**Лабораторна робота №2**

**Побудова мінімального остовного дерева**

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

**Завдання**

Розробити програму побудови мінімального остовного дерева графу з використанням структури даних для подання неорієнтованого графу з Л.Р. №1 та обходу вершин графу.

*Розв’язання*

1. **Текст програми**

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

**scripts/../graph/algorithms/getMST.ts**

import Graph from "../graph";

import Edge from "../edge";

/\*\*

\* Возвращает минимальное остовное дерево графа

\*

\* Основан на алгоритме Ярника-Прима-Дейкстры

\* Получает связный неориентированный граф

\*

\* @param graph Связный неориентированный граф

\* @param selectEdge функция выбора минимального ребра

\* (по умолчанию выберает ребро с наименьшим весом)

\*

\* @returns минимальное остовное дерево графа

\*/

export default function getMST<E, V>(

graph: Graph<E, V>,

getWeight?: (edge: Edge<E, V>) => number

): Graph<E, V> {

const mst = new Graph<E, V>();

const vertices = graph.getVertices();

let vLen = vertices.length;

if (vLen <= 1) {

mst.addAllVertices(vertices);

return mst;

}

const edges = graph.getEdges();

const v1 = vertices[0];

const e1 = selectEdge(graph.getVEdges(v1), getWeight);

mst.addEdge(e1);

vLen -= 2;

while (vLen) {

const nextEdge = selectEdge(getCandidateEdges(edges, mst), getWeight);

if (!nextEdge) return mst;

mst.addEdge(nextEdge);

vLen -= 1;

}

return mst;

}

//Возвращает ребра, в которых только 1 вершина принадлежит mst

function getCandidateEdges<E, V>(edges: Edge<E, V>[], mst: Graph<E, V>): Edge<E, V>[] {

return edges.filter((e) => {

return (mst.containsVertex(e.v1) !== mst.containsVertex(e.v2));

});

}

function selectEdge<E, V>(edges: Edge<E, V>[], getWeight: any = defGetWeight): Edge<E, V> | null {

if (!edges.length) return null;

let minEdge = edges[0];

let minWeight = getWeight(minEdge);

for (let i = 1; i < edges.length; i++) {

const weight = getWeight(edges[i]);

if (weight < minWeight) {

minWeight = weight;

minEdge = edges[i];

}

}

return minEdge;

};

//Дефолтное определение веса ребер

function defGetWeight <E, V>(edge: Edge<E, V>): number {

return edge.getWeight();

};

**main.ts**

let graph = graphDemo2;

const workspace = new Workspace(document.querySelector('.workspace'));

workspace.getData().wsGraph.graph = graph;

workspace.start();

workspace.addEvent('tik', () => {

const mst = getMST(graph, (edge) => {

return Vector.getSquareDistance(

edge.v1.targ.coords, edge.v2.targ.coords

) \* edge.getWeight();

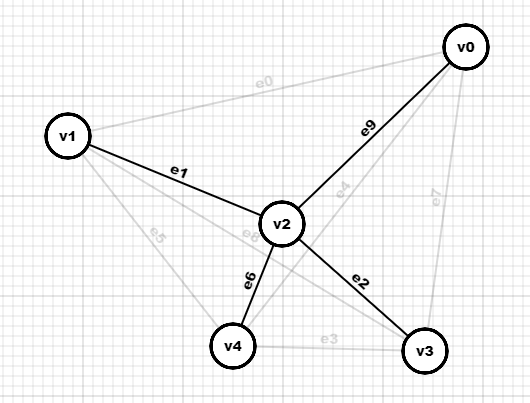
});

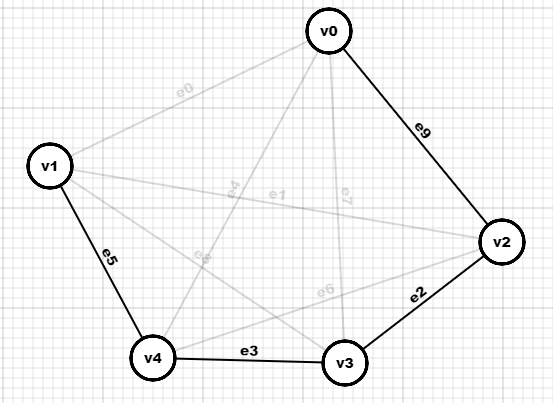
graph.getEdges().forEach(e => e.targ.style.color = "rgba(0,0,0,.15)");

mst.getEdges().forEach(e => e.targ.style.color = "#000");

});

1. Результат виконання програми:





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