J	6	0
11	D	0	0

Stabla - obilasci

- Kod binarnog stabla definirani su obilasci po dubini. Takav obilazak prati put od korjena stabla "u dubinu" do najniže razine.



- Preorder (NLD - korjen, lijevi, desni): R, a, c, f, d, b, e, g, h
- Inorder (LND - lijevi, korjen, desni): f, c, a, d, R, b, g, e, h
- Postorder (LDN - lijevi, desni, korjen): f, c, d, a, g, h, e, b, R

Stabla - tipovi

Prošireno binarno stablo

- čvor može imati ili dva ili nijedan nasljednik.

Potpuno binarno stablo

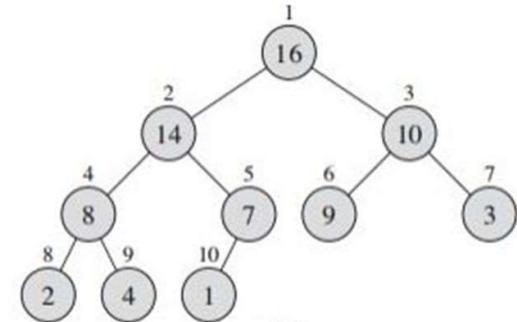
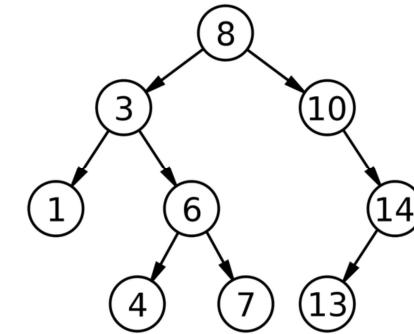
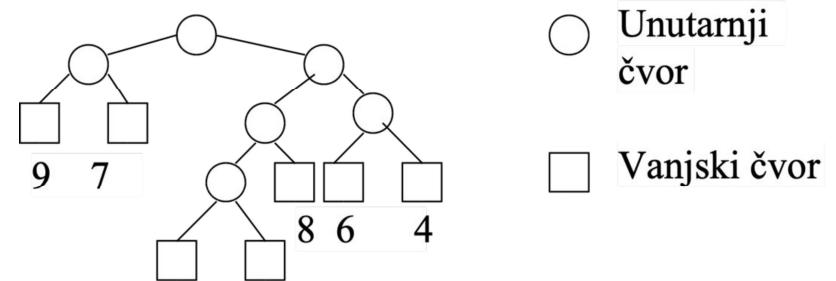
- sve razine popunjene (osim eventualno zadnje).
Popunjava se s lijeva na desno.

Poredano binarno stablo

- čvorovi označeni međusobno usporedivim oznakama
(lijevi manja, a desni veća).

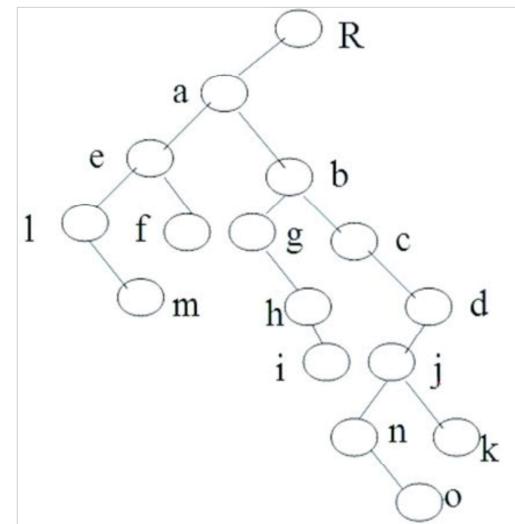
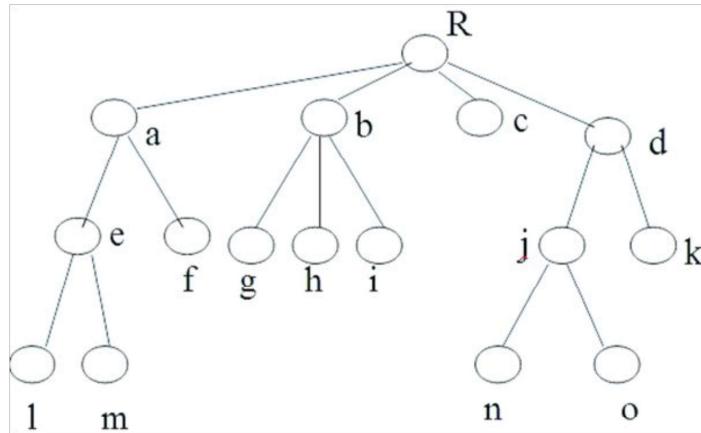
Hrpa

- potpuno binarno stablo koje zadovoljava dodatan uvjet
da je nadređeni čvor veći ili jednak od oba podređena
čvora.



Stabla - binarno stablo

- Binarno stablo predstavlja stablo kod kojeg svaki čvor može imati najviše dva nasljednika (lijevi i desni).
- Bilo koje opće stablo može se prikazati binarnim stablom tako da za svaki čvor prvi nasljednik postane lijevi, a prvi susjed postane desni nasljednik u binarnom stablu.

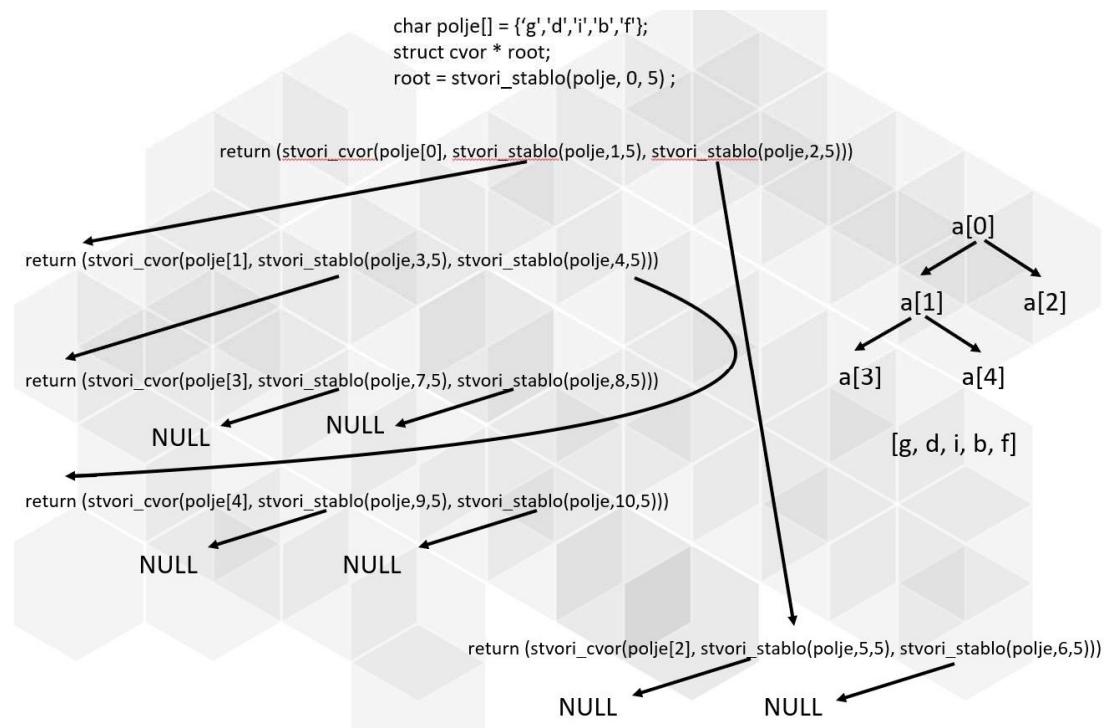


Stabla - binarno stablo

```
struct cvor{
    char d;
    struct cvor *left;
    struct cvor *right;
};

struct cvor * stvori_cvor(char dd, struct cvor * lijevi, struct cvor * desni){
    struct cvor * t;
    //new cvor
    t = (cvor*)malloc(sizeof(cvor))
    t->d = dd;
    t->left = lijevi;
    t->right = desni;
    return t;
}

struct cvor * stvori_stablo(char polje[], int i, int velicina){
    if (i >= velicina)
        return NULL;
    else
        return(stvori_cvor(polje[i], stvori_stablo(polje, 2*i+1, velicina),
stvori_stablo(polje, 2*i+2, velicina)));
}
```



Stabla - poredano binarno stablo

```
struct cvor{  
    int x;  
    struct cvor *left,*right;  
};  
void ubaci(struct cvor *r,struct cvor *p){  
    if((r->right==NULL)&&(p->x > r->x)){  
        r->right=p;  
    }  
    else if((r->right!=NULL)&&(p->x > r->x)){  
        ubaci(r->right,p);  
    }  
    if((r->left==NULL)&&(p->x < r->x)){  
        r->left=p;  
    }  
    else if((r->left!=NULL)&&(p->x < r->x)){  
        ubaci(r->left,p);  
    }  
}
```

Stabla - poredano binarno stablo

```
int pretrazi(struct cvor *glava, int broj)
{
    cvor *p = glava;
    while (glava != NULL)
    {
        if (broj > glava->x){
            glava = glava->right;
        }
        else if (broj < glava->x){
            glava = glava->left;
        }
        else{
            return 1;
        }
    }
    return 0;
}
```

Min - Max ??

Stabla – poredano binarno stablo

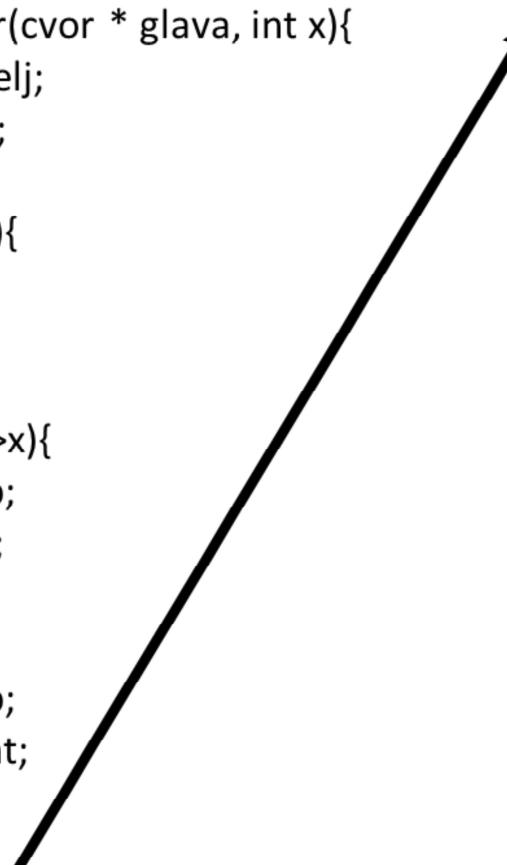
Brisanje elemenata poredanog binarnog stabla

Postoji tri moguća slučaja s kojima se možemo susresti prilikom brisanja elementa poredanog binarnog stabla:

- Krajnji čvor
- Jedan nasljednik (lijevi ili desni)
- Dva nasljednika (lijevi i desni)

Stabla – poredano binarno stablo

```
cvor * obrisi_cvor(cvor * glava, int x){  
    cvor * p,*roditelj;  
    roditelj = NULL;  
    p = glava;  
    while(p!=NULL){  
        if(x == p->x){  
            break;  
        }  
        else if(x < p->x){  
            roditelj = p;  
            p = p->left;  
        }  
        else{  
            roditelj = p;  
            p = p->right;  
        }  
    }  
    //u slucaju da broj nije pronadjen  
    if(p==NULL){  
        return glava;  
    }  
    if((p->lijevi == NULL) && (p->right == NULL)){  
        if(roditelj != NULL){  
            if(roditelj->left->x == x){  
                roditelj->left = NULL;  
            }else{  
                roditelj->right = NULL;  
            }  
            free(p);  
        }  
    }  
}
```



Stabla – poredano binarno stablo

ako je ($p->left=NULL$) **AND** ($p->right!=NULL$) **onda**

ako je $roditelj!=NULL$ **onda**

ako je $roditelj->left!=NULL$ **onda**

ako je $roditelj->left->x=x$ **onda**

$roditelj->left=p->right$

ako je $roditelj->desno!=NULL$ **onda**

ako je $roditelj->right->x=x$ **onda**

$roditelj->right=p->right$

 oslobodi(p)

ako je ($p->left!=NULL$) **AND** ($p->right=NULL$) **onda**

ako je $roditelj!=NULL$ **onda**

ako je $roditelj->left!=NULL$ **onda**

ako je $roditelj->left->x=x$ **onda**

$roditelj->left=p->left$

ako je $roditelj->right!=NULL$ **onda**

ako je $roditelj->right->x=x$ **onda**

$roditelj->right=p->left$

 oslobodi(p)

Stabla - poredano binarno stablo

ako je (p->left!=NULL) **AND** (p->right!=NULL) **onda**

ako je roditelj!=NULL **onda**

ako je roditelj->left!=NULL **onda**

ako je roditelj->left->x=x **onda**

r=nadji_max(p->left)

novi=alokacija_cvor()

novi->x=r

ako je p->left->x=r **onda**

novi->left=NULL

inače

novi->left=p->left

novi->right=p->right

roditelj->left=novi

ako je roditelj->right!=NULL **onda**

ako je roditelj->right->x=x **onda**

r=nadji_max(p->left)

novi=alokacija_cvor()

novi->x=r

ako je p->left->x=r **onda**

novi->left=NULL

inače

novi->left=p->left

novi->right=p->right

roditelj->right=novi

vrati r

Ulez: r-čvor stabla

Algoritam funkcije nadji_max(r)

p,rod //pokazivaci na strukturu

var //podatak

p=r

dok je p->right!=NULL **činiti**

rod=p

p=p->right

ako je p->x=rod->x **onda**

var=p->x

inače

var=p->x

rod->right=NULL

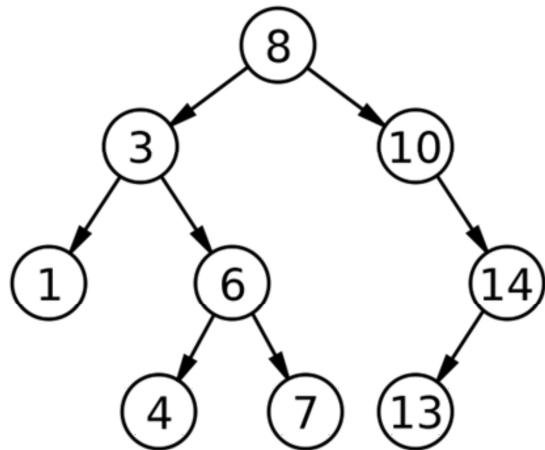
vrati var

Stabla - obilasci

```
void preOrder (struct cvor* root){  
    if(root==NULL) return;  
    else{  
        printf(root->x);  
        preOrder( root->left);  
        preOrder( root->right);  
    }  
}
```

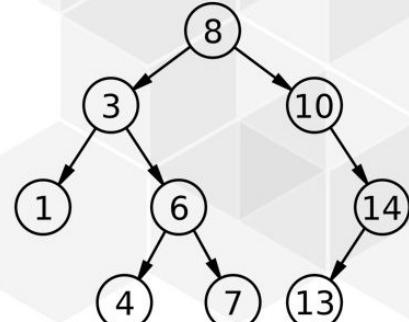
```
void inOrder (struct cvor* root){  
    if(root==NULL) return;  
    else{  
        inOrder(root->left);  
        printf(root->x);  
        inOrder(root->right);  
    }  
}
```

```
void postOrder (struct cvor* root){  
    if(root==NULL) return;  
    else{  
        postOrder(root->left);  
        postOrder(root->right);  
        printf(root->x);  
    }  
}
```

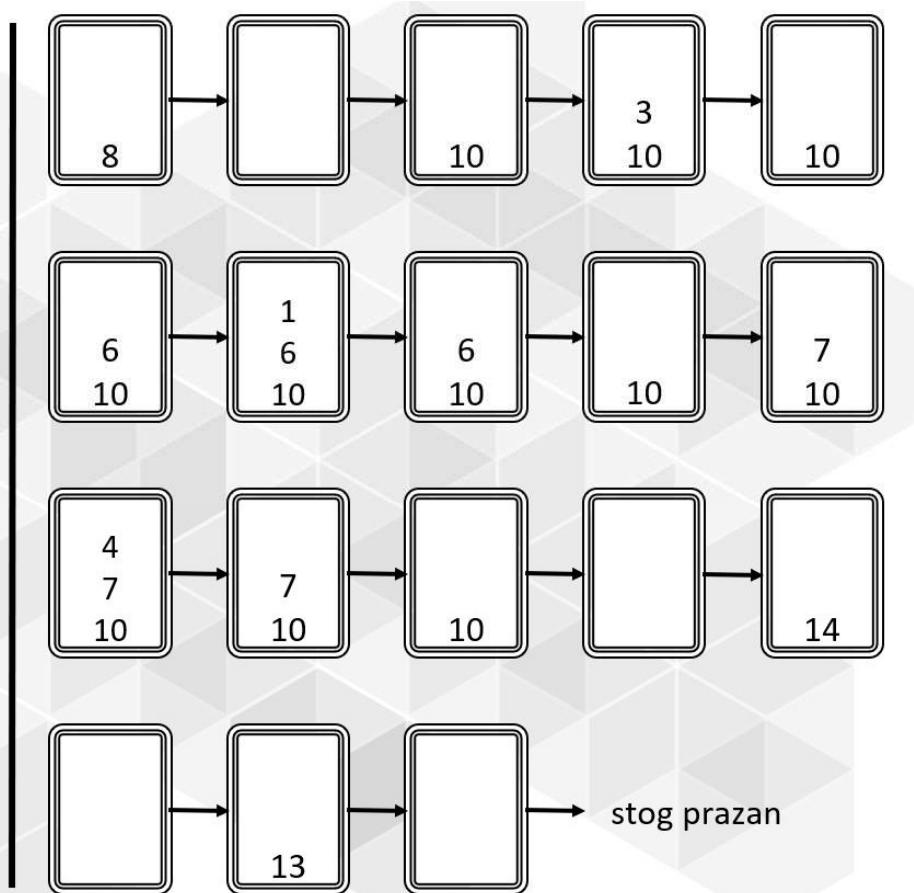


Stabla – obilasci

```
void preorder(cvor * glava){
    cvor * p = glava;
    cvor * p_temp = glava;
    push(p);
    while(!emp()){
        p_temp = pop();
        printf("%d\n",p_temp->x);
        if(p_temp->right){
            push(p_temp->right);
        }
        if(p_temp->left){
            push(p_temp->left);
        }
    }
}
```

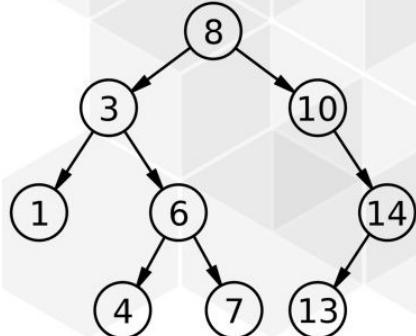


Izlaz: 8, 3, 1, 6, 4, 7, 10, 14, 13

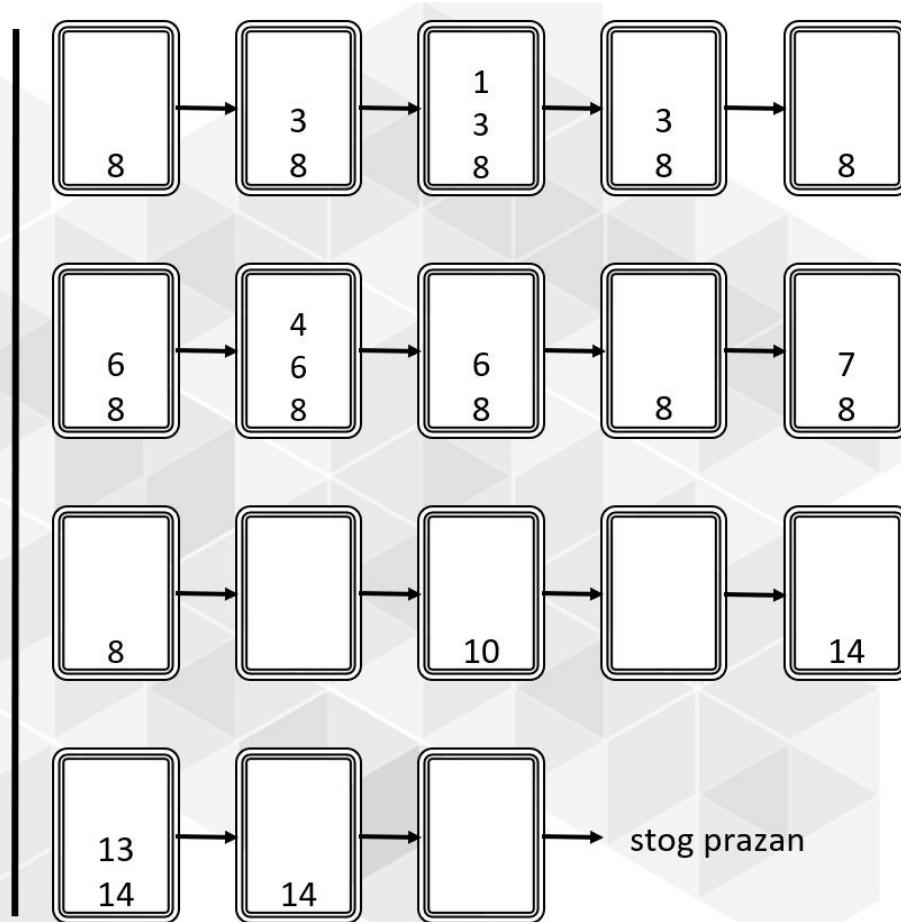


Stabla - obilasci

```
void inorder(cvor * glava){  
    cvor * p = glava;  
    while(true){  
        while(p!=NULL){  
            push(p);  
            p=p->left;  
        }  
        if(emp())  
            return;  
        p=pop();  
        printf("%d\n",p->x);  
        p = p->right;  
    }  
}
```



Izlaz: 1, 3, 4, 6, 7, 8, 10, 13, 14



Zadaci

31. Nacrtati binarno stablo za koje su poznati sljedeći obilasci:

Preorder: 1,2,4,6,5,7,8,3

Inorder: 6,4,2,7,5,8,1,3

32. Napisati rekurzivnu funkciju koja će zrcalno obrnuti binarno stablo. Korijen stabla se ne mijenja dok lijevo i desno podstablo mijenjaju mesta na svakoj razini.
33. Napisati rekurzivnu funkciju koja će od dva stabla napraviti novo u kojem elementi stabla čine zbroj elemenata prvog stabla i elemenata drugog stabla. Ako jedno od ulaznih stabala nema neki čvor koji ima drugo stablo zbroj čini samo čvor iz postojećeg stabla (nepostojeci čvor ima vrijednost 0 kod zbrajanja).
34. Napisati rekurzivnu funkciju koja čvorovima stabla koji imaju samo jedno dijete dodaje drugo dijete koje je list, a vrijednost mu je jednaka zbroju vrijednosti svih čvorova na putu od tog djeteta do korijena (uključujući i vrijednost korijena).
35. Napisati rekurzivnu funkciju koja će u svakom čvoru stabla zamijeniti vrijednost s 0 ako je zbroj svih potomaka tog čvora paran, odnosno, s 1 ako je zbroj svih potomaka neparan.

Zadatak 32

```
...
struct cvor *root = stvori_stablo(stablo,0,5);
okreni(root);
```

```
...
```

```
void okreni( cvor *korijen ) {
    cvor *temp;
    if( korijen ) {
        okreni( korijen -> left );
        okreni( korijen -> right );
        temp = korijen -> left;
        korijen -> left = korijen -> right;
        korijen -> right = temp;
    }
}
```

Zadatak 33

```
...
int stablo1[] = {5,10,18,100,2};
int stablo2[] = {1,1,18,100,200,300,500,4};
struct cvor * root1= stvori_stablo(stablo1, 0, 5) ;
struct cvor * root2= stvori_stablo(stablo2, 0, 8) ;
struct cvor* novi = zbroji(root1,root2);

...
cvor *zbroji(cvor *s1, cvor *s2) {
    if (s1 == NULL && s2 == NULL) return NULL;
    cvor* c = (cvor*)malloc(sizeof(cvor));
    c->d = ((s1)?s1->d:0)+ ((s2)?s2->d:0);
    c->left = zbroji((s1)?s1->left:NULL, (s2)?s2->left:NULL);
    c->right = zbroji((s1)?s1->right:NULL, (s2)?s2->right:NULL);
    return c;
}
```

Zadatak 34

```
...
int stablo1[] = {1,1,18,100,200,300,500,4};
struct cvor * root1= stvori_stablo(stablo1, 0, 8) ;
nadopuni(root1,0);
...
void nadopuni(cvor *korijen, int tmp){
    cvor *novi;
    if (!korijen) return;
    if (!korijen->left && !korijen->right) return;

    tmp +=korijen->d;
    if (korijen->left && korijen->right){
        nadopuni(korijen->left,tmp);
        nadopuni(korijen->right,tmp);
    }
}
else{
    novi=(cvor *) malloc(sizeof(cvor));
    novi->left=novi->right=NULL;
    novi->d=tmp;
    if (!korijen->left){
        nadopuni(korijen->right,tmp);
        korijen->left=novi;
    }
    else{
        nadopuni(korijen->left,tmp);
        korijen->right=novi;
    }
}
```

Zadatak 35

...

```
int stablo1[] = {1,1,18,100,200,300,500,4};  
struct cvor * root1= stvori_stablo(stablo1, 0, 8) ;  
f(root1);
```

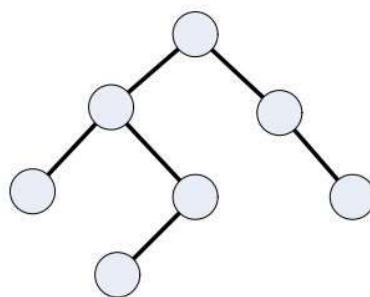
...

```
int f(cvor *root){  
    int suma;  
    int trenutnaVrijednost;  
    if(root==NULL){  
        return 0;  
    }  
    suma=f(root->left)+f(root->right);  
    trenutnaVrijednost = root->d;  
    root->d=suma%2;  
    return suma+trenutnaVrijednost;  
}
```

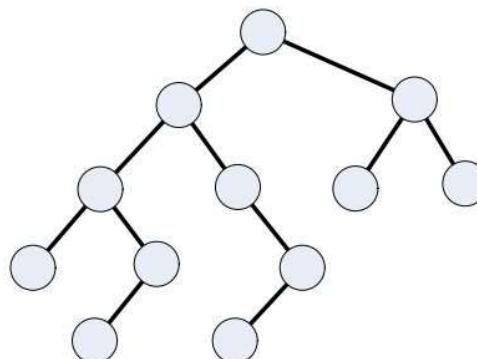
Zadatak - ponavljanje stabla

Potrebno je provjeriti je li binarno stablo balansirano s obzirom na visinu. Prazno stablo je balansirano, a neprazno binarno stablo je balansirano ako vrijedi:

- Lijevo i desno podstablo je balansirano
- Razlika u visini lijevog i desnog podstabla nije više od 1



Stablo balansirano po visini



Stablo koje nije balansirano po visini

Napisati funkciju koja će izračunati visinu stabla. Ako je stablo prazno vraća 0.

Napisati funkciju koja će provjeriti da li je stablo balansirano. Ako je balansirano vraća 1, inače 0. Dopušteno je koristiti funkciju visina.

```
int visina(cvor * korijen) {  
    int visinaLijevo, visinaDesno;  
    if (korijen == NULL)  
        return 0;  
    visinaLijevo = visina(korijen-> lijevo_dijete);  
    visinaDesno = visina(korijen-> desno_dijete);  
    if (visinaLijevo >= visinaDesno) return 1 + visinaLijevo;  
    else return 1 + visinaDesno;  
}
```

```
int balans(cvor *korijen){  
    int lijevo, desno;  
    if(korijen == NULL) return 1;  
    lijevo = visina(korijen-> lijevo_dijete);  
    desno = visina(korijen-> desno_dijete);  
    return abs(lijevo - desno) <= 1 && balans(korijen->lijevo_dijete) && balans(korijen-> desno_dijete);  
}
```

Zadatak - ponavljanje stabla

Čvor stabla definiran je odsječkom:

```
struct cv{  
    int vrijednost;  
    struct cv *lijevo, *desno;  
};
```

Zadani su prototipovi dviju funkcija:

```
int sirinaRazine(cvor* korjen, int razina);  
int sirinaStabla(cvor* korijen);
```

Funkcija sirinaRazine treba vratiti broj čvorova na razini razina. Funkcija sirinaStabla treba vratiti najveću širinu svih razina u binarnom stablu, dakle broj čvorova na najširoj razini.

Zadatak - ponavljanje stabla

```
int sirinaRazine(cvor* korijen, int razina){  
    if(korijen==NULL || razina<1) return 0;  
    if(razina==1) return 1;  
  
    return sirinaRazine(korijen->lijevo, razina-1)+sirinaRazine(korijen->desno, razina-1);  
}
```

sirinaStabla → Domaća zadaća



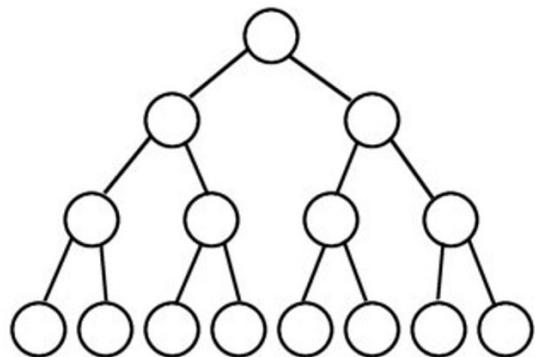
Auditorne vježbe

Hrpe

Ponavljanje

Stablo u kojem su sve razine popunjene, a jedino posljednja razina ne mora biti popunjena zove se POTPUNO BINARNO STABLO.

Svaka razina popunjava se s lijeva na desno!



Maksimalni broj čvorova stabla dubine k jednak je $2^k - 1$ za $k > 0$.

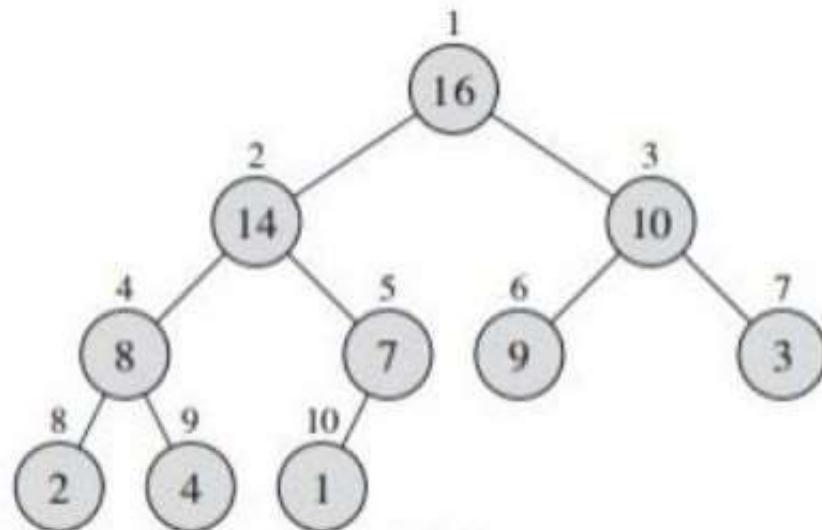
Maksimalni broj čvorova na k-toj razini jednak je 2^{k-1}

Visina za određeni N $\rightarrow \log_2(N+1)$

Razina čvora $\rightarrow \log_2(\text{index}+1)$

Hrpe

- Hrpa predstavlja potpuno binarno stablo.
- Postoje dvije vrste hrpa (min-hrpa, max-hrpa).
- Kod max-hrpe najveći element nalazi se u korjenu te svaki čvor ima oznaku koja je veća ili jednaka od oznake svoje djece.

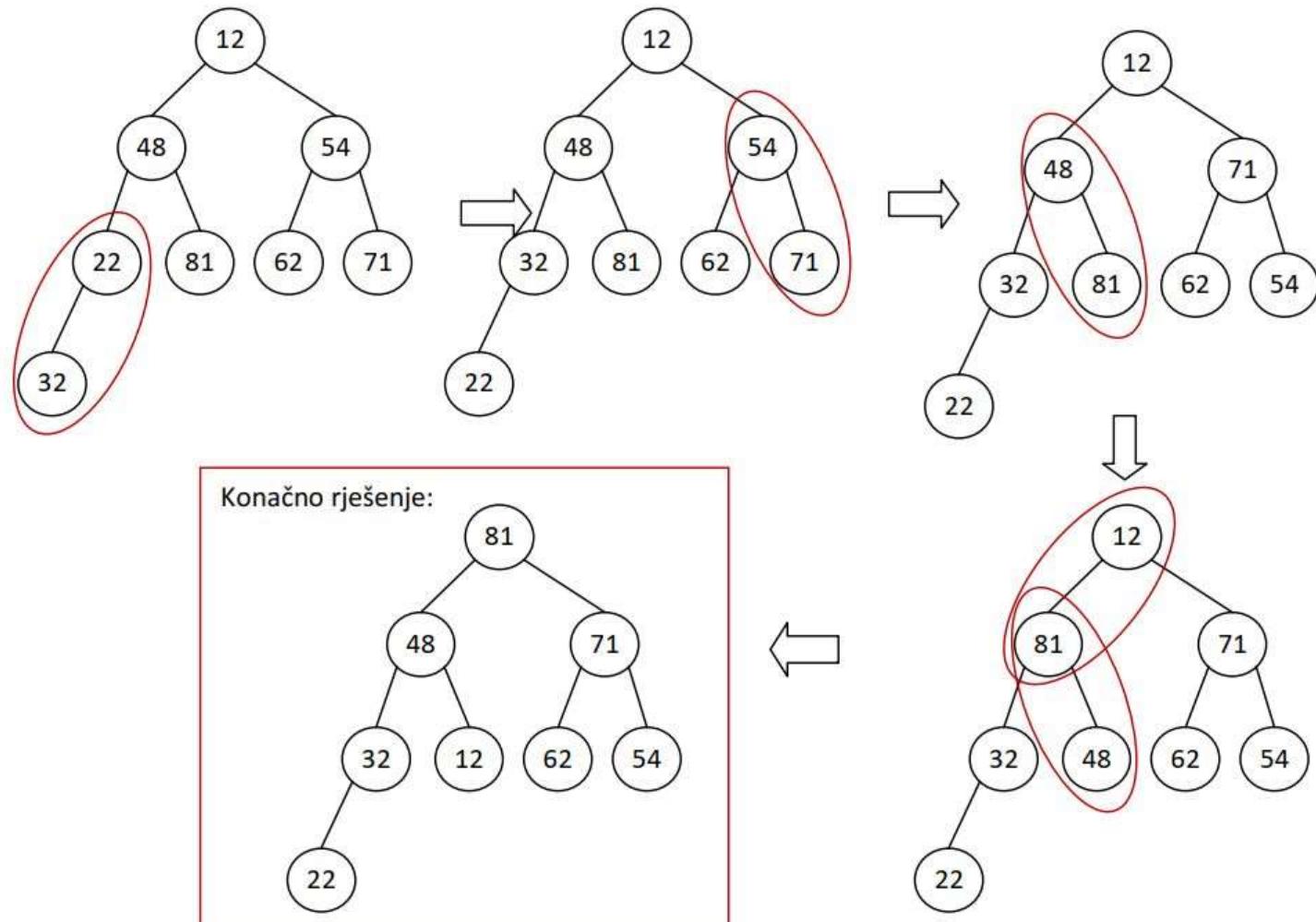


Hrpe

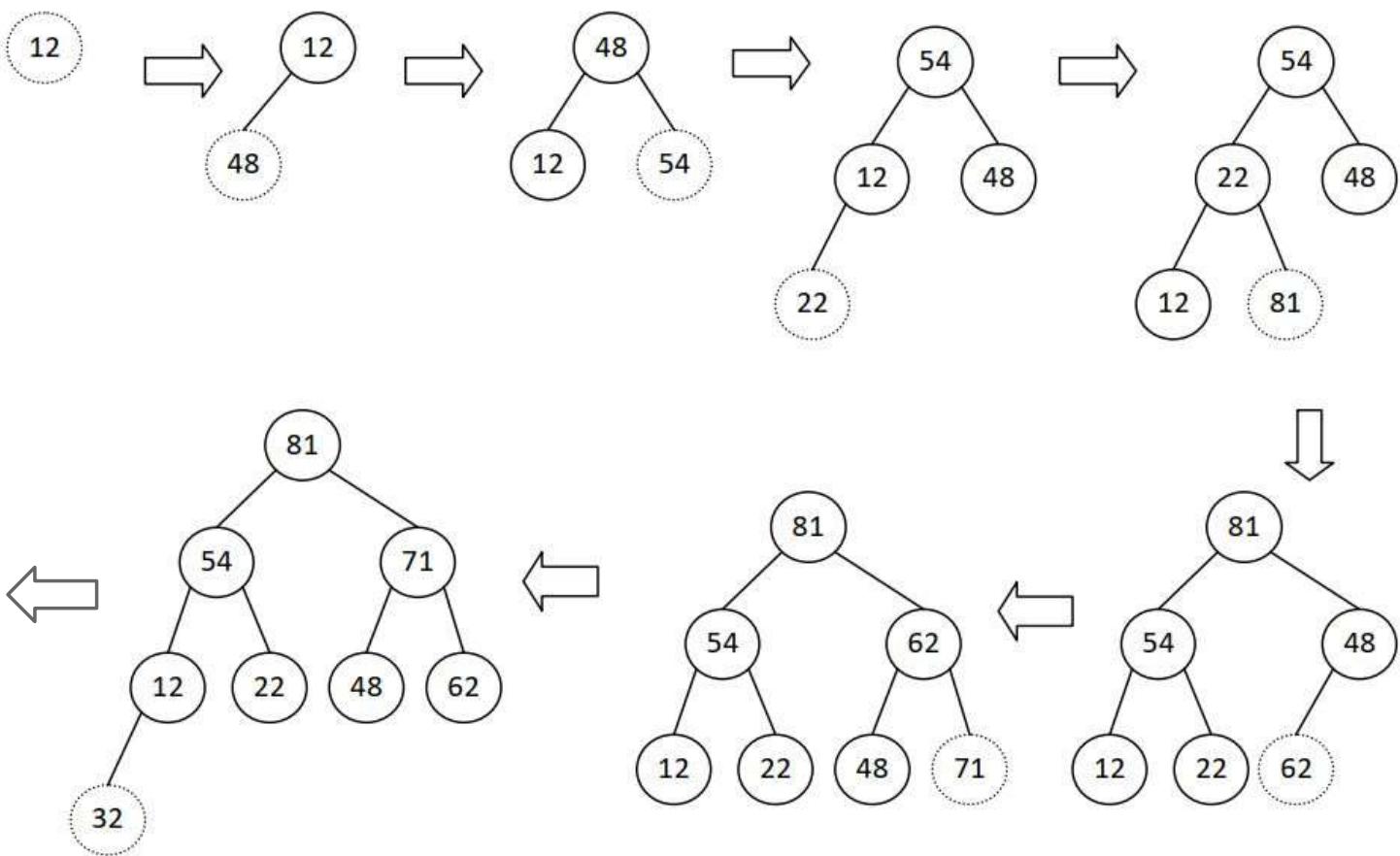
Zadan je niz brojeva: 12, 48, 54, 22, 81, 62, 71, 32.

- Ilustrirati (nacrtati stablo nakon svake promjene) stvaranje gomile (max hrpa) od zadanog vektora algoritmom čija je složenost za najgori slučaj $O(n)$.
- Ilustrirati (nacrtati stablo nakon svake promjene) stvaranje gomile (max hrpa) od zadanog vektora algoritmom čija je složenost za najgori slučaj $O(n\log n)$.

Hrpe

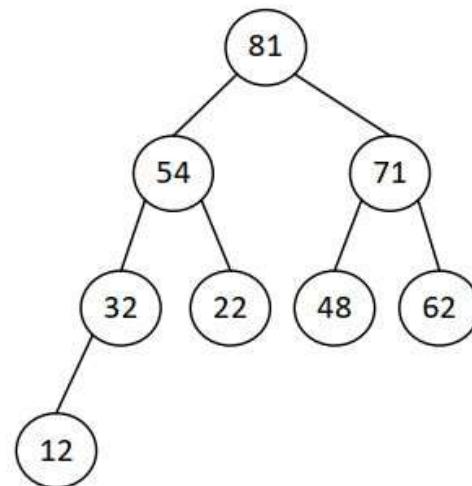


Hrpe



Hrpe

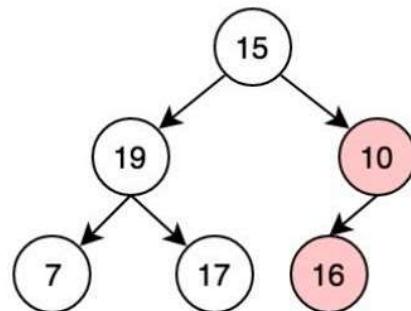
Potrebno je silazno sortirati zadano polje metodom heapsort uz prikazivanje svakog koraka sortiranja (potrebno je napisati sadržaj polja nakon svake promjene).



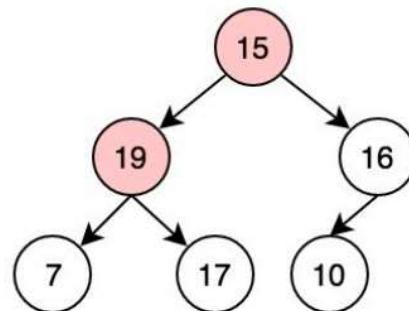
Hrpe

Primjer 2 za ulazni niz: 15, 19, 10, 7, 17, 16

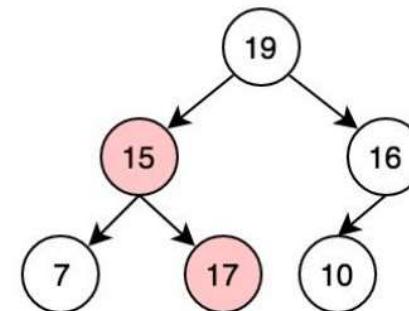
Uhrpljavanje



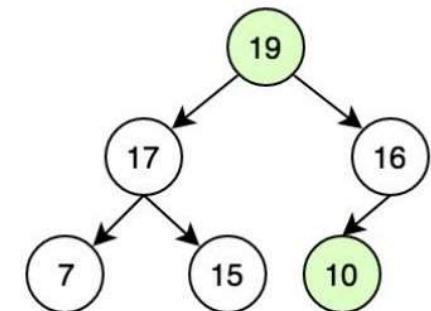
Uhrpljavanje



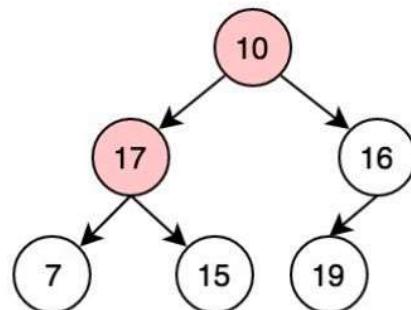
Uhrpljavanje



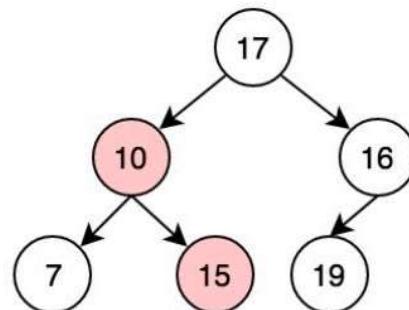
Max hrpa



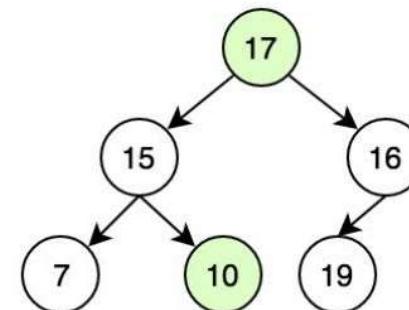
Uhrpljavanje



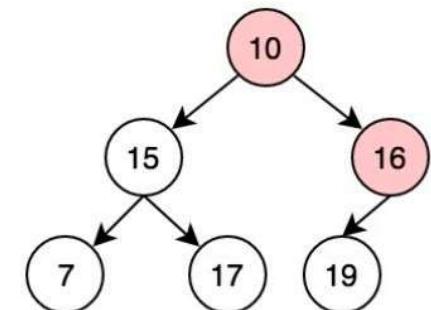
Uhrpljavanje



Max hrpa



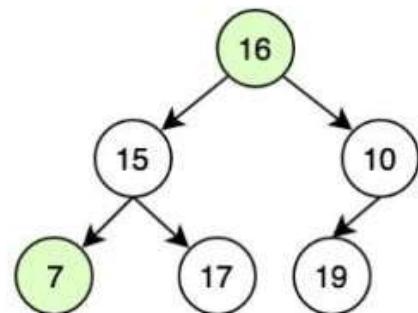
Uhrpljavanje



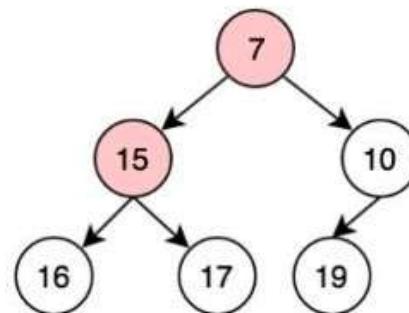
Hrpe

Primjer 2 za ulazni niz: 15, 19, 10, 7, 17, 16

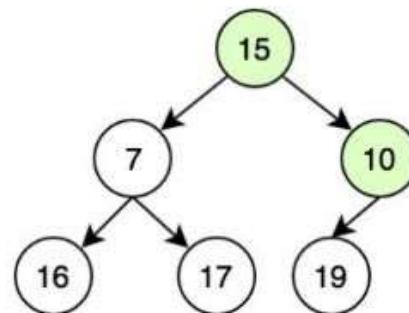
Max hrpa



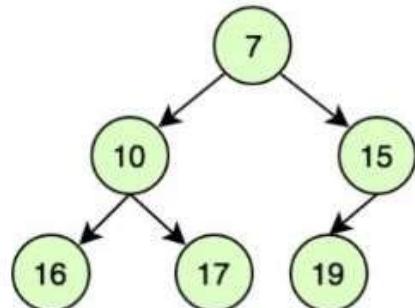
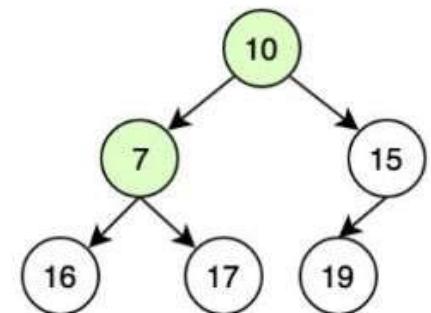
Uhrpljavanje



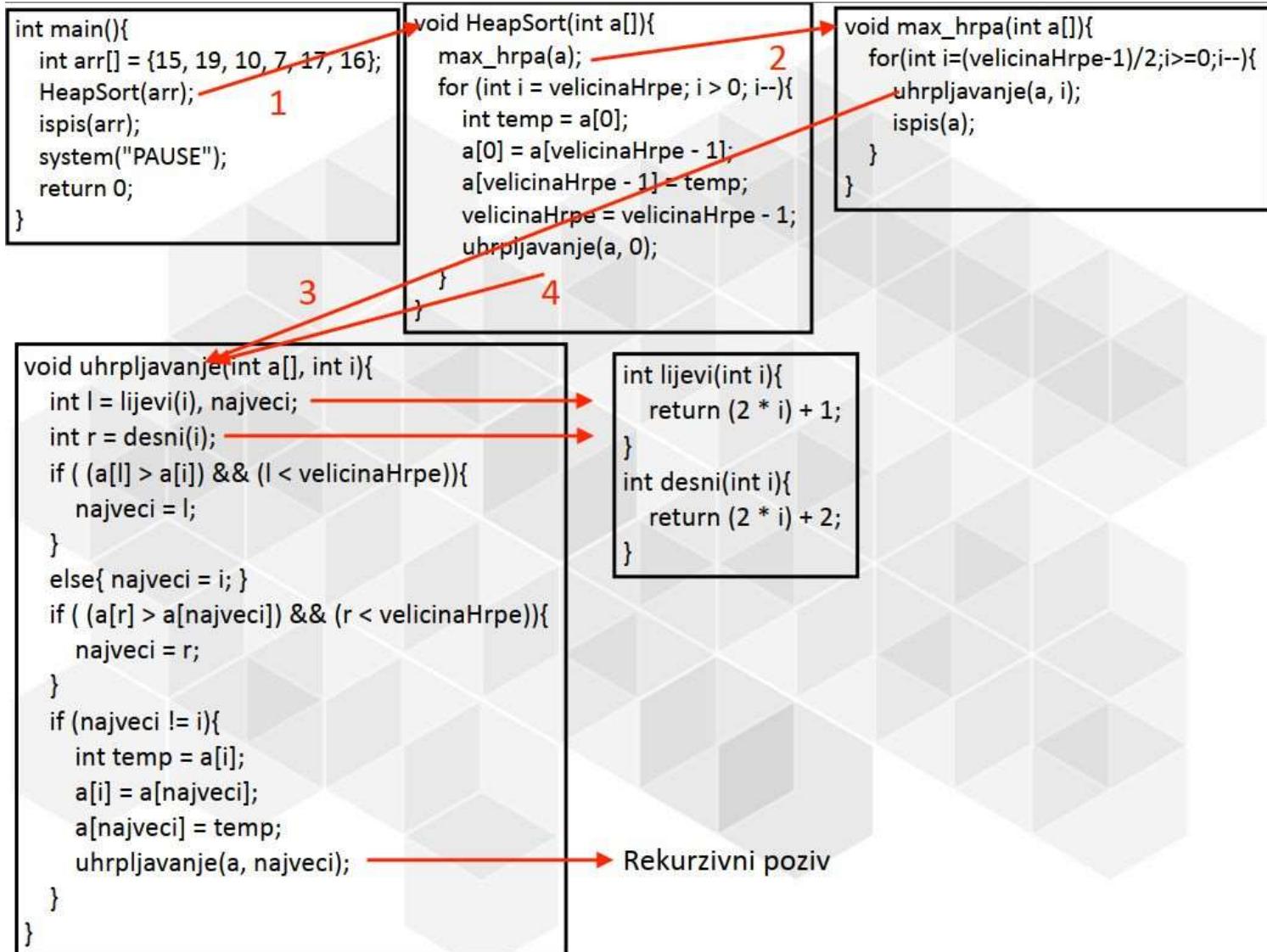
Max hrpa



Max hrpa



Hrpe



Zadatak

Napišite funkciju koja će za zadano polje prirodnih brojeva provjeriti je li max/min hrpa. Funkcija treba vratiti 1 ako je zadano polje gomila, a -1 ako nije.

```
bool provjeri(int niz[], int n) {
    for (int i = 0; i <= (n - 2) / 2; i++) {
        int lijevoDijete = 2 * i + 1;
        int desnoDijete = 2 * i + 2;

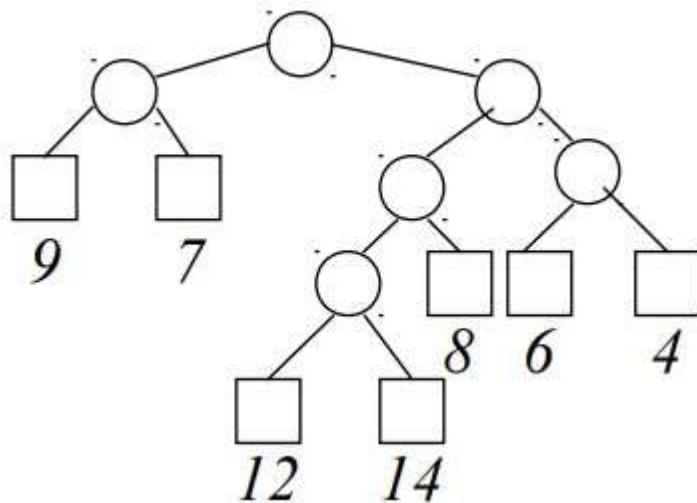
        if (lijevoDijete < n && niz[i] < niz[lijevoDijete]) return false;
        if (desnoDijete < n && niz[i] < niz[desnoDijete]) return false;
    }
    return true;
}
```

Auditorne vježbe

Huffmanovo kodiranje / Grafovi

Prošireno binarno stablo

- svaki čvor može imati ili dva ili niti jedan nasljednik
- težinski put do definiran je umnoškom $li \cdot wi$
- ukupni težinski put stabla definiran je s $W = \sum(li \cdot wi)$ za sve vanjske čvorove.
- vanjski čvorovi predstavljaju podatke kao što su znakovi u Huffmanovom kodiranju ili drugi elementi s određenim težinama / frekvencijama.



$$W = (9+7) \cdot 2 + (8+6+4) \cdot 3 + (12+14) \cdot 4 = 190$$

Huffmanovo stablo

- prošireno binarno stablo koje se iz zadanih vanjskih čvorova generira pomoću Huffmanovog algoritma (algoritam najmanjeg težinskog puta).
- vrijednosti vanjskih čvorova (frekvencija znakova) dolaze iz distribucije podataka koje se kodiraju.

Primjer:

ABCD

A - 0.16

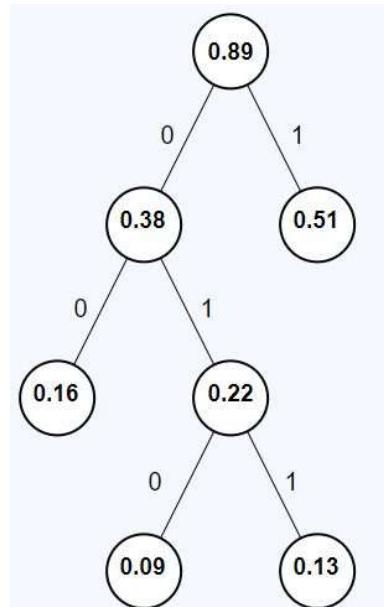
B - 0.51

C - 0.09

D - 0.13

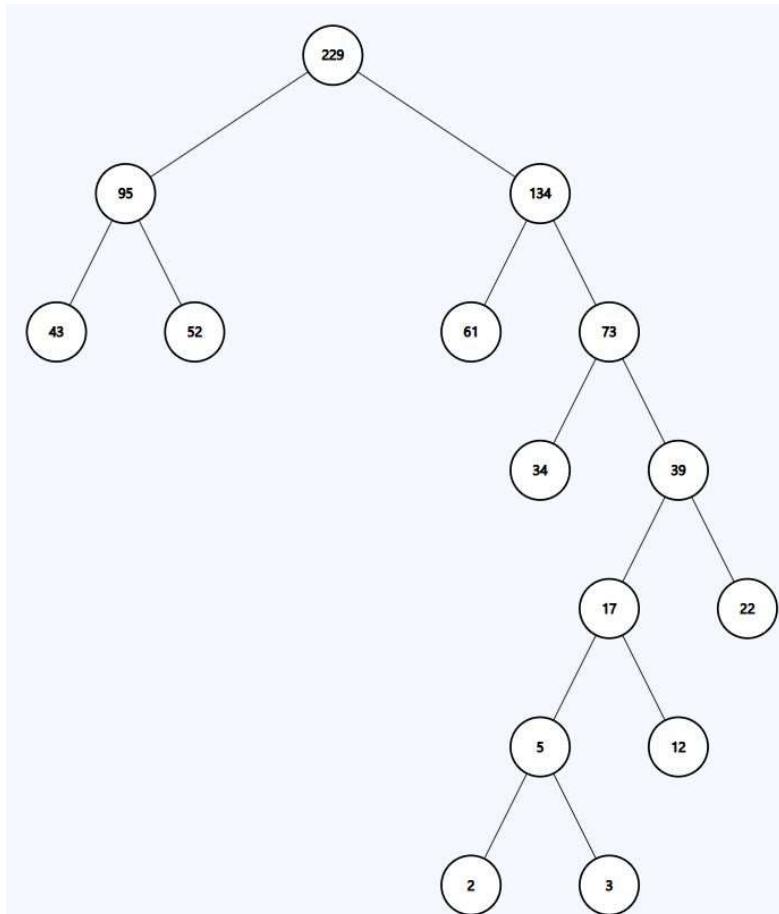
ABCD → 001010011 (kodirana poruka)

$$W = 3 * (0.09+0.13) + 2 * 0.16 + 1 * 0.15 = 1.49$$



Huffmanovo stablo

Napraviti Huffmanovo stablo za zadane vanjske čvorove $W = [2, 43, 3, 12, 52, 61, 34, 22]$. Ispisati Huffmanove kodove za svaki vanjski čvor.



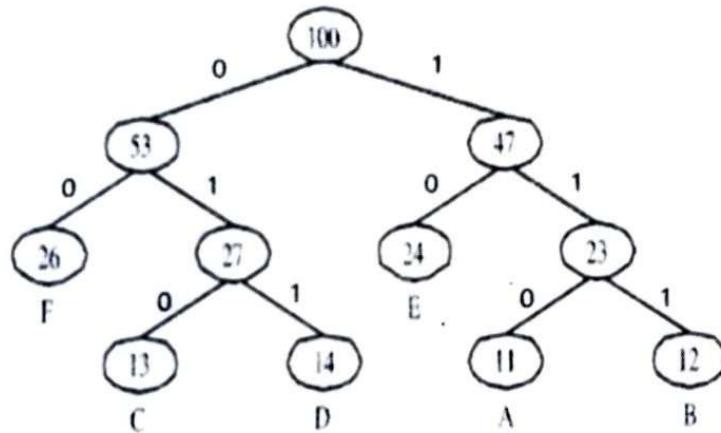
43 - 00
52 - 01
61 - 10
34 - 110
22 - 1111
12 - 11101
2 - 111000
3 - 111001

Huffmanovo stablo

Neka je zadano Huffmanovo stablo kao na slici:

Potrebno je:

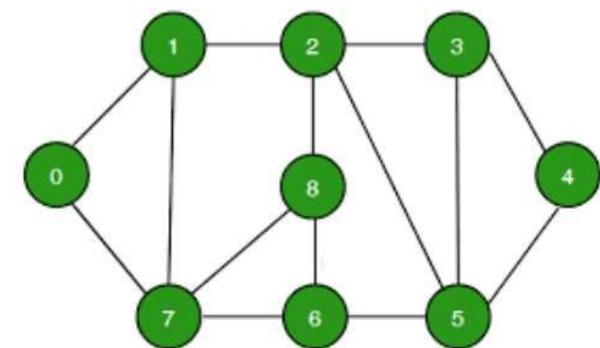
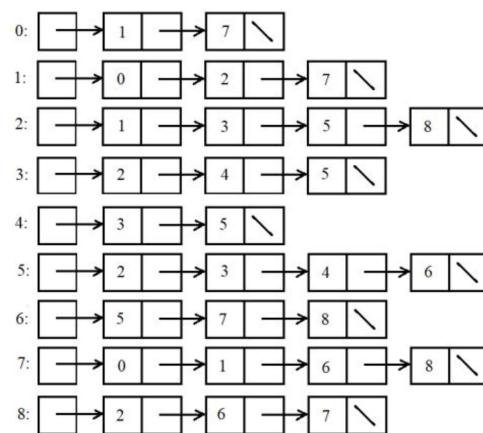
- kodirati tekst: D E A F E D (011101100010011)
- dekodirati poruku: 11110111110011110
- izračunati težinski put stabla
- ispisati POSTORDER obilazak koristeći numeričke vrijednosti čvora.
 $3*(13+14+11+12)+2*(26+24)=$



Graf Z1

Za graf na slici potrebno je napisati matricu susjedstva i niz povezanih listi kojima se takav graf prikazuje u računalu.

0	1	0	0	0	0	0	1	0
1	0	1	0	0	0	0	1	0
0	1	0	1	0	1	0	0	1
0	0	1	0	1	1	0	0	0
0	0	0	1	0	1	0	0	0
0	0	1	1	1	0	1	0	0
0	0	0	0	0	1	0	1	1
1	1	0	0	0	0	1	0	1
0	0	1	0	0	0	1	1	0



Graf Z2

Dati algoritam koji će izračunati i ispisati koliko ima čvorova u grafu G koji ima N čvorova i M grana koji imaju stupanj jednak K. (Stupanj je broj bridova koji ulaze ili izlaze iz određenog čvora). Npr. za graf na prethodnoj slici, i za zadani K=3, ovaj algoritam bi trebao ispisati 4, jer postoje 4 čvora koji imaju stupanj K=3, a to su konkretno: 1, 3, 6 i 8. Napisati kolika je složenost Vašeg algoritma.

Za svaki $i=1$ do N

 deg = 0

Za svaki $j=1$ do N

Ako je $S_{i,j}=1$ onda deg = deg + 1

Ako je deg=K onda ispiši i

Za svaki $i=1$ do N

 t = P_i

 deg = 0

Sve dok je $t \neq \text{NULL}$ činiti

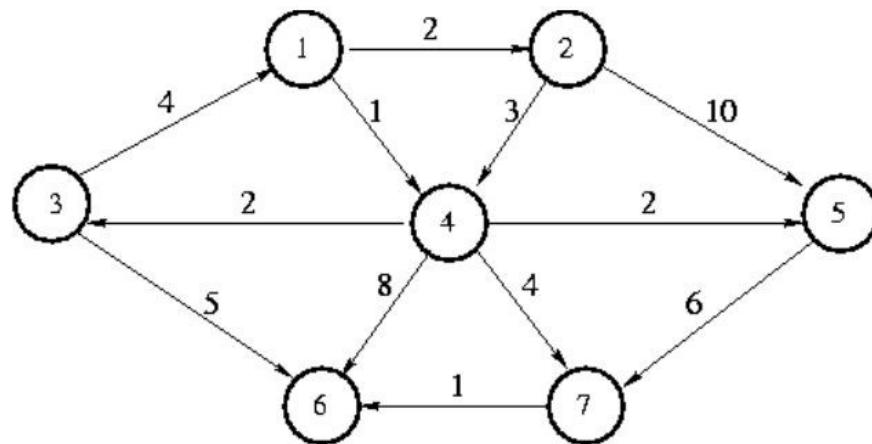
 deg = deg + 1

 t = t.sljedeci

Ako je deg=K onda ispiši i

Graf Z3

Za zadani usmjereni graf odrediti najkraće puteve s obzirom na zadane težine koristeći DIJKSTRIN algoritam. Zadani početni čvor je 3. Napisati dobiveni niz d. Kuda ide najkraći put i niz π nije potrebno napisati.



$$d=[4, 6, 0, 5, 7, 5, 9]$$

Graf Z4

Dati algoritam koji će za zadani graf G koji ima N čvorova i M bridova i zadani čvor c izračunati i ispisati sve čvorove koji su najudaljeniji od njega s obzirom na broj bridova. Napisati koja je složenost Vašeg algoritma. Npr. za graf na slici gore i za zadani čvor c=3, postoje dva najudaljenija čvora to su 0 i 7 i njih algoritam treba ispisati, a ta udaljenost je 3 grane.

Za svaki $i=1$ do N

Za svaki $j=1$ do N

$$W_{i,j} = S_{i,j}$$

Ako je $S_{i,j} = 0$ onda $W_{i,j} = \infty$

$$pc = c$$

Pozovi DIJKSTRIN algoritam

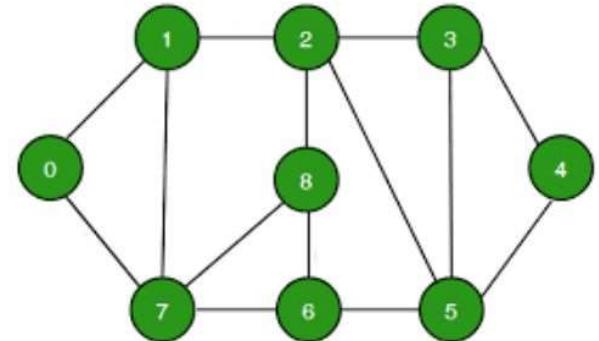
$$\max = -\infty$$

Za svaki $i=1$ do N

Ako je $d_i \neq \infty$ i $d_i > \max$ onda $\max = d_i$

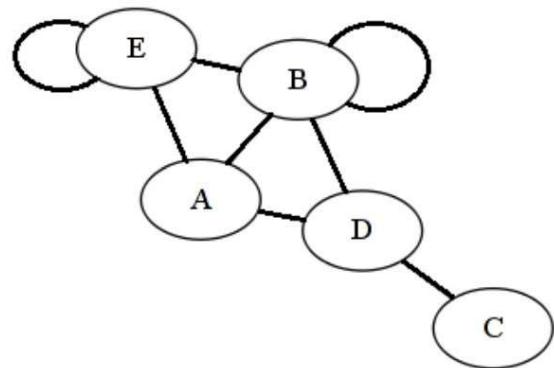
Za svaki $i=1$ do N

Ako je $d_i = \max$ onda ispiši i



Graf Z5

Zadan je graf kao na slici:



Potrebno je izračunati koliko ima puteva duljine 4 od čvora A do čvora B.

	A ₁	A ₂	A ₃	A ₄	A ₅
1	0	1	0	1	1
2	1	1	0	1	1
3	0	0	0	1	0
4	1	1	1	0	0
5	1	1	0	0	1

	A ₁	A ₂	A ₃	A ₄	A ₅
1	3	3	1	1	2
2	3	4	1	2	3
3	1	1	1	0	0
4	1	2	0	3	2
5	2	3	0	2	3

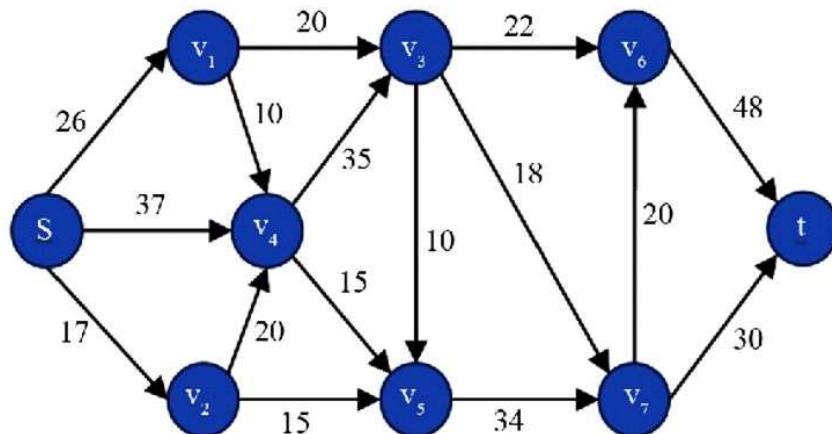
	A ₁	A ₂	A ₃	A ₄	A ₅
1	6	9	1	7	8
2	9	12	2	8	10
3	1	2	0	3	2
4	7	8	3	3	5
5	8	10	2	5	8

	A ₁	A ₂	A ₃	A ₄	A ₅
1	24	30	7	16	23
2	30	39	8	23	31
3	7	8	3	3	5
4	16	23	3	18	20
5	23	31	5	20	26



Graf Z6

Neka je zadana mreža kao na slici. Koristeći Edmonds-Karp-ov algoritam izračunati maksimalni protok ove mreže. Napisati svaki put duž kojeg se tok uvećavao te ukupni iznos maksimalnog protoka.



$$f_{\max}=72$$