Algoritmi i Strukture podataka

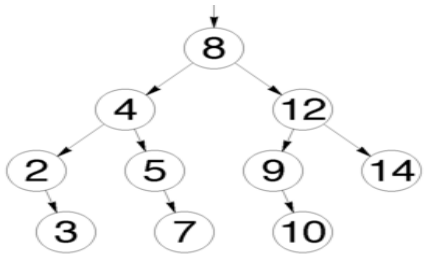
- ***Zadaci iz stabala-***

**** Svi zadaci su testirani****

```
** Klasa binarnog stabla
	package Branko-Stabla;
	public class BinarnoStablo {
		public CvorStabla koren;

		public BinarnoStablo() {
		koren = null;
	}
}
** Metoda koja broji elemente u stablu
	public int prebrojCvorove(CvorStabla cvor) {
		if (cvor == null) return 0;
		return 1+prebrojCvorove(cvor.levo) + prebrojCvorove(cvor.desno);
	}

** Metoda koja racuna zbir svih elemenata stabla
	public int zbirCvorova(CvorStabla cvor) {
		if (cvor == null)
			return 0;
		return  cvor.podatak+  zbirCvorova(cvor.levo)+zbirCvorova(cvor.desno);

	}
```

**Prefiksni prolaz kroz stablo se zasniva na prolazu kroz stablo tako da se prvo poseti koren, pa sva leva deca, pa sva desna deca

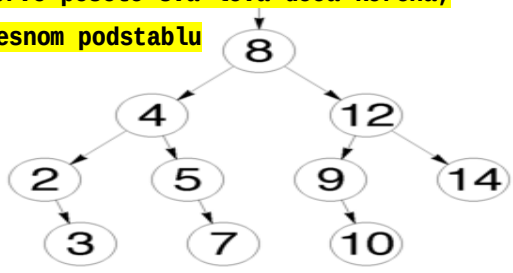
```
	public void prefiksniProlaz(CvorStabla cvor) {
		if (cvor == null)
			return;
		System.out.println(cvor.podatak);
		prefiksniProlaz(cvor.levo);
		prefiksniProlaz(cvor.desno);
	}
```



8, 4, 2, 3, 5, 7, 12, 9, 10, 14

** Infiksni prolaz kroz stablo je prolaz gde se prvo posete sva leva deca korena, pa sam koren, pa na kraju deca koja se nalaze u desnom podstablu

```
public void infiksniProlaz(CvorStabla cvor) {
		if (cvor == null)
			return;
		infiksniProlaz(cvor.levo);
		System.out.println(cvor.podatak);
		infiksniProlaz(cvor.desno);
```

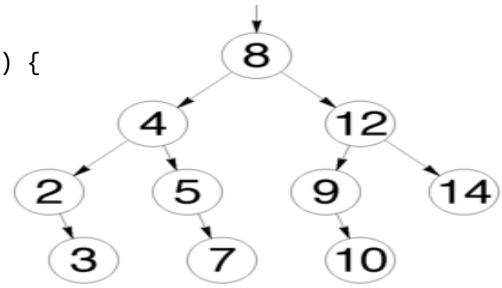


2, 3, 4, 5, 7, 8, 9, 10, 12, 14

```
	}
```

**** Postfiksni prolaz prvo ispisuje levu decu podstabla, potom desnu pa na kraju ispise koren stabla**

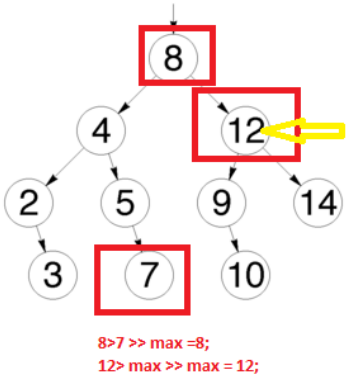
```
	public void postfiksniProlaz(CvorStabla cvor) {
		if (cvor == null)
			return;
		postfiksniProlaz(cvor.levo);
		postfiksniProlaz(cvor.desno);
		System.out.println(cvor.podatak);
	}
```



3, 2, 5, 7, 4, 10, 9, 14, 12, 8

Metoda koja nalazi i vraca vrednost maksimalnog elementa u stablu

```
	public int maxElement(CvorStabla cvor) {
		if (cvor == null)
			return Integer.MIN_VALUE;
		int max = cvor.podatak;
		int l = maxElement(cvor.levo);
		int d = maxElement(cvor.desno);

		if (max < l)
			max = l;
		if (max < d)
			max = d;
		return max;
	}
```



Metoda koja vraca najveci cvor u stablu

```
	public CvorStabla maxCvor(CvorStabla cvor) {
		if (cvor == null)
			return null;
		CvorStabla max = cvor;
		CvorStabla maxl = maxCvor(cvor.levo);
		CvorStabla maxd = maxCvor(cvor.desno);

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

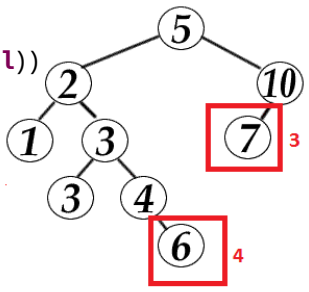
```
            return max;
      }
```

**Metoda koja pronalazi i vraca vrednost minimalnog elementa u stablu. Uslov za izlazak iz rekurzije je da ako je cvor stable null, da povratna vrednost bude Integer.*MAX_VALUE*. Pretpostavimo da je minimalni elemenat koren. Nakon toga nadjemo najmanji elemenat u levom podstabu, pa nakon toga u desnom podstablu, nakon cega sledi uporedjivanje korena na nadjenim minimumima. Vrednost najmanjeg od njih bice povratna vrednost metode.

```
      public int minElement(CvorStabla cvor) {
            if (cvor == null)
                  return Integer.MAX_VALUE;
            int min = cvor.podatak;
            int l = minElement(cvor.levo);
            int d = minElement(cvor.desno);

            if (min > l)
                  min = l;
            if (min > d)
                  min = d;
            return min;
      }
```

**Metoda koja racuna visinu stable. Visina je jednaka zbiru nivoa na kome se nalazi koren i dubini na kojoj se nalazi najdublji list

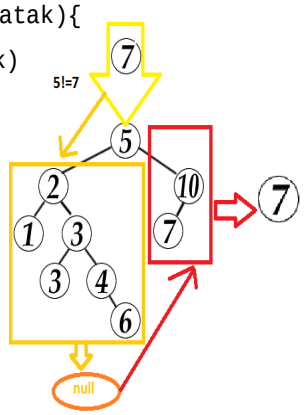
```
      public int visina(CvorStabla cvor) {
        if (cvor==null||(cvor.levo == null && cvor.desno == null))
            return 0;
      return 1 + max(visina(cvor.levo), visina(cvor.desno));
      }
      public int max(int a, int b) {
            if (a>b) {
                  return a;
            }
            return b; }
```



**Metoda koja pronalazi i vraca cvor koji sadrzi trazeni podatak. Najpre sledi provera da li je stablo prazno i da li je koren stable zapravo trazeni cvor. Pocetni uslov je zapravo i uslova za izlazak iz rekurzije. Nakon toga pokusamo pronaci elemenat u levom podstablu, i ako ga nadjemo vratimo tu vrednost. Ako ga

nismo pronasli u levom trazice se u desnom podstablu. Ako elemenat nit u nije pronadjen u skladu sa pocetnim uslovom metoda ce vratiti null.

```
public 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;
        return pronadji(tekuci.desno, podatak);
}
```



**** Metoda koja pronalazi i vraca trazeni cvor**

```
public CvorStabla pronadjiCvor(CvorStabla tekuci, CvorStabla P) {
        if (tekuci == null || tekuci==P)
                return tekuci;
        CvorStabla l = pronadjiCvor(tekuci.levo, P);
        if (l != null)
                return l;
        return pronadjiCvor(tekuci.desno, P);
}
```

**** Metoda koja vraca broj svih cvorova koji su veci od potomaka. Koren je veci od svojih potomaka ako je veci od levog i desnog deteta.**

```
public int prebroj (CvorStabla koren) {
if (koren == null || (koren.levo==null &&koren.desno==null))
                return 0;
if (koren.podatak>koren.levo.podatak && koren.podatak >koren.desno.podatak)
        return 1 + prebroj (koren.levo) + prebroj (koren.desno);
return prebroj(koren.levo)+prebroj(koren.desno);
}
```

**** Metoda koja vraca visinu na kome se nalazi trazeni cvor**

```
public int nivoDatogCvora (CvorStabla koren, CvorStabla p){
        if (koren==null) {
                return 0;
        }

        if(pronadjiCvor(koren.levo, p)!=null)
                return 1+nivoDatogCvora(koren.levo, p);

        if(pronadjiCvor(koren.desno, p)!=null)
                return 1+nivoDatogCvora(koren.desno, p);
```

```
		return 0;

	}
```

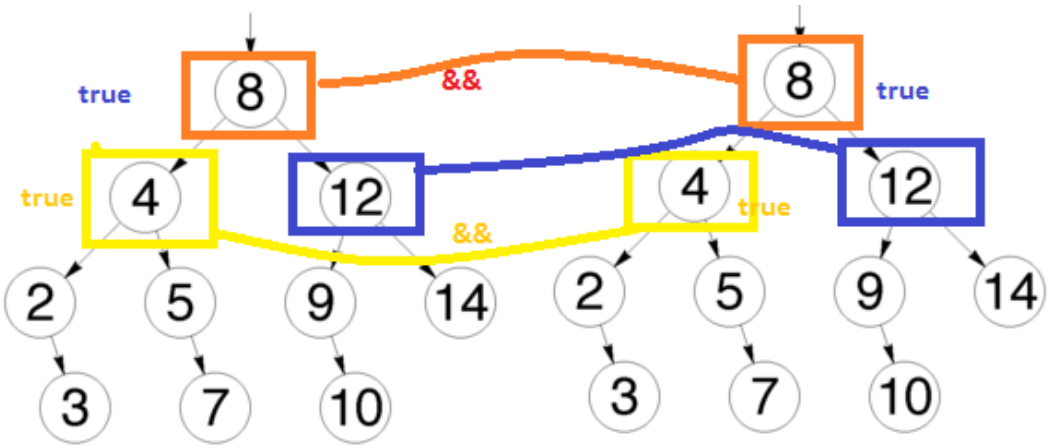
**Metoda vraca list koji se nalazi na najvecoj dubini.

```
	public CvorStabla najdubljiCvor(CvorStabla koren){
		if (koren==null || koren.levo==null && koren.desno==null) {
			return koren;
		}
		if (visina(koren.levo)>visina(koren.desno)) {
			return najdubljiCvor(koren.levo);
		}
		return najdubljiCvor(koren.desno);
	}
	public int visina(CvorStabla cvor) {
		if (cvor == null || (cvor.levo == null && cvor.desno == null))
			return 0;


		return 1 + max(visina(cvor.levo), visina(cvor.desno));
	}
	public int max(int a, int b) {
		if (a>b) {
			return a;
		}
		return b;
	}
```

**Metoda vraca najplici list

```
	public CvorStabla najpliciCvor(CvorStabla koren){

		if (koren==null|| (koren.levo==null && koren.desno==null)) {
			return null;
		}
		if (prviList(koren.levo)<prviList(koren.desno)) {
			return najpliciCvor(koren.levo);
		}
		return najpliciCvor(koren.desno);
	}

	public int prviList (CvorStabla koren){

		if (koren==null|| (koren.levo==null&&koren.desno==null)) {
			return 0;
		}
		return 1+ Math.min(prviList(koren.levo), prviList(koren.desno));
	}
```

Metoda proverava da li su dva stable identicna




```
	public boolean identicna(CvorStabla koren1, CvorStabla koren2){
		if (koren1==null && koren2==null ) {
			return true;
		}
		if (koren1!=null && koren2!=null) {
			if(koren1.podatak== koren2.podatak)
				return identicna (koren1.levo, koren2.levo) &&
					identicna (koren1.desno, koren2.desno);
		}
		return false;
	}
```

**** Metoda proverava jesu li dva stable slicna u ogledalu**

```
	public boolean jesuLiSlicna (CvorStabla koren1, CvorStabla koren2){
		if (koren1==null && koren2==null ) {
			return true;
		}
		if (koren1!=null && koren2!=null) {
			if(koren1.podatak== koren2.podatak)
				return	jesuLiSlicna(koren1.levo,	koren2.desno)&&
jesuLiSlicna(koren1.desno, koren2.levo);
		}
	return false;
}
```

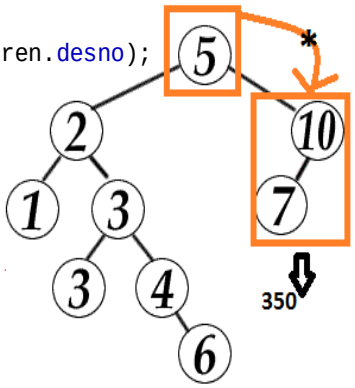
**** Broj cvorova koji imaju vecu vrednost od zadane**

```
	public int preboj (CvorStabla koren,int podatak){
		if (koren==null) {
```

```
                return 0;
            }
        if (koren.podatak>podatak)
            return 1+prebroj(koren.levo,podatak)+prebroj(koren.desno,podatak);

            return prebroj(koren.levo, podatak)+prebroj(koren.desno, podatak);
}
```

**Metoda racuna proizvod elemenata desnog podstabla

```
        public int proizvod (CvorStabla koren){
            if(koren==null)
                return 1;
        return koren.podatak*proizvod(koren.levo)*proizvod(koren.desno);
        }
        public int proizvodDesnoPodstabla (CvorStabla koren){
            if(koren==null|| koren.desno==null)
                return 0;
            return koren.podatak*proizvod(koren.desno);
}
```

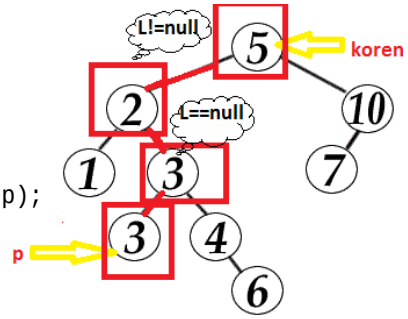


** Metoda iterativno ispisuje vrednosti od korena do zadatog cvora

```
        public void stampajOdKorenaDoZadatogCvora (CvorStabla koren, CvorStabla p){
            if(koren == null || koren==p)
                return;
            System.out.println(koren.podatak);
            while (koren!=p){
                if(pronadjiCvor(koren.levo, p)!=null){
                    System.out.println(koren.levo.podatak);
                    koren= koren.levo;
                    }else{
                        System.out.println(koren.desno.podatak);
                        koren=koren.desno;
                    }
                }
        }
```

Metoda rekurzivno stampa vrednosti u cvorovima od korena do datog cvora

```
        public void stampajOdKorenaDoCvoraRekurzivno(CvorStabla koren,CvorStabla
p){
        if(koren==null)
```




```
                return;
        System.out.println(koren.podatak);
        if (pronadjiCvor(koren.levo, p)!=null) {
           stampajOdKorenaDoCvoraRekurzivno(koren.levo, p);
                }

        if (pronadjiCvor(koren.desno, p)!=null) {
           stampajOdKorenaDoCvoraRekurzivno(koren.desno, p);
                }
}
```

**** Metoda ispisuje putanju od korena do cvora koji ima maksimalnu vrednost**

```
        public void stampajOdKorenaDoMaxCvora(){
                if(koren==null)
                        return;
                CvorStabla max = maxCvor(koren);
                stampajOdKorenaDoCvoraRekurzivno(koren, max);
                }
        public void stampajOdKorenaDoCvoraRekurzivno (CvorStabla koren, CvorStabla
p){
                if(koren==null)
                        return;
                System.out.println(koren.podatak);
                if (pronadjiCvor(koren.levo, p)!=null) {
                        stampajOdKorenaDoCvoraRekurzivno(koren.levo, p);
                        }

                if (pronadjiCvor(koren.desno, p)!=null) {
                        stampajOdKorenaDoCvoraRekurzivno(koren.desno, p);
                        }
                }
        public CvorStabla maxCvor(CvorStabla cvor) {
                if (cvor == null)
                        return null;
                CvorStabla max = cvor;
                CvorStabla maxl = maxCvor(cvor.levo);
                CvorStabla maxd = maxCvor(cvor.desno);
                if (maxl != null)
                        if (max.podatak < maxl.podatak)
                                max = maxl;
                if (maxd != null)
```

```
                if (max.podatak < maxd.podatak)
                        max = maxd;
        return max;
    }
```

**** Metoda racuna zbir vrednosti u cvorovima koji se nalaze na putanji od korena do zadatog cvora**

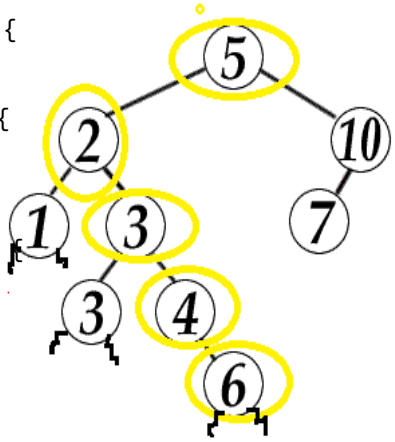
```
    public int zbirPutanja (CvorStabla pocetni, CvorStabla krajni) {
        if(pocetni==null)
            return 0;

        if (pronadjiCvor(pocetni.levo, krajnji)!=null) {
            return pocetni.podatak + zbirPutanja(pocetni.levo, krajnji);
        }
        if (pronadjiCvor(pocetni.desno, krajnji)!=null) {
            return pocetni.podatak + zbirPutanja(pocetni.desno, krajnji);
        }
        return krajnji.podatak;
    }
```

****Metoda stampa putanju izmedju dva zadana cvora u Avl stablu**

```
    public void stampajPutanjuAVLIzmedjuDvaCvora (CvorStabla koren, CvorStabla
    datiCvor) throws Exception{
        if(koren==null)
            throw new Exception("Greska");
        while(koren.podatak>datiCvor.podatak){
            System.out.println(koren.podatak);
            if(koren.podatak<datiCvor.podatak){
                koren=koren.desno;
                }
            else koren=koren.levo;
            }
        System.out.println(datiCvor.podatak);
    }
```

**** Metoda ispisuje putanju koja ima najvecu sumu od korena do lista**

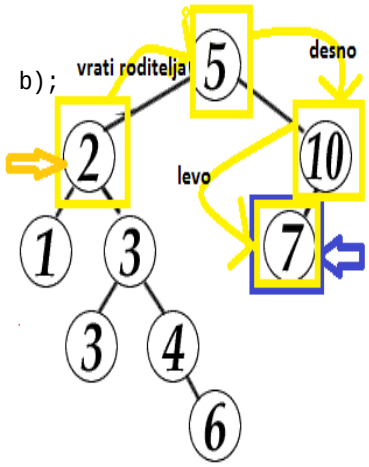
```
    public void ispisiputanjuSMaxSDoLista (CvorStabla k){
        System.out.println(k.podatak);
        if (k==null|| (k.levo==null&&k.desno==null)) {
            return;
        }
        if (zbirCvorova(k.levo)>zbirCvorova(k.desno)) {
            ispisiputanjuSMaxSDoLista(k.levo);
        }
        else
```



```
                ispisiputanjuSMaxSDoLista(k.desno);
        }

        public int zbirCvorova(CvorStabla cvor) {
                if (cvor == null)
                        return 0;
                return  cvor.podatak+  zbirCvorova(cvor.levo)+zbirCvorova(cvor.desno);

        }
```

****Metoda ispisuje putanju izmedju dva cvora u binarnom stablu**

```
        public void ispisiPutanju (CvorStabla k, CvorStabla a, CvorStabla b){
                if(k==null)
                        return;
                System.out.println(a.podatak);
                if (pronadjiCvor(a, b)==null) {
                  ispisiPutanju(k, vratiRoditelja(k, a), b);
                }
                if(pronadjiCvor(a.levo, b)!=null){
                   ispisiPutanju(k, a.levo, b);
                }
                if (pronadjiCvor(a.desno,b)!=null) {
                        ispisiPutanju(k, a.desno, b);
                }
           }
        }
```




```
        public CvorStabla pronadjiCvor(CvorStabla tekuci, CvorStabla P) {
                if (tekuci == null || tekuci==P)
                        return tekuci;
                CvorStabla l = pronadjiCvor(tekuci.levo, P);
                if (l != null)
                        return l;
                CvorStabla d = pronadjiCvor(tekuci.desno, P);
                if (d != null)
                        return d;
                return null;
                }
```

**** Metoda koja proverava da li je stablo BST**

```
        boolean daLiJeBst(CvorStabla koren) {
                if (koren == null)
```

```
                return true;
        if (koren.podatak > maxElement(koren.levo)
                        && koren.podatak < minElement(koren.desno)) {

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

    public int maxElement(CvorStabla cvor) {
        if (cvor == null)
                return Integer.MIN_VALUE;
        int max = cvor.podatak;
        int l = maxElement(cvor.levo);
        int d = maxElement(cvor.desno);
        if (max < l)
                max = l;
        if (max < d)
                max = d;
    return max;
    }
```

**** Metoda proverava da li je stablo AVL**

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

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

                if (rv > -2 && rv < 2)
                        return daLiJeAvl(cvor.levo) && daLiJeAvl(cvor.desno);
        }
        return false;
    }
    public boolean daLiJeBst(CvorStabla koren) {
        if (koren == null)
                return true;
        if (koren.podatak > maxElement(koren.levo)
                        && koren.podatak < minElement(koren.desno)) {

                return daLiJeBst(koren.levo) && daLiJeBst(koren.desno);
        }

        return false;
    }

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

```
		int l = maxElement(cvor.levo);
		int d = maxElement(cvor.desno);
		if (max < l)
			max = l;
		if (max < d)
			max = d;
		return max; }
```

Metoda koja vraca najveci cvor na putanji izmedju korena i cvora na koji je dat pokazivac

```
	public CvorStabla  vratiNajveciNaPutanji (CvorStabla k, CvorStabla b){

		if(k == null)
			return null;
		CvorStabla max = b;

		while (k!=b){
			if(k.podatak>max.podatak)
				max = k;
			if (pronadjiCvor(k.levo, b)!=null)
				k=k.levo;
			else
				k=k.desno;
		}

		return max;
	}
```

Metoda koja rekurzivno vraca najveci cvor na putanji izmedju korena i cvora na koji je dat pokazivac

```
	public CvorStabla vratiRekurzivno (CvorStabla k, CvorStabla b){
		if(k==null|| maxCvor(k)==k)
			return k;
		CvorStabla max=k;
		if(pronadjiCvor(k.levo, b)!=null){
			max = vratiRekurzivno (k.levo, b);
		}
		if(pronadjiCvor(k.desno, b)!=null){
			max = vratiRekurzivno (k.desno, b);
		}
		return max;
	}
	public CvorStabla pronadjiCvor(CvorStabla tekuci, CvorStabla P) {
		if (tekuci == null || tekuci==P)
			return tekuci;

		CvorStabla l = pronadjiCvor(tekuci.levo, P);
		if (l != null)
			return l;

		return pronadjiCvor (tekuci.desno, P);
		}
```

Dati su pokazivaci na koren i dva cvora u stablu. Metoda vraca zbir zajednickih predaha

```
	public int zbirPredaha (CvorStabla k, CvorStabla a, CvorStabla b){
		if(k==null)
			return 0;
		if (pronadjiCvor(k, a)!=null&& pronadjiCvor(k, b)!=null) {
			return k.podatak + zbirPredaha(k.levo,
a,b)+zbirPredaha(k.desno, a, b);
		}
		return 0;
	}
	public CvorStabla pronadjiCvor(CvorStabla tekuci, CvorStabla P) {
		if (tekuci == null || tekuci==P)
			return tekuci;

		CvorStabla l = pronadjiCvor(tekuci.levo, P);
		if (l != null)
			return l;

		return pronadjiCvor (tekuci.desno, P);
		}
```

B stabla

Ubacivanje:

– (1) Ako ima mesta u listu, uneti kljuc u list uz eventualna pomeranja

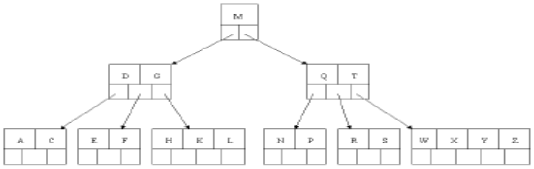
– (2) Ako je list pun, treba ga “pocepati”:
• (2.1) oko polovine kljuceva ostaje na starom listu
• (2.2) oko polovine kljuceva ide na novi list desno od postojeceg
• (2.3) srednji kljuc se “penje” u roditeljski cvor, pomeraju se kljucevi i pokazivac
– (3) Ako u roditeljskom (unutrašnjem) cvoru nema
mesta, i on se “cepa”:
• (3.1) oko polovine kljuceva ostaje na starom cvoru
• (3.2) oko polovine kljuceva ide u novi cvor desno od postojeceg
• (3.3.) srednji kljuc se “penje” u roditeljski cvor, uz pomeranje kljuceva i pokazivaca (rekurzivno korak 3)
– (4) Ako se “pocepa” koreni cvor, srednji kljuc ide u
novi koreni cvor; visina stabla se povecava za 1

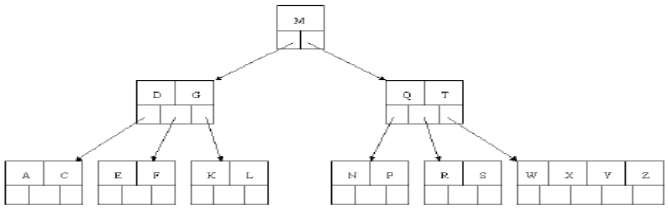
Izbacivanje:

• (1) Ako se briše kljuc iz lista onda
– (1.1) Ako ostaje dovoljno kljuceva, vrši se samo pomeranje
– (1.2) Ako ne ostaje dovoljno kljuceva onda
• (1.2.1) Ako susedni brat (cvor) ima “višak” kljuceva, najmanji kljuc iz desnog (odnosno najveci iz levog brata) se “penje” u cvor-prethodnik, a kljuc iz prethodnika se “spušta” u cvor iz kojeg smo izbrisali kljuc
• (1.2.2) Ako nijedan susedni brat nema “višak” kljuceva, cvor iz kojeg smo izbrisali kljuc spaja se sa levim ili desnim bratom, uz dodavanje “razdvojnog” kljuca iz cvora-prethodnika. Ako cvor-prethodnik nema dovoljno kljuceva, vrši se rotacija preko njegovog cvora-prethodnika, ili spajanje sa bratom

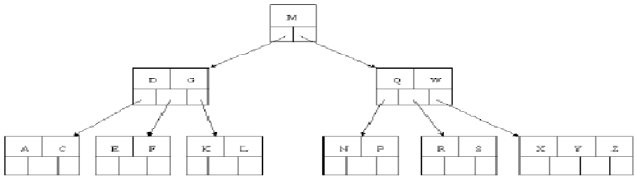
• (2) Ako se briše kljuc iz unutrašnjeg cvora, onda se umesto njega “penje” najmanji kljuc iz njegovog sledbenika; kljuc koji se penje briše se iz cvora u kome je bio – pravilo se primenjuje rekurzivno i
svodi na brisanje iz lista

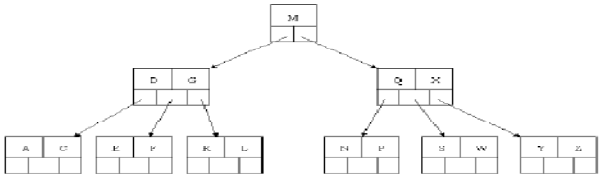
1.1 brise se H



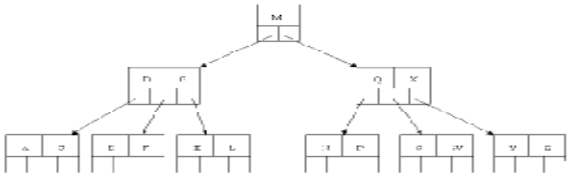


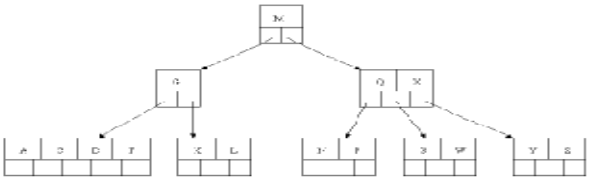
1.2.1 brise se R

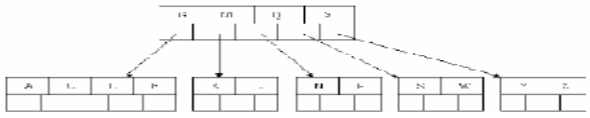




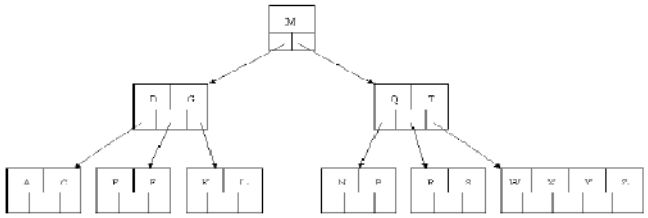
1.2.2 brise se E

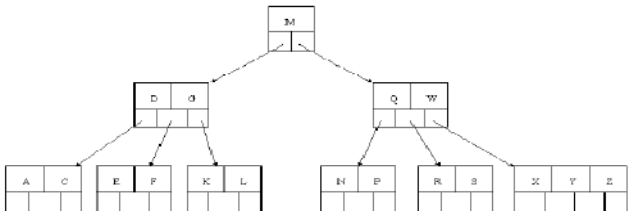




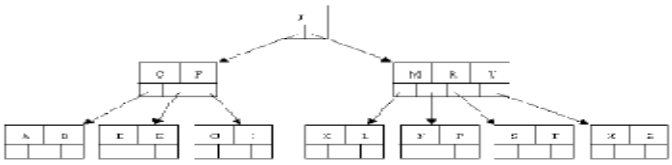


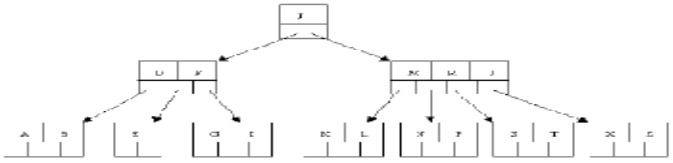
2. brise se T



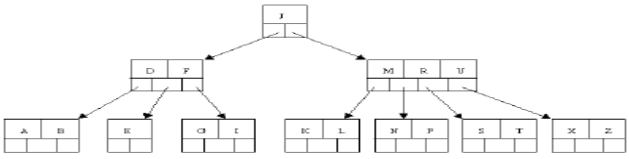


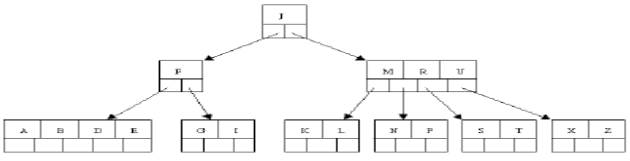
- Izbrisati ključ C – korak 1 (pravilo 2):



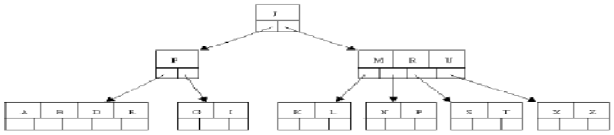


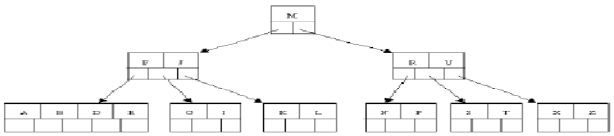
- Korak 2 (pravilo 1.2.2)





- Korak 3 (pravilo 1.2.2 – nast.). Čvor sa ključevima K, L promenio je prethodnika – i braću!



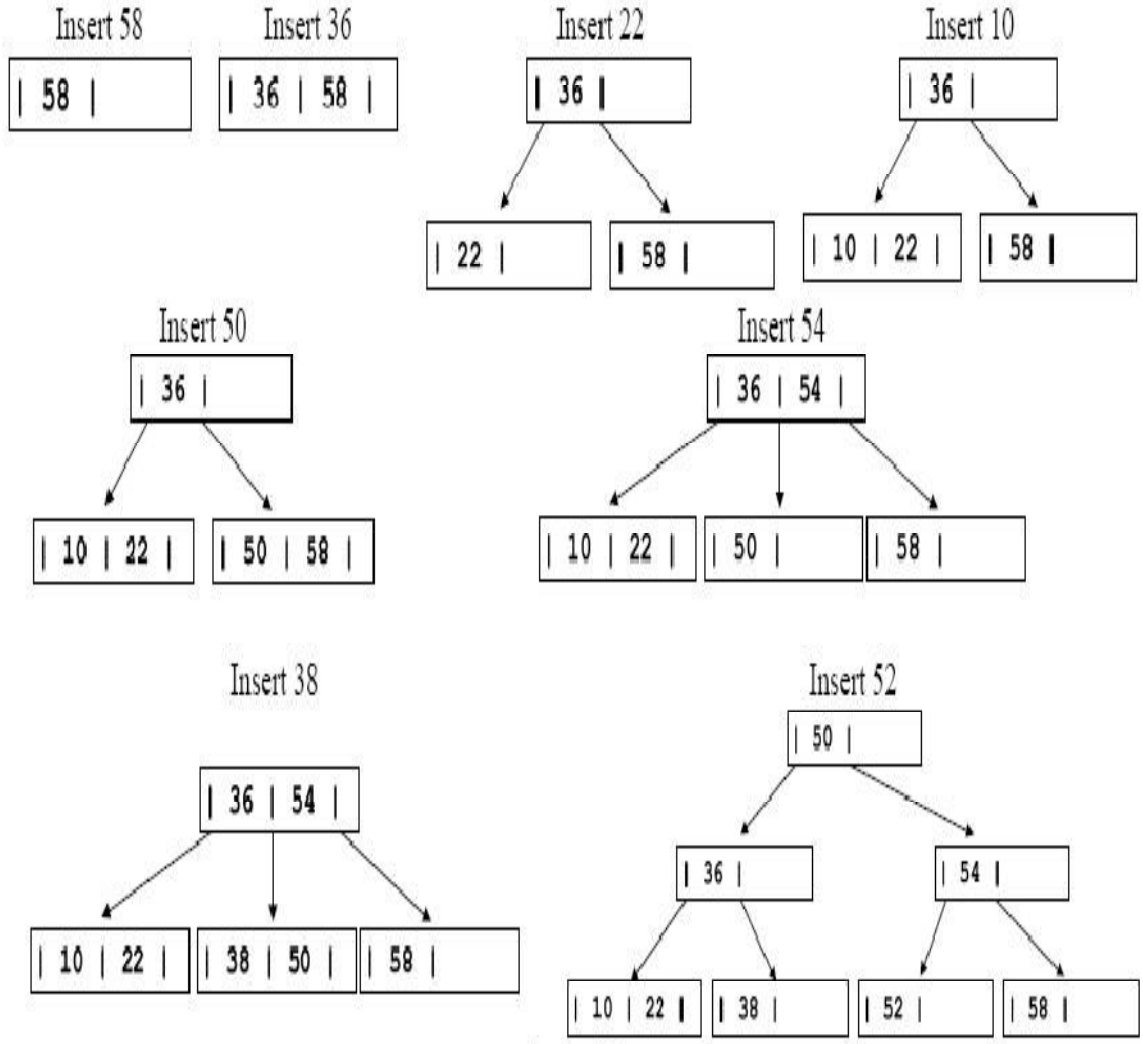


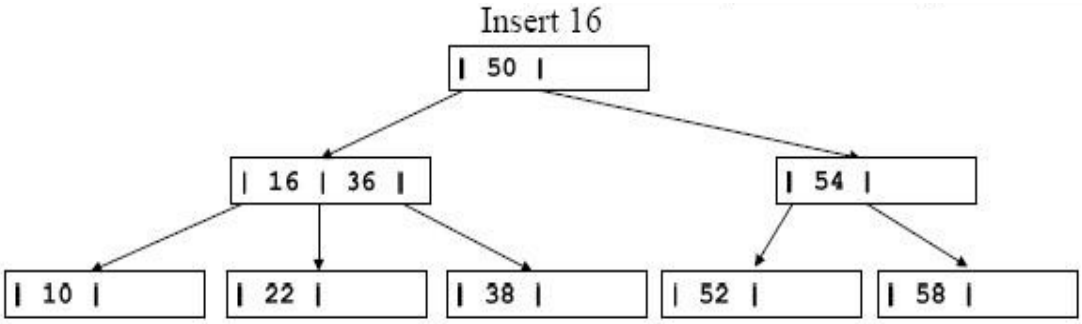
U inicijalno prazno B-stablo reda 3 umeću se redom ključevi 58, 36, 22, 10, 50, 54, 38, 52, 16, 40, 42, 12, 6, 27, 21 a zatim se redom brišu ključevi 50, 42, 38, 40, 52. Nacrtati izgled stabla nakon svake od navedenih izmena.

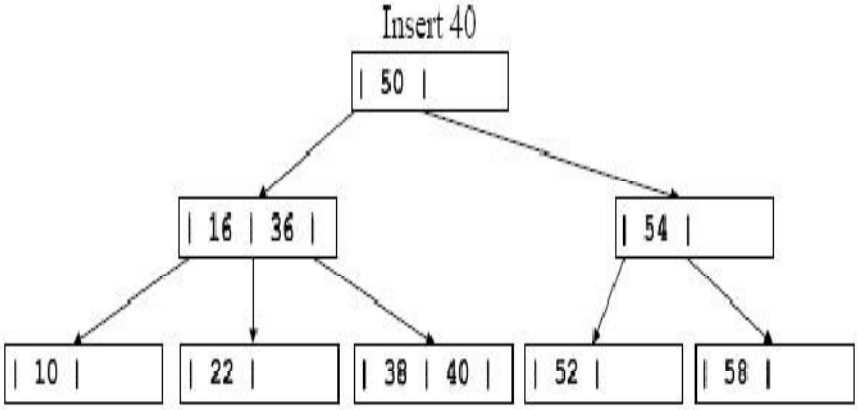
m=3

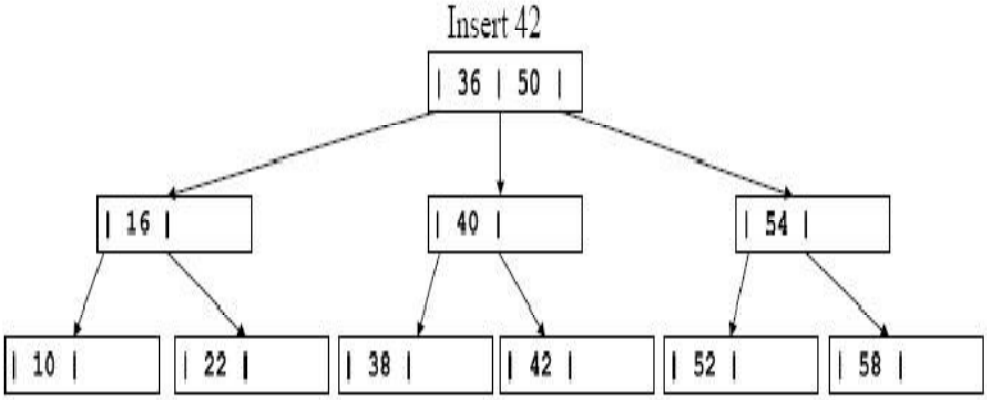
• svi čvorovi osim korena i listova imaju najmanje ⌈3/2⌉ = 2 podstabla

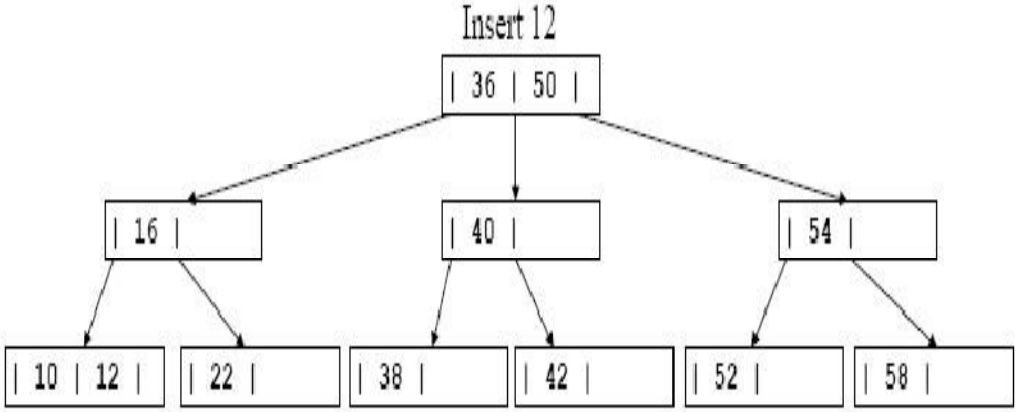
• listovi imaju najmanje ⌈3/2⌉ - 1 = 1 ključeva

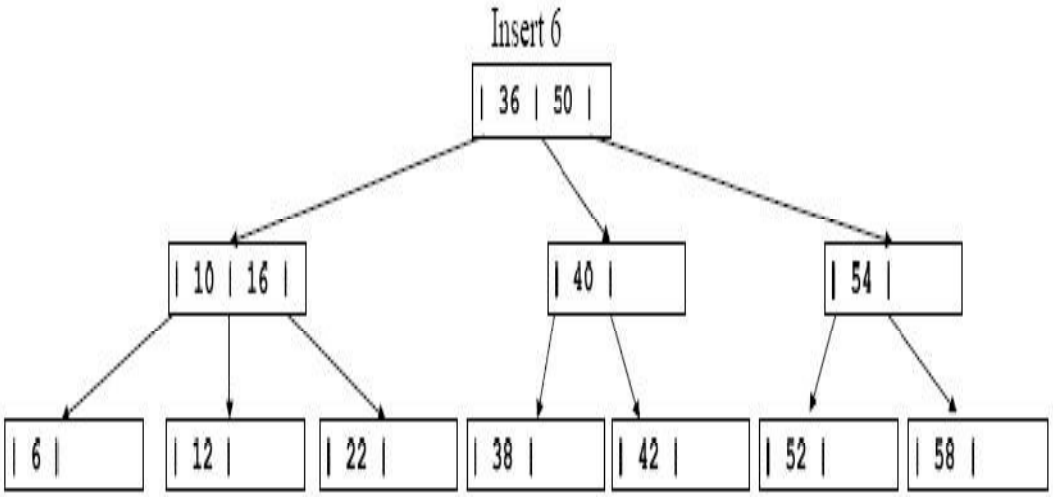


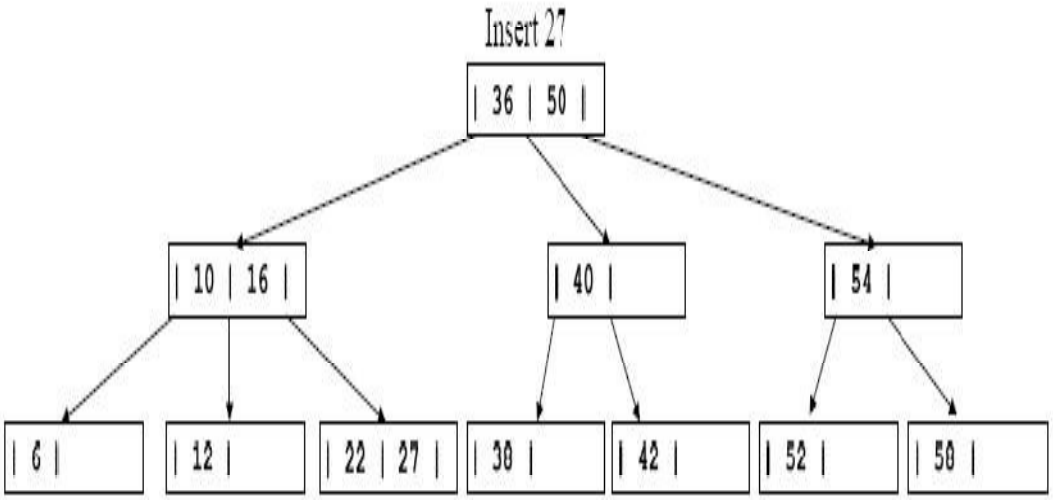

Insert 16
50
16
36
54
10
22
38
52
58

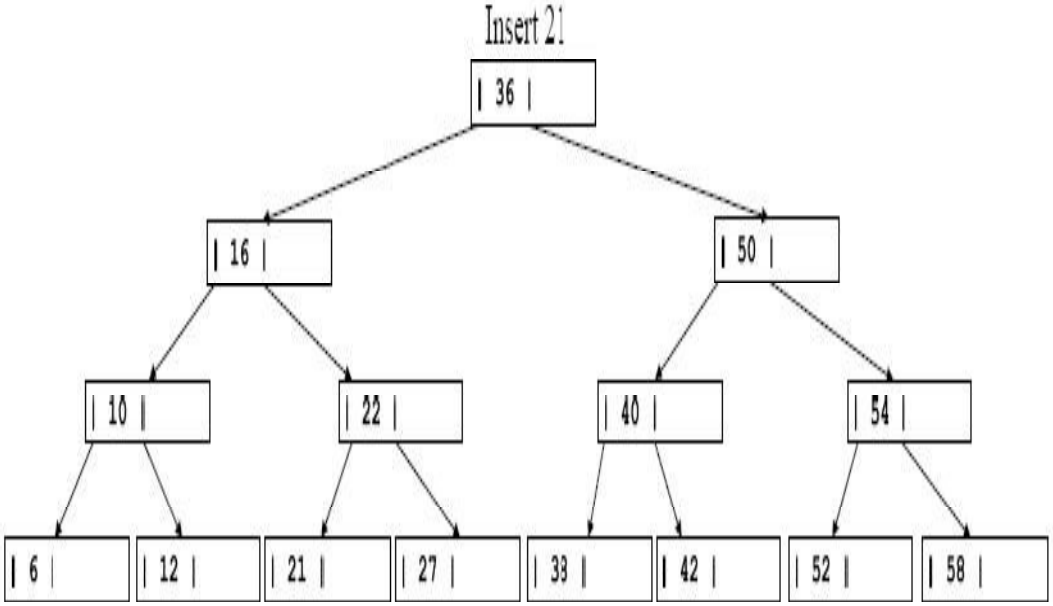

Insert 40
50
16
36
54
10
22
38
40
52
58

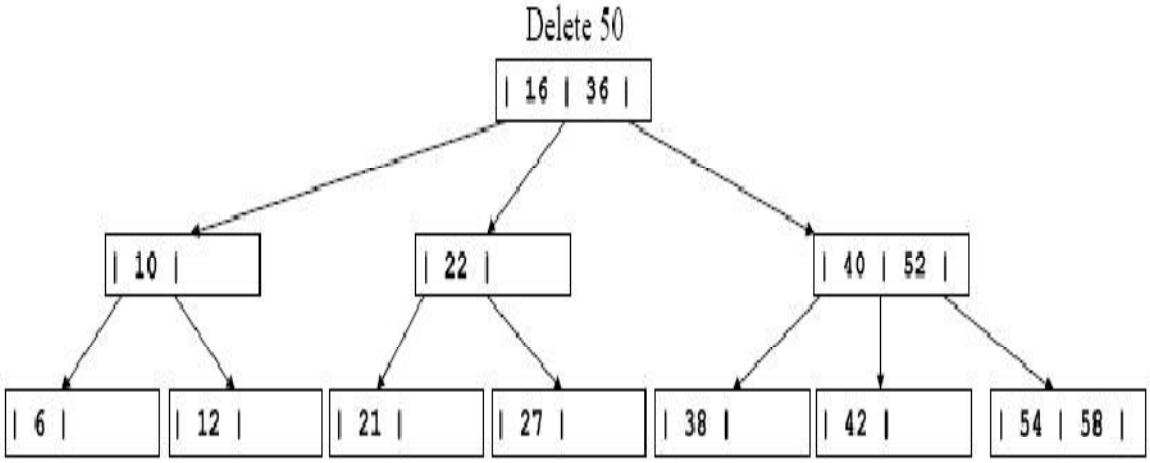

Insert 42
36
50
16
40
54
10
22
38
42
52
58

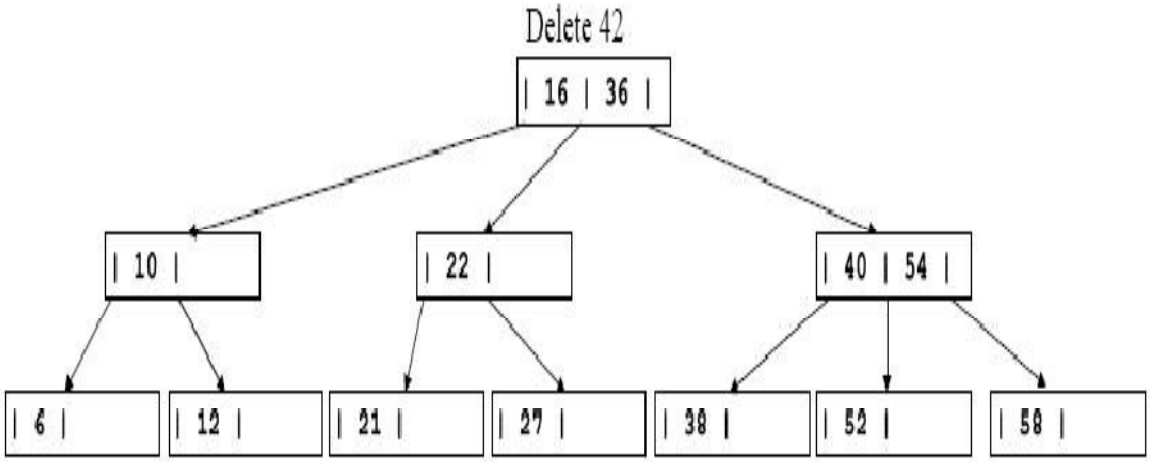

Insert 12
36
50
16
40
54
10
12
22
38
42
52
58

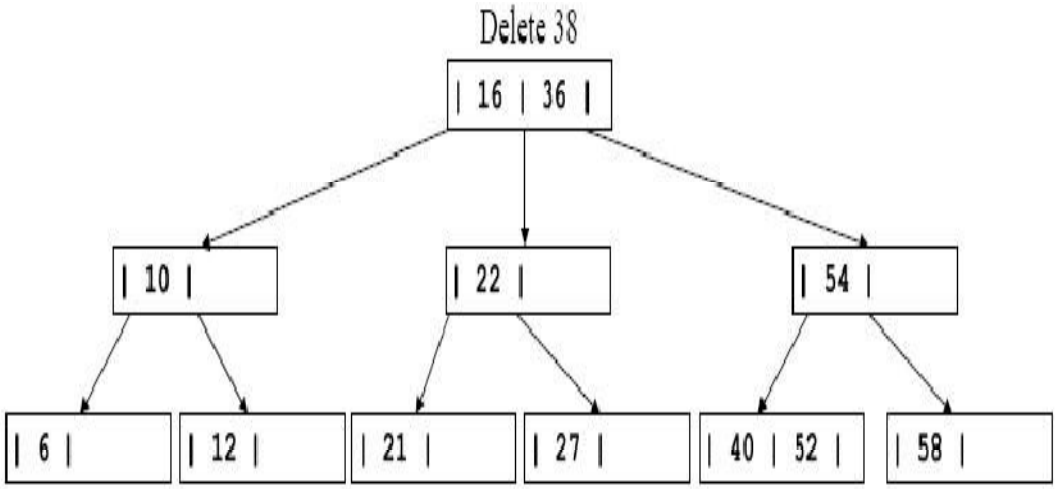

Insert 6
| 36 | 50 |
| 10 | 16 |
| 40 |
| 54 |
| 6 |
| 12 |
| 22 |
| 38 |
| 42 |
| 52 |
| 58 |

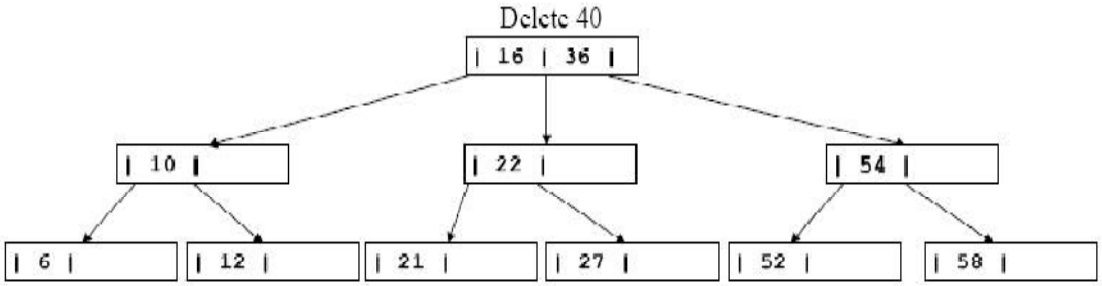

Insert 27
| 36 | 50 |
| 10 | 16 |
| 40 |
| 54 |
| 6 |
| 12 |
| 22 | 27 |
| 38 |
| 42 |
| 52 |
| 58 |

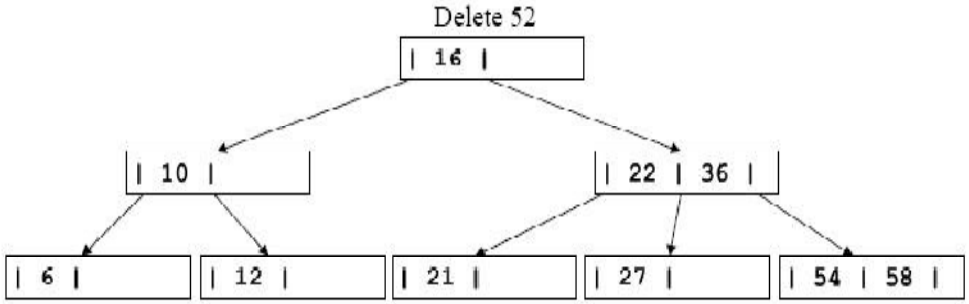

Insert 21
| 36 |
| 16 |
| 50 |
| 10 |
| 22 |
| 40 |
| 54 |
| 6 |
| 12 |
| 21 |
| 27 |
| 33 |
| 42 |
| 52 |
| 58 |


Delete 50
| 16 | 36 |
| 10 |
| 22 |
| 40 | 52 |
| 6 |
| 12 |
| 21 |
| 27 |
| 38 |
| 42 |
| 54 | 58 |

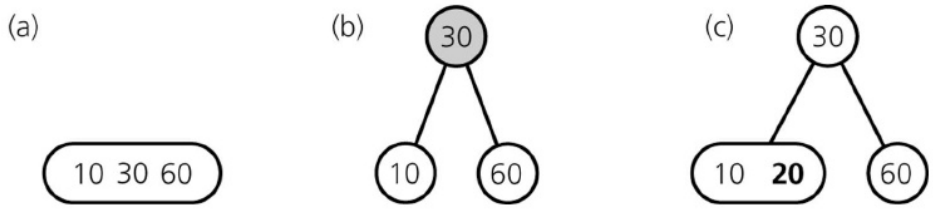

Delete 42
| 16 | 36 |
| 10 |
| 22 |
| 40 | 54 |
| 6 |
| 12 |
| 21 |
| 27 |
| 38 |
| 52 |
| 58 |


Delete 38
| 16 | 36 |
| 10 |
| 22 |
| 54 |
| 6 |
| 12 |
| 21 |
| 27 |
| 40 | 52 |
| 58 |

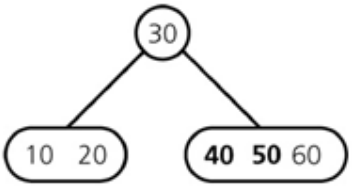




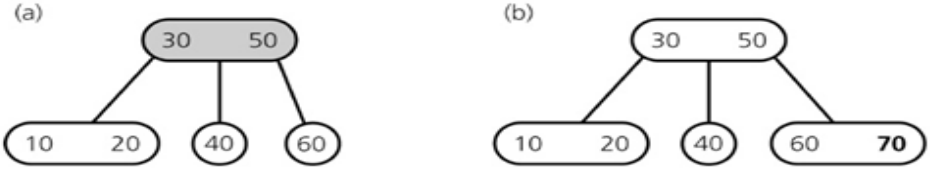
****U B stablo se ubacuju redom 60, 30, 10, 20 ...**



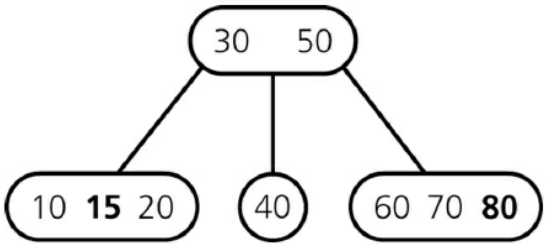
... 50, 40 ...



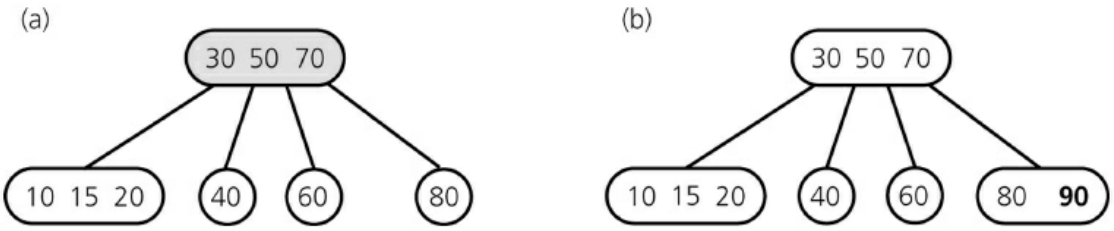
...70...



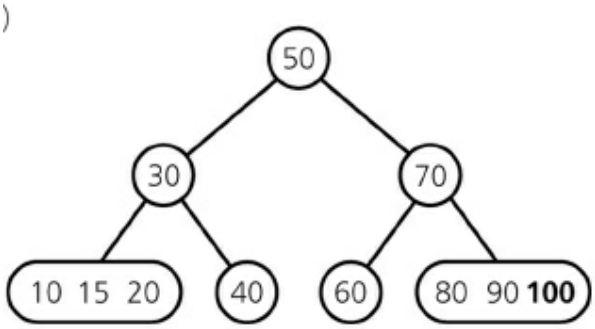
...80 15...



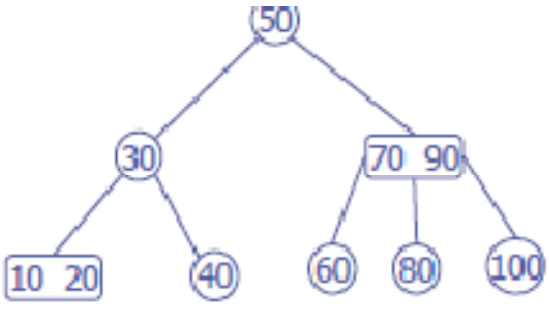
...90...

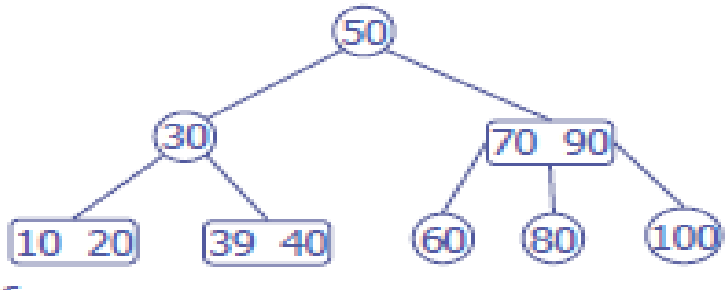


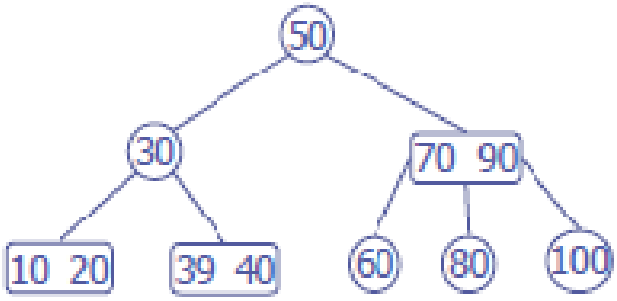
...100...

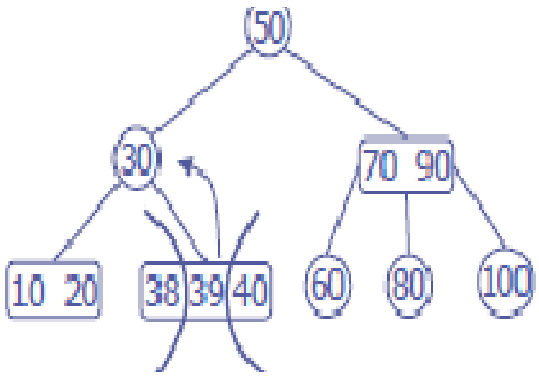


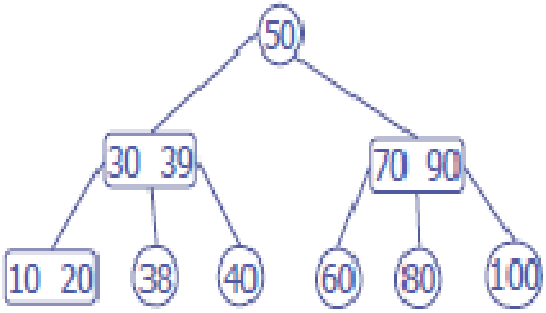
Neka je dato B stablo u koje se vrsi ubacivanje sledecih elemenata 39 38 36 35 34 33

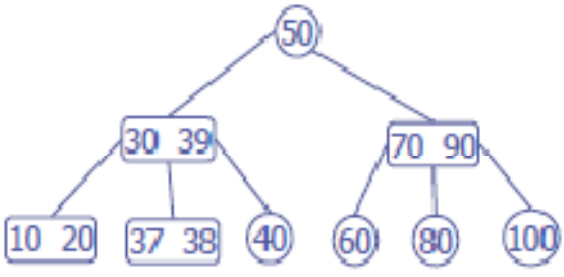


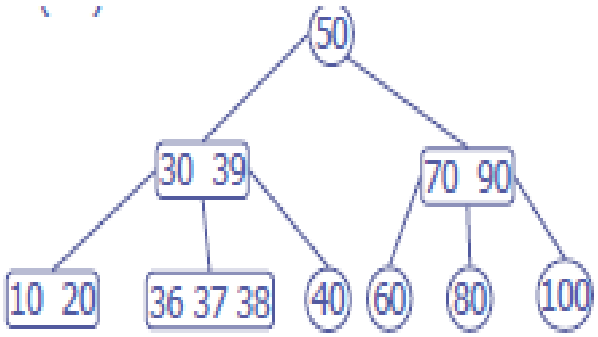

50
30
70 90
10 20
39 40
60
80
100

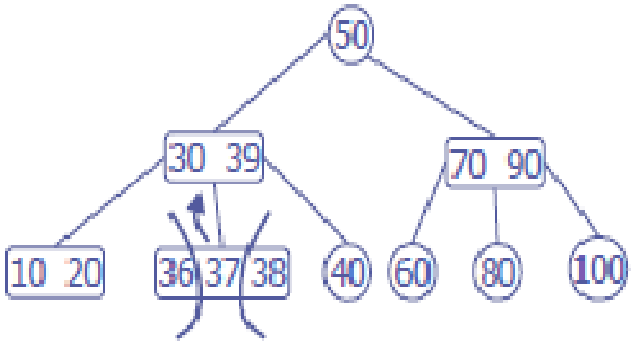

50
30
70 90
10 20
39 40
60
80
100

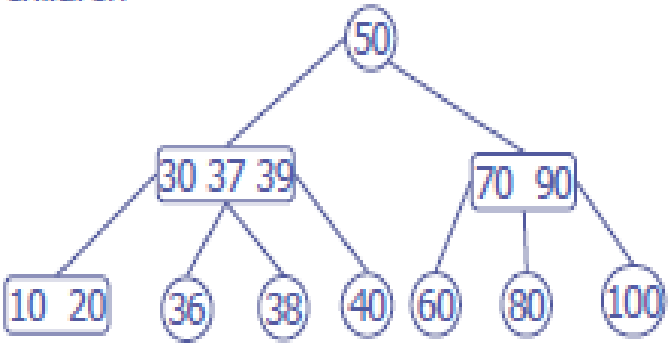

50
30
70 90
10 20
38 39 40
60
80
100

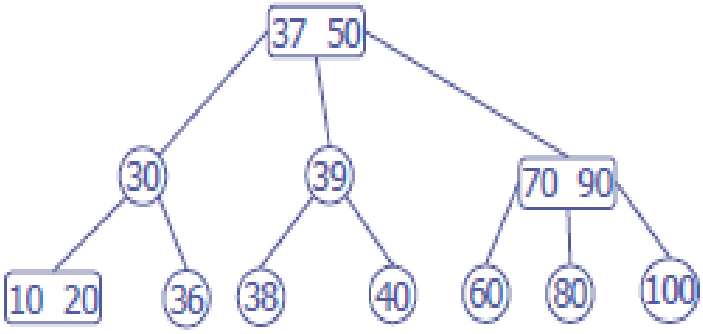

50
30 39
70 90
10 20
38
40
60
80
100

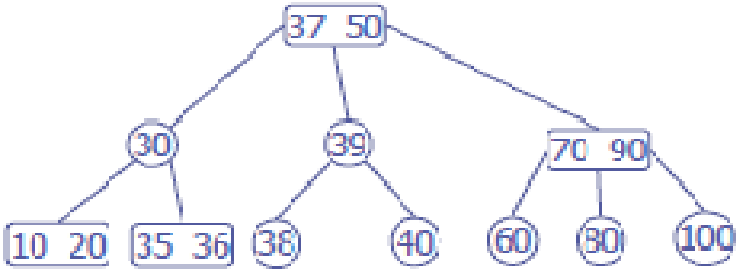

50
30 39
70 90
10 20
37 38
40
60
80
100

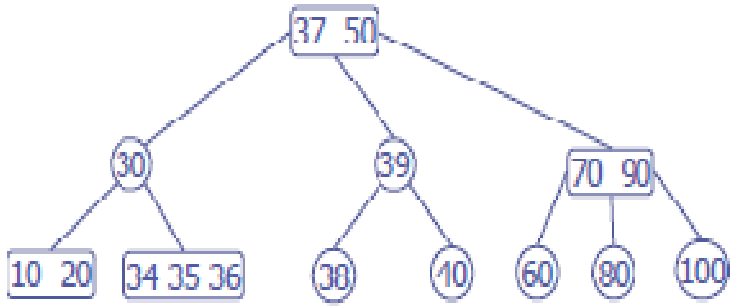

50
30 39
70 90
10 20
36 37 38
40
60
80
100

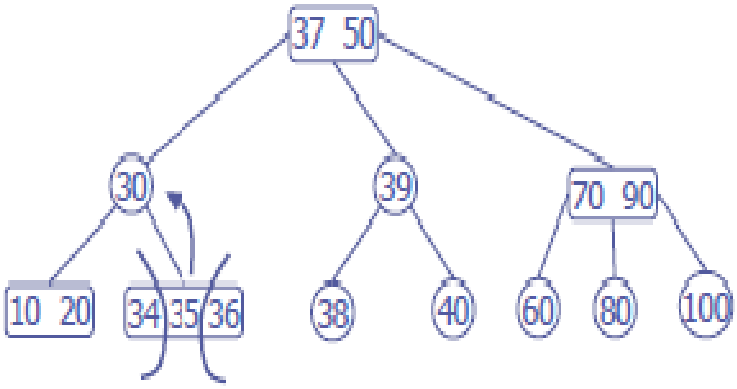

50
30 39
70 90
10 20
36 37 38
40
60
80
100

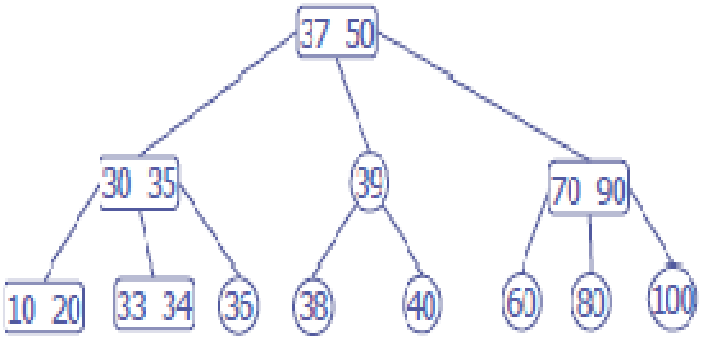

50
30 37 39
70 90
10 20
36
38
40
60
80
100


37 50
30
39
70 90
10 20
36
38
40
60
80
100

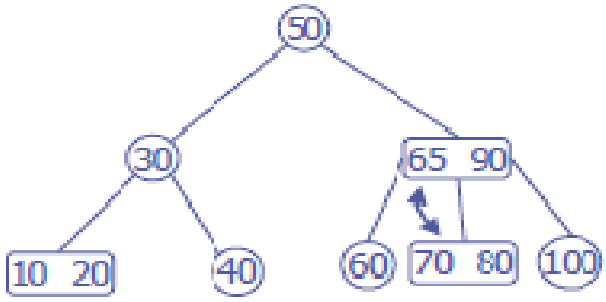

37 50
30
39
70 90
10 20
35 36
38
40
60
80
100

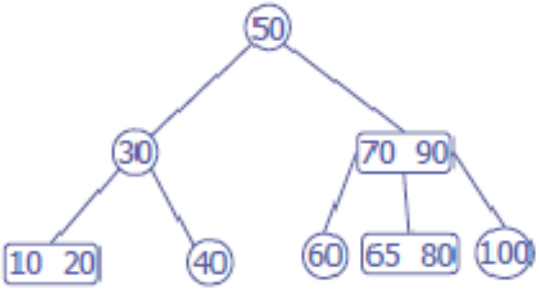


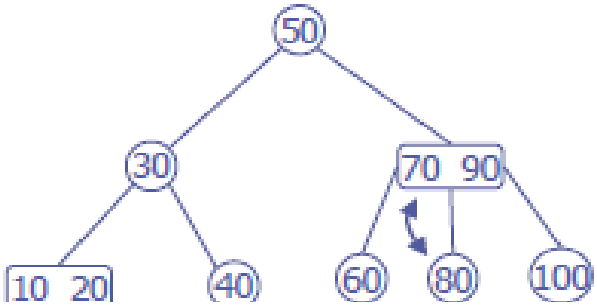


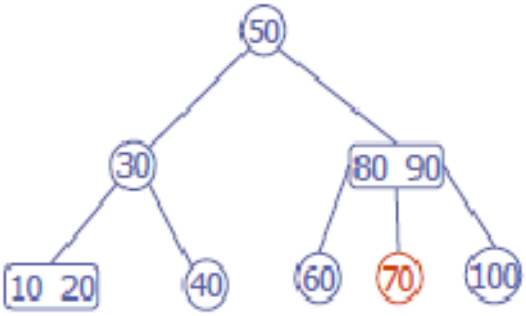


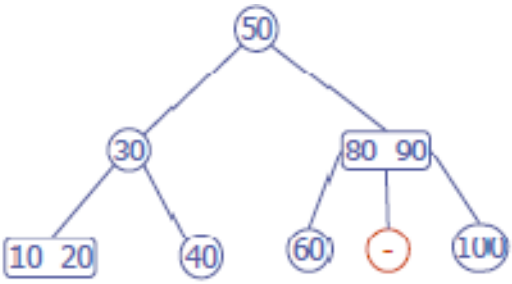
*****Sada se vrsi brisanje sledecih cvorova 65 70 100 80*****

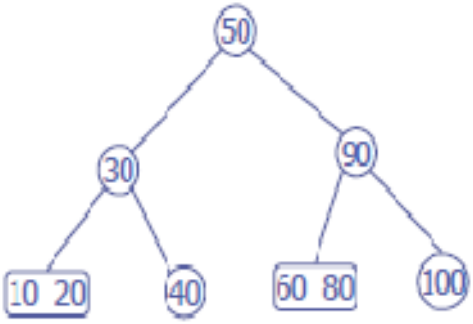


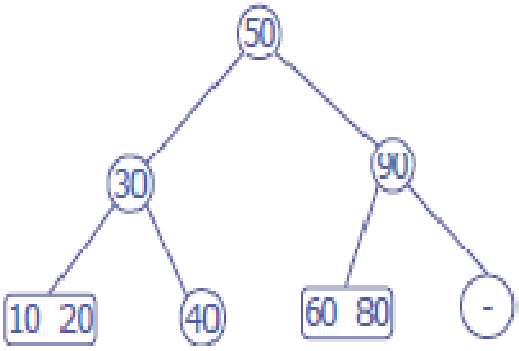


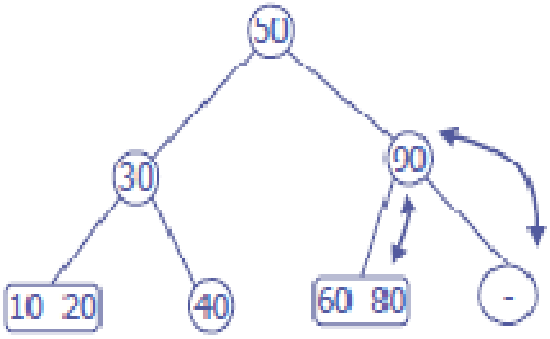

50
30
70 90
10 20
40
60
80
100

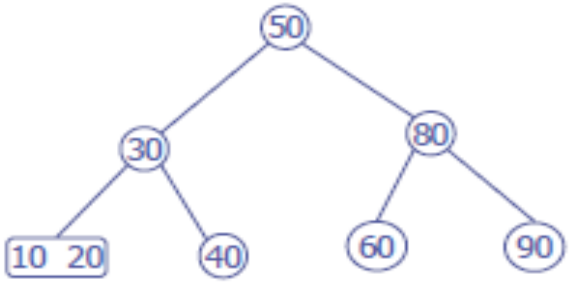

50
30
80 90
10 20
40
60
70
100

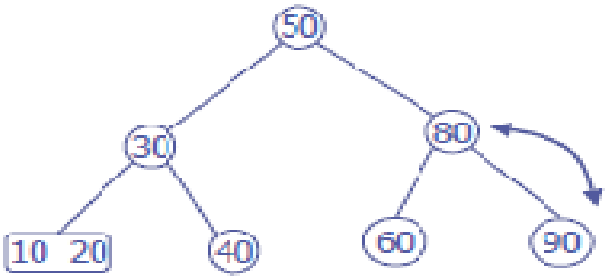

50
30
80 90
10 20
40
60
-
100

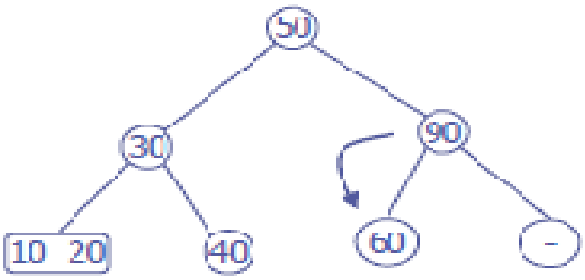

50
30
90
10 20
40
60 80
100

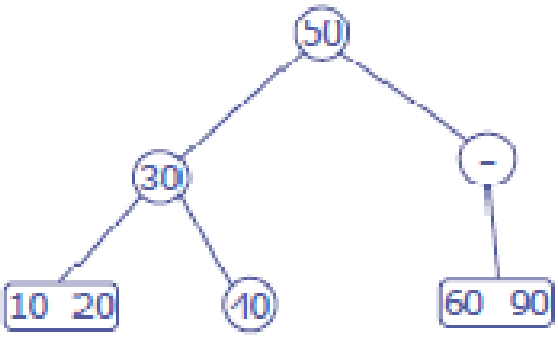

50
30
90
10 20
40
60 80
-

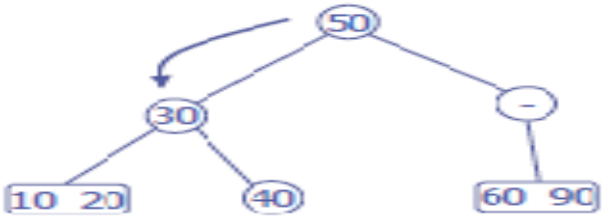

50
30
90
10 20
40
60 80
-

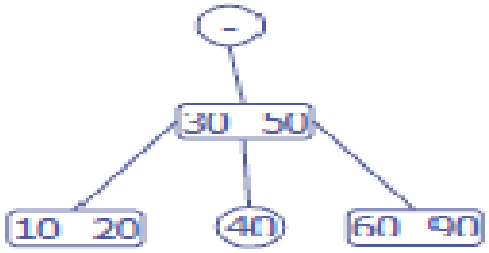

50
30
80
10 20
40
60
90

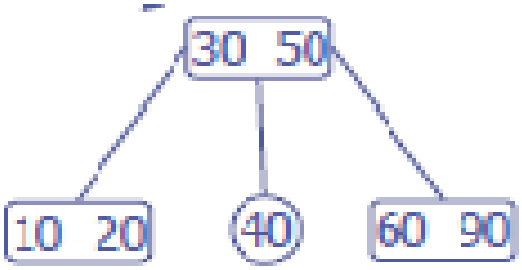

50
30
80
10 20
40
60
90


50
30
90
10 20
40
60
-


50
30
-
10 20
40
60 90







B* stablo

Unošenje (uvek u list) – analogno unošenju u B-stablo, OSIM što:

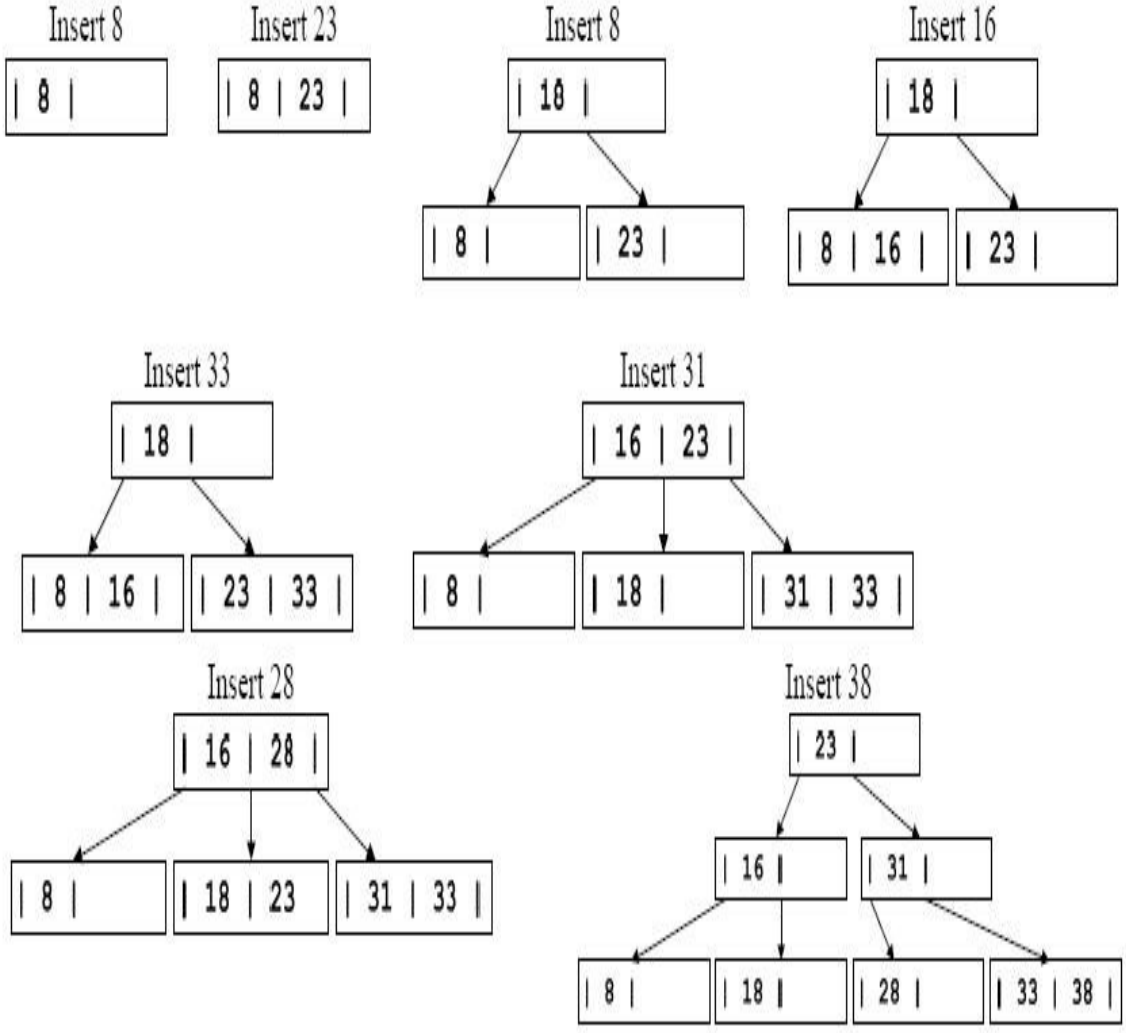
– Ako je list pun, pokušava se *prelivanje* desnim ili levim bratom:
• urede se svi kljucevi iz lista, izabranog brata, razdvojni kljuc (iz neposrednog prethodnika) i kljuc koji se unosi (ukupno m-1+k+1+1 kljuceva)
• floor((m+k+1)/2) kljuceva (manjih ili vecih, u zavisnosti od toga da li se preliva iz desnog ili levog brata) ostaje na starom listu, a ostali idu u desni / levi brat (središnji se “penje”)
– Ako prelivanje ne uspe, cepaju se 2 cvora (i puni list i njegov brat) na 3 cvora, tako što u prvi ide floor((2m-2)/3), u drugi floor((2m-1)/3) a u treci floor((2m)/3) kljuceva, a u cvor neposredni prethodnik idu dva razdvojna kljuca

– Prethodno pravilo se propagira po potrebi sve do korena

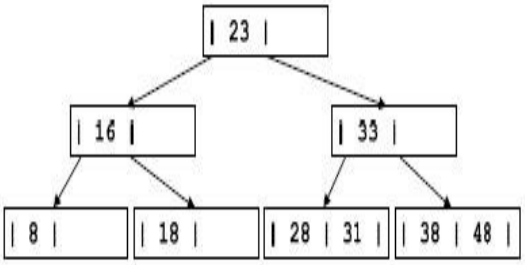
**U prazno B*-stablo reda 3 umeću se redom ključevi 8, 23, 18, 16, 33, 31, 28, 38, 48, 40, 57 a zatim se redom brišu ključevi 18, 31, 57, 40. Nacrtati izgled stabla nakon svake od navedenih izmena.

m=3

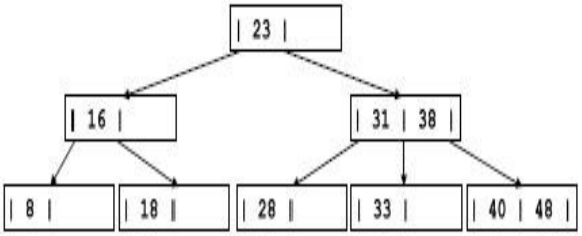
- svaki čvor osim listova i korena ima najmanje $\lceil \frac{2m-1}{3} \rceil = \lceil \frac{6-1}{3} \rceil = 2$ podstabla
- koren ima najmanje 2, a najviše $2\lfloor \frac{2m-2}{3} \rfloor + 1 = 2\lfloor \frac{6-2}{3} \rfloor + 1 = 3$ podstabla
- nakon podele nekog čvora, ključevi se raspoređuju u odnosu 1-1-2



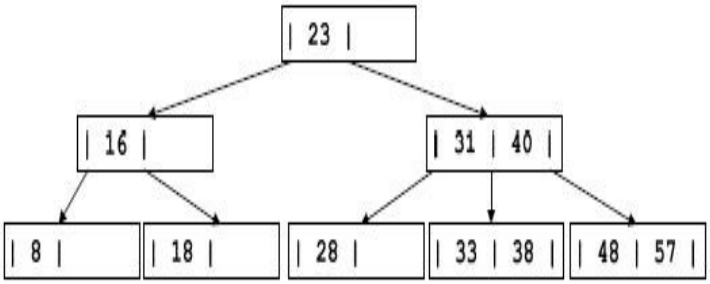
Insert 48



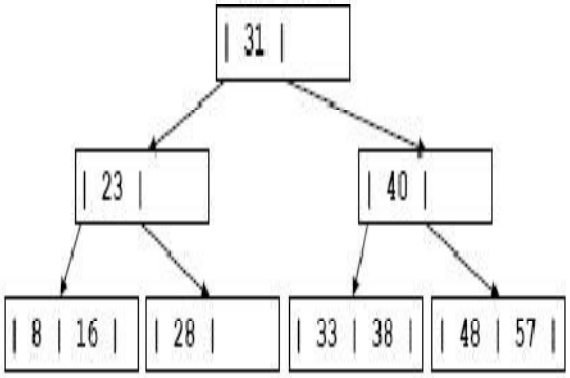
Insert 40



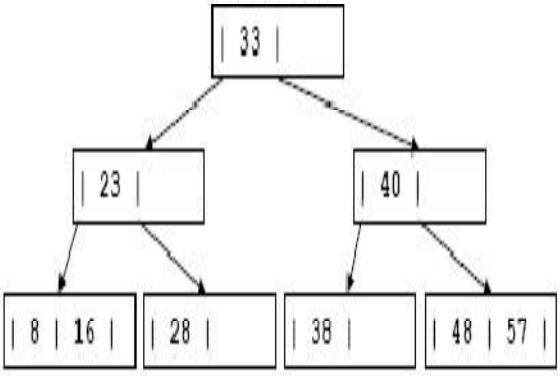
Insert 57



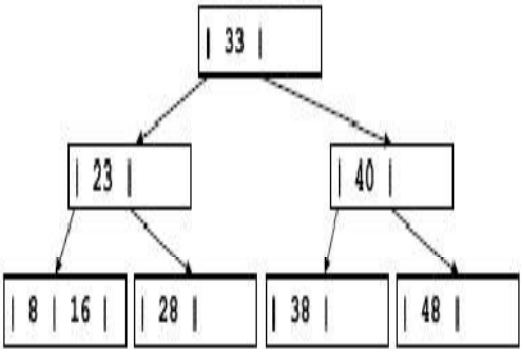
Delete 18



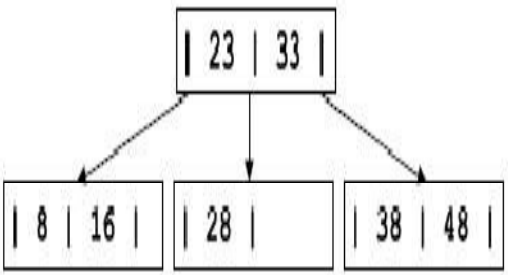
Delete 31



Delete 57



Delete 40

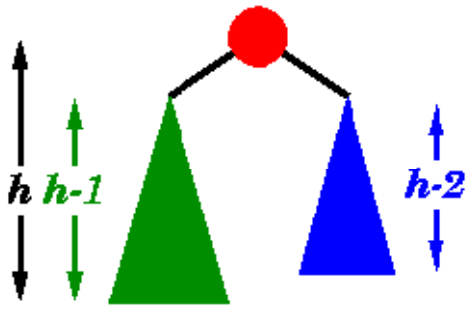


**AVL stablo*

AVL stablo je binarno stablo pretrage kod koga je za svaki čvor apsolutna vrednost razlike visina levog i desnog podstabla manja ili jednaka od jedan.

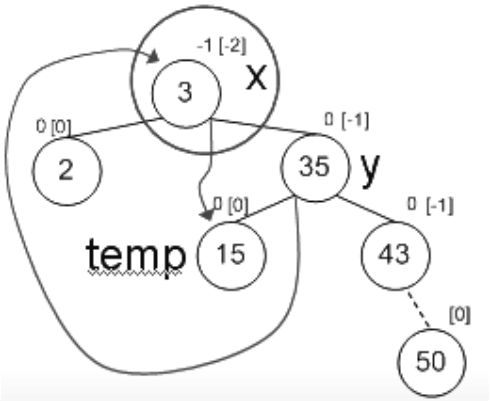
Visina stabla je visina njegovog korena, tj. maksimalno rastojanje od korena do nekog čvora..

Visina čvora *x* je rastojanje od *x* do najdaljeg potomka.



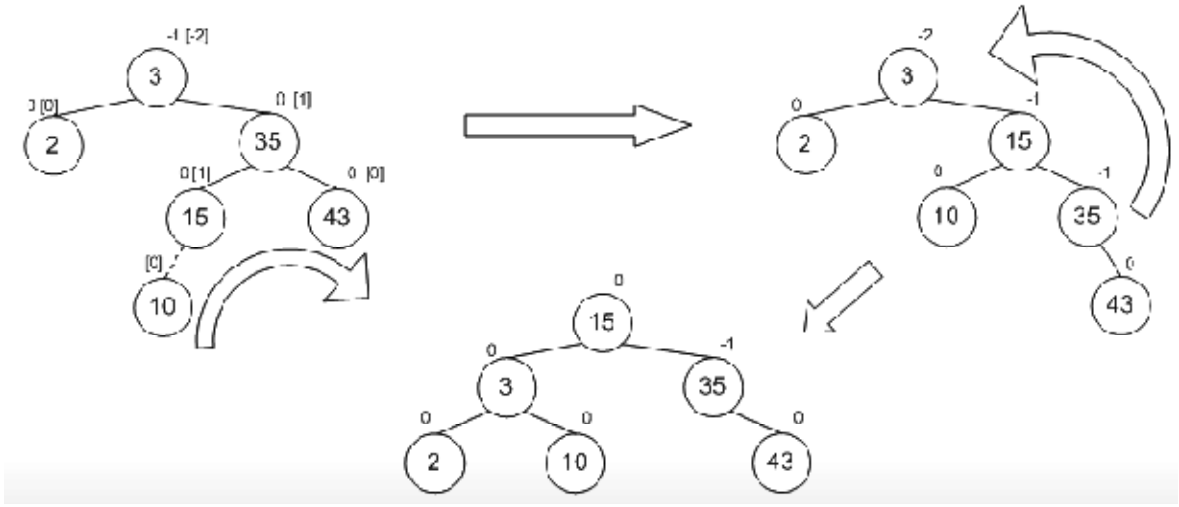
Potreba za rebalansiranjem AVL stabla se javlja kod umetanja lista, kada je neki njegov predak već nagnut (**kritičan čvor**) i nakon umetanja nagnuće prelazi dozvoljenu granicu. Primer rebalansiranja umetanjem čvora 50 na slici u primeru 1:

Nakon umetanja lista, sin kritičnog čvora naginje na istu stranu kao i kritični čvor (potrebna rotacija ulevo ili udesno).

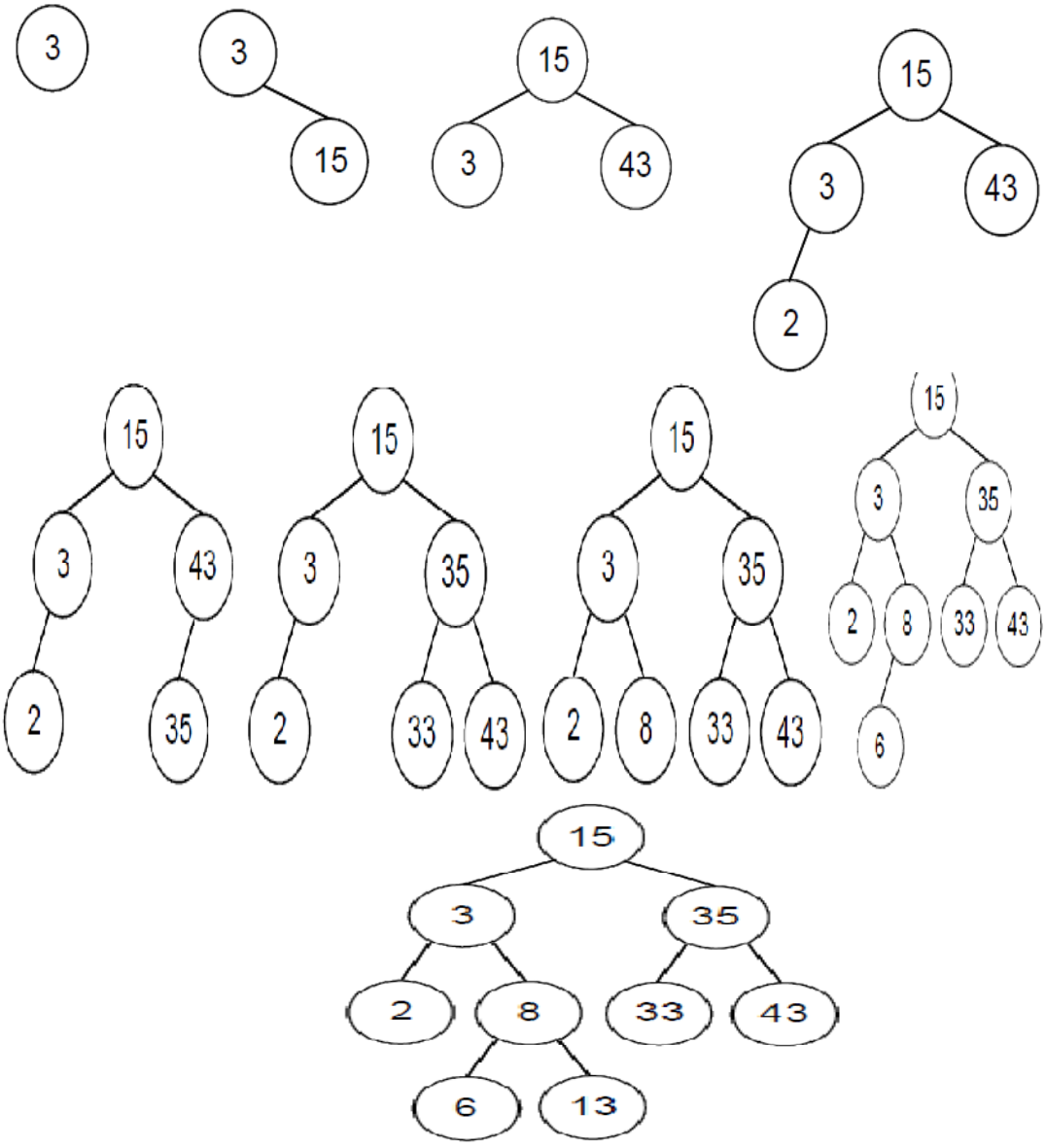


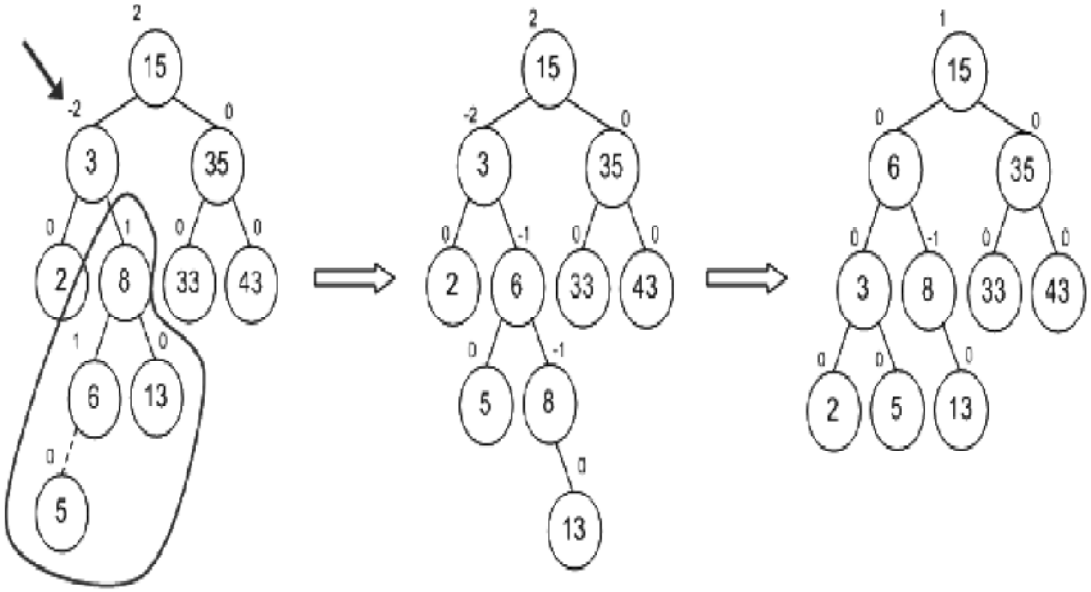
Primer rebalansiranja umetanjem čvora 10 na slici u primeru 1:

Nakon umetanja lista, sin kritičnog čvora naginje na suprotnu stranu od kritičnog čvora (potrebna dvostruka rotacija, u različitim smerovima).



***** Formirati AVl stablo od sledecih brojeva 3 15 43 2 35 33 8 16 13**




2
15
-2
0
3
35
0
1
0
0
2
8
33
43
1
0
6
13
0
5
2
15
-2
0
3
35
0
-1
0
0
2
6
33
43
0
-1
5
8
0
13
1
15
0
0
6
35
0
-1
0
0
3
8
33
43
0
0
0
2
5
13

***** Hashing clasa ****

```
public class Hashing {

public int[] niz;
public int brEl;

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

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

***** Metoda vrsi ubacivanje novog elementa pomocu hash funkcije*****

```
public void ubaci(int podatak) {

	if (brEl == niz.length)
		return;

	int adresa = hashFunkcija(podatak);

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

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

***** Metoda pronalazi trazeni podatak preko hash funkcije*****

```
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 novi element koristeci najpre hash fukciju da ga pronadje*****

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

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

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

*****Hashing Olancavanjem*****

```
	public class HasningOlancavanjem {

	public CvorJSListe[] niz;

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

***** Metoda ubacuje podatak u hash niz*****

```
	public void ubaci(int podatak) {

		if (niz.length == 0)
			return;

		int adresa = hashFunkcija(podatak);
		if (niz[adresa] == null)
			niz[adresa] = new CvorJSListe(podatak, null);
		else {
			CvorJSListe pom = niz[adresa];
			while (pom.sledeci != null)
				pom = pom.sledeci;

			pom.sledeci = new CvorJSListe(podatak, null);
		}
	}
```

***** Metoda trazi prosledjenu vrednost i 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 i 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*****

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

		if (adresa == -1)
			return;
		CvorJSListe pom = niz[adresa];
		if (pom.podatak == podatak)
			niz[adresa] = pom.sledeci;
		else {
			while (pom.sledeci.podatak != podatak)
				pom = pom.sledeci;
			pom.sledeci = pom.sledeci.sledeci;
		}
	}
```