AtCoder Regular Contest 079 Editorial

Kohei Morita(yosupo)

平成 29 年 7 月 29 日

A: ABCxxx

まず、標準入力から n を入力します。
次に、ABC という文字列を出力します。
最後に、最初に入力した n を出力し、改行文字を出力します。
C++のコード例は以下です。

```cpp
#include <iostream>

using namespace std;

int main() {
    int n;
    cin >> n; // n を入力
    cout << "ABC"; // ABC を出力
    cout << n; // n を出力
    cout << endl; // 改行文字
    return 0;
}
```

B: Break Number

たとえば、2 で 3 回割れる数のうち最も小さいものはなんでしょうか？

これは、2 を 3 回かけた数、つまり 8 です。

同様に、2 で k 回割れる数のうち最も小さいものは 2 を k 回かけた数です。

2 を k 回かけた数、という形になっているのは、100 以下だと $1, 2, 4, 8, 16, 32, 64$ の 7 種類しかありません。

よって、この 7 種類のうち、N 以下で最も大きいものを出力すれば良いです。

C: Cat snuke and a voyage

島 $i(i = 2, 3, ..., N-1)$ について、島 1 と島 i、島 i と島 N のどちらの間にも定期便が通っているような i が存在するかどうかを調べれば良いです。

最初に辺をハッシュマップに入れると、島 1 と島 i の間に辺があるかを $O(1)$ で調べることができます。

他にも、長さ N の boolean 配列を 2 つ用意して、島 1 と島 i の間に辺があるか、島 i と島 N の間に辺があるか、を格納しても良いです。こちらのほうが定数倍が速いです。

よって、$O(N)$ で判定を行うことができます。

辺の前処理に $O(M)$ かかるため、まとめて $O(N + M)$ です。

なお、ダイクストラ法というアルゴリズムを使用すると、最小でいくつの定期便を使えば島 N にたどり着けるかも調べることができます。

D: Decrease (Contestant ver)

操作を逆から考えます。

すると、1 つ要素を選び、値を N 増やす。そしてそれ以外を 1 減らす。となります。更に条件として、

1. 逆操作後に、選んだ要素の値が最大になっている
2. 逆操作後に、選ばれなかった要素の値が -1 以下になっていない

を満たしている必要があります。

すべての要素が 0 以上 $N-1$ の数列から、以上の条件を満たしながら K 回逆操作を行うことが出来れば、その数列が答えとなることが示せます。

結論から言うと、$0, 1, 2, ..., N-1$ という数列を考え、$1, 2, ..., N, 1, 2, ..., N, 1, 2, ...$ 番目の順で逆操作を行えばよいです。

これを発想するのは難しいですが、これが以上の条件を満たすことを示すのはあまり難しくありません。

$1, 2, ..., N$ 番目に 1 回ずつ逆操作を行うと、全体が $+1$ されることに注目すると、最後のたかだか $N-1$ 回の操作のみ真面目にシミュレーションすればよいので、十分高速です。

E: Decrease (Judge ver)

> 数列のうち最も大きい要素を求める、複数ある場合はどれか 1 つ選ぶ。この要素の値を N 減らす。これ以外の要素の値を 1 増やす。

という操作を

> 数列のうち N 以上の要素を好きに 1 つ選ぶ。この要素の値を N 減らす。これ以外の要素の値を 1 増やす。

としても、操作回数に変わりはないという性質が重要です。

これは、直感的にはあまりおかしくは感じないと思うのですが、証明するのはやや大変です。

ある手順で操作をすると A 回で全部の要素が $N-1$ 以下になったとします。このとき、どのような手順で操作しても A 回操作できたならば、その時点での数列は一意になります。

なぜならば、この操作というのは $mod(N+1)$ で考えるとすべての要素に $+1$ という操作になります。よってどのような手順で操作しても $mod(N+1)$ は不変です。また、どのような手順で操作をしても $\sum a_i$ は不変で、操作回数のみに依存します。

更に、すべての要素が 0 以上 $N-1$ 以下になる手順が存在することより、操作後の数列が一意になることがわかります。

以上より、どのような手順で操作をしても操作回数が一意になることが示せました。

この性質により、

> $S = \sum_{i=1}^{N} \lfloor a_i/N \rfloor$ とする。各 i について a_i を $\lfloor a_i/N \rfloor N$ 減らし、$S - \lfloor a_i/N \rfloor$ 増やす。

という行動を考えることができます。

これは、S 回の操作をまとめて考えているだけです。内訳は、i 番目の要素には $\lfloor a_i/N \rfloor$ 回操作をしています。

この行動を、すべての要素が $N-1$ 以下になるまで繰り返せば答えが求まります。

そして、実はこれは十分高速であることが示せます。

行動前の数列を a_i、行動後の数列を b_i とします。

$S = \sum \lfloor a_i/N \rfloor \geq \sum (a_i - (N-1))/N$ ゆえ、

$\sum b_i \leq \sum a_i - \sum (a_i - (N-1))/N$

よって

$\sum (b_i - (N-1)) \leq \sum (a_i - (N-1))(1 - 1/N) = (1 - 1/N) \sum (a_i - (N-1))$

以上より、$\sum(a_i-(N-1))$ は指数的に減っていくことがわかり、この指数の底は $(1-1/N)\leq 49/50$ であることがわかります。

x を行動回数として、

${}^{50\times(10^{16}+1000)}/_{(49/50)^x}<1$、つまり

$(49/50)^x>50\times(10^{16}+1000)$ となるくらい行動すると、

$\sum(a_i-(N-1))$ が 0 以下となります。なお、この x を計算すると 2000 弱となります。

そしてこのとき、

$\sum a_i\leq N(N-1)\leq 2450$ となり、

1 回の行動で $\sum a_i$ が 1 は減ることを考えると、

ここからの行動回数はかならず 2450 回以下です。

よって、多くとも 4450 回ほど行動を行えば操作は終了し、1 回の行動は $O(N)$ でシミュレーションできるため、十分間に合います。

F: Namori Grundy

まず、定義より、グラフはどのような形をしているかを考えます。

これは、サイクルが 1 つあり、サイクルの各頂点から木が生えているグラフになっています。

また、a_i は、頂点 i から辺が伸びている頂点に書かれていない値のうち最小です。(これは Grundy Numbers と呼ばれる数の定義に酷似しており、今回の問題タイトルの由来となっています)

まず、サイクルに含まれない頂点たちについては、葉からどんどん削っていく幅優先探索などを使い、簡単に a_i を定めることができます。

サイクルに含まれる頂点の値を考えます。まず、サイクルの頂点のうち 1 つを適当に選びます。そして、この頂点から辺が伸びている頂点で、サイクルに含まれないものに書かれていない値のうち、最小と 2 番目に小さいものを求めます。

すると、この頂点の値はこの 2 つのうちどちらかになることがわかります。これは、この頂点から辺が伸びている、サイクルに含まれる頂点の値で場合分けをするとわかります。

そしてこの頂点の値を定めると、他のサイクルの頂点の値もすべて決まるため、すべて決めた後に整合性を保っているか判定すれば良いです。

計算量は O(N) となります。

AtCoder Regular Contest 079 Editorial

Kohei Morita(yosupo)

July 29, 2017

A: ABCxxx

```
#include <iostream>

using namespace std;

int main() {
    int n;
    cin >> n;
    cout << "ABC";
    cout << n;
    cout << endl;
    return 0;
}
```

B: Break Number

```
#include <iostream>

using namespace std;

int main() {
    int n;
    cin >> n;
    for (int i = 0; i < 20; i++) {
        if (n < (1<<(i+1))) {
            cout << (1<<i) << endl;
            break;
        }
    }
    return 0;
}
```

C: Cat snuke and a voyage

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    int n, m;
    cin >> n >> m;

    vector<bool> vis(n+1, false);
    bool ans = false;
    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        assert(1 <= a && a < b && b <= n);
        if (a == 1) {
            if (vis[b]) {
                ans = true;
            }
            vis[b] = true;
        } else if (b == n) {
            if (vis[a]) {
                ans = true;
            }
            vis[a] = true;
        }
    }

    if (!ans) cout << "IM";
    cout << "POSSIBLE" << endl;
    return 0;
}
```

D: Decrease (Contestant ver)

Consider the inverse of an operation.

In each inverse-operation, you choose an element and increase it by N, and decrease each of the other elements by 1. Furthermore, the following conditions must be satisfied:

1. After the inverse-operation, the chosen element must become the largest.
2. After the inverse-operation, all elements must be non-negative.

Let's start from a sequence whose elements are betwen 0 and $N-1$, inclusive. If you can perform inverse-operations K times on this sequence, that is a valid answer for this problem.

For example, if you start from the sequence $\{0, 1, 2, ..., N-1\}$, you can perform the inverse-operations infinite number of times, by choosing elements $1, 2, ..., N, 1, 2, ..., N, 1, 2, ...$ in this order. (You should use $N = 50$ to fit within the constraints of output.)

Notice that if you perform the inverse-operations once for each of the elements $1, 2, ..., N$, each element will be incremented by one. Using this, you can easily simulate $floor(K/N) * K$ steps. Also, you can simulate the remaining $K\%N$ steps by brute force.

E: Decrease (Judge ver)

In each operation, the sum of all elements in the sequence decreases by exactly one. Let S be the sum of all elements in the initial sequence. Until the maximum becomes at most $N-1$, we must perform at least $S-N^2$ operations.

Let's perform X operations, for a given large X. Instead of performing the operations as they are described in the statement,

- We first increase each element by X.
- Then, repeat the following X times: choose the maximum element, and decrease it by $D = N+1$.

How can we perform the second step quickly? By binary search, we can find the smallest integer Y such that

$$T = floor(\max\{(a_1 - Y)/D, 0\}) + \ldots + floor(\max\{(a_N - Y)/D, 0\}) \ldots X \quad (1)$$

Once we find this Y, we can perform T steps at once, by replacing each a_i by $a_i - floor(\max\{(a_i - Y)/D, 0\})$. Since $X - T \leq N$, we can simulate the remaining $X - T$ steps by brute force.

After the simulation of $S - N^2$ steps, the sum of elements will be at most N^2, so there will be at most N^2 remaining steps/ Therefore, we can handle the remaining part by straightforward simulations.

F: Namori Grundy

It is well-known that in this type of graph, there is exactly one cycle in the graph, and from each vertex on the cycle a tree may "grow".

We want to assign integers a_i to each vertex, such that a_i is the smallest non-negative integer that doesn't appear in $a_{v_1}, \ldots, a_{v_k}$, where $v_1, \ldots, v_k$ are the destinations of edges from the vertex i. (This is known as Grundy Numbers).

Let's call a vertex in the cycle "cycle vertex", and call the others "forest vertex".

First, for each forest vertex, we can uniquely determine their Grundy Numbers in the order from leaves to roots.

How can we determine the Grundy Number of a cycle vertex v? Let $c_1, \ldots, c_k$ be forest vertices that are direct children of v, and let w be the cycle vertex that follows v. Then, a_v must be the smallest non-negative integer that doesn't appear in the set $a_{c_1}, \ldots, a_{c_k}, a_w$. We know the values of $a_{c_1}, \ldots, a_{c_k}$, but we don't know the value of a_w. However, regardless of the value of a_w, there are only two candidates for a_v: one of the two smallest integers that don't appear in $a_{c_1}, \ldots, a_{c_k}$.

Now, choose an arbitrary cycle vertex x. There are two candidates for a_x, so we try both. Once we fix a_x, we can determine all Grundy Numbers along the cycle, and we can verify if the Grundy Numbers we get are valid.

This solution works in $O(N)$.