ABC 156 解説

writer: gazelle, kort0n, namonakiacc

2020 年 2 月 22 日

For International Readers: English editorial will be published in a few days.

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

```
#include<bits/stdc++.h>
using namespace std;

int main() {
    int N, R;
    cin >> N >> R;
    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 $a_0, a_2, ..., a_{D-1}$ $(0 \leq a_i < K, a_{D-1} \neq 0)$ が存在して、

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

と書くことができます。ここで、

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

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、

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

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 1 に示します。

Listing 3　C++ による実装例

```
#include<bits/stdc++.h>
using namespace std;

int main() {
    int N, K;
    cin >> N >> K;
    int ans = 0;
    while(N > 0) {
        N /= K; // N をK で割った商に置き換える
        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 1 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 2 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

```
#include<bits/stdc++.h>
using namespace std;
#define rep(i, c) for(int i = 0; i < (int)c; i++)
const int inf = 1000000000; // 10^9

int main() {
    int N;
    cin >> N;
    vector<int> X(N);
    rep(i, N) cin >> X[i];
    int L = X[0], R = X[0];
    rep(i, N) {
        L = min(L, X[i]);
        R = max(R, X[i]);
    }
    int ans = inf;
    for(int i = L; i <= R; i++) {
```

```
        int cost = 0;
        rep(j, N) cost += (X[j] - i) * (X[j] - i);
        ans = min(ans, cost);
    }
    cout << ans << endl;
    return 0;
}
```

Listing 5　C++ による実装例 2

```
#include<bits/stdc++.h>
using namespace std;
#define rep(i, c) for(int i = 0; i < (int)c; i++)
const int inf = 1000000000; // 10^9

int main() {
    int N;
    cin >> N;
    vector<int> X(N);
    rep(i, N) cin >> X[i];
    int ans = inf;
    for(int i = 1; i <= 100; i++) { // L やR の計算が不要
        int cost = 0;
        rep(j, N) cost += (X[j] - i) * (X[j] - i);
        ans = min(ans, cost);
    }
    cout << ans << endl;
    return 0;
}
```

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はできないので、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、

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

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 $c_1, c_2, ..., c_n$ 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 $c_1, c_2, ..., c_n$ 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 $c_1 = 0,\ c_2 = 0,\ ...,\ c_m = 0$ 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、そのあとで部屋 1 から m までにいる人が 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 $c_1, c_2, ..., c_n$ 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、

$$ {}_n\mathrm{C}_m \times {}_{n-m}\mathrm{H}_m $$

と表せます。ここで ${}_x\mathrm{H}_y$ 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 ${}_x\mathrm{H}_y = {}_{x+y-1}C_{x-1}$ が成り立ちます。これは、$x-1$ 個の仕切りと y 個のボールを一列に並べて、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も分かります。

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 $m = 0, 1, ..., n-1$ 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。