**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 №6**

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 1**: коефіцієнт k = 1.0 – n3 \* 0.01 – n4 \* 0.005 – 0.05.

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.

**Відмінність 2:** матриця ваг W формується таким чином.

1. матриця B розмiром n ‧ n заповнюється згенерованими випадковими числами в дiапазонi [0, 2.0) (параметр генератора випадкових чисел той же самий, n1n2n3n4);
2. одержується матриця C:

1. одержується матриця D, у якiй

якщо ,

якщо

1. одержується матриця H, у якiй

якщо ,

та в іншому випадку;

1. Tr — верхня трикутна матриця з одиниць ();
2. матриця ваг W симетрична, i її елементи одержуються за формулою: .
3. Створити програму для знаходження мiнiмального кiстяка за алгоритмом Краскала при n4 — парному i за алгоритмом Прiма — при непарному. При цьому у програмi.

* графи представляти у виглядi динамiчних спискiв, обхiд графа, додавання, вiднiмання вершин, ребер виконувати як функцiї з вершинами вiдповiдних спискiв;
* у програмi виконання обходу вiдображати покроково, черговий крок виконувати за натисканням кнопки у вiкнi або на клавiатурi;

1. Пiд час обходу графа побудувати дерево його кiстяка. У програмi дерево кiстяка виводити покроково у процесi виконання алгоритму. 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.

При зображеннi як графа, так i його кiстяка, вказати ваги ребер.

**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 <SFML/Graphics/Color.hpp>

#include <string>

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 unsigned LABEL\_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 "vertex.hpp"

#include "graph.hpp"

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

namespace draw {

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

std::function<void(sf::RenderWindow&)> skeletonClosure(

const matrix\_t& matrix,

const mst\_t& path,

size\_t sides,

int size

);

}

Файл headers/graph.hpp

#pragma once

#include <vector>

#include "matrix.hpp"

using matrix::matrix\_t;

namespace graph {

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

std::tuple<matrix\_t, mst\_t, size\_t> kruskal(const matrix\_t& weighted);

}

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);

matrix\_t weightedMatrix(const matrix\_t& a, int seed);

}

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

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

);

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);

}

Файл headers/vertex.hpp

#pragma once

#include <SFML/Graphics.hpp>

#include <functional>

#include <string>

#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::Vertex& from,

const vertex::Vertex& to,

const std::string& txt,

bool shift = false,

bool dir = true,

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

);

void arcConnect(

sf::RenderWindow& window,

const Vertex& from,

const Vertex& to,

const std::string& txt,

bool dir,

const sf::Color& color

);

void loop(

sf::RenderWindow& window,

const Vertex& vertex,

const std::string& txt,

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 unsigned LABEL\_SIZE{ 16 };

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 - 0.05f };

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 <functional>

#include <memory>

#include <string>

#include <cmath>

#include "draw.hpp"

#include "vertex.hpp"

#include "matrix.hpp"

#include "graph.hpp"

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

using namespace std::placeholders;

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() };

const std::string str{ std::to\_string(matrix[i][j]) };

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

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

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

}

else {

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

vertex::lineConnect(window, from, to, str, 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, false)

};

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

const auto vertex{ getVertex(i) };

vertex::draw(window, vertex);

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

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

const auto otherVertex{ getVertex(j) };

connect(window, vertex, otherVertex);

}

}

}

std::function<void(sf::RenderWindow&)> draw::skeletonClosure(

const matrix\_t& matrix,

const mst\_t& path,

size\_t sides,

int windowSize

) {

const auto size{ matrix.size() };

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

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

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

return [&path, getVertex, connect, stepP](sf::RenderWindow& window) {

const auto step{ \*stepP };

if (step >= path.size()) return;

const auto [i, j]{ path[step] };

const auto from{ getVertex(i) };

const auto to{ getVertex(j) };

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

\*stepP += 1;

window.display();

};

};

Файл graph.cpp

#include <algorithm>

#include <vector>

#include <queue>

#include "matrix.hpp"

#include "graph.hpp"

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

auto minEdge(const matrix\_t& matrix) {

const auto size{ matrix.size() };

size\_t min{ SIZE\_MAX };

size\_t row{ SIZE\_MAX };

size\_t col{ SIZE\_MAX };

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

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

const auto item{ matrix[i][j] };

if (item == 0 || min <= item) continue;

min = item;

row = i;

col = j;

}

}

return std::make\_tuple(min, row, col);

}

bool hasLoop(const matrix\_t& matrix, size\_t start) {

const auto size{ matrix.size() };

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

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

visited[start] = true;

queue.push({ SIZE\_MAX, start });

while (!queue.empty()) {

const auto [from, vertex]{ queue.front() };

queue.pop();

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

if (from == i) continue;

if (matrix[vertex][i] && visited[i]) return true;

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

visited[i] = true;

queue.push({vertex, i});

}

}

return false;

}

std::tuple<matrix\_t, mst\_t, size\_t> graph::kruskal(const matrix\_t& weighted) {

const auto size{ weighted.size() };

matrix\_t matrix{ weighted };

mst\_t path{ };

path.reserve(size - 1);

matrix\_t graph(size);

std::generate(graph.begin(), graph.end(), [size]() { return row\_t(size); });

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

size\_t weight{ 0 };

while (std::find(visited.begin(), visited.end(), false) != visited.end()) {

const auto [min, row, col]{ minEdge(matrix) };

matrix[row][col] = matrix[col][row] = 0;

visited[row] = visited[col] = true;

graph[row][col] = graph[col][row] = min;

if (hasLoop(graph, row) || hasLoop(graph, col)) {

graph[row][col] = graph[col][row] = 0;

continue;

}

weight += min;

path.push\_back({ row, col });

}

return { graph, path, weight };

}

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 matrix::matrix\_t;

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, const char\* 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) };

const auto undirected{ matrix::toUndirected(directed) };

const auto weighted{ matrix::weightedMatrix(undirected, config::SEED) };

drawGraph(window, weighted);

const auto [mst, path, weight]{ graph::kruskal(weighted) };

std::cout << "Directed:\n" << directed << std::endl;

std::cout << "Weighted:\n" << weighted << std::endl;

std::cout << "Minimum spanning tree:\n" << mst << std::endl;

std::cout << "Sum of the weights of the minimum spanning tree: " << weight << std::endl;

const auto triger{ utils::onKeyDown(sf::Keyboard::Space) };

const auto event{ draw::skeletonClosure(weighted, path, config::SIDES, config::WIDTH) };

utils::pollEvents(window, triger, event);

return 0;

}

Файл matrix.cpp

#include <functional>

#include <iostream>

#include <iomanip>

#include <vector>

#include <string>

#include <cmath>

#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 auto 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;

}

template<typename T, typename U>

std::vector<std::vector<U>> map(

const std::vector<std::vector<T>>& matrix,

const std::function<U(T, size\_t, size\_t)>& mapFn

) {

const auto size{ matrix.size() };

std::vector<std::vector<U>> result(size);

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

std::vector<U> row(size);

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

row[j] = mapFn(matrix[i][j], i , j);

}

result[i] = row;

}

return result;

}

std::vector<std::vector<float>> randomMatrix(size\_t size) {

std::vector<std::vector<float>> result(size);

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

std::vector<float> row(size);

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

row[j] = random(0.0f, 2.0f);

}

result[i] = row;

}

return result;

}

matrix\_t matrix::weightedMatrix(const matrix\_t& a, int seed) {

const auto size{ a.size() };

const auto b{ randomMatrix(size) };

const auto c{ map<float, int>(b, [&a](const float& x, size\_t i, size\_t j) {

return static\_cast<int>(ceil(x \* 100 \* a[i][j]));

}) };

const auto d{ map<int, int>(c, [](const int& x, size\_t i, size\_t j) {

return x != 0;

}) };

const auto h{ map<int, int>(d, [&d](const int& x, size\_t i, size\_t j) {

return d[i][j] != d[j][i];

}) };

matrix\_t upperTriangle(size);

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

row\_t row(size);

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

row[j] = 1;

}

upperTriangle[i] = row;

}

matrix\_t w(size);

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

w[i] = row\_t(size);

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

w[i][j] = w[j][i] = (d[i][j] + h[i][j] \* upperTriangle[i][j]) \* c[i][j];

}

}

return w;

}

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 (int 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 (int i = 0; i < length; ++i) {

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

for (int 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 "config.hpp"

#include "utils.hpp"

using utils::events\_t;

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,

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

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

) {

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);

}

}

}

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));

}

float distance(const sf::Vector2f& p1, const sf::Vector2f& p2) {

return distance(p1.x, p1.y, p2.x, p2.y);

}

float length(const sf::Vector2f& vector) {

return sqrtf(vector.x \* vector.x + vector.y \* vector.y);

}

std::function<Vector2f(size\_t)> bezierCurve2Closure(

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,

} };

sf::CircleShape circle{ 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.setOrigin(r.getPosition() + r.getSize() / 2.f);

text.setPosition(posc);

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 signLine(

RenderWindow& window,

const sf::Vector2f& pos,

const sf::Vector2f& normal,

const std::string& txt,

float fi,

const sf::Color& color,

bool shift = false

) {

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

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

const auto r{ text.getGlobalBounds() };

text.setFillColor(color);

text.setOrigin(r.getPosition() + r.getSize() / 2.f);

text.rotate(toDegrees(fi));

if (shift) {

const auto direction{ sf::Vector2f{ -normal.y, normal.x } };

text.setPosition(pos + direction \* r.width);

}

else text.setPosition(pos);

text.move(normal \* (r.height + config::LINE\_WIDTH) / 2.f);

window.draw(text);

}

void vertex::lineConnect(

sf::RenderWindow& window,

const Vertex& from,

const Vertex& to,

const std::string& txt,

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);

const auto dx{ x2 - x1 };

const auto dy{ y2 - y1 };

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

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

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

const auto theta{ atanf(dy / dx) };

signLine(window, center, normal, txt, theta, color, true);

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

}

void vertex::arcConnect(

sf::RenderWindow& window,

const Vertex& from,

const Vertex& to,

const std::string& txt,

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 normal{ sf::Vector2f{ dy, -dx } / length };

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

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

const auto bezier{ bezierCurve2Closure({ 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);

}

const auto fl{ floorf(config::CURVE\_ITEMS / 2.f) };

const auto ce{ ceilf(config::CURVE\_ITEMS / 2.f) };

const auto position{ (bezier(fl) + bezier(ce)) / 2.f };

const auto theta{ atan(dy / dx) };

signLine(window, position, normal, txt, theta, 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,

const std::string& txt,

bool dir,

const sf::Color& color

) {

const auto radius{ config::VERTEX\_RADIUS };

const auto x{ vertex.x };

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

const auto [x1, y1]{ rotate(x, y, radius, -PI / 4) };

const auto [x2, y2]{ rotate(x, y, 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);

const auto lh{ fabs(x1 - x2 + 2.f \* config::LINE\_WIDTH / 3.f) };

const auto l{ sqrtf(radius \* radius - lh \* lh / 4.f) };

const auto position{ sf::Vector2f{ x, y - l } };

const auto normal{ sf::Vector2f{ 0.f, 1.f } };

signLine(window, position, normal, txt, 0.f, 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 1 0 0 0 0 1 1 1 1 0

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

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

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

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

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

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

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

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

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

**Згенерована матриця ваг графа:**

0 1 2 3 4 5 6 7 8 9 10

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

0 | 0 0 144 0 132 0 104 9 138 118 0

1 | 0 149 195 0 46 52 71 81 45 103 146

2 | 144 195 0 167 182 123 0 175 70 0 0

3 | 0 0 167 194 141 0 0 0 22 195 139

4 | 132 46 182 141 61 171 109 4 172 68 0

5 | 0 52 123 0 171 0 0 67 0 18 151

6 | 104 71 0 0 109 0 0 16 132 0 115

7 | 9 81 175 0 4 67 16 0 0 50 0

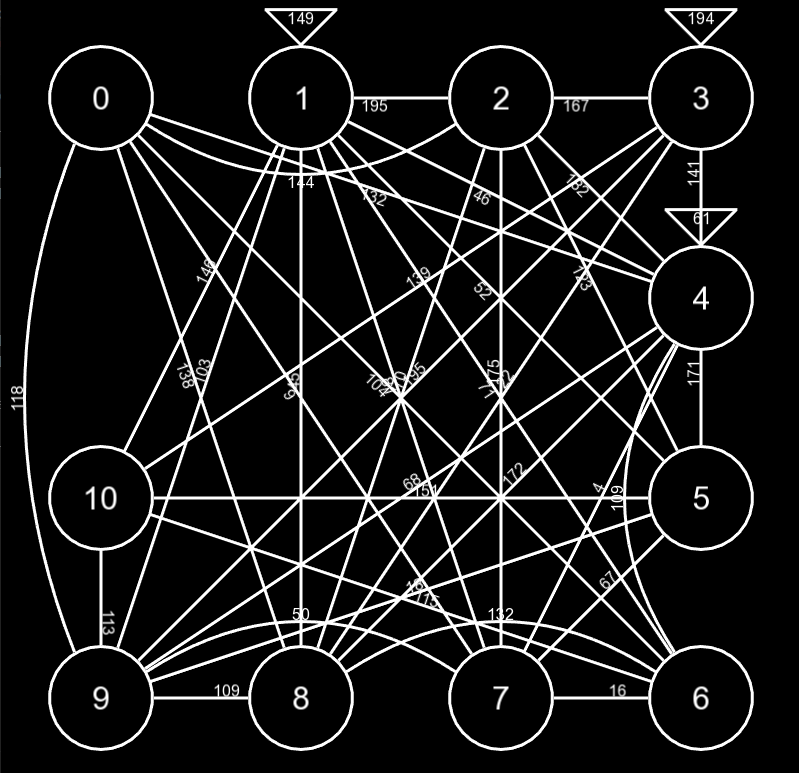
8 | 138 45 70 22 172 0 132 0 0 109 0

9 | 118 103 0 195 68 18 0 50 109 0 113

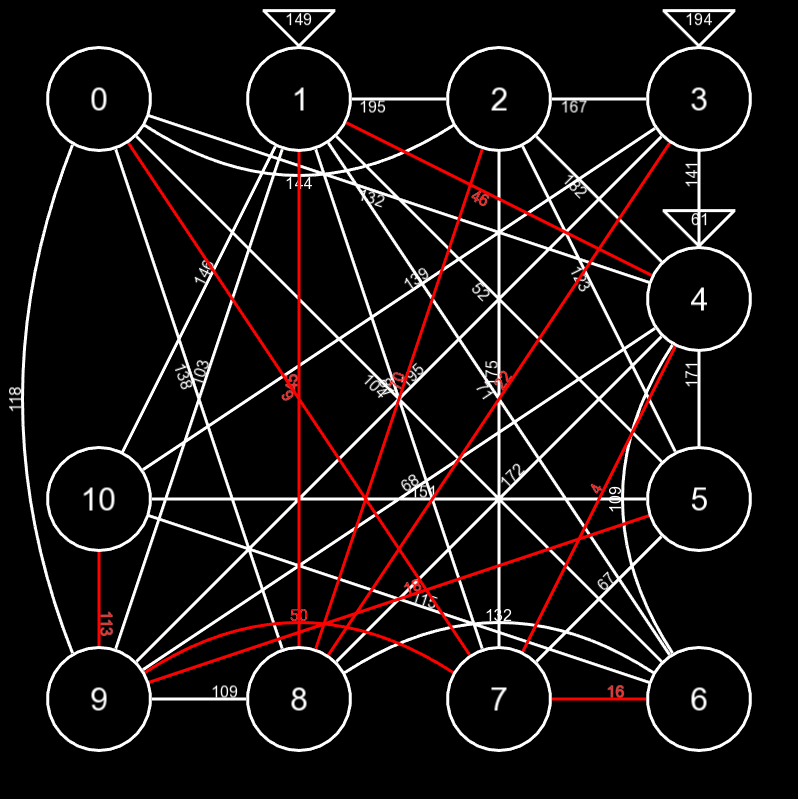
10 | 0 146 0 139 0 151 115 0 0 113 0

**Скриншоти зображення графа та його мінімального кістяка**

Граф:



Мінімальний кістяк:



**Сума ваг ребер знайденого мiнiмального кiстяка:**

Sum of the weights of the minimum spanning tree: 393

**Висновки**

Протягом виконання лабораторної роботи я вивчив методи розв’язання задачi знаходження мiнiмального кiстяка графа. Я створив програму, яка покроково відображає мінімальний кістяк графа. Для розв’язання даної задачі я ознайомився з двума алгоритмами - алгоритм Пріма та алгоритм Краскала. Алгоритм Краскала був реалізований на практиці. Особливістю йього алгоритму є те, що він на кожному кроці вибирає найменше ребро і, якщо воно не утворює цикл, додає його в мінімальний кістяк. Цей алгоритм є дуже корисним і часто використовується на практиці в сферах де потрібно економно прокласти шлях між вузлами.

Як результат виконання лабораторної роботи, я закріпив знання про графи, їх структуру та властивості, навчився реалізовувати задачу знаходження мінімального кістяка графу та покращив свої навички у створенні алгоритмів.