

3.3 栈和队列求解迷宫问题

栈和队列都是存放多个数据的容器。通常用于存放临时数据：

- 如果先放入的数据先处理，则使用队列。
- 如果后放入的数据先处理，则使用栈。

1、用栈求解迷宫问题

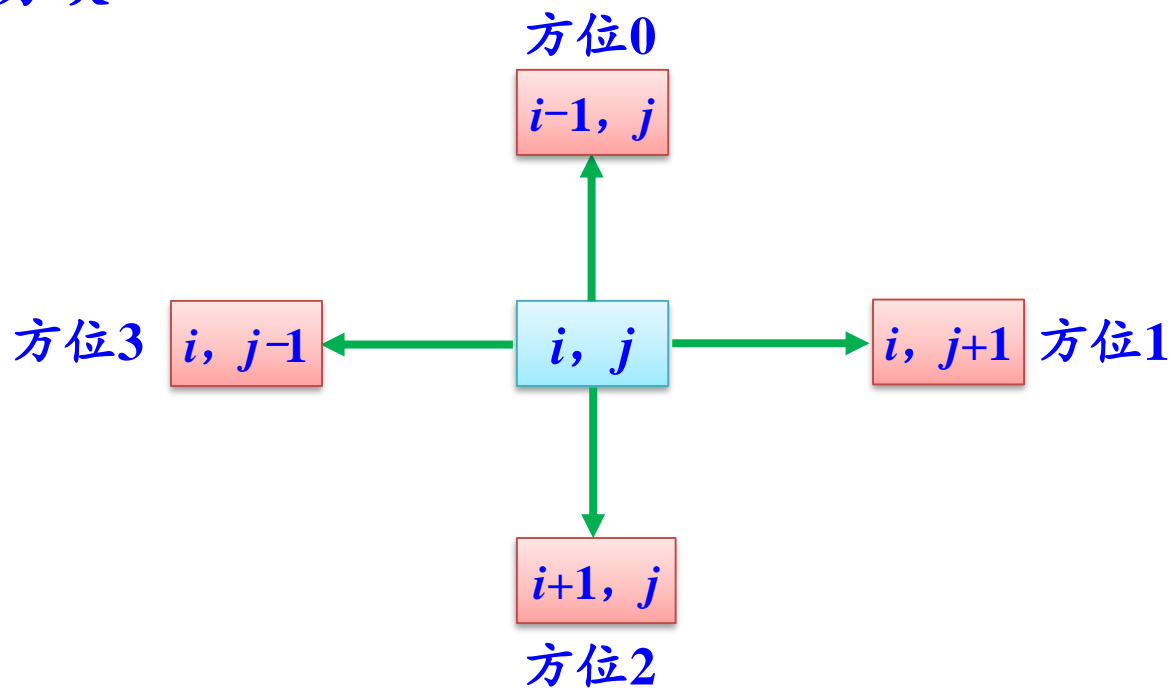


问题描述

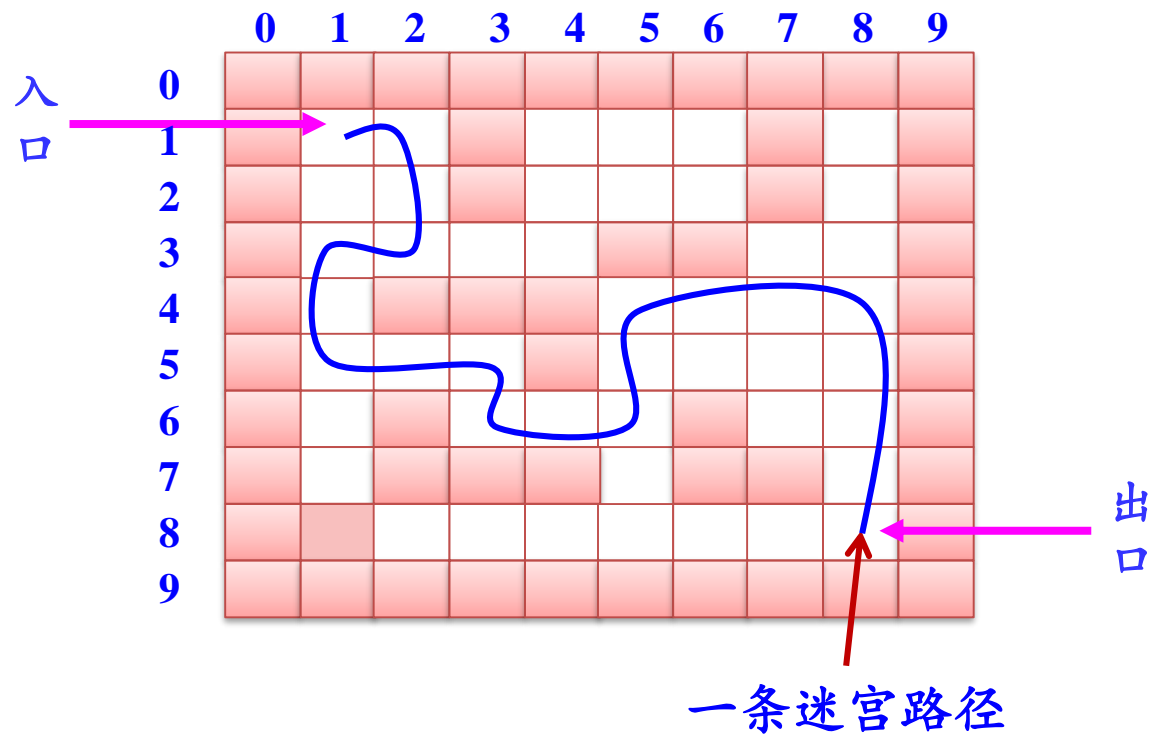
给定一个 $M \times N$ 的迷宫图、入口与出口、行走规则。求一条从指定入口到出口的路径。

所求路径必须是简单路径，即路径不重复。

行走规则：上、下、左、右相邻方块行走。其中 (i, j) 表示一个方块



例如， $M=8$ ， $N=8$ ，图中的每个方块，用空白表示通道，用阴影表示障碍物。为了算法方便，一般在迷宫外围加上了一条围墙。






数据组织

设置一个迷宫数组mg，其中每个元素表示一个方块的状态，为0时表示对应方块是通道，为1时表示对应方块不可走。

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

mg[2][1]=0

mg[2][7]=1



	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

```
int mg[M+2][N+2]=
```

```
{
  {1, 1,1,1,1,1,1,1,1, 1},
  {1, 0,0,1,0,0,0,1,0, 1},
  {1, 0,0,1,0,0,0,1,0, 1},
  {1, 0,0,0,0,1,1,0,0, 1},
  {1, 0,1,1,1,0,0,0,0, 1},
  {1, 0,0,0,1,0,0,0,0, 1},
  {1, 0,1,0,0,0,1,0,0, 1},
  {1, 0,1,1,1,0,1,1,0, 1},
  {1, 1,0,0,0,0,0,0,0, 1},
  {1, 1,1,1,1,1,1,1,1, 1}
};
```

M×N

在算法中用到的栈采用顺序栈存储结构，即将栈定义为：

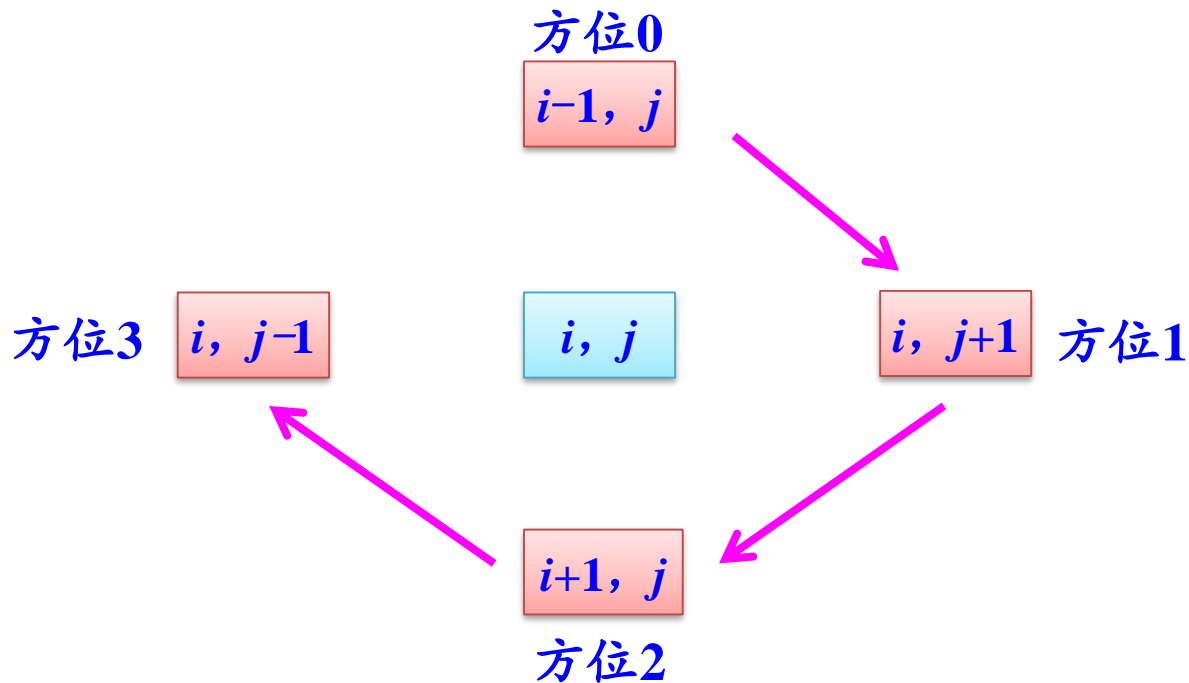
```
typedef struct
{   int i;           //当前方块的行号
    int j;           //当前方块的列号
    int di;          //di是下一可走相邻方位的方位号
} Box;
typedef struct
{   Box data[MaxSize];
    int top;          //栈顶指针
} StType;            //定义顺序栈类型
```





算法设计

试探顺序：从方位0开始，顺时针方向



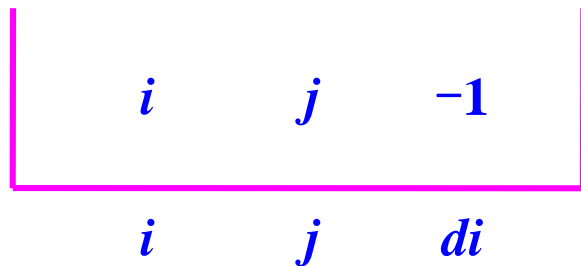
初始时，入口 (i, j) 作为当前方块。

i, j



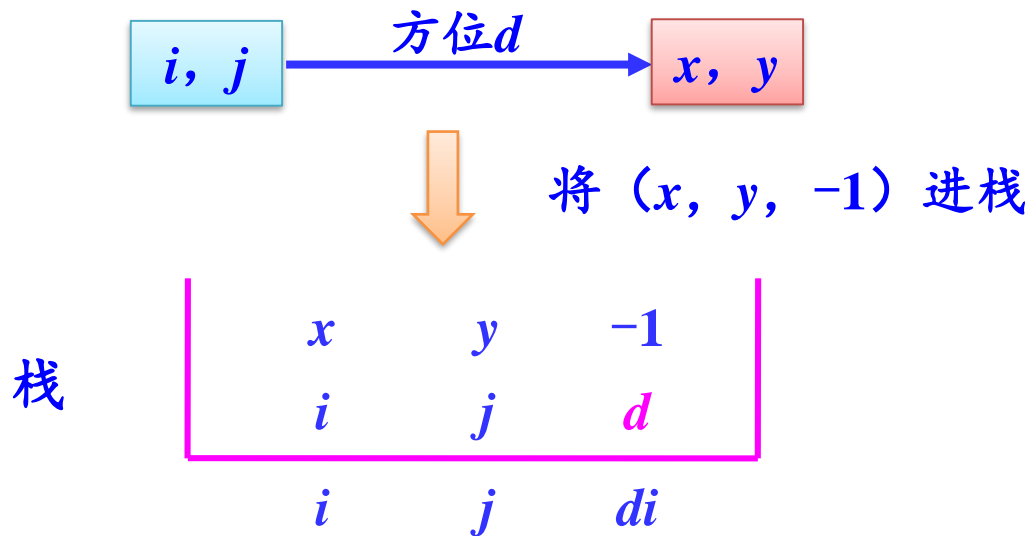
将 $(i, j, -1)$ 进栈

栈

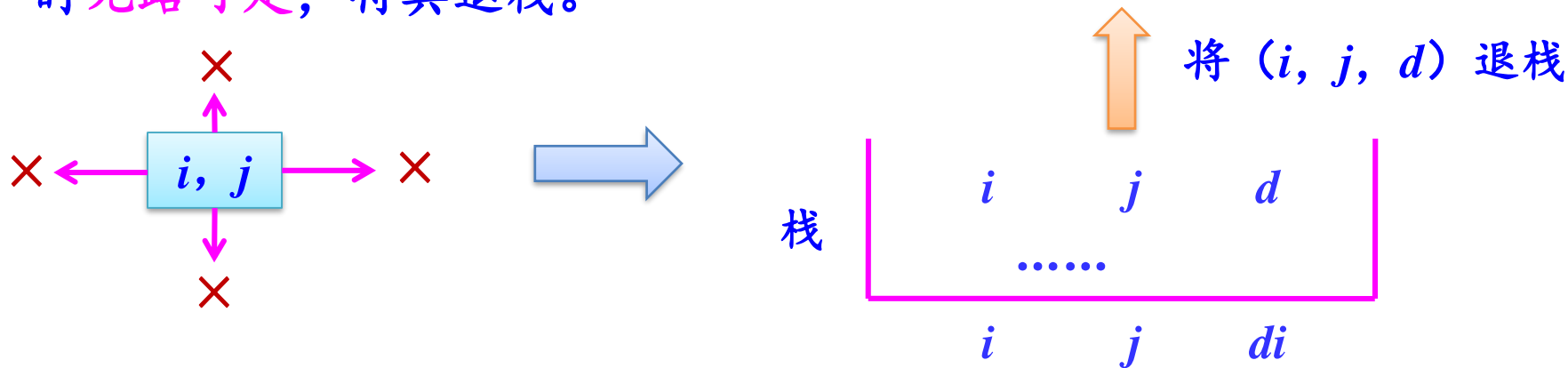


所有走过的方块都会进栈！

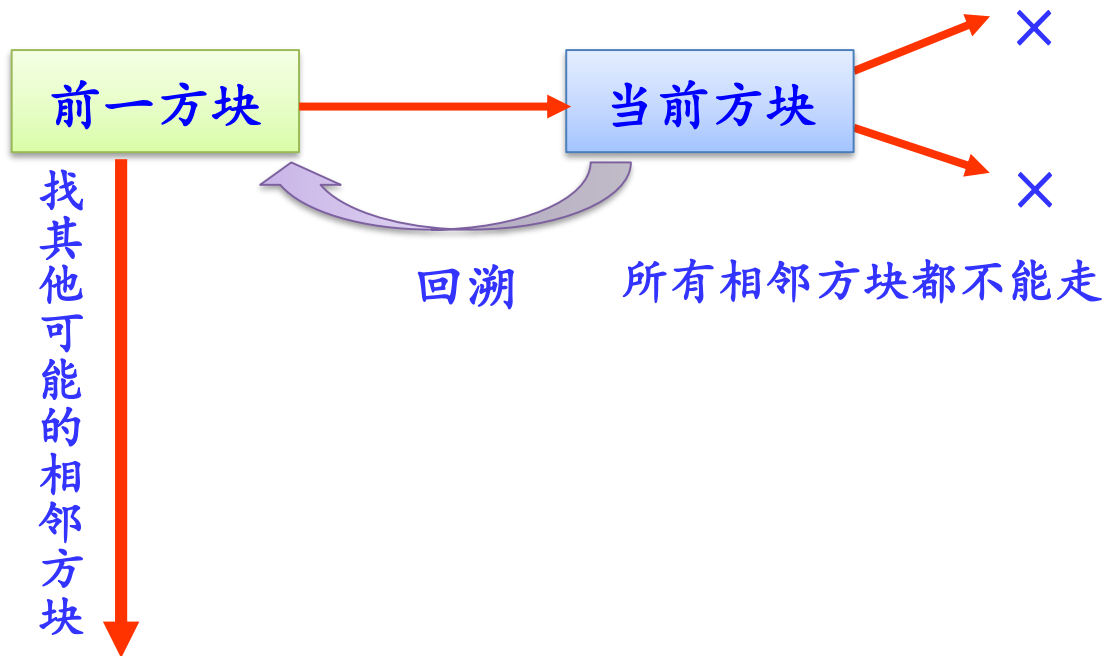
如果一个当前方块 (i, j) 找到一个相邻可走方块 (x, y) ，就继续从 (x, y) 走下去。



如果一个当前方块 (i, j) 没有找到任何相邻可走方块，表示此时无路可走，将其退栈。



求解迷宫路径的过程：



用栈求一条迷宫路径的算法: $(xi, yi) \Rightarrow (xe, ye)$

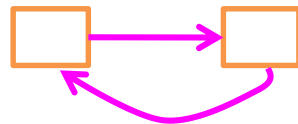
```
bool mgpath(int xi,int yi,int xe,int ye)
{
    int i,j,k,di,find;
    StType st;st.top=-1;           //定义栈st并初始化
    st.top++;                       //初始方块进栈
    st.data[st.top].i=xi; st.data[st.top].j=yi;
    st.data[st.top].di=-1;  mg[xi][yi]=-1;
```

	0	1	2	3	4	5
0						
1		●				
2						
3						
4					●	
5						

1	1	-1
<i>i</i>	<i>j</i>	<i>di</i>

一个栈

为了避免重复, 当一个方块进栈时, 将迷宫值改为-1

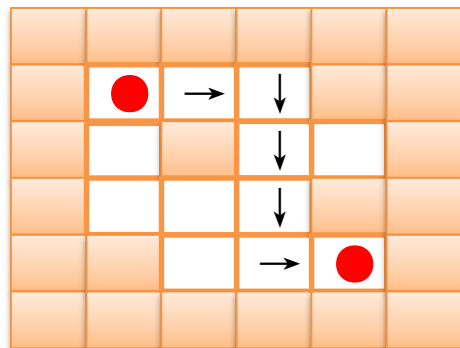


```

while (st.top>-1)                //栈不空时循环
{
    i=st.data[st.top].i;j=st.data[st.top].j;
    di=st.data[st.top].di;       //取栈顶方块
    if (i==xe && j==ye)          //找到了出口,输出路径
    {
        printf("迷宫路径如下:\n");
        for (k=0;k<=st.top;k++)
        {
            printf("\t(%d,%d)",st.data[k].i,st.data[k].j);
            if ((k+1)%5==0) //每输出每5个方块后换一行
                printf("\n");
        }
        printf("\n");
        return true;             //找到一条路径后返回true
    }
}

```

4	4	-1
4	3	1
3	3	1
2	3	2
1	3	2
1	2	1
1	1	1
<i>i</i>	<i>j</i>	<i>di</i>

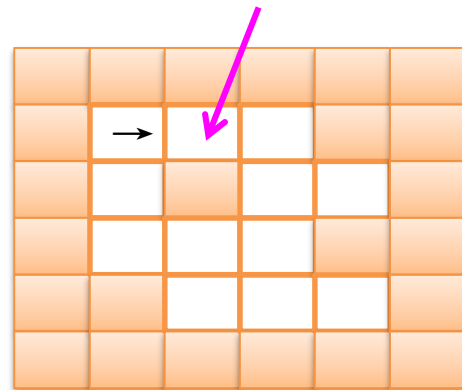


```

find=0;
while (di<4 && find==0) //找下一个可走方块
{
    di++;
    switch(di)
    {
        case 0:i=st.data[st.top].i-1; j=st.data[st.top].j;
            break;
        case 1:i=st.data[st.top].i; j=st.data[st.top].j+1;
            break;
        case 2:i=st.data[st.top].i+1; j=st.data[st.top].j;
            break;
        case 3:i=st.data[st.top].i, j=st.data[st.top].j-1;
            break;
    }
    if (mg[i][j]==0) find=1;
    //找到了下一个可走相邻方块(i, j)
}

```

从入口 (1,1) 出发找到一个可走方块 (1, 2)

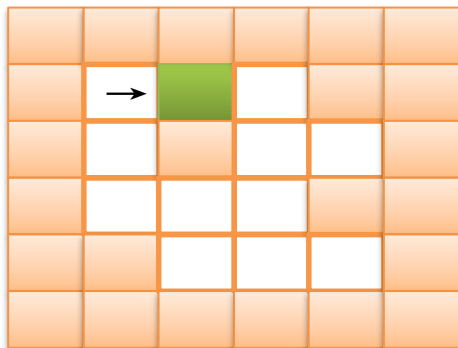


```

if (find==1)                                //找到了下一个可走方块
{
    st.data[st.top].di=di;                  //修改原栈顶元素的di值
    st.top++;                               //下一个可走方块进栈
    st.data[st.top].i=i; st.data[st.top].j=j;
    st.data[st.top].di=-1;
    mg[i][j]=-1;                            //避免重复走到该方块
}

```

从入口 (1,1) 出发找到一个可走方块
 (1, 2) : 将 (1, 2, -1) 进栈



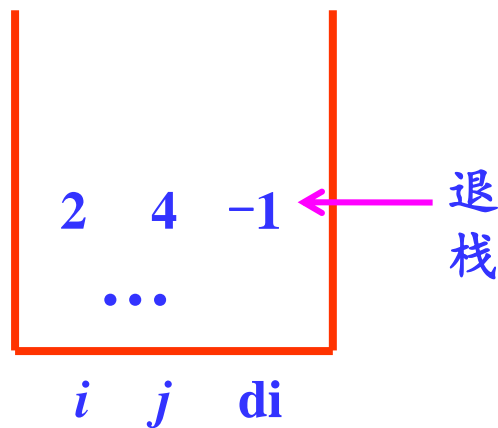
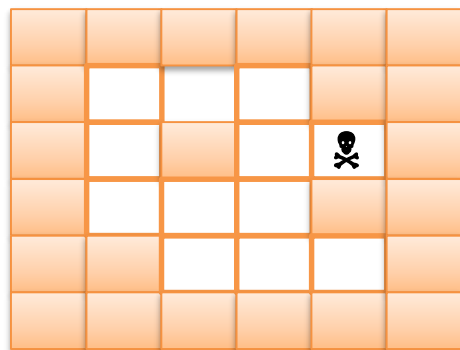
1	2	-1
1	1	1
<i>i</i>	<i>j</i>	di


```

else                                     //没有路径可走,则退栈
{   mg[st.data[st.top].i][st.data[st.top].j]=0;
    //让该位置变为其他路径可走方块
    st.top--;
    //将该方块退栈
}
}
return false;                          //表示没有可走路径,返回false
}

```

(2,4) 方块没有通路





设计求解程序

建立如下主函数调用上述算法：

```
void main()
{   if (!mgpath(1,1,M,N))
        printf("该迷宫问题没有解!");
}
```

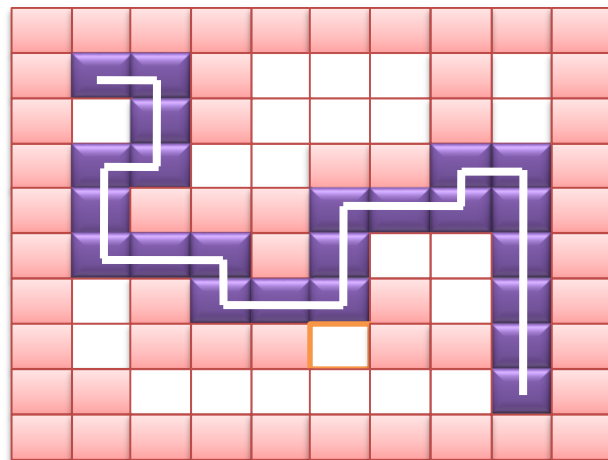


运行结果

对于图3.7的迷宫，求解结果如下：

迷宫路径如下：

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



显然，这个解不是最优解，即不是最短路径。

2、用队列求解迷宫问题



数据组织

使用一个队列qu记录走过的方块，该队列的结构如下：

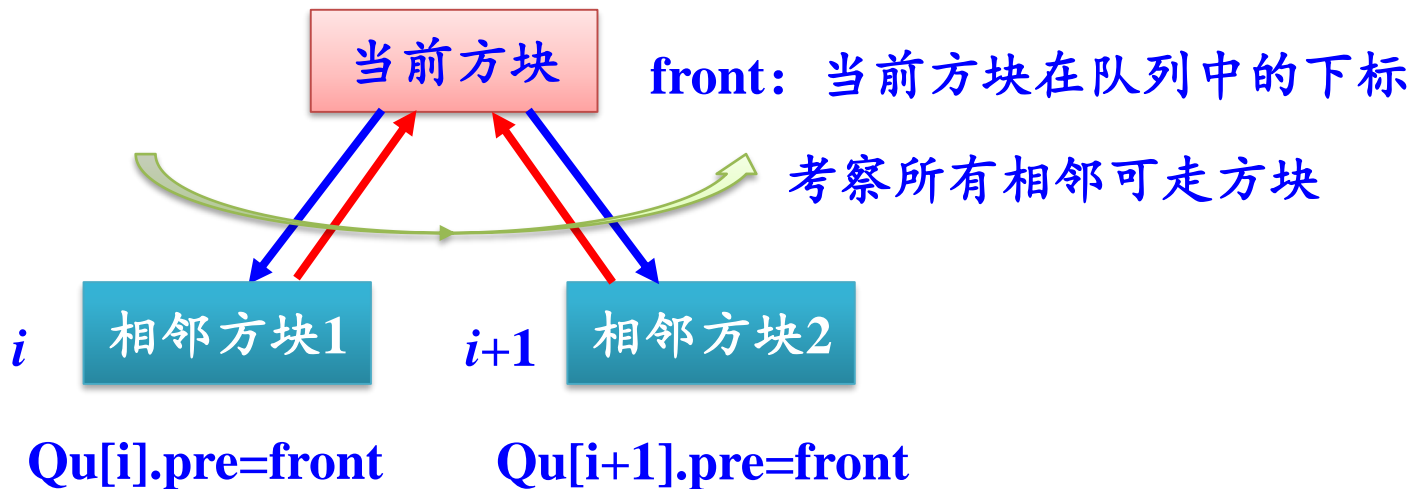
```
typedef struct
{
    int i,j;           //方块的位置
    int pre            //本路径中上一方块在队列中的下标
} Box;                //方块类型
typedef struct
{
    Box data[MaxSize];
    int front,rear;    //队头指针和队尾指针
} QuType;             //定义顺序队类型
```

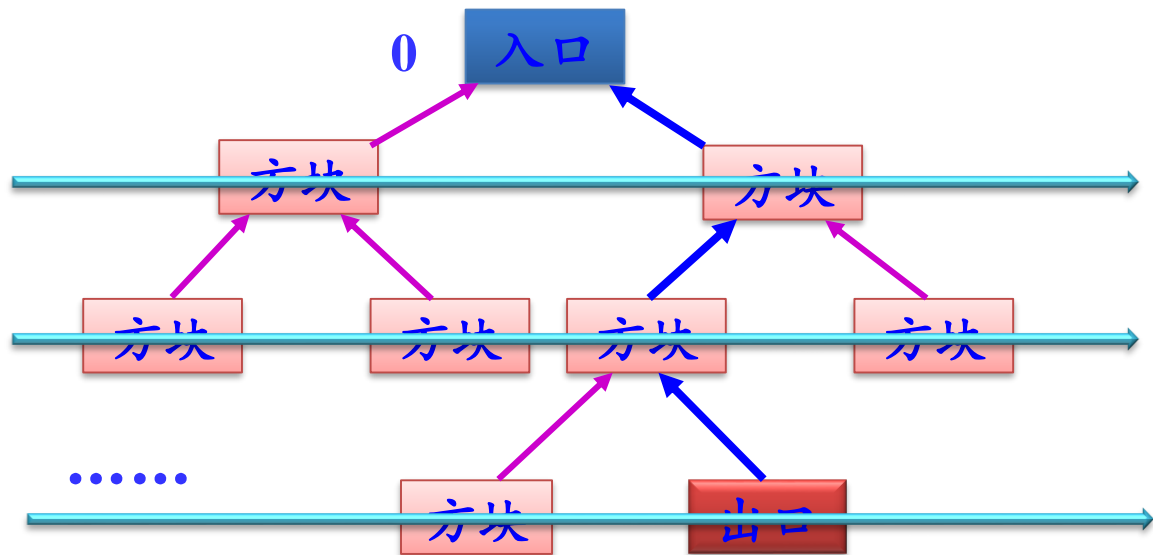
这里使用的队列qu不是环形队列（因为要利用出队的元素找路径），因此在出队时，不会将出队元素真正从队列中删除，因为要利用它输出路径。



算法设计

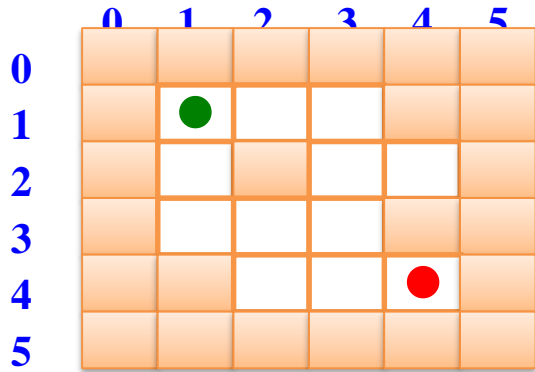
首先将入口进队。出队一个方块，考察如下：





所有搜索过的方块都在队列中。

最后通过队列找出从出口 \Rightarrow 入口的一条迷宫路径。



```
int mg[M+2][N+2]=    //M=4,N=4
{ {1, 1, 1, 1, 1, 1},
  {1, 0, 0, 0, 1, 1},
  {1, 0, 1, 0, 0, 1},
  {1, 0, 0, 0, 1, 1},
  {1, 1, 0, 0, 0, 1},
  {1, 1, 1, 1, 1, 1} };
```

迷宫路径:

(4,4)

(4,3)

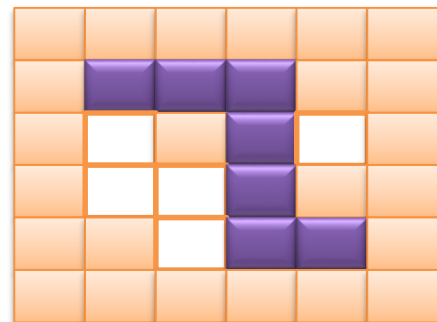
(3,3)

(2,3)

(1,3)

(1,2)

(1,1)



入口

0:(1,1) -1

1:(1,2) 0

2:(2,1) 0

3:(1,3) 1

4:(3,1) 2

5:(2,3) 3

6:(3,2) 4

7:(2,4) 5

8:(3,3) 5

9:(4,2) 6

10:(4,3) 8

出口

11:(4,4) 10

用队列求一条迷宫路径的算法： $(x_i, y_i) \Rightarrow (x_e, y_e)$

```
bool mgpath1(int xi,int yi,int xe,int ye)
//搜索路径为:(xi,yi)->(xe,ye)
{   int i,j,find=0,di;
    QuType qu;                //定义顺序队
    qu.front=qu.rear=-1;
    qu.rear++;
    qu.data[qu.rear].i=xi; qu.data[qu.rear].j=yi;    //(xi,yi)进队
    qu.data[qu.rear].pre=-1;
    mg[xi][yi]=-1; //将其赋值-1,以避免回过来重复搜索
```



```
while (qu.front!=qu.rear && !find)           //队列不空循环
{
    qu.front++;                             //出队
    i=qu.data[qu.front].i; j=qu.data[qu.front].j;
    if (i==xe && j==ye)                     //找到了出口,输出路径
    {
        find=1;
        print(qu,qu.front);
        return true;
    }
}
```

```

for (di=0;di<4;di++)
{
    switch(di)
    {
        case 0: i=qu.data[qu.front].i-1;
                j=qu.data[qu.front].j; break;
        case 1: i=qu.data[qu.front].i;
                j=qu.data[qu.front].j+1; break;
        case 2: i=qu.data[qu.front].i+1;
                j=qu.data[qu.front].j; break;
        case 3: i=qu.data[qu.front].i;
                j=qu.data[qu.front].j-1; break;
    }
    if (mg[i][j]==0)
    {
        qu.rear++;           //将该相邻方块插入到队列中
        qu.data[qu.rear].i=i; qu.data[qu.rear].j=j;
        qu.data[qu.rear].pre=qu.front; mg[i][j]=-1;
    }
}
return false;              //未找到一条路径时返回false
}

```

把当前出队方块的每个相邻可走的方块进队

对于图3.7的迷宫，求解(1,1)→(8,8)时队列qu中结果如下：

下标	i	j	pre
0	1	1	-1
1	1	2	0
2	2	1	0
3	2	2	1
4	3	1	2
5	3	2	3
6	4	1	4
7	3	3	5
8	5	1	6
9	3	4	7
10	5	2	8
11	6	1	8
12	2	4	9
13	5	3	10
14	7	1	11

下标	i	j	pre
15	1	4	12
16	2	5	12
17	6	3	13
18	1	5	15
19	2	6	16
20	6	4	17
21	1	6	18
22	6	5	20
23	5	5	22
24	7	5	22
25	4	5	23
26	5	6	23
27	8	5	24
28	4	6	25
29	5	7	26

下标	i	j	pre
30	8	6	27
31	8	4	27
32	4	7	28
33	5	8	29
34	6	7	29
35	8	7	30
36	8	3	31
37	3	7	32
38	4	8	32
39	6	8	33
40	8	8	35

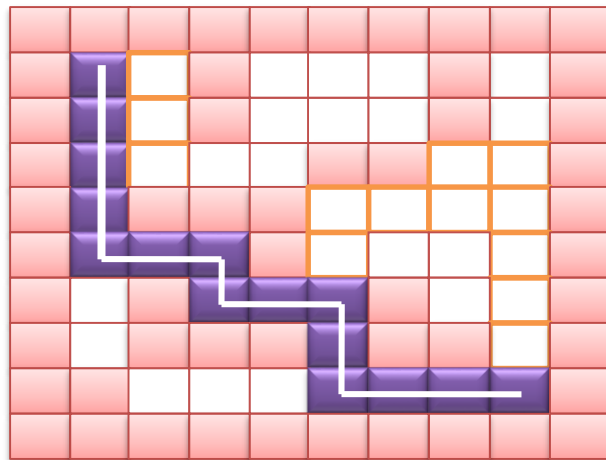


运行结果

对于图3.7的迷宫，求解结果如下：

迷宫路径如下：

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



显然，这个解是最优解，即是最短路径。



思考题：

用队列和用栈求解迷宫问题有什么不同？

——本章完——