AtCoder Grand Contest 005 Editorial

Kohei Morita(yosupo)

October 1st, 2016

A: STring

まず、問題分には 10^{10000} 回という非常に大きな数が書いてありますが、実際には多くとも削除は $|X|/2$ 回しか行えないため、それ以上は意味がありません。

0.1 部分点

文字列から ST を探すのは、$O(|X|)$ で可能です。文字列検索はほぼ全ての言語に標準ライブラリとして入っているはずです。また、文字列の途中から ST を削除するのも $O(|X|)$ で可能です。以上より、一回の操作は $O(|X|)$ で行うことが出来ます。

そして、操作は $|X|/2$ 回しか意味がないため、それ以上は行わなくて良いです。よって、$O(|X|^2)$ でこの問題は解くことが出来ます。

0.2 満点

$O(|X|^2)$ では間に合わないため、更に高速化する必要があります。
実はスタックを使い、以下の操作を行うと $O(|X|)$ で解くことが出来ます。

- 文字列の文字を先頭から見ていき、
- 見ている文字が S ならば、スタックに push
- 見ている文字が T かつ、スタックの先頭の文字が T or 空ならばスタックに push
- 見ている文字が T かつ、スタックの先頭の文字が S ならばスタックを pop
- を文字列の最後まで繰り返す。

この操作によりスタックに残ったものが答えの文字列です。

イメージとしては、文字列を前からスキャンし、ST を発見したら適時削除していくイメージです。

B: Minimum Sum

この問題は、${}_N\mathrm{C}_2$ 個の区間それぞれについて最小値を求め、それを足しあわせたものを求める問題です。区間は $O(N^2)$ 個ありますが、$O(N^2)$ ではTLE するので、区間それぞれについて最小値を求めていては間に合いません。よって、視点を変える必要があります。

ここで、”最小値が a_i となる区間はいくつあるか？”をすべての i について求めることを考えます。

- a_i より左側で、a_i より小さいもののうち最も a_i に近いものを a_l
- a_i より右側で、a_i より小さいもののうち最も a_i に近いものを a_r

とすると、最小値が a_i となる区間は、左側が $l+1, l+2, ..., i$ のうちどれかで、右側が $i, i+1, i+2, ..., r-1$ となる区間です。よって、$(i-l+1)\times(r-i+1)$ 個の区間で最小値が a_i となります。これを足し合わせれば答えが得られるので、すべての i についてこの l, r を求めれば良いことがわかります。

$a_i = N, N-1, N-2, ...$ の順で l, r を求めていくことを考えます。すると、l, r は今まで見た要素のなかで、i 以下最大のものと i 以上最小のものであることがわかります。これは、平衡二分木 (例えば C++ならば std::set) に今まで見た要素の位置を入れていけば求めることが出来ます。

以上より、この問題は $O(NlogN)$ で解くことが出来ました。

C: Tree Restoring

はじめに、頂点 u, v について、$\mathrm{dist}(u, v)$ を 2 点間の距離とします。

仮に条件を満たす木が存在した場合、その木の直径 (木から 2 頂点 u, v を選んだ時の $\mathrm{dist}(u, v)$ の最大値) はいくつになるでしょうか。当然 $\max(a_1, a_2, ..., a_N)$ となります。$\mathrm{dist}(a, b)$ が木の直径となる時、パス $a - b$ 上の頂点について考えます。実は、$\mathrm{dist}(a, b)$ が木の直径であるとき、任意の p について

$$\max(\mathrm{dist}(p, 1), \mathrm{dist}(p, 2), ..., \mathrm{dist}(p, N)) = \max(\mathrm{dist}(p, a), \mathrm{dist}(p, b))$$

という性質があります。

この性質を使うと、直径を K とすると、パス $a - b$ 上の頂点の、他の頂点との距離の最大値は

- K が偶数の時、$K, K-1, K-2, ..., K/2+1, K/2, K/2+1, ..., K-1, K$
- K が奇数の時、$K, K-1, K-2, ..., (K+1)/2+1, (K+1)/2, (K+1)/2, (K+1)/2+1, ..., K-1, K$

となることがわかります。

よってまずはこのパスを構築します。もし構築できないならば Impossible です。そして、このパスのどこに頂点をつけても最大距離は偶数ならば $K/2+1$ 以上、奇数ならば $(K+1)/2+1$ 以上になります。なのでこれ未満が残っていたら、結果は Impossible です。これ以上のものしか無いとします。するとそれらを、それより最大距離が 1 小さいパス上の頂点につけます。すると条件を満たす木が構築できます。よって Possible です。

D: ～K Perm Counting

はじめに、この問題は ”どれかの i について $|a_i - i| = K$ を満たすもの” を数えても良いので、こちらを数えます。更に条件を、$a_i - i = K$ と $a_i + i = K$ という $2N$ 個の条件に分解しておきます。すると数えるべきは、この $2N$ 個の条件のうちどれか 1 つは満たす順列の個数です。

実は、長さ N の順列と、頂点数 $2 \times N$ の二部完全グラフのマッチングには全単射が存在するので、二部グラフの完全マッチングで考えます。全単射は、二部完全グラフを $L_1, L_2, ..., L_N, R_1, R_2, ..., R_N$ として、$a_i = j$ と辺 (L_i, R_j) を対応させると作れます。

するとこの問題で数えるべきは、辺 (L_i, R_{K+i})、辺 (L_i, R_{K-i}) のうちどれか 1 つは使う完全マッチングの個数、と考えることができます。ここで包除原理を使用します。すると、上記の辺のうちいくつかを選び、それらを全て含む完全マッチングの個数を数える問題になります。

ここで注目したいのは、辺を A 本選んだとすると、条件を満たす完全マッチングの個数は、0 か $(N - A)!$ であるということです。もし選んだ辺たちがマッチングではないならば当然 0 個で、マッチングならば残りの頂点の割り当て方が $(N - A)!$ 通りあります。

これにより、$2N$ 個について包除しているため本来 2^{2N} 通りについて調べる必要があるのが、選んだ個数ごとにまとめることができます。つまり、$2N$ 個の辺からマッチングとなるように A 個選ぶ方法は何通りか？という問題を $A = 1, 2, ..., 2N$ について解けば良いことがわかります。

辺たちの形が非常に単純で、直線型のグラフが何本か合わさった形になっているので、これは簡単な DP で $O(N^2)$ で求めることができます。

以上よりこの問題は $O(N^2)$ で解けました。

おまけ: 以上の考察をもう少し進めると、この問題は $O(NlogN)$ で解くことが出来ます。

E: Sugigma: The Showdown

この問題はしぐま君とすぎむ君がそれぞれ木を持っていて、その上でしぐま君は逃げて、すぎむ君は追う問題です。

まず、しぐま君の辺 (u, v) について、$u - v$ 間のすぎむ君の辺での距離が 3 以上ならば、その辺をつかうとしぐま君は永遠に逃げ続けられます。その辺の端点で待機し、すぎむ君が近づいてきたら辺を使い移動する、を繰り返えすだけです。よって、そのような辺が生えている頂点はしぐま君にとっての必勝頂点 (その頂点でしぐま君が手番をとったら永遠に逃げ続けられる) としてよいです。

以上の考察より、しぐま君は、距離 1,2 の辺 + いくつかの頂点が必勝頂点を持っていると考えられます。

また、すぎむ君は、常にしぐま君に近づくように動くのが最善戦略です。なぜならば、もしすぎむ君が離れたら、しぐま君はすぎむ君が戻ってくるまでパスをし続けることでターン数を稼ぐことができるからです。よって、実はすぎむ君のするべき行動は一意に定まり、あとはしぐま君の行動について考えればよいです。

すぎむ君の木を、すぎむ君が最初にいる頂点を根とした根付き木に変形し、その上で考えると見通しが良くなります。すると、「根付き木が与えられ、しぐま君はどこかにいて長さ 1 や 2 のジャンプが出来て、すぎむ君は最初は根にいて長さ 1 のジャンプが出来る。また、いくつかの頂点でしぐま君が手番を取るとしぐま君の勝ち」という問題になります。長さ 1 や 2 のジャンプしか出来ない時に重要な性質として、「しぐま君がすぎむ君を飛び越すと、その次のターンでしぐま君は捕まる」という性質があります。もし飛び越えても、すぎむ君の隣にしか行けないからです。そして、飛び越えてもターン数が伸びるわけではないので、飛び越える利点はないです。よって、「しぐま君はすぎむ君を飛び越えることはない」としてよいです。

よってしぐま君は常にすぎむ君の下にいて、すぎむ君は毎ターンしぐま君の方に降りていく、という行動をすることになります。これは、「しぐま君は i 回目の手番では深さ i+1 以上のところのみを動ける」、という解釈にすることが出来ます。

このように解釈すると、ゲームからすぎむ君を消して、しぐま君の一人ゲームとして考えることが出来ます。しぐま君が、毎ターン strict に範囲が狭まっていく木の中で、一人で逃げ続けるゲームです。すると、

- しぐま君は、後戻りをして得をすることはない
- しぐま君は、一度パスをしたならばその後ずっとパスをし続けるのが最善戦略

ということが簡単に示せます。

なので、しぐま君の動きを、しぐま君の木において dfs をすることで全探索し、この問題は、$O(N)$ で解くことが出来ます。

F: Many Easy Problems

まず、K が固定されている場合から考えます。もちろん、${}_N\mathrm{C}_K$ 通りそれぞれについて調べていてはとても間に合わないので、視点を変える必要があります。

辺ごとに、その辺が部分木に使われるような選び方は何通りか？というのを求め、それらの総和を取ることを考えます。辺の左右の部分木からそれぞれ 1 個以上頂点を選んだ場合、その辺は部分木に含まれることがわかります。よって、辺がサイズ $A, N-A$ の木をつないでいるとすると、この辺を使う選び方は ${}_N\mathrm{C}_K - {}_A\mathrm{C}_K - {}_{N-A}\mathrm{C}_K$ 通りです。

なので、まず $A, N-A$ を最初に列挙しておきます。そしてそれらを並べ、数列 a_i としておくと、

$$(N-1){}_N\mathrm{C}_K - \sum_{i=1}^{2(N-1)} {}_{a_i}\mathrm{C}_K$$

が答えとなります。

これは階乗を前計算しておけば $O(1)$ で求めることができるので、K が固定されている場合は $O(N)$ で解くことが出来ました。

満点解法を考えます。まず、数列 b_i を、$b_i = (a_1, a_2, ...$ のうちの i の個数 $)$ とすると、

$$(N-1){}_N\mathrm{C}_K - \sum_{i=1}^{N-1} b_i \times {}_i\mathrm{C}_K$$

を求める問題になります。

ここで、${}_N\mathrm{C}_K = \frac{N!}{K!(N-K)!}$ を使うと、

$$\sum_{i=1}^{N-1} b_i \times {}_i\mathrm{C}_K$$

$$= \sum_{i=1}^{N-1} b_i \frac{i!}{K!(i-K)!}$$

$$= \frac{1}{K!} \sum_{i=1}^{N-1} (b_i \times i!) \frac{1}{(i-K)!}$$

と変形できます。よって、$c_i = b_i \times i!$、$d_i = 1/i!$ とおくと、$c_i \times d_{i-K}$ の総和を求めれば良いことがわかります。これは NFT を用いることで $K = 1, 2, ..., N$ について、まとめて $O(NlogN)$ で求めることが出来ます。

今回は mod が $10^9 + 7$ では無く 924844033 です。1 を引いて素因数分解をしてみると、$924844033 = 2^{21} * 441 + 1$ であることがわかります。これが今回 mod を変更した理由で、$Z/[924844033]$ において、2^{21} 乗すると初めて 1 になる元が存在します。なので、他に特殊な mod を用意しなくても、直接

NFT を行うことが出来ます。(Time Limit は他の mod を経由して NFT をしても十分に間に合うように調整しました。) 余談ですが、原始元は 5 なので、$5^{441} = 44009197$ が 2^{21} 乗すると初めて 1 になる元です。

以上より、この問題は $O(NlogN)$ で解けました。

AtCoder Grand Contest 005 Editorial

Kohei Morita(yosupo)

October 1st, 2016

A: STring

Instead of 10^{10000} operations, $|X|/2$ operations are sufficient.

0.1 200 points

You can find the leftmost occurrence of the string "ST" in the given string in $O(|X|)$. Also, you can remove this substring from the string in $O(|X|)$. Since there are at most $|X|/2$ operations, you can solve this problem in $O(|X|^2)$ by simulation.

0.2 300 points

You can get an intuitive understanding of this task by replacing 'S' to '(' and 'T' to ')'. In each step, you remove a pair of matching parenthesis.

Prepare an empty stack. Scan the string from the first character the last character, and do the following:

- If the current character is 'S' (or '('), push it to the stack
- If the current character is 'T' (or ')'), and the topmost character in the stack is 'T' (')') or the stack is empty, push it to the stack
- If the current character is 'T' (or ')'), and the topmost character in the stack is 'S' ('('), pop the topmost character from the stack

The answer is the size of the stack after this process. This way you can solve the task in $O(|X|)$ time.

B: Minimum Sum

In this task, for each interval you compute the minimum, and the answer is the sum of these minimums. There are $O(N^2)$ intervals and straightforward solutions will get TLE.

Instead, for each i, compute the number of intervals whose minimum is a_i. Let $f(i)$ be this number. Then, the answer is the sum of $a_i f(i)$.

For each index i, we compute l and r:

- l is the biggest integer such that $l < i$ and $a_l < a_i$
- r is the smallest integer such that $r > i$ and $a_r < a_i$

Then, in order for a_i to be minimum, the leftmost value in the interval must be one of $l+1, l+2, \ldots, i$, and the rightmost value must be one of $i, i+1, i+2, \ldots, r-1$. Thus, there are $(i-l) \times (r-i)$ such intervals.

We can compute the values of l and r in the increasing order of a_i. We keep a set of indices (for example std::set in C++), and in the decreasing order of a_i, add i to the set. The values of l, r are the two integers adjacent to i in this set.

Thus, this problem can be solved in $O(NlogN)$.

C: Tree Restoring

Let $\text{dist}(u, v)$ be the distance between two vertices u and v.

The diameter of the tree must be $\max(a_1, a_2, ..., a_N)$. Let a, b be the two endpoints of a diameter.

In general, we can prove that for any vertex p, the following holds:

$$\max(\text{dist}(p, 1), \text{dist}(p, 2), ..., \text{dist}(p, N)) = \max(\text{dist}(p, a), \text{dist}(p, b)) \quad (1)$$

Let K be the length of the diameter. There are $K + 1$ vertices on the path between a and b, and by using the property above, the values that correspond to these vertices are:

- When K is even, the values are
  $\{K, K-1, K-2, ..., K/2+1, K/2, K/2+1, ..., K-1, K\}$
- When K is odd, the values are
  $\{K, K-1, K-2, ..., (K+1)/2+1, (K+1)/2,$
  $(K+1)/2, (K+1)/2+1, ..., K-1, K\}$

First, from the given multiset of values, remove values that correspond to these values. If this is impossible, the answer is clearly "Impossible".

Consider a vertex x that is not on this path. We can see that when K is even, the value corresponds to x must be at least $K/2 + 1$, and when K is odd, the value must be at least $(K + 1)/2 + 1$. On the other hand, by choosing an appropriate vertex on the path and add a vertex adjacent to this vertex, we can construct all other cases.

Thus, if all the remaining values are at least this bound, the answer is "Possible". Otherwise the answer is "Impossible".

D: K Perm Counting

Consider a bipartite graph with $2N$ vertices. The vertices are labelled $L_1, \ldots, L_N, R_1, \ldots, R_N$. A perfect matching in this bipartite graph corresponds to a permutation. A permutation $p_1, \ldots, p_N$ corresponds to a matching that uses the edges (L_i, R_{p_i}). In this task, you are asked to count the number of perfect matchings that doesn't use edges (L_i, R_{i+K}) and (L_i, R_{i-K}).

By inclusion-exclusion principle, we get the following. Consider a bipartite graph that consists of bad edges: (L_i, R_{i+K}) and (L_i, R_{i-K}) for each i. Let M_k be the number of matchings of size k in this graph. Then, the answer is

$$\sum_{i=0}^{N} M_i (N-i)!(-1)^i \tag{2}$$

Thus, it is sufficient to compute the value of M_k for each k.

Since this graph is a disjoint union of paths, the values of M_k can be easily computed in $O(N^2)$ time in DP. Therefore, this problem can be solved in $O(N^2)$.

Exercise: can you solve this problem in $O(NlogN)$?

E: Sugigma: The Showdown

In this task, each of Sigma and Sugim has a tree (on the same set of vertices). Sigma is the first player. He starts from the vertex X and tries to escape from Sugim using the red tree. Sugim is the second player. He starts from the vertex Y and tries to capture Sigma using the blue tree.

In this editorial, let $D_r(u, v)$ be the distance between the vertices u and v along the red tree. Define $D_b(u, v)$ similarly for the blue tree.

Suppose that there is an edge $u - v$ in the red tree, and $D_b(u, v) \geq 3$. Let's call this type of edges *long*. If Sigma reaches one of the vertices u or v and Sugim can't capture him right after that, we can prove that Sigma can escape forever. When Sigma is at the vertex u, he waits at this vertex until Sugim reaches one of the vertices adjacent to u (in the blue tree). Then, right after Sugim reaches one of the adjacent vertices (let's call it x), he moves to the vertex v. Since $D_b(u, v) \leq 3$ and $D_b(u, x) = 1$, $D_b(v, x) \leq 2$ and he can't capture Sigma in the next turn. Similarly, when Sigma is at the vertex v, he wais there until Sugim reaches one of the adjacent vertices, and then return to u again. This way Sigma can escape from Sugim forever.

Consider the blue tree as a rooted tree rooted at Y. Some vertices of this rooted tree are marked as "special". These vertices correspond to the endpoints of the long edges, and if Sigma reaches one of thesse vertices (and Sugim can't capture him right after that), Sigma wins. Also, there are some additional red edges. When a red edge connects two vertices u and v, $D_b(u, v)$ is 1 or 2.

We can consider these red edges as "jumps". The game becomes a game on a rooted tree. Sigma jumps around the tree (the jumps are always short), and tries to reach one of the special vertices or tries to escape from Sugim as long as Sugim.

An important observation is that, we can assume that Sigma never jumps over Sugim. When Sigma jumps over Sugim, since the jump length is at most two, both endpoints of the jump must be adjacent to Sugim, and Sigma will be captured right after the jump. It is not better than staying at the vertex he jumped from. In other words, Sigma is always in the subtree rooted at Sugim.

We can assume that Sugim always move downwards to Sigma's direction. If Sugim follows this strategy, whenever Sigma reaches vertex x after t steps such that $D_b(Y, x) \leq t$, he can capture Sigma. On the other hand, as long as Sigma keeps the condition $D_b(Y, x) > t$, obviously Sugim can't capture Sigma. Thus, this is the optimal strategy for Sugim.

Now this game can be regarded as a single-player game for Sugim. In

Sigma's path, if the i-th (0-based) vertex is v_i, it must satisfy $D_b(v_i, Y) > i$. Under this constraints, if he can reach one of the special vertices, he wins (and the answer is -1). Otherwise, the answer corresponds to the maximum length of such paths. This can be done by a simple dfs.

Therefore, this problem can be solved in $O(N)$ time.

F: Many Easy Problems

First, solve the problem for a fixed value of K.

For each edge e, count the number of ways to choose K vertices such that the edge e is included in the subtree. If we compute the sum of these values for all edges, we can compute the sum of number of edges in all subtrees defined by K vertices. Since the number of vertices in a tree is the number of edges plus one, the answer is this sum plus the total number of ways to choose K vertices from the N vertices.

Assume that if we cut the given tree by the edge e, we get two subtrees of the sizes A and $N - A$. Then, the number of ways to choose K vertices that include this edge is $\binom{N}{K} - \binom{A}{K} - \binom{N-A}{K}$. Thus, for each we can compute this value, and the answer is the sum of these values plus $\binom{N}{K}$. This can be done in $O(N)$.

We can simplify this solution a bit. First, compute the frequency list of the sizes of subtrees obtained by cutting a single edge. We can convert it to a sequence of coefficients $b_0, b_1, \ldots, b_N$ ($b_i = N$ if $i = N$, otherwise b_i is minus the number of subtrees of size i), and the answer is simply the following:

$$\sum b_i \binom{i}{K} \tag{3}$$

Let's compute this value for all K efficiently.

By using $\binom{i}{K} = \frac{i!}{K!(i-K)!}$,

$$\sum b_i \binom{i}{K} \tag{4}$$

$$= \sum b_i \frac{i!}{K!(i-K)!} \tag{5}$$

$$= \frac{1}{K!} \sum (b_i \times i!) \frac{1}{(i-K)!} \tag{6}$$

Let $c_i = b_i \times i!$ and $d_i = 1/(-i)!$. Then, this value can be computed by the sum of $c_i \times d_{K-i}$. This can be seen as a convolution, thus we can compute these values for all K using FFT in $O(NlogN)$.

Since the modulo is 924844033, we can use NFT instead of FFT to avoid precision errors. The modulo is $924844033 = 2^{21} * 441 + 1$, and thus there exists a g such that $g^{2^{21}} = 1$ and any smaller powers of g is not one. For example, if we use a primitve root 5, we can get $g = 5^{441} = 44009197$. We can use this g instead of a root of unity in FFT.

In summary, this problem can be solved in $O(NlogN)$ time.