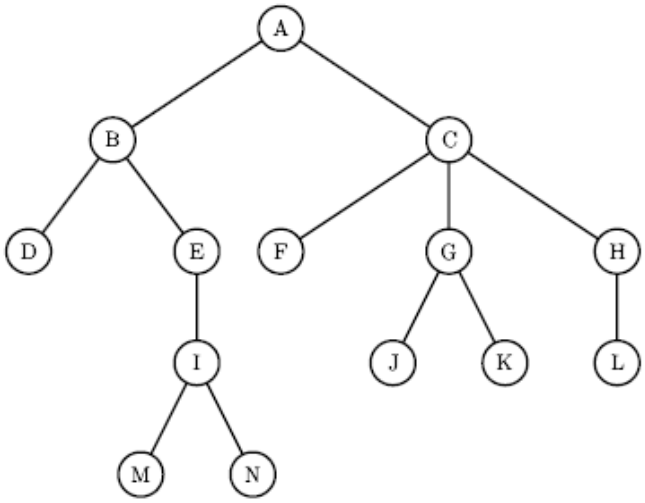
ALGORITMI I STRUKTURE PODATAKA

III čas

Zaokružite slovo ispred tačnog tvrđenja. Netačna tvrđenja obrazložiti (u zavisnosti od tvrđenja: primerom, tačnim tvrđenjem,...).

1. Uklanjanje elementa sa steka obavlja se po FIFO principu (First In First Out).
2. Umetanje i uklanjanje elementa iz reda se obavlja sa istog pristupnog kraja, tj. vrha reda.
3. Vreme izvršavanja operacija PUSH i POP za rad sa stekom je $O(1)$.
4. Vreme izvršavanja operacije umetanja elementa u uređenu jednostruko povezanu listu je $O(1)$.



5. Na datom stablu listovi su čvorovi *D,M,N, F, J,K* i *L*.
6. Na datom stablu koren je čvor *A*.
7. Na datom stablu roditelj čvora *C* je čvor *A*.
8. Na datom stablu deca čvora *C* su čvorovi *F*, *G* i *H*.
9. Na datom stablu preci čvora *E* su čvorovi *B* i *A*, a potomci čvora *E* su čvorovi *I*, *M* i *N*.

Definicija: Ako u stablu postoji put od čvora *x* do čvora *y*, tada je čvor *x* predak čvora *y*, dok čvor *y* je potomak čvora *x*

10. Na datom stablu dubina čvora *C* je 1.

Definicija: Dužna putanje od čvora *x* do čvora *y* jednaka broju grana koje smo sledili duž te putanje. Dubina čvora je dužina jedinstvene putanje od korena do tog čvora.

11. Na datom stablu visina čvora *C* je 2.

Definicija: Visina stabla *S*, koja se označava sa $h(S)$ je dužina najduže putanje od korena do nekog lista. Visina čvora je visina podstabla u čijem korenu se nalazi taj čvor.

12. Rezultat preorder (KLD) obilaska datog stabla je *A,B,D,E, I,M,N,C, F, G, J,K,H,L*
13. Rezultat inorder (LKD) obilaska datog stabla je: *D,B,M, I,N,E,A, F,C, J,G,K,L,H*
14. Rezultat postorder (LDK) obilaska datog stabla je: *D,M,N, I,E,B, F, J,K,G,L, H,C,A*
15. Rezultat levelorder (po nivoima) obilaska datog stabla je: *A,B,C,D,E, F, G,H, I, J,K, L,M,N*

3 Strukture podataka: binarno stablo pretrage, hip, heš tabele

3.1 Binarna stabla pretrage (BSP)

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.

1. 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.
```

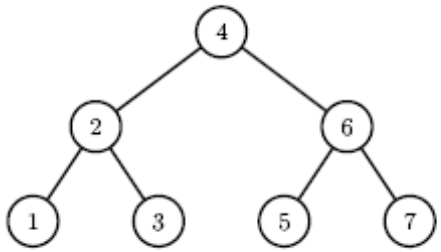
```
*/

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 */
}
```

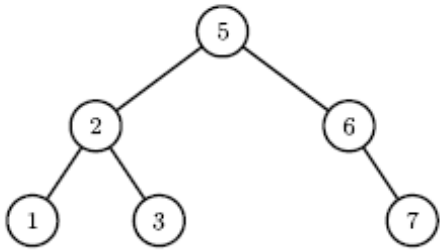
2. Za sekvencu ključeva 4, 2, 1, 3, 6, 5, 7 odredite binarno stablo pretraživanja koje se dobija kada se u početno prazno stablo ti čvorovi dodaju jedan po jedan u datom redosledu.

Rešenje



3. Skicirajte stablo koje se dobija kada se ukloni koren sa stabla iz prethodnog zadatka.
Podsetite se funkcije *izost_u(Stablo koren, int b)* iz prvog zadatka.

Rešenje



4. 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.

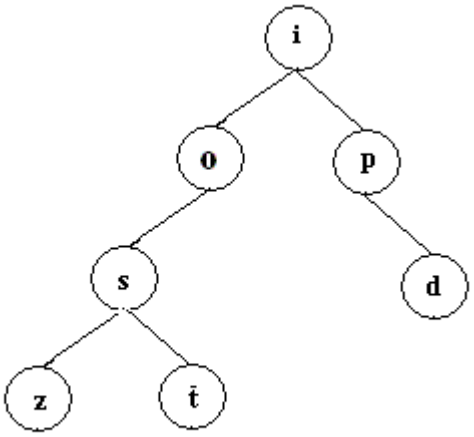
2. Ako je *n* ukupan broj čvorova i ako je *i* pozicija korena u nisci LKD, onda je

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*)



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

6. **U čvoru binarnog stabla zapisano je slovo engleske abecede. Ispisati rezultat POSTORDER obilaska, ako je rezultat INORDER (LKD) obilaska *deacbgfhjlkmi,* a PREORDER (KLD) obilaska *hgedcabfijklm***

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

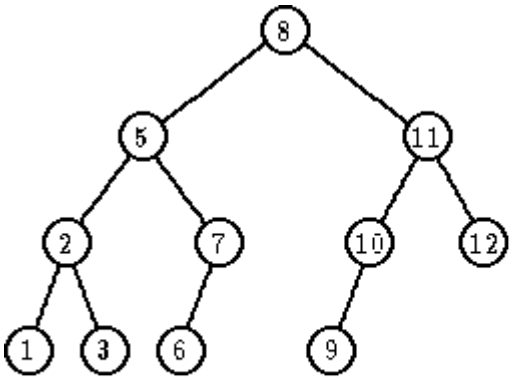
B

Stablo A

B

9. Elementi skupa A (međusobno različiti) smešteni su u BSP. Konstruisati algoritam koji za dati element $a \in A$ određuje sledeći po veličini element skupa A. Algoritam treba da bude složenosti $O(h)$, gde je h visina stabla.

Rešenje:
Pretpostavimo da je stablo građeno tako da je informaciono polje (ključ) desnog sina veće od informacionog polja (ključa) oca.



Sledbenik zadatog čvora je čvor sa najmanjim ključem koji je veći od ključa datog čvora.
Nađite sledbenika za a=1, a=2, a=3, a=5, a=7, a=11, a=12
Da li je sledbenik potomak u stablu ili predak?

Koji čvor nema sledbenika? Samo najdešnji čvor u stablu (koji nema desnog sina i koji je desni sin svog oca), jer je on čvor sa maksimalnom vrednošću ključa. (Na primer a=12)

Karakteristični slučajevi za čvor *v* čije informaciono polje (ključ) ima vrednost elementa ***a*** iz skupa ***S*** su:

1. *v* ima desno podstablo

   => Sledbenik se traži kao najmanji u desnom podstablu (ključ najlevljeg čvora u desnom podstablu)

   Na primer: a=8 ili a=5 ili a=11

2. *v* nema desnog sina

   => Sledbenik se traži tako što se traži ključ najnižeg pretka *w* takvog da levi sin od *w* je predak čvora *v*.

   Ova operacija se najefikasnije izvodi ako se za svaki čvor čuva i polje *otac* sa pokazivačem na oca čvora. Tako se ide od čvora *v* prema korenu sve dok se ne nađe čvor *w* koji je levi sin svog oca. U slučaju da nema takvog pretka *w*, onda je *a* najveći element skupa i nema svog sledbenika.

   Na primer: a=12 ili a=3

U svakoj situaciji, sledbenik se nalazi bez poređenja ključeva zahvaljujući principu uređenosti stabla.

Sledi algoritam za nalaženje sledbenika za zadatu vrednost ključa čvora ukazanog sa *v* koji vraća adresu sledbenika ili NULL. Koristi se i poziv algoritma BST_MIN koji u BST (stablu binarnog pretraživanja) traži ključ sa minimalnom vrednošću.

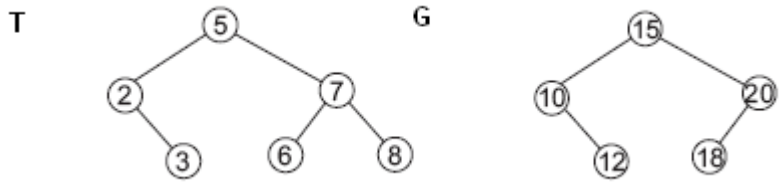
Algoritam BST_MIN za BST čiji je koren ukazan argumentom *node* ide po lancu levih pokazivača sve dok ne dođe do čvora koji nema levog sina. Prikazan je u prvom zadatku kao funkcija min_u.

Algoritam BST_MIN pokriva 1. situaciju kada je potrebno pronači ključ sa minimalnom vrednošću u desnom podstablu čvora *v*.

```
Algoritam BST_sledbenik(v)
Ulaz v (čvor v)
Izlaz čvor q ili NULL ako ne postoji
{
pom = v;
if (pom- > Desni! = NULL) return BST_min(pom- > Desni);
else q= pom- > Otac;
while (q! = NULL && pom == q- > Desni) {
     pom = q;
   q = pom- > Otac;  //pom = koren => q=NULL, situacija kod a=12
}
return q;
}
Algoritam BST_min(Koren)
Ulaz Koren (pokazivač na koren BSP)
Izlaz čvor sa minimalnom vrednošću ključa
{
  p = Koren;
  while (p- > levi! = NULL) p = p- > Levi;
  return p;
}
```

10. Konkatenacija je operacija nad dva skupa koja zadovoljavaju uslov da su svi ključevi u jednom skupu manji od svih ključeva u drugom skupu. Rezultat konkatenacije je unija skupova. Konstruisati algoritam za konkatenaciju dva binarna stabla pretrage u jedno stablo. Vremenska složenost algoritma mora biti $O(h)$ u najgorem slučaju, gde je h veća od visina dva stabla.

REŠENJE:



Neka su zadata dva BSP stabla T, G, tako da svi ključevi stabla G su veći od svih ključeva stabla T. Neka (bez smanjenja opštosti) visina h stabla G nije manja od visine stabla T.

1. Pronaći u stablu G čvor *v* sa najmanjim ključem (npr, sve dok je moguće spuštati se niz stablo polazeći od korena ulevo i to se može obaviti za O(h) vremena u najgorem slučaju)

2. Ukloniti čvor *v* iz stabla G (O(h) vremena u najgorem slučaju)

3. Formirati za O(h) vremena novo stablo sa korenom *v* čija podstabla su stabla T (levo) i stablo G bez čvora *v* (desno)). Novoformirano stablo ostaje binarno stablo pretrage, jer koren, tj. čvor *v* je sa najmanjim ključem u G, tako da desno podstablo može biti G. S druge strane, po uslovu s početka, svi ključevi stabla G (pa i *v*) su veći od svih ključeva stabla T, te T može biti levo podstablo.

11. Napisati C potprogram kojim se proverava da li su dva zadata binarna stabla ekvivalentna po strukturi i sadržaju unutar čvorova.

SKICA RADA

Problem se moze resavati rekurzivnim nacinom razmisljanja.
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) );

}
```

12. Napisati C potprogram kojim se proverava da u datom binarnom stablu d1 postoji podstablo koji je u smislu prethodnog zadatka ekvivalentno po strukturi i sadržaju sa zadatim stablom d2. Vratiti ili pokazivac na nadjeno podstablo ili NULL u suprotnom.

```
int podstablo(Stablo d1, Stablo d2){

  if (EKV(d1,d2)) return d1;

  /*inace, ako d1 nije EKV sa d2, a d1 je neprazno podstblo, onda ...*/

  if (d1){

     Stablo d=podstablo(d1->levo,d2);

     /*...  ako u levom podstablu od d1 postoji EKV stablo sa d2 */

     if (d) return d;

     /* u suprotnom trazi se EKV podstablo u desnom podstablu od d1*/

     else return podstablo(d1->desno,d2);
```

```
    }

  return NULL;

}
```

13. Da li je operacija brisanja čvora komutativna u smislu da brisanje čvora *x,* te potom *y* iz BST daje stablo koje je jednako onom koje se dobija brisanjem čvora *y*, a zatim čvora *x*? Obrazložiti odgovor ili dati kontraprimer.