ABC 169 解説

kyopro_friends, latte0119, tempura0224, ynymxiaolongbao, yuma000

2020 年 5 月 31 日

For International Readers: English editorial starts on page 8.

A: Multiplication 1

入力を読み込み、計算結果を出力します。

C 言語での実装例

```c
#include<stdio.h>
int main(){
    int a,b;
    scanf("%d%d",&a,&b);
    printf("%d\n",a*b);
}
```

python の実装例

```python
a,b = map(int,input().split())
print(a*b)
```

B: Multiplication 2

A_i に 0 が含まれる場合、答えは 0 です。そうでない場合を考えます。

64bit 整数でそのまま掛け算を行うとオーバーフローします。倍精度浮動小数点数で計算を行うと精度が足りません。多倍長整数で最後まで計算すると TLE します。したがって何らかの工夫が必要です。

Python などの多倍長整数が使える言語や、128bit 整数が使える環境であれば「積が 10^{18} が超えた時点で -1 を出力する」とすることで解くことができます。

Python での実装例

```python
def main():
    N = int(input())
    A = list(map(int,input().split()))

    if 0 in A:
        print(0)
        return

    prod = 1
    for a in A:
        prod *= a
        if prod > 1000000000000000000:
            print(-1)
            return

    print(prod)

main()
```

そうでない場合も、10^{18} を超えるかどうかの判定を乗算ではなく除算で行う必要がありますが、同様の方針で解くことができます。

C 言語での実装例

```c
#include<stdio.h>
long long a[100010];
int main(){
    int n;
    scanf("%d",&n);
    for(int i=0;i<n;i++)scanf("%lld",&a[i]);

    int zero=0;
```

```
    for(int i=0;i<n;i++)if(a[i]==0)zero++;
    if(zero>0){
        // a[i] contains 0
        printf("0\n");
        return 0;
    }

    long long prod=1;
    for(int i=0;i<n;i++){
        if(a[i]<=1000000000000000000/prod){
            // This condition is equivalent to prod*a[i]<=10^18
            prod*=a[i];
        }else{
            printf("-1\n");
            return 0;
        }
    }
    printf("%lld\n",prod);
}
```

C: Multiplication 3

浮動小数点数として掛け算を行うと精度が足りません。B を 100 倍して、 $(A \times (B \times 100))/100$ とすることで、整数の範囲で誤差なく計算することができます。B を 100 倍して整数に変換する際にも誤差に注意してください。

D: Div Game

$N = p_1^{e_1} \times ... \times p_k^{e_k}$ と素因数分解されるとします。このとき、各 p_i に対して、 $z = p_i^1, p_i^2, ...$ と順番に選んでいくのが最適です。したがって、 N を素因数分解することができれば十分です。これは $O(\sqrt{N})$ で実装することができ、十分高速です。

E. Count Median

N が奇数のとき A の中央値以上 B の中央値以下のすべての整数、N が偶数のとき A の中央値以上 B の中央値以下のすべての $\frac{1}{2}$ の倍数が X の中央値になり得ます。X の値のうちひとつを 1 増やしたときの中央値の変化は、N が奇数なら 0 か 1 、偶数なら 0 か $\frac{1}{2}$ であり、X の値のうちひとつを 1 増やす操作の繰り返しで X の中央値を A から B へ変化させることができることを踏まえると、その経路上に先に述べた実数のすべてが中央値として現れることがわかります。それ以外の実数は中央値として現れないため、これで十分です。

F: Knapsack for All Subsets

問題は以下のように言い換えることができます。

> $\{1, 2, \ldots, N\}$ の空でない部分集合 (T, U) の組であって以下の条件を満たすものは何通りありますか？
>
> 1. U は T の部分集合である
> 2. $U = \{x_1, x_2, \ldots, x_k\}$ とすると $a_{x_1} + a_{x_2} + \cdots + a_{x_k} = S$

このとき、各 i $(1 \leq i \leq N)$ について、

1. U にも T にも入れる
2. T には入れるが U には入れない
3. U にも T にも入れない

の 3 つの選択肢があり、1 つめの選択肢を選んだときのみ和が a_i だけ増えます。
以上の考察により、

$dp[i][j] =$ i 番目までの選択をして、1 つめの選択肢を選んだ k に対する a_k の和が j であるような場合の数

と定め、動的計画法を用いるとよいということがわかります。
求めたい答えは $dp[N][S]$ であり、計算量は $O(NS)$ です。
実装例: https://atcoder.jp/contests/abc169/submissions/13772481

A: Multiplication 1

Read the input and output the calculation result.

Sample code in C

```c
#include<stdio.h>
int main(){
    int a,b;
    scanf("%d%d",&a,&b);
    printf("%d\n",a*b);
}
```

Sample code in python

```python
a,b = map(int,input().split())
print(a*b)
```

B: Multiplication 2

If A_i contains 0, then the answer is 0. We will consider otherwise.

If you directly perform multiplication on 64-bit integers, it causes overflow. Double-sized floating point numbers do not have enough precisions. Multiplying entirely using multiple-precision integers leads to TLE. Therefore, we need to devise some good way.

In the programming language where multiple-precision integers like Python, or in the environment where 128-bit integers are available, it can be solved by "outputting -1 immediately after the product exceeded 10^{18}."

Sample code in Python

```python
def main():
    N = int(input())
    A = list(map(int,input().split()))

    if 0 in A:
        print(0)
        return

    prod = 1
    for a in A:
        prod *= a
        if prod > 1000000000000000000:
            print(-1)
            return

    print(prod)

main()
```

Otherwise, you can still solve it in a similar way, although you have to check if it exceed 10^{18} by performing division instead of multiplication.

Sample code in C

```c
#include<stdio.h>
long long a[100010];
int main(){
    int n;
    scanf("%d",&n);
    for(int i=0;i<n;i++)scanf("%lld",&a[i]);

    int zero=0;
```

```
    for(int i=0;i<n;i++)if(a[i]==0)zero++;
    if(zero>0){
        // a[i] contains 0
        printf("0\n");
        return 0;
    }

    long long prod=1;
    for(int i=0;i<n;i++){
        if(a[i]<=1000000000000000000/prod){
            // This condition is equivalent to prod*a[i]<=10^18
            prod*=a[i];
        }else{
            printf("-1\n");
            return 0;
        }
    }
    printf("%lld\n",prod);
}
```

C: Multiplication 3

Multiplying as floating point numbers does not provide sufficient precision. By multiplying B by 100 and calculating $(A \times (B \times 100))/100$, it can be calculated within the range of integer without errors. Note that you have to be careful also when converting B to an integer by multiplying by 100.

D: Div Game

Assume that it can be factorized as $N = p_1^{e_1} \times \ldots \times p_k^{e_k}$. Then, for each p_i, it is optimal to choose in the order of $p_i^1, p_i^2, \ldots$. Therefore, it is sufficient to factorize N. This can be implemented in a total of $O(\sqrt{N})$ time, which is fast enough.

E. Count Median

If N is odd, then the median of X can be every integer between the median of A and the median of B, and if N is even, then the median of X can be every multiple of $\frac{1}{2}$ between the median of A and the median of B, inclusive. The variation of the median of X when one of its elements is increased by one is either 0 or 1 when N is odd, and either 0 or $\frac{1}{2}$ when N is even. Considered that the median of X can be increased from A to B by repeating the operation of increasing one of the elements of X by 1, it can be seen that all of the real numbers mentioned above appears on that path as the median. The other real number cannot appear as the median, so it is sufficient.

F: Knapsack for All Subsets

The problem can be paraphrased as follows:

How many pairs of sets (T, U) such that both are non-empty subsets of $\{1, 2, \ldots, N\}$ and also satisfies the conditions below?

1. U is a subset of T
2. Let $U = \{x_1, x_2, \ldots, x_k\}$, then $a_{x_1} + a_{x_2} + \cdots + a_{x_k} = S$

Then, for each $i\,(1 \leq i \leq n)$, there are three options:

1. put to both U and T
2. put to T, but do not put to U
3. do not put to both U and T

and the sum increases by a_i only if the first option was chosen.

By the observation above, it can be seen that performing Dynamic Programming with the following recurrence relations will do:

$dp[i][j]$ = When the choices for the first i options is already determined, the number of combination such that the sum of a_k for each k that the first option was chosen is equal to j

The desired answer is $dp[N][S]$, and the total time complexity is $O(NS)$.

Sample Code: https://atcoder.jp/contests/abc169/submissions/13772481