

Міністерство освіти і науки України  
Національний технічний університет України «Київський політехнічний інститут  
імені Ігоря Сікорського»  
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 8 з дисципліни  
«Алгоритми та структури даних-2.  
Структури даних»

«Жадібні алгоритми»

Варіант 5

Виконав студент

ІП-15, Буяло Дмитро Олександрович  
(шифр, прізвище, ім'я, по батькові)

Перевірив

Соколовський Владислав Володимирович  
(прізвище, ім'я, по батькові)

Київ 2022

## **Лабораторна робота 8**

### **Жадібні алгоритми**

**Мета** – вивчити основні жадібні алгоритми на варіанту задачі комівояжера та способи їх імплементації.

#### **Індивідуальне завдання**

#### **Варіант 5**

#### **Завдання**

У даній роботі необхідно запропонувати жадібний алгоритм для задачі комівояжера. Задача комівояжера формулюється для повного графу. Для зваженого повного графу  $G$  з  $n$  вершинами задані відстані між усіма парами вершин  $(i, j)$ . Необхідно знайти найкоротший маршрут, який проходить через всі вершини графу та заходить в кожну вершину лише один раз.

В даній роботі розглядається симетричний варіант задачі, коли відстань від міста  $i$  до міста  $j$  дорівнює відстані від  $j$  до  $i$  (відстані між  $(i, j)$  та  $(j, i)$  рівні). Задача комівояжера відноситься до NP-повних задач і для неї не існує оптимального алгоритму, який би працював за поліноміальний час. Тому для її розв'язання часто використовуються евристичні алгоритми, до яких відносяться і жадібні алгоритми.

Необхідно запропонувати ідею жадібного алгоритму та перевірити його роботу на декількох екземплярах задач.

## 1. Програмна реалізація алгоритму

### 1.1 Вихідний код

Main.cpp:

```
#include "Functions.h"

int main()
{
    int topCount = 0;
    int** point = input("input.txt", topCount, 2);
    double** graph = graphMake(topCount, point);

    int min = INT_MAX;
    vector<int> combinations;
    alg(graph, topCount, min, combinations);
    writer("output.txt", min, combinations);
}
```

Functions.h:

```
#pragma once
#include <iostream>
#include <string>
#include <fstream>
#include <math.h>
#include <vector>

using namespace std;

int** input(string, int&, int);
double distance(int, int, int, int);
double** graphMake(int, int**);
double find(vector<int>, double**);
bool duplicates(int, vector<int>);
void search(int, vector<int>, double**, int&, vector<int>&);
void alg(double**, int, int&, vector<int>&);
void writer(string, int, vector<int>&);
```

Functions.cpp:

```
#include "Functions.h"

int** input(string name, int& vert, int col) {
    ifstream inFile(name);
    string line;
    getline(inFile, line);
    vert = stoi(line);

    int** arr = new int* [vert];
    int x, y, space;
    for (int i = 0; i < vert; i++) {
        arr[i] = new int[col];
        getline(inFile, line);
        space = line.find(' ', 0);
        x = stoi(line.substr(0, space));
        y = stoi(line.substr(space, line.length() - space));
        arr[i][0] = x;
        arr[i][1] = y;
    }

    inFile.close();
    return arr;
}

double** graphMake(int vert, int** coordinates) {
    int x1, y1;
    const int x2 = 0, y2 = 1;
    double** graph = new double* [vert];
    for (int i = 0; i < vert; i++) {
        graph[i] = new double[vert];
        x1 = coordinates[i][x2];
        y1 = coordinates[i][y2];
        for (int j = 0; j < vert; j++) {
            if (i == j) {
                graph[i][j] = 0;
            }
            else {
                graph[i][j] = distance(x1, y1, coordinates[j][x2],
coordinates[j][y2]);
            }
        }
    }
    return graph;
}

double distance(int x1, int y1, int x2, int y2) {
    double distance = sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));
    return distance;
}
```

```

void alg(double** graph, int vert, int& min, vector<int>& combinations) {
    vector<int> c;
    c.push_back(0);
    search(vert, c, graph, min, combinations);
    combinations.push_back(0);
}

void search(int vert, vector<int> c, double** graph, int& min, vector<int>& combinations) {
    if (c.size() == vert) {
        int sum = find(c, graph);
        if (sum < min) {
            min = sum;
            combinations = c;
        }
        return;
    }
    vector<int> newC = c;
    for (int i = 1; i < vert; i++) {
        if (duplicates(i, newC)) {
            newC.push_back(i);
            search(vert, newC, graph, min, combinations);
            newC.pop_back();
        }
    }
}

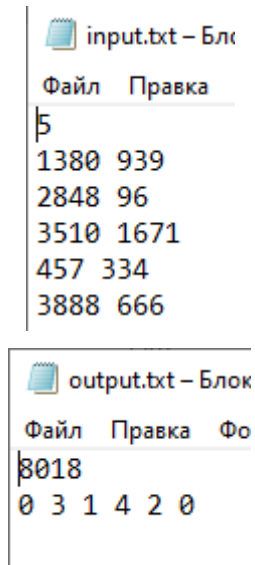
double find(vector<int> c, double** graph) {
    c.push_back(0);
    double sum = 0;
    for (int i = 0; i < c.size() - 1; i++) {
        sum += graph[c[i]][c[i + 1]];
    }
    return sum;
}

bool duplicates(int ind, vector<int> c) {
    for (int i = 0; i < c.size(); i++) {
        if (c[i] == ind) {
            return false;
        }
    }
    return true;
}

void writer(string name, int min, vector<int>& c) {
    ofstream outFile(name);
    outFile << min << "\n";
    for (int i = c.size() - 1; i >= 0; i--) {
        outFile << c[i] << " ";
    }
    outFile.close();
}

```

## 1.2 Приклад роботи



The image shows two windows of a text editor. The top window is titled 'input.txt - Блок' and contains the following text:

```
5
1380 939
2848 96
3510 1671
457 334
3888 666
```

The bottom window is titled 'output.txt - Блок' and contains the following text:

```
8018
0 3 1 4 2 0
```