ISPIT IZ ALGORITAMA I STRUKTURA PODATAKA *17. februar 2009.*

1. [30] Složen softverski projekat sastoji se od skupa međusobno zavisnih programskih modula. Moduli su označeni jednoslovnim oznakama. Od interesa je zavisnost koja se manifestuje prilikom prevođenja projekta: ako modul A zavisi od modula B, prevođenje modula A se mora obaviti nakon prevođenja modula B. Jedan modul može da zavisi od nula ili više drugih modula.

   Predložiti efikasnu strukturu podataka za predstavljanje opisanog softverskog projekta. Precizno navesti operacije nad predloženom strukturom podataka koje su potrebne za rešavanje problema određivanja redosleda prevođenja programskih modula. Na jeziku C ili C++ napisati program koji, korišćenjem predložene strukture podataka i navedenih operacija, na standardnom izlazu ispiše oznake modula u redosledu u kom treba da budu prevedeni. Za predloženu strukturu podataka i njene operacije dati samo deklarativni deo. Koristiti fiksne ulazne podatke (nije potrebno čitati podatke).

2. [20] Data je heš tabela veličine 11 ulaza. Tabela se redom popunjava ključevima 21, 12, 23, 15, 32, 18, 46, 22, 65. Primarna heš funkcija je h(k)=k mod 11. Nacrtati izgled heš tabele posle umetanja ključeva ako se za razrešavanje kolizija koristi:
   a) dvostruko heširanje sa sekundarnom heš funkcijom g(k)=1+k mod 3.
   b) objedinjeno ulančavanje.

   Za oba slučaja odrediti prosečan broj pristupa pri uspešnom pretraživanju nakon umetanja svih ključeva. Samo za slučaj pod b) odrediti verovatnoću popunjavanja poslednjeg slobodnog ulaza u tabeli.
3. [25] Skicirati i objasniti algoritam sortiranja brojanjem. Pokazati da li je ovaj algoritam stabilan. Postupak ilustrovati po koracima na primeru sortiranja niza 5, 2, 3, 2, 5, 1, 7 i 2.
4. [25] Pitanja:
   a) Uporediti performanse i primenljivost sekvencijalnog pretraživanja sa prebacivanjem na početak i sa transpozicijom.
   b) Definisati optimalno stablo binarnog pretraživanja.
   c) Uporediti logičku i fizičku strukturu čvorova grananja i listova u B+-stablu.

Ispit traje 2 sata i 15 minuta

Rešenja:

1.

Struktura podataka: usmereni netežinski graf.
Potrebne operacije: **umetanje čvora, povezivanje dva čvora usmerenom granom, uklanjanje čvora, nalaženje čvora sa ulaznim stepenom 0**.
Delovi programa koji predstavljaju rešenje zadatka su **zasenčeni**.

```
#ifndef GRAF_H__
#define GRAF_H__
#include <list>
using namespace std;
class CvorPostojiGreska { };
class CvorNePostojiGreska { };

template<typename T>
class Graf {
  class Cvor {
    T *podatak;
    int ulazniStepen;
    list<Cvor *> listaSuseda;
  public:
    Cvor(T *pod) : podatak(pod), ulazniStepen(0) { }
    bool povezanSa(Cvor *c) const;
    static void povezi(Cvor *od, Cvor *ka) {
      od->listaSuseda.push_back(ka);
      ka->ulazniStepen++;
    }
    static void rastavi(Cvor *od, Cvor *ka);
    int dohvatiUlazniStepen() const { return ulazniStepen; }
    T *dohvatiPodatak() { return podatak; }
  };
  list<Cvor *> cvoroviGrafa;
  void ukloniCvor(Cvor *c);
  Cvor *pronadjiCvor(char ozn) const;
  void operator=(const Graf &g) { }
  Graf(const Graf &g) { }
public:
  Graf() { }
  ~Graf();

  void dodajCvor(T *pod) {
    Cvor *c = pronadjiCvor( pod->dohvatiOznaku() );
    if( c != NULL ) throw CvorPostojiGreska();
    cvoroviGrafa.push_back( new Cvor(pod) );
  }
  unsigned int brojCvorova() const {
    return cvoroviGrafa.size();
  }
  void ukloniCvor(T *cvor);
  void poveziCvorove(T *od, T *ka) {
    Cvor *c1 = pronadjiCvor( od->dohvatiOznaku() );
    Cvor *c2 = pronadjiCvor( ka->dohvatiOznaku() );
    if( c1 == NULL || c2 == NULL ) throw CvorNePostojiGreska();
    if( ! c1->povezanSa(c2) ) Cvor::povezi(c1, c2);
  }
  T *nadjiUlazniStepen0() const;
};
```

```
template<typename T>
Graf<T>::~Graf() {
  for(list<Cvor *>::iterator it=cvoroviGrafa.begin();
    it!=cvoroviGrafa.end(); it++ )   delete *it;
  cvoroviGrafa.clear();
}
template<typename T>
typename Graf<T>::Cvor* Graf<T>::pronadjiCvor( char ozn) const {
  for( list<Cvor *>::const_iterator it = cvoroviGrafa.begin();
     it != cvoroviGrafa.end(); it++ )
  if( (*it)->dohvatiPodatak()->dohvatiOznaku() == ozn )  return *it;
  return NULL;
}
template<typename T>
void Graf<T>::ukloniCvor(Cvor *c) {
  for( list<Cvor *>::iterator it = cvoroviGrafa.begin();
                              it != cvoroviGrafa.end(); it++ )
  if( *it == c )  {  cvoroviGrafa.erase(it);   break;  }
}
template<typename T>
void Graf<T>::ukloniCvor(T *pod) {
Cvor *c = pronadjiCvor(pod->dohvatiOznaku());
  if( c != NULL )  {
    for( list<Cvor *>::iterator it = cvoroviGrafa.begin();
                                it != cvoroviGrafa.end(); it++ ) {
      Cvor::rastavi(c, *it);
      Cvor::rastavi(*it, c);
    }
  ukloniCvor(c);
  }
}
template<typename T>
T *Graf<T>::nadjiUlazniStepen0()  const {
  for( list<Cvor *>::const_iterator it = cvoroviGrafa.begin();
                                    it != cvoroviGrafa.end(); it++ )
    if( (*it)->dohvatiUlazniStepen() == 0 )
      return (*it)->dohvatiPodatak();
  return NULL;
}
template<typename T>
bool Graf<T>::Cvor::povezanSa(Cvor *c)  const {
  for( list<Cvor *>::const_iterator it = listaSuseda.begin();
                                    it != listaSuseda.end(); it++ )
    if( *it == c )
      return true;
  return false;
}
template<typename T>
void Graf<T>::Cvor::rastavi(Cvor *od, Cvor *ka) {
  for( list<Cvor *>::iterator it = od->listaSuseda.begin();
                              it != od->listaSuseda.end(); it++ )
  if( *it == ka ) {
    od->listaSuseda.erase(it);
    ka->ulazniStepen--;
    break;
  }
}
#endif
```

```
#ifndef PROG_MODUL_H__
#define PROG_MODUL_H__
class ProgModul
{
      char  oznaka;
public:
      explicit ProgModul(char o) : oznaka(o) { }
      char dohvatiOznaku() const
      {
            return oznaka;
      }
};
#endif

// GLAVNI PROGRAM
#include "Graf.h"
#include "ProgModul.h"
#include <list>
#include <iostream>
using namespace std;

class KruznaZavisnostGreska { };

void main()
{
Graf<ProgModul>   graf;
ProgModul *moduli[6];

      try
      {
          for(int i = 0; i < 6; i++)
          {
                moduli[i] = new ProgModul( 'A' + i );
                graf.dodajCvor( moduli[i] );
          }

          graf.poveziCvorove( moduli[0], moduli[1] );
          graf.poveziCvorove( moduli[0], moduli[2] );
          graf.poveziCvorove( moduli[1], moduli[3] );
          graf.poveziCvorove( moduli[0], moduli[4] );
          graf.poveziCvorove( moduli[4], moduli[3] );
          graf.poveziCvorove( moduli[3], moduli[5] );
          graf.poveziCvorove( moduli[5], moduli[2] );

      while( graf.brojCvorova() > 0 )
      {
      ProgModul *p = graf.nadjiUlazniStepen0();

            if( p == NULL ) throw KruznaZavisnostGreska();
            cout << p->dohvatiOznaku() <<' ';
            graf.ukloniCvor(p);
      }
   } // try
   catch(KruznaZavisnostGreska & )
   {
     cout << "Postoji kruzna zavisnost medju modulima." <<
             "Resenje ne postoji." << endl;
   }
   catch( CvorPostojiGreska & )
   {
     cout << "Dodavanje postojeceg modula." << endl;
   }
   catch( CvorNePostojiGreska & )
   {
     cout << "Trazenje nepostojeceg modula." << endl;
   }

   for(int i = 0; i < 6; i++)  delete moduli[i];
   cout << endl;
}
```

Izlaz:

A B E D F C

2. a)

0	22
1	12
2	32
3	
4	23
5	15
6	46
7	18
8	65
9	
10	21

b)

0	22	-1
1	12	9
2	46	-1
3		-1
4	15	-1
5		-1
6	65	-1
7	18	-1
8	32	6
9	23	-1
10	21	8

Prosečan broj pristupa pri uspešnom pretraživanju u slučaju pod a) iznosi
1/9*(1+1+2+2+2+1+3+1+3) =16/9=1.778, a u slučaju pod b)
1/9*(1+1+2+1+2+1+1+1+3) = 13/9=1.444
Ulaz broj 3 će se popuniti samo u slučaju da je primarna heš funkcija vratila ostatak 3. Verovatnoća za to je 1/11. U svim ostalim slučajevima popuniće se ulaz 5, pa je verovatnoća za to 10/11=0.909.