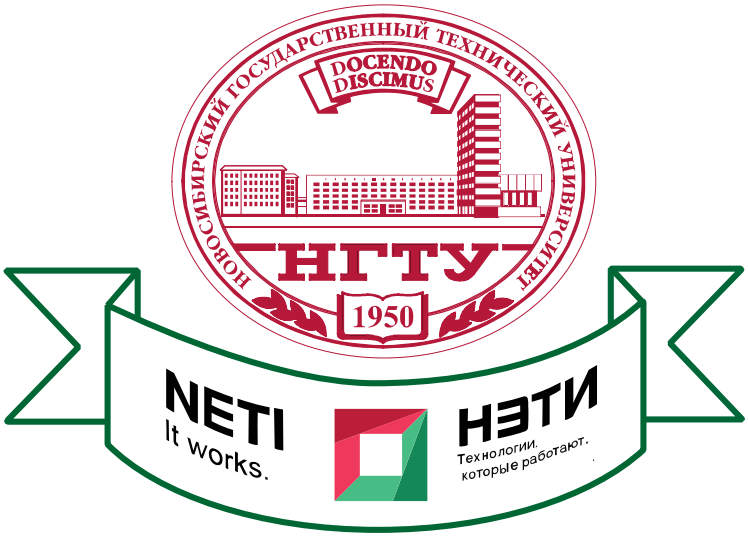
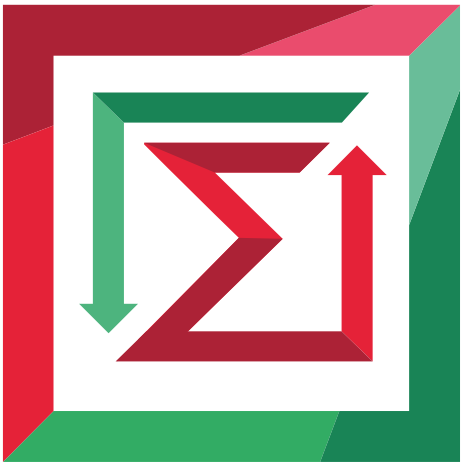
Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Новосибирский государственный технический университет»



Кафедра прикладной математики

Курсовая работа
по дисциплине «Структуры данных и алгоритмы»



Факультет: ПМИ

Группа: ПМ-71

Студент: Востриков В.А.

Преподаватель: Тракимус Ю.В.

Новосибирск
2019

1. Цель работы

Найти кратчайший путь передвижения ладьи по заданному клеточному полю, соединяющий две указанные его клетки.

2. Анализ задачи

2.1. Исходные данные задачи: «карта» шахматной доски с информацией о начальной и конечной точек маршрута, о положении «стен» на «карте».

2.2. Результат: в случае успеха координаты движения ладьи, провала – сообщение об отсутствии решения либо ошибки.

2.3. Решение: улучшить алгоритм волновой трассировки (алгоритм Ли) согласно нашей задачи.

Алгоритм предназначен для поиска кратчайшего пути от стартовой ячейки к конечной ячейке, если это возможно.

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.

Соседние ячейки принято классифицировать двояко: в смысле окрестности Мура и окрестности фон Неймана, отличающийся тем, что в окрестности фон Неймана соседними ячейками считаются только 4 ячейки по вертикали и горизонтали, в окрестности Мура — все 8 ячеек, включая диагональные (В данном случае классифицируем в смысле окрестности фон Неймана).

При выполнении условий проходимости и непринадлежности её к ранее помеченным в пути ячейкам, в атрибут ячейки записывается число, равное количеству шагов от стартовой ячейки, от стартовой ячейки на первом шаге это будет 1. Каждая ячейка, меченная числом шагов от стартовой ячейки, становится стартовой и из неё порождаются очередные шаги в соседние ячейки. Очевидно, что при таком переборе будет найден путь от начальной ячейки к конечной, либо очередной шаг из любой порождённой в пути ячейки будет невозможен.

Восстановление кратчайшего пути происходит в обратном направлении: при выборе ячейки от финишной ячейки к стартовой на каждом шаге выбирается ячейка, имеющая атрибут расстояния от стартовой на единицу меньше текущей ячейки. Очевидно, что таким образом находится кратчайший путь между парой заданных ячеек. Трасс с минимальной числовой длиной пути, как при поиске пути в окрестностях Мура, так и фон Неймана может существовать несколько. Выбор окончательного пути в приложениях диктуется другими соображениями, находящимися вне этого алгоритма. Например, при трассировке печатных плат — минимумом линейной длины проложенного проводника.

Для лучшего понимания ниже приведены рисунки этапов распространения волны и восстановления пути.

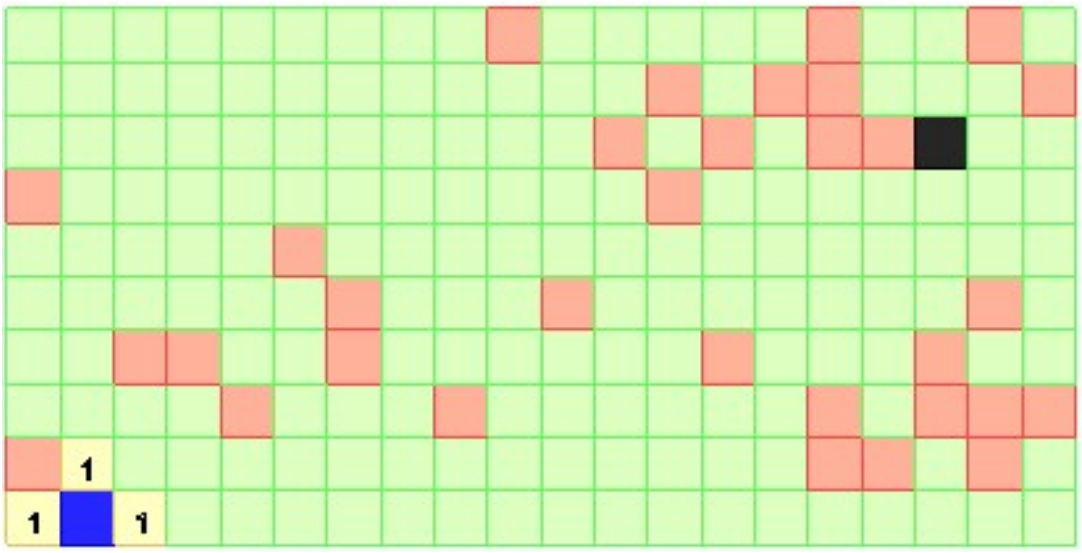


Рис.1. Этап распространения волны №1.

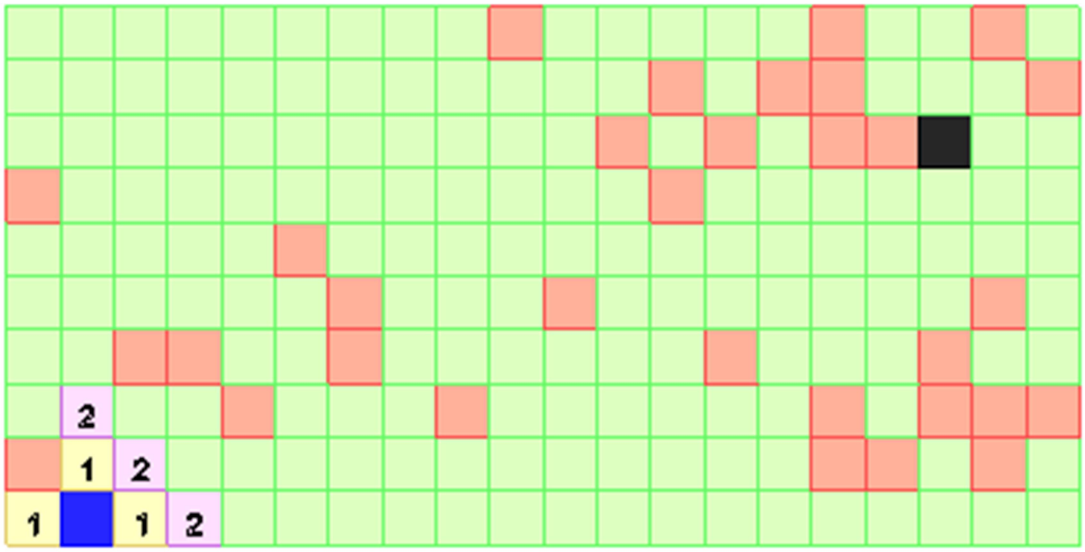


Рис.2. Этап распространения волны №2.

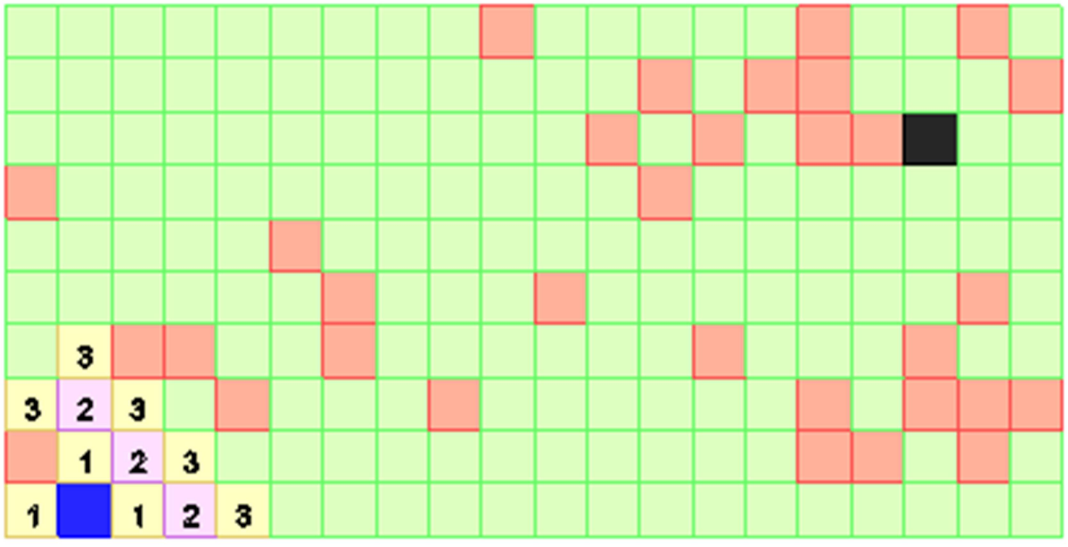


Рис.3. Этап распространения волны №3.

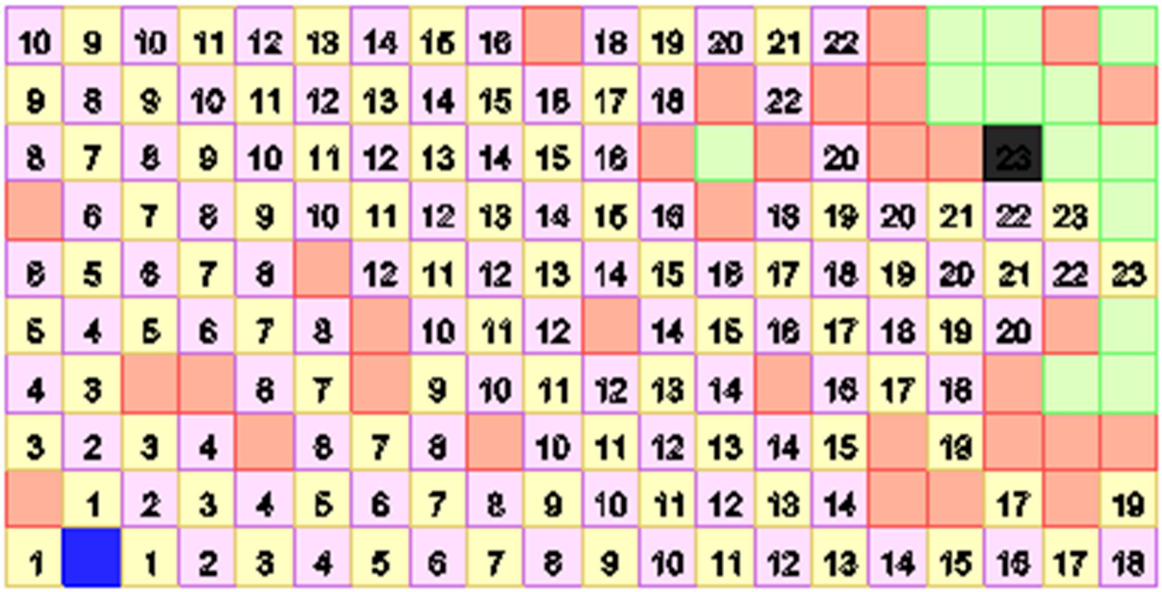


Рис.4. Этап распространения волны №4 (конец алгоритма распространения).

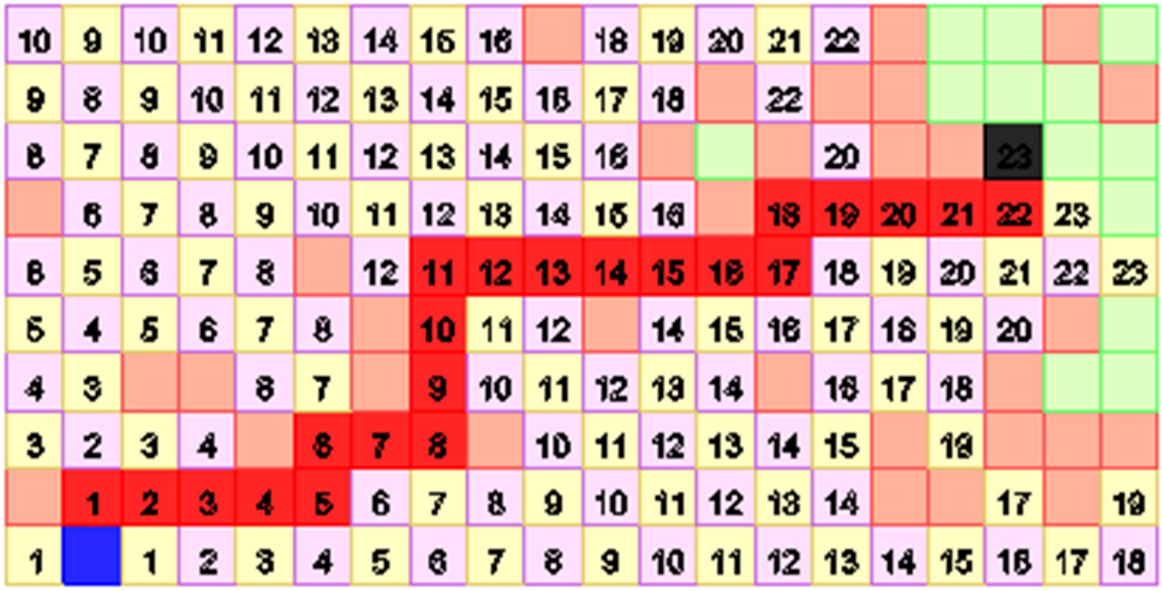


Рис.5. Этап восстановления пути.

На картинках выше показана работа алгоритма Ли в классификации окрестности фон Неймана. В таком случае необходимо понимать, что работа алгоритма Ли не будет учитывать количество совершенных ходов ладьи для достижения конечной клетки.

Тогда идеальным вариантом для решения этой задачи является решение, которое учитывает наименьшее количество ходов и наименьшее количество пройденных клеток ладьи (программа, которая будет приведена ниже, решает только задачу с наименьшим количеством ходов).

Постановка задачи:

Найти путь с наименьшим количеством ходов ладьи до конечной клетки.

3. Представление данных

3.1. Входные данные:

В качестве входных данных из файла (in.txt) считывается двумерный массив объектов (object_matrix[][]), который содержит информацию о положениях конечной и начальной позиции и стен (0 – пустая клетка, 1 – стена, 2 – начальная точка, 3 – конечная точка).

3.2. Выходные данные:

В качестве выходных данных в файл (out.txt) выводятся координаты точек маршрута из массивов px[] и py[], которые записывают координаты по оси Ox и Oy соответственно.

4. Алгоритм решения задачи

Подпрограмма int main() сначала запускает подпрограмму считывания матрицы объектов read_objects(), после запускает подпрограмму корректировки данных correct_matrix(), после чего запускает улучшенный алгоритм трассировки lee_upgrade(), в конце которого запускается подпрограмма вывода out_func().

Рассмотрим lee_upgrade() поподробнее. Как говорилось выше, алгоритм состоит из трех этапов: инициализация, распространение волны и восстановление пути. Инициализация происходит в подпрограмме correct_matrix(). В подпрограмме lee_upgrade() остается 2 этапа: распространение волны и восстановление пути.

5. Текст программы

```
#include <stdio.h>
#include <iostream>

/*
0 - пустая клетка
1 - стена
2 - начало
3 - конечная точка
*/

const int W = 8;  // ширина рабочего поля
const int H = 8;  // высота рабочего поля
const int WALL = -1;   // непроходимая ячейка
const int BLANK = -2;   // свободная непомеченная ячейка

int object_matrix[H][W];
int xs, ys, xf, yf;  // начальные и конечные координаты
// s - start, f - final

int px[W * H], py[W * H]; // координаты ячеек, входящих в путь
int len; // длина пути


int read_objects()
{
    FILE* in = fopen("in.txt", "r");
    int i, j;
    for (i = 0; i < 8; i++)
    {
        for (j = 0; j < 8; j++)
            fscanf(in, "%d", &object_matrix[i][j]);
        j = 0;
    }
    fclose(in);
```

```
    return 1;
}

int correct_matrix()
{
   xf = -3, yf = -3, xs = -3, ys = -3; // присваиваем недопустимые значения
   for (int i = 0; i < 8; i++)
      for (int j = 0; j < 8; j++)
      {
         if (object_matrix[i][j] == 0)
            object_matrix[i][j] = BLANK;
         else
         {
            if (object_matrix[i][j] == 1)
               object_matrix[i][j] = WALL;
            else
            {
               if (object_matrix[i][j] == 2)
               {
                  xs = i;
                  ys = j;
               }
               else
               {
                  xf = i;
                  yf = j;
               }
               object_matrix[i][j] = -2;
            }
         }
      }
   if (xf == -3 && xs == -3)
   {
      printf_s("Отсутствую данные о начальной и конечной точках.\n");
      return 0;
   }
   else if (xf == -3)
   {
      printf_s("Отсутствую данные о конечной точке.\n");
      return 0;
   }
   else if (xs == -3)
   {
      printf_s("Отсутствую данные о начальной точке.\n");
      return 0;
   }
    return 1;
}

void out_func()
{
    FILE* out = fopen("out.txt", "w");

    for (size_t i = 0; i < len + 1; i++)
        fprintf(out, "(%d, %d) ", px[i], py[i]);
}

bool lee_upgrade(int ax, int ay, int bx, int by)  // поиск пути из ячейки (ax, ay) в ячейку
(bx, by)
{
    int dx[4] = { 1, 0, -1, 0 };   // смещения, соответствующие соседям ячейки
    int dy[4] = { 0, 1, 0, -1 };   // справа, снизу, слева и сверху
    int d, x, y, k;
    bool stop;
```

```
    if (object_matrix[ax][ay] == WALL || object_matrix[bx][by] == WALL) return false;  //
ячейка (ax, ay) или (bx, by) - стена
   if (ax == bx && ay == by)
   {
      len = 0;
      px[0] = ay;
      py[0] = ax;
      out_func();
      return true;
   }
    // распространение волны
    d = 0;
    object_matrix[ax][ay] = 0; // стартовая ячейка помечена 0
    do
    {
        stop = true;   // предполагаем, что все свободные клетки уже помечены
        for (y = 0; y < H; ++y)    // Ищем клетку с меткой d
            for (x = 0; x < W; ++x)
                if (object_matrix[x][y] == d) // ячейка (x, y) помечена числом d
                {
                    for (k = 0; k < 4; ++k)  // проходим по всем непомеченным соседям
                    {
                        int iy = y + dy[k], ix = x + dx[k];
                        while (object_matrix[ix][iy] != WALL && iy >= 0 && iy < H && ix >= 0
&& ix < W) // ... Пока не упремся в стену
                        {
                            if (iy >= 0 && iy < H && ix >= 0 && ix < W &&
object_matrix[ix][iy] == BLANK && object_matrix[ix][iy] != WALL) // Ищем непомеченные клетки
                            {
                                stop = false;    // найдены непомеченные клетки
                                object_matrix[ix][iy] = d + 1;   // распространяем волну
                            }
                            iy += dy[k];
                            ix += dx[k];
                        }
                    }
                }
        d++;
    } while (!stop && object_matrix[bx][by] == BLANK);

    if (object_matrix[bx][by] == BLANK) return false;  // путь не найден

    // восстановление пути
    len = object_matrix[bx][by];  // количество ходов кратчайшего пути из (ax, ay) в (bx,
by)
    x = bx;
    y = by;
    d = len;


    while (d > 0) // Пока не дойдем до начальной точки
    {
        // записываем ячейку (x, y) в путь
        px[d] = y;
        py[d] = x;
        d--;
        for (k = 0; k < 4 && !stop; ++k) // Проверяем всевозможные направления
        {
            int iy = y, ix = x;
            while (iy >= 0 && iy < H && ix >= 0 && ix < W && object_matrix[ix][iy] > 0) //
двигаемся до того момента, пока упремся в стену
            {
                iy = iy + dy[k], ix = ix + dx[k];
```

```
                if (ix < 8 && iy < 8 && object_matrix[ix][iy] == d) // если нашли клетку с
меньшим ходом на 1
                {
                    stop = true;
                    x = ix;  // переходим в ячейку, которая на 1 ближе к старту
                    y = iy;
                    break;
                }
            }

        }
        stop = false;
    }
    // теперь px[0..len] и py[0..len] - координаты ячеек пути
    px[0] = ay;
    py[0] = ax;

    out_func();
    return true;
}

int main()
{
    setlocale(LC_ALL, "rus");
    if (read_objects())
    {
        if (correct_matrix())
        {
            if (lee_upgrade(xs, ys, xf, yf))
                printf_s("Решение найдено!\n");
            else
                printf_s("Решение не найдено!\n");
        }
        else
            printf_s("Ошибка в данных.\n");
    }
    else
        printf_s("Ошибка при считывании из файла.\n");
    printf_s("Нажмите любую клавишу для выхода из программы.\n");
    getchar();
}
```

6. Набор тестов

№	Содержимое in.txt	Назначение
1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Минимальный тест
2	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 2 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 3 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0	Тест с препятствиями (проходимый)
3	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 2 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 3 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0	Тест с препятствиями (непроходимый)
4	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0	Тест, в котором отсутствуют начальная и конечная точки
5	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 3 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0	Тест, в котором отсутствует начальная точка
6	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 0 2 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0	Тест, в котором отсутствует конечная точка

7. Результаты работы программы

№	Содержимое out.txt	Вывод консоли
1	(1, 3) (7, 3) (7, 5)	Решение найдено!
2	(1, 3) (1, 1) (5, 1) (5, 5) (7, 5)	Решение найдено!

3	----------	Решение не найдено!
4	----------	Отсутствую данные о начальной и конечной точках. Ошибка в данных.
5	----------	Отсутствую данные о начальной точке. Ошибка в данных.
6	----------	Отсутствую данные о конечной точке. Ошибка в данных.