For each node, we’ll store 3 values –

1. The number of TNS ending at that node 🡪 tns[i]
2. The number of TNS passing through that node 🡪 pass[i]. This means all the TNS that don’t end at that node itself.
3. The number of PNS starting from that node 🡪 pns[i]

PNS 🡪 Pseudo Number Sequence. These sequences don’t necessarily have to start at 1. For example, 4 – 5 – 6 – 7 is a PNS.

Initially, the answer is equal to sum of tns[i].

When we make an update on any value, say on node A by changing it from x to y:

1. First, we will have to subtract pass[A] from sum.
2. Next, we have to add all the new TNS formed by changing this value to y. To do this, we find out all the nodes adjacent to A that have value ‘y + 1’, and let’s call the sum of their PNS count (pns[i]) as “first”. Next, we find all the nodes adjacent to A that have the value ‘y – 1’, and let’s call the sum of their TNS count (tns[i]) as “second”. The sum will be added by first \* second.

(First query deletes the TNS that can no longer exist after this update. Second query adds all the new TNS that are formed after this update).

So, for each query, the TNS count will be = original TNS count – pass[A] + (first \* second + second).

(The extra “second” is added for all the new TNS ending at node A itself after the update).

We can precompute first and second values for all nodes and store them in a map.

Time Complexity : O(N) for precomputation, O(1) per query afterward.