数据结构项目文档

> 题目：勇闯迷宫游戏
>
> 指导教师：张颖
>
> 姓名：王亮
>
> 学号：1653340

数据结构项目文档

- 一、题目分析
  - 1. 项目简介
  - 2. 功能需求
  - 3. 设计思路
- 二、设计
  - 1. 数据结构设计
    - 1.1 迷宫的二维数组表示
    - 1.2 栈
  - 2. 类的设计
    - 2.1 栈中元素数据
    - 2.1 SeqStack
  - 3. 函数设计
    - 3.1 主函数
    - 3.2 寻找路径
    - 3.3 输出迷宫和路线
- 三、实现
  - 1. 寻找路径
    - 1.1 流程图
    - 1.2 关键代码
  - 2. 输出地图和路线
    - 2.1 输出地图
    - 2.2 输出路线
- 四、测试
  - 1. 功能测试

一、题目分析

1. 项目简介

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。

2. 功能需求

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。

3. 设计思路

这道题采用回溯法求解迷宫问题，回溯法也称为试探法，这种法方法将问题的候选解按某种顺序逐一枚举和检验。用回溯法求解问题时常常使用递归方法进行试探，或使用栈帮助向前试探和回溯。

二、设计

1. 数据结构设计

1.1 迷宫的二维数组表示

用一个二维数组`maze[row+2][col+2]`来表示迷宫，当数组元素`maze[i][j] = 1`时，表示该位置是墙壁，不能通行；当`maze[i][j] = 0`时，表示该位置是通路。

迷宫地图由题目给出，所以直接写进代码中：

```
int maze[row+2][col+2] =
{
    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, 0, 0, 1,
    1, 0, 1, 0, 1, 0, 1,
    1, 1, 1, 1, 1, 1, 1
};
```

定义一个前进方向表，给出各个向各个方向的偏移量，用于实现向四个方向的试探前进。

```
//前进方向表
struct offsets
{
    int a, b;
    char* dir;
};

offsets direction[4] = { {-1, 0, "up"}, {0, 1, "right"}, {1, 0, "down"},
{0, -1, "left"}};
```

为了防止重走原路，另外设置一个标志矩阵`mark[row+2][col+2]`，它的所有元素都初始化为0.一旦走到迷宫的某个位置，则将mark矩阵中该位置对应的元素置为1。下次这个位置就不能再走了。

1.2 栈

在这个项目中，使用一个栈来存储试探过程中走过的路径，从而将回溯的递归算法改为非递归算法。

2. 类的设计

为了实现非递归的回溯，需要实现一个栈。这里使用数组实现一个顺序栈。

2.1 栈中元素数据

栈中每个元素保存一个迷宫中位置相关的数据，用一个三元组结构存储。

```
//栈中的三元组结构
struct items
{
    int x, y, dir;     //dir为前进方向表中的下标
};
```

2.1 SeqStack

顺序栈的模板类：

```
const int stackIncreament = 20; //栈溢出时扩展空间的增量

template <typename T>class SeqStack{
public:
    SeqStack(int sz =50);
    ~SeqStack(){
        delete []elements;
    }
    void Push(const T &x);
    bool Pop(T &x);
    bool getTopData(T &x)const;
    bool getElementByIndex(int index, T &x)const;
```

```
    bool IsEmpty()const{
        return top == -1;
    }
    bool IsFull()const{
        return top == maxSize-1;
    }
    int getSize()const{
        return top+1;
    }
    void MakeEmpty(){
        top = -1;
    }
    friend ostream& operator << (ostream &out, SeqStack<T> &s)  {
        out << "Index of top is: " << s.top << endl;
        for (int i = 0; i <= s.top; i ++)
            out << i << ": " << s.elements[i] << endl;
        return out;
    }
private:
    T *elements;
    int top;
    int maxSize;
    void overflowProcess();
};
```

类中主要包括三个成员数据：元素数组指针、栈顶位置下标、最大栈空间。

求解迷宫问题主要用到类中的 `Push(const T& x)` 和 `Pop(T &x)` 两个方法。

最后输出迷宫路线结果用到类中的`getElementByIndex(int index, T& x) const`方法。

3. 函数设计

3.1 主函数

主函数首先需要初始化用到的数据，由于迷宫直接在代码中定义出，所以不需要手动输入。主函数直接调用寻找路线的函数，并把结果输出即可。

3.2 寻找路径

利用栈进行回溯。

3.3 输出迷宫和路线

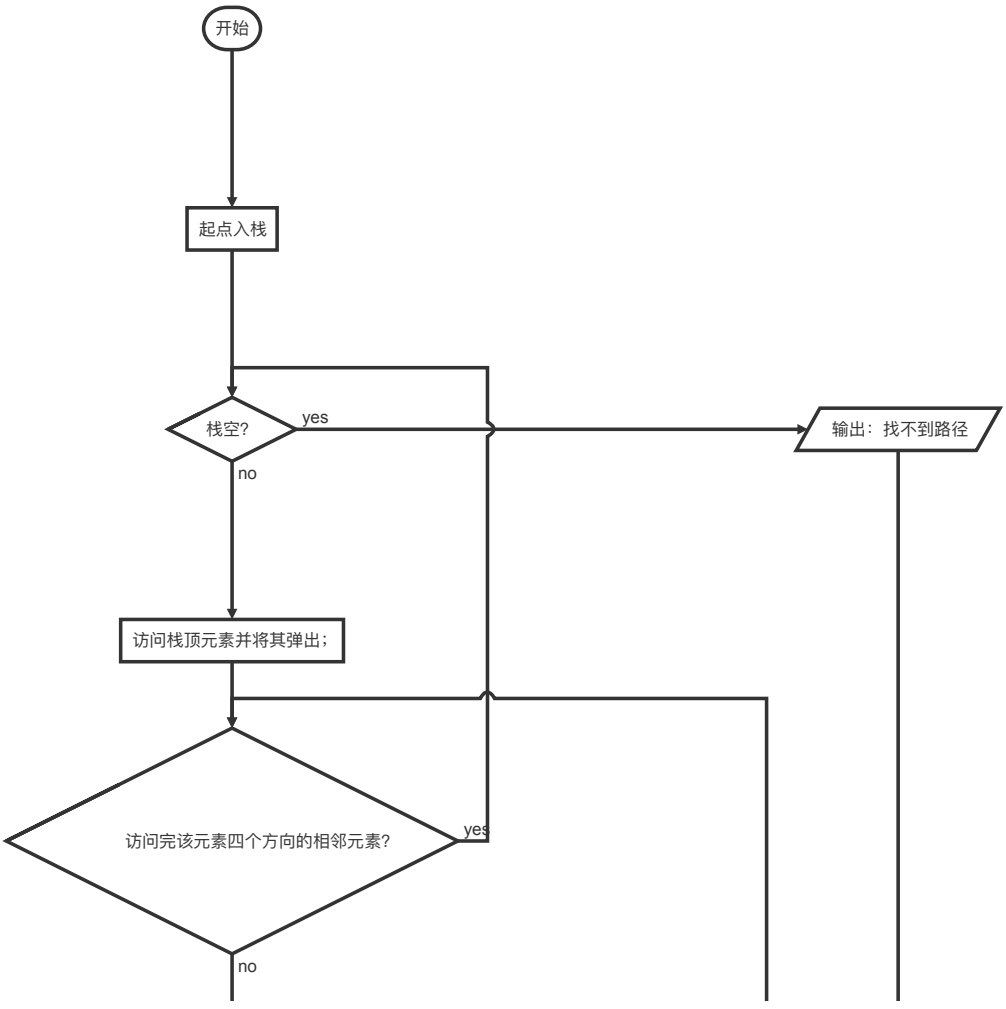
迷宫用int数组保存，输出时按序遍历数组的每个元素，根据元素的值，将其转换成题目规定的输出形式。

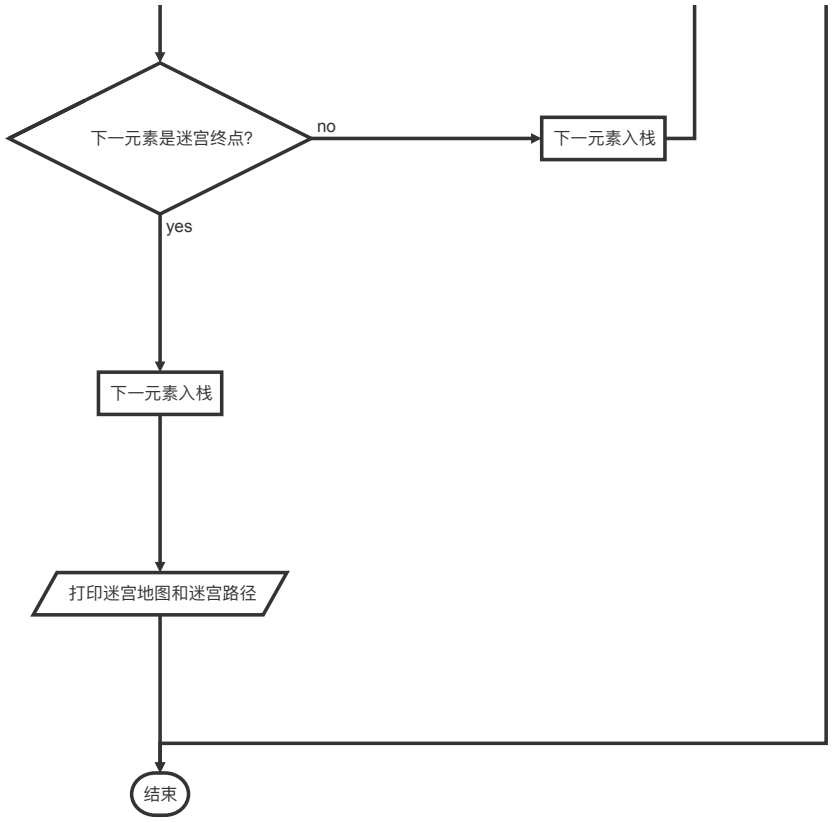
找到的路线保存在栈中，从栈底开始访问，知道栈顶，按规定形式输出栈中保存的位置坐标。

三、实现

1. 寻找路径

1.1 流程图


开始
起点入栈
栈空?
yes
输出：找不到路径
no
访问栈顶元素并将其弹出；
访问完该元素四个方向的相邻元素?
yes
no



1.2 关键代码

```
/*
 @brief: 使用栈进行回溯从而求解迷宫问题的算法
 */
void seekPath(int targetX, int targetY)
{
    int i, j, d, g, h;
    mark[1][1] = 1;
```

```
    SeqStack<items> st;
    items tmp;
    tmp.x = 1; tmp.y = 1; tmp.dir = 1;           //首结点（起点）的第一个搜索方
向，（不必向上方搜索）
    st.Push(tmp);
    maze[tmp.x][tmp.y] = 2;
    while (st.IsEmpty() == false)
    {
        st.Pop(tmp);
        maze[tmp.x][tmp.y] = 0;
        i = tmp.x; j = tmp.y; d = tmp.dir;
        while (d < 4)
        {
            g = i+direction[d].a;
            h = j+direction[d].b;
            if (g == targetX && h == targetY)              //到达出口
            {
                tmp.x = i; tmp.y = j; tmp.dir = d;
                st.Push(tmp);
                maze[tmp.x][tmp.y] = 2;
                maze[targetX][targetY] = 2;
                cout<<"迷宫地图："<<endl;
                printMaze();
                cout<<"迷宫路径："<<endl;
                printSt(st);
                cout<<"("<<targetX<<", "<<targetY<<")"<<endl;
                return;
            }
            if (maze[g][h] == 0 && mark[g][h] == 0)
            {
                mark[g][h] = 1;
                tmp.x = i; tmp.y = j; tmp.dir = d;
                st.Push(tmp);
                maze[tmp.x][tmp.y] = 2;
                i = g; j = h; d = 0;
            }
            else d++;
        }
    }
    cout<<"No path in maze."<<endl;
}
```

2. 输出地图和路线

2.1 输出地图

遍历迷宫数组每个元素，根据元素中数据的值，使用switch结构控制输出形式。

```
/*
```

```
 *@brief: 打印迷宫地图
 */
void printMaze()
{
    int cnt1, cnt2;
    cout<<setiosflags(ios::left);
    //输出第一行
    cout<<setw(7)<<"";
    for (cnt1=0; cnt1<=col+1; cnt1++)
    {
        cout<<setw(8)<<to_string(cnt1)+"列";
    }
    cout<<endl;
    //输出迷宫各行
    for (cnt1=0; cnt1<=row+1; cnt1++)
    {
        cout<<setw(8)<<to_string(cnt1)+"行";
        for (cnt2=0; cnt2<=col+1; cnt2++)
        {
            switch (maze[cnt1][cnt2]) {
                case 1:
                    cout<<setw(7)<<"#       ";
                    break;
                case 0:
                    cout<<setw(7)<<"0";
                    break;
                case 2:
                    cout<<setw(7)<<"x";
                    break;
                default:
                    break;
            }
        }
        cout<<endl;
    }
}
```

2.2 输出路线

从栈底到栈顶遍历访问栈中的每个元素，按规定形式输出路线上每个位置结点的坐标。

```
/*
 @brief: 打印路径，不包含出口结点
 */
void printSt(SeqStack<items>& s)
{
    items tempResult;        //用于暂存每个结果
    int cnt;
    for (cnt = 0; cnt <= s.getSize()-1; cnt++)
```

```
    {
        if(s.getElementByIndex(cnt, tempResult))
        {
            cout<<"("<<tempResult.x<<", "<<tempResult.y<<") ---> ";
        }
    }
}
```

四、测试

1. 功能测试

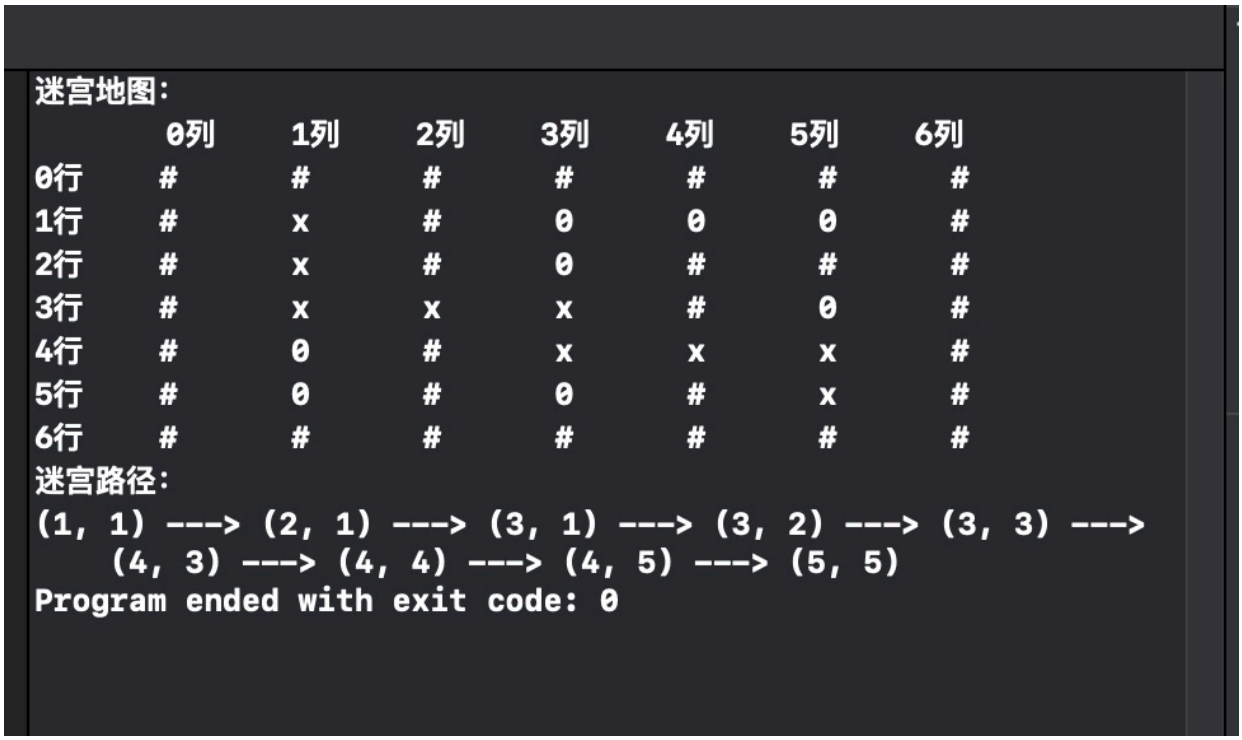
初始地图数据（二维数组）：

```
int maze[row+2][col+2] =
{
    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, 0, 0, 1,
    1, 0, 1, 0, 1, 0, 1,
    1, 1, 1, 1, 1, 1, 1
};
```

1表示墙壁，0表示可以走的通路。

规定左上角为起点，右下角为终点。

程序运行结果：



寻找到的路径在地图中用 'x' 符号标出，并输出了路径中的每个位置的坐标。

测试结果正确。