ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 8

«Подсчет числа связей между элементами, middle»

Выполнил работу

Дубинина Диана

Академическая группа C3100

Принято

Ментор Вершинин Владислав

Санкт-Петербург

2024

1. **Введение**

В данной лабораторной работе рассматривается задача нахождения количества связей для каждого элемента графа при заданной матрице смежности. Граф может быть пустым, разреженным или полным. Цель: реализовать работающий алгоритм с учетом всех необходимых ограничений. Задачи: написать алгоритм и провести его тестирование для различных случаев.

1. **Теоретическая подготовка**

Для выполнения лабораторной работы по теме "Работа с графами" необходимо изучить основные методы представления графов и алгоритмы работы с ними. В данной ЛР граф представлен матрицей смежности, и задача заключается в подсчёте количества связей для каждой вершины. Матрица смежности эффективно отображает рёбра графа, где элемент матрицы указывает на наличие или отсутствие связи между вершинами. Реализация использует динамические массивы и векторы, позволяя эффективно хранить графы. Алгоритм проверяется через несколько тестов, включая пустой граф и графы с различным числом рёбер. Оценка эффективности алгоритма проводится по времени выполнения и использованию памяти. Знание этих методов важно для решения задач с графами и анализа их сложности.

Используются типы данных - целые числа(int); структуры, такие как вложенные циклы for, матрица смежности, составленная из vector-ов, функции; для проведения тестов используется метод библиотеки cassert, ограничением является изначальное количество элементов в графе до 1е5.

1. **Реализация**

В процессе работы использовалась стандартная библиотека C++: iostream - для вывода сообщений на экран; vector - для хранения элементов, так как вектор позволяет динамически изменять размер и эффективно работать с данными; cassert - для реализации проверок с помощью assert.

Матрица смежности это основной способ представления графа в этой программе. Она задаётся в виде двумерного вектора (vector<vector<int>>), где каждая строка и столбец соответствует вершине графа. Элементы матрицы могут принимать значения 1 или 0, что указывает на наличие или отсутствие рёбер между вершинами. Прописывается это в главной функции countConnections:

void countConnections(const vector<vector<int>>& adjacencyMatrix, vector<int>& connections) {

int n = adjacencyMatrix.size();

connections.resize(n, 0);

if (n == 0){

cout << "пустой граф" << endl;

} // O(1)

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

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

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

connections[i]++;

}

}

cout << "элемент " << i << " содержит " << connections[i] << " связей" << endl;

}

}

Целью этой функции является определить количество связей для каждой вершины графа и сохранить их в векторе connections. Одновременно происходит вывод количества связей для каждой вершины в консоль.

Вектор connections используется для хранения результата работы функции countConnections. Он представляет собой одномерный массив, где каждая ячейка соответствует вершине графа. connections[i] хранит количество рёбер (или связей) для вершины i. Например, если у вершины 1 есть две связи, то connections[1] = 2.

Шаги реализации:

1. Инициализация: Получаем количество вершин графа int n = adjacencyMatrix.size(); Это определяет, сколько строк и столбцов содержит матрица. Инициализируем вектор connections нулями: connections.resize(n, 0); - Теперь connections имеет n элементов, каждый из которых равен 0.
2. Проверяем, пустой ли граф, используя значение количества элементов, если оно равно нулю, выводится сообщение:

if (n == 0) {

cout << "пустой граф" << endl;

}

1. Далее подсчет связей между элементами с помощью вложенных циклов. Для каждой вершины i перебираем все вершины j. Если adjacencyMatrix[i][j] == 1, увеличиваем счётчик связей для вершины i. Это означает, что вершина i соединена с вершиной j.
2. После подсчёта для вершины i выводим результат:

cout << "элемент " << i << " содержит " << connections[i] << " связей" << endl;

1. Тестирование кода в функции runTests. Реализовано с помощью метода assert, проводится 6 тестов на разные случаи. Пример теста:

// Тест 3: Граф из двух элементов с одной связью

{ cout << "тест 3" << endl;

vector<vector<int>> adjacencyMatrix = {

{0, 1},

{1, 0}

};

vector<int> connections;

countConnections(adjacencyMatrix, connections);

assert((connections.size() == 2 && connections[0] == 1 && connections[1] == 1));

}

Вывод программы для пройденных тестов:

тест 1

пустой граф

тест 2

элемент 0 содержит 0 связей

тест 3

элемент 0 содержит 1 связей

элемент 1 содержит 1 связей

тест 4

элемент 0 содержит 2 связей

элемент 1 содержит 2 связей

элемент 2 содержит 2 связей

тест 5

элемент 0 содержит 1 связей

элемент 1 содержит 2 связей

элемент 2 содержит 2 связей

элемент 3 содержит 1 связей

тест 6

элемент 0 содержит 1 связей

элемент 1 содержит 1 связей

элемент 2 содержит 0 связей

All tests passed!

1. **Заключение**

В ходе работы была реализована функция подсчёта количества связей каждой вершины графа на основе матрицы смежности. Реализация была протестирована на различных типах графов: пустом, полном, с изолированными вершинами и разреженном. Тесты подтвердили корректность алгоритма.

Анализ показал, что алгоритм имеет квадратичную сложность O(N^2), что связано с необходимостью перебора всех элементов матрицы. Он эффективно работает для графов небольшого размера, но для больших графов его производительность снижается.

Алгоритм подходит для анализа небольших графов. Для работы с большими графами стоит рассмотреть другие структуры данных, такие как список смежности, и более оптимальные методы.

ПРИЛОЖЕНИЕ А

Листинг кода, реализованного в лабораторной работе

#include <iostream>

#include <vector>

#include <cassert> // Подключение заголовочного файла для assert

using namespace std;

void countConnections(const vector<vector<int>>& adjacencyMatrix, vector<int>& connections) {

int n = adjacencyMatrix.size(); // Количество элементов

connections.resize(n, 0); // Обнуление массива для хранения связей

if (n == 0){

cout << "пустой граф" << endl;

}

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

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

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

connections[i]++;

}

}

cout << "элемент " << i << " содержит " << connections[i] << " связей" << endl;

}

}

void runTests() {

// Тест 1: Пустой граф

{ cout << "тест 1" << endl;

vector<vector<int>> adjacencyMatrix = {};

vector<int> connections;

countConnections(adjacencyMatrix, connections);

assert(connections == vector<int>{}); // Проверяем, что connections пуст

}

// Тест 2: Граф из одного элемента без связей

{ cout << "тест 2" << endl;

vector<vector<int>> adjacencyMatrix = {{0}};

vector<int> connections;

countConnections(adjacencyMatrix, connections);

assert(connections.size() == 1 && connections[0] == 0);

}

// Тест 3: Граф из двух элементов с одной связью

{ cout << "тест 3" << endl;

vector<vector<int>> adjacencyMatrix = {

{0, 1},

{1, 0}

};

vector<int> connections;

countConnections(adjacencyMatrix, connections);

assert((connections.size() == 2 && connections[0] == 1 && connections[1] == 1));

}

// Тест 4: Полный граф на трёх вершинах

{ cout << "тест 4" << endl;

vector<vector<int>> adjacencyMatrix = {

{0, 1, 1},

{1, 0, 1},

{1, 1, 0}

};

vector<int> connections;

countConnections(adjacencyMatrix, connections);

assert((connections.size() == 3 && connections[0] == 2 && connections[1] == 2 && connections[2] == 2));

}

// Тест 5: Разреженный граф на 4 вершинах

{ cout << "тест 5" << endl;

vector<vector<int>> adjacencyMatrix = {

{0, 1, 0, 0},

{1, 0, 1, 0},

{0, 1, 0, 1},

{0, 0, 1, 0}

};

vector<int> connections;

countConnections(adjacencyMatrix, connections);

assert((connections.size() == 4 && connections[0] == 1 && connections[1] == 2 &&

connections[2] == 2 && connections[3] == 1));

}

// Тест 6: Матрица смежности с изолированной вершиной

{ cout << "тест 6" << endl;

vector<vector<int>> adjacencyMatrix = {

{0, 1, 0},

{1, 0, 0},

{0, 0, 0}

};

vector<int> connections;

countConnections(adjacencyMatrix, connections);

assert((connections.size() == 3 && connections[0] == 1 && connections[1] == 1 && connections[2] == 0));

}

cout << "All tests passed!" << endl;

}

int main() {

runTests();

return 0;

}