МИНОБРНАУКИ РОССИИ

ФГБОУ ВО «ИжГТУ им. М.Т. Калашникова»

Кафедра «Программное обеспечение»

Отчет

по лабораторной работе №3

по дисциплине «Операционные системы»

Вариант №4

Выполнил:

Студент группы Б18-191-2 Р. А. Гумметов

Принял:

**к.т.н., профессор**  В. Г. Тарасов

Ижевск 2021

ЦЕЛЬ РАБОТЫ

Разработать программу, ищущую кратчайший путь и визуализирующую данный путь на языке c++в программе Qt Creator 4 на операционной системе OS Free BSD.

ПОСТАНОВКА ЗАДАЧИ И ИНСТРУМЕНТЫ ДЛЯ ЕЕ РЕШЕНИЯ

Разработать программу, ищущую кратчайший путь и визуализирующую данный путь.

Расположить на форме кнопки, позволяющие пользователю взаимодействовать с программой, а также само поле.

Объект начинает свое движение из левого верхнего угла поля. Передвижение возможно только буквой "Г". На каждом шаге идет подсчет, какой номер хода происходит в данный момент. Когда объект достигает правый нижний угол, программа выводит количество сделанных шагов. Появляется кнопка "Начать сначала", нажав которую, программа возвращается к началу.

АЛГОРИТМ ПОИСКА КРАТЧАЙШЕГО ПУТИ И ЕГО ПОСТРОЕНИЯ

В программе для поиска кратчайшего пути используется волновой алгоритм. 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. 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.

РАБОТА ПРОГРАММЫ

Вид формы при запуске программы:

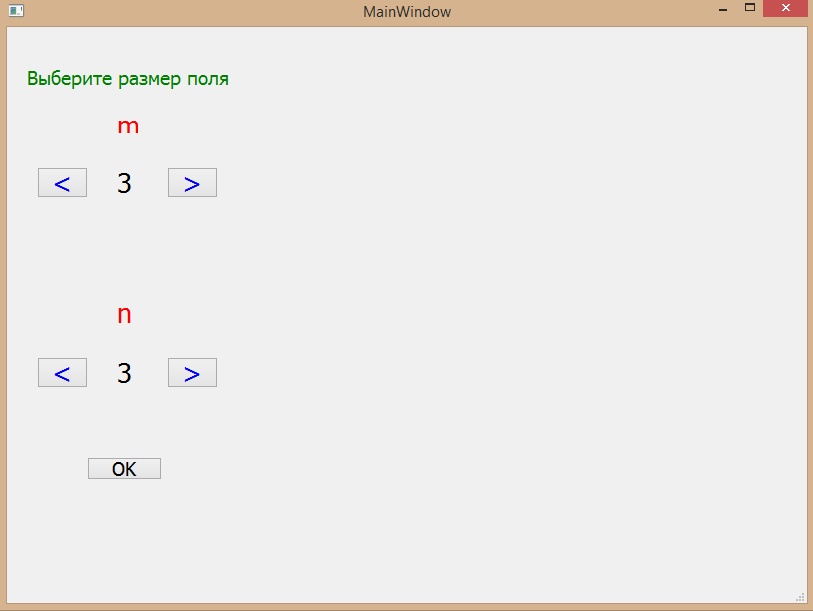


Рис. 1 - Форма при запуске программы

После выбора размеров поля, и нажатии кнопки "ОК" под двумерный массив field выделяется память в соответствии с выбранными размерами поля, стартовая точка добавляется в очередь, программа вызывает функцию testP поиска кратчайшего пути волновым алгоритмом, после этого вызывается функция find\_only\_way восстановления пути в обратном направлении. В массив way записываются индексы клеток, по которым построен путь.

Кнопки для изменения размеров поля блокируются, появляется кнопка "Начать", которая в этот момент единственная доступная для нажатия, а также появляется само поле для игры:

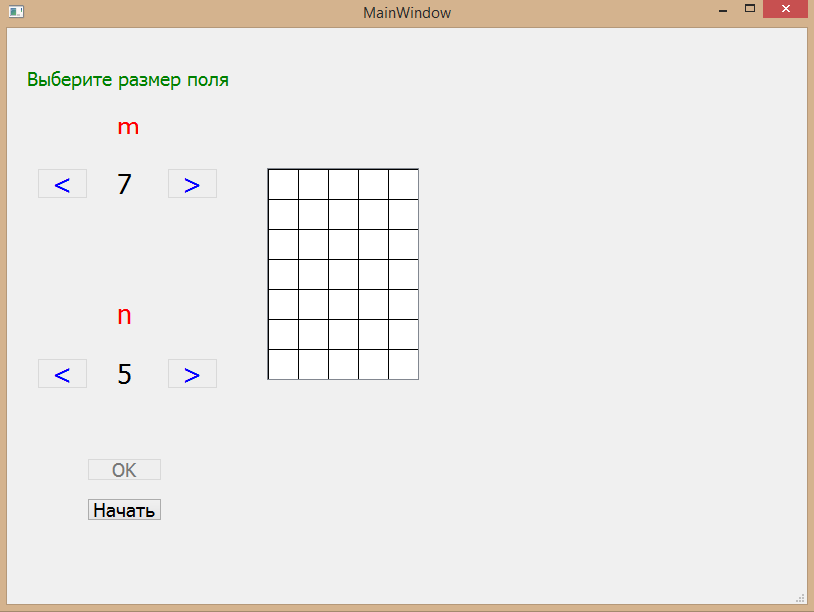


Рис. 2 - Появление кнопки "Начать" и самого поля

После нажатия на кнопку "Начать" запускается таймер с интервалом 1с. Каждую секунду на поле закрашивается одна клетка - место, в которое сделан очередной шаг. Справа от поля выводится номер сделанного хода:

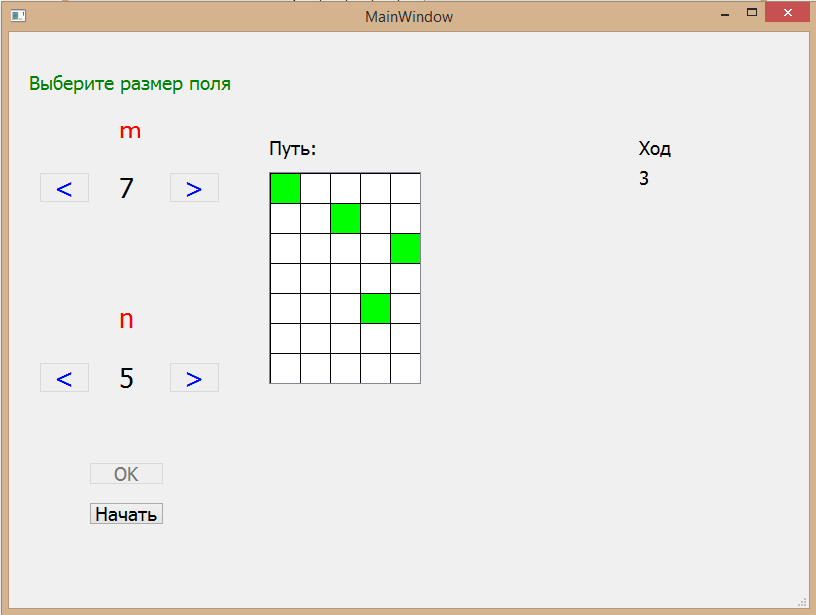


Рис. 3 - Визуализация пути

В момент, когда закрасится клетка в правом нижнем углу, таймер останавливается, снизу от поля выводится число, показывающее общее количество сделанных шагов и появляется кнопка "Начать сначала":

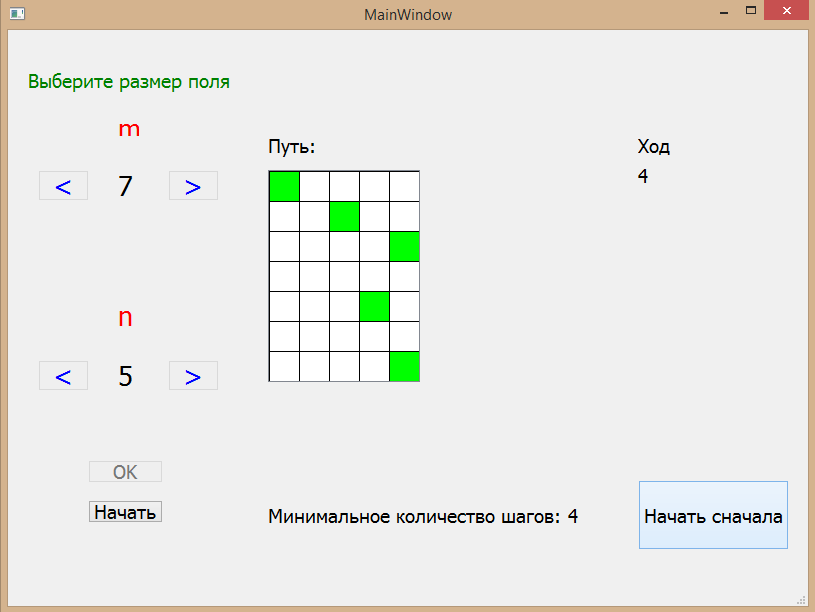


Рис. 4 - Конец пути

После нажатия кнопки "Начать сначала" кнопки для выбора размера поля снова становятся доступны, поле стирается, очищаются массивы для поля, найденного пути. Все оставшиеся в очереди элементы из очереди извлекаются. Переменные принимают свои начальные значения.

После этого форма приобретает начальный вид, программу можно начать сначала:

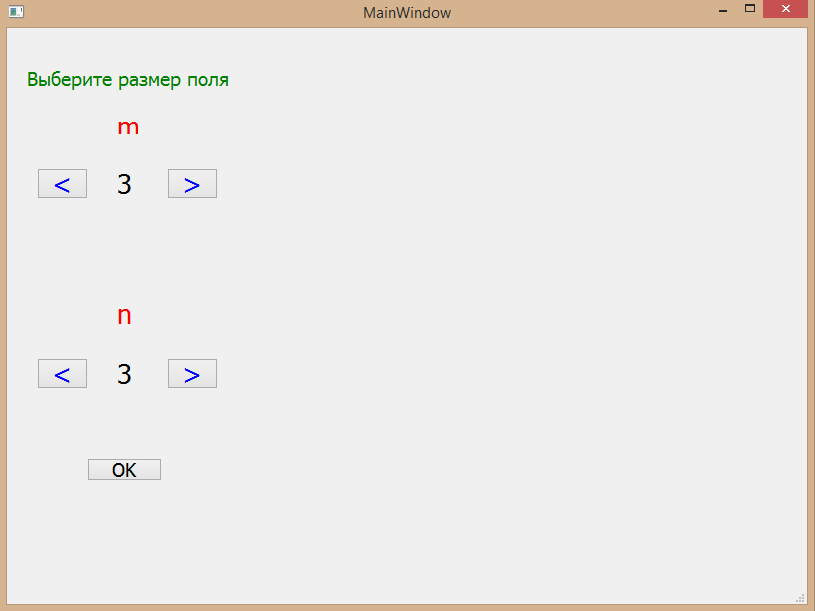


Рис. 5 - Начать сначала

ВЫВОД

В ходе лабораторной работы разработана программа, ищущая кратчайший путь и ее графическая визуализация.

Приложение реализовано на операционной системе Free BSD в программе Qt Creator на языке c++.

ТЕКСТ ПРОГРАММЫ

файл mainwindow.h:

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QTimer>

#include <QTime>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

**MainWindow**(QWidget \*parent = nullptr);

~***MainWindow***();

private:

Ui::MainWindow \*ui;

QTimer \*tmr;

private slots:

void **updateTime**();

private slots:

void **m\_reduce**();

private slots:

void **m\_increase**();

private slots:

void **n\_reduce**();

private slots:

void **n\_increase**();

private slots:

void **ok**();

private slots:

void **begin**();

private slots:

void **restart**();

private:

void **testP**(int i, int j, int count, int \*field[]);

private:

void **find\_only\_way**(int i, int j, int search, int \*field[], int \*way[], int current[]);

};

#endif // MAINWINDOW\_H

файл mainwindow.cpp:

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QLabel>

#include <string>

#include <queue>

#include <QTimer>

#include <QTime>

using namespace std;

int m = 3, n = 3;

int side\_square = 30;

queue <vector<int>> numbers;

int C; //количество ходов

int way[2][10];

int second = 0;

QBrush br(Qt::green);

QPen pen(Qt::black);

//QTimer \*tmr;

QGraphicsScene \*PalletScene;

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(*parent*)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

ui->pb\_begin->*setVisible*(false);

ui->pb\_restart->*setVisible*(false);

ui->label\_counter->*setVisible*(false);

ui->label\_way->*setVisible*(false);

ui->label\_motion->*setVisible*(false);

ui->label\_min\_motion->*setVisible*(false);

ui->label\_mincount->*setVisible*(false);

ui->graphicsView->*setVisible*(false);

connect(ui->pb\_mleft, SIGNAL(clicked()), this, SLOT(m\_reduce()));

connect(ui->pb\_mright, SIGNAL(clicked()), this, SLOT(m\_increase()));

connect(ui->pb\_nleft, SIGNAL(clicked()), this, SLOT(n\_reduce()));

connect(ui->pb\_nright, SIGNAL(clicked()), this, SLOT(n\_increase()));

connect(ui->pb\_ok, SIGNAL(clicked()), this, SLOT(ok()));

connect(ui->pb\_begin, SIGNAL(clicked()), this, SLOT(begin()));

connect(ui->pb\_restart, SIGNAL(clicked()), this, SLOT(restart()));

}

MainWindow::~***MainWindow***()

{

delete ui;

}

void MainWindow::**m\_reduce**() {

QString label;

if (m > 3)

{

m--;

label = QString::number(m);

ui->label\_mdigit->setText(label);

}

}

void MainWindow::**m\_increase**() {

QString label;

if (m < 10)

{

m++;

label = QString::number(m);

ui->label\_mdigit->setText(label);

}

}

void MainWindow::**n\_reduce**() {

QString label;

if (n > 3)

{

n--;

label = QString::number(n);

ui->label\_ndigit->setText(label);

}

}

void MainWindow::**n\_increase**() {

QString label;

if (n < 10)

{

n++;

label = QString::number(n);

ui->label\_ndigit->setText(label);

}

}

void MainWindow::**ok**() {

ui->pb\_mleft->setEnabled(false);

ui->pb\_mright->setEnabled(false);

ui->pb\_nleft->setEnabled(false);

ui->pb\_nright->setEnabled(false);

ui->pb\_ok->setEnabled(false);

ui->pb\_begin->*setVisible*(true);

ui->graphicsView->*setVisible*(true);

ui->graphicsView->resize(n \* side\_square + 2, m \* side\_square + 2);

PalletScene = new QGraphicsScene(0, 0, n \* side\_square, m \* side\_square, this);

ui->graphicsView->setScene(*PalletScene*);

for (int i = 0; i <= n \* side\_square + 2; i += side\_square)

{

PalletScene->addLine(i, 0, i, m \* side\_square + 2);

}

for (int i = 0; i <= m \* side\_square + 2; i += side\_square)

{

PalletScene->addLine(0, i, n \* side\_square + 2, i);

}

int field[m][n];

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

{

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

{

field[i][j] = 0; //заполняем поле нулями

}

}

field[0][0] = -1; //стартовая точка = -1

vector<int> mas = { 0, 0 }; //массив в индексами стартовой точки

numbers.push(mas); //добавление стартовой точки в очередь

int \*p[m];

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

p[i] = field[i];

while (field[m - 1][n - 1] == 0)

{

vector<int> point = numbers.front(); //извлекли из очереди точку

numbers.pop();

if (point[0] == 0 && point[1] == 0)

testP(point[0], point[1], 1, p);

else

testP(point[0], point[1], field[point[0]][point[1]] + 1, p);

}

int count = field[m - 1][n - 1]; //кол-во ходов сделано

C = count;

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

{

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

{

if (field[i][j] == 0)

field[i][j] = -1;

}

}

field[0][0] = 0;

int \*pWay[2];

for (int i = 0; i < 2; i++) {

pWay[i] = way[i];

}

way[0][count] = m - 1;

way[1][count] = n - 1;

int current[2];

current[0] = m - 1;

current[1] = n - 1;

while (count != 0)

{

find\_only\_way(current[0], current[1], count - 1, p, pWay, current);

count--;

}

}

void MainWindow::**begin**() {

ui->label\_motion->*setVisible*(true);

ui->label\_counter->*setVisible*(true);

ui->label\_way->*setVisible*(true);

tmr = new QTimer();

connect(tmr, SIGNAL(timeout()), this, SLOT(updateTime()));

tmr->setInterval(1000);

tmr->start();

}

void MainWindow::**updateTime**()

{

if (second > C)

{

ui->label\_min\_motion->*setVisible*(true);

ui->label\_mincount->*setVisible*(true);

ui->label\_mincount->setText(QString::number(C));

ui->pb\_restart->*setVisible*(true);

tmr->stop();

}

else

{

PalletScene->addRect(way[1][second] \* 30, way[0][second] \* 30, 30, 30, pen, br);

ui->label\_counter->setText(QString::number(second));

second++;

}

}

void MainWindow::**restart**() {

ui->pb\_mleft->setEnabled(true);

ui->pb\_mright->setEnabled(true);

ui->pb\_nleft->setEnabled(true);

ui->pb\_nright->setEnabled(true);

ui->pb\_ok->setEnabled(true);

ui->pb\_begin->*setVisible*(false);

ui->graphicsView->*setVisible*(false);

ui->label\_motion->*setVisible*(false);

ui->label\_min\_motion->*setVisible*(false);

ui->pb\_restart->*setVisible*(false);

ui->label\_way->*setVisible*(false);

second = 0;

m = 3;

n = 3;

C = 0;

ui->graphicsView->setEnabled(false);

ui->label\_mdigit->setText("" + QString::number(m));

ui->label\_ndigit->setText("" + QString::number(n));

ui->label\_counter->setText("");

ui->label\_mincount->setText("");

while(numbers.size() > 0) {

numbers.pop();

}

}

void MainWindow::**testP**(int i, int j, int count, int \*field[]) {

if ((i - 2) >= 0) //на 2 вверх

{

if ((j - 1) >= 0) //на 1 влево

{

if (field[i - 2][j - 1] == 0)

{

field[i - 2][j - 1] = count;

vector<int> mas = { i - 2, j - 1 };

numbers.push(mas);

}

}

if ((j + 1) < n) //на 1 вправо

{

if (field[i - 2][j + 1] == 0)

{

field[i - 2][j + 1] = count;

vector<int> mas = { i - 2, j + 1 };

numbers.push(mas);

}

}

}

if ((i + 2) < m) //на 2 вниз

{

if ((j - 1) >= 0) //на 1 влево

{

if (field[i + 2][j - 1] == 0)

{

field[i + 2][j - 1] = count;

vector<int> mas = { i + 2, j - 1 };

numbers.push(mas);

}

}

if ((j + 1) < n) //на 1 вправо

{

if (field[i + 2][j + 1] == 0)

{

field[i + 2][j + 1] = count;

vector<int> mas = { i + 2, j + 1 };

numbers.push(mas);

}

}

}

if ((j - 2) >= 0) //на 2 влево

{

if ((i - 1) >= 0) //на 1 вверх

{

if (field[i - 1][j - 2] == 0)

{

field[i - 1][j - 2] = count;

vector<int> mas = { i - 1, j - 2 };

numbers.push(mas);

}

}

if ((i + 1) < m) //на 1 вниз

{

if (field[i + 1][j - 2] == 0)

{

field[i + 1][j - 2] = count;

vector<int> mas = { i + 1, j - 2 };

numbers.push(mas);

}

}

}

if ((j + 2) < n) //на 2 вправо

{

if ((i - 1) >= 0) //на 1 вверх

{

if (field[i - 1][j + 2] == 0)

{

field[i - 1][j + 2] = count;

vector<int> mas = { i - 1, j + 2 };

numbers.push(mas);

}

}

if ((i + 1) < m) //на 1 вниз

{

if (field[i + 1][j + 2] == 0)

{

field[i + 1][j + 2] = count;

vector<int> mas = { i + 1, j + 2 };

numbers.push(mas);

}

}

}

}

void MainWindow::**find\_only\_way**(int i, int j, int search, int \*field[], int \*way[], int current[]) {

bool f = false;

while (f == false)

{

if ((i - 2) >= 0) //на 2 вверх

{

if ((j - 1) >= 0) //на 1 влево

{

if (field[i - 2][j - 1] == search)

{

current[0] = i - 2;

current[1] = j - 1;

way[0][search] = i - 2;

way[1][search] = j - 1;

f = true;

break;

}

}

if ((j + 1) < n) //на 1 вправо

{

if (field[i - 2][j + 1] == search)

{

current[0] = i - 2;

current[1] = j + 1;

way[0][search] = i - 2;

way[1][search] = j + 1;

f = true;

break;

}

}

}

if ((i + 2) < m) //на 2 вниз

{

if ((j - 1) >= 0) //на 1 влево

{

if (field[i + 2][j - 1] == search)

{

current[0] = i + 2;

current[1] = j - 1;

way[0][search] = i + 2;

way[1][search] = j - 1;

f = true;

break;

}

}

if ((j + 1) < n) //на 1 вправо

{

if (field[i + 2][j + 1] == search)

{

current[0] = i + 2;

current[1] = j + 1;

way[0][search] = i + 2;

way[1][search] = j + 1;

f = true;

break;

}

}

}

if ((j - 2) >= 0) //на 2 влево

{

if ((i - 1) >= 0) //на 1 вверх

{

if (field[i - 1][j - 2] == search)

{

current[0] = i - 1;

current[1] = j - 2;

way[0][search] = i - 1;

way[1][search] = j - 2;

f = true;

break;

}

}

if ((i + 1) < m) //на 1 вниз

{

if (field[i + 1][j - 2] == search)

{

current[0] = i + 1;

current[1] = j - 2;

way[0][search] = i + 1;

way[1][search] = j - 2;

f = true;

break;

}

}

}

if ((j + 2) < n) //на 2 вправо

{

if ((i - 1) >= 0) //на 1 вверх

{

if (field[i - 1][j + 2] == search)

{

current[0] = i - 1;

current[1] = j + 2;

way[0][search] = i - 1;

way[1][search] = j + 2;

f = true;

break;

}

}

if ((i + 1) < m) //на 1 вниз

{

if (field[i + 1][j + 2] == search)

{

current[0] = i + 1;

current[1] = j + 2;

way[0][search] = i + 1;

way[1][search] = j + 2;

f = true;

break;

}

}

}

}

}