GRAFOVI

1.PREDSTAVLJANJE GRAFOVA

1.1.Matrice povezanosti

1.2. Liste susedstva

(1.3. Pomocu Hash Set)

static void Main(string[] args)

{

Console.WriteLine("Uneti broj cvorova");

int n = int.Parse(Console.ReadLine());

//Матрица повезаности (суседства)

bool[,] matrica = new bool[n, n];//graf sa true i false

int[,] matrica1 = new int[n, n];//graf sa 0 i 1

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

matrica1[i, j] = 0;

}

}

//листа суседа(повезаности)

List<int>[] niz = new List<int>[n];

for (int cvorOd = 0; cvorOd < n; cvorOd++)

{

niz[cvorOd] = new List<int>();

}

Console.WriteLine("Uneti broj grana");

int m = int.Parse(Console.ReadLine());

for(int i = 0; i < m; i++)

{

Console.WriteLine("uneti pojedinacne grane");//npr: 1 2 (grana ide od dvora 1 do cvora 2)

string[] grana = Console.ReadLine().Split();

int grana1 = int.Parse(grana[0]);

int grana2 = int.Parse(grana[1]);

matrica[grana1 - 1, grana2 - 1] = true;//graf sa true i false

matrica1[grana1 - 1, grana2 - 1] = 1;//graf sa 0 i 1

niz[grana1].Add(grana2);

//niz[grana2].Add(grana1) ZA NEUSMERENI GRAF

}

//prikaz grafa true false

for (int i = 0; i < n; i++)

{

Console.WriteLine("");

for(int j = 0; j < n; j++)

{

Console.Write(" " + matrica[i, j] + " ");

}

}

Console.WriteLine("-------------------------------------");

//prikaz grafa sa 0 i 1

for (int i = 0; i < n; i++)

{

Console.WriteLine("");

for (int j = 0; j < n; j++)

{

Console.Write(" " + matrica1[i, j] + " ");

}

}

Console.WriteLine("-------------------------------------");

//prikaz liste

for (int i = 0; i < n; i++)

{

Console.WriteLine("");

for (int j = 0; j < niz[i].Count; j++)

{

Console.Write(" " + niz[i][j] + " ");

}

}

}

2. PRETRAGA GRAFOVA

dodaj početni čvor u kolekciju K

dok kolekcija K nije prazna

uzmi čvor C iz kolekcije K

ako C nije označen:

označi C

za svaku granu CC’

dodaj C’ u kolekciju K

Ako je kolekcija K stek radi se o *pretrazi u dubinu* a ako je red radi se o *pretrazi u širinu.*

2.1. PRETRAGA U DUBINU(DFS – Depth-First Search)

Pretraga u dubinu podrazumeva da se duž neku putanju spuštamo dokle god možemo i tek kada ne možemo da nađemo više neposećenih čvorova , onda se vraćamo nazad i razmatramo druge grane iz prethodno posećenih čvorova.

Preko rekurzije(usmereni graf)

static bool DBS(int c1, int c2, List<int>[] niz)//da li postoji put izmedju dva cvora(c1,c2)

{

if (c1 == c2) return true;

foreach(int x in niz[c1])

{

if (DBS(x, c2, niz)) return true;

}

return false;

}

Preko rekurzije(neusmereni graf)

static bool DBS1(int c1, int c2, List<int>[] niz, bool[]posecen)// da li postoji put izmedju dva cvora(c1,c2)

{

if (c1 == c2) return true;

if (posecen[c1]) return false;

posecen[c1] = true;

foreach (int x in niz[c1])

{

if (DBS1(x, c2, niz, posecen)) return true;

}

return false;

}

static bool matrica(int cvor1, int cvor2, bool[,] graf,bool[]posecen) //za matricu neusmereni

{

if (cvor1 == cvor2) return true;

if (posecen[cvor1]) return false;

posecen[cvor1] = true;

for (int i = 0; i < graf.GetLength(0); i++)

{

if (graf[cvor1, i])

{

if (matrica(i, cvor2, graf,posecen)) { return true; }

}

}

return false;

}

Preko steka

static bool DBS2(int c1, int c2, List<int>[] niz, bool[] posecen, bool postoji) {

Stack<int> stek = new Stack<int>();

postoji = false;

posecen[c1] = true;

stek.Push(c1);

while(stek.Count>0 && !postoji)

{

int p = stek.Pop();

foreach(int x in niz[p])

{

if (x == c2)

{

postoji = true;

break;

}

if (!posecen[x])

{

posecen[x] = true;

stek.Push(x);

}

}

}

return postoji;

}

2.2. PRETRAGA U ŠIRINU(BFS – Breadth – First Search )

Pretraga u širinu podrazumeva da se graf obilazi po nivoima. Prvo se obilazi polazni čvor, zatim oni čvorovi do kojih se iz polaznog može stići putem jedne grane, zatim oni čvorovi do kojih se iz početnog može stići putem 2 grane i tkd.

Preko reda

static bool BFS(int c1, int c2, List<int>[] niz, bool[] posecen, bool postoji) {

Queue<int> red = new Queue<int>();

postoji = false;

posecen[c1] = true;

red.Enqueue(c1);

while (red.Count > 0 && !postoji)

{

int p = red.Dequeue();

foreach (int x in niz[p])

{

if (x == c2)

{

postoji = true;

break;

}

if (!posecen[x])

{

posecen[x] = true;

red.Enqueue(x);

}

}

}

return postoji;

}

**Zadaci:**

**1.** Speleolozi se nalaze u ulaznoj dvorani pećine, na visini tla, čija je nadmorska visina poznata. Pećina ima ukupno n dvorana obeleženih brojevima od 0 do n − 1 (ulazna dvorana je obeležena brojem 0) i do svake od njih se može stići korišćenjem n − 1 hodnika koji ih povezuju. Ako se za svaki hodnik zna koje dve dvorane povezuje i kolika je visinska razlika između njih, napisati program koji određuje najnižu nadmorsku visinu na koju se speleolozi u pećini mogu spustiti.

struct par

{

public int c2;

public int razlika;

public par(int x, int y)

{

c2 = x;

razlika = y;

}

}

static int MinVisinaDFS(int dvorana, int visina, List<par>[] hodnici)

{

int minVisina = visina;

foreach (par h in hodnici[dvorana])

{

int v = MinVisinaDFS(h.c2, visina + h.c2, hodnici);

if (v < minVisina) minVisina = v;

}

return minVisina;

}

3. PODGRAFOVI

Pronalazenje broja nezavisnih grafova

static bool povezani(List<int>[]lista, int cvor, bool[] posecen)//proverava sa cime je cvor povezan

{

if (posecen[cvor]) return false;

posecen[cvor] = true;

foreach(int i in lista[cvor])

{

povezani(lista, i, posecen);

}

return true;

}

static int brojKomponenti(List<int>[] lista)//broji koliko imamo nezavisnih podgrafova

{

int br = 0;

bool[] posecen = new bool[lista.Length];

for(int i = 0; i < lista.Length; i++)

{

if (povezani(lista, i, posecen)) br++;

}

return br;

}

pronalaženje najvećeg broja čvorova u delu grafa/veličina najvećeg podgrafa

static int komponenta(List<int>[]lista, int cvor , bool[] posecen)//broj komponenata u okviru jednog podgrafa

{

if (posecen[cvor]) return 0;

posecen[cvor] = true;

int br = 1;

foreach(int x in lista[cvor])

{

br += komponenta(lista, x, posecen);

}

return br;

}

static int najveciBroj(List<int>[] lista)//maksimalan broj komponenata u najvecem podgrafu

{

int maks = 0;

bool[] posecen = new bool[lista.Length];

for(int i = 0; i < lista.Length; i++)

{

int x = komponenta(lista, i, posecen);

if (maks < x) { maks = x; }

}

return maks;

}

4. TOPOLOŠKO SORTIRANJE

Da bi se izgradio automobil, potrebno je uraditi niz poslova. Neki poslovi zavise od drugih (na primer, pre nego što se ugrade točkovi, potrebno je da se ugrade osovine). Napiši program koji određuje mogući redosled izvrašavanja ovih poslova u kome su sva ograničenja zadovoljena. Na standardni izlaz ispisati n brojeva poslova u nekom redosledu u kom ih je moguće izvršiti .

static void TopoloskoS(List<int>[] lista, int[] stepen)//lista

{

Queue<int> red = new Queue<int>();

for (int i = 0; i < stepen.Length; i++)

{

if (stepen[i] == 0) red.Enqueue(i);

}

while (red.Count > 0)

{

int p = red.Dequeue();

Console.WriteLine(p);

foreach (var x in lista[p])

{

stepen[x]--;

if (stepen[x] == 0) red.Enqueue(x);

}

}

}

static void TopoloskoS1(int[,] matrica, int[] stepen, int n)//matrica

{

Queue<int> red = new Queue<int>();

for (int i = 0; i < stepen.Length; i++)

{

if (stepen[i] == 0) red.Enqueue(i);

}

while (red.Count > 0)

{

int p = red.Dequeue();

Console.WriteLine(p);

for (int i = 0; i < n; i++)

{

if (matrica[p, i] == 1)

{

stepen[i]--;

if (stepen[i] == 0) red.Enqueue(i);

}

}

}

}

static void Main(string[] args)

{

int n = 6;

int m = 6;

int[] stepen = new int[n];//za listu

int[] stepen1 = new int[n];//za matricu

int[,] matrica = new int[n, n];

List<int>[] lista = new List<int>[n];

for (int i = 0; i < n; i++)

{

lista[i] = new List<int>();

}

//usmerena

for (int i = 0; i < m; i++)

{

Console.WriteLine("uneti redni brojeve 2 povezana posla ");

string[] s = Console.ReadLine().Split();

lista[int.Parse(s[0])].Add(int.Parse(s[1]));

matrica[int.Parse(s[0]), int.Parse(s[1])] = 1;

stepen[int.Parse(s[1])]++;

stepen1[int.Parse(s[1])]++;

}

Console.WriteLine("pomocu liste: ");

TopoloskoS(lista, stepen);

Console.WriteLine("pomocu matrice: ");

TopoloskoS1(matrica, stepen1, n);

Console.ReadKey();

}

}

**5. Dijskstra algoritam**

**1.** Napiši program koji omogućuje pronalaženje najkraćeg puta između dva grada i ispisuje put predjen od početnog do krajnjeg.

static int najbrziPut(double[,] graf, int n, List<int> poseceni, List<int> neposeceni, int[] put ,double[] udaljenost, int a, int b)

{

poseceni.Add(a);

neposeceni.Remove(a);

int x = a;//trenutni grad;

int indeks = 0;

while (neposeceni.Count > 0)

{

double min = int.MaxValue;

for (int i = 0; i < n; i++)

{

if ((graf[x, i] > 0) && (poseceni.Contains(i) == false))

{

if (udaljenost[x] + graf[x, i] < udaljenost[i])

{

udaljenost[i] = udaljenost[x] + graf[x, i];

put[i] = x;

}

if (udaljenost[x] + graf[x, i] < min)

{

min = udaljenost[x] + graf[x, i];

indeks = i;

}

}

}

x = indeks;

poseceni.Add(x);

neposeceni.Remove(x);

}

return x;

}

static void Main(string[] args)

{

const double INF = double.PositiveInfinity;

int m = 6;

int n = 4;

int a = 0;//pocetni grad

int b = 3;//cilj

double[,] graf = new double[n, n];

for (int i = 0; i < m; i++)

{

Console.WriteLine("uneti granu i njenu tezinu");

string[] grana = Console.ReadLine().Split();

int grana1 = int.Parse(grana[0]);

int grana2 = int.Parse(grana[1]);

double tezina = double.Parse(grana[2]);

graf[grana1, grana2] = tezina;

}

List<int> poseceni = new List<int>();//poseceni cvrovi

List<int> neposeceni = new List<int>();

int[] put = new int[n]; //poseceni gradovi od pocetnog do cilja

double[] udaljenost = new double[n];//udaljenost od pocetnog grada

for(int i = 0; i < n; i++)

{

neposeceni.Add(i);

udaljenost[i] = INF;

}

udaljenost[a] = 0;

int k = najbrziPut(graf, n, poseceni, neposeceni,put, udaljenost, a, b);

Console.WriteLine("najkrace vreme od grada {0} do grada {1} je " + udaljenost[b]+", put: ");

while (k != a)

{

Console.Write(" " + k);

k = put[k];

}

Console.Write(" " + a);

}

}

**6. Floyd Warshall Algoritam**

const int INF = int.MaxValue;

static int[,] sviNajkraciPutevi(int[,] M, int n)

{

for (int k = 0; k < n; k++)

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

if (( M[i, k] + M[k, j] < M[i,j])&&(M[i, k]!=INF)&&(M[k,j]!=INF))

{

M[i, j] = M[i, k] + M[k, j];

}

}

}

}

return M;

}

static void ispis(int[,] M, int n)

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

Console.Write(" " + M[i, j] + " ");

}

Console.WriteLine();

}

}

static void Main(string[] args)

{

int n = 4;//broj cvorova

int m = 12;//broj grana

int[,] M = new int[n, n];//pocetna matrica,graf

for (int i=0; i < n; i++)

{

for(int j = 0; j < n; j++)

{

if (i == j) M[i, j] = 0;

else M[i, j] = INF;

}

}

for(int i = 0; i < m; i++)

{

Console.WriteLine("uneti granu i tezinu");

string[] s = Console.ReadLine().Split();

M[int.Parse(s[0]), int.Parse(s[1])] = int.Parse(s[2]);

}

M = sviNajkraciPutevi(M, n);

ispis(M, n);

}

**1. Svi najkraći putevi u gustom grafu**

Na jednoj teritoriji između svaka dva grada postoji direktan put. Neki putevi su lošeg kvaliteta,pa je nekad brže stići od grada do grada ako se ide nekim od okolnih puteva. Ako je poznato vreme da se pređe svaki od direktnih puteva (ono može zavisiti i od smera u kojem se put prelazi), napiši program koji za svaki par gradova određuje koliko se vremena može uštedeti ako se ne ide direktnim putem.

Na standardni izlaz ispisati kvadratnu matricu koja sadrži uštede za svaki par gradova.

static void Main(string[] args)

{

int n = 4;//broj gradova

int[,] M = new int[n, n];

int[] putevi = {0, 190, 300, 120, 180, 0, 240, 350, 290, 430, 0, 80, 120, 170, 90, 0 };//glavni putevi

int p = -1;

for(int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

p++;

M[i, j] = putevi[p];

}

}

int[,] M1 = (int[,])M.Clone();//duplikat

for (int k = 0; k < n; k++)

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

if ((M1[i, k] + M1[k, j]) < M1[i, j])

{

M1[i, j] = (M1[i, k] + M1[k, j]);

}

}

}

}

//ustede tj. oduzimanje M1 - M

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

M1[i, j] = Math.Abs(M1[i, j] - M[i, j]);

}

}

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

Console.Write(M1[i, j] + " ");

}

Console.WriteLine();

}

}

**2. Svi najkraći putevi u retkom grafu**

Sa standardnog ulaza se učitava broj rutera n, nakon toga ukupan broj veza između rutera m, a nakon toga u narednih m redova opis svake veze: svaka linija sadrži broj polaznog rutera, broj dolaznog rutera i vreme potrebno da se pošalje poruka od polaznog do dolaznog rutera, razdvojene sa po jednim razmakom(brojevi rutera su brojevi od 0 do n − 1. Na standardni izlaz ispisati ceo broj koji predstavlja najveće skraćenje komunikacije korišćenjem posrednih rutera.

static void Main(string[] args)

{

const double INF = double.PositiveInfinity;

int n = 6;//broj rutera

int m = 7;

double[,] M = new double[n, n];

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

if (i == j) M[i, j] = 0;

else M[i, j] = INF;

}

}

for (int i = 0; i < m; i++)

{

Console.WriteLine("uneti pocetni i krajnji ruter kao i vreme potrebno da signal dodje od jednog do drugog");

string[] s = Console.ReadLine().Split();

M[int.Parse(s[0]), int.Parse(s[1])] = double.Parse(s[2]);

}

double[,] M1 = (double[,])M.Clone();//duplikat

for (int k = 0; k < n; k++)

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

if ((M1[i, k] != INF) && (M1[k, j]!= INF))

{

if ((M1[i, k] + M1[k, j]) < M1[i, j])

{

M1[i, j] = (M1[i, k] + M1[k, j]);

}

}

}

}

}

//ustede tj. oduzimanje M1 - M

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

//if (M[i, j] != INF) ne ispitujemo jer nam ne trebaju komunikacije koje nisu imale direktan put na pocetku

M1[i, j] = Math.Abs(M1[i, j] - M[i, j]);

}

}

//nalazenje najveceg skracenja vremena

double max = double.MinValue;

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

if (M1[i, j] != INF)

{

if (M1[i, j] > max) max = M1[i, j];

}

}

}

Console.WriteLine(max);

}

**6. Primov(0) i Kruskalov(1) alg za minimalno razapinjuće stablo**

**(0)**

static void Main(string[] args)

{

const double INF = double.PositiveInfinity;

int n = 9;

int m = 14;

double[,] graf = new double[n, n];

for (int i = 0; i < m; i++)

{

Console.WriteLine("uneti racunare i duzinu kabla od jednog do drugog");

string[] str = Console.ReadLine().Split();

int cvor1 = int.Parse(str[0]);

int cvor2 = int.Parse(str[1]);

int duzina = int.Parse(str[2]);

graf[cvor1, cvor2] = duzina;

graf[cvor2, cvor1] = duzina;

}

int trenutniCvor = 0;

double[] udaljenost = new double[n];//udaljenost cvorova od trenutnog

int[] parent = new int[n];//roditelji cvorova u stablu;

bool[] poseceni = new bool[n];

for (int i = 0; i < n; i++)

{

udaljenost[i] = INF;

}

udaljenost[trenutniCvor] = 0;

parent[trenutniCvor] = -1;

poseceni[trenutniCvor] = true;

int brojCvorova = 1;//broj cvorova u minimalno razapinjucem

double ukupnaDuzina = 0;

int indeks = 0;

while (brojCvorova < n)

{

for (int i = 0; i < n; i++)

{

if (graf[trenutniCvor, i] > 0 && poseceni[i] == false)

{

if ( graf[trenutniCvor, i] < udaljenost[i])

{

udaljenost[i] = graf[trenutniCvor, i];

parent[i] = trenutniCvor;

}

double min = double.MaxValue;

for (int j = 0 ; j < udaljenost.Length; j++)

{

if (udaljenost[j] < min && poseceni[j]==false)

{

min = udaljenost[j];

indeks = j;

}

}

}

}

ukupnaDuzina += udaljenost[indeks];

poseceni[indeks] = true;

trenutniCvor = indeks;

brojCvorova++;

}

Console.WriteLine(ukupnaDuzina);

}

**(1)**

struct par

{

public int cvor1;

public int cvor2;//cvor do

public int duzina;

public par(int x, int y,int z)

{

cvor1 = x;

cvor2 = y;

duzina = z;

}

}

static void Main(string[] args)

{

int n = 9;

int m = 13;

par[] grane = new par[m];

for (int i = 0; i < m; i++)

{

Console.WriteLine("uneti racunare i vreme potrebno da signal dodje od jednog do drugog");

string[] str = Console.ReadLine().Split();

int cvor1 = int.Parse(str[0]);

int cvor2 = int.Parse(str[1]);

int duzina = int.Parse(str[2]);

grane[i] = new par(cvor1, cvor2, duzina);

}

for(int i = 0; i < m - 1; i++)

{

for (int j = i + 1; j < m; j++)

{

if (grane[i].duzina > grane[j].duzina)

{

par x = grane[i];

grane[i] = grane[j];

grane[j] = x;

}

}

}

int[] grupa = new int[n];

for(int i = 0; i < n; i++)

{

grupa[i] = i;

}

double ukupnaDuz = 0.0;

int dodatoGrana = 0;

for(int i=0;i<m && dodatoGrana <= n-1; i++)

{

int c1 = grane[i].cvor1;

int c2 = grane[i].cvor2;

if (grupa[c1] != grupa[c2])

{

int g1 = grupa[c1];

for(int j = 0; j < n; j++)

{

if (grupa[j] == g1)

{

grupa[j] = grupa[c2];

}

}

ukupnaDuz += grane[i].duzina;

dodatoGrana++;

}

}

Console.WriteLine(ukupnaDuz);

}