Code festival 予選 B 解説

maroonrk, snuke, rng_58

2017/10/08

*For International Readers: English editorial starts on page *.*

A: XXFESTIVAL

入力を読み、最初の 文字列の長さ − 8 文字を出力すれば良いです。

以下は C++ での実装例です。

```
#include <iostream>
#include <string>

using namespace std;

int main(void){
  string s;
  cin >> s;
  cout << s.substr(0, s.length() - 8) << endl;
  return 0;
}
```

B: Problem Set

全ての難易度 x について、「難易度が x の問題案の個数」が「問題セットに必要な難易度 x の問題案の個数」以上であれば問題セットを完成させることができます。

まず、各値がそれぞれ何回現れるかを D と T についてそれぞれ求めておきます。これは例えば C++ では map などを使うことによって計算できます。そして、T に現れる値それぞれについて上記のような比較を行えば良いです。

以下は C++ での実装例です。

```
#include <iostream>
#include <map>

using namespace std;

int A;
int a[100010];
int B;
int b[100010];
map <int , int> mpa,mpb;

int main(void){
  int i;

  cin >> A;
  for(i=0;i<A;i++) scanf("%d", &a[i]);
  cin >> B;
  for(i=0;i<B;i++) scanf("%d", &b[i]);

  for(i=0;i<A;i++) mpa[a[i]]++;
  for(i=0;i<B;i++) mpb[b[i]]++;

  for(i=0;i<B;i++){
    int x = b[i];
    if(mpb[x] > mpa[x]){
      cout << "NO" << endl;
      return 0;
    }
  }
```

```
    cout << "YES" << endl;

    return 0;
}
```

C: 3 Steps

まず、頂点 s と t の間に奇数長のパス（単純パスでなくても良い）が存在したとすると必ず s と t の間に辺が張られることを示します。このパスの長さを k、i 番目に通る頂点 v_i とします。k は奇数であること、$v_0 = s$ かつ $v_k = t$ であることに注意してください。

k が 1 の場合、すでに s と t の間には辺があります。$k \geq 3$ の場合、v_{k-3} と v_k の間に辺を張ることができ、長さが $k-2$ 奇数長の $s-t$ パスを得ることができます。これを繰り返すことによってパスの長さは 2 ずつ短くなり、長さがちょうど 3 になったとき s と t の間に辺が張られることになります。つまり、ある頂点間に奇数長のパスがあるかどうかということが重要となります。ある頂点間に奇数長のパスがあるかどうかはグラフが二部グラフであるかどうかに影響されます。

ケース 1. 二部グラフでない場合

二部グラフでないグラフには奇サイクルが存在します。v を奇サイクル上のある頂点とします。グラフは連結であるため、任意の頂点の組 (s,t) に対して、$s \to v \to t$ という順に頂点を訪れるパスが存在します。もしそのパスが偶数であった場合、頂点 v に訪れた時に余計に奇サイクルを辿ることによってパスの長さを奇数にすることができます。

つまり、任意の 2 頂点間に奇数長のパスが存在することになり、グラフが完全グラフになるまで辺を追加することができます。したがって、このケースでは単に $N(N-1)/2 - M$ を出力すれば良いです。

ケース 2. 二部グラフである場合

このケースでは、頂点を黒と白に塗り分け、全ての辺が異なる色の頂点どうしを結んでいるようにすることができます。任意の黒の頂点 b と白の頂点 w をとります。グラフは連結であるため、b と w の間にパスが存在します。このパスは明らかに奇数となります。つまり、b と w の間にはいつか辺が張られることになります。

一方、同じ色の頂点の間には絶対に辺が張られることはありません。辺を張る操作で選ぶ 2 頂点は必ず異なる色になるためです。

このケースでは、B を黒の頂点の個数、W を白の頂点の個数としたとき、$BW - M$ を出力すれば良いです。

D: 101 to 010

X, Y, Z　に操作をするとき、X と Z のトークンを Y に動かすものとします。初期状態の各トークンは、最終状態のどれかのトークンに対応します。(別のトークンが同じトークンに対応するかもしれません。)

操作を逆から見ると。最終状態の '1'　は、 もともと　"1"　であったか、"101"　に操作をしたものです。さらに "101" の中の 1 も操作した結果のものである可能性があるので、"1011" や　"1101" であった可能性もあります。また、例えば "1101" の真ん中の 1 は 0 に囲まれていないので元からあるものだと分かります。

これを繰り返すと、

- 　('1').
- "111 $\cdots$ 11101" (連続する 1 が k 個のとき $k-1$ 回の操作)
- "10111 $\cdots$ 111" (連続する 1 が k 個のとき $k-1$ 回の操作)

が最終状態での一個のトークンに対応する可能性があります。

つまり、 s から重ならないよい部分文字列を選んでそのコストの和を最大化する問題になります。

ただし良い文字列とは

- "111 $\cdots$ 11101" (コスト $k-1$)
- "10111 $\cdots$ 111" (コスト $k-1$)

です。

これで簡単に $O(N^2)$ の　DP　がかけます。

また、

$\cdots 1110111 \cdots$ (a個)$1110111 \cdots$ (b個)$1110111 \cdots$

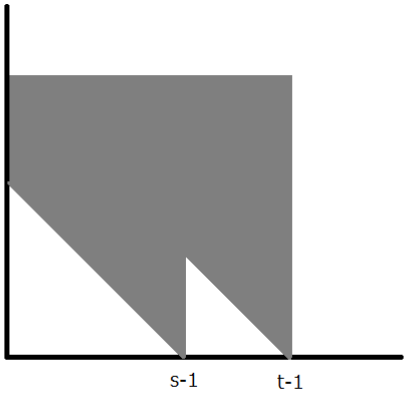
という文字列で真ん中の 0 を含む良い部分文字列が $a+b$ 個であることに注目すると、よい部分文字列は全体で $O(N)$ 個であることが分かるので、これを利用すると DP は $O(N)$ となります。

E: Popping Balls

ボールの列を、赤では下に進み、青では左に進むように平面状にプロットします。(A, B) から $(0, 0)$ への経路数を数える問題になります。

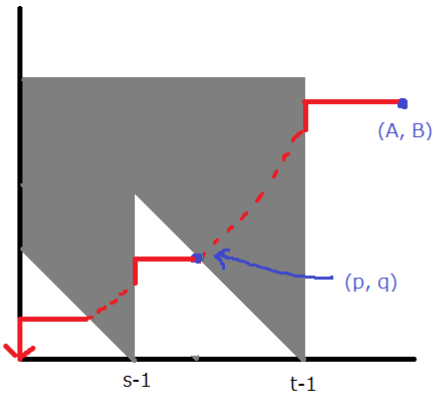
s と t が固定されているとき、

まず、最初のボールをとることによって、必ず左に進むことができます。また、$1, s, t$ 番目のどれかが青いとき下に進めるので、図で灰色の部分では下に進めます。



次に、s, t が固定されていない場合を考えます。パスが与えられたとき、最初に下に進む部分が t によってできる領域の境界と一致するように、t によってできた領域の外側で最初に下に進むときの場所が s によってできる領域の境界と一致するように、s, t を決めるのが最適と分かります。

つまり、下図のように見えるパスの個数をすべての (s, t) について数え、合計したものが答えとなります。



図の (p, q) を全探索します。(p, q)　が固定されたとき、(A, B) から (p, q) へのパスの個数は二項係数とし

て表せ、また、(p,q) から $(0,0)$ へのパスの個数は二項係数やその和を前計算することによって $O(1)$　で求めることができます。

答えは、この二つの積を全ての (p,q)　について足したものとなります。

計算量は　$O(N^2)$　です。

F: Largest Smallest Cyclic Shift

二分探索をすると、文字列 U (と整数 X, Y, Z) が与えられたとき以下を満たす S の存在を判定する問題になります。

Problem 1.

- S は X 'a', Y 'b', Z 'c' からなる
- すべての S の巡回シフトは U 以上

これは、次の問題と同じであることが分かります。

Problem 2.

- S は X 'a', Y 'b', Z 'c' からなる
- $U \leq S$
- S は S の巡回シフトのうち最小

これは、 $S = S_1$ が最初の条件をみたすとき S_1 の最小巡回シフトが Problem 2 の答えとなるためです。
整数 i に対し最初の i 文字とそれ以外を分けて $S = p_i q_i$ とします。
S が Problem 2 の条件を満たすとき、 $q_i p_i$ は S 以上なので、それを無限回繰り返して
$q_i p_i q_i p_i q_i p_i \cdots = q_i SSS \cdots \geq SSS \cdots$
$S \geq U$ なので、 S を U にしても S の影響が弱まるだけなので成り立ちます:
$q_i UUU \cdots \geq UUU \cdots$
また、このとき、 $S \geq U$ なので
$q_i p_i q_i p_i q_i p_i \cdots \geq q_i SSS \cdots \geq q_i UUU \geq UUU \cdots$
となり $q_i u_i \geq U$ となるので Problem 1 の条件が成り立つことが分かります。
つまり、 Problem 1, Problem 2 は以下と同値です:

Problem 3.

文字列 U (と整数 X, Y, Z) が与えられたとき以下を満たす S の存在を判定せよ。

- S は X 'a', Y 'b', Z 'c' からなる
- 任意の i に対し $q_i UUU \cdots \geq UUU \cdots$

これは、以下の DP で解けます。
$dp[i][j][k]$ を以下を満たす最大の文字列とします:

- 文字列は i 'a', j 'b', k 'c' からなる
- 文字列は S の suffix となることができる

DP が $O(ABC)$ 状態、遷移に $O(A+B+C)$、二分探索に $O(A+B+C)$ で計算量は $O(ABC(A+B+C)^2)$ となります。

CODE FESTIVAL 2017 Qualification Round B Editorial

maroonrk, snuke, rng_58

2017/10/08

A: XXFESTIVAL

Read the input and print its first $N - 8$ characters, where N is the length of the string.

Here is an example C/C++ implementation:

```
#include <iostream>
#include <string>

using namespace std;

int main(void){
        string s;
        cin >> s;
        cout << s.substr(0, s.length() - 8) << endl;
        return 0;
}
```

B: Problem Set

We can complete the problemset if for all x, the number of problems of difficulty x we have (i.e., the number of occurrences of x in D) is greater than or equal to the number of problems of difficulty x we need (i.e., the number of occurrences of x in T).

First, we should count the frequencies of each value in D and T, for example by using C++ map. Then, for each x in T, compare the two values mentioned above.

Here is an example C/C++ implementation:

```
#include <iostream>
#include <map>

using namespace std;

int A;
int a[100010];
int B;
int b[100010];
map <int , int> mpa,mpb;

int main(void){
        int i;

        cin >> A;
        for(i=0;i<A;i++) scanf("%d", &a[i]);
        cin >> B;
        for(i=0;i<B;i++) scanf("%d", &b[i]);

        for(i=0;i<A;i++) mpa[a[i]]++;
        for(i=0;i<B;i++) mpb[b[i]]++;

        for(i=0;i<B;i++){
                int x = b[i];
                if(mpb[x] > mpa[x]){
                        cout << "NO" << endl;
                        return 0;
                }
        }
```

```
        cout << "YES" << endl;

        return 0;
}
```

C: 3 steps

Suppose that there is a (not necessarily simple) path of odd length between two vertices s and t. That is, for some odd number k, there exists a sequence of vertices $s = v_0, v_1, \ldots, v_k = t$, such that for each i, v_i and v_{i+1} are adjacent.

If $k = 1$, it means that there is an edge between s and t. If $k \geq 3$, by performing the operation between v_{k-3} and v_k, we can add an edge between them and now we have a path of length $k - 2$ between s and t. By repeating this, we can keep decreasing the length of the path by two and eventually we can add an edge between s and t. This suggests that the existence of odd-length paths is important - and the bipartiteness is also important.

Case 1. The graph is not bipartite.

In this case, the graph contains an odd cycle. Let v be an arbitrary vertex in the cycle. Since the graph is connected, for arbitrary pair of two vertices (s, t), we can find a (not necessarily simple) path that visits $s \to v \to t$. If the length of this path is even, by attaching the odd cycle at v, we can make the length odd.

Therefore, we can find an odd path between arbitrary two vertices, and by the observation mentioned above, we can make the graph complete. The answer is simply $N(N-1)/2 - M$.

Case 2. The graph is bipartite.

In this case, we can color the vertices black and white, such that each edge connects a black vertex and a white vertex. Consider an arbitrary black vertex b and a white vertex w. Since the graph is connected, there exists a path between b and w, and obviously, the length of this path must be odd. Thus, we can eventually add an edge between b and w.

On the other hand, we can never add an edge between two vertices of the same color. This is because in each operation we have to choose two vertices of different colors.

The answer is $BW - M$, where B is the number of black vertices and W is the number of white vertices.

D: 101 to 010

Let's call the input "the initial state", and call the state after you perform all operations "the final state". When we perform an operation on three consecutive cells X, Y, Z, assume that the tokens on X and Z are "moved" to Y and grouped together. Then, each token in the initial state corresponds to a token in the final state. Note that different tokens in the initial state may correspond to the same token in the final state.

Let's see the operation in reverse order. A token ('1') in the final state may be a token in the initial state ("1"), or it may be a result of an operation ("101"). Some '1' in "101" may also be a result of another (former) operation, so it may correspond to "1011" or "1101" in the initial state. The first '1' in "1011" may also be a result of another operation, so it may correspond to "10111". However, we are sure that the second '1' in "1011" is not a result of operation: after an operation, the token must be surrounded by empty cells. And so on.

To summarize, each token in the final string corresponds to one of the following in the initial state:

- A token ('1').
- "$111 \cdots 11101$" (k ones followed by a zero followed by a one, for some positive integer k). We performed $k-1$ operations.
- "$10111 \cdots 111$" (A one followed by a zero followed by k ones, for some positive integer k). We performed $k-1$ operations.

Therefore, we need to solve the following problem: You want to choose some non-overlapping *good* substrings from s and maximize the sum of their scores. What is the maximum score?

Here, the following strings are *good* (and the scores are defined as follows):

- "$111 \cdots 11101$" (k ones followed by a zero followed by a one, for some positive integer k). The score is $k-1$.
- "$10111 \cdots 111$" (A one followed by a zero followed by k ones, for some positive integer k). The score is $k-1$.

This will lead to an $O(N^2)$ DP.

To make it $O(N)$, notice that the number of good substrings in s is only $O(N)$. This is because, in the following situation, the number of good substrings that contains the '0' in the middle is $O(a+b)$ (more precisely, $a+b-1$):

$\cdots 1110111 \cdots (\mathrm{a}ones)1110111 \cdots (\mathrm{b}ones)1110111 \cdots$

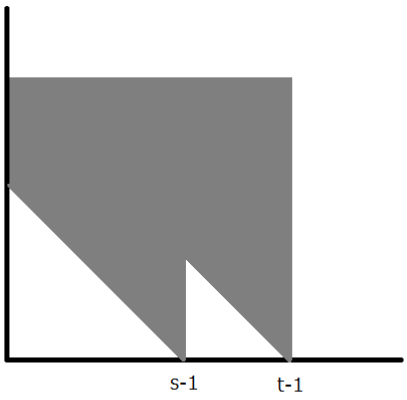
Therefore, for example, you can pre-compute the positions of all good substrings, and apply DP-transitions only for those intervals. This way the solution works in $O(N)$.

E: Popping Balls

Let's map a sequence of balls to a path in 2-dimensional grid. We start from (A, B). When you give a red ball to Snuke, go down, and when you given a blue ball to Snuke, go left. We want to count the number of different paths from (A, B) to $(0, 0)$ (that can be obtained this way).

First, assume that s and t are fixed. From (x, y), we can always go to $(x - 1, y)$, unless $x = 0$. (We can just pop the first ball.)

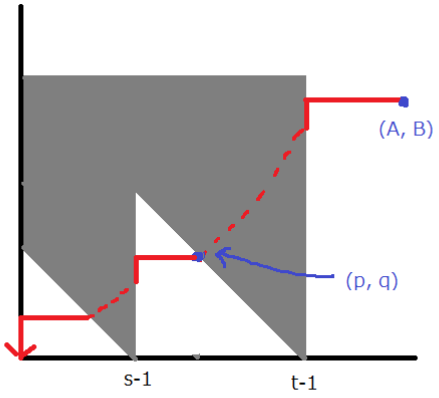
We can go to $(x, y-1)$ only when at least one of the 1-st, s-th, or the t-th ball is blue. In the following picture, we can go down when we are in the leftmost edge or $(x, y - 1)$ is in the shaded region. We can always go left. Under these conditions, we want to count the number of paths from (A, B) to $(0, 0)$.



Now, assume that we are given a path from (A, B) to $(0, 0)$, and we want to determine if this path is valid (by choosing appropriate s, t). From (A, B), the path goes left for zero or more steps, and at (A_0, B) it first goes down. Then, $t \geq A_0$ must be hold. Also, it doesn't make sense to make t too big: it simply shrinks the shaded area in the region $x \leq A_0$ (and we are not interested in the area $x > A_0$). Thus, $t = A_0$.

Similarly, we should set s when we first go down outside the shaded region covered by t.

Therefore, we want to count the number of paths that look as follows, and compute the sum of number of these paths over all pairs (s, t):



We try all possibilities of the pair (p, q) in the picture above. For a fixed (p, q), the number of paths from (A, B) to (p, q) in the form above can be represented as a binomial coefficient. Similarly, we can compute the number of paths from (p, q) to $(0, 0)$ in the form above in $O(1)$, by pre-computing binomial coefficients and various its sums.

The answer is the sum of the product of these two values over all pairs (p, q).

This solution works in $O(N^2)$.

F: Largest Smallest Cyclic Shift

We believe there are various ways to solve this problem. We first describe the intended (and proved) solution.

Let's solve the problem by binary search. For a given string U (and three integers X, Y, Z), we want to determine whether there exists a string S such that

Problem 1.

- S consists of X 'a's, Y 'b's, and Z 'c's.
- All cyclic shifts of S are greater than or equal to U.

Notice that even if we slightly strengthen the conditions as follows, the answer (yes/no) remains the same.

Problem 2.

- S consists of X 'a's, Y 'b's, and Z 'c's.
- $U \leq S$.
- S is the smallest among all its cyclic shifts.

This is because $S = S_1$ is an answer to problem 1, the smallest cyclic shift of S_1 satisfies the conditions of problem 2.

For each integer i, let $S = p_i q_i$, i.e, p_i is the prefix of S of length i, and q_i is the remaining part.

When S satisfies the conditions in Problem 2, a cyclic shift of S ($q_i p_i$) must be greater than or equal to S. By comparing their infinite repetitions, we get

$q_i p_i q_i p_i q_i p_i \cdots = q_i SSS \cdots \geq SSS \cdots$

Here, since $S \geq U$, even if we replace S by U, the inequality holds (because the "impact" by S will be weakened):

$q_i UUU \cdots \geq UUU \cdots$

On the other hand, when this inequality holds, since $S \geq U$, we get

$q_i p_i q_i p_i q_i p_i \cdots \geq q_i SSS \cdots \geq q_i UUU \geq UUU \cdots$

and we get $q_i u_i \geq U$, thus the conditions in Problem 1 will be satisfied.

Therefore, Problem 1 and Problem 2 are equivalent to the following:

Problem 3.

For a given string U (and three integers X, Y, Z), determine whether there exists a string S such that

- S consists of X 'a's, Y 'b's, and Z 'c's.
- For each i, $q_i UUU \cdots \geq UUU \cdots$ (here q_i is the suffix of S of length $|S| = i$).

Now the problem can be easily solved by a DP.

Define $dp[i][j][k]$ as the largest string such that

- The string consists of i 'a's, j 'b's, and k 'c's.

- The string can be a suffix of S that satisfies the conditions in Problem 3.

This DP has $O(ABC)$ states, each transition takes $O(A+B+C)$ time (for string comparisons), and since there are $O(A+B+C)$ steps for the binary search, this solution works in $O(ABC(A+B+C)^2)$.