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

Кафедра автоматизованих систем обробки інформації і управління

Звіт

з лабораторної роботи № 3 з дисципліни
«Алгоритми та структури даних 2. Структури даних»

«Евристичні алгоритми»

Виконав(ла) *ІП-02 Гущін Д.О.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Головченко М.М.* _____
(прізвище, ім'я, по батькові)

Київ 2021

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні підходи формалізації евристичних алгоритмів і вирішення типових задач з їх допомогою.

2 ЗАВДАННЯ

Для **задачі про найкоротший шлях**, вибрати 15 міст в країні згідно варіанту (таблиця 2.1) і записати для них найкоротшу відстань по дорозі, у випадку прямого сполучення між ними і відстань по прямій в окремі таблиці. Для визначення відстані рекомендується використовувати інтернет сервіси (наприклад Google Maps).

Для **задачі комівояжера**, вибрати 15 міст в країні згідно варіанту (таблиця 2.1) і записати для них найкоротшу відстань по дорозі, у випадку прямого сполучення між ними. Для визначення відстані рекомендується використовувати інтернет сервіси (наприклад Google Maps).

Для **задачі розфарбовування графа**, вибрати 15 адміністративних одиниць (областей, районів) в країні згідно варіанту (таблиця 2.1) і записати для них суміжність один з одним. Для визначення суміжностей рекомендується використовувати інтернет сервіси (наприклад Google Maps).

Для **задачі побудови мінімального вершинного покриття**, вибрати 15 міст в країні згідно варіанту (таблиця 2.1) і записати для них суміжність один з одним, у випадку прямого сполучення між ними. Для визначення відстані рекомендується використовувати інтернет сервіси (наприклад Google Maps).

Записати алгоритми методів відповідно до варіанту.

Виконати програмну реалізацію алгоритму використовуючи задані методи та евристики, надати відповідь згідно опису нижче.

Для **задачі про найкоротший шлях.** Розробити програму, яка буде знаходити найкоротші маршрути між кожною парою міст. У якості методів знаходження маршруту вибрати Жадібний пошук і пошук A*. У якості евристики вибрати відстань по прямій.

Відповідь вивести у вигляді (Місто1-Місто2 Відстань: 234км Маршрут: Місто1 → Місто3 → Місто4 → Місто2). **Вивести кожну пару міст, для обох алгоритмів.**

Для **задачі комівояжера.** Розробити програму, яка буде знаходити маршрут мінімальної довжини, що включає усі міста. У якості методів знаходження маршруту вибрати заданий за варіантом жадібний метод.

Відповідь вивести у вигляді (Маршрут: Місто1 → Місто3 → Місто4 → Місто2 → Місто1, Довжина: 234км).

Для **задачі розфарбовування графа.** Розробити програму, яка буде знаходити хроматичне число графа та кольори вершин. У якості методів знаходження хроматичного числа обрати пошук з поверненнями з заданою відповідно до варіанту евристикою.

Відповідь вивести у вигляді (Розфарбування: Місто 1 – Колір 1, Місто 2 – Колір 2, Місто 3 – Колір 1 …., Хроматичне число: 4).

Для **задачі побудови мінімального вершинного покриття.** Розробити програму, яка буде знаходити мінімальне вершинне покриття. У якості методів знаходження покриття вибрати жадібний метод та метод апроксимації.

Відповідь вивести у вигляді (Покриття: Місто1, Місто3, Місто2, Розмірність: 3).

Зробити узагальнений висновок з лабораторної роботи, в якому оцінити якість алгоритмів.

+1 додатковий бал можна отримати за програмне формування таблиць відстаней, суміжностей, тощо (за допомогою API інтернет сервісів) або за графічну демонстрацію роботи алгоритмів (на графі за допомогою десктопного інтерфейсу), отримати можна лише +1 бал.

Таблиця 2.1 – Варіанти алгоритмів

№	**Задача**	**Алгоритми**	**Евристика**	**Країна/Карта**
1	Задача про найкоротший шлях, пошук шляху та його	Жадібний пошук, A*	Відстань по прямій	Індонезія

	довжини			
2	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод найближчого сусіда	-	Австралія
3	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод включення найближчої вершини	-	Австрія
4	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод найдешевшого включення	-	Азербайджан
5	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	MRV	Іспанія
6	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	Ступенева евристика	Албанія
7	Задача розфарбовування графа, пошук хроматичного	Пошук з поверненнями	Попередня перевірка значень	Алжир

	числа			
8	Задача побудови мінімального вершинного покриття	Жадібний пошук, approx vertex cover	-	Італія
9	Задача про найкоротший шлях, пошук шляху та його довжини	Жадібний пошук, A*	Відстань по прямій	Ангола
10	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод найближчого сусіда	-	ОАЕ
11	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод включення найближчої вершини	-	Аргентина
12	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод найдешевшого включення	-	Вірменія
13	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	MRV	Мексика

14	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	Ступенева евристика	Афганістан
15	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	Попередня перевірка значень	Молдова
16	Задача побудови мінімального вершинного покриття	Жадібний пошук, approx vertex cover	-	Бангладеш
17	Задача про найкоротший шлях, пошук шляху та його довжини	Жадібний пошук, A*	Відстань по прямій	Барбадос
18	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод найближчого сусіда	-	Польща
19	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод включення найближчої вершини	-	Білорусь
20	Задача	Жадібний пошук	-	Португалія

	комівояжера, пошук маршруту та його довжини	метод найдешевшого включення		
21	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	MRV	Бельгія
22	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	Ступенева евристика	Сербія
23	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	Попередня перевірка значень	Болгарія
24	Задача побудови мінімального вершинного покриття	Жадібний пошук, approx vertex cover	-	Словаччина
25	Задача про найкоротший шлях, пошук шляху та його довжини	Жадібний пошук, A*	Відстань по прямій	Норвегія
26	Задача	Жадібний пошук	-	Нідерланди

	комівояжера, пошук маршруту та його довжини	метод найближчого сусіда		
27	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод включення найближчої вершини	-	Перу
28	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод найдешевшого включення	-	Франція
29	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	MRV	Таїланд
30	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	Ступенева евристика	Туреччина
31	Задача розфарбовування графа, пошук хроматичного числа	Пошук з поверненнями	Попередня перевірка значень	Хорватія

32	Задача побудови мінімального вершинного покриття	Жадібний пошук, approx vertex cover	-	Чехія
33	Задача про найкоротший шлях, пошук шляху та його довжини	Жадібний пошук, A*	Відстань по прямій	Швеція
34	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод найближчого сусіда	-	Еквадор
35	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод включення найближчої вершини	-	ЮАР
36	Задача комівояжера, пошук маршруту та його довжини	Жадібний пошук метод найдешевшого включення	-	ЮАР

3 ВИКОНАННЯ

3.1 Псевдокод алгоритмів

```
Function Backtracking(Matrix, ColorsArr, CurrVert, VertexAmount, possibleColors)
    if CurrVert == VertexAmount
        return true
    end if
    for (i = possibleColors[CurrVert][0]; i < possibleColors[CurrVert].size(); i++)
        if isSafe(CurrVert, Matrix, ColorsArr, i, VertexAmount)
            ColorsArr[CurrVert] = i
            for (int k = 0; k < VertexAmount; k++)
                if Matrix[CurrVert][k] != 0
                    int colorUsedInNode = ColorsArr[CurrVert]
                    possibleColors[k][colorUsedInNode] = -1
                end if
            end for
        end if
    end for
    if Backtracking(Matrix, ColorsArr, CurrVert + 1, VertexAmount, possibleColors)
            return true
            ColorsArr[CurrVert] = 0
    end if
    return false
end Function
```

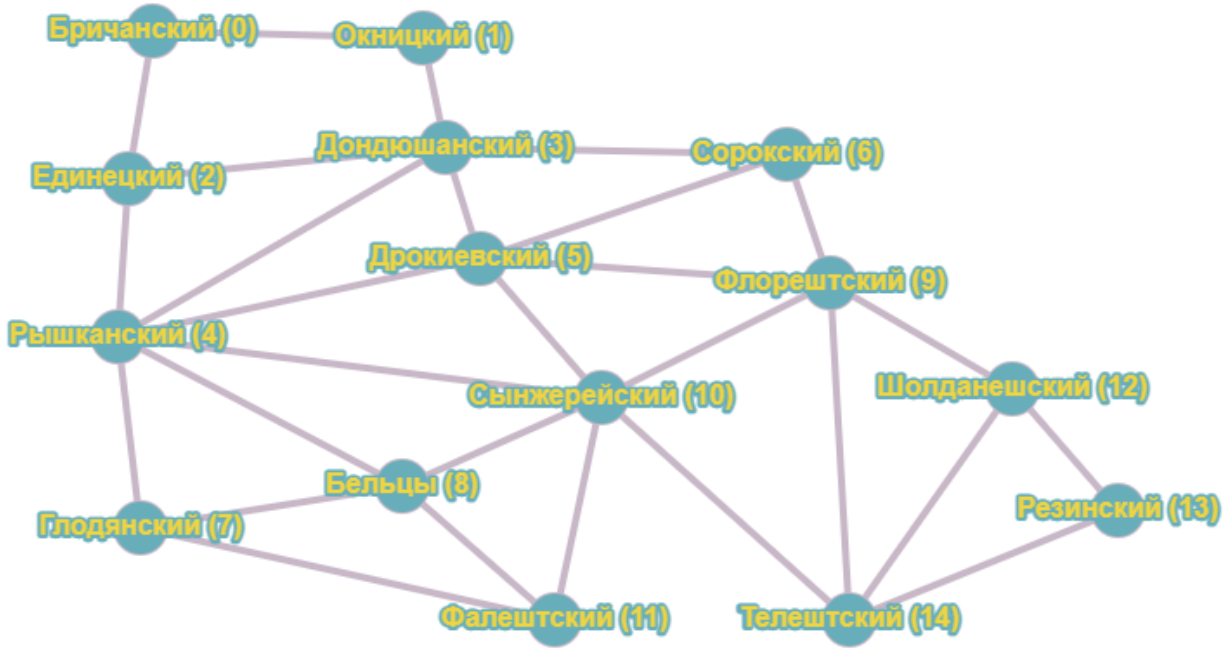
3.2 Вхідні дані задачі

Матриця суміжності:

0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	1	1	1	0	0	0	0	0	0	0	0
0	0	1	1	0	1	0	1	1	0	1	0	0	0	0
0	0	0	1	1	0	1	0	0	1	1	0	0	0	0
0	0	0	1	0	1	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	1	0	0	1	0	0	0
0	0	0	0	1	0	0	1	0	0	1	1	0	0	0
0	0	0	0	0	1	1	0	0	0	1	0	1	0	1
0	0	0	0	1	1	0	0	1	1	0	1	0	0	1
0	0	0	0	0	0	0	1	1	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	0	0	1	1	0	1	1	0

Карта:

Граф:



3.3 Програмна реалізація

3.3.1 Вихідний код

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <map>
#include <iomanip>
```

```
using namespace std;

int** ReadFile(int& size);
void FindSize(int& size);
int** CreateMatrix(int size);
void FillMatrix(int** Matrix, int size);
void DeleteMatrix(int** Matrix, int size);
void PrintMatrix(int** Matrix, int& size, string str);
vector<string> delim(string str, char delim);
void Result(int* color, int VertexAmount);
bool isSafe(int v, int** graph, int* color, int c, int VertexAmount);
bool Backtracking(int** graph, int* color, int v, int VertexAmount, map<int, vector<int>>
possibleColors);
bool ColoringAlgo(int** graph, int CurrentColor, int VertexAmount);
int FindChrom(int** Matrix, int VertexAmount);

int main()
{
    setlocale(LC_ALL, "Russian");
    int size;
    int** Matrix = ReadFile(size);
    int VertexAmount = 15;
    int ChromNum = FindChrom(Matrix, VertexAmount);
    std::cout << "Chromatic number: " << ChromNum << std::endl;
    ColoringAlgo(Matrix, ChromNum, VertexAmount);
    DeleteMatrix(Matrix, size);
    system("pause");
    return 0;
}

int** ReadFile(int& size) {
    FindSize(size);
    int** Matrix = CreateMatrix(size);
    cout << "Reading file..." << endl << endl;
    FillMatrix(Matrix, size);
    PrintMatrix(Matrix, size, "Adjacency matrix: ");
    return Matrix;
}

void FindSize(int& size) {
    ifstream input("..\\Moldova.txt");
    string currStr;
    size = 0;
    while (!input.eof()) {
        getline(input, currStr);
        size++;
    }
    input.close();
    input.clear();
}

int** CreateMatrix(int size) {
    int** Matrix = new int* [size];
    for (int i = 0; i < size; i++)
        Matrix[i] = new int[size];
    return Matrix;
}

void PrintMatrix(int** Matrix, int& size, string str) {
    cout << str << endl;
    cout << "   ";
    for (int i = 1; i <= size; i++)
        cout << setw(3) << i;
    cout << endl << endl;
    for (int i = 0; i < size; i++)
```

```
    {
        cout << setw(2) << i + 1 << "  ";
        for (int j = 0; j < size; j++)
            cout << setw(2) << Matrix[i][j] << ' ';
        cout << endl;
    }
    cout << endl;
}

void FillMatrix(int** Matrix, int size) {
    ifstream input("..\\Moldova.txt");
    for (int i = 0; i < size; i++)
    {
        string currStr;
        getline(input, currStr);
        vector<string> tmp = delim(currStr, ' ');
        for (int j = 0; j < size; j++)
            Matrix[i][j] = stoi(tmp[j]);
    }
    input.close();
    input.clear();
}

void DeleteMatrix(int** Matrix, int size) {
    for (int i = 0; i < size; i++)
        delete[] Matrix[i];
    delete Matrix;
}

vector<string> delim(string str, char delim) {
    int start, end = 0;
    vector<string> vect;
    while ((start = str.find_first_not_of(delim, end)) != string::npos)
    {
        end = str.find(delim, start);
        int length = end - start;
        if (length == 0) continue;
        string word = string(str, start, length);
        vect.push_back(word);
    }
    return vect;
}

void Result(int* ColorsArr, int VertexAmount)
{
    int max = 0;
    for (int i = 0; i < VertexAmount; i++)
        if (max < ColorsArr[i])
            max = ColorsArr[i];
    std::cout << "Minimumal chromatic number: " << max << endl << endl;
    vector<string> names = {
"Бричанский","Окницкий","Единецкий","Дондюшанский","Рышканский","Дрокиевский","Сорокский","Г
лодянский", "Бельцы","Флорештский","Сынжерейский","Фалештский","Шолданешский", "Резинский",
"Телештский" };
    cout << "Graph coloring: " << endl;
    for (int i = 0; i < VertexAmount; i++)
        cout << setw(15) << names[i] << setw(3) << ColorsArr[i] << endl;
}

bool isSafe(int v, int** Matrix, int* ColorsArr, int CurrColor, int VertexAmount)
{
    for (int i = 0; i < VertexAmount; i++)
        if (Matrix[v][i] && (CurrColor == ColorsArr[i] || CurrColor == -1))
            return false;
    return true;
```

```
}

bool Backtracking(int** Matrix, int* ColorsArr, int CurrVert, int VertexAmount, map<int,
vector<int>> possibleColors)
{
    if (CurrVert == VertexAmount)
        return true;
    for (int i = possibleColors[CurrVert][0]; i < possibleColors[CurrVert].size(); i++) {
        if (isSafe(CurrVert, Matrix, ColorsArr, i, VertexAmount))
        {
            ColorsArr[CurrVert] = i;
            for (int k = 0; k < VertexAmount; k++) {
                if (Matrix[CurrVert][k] != 0) {
                    int colorUsedInNode = ColorsArr[CurrVert];
                    possibleColors[k][colorUsedInNode] = -1;
                }
            }
    if (Backtracking(Matrix, ColorsArr, CurrVert + 1, VertexAmount, possibleColors))
            return true;
            ColorsArr[CurrVert] = 0;
        }
    }
    return false;
}

bool ColoringAlgo(int** Matrix, int CurrentColor, int VertexAmount)
{
    int* ColorsArr = new int[VertexAmount];
    for (int i = 0; i < VertexAmount; i++) ColorsArr[i] = 0;
    //Heuristic
    vector<int> allColors;
    for (int i = 0; i < CurrentColor; i++) allColors.push_back(i + 1);
    map<int, vector<int>> possibleColors;
    for (int i = 0; i < VertexAmount; i++) possibleColors[i] = allColors;
    // Backtracking
    if (!Backtracking(Matrix, ColorsArr, 0, VertexAmount, possibleColors)) return false;
    Result(ColorsArr, VertexAmount);
    delete[] ColorsArr;
    return true;
}

int FindChrom(int** Matrix, int VertexAmount)
{
    int* chrom = new int[VertexAmount];
    for (int i = 0; i < VertexAmount; i++)
    {
        chrom[i] = 0;
    }
    int max = 0;
    for (int i = 0; i < VertexAmount; i++)
    {
        for (int j = 0; j < VertexAmount; j++)
        {
            if (Matrix[i][j] == 1) chrom[i]++;
        }
        if (chrom[i] > max) max = chrom[i];
    }
    delete[] chrom;
    return max + 1;
}
```

3.3.2 Приклади роботи

На рисунку 3.1 показано приклад роботи програми.

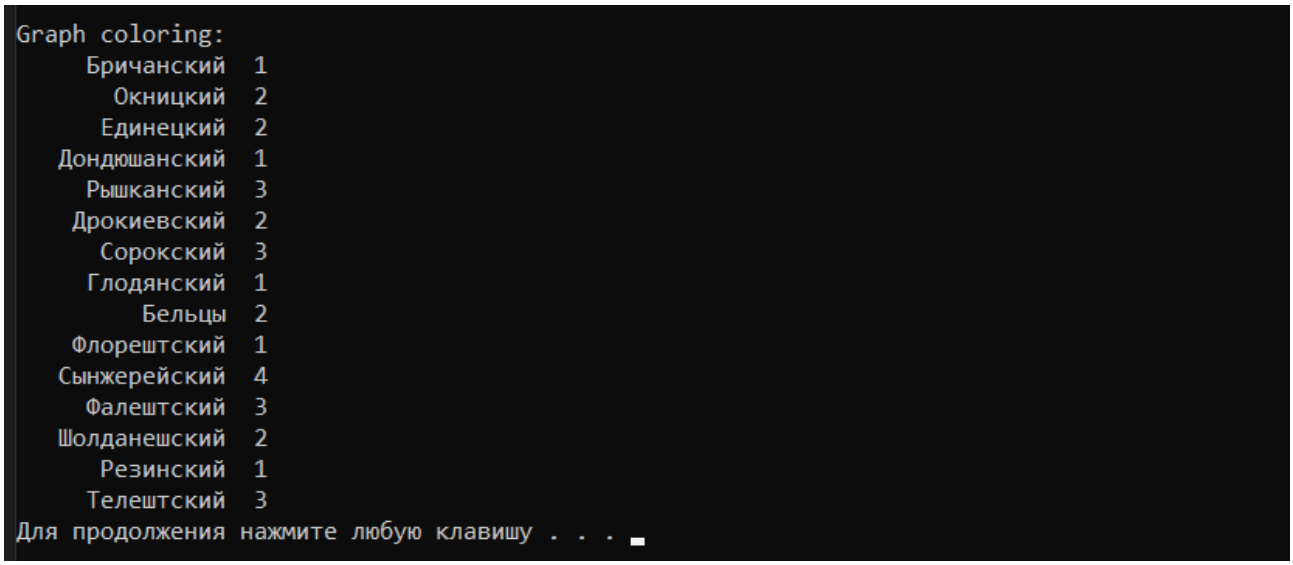


Рисунок 3.1

ВИСНОВОК

При виконанні даної лабораторної роботи ми вивчили основні підходи формалізації евристичних алгоритмів, а також способи їх імплементації. Окрім цього, ми виконали програмну реалізацію алгоритму на мові C++ на прикладі конкретного графа. Можна зробити висновок, що евристичні алгоритми дозволяють прискорити вирішення завдання і спрощують рішення практичних задач.

КРИТЕРІЇ ОЦІНЮВАННЯ

За умови здачі лабораторної роботи до 13.04.2020 включно максимальний бал дорівнює – 5. Після 13.04.2020 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

- псевдокод алгоритму – 10%;
- програмна реалізація алгоритму – 80%;
- висновок – 10%.
- +1 додатковий бал можна отримати за програмне формування таблиць відстаней, суміжностей, тощо (за допомогою API інтернет сервісів) або за графічну демонстрацію роботи алгоритмів (на графі за допомогою десктопного інтерфейсу), отримати можна лише +1 бал.