同济大学 Tongji University

数据结构课程设计项目说明文档

——勇闯迷宫游戏

作者姓名：陈垲昕

学号：[redacted]

指导教师：张颖

学院/专业：软件学院/软件工程

1. 分析

1.1 背景分析

迷宫求解问题是基本算法问题，是计算机图论的入门基础。本题为设计一个简单的寻路算法，目的在于求解一个有一个起点与一个终点的迷宫并给出任意一条求解路径。

1.2 功能分析

迷宫只有两个门，一个门叫入口，另一个门叫出口。一个骑士骑马从入口进入迷宫，迷宫设置很多障碍，骑士需要在迷宫中寻找通路以到达出口。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，则沿原路返回前一点，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，或无路可走又返回入口点。在求解过程中，为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从下一个方向向前试探，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。

2.设计

2.1 主要数据结构设计

由于本题只要求求出迷宫是否有解以及有解的情况下任意一个合法路径，故可使用深度优先搜索(Depth First Search)的思想。为避免使用递归方法进行深度优先搜索时由于调用递归造成的效率损失，故考虑使用非递归方法——使用存储路径信息的栈进行搜索。基本思路是：从起点开始搜索时，取出栈顶的当前坐标，并将此后可行的坐标放入栈中，到达终点时，可以生成一个反映可行路径的栈，若栈为空，则该迷宫不可解。

本课程设计中的栈为模板栈，使用链表形式存储，节点单位为模板链表节点（包含指定类型数据以及下一个结点的指针类， 支持默认构造与值初始化构造）。具体实现的功能包括：入栈，出栈，判断是否为空，获取顶部元素，从栈顶向栈底依次输出节点信息，以及析构清空等操作。

链表结点结构体设计

```
template <class T>
struct LinkNode {
    //数据成员
    T data;
    LinkNode* nextNode;
    //构造函数
    LinkNode() :nextNode(nullptr) {}
    LinkNode(const T& value) :data(value), nextNode(nullptr) {}
};
```

链表栈类设计

```
template <class T>
class LinkStack {
public:
    //构造与析构函数
    LinkStack() = default;
    ~LinkStack() { clear(); };
    //内容获取
```

```
    bool empty()const { return stackSize == 0; }
    T top() { return _pTop->data; }
    //链表单元操作
    void push(const T& x);
    void pop();
    void printFromTop()const;
    //链表操作
    void clear();
private:
    //栈顶指针
    LinkNode<T>* _pTop;
    //元素个数
    size_t stackSize;
};
```

2.2 坐标信息结构体设计

由于使用二维数组存储迷宫信息，故迷宫每个位置都是由一个横坐标x，纵坐标y形容的坐标点，在用栈存储坐标点时，还需存储下一个节点的遍历方向的信息，以供路径的选择与生成。本课程设计将横坐标x、纵坐标y以及表示方向的信息direction(int类型)封装于一个结构体中Point中，作为坐标信息基本单元。

Point结构体提供默认构造，值初始化、复制构造函数。默认情况下，x=y=-1表示空坐标点，direction初始化为0。（DFS搜索方向为四个，对应direction的值为0-3,只有direction=0,1,2,3时才是合法的搜索方向）。在此基础上，重载了==与<<符号便于后续程序设计表示

```
struct Point {
    Point() :x(-1), y(-1),direction(0) {}                          //默认构造函数
    Point(int a, int b) :x(a), y(b), direction(0) {}               //值初始化构造函数
    Point(const Point& pt) :x(pt.x), y(pt.y), direction(0) {}      //复制构造函数

    bool operator==(const Point& rhs) { return x == rhs.x && y ==rhs.y; }   //重载==
    friend std::ostream& operator<<(std::ostream& os, const Point& pt)      //重载<<
    {
        os << "<" << pt.x << " " << pt.y << ">";
        return os;
    }

    //坐标数组
    int x; int y;

    //方向，默认构造的情况下x=-1,y=-1方向指示为0,1,2,3分别代表DFS的四个方向,超过3后无效
    int direction;
};
```

2.3 系统类设计

本课设通过MazeSolver类运行系统，此类成员包括：1.实现DFS所需要的链表栈，信息以及标记记录（具体见代码段）;2.构造与析构函数；3.表示流程的运行函数(具体见下) ;4.主程序中使用的程序运行程序

```
class MazeSolver {
public:
    //构造函数与析构函数
    MazeSolver() :_mazeMap(nullptr), _currentRow(0),
_currentColumn(0),_startCoord(-1,-1),_endCoord(-1,-1) {}
    ~MazeSolver();

    //输入迷宫
    void inputMaze();
    //展示迷宫
    void showMaze()const;

    //从start到end使用非递归方法解决dfs问题
    void dfsSolveMaze(Point start,Point end);

    //程序运行程序
    void run();

private:
    char** _mazeMap;              //表示迷宫的二维动态指针
    bool** _markMap;              //表示当前节点是否可走标记的二维动态指针,可走时为true，不
可走时为false，防止陷入死循环
    int _currentRow;              //当前迷宫的行数，为空时设为-1
    int _currentColumn;           //当前迷宫的列数，为空时设为-1

    Point _startCoord;            //起始坐标，为空时设为(-1,-1)
    Point _endCoord;              //终止坐标，为空时设为(-1,-1)

    LinkStack<Point> _dfsStack; //dfs栈，节点为Point类型
};
```

2.4 基本信息设定

```
#define START 'S'    //输入迷宫时，若输入'S'，代表此处为起点，可在代码中修改
#define ROUTE 'X'    //输入迷宫时，若输入'X'，代表此处为通路，可在代码中修改
#define END 'E'      //输入迷宫时，若输入'E'，代表此处为终点，可在代码中修改

//用于表示搜索方向的direction数组
const int directionX[4] = { -1,0,1,0 };
const int directionY[4] = { 0,-1,0,1 };
```

3.核心功能实现

3.1 DFS非递归寻路算法实现

描述：通过 MazeSolver 类的 dfsSolveMaze 函数实现，输入地图中表示起点的坐标与表示终点的坐标即可生成路径。

思路：先将终点坐标放入栈中（direction=0），设置起点处为不可走。此后进行循环：在一个以栈空为终止条件的条件循环中，在每次循环开始前获取栈顶元素。对该点的四个方向遍历，获取由目前点的direction得到的下一个点的可走坐标，使其入栈并重新开始新一轮循环。当遍历完4个方向后（走入了死路），使坐标点退栈，引发回溯并尝试新的路径。反复循环直至栈空或达到了起点后退出循环。最后根据栈的存储情况告知用户迷宫的解：若栈为空，无解；若栈不为空，则从栈顶（起点）到栈底（终点）顺序输出路径。

```
void MazeSolver::dfsSolveMaze(Point start,Point end)
{
    //由于打印栈是从栈顶到栈底打印，故从终点开始向起点遍历
    //把终点坐标放入栈中，设该点处不可走
    Point endCoord(end);
    _dfsStack.push(endCoord);
    _markMap[endCoord.x][endCoord.y] = false;


    while (!_dfsStack.empty())
    {
        //获取栈顶元素
        LinkNode<Point>* currPointNode=_dfsStack.getTop();
        //判断终止条件，即dfs走到了start（终点）
        if (currPointNode->data == start)
        {
            break;
        }

        //对当前点的四个方向遍历，（0,1,2,3）分别代表dfs的四个方向（上下左右）
        for(;currPointNode->data.direction<4;currPointNode->data.direction++)
        {
            int i = currPointNode->data.direction;
            Point tempPoint(currPointNode->data);
            //边界判断，如果到了边界，则直接处理下一个方向
            if (tempPoint.x + directionX[i] < 0 || tempPoint.x + directionX[i]
>= _currentColumn ||
                tempPoint.y + directionY[i] < 0 || tempPoint.y + directionY[i]
>= _currentRow)
            {
                continue;
            }
            //若不是边界，获取由目前点的direction得到的下一个点的坐标
            Point nextPoint(tempPoint.x + directionX[i], tempPoint.y +
directionY[i]);
            //如果该坐标可走（该点的)_markMap为真，入栈,设为不可走
            if (_markMap[nextPoint.x][nextPoint.y])
            {
                _dfsStack.push(nextPoint);
                _markMap[nextPoint.x][nextPoint.y] = false;
                break;
            }
        }
```

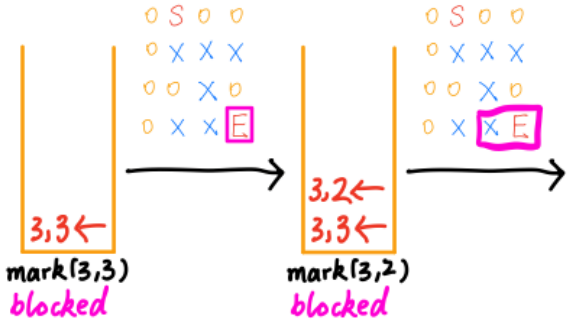
```
        //如果当前点的direction为4，说明遍历完了四个方向，走到死路，回退栈并设为可走
        if (currPointNode->data.direction == 4)
        {
            _markMap[currPointNode->data.x][currPointNode->data.y] = true;
            _dfsStack.pop();
            if (!_dfsStack.empty())
            {
                _dfsStack.getTop()->data.direction++;
            }
        }
    }

    //迷宫最终解处理
    if (_dfsStack.empty())
    {
        std::cout << "该迷宫没有解" << std::endl;
    }
    else
    {
        std::cout << "该迷宫的解为:" << std::endl;
        _dfsStack.printFromTop();
    }
}
```

搜索流程图例与相应实验结果

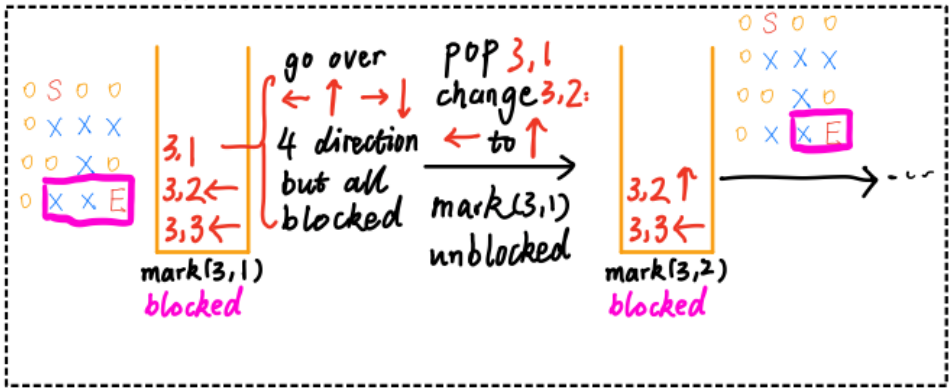
搜索流程图例

maze instance

	0C	1C	2C	3C
0R	O	start: S	O	O
1R	O	X	X	X
2R	O	O	X	O
3R	O	X	X	E end



a retrospective process:



several retrospective process

...

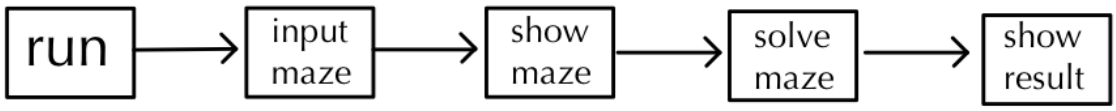
print from top to bottom

相应实验结果

```
请输入迷宫的行数与列数:
4 4
请输入迷宫(S为起点,E为终点,X为通路,其余为墙):
0S00
0XXX
00X0
0XXE
此迷宫的图样:
        0列     1列     2列     3列
0行     0       S       0       0
1行     0       X       X       X
2行     0       0       X       0
3行     0       X       X       E
该迷宫的解为:
<0 1>-><1 1>-><1 2>-><2 2>-><3 2>-><3 3>
```

3.2 运行函数run()基本描述

下图为程序运行主体框架




```
void MazeSolver::run()
{
    inputMaze();                                //输入迷宫
    showMaze();                                 //展示迷宫
    dfsSolveMaze(_startCoord,_endCoord);        //根据起始坐标与终止坐标生成搜索路径并告知用
户结果
}

int main(int argc,char* argv[])
{
    MazeSolver ms;
    ms.run();                                   //运行MazeSolver类
}
```

4.测试

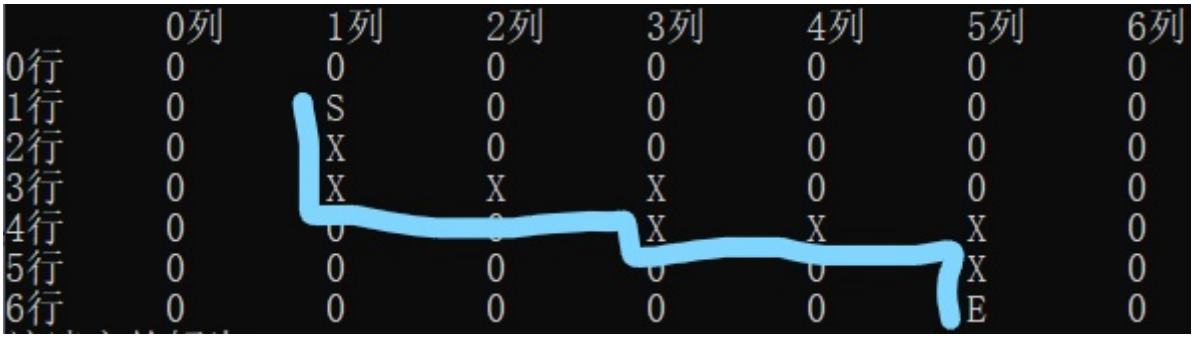
4.1 常规结果测试

Task 1 单通路迷宫

测试样例：

```
0000000
0S00000
0X00000
0XXX000
000XXX0
00000X0
00000E0
```

预期结果：



从图中看，迷宫解为：<1 1>-><2 1>-><3 1>-><3 2>-><3 3>-><4 3>-><4 4>-><4 5>-><5 5>-><6 5>

实际结果：

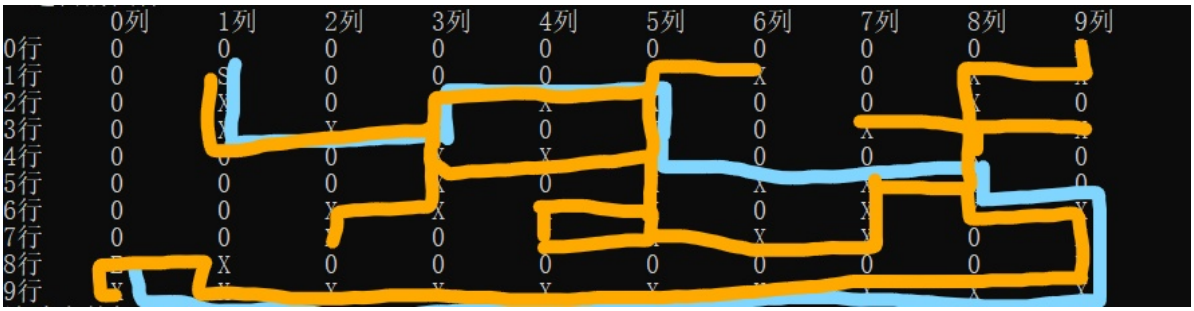
```
此迷宫的图样:
      0列     1列     2列     3列     4列     5列     6列
0行     0       0       0       0       0       0       0
1行     0       S       0       0       0       0       0
2行     0       X       0       0       0       0       0
3行     0       X       X       X       0       0       0
4行     0       0       0       X       X       X       0
5行     0       0       0       0       0       X       0
6行     0       0       0       0       0       E       0
该迷宫的解为:
<1 1>-><2 1>-><3 1>-><3 2>-><3 3>-><4 3>-><4 4>-><4 5>-><5 5>-><6 5>
```

Task 2 多岔路迷宫

测试样例：

```
000000000X
0S000XX0XX
0X0XXX00X0
0XXX0X0XXX
000XXX00X0
000X0XXXX0
00XXXX0XXX
00X0XXXX0X
EX0000000X
XXXXXXXXXX
```

预期结果：




```
从图中看(橙色为所有岔路分支，蓝色为一种可行解)，迷宫解为：<1 1>-><2 1>-><3 1>-><3 2>-><3 3>-><2 3>-><2 4>-><2 5>-><3 5>-><4 5>-><5 5>-><5 6>-><5 7>-><5 8>-><6 8>-><6 9>-><7 9>-><8 9>-><9 9>-><9 8>-><9 7>-><9 6>-><9 5>-><9 4>-><9 3>-><9 2>-><9 1>-><9 0>-><8 0>
```

实际结果：

```
此迷宫的图样:
        0列     1列     2列     3列     4列     5列     6列     7列     8列     9列
0行     0       0       0       0       0       0       0       0       0       X
1行     0       S       0       0       0       X       X       0       X       X
2行     0       X       0       X       X       X       0       0       X       0
3行     0       X       X       X       0       X       0       X       X       X
4行     0       0       0       X       X       X       0       0       X       0
5行     0       0       0       X       0       X       X       X       X       0
6行     0       0       X       X       X       X       0       X       X       X
7行     0       0       X       0       X       X       X       X       0       X
8行     E       X       0       0       0       0       0       0       0       X
9行     X       X       X       X       X       X       X       X       X       X
该迷宫的解为:
<1 1>-><2 1>-><3 1>-><3 2>-><3 3>-><2 3>-><2 4>-><2 5>-><3 5>-><4 5>-><5 5>-><5 6>-><5 7>-><5 8>-><6 8>-><6 9>-><7 9>-><8 9>-><9 9>-><9 8>-><9 7>-><9 6>-><9 5>-><9 4>-><9 3>-><9 2>-><9 1>-><9 0>-><8 0>
```

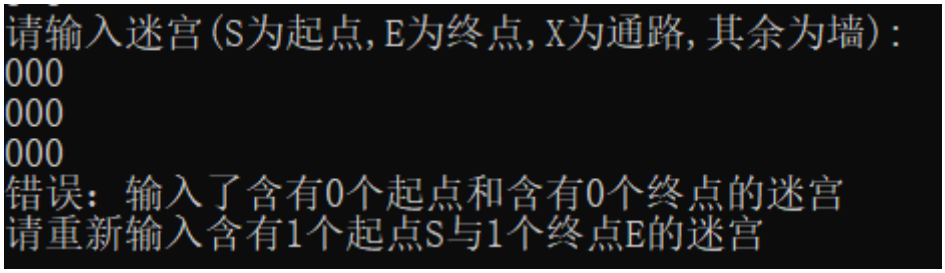
4.2 错误结果测试

Task 1 无起点迷宫

测试样例:

```
000
000
000
```

实际结果：

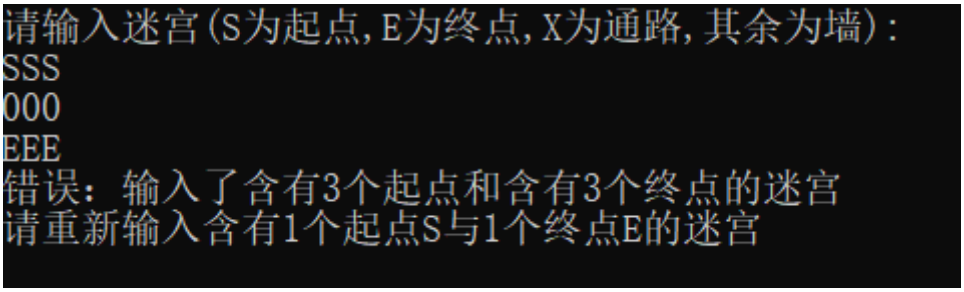


Task 2 多起点（终点）迷宫

测试样例:

```
SSS
000
EEE
```

实际结果:



Task 3 无解迷宫

测试样例:

```
S00
00E
XXX
可见从S到E无通路
```

实际结果: