**Виконав:** Герман І.В.

**Група:** КН-404

**Кафедра:** САПР

**Дисципліна:** Дискретні моделі в САПР

# Варіант: 6

**Перевірив:** канд. техн. наук , доц. Кривий Р.З.

**Дата:** 22.02.2021

**GitHub:** https://github.com/Gera-byte/DM\_CAD/tree/main/Lab2

# ЗВІТ

до лабораторної роботи №2

на тему: АЛГОРИТМ РІШЕННЯ ЗАДАЧІ ЛИСТОНОШІ

# 1.МЕТА РОБОТИ

Метою даної лабораторної роботи є вивчення алгоритмів рішення задачі

листоноші.

**2. КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ**

Будь-який листоноша перед тим, як відправитись в дорогу повинен підібрати на пошті листи, що відносяться до його дільниці, потім він повинен рознести їх адресатам, що розмістились вздовж маршрута його проходження, і повернутись на пошту. Кожен листоноша, бажаючи втратити якомога менше сил, хотів би подолати свій маршрут найкоротшим шляхом. Загалом, задача листоноші полягає в тому, щоб пройти всі вулиці маршрута і повернутися в його початкову точку, мінімізуючи при цьому довжину пройденого шляху.

Перша публікація, присвячена рішенню подібної задачі, появилась в одному з китайських журналів, де вона й була названа задачею листоноші.

Очевидно, що така задача стоїть не тільки перед листоношею. Наприклад, міліціонер хотів би знати найбільш ефективний шлях патрулювання вулиць свого району, ремонтна бригада зацікавлена у виборі найкоротшого шляху переміщення по всіх дорогах.

Задача листоноші може бути сформульована в термінах теорії графів.

Для цього побудуємо граф G = (X , E), в якому кожна дуга відповідає вулиці в маршруті руху листоноші, а кожна вершина - стик двох вулиць. Ця задача являє собою задачу пошуку найкоротшого маршруту, який включає кожне ребро хоча б один раз і закінчується у початковій вершині руху.

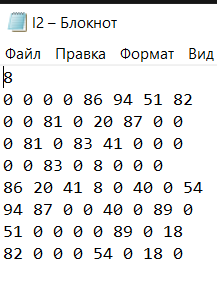
Нехай S-початкова вершина маршруту і a(i,j)>0 - довжина ребра (і, j) . В графі на рис. 1існує декілька шляхів, по яким листоноша може обійти всі ребра і повернутись у вершину S.

ЕЙЛЕРОВИЙ ЦИКЛ

Ейлеревим циклом в графі називається шлях, який починається і закінчується в тій самій вершині, при чому всі ребра графа проходяться тільки один раз. Ейлеревим шляхом називається шлях, який починається в вершині А, а закінчується в вершині Б, і всі ребра проходяться лише по одному разу. Граф, який включає в себе ейлерів цикл називається ейлеревим.

# Завдання

Вирішити задачу листоноші



**Код программи:**

#include <iostream>

#include "O:\НУ ЛП\DM\_CAD\lab1\DM\_CAD\Lab2\untitled4\97c39f35dab0c45a5d924dd9e50c445f-47ecad34033f986b0972cdbf4636e22f838a1313\stdc++.h"

#include "Graph.cpp"

using namespace std;

int main() {

const string filename = R"(O:\\l2.txt)";

Graph object;

object.Process(filename);

return 0;

}

#include "Graph.h"

#include <iostream>

#include <string>

#include "O:\НУ ЛП\DM\_CAD\lab1\DM\_CAD\Lab2\untitled4\97c39f35dab0c45a5d924dd9e50c445f-47ecad34033f986b0972cdbf4636e22f838a1313\stdc++.h"

using namespace std;

void Graph::ReadIntGraphFromFile(const string& filename)

{

size\_t tmp = 0;

ifstream in(filename);

string firstLine;

if(!in.is\_open())

{

cout << "File not open!" << endl;

}

getline(in, firstLine);

// size = static\_cast<const int>(firstLine); //1

//

// size = stoi(firstLine); //2

//

// istringstream ( firstLine ) >> size; //3

//

// stringstream geek(firstLine); //4

//

// geek >> size;

for (auto & i : intGraph)

{

for (auto & j : i)

{

in >> tmp;

if (tmp != ' ')

{

j = tmp;

}

}

}

}

void Graph::ReadBoolGraphFromFile(const string& filename)

{

size\_t tmp = 0;

ifstream in(filename);

string firstLine;

if(!in.is\_open())

{

cout << "File not open!" << endl;

}

getline(in, firstLine);

for (auto & i : boolGraph) {

for (auto & j : i) {

in >> tmp;

if (tmp != ' ')

{

if (tmp > 0)

{

j = true;

}

else

{

j = false;

}

}

}

}

}

void Graph::ShowGraph()

{

for (auto & i : intGraph)

{

for (auto j : i)

{

cout << j << '\t';

}

cout << endl;

}

cout << endl;

}

bool Graph::CalculateNeighbor(size\_t sizeGraph, size\_t neighbor, size\_t startPoint, vector<size\_t> graph)

{

for (int i = sizeGraph - 1; i >= 0; i--)

{

if (graph[i] % 2 == 1)

{

neighbor++;

startPoint = i;

}

}

if (neighbor > 2)

{

cout << "Eulerian cycle does not exist! " << endl;

return false;

}

return true;

}

bool Graph::FindEulerianPath()

{

vector<size\_t> graph;

size\_t sizeGraph = sizeof(intGraph) / sizeof(intGraph[0]);

graph.reserve(sizeGraph);

for(auto & i : intGraph)

{

graph.push\_back(accumulate(i, i + 5, 0));

}

size\_t startPoint = 0, neighbor = 0;

if (!CalculateNeighbor(sizeGraph, neighbor, startPoint, graph))

{

return false;

}

stack<size\_t> stack;

vector<size\_t> path;

size\_t currentPoint = startPoint;

while (!stack.empty() or

accumulate(intGraph[currentPoint], intGraph[currentPoint] + 5, 0) != 0)

{

if (accumulate(intGraph[currentPoint], intGraph[currentPoint] + 5, 0) == 0)

{

path.push\_back(currentPoint);

currentPoint = stack.top();

stack.pop();

}

else

{

for (size\_t i = 0; i < sizeGraph; i++)

{

if (intGraph[currentPoint][i] == 1)

{

stack.push(currentPoint);

intGraph[currentPoint][i] = 0;

intGraph[i][currentPoint] = 0;

currentPoint = i;

break;

}

}

}

}

SolutionOfEulerian(path, currentPoint);

return true;

}

void Graph::SolutionOfEulerian(const vector<size\_t>& path, size\_t currentPoint)

{

cout << "Eulerian cycle: " << endl;

for (auto elements : path)

{

cout << elements << " -> ";

}

cout << currentPoint << endl;

}

bool Graph::IsSafe(int i, const int\* cycle, int index)

{

if (boolGraph[cycle[index - 1]][i] == 0)

{

return false;

}

for (int j = 0; j < index; j++)

{

if (cycle[j] == i)

{

return false;

}

}

return true;

}

bool Graph::HamiltonianCycle(int\* cycle, int index)

{

if (index == size)

{

if (boolGraph[cycle[index - 1]][cycle[0]] == 1)

{

return true;

}

else

{

return false;

}

}

for (int i = 1; i < size; i++)

{

if (IsSafe(i, cycle, index))

{

cycle[index] = i;

if (HamiltonianCycle(cycle, index + 1))

{

return true;

}

cycle[index] = -1;

}

}

return false;

}

bool Graph::HamiltonianCycle()

{

int \*cycle = new int[size];

for (int i = 0; i < size; i++)

{

cycle[i] = -1;

}

cycle[0] = 0;

if (!HamiltonianCycle(cycle, 1))

{

cout << "Hamiltonian cycle does not exist! " << endl;

return false;

}

SolutionOfHamiltonian(cycle);

return true;

}

void Graph::SolutionOfHamiltonian(int\* cycle)

{

cout << "Hamiltonian cycle: " << endl;

for (int i = 0; i < size; i++)

{

cout << cycle[i] << " -> ";

}

cout << cycle[0] << endl;

}

void Graph::Process(const string& filename)

{

Graph object;

object.ReadBoolGraphFromFile(filename);

object.ReadIntGraphFromFile(filename);

cout << "Current graph: " << endl;

object.ShowGraph();

cout << "Search for the Euler cycle..." << endl;

bool EuleriaResult = object.FindEulerianPath();

if (!EuleriaResult)

{

cout << "Search for the Hamiltonian cycle..." << endl;

object.HamiltonianCycle();

}

}

Graph::Graph() = default;

# 

Рис.1. Результат виконання програми

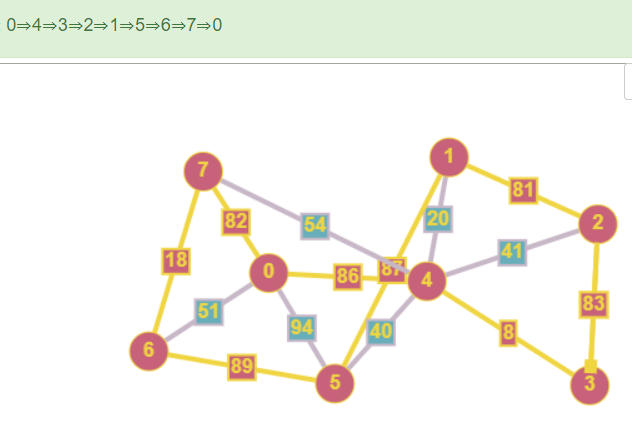


Рис.2. Візуалізація

## Висновок

На лабораторній роботі розглянуто алгоритм рішення задачі листоноші. В ході роботи мною було розроблено програмну реалізацію алгоритму.