Министерство образования Российской Федерации

Пензенский Государственный Университет

Кафедра “Вычислительная техника”

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К курсовому проекту

По курсу “Логика и основы алгоритмизации

в инженерных задачах”

на тему “Реализация рекурсивного алгоритма обхода графа в глубину”

Выполнил:

Урядов В.Д.

Принял:

Митрохин М. А.

Пенза 2020

Содержание

[**Реферат** 3](#_Toc59020615)

[**1.** **Введение** 4](#_Toc59020616)

[**2.** **Постановка задачи** 5](#_Toc59020617)

[**3.** **Теоретическая часть задания** 6](#_Toc59020618)

[**4.** **Описание алгоритма программы** 7](#_Toc59020619)

[**5.** **Описание программы** 8](#_Toc59020620)

[**6.** **Тестирование** 12](#_Toc59020621)

[**7.** **Ручной расчёт задачи** 15](#_Toc59020622)

[**Заключение** 16](#_Toc59020623)

[**Список литературы** 17](#_Toc59020624)

[**Приложение А. Листинг программы.** 18](#_Toc59020625)

# **Реферат**

Отчет 22 стр., 15 рисунков.

**Рекурсивный обход графа в глубину**.

Цель исследования – разработка программы, способной находить вершины в графе с помощью рекурсивного алгоритма обхода в глубину. Получить навыки в разработке консольного приложения с пользовательским интерфейсом. Сохранить результаты работы программы в файл.

# **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 – Си/C++.

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

# **Постановка задачи**

Требуется разработать программу, которая выводит исходную матрицу смежности и пройденные вершины. Сначала необходимо, чтобы пользователь задал ввод матрицы: автоматический (случайная генерация) или ручной (в самой программе или из файла). Исходный граф в программе должен задаваться матрицей смежности, причём при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. После обработки этих данных на экран должна выводиться матрица смежности графа с пройденными вершинами и сохранение результатов в файл. Устройство ввода – клавиатура и мышь.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | **Описание теста** | **Как достичь** | **Ожидаемый результат** | **Полученный результат** | | Запуск программы | Открыть exe - файл | Вывод сообщения о выборе:  сгенерировать матрицу или  вывести матрицу из памяти или ввести её самому. | Верно | | Выбор генерации матрицы | Ввести в программе номер нужной команды | Вывод сообщения о  количестве вершин в графе  и выбора его вида. | Верно | | Вывод матрицы | Открыть созданный программой файл в папке Debug | Сохранение результатов работы программы в файл | Верно | | Ввод неверной команды или элемента | Ввести в программе или файле неверные значения | Вывод сообщения с просьбой изменить введённое значение. | Верно |   Таблица 1 – Описание поведения программы при тестировании |  |  |  |

# **Теоретическая часть задания**

Поиск в глубину  — один из методов обхода [графа](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)). Стратегия поиска в глубину состоит в том, чтобы идти «вглубь» графа, насколько это возможно. Алгоритм поиска описывается рекурсивно: перебираем все исходящие из рассматриваемой вершины рёбра. Если ребро ведёт в вершину, которая не была рассмотрена ранее, то запускаем алгоритм от этой нерассмотренной вершины, а после возвращаемся и продолжаем перебирать рёбра. Возврат происходит в том случае, если в рассматриваемой вершине не осталось рёбер, которые ведут в нерассмотренную вершину. Если после завершения алгоритма не все вершины были рассмотрены, то необходимо запустить алгоритм от одной из нерассмотренных вершин

# **Описание алгоритма программы**

Реализация состоит из подготовительной части, в которой все вершины помечаются как не помеченные и осуществляется запуск процедуры обхода для вершин графа. И непосредственно процедуры обхода, которая помечает текущую (т.е. ту, в которой на текущей итерации находится алгоритм) вершину как посещенную. Затем выводит номер текущей вершины на экран и в цикле просматривает v-ю строку матрицы смежности графа G(v,i). Как только алгоритм встречает смежную с v не посещенную вершину, то для этой вершины вызывается процедура обхода.

Псевдокод рекурсивного алгоритма обхода в глубину:

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) = = 1И NUM[i] = = False

2.5. ТО

2.6. {

2.7. DFS(i);

2.8. }

# **Описание программы**

Данная программа является многомодульной, поскольку состоит из

нескольких функций: main, vivod для вывода матрицы, dfs для обхода вершин в глубину.

Работа программы начинается с запроса генерации матрицы. Если

пользователь выбрал сгенерировать матрицу смежности, то на экран выводится запрос на количество вершин в графе. Выбор осуществляется с помощью переменной vibor.

Рассмотрим часть программы с выбором случайной генерации матрицы:

Вводим количество вершин матрицы и вероятность появления компоненты связности

if (vibor == 1)

{

printf("Введите размер генерируемой матрицы: >> ");

scanf\_s("%d", &n);

printf("\nВведите вероятность от 0 до 100 %: ");

cin >> ver;

Добавим srand(time(NULL)), чтобы при каждом запуске генерировалась новая случайная матрица.

srand(time(NULL));

p1 = (int\*\*)malloc(n \* sizeof(int\*));

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

{

p1[i] = (int\*)malloc(n \* sizeof(int));

}

vis = (int\*)malloc(n \* sizeof(int));

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

{

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

{

if (i == j)

{

p1[i][j] = 0;

}

else

{

tmp = rand() % 100;

if (tmp < ver)

p1[i][j] = 1;

else

p1[i][j] = 0;

}

}

}

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

{

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

{

p1[j][i] = p1[i][j];

}

}

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

vis[i] = 0;

}

Далее мы выводим исходную матрицу на экран с помощью функции vivod.

vivod(n, p1, FILE);

Далее выполняется алгоритм dfs

dfs(v, p1, n, vis, FILE);

Далее снова вызываем функцию если остались не пройденные вершины

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

{

if (vis[i] == 0)

{

dfs(i, p1, n, vis, FILE);

printf("\n");

}

}

Функция dfs:

Функция принимает матрицу p1, количество вершин n, матрицу vis для проверки пройденности вершины и файл FILE для сохранения результата в файл.

void dfs(int v, int\*\* p1, int n, int\* vis, ofstream & FILE)

{

vis[v] = true;

printf(">> %d ", v + 1);

FILE << v + 1 << " ";

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

{

if ((p1[v][i] == 1) and (vis[i] == 0))

{

dfs(i, p1, n, vis, FILE);

}

}

}

Функция vivod:

Функция принимает матрицу p, количество вершин n и файл FILE для сохранения результатов в файл.

void vivod(int n, int\*\* p, ofstream & FILE ) {

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

{

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

{

printf("%d ", p[i][j]);

FILE << p[i][j] << " ";

}

printf("\n");

FILE << endl;

}

}

Пользователь может также может ввести матрицу в самой программе или из файла. Полный код см. в приложении А.

Ниже можно увидеть оформление начального запроса и дальнейшие

действия с ним.

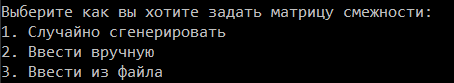


Рисунок 1 - Выбор ввода

Ввод количества вершин.



Рисунок 2 - Ввод количества вершин

Ввод вероятности компоненты связности.



Рисунок 3 - Ввод вероятности компоненты связности

Вывод полученной матрицы.

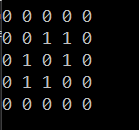


Рисунок 4 - Вывод полученной матрицы.

Вводим вершину, с которой начинаем обход. Получаем результат.

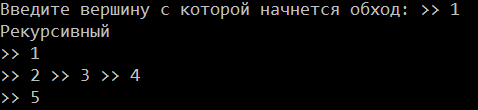


Рисунок 5 – Ввод вершины и получение результата

Имеем:

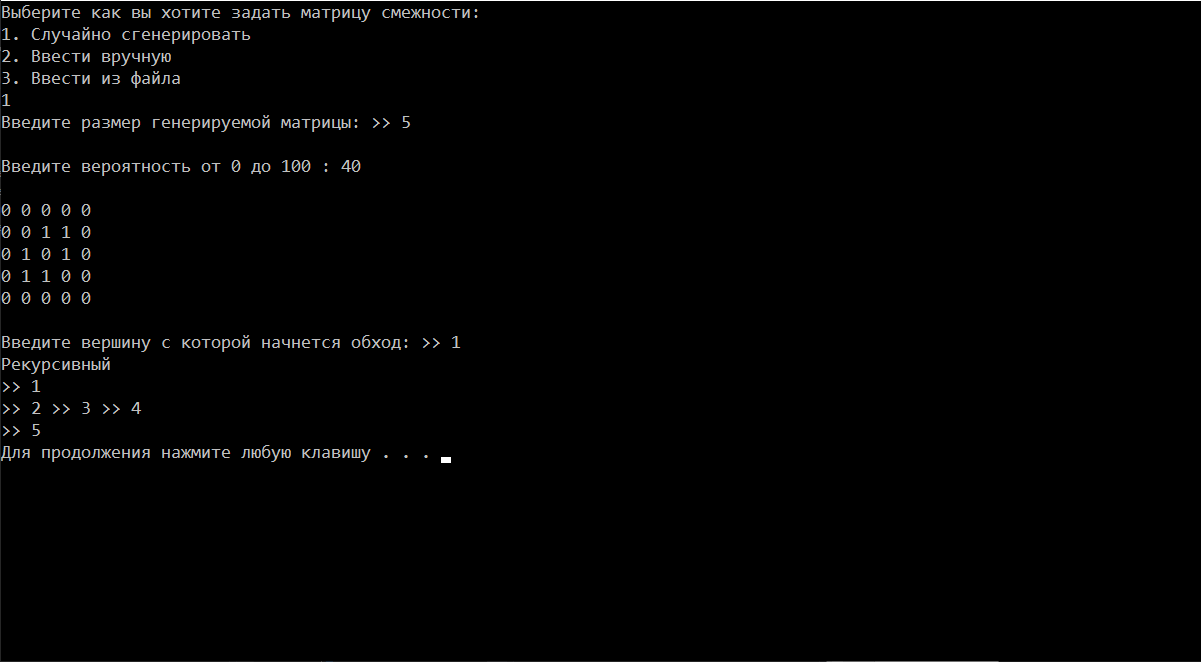


Рисунок 6 - конечный результат

# **Тестирование**

Среда разработки MicrosoftVisualStudio 2019 предоставляет все

средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки,

после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

Ниже продемонстрированы результаты тестирования программы при

всех возможных вариантах ввода.

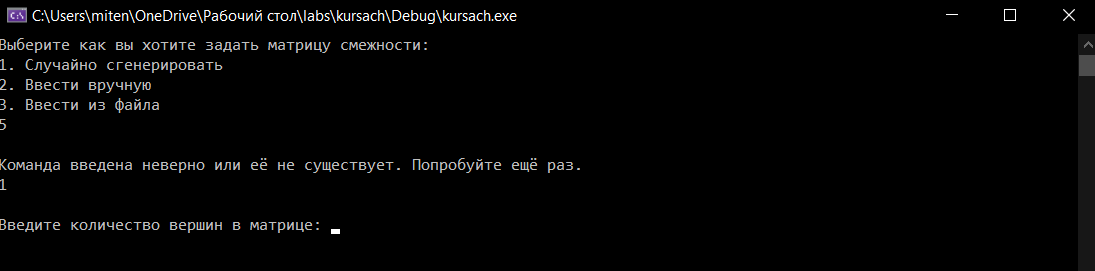


Рисунок 7 - неверный ввод

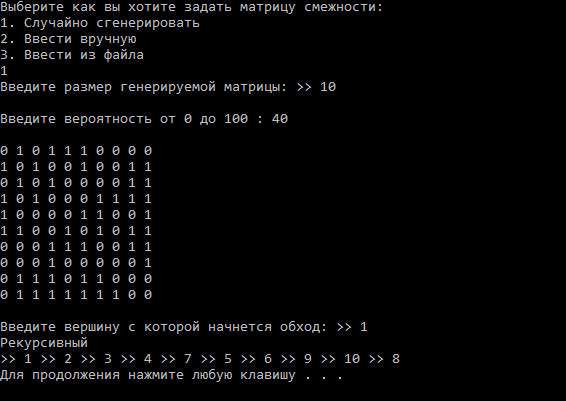


Рисунок 8 - верный ввод

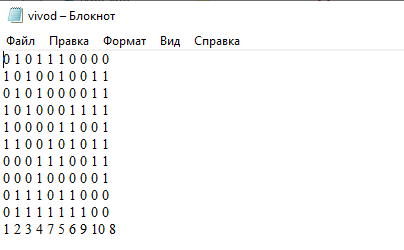


Рисунок 9 - сохранение результата в файл

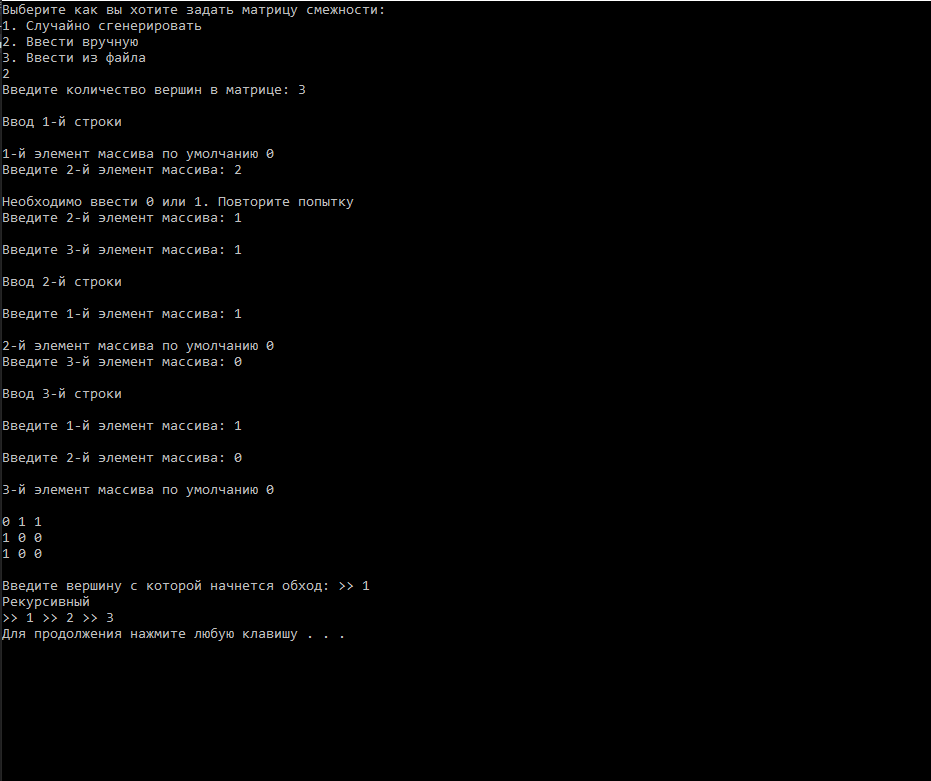


Рисунок 10 - ручной ввод с ошибкой и без

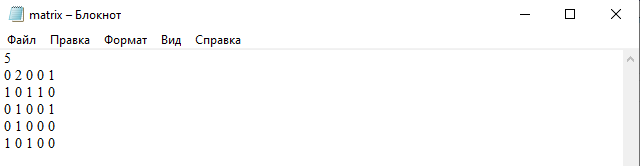


Рисунок 11 - файл с исходной матрицей с ошибкой

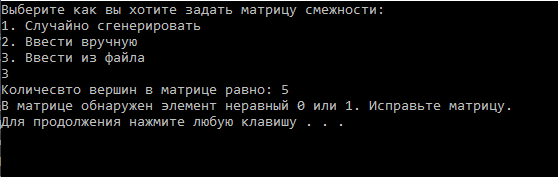


Рисунок 12 - вывод ошибки

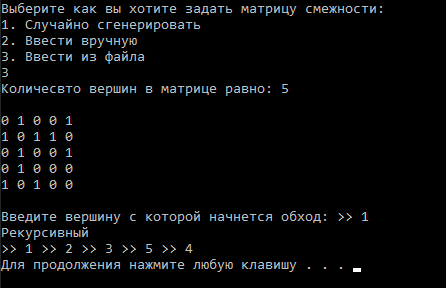


Рисунок 13 - вывод нормальной работы программы

В результате тестирования было выявлено, что программа успешно

проверяет данные на соответствие необходимым требованиям.

# **Ручной расчёт задачи**

Ручной расчет задачи:

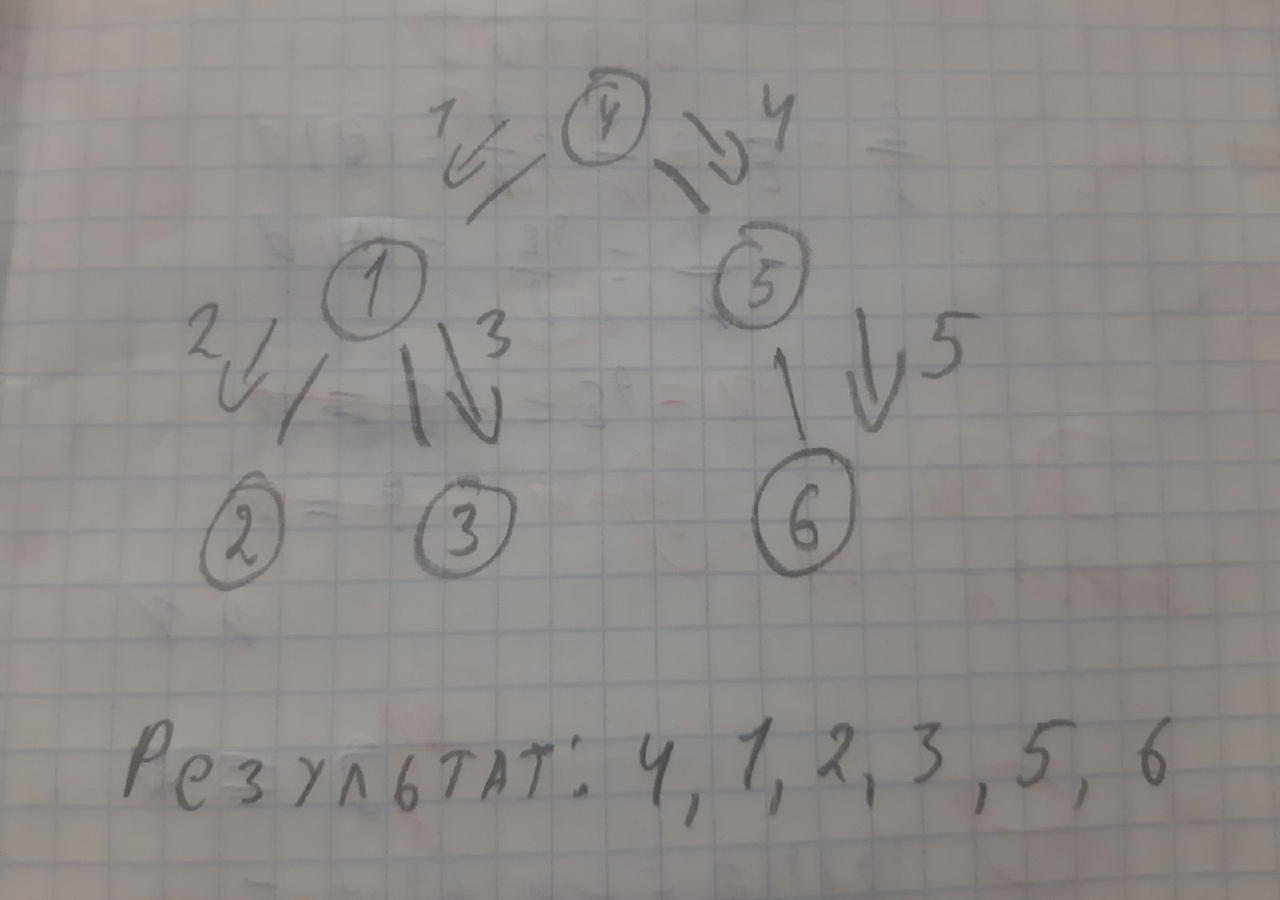


Рисунок 14 - проверка

Результат работы программы:

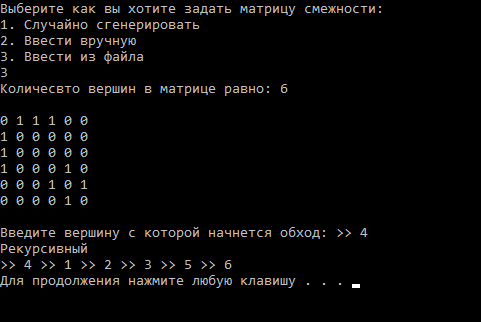


Рисунок 15 - проверка

# **Заключение**

Таким образом, в процессе создания данного проекта разработана

программа, реализующая алгоритм рекурсивного обхода в глубину в MicrosoftVisualStudio 2019.

При выполнении данной курсовой работы были получены навыки

разработки программ и освоены приемы создания матриц смежностей,

основанных на теории графов. Углублены знания языка программирования Cи/C++.

Недостатком разработанной программы является примитивный

пользовательский интерфейс. Программа работает в консольном

режиме, не добавляющем к сложности языка сложность программного

оконного интерфейса.

Программа имеет небольшой, но достаточный для использования

функционал возможностей.

# **Список литературы**

1. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: Построение и

анализ - М.: МЦНМО, 2001. - 960 с.

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

1978

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

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

5. Харви Дейтел, Пол Дейтел. Как программировать на C/C++. 2009 г.

# **Приложение А. Листинг программы.**

#define CRT\_SECURE\_NO\_WARNINGS

#include <fstream>

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <iostream>

#include <stack>

#include <time.h>

using namespace std;

void dfs(int v, int\*\* p1, int n, int\* vis, ofstream & FILE)

{

vis[v] = true;

printf(">> %d ", v + 1);

FILE << v + 1 << " ";

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

{

if ((p1[v][i] == 1) and (vis[i] == 0))

{

dfs(i, p1, n, vis, FILE);

}

}

}

void vivod(int n, int\*\* p, ofstream & FILE ) {

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

{

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

{

printf("%d ", p[i][j]);

FILE << p[i][j] << " ";

}

printf("\n");

FILE << endl;

}

}

int main() {

setlocale(LC\_ALL, "Rus");

ofstream FILE("vivod.txt");

double ver, tmp;

int i = 0, j = 0, n, v, vibor;

int\*\* p1, \* vis;

bool proverka = false;

cout << "Выберите как вы хотите задать матрицу смежности: "

<< endl << "1. Случайно сгенерировать"

<< endl << "2. Ввести вручную"

<< endl << "3. Ввести из файла"

<< endl;

while (!proverka)

{

cin >> vibor;

if (vibor == 1 || vibor == 2 || vibor == 3) {

proverka = true;

}

else

{

printf("Команда введена неверно или её не существует. Попробуйте ещё раз.\n");

}

}

if (vibor == 1)

{

printf("Введите размер генерируемой матрицы: >> ");

scanf\_s("%d", &n);

printf("\nВведите вероятность от 0 до 100 %: ");

cin >> ver;

srand(time(NULL));

p1 = (int\*\*)malloc(n \* sizeof(int\*)); //выделение памяти под всё, заполнение и отзеркаливание

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

{

p1[i] = (int\*)malloc(n \* sizeof(int));

}

vis = (int\*)malloc(n \* sizeof(int));

for (i = 0; i < n; i++) //заполнение массива p1

{

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

{

if (i == j)

{

p1[i][j] = 0;

}

else

{

tmp = rand() % 100;

if (tmp < ver)

p1[i][j] = 1;

else

p1[i][j] = 0;

}

}

}

for (i = 0; i < n; i++) //отзеркаливание матрицы

{

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

{

p1[j][i] = p1[i][j];

}

}

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

vis[i] = 0;

}

printf("\n");

vivod(n, p1, FILE);

printf("\n");

printf("Введите вершину с которой начнется обход: >> ");

scanf\_s("%d", &v);

v -= 1;

printf("Рекурсивный\n");

dfs(v, p1, n, vis, FILE);

printf("\n");

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

{

if (vis[i] == 0)

{

dfs(i, p1, n, vis, FILE);

printf("\n");

}

}

}

if (vibor == 2)

{

cout << "Введите количество вершин в матрице: ";

scanf\_s("%d", &n);

cout << endl;

int\*\* matrix;

int numb;

p1 = (int\*\*)malloc(n \* sizeof(int\*)); //выделение памяти под всё, заполнение и отзеркаливание

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

{

p1[i] = (int\*)malloc(n \* sizeof(int));

}

vis = (int\*)malloc(n \* sizeof(int));

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

cout << "Ввод "

<< i + 1

<< "-й строки"

<< endl

<< endl;

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

if (i == j)

{

cout << j + 1

<< "-й элемент массива по умолчанию 0"

<< endl;

p1[i][j] = 0;

}

else

{

proverka = false;

while (!proverka) {

cout << "Введите "

<< j + 1

<< "-й элемент массива: ";

cin >> numb;

cout << endl;

if (numb == 0 || numb == 1) {

p1[i][j] = numb;

proverka = true;

}

else

{

cout << "Необходимо ввести 0 или 1. Повторите попытку"

<< endl;

proverka = false;

}

}

}

}

}

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

vis[i] = 0;

}

printf("\n");

vivod(n, p1, FILE);

printf("\n");

printf("Введите вершину с которой начнется обход: >> ");

scanf\_s("%d", &v);

v -= 1;

printf("Рекурсивный\n");

dfs(v, p1, n, vis, FILE);

printf("\n");

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

{

if (vis[i] == 0)

{

dfs(i, p1, n, vis, FILE);

printf("\n");

}

}

}

if (vibor == 3)

{

ifstream file("matrix.txt");

if (!file.is\_open()) { // если файл не открыт

cout << "Файл не может быть открыт!\n";

system("pause");

return 1;

}

file >> n;

cout << "Количесвто вершин в матрице равно: "

<< n

<< endl;

p1 = (int\*\*)malloc(n \* sizeof(int\*)); //выделение памяти под всё, заполнение и отзеркаливание

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

{

p1[i] = (int\*)malloc(n \* sizeof(int));

}

vis = (int\*)malloc(n \* sizeof(int));

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

vis[i] = 0;

}

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

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

file >> p1[i][j];

if (p1[i][j] != 0 && p1[i][j] != 1) {

cout << "В матрице обнаружен элемент неравный 0 или 1. Исправьте матрицу."

<< endl;

system("pause");

return 1;

}

if (i == j) {

if (p1[i][j] != 0) {

cout << "В матрице не все элементы главной диагонали нули. Исправьте матрицу."

<< endl;

system("pause");

return 1;

}

}

}

}

printf("\n");

vivod(n, p1, FILE);

printf("\n");

printf("Введите вершину с которой начнется обход: >> ");

scanf\_s("%d", &v);

v -= 1;

printf("Рекурсивный\n");

dfs(v, p1, n, vis, FILE);

printf("\n");

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

{

if (vis[i] == 0)

{

dfs(i, p1, n, vis, FILE);

printf("\n");

}

}

}

FILE.close();

system("pause");

}