

```

/**
 * Čvor binarnog stabla.
 * Svaki čvor binarnog stabla u sebi čuva podatak (kod ovog čvora to je ceo broj)
 * i pokazivače na levo i desno dete.
 *
 * @author Kostandin Dimitrijević, Strukture podataka i algoritmi, FON, 2010 */
public class CvorStabla {
    // podatak koji se čuva u čvoru.
    // Ako čvor treba da čuva neki drugi tip elementa, ovde to treba promeniti
    int podatak;

    // "pokazivač" na levi čvor (levo dete - koren levog podstabla)
    CvorStabla levo;
    // "pokazivač" na desni čvor (desno dete - koren desnog podstabla)
    CvorStabla desno;

    /**
     * Konstruktor. Prihvata i postavlja sve elemente čvora.
     * @param p Podatak koji se čuva u čvoru.
     * @param l Pokazivač na levo dete
     * @param d Pokazivač na desno dete
     */
    public CvorStabla(int p, CvorStabla l, CvorStabla d){
        podatak = p;
        levo = l;
        desno = d;
    }

    /**
     * Konstruktor. Inicijalizuje samo podatak. Ovaj konstruktor se poziva
     * kada se kreira novi list u stablu.
     * @param p Podatak koji se čuva u čvoru.
     */
    public CvorStabla(int p){
        // poziva prethodni konstruktor i prosleđuje mu odgovarajuće parameter
        this(p, null, null);
    }
}

```

```

package NelinearneStrukture;

```

```

/**
 * Klasa koja implementira ponašanje binarnog stabla.
 * Ovo je dinamička razgranata struktura kod koje svaki element ima
 * najviše jednog prethodnika i najviše dva sledbenika. Elementi mogu da se
 * dodaju bilo gde u strukturu i bilo koji element može da se izbaci.
 * Redosled ubacivanja i izbacivanja nije eksplicitno definisan.
 *
 * @author Kostandin Dimitrijević, Strukture podataka i algoritmi, FON, 2010 */
public class BinarnoStablo {
    // Pokazivač na koreni (prvi) čvor u stablu.
    CvorStabla koren;

    public BinarnoStablo(){
        koren = null;
    }

    /**
     * Provera da li je stablo prazno. Stablo je prazno kada ne postoji
     * pokazivač na koren.
     * @return true ako je stablo prazno. u suprotnom false
     */
    public boolean praznoStablo(){
        return koren == null;
    }
}

```

```

}

/**
 * Primer implementacije prefiksnog prolaza kroz stablo.
 * U ovom primeru se vrši ispisivanje elemenata u prefiksnom redosledu.
 * Stablo se obilazi u redosledu koren - levo - desno.
 * Levo i desno podstablo se obilaze na isti način.
 * @param cvor Pokazivač na koren stabla koje se obilazi.
 */
public void prefiksniProlaz(CvorStabla cvor){
    if (cvor == null) return;

    // ispiši koren (trenutni element)
    System.out.println(cvor.podatak);
    // obiđi prefiksno levo podstablo
    prefiksniProlaz(cvor.levo);
    // obiđi prefiksno desno podstablo
    prefiksniProlaz(cvor.desno);
}

/**
 * Primer implementacije infiksnog prolaza kroz stablo.
 * U ovom primeru se vrši ispisivanje elemenata u infiksnom redosledu.
 * Stablo se obilazi u redosledu levo - koren - desno.
 * Levo i desno podstablo se obilaze na isti način.
 * @param cvor Pokazivač na koren stabla koje se obilazi.
 */
public void infiksniProlaz(CvorStabla cvor){
    if (cvor == null) return;

    // obiđi infiksno levo podstablo
    infiksniProlaz(cvor.levo);
    // ispiši koren (trenutni element)
    System.out.println(cvor.podatak);
    // obiđi infiksno desno podstablo
    infiksniProlaz(cvor.desno);
}

/**
 * Primer implementacije postfiksnog prolaza kroz stablo.
 * U ovom primeru se vrši ispisivanje elemenata u postfiksnom redosledu.
 * Stablo se obilazi u redosledu levo - desno - koren.
 * Levo i desno podstablo se obilaze na isti način.
 * @param cvor Pokazivač na koren stabla koje se obilazi.
 */
public void postfiksniprolaz(CvorStabla cvor){
    if (cvor == null) return;

    // obiđi postfiksno levo podstablo
    postfiksniprolaz(cvor.levo);
    // obiđi postfiksno desno podstablo
    postfiksniprolaz(cvor.desno);
    // ispiši koren (trenutni element)
    System.out.println(cvor.podatak);
}

/**
 * Metoda za prebrojavanje čvorova u stablu.
 * Prolazi kroz celo stablo na prefiksni način i pri obilasku svakog čvora
 * dodaje 1 na ukupan broj čvorova.
 * @param cvor Pokazivač na koren stabla za koje tražimo broj elemenata
 * @return Broj čvorova u stablu.
 */

```

```

    public int brojElemenata(CvorStabla cvor){
        if (cvor == null) return 0;
        // broj čvorova u stablu je jednak
        // koren (1) + broj elemenata u levom podstablu + broj elemenata u desnom
        podstablu
        // prvi nacin
        // int br = 1; // računaj trenutni čvor
        // dodaj broj elemenata u levom podstablu
        // br += brojElemenata(cvor.levo);
        // dodaj broj elemenata u desnom podstablu
        // br += brojElemenata(cvor.desno);
        // return br;

        // drugi nacin
        return 1 + brojElemenata(cvor.levo) + brojElemenata(cvor.desno);
    }

    /**
     * Metoda za izračunavanje zbira elemenata čvorova stabla.
     * Prolazi kroz celo stablo na prefiksni način i pri obilasku svakog čvora
     * dodaje njegovu vrednost na ukupan zbir.
     * @param cvor Pokazivač na koren stabla za koje tražimo zbir elemenata
     * @return Zbir elemenata čvorova u stablu.
     */
    public int zbirElemenata(CvorStabla cvor){
        if (cvor == null) return 0;
        // zbir elemenata čvorova u stablu je jednak
        // vrednost u korenu + zbir elemenata u levom podstablu + zbir elemenata u
        desnom podstablu
        return cvor.podatak + zbirElemenata(cvor.levo) + zbirElemenata(cvor.desno);
    }

    /**
     * Metoda za pronalazak maksimalne vrednosti u stablu.
     * Rekurzivno pronalazi maksimalni element u levom podstablu i
     * maksimalni element u desnom podstablu.
     * Nakon toga upoređuje te dve vrednosti sa vrednošću u trenutnom čvoru
     * i vraća najveću vrednost od te tri vrednosti.
     * @param cvor Pokazivač na koren stabla za koje tražimo maksimalni element
     * @return Vrednost maksimalnog elementa u stablu.
     */
    public int maxElement(CvorStabla cvor){
        if (cvor == null) return Integer.MIN_VALUE;

        // prvi nacin
        int max = cvor.podatak;
        // nađi maksimalnu vrednost u levom podstablu
        int maxl = maxElement(cvor.levo);
        // nađi maksimalnu vrednost u desnom podstablu
        int maxd = maxElement(cvor.desno);

        // upoređivanje vrednosti. max će uvek uzimati veću vrednost
        if (max < maxl)
            max = maxl;

        if (max < maxd)
            max = maxd;

        return max;

        // drugi nacin
        //return Math.max(Math.max(cvor.podatak, maxElement(cvor.levo)),
        maxElement(cvor.desno));
    }
}

```

```

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

    // prvi nacin
    int min = cvor.podatak;
    int minl = minElement(cvor.levo);
    int mind = minElement(cvor.desno);

    if (min > minl)
        min = minl;

    if (min > mind)
        min = mind;

    return min;
}

int visina(CvorStabla cvor){
    if (cvor == null || (cvor.levo == null && cvor.desno == null))
        return 0;

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

CvorStabla pronadji(CvorStabla tekuci, int podatak){
    if (tekuci == null || tekuci.podatak == podatak)
        return tekuci;

    CvorStabla l = pronadji(tekuci.levo, podatak);
    if (l != null)
        return l;

    // prvi nacin
    CvorStabla d = pronadji(tekuci.desno, podatak);
    if (d != null)
        return d;

    return null;

    // drugi nacin
    //return pronadji(tekuci.desno, podatak);
}

boolean daLiJeBst(CvorStabla cvor){
    if (cvor == null)
        return true;

    if (cvor.podatak > maxElement(cvor.levo) &&
        cvor.podatak < minElement(cvor.desno)) {

        return daLiJeBst(cvor.levo) && daLiJeBst(cvor.desno);
    }
    return false;
}

```

```

boolean daLiJeAvl(CvorStabla cvor){
    if (cvor == null)
        return true;

    if (daLiJeBst(cvor)){
        int rv = visina(cvor.levo) - visina(cvor.desno);

//        if (Math.abs(rv) < 2)
        if (rv > -2 && rv < 2)
            return daLiJeAvl(cvor.levo) && daLiJeAvl(cvor.desno);
    }
    return false;
}

boolean daLiJeHeap(CvorStabla cvor){
    if (cvor == null)
        return true;

    if ((cvor.levo == null || cvor.podatak > cvor.levo.podatak) &&
        cvor.desno == null || cvor.podatak > cvor.desno.podatak)

//    if (cvor.podatak == maxElement(cvor))
        return daLiJeHeap(cvor.levo) && daLiJeHeap(cvor.desno);

    return false;
}
}

```

```
package NelinearneStrukture;
```

```
/**
 * Klasa koja implementira BST stablo.
 * BST stablo je binarno stablo optimizovano za pretragu
 * (Binarno Stablo Traženja ili Binary Search Tree). Kod ovog stabla su
 * elementi u njemu raspoređeni tako da za svaki vor važi da su vrednosti
 * u čvorovima njegovog levog podstabla manje od vrednosti u tom čvoru,
 * i vrednosti u čvorovima njegovog desnog podstabla veće od vrednosti u tom čvoru.
 * Pošto je ovo binarno stablo, ova klasa nasleđuje klasu BinarnoStablo.
 */
@author Kostandin Dimitrijević, Strukture podataka i algoritmi, FON, 2010
*/
public class BstStablo extends BinarnoStablo {

    /**
     * Glavna (javna) metoda za dodavanje novog elementa u stablo.
     * Poziva privatnu metodu sa odgovarajućim parametrima.
     * @param podatak Podatak koji se ubacuje u stablo.
     */
    public void ubaciUStablo(int podatak){
        ubaci(koren, podatak);
    }

    /**
     * Metoda koja implementira algoritam za ubacivanje novog elementa u BST stablo.
     * Nov element se uvek ubacuje kao novi list u stablu, a mesto mu se određuje
     * upoređivanjem vrednosti sa vrednošću u trenutnom čvoru.
     *
     * @param tekuci Pokazivač na tekući čvor u stablu. Novi podatak ubacujemo u
     * njegovo levo ili desno podstablo
     * @param podatak Element koji se ubacuje u stablo
     */
    private void ubaci(CvorStabla tekuci, int podatak){
        if (praznoStablo())
            koren = new CvorStabla(podatak);
        else {
            // // proveravamo da li je vrednost podatka koju ubacujemo
            // veća ili manja od vrednosti u tekućem čvoru
            if (podatak < tekuci.podatak){
                // ako je vrednost manja onda se novi element ubacuje u levo podstablo
                if (tekuci.levo != null)
                    // ubacivanje u levo podstablo
                    ubaci(tekuci.levo, podatak);
                else
                    // ako ne postoji levo dete, onda se novi čvor ubacuje levo od tekućeg
                    tekuci.levo = new CvorStabla(podatak);
            } else if (podatak > tekuci.podatak){
                // ako je vrednost veća onda se novi element ubacuje u desno podstablo
                if (tekuci.desno != null)
                    // ubacivanje u desno podstablo
                    ubaci(tekuci.desno, podatak);
                else
                    // ako ne postoji desno dete, onda se novi čvor ubacuje desno od tekućeg
                    tekuci.desno = new CvorStabla(podatak);
            }
            // ako je podatak jednak podatku u tekućem čvoru ne vrši se nikakva
            // jer stablo ne može da ima duple vrednosti
        }
    }
}
```

```

/**
 * Pretraživanje BST stabla. Koristi se organizacija stabla kako bi se
 * što brže došlo do traženog elementa
 * @param tekuci Pokazivač na tekući element u stablu
 * @param podatak Podatak koji se traži
 * @return Pokazivač na čvor koji ima traženu vrednost. Ako čvor ne postoji vraća
null.
 */
CvorStabla pronadji(CvorStabla tekuci, int podatak){
    // ako smo stigli do "kraja" stabla (prvi uslov) ili smo pronašli čvor
    if (tekuci == null || tekuci.podatak == podatak)
        return tekuci;

    if (podatak < tekuci.podatak)
        return pronadji(tekuci.levo, podatak);
    else
        return pronadji(tekuci.desno, podatak);
}

/**
 * Pronalazak roditeljskog čvora za trenutni čvor
 * @param tekuci Pokazivač na tekući čvor
 * @param podatak Podatak za koji se traži roditeljski čvor
 * @return Pokazivač na roditeljski čvor. Ako se podatak ne nalazi u stablu
 * ili ako se podatak nalazi u korenu, vraća null.
 */
public CvorStabla nadjiRoditelja(CvorStabla tekuci, int podatak){
    if (tekuci == null || tekuci.podatak == podatak)
        return null;

    if (podatak < tekuci.podatak){
        // ako se podatak nalazi u levom detetu, onda je tekući njegov roditelj
        if (tekuci.levo != null && tekuci.levo.podatak == podatak)
            return tekuci;
        // probaj da pronađeš element u levom podstablu
        return nadjiRoditelja(tekuci.levo, podatak);
    } else {
        // ako se podatak nalazi u desnom detetu, onda je tekući njegov roditelj
        if (tekuci.desno != null && tekuci.desno.podatak == podatak)
            return tekuci;
        // probaj da pronađeš element u desnom podstablu
        return nadjiRoditelja(tekuci.desno, podatak);
    }
}

/**
 * Ispisivanje putanje od korena do nekog čvora u stablu.
 * Koristi organizaciju BST stabla da bi se kretao pravom putanjom do čvora
 * @param tekuci Pokazivač na tekući čvor na putanji
 * @param kraj Pokazivač na krajnji čvor na putanji
 */
void ispisiPutanju(CvorStabla tekuci, CvorStabla kraj){
    System.out.println(tekuci.podatak);

    if (tekuci == kraj)
        return;
    // na osnovu vrednosti u čvorovima se određuje da li se dalje treba
    // kretati levo ili desno
    if (kraj.podatak < tekuci.podatak)
        ispisiPutanju(tekuci.levo, kraj);
    else
        ispisiPutanju(tekuci.desno, kraj);
}

```

```

/**
 * Izračunavanje broja čvorova na putanji od korena do nekog čvora u stablu.
 * Koristi organizaciju BST stabla da bi se kretao pravom putanjom do čvora.
 * Može se koristiti i za određivanje nivoa nekog čvora u stablu.
 * @param tekuci Pokazivač na tekući čvor na putanji
 * @param kraj Pokazivač na krajnji čvor na putanji
 * @return Broj elemenata na putanji od korena do datog čvora.
 */
int brElNaPutanji(CvorStabla tekuci, CvorStabla kraj){
    if (tekuci == kraj)
        return 0;
    if (kraj.podatak < tekuci.podatak)
        return 1 + brElNaPutanji(tekuci.levo, kraj);
    else
        return 1 + brElNaPutanji(tekuci.desno, kraj);
}

/**
 * Pronalazak čvora sa maksimalnom vrednošću u stablu.
 * Čvor sa najvećom vrednošću u BST stablu se uvek nalazi krajnje desno.
 * @param tekuci Pokazivač na tekući element stabla (koren).
 * @return Čvor sa najvećom vrednošću.
 */
CvorStabla maxCvor(CvorStabla tekuci){
    if (tekuci == null) return null;
    while (tekuci.desno != null)
        tekuci = tekuci.desno;
    return tekuci;
}

/**
 * Pronalazak čvora sa minimalnom vrednošću u stablu.
 * Čvor sa najmanjom vrednošću u BST stablu se uvek nalazi krajnje levo.
 * @param tekuci Pokazivač na tekući element stabla (koren).
 * @return Čvor sa najmanjom vrednošću.
 */
CvorStabla minCvor(CvorStabla tekuci){
    if (tekuci == null) return null;
    while (tekuci.levo != null)
        tekuci = tekuci.levo;
    return tekuci;
}

/**
 * Metoda za izbacivanje lista ili polulista iz BST stabla.
 * List se izbacuje tako što se pokazivač roditeljskog čvora ka listu razveže
 * i postavi na null.
 * Polulist se izbacuje tako što se pokazivač roditeljskog čvora ka polulistu
 * preveže na jedino dete polulista
 * @param cvor Pokazivač na čvor koji se izbacuje
 */
private void izbaciListPolulist(CvorStabla cvor){
    // r je roditeljski čvor. Ako se izbacuje koren r će biti null
    CvorStabla r = nadjiRoditelja(koren, cvor.podatak);
    // dete je pokazivač na jedino dete polulista ili null ako je čvor list
    CvorStabla dete = cvor.levo != null ? cvor.levo : cvor.desno;
    // drugi način:
    CvorStabla dete;
    if (cvor.levo != null)
        dete = cvor.levo;
    else
        dete = cvor.desno;
}

```



```

        if (r == null) // izbacujemo koren. Dete postaje novi koren
            koren = dete;
        else { // prevezivanje roditeljskog čvora na dete
            if (r.levo == cvor)
                r.levo = dete;
            else
                r.desno = dete;
        }
    }

/**
 * Izbacivanje elementa iz BST stabla.
 * Postoje 3 slučaja: izbacuje se list, polulist ili unutrašnji čvor.
 * Ako se izbacuje unutrašnji čvor, na njegovo mesto se postavlja
 * najveći levi podatak (ili najmanji desni), a čvor sa tim postavljenim podatkom
 * se izbacuje iz stabla, te se ovaj slučaj svodi na izbacivanje lista ili polulista.
 * Ako se izbacuje list ili polulist, poziva se metoda za izbacivanje lista ili
polulista.
 */
 * @param podatak Vrednost koja se izbacuje iz stabla
 */
public void izbaci(int podatak){
    // pronaći pokazivač na čvor koji ima vrednost koja se izbacuje
    CvorStabla cvor = pronadji(koren, podatak);
    // ako je prazno stablo ili ako vrednost ne postoji u stablu prekini algoritam
    if (praznoStablo() || cvor == null) return;

    // proveriti da li je čvor koji se izbacuje unutrašnji čvor
    if (cvor.levo != null && cvor.desno != null){
        // nađi najveći levi kako bi se on postavio na mesto čvora koji izbacujemo
        CvorStabla maxL = maxCvor(cvor.levo);
        // izbaci najveći levi
        izbaciListPolulist(maxL);
        // vrednost iz (izbačenog) najvećeg levog prebaci na mesto elementa koji
se izbacuje
        cvor.podatak = maxL.podatak;
    }
    else
        izbaciListPolulist(cvor);
}
}

```