

# 枚举 — 讨厌的青蛙

郭 炜 刘家瑛



北京大學

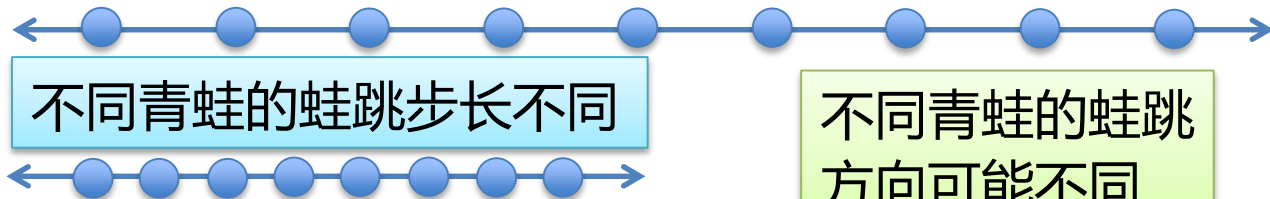


# 讨厌的青蛙

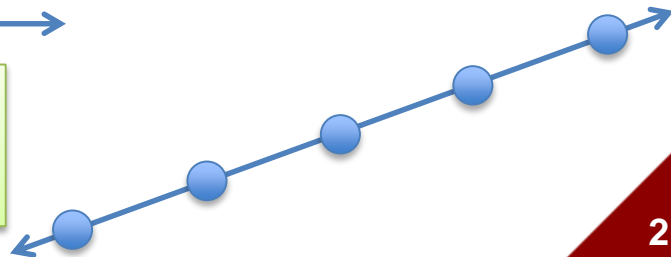


## 问题描述

- 在韩国, 有一种小青蛙
- 每到晚上, 这种青蛙会跳跃稻田, 从而踩踏稻子
- 农民早上看到被踩踏的稻子, 希望找到造成最大损害的那只青蛙经过的路径
- 每只青蛙总是沿着一条**直线**跳跃稻田
- 且**每次跳跃的距离都相同**

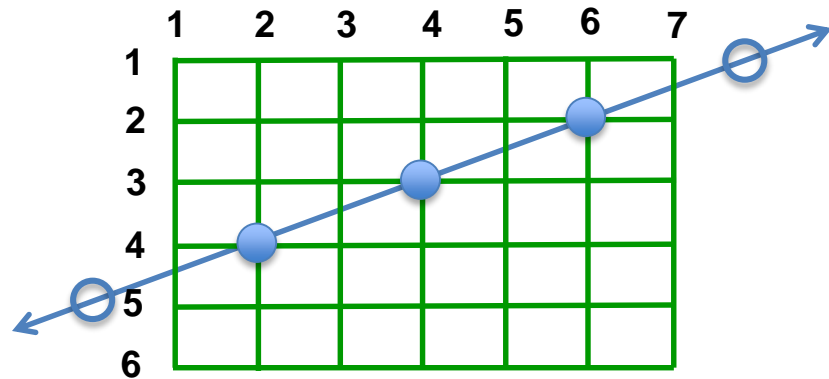
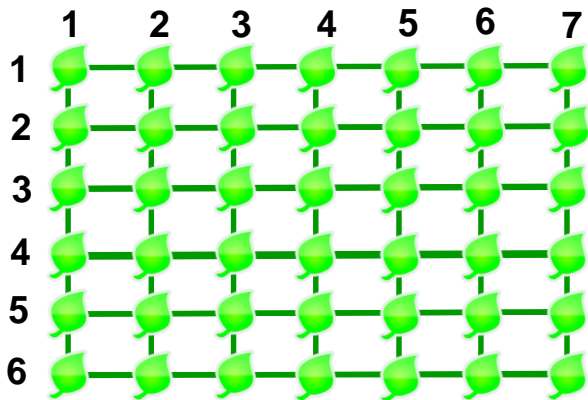


不同青蛙的蛙跳  
方向可能不同





- 稻田里的稻子组成一个栅格, 每棵稻子位于一个格点上
- 而青蛙总是从稻田的一侧跳进稻田, 然后沿着某条直线穿越稻田, 从另一侧跳出去





- 可能会有多只青蛙从稻田穿越
- 青蛙每一跳都恰好踩在一棵水稻上, 将这棵水稻拍倒
- 有些水稻可能被多只青蛙踩踏
- 农民所见到的的是图4中的情形, 并看不到图3中的直线, 也见不到别人家田里被踩踏的水稻

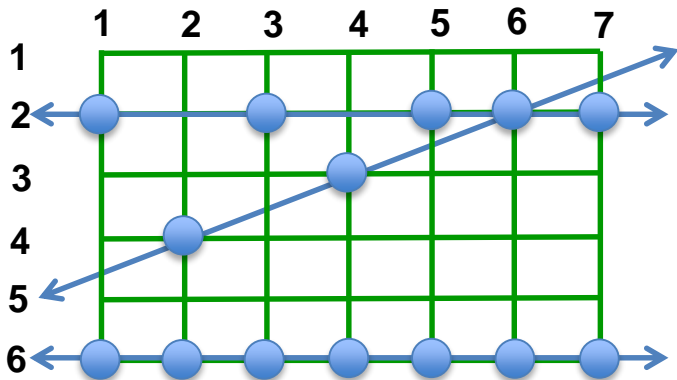


图 3

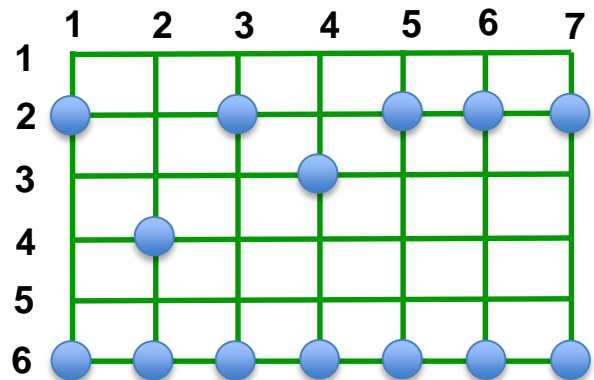
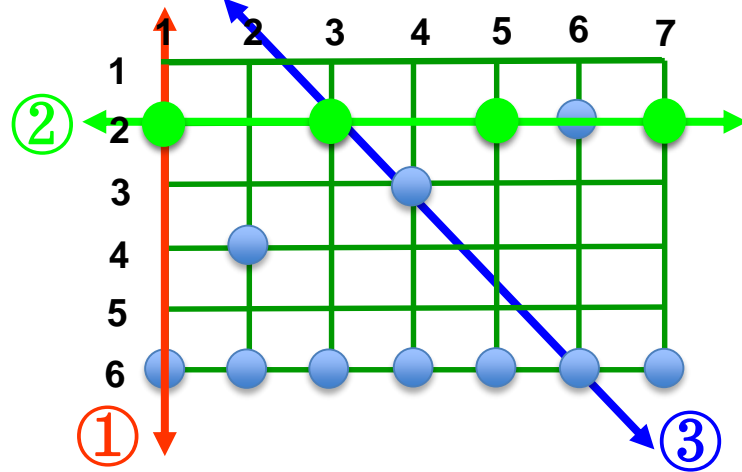
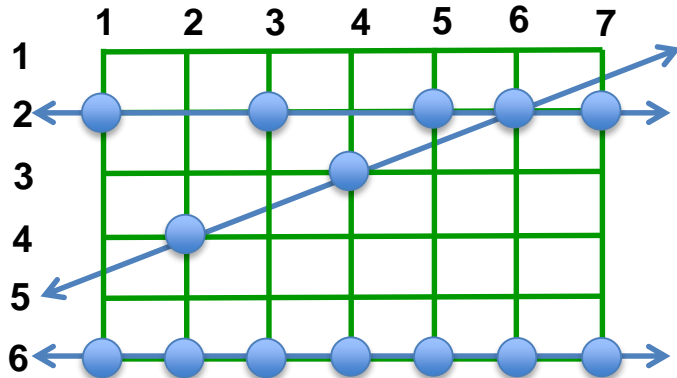


图 4



■ 而在一条青蛙行走路径的直线上, 也可能会有些被踩踏的水稻不属于该行走路径

- ① 不是一条行走路径: 只有2棵被踩踏的水稻
- ② 是一条行走路径, 但不包括 (2,6)上的水稻
- ③ 不是一条行走路径: 虽然有3棵被踩踏的水稻, 但这3棵水稻之间的距离间隔不相等



# 题目要求

请写一个程序, 确定:

- 在各条青蛙行走路径中, 踩踏水稻最多的那一条上, 有多少颗水稻被踩踏
- 例如, 图4中答案是7, 因为第6行上全部水稻恰好构成一条青蛙行走路径

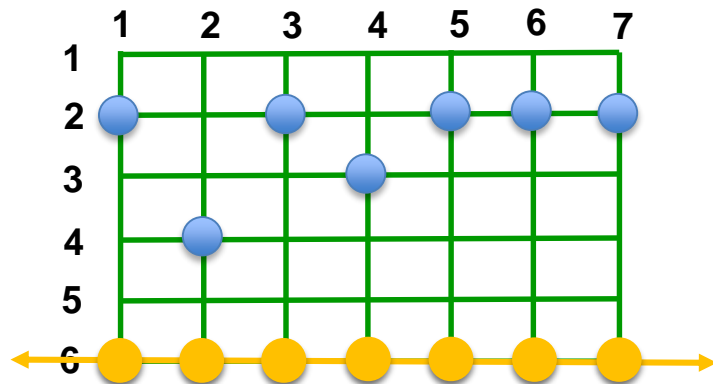


图 4



## 程序输入

- 从标准输入设备上读入数据
- 第一行上两个整数R, C, 分别表示稻田中水稻的行数和列数,  $1 \leq R, C \leq 5000$
- 第二行是一个整数N, 表示被踩踏的水稻数量,  $3 \leq N \leq 5000$
- 在剩下的N行中, 每行有两个整数, 分别是一颗被踩踏水稻的行号(1~R)和列号(1~C), 两个整数用一个空格隔开
- 且每棵被踩踏水稻只被列出一次



## ■ 程序输出

- 从标准输出设备上输出一个整数
- 如果在稻田中存在青蛙行走路径, 则输出包含最多水稻的青蛙行走路径中的水稻数量, 否则输出0





## ▲ 样例输入

6 7 //6行7列

14

2 1

6 6

4 2

2 5

2 6

2 7

3 4

6 1

6 2

2 3

6 3

6 4

6 5

6 7

## ▲ 样例输出

7



# 解题思路 — 枚举

## ■ 枚举什么？

- 枚举每个被踩的稻子作为行走路径起点 (5000个)
- 对每个起点, 枚举行走方向 (5000种)
- 对每个方向枚举步长 (5000种)
- 枚举步长后还要判断是否每步都踩到水稻

■ 时间:  $5000 * 5000 * 5000 *$ , 不行!



# 解题思路 — 枚举

- 枚举什么? 枚举路径上的开始两点
- 每条青蛙行走路径中至少有3棵水稻
- 假设一只青蛙进入稻田后踩踏的前两棵水稻分别是  $(X_1, Y_1), (X_2, Y_2)$ . 那么:
  - 青蛙每一跳在  $X$  方向上的步长  $dX = X_2 - X_1$ ,  
在  $Y$  方向上的步长  $dY = Y_2 - Y_1$ ;
  - $(X_1 - dX, Y_1 - dY)$  需要落在稻田之外;
  - 当青蛙踩在水稻  $(X, Y)$  上时, 下一跳踩踏的水稻是  $(X + dX, Y + dY)$ ;
  - 将路径上的最后一棵水稻记作  $(X_K, Y_K)$ ,  $(X_K + dX, Y_K + dY)$  需要落在稻田之外;



# 解题思路:猜测一条路径

- 猜测的办法需要保证:
- 每条可能的路径都能够被猜测到
  - 从输入的水稻中任取两棵
    - 作为一只青蛙进入稻田后踩踏的前两棵水稻
    - 看能否形成一条穿越稻田的行走路径



# 解题思路:猜测一条路径

- 猜测的过程需要**尽快排除错误的答案**
  - 猜测 $(X1, Y1), (X2, Y2)$  -- 所要寻找的行走路径上的前两棵水稻
- 当下列条件之一满足时, 这个猜测就不成立：
  - 青蛙不能经过一跳从稻田外跳到 $(X1, Y1)$ 上
  - 按照 $(X1, Y1), (X2, Y2)$ 确定的步长, 从 $(X1, Y1)$ 出发, 青蛙最多经过 $(MAXSTEPS - 1)$ 步, 就会跳到稻田之外
  - $MAXSTEPS$ 是当前已经找到的最好答案



# 选择合适的数据结构

- 选择合适的数据结构
  - 采用的数据结构需要与问题描述中的概念对应
- 关于被踩踏的水稻的坐标, 该如何定义?
  - 方案1: struct {  
                  int x, y;  
                  } plants[5000];
  - 方案2: int plantsRow[5000], plantsCol[5000];
- 显然方案1更符合问题本身的描述



# 设计的算法要简洁

- ▲ 尽量使用C提供的函数完成计算的任务
- ▲ 猜测一条行走路径时, 需要从当前位置( $X, Y$ )出发上时, 看看( $X + dX, Y + dY$ )位置的水稻是否被踩踏 ;
  - 方案1: 自己写一段代码, 看看( $X + dX, Y + dY$ ) 是否在数组 plants 中
  - 方案2: 先用 qsort() 对 plants 中的元素排序, 然后用 bsearch() 从中查找元素( $X + dX, Y + dY$ )
- ▲ 基于方案2设计的算法更简洁, 更容易实现, 更不容易出错误
- ▲ 通常, 所选用的数据结构对算法的设计有很大影响



## 注意:

- 一个有 $n$ 个元素的数组, 每次取两个元素, 遍历所有取法
- 代码写法:

```
for( int i = 0; i < n - 1; i ++ )  
    for( int j = i + 1; j < n; j ++ ) {  
        a[i] = ...;  
        a[j] = ...;  
    }
```





# 参考程序

```
#include <stdio.h>
#include <stdlib.h>
#include <algorithm>
using namespace std;
int r, c, n;
struct PLANT {
    int x, y;
};
PLANT plants[5001];
PLANT plant;
int searchPath(PLANT secPlant, int dX, int dY) ;
```



```
int main()
```

```
{
```

```
    int i, j, dX, dY, pX, pY, steps, max = 2;
```

```
    scanf("%d %d", &r, &c);
```

```
    //行数和列数, x方向是上下, y方向是左右
```

```
    scanf("%d", &n);
```

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

```
        scanf("%d %d", &plants[i].x, &plants[i].y);
```

```
    //将水稻按x坐标从小到大排序, x坐标相同按y从小到大排序
```

```
    sort(plants, plants + n);
```



```
for (i = 0; i < n - 2; i++) //plants[i]是第一个点
    for (j = i + 1; j < n - 1; j++) { //plants[j]是第二个点
        dX = plants[j].x - plants[i].x;
        dY = plants[j].y - plants[i].y;
        pX = plants[i].x - dX;
        pY = plants[i].y - dY;
        if (pX <= r && pX >= 1 && pY <= c && pY >= 1)
            continue;
        //第一点的前一点在稻田里,
        //说明本次选的第二点导致的x方向步长不合理(太小)
        // 取下一个点作为第二点
        if (plants[i].x + (max - 1) * dX > r)
            break;
```

//x方向过早越界了. 说明本次选的第二点不成立

//如果换下一个点作为第二点, x方向步长只会更大, 更不成立, 所以应该

//认为本次选的第一点必然是不成立的, 那么取下一个点作为第一点再试



```
pY = plants[ i ].y + (max - 1) * dY;  
if ( pY > c || pY < 1)  
    continue; //y方向过早越界了, 应换一个点作为第二点再试  
steps = searchPath(plants[ j ], dX, dY);  
//看看从这两点出发, 一共能走几步  
if (steps > max)    max = steps;  
}  
if ( max == 2 ) max = 0;  
printf("%d\n", max);  
}  
bool operator < ( const PLANT & p1, const PLANT & p2)  
{  
    if ( p1.x == p2.x )  
        return p1.y < p2.y;  
    return p1.x < p2.x ;  
}
```



//判断从 secPlant点开始, 步长为dx, dy, 那么最多能走几步

```
int searchPath(PLANT secPlant, int dX, int dY){
    PLANT plant;
    int steps;
    plant.x = secPlant.x + dX;
    plant.y = secPlant.y + dY;
    steps = 2;
    while (plant.x <= r && plant.x >= 1 && plant.y <= c && plant.y >= 1) {
        if (! binary_search(plants, plants + n, plant)) {
            //每一步都必须踩倒水稻才算合理, 否则这就不是一条行走路径
            steps = 0;
            break;
        }
        plant.x += dX;
        plant.y += dY;
        steps++;
    }
    return(steps);
}
```



# 小结

- 枚举顺序十分重要, 好的枚举顺序, 能够及早排除不可能的情况, 减少枚举次数
- 本题将踩倒的水稻按照位置进行排序, 枚举的时候先枚举序号低的, 就能够有效减少枚举次数, 体现在:

```
if (plants[i].x + (max - 1) * dX > r) break;
```

//x方向过早越界了. 说明本次选的第二点不成立

//如果换下一个点作为第二点, x方向步长只会更大, 更不成立

//所以应该认为本次选的第一点必然是不成立的,

//那么取下一个点作为第一点再试