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 №6

з дисципліни
«Алгоритми і структури даних»

Виконала:
студентка групи ІМ-41
Куц Анна Василівна
номер у списку групи: 11

Перевірив:
Сергієнко А.М.

Київ 2025

Постановка задачі

1. Представити зважений ненапрямлений граф із заданими параметрами так само, як у лабораторній роботі №3.

   Відмінність 1: коефіцієнт $k = 1.0 - n_3 * 0.01 - n_4 * 0.005 - 0.05$.

   Отже, матриця суміжності A_{dir} напрямленого графа за варіантом формується таким чином:

   1) встановлюється параметр (seed) генератора випадкових чисел, рівне номеру варіанту $n_1 n_2 n_3 n_4$;
   2) матриця розміром $n \cdot n$ заповнюється згенерованими випадковими числами в діапазоні $[0, 2.0)$;
   3) обчислюється коефіцієнт $k = 1.0 - n_3 * 0.01 - n_4 * 0.005 - 0.05$, 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.

   Матриця A_{undir} ненапрямленого графа одержується з матриці A_dir так само, як у ЛР №3.

   Відмінність 2: матриця ваг W формується таким чином.

   1) матриця B розміром $n \cdot n$ заповнюється згенерованими випадковими числами в діапазоні $[0, 2.0)$ (параметр генератора випадкових чисел той же самий, $n_1 n_2 n_3 n_4$);
   2) одержується матриця C:
      $c_{ij} = \text{ceil}(b_{ij} \cdot 100 \cdot a_{undir_{i,j}}), \qquad c_{i,j} \in C,\ b_{ij} \in B,\ a_{undir_{i,j}} \in A_{undir},$
      де ceil — це функція, що округляє кожен елемент матриці до найближчого цілого числа, більшого чи рівного за дане;
   3) одержується матриця D, у якій
      $d_{ij} = 0$, якщо $c_{ij} = 0$,
      $d_{ij} = 1$, якщо $c_{ij} > 0, \quad d_{ij} \in D, c_{ij} \in C$;
   4) одержується матриця H, у якій
      $h_{ij} = 1$, якщо $d_{ij} \neq d_{ji}$,
      та $h_{ij} = 0$ в іншому випадку;
   5) Tr — верхня трикутна матриця з одиниць ($tr_{ij} = 1$ при $i < j$);
   6) матриця ваг W симетрична, і її елементи одержуються за формулою: $w_{ij} = w_{ji} = (d_{ij} + h_{ij} \cdot tr_{ij}) \cdot c_{ij}$.

2. Створити програму для знаходження мінімального кістяка за алгоритмом Краскала при n_4 — парному і за алгоритмом Пріма — при непарному. При цьому у програмі:

   - графи представляти у вигляді динамічних списків, обхід графа, додавання, віднімання вершин, ребер виконувати як функції з вершинами відповідних списків;
   - у програмі виконання обходу відображати покроково, черговий крок виконувати за натисканням кнопки у вікні або на клавіатурі.

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

   - або виділяти іншим кольором ребра графа;
   - або будувати кістяк поряд із графом.

Варіант №4111
$n_1n_2n_3n_4 = 4111$
k = 1.0 - n3 * 0.01 - n4 * 0.005 - 0.05 = 1.0 - 1 * 0.01 - 1 * 0.005 - 0.05 = 0.935
Розміщення колом. Кількість вершин = 10 + n_3 = 11. Алгоритм Пріма.

Алгоритм було створено мовою програмування JavaScript з використанням математичних бібліотек та вбудовану графічну бібліотеку <canvas>, для відображення використовувався HTML та CSS.

Текст програми
index. html

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Lab6</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
<h1>Variant 4111🎀</h1>

<div>
    <button id="btnDirected">Directed</button>
    <button id="btnUndirected">Undirected</button>
```

```
    <button id="btnWeighted">Weighted</button>
    <button id="btnNext" disabled>Next</button>
    <button id="btnMST">Prim</button>
</div>
<canvas id="canvas" width="800" height="800"></canvas>

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

styles.css

```
body {
    font-family: "Times New Roman";
    display: flex;
    flex-direction: column;
    align-items: center;
    background: #ffe6f2;
}

canvas {
    border: 1px solid #ccc;
    background: #fff;
    margin-top: 1rem;
}

h1 {
    color: hotpink;
}

button {
    margin: 0.5rem;
    padding: 0.5rem 1rem;
    font-size: 16px;
    cursor: pointer;
    background: hotpink;
    border: none;
    border-radius: 10px;
    color: white;
    font-weight: bold;
    transition: background 0.2s, transform 0.2s;
}

button:hover {
    background: #ff69b4;
    transform: scale(1.05);
}
```

script.js

```
document.addEventListener("DOMContentLoaded", () => {
    const canvas = document.getElementById("canvas");
    const ctx = canvas.getContext("2d");

    const seed = 4111;
    const n3 = 1;
    const n4 = 1;
    const n = 10 + n3;
    const k = 1.0 - n3 * 0.01 - n4 * 0.005 - 0.05;

    function genRand(seed) {
        const MOD = 2147483647;
        let val = seed % MOD;
        return function () {
            val = (val * 16807) % MOD;
            return (val - 1) / MOD;
        };
    }

    const rand = genRand(seed);

    function genDirMatrix() {
        const raw = Array.from({length: n}, () =>
            Array.from({length: n}, () => rand() * 2)
        );

        const dir = raw.map(row => row.map(v => (v * k >= 1 ? 1 :
0)));

        for (let i = 0; i < n; i++) {
            for (let j = i + 1; j < n; j++) {
                if (dir[i][j] && dir[j][i]) {
                    if (rand() < 0.5) dir[i][j] = 0;
                    else dir[j][i] = 0;
                }
            }
        }

        return dir;
    }

    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] || dir[j][i]) {
                    undir[i][j] = undir[j][i] = 1;
                }
            }
        }
        return undir;
```

```
    }

    function genMatrixB() {
        return Array.from({length: n}, () =>
            Array.from({length: n}, () => rand() * 2)
        );
    }

    function createMatrixC(B, Aundir) {
        return Array.from({length: n}, (_, i) =>
            Array.from({length: n}, (_, j) =>
                Math.ceil(B[i][j] * 100 * Aundir[i][j])
            )
        );
    }

    function createMatrixD(C) {
        return Array.from({length: n}, (_, i) =>
            Array.from({length: n}, (_, j) => C[i][j] > 0 ? 1 : 0)
        );
    }

    function createMatrixH(D) {
        return Array.from({length: n}, (_, i) =>
            Array.from({length: n}, (_, j) => D[i][j] !== D[j][i] ?
1 : 0)
        );
    }

    function createMatrixTr() {
        return Array.from({length: n}, (_, i) =>
            Array.from({length: n}, (_, j) => i < j ? 1 : 0)
        );
    }

    function createMatrixW(D, H, Tr, C) {
        const W = 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) {
                    const weight = (D[i][j] + H[i][j] * Tr[i][j]) *
C[i][j];
                    W[i][j] = W[j][i] = weight;
                }
            }
        }
        return W;
    }

    function getAllEdges(W) {
        const edges = [];
        for (let i = 0; i < n; i++) {
            for (let j = i + 1; j < n; j++) {
                if (W[i][j] > 0) {
```

```
                    edges.push({
                        from: i,
                        to: j,
                        weight: W[i][j]
                    });
                }
            }
        }
        return edges.sort((a, b) => a.weight - b.weight);
    }

    function primMSTSteps(W) {
        const n = W.length;
        const selected = Array(n).fill(false);
        const parent = Array(n).fill(-1);
        const key = Array(n).fill(Infinity);
        const steps = [];

        key[0] = 0;
        for (let count = 0; count < n; count++) {
            let minVal = Infinity;
            let minIndex = -1;

            for (let v = 0; v < n; v++) {
                if (!selected[v] && key[v] < minVal) {
                    minVal = key[v];
                    minIndex = v;
                }
            }

            if (minIndex === -1) break;

            const u = minIndex;
            selected[u] = true;
            if (parent[u] !== -1) {
                steps.push({
                    from: parent[u],
                    to: u,
                    weight: W[u][parent[u]],
                    selected: [...selected],
                    currentStep: count
                });
            }

            for (let v = 0; v < n; v++) {
                if (W[u][v] && !selected[v] && W[u][v] < key[v]) {
                    parent[v] = u;
                    key[v] = W[u][v];
                }
            }
        }

        return {
            steps: steps,
```

```
            parent: parent
        };
    }

    function primMST(W) {
        const n = W.length;
        const selected = Array(n).fill(false);
        const parent = Array(n).fill(-1);
        const key = Array(n).fill(Infinity);

        key[0] = 0;

        for (let count = 0; count < n - 1; count++) {
            let minVal = Infinity;
            let minIndex = -1;

            for (let v = 0; v < n; v++) {
                if (!selected[v] && key[v] < minVal) {
                    minVal = key[v];
                    minIndex = v;
                }
            }

            const u = minIndex;
            selected[u] = true;

            for (let v = 0; v < n; v++) {
                if (W[u][v] && !selected[v] && W[u][v] < key[v]) {
                    parent[v] = u;
                    key[v] = W[u][v];
                }
            }
        }

        return parent;
    }

    function createMSTMatrix(parent) {
        const mstMatrix = Array.from({length: n}, () =>
Array(n).fill(0));
        for (let i = 1; i < n; i++) {
            if (parent[i] !== -1) {
                mstMatrix[i][parent[i]] = mstMatrix[parent[i]][i] =
1;
            }
        }
        return mstMatrix;
    }

    function createPartialMSTMatrix(steps, currentStep) {
        const mstMatrix = Array.from({length: n}, () =>
Array(n).fill(0));
        for (let i = 0; i <= currentStep && i < steps.length; i++)
{
```

```
            const {from, to} = steps[i];
            mstMatrix[from][to] = mstMatrix[to][from] = 1;
        }
        return mstMatrix;
    }

    function printMatrix(matrix, title) {
        console.log(`\n${title}:`);
        matrix.forEach(row => console.log(row.join(" ")));
    }

    function printEdges(edges, title) {
        console.log(`\n${title}:`);
        edges.forEach(edge =>
            console.log(`Ребро ${edge.from + 1}-${edge.to + 1},
вага: ${edge.weight}`)
        );
    }

    function calculateMSTWeight(parent, weightMatrix) {
        let totalWeight = 0;
        for (let i = 1; i < n; i++) {
            if (parent[i] !== -1) {
                totalWeight += weightMatrix[i][parent[i]];
            }
        }
        return totalWeight;
    }

    function printMSTInfo(parent, weightMatrix) {
        let totalWeight = 0;
        console.log("\nРебра кістяка:");
        for (let i = 1; i < n; i++) {
            if (parent[i] !== -1) {
                console.log(`Ребро ${parent[i] + 1}-${i + 1}, вага:
${weightMatrix[i][parent[i]]}`);
                totalWeight += weightMatrix[i][parent[i]];
            }
        }
        console.log(`Загальна вага: ${totalWeight}`);
        return totalWeight;
    }

    const w = canvas.width;
    const h = canvas.height;
    const RAD = 20;
    const centerX = w / 2;
    const centerY = h / 2;
    const radius = 280;

    const positions = Array.from({length: n}, (_, i) => {
        const angle = (2 * Math.PI * i) / n - Math.PI / 2;
        return {
            x: centerX + radius * Math.cos(angle),
```

```
            y: centerY + radius * Math.sin(angle),
            index: i
        };
    });

    function distanceToLine(p1, p2, p) {
        const A = p.x - p1.x, B = p.y - p1.y;
        const C = p2.x - p1.x, D = p2.y - p1.y;
        const dot = A * C + B * D, len2 = C * C + D * D;
        const param = dot / len2;

        const xx = param < 0 ? p1.x : param > 1 ? p2.x : p1.x +
param * C;
        const yy = param < 0 ? p1.y : param > 1 ? p2.y : p1.y +
param * D;

        const dx = p.x - xx, dy = p.y - yy;
        return Math.hypot(dx, dy);
    }

    function calculateControlPoint(p1, p2) {
        const mid = {x: (p1.x + p2.x) / 2, y: (p1.y + p2.y) / 2};
        const perp = {x: -(p2.y - p1.y), y: p2.x - p1.x};
        const len = Math.hypot(perp.x, perp.y);
        const dirSign = p1.index < p2.index ? 1 : -1;
        return {
            x: mid.x + dirSign * (perp.x / len) * 90,
            y: mid.y + dirSign * (perp.y / len) * 90
        };
    }

    function needsCurvedEdge(p1, p2, positions) {
        for (let k = 0; k < positions.length; k++) {
            if (k === p1.index || k === p2.index) continue;
            if (distanceToLine(p1, p2, positions[k]) < 25) {
                return true;
            }
        }
        return false;
    }

    function drawArrow(p1, p2, cp = null, offsetForWeight = false)
{
        let angle, arrowStart;

        if (cp) {
            const t = offsetForWeight ? 0.88 : 0.95;
            const x = (1 - t) ** 2 * p1.x + 2 * (1 - t) * t * cp.x
+ t ** 2 * p2.x;
            const y = (1 - t) ** 2 * p1.y + 2 * (1 - t) * t * cp.y
+ t ** 2 * p2.y;
            const dx = 2 * (1 - t) * (cp.x - p1.x) + 2 * t * (p2.x
- cp.x);
            const dy = 2 * (1 - t) * (cp.y - p1.y) + 2 * t * (p2.y
```

```
- cp.y);
            angle = Math.atan2(dy, dx);
            arrowStart = {x, y};
        } else {
            const dx = p2.x - p1.x, dy = p2.y - p1.y;
            angle = Math.atan2(dy, dx);
            const offset = offsetForWeight ? RAD * 1.5 : RAD;
            arrowStart = {
                x: p2.x - offset * Math.cos(angle),
                y: p2.y - offset * Math.sin(angle)
            };
        }

        ctx.beginPath();
        ctx.moveTo(arrowStart.x, arrowStart.y);
        ctx.lineTo(
            arrowStart.x - 10 * Math.cos(angle - Math.PI / 8),
            arrowStart.y - 10 * Math.sin(angle - Math.PI / 8)
        );
        ctx.lineTo(
            arrowStart.x - 10 * Math.cos(angle + Math.PI / 8),
            arrowStart.y - 10 * Math.sin(angle + Math.PI / 8)
        );
        ctx.closePath();
        ctx.fill();
    }

    function drawWeightLabel(x, y, weight, color = "#333") {
        ctx.font = "bold 14px Times New Roman";
        ctx.textAlign = "center";
        ctx.textBaseline = "middle";

        const textWidth = ctx.measureText(weight.toString()).width;

        ctx.fillStyle = "#fff";
        ctx.fillRect(x - textWidth/2 - 5, y - 10, textWidth + 10,
20);

        ctx.strokeStyle = color;
        ctx.lineWidth = 1;
        ctx.strokeRect(x - textWidth/2 - 5, y - 10, textWidth + 10,
20);

        ctx.fillStyle = color;
        ctx.fillText(weight.toString(), x, y);
    }

    function drawSelfLoop(nodeX, nodeY, directed) {
        const arcR = RAD * 0.75;
        const offset = RAD + 10;

        const dx = nodeX - centerX;
        const dy = nodeY - centerY;
        let theta = Math.atan2(dy, dx) * 180 / Math.PI;
```

```
        if (theta < 0) theta += 360;

        let cx, cy, start, end;
        if (theta >= 315 || theta < 45) {
            cx = nodeX + offset; cy = nodeY;
            start = -135 * Math.PI / 180; end = 135 * Math.PI /
180;
        } else if (theta >= 45 && theta < 135) {
            cx = nodeX; cy = nodeY + offset;
            start = 225 * Math.PI / 180; end = 135 * Math.PI / 180;
        } else if (theta >= 135 && theta < 225) {
            cx = nodeX - offset; cy = nodeY;
            start = 45 * Math.PI / 180; end = -45 * Math.PI / 180;
        } else {
            cx = nodeX; cy = nodeY - offset;
            start = 45 * Math.PI / 180; end = -45 * Math.PI / 180;
        }

        ctx.beginPath();
        ctx.arc(cx, cy, arcR, start, end, false);
        ctx.stroke();

        if (!directed) return;

        const ax = cx + arcR * Math.cos(end);
        const ay = cy + arcR * Math.sin(end);
        const arrowAngle = Math.atan2(nodeY - ay, nodeX - ax);
        const L = 0.55 * arcR;

        ctx.beginPath();
        ctx.moveTo(ax, ay);
        ctx.lineTo(
            ax - L * Math.cos(arrowAngle - Math.PI / 6),
            ay - L * Math.sin(arrowAngle - Math.PI / 6)
        );
        ctx.lineTo(
            ax - L * Math.cos(arrowAngle + Math.PI / 6),
            ay - L * Math.sin(arrowAngle + Math.PI / 6)
        );
        ctx.closePath();
        ctx.fill();
    }

    function drawEdge(p1, p2, options) {
        const { weight = null, isMstEdge = false, curved = false,
            cp = null, isHighlighted = false, directed = false } =
options;

        ctx.save();

        const color = isHighlighted ? "#FF5722" : isMstEdge ?
"hotpink" : "#333";
        ctx.strokeStyle = color;
        ctx.lineWidth = isHighlighted ? 4 : isMstEdge ? 3 : 1;
```

```
        ctx.beginPath();
        ctx.moveTo(p1.x, p1.y);
        if (curved && cp) {
            ctx.quadraticCurveTo(cp.x, cp.y, p2.x, p2.y);
        } else {
            ctx.lineTo(p2.x, p2.y);
        }
        ctx.stroke();

        if (weight !== null) {
            const weightX = curved && cp ? cp.x : (p1.x + p2.x) /
2;
            const weightY = curved && cp ? cp.y : (p1.y + p2.y) /
2;
            drawWeightLabel(weightX, weightY, weight, color);
        }

        if (directed) {
            drawArrow(p1, p2, curved ? cp : null, weight !== null);
        }

        ctx.restore();
    }

    function drawVertex(x, y, index, isSelected) {
        ctx.beginPath();
        ctx.fillStyle = isSelected ? "hotpink" : "#fff";
        ctx.arc(x, y, RAD, 0, 2 * Math.PI);
        ctx.fill();

        if (!isSelected) {
            ctx.stroke();
        }

        ctx.fillStyle = isSelected ? "#fff" : "#000";
        ctx.font = "14px Times New Roman";
        ctx.textAlign = "center";
        ctx.textBaseline = "middle";
        ctx.fillText(index + 1, x, y);
    }

    function isVertexInMST(vertex, mst) {
        if (!mst) return false;
        if (vertex === 0) return true;

        for (let i = 0; i < mst.length; i++) {
            if (mst[i] === vertex) return true;
        }

        return mst[vertex] !== -1;
    }

    function renderGraph(options) {
```

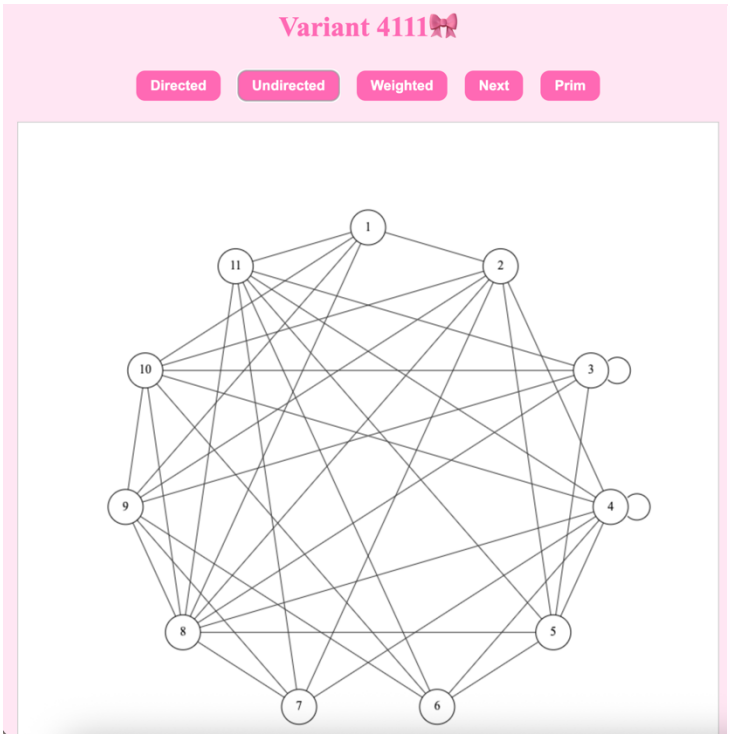
```
        const { matrix, directed = false, withWeights = false,
            mst = null, step = null, partialMatrix = null,
            infoText = null, infoColor = null } = options;

        ctx.clearRect(0, 0, w, h);

        for (let i = 0; i < n; i++) {
            if (matrix[i][i]) {
                drawSelfLoop(positions[i].x, positions[i].y,
directed);
            }
        }

        for (let i = 0; i < n; i++) {
            for (let j = directed ? 0 : i + 1; j < n; j++) {
                if (i === j || !matrix[i][j]) continue;

                const p1 = positions[i], p2 = positions[j];

                let curved = false, cp = null;
                if (needsCurvedEdge(p1, p2, positions)) {
                    curved = true;
                    cp = calculateControlPoint(p1, p2);
                }

                const isMstEdge = partialMatrix ?
                    partialMatrix[i][j] === 1 :
                    mst && (mst[j] === i || mst[i] === j);

                const isHighlighted = step &&
                    ((step.from === i && step.to === j) ||
                        (step.from === j && step.to === i));

                drawEdge(p1, p2, {
                    weight: withWeights ? weightMatrix[i][j] :
null,
                    isMstEdge,
                    curved,
                    cp,
                    isHighlighted,
                    directed
                });
            }
        }

        for (let i = 0; i < n; i++) {
            let isSelected = false;
            if (step) {
                isSelected = step.selected[i];
            } else if (mst) {
                isSelected = isVertexInMST(i, mst);
            }

            drawVertex(positions[i].x, positions[i].y, i,
```

```
isSelected);
        }
    }

    let mstParent = null;
    let weightMatrix = null;
    let mstSteps = [];
    let currentMSTStepIndex = -1;
    let isMSTVisualizationReady = false;

    const dirMatrix = genDirMatrix();
    const undirMatrix = genUndirMatrix(dirMatrix);
    const B = genMatrixB();
    const C = createMatrixC(B, undirMatrix);
    const D = createMatrixD(C);
    const H = createMatrixH(D);
    const Tr = createMatrixTr();
    weightMatrix = createMatrixW(D, H, Tr, C);

    document.getElementById("btnDirected").onclick = () => {
        console.clear();
        printMatrix(dirMatrix, "Directed Matrix (Adir)");

        renderGraph({
            matrix: dirMatrix,
            directed: true
        });

        document.getElementById("btnNext").disabled = true;
        isMSTVisualizationReady = false;
        document.getElementById("btnNext").textContent = "Next";
    };

    document.getElementById("btnUndirected").onclick = () => {
        console.clear();
        printMatrix(undirMatrix, "Undirected Matrix (Aundir)");

        renderGraph({
            matrix: undirMatrix
        });

        document.getElementById("btnNext").disabled = true;
        isMSTVisualizationReady = false;
        document.getElementById("btnNext").textContent = "Next";
    };

    document.getElementById("btnWeighted").onclick = () => {
        console.clear();
        printMatrix(weightMatrix, "Матриця ваг (W)");

        const allEdges = getAllEdges(weightMatrix);
        printEdges(allEdges, "Усі ребра графа (відсортовані за
вагою)");
```

```
        const mstResult = primMSTSteps(weightMatrix);
        mstSteps = mstResult.steps;
        mstParent = mstResult.parent;
        currentMSTStepIndex = -1;
        isMSTVisualizationReady = true;

        renderGraph({
            matrix: undirMatrix,
            withWeights: true
        });

        document.getElementById("btnNext").disabled = false;
        document.getElementById("btnNext").textContent = "Next";

        console.log("\nНатисніть кнопку 'Next' для покрокової
візуалізації алгоритму Прима");
    };

    document.getElementById("btnNext").onclick = () => {
        if (!isMSTVisualizationReady) return;

        currentMSTStepIndex++;

        if (currentMSTStepIndex < mstSteps.length) {
            const step = mstSteps[currentMSTStepIndex];
            const partialMatrix = createPartialMSTMatrix(mstSteps,
currentMSTStepIndex);

            renderGraph({
                matrix: undirMatrix,
                withWeights: true,
                step,
                partialMatrix
            });

            console.log(`Крок ${currentMSTStepIndex + 1}: Додаємо
ребро ${step.from + 1}-${step.to + 1} з вагою ${step.weight}`);
        } else if (currentMSTStepIndex === mstSteps.length) {
            const mstMatrix = createMSTMatrix(mstParent);
            const totalWeight = calculateMSTWeight(mstParent,
weightMatrix);

            renderGraph({
                matrix: mstMatrix,
                withWeights: true,
                mst: mstParent,
                infoText: `Загальна вага кістяка: ${totalWeight}`,
                infoColor: "hotpink"
            });

            console.log(`\nЗагальна вага: ${totalWeight}`);
            console.log("\nРебра кістяка:");
            for (let i = 1; i < n; i++) {
                if (mstParent[i] !== -1) {
```

```
                    console.log(`Ребро ${mstParent[i] + 1}-${i +
1}, вага: ${weightMatrix[i][mstParent[i]]}`);
                }
            }

            document.getElementById("btnNext").textContent =
"Reset";
        } else {
            renderGraph({
                matrix: undirMatrix,
                withWeights: true
            });

            currentMSTStepIndex = -1;
            document.getElementById("btnNext").textContent =
"Next";
        }
    };

    document.getElementById("btnMST").onclick = () => {
        console.clear();
        printMatrix(weightMatrix, "Матриця ваг (W)");
        mstParent = primMST(weightMatrix);

        const mstMatrix = createMSTMatrix(mstParent);
        printMatrix(mstMatrix, "Матриця кістяка");

        const allEdges = getAllEdges(weightMatrix);
        printEdges(allEdges, "Усі ребра графа (відсортовані за
вагою)");

        const totalWeight = printMSTInfo(mstParent, weightMatrix);

        renderGraph({
            matrix: mstMatrix,
            withWeights: true,
            mst: mstParent,
            infoText: `Загальна вага кістяка: ${totalWeight}`,
            infoColor: "hotpink"
        });

        document.getElementById("btnNext").disabled = true;
        isMSTVisualizationReady = false;
        document.getElementById("btnNext").textContent = "Next";
    };

    document.getElementById("btnDirected").click();
});
```

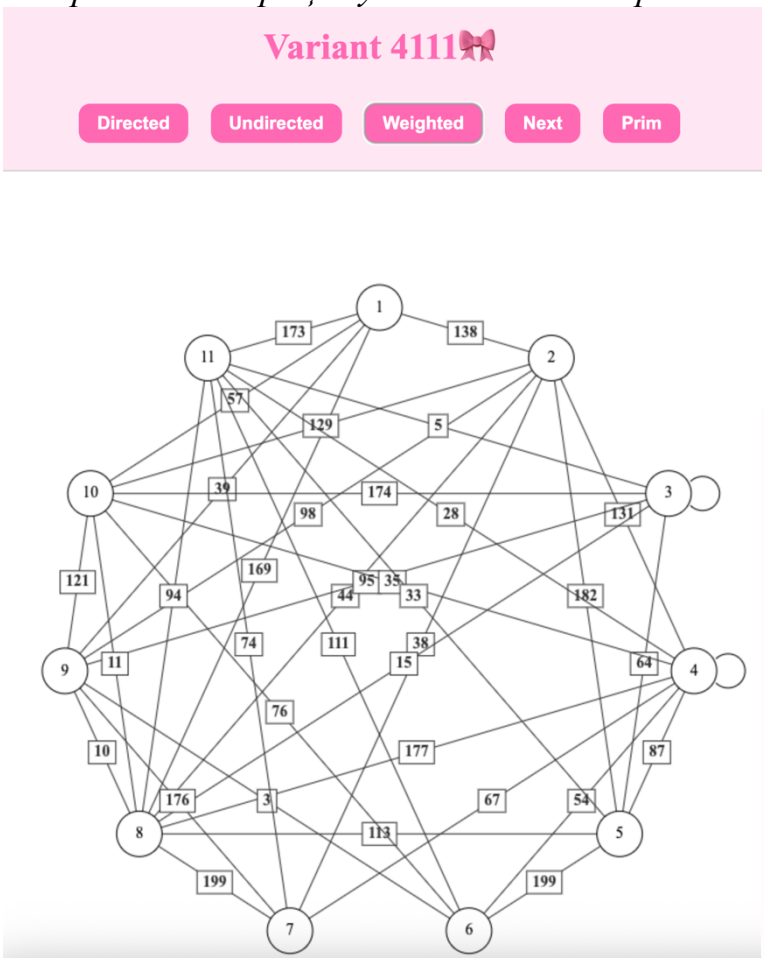
Результати тестування програм



Undirected Matrix (Aundir):

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

Згенерована матриця суміжності ненапрямленого графа



Матриця ваг (W):

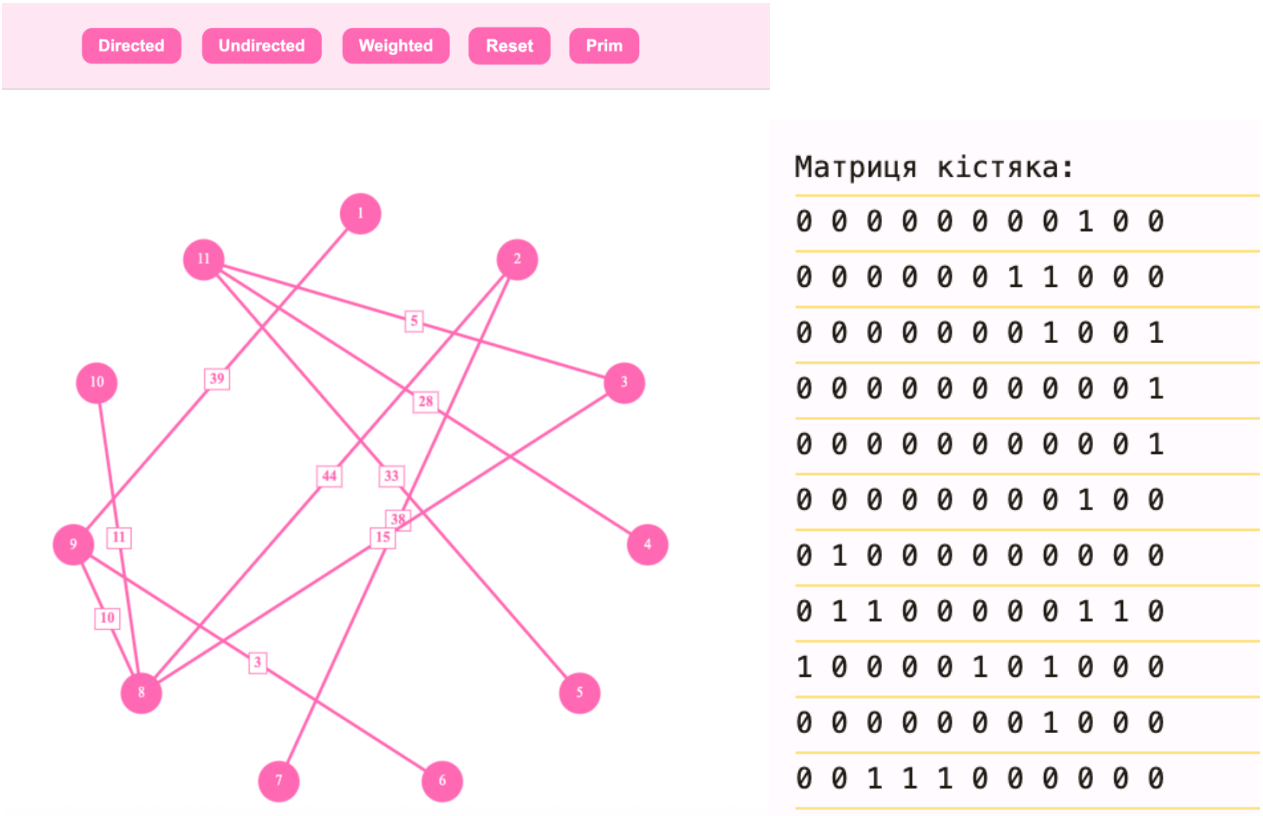
0	138	0	0	0	0	0	169	39	57	173
138	0	0	131	182	0	38	44	98	129	0
0	0	0	0	64	0	0	15	95	174	5
0	131	0	0	87	54	67	177	0	35	28
0	182	64	87	0	199	0	113	0	0	33
0	0	0	54	199	0	0	0	3	76	111
0	38	0	67	0	0	0	199	176	0	74
169	44	15	177	113	0	199	0	10	11	94
39	98	95	0	0	3	176	10	0	121	0
57	129	174	35	0	76	0	11	121	0	0
173	0	5	28	33	111	74	94	0	0	0

Згенерована матриця ваг графа та його зображення

```
Усі ребра графа (відсортовані за вагою):
Ребро 6–9, вага: 3
Ребро 3–11, вага: 5
Ребро 8–9, вага: 10
Ребро 8–10, вага: 11
Ребро 3–8, вага: 15
Ребро 4–11, вага: 28
Ребро 5–11, вага: 33
Ребро 4–10, вага: 35
Ребро 2–7, вага: 38
Ребро 1–9, вага: 39
Ребро 2–8, вага: 44
Ребро 4–6, вага: 54
Ребро 1–10, вага: 57
Ребро 3–5, вага: 64
Ребро 4–7, вага: 67
Ребро 7–11, вага: 74
Ребро 6–10, вага: 76
Ребро 4–5, вага: 87
Ребро 8–11, вага: 94
Ребро 3–9, вага: 95
Ребро 2–9, вага: 98
Ребро 6–11, вага: 111
Ребро 5–8, вага: 113
Ребро 9–10, вага: 121
Ребро 2–10, вага: 129
Ребро 2–4, вага: 131
Ребро 1–2, вага: 138
Ребро 1–8, вага: 169
Ребро 1–11, вага: 173
Ребро 3–10, вага: 174
Ребро 7–9, вага: 176
Ребро 4–8, вага: 177
Ребро 2–5, вага: 182
Ребро 5–6, вага: 199
```

```
Натисніть кнопку 'Next' для покрокової візуалізації алгоритму Прима
Крок 1: Додаємо ребро 1–9 з вагою 39
Крок 2: Додаємо ребро 9–6 з вагою 3
Крок 3: Додаємо ребро 9–8 з вагою 10
Крок 4: Додаємо ребро 8–10 з вагою 11
Крок 5: Додаємо ребро 8–3 з вагою 15
Крок 6: Додаємо ребро 3–11 з вагою 5
Крок 7: Додаємо ребро 11–4 з вагою 28
Крок 8: Додаємо ребро 11–5 з вагою 33
Крок 9: Додаємо ребро 8–2 з вагою 44
Крок 10: Додаємо ребро 2–7 з вагою 38

Загальна вага: 226

Ребра кістяка:
Ребро 8–2, вага: 44
Ребро 8–3, вага: 15
Ребро 11–4, вага: 28
Ребро 11–5, вага: 33
Ребро 9–6, вага: 3
Ребро 2–7, вага: 38
Ребро 9–8, вага: 10
Ребро 1–9, вага: 39
Ребро 8–10, вага: 11
Ребро 3–11, вага: 5
```

Ваги ребер, обчислені під час підрахунків



Зображення мінімального кістяка графа та його матриця

Висновок:

Під час виконання лабораторної роботи №6 «Мінімальний кістяк графа» є вивчила методи розв’язання задачі знаходження мінімального кістяка графа та застосувала їх на практиці. За моїм варіантом я реалізувала алгоритм Пріма. Прім стартує з довільної вершини і поступово додає нові вершини через найменше ребро, яке з’єднує вже включені у дерево вершини з тими, що ще не включені. У той час як Краскала працює з усіма ребрами одразу: він сортує всі ребра за вагою і послідовно додає до дерева ті, які не створюють циклу, тобто об'єднує множини вершин. Пріма в моєму коді реалізує primMSTSteps, яка використовує масиви key, parent, selected і на кожному кроці обирає ту вершину, яка має найменший ключ серед невідвіданих. Покрокова візуалізація додає на canvas одне ребро за раз, підсвічує його, і поступово формується дерево. Кнопки "Weighted", "Prim", "Next" дозволяють ініціалізувати граф із зображеними вагами, і далі натискати "Next", щоб бачити послідовне додавання ребер. Ця поведінка відповідає ідеї алгоритму Пріма: ми не розглядаємо всі ребра відразу, як це робить Краскала, а щоразу обираємо локально найкраще ребро для зростаючого дерева. Матриця ваг створена за визначеним алгоритмом, поданий у вимогах лабораторної роботи.