Coding Competitions Farewell Rounds - Round D

Analysis: Indispensable Overpass

Test Set 1

A brute-force solution would be to calculate and sum the distances between every pair of stations, then divide by the total number of pairs. The total number of stations is $\mathbf{W} + \mathbf{E}$, which means that there are $N = \frac{(\mathbf{W}+\mathbf{E}) \times (\mathbf{W}+\mathbf{E}-1)}{2}$ total pairs of stations. Calculating the distance between any given pair takes $O(\mathbf{W} + \mathbf{E})$ time, so this solution has a time complexity of $O((\mathbf{W} + \mathbf{E})^3)$, which is too slow.

First notice that the western stations make a tree, because there is exactly one path between any two stations. This also applies to the eastern stations. Let us define the western tree as T_W, the eastern tree as T_E, and the tree created by connecting the two trees as T_{W+E}.

Instead of calculating the distances between every pair of stations, we can instead compute how many paths go through each edge, which we call the "contribution" of each edge, in T_{W+E}. The total sum of all distances between pairs of nodes is equal to the total sum of edge contributions. We just have to sum up these contributions and divide by N to get the average distance for the given case.

To compute an edge's contribution, we need two depth-first-search (DFS) traversals. First, choose an arbitrary root r in T_{W+E}. For the first traversal, for each node n starting from r, compute and store the size of the subtree S_n under n. This can be calculated recursively by $S_n = 1 + \sum_i S_i$, where the sum is over all children i of n. Fig.1 has an example shown below.

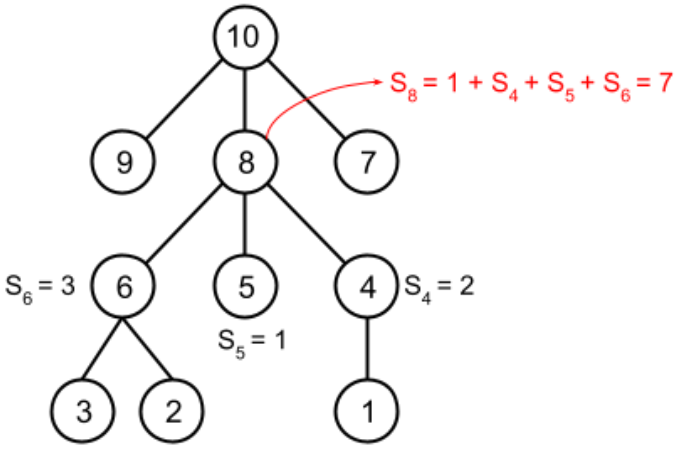

Fig.1 - The size of a subtree under a node is the sum of subtrees under each child of the node, plus the node itself.

Then, traverse the tree a second time to compute each edge's contribution. The key insight here is that for an edge between nodes n and p, where n is the child of p, the edge's contribution is equal to $S_n \times (\mathbf{W} + \mathbf{E} - S_n)$. Thus, the second traversal goes through all the edges in T_{W+E} and calculates their contributions using the stored subtree sizes from the first traversal. This is demonstrated below in Fig.2.

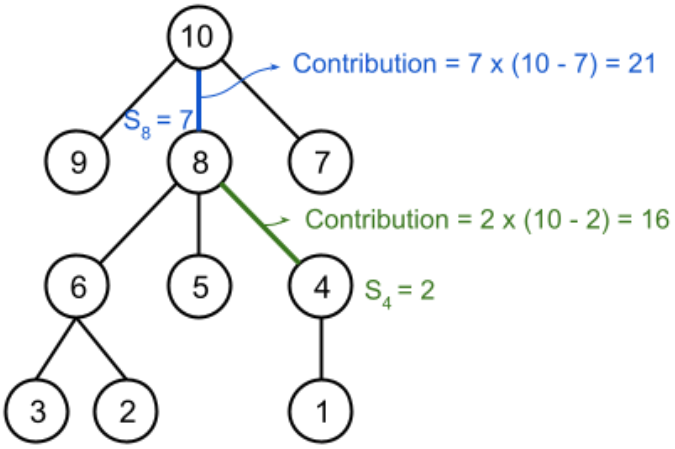

Fig.2 - The edge contribution, which is the number of node pairs that cross that edge, can be calculated as the number of nodes on one side of the edge times the number of nodes on the other side of the edge.

Afterwards, summing up the edge contributions and dividing by N yields the answer. Since the two traversals take $O(\mathbf{W}+\mathbf{E})$ each, and we need to redo the traversals for each of the $\mathbf{C}$ cases, the total time complexity is $O((\mathbf{W}+\mathbf{E})\times\mathbf{C})$. This is sufficient for Test Set 1, but not for Test Set 2.

Test Set 2

The key insight for Test Set 2 is that the total path sum of the N pairs can be computed from the formula $\mathbf{E}\times P^W_{\mathbf{A_k}}+\mathbf{W}\times P^E_{\mathbf{B_k}}+\mathbf{W}\times\mathbf{E}+Q^W+Q^E$, where P^W_n is the path sum from node n to all other nodes in T_W, Q^W is the total path sum within T_W, and likewise for P^E_n and Q^E. If these values are precomputed already, then whenever we are given a connection $(\mathbf{A_k},\mathbf{B_k})$, we can return the average distance in a constant time operation using the formula above. Before deriving this formula, let us first find out how to precompute Q^W, Q^E, and P^W_n, P^E_n for each node n efficiently first. This can be accomplished with two DFS traversals for both T_W and T_E individually.

Let us look at T_W for convenience. For the first DFS, starting from an arbitrary root r, calculate at each node n:

1. The size of the subtree rooted by n, defined as S^W_n. This can be calculated recursively by $S^W_n = 1+\sum_i S^W_i$, where the sum is over all children i of n.
2. The path sum from n to all its descendants, defined as D^W_n. This can be computed recursively by $D^W_n = \sum_i D^W_i + (S_n - 1)$, where the sum is over all children i of n.

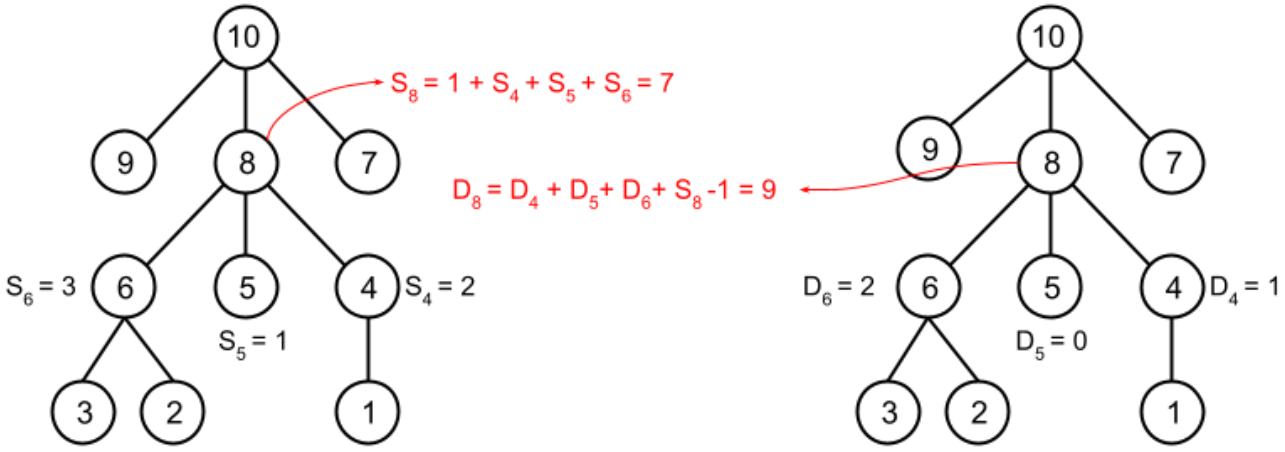

Fig.3 - The path sum from a node to its descendants can be calculated as the sum of the path sums of the node's children to their descendants, plus the contribution of the edges between the node and its children, which is the total number of descendants the node has.

This first traversal calculates the number of subtrees and subtree path sums below n. However, D^W_n is not the full sum of all paths from n to other nodes in T_W because we only counted paths travelling downwards from n. For

every node, we also need to compute path sums that go above and through its parent. This is done via a second DFS. In this traversal, given a node n and its parent p, the actual path sum P_n^W is computed as follows:

$$P_n^W = D_n^W + (\mathbf{W} - S_n^W) + (P_p^W - (D_n^W + S_n^W))$$

Essentially, we add onto D_n^W two values:

1. $(\mathbf{W} - S_n^W)$, the contribution of the edge connecting n to p
2. $P_p^W - (D_n^W + S_n^W)$, the path sums starting from p that do not go through n.

For the root r of T_W, $P_r^W = D_r^W$, as shown below in Fig.4.

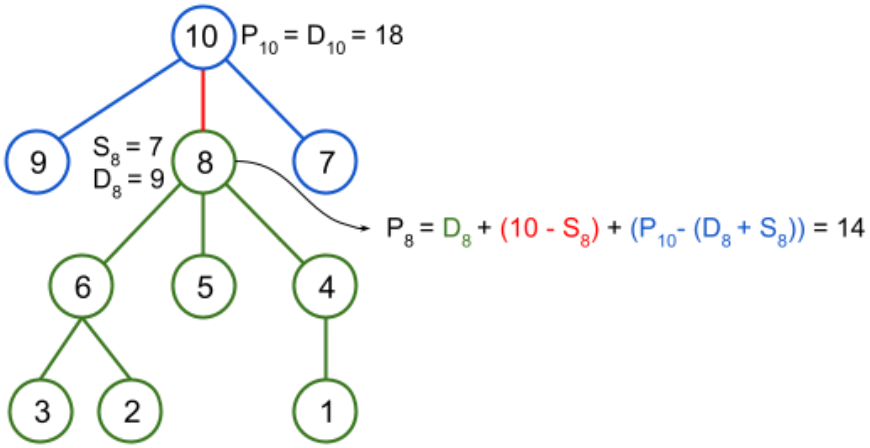

Fig.4 - The path sum from a node to all other nodes can be calculated as the path sum from the node to all its descendants (e.g. path sum from 8 to all green nodes), plus the edge contribution between the node and its parent (e.g. the number of blue nodes), plus the path sum from the node's parent to all other nodes (e.g. path sum from 10 to all blue nodes).

With P_n^W computed for all nodes n in T_W and again P_n^E computed for all nodes in T_E, we can calculate the average distance for each case in constant time. Now let us derive the formula mentioned earlier. Suppose we are given a connection between $\mathbf{A_k}$ from the west, and $\mathbf{B_k}$from the east. We first note that the total path sum can be computed as:

$$\sum_{i \in T_W} \sum_{j \in T_E} d_{i,j} + Q^W + Q^E$$

where $d_{i,j}$ is the distance between nodes i and j. Here, Q^W is the sum of all paths in T_W, and Q^E is the sum of all paths in T_E. We can already calculate them just by summing the P_ns on each side.

$$Q^W = \frac{\sum_{n \in T_W} P_n^W}{2}, Q^E = \frac{\sum_{n \in T_E} P_n^E}{2}$$

The first sum about $d_{i,j}$ is trickier, but we can separate it into three terms that incorporate $\mathbf{A_k}$ and $\mathbf{B_k}$:

$$\sum_{i \in T_W} \sum_{j \in T_E} (d_{i,\mathbf{A_k}} + d_{j,\mathbf{B_k}} + 1).$$

All the terms now allow us to simplify the double summation to single summations. The above equation is equivalent to:

$$\mathbf{E} \times \sum_{i \in T_W} d_{i,\mathbf{A_k}} + \mathbf{W} \times \sum_{j \in T_E} d_{j,\mathbf{B_k}} + \mathbf{W} \times \mathbf{E}.$$

Because we precomputed all the path sums, $\sum_{i \in T_W} d_{i,\mathbf{A_k}}$ is equal to $P_{\mathbf{A_k}}^W$, and $\sum_{j \in T_E} d_{j,\mathbf{B_k}}$ is equal to $P_{\mathbf{B_k}}^E$. Thus, the total path distances between the two trees is

$$\mathbf{E} \times P_{\mathbf{A_k}}^W + \mathbf{W} \times P_{\mathbf{B_k}}^E + \mathbf{W} \times \mathbf{E} + Q^W + Q^E.$$

Divide that by N to get the average distance for each query. Each traversal for T_W and T_E is $O(\mathbf{W})$ and $O(\mathbf{E})$ respectively, which makes the overall complexity $O(\mathbf{W} + \mathbf{E} + \mathbf{C})$ for $\mathbf{C}$ cases.