Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**Пояснительная Записка**

К курсовому проектированию

По курсу «Логика и основы алгоритмизации

В инженерных задачах»

На тему «Рекурсивный алгоритм обхода графа в глубину»

Выполнил:

студент группы 21ВВ3

Димитренко Н.С.

Приняли:

д.т.н. профессор Митрохин М.А.

к.т.н. доцент Юрова О.В.

Пенза 2022

**Содержание**

[**Реферат** 2](#_heading=h.gjdgxs)

[**Введение** 3](#_heading=h.30j0zll)

[**1.Постановка задачи** 4](#_heading=h.1fob9te)

[**2.Теоретическая часть задания** 5](#_heading=h.3znysh7)

[**3.Описание алгоритма программы**](#_heading=h.2et92p0) 6

[**4. Описание программы** 7](#_heading=h.tyjcwt)

[**5.Тестирование**](#_heading=h.3dy6vkm) 11

[**6.Ручной расчет задачи** 14](#_heading=h.1t3h5sf)

**Заключение** 15

[**Список литературы** 16](#_heading=h.4d34og8)

[**Приложение A. Листинг программы** 17](#_heading=h.2s8eyo1)

**Реферат**

Отчет 21 стр, 8 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, ОБХОД В ГЛУБИНУ.

Цель исследования – разработка программы, способной обходить граф в глубину.

В работе рассмотрена рекурсивная реализация алгоритма обхода графа в глубину.

# 

# Введение

Обход графа – одна из наиболее распространенных операций с графами. Задачей обхода является прохождение всех вершин в графе. Обходы применяются для поиска информации, хранящейся в узлах графа, нахождения связей между вершинами или группами вершин и т.д.

Одним из способов обхода графов является поиск в глубину. Идея такого обхода состоит в том, чтобы начав обход из какой-либо вершины всегда переходить по первой встречающейся в процессе обхода связи в следующую вершину, пока существует такая возможность. Как только в процессе обхода исчерпаются возможности прохода, необходимо вернуться на один шаг назад и найти следующий вариант продвижения. Таким образом, итерационно выполняя описанные операции, будут пройдены все доступные для прохождения вершины. Чтобы не заходить повторно в уже пройденные вершины, необходимо их пометить как пройденные.

В качестве среды разработки мною была выбрана среда MicrosoftVisualStudio2019, язык программирования – С++.

Целью данной курсовой работы является разработка программы на языке С++, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм рекурсивного обхода графа в глубину.

**1.Постановка задачи**

Требуется разработать программу, которая осуществляет рекурсивный обход графа в глубину. Исходный граф в программе должен задаваться матрицей смежности. Изначально в программе пользователь должен выбрать способ указания количества вершин графа (случайный, чтение из файла, вручную), после чего может создать граф. После обработки этих данных на экран должна выводиться матрица смежности, пути по которым прошел алгоритм и вершины, которые он обошёл. Устройство ввода – клавиатура и мышь.

# 

# 2.Теоретическая часть задания

Граф G (рисунок 1) задается множеством вершин X1, X2, ..., Xn. и множеством ребер, соединяющих между собой определенные вершины. Ребра из множества А ориентированы, что показывается стрелкой, которая указывает достижимость данной вершины, граф с такими ребрами называется неориентированным графом.

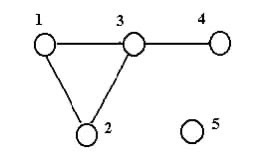


Рисунок 1 – Граф

При представлении графа матрицей смежности информация о ребрах графа хранится в квадратной матрице, где присутствие пути из одной вершины в другую обозначается единицей, иначе нулем.

# 3.Описание алгоритма программы

Цель алгоритма рекурсивного обхода графа в глубину в матрице смежности состоит в том, чтобы пометить каждую вершину как “Пройденная”, избегая при этом циклов. Ниже представлен псевдокод:

**Вход**: G – матрица смежности графа.

**Выход**: номера вершин в порядке их прохождения на экране.

**Алгоритм ПОГ**

1.1. для всех i положим NUM[i] = False пометим как "не посещенную";

1.2. **ПОКА** существует "новая" вершина v

1.3. **ВЫПОЛНЯТЬ** DFS (v).

**Алгоритм** DFS(v):

2.1. пометить v как "посещенную" NUM[v] = True;

2.2. вывести на экран v;

2.3. **ДЛЯ** i = 1 **ДО** size\_G **ВЫПОЛНЯТЬ**

2.4. **ЕСЛИ** G(v,i) = = 1**И** NUM[i] = = False

2.5. **ТО**

2.6. {

2.7. DFS(i);

2.8. }

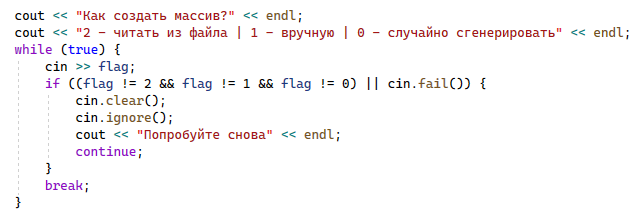
# 

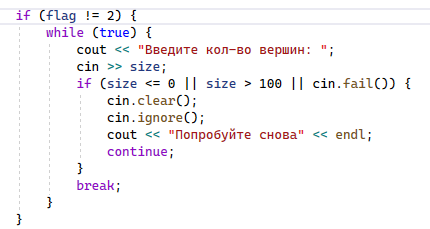
# 4.Описание программы

Для написания данной программы использован язык программирования С++. Язык программирования С++ - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней. Проект был создан в виде консольного приложения Win32 (Visual C++).

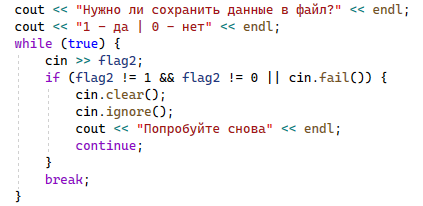
Для реализации программы понадобится двумерный динамический массив, куда в дальнейшем будут записывать значения матрицы смежности, переменные, хранящие количество вершин графа и текущую вершину.

Сначала мы выбираем способ создания массива

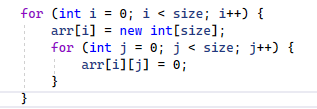


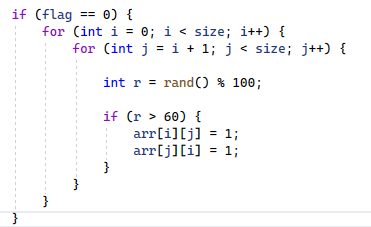
Далее выбираем необходимое количество вершин в графе.

Затем выбираем, сохранять ли данные в файл.

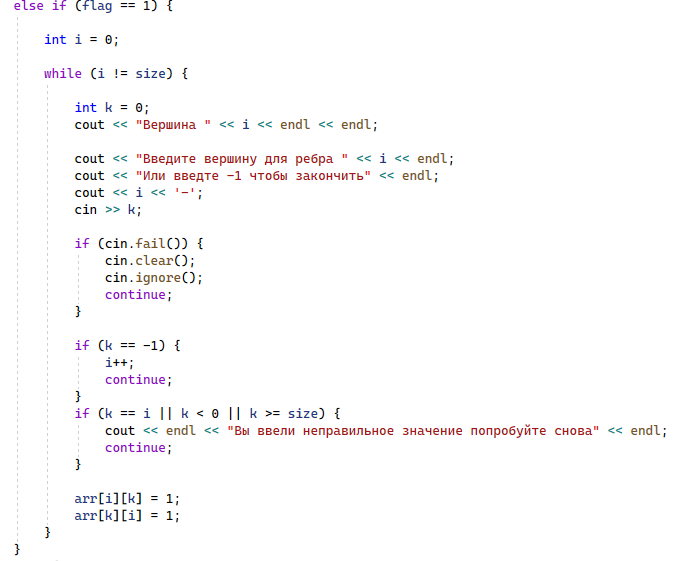


При выборе случайного заполнения создается массив для матрицы и заполняется нулями.

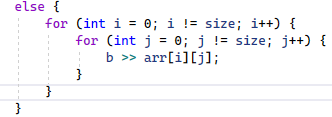


Далее, массив заполняется единицами случайным образом.

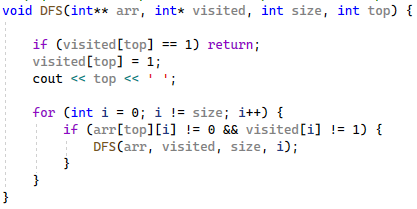
При выборе заполнения в ручную, массив заполняется введенными значениями.



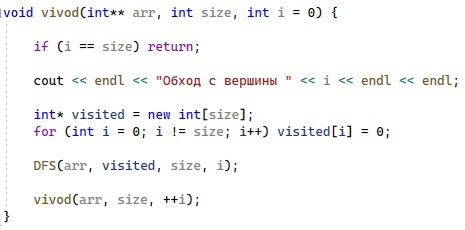
При выборе считывания из файла, массив заполняется данными из него.



Далее выполняется основная функция алгоритма по рекурсивному обходу графа в глубину.



После этого выполняется функция вывода.



# 

# 5.Тестирование

Среда разработки MicrosoftVisualStudio 2019 предоставляет все средства, необходимые при разработке и отладке программы. Тестирование проводилось в процессе разработки, после завершения написания программы. В ходе тестирования проблем не было выявлено.

Таблица 1 - Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск | Запуск программы | Верно |
| Проверка на неправильный выбор варианта создания массива | Программа должна оповестить пользователя  об ошибке, предложить снова варианты | Верно |
| Проверка на неправильный выбор варианта указания количества вершин. | Программа должна оповестить пользователя  об ошибке, предложить снова варианты | Верно |
| Проверка на не правильный выбор сохранения файла. | Программа должна оповестить пользователя  об ошибке, предложить снова варианты | Верно |
| Проверка ручного заполнения и его результаты | Программа должна оповестить пользователя  об ошибке, предложить снова варианты | Верно |
| Проверка считывания из файла. | Программа должна оповестить пользователя  об ошибке | Верно |

Тест 1. Проверка на неправильный выбор варианта создания массива.

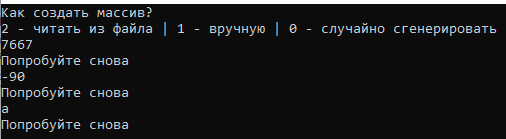


Рисунок 2 – Варианты создания массива.

Тест 2. Проверка на неправильный выбор варианта указания количества вершин.

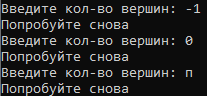


Рисунок 3 – Количество вершин

Тест 3. Проверка на не правильный выбор сохранения в файл.

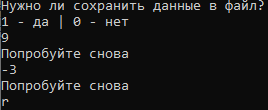


Рисунок 4 – Способ сохранения.

Тест 4. Проверка ручного заполнения и его результаты

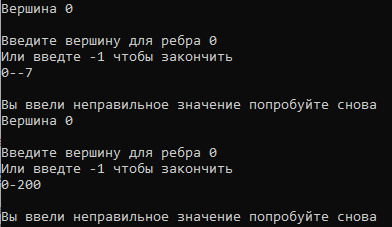


Рисунок 5 – Ручной ввод.

Тест 5. Проверка считывания из файла.

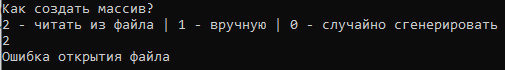


Рисунок 6 – Считывание из файла.

# 6.Ручной расчёт задачи

Проведем проверку программы посредством ручных вычислений на примере рисунка 7



Рисунок 7 – Граф

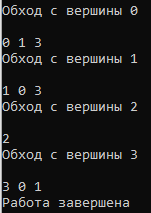


Рисунок 8 – Результат

Проходим из вершины 0 в 1, отмечаем их, так как они … , возвращаемся в 0, затем идем во 2, где видим, что они не соприкасаются, далее идем с вершины 0 в вершину 3, отмечаем, как посещенную. Обходим с вершины 1 - отмечаем саму вершину, идём в 0, имеется общее ребро, отмечаем, идем к вершине 2, они не связаны, возвращаемся, посещаем 3 вершину, отмечаем. Начинаем обход с вершины 2, не находим общие ребра с другими вершинами, записываем только её, она является изолированной. Обходим с вершины 3 - имеется общее ребро с 0, записываем обе, с 3 идем в 1, записываем, потому что они связаны, после этого посещаем вершину 2, видим, что она не имеет общего ребра, заканчиваем алгоритм. Таким образом, сделав ручной подсчет на данном примере, мы удостоверились в правильности данного алгоритма.

**Заключение**

Таким образом, в процессе создания данного проекта разработана программа, реализующая рекурсивный алгоритм обхода графа в глубину MicrosoftVisualStudio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежности, основанных на теории графов. Приобретены навыки по осуществлению алгоритма рекурсивного обхода графа в глубину. Углублены знания языка программирования C++.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

# Список литературы

1. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978

2. Уилсон Р. Введение в теорию графов. Пер. с анг. 1977. 208 с.

3. Ф. А. Новиков ДИСКРЕТНАЯ МАТЕМАТИКА ДЛЯ ПРОГРАММИСТОВ 3-е издание

4. Герберт Шилдт «Полный справочник по C++» - Вильямс, 2006

5. Роберт Седжвик: Алгоритмы на C++. Анализ структуры данных. Сортировка. Поиск. Алгоритмы на графах.

# Приложение A.

# Листинг программы.

#include <ctime>

#include <stack>

#include <iostream>

#include <fstream>

#include "windows.h"

using namespace std;

void DFS(int\*\* arr, int\* visited, int size, int top) {

if (visited[top] == 1) return;

visited[top] = 1;

cout << top << ' ';

for (int i = 0; i != size; i++) {

if (arr[top][i] != 0 && visited[i] != 1) {

DFS(arr, visited, size, i);

}

}

}

void vivod(int\*\* arr, int size, int i = 0) {

if (i == size) return;

cout << endl << "Обход с вершины " << i << endl << endl;

int\* visited = new int[size];

for (int i = 0; i != size; i++) visited[i] = 0;

DFS(arr, visited, size, i);

vivod(arr, size, ++i);

}

int main()

{

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

int\*\* arr;

int size;

int flag;

int flag2;

ifstream b;

cout << "Как создать массив?" << endl;

cout << "2 - читать из файла | 1 - вручную | 0 - случайно сгенерировать" << endl;

while (true) {

cin >> flag;

if ((flag != 2 && flag != 1 && flag != 0) || cin.fail()) {

cin.clear();

cin.ignore();

cout << "Попробуйте снова" << endl;

continue;

}

break;

}

if (flag != 2) {

while (true) {

cout << "Введите кол-во вершин: ";

cin >> size;

if (size <= 0 || size > 100 || cin.fail()) {

cin.clear();

cin.ignore();

cout << "Попробуйте снова" << endl;

continue;

}

break;

}

}

else {

b.open("ddd.txt");

if (b.fail()) {

cout << "Ошибка открытия файла" << endl;

return 0;

}

b >> size;

}

arr = new int\* [size];

cout << "Нужно ли сохранить данные в файл?" << endl;

cout << "1 - да | 0 - нет" << endl;

while (true) {

cin >> flag2;

if (flag2 != 1 && flag2 != 0 || cin.fail()) {

cin.clear();

cin.ignore();

cout << "Попробуйте снова" << endl;

continue;

}

break;

}

ofstream a;

if (flag2 == 1) {

a.open("ddd.txt");

a << size << endl;

}

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

arr[i] = new int[size];

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

arr[i][j] = 0;

}

}

if (flag == 0) {

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

for (int j = i + 1; j < size; j++) {

int r = rand() % 100;

if (r > 60) {

arr[i][j] = 1;

arr[j][i] = 1;

}

}

}

}

else if (flag == 1) {

int i = 0;

while (i != size) {

int k = 0;

cout << "Вершина " << i << endl << endl;

cout << "Введите вершину для ребра " << i << endl;

cout << "Или введте -1 чтобы закончить" << endl;

cout << i << '-';

cin >> k;

if (cin.fail()) {

cin.clear();

cin.ignore();

continue;

}

if (k == -1) {

i++;

continue;

}

if (k == i || k < 0 || k >= size) {

cout << endl << "Вы ввели неправильное значение попробуйте снова" << endl;

continue;

}

arr[i][k] = 1;

arr[k][i] = 1;

}

}

else {

for (int i = 0; i != size; i++) {

for (int j = 0; j != size; j++) {

b >> arr[i][j];

}

}

}

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

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

if (flag2 == 1) a << arr[i][j] << ' ';

cout << arr[i][j] << ' ';

}

if (flag2 == 1) a << endl;

cout << endl;

}

vivod(arr, size, 0);

cout << endl << "Работа завершена" << endl;

if (flag2 == 1) a.close();

return 0;

}