Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Алтайский государственный технический университет
им. И.И. Ползунова»
Факультет (институт) Информационных технологий
Кафедра Прикладная математика

Отчет защищен с оценкой_____
А.И.Потупчик
(подпись преподавателя) (инициалы, фамилия)
“____”_____ 2024 г.

Отчет
по лабораторной работе №5

Разработка и анализ программ обхода графа

(название лабораторной работы)

по дисциплине Типы и структуры данных
(наименование дисциплины)

ЛР 09.03.04.13.000 ОТ
(обозначение документа)

Студент группы ПИ-21 А.А.Лихтинфельд
(инициалы, фамилия)
Преподаватель доцент, доцент А.И.Потупчик
(должность, ученое звание) (инициалы, фамилия)

Барнаул 2024

Задание:

1. Разработать и отладить программу на языке C++, реализующую работу с графом в соответствии с вариантом. Выполнить оценку временной и емкостной сложности программы.
2. Исходные данные поместить в файл input.dat
3. Результаты вывести на экран. Исходные данные и результаты вывести также в выходной файл output.dat

Вариант №3

Найти самый длинный простой путь в графе.

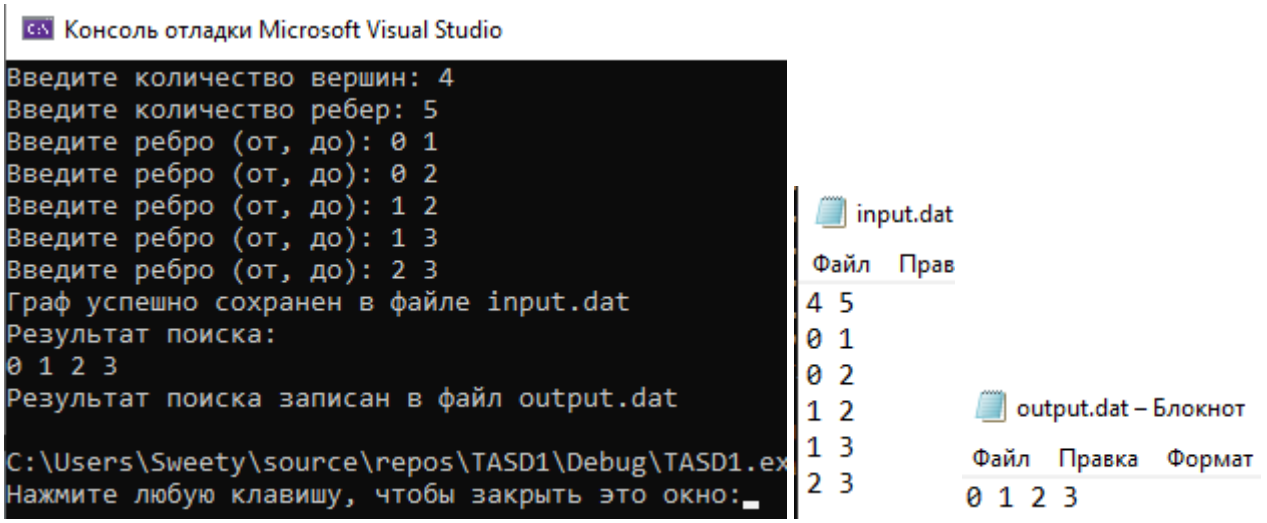
Текст программы:

```
//Алгоритм обхода в глубину состоит из трех основных шагов.
//Шаг 1. Всем вершинам графа присваивается значение не посещенная.
//Выбирается первая вершина и помечается как посещенная.
//Шаг 2. Для последней помеченной как посещенная вершины выбирается
//смежная вершина, являющаяся первой помеченной как не посещенная, и ей
//присваивается значение посещенная.Если таких вершин нет, то берется
//предыдущая помеченнаявершина.
//Шаг 3. Повторять шаг 2 до тех пор, пока все вершины не будут помечены
//как посещенные
//После завершения алгоритма, найден самый длинный простой путь в графе.

#include <iostream>
#include <vector>
#include <fstream>
#include <locale>

using namespace std;

vector<vector<int>> graph;// Граф представленный в виде списка смежности
vector<bool> visited;// Вектор для отслеживания посещенных вершин
vector<int> longestPath;// Вектор для хранения самого длинного пути
int longestPathLength = 0;// Переменная для хранения длины самого длинного пути

// Функция обхода в глубину для поиска самого длинного пути в графе
void dfs(int node, vector<int>& currentPath, int pathLength) {
    visited[node] = true;// Помечаем текущую вершину как посещенную
    currentPath.push_back(node); // Добавляем текущую вершину в текущий путь

    // Если длина текущего пути больше длины самого длинного пути, обновляем
значения
    if (pathLength > longestPathLength) {
        longestPathLength = pathLength;
        longestPath = currentPath;
    }
```

```
    // Проходим по всем смежным вершинам текущей вершины
    for (int i = 0; i < graph[node].size(); ++i) {
        int nextNode = graph[node][i];// Получаем следующую вершину
        // Если следующая вершина не посещена, запускаем обход в глубину из неё
        if (!visited[nextNode]) {
            dfs(nextNode, currentPath, pathLength + 1);// Рекурсивно вызываем dfs
        }
    }

    currentPath.pop_back();// Удаляем текущую вершину из текущего пути
    visited[node] = false;// Помечаем текущую вершину как непосещенную (для других
путей)
}

void input_file(int numVertices, int numEdges) {
    // Открытие выходного файла для сохранения графа
    ofstream outputFile("input.dat");
    outputFile << numVertices << " " << numEdges << endl;

    // Считывание ребер и сохранение их в файл
    for (int i = 0; i < numEdges; ++i) {
        int from, to;
        cout << "Введите ребро (от, до): ";
        cin >> from >> to;
        outputFile << from << " " << to << endl;
    }

    outputFile.close();

    cout << "Граф успешно сохранен в файле input.dat" << endl;
}

int main() {
    setlocale(LC_ALL, "Rus");
    int numVertices, numEdges;
    // Пример данных графа
    //vector<vector<int>> graph = { {1, 2}, {0, 2, 3}, {0, 1, 3}, {1, 2} };
    // Ввод количества вершин и ребер с консоли
    cout << "Введите количество вершин: ";
    cin >> numVertices;
    cout << "Введите количество ребер: ";
    cin >> numEdges;

    input_file(numVertices, numEdges);
    ifstream inputFile("input.dat");
    ofstream outputFile("output.dat");
```

```
    inputFile >> numVertices >> numEdges;// Считываем количество вершин и рёбер
    graph.resize(numVertices);// Изменяем размер списка смежности графа
    visited.assign(numVertices, false);// Инициализируем вектор посещенных вершин
    // Считываем рёбра и строим граф
    for (int i = 0; i < numEdges; ++i) {
        int from, to;
        inputFile >> from >> to;// Считываем ребро
        graph[from].push_back(to);// Добавляем ребро в список смежности
        graph[to].push_back(from); // Для неориентированного графа добавляем ребро в
обратную сторону
    }
    vector<int> currentPath;// Вектор для хранения текущего пути
    // Запускаем обход в глубину из каждой вершины графа
    for (int i = 0; i < numVertices; ++i) {
        dfs(i, currentPath, 1);
    }
    cout << "Результат поиска: " << endl;
    // Записываем самый длинный путь в выходной файл
    for (int node : longestPath) {
        outputFile << node << " ";
        cout << node << " ";
    }
    outputFile << endl;
    cout << endl;
    outputFile.close();
    inputFile.close();
    cout << "Результат поиска записан в файл output.dat" << endl;
    return 0;
}
```

Пример работы программы:



Оценка временной сложности программы: $O(V+E)$, где V - количество вершин, E - количество рёбер

Оценка ёмкостной сложности программы: $O(V+E)$, где V - количество вершин, E - количество рёбер