SPA

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

I deo

Linearne strukture - Statičke

1. Nizovi

public class Nizovi {

Data su dva niza sortirana u rastucem redosledu. Napisati algoritam koji vraca niz od elemenata prethodna dva niza sortiran u neopadajucem redosledu.
@param a prvi niz
@param b drugi niz
@return int[] novi niz
@throws Exception

```
public static int[] spojiDva(int[] a, int[] b) throws Exception {
		// ako je duzina oba niza jednaka nuli "bacamo" Exception
		if (a.length == 0 && b.length == 0)
			throw new Exception("Nisu prosledjeni nizovi");

		// kreiramo novi niz, a njegova duzija je jednaka zbiru duzina
		// prosledjenih nizova
		int duzinaNovog = a.length + b.length;
		int[] novi = new int[duzinaNovog];

		// dodajemo pomocne "pokazivace" za pracenje pozicije
		int indexPrvogNiza = 0;
		int indexDrugogNiza = 0;

		for (int i = 0; i < duzinaNovog; i++) {
			// ako je zavrsen prvi niz ili nije zavrsen drugi niz i element u
			// drugom nizu je manje od elementa u prvom nizu
			if (indexPrvogNiza >= a.length
					|| (indexDrugogNiza < b.length && a[indexPrvogNiza] > b[indexDrugogNiza]))
{
				// ubacujemo element iz drugog niza
				novi[i] = b[indexDrugogNiza];
				// povecavamo pokazivac pozicije drugog niza za 1
				indexDrugogNiza++;

				// ako je zavrsen drugi niz ili nije zavrsen prvi niz i element
				// u
				// prvom nizu je manje od elementa u drugom nizu
```

```
                } else if (indexDrugogNiza >= b.length
                                || (indexPrvogNiza < a.length && a[indexPrvogNiza] <= b[indexDrugogNiza])) {
                        // ubacujemo element iz drugog niza
                        novi[i] = a[indexPrvogNiza];
                        // povecavamo pokazivac pozicije prvog niza za 1
                        indexPrvogNiza++;
                }
        }
        return novi;
    }
```

Izracunava zbir elemenata niza
@param niz niz ciji se zbir elemenata trazi
@return zbir elemenata datog niza

```
static int zbir(int[] niz) {
        int s = 0;
        for (int i = 0; i < niz.length; i++) {
                s += niz[i];
        }
        return s;
    }
```

Izracunava zbir elemenata niza rekurzivno
@param niz niz za koji racunamo zbir
@param index "pokazivac" na tekuci element
@return zbir elemenata datog niza
@throws Exception

```
public static int saberiRekurzivno(int[] niz, int index) throws Exception {
        if (niz == null || niz.length == 0)
                throw new Exception("Niz je prazan");
        if (index >= niz.length)
                return 0;
        return niz[index] + saberiRekurzivno(niz, index + 1);
    }
```

Ispisuje sve elemente niza
@param niz niz ciji se elementi ispisuju

```
static void ispisi(int[] niz) {
        for (int i = 0; i < niz.length; i++) {
                System.out.println("niz[" + i + "]=" + niz[i]);
        }
    }
}
```

2. Statički stek

Implementacija Steka na staticki nacin koriscenjem niza. Stek je linearna struktura koja funkcionise po LIFO (Last In First Out) principu, odnosno poslednji element koji je ubacen na stek se prvi izbacuje. Elementi se ubacuju i izbacuju sa istog kraja strukture.

```
public class StatickiStek {
        // niz u kome se cuvaju elementi steka
        private int[] niz;

        // indeks poslednjeg ubacenog elementa
        private int vrh;
```

Konstruktor. Inicijalizuje sve promenljive koje se koriste u implementaciji
@param kapacitet : maksimalni broj elemenata u steku

```
public StatickiStek(int kapacitet) {
                niz = new int[kapacitet];

                // posto prilikom inicijalizacije niza nemamo unetih elemenata,
                vrh = -1;
        }
```

Vraca broj elemenata na steku
@return broj elemenata

```
protected int brojElemenata() {
                return vrh + 1;
        }
```

Proverava da li je stek pun
@return true ako je stek pun. U suprotnom false

```
public boolean punStek() {
                // prvi nacin
                // if(vrh == dimenzija - 1)
                // return true;
                // else
                // return false;

                // drugi nacin
                return (vrh == niz.length - 1);
        }
```

Proverava da li je stek prazan.
@return true ako je stek prazan. U suprotnom false.

```
public boolean prazanStek() {
		return vrh == -1;
	}
```

Stavlja novi element na stek (ubacuje novi element)
@param element vrednost koja se ubacuje na stek
@return true ako je element ubacen, false ako nema mesta (stek je pun)

```
public boolean push(int element) {
		if (punStek())
			return false;

		// prvi nacin
		vrh++; // ili vrh = vrh + 1; // ili vrh += 1; // ili ++vrh;
		niz[vrh] = element;

		// drugi nacin
		// niz[++vrh] = element;
		return true;
	}
```

Izbacuje poslednje ubaceni element (element na vrhu steka)
@return izbaceni element
@throws EmptyStackException Ako je stek prazan, baca se izuzetak.

```
public int pop() throws EmptyStackException {
		if (prazanStek())
			throw new EmptyStackException();

		// prvi nacin
		int element = niz[vrh];
		vrh--;
		return element;
		// drugi nacin
		// return niz[vrh--];
	}
```

Uzima element sa vrha steka bez izbacivanja
@return element sa vrha steka
@throws EmptyStackException Ako je stek prazan, baca se izuzetak.

```
public int peek() throws EmptyStackException {
		if (prazanStek())
			throw new EmptyStackException();

		return niz[vrh];
	}

	public static void main(String[] args) {
		StatickiStek ss = new StatickiStek(4);
		ss.push(234);
		ss.pop();
	}

}
```

3. Statički red

Implementacija Reda na staticki nacin koriscenjem niza. Red je linearna struktura koja funkcionise po FIFO (First In First Out) principu, odnosno prvi element koji je ubacen u red se prvi izbacuje. Elementi se ubacuju na jednom, a izbacuju sa drugog kraja strukture.

```
public class StatickiRed {
        // niz u kome se cuvaju elementi reda
        private int[] niz;
        // indeks prvog elementa u redu (pocetak reda)
        private int p;
        // indeks poslednjeg elementa u redu (kraj reda)
        private int k;
        // broj elemenata u redu
        private int brElemenata;
```

Konstruktor. Inicijalizuje sve promenljive koje se koriste u implementaciji
@param kapacitet Maksimalni broj elemenata u redu

```
public StatickiRed(int kapacitet) {
                niz = new int[kapacitet];
                brElemenata = 0;
                p = 0;
                k = -1;
        }
```

Proverava da li je red pun
@return true ako je red pun. U suprotnom false.

```
public boolean punRed() {
                return brElemenata == niz.length;
        }
```

Proverava da li je red prazan.
@return true ako je red prazan. U suprotnom false.

```
public boolean prazanRed() {
                return brElemenata == 0;
        }
```

Ubacuje novi element u red
@param element vrednost koja se ubacuje u red
@return true ako je element ubacen, false ako nema mesta (red je pun)

```
public boolean enqueue(int element) {
		if (punRed())
			return false;

		// prvi nacin
		k++;
		if (k == niz.length)
			k = 0;
		niz[k] = element;
		// drugi nacin
		// niz[k = ++k % dimenzija] = element;
		brElemenata++;
		return true;
	}
```

Izbacuje prvi element iz reda (element na pocetku)
@return izbaceni element sa pocetka reda.
@throws Exception
Ako je red prazan, baca se izuzetak.

```
public int dequeue() throws Exception {
		if (prazanRed())
			throw new Exception("Red je prazan");

		int podatak = niz[p];
		p++;
		if (p == niz.length)
			p = 0;
		brElemenata--;
		return podatak;
	}
```

Vraca element koji se nalazi na pocetku reda
@return prvi element u redu.
@throws Exception Ako je red prazan, baca se izuzetak.

```
public int pocetak() throws Exception {
		if (prazanRed())
			throw new Exception("Red je prazan");

		return niz[p];
	}
```

Vraca element koji se nalazi na kraju reda
@return poslednji element u redu.
@throws Exception Ako je red prazan, baca se izuzetak.

```
public int kraj() throws Exception {
		if (prazanRed())
			throw new Exception("Red je prazan");

		return niz[k];
	}
```

Ispisuje sve elemente reda od pocetka do kraja (for petlja)

```
public void ispisi() {
		if (prazanRed())
			return;
		for (int i = p; i != k;) {
			System.out.println(niz[i]);
			// prvi nacin
			// i = ++i % dimenzija;

			// drugi nacin
			i++;
			if (i == niz.length)
				i = 0;
		}
		System.out.println(niz[k]);
	}
```

Ispisuje sve elemente reda od pocetka do kraja (while petlja)

```
public void ispisi2() {
		int i = p;
		int br = 0;
		while (br < brElemenata) {
			System.out.println(niz[i]);
			br++;
			// prvi nacin
			// i = ++i % dimenzija;
			// drugi nacin
			i++;
			if (i == niz.length)
				i = 0;
		}
	}
}
```

Linearne strukture - Dinamičke

1. Čvor jednostruko spregnute liste

Cvor jednostruko spregnute liste. Svaki cvor liste u sebi cuva podatak (kod ovog cvora to je ceo broj) i pokazivac na sledeci cvor.

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

        // "pokazivac" na sledeci cvor u listi
        public CvorJSListe sledeci;
```

Konstruktor. Prihvata i postavlja sve elemente cvora.
@param p Podatak koji se cuva u cvoru.
@param sled Pokazivac na sledeci element za cvor koji se kreira

```
public CvorJSListe(int p, CvorJSListe sled) {
                podatak = p;
                sledeci = sled;
        }
}
```

2. Jednostruko spregnuta lista

Jednostruko spregnuta lista. Dinamicka opsta lista - elementi mogu da se dodaju bilo gde u strukturu i bilo koji element moze da se izbaci. Redosled ubacivanja i izbacivanja nije definisan. Elementi su uvezani preko jedne veze "unapred", tako da je moguce direktno kretanje kroz strukturu samo u jednom smeru od prvog do poslednjeg. Na svaki cvor pokazuje tacno jedan pokazivac, te je pri ubacivanju novog cvora potrebno povezati jedan pokazivac na njega, dok je pri izbacivanju potrebno razvezati jedan pokazivac.

```
public class JSLista {
```

Pokazivav na prvi vvor u listi. Posto su cvorovi medjusobno uvezani, dovoljno je pamtiti gde se nalazi prvi cvor, pa preko njega pristupiti ostalim cvorovima.

```
    CvorJSListe prvi;

    public JSLista() {
        prvi = null;
    }
```

Proverava da li je lista prazna. Lista je prazna kada ne postoji pokazivac na prvi cvor.
@return true ako je lista prazna. u suprotnom false

```
public boolean praznaLista() {
        return prvi == null;
    }
```

Ubacivanje novog elementa na pocetak liste.
@param podatak Podatak koji ce se cuvati u novom cvoru

```
public void ubaciNaPocetak(int podatak) {

        // I nacin

        // Kreiranje novog cvora. Posto se novi cvor kreira na pocetku
        // strukture, njegov sledeci element ce biti trenutno prvi element.
        CvorJSListe novi = new CvorJSListe(podatak, prvi);
        // nakon kreiranja cvora postavlja se da je novokreirani cvor prvi cvor.
        prvi = novi;

        // II nacin

        // prvi = new CvorJSListe(podatak, prvi);

    }
```

Ubacivanje novog elementa na kraj liste. Potrebno je naci poslednji cvor i ubaciti novi nakon njega.
@param podatak Podatak koji ce se cuvati u novom cvoru

```
public void ubaciNaKraj(int podatak) {
        if (praznaLista()) {
                // ako je lista prazna, algoritam za ubacivanje novog cvora na kraj
                // liste je isti kao i za ubacivanje na pocetak, jer se ubacuje
                // prvi koji je istovremeno i poslednji cvor
                ubaciNaPocetak(podatak);
        } else {
                // pomocni pokazivac preko koga se krece kroz listu
                CvorJSListe pom = prvi;
                // pronalazak poslednjeg cvora u listi (onaj cvor koji nema
                // sledbenika)
                while (pom.sledeci != null) {
                        pom = pom.sledeci;
                }
                // kreiranje novog cvora. Posto se ubacuje na kraj liste,
                // onda on nema sledbenika.
                CvorJSListe novi = new CvorJSListe(podatak, null);
                // povezivanje da je novokreirani cvor sledbenik do tada poslednjem
                // cvoru
                pom.sledeci = novi;
        }
    }
```

Izbacivanje elementa sa pocetka liste. Element se izbacuje tako sto se pokazivac na prvi cvor preveze da pokazuje na drugi cvor. Time niko vise ne pokazuje na taj cvor cime se on brise iz memorije.
@return Vrednost iz cvora koji je izbacen.
@throws Exception Ako je lista prazna baca se izuzetak.

```
public int izbaciSaPocetka() throws Exception {
        // Ako je lista prazna baca se izuzetak
        if (praznaLista())
                throw new Exception("lista je prazna");

        // sacuvamo podatak iz prvog cvora u pomocnu promenljivu
        int podatak = prvi.podatak;
        // prebacujemo prvi na drugi cvor
        prvi = prvi.sledeci;
        // vracamo sacuvani podatak
        return podatak;
    }
```

Izbacivanje elementa sa kraja liste. Da bismo izbacili poslednji cvor potrebno je da nadjemo pretposlednji cvor i da razvezemo vezu izmedju pretposlednjeg i poslednjeg cvora.
@return Vrednost iz cvora koji je izbacen.
@throws Exception Ako je lista prazna baca se izuzetak

```
public int izbaciSaKraja() throws Exception {
		if (praznaLista())
			throw new Exception("lista je prazna");

		// ako lista ima samo jedan element, onda se izbacuje prvi
		if (prvi.sledeci == null)
			return izbaciSaPocetka();
		// pronalazimo pretposlednji cvor
		CvorJSListe pom = prvi;
		// pretposlednji cvor ima samo jedan element ispred sebe
		while (pom.sledeci.sledeci != null) {
			pom = pom.sledeci;
		}
		int podatak = pom.sledeci.podatak;
		// razvezujemo vezu izmedju pretposlednjeg i poslednjeg
		pom.sledeci = null;
		return podatak;
	}
```

Ubacivanje elementa u sortiranu listu tako da ona ostane sortirana u rastucem redosledu. Prolazi se kroz listu i trazi onaj cvor kod koga je podatak u njegovom sledecem veci od elementa koji se ubacuje. Nakon toga se novi cvor umetne izmedju trenutnog i njegovog sledeceg.
@param podatak Podatak koji ce se cuvati u novom cvoru

```
public void ubaciUSort(int podatak) {
		// ako je lista prazna ili ako je podatak u prvom cvoru nije manji od
		// podatka koji se ubacuje, onda se vrsi ubacivanje na pocetak liste
		if (praznaLista() || prvi.podatak >= podatak)
			prvi = new CvorJSListe(podatak, prvi);
		else {
			CvorJSListe pom = prvi;
			// trazimo element ciji sledeci cvor ima podatak koji je
			// veci od elementa koji ubacujemo ili dok ne dodjemo do kraja liste
			while (pom.sledeci != null && pom.sledeci.podatak < podatak) {
				pom = pom.sledeci;
			}
			// umetanje novog cvora
			CvorJSListe novi = new CvorJSListe(podatak, pom.sledeci);
			pom.sledeci = novi;
		}
}
```

Prolazak kroz listu i ispisivanje svih elemenata u cvorovima od prvog do poslednjeg.

```
public void ispisi() {
		CvorJSListe pom = prvi;
		while (pom != null) {
			System.out.println(pom.podatak);
			pom = pom.sledeci;
		}
	}
```

Prolazak kroz listu i ispisivanje svih elemenata u cvorovima od prvog do poslednjeg. Rekurzivna metoda. Ideja algoritma: prove je potrebno da se prvo ispise element na kome se nalazi pokazivac a nakon toga se poziva metoda rekurzivno poziva i prosledjuje se pokazivac na sledeci element u listi.

```
public static void ispisiRekurzivno(CvorJSListe pom) {
		if (pom == null)
			return;
		System.out.println(pom.podatak);
		ispisiRekurzivno(pom.sledeci);
	}
```

Ispisivanje svih elemenata u cvorovima u obrnutom redosledu - od poslednjeg do prvog. Rekurzivna metoda. Ideja algoritma: prvo je potrebno ispisati sve elemente nakon trenutnog u obrnutom redosledu, a na kraju ispisati trenutni element. Ovo se rekurzivno ponavlja za svaki cvor.

```
public void ispisiObrnuto(CvorJSListe pom) {
		if (pom == null)
			return;
		if (pom.sledeci != null)
			ispisiObrnuto(pom.sledeci);
		System.out.println(pom.podatak);
	}
```

Izbacivanje elementa na koji pokazuje t. Potrebno je da se nadje element ciji je sledeci t. Ako t pokazuje na prvi, element se izbacuje tako sto se pokazivac na prvi cvor preveze da pokazuje na drugi cvor. U suprotnom, potrebno je naci element ciji je sledeci t. Element se izbacuje tako sto se njegov pokazivac preveze da ne pokazuje na t nego element posle t.
@param t "pokazivac" na neki element u listi
@return vrednost cvora koji je izbacen
@throws Exception

```
public int izbaciTrenutni(CvorJSListe t) throws Exception {
		// provera da li je lista prazna
		if (prvi == null)
			// ako je prazna, bacamo exception
			throw new Exception("Lista je prazna!");
```

```
		// provera da li t pokazuje na prvi
		if (t == prvi)
			// ako pokazuje, pomeramo prvi na sledeci element
			prvi = prvi.sledeci;
		else {
			// ako t ne pokazuje na prvi, uvodimo pomocni pokazivac p
			CvorJSListe p = prvi;
			// sve dok p.sledeci ne pokazuje na t
			while (p.sledeci != t) {
				// pomeramo p za po jedno mesto
				p = p.sledeci;
			}
			// kada pronadjemo element ciji je sledeci t
			// prevezemo tako da njegov sledeci ne pokazuje vise na t, nego na
			// element posle t
			p.sledeci = t.sledeci; // p.sledeci = p.sledeci.sledeci;
		}
		// vracamo vrednost elementa koji izbacujemo
		return t.podatak;

		// NAPOMENA: OVO JE SPECIJALAN SLUCAJ DA NE MORAMO UZETI VREDNOST IZ
		// ELEMENTA PRE NEGO STO GA IZBACIMO!!!!!
	}

Metoda racuna zbir elemenata u ciklicnoj listi
@return zbir elemenata
@throws Exception

public int zibrElemenataUCiklicnoj() throws Exception {
		if (prvi == null)
			throw new Exception("Lista je prazna!");
		int s = prvi.podatak;
		CvorJSListe pom = prvi.sledeci;
		while (pom != prvi) {
			s += pom.podatak;
		}
		return s;
	}
}
```

3. Čvor dvostruko spregnute liste

Cvor dvostruko spregnute liste. Svaki cvor liste u sebi cuva podatak (kod ovog cvora to je ceo broj) i pokazivace na prethodni i sledeci cvor.

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

        // "pokazivac" na prethodni cvor u listi
        CvorDSListe prethodni;

        // "pokazivac" na sledeci cvor u listi
        CvorDSListe sledeci;
```

Konstruktor. Prihvata i postavlja sve elemente cvora.
@param p Podatak koji se cuva u cvoru.
@param pret Pokazivac na prethodni element liste
@param sled Pokazivac na sledeći element za cvor koji se kreira

```
public CvorDSListe(int p, CvorDSListe pret, CvorDSListe sled) {
                podatak = p;
                prethodni = pret;
                sledeci = sled;
        }
}
```

4. Dvostruko spregnuta lista

Dvostruko spregnuta lista. Dinamicka opsta lista - elementi mogu da se dodaju bilo gde u strukturu i bilo koji element moze da se izbaci. Redosled ubacivanja i izbacivanja nije definisan. Elementi su uvezani preko dve veze "unapred" i "unazad", tako da je moguce direktno kretanje kroz strukturu u smeru od prvog do poslednjeg i od poslednjeg do prvog. Na svaki cvor pokazuje tacno dva pokazivaca, te je pri ubacivanju novog cvora potrebno povezati dva pokazivaca na njega, dok je pri izbacivanju potrebno razvezati dva pokazivaca. Gledano sa oba kraja, lista je strukturno potpuno simetricna.

```
public class DSLista {
	// Pokazivac na prvi cvor u listi.
	private CvorDSListe prvi;

	// Pokazivac na poslednji cvor u listi.
	private CvorDSListe poslednji;

	private DSLista() {
		prvi = poslednji = null;
	}
```

Proverava da li je lista prazna. Lista je prazna kada ne postoji pokazivac na prvi cvor.
@return true ako je lista prazna. u suprotnom false

```
public boolean praznaLista() {
		return prvi == null;
	}
```

Ubacivanje novog elementa na pocetak liste.
@param podatak Podatak koji ce se cuvati u novom cvoru

```
public void ubaciNaPocetak(int podatak) {
		// Kreiranje novog cvora. Posto se novi cvor kreira na pocetku
		// strukture, on nema prethodnika, a njegov sledeci element ce biti
		// trenutno prvi element.
		CvorDSListe novi = new CvorDSListe(podatak, null, prvi);

		// ako je lista prazna, onda je novokreirani element i poslednji element
		if (praznaLista())
			poslednji = novi;
		else
			// ako lista nije prazna onda je novokreirani element prethodnik
			// trenutno prvom elementu
			prvi.prethodni = novi;
		// nakon povezivanja treba postaviti da je novi element prvi
		prvi = novi;
	}
```

Ubacivanje novog elementa na kraj liste.
@param podatak Podatak koji ce se cuvati u novom cvoru

```
public void ubaciNaKraj(int podatak) {
		// Kreiranje novog cvora. Pocto se novi cvor kreira na kraju
		// strukture, on nema sledbenika, a njegov prethodnik ce biti
		// trenutno poslednji element.
		CvorDSListe novi = new CvorDSListe(podatak, poslednji, null);

		// ako je lista prazna, onda je novokreirani element i prvi element
		if (praznaLista())
			prvi = novi;
		else
			// ako lista nije prazna onda je novokreirani element sledbenik
			// trenutno poslednjem elementu
			poslednji.sledeci = novi;
		// nakon povezivanja treba postaviti da je novi element prvi
		poslednji = novi;
	}
```

Izbacivanje elementa sa pocetka liste. Element se izbacuje tako sto se pokazivac na prvi cvor prevece da pokazuje na drugi cvor i razveze veza "unazad" izmedju drugog i prvog cvora. Time niko vise ne pokazuje na taj cvor cime se on brise iz memorije.
@return Vrednost iz cvora koji je izbacen.
@throws Exception Ako je lista prazna baca se izuzetak.

```
public int izbaciSaPocetka() throws Exception {
		// Ako je lista prazna baca se izuzetak
		if (praznaLista())
			throw new Exception("lista je prazna");

		// sacuvamo podatak iz prvog cvora u pomocnu promenljivu
		int podatak = prvi.podatak;
		// prebacujemo prvi na drugi cvor
		prvi = prvi.sledeci;
		// ako je lista imala samo jedan element, onda je sada prvi == null,
		// pa treba postaviti i da je poslednji == null
		if (prvi == null)
			poslednji = null;
		else
			// razvezivanje veze unazad
			prvi.prethodni = null;
		return podatak;
	}
```

Izbacivanje elementa sa kraja liste. Element se izbacuje tako sto se pokazivac na poslednji cvor preveze da pokazuje na pretposlednji cvor i razvece veza "inapred" izmedju pretposlednjeg i poslednjeg cvora. Time niko vice ne pokazuje na taj cvor cime se on brise iz memorije.
@return Vrednost iz cvora koji je izbacen.
@throws Exception Ako je lista prazna baca se izuzetak.

```
public int izbaciSaKraja() throws Exception {
		// Ako je lista prazna baca se izuzetak
		if (praznaLista())
			throw new Exception("lista je prazna");

		// sacuvamo podatak iz poslednjeg cvora u pomocnu promenljivu
		int podatak = poslednji.podatak;
		// prebacujemo poslednji na pretposlednji cvor
		poslednji = poslednji.prethodni;
		// ako je lista imala samo jedan element, onda je sada poslednji ==
		// null,
		// pa treba postaviti i da je prvi == null
		if (poslednji == null)
			prvi = null;
		else
			// razvezivanje veze unapred
			poslednji.sledeci = null;
		return podatak;
	}
```

Ubacivanje elementa u sortiranu listu tako da ona ostane sortirana u rastucem redosledu. Prolazi se kroz listu i trazi onaj cvor ciji je podatak veci od elementa koji se ubacuje. Nakon toga se novi cvor ubacuje izmedju trenutnog i njegovog prethodnika.
@param podatak Podatak koji ce se cuvati u novom cvoru

```
public void ubaciUSort(int podatak) {
		// Ako je lista prazna ili ako je podatak u poslenjem cvoru nije manji
		// od
		// podatka koji se ubacuje, onda se vrsi ubacivanje na kraj liste.
		if (praznaLista() || poslednji.podatak <= podatak)
			ubaciNaKraj(podatak);
		else {
			CvorDSListe pom = prvi;
			// trazimo element ciji je podatak veci od elementa koji
			// ubacujemo
			while (pom.podatak < podatak) {
				pom = pom.sledeci;
			}
			// umetanje novog cvora
			CvorDSListe novi = new CvorDSListe(podatak, pom.prethodni, pom);
```

```
                pom.prethodni = novi;
                if (novi.prethodni == null)
                        prvi = novi;
                else
                        novi.prethodni.sledeci = novi;
        }
}
```

Dat je pokazivac na neki cvor dvostruko spregnute ciklicne liste. Napisati algoritam koji od tog cvora nalazi prvi element date liste koji ima vrednost manju od proseka liste i postavlja taj element da bude sledeci od elementa na koji pokazuje dati pokazivac.

```
public void premestiManjiOdProseka(CvorDSListe p) {
        // ako je lista prazna, metoda ne radi nista
        if (p == null) {
                return;
        }

        double zbir = p.podatak;
        int brojac = 1;
        CvorDSListe pom = p.sledeci;

        // racunamo zbir elemenata i prebrojavamo elemente u listi
        while (pom != p) {
                zbir += pom.podatak;
                brojac++;
                pom = pom.sledeci;
        }

        // racunamo prosek
        double prosek = zbir / brojac;

        // ako element na drugoj pozijici ima vrednost manju od proseka.
        if (p.sledeci.podatak < prosek)
                return;

        pom = p;
        while (pom.sledeci != p && pom.sledeci.podatak >= prosek) {
                pom = pom.sledeci;
        }

        // razvezujemo pokazivace
        // prvo razvezujemo pokazivace koji pokazuju na pom
        pom.prethodni.sledeci = pom.sledeci;
        pom.sledeci.prethodni = pom.prethodni;
```

```
                // razvezujemo pokazivace tako da pom pokazuje na svoj novi sledeci i
                // prethodni
                pom.sledeci = p.sledeci;
                pom.prethodni = p;

                // razvezujemo pokazivace tako da novi prethodi i novi sledeci pokazuju
                // na pom
                p.sledeci.prethodni = pom;
                p.sledeci = pom;

        }
}
```