Министерство образования и науки РФ

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

Дискретная математика

Лабораторная работа № 1

Тема: «Определение компонент связности»

Выполнил: студентка группы ИВТ-23-1б

Хорошилова К. П.

Проверил: ст. преподаватель

Рустамханова Г. И.

г. Пермь – 2025**Цель работы**

Изучить алгоритмы поиска компонент связности в графе, освоение методов работы с матрицами смежности.

**Задачи работы**

1. Разработать алгоритм построения матрицы достижимости на основе матрицы смежности, используя степенные итерации матриц.
2. Реализовать алгоритм определения компонент слабой связности, используя обход в глубину.
3. Исследовать влияние преобразования ориентированного графа в неориентированный на структуру связности.
4. Провести тестирование алгоритма на различных входных данных.
5. Написать отчёт.

# Код программы

#include <iostream>

#include <fstream>

#include <vector>

#include <stack>

using namespace std;

// Преобразование ориентированного графа в неориентированный

void makeUndirectedGraph(vector<vector<int>>& graph, int n) {

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

if (graph[i][j] == 1) {

graph[j][i] = 1;

}

}

}

}

// Поиск в глубину для нахождения компоненты связности

void dfs(int v, vector<vector<int>>& graph, vector<bool>& visited, vector<int>& component) {

stack<int> stk;

stk.push(v);

visited[v] = true;

while (!stk.empty()) {

int node = stk.top();

stk.pop();

component.push\_back(node);

for (int i = 0; i < graph.size(); ++i) {

if (graph[node][i] == 1 && !visited[i]) {

visited[i] = true;

stk.push(i);

}

}

}

}

// Функция для вычисления матрицы достижимости

void computeReachabilityMatrix(vector<vector<int>>& graph, vector<vector<int>>& reachability, int n) {

reachability = graph;

for (int k = 0; k < n; ++k) {

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

if (reachability[i][k] && reachability[k][j]) {

reachability[i][j] = 1;

}

}

}

}

}

void readMatrixFromFile(const string& filename, vector<vector<int>>& matrix, int n) {

ifstream file(filename);

if (!file.is\_open()) {

cerr << "Ошибка при открытии файла!" << endl;

exit(1);

}

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

file >> matrix[i][j];

}

}

file.close();

}

void printMatrix(const vector<vector<int>>& matrix, int n, const string& message) {

cout << message << endl;

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

cout << matrix[i][j] << " ";

}

cout << endl;

}

}

int main() {

setlocale(LC\_ALL, "Russian");

int n = 10;

vector<vector<int>> graph(n, vector<int>(n));

string filename = "graph.txt";

readMatrixFromFile(filename, graph, n);

printMatrix(graph, n, "Исходная матрица смежности:");

makeUndirectedGraph(graph, n);

vector<bool> visited(n, false);

vector<vector<int>> components;

for (int i = 0; i < n; ++i) {

if (!visited[i]) {

vector<int> component;

dfs(i, graph, visited, component);

components.push\_back(component);

}

}

cout << "\nКоличество компонент связности: " << components.size() << "\n";

for (int i = 0; i < components.size(); ++i) {

cout << "Компонента " << i + 1 << ": ";

for (int v : components[i]) {

cout << v + 1 << " ";

}

cout << endl;

}

vector<vector<int>> reachability(n, vector<int>(n, 0));

computeReachabilityMatrix(graph, reachability, n);

printMatrix(reachability, n, "\nМатрица достижимости:");

return 0;

}

**Анализ кода**

Входные данные представляют собой матрицу смежности размером 10×10, где 1 означает наличие ребра между вершинами, а 0 – его отсутствие. Если граф ориентированный, то его предварительно преобразуют в неориентированный путем симметричного дополнения матрицы смежности.

Алгоритм нахождения матрицы достижимости основан на возведении матрицы смежности в степени и суммировании полученных результатов до стабилизации. Если две последовательные итерации дают одинаковую матрицу, значит, все возможные пути найдены. В результате получается матрица достижимости, в которой элемент (i, j) равен 1, если из вершины i можно попасть в вершину j.

Для определения компонент связности используется обход в глубину (DFS). Алгоритм стартует с непосещенной вершины, помечает все достижимые вершины как принадлежащие одной компоненте и повторяет процесс, пока все вершины не будут распределены по компонентам.

Функция makeUndirectedGraph преобразует ориентированный граф в неориентированный, дополняя матрицу смежности симметричными значениями. Если существует ребро из вершины i в j, то добавляется обратное ребро из j в i.

Функция dfs выполняет поиск в глубину, используя стек для итеративного обхода графа. Она помечает посещённые вершины и добавляет их в список текущей компоненты связности.

Функция computeReachabilityMatrix реализует алгоритм Флойда-Уоршелла для вычисления матрицы достижимости. Она проходит через все возможные промежуточные вершины и обновляет информацию о достижимости между парами вершин.

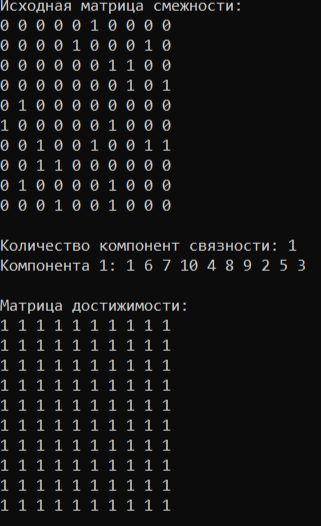
Функция readMatrixFromFile считывает матрицу смежности из файла, заполняя переданную ей матрицу.

Функция printMatrix отвечает за вывод матрицы в консоль. Она проходит по всем элементам матрицы и выводит их в виде таблицы.

На выходе программа формирует матрицу достижимости, количество компонент связности и список вершин, входящих в каждую компоненту.

Ссылка на Github: <https://github.com/ksinph/disc_math>

**Тесты**



# Заключение

# В ходе лабораторной работы я определила количество компонент связности графа и матрицу достижимости, реализовав на c++.