Problem Tutorial: "Add One"

Note that for $x = (????0\underbrace{11\cdots 11}_{k})_2$, $x+1$ equals to $x \oplus (2^{k+1}-1)$. So we can enumerate the value of k, and check if we can get a value in the form of $(????0\underbrace{11\cdots 11}_{k})_2$. This can be done by using the linear basis.

Time Complexity is $O(nw^2)$ if we assume $a_i = O(2^w)$. It can also be optimized to $O((n+w)\cdot w)$ (but is not required).

Problem Tutorial: "Be Careful"

Let's denote the set of the sons of vertex u as S_u, and $d_u = |S_u|, \delta_u = \max_{v\in S_u} d_v, t_u = \sum_{v\in S_u}[d_v > 0]$. Note that for all non-leaf vertex u, the number written on vertex u will be between 0 and d_u.

Our first insight is using dynamic programming to solve the problem. Let $f_{u,v}$ denotes the number of ways to fill the subtree of the vertex u, so that the number writing on vertex u will be exactly v.

Part 1

The first way to calculate f is using the inclusion-exclusion principle. Let dp_S be the number of ways if all the numbers written on its children don't belong to S. First, it seems we need $O(2^{d_u})$ states, but note that we don't need to add the contribution of the leaf children to the state. So there are only $O(2^{\delta_u})$ states.

Part 2

The second way to calculate f is using bitmasks dp. Let $f_{i,S}$ be the number of ways that:

+ Let's fill the numbers from small to large, + We have considered the numbers $0, 1, \cdots, i$, + The set of unfilled sons is S

There will be $O(d_u \cdot 2^{d_u})$ states. But we can also treat all leaves independently. Let's add a dimension $[k]$ in our dp, which means an additional constraint that there are k leaves that have been filled. By using SOS dp, the time complexity will be $O(2^{t_u}\cdot \text{poly}(t_u))$

Part 3

If we use these two ways at the same time, we will have an $O(\min(2^{t_u}\cdot\text{poly}(t_u), 2^{\delta_u}\cdot\text{poly}(\delta_u)))$ solution, but it's still not enough to pass it, because t_u and δ_u can be large at the same time.

So let's try to split the set of the sons of u and use an sqrt-decomposition-like process. For a threshold B, let's split all the sons with $d_v \le B$ to the set A_u, and the others to the set B_u.

Let's run the DP described in Part 1 for A_u, and run the DP described in Part 2 for B_u simultaneously. When solving A_u, we need to know the states in B_u. Let $l = |B_u|$, then the time complexity will be $O(2^{B+l})$.

Note that if all $B+l$ is at least M, then we have $\sum_v d_v \ge \frac{M(M+1)}{2}$, but $\sum_i d_i = n-1$, so $M = O(\sqrt{n})$ (or more precisely, $2\sqrt{n}+O(1)$).

By selecting an optimal M while doing the dp process, the time complexity will be $O(2^{\sqrt{2n}}\cdot\text{poly}(n))$

Problem Tutorial: "Counting Sequence"

Let $f_{i,j}$ be the sum of sequences that satisfy the sum of its elements is i and the last element is j. We have $f_{0,0}=1$, $f_{i,0}=0 (i>0)$ and $f_{i,j} = c\cdot f_{i-j,j-1} + f_{i-j,j+1}$. But there are $O(n^2)$ reachable states, so it's not acceptable.

Note that if $a_1 = B$, then the sum of $a_1, a_2, \cdots, a_m$ is not less than $\sum_{i=1}^{m}(B-i) = mB - \frac{m(m+1)}{2}$. Because the sum should be exactly n, the length of the sequence will be no more than $O(n/B)$.

So we can consider the sqrt-decomposition trick. If $a_1 \le B$, then we can run the naive DP solution, the time complexity is $O(nB)$. If $a_1 > B$, then the length of the sequence is $O(n/B)$, and the maximum value of $|a_i - a_1|$ will also be $O(n/B)$. Let $g_{i,j,k}$ be the sum of sequences that satisfy:

+ The length of the sequence is $i + \sum_{t \le i}(a_t - a_1) = j + a_i - a_1 = k$

The time complexity will be $O(nB + (n/B)^4)$. If we take $B = O(n^{3/5})$, then the overall time complexity is $O(n^{8/5})$. With some constant optimization, you might be able to pass it with this solution.

But we can actually go further. Let's not insert an element at the end of the sequence, but at the beginning of the sequence. Let $f_{i,j}$ be the sum of the sequences that satisfy the length of it is i, and the sum of $a_t - a_1 = j$. When doing the transaction, we can either add an $a_1 - 1$ or $a_1 + 1$ as a new a'_1, but note that it contributes $\sum(a_t - a_1)$ either i or $-i$. The time complexity is $O(nB + (n/B)^2)$, which is $O(n\sqrt{n})$ if we take $B = O(\sqrt{n})$.

After calculating $f_{i,j}$, we can just enumerate the value of $a_1 > B$ and use the DP table to update the answer. The time complexity is $O(n\sqrt{n}))$.

Problem Tutorial: "Exciting Travel"

The problem is (almost) equivalent to assigning each vertex to at most one path. Your task is to maximize the number of paths such that all vertices in it have been assigned to it. Note that for two adjacent paths, the common endpoints of the two paths can be used twice.

For each query, let's build a virtual tree of the endpoints of the paths. Let f_u denote the answer to the subtree of u. For $l_i = \text{LCA}(p_i, p_{i+1})$, enumerate whether this path is selected at lca. Consider all transitions like $\sum_{u \in path(p_i,p_{i+1}), v \notin path(p_i,p_{i+1})} f_v + 1$.

Let $g_u = \sum_{v \in son(u)} f_v - f_u$, this formula can be rewritten as $\sum_{u \in path(p_i,p_{i+1}), u \ne l_i} g_u + 1 + f_x + f_y$, where $fa_x = fa_y = l_i, x \in path(l_i, p_i), y \in path(l_i, p_{i+1})$. There is a slight difference when p_i or p_{i+1} is an ancestor of another. It is enough to maintain the prefix sum of g in the process of dp. By using the small-to-large technique, the time complexity is $O(N \log N)$, where $N = n + \sum_{i=1}^{q} k_i$.

Problem Tutorial: "Flower's Land"

Solution 1

Consider centroid decomposition, the centroid is rt, calculate the maximum value of the connected component of size k that must contain rt, u. Notice that the path of rt to u must be selected, the remaining part can be divided into two parts: vertex with order greater than dfn_u in preorder traversal and vertex with order less than or equal to $dfn_u - size_u$ in the postorder traversal.

Let $f_{i,j}$ denote that the preorder traversal is $[i, n]$, where j is selected and the order of u, fa_u is $[i, n]$ and u is selected, enumerate whether to select u to decide whether to skip the subtree of u, then $f_{i,j} = \max(f_{i+1,j-1} + a_u, f_{i+size_u,j})$. Similarly, let $g_{i,j}$ denote the values of choice j in the postorder traversal of $[1, i]$, which can be transferred similarly. To find the answer, you can $O(k)$ to combine the $k - \text{dis}(rt, u)$ item of the two arrays. Terminate immediately when the size of the current connected component is less than k, so the time complexity is $O(nk \log \frac{n}{k})$, which is enough to pass.

Solution 2

Consider up and down dp, let $f_{u,i}$ denote the maximum value of the connected component of size i in the subtree of u, and $g_{u,i}$ denote the maximum value of the connected component of size i outside the subtree of u. Consider transition of f, $f_{u,x} = \max_{\sum_v s_v = x-1} f_{v,s_v} + a_u$. For the sons of u $v_1, v_2, \cdots, v_m$, in order to transition g_{v_i}, we define two auxiliary arrays $h^{(1)}_{i,j}$ denotes that outside the subtree u and in the subtree of $v_j (j < i)$, a connected component of size $k - j$ can be selected. And, similarly, $h^{(2)}_{i,j}$ denotes that j is selected from the subtree of $v_j (j > i)$ The maximum value of the vertex. Then there is $g_{v_i,j} = \max_{x-y=j} h^{(1)}_{i,x} + h^{(2)}_{i,y}$. Calculate $h^{(1)}, h^{(2)}$ can be done in a similar way to calculate f, with a time complexity of $O(nk)$.

Problem Tutorial: "Games"

First, consider that no label is given, we can solve it with a tree dp. Let $f(u, i, 0/1)$ denote the number of ways in which only vertices inside the subtree u is considered, a_u is equal to i, and the subtree has/hasn't the value greater than u.

The transition is simple, with prefix sum optimization, and the time complexity is $O(nm)$.

Since only one label will be given, we can consider up and down dp. Similarly, let $g(u, i, 0/1)$ denote the number of ways in which only points outside the subtree u is considered, a_u is equal to i, and the subtree has/hasn't the value greater than u.

The transition from father u to some son v needs to use the f values of u's sons other than v, this can be done by processing prefix/suffix sums.

Finally, the answer to the query can be calculated by combining f_u and g_u.

The total time complexity $O(nm + q)$.

Problem Tutorial: "Half Plane"

The intended solution is applying the algorithms described in *Offline Optimal Range Query and Update Algorithm* by Xinlong Li and Chengze Cai (also the author of this problem). The detailed explanation can be found here: `https://qoj.ac/files/ISAAC_2021_paper_100.pdf`

The short explanation for this task will be added soon.

Problem Tutorial: "Inverse Line Graph"

Consider that the problem is transformed into, assigning two colors to each vertex of the new graph, which represents the endpoints of the original graph, so it is necessary to ensure that the derived subgraph of any color is a complete graph, and each edge is in exactly one complete graph, An edge covered by the color is said to be in the derived subgraph of that color. We choose any vertex u and consider dividing the outgoing edges of u into two sets, representing two colors respectively. Let the out-degree of u be S, and the sets of two colors are S_1, S_2 respectively. Assuming we get the partition right, for each vertex in S, the color has been used, and the remaining edges that are not covered, must become a new complete graph, so it can be processed recursively. Let $x \in S_1, y_1, y_2 \in S_2$ and $(x, y_1), (x, y_2)$, then (x, y_1) and (x, y_2) must be covered a color, but The edges of y_1 to y_2 have been covered, so there is no solution. So the edges between S_1, S_2 are a match. Choose any $p \in S$, let $S_1 = \{p\}, S_2 = S - S_1$, and check it. Otherwise, Then there must be $p, q \in S_1, (p, q)$. Suppose we have the initial condition of $p, q \in S_1$, then for $x \in S$, if there is an edge with p, q, it must be in S_1, otherwise in S_2. For $t \in S_2, (p, t)$, check it again by the above method. If it is still not satisfied, it means that p, q is not in a set, p, t is not in a set, because $t \in S_2$, so t, q has no edge so t, q is not in a set, so there is no solution. Therefore, it can be converted into up to 3 times of testing, each time complexity $O(n + m)$. Total time complexity is $O(\sum n + \sum m)$.

Problem Tutorial: "Just Another Number Theory Problem"

Consider that each number x corresponds to a n tuple $(q_{x,1}, q_{x,2}, \cdots, q_{x,n}), q_{x,i} = x \bmod p_i$, which forms a set of bijective relations.

Note that $(a_i - a_{i-1})^2$ equals to the number of ways to choose two numbers from $[a_{i-1}, a_i)$. Let's call these two numbers l and r, and WLOG let $l \le r$. Then for all x in $[l, r]$, we have $x \bmod p_i \ne 0$ for all $1 \le i \le n$. That is, $q_{x,i} + r - l < p_i$. Let's enumerate $k = r - l$. For $0 \le q_{x,i} < p_i - k$ there are $p_i - k$ values. Swap is also valid when $l \ne r$, so the answer is $\sum_{i=0}^{p_1-1}(1 + [i > 0]) \prod_{j=1}^{n}(p_j - i)$.

The time complexity is $O(np_1)$.

Problem Tutorial: "Matrix Counting"

Let the continuous segment denote an interval in which both the range and position are continuous.

The problem we need to solve is to count the number of permutations that do not have consecutive segments of length $k+1, \dots, n-1$. That is, the length of all consecutive segments, except the permutation itself, does not exceed k.

Assuming that we can replace all the current non-trivial consecutive segments (consecutive segments whose length is not 1 or n) with a number each time, then repeat the operation, and finally get a permutation without non-trivial consecutive segments. For the count of answers, we only care if the first replaced consecutive segment does not exceed k.

Suppose we count the number of permutations $F(x)$ (also called single permutations) without nontrivial consecutive segments, the answer seems to be $(F - x) \circ (\sum_{i=1}^{k} i!x^i)$.

However, this is incorrect. If the final permutation is $[1, 2]$ or $[2, 1]$, then we are not unique in how we replace consecutive segments. It depends on how many consecutive segments the current permutation can be divided into in just order or reverse order. So if this situation is not considered, the answer should be $(F - x - 2x^2) \circ (\sum_{i=1}^{k} i!x^i)$.

Let $P(x) = \sum_{i \geq 1} i!x^i$. Let's first consider the permutations that end up with $[1, 2]$. Consider the OGF $G(x)$ corresponding to each segment it is divided into. We have

$$P = \sum_{i \geq 1} G^i = \frac{G}{1 - G}$$

$G(x)$ cannot get the number of permutations of $[1, 2]$, and the permutations in $P(x) - G(x)$ can be divided by $G(x)$. That is

$$G = \frac{P}{1 + P}$$

We consider how to get the permutation of $[1, 2], [2, 1]$ to satisfy the condition. Note that after division, any segment within an interval can constitute a continuous segment. Then we need to satisfy that the largest proper prefix and proper suffix do not exceed k. That is, the first and last paragraphs are not less than $n - k$. This part is

$$[x^n]2(P + 1) \left(\sum_{i \geq n-k} g_i x^i \right)^2$$

Next, we consider how to calculate a coefficient of F and its composite $\sum_{i=1}^{k} i!x^i$. We split all permutations into two: permutations that would get $[1, 2], [2, 1]$ or not.

The formula for the terms that would get $[1, 2], [2, 1]$, which we have considered, is

$$2(P - G) = 2 \left(P - \frac{P}{1 + P} \right) = \frac{2P^2}{1 + P}$$

and what will not get, is

$$(F - x - 2x^2) \circ P = F(P) - P - 2P^2$$

Then add the special case of length 1 to get

$$F(P) - P - 2P^2 = P - \frac{2P^2}{1 + P} - x$$

$$(F - x - 2x^2) \circ H = H - \frac{2H^2}{1+H} - P^{\langle -1 \rangle}(H)$$

Substituting $H(x) = \sum_{i=1}^{k} i!x^i$, we can solve this problem.

The key is to calculate $P^{\langle -1 \rangle}(H)$. Let $A = P^{\langle -1 \rangle}$, we first need to calculate A. By integer recurrence of P, the ordinary differential equation is

$$P = x(1 + P + xP')$$

Substitute $x = A$, there is

$$x = A(1 + x + AP'(A)) = A\left(1 + x + \frac{A}{A'}\right)$$

$$xA' = (1+x)AA' + A^2$$

Let $B = A(H)$, we have

$$\frac{H}{H'}B' = \frac{1+H}{H'}BB' + B^2$$

Let $C = H/H', D = (1+H)/H', E = B(D-1)$, we have

$$\begin{cases} e_n & = \sum_{k=0}^{n-1} b_k d_{n-k} \\ -(b_n + e_n) & = \sum_{k=0}^{n-2}(k+1)b_{k+1}([k \neq 0](e_{n-k} + b_{n-k}) - c_{n-k}) + \sum_{k=1}^{n-1} b_k b_{n-k} \end{cases}$$

We can calculate everything by using D&C and FFT in $O(n \log^2 n)$, which is able to pass.

The author's solution is using $O(n \log^2 n / \log\log n)$ approach for relaxed multiplication. You may find more details here:

- https://hly1204.blog.uoj.ac/blog/7319
- http://www.texmacs.org/joris/newrelax/newrelax.html

Problem Tutorial: “No!”

We only care about the position where h is the maximum value of the prefix, so after sorting the walls from largest to smallest, let f_i denote that the i wall must be selected, regardless of the answer that the i will fall. Then there is $f_i = \max_{j=1}^{i-1} \min(f_j, \frac{h_j - h_i}{a_j})$. For the contributions of two $j_1 < j_2$, there is at most one intersection point regardless of the upper limit. So we can find a division point p, when $h_i \leq p$, then the decision point j_1 is not bad, when $f_i > p$ decision point j_2 is not bad. And we can find it in $O(1)$, so consider maintaining it in a method similar to CHT.

It can be done $O((n+q)(\log n + \log m))$, which is able to pass. If you use radix sort and gcd tricks, it can be done in $O(n + m + q), m = \max_{i=1}^{n} h_i$, but it is not required.