# ABC 165 解説

beet, DEGwer, kort0n, sheyasutaka, tozangezan, ynymxiaolongbao

2020 年 5 月 2 日

*For International Readers: English editorial starts on page 7.*

## A: We Love Golf

いろいろな方法がありますが，ここでは $B$ 以下の最大の $K$ の倍数を求め，それが $A$ 以上かどうか確かめる方法を採ってみましょう．

実装はたとえば以下のようになります．

Listing 1　C での実装例

```c
#include <stdio.h>

int main(void) {
        int k, a, b;

        scanf("%d", &k);
        scanf("%d%d", &a, &b);

        int largest = (b / k) * k;
        if (a <= largest) {
                puts("OK");
        } else {
                puts("NG");
        }

        return 0;
}
```

## B: 1%

今持っている金額を $P$ とすると、$P < X$ のあいだ $P$ に $\left\lfloor \frac{P}{100} \right\rfloor$ を足せばよいです。
入力例 2 からもわかるように、 ループ回数は高々 3760 回となり、十分高速です。
64bit 整数を扱う言語ではオーバーフローに注意してください。
($P' = \left\lfloor \frac{P \times 101}{100} \right\rfloor$ のように計算するとオーバーフローする場合があります)
以下は C++ による回答例です。

```
#include <bits/stdc++.h>
using namespace std;
int main(){
  long long X;
  cin >> X;
  long long P = 100, step = 0;
  while(P < X){
    P += P / 100;
    step++;
  }
  cout << step << endl;
  return 0;
}
```

## C: Many Requirements

結論から述べると, 問の条件を満たす数列を全探索します.
問の条件を満たす数列は, $N$ 個の仕切りと $M-1$ 個のボールを並び替えた列と一対一に対応します.
実際, 各列に対して,

$$A_i = i \text{ 番目の仕切りより左側に存在するボールの数} + 1$$

とすれば, 前述の一対一対応が得られます.
$N$ 個の仕切りと $M-1$ 個のボールを並び替える列の数は, ${}_{N+M-1}\mathrm{C}_N$ 通り存在します ($N+M-1$ 個の場所から仕切りを置く $N$ 箇所を選ぶ方法の数え上げに相当).
各数列を深さ優先探索で生成する場合, 各数列の生成に要する時間は $O(N)$ ですから, 全ての数列の生成に要する時間は $O\left(N\,{}_{N+M-1}\mathrm{C}_N\right)$ です.
数列を決めればその数列のスコアは $O(Q)$ で計算出来ます.
以上より, この問題は $O\left((N+Q)\,{}_{N+M-1}\mathrm{C}_N\right)$ で解けました.

## D: Floor Function

$f(x) = floor(Ax/B) - A \times floor(x/B)$ とします。

$f(x+B) = f(x)$ であることは実際に代入することで容易に分かります。なので、$0 \leq x \leq B-1$ の場合のみを考えます。

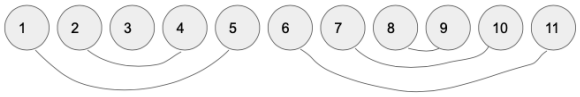
このとき、$f(x) = floor(Ax/B) - A \times floor(x/B) = floor(Ax/B)$ で、$floor(Ax/B)$ は広義単調増加なので、与えられた制約 $(0 \leq x \leq B-1,\ 0 \leq x \leq N)$ の中で最も大きい $x$ について、$f(x)$ が求める最大値です。

よって、答えは $f(min(B-1, N))$ です。

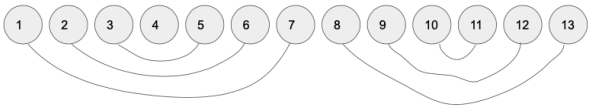
## E:Rotation Matching

$N = 2 \times M + 1$ のケースを考えます。すると、以下のような割り振りが条件を満たすことがわかります。ただし、図では、線で結ばれた数字を同じ対戦場に割り振ることを意味します。

$M$ が奇数のとき（具体例は $M = 5$ のとき）




$M$ が偶数のとき（具体例は $M = 6$ のとき）




$N$ が $2 \times M + 1$ より大きい場合でも、このような割り振りが条件を満たします。

## F: LIS on Tree

結論から述べると、この問題は最長増加部分列問題を解く動的計画法に巻き戻しのテクニックを適応させることで解くことができます。

はじめに、最長増加部分列問題 (Longest Increasing Subsequence) を解く動的計画法について説明します。LIS には様々なアプローチがありますが、二分探索を用いた動的計画法が最も有名です。$dp_i$ を長さが $i$ である増加部分列における最終要素の最小値（存在しない場合は無限大）として、$dp_1, dp_2, .., dp_N$ を無限大に初期化し、$dp_i$ の値が $A_j$ 以上になる最も小さな $i$ に対して $dp_i$ を $A_j$ の値に更新するということを $j = 1, 2, .., N$ について行っていくといった方法です。

次に、巻き戻しのテクニックについて説明します。ここでの巻き戻しとは、変数の値の更新と質問を次々に処理するという状況の中で、直前の更新をする前の状態に戻すことをいいます。以下のような処理をすることで実現できます。

- ある変数の値を更新したとき、
  - どの変数を更新したか
  - その変数の元の値は何だったか

  という情報を stack に push する。
- 巻き戻しをするとき、stack の top に示された変数をその元の値に変更し、stack を pop する。

最後に、この問題への解法を説明します。頂点 1 を始点として深さ優先探索をし、各頂点に対応する再帰関数内では、はじめに $dp$ の値を更新して答えを求めた後、隣接する頂点に対応する再帰関数を呼び出し、最後に $dp$ の値を巻き戻すことで、すべての頂点までの答えを正しく求めることができます。計算量は $O(NlogN)$ です。

## A: We Love Golf

There are many ways to solve this problem, but let us introduce the following way: find the maximum multiple of $K$ which is less than or equal to $B$, and check if it is equal to $A$.

The implementation would be as follows for example.

Listing 2 Sample Code in C

```
#include <stdio.h>

int main(void) {
        int k, a, b;

        scanf("%d", &k);
        scanf("%d%d", &a, &b);

        int largest = (b / k) * k;
        if (a <= largest) {
                puts("OK");
        } else {
                puts("NG");
        }

        return 0;
}
```

## B: 1%

Let $P$ be the amount of money he has, then it is sufficient to add $\left\lfloor \frac{P}{100} \right\rfloor$ to $P$ while $P < X$.

As it can be seen from Sample Input 2, the maximum number of loops is 3760 times, which is fast enough.

In the language which treats 64-bit integers, be careful of overflows.

(Calculating like $P' = \left\lfloor \frac{P \times 101}{100} \right\rfloor$ may lead to overflow.)

The following is a sample code in a C++.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
  long long X;
  cin >> X;
  long long P = 100, step = 0;
  while(P < X){
    P += P / 100;
    step++;
  }
  cout << step << endl;
  return 0;
}
```

## C: Many Requirements

To come to the point, perform a brute-force search through all the sequences that satisfies the conditions given in the problem.

A sequence that satisfies the conditions corresponds one-to-one to a rearranged sequence of $N$ bars and $M-1$ balls. Actually, for each sequence, let

$$A_i = \text{the number of balls that are left to the } i\text{-th bar},$$

then the aforementioned one-to-one correspondence can be obtained. The number of arrangements of $N$ bars and $M-1$ balls is ${}_{N+M-1}\mathrm{C}_N$ (which is equivalent to counting the number of ways of choosing the positions of $N$ bars out of $N+M-1$ places).

When generating each sequence with depth-first searching, each sequence needs a total of $O(N)$ time to be generated, so the total time needed to generate all the sequences is $O(N\,{}_{N+M-1}\mathrm{C}_N)$.

Once a sequence is fixed, its score can be calculated in a total of $O(Q)$ time.

Therefore, this problem could be solved in a total of $O((N+Q)\,{}_{N+M-1}\mathrm{C}_N)$ time.

## D: Floor Function

Let us define$f(x) = floor(Ax/B) - A \times floor(x/B)$.

It can be seen easily that $f(x + B) = f(x)$ by actually assigning. So, we consider only $0 \leq x \leq B - 1$.

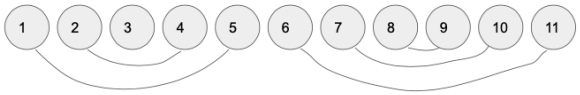
Then, $f(x) = floor(Ax/B) - A \times floor(x/B) = floor(Ax/B)$, and since $floor(Ax/B)$ is monotonically non-decreasing, so for the maximum $x$ within the given constraints ($0 \leq x \leq B-1$, $0 \leq x \leq N$), $f(x)$ is the desired maximum value.

Therefore, the answer is $f(min(B - 1, N))$.

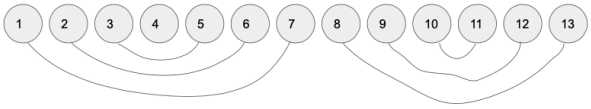
## E:Rotation Matching

Let us consider the case where $N = 2 \times M + 1$. Then, it appears that the following assignment satisfies the conditions. Note that in the figure a line that connects two numbers denote that they are assigned to the same arena.

When $M$ is odd (specifically, when $M = 5$)




When $M$ is odd (specifically, when $M = 6$)




Even when $N$ is greater than $2 \times M + 1$, such assignments still satisfies the conditions.

## F: LIS on Tree

To come to the point, this problem can be solved by applying rollback technique to the dynamic programming of solving the longest increasing subsequence problem.

First, we will explain about the dynamic programming to solve Longest Increasing Subsequence problem. Among various approaches of LIS, the most famous one is to use binary searching. Specifically, this can be done as follows: let $dp_i$ be the minimum last element of increasing subsequence of length $i$ (or infinity if not exists) initialize $dp_1, dp_2, .., dp_N$ with infinity, then for each $j = 1, 2, .., N$, find the smallest $i$ such that $dp_i$ is greater than or equal to $A_j$, and then update $dp_i$ to $A_j$

Next, we will explain about the rollback technique. Here the rollback refers to, when processing updates of values of variables and questions one after another, withdrawing the changes as it was before the latest changes. This can be achieved by the following processes:

- When the value of a variable has been updated, record
  - which variable has been updated and
  - what was the original value of the variable

  and push them to a stack.
- When rolling back, update the variable indicated by the top of the slack to the original value, and pop the stack.

At last, we will explain how to solve this problem. Perform a depth-first searching starting from vertex 1, and in the recurrence function corresponding to each vertex, first update the value of dp and find the answer, then call the recurrence functions corresponding to the adjacent vertices, and finally roll back the value of dp; this way, you can find the answer for all the vertices. The time complexity is $O(NlogN)$.