T172 City Reform

Problem by: Alex Tung

Data by: Charlie Li

Problem Statement

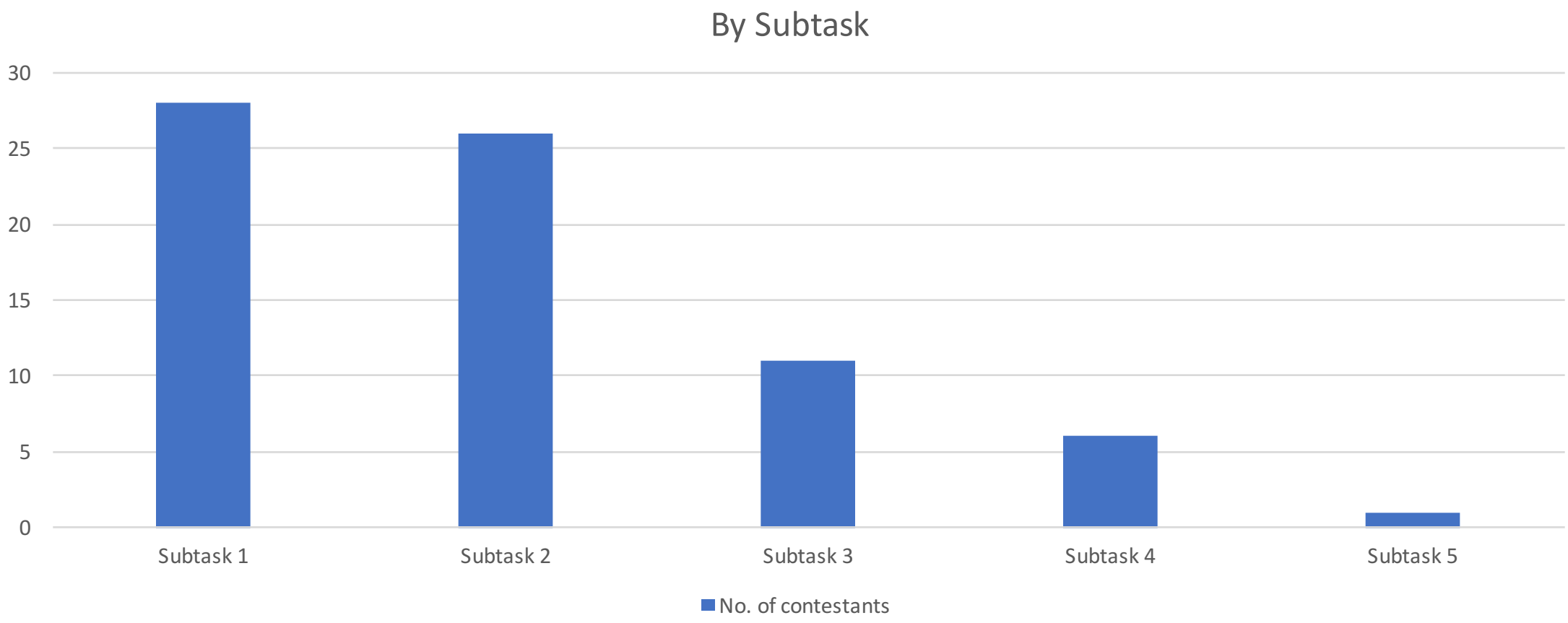
- Given a tree with N nodes. Every edge has a weight l_i.
- Need to answer Q queries, each query will connect two different nodes to form a circle, you need to find the farthest distance from any nodes to the cycle.

Subtask

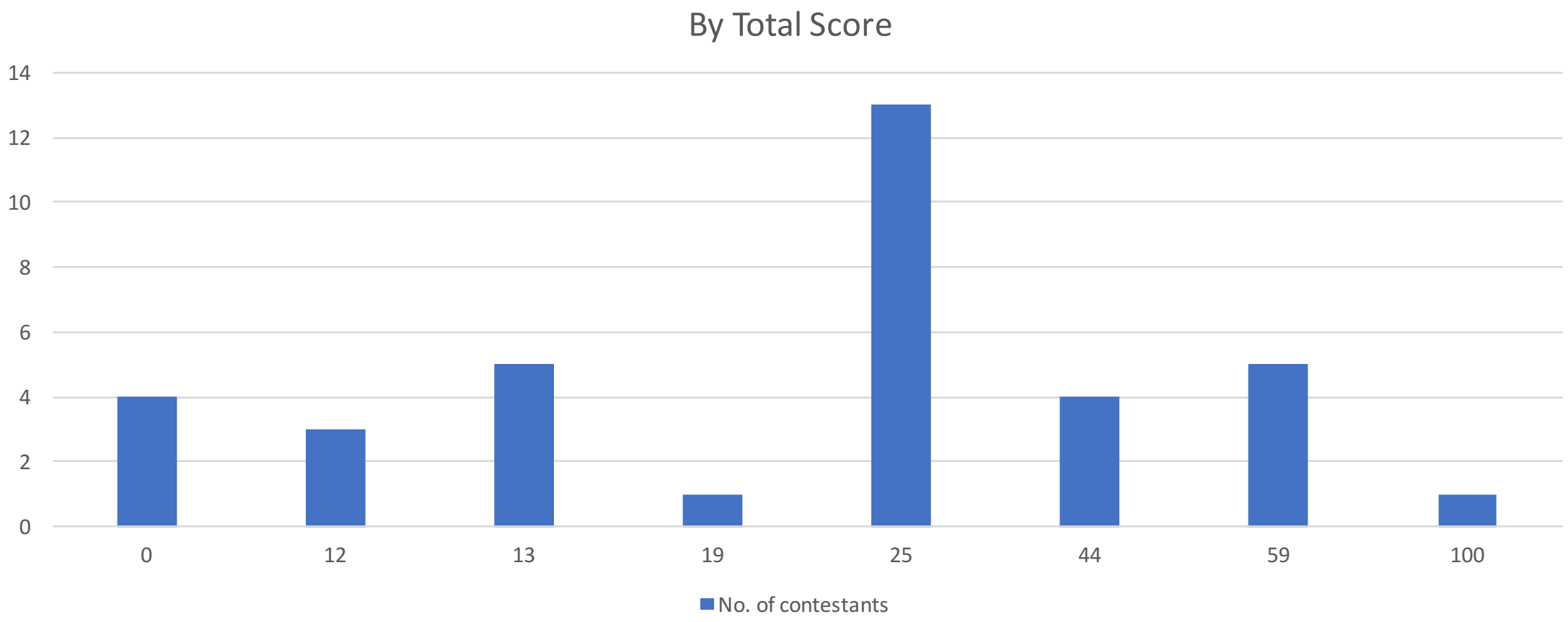
$2 <= N <= 200000,\ 1 <= Q <= 100000$

- Subtask 1 (13 points): The i^{th} road connects city i and city i + 1
- Subtask 2 (12 points): The i^{th} road connects city 1 and city i + 1
- Subtask 3 (19 points): $Q <= 50, N <= 50$
- Subtask 4 (15 points): $Q <= 5000, N <= 5000$
- Subtask 5 (41 points): No additional constraint

Score Distribution



Score Distribution



Subtask 1

- Subtask 1 (13 points): The i^{th} road connects city i and city i + 1
- The given tree is a chain.
- We only need to store the distance form city 1 and city N to every cities.
- Let L[i] be the distance from city 1 to city i
- Let R[i] be the distance from city N to city i
- Then the answer for connecting city u and city v is

$$\max(\min(L[u], L[v]), \min(R[u], R[v]))$$

or

$$\max(L[\min(u, v)], R[\max(u, v)])$$

Subtask 1

- How to compute L[i] and R[i] efficiently?
- We know that l[i] is the length of the road from city i to city i+1
- We can compute L[i] and R[i] using two for loops
- Base case: L[1] = 0, R[N] = 0
- Then L[i] = L[i-1] + l[i-1], R[i] = R[i+1] + l[i]

- Then $\min(L[u], L[v])$ or $L[\min(u, v)]$ is the distance from city 1 to the left city
- Also $\min(R[u], R[v])$ or $R[\max(u, v)]$ is the distance from city N to the right city

Subtask 2

- Subtask 2 (12 points): The i^{th} road connects city 1 and city i + 1
- The given tree is a star.
- We only need to know the largest three distance from city 1 and the respective city.

- Let m1, m2 and m3 be the largest three distance.
- Let c1, c2 and c3 be the respective cities.

Subtask 2

- Then for any query (u, v)
  - If (u != m1) and (v != m1)
    - Answer = m1
  - Else if (u != m2) and (v != m2)
    - Answer = m2
  - Otherwise
    - Answer = m3

Subtask 3

- Subtask 3 (19 points): Q <= 50, N <= 50

- We can just follow what the problem ask as to do.
- First, find the cycle, then find the farthest distance from the cycle
- So for every query (u, v), just add an edge from u to v
- To check whether a city is inside the cycle, we can do a DFS starting from that city and see whether it can return to the original city.
  - Be careful, need to special handle the case that u and v are already connected by a road before adding the new road

Subtask 3

- So now we can check whether a city is inside the cycle in O(N)
- To Check all cities, we need $O(N^2)$
- After finding the cycle, then we can start a DFS on every cities inside the cycle to find the farthest distance.
  - This part is only O(N) but not $O(N^2)$ since we will only visit every node once.
- And after finding the farthest distance, remember to delete the new added edge from the original graph.
- So our algorithm have an overall time complexity of $O(QN^2)$ (or $O(QN^3)$ if you use adjacency matrix instead of adjacency list)

Subtask 4

- Subtask 4 (15 points): Q <= 5000, N <= 5000

- Base on the solution of subtask 3, we need to use adjacency list.
- And, we don’t need to check whether every cities is inside the cycle.
- We only need to check once.

Subtask 4

- When doing the first dfs, we should mark all visited nodes.
- Whenever an edge is connecting a visited node X, we can immediately terminate the DFS and mark the nodes on the path from X to current node as inside the cycle. We can do this recursively and stop until we get back to X.
- Then we get the cycle in O(N).

- So the overall complexity become O(QN) now.

Subtask 5

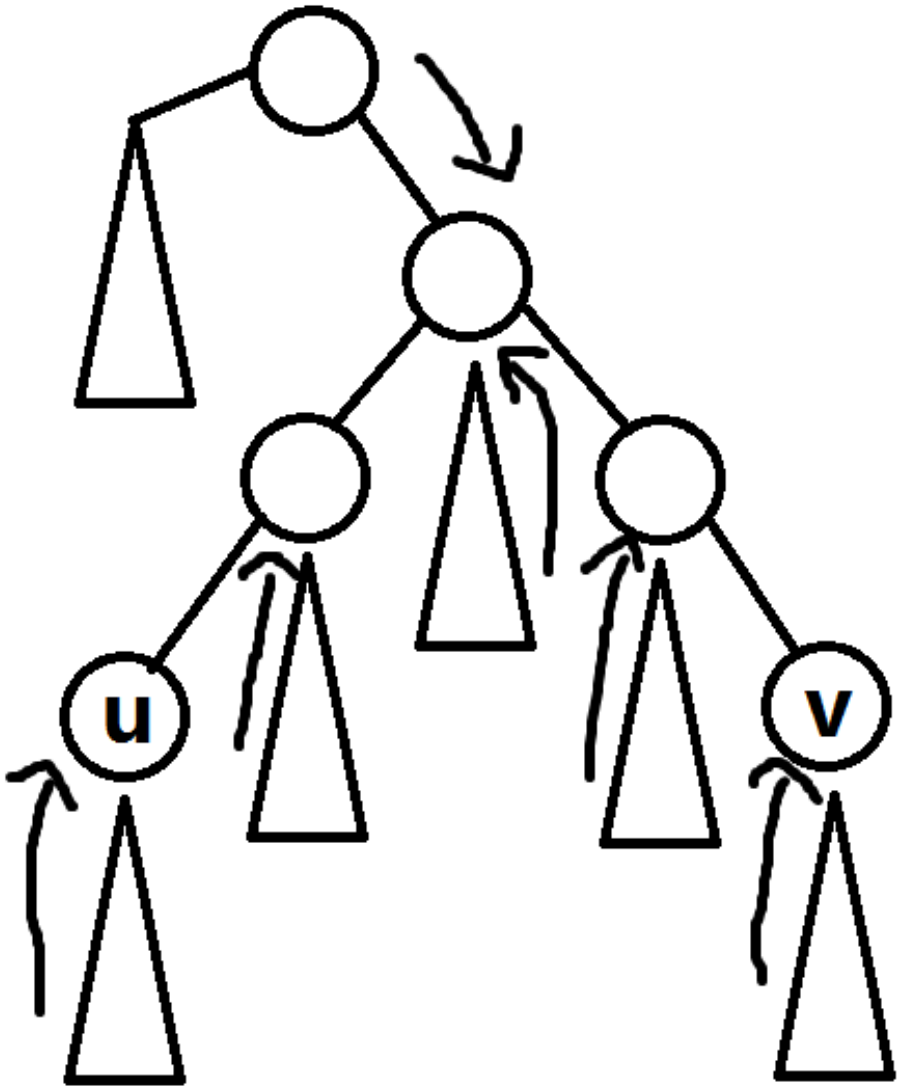
- Subtask 5 (41 points): No additional constraint

- For every query, consider the cycle, if we omit the new added edge, it is just a chain.
- If we find the LCA of u and v, we can further break the chain into two chains, one from u to LCA, another from v to LCA.

- For every node x, let y be x's father, define d[x] to be the farthest from y's subtree to y without passing through x. If y have only one subtree then d[x] = 0.

Subtask 5

- Then our answer will be finding the max of
  - d[x] along those two chains (except the LCA and its children)
  - farthest distance from u's subtrees
  - farthest distance from v's subtrees
  - farthest distance from LCA's subtree excluding the subtrees containing the chains
  - farthest distance from LCA's parent.



Subtask 5

- Therefore we need to precompute the max, second max and third max distance from three different subtrees of a node.
- Also, we need the farthest distance from parent.
- All of these can be done using an O(N) dfs.
- Also, we can use two sparse tables (please refer to Data Structure (IV)) to store the ancestors and maximum along the chains, this require an O(N log N) time to precompute and O(N log N) memory.
- Then for every query, we can get the ancestor and the maximum along the chain in O(log N).
- So now our algorithm is O(N log N + Q log N).