The 2021 ICPC Asia Macau Regional Contest

Prepared by SUA Programming Contest Setter Team

April 3, 2022

A. So I'll Max Out My Constructive Algorithm Skills

- Find any simple path, either itself or the reversed will satisfy the requirement.

B. the Matching System

Construction

- We can find the worst construction of both max matching and min matching by hands or bruteforce.
- Max matching: **...**0* and 011...11
- Min matching: **...**^ ^...^ ^ (* is 1 or 2 more than ^) and 00..00
- We can find that the latter's cost increasing is much slower.

Counting

Besides direct counting by Combinatorics, there is another way to compute the cost: Define $dp_{i,j}$ as the cost of the mode string to i and the matching string to j. Transition can be done by a bit of case analysis. The pure time complexity is $O(n^3)$, which can be optimized to $O(n^2)$ by prefix sums. After we find the construction, we can use this method to avoid the trouble of Combinatorics.

C. Laser Trap

- For Baobao to escape, $(0, 0)$ must be outside the convex hull of laser generators
- Thus, all survived laser generators must be within a cone of $\frac{\pi}{2}$ degree (a closed half-space).
- Sort all generators by their polar angle, find the maximum number with a two-pointer.
- The time complexity is $O(n \log n)$.

D. Shortest Path Fast Algorithm

Observation

After adding node u to the heap and before popping it out, his neighbor nodes can still update u's shortest path. However, the corresponding value in the heap won't change, which might cause huge difference.

General Idea

We can push u in the heap with a very large value, then update to a relatively value by his neighbor nodes. Then we let u to be a key node such that most nodes' shortest path must pass u. Therefore, their shortest path can only be updated correctly after popping u. However, u's initial large value will 'stuck' it in the heap.

D. Shortest Path Fast Algorithm

A Basic Union of 4 nodes

- Connect 1 and 2 with a large edge.
- Connect 3 and 4 with a large edge but smaller than $(1, 2)$.
- Connect 1 and 3, 2 and 3, 2 and 4 with an edge of weight 1.

Result

Starting from 1, 2 and 3 will enter the heap first. Then popping 3 will update the shortest path to 2 and 4. Because 2 has a larger value, 4 will be popped first. In this way, 2 will become the 'stuck' key node.

D. Shortest Path Fast Algorithm

- Connect multiple basic unions one by one and adjust the weight of edges which they are not of weight 1 initially.
- Each key node of unions will cause a round of every nodes being updated wrongly before it got popped out. In this way, we can hack the algorithm.
- Our hacking method is related to the number of nodes, edges and range of edge weight.

E. Pass the Ball!

- By observation, we can know that $p_1, p_2, \dots, p_n$ is a misplaced permutation of $1, 2, \dots, n$.
- construct a directed graph of n nodes. Connect all $p_i \rightarrow i$.
- This graph is consist of multiple disjoint directed cycles.
- After k rounds of passing, for person i, b_i is the number of nodes starting from i after k steps.

E. Pass the Ball!

- For each cycle of length m, the passing ball process will cycle every m rounds.
- For $t = 0, 1, 2, \ldots, m-1$ rounds' result, we compute the sum of nodes' indexes $\times$ the succeed nodes' index after t rounds separately.
- This part is similar to computing the cyclic convolution between two sequence. We reverse one of the sequences and use Fast Fourier Transform to compute. In this way, we can achieve the complexity of $O(m \log m)$. The total time complexity of this part is $O(n \log n)$.
- There are only $O(\sqrt{n})$ kinds of different length. So we add up the contribution, then answer each query by enumerating length of cycles. This part is $O(q\sqrt{n})$.
- The total time complexity is $O(n \log n + q\sqrt{n})$.

F. Sandpile on Clique

Theorem

If at least $n-1$ times of topping happen, this sandpile will never terminate.

Proof

For any continuous $n-1$ times of topping, there must be at least one node which is not included in these $n-1$ times. For this node, it must receive $n-1$ chips without losing any of them. Then it will reach the condition of topping.

Use any data structure to simulate the first $n-1$ rounds.

G. Cyclic Buffer

- If there is one integer cannot be accessed, we must move it to either the 1_{st} or k_{th} position.
- Denote $f_{i,0}$ as the minimum cost when visiting i and i is at the 1_{st} position.
- Denote $f_{i,1}$ as the minimum cost when visiting i and i is at the k_{th} position.
- We can use Fenwick Tree to maintain $w_{i,0}$ and $w_{i,1}$, which is denoted as the number of integers that can be accessed directly after visiting i at position 1_{st} or k_{th}.
- $f_{i,0} \to f_{i+w_{i,0},0/1}$.
- $f_{i,1} \to f_{i+w_{i,1},0/1}$.

H. Permutation on Tree

Observation

We enumerate x from 1 to $n-1$. For a permutation, we change all integers $\leq x$ as 0. Otherwise change them as 1. The answer will be the number of 01 flipping in all sequence generating in this way.

W

e maintain the following 3 values x, y, z:

- x: How many kinds of 01 sequences of the subtree rooted at u.
- y: How many kinds of 01 sequences of the subtree rooted at u such that k is $0/1$.
- z: How many kinds of 01 sequences of the subtree rooted at u such that the k_{th} bit is different from $(k+1)_{th}$.

H. Permutation on Tree

- For y, we enumerate the k_{th} bit occurs in which subtree's sequence's some positions, then we merge the answer with the subtree we enumerated before.
- Note that we don't directly enumerate k, but the subtree u and position x, and how many nodes y occurs before k in previous enumerated subtree. Thus $k = x + y$. For the whole tree, the complexity of this part is $O(n^2)$.
- For z, we discuss whether the k_{th} and $(k+1)_{th}$ bit are from the same or different position. We also enumerate the three values as we do in computing y. The complexity of this part is also $O(n^2)$.
- the total time complexity is $O(n^3)$. The answer is the sum of z in the subtree rooted at r.

I. LCS Spanning Tree

- By constructing the extended suffix automaton of all strings, for all suffixes' appearance of the same string on the automaton, we call it as a color. Therefore, for a common substring of string i and j, it corresponds to a right set such that includes both i and j.
- Consider the process of the Kruskal algorithm, we enumerate every node first by their length from small to large then by their depth from large to small.
- For each node u, when it is enumerated, because every child of it has a length no smaller than itself, thus for each child v, all colors in v's right set have already been merged to one component.

I. LCS Spanning Tree

- In this way, we only have to merge colors from different subtrees. Since each subtree only corresponds to one component, we can merge them directly.
- The total time complexity is $O(n \log n)$.
- If you use Suffix Array and construct the Cartesian Tree of the Height array, the following process is the same. But due to the judge's speed, you might need to use the faster algorithm such as SAIS or DC3.

J. Colorful Tree

- We can build the tree at the beginning and activate nodes one by one. Without considering colors, we have the following two theorems for computing diameter under rounds of activation:

Theorem 1

For a node set S, if we already know the node pair (x, y) of the diameter. When adding a new node p into S, we only need to try updating the original diameter by the path between p and x/y.

Theorem 2

For two node set S_1 and S_2 and their node pair (x_1, y_1), (x_2, y_2) of the the diameter. When merging S_1 and S_2, the new diameter node pair must consist of two nodes from x_1, y_1, x_2, y_2.

J. Colorful Tree

- With these two theorems, if we compute sparse table in $O(n \log n)$ for answering LCA queries in $O(1)$, we can maintain the diameter for adding a node or merging node sets in $O(1)$.
- We can use a segment tree to maintain the diameter node pair. The index is the tree node. Initially, all leaves are inactivated. Each activation will trigger a process merging from leaf to root on the segment tree.

J. Colorful Tree

- Now let's consider colors. We define the total number of operations as m. If we are computing the diameter node pair of the same color, we can build a segment tree T_c for each color c. The leave represent every node that might become this color. This part can be precompute offline. Since the total number of $(node, color)$ is $O(m)$. For each operation, we modify the leaf and update the path from leaf to root.

J. Colorful Tree

- Now let's consider computing the diameter node pair of different colors. We still maintain a segment tree T_c for each color. Additionally, we maintain a segment tree G of m leaves.
- In G, we not only maintain the diameter node pair (x, y) and corresponding distance *dis* in each node, but also the diameter distance *apart* of different colors. When we activate a node or modify one's color, after merging T_c, we put its root information (x, y) and *dis* to G's leaf node c. This leaf's *apart* is 0. Then we update from leaf to root.
- The answer will be the *apart* of G's root. The time complexity is $O(m \log m)$ and the space complexity is also $O(m \log m)$ (the sparse table).

K. Link-Cut Tree

- Because the weight of 2^i, we can simply sort all edges by weight, then add them one by one until the first cycle showing up.
- Maintain with Disjoint set, similar to the classic Kruskal algorithm for finding the minimum spanning tree.

Thank you!