**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”**

**Н-Н ІНСТИТУТ ПІДПРИЄМНИЦТВА ТА ПЕРСПЕКТИВНИХ ТЕХНОЛОГІЙ**



**ЗВІТ**

до виконаної лабораторної роботи №7

на тему

***«*Алгоритм Дейкстри*»***

Виконав:

студент гр. КН-214

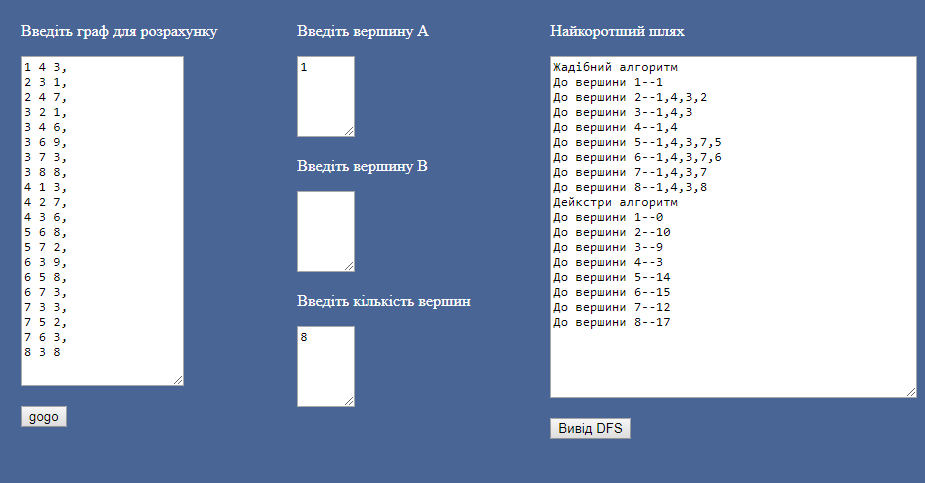
**Дубницький Ю. І.**

Прийняв:

**Слюсарчук Ю.М.**

**Львів–2019**

**Мета:**розробити програму знаходження найкоротшого шляху за допомогою алгоритму Дейкстри



function start() {

var text = document.getElementById('start').value;

function starting( text){

var array = text.split(",");

var number = [];

var a;

for(var i =0;i<array.length;i++){

a = [array[i].trim().split(" ")];

number[i] = a;

}

return number;

}

var number = starting(text);

console.log(number);

return number

}

function district() {

var starting = document.getElementById('verA').value;

var end = document.getElementById('verB').value;

var vers = document.getElementById('vers').value;

var text = document.getElementById('start').value;

var number = start(text);

var arraymax=[];

var arraynow=[];

var arraymin=[];

var obg={};

var visit= new Array(8);

var graf = new districtGraf();

for(var m = 1;m<=vers;m++){

graf.addVertex(''+m+'',searh(m,number))

}

var output = document.getElementById('districtValue');

output.value += ('Жадібний алгоритм'+'\n');

for(var b =1;b<=vers;b++){

output.value += 'До вершини '+b+'--'+(graf.shortestPath(starting, b+'').concat([starting]).reverse())+'\n';

}

for(var i =0;i<vers;i++){

arraymax[i]=100;

arraynow[i]=0;

visit[i]=0

}

for(var i = 0 ; i < vers ; i++){var verhin=100;

for(var k =0;k<number.length;k++){

if(starting == number[k][0][0] && visit[number[k][0][1]-1] != 1){

if(arraynow[number[k][0][1]-1] == 0){

if(arraymax[number[k][0][1]-1] > (+arraynow[number[k][0][0]-1] + +number[k][0][2])){

arraynow[number[k][0][1]-1]=+arraynow[number[k][0][0]-1] + +number[k][0][2]

obg[number[k][0][1]]=arraynow[number[k][0][1]-1]

}

}else if(arraynow[number[k][0][1]-1] != 0){

if(arraynow[number[k][0][1]-1] > (+arraynow[number[k][0][0]-1] + +number[k][0][2])){

arraynow[number[k][0][1]-1]=+arraynow[number[k][0][0]-1] + +number[k][0][2]

obg[number[k][0][1]]=arraynow[number[k][0][1]-1]

}

}

visit[number[k][0][0]-1]=1;

}

}

starting = searchmin(obg);

for (const prop of Object.getOwnPropertyNames(obg)) {

delete obg[prop];

}

if(starting==undefined){

starting=search2(number,lastStarting)

}

var lastStarting=starting

}

console.log(arraynow);

console.log(arraymax);

console.log(visit);

console.log(obg)

output.value += ('Дейкстри алгоритм'+'\n');

for(var q =1;q<=vers;q++){

output.value += 'До вершини '+q+'--'+arraynow[q-1]+'\n';

}

}

function search2(array,numb){

for(var g=0;g<array.length;g++){

if(numb==array[g][0][0]){

return array[g][0][1]

}

}

}

function searchmin(obg){

var min =Math.min.apply(Math,Object.values(obg))

for(var item in obg){

if(obg[item] == min){

return item

}

}

}

function searh(ver,array){

var obg={};

for(var i =0 ;i<array.length;i++){

if(array[i][0][0]==ver){

obg[array[i][0][1]]=array[i][0][2]

}

}

return obg

}

function PriorityQueue () {

this.\_nodes = [];

this.enqueue = function (priority, key) {

this.\_nodes.push({key: key, priority: priority });

this.sort();

};

this.dequeue = function () {

return this.\_nodes.shift().key;

};

this.sort = function () {

this.\_nodes.sort(function (a, b) {

return a.priority - b.priority;

});

};

this.isEmpty = function () {

return !this.\_nodes.length;

};

}

function districtGraf(){

var INFINITY = 1/0;

this.vertices = {};

this.addVertex = function(name, edges){

this.vertices[name] = edges;

};

this.shortestPath = function (start, finish) {

var nodes = new PriorityQueue(),

distances = {},

previous = {},

path = [],

smallest, vertex, neighbor, alt;

for(vertex in this.vertices) {

if(vertex === start) {

distances[vertex] = 0;

nodes.enqueue(0, vertex);

}

else {

distances[vertex] = INFINITY;

nodes.enqueue(INFINITY, vertex);

}

previous[vertex] = null;

}

while(!nodes.isEmpty()) {

smallest = nodes.dequeue();

if(smallest === finish) {

path = [];

while(previous[smallest]) {

path.push(smallest);

smallest = previous[smallest];

}

break;

}

if(!smallest || distances[smallest] === INFINITY){

continue;

}

for(neighbor in this.vertices[smallest]) {

alt = distances[smallest] + this.vertices[smallest][neighbor];

if(alt < distances[neighbor]) {

distances[neighbor] = alt;

previous[neighbor] = smallest;

nodes.enqueue(alt, neighbor);

}

}

}

return path;

};

}

**Висновок:** я зробив програму знаходження найкоротшого шляху в графі.