



北京大学
PEKING UNIVERSITY

信息科学技术学院

程序设计实习

算法基础

郭 炜

微信公众号



微博: <http://weibo.com/guoweiofpku>

学会程序和算法，走遍天下都不怕!

讲义内照片均为郭炜拍摄



北京大学
PEKING UNIVERSITY

信息科学技术学院

配套教材：

高等教育出版社

《算法基础与在线实践》

刘家瑛 郭炜 李文新 编著

本讲义中所有例题，根据题目名称在
<http://openjudge.cn>
“百练”组进行搜索即可提交





枚 举

枚举

- 基于逐个尝试答案的一种问题求解策略
 - 例如: 求小于N的最大素数
 - 找不到一个数学公式, 使得根据N就可以计算出这个素数
 - $N-1$ 是素数吗? $N-2$ 是素数吗?
- 判断 $N-i$ 是否是素数的问题
- 转化为求小于N的全部素数(可以用筛法)



北京大学
PEKING UNIVERSITY

信息科学技术学院

例题: 完美立方



富士山

例题1：完美立方

- 形如 $a^3 = b^3 + c^3 + d^3$ 的等式被称为完美立方等式。例如 $12^3 = 6^3 + 8^3 + 10^3$ 。编写一个程序，对任给的正整数N ($N \leq 100$)，寻找所有的四元组(a, b, c, d)，使得 $a^3 = b^3 + c^3 + d^3$ ，其中a,b,c,d 大于 1, 小于等于N，且 $b \leq c \leq d$ 。
- 输入
一个正整数N ($N \leq 100$)。
- 输出
每行输出一个完美立方。输出格式为：
Cube = a, Triple = (b,c,d)
其中a,b,c,d所在位置分别用实际求出四元组值代入。

完美立方

请按照a的值，从小到大依次输出。当两个完美立方等式中a的值相同，则b值小的优先输出、仍相同则c值小的优先输出、再相同则d值小的先输出。

- 样例输入

24

完美立方

- 样例输出

Cube = 6, Triple = (3,4,5)

Cube = 12, Triple = (6,8,10)

Cube = 18, Triple = (2,12,16)

Cube = 18, Triple = (9,12,15)

Cube = 19, Triple = (3,10,18)

Cube = 20, Triple = (7,14,17)

Cube = 24, Triple = (12,16,20)

完美立方

- 解题思路

四重循环枚举 a, b, c, d ， a 在最外层， d 在最里层，每一层都是从小到大枚举， a 枚举范围 $[2, N]$

完美立方

- 解题思路

四重循环枚举 a, b, c, d ， a 在最外层， d 在最里层，每一层都是从小到大枚举，

a 枚举范围 $[2, N]$

b 范围 $[2, a-1]$

完美立方

- 解题思路

四重循环枚举 a, b, c, d ， a 在最外层， d 在最里层，每一层都是从小到大枚举，

a 枚举范围 $[2, N]$

b 范围 $[2, a-1]$

c 范围 $[b, a-1]$

d 范围 $[c, a-1]$

完美立方

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
    int N;
    scanf("%d", &N);
    for(int a = 2; a <= N; ++a)
        for(int b = 2; b < a; ++b)
            for(int c = b; c < a; ++c)
                for(int d = c; d < a; ++d)
                    if( a*a*a == b*b*b + c*c*c + d*d*d)
                        printf("Cube = %d, Triple = (%d,%d,%d)\n", a, b, c,
d);
    return 0;
}
```



北京大学
PEKING UNIVERSITY

信息科学技术学院

例题: 生理周期



京都金阁寺

例题2：生理周期

- 人有体力、情商、智商的高峰日子，它们分别每隔23天、28天和33天出现一次。对于每个人，我们想知道何时三个高峰落在同一天。给定三个高峰出现的日子p,e和i（不一定是第一次高峰出现的日子），再给定另一个指定的日子d，你的任务是输出日子d之后，下一次三个高峰落在同一天的日子（用距离d的天数表示）。例如：给定日子为10，下次出现三个高峰同一天的日子是12，则输出2。

生理周期

- 输入

输入四个整数：p, e, i和d。 p, e, i分别表示体力、情感和智力高峰出现的日子。d是给定的日子，可能小于p, e或i。所有给定日子是非负的并且小于或等于365，所求的日子小于或等于21252。

- 输出

从给定日子起，下一次三个高峰同一天的日子（距离给定日子的天数）。

生理周期

- 输入样例

0 0 0 0

0 0 0 100

5 20 34 325

4 5 6 7

283 102 23 320

203 301 203 40

-1 -1 -1 -1

生理周期

● 输出样例

Case 1: the next triple peak occurs in 21252 days.

Case 2: the next triple peak occurs in 21152 days.

Case 3: the next triple peak occurs in 19575 days.

Case 4: the next triple peak occurs in 16994 days.

Case 5: the next triple peak occurs in 8910 days.

Case 6: the next triple peak occurs in 10789 days.

生理周期

- 解题思路

- 从d+1天开始，一直试到第21252 天，对其中每个日期k,看是否满足

$$(k - p) \% 23 == 0 \ \&\& \ (k - e) \% 28 == 0 \ \&\&$$

$$(k - i) \% 33 == 0$$

- 如何试得更快?

生理周期

- 解题思路

- 从d+1天开始，一直试到第21252 天，对其中每个日期k,看是否满足

$$(k - p) \% 23 == 0 \ \&\& \ (k - e) \% 28 == 0 \ \&\&$$

$$(k - i) \% 33 == 0$$

- 如何试得更快?

跳着试!

```

#include <iostream>
#include <cstdio>
using namespace std;
#define N 21252

int main(){
    int p,e,i,d,caseNo = 0;
    while( cin >> p >> e >>i >>d && p!= -1) {
        ++ caseNo;
        int k;
        for(k = d+1; (k-p)%23; ++k);
        for(; (k-e)%28; k+= 23);
        for(; (k-i)%33; k+= 23*28);
        cout << "Case " << caseNo <<
            ": the next triple peak occurs in " << k-d << " days." << endl;
    }
    return 0;
}

```



北京大学
PEKING UNIVERSITY

信息科学技术学院

例题: 称硬币



日本奈良东大寺

例题3: [POJ1013](#) 称硬币

有12枚硬币。其中有11枚真币和1枚假币。假币和真币重量不同，但不知道假币比真币轻还是重。现在，用一架天平称了这些币三次，告诉你称的结果，请你找出假币并且确定假币是轻是重（数据保证一定能找出来）。

例题： [POJ1013](#) 称硬币

● 输入

第一行是测试数据组数。

每组数据有三行，每行表示一次称量的结果。银币标号为A-L。每次称量的结果用三个以空格隔开的字符串表示：天平左边放置的硬币 天平右边放置的硬币 平衡状态。其中平衡状态用`up`, `down`, 或 `even`表示, 分别为右端高、右端低和平衡。天平左右的硬币数总是相等的。

● 输出

输出哪一个标号的银币是假币，并说明它比真币轻还是重。

例题： [POJ1013](#) 称硬币

- 输入样例

1

ABCD EFGH even

ABCI EFJK up

ABIJ EFGH even

- 输出样例

K is the counterfeit coin and it is light.

例题：[POJ1013](#) 称硬币

● 解题思路

对于每一枚硬币先假设它是轻的，看这样是否符合称量结果。如果符合，问题即解决。如果不符合，就假设它是重的，看是否符合称量结果。把所有硬币都试一遍，一定能找到特殊硬币

```
#include <iostream>
#include <cstring>
using namespace std;
char Left[3][7];    //天平左边硬币
char Right[3][7];   //天平右边硬币
char result[3][7];  //结果
bool IsFake(char c,bool light) ;
//light 为真表示假设假币为轻，否则表示假设假币为重
```

```
int main() {
    int t;
    cin >> t;
    while(t--) {
        for(int i = 0; i < 3; ++i) cin >> Left[i] >> Right[i] >> result[i];
        for(char c='A'; c<='L';c++) {
            if( IsFake(c,true) ){
                cout << c << " is the counterfeit coin and it is light.\n";
                break;
            }
            else if( IsFake(c,false) ){
                cout << c << " is the counterfeit coin and it is heavy.\n";
                break;
            }
        }
    }
    return 0; }
```

```
bool IsFake(char c,bool light)
//light 为真表示假设假币为轻，否则表示假设假币为重
{
    for(int i = 0;i < 3; ++i) {
        char * pLeft,*pRight; //指向天平两边的字符串
        if(light) {
            pLeft = Left[i];
            pRight = Right[i];
        }
        else {
            pLeft = Right[i];
            pRight = Left[i];
        }
    }
}
```

```
switch(result[i][0]) {
    case 'u':
        if ( strchr(pRight,c) == NULL)
            return false;
        break;
    case 'e':
        if( strchr(pLeft,c) || strchr(pRight,c))
            return false;
        break;
    case 'd':
        if ( strchr(pLeft,c) == NULL)
            return false;
        break;
}
}
return true;
}
```



北京大学
PEKING UNIVERSITY

信息科学技术学院

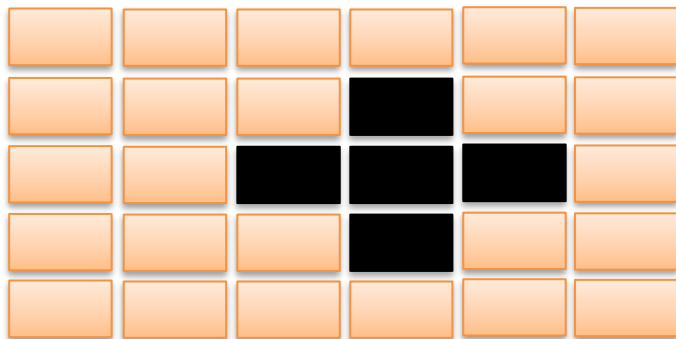
例题: 熄灯问题



日本大阪天守阁

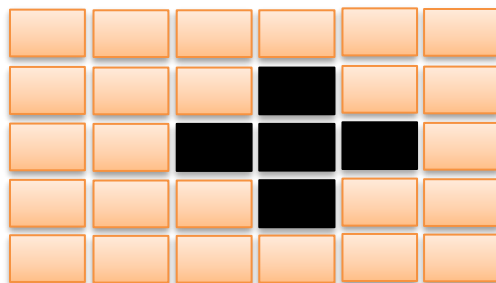
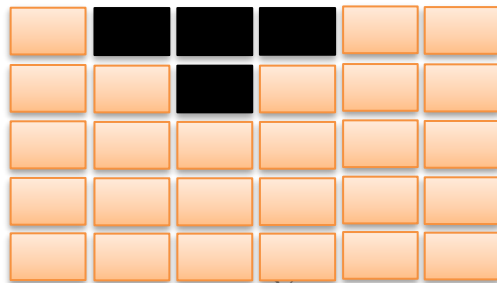
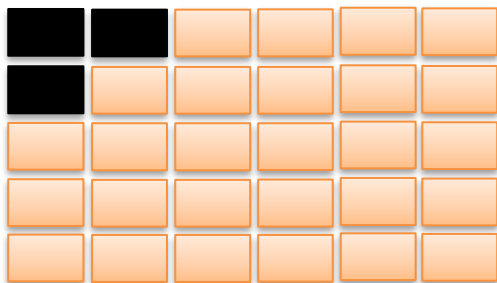
例题4：熄灯问题 [POJ1222](#)

- 有一个由按钮组成的矩阵, 其中每行有6个按钮, 共5行
- 每个按钮的位置上有一盏灯
- 当按下一个按钮后, 该按钮以及周围位置(上边, 下边, 左边, 右边)的灯都会改变状态



熄灯问题 [POJ1222](#)

- 如果灯原来是点亮的, 就会被熄灭
- 如果灯原来是熄灭的, 则会被点亮
 - 在矩阵**角上**的按钮改变**3盏灯**的状态
 - 在矩阵**边上**的按钮改变**4盏灯**的状态
 - **其他的按钮**改变**5盏灯**的状态



熄灯问题 [POJ1222](#)

- 与一盏灯毗邻的多个按钮被按下时,一个操作会抵消另一次操作的结果
- 给定矩阵中每盏灯的初始状态, 求一种按按钮方案, 使得所有的灯都熄灭

熄灯问题 [POJ1222](#)

● 输入:

- 第一行是一个正整数N, 表示需要解决的案例数
- 每个案例由5行组成, 每一行包括6个数字
- 这些数字以空格隔开, 可以是0或1
- **0** 表示灯的初始状态是**熄灭**的
- **1** 表示灯的初始状态是**点亮**的

熄灯问题 [POJ1222](#)

- 输出:
 - 对每个案例, 首先输出一行,
输出字符串 “PUZZLE #m”, 其中m是该案例的序号
 - 接着按照该案例的输入格式输出5行
 - **1** 表示需要把对应的按钮按下
 - **0** 表示不需要按对应的按钮
 - 每个数字以一个空格隔开

熄灯问题 [POJ1222](#)

● 样例输入

```
2
0 1 1 0 1 0
1 0 0 1 1 1
0 0 1 0 0 1
1 0 0 1 0 1
0 1 1 1 0 0
0 0 1 0 1 0
1 0 1 0 1 1
0 0 1 0 1 1
1 0 1 1 0 0
0 1 0 1 0 0
```

● 样例输出

PUZZLE #1

```
1 0 1 0 0 1
1 1 0 1 0 1
0 0 1 0 1 1
1 0 0 1 0 0
0 1 0 0 0 0
```

PUZZLE #2

```
1 0 0 1 1 1
1 1 0 0 0 0
0 0 0 1 0 0
1 1 0 1 0 1
1 0 1 1 0 1
```

解题分析

- 第2次按下同一个按钮时,
将抵消第1次按下时所产生的结果

解题分析

- 第2次按下**同一个按钮**时,
将抵消第1次按下时所产生的结果
→每个按钮最多只需要按下一次

解题分析

- 第2次按下**同一个按钮**时,
将抵消第1次按下时所产生的结果
→每个按钮最多只需要按下一次
- 各个按钮被按下的顺序对最终的结果没有影响

解题分析

- 第2次按下**同一个按钮**时,
将抵消第1次按下时所产生的结果
→每个按钮最多只需要按下一次
- 各个按钮被按下的顺序对最终的结果没有影响
- 对第1行中每盏点亮的灯, 按下第2行对应的按钮, 就可以熄灭第1行的全部灯
- 如此重复下去, 可以熄灭第1, 2, 3, 4行的全部灯

解题分析

- 第一想法: 枚举所有可能的按钮(开关)状态, 对每个状态计算一下最后灯的情况, 看是否都熄灭
 - 每个按钮有两种状态(按下或不按下)
 - 一共有30个开关, 那么状态数是 2^{30} , 太多, 会超时

解题分析

- 第一想法: 枚举所有可能的按钮(开关)状态, 对每个状态计算一下最后灯的情况, 看是否都熄灭
 - 每个按钮有两种状态(按下或不按下)
 - 一共有30个开关, 那么状态数是 2^{30} , 太多, 会超时
- 如何减少枚举的状态数目呢?

基本思路: 如果存在某个局部, 一旦这个局部的状态被确定, 那么剩余其他部分的状态只能是确定的一种, 或者不多的n种, 那么就只需枚举这个局部的状态即可

解题分析

- 本题是否存在这样的 “局部” 呢?
- 经过观察, 发现第1行就是这样的 一个 “局部”
 - 因为第1行的各开关状态确定的情况下, 这些开关作用过后, 将导致第1行某些灯是亮的, 某些灯是灭的
 - 要熄灭第1行某个亮着的灯(假设位于第 i 列), 那么唯一的办法就是按下第2行第 i 列的开关
 - (因为第1行的开关已经用过了, 而第3行及其后的开关不会影响到第1行)
 - 为了使第1行的灯全部熄灭, 第2行的合理开关状态就是唯一的

解题分析

- 第2行的开关起作用后,
 - 为了熄灭第2行的灯, 第3行的合理开关状态就也是唯一的
 - 以此类推, 最后一行的开关状态也是唯一的
- 只要第1行的状态定下来, 记作A, 那么剩余行的情况就是确定唯一的了

解题分析

- 推算出最后一行的开关状态, 然后看看最后一行的开关起作用后, 最后一行的所有灯是否都熄灭:
- 如果是, 那么A就是一个解的状态
 - 如果不是, 那么A不是解的状态, 第1行换个状态重新试试
 - 只需枚举第1行的状态, 状态数是 $2^6 = 64$

有没有状态数更少的做法？

- 枚举第一列, 状态数是 $2^5 = 32$

```
#include <memory>
#include <string>
#include <cstring>
#include <iostream>
using namespace std;
int GetBit(char c,int i) {
    //取c的第i位
    return ( c >> i ) & 1;
}
void SetBit(char & c,int i, int v) {
    //设置c的第i位为v
    if( v )
        c |= ( 1 << i );
    else
        c &= ~( 1 << i );
}
void Flip(char & c, int i) {
    //将c的第i位为取反
    c ^= ( 1 << i );
}
```

```
void OutputResult(int t, char result[]) //输出结果
```

```
{  
    cout << "PUZZLE #" << t << endl;  
    for( int i = 0; i < 5; ++i ) {  
        for( int j = 0; j < 6; ++j ) {  
            cout << GetBit(result[i], j);  
            if( j < 5 )  
                cout << " ";  
        }  
        cout << endl;  
    }  
}
```



```
int main()    {
    char oriLights[5]; //最初灯矩阵，一个比特表示一盏灯
    char lights[5];    //不停变化的灯矩阵
    char result[5];    //结果开关矩阵
    char switchs;      //某一行的开关状态
    int T;
    cin >> T;
    for( int t = 1; t <= T; ++ t) {
        memset(oriLights,0,sizeof(oriLights));
        for( int i = 0;i < 5; i ++ ) { //读入最初灯状态
            for( int j = 0; j < 6; j ++ ) {
                int s;
                cin >> s;
                SetBit(oriLights[i],j,s);
            }
        }
    }
}
```

```
for( int n = 0; n < 64; ++n ) { //遍历首行开关的64种状态
    memcpy(lights,oriLights,sizeof(oriLights));
    switchs = n; //第i行的开关状态
    for( int i = 0;i < 5; ++i ) {
        result[i] = switchs; //第i行的开关方案
        for( int j = 0; j < 6; ++j ) {
            if( GetBit(switchs,j)) {
                if( j > 0)
                    Flip(lights[i],j-1); //改左灯
                Flip(lights[i],j); //改开关位置的灯
                if( j < 5 )
                    Flip(lights[i],j+1); //改右灯
            }
        }
        if( i < 4 )
            lights[i+1] ^= switchs; //改下一行的灯
        switchs = lights[i]; //第i+1行开关方案和第i行灯情况同
    }
}
```

```
        if( lights[4] == 0 ) {  
            OutputResult(t,result);  
            break;  
        }  
    } // for( int n = 0; n < 64; n ++ )  
}  
return 0;  
}
```

使用bitset的解法

```
#include <string>
#include <cstring>
#include <iostream>
#include <bitset>
#include <algorithm>
using namespace std;
void OutputResult(int t,bitset<6> result[]) //输出结果
{
    cout << "PUZZLE #" << t << endl;
    for( int i = 0;i < 5; ++i ) {
        for( int j = 0; j < 6; ++j ) {
            cout << result[i][j];
            if( j < 5 )
                cout << " ";
        }
        cout << endl;
    }
}
```

```
int main()    {
    bitset<6> oriLights[8]; //最初灯矩阵，一个比特表示一盏灯
    bitset<6> lights[8];    //不停变化的灯矩阵
    bitset<6> result[8];    //结果开关矩阵
    bitset<6> switchs;      //某一行的开关状态
    int T;
    cin >> T;
    for( int t = 1; t <= T; ++ t) {
        for( int i = 0; i < 5; i ++ ) { //读入最初灯状态
            for( int j = 0; j < 6; j ++ ) {
                int s;
                cin >> s;
                oriLights[i][j] = s;
            }
        }
    }
```

```
for( int n = 0; n < 64; ++n ) { //遍历首行开关的64种状态
    copy(oriLights,oriLights+8,lights);
    switchs = n; //第i行的开关状态
    for( int i = 0;i < 5; ++i ) {
        result[i] = switchs; //第i行的开关方案
        for( int j = 0; j < 6; ++j ) {
            if( switchs[j]) {
                if( j > 0)
                    lights[i].flip(j-1); //改左灯
                lights[i].flip(j); //改开关位置的灯
                if( j < 5 )
                    lights[i].flip(j+1); //改右灯
            }
        }
        if( i < 4 )
            lights[i+1] ^= switchs; //改下一行的灯
        switchs = lights[i]; //第i+1行开关方案和第i行灯情况同
    }
}
```

```
        if( lights[4] == 0 ) {  
            OutputResult(t,result);  
            break;  
        }  
    } // for( int n = 0; n < 64; n ++ )  
}  
return 0;  
}
```



北京大学
PEKING UNIVERSITY

信息科学技术学院

例题: 讨厌的青蛙



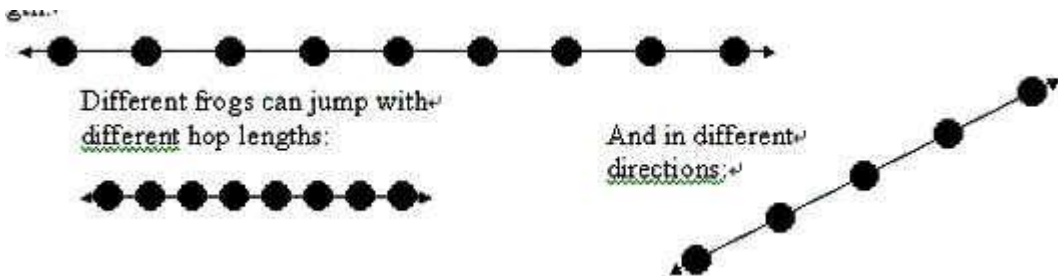
日本京都清水寺

例题5：讨厌的青蛙

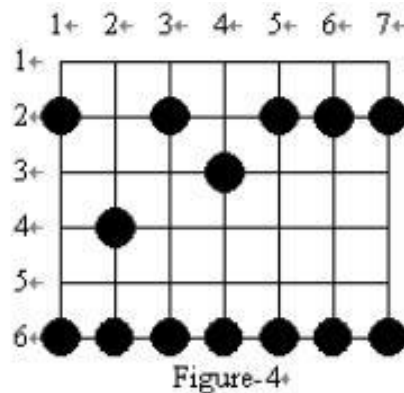
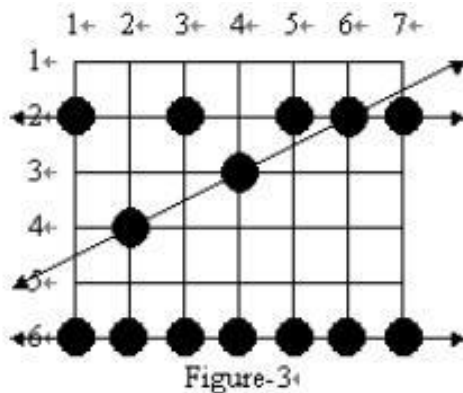


– POJ1054:The Troublesome Frog

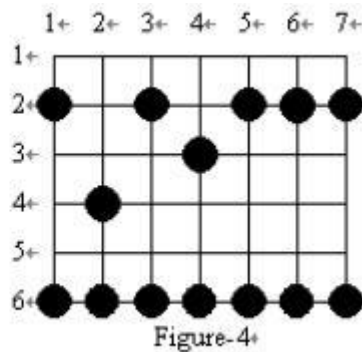
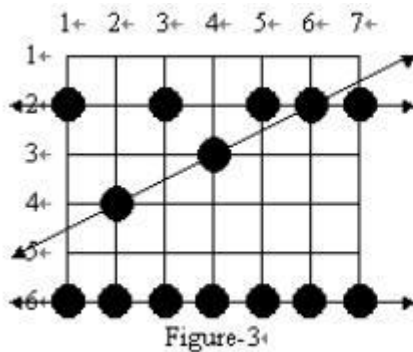
- 在韩国, 有一种小青蛙
- 每到晚上, 这种青蛙会跳跃稻田, 从而踩踏稻子
- 农民早上看到被踩踏的稻子, 希望找到造成最大损害的那只青蛙经过的路径
- 每只青蛙总是沿着一条直线跳跃稻田
- 且每次跳跃的距离都相同

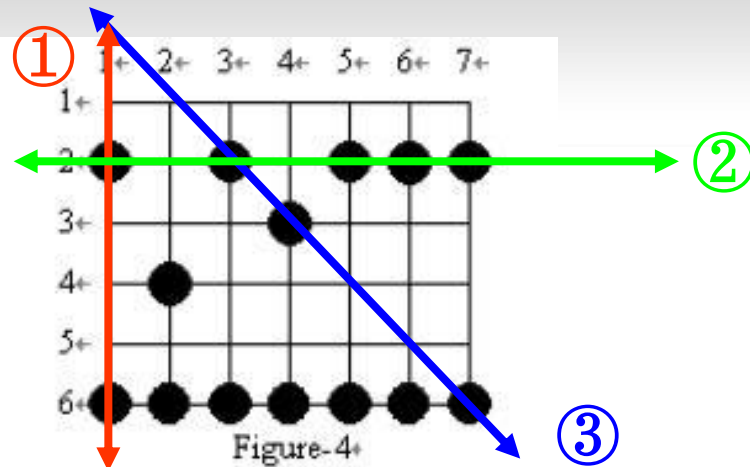
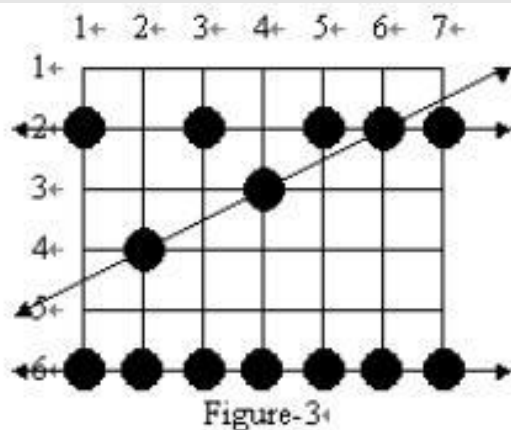


- 稻田里的稻子组成一个栅格, 每棵稻子位于一个格点上
- 而青蛙总是从稻田的一侧跳进稻田, 然后沿着某条直线穿越稻田, 从另一侧跳出去



- 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





- 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

- 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

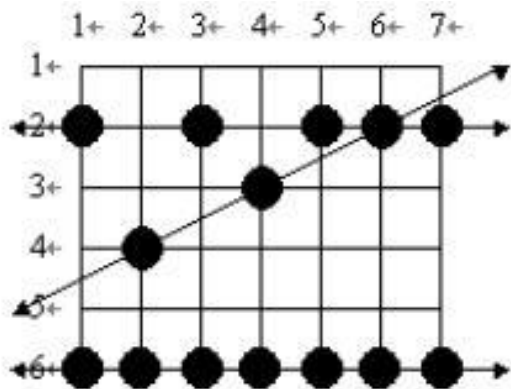


Figure-3+

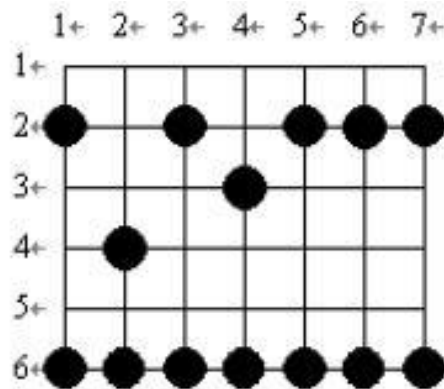


Figure-4+

● 输入

- 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($1 \sim R$)和列号($1 \sim C$), 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

● 输出

- 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?

- 1) 枚举青蛙踩踏的前两点 (5000×5000), 步长和方向也就随之确定

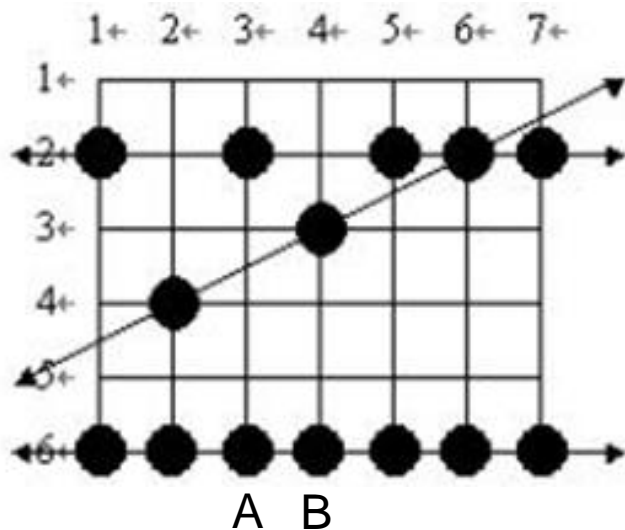
- 假设一只青蛙进入稻田后踩踏的前两棵水稻分别是 $(X_1, Y_1), (X_2, Y_2)$. 那么:
- 青蛙每一跳在 X 方向上的步长 $dX = X_2 - X_1$,
在 Y 方向上的步长 $dY = Y_2 - Y_1$;

解题思路 — 枚举

- 2) 接下来判断每走一步是否都会踩到水稻($5000 * T$), 如果能, 则记录一共走多少步后出稻田(T 是判断时间)
- 时间: $5000 * 5000 * 5000 * T$, 似乎惊人, 但其实运气不会这么坏

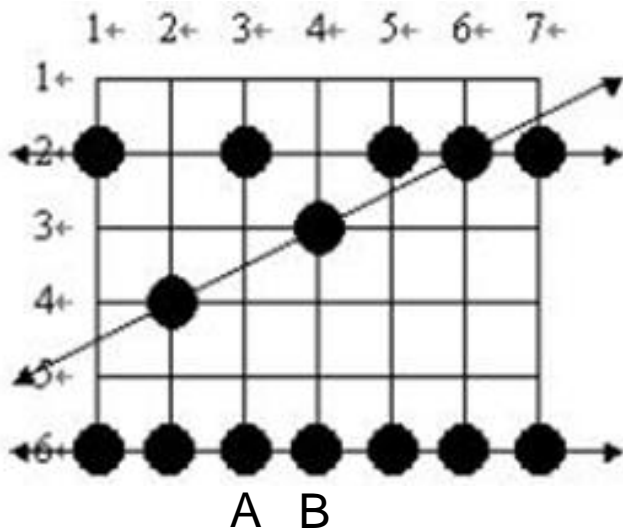
及早排除不必要的尝试

1) 要判断前两点的合法性



及早排除不必要的尝试

1) 要判断前两点的合法性



若选了A,B作为前两点，
立即能判断这不成立。因
步长为1，而青蛙不可能
从稻田外一步跳到A。

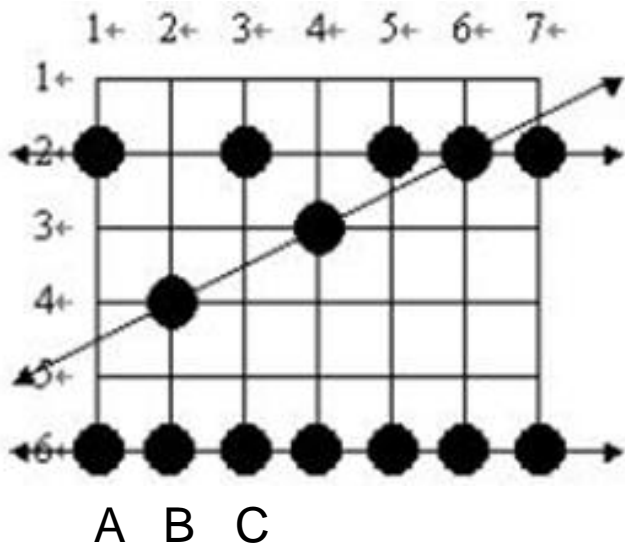
直接否定这前两点的假设
，不必进行后续判断

及早排除不必要的尝试

2) 如果已经发现踩倒 n 棵水稻的路径, 那么, 最多不超过 n 步就会跳出稻田的前两点的方案, 就可以直接否定。

选择合适枚举顺序

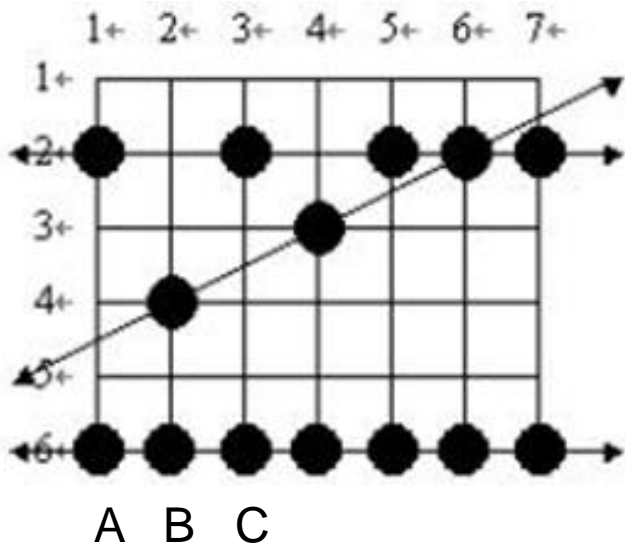
3) 有序，而非随机枚举前两点



若先枚举了A 和B作为前两点并发现可行，则枚举AC作为前两点就没有必要了。

选择合适枚举顺序

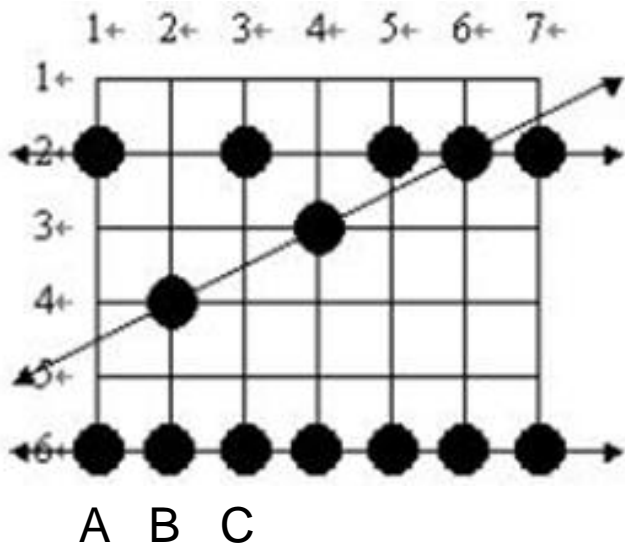
3) 有序，而非随机枚举前两点



为避免对AC的没必要枚举，应该确定枚举顺序，即在第一点相同时，优先枚举X方向上步长短的（Y方向亦可）。如何做到这个枚举顺序？

选择合适枚举顺序

3) 有序，而非随机枚举前两点



将所有水稻按X坐标从小到大排序，X坐标相同则按Y坐标排序。枚举时确保第一点X坐标一定小于等于第二点X坐标。

如何判断某一步是否踩到水稻

- 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

如何判断某一步是否踩到水稻

- 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:** 设置标记数组, 常数时间即可知道某点上有无水稻。代价: 5000×5000 的数组

如何判断某一步是否踩到水稻

- 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2: 用 `binary_search()` 从已经排好序的水稻数组中查找 ($X + dX, Y + dY$) 看是否存在

```
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <cstring>
#include <bitset>
using namespace std;
int r, c, n;
struct Plant { //水稻排序规则,先按x坐标从小到大排, x坐标相同则按y坐标从小到大排
    int x, y;
    bool operator < (const Plant & p) {
        if (x == p.x )
            return y < p.y;
        return x < p.x ;
    }
};
Plant plants[5010];
//char plantExists[5010][5010]; //plantExists[i][j] == 1表示 (i,j)处有水稻
bitset<5010> plantExists[5010]; //更节省空间
int maxSteps(Plant secPlant, int dX, int dY); //求最多步数的函数
```

```
int main()
{
    int i, j, dX, dY, pX, pY, steps, max = 2;
    memset(plantExists, 0, sizeof(plantExists));
    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);
        plantExists[plants[i].x][plants[i].y] = 1;
    }
    //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; // (pX, pY) 是第一点之前的点
            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 = maxSteps(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);
```

```
}
```

//判断以 secPlant作为第2点, 步长为dx, dy, 那么最多能走几步

```
int maxSteps(Plant secPlant, int dX, int dY)
{
    Plant p;
    int steps = 2;
    p.x = secPlant.x + dX;
    p.y = secPlant.y + dY;
    while (p.x <= r && p.x >= 1 && p.y <= c && p.y >= 1) {
        if( ! plantExists[p.x][p.y])
            //或: if(!binary_search(plants,plants+n,p))
            return 0; //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
        p.x += dX;
        p.y += dY;
        ++steps;
    }
    return steps;
}
```

小 结

- 注意枚举顺序
- 尽早排除不可能的情况
- 进行最优性剪枝