

# ABC 156 解説

writer: gazelle, kort0n, namonakiacc

2020 年 2 月 22 日

*For International Readers: English editorial starts on page 10.*

## A: Beginner

(原案: kort0n, 準備・解説: gazelle)

コンテストの参加回数によって、内部レーティングと表示レーティングの間に成り立つ関係式が異なっています。よって、高橋君のコンテスト参加回数  $N$  が 10 以上かどうかで場合分けをして考えることにします。

### $N$ が 10 以上の場合

このとき、内部レーティングと表示レーティングは等しくなるので、入力として受け取った高橋君の表示レーティングの値を、そのまま高橋君の内部レーティングの値として出力すればいいです。

### $N$ が 10 未満の場合

このとき、

$$\text{表示レーティング} = \text{内部レーティング} - 100 \times (10 - N)$$

の関係が成り立っています。よって高橋君の内部レーティングの値は、表示レーティングの値に  $100 \times (10 - N)$  を足したものに等しいことが分かります。

以上のロジックの C++ での実装例を Listing 1 に示します。

なお Listing 2 のコードのように、min 関数を用いて if 文を使わないコードにすることも可能です。

Listing 1 C++ による実装例

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     int N, R;
6     cin >> N >> R;
7     if(N >= 10) cout << R << endl;
```

```
8     else cout << R + 100 * (10 - N) << endl;
9     return 0;
10 }
```

---

Listing 2 C++ による実装例 2

---

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     int N, R;
6     cin >> N >> R;
7     cout << R + 100 * (10 - min(10, N)) << endl;
8     return 0;
9 }
```

---

## B: Digits

(原案・準備・解説: gazelle)

整数  $N$  を  $K$  進数で表したときの桁数が  $D$  であるとしします。

このとき位取り記数法の定義より、ある整数列  $a_0, a_1, \dots, a_{D-1}$  ( $0 \leq a_i < K, a_{D-1} \neq 0$ ) が存在して、

$$N = a_{D-1}K^{D-1} + a_{D-2}K^{D-2} + \dots + a_0K^0$$

と書くことができます。ここで、

$$N = (a_{D-1}K^{D-2} + a_{D-2}K^{D-3} + \dots + a_1K^0) \times K + a_0$$

と変形できることから、 $0 \leq a_0 < K$  を考えると  $N$  を  $K$  で割ったときの商が、

$$a_{D-1}K^{D-2} + a_{D-2}K^{D-3} + \dots + a_1K^0$$

になることが分かります。 $a_{D-1} \neq 0$  なので、これは  $K$  進数で表したときの桁数が  $D-1$  になる数です。

この議論を繰り返し適用することで、 $N$  を  $K$  進数で表したときの桁数を求めることができます。つまり、 $N$  を  $K$  で割ったときの商で置き換える、という操作を  $N$  が  $0$  になるまで行ったとき、操作を行った回数が  $N$  を  $K$  進数で表したときの桁数になります (厳密には数学的帰納法を使って示します)。

この計算は愚直に実行すると、桁数に比例する時間がかかります。桁数が非常に大きい場合に実行時間制限を超えてしまいそうに見えますが、実はこの問題の入力制約下では  $N$  の桁数は多くても数十程度になります。例えば  $N = 10^9$ 、 $K = 2$  のとき、 $2^{30} = 1073741824 > 10^9$  であることから、 $N$  を  $2$  進数で表したときの桁数は高々  $30$  です。 $N$  が小さかったり  $K$  が大きい場合には桁数はこれより少なくなります。一般に整数  $N$  を  $K$  進数で表したときの桁数は、 $N$  の対数に比例します。これは  $N$  と比べて非常に小さい値です。

C++ による解法の実装例を Listing 3 に示します。

Listing 3 C++ による実装例

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     int N, K;
6     cin >> N >> K;
7     int ans = 0;
8     while(N > 0) {
9         N /= K; // N をK で割った商に置き換える
10        ans++;

```

```
11     }  
12     cout << ans << endl;  
13     return 0;  
14 }
```

---

## C: Rally

(原案: namonakiacc, 準備・解説: gazelle)

人が住んでいる座標の中で、最も小さいものを  $L$ 、最も大きいものを  $R$  とします。このとき、 $x < L$  または  $R < x$  となるような座標  $x$  で集会を開くケースを考える必要はありません。というのも、そのようなケースでは、座標  $L$  や  $R$  で集会を開くときと比べて参加するために消費する体力が小さくなることはないからです。

厳密には、例えば  $x = R + d$  ( $d > 0$ ) と書ける座標  $x$  で集会を開く場合、 $i$  番目の人が集会に参加するために消費する体力は、

$$(X_i - (R + d))^2 = ((R + d) - X_i)^2 = (R - X_i)^2 + d(R - X_i) + d^2$$

となり、最右辺の第 2 項は非負、第 3 項は正であることから、座標  $R$  で集会を開くときに比べて、真に消費する体力が増えます。 $x < L$  となるような座標  $x$  で集会を開く場合も同様です。

このアイデアを元に、集会開催場所の候補として調べる必要のある座標を高々 100 個まで減らすことができます。集会の開催場所を 1 つ固定したとき、 $N$  人が消費する体力の総和は  $O(N)$  回の演算で計算することが可能です。よって、残った開催場所の候補全てを調べて、最も消費する体力の総和が小さいものを見つけるアルゴリズムが、十分高速に動作します。

Listing 4 に C++ による解法の実装例を示します。

なお入力制約を考慮すると、入力によっては多少実行時間が遅くなる可能性はありますが、Listing 5 のように少しシンプルに書くことが可能です。

Listing 4 C++ による実装例

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 #define rep(i, c) for(int i = 0; i < (int)c; i++)
4 const int inf = 1000000000; // 10^9
5
6 int main() {
7     int N;
8     cin >> N;
9     vector<int> X(N);
10    rep(i, N) cin >> X[i];
11    int L = X[0], R = X[0];
12    rep(i, N) {
13        L = min(L, X[i]);
14        R = max(R, X[i]);
15    }
16    int ans = inf;
17    for(int i = L; i <= R; i++) {
```

```

18     int cost = 0;
19     rep(j, N) cost += (X[j] - i) * (X[j] - i);
20     ans = min(ans, cost);
21 }
22 cout << ans << endl;
23 return 0;
24 }

```

---

Listing 5 C++ による実装例 2

---

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define rep(i, c) for(int i = 0; i < (int)c; i++)
4 const int inf = 1000000000; // 10^9
5
6 int main() {
7     int N;
8     cin >> N;
9     vector<int> X(N);
10    rep(i, N) cin >> X[i];
11    int ans = inf;
12    for(int i = 1; i <= 100; i++) { // L やR の計算が不要
13        int cost = 0;
14        rep(j, N) cost += (X[j] - i) * (X[j] - i);
15        ans = min(ans, cost);
16    }
17    cout << ans << endl;
18    return 0;
19 }

```

---

## D: Bouquet

(原案・準備・解説: gazelle)

まず「 $a, b$  本の花からなる花束は作ることができない」という制約を無視して問題を解いてみます。このとき答えは  $2^n - 1$  になります。これは、種類 1 の花を選ぶ / 選ばない、種類 2 の花を選ぶ / 選ばない、...、種類  $n$  の花を選ぶ / 選ばないという枝分かれを考えると分かります (1 本も花を使わない花束を作ることができないので、1 を引きます)。  $2^n \bmod (10^9 + 7)$  は、繰り返し二乗法と呼ばれるテクニックを用いて高速に計算可能です。

ここから  $a$  本の花からなる花束、および  $b$  本の花からなる花束の種類数を引くことで、元の問題を解くことが可能です。

$a$  本の花からなる花束の種類数  $\bmod (10^9 + 7)$  は、

$${}_n C_a \bmod (10^9 + 7) = \frac{n \times (n-1) \times \dots \times (n-a+1)}{a \times (a-1) \times \dots \times 1} \bmod (10^9 + 7)$$

として計算できます。右辺の分子の値を  $X$ 、分母の値を  $Y$  と置きます。

$$\frac{X}{Y} \bmod (10^9 + 7)$$

をどのように計算すればいいのでしょうか？  $Y$  で割る、具体的には  $Y$  に掛けると  $\bmod (10^9 + 7)$  の世界で 1 になるような数  $Y^{-1}$  が何であるかを探してみます。ここで  $10^9 + 7$  が素数であることから、フェルマーの小定理より、

$$Y \times Y^{(10^9+7)-2} \bmod (10^9 + 7) = 1$$

が成り立ちます。両辺に  $Y^{-1}$  を掛けると、

$$Y^{(10^9+7)-2} \bmod (10^9 + 7) = Y^{-1}$$

になります。つまり  $\bmod (10^9 + 7)$  の世界では、 $Y$  で割ることと  $Y^{(10^9+7)-2}$  を掛けることが等価になっています (この部分の正確な議論については「競技プログラミング 剰余」などで検索してみてください)。よって、

$$\frac{X}{Y} \bmod (10^9 + 7) = X \times Y^{(10^9+7)-2} \bmod (10^9 + 7)$$

です。  $Y^{(10^9+7)-2}$  の値は、  $2^n$  と同様に繰り返し二乗法を用いて求めることができるので、この値は高速に計算可能です。

同様の方法で  $b$  本の花からなる花束の種類数  $\bmod (10^9 + 7)$  も分かるので、これらを  $2^n - 1$  から引くことで問題を解くことができました。

## E: Roaming

(原案・準備・解説: gazelle)

現在、部屋  $i$  にいる人の数を  $c_i$  として、非負整数列  $c_1, c_2, \dots, c_n$  が満たすべき条件を考えます。

まず、明らかに  $\sum_{i=1}^n c_i = n$  である必要があります。また、 $c_i = 0$  である  $i$  の個数は  $k$  以下でなくてはなりません。というのも、1 回の人の移動で、そのような  $i$  の個数は高々 1 しか増えないからです。

実は逆に、この条件を満たしている数列  $c_1, c_2, \dots, c_n$  に対して、その結果を与えるような人の移動の列が存在します。以下にこのことを示します。

$c_i = 0$  である  $i$  が 0 個の場合

まず、最初に部屋 1 にいた人が部屋 2 に移動して、そのあと  $k-2$  回、部屋 2 と 3 を行ったり来たりして、最後に部屋 1 に戻ればいいです。

$c_i = 0$  である  $i$  が 1 個の場合

一般性を失わずに  $c_1 = 0, c_2 = 2$  であると仮定します。このとき、最初に部屋 1 にいた人が  $k-1$  回、部屋 1 と 3 を行ったり来たりして、最後に部屋 2 に行けばいいです。

$c_i = 0$  である  $i$  が 2 個以上の場合

$c_i = 0$  である  $i$  の数が  $m$  ( $2 \leq m < n$ ) 個であるとして、一般性を失わずに  $c_1 = 0, c_2 = 0, \dots, c_m = 0$  であると仮定します。このとき、最初に部屋 1 にいた人が  $k-m$  回、部屋 1 と 2 を行ったり来たりして、そのあとで最初に部屋 1 から  $m$  までにいた人が 1 人ずつ、条件を満たすように番号  $m+1$  以上の部屋に移動していけばいいです。

以上より、この問題は先述の条件を満たす非負整数列  $c_1, c_2, \dots, c_n$  の数え上げに帰着されました。次に、そのような数列を数え上げる方法を考えます。

$c_i = 0$  である個数が  $m$  となるような数列の数は、

$${}^n C_m \times {}_{n-m} H_m$$

と表せます。ここで  ${}_x H_y$  は、 $x$  種類のものから重複を許して  $y$  個を選ぶ場合の数 (重複組合せ) を意味します。一般に  ${}_x H_y = {}_{x+y-1} C_{x-1}$  が成り立ちます。これは、 $x-1$  個の仕切りと  $y$  個のボールを一行に並べて、一番左の仕切りより左側にあるボールの数だけ種類 1 のものを選び、一番左の仕切りと二番目に左の仕切りとの間にあるボールの数だけ種類 2 のものを選び、……、と対応付けることが可能なことから分かります。

適切な前処理を行っておけば、この式の計算に必要な二項係数は全て  $O(1)$  で計算することができます。従って  $m = 0, 1, \dots, n-1$  の各場合について式の値を求めて、その総和を計算するアルゴリズムが十分高速に動作します。

## F: Modularness

(原案・準備・解説: gazelle)

1 つのクエリに  $O(k)$  で答えることを目指します。 $i$  番目のクエリに答えるものとして、 $n = n_i$ ,  $x = x_i$ ,  $m = m_i$  と書くことにします。あらかじめ  $d$  の要素それぞれを  $m$  で割った余りで置き換えておいても、クエリに対する答えは変わりません。以下そのように置き換えたものとして議論を進めます。

$(a_i \bmod m) < (a_{i+1} \bmod m)$  とならない、つまり  $(a_i \bmod m) = (a_{i+1} \bmod m)$  もしくは  $(a_i \bmod m) > (a_{i+1} \bmod m)$  となるような  $i$  ( $0 \leq i < n-1$ ) がいくつあるかを考えます。

$(a_i \bmod m) = (a_{i+1} \bmod m)$  となる場合

このとき、 $d_i \bmod k = 0$  になっています。そのような  $i$  の個数は、 $d$  の 0 である要素がそれぞれ数列  $a$  に何回足されるかを考えることで  $O(k)$  で計算可能です。

$(a_i \bmod m) > (a_{i+1} \bmod m)$  となる場合

このとき、 $d$  の要素が非負であること、およびあらかじめ行った前処理を考えると、 $a_{i+1}$  を  $m$  で割った商が  $a_i$  を  $m$  で割った商よりちょうど 1 大きくなっています。また商が増えるのは、そのような場合に限ることも分かります。そのような  $i$  の個数は、 $a_{n-1}$  を  $m$  で割った商から  $a_0$  を  $m$  で割った商を引くことで求めることができます。 $a_{n-1}$  の値は、 $d$  の要素がそれぞれ数列  $a$  に何回足されるかを考えることで  $O(k)$  で計算可能です。

以上より、 $(a_i \bmod m) < (a_{i+1} \bmod m)$  とならない  $i$  の個数を  $O(k)$  で求めることができます。これらを  $n-1$  から引くことで、クエリに  $O(k)$  で答えることができました。

## A: Beginner

(draft: kort0n, preparation, editorial: gazelle)

Depending on the number of participation in the contest, the relational expression between the Inner Rating and Displayed Rating is different. Therefore, we will split into case depending on whether the number of Takahashi's participation in contest,  $N$ , is 10 or more, or not.

If  $N$  is more than or equal to 10

In such case, the Inner Rating and the Displayed Rating is the same, so you can directly output the value of Takahashi's displayed rating you received as an input as the value of Takahashi's Inner Rating as an output.

If  $N$  is less than 10

In such case, a relational expression of

$$\text{Displayed rating} = \text{Inner rating} - 100 \times (10 - N)$$

holds. Therefore, it appears that the value of Takahashi's Inner rating is equal to the value of the Displayed Rating, added by  $100 \times (10 - N)$ .

We will show an implementation example of the logic mentioned above in C++ in Listing 6. Also, like the code in Listing 7, you can also write a code without if statement by using min function.

Listing 6 Sample Code in C++

---

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     int N, R;
6     cin >> N >> R;
7     if(N >= 10) cout << R << endl;
8     else cout << R + 100 * (10 - N) << endl;
9     return 0;
10 }
```

---

Listing 7 Sample Code in C++ 2

---

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     int N, R;
```

```
6     cin >> N >> R;
7     cout << R + 100 * (10 - min(10, N)) << endl;
8     return 0;
9 }
```

---

## B: Digits

(draft, preparation, editorial: gazelle)

Assume that the number of digits of integer  $N$  in base  $K$  is  $D$ .

Then, by the definition of positional notation, there exists a sequence of integer  $a_0, a_1, \dots, a_{D-1}$  ( $0 \leq a_i < K, a_{D-1} \neq 0$ ) such that

$$N = a_{D-1}K^{D-1} + a_{D-2}K^{D-2} + \dots + a_0K^0.$$

Since it can be deformed to

$$N = (a_{D-1}K^{D-2} + a_{D-2}K^{D-3} + \dots + a_1K^0) \times K + a_0,$$

given that  $0 \leq a_0 < K$ , it appears that the quotient of  $N$  divided by  $K$  is

$$a_{D-1}K^{D-2} + a_{D-2}K^{D-3} + \dots + a_1K^0.$$

Since  $a_{D-1} \neq 0$ , it has  $D - 1$  digits in base  $K$ .

By applying this discussion repeatedly, we can find the number of digits of  $N$  in base  $K$ . Therefore, when a operation of replacing an integer with its quotient divided by  $K$  is performed repeatedly until  $N$  becomes 0, The number of operations applied will be the number of digits of  $N$  in base  $K$  (strictly, it can be show by induction).

When performed naively, this calculation takes time proportional to the number of digits. If the number of digits is very large, it seems that it may exceed the time limit, but in fact under the input constraints of this problem,  $N$  has at most several dozen of digits. For example, if  $N = 10^9$ ,  $K = 2$ , since  $2^{30} = 1073741824 > 10^9$ ,  $N$  has at most 30 digits in base 2. When  $N$  is smaller or  $K$  is larger, the number of digits is less than that. Generally, the number of digits of  $N$  in base  $K$  is proportional to logarithm of  $N$ , that is far less than  $N$ .

C++ による解法の実装例を Listing 1 に示します。We will show an implementation example of solution in C++ in Listing 8.

Listing 8 Sample code in C++

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     int N, K;
6     cin >> N >> K;
7     int ans = 0;
8     while(N > 0) {
9         N /= K; // replace N with its quotient divided by K
```

```
10     ans++;
11 }
12 cout << ans << endl;
13 return 0;
14 }
```

---

## C: Rally

(draft: namonakiacc, preparation, editorial: gazelle)

Let  $L$  be the minimum coordinate, and  $R$  be the maximum coordinate, where a person lives. Then, we do not have to consider holding a meeting at coordinate  $x$  such that  $x < L$  or  $R < x$ . That is, in such case the total points of stamina is no less than when holding a meeting at coordinates  $L$  or  $R$ .

Strictly speaking, when holding a meeting at coordinate  $x$  which can be represented as  $x = R + d$  ( $d > 0$ ) for example, the points of stamina that  $i$ -th person will spend is:

$$(X_i - (R + d))^2 = ((R + d) - X_i)^2 = (R - X_i)^2 + d(R - X_i) + d^2,$$

and since the second term of the rightmost hand side is non-negative and the third term is positive, the points of stamina spent is strictly greater than when holding a meeting at coordinate  $R$ . It is the same when holding a meeting at coordinate  $x$  such that  $x < L$ .

Based on this idea, we can decrease the number of coordinates we have to consider as candidates of meeting to at most 100 places. When a place of holding the meeting is fixed, the sum of points of stamina can be calculated in a total of  $O(N)$  operations. Therefore, the algorithm of looking up all the candidates left and finding the minimum sum of points of stamina consumed is fast enough.

We will show an implementation example in C++ in Listing 9.

Also, considering the given input constraints, though it may be more or less slower, it can be rather simply like Listing 10.

Listing 9 Sample Code in C++

---

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 #define rep(i, c) for(int i = 0; i < (int)c; i++)
4 const int inf = 1000000000; // 10^9
5
6 int main() {
7     int N;
8     cin >> N;
9     vector<int> X(N);
10    rep(i, N) cin >> X[i];
11    int L = X[0], R = X[0];
12    rep(i, N) {
13        L = min(L, X[i]);
14        R = max(R, X[i]);
15    }
```

```

16     int ans = inf;
17     for(int i = L; i <= R; i++) {
18         int cost = 0;
19         rep(j, N) cost += (X[j] - i) * (X[j] - i);
20         ans = min(ans, cost);
21     }
22     cout << ans << endl;
23     return 0;
24 }

```

---

Listing 10 Sample Code in C++

---

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 #define rep(i, c) for(int i = 0; i < (int)c; i++)
4 const int inf = 1000000000; // 10^9
5
6 int main() {
7     int N;
8     cin >> N;
9     vector<int> X(N);
10    rep(i, N) cin >> X[i];
11    int ans = inf;
12    for(int i = 1; i <= 100; i++) { // No need of calculations of L and R
13        int cost = 0;
14        rep(j, N) cost += (X[j] - i) * (X[j] - i);
15        ans = min(ans, cost);
16    }
17    cout << ans << endl;
18    return 0;
19 }

```

---

## D: Bouquet

(draft, preparation, editorial: gazelle)

First, we will solve the problem ignoring the condition that the number of flowers in the bouquet cannot be  $a$  or  $b$ . In such case, the answer will be  $2^n - 1$ . This can be understood by considering a tree diagram of choosing / not choosing flower 1, choosing / not choosing flower 2,  $\dots$ , choosing / not choosing flower  $n$  (since a bouquet with no flowers cannot be made, it is subtracted by 1).  $2^n \bmod (10^9 + 7)$  can be calculated rapidly by a technique called fast exponentiation algorithm.

By subtracting from it the number of bouquet with  $a$  flowers and the number of bouquet with  $b$  flowers, the original problem can be solved.

The number of bouquet with  $a$  flowers  $\bmod (10^9 + 7)$  can be calculated by

$${}_n C_a \bmod (10^9 + 7) = \frac{n \times (n-1) \times \dots \times (n-a+1)}{a \times (a-1) \times \dots \times 1} \bmod (10^9 + 7).$$

We denote the numerator of the left hand side by  $X$ , and the denominator by  $Y$ . How can we calculate

$$\frac{X}{Y} \bmod (10^9 + 7)?$$

Let's try dividing by  $Y$ , specifically let's find  $Y^{-1}$ , such integer that when multiplied by  $Y$  it will become 1 in a world of  $\bmod (10^9 + 7)$ . Here, since  $10^9 + 7$  is a prime, by Fermat's little theorem, it holds that

$$Y \times Y^{(10^9+7)-2} \bmod (10^9 + 7) = 1.$$

multiplying  $Y^{-1}$  to the both sides, we obtain

$$Y^{(10^9+7)-2} \bmod (10^9 + 7) = Y^{-1}.$$

Therefore, in a world of  $\bmod (10^9+7)$ , dividing by  $Y$  and multiplying by  $Y^{(10^9+7)-2}$  are equivalent (for more strict discussion, please search by "competitive programming modulo" or something).

Therefore, it holds that

$$\frac{X}{Y} \bmod (10^9 + 7) = X \times Y^{(10^9+7)-2} \bmod (10^9 + 7).$$

The value of  $Y^{(10^9+7)-2}$  can be calculated by fast exponentiation just like  $2^n$ , so this value can be calculated fast.

We can similarly calculate the number of bouquets with  $b$  flowers  $\bmod (10^9 + 7)$ , So by subtracting them from  $2^n - 1$ , the problem could be solved.

## E: Roaming

(draft, preparation, editorial: gazelle)

Let  $c_i$  be the number of people in room  $i$  currently, and let us consider the condition that the sequence of non-negative integers  $c_1, c_2, \dots, c_n$  should satisfy.

First, obviously  $\sum_{i=1}^n c_i = n$  should hold. Also, the number of  $i$  such that  $c_i = 0$  should be no more than  $k$ . That is, in one move, such  $i$  increases by at most 1.

Actually, conversely, for any sequence  $c_1, c_2, \dots, c_n$  such that satisfies the condition mentioned above, there exists a sequence of move that gives such results. We will prove that.

If no  $i$  satisfies  $c_i = 0$

If the person who initially was in room 1 first moves to room 2, then moves back and forth between room 2 and 3 for  $k - 2$  times, then at last goes back to room 1, then the final state satisfies the conditions.

If there exists exactly one  $i$  such that  $c_i = 0$

Without loss of generality, we assume that  $c_1 = 0, c_2 = 2$ . In such case, if the person who initially was in room  $i$  moves back and forth between room 1 and 3 for  $k - 1$  times, and then at last goes to room 2, then the final state satisfies the conditions.

If there exist more than one  $i$  such that  $c_i = 0$

Assume that the number of  $i$  such that  $c_i = 0$  is  $m$  ( $2 \leq m < n$ ), and without loss of generality, we assume that  $c_1 = 0, c_2 = 0, \dots, c_m = 0$ . In such case, if the person who initially was in room 1 moves back and forth between room 1 and 2, then each of the people who were initially in between room 1 to  $m$  can move to the room with index more than  $m$  so that the final state satisfies the conditions.

Therefore, the problem has been attributed to a problem of counting the number of sequence of non-negative integers  $c_1, c_2, \dots, c_n$  such that satisfies the condition mentioned above. Next, we will consider how to count such sequences.

The number of sequence such that the number of  $i$  such that  $c_i = 0$  is  $m$  can be represented as:

$${}^n C_m \times {}_{n-m} H_m,$$

where  ${}_x H_y$  denotes the number of combination of choosing  $y$  items from  $x$  types of items, possibly with duplicates of types (multiset coefficient). In general, it holds that  ${}_x H_y = {}_{x+y-1} C_{x-1}$ . This can be understood from the fact that it can be associated to the arrangement of  $x - 1$  bars and  $y$  balls in a row, where the number of balls left to the leftmost bar represents the number of the first type, the number of balls between the leftmost and the second leftmost bar

represents the number of the second type,  $\dots$  and so on.

With proper precalculations, all the binomial coefficients needed to calculate this expression can be calculated in a  $O(1)$  time each. Therefore, an algorithm of calculating the expression for each  $m = 0, 1, \dots, n - 1$ , and then finding the sum of them works fast enough.

## F: Modularness

(draft, preparation, editorial: gazelle)

We aim at answering each query in  $O(k)$  time. Consider answering  $i$ -th query, and we denote  $n = n_i$ ,  $x = x_i$ ,  $m = m_i$ . Replacing each element of  $d$  with its remainder by  $m$  does not change the answer for the query. Hereinafter we assume that such applied such replacement.

We will consider how many  $i$  ( $0 \leq i < n - 1$ ) do not satisfy  $(a_i \bmod m) < (a_{i+1} \bmod m)$ , in other words, such  $i$  that  $(a_i \bmod m) = (a_{i+1} \bmod m)$  or  $(a_i \bmod m) > (a_{i+1} \bmod m)$ .

If  $(a_i \bmod m) = (a_{i+1} \bmod m)$

In such case,  $d_i \bmod k = 0$ . The number of such  $i$  can be calculated in  $O(K)$  time by considering how many times each of elements of  $d$  of value 0 is added to  $a$ .

If  $(a_i \bmod m) > (a_{i+1} \bmod m)$

In such case, given that the elements of  $d$  are non-negative, and also considering the replacement done beforehand, it appears that the quotient of  $a_{i+1}$  divided by  $m$  is exactly larger by 1 than the quotient of  $a_i$  divided by  $m$ . Also, it can be seen that the quotient increases only in such cases. The number of such  $i$  can be calculated by subtracting the quotient of  $a_0$  divided by  $m$  from the quotient of  $a_{n-1}$  divided by  $m$ . The value of  $a_{n-1}$  can be calculated in  $O(k)$  time by considering how many times each element of  $d$  are added to  $a$ .

Therefore, we can count the number of such  $i$  that does not satisfy  $(a_i \bmod m) < (a_{i+1} \bmod m)$ . By subtracting this from  $n - 1$ , we could answer the query in a total of  $O(k)$  time.