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 № 1 з дисципліни
«Алгоритми та структури даних 2. Структури даних»

«Прикладні задачі теорії графів ч.1»

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 прикладні алгоритми на графах та способи їх імплементації.

2 ЗАВДАННЯ

Згідно варіанту (таблиця 2.1), розробити та записати алгоритм задачі на графах за допомогою псевдокоду (чи іншого способу за вибором).

Виконати програмну реалізацію алгоритму на будь-якій мові програмування для довільного графа, передбачити введення розмірності графа та введення даних графа вручну чи випадковим чином.

Для самостійно обраного графа (розмірності не менше 9 вершин) розв’язати задану за варіантом задачу вручну.

Зробити узагальнений висновок з лабораторної роботи, у якому порівняти програмне та ручне розв’язання задачі.

Таблиця 2.1 – Варіанти алгоритмів

№	**Задача**	**Алгоритм**	**Тип графу**	**Спосіб задання графу**
1	Обхід графу	DFS	Неорієнтований	Матриця суміжності
2	Обхід графу	BFS	Неорієнтований	Матриця суміжності
3	Пошук маршруту у графі	Террі	Неорієнтований	Матриця суміжності
4	Пошук відстані між вершинами графа	Хвильовий	Неорієнтований	Матриця суміжності
5	Пошук найкоротшого шляху між парою вершин	Дейкстри	Орієнтований	Матриця вагів
6	Пошук найкоротшого шляху між парою вершин	Беллмана-Форда	Орієнтований	Матриця вагів

7	Побудова мінімальних покриваючих дерев	Прима	Неорієнтований	Матриця вагів
8	Побудова мінімальних покриваючих дерев	Крускала	Неорієнтований	Матриця вагів
9	Побудова мінімальних покриваючих дерев	Борувки	Неорієнтований	Матриця вагів
10	Побудова Ейлерового циклу	За означенням	Неорієнтований	Матриця суміжності
11	Побудова Ейлерового циклу	Флері	Неорієнтований	Матриця суміжності
12	Побудова Гамільтонового циклу	Пошук із поверненнями	Неорієнтований	Матриця суміжності
13	Обхід графу	DFS	Неорієнтований	Матриця інцидентності
14	Обхід графу	BFS	Неорієнтований	Матриця інцидентності
15	Пошук маршруту у графі	Террі	Неорієнтований	Матриця інцидентності
16	Пошук відстані між вершинами графа	Хвильовий	Неорієнтований	Матриця інцидентності
17	Пошук найкоротшого шляху між парою вершин	Дейкстри	Орієнтований	Матриця вагів
18	Пошук	Беллмана-	Орієнтований	Матриця вагів

	найкоротшого шляху між парою вершин	Форда		
19	Побудова мінімальних покриваючих дерев	Прима	Неорієнтований	Матриця вагів
20	Побудова мінімальних покриваючих дерев	Крускала	Неорієнтований	Матриця вагів
21	Побудова мінімальних покриваючих дерев	Борувки	Неорієнтований	Матриця вагів
22	Побудова Ейлерового циклу	За означенням	Неорієнтований	Матриця інцидентності
23	Побудова Ейлерового циклу	Флері	Неорієнтований	Матриця інцидентності
24	Побудова Гамільтонового циклу	Пошук із поверненнями	Неорієнтований	Матриця інцидентності
25	Обхід графу	DFS	Неорієнтований	Матриця суміжності
26	Обхід графу	BFS	Неорієнтований	Матриця суміжності
27	Пошук маршруту у графі	Террі	Неорієнтований	Матриця суміжності
28	Пошук відстані між вершинами графа	Хвильовий	Неорієнтований	Матриця суміжності
29	Пошук найкоротшого	Дейкстри	Орієнтований	Матриця вагів

	шляху між парою вершин			
30	Пошук найкоротшого шляху між парою вершин	Беллмана-Форда	Орієнтований	Матриця вагів
31	Побудова мінімальних покриваючих дерев	Прима	Неорієнтований	Матриця вагів
32	Побудова мінімальних покриваючих дерев	Крускала	Неорієнтований	Матриця вагів
33	Побудова мінімальних покриваючих дерев	Борувки	Неорієнтований	Матриця вагів
34	Побудова Ейлерового циклу	За означенням	Неорієнтований	Матриця суміжності
35	Побудова Ейлерового циклу	Флері	Неорієнтований	Матриця суміжності
36	Побудова Гамільтонового циклу	Пошук із поверненнями	Неорієнтований	Матриця суміжності

3 ВИКОНАННЯ

3.1 Псевдокод алгоритму

```
  if (beg == end) then
    stack.push(beg)
    flag = true;
  else
    for j = 0 to max_edge do
      if (A[beg][j] == 1) then
        if (in_stack(stack, beg) == false)
          then stack.push(beg)
        end if
        vetrex = find_vetr(A, max_vetrex, j, stack)
        if (vetrex != -1)
          then Terri_algo(A, max_vetrex, max_edge, vetrex, end, stack, flag)
        end if
      end if
    end for
    if (flag == false)
      then stack.pop()
    end if
```

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

```
#include <iostream>
#include <iomanip>
#include <stack>
using namespace std;

int** init_matr(int n, int m);
void print_matr(int** matr, int n, int m);
void delete_matr(int** matr, int n);
void Terri_algo(int** matr, int n, int m, int beg_v, int end_v, stack <int>& res, bool& flag);
void print_stack(stack<int> s);
int find_vetr(int** matr, int n, int j, stack <int> res);
bool in_stack(stack <int> s, int v);

int main(){
  int vetr, edge;
  cout << "Enter number of vetrex: "; cin >> vetr;
  cout << "Enter number of edges: "; cin >> edge;

  int** matr = init_matr(vetr, edge);
  print_matr(matr, vetr, edge);

  int beg_v, end_v;
  cout << "Enter beginning vetrex: "; cin >> beg_v;
  cout << "Enter ending vetrex: "; cin >> end_v;
  beg_v--; end_v--;

  stack <int> res;
  bool flag = false;
  Terri_algo(matr, vetr, edge, beg_v, end_v, res, flag);
  delete_matr(matr, vetr);
  system("pause >> void");
  return 0;
}
```

```
int** init_matr(int n, int m)
{
    int** matr = new int* [n];
    cout << endl;
    for (int i = 0; i < n; i++) {
        matr[i] = new int[m];
        for (int j = 0; j < m; j++) {
          cout << "Vetrex: " << i + 1 << '\t' << "Edge: " << j + 1 << " - ";
          cin >> matr[i][j];
        }
        cout << endl;
    }
    return matr;
}

void print_matr(int** matr, int n, int m)
{
    cout << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cout << setw(3) << matr[i][j];
        }
        cout << endl;
    }
    cout << endl;
}

void delete_matr(int** matr, int n)
{
    for (int i = 0; i < n; i++) delete []matr[i];
    delete[]matr;
}

void Terri_algo(int** matr, int n, int m, int beg_v, int end_v, stack <int>& res, bool& flag)
{
  if (beg_v == end_v)
  {
    res.push(beg_v);
    cout << "Route: ";
    print_stack(res);
    flag = true;
  }
  else
  {
    for (int j = 0; j < m; j++){
      if (matr[beg_v][j] == 1)
      {
        if (in_stack(res, beg_v) == false) res.push(beg_v);
        int vetr = find_vetr(matr, n, j, res);
        if (vetr != -1) Terri_algo(matr, n, m, vetr, end_v, res, flag);
      }
    }
    if (flag == false) res.pop();
  }
}

void print_stack(stack<int> s)
{
  if (s.empty()) return;
  int x = s.top();
  s.pop(); // Pop the top element of the stack
  print_stack(s); // Recursively call the function print_stack
  cout << x + 1 << ' '; //Print the stack element starting from the bottom
```

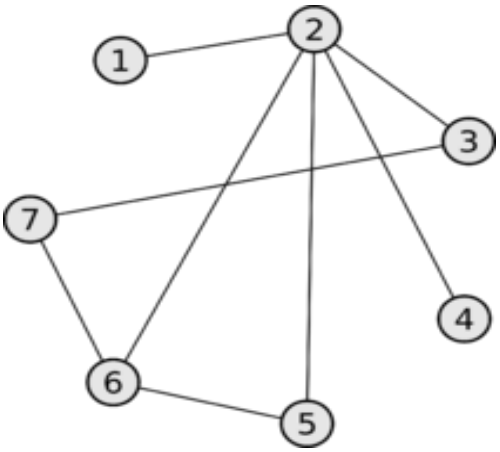
```
  s.push(x); // Push the same element onto the stack to preserve the order
}

int find_vetr(int** matr, int n, int edge, stack <int> res)
{
  int v = -1;
  for (int i = 0; i < n; i++)
    if (matr[i][edge] == 1 && in_stack(res, i) == false) {
      v = i;
      break;
    }
  return v;
}

bool in_stack(stack <int> s, int v)
{
  bool in_flag = false;
  while (s.empty() == false){
    if (s.top() == v) {
      in_flag = true;
      break;
    }
    s.pop();
  }
  return in_flag;
}
```

3.2.2 Приклад роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми для графів на 7 і 15 вершин відповідно.




```
  1  0  0  0  0  0  0  0
  1  1  1  1  1  0  0  0
  0  1  0  0  0  1  0  0
  0  0  1  0  0  0  0  0
  0  0  0  1  0  0  1  0
  0  0  0  0  1  0  1  1
  0  0  0  0  0  1  0  1

Enter beginning vetrex: 1
Enter ending vetrex: 7

1
1 2
1 2 3
Route: 1 2 3 7
```

Рисунок 3.1 – Задача знаходження маршруту у заданому графі

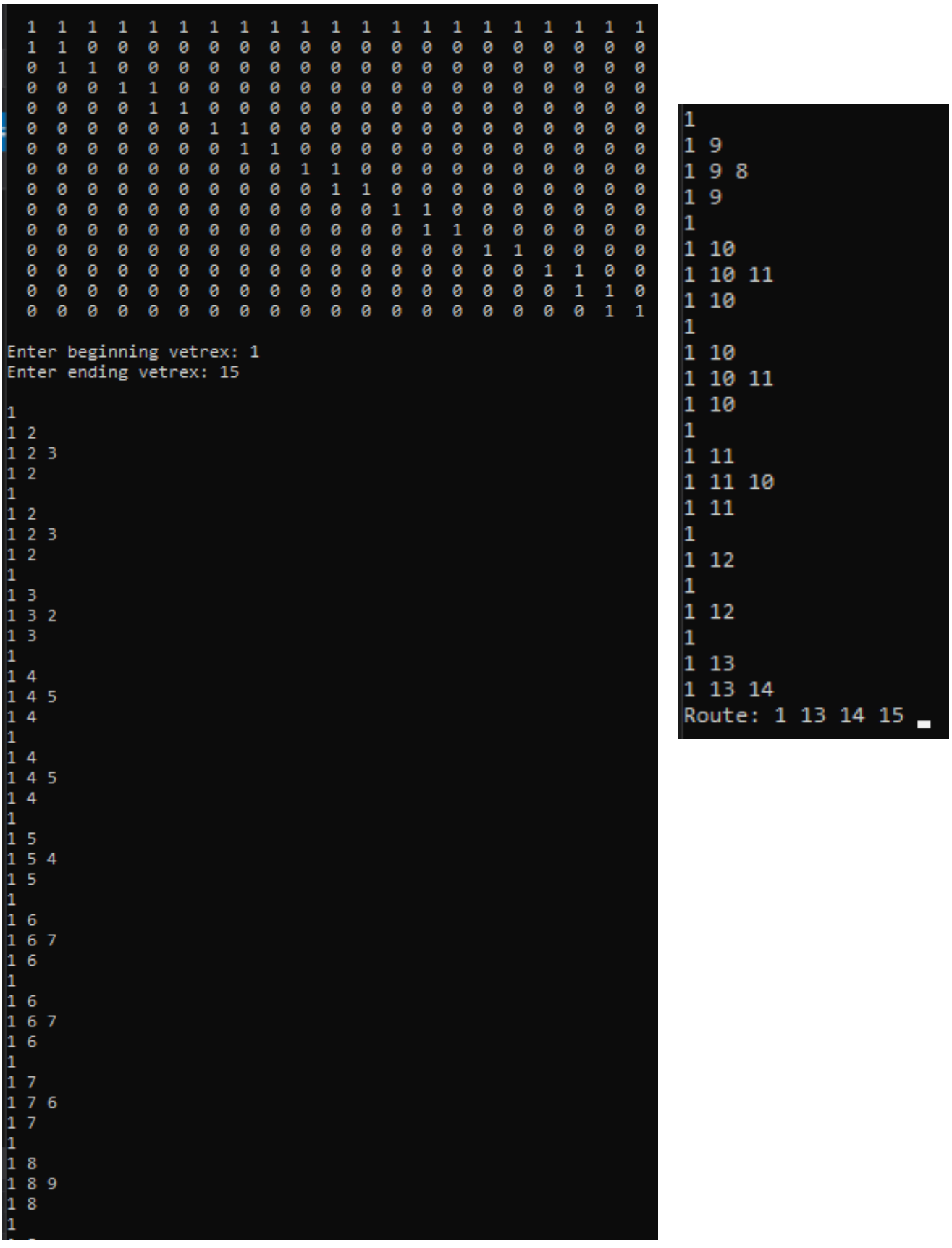


Рисунок 3.2 – Задача знаходження маршруту у заданому графі

3.3 Розв’язання задачі вручну

На рисунку 3.1 наведено розв’язання задачі знаходження маршруту у заданому графі за допомогою алгоритма Террі вручну.

0 - початкова вершина;

9 - кінцева вершина;

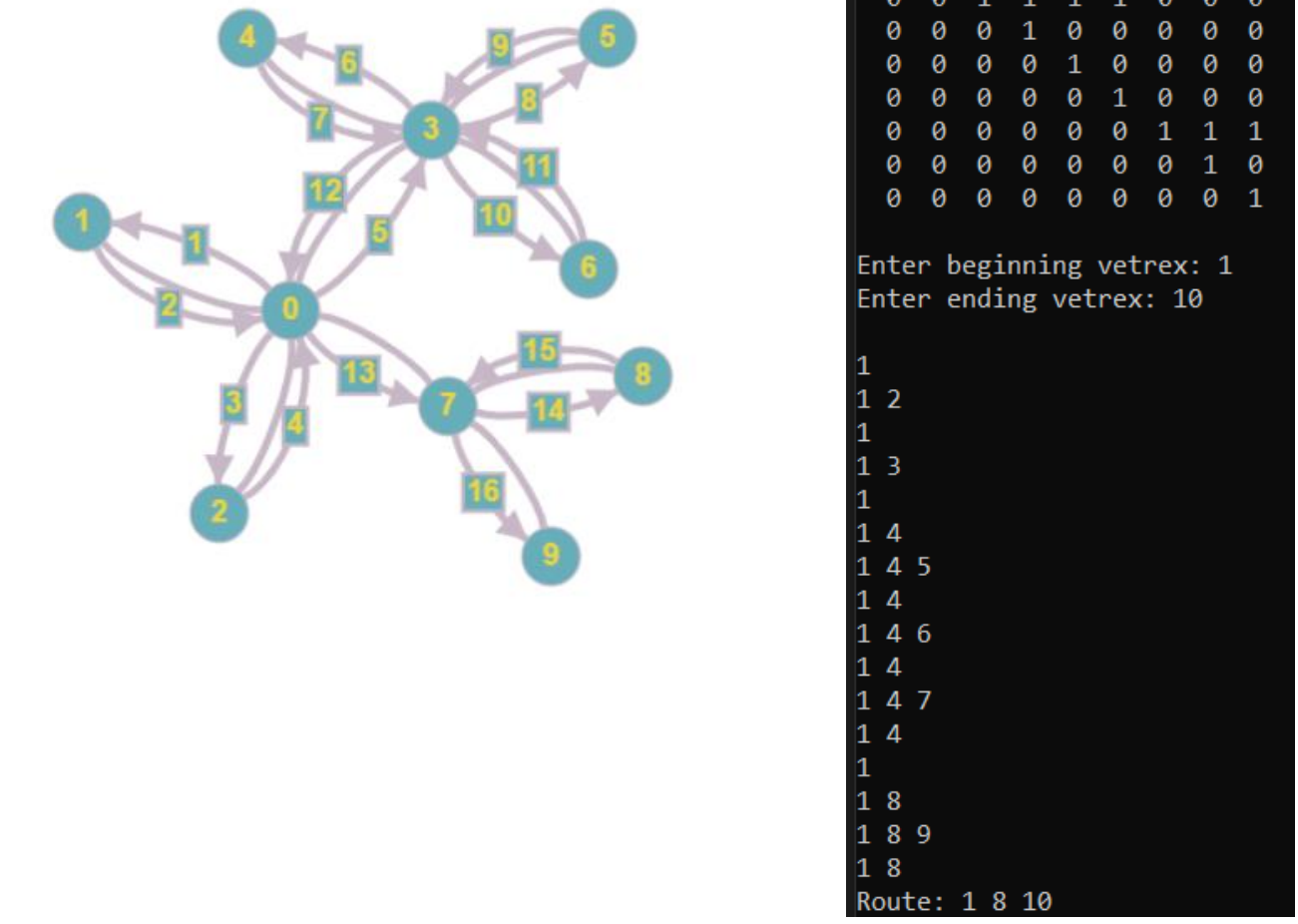


Рисунок 3.1 – Розв’язання задачі вручну

ВИСНОВОК

При виконанні даної лабораторної роботи ми розглянули основні прикладні алгоритми на графах та способи їх імплементації, розробили та записали алгоритм Террі знаходження маршруту у довільному графі за допомогою псевдокоду, виконали програмну реалізацію алгоритму на мові C++, а також розв’язали задану цю задачу на прикладі конкретного графа вручну. Можна зробити висновок, що програмна реалізація алгоритму дозволяє зекономити час і сили необхідні для вирішення цієї задачі вручну.

КРИТЕРІЇ ОЦІНЮВАННЯ

У випадку здачі лабораторної роботи до 15.03.2021 включно максимальний бал дорівнює – 5. Після 15.03.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

- псевдокод алгоритму – 10%;
- програмна реалізація алгоритму – 50%;
- розв’язання задачі вручну – 20%;
- відповідь на 3 теоретичні питання по темі роботи 15%
- висновок – 5%.