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 №4

з дисципліни
«Алгоритми і структури даних»

Виконав

студента групи ІМ-42
Ватолкін Михайло Андрійович
номер у списку групи: 8

Перевірила:

Сергієнко А. М.

Київ 2024

Завдання:

1.Представити напрямлений та ненапрямлений графи із заданими параметрами так само, як у лабораторній роботі №3.

Відмінність: коефіцієнт $k = 1.0 - n_3 \quad 0.01 - n_4 \quad 0.01 - 0.3.$

Отже, матриця суміжності Adir напрямленого графа за варіантом формується таким чином:

1) встановлюється параметр (seed) генератора випадкових чисел, рівне номеру варіанту n1n2n3n4;
2) матриця розміром n n заповнюється згенерованими випадковими числами в діапазоні [0, 2.0);
3) обчислюється коефіцієнт k 1.0 n3 0.01 n4 0.01 0.3, кожен елемент матриці множиться на коефіцієнт k;
4) елементи матриці округлюються: 0 — якщо елемент менший за 1.0, 1 — якщо елемент більший або дорівнює 1.0.

2. Обчислити:

1) степені вершин напрямленого і ненапрямленого графів;
2) напівстепені виходу та заходу напрямленого графа;
3) чи є граф однорідним (регулярним), і якщо так, вказати степінь однорідності графа;
4) перелік висячих та ізольованих вершин.

Результати вивести у графічне вікно, консоль або файл.

3. ***Змінити матрицю*** A_{dir}, коефіцієнт $k = 1.0 - n_3 \quad 0.005 - n_4 \quad 0.005 - 0.27.$

4. Для нового орграфа обчислити:

1) півстепені вершин;

2) всі шляхи довжини 2 і 3;

3) матрицю досяжності;

4) матрицю сильної зв’язності;

5) перелік компонент сильної зв’язності;

6) граф конденсації.

Результати вивести у графічне вікно, в консоль або файл.

Шляхи довжиною 2 і 3 слід шукати за матрицями A^2 і A^3 відповідно. Як результат вивести перелік шляхів, включно з усіма проміжними вершинами, через які проходить шлях.

Матрицю досяжності та компоненти сильної зв’язності слід шукати за

допомогою операції транзитивного замикання. У переліку компонент

слід вказати, які вершини належать до кожної компоненти.

Граф конденсації вивести у графічне вікно.

Номер варіанту: 4208

Текст програми:

HTML файл:

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8"/>
    <title>Графічне представлення графів</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            padding: 20px;
        }

        canvas {
            border: 1px solid black;
            background: #fff;
            display: inline-block;
            margin: 20px auto;
            width: 700px;
            height: 700px;
        }


        button {
            margin-right: 10px;
            padding: 10px 15px;
        }
    </style>
</head>
<body>
<h1>Варіант 4208</h1>
<div id="buttons">
    <button onclick="drawDirected1()">Напрямлений граф(k1)</button>
    <button onclick="drawUndirected1()">Ненапрямлений граф(k1)</button>
    <button onclick="drawDirected2()">Напрямлений граф(k2)</button>
    <button onclick="drawCondense()">Граф конденсації (k2)</button>
</div>

<canvas id="canvas" width="600" height="600"></canvas>

<script src="script.js"></script>
</body>
</html>
```

JS скрипт:

```
//config
const canvas = document.getElementById('canvas');
const ctx = canvas.getContext('2d');
const seed = 4208;
const n3 = 0;
const n4 = 8;
const n = 10;
const k1 = 1.0 - n3 * 0.01 - n4 * 0.01 - 0.3;
const k2 = 1.0 - n3 * 0.005 - n4 * 0.005 - 0.27;
const w = canvas.width;
const h = canvas.height;

//matrix
function genRandNum(seed) {
    const RANDOM_NUMBER = 2147483647;
    let value = seed % RANDOM_NUMBER;
    if (value <= 0) value += RANDOM_NUMBER;

    return function () {
        value = (value * 16807) % RANDOM_NUMBER;
        return (value - 1) / RANDOM_NUMBER;
    };
}

function genDirMatrix(rand, k) {
    const rawMatrix = Array.from({length: n}, () =>
        Array.from({length: n}, () => rand() * 2.0)
    );

    const dirMatrix = rawMatrix.map((row) =>
        row.map((value) => (value * k >= 1.0 ? 1 : 0))
    );

    return dirMatrix;
}

function genUndirMatrix(dir) {
    const undir = Array.from({length: n}, () => Array(n).fill(0));

    for (let i = 0; i < n; i++) {
        for (let j = 0; j < n; j++) {
            if (dir[i][j] === 1 || dir[j][i] === 1) {
                undir[i][j] = 1;
                undir[j][i] = 1;
            }
        }
    }

    return undir;
}

function printMatrix(matrix, title) {
    console.log(`\n${title}\n`);
    matrix.forEach((row) => {
        const line = row.map((v) => String(v).padStart(2, ' ')).join(' ') + ' ';
        console.log(line);
    });
}

function printMatrix2(matrix, title) {
    console.log(`\n${title}\n`);
    matrix.forEach((row, i) => {
        const line = row.map((v) => String(v).padStart(2, ' ')).join(' ');
        console.log(`${line}  // row ${i + 1}`);
```

```
    });
}

//drawing
const PADD = 50;

const positions = [
    {x: PADD, y: PADD},
    {x: w / 2, y: PADD},
    {x: w - PADD, y: PADD},
    {x: w - PADD, y: h / 2},
    {x: w - PADD, y: h - PADD},
    {x: (w / 3) * 2, y: h - PADD},
    {x: w / 3, y: h - PADD},
    {x: PADD, y: h - PADD},
    {x: PADD, y: h / 2},
    {x: w / 2, y: h / 2},
];

function distanceToLine(p1, p2, p) {
    const A = p.x - p1.x;
    const B = p.y - p1.y;
    const C = p2.x - p1.x;
    const D = p2.y - p1.y;

    const scal = A * C + B * D;
    const len2 = C * C + D * D;
    const param = scal / len2;

    let xx;
    let yy;

    if (param < 0) {
        xx = p1.x;
        yy = p1.y;
    } else if (param > 1) {
        xx = p2.x;
        yy = p2.y;
    } else {
        xx = p1.x + param * C;
        yy = p1.y + param * D;
    }

    const vx = p.x - xx;
    const vy = p.y - yy;
    return Math.sqrt(vx * vx + vy * vy);
}

function drawArrow(from, to, rad, controlPoint) {
    let angle;

    if (controlPoint) {
        const t = 0.95;
        const x =
            2 * (1 - t) * (controlPoint.x - from.x) + 2 * t * (to.x -
controlPoint.x);
        const y =
            2 * (1 - t) * (controlPoint.y - from.y) + 2 * t * (to.y -
controlPoint.y);
        angle = Math.atan2(y, x);
    } else {
        angle = Math.atan2(to.y - from.y, to.x - from.x);
    }

    const x = to.x - rad * Math.cos(angle);
    const y = to.y - rad * Math.sin(angle);
```

```
    ctx.beginPath();
    ctx.moveTo(x, y);
    ctx.lineTo(
        x - 10 * Math.cos(angle - Math.PI / 10),
        y - 10 * Math.sin(angle - Math.PI / 10)
    );
    ctx.lineTo(
        x - 10 * Math.cos(angle + Math.PI / 10),
        y - 10 * Math.sin(angle + Math.PI / 10)
    );
    ctx.fill();
    ctx.closePath();
}

function getCurvPoint(p1, p2, i, j, OFFSET, direction) {
    const midX = (p1.x + p2.x) / 2;
    const midY = (p1.y + p2.y) / 2;

    const vx = p2.x - p1.x;
    const vy = p2.y - p1.y;

    const perp = {x: -vy, y: vx};

    const length = Math.sqrt(perp.x * perp.x + perp.y * perp.y);

    const dir = direction ? direction : i < j ? 1 : -1;

    const point = {
        x: midX + dir * (perp.x / length) * OFFSET,
        y: midY + dir * (perp.y / length) * OFFSET,
    };

    return point;
}

function drawGraph(matrix, directed = true, n) {
    ctx.clearRect(0, 0, w, h);

    const RAD = 20;

    for (let i = 0; i < n; i++) {
        for (let j = 0; j < n; j++) {
            if (matrix[i][j] === 1) {
                if (directed && matrix[j][i] === 1 && j < i) continue;
                if (!directed && j < i) continue;
                if (i === j) {
                    let offsetX;
                    let offsetY;
                    const x = positions[i].x;
                    const y = positions[i].y;

                    if (y === PADD) {
                        offsetX = 0;
                        offsetY = -20;
                    } else if (y === h / 2 && x === w - PADD) {
                        offsetX = 20;
                        offsetY = 0;
                    } else if (y === h / 2 || (x === w / 2 && y === h / 2)) {
                        offsetX = -20;
                        offsetY = 0;
                    } else if (y === h - PADD) {
                        offsetX = 0;
                        offsetY = 20;
                    }
                    const cx = positions[i].x + offsetX;
```

```
                    const cy = positions[i].y + offsetY;

                    ctx.beginPath();
                    ctx.arc(cx, cy, RAD, Math.PI, -Math.PI);
                    ctx.stroke();

                    continue;
                }
                const p1 = positions[i];
                const p2 = positions[j];

                let curved = false;
                let curvPoint = null;

                for (let k = 0; k < n; k++) {
                    if (j === i) break;
                    if (k === i || k === j) continue;
                    const pk = positions[k];

                    if (distanceToLine(p1, p2, pk) < 25) {
                        curved = true;

                        curvPoint = getCurvPoint(p1, p2, i, j, 90);
                    }
                }

                ctx.beginPath();
                ctx.moveTo(p1.x, p1.y);

                if (curved) {
                    ctx.quadraticCurveTo(curvPoint.x, curvPoint.y, p2.x, p2.y);
                } else {
                    ctx.lineTo(p2.x, p2.y);
                }
                ctx.stroke();

                if (directed) {
                    drawArrow(p1, p2, RAD, curved ? curvPoint : null);

                    if (matrix[j][i] === 1) {
                        const curvPoint2 = getCurvPoint(p1, p2, i, j, 90, -1);

                        ctx.beginPath();
                        ctx.moveTo(p1.x, p1.y);

                        ctx.quadraticCurveTo(curvPoint2.x, curvPoint2.y, p2.x,
p2.y);
                        ctx.stroke();

                        drawArrow(p2, p1, RAD, curvPoint2);
                    }
                }
            }
        }
    }

    for (let i = 0; i < n; i++) {
        ctx.beginPath();
        ctx.arc(positions[i].x, positions[i].y, RAD, Math.PI, -Math.PI);
        ctx.fillStyle = 'white';
        ctx.fill();
        ctx.stroke();
        ctx.fillStyle = 'black';
        ctx.font = '15px Arial';
        ctx.fillText(i + 1, positions[i].x - 5, positions[i].y + 5);
    }
```

```
}

//button's functions
const rand = genRandNum(seed);
const dirMatrix1 = genDirMatrix(rand, k1);
const undirMatrix1 = genUndirMatrix(dirMatrix1);
const dirMatrix2 = genDirMatrix(rand, k2);

printMatrix(dirMatrix1, 'Directed Matrix(k1)');
printMatrix(undirMatrix1, 'Undirected Matrix(k1)');
printMatrix(dirMatrix2, 'Directed Matrix(k2)');

function drawDirected1() {
    drawGraph(dirMatrix1, true, n);
}

function drawUndirected1() {
    drawGraph(undirMatrix1, false, n);
}

function drawDirected2() {
    drawGraph(dirMatrix2, true, n);
}

//******************************
//lab4
//******************************

console.log('\ninfo about k1 graphs:\n');

function getDegrees(matrix, directed = false) {
    const degrees = [];
    for (let i = 0; i < n; i++) {
        let deg = 0;
        for (let j = 0; j < n; j++) {
            deg += matrix[i][j];
            if (directed) {
                deg += matrix[j][i];
            } else if (matrix[j][i] === 1 && j === i) {
                deg += 1;
            }
        }
        degrees.push(deg);
        console.log(`apex ${i + 1} : ${deg} connection(s)`);
    }
    return degrees;
}

console.log('\ndirected graph:');
const dirDegs = getDegrees(dirMatrix1, true);
console.log('undirected graph:');
const undirDegs = getDegrees(undirMatrix1, false);

function showInOutDegrees(matrix) {
    for (let i = 0; i < n; i++) {
        let inDeg = 0;
        let outDeg = 0;
        for (let j = 0; j < n; j++) {
            outDeg += matrix[i][j];
            inDeg += matrix[j][i];
        }
        console.log(`apex ${i + 1}: out: ${outDeg} in: ${inDeg}`);
    }
}

console.log('\nin/out connections of directed matrix:');
```

```
showInOutDegrees(dirMatrix1);

function isRegular(degrees) {
    const first = degrees[0];
    const check = degrees.every((deg) => deg === first);
    console.log(`${check}, degree: ${check ? first : 'none'}`);
}

console.log('\n is directed matrix regular?');
isRegular(dirDegs);
console.log('is undirected matrix regular?');
isRegular(undirDegs);

function getHangingApexes(degrees) {
    const apexes = [];
    for (let i = 0; i < n; i++) {
        if (degrees[i] === 1) {
            apexes.push(i + 1);
        }
    }
    return apexes;
}

console.log('\nhanging apexes of directed matrix:');
console.log(...getHangingApexes(dirDegs));
console.log('hanging apexes of undirected matrix:');
console.log(...getHangingApexes(undirDegs));

function getIsolatedApexes(degrees) {
    const apexes = [];
    for (let i = 0; i < n; i++) {
        if (degrees[i] === 0) {
            apexes.push(i + 1);
        }
    }
    if (apexes.length === 0) {
        apexes.push('none');
    }
    return apexes;
}

console.log('isolated apexes of directed matrix:');
console.log(...getIsolatedApexes(dirDegs));
console.log('isolated apexes of undirected matrix:');
console.log(...getIsolatedApexes(undirDegs));

console.log('\ninfo about k2 directed graph:\n');

console.log('\nin/out connections of directed matrix:');
showInOutDegrees(dirMatrix2);

function multiMatrix(M1, M2) {
    const result = Array.from({length: n}, () => Array(n).fill(0));

    for (let i = 0; i < n; i++) {
        for (let j = 0; j < n; j++) {
            for (let k = 0; k < n; k++) {
                result[i][j] += M1[i][k] * M2[k][j];
            }
        }
    }

    return result;
}

const mMatrix2 = multiMatrix(dirMatrix2, dirMatrix2);
```

```
const mMatrix3 = multiMatrix(mMatrix2, dirMatrix2);

function getPairs(matrix) {
    const pairs = [];

    for (let i = 0; i < n; i++) {
        for (let j = 0; j < n; j++) {
            if (matrix[i][j] > 0) {
                pairs.push([i, j]);
            }
        }
    }

    return pairs;
}

const pairs2 = getPairs(mMatrix2);
const pairs3 = getPairs(mMatrix3);

function find2Paths(matrix, pairs) {
    const paths = [];

    for (const [i, j] of pairs) {
        for (let k = 0; k < n; k++) {
            if (matrix[i][k] === 1 && matrix[k][j] === 1) {
                paths.push([i + 1, k + 1, j + 1]);
            }
        }
    }

    return paths;
}

const paths2 = find2Paths(dirMatrix2, pairs2);
console.log(`\npaths with length 2 (${paths2.length})`);
paths2.forEach((path) => {
    console.log(path.join(' -> '));
});

function find3Paths(matrix, pairs) {
    const paths = [];

    for (const [i, j] of pairs) {
        for (let k = 0; k < n; k++) {
            if (matrix[i][k] !== 1) continue;
            for (let m = 0; m < n; m++) {
                if (matrix[k][m] !== 1) continue;
                if (matrix[m][j] === 1) {
                    paths.push([i + 1, k + 1, m + 1, j + 1]);
                }
            }
        }
    }

    return paths;
}

const paths3 = find3Paths(dirMatrix2, pairs3);
console.log(`paths with length 3 (${paths3.length})`);
paths3.forEach((path) => {
    console.log(path.join(' -> '));
});

function warshall(matrix) {
    const n = matrix.length;
    const reach = matrix.map((row) => [...row]);
```

```
    for (let k = 0; k < n; k++) {
        for (let i = 0; i < n; i++) {
            for (let j = 0; j < n; j++) {
                reach[j][j] = 1;
                if (reach[i][k] === 1 && reach[k][j] === 1) {
                    reach[i][j] = 1;
                }
            }
        }
    }

    return reach;
}

const reachMatrix = warshall(dirMatrix2);
printMatrix2(reachMatrix, '\nreach matrix');

function checkStrongConn(matrix) {
    const result = Array.from({length: n}, () => Array(n).fill(0));

    for (let i = 0; i < n; i++) {
        for (let j = 0; j < n; j++) {
            if (i < j) continue;
            if (matrix[i][j] === 1 && matrix[j][i] === 1) {
                result[i][j] = 1;
                result[j][i] = 1;
            }
        }
    }

    return result;
}

const strongConnMatrix = checkStrongConn(reachMatrix);
printMatrix2(strongConnMatrix, '\nmatrix of strong connections');

function getStrongComponents(matrix) {
    const visited = Array(n).fill(false);
    const components = [];

    for (let i = 0; i < n; i++) {
        if (visited[i]) continue;

        const comp = [];
        for (let j = 0; j < n; j++) {
            if (i > j) continue;
            if (!visited[j] && matrix[i][j] === 1) {
                comp.push(j + 1);
                visited[j] = true;
            }
        }

        components.push(comp);
    }

    return components;
}

const components = getStrongComponents(strongConnMatrix);


console.log('\nStrong connection components:');
components.forEach((comp, i) => {
    console.log(`${i + 1}) ${comp.join(', ')}`);
});
```

```
function condensedMatrix(matrixOG, comps) {
    const n = comps.length;
    const condensed = Array.from({length: n}, () => Array(n).fill(0));

    const apexesToComps = new Map();
    comps.forEach((comp, i) => {
        comp.forEach((apex) => {
            apexesToComps.set(apex, i);
        });
    });

    for (let i = 0; i <matrixOG.length; i++) {
        for (let j = 0; j <matrixOG.length; j++) {
            if (matrixOG[i][j] === 1) {
                const compI = apexesToComps.get(i + 1);
                const compJ = apexesToComps.get(j + 1);
                if (compI !== undefined && compJ !== undefined && compI !==
compJ) {
                    condensed[compI][compJ] = 1;
                }
            }
        }
    }

    return condensed;
}


const condensed = condensedMatrix(dirMatrix2, components);


printMatrix2(condensed, '\nCondensed matrix:');

function drawCondense() {
    drawGraph(condensed, true, condensed.length);
}
```

За п. 1 завдання: згенеровані матриці суміжності напрямленого та не-

напрямленого графів:

Directed Matrix(k1)

0	0	0	1	1	0	1	0	0	0
0	0	0	0	1	1	0	0	0	0
0	1	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0
0	0	1	0	0	1	1	1	0	0
0	0	0	1	0	0	0	0	1	1
0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	1

Undirected Matrix(k1)

0	0	0	1	1	0	1	1	0	0
0	0	1	0	1	1	0	0	0	0
0	1	0	0	1	0	1	1	0	0
1	0	0	0	0	1	0	1	0	1
1	1	1	0	0	1	1	1	0	0
0	1	0	1	1	0	0	0	1	1
1	0	1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	1	0	1	0	0	0	1

За п. 2: результат перевірки на однорідність, переліки висячих та ізольованих вершин,перелік степенів, півстепенів,:

```
 is directed matrix regular?
false, degree: none
is undirected matrix regular?
false, degree: none

hanging apexes of directed matrix:
9
hanging apexes of undirected matrix:
9
isolated apexes of directed matrix:
none
isolated apexes of undirected matrix:
none
```

```
directed graph:
apex 1 : 4 connection(s)
apex 2 : 3 connection(s)
apex 3 : 4 connection(s)
apex 4 : 4 connection(s)
apex 5 : 6 connection(s)
apex 6 : 6 connection(s)
apex 7 : 5 connection(s)
apex 8 : 4 connection(s)
apex 9 : 1 connection(s)
apex 10 : 5 connection(s)
undirected graph:
apex 1 : 4 connection(s)
apex 2 : 3 connection(s)
apex 3 : 4 connection(s)
apex 4 : 4 connection(s)
apex 5 : 6 connection(s)
apex 6 : 5 connection(s)
apex 7 : 5 connection(s)
apex 8 : 4 connection(s)
apex 9 : 1 connection(s)
apex 10 : 4 connection(s)

in/out connections of directed matrix:
apex 1: out: 3 in: 1
apex 2: out: 2 in: 1
apex 3: out: 2 in: 2
apex 4: out: 1 in: 3
apex 5: out: 4 in: 2
apex 6: out: 3 in: 3
apex 7: out: 1 in: 4
apex 8: out: 2 in: 2
apex 9: out: 0 in: 1
apex 10: out: 3 in: 2
```

За п. 3: матриця другого орграфа:

Directed Matrix(k2)

1	1	0	0	1	0	0	0	1	0
0	1	0	0	0	0	0	0	0	1
1	0	0	0	0	1	0	0	0	0
0	1	0	0	0	0	1	1	1	0
0	0	0	0	1	0	1	0	1	0
0	0	0	0	0	1	0	0	1	0
1	1	0	0	0	0	0	0	0	0
0	1	1	0	0	0	1	0	0	1
1	0	0	0	0	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0

За п. 4: переліки півстепенів, шляхів, матриці досяжності та сильної зв’язності, перелік компонент сильної зв’язності, граф конденсації:

in/out connections of directed matrix:

apex 1: out: 4 in: 4

apex 2: out: 2 in: 5

apex 3: out: 2 in: 1

apex 4: out: 4 in: 0

apex 5: out: 3 in: 2

apex 6: out: 2 in: 3

apex 7: out: 2 in: 4

apex 8: out: 4 in: 2

apex 9: out: 4 in: 4

apex 10: out: 0 in: 2

paths with length 2 (72)

1 -> 1 -> 1

1 -> 9 -> 1

1 -> 1 -> 2

1 -> 2 -> 2

1 -> 1 -> 5

1 -> 5 -> 5

1 -> 9 -> 6

1 -> 5 -> 7

1 -> 9 -> 7

1 -> 9 -> 8

1 -> 1 -> 9

1 -> 5 -> 9

1 -> 2 -> 10

2 -> 2 -> 2

2 -> 2 -> 10

3 -> 1 -> 1

3 -> 1 -> 2

3 -> 1 -> 5

3 -> 6 -> 6

3 -> 1 -> 9

3 -> 6 -> 9

4 -> 7 -> 1

4 -> 9 -> 1

4 -> 2 -> 2

4 -> 7 -> 2

4 -> 8 -> 2

4 -> 8 -> 3

4 -> 9 -> 6

4 -> 8 -> 7

4 -> 9 -> 7

4 -> 9 -> 8

4 -> 2 -> 10

4 -> 8 -> 10

5 -> 7 -> 1

5 -> 9 -> 1

5 -> 7 -> 2

5 -> 5 -> 5

5 -> 9 -> 6

```
5 -> 5 -> 7
5 -> 9 -> 7
5 -> 9 -> 8
5 -> 5 -> 9
6 -> 9 -> 1
6 -> 6 -> 6
6 -> 9 -> 6
6 -> 9 -> 7
6 -> 9 -> 8
6 -> 6 -> 9
7 -> 1 -> 1
7 -> 1 -> 2
7 -> 2 -> 2
7 -> 1 -> 5
7 -> 1 -> 9
7 -> 2 -> 10
8 -> 3 -> 1
8 -> 7 -> 1
8 -> 2 -> 2
8 -> 7 -> 2
8 -> 3 -> 6
8 -> 2 -> 10
9 -> 1 -> 1
9 -> 7 -> 1
9 -> 1 -> 2
9 -> 7 -> 2
9 -> 8 -> 2
9 -> 8 -> 3
9 -> 1 -> 5
9 -> 6 -> 6
9 -> 8 -> 7
9 -> 1 -> 9
9 -> 6 -> 9
9 -> 8 -> 10
```

paths with length 3 (188)
1 -> 1 -> 1 -> 1
1 -> 1 -> 9 -> 1
1 -> 5 -> 7 -> 1
1 -> 5 -> 9 -> 1
1 -> 9 -> 1 -> 1
1 -> 9 -> 7 -> 1
1 -> 1 -> 1 -> 2
1 -> 1 -> 2 -> 2
1 -> 2 -> 2 -> 2
1 -> 5 -> 7 -> 2
1 -> 9 -> 1 -> 2
1 -> 9 -> 7 -> 2
1 -> 9 -> 8 -> 2
1 -> 9 -> 8 -> 3
1 -> 1 -> 1 -> 5
1 -> 1 -> 5 -> 5
1 -> 5 -> 5 -> 5
1 -> 9 -> 1 -> 5
1 -> 1 -> 9 -> 6
1 -> 5 -> 9 -> 6
1 -> 9 -> 6 -> 6
1 -> 1 -> 5 -> 7
1 -> 1 -> 9 -> 7
1 -> 5 -> 5 -> 7
1 -> 5 -> 9 -> 7
1 -> 9 -> 8 -> 7
1 -> 1 -> 9 -> 8
1 -> 5 -> 9 -> 8
1 -> 1 -> 1 -> 9
1 -> 1 -> 5 -> 9
1 -> 5 -> 5 -> 9
1 -> 9 -> 1 -> 9
1 -> 9 -> 6 -> 9
1 -> 1 -> 2 -> 10
1 -> 2 -> 2 -> 10
1 -> 9 -> 8 -> 10
2 -> 2 -> 2 -> 2
2 -> 2 -> 2 -> 10
3 -> 1 -> 1 -> 1
3 -> 1 -> 9 -> 1

3 -> 6 -> 9 -> 1

3 -> 1 -> 1 -> 2

3 -> 1 -> 2 -> 2

3 -> 1 -> 1 -> 5

3 -> 1 -> 5 -> 5

3 -> 1 -> 9 -> 6

3 -> 6 -> 6 -> 6

3 -> 6 -> 9 -> 6

3 -> 1 -> 5 -> 7

3 -> 1 -> 9 -> 7

3 -> 6 -> 9 -> 7

3 -> 1 -> 9 -> 8

3 -> 6 -> 9 -> 8

3 -> 1 -> 1 -> 9

3 -> 1 -> 5 -> 9

3 -> 6 -> 6 -> 9

3 -> 1 -> 2 -> 10

4 -> 7 -> 1 -> 1

4 -> 8 -> 3 -> 1

4 -> 8 -> 7 -> 1

4 -> 9 -> 1 -> 1

4 -> 9 -> 7 -> 1

4 -> 2 -> 2 -> 2

4 -> 7 -> 1 -> 2

4 -> 7 -> 2 -> 2

4 -> 8 -> 2 -> 2

4 -> 8 -> 7 -> 2

4 -> 9 -> 1 -> 2

4 -> 9 -> 7 -> 2

4 -> 9 -> 8 -> 2

4 -> 9 -> 8 -> 3

4 -> 7 -> 1 -> 5

4 -> 9 -> 1 -> 5

4 -> 8 -> 3 -> 6

4 -> 9 -> 6 -> 6

4 -> 9 -> 8 -> 7

4 -> 7 -> 1 -> 9

4 -> 9 -> 1 -> 9

4 -> 9 -> 6 -> 9

4 -> 2 -> 2 -> 10

4 -> 7 -> 2 -> 10

4 -> 8 -> 2 -> 10

4 -> 9 -> 8 -> 10

5 -> 5 -> 7 -> 1

5 -> 5 -> 9 -> 1

5 -> 7 -> 1 -> 1

5 -> 9 -> 1 -> 1

5 -> 9 -> 7 -> 1

5 -> 5 -> 7 -> 2

5 -> 7 -> 1 -> 2

5 -> 7 -> 2 -> 2

5 -> 9 -> 1 -> 2

5 -> 9 -> 7 -> 2

5 -> 9 -> 8 -> 2

5 -> 9 -> 8 -> 3

5 -> 5 -> 5 -> 5

5 -> 7 -> 1 -> 5

5 -> 9 -> 1 -> 5

5 -> 5 -> 9 -> 6

5 -> 9 -> 6 -> 6

5 -> 5 -> 5 -> 7

5 -> 5 -> 9 -> 7

5 -> 9 -> 8 -> 7

5 -> 5 -> 9 -> 8

5 -> 5 -> 5 -> 9

5 -> 7 -> 1 -> 9

5 -> 9 -> 1 -> 9

5 -> 9 -> 6 -> 9

5 -> 7 -> 2 -> 10

5 -> 9 -> 8 -> 10

6 -> 6 -> 9 -> 1

6 -> 9 -> 1 -> 1

6 -> 9 -> 7 -> 1

6 -> 9 -> 1 -> 2

6 -> 9 -> 7 -> 2

6 -> 9 -> 8 -> 2

6 -> 9 -> 8 -> 3

6 -> 9 -> 1 -> 5

6 -> 6 -> 6 -> 6

6 -> 6 -> 9 -> 6

6 -> 9 -> 6 -> 6

6 -> 6 -> 9 -> 7

6 -> 9 -> 8 -> 7

6 -> 6 -> 9 -> 8

6 -> 6 -> 6 -> 9

6 -> 9 -> 1 -> 9

6 -> 9 -> 6 -> 9

6 -> 9 -> 8 -> 10

7 -> 1 -> 1 -> 1

7 -> 1 -> 9 -> 1

7 -> 1 -> 1 -> 2

7 -> 1 -> 2 -> 2

7 -> 2 -> 2 -> 2

7 -> 1 -> 1 -> 5

7 -> 1 -> 5 -> 5

7 -> 1 -> 9 -> 6

7 -> 1 -> 5 -> 7

7 -> 1 -> 9 -> 7

7 -> 1 -> 9 -> 8

7 -> 1 -> 1 -> 9

7 -> 1 -> 5 -> 9

7 -> 1 -> 2 -> 10

7 -> 2 -> 2 -> 10

8 -> 3 -> 1 -> 1

8 -> 7 -> 1 -> 1

8 -> 2 -> 2 -> 2

8 -> 3 -> 1 -> 2

8 -> 7 -> 1 -> 2

8 -> 7 -> 2 -> 2

8 -> 3 -> 1 -> 5

8 -> 7 -> 1 -> 5

8 -> 3 -> 6 -> 6

8 -> 3 -> 1 -> 9

8 -> 3 -> 6 -> 9

8 -> 7 -> 1 -> 9

8 -> 2 -> 2 -> 10

8 -> 7 -> 2 -> 10

9 -> 1 -> 1 -> 1

9 -> 1 -> 9 -> 1

9 -> 6 -> 9 -> 1

9 -> 7 -> 1 -> 1

9 -> 8 -> 3 -> 1

9 -> 8 -> 7 -> 1

9 -> 1 -> 1 -> 2

9 -> 1 -> 2 -> 2

```
9 -> 8 -> 2 -> 2
9 -> 8 -> 7 -> 2
9 -> 1 -> 1 -> 5
9 -> 1 -> 5 -> 5
9 -> 7 -> 1 -> 5
9 -> 1 -> 9 -> 6
9 -> 6 -> 6 -> 6
9 -> 6 -> 9 -> 6
9 -> 8 -> 3 -> 6
9 -> 1 -> 5 -> 7
9 -> 1 -> 9 -> 7
9 -> 6 -> 9 -> 7
9 -> 1 -> 9 -> 8
9 -> 6 -> 9 -> 8
9 -> 1 -> 1 -> 9
9 -> 1 -> 5 -> 9
9 -> 6 -> 6 -> 9
9 -> 7 -> 1 -> 9
9 -> 1 -> 2 -> 10
9 -> 7 -> 2 -> 10
9 -> 8 -> 2 -> 10
```

reach matrix

```
 1  1  1  0  1  1  1  1  1  1  // row 1
 0  1  0  0  0  0  0  0  0  1  // row 2
 1  1  1  0  1  1  1  1  1  1  // row 3
 1  1  1  1  1  1  1  1  1  1  // row 4
 1  1  1  0  1  1  1  1  1  1  // row 5
 1  1  1  0  1  1  1  1  1  1  // row 6
 1  1  1  0  1  1  1  1  1  1  // row 7
 1  1  1  0  1  1  1  1  1  1  // row 8
 1  1  1  0  1  1  1  1  1  1  // row 9
 0  0  0  0  0  0  0  0  0  1  // row 10
```

matrix of strong connections

```
1  0  1  0  1  1  1  1  1  0  // row 1
0  1  0  0  0  0  0  0  0  0  // row 2
1  0  1  0  1  1  1  1  1  0  // row 3
0  0  0  1  0  0  0  0  0  0  // row 4
1  0  1  0  1  1  1  1  1  0  // row 5
1  0  1  0  1  1  1  1  1  0  // row 6
1  0  1  0  1  1  1  1  1  0  // row 7
1  0  1  0  1  1  1  1  1  0  // row 8
1  0  1  0  1  1  1  1  1  0  // row 9
0  0  0  0  0  0  0  0  0  1  // row 10
```

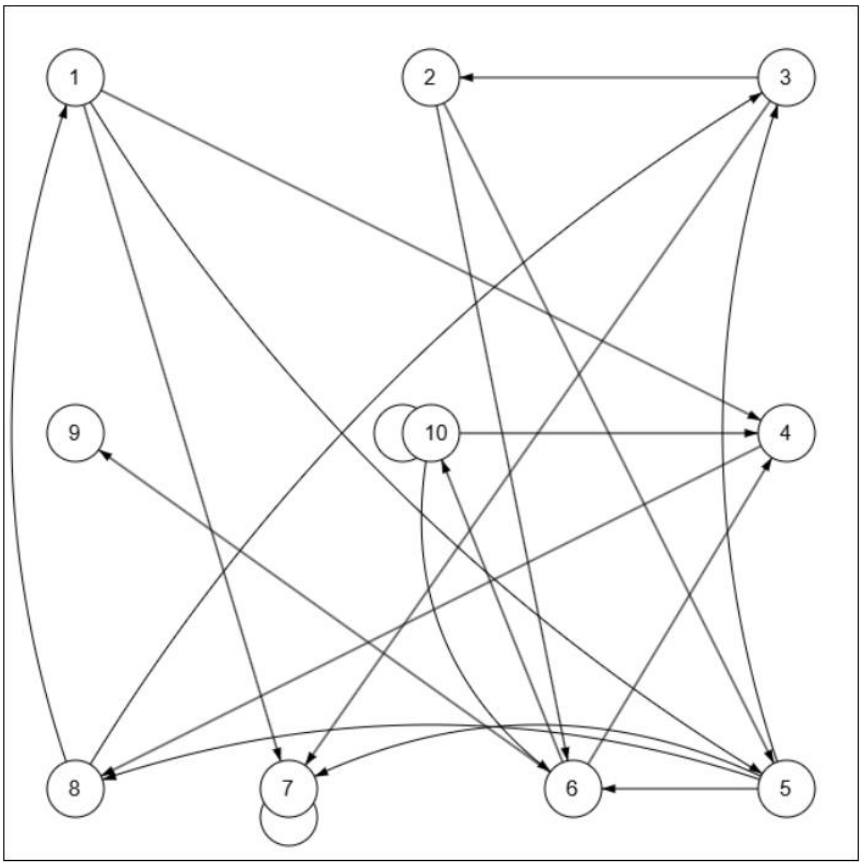
Strong connection components:

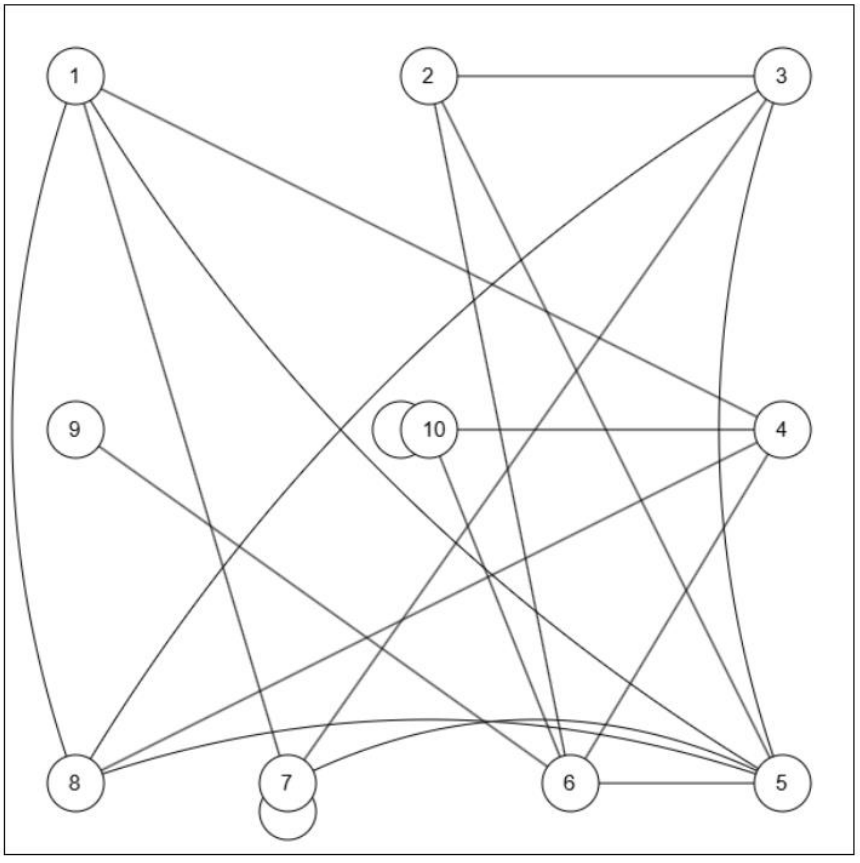
```
1) 1, 3, 5, 6, 7, 8, 9
2) 2
3) 4
4) 10
```

Condensed matrix:

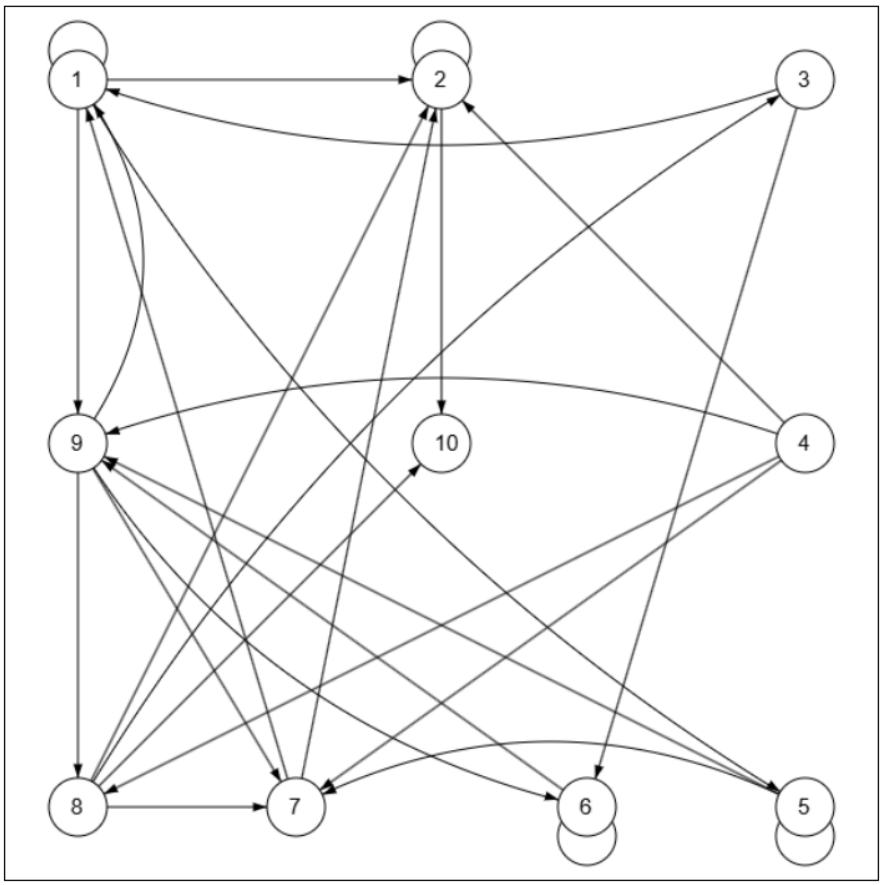
```
0  1  0  1  // row 1
0  0  0  1  // row 2
1  1  0  0  // row 3
0  0  0  0  // row 4
```

Початкові орієнтований і неорієнтований графи:

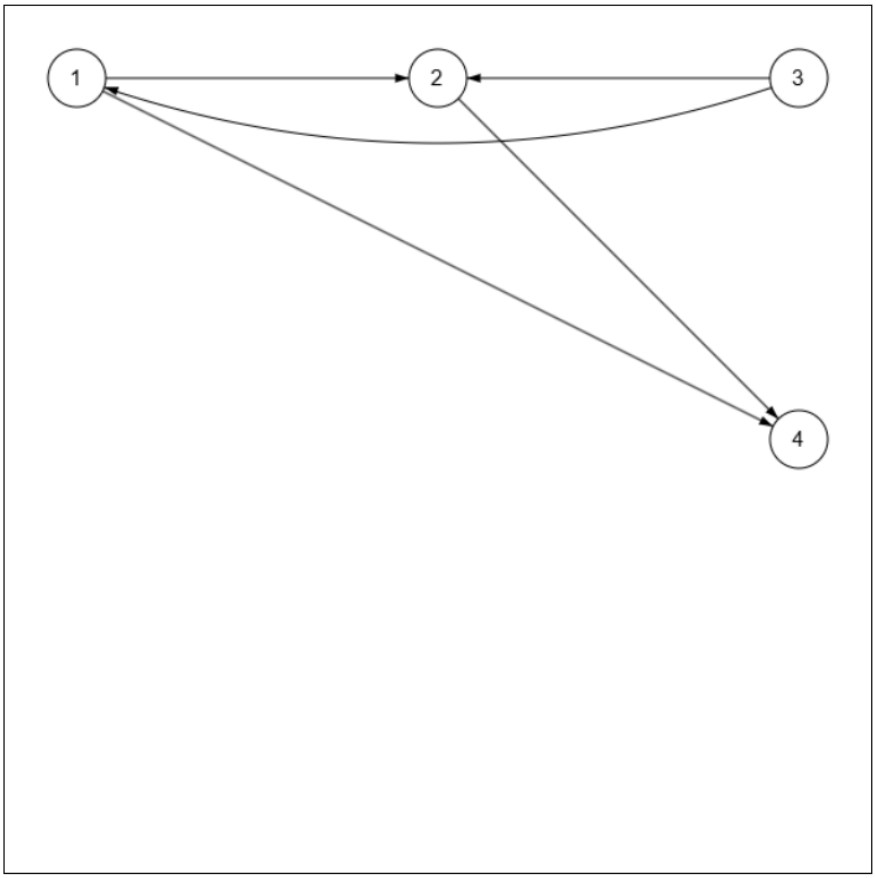




Модифіковий орієнторваний граф:



Граф кондинсації:



Висновок:

В ході виконання лабораторної роботи №4 «Характеристики та зв’язність графа» були досліджені характеристики графів та отримано навички визначення їх на конкретних прикладах, вивчено метод транзитивного замикання. Під час виконання роботи довелось повторити багато матеріалу з комп’ютерної дискретної математики. Текст програми був довгим у написанні.