Министерство 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 «Реализация алгоритма обхода графа в ширину»

Выполнил:

студент группы 20ВВ3

Кузин Д.Д.

Принял:

Митрохин М.А

Пенза 2022

**ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**Факультет Вычислительной техники

Кафедра “Вычислительная техника”

“УТВЕРЖДАЮ”

Зав. кафедрой ВТ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_20\_\_

**ЗАДАНИЕ**

**на курсовое проектирование по курсу**

**\_\_\_\_\_\_\_**”Логика\_и\_основы\_алгоритмизации\_в\_инженерных\_задачах”**\_\_\_\_\_\_**Студенту\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Группа\_\_\_\_\_\_\_\_\_\_  
Тема\_проекта\_\_\_\_\_Реализация\_алгоритма\_обхода\_графа\_в\_ширину\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Исходные данные (технические требования) на проектирование

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Объем работы по курсу**

1. Расчетная часть

Ручной\_расчёт\_работы\_алгоритма\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Графическая часть

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Экспериментальная часть

Тестирование\_программы;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Результат\_работы\_программы\_на\_\_тестовых\_данных;\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Срок выполнения проекта по разделам

1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_4\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_5\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_6\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_8\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания “\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты проекта “\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Задание получил**      “\_\_\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

**Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Содержание

Реферат 5

Введение6

Постановка задачи7

Теоретическая часть задания 8

Описание алгоритма программы 10

Описание программы 12

Тестирование 14

Ручной расчёт задачи17

Заключение 18

Список литературы 19

Приложение А. Листинг программы20

**Реферат**

Отчет 23 стр, 14 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, ПОИСК В ШИРИНУ

Цель исследования – разработка программы, для нахождения связанных между собой вершин в неориентированном графе, используя алгоритм поиска в ширину.

В данной работе были рассмотрены правила поиска в ширину, на основе которых происходит обход графа и поиск связанных в нём вершин.

**Введение**

Алгоритм поиска в ширину (breadth-first search) заключается в том, чтобы посетить сначала все соседние вершины и только после этого опускать на уровень ниже. При каждом новом шаге посещаются вершины, расстояние от которых до начальной на единицу больше предыдущего.

Главное отличие поиска в ширину(BFS) от поиска в глубину(DFS) это использование очереди для хранения вершин, в тоже время как DFS использует принцип стека. Алгоритм поиска в ширину оптимален всего для нахождения наикратчайшего пути, за счёт того, что обрабатывает сразу все вершины графа. В тоже время наиболее оптимально пространство памяти использует DFS алгоритм, так как ему не обязательно хранить сразу все узлы, в отличие от BFS.

Среда для разработки выбранная мной – Microsoft Visual Studio, язык программирования – Си.

Основной целью данной курсовой работы является реализация алгоритма поиска в ширину для обхода в неориентированном графе.

**Постановка задачи**

Требуется разработать программу, которая выполнит перебор вершин графа, используя алгоритм поиска в ширину.

Граф должен задаваться матрицей смежности. Количество вершин в графе задаётся пользователем через ввод с клавиатуры или вручную написанной матрицей в текстовом документе, который считывает программа. Завершением работы программы считается текстовый документ с порядком вершин.

Устройство ввода – клавиатура и мышь.

**Теоретическая часть**

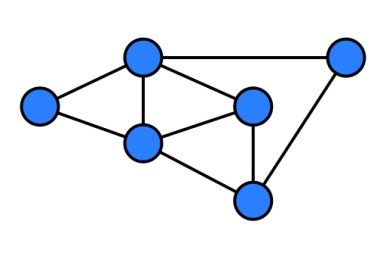
Граф G – фигура, состоящая из заданных точек (вершин), соединенных отрезками, которые называются ребрами (дугами) графа G. Используется для отображения связей между элементами предметной области или задания последовательности действий (рис.1).

Рисунок 1 – Пример графа

В математике графом называют пару (V,E) где V это множество вершин, а E множество пар, каждая их которых представляет собой свзяь(эти пары называются ребрами).

Обход графа – это определение всех вершин, достижимых от заданной вершины. Является одной из основных операций, которые выполняются с графами. Двумя основными алгоритмами обхода графа являются поиск в глубину и поиск в ширину. Расскажу более подробно о втором методе перебора вершин.

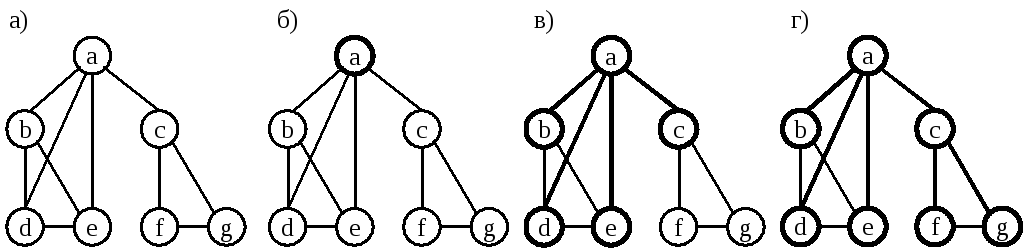
Поиск в ширину(рис.2) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *breadth-first search*, BFS) — один из методов обхода [графа](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)). Пусть задан граф {\displaystyle G=(V,E)} и выделена исходная вершина v1{\displaystyle s}. Алгоритм поиска в ширину систематически обходит все ребра {\displaystyle G} для «открытия» всех вершин, достижимых из v1{\displaystyle s}, вычисляя при этом расстояние (минимальное количество рёбер) от {\displaystyle s} v1 до каждой достижимой из {\displaystyle s} вершины. Алгоритм работает как для [ориентированных](https://ru.wikipedia.org/wiki/%D0%9E%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B3%D1%80%D0%B0%D1%84), так и для неориентированных графов.

Рисунок 2 – Поиск в ширину

Пусть задан граф, и выбрана исходная вершина v1. Алгоритм начинает поиск всех смежных с v1 вершин и поочерёдно посещает каждую из них. Моделировать это можно следующим образом. Создадим очередь Q, в которую будут помещаться посещенные вершины, а также заведём нулевой массив A, в котором для каждой вершины будем отмечать, была ли она уже посещена. Изначально в очередь помещается только вершина v1, которая сразу помечается посещенной.

Затем алгоритм представляет собой такой цикл: пока очередь Q не пуста, взять вершину v, просмотреть все рёбра, исходящие из этой вершины, и если какие-то из смежных вершин ещё не посещенные, пометить их и поместить в конец очереди. В итоге, когда очередь опустеет, мы по одному разу обойдём все достижимые из v1 вершины и повторим цикл для каждой из них. После того, когда очередь будет пуста, алгоритм прекратит работу.

**Описание алгоритма программы**

Для реализации алгоритма понадобиться: массив num (int) для хранения информации о посещенных вершинах и пустая очередь Q <int> в которую будут заноситься смежные вершины. Мы имеем некий граф G, в котором каждая вершина, изначально помечена как не посещенная.

Пользователь выбирает исходную вершину v, из которой будет происходить обход, которую сразу заносят в очередь Q.push(v) и она отмечается как посещенная num[v] = 1. В очередь заносятся все смежные с v вершины, если очередь Q не является пустой, то вершина будет установлена в качестве текущей Q.front(v) и далее удалена из очереди Q.pop(v). После чего на экран выводиться номер вершины. Далее итерационно, пока очередь не опустеет, из нее извлекается первый элемент, который становится текущей вершиной и цикл повторяется.

Ниже представлен псевдокод функции BFS() и частично main().

**BFS()**

1. Q.push(v)

2. Помечаем вершину как посещенную num[v]=1

3. **ПОКА**  Q не пуста **ВЫПОЛНЯТЬ**

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

5. Q.pop();

6. Вывести на экран v;

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

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

9. Q.push(i);

10. num[i] = 1;

11. Конец цикла

**Описание программы**

Программа начинает свою работу с запроса генерации матрицы. Даются 2 варианта – автоматически сгенерировать матрицу или считывать матрицу из файла. Если пользователь выбрал первый вариант, то происходит запрос на количество вершин в графе. После чего на экран выводиться матрица смежности и пользователю предлагают выбрать стартовую вершину (рис.3).

printf("Введите тип ввода:\n1-Автоматическая генерация\n2-Считывать из файла\n");

int answer;

printf("Выбрать: ");

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

if (answer == 1) {

printf("Введите размер матрицы: ");

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

num = (int\*)malloc(size \* sizeof(int));

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

{

num[i] = 0;

}

printf("\n");

matrix = (int\*\*)malloc(size \* sizeof(int\*));

srand(time(NULL));

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

{

matrix[i] = (int\*)malloc(size \* sizeof(int));

}

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

{

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

{

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

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

if (matrix[i] == matrix[j])

matrix[i][j] = 0;

}

}

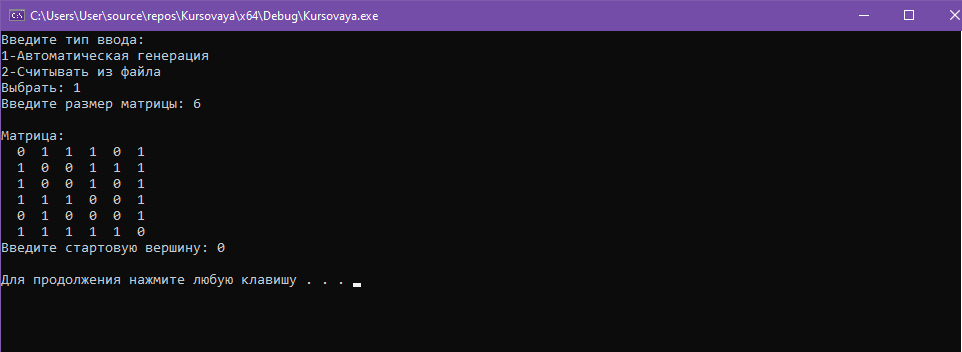
 break;

Рисунок 3 – Случайна генерация матрицы

Если пользователь выбрал второй вариант, то на экран выводиться матрица, которая заранее была записана в файл, и также предлагается выбрать стартовую вершину (рис.4).

else if (answer == 2) {

FILE\* input = fopen("test.txt", "r");

char a[10];

fgets(a,10,input);

for (int i = 0; i < 10; i++)

{

if (a[i] >= '0' && a[i] <= '9') {

size = size \* 10 + (a[i] - '0');

}

}

num = (int\*)malloc(size \* sizeof(int));

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

{

num[i] = 0;

}

printf("\n");

matrix = (int\*\*)malloc(size \* sizeof(int\*));

srand(time(NULL));

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

{

matrix[i] = (int\*)malloc(size \* sizeof(int));

}

char\* buffer = (char\*)malloc(sizeof(char) \* 2 \* size+1);

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

{

fgets(buffer, size \* 2+1, input);

for (int j = 0; j < size \* 2; j = j + 2)

{

matrix[i][j / 2] = (int)buffer[j] - '0';

}

}

fclose(input);

break;

}

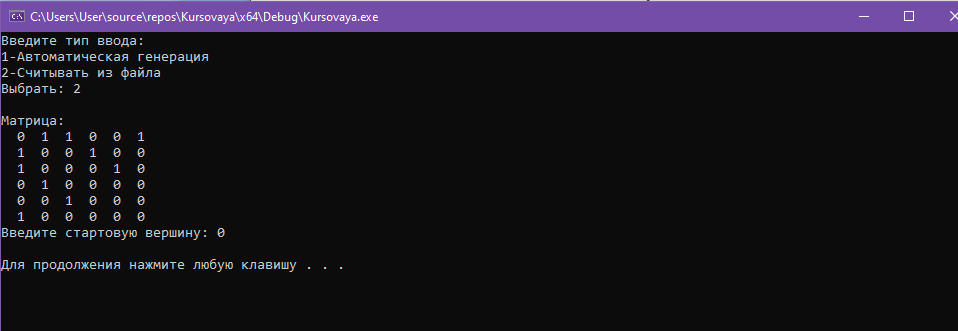


Рисунок 4 – Выбор матрицы из файла

**Тестирование**

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.

После разработки программы был проведен ряд тестов в ходе которых, были исправлены некоторые проблемы, связанные с дизайном, выводом данных и их вводом.

Было проведено тестирование программы с различным количеством вершин.

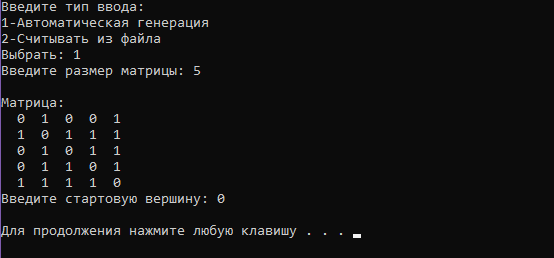


 Рисунок 5 – при вводе количества вершин = 5

Рисунок 6 – результат при кол. вершин = 5

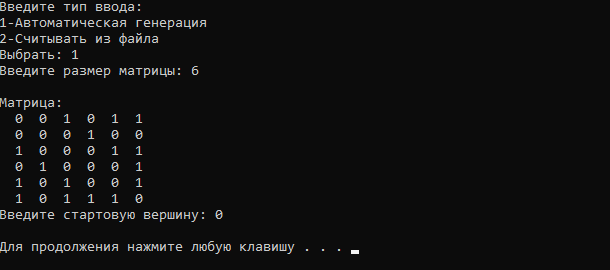


 Рисунок 7 – при вводе количества вершин = 6

Рисунок 8 – результат при кол. вершин = 6

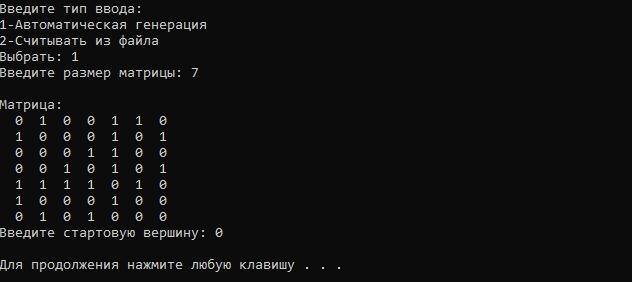


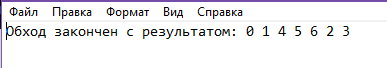
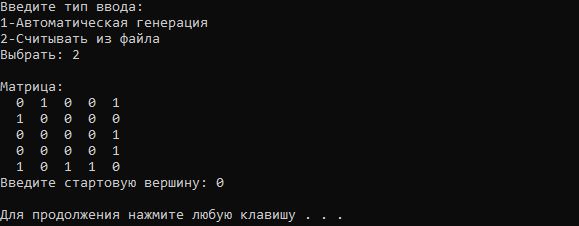
Рисунок 9 – при вводе количества вершин = 7

Рисунок 8 – результат при кол. вершин = 7



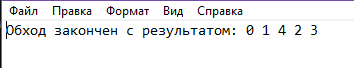
 Рисунок 10 – чтение матрицы с количеством вершин = 5

Рисунок 11 – результат при кол. вершин = 5

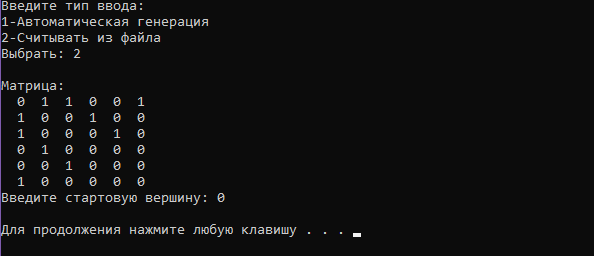


 Рисунок 12 – чтение матрицы с количеством вершин = 6

Рисунок 13 – результат при кол. вершин = 6

**Ручной расчёт задачи**

Проведем проверку программы на примере графа с 6 вершинами (рисунок 7)

Начинаем обход с 0. Ищем есть ли путь из этой вершины в другие. Находим три пути 2, 4 и 5. Выводим ноль и продолжаем цикл. В очереди находиться три вершины {2,4,5}, значит берём 2 вершину и смотрим с какой она смежная. Из вершины 2 пути идут в 4 и 5, но они уже есть в очереди, значит просто выводим 2. Следующая в очереди {4,5} идёт 4 из неё путь в 0, 2 и 5, но 0 и 2 мы уже вывели, а 5 находиться в очереди, выводим 4. Далее идёт 5 вершина и из ещё не посещенных мы можем попасть в 3, заносим её в очередь, выводим 5. Из 3 есть путь в 1, добавляем в очередь и выводим 3. И последняя вершина 1, путей из неё нет, значит выводим её и цикл заканчивается, так как очередь стала пустой.

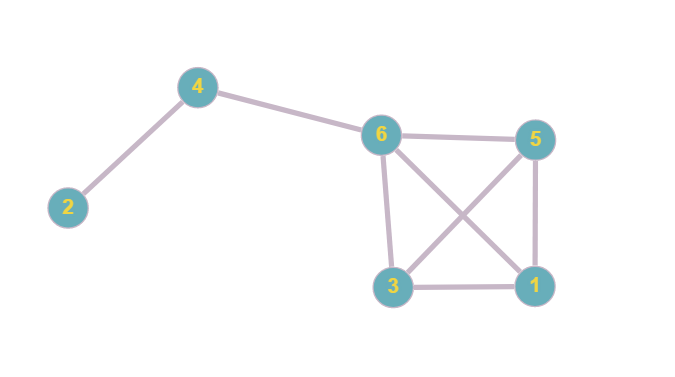


Рисунок 14 – пример графа из рис. 7

**Заключение**

Подводя итог курсовой работы, была разработана программа реализующая алгоритм поиска в ширину в среде разработки Microsoft Visual Studio.

В процессе которой, были приобретены навыки работы с графами, способы созданий матриц смежности и изучение принципов работы алгоритма поиска в ширину. Также были улучшены навыки и знания программирования на языке Си.

Главными недостатками программы является простой пользовательский интерфейс без введения дополнительных элементов дизайна

Преимущество данной работы считаю потенциал для дальнейшего развития и усовершенствования.

**Список литературы**

1. Белов В. В., Воробьев Е. М., Шаталов В. Е. - Теория графов. — М.: Высш. школа, 1976. — С. 392.

2. Берж К. - Теория графов и ее приложения. М.: ИЛ, 1962. 320c.

3. Интернет ресурс MAXimal - <https://e-maxx.ru/algo/bfs>

4. Интернет ресурс Habr - <https://habr.com/ru/post/504374/>

**Приложение А.**

**Листинг программы.**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include "conio.h"

#include "stdlib.h"

#include "time.h"

#include "locale.h"

#include <queue>

std::queue <int> Q;

int\* num;

void bfs(int v, int\*\*matrix,int size)

{

FILE\* file = fopen("results.txt", "a");

Q.push(v);

num[v] = 1;

while (Q.empty() != true)

{

v = Q.front();

Q.pop();

fprintf(file,"%d ", v);

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

{

if ((matrix[v][i] == 1) && (num[i] == 0))

{

Q.push(i);

num[i] = 1;

}

}

}

fclose(file);

}

int main()

{

FILE\* file = fopen("results.txt", "w");

int s;

int size = 0;

int\*\* matrix;

setlocale(LC\_ALL, "Russian");

while (true) {

printf("Введите тип ввода:\n1-Автоматическая генерация\n2-Считывать из файла\n");

int answer;

printf("Выбрать: ");

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

if (answer == 1) {

printf("Введите размер матрицы: ");

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

num = (int\*)malloc(size \* sizeof(int));

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

{

num[i] = 0;

}

printf("\n");

matrix = (int\*\*)malloc(size \* sizeof(int\*));

srand(time(NULL));

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

{

matrix[i] = (int\*)malloc(size \* sizeof(int));

}

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

{

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

{

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

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

if (matrix[i] == matrix[j])

matrix[i][j] = 0;

}

}

break;

}

else if (answer == 2) {

FILE\* input = fopen("test.txt", "r");

char a[10];

fgets(a,10,input);

for (int i = 0; i < 10; i++)

{

if (a[i] >= '0' && a[i] <= '9') {

size = size \* 10 + (a[i] - '0');

}

}

num = (int\*)malloc(size \* sizeof(int));

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

{

num[i] = 0;

}

printf("\n");

matrix = (int\*\*)malloc(size \* sizeof(int\*));

srand(time(NULL));

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

{

matrix[i] = (int\*)malloc(size \* sizeof(int));

}

char\* buffer = (char\*)malloc(sizeof(char) \* 2 \* size+1);

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

{

fgets(buffer, size \* 2+1, input);

for (int j = 0; j < size \* 2; j = j + 2)

{

matrix[i][j / 2] = (int)buffer[j] - '0';

}

}

fclose(input);

break;

}

else {

printf("Введенно неверное значение, повторите попытку!\n");

printf("\n");

}

}

printf("Матрица:\n");

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

{

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

{

printf("%3d", matrix[i][j]);

}

printf("\n");

}

printf("Введите стартовую вершину: ");

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

printf("\n");

fprintf(file, "Обход закончен с результатом: ");

fclose(file);

bfs(s,matrix,size);

system("PAUSE");

return 0;

}