**Лабораторна робота №3**

**Побудова мінімального остовного дерева**

**Мета роботи**: закріпити на практиці знання з теорії графів, отримання навичок конструювання ПЗ, розв’язання задач з використанням графів

**Завдання**

1.Надати відповіді на контрольні питання до роботи.

2.Розробити структури даних для подання неорієнтованого та орієнтованого графів за різними методам (обравши певну мову програмування):

* матрицею інцидентності;
* матрицею суміжності;
* списком ребер;
* об’єктно-орієнтованою моделлю.

Порівняти особливості і переваги кожного варіанту подання графів..

3. Розробити функції уводу та виводу графів.

*Розв’язання*

Мова програмування: JavaScript(TypeScript), HTML (Jade), CSS (SASS).

1. **Текст програми**

*Повний текст програми: https://github.com/DimaGashko/graphEditor*

***scripts/math/graph/graph.ts***

import Vertex from "./vertex";

import Edge from "./edge";

/\*\*

\* Представление графа

\*

\* Испльзование:

\*

\* let graph: Graph = new Graph();

\*

\* let v0 = new Vertex(0);

\* let v1 = new Vertex(1);

\*

\* let e0 = new Edge(v0, v1, 'uni', 1);

\*

\* graph.addVertex(v0);

\* graph.addVertex(v1);

\*

\* graph.addEdge(e1);

\*

\* Получим граф вида:

\* (0) ---> (1)

\*

\* uni - однонаправленное ребро

\* bi - двунаправленное

\*

\* @class

\*/

export default class Graph {

private vertices: Vertex[] = [];

private edges: Edge[] = [];

constructor() {

}

public addVertex(vertex: Vertex): void {

if (this.vertices.indexOf(vertex) !== -1) return;

this.vertices.push(vertex);

}

public addEdge(edge: Edge): void {

if (this.edges.indexOf(edge) !== -1) return;

this.edges.push(edge);

this.addVertex(edge.v1);

this.addVertex(edge.v2);

}

public getVertices(): Vertex[] {

return this.vertices.slice();

}

public getEdges(): Edge[] {

return this.edges.slice();

}

/\*\*

\* Возвращает граф в виде матрицы смежности

\* Если в графе есть кратные ребра, то они "склеиваются"

\* (Кратные ребра остаются, если они имеют разные направление)

\*/

public toAdjacencyMatrix(): number[][] {

let matrix: number[][] = [];

this.vertices.forEach(() => {

matrix.push(new Array(this.vertices.length).fill(0));

});

this.edges.forEach((edge) => {

let v1 = this.vertices.indexOf(edge.v1);

let v2 = this.vertices.indexOf(edge.v2);

if (v1 === -1 || v2 === -1) return;

if (v1 === v2) {

matrix[v1][v2] += edge.weight;

} else {

matrix[v1][v2] += edge.weight;

if (edge.type === "bi") {

matrix[v2][v1] += edge.weight;

}

}

});

return matrix;

}

/\*\*

\* Возвращает граф в виде матрицы инцидентности

\*

\* Ряды матрицы представляют ребра, столбцы - вершины

\* Вес ребер игнорируется

\*/

public toIncidenceMatrix(): number[][] {

let matrix: number[][] = [];

this.edges.forEach((edge, i) => {

matrix[i] = new Array(this.vertices.length).fill(0);

let v1 = this.vertices.indexOf(edge.v1);

let v2 = this.vertices.indexOf(edge.v2);

if (v1 === -1 || v2 === -1) return;

if (v1 === v2) {

matrix[i][v1] += 2;

} else {

matrix[i][v1] = (edge.type === "bi") ? 1 : -1;

matrix[i][v2] = 1;

}

});

return matrix;

}

/\*\*

\* На основании переданной матрицы смежности возвращает граф

\*

\* @param {number[][]} matrix матрица смежности

\* @returns Graph

\*/

static parseAdjacencyMatrix(matrix: number[][]): Graph {

let graph = new Graph();

let vertices: Vertex[] = [];

let len = matrix.length;

//Создаем вершины

//Добавляем их в граф сразу (что бы сохранить последовательность)

//Сохраняем их в массиве, что бы иметь быстрый доступ по индексу

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

let vertex = new Vertex();

vertices.push(vertex);

graph.addVertex(vertex);

}

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

for (let j = 0; j < len; j++) {

//Нет ребра

if (matrix[i][j] === 0) continue;

//Ориентированное ребро

if (matrix[i][j] !== matrix[j][i]) {

graph.addEdge(

new Edge(vertices[i], vertices[j], 'uni', matrix[i][j])

);

//Не ориентированное ребро

} else {

//не ориентированные ребра добавляем только в одной половине матрицы

if (i < j) continue;

graph.addEdge(

new Edge(vertices[i], vertices[j], 'bi', matrix[i][j])

);

}

}

}

return graph;

}

/\*\*

\* На основании переданной матрицы инцидентности возвращает граф

\*

\* @param {number[][]} matrix матрица смежности

\* @returns Graph

\*/

static parseIncidenceMatrix(matrix: number[][]): Graph {

if (!matrix.length) return new Graph();

let graph = new Graph();

let vertices: Vertex[] = [];

let verticesLen = matrix[0].length;

let edgesLen = matrix.length;

//Cоздаем все вершины и добавляем их в граф

//Что бы их последовательность была правильной,

//Добавть их нужно перед добавлением ребер

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

let v: Vertex = new Vertex();

vertices.push(v);

graph.addVertex(v);

}

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

let incidence: {val: number, index: number}[] = [];

matrix[i].forEach((val, i) => {

if (val === 0) return;

incidence.push({val: val, index: i});

});

if (incidence.length === 0) continue;

else if (incidence.length === 1) {

//Петля

let v = vertices[incidence[0].index];

graph.addEdge(new Edge(v, v));

} else {

let v1 = vertices[incidence[0].index];

let v2 = vertices[incidence[1].index];

if (incidence[0].val === incidence[1].val) {

graph.addEdge(new Edge(v1, v2, 'bi'));

} else if (incidence[0].val < 0) {

graph.addEdge(new Edge(v1, v2, 'uni'));

} else {

graph.addEdge(new Edge(v2, v1, 'uni'));

}

}

}

return graph;

}

/\*\*

\* Проверяет, являются ли переданные ребра кратными

\*

\* @param e1 первое ребро

\* @param e2 второе ребро

\*/

static isMultipleEdges(e1: Edge, e2: Edge): boolean {

return (

(e1.v1 === e2.v1 && e1.v2 === e2.v2) ||

(e1.v1 === e2.v2 && e1.v2 === e2.v1)

);

}

}

***scripts/math/graph/edge.ts***

import Vertex from "./vertex";

/\*\*

\* Ребро графа

\*

\* @class

\*/

export default class Edge {

constructor(

public v1: Vertex,

public v2: Vertex,

public type: 'uni' | 'bi' = 'bi',

public weight: number = 1,

public targ?: any

) {

}

}

***scripts/math/graph/vertex.ts***

/\*\*

\* Вершина графа

\*

\* @class

\*/

export default class Vertex {

constructor(public targ?: any) {

}

}

***scripts/../workspace/ws\_graph/ws\_vertex.ts***

import Vector from "../../../math/vector/vector";

import WSGraphComponent from "../ws\_graph\_component";

export default class WSVertex extends WSGraphComponent {

public radius: Vector = new Vector(22, 22);

constructor(

public coords: Vector = new Vector(),

name?: string

) {

super(name || 'v');

}

public style: IVertextStyle = {

...

}

/\*\*

\* Проверяет, находится ли переданная точка внутри

\*/

public checkContainPoint(point: Vector) {

let a = point.x - this.coords.x;

let b = point.y - this.coords.y;

return (a \* a + b \* b <= this.radius.x \* this.radius.x);

}

}

***scripts/../workspace/ws\_graph/ws\_edge.ts***

import WSGraphComponent from "../ws\_graph\_component";

export default class WSEdge extends WSGraphComponent {

constructor(name?: string) {

super(name || 'e');

}

public style: IEdgeStyle = {

...

}

}

***scripts/../workspace/ws\_graph/ws\_graph.ts***

import Graph from "../../../math/graph/graph";

import WSVertex from "./ws\_vertex";

import WSEdge from "./ws\_edge";

import { getRandomVector } from "../../../math/math";

export default class WSGraph {

public graph: Graph = new Graph();

constructor() {

}

public createByGraph(graph: Graph): void {

graph.getEdges().forEach((edge, i) => {

if (edge.targ instanceof WSEdge) return;

edge.targ = new WSEdge(`e${i + 1}`);

});

graph.getVertices().forEach((vertex, i) => {

if (vertex.targ instanceof WSVertex) return;

vertex.targ = new WSVertex(getRandomVector(-300, 300), `v${i + 1}`);

});

this.graph = graph;

}

}

***scripts/main.ts***

***...***

let editor = new GraphEditor(root);

let projectManager = editor.projectManager;

let project1 = projectManager.createNewProject('Случайный');

let project2 = projectManager.createNewProject('Кратные ребра');

let project3 = projectManager.createNewProject('Мартица смежности');

let project4 = projectManager.createNewProject('Матрица инцедентности');

let project5 = projectManager.createNewProject('Demo Graph');

project3.getWorkspace().wsGraph.createByGraph(

Graph.parseAdjacencyMatrix([

[0, 5, 0, 1, 0],

[5, 0, 0, 0, 0],

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

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

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

])

);

...

1. Результат виконання програми:

Граф заданий через объекты:

let graph = new Graph;

let v1 = new Vertex(new WSVertex(new Vector(-200, 100), 'v1'));

let v2 = new Vertex(new WSVertex(new Vector(0, 100), 'v2'));

let v3 = new Vertex(new WSVertex(new Vector(130, -70), 'v3'));

let v4 = new Vertex(new WSVertex(new Vector(150, 200), 'v4'));

graph.addEdge(new Edge(v2, v1, 'uni', 1, new WSEdge('e1')));

graph.addEdge(new Edge(v2, v3, 'uni', 1, new WSEdge('e2')));

graph.addEdge(new Edge(v1, v3, 'bi', 2, new WSEdge('e3')));

graph.addEdge(new Edge(v2, v4, 'uni', 1, new WSEdge('e4')));

graph.addEdge(new Edge(v3, v4, 'bi', 1, new WSEdge('e5')));

graph.addEdge(new Edge(v3, v3, 'uni', 1, new WSEdge('e6')));

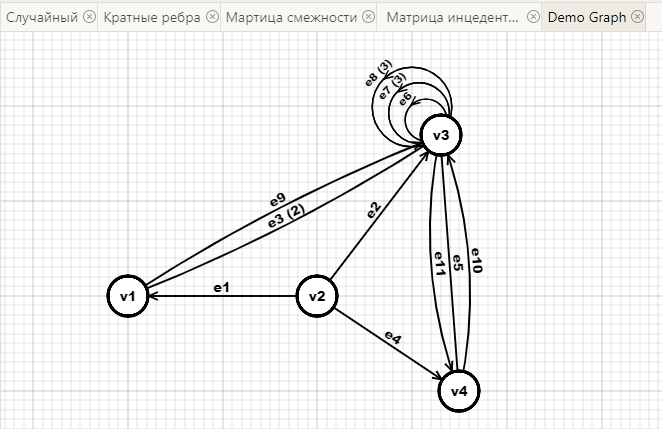
graph.addEdge(new Edge(v3, v3, 'bi', 3, new WSEdge('e7')));

graph.addEdge(new Edge(v3, v3, 'uni', 3, new WSEdge('e8')));

graph.addEdge(new Edge(v1, v3, 'uni', 1, new WSEdge('e9')));

graph.addEdge(new Edge(v4, v3, 'uni', 1, new WSEdge('e10')));

graph.addEdge(new Edge(v3, v4, 'uni', 1, new WSEdge('e11')));



**Через матрицю суміжності:**Graph.parseAdjacencyMatrix([

[0, 5, 0, 1, 0],

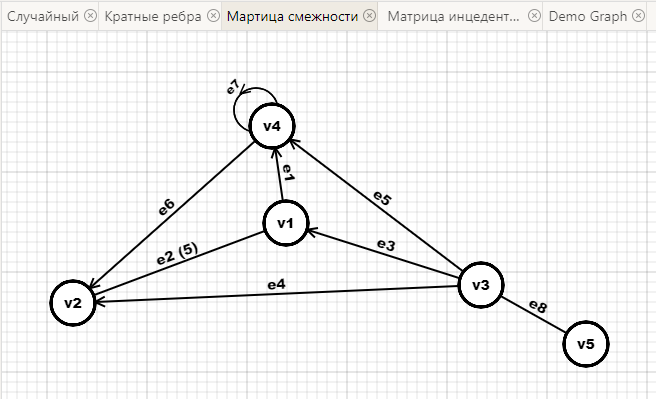
[5, 0, 0, 0, 0],

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

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

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

])



**Через матрицю інцидентності:**

Graph.parseIncidenceMatrix([

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

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

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

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

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

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

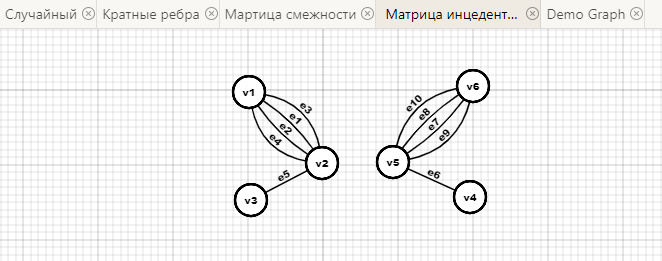
[0, 0, 0, 0, 1, 1],

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

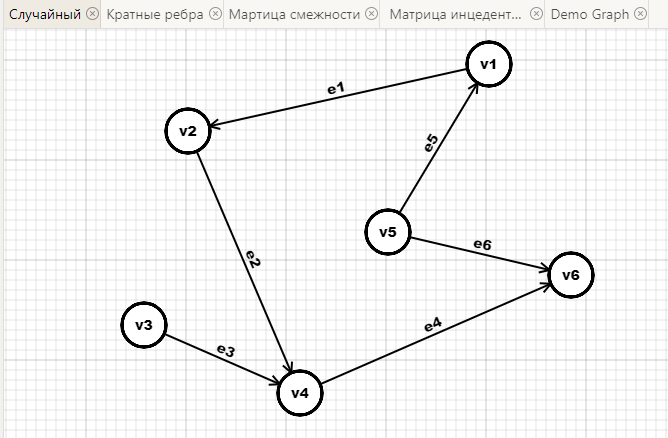
[0, 0, 0, 0, 1, 1],

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

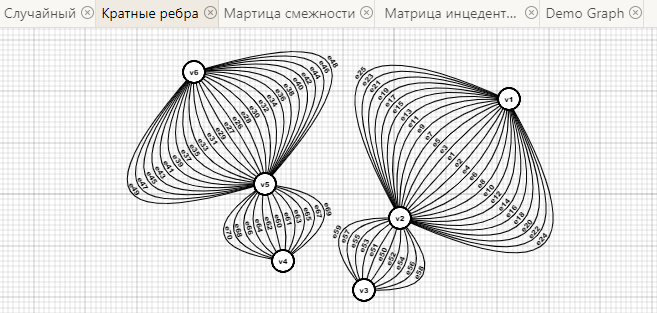
])



**Через випадково-згенеровану матрицю суміжності:**



**Через матрицю інцидентності з великою кількістю кратних ребер:**



**Висновок:** на цій лабораторній роботі я закріпив на практиці знання з теорії графів, отримав навички конструювання ПЗ, розв’язання задач з використанням графів*.*