Kick Start 2020 - Round D

Analysis: Locked Doors

Test Set 1

We can say that the rooms which Bangles has covered in the journey will always form a contiguous subarray. This is because, to go from room i to room j, Bangles must have unlocked the doors that occur between room i and room j once, only then Bangles would be able to move to room j. At any point of time, if Bangles has visited rooms from l to r, Bangles need to make a decision whether to go from room r to r+1 or from room l to l-1 depending which door has lower difficulty. Accordingly, Bangles can update l and r. For deciding which room to go next, it would take constant time. Thus, for taking **K** pictures, it would take O(**K**) time. **K** can be at most **N**. For each query, it would take O(**N**) time to process the query. Thus, overall complexity of the solution would be O(**Q** * **N**).

Test Set 2

The naive solution would time out for the constraints of test set 2. Thus, we can use the following approach to solve this:

Let InterestingLeft[i] be the door which is the closest door to the left of door i and has difficulty higher than door i. In case there is no such door, assign -1 to it. Similarly, let InterestingRight[i] be the door which is the closest door to the right of door i and has difficulty higher than door i. In case there is no such door, assign -1 to it. InterestingLeft[i] and InterestingRight[i] can be calculated by using the stack method in O(**N**) time.

After we are done unlocking the door i and all doors with difficulty lower than **D**[i] between doors InterestingLeft[i] and InterestingRight[i], we would either unlock door InterestingLeft[i] or door InterestingRight[i] next depending on which one has lower difficulty. If we have have only one choice, then we unlock that door itself. For generality, let us assume the next door that would be unlocked would be j. j would be equal to either InterestingLeft[i] or InterestingRight[i].

Let us construct a tree using the given relations. For each door i, we find such j and assign j as parent of i. We can see that a node can have at most 2 children: at most one from the left side and at most one from the right side. Thus the relations would form a Cartesian tree. From the given tree, we can make the following observations:

- If a node has only one child:
  - If the starting node is inside the subtree of current node, then the node will be visited only after all the nodes in its subtree have been visited.
  - If the starting node is outside the subtree of current node, then first the node will be visited and after that the nodes in its subtree will be visited.
- If a node has two children:
  - If the starting node is in left subtree, then first all the nodes in subtree of left child will be visited, then the current node will be visited, and finally the subtree of right child will be visited.
  - Similarly, if the starting node is in right subtree, then first all the nodes in subtree of right child will be visited, then the current node will be visited, and finally the subtree of left child will be visited.

Building this tree can be done in O(**N**) time because assigning the parents for each node according to InterestingLeft[i] and InterestingRight[i] takes constant time.

Let the subtree size of node i be Size[i]. Consider a query with starting room **S** and we need to find **K**-th room we will be in. Let the starting door X be the first door that we unlock. This can be determined in constant time by comparing the doors adjacent to room **S**. In the constructed tree, if size[X] < **K**, the whole subtree will be visited and we will move to its parent node. This will continue until we reach the node u such that size[u] ≥ **K**, so that we know that our answer lies within this subtree. To find such node, we can use the binary lifting approach similar to finding the lowest common ancestor between two nodes in a tree. This can be precomputed in O(**N** * log(**N**)) time. In a query we have following cases:

- If size[X] ≥ **K**, then:
  - If X is the door left to **S**, then answer is **S** - **K**.
  - Otherwise, answer is **S**+**K.**
- Otherwise, find node Y which is the first node on path from X to root such that size[Y] ≥ **K**. This can be done using binary lifting in O(log**N**). Now we can find the answer in constant time as follows:
  - If X < Y, then answer is Y + **K** - size[leftChild[Y]]
  - Otherwise, answer is Y + 1 - (**K** - size[rightChild[Y]])

Each query takes O(log**N**) time and building the binary tree and preprocessing for binary lifting takes O(**N** log**N**) time. Hence, overall complexity of the solution is O(**N** log**N** + **Q** log**N**).