ISPIT IZ ALGORITAMA I STRUKTURA PODATAKA *28. januar 2009.*

1. [30] U nekoj državi postoji tačno 100 gradova od kojih su mnogi povezani direktnim **dvosmernim** putem. Gradovi su numerisani celim brojevima od 0 do 99. Predložiti **memorijski efikasnu** strukturu podataka za predstavljanje puteva u ovoj državi (neusmereni graf) koja omogućuje **konstantnu složenost** ispitivanja postojanja **direktnog** puta između gradova *i* i *j*, i napisati funkciju (ili metodu) na programskom jeziku C (ili C++) koja vrši to ispitivanje. Napisati nerekurzivnu funkciju koja, počev od grada *i*, posećuje sve gradove u državi, obilaskom odgovarajućeg grafa po širini i na standardnom izlazu štampa redne brojeve posećenih gradova u redosledu obilaska.

2. [20] U prazno B-stablo reda 3 umeću se sledeći celobrojni ključevi: 30, 18, 39, 21, 10, 6, 35, 15, 41, 19, 25, a zatim se brišu ključevi: 21, 6 i 39. Prikazati izgled stabla nakon svake izmene. Odrediti prosečan broj pristupa za uspešnu i neuspešnu pretragu stabla nakon umetanja svih ključeva i u konačnom stanju.

3. [25] Precizno opisati postupak sortiranja preko stabla selekcije. Izvesti vremensku složenost u najboljem i najgorem slučaju. Ilustrovati rad algoritma na primeru sortiranja niza 17, 99, 53, 42, 71 i 33.

4. [25] Pitanja:
   a) Napisati izraz za izračunavanje pozicije elementa niza sa kojim se vrši poređenje kod interpolacionog pretraživanja i objasniti ga.
   b) Objasniti kako se nalazi sledbenik zadatog čvora u stablu binarnog pretraživanja i ilustrovati primerom.
   c) Analitički definisati metod kvadratnog pretraživanja kod heširanja i objasniti njegove prednosti.

Ispit traje 2 sata i 15 minuta

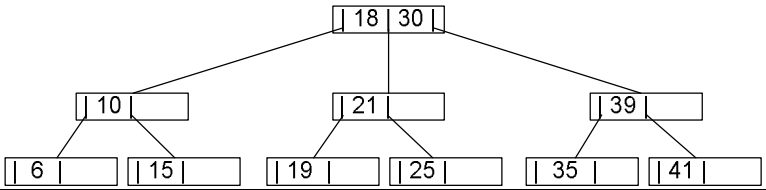
1. Predložena struktura podataka: trougaona matrica

```
class FIFO{
  struct Elem {
     int inf;
     Elem *next;
     Elem(int f) :inf(f){next=0;}
    };
   Elem *first, *last;
   void operator=(const FIFO &) { }
   FIFO(const FIFO &) { }
public:
  FIFO(){first=last=0;}
  void put(int i){
    Elem *e=new Elem(i);
    if(first==0) first=e;
    else last->next=e;
    last=e;
  }
  int get(){
    if(first==0) return -1;
    int i=first->inf;
    Elem *old = first;
    first=first->next;
    delete old;
    if(first==0) last=0;
    return i;
  }
  bool isEmpty() {return first==0;}
  ~FIFO() {
    while(first){
        Elem *old = first;
        first = first->next;
        delete old;
    }
       }
};
```

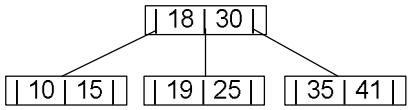
```
class BFS{

public:
  static const int MAX=100;
  static const int SIZE=MAX*(MAX-1)/2;
  BFS();
  BFS(bool *graph){this->graph=graph;}
  bool isConnected(int i, int j) const{
    if(i==j) return false;
    return graph[address(i,j)];
  }
  //////////////////nije deo resenja
  void setConnection(int i, int j, bool x){
    if(i!=j) graph[address(i,j)]=x;
  }
  bool connect(int i, int j){
                setConnection(i,j,true);}
  void disconnect(int i, int j){
               setConnection(i,j,false);}
      ///////////////////
  void traverse(int node);
  virtual ~BFS(){delete [] graph;}
protected:
  bool *graph;
  static int address(int i, int j) {
    if(i>j) return i*(i-1)/2+j;
    else return j*(j-1)/2+i;
  }
};
#include "BFS.h"
#include "FIFO.h"
#include <iostream>
using namespace std;
BFS::BFS(){
  graph=new bool[SIZE];
  for(int i=0; i<SIZE; i++) graph[i]=false;
}
void BFS::traverse(int node){
  bool visited[MAX];
  for(int i=0; i<MAX; i++) visited[i]=false;
  visited[node]=true;
  FIFO fifo;
  fifo.put(node);
  while(!fifo.isEmpty()){
    int k=fifo.get();
    cout<<k<<"  ";
    for(int i=0; i<MAX; i++)
      if(isConnected(k,i) &&
!visited[i]){
        fifo.put(i);
        visited[i]=true;
      }
  }
}
```

2. B-stablo



Pu=20/11
Pnu=3



Pu=14/8
Pnu=2