

# Chapter 1

## Atcoder

## 1.1 散场电影

### 1.1.1 问题描述

电影院有  $N$  排座椅，每排座椅有  $N$  个座位，这样整个电影院就有  $N^2$  个座位，每排都按相同方向从左向右计序号，第 1 排从左到右序号分别是  $1, 2, 3, \dots, N$ ，第 2 排从左到右分别为  $N+1, N+2, \dots, 2N$ ，最后一排分别是  $N^2-N+1, N^2-N+2, \dots, N^2$ 。

这一场电影坐无虚席。散场时所有观众将会按照序列  $\{P_i\}$  的顺序离场，但是如果  $x$  离场的时候，与他的座位相邻上下左右四个方向都还有其它观众没离场，那么  $x$  的离场就需要强行越过这个人出去，那么就会招致这个观众  $y$  对  $x$  的不满，将这一组不满意记录记为  $(x, y)$ 。此问题给出电影院的规模  $N$  和离场顺序，计算这个离场顺序最少有多少不满意记录。

### 1.1.2 分析

For each  $1 \leq i \leq N^2$ , we compute the minimum number of viewers that will hate viewer  $i$  forever (the answer is the sum of these values). This number coincides with the minimum cost of a path from the seat of viewer  $i$  to the sides of the square, considering that going through an empty seat has cost 0 and going through an occupied seat has cost 1. Let  $H_k(i)$  be the minimum cost (as defined above) of a path from the seat of viewer  $i$  to the outside after the first  $k$  viewers have left the cinema.

Key observation

The values  $H_k(i)$  are decreasing (for  $k$  going from 0 to  $N^2$ ) and at the beginning we have

对每个  $i$  我们都计算一下被嫌弃的最小人数（这些之和就是答案）。这个数字与它到正方形的边界上的观众数相等。

$$\sum_{k=1}^{N^2} H_0(k) \approx \frac{N^3}{6}$$

Our strategy is to keep all values  $H_k(i)$  updated at all times. Initializing  $H_0(1), \dots, H_0(N^2)$  in  $O(N^2)$  is straightforward. Let us show how to update  $H_{k-1}(1), \dots, H_{k-1}(N^2)$  to get  $H_k(1), \dots, H_k(N^2)$ . When the viewer  $P_k$  goes away, we perform a breadth-first search (or a depth-first search) starting from the seats of  $P_k$  and updating the values. During the  $k$ -th breadth-first search, we will visit only the seats  $i$  such that  $H_k(i) < H_{k-1}(i)$ , hence the total number of seats visited in the  $N^2$  steps (for  $1 \leq k \leq N^2$ ) is  $O(N^3)$  (see the key observation).

我们的方案是保持所有  $H_k(i)$  总在更新。 $O(N^2)$  的时间初始化  $H_0(1), \dots, H_0(N^2)$ 。接下来将讨论如何从  $H_{k-1}(1), \dots, H_{k-1}(N^2)$  得出  $H_k(1), \dots, H_k(N^2)$ 。当观众  $P_k$  离开时，我们用 BFS（或 DFS）从  $P_k$  开始更新。当 BFS 进行到第  $k$  名观众时，就只访问那些  $H_k(i) < H_{k-1}(i)$  的位置，这样访问所有  $N^2$  个位置之后时间复杂度是  $O(N^3)$ 。

The time complexity of this solution is  $O(N^3)$  with a small constant which is sufficient to get accepted (some optimization might be required in slow languages such as python).

这样的解法时间复杂度是常数比较小的  $O(N^3)$ , 这足够 Accepted(不过像 Python 等比较慢的语言需要额外优化一下)

## 1.2 冷冻沙丁鱼

### 1.2.1 问题描述

钓了  $N$  条沙丁鱼，每条沙丁鱼的美味度为  $A$ ，鲜度为  $B$ 。现要将沙丁鱼存放到冰箱里冷冻。但是沙丁鱼的冷冻是有特殊要求的：如果有两条沙丁鱼  $i, j$  的美味度和鲜度满足： $A_i A_j + B_i B_j = 0$ ，则这两条鱼会急速腐化，导致整个冰箱中的鱼都坏掉，所以不能让这样的鱼同时冷冻在冰箱中。分别给出  $N$  条沙丁鱼的美味度  $A_i$  与鲜度  $B_i$ ，计算有多少种选法可以正常进行冷冻沙丁鱼而不致其腐化。（结果可能较大，所以输出对 1000000007 取余的值）

[https://vjudge.net/problem/AtCoder-abc168\\_e](https://vjudge.net/problem/AtCoder-abc168_e)

### 1.2.2 分析

$(A_i, B_i) = (0, 0)$  のイワシは他のどの個体とも仲が悪いので、「 $(A_i, B_i) = (0, 0)$  なイワシをどれか 1 匹だけ選ぶ」「 $(A_i, B_i) = (0, 0)$  なイワシをどれも選ばない」のいずれかです。勿論、前者の選び方はそのようなイワシの個数に等しいので、以下では  $(A_i, B_i) = (0, 0)$  なイワシを除外して考えます。

イワシの「傾き」を  $\frac{A_i}{B_i}$  として、これを既約分数で表すことを考えます。具体的には、次のように決めます

1.  $B_i = 0$  のとき、傾きは  $1/0$

2.  $B_i > 0$  のとき、傾きは  $\frac{A_i}{\gcd(A_i, B_i)} / \frac{B_i}{\gcd(A_i, B_i)}$

3.  $B_i < 0$  のとき、傾きは  $-\frac{A_i}{\gcd(A_i, B_i)} / -\frac{B_i}{\gcd(A_i, B_i)}$

ちょっとした場合分けから、傾きと仲の悪さの関係について次のことが言えます。

1. 傾き  $1/0$  のイワシと仲が悪いのは、傾き  $0/1$  のイワシ全てのみ。逆もまた然り。

斜率  $1/0$  の沙丁鱼会与斜率为  $0/1$  的发生腐化，反之亦然

2. 傾き  $a/b$  ( $a, b \neq 0$ ) のイワシと仲が悪いのは、傾き  $-b/a$  (を  $a$  の符号で通分したもの) のイワシ全てのみ。逆もまた然り。

$(A_i, B_i) = (0, 0)$  の沙丁鱼会和任意的沙丁鱼组合都会腐化，就有只选一只  $(A_i, B_i) = (0, 0)$  的沙丁鱼，和  $(A_i, B_i) = (0, 0)$  一只都不选这两类，显然，前者的选法和  $(A_i, B_i) = (0, 0)$  的沙丁鱼数量一样。接下来主要对后者情况讨论，即  $(A_i, B_i) \neq (0, 0)$ 。

定义沙丁鱼的“斜率”为  $\frac{A_i}{B_i}$ ，并且这个数值要表示为分数约分后的结果。更具体的规则如下所示

$B_i = 0$  时，斜率为  $1/0$

$B_i > 0$  时，斜率为  $\frac{A_i}{\gcd(A_i, B_i)} / \frac{B_i}{\gcd(A_i, B_i)}$

$B_i < 0$  时，斜率为  $-\frac{A_i}{\gcd(A_i, B_i)} / -\frac{B_i}{\gcd(A_i, B_i)}$

以下的情况中描述了斜率和相互腐化的关系

斜率为  $a/b$  ( $a, b \neq 0$ ) 的沙丁鱼会与斜率为  $-b/a$  的发生腐化，反之亦然

たとえば以下の傾きが，互いに”仲の悪い”  
がい”です。

例如以下几组斜率是相互被腐化的

$$1/0 \longleftrightarrow 0/1$$

$$5/3 \longleftrightarrow -3/5$$

$$2/1 \longleftrightarrow -1/2$$

したがって，”仲の悪い”がい”になる傾きの  
ペア (高々  $N$  通り) 全てについて，連想配列など  
を使って各傾きになるイワシの個数が求まれば，  
その後は基礎的な数え上げの範疇です。オーバー  
フローには気を付けてください。

$(A_i, B_i)$  の制約が大きいのので，傾きを有理数  
の代わりに実数で表現しようとする (long dou-  
ble でも) 精度が足りないおそれがあります。

因此,可以使用组成“沙丁鱼腐化斜率数对”(共  
 $N$  个), 把斜率出现的个数存储在关联数组 (例如  
C++ 的 map 或 Python 的 dict) 当中, 然后进行  
计数即可。请小心数据溢出。

由于此题  $(A_i, B_i)$  的数值可能很大, 如果用相  
应的小数代替分数, 即使是用 long double 精度仍  
然不够。

```

1  #include <cstdio>
2  #include <map>
3  using std::map;
4
5  class ratNode
6  {
7      private:
8          long long num, den;
9          long long gcd(long long a, long long b)
10         {
11             if (a==0)
12                 return b;
13             if (b==0)
14                 return a;
15             if (a%b==0)
16             {
17                 return b;
18             }
19             else
20             {
21                 return gcd(b, a%b);
22             }
23         }
24     }

```

```
25     public :
26         ratNode(long long n, long long d)
27         {
28             if (d==0)
29             {
30                 den=0;
31                 if (n>0)
32                     num=1;
33                 else if (n<0)
34                     num=-1;
35                 else
36                     num=0;
37             }
38             else if (d>0)
39             {
40                 long long g = gcd(n,d);
41                 num = n/g;
42                 den = d/g;
43             }
44             else
45             {
46                 long long g = gcd(n,d);
47                 num = -n/g;
48                 den = -d/g;
49             }
50         }
51
52         bool operator< (const ratNode &o) const
53         {
54             if (num*o.num<0)
55             {
56                 return num<o.num;
57             }
58             else
59             {
60                 return o.den*num<o.num*den;
61             }
62         }
63     };
64
65     int main(void)
66     {
```

```
67     map<ratNode ,int> m;
68     int i ,j ;
69     for ( i=-2;i <5;i++)
70     {
71         for ( j=-2;j <5;j++)
72         {
73             if(m.count ( ratNode ( i , j ))==0)
74                 m[ ratNode ( i , j )]= i+j ;
75         }
76     }
77     for ( i=-2;i <5;i++)
78     {
79         for ( j=-2;j <5;j++)
80         {
81             printf ( "%d,%d\t%d\n" , i , j ,m[ ratNode ( i , j )] );
82         }
83     }
84     return 0;
85 }
```

## 1.3 Pay to Win

### 1.3.1 问题描述

你现在手里只有数字 0，但你有非常多的筹码。

1. 花费 A 枚筹码将手中的数字变为 2 倍
2. 花费 B 枚筹码将手中的数字变为 3 倍
3. 花费 C 枚筹码将手中的数字变为 5 倍
4. 花费 D 枚筹码将手中的数字增加 1 或者减少 1.

你的筹码非常多，但是还是要省着用。这一场给定一个数字 N，试计算要将手中的 0 变为数字 N，最少消耗几枚筹码。

[https://atcoder.jp/contests/agc044/tasks/agc044\\_a](https://atcoder.jp/contests/agc044/tasks/agc044_a)

### 1.3.2 分析

我们可以把这个问题反过来想：手中初始是数字 N 最终变为 0。可以采取以下的操作：

1. 如果  $x$  能被 2 整除，则用  $\frac{x}{2}$  替代  $x$  并花费 A 枚筹码
2. 如果  $x$  能被 3 整除，则用  $\frac{x}{3}$  替代  $x$  并花费 B 枚筹码
3. 如果  $x$  能被 5 整除，则用  $\frac{x}{5}$  替代  $x$  并花费 C 枚筹码
4. 用  $x+1$  或  $x-1$  替代  $x$ ，花费 D 枚筹码

一种方法是，我们全用花费 D 的方法，这样从 N 到 0 的花费为  $ND$ 。另外，我们可以采用这样的策略：

$$N \xrightarrow{D} \dots \xrightarrow{D} ky \xrightarrow{k} y$$

其中  $k \in \{2, 3, 5\}$ 。需要说明的是  $y \in \{\lfloor \frac{N}{k} \rfloor, \lceil \frac{N}{k} \rceil\}$ 。事实上，如果  $y < \lfloor \frac{N}{k} \rfloor$  (同样的理由对  $y > \lceil \frac{N}{k} \rceil$  也适用)，就有如下的变换序列，比上述序列的花费更低

$$N \xrightarrow{D} \dots \xrightarrow{D} k \lfloor \frac{N}{k} \rfloor \xrightarrow{k} \lfloor \frac{N}{k} \rfloor \xrightarrow{D} \dots \xrightarrow{k} y$$

所以，我们的最优策略是直接利用  $\pm 1$  的方式由 N 到 0，或到某个  $\{\lfloor \frac{N}{k} \rfloor, \lceil \frac{N}{k} \rceil\}$  ( $k \in \{2, 3, 5\}$ )。特别地，如果记  $f(n)$  为从 N 到 0 的最小花费，那么如果我们知道了  $f(\lfloor \frac{N}{k} \rfloor)$  和  $f(\lceil \frac{N}{k} \rceil)$ ，那么  $f(n)$  便可容易计算出。

上述讨论可以用动态规划方法实现，但是还需要解决从 N 到 0 有多少状态的问题。可以证明，可达的数字将由以下两式限定

$$\lfloor \frac{N}{2^a 3^b 5^c} \rfloor, \lceil \frac{N}{2^a 3^b 5^c} \rceil$$

由于  $0 \leq a \leq 60, 0 \leq b \leq 40, 0 \leq c \leq 30$  (不然分母将大于分子)，可达的数字小于  $2 \times 61 \times 41 \times 31 = 155062$  (为了 Accepted，程序要保证  $f(n)$  总是在 n 的可达数字以内)



```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long LL;
4
5  LL A, B, C, D;
6  map<LL, LL> memo;
7  LL solve(LL N) {
8      if (N == 0) return 0;
9      if (N == 1) return D;
10     if (memo.count(N)) return memo[N];
11     LL l2 = (N/2)*2;
12     LL r2 = ((N+1)/2)*2;
13     LL l3 = (N/3)*3;
14     LL r3 = ((N+2)/3)*3;
15     LL l5 = (N/5)*5;
16     LL r5 = ((N+4)/5)*5;
17
18     LL res = 1e18;
19     if (N < res/D) res = N*D;
20     res = min(res, llabs(l2-N)*D + A + solve(l2/2));
21     res = min(res, llabs(r2-N)*D + A + solve(r2/2));
22
23     res = min(res, llabs(l3-N)*D + B + solve(l3/3));
24     res = min(res, llabs(r3-N)*D + B + solve(r3/3));
25
26     res = min(res, llabs(l5-N)*D + C + solve(l5/5));
27     res = min(res, llabs(r5-N)*D + C + solve(r5/5));
28
29     memo[N] = res;
30
31     return res;
32 }
33
34 int main() {
35     int T;
36     cin >> T;
37     for (int t = 0; t < T; t++) {
38         memo.clear();
39         LL N;
40         cin >> N;
41         cin >> A >> B >> C >> D;
42         cout << solve(N) << "\n";

```

```
43     }  
44     return 0;  
45 }
```

## 1.4 Xor Battle

### 1.4.1 问题描述

有 0 和 1 两人进行对战。对于变量  $x$  (初始为 0)，两人将按照给定的长度为  $N$  的序列  $\{A_i\}$  进行操作，长为  $N$  的 01 串  $S_i$  表示第  $i$  步由  $S_i$  号选手进行操作。第  $i$  人可以将当前变量  $x$  变成  $x \text{ XOR } A_i$ ，也可以什么都不做放弃这个  $A_i$ 。这  $N$  个  $A_i$  都用完之后如果变量  $x$  为 0，则 0 胜，否则 1 胜。假定两人对战时都会出最优的策略，请判断 0 能否取胜。

### 1.4.2 分析

この解説では，0-based index を用います．次のような DP を考えます．

$DP[i]$  = 次の条件を満たす整数  $t$  の集合． - ラウンド  $i$  (0-based) の直前で  $x = t$  であるとき，ここからゲームを続けて，最終的に  $x = 0$  になる．

この DP を後ろから埋めていきます．まず， $DP[N] = \{0\}$  です．

次に  $DP[i]$  の遷移を考えます． $S_i = 0$  の場合は簡単で，

用数组 DP (下标从 0 计) 表示。 $DP[i]$  の值是满足下列整数  $t$  的集合

首先初始时  $DP[N] = \{0\}$

接下来考虑 DP 的转移方程。 $S_i = 0$  的情况比较简单：

$$DP[i] = DP[i+1] \cup \{v \oplus A_i | v \in DP[i+1]\}$$

です．

つぎに  $S_i = 1$  の場合です．まず  $A_i \in DP[i+1]$  の場合を考えます．ラウンド  $i$  の時点で  $x \notin DP[i+1]$  なら，何もしなければよいです．逆に， $x \in DP[i+1]$  の場合， $x \oplus A_i \in DP[i+1]$  なので，操作をしても意味がありません．よって  $DP[i] = DP[i+1]$  となります．

$A_i \notin DP[i+1]$  の場合，どんな  $x$  に対しても， $x$  または  $x \oplus A_i$  の少なくとも一方は  $DP[i+1]$  に含まれません．よって， $DP[i] =$  です．

この DP は，xor の基底を管理することで，効率的に計算することができます．計算量は 1 ケースあたり  $O(N \log(\max(A_i)))$  になります．

对于  $S_i = 1$  的情况，先看  $A_i \in DP[i+1]$  时，ラウンド  $i$  の時点で  $x \notin DP[i+1]$  なら，什么都可以。而  $x \in DP[i+1]$  时， $x \oplus A_i \in DP[i+1]$  なので就没有意义。所以直接  $DP[i] = DP[i+1]$  就可以。

DP 数组在 XOR 运行下可以比较高效完成，每个 case 可以在  $O(N \log(\max(A_i)))$  完成

## 1.5 01 Unbalanced

### 1.5.1 问题描述

给出一个仅由 '0', '1' 或 '?' 构成的字符串  $S$ 。将  $S$  中的 '?' 替换为 '0' 和 '1' (各个 '?' 互不影响相互独立) 之后得到新字符串  $S$ 。定义  $S$  的不平衡度为:  $S$  中第  $l$  个字符到第  $r$  个字符中 0 与 1 的个数之差的绝对值的最大值,  $1 \leq l \leq r \leq N$ 。给定一个  $S$ , 试求一个最小  $S$  的不平衡度。

### 1.5.2 分析

## 1.6 Range Set

### 1.6.1 问题描述

有一长为  $N$  的字符串，初始全为 0。Alice 可以对这个字符串做这样的操作：

- 选择连续的  $A$  个字符置为 0
- 选择连续的  $B$  个字符置为 1

不限制 Alice 执行的次数和顺序，计算 Alice 可以得到多少种不同字符串。结果较大，输出时结果对  $10^9+7$  取余。

### 1.6.2 分析