Tasks from Sasha

Author and problem developer: German Perov

Restatement of the problem

We need to find the number of ways to split the vertices into $k+1$ sets, such that the lowest common ancestor of the i-th set ($1 \le i \le k$) is equal to x_i (the set numbered $k+1$ contains the vertices corresponding to the residents whom Sasha will not invite to solve the problems).

Solution for $k = 1$

Notice that Sasha can invite residents only from the subtree of vertex x_1.

- if Sasha invites the resident from vertex x_1, then any resident from the subtree can be either invited or not invited. There are $2^{sz(x_1)-1}$ such ways
- if Sasha does not invite the resident from vertex x_1, then he must invite at least one resident from at least two subtrees. It is a bit more convenient to compute this value as $cnt_{all} - cnt_1 - cnt_0$, where cnt_{all} — the total number of ways to invite residents from the subtrees (that is, $2^{sz(x_1)-1}$), cnt_1 — the number of ways to invite at least one resident from only one subtree (that is, $\sum\limits_{(x_1,u)\in E} (2^{sz(u)}-1)$), and cnt_0 — the number of ways to invite no residents at all (that is, 1).

Solution for $p_i = i - 1$

- each x_i must belong to the i-th set;
- a vertex v whose task will not be solved can belong to set j ($j \le k$) if and only if x_j is an ancestor of v. To find the number of ways to choose a set for a vertex, it is enough to count how many vertices above it are in the array x. There is also an option to put this vertex into set $k+1$.

It is enough to compute the product of the number of ways to choose a set for v over all vertices.

Solution in $O(nk)$

Let us reformulate the problem so that instead of assigning vertices to sets, we color them. We need the lowest common ancestor of the vertices of color i to be equal to x_i (there will also be a dummy color $k+1$, for which no such condition is required).

We will define $dp[v][c]$ as the number of ways to correctly color the subtree of v, while having c extra colors (which will later be needed to cover vertices from x outside the subtree of v)

- if vertex v is not in x, then simply $dp[v][c] = c \cdot \prod\limits_{(v,u)\in E} dp[u][c]$. This follows from the fact that to obtain c extra colors, it is enough to take the corresponding colorings for the subtrees and choose the color for v itself
- if vertex v is in x, then this vertex must be covered. We will process the children one by one and maintain three values: $cnt[c][0]$, $cnt[c][1]$, and $cnt[c][2]$, where $cnt[c][i]$ is the number of ways to end up with c free colors, while coloring at least one vertex in i child subtrees with the color of vertex v (states with $i \ge 2$ are indistinguishable for us, so we assume that $cnt[c][2]$ stores the total number of ways for all $i \ge 2$). Then adding the next child u is expressed as follows:

- if a free color will be used in u (it is colored with the color of v) and c colors must remain, then $newcnt[c][min(i+1,2)]$ should be increased by $cnt[c][i] \cdot (dp[u][c+1] - dp[u][c])$
- if a free color will not be used in u, then $newcnt[c][i]$ should be increased by $cnt[c][i] \cdot dp[u][c]$

Then $dp[v][c] = (cnt[c][0] + cnt[c][1] + cnt[c][2]) + cnt[c][2]$. The first three terms correspond to the case "v is colored with its own color, the subtrees can be colored arbitrarily". The second term corresponds to the case "v is colored with a free color, and vertices must be colored in at least two subtrees".

The answer to the problem is $dp[root][1]$ (the one means that we have one free color left for $k+1$).

Thus, we obtained a dynamic programming solution computable in $O(nk)$.

Solution in $O(n + k^2 + k^{1.5}\sqrt{n})$

Let us call vertices from x bad, and vertices not in x good.

Suppose that a good vertex v has a good child w.

We obtain the formula

$$dp[v][c] = c \cdot \prod_{(v,u) \in E} dp[u][c] = c \cdot \left(\prod_{w \neq u \ (v,u) \in E} dp[u][c] \right) \cdot dp[w][c]$$

$$dp[v][c] = c \cdot \left(\prod_{w \neq u \ (v,u) \in E} dp[u][c] \right) \cdot c \left(\prod_{(w,t) \in E} dp[t][c] \right)$$

Notice that we can separately group the power of c and the product of dynamic programming states for a fixed c. This corresponds to the fact that if there is an edge (v, w) between two good vertices, then we can remove the lower vertex and attach the subtrees of w to v (while not forgetting that the vertex w itself can also be colored; for this, we propose maintaining a weight at vertices — how many vertices this one corresponds to).

After this processing, there will be no two good vertices connected by an edge. Therefore, in the resulting tree there will be $2k$ vertices plus some number of good leaves (possibly n). At the same time, note that the total weight of the good leaves cannot exceed n.

Thus, we obtained a solution in $O(n + k^2 + leaves \cdot k)$, where $leaves$ is the number of good leaves after the tree processing described above. It remains to learn how to efficiently process a vertex to which a set of good leaves of some weight is attached.

Let us compute the array $cnt[c][i]$ on good leaves. For convenience, let us assume that l leaves with weights w_1, w_2, …, w_l are attached to a vertex. For now, we know how to compute $cnt[c][i]$ for all c in $O(k \cdot l)$ and want to do it faster.

- $cnt[c][0] = c^{\sum w_j}$. This corresponds to simply coloring vertices in each subtree with c free colors.
- $cnt[c][2] = (c+1)^{\sum w_j} - cnt[c][0] - cnt[c][1]$. This follows from the fact that $cnt[c][0]$, $cnt[c][1]$, and $cnt[c][2]$ cover all possible ways to color the children's subtrees while leaving c free colors. The total number of such ways is, in turn, $(c+1)^{\sum w_j}$
- it remains to understand how to compute $cnt[c][1]$. Notice that when adding the next w, we get

$$newcnt[c][1] = cnt[c][0] \cdot ((c+1)^w - c^w) + cnt[c][1] \cdot c^w = newcnt[c][0] \cdot \left(\left(\frac{c+1}{c} \right)^w - 1 \right) + cnt[c][1] \cdot c^w$$

From this it follows that

$$\frac{newcnt[c][1]}{newcnt[c][0]} = \frac{cnt[c][1]}{cnt[c][0]} + \left(\frac{c+1}{c}\right)^w - 1$$

So, we can maintain the value $\frac{cnt[c][1]}{cnt[c][0]}$ and update it when a new weight is added. Thanks to this trick, we can add not one weight, but several equal weights at once.

Having the array $w_1, \ldots, w_l$, let us count how many times each weight appears and feed the algorithm not only the weights, but also their multiplicities. We obtain the computation of $cnt[c][i]$ for all c in $O(k \cdot distinct(w_1, \ldots, w_l)) \le O\left(k \cdot \sqrt{\sum w_j}\right)$ for a fixed vertex.

We obtained the bound $O\left(k \cdot \sqrt{\sum w_j}\right)$ for processing the good leaves of one bad vertex. Now let us estimate the total time for processing all leaves.

Let W_i be the total weight of good leaves attached to vertex x_i. Since the total number of vertices is at most n, we have the constraint $\sum_{i=1}^{k} W_i \le n$, and the running time is $\sum_{i=1}^{k} k\sqrt{W_i}$. The worst case is $W_i = \frac{n}{k}$ for all i, and the total time for processing leaves is $O\left(\sum_{i=1}^{k} k\sqrt{\frac{n}{k}}\right) = O\left(k^{1.5}\sqrt{n}\right)$

Thus, the total running time of the whole algorithm is $O\left(n + k^2 + k^{1.5}\sqrt{n}\right)$