



同濟大學  
TONGJI UNIVERSITY

项目说明文档

勇闯迷宫游戏

指导老师：张颖

1851009 沈益立

# 目录

---

## 目录

### 1.分析

- 1.1 背景分析
- 1.2 功能分析

### 2.设计

- 2.1 数据结构设计
- 2.2 类结构设计
- 2.3 成员和操作设计
- 2.4 系统设计

### 3.实现

- 3.1 栈压入功能的实现
  - 3.1.1 栈压入功能流程图
  - 3.1.2 栈压入功能核心代码实现
- 3.2 栈弹出功能的实现
  - 3.2.1 栈弹出功能流程图
  - 3.2.2 栈弹出功能核心代码实现
- 3.3 寻找路径功能的实现
  - 3.3.1 寻找路径功能流程图
  - 3.3.2 寻找路径核心代码实现

### 4.测试

- 4.1 功能测试
  - 4.1.1 一般情况测试
  - 4.1.2 无法到达的情况
  - 4.1.3 仅有一个点是墙的情况
  - 4.1.4 仅有一个点是非墙的情况
  - 4.1.5 迷宫为一维的情况
- 4.2 代码鲁棒性测试
  - 4.2.1 行数为负测试
  - 4.2.2 列数为负测试
  - 4.2.3 入口非法测试
  - 4.2.4 出口非法测试
  - 4.2.5 迷宫坐标非法测试

# 1.分析

---

## 1.1 背景分析

---

已知有这样一个迷宫，它只存在一个入口和出口。一个骑士骑马从入口进入迷宫，他需要在迷宫中 寻找一条通路路从入口走到出口。

## 1.2 功能分析

---

本次设计对功能的要求较为具体，描述如下： 迷宫问题的求解过程可以采用回溯法即在一定的约束条件下试探地搜索前进，若前进中受阻，则及时回头纠正错误另择通路继续搜索的方法。从入口出发，按某一方向向前探索，若能走通，即某处 可达，则到达新点，否则探索下一个方向；若所有的方向均没有通路路，则沿原路路返回前一点，换下 一个方向再继续试探，直到所有可能的道路都探索到，或找到一条通路路，或无路路可走又返回入口点。在求解过程中，为了了保证在达到某一个点后不能向前继续行走时，能正确返回前一个以便便从下一个 方方向向前试探，则需要在试探过程中保存所能够达到的每个点的下标以及该点前进的方向，当找到 出口时试探过程就结束了。

即要求用上文指定的方法实现路路径的寻找和输出。

## 2.设计

---

### 2.1 数据结构设计

---

由题意，很容易联想到图搜索算法、回溯法，可以使用深度优先搜索来实现。而深度优先搜索运用到了栈的数据结构，因此本程序主要采用一个栈作为路径的存储数据结构。

此外，除了栈以外，为了合理地表示出一个坐标，需要自己设计一个至少有二元数据对 `x, y` 的类或结构体，作为坐标节点类。

### 2.2 类结构设计

---

由于作业的相关规定，实现了自己定义的用数组模拟的可变长的栈 `Stack<T>`，以及表示二元数对的 `MyPair<T1, T2>`。为简化IO、增加代码复用性和美观性，同时设计了类 `Maze`，用于输入迷宫、解决迷宫问题和输出迷宫中可行的路径。

### 2.3 成员和操作设计

---

数组式栈类 `Stack`：

公有操作：

```

Stack();           //构造函数
~Stack();          //析构函数
void push(const T &val); //推入新元素到栈顶
void showElements(); //在屏幕上按次序输出栈内元素
bool isEmpty();    //获取是否栈空
void makeEmpty();  //将栈复位
T pop();           //弹出栈顶元素并返回其值
T top();           //返回栈顶元素
int getSize();     //返回栈大小

```

### 私有成员:

```

int m_top;         //表示栈顶
int m_maxSize;     //栈目前的最大大小
T* m_elements;     //数组的首元地址

```

## 二元数类MyPair<T1, T2>:

### 公有操作:

```

MyPair(const T1& a, const T2& b); //含两个参数的构造函数，直接赋值
bool operator ==(const MyPair<T1, T2>& b);
//==运算符重载，判定两个元素是否完全相等
MyPair operator +(const MyPair<T1, T2>& b);
//+运算符重载，返回(x1 + x2, y1 + y2)
friend ostream& operator<<(ostream& output, const
MyPair<T1, T2>& p); //输出流重载，输出为(x, y)形式

```

### 公有成员:

```

T1 first; //第一个元素
T2 second; //第二个元素

```

## 迷宫类Maze

### 公有操作:

```

Maze(); //构造函数
void getMazeData(); //通过IO实现迷宫信息的初始化
bool isInRange(MyPair<int, int> p); //判断某个点是否在迷宫内
void dfsTrace(); //通过回溯找到可行的路径
void showTrace(); //将找到的路径展示在屏幕上

```

### 私有成员:

```

int m_rows; //行数
int m_cols; //列数
MyPair<int, int> m_start; //入口点位
MyPair<int, int> m_end; //出口点位
int m_map[11][11]; //数组模拟迷宫
int m_visited[11][11]; //数组模拟访问过的点
MyPair<int, int> dirs[4] = { //搜索的四个方向
    {1, 0}, {0, 1}, {-1, 0}, {0, -1}
};
Stack<MyPair<int, int>> m_traceStack; //路径栈
int m_fFindWay; //是否找到路径的flag

```

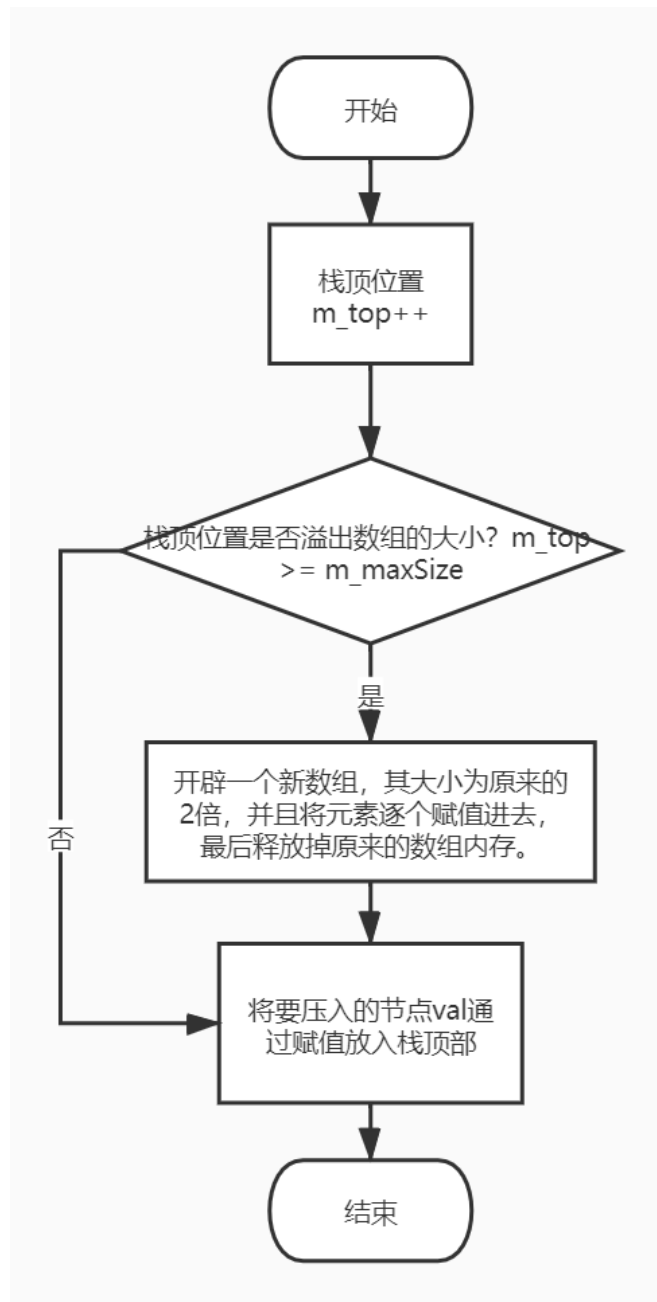
## 2.4 系统设计

程序运行后，系统会自动创建一个迷宫实体化的对象 `probMaze`。之后调用 `getMazeData()` 方法来进行初始化的IO，得到迷宫的数据和出入口位置。随后调用 `dfsTrace()` 来寻找路径并将其压入路径栈中。最后调用 `showTrace()` 将可行的路径打印在屏幕上，若无可行解则会报错。

## 3.实现

### 3.1 栈压入功能的实现

### 3.1.1 栈压入功能流程图



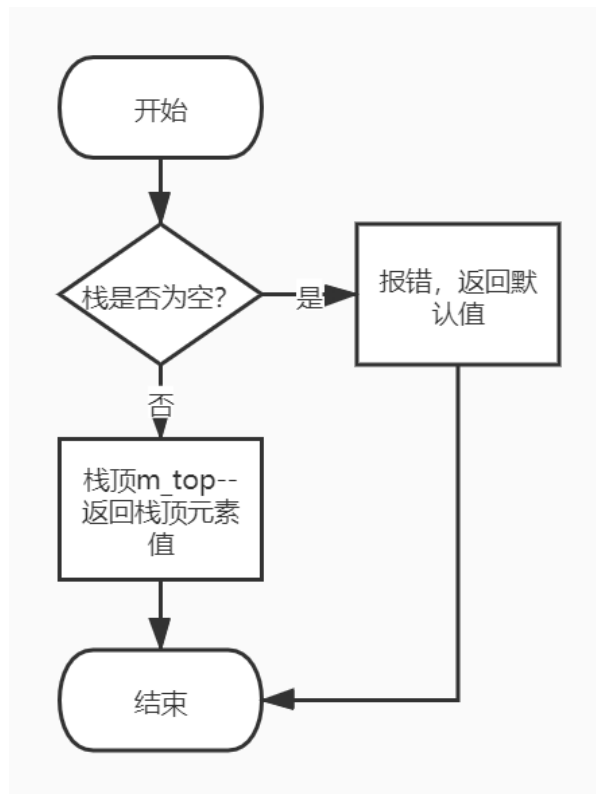
### 3.1.2 栈压入功能核心代码实现

```
template<class T>
void Stack<T>::push(const T& val){
    m_top++;
    if (m_top >= m_maxSize) { //栈溢出
        T* tmp = m_elements;
        m_maxSize *= 2;
        m_elements = new T[m_maxSize];
        for (int i = 0; i < m_maxSize / 2; i++) {
```

```
        m_elements[i] = tmp[i];  
    } //copy  
    delete[] tmp;  
}  
m_elements[m_top] = val;  
}
```

## 3.2 栈弹出功能的实现

### 3.2.1 栈弹出功能流程图



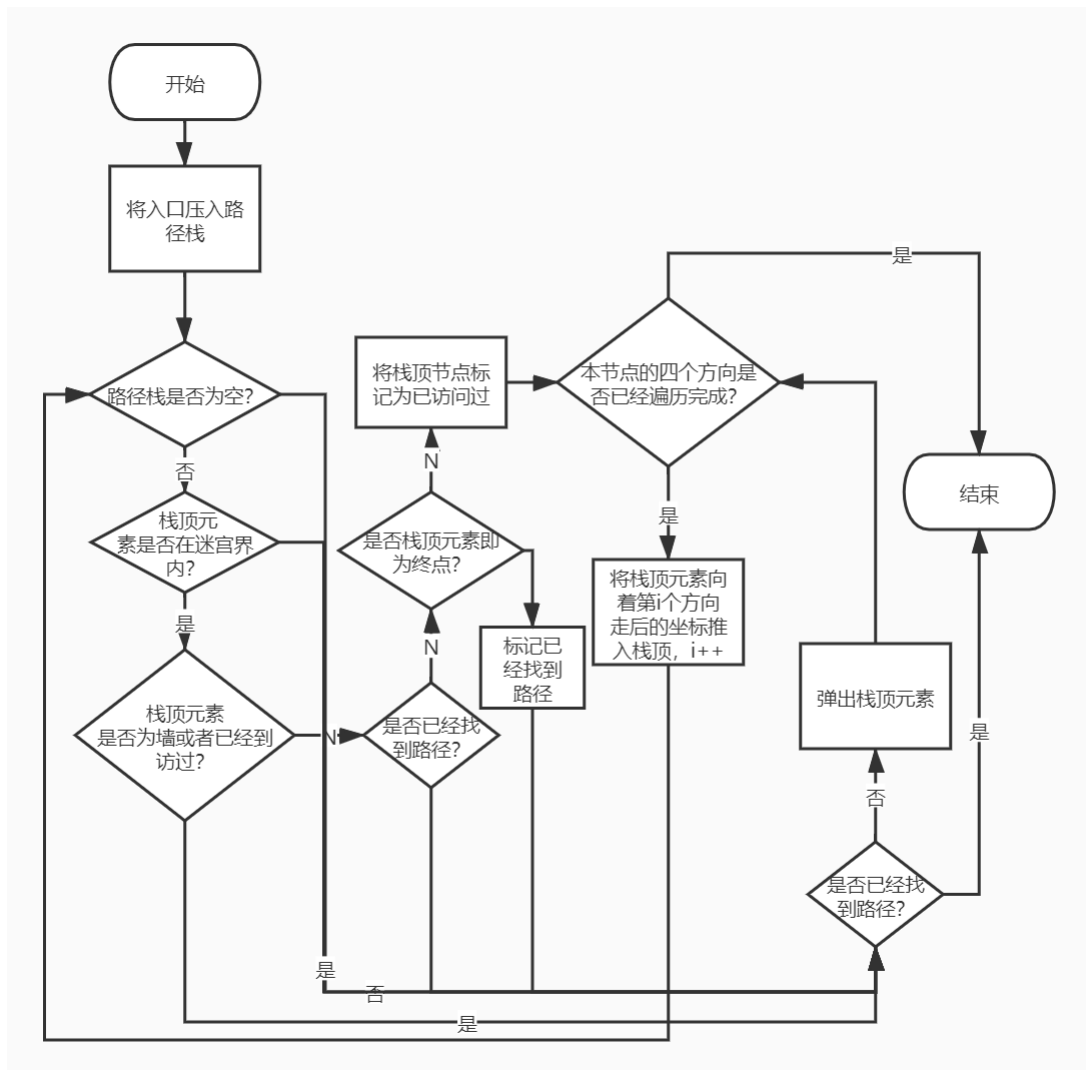
### 3.2.2 栈弹出功能核心代码实现



```
template<class T>
T Stack<T>::pop(){
    if (isEmpty()) {
        cerr << "Empty stack, nothing to pop." << endl;
        return T();
    }
    T retVal = m_elements[m_top--];
    return retVal;
}
```

### 3.3 寻找路径功能的实现

### 3.3.1 寻找路径功能流程图



### 3.3.2 寻找路径核心代码实现

```
void Maze::dfsTrace(){
    if (!m_traceStack.getSize()) {
        return;
    }
    if (!isInRange(m_traceStack.top())) { //出界
        return;
    }
    if (m_map[m_traceStack.top().first]
[m_traceStack.top().second]
    || m_visited[m_traceStack.top().first]
[m_traceStack.top().second]) { //踩墙或者已经访问过
        return;
    }
    if (m_fFindWay) {
        return;
    }
    if (m_traceStack.top() == m_end) {
        m_fFindWay = 1;
        return;
    }
    m_visited[m_traceStack.top().first]
[m_traceStack.top().second] = 1;
    for (int i = 0; i < 4; i++) {
        m_traceStack.push(m_traceStack.top() + dirs[i]);
        dfsTrace();
        if (m_fFindWay) {
            return;
        }
        m_traceStack.pop();
    }
}
```

# 4.测试

## 4.1 功能测试

### 4.1.1 一般情况测试

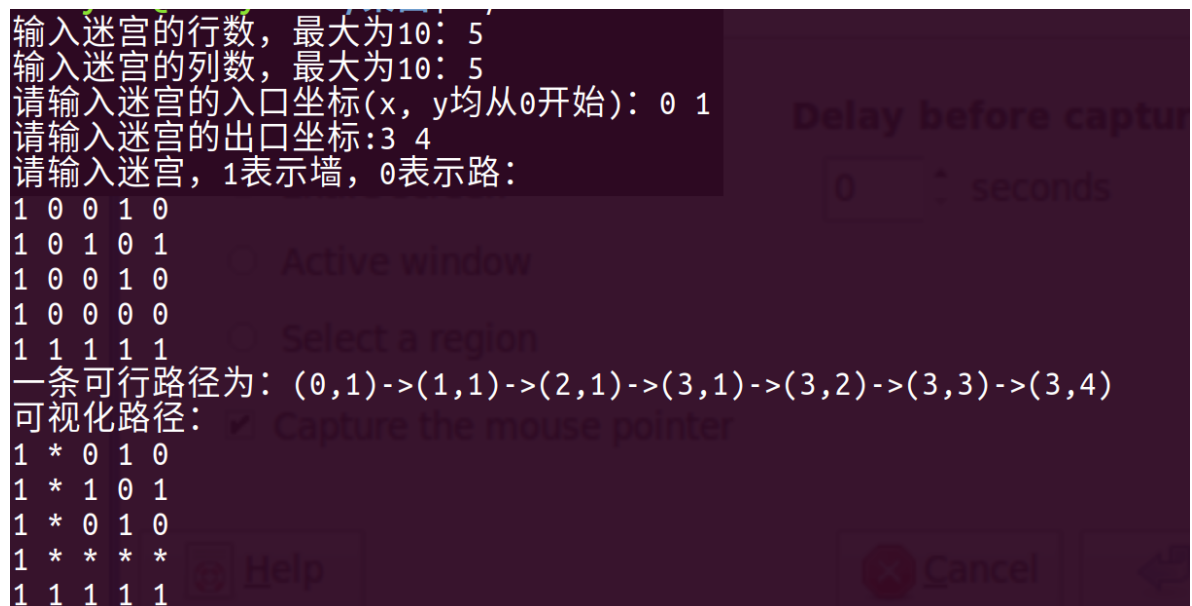
测试用例：

```
5
5
0 1
3 4
1 0 0 1 0
1 0 1 0 1
1 0 0 1 0
1 0 0 0 0
1 1 1 1 1
```

预期结果：

任意一条可行的路径

实验结果：



```
输入迷宫的行数，最大为10：5
输入迷宫的列数，最大为10：5
请输入迷宫的入口坐标(x, y均从0开始)：0 1
请输入迷宫的出口坐标：3 4
请输入迷宫，1表示墙，0表示路：
1 0 0 1 0
1 0 1 0 1
1 0 0 1 0
1 0 0 0 0
1 1 1 1 1
一条可行路径为：(0,1)->(1,1)->(2,1)->(3,1)->(3,2)->(3,3)->(3,4)
可视化路径：
1 * 0 1 0
1 * 1 0 1
1 * 0 1 0
1 * * * *
1 1 1 1 1
```

## 4.1.2 无法到达的情况

测试用例：

```
7
7
1 1
5 5
1 1 1 1 1 1 1
1 0 1 0 0 0 1
1 0 1 0 1 1 1
1 0 0 0 1 0 1
1 0 1 0 1 0 1
1 0 1 0 1 0 1
1 1 1 1 1 1 1
```

预期结果：

没有合理解

实验结果：

```
shenyili@shenyili:~/桌面$ ./a
输入迷宫的行数，最大为10：7
输入迷宫的列数，最大为10：7
请输入迷宫的入口坐标(x, y均从0开始)：1 1
请输入迷宫的出口坐标：5 5
请输入迷宫，1表示墙，0表示路：
1 1 1 1 1 1 1
1 0 1 0 0 0 1
1 0 1 0 1 1 1
1 0 0 0 1 0 1
1 0 1 0 1 0 1
1 0 1 0 1 0 1
1 1 1 1 1 1 1
对于给定的入口、出口、迷宫，没有合理解
```

## 4.1.3 仅有一个点是墙的情况

测试用例：

```
1
1
0 0
0 0
1
```

预期结果:

```
NULL
```

实验结果:

```
shenyili@shenyili:~/桌面$ ./a
输入迷宫的行数，最大为10: 1
输入迷宫的列数，最大为10: 1
请输入迷宫的入口坐标(x, y均从0开始): 0 0
请输入迷宫的出口坐标: 0 0
请输入迷宫，1表示墙，0表示路:
1
对于给定的入口、出口、迷宫，没有合理解
```

#### 4.1.4 仅有一个点是非墙的情况

测试用例:

```
1
1
0 0
0 0
0
```

预期结果:

```
*
```

实验结果:

```
shenyili@shenyili:~/桌面$ ./a
输入迷宫的行数，最大为10: 1
输入迷宫的列数，最大为10: 1
请输入迷宫的入口坐标(x, y均从0开始): 0 0
请输入迷宫的出口坐标: 0 0
请输入迷宫，1表示墙，0表示路:
0
一条可行路径为: (0,0)
可视化路径:
*
```

## 4.1.5 迷宫为一维的情况

测试用例:

```
1
10
0 1
0 8
0 0 0 0 0 0 0 0 0 0
```

预期结果:

```
0 * * * * * * * * 0
```

实验结果:

```
shenyili@shenyili:~/桌面$ ./a
输入迷宫的行数，最大为10: 1
输入迷宫的列数，最大为10: 10
请输入迷宫的入口坐标(x, y均从0开始): 0 1
请输入迷宫的出口坐标: 0 8
请输入迷宫，1表示墙，0表示路:
0 0 0 0 0 0 0 0 0 0
一条可行路径为: (0,1)->(0,2)->(0,3)->(0,4)->(0,5)->(0,6)->(0,7)->(0,8)
可视化路径:
0 * * * * * * * * 0
```

## 4.2 代码鲁棒性测试

## 4.2.1 行数为负测试

测试用例：

-1

预期结果：

行数非法

实验结果：

```
shenyili@shenyili:~/桌面/maze$ ./Maze
输入迷宫的行数，最大为10： -1
行数非法，请重新输入合法的行数
```

## 4.2.2 列数为负测试

测试用例：

1  
-1

预期结果：

列数非法

实验结果：

```
shenyili@shenyili:~/桌面/maze$ ./Maze
输入迷宫的行数，最大为10： 1
输入迷宫的列数，最大为10： -1
列数非法，请重新输入合法的列数：
```

## 4.2.3 入口非法测试

测试用例：

```
1
1
-1 -1
```

预期结果：

坐标非法

实验结果：

```
shenyili@shenyili:~/桌面/maze$ ./Maze
输入迷宫的行数，最大为10： 1
1
-1 -1输入迷宫的列数，最大为10： 请输入迷宫的入口坐标(x
坐标非法，请重新输入合法的坐标：█
```

## 4.2.4 出口非法测试

测试用例：

```
1
1
0 0
-1 -1
```

预期结果：

坐标非法

实验结果：

```
shenyili@shenyili:~/桌面/maze$ ./Maze
输入迷宫的行数，最大为10： 1
1
0 0
-1 -1输入迷宫的列数，最大为10： 请输入迷宫的入口坐
出口坐标：
坐标非法，请重新输入合法的坐标：█
```

## 4.2.5 迷宫坐标非法测试



测试用例：

```
1
1
0 0
0 0
2
```

预期结果：

迷宫有非法数据输入

```
shenyili@shenyili:~/桌面/maze$ ./Maze
输入迷宫的行数，最大为10： 1
1
0 0
0 0
2输入迷宫的列数，最大为10： 请输入迷宫的入
坐标：请输入迷宫，1表示墙，0表示路：

迷宫有非法数据，请重新输入。
```