**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 No3

з дисципліни

«Алгоритми і структури даних»

Виконав:
студент групи ІМ-33
Козарезов Кирил Олександрович
номер у списку групи: 12

Перевірила:
Молчанова А. А.

Київ 2024

## Мета лабораторної роботи

Метою лабораторної роботи №3 «Графічне представлення графів» є набуття практичних навичок представлення графів у комп'ютері та ознайомлення з принципами роботи ОС.

## Постановка задачі

1. Представити у програмі напрямлений і ненапрямлений графи з заданими параметрами:
   - кількість вершин $n$;
   - розміщення вершин;
   - матриця суміжності A.
2. Створити програму для формування зображення напрямленого і ненапрямленого графів у графічному вікні.

***Кількість вершин*** $n$ дорівнює $10 + n_3$.

***Розміщення вершин*** трикутником

**Текст програми :**

```
const variant = 3312;
const variantString = variant.toString();
const n1 = parseInt(variantString[0]);
const n2 = parseInt(variantString[1]);
const n3 = parseInt(variantString[2]);
const n4 = parseInt(variantString[3]);

const numTops = n3 + 10;
const coefficient = 1 - n3 * 0.01 - n4 * 0.005 - 0.05;

let matrixDirected = [];
let matrixUndirected = [];

for (let i = 0; i < numTops; i++) {
    let row = [];
    for (let j = 0; j < numTops; j++) {
        let elem = (Math.random() * 2) * coefficient;
        row.push(Math.floor(elem));
    }
    matrixDirected.push(row);
}

for (let i = 0; i < numTops; i++) {
    let row = [];
    for (let j = 0; j < numTops; j++) {
```

```
        row.push(matrixDirected[i][j] || matrixDirected[j][i]);
    }
    matrixUndirected.push(row);
}

console.log("Directed matrix:");
console.log(matrixDirected);
console.log("Undirected matrix:");
console.log(matrixUndirected);

const width = 600;
const height = 700;
const vertexRadius = 30;
const arrOfNode = [];
const arrOfNode2=[];
const arrOfThreeNodes = [
    { x: -200, y: 0 },    // Coordinates for AC
    { x: 0, y: -400 },    // Coordinates for BC
    { x: 200, y: 0 }      // Coordinates for AB
];
const arrOfThreeNodesArrow=[
    { x: -200, y: 0 },    // Coordinates for AC
    { x: 0, y: -400 },    // Coordinates for BC
    { x: 200, y: 0 }
];



//Undirected graph
const canvas = document.getElementById('graphCanvas');
const ctx = canvas.getContext('2d');

canvas.width = width;
canvas.height = height;

function transformCoordinateArea() {
    ctx.translate(width / 2, height / 1.25);
}

function drawCircle(x, y, num, radius) {
    ctx.beginPath();
    ctx.arc(x, y, radius, 0, Math.PI * 2);
    ctx.fillStyle = 'purple';
    ctx.fill();
    ctx.closePath();

    ctx.fillStyle = 'white';
    ctx.font = 'bold 20px Arial';
    ctx.textAlign = 'center';
    ctx.textBaseline = 'middle';

    ctx.fillText(num, x, y);
}

transformCoordinateArea();

function createVertex() {
    const lengthAB = Math.sqrt(((-200 - 200) ** 2) + (0 ** 2));
    const lengthBC = Math.sqrt(((200) ** 2) + (0 - (-400)) ** 2);
    const lengthAC = lengthBC;

    const cosAngle = 200 / lengthAC;
    const sinAngle = Math.sqrt(1 - cosAngle ** 2);

    const dXAC = (lengthAC * cosAngle) / 4;
    const dYAC = (lengthAC * sinAngle) / 4;
```

```
    const dXBC = (lengthAC * cosAngle) / 4;
    const dYBC = (lengthAC * sinAngle) / 4;

    const dXAB = lengthAB / 3;

    let XforAC = -200;
    let YforAC = 0;
    let XforBC = 0;
    let YforBC = -400;
    let XforAB = 200;

    for (let i = 0; i < 4; i++) {
        arrOfNode.push({ x: XforAC, y: YforAC });
        drawCircle(XforAC, YforAC, String.fromCharCode(65 + i), vertexRadius);
// A, B, C, D
        XforAC += dXAC;
        YforAC -= dYAC;
    }

    for (let i = 0; i < 4; i++) {
        arrOfNode.push({ x: XforBC, y: YforBC });
        drawCircle(XforBC, YforBC, String.fromCharCode(69 + i), vertexRadius);
// E, F, G, H
        XforBC += dXBC;
        YforBC += dYBC;
    }

    for (let i = 0; i < 3; i++) {
        arrOfNode.push({ x: XforAB, y: 0 });
        drawCircle(XforAB, 0, String.fromCharCode(73 + i), vertexRadius); // I,
J, K
        XforAB -= dXAB;
    }
}

createVertex();

function drawEdges() {
    for (let i = 0; i < numTops; i++) {
        for (let j = i; j < numTops; j++) {
            if (matrixUndirected[i][j] === 1) {
                if (i === j) {
                    drawSelfLoop(arrOfNode[j]);
                } else {
                    chooseLine(arrOfNode[i].x, arrOfNode[i].y, arrOfNode[j].x,
arrOfNode[j].y);
                }
            }
        }
    }
}

function drawSelfLoop(coordinate) {
    ctx.beginPath();
    ctx.arc(coordinate.x - 45, coordinate.y, 20, Math.PI / 6, (Math.PI * 11) /
6);
    ctx.stroke();
    ctx.closePath();
}

function drawArcLine(start, end, bendAngle = Math.PI / 8) {
    const midX = (start.x + end.x) / 2;
    const midY = (start.y + end.y) / 2;

    let controlX, controlY;
```

```
    if (start.x !== end.x && start.y !== end.y) {
        controlX = midX + Math.cos(bendAngle) * (midY - start.y);
        controlY = midY + Math.sin(bendAngle) * (midX - start.x);
    } else if (start.x === end.x) {
        controlX = midX + 100;
        controlY = midY;
    } else {
        controlX = midX;
        controlY = midY + 100;
    }

    ctx.beginPath();
    ctx.moveTo(start.x, start.y);
    ctx.quadraticCurveTo(controlX, controlY, end.x, end.y);
    ctx.stroke();
    ctx.closePath();
}

function chooseLine(x1, y1, x2, y2) {
    if (checkFunction({ x: x1, y: y1 }, { x: x2, y: y2 })) {
        drawArcLine({ x: x1, y: y1 }, { x: x2, y: y2 });
    } else {
        ctx.beginPath();
        ctx.moveTo(x1, y1);
        ctx.lineTo(x2, y2);
        ctx.stroke();
        ctx.closePath();
    }
}

function checkFunction(start, end) {
    const [A, B, C] = arrOfThreeNodes;

    function isPointOnLineSegment(P, A, B) {
        const crossProduct = (P.y - A.y) * (B.x - A.x) - (P.x - A.x) * (B.y -
A.y);
        if (Math.abs(crossProduct) !== 0) return false;

        const dotProduct = (P.x - A.x) * (B.x - A.x) + (P.y - A.y) * (B.y -
A.y);
        if (dotProduct < 0) return false;

        const squaredLengthBA = (B.x - A.x) ** 2 + (B.y - A.y) ** 2;
        if (dotProduct > squaredLengthBA) return false;

        return true;
    }

    function isOnSameEdge(P1, P2, A, B) {
        return isPointOnLineSegment(P1, A, B) && isPointOnLineSegment(P2, A, B);
    }

    return (
        isOnSameEdge(start, end, A, B) ||
        isOnSameEdge(start, end, B, C) ||
        isOnSameEdge(start, end, C, A)
    );
}

drawEdges();
console.log(arrOfNode);
```

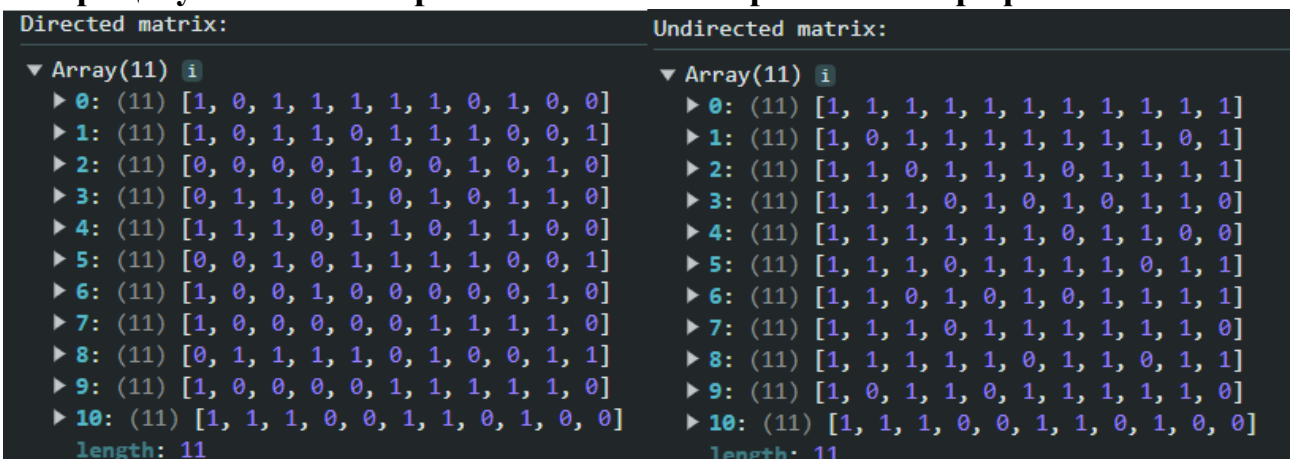
```
//Directed graph
const canvasArrow = document.getElementById('graphCanvasWithArrows');
const ctxArrow = canvasArrow.getContext('2d');
ctxArrow.strokeStyle='black';

const widthArrow = 600;
const heightArrow = 700;
const vertexRadiusArrow = 15;

canvasArrow.width = widthArrow;
canvasArrow.height = heightArrow;

ctxArrow.arc(widthArrow / 2, heightArrow / 2, vertexRadiusArrow, 0, Math.PI *
2);
ctxArrow.fillStyle = "green";

function drawCircleArrow(x, y, num, radius) {
    ctxArrow.beginPath();
    ctxArrow.arc(x, y, radius, 0, Math.PI * 2);
    ctxArrow.fillStyle = 'green';
    ctxArrow.fill();
    ctxArrow.closePath();

    ctxArrow.fillStyle = 'white';
    ctxArrow.font = 'bold 20px Arial';
    ctxArrow.textAlign = 'center';
    ctxArrow.textBaseline = 'middle';

    ctxArrow.fillText(num, x, y);
}
function transformCoordinateAreaArrow() {
    ctxArrow.translate(widthArrow / 2, heightArrow / 1.25);

}
transformCoordinateAreaArrow();
function createVertexArrow() {
    const lengthABArrow = Math.sqrt(((-200 - 200) ** 2) + (0 ** 2));
    const lengthBCArrow = Math.sqrt(((200) ** 2) + (0 - (-400)) ** 2);
    const lengthACArrow = lengthBCArrow;

    const cosAngle = 200 / lengthACArrow;
    const sinAngle = Math.sqrt(1 - cosAngle ** 2);

    // Gap between which the circles (vertices) will be drawn
    const dXACArrow = (lengthACArrow * cosAngle) / 4;
    const dYACArrow = (lengthACArrow * sinAngle) / 4;
    const dXBCArrow = (lengthACArrow * cosAngle) / 4;
    const dYBCArrow = (lengthACArrow * sinAngle) / 4;
    const dXABArrow = lengthABArrow / 3;

    let XforACArrow = -200;
    let YforACArrow = 0;
    let XforBCArrow = 0;
    let YforBCArrow = -400;
    let XforABArrow = 200;
    // Draw vertices A, B, C
    for (let i = 0; i < 4; i++) {
        arrOfNode2.push({ x: XforACArrow, y: YforACArrow });
        drawCircleArrow(XforACArrow, YforACArrow, String.fromCharCode(65 + i),
30); // A, B, C, D
        XforACArrow += dXACArrow;
        YforACArrow -= dYACArrow;
    }
    // Draw vertices E, F, G
    for (let i = 0; i < 4; i++) {
```

```
        arrOfNode2.push({ x: XforBCArrow, y: YforBCArrow });
        drawCircleArrow(XforBCArrow, YforBCArrow, String.fromCharCode(69 + i),
30); // E, F, G, H
        XforBCArrow += dXBCArrow;
        YforBCArrow += dYBCArrow;
    }
    // Draw vertices I, J, K
    for (let i = 0; i < 3; i++) {
        arrOfNode2.push({ x: XforABArrow, y: 0 });
        drawCircleArrow(XforABArrow, 0, String.fromCharCode(73 + i), 30); // I,
J, K
        XforABArrow -= dXABArrow;
    }
}

createVertexArrow();

function drawEdgesArrow() {
    for (let i = 0; i < 11; i++) {
        for (let j = i; j < 11; j++) {
            if (matrixDirected[i][j] === 1 && i===j) {
                drawSelfLoopArrow(arrOfNode2[j]);
            } else if(matrixDirected[i][j] === 1) {
                chooseLineArrow(arrOfNode2[i].x, arrOfNode2[i].y,
arrOfNode2[j].x, arrOfNode2[j].y);
            }

        }
    }
}


function drawSelfLoopArrow(coordinateArrow) {
    const arrowheadSize=10
    ctxArrow.beginPath();
    ctxArrow.arc(coordinateArrow.x - 45, coordinateArrow.y, 20, Math.PI / 6,
(Math.PI * 11) / 6);
    ctxArrow.stroke();
    ctxArrow.closePath();


    ctxArrow.save();
    ctxArrow.translate(coordinateArrow.x-34+ Math.cos((Math.PI * 11) / 6) * (20
- 10), coordinateArrow.y + Math.sin((Math.PI * 11) / 6) * (20 - 10));
    ctxArrow.rotate(45);
    ctxArrow.beginPath();
    ctxArrow.moveTo(0, 0);
    ctxArrow.lineTo(-arrowheadSize, arrowheadSize / 2);
    ctxArrow.lineTo(-arrowheadSize, -arrowheadSize / 2);
    ctxArrow.closePath();
    ctxArrow.fill();
    ctxArrow.restore();
    ctxArrow.fillStyle = 'black';
    ctxArrow.beginPath();
    ctxArrow.arc(coordinateArrow.x-45, coordinateArrow.y , 20, Math.PI / 6,
Math.PI * 11 / 6);
    ctxArrow.stroke();
    ctxArrow.closePath();
}

function drawEdgeLineArrow(start, end, bendAngle = Math.PI / 8, arrowDistance =
20) {
    const arrowSize = 15; // Налаштуйте розмір стрілки за необхідності

    const midXArrow = (start.x + end.x) / 2;
    const midYArrow = (start.y + end.y) / 2;
```

```
    let controlX, controlY;

    if (start.x !== end.x && start.y !== end.y) {
        controlX = midXArrow + Math.cos(bendAngle) * (midYArrow - start.y);
        controlY = midYArrow + Math.sin(bendAngle) * (midXArrow - start.x);
    } else if (start.x === end.x) {
        controlX = midXArrow + 100;
        controlY = midYArrow;
    } else {
        controlX = midXArrow;
        controlY = midYArrow + 100;
    }


    ctxArrow.beginPath();
    ctxArrow.moveTo(start.x, start.y);
    ctxArrow.quadraticCurveTo(controlX, controlY, end.x, end.y);
    ctxArrow.stroke();
    ctxArrow.closePath();


    const angle = Math.atan2(end.y - controlY, end.x - controlX);


    const newEndX = end.x - arrowDistance * Math.cos(angle);
    const newEndY = end.y - arrowDistance * Math.sin(angle);


    ctxArrow.save();
    ctxArrow.translate(newEndX, newEndY);
    ctxArrow.rotate(angle);
    ctxArrow.fillStyle = 'black';
    ctxArrow.beginPath();
    ctxArrow.moveTo(0, 0);
    ctxArrow.lineTo(-arrowSize, arrowSize / 2);
    ctxArrow.lineTo(-arrowSize, -arrowSize / 2);
    ctxArrow.closePath();
    ctxArrow.fill();
    ctxArrow.restore();
}

function chooseLineArrow(x1, y1, x2, y2) {
    if (checkFunctionArrow({ x: x1, y: y1 }, { x: x2, y: y2 })) {
        drawEdgeLineArrow({ x: x1, y: y1 }, { x: x2, y: y2 });
    } else {
        ctxArrow.beginPath();
        ctxArrow.moveTo(x1, y1);
        ctxArrow.lineTo(x2, y2);
        ctxArrow.stroke();
        ctxArrow.closePath();


        const angle = Math.atan2(y2 - y1, x2 - x1);


        const arrowSize = 12;
        const gapX = Math.cos(angle) * 10;
        const gapY = Math.sin(angle) * 10;


        ctxArrow.beginPath();
        ctxArrow.fillStyle = 'black';
        ctxArrow.moveTo(x2 - gapX, y2 - gapY);
        ctxArrow.lineTo(
            x2 - arrowSize * Math.cos(angle - Math.PI / 6) - gapX,
```

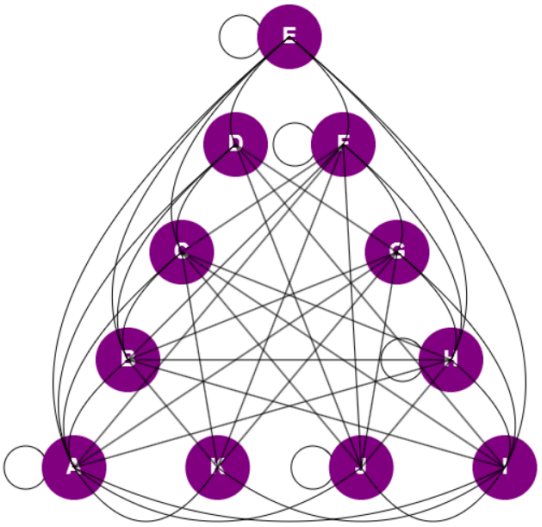
```
            y2 - arrowSize * Math.sin(angle - Math.PI / 6) - gapY
        );
        ctxArrow.lineTo(
            x2 - arrowSize * Math.cos(angle + Math.PI / 6) - gapX,
            y2 - arrowSize * Math.sin(angle + Math.PI / 6) - gapY
        );
        ctxArrow.closePath();
        ctxArrow.fill();
    }
}

function checkFunctionArrow(start, end) {
    const [A, B, C] = arrOfThreeNodesArrow;

    function isPointOnLineSegmentArrow(P, A, B) {
        const crossProductArrow = (P.y - A.y) * (B.x - A.x) - (P.x - A.x) * (B.y
- A.y);
        if (Math.abs(crossProductArrow) !== 0) return false;

        const dotProductArrow = (P.x - A.x) * (B.x - A.x) + (P.y - A.y) * (B.y -
A.y);
        if (dotProductArrow < 0) return false;

        const squaredLengthBA = (B.x - A.x) ** 2 + (B.y - A.y) ** 2;
        if (dotProductArrow > squaredLengthBA) return false;

        return true;
    }

    function isOnSameEdgeArrow(P1, P2, A, B) {
        return isPointOnLineSegmentArrow(P1, A, B) &&
isPointOnLineSegmentArrow(P2, A, B);
    }

    return (
        isOnSameEdgeArrow(start, end, A, B) ||
        isOnSameEdgeArrow(start, end, B, C) ||
        isOnSameEdgeArrow(start, end, C, A)
    );
}

drawEdgesArrow();
```

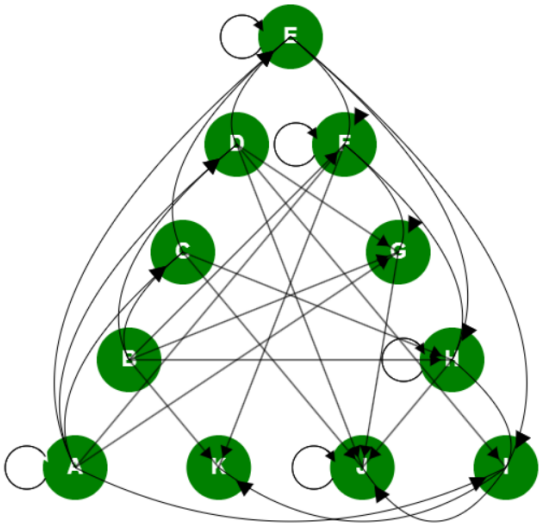
**Матриці суміжності напрямленого та ненапрямленого графів:**




*Матриця напрямленого графа* *Матриця ненапрямленого графа*

**Зображення ненапрямленого та напрямленого графів:**




**Висновок:**

Під час виконання лабораторної роботи я освоїв концепцію графів та навчився візуалізувати їх за допомогою мови JavaScript. Результати виконання лабораторної роботи показали, що графи  є потужним інструментом для представлення різноманітних зв’язків та сутностей.