

一、实验目的：

学习栈的应用和回溯法。

迷宫问题是栈应用的一个典型例子。求解过程可采用回溯法。回溯法是一种不断试探且及时纠正错误的搜索方法。

二、实验要求：

迷宫只有两个门，一个叫做入口，另一个叫做出口。把一只老鼠从一个无顶盖的大盒子的入口处赶进迷宫。迷宫中设置很多隔壁，对前进方向形成了多处障碍，在迷宫的唯一出口处放置了一块奶酪，吸引老鼠在迷宫中寻找通路以到达出口。求解迷宫问题，即找出从入口到出口的路径。

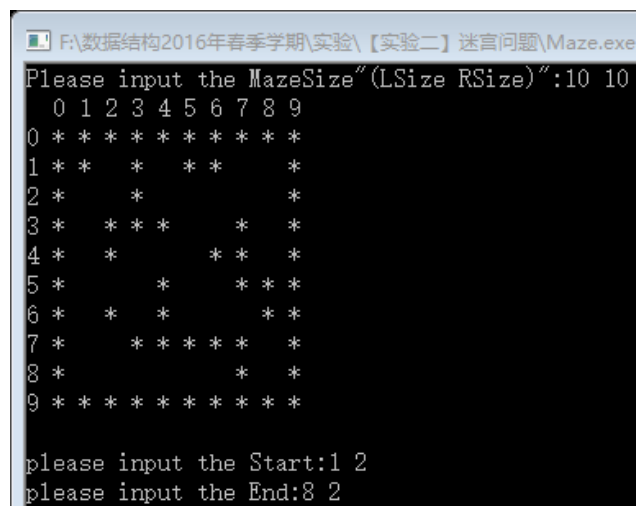
三、实验内容和实验步骤：

1、需求分析：

(1) 输入形式：

- 迷宫大小： LSize RSize 两个整型数，空格隔开，要求输入是大于等于 2 的整数。
- 输入起点（和终点）坐标： ij 两个整型数，空格隔开，要求起点和终点均不能是有障碍的坐标，并且不能是迷宫外的坐标。

c) 截图：



```
F:\数据结构2016年春季学期\实验\【实验二】迷宫问题\Maze.exe
Please input the MazeSize"(LSize RSize)":10 10
 0 1 2 3 4 5 6 7 8 9
0 * * * * * * * * *
1 * *   *   *   *
2 *   *       *
3 *   * * *   *   *
4 *   *       * * *
5 *       *   * * *
6 *   *   *       *
7 *       * * * * *
8 *               *
9 * * * * * * * * *

please input the Start:1 2
please input the End:8 2
```

(2) 输出形式：解迷宫路径：图示和坐标

- 图示起点用 S 标志，终点用 E 标志，图中路径用 "." 标示。
- 坐标用括号加以标识，中间用 "-">" 连接。
- 截图：

d)

```

F:\数据结构2016年春季学期\实验\【实验二】迷宫问题\Maze.exe

please input the Start:1 2
please input the End:8 2

(1,2) -> (2,2) -> (2,1) -> (3,1) -> (4,1) -> (5,1) -> (6,1) -> (7,1) -> (7,2) ->
(8,2)

'*' is obstacle, '+' is the pass road.
 0 1 2 3 4 5 6 7 8 9
0 * * * * * * * * *
1 * * S * * * * *
2 * + + * * * * *
3 * + * * * * * *
4 * + * * * * * *
5 * + * * * * * *
6 * + * * * * * *
7 * + + * * * * *
8 * E * * * * * *
9 * * * * * * * *

```

(3) 实现功能：根据用户输入数据创建迷宫和设置起始点，寻找求解路径并输出。

2、概要设计：

(1) 数据结构：

- 坐标结构体，简单的记录坐标的横纵坐标。
- 栈的元素类型，记录该坐标是路径上的序、坐标位置、下一个要探索的方向。
- 包含栈底栈顶元素的指针，已经当前栈的空间大小（以栈元素为单位）

```

typedef struct{
    int i;
    int j;
}PosType; //通道块在迷宫中的坐标位置

typedef struct{
    int ord; //通道在路径上的“序号”
    PosType seat; //通道块在迷宫中的坐标位置
    int di; //从此通道块走向下一通道块的“方向”
}SElemType; //栈的元素类型

typedef struct{
    SElemType *base; //在栈构造之前和销毁之后，base值为NULL
    SElemType *top; //栈顶指针
    int stacksize; //当前已分配的存储空间，以元素为单位
}SqStack; //栈

```

(2) 主函数流程：

- 输入迷宫的大小、创建并打印迷宫。
- 输入起始点，寻找出路并打印路径。

(3) 各程序模块之间的调用关系

```

+ SqStack InitStack(SqStack S) { //创建栈
+ int IsEmpty(SqStack S) { //若栈不空，则删除s的栈顶元素，用e返回其值，并返回OK，否则返回ERROR
+ void DestroyStack(SqStack S) { //销毁栈
+ SqStack Push(SqStack S, SElemType e) { //插入元素e为新的栈顶元素
+ SqStack Pop(SqStack S) { //出栈
+ SElemType GetTop(SqStack S) { //返回栈顶元素
+ void Create_Maze(int **maze, int line, int row) { //创建迷宫，随机设置障碍物
+ void Print_Maze(int **maze, int line, int row) { //打印迷宫
+ PosType NextPos(PosType pos, int di) { //下一个要走的位置
+ void FindOut(int **maze, PosType Start, PosType End) { //寻找出路

```

其中 Create_Maze 函数的功能是设置障碍，障碍物随机设置（调用随机函数），并给外围设置围墙；Print_Maze 函数的打印功能兼容路径输出的打印，通过控制 maze 数组的值来判断该格是空、障碍还是通路。FindOut 函数除了寻找出路，还会调用 FindOut 函数打印路径，函数结束会释放存储路径顺序的栈。

3、详细设计（主要展示 FindOut 函数）

（1）设计思路

```

    设定当前位置的初值为入口位置；
do{
    if(当前位置可通){
        将当前位置插入栈顶； //纳入路径
        if(该位置是出口位置) //求得路径存放在栈中
            结束；
        else
            切换当前位置的冬邻方块为新的当前位置；
    }
    else{
        if(栈不空 且 栈顶位置尚有其他方向未经探索)
            设定新的当前位置为沿顺时针方向旋转找到的栈顶位置的下一相邻块；
        if(栈不空 但 栈顶位置的四周均不可通){
            删去栈顶位置； //从路径中删去改通道块
            if(栈不空)
                重新测试新的栈顶位置，直到找到一个可通的相邻块或出栈至栈空；
        }
    }
}while(栈不空);

```

（当前位置可通的意思是未曾走到过的通道块。）

（2）除了实现寻找出路外，还负责打印路径，存放路径的栈在此创建并销毁。如没出路则输出无解。代码截图如下：

```

void FindOut(int **maze, PosType Start, PosType End) { //寻找出路
    SqStack S;
    S = InitStack(S);

    PosType curpos; //当前位置
    curpos.i = Start.i;
    curpos.j = Start.j;

    SElemType e; //当前通道块

    int curstep = 1; //探索第一步:

    do{
        if(maze[curpos.i][curpos.j] == 1){
            maze[curpos.i][curpos.j] = 2; //表示留下足迹

            e.di = 1;
            e.seat.i = curpos.i;
            e.seat.j = curpos.j;
            e.ord = curstep;

            S = Push(S, e); //加入路径

            if(curpos.i == End.i && curpos.j == End.j) //到达终点
                break;
            curpos = NextPos(curpos, 1);
            curstep++; //探索下一步
        } //if

        else{
            if(!IsEmpty(S)){
                e = GetTop(S); //由于Pop的不完善, 所以这两步加起来才是真正的Pop功能, 下同
                S = Pop(S);
                curstep--;
                while(e.di == 4 && !IsEmpty(S)){
                    maze[e.seat.i][e.seat.j] = 3; //3表示不能通过的标记
                    e = GetTop(S);
                    S = Pop(S);
                    curstep--;
                }
                if(e.di < 4){
                    e.di++;
                    S = Push(S, e);
                    curstep++;
                    curpos = NextPos(e.seat, e.di);
                } //if
            } //if
        } //else
    } while(!IsEmpty(S));
}

```

```

if(IsEmpty(S)){
    maze[Start.i][Start.j] = 1; // 因为无论如何起点都会留下足迹
    printf("\n无解");
} // if
else{//打印路径
    PosType Path[curstep];
    int i;
    while(!IsEmpty(S)){
        e = GetTop(S);
        S = Pop(S);
        Path[e.ord - 1].i = e.seat.i; // ord是从1开始计算的，而数组是从0开始
        Path[e.ord - 1].j = e.seat.j;
    } // while
    for(i = 0; i < curstep - 1; i++) // 最后一个单独打印
        printf("(%d,%d) -> ", Path[i].i, Path[i].j);
    printf("(%d,%d)\n", Path[i].i, Path[i].j);
} // else

DestroyStack(S);
} // FindOut

```

4、调试分析：

(1) 动态开辟存储空间：（二维数组）

由于允许用户输入迷宫的大小和起始点，所以创建迷宫时要动态创建迷宫二维数组，在 C++ 可以直接用 new 关键字简单地完成，但 C 语言不一样。之前只学习了如何动态分配一位数组（指针）的。双指针虽然实际上也是指针，但在开辟空间时却要先对一维指针开辟空间，在对每个一维指针开辟空间。这是在本次实验自己从网上学习到的。

```

// 动态分配一维数组内存
do{//确保开辟成功
    maze = (int**)malloc(sizeof(int*) * LSize);
}while(maze == NULL);
for(i = 0; i < LSize; i++)
    do{
        *(maze + i) = (int*)malloc(sizeof(int) * RSize);
    }while(maze + i == NULL);

```

(2) 容错性考虑：

- 若内存分配失败，则要重新分配。如上图所示。
- 若用户输入的起始点已经有障碍或者说在迷宫范围外，则应当提示输入错误并重新输入。

```

// 输入起点 终点
printf("please input the Start:");
scanf("%d %d", &Si, &Sj);
while(Si > LSize - 1 || Sj > RSize - 1 || maze[Si][Sj] == 0){ // 最后
    printf("Error Input!\nPlease input the Start again:");
    scanf("%d %d", &Si, &Sj);
}
printf("please input the End:");
scanf("%d %d", &Ei, &Ej);
while(Ei > LSize - 1 || Ej > RSize - 1 || maze[Ei][Ej] == 0){
    printf("Error Input!\nPlease input the End again:");
    scanf("%d %d", &Ei, &Ej);
}

```

(3) 调试错误分析

- 判断多条件时各条件的顺序关系？

```

//输入起点 终点
printf("please input the Start:");
scanf("%d %d",&Si,&Sj);
while(Si > LSize - 1 || Sj > RSize - 1 || maze[Si][Sj] == 0){
    printf("Error Input!\nPlease input the Start again:");
    scanf("%d %d",&Si,&Sj);
}

```

注意到 while 里的条件，最后一项表达式不能放到前面，不然会越界。若用户输入的 Si,Sj 不在迷宫范围内，那么进入判断时不先判断是否越界而是直接判断是否存在障碍物，那么就会越界。程序出错。

所以当多个判断条件组合时，要考虑条件的顺序。对于“或”运算，要考虑条件之间可能会有一定的先后顺序和把条件顺序能影响判断语句的效率。

b. 创建迷宫时横必须大于竖？

调试过程发现输入迷宫大小时，若前一个数大于后一个，程序久会出错。定位错误一定是在开辟空间时出问题了，最后发现是开辟二维数组时循环结束条件错了。

```

//动态分配一维数组内存
do{//确保开辟成功
    maze = (int**)malloc(sizeof(int*) * LSize);
}while(maze == NULL);
for(i = 0; i < LSize; i++)
    do{
        *(maze + i) = (int*)malloc(sizeof(int) * RSize);
    }while(maze + i == NULL);

```

在 for 循环里，错误结束条件是 i<RSize，这也能解释为什么若前一个数大于后一个，程序就会出错，因为越界了。

所以对于指针的运用要谨慎，否则容易造成内存越界。同时调试时可以对多组数组进行分析通过统计发现错误。刚开始出现错误时，我想不通，于是不停测试其他迷宫大小样例，最后发现一个规律就是产生错误的样例都是前一个数大于后一个，稍加分析就能立刻定位错误地方。

c. 由于创建栈时不是创建栈结构体指针导致的问题。

在这个函数中会不停调用栈操作函数，会不停修改栈，如果能把栈设成指针，自然不需要每个函数都返回该栈，但是由于栈结构体中含指针，如果把栈定义为指针，分配空间时就要用其他做法，但是没有找到相关的解决办法，同时衡量到，就算不把栈定义为指针，所有栈操作函数返回栈，也不会影响功能，所以最终还是返回栈。

受到影响的栈操作函数是 Pop 函数，Pop 函数的应当返回栈顶元素和栈，由于上述原因，Pop 函数并没有完成该功能。解决办法是，在调用 Pop 函数时，先调用 GetTop 函数，得到栈顶元素，后再调用 Pop 函数，Pop 函数在此实现的是改变栈顶指针而已（如下图）。在 FindOut 函数中需要调用 Pop 函数的地方只有两个，所以这种折中方案还是可以接受的。

同时使得 FindOut 函数还需要承担打印路径的功能，不太符合结构化设计的思想。

```

SqStack Pop(SqStack S) { //出栈
    //由于不懂如何为*s (s结构体里又有指针) 开辟空间, 导致无法设置形参为指针类型
    //而该函数至少要返回栈顶元素, 但是栈s又要返回
    //考虑到返回栈顶元素有专门的函数, 所以该函数返回栈s, 并非真正意义上的出栈函数, 仅仅修改栈顶指针
    if (S.top == S.base)
        exit(0);
    S.top--;
    return S;
} //Pop

else{
    if (!IsEmpty(S)) {
        e = GetTop(S); //由于Pop的不完善, 所以这两步加起来才是真正的Pop功能, 下同
        S = Pop(S);
        curstep--;
    }
}

```

(4) 时空复杂度分析

- void Create_Maze(int **maze, int line, int row); //创建迷宫, 随机设置障碍物
O (line * row + (line + row) * 2) //调用随机函数的次数是: (line + row) * 2
- void Print_Maze(int **maze, int line, int row); //打印迷宫
O (line * row + row)
- 整个程序的空间复杂度分析: 使用内存较大的变量有栈和迷宫, 均采用动态内存分配, 使用完后及时释放。

四、实验结果:

1、有解情况:

```

F:\数据结构2016年春季学期\实验\【实验二】迷宫问题\Maze.exe
Please input the MazeSize"(LSize RSize)":16 12
 0 1 2 3 4 5 6 7 8 9 10 11
0 * * * * * * * * * *
1 *      *      *
2 *      *      * *
3 *      *      *
4 *      *      * *
5 *      *      * *
6 *      *      *
7 *      *      * *
8 * *      * * *      *
9 *      *      * * *
10 *      *      *
11 * *      *
12 * * * *      * *
13 *      *      *
14 *      *      *
15 * * * * * * * * * *

please input the Start:1 1
please input the End:11 9

```

```

F:\数据结构2016年春季学期\实验\【实验二】迷宫问题\Maze.exe
(1,1) -> (1,2) -> (2,2) -> (3,2) -> (3,3) -> (3,4) -> (3,5) -> (3,6) -> (3,7) ->
(2,7) -> (2,8) -> (2,9) -> (3,9) -> (4,9) -> (5,9) -> (6,9) -> (7,9) -> (8,9) ->
(8,8) -> (7,8) -> (6,8) -> (6,7) -> (6,6) -> (7,6) -> (7,5) -> (7,4) -> (8,4)
-> (9,4) -> (9,5) -> (10,5) -> (10,6) -> (10,7) -> (11,7) -> (11,8) -> (11,9)

'*' is obstacle, '+' is the pass road.
 0 1 2 3 4 5 6 7 8 9 10 11
0 * * * * * * * * * *
1 * S + * * * *
2 * + * * * + + *
3 * + + + + + * + *
4 * * * * * + * *
5 * * * * * + * *
6 * * * * * + *
7 * * * * * + *
8 * * * * * + *
9 * * * * * * *
10 * * * * * *
11 * * * * * E *
12 * * * * * *
13 * * * * * *
14 * * * * * *
15 * * * * * *

搜狗拼音输入法 全 : (0x0) execution time : 25.608 s

```

2、无解情况：

```

F:\数据结构2016年春季学期\实验\【实验二】迷宫问题\Maze.exe
Please input the MazeSize"(LSize RSize)":12 16
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 * * * * * * * * * * * *
1 * * * * * * * * * * *
2 * * * * * * * * * *
3 * * * * * * * * * *
4 * * * * * * * * * *
5 * * * * * * * * * *
6 * * * * * * * * * *
7 * * * * * * * * * *
8 * * * * * * * * * *
9 * * * * * * * * * *
10 * * * * * * * * * *
11 * * * * * * * * * *

please input the Start:4 14
please input the End:7 5

```



```
F:\数据结构2016年春季学期\实验\【实验二】迷宫问题\Maze.exe
please input the Start:4 14
please input the End:7 5

无解
'*' is obstacle, '+' is the pass road.
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 * * * * * * * * * * * * * *
1 *      *      *      *      *
2 * * *      *      *
3 * * *      *      *
4 *      *      *      * S *
5 *      *      *      *      *
6 *      *      *      *      *
7 * *      E      *      *      *
8 *      *      *      *      *
9 *      *      *      *      *
10 *      *      *      *      *
11 * * * * * * * * * * * * * *

Process returned 0 (0x0)    execution time : 23.880 s
Press any key to continue.

搜狗拼音输入法 全 :
```

3、起始点输入错误情况:

```
F:\数据结构2016年春季学期\实验\【实验二】迷宫问题\Maze.exe
Please input the MazeSize"(LSize RSize)":12 12
 0 1 2 3 4 5 6 7 8 9 10 11
0 * * * * * * * * * *
1 * * *      *      *
2 *      * * *      *
3 *      *      *
4 * *      *      * *
5 *      *      *      *
6 *      * *      *
7 * *      *      * *
8 * *      *      *
9 *      *      *      *
10 *      *      * * *
11 * * * * * * * * *

please input the Start:1 1
Error Input!
Please input the Start again:3 1
please input the End:9 8
Error Input!
Please input the End again:9 7
```

```
F:\数据结构2016年春季学期\实验\【实验二】迷宫问题\Maze.exe
please input the End:9 8
Error Input!
Please input the End again:9 7

(3,1) -> (3,2) -> (4,2) -> (5,2) -> (6,2) -> (6,3) -> (7,3) -> (7,4) -> (7,5) ->
(8,5) -> (8,6) -> (8,7) -> (9,7)

' * ' is obstacle, ' + ' is the pass road.
 0 1 2 3 4 5 6 7 8 9 10 11
0 * * * * * * * * * *
1 * * * * * * * * *
2 * * * * * * * *
3 * S + * * * * *
4 * * + * * * * *
5 * + * * * * *
6 * + + * * * *
7 * * + + + * *
8 * * * + + + *
9 * * * * E * *
10 * * * * *
11 * * * * * * * *

Process returned 0 (0x0)   execution time : 33.380 s
Press any key to continue.
搜狗拼音输入法 全 :
```

五、实验总结:

- 1、迷宫问题和背包问题是类似的，通过栈结构来实现穷举。由于栈的结构特点是 FILO，使用时要注意栈顶元素和顶指针的变化。
- 2、学会了如何对双指针（二维数组）动态分为内存空间，多维数组的开辟同理。
- 3、强化了数组也是指针的概念。在程序中迷宫的二维数组是贯穿这个程序的，然而传递二维数组时，必须要至少给出第二维的维数，但是数组大小是动态分配的，编译时是未知的，这时候就需要利用对应的双指针解决问题。通过这次实验更加清除指针、一维数组、双指针、二维数组之间的联系和区别。
- 4、迷宫问题的解决一般不止一条通路，但是本程序只能提供一种解决方案。实际上，只需要通过修改探索方向的先后就能得到另一种解决方案。