

# 递归 — 小游戏

郭 炜 刘家瑛



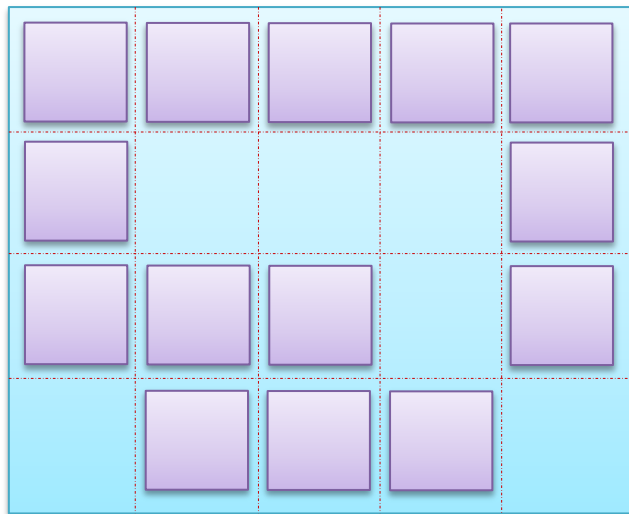
北京大学



# 小游戏

## 问题描述

- 一天早上,你起床的时候想“我编程序这么牛,为什么不能靠这个赚点小钱呢?”因此你决定编写一个小游戏
- 游戏在一个分割成  $w * h$  个正方格子的矩形板上进行
- 每个正方格子上可以有一张游戏卡片,当然也可以没有





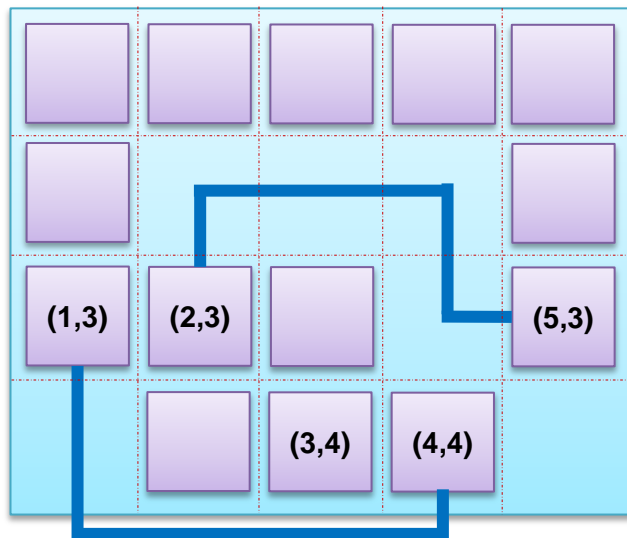
# 小游戏

## 问题描述

- 当下面的情况满足时,  
认为两个游戏卡片之间有一条路径相连:
  - 路径只包含水平或者竖直的直线段
  - 路径不能穿过别的游戏卡片
  - 但是允许路径临时的离开矩形板



- 这是一个例子:



- 在 (1,3)和 (4,4)处的游戏卡片是可以相连的
- 而在 (2,3) 和 (3,4) 处的游戏卡是不相连的, 因为连接它们的每条路径都必须穿过别的游戏卡片
- 现在要在小游戏里面判断:  
是否存在一条满足题意的路径能连接给定的两个游戏卡片



## 输入 (1/2)

- 输入包括多组数据: 一个矩形板对应一组数据
- 第一行包括两个整数  $w$  和  $h$  ( $1 \leq w, h \leq 75$ ), 分别表示矩形板的宽度和长度
- 下面的  $h$  行, 每行包括  $w$  个字符, 表示矩形板上的游戏卡片分布情况:
  - 使用 'X' 表示这个地方有一个游戏卡片
  - 使用 空格 表示这个地方没有游戏卡片



## ■ 输入 (2/2)

- 之后每行上包括4个整数:

$x_1, y_1, x_2, y_2$  ( $1 \leq x_1, x_2 \leq w, 1 \leq y_1, y_2 \leq h$ )

- 给出两个卡片在矩形板上的位置

注意: 矩形板左上角的坐标是(1,1)

输入保证这两个游戏卡片所处的位置是不相同的

如果一行上有4个0, 表示这组测试数据的结束

- 如果一行上给出 $w = h = 0$ , 那么表示所有的输入结束了



## 输出

- 对每一个矩形板, 输出一行 “Board #n:”, n是输入数据的编号
- 对每一组需要测试的游戏卡片输出一行. 这一行的开头是 “Pair m: ”, 这里m是测试卡片的编号 ( 对每个矩形板, 编号都从1开始 )
- 如果可以相连, 找到连接这两个卡片的所有路径中包括线段数最少的路径, 输出 “k segments.”  
k是找到的最优路径中包括的线段的数目
- 如果不能相连, 输出 “impossible.”
- 每组数据之后输出一个空行



## ■ 样例输入

```
5 4
X X X X X
X       X
X X X   X
      X X X
2  3  5  3
1  3  4  4
2  3  3  4
0  0  0  0
0  0
```

## ■ 样例输出

Board #1:

Pair 1: 4 segments.

Pair 2: 3 segments.

Pair 3: impossible.





# 问题分析 (1)

## ■ 迷宫求解问题

自相似性表现在每走一步的探测方式相同,  
可以用递归方法求解

## ■ 通过枚举方式找到从起点到终点的路径, 朝一个方向走下去:

- 如果走不通, 则换个方向走
  - 四个方向都走不通, 则回到上一步的地方, 换个方向走
  - 依次走下去, 直到走到终点

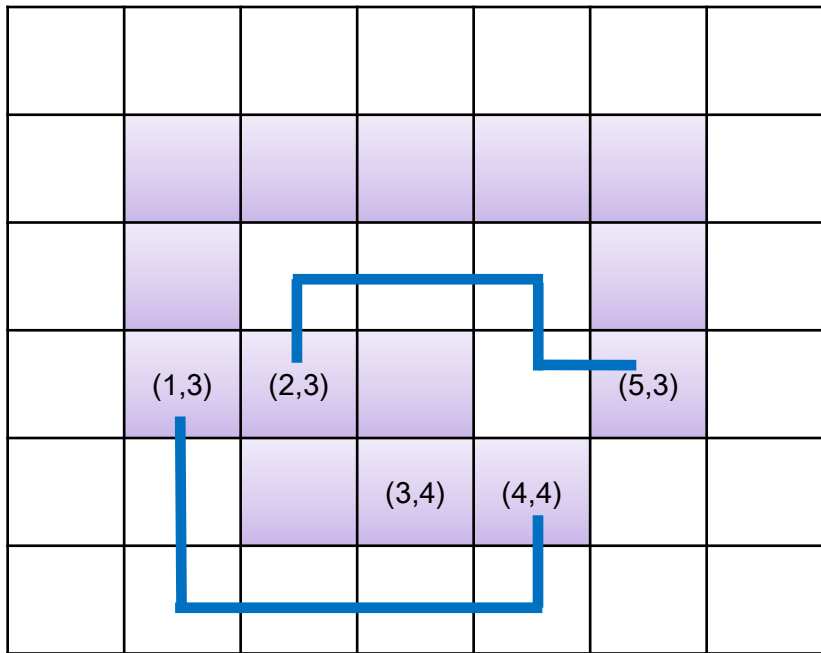


# 问题分析 (1)

- ▲ 计算路径数目:
- ▲ 普通迷宫问题的路径数目是经过的格子数目
- ▲ 而该问题路径只包含水平或者竖直的直线段,
- ▲ 所以需要记录每一步走的方向
  - 如果上一步走的方向和这一步走的方向相同, 递归搜索时路径数不变, 否则路径数加1



## 问题分析 (2)



- ▲ 路径只包含水平或者竖直的直线段. 路径不能穿过别的游戏卡片. 但是允许路径临时的离开矩形板
- ▲ 所以在矩形板最外层增加一圈格子, 路径可以通过这些新增加的格子



# 问题分析 (3)

## 描述迷宫:

1. 设置迷宫为二维数组board[], 数组的值是:  
空格: 代表这个地方没有游戏卡片  
'X': 代表这个地方有游戏卡片
2. 在搜索过程中, 用另外一个二维数mark[][]标记格子是否已经走过了  
mark[i][j]=0 //格子(i, j)未走过  
mark[i][j]=1 //格子(i, j)已经走过
3. int minstep, w, h; //全局变量  
//minstep, 记录从起点到达终点最少路径数,  
//初始化为一个很大的数  
//w, h矩形板的宽度和高度



## 问题分析 (4)

- 设置搜索方向顺序是东, 南, 西, 北

```
int to[4][2] = {{0,1},{1,0},{0,-1},{-1,0}};
```

```
//now_x, now_y, 当前位置
```

```
//x, y下一步位置
```

```
for(i = 0; i < 4; i ++){
```

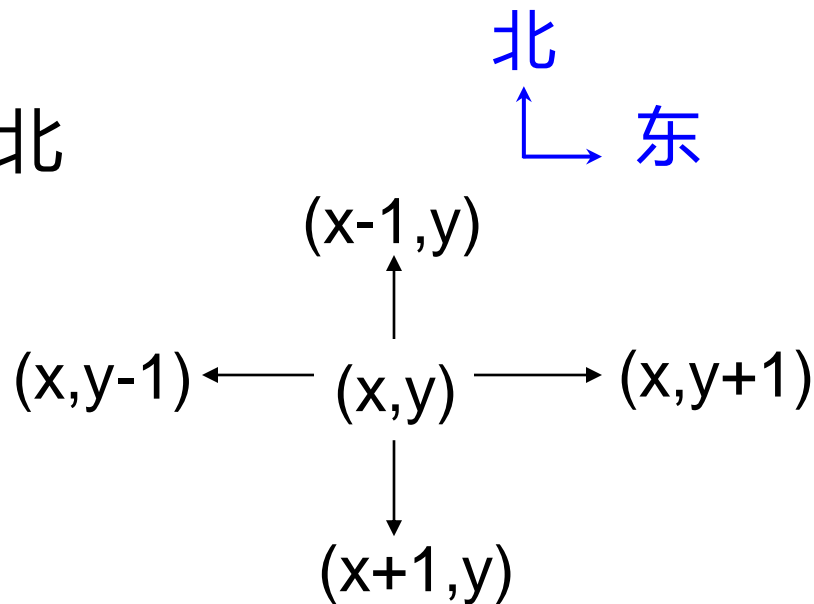
```
    int x = now_x + to[i][0];
```

```
    int y = now_y + to[i][1];
```

```
    f=i; //方向, 0,1,2,3分别表示东,南,西,北
```

```
    ...
```

```
}
```



# 问题分析 (5)

## 判断新位置(x, y)是否有效

### •T1: (x, y)在边界之内

$(x > -1) \ \&\& \ (x < w + 2) \ \&\& \ (y > -1) \ \&\& \ (y < h + 2)$

### •T2: 该位置没有游戏卡片并且未曾走过

$((\text{board}[y][x] == ' ') \ \&\& \ (\text{mark}[y][x] == \text{false}))$

### •T3: 已经到达终点

$(x == \text{end\_x}) \ \&\& \ (y == \text{end\_y}) \ \&\& \ (\text{board}[y][x] == 'X')$

综上, (x,y)有效的条件是  $T1 \ \&\& \ (T2 \ || \ T3)$

$((x > -1) \ \&\& \ (x < w + 2) \ \&\& \ (y > -1) \ \&\& \ (y < h + 2)$

$\ \&\& \ (((\text{board}[y][x] == ' ') \ \&\& \ (\text{mark}[y][x] == \text{false})))$

$\ || \ (((x == \text{end\_x}) \ \&\& \ (y == \text{end\_y}) \ \&\& \ (\text{board}[y][x] == 'X'))))$



# 递归方法

## 构造递归函数

```
void Search(int now_x, int now_y, int end_x, int end_y, int step, int f) ;
```

```
//now_x, now_y当前位置
```

```
//end_x, end_y结束位置
```

```
//step已经走过的路径数目
```

```
//f从上一步走到(now_x, now_y)时的方向
```



# 参考程序

```
#include <stdio.h>
```

```
#include <memory.h>
```

```
#define MAXIN 75
```

```
char board[MAXIN + 2][MAXIN + 2]; //定义矩形板
```

```
int minstep, w, h, to[4][2] = {{0,1},{1,0},{0,-1},{-1,0}}; //定义方向
```

```
bool mark[MAXIN + 2][MAXIN + 2]; //定义标记数组
```





```
void Search(int now_x, int now_y, int end_x, int end_y, int step, int f){  
    if(step > minstep) return; //当前路径数大于minstep, 返回→优化策略  
    if(now_x == end_x && now_y == end_y){ //到达终点  
        if(minstep > step) //更新最小路径数  
            minstep = step;  
        return;  
    }  
}
```



```
for(int i = 0; i < 4; i ++){ //枚举下一步的方向
    int x = now_x + to[i][0]; //得到新的位置
    int y = now_y + to[i][1];
    if ((x > -1) && (x < w + 2) && (y > -1) && (y < h + 2)
        && (((board[y][x] == ' ') && (mark[y][x] == false)) || ((x == end_x)
        && (y == end_y) && (board[y][x] == 'X'))))){
        mark[y][x] = true; //如果新位置有效标记该位置
        //已经过上一步方向和当前方向相同,
        //则递归搜索时step不变, 否则step+1
        if(f == i) Search(x, y, end_x, end_y, step, i);
        else Search(x, y, end_x, end_y, step + 1, i);
        mark[y][x] = false; //回溯, 该位置未曾走过
    }
}
```



```
int main(){
```

```
    int Boardnum = 0;
```

```
    while(scanf("%d %d", &w, &h)){ //读入数据
```

```
        if(w == 0 && h == 0)break;
```

```
        Boardnum ++;
```

```
        printf("Board #%d:\n", Boardnum);
```

```
        int i, j;
```

```
        for (i = 0; i < MAXIN + 2; i ++){board[0][i] = board[i][0] = ' ';
```

```
        for(i = 1; i <= h; i ++){ //读入矩形板的布局
```

```
            getchar();
```

```
            for(j = 1; j <= w; j ++) board[i][j] = getchar();
```

```
        }
```

```
    //在矩形板最外层增加一圈格子
```

```
    for (i = 0; i <= w; i ++)
```

```
        board[h + 1][i + 1] = ' ';
```

```
    for (i = 0; i <= h; i ++)
```

```
        board[i + 1][w + 1] = ' ';
```



```
int begin_x, begin_y, end_x, end_y, count = 0;
while(scanf("%d %d %d %d", &begin_x, &begin_y, &end_x, &end_y)
&& begin_x > 0){ //读入起点和终点
    count ++;
    minstep = 100000; //初始化minstep为一个很大的值
    memset(mark, false, sizeof(mark));
    //递归搜索
    Search(begin_x, begin_y, end_x, end_y, 0, -1);
    //输出结果
    if(minstep < 100000)printf("Pair %d: %d segments.\n", count, minstep);
    else printf("Pair %d: impossible.\n", count);
}
printf("\n");
}
return 0;
}
```



# 问题小结

## 递归的条件

- 自相似性表现在每走一步的探测方式相同, 可以用递归算法求解

## 定义并记录路径方向

## 判断下一步的位置是否符合要求

## 搜索过程Search( )

- 朝一个方向走下去, 如果走不通, 则换个方向走; 四个方向都走不通, 则回到上一步的地方, 换个方向走; 依次走下去, 直到走到终点

## 计算路径数目

- 需要记录每一步走的方向, 如果上一步走的方向和这一步走的方向相同, 递归搜索时路径数不变, 否则路径数加1