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 №5

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 граф із заданими параметрами так само, як у лабораторній роботі No3.

Відмінність: коефіцієнт k = 1.0 - n3 * 0.01 - n4 * 0.005 - 0.15.

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 1.0n3 0.01n4 0.0050.15, 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. Створити програму, яка виконує обхід напрямленого графа вшир (BFS) та вглиб (DFS).

• обхід починати з вершини із найменшим номером, яка має щонайменше одну вихідну дугу;

• при обході враховувати порядок нумерації;

• у програмі виконання обходу відображати покроково, черговий крок виконувати за натисканням кнопки у вікні або на клавіатурі.

3.Під час обходу графа побудувати дерево обходу. У програмі дерево обходу виводити покроково у процесі виконання обходу графа. Це можна виконати одним із двох способів:

• або виділяти іншим кольором ребра графа;

• або будувати дерево обходу поряд із графом.

4. Зміну статусів вершин у процесі обходу продемонструвати зміною кольорів вершин, графічними позначками тощо, або ж у процесі обходу виводити протокол обходу у графічне вікно або в консоль.

5. Якщо після обходу графа лишилися невідвідані вершини, продовжувати обхід з невідвіданої вершини з найменшим номером, яка має щонайменше одну вихідну дугу.

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-container {
            display: flex;
            align-items: flex-start;
        }

        canvas {
            border: 1px solid black;
            background: #fff;
            margin-right: 20px;
            margin-top: 20px;
            margin-bottom: 20px;
            width: 700px;
            height: 700px;
        }

        button {
            margin-right: 10px;
            margin-top: 20px;
            padding: 10px 15px;
        }

    </style>
</head>
<body>
<h1>Варіант 4208</h1>
Спочатку проклацайте до кінця DFS,потім натисніть clear і проклацайте BFS
<div id="buttons">
    <button id="dfsButton">DFC</button>
    <button id="clButton">clear</button>
    <button id="bfsButton">BFC</button>
</div>

<div class="canvas-container">
    <canvas id="canvas2" width="700" height="700"></canvas>
    <canvas id="canvas1" width="700" height="700"></canvas>
</div>

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

JS скрипт:

```
//config
const canvas2 = document.getElementById('canvas2');
const canvas1 = document.getElementById('canvas1');
const ctx2 = canvas2.getContext('2d');
const ctx1 = canvas1.getContext('2d');
const seed = 4208;
const n3 = 0;
const n4 = 8;
const n = 10;
const k1 = 1.0 - n3 * 0.01 - n4 * 0.005 - 0.15;
const w = canvas1.width;
const h = canvas1.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 printMatrix(matrix, title) {
    console.log(`\n${title}\n`);
    let i = 0
    matrix.forEach((row) => {
        const line = row.map((v) => String(v).padStart(2, ' ')).join(' ') + ' ';
        console.log(line + `  row ${i + 1}`);
        i++
    });
}

//drawing
const PADD = 50;
const RAD = 20;

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

    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, ctx);

                    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, ctx);
                    }
                }
            }
        }
    }

    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);

printMatrix(dirMatrix1, 'Directed Matrix');

drawGraph(dirMatrix1, ctx1, true, n);

//**************
//lab 5
//**************

function getDFSorder(matrix) {
    const visited = Array(n).fill(false);
    const order = [];
    const stack = [];

    for (let startApex = 0; startApex < n; startApex++) {
        if (!visited[startApex]) {
            stack.push(startApex);
            visited[startApex] = true;

            while (stack.length > 0) {
                const currApex = stack.pop();

                for (let j = 0; j < n; j++) {
                    if (matrix[currApex][j] === 1 && !visited[j]) {
                        order.push([currApex, j]);
                        visited[j] = true;
                        stack.push(currApex);
                        stack.push(j);
                        break;
```

```
                    }
                }
            }
        }
    }

    return order;
}

function getBFSorder(matrix) {
    const visited = Array(n).fill(false);
    const order = {};
    let step = 1;

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

        const queue = [[-1, startApex]];
        visited[startApex] = true;

        while (queue.length > 0) {
            const stepOrder = [];
            const len = queue.length;

            for (let c = 0; c < len; c++) {
                const [parent, i] = queue.shift();

                if (parent !== -1) {
                    stepOrder.push([parent, i]);
                }

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

            if (stepOrder.length > 0) {
                order[`step${step}`] = stepOrder;
                step++;
            }
        }
    }

    return order;
}


function* drawTree(OGorder, bfs = false) {
    let firstDenie = true
    let lastJ = -1
    let order = null

    if (bfs) {
        order = adapter(OGorder);
    } else {
        order = OGorder;
    }

    //start function draw on canvas2
    for (let i = 0; i < n; i++) {
        ctx2.beginPath();
        ctx2.arc(positions[i].x, positions[i].y, RAD, Math.PI, -Math.PI);
        ctx2.fillStyle = 'white';
```

```
        ctx2.fill();
        ctx2.stroke();
        ctx2.fillStyle = 'black';
        ctx2.font = '15px Arial';
        ctx2.fillText(i + 1, positions[i].x - 5, positions[i].y + 5);
    }

    for (const [i, j] of order) {
        if (lastJ !== i && lastJ !== -1) {
            ctx2.beginPath();
            ctx2.arc(positions[lastJ].x, positions[lastJ].y, RAD, Math.PI, -
Math.PI);
            ctx2.fillStyle = 'blue';
            ctx2.fill();
            ctx2.strokeStyle = 'black';
            ctx2.stroke();
            ctx2.strokeStyle = 'green';
            ctx2.fillStyle = 'black';
            ctx2.font = '15px Arial';
            ctx2.fillText(lastJ + 1, positions[lastJ].x - 5, positions[lastJ].y
+ 5);

            ctx2.beginPath();
            ctx2.arc(positions[i].x, positions[i].y, RAD, Math.PI, -Math.PI);
            ctx2.fillStyle = 'green';
            ctx2.fill();
            ctx2.strokeStyle = 'black';
            ctx2.stroke();
            ctx2.strokeStyle = 'green';
            ctx2.fillStyle = 'black';
            ctx2.font = '15px Arial';
            ctx2.fillText(i + 1, positions[i].x - 5, positions[i].y + 5);

            yield
        }
        if (firstDenie) {
            ctx2.beginPath();
            ctx2.arc(positions[i].x, positions[i].y, RAD, Math.PI, -Math.PI);
            ctx2.fillStyle = 'green';
            ctx2.fill();
            ctx2.strokeStyle = 'black';
            ctx2.stroke();
            ctx2.strokeStyle = 'green';
            ctx2.fillStyle = 'black';
            ctx2.font = '15px Arial';
            ctx2.fillText(i + 1, positions[i].x - 5, positions[i].y + 5);

            firstDenie = false

            yield
        }

        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);
            }
        }
        ctx2.beginPath();
```

```
            ctx2.moveTo(p1.x, p1.y);
            if (curved) {
                ctx2.quadraticCurveTo(curvPoint.x, curvPoint.y, p2.x, p2.y);
            } else {
                ctx2.lineTo(p2.x, p2.y);
            }
            ctx2.strokeStyle = 'blue'
            ctx2.stroke();
            ctx2.strokeStyle = 'green'
            drawArrow(p1, p2, RAD, curved ? curvPoint : null, ctx2);
            ctx2.beginPath();
            ctx2.arc(positions[i].x, positions[i].y, RAD, Math.PI, -Math.PI);
            ctx2.fillStyle = 'blue';
            ctx2.fill();
            ctx2.strokeStyle = 'black';
            ctx2.stroke();
            ctx2.strokeStyle = 'blue';
            ctx2.fillStyle = 'black';
            ctx2.font = '15px Arial';
            ctx2.fillText(i + 1, positions[i].x - 5, positions[i].y + 5);

            ctx2.beginPath();
            ctx2.arc(positions[j].x, positions[j].y, RAD, Math.PI, -Math.PI);
            ctx2.fillStyle = 'green';
            ctx2.fill();
            ctx2.strokeStyle = 'black';
            ctx2.stroke();
            ctx2.strokeStyle = 'green';
            ctx2.fillStyle = 'black';
            ctx2.font = '15px Arial';
            ctx2.fillText(j + 1, positions[j].x - 5, positions[j].y + 5);

            lastJ = j
            yield
        }
    }

    function orderToMatrix(OGorder, bfs = false) {
        let order = null
        const matrix = Array.from({length: n}, () => Array(n).fill(0));

        if (bfs) {
            order = adapter(OGorder);
        } else {
            order = OGorder;
        }

        for (const [i, j] of order) {
            matrix[i][j] = 1;
        }

        return matrix;
    }

    function getVertex(OGorder, bfs = false) {
        let order = null
        const vertex = [];
        const oldToNew = {};
        let newIndex = 0;

        if (bfs) {
            order = adapter(OGorder);
        } else {
            order = OGorder;
        }
```

```
    function vertexSetter(index) {
        if (!(index in oldToNew)) {
            oldToNew[index] = newIndex;
            vertex[newIndex] = index + 1;
            newIndex++;
        }
    }

    for (const [i, j] of order) {
        vertexSetter(i)
        vertexSetter(j)
    }

    for (let i = 0; i < n; i++) {
        vertexSetter(i)
    }

    return vertex;
}

function adapter(bfsObj) {
    const result = [];

    for (const key in bfsObj) {
        result.push(...bfsObj[key]);
    }

    return result;
}


function printDFSorder(order) {
    const flatOrder = order.flat(100)
    console.log('\nDFS order:')
    for (let i = 0; i < flatOrder.length; i += 2) {
        console.log(`${flatOrder[i] + 1} -> ${flatOrder[i + 1] + 1}`);
    }
}

function printBFSorder(order) {
    console.log('\nBFS order:');
    for (const step in order) {
        console.log(`${step}:`);
        for (const [i, j] of order[step]) {
            console.log(`${i + 1} -> ${j + 1}`);
        }
    }
}

const DFSorder = getDFSorder(dirMatrix1)
const BFSorder = getBFSorder(dirMatrix1)

const DFSmatrix = orderToMatrix(DFSorder, false);
const BFSmatrix = orderToMatrix(BFSorder, true);

printMatrix(DFSmatrix, 'DFS tree matrix')
printMatrix(BFSmatrix, 'BFS tree matrix')
printDFSorder(DFSorder)
printBFSorder(BFSorder)

const DFSgen = drawTree(DFSorder, false)
const BFSgen = drawTree(BFSorder, true)

const dfsButton = document.getElementById('dfsButton');
const clButton = document.getElementById('clButton');
const bfsButton = document.getElementById('bfsButton');
```

```
console.log('\nVector with new sequence of apexes')
console.log('DFS:')
console.log(JSON.stringify(getVertex(DFSorder, false)))
console.log('BFS:')
console.log(JSON.stringify(getVertex(BFSorder, true)))

dfsButton.addEventListener('click', () => {
    DFSgen.next()
});

clButton.addEventListener('click', () => {
    ctx2.clearRect(0, 0, w, h);
    ctx2.strokeStyle = 'black';
});

bfsButton.addEventListener('click', () => {
    BFSgen.next()
});
```

Тестування програми :

Згенерована матриця суміжності напрямленого графа:

Directed Matrix

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

Матриці суміжності дерев обходу:

DFS tree matrix

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

BFS tree matrix

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

Список (вектор) відповідності номерів вершин і їх нової нумерації, набутої в процесі обходу:

Vector with new sequence of apexes

DFS:

[1,4,8,3,2,5,6,9,10,7]

BFS:

[1,4,5,7,8,3,6,2,9,10]

Скриншоти зображення графа та дерева обходу:

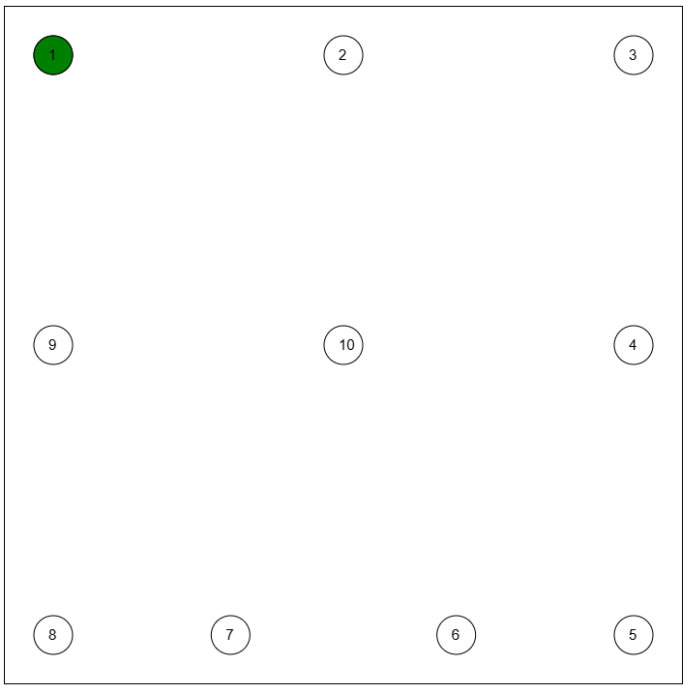
DFS:

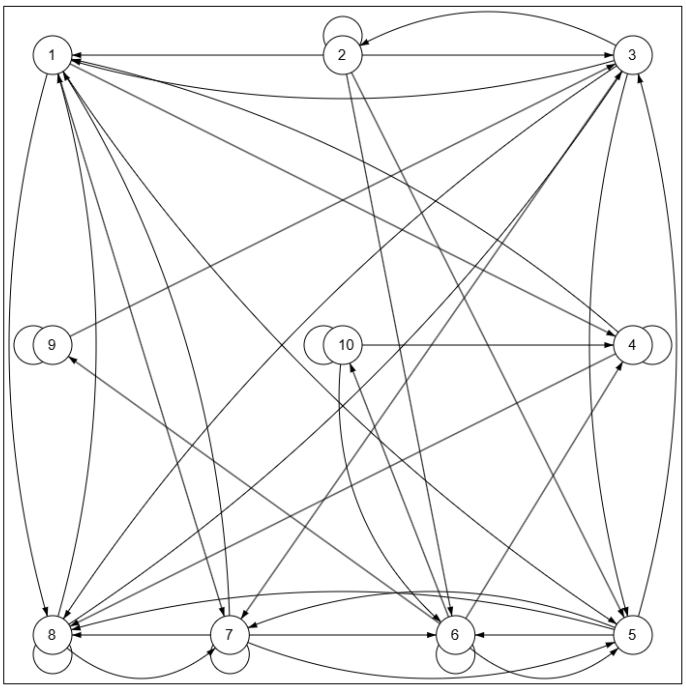
1)

Варіант 4208

Спочатку проклацайте до кінця DFS,потім натисніть clear і проклацайте BFS

DFC clear BFC



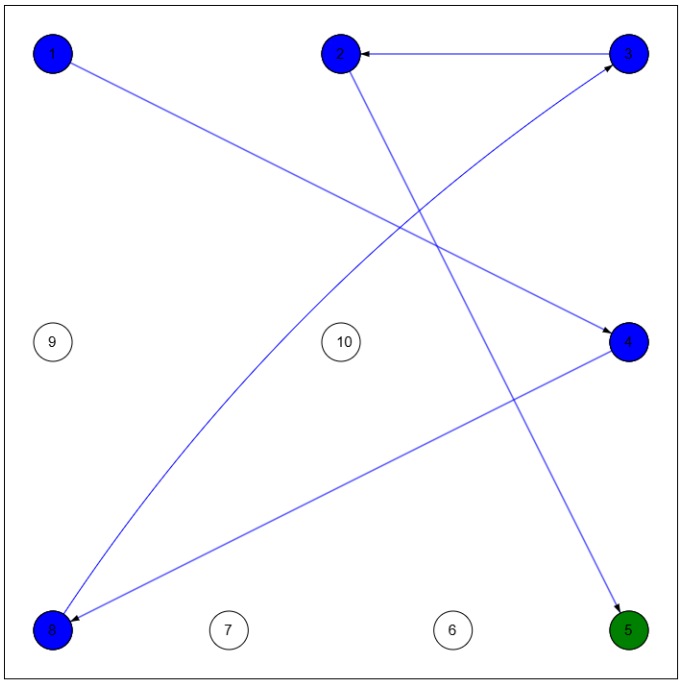


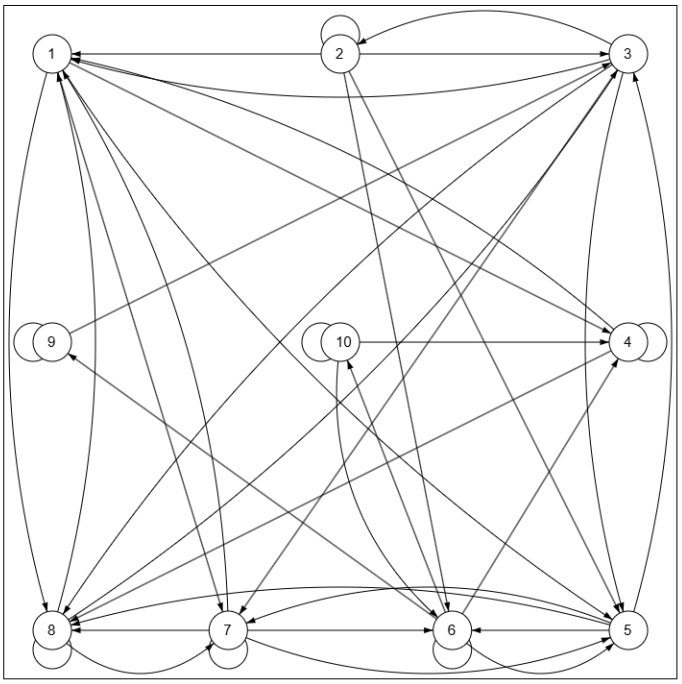
2)

Варіант 4208

Спочатку проклацайте до кінця DFS,потім натисніть clear і проклацайте BFS

DFC clear BFC



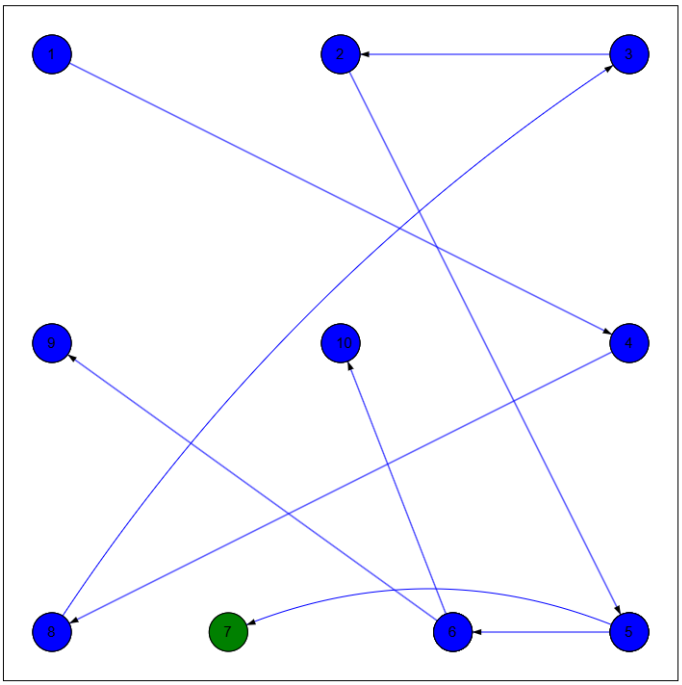


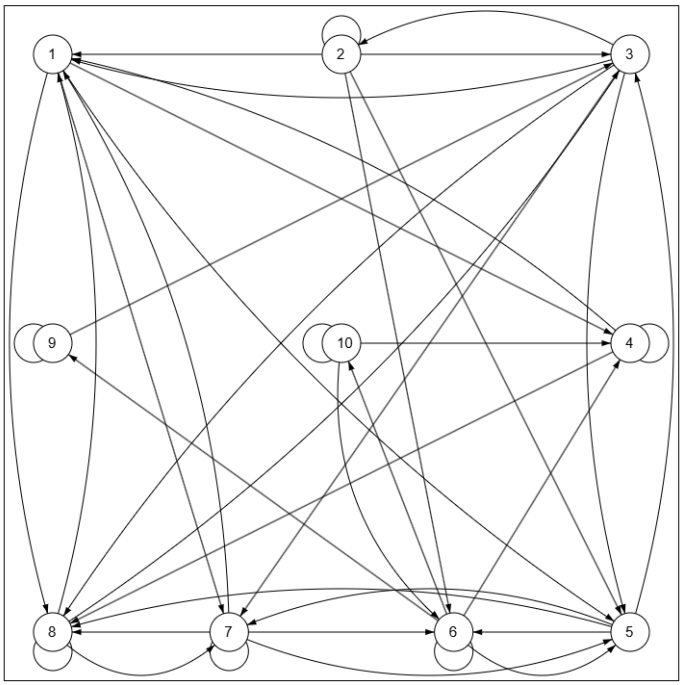
3)

Варіант 4208

Спочатку проклацайте до кінця DFS,потім натисніть clear і проклацайте BFS

DFC clear BFC





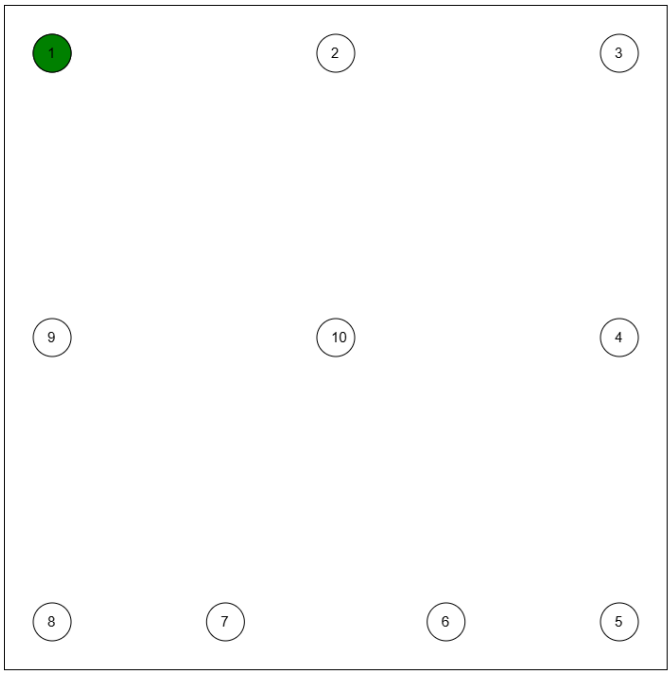
BFS:

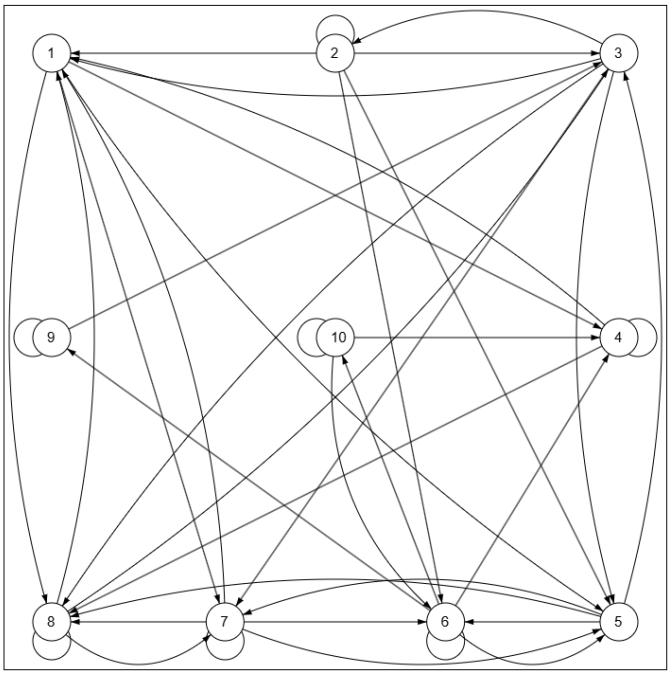
1)

Варіант 4208

Спочатку проклацайте до кінця DFS,потім натисніть clear і проклацайте BFS

DFC clear BFC



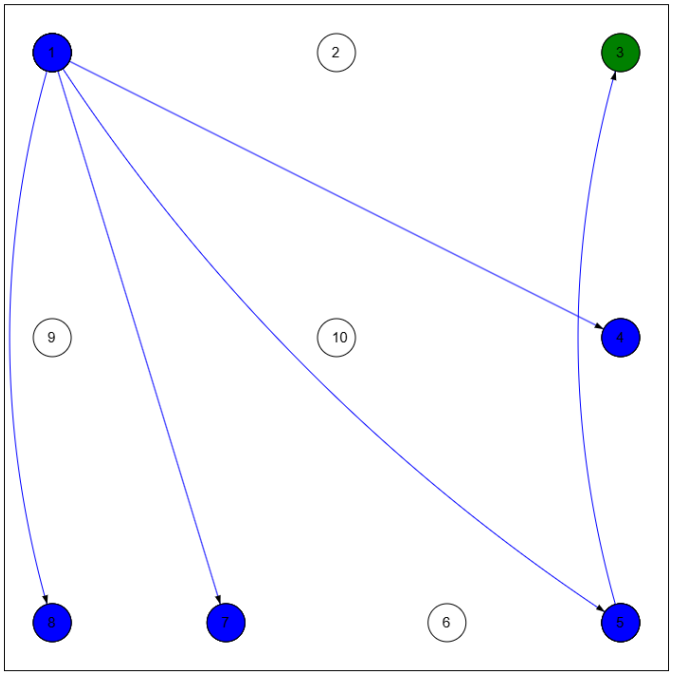


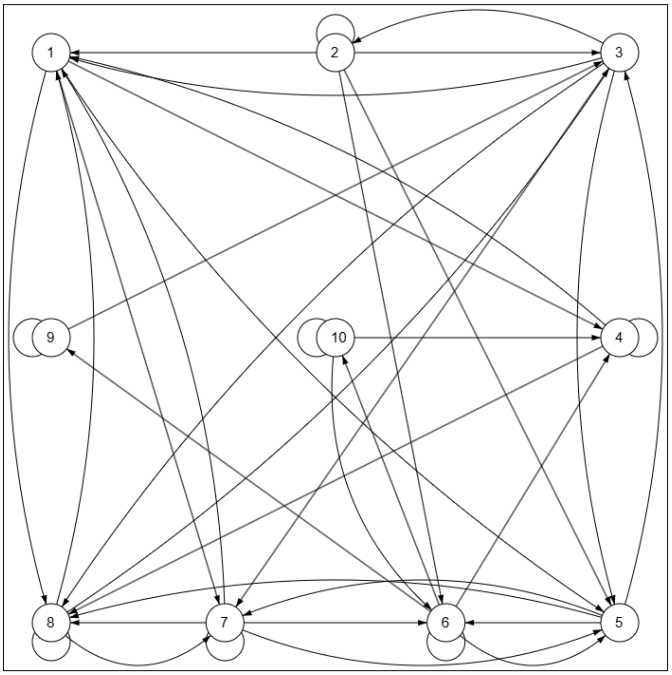
2)

Варіант 4208

Спочатку проклацайте до кінця DFS,потім натисніть clear і проклацайте BFS

DFC clear BFC



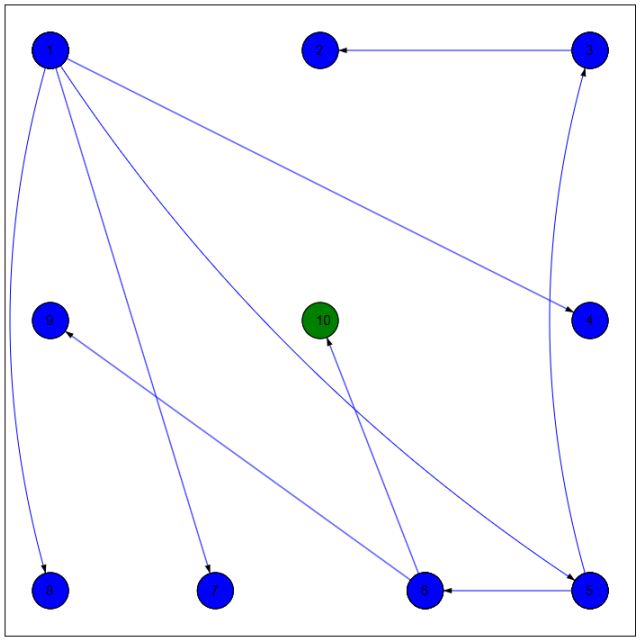


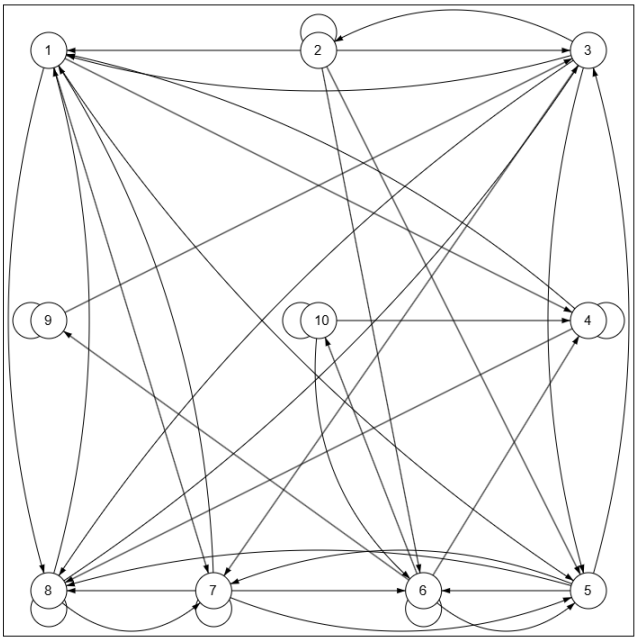
3)

Варіант 4208

Спочатку проклацайте до кінця DFS,потім натисніть clear і проклацайте BFS

DFC clear BFC





Висновок:

В ході виконання лабораторної роботи номер 5 був вивчений метод дослідження графа за допомогою обходу його вершин в глибину та в ширину.

Обхід в глибину виявився простішим в реалізації, надмірно складних частин для виконання робота не мала.

Найдовшою частиною для виконання виявилось складання функцій getDFSorder і getBFSorder