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 №2

з дисципліни
«Дискретні структури»

на тему «Характеристики графів»

Виконав:

студент групи ІП-95

Грибинюк Олександр Сергійович
номер залікової книжки: 9504

Перевірила:

Сергієнко А. А.

Київ 2020

Завдання на лабораторну роботу

1. Представити напрямлений граф із заданими параметрами так само, як у лабораторній роботі No1.

— число вершин *n*;

— розміщення вершин;

— матриця суміжності A.

2. Визначити степені вершин напрямленого і ненапрямленого графів.

Програма на екран виводить степені усіх вершин ненапрямленого графу і напівстепені виходу та заходу напрямленого графу. Визначити, чи граф є однорідним та якщо так, то вказати степінь однорідності графу.

3. Визначити всі висячі та ізольовані вершини. Програма на екран виводить перелік усіх висячих та ізольованих вершин графу.

Варіант 9504:

n = 14

розміщення вершин: трикутником при n = 4

Текст програми для (JavaScript Canvas)

//JGraph.js

```
const Graph = function(ctx, adjM, directed = false, radius = 10) {
   this.directed = directed;
   this.adjM = adjM;
   this.vertices = [];
   this.ctx = ctx;
   this.radius = radius;
 };

 Graph.prototype.connect = function(v1, v2, directed = false, double = false) {
```

```
this.ctx.beginPath()
if (v1 === v2) {
  this.ctx.arc(v1.x + 7, v1.y + 14, this.radius - 3, 0, 2 * Math.PI);
  this.ctx.stroke();
  if (directed) {
    v1.out++;
    v1.in++;
  } else {
    v1.deg += 2;
  }
  return;
}
const xr = v2.x - v1.x;
const yr = v2.y - v1.y;
const k = yr / xr;
const xinter = Math.sqrt((this.radius ** 2)/(1 + k ** 2));
const yinter = k * xinter;
if (directed) {
  v1.out++;
  v2.in++;

  if (v1.x === v2.x) {
    if (v2.y > v1.y) {
      canvas_arrow(this.ctx, v1.x, v1.y + this.radius, v2.x, v2.y - this.radius);
    } else {
      canvas_arrow(this.ctx, v1.x, v1.y - this.radius, v2.x, v2.y + this.radius);
    }
  } else {
    if (v2.x < v1.x) {
      canvas_arrow(this.ctx, v1.x - xinter, v1.y - yinter, v2.x + xinter, v2.y + yinter);
    } else {
      canvas_arrow(this.ctx, v1.x + xinter, v1.y + yinter, v2.x - xinter, v2.y - yinter);
    }
  }
} else {
```

```
  v1.deg++;
  v2.deg++;

  if (v1.x === v2.x) {
    if (v2.y > v1.y) {
      this.ctx.moveTo(v1.x, v1.y + this.radius);
      this.ctx.lineTo(v2.x, v2.y - this.radius);
    } else {
      this.ctx.moveTo(v1.x, v1.y - this.radius);
      this.ctx.lineTo(v2.x, v2.y + this.radius);
    }
  } else {
    if (v2.x < v1.x) {
      this.ctx.moveTo(v1.x - xinter, v1.y - yinter);
      this.ctx.lineTo(v2.x + xinter, v2.y + yinter);
    } else {
      this.ctx.moveTo(v1.x + xinter, v1.y + yinter);
      this.ctx.lineTo(v2.x - xinter, v2.y - yinter);
    }
  }
}
if (double) {

  v1.in++;
  v2.out++;

  const d = Math.sqrt((v2.x - v1.x) ** 2 + (v2.y - v1.y) ** 2);
  const k = (v2.y - v1.y) / (v2.x - v1.x);
  const phi = Math.atan(k);
  this.ctx.save();
  this.ctx.translate(v1.x, v1.y);
  this.ctx.rotate(phi);
  this.ctx.moveTo(this.radius, 0);
  canvas_arrow(this.ctx, d / 2, d / 8, this.radius, 0)
  this.ctx.moveTo(d / 2, d / 8);
  this.ctx.lineTo(d - this.radius, 0);
```

```
    this.ctx.restore();
  }
  this.ctx.stroke();
};

Graph.prototype.vertex = function(x, y, n) {
  const v = { x: x,  y: y, num: n, in: 0, out: 0, deg: 0,
  };
  this.vertices[n - 1] = v;
};

Graph.prototype.draw = function() {
  this.ctx.textAlign = 'center';
  this.ctx.textBaseline = 'middle';
  for (const vert of this.vertices) {
    this.ctx.beginPath();
    this.ctx.arc(vert.x, vert.y, this.radius, 0, 2 * Math.PI);
    this.ctx.fillText(vert.num, vert.x, vert.y);
    this.ctx.stroke();
  }
  if (!this.directed){
    for (const i in this.adjM) {
      for (const j in this.adjM[i]) {
        if (this.adjM[i][j] === 1 && i > j) {
          const v1 = this.vertices[i];
          const v2 = this.vertices[j];
          this.connect(v1, v2);
        } else if (i === j && this.adjM[i][j] === 1) {
          const v1 = this.vertices[i];
          this.connect(v1, v1);
        }
      }
    }
  } else {
    for (const i in this.adjM) {
      for (const j in this.adjM[i]) {
```

```
        if (this.adjM[i][j] === 1 && this.adjM[j][i] !== 1) {
          const v1 = this.vertices[i];
          const v2 = this.vertices[j];
          this.connect(v1, v2, true);
        }
        else if (this.adjM[i][j] === 1 && this.adjM[j][i] === 1 && i >= j) {
          const v2 = this.vertices[i];
          const v1 = this.vertices[j];
          this.connect(v1, v2, true, true);
        }
      }
    }
  }
};

Graph.prototype.triangle = function() {
  for (let i = 1; i <= 10; i++) {
    let x = 100 * i;
    let y = 50;
    if (i == 6) {
      x = 165;
      y= 170;
    };
    if( i == 7) {
      x = 200;
      y = 300;
    };
    if( i == 8) {
      x = 300;
      y = 500;
    };
    if (i == 9) {
      x = 430;
      y = 170;
    };
    if (i == 10) {
```

```
      x = 400;
      y = 300;
    };
    this.vertex(x, y, i);
  }
  this.draw();
};

Graph.prototype.info = function(container) {
  let exp = '';
  let info = '';
  let homo = false;
  let Degree = this.vertices[0].deg;
  if (this.directed) Degree = this.vertices[0].in + this.vertices[0].out;

  for (let vert of this.vertices) {
    if (!this.directed) {
      if (vert.deg === Degree) homo = true;
      info = `Vertex: ${vert.num}, deg: ${vert.deg}<br>`;
      } else {
      info = `Vertex: ${vert.num}, in: ${vert.in}, out: ${vert.out}<br>`;
    }
    exp += info;
  }

  for (let vert of this.vertices) {
      if (vert.deg === 0 && (vert.in === 0 && vert.out === 0)) {
        exp += `Ізольована вершина: ${vert.num} <p>`;
      } else if  ((vert.deg === 1) || (vert.in === 1 && vert.out === 0) || (vert.out === 1 &&
vert.in === 0)) {
        exp += `Лежача вершина: ${vert.num}> <p>`;
      }
    }
    if (homo) exp += 'Однорідний:';
  container.innerHTML = exp;
};
```

```
  const canvas_arrow = (context, fromx, fromy, tox, toy) => {
    const headlen = 10;
    const dx = tox - fromx;
    const dy = toy - fromy;
    const angle = Math.atan2(dy, dx);
    context.moveTo(fromx, fromy);
    context.lineTo(tox, toy);
    context.lineTo(tox - headlen * Math.cos(angle - Math.PI / 6), toy - headlen * Math.sin(angle - Math.PI / 6));
    context.moveTo(tox, toy);
    context.lineTo(tox - headlen * Math.cos(angle + Math.PI / 6), toy - headlen * Math.sin(angle + Math.PI / 6));
  }
```

//Lab2.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>Graph</title>
    <script type="text/javascript" src="JGraph.js" charset="utf-8"></script>
    <script type="text/javascript" src="lab2.js" charset="utf-8"></script>
    <style type="text/css">
      canvas { border: 1px solid black; }
      body { text-align: center; }
      div { text-align: right; border: 1px solid black; position: center; padding: 20px; text-align: center; margin-left: 20px;}
    </style>
  </head>
  <body onload="draw();">
    <h1>Directed graph</h1>
    <div id="graph1_info"></div>
    <canvas id="graph1" width="550" height="550"></canvas>
    <h1>Undirected graph</h1>
    <div id="graph2_info"></div>
    <canvas id="graph2" width="550" height="550"></canvas>
```

```
  </body>
</html>
```

//lab2.js

```
const matrix1 = [
  [0,   0,   0,   0,   0,   0,   0,   0,   0,   1],
  [0,   0,   1,   0,   0,   0,   0,   0,   0,   0],
  [0,   1,   0,   0,   0,   0,   0,   0,   0,   0],
  [0,   0,   0,   0,   1,   0,   0,   0,   1,   0],
  [0,   0,   0,   1,   1,   0,   0,   0,   0,   0],
  [0,   0,   0,   0,   0,   0,   1,   0,   0,   0],
  [0,   0,   0,   0,   0,   1,   0,   0,   1,   0],
  [0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
  [0,   0,   0,   1,   0,   0,   1,   0,   0,   0],
  [1,   0,   0,   0,   0,   0,   0,   0,   0,   0],
   ];

  const matrix2 = matrix1.map(arr => [...arr]);
  for (let i = 0; i < matrix2.length; i++) {
    for (let j = 0; j < matrix2.length; j++) {
      if (matrix2[i][j] === 1 && matrix2[j][i] !== 1) {
        matrix2[j][i] = 1;
      }
    }
  }

  const draw = () => {
    const canvas1 = document.getElementById('graph1');
    if (canvas1.getContext) {
      const container = document.getElementById('graph1_info');
      const ctx = canvas1.getContext('2d');
      const graph1 = new Graph(ctx, matrix1, true, 10);
      graph1.triangle();
      graph1.info(container);
    }
```

```
  const canvas2 = document.getElementById('graph2');
  if (canvas2.getContext) {
    const container = document.getElementById('graph2_info');
    const ctx = canvas2.getContext('2d');
    const graph2 = new Graph(ctx, matrix2, false, 10);
    graph2.triangle();
    graph2.info(container);
  }
```

};**Згенеровані матриці суміжності**

Матриця суміжності напрямленого графу:

```
[0,  0,  0,  0,  0,  0,  0,  0,  0,  1],
[0,  0,  1,  0,  0,  0,  0,  0,  0,  0],
[0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
[0,  0,  0,  0,  1,  0,  0,  0,  1,  0],
[0,  0,  0,  0,  1,  0,  0,  0,  0,  0],
[0,  0,  0,  0,  0,  0,  1,  0,  0,  0],
[0,  0,  0,  0,  0,  0,  0,  0,  1,  0],
[0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
[0,  0,  0,  1,  0,  0,  0,  0,  0,  0],
[0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
```

Матриця суміжності ненапрямленого графу:

```
[0,  0,  0,  0,  0,  0,  0,  0,  0,  1],
[0,  0,  1,  0,  0,  0,  0,  0,  0,  0],
[0,  1,  0,  0,  0,  0,  0,  0,  0,  0],
[0,  0,  0,  0,  1,  0,  0,  0,  1,  0],
[0,  0,  0,  1,  1,  0,  0,  0,  0,  0],
[0,  0,  0,  0,  0,  0,  1,  0,  0,  0],
[0,  0,  0,  0,  0,  1,  0,  0,  1,  0],
[0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
[0,  0,  0,  1,  0,  0,  1,  0,  0,  0],
[1,  0,  0,  0,  0,  0,  0,  0,  0,  0],
```

Результати виконання програми

Ненапрямлений граф:

Vertex: 1, deg: 1
Vertex: 2, deg: 1
Vertex: 3, deg: 1
Vertex: 4, deg: 2
Vertex: 5, deg: 3
Vertex: 6, deg: 1
Vertex: 7, deg: 2
Vertex: 8, deg: 0
Vertex: 9, deg: 2
Vertex: 10, deg: 1
Лежача вершина: 1>

Лежача вершина: 2>

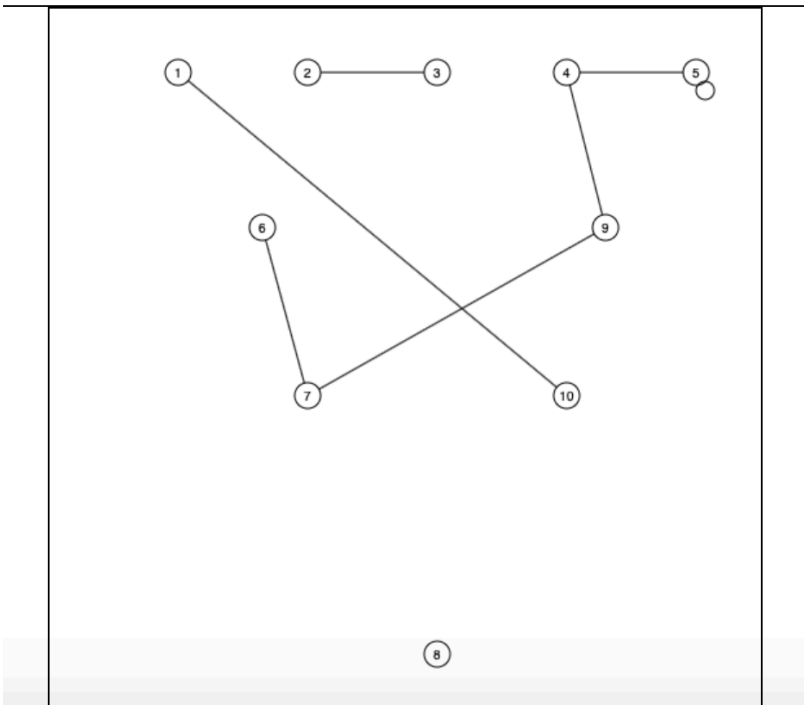
Лежача вершина: 3>

Лежача вершина: 6>

Ізольована вершина: 8

Лежача вершина: 10>

Однорідний:



Напрямлений граф:

Vertex: 1, in: 0, out: 1
Vertex: 2, in: 0, out: 1
Vertex: 3, in: 1, out: 0
Vertex: 4, in: 1, out: 2
Vertex: 5, in: 2, out: 1
Vertex: 6, in: 0, out: 1
Vertex: 7, in: 1, out: 1
Vertex: 8, in: 0, out: 0
Vertex: 9, in: 2, out: 1
Vertex: 10, in: 1, out: 0
Лежача вершина: 1>

Лежача вершина: 2>

Лежача вершина: 3>

Лежача вершина: 6>

Ізольована вершина: 8

Лежача вершина: 10>