Ministerul Educaţiei și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică şi Microelectronică

**RAPORT**

Lucrare de laborator Nr. 5

*la Matematica Discretă*

Tema: DETERMINAREA FLUXULUI MAXIM ÎNTR-O REŢEA DE TRANSPORT

A efectuat: st. gr. SI-212 Șeremet Alexandru

A verificat: lect. asist. Popovici Nadejda

Chişinău 2022

1. **SCOPUL LUCRĂRII:**

* Studierea noţiunilor de bază leagate de reţelele de transport;
* Programarea algoritmului Ford-Fulkerson pentru determinarea fluxului maxim într-o reţea de transport.

**2.NOTE DE CURS**

**Reţele de transport**

Un graf orientat *G = (X, U)* se numeşte *reţea de transport* dacă satisface următoarele condiţii:

1. există un vârf unic *a* din *X* în care nu intră nici un arc sau *d\_(a)=0*;
2. există un vârf unic *b* din *X* din care nu iese nici un arc sau *d+(a)=0*;
3. *G* este conex şi există drumuri de la *a* la *b* în *G*;
4. s-a definit o funcţie *c*: *UR* astfel încât *c(u) 0* pentru orice arc *u* din *U*.

Vârful *a* se numeşte intrarea reţelei, vârful *b* se numeşte ieşirea reţelei, iar *c(u)* este capacitatea arcului u.

O funcţie *f: UR* astfel încât *f(u) 0* pentru orice arc *u* se numeşte *flux* în reţeaua de transport *G* cu funcţia de capacitate *c*, care se notează *G = (X, U, c)*, dacă sunt îndeplinite următoarele două condiţii:

1. Condiţia de conservare a fluxului: Pentru orice vârf *x* diferit de *a* şi *b* suma fluxurilor pe arcele care intră în *x* este egală cu suma fluxurilor pe arcele care ies din *x*.
2. Condiţia de mărginire a fluxului: Există inegalitatea *f(u) c(u)* pentru orice arc *u*  *U*.

Dacă *f(u) = c(u)* arcul se numeşte *saturat*. Un *drum* se va numi *saturat* dacă va conţine cel puţin un arc saturat. Fluxul, toate drumurile căruia sunt saturate se va numi *flux complet*. Cel mai mare dintre fluxurile complete se numeşte *flux maxim*.

Pentru orice mulţime de vârfuri *A U* vom defini o *tăietură w\_(A) = {(x, y) | x* A, *y*  *A*, *(x,yU}*, adică mulţimea arcelor care intră în mulţimea *A* de vârfuri.

Prin *w+(A)* vom nota mulţimea arcelor care ies din mulţimea *A* de vârfuri.

Este justă afirmaţia: suma *f(u)* pentru *u w+(A)* este egală cu suma *f(u)* pentru arcele *uw\_(A)*. Această valoare comună se va nota *fb*.

**Algoritmul Ford-Fulkerson**

Are loc următoarea **teoremă** (Ford-Fulkerson):

Pentru orice reţea de transport *G = (X, U, c)* cu intrarea *a* şi ieşirea *b* valoarea maximă a fluxului la ieşire este egală cu capacitatea minimă a unei tăieturi, adică:

*max fb = min c(w\_(A))*.

În baza acestei teoreme a fost elaborat următorul algoritm de determinare a fluxului maxim (Ford-Fulkerson) la ieşirea *b* a unei reţele de transport *G = (X, U, c)*, unde capacitatea *c* ia numai valori întregi:

1. Se defineşte fluxul iniţial având componente nule pe fiecare arc al reţelei, adică *f(u) = 0* pentru orice arc *u U*;

2. Se determină lanţurile nesaturate de la *a* la *b* pe care fluxul poate fi mărit, prin următorul procedeu de etichetare:

a) Se marchează intrarea a cu [+];

b) Un vârf *x* fiind marcat, se va marca:

cu [+*x*] oricare vârf *y* nemarcat cu proprietatea că arcul *u = (x, y)* este nesaturat, adică *f(u)<c(u)*;

cu [-*x*] - orice vârf *y* nemarcat cu proprietatea că arcul *u = (x, y)* are un flux nenul, adică *f(u)>0*.

Dacă prin acest procedeu de marcare se etichetează ieşirea *b*, atunci fluxul *fb* obţinut la pasul curent nu este maxim. Se va considera atunci un lanţ format din vârfurile etichetate (ale căror etichete au respectiv semnele + sau -) care uneşte pe *a* cu *b* şi care poate fi găsit uşor urmărind etichetele vârfurilor sale în sensul de la *b* către *a*.

Dacă acest lanţ este *v*, să notăm cu *v+* mulţimea arcelor *(x, y)*, unde marcajul lui *y* are semnul “+”, deci care sunt orientate în sensul de la *a* către *b* şi cu *v\_* mulţimea arcelor *(x, y)*, unde marcajul lui *y* are semnul “-“, deci care sunt orientate în sensul de la *b* către *a*.

Determinăm cantitatea:

*e* *= min {min(c(u) - f(u)), min f(u)}*. *u v+*, *u v\_*

Din modul de etichetare rezultă *e > 0*.

Vom mări cu *e* fluxul pe fiecare arc *u* din *v+* şi vom micşora cu *e* fluxul pe fiecare arc *u*  *v\_*, obţinând la ieşire un flux egal cu *fb+e*. Se repetă aplicarea pasului 2 cu fluxul nou obţinut.

Dacă prin acest procedeu de etichetare nu putem marca ieşirea *b*, fluxul *fb* are o valoare maximă la ieşire, iar mulţimea arcelor care unesc vârfurile marcate cu vârfurile care nu au putut fi marcate constituie o tăietură de capacitate minimă (demonstraţi că se va ajunge în această situaţie după un număr finit de paşi).

**3. SARCINA DE BAZĂ**

1. Realizaţi procedura introducerii unei reţele de transport cu posibilităţi de verificare a corectitudinii datelor introduse;
2. În conformitate cu algoritmul Ford-Fulkerson elaboraţi procedura determinării fluxului maxim pentru valori întregi ale capacităţilor arcelor;
3. Elaboraţi programul care va permite îndeplinirea următoarelor deziderate:

* introducerea reţelei de transport în memorie;
* determinarea fluxului maxim pentru reţeaua concretă;
* extragerea datelor obţinute (fluxul maxim şi fluxul fiecărui arc) la display şi printer.

**4. CODUL PROGRAMULUI**

// C++ program for implementation of Ford Fulkerson

// algorithm

#include <iostream>

#include <limits.h>

#include <queue>

#include <string.h>

using namespace std;

// Number of vertices in given graph

int V;

/\* Returns true if there is a path from source 'from' to sink

'to' in residual graph. Also fills parent[] to store the

path \*/

bool bfs(int \*\*rGraph, int from, int to, int parent[])

{

// Create a visited array and mark all vertices as not

// visited

bool visited[V];

memset(visited, 0, sizeof(visited));

// Create a queue, enqueue source vertex and mark source

// vertex as visited

queue<int> queue;

queue.push(from);

visited[from] = true;

parent[from] = -1;

// Standard BFS Loop

while (!queue.empty())

{

int j = queue.front();

queue.pop();

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

{

if (visited[i] == false && rGraph[j][i] > 0)

{

// If we find a connection to the sink node,

// then there is no point in BFS anymore We

// just have to set its parent and can return

// true

if (i == to)

{

parent[i] = j;

return true;

}

queue.push(i);

parent[i] = j;

visited[i] = true;

}

}

}

// We didn't reach sink in BFS starting from source, so

// return false

return false;

}

// Returns the maximum flow in the given graph

int fordFulkerson(int \*\*graph, int from, int to)

{

int i, j;

// Create a residual graph and fill the residual graph

// with given capacities in the original graph as

// residual capacities in residual graph

int \*\*rGraph = new int \*[V];

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

rGraph[i] = new int[V]; // Residual graph where rGraph[j][i]

// indicates residual capacity of edge

// from j to i (if there is an edge. If

// rGraph[j][i] is 0, then there is not)

for (i = 0; i < V; i++)

for (j = 0; j < V; j++)

rGraph[i][j] = graph[i][j];

int parent[V]; // This array is filled by BFS and to

// store path

int max\_flow = 0; // There is no flow initially

// Augment the flow while there is path from source to

// sink

while (bfs(rGraph, from, to, parent))

{

// Find minimum residual capacity of the edges along

// the path filled by BFS. Or we can say find the

// maximum flow through the path found.

int path\_flow = INT\_MAX;

for (j = to; j != from; j = parent[j])

{

i = parent[j];

path\_flow = min(path\_flow, rGraph[i][j]);

}

// update residual capacities of the edges and

// reverse edges along the path

for (j = to; j != from; j = parent[j])

{

i = parent[j];

rGraph[i][j] -= path\_flow;

rGraph[j][i] += path\_flow;

}

// Add path flow to overall flow

max\_flow += path\_flow;

}

// Return the overall flow

return max\_flow;

}

void introduceEdges(int \*\*graph)

{

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

{

int j;

while (true)

{

std::cout << "Ce varf se uneste cu varful " << i << "? (introduceti -1 pentru a continua)\n";

cin >> j;

if (j < -1 || j > V)

{

std::cout << "Valoare invalida, introduceti din nou" << endl;

}

else if (j == -1)

break;

else

{

std::cout << "Ponderea: ";

cin >> graph[i][j];

}

}

}

}

// Driver program to test above functions

int main()

{

cout << "numarul de varfuri in graf: ";

cin >> V;

// Let us create a graph shown in the above example

int \*\*graph = new int \*[V];

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

{

graph[i] = new int[V];

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

graph[i][j] = 0;

}

introduceEdges(graph);

/\*graph[0][1] = 27;

graph[0][2] = 23;

graph[0][3] = 30;

graph[1][6] = 13;

graph[1][4] = 23;

graph[2][4] = 15;

graph[2][6] = 9;

graph[3][5] = 25;

graph[3][6] = 6;

graph[4][7] = 32;

graph[5][4] = 8;

graph[5][7] = 19;

graph[6][7] = 29;\*/

cout << "Care varf este sursa? ";

int source = 1;

cin >> source;

cout << "Care varf este destinatia? ";

int dest = 7;

cin >> dest;

cout << "The maximum possible flow is "

<< fordFulkerson(graph, source, dest);

return 0;

}

**5. EXECUTIA CODULUI**





**6. CONCLUZII:**

* Algoritmul Ford-Fulkerson este unul din algoritmii cei mai simpli care rezolvă problema “Debitului maxim”. Constă în identificarea succesivă a unor drumuri de creștere până în momentul în care nu mai există nici un astfel de drum.
* De obicei, este necesară și determinarea valorilor funcției f (fluxul pe un anumit arc). Pentru aceasta este suficient să se afișeze costurile arcelor speciale.