**MINISTERUL EDUCAŢIEI AL REPUBLICII MOLDOVA  
Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare Informatică şi Microelectronică  
Catedra Automatica și Tehnologii Informaționale**

RAPORT

**Lucrarea de laborator nr. 5**

**Disciplina: Matematica Discretă**

**Tema :**

**Algoritmul de determinare a grafului de acoperire**

**A efectuat: st. gr. AI-151 Brăduleav Vadim**

**A verificat: Dohotaru L.**

# LUCRAREA DE LABORATOR Nr. 5

# TEMA: ALGORITMUL DETERMINĂRII GRAFULUI DE ACOPERIRE

1. **SCOPUL LUCRĂRII:**

### Studierea algoritmului de determinare a grafului de acoperire şi elaborarea programelor care vor realiza acest algoritm.

1. **NOTE DE CURS**

## Noţiune de graf de acoperire

Fie *H* un subgraf care conţine toate vârfurile unui graf arbitrar *G*. Dacă pentru fiecare componentă de conexitate a lui *G* subgraful *H* va defini un arbore atunci *H* se va numi graf de acoperire (scheletul sau carcasă) grafului G. Este evident că graful de acoperire există pentru oricare graf: eliminând ciclurile din fiecare componentă de conexitate, adică eliminând muchiile care sunt în plus, vom ajunge la graful de acoperire.

Se numeşte graf aciclic orice graf care nu conţine cicluri. Pentru un graf arbitrar *G* cu *n* vârfuri şi *m* muchii sunt echivalente următoarele afirmaţii:

1. *G* este arbore;
2. *G* este un graf conex şi *m* = *n - 1*;
3. *G* este un graf aciclic şi *m = n - 1*;
4. oricare două vârfuri distincte (diferite) ale lui *G* sunt unite printr-un lanţ simplu care este unic;
5. *G* este un graf aciclic cu proprietatea că, dacă o pereche oarecare de vârfuri neadiacente vor fi unite cu o muchie, atunci graful obţinut va conţine exact un ciclu.

**Consecinţă**: numărul de muchii pentru un graf arbitrar G, care va fi necesar a fi eliminate spre a obţine un graf de acoperire nu depinde de ordinea eliminării lor şi este egal cu

*m(G)-n(G)+k(G)*,

unde *m(G), n(G)* şi *k(G)* sunt numărul de muchii, vârfuri şi componente conexe, respectiv.

Numărul *s(G) = m(G)-n(G)+ k(G)* se numeşte *rang* *ciclic* sau număr *ciclomatic* al grafului *G*. Numărul *r(G) = n(G)-k(G)* – *rang cociclomatic* sau *număr cociclomatic.*

Deci, *s(G)+r(G)=m(G)*.

Este adevărată următoarea afirmaţie: orice subgraf a unui graf arbitrar *G* se conţine într-un graf de acoperire a grafului *G*.

## Algoritmul de determinare a grafului de acoperire

Există mai mulţi algoritmi de determinare a grafului de acoperire. Algoritmul de mai jos nu este un algoritm-standard, ci este unul elaborat în bază algoritmului de căutare în lărgime. Esenţa algoritmului constă în aceea că folosind două fire de aşteptare în unul din care sunt înscrise (pe rând) numerele vârfurilor adiacente cu vârfurile din celălalt FA (ca şi în cazul căutării în lărgime), vor fi eliminate muchiile dintre vârfurile unui FA şi toate muchiile în afară de una dintre fiecare vârf al FA curent şi vârfurile din FA precedent. în cazul În care ambele FA vor deveni vide procedura se va termina.

Pentru a nu admite ciclarea şi ca să fim siguri că au fost prelucrate toate componentele conexe se va utiliza marcarea vârfurilor. Dacă după terminarea unui ciclu ordinar nu au mai rămas vârfuri nemarcate procedura ia sfârşit, în caz contrar în calitate de vârf iniţial se va lua oricare din vârfurile nemarcate.

**Descrierea algoritmului:**

1. Se vor declara două FA (FA1 şi FA2) vide.
2. Se va lua în calitate de vârf iniţial un vârf arbitrar al grafului.
3. Se va introduce vârful iniţial în firul de aşteptare vid FA1 şi se va marca acest vârf.
4. Se vor introduce în FA2 toate vârfurile adiacente cu vârfurile din FA1 şi se vor marca. Dacă FA2 este vid se va trece la p.7, în caz contrar - la p. 4.
5. Se vor elimina toate muchiile care leagă vârfurile din FA2.
6. Pentru toate vârfurile din FA2 vor fi eliminate toate muchiile în afară de una care leagă vârful dat cu vârfurile din FA1.
7. Se vor schimba cu numele FA1 şi FA2 (FA1 va deveni FA2 şi invers).
8. Dacă există cel puţin un vârf nemarcat se va lua în calitate de vârf iniţial oricare din acestea şi se va trece la p.1, altfel
9. STOP.

Graful obţinut este graful de acoperire.

1. **SARCINA DE BAZĂ**
2. Elaboraţi organigrama algoritmului şi programul procedurii de determinare a grafului de acoperire cu posibilităţi de pornire a procedurii din oricare vârf al grafului.
3. Utilizând procedurile de introducere a grafului în memoria CE din lucrarea Nr. 1, elaboraţi un program cu următoarele facilităţi:

* introducerea grafului care este dat sub formă de matrice de incidenţă, adiacenţă sau listă de adiacenţă;
* determinarea grafului de acoperire, pornind de la un vârf arbitrar;
* extragerea informaţiei la display sau imprimantă în oricare din formele numite.

**Imaginea și listingul aplicatiei**

**MD.css**

html {

-webkit-touch-callout: none;

-webkit-user-select: none;

user-select: none;

}

body {

height: 598px;

color: #333;

font-family: "Open Sans",X-LocaleSpecific,sans-serif;

font-size: 0.875rem;

line-height: 1.5;

margin: 0;

padding: 0;

}

#Menu {

-webkit-app-region: drag;

color: #00FF00;

width: 100%;

background-color: #00FF00;

}

#Close, #Minimize {

-webkit-app-region: no-drag;

outline: 0;

color: #00FF00;

background-color: #00FF00;

border: none;

float: right;

width: 25px;

height: 21px;

text-align: center;

}

#Close:hover, #Minimize:hover {

background-color: #FF0000;

}

#Close:active, #Minimize:active{

background-color: #00FF00;

}

table {

border-collapse: collapse;

margin: 0 auto;

}

table td {

border: 1px solid #00FF00;

padding: 0;

}

table tr:first-child td, table td:first-child {

background-color: #00FF00;

padding: 1px 3px;

font-weight: bold;

text-align: center;

font-size: 12px;

}

table input {

border: none;

height:100%;

width: 20px;

font-size: 14px;

padding: 2px;

text-align: center;

}

table input:hover {

background-color: #00FF00;

}

table input:focus {

background-color: #00FF00;

outline: none;

}

**Index.html**

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<link rel="stylesheet" href="MD.css">

<title>Brăduleac Vadim -> Matematica discreta</title>

</head>

<body ondragstart="return false;" ondrop="return false;">

<center>

<br>

<table id="R">

<tbody>

<tr><td></td><td></td></tr>

<tr><td>Fluxum-maxim:</td><td></td></tr>

<tr><td>Graful de acoperire:<br>V1</td><td></td></tr>

<tr><td>Graful de acoperire:<br>V2</td><td></td></tr>

</tbody>

</table>

<br>

<center>

<input id="n" value="" style="width:30px" />

<button onclick="CTable(parseInt(n.value))">Genereaza</button>

<input id="s" value="" style="width:30px" />

<input id="f" value="" style="width:30px" />

<button onclick="Calc(parseInt(s.value),parseInt(f.value))">Calculeaza</button>

</center>

<br>

<table id="T"></table>

<br>

</center>

</body>

</html>

<script>

var FlowNetwork = function() {

var Edge = function(source, sink, capacity) {

this.source = source;

this.sink = sink;

this.capacity = capacity;

this.reverseEdge = null;

this.flow = 0;

};

this.edges = {};

this.findEdgeInPath = function(path, edge, residual) {

for(var p=0;p<path.length;p++)

if(path[p][0] == edge && path[p][1] == residual)

return true;

return false;

};

this.addEdge = function(source, sink, capacity) {

if(source == sink) return;

var edge = new Edge(source, sink, capacity);

var reverseEdge = new Edge(sink, source, 0);

edge.reverseEdge= reverseEdge;

reverseEdge.reverseEdge = edge;

if(this.edges[source] === undefined) this.edges[source] = [];

if(this.edges[sink] === undefined) this.edges[sink] = [];

this.edges[source].push(edge);

this.edges[sink].push(reverseEdge);

};

this.findPath = function(source, sink, path) {

if(source == sink) return path;

for(var i=0;i<this.edges[source].length;i++) {

var edge = this.edges[source][i];

var residual = edge.capacity - edge.flow;

if(residual > 0 && !this.findEdgeInPath(path, edge, residual)) {

var tpath = path.slice(0);

tpath.push([edge, residual]);

var result = this.findPath(edge.sink, sink, tpath);

if(result != null) return result;

}

}

return null;

};

this.maxFlow = function(source, sink) {

var path = this.findPath(source, sink, []);

while(path != null) {

var flow = 999999;

// Find the minimum flow

for(var i=0;i<path.length;i++)

if(path[i][1] < flow) flow = path[i][1];

// Apply the flow to the edge and the reverse edge

for(var i=0;i<path.length;i++) {

path[i][0].flow += flow;

path[i][0].reverseEdge.flow -= flow;

}

path = this.findPath(source, sink, []);

}

var sum = 0;

for(var i=0;i<this.edges[source].length;i++)

sum += this.edges[source][i].flow;

return sum;

};

};

var MD = (function(){

var M = [];

return {

MA: function(a){

M = []

for (var i = 0, n = (a||[]).length; i < n; i++){

M[i+1]=[]

for (var j = 0; j < n; j++) M[i+1][j+1]=parseInt(a[i][j])||0

}

},

PL: function(s){

var L = [], S = [], R = [],ll = []

for(i=1;i<M.length;i++){R[i] = [];for(j=1;j<M[i].length;j++)R[i][j] = 0}

PL(s)

for(var i=1,n=M.length;n>i;i++) PL(i)

for(i=1,h = '';i<R.length;i++){

for(j=1;j<R[i].length;j++) h += " | " + R[i][j] + " | "

h += '<br>'

}

return h

//return L

function PL(l){

var A\_1 = [], A\_2 = []

if (!S[l]) A\_1 = [l];

while(1){

for (var i = 0, n = A\_1.length; i < n; i++) {

if (!S[A\_1[i]]) { S[A\_1[i]] = 1; L.push(A\_1[i])

for (var j = 1, k = M[A\_1[i]].length; j < k; j++) {

if (M[A\_1[i]][j]){

if (!S[j] && !ll[j]){

ll[j] = 1;

R[A\_1[i]][j] = 1

}

A\_2.push(j)

}

}

}

}

if (A\_2.length == 0) break;

A\_1 = A\_2; A\_2 = []

}

}

return L;

},

PA: function(s){

var S=[],L=[], R = []

for(i=1;i<M.length;i++){R[i-1] = []; for(j=1;j<M[i].length;j++) R[i-1][j-1] = 0 }

PA(s); for(var i=1,n=M.length;n>i;i++) PA(i);

for(i=0,h = '';i<R.length;i++){

for(j=0;j<R[i].length;j++) h += " | " + R[i][j] + " | "

h += '<br>'

}

return h

function PA(i) {

if (!S[i]) {

S[i] = 1; L.push(i);

for(var j=1,n=(M[i]||[]).length;n>j;j++)

if(M[i][j]){

if(!S[j])R[i-1][j-1]=1

PA(j)

}

}

}

}

}

})();

function CTable(x){

for (var i = 0, HTML = ""; i <= x; i++) {

HTML += "<tr>"

for (var j = 0; j <= x; j++) {

HTML += "<td>"+(i&&j ? "<input name="+(i-1)+"|"+(j-1)+" onclick='this.select()' value=0 />" : "x<sub>" + (!i?j:i) + "</sub>")+"</td>"

}

HTML += "</tr>"

}

T.innerHTML = HTML;

}

function Calc(s, f){

var fn = new FlowNetwork(), data = [];

for(i=1;i<T.rows.length;i++){

data[i-1] = [];

for(j=1;j<T.rows[i].cells.length;j++){

data[i-1][j-1] = parseInt(T.rows[i].cells[j].children[0].value)||0;

if(data[i-1][j-1] > 0) fn.addEdge(i+'',j+'', data[i-1][j-1]);

}

} MD.MA(data);

R.rows[1].cells[1].textContent = fn.maxFlow(s+'',f+'');

R.rows[2].cells[1].innerHTML = MD.PL(s);

R.rows[3].cells[1].innerHTML = MD.PA(s);

}

</script>

**Concluzie:** În urma efectuării lucrarii de laborator am studiat algoritmul de determinare a grafului de acoperire. Cu ajutorul acestui algoritm am elaborat programa care îl realizează. Am elaborat organigrama algoritmului şi programul procedurii de determinare a grafului de acoperire cu posibilităţi de pornire a procedurii din oricare vârf al grafului.