Zadaci za samostalnu vježbu za 3. blic

ZADACI:

Red

Zadatak 1

Napisati funkcije DodajURed, SkiniIzReda i Prebroji za red realiziran cirkularnim poljem, ako su u glavnom programu definirane sljedeće varijable:

```
int red[MAXRED];
int ulaz, izlaz;
```

Funkcije DodajURed i SkiniIzReda u slučaju pogreške vraćaju 0, inače 1.

Zadatak 2

Napišite funkcije DodajURed, SkiniIzReda i Prebroji za red realiziran listom, ako je zadana struktura:

```
struct at {
      int element;
      struct at *sljed;
};
typedef struct at atom;
```

Jednostruko i dvostruko povezane liste

Zadatak 1

Napišite funkciju za dodavanje elemenata na stog realiziran jednostruko povezanom listom ako je zadan sljedeći prototip funkcije:

```
int dodaj (cvor **vrh, int element);
```

Zadatak 2

U jednostruko povezanu listu spremaju se zapisi sljedećeg tipa:

```
typedef struct s{
  int pbr;                  // poštanski broj
  char mjesto[100+1];       // naziv mjesta
  struct s *sljed;
} zapis;
```

Kako glasi funkcija koja pronalazi u listi zapis o mjestu sa zadanim poštanskim brojem i vraća taj zapis u glavni program. Ako takav zapis ne postoji u listi funkcija vraća NULL. Lista nije sortirana.

Stabla

Zadatak 1

Napisati **nerekurzivnu** funkciju koja vraća razinu na kojoj se nalazi najveći element u stablu. U stablu su pohranjeni cjelobrojni elementi i stablo je sortirano (lijevo manji, desno veći).

Zadatak 2

Napisati **rekurzivnu** funkciju koja vraća razinu na kojoj se nalazi najveći element u stablu. U stablu su pohranjeni cjelobrojni elementi i stablo je sortirano (lijevo manji, desno veći).

Gomila kao prioritetni red, heapsort

Zadatak 1

Kako izgleda ispis gomile po razinama ako je gomila formirana za ulazni niz 25 35 12 40 1 90 7 15 algoritmom čija je složenost za najgori slučaj O(n)?

Zadatak 2

Zadana je gomila koja je pohranjena u polju:

88	66	77	33	55	44	11	22

Prikažite postupak uzlaznog heapsorta.

RJEŠENJA:

Red

Zadatak 1

```
int DodajURed (tip element, tip red[], int n, int izlaz, int *ulaz) {
      if (((*ulaz+1) % n) == izlaz) return 0;
      (*ulaz)++;
      *ulaz %= n;
      red [*ulaz] = element;
      return 1;
}

int SkiniIzReda (tip *element, tip red[], int n, int *izlaz, int ulaz) {
      if (ulaz == *izlaz) return 0;
      (*izlaz) ++;
      *izlaz %= n;
      *element = red[*izlaz];
      return 1;
}

int Prebroji (int n, int izlaz, int ulaz) {
      if (ulaz >= izlaz) {
            return (ulaz - izlaz);        // standardno
      } else {
            return (ulaz - izlaz + n);    // cirkularnost
      }
}
```

Zadatak 2

```
int DodajURed (int element, atom **ulaz, atom **izlaz) {
      atom *novi;
      if (novi = malloc (sizeof (atom))) {
            novi->element = element;
            novi->sljed = NULL;
            if (*izlaz == NULL) {
                  *izlaz = novi;          // ako je red bio prazan
            } else {
                  (*ulaz)->sljed = novi;  // inace, stavi na kraj
            }
            *ulaz = novi;                 // zapamti zadnjeg
            return 1;
      }
      return 0;
}

int SkiniIzReda (int *element, atom **ulaz, atom **izlaz) {
      atom *stari;
      if (*izlaz) {                       // ako red nije prazan
            *element = (*izlaz)->element; // element koji se skida
            stari = *izlaz;               // zapamti trenutni izlaz
            *izlaz = (*izlaz)->sljed;     // novi izlaz
            free (stari);                 // oslobodi memoriju skinutog
            if (*izlaz == NULL) *ulaz = NULL; // prazan red
            return 1;
  }
      return 0;
}
```

```
int Prebroji (atom *izlaz) {
      int n;
      for (n = 0; izlaz; n++, izlaz = izlaz->sljed);
      return n;
}
```

Jednostruko i dvostruko povezane liste

Zadatak 1

```
int dodaj (atom **vrh, int element) {
  cvor *novi;
  if ((novi= (atom *) malloc(sizeof(cvor))) == NULL) return 0;
  novi->element = element;
  novi->sljed = *vrh;
  *vrh = novi;

  return 1;
}
```

Zadatak 2

```
  zapis *nadji(zapis *glava, int pbr) {
    while (glava && (glava->pbr != pbr))
      glava = glava->sljed;
    return glava;
  }
```

Stabla

Zadatak 1

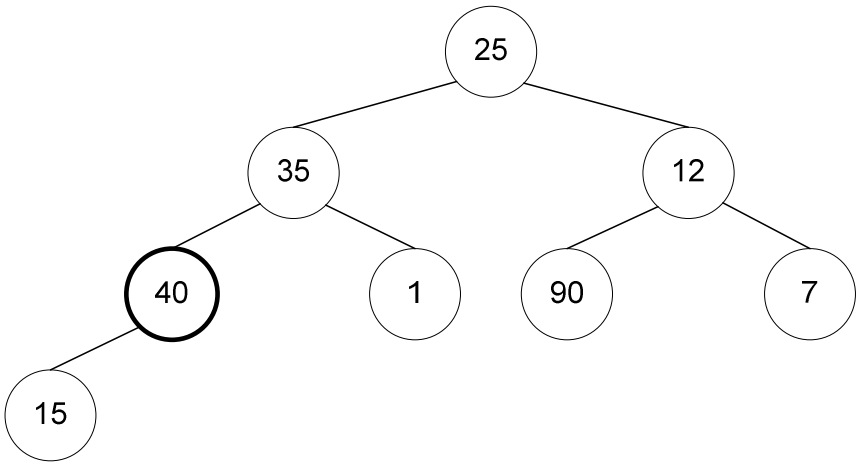
```
int rNajveci(cvor *korijen)
{
      int razina=0;
      while(korijen)
      {
            korijen=korijen->d;
            razina++;
      }
      return razina;
}
```

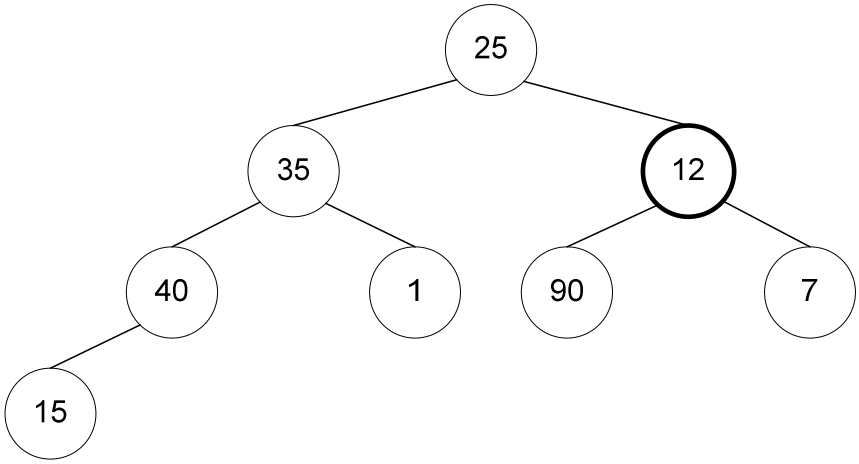
Zadatak 2

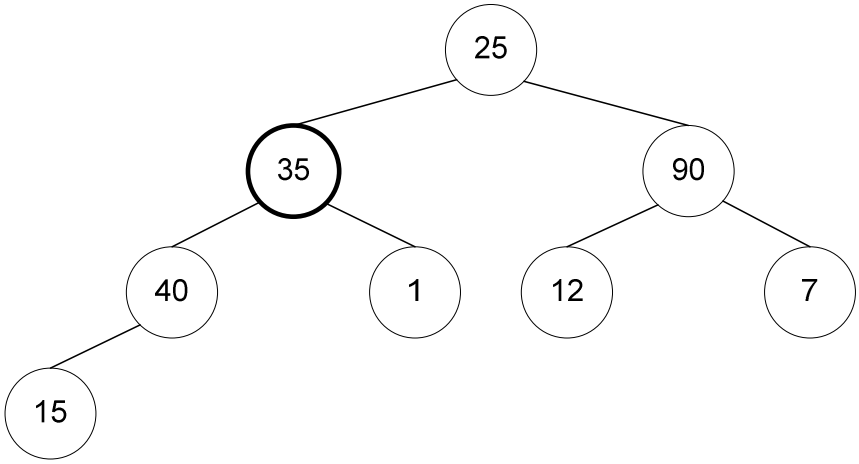
```
int rNajveciRek(cvor *korijen)
{
      if(!korijen)
            return 0;
      else
            return (1+rNajveciRek(korijen->d));
}
```

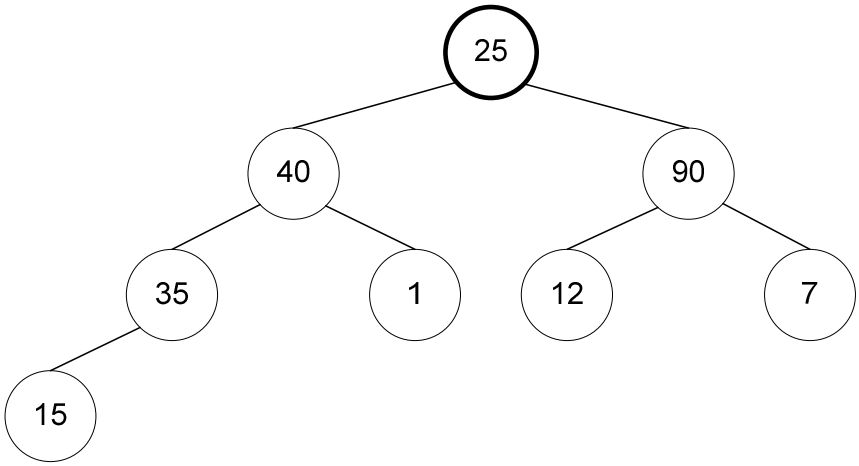
Gomila kao prioritetni red, heapsort

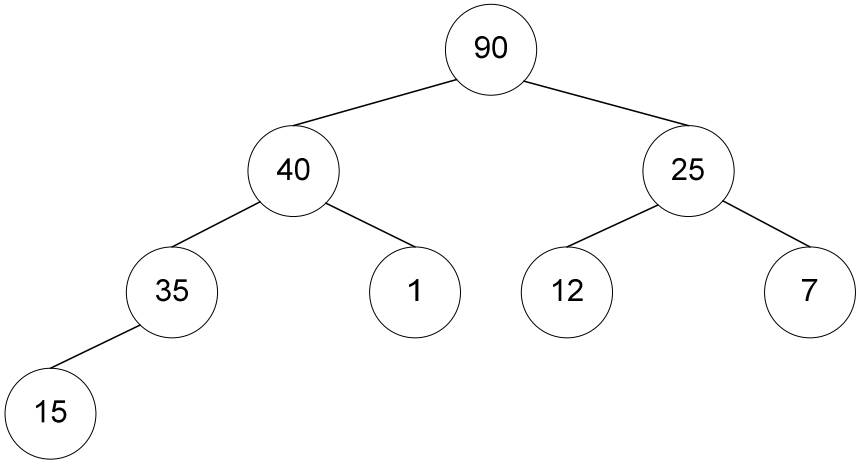
Zadatak 1



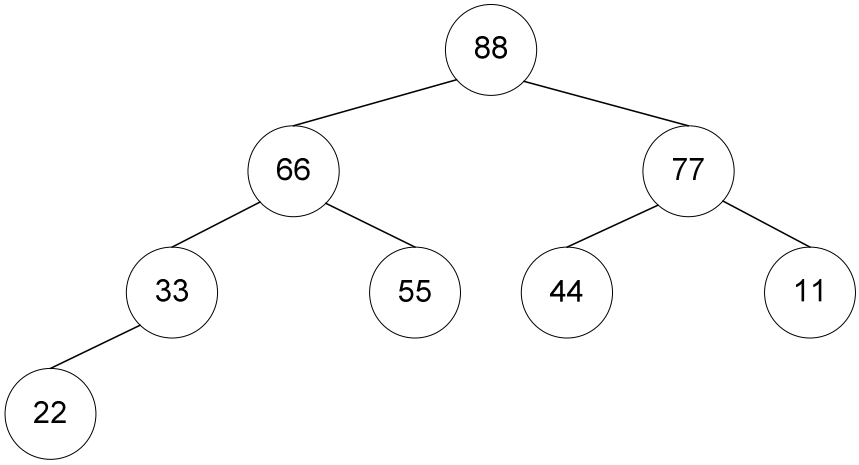




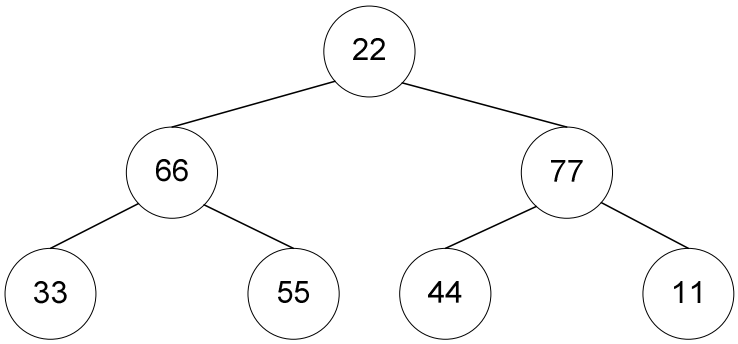




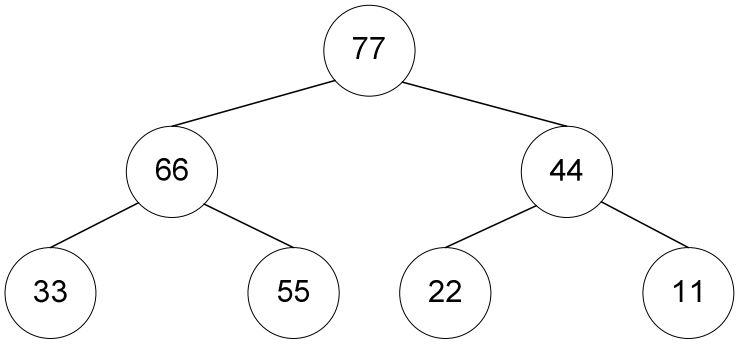
Zadatak 2



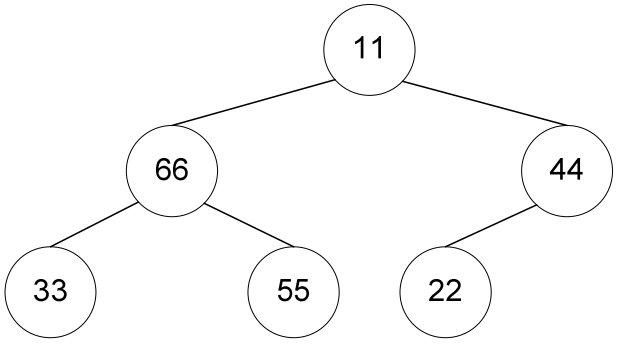
88	66	77	33	55	44	11	22



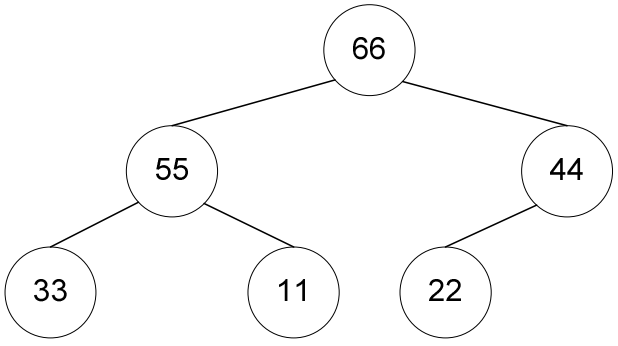
22	66	77	33	55	44	11	88



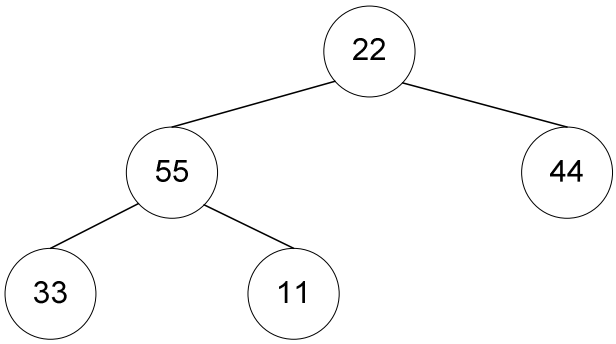
77	66	44	33	55	22	11	88



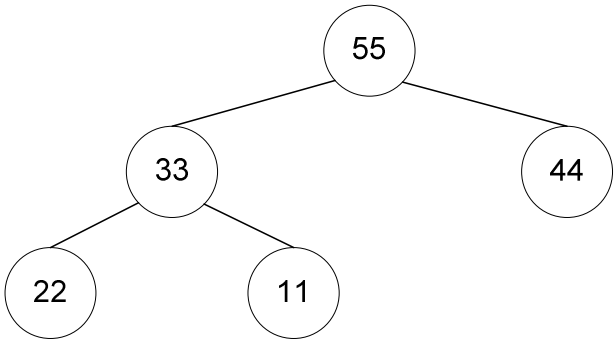
11	66	44	33	55	22	77	88



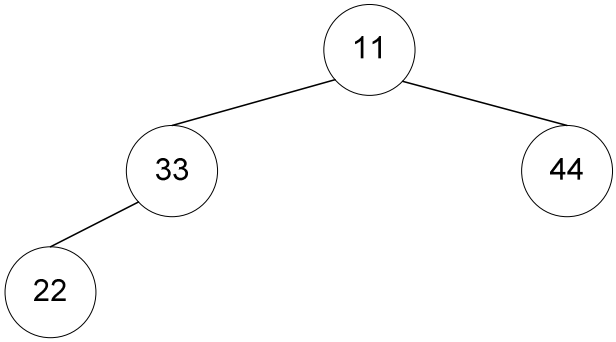
66	55	44	33	11	22	77	88



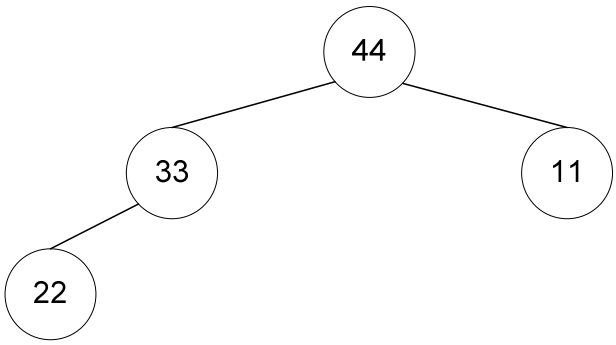
22	55	44	33	11	66	77	88



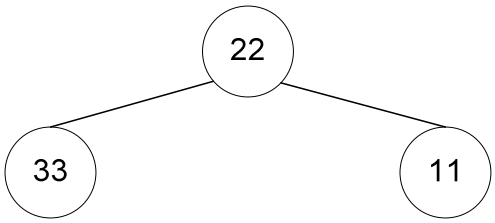
55	33	44	22	11	66	77	88



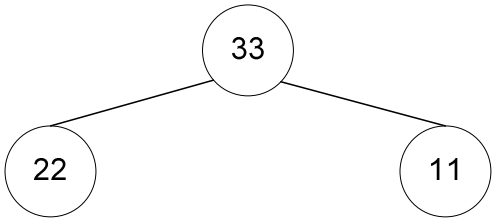
11	33	44	22	55	66	77	88



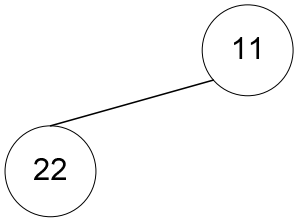
44	33	11	22	55	66	77	88



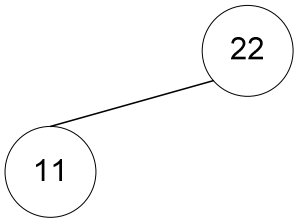
22	33	11	44	55	66	77	88



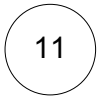
33	22	11	44	55	66	77	88



11	22	33	44	55	66	77	88



22	11	33	44	55	66	77	88



11	22	33	44	55	66	77	88

11	22	33	44	55	66	77	88