**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 «Програмне забезпечення комп'ютерних систем - 2»*

*на тему: «Генерування випадкових графів задач»*

**Виконав:**

студент 5-го курсу ФІОТ

групи ІО – 82мп

*Барабаш Т.А.*

**Перевірила:**

доцент

*Русанова О. В.*

**ЛАБОРАТОРНА РОБОТА №5**

*Генерування випадкових графів задач*

**Мета:** Навчитись генерувати випадкові графи задачі із заданими параметрами

## І. Алгоритм роботи генератора

## Вхідні дані: N – кількість вершин, Wmax – максимальна вага вершини, Wmin – мінімальна вага вершини, Cor – зв’язність вершин.

## Згенерувати N вершин та призначити їм вагу у межах Wmax, Wmin.

## Обчислити суму ваг всіх вершин.

## Використовуючи формулу: , де - сумарна вага вершин графу задачі; - сумарна вага дуг графу задачі; n - кількість вершин у графі задачі, виразити та обчислити значення.

## Генерувати випадкові дуги та додавати 1 до її ваги допоки сумарна вага генерованих дуг не буде рівною тої, що було отримано у пункті 3.

## Отриманий граф перевірити на циклічність, використовуючи алгоритм з першої лаб. роботи. Якщо знайдено цикл, видалити дугу, що створює цикл, та додати її вагу іншій випадковій дузі.

**ІI. Код програми**

handleGenerate = (e, formData) => {

e.preventDefault();

const { vertexNumb, maxWeight, minWeight, correlation } = formData;

this.generate(vertexNumb, maxWeight, minWeight, correlation);

};

generate = (vertexNumb, maxWeight, minWeight, correlation) => {

const { edges, nodes } = this.state.data;

const newNodes = [];

for (let i = 0; i < vertexNumb; i++) {

const weight = this.\_getRandomInt(minWeight, maxWeight);

newNodes.push({

id: i,

number: i,

weight,

label: `${i}\n—\n${weight}`,

x: this.\_getRandomInt(-500, 500),

y: this.\_getRandomInt(-500, 500)

});

}

const weights = newNodes.reduce((sum, node) => (sum += node.weight), 0);

const lengths = Math.round((weights \* (1 - correlation)) / correlation);

const matrix = [];

const arr = [];

for (let i = 0; i < vertexNumb; i++) arr.push(0);

for (let i = 0; i < vertexNumb; i++) matrix.push(arr.slice());

let cur\_l\_w = 0;

while (cur\_l\_w < lengths) {

let number1 = this.\_getRandomInt(0, vertexNumb - 1);

let number2 = this.\_getRandomInt(0, vertexNumb - 1);

if (number1 === number2) number2 = number1 + 1;

if (matrix[number2][number1] > 0) {

const tmp = number1;

number1 = number2;

number2 = tmp;

}

matrix[number1][number2] += 1;

cur\_l\_w += 1;

}

const newEdges = [];

matrix.forEach((arr, i) => {

arr.forEach((value, j) => {

if (value > 0)

newEdges.push({ from: i, to: j, weight: value, label: `${value}` });

});

});

nodes.clear();

edges.clear();

nodes.add(newNodes);

edges.add(newEdges);

this.graphToMatrix();

let cycle = this.isCyclic();

while (cycle) {

console.log(cycle);

// eslint-disable-next-line

edges.forEach(edge => {

console.log(

`${JSON.stringify(edge)}, cycle: ${JSON.stringify(

cycle

)} ${edge.from === cycle.from && edge.to === cycle.to}`

);

if (edge.from === cycle.from && edge.to === cycle.to) {

const weight = edge.weight;

edges.remove(edge.id);

const ids = edges.getIds();

const randomEdgeId = ids[this.\_getRandomInt(0, ids.length)];

console.log(edges.get(randomEdgeId));

const randowEdge = edges.get(randomEdgeId);

edges.update({

id: randomEdgeId,

weight: randowEdge.weight + weight,

label: `${randowEdge.weight + weight}`

});

console.log(edges.get(randomEdgeId));

this.graphToMatrix();

cycle = this.isCyclic();

return;

}

});

}

// if (this.isCyclic())

// this.generate(vertexNumb, maxWeight, minWeight, correlation);

this.setState({ generated: true });