ABC 149 解説

writers: beet, DEGwer, IH19980412, kyopro_friends, tozangezan, yuma000

2019 年 12 月 29 日

For International Readers: English editorial starts on page 7.

A: Strings

2 つの文字列 S, T を順に読み込み、T, S の順にスペースを空けずに出力することで目的の文字列を出力したことになります。(T,S の順に文字列を連結することで新しい文字列を作っても良いです。)

Listing 1　C++ での実装例

```
#include <stdio.h>

char S[110];
char T[110];

int main(){
        scanf("%s%s",S,T);
        printf("%s%s\n",T,S);
}
```

B: Greedy Takahashi

K の値によって次のように 3 通りに場合分けすることで解けます。

- $K \leq A$ のとき、高橋君は自分の持っているクッキーを K 枚食べます。
- $A < K \leq A + B$ のとき、高橋君は最初の A 回の行動で自分の持っているクッキーを全て食べ、残りの $K - A$ 回の行動で青木君の持っているクッキーを $K - A$ 枚食べます。
- $A + B < K$ のとき、高橋君は最初の A 回の行動で自分の持っているクッキーを全て食べ、次の B 回の行動で青木君の持っているクッキーを全て食べます。

C++ による実装例は以下の通りです。

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

int main(){
        long long a, b, k;
        cin >> a >> b >> k;

        if(k <= a){
                cout << a - k << '␣' << b <<endl;
        }else if(k <= a+b){
                cout << 0 << '␣' << b - (k-a) <<endl;
        }else{
                cout << 0 << '␣' << 0 <<endl;
        }
}
```

C: Next Prime

X 以上の素数のうち、最小のものを求める問題です。

X 以上の数を小さい順に素数かどうか判定していけばよいです。ある数 A が素数であるかどうかは $O(\sqrt{A})$ で判定できます。100003 は素数なので、答えは 100003 以下になり、十分高速です。

以下は C++ における実装例です。

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

bool is_prime(int x){
  if(x<=1) return false;
  for(int i=2;i*i<=x;i++)
    if(x%i==0) return false;
  return true;
}

signed main(){
  int x;
  cin>>x;

  int p=x;
  while(!is_prime(p)) p++;

  cout<<p<<endl;
  return 0;
}
```

D: Prediction and Restriction(writer : yuma000)

まず前提として、 a 回目にこちらが出す手と b 回目にこちらが出す手は、a mod K != b mod K ならば、独立に選べます (どちらかの選択がもう片方の選択に制限を与えることはない)。

N 回のじゃんけんを、mod K によって K 個のグループに分けることを考えます。この時、各グループに対して、以下の sub 問題を解くことができれば、この問題を解くことができます。

- sub 問題 このグループに属するじゃんけんで、最大何点とれるか？ただし、連続で同じ手を出すことはできない。

以下が、sub 問題への解法です。

解法 1(DP)

前から順に、前回にどんな手を出したかを保持しておきながら DP をします。各 sub 問題に対しての時間計算量は O(N/K) なので、トータルでの時間計算量は O(N) です。

解法 2(貪欲法)

貪欲法でも解けます (こっちの方が実装は楽です)。具体的には前から順に、

- 勝てるならば勝てる手を出す。
- (前回自分が出した手の制限のせいで) 勝てない場合は、次の手の邪魔にならない手を出す。

以上のようにすることで最適な解が求まります。

E: Handshake(writer : yuma000)

もちろん、もっとも上がる幸福度が高い握手から順番に M 回することで、最終的な幸福度を最大にできます。が、愚直に実装すると間に合いません。以下でこれを効率的に求める方法を解説します。

解法 1(二分探索を使った解法)

まず、増える幸福度が、ある定数 X 以上になる握手の方法が何通りあるか、を求めることを考えます。左手で握手する人を先に決めたとき、右手で握手できる人が何人いるかは累積和を用いることで O(1) で求まるのでトータルで O(N) で計算できます。

後は、X を二分探索することで、M 通り以上の握手ができる最小の X が求まるので、もう一度累積和を用いて最終的な幸福度が計算できます。時間計算量は O(NlogN) です。

解法 2(高速フーリエ変換)(おまけ)

$A_i \leq 100,000$ という条件から、高速フーリエ変換 (以下 FFT) でも解けます。FFT の詳細については、以下のページをご覧ください。

`https://atcoder.jp/contests/atc001/tasks/fft_c`

具体的には、

- i 番目の要素にパワー i の人の人数が格納されている配列を二つ用意する。
- それらの配列を畳み込む。(FFT をすることで、時間計算量 $O(max(A_i)log(max(A_i))$ でできます。)

という風にすることで、増える幸福度ごとの通り数が分かります。後はそれを増える幸福度が高い順に加えていくことで、この問題を解くことができます。

F: Number of Surrounded Nodes

解法 1: 辺に注目する方法

求めるものは「S の頂点の個数 − 黒く塗られた頂点の個数」と等しいです。期待値の線形性から、それぞれの項の期待値が求まればよいです。黒く塗られた頂点の個数の期待値は明らかに $N/2$ です。S の頂点の個数の期待値を考えます。

木の頂点の個数は (空グラフの場合を除き) 辺の個数 $+1$ なので、S の頂点の個数のかわりに、S の辺の個数を考えます。期待値の線形性から、各辺が S に含まれる確率を計算し、その合計を求めればよいです。辺 e が S に含まれるための必要十分条件は、T から e を取り除いてできる 2 つの部分木の両方に黒く塗られた頂点が存在することです。2 つの部分木の頂点数を x_e, $N - x_e$ とすると、そのような確率は $\left(1 - \left(\frac{1}{2}\right)^{x_e}\right)\left(1 - \left(\frac{1}{2}\right)^{N-x_e}\right)$ となるので、x_e がわかれば計算することができます。

以上より、各辺 e について x_e が求まればよく、これは DFS などにより $O(N)$ で求めることができます。空グラフの扱いに注意してください。

解法 2: 頂点に注目する方法

期待値の線形性から、各頂点 v について、v が S に属しかつ白く塗られているような確率を計算し、その合計を求めればよいです。頂点 v について、T から v を取り除いてできる部分木たちを $T_{v1}, T_{v2}, ...$ とします。このとき、v が S に含まれるための必要十分条件は、T_{vi} たちのうち少なくとも 2 つが黒く塗られた頂点を含むことです。これは、T_{vi} に属する頂点の個数 x_{vi} がわかれば、余事象を考えることで求めることができます。

以上より、各頂点 v について x_{vi} たちが求まればよく、これは全方位木 DP (ReRooting) などにより $O(N)$ で求めることができます。

また、式を整理することで解法 1 と同じ式を得ることもできます。

A: Strings

Receive two strings S, T in this order, and output them in the order of T, S without printing whitespaces, then you will have outputted the desired string. (You can also make a new string by concatenating the strings in the order of T, S.)

Listing 2 Sample Code in C++

```
#include <stdio.h>

char S[110];
char T[110];

int main(){
        scanf("%s%s",S,T);
        printf("%s%s\n",T,S);
}
```

B: Greedy Takahashi

You can divide into the following three cases depending on the value of K.

- If $K \leq A$, then Takahashi will eat K cookies he owns.
- If $A < K \leq A + B$, then Takahashi will eat all the cookies he own in the first A actions, and then $K - A$ cookies of Aoki's in the $K - A$ actions left.
- If $A + B < K$, then Takahashi will eat all the cookies he owns in the first A actions, and then eat all of Aoki's cookies in the next B actions.

The following is an implementation example in C++.

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

int main(){
        long long a, b, k;
        cin >> a >> b >> k;

        if(k <= a){
                cout << a - k << '␣' << b <<endl;
        }else if(k <= a+b){
                cout << 0 << '␣' << b - (k-a) <<endl;
        }else{
                cout << 0 << '␣' << 0 <<endl;
        }
}
```

C: Next Prime

You are asked to find the minimum prime greater than or equal to X.

It is enough to iterate through the numbers greater than or equal to X in order and check if it is a prime number. You can check if a number A is a prime number or not in a $O(\sqrt{A})$ time. Since 100003 is a prime number, the answer is less than or equal to 100003, so it is fast enough.

The following is an implementation example in C++.

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

bool is_prime(int x){
  if(x<=1) return false;
  for(int i=2;i*i<=x;i++)
    if(x%i==0) return false;
  return true;
}

signed main(){
  int x;
  cin>>x;

  int p=x;
  while(!is_prime(p)) p++;

  cout<<p<<endl;
  return 0;
}
```

D: Prediction and Restriction(writer : yuma000)

First, as a premise, you can choose the a-th hand and the b-th hand independently if a mod K != b mod K (one choice does not restrict another choice).

Consider distributing N rounds of Rock Papers Scissors into K groups depending on mod K. Then, if the following sub problem can be solved for each group, this problem can also be solved.

- sub problem: what is the maximum score earned in the rounds belonging to this group? You cannot use the same hand contiguously.

The following is the solution for the sub problem.

Solution 1(DP)

From the begin to the end, perform DP while retaining the last hand you used. The time complexity for each sub problem is O(N/K), so the total time complexity is O(N).

Solution 2(Greedy Algorithm)

You can also solve with greedy algorithm (the implementation is simpler). Specifically, from the beginning to the end,

- If you can win, use the hand that you can win.
- If you cannot win (because of the hand last you used), use that hand that does not obstruct the next hand.

By doing so, you can find the optimal solution.

E: Handshake(writer : yuma000)

Of course, you can maximize the ultimate happiness M times from the handshake that increases happiness the most to the least. But, if you implement naively it won't finish in time. We will explain the way of finding it efficiently.

Solution 1(Solution using binary search)

First, consider how many ways of handshakes are there by which the happiness increases by more than or equal to a constant X. When a person to shake hand for his left hand is fixed, you can find how many people are there whom he can shake hand with for his right hand in O(1) time using cumulative sums, so it can be calculated in a total of O(N) time.

Then, you can perform binary search for X so as to find the minimum X such that he can perform M ways of handshakes, so you can find the ultimate happiness by using cumulative sums again. The time complexity is O(NlogN).

Solution 2(Fast Fourier Transform)(Bonus)

Since $A_i \leq 100,000$, you can solve this problem by Fast Fourier Transform (hereinafter FFT). For more details about FFT, please look for many resources online. (There are an educational problem on AtCoder, but the problem statement is provided only in Japanese. For your information, the URL is as follows:)

`https://atcoder.jp/contests/atc001/tasks/fft_c`

Specifically,

- Prepare two arrays, each of whose i-th element contains the number of people with power i.
- Find the convolution of those array. (By performing FFT, it can be done in a total of $O(max(A_i)log(max(A_i))$ time.)

so that you can find the number of ways for each increasing happinesses. Lastly count them in the decreasing order of increasing happinesses, and you can solve the problem.

F: Number of Surrounded Nodes

Solution 1: Focusing on edges

What you want to calculate is equal to "the number of vertices of S − the number of vertices painted black." By the linearity of expected values, it is enough to find the expected value of each term. The number of vertices painted black is obviously N_2. Now let's consider the expected number of vertices of S.

The number of vertices of a tree is the number of edges +1 (except for an empty graph), so let's consider the number of edges of S instead of the number of vertices of S. An edge e is contained in S if and only if, when e was removed from T, the resulting two parts both contain vertices painted black. Let x_e, $N - x_e$ be the number of vertices of two subtrees, then such probability is $\left(1 - \left(\frac{1}{2}\right)^{x_e}\right)\left(1 - \left(\frac{1}{2}\right)^{N-x_e}\right)$, so if you found x_e you can calculate them.

Therefore, it is enough to find x_e for each edge e, and this can be found in a $O(N)$ time by DFS etc. Please be careful how to treat empty graphs.

Solution 2: Focusing on vertices

By the linearity of expected values, it is enough to find the probability that, for each vertex v, v is contained in S and also painted white, and to sum them up. For a vertex v, let $T_{v1}, T_{v2}, ...$ be the subtrees when v is removed from T. Then, v is contained in S if and only if at least two subtrees out of T_{vi}'s contain vertices pained black. You can calculate this by considering complementary events by finding x_{vi}, the number of vertices of T_{vi}.

Therefore, it is enough to find x_{vi}'s for each vertex v, and this can be found in a $O(N)$ time by ReRooting etc.

Also, by transforming the equation you can obtain the same equation as Solution 1.