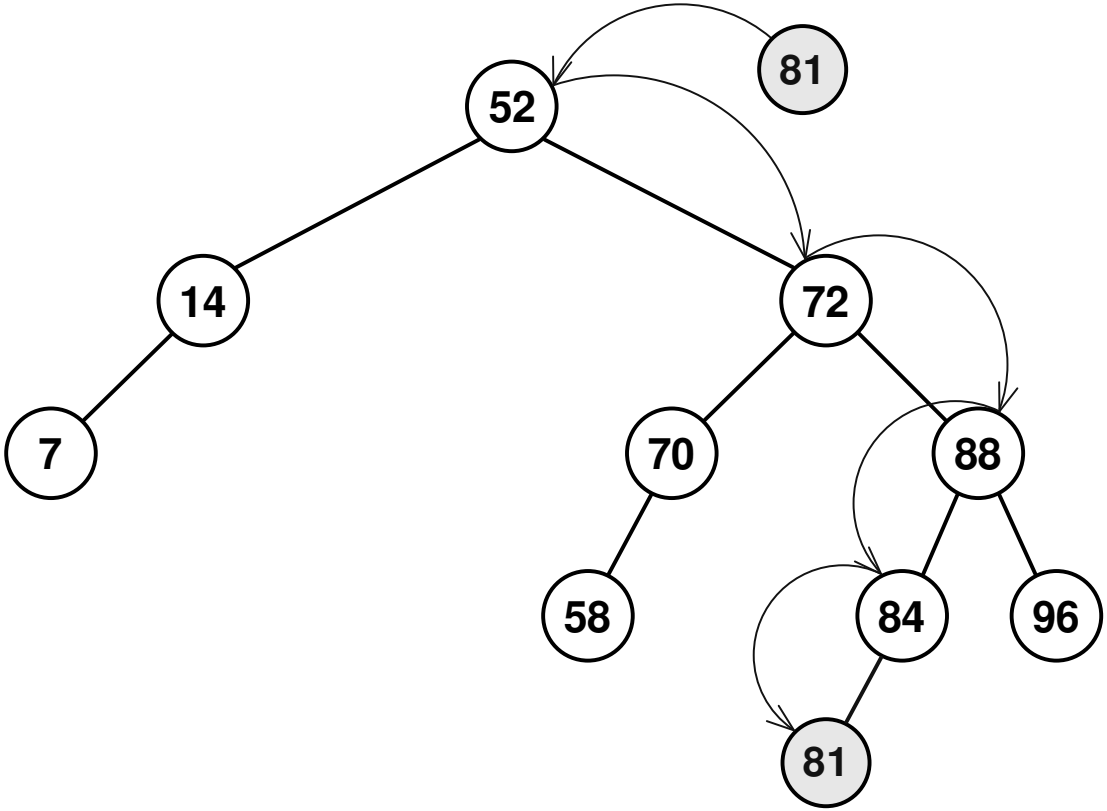
Binarno stablo traženja

- Binarno stablo T je **binarno stablo traženja** ako su ispunjeni sljedeći uvjeti:
  - čvorovi od T su označeni podacima nekog tipa na kojem je definiran totalni uređaj $\leq$.
  - neka je *i* bilo koji čvor od T. Tada su oznake svih čvorova u lijevom podstablu od *i* manje od oznake od *i*. Također, oznake svih čvorova u desnom podstablu od *i* su veće ili jednake od oznake od *i*.

- Zadatak: Nacrtati izgled binarnog sortiranog stabla (binarnog stabla traženja) nakon dodavanja sljedećih podataka: 52, 72, 14, 88, 70, 84, 7, 58 i 96. Nacrtati postupak dodavanja podatka 81 u tako već formirano binarno stablo.



- Zadatak: Ukoliko se stablo prikazuje jednodimenzionalnim poljem, s koliko se članova mora dimenzionirati u najgorem i najboljem slučaju ako je potrebno pohraniti 52 različita elementa?

- Općenito u stablo dubine *k* stane 2^k-1 članova.

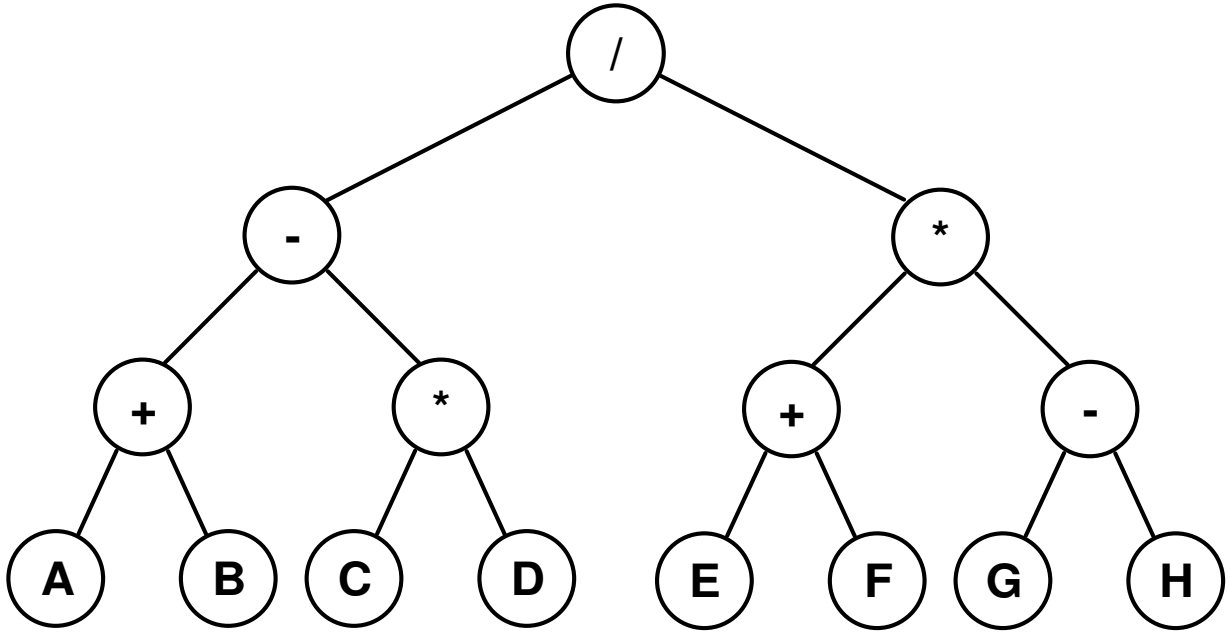
- Najbolji slučaj – stablo će biti **potpuno,** pa je za pohranu potrebna dubina koja je jednaka najmanjoj potenciji broja 2 koja je veća od 52, a to je $64=2^6$. Dakle, u stablo dubine 6 stane 63 elementa te je to dovoljno za pohranu 52 elementa. U najboljem slučaju potrebno je polje dimenzionirati na 63.
- Najgori slučaj – stablo će biti potpuno **koso** i dubina stabla će biti 52. Za pohranu je potrebno polje dimenzionirati s 2^{52}-1= 4,503,599,627,370,495 ($\approx 4.5*10^{15}$)

- Zadatak: Pretražuje se stablo u kojem se nalazi 27 različitih elemenata. Potrebno je odrediti koliko se operacija uspoređivanja obavi u najboljem i najgorem slučaju za potpuno i koso stablo.

- Potpuno stablo:
- najbolji slučaj – pri prvoj usporedbi nađe se element $\Rightarrow$ obavljena je jedna usporedba
- najgori slučaj – podatak se nalazi u "najdubljem" *listu*. Najmanja potencija broja 2 koja je veća od 27 je $32=2^5$. Dakle stablo ima dubinu 5 te je potrebno obaviti najviše pet operacija uspoređivanja.

- Koso stablo:
- najbolji slučaj – pri prvoj usporedbi nađe se element $\Rightarrow$ obavljena je jedna usporedba
- najgori slučaj – podatak se nalazi u "najdubljem" *listu*. Budući da je u najgorem slučaju stablo u potpunosti koso, razina najdubljeg čvora je 27. Dakle, u najgorem slučaju potrebno je obaviti 27 operacija uspoređivanja.

- Zadatak: Za već formirano binarno stablo potrebno je napisati što se dobije *inorder* i *postorder* obilaskom. Stablo izgleda na sljedeći način:



- Inorder obilazak stabla daje matematički izraz:
- A + B - C * D / E + F * G – H
- Postorder obilazak također daje matematički izraz ali u tzv. RPN notaciji:
- A B + C D * - E F + G H - * /

Implementacija rječnika pomoću binarnog stabla traženja

- Pomoću binarnog stabla traženja možemo na efikasan način implementirati operacije umetanja, traženja i brisanja elemenata u skupu podataka - obradit ćemo primjer rječnika s predavanja

- Operacije koje ćemo implementirati su sljedeće:
  - Insert(x, T) – operacija koja ubacuje novi element *x* u binarno stablo traženja T.
  - Member (x, T) – operacija koja vraća vrijednost 1 ako se element x nalazi u stablu traženja T, a u suprotnom vraća 0
  - Delete(x, T) – operacija koja uklanja element x iz binarnog stabla traženja T.
  - Inorder(T) – operacija koja obilazi binarno stablo traženja T čvor po čvor algoritmom "inorder" i ispisuje sve njegove elemente.

- Operacije Insert i Member rade slično: traže mjesto u stablu na kojem bi morao biti (novi) element, te ubacuje čvor na to mjesto ili javlja da čvor (ne)postoji

- Nešto je složenija operacija Delete(x, T). Imamo tri slučaja:
  - *x* je u listu; tada jednostavno izbacimo list iz stabla.
  - *x* je u čvoru koji ima samo jedno dijete. Tada nadomjestimo čvor od *x* njegovim djetetom.
  - *x* je u čvoru koji ima oba djeteta. Tada nađemo najmanji element *y* u desnom podstablu čvora *x*. Izbacimo čvor od *y* (jedan od dva prethodna slučaja). U čvor *x* spremimo *y* umjesto *x*.
- Sve se ovo spretno može zapisati ako uvedemo pomoćnu funkciju DeleteMin(T). Ta funkcija iz nepraznog stabla T izbacuje čvor s najmanjim elementom, te vraća taj najmanji element.

- Svaka od funkcija Member, Insert, Delete prolazi jednim putem, od korijena binarnog stabla do nekog čvora. Zato je vrijeme izvršavanja svih operacija ograničeno visinom stabla. Pretpostavimo da u stablu imamo *n* elemenata. Visina stabla tada varira između $(\log_2(n+1))-1$ i $n - 1$. Ekstremni slučajevi su potpuno stablo i "ispruženo" stablo – lanac.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
typedef int elementtype;
struct node{
     elementtype el;
     struct node * left;
     struct node * right; };
typedef struct node * BSTree;
int Member(elementtype x, BSTree t){
     if (t == NULL) return 0;
     if (x == t->el) return 1;
     if (x < t->el) return Member(x, t->left);
     else return Member(x, t->right);}

int Insert(elementtype x, BSTree * t){
     BSTree tree = *t;
     if (tree == NULL){
              tree = malloc(sizeof(struct node));
              tree->el = x;
              tree->left = tree->right = NULL;
              *t = tree;
              return 1;  }
     if (x == tree->el) return 0; // x je već u rječniku
     if (x < tree->el) return Insert(x, &tree->left);
     else    return Insert(x, &tree->right);}
```

```
elementtype DeleteMin(BSTree * t)
{
    BSTree temp;
    BSTree tree = *t;
    elementtype minval;

    if (tree->left == NULL)
    {
        // tree pokazuje na cvor s najmanjim elementom. Nadomjestimo taj cvor njegovim
        // desnim djetetom
        minval = tree->el;
        temp = tree;
        tree = tree->right;
        free(temp);
    }
    else
        //cvor na kojeg pokazuje tree ima lijevo dijete
        minval = DeleteMin(&tree->left);

    *t = tree;
    return minval;
}
```

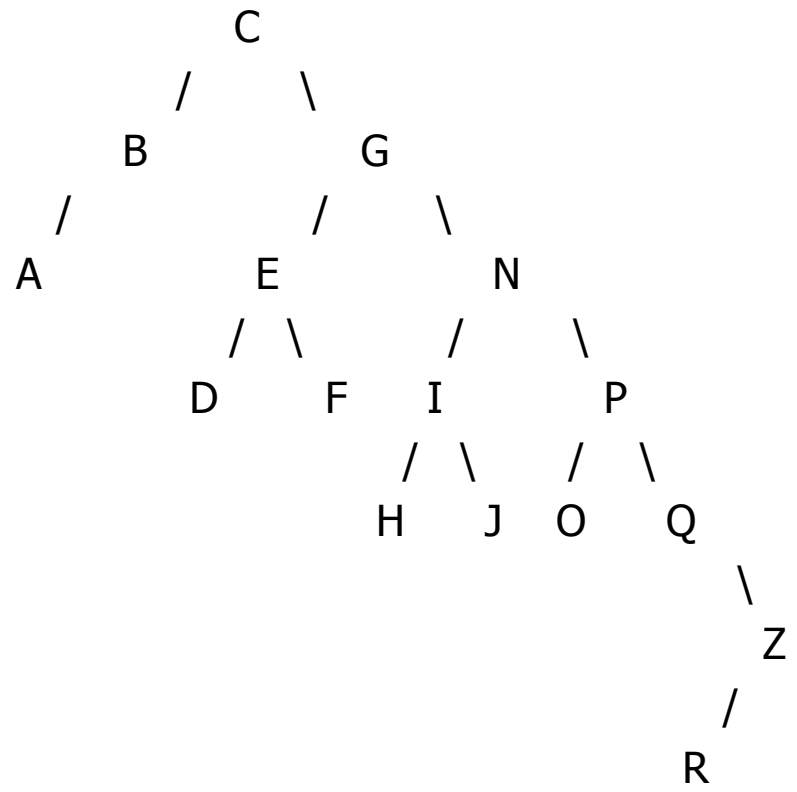
```
void Delete(elementtype x, BSTree * t) {
     BSTree temp;
     BSTree tree = *t;
     if (tree == NULL) return;
     if (x < tree->el)
               Delete(x, &tree->left);
     else if (x > tree->el)
               Delete(x, &tree->right);
               //ako dođemo ovamo, tada je x u čvoru na kojeg pokazuje t
               else if (tree->left == NULL && tree->right == NULL)          {
                                        free(tree);
                                        tree = NULL;
                              } else {
                                        if (tree->left == NULL)
                                        {   //nadomjestimo čvor od x njegovim desnim djetetom
                                                  temp = tree;
                                                  tree = tree->right;
                                                  free(temp);
                                        } else if (tree->right == NULL)  {// lijevo dijete u čvor od x
                                                            temp = tree;
                                                            tree = tree->left;
                                                            free(temp);
                                                  } else //postoje oba djeteta
                                                            tree->el = DeleteMin(&tree->right);
                              } *t = tree;
}
```

```
void Inorder(BSTree t) {
     if (t == NULL) return;
     if (t->left != NULL)
              Inorder(t->left);
     printf("%i ", t->el);
     if (t->right != NULL)
              Inorder(t->right);
}

int main() {
     BSTree myTree = NULL;   // kreiramo prazno stablo i ubacimo nekoliko elemenata
      int broj,ind=0,fg;
     elementtype clan;

     printf ("Koliko clanova zelite upisati u listu\n");
     scanf("%d",&broj);
     printf("\n");
      srand(time(NULL));
      do {     clan = (elementtype) 100 * ((float)rand() / (RAND_MAX + 1));
              fg=Insert(clan,&myTree);
               ind+=fg,
              } while(ind<broj)

 //prosetamo po stablu algoritmom inorder i ispisemo elemente
     Inorder(myTree);
     printf("\n");
```

```
//izbrisemo element
     printf("Upisite element koji zelite izbrisati, -1 za kraj\n");
     do {
        scanf("%d",&clan);
            Delete(clan,&myTree);
            Inorder(myTree);
        printf("\n");
     }
            while (clan != -1);
// provjera da li je element u rjecniku

     printf("Upisite element koji trazite, -1 za kraj\n");
     do {
        scanf("%d",&clan);
            if (Member(clan, myTree))
            printf("Broj %d se nalazi u stablu",clan);
     else
            printf("Broj %d se ne nalazi u stablu",clan);
     printf("\n");
     } while (clan != -1);
      system("PAUSE");
      return 0;
 }
```

Drugačija izvedba binarnog stabla traženja

- Primjer: napraviti binarno stablo traženja u koje se unosi do 14 znakova (char) po uređaju abecede. Ispisati stablo po inorder, preorder i postorder algoritmu. Također ispisati strukturu stabla. Napisati funkciju koja pretražuje stablo i javlja da li se traženi element nalazi u stablu.
- Neka su ulazni podaci {C, B, G, A, E, F, N, D, P, C, Q, Z, I, R, H, J, O}. Stablo će onda biti:



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

struct cv {
  char element[15];
  struct cv *lijevo;
  struct cv *desno;
};
typedef struct cv cvor;

// upisuje u stablo podatke: lijevo manji, desno veci
cvor *upis (cvor *glava, char element[]) {
  int smjer; // odluka o podstablu

  if (glava == NULL) { // prazno (pod)stablo
            glava = (cvor *) malloc (sizeof (cvor));
            if (glava) {
              strcpy (glava->element, element);
              glava->lijevo = glava->desno = NULL;
            } else {
                        printf ("U memoriji nema mjesta za upisati '%s'\n", element);
            }
           }
```

```
  else if ((smjer = strcmp (element, glava->element)) < 0) {
            glava->lijevo = upis (glava->lijevo, element);
        } else if (smjer > 0) {
             glava->desno  = upis (glava->desno, element);
        } else {
                        printf ("Podatak '%s' vec postoji!\n", element);
         }
     return glava; // pokazivac na zadnji element
 }

// obilazak inorder lijevo-desno:obilazi najlijevije podstablo, pa korijen, pa onda ostala
void ispisinld (cvor *glava) {
  if (glava != NULL) {
            ispisinld (glava->lijevo);
    printf ("%s \n", glava->element);
    ispisinld (glava->desno);
  }
}

// obilazak inorder desno-lijevo
void ispisindl (cvor *glava) {
  if (glava != NULL) {
            ispisindl (glava->desno);
    printf ("%s \n", glava->element);
    ispisindl (glava->lijevo);
  } }
```

```
// obilazak preorder: prvo korijen, pa obilazi podstabla od najlijevijeg
void ispispre (cvor *glava) {
  if (glava != NULL) {
    printf ("%s \n", glava->element);
    ispispre (glava->lijevo);
    ispispre (glava->desno);
  } }

// obilazak postorder:obilazi podstabla od najlijevijeg, korijen zadnji
void ispispost (cvor *glava) {
  if (glava != NULL) {
    ispispost (glava->lijevo);
    ispispost (glava->desno);
    printf ("%s \n", glava->element);
  } }

// ispis stabla
void ispissta (cvor *glava, int nivo) {
  int i;
  if (glava != NULL) {
    ispissta (glava->desno, nivo+1);
    for (i = 0; i < nivo; i++) printf("    ");
    printf ("%s \n", glava->element);
    ispissta (glava->lijevo, nivo+1);
  } }
```

```
// trazenje cvora u binarnom stablu
cvor *trazi (cvor *glava, char element[]) {
      int smjer;
      if (glava) {
            if ((smjer = strcmp (element, glava->element)) < 0) {
                  return trazi (glava->lijevo, element);
            } else if (smjer > 0) {
                  return trazi (glava->desno, element);
            } }
      return glava; // ili je pronadjen ili NULL; }

void main(void) {
  FILE *fi;                                          // ulazna datoteka
  int j;                                                    // brojac podataka
  cvor *glava, *p;    // pokazivac na korijen, pomocni pokazivac
  char ime[15];                                 //

  fi = fopen ("UlazZaSortiranoStablo.txt", "r");
  if (fi) {                 // inicijalizacija i citanje podataka
    j = 1;
    glava = NULL;
    while (fscanf (fi, "%s", &ime) != EOF) {
      printf ("%d. ulazni podatak je %s \n", j++, ime);
      glava = upis (glava, ime);
    }  fclose (fi);
```

```
   getchar (); // obilazak i ispis stabla
   printf ("Ispis inorder lijevo-desno\n");
   ispisinld (glava);  getchar ();
   printf ("Ispis inorder desno-lijevo\n");
   ispisindl (glava);  getchar ();
   printf ("Ispis preorder\n");
   ispispre (glava);  getchar ();
   printf ("Ispis postorder\n");
   ispispost (glava); getchar ();
   printf ("Ispis stabla\n");
   ispissta (glava, 0);
            while (1) {  // trazenje elementa
                     printf ("Unesite element koji trazite, ili KRAJ >");
                     scanf ("%s", ime);
                     if (stricmp (ime, "KRAJ") == 0) break;
                     p = trazi (glava, ime);
                     if (p) {
                              printf ("Pronadjen je element: %s\n", p->element);
                     } else {
                              printf ("Nije pronadjen element: %s\n", ime);
                     }           } }
else {
   printf ("Nema ulaznih podataka\n");
            exit (1); }
  system("PAUSE");
  exit (0); }
```

Primjer baratanja s podatcima u binarnom stablu traženja

- Napisati funkciju koja upisuje u čvor BST podatke tipa: naziv proizvoda (char 15) i cijena (float). Uređaj između zapisa u čvorovima određuje naziv proizvoda (manje znači bliže početku abecede). Napisati funkciju koja ispiše zapise u BST po algoritmu INORDER, te funkciju koja izračuna prosječnu cijenu proizvoda u cjeniku.

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

typedef struct {
  char artikl[15+1];
  float cijena;
} el;

typedef struct cv {
  el element;
  struct cv *lijevo;
  struct cv *desno;
} cvor;

typedef struct {
  float suma;
  int broj;
} pros;
```

```
// upisuje u stablo podatke: lijevo manji, desno veci
cvor *upis (cvor *glava, el element) {
  int smjer;
  if (glava == NULL) {
    glava = (cvor *) malloc (sizeof (cvor));
            if (glava) {
                        glava->element = element;
                        glava->lijevo = glava->desno = NULL;
            } else {
                        printf ("Nema dovoljno memorije!\n");
            }
  }
    else if ((smjer = strcmp (element.artikl, glava->element.artikl)) < 0) {
    glava->lijevo = upis (glava->lijevo, element);
  }
    else if (smjer > 0) {
    glava->desno  = upis (glava->desno, element);
  }
    else {
    printf ("Podatak '%s' vec postoji!\n", element.artikl);
  }
  return glava;
}
```

```
// ispis inorder
void ispisin (cvor *glava) {
  if (glava != NULL) {
    ispisin (glava->lijevo);
    printf ("%-15s %6.2f\n", glava->element.artikl, glava->element.cijena);
    ispisin (glava->desno);
  }
}

// sumiranje cijena i brojanje elemenata
void prosjek (cvor *glava, pros *prs) {
  if (glava != NULL) {
            prs->suma += glava->element.cijena;
            prs->broj++;
    prosjek (glava->lijevo, prs);
    prosjek (glava->desno, prs);
  }
}

void main(void) {
  FILE *fi;
  int j;
  cvor *glava;                    // pokazivac na korijen
  el element;                     // sadrzaj cvora
  pros prs;                       // broj elemenata i suma cijena
```

```
  prs.suma = 0.; prs.broj = 0;
  fi = fopen ("UlazZaProsjekUStablu.txt", "r");
  if (!fi) {
    printf ("Nema ulaznih podataka\n");
              exit (1);
  }
  j = 1;
  glava = NULL;
  while (fscanf (fi, "%s %f", element.artikl, &element.cijena) != EOF) {
    printf ("%2d. ulazni podatak je %-15s %6.2f\n", j++, element.artikl, element.cijena);
    glava = upis (glava, element);
  }
  fclose (fi);
     // ispis, racun sume cijena i broja elemenata
     getchar ();
     ispisin (glava);
     getchar ();
     prosjek (glava, &prs);
     if (prs.broj) {
          printf ("Suma=%6.2f, Broj cvorova=%d, Prosjek=%6.2f\n",
                              prs.suma, prs.broj, prs.suma / prs.broj);
              getchar ();
  }
     exit (0);
}
```

Nešto složeniji primjer binarnog stabla traženja

Zadatak: Zadani su podaci o studentu: matični broj long, ime 25 znakova, prezime 25 znakova i godina rođenja short. Uz uporabu dinamičke strukture binarnog sortiranog stabla (tj. binarno stablo traženja) potrebno je napisati funkcije za:

a) pohranu podataka o studentima uz uvjet da je omogućeno brže pronalaženje studenata po prezimenu

b) pronalaženje studenta po prezimenu

c) ispis čvorova po INORDER algoritmu

d) ispis čvorova na zadanoj razini

e) izračunavanje dubine stabla i najmanje razine listova

f) ispis listova stabla

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

struct podaci {
  long matBroj;
  char ime[25+1];
  char prezime[25+1];
  short godRod;
};
typedef struct podaci Podaci;

struct cvor {
  Podaci * student;
  struct cvor * lijevo;
  struct cvor * desno;
};
typedef struct cvor Cvor;
```

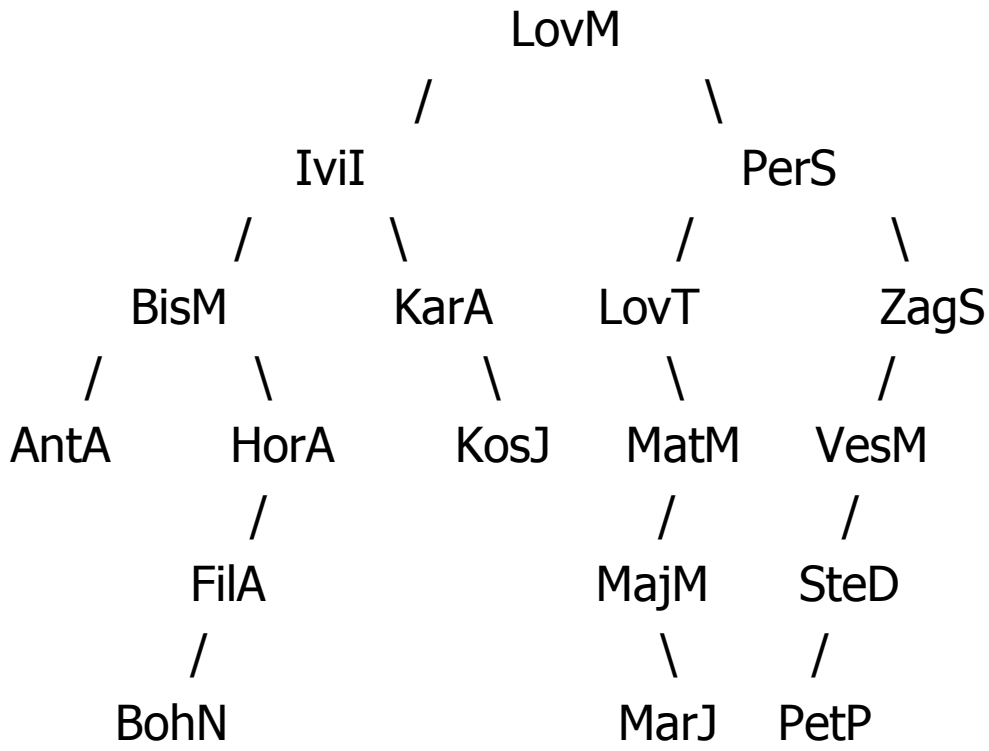
```
// funkcija dodaje podatak u binarno sortirano stablo, kljuc je prezime
Cvor * dodajPrezime (Cvor * korijen, Podaci * student) {
  int smjer;
  if (korijen == NULL) {
    korijen = (Cvor *) malloc (sizeof(Cvor));
    if (korijen) {
      korijen->student = student;
      korijen->lijevo = korijen->desno = NULL;    }
  } else if ((smjer = strcmp(student->prezime, korijen->student->prezime)) < 0)
           korijen->lijevo = dodajPrezime (korijen->lijevo, student);
        else if (smjer > 0)
               korijen->desno = dodajPrezime (korijen->desno, student);
             else
               printf("Podatak %s vec postoji!\n", student->prezime);
  return korijen;}

// funkcija pretrazuje binarno stablo trazenja po kljucu prezime
Cvor * potraziPrezime (Cvor * korijen, char * prezime) {
  int smjer;
  if (korijen) {
    if ((smjer = strcmp(prezime, korijen->student->prezime)) < 0)
      return potraziPrezime (korijen->lijevo, prezime);
    else if (smjer > 0)
           return potraziPrezime (korijen->desno, prezime); }
  return korijen; }
```

```
// funkcija inorder ispisuje zadano binarno stablo
void inOrder (Cvor * korijen) {
  if (korijen) {
    inOrder (korijen->lijevo);
    printf("%s, %s %d %d\n", korijen->student->prezime, korijen->student->ime,
      korijen->student->matBroj,korijen->student->godRod);
    inOrder (korijen->desno);
  }
}

// funkcija ispisuje sve cvorove zadane dubine
// poziv ispis(korijen, 1, n) gdje je n trazena dubina
void ispisi (Cvor * korijen, int trenutnaDubina, int trazenaDubina) {
  if (korijen) {
    if (trenutnaDubina == trazenaDubina)
      printf("%d %s %s %d\n",korijen->student->matBroj, korijen->student->ime,
      korijen->student->prezime,korijen->student->godRod);
    ispisi (korijen->lijevo, trenutnaDubina + 1, trazenaDubina);
    ispisi (korijen->desno, trenutnaDubina + 1, trazenaDubina);
  }
}
```

```
// funkcija trazi listove s najvecom i najmanjom razinom
void dubine(Cvor * korijen, int trenDub, int * maxDub, int * minDub) {
  if (korijen) {
    if (!korijen->lijevo & !korijen->desno) {
      if (*maxDub == 0 || trenDub > *maxDub)
        *maxDub = trenDub;
      if (*minDub == 0 || trenDub < *minDub)
        *minDub = trenDub;
    }
    else {
      dubine(korijen->lijevo, trenDub + 1, maxDub, minDub);
      dubine(korijen->desno, trenDub + 1, maxDub, minDub);
     }
  }
}

// funkcija za zadano binarno stablo ispisuje listove
void ispisiListove(Cvor * korijen) {
  if (korijen) {
    if (!korijen->lijevo & !korijen->desno)
      printf("%s %s %d %d; ", korijen->student->ime, korijen->student->prezime,
            korijen->student->matBroj,korijen->student->godRod);
    ispisiListove(korijen->lijevo);
    ispisiListove(korijen->desno);
  }
}
```

```
void main () {
  FILE * fUl;
  char buf[256], trazim[25+1];
  Podaci * student;
  Cvor * korijenPrezime = NULL;
  Cvor * trazeni;
  int minDubina = 0, maxDubina = 0;
  int i;

  if ((fUl = fopen("studenti.txt", "r")) == NULL) {
    printf("Ne mogu otvoriti 'studenti.txt'\n");
    exit(1);
  }
  while (fgets(buf, 256, fUl)) {
    student = (Podaci *) malloc(sizeof(Podaci));
    sscanf(buf, "%ld;%[^;];%[^;];%d", &(student->matBroj), student->ime,
      student->prezime, &(student->godRod));
    korijenPrezime = dodajPrezime(korijenPrezime, student);
  }
  fclose(fUl);

  inOrder(korijenPrezime);
  printf("\n");
```

```
while (1){
      printf("Upisite prezime koje se trazi, kraj za prekid trazenja\n");
      scanf("%s",trazim);
      if (strcmp(trazim,"kraj")==0) break;
      if(trazeni = potraziPrezime(korijenPrezime, trazim))
        printf("Pronasao: %s, %s %ld %d\n", trazeni->student->prezime,
        trazeni->student->ime, trazeni->student->matBroj, trazeni->student->godRod);
      else
              printf("Nije nadjen student %s\n",trazim);
}

  printf("\nIspis listova:\n");
  ispisiListove(korijenPrezime);

  dubine(korijenPrezime, 1, &maxDubina, &minDubina);
  printf("\nMaxDubina=%d  MinDubina=%d\n", maxDubina, minDubina);

  for (i=1; i<=maxDubina; i++) {
    printf("%d razina:\n", i);
    ispisi(korijenPrezime, 1, i);
  }
 system("PAUSE");
}
```


LovM
IviI
PerS
BisM
KarA
LovT
ZagS
AntA
HorA
KosJ
MatM
VesM
FilA
MajM
SteD
BohN
MarJ
PetP