ABC 154 解説

writer: DEGwer, kyopro_friends, latte0119, sheyasutaka, tozangezan

2020 年 2 月 9 日

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

A: Remaining Balls

$S = U, T = U$ のどちらであるか判定し、場合分けすればよいです。以下は C++ における実装例です。

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

int main(){
        string S,T,U;
        int A,B;
        cin>>S>>T>>A>>B>>U;
        if(S==U)A--;
        else B--;
        cout<<A<<"␣"<<B<<endl;
}
```

B: I miss you...

問題文の通りに実装してもよいですし，S の長さと同じ数だけ x を出力すると捉えて簡潔に書くこともできます.

実装例は次のとおりです.

Listing 1　C での実装例

```c
#include <stdio.h>

char s[101];

int main(void){
        scanf("%s", s);

        for (int i = 0; s[i]; i++) {
                s[i] = 'x';
        }

        puts(s);

        return 0;
}
```

Listing 2　Python3 での実装例

```python
s = input().rstrip()
print('x' * len(s))
```

C: Distinct

単純に全ての 2 つの整数の組に関してループで回し等しいか判定する場合、計算量が $O(N^2)$ となり間に合いません。

ところが、与えられた整数列に等しい値がある場合、この整数列をソートすると隣接する場所にこれらの値が集まることになります。

そのため、一度与えられた数列をソートすると、全ての隣接する 2 項を比較し、等しい値がなければ YES で、等しい値があれば NO です。

C++ の std::set のようなデータ構造を用いて異なる要素数を取得しても大丈夫です。

Listing 3　C++ での実装例

```
#include <stdio.h>
#include <algorithm>
using namespace std;
int A[210000];
int main(){
        int N;scanf("%d",&N);
        for(int i=0;i<N;i++)scanf("%d",A+i);
        std::sort(A,A+N);
        for(int i=0;i<N-1;i++){
                if(A[i]==A[i+1]){
                        printf("NO\n");return 0;
                }
        }
        printf("YES\n");
}
```

D: Dice in Line

それぞれのサイコロの出目は独立に決まります。そのため、いくつかのサイコロの出目の合計の期待値は、それぞれのサイコロの出目の期待値の合計に等しいです。

そのため、あらかじめそれぞれのサイコロの出目の期待値を求めておけば、それらを合計することでサイコロの出目の合計の期待値を求めることができます。

1 から P までの P 種類の目が等確率で出るサイコロの出目の期待値は $(1+P)/2$ です。

全ての隣接する K 個のサイコロについて、この値の合計を高速に求める必要があります。これは、累積和を使って求めることができます。具体的には、E_i を i 番目のサイコロの出目の期待値としたとき、$S_i = E_1 + ... + E_i$ となる S_i $(0 \quad i \quad N)$ を計算しておけば、隣接する K 個の E_i の和は $S_{j+K} - S_j$ のように求められます。

計算量は期待値の前計算、累積和の計算、隣接する K 項の和の計算、どれも $O(N)$ となります。

E: Almost everywhere zero

必要なら上の桁を 0 で埋めることで、N と同じ桁数の数だけを考えれば良いです。
桁 DP と呼ばれる考え方を用いて解くことができます。

$$dp0[i][j] = \text{上から } i \text{ 桁目まで決めて、0 でない桁が } j \text{ 個あり、N より小さいことが確定している}$$
$$dp1[i][j] = \text{上から } i \text{ 桁目まで決めて、0 でない桁が } j \text{ 個あり、N より小さいことが確定していない}$$

として、i が小さい方から順に計算していきます。求める答えは N の桁数を L として、$dp0[L][K] + dp1[L][K]$ です。計算量は $O(LK)$ です (実装によっては基数 10 が定数倍としてつきます)。

F: Many Many Paths

$g(r,c)$ を $0 \quad i \quad r$ かつ $0 \quad j \quad c$ を満たす全ての整数の組 (i,j) に対する $f(i,j)$ の総和とします。すると、求める値は $g(r2,c2)-g(r2,c1-1)-g(r1-1,c2)+g(r1-1,c1-1)$ となります。よって、 $g(r,c)$ が高速に計算できればこの問題が解けます。

唐突ですが、$f(r+1,c)=f(r,0)+f(r,1)+...+f(r,c)$ が成り立ちます。これは、点 $(0,0)$ から点 $(r+1,c)$ への経路について考えてみると説明できます。点 $(0,0)$ から点 $(r+1,c)$ へのどの経路も、「点 $(0,0)$ から点 $(r,0)$ に至り、そのまま $(r+1,0)$ に移動し、点 $(r+1,c)$ へ移動」「点 $(0,0)$ から点 $(r,1)$ に至り、そのまま $(r+1,1)$ に移動し、点 $(r+1,c)$ へ移動」「点 $(0,0)$ から点 $(r,2)$ に至り、そのまま $(r+1,2)$ に移動し、点 $(r+1,c)$ へ移動」...「点 $(0,0)$ から点 (r,c) に至り、そのまま $(r+1,c)$ に移動し、点 $(r+1,c)$ へ移動」のうちいずれか 1 つに必ず該当し、2 つ以上に該当することはありません。

ところで点 $(0,0)$ から点 $(r+1,c)$ への経路数が $f(r+1,c)$ で、点 $(0,0)$ から点 (r,i) に至り、そのまま $(r+1,i)$ に移動し、点 $(r+1,c)$ へ移動する経路数は、「点 $(0,0)$ から点 (r,i) に至る」のが $f(r,i)$ 通り、そのまま 1 回下に移動、残りはひたすら右に移動を繰り返すことが強制されているので、点 (r,i) から先は 1 通り、すなわち全部で $f(r,i)$ 通りです。以上より式 $f(r+1,c)=f(r,0)+f(r,1)+...+f(r,c)$ が成り立ちます。

また $f(r,c)=_{r+c}C_c=(r+c)!/(r!c!)$ です。よって、求めたい値は $O(N)$ 個の $f(x,y)$ の和で書き表わせるので、$f(x,y)$ が $O(1)$ で求められれば良いです。これは階乗と階乗の逆元を前計算しておけば良いです。

また、$f(r,c+1)=f(0,c)+f(1,c)+...+f(r,c)$ をもう一度適用すると、求めたい値は定数個の $f(x,y)$ の和でも書き表すことができます。