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

Перевірила:
Молчанова А. А.

Київ 2024

Постановка задачі

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

Відмінність: коефіцієнт $k = 1.0 - n_3 * 0.01 - n_4 * 0.05 - 0.15$.

Отже, матриця суміжності A_{dir} напрямленого графа за варіантом формується таким чином:

1) встановлюється параметр (seed) генератора випадкових чисел, рівний номеру варіанту $n_1n_2n_3n_4$;

2) матриця розміром n * n заповнюється згенерованими випадковими числами в діапазоні [0, 2.0);

3) обчислюється коефіцієнт $k = 1.0 - n_3 * 0.01 - n_4 * 0.05 - 0.15$, кожен елемент матриці множиться на коефіцієнт k;

4) елементи матриці округлюються: 0 — якщо елемент менший за 1.0, 1 — якщо елемент більший або дорівнює 1.0.

2. Створити програму, яка виконує обхід напрямленого графа вшир (BFS) та вглиб (DFS).

- обхід починати з вершини із найменшим номером, яка має щонайменше одну вихідну дугу;
- при обході враховувати порядок нумерації;
- у програмі виконання обходу відображати покроково, черговий крок виконувати за натисканням кнопки у вікні або на клавіатурі.

3. Під час обходу графа побудувати дерево обходу. У програмі дерево обходу виводити покроково у процесі виконання обходу графа. Це можна виконати одним із двох способів:

- або виділяти іншим кольором ребра графа;
- або будувати дерево обходу поряд із графом.

4. Зміну статусів вершин у процесі обходу продемонструвати зміною кольорів вершин, графічними позначками тощо, або ж у процесі обходу виводити протокол обходу у графічне вікно або в консоль.

5. Якщо після обходу графа лишилися невідвідані вершини, продовжувати обхід з невідвіданої вершини з найменшим номером, яка має щонайменше

одну вихідну 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 суміжності 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 суміжності 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) відповідності номерів вершин і їх нової нумерації, набутої в процесі 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
```

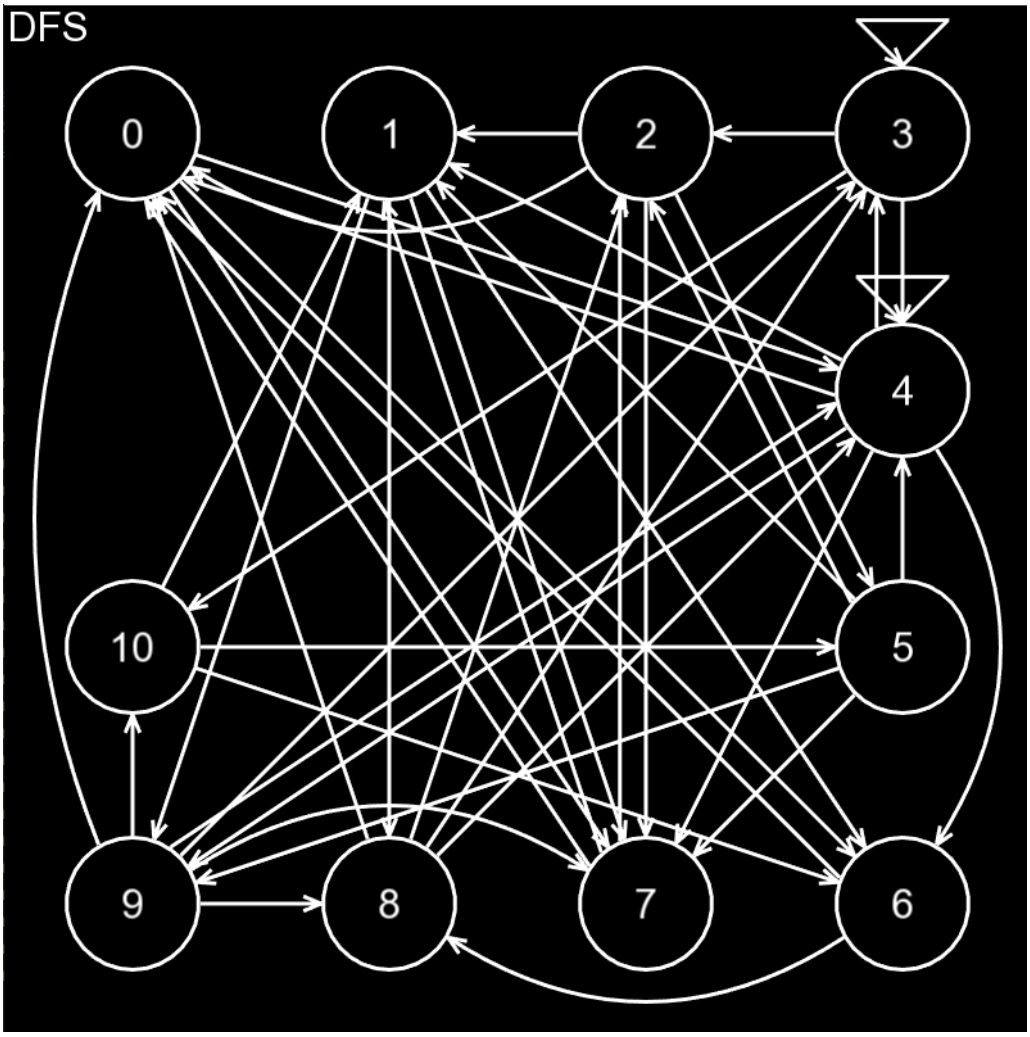
BFS:

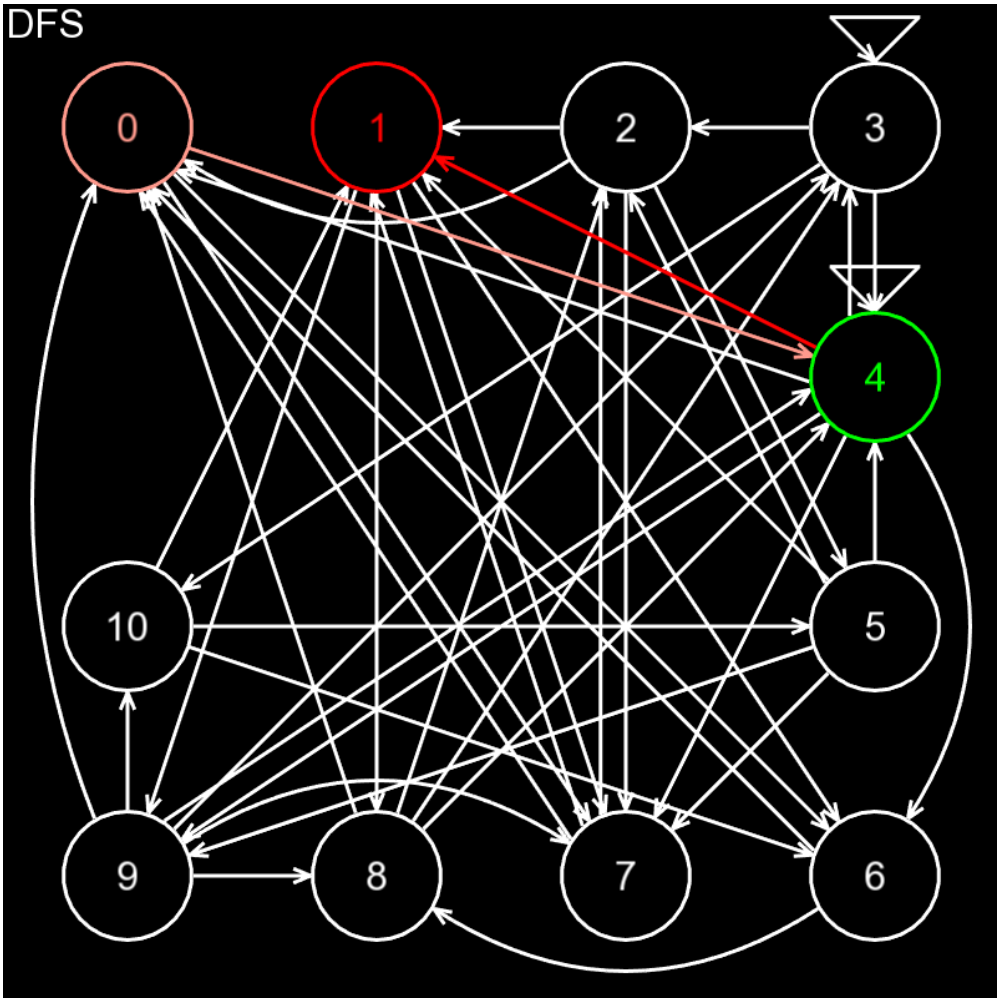
```
Vertex Index: 0, the number of the vertex in the detour: 0
Vertex Index: 1, the number of the vertex in the detour: 5
Vertex Index: 2, the number of the vertex in the detour: 11
Vertex Index: 3, the number of the vertex in the detour: 6
```

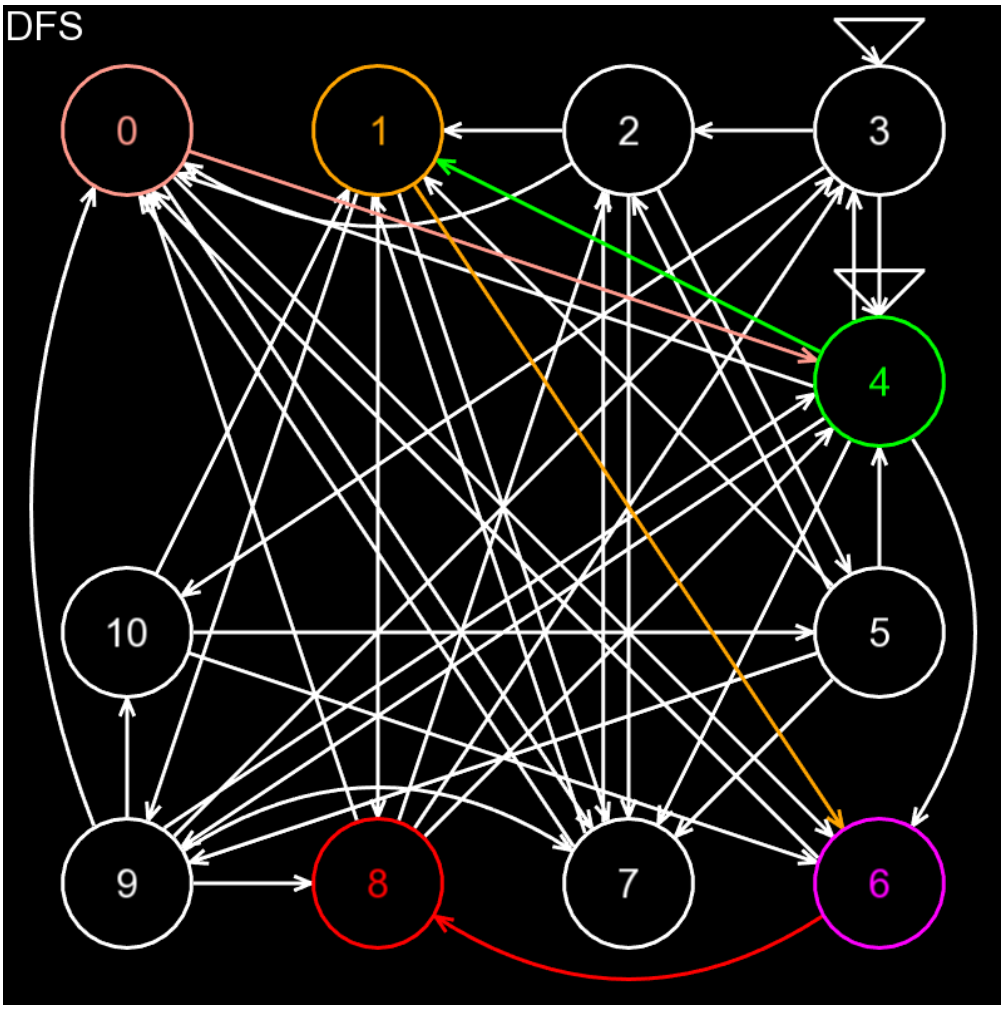
```
Vertex Index: 4, the number of the vertex in the detour: 1
Vertex Index: 5, the number of the vertex in the detour: 18
Vertex Index: 6, the number of the vertex in the detour: 2
Vertex Index: 7, the number of the vertex in the detour: 3
Vertex Index: 8, the number of the vertex in the detour: 9
Vertex Index: 9, the number of the vertex in the detour: 7
Vertex Index: 10, the number of the vertex in the detour: 14
```

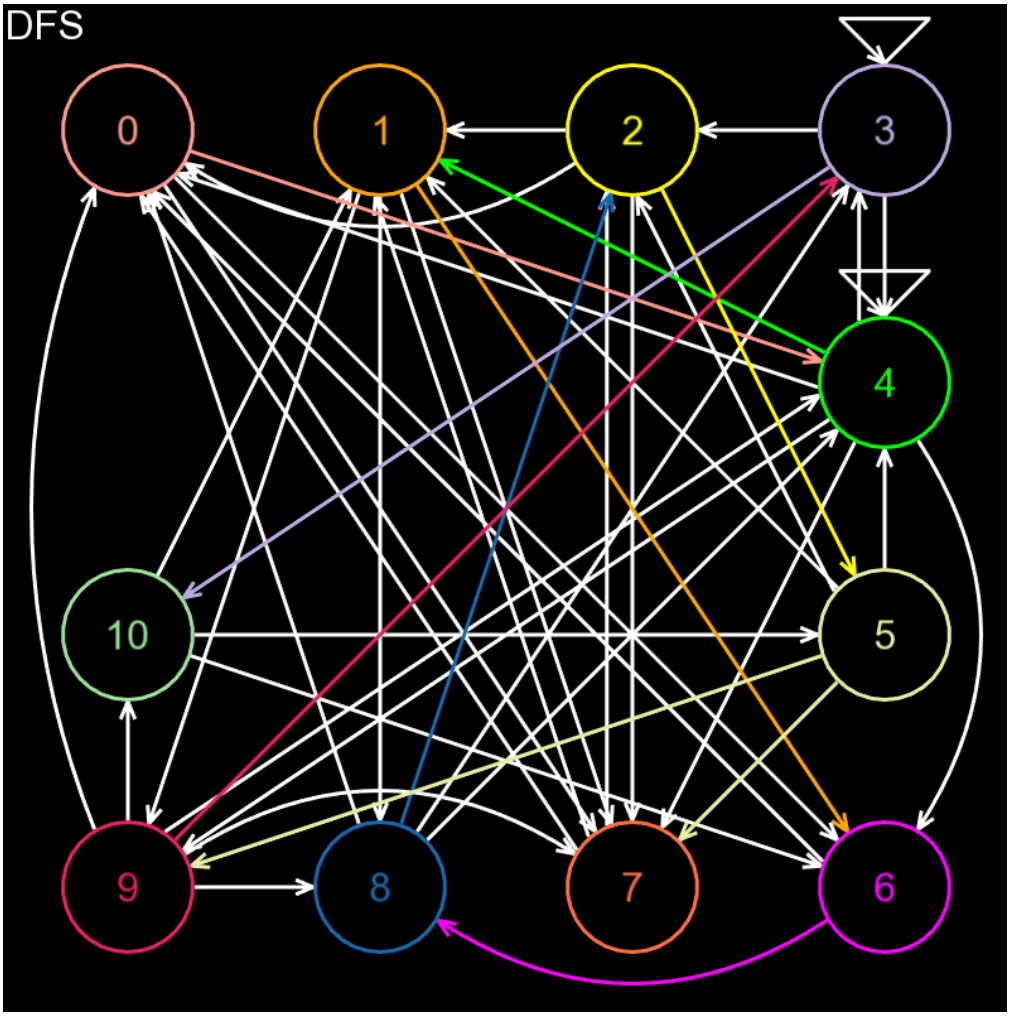
Скриншоти зображення графа та дерева обходу

DFS:

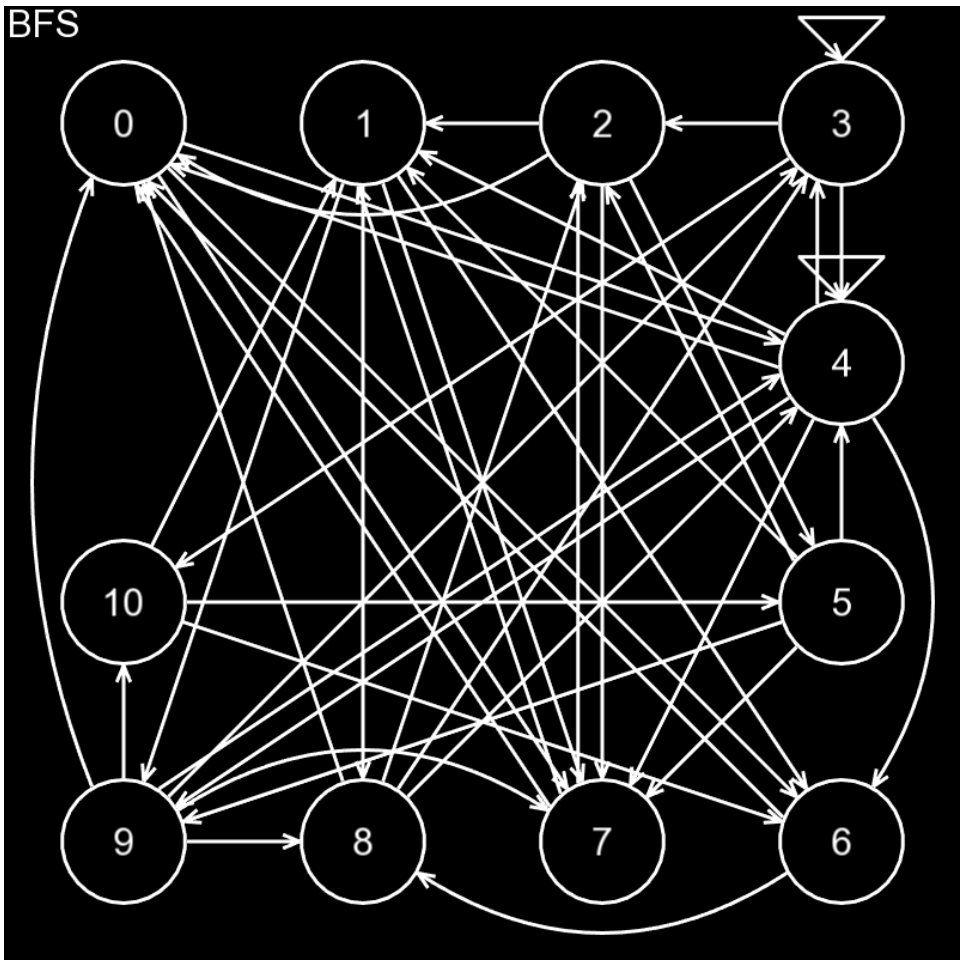


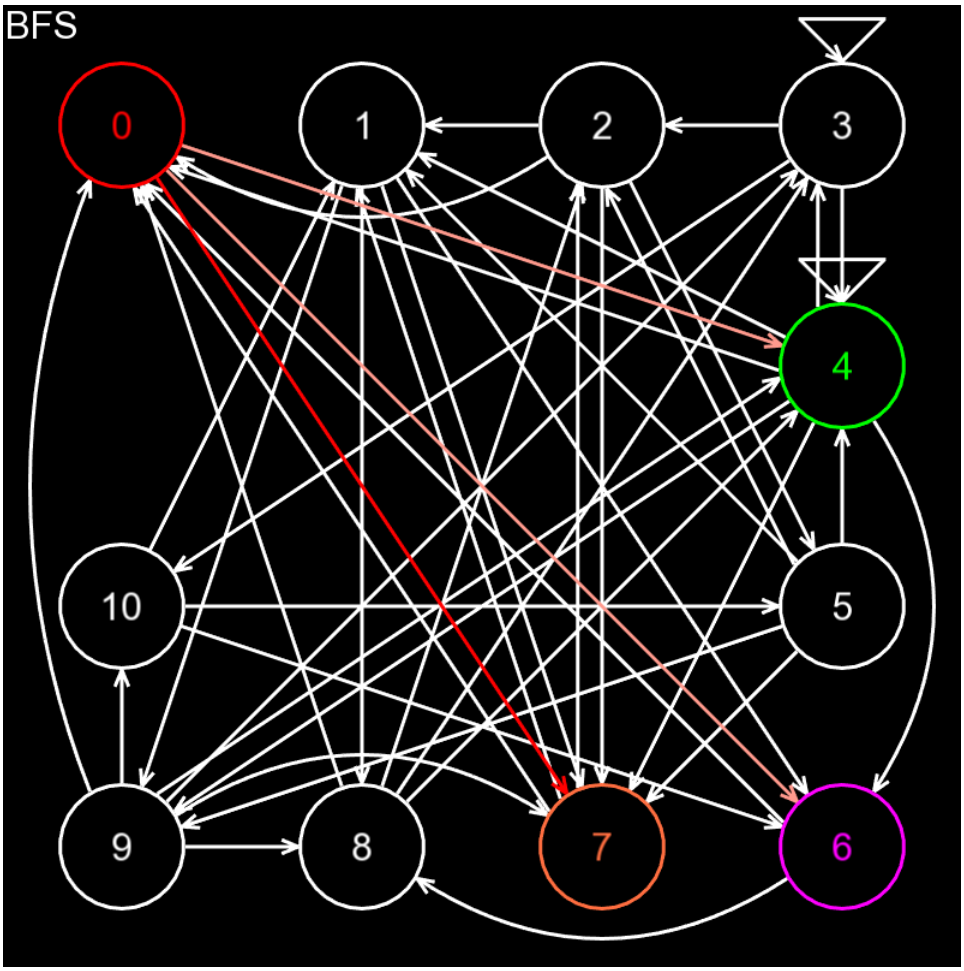

DFS
0
1
2
3
4
10
5
9
8
7
6

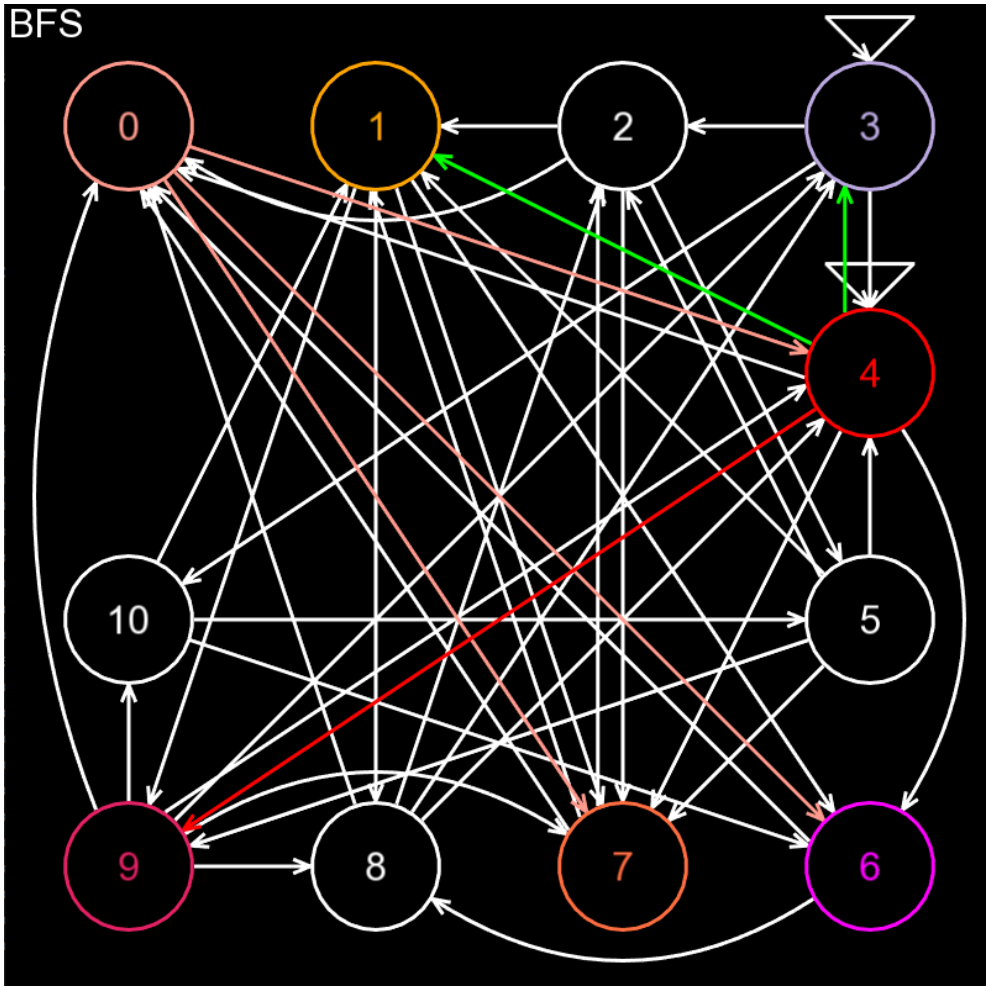

DFS
0
1
2
3
4
10
5
9
8
7
6

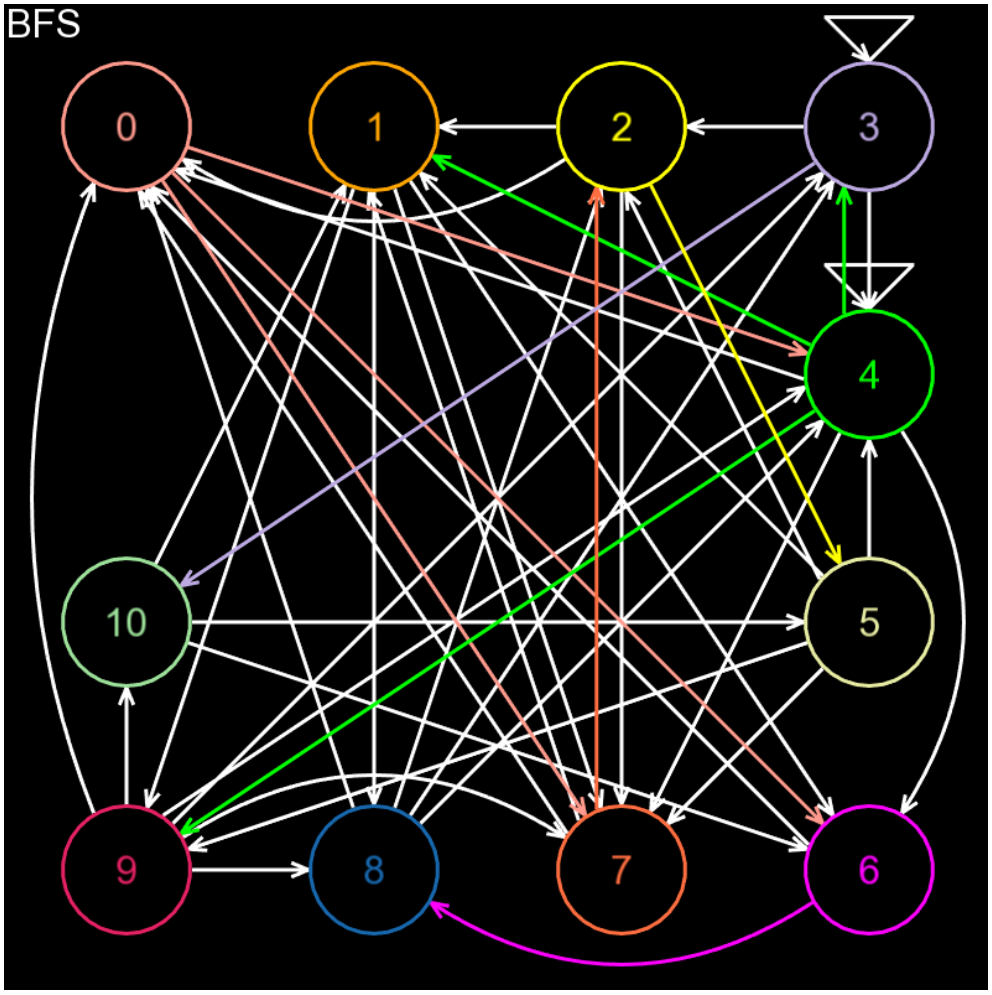

DFS
0
1
2
3
4
10
5
9
8
7
6

BFS:






BFS
0
1
2
3
4
10
5
9
8
7
6


BFS
0
1
2
3
4
10
5
9
8
7
6

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.