1. Zadatak za zagrevanje
2.1. Binarno stablo
2.2. Binarno stablo pretrage
3. Visinski balansirano stablo pretrage
4. Turnirsko stablo
5. Fenwick stablo
6. Segmentno stablo
7. Uvod u grafove - MINESWEEPER

1. Telefonska kompanija je odlucila da clanovima sahovskog kluba dodeli sedmocifrene brojeve koji nece pocinjati sa 0 i sa 8, ali tako da se brojevi formiraju kretanjem konja po telefonskom brojcaniku kome su cifre rasporedjene na sledeci nacin:

1	**2**	**3**
4	**5**	**6**
7	**8**	**9**
	0	

Napisati program kojim se odredjuje koliko se ukupno brojeva moze formirati na ovaj nacin.

Resenje:
Rešenje ovog problema se realizuje formiranjem matrice, dimenzija $n \times 10$, čiji elementi **dp(i,j)** su jednaki broju šahovskih brojeva dužine **i** koji počinju sa **j**. Jasno je da **dp(i,j)** jednak zbiru **dp(i-1,к)** gde je **k** prolazi sve one brojeve sa kojih se može skočiti na **j**.Dakle svaka vrsta se može izračunati iz prethodne. Prva vrsta ima sve elemente jednake 1, očigledno da šahovskih brojeva dužine jedan(**i=1)** од **0..9** ima po jedan. Elementi druge vrste se računaju: **dp(2,1)=dp(1,6)+ dp(1,8), dp(2,2)= dp(1,7)+ dp(1,9) ...,**na isti način se računaju ostale vrste matrice sve do **n-te.** Rešenje je zbir svih elemenata zadnje vrste izuzimajući **dp(n,0)** и **dp(n,8).**

Kako se svaka vrsta matrice računa samo na osnovu prethodne to se u rešavanju ovog zadatka umesto matrice mogu koristiri samo dva niza **dp_1,** prethodna vrsta i **dp_2** vrsta koja se računa.

PSEUDO KOD

```
begin

read(n);
for i:=0 to 9 do dp_1 [i] := 1;

for i:=2 to n do
begin
dp_2[0] := dp_1[4]+dp_1[6];
dp_2[1] := dp_1[6]+dp_1[8];
dp_2[2] := dp_1[7]+dp_1[9];
dp_2[3] := dp_1[4]+dp_1[8];
dp_2[4] := dp_1[3]+dp_1[9]+dp_1[0];
dp_2[5] := 0;
dp_2[6] := dp_1[1]+dp_1[7]+dp_1[0];
dp_2[7] := dp_1[2]+dp_1[6];
dp_2[8] := dp_1[1]+dp_1[3];
dp_2[9] := dp_1[2]+dp_1[4];
dp_1 := dp_2;
end;

for i := 1 to 7 do inc(sol , dp_1[i]);

inc(sol , dp_1[9]);

writeln(sol);
end.
```

```
#include <stdio.h>
main()
{
int dp_1[10], dp_2[10], brojac, brojac2, n, zbir;
scanf("%d", &n);
for (brojac = 0; brojac <= 9; dp_1[brojac] = 1,
brojac++);
for (brojac = 2; brojac <= n; brojac++)
{
dp_2[0] = dp_1[4] + dp_1[6];
dp_2[1] = dp_1[6] + dp_1[8];
dp_2[2] = dp_1[7] + dp_1[9];
dp_2[3] = dp_1[4] + dp_1[8];
dp_2[4] = dp_1[3] + dp_1[9] + dp_1[0];
dp_2[5] = 0;
dp_2[6] = dp_1[1] + dp_1[7] + dp_1[0];
dp_2[7] = dp_1[2] + dp_1[6];
dp_2[8] = dp_1[1] + dp_1[3];
dp_2[9] = dp_1[2] + dp_1[4];
for( brojac2=0; brojac2 <= 9;
dp_1[brojac2] = dp_2[brojac2], brojac2++ );
}
for (brojac = 1, zbir = 0; brojac <= 7;
brojac++, zbir += dp_1[brojac]);
zbir += dp_1[9];
printf("Broj razlicitih brojeva telefona za
sahiste duzine %d je: %d.\n",n,zbir);
}
```

2.1. Binarna stabla

typedef struct cvor { int broj; struct cvor *levo, *desno; } Cvor;
typedef Cvor *Stablo;

Binarno stablo je skup čvorova koji su povezani na sledeci nacin:
1. Svaki cvor moze imati levog i desnog SINA (pri tom svaki od sinova moze biti i izostavljen, ili oba). Kazemo da je dati cvor RODITELJ svojim sinovima. Ako je cvor U roditelj cvoru V, tada pisemo da je U < V . Cvor koji nema ni levog ni desnog sina naziva se LIST.
2. Postoji jedan jedinstveni cvor takav da nema roditelja. Ovaj cvor nazivamo KOREN stabla.
3. U stablu nema ciklusa

PITANJA
1. Da li je binarni stablo povezana struktura (povezan graf)?
2. Ako binarno stablo ima visinu h, na koliko razlicitih nacina se moze stici od korena k do lista l?
3. Ako binarno stablo ima visinu h, na koliko razlicitih nacina se moze stici od lista r iz desnog stabla do lista l u levom stablu?

1. Dva binarna stabla su identična ako su ista po strukturi i sadržaju, odnosno oba korena imaju isti sadržaj i njihova odgovarajuća podstabla su identična. Napistati funkciju koja proverava da li su dva binarna stabla identična.

Analiza rešenja
Problem se može rešavati rekurzivnim načinom razmišljanja.

Testira se da li su dva zadata binarna stabla neprazna:
• ako su oba prazna, ekvivalentni su (return 1)
• ako su oba stabla neprazna, proverava se da li oba korena imaju isti koren
  1. ako ga imaju, rekurzivno se proverava da li su ekvivalentna leva podstabla
     - Ako su ekvivalentna leva podstabla, proveri se da li oba korena imaju istog desnog sina i ako ga imaju, rekurzivno se proveri da li su ekvivalentna desna podstabla. Ako jesu, onda su stabla ekvialentna.
     - Inace, nisu stabla ekvivalentna (return 0)

```
int EKV(Stablo d1, Stablo d2){
     if (!d1 && !d2) return 1; /* ako su oba prazna, ekvivalentni su */

     if (!d1 || !d2) return 0; /* inace, ako je jedno prazno, nisu ekvivalentni */

     if (d1->broj != d2->broj) return 0; /* razliciti koreni */

    return (EKV(d1->levo,d2->levo) && EKV(d1->desno,d2->desno) );
}
```

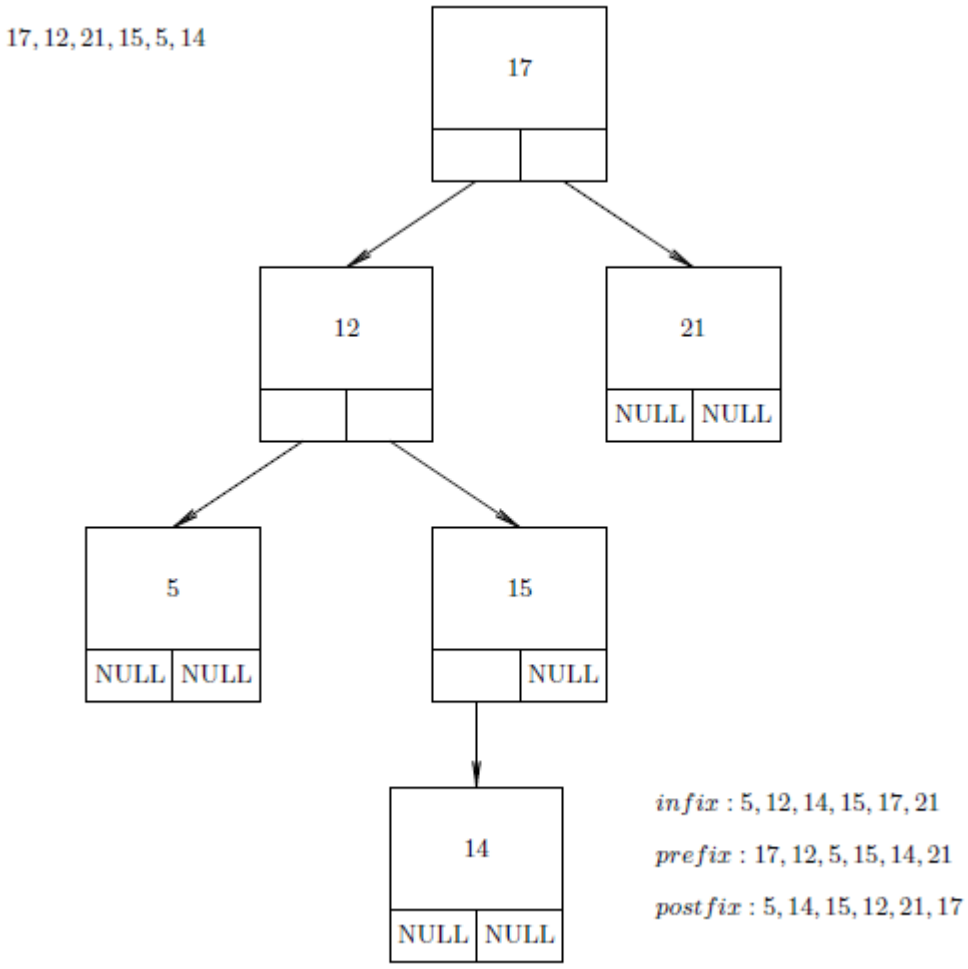
2. Napisati rekurzivnu funkciju sumaNivoa koja određuje sumu elemenata na N-tom nivou binarnog drveta. Koren tretirati kao 0-ti nivo.

```
int SumaNivoa (int n, Cvor *T)
{
   if (T)
     if (n)  return SumaNivoa(n-1,T->levo) + SumaNivoa(n-1,T->desno);
     else return T->broj;
  else return 0;
}
```

2.2. Binarno stablo pretrage

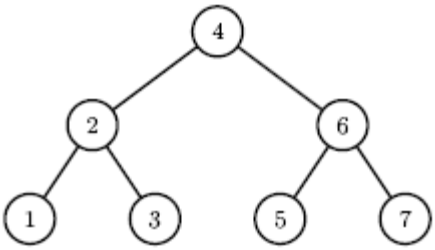
Binarno stablo pretrage je binarno stablo u kome važi: ključ svakog čvora veći je od ključeva svih čvorova levog podstabla, a manji od ključeva svih čvorova desnog podstabla. Pretpostavimo zbog jednostavnosti da su ključevi svih čvorova različiti. Ovakva organizacija omogu ćava efikasno pretraživanje.

INFIX, INORDER obilazak stabla = levo, koren, desno
PREFIX, PREORDER obilazak stabla = koren, levo, desno,
POSTFIX, POSTORDER obilazak stabla = levo, desno, koren



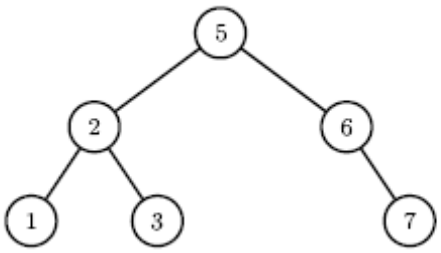
3. Za sekvencu ključeva 4, 2, 1, 3, 6, 5, 7 odredite binarno stablo pretraživanja koje se dobija kada se u početno prazno stablo ti čvorovi dodaju jedan po jedan u datom redosledu.

Rešenje



4. Skicirajte stablo koje se dobija kada se ukloni koren sa stabla iz prethodnog zadatka.
Podsetite se funkcije izost_u(Stablo koren, int b) iz 10. zadatka, kao i uklanjanja korena iz hipa.

Rešenje



5. Nacrtati binarno stablo za koje INORDER (LKD) obilazak daje *zstoipd*, a PREORDER (KLD) obilazak *iosztpd*. Ključ čvora je slovo engleske abecede.

SKICA RESENJA:

1. KLD[0] mora biti koren stabla.

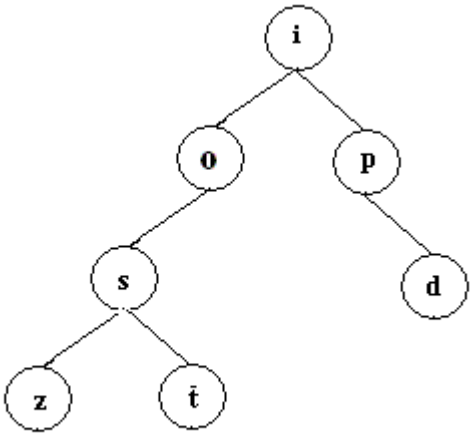
2. Ako je *n* ukupan broj čvorova i ako je *i* pozicija korena u nisci LKD, onda je

LKD[0..i-1] zapis levog podstabla u redosledu LKD i

LKD[i+1..n-1] zapis desnog podstabla u redosledu LKD.

3. Ako je *i* pozicija korena u nisci LKD, onda je KLD[1..i] zapis levog poddrveta u redosledu KLD (u levom poddrvetu je *i* cvorova). Slicno, KLD[i+1..n-1] je zapis desnog poddrveta u poretku KLD.

4. Rekurzivno primenimo pravila 1..3 (tj. KLD[1] je koren levog podstabla, KLD[i+1] je koren desnog stabla, i=4 jer LKD[4]=koren=*e*)



6. Odrediti binarno stablo za koje INORDER (LKD) obilazak daje *dacbfegijhkl,* a PREORDER (KLD) obilazak *gfdcabehijkl* U čvoru stabla zapisano je slovo engleske abecede.

7. Napisati C potprogram koji za date inorder i preorder zapise (u vidu niski *lkd* i *kld*) jednog istog drveta nalazi i ispisuje njegov postorder zapis. Pretpostaviti da stablo ima do 80 cvorova oznacenih jednim ASCII znakom.

8. Znamo da iz inorder i preorder obilaska binarnog stabla pretrage se moze rekonstruisati graficka predstava tog stabla. Da li je to moguce ako imamo zadate preorder i postorder obilaske? Odgovor obrazloziti primerom ili recima. (pogledati slican zadatak sa vezbi)

RESENJE: NE!!!
Primer:
Preorder: AB
Postorder: BA

Postoje dva stabla koja odgovaraju ovim obilascima

Stablo: A

B

Stablo A

B

9. OSNOVNI RAD SA BINARNIM STABLOM PRETRAGE

```
#include <stdio.h>

/* Struktura koja predstavlja cvor drveta */
typedef struct _cvor
{ int broj;
  struct _cvor *l, *d;
} cvor;


/* Pomocna funkcija za kreiranje cvora. */
cvor* napravi_cvor(int b) {
cvor* novi = (cvor*)malloc(sizeof(cvor));
if (novi == NULL)
{ fprintf(stderr, "Greska prilikom alokacije memorije"); exit(1);}
novi->broj = b;
novi->l = NULL;
novi->d = NULL;
return novi;
}

/* Funkcija umece broj b u drvo ciji je koren dat preko pokazivaca koren. Funkcija vraca pokazivac na koren
novog drveta */
cvor* ubaci_u_drvo(cvor* koren, int b)
{
if (koren == NULL) return napravi_cvor(b);
if (b < koren->broj) koren->l = ubaci_u_drvo(koren->l, b);
else koren->d = ubaci_u_drvo(koren->d, b);
return koren;
}
```

```
/* Funkcija proverava da li dati broj postoji u drvetu */
int pronadji(cvor* koren, int b)
{
if (koren == NULL) return 0;
if (koren->broj == b) return 1;
if (b < koren->broj) return pronadji(koren->l, b);
else return pronadji(koren->d, b);
}

/* Funkcija ispisuje sve cvorove drveta u infiksnom redosledu */
void ispisi_drvo(cvor* koren) {
if (koren != NULL)
{
ispisi_drvo(koren->l); printf("%d ", koren->broj); ispisi_drvo(koren->d);
}
}

/* Funkcija oslobadja memoriju koju je drvo zauzimalo */
void obrisi_drvo(cvor* koren) {
if (koren != NULL)
{
/* Oslobadja se memorija za levo poddrvo */
obrisi_drvo(koren->l);
/* Oslobadja se memorija za desno poddrvo */
obrisi_drvo(koren->d);
/* Oslobadja se memorija koju zauzima koren */
free(koren);
}
}

/* Funkcija sumira sve vrednosti binarnog stabla */
int suma_cvorova(cvor* koren)
{
if (koren == NULL) return 0;
return suma_cvorova(koren->l) +koren->broj +suma_cvorova(koren->d);
}

/* Funkcija prebrojava broj cvorova binarnog stabla */
int broj_cvorova(cvor* koren)
{
if (koren == NULL) return 0;
return broj_cvorova(koren->l) +1 +broj_cvorova(koren->d);
}

/* Funkcija prebrojava broj listova binarnog stabla */
int broj_listova(cvor* koren)
{
if (koren == NULL) return 0;
if (koren->l == NULL && koren->d == NULL) return 1;
return broj_listova(koren->l) +broj_listova(koren->d);
}

/* Funkcija izracunava sumu listova binarnog stabla */
int suma_listova(cvor* koren)
{
if (koren == NULL) return 0;
if (koren->l == NULL && koren->d == NULL) return koren->broj;
```

```
return suma_listova(koren->l) +suma_listova(koren->d);
}

/* Funkcija ispisuje sadrzaj listova binarnog stabla */
void ispisi_listove(cvor* koren)
{
if (koren == NULL) return;
ispisi_listove(koren->l);
if (koren->l == NULL && koren->d == NULL)
printf("%d ", koren->broj);
ispisi_listove(koren->d);
}

/* Funkcija pronalazi maksimalnu vrednost u drvetu
Koristi se cinjenica da je ova vrednost smestena u najdesnjem listu */
int max_vrednost(cvor* koren)
{
if (koren==NULL) return 0;
if (koren->d==NULL) return koren->broj;
return max_vrednost(koren->d);
}

/* Iterativna funkcija za pronalazenje maksimalne vrednosti. */
int max_vrednost_nerekurzivno(cvor* koren)
{
if (koren==NULL) return 0;
else
{
cvor* tekuci;
for (tekuci=koren; tekuci->d!=NULL; tekuci=tekuci->d);
return tekuci->broj;
}
}

#define max(a,b) (((a)>(b))?(a):(b))

/* Funkcija racuna "dubinu" binarnog stabla */
int dubina(cvor* koren)
{
if (koren==NULL) return 0;
else
{ int dl=dubina(koren->l);
int dd=dubina(koren->d);
return 1+max(dl,dd);
}
}

/* Program koji testira rad prethodnih funkcija */
main()
{
cvor* koren = NULL;
koren = ubaci_u_drvo(koren, 1);
koren = ubaci_u_drvo(koren, 8);
koren = ubaci_u_drvo(koren, 5);
koren = ubaci_u_drvo(koren, 3);
koren = ubaci_u_drvo(koren, 7);
koren = ubaci_u_drvo(koren, 6);
```

```
koren = ubaci_u_drvo(koren, 9);
if (pronadji(koren, 3)) printf("Pronadjeno 3\n");
if (pronadji(koren, 2)) printf("Pronadjeno 2\n");
if (pronadji(koren, 7)) printf("Pronadjeno 7\n");
ispisi_drvo(koren);
putchar('\n');
printf("Suma cvorova : %d\n", suma_cvorova(koren));
printf("Broj cvorova : %d\n", broj_cvorova(koren));
printf("Broj listova : %d\n", broj_listova(koren));
printf("Suma listova : %d\n", suma_listova(koren));
printf("Dubina drveta : %d\n", dubina(koren));
printf("Maximalna vrednost : %d\n", max_vrednost(koren));
ispisi_listove(koren);
obrisi_drvo(koren);
}

/*
Pronadjeno 3
Pronadjeno 7
1 3 5 6 7 8 9
Suma cvorova : 39
Broj cvorova : 7
Broj listova : 3
Suma listova : 18
Dubina drveta : 5
Maximalna vrednost : 9
3 6 9
*/
```

10. Implementacija funkcija koje vrse specificnu obradu nad cvorovima binarnog stabla

```
#include <stdio.h>
#include <stdlib.h>


typedef struct cvor { int broj; struct cvor *levo, *desno; } Cvor;
typedef Cvor *Stablo;

Stablo stvori (void);                /* Stavaranje praznog stabla.      */
int vel (Stablo koren);               /* Broj cvorova u stablu.          */
int zbir (Stablo koren);              /* Zbir brojeva u stablu.          */
void pisi_kld (Stablo koren);         /* Prefiksno ispisivanje.          */
void pisi_lkd (Stablo koren);         /* Infiksno ispisivanje.           */
void pisi_ldk (Stablo koren);         /* Postfiksno ispisivanje.         */
void crtaj (Stablo koren, int nivo);  /* Graficki prikaz stabla.         */
int pojav (Stablo koren, int b);      /* Broj pojavljivanja u stablu.    */
int min_u (Stablo koren);             /* Najmanji u uredjenom stablu.    */
int max_u (Stablo koren);             /* Najveci u uredjenom stablu.     */
int min_n (Stablo koren);             /* Najmanji u neuredjenom stablu.  */
int max_n (Stablo koren);             /* Najveci u neuredjenom stablu.   */
int uredjeno (Stablo koren);          /* Da li je stablo uredjeno?       */
Cvor *nadji_u (Stablo koren, int b);  /* Trazenje u uredjenom stablu.    */
Cvor *nadji_n (Stablo koren, int b);  /* Trazenje u neuredjenom stablu.  */
Stablo dodaj_u (Stablo koren, int b); /* Dodavanje u uredjeno stablo.    */
Stablo dodaj_n (Stablo koren, int b); /* Dodavanje u neuredjeno stablo.  */
```

```
Stablo citaj_u (int n);              /* Citanje uredjenog stabla.       */
Stablo citaj_n (int n);              /* Citanje neuredjenog stabla.     */
Stablo brisi (Stablo koren);         /* Brisanje celog stabla.          */
Stablo izost_u (Stablo koren, int b);    /* Izost. iz uredjenog stabla.     */
Stablo izost_n (Stablo koren, int b);    /* Izost. iz neuredjenog stabla.   */
Stablo balans_u (Stablo koren);          /* Balansiranje uredjenog stabla.  */
Stablo balans_n (Stablo koren);          /* Balansiranje neuredjenog satbla.*/
int moze (Stablo koren);             /* Da li moze uredjena radnja? */
Stablo radi (Stablo (*f)(Stablo,int), Stablo koren); /* Primena operacije na stablo za svaki procitani broj */

void main () {
  Stablo koren = stvori ();   //stablo
  int kraj = 0, broj, n;  //indikator kraja rada, element u cvoru stabla, duzina
  char izbor[2];          //izbor korisnika sa menija opcija

 //obrada menija opcija koje se prikazuju korisniku
  while (!kraj) {
    printf ("\nDodavanje brojeva:       a) uredjeno     b) neuredjeno\n"
            "Izostavljanje brojeva:   c) uredjeno     d) neuredjeno\n"
            "Citanje stabla:          e) uredjeno     f) neuredjeno\n"
            "Najmanji element:        g) uredjeno     h) neuredjeno\n"
            "Najveci element:         i) uredjeno     j) neuredjeno\n"
            "Pretrazivanje:           k) uredjeno     l) neuredjeno\n"
            "Balansiranje:            m) uredjeno     n) neuredjeno\n"
            "Pisanje stabla:          p) koren-levo-desno\n"
            "                         q) levo-koren-desno (uredjeno)\n"
            "                         r) levo-desno-kren\n"
            "                         s) crtanje\n"
            "1. Velicina stabla       2. Zbir elemenata\n"
            "3. Broj pojavljivanja    4. Praznjenje stabla\n"
            "                         0. Zavrsetak rada\n\n"
            "Vas izbor? "
          );
    scanf ("%s", &izbor);
    switch (izbor[0]) {
    case 'a': case 'A': /* Dodavanje brojeva u uredjeno stablo: */
      if (moze (koren)) koren = radi (dodaj_u, koren); break;
    case 'b': case 'B': /* Dodavanje brojeva u neuredjeno stablo: */
                  koren = radi (dodaj_n, koren); break;
    case 'c': case 'C': /* Izostavljanje brojeva iz uredjenog stabla: */
      if (moze (koren)) koren = radi (izost_u, koren); break;
    case 'd': case 'D': /* Izostavljanje brojeva iz neuredjenog stabla: */
                  koren = radi (izost_n, koren); break;
    case 'e': case 'E': /* Citanje uredjenog stabla: */
      printf ("Duzina?    "); scanf ("%d", &n);
      printf ("Brojevi?   "); koren = brisi (koren); koren = citaj_u (n);
      break;
    case 'f': case 'F': /* Citanje neuredjenog stabla: */
      printf ("Duzina?    "); scanf ("%d", &n);
      printf ("Brojevi?   "); koren = brisi (koren); koren = citaj_n (n);
      break;
    case 'g': case 'G': case 'h': case 'H':
    case 'i': case 'I': case 'j': case 'J':
      if (koren) switch (izbor[0]) {
        case 'g': case 'G': /* Najmanji element uredjenog stabla: */
          if (moze (koren)) printf ("min=       %d\n", min_u (koren)); break;
        case 'h': case 'H': /* Najmanji element neuredjenog stabla: */
```

```
                printf ("min=       %d\n", min_n (koren)); break;
      case 'i': case 'I': /* Najveci element uredjenog stabla: */
        if (moze (koren)) printf ("max=       %d\n", max_u (koren)); break;
      case 'j': case 'J': /* Najveci element neuredjenog stabla: */
                printf ("max=       %d\n", max_n (koren)); break;
    } else printf ("*** Stablo je parzno! ***\a\n");
    break;
  case 'k': case 'K': /* Broj pojavljivanja u uredjenom stablu: */
    if (moze (koren)) {
      printf ("Broj?      "); scanf ("%d", &broj);
      printf ("Broj se%s nalazi u stablu.\n",
            (nadji_u (koren, broj) != NULL ? "" : " NE"));
    } break;
  case 'l': case 'L': /* Broj pojavljivanja u neuredjenom stablu: */
      printf ("Broj?      "); scanf ("%d", &broj);
      printf ("Broj se%s nalazi u stablu.\n",
            (nadji_n (koren, broj) != NULL ? "" : " NE"));
      break;
  case 'm': case 'M': /* Balansiranje uredjenog stabla: */
    if (moze (koren)) koren = balans_u (koren); break;
  case 'n': case 'N': /* Balansiranje neuredjenog stabla: */
                koren = balans_n (koren); break;
  case 'p': case 'P': /* Pisanje stabla koren-levo-desno: */
    printf ("Stablo=    "); pisi_kld (koren); putchar ('\n'); break;
  case 'q': case 'Q': /* Pisanje stabla levo-koren-desno: */
    printf ("Stablo=    "); pisi_lkd (koren); putchar ('\n'); break;
  case 'r': case 'R': /* Pisanje stabla levo-desno-koren: */
    printf ("Stablo=    "); pisi_ldk (koren); putchar ('\n'); break;
  case 's': case 'S': /* Crtanje stabla: */
    crtaj (koren, 0); break;
  case '1':           /* Velicina stabla: */
    printf ("Vel=       %d\n", vel (koren)); break;
  case '2':           /* Zbir elemenata stabla: */
    printf ("Zbir=      %d\n", zbir (koren)); break;
  case '3':           /* Broj pojavljivanja datog broja: */
    printf ("Broj?      "); scanf ("%d", &broj);
    printf ("Broj se pojavljuje %d puta.\n", pojav (koren, broj));
    break;
  case '4':           /* Praznjenje stabla: */
    koren = brisi (koren); break;
  case '0':           /* Zavrsetak rada: */
    kraj = 1; break;
  default: /* Pogresan izbor: */
    printf ("*** Nedozvoljeni izbor! ***\a\n"); break;
  }
 }
}


Stablo stvori (void) { return NULL; }  /* Stvaranje praznog stabla.      */

int vel (Stablo koren)                  /* Broj cvorova u stablu.         */
  { return koren ? 1 + vel (koren->levo) + vel (koren->desno) : 0; }

int zbir (Stablo koren)                 /* Zbir brojeva u stablu.         */
  { return koren ? koren->broj + zbir (koren->levo) + zbir (koren->desno) : 0; }
```

```
void pisi_lkd (Stablo koren) {          /* Infiksno ispisivanje.          */
  if (koren) {
    pisi_lkd (koren->levo);
    printf ("%d ", koren->broj);
    pisi_lkd (koren->desno);
  }
}
```

/* Broj rekurzivnih poziva jednak je broju čvorova stabla *n*, a trajanje svakog poziva je *O(1)*, te je složenost algoritma *O(n)*. Željeni niz dobija se nadovezivanjem niza koji odgovara levom podstablu, ključa korena stabla i niza koji odgovara desnom podstablu.
***/**

```
void pisi_kld (Stablo koren) {          /* Prefiksno ispisivanje.          */
  if (koren) {
    printf ("%d ", koren->broj);
    pisi_kld (koren->levo);
    pisi_kld (koren->desno);
  }
}

void pisi_ldk (Stablo koren) {          /* Postfiksno ispisivanje.         */
  if (koren) {
    pisi_ldk (koren->levo);
    pisi_ldk (koren->desno);
    printf ("%d ", koren->broj);
  }
}

void crtaj (Stablo koren, int nivo) {    /* Graficki prikaz stabla.        */
  if (koren) {
    crtaj (koren->desno, nivo+1);
    printf ("%*s%d\n", 4*nivo, "", koren->broj);
    crtaj (koren->levo,  nivo+1);
  }
}

int pojav (Stablo koren, int b)         /* Broj pojavljivanja broja b u stablu.    */
{return
    koren ? (koren->broj==b)+pojav(koren->levo,b)+pojav(koren->desno,b) : 0;
}
```

/*Iz definicije stabla binarne pretrage (da leva deca su manja ili jednaka od roditelja) sledi da čvor sa najmanjom vrednošću se nalazi u najlevljem čvoru stabla. Zato bi skica algoritma koji traži najmanju vrednost u uređenom stablu bila da se počev od korena vrši spuštanje u stablu po levim pokazivačima sve dok se ne dođe do čvora koji nema levog sina.
SKICA ITERATIVNE VERZIJE:

```
   p=koren;
   while (p->levo) p=p->levo;
   return p;
```

ILI REKURZIVNA VERZIJA REŠENJA */

```
int min_u (Stablo koren)                /* Najmanji u uredjenom stablu.    */
  { return koren->levo  ? min_u (koren->levo ) : koren->broj; }
```

/* Slično se nalazi ključ sa najvećom vrednošću u stablu koji je lociran u najdešnjem čvoru.

SKICA ITERATIVNE VERZIJE:
p=koren;
while (p->desno) p=p->desno;
return p;
ILI REKURZIVNA VERZIJA REŠENJA */

```
int max_u (Stablo koren)              /* Najveci u uredjenom stablu.     */
  { return koren->desno ? max_u (koren->desno) : koren->broj; }
```

/* Kako proveriti da li je stablo uredjeno?
Uređeno je svako prazno stablo.
U neuređenom stablu važi
Ili je levo podstablo neprazno i neuređeno
Ili je desno podstablo neprazno i neuređeno

U neuređenom levom podstablu važi da najveći čvor levog podstabla je veći od korena stabla

U neuređenom desnom podstablu važi da najmanji čvor desnog podstabla je manji od korena stabla
***/**

```
int uredjeno (Stablo koren) {          /* Da li je stablo uredjeno?       */
  if (! koren) return 1;
  if (koren->levo  && (! uredjeno (koren->levo ) ||
                  max_u (koren->levo)  > koren->broj)) return 0;
  if (koren->desno && (! uredjeno (koren->desno) ||
                  min_u (koren->desno) < koren->broj)) return 0;
  return 1;
}
```

/* NALAŽENJE ČVORA SA NAJMANJOM I NAJVEĆOM VREDNOŠĆU U NEUREĐENOM BINARNOM STABLU
IDEJA: Najmanji čvor stabla je minimum tri broja
min(koren->broj, min(koren->levo), min (koren->desno))

Slično i za najveći čvor.
*/

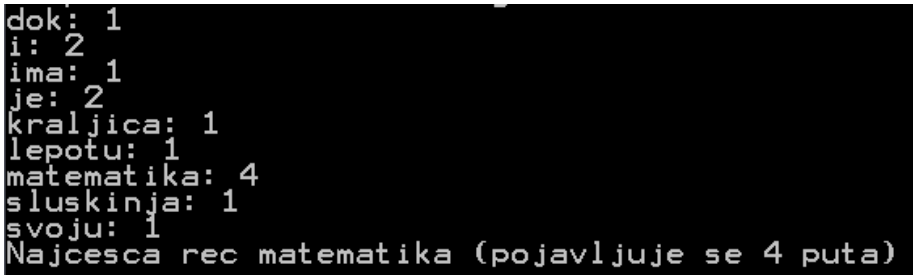
```
int min_n (Stablo koren) {             /* Najmanji u neuredjenom stablu.  */
  int m = koren->broj, k;

  /*m=min(koren->broj, min(koren->levo)) */
  if (koren->levo ) { k = min_n (koren->levo ); if (k < m) m = k; }

    /*m=min(koren->broj, min(koren->desnoo)) */
 if (koren->desno) { k = min_n (koren->desno); if (k < m) m = k; }


   /* na kraju   m=min(koren->broj, min(koren->levo), min(koren->desno)) */
  return m;
}

int max_n (Stablo koren) {             /* Najveci u neuredjenom stablu.   */
  int m = koren->broj, k;
  if (koren->levo ) { k = max_n (koren->levo ); if (k > m) m = k; }
```

```
  if (koren->desno) { k = max_n (koren->desno); if (k > m) m = k; }
  return m;
}

Cvor *nadji_u (Stablo koren, int b) {    /* Trazenje u uredjenom stablu.    */
  if (! koren)          return NULL;
  if (koren->broj == b) return koren;
  if (koren->broj >  b) return nadji_u (koren->levo,  b);
                        return nadji_u (koren->desno, b);
}

Cvor *nadji_n (Stablo koren, int b) {    /* Trazenje u neuredjenom stablu.  */
  if (! koren)          return NULL;
  if (koren->broj == b) return koren;
  { Cvor *cvr = nadji_n (koren->levo, b); if (cvr) return cvr; }
  return nadji_n (koren->desno, b);
}

Stablo dodaj_u (Stablo koren, int b) {   /* Dodavanje u uredjeno stablo.    */
  if (! koren) {
    koren = malloc (sizeof(Cvor));
    koren->broj = b; koren->levo = koren->desno = NULL;
  } else if (koren->broj > b)
    koren->levo  = dodaj_u (koren->levo,  b);
  else if (koren->broj < b)
    koren->desno = dodaj_u (koren->desno, b);
  else if (rand() / (RAND_MAX+1.) < 0.5)
    koren->levo  = dodaj_u (koren->levo,  b);
  else
    koren->desno = dodaj_u (koren->desno, b);
  return koren;
}

Stablo dodaj_n (Stablo koren, int b) {   /* Dodavanje u neuredjeno stablo.  */
  if (! koren) {
    koren = malloc (sizeof(Cvor));
    koren->broj = b; koren->levo = koren->desno = NULL;
  } else if (rand() / (RAND_MAX+1.) < 0.5)
    koren->levo  = dodaj_u (koren->levo,  b);
  else
    koren->desno = dodaj_u (koren->desno, b);
  return koren;
}

Stablo citaj_u (int n) {              /* Citanje uredjenog stabla.       */
  Stablo koren = NULL; int i, b;
  for (i=0; i<n; i++) { scanf ("%d", &b); koren = dodaj_u (koren, b); }
  return koren;
}

Stablo citaj_n (int n) {              /* Citanje neuredjenog stabla.     */
  Stablo koren = NULL; int i, b;
  for (i=0; i<n; i++) { scanf ("%d", &b); koren = dodaj_n (koren, b); }
  return koren;
}

Stablo brisi (Stablo koren) {         /* Brisanje celog stabla.          */
```

```
  if (koren) {
    koren->levo = brisi (koren->levo); koren->desno = brisi (koren->desno);
    free (koren); koren = NULL;
  }
  return koren;
}

Stablo izost_u (Stablo koren, int b) {   /* Izostavi b iz uredjenog stabla.     */
  if (koren) {
    if      (koren->broj > b) koren->levo  = izost_u (koren->levo,  b);
    else if (koren->broj < b) koren->desno = izost_u (koren->desno, b);
    else if (koren->levo) {
      int m = max_u (koren->levo);
      koren->broj = m; koren->levo  = izost_u (koren->levo,  m);
    } else if (koren->desno) {
      int m = min_u (koren->desno);
      koren->broj = m; koren->desno = izost_u (koren->desno, m);
    } else {
      free (koren); koren = NULL;
    }
  }
  return koren;
}

Stablo izost_n (Stablo koren, int b) {   /* Izostavi b iz neuredjenog stabla.   */
  if (koren) {
    if (koren->broj == b) {
      if       (koren->levo ) {
        koren->broj  = koren->levo->broj;
        koren->levo  = izost_n (koren->levo, koren->broj);
      } else if (koren->desno) {
        koren->broj  = koren->desno->broj;
        koren->desno = izost_n (koren->desno, koren->broj);
      } else { free (koren); koren = NULL; }
    } else {
      int v = vel (koren->levo);  koren->levo  = izost_n (koren->levo,  b);
      if (v == vel (koren->levo)) koren->desno = izost_n (koren->desno, b);
    }
  }
  return koren;
}

Stablo balans_u (Stablo koren) {         /* Balansiranje uredjenog stabla.  */
  if (koren) {
    int k = vel (koren->levo) - vel (koren->desno);
    for (; k>1; k-=2) {
      koren->desno = dodaj_u (koren->desno, koren->broj);
      koren->broj  = max_u (koren->levo );
      koren->levo  = izost_u (koren->levo , koren->broj);
    }
    for (; k<-1; k+=2) {
      koren->levo  = dodaj_u (koren->levo , koren->broj);
      koren->broj  = min_u (koren->desno);
      koren->desno = izost_u (koren->desno, koren->broj);
    }
    koren->levo  = balans_u (koren->levo );
    koren->desno = balans_u (koren->desno);
```

```
  }
  return koren;
}

Stablo balans_n (Stablo koren) {         /* Balansiranje neuredjenog stabla.*/
  if (koren) {
    int k = vel (koren->levo) - vel (koren->desno);


/*POREMECAJ BALANSA NA LEVOJ STRANI */
    for (; k>1; k-=2) {

     /*1. DOPUNA DESNOG PODDRVETA KORENOM */
            koren->desno = dodaj_n (koren->desno, koren->broj);

     /*2. NOVI KOREN STABLA JE KOREN LEVOG PODDSTABLA, RAZLIKA U ODNOSU NA KANDIDATA
      ZA KOREN KOD   UREDJENOG STABLA. TAMO JE NOVI KOREN MORA BITI NAJVECI CVOR
      LEVOG PODDSTABLA*/
      koren->broj  = koren->levo ->broj;


      koren->levo  = izost_n (koren->levo , koren->broj);
    }

/*POREMECAJ BALANSA NA DESNOJ STRANI */
   for (; k<-1; k+=2) {
      koren->levo  = dodaj_n (koren->levo , koren->broj);
      koren->broj  = koren->desno->broj;
      koren->desno = izost_n (koren->desno, koren->broj);
    }
    koren->levo  = balans_n (koren->levo );
    koren->desno = balans_n (koren->desno);
  }

  return koren;
}

int moze (Stablo koren) { /* Da li moze uredjena radnja? */
  if (! uredjeno (koren)) {
    printf ("*** Stablo nije uredjeno! ***\a\n");
    return 0;
  }
  else return 1;
}

/* Primena operacije na stablo za svaki procitani broj: */
Stablo radi (Stablo (*f)(Stablo,int), Stablo koren) {
  int b; char zn;
  printf ("Brojevi?   ");
  do { scanf ("%d%c", &b, &zn); koren = (*f) (koren, b); } while (zn != '\n');
  return koren;                                  /* do kraja reda */
}
```

11. Program sa ulaza cita tekst i ispisuje broj pojavljivanja svake od reci koje su se javljale u tekstu. Radi poboljsanja efikasnosti, prilikom brojanja reci koristi se struktura podataka pogodna za leksikografsku pretragu, tj. u ovom slucaju binarno pretrazivacko drvo. Naziv datoteke se zadaje kao argument komandne linije.

ULAZ	IZLAZ
matematika ima svoju lepotu i matematika je kraljica i sluskinja dok matematika matematika je	

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define MAX 1024

typedef struct cvor {
char rec[MAX]; /* rec */
int brojac; /* broj pojavljivanja reci */
struct cvor * levi;
struct cvor * desni;
} Cvor;

/* Pomocna funkcija za kreiranje cvora. Funkcija vraca adresu novokreiranog cvora */
Cvor * napravi_cvor (char * rec)
{
/* dinamicki kreiramo cvor */
Cvor * novi = (Cvor *) malloc (sizeof(Cvor));
/* u slucaju greske ... */
if (novi == NULL)
{
fprintf (stderr,"malloc() greska\n");
exit (1);
}

/* inicijalizacija */
strcpy(novi->rec, rec);
novi->brojac = 1;
novi->levi = NULL;
novi->desni = NULL;
/* vracamo adresu novog cvora */
return novi;
}

/* Funkcija dodaje novi cvor u stablo sa datim korenom. Ukoliko rec vec postoji u stablu, uvecava dati brojac.
Cvor se kreira funkcijom napravi_cvor(). Funkcija vraca koren stabla nakon ubacivanja novog cvora. */
Cvor * dodaj_u_stablo (Cvor * koren, char *rec)
{
/* izlaz iz rekurzije: ako je stablo bilo prazno,
novi koren je upravo novi cvor */
if (koren == NULL)
return napravi_cvor (rec);
/* Ako je stablo neprazno, i koren sadrzi leksikografski manju rec od date reci, broj se umece u desno podstablo,
rekurzivnim pozivom */
if (strcmp(koren->rec, rec) < 0)
koren->desni = dodaj_u_stablo (koren->desni, rec);
```

```
/* Ako je stablo neprazno, i koren sadrzi vecu vrednost od datog broja, broj se umece u levo podstablo,
rekurzivnim pozivom */
else if (strcmp(koren->rec, rec) > 0)
koren->levi = dodaj_u_stablo (koren->levi, rec);
else
/* Ako je data rec vec u stablu, samo uvecavamo brojac. */
koren->brojac++;
/* Vracamo koren stabla */
return koren;
}

/* Funkcija pronalazi najfrekventniju rec, tj. cvor ciji brojac ima najvecu vrednost. Funkcija vraca NULL, ako
je stablo prazno, odnosno adresu cvora koji sadrzi najfrekventniju rec u suprotnom. */
Cvor * nadji_najfrekventniju (Cvor * koren)
{
Cvor *max, *max_levi, *max_desni;
/* Izlaz iz rekurzije */
if (koren == NULL) return NULL;
/* Odredjujemo najfrekventnije reci u levom i desnom podstablu */
max_levi = nadji_najfrekventniju(koren->levi);
max_desni = nadji_najfrekventniju(koren->desni);
/* Odredjujemo MAX(koren, max_levi, max_desni) */
max = koren;
if(max_levi != NULL && max_levi->brojac > max->brojac) max = max_levi;
if(max_desni != NULL && max_desni->brojac > max->brojac) max = max_desni;
/* Vracamo adresu cvora sa najvecim brojacem */
return max;
}

/* Prikazuje reci u leksikografskom poretku, kao i broj pojava svake od reci */
void prikazi_stablo(Cvor *koren)
{
if(koren == NULL) return;
prikazi_stablo(koren->levi);
printf("%s: %d\n", koren->rec, koren->brojac);
prikazi_stablo(koren->desni);
}

/* Funkcija oslobadja prostor koji je alociran za cvorove stabla. Funkcija vraca NULL, zato sto je
nakon oslobadjanja stablo prazno. */
void oslobodi_stablo (Cvor *koren)
{
/* Izlaz iz rekurzije */
if(koren == NULL) return;
oslobodi_stablo (koren->levi);
oslobodi_stablo (koren->desni);
free(koren);
}

/* Funkcija ucitava sledecu rec iz fajla i upisuje je u niz na koji pokazuje rec, maksimalne duzine max.
Funkcija vraca EOF ako nema vise reci, 0 u suprotnom. Rec je niz alfabetskih karaktera.*/
int sledeca_rec(FILE *f, char * rec, int max)
{
int c;
int i = 0;
/* Dokle god ima mesta za jos jedan karakter u stringu, i dokle god nismo stigli do kraja fajla... */
while(i < max - 1 && (c = fgetc(f)) != EOF)
```

```
{
/* Ako je slovo, ubacujemo ga u rec */
if(isalpha(c))
rec[i++] = tolower(c);
/* U suprotnom, ako smo procitali bar jedno slovo prethodno, onda imamo rec. Inace idemo na sledecu
iteraciju */
else if(i > 0)
break;
}
/* Zatvaramo string */
rec[i] = '\0';
/* Vracamo 0 ako imamo rec, EOF u suprotnom */
return i > 0 ? 0 : EOF;
}

/* test program */
int main(int argc, char **argv)
{
Cvor *koren = NULL, *max;
FILE *f;
char rec[MAX];
/* Proveravamo da li je navedeno ime datoteke */
if(argc < 2)
{
fprintf(stderr, "Morate uneti ime datoteke sa tekstom!\n");
exit(0);
}
/* Otvaramo datoteku */
if((f = fopen(argv[1], "r")) == NULL)
{
fprintf(stderr, "fopen() greska\n");
exit(1);
}
/* Ucitavamo reci iz datoteke. Rec je niz alfabetskih karaktera. */
while(sledeca_rec(f, rec, MAX) != EOF)
{
koren = dodaj_u_stablo(koren, rec);
}
/* Zatvaramo datoteku */
fclose(f);
/* Prikazujemo sve reci i brojeve njihovih pojavljivanja */
prikazi_stablo(koren);
/* Pronalazimo najfrekventniju rec */
if((max = nadji_najfrekventniju(koren)) == NULL)
printf("U tekstu nema reci!\n");
else
printf("Najcesca rec %s (pojavljuje se %d puta)\n", max->rec, max->brojac);
/* Oslobadjamo stablo */
oslobodi_stablo(koren);
return 0;
}
```

12. Napisati program koji na standardni izlaz ispisuje naziv (BEZ ATRIBUTA) najčešće korišćene etikete u datoteci ulaz.htm. Ako ima više takvih, ispisati ma koju. Koristiti uređeno binarno stablo. Pretpostaviti da je ulazna datoteka sintaksno korektna.

ULAZ
http://www.mg.edu.rs/

IZLAZ

```
Najcesca etiketa je: a sa brojem pojava 175
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAX 31

typedef struct BSP_cvor
{
char *etiketa;                  /*naziv etikete */
int broj;                       /*broj pojave etikete*/
struct BSP_cvor *levi;          /* levi ogranak */
struct BSP_cvor *desni;         /* desni ogranak */
     } drvo;

 /*FUNKCIJE */
 /* ucitava etiketu otvaraca iz datoteke u nisku s*/
int uzmi_etiketu(FILE * f, char * s);
/* ubacuje etiketu u drvo */
drvo *ubaci(drvo *, char *);
/* alokacija  cvora BSP  */
drvo *alociraj( void );
/* kreira  kopiju niske  */
char *strdpl(char *);
/*poredjenje naziva dve etikete ignorisuci razliku malih, velikih slova*/
int uporedi(char *, char *);
/* stampa drvo - infiksni poredak za BSP*/
void infiks(drvo *);
/* oslobadja zauzeti prostor za drvo */
void osloboditi( drvo *);

int max=0; /*najveci br.pojava etikete*/
char szMax_Etik[MAX]; /*najcesca etiketa*/

main()
{
  drvo *koren;          /*koren stabla pretrazivanja */
  char etiketa[MAX];   /*naziv etikete sa ulaza */
  FILE *dat;


  dat=fopen("ulaz.htm","r");
   if (dat==NULL)  exit(1);


  /*formiranje stabla od ucitanih razlicitih etiketa */
  koren = NULL;
  while( 1 )
  {  int kraj = uzmi_etiketu(dat,etiketa);
  /*dodavanje cvora u stablo  cvora, ako etiketa nije prisutna u stablu */
```

```
      if  ( strlen(etiketa)) koren = ubaci( koren, etiketa);
      if(kraj == EOF) break;
    }
     fclose(dat);


 printf("\nNajcesca etiketa je: %s sa brojem pojava %d\n", szMax_Etik, max);
   osloboditi(koren);
   return 0;
}

drvo *alociraj(void)
{ return (drvo *) malloc(sizeof(drvo)); }

drvo *ubaci( drvo *k, char *tag )
{ int test;
   if( k == NULL )
   {  /* naisao nov naziv */  k = alociraj();
              k->etiketa = strdpl(tag); k->broj=1;
      k->levi = k->desni = NULL;
   }
   else if ((test = uporedi(tag,k->etiketa))< 0 ) /* idi levo */
           k->levi = ubaci(k->levi, tag);
   else  if (test >0)                          /* idi desno */
           k->desni = ubaci(k->desni, tag);
   else /*ponovljena etiketa*/
         {k->broj++;
          if (max < k->broj) {max=k->broj; strcpy(szMax_Etik,k->etiketa); }
         }
   return k;
}

void infiks(drvo  *k)
{
    if( k != NULL )
    {  infiks( k->levi );
       printf("%s\n", k->etiketa );
       infiks( k->desni );
    }
}

int uzmi_etiketu(FILE *f,char s[])
{  char c; /*tekuci ulazni karakter */
    int i = 0; /*brojac*/
      if ( (c = fgetc(f)) == '<')
               while (isalnum(c = fgetc(f))   )  s[i++] = c;
      s[i] = '\0';
      if( c==EOF ) return EOF;
      return i;
}

/*poredi etikete e1, e2 tako da poredjenje bude case insensitive*/
int uporedi(char *e1, char *e2)
{ char pom1[MAX], pom2[MAX];
   int i;
   for(i=0;e1[i]!='\0';i++)  pom1[i]=tolower(e1[i]);
   pom1[i]='\0';
```

```
  for(i=0;e2[i]!='\0';i++)  pom2[i]=tolower(e2[i]);
  pom2[i]='\0';
  return strcmp (pom1, pom2);
}


char *strdpl(char *s)
{   char *pom;
      pom = (char *) malloc(strlen(s) + 1 );
      if( pom != NULL ) strcpy(pom,s);
      return pom;
 }
void osloboditi( drvo *k)
{   if (k->levi)  osloboditi (k->levi);
    if (k->desni)  osloboditi (k->desni);
   free (k);
}
```

13. Napisati program koji će iz datoteke prva.htm prepisati nazive međusobno različitih otvarajućih etiketa (bez atributa) u binarno stablo pretrage i potom:
a) štampati sadržaj drveta u infiksnom poretku u datoteci izlaz.txt, ako je kao parametar komandne linije zadata opcija -t
b) na standardni izlaz ispisati ukupan broj čvorova drveta, ako je kao parametar komandne linije zadata opcija -u
Obezbediti da program radi i kada su prisutni svi opcioni argumenti. Pretpostaviti da naziv otvarajuće etikete nije duži od 20 karaktera.
**Primer: <P align= center > <IMG src= ul.gif >

 Nova slika </p>**
Otvarajuće etikete bez atributa su: BR, IMG, P

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>

#define MAXREC 21
#define TRUE 1
#define FALSE 0

typedef struct drvo_tag
        {
                char *rec;                  /* rec  teksta */
                struct drvo_tag *levo;      /* leva grana */
                struct drvo_tag *desno;     /* desna grana */
        } drvo;

/* Funkcije */
drvo *addtree(drvo *, char *);    /* ubacuje rec u drvo */
void treeprint(drvo *);           /* stampa drvo – infiksni poredak za BSP*/
int uzmi_tag(FILE *f, char *s, int k);   /* ucitava tagove (do k karaktera) iz datoteke u nisku s*/
drvo *talloc( void );   /* alokacija  cvora BSP  */
char *strdpl(char *s);     /* kreira  kopiju niske s */
void osloboditi( drvo *k);   /* oslobadja zauzeti prostor za drvo */
int brcvorova(drvo *k);    /* vraca broj cvorova stabla */

/*glavni program*/
main(int argc, char *argv[])
{
```

```
   drvo *koren;         /*koren stabla pretrazivanja */
   char rec[MAXREC];    /*sadrzaj reci sa ulaza */
   FILE *ulaz;
   int i,j;          /* brojacke promenljive */
   int node=FALSE, printt=FALSE;  /*indikatori postojanja opcija u komandnoj liniji */

   ulaz=fopen("prva.htm","rt");
    if (ulaz==NULL)  exit(1);

/*prosledjivanje argumenata komandne linije */
  for (i = 1; i < argc && argv[i][0] == '-'; i++)
{   /*prosledjivanje opcija ili niti jedne korektne opcije  ili od jedne do cetiri opcije*/
   for (j=1; argv[i][j] !='\0'; j++)
      {  /* test prisustva opcionih argumenata -n, -l, -d,-p  */
         switch (argv[i][j])
              { case 'u':  node = TRUE;    break;
                case 't':  printt=TRUE;      break;
                default:   fprintf(stderr, "Nekorektan opcioni argument '%s'\n",       argv[i]);
                           return EXIT_FAILURE;
              }
      }
 }
/*ucitavanje reci ciji broj pojavljivanja se broji */
   koren = NULL;
   while( 1 )
   {
          int kraj = uzmi_tag(ulaz,rec, MAXREC);
     /*dodavanje cvora u stablo  cvora, ako rec ne postoji u stablu */
          if  ( strlen(rec)) koren = addtree( koren, rec);
          if(kraj == EOF) break;   /*ucitavanje se vrsi do markera kraja */
   }
    fclose(ulaz);

      /*stampanje broja cvorova stabla */
    if (node)  printf("\nCvorova: %d", brcvorova(koren));

       /*stampanje sadrzaja stabla*/
    if (printt)   treeprint(koren);


   /*oslobadjanje zauzetog prostora za  stablo pretrage */
   osloboditi(koren);
   return 0;
}

/* addtree - dodaje cvor sa tekstom na koji pokazuje w, na ili ispod p u drvetu*/
drvo *addtree( drvo *p, char *w )
{
   int cond;
   if( p == NULL )       /* naisla nova rec */
   {
          p = talloc();                 p->rec = strdpl(w);
          p->levo = p->desno = NULL;
   }
   else if ((cond = strcmp(w,p->rec))< 0 )   /*manja rec => levi ogranak*/
          p->levo = addtree(p->levo, w);
   else  if (cond >0)                    /*vece =>desni ogranak*/
```

```
        p->desno = addtree(p->desno, w);
  return p;
}

void treeprint(drvo  *p)  /* treeprint - rekurzivno stampanje drveta*/
{
   if( p != NULL )
   {
                treeprint( p->levo );
                printf("%s\n", p->rec );
                treeprint( p->desno );
   }
}

int uzmi_tag(FILE *f,char s[], int lim)
{           char c; int i = 0;
        /* preskociti  sve znake do znaka <*/
        while( ( (c =fgetc(f))  != '<') && c!=EOF);
        if( c==EOF ) {s[0] = '\0'; return EOF;} /*  prazna rec i vratiti EOF */

        /* ucitati ostatak etikete; OBRATITI PAZNJU DA HTML JE CASE INSENSITIVE, A DA f-ja strcmp
           pravi razliku u velicini slova, tj. <table> i <TaBLE> su 2 pojave iste etikete*/
        while( (c = fgetc(f)) != EOF && isalnum(c) && i < lim)
                {s[i] = toupper(c); i++; }


        s[++i] =  '\0';    /* zavrsiti  nisku */

        if( c==EOF ) return EOF;
        return i;
}


int brcvorova(drvo *c)
{
  if(c == NULL)
    return 0;
  else      return 1+brcvorova(c->levo)+brcvorova(c->desno);
}

drvo *talloc(void)
{ return (drvo *) malloc(sizeof(drvo)); }
char *strdpl(char *s)   /* pravi se kopija niske s */
{   char *p;      p = (char *) malloc(strlen(s) + 1 );
     if( p != NULL ) strcpy(p,s);
    return p;
  }
void osloboditi( drvo *k)
{ /*rekurzivno se oslobadja levo i desno podstablo korena zadatog stabla */
  if (k->levo)  osloboditi (k->levo);
  if (k->desno)  osloboditi (k->desno);
  free (k);   /*brisanje cvora koji predstavlja koren zadatog stabla */
}
```

14. Napisati program koji implementira telefonski imenik. U fajlu, koji se da zadaje kao argument komandne linije, nalazi se niz linija oblika:
Ime Prezime ***/***-****

tj. za svaku osobu je naveden broj telefona (pretpostaviti da je datoteka ispravna). Kada se program pokrene, korisnik treba da unese ime i prezime, a program treba da pronade broj telefona te osobe, ili da obavesti korisnika da se osoba ne nalazi u imeniku. Ovaj postupak treba ponavljati dokle god korisnik ne unese komandu za kraj rada. Informacije o brojevima telefona uneti u mapu koja je implementirana preko binarnog stabla pretrage.

15. Napisati biblioteku koja implementira rad sa skupovima celih brojeva koriscenjem binarnih stabala. Obezbediti funkciju konstrukcije skupa na osnovu datog niza, provere da li element pripada skupu, proveru da li je jedan skup podskup drugog, funkciju za dodavanje elementa skupu, funkcije za pronalaˇzenje unije, preseka i razlike dva skupa, funkciju za ispis skupa, itd.

16. Napisati program koji broji pojavljivanja različitih etiketa u HTML datoteci (koristiti binarno pretraživačko stablo).

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
/* Makimalna duzina imena etikete */
#define MAX_ETIKETA 32
/* Definisemo stanja prilikom citanja html datoteke */
#define VAN_ETIKETE 1
#define U_ETIKETI 2
/* Struktura koja predstavlja cvor stabla */
typedef struct cvor {
char etiketa[MAX_ETIKETA]; /* Ime etikete */
int brojac; /* Brojac pojavljivanja etkete u tekstu */
struct cvor *levi; /* Pokazivaci na levo */
struct cvor *desni; /* i desno podstablo */
} Cvor;
/* Funkcija dodaje ime html etikete u binarno stablo,
u rastucem leksikografskom poretku. Funkcija vraca
koren stabla nakon modifikacije. U slucaju da je
etiketa sa istim imenom vec u stablu, uvecava se
brojac pojavljivanja. */
Cvor * dodaj_leksikografski(Cvor *koren, char *etiketa)
{
int cmp;
/* Izlaz iz rekurzije */
if(koren == NULL)
{
if((koren = malloc(sizeof(Cvor))) == NULL)
{
fprintf(stderr,"malloc() greska\n");
exit(1);
}
/* Ime etikete se kopira u novi cvor, a brojac se inicijalizuje na 1 (prvo pojavljivanje) */
strcpy(koren->etiketa, etiketa);
koren->brojac = 1;
koren->levi = NULL;
koren->desni = NULL;
return koren;
}
/* Rekurzivni pozivi */
if((cmp = strcmp(koren->etiketa,etiketa)) < 0)
koren->desni = dodaj_leksikografski(koren->desni, etiketa);
else if(cmp > 0)
koren->levi = dodaj_leksikografski(koren->levi, etiketa);
```

```
else
koren->brojac++; /* uvecanje brojaca za vec prisutne etikete */
return koren;
}
/* Funkcija cita ulazni fajl i iz njega izdvaja sve otvorene html etikete (npr. <html>, <br> itd, ali ne
i </html>, </body> itd.) i njihova imena (bez znakova (< i > kao i bez eventualnih atributa) ubacuje u stablo
u leksikografskom poretku. Tom prilikom ce se prebrojati i pojavljivanja svake od etiketa. Funkcija vraca pokazivac
na koren kreiranog stabla. */
Cvor * ucitaj(FILE *f)
{
Cvor * koren = NULL;
int c;
int stanje = VAN_ETIKETE;
int i;
char etiketa[MAX_ETIKETA];
/* NAPOMENA: prilikom izdvajanja etiketa pretpostavlja se da je html datoteka sintaksno ispravna, kao i da nakon
znaka '<' nema belina. Zato se otvorene etikete mogu lako prepoznati tako sto pronadjemo znak '<' i nakon toga
izdvojimo sva slova koja slede. Dodatno etikete mogu da sadrze i cifre kao na primer <h1>,<h2>... u kom slucaju
koristimo funkciju isalnum za proveru karaktera. Ako postoji neprazan niz slova nakon znaka '<', tada je to upravo
ime etikete. Ako ne, tada je verovatno u pitanju zatvorena etiketa, npr. </html>, pa je ignorisemo. */
while((c = fgetc(f)) != EOF)
{
switch(stanje)
{
/* u ovom stanju trazimo prvu sledecu pojavu znaka '<' */
case VAN_ETIKETE:
if(c == '<')
{
stanje = U_ETIKETI; /* sada smo u etiketi */
i = 0;
}
break;
/* u ovom stanju citamo slova koja slede, i nakon toga ubacujemo procitanu etiketu u stablo */
case U_ETIKETI:
/* Ako je slovo, i nismo prekoracili duzinu niza dodajemo slovo u niz */
if(isalpha(c) && i < MAX_ETIKETA - 1)
etiketa[i++] = c;
/* u suprotnom se vracamo u stanje VAN_ETIKETE */
else {
stanje = VAN_ETIKETE;
/* Ako je niz slova nakon '<' bio neprazan... */
if(i > 0)
{
etiketa[i]='\0';
/* ubacujemo procitanu etiketu u stablo */
koren = dodaj_leksikografski(koren, etiketa);
}
}
break;
}
}
return koren;
}
/* Funkcija dodaje u stablo etiketu ciji je broj pojavljivanja poznat (broj), i to u opadajucem poretku po broju
pojavljivanja. Funkcija vraca pokazivac na koren tako modifikovanog stabla */
Cvor * dodaj_po_broju(Cvor * koren, char * etiketa, int broj)
{
```

```
/* Izlaz iz rekurzije */
if(koren == NULL)
{
if((koren = malloc(sizeof(Cvor))) == NULL)
{
   fprintf(stderr,"malloc() greska\n"); exit(1);
}
/* Kopiramo u novi cvor ime i broj pojavljivanja etikete */
strcpy(koren->etiketa, etiketa);
koren->brojac = broj;
koren->levi = NULL;
koren->desni = NULL;
return koren;
}
/* NAPOMENA: s obzirom da dve ili vise etiketa mogu imati isti broj pojavljivanja, etiketa se mora dodavati u
stablo, cak i ako ima isti broj pojavljivanja sa korenom. Zato cemo u levo podstablo dodavati etikete koje imaju veci
broj pojavljivanja od korena, dok cemo u desno podstablo dodavati etikete koje imaju manji ili jednak broj
pojavljivanja od korena. */
/* Rekurzivni pozivi */
if(koren->brojac >= broj)
koren->desni = dodaj_po_broju(koren->desni, etiketa, broj);
else if(koren->brojac < broj)
koren->levi = dodaj_po_broju(koren->levi, etiketa, broj);
return koren;
}

/* Funkcija pretpostavlja da je novo stablo ili prazno, ili uredjeno prema opadajucem poretku broja pojavljavanja
etiketa. Funkcija u takvo stablo rekurzivno dodaje sve etikete iz starog stabla, i vraca koren na tako modifikovano
novo stablo. */
Cvor * resortiraj_stablo(Cvor * staro, Cvor * novo)
{
/* Izlaz iz rekurzije - nemamo sta da dodamo u novo stablo */
if(staro == NULL)
return novo;
/* Dodajemo etiketu iz korena starog stabla u novo stablo,
sortirano opadajuce prema broju pojavljivanja */
novo = dodaj_po_broju(novo, staro->etiketa, staro->brojac);
/* Rekurzivno dodajemo u novo stablo sve cvorove iz levog i desnog podstabla starog stabla. */
novo = resortiraj_stablo(staro->levi, novo);
novo = resortiraj_stablo(staro->desni, novo);
return novo;
}

/* Ispis stabla - s leva na desno */
void ispisi_stablo(Cvor * koren)
{
if(koren == NULL)
return;
ispisi_stablo(koren->levi);
printf("%s: %d\n", koren->etiketa, koren->brojac);
ispisi_stablo(koren->desni);
}

/* Oslobadjanje dinamicki alociranog prostora */
void oslobodi_stablo(Cvor *koren)
{
if(koren == NULL)
```

```
return;
oslobodi_stablo(koren->levi);
oslobodi_stablo(koren->desni);
free(koren);
}

/* glavni program */
int main(int argc, char **argv)
{
FILE *in = NULL; /* Ulazni fajl */
Cvor * koren = NULL; /* Stablo u leksikografskom poretku */
Cvor * resortirano = NULL; /* stablo u poretku po broju pojavljivanja */
/* Mora se zadati bar ime ulaznog fajla na komandnoj liniji. Opciono se kao drugi argument moze zadati opcija "-b"
sto dovodi do ispisa po broju pojavljivanja (umesto leksikografski) */
if(argc < 2)
{
fprintf(stderr, "koriscenje: %s ime_fajla [-b]\n", argv[0]);
exit(1);
}
/* Otvaramo fajl */
if((in = fopen(argv[1], "r")) == NULL)
{
fprintf(stderr, "fopen() greska\n");
exit(1);
}
/* Formiramo leksikografsko stablo etiketa */
koren = ucitaj(in);

/* Ako je korisnik zadao "-b" kao drugi argument, tada se vrsi resortiranje, i ispisuje izvestaj sortiran
opadajuce po broju pojavljivanja */
if(argc == 3 && strcmp(argv[2], "-b") == 0)
{
resortirano = resortiraj_stablo(koren, resortirano);
ispisi_stablo(resortirano);
}

/* U suprotnom se samo ispisuje izvestaj u leksikografskom poretku */
else
ispisi_stablo(koren);
/* Oslobadjamo stabla */
oslobodi_stablo(koren);
oslobodi_stablo(resortirano);
return 0;
}
```

17.

Argumenti komandne linije su nazivi dve datoteke koje u svakoj liniji sadrže po jednu nisku sa ne više od 80 karaktera. Napisati program koji na standardni izlaz ispisuje sve niske prve datoteke koje nisu sadržane u drugoj datoteci. Zadatak realizovati korišćenjem binarnog stabla pretrage.

```
/*Analiza rešenja zadatka: postaviti niske 2. datoteke u
binarno stablo pretrage i za svaku reč 1. datoteke proveriti da li se nalazi u 2. datoteci */

#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
```

```
#define MAXREC 81

typedef struct drvo_tag
    {
        char *rec;                  /* pokazuje na rec  teksta */
        struct drvo_tag *levo;      /* leva grana */
        struct drvo_tag *desno;     /* desna grana */
    } drvo;
/* Prototipovi funkcija */
drvo *addtree(drvo *, char *);
drvo* nadji (char *, drvo *);
drvo *talloc( void );
char *strdpl(char *);
void oslobodi( drvo *k);


main(int argc, char ** argv)
{
  drvo *koren;        /*koren stabla pretrazivanja */
  char rec[MAXREC];   /*sadrzaj reci iz datoteke */
  FILE *ul1, *ul2;


ul1=fopen(argv[1], "rt");
ul2=fopen(argv[2], "rt");

  /*ucitavanje reci druge datoteke u binarno pretrazivacko stablo */
  koren = NULL;
  while( fgets(rec, MAXREC,ul2) )
    /*dodavanje novog cvora u stablo, ako vec rec nije ubacena u stablo */
      koren = addtree( koren, rec);


    /*stampa reci iz 1. datoteke koje se ne sadrze u 2. datoteci*/
  while( fgets(rec, MAXREC,ul1) )
       if (nadji(rec,koren) == NULL) printf("\n%s", rec);


  /*oslobadjanje zauzetog prostora za  stablo pretrage */
  oslobodi(koren);
  fclose(ul1); fclose(ul2);
  return 0;
}


/* addtree - dodaje cvor sa tekstom na koji pokazuje w, na ili ispod p u drvetu*/
drvo *addtree( drvo *p, char *w )
{
  int cond;
  if( p == NULL )       /* naisla nova rec */
  {
    p = talloc();
        p->rec = strdpl(w);
        p->levo = p->desno = NULL;
  }
  else  if ( (cond = strcmp(w,p->rec)) < 0 ) /* manje => levi  ogranak  */
```

```
    p->levo = addtree(p->levo, w);
  else if (cond > 0)                    /*vece =>desni ogranak*/
    p->desno = addtree(p->desno, w);
  return p;
}

drvo* nadji (char *rec, drvo *koren) /*vraca NULL ako se rec ne nalazi u stablu*/
{   int cond;
  if (koren==NULL) return NULL;
  if( (cond=(strcmp(koren->rec, rec))==0)) return koren;
  else if (cond <0) return (rec, koren->levo);
  else return (rec, koren->desno);
}



/* talloc pravi jedan cvor drveta */
drvo *talloc(void)
{ return (drvo *) malloc(sizeof(drvo)); }

char *strdpl(char *s)   /* pravi se kopija niske s */
{
  char *p;
  p = (char *) malloc(strlen(s) + 1 );
  if( p != NULL ) strcpy(p,s);
  return p;
  /* u  postoji standardna funkcija "strdup" koja obavlja navedene operacije*/
}

 void osloboditi( drvo *k)
{ /*rekurzivno se oslobadja levo i desno podstablo korena zadatog stabla */
  if (k->levo)  osloboditi (k->levo);
  if (k->desno)  osloboditi (k->desno);
  free (k);   /*brisanje cvora koji predstavlja koren zadatog stabla */}
```

18. Napisati program koji čita datoteku ulaz.txt i na standardni izlaz ispisuje sve različite reči (niske do 20 karaktera koje pocinju slovom i ne sadrže blanko, tab, znak prelaza u novi red) koje se pojavljuju u datoteci, sa brojevima redova u
kojima se pojavljuju. Reči ispisati u rastućem leksikografskom poretku.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAX  21

typedef struct drvo_CV drvo;     /* BSP uredjeno po recima */
typedef struct redovi_CV redovi; /* lista rednih brojeva linija u kojima se pojavljuje jedna rec*/
struct drvo_CV
{    /* za svaku rec  cuva se lista pojavnih redova r */
  char rec[MAX];
  redovi *r;
  drvo *levo,*desno;
};

struct redovi_CV
{     /* cuvaju se redni brojevi linija pojava jedne reci */
```

```
   int br;       /* redni broj linije */
   redovi *sl;
};

char getWord(FILE *f, char *s);  /* izdvaja rec s iz ulazne datoteke i vraca tekuci karakter ulaza*/
drvo * addTree(drvo*d, char *s, int br); /* unosi rec s koja se pojavila u redu br u BSP d*/
void printTree(drvo *);   /* stampa reci (inorder) i redne brojeve linija u kojima su se pojavile bar jednom */
void  freeTree(drvo *);       /* oslobadja prostor */

 main()
{
  drvo *koren;
  char rec[MAX];  /* tekuca rec sa ulaza */
  int brlinije;   /* redni broj tekuce linije */
  char c;        /* tekuci karakter sa ulaza */
  FILE *ul;

    /* inicijalizacije */
    koren = NULL;
    brlinije = 1;
    ul = fopen("ulaz.txt","rt");

   /* citanje sadrzaja datoteke i formiranje BSP-a*/
while(!feof(ul))
 {
   c = getWord(ul, rec);
   if(strlen(rec) > 0)   koren=addTree(koren, rec, brlinije);
   if(c=='\n') brlinije++;   /* postavlja redni broj tekuceg reda */
 }
    fclose(ul);
    printTree(koren);
    freeTree(koren);

   return 0;

}

char getWord(FILE *fin, char *word)
{ char c;    /* tekuci karakter ulazne datotke */
  int k;     /* tekuca pozicija niske word koja se formira */

   /*ignorisati karaktere kojima ne pocinje rec ili ne ulaze u sastav reci*/
   while(!isalpha(c = fgetc(fin)) && c!='\n' && !feof(fin)) ;
    /*formiranje niske word */
    k = 0;
    while(c!=' ' && c!= '\t' && c!='\n' && !feof(fin))
      {
         word[k++] = c;
         c = fgetc(fin);
      }
     word[k] = '\0';
    return c;
}  /* izdvojrec */


drvo * addTree(drvo *d, char *w, int br)
{ redovi *tek, *pom;
```

```
  if(d == NULL)   /* rec se prvi put pojavila */
   {  d = (drvo *)malloc(sizeof(drvo));
      strcpy(d->rec, w);
      d->levo = NULL;
      d->desno = NULL;
      /*ubacivanje u listu rednog broja tekuceg reda u kom se rec pojavila*/
      d->r = (redovi *)malloc(sizeof(redovi));
      d->r->br = br;
      d->r->sl = NULL;
   }
  else  /*rec se opet pojavila*/
    if(strcmp(w, d->rec) == 0)
     {
            for(tek=d->r; tek->sl!=NULL; tek=tek->sl)
                  if(br == tek->br) break; /*ako se rec vec pojavila u istom redu, njegov redni broj se ne ubacuje u listu */
            if(br != tek->br)
                  {                 /* ako se rec 1. put pojavila u tekucem redu,
                    onda  dodati taj red */
                     tek->sl = (redovi *)malloc(sizeof(redovi));
                     tek = tek->sl;
             tek->br = br;
             tek->sl = NULL;
                    }
     }
    else if(strcmp(w, d->rec) < 0)  d->levo=addTree( d->levo, w, br);
              else          d->desno=addTree(d->desno, w, br);

  return d;

}

void freeTree(drvo *d)
{  if(d != NULL)
   {     freeTree(d->levo);
         freeTree(d->desno);
         free(d);
   }
}

void printTree(drvo *d)
{ redovi *tekuci;

  if(d != NULL)
   { printTree(d->levo);
     /* stampati sadrzaj reci*/
     printf("%s ",d->rec);
     /* stampati listu rednih brojeva linija u kojima se pojavila rec bar jednom*/
     for(tekuci=d->r; tekuci!=NULL; tekuci=tekuci->sl)
         printf("%d ",tekuci->br);
     printf("\n");

     printTree(d->desno);
   }
}
```

MAPA

19. Mapa je apstraktna struktura podataka koja u sebi sadrzi parove oblika (kljuc, vrednost). Pri tom su i kljuc i vrednost unapred odredjenog (ne obavezno istog) tipa (na primer, kljuc nam moze biti string koji predstavlja ime studenta, a vrednost je npr. broj koji predstavlja njegov prosek). Operacije koje mapa mora
efikasno da podrzava su:
_ dodavanje para (kljuc, vrednost) u mapu
_ brisanje para (kljuc, vrednost) iz mape
_ pronalazenje vrednosti za dati kljuc
_ promena vrednosti za dati kljuc

Jedan od najcescih nacina implementacije mape je preko binarnog stabla pretrage.
Svaki cvor ovog stabla sadrzace odgovarajuci par (kljuc, vrednost), tj. svaki cvor ce sadrzati dva podatka. Pri tom ce poredak u stablu biti odredjen poretkom koji je definisan nad kljucevima. Ovim se postize da se pretraga po kljucu moze obaviti na uobicajen nacin, kao i sve ostale navedene operacije.
Jasno je da ovako implementirana mapa nece efikasno podrzavati operaciju pretrage po vrednosti (npr. naci sve kljuceve koji imaju datu vrednost), zato sto poredak
u stablu nije ni na koji na_cin u vezi sa poretkom medju vrednostima. Medjutim, u mapi se najcesce i ne zahteva takva operacija.
Sledi primer koji demonstrira implementaciju mapa pomocu binarnog stabla.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 1024
/* Struktura koja predstavlja cvor stabla */
typedef struct cvor {
char naziv[MAX];
int cena;
struct cvor * levi;
struct cvor * desni;
} Cvor;
/* Funkcija kreira novi cvor i vraca njegovu adresu */
Cvor * napravi_cvor (char * naziv, int cena)
{
/* dinamicki kreiramo cvor */
Cvor * novi = (Cvor *) malloc (sizeof(Cvor));
/* u slucaju greske ... */
if (novi == NULL)
{
fprintf (stderr,"malloc() greska\n");
exit (1);
}

/* inicijalizacija */
strcpy(novi->naziv, naziv);
novi->cena = cena;
novi->levi = NULL;
novi->desni = NULL;
/* vracamo adresu novog cvora */
return novi;
}
/* Funkcija dodaje novi cvor u stablo sa datim korenom. U cvor se upisuje vrednost (naziv, cena). Ukoliko naziv vec postoji u stablu, tada se azurira njegova cena. Cvor se kreira funkcijom napravi_cvor(). Funkcija
vraca koren stabla nakon ubacivanja novog cvora. */
Cvor * dodaj_u_stablo (Cvor * koren, char * naziv, int cena)
{
/* izlaz iz rekurzije: ako je stablo bilo prazno,
novi koren je upravo novi cvor */
```

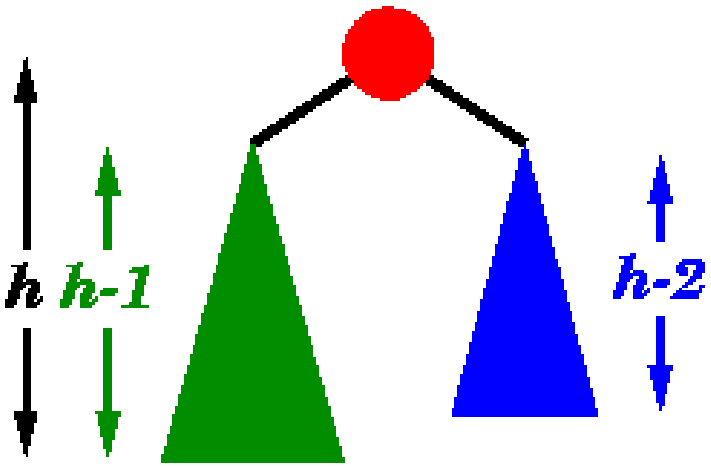
```
if (koren == NULL)
return napravi_cvor (naziv, cena);
/* Ako je stablo neprazno, i koren sadrzi naziv koji je leksikografski manji od datog naziva, vrednost se umece
u desno podstablo, rekurzivnim pozivom */
if (strcmp(koren->naziv, naziv) < 0)
koren->desni = dodaj_u_stablo (koren->desni, naziv, cena);
/* Ako je stablo neprazno, i koren sadrzi naziv koji je
leksikografski veci od datog naziva, vrednost se umece
u levo podstablo, rekurzivnim pozivom */
else if (strcmp(koren->naziv, naziv) > 0)
koren->levi = dodaj_u_stablo (koren->levi, naziv, cena);
/* Ako je naziv korena jednak nazivu koja se umece, tada se samo
azurira cena. */
else
koren->cena = cena;
/* Vracamo koren stabla */
return koren;
}

/* Funkcija pretrazuje binarno stablo. Ukoliko pronadje cvor sa vrednoscu naziva koja je jednaka
datom nazivu, vraca adresu tog cvora. U suprotnom vraca NULL */
Cvor * pretrazi_stablo (Cvor * koren, char * naziv)
{
/* Izlaz iz rekurzije: ako je stablo prazno,
tada trazeni broj nije u stablu */
if (koren == NULL) return NULL;
/* Ako je stablo neprazno, tada se pretrazivanje nastavlja u levom ili desnom podstablu, u
zavisnosti od toga da li je trazeni naziv respektivno manji ili veci od vrednosti naziva
korena. Ukoliko je pak trazeni naziv jednak nazivu korena, tada se vraca adresa korena. */
if (strcmp(koren->naziv, naziv) < 0)
return pretrazi_stablo (koren->desni, naziv);
else if (strcmp(koren->naziv, naziv) > 0)
return pretrazi_stablo (koren->levi, naziv);
else
return koren;
}
/* cvor liste */
typedef struct cvor_liste {
char naziv[MAX];
int cena;
struct cvor_liste * sledeci;
} Cvor_liste;
/* Pomocna funkcija koja kreira cvor liste */
Cvor_liste * napravi_cvor_liste(char * naziv, int cena)
{
Cvor_liste * novi;
if((novi = malloc(sizeof(Cvor_liste))) == NULL)
{
fprintf(stderr, "malloc() greska\n");
exit(1);
}

strcpy(novi->naziv, naziv);
novi->cena = cena;
novi->sledeci = NULL;
return novi;
}
```

```
/* Funkcija pronalazi sve nazive cija je cena manja ili jednaka od date, i formira listu koja sadrzi nadjene parove
(naziv, cena) u leksikogra-fskom poretku po nazivima. Prilikom pocetnog poziva, treci argument
treba da bude NULL. Funkcija obilazi stablo sa desna u levo, kako bi se prilikom dodavanja na pocetak liste poslednji
dodao onaj koji je leksikografski najmanji (tj. on ce biti na pocetku). */
Cvor_liste * pronadji_manje (Cvor * koren, int cena, Cvor_liste * glava)
{
if(koren == NULL)
return glava;
/* Dodajemo na pocetak liste sve cvorove desnog podstabla cija je cena manja od date. */
glava = pronadji_manje(koren->desni, cena, glava);
/* Dodajemo koren u listu, ako mu je cena manja od date */
if(koren->cena <= cena)
{
Cvor_liste * novi = napravi_cvor_liste(koren->naziv, koren->cena);
novi->sledeci = glava;
glava = novi;
}
/* Dodajemo na pocetak liste sve cvorove levog podstabla cija je cena manja od date */
glava = pronadji_manje(koren->levi, cena, glava);
/* Vracamo glavu liste nakon svih modifikacija */
return glava;
}
/* Funkcija prikazuje listu */
void prikazi_listu(Cvor_liste * glava)
{
if(glava == NULL)  return;
printf("%s = %d\n", glava->naziv, glava->cena);
prikazi_listu(glava->sledeci);
}
/* Funkcija oslobadja listu */
Cvor_liste * oslobodi_listu(Cvor_liste * glava)
{
if(glava == NULL)
return NULL;
glava->sledeci = oslobodi_listu(glava->sledeci);
free(glava);
return NULL;
}

/* Funkcija oslobadja stablo. */
Cvor * oslobodi_stablo (Cvor *koren)
{
if(koren == NULL)
return NULL;
koren->levi = oslobodi_stablo (koren->levi);
koren->desni = oslobodi_stablo (koren->desni);
free(koren);
return NULL;
}
/* test program */
int main(int argc, char ** argv)
{
Cvor * koren = NULL, * pomocni;
Cvor_liste * glava = NULL;
FILE *f;
char naziv[MAX];
```

```
int cena;
/* Proveravamo da li je navedeno ime datoteke */
if(argc < 2)
{
fprintf(stderr, "Morate navesti ime datoteke!\n");
exit(0);
}

/* Otvaramo datoteku */
if((f = fopen(argv[1], "r")) == NULL)
{
fprintf(stderr, "fopen() greska\n");
exit(1);
}
/* Ubacujemo proizvode u stablo */
while(fscanf(f, "%s%d", naziv, &cena) == 2)
{
koren = dodaj_u_stablo(koren, naziv, cena);
}
fclose(f);

/* Testiranje pretrage po nazivu (efikasna operacija) */
printf("Uneti naziv proizvoda koji vas zanima: ");
scanf("%s", naziv);
if((pomocni = pretrazi_stablo(koren, naziv)) == NULL)
printf("Trazeni proizvod ne postoji\n");
else
printf("Cena trazenog proizvoda je: %d\n", pomocni->cena);

/* Testiranje pretrage po ceni (neefikasno) */
printf("Unesite maksimalnu cenu: ");
scanf("%d", &cena);
glava = pronadji_manje(koren, cena, NULL);
prikazi_listu(glava);

/* Oslobadjanje memorije */
glava = oslobodi_listu(glava);
koren = oslobodi_stablo(koren);
return 0;
}
```

20.
Napisati program koji implementira podsetnik za rodendane. U fajlu se nalazi niz linija oblika:
Ime Prezime DD. MM. YYYY.
tj. za svaku osobu je naveden datum rodenja. Kada se program pokrene, korisnik treba da unese datum, a program treba da pronade osobu ˇciji je rodendan najbliˇzi zadatom datumu. Ovaj postupak treba ponavljati dokle god korisnik ne unese komandu za kraj rada. Informacije o rodendanima uneti u mapu koja je implementirana preko binarnog stabla pretrage.

3. Visinski balansirano stablo pretrage
http://poincare.matf.bg.ac.rs/~jelenagr/AIDA2/cas3.pdf



21. AVL-stablo je binarno stablo pretrage kod koga apsolutna razlika visina levog i desnog podstabla svakog elementa nije veća od jedan. Napisati funkciju koja za dato binarno stablo pretrage proverava da li je AVL. Napisati potom i program koji testira ovu funkciju.

```
#include<stdio.h>
#include<stdlib.h>
typedef struct cvor{
 int vrednost;
 struct cvor *levi;
 struct cvor *desni;
} Cvor;

/* pratece funkcije za rad da binarnim stablom. */
Cvor *napravi_cvor(int broj){
 Cvor *novi=malloc(sizeof(Cvor));
 if(novi==NULL){
 printf("Greska pri alokaciji\n");
 exit(1);
 }
 novi->vrednost=broj;
 novi->levi=NULL;
 novi->desni=NULL;
 return novi;
}

Cvor *dodaj_u_stablo(Cvor *koren, int broj){
 if(koren==NULL)
 return napravi_cvor(broj);
 if(broj<koren->vrednost)
 koren->levi=dodaj_u_stablo(koren->levi, broj);
 else if(broj>koren->vrednost)
 koren->desni=dodaj_u_stablo(koren->desni, broj);
 return koren;
}

Cvor *pretrazi_stablo(Cvor *koren, int broj){
 if(koren==NULL)
 return NULL;
 if(broj<koren->vrednost)
 return pretrazi_stablo(koren->levi, broj);
 else if(broj>koren->vrednost)
 return pretrazi_stablo(koren->desni, broj); return koren;
```

```
}

void prikazi_stablo(Cvor *koren){
 if(koren==NULL)
 return;
 prikazi_stablo(koren->levi);
 printf("%d ", koren->vrednost);
 prikazi_stablo(koren->desni);
}

void oslobodi_stablo(Cvor *koren){
 if(koren==NULL)
 return;
 oslobodi_stablo(koren->levi);
 oslobodi_stablo(koren->desni);
 free(koren);
}

/* pomocna funkcija koja računa visinu stabla */
int visina(Cvor *koren){
 int l, d;
 if(koren==NULL)
 return -1;
 l=visina(koren->levi);
 d=visina(koren->desni);
 return (l>d)? l+1: d+1;
}

void koeficijenti_ravnoteze(Cvor *koren){
 if(koren==NULL)
 return;
 koeficijenti_ravnoteze(koren->levi);
 printf("Cvor %d, koeficijent_ravnoteze: %d\n", koren->vrednost,
 visina(koren->levi)-visina(koren->desni));
 koeficijenti_ravnoteze(koren->desni);
}

/* TRAZENA FUNKCIJA */
/* AVL stablo: svaki cvor ima koef. ravnoteze 1, 0 ili -1 */
int AVL(Cvor *koren){
 int d;
 if(koren==NULL)
 return 1;
 d=visina(koren->levi)-visina(koren->desni);
 return (d==0 || d==1 || d==-1) && AVL(koren->levi) && AVL(koren->desni);}

int main(){
 Cvor *koren=NULL;
 int broj;
 Cvor *sledb;

 printf("Unesite sledeci broj (Ctrl+D u Linux-u za kraj)\n");
 while(scanf("%d", &broj)==1){
 koren=dodaj_u_stablo(koren, broj);
 printf("Unesite sledeci broj (Ctrl+D u Linux-u za kraj)\n");
  }
```

```
 printf("Uneto je stablo: ");
 prikazi_stablo(koren);
 printf("\n");

 koeficijenti_ravnoteze(koren);

 if(AVL(koren)) printf("JESTE AVL\n");
 else   printf("nije AVL\n");

 oslobodi_stablo(koren);
 return 0;
}
```

5. Fenwick stablo

Fenwick stablo (ili logaritamska struktura) je struktura podataka (vektor u STL-u) koji nam omogućuje da u vremenskoj složenosti ubacimo elemente u stablo i rešimo problem: koliko elemenata u strukturi je manje ili jednako od date vrednosti x.

Autor stabla je profesor sa Novog Zelanda, Peter Fenwick, 1994. godine u radu A New Data Structure for Cumulative Frequency Tables (1994)

Fenwick stablo (FWT) i tablica zaduženja elementa

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
stablo	1	1..2	3	1..4	5	5..6	7	1..8	9	9..10	11	9..12	13	13..14	15	1..16

U FWT je svaki element vektora zadužen da pamti koliko brojeva iz nekog intervala postoji u stablu. Na primer, element na indeksu 14 pamti koliko brojeva 13 ili 14 je ubačeno u FWT, dok element na indeksu 3 pamti koliko brojeva 3 je ubačeno u stablo.

FWT u elementu s indeksom x pamti koliko brojeva iz segmenta $[x+1-2^t..x]$ je u stablu gde 2^t je najmanja potencija broja dva sadržana u binarnom zapisu broja x. Na primer, $11=8+2+1=2^3+2^1+2^0$, te najmanja potencija broja dva sadrzana u 11 je 2^0, te element s indeksom 11 pamti koliko brojeva iz segmenta $[11+1-2^0..11]$ postoji u stablu, tj. element s indeksom 11 pamti koliko brojeva 11 postoji u strukturi podataka(stablu).
Slicno, $12=8+4$, , te najmanja potencija broja dva sadrzana u 12 je 2^2, te element s indeksom 12 pamti koliko brojeva iz segmenta $[12+1-2^2..12]=[9..12]$ postoji u stablu, tj. element s indeksom 12 pamti koliko brojeva 9,10,11,12 postoji u strukturi podataka(stablu).
Slicno, $16=2^4$, , te najmanja potencija broja dva sadrzana u 16 je 2^4, te element s indeksom 16 pamti koliko brojeva iz segmenta $[16+1-2^4..16]=[1..16]$ postoji u stablu, tj. element s indeksom 16 pamti koliko brojeva 1..16 postoji u strukturi podataka(stablu).

Na gornjoj slici ilustrovano je koji je element zadužen za pamćenje kojih elemenata. Element s indeksom 0 se ne koristi u strukturi i u strukturi se pamte samo prirodni brojevi.
Kako ubaciti element u FWT?
Zelimo li ubaciti broj 11, vidimo na slici da pojavu broja 11 broje elementi na indeksima 11,12,16 (a ako bi FWT bilo veće, to bi pamtili i brojevi 32, 64, 128,...) te elementima na tim indeksima pri ubacivanju broja 11 povećamo vrednost.
Dakle, ubacivanje broja 11 se svodi na:
FWT[11]++; FWT[12]++; FWT[16]++;
gde FWT je naziv vektora koji Fenwick stablo koristi.

Ako želimo izvaditi 11-icu iz strukture, postupimo slično:
FWT[11]--; FWT[12]--; FWT[16]--;

Kako izračunati na kojim indeksima treba obaviti operaciju ++ pri ubacivanju broja 11?

Sa slike smo videli da su to 11,12,16,...
1. Počinjemo od indeksa koji je jednak broju koji ubacujemo. Neka je to 11.
2. Nakon toga, broj povećavamo za njegovu najmanju potenciju broja dva i to ponavljamo.
Dakle $11=2^3+2^1+2^0$
Dakle, sledeci indeks je $11+2^0=12$
$12= 2^3+2^2$, te je sledeci indeks $12+2^2=16$
$16= 2^4$, te je sledeci $16+2^4=32$

Ako zelimo u FWT ubaciti npr. broj 7, onda povecavamo vrednost na indeksu 7, zatim na 8 (jer $7=2^2+2^1+2^0$, te je $7+2^0=8$), zatim 16 (jer $8=2^3$, potom $8+2^3=16$),...

Vrednost najmanje potencije broja dva u broju x mozemo jednostavno izracunati kao x&-x
To je koristan trik za dobijanje najmanje potencije broja dva u nekom broju

Kako koristeći FWT odgovoriti na pitanje: Koliko je ubačenih brojeva koji su manji ili jednaki x?
Neka x=11. Tada možemo kao rezultat dobiti FWT[8]+FWT[10]+FWT[11]. Zašto?
Zato što FWT[11] pamti koliko je ubačenih brojeva 11, FWT[10] pamti koliko je ubačenih brojeva 9 i 10, FWT[8] pamti koliko je ubačenih brojeva 1,2,3,4,5,6,7,8, te FWT[8]+FWT[10]+FWT[11] pamti koliko je ubačenih brojeva od 1 do 11, tj. manjih ili jednakih od x.

Kako izračunato da se radi baš o indeksima 8, 10, 11?
Počinjemo od indeksa koji je jednak broju x. Recimo da je to 11. Nakon toga smanjujemo taj broj za njegovu najmanju potenciju broja dva (x&-x) i to ponavljamo.
$11-2^0=10$, $10-2^1=8$

Sledi programski kod za FWT

```
#define MAX 100000
vector<int> FWT(MAX,0);
void ubaci(int x, int koliko)
{
  //x mora biti pozitivan
  for (;x< MAX;x+=x&-x) FWT[x]+=koliko;
}
```

/*Funkcija ubaci prima dva argumenta: prvi je broj x, a drugi koliko x-ova ubacujemo u FWT. Ako ubacujemo jedan x, onda koliko=1. Ako uklanjamo jedan x, onda koliko=-1 */

```
int upit(int x)
{
  //x mora biti pozitivan
 int rez=0;
  for (;x>0;x-=x&-x) rez+= FWT[x];
 return rez;
}
```

/*Funkcija upit odgovara na pitanje: **Koliko je ubačenih brojeva u Fenwick stablo koji su manji ili jednaki x?***/

VAŽAN TRIK: Želimo li u FWT ubaciti brojeve iz intervala [A,B], gde A I B mogu biti negativni, onda svaki broj pre ubacivanja umanjimo za A-1, a nakon izvlacenja uvecamo za A-1.

Poznati problemi:

1. KARTE OKRENUTE (postoji i na z-trening)

Statement:

There is an array of **n** cards. Each card is putted face down on table. You have two queries:

1. T i j (turn cards from index i to index j, include i-th and j-th card - card which was face down will be face up; card which was face up will be face down)
2. Q i (answer 0 if i-th card is face down else answer 1)

Solution:

This has solution for each query (and 1 and 2) has time complexity O(log n). In array **f** (of length **n + 1**) we will store each query **T (i , j)** - we set **f[i]++** and **f[j + 1]--**. For each card **k** between **i** and **j** (include **i** and **j**) sum **f[1] + f[2] + ... + f[k]** will be increased for 1, for all others will be same as before (look at the image 2.0 for clarification), so our solution will be described sum (which is same as cumulative frequency) module 2.

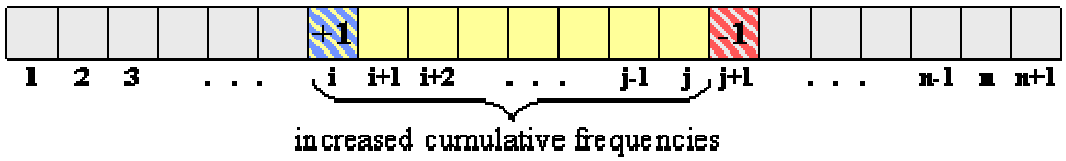


Image 2.0

Use **BIT** to store (increase/decrease) frequency and read cumulative frequency.

2. Floating median

http://community.topcoder.com/stat?c=problem_statement&pm=6551&rd=9990

Vremenska slozenost

U FWT se u slozenosti O(log MAX) moze ubaciti element.
U FWT se u slozenosti O(log MAX) moze reci koliko je ubacenih elemenata manje ili jednako x.
Ako nas zanima koliko brojeva postoji iz intervala [x,y] u strukturi, mozemo za O(log MAX) izracunati upit(y)-upit(x-1)/
Ako nas zanima koji je element x-ti po redu, mozemo u slozenosti O($\log^2$ n) uraditi sledece:
1. napraviti binarnu pretragu po upitima (O(log n))
2. izvrsiti upit nad FWT (sto je O(log n)) ili direktno implementirati O (log n) resenje.

22. Pljeskavice

Na rostilju se nalazi n pljeskavica (indeksiranih pocev od 1). Svaka pljeskavica ima dve strane A i B. Na pocetku su sve pljeskavice okrenute na stranu A. Sa standardnog ulaza se unosi n iz segmenta [1, 100000] i m iz segmenta [1,100000], a zatim m parova brojeva x i y. Par brojeva x,y opisuje opisuje neki od dva moguća događaja. Ako x jednako 0, onda ispisati na standardni izlaz na koju stranu je okrenuta pljeskavica y (strana A ili B). Ako x nije nula, onda na roštilju okrećemo sve pljeskavice od x do y (uključeno x i y).

ULAZ	IZLAZ
10 6 2 5 0 3 4 6 1 8 0 1 0 6	B B A

IDEJA:

Zadatak rešimo tako da u FWT ubacimo intervale pljeskavica koje okrećemo. To se postiže tako što se za interval [x,y] u FWT ubaci x, a izvadi y+1 (broj se moze izvuci i ako nije ubacen!!!).

	1.pljesk 0	2.pljesk 0	3.pljesk 0	4.pljesk 0	5.pljesk 0	6.pljesk 0	7.pljesk 0	8.pljesk 0	9.pljesk 0	10.pljesk 0
2 5	A	B (+1)	B (+1)	B (+1)	B (+1)	A (-1)	A	A	A	A
0 3										
4 6				A (+1)	A(+1)	B (+1)	A (-1)			
1 8	B (+1)	A (+1)	A(+1)	B(+1)	B(+1)	A(+1)	B(+1)	B(+1)	A(-1)	

0 1										
0 6										

Ideja je da ako u stanju 2 5 se okrecu pljeskavice (umesto da postavimo niz pljeskavica na nula i onda za pljeskavice od 2..5 povecamo vrednosti polja za +1), onda u strukturu ubacimo pocetak i kraj okretanja, tj. pocetak okretanja i sledeci pocetak neokretanja tako sto dodamo jedan na pocetku intervala, a oduzmemo jedan na kraju intervala.
U svakom redu suma elemenata, tj. brojeva +1 i -1 predstavlja broj pljeskavica koji je okrenut u stanju i.
Setite se zadatka o uciteljici Danki!!!

Dakle, tim se u FWT pri postavljanju pitanja za bilo koji broj iz intervala dobije 1, a van intervala 0.
Pri odgovoru na pitanja, pogledamo da li je pljeskavica okrenuta paran ili neparan broj puta i prema tome se ispise A ili B.

```
#include <iostream>
#include <vector>
using namespace std;

#define MAX 100000
vector<int> FWT(MAX,0);
void ubaci(int x, int koliko)
{
  //x mora biti pozitivan
  for (;x< MAX;x+=x&-x) FWT[x]+=koliko;
}

int upit(int x)
{
  //x mora biti pozitivan
 int rez=0;
  for (;x>0;x-=x&-x) rez+= FWT[x];
 return rez;
}

int main()
{ int n,m,x,y;
  cin >> n >> m;
  for(int i=0;i<m;i++)
  {
     cin >>x >>y;
/*
Dakle, u glavnom programu ako x nije nula,
   onda ubacimo 1 na mesto x i -1 na mesto y+1
Dakle, mozemo reci da ubacimo x, a izvadimo y+1.
Ako je x==0, ispisemo A ili B
*/

     if(x){
     ubaci(x,1); ubaci(y+1,-1);
     }
     else
      cout << (char) ((upit(y)&1)+'A') << endl;
  }
   return 0;
}
```

Zadatak sa uciteljicom: Uciteljica vodi decu u zabavni park. Kada se deca zabavljaju uciteljica to mora platiti, a kad se ne zabavljaju ne mora. Svako dete unapred odabere od kada do kada ce se zabavljati. Uciteljica placa svaki minut zavisno od broja dece koji se taj minut zabavljaju. Uciteljica dobije popust ako se zabavlja vise dece. Sa standardnog

ulaza se unose dva prirodna broja n, m (n<100000, m<1000000). Potom se ucitava n parova a i b (1<=a<=b<=10000) tako da svaki par brojeva predstavlja ukljucni interval vremena unutar kog se pojedino dete zabavlja. Ako se u nekom miniti zbavlja x dece, onda uciteljica tada placa (m-x)*x novaca, a nikad se nece zbavljati vise od m dece. Koliko novaca ce uciteljica potrositi?
ULAZ
4 101
1 5
3 4
2 13
12 14
IZLAZ
2180

23. Uvod u grafove – MINESWEEPER
MINESWEEPER - primer jednostavne igrice.

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
/* Dimenzija table */
int n;
/* Tabla koja sadrzi 0 i 1 u zavisnosti od toga da li na polju postoji bomba */
int** bombe;
#define PRAZNO (-1)
#define ZATVORENO 0
#define ZASTAVICA 9
/* Tabla koja opisuje tekuce stanje igre.
Moze da sadrzi sledece vrednosti :
ZATVORENO - opisuje polje koje jos nije bilo otvarano
PRAZNO - polje na kome ne postoji ni jedna bomba
BROJ od 1-8 - polje koje je otvoreno i na kome pise koliko bombi postoji u okolini
ZASTAVICA - polje koje je korisnik oznacio zastavicom
*/

int** stanje;

/* Ukupan broj bombi */
int broj_bombi;

/* Ukupan broj postavljenih zastavica */
int broj_zastavica = 0;

/* Pomocne funkcije za rad sa matricama */
int** alociraj(int n)
{
int i;
int** m=(int**)malloc(n*sizeof(int*));
if(m==NULL) {printf("Greska prilikom alokacije memorije\n"); exit(1);}
for (i=0; i<n; i++)
{ m[i]=(int *)calloc(n,sizeof(int));
if(m[i]==NULL) {printf("Greska prilikom alokacije memorije\n"); exit(1);}
}
return m;
}
void obrisi(int** m, int n)
{ int i;
```

```
for (i=0; i<n; i++) free(m[i]);
free(m);
}

/* Funkcija postavlja bombe */
void postavi_bombe()
{
broj_bombi=(n*n)/6;
int kolona;
int vrsta;
int i;
/* Inicijalizujemo generator slucajnih brojeva */
srand(time(NULL));

for (i=0; i<broj_bombi; i++)
{ /* Racunamo slucajni polozaj bombe */
kolona=rand()%n;
vrsta=rand()%n;
/* Ukoliko bomba vec postoji tu, opet idemo u istu iteraciju */
if (bombe[vrsta][kolona]==1)
{ i--;  continue;}

/* Postavljamo bombu */
bombe[vrsta][kolona]=1;
}
}

/* Funkcija ispisuje tablu sa bombama */
void ispisi_bombe()
{
int i,j;
for (i=0; i<n; i++)
{ for (j=0; j<n; j++) printf("%d",bombe[i][j]);
   printf("\n");
}
}

/* Funkcija ispisuje tekuce stanje */
void ispisi_stanje()
{ int i,j;
for (i=0; i<n; i++)
{ for (j=0; j<n; j++) { if (stanje[i][j]==ZATVORENO) printf(".");
                        else if (stanje[i][j]==PRAZNO)  printf(" ");
                       else if (stanje[i][j]==ZASTAVICA) printf("*");
                       else printf("%d",stanje[i][j]);
}
printf("\n");
}
}

/* Funkcija postavlja zastavicu na dato polje ili je uklanja
ukoliko vec postoji */
void postavi_zastavicu(int i, int j)
{
if (stanje[i][j]==ZATVORENO)
{ stanje[i][j]=ZASTAVICA; broj_zastavica++;}
else if (stanje[i][j]==ZASTAVICA)
```

```
{ stanje[i][j]=ZATVORENO; broj_zastavica--;}
}

/* Funkcija izracunava koliko bombi postoji u okolini date bombe */
int broj_bombi_u_okolini(int v, int k)
{ int i, j;
int br=0;
/* Prolazimo kroz sva okolna polja */
for (i=-1; i<=1; i++)
   for(j=-1; j<=1; j++)
{ /* preskacemo centralno polje */
    if (i==0 && j==0) continue;
/* preskacemo polja "van table" */
if (v+i<0 || k+j<0 || v+i>=n || k+j>=n) continue;
if (bombe[v+i][k+j]==1) br++;
}
return br;
}

/* Centralna funkcija koja vrsi otvaranje polja i pritom se otvaranje "siri" i na polja koja su oko datog */
void otvori_polje(int v, int k) {
/* Ukoliko smo "nagazili" bombu
zavrsavamo program */
if (bombe[v][k]==1)
{ printf("BOOOOOOOOOOOOOOOM!!!!\n"); ispisi_bombe(); exit(1);}
else
{
/* Brojimo bombe u okolini */
int br=broj_bombi_u_okolini(v,k);
/* Azuriramo stanje ovog polja */
stanje[v][k]=(br==0)?PRAZNO:br;
/* Ukoliko u okolini nema bombi, rekurzivno otvaramo sva polja u okolini koja su zatvorena */
if (br==0)
{
/* Petlje indeksiraju sva okolna polja */
int i,j;
for (i=-1; i<=1; i++)
for (j=-1; j<=1; j++)
{
/* Preskacemo centralno polje */
/* if (i==0 && j==0) continue; */
/* Preskacemo polja van table */
if (v+i<0 || v+i>=n || k+j<0 || k+j>=n) continue;
/* Ukoliko je okolno polje zatvoreno, otvaramo ga */
if (stanje[v+i][k+j]==ZATVORENO) otvori_polje(v+i, k+j);
}
}
}
}

/* Funkcija utrdjuje da li je partija gotova
Partija je gotova u trenutku kada su sve bombe pokrivene zastavicama i kada nijedno drugo polje nije
pokriveno zastavicom */
int gotova_partija()
{ int i,j;
for (i=0; i<n; i++)
for (j=0; j<n; j++)
```

```
{ /* Ukoliko postoji nepokrivena bomba, partija nije zavrsena */
if (bombe[i][j]==1 && stanje[i][j]!=ZASTAVICA) return 0;
}

/* Partija je zavrsena samo ukoliko je broj zastavica jednak broj bombi */
return broj_zastavica==broj_bombi;
}

main() {
/* Unosimo dimenziju table */
printf("Unesite dimenziju table : ");
scanf("%d",&n);

/* Alociramo table */
bombe=alociraj(n);
stanje=alociraj(n);

/* Postavljamo bombe */
postavi_bombe();

/* Sve dok partija nije gotova */
while(!gotova_partija())
{ int v,k;
  char akcija;

/* Ispisujemo tekuce stanje */
ispisi_stanje();

/* Sve dok korisnik ne unese o ili zatrazimo od njega da upise odgovarajucu akciju */
do
{
getchar();
printf("Unesi akciju (o - otvaranje polja,z - postavljanje zastavice) : ");
scanf("%c",&akcija);
} while (akcija!='o' && akcija!='z');

/* Trazimo od korisnika da unese koordinate polja sve dok ih ne unese ispravno
Korisnicke koordinate krecu od 1, a interne od 0 */
do
{
printf("Unesi koordinate polja : ");
scanf("%d",&v);
scanf("%d",&k);
} while(v<1 || v>n || k<1 || k>n);

/* Reagujemo na akciju */
switch(akcija)
{ case 'o': otvori_polje(v-1,k-1);break;
case 'z': postavi_zastavicu(v-1,k-1);
}
}

/* Konstatujemo pobedu */
ispisi_stanje();
printf ("Cestitam! Pobedili ste\n");
obrisi(stanje,n);
obrisi(bombe,n);
```

}