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

Лабораторна робота №3

з дисципліни
«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 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.

Згадані вище параметри графа задаються на основі чотиризначного номера варіанту n1n2n3n4, де n1n2 це десяткові цифри номера групи,

а n3n4 — десяткові цифри номера варіанту, який був у студента для двох попередніх робіт.

Кількість вершин n дорівнює 10 + n3.

Розміщення вершин:

• колом при n4 = 0, 1;

• квадратом (прямокутником) при n4 = 2, 3;

• трикутником при n4 = 4, 5;

• колом з вершиною в центрі при n4 = 6, 7;

• квадратом (прямокутником) з вершиною в центрі при n4 = 8, 9.

Матриця суміжності Adir напрямленого графа за варіантом формується таким чином:

1) встановлюється параметр (seed) генератора випадкових чисел, рівний номеру варіанту n1n2n3n4 — детальніше див. с. 12;

2) матриця розміром n n заповнюється згенерованими випадковими числами в діапазоні [0, 2.0);

3) обчислюється коефіцієнт k 1.0 n3 0.02 n4 0.005 0.25;

4) кожен елемент матриці множиться на коефіцієнт k;

5) елементи матриці округлюються: 0 — якщо елемент менший за 1.0, 1 — якщо елемент більший або дорівнює 1.0.

Матриця суміжності Aundir ненапрямленого графа одержується з матриці Adir:
adir(i,j) = 1 => aundiri(j) = 1, aundirj(i) = 1.

Номер варіанту: 4208

Тексти програми на JS та HTML (з технічний причин було прийнято рішення написати JS код в одному файлі через обмеження модульності в браузерному середовищі):

index.html :

```
<!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;
    }

    .buttons {
      display: inline-block;
    }

    button {
      margin-right: 10px;
      padding: 10px 15px;
    }
  </style>
</head>
<body>
  <h1>Варіант 4208</h1>
  <div id="buttons">
    <button onclick="drawDirected()">Напрямлений граф</button>
    <button onclick="drawUndirected()">Ненапрямлений граф</button>
  </div>

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

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

script.js :

```
//config
const canvas = document.getElementById('canvas');
const ctx = canvas.getContext('2d');
const seed = 4208;
const n3 = 0;
const n4 = 8;
const n = 10;
const k = 1.0 - n3 * 0.02 - n4 * 0.005 - 0.25; //0.71
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} `);
  matrix.forEach((row) => {
    const line = row.map((v) => (v === 1 ? '1 ' : '0 ')).join(' ');
    console.log(line);
  });
}
```

```
//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, point = null) {
  let angle;
  let vx;
  let vy;
  if (point) {
    vx = to.x - point.x;
    vy = to.y - point.y;
  } else {
    vx = to.x - from.x;
    vy = to.y - from.y;
  }
  angle = Math.atan2(vy, vx);

  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.closePath();
  ctx.fill();
}

function drawGraph(matrix, directed = true) {
  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 (k === i || k === j || j === i) continue;
  const pk = positions[k];

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

    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 OFFSET = 90;

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

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

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

  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 dirMatrix = genDirMatrix(rand, k);
const undirMatrix = genUndirMatrix(dirMatrix);

printMatrix(dirMatrix, 'Directed Matrix');
printMatrix(undirMatrix, 'Undirected Matrix');

function drawDirected() {
  drawGraph(dirMatrix, true);
}

function drawUndirected() {
  drawGraph(undirMatrix, false);
}
```

Матриці суміжності:

Напрямлена

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

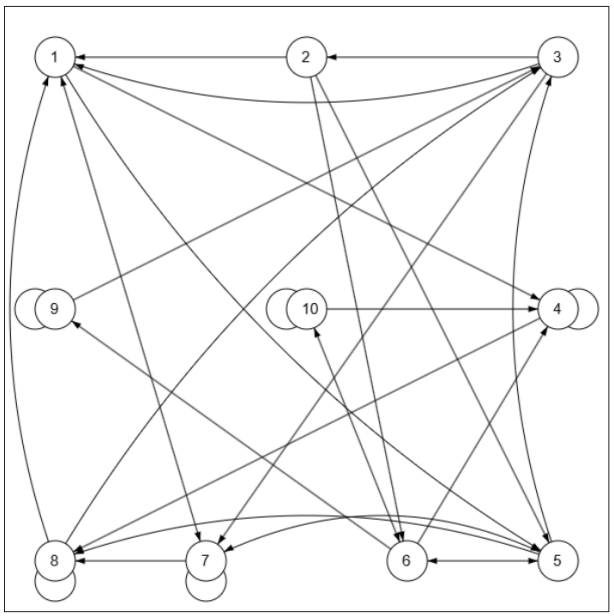
Не напрямлена

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

Результати тестування програми:

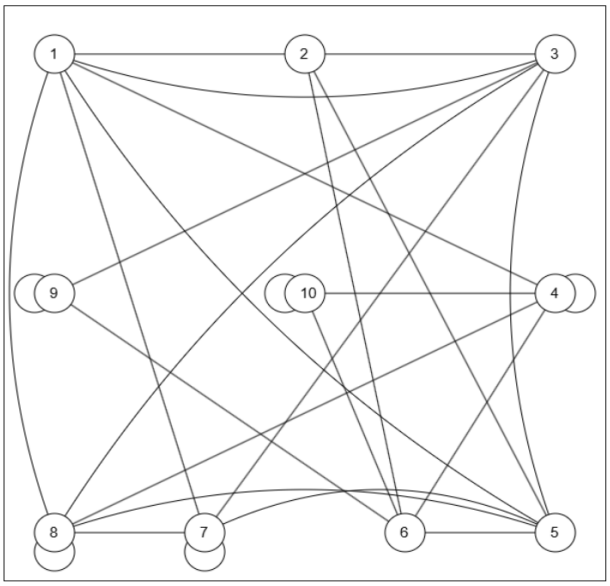
Варіант 4208

Напрямлений граф | Ненапрямлений граф



Варіант 4208

Напрямлений граф | Ненапрямлений граф



Висновок:

В ході виконання лабораторної роботи були набуті практичні навички представлення графів у комп’ютері та їх візуалізації.

Задача не виявилась легкою, потребувала повторення знань з курсу аналітичної геометрії. Текст програми виявився набагато більшим за тексти програм минулих лабораторних робіт.