CODE FESTIVAL 2016 Elimination Tournament Round 1 A 問題 解説

yutaka1999

i 回目のゲームを考える。このとき、選ぶ辺集合が満たすべき条件を言い換えると、以下のようになる。

条件． これらの辺によってつくられる連結成分がいずれも S_i または T_i を含む。

よって、S_i と T_i を同一視した場合に、その辺集合が全域木を含むことが条件であり、その重みの和が最小となるのは明らかに最小全域木（以下、MST と略す）になるときである。さらに、S_i と T_i を同一視することは、S_i と T_i の間に重み 0 の辺を追加することとしてもよいので、以下の問題を解けばよいことになる。

問題． S_i と T_i の間に重み 0 の辺を追加したときの MST の重みを求めよ。

よって、$Q = 1$ の時はクラスカル法などを用いれば $O(M\alpha(N))$ で解くことができる。次に、部分点 2 の制約を考える上で、以下の補題を示す。

補題． S_i と T_i の間に重み 0 の辺を追加したときの MST の重みは元のグラフの MST と追加した辺からなるグラフの MST の重みに等しい。

これは、クラスカル法のアルゴリズムを考えれば明らかである。辺を追加したグラフに対してクラスカル法を行った場合、元からある辺が MST に含まれるかどうか判定される順番を変える必要がない。その場合、元のグラフで MST に使われないと判定された辺は明らかにこのグラフ上でも MST に使われないと判定される。よって、示された。

以上を用いると、部分点 2 では、クラスカル法に用いる辺が毎回 N 本となるため、$O(QN\alpha(N))$ で解くことができる。

次に、満点を考える。正の重みの辺からなる MST に重み 0 の辺を加える場合、明らかにその辺は MST に使われ、それにより MST として使われなくなる辺がちょうど一つ存在する。その辺はもとの MST 上で S_i と T_i を結ぶパス上にある重みが最大の辺である

ことがクラスカル法のアルゴリズムを考えると分かる。よって、MST 上のパスの最大重みを求めればよく、これはあらかじめ $O(N^2)$ かけて前計算しておくことで、各ゲームで $O(1)$ で答えることができる。

以上より、$O(N^2 + Q)$ で解くことができるので、満点を得ることができる。

CODE FESTIVAL 2016 Tournament Round1 B 問題解説

chokudai

部分点 1

文字列長が 2 以下の時、カンマを間に入れるか入れないかの 2 通りしか存在しない。

部分点 2

文字列長が 16 以下の時、カンマを入れることが可能な箇所は 15 箇所以下である。よって、カンマを入れる/入れないについて、全通り試せば良い。

部分点 3・満点

解としてあり得る値について考える。文字列の長さを N とすると、$L = (N+K)/(K+1)$ とした時、必ず長さ L 以下にすることが出来る。何故なら、カンマを利用することで、長さ N の文字列を出来るだけ均等な長さに分けた時、長さ L の文字列と、長さ $L-1$ の文字列だけに分けることが出来るからである。

長さ $L-1$ 以下にすることが出来ないのも同様に示せるため、求める答えは、 S から、連続する L 文字の部分文字列を抜き出したものであることが解る。

この文字列は、何文字目から文字列を取り出すかを考えれば、最大でも $N-L+1$ 通りしか存在しない。S の i 文字目から L 文字取り出した文字列が表す整数を A_i とした時、答えが A_i 以下であるか、つまり、A_i 以下の整数のみに分割する方法が存在するかどうかを高速に判定出来ると良い。

この判定は、文字列 S の頭から、貪欲に整数を取り出していくようなアルゴリズムで行うことが出来る。文字列の先頭 L が表す整数が A_i 以下であれば L 文字後にカンマを入

れ、そうでなければ $L-1$ 文字後にカンマを入れる、という処理を K 回繰り返すことで、実際にカンマを用いて全ての整数を A_i 以下に出来るか高速に判定することが出来る。

ここで、A_i と A_j の大小関係が O(1) で判定出来れば、O(N) で判定を行うことが出来る。これは、Suffix Array を事前に構築しておくことで、そのまま比較することが可能である。

ここまで出来れば、後は、答えを二分探索すれば良い。前述したとおり、解候補は $N-L+1$ 通りしか存在しないため、Suffix Array を使って二分探索を用い、答えを求めることが可能となる。

CODE FESTIVAL 2016 Elimination Tournament Round 1 Problem A Editorial

yutaka1999

Consider the i-th round of the game. The condition that should be satisfied by the set of the chosen edges can be rephrased as:

Condition . *Every connected component formed by the edges contains either S_i or T_i.*

Thus, the condition is equivalent to the following: if S_i and T_i are regarded as the same vertex, the set of edges contains a spanning tree. Obviously, the total weight is minimized when the set is a minimum spanning tree (MST) itself. Regarding S_i and T_i as the same vertex is equivalent to adding an edge of cost 0 connecting S_i and T_i, so the problem is equivalent to:

Problem . *Find the weight of an MST when there is an additional edge of cost 0 connecting S_i and T_i.*

Therefore, the case where $Q = 1$ can be solved in $\mathrm{O}(M\alpha(N))$ time using a method such as Kruskal's algorithm. To solve the problem under the constraints for the second partial score, we will show the following lemma:

Lemma . *The weight of an MST when there is an additional edge of cost 0 connecting S_i and T_i, is equal to the weight of an MST of the graph obtained by adding an edge of cost 0 connecting S_i and T_i to an MST of the original graph.*

This is obvious considering the Kruskal's algorithm. When Kruskal's algorithm is executed on the graph with an additional edge, the edges that are originally in the graph are examined in the same order as when Kruskal's algorithm is executed on the original graph. Thus, the edges that are not selected by the Kruskal's algorithm on

the original graph will also be not selected by the Kruskal's algorithm on the graph with an additional edge. Therefore, the lemma is shown.

Using this, the number of the edges on which Kruskal's algorithm is executed will become N under the constraints for the second partial score, and the problem can be solved in $\mathrm{O}(QN\alpha(N))$ time.

We will now consider a solution for full credit. When an edge of cost 0 is added to an MST consisting of edges of positive costs, that edge is obviously used in an MST, and there is one edge that is no longer used in an MST because of the addition of that edge. Considering the Kruskal's algorithm, that edge is an edge with the maximum weight along the path connecting S_i and T_i. Thus, we can precalculate the maximum weight of a vertex connecting each pair of vertices in the MST in $\mathrm{O}(N^2)$ time, to process each round of a game in $\mathrm{O}(1)$ time.

Therefore, the problem can be solved in $\mathrm{O}(N^2 + Q)$ time, which is enough to earn full credit.

CODE FESTIVAL 2016 Tournament Round1 Problem B Editorial

chokudai

Partial Score 1

When the length of the string is at most 2, there are at most two cases: whether a comma is inserted or not.

Partial Score 2

When the length of the string is at most 16, there are at most 15 positions where a comma can be inserted. Thus, we can try every possibility for inserting comma to each position.

Partial Score 3 / Full Credit

We will consider what values can be the answer. Let the length of the string be N, and let $L = (N + K)/(K + 1)$, then we can separate the string so that the string corresponding to the maximum number has a length of most L. This is because the string can be separated into strings of length L and $L - 1$, by separating it as evenly as possible.

It can be similarly shown that we cannot separate the string so that the string corresponding to the maximum number has a length of less than L, thus the answer is a contiguous substring of S with length L.

There exists at most $N - L + 1$ such substrings, considering the starting position of a substring. Let A_i be the contiguous substring of S with length L, starting from the

i-th character in S. We want to efficiently determine whether the answer is at most A_i, that is, there exists a way to separate the string into numbers, all of which are at most A_i.

This can be done by a greedy algorithm to cut out as large substrings as possible from the beginning of S. That is, if the number represented by the first L characters of the string is at most A_i, then we cut out the first L characters, otherwise we cut out the first $L-1$ characters. By executing this process K times, we can determine whether it is possible to separate the string into numbers, all of which are at most A_i.

If the magnitude relationship between A_i and A_j can be determined in $\mathrm{O}(1)$ time, the algorithm above can run in $\mathrm{O}(N)$ time. This comparison is possible by constructing Suffix Array in advance.

Then, we can perform binary search on the answer. As mentioned above, there are only $N-L+1$ candidates for the answer, thus the problem can be solved by performing binary search using Suffix Array.