1.5 回溯算法

主讲老师：党东

【例1】走迷宫 --1487

【问题描述】

有一个m*n格的迷宫(表示有m行、n列)，其中有可走的也有不可走的，如果用1表示可以走，0表示不可以走，文件读入这m*n个数据和起始点、结束点(起始点和结束点都是用两个数据来描述的，分别表示这个点的行号和列号)。现在要你编程找出所有可行的道路，要求所走的路中没有重复的点，走时只能是上下左右四个方向。如果一条路都不可行，则输出相应信息(用-1表示无路)。

【输入文件】

第一行是两个数m，n，接下来是m行n列由1和0组成的数据，最后两行是起始点和结束点(m,n>1且 m,n<15)。

【输出文件】

输出所有可能路径条数，如果没有一条可行的路则输出-1。

【输入样例】

```
5 6
1 0 0 1 0 1
1 1 1 1 1 1
0 0 1 1 1 0
1 1 1 1 1 0
1 1 1 0 1 1
1 1
5 6
```

【输出样例】12

【例1】走迷宫 --1487

【图的输入】

```
char map[100][100];
cin>>n>>m;
for(i=1;i<=n;i++)
  for(j=1;j<=m;j++)
    cin>>map[i][j];
```

分析：

1.Map[n][m]表示迷宫

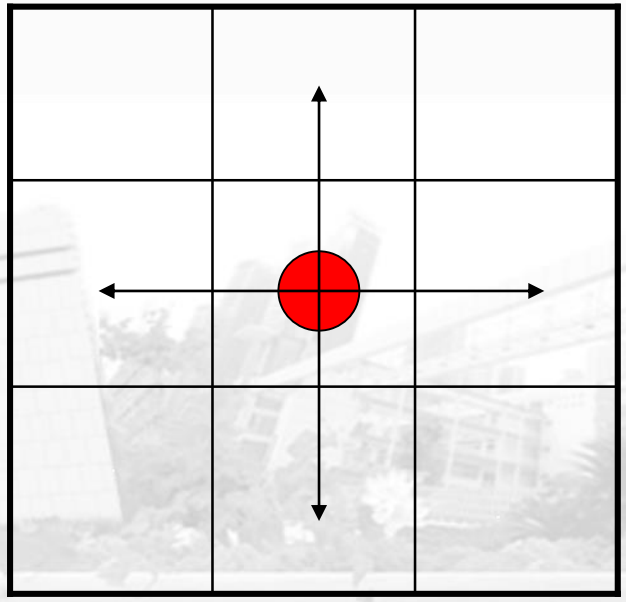
2.增量和方向表示

dx=(1,-1,0 , 0)

dy=(0, 0 ,1,-1)

【控制方向】

```
for(i=0;i<4;i++)
    {
        tx=x+dx[i];
        ty=y+dy[i];
    }
```



【例1】走迷宫 --1487

0	1	2	3	4	5	6
1	1	0	0	1	0	1
2	1	1	1	1	1	1
3	0	0	1	1	1	0
4	1	1	1	1	1	0
5	1	1	1	0	1	1

分析：

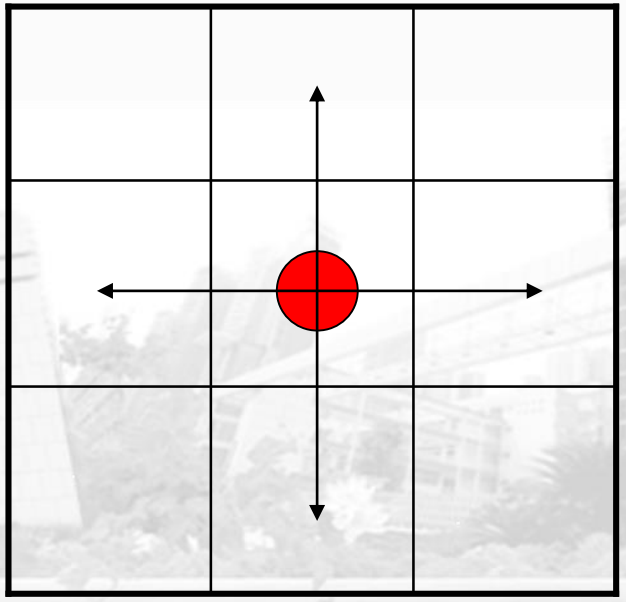
1.Map[n][m]表示迷宫

2.(x1,y1)入口；(x2,y2)出口

3.增量和方向表示

dx=(1,-1,0 , 0)

dy=(0, 0 ,1,-1)



【例1】走迷宫 --1487

0	1	2	3	4	5	6
1	1	0	0	1	0	1
2	1	1	1	1	1	1
3	0	0	1	1	1	0
4	1	1	1	1	1	0
5	1	1	1	0	1	1

【约束分析】

1.将要走的位置可以走

即map[x+x[i]][y+y[i]]==1

2.不能走出边界

即下一个位置(x+x[i], y+y[i])未越界

3.已走过的位置不在走

即map[x][y]=0;

【例1】走迷宫 --1487

【参考代码】

```
int map[16][16];
int m,n;
int ta,tb; //代表出口
long long count;
```

【例1】走迷宫 --1487


```
int x[4]={0,1,0,-1};
int y[4]={1,0,-1,0};
void f(int a,int b)
{
        if(a==ta&&b==tb) count++;
        for(int i=0;i<=3;i++)   //四个方向走
        {
                if(map[a+x[i]][b+y[i]]&&     //下一个位置可走
                   a+x[i]>=1&&a+x[i]<=m&&b+y[i]>=1&&b+y[i]<=n ) //是否越界
                {
                        map[a+x[i]][b+y[i]]=0; //走之前标记
                        f(a+x[i],b+y[i]); //走下一步
                        map[a+x[i]][b+y[i]]=1; //恢复状态
                }
        }
}
```

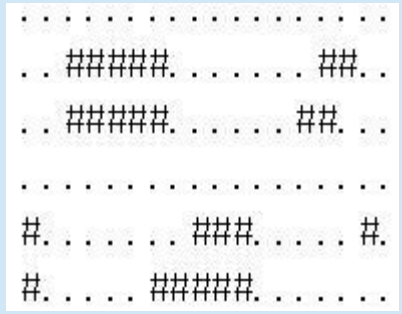
【例1】走迷宫 --1487


```
int main()
{
        cin>>m>>n;
        for(int i=1;i<=m;i++)
                for(int j=1;j<=n;j++)
                        cin>>map[i][j];
        int i1,j1;
        cin>>i1>>j1>>ta>>tb;
        map[i1][j1]=0; //走之前标记
        f(i1,j1);
        if(count) cout<<count;
        else cout<<"-1";
        return 0;
}
```

【例2】卫星照片 --1128

【问题描述】

农夫 John 正在研究他的农场的卫星照片.照片为一个R (1<= R <= 75) 行C (1<=C<=75) 列的字符矩阵表示。如下图:



图上的一块相连通的 "#" 表示一群奶牛或一个房间, 两个子"#" 连通的意思是说左右或上下相连。而下面的两块则是分开的:

John现在根据卫星照片上的的这些"#"块的形状来判断哪些是牛群，哪些是房间。如果矩形内只有‘#’，则是房间。其它的则认为都是牛群。在第一个图中,有三个房间 (2x1, 2x5, and 1x1)和2群牛。

根据输入文件的数据，统计出房间数和牛群数，数据中牛群不会包围另一个牛群或房间。

【例2】卫星照片 --1128

【输入文件】

第一行,两个整数: R 和 C.

第2..R+1行: 第 i+1 行表示照片的第 i 行情况，由 C 字符组成。

【输出文件】

第一行: 房间数。

第二行: 牛群数。

【输入文件】

```
5 8
######..#
######.##
......#.
.###...#
.###..##
```

【输出文件】

```
2
2
```

【例2】卫星照片 --1128

【算法分析】

- 根据题意可以看出，如果一个"#"块形状的边是水平或垂直的矩形，则是房间，其它的则认为都是牛群。在第一个图中，有三个房间（2*1，2*5和1*1）和2群牛。
- 请根据输入文件中的数据，统计出房间数和牛群数。
- 题目数据中牛群不会包围另一个牛群或房间。
- 连通块判断：方格<=75*75个点。
- 判断连通块是否房间：
  1）左上角(X_1,Y_1)
  2）右下角(X_2,Y_2)
  3）数量=(X_2-X_1+1)*(Y_2-Y_1+1)

```
....................
..######......##..
..######......##...
....................
#.......###.....#.
#.....#####.......
```

【例2】卫星照片 --1128

【图的输入】

```
char map[100][100];
cin>>n>>m;
for(i=1;i<=n;i++)
  for(j=1;j<=m;j++)
    cin>>map[i][j];
```

分析：

1.Map[n][m]表示迷宫

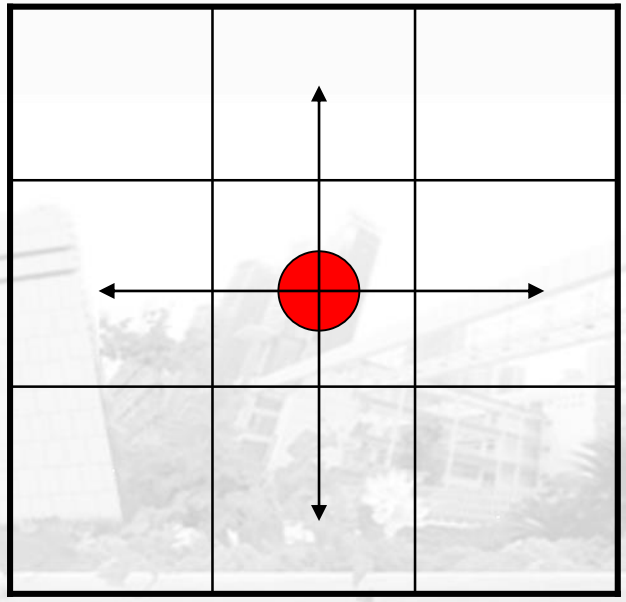
2.增量和方向表示

dx=(1,-1,0 , 0)

dy=(0, 0 ,1,-1)

【控制方向】

```
for(i=0;i<4;i++)
    {
        tx=x+dx[i];
        ty=y+dy[i];
    }
```



【例2】卫星照片 --1128

【约束分析】

1.将要走的位置可以走

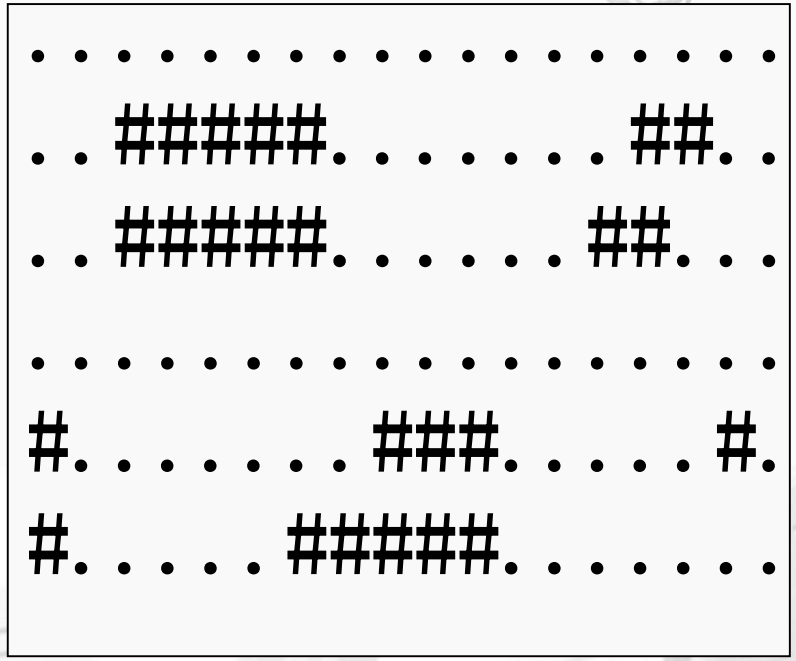
即map[x+x[i]][y+y[i]]== "#"

2.不能走出边界

即下一个位置(x+x[i], y+y[i])未越界

3.已走过的位置不在走

即v[x][y]=1;



【例2】卫星照片 --1128

【参考程序】

```
void Solve()
{    int i,j,Room=0,Cattle=0;
     for(i=1;i<=n;i++)
         for(j=1;j<=m;j++)
             if(!v[i][j]&&map[i][j]=='#')
             {   MaX=0;MaY=0;MiX=n;MiY=m;Total=0;
                  Dfs(i,j);
                  if((MaX-MiX+1)*(MaY-MiY+1)==Total)Room++;
                                  else Cattle++;
             }
       cout<<Room<<endl<<Cattle;
}
```

【例2】卫星照片 --1128


```
void Dfs(int x,int y)
{    int i,tx,ty;
     Total++;
     if(x>MaX)MaX=x;
     if(y>MaY)MaY=y;
     if(x<MiX)MiX=x;
     if(y<MiY)MiY=y;
     v[x][y]=1;
     for(i=0;i<4;i++)
     {     tx=x+dx[i];
           ty=y+dy[i];
           if(tx>=1&&tx<=n&&ty>=1&&ty<=m&&       //是否越界
                 map[tx][ty]=='#'&&!v[tx][ty])Dfs(tx,ty);
      }
}
```