UNIVERZITET U BEOGRADU

FAKULTET ORGANIZACIONIH NAUKA

STRUKTURE PODATAKA I ALGORITMI

– ZBIRKA ZADATAKA –

2011.

NAPOMENA:

Ovaj materijal nastao je kao rezultat pripreme za polaganje ispita kolege Dejana Ilića. Deo materijala je skinut sa Interneta (forum FON i drugi domaci i strani sajtovi), dok je drugi deo sam napisao.

Zahvaljujemo se kolegi Dejanu Iliću i studentima koji su pomogli u ispravci zadataka i kreiranju ove zbirke.

ISPRAVNI ZADACI

ZADACI SA NIZOVIMA

Ubacivanje elementa u rastuce sortiran niz

```
public void InsertSortArray(int value, int[] niz) throws RuntimeException {
     if (brEleme == niz.length)
        throw new RuntimeException();
     int j = 0; // pozicija na koju se ubacuje nova vrednsto
     while (j < brEleme && value > niz[j])
        j++;
     brEleme++;
     for (int k = brEleme; k > j; k--)
        niz[k] = niz[k - 1];
     niz[j] = value;
  }
```

Metoda int prosekParnih(int [] niz) koja računa i vraća prosek parnih elemenata niza (onih koji imaju parne vrednosti, a ne na parnim mestima u nizu).

```
public double prosekParnih(int[] niz) {
     double zbir = 0;
     double broj = 0;
     for (int i = 0; i <= niz.length - 1; i++) {
        if (niz[i] % 2 == 0) {
           zbir += niz[i];
           broj++;
        }
     }
     if (broj == 0)
        return 0;
     else
        return zbir / broj;
  }
```

PRETRAŽIVANJA

Sekvencijalno – iterativno

```
public static int sekvencijalno(int podatak, int[] n) {
     for (int i = 0; i < n.length; i++) {
        if (n[i] == podatak)
           return i;
     }
     return -1;
  }
```

Sekvencijalno – rekurzivno

```
public static int sekvencijalnoR(int podatak, int[] n, int i) {
     if (i > -1 && i < n.length) {
        if (n[i] == podatak)
           return i;
        return Pretrazivanje.sekvencijalnoR(podatak, n, ++i);
     }
     return -1;
  }
```

Binarno – iterativno

```
public static int binarno(int podatak, int[] n) {
     int l = 0;
     int d = n.length - 1;

     while (l <= d) {
        int s = (l + d) / 2;
        if (n[s] == podatak)
           return s;
        if (n[s] > podatak)
           d = s - 1;
        else
           l = s + 1;
     }
     return -1;
  }
```

Binarno – rekurzivno

```
public static int binarnoR(int podatak, int[] n, int l, int d) {
     if (l <= d) {
        int s = (l + d) / 2;
        if (n[s] == podatak)
           return s;
        if (n[s] > podatak)
           return binarnoR(podatak, n, l, s - 1);
        else
           return binarnoR(podatak, n, s + 1, d);
     }
     return -1;
  }
```

Interpolaciono – iterativno

```
public static int interpolaciono(int podatak, int[] n) {
     int l = 0;
     int d = n.length - 1;

     while (l <= d) {
        int index = l + (d - l) * (podatak - n[l]) / (n[d] - n[l]);
        if (n[index] == podatak)
           return index;
        if (n[index] > podatak)
           d = index - 1;
        else
           l = index + 1;
     }
     return -1;
  }
```

Interpolaciono – rekurzivno

```
public static int interpolacionoR(int podatak, int[] n, int l, int d) {
     if (l <= d) {
        int index = l + (d - l) * (podatak - n[l]) / (n[d] - n[l]);
        if (n[index] == podatak)
           return index;
        if (n[index] > podatak)
           return interpolacionoR(podatak, n, l, index - 1);
        else
           return interpolacionoR(podatak, n, index + 1, d);
     }
     return -1;
  }
```

JEDNOSTRUKO-SPREGNUTA LISTA

Metoda int Prebroj(CvorListe Vrh) koja će vratiti koliko elemenata liste ima vrednost veću od elementa na početku liste.

```
public int prebroj(CvorJSListe pocetak) {
     if (pocetak == null) {
        return 0;
     }
     int broj = 0;
     CvorJSListe pom = pocetak;
     while (pom != null) {
        if (pom.podatak > pocetak.podatak) {
           broj++;
        }
        pom = pom.sledeci;
     }
     return broj;
  }
```

Metoda koja vraća broj ponavljanja zadate vrednosti u listi.

```
public int brPonavljanja(CvorJSListe prvi, int pod) {
     if (prvi == null)
        return 0;
     int broj = 0;
     CvorJSListe pom = prvi;
     while (pom != null) {
        if (pom.podatak == pod)
           broj++;
        pom = pom.sledeci;
     }
     return broj;
  }
```

Metoda koja izbacuje element koji se nalazi nakon elementa sa najmanjom vrednošću.

```
public int posleNajmanjeg(CvorJSListe prvi) {
     if (prvi == null)
        return Integer.MIN_VALUE;
     CvorJSListe min = minElement(prvi);

     if (min.sledeci == null)
        return Integer.MIN_VALUE;

     broj = min.sledeci.podatak;
     min.sledeci = min.sledeci.sledeci;
     return broj;
  }
```

Broj pozitivnih elemenata – rekurzivno

```
public int izbrojPozitivne(CvorJSListe pom) {
     if (pom == null)
        return 0;
     else {
        if (pom.podatak > 0)
           return 1 + izbrojPozitivne(pom.sledeci);
        else
           return izbrojPozitivne(pom.sledeci);
     }
  }
```

Metoda koja vraća invertovanu JSListu

```
public CvorJSListe invertuj(CvorJSListe p1){
     CvorJSListe p2 = null;
     while(p1 != null){
        p2 = new CvorJSListe(p1.podatak, p2);
        p1 = p1.sledeci;
     }
     return p2;
  }
```

Metoda koja koja pravi indentičnu kopiju date JSListe

```
public CvorJSListe klonirana(CvorJSListe pom) {
     if (pom != null) {
        CvorJSListe novi = new CvorJSListe(pom.podatak, klonirana(pom.sledeci));
        return novi;
     } else {
        return null;
     }
  }
```

Metoda koja od JSListe pravi niz

```
public int[] transform(JSLista l) {
     int br = 0;
     CvorJSListe pomocni = l.prvi;
     while (pomocni != null) {
        br++;
        pomocni = pomocni.sledeci;
     }
     pomocni = l.prvi;
     int[] stack = new int[br];

     for (int i = 0; i < stack.length; i++) {
        stack[i] = pomocni.podatak;
        pomocni = pomocni.sledeci;
     }
     return stack;
  }
```

Metoda koja od dve JSListe vraća treću listu koja predstavlja uniju datih listi

```
public CvorJSListe unija(CvorJSListe c1, CvorJSListe c2) {
     if (c1 == null && c2 == null)
        return null;

     CvorJSListe novaLista = null;

     while (c1 != null) {
        if (!postoji(novaLista, c1.podatak)) {
           novaLista = new CvorJSListe(c1.podatak, novaLista);
        }
        c1 = c1.sledeci;
     }

     while (c2 != null) {
        if (!postoji(novaLista, c2.podatak)) {
           novaLista = new CvorJSListe(c2.podatak, novaLista);
        }
        c2 = c2.sledeci;
     }
     return novaLista;
  }
```

Metoda koja proverava da li postoji vrednost u JSListi

```
private boolean postoji(CvorJSListe c2, int p) {
     while (c2 != null) {
        if (c2.podatak == p) {
           return true;
        }
        c2 = c2.sledeci;
     }
     return false;
  }
```

DVOSTRUKO-SPREGNUTA LISTA

Metoda koja prebacuje čvor sa najvećom vrednošću na početak te liste.

```
public void prebaciMaxElement(CvorDSListe poslednji) {
     CvorDSListe max = poslednji;
     CvorDSListe pom = poslednji;

     while (pom.prethodni != null) {
        pom = pom.prethodni;
        if (pom.podatak > max.podatak) {
           max = pom;
        }
     }

     if(max.prethodni == null)
        return;

     max.prethodni.sledeci = max.sledeci;
     if(max.sledeci != null)
        max.sledeci.prethodni = max.prethodni;
     max.prethodni = null;
     max.sledeci = pom;
     pom.prethodni = max;
  }
```

Napisati metodu koja prikazuje (štampa na ekranu) onu polovinu liste (levo od p ili desno od p) koja ima veći zbir

```
public void veciZbir(CvorDSListe p) {
     CvorDSListe pom = p;
     int zbir1 = 0;
     int zbir2 = 0;
     while (pom != null) {
        zbir1 = zbir1 + pom.podatak;
        pom = pom.sledeci;
     }
     pom = p;
     while (pom != null) {
        zbir2 = zbir2 + pom.podatak;
        pom = pom.prethodni;
     }
     if (zbir1 > zbir2) {
        pom = p.sledeci;
        while (pom != null) {
           System.out.println(pom.podatak);
           pom = pom.sledeci;
        }
```

```
    } else {
       pom = p.prethodni;
       while (pom != null) {
          System.out.println(pom.podatak);
          pom = pom.prethodni;
       }
    }
 }
```

Dat je pokazivač na prvi element DSListe čiji čvorovi predstavljaju elemente reči (slovo ili prazno mesto). Napisati metodu koja proverava da li je data rečenica (reč) palindrom. Palindrom je reč (rečenica) koja se čita isto sa obe strane (i sleva na desno i s desna na levo).

```
public static boolean daLiJePalindromDS(CvorDSListe p) {
     if (p == null) {
        return false;
     }
     CvorDSListe pom = p;
     while (pom.sledeci != null) {
        pom = pom.sledeci;
     }

     while (p != pom && p.prethodni != pom) {
        if (p.podatak == ' ') {
           p = p.sledeci;
        }
        if (pom.podatak == ' ') {
           pom = pom.prethodni;
        }
        if (p.podatak == pom.podatak) {
           p = p.sledeci;
           pom = pom.prethodni;
        } else {
           return false;
        }
     }
     return true;
     }
```

Data je dvostruko spregnuta (DS) lista čiji su elementi čvorova pokazivači na početak jednostruko spregnute (JS) liste. Napisati klasu koja opisuje čvor ovakve DS liste, a zatim napisati algoritam za ubacivanje novog elementa u ovako definisanu strukturu, koji funkcioniše po sledećem principu: kreće se od početka DS liste. Ako je element koji se ubacuje manji od prvog elementa JS liste trenutnog čvora DS liste, onda se taj element ubacuje na kraj te JS liste. U suprotnom, prelazi se na sledeći čvor DS liste i algoritam se ponavlja. Ako se stigne do kraja DS liste, onda se kreira novi čvor i u njegovu JS listu se ubacuje novi element. Početna metoda prihvata pokazivač na početak DS liste i ceo broj koji se ubacuje.

```
public class CvorListe {
     private CvorListe sledeci;
     private CvorListe prethodni;
     private CvorJSListe prvi;

     public void ubaci(CvorListe p, int pod) {
        if (p == null) {

           p = new CvorListe();
           p.prvi = new CvorJSListe(pod, null);
        } else {
           while (p.sledeci != null && p.prvi.podatak <= pod) {
              p = p.sledeci;
           }
```

```
            if (p.prvi.podatak <= pod) {
               p.sledeci = new CvorListe();
               p.sledeci.prethodni = p;
               p.sledeci.prvi = new CvorJSListe(pod, null);
            } else {
               CvorJSListe pom = prvi;
               while (pom.sledeci != null) {
                  pom = pom.sledeci;
               }
               pom.sledeci = new CvorJSListe(pod, null);
            }
         }
      }

   }
```

Dat je pokazivac na neki element DS liste celih brojeva. Napisati funkciju koja ce izbaciti iz liste onaj element koji sadrži najmanju vrednost u listi.

```
public int izbaciMin(CvorDSListe cvor) {
      if (cvor == null)
         return Integer.MAX_VALUE;

      while (cvor.prethodni != null) {
         cvor = cvor.prethodni;
      }
      CvorDSListe min = minElement(cvor);
      if (min.prethodni != null)
         min.prethodni.sledeci = min.sledeci;
      if (min.sledeci != null)
         min.sledeci.prethodni = min.prethodni;
      return min.podatak;
   }
```

Dat je pokazivač na prvi element DS liste. Napisati metodu koja vraća pokazivač na element sa najmanjom vrednošću u listi.

```
private CvorDSListe minElement(CvorDSListe cvor) {
      CvorDSListe min = cvor;
      while (cvor != null) {
         if (cvor.podatak < min.podatak)
            min = cvor;
         cvor = cvor.sledeci;
      }
      return min;
   }
```

Dat je pokazivac na pocetni cvor DS liste sortirane u rastucem redosledu koja sadrži pozitivne cele brojeve. Napisati funkciju koja ce izmedju svih onih elementa liste koji se po vrednosti razlikuju za više od 1 ubaciti u datu listu nove elemente tako da lista posle poziva operacije ima u sebi sukcesivne cele brojeve. Na primer ako lista sadrži 3, 5, 8 nakon poziva ove opercije sadržace 3, 4, 5, 6, 7, 8

```
public void popuni(CvorDSListe prvi) {
      if (prvi == null || prvi.sledeci == null)
         return;
      if ((prvi.sledeci.podatak - prvi.podatak) > 1) {
         CvorDSListe novi = new CvorDSListe(prvi.podatak + 1, prvi, prvi.sledeci);
         prvi.sledeci.prethodni = novi;
         prvi.sledeci = novi;
      }
      popuni(prvi.sledeci);
   }
```

Dat je pokazivac na neki cvor DS liste celih brojeva koja sigurno sadrži najmanje 4 elementa. Napisati funkciju ubaciNti(int A, int N) koja ce ubaciti novi element sa sadržajem A i to tako da on bude na N-toj poziciji od pocetka.

```
public void ubaciN(CvorDSListe cvor, int a, int n) {
     if (cvor == null)
        return;
     while (cvor.prethodni != null)
        cvor = cvor.prethodni;
     CvorDSListe pom = new CvorDSListe(a, null, null);
     if (n < 0) {
        return;
     }
     if (n == 0) {
        pom.prethodni = null;
        pom.sledeci = cvor;
        if (pom.sledeci != null)
           pom.sledeci.prethodni = pom;
     }
     int i = 0;
     while (i++ < n - 1) {
        if (cvor.sledeci == null)
           return;
        cvor = cvor.sledeci;
     }
     pom.prethodni = cvor;
     pom.sledeci = cvor.sledeci;
     if (cvor.sledeci != null)
        cvor.sledeci.prethodni = pom;
     cvor.sledeci = pom;
  }
```

Metoda koja invertuje dvostruko-spregnutu listu.

```
public CvorDSListe invertujLIstu(CvorDSListe cvor) {
     CvorDSListe c = null;
     while (cvor != null) {
        c = new CvorDSListe(cvor.podatak, null, c);
        if (c.sledeci != null)
           c.sledeci.prethodni = c;
        cvor = cvor.sledeci;
     }
     return c;
  }
```

Metoda koja vraća DSListu koja predstavlja razliku (skupova) dve DSListe

```
public CvorDSListe razlika(CvorDSListe c1, CvorDSListe c2) {
     if (c1 == null || c2 == null)
        return c1;
     CvorDSListe novi = null;
     CvorDSListe pom = c1;
     while (pom != null) {
        if (!pronadji(pom.podatak, c2)) {
           novi = new CvorDSListe(pom.podatak, null, novi);
           if (novi.sledeci != null) {
              novi.sledeci.prethodni = novi;
           }
        }
        pom = pom.sledeci;
     }
     return novi;
  }
```

Metoda koja poslednji element DSListe prebacuje na početak te DSListe. Dati su pokazivačni na prvi i poslednji element

```
public void prebaciPoslednjiNaPocetak() {
     if (prvi == null) {
        return;
     }
     if (prvi.sledeci == null) {
        return;
     }
     CvorDSListe pom = poslednji;
     poslednji = pom.prethodni;
     pom.sledeci = prvi;
     prvi.prethodni = pom;
     pom.prethodni.sledeci = null;
     pom.prethodni = null;
     prvi = pom;
  }
```

Metoda koja računa zbir elemenata koji se ponavljaju. Primer: Ulaz 5 7 2 2 5 2. Rezultat je 2+5=7

```
public int ZbirDuplih(CvorDSListe p1) {
     if (p1 == null)
        return 0;
     CvorDSListe tek = p1;
     CvorDSListe tek1 = p1.sledeci;
     CvorDSListe tek2 = tek.prethodni;
     int ima = 0;
     int suma = 0;
     for (; tek != null; tek = tek.sledeci) {
        for (; tek2 != null; tek2 = tek2.prethodni) {
           if (tek.podatak == tek2.podatak) {
              ima = 1;
              break;
           }
        }
        tek2 = tek;
        if (ima == 0) {
           for (; tek1.sledeci != null; tek1 = tek1.sledeci) {
              if (tek.podatak == tek1.podatak) {
                 suma = suma + tek.podatak;
                 break;
              }
           }
        }
        if (tek1.sledeci != null) {
           tek1 = tek1.sledeci;
        }
     }
     return suma;
  }
```

Metoda koja vraća pokazivač na prvi element ciklične DSListe sortirane u rastućem redosledu. Dat je pokazivač na neki element u listi.

```
public CvorDSListe vratiPokNaPrviEl(CvorDSListe p) {
     if (p == null || (p.sledeci == p && p.prethodni == p))
        return p;
     CvorDSListe pom = p;
     while (pom.sledeci.podatak > pom.podatak
          && pom.prethodni.podatak < pom.podatak)
        pom = pom.sledeci;
     return pom.sledeci;
  }
```

STABLA

Stampa cvorove binarnog stabla od korena do čvora koji ima najveću vrednost u stablu

```
public void printToMax(CvorStabla koren) {
     if (koren == null)
        return;
     System.out.println(koren.podatak);
     if (maxElement(koren) == koren.podatak)
        return;
     if (maxElement(koren.levo) > maxElement(koren.desno))
        printToMax(koren.levo);
     else
        printToMax(koren.desno);
  }
```

Dat je pokazivač na koren binarnog stabla. Napisati metodu koja računa zbir vrednosti cvorova u stablu.

```
public int zbir(CvorStabla koren) {
     if (koren == null)
        return 0;
     return koren.podatak + zbir(koren.desno) + zbir(koren.levo);
  }
```

Dat je pokazivač na koren binarnog stabla. Napisati metodu koja vraća broj elemenata u stablu.

```
public int preboj(CvorStabla koren) {
     if (koren == null)
        return 0;
     return 1 + preboj(koren.levo) + preboj(koren.desno);
  }
```

Dat je pokazivač na koren binarnog stabla. Napisati metodu koja računa prosek vrednosti cvorova u stablu.

```
public double prosek(CvorStabla koren) {
     if (koren == null)
        return 0;
     double zbir = zbir(koren);
     int clan = preboj(koren);
     return zbir / clan;
  }
```

Dat je pokazivač na koren binarnog stabla. Napisati metodu koja vraća roditelja čvora koji sadrži zadatu vredost.

```
public CvorStabla nadjiRoditelja(CvorStabla tekuci, int podatak) {
     if (tekuci == null || tekuci.podatak == podatak)
        return null;
     if ((tekuci.levo != null && tekuci.levo.podatak == podatak)
           || (tekuci.desno != null && tekuci.desno.podatak == podatak))
        return tekuci;
     CvorStabla s = pronadji(tekuci.levo, podatak);
     if (s != null)
        return nadjiRoditelja(tekuci.levo, podatak);
     return nadjiRoditelja(tekuci.desno, podatak);
  }
```

Dat je pokazivač na koren binarnog stabla i na neki čvor u stablu. Napisati metodu koja ispisuje putanju od tog čvora do korena datog stabla.

```
public void ispisiPutanju(CvorStabla cvor, CvorStabla koren) {
     if (koren == null || cvor == null)
        return;
     System.out.println(cvor.podatak);
     ispisiPutanju(nadjiRoditelja(koren, cvor.podatak), koren);
  }
```

Dat je pokazivač na koren BST stabla i na neki čvor u stablu. Napisati metodu koja ispisuje putanju od korena do tog čvora datog stabla.

```
public void ispisiPutanju(CvorStabla tekuci, CvorStabla kraj) {
     System.out.println(tekuci.podatak);
     if (tekuci == kraj)
        return;
     if (kraj.podatak < tekuci.podatak)
        ispisiPutanju(tekuci.levo, kraj);
     else
        ispisiPutanju(tekuci.desno, kraj);
  }
```

Dat je pokazivač na koren binarnog stabla. Napisati metodu koja pronalazi najmanju vrednost u stablu.

```
public int minVrednost(CvorStabla cvor) {
     if (cvor == null)
        return Integer.MAX_VALUE;

     return Math.min(cvor.podatak,
          Math.min(minVrednost(cvor.levo), minVrednost(cvor.desno)));
  }
```

Dat je pokazivač na koren binarnog stabla. Napisati metodu koja pronalazi najveću vrednost u stablu.

```
public int maxVrednost(CvorStabla cvor) {
     if (cvor == null)
        return Integer.MIN_VALUE;

     return Math.max(cvor.podatak,
          Math.max(maxVrednost(cvor.levo), maxVrednost(cvor.desno)));
  }
```

Dat je pokazivač na koren binarnog stabla i pokazivač na neki čvor u tom stablu. Napisati metodu koja štampa elemente na putanji od korena do datog čvora.

```
public void odstampajPutanjuOdKorenaDoCvora(CvorStabla k, CvorStabla t) {
     if (k == null || t == null)
        return;
     System.out.println(k.podatak);
     if (k == t)
        return;
     boolean levo = daLiJeUStablu(k.levo, t);
     if (levo)
        odstampajPutanjuOdKorenaDoCvora(k.levo, t);
     else
        odstampajPutanjuOdKorenaDoCvora(k.desno, t);
  }
```

Dat je pokazivač na koren binarnog stabla i pokazivač na neki čvor u tom stablu. Napisati metodu koja štampa elemente na putanji od datog čvora do korena.

```
public void odstampajPutanjuOdCvoraDoKorena(CvorStabla k, CvorStabla t) {
     if (k == null || t == null)
        return;
     System.out.println(t.podatak);
     CvorStabla r = nadjiRoditelja(k, t.podatak);
     odstampajPutanjuOdCvoraDoKorena(k, r);
  }
```

Dat je pokazivač na koren binarnog stabla. Napisati metodu koja vraća pokazivač na čvor na najvećoj dubini.

```
public CvorStabla najdubljiCvor(CvorStabla cvor) {
     if (cvor == null)
        return null;
     if (cvor.levo == null && cvor.desno == null)
        return cvor;
     if (visina(cvor.levo) > visina(cvor.desno))
        return najdubljiCvor(cvor.levo);
     else
        return najdubljiCvor(cvor.desno);
  }
```

Dat je pokazivač na koren binarnog stabla. Napisati metodu koja vraća visinu stabla.

```
int visina(CvorStabla cvor) {
     if (cvor == null)
        return 0;

     return 1 + Math.max(visina(cvor.levo), visina(cvor.desno));
  }
```

Dat je pokazivač na koren binarnog stabla. Napisati metodu koja proverava da li je stablo HEAP.

```
public boolean daLiJeHeap(CvorStabla cvor) {
     if (cvor == null)
        return true;
     if (cvor.podatak > maxVrednost(cvor.levo)
          && cvor.podatak > maxVrednost(cvor.desno)) {
        return daLiJeHeap(cvor.levo) && daLiJeHeap(cvor.desno);
     }
     return false;
  }
```

ZADACI SA GREŠKAMA

SORTIRANJE

Selection sort

```
public void SelectionSort(int[] niz)
{
   for (int i = 0; i < niz.length - 1; i++)
   {
      int min = i;
      for (int j = i + 1; j < niz.length; j++)
      {
         if (niz[j] < niz[min])
            min = j;
      }
      if(min!=i) {
          int pom = niz[i];
          niz[i] = niz[min];
          niz[min] = pom;
      }
   }
}
```

Bubble sort

```
public void BubbleSort(int[] niz)
{
    for(int i=niz.length-1; i>0; i--) {
        for(int j=0; j<i; j++) {
            if(niz[j]>niz[j+1]) {
                int pom = niz[j];
                niz[j] = niz[j+1];
                niz[j+1] = pom;
            }
        }
    }
}
```

Insertion sort

```
public void InsertionSort(int[] niz)
{
  for (int i = 1; i < niz.length; i++)
  {
    int pom = niz[i];
    int j = i;
    while ((j > 0) && (niz[j-1] > pom))
    {
      niz[j] = niz[j-1];
      j--;
    }
    niz[j] = pom;
  }
}
```

Shell sort

```
public void ShellSort(int[] niz)
{
    int inc = niz.length/2;
    while(inc > 0) {
        for (int i = 1; i < niz.length; i++)
        {
            int pom = niz[i];
            int j = i;
            while ((j >=inc) && (niz[j-inc] > pom))
            {
                niz[j] = niz[j-inc];
                j-=inc;
            }
            niz[j] = pom;
        }
        inc = inc/2;
    }
}
```

Quick Sort

```
public void QuickSort(int[] niz, int low, int high) {
    if(high<=low) return;
    int pivot = partition(niz, low, high);
    QuickSort(niz, low, pivot-1);
    QuickSort(niz, pivot+1, high);
}

public int partition(int[] niz, int low, int high) {
    int i = low – 1;
    int j = high;
    int pivot = a[high];
    for(;;) {
        while(niz[++i]<pivot);
        while(niz[--j]>pivot) if (j==low) break;
        if(i>=j) break;
        zameni(niz, i, j);
    }
    zameni(niz, i, r);
    return(i);
}

public void zameni(int[] niz, int i, int j) {
    int pom = niz[i];
    niz[i] = niz[j];
    niz[j] = pom;
}
```

Pokazivac na cvor binarnog stabla na najmanjoj dubini

Dat je pokazivac na koren binarnog stabla ciji cvorovi sadrže cele brojeve. Napisati funkciju koja ce vratiti pokazivac na cvor list koji je na najmanjoj dubini u stablu.

```
public static Cvor najpliciCvor(Cvor koren) {
    if (koren == null)
        return null;
    /*ako je to list, znaci nasli smo ga*/
    if (koren.levo == null && koren.desno == null)
        return koren;
    /*ako nije to list, idi ka plicem listu*/
    if (firstLeafLevel(koren.levo) < firstLeafLevel(koren.desno))
        return najpliciCvor (koren.levo);
    else
        return najpliciCvor (koren.desno);
    }

    private static int firstLeafLevel(Cvor koren) {
        if (koren == null) /*na ovaj nacin ignorisu se cvorovi kojima je samo jedno dete null */
            return Integer.MAX_VALUE;
    /*ako je to list, nasli smo ga, vrati 0*/
    if (koren.levo == null && koren.desno == null)
        return 0;
    /*ako nije to list, trazi dalje*/
    return 1 + Math.min(firstLeafLevel(koren.desno), firstLeafLevel(koren.levo));
}
```

Broj cvorova binarnog stabla koji su veci od svih svojih potomaka

Dat je pokazivac na koren binarnog stabla ciji cvorovi sadrže cele brojeve. Napišite funkciju koja ce vratiti broj cvorova koji su po sadržaju veci od sadržaja svih svojih potomoka.

```
public static int brojVecih(Cvor koren) {
    if (koren == null)
        return 0;
    if (koren.levo == null && koren.desno == null)
        return 0;// ignorisi listove
    int increment = 0;
    if (koren.data > max(koren.levo) && koren.data > max(koren.desno))
        increment = 1;
    return increment + brojVecih (koren.levo) + brojVecih (koren.desno);
}
private static int max(Cvor koren) {
    if (koren == null)
    return Integer.MIN_VALUE;
    return Math.max(koren.data, Math.max(max(koren.levo), max(koren.desno)));
}
```

Nivo datog cvora binarnog stabla

Napišite funkciju int nivo(cvor k, cvor p) koja prihvata pokazivac na koren binarnog stabla i pokazivac na neki cvor u stablu i vraca nivo na kome se pokazani cvor nalazi.

```
public static int nivo(Cvor koren, Cvor n, int level) {
    if (koren == null)
        return -1;
    if (koren == n)
        return level;
    int temp = nivo(koren.levo, n, level + 1);
    if (temp != -1)
        return temp;
    return nivo(koren.desno, n, level + 1);
}
```

Provera da li su dva binarna stabla identična

```
public static boolean identicnaStabla(Cvor k1, Cvor k2) {
    if (k1 == null && k2 == null)
       return true;
    if (k1 != null && k2 != null) {
       if (k1.podatak == k2.podatak)
          return identicnaStabla(k1.levo, k2.levo) && identicnaStabla(k1.desno, k2.desno);
    }
    return false;
}
```

Broj cvorova sa sadrzajem vecim od zadatog broja

```
public static int izbrojVece(Cvor c, int v){
    if (c == null)
      return 0;
    if (c.podatak > v)
      return 1 + izbrojVece (c.levo, v) + izbrojVece (c.desno, v);
    else
      return izbrojVece (c.levo, v) + izbrojVece (c.desno, v);
}
```

Prosek elemenata desnog podstabla datog cvora

```
public static double Prosek(Cvor t) {
  if(t == null && t.desno ==null)
     return 0;
  int zbir = Zbir(t.desno);
  int brElemenata = Prebroj(t.desno);
  double prosek = zbir/brElemenata;
  return prosek;
}
public static int Zbir(Cvor c) {
  if(c == null)
     return 0;
  return c.pod + Zbir(c.levo) + Zbir(c.desno);
}
public static int Prebroj(Cvor c){
    if (c == null)
      return 0;
    return 1 + Prebroj (c.levo) + Prebroj (c.desno);
}
```

Uredjuje binarno stablo da cvor bude manji od njegove dece

Dat je pokazivac na koren binarnog stabla celih brojeva. Napisite algoritam koji ce dato stablo urediti tako da za svaki cvor vazi da je element u njemu manji od elemenata njegove dece (misli se samo na njegovo levo I desno dete).

```
public void Uredi(Cvor c)
{
    if(c==null)
       break;

    if(c.Data>c.levo.Data)
    {
        int pom=c.Data;
        c.Data=c.levo.Data;
        c.levo.Data=pom;
    }
    if(c.Data>c.desno.Data)
    {
        int pom=c.Data;
        c.Data=c.desno.Data;
        c.desno.Data=pom;
    }
    Uredi(c.levo);
    Uredi(c.desno);
}
```

Stampa cvorove binarnog stabla od korena do cvora koji ima najmanju vrednost u stablu

Napisati proceduru koja štampa sadržaj svih cvorova binarnog stabla na putanji od korena do cvora koji ima najmanju vrednost u stablu.

```
public static void printToMin(Cvor koren) {
    if (koren == null)
        return;
    System.out.println(koren.data);
    if (min(koren) == koren.data)
        return;
    if (min(koren.levo) < min(koren.desno))
        printToMin(koren.levo);
    else
        printToMin(koren.desno);
}

private static int min(Cvor koren) {
    if (koren == null)
        return Integer.MAX_VALUE;
    return Math.min(koren.data, Math.min(min(koren.levo), min(koren.desno)));
}
```

Broj cvorova koji ne ispunjavanju uslove za AVL stablo

Dat je pokazivac na koren binarnog stabla. Napišite funkciju koja ce vratiti broj cvorova koji ne ispunjavaju uslove za AVL stablo.

```
public static int howManyNonAVL(Cvor koren) {
    if (koren == null)
        return 0;
    int balanceFactor = height(koren.levo) - height(koren.desno);
    if (!(isBSTCvor(koren) && balanceFactor >= -1 && balanceFactor <= 1))
        return 1 + howManyNonAVL(koren.levo) + howManyNonAVL(koren.desno);
    else
        return howManyNonAVL(koren.levo) + howManyNonAVL(koren.desno);
}
```

Zbir putanje u AVL stablu

```
public static int zbirPutanje(Cvor a, Cvor b) {
    int zbir = 0;
    while (a.podatak != b.podatak) {
      zbir += a.podatak;
      if (a.podatak > b.podatak)
        a = a.levo;
      else if (a.podatak < b.podatak)
        a = a.desno;
    }
    zbir += b.podatak;
    return zbir;
}
```

Stampa cvorove na putanji od korena do datog cvora AVL stabla

Dat je pokazivac na koren AVL stabla ciji cvorovi sadrže cele brojeve i drugi pokazivac na neki cvor u stablu. Napisati funkciju koja ce odštampati sve cvorove koji su na putanji od korena do datog cvora.

```
public static void printRootToCvor(Cvor koren, Cvor cvor) {
    if (koren == null) {
        System.out.println("Not found");
        return;
    }
    if (koren.data == cvor.data) {
        System.out.println("found: " + koren.data);
        return;
    } else
        System.out.println(koren.data);
    if (koren.data > cvor.data)
        printRootToCvor(koren.levo, cvor);
    else
        printRootToCvor(koren.desno, cvor);
}
```

Stampa putanju u AVL stablu od cvora do cvora - iterativno

```
public static void stampajPutanju(Cvor a, Cvor b){
    if (a == null || b == null)
      return;
    while (a.podatak != b.podatak){
      System.out.println(a.podatak);
      if (a.podatak > b.podatak)
        a = a.levo;
      else
        a = a.desno;
    }
    System.out.println(b.podatak);
}
```

Stampa putanju u AVL stablu od cvora do cvora - rekurzivno

```
public static void stampajPutanjuR(Cvor a, Cvor b) {
    if (a==null || b==null)
      return;
    System.out.println(a.podatak);
    if (a.podatak > b.podatak)
      stampajPutanjuR(a.levo, b);
    if (a.podatak < b.podatak)
      stampajPutanjuR(a.desno, b);
}
```