**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)**

(шифр, прізвище, ім'я, по батькові)

*ІП-02 Гущін Д. О.*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

Київ 2021

Зміст

[1 Мета лабораторної роботи 3](#_Toc49622506)

[2 ЗаВдання 4](#_Toc49622507)

[3 Виконання 6](#_Toc49622508)

[3.1 Псевдокод алгоритму 6](#_Toc49622509)

[3.2 Програмна реалізація алгоритму 6](#_Toc49622510)

[3.2.1 Вихідний код 6](#_Toc49622511)

[Висновок 7](#_Toc49622512)

[Критерії оцінювання 8](#_Toc49622513)

# Мета лабораторної роботи

Мета роботи – вивчити основні прикладні алгоритми на графах та способи їх імплементації.

# 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 найкоротшого шляху між парою вершин | Беллмана-Форда | Орієнтований | Матриця вагів |
| 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 шляху між парою вершин | Дейкстри | Орієнтований | Матриця вагів |
| 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 |

# Виконання

## 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**

## Програмна реалізація алгоритму

### 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;

}

### Приклад роботи

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

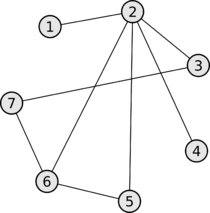
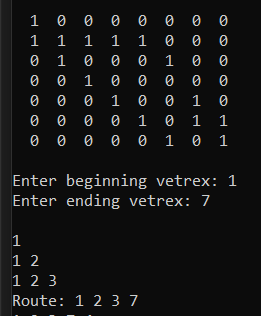
 

Рисунок 3.1 – Задача знаходження маршруту у заданому графі

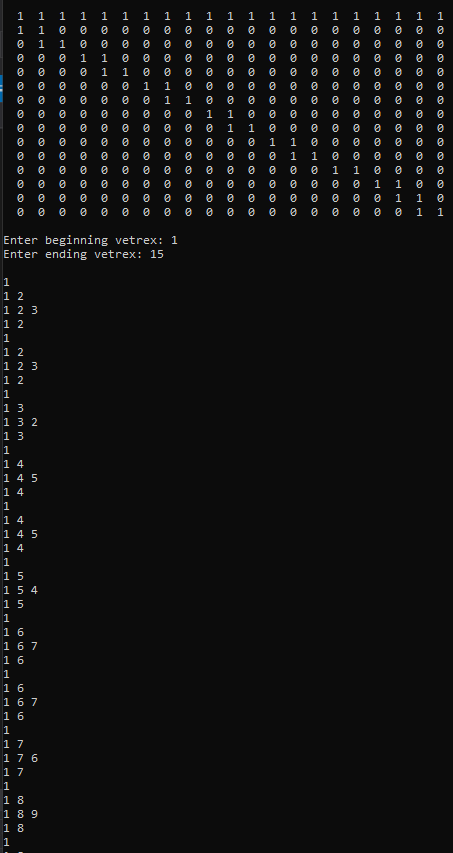
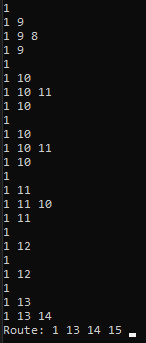
 

Рисунок 3.2 – Задача знаходження маршруту у заданому графі

## 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;

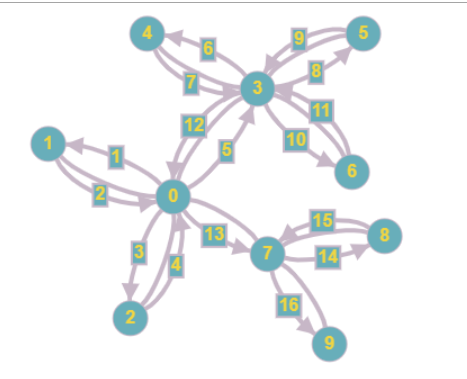
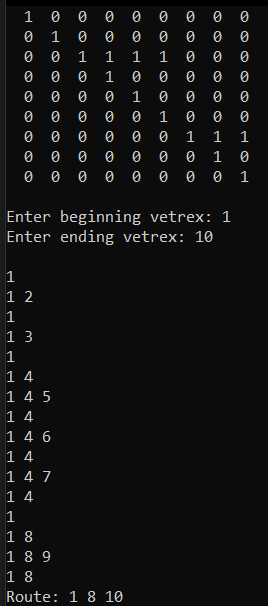
 

Рисунок 3.1 – Розв’язання задачі вручну

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 С++, 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%.