МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Институт математики и информационных систем
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Дата сдачи на проверку:

«__»_____ 2025 г.

Проверено:

«__»_____ 2025 г.

ГРАФЫ. СВЯЗНОСТЬ ГРАФОВ.

Отчёт по лабораторной работе №5

по дисциплине

«Дискретная математика»

Разработал студент гр. ИВТб-1301-05-00 _____/Черкасов А. А./

(подпись)

Проверила преподаватель _____/Пахарева И. В./

(подпись)

Работа защищена «__»_____ 2025 г.

Киров

2025

Цель

Цель работы: изучить представление ориентированного графа с помощью матрицы инцидентности, освоить преобразование этой матрицы в матрицу смежности, сформировать матрицу достижимости (матрицу связности) и проверить условие односторонней связности заданного графа.

Задание

– Ориентированный граф G задаётся матрицей инцидентности, записанной в файле `input.txt`. Ограничить размерность: число вершин и число дуг (ребер) ≥ 4.

– Сформировать из матрицы инцидентности матрицу смежности A.

– По матрице смежности A построить матрицу достижимости R (матрицу связности) методом последовательного умножения или алгоритмом Вархалла–Флойда.

– На основании матрицы достижимости определить, является ли граф односторонне связным: для любой пары вершин u, v должно выполняться $\big(R_{uv} = 1\big)$ или $\big(R_{vu} = 1\big)$.

Решение

Вывод

В ходе выполнения лабораторной работы были отработаны навыки работы с матрицей инцидентности для представления ориентированного графа и преобразования её в матрицу смежности. Далее с помощью алгоритма построения матрицы достижимости (матрицы связности) удалось проверить, является ли заданный граф односторонне связным. Реализованная программа корректно читает матрицу инцидентности из файла `input.txt`, формирует матрицу смежности, вычисляет матрицу достижимости и делает заключение о наличии односторонней связности. Работа способствовала пониманию фундаментальных операций над графами и закреплению алгоритмов поиска путей в ориентированных графах.

Приложение А1. Исходный код

```
// src/main.cpp
#include "graphviz.hpp"
#include <cstdlib>
#include <cstring>
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>

static const std::string INPUT_FILE = "input.txt";

bool read_incidence_matrix(int &n_vertices, int &n_edges,
                           std::vector<std::vector<int>> &inc_mat) {
  std::ifstream fin(INPUT_FILE);
  if (!fin.is_open()) {
    std::perror(("Не удалось открыть " + INPUT_FILE).c_str());
    return false;
  }

  std::vector<std::vector<int>> temp;
  std::string line;
  while (std::getline(fin, line)) {
    std::istringstream check(line);
    int val;
    std::vector<int> row;
    while (check >> val) {
      if (val != -1 && val != 0 && val != 1) {
        std::cerr << "Недопустимое значение: " << val << "\n";
        return false;
      }
      row.push_back(val);
```

```
    }
    if (!row.empty())
      temp.emplace_back(std::move(row));
  }
  fin.close();

  if (temp.empty()) {
    std::cerr << "Пустой файл или некорректные строки\n";
    return false;
  }

  n_vertices = static_cast<int>(temp.size());
  n_edges = static_cast<int>(temp[0].size());
  if (n_vertices < 4 || n_edges < 4) {
    std::cerr << "Нужно >=4 вершин и >=4 дуг, а получено n=" << n_vertices
              << ", m=" << n_edges << "\n";
    return false;
  }
  for (int i = 0; i < n_vertices; ++i) {
    if (static_cast<int>(temp[i].size()) != n_edges) {
      std::cerr << "Строка " << (i + 1) << ": ожидалось " << n_edges
                << " чисел, а найдено " << temp[i].size() << "\n";
      return false;
    }
  }
  inc_mat = std::move(temp);
  return true;
}

bool incidence2adjacency(int n_vertices, int n_edges,
                         const std::vector<std::vector<int>> &inc_mat,
                         std::vector<std::vector<int>> &adj,
                         std::vector<std::pair<int, int>> &edges) {
  adj.assign(n_vertices, std::vector<int>(n_vertices, 0));
```

```
  edges.clear();

  for (int j = 0; j < n_edges; ++j) {
    int src = -1, dst = -1;
    for (int i = 0; i < n_vertices; ++i) {
      if (inc_mat[i][j] == -1) {
        if (src != -1) {
          std::cerr << "Столбец " << (j + 1) << ": более одного -1\n";
          return false;
        }
        src = i;
      } else if (inc_mat[i][j] == 1) {
        if (dst != -1) {
          std::cerr << "Столбец " << (j + 1) << ": более одного +1\n";
          return false;
        }
        dst = i;
      }
    }
    if (src < 0 || dst < 0) {
      std::cerr << "Столбец " << (j + 1) << ": нет -1 и +1\n";
      return false;
    }
    adj[src][dst] = 1;
    edges.emplace_back(src, dst);
  }
  return true;
}

std::vector<std::vector<int>>
compute_reachability(int n, const std::vector<std::vector<int>> &adj) {
  std::vector<std::vector<int>> reach(n, std::vector<int>(n, 0));
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
```

```
      reach[i][j] = adj[i][j];
    }
    reach[i][i] = 1;
  }
  for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
      if (!reach[i][k])
        continue;
      for (int j = 0; j < n; ++j) {
        if (reach[k][j])
          reach[i][j] = 1;
      }
    }
  }
  return reach;
}

bool is_one_way_connected(int n, const std::vector<std::vector<int>> &reach) {
  for (int i = 0; i < n; ++i) {
    for (int j = i + 1; j < n; ++j) {
      if (!reach[i][j] && !reach[j][i])
        return false;
    }
  }
  return true;
}

void print_matrix(const std::string &title, int n,
                  const std::vector<std::vector<int>> &mat) {
  std::cout << title << " (" << n << "×" << n << "):\n";
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
      std::cout << mat[i][j] << " ";
    }
```

```
    std::cout << "\n";
  }
  std::cout << "\n";
}

int main(int argc, char *argv[]) {
  // bool visualize = false;
  // for (int i = 1; i < argc; ++i) {
  //   if (std::strcmp(argv[i], "-viz") == 0 ||
  //       std::strcmp(argv[i], "--visualize") == 0) {
  //     visualize = true;
  //   }
  // }

  int n_vertices = 0, n_edges = 0;
  std::vector<std::vector<int>> inc_mat;
  if (!read_incidence_matrix(n_vertices, n_edges, inc_mat)) {
    return EXIT_FAILURE;
  }

  std::vector<std::vector<int>> adj;
  std::vector<std::pair<int, int>> edges;
  if (!incidence2adjacency(n_vertices, n_edges, inc_mat, adj, edges)) {
    return EXIT_FAILURE;
  }

  print_matrix("Матрица смежности", n_vertices, adj);

  auto reach = compute_reachability(n_vertices, adj);
  print_matrix("Матрица достижимости", n_vertices, reach);

  bool one_way = is_one_way_connected(n_vertices, reach);
  std::cout << (one_way ? "Граф является односторонне связным.\n"
                        : "Граф НЕ является односторонне связным.\n");
```

```
  // if (visualize) {
  //   const std::string dot_file = "graph.dot";
  //   if (generate_dot(n_vertices, edges, dot_file)) {
  //     if (!dot_to_png(dot_file)) {
  //       std::cout << "Сгенерирован файл \"" << dot_file << "\".\n"
  //                 << "Для PNG: dot -Tpng -o graph.png " << dot_file << "\n";
  //     }
  //   } else {
  //     std::cerr << "Не удалось создать DOT-файл.\n";
  //   }
  // }
  return EXIT_SUCCESS;
}
```

```cpp
// src/graphviz.hpp
#ifndef GRAPHVIZ_HPP
#define GRAPHVIZ_HPP

#include <string>
#include <utility>
#include <vector>

// Генерация DOT-файла по списку дуг edges
bool generate_dot(int n, const std::vector<std::pair<int, int>> &edges,
                  const std::string &dot_name);

bool dot_to_png(const std::string &dot_name);

#endif // GRAPHVIZ_HPP

// src/graphviz.cpp
#include "graphviz.hpp"
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <sstream>

bool generate_dot(int n, const std::vector<std::pair<int, int>> &edges,
                  const std::string &dot_name) {
  std::ofstream fout(dot_name);
  if (!fout.is_open()) {
    std::perror(("Не удалось открыть " + dot_name).c_str());
    return false;
  }
  fout << "digraph G {\n"
       << "  rankdir=LR;\n"
       << "  node [shape=circle, fontsize=12];\n";
  for (size_t k = 0; k < edges.size(); ++k) {
```

```
    int src = edges[k].first;
    int dst = edges[k].second;
    fout << "  \"" << (src + 1) << "\" -> \"" << (dst + 1) << "\" [label=\"e"
         << (k + 1) << "\"];\n";
  }
  fout << "}\n";
  return true;
}

bool dot_to_png(const std::string &dot_name) {
  if (dot_name.size() < 4 || dot_name.substr(dot_name.size() - 4) != ".dot") {
    std::cerr << "viz: имя файла должно оканчиваться на .dot\n";
    return false;
  }

  if (std::system("dot -V >/dev/null 2>&1") != 0) {
    std::cerr << "viz: команда 'dot' не найдена в PATH\n";
    return false;
  }

  std::string base = dot_name.substr(0, dot_name.size() - 4);
  std::string png = base + ".png";
  std::ostringstream cmd;
  cmd << "dot -Tpng -o \"" << png << "\" \"" << dot_name << "\"";
  int ret = std::system(cmd.str().c_str());
  if (ret != 0) {
    std::cerr << "viz: ошибка dot (код " << ret << ")\n";
    return false;
  }
  std::cout << "Сохранено: " << dot_name << " → " << png << "\n";
  return true;
}
```