

ГУАП
КАФЕДРА № 52

ПРЕПОДАВАТЕЛЬ

Канд. Техн. наук		Е.М. Линский
должность , уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О КУРСОВОЙ РАБОТЕ
Выход из лабиринта
СОЗДАНИЕ ПРОГРАММЫ НА ЯЗЫКЕ C/C++

по курсу: ОСНОВЫ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	5023		Грицацев И.А.
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2021


Оглавление


Постановка задачи	3
Алгоритм	4
Псевдокод алгоритма	7
Инструкция пользователя	19
Текстовые примеры	20
Список источников	23

Постановка задачи


Задачей данной курсовой работы является разработка программы, которая позволяет находить путь для выхода из трёхмерного лабиринта на основе волнового алгоритма. Можно совершать ходы в 8 направлениях : вверх на одну клетку, вправо на одну клетку, вниз на одну клетку и влево на одну клетку, по диагонали вверх и вправо, по диагонали вверх и влево, по диагонали вниз и вправо, по диагонали вниз и влево.


Лабиринт задан матрицей:

START (1) – вход 

FINISH (2) – выход 

FREE (3) – свободная клетка 

UNFREE (4) – стена 

WAY (*) - путь 

- Задача имеет смысл: лабиринт может вообще не иметь стен(см.

Рисунок 1.), а может иметь и стены (см. Рисунок2.).



Рисунок 1. Лабиринт без стенок

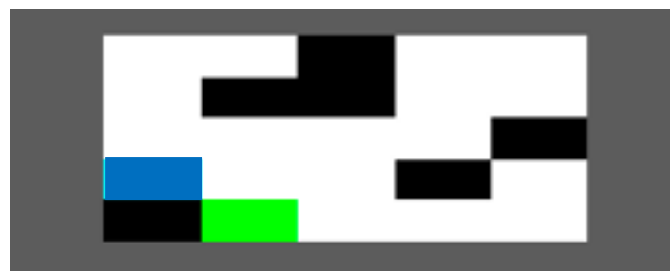


Рисунок 2. Лабиринт со стенами

Алгоритм

Алгоритм построен на идеях волнового алгоритма [1].

Структуры данных, которые использует алгоритм: `std::pair<unsigned, unsigned> point` , `std::pair< point, point > find_start_finish`, `std::pair<bool, point> res`, `std::vector<unsigned> wave` , `std::queue <point> _queue` и `std::vector <eCell> mMatrix`

Покажем работу алгоритма на примере простого лабиринта (см. Рисунок 3.).



Рисунок 3. Пример лабиринта

Сначала алгоритм смотрит, есть ли пустая (свободная) клетка (которые заданы в `std::vector <eCell> mMatrix (FREE)`) на одну клетку выше входа. Если есть, то заносит значение на 1 большее значения для входа (который задан 1 (START) в `std::vector <eCell> mMatrix`) в `std::vector<unsigned> wave` и добавляет координаты точки в очередь (изначально в `std::vector<unsigned> wave` хранился 0 для координаты входа и произведение количества столбцов и колонок для координат остальных клеток). Далее алгоритм смотрит, есть ли пустая клетка на 1 клетку выше и правее по диагонали от входа – если есть, то проставляет в `std::vector<unsigned> wave` для этой клетки значение на 1 большее значения для входа и добавляет в очередь точку. После он смотрит, есть ли пустая клетка на 1 клетку правее входа – если есть - проставляет для этой клетки значение на 1 большее значения входа в `std::vector<unsigned> wave` и добавляет точку в очередь . Затем алгоритм смотрит, есть ли пустая клетка на 1 клетку выше и левее по диагонали от входа – если есть, то проставляет в `std::vector<unsigned> wave` для этой клетки значение на 1

большее значения для входа и добавляет в очередь точку. После этого смотрит, есть ли пустая клетка на 1 клетку ниже входа – если есть, то проставляет в `std::vector<unsigned> wave` для этой клетки значение на 1 большее значения для входа и добавляет в очередь точку. Затем алгоритм смотрит, есть ли пустая клетка на 1 клетку ниже и правее по диагонали от входа – если есть, то проставляет в `std::vector<unsigned> wave` для этой клетки значение на 1 большее значения для входа и добавляет в очередь точку. Далее алгоритм смотрит, есть ли пустая клетка на 1 клетку левее от входа – если есть, то проставляет в `std::vector<unsigned> wave` для этой клетки значение на 1 большее значения для входа и добавляет в очередь точку. Затем алгоритм смотрит, есть ли пустая клетка на 1 клетку ниже и левее по диагонали от входа – если есть, то проставляет в `std::vector<unsigned> wave` для этой клетки значение на 1 большее значения для входа и добавляет в очередь точку. На следующей итерации он удаляет из очереди первую координату точки. Далее алгоритм продолжается от первой оставшейся в очереди точки и далее по порядку, пока очередь не пуста. После того, как алгоритм нашел выход (FINISH в `std::vector<eCell> mMatrix`) методом “волны”, ему нужно проставить путь (WAY) между входом (START) и выходом (FINISH) в `std::vector<eCell> mMatrix`. Начиная от выхода (FINISH), алгоритм проставляет путь (WAY) в `std::vector<eCell> mMatrix` в клетку значение которой в `std::vector<unsigned> wave` меньше на 1, чем в клетке (FINISH), а затем уже начиная от последней проставленной клетки (WAY) в `std::vector<eCell> mMatrix` продолжает проставлять путь (WAY) в клетку имеющую значение на 1 меньшее (в `std::vector<unsigned> wave`) пока не дойдет до входа (START) (значения 0 в `std::vector<unsigned> wave`)

Результат выполнения алгоритма представлен ниже (см. Рисунок 4.).



Рисунок 4. Результат выполнения алгоритма

Псевдокод алгоритма

```
std::vector<eCell> mMatrix;
```

```
std::pair<point,point> res
```

Считывание матрицы из файла в `std::vector<eCell> mMatrix ;`

нахождение координат входа (`mStart = res.first`) и

выхода (`mFinish = res.second`);

Заполнение всех клеток `std::vector<unsigned> wave`

(массив, выделенный под распространение волны) максимально

возможным значением волны, равным произведению

произведению количества строк на количество

столбцов (`mMaxCellCost`) ;

Занесение в `std::vector<unsigned> wave` стартового значения
`d= 0` по координатам входа в лабиринт ;

Метод "волны":

```
std::queue <point> _queue;
```

```
res = std::pair<bool,point>;
```

```
unsigned l, c ;//координаты по строке
```

и столбцу соответственно

```
way_is_found = False;//изначально
```

статус «Путь не найден»

```
queue.push(mStart.first,
```

```
mStart.second) ;// помещаем в очередь
```

координаты входа

```

while (queue.size>0) //пока очередь не
пуста
{

    tmp = queue.front //сохраняем
    первую из координату очереди во
    временную переменную

    d = wave[tmp.first, tmp.second] ;
    //присваиваем текущее значение волны
    в переменную

    queue.pop; //удаляем первый элемент
    в очереди

    if ( соседняя клетка сверху свободна и
    равна mMaxCellCost ) {

        wave[tmp.first-1, tmp.second] =
        d+1 ; // изначально d = 0

        queue.push({tmp.first -1,
        tmp.second}) ; // добавляем в
        очередь координаты этой точки
    }

    if ( соседняя клетка сверху является
    выходом)

    {

```



```

wave[tmp.first-1, tmp.second] =
d+1 ;

Way_is_found = TRUE; // ставим
статус «Путь найден»

}

```

```

if ( соседняя клетка справа свободна
и равна mMaxCellCost ) {

    wave[tmp.first, tmp.second+1] =
d+1 ; // изначально d = 0, помещаем
значения на 1 большее, чем в
предыдущей клетке

    queue.push({tmp.first ,
tmp.second+1}) ; // добавляем в
очередь координаты этой точки

}

if ( соседняя клетка справа является
выходом)

{

    wave[tmp.first, tmp.second+1] =
d+1 ;

    Way_is_found = TRUE; // ставим
статус «Путь найден»

}

```

```

if ( соседняя клетка слева свободна и
равна mMaxCellCost ) {

    wave[tmp.first, tmp.second -1] =
d+1 ; // изначально d = 0

queue.push({tmp.first , tmp.second
-1}) ; // добавляем в очередь
координаты этой точки

}

if ( соседняя клетка слева является
выходом)

{

wave[tmp.first, tmp.second -1] =
d+1 ;

Way_is_found = TRUE; // ставим
статус «Путь найден»

}

if ( соседняя клетка снизу свободна и
равна mMaxCellCost ) {

    wave[tmp.first+1, tmp.second] =
d+1 ; // изначально d = 0

queue.push({tmp.first +1,
tmp.second}) ; // добавляем в
очередь координаты этой точки

}

```

```

if ( соседняя клетка снизу является
выходом)

{

wave[tmp.first+1, tmp.second] =
d+1 ;

Way_is_found = TRUE; // ставим
статус «Путь найден»

}


if ( соседняя клетка по верхней левой
диагонали свободна и равна
mMaxCellCost ) {

    wave[tmp.first-1, tmp.second-1] =
d+1 ; // изначально d = 0

    queue.push({tmp.first -1,
tmp.second-1}) ; // добавляем в
очередь координаты этой точки

}


if ( соседняя клетка по верхней левой
диагонали является выходом)

{

wave[tmp.first-1, tmp.second - 1]
= d+1 ;

Way_is_found = TRUE; // ставим
статус «Путь найден»

```

```

}

if ( соседняя клетка по верхней правой
диагонали свободна и равна
mMaxCellCost ) {

    wave[tmp.first-1, tmp.second+1] =
d+1 ; // изначально d = 0

    queue.push({tmp.first -1,
tmp.second+1}) ; // добавляем в
очередь координаты этой точки

}

if ( соседняя клетка по верхней правой
диагонали является выходом)

{

wave[tmp.first-1, tmp.second + 1]
= d+1 ;

Way_is_found = TRUE; // ставим
статус «Путь найден»

}

if ( соседняя клетка по нижней правой
диагонали свободна и равна
mMaxCellCost ) {

    wave[tmp.first+1, tmp.second+1] =
d+1 ; // изначально d = 0

```

```

queue.push({tmp.first +1,
tmp.second-1}) ; // добавляем в
очередь координаты этой точки

}

if ( соседняя клетка по нижней правой
диагонали является выходом)

{

wave[tmp.first+1, tmp.second +1] =
d+1 ;

Way_is_found = TRUE; // ставим
статус «Путь найден»

}

if ( соседняя клетка по нижней левой
диагонали свободна и равна
mMaxCellCost ) {

    wave[tmp.first+1, tmp.second-1] =
d+1 ; // изначально d = 0

queue.push({tmp.first +1,
tmp.second-1}) ; // добавляем в
очередь координаты этой точки

}

if ( соседняя клетка по нижней левой
диагонали является выходом)

{

```

```

    wave[tmp.first+1, tmp.second - 1]
    = d+1 ;

    Way_is_found = TRUE; // ставим
    статус «Путь найден»

}

}

if(way_is_found) //если выход найден

{

l = mFinish.first;

c = mFinish.second; // занесение
координат финиша в переменные

val = wave[ l, c ]; // текущее значение
ВОЛНЫ

for (;;) // цикл продолжится, пока не
будет выхода из функции

index = 0;

up = wave[l-1, c];

down = wave[l+1, c];

right = wave[l, c+1];

left = wave[l, c-1];

up_right = wave[l-1, c+1];

up_left = wave[l-1, c-1];

down_right = wave[l+1, c+1];

```

```

down_left = wave[l+1, c-1]; //
занесение значений волны клеток, соседних
с выходом в переменные

if(up == val -1) { val = up; index =
3;}

if(down == val -1) { val = down;
index = 1;}

if(right == val -1) { val = right;
index = 4;}

if(left == val -1) { val = left;
index = 2;}

if(up_right == val -1) { val =
up_right; index = 5;}

if(up_left == val -1) { val =
up_left; index = 6;}

if(down_right == val -1) { val =
down_right; index = 7;}

if(down_left == val -1) { val =
down_left; index = 8;} // Одно из 8
направлений движения выбирается, если
значение волны в
std::vector<unsigned> wave для него на
1 меньше предыдущей выбранной клетки.

if(val == 0) {index = 0;} // если
достигли нуля, то индекс = 0

switch(index)
{
    case 0:

```

```

        {
            return; //выход из функции
        }
    case 1:
    {
        dl = 1;
        mat.insert(l+dl, c+dc, eCell::WAY);
        break;

// предоставляем путь в std::vector<eCell> mMatrix
    }

    case 2:
    {
        dc = -1;
        mat.insert(l+dl, c+dc, eCell::WAY);
        break;

// предоставляем путь в std::vector<eCell> mMatrix
    }

    case 3:
    {
        dl = -1;
        mat.insert(l+dl, c+dc, eCell::WAY);
        break;

// предоставляем путь в std::vector<eCell> mMatrix
    }

    case 4:
    {
        dc = 1;
        mat.insert(ln+dl, c+dc, eCell::WAY);
        break;

// предоставляем путь в std::vector<eCell> mMatrix
    }

    case 5:
    {
        dl = -1;
        dc = 1;

```



```

        mat.insert(ln+dl, c+dc, eCell::WAY);
        break;

// проставляем путь в std::vector<eCell> mMatrix

    }

    case 6:
    {
        dl = -1;
        dc = -1;
        mat.insert(ln+dl, c+dc, eCell::WAY);
        break;

// проставляем путь в std::vector<eCell> mMatrix

    }

    case 7:
    {
        dl = 1;
        dc = 1;
        mat.insert(ln+dl, c+dc, eCell::WAY);
        break; // проставляем путь в
std::vector<eCell> mMatrix

    }

    case 8:
    {
        dl = 1;
        dc = -1;
        mat.insert(l+dl, c+dc, eCell::WAY);
        break;

// проставляем путь в std::vector<eCell> mMatrix

    }
}

ln += dl;
cn += dc; //меняем координаты в соответствие с
выбранным направлением
dl = 0;
dc = 0;

}

```

```
}
```

```
}
```

Запись в файл `std::vector<eCell>`

`mMatrix` (матрица со стенами, свободными
клетками и проложенным путем) ;

В худшем случае (когда у лабиринта нет стенок, то есть волна пройдет по всем $M \times N$ клеткам) , в процессе построения пути мы пройдем по $N+M-1$ клеткам(клетку выхода вычитаем, т.к на ней мы уже стоим).

Сложность алгоритма может быть оценена, как:

$O(N * M + N + M - 1)$

Здесь N – количество строк в матрице ; M - количество столбцов в матрице

Инструкция пользователя

Перед запуском программы необходимо добавить аргументы командной строки, содержащие пути к входным и выходным файлам для считывания лабиринта и записи лабиринта с проложенным путём(сначала в список аргументов добавляется путь к входному файлу для считывания, а следом за ним выходной файл для записи). Далее необходимо запустить программу. После завершения работы программы появиться надпись «Ways are found».

Входной и выходной файл имеют формат .txt. Во входном файле должно быть написано количество строк и столбцов через пробел, дальше через одну строку записан сам лабиринт без пробелов между числами.

Текстовые примеры

1)

Входной файл:

```
9 20
44444444444444444444
41343333333344333334
43333444444334344434
44343334333343334334
43343444344443444444
43443433333333433334
43343443434444434334
44343333433333334234
44444444444444444444
```

Выходной файл:

```
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 1 3 4 3 3 3 3 3 3 3 4 4 3 3 3 3 4
4 3 * * 3 4 4 4 4 4 3 3 4 3 4 4 4 3 4
4 4 3 4 * 3 3 4 3 3 3 3 4 3 3 3 4 3 3 4
4 3 3 4 * 4 4 4 3 4 4 4 4 3 4 4 4 4 4
4 3 4 4 * 4 3 3 * 3 3 3 3 3 4 3 * 3 3 4
4 3 3 4 * 4 4 * 4 * 4 4 4 4 4 * 4 * 3 4
4 4 3 4 3 * * 3 4 3 * * * * 3 4 2 * 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
```

2)

Входной файл:

```
14 20
44444444444444444444
4143334433333333334
4343433444443444434
4343443433343433334
4343433434343434444
4333434434343433334
4444433334333444434
4343444434444333334
4343333433343434444
4333443444343433343
4443343433343334343
4334443434444443434
4333343333333243334
4444444444444444444
```

Выходной файл:

```
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 1 4 3 * 3 4 4 3 3 3 3 3 3 3 3 3 4
4 * 4 * 4 * 3 4 4 4 4 4 3 4 4 4 4 4 3 4
4 * 4 * 4 4 * 4 3 3 3 4 3 4 3 3 3 3 3 4
4 * 4 * 4 * 3 4 3 4 3 4 3 4 3 4 4 4 4 4
4 3 * 3 4 * 4 4 3 4 3 4 3 4 3 3 3 3 4
4 4 4 4 4 3 * * 3 4 3 3 3 4 4 4 4 3 4
4 3 4 3 4 4 4 4 * 4 4 4 4 4 3 3 3 3 4
4 3 4 3 3 3 3 4 3 * 3 4 3 4 3 4 4 4 4
4 3 3 3 4 4 3 4 4 4 * 4 3 4 3 3 3 4 3 4
4 4 4 3 3 4 3 4 3 * 3 4 3 3 3 4 3 4 3 4
4 3 3 4 4 4 3 4 * 4 4 4 4 4 4 4 3 4 3 4
4 3 3 3 3 4 3 3 3 * * * * 2 4 3 3 3 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
```

3)

Входной файл:

```
20 20
44444444444444444444
43333333323333333334
43344444444444443334
43333333333333333334
43433444444444334334
43433333333333334334
434334333333334334334
43334444434444433334
43334333333333433334
43334343444343433334
43333343313343333334
43334343444343433334
43334333333333433334
43344444434444433334
434334333333334334334
43433333333333334334
43433444444444334334
43333333333333333334
43344333333333433334
44444444444444444444
```

Выходной файл:

```
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
4 3 3 * * * * * 2 3 3 3 3 3 3 3 3 3 4
4 3 * 4 4 4 4 4 4 4 4 4 4 4 3 3 3 4
4 3 3 * 3 3 3 3 3 3 3 3 3 3 3 3 3 4
4 3 4 3 * 4 4 4 4 4 4 4 4 3 3 4 3 3 4
4 3 4 3 3 * * * 3 3 3 3 3 3 3 3 4 3 3 4
4 3 4 3 3 4 3 3 * 3 3 3 3 4 3 3 4 3 3 4
4 3 3 3 4 4 4 4 4 * 4 4 4 4 3 3 3 3 4
4 3 3 3 4 3 3 3 3 3 * 3 3 3 4 3 3 3 3 4
4 3 3 3 4 3 4 3 4 4 4 * 4 3 4 3 3 3 3 4
4 3 3 3 3 3 4 3 3 1 * 3 4 3 3 3 3 3 3 4
4 3 3 3 4 3 4 3 4 4 4 3 4 3 4 3 3 3 3 4
4 3 3 3 4 3 3 3 3 3 3 3 3 3 4 3 3 3 3 4
4 3 3 4 4 4 4 4 4 3 4 4 4 4 3 3 3 3 4
4 3 4 3 3 4 3 3 3 3 3 3 3 4 3 3 4 3 3 4
4 3 4 3 3 3 3 3 3 3 3 3 3 3 3 4 3 3 4
4 3 4 3 3 4 4 4 4 4 4 4 4 3 3 4 3 3 4
4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4
4 3 3 4 4 3 3 3 3 3 3 3 3 4 4 3 3 3 4
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
```

Список источников

- [1] С. Окулов, "Программирование в алгоритмах", Москва, 2014.