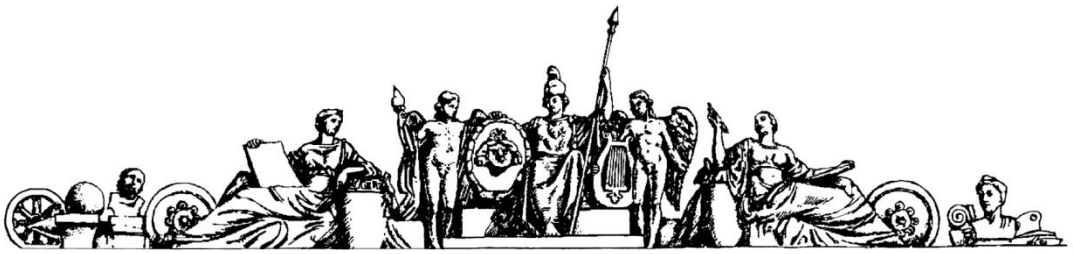

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования

"Московский государственный технический университет им. Н.Э. Баумана (национальный исследовательский университет)"
(МГТУ им. Н.Э. Баумана)

Факультет: Информатика и системы управления (ИУ)
Кафедра: Информационная безопасность (ИУ8)

Отчет по домашнему заданию по дисциплине: "Алгоритмы и структуры данных"

Преподаватели: Чесноков В.О.
Ключарев П.Г.

Студент: Теплухин Т.Г.
группа ИУ8-53

Москва, 2016

1. Постановка задачи

На некотором квадратном поле каждая клетка поля может либо содержать одно число от 1 до 6, либо быть пустой. Выделены начальная и конечная клетки, в каждой обязательно есть число. С каждой клетки можно двигаться на 1 шаг по одному из 8 направлений. Задача заключается в том, чтобы из начальной точки попасть в конечную, задав в соответствие каждой цифре направление движения. При этом путь не должен содержать пустых клеток.

Вход — поле с цифрами, координаты начальной и конечной точки.

Выход — набор пар цифра-направление.

Если всем числам сопоставить направление, то задача сводится к поиску пути в лабиринте, причем путь только 1 без ответвлений. Если встречается клетка без цифры или уже пройденная клетка, то пути к конечной клетке не существует.

Так как цифрам не сопоставлены направления, то образуется большое число лабиринтов.

Количество цифр 6, каждой может соответствовать одно направление, разным цифрам не может соответствовать одно и то же направление.

Число размещений из n по k:

$$A_n^k = \frac{n!}{(n-k)!}$$

В нашем случае n = 8, k = 6. Количество вариантов соответствия цифр направлениям равно 8!/(8-6)! = 40320 / 2 = 20160.

Для поиска решения можно использовать поиск в глубину или поиск в ширину.

Список возможных путей от начальной точки можно представить в виде дерева в вершине которой находится цифра в начальной точке. От вершины расходятся до 8 ребер (по количеству непустых клеток вокруг). Далее от каждой ранее не встречающейся цифры также расходятся до 8 ребер. От ранее встречающихся цифр выходит по 1 ребру. Получается дерево, которое в ветвится у вершины. Глубина дерева не более N*N-1, где N — размер поля).

Применение поиска в ширину потребует много ресурсов, особенно учитывая большое ветвление дерева.

Поэтому применим поиск в глубину, так как не требуется поиск самого короткого пути, глубина дерева небольшая (потребуется небольшое количество ресурсов).

Чтобы ускорить поиск результата, сделаем его эвристическим. Так как известны координаты начальной и конечной точки, то сначала будем выбирать направление, которое наиболее приближает к конечной вершине.

Для решения задача необходимо иметь матрицу NxN с цифрами, которые содержатся в клетках поля. Также необходимо отмечать пройденные клетки поля, для этого также будет использоваться матрица NxN.

Требуется запоминать, какие цифры для каких направлений использовались. Для этого нужен массив из 6 значений, где для каждой цифры задается направление, или признак того, что направление еще не задано.

2. Инструкция по использования программы

Программа разработана в среде программирования Visual Studio 2015 на языке программирования C++. Проект Maze.vcxproj включен в решение Mase.sln. После компиляции проекта создается исполняемый файл Maze.exe.

Программа является консольной. Входные параметры, имя входного и имя выходного файла, передаются как параметры командной строки.

Во входном файле описываются числа поля, координаты начальной и конечной точки. В выходной файл записывается код результата выполнения и сопоставление направлений цифрам.

3. Описание логики программы

Данные для решения задачи хранятся в структуре *Data*, чтобы их было удобно передавать в рекурсивной функции:

```
struct Data
{
        int N; // размер поля
        int **matr; // поле
        int ni, nj, ki, kj; // начальная и конечная точки

        int **check; // отметка пройденных клеток
        int dir[6]; // направление для каждой цифры (1..6)
        int digit[8]; // цифра для каждого направления (1..8)
};
```

Память для матрицы поля и матрицы отметки посещенных клеток в программе выделяется динамически. Для выделения и освобождения памяти разработаны функции *void new_matr(int ** &m, int n)* и *void free_matr(int ** &m, int n).*

Загрузка данных из входного файла выполняется в функции *int load_data(char * infile, Data *data)* на вход которой передается имя входного файла и структура хранящая данные.

Для определения *k*-го по значимости направления от текущей клетки *(i1,j1)* к конечной клетке *(i2,j2)* разработана функция *int best_dir(int i1, int j1, int i2, int j2, int k)*.

В данной функции сначала выбирается лучшее направление.

- Если номера строк совпадают, то направление — влево (если *j2* меньше *j1*) или вправо (если *j2* больше *j1*).
- Если номера столбцов совпадают, то направление — вверх (если *i2* меньше *i1*) или вниз (если *i2* больше *i1*).
- Если *i2* меньше *i1*, то направление — влево вверх (если *j2* меньше *j1*) или вправо вверх (если *j2* больше *j1*).
- Если *i2* больше *i1*, то направление — влево вниз (если *j2* меньше *j1*) или вправо вниз (если *j2* больше *j1*).
- Если *k* равно 0, то выдается найденное лучшее направление. Иначе при *k* равном 1 выдается ближайшее по часовой стрелке направление, при *k* равном 2 — ближайшее против часовой стрелки направление, при *k* равном 3 — следующее по часовой стрелке направление и т.д.

Функция *int solve(int pi, int pj, Data *data)* рекурсивно находит решение задачи. Входные параметры *pi* и *pj* — номер текущей клетки, *data* — указатель на структуру с данными.

- Если *pi,pj* — конечная клетка, то решение найдено, возвращаем значение 1.
- Если цифре в клетке уже присвоено направление — и нельзя переместиться в данном направлении (нет клетки или она пустая) — решение не найдено, возвращаем значение 0.

Отмечаем текущую клетка как пройденную, запускаем поиск для следующей клетки. Если найдено решение — выйти из функции со значением 1. Иначе убираем отметку текущей клетки как пройденной и возвращаем значение 0.

Если цифре в клетке не присвоено значение, то в цикле берем все направления в порядке уменьшения их значимости для направления от текущей клетки к конечной.

Если нельзя переместиться в данном направлении (нет клетки или она пустая) — переходим к следующей итерации цикла.

Иначе отмечаем текущую клетку как пройденную, назначаем цифре направление и направлению цифру, запускаем поиск для следующей клетки.

Если найдено решение – выйти из функции со значением 1. Иначе убираем отметку текущей клетки как пройденной, убираем для цифры направление и для направления цифру, переходим к следующей итерации цикла.

Если направления закончились — решение не найдено, возвращаем 1.

В функции *int solve_task(char * infile, char * outfile)* выполняется чтения данных из входного файла *infile*, решение задачи и вывод результата в выходной файл *outfile*.

4. Формат входных и выходных данных

Имя входного и выходного файла передается программе через параметры командной строки.

Формат входного файла:

1. размер поля *N* (положительное число);
2. поле с цифрами. N строк, в каждой строке *N* цифр от 1 до 6;
3. координаты начальной точки (два числа от 0..N-1);
4. координаты конечной точки точки (два числа от 0..N-1).

Формат выходного файла:

1. 1 — задача решена, 0 — задача не имеет решения, -1 — неправильные параметры во входном файле
2. если задача решена – список пар <цифра><направление>. Каждая пара начинается с новой строки. <Цифра> — число от 1 до 6, <направление> — число от 1 до 8.
3. если неправильные параметры во входном файле — код ошибки: 1 — ошибка при открытии файла, 2 — неправильный размер поля, 3 — неправильная цифра в поле, 4 — неправильные координаты начальной точки, 5 — неправильные координаты конечной точки.

Числа, соответствующие направлениям, приведены в следующей таблице:

Число	**Направление**
1	вверх
2	вправо, вверх
3	вправо
4	вправо, вниз
5	вниз
6	влево, вниз
7	влево
8	влево, вверх

5. Инструкцию по использованию тестирующего ПО

Для создания юнит-тестов была использована библиотека GoogleTest для C++.В папке googletest-master\googletest\msvc находится проект gtest-md.sln. При его компиляции создаются библиотеки gtest.lib (Release) и gtestd.lib (Debug). Необходимые заголовочные файлы расположены в каталоге googletest-master\googletest\include\gtest.

Заголовочные файлы и скомпилированные библиотеки добавлены в каталог gtest решения Maze в подкаталоги include и lib соответственно. Для выполнения юнит-тестов в решения добавлен проект UnitTests. После компиляции создается исполняемый файл UnitTests.exe.

Исполняемый файл выполняет запуск тестов и вывод результатов выполнения.

6. Описание тестов

6.1. Тесты ввода-вывода

Список тестов:

1. Корректные данные, размер поля 10, входной файл test1_1_i.txt, выходной файл test1_1_o.txt.
   Ожидаемый результат: в первой строке файла код 0 или 1. Если 1, то дальше следуют пары <цифра><направление>.
2. Корректные данные, размер поля 5, входной файл test1_2_i.txt, выходной файл test1_2_o.txt.
   Ожидаемый результат: в первой строке файла код 0 или 1. Если 1, то дальше следуют пары <цифра><направление>.
3. Входной файл отсутствует. Входной файл test1_3_i.txt, выходной файл test1_3_o.txt.
   Ожидаемый результат: в первой строке файла код -1, во второй строке код ошибки 1.
4. Неправильный размер поля. Входной файл test1_4_i.txt, выходной файл test1_4_o.txt.
   Ожидаемый результат: в первой строке файла код -1, во второй строке код ошибки 2.
5. Неправильное значение в поле. Входной файл test1_5_i.txt, выходной файл test1_5_o.txt.
   Ожидаемый результат: в первой строке файла код -1, во второй строке код ошибки 3.
6. Неправильная координата начальной точки. Входной файл test1_6_i.txt, выходной файл test1_6_o.txt.

Ожидаемый результат: в первой строке файла код -1, во второй строке код ошибки 4.

7. Неправильная координата конечной точки. Входной файл test1_7_i.txt, выходной файл test1_7_o.txt.
   Ожидаемый результат: в первой строке файла код -1, во второй строке код ошибки 5.

6.2 Тесты решения

Список тестов:

1. Размер поля 10, решение существует, входной файл test2_1_i.txt, выходной файл test2_1_o.txt.
2. Размер поля 30, решение существует, входной файл test2_2_i.txt, выходной файл test2_2_o.txt.
3. Размер поля 4, решение существует, входной файл test2_3_i.txt, выходной файл test2_3_o.txt.
4. Размер поля 1, решение существует, входной файл test2_4_i.txt, выходной файл test2_4_o.txt.
5. Размер поля 10, решения не существует, входной файл test2_5_i.txt, выходной файл test2_5_o.txt.
6. Размер поля 30, решения не существует, входной файл test2_6_i.txt, выходной файл test2_6_o.txt.
7. Размер поля 4, решения не существует, входной файл test2_7_i.txt, выходной файл test2_7_o.txt.

Если решение существует, то в первой строке ожидается код 1, далее в шести строках пары <цифра><направление>.

Если решения не существует, то в первой строке ожидается код 0.

6.3 Тесты корректности алгоритма

Список юнит-тестов:

1. Тест функции *load_data*. Загружаются данные из файла test2_1_i.txt и контролируются данные, записанные в структуру Data: размер, цифры поля, координаты начальной и конечной точки.
2. Тест функции *best_dir*. Функция анализирует разность координат начальной и конечной точки. В тесте передаются координаты, для каждого направления, запрашивается первое (самое приоритетное) направление. Для двух вариантов координат запрашиваются все 8 вариантов направлений. Контролируется возвращаемое значение.

3. Тест функции *solve*. Функции передаются данные поля, в котором можно найти путь, и данные поля, в котором невозможно найти путь. Контролируется результат поиска решения.

7. Оценка сложности алгоритма программы

Основой алгоритма является поиск в глубину, оценим его сложность.

Количество цифр 6, каждой может соответствовать одно направление, разным цифрам не может соответствовать одно и то же направление.

Число размещений из 8 по 6, т.е. число вариантов распределения направлений по цифрам равно 20160. Максимальная длина пути для каждого из вариантов N*N-1 , где N — размер поля. Поэтому количество шагов рекурсивной функции не превысит $20160*(N^2-1)$. Таким образом сложность алгоритма оценивается как $O(N^2)$.

8. Оценка использования памяти алгоритмом

Алгоритм использует матрицу цифр поля (NxN значений), матрицу для отметки посещенных клеток поля (NxN значений). Рекурсивная функция, реализующая алгоритм вызывается на глубину N^2.

Таким образом объем памяти, используемый алгоритмом, кратен N^2.

Приложение 1
Листинг программы

Листинг модуля Maze.cpp:

```
include <iostream>
#include <fstream>

using namespace std;

#include "Solve.h"

int main(int argc, char* argv[]) {
    if (argc < 3) {
        cout << "Usage: maze.exe <input file> <output file>" << endl;
        return 1;
    }

    char * infile = argv[1];  //имя входного файла
    char * outfile = argv[2];  //имя выходного файла

    return solve_task(infile, outfile);
}
```

Листинг модуля Solve.h:

```
#ifndef MAZE_H
#define MAZE_H

//данные для решения задачи
struct Data {
    int N; //размер поля
    int **matr; //поле
    int ni, nj, ki, kj; //начальная и конечная точки

    int **check; //отметка пройденных клеток
    int dir[6]; //направление для каждой цифры (1..6)
    int digit[8]; //цифра для каждого направления (1..8)
};

void new_matr(int ** &m, int n);
void free_matr(int ** &m, int n);
int load_data(char * infile, Data *data);
int best_dir(int i1, int j1, int i2, int j2, int k);
int solve(int pi, int pj, Data *data);
int solve_task(char * infile, char * outfile);

#endif
```

Листин гмодуля Solve.cpp:

```
#include <iostream>
#include <fstream>

using namespace std;

#include "Solve.h"

//выделение памяти для матрицы
void new_matr(int ** &m, int n) {
    m = new int*[n];
```

```
    for (int i = 0; i < n; i++) {
        m[i] = new int[n];
        memset(m[i], 0, sizeof(int)*n);
    }
}

//освобождение памяти выделенной матрице
void free_matr(int ** &m, int n) {
    if (m != NULL) {
        for (int i = 0; i < n; i++)
            if (m[i] != NULL) delete[] m[i];
        delete[] m;
    }
    m = NULL;
}


//определение k-го по значимости направления от клетки i1,j1 к клетке i2,j2
int best_dir(int i1, int j1, int i2, int j2, int k) {
    int di = i2 - i1;
    int dj = j2 - j1;
    int best;

    //выбор лучшего направления
    if (di == 0) {
        if (dj > 0) best = 3; else best = 7;
    } else
        if (dj == 0) {
            if (di > 0) best = 5; else best = 1;
        } else
            if (di < 0) {
                if (dj < 0) best = 8; else best = 2;
            } else {
                if (dj < 0) best = 6; else best = 4;
            }

    //первое направление
    if (k == 0) return best;

    if (k & 1) { //k - нечетное
        best += (k + 1) / 2;
        if (best > 8) best -= 8;
    } else { //k - четное
        best -= (k + 1) / 2;
        if (best < 1) best += 8;
    }
    return best;
}

//загрузка входных данных
int load_data(char * infile, Data *data) {
    ifstream fin(infile);
    if ((fin.rdstate() & std::ifstream::failbit) != 0) return 1;
    fin >> data->N;
    if ((fin.rdstate() & std::ifstream::failbit) != 0 || data->N<1) return 2;

    //выделение памяти для поля N на N клеток
    new_matr(data->matr,data->N);

    //чтение значений поля
    for (int i = 0; i<data->N; i++)
        for (int j = 0; j < data->N; j++) {
            fin >> data->matr[i][j];
            if ((fin.rdstate() & std::ifstream::failbit) != 0
                || data->matr[i][j]<0 || data->matr[i][j]>6) return 3;
```

```
        }

    //чтение начальной и конечной точки
    fin >> data->ni >> data->nj;
    if ((fin.rdstate() & std::ifstream::failbit) != 0 || data->ni<0 || data->nj<0
        || data->ni >= data->N || data->nj >= data->N || data->matr[data->ni][data-
>nj]==0) return 4;
    fin >> data->ki >> data->kj;
    if ((fin.rdstate() & std::ifstream::failbit) != 0  || data->ki<0 || data->kj<0
        || data->ki >= data->N || data->kj >= data->N || data->matr[data->ki][data->kj]
== 0) return 5;

    fin.close();
    return 0;
}

int solve(int pi, int pj, Data *data) {
    if (pi == data->ki && pj == data->kj)
        return 1; //попали в конечную клетку
    int v = data->matr[pi][pj]; //значение в текущей клетке
    int gi, gj;
    if (data->dir[v - 1] > 0) {  //значению сопоставлено направление
        //кооридната следующей клетки
        gi = pi;
        gj = pj;
        switch (data->dir[v - 1]) {
            case 1: gi--;       break;
            case 2: gi--; gj++; break;
            case 3:       gj++; break;
            case 4: gi++; gj++; break;
            case 5: gi++;       break;
            case 6: gi++; gj--; break;
            case 7:       gj--; break;
            case 8: gi--; gj--; break;
        }
        //невозможно продвинуться по данному направлению - выход
        if (gi < 0 || gj < 0 || gi >= data->N || gj >= data->N
            || data->matr[gi][gj] == 0 || data->check[gi][gj] != 0) return 0;
        data->check[gi][gj] = 1;
        //продвинуться в заданном направлении
        if (solve(gi, gj, data)) return 1;
        data->check[gi][gj] = 0;
        return 0;
    } else {
        //8 возможных направлений
        for (int i = 0; i < 8; i++) {
            int bdir = best_dir(pi, pj, data->kj, data->kj, i);
            if (data->digit[bdir - 1] > 0) continue; //направление уже назначено другой
цифре
            //кооридната следующей клетки
            gi = pi;
            gj = pj;
            switch (bdir) {
                case 1: gi--;       break;
                case 2: gi--; gj++; break;
                case 3:       gj++; break;
                case 4: gi++; gj++; break;
                case 5: gi++;       break;
                case 6: gi++; gj--; break;
                case 7:       gj--; break;
                case 8: gi--; gj--; break;
            }
            //невозможно продвинуться по данному направлению - к следующей итерации
            if (gi < 0 || gj < 0 || gi >= data->N || gj >= data->N
                || data->matr[gi][gj] == 0 || data->check[gi][gj] != 0) continue;
```

```
            data->dir[v - 1] = bdir;
            data->digit[bdir - 1] = v;
            data->check[gi][gj] = 1;
            //продвинуться в заданном направлении
            if (solve(gi, gj, data)) return 1; //найдено решение
            data->dir[v - 1] = 0;
            data->digit[bdir - 1] = 0;
            data->check[gi][gj] = 0;
        }
        return 0; //решение не найдено
    }
}

int solve_task(char * infile, char * outfile) {
    Data data;

    //загрузить данные из файла
    int res = load_data(infile,&data);

    ofstream fout(outfile);

    if (res > 0) {
        fout << "-1" << endl;
        fout << res << endl;
        return 1;
    }

    //для всех цифр направления не заданы
    for (int i = 0; i < 6; i++) data.dir[i] = 0;
    //для всех направлений цифры не заданы
    for (int i = 0; i < 8; i++) data.digit[i] = 0;

    //выделение памяти для отметки клеток (массив N на N)
    new_matr(data.check, data.N);

    if (solve(data.ni, data.nj, &data)) {

        //задать неуиспользванным цифрам свободные направления
        for (int i = 0; i < 6; i++) {
            if (data.dir[i] == 0) {
                for (int j = 0; j<8; j++)
                    if (data.digit[j] == 0) {
                        data.dir[i] = j + 1;
                        data.digit[j] = i + 1;
                        break;
                    }
            }
        }

        fout << "1" << endl;
        for (int i = 0; i<6; i++)
            fout << i + 1 << " " << data.dir[i] << endl;
    } else {
        fout << "0" << endl;
    }
    fout.close();

    //освобождение памяти
    free_matr(data.matr, data.N);
    free_matr(data.check, data.N);
    return 0;
}
```

Листинг модуля UnitTests.cpp:

```
#include <gtest/gtest.h>
#include "../Maze/Solve.h"

class CTest_load_data : public ::testing::Test {
};

class CTest_best_dir : public ::testing::Test {
};

class CTest_solve : public ::testing::Test {
};

//тест функции load_data
TEST_F(CTest_load_data, CheckPerimeter) {
    Data data;
    //вызов функции
    ASSERT_EQ(0, load_data("test3_1_i.txt", &data));

    //проверка прочитанных из файла данных
    ASSERT_EQ(5, data.N);

    ASSERT_EQ(1, data.matr[0][0]);
    ASSERT_EQ(3, data.matr[0][1]);
    ASSERT_EQ(0, data.matr[0][2]);
    ASSERT_EQ(3, data.matr[0][3]);
    ASSERT_EQ(1, data.matr[0][4]);

    ASSERT_EQ(0, data.matr[1][0]);
    ASSERT_EQ(5, data.matr[1][1]);
    ASSERT_EQ(6, data.matr[1][2]);
    ASSERT_EQ(3, data.matr[1][3]);
    ASSERT_EQ(3, data.matr[1][4]);

    ASSERT_EQ(6, data.matr[2][0]);
    ASSERT_EQ(2, data.matr[2][1]);
    ASSERT_EQ(2, data.matr[2][2]);
    ASSERT_EQ(4, data.matr[2][3]);
    ASSERT_EQ(0, data.matr[2][4]);

    ASSERT_EQ(0, data.matr[3][0]);
    ASSERT_EQ(5, data.matr[3][1]);
    ASSERT_EQ(6, data.matr[3][2]);
    ASSERT_EQ(2, data.matr[3][3]);
    ASSERT_EQ(6, data.matr[3][4]);

    ASSERT_EQ(0, data.matr[4][0]);
    ASSERT_EQ(1, data.matr[4][1]);
    ASSERT_EQ(5, data.matr[4][2]);
    ASSERT_EQ(5, data.matr[4][3]);
    ASSERT_EQ(0, data.matr[4][4]);

    ASSERT_EQ(2, data.ni);
    ASSERT_EQ(2, data.nj);

    ASSERT_EQ(4, data.ki);
    ASSERT_EQ(1, data.kj);

    free_matr(data.matr, data.N);
}

//тест функции best_dir
TEST_F(CTest_best_dir, CheckPerimeter) {
    //первый вариант направления
```

```
    ASSERT_EQ(1, best_dir(6, 3, 4, 3, 0));
    ASSERT_EQ(3, best_dir(8, 3, 8, 5, 0));
    ASSERT_EQ(4, best_dir(2, 4, 8, 5, 0));
    ASSERT_EQ(5, best_dir(3, 2, 4, 2, 0));
    ASSERT_EQ(6, best_dir(10, 30, 30, 20, 0));
    ASSERT_EQ(7, best_dir(4, 8, 4, 5, 0));
    ASSERT_EQ(8, best_dir(8, 3, 1, 0, 0));

    //все варианты направлений
    ASSERT_EQ(2, best_dir(8, 3, 4, 5, 0));
    ASSERT_EQ(3, best_dir(8, 3, 4, 5, 1));
    ASSERT_EQ(1, best_dir(8, 3, 4, 5, 2));
    ASSERT_EQ(4, best_dir(8, 3, 4, 5, 3));
    ASSERT_EQ(8, best_dir(8, 3, 4, 5, 4));
    ASSERT_EQ(5, best_dir(8, 3, 4, 5, 5));
    ASSERT_EQ(7, best_dir(8, 3, 4, 5, 6));
    ASSERT_EQ(6, best_dir(8, 3, 4, 5, 7));

    //все варианты направлений
    ASSERT_EQ(5, best_dir(3, 2, 4, 2, 0));
    ASSERT_EQ(6, best_dir(3, 2, 4, 2, 1));
    ASSERT_EQ(4, best_dir(3, 2, 4, 2, 2));
    ASSERT_EQ(7, best_dir(3, 2, 4, 2, 3));
    ASSERT_EQ(3, best_dir(3, 2, 4, 2, 4));
    ASSERT_EQ(8, best_dir(3, 2, 4, 2, 5));
    ASSERT_EQ(2, best_dir(3, 2, 4, 2, 6));
    ASSERT_EQ(1, best_dir(3, 2, 4, 2, 7));
}

//тест функции solve
TEST_F(CTest_solve, CheckPerimeter) {
    Data data;

    //путь существует

    data.N = 4;
    new_matr(data.matr, data.N);
    data.matr[0][0] = 3; data.matr[0][1] = 1; data.matr[0][2] = 3; data.matr[0][3] = 2;
    data.matr[1][0] = 4; data.matr[1][1] = 5; data.matr[1][2] = 5; data.matr[1][3] = 0;
    data.matr[2][0] = 3; data.matr[2][1] = 4; data.matr[2][2] = 1; data.matr[2][3] = 3;
    data.matr[3][0] = 1; data.matr[3][1] = 0; data.matr[3][2] = 2; data.matr[3][3] = 1;
    data.ki = 3;
    data.kj = 3;
    new_matr(data.check, data.N);
    memset(data.dir, 0, sizeof(data.dir));
    memset(data.digit, 0, sizeof(data.digit));

    //вызов функции
    ASSERT_EQ(1, solve(0,0,&data));

    free_matr(data.matr, data.N);
    free_matr(data.check, data.N);

    //путь не существует

    data.N = 4;
    new_matr(data.matr, data.N);
    data.matr[0][0] = 3; data.matr[0][1] = 1; data.matr[0][2] = 3; data.matr[0][3] = 2;
    data.matr[1][0] = 4; data.matr[1][1] = 5; data.matr[1][2] = 5; data.matr[1][3] = 0;
    data.matr[2][0] = 3; data.matr[2][1] = 4; data.matr[2][2] = 0; data.matr[2][3] = 0;
    data.matr[3][0] = 1; data.matr[3][1] = 0; data.matr[3][2] = 0; data.matr[3][3] = 1;
    data.ki = 3;
    data.kj = 3;
    new_matr(data.check, data.N);
    memset(data.dir, 0, sizeof(data.dir));
```

```
    memset(data.digit, 0, sizeof(data.digit));

    //вызов функции
    ASSERT_EQ(0, solve(0, 0, &data));

    free_matr(data.matr, data.N);
    free_matr(data.check, data.N);
}


int main(int argc, char **argv) {
    ::testing::InitGoogleTest(&argc, argv);
    int res = RUN_ALL_TESTS();
    system("pause");
    return res;
}
```