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 алгоритма обхода в ширину”

Выполнил:

Махнаев Н.С.

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** 8](#_Toc59020619)

[**5.** **Описание программы** 14](#_Toc59020620)

[**6.** **Тестирование** 19](#_Toc59020621)

[**7.** **Ручной расчёт задачи** 24](#_Toc59020622)

[**Заключение** 25](#_Toc59020623)

[**Список литературы** 26](#_Toc59020624)

[**Приложение А. Листинг программы.** 27](#_Toc59020625)

# **Реферат**

Отчет 32 стр, 21 рисунок.

**Алгоритм обхода в ширину**.

Цель исследования – разработка программы, способная находить все вершины в графе. Получить навыки в разработке консольного приложения с пользовательским интерфейсом. Сохранить результаты работы программы в файл.

# **Введение**

Алгоритм обхода в ширину предназначен для нахождения всехвершин в графе. Алгоритм осуществляется по так называемой **«матрице смежности»** размера NxN, где N — количество вершин графа.

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 обхода в ширину.

# **Постановка задачи**

Требуется разработать программу, которая выводила исходную матрицу смежности и пройденные вершине. 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 вершинами и сохранение результатов в файл. Устройство ввода – клавиатура и мышь.

Таблица 1 – Описание поведения программы при тестировании

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

# **Теоретическая часть задания**

Обход графа в ширину – еще один распространенный способ обхода графов.

Основная идея такого обхода состоит в том, чтобы посещать вершины по уровням удаленности от исходной вершины. Удалённость в данном случае понимается как количество ребер, по которым необходимо прейти до достижения вершины. Например, если для графа на рисунке 1 начать обход из первой вершины, то вершины 3, 6 и 2 будут находиться на уровне удаленности в 1 ребро, а вершины 5 и 4 на уровне удаленности в 2 ребра.



Рисунок 1 – Граф

Тогда при обходе этого графа в ширину, мы сначала посетим вершины первого уровня удаленности (с номерами 2, 3 и 6), и только после того, как закончатся не посещенные вершины на этом уровне, мы перейдем к следующему. На втором уровне мы посетим все вершины, которые удалены от исходной на 2 ребра (вершины 4 и 5).

Так, алгоритм обхода в ширину продолжает осматривать уровень за уровнем, пока не пройдет все доступные вершины.

Чтобы не заходить повторно в уже пройденные вершины, они помечаются, как и в алгоритме обхода в глубину.

Для того, чтобы проход осуществлялся по уровням необходимо хранить информацию о требуемом порядке посещения вершин. Вершины, которые являются ближайшими соседями исходной вершины (из которой начат обход) должны быть посещены раньше, чем соседи соседей и т.д. Такой порядок позволяет задать структура данных «очередь». Просматривая строку матрицы смежности (или список смежности) для текущей вершины мы помещаем всех её ещё не посещенных соседей в очередь. На следующей итерации текущей вершиной становится та, которая стоит в очереди первой и уже её не посещенные соседи будут помещены в очередь. Но место в очереди они займут после тех вершин, которые были помещены туда на предыдущих итерациях.

# **Описание алгоритма программы**

Таким образом, можно предложить следующую реализацию алгоритма обхода в ширину.

Вход: G – матрица смежности графа.

Выход: номера вершин в порядке их прохождения на экране.

Алгоритм ПОШ

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

1.2. ПОКА существует "новая" вершина v

1.3. ВЫПОЛНЯТЬ BFS (v).

Алгоритм BFS(v):

2.1. Создать пустую очередь Q = {};

2.2. Поместить v в очередь Q.push(v);

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

2.4. ПОКА Q != ∅ очередь не пуста ВЫПОЛНЯТЬ

2.5. v = Q.front() установить текущую вершину;

2.6. Удалить первый элемент из очереди Q.pop();

2.7. вывести на экран v;

2.8. ДЛЯ i = 1 ДО size\_G ВЫПОЛНЯТЬ

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

2.10. ТО

2.11. Поместить i в очередь Q.push(i);

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

Реализация состоит из подготовительной части, в которой все вершины помечаются как не поcещенные (п.1.1) и осуществляется запуск процедуры обхода для вершин графа (п.1.2, 1.3).

В самой процедуре обхода сначала создается пустая очередь (п. 2.1), в которую помещается исходная вершина, из которой начат обход (п.2.2).

Далее итерационно, пока очередь не опустеет, из нее извлекается первый элемент, который становится текущей вершиной (п. 2.5, 2.6). Затем в цикле просматривается v-я строка матрицы смежности графа G(v,i). Как только алгоритм встречает смежную с v не посещенную вершину (п.2.9), эта вершина помещается в очередь (п.2.11) и помечается как посещенная (п.2.12). После просмотра строки матрицы смежности алгоритм делает следующую итерацию цикла 2.4 или заканчивает работу, если очередь пуста.

Так, если для графа на рисунке 1, мы начнем обход из первой вершины, то на шаге 2.2 она будет помещена в очередь и помечена как посещенная (NUM[1] = True). Условие цикла 2.4 будет выполнено, вершина 1 будет установлена в качестве текущей (п.2.5) и удалена из очереди (п.2.6). На экран будет выведена единица.

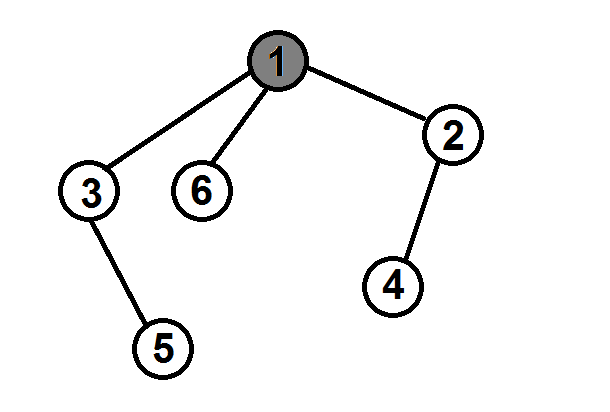
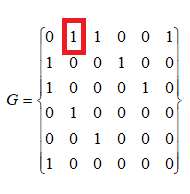
 

Рисунок 2 – Итерация 1

При просмотре 1-й строки матрицы смежности



будет найдена смежная вершина с индексом 2 (G(1,2) = = 1), которая не посещена (NUM[2] = = False) и она будет помещена в очередь и помечена (NUM[2] = = True). Просмотр строки продолжится, и в очередь будут также помещены и помечены как посещенные вершины с индексами 3 и 6.

К концу первой итерации очередь будет содержать Q = {2, 3, 6}.

Условие цикла while будет выполнено и на следующей итерации вершина 2 будет установлена как текущая и извлечена из очереди, на экран будет выведена двойка.

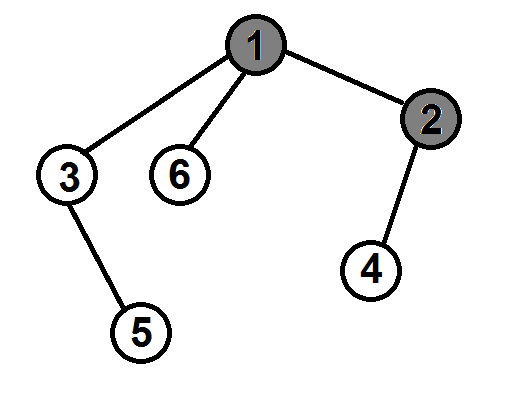
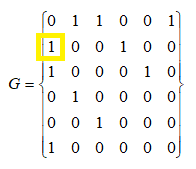
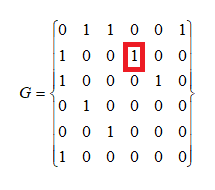


Рисунок 3 – Итерация 2

И алгоритм перейдет к просмотру второй строки матрицы смежности. Первая смежная с вершиной 2 - вершина с индексом 1(G(2,1) = =1),



которая к настоящему моменту уже посещена (NUM[1] = = True), она не будет помещена в очередь. Цикл 2.9 продолжит просмотр матрицы смежности.



Следующая найденная вершина, смежная со второй, будет иметь индекс 4 (G(2,4) = =1), она не посещена (NUM[4] = = False) и она будет помещена в очередь и помечена.

К концу второй итерации очередь будет содержать Q = {3, 6, 4}.

Условие цикла while будет выполнено и на следующей итерации вершина 3 будет установлена как текущая и извлечена из очереди, на экран будет выведена тройка.

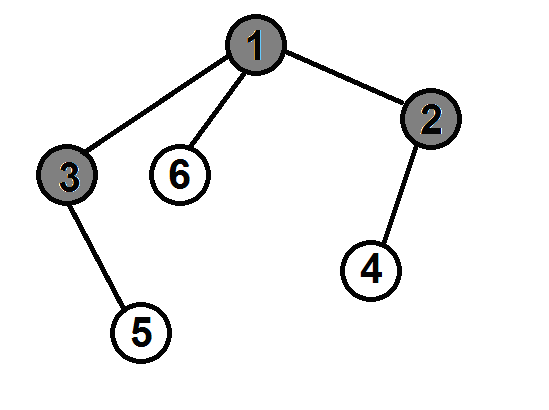
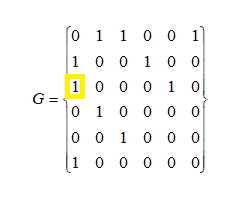
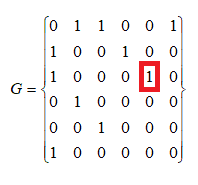


Рисунок 4 – Итерация 3

При просмотре 3-й строки матрицы будет найдена вершина 1, но она уже посещена (NUM[1] = True), поэтому в очередь помещена не будет.



Следующая найденная вершина, смежная с третьей – вершина с номером 5 (G(3,5) = =1), она не посещена (NUM[5] = = False) и будет помещена в очередь.



К концу третьей итерации очередь будет содержать Q = {6, 4, 5}.

Условие цикла while будет выполнено. На следующей итерации вершина 6 будет установлена как текущая и извлечена из очереди, на экран будет выведена цифра шесть.

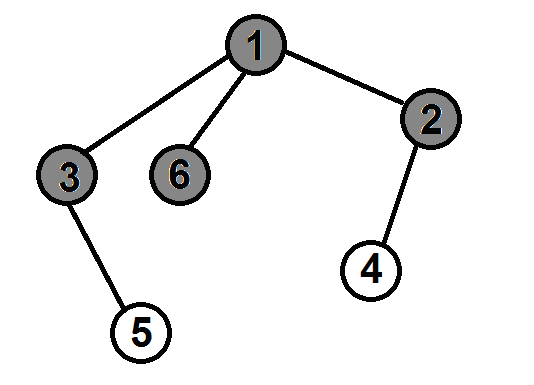


Рисунок 5 – Итерация 4

При просмотре 6-й строки матрицы будет найдена вершина 1, но она уже посещена (NUM[1] = True), поэтому в очередь помещена не будет.

К концу четвертой итерации очередь будет содержать Q = {4, 5}.

Далее из очереди будет извлечена вершина 4, установлена как текущая и выведена на экран.

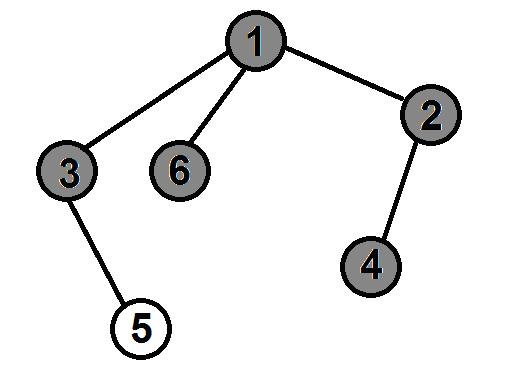


Рисунок 6 – Итерация 5

Так как все вершины кроме пятой уже посещены, то при просмотре 4-й строки в матрице смежности в очередь не будут добавлены вершины.

К концу пятой итерации очередь будет содержать только одну вершину Q = {5}.

На шестой итерации вершина с номером пять будет установлена как текущая и извлечена из очереди, на экран будет выведена цифра пять.

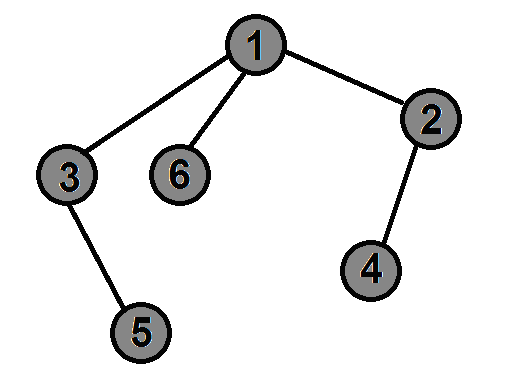


Рисунок 7 – Результат работы обхода

Просмотр 5-й строки матрицы смежности не добавит в очередь новых вершин, так как к этому моменту они уже все помечены как посещенные.

После этой итерации очередь окажется пуста Q = {}и алгоритм завершит свою работу.

В конце работы алгоритма все вершины будут посещены. А на экран будут выведены номера вершин в порядке их посещения алгоритмом.

# **Описание программы**

Данная программа является многомодульной, поскольку состоит из

нескольких функций: main, printMatrix для вывода матрицы, bfs для обхода всех путей.

Работа программы начинается с запроса генерации матрицы. Если

пользователь выбрал сгенерировать матрицу смежности, то на экран выводится запрос на количество вершин в графе, а затем выбор ориентированного графа или нет. Выбор осуществляется с помощью переменной numberEnter.

Рассмотрим часть программы с выбором случайной генерации матрицы:

if (numberEnter == 1) {

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

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

cout << endl;

cout << "Выберите вид графа: " << endl << "1. Ориентированный" << endl << "2. Неориентированный" << endl;

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

srand(time(NULL));

int\*\* matrix;

matrix = (int\*\*)malloc(sizeof(int) \* n); //выделяем память для

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

matrix[i] = (int\*)malloc(sizeof(int) \* n); // заполнения матрицы

}

С помощью переменной proverka и цикла while проверяем правильность введённого значения.

proverka = false;

while (!proverka)

{

cin >> numberVec;

if (numberVec == 1 || numberVec == 2) {

proverka = true;

}

else

{

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

}

}

При вводе верной команды будет выполнена генерация матрицы

// Случайная генерация матрицы

if (numberEnter == 1 || numberEnter == 2) {

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

{

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

{

Генерируем в матрице главную диагональ состоящую из нулей

if (i == j)

{

matrix[i][j] = 0;

}

Далее генерируем элементы в матрице. С помощью переменной ver мы будем создавать связь между вершинами с вероятностью 50%.

else

{

ver = rand() % 2;

// вершины связаны

if (ver == 1) {

matrix[i][j] = rand() % 100;

}

// вершины не связаны

else {

matrix[i][j] = INF;

}

}

}

}

Если был выбран неориентированный граф выполняется следующее условие:

if (numberVec == 2) {

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

{

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

{

matrix[j][i] = matrix[i][j];

}

}

}

}

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

printMatrix(matrix, n, fout);

Затем мы вводим вершину, с которой начнётся обход

proverka = false;

while (!proverka) {

cout << "Введите вершину для начала поиска в ширину: ";

cin >> v;

cout << endl;

if (v > n || v <= 0) {

cout << "Неверный ввод вершины. Повторите попытку" << endl;

proverka = false;

}

else

{

proverka = true;

}

}

v--;

Наконец, вызываем функцию с алгоритмом обхода в ширину

bfs(v, matrix, n, vis, fout);

Функция bfs

Функция принимает матрицу matrix и количество вершин n

void bfs(int v, int\*\* p, int n, int\* vis, ofstream& fout)

{

queue <int> q{};

q.push(v);

vis[v] = true;

while (!q.empty())

{

v = q.front();

q.pop();

cout << ">> " << v + 1 << " ";

fout << ">> " << v + 1 << " ";

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

{

if (p[v][i] == 1 and vis[i] == false)

{

q.push(i);

vis[i] = true;

}

}

}

}

Функция printMatrix

Функция принимает матрицу matrix, количество вершин n и файл fout для сохранения результатов в файл.

void printMatrix(int\*\* matrix, int n, ofstream& fout) {

cout << "Исходная матрица: " << endl;

fout << "Исходная матрица: " << endl;

fout << "Количество вершин: " << n << endl;

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

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

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

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

}

cout << endl;

fout << endl;

}

cout << endl;

fout << 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.

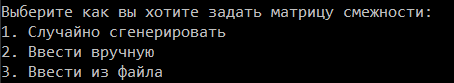


Рисунок 8- Выбор ввода

Ввод количества вершин.



Рисунок 9 - Ввод количества вершин

Ввод вида графа.

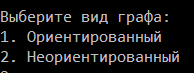


Рисунок 10 - Ввод вида графа

Ввод вершины.



Рисунок 11 - Ввод вершины

Имеем:

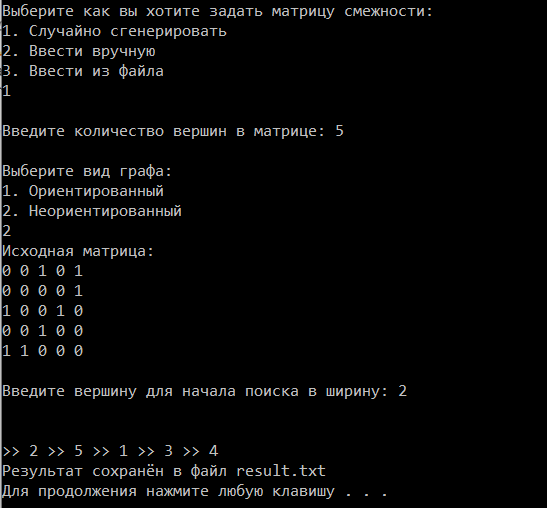


Рисунок 12 - конечный результат

# **Тестирование**

Среда разработки MicrosoftVisualStudio 2010 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.

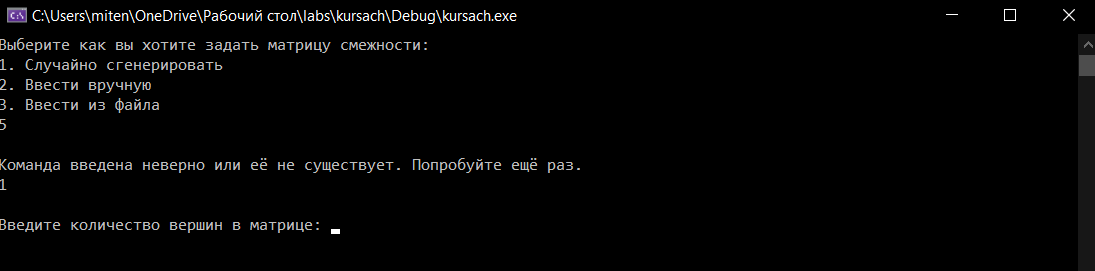


Рисунок 13 - неверный ввод

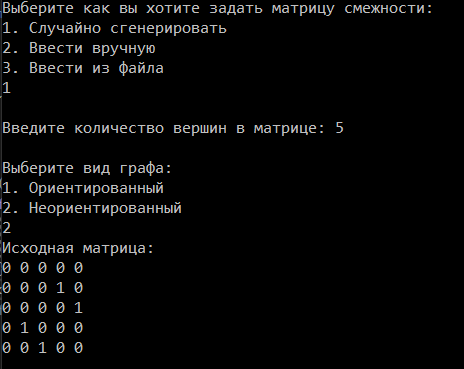


Рисунок 14 - автоматический ввод неориентированного графа

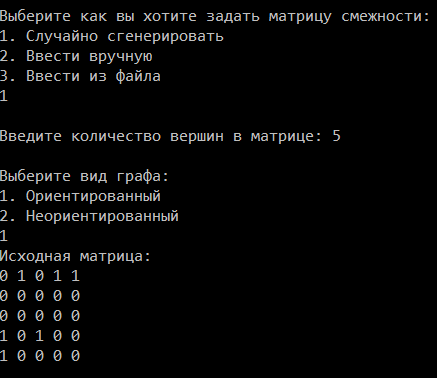


Рисунок 15 - автоматический ввод ориентированного графа

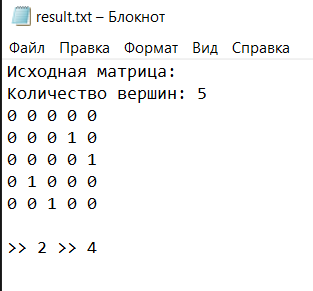


Рисунок 16 - сохранение результата в файл

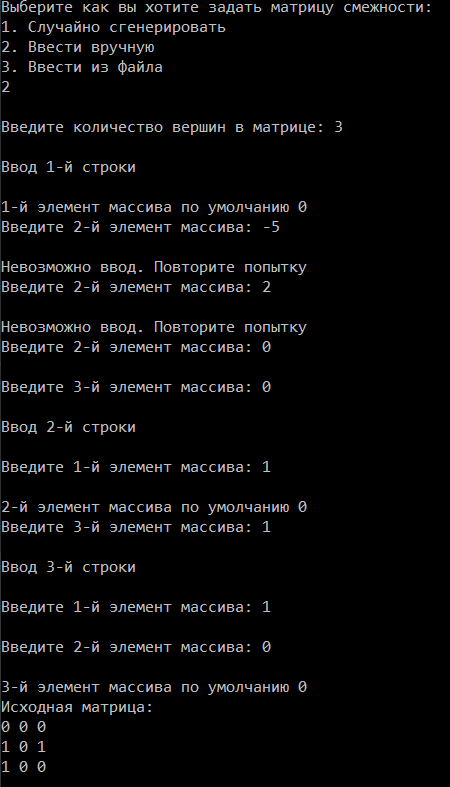


Рисунок 17 - ручной ввод с ошибкой и без

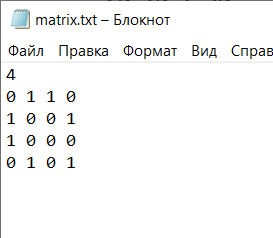


Рисунок 18 - файл с исходной матрицей с ошибкой

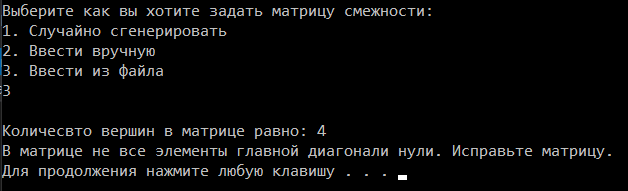


Рисунок 19 - вывод ошибки

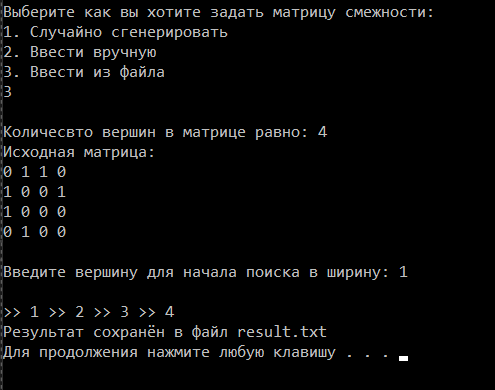


Рисунок 20 - вывод нормальной работы программы

В результате тестирования было выявлено, что программа успешно

проверяет данные на соответствие необходимым требованиям.

# **Ручной расчёт задачи**

Например, дан граф:

1

5

2

4

6

3

Обход: 1, 2, 4, 5, 3, 6

Результат работы программы:

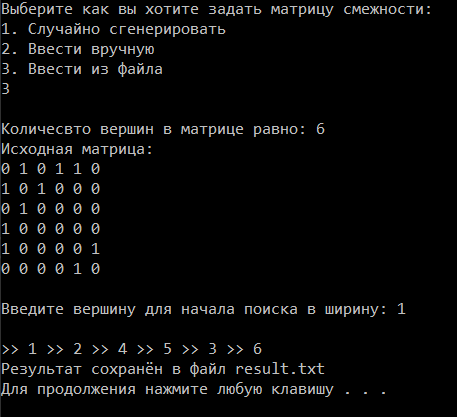


Рисунок 21 - проверка

# **Заключение**

Таким образом, в процессе создания данного проекта разработана

программа, реализующая алгоритм обхода в ширину для нахождения вершин в графе 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. Потому что 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.**

#include <iostream>

#include <algorithm>

#include <fstream>

#include <time.h>

#include <locale.h>

#include <queue>

using namespace std;

// Вывод матрицы

void printMatrix(int\*\* matrix, int n, ofstream& fout) {

cout << "Исходная матрица: " << endl;

fout << "Исходная матрица: " << endl;

fout << "Количество вершин: " << n << endl;

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

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

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

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

}

cout << endl;

fout << endl;

}

cout << endl;

fout << endl;

}

//Алгоритм обхода в ширину

void bfs(int v, int\*\* p, int n, int\* vis, ofstream& fout)

{

queue <int> q{};

q.push(v);

vis[v] = true;

while (!q.empty())

{

v = q.front();

q.pop();

cout << ">> " << v + 1 << " ";

fout << ">> " << v + 1 << " ";

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

{

if (p[v][i] == 1 and vis[i] == false)

{

q.push(i);

vis[i] = true;

}

}

}

}

int main(int argc, char\*\* argv) {

//matrix - матрица смежности

ofstream fout("result.txt");

setlocale(LC\_ALL, "Rus");

// n - количество вершин

// numberEnter - номер выбранного варианта ввода матрицы

// numberVec - номер ввода для вида направленности графа

// v - вершина, с которой начинается обход

int n, numberEnter, numberVec, v;

// proverka - переменная для проверки правильности варианта ввода

bool proverka = false;

//ver - переменная, отвечающая за вероятность генерации связанной вершины

int ver;

cout << "Выберите как вы хотите задать матрицу смежности: " << endl << "1. Случайно сгенерировать" << endl << "2. Ввести вручную" << endl << "3. Ввести из файла" << endl;

while (!proverka) {

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

cout << endl;

if (numberEnter == 1 || numberEnter == 2 || numberEnter == 3) proverka = true;

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

}

if (numberEnter == 1) {

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

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

cout << endl;

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

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

{

vis[i] = 0;

}

cout << "Выберите вид графа: " << endl << "1. Ориентированный" << endl << "2. Неориентированный" << endl;

srand(time(NULL));

int\*\* matrix;

matrix = (int\*\*)malloc(sizeof(int) \* n); //выделяем память для

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

matrix[i] = (int\*)malloc(sizeof(int) \* n); // заполнения матрицы

}

proverka = false;

while (!proverka)

{

cin >> numberVec;

if (numberVec == 1 || numberVec == 2) {

proverka = true;

}

else

{

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

}

}

// Случайная генерация матрицы

if (numberEnter == 1) {

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

{

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

{

if (i == j)

{

matrix[i][j] = 0;

}

else

{

ver = rand() % 2;

// вершины связаны

if (ver == 1) {

matrix[i][j] = rand() % 2;

}

// вершины не связаны

else {

matrix[i][j] = 0;

}

}

}

}

if (numberVec == 2) {

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

{

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

{

matrix[j][i] = matrix[i][j];

}

}

}

}

printMatrix(matrix, n, fout);

proverka = false;

while (!proverka) {

cout << "Введите вершину для начала поиска в ширину: ";

cin >> v;

cout << endl;

if (v > n || v <= 0) {

cout << "Неверный ввод вершины. Повторите попытку" << endl;

proverka = false;

}

else

{

proverka = true;

}

}

v--;

cout << endl;

bfs(v, matrix, n, vis, fout);

cout << endl << "Результат сохранён в файл result.txt" << endl;

}

// Ручной ввод

if (numberEnter == 2) {

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

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

cout << endl;

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

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

{

vis[i] = 0;

}

int\*\* matrix;

int numb;

matrix = (int\*\*)malloc(sizeof(int) \* n); //выделяем память для

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

matrix[i] = (int\*)malloc(sizeof(int) \* n); //заполнения матрицы

}

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;

matrix[i][j] = 0;

}

else

{

proverka = false;

while (!proverka) {

cout << "Введите " << j + 1 << "-й элемент массива: ";

cin >> numb;

cout << endl;

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

matrix[i][j] = numb;

proverka = true;

}

else

{

cout << "Невозможно ввод. Повторите попытку" << endl;

proverka = false;

}

}

}

}

}

printMatrix(matrix, n, fout);

proverka = false;

while (!proverka) {

cout << "Введите вершину для начала поиска в ширину: ";

cin >> v;

cout << endl;

if (v > n || v <= 0) {

cout << "Неверный ввод вершины. Повторите попытку" << endl;

proverka = false;

}

else

{

proverka = true;

}

}

v--;

bfs(v, matrix, n, vis, fout);

cout << endl

<< "Результат сохранён в файл result.txt" << endl;

}

// с Ввод с файла

if (numberEnter == 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;

int\*\* matrix = (int\*\*)malloc(sizeof(int) \* n); //выделяем память для

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

matrix[i] = (int\*)malloc(sizeof(int) \* n); //заполнения матрицы

}

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

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

{

vis[i] = 0;

}

//Считываем матрицу

//return 1 - аварийное завершение

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

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

file >> matrix[i][j];

if (matrix[i][j] < 0) {

cout << "В матрице обнаружет отрицательный элемент. Исправьте матрицу." << endl;

system("pause");

return 1;

}

if (i == j) {

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

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

system("pause");

return 1;

}

}

}

}

file.close(); // закрываем файл

printMatrix(matrix, n, fout);

proverka = false;

while (!proverka) {

cout << "Введите вершину для начала поиска в ширину: ";

cin >> v;

cout << endl;

if (v > n || v <= 0) {

cout << "Неверный ввод вершины. Повторите попытку" << endl;

proverka = false;

}

else

{

proverka = true;

}

}

v--;

bfs(v, matrix, n, vis, fout);

cout << endl << "Результат сохранён в файл result.txt" << endl;

}

system("pause");

return 0;

}