Department of education and science of Ukraine

National technical university of Ukraine

«Kyiv polytechnic institute the name of Igor Sikorsky»

Faculty of informatics and computing engineering

Department of the computing engineering

Laboratory work №7

Discipline: «The algorithms theory»

Topic: «Greedy algorithms»

EXECUTED:

The first-year student

of FICT group IP-95

*Guskov Danil*

The Student book number- 9505

CHECKED:

Associate Professor

The Computer Engineering Department

PhD, SR

Antoniuk A.I.

Kyiv - 2020

**TASK**

**Goal:**

Implementation of the greedy algorithm for the traveling salesman task.

Greedy algorithms allow you to quickly find the solution for most tasks. However, the solution they often get is not optimal. Thus, not all tasks are guaranteed the existence of a correct greedy algorithm, that is, one that will find the optimal solution of the tasks.

**Task variant: 5**

In this paper, it is necessary to propose a greedy algorithm for the task of a salesman.

The task of the salesman is formulated for a complete graph. For a weighted complete graph G with n vertices, the distances between all pairs of vertices (i, j) are given. You need to find the shortest route that runs through all vertices of the graph and enters each vertex only once.

In this paper, we consider a symmetric variant of the problem when the distance from city i to city j is equal to the distance from j to i (distance between (i, j) and (j, i) is equal).

The task of the salesman belongs to NP-complete tasks and for him there is no optimal algorithm that would work in polynomial time. Therefore, heuristic algorithms are often used to solve it, including greedy algorithms.

You need to come up with the idea of ​​a greedy algorithm and test it for multiple copies of tasks.

**SOFTWARE CODE**

**'use strict';**

**const getMatrix = n => {**

**const matrix = [];**

**for (let i = 0; i < n; i++) {**

**matrix.push([]);**

**}**

**for (let i = 0; i < n; i++) {**

**for (let k = 0; k < n; k++) {**

**let opposite = matrix[k][i];**

**if (i === k) {**

**matrix[i][k] = 0;**

**} else if (opposite) {**

**matrix[i][k] = opposite;**

**} else {**

**matrix[i][k] = Math.floor(Math.random() \* n) + 1**

**}**

**}**

**}**

**return matrix;**

**}**

**const getShortestWay = matrix => {**

**const way = [];**

**for (let i = 0; i < matrix.length; i++) {**

**const peak = matrix[i];**

**let shortest = peak[0] === 0 ? peak[matrix.length - 1] : peak[0];**

**let save = shortest;**

**for (let k = 0; k < peak.length; k++) {**

**const current = peak[k];**

**if (i != k && current < shortest && current != 0) {**

**shortest = current;**

**way[i] = `${i + 1} > ${k + 1}`;**

**}**

**if (k === peak.length - 1 && save === shortest) {**

**shortest = peak.indexOf(save);**

**way[i] = `${i + 1} > ${peak.indexOf(save) + 1}`;**

**};**

**};**

**};**

**return way;**

**}**

**const gAlhorithm = matrix => {**

**const short = [];**

**const way = [];**

**const find = (arr, store, counter) => {**

**const save = arr.slice();**

**arr.sort((a, b) => a - b);**

**const first = arr.filter(item => item != 0)[0];**

**const index = save.indexOf(first);**

**const check = i => {**

**if (i === matrix.length) i = 0;**

**return store.includes(i) ? check(++i) : i;**

**}**

**store.push(check(index));**

**};**

**for (const index in matrix) {**

**find(matrix[index], way, index);**

**}**

**console.log('\n');**

**for (let i = 0; i < matrix.length; i++) {**

**const peak = matrix[i];**

**let save = [];**

**let shortest = peak[0] === 0 ? peak[matrix.length - 1] : peak[0];**

**for (let k = 0; k < peak.length; k++) {**

**const current = peak[k];**

**save.push(shortest);**

**if (current < shortest && i != k) {**

**shortest = current;**

**}**

**}**

**console.log(save);**

**short.push(peak.indexOf(shortest));**

**}**

**let s = 'The Way: ';**

**for (let i = 0; i < way.length; i++) {**

**const item = way[i];**

**if (i === 0) s += `${item}`**

**else s += ` > ${item}`**

**};**

**return s;**

**}**

**const g1 = getMatrix(4)**

**console.log('\n');**

**for (const item of g1) {**

**console.log(item);**

**}**

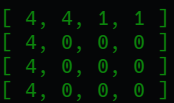
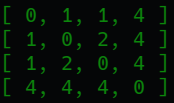
**console.log('The shortest way: ', getShortestWay(g1));**

**console.log(gAlhorithm(g1));**

**console.log('\n');**

**RESULTS OF THE PROGRAM WORK**

The input:

****

Output:





**CONCLUSIONS**

Familiarized with the topic of laboratory work.

Have acquired relevant work skills.

An appropriate test program has been developed.

The results of the successful work of the test program above confirm the correctness of the chosen decisions, the ultimate goal of the work has been achieved.