Editorial — Ball Machine

For any leaf l, let P_l denote the path from leaf l to the root, including both endpoints. Let the **depth** of a node v, denoted by $depth(v)$, be the number of nodes on the path from the root to v, also including both endpoints.

Throughout the editorial, a path **length** means the number of nodes on the path. This convention is consistent with `depth`: a root-to-leaf path containing d nodes has length d.

Subtask 1: The root has M children (5 points)

If the root has M children, each child has at least one leaf in its subtree. Since there are exactly M leaves, every child of the root has exactly one leaf in its subtree.

Let u be a child of the root, and let l be the unique leaf in the subtree of u. We claim that the whole subtree of u is just the path P_l. Suppose there were a node v in the subtree of u that was not on P_l. Then l would not belong to the subtree of v. Every node has at least one leaf in its subtree, so v would have another leaf in its subtree. That leaf would also be in the subtree of u, contradicting the fact that u contains only the leaf l.

Therefore every subtree of a child of the root is a single root-to-leaf path. The tree is thus a collection of M paths sharing only the root.

To reconstruct the tree, it is enough to know the length of P_l for every leaf l. For a fixed leaf l, repeatedly insert balls with value 0 into l until l becomes occupied. At that point, every node of P_l is occupied, and no node outside P_l is occupied. Hence the number of successful insertions is exactly $|P_l|$, the length of the path.

After recording this value, clear the machine and repeat for the next leaf. Knowing all M path lengths completely determines the tree.

Depth of the LCA

For any two distinct leaves u and v, we can determine $depth(LCA(u, v))$.

First, determine $|P_u|$ and $|P_v|$ using the procedure above. Then fill the machine by inserting balls into u until u is occupied, followed by inserting balls into v until v is occupied. The number of successful insertions is exactly the number of nodes in the union $P_u \cup P_v$.

The paths P_u and P_v coincide exactly from the root to $LCA(u, v)$. Therefore $P_u \cap P_v$ is precisely the path from the root to $LCA(u, v)$, whose size is $depth(LCA(u, v))$. By inclusion-exclusion,

$$depth(LCA(u, v)) = |P_u \cap P_v| = |P_u| + |P_v| - |P_u \cup P_v|.$$

In other words, record both successful insertion counts:

$$a = |P_u|, \qquad b = |P_v| - depth(LCA(u, v)).$$

The second counts the part of P_v below the already occupied common prefix from the root to $LCA(u, v)$. Once all leaf depths are known, we obtain

$$depth(LCA(u, v)) = |P_v| - b.$$

Remember to clear the machine after processing each pair.

Subtask 2: $M \leq 3$ (10 points)

There are four possible types of underlying tree.

1. **One leaf.**

By the argument from Subtask 1, the whole tree is a single path from the root to the leaf.

2. **Two leaves u, v.**

The tree consists of a path from the root to $LCA(u, v)$, followed by one path from this node to u and one path to v. In other words, it is a three-branch star centered at $LCA(u, v)$.

3. **Three leaves u, v, w.**

There are two possibilities.

- **The tree is a three-branch star.** There is one branching node c, with

$$LCA(u, v) = LCA(u, w) = LCA(v, w) = c.$$

Consequently all three pairwise LCA depths are equal.

- **The tree is not a three-branch star.** There is a first branching node x on the path from the root. One child of x contains a single leaf u, while another child contains the other two leaves v, w. Let $y = LCA(v, w)$. Then

$$x = LCA(u, v) = LCA(u, w), \quad y = LCA(v, w)$$

and y lies strictly below x. Hence

$$depth(LCA(u, v)) = depth(LCA(u, w)) < depth(LCA(v, w))$$

We can distinguish and construct these cases as follows. First, find $|P_l|$ for every leaf l. For every pair of leaves, compute the depth of its LCA using the previous section.

1. **One leaf.** Construct a path of length $|P_l|$ from the root to l.

2. **Two leaves u, v.** Let $c = LCA(u, v)$. We know $depth(c)$, so construct a path of length $depth(c)$ from the root to c. Then attach paths of lengths

$$|P_u| - depth(c), \qquad |P_v| - depth(c)$$

from c to u and v, respectively.

3. **Three leaves u, v, w.**

- If all three pairwise LCA depths are equal, the tree is the three-branch star. Let $c = LCA(u, v)$. Construct a path of length $depth(c)$ from the root to c, then attach three paths of lengths

$$|P_u| - depth(c), \quad |P_v| - depth(c), \quad |P_w| - depth(c).$$

- Otherwise, reorder the leaves so that

$$depth(LCA(u, v)) = depth(LCA(u, w)) < depth(LCA(v, w)).$$

Let $x = LCA(u, v), y = LCA(v, w)$.

Construct:

- a path of length $depth(x)$ from the root to x;
- a path of length $|P_u| - depth(x)$ from x to u;
- a path of length $depth(y) - depth(x)$ from x to y;
- paths of lengths $|P_v| - depth(y)$ and $|P_w| - depth(y)$ from y to v and w.

All internal nodes may receive arbitrary distinct labels, because the statement only requires the reconstructed tree up to a relabeling of non-leaf, non-root nodes.

Subtask 3: $N \leq 200,\ M \leq 45$ (25 points)

There are several ways to solve this subtask. We first describe the approach based on the depths of the LCA of all pairs of leaves. A simpler approach appears later and leads naturally to the full solution.

LCA depth of all pairs

If the tree has only one leaf, the previous construction is sufficient. Otherwise, compute

$$S = \{(depth(LCA(u, v)), u, v) : u \neq v \text{ are leaves}\}.$$

Consider the node x having at least two children with the smallest depth. Such a node is unique: if two distinct nodes had the same minimum depth and both had at least two children, their LCA would have strictly smaller depth and would also have at least two children, a contradiction.

The node x is the LCA of some pair of leaves belonging to two different child subtrees of x. Therefore $depth(x)$ occurs as a pairwise LCA depth. On the other hand, every LCA of two distinct leaves has at least two child subtrees below it, so it cannot have smaller depth than x. Thus

$$depth(x) = \min_{u \neq v} depth(LCA(u, v)).$$

Now consider a child y of x. For any two leaves in the subtree of y, their LCA lies strictly below x, so their LCA depth is greater than $depth(x)$. Conversely, leaves from two different children of x have LCA exactly x.

Therefore, after finding x, partition the leaves into groups corresponding to the child subtrees of x. Two leaves belong to the same group exactly when their LCA depth is greater than $depth(x)$. We create a path of length $depth(x)$ from the root to x, recursively reconstruct each group and attach the resulting subtree to x.

It remains to count the number of `collect` operations. A straightforward implementation uses one clear operation for every leaf to compute the depth and one for every unordered pair of leaves, giving

$$M + \binom{M}{2} \leq 1035.$$

This is just above the allowed limit of 1000. We can save the M separate leaf-depth operations. For $M \geq 3$, orient the pairs so that every leaf occurs as the first endpoint of at least one pair; for example, use the cyclic pairs

$$(0,1), (1,2), \ldots, (M-2, M-1), (M-1, 0).$$

Whenever a leaf is the first endpoint, the corresponding first count gives its full path length. Thus all leaf depths can be obtained while processing the pairs themselves. The total number of clear operations is then at most

$$\binom{M}{2} \leq \binom{45}{2} = 990.$$

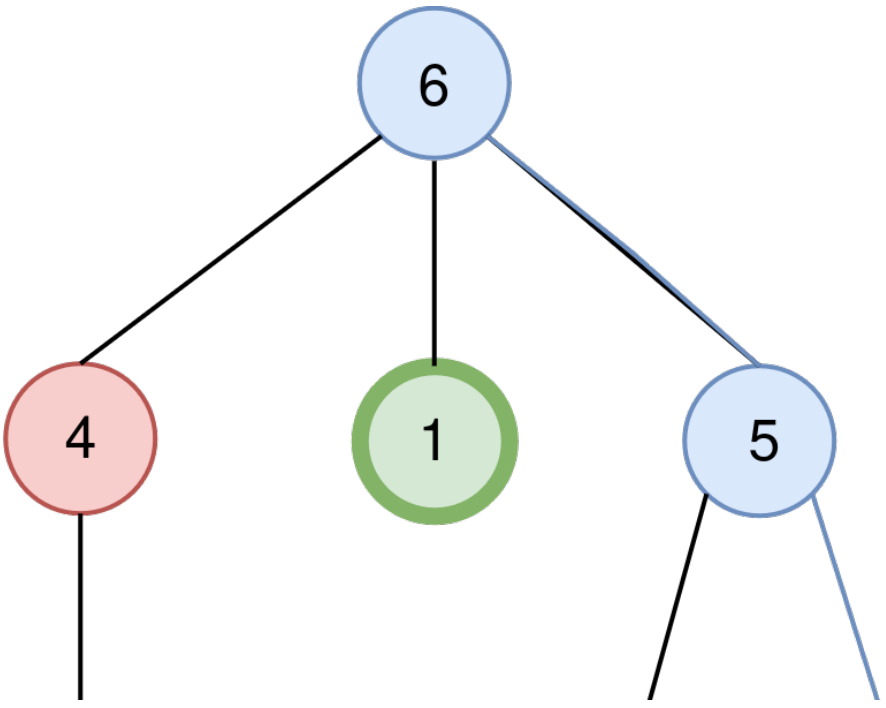
(The case $M \leq 2$ is already covered by the earlier constructions.)

Subtask 4: (60 points)

First approach $C \leq 200$

The following simpler approach to Subtask 3 leads towards the full solution. Observe that, so far, we have never used the values on the balls. We have solely relied on the depths of individual nodes or sets of nodes. In this approach we will utilize distinct ball values to represent distinct leaves. Consider the following decomposition of the tree into paths, starting from the leaves:

For each leaf i, in decreasing order of the leaf's index from $M-1$ to 0, create a path consisting of all ancestors of i (including i itself) that do not already belong to a path. This procedure decomposes the tree into disjoint paths, i.e. each node belongs to exactly one path. Let's denote by $Path_i$ the path corresponding to the leaf i and by Top_i the node at the top of $Path_i$. If we know the length of each path and the parent of the top node of each path, we can reconstruct the tree. The following diagram illustrates the path decomposition described above for the tree in the example.





In order to find the path decomposition of the tree, we use the following procedure. For each leaf i, in decreasing order of the index of the leaf from $M-1$ to 0, insert balls with value i until leaf i is occupied. After filling the whole tree using this procedure, clear the tree. For each path in the path decomposition, let's call the leaf l corresponding to the path P the owner of the path P. Similarly, let's call the path corresponding to leaf l the path of l or path of the leaf.

Observe that this procedure fills each path from the decomposition with the index of the owner of the path. For a node u, let L_u be the owner of the path where u belongs. Since we insert the values equal to the index of the leaf into the path of each leaf, L_u is also the value of the ball occupying node u.

We traversed the leaves in decreasing order of their labels. Consider any node u of the tree. Since the ball values are the leaf indices, each child of u is occupied by a distinct valued ball. Moreover, since node u was filled with a ball through leaf L_u before any other leaf in the subtree of u, leaf L_u must have the largest label among all leaves in the subtree. In other words, for any child y of node u, we have $L_y \leq L_u$. Furthermore, for each node v in the subtree of u, we can infer that $L_v \leq L_u$.

Let $S = [s_1, s_2, \ldots s_N]$ be the sequence returned by `collect()`. Starting from the root as the current node, denoted by u, for each child x of u, we will identify the range in S representing the subtree of x and recursively construct the subtree of x. After constructing the subtrees of all children of u, we can build the subtree of u. Initially, the range representing the subtree of the root is the whole sequence S; hence, we start with $l = 1, r = N$.

`reconstruct`(l, r):

> If $l = r$, the subtree of the current node consists of a single node, i.e. a leaf, and the value on the ball in a leaf is the label of the leaf. Hence, we return a single node with label s_l.
>
> Otherwise, $l < r$ and $s_l = L_u$. Node corresponding to s_{l+1} must be a child, denoted by x, of u. Moreover, since x is visited first, it has the smallest L_x among the children of u. All collected balls in the subtree of x have values no larger than L_x, while the next traversed child of u must have a strictly larger value than x. Hence, we can uniquely identify the position where the traversal of the subtree of x finishes as the smallest $j > l + 1$ with $s_j > s_{l+1}$. Therefore, we can construct the subtrees of the children of u in the following way:
>
> - Let $k_0 = l + 1$. In increasing order of $i = 0, 1, \ldots$ starting with $i = 0$, if possible define k_{i+1} to be the smallest position with $k_i < k_{i+1} \leq r$ such that $s_{k_i} < s_{k_{i+1}}$. The range $[k_i, k_{i+1} - 1]$ corresponds to the subtree of some child of u, while k_{1+1} represents the next child of u in increasing order of the occupied ball value. If no such k_{1+1} exists, then u has a single child. Suppose k_{t-1} is the last position defined; set $k_t = r + 1$. For each $0 \leq j < t$, the range $[k_j, k_{j+1} - 1]$ corresponds to the subtree of one child of u. Thus, recursively call **`reconstruct`**$(k_l, k_{l+1} - 1)$ for each range. Finally, connect the resulting subtree roots to a new node with an arbitrary distinct label for u, and return the constructed tree.

The method above uses balls with values at most $M - 1$ and a single `collect` operation. Hence, there are

at most $M \leq 200$ operations in total. To improve the resource bound, we might want to reuse values across several paths and use additional `collect` operations to distinguish them.

Improved approach $C \leq 72$

First observe that, for any chosen group G of leaves, using the reconstruction above, we can determine the depth of all leaves in that group using one `collect` and $|G|$ values.

To reconstruct the tree, we use a square-root decomposition. Choose positive integers b, d, and define

$$B = \{0, 1, \ldots, b-1\}, \qquad D = \{b, b+1, \ldots, b+d-1\}.$$

Partition the leaves into groups of size at most b, in decreasing order of their labels. For each group G, by invoking **`reconstruct`**$()$ on the current group alone, we can reconstruct the part of the tree covered by the paths from the leaves in the group to the root. Hence, we can determine the depth of every leaf in the group.

For each leaf l, remains to find a leaf r such that the parent of Top_l is on $Path_r$. Knowing the lengths of $Path_l$ and $Path_r$ as well as depths of l and r, we can identify the parent of Top_l within $Path_r$. Let's call r the **linked leaf** of l.

For the first group, all relevant linked leafs are inside the group, so the previous $C \leq 200$ reconstruction can be used directly. This gives the structure of the union of the paths belonging to that group and, in particular, the length of each path.

For remaining groups process them in the order of partitioning. Let the current group be Y, let X be the set of leaves in all previously processed groups. By the order in which the path decomposition was created, the linked leaf of every leaf in Y belongs to $X \cup Y$.

For each $l \in X$, $Path_l$ is represented by a value from D. For each $l' \in Y$, $Path_{l'}$ represented by a value from B. The smaller values therefore identify the paths in the current group Y, while the larger values identify the already processed paths in X.

We still need to tell which exact path in X is the parent of paths in Y. Represent each leaf index in base d. For each possible digit position t in base d:

> Let f_l be the t-th digit of l. For every $l \in X$, insert $f_l + b$ into l until l is occupied. For the current group Y, assign values $b-1, b-2, \ldots, 1, 0$ to leaves in Y in decreasing leaf order, and insert the assigned value into each leaf until it is occupied. Finally, perform a collect operation and let $S = [s_1, s_2, \ldots s_k]$ be the collected sequence.
>
> Consider traversal reaches a node u belonging to a path with owner in X, $L_u \in D$. Any child x that starts a path with owner in Y has a value in B. Hence, entire subtree of x is traversed before the next child of u. Consequently, maximal ranges $[l, r]$ of S with values below b describe the subtrees of nodes that are the top nodes of paths with owners in Y.
>
> Find every range $[l_i, r_i]$ such that:
>
> - $s_j < b$ for every $j \in [l_i, r_i]$;
> - either $l_i = 1$ or $s_{l_i - 1} \geq b$;
> - either $r_i = k$ or $s_{r_i + 1} \geq b$.

Actually, the first collected value always comes from path of a leaf in X, so $s_1 \geq b$; therefore every such range has $l_i > 1$. The part of the tree corresponding to the range $[l_i, r_i]$ can be reconstructed using `reconstruct`$(l_i - 1, r_i)$:

Since the range $[l_i, r_i]$ can cover multiple subtrees, $l_i - 1$ is used to linked them together. Moreover, $l_i - 1$ represents the linked leafs in X.

From this reconstruction, for each leaf $l \in Y$ we learn the value x inserted to the linked leaf of l.

- $x \in B$: we learn the exact leaf within the current group.
- $x \in D$: we learn t-th digit in base d representation of the linked leaf of l.

By repeating the procedure for each digit in base d, we learn the linked leaf of every leaf in Y and proceed to the next group.

There are $\lceil \frac{M}{b} \rceil$ groups. The first group needs one `collect`, and each later group needs $\lceil \log_d M \rceil$ digit rounds. Thus a simple upper bound on the number of collects is $\lceil \frac{M}{b} \rceil (\lceil \log_d M \rceil + 1) + 1$, where the extra 1 comes from the initial `collect` for the length of each path.

The largest inserted value is at most $b + d - 1$, so

$$C \leq \left\lceil \frac{M}{b} \right\rceil (\lceil \log_d M \rceil + 1) + b + d.$$

Taking $b = d = 15$ and gives a bound $C \leq 72$. Note that the constants can be tuned further.

Towards the full score $C \leq 44$

The main weakness of the previous construction is that every path receives one dedicated value. We can do better by using **two values per path**, which allows us to process all paths of length at least 2 in a single collect. Paths of length 1 will be handled separately.

Let T be the set of leaves whose corresponding decomposition path has length at least 2. Choose a positive integer b. Assign every $l \in T$ ordered pairs (x_l, y_l) with $0 \leq x_l, y_l < b$ such that the ordered pairs are distinct. Only the pair as whole needs to be distinct; the individual coordinates do not.

For each $l \in T$, in decreasing order of l:

1. Insert x_l once into l.
2. Insert b exactly $|Path_l| - 2$ times into l.
3. Insert $y_l + b + 1$ once into l.

The resulting occupied path (from top to bottom) has values

$$x_l, \underbrace{b, b, \ldots, b}_{|Path_l| - 2 \text{ times}}, y_l + b + 1.$$

Perform a single `collect`.

Starting from the root, there are three kinds of values:

- $s_i < b$: this is the top node of a new path;

- $s_i = b$: this continues the current path;
- $s_i > b$: this is the leaf at the end of the current path.

The traversal order lets us maintain the current path on a stack. When a value below b is encountered, a new path is started from the current node. Values equal to b extend that path. A value above b finishes the path, so we pop it and return to the previous path.

Thus one collect reconstructs every path in T, including where top of each path is attached to its parent. Since every leaf in T has at least two nodes in its path, the two endpoint values x_l and $y_l + b + 1$ are enough to identify the path uniquely.

It remains to handle paths of length 1. Let O be the set of leaves whose path has length 1, and partition O into groups of size at most b'. For one group G, first perform the same construction as above for all paths of length at least 2, but shift the inserted values by adding b'. To each $l \in G$ in decreasing order of l, assign values $b' - 1, b' - 2, \ldots, 1, 0$ to z_l and insert z_l once into l.

The value ranges now have four meanings:

- $s_i < b'$: a length-one path, i.e. a leaf in G;
- $b' \le s_i < b' + b$: the top of a path of length at least 2;
- $s_i = b' + b$: an internal continuation of such a path;
- $s_i > b' + b$: the leaf at the end of such a path.

The only additional information needed for leafs in G is their linked leafs and where the parent of the leaf is. While scanning the collect sequence, we keep the current path and its exact position. When a value $z_l < b'$ is encountered, the leaf l is linked to that recorded position.

After processing all groups of O, every path parent is known, so the complete tree can be reconstructed.

For the resource bound, the maximum inserted value is $b' + 2b$.

There is one collect to determine all path lengths, one collect for all paths of length at least 2, and at most $\left\lceil \frac{|O|}{b'} \right\rceil$ collects for the length-one groups. Therefore $K \le 2 + \left\lceil \frac{|O|}{b'} \right\rceil$.

$$C \le K + B \le 2 + \left\lceil \frac{|O|}{b'} \right\rceil + b' + 2b$$

Choose $b = \lceil \sqrt{|T|} \rceil, b' = \lceil \sqrt{|O|} \rceil$. Then $\left\lceil \frac{|O|}{b'} \right\rceil \le b'$, hence,

$$C = K + B \le 2 + b' + b' + 2b = 2(b + b' + 1).$$

Finally, since $|T| + |O| = M \le 200$, the maximum possible value of $\lceil \sqrt{|T|} \rceil + \lceil \sqrt{|O|} \rceil$ is 21, by $|T| = 82$ and $|O| = 118$. Hence

$$C \le 2(21 + 1) = 44.$$

This completes the required full-score construction.