SPA

- sređeni kodovi sa vežbi za II deo (by Stepke, školska 2011-12) -

II deo

Nelinearne strukture

1. Čvor stabla

Cvor binarnog stabla. Svaki cvor binarnog stabla u sebi cuva podatak (kod ovog cvora to je ceo broj) i pokazivace na levo i desno dete.

```
public class CvorStabla {
        // podatak koji se cuva u cvoru.
        // Ako cvor treba da cuva neki drugi tip elementa, ovde to treba promeniti
        int podatak;

        // "pokazivac" na levi cvor (levo dete - koren levog podstabla)
        CvorStabla levo;
        // "pokazivac" na desni cvor (desno dete - koren desnog podstabla)
        CvorStabla desno;
```

Konstruktor. Prihvata i postavlja sve elemente cvora.
@param p Podatak koji se cuva u cvoru.
@param l Pokazivac na levo dete
@param d Pokazivac 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 cuva u cvoru.

```
public CvorStabla(int p){
                // poziva prethodni konstruktor i prosledjuje mu odgovarajuce parametre
                this(p, null, null);
        }
}
```

2. Binarno stablo

Klasa koja implementira ponasanje binarnog stabla. Ovo je dinamicka razgranata struktura kod koje svaki element ima najvise jednog prethodnika i najvise dva sledbenika. Elementi mogu da se dodaju bilo gde u strukturu i bilo koji element moze da se izbaci. Redosled ubacivanja i izbacivanja nije eksplicitno definisan.

```
public class BinarnoStablo {
        // Pokazivac na koreni (prvi) cvor u stablu.
        public CvorStabla koren;

        public BinarnoStablo() {
                koren = null;
        }
```

Proverava da li je stablo prazno. Stablo je prazno kada ne postoji pokazivac 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 vrsi ispisivanje elemenata u prefiksnom redosledu. Stablo se obilazi u redosledu koren - levo - desno. Levo i desno podstablo se obilaze na isti nacin.
@param cvor Pokazivac na koren stabla koje se obilazi.

```
public void prefiksniProlaz(CvorStabla cvor) {
                if (cvor == null)
                        return;

                // ispisi koren (trenutni element)
                System.out.println(cvor.podatak);
                // obidji prefiksno levo podstablo
                prefiksniProlaz(cvor.levo);
                // obidji prefiksno desno podstablo
                prefiksniProlaz(cvor.desno);
        }
```

Primer implementacije infiksnog prolaza kroz stablo. U ovom primeru se vrsi ispisivanje elemenata u infiksnom redosledu. Stablo se obilazi u redosledu levo - koren - desno. Levo i desno podstablo se obilaze na isti nacin.
@param cvor Pokazivac na koren stabla koje se obilazi.

```
public void infiksniProlaz(CvorStabla cvor) {
                if (cvor == null)
                        return;

                // obidji infiksno levo podstablo
                infiksniProlaz(cvor.levo);
                // ispisi koren (trenutni element)
                System.out.println(cvor.podatak);
                // obidji infiksno desno podstablo
                infiksniProlaz(cvor.desno);
        }
```

Primer implementacije postfiksnog prolaza kroz stablo. U ovom primeru se vrsi ispisivanje elemenata u postfiksnom redosledu. Stablo se obilazi u redosledu levo - desno - koren. Levo i desno podstablo se obilaze na isti nacin.
@param cvor Pokazivac na koren stabla koje se obilazi.

```
public void postfiksniProlaz(CvorStabla cvor) {
        if (cvor == null)
            return;

        // obidji postfiksno levo podstablo
        postfiksniProlaz(cvor.levo);
        // obidji postfiksno desno podstablo
        postfiksniProlaz(cvor.desno);
        // ispisi koren (trenutni element)
        System.out.println(cvor.podatak);
    }
```

Metoda za prebrojavanje cvorova u stablu. Prolazi kroz celo stablo na prefiksni nacin i pri obilasku svakog cvora dodaje 1 na ukupan broj cvorova.
@param cvor Pokazivac na koren stabla za koje trazimo broj elemenata
@return Broj cvorova u stablu.

```
public int brojElemenata(CvorStabla cvor) {
        if (cvor == null)
            return 0;
        // broj cvorova u stablu je jednak
        // koren (1) + broj elemenata u levom podstablu + broj elemenata u
        // desnom podstablu
        // prvi nacin
        // int br = 1; // racunaj trenutni cvor
        // 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 izracunavanje zbira elemenata cvorova stabla. Ako je stablo prazno, baca Exception. Prolazi kroz celo stablo na prefiksni nacin i pri obilasku svakog cvora dodaje njegovu vrednost na ukupan zbir.
@param cvor Pokazivac na koren stabla za koje trazimo zbir elemenata
@return Zbir elemenata cvorova u stablu.
@throws Exception - ako je stablo prazno

```
public int zbirElemenata(CvorStabla cvor) throws Exception {
        if (cvor == null)
            throw new Exception("Stablo je prazno");
        int suma = cvor.podatak;
        // ako postoji levo, sabiramo levo podstablo
        if (cvor.levo != null)
            suma += zbirElemenata(cvor.levo);
        // ako postoji desno, sabiramo desno podstablo
```

```
        if (cvor.desno != null)
            suma += zbirElemenata(cvor.desno);
        // zbir elemenata cvorova u stablu je jednak
        // vrednost u korenu + zbir elemenata u levom podstablu + zbir elemenata
        // u desnom podstablu
        return suma;
    }
```

Metoda za pronalazak maksimalne vrednosti u stablu. Rekurzivno pronalazi maksimalni element u levom podstablu i maksimalni element u desnom podstablu. Nakon toga uporedjuje te dve vrednosti sa vrednoscu u trenutnom cvoru i vraca najvecu vrednost od te tri vrednosti.
@param cvor Pokazivac na koren stabla za koje trazimo 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;
        // nadji maksimalnu vrednost u levom podstablu
        int maxl = maxElement(cvor.levo);
        // nadji maksimalnu vrednost u desnom podstablu
        int maxd = maxElement(cvor.desno);

        // uporedjivanje vrednosti. max ce uvek uzimati vecu 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));
    }
```

Metoda za pronalazak maksimalne vrednosti u stablu.
@param cvor
@return minimalna vrednost u stablu

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

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

Metoda za pronalazak cvora koji sadrzi maksimalnu vrednost u stablu. Rekurzivno pronalazi maksimalni element u levom podstablu i maksimalni element u desnom podstablu. Nakon toga uporedjuje te dve vrednosti sa vrednoscu u trenutnom cvoru i vraca cvor sa najvecom vrednoscu od te tri vrednosti.
@param cvor - Pokazivac na koren stabla za koje trazimo maksimalni element
@return pokazivac na cvor koji sadrzi maksimalni element u stablu.

```
public CvorStabla maxCvor(CvorStabla cvor) {
        if (cvor == null)
            return null;

        CvorStabla max = cvor;
        // nadj maksimalnu vrednost u levom podstablu
        CvorStabla maxl = maxCvor(cvor.levo);
        // nadj maksimalnu vrednost u desnom podstablu
        CvorStabla maxd = maxCvor(cvor.desno);

        // uporedjivanje vrednosti. max ce uvek uzimati vecu vrednost
        if (maxl != null && max.podatak < maxl.podatak)
            max = maxl;

        if (maxd != null && max.podatak < maxd.podatak)
            max = maxd;

        return max;

    }

    int visina(CvorStabla cvor) {
        if (cvor == null)
            return 1;

        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);
    }
```

Pronalazak roditeljskog cvora za trenutni cvor
@param tekuci Pokazivac na tekuci cvor
@param podatak Podatak za koji se trazi roditeljski Ächvor
@return Pokazivac na roditeljski cvor. Ako se podatak ne nalazi u stablu ili ako se podatak nalazi u korenu, vraca null.

```
public CvorStabla nadjiRoditelja(CvorStabla tekuci, int podatak) {
		if (tekuci == null || tekuci.podatak == podatak)
			return null;

		// ako se podatak nalazi u levom ili desnom detetu, onda je tekuci
		// njegov roditelj
		if ((tekuci.levo != null && tekuci.levo.podatak == podatak)
				|| (tekuci.desno != null && tekuci.desno.podatak == podatak))
			return tekuci;

		// probaj da pronadjes element u levom podstablu
		CvorStabla s = pronadji(tekuci.levo, podatak);
		// ako element jeste u levom
		if (s != null)
			// idi u levo podstablo
			return nadjiRoditelja(tekuci.levo, podatak);
		// u suprotnom probaj da pronadjes element u levom podstablu
		return nadjiRoditelja(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;
	}
}
```

3. BST stablo

Klasa koja implementira BST stablo. BST stablo je binarno stablo optimizovano za pretragu (Binarno Stablo Trazenja ili Binary Search Tree). Kod ovog stabla su elementi u njemu rasporedjeni tako da za svaki cvor vazi da su vrednosti u njegovom levom podstablu manji od vrednosti u tom cvoru, a vrednosti u njegovom desnom podstablu vece od vrednosti u tom cvoru Posto je ovo binarno stablo, ova klasa nasledjuje klasu BinarnoStablo.

```
public class BstStablo extends BinarnoStablo {

    /**
     * Glavna (javna) metoda za dodavanje novog elementa u stablo. Poziva
     * privatnu metodu sa odgovarajucim 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 odredjuje uporedjivanjem vrednosti sa vrednoscu u trenutnom cvoru.
@param tekuci Pokazivac na tekuci cvor 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
            // veca ili manja od vrednosti u tekucem cvoru
            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 cvor ubacuje levo
                    // od tekuceg
                    tekuci.levo = new CvorStabla(podatak);
            } else if (podatak > tekuci.podatak) {
                // ako je vrednost veca 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 cvor ubacuje
                    // desno od tekuceg
                    tekuci.desno = new CvorStabla(podatak);
            }
            // ako je podatak jednak podatku u tekucem cvoru ne vrsi se
```

```
                // nikakva operacija
                // jer stablo ne moze da ima duple vrednosti
            }
        }
```

Pretrazivanje BST stabla. Koristi se organizacija stabla kako bi se sto brze doslo do trazenog elementa
@param tekuci Pokazivac na tekuci element u stablu
@param podatak Podatak koji se trazi
@return Pokazivac na cvor koji ima trazenu vrednost. Ako cvor ne postoji, metoda vraca null.

```
CvorStabla pronadjiR(CvorStabla tekuci, int podatak) {
            // ako smo stigli do "kraja" stabla (prvi uslov) ili smo pronasli cvor
            if (tekuci == null || tekuci.podatak == podatak)
                return tekuci;

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

Pretrazivanje BST stabla. Koristi se organizacija stabla kako bi se sto brze doslo do trazenog elementa
@param tekuci Pokazivac na tekuci element u stablu
@param podatak Podatak koji se trazi
@return Pokazivac na cvor koji ima trazenu vrednost. Ako cvor ne postoji, metoda vraca null.

```
CvorStabla pronadji (CvorStabla tekuci, int podatak) {
            while (tekuci != null) {
                if (tekuci.podatak == podatak)
                    return tekuci;
                if (tekuci.podatak < podatak) {
                    tekuci = tekuci.desno;
                } else {
                    tekuci = tekuci.levo;
                }
            }

            return null;
        }
```

Pronalazak roditeljskog cvora za trenutni cvor
@param koren Pokazivac na tekuci cvor
@param tekuci Podatak za koji se trazi roditeljski cvor
@return Pokazivac na roditeljski cvor. Ako se podatak ne nalazi u stablu ili ako se podatak nalazi u korenu, vraca null.

```
public CvorStabla nadjiRoditelja(CvorStabla koren, CvorStabla tekuci) {
            if (koren == null || koren == tekuci)
                return null;

            if (tekuci.podatak < koren.podatak) {
                // ako se podatak nalazi u levom detetu, onda je tekuci njegov
                // roditelj
                if (koren.levo != null && koren.levo == tekuci)
```

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

Ispisivanje putanje od korena do nekog cvora u stablu. Koristi organizaciju BST stabla da bi se kretao pravom putanjom do cvora
@param tekuci Pokazivac na tekuci cvor na putanji
@param kraj Pokazivac na krajnji cvor na putanji

```
void ispisiPutanju(CvorStabla tekuci, CvorStabla kraj) {
        System.out.println(tekuci.podatak);

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

Metoda za pronalazak cvora koji sadrzi maksimalnu vrednost u stablu.

```
public CvorStabla maxCvor(CvorStabla tekuci) {
        while (tekuci.desno != null)
            tekuci = tekuci.desno;
        return tekuci;
    }
```

Metoda izbacuje zadati element stabla koji je list ili polulist
@param cvor

```
private void izbaciListPolulist(CvorStabla cvor) {
        // nalazimo roditelja za zadati element
        CvorStabla r = nadjiRoditelja(koren, cvor);
        // nalazimo dete (ako postoji) zadatog elementa

        CvorStabla dete = null;
        // I nacin
        dete = cvor.levo != null ? cvor.levo : cvor.desno;

        // II nacin
        // if (cvor.levo != null) {
```

```
                // dete = cvor.levo;
                // } else {
                // dete = cvor.desno;
                // }

                // ako ne postoji roditelj, to znaci da je element koji se izbacuje
                // koren stabla
                if (r == null)
                        koren = dete;
                else {
                        if (r.levo == cvor)
                                r.levo = dete;
                        else
                                r.desno = dete;
                }
        }
```

Metoda izbacuje element iz BST stabla koji sadrži zadatu vrednost
@param podatak zadata vrednost za izbacivanje elementa

```
public void izbaci(int podatak) {
                // pronalazimo cvor koji sadrzi zadati podatak
                CvorStabla cvor = pronadji(koren, podatak);
                // ako je stablo prazno ili ne postoji cvor koji sadrzi zadatu vrednost
                if (cvor == null)
                        return;

                // ako cvor postoji, proveravamo da li ima oba deteta
                if (cvor.levo != null && cvor.desno != null) {
                        // ako ima oba deteta, trazimo najveci element u njegovom levom
                        // podstablu
                        CvorStabla maxL = maxCvor(cvor.levo);
                        // zamenjujemo vrednost
                        cvor.podatak = maxL.podatak;
                        // i izbacujemo najveci levi element
                        izbaciListPolulist(maxL);

                } else {
                        // ako element nema oba deteta, izbacujemo taj element
                        izbaciListPolulist(cvor);
                }
        }

}
```

Hashing

Hashing

```
public class Hashing {

	public int[] niz;
	public int brEl;

	public Hashing(int dimenzija) {
		niz = new int[dimenzija];
		brEl = 0;
	}
```

Metoda ubacuje podatak u hash niz
@param podatak

```
	public void ubaci(int podatak) {
		// ako je niz pun ili ako je niz.length == 0
		if (brEl == niz.length)
			return;

		int adresa = hashFunkcija(podatak);

		while (niz[adresa] > 0)
			adresa = (adresa + 1) % niz.length;

		niz[adresa] = podatak;
		brEl++;
	}
```

Metoda trazi zadati podatak u hash nizu
@param podatak
@return index elementa u nizu ako ga pronadje, a ako ga ne pronadje, vraca -1

```
	public int pronadji(int podatak) {
		if (niz.length == 0 || brEl == 0)
			return -1;

		int adresa = hashFunkcija(podatak);
		int pom = adresa;
		while ((niz[adresa] != podatak) && (niz[adresa] != 0)) {
			adresa = (adresa + 1) % niz.length;
			if (adresa == pom)
				return -1;
		}
```

```
                if (niz[adresa] == podatak)
                        return adresa;
                return -1;
        }
```

Metoda izbacuje prosledjeni podatak iz hash niza
@param podatak

```
public void izbaci(int podatak, int a, int b) {
                int adresa = pronadji(podatak);

                if (adresa == -1)
                        return;
                niz[adresa] = -1;
        }
```

Hash funcija koja racuna adresu u nizu
@param podatak
@return adresa u nizu

```
public int hashFunkcija(int podatak) {
                return podatak % niz.length;
        }
}
```

Hashing olančavanjem

```
public class HasningOlancavanjem {

	public CvorJSListe[] niz;

	public HasningOlancavanjem(int dimenzija) {
		niz = new CvorJSListe[dimenzija];
	}
```

Metoda ubacuje podatak u hash niz
@param podatak

```
public void ubaci(int podatak) {
		if (niz.length == 0)
			return;

		int adresa = hashFunkcija(podatak);
		CvorJSListe novi = new CvorJSListe(podatak, niz[adresa]);
		niz[adresa] = novi;
	}
```

Metoda trazi prosledjenu vrednost
@param podatak
@return vraca index elementa u nizu, a ako ne postoji, vraca -1

```
public int pronadji(int podatak) {
		if (niz.length == 0)
			return -1;

		int adresa = hashFunkcija(podatak);
		if (niz[adresa] == null)
			return -1;
		CvorJSListe pom = niz[adresa];
		while ((pom != null) && (pom.podatak != podatak))
			pom = pom.sledeci;

		if (pom == null)
			return -1;
		return adresa;
	}
```

Metoda trazi prosledjenu vrednost
@param podatak
@return vraca CvorJSLIste, ako ne postoji, vraca null

```
public CvorJSListe pronadjiCvor(int podatak) {
		if (niz.length == 0)
			return null;

		int adresa = hashFunkcija(podatak);
		if (niz[adresa] == null)
			return null;
		CvorJSListe pom = niz[adresa];
		while ((pom != null) && (pom.podatak != podatak))
			pom = pom.sledeci;
		return pom;
	}
```

Metoda izbacuje prosledjeni podatak
@param podatak

```
public void izbaci(int podatak) {
		int adresa = hashFunkcija(podatak);

		CvorJSListe pom = niz[adresa];
		if (pom.podatak == podatak)
			niz[adresa] = pom.sledeci;
		else {
			while (pom.sledeci != null && pom.sledeci.podatak != podatak)
				pom = pom.sledeci;
			if (pom.sledeci != null)
				pom.sledeci = pom.sledeci.sledeci;
		}
	}
```

Hash funcija koja racuna adresu u nizu
@param podatak
@return adresa u nizu

```
public int hashFunkcija(int podatak) {
		return podatak % niz.length;
	}
}
```