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

**Лабораторна робота №5**

з дисципліни  
«Алгоритми і структури даних»

Виконав: Перевірила:

студент групи ІМ-31 Молчанова А. А.  
Литвиненко Сергій Андрійович  
номер у списку групи: 12

Київ 2024

**Постановка задачі**

1. Представити у програмi напрямлений i ненапрямлений графи з заданими параметрами так само, як у лабораторній роботі №3.

**Відмінність**: коефіцієнт k = 1.0 – n3 \* 0.01 – n4 \* 0.05 – 0.15.

Отже, матриця суміжності Adir напрямленого графа за варіантом формується таким чином:

1. встановлюється параметр (seed) генератора випадкових чисел, рiвний номеру варiанту n1n2n3n4;
2. матриця розмiром n \* n заповнюється згенерованими випадковими числами в дiапазонi [0, 2.0);
3. обчислюється коефiцiєнт k = 1.0 – n3 \* 0.01 – n4 \* 0.05 – 0.15, кожен елемент матрицi множиться на коефiцiєнт k;
4. елементи матрицi округлюються: 0 — якщо елемент менший за 1.0, 1 — якщо елемент більший або дорівнює 1.0.
5. Створити програму, яка виконує обхiд напрямленого графа вшир (BFS) та вглиб (DFS).

* обхiд починати з вершини iз найменшим номером, яка має щонайменше одну вихiдну дугу;
* при обходi враховувати порядок нумерацiї;
* у програмi виконання обходу вiдображати покроково, черговий крок виконувати за натисканням кнопки у вiкнi або на клавiатурi.

1. Пiд час обходу графа побудувати дерево обходу. У програмi дерево обходу виводити покроково у процесi виконання обходу графа. Це можна виконати одним iз двох способiв:

* або видiляти iншим кольором ребра графа;
* або будувати дерево обходу поряд iз графом.

1. Змiну статусiв вершин у процесi обходу продемонструвати змiною кольорiв вершин, графiчними позначками тощо, або ж у процесi обходу виводити протокол обходу у графiчне вiкно або в консоль.
2. Якщо пiсля обходу графа лишилися невiдвiданi вершини, продовжувати обхiд з невiдвiданої вершини з найменшим номером, яка має щонайменше одну вихiдну дугу.

**Варіант 12**

n1 = 3, n2 = 1, n3 = 1, n4 = 2;

Кількість вершин - 10 + n3 = 11;

Розміщення вершин – квадрат (прямокутник);

**Текст програми**

Файл headers/config.hpp

#pragma once

#include <string>

#include <SFML/Graphics/Color.hpp>

namespace config {

extern const char\* TITLE;

extern const unsigned WIDTH;

extern const unsigned HEIGHT;

extern const float LINE\_WIDTH;

extern const float VERTEX\_RADIUS;

extern const int SMOOTHING;

extern const sf::Color LINE\_COLOR;

extern const sf::Color BACKGROUND\_COLOR;

extern const sf::Color ACTIVE\_VERTEX\_COLOR;

extern const std::string FONT\_PATH;

extern const unsigned TEXT\_SIZE;

extern const float ARROWS\_LENGTH;

extern const unsigned CURVE\_ITEMS;

extern const int n1, n2, n3, n4;

extern const float k;

extern const size\_t VERTICES\_COUNT;

extern const size\_t SIDES;

extern const int SEED;

}

Файл headers/draw.hpp

#pragma once

#include <SFML/Graphics.hpp>

#include <functional>

#include "config.hpp"

#include "matrix.hpp"

#include "graph.hpp"

#include "vertex.hpp"

using matrix::matrix\_t, graph::dfs\_path, graph::bfs\_path;

namespace draw {

void drawGraph(sf::RenderWindow& window, const matrix\_t& matrix, size\_t sides, int size);

std::function<void(sf::RenderWindow&, bool)> drawDFSRouteClosure(

const matrix\_t& matrix,

const dfs\_path& route,

size\_t sides = config::SIDES,

int size = config::WIDTH

);

std::function<void(sf::RenderWindow&, bool)> drawBFSRouteClosure(

const matrix\_t& matrix,

const bfs\_path& route,

size\_t sides = config::SIDES,

int size = config::WIDTH

);

}

Файл headers/graph.hpp

#pragma once

#include <vector>

#include <functional>

#include "matrix.hpp"

using matrix::matrix\_t;

namespace graph {

using dfs\_item = std::pair<size\_t, bool>;

using bfs\_item = std::pair<size\_t, std::vector<size\_t>>;

using dfs\_path = std::vector<dfs\_item>;

using bfs\_path = std::vector<bfs\_item>;

template<typename T>

using search\_t = std::function<std::pair<std::vector<T>, matrix\_t>(const matrix\_t&, size\_t, std::vector<bool>&, std::vector<T>&, matrix\_t&)>;

std::pair<dfs\_path, matrix\_t> dfs(

const matrix\_t& matrix,

size\_t start,

std::vector<bool>& visited,

dfs\_path& path,

matrix\_t& bfsMatrix

);

std::pair<bfs\_path, matrix\_t> bfs(

const matrix\_t& matrix,

size\_t start,

std::vector<bool>& visited,

bfs\_path& paths,

matrix\_t& bfsMatrix

);

template<typename T>

std::pair<std::vector<T>, matrix\_t> getAllPaths(

const matrix\_t& matrix,

size\_t start,

search\_t<T> search

);

}

#include "graph.tcc"

std::ostream& operator<<(std::ostream& os, const graph::dfs\_path& path);

std::ostream& operator<<(std::ostream& os, const graph::bfs\_path& path);

Файл headers/matrix.hpp

#pragma once

#include <SFML/Graphics.hpp>

#include <vector>

namespace matrix {

using row\_t = std::vector<int>;

using matrix\_t = std::vector<row\_t>;

matrix\_t adjacencyMatrix(int size, int seed, float k);

matrix\_t toUndirected(const matrix\_t& matrix);

}

std::ostream& operator<<(std::ostream& os, const matrix::matrix\_t& matrix);

Файл headers/utils.hpp

#pragma once

#include <SFML/Graphics.hpp>

#include <string>

#include "config.hpp"

#include "draw.hpp"

#include <functional>

namespace utils {

using events\_t = std::vector<

std::tuple<

std::function<bool(const sf::Event&)>,

std::function<void(sf::RenderWindow&, bool)>,

std::string

>

>;

sf::RenderWindow& manageWindow(

sf::RenderWindow& window,

unsigned width,

unsigned height,

const char\* title

);

void pollEvents(

sf::RenderWindow& window,

const events\_t& events,

const std::function<void(sf::RenderWindow&)>& reset

);

void pollEvents(sf::RenderWindow& window);

sf::Font getFont(const std::string& path = config::FONT\_PATH);

std::function<bool(const sf::Event&)> onKeyDown(const sf::Keyboard::Key& key);

void clearWindow(sf::RenderWindow& window, const sf::Color& color);

size\_t getFistOutVertex(const matrix\_t& matrix);

void printNewVertexNumberingDFS(const dfs\_path& path);

void printNewVertexNumberingBFS(const bfs\_path& path);

}

Файл headers/vertex.hpp

#pragma once

#include <SFML/Graphics.hpp>

#include <functional>

#include "config.hpp"

namespace vertex {

struct Vertex {

float x;

float y;

size\_t index;

};

void draw(sf::RenderWindow& window, const vertex::Vertex& vertex, const sf::Color& color = config::LINE\_COLOR);

void lineConnect(

sf::RenderWindow& window,

const Vertex& from,

const Vertex& to,

bool shift = false,

bool dir = true,

const sf::Color& color = config::LINE\_COLOR

);

void arcConnect(

sf::RenderWindow& window,

const Vertex& from,

const Vertex& to,

bool dir = true,

const sf::Color& color = config::LINE\_COLOR

);

void loop(

sf::RenderWindow& window,

const Vertex& vertex,

bool dir = true,

const sf::Color& color = config::LINE\_COLOR

);

std::function<Vertex(size\_t)> getVertexClosure(

size\_t count,

size\_t sides = config::SIDES,

int width = config::WIDTH

);

void drawText(

sf::RenderWindow& window,

const sf::Vector2f& posc,

const std::string& txt,

const sf::Color& color = config::LINE\_COLOR

);

}

Файл config.cpp

#include <SFML/Graphics/Color.hpp>

#include <string>

#include "config.hpp"

namespace config {

const char\* TITLE{ "Lytvynenko Serhiy, IM-31" };

const unsigned WIDTH{ 800 };

const unsigned HEIGHT{ 800 };

const float LINE\_WIDTH{ 3.f };

const float VERTEX\_RADIUS{ 50.f };

const int SMOOTHING{ 8 };

const sf::Color LINE\_COLOR{ sf::Color::White };

const sf::Color BACKGROUND\_COLOR{ sf::Color::Black };

const sf::Color ACTIVE\_VERTEX\_COLOR{ sf::Color::Red };

const std::string FONT\_PATH{ "./fonts/arial.ttf" };

const unsigned TEXT\_SIZE{ 32 };

const float ARROWS\_LENGTH{ 15 };

const unsigned CURVE\_ITEMS{ 20 };

const int n1{ 3 }, n2{ 1 }, n3{ 1 }, n4{ 2 };

const float k{ 1.f - n3 \* 0.01f - n4 \* 0.005f - 0.15f };

const size\_t VERTICES\_COUNT{ 10 + n3 };

const size\_t SIDES{ 4 };

const int SEED{ n1 \* 1000 + n2 \* 100 + n3 \* 10 + n4 };

}

Файл draw.cpp

#include <SFML/Graphics.hpp>

#include <cmath>

#include <memory>

#include <functional>

#include "draw.hpp"

#include "graph.hpp"

#include "vertex.hpp"

#include "matrix.hpp"

using matrix::matrix\_t, graph::dfs\_path, graph::bfs\_path;

using namespace std::placeholders;

const std::array colors{

sf::Color(255, 154, 141), // Salmon

sf::Color(255, 165, 0), // Orange

sf::Color::Yellow, // Yellow

sf::Color(188, 169, 225), // Light Purple

sf::Color::Green, // Green

sf::Color(231, 236, 163), // Light Yellow

sf::Color::Magenta, // Magenta

sf::Color(255, 110, 64), // Red-orange

sf::Color(24, 104, 174), // Burnt sienna

sf::Color(229, 33, 101), // Pink

sf::Color(157, 225, 154), // Ligh Green

sf::Color(178, 2, 56), // Brick

sf::Color(255, 193, 59), // Mango

};

bool isNeighbours(size\_t count, size\_t i, size\_t j) {

if (i > j) std::swap(i, j);

return i == j - 1 || (i == 0 && j == count - 1);

}

bool inOneLine(size\_t count, size\_t sides, size\_t i, size\_t j) {

if (i > j) std::swap(i, j);

const auto split{ static\_cast<size\_t>(ceil(static\_cast<double>(count) / sides)) };

const auto cnt{ count - 1 };

const auto max{ split \* sides - 1 };

if (i == 0 && j > max - split) return true;

const auto start{ i - i % split };

const auto end{ start + split };

return j >= start && j <= end;

}

void connectVertices(

sf::RenderWindow& window,

const matrix\_t& matrix,

size\_t sides,

const vertex::Vertex& from,

const vertex::Vertex& to,

const sf::Color& color,

bool directed

) {

const auto i{ from.index };

const auto j{ to.index };

const auto count{ matrix.size() };

if (i == j) vertex::loop(window, from);

else if (!isNeighbours(count, i, j) && inOneLine(count, sides, i, j)) {

vertex::arcConnect(window, from, to, directed, color);

}

else {

const bool shift{ j < i && matrix[j][i] };

vertex::lineConnect(window, from, to, shift, directed, color);

}

}

void draw::drawGraph(sf::RenderWindow& window, const matrix\_t& matrix, size\_t sides, int size) {

const auto count{ matrix.size() };

const auto getVertex{ vertex::getVertexClosure(count, sides) };

const auto connect{

std::bind(connectVertices, \_1, matrix, sides, \_2, \_3, config::LINE\_COLOR, true)

};

for (size\_t i{ 0 }; i < count; i++) {

const auto vertex{ getVertex(i) };

vertex::draw(window, vertex);

for (size\_t j{ 0 }; j < count; j++) {

if (!matrix[i][j]) continue;

const auto otherVertex{ getVertex(j) };

connect(window, vertex, otherVertex);

}

}

}

std::function<void(sf::RenderWindow&, bool)> draw::drawDFSRouteClosure(

const matrix\_t& matrix,

const dfs\_path& route,

size\_t sides,

int size

) {

const auto getVertex{ vertex::getVertexClosure(matrix.size(), sides, size) };

const auto connect{ std::bind(connectVertices, \_1, matrix, sides, \_2, \_3, \_4, true) };

const auto stepP{ std::make\_shared<size\_t>(0) };

const auto routeSize{ route.size() };

const auto drawRoute {

[getVertex, connect, &route](sf::RenderWindow& window, size\_t step) {

const auto colorSize{ colors.size() };

if (step == route.size()) {

const auto i{ route[step - 1].first };

const auto [j, needConnect]{ route[step - 2] };

const auto from{ getVertex(j) };

const auto to{ getVertex(i) };

const auto active{ getVertex(i) };

const auto color{ colors[i % colorSize] };

if (needConnect) connect(window, from, to, color);

vertex::draw(window, to, color);

return;

}

const auto [i, needConnect]{ route[step] };

const auto active{ getVertex(i) };

vertex::draw(window, active, config::ACTIVE\_VERTEX\_COLOR);

if (step == 0) return;

const auto [j, needPrevConnect]{ route[step - 1] };

const auto prevActive{ getVertex(j) };

const auto color{ colors[j % colorSize] };

if (needConnect) connect(window, prevActive, active, config::ACTIVE\_VERTEX\_COLOR);

vertex::draw(window, prevActive, color);

if (step >= 2) {

const auto k{ route[step - 2].first };

const auto ppActive{ getVertex(k) };

const auto color{ colors[k % colorSize] };

if (needPrevConnect) connect(window, ppActive, prevActive, color);

}

}

};

return [drawRoute, stepP, routeSize](sf::RenderWindow& window, bool end) {

const auto step { \*stepP };

if (end) {

\*stepP = 0;

return;

}

if (step > routeSize) return;

drawRoute(window, step);

\*stepP += 1;

window.display();

};

}

std::pair<size\_t, size\_t> getNextBFSPair(const graph::bfs\_path& path, size\_t step) {

const auto size{ path.size() };

size\_t count{ 0 };

for (size\_t i{ 0 }; i < size; i++) {

const auto [active, neighbours] { path[i] };

const auto nsize{ neighbours.size() };

if (step == count) return std::make\_pair(active, SIZE\_MAX);

if (count < step && step <= count + nsize) {

return std::make\_pair(active, neighbours[step - count - 1]);

}

count += nsize + 1;

}

return std::make\_pair(SIZE\_MAX, SIZE\_MAX);

}

size\_t totalBFSSteps(const graph::bfs\_path& path) {

size\_t count{ 0 };

const auto size{ path.size() };

for (size\_t i{ 0 }; i < size; i++) {

count += path[i].second.size() + 1;

}

return count;

}

std::function<void(sf::RenderWindow&, bool)> draw::drawBFSRouteClosure(

const matrix\_t& matrix,

const bfs\_path& route,

size\_t sides,

int size

) {

const auto getVertex{ vertex::getVertexClosure(matrix.size(), sides, size) };

const auto connect{ std::bind(connectVertices, \_1, matrix, sides, \_2, \_3, \_4, true) };

const auto stepP{ std::make\_shared<size\_t>(0) };

const auto steps{ totalBFSSteps(route) };

const auto drawRoute{

[getVertex, connect, &route, steps](sf::RenderWindow& window, size\_t step) {

const auto colorSize{ colors.size() };

if (step != 0) {

const auto [active, neighbour]{ getNextBFSPair(route, step - 1) };

const auto from{ getVertex(active) };

const auto color{ colors[active % colorSize] };

vertex::draw(window, from, color);

if (neighbour != SIZE\_MAX) {

const auto to{ getVertex(neighbour) };

connect(window, from, to, color);

}

if (step == steps) return;

}

const auto [active, neighbour]{ getNextBFSPair(route, step) };

const auto from{ getVertex(active) };

vertex::draw(window, from, config::ACTIVE\_VERTEX\_COLOR);

if (neighbour == SIZE\_MAX) return;

const auto to{ getVertex(neighbour) };

vertex::draw(window, to, colors[neighbour % colorSize]);

connect(window, from, to, config::ACTIVE\_VERTEX\_COLOR);

}

};

return [drawRoute, stepP, steps](sf::RenderWindow& window, bool end) {

const auto step { \*stepP };

if (end) {

\*stepP = 0;

return;

}

if (step > steps) return;

drawRoute(window, step);

\*stepP += 1;

window.display();

};

}

Файл graph.cpp

#include <iostream>

#include <queue>

#include <stack>

#include "matrix.hpp"

#include "graph.hpp"

using matrix::matrix\_t, matrix::row\_t, graph::dfs\_path, graph::bfs\_path, graph::search\_t;

std::pair<dfs\_path, matrix\_t> graph::dfs(

const matrix\_t& matrix,

size\_t start,

std::vector<bool>& visited,

dfs\_path& path,

matrix\_t& dfsMatrix

) {

const auto size{ matrix.size() };

visited[start] = true;

auto stack{ std::stack<size\_t>{ } };

stack.push(start);

path.push\_back({ start, false });

auto returns{ false };

while (!stack.empty()) {

const auto vertex{ stack.top() };

if (returns) path.push\_back({ vertex, false });

bool flag{ false };

for (size\_t i{ 0 }; i < size; i++) {

if (!matrix[vertex][i] || visited[i]) continue;

returns = false;

flag = true;

dfsMatrix[vertex][i] = 1;

stack.push(i);

path.push\_back({ i, true });

visited[i] = true;

break;

}

if (!flag) {

stack.pop();

returns = true;

}

}

return std::make\_pair(path, dfsMatrix);

}

std::pair<bfs\_path, matrix\_t> graph::bfs(

const matrix\_t& matrix,

size\_t start,

std::vector<bool>& visited,

bfs\_path& path,

matrix\_t& bfsMatrix

) {

const auto size{ matrix.size() };

auto q{ std::queue<size\_t>{ } };

visited[start] = true;

q.push(start);

while (!q.empty()) {

const size\_t vertex{ q.front() };

q.pop();

auto neighbours{ std::vector<size\_t>{ } };

for (size\_t i{ 0 }; i < size; i++) {

if (!matrix[vertex][i] || visited[i]) continue;

neighbours.push\_back(i);

bfsMatrix[vertex][i] = 1;

visited[i] = true;

q.push(i);

}

path.push\_back({ vertex, neighbours });

}

return std::make\_pair(path, bfsMatrix);

}

std::ostream& operator<<(std::ostream& os, const graph::dfs\_path& path) {

const auto size{ path.size() };

for (size\_t i{ 0 }; i < size - 1; i++) {

os << path[i].first << " --> ";

}

os << path[size - 1].first;

return os;

}

std::ostream& operator<<(std::ostream& os, const graph::bfs\_path& path) {

for (const auto& [from, neighbours]: path) {

const auto size{ neighbours.size() };

os << from << " --> ";

if (size >= 1) {

for (size\_t i{ 0 }; i < size - 1; i++) {

os << neighbours[i] << ", ";

}

os << neighbours[size - 1];

}

os << std::endl;

}

return os;

}

Файл graph.tcc

#include <vector>

#include "graph.hpp"

#include "matrix.hpp"

using matrix::matrix\_t, graph::search\_t;

template<typename T>

std::pair<std::vector<T>, matrix\_t> graph::getAllPaths(

const matrix\_t& matrix,

size\_t start,

search\_t<T> search

) {

const auto size{ matrix.size() };

auto visited{ std::vector<bool>(size, false) };

auto paths{ std::vector<T>{ } };

paths.reserve(size - 1);

auto traversalTree{ matrix::initMatrix(size) };

auto hasUnvisited{ false };

auto startIndex{ start };

do {

hasUnvisited = false;

search(matrix, startIndex, visited, paths, traversalTree);

for (size\_t i{ 0 }; i < size; i++) {

if (visited[i]) continue;

hasUnvisited = true;

startIndex = i;

break;

}

} while (hasUnvisited);

return std::make\_pair(paths, traversalTree);

}

Файл main.cpp

#include <SFML/Graphics.hpp>

#include <iostream>

#include "config.hpp"

#include "utils.hpp"

#include "vertex.hpp"

#include "matrix.hpp"

#include "draw.hpp"

#include "graph.hpp"

using namespace std::placeholders;

using utils::events\_t, graph::dfs\_item, graph::bfs\_item;

void drawGraph(sf::RenderWindow& window, const matrix\_t& matrix) {

utils::clearWindow(window, config::BACKGROUND\_COLOR);

draw::drawGraph(window, matrix, config::SIDES, config::WIDTH);

window.display();

}

int main(int argc, char\* const argv[]) {

sf::RenderWindow window;

utils::manageWindow(window, config::WIDTH, config::HEIGHT, config::TITLE);

utils::clearWindow(window, config::BACKGROUND\_COLOR);

const auto directed{ matrix::adjacencyMatrix(config::VERTICES\_COUNT, config::SEED, config::k) };

std::cout << "Adjacency Matrix:\n" << directed << std::endl;

const size\_t start{ utils::getFistOutVertex(directed) };

if (start == SIZE\_MAX) {

std::cout << "All vertices are unconnected" << std::endl;

drawGraph(window, directed);

utils::pollEvents(window);

return 0;

}

const auto [dfs, dfsMatrix]{ graph::getAllPaths<dfs\_item>(directed, start, graph::dfs) };

const auto [bfs, bfsMatrix]{ graph::getAllPaths<bfs\_item>(directed, start, graph::bfs) };

std::cout << "DFS Matrix:\n" << dfsMatrix << std::endl;

std::cout << "BFS Matrix:\n" << bfsMatrix << std::endl;

std::cout << "Dfs:\n" << dfs << std::endl;

std::cout << "Bfs:\n" << bfs << std::endl;

std::cout << "DFS new numbering:\n";

utils::printNewVertexNumberingDFS(dfs);

std::cout << "BFS new numbering:\n";

utils::printNewVertexNumberingBFS(bfs);

const events\_t myEvents{

std::make\_tuple(

utils::onKeyDown(sf::Keyboard::Space),

draw::drawDFSRouteClosure(directed, dfs),

"DFS"

),

std::make\_tuple(

utils::onKeyDown(sf::Keyboard::Space),

draw::drawBFSRouteClosure(directed, bfs),

"BFS"

),

};

utils::pollEvents(window, myEvents, std::bind(drawGraph, \_1, directed));

return 0;

}

Файл matrix.cpp

#include <iostream>

#include <iomanip>

#include <vector>

#include <cmath>

#include <string>

#include "matrix.hpp"

using matrix::matrix\_t, matrix::row\_t;

float random(float min, float max) {

const auto r{ static\_cast<float>(rand()) / (RAND\_MAX + 1) };

return r \* (max - min) + min;

}

matrix\_t matrix::adjacencyMatrix(int size, int seed, float k) {

srand(seed);

matrix\_t result(size);

for (size\_t i{ 0 }; i < size; i++) {

row\_t row(size);

for (size\_t j{ 0 }; j < size; j++) {

const int value{ static\_cast<int>(floor(random(0.f, 2.f) \* k)) };

row[j] = value;

}

result[i] = row;

}

return result;

}

matrix\_t matrix::toUndirected(const matrix\_t &matrix) {

const auto size{ matrix.size() };

matrix\_t result(size);

for (size\_t i{ 0 }; i < size; i++) {

row\_t row(size);

result[i] = row;

for (size\_t j{ 0 }; j < i + 1; j++) {

result[i][j] = result[j][i] = matrix[i][j] || matrix[j][i];

}

}

return result;

}

std::ostream& operator<<(std::ostream& os, const matrix\_t& matrix) {

const auto length{ matrix.size() };

const auto indent{ std::to\_string(length).length() };

const auto width{ indent \* 2 };

os << std::setw(width) << ' ';

for (size\_t i{ 0 }; i < length; ++i) {

os << std::setw(width) << i << ' ';

}

os << std::endl << std::setw(width) << ' ';

os << std::setw((width + 1) \* length) << std::setfill('-') << '-';

os << std::setfill(' ') << std::endl;

for (size\_t i{ 0 }; i < length; ++i) {

os << std::setw(indent) << i << " |";

for (size\_t j{ 0 }; j < length; ++j) {

os << std::setw(width) << matrix[i][j] << ' ';

}

os << std::endl;

}

return os;

}

Файл utils.cpp

#include <SFML/Graphics.hpp>

#include <functional>

#include <set>

#include <iostream>

#include "config.hpp"

#include "utils.hpp"

using utils::events\_t;

using pair\_set = std::set<std::pair<size\_t, size\_t>, decltype([](const auto& x, const auto& y) {

return x.first < y.first;

})>;

sf::RenderWindow& utils::manageWindow(

sf::RenderWindow& window, unsigned width, unsigned height, const char\* title

) {

sf::ContextSettings settings;

settings.antialiasingLevel = config::SMOOTHING;

window.create(

sf::VideoMode{ width, height },

title,

sf::Style::Default,

settings

);

window.setKeyRepeatEnabled(false);

return window;

}

size\_t utils::getFistOutVertex(const matrix\_t& matrix) {

const size\_t size{ matrix.size() };

for (size\_t i{ 0 }; i < size; i++) {

for (size\_t j{ 0 }; j < size; j++) {

if (matrix[i][j] && i != j) return i;

}

}

return SIZE\_MAX;

}

std::function<bool(const sf::Event&)> utils::onKeyDown(const sf::Keyboard::Key& key) {

return [&key](const sf::Event& event) {

return (

event.type == sf::Event::KeyPressed &&

event.key.code == key

);

};

}

void utils::clearWindow(sf::RenderWindow& window, const sf::Color& color) {

window.clear(config::BACKGROUND\_COLOR);

window.display();

}

sf::Font utils::getFont(const std::string& path) {

static sf::Font font;

static bool isDefined = false;

if (isDefined) return font;

if (!font.loadFromFile(path)) {

throw std::runtime\_error{ "Cannot load font!" };

}

isDefined = true;

return font;

}

const std::vector globalEvents {

std::make\_pair(

[](const sf::Event& event) {

const bool first{ event.type == sf::Event::Closed };

const bool second{

event.type == sf::Event::KeyPressed &&

event.key.code == sf::Keyboard::Escape

};

return first || second;

},

[](sf::RenderWindow& window){

window.close();

}

),

};

void leftCornerText(sf::RenderWindow& window, const std::string& txt) {

const sf::Font font{ utils::getFont() };

sf::Text text{ txt, font, config::TEXT\_SIZE };

text.setFillColor(config::LINE\_COLOR);

const sf::Vector2f pos{ 5.f, 5.f };

text.setPosition(pos - text.getGlobalBounds().getPosition());

window.draw(text);

window.display();

}

void utils::pollEvents(sf::RenderWindow& window) {

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

for (const auto &[triger, callback] : globalEvents) {

if (triger(event)) callback(window);

}

}

}

}

void printNewVertexNumbering(const pair\_set& set) {

for (const auto& [oldIndex, newIndex]: set) {

std::cout << "Vertex Index: " << oldIndex

<< ", the number of the vertex in the detour: " << newIndex << std::endl;

}

}

void utils::printNewVertexNumberingDFS(const dfs\_path& path) {

const auto size{ path.size() };

auto set{ pair\_set{ } };

for (size\_t i{ 0 }; i < size; i++) {

const auto item{ path[i] };

if (!item.second) continue;

set.insert({ item.first, i });

}

printNewVertexNumbering(set);

}

void utils::printNewVertexNumberingBFS(const bfs\_path& path) {

const auto size{ path.size() };

auto set{ pair\_set{ } };

auto index{ size\_t{ 0 } };

if (size) set.insert({ path[0].first, index });

for (const auto &[vertex, neighbours]: path) {

index++;

for (const auto& neighbour: neighbours) {

set.insert({ neighbour, index });

index++;

}

}

printNewVertexNumbering(set);

}

void utils::pollEvents(

sf::RenderWindow& window,

const events\_t& events,

const std::function<void(sf::RenderWindow&)>& reset

) {

reset(window);

size\_t screen{ 0 };

const auto eventChangeTriger{ onKeyDown(sf::Keyboard::Right) };

auto [triger, callback, txt]{ events[screen] };

leftCornerText(window, txt);

while (window.isOpen()) {

sf::Event event;

while (window.pollEvent(event)) {

for (const auto &[triger, callback] : globalEvents) {

if (triger(event)) callback(window);

}

if (triger(event)) callback(window, false);

if (eventChangeTriger(event)) {

reset(window);

callback(window, true);

screen = (screen + 1) % events.size();

triger = std::get<0>(events[screen]);

callback = std::get<1>(events[screen]);

txt = std::get<2>(events[screen]);

leftCornerText(window, txt);

}

}

}

}

Файл vertex.cpp

#define \_USE\_MATH\_DEFINES

#include <SFML/Graphics.hpp>

#include <functional>

#include <string>

#include <cmath>

#include "config.hpp"

#include "vertex.hpp"

#include "utils.hpp"

using sf::RenderWindow, sf::Vector2f, std::string, vertex::Vertex;

const auto PI{ static\_cast<float>(M\_PI) };

std::pair<float, float> rotate(float x, float y, float l, float fi) {

return std::make\_pair(

x + l \* cos(fi),

y + l \* sin(fi)

);

}

float toDegrees(float radians) {

return radians \* 180 / PI;

}

float distance(float x1, float y1, float x2, float y2) {

return sqrtf((x2 - x1) \* (x2 - x1) + (y2 - y1) \* (y2 - y1));

}

std::function<Vector2f(size\_t)> bezierCurve(

const Vector2f& p1,

const Vector2f& p2,

const Vector2f& p3,

int items

) {

const auto step{ 1.f / items };

return [&p1, &p2, &p3, step](size\_t i) {

const auto t{ step \* i };

const auto t1{ 1 - t };

return t1 \* t1 \* p1 + 2 \* t1 \* t \* p2 + t \* t \* p3;

};

}

void drawCircle(RenderWindow& window, const Vector2f& posc, const sf::Color& color) {

const auto position { Vector2f{

posc.x - config::VERTEX\_RADIUS,

posc.y - config::VERTEX\_RADIUS,

} };

auto circle{ sf::CircleShape{ config::VERTEX\_RADIUS } };

circle.setPosition(position);

circle.setFillColor(config::BACKGROUND\_COLOR);

circle.setOutlineThickness(config::LINE\_WIDTH);

circle.setOutlineColor(color);

window.draw(circle);

}

void vertex::drawText(RenderWindow& window, const Vector2f& posc, const string& txt, const sf::Color& color) {

const auto font{ utils::getFont() };

auto text{ sf::Text{ txt, font, config::TEXT\_SIZE } };

text.setFillColor(color);

const auto r{ text.getGlobalBounds() };

text.setPosition(posc - r.getPosition() - r.getSize() / 2.f);

window.draw(text);

}

void vertex::draw(RenderWindow& window, const Vertex& vertex, const sf::Color& color) {

const auto vector{ Vector2f{ vertex.x, vertex.y } };

drawCircle(window, vector, color);

drawText(window, vector, std::to\_string(vertex.index), color);

}

void line(

RenderWindow& window,

const Vector2f& from,

const Vector2f& to,

const sf::Color& color

) {

const auto length{ distance(from.x, from.y, to.x, to.y) };

const auto fi{ atan2f(to.y - from.y , to.x - from.x) };

sf::RectangleShape rec{ { length, config::LINE\_WIDTH } };

rec.setFillColor(color);

rec.setOrigin({ 0, config::LINE\_WIDTH / 2 });

rec.setPosition(from.x, from.y);

rec.rotate(toDegrees(fi));

window.draw(rec);

}

void arrows(

RenderWindow& window,

float x,

float y,

float fi,

float delta,

const sf::Color& color

) {

const auto [lx, ly]{ rotate(x, y, config::ARROWS\_LENGTH, fi + delta) };

const auto [rx, ry]{ rotate(x, y, config::ARROWS\_LENGTH, fi - delta) };

line(window, { lx, ly }, { x, y }, color);

line(window, { x, y }, { rx, ry }, color);

}

void vertex::lineConnect(

RenderWindow& window,

const Vertex& from,

const Vertex& to,

bool shift,

bool dir,

const sf::Color& color

) {

const auto fi{ atan2f(to.y - from.y, to.x - from.x) };

const auto f1{ shift ? fi - PI / 8 : fi };

const auto f2{ shift ? fi + PI + PI / 8 : fi + PI };

const auto [x1, y1]{ rotate(from.x, from.y, config::VERTEX\_RADIUS + config::LINE\_WIDTH, f1) };

const auto [x2, y2]{ rotate(to.x, to.y, config::VERTEX\_RADIUS + config::LINE\_WIDTH, f2) };

line(window, { x1, y1 }, { x2, y2 }, color);

if (dir) arrows(window, x2, y2, fi + PI, PI / 8, color);

}

void vertex::arcConnect(

sf::RenderWindow& window,

const Vertex& from,

const Vertex& to,

bool dir,

const sf::Color& color

) {

const auto fi{ atan2f(to.y - from.y, to.x - from.x) };

const auto [x1, y1]{ rotate(from.x, from.y, config::VERTEX\_RADIUS + config::LINE\_WIDTH, fi - PI / 6) };

const auto [x2, y2]{ rotate(to.x, to.y, config::VERTEX\_RADIUS + config::LINE\_WIDTH, fi + PI + PI / 6) };

const auto dx{ x2 - x1 };

const auto dy{ y2 - y1 };

const auto height{ 2.f \* config::VERTEX\_RADIUS };

const auto length{ sqrtf(dx \* dx + dy \* dy) };

const auto parallel{ sf::Vector2f{ dy, -dx } / length };

const auto center{ sf::Vector2f{ (x1 + x2) / 2.f, (y1 + y2) / 2.f } };

const auto top{ center + height \* parallel };

const auto bezier{ bezierCurve({ x1, y1 }, top, { x2, y2 }, config::CURVE\_ITEMS) };

for (size\_t i{ 0 }; i < config::CURVE\_ITEMS; i++) {

line(window, bezier(i), bezier(i + 1), color);

}

if (dir) {

const auto f{ atan2f(top.y - y2, top.x - x2) };

arrows(window, x2, y2, f, PI / 8, color);

}

}

void vertex::loop(RenderWindow& window, const Vertex& vertex, bool dir, const sf::Color& color) {

const auto x{ vertex.x };

const auto y{ vertex.y - config::VERTEX\_RADIUS - config::LINE\_WIDTH };

const auto [x1, y1]{ rotate(x, y, config::VERTEX\_RADIUS, -PI / 4) };

const auto [x2, y2]{ rotate(x, y, config::VERTEX\_RADIUS, -3 \* PI / 4) };

line(window, { x, y }, { x1, y1 }, color);

line(window, { x1 + config::LINE\_WIDTH / 3, y1 }, { x2 - config::LINE\_WIDTH / 3, y2 }, color);

line(window, { x2, y2 }, { x, y }, color);

if (dir) arrows(window, x, y, PI / 4 + PI, PI / 8, color);

}

float calculateStep(float size, int count, int sides) {

const auto denominator{ ceilf(static\_cast<float>(count) / sides) + 1 };

return static\_cast<float>(size) / denominator;

}

const std::function<std::pair<float, float>(int, int, float, float)> cases[] {

[](int i, int sp, float st, float start) { return std::make\_pair(

start + st \* i, start

); },

[](int i, int sp, float st, float start) { return std::make\_pair(

start + st \* sp, start + st \* i

); },

[](int i, int sp, float st, float start) { return std::make\_pair(

start + st \* (sp - i), start + st \* sp

); },

[](int i, int sp, float st, float start) { return std::make\_pair(

start, start + st \* (sp - i)

); }

};

std::function<Vertex(size\_t)> vertex::getVertexClosure(size\_t count, size\_t sides, int width) {

const auto split{ static\_cast<int>(ceilf(static\_cast<float>(count) / sides)) };

const auto step{ calculateStep(width, count, sides) };

const auto start{ step / 2.f };

return [split, step, start](size\_t index) {

const auto side{ static\_cast<int>(floorf(static\_cast<float>(index) / split)) };

const auto [x, y]{ cases[side](index % split, split, step, start) };

return Vertex{ x, y, index };

};

}

**Згенерована матриця сумiжностi напрямленого графа:**

0 1 2 3 4 5 6 7 8 9 10

--------------------------------------------------

0 | 0 0 0 0 1 0 1 1 0 0 0

1 | 0 0 0 0 0 0 1 1 1 1 0

2 | 1 1 0 0 0 1 0 1 0 0 0

3 | 0 0 1 1 1 0 0 0 0 0 1

4 | 1 1 0 1 1 0 1 1 0 1 0

5 | 0 1 1 0 1 0 0 1 0 1 0

6 | 1 0 0 0 0 0 0 0 1 0 0

7 | 1 1 1 0 0 0 0 0 0 0 0

8 | 1 0 1 1 1 0 0 0 0 0 0

9 | 1 0 0 1 1 0 0 1 1 0 1

10 | 0 1 0 0 0 1 1 0 0 0 0

**Матриця сумiжностi дерева обходу**

DFS:

0 1 2 3 4 5 6 7 8 9 10

--------------------------------------------------

0 | 0 0 0 0 1 0 0 0 0 0 0

1 | 0 0 0 0 0 0 1 0 0 0 0

2 | 0 0 0 0 0 1 0 0 0 0 0

3 | 0 0 0 0 0 0 0 0 0 0 1

4 | 0 1 0 0 0 0 0 0 0 0 0

5 | 0 0 0 0 0 0 0 1 0 1 0

6 | 0 0 0 0 0 0 0 0 1 0 0

7 | 0 0 0 0 0 0 0 0 0 0 0

8 | 0 0 1 0 0 0 0 0 0 0 0

9 | 0 0 0 1 0 0 0 0 0 0 0

10 | 0 0 0 0 0 0 0 0 0 0 0

BFS:

0 1 2 3 4 5 6 7 8 9 10

--------------------------------------------------

0 | 0 0 0 0 1 0 1 1 0 0 0

1 | 0 0 0 0 0 0 0 0 0 0 0

2 | 0 0 0 0 0 1 0 0 0 0 0

3 | 0 0 0 0 0 0 0 0 0 0 1

4 | 0 1 0 1 0 0 0 0 0 1 0

5 | 0 0 0 0 0 0 0 0 0 0 0

6 | 0 0 0 0 0 0 0 0 1 0 0

7 | 0 0 1 0 0 0 0 0 0 0 0

8 | 0 0 0 0 0 0 0 0 0 0 0

9 | 0 0 0 0 0 0 0 0 0 0 0

10 | 0 0 0 0 0 0 0 0 0 0 0

**Список (вектор) вiдповiдностi номерiв вершин i їх нової нумерацiї, набутої в процесi обходу**

DFS:

Vertex Index: 0, the number of the vertex in the detour: 0

Vertex Index: 1, the number of the vertex in the detour: 2

Vertex Index: 2, the number of the vertex in the detour: 5

Vertex Index: 3, the number of the vertex in the detour: 10

Vertex Index: 4, the number of the vertex in the detour: 1

Vertex Index: 5, the number of the vertex in the detour: 6

Vertex Index: 6, the number of the vertex in the detour: 3

Vertex Index: 7, the number of the vertex in the detour: 7

Vertex Index: 8, the number of the vertex in the detour: 4

Vertex Index: 9, the number of the vertex in the detour: 9

Vertex Index: 10, the number of the vertex in the detour: 11

BFS:

Vertex Index: 0, the number of the vertex in the detour: 0

Vertex Index: 1, the number of the vertex in the detour: 5

Vertex Index: 2, the number of the vertex in the detour: 11

Vertex Index: 3, the number of the vertex in the detour: 6

Vertex Index: 4, the number of the vertex in the detour: 1

Vertex Index: 5, the number of the vertex in the detour: 18

Vertex Index: 6, the number of the vertex in the detour: 2

Vertex Index: 7, the number of the vertex in the detour: 3

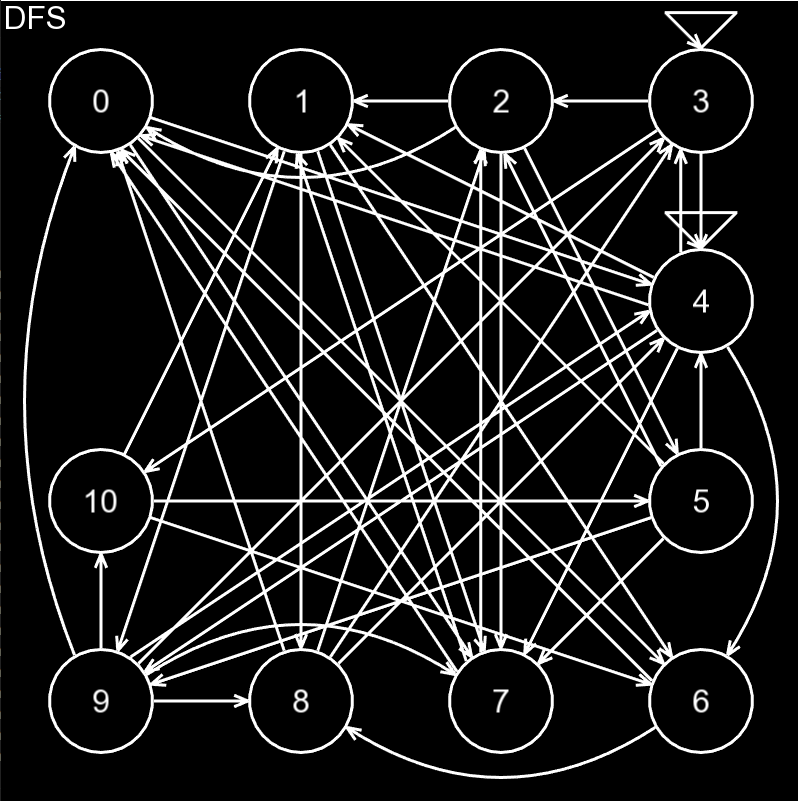
Vertex Index: 8, the number of the vertex in the detour: 9

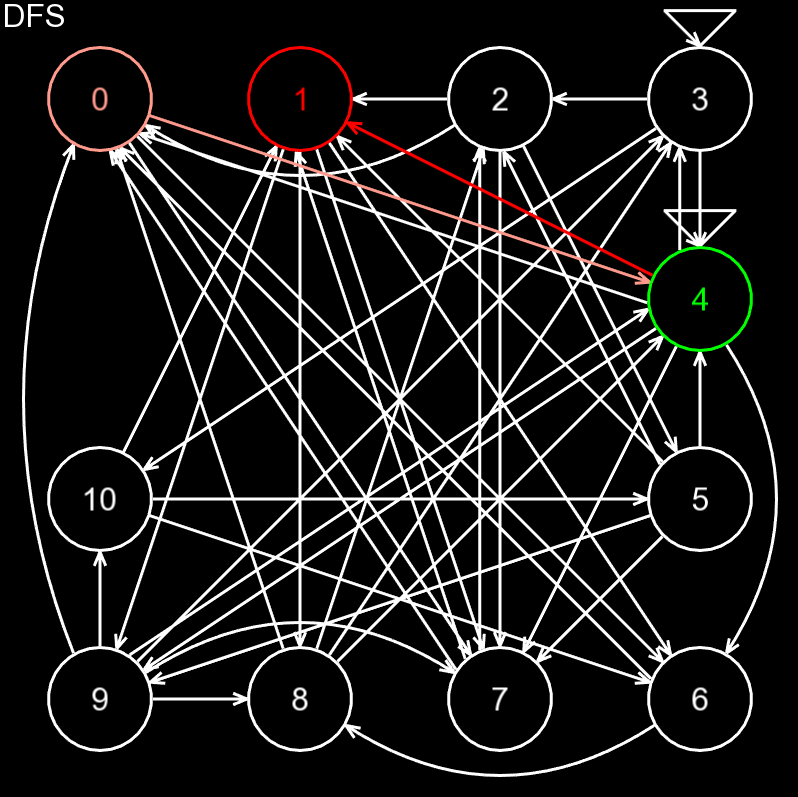
Vertex Index: 9, the number of the vertex in the detour: 7

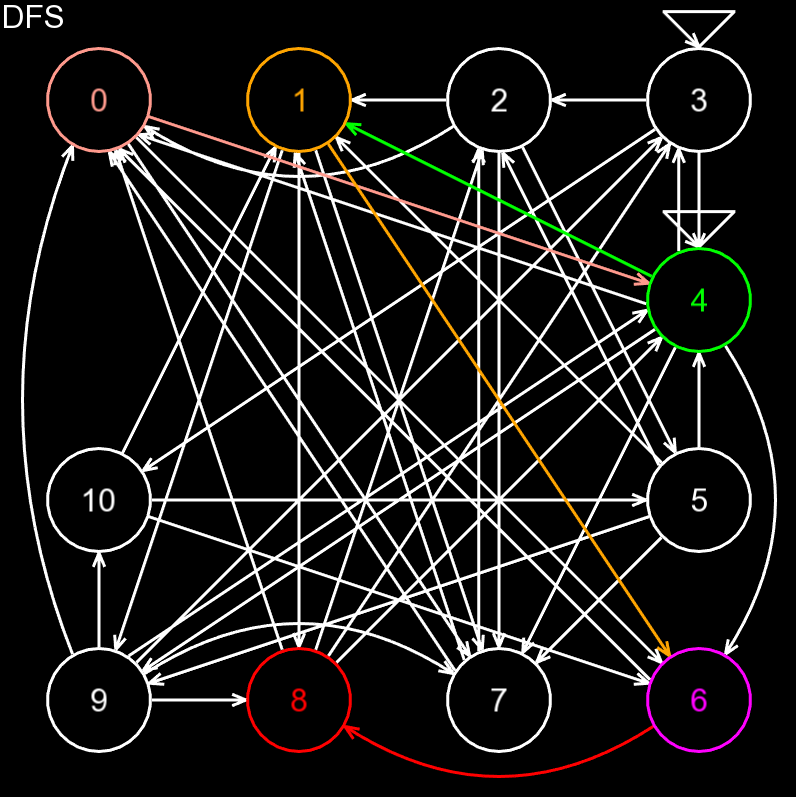
Vertex Index: 10, the number of the vertex in the detour: 14

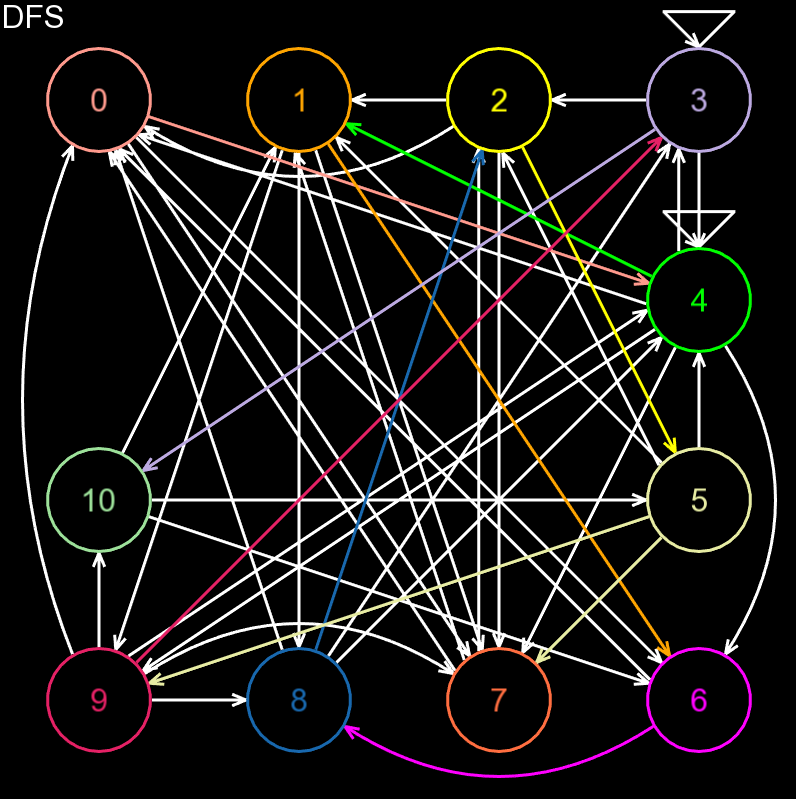
**Скриншоти зображення графа та дерева обходу**

DFS:

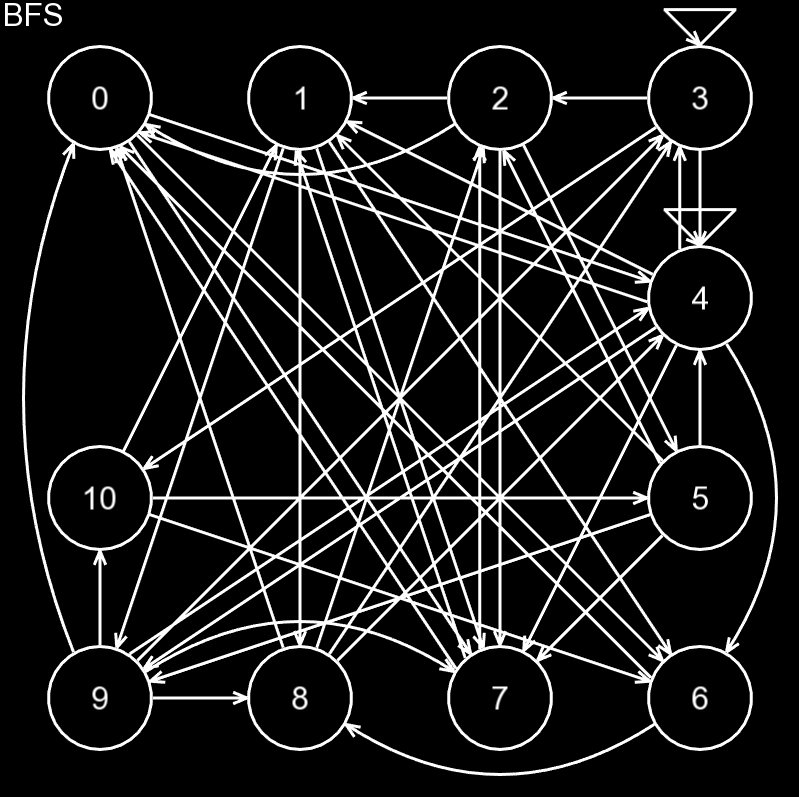


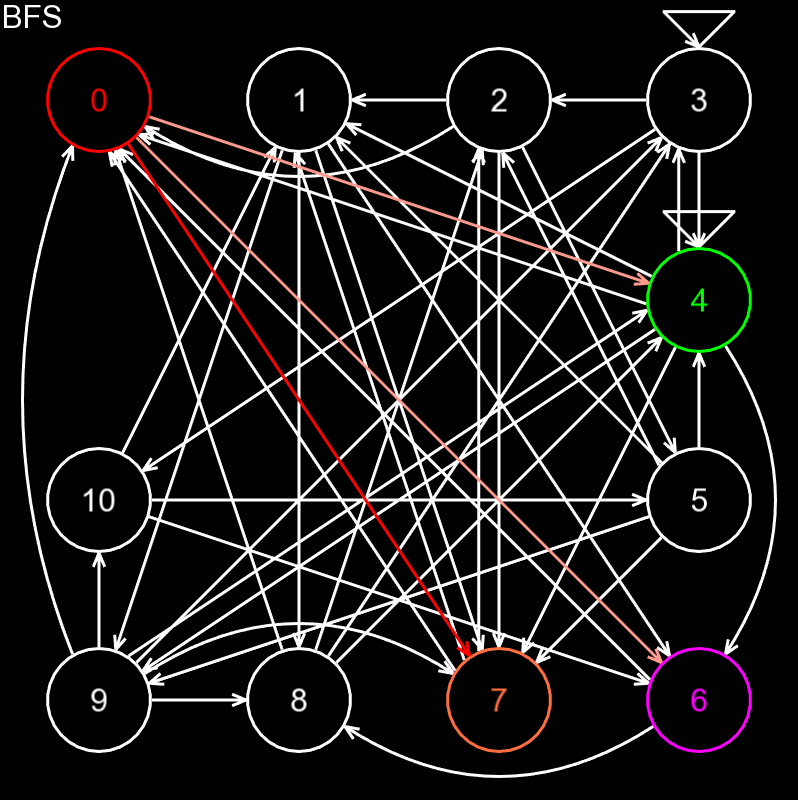


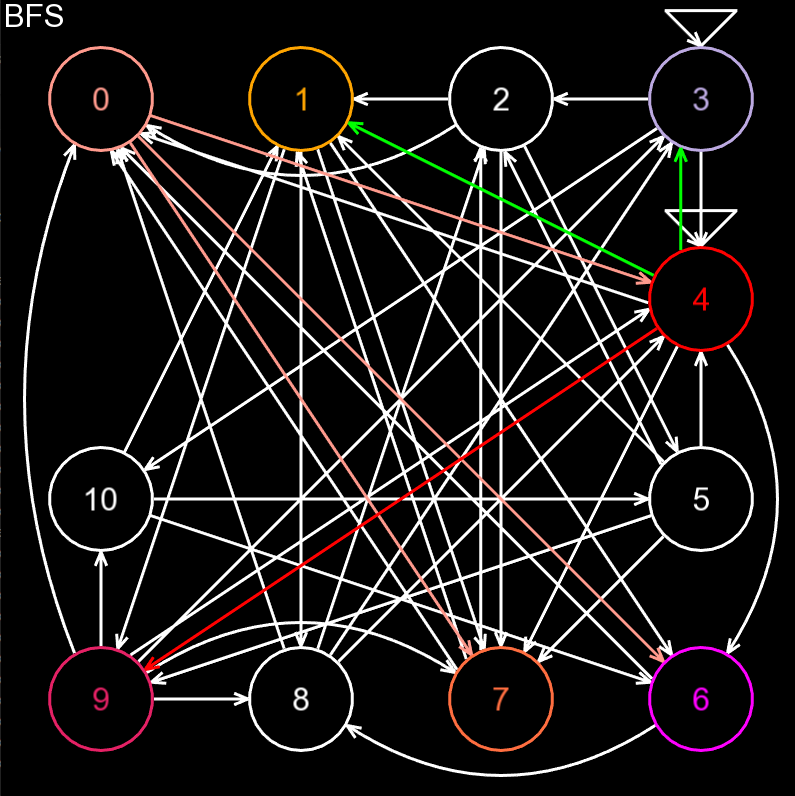


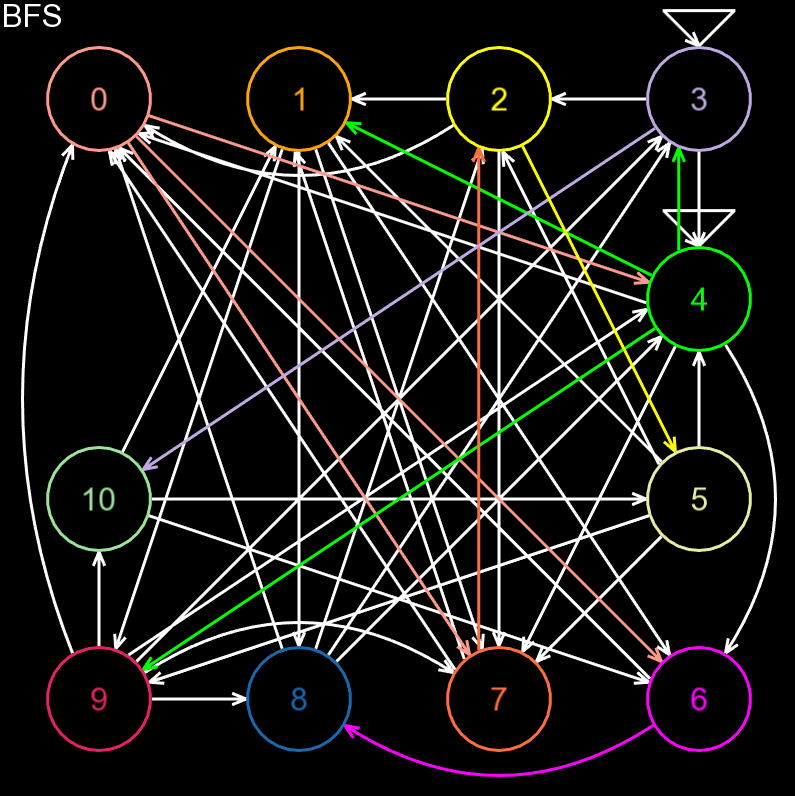


BFS:









**Висновки**

Протягом виконання лабораторної роботи я опанував методи дослідження графа за допомогою обходу його вершин в глибину та в ширину. Я написав програму, яка графічно, крок за кроком, відображає обхід графа вглибину та в ширину за матрицею суміжності. Для цього було розроблено відповідні функції обходу графа, причому пошук в глибину був реалізований двума способами – ітеративним та рекурсивним. Ці способи обходу є досить корисними, адже вони лежать в основі більш складних алгоритмів на графах. Проте, самі по собі вони теж можуть бути використані на практиці, наприклад, використовуючи будь-який вид обходу, можна визначити компоненту зв'язності, до якої належить вершина, з якої починався обхід.

Як результат виконання лабораторної роботи, я закріпив знання про графи, їх структуру та властивості, навчився реалізовувати основні способи обходу графів та покращив навички в програмуванні алгоритмів.