Konstrukcija i analiza algoritama 2 (ispit, smer R, smer I)

1. Odrediti izgled **AVL stabla** dobijenog izvršavanjem narednog niza operacija nad praznim stablom: (umetni,5),(umetni,1),(umetni,8),(umetni,9), (umetni,15),(umetni,7),(umetni,6),(obrisi,1),(umetni,10),(umetni,4).

2. Konstruisati (ili u pseudo-kodu ili C/C++) algoritam vremenske složenosti O(n) koji učitava sa ulaza niz realnih brojeva **a**, dimenzije **n** i pronalazi i ispisuje na izlaz neki od članova niza čiji broj pojava u nizu je veći od n/4.
Na primer u nizu: 1 2 1 2 3 1 2 1 2 rešenje je 1 ili 2

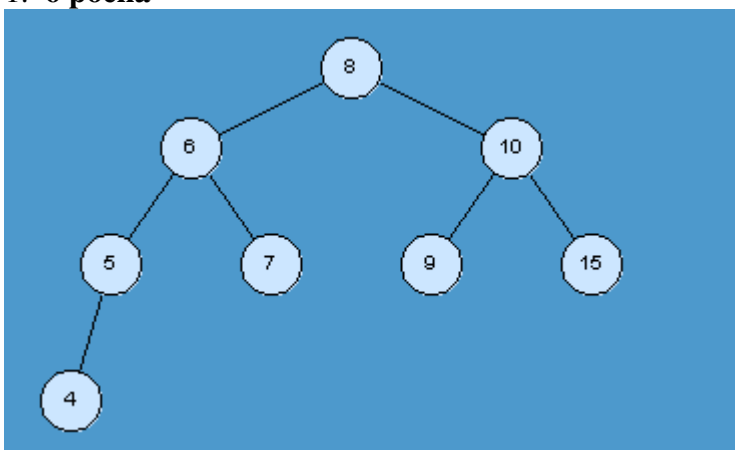
3. Konstruisati paralelni algoritam za izračunavanje vrednosti polinoma $a_0 + a_1x + \ldots + a_{n-1}x^{n-1}$ sa vremenom izvršavanja **O(log n)** na računaru sa n procesora koji funkcionise po **EREW** modelu ($n = 2^k$). Koeficijenti su smešteni u vektor duzine n u zajedničkoj memoriji. Kolika je efikasnost algoritma, i koliko se moze popraviti primenom Brentove leme?

4. Da li su sledeća tvrđenja tačna? Obrazložiti netačna tvrđenja primerom ili tačnim tvrđenjem.
a) Nedostatak skip liste je što u najgorem slučaju pretraživanje elementa može da ima vremensku složenost $O(n^2)$.
b) Model CREW ne dozvoljava da dva procesora istovremeno pristupaju istoj memorijskoj lokaciji (u smislu čitanja i pisanja).
c) U neusmerenom grafu sa 2n čvorova nije moguće naći optimalno uparivanje ako je stepen svakog čvora ne manji od n.
d) Svako optimalno uparivanje je i maksimalno uparivanje.

5. Zadat je aciklički usmeren graf G=(V,E). Konstruisati algoritam linearne složenosti koji utvrđuje da li u G postoji neki prost put koji sadrži sve čvorove grafa.

REŠENJA

1. **6 poena**



2. (**5+1 poen**)

Ideja:
Algoritam ima dve važne faze. U prvoj fazi generisšemo tri različita kandidata za rešenje zadatka, a u drugoj fazi proveravamo njihovu višestrukost.
Svaki element niza poredimo sa tri tekuća kandidata, te ako je različit od svih, njihove tekuće vrednosti višestrukosti smanjujemo za 1. Ako, pri tome, višestrukost nekog od njih padne na 0, on ispada iz spiska kandidata.
Ako trenutno ima manje od 3 kandidata, onda se tekući element niza uzima za novog kandidata sa tekućem višestrukošću 1.
Na kraju se preostali kandidati porede sa svim ostalim elementima niza da bi im se odredile tačne višestrukosti.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
#define NUM 4
int main()
{
  int a[MAX],i,n,j,k,prazno,x[NUM],p[NUM];
  printf("Unesite dimenziju niza! ");
  scanf("%d",&n);
  printf("\nUnesite clanove niza! ");
  for(i=0;i<n;i++) scanf("%d",a+i);    //T(n)=O(n)
```

```
  for(j=0;j<NUM;j++) p[j]=-1;          //T(n)=O(1)

  for(i=0;i<n;i++)                     //T(n)=O(n)*O(1)
  {
    prazno=-1;
    for(j=0;j<NUM;j++)                 //T(n)=O(1)
    {
       if ((p[j]>=0) && (x[j]==a[i])) p[j]+=3;
       else p[j]-=1;
       if(p[j]<=0) prazno=j;
    }
    if(prazno>=0)
    {x[prazno]=a[i];
     p[prazno]=3;
    }
  }
   for(j=0;j<NUM;j++) p[j]=0;    //T(n)=O(1)

 for(i=0;i<n;i++)                      //T(n)=O(n)*O(1)
   for(j=0;j<NUM;j++)                  //T(n)=O(1)
     if(a[i]==x[j]) p[j]++;
       k=(n/4)+1;
   for(j=0;j<NUM;j++)                  //T(n)=O(1)
     if(p[j]>=k)
     {printf("%d ",x[j]); return 0;}
  printf("\nNe postoji takav element\n");
   return 0;
}
```

Složenost: T(n)= O(n) + O(1) + O(n)*O(1) + O(n) + O(1) + O(n)*O(1) = O(n)

3. **(3+1+2 poena)**

Uputstvo: Neka su procesorima P*i* (1<=i<=n-1) pridružene redom memorijske lokacije v_i (1<=i<=n-1)

Koraci algoritma:

1. u prvom koraku procesor P1 čita vrednost argumenta *x* i upisuje ga u lokaciju v_1
   procesori P_i, (2<=i<=n-1) respektivno upisuju u memorijske lokacije v_i (2<=i<=n-1) vrednost 1
   potrebno vreme O(1)
2. primenom algoritma za paralelni prefiks za množenje ili broadcast algoritma na lokacije v_i (1<=i<=n-1) njihov sadržaj postaje x, x,...,x
   potrebno vreme O(log n)
3. primenom algoritma za paralelni prefiks za množenje na lokacije v_i (1<=i<=n-1) njihov sadržaj postaje $x, x^2, x^3,...,x^{n-1}$
   potrebno vreme O(log n)
4. procesor P_i (1<=i<=n-1) čita vrednost a_i, množi je sadržajem lokacije v_i i rezultat upisuje u istu tu lokaciju: time sadržaj lokacije v_i (1<=i<=n-1) postaje $a_1x, a_2x^2, a_3x^3,...,a_{n-1}x^{n-1}$
   potrebno vreme O(1)
5. primenom algoritma za paralelni prefiks za sabiranje na lokacije v_i (1<=i<=n-1) sadržaj lokacije v_{n-1} postaje $a_1x+...+a_{n-1}x^{n-1}$
   potrebno vreme O(log n)
6. Procesor P_{n-1} čita vrednost a_0, sabira je sa sadržajem lokacije v_{n-1} i rezultat upisuje u istu tu lokaciju. Time sadržaj lokacije v_{n-1} postaje $a_0+a_1x+...+a_{n-1}x^{n-1}$ što je vrednost koju je trebalo izračunati
   potrebno vreme O(1)

Ukupno potrebno vreme O(log n)

Efikasnost algoritma: $E(n,n)=\frac{n}{n \cdot \log n}=\frac{1}{\log n}$

Efikasnost se može popraviti primenom Brentove leme tako sa se koristi samo $d = n/\log n$ procesora.
Polinom se predstavlja u obliku polinoma stepena d od x^l, $l = n/d$, čiji su koeficijenti "mali" polinomi stepena l- 1, čije vrednosti pojedini procesori izračunavaju sekvencijalno. Hornerovom šemom za vreme $O(l) = O(\log n)$. Vrednost "velikog" polinoma stepena $d = n/\log n$ izračunava se na već opisani način na d procesora za vreme $O(\log d) = O(\log n)$.
Efikasnost konstruisanog algoritma je $E(n, d)=O(n/(d \log n)) = O(1)$.

4. **(1.5*4 poena)**
a) Ne. Kod skip liste u najgorem slučaju pretraživanje elementa može da ima vremensku složenost O(n).
b) Ne. Model CREW ne dozvoljava da dva procesora istovremeno pristupaju istoj memorijskoj lokaciji (samo u smislu pisanja).
c) Ne. U neusmerenom grafu sa 2n čvorova moguće je naći savršeno uparivanje ako je stepen svakog čvora ne manji od n. (po tvrđenju 6.9.1 iz udžbenika)
d) Da. Svako optimalno uparivanje je i maksimalno uparivanje.

5. **(5+1 poen)**
Put je prost, ako se svaki čvor pojavljuje u njemu samo jednom.
Hamiltonov put je prosti put u kom se svaki čvor grafa pojavljuje tačno jednom.
U ovom zadataku zahtev je da se konstruiše algoritam za pronalazak Hamiltonovog puta u grafu G.
Ideja je da se konstruiše topološki redosled čvorova grafa G, a potom se proveri da li postoji put tako što se proveri da li susedni čvorovi u topološkom redosledu jesu putno povezani. Ukoliko su povezani, onda Hamiltonov put čine topološki uređeni čvorovi.
U glavi 6.4 knjige je prikazan algoritam za topološko sortiranje acikličkog usmerenog grafa G.
Algoritam topološkog sortiranja je složenosti $O(|E| + |V|)$, tj. linearne složenosti kod grafova.

S druge strane, ako postoji Hamiltonov put, onda topološko sortiranje mora da dovede upravo do onog redosleda čvorova kojim se oni nižu u putu!!!