**Міністерство освіти і науки України  
Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
Факультет інформатики та обчислювальної техніки  
Кафедра обчислювальної техніки**

Лабораторна робота №3

з дисципліни  
«Дискретні структури»

на тему «Характеристики графів»

Виконав: Перевірила:

студент групи ІП-95 Сергієнко А. А.

Грибинюк Олександр Сергійович  
номер залікової книжки: 9504

Київ 2020

**Завдання на лабораторну роботу**

1. Представити напрямлений граф із заданими параметрами так само, як у лабораторній роботі No1.

Відміна: матриця А напрямленого графа за варіантом формується за командами Scilab:

rand("seed", п1п2п3п4);  
T = rand(n,n) + rand(n,n);  
A = floor((1.0 - п3\*0.005 - п4\*0.005 - 0.27)\*T)  
2. Створити скрипт для Scilab для обчислення наступних результатів:

* 1)  матриця суміжності;
* 2)  півстепені вузлів;
* 3)  всі шляхи довжини 2 і 3;
* 4)  матриця досяжності;
* 5)  компоненти сильної зв’язності;
* 6)  матриця зв’язності;
* 7)  граф конденсації.

Шляхи довжиною 2 і 3 слід шукати за матрицями А і А , відповідно. Для

спрощення пошуку маршрутів, перед обчисленням цих матриць у матриці А слід заповнити нулями головну діагональ. Матрицю досяжності та компоненти сильної зв’язності слід шукати за допомогою операції транзитивного замикання.

Варіант 9504:  
*п = 14*  
розміщення вершин: трикутником при n = 4

**Текст програми для (JavaScript Canvas)**

**//JGraph.js**

const Graph = function(ctx, adjM, directed = false, radius = 10) {

this.directed = directed;

this.adjM = adjM;

this.vertices = [];

this.ctx = ctx;

this.radius = radius;

};

Graph.prototype.connect = function(v1, v2, directed = false, double = false) {

this.ctx.beginPath()

if (v1 === v2) {

this.ctx.arc(v1.x, v1.y + 28, this.radius - 13, 0, 2 \* Math.PI);

this.ctx.stroke();

if (directed) {

v1.out++;

v1.in++;

} else {

v1.deg += 2;

}

return;

}

const xr = v2.x - v1.x;

const yr = v2.y - v1.y;

const k = yr / xr;

const xinter = Math.sqrt((this.radius \*\* 2)/(1 + k \*\* 2));

const yinter = k \* xinter;

if (directed) {

v1.out++;

v2.in++;

if (v1.x === v2.x) {

if (v2.y > v1.y) {

canvas\_arrow(this.ctx, v1.x, v1.y + this.radius, v2.x, v2.y - this.radius);

} else {

canvas\_arrow(this.ctx, v1.x, v1.y - this.radius, v2.x, v2.y + this.radius);

}

} else {

if (v2.x < v1.x) {

canvas\_arrow(this.ctx, v1.x - xinter, v1.y - yinter, v2.x + xinter, v2.y + yinter);

} else {

canvas\_arrow(this.ctx, v1.x + xinter, v1.y + yinter, v2.x - xinter, v2.y - yinter);

}

}

} else {

v1.deg++;

v2.deg++;

if (v1.x === v2.x) {

if (v2.y > v1.y) {

this.ctx.moveTo(v1.x, v1.y + this.radius);

this.ctx.lineTo(v2.x, v2.y - this.radius);

} else {

this.ctx.moveTo(v1.x, v1.y - this.radius);

this.ctx.lineTo(v2.x, v2.y + this.radius);

}

} else {

if (v2.x < v1.x) {

this.ctx.moveTo(v1.x - xinter, v1.y - yinter);

this.ctx.lineTo(v2.x + xinter, v2.y + yinter);

} else {

this.ctx.moveTo(v1.x + xinter, v1.y + yinter);

this.ctx.lineTo(v2.x - xinter, v2.y - yinter);

}

}

}

if (double) {

v1.in++;

v2.out++;

const d = Math.sqrt((v2.x - v1.x) \*\* 2 + (v2.y - v1.y) \*\* 2);

let k = (v2.y - v1.y) / (v2.x - v1.x);

let phi = Math.atan(k);

this.ctx.save();

if (v2.x >= v1.x) {

this.ctx.translate(v1.x, v1.y);

this.ctx.rotate(phi);

canvas\_arrow(this.ctx, d / 2, d / 8, this.radius, 0);

this.ctx.moveTo(d / 2, d / 8);

this.ctx.lineTo(d - this.radius, 0);

} else {

this.ctx.translate(v2.x, v2.y);

this.ctx.rotate(phi);

canvas\_arrow(this.ctx, d / 2, d / 8, d - this.radius, 0);

this.ctx.moveTo(d / 2, d / 8);

this.ctx.lineTo(this.radius, 0);

}

this.ctx.restore();

}

this.ctx.stroke();

};

Graph.prototype.vertex = function(x, y, n) {

const v = {

x: x,

y: y,

num: n,

deg: 0,

in: 0,

out: 0,

};

this.vertices[n - 1] = v;

};

Graph.prototype.draw = function() {

this.ctx.textAlign = 'center';

this.ctx.textBaseline = 'middle';

for (const vert of this.vertices) {

this.ctx.beginPath();

this.ctx.arc(vert.x, vert.y, this.radius, 0, 2 \* Math.PI);

this.ctx.fillText(vert.num, vert.x, vert.y);

this.ctx.stroke();

}

if (!this.directed){

for (const i in this.adjM) {

for (const j in this.adjM[i]) {

if (this.adjM[i][j] === 1 && i > j) {

const v1 = this.vertices[i];

const v2 = this.vertices[j];

this.connect(v1, v2);

} else if (i === j && this.adjM[i][j] === 1) {

const v1 = this.vertices[i];

this.connect(v1, v1);

}

}

}

} else {

for (const i in this.adjM) {

for (const j in this.adjM[i]) {

if (this.adjM[i][j] === 1 && this.adjM[j][i] !== 1) {

const v1 = this.vertices[i];

const v2 = this.vertices[j];

this.connect(v1, v2, true);

}

else if (this.adjM[i][j] === 1 && this.adjM[j][i] === 1 && i >= j) {

const v1 = this.vertices[i];

const v2 = this.vertices[j];

this.connect(v1, v2, true, true);

}

}

}

}

};

Graph.prototype.circle = function(radius, xc, yc) {

for (let i = 1; i <= this.adjM.length; i++) {

const delta = 2 \* Math.PI / this.adjM.length;

const x = radius \* Math.cos(i \* delta) + xc;

const y = radius \* Math.sin(i \* delta) + yc;

this.vertex(x, y, i);

}

this.draw();

};

Graph.prototype.triangle = function() {

for (let i = 1; i <= 10; i++) {

let x = 100 \* i;

let y = 50;

if (i == 6) {

x = 175;

y= 170;

};

if( i == 7) {

x = 200;

y = 300;

};

if( i == 8) {

x = 300;

y = 500;

};

if (i == 9) {

x = 420;

y = 170;

};

if (i == 10) {

x = 400;

y = 300;

};

this.vertex(x, y, i);

}

this.draw();

};

Graph.prototype.info = function(container) {

let result = '';

let data = '';

let homo = true;

let Degree = this.directed ? this.vertices[0].in + this.vertices[0].out : this.vertices[0].deg;

for (let vert of this.vertices) {

if (this.directed && (vert.in + vert.out !== Degree)) homo = false;

if (!this.directed && vert.deg !== Degree) homo = false;

if (!this.directed) {

data = `Vertex: ${vert.num}, deg: ${vert.deg}<br>`;

} else {

data = `Vertex: ${vert.num}, In: ${vert.in}, Out: ${vert.out}<br>`;

}

result += data;

}

for (let vert of this.vertices) {

if (vert.deg === 0 && (vert.in === 0 && vert.out === 0)) {

result += `<font color="red">Isolated vertex ${vert.num}</font><br>`;

} else if (vert.deg === 1) {

result += `<font color="green">Leaf vertex ${vert.num}</font><br>`;

} else if (vert.in === 1 && vert.out === 0) {

result += `<font color="green">Leaf vertex ${vert.num}</font><br>`;

} else if (vert.out === 1 && vert.in === 0) {

result += `<font color="green">Leaf vertex ${vert.num}</font><br>`;

}

}

if (homo) result += 'Graph is <strong>homogeneous</strong> ' + Degree;

container.innerHTML = result;

};

Graph.prototype.trans\_info = function(container) {

let routes2 = 'Routes with length 2<br>';

let routes3 = 'Routes with length 3<br>';

const deg2 = closure(this.adjM, this.adjM);

for (let from = 0; from < deg2.length; from++) {

for (let to = 0; to < deg2.length; to++) {

if (deg2[from][to] !== 0) {

this.adjM[from].forEach((val1, index1) => {

if (val1 !== 0 && this.adjM[index1][to] !== 0) {

routes2 += `v${from + 1} v${index1 + 1} v${to + 1}<br>`;

for (let index2 = 0; index2 < this.adjM.length; index2++) {

if (this.adjM[to][index2] !== 0) routes3 += `v${from + 1} v${index1 + 1} v${to + 1} v${index2 + 1}<br>`;

}

}

})

}

}

}

container.innerHTML += routes2;

container.innerHTML += routes3;

};

const canvas\_arrow = (context, fromx, fromy, tox, toy) => {

const headlen = 10; // length of head in pixels

const dx = tox - fromx;

const dy = toy - fromy;

const angle = Math.atan2(dy, dx);

context.moveTo(fromx, fromy);

context.lineTo(tox, toy);

context.lineTo(tox - headlen \* Math.cos(angle - Math.PI / 6), toy - headlen \* Math.sin(angle - Math.PI / 6));

context.moveTo(tox, toy);

context.lineTo(tox - headlen \* Math.cos(angle + Math.PI / 6), toy - headlen \* Math.sin(angle + Math.PI / 6));

};

const closure = (matrix1, matrix2) => {

let res = matrix1.map(arr => [...arr]);

res.forEach(val => val.fill(0));

for (const i in matrix1) {

for (const j in matrix1) {

if (matrix1[i][j] !== 0) {

for (const vert in matrix2[j]) {

if (matrix2[j][vert] !== 0) res[i][vert] = 1;//matrix2[j][vert];

}

}

}

}

return res;

};

const arraysEqual = (a1, a2) => {

return JSON.stringify(a1) === JSON.stringify(a2);

};

const trans\_closure = m1 => {

let m2 = m1;

let temp = [];

while (!arraysEqual(m2, temp)) {

temp = m2;

m2 = closure(m1, m2);

for (let i = 0; i < m2.length; i++)

for (let j = 0; j < m2.length; j++) {

if (temp[i][j] === 1 && m2[i][j] !== 1) m2[i][j] = 1;

if (m2[i][i] !== 1) m2[i][i] = 1;

}

}

return m2;

};

const transpose = m => {

let res = m.map(arr => [...arr]);

let temp = 0;

for (let i = 0; i < m.length; i++)

for (let j = 0; j < m.length; j++) {

if (i > j) {

temp = res[j][i];

res[j][i] = res[i][j];

res[i][j] = temp;

}

}

return res;

};

const product\_m = (m1, m2) => {

let res = m1.map(arr => [...arr]);

for (let i = 0; i < m1.length; i++)

for (let j = 0; j < m1.length; j++)

res[i][j] = m1[i][j] && m2[i][j];

return res;

};

//Lab2.html

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8"/>

<title>Graph</title>

<script type="text/javascript" src="JGraph.js" charset="utf-8"></script>

<script type="text/javascript" src="lab2.js" charset="utf-8"></script>

<style type="text/css">

canvas { border: 1px solid black; }

body { text-align: center; }

div { text-align: right; border: 1px solid black; position: center; padding: 20px; text-align: center; margin-left: 20px;}

</style>

</head>

<body onload="draw();">

<h1>Directed graph</h1>

<div id="graph1\_info"></div>

<canvas id="graph1" width="550" height="550"></canvas>

<h1>Undirected graph</h1>

<div id="graph2\_info"></div>

<canvas id="graph2" width="550" height="550"></canvas>

</body>

</html>

**//lab3.js**

**const matrix = [**

**[0, 0, 0, 0, 0, 0, 1, 0, 0, 1],**

**[0, 0, 1, 0, 0, 0, 0, 0, 0, 0],**

**[0, 0, 0, 0, 0, 0, 0, 0, 1, 0],**

**[0, 0, 0, 0, 1, 1, 0, 0, 1, 0],**

**[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],**

**[0, 0, 0, 0, 0, 0, 1, 0, 0, 0],**

**[0, 0, 0, 0, 0, 0, 1, 0, 1, 0],**

**[0, 0, 1, 0, 0, 0, 0, 0, 0, 0],**

**[0, 0, 0, 1, 1, 0, 0, 0, 1, 0],**

**[0, 0, 0, 0, 1, 0, 0, 1, 0, 0],**

**];**

**const test = [**

**[0, 0, 0, 1, 0, 0],**

**[0, 0, 0, 0, 1, 0],**

**[1, 0, 0, 0, 0, 0],**

**[0, 0, 1, 0, 0, 0],**

**[0, 1, 0, 1, 0, 0],**

**[0, 0, 1, 1, 0, 0],**

**];**

**const draw = () => {**

**const canvas1 = document.getElementById('graph1');**

**const canvas2 = document.getElementById('condensed');**

**if (canvas1.getContext && canvas2.getContext) {**

**const container1 = document.getElementById('graph1\_info');**

**const container2 = document.getElementById('graph1\_trans\_info');**

**const ctx = canvas1.getContext('2d');**

**const ctx2 = canvas2.getContext('2d');**

**const graph1 = new Graph(ctx, matrix, true, 20);**

**graph1.triangle();**

**graph1.info(container1);**

**graph1.trans\_info(container2);**

**const reachM = trans\_closure(test);**

**const strongM = product\_m(reachM, transpose(reachM));**

**let pool = [];**

**const check = new Array(strongM.length).fill(0);**

**let found = false;**

**strongM.forEach((a1, index1) => {**

**strongM.forEach((a2, index2) => {**

**if (arraysEqual(a1, a2) && index1 > index2) {**

**found = false;**

**for (let i = 0; i < pool.length; i++) {**

**pool[i].forEach(val => {**

**if (index1 === val) {**

**pool[i].push(index2);**

**check[index2] = 1;**

**found = true;**

**} else if (index2 === val) {**

**pool[i].push(index1);**

**check[index1] = 1;**

**found = true;**

**}**

**})**

**}**

**if (!found) pool.push([index1, index2]); check[index2] = 1; check[index1] = 1;**

**}**

**})**

**});**

**check.forEach((val, index) => {if (val === 0) pool.push([index])});**

**pool = pool.map(val => [...new Set(val)]);**

**const condenseM = new Array(pool.length).fill(0);**

**for (let i = 0; i < condenseM.length; i++) condenseM[i] = new Array(pool.length).fill(0);**

**pool.forEach((k1, index1) => {**

**pool.forEach((k2, index2) => {**

**if (index1 > index2) {**

**k1.forEach(i => {**

**k2.forEach(j => {**

**if (test[i][j] === 1) condenseM[index1][index2] = 1;**

**if (test[j][i] === 1) condenseM[index2][index1] = 1;**

**})**

**})**

**}**

**})**

**});**

**const condensed\_graph = new Graph(ctx2, condenseM, true, 20);**

**condensed\_graph.circle(100, 200, 200);**

**const component\_container = document.getElementById('component\_info');**

**let component\_info = 'Components:<br>';**

**pool.forEach((val, index) => component\_info += `K${index + 1}: ${val.map(val => ++val)}<br>`);**

**component\_container.innerHTML = component\_info;**

**console.log('Adjacency matrix');**

**console.table(matrix);**

**console.log('Reachability matrix');**

**console.table(reachM);**

**console.log('Strong reachability matrix');**

**console.table(strongM);**

**}**

**};**

**Згенеровані матриці суміжності**

Матриця суміжності напрямленого графу:

[0, 0, 0, 0, 0, 0, 1, 0, 0, 1],

[0, 0, 1, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0, 1, 0],

[0, 0, 0, 0, 1, 1, 0, 0, 1, 0],

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 1, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 1, 0, 1, 0],

[0, 0, 1, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 1, 1, 0, 0, 0, 1, 0],

[0, 0, 0, 0, 1, 0, 0, 1, 0, 0],

Тестова матриця

**Результати виконання програми**



