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 поиска путей в лабиринте”

Выполнила:

студентка группы 19вв3

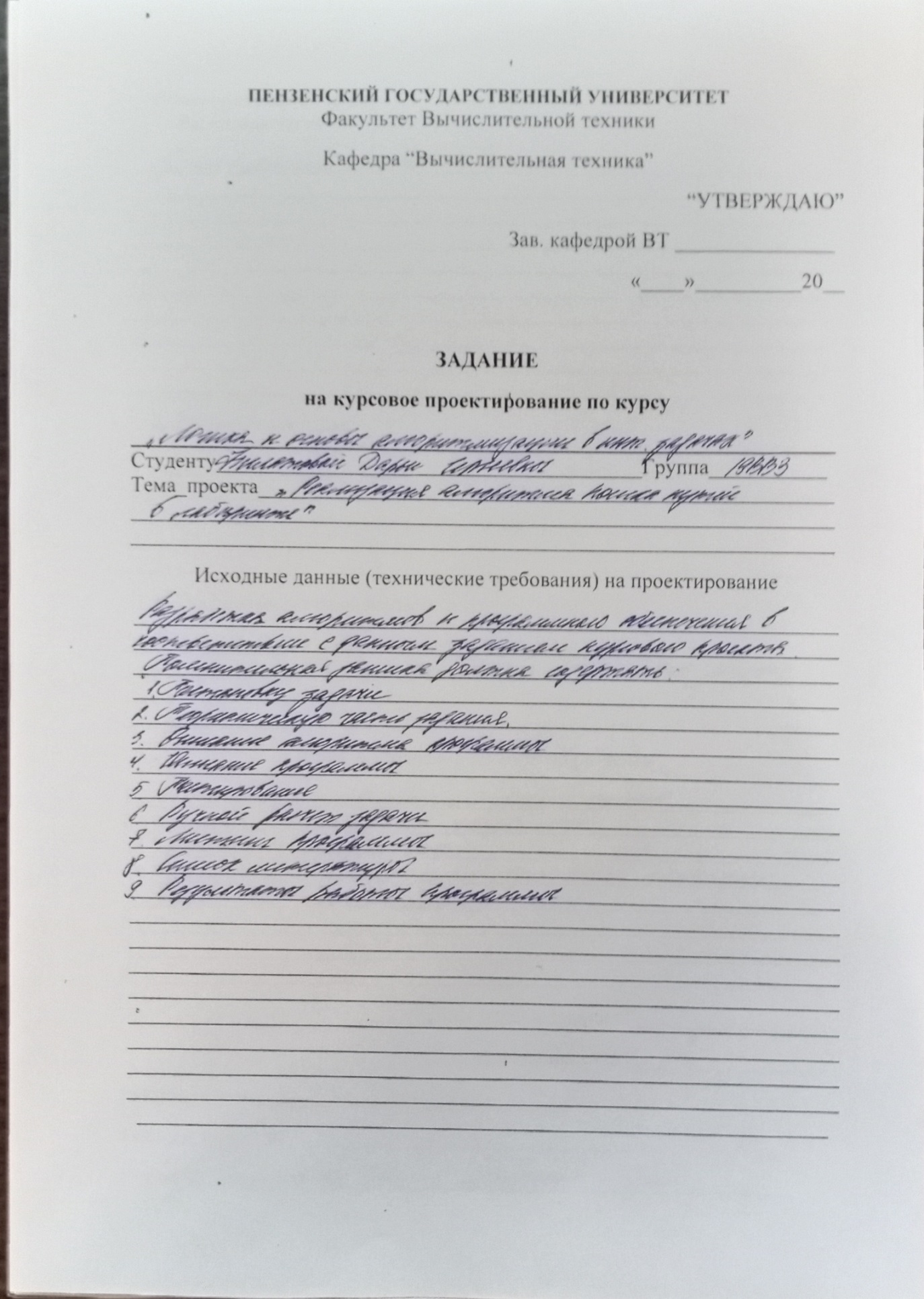
Филатова Д.С.

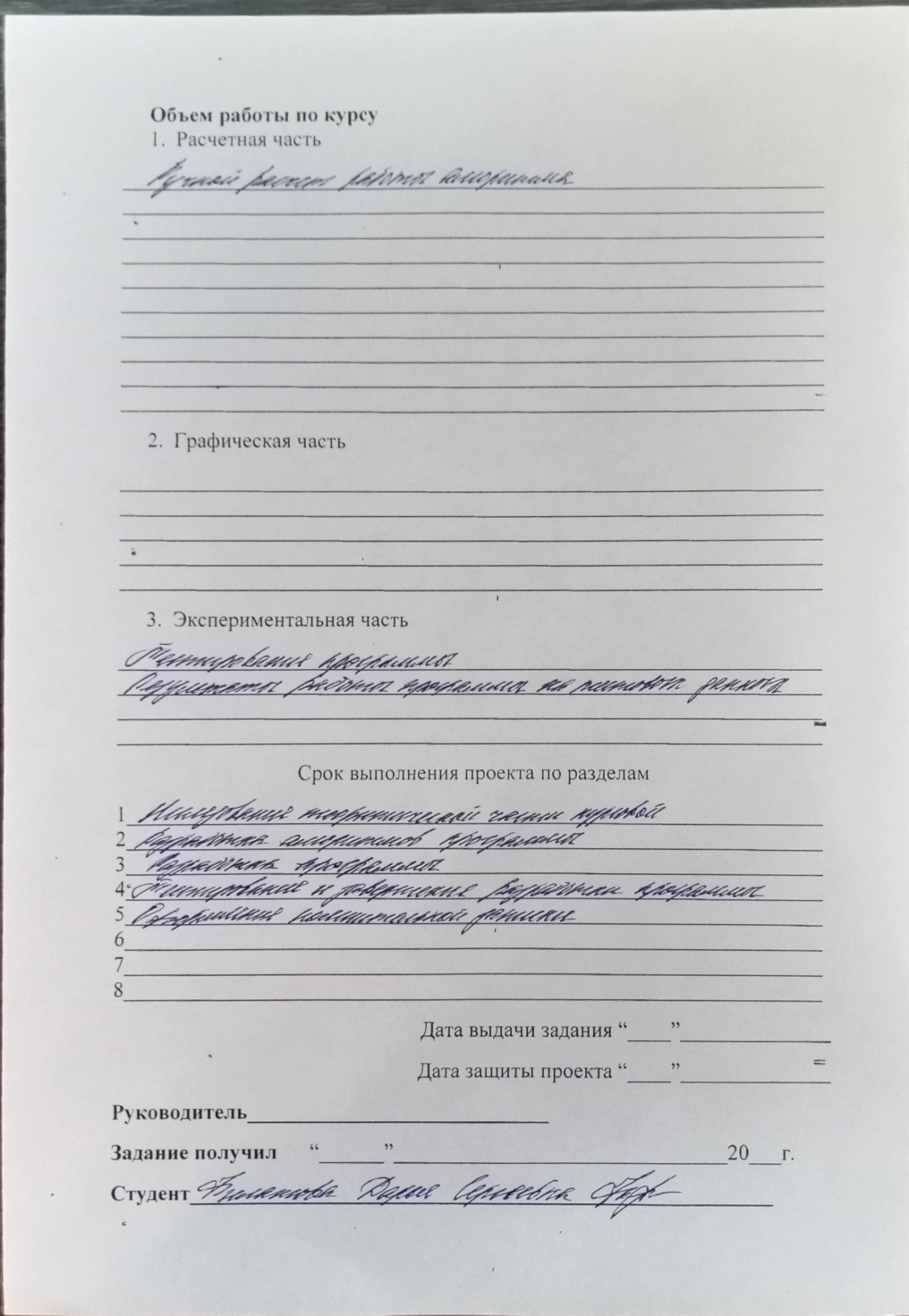
Принял:

Зав. кафедры ВТ

Митрохин М.А.

Пенза 2020





**Содержание**

[Реферат 4](#_TOC_250009)

[Введение 5](#_TOC_250008)

1.[Постановка задачи 6](#_TOC_250007)

2.[Теоретическая часть задания 7](#_TOC_250006)

3.[Описание алгоритма программы](#_TOC_250005) 8

4.[Описание программы 1](#_TOC_250004)0

5.[Тестирование](#_TOC_250003) 12

6.[Ручной расчёт задачи](#_TOC_250002) 14

[Заключение](#_TOC_250001) 17

[Список литературы](#_TOC_250000) 18

Приложение A. Листинг программы 19

# Реферат

Отчет 28 стр, 7 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, ОРГРАФ, ДОСТИЖИМОСТЬ,

ЛАБИРИНТ, АЛГОРИТМ ЛИ.

Цель исследования – разработка программы для поиска путей в лабиринте.

В работе рассмотрена реализация алгоритма ли для поиска путей в лабиринте. Установлено, что с помощью данного алгоритма можно найти кротчайший путь в лабиринте.

# Введение

Алгоритм волновой трассировки (волновой алгоритм, алгоритм Ли) —алгоритм поиска пути, алгоритм поиска кратчайшего пути на планарном графе. Принадлежит к алгоритмам, основанным на методах поиска в ширину.

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 поиска кротчайшего пути в лабиринте.

# 1.Постановка задачи

Требуется разработать программу, которая находит путь в лабиринте.

Исходный лабиринт в программе должен задаваться матрицей, причём при генерации данных должны быть предусмотрены граничные условия. После обработки этих данных на экран должен выводиться лабиринт и путь. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работала правильно.

Задания выполняются в соответствии с вариантом №12.

# 2.Теоретическая часть задания

Алгоритм работает на дискретном рабочем поле (ДРП), представляющем собой ограниченную замкнутой линией фигуру, не обязательно прямоугольную, разбитую на прямоугольные ячейки, в частном случае — квадратные. Множество всех ячеек ДРП разбивается на подмножества: «проходимые» (свободные), т. е при поиске пути их можно проходить, «непроходимые» (препятствия), путь через эту ячейку запрещён, стартовая ячейка (источник) и финишная (приемник). Назначение стартовой и финишной ячеек условно, достаточно — указание пары ячеек, между которыми нужно найти кратчайший путь.

Алгоритм предназначен для поиска кратчайшего пути от стартовой ячейки к конечной ячейке, если это возможно, либо, при отсутствии пути, выдать сообщение о непроходимости.

Работа алгоритма включает в себя три этапа: **инициализацию**, **распространение волны** и **восстановление пути**.

Во время инициализации строится образ множества ячеек обрабатываемого поля, каждой ячейке приписываются атрибуты проходимости/непроходимости, запоминаются стартовая и финишная ячейки.

Далее, от стартовой ячейки порождается шаг в соседнюю ячейку, при этом проверяется, проходима ли она, и не принадлежит ли ранее меченной в пути ячейке.

# 3.Описание алгоритма программы

Для программной реализации алгоритма понадобиться массив: int map () – для хранения лабиринта. С помощью функции mazemake() мы генерируем лабиринт.

Вначале лабиринт массив заполняется стенками. Затем мы выбираем в лабиринте ячейку (1; 1) и начиная от нее начинаем создавать проходы. Далее начиная с ячейки (1; 1) мы начинаем поиск путей в лабиринте к ячейке (N-1; N-1), где N – размер лабиринта. Поиск происходит следующим образом, ячейку (1; 1) помечаем номером 0, далее используя вектор мы находим проходы рядом с текущей ячейкой и помечаем их номер на 1 больше, чем исходная, и заносим их координаты в вектор. Далее мы таким же образом просматриваем ячейки, которые поместили в вектор, пока не дойдём до ячейки (N-1; N-1). После мы очищаем вектор и начиная с ячейки (N-1; N-1) заносим ячейку значение которой на единицу меньше, чем у исходной, пока не доймём до 1. Затем выводим наш маршрут на экран.

Ниже представлен псевдокод части программы, которая отвечает за поиск кротчайшего пути в лабиринте.

1. Координаты ячейки (1; 1) заносятся в вектор oldWave.
2. nstep = 0;
3. ячейке матрицы (1; 1) присваивается значение 0.
4. Создается массив dx содержащий в себе значения { 0, 1, 0, -1 };
5. Создается массив dy содержащий в себе значения { -1, 0, 1, 0 };
6. Пока вектор oldWave > 0
7. nstep++;
8. вектор wave очищается.
9. Для i = началу вектора oldWave; пока i не равно концу вектора oldWave; i++;
   1. для d = 0; пока d < 4; d++;
      1. nx равно первой координате вектора + dx[d];
      2. ny равно первой координате вектора + dy[d];
      3. если в этой ячейке есть проход, то
         1. в вектор wave заносятся координаты nx, ny
         2. значение ячейки = nstep
         3. если nx = N – 2 и ny = N – 2, то выходим из всех циклов.
10. oldWave = wave;
11. x = N – 2;
12. y = N – 2;
13. чистим вектор wave;
14. заносим в вектор wave координаты x, y;
15. пока значение в ячейке не равно 0;
    1. для d = 0; пока d < 4; d++;
       1. nx равно первой координате вектора + dx[d];
       2. ny равно первой координате вектора + dy[d];
       3. если в этой ячейке с координатами x, y значение на единице меньше, чем в ячейке с координатами nx, ny, то
          1. x = nx;
          2. y = ny;
          3. в вектор wave заносятся координаты x, y.
16. вывод на экран.

Полный код программы можно увидеть в Приложении А.

# 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++).

Данная программа является многомодульной, поскольку состоит из нескольких функций: deadend, endend, mazemake, drawing, main.

Программа выводит сгенерированный лабиринт и кротчайший путь на экран. Сгенерированный лабиринт заносится в файл и при последующем запуске программы он берется из файла, а не генерируется снова.

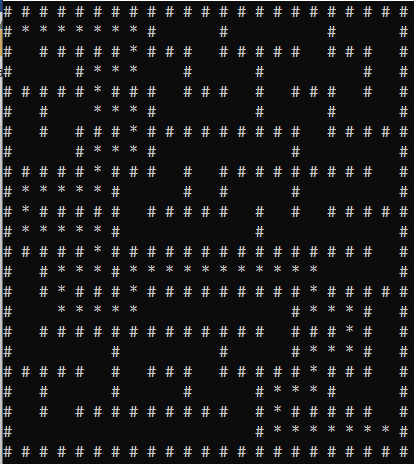


Рисунок 1 Выведенный на экран лабиринт

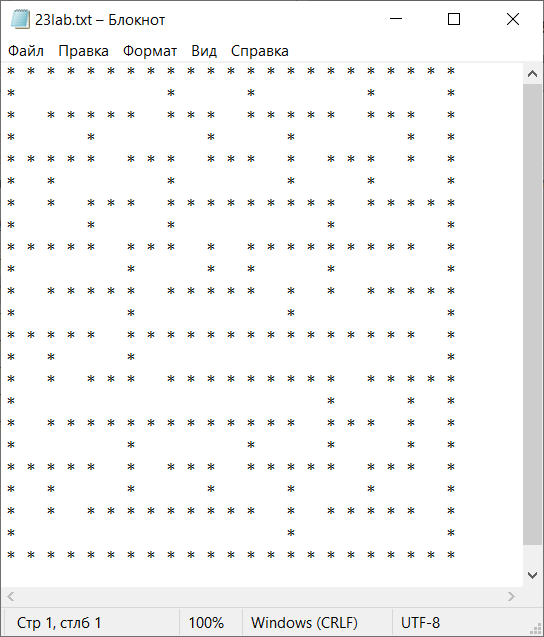


Рисунок 2 Лабиринт сохранённый в файле

# 5.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, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

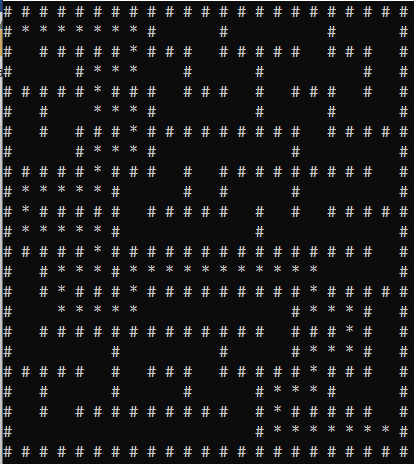


Рисунок 3

На рисунке 3 продемонстрирован результат тестирования при размерах лабиринта 23x23.



Рисунок 4

На рисунке 4 продемонстрирован результат тестирования при размерах лабиринта 43x43.

Таблица 1 – Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Запуск окна | Верно |
| Генерирование лабиринта | Вывод сгенерированного лабиринта на экран | Верно |
| Поиск пути | Вывод на экран кротчайшего пути в лабиринте | Верно |

В результате тестирования было выявлено, что программа успешно генерирует лабиринт и находит кротчайший путь.

**6.Ручной расчёт задачи**

Проведем проверку программы посредством ручных вычислений на примере лабиринта с размерами 23x23 (рисунок 3).

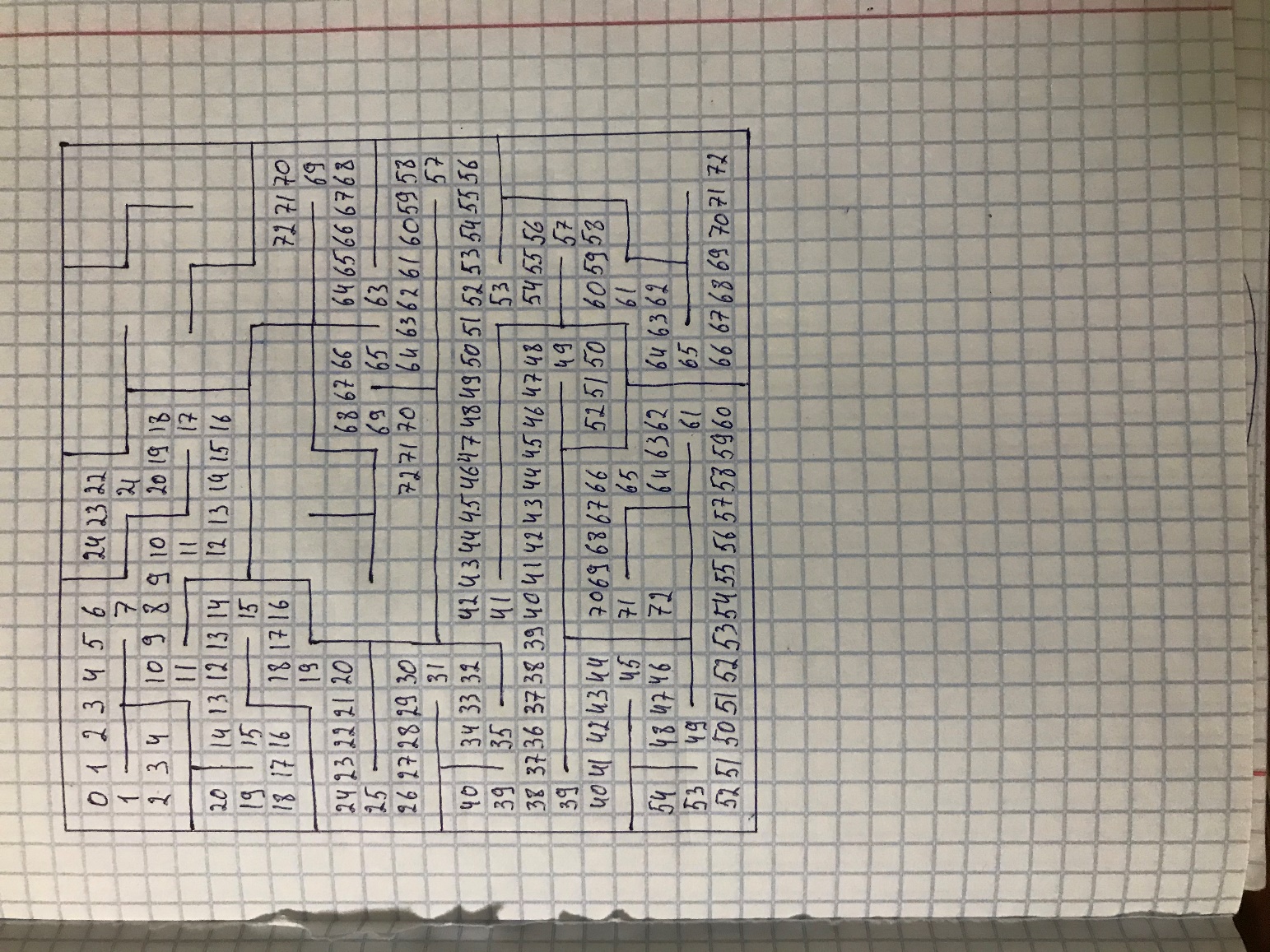
Начинаем из ячейки с координатами (1; 1). Помечаем эту ячейку нулём, проверяем есть ли рядом с ней свободные пути. В нашем случае есть свободный путь вниз и вправо. Помечаем их единицами. Далее просматриваем ячейки помеченные единицами и смотрим, есть ли рядом с ними свободные пути если есть, то помечаем свободные пути двойками и аналогично продолжаем проверять. Пока не дойдём до ячейки (22;22). 

Рисунок 5 ручной расчёт путей в лабиринте.

После того, как мы дошли до ячейки (22; 22), мы должны начиная с этой ячейки искать поблизости ячейку со значение на единицу меньше и так пока не дойдём до нуля, это будет наш кротчайший путь.

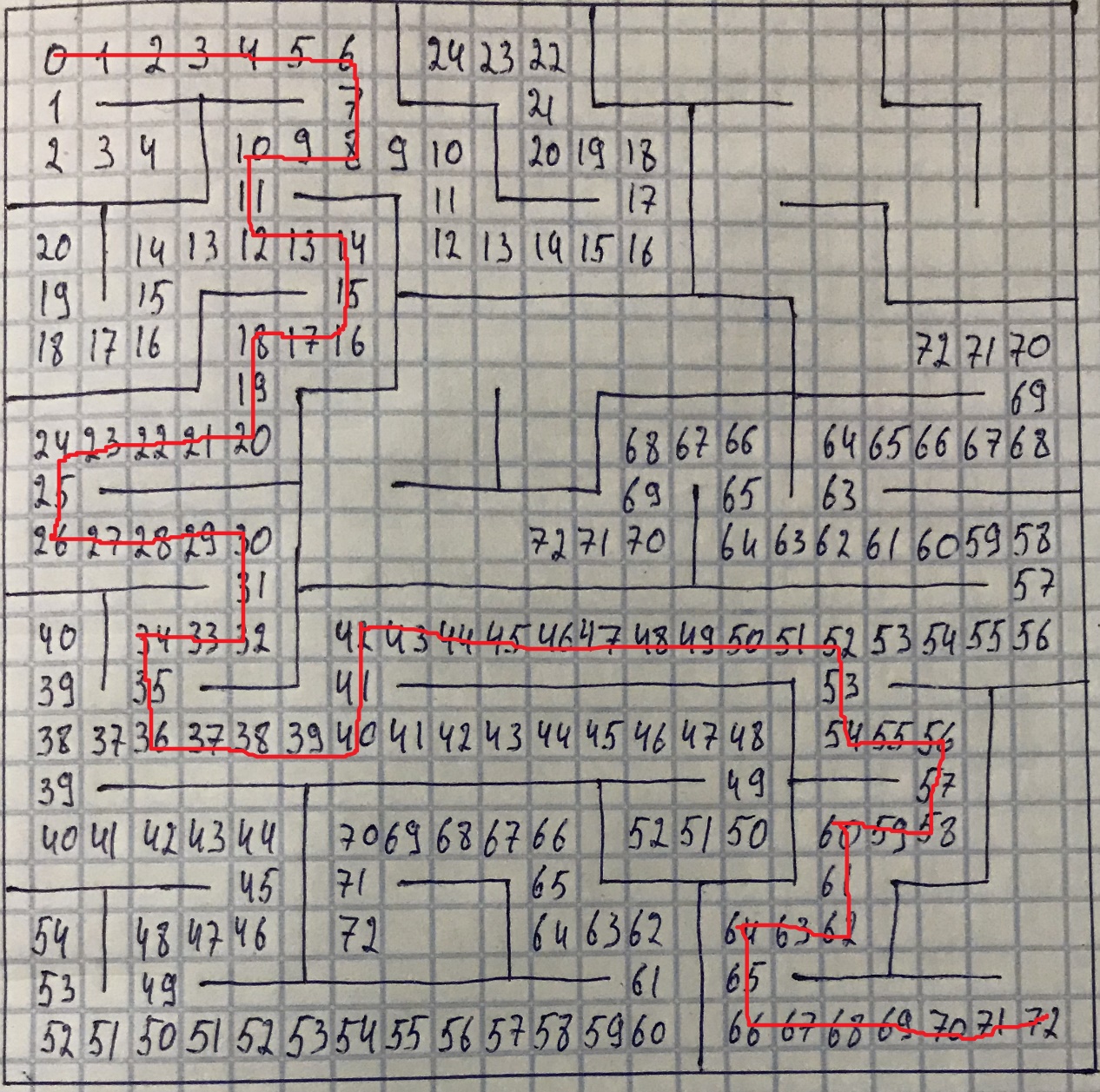


Рисунок 6 ручной расчёт восстановление пути.

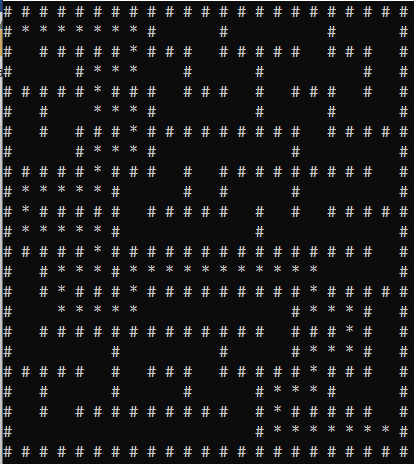


Рисунок 7 результат работы программы.

Сравнив результат работы программ и ручной расчёт, можно сделать вывод, что программа работает верно.

# Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм поиска кротчайшего пути в лабиринте в MicrosoftVisualStudio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания алгоритма, для генерации лабиринте. Приобретены навыки по осуществлению алгоритма поиска кротчайшего пути в лабиринте. Углублены знания языка программирования 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.
  6. 3. Оре О. Графы и их применение: Пер. с англ. 1965. 176 с.

## Приложение А.

**Листинг программы.**

#include <stdlib.h>

#include <vector>

#include <algorithm>

#include <ctime>

#include <string>

#include <fstream>

#include <iostream>

using namespace std;

const int N = 33;

const int way = -3, wall = -2, pass = -1;

int map[N][N];

vector<pair<int, int>> wave;

bool deadend(int x, int y, int maze[N][N], int height, int width) {

int a = 0;

if (x != 1) {

if (maze[y][x - 2] == pass)

a += 1;

}

else a += 1;

if (y != 1) {

if (maze[y - 2][x] == pass)

a += 1;

}

else a += 1;

if (x != width - 2) {

if (maze[y][x + 2] == pass)

a += 1;

}

else a += 1;

if (y != height - 2) {

if (maze[y + 2][x] == pass)

a += 1;

}

else a += 1;

if (a == 4)

return 1;

else

return 0;

}

bool ended(int maze[N][N], int height, int width) {

bool b = 1;

for (int i = 1; i < (height - 1); i += 2)

for (int j = 1; j < (width - 1); j += 2)

if (maze[i][j] == wall)

b = 0;

return b;

}

void mazemake(int maze[N][N], int height, int width) {

int x, y, c, a;

bool b;

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

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

maze[i][j] = wall;

x = 3; y = 3; a = 0;

while (1)

{

a++;

if (a % 100 == 0)

if (ended(maze, height, width))

break;

maze[y][x] = pass;

while (1) {

c = rand() % 4;

switch (c) {

case 0: if (y != 1)

if (maze[y - 2][x] == wall) {

maze[y - 1][x] = pass;

maze[y - 2][x] = pass;

y -= 2;

}

case 1: if (y != height - 2)

if (maze[y + 2][x] == wall) {

maze[y + 1][x] = pass;

maze[y + 2][x] = pass;

y += 2;

}

case 2: if (x != 1)

if (maze[y][x - 2] == wall) {

maze[y][x - 1] = pass;

maze[y][x - 2] = pass;

x -= 2;

}

case 3: if (x != width - 2)

if (maze[y][x + 2] == wall) {

maze[y][x + 1] = pass;

maze[y][x + 2] = pass;

x += 2;

}

}

if (deadend(x, y, maze, height, width))

break;

}

if (deadend(x, y, maze, height, width))

do {

x = 2 \* (rand() % ((width - 1) / 2)) + 1;

y = 2 \* (rand() % ((height - 1) / 2)) + 1;

} while (maze[y][x] != pass);

}

}

void drawing()

{

system("cls");

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

{

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

{

if (map[i][j] == wall)

cout << "# ";

else if (map[i][j] == way)

cout << "\* ";

else

cout << " ";

}

cout << '\n';

}

}

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

{

srand(time(NULL));

string name = std::to\_string(N) + "lab.txt";

ifstream fin(name);

if (!fin.is\_open())

{

fin.close();

ofstream fout(name);

mazemake(map, N, N);

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

{

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

{

if (map[i][j] == wall)

fout << "\* ";

else

fout << " ";

}

fout << '\n';

}

fout.close();

}

else

{

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

{

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

{

if (fin.get() == '\*')

map[i][j] = wall;

else

map[i][j] = pass;

fin.seekg(1, ios\_base::cur);

}

fin.seekg(2, ios\_base::cur);

}

fin.close();

}

drawing();

map[1][1] = pass;

vector<pair<int, int>> oldWave;

oldWave.push\_back(pair<int, int>(1, 1));

int nstep = 0;

map[1][1] = nstep;

const int dx[] = { 0, 1, 0, -1 };

const int dy[] = { -1, 0, 1, 0 };

while (oldWave.size() > 0)

{

++nstep;

wave.clear();

for (vector < pair<int, int>> ::iterator i = oldWave.begin(); i != oldWave.end(); i++)

{

for (int d = 0; d < 4; ++d)

{

int nx = i->first + dx[d];

int ny = i->second + dy[d];

if (map[nx][ny] == pass)

{

wave.push\_back(pair<int, int>(nx, ny));

map[nx][ny] = nstep;

if (nx == N - 2 && ny == N - 2)

goto done;

}

}

}

oldWave = wave;

}

done:

int x = N - 2;

int y = N - 2;

wave.clear();

wave.push\_back(pair<int, int>(x, y));

while (map[x][y] != 0)

{

for (int d = 0; d < 4; ++d)

{

int nx = x + dx[d];

int ny = y + dy[d];

if (map[x][y] - 1 == map[nx][ny])

{

x = nx;

y = ny;

wave.push\_back(pair<int, int>(x, y));

}

}

}

for (vector < pair<int, int>> ::iterator i = wave.begin(); i != wave.end(); ++i)

{

map[i->first][i->second] = way;

//drawing();

}

drawing();

}