ABC 161 解説

chokudai, evima, kyopro_friends, latte0119,
namonakiacc, sheyasutaka, tozangezan, ynymxiaolongbao

2020 年 4 月 4 日

For International Readers: English editorial starts on page 7.

A: ABC swap

実際にシミュレーションを行えばよいです。以下は C++ における実装例です。

```
#include<iostream>
using namespace std;

int main(){
        int a,b,c;
        cin>>a>>b>>c;
        swap(a,b);
        swap(a,c);
        cout<<a<<" "<<b<<" "<<c<<endl;
        return 0;
}
```

B: Popular Vote

M 位の商品が総得票数の $\frac{1}{4M}$ 以上の票を獲得していれば YES、そうでなければ NO です。これは得票数順にソートすることで判定できます。

```
N, M = map(int,input().split())
A = list(map(int,input().split()))
A.sort(reverse=True)
S = sum(A)
if A[M-1] >= S / (4*M):
    print("Yes")
else:
    print("No")
```

その他、「総得票数の $\frac{1}{4M}$ 以上の票を獲得している商品が M 個以上あれば YES、そうでなければ NO」として解くこともできます。この場合ソートは不要です。

```
N, M = map(int,input().split())
A = list(map(int,input().split()))
S = sum(A)
cnt = 0
for a in A:
    if a >= S / (4*M):
        cnt += 1
if cnt >= M:
    print("Yes")
else:
    print("No")
```

なお、C++,Java などの言語を利用している場合、除算の挙動に注意してください (整数型変数同士の除算では、結果は整数に切り捨てられます)。

C: Replacing Integer

x が K 以上のとき、操作を行うことで $x-K$ となります。すなわち、N から N/K 回操作を行うことで、整数は N を K で割った余りとなります。

N を K で割った余りを t とします。t に操作を行うと、 $K-t$ となります。$K-t$ に操作を行うと、t に戻るだけです。すなわち K 以下の値として取りうるものは t と $K-t$ のいずれかのみとなります。

よって答えは t と $K-t$ のうち小さい方、すなわち N を K で割った余りと、$K-(\mathrm{N}\text{を}\mathrm{K}\text{で割った余り})$ のうち小さい方です。

D: Lunlun Number

この問題は、Queue というデータ構造を用いることで効率的に解くことができます。まず、空の Queue を 1 つ用意し、$1, 2, ..., 9$ を順に Enqueue します。それから、以下の操作を K 回行います。

- Queue に対して Dequeue を行う。取り出した要素を x とする。
- $x \, mod \, 10 \neq 0$ なら、 $10x + (x \, mod \, 10) - 1$ を Enqueue する。
- $10x + (x \, mod \, 10)$ を Enqueue する。
- $x \, mod \, 10 \neq 9$ なら、 $10x + (x \, mod \, 10) + 1$ を Enqueue する。

K 回目の操作において取り出した数が、 K 番目の Lunlun Number となっています。

E:Yutori

ある期間内に働く日数を最大化するためには、前から貪欲に働く日を決めるのが最適です。したがって前から貪欲に働く日を決めた場合を考えることで「x 回目に働く日は $L[x]$ 日目以降」という配列 L を求めることができます。同様に後ろから貪欲に働く日を決めた場合を考えることで「x 回目に働く日は $R[x]$ 日目以前」という配列 R を求めることができます。i 日目に必ず働くのは、$L[x] = R[x] = i$ となる x が存在するときそのときに限るので、この問題は $O(N)$ で解けました。

F:Division or Substraction

割り算の操作を 1 度もしない場合、操作によって $N \bmod K$ は変化しません。したがって、最終的に 1 になるためには $N \bmod K = 1$ であることが必要十分です。そのような K は $N - 1$ の約数のうち、1 でないもの全てであり、その個数は $O(\sqrt{N})$ で求めることができます。

割り算の操作を 1 度以上する場合、そのような K は N の約数です。実際に K で割り切れなくなるまで操作をしたあと、$N \mod K = 1$ になっているかを確かめることにより、各約数毎に $O(\log N)$ で判定することができます。

よって以上により問題が解けました。N および $N - 1$ の約数を求める部分がボトルネックとなり、計算量は $O(\sqrt{N})$ となります。

A: ABC swap

You can actually perform the simulation. The following is a sample code in C++.

```
#include<iostream>
using namespace std;

int main(){
	int a,b,c;
	cin>>a>>b>>c;
	swap(a,b);
	swap(a,c);
	cout<<a<<" "<<b<<" "<<c<<endl;
	return 0;
}
```

B: Popular Vote

If the M-th popular item is with more than or equal to $\frac{1}{4M}$ of the total number of votes, then the answer is YES, otherwise the answer is NO. This can be checked by sorting in the order of votes received.

```
N, M = map(int,input().split())
A = list(map(int,input().split()))
A.sort(reverse=True)
S = sum(A)
if A[M-1] >= S / (4*M):
    print("Yes")
else:
    print("No")
```

Otherwise, it can also be solved by "if there are more than or equal to M items with more than or equal to $\frac{1}{4M}$ of the total number of votes, then the answer is YES, otherwise the answer is NO." In such case, there is no need of sorting.

```
N, M = map(int,input().split())
A = list(map(int,input().split()))
S = sum(A)
cnt = 0
for a in A:
    if a >= S / (4*M):
        cnt += 1
if cnt >= M:
    print("Yes")
else:
    print("No")
```

Be careful of the behavior of division if you are using languages like C++ or Java (quotient between the integers are rounded down).

C: Replacing Integer

If x is more than or equal to K, then by performing the operation it becomes $x-K$. Therefore, by applying the operation N/K times to N, the integer becomes the remainder of N divided by K.

Let t be the remainder of t. When the operation is performed to t, it becomes $K-t$. When the operation is performed to $K-t$, it only goes back to t. Therefore, the possible value less than or equal to K are only either t or $K-t$.

Therefore, the answer is the smaller of t and $K-t$, that is, the smaller of the remainder of N divided by K and $K-(thedivisorof$ N $dividedby$ K).

D: Lunlun Number

This problem can be solved by utilizing a data structure called Queue. First, prepare an empty queue, and Enqueue $1, 2, ..., 9$ in this order. Then perform the following operations K times.

- Perform Dequeue from the Queue. Let x be the dequeued element.
- If $x \, mod \, 10 \neq 0$, then Enqueue $10x + (x \, mod \, 10) - 1$.
- Enqueue $10x + (x \, mod \, 10)$.
- If $x \, mod \, 10 \neq 9$, then Enqueue $10x + (x \, mod \, 10) + 1$.

The dequeued number in the K-th operation is the K-th Lunlun Number.

E:Yutori

To maximize the number of workdays in a certain period, it is optimal to greedily determining the workdays. Therefore, by considering determining the workdays greedily from the beginning to the end, one can obtain an array K such that "the x-th workday is no earlier than Day $L[x]$." Similarly, by considering determining the workdays greedily from the end to the beginning, one can obtain an array R such that "the x-th workday is no later than Day $R[x]$." He is bound to work on i-th day if and only if there exists a x such that $L[x] = R[x] = i$, so the problem can be solved in a total of $O(N)$ time.

F:Division or Substraction

If the operation of division is never performed, then $N \bmod K$ stays constant by the operation. So, it will become 1 in the end if and only if $N \bmod K = 1$. Such K are all the divisors of $N - 1$ except for 1, and the number of them can be counted in a total of $O(\sqrt{N})$ time.

If the operation of division is performed more than once, then such K is a divisor of N. Perform the operation until it is indivisible by K and check whether $N \mod K = 1$, and it can be check in an $O(\log N)$ time for each divisor.

Therefore, here the problem has been solved. The bottleneck part is finding the divisors of N and $N - 1$, and the total time complexity is $O(\sqrt{N})$.