Problem Name: Rotating Tree

Problem Statement:

You are given a binary tree with N nodes and N-1 edges rooted at node 1, but this tree has a unique property of cyclically rotating its nodes at every level after each second. For ex if nodes at level 3 are [4, 5, 6, 7] then after 1 second they will be [7, 4, 5,6]. You need to find sum of middle elements of all the levels of the tree after K seconds. If there are X nodes at some level then middle node would be $((X/2)+1)^{th}$ node.
Initially the nodes at each level are in ascending order of node value.
Note: Here floor division is used for finding middle element.

Brute Force:

The brute force approach would be to use bfs traversal to find nodes at each level of the tree and for each level will right rotate the level K times and find the middle element of that level. We can repeat the same procedure for every level of the tree and add the middle element of all the levels.

Time Complexity:

BFS traversal: O(n)

Each call for right rotating takes $O(n\log n)+O(K\cdot n)$ and there is one call per level. Since it's a binary tree, there are $\log n$ levels in the worst case.

Thus, the overall time complexity is $O(n \log n + K\cdot n)$

Space Complexity:

Space complexity would be O(N).

Implementation:

The code reads the number of nodes `n` and constructs an adjacency list `adj` to represent the tree by reading `n-1` edges. It then reads the value `K` and performs a breadth-first search (BFS) starting from the root node (1), using a queue to manage the traversal. As it traverses the tree, it collects nodes at each level in a vector `temp`. When it encounters nodes at a new level, it calculates a middle

value using the `temp` vector and `K` through the `calculate` function by right rotating the array K times, adds this value to the cumulative answer `ans`, clears the `temp` vector, and continues. After the BFS completes, it makes a final calculation with the remaining nodes in `temp` and outputs the total answer `ans`.

Code:

```
#include <bits/stdc++.h>

using namespace std;
void leftrotate(vector<long int>& v, int d)
{
   reverse(v.begin(), v.begin() + d);
   reverse(v.begin() + d, v.end());
   reverse(v.begin(), v.end());
}
void rightrotate(vector<long int>& v, int d)
{
   leftrotate(v, v.size() - d);
}

long int calculate(vector<long int>& temp, long int K){
   sort(temp.begin(),temp.end());
   int middle= temp.size()/2 + 1;
   int size = temp.size();
   while (K--) {
      rightrotate(temp, 1);
   }
   return temp[middle - 1];
}
int main()
{
   long int n,u,v,ans=0,i,K;
   cin>>n;
   vector<long int>adj[n+1];
   for(i=0;i<n-1;i++){
      cin>>u>>v;
      adj[u].push_back(v);
   }
   cin>>K;
   queue<vector<long int>>q;
   q.push({1, 0});
   int cur=0;
   vector<long int>temp;
   while(!q.empty()){
      long int node = q.front()[0];
      long int level = q.front()[1];
      q.pop();
      if(level==cur)temp.push_back(node);
      else{
         ans+= calculate(temp, K);
         temp.clear();
         cur=level;
         temp.push_back(node);
      }
      for(auto it:adj[node]){
         q.push({it, level+1});
      }

   }
   ans+= calculate(temp, K);
   cout<<ans<<endl;

   return 0;
}
```

Optimized Solution:

Optimized approach makes use of bfs traversal to traverse and store every level of the tree and finds the

middle element of the level in constant time without the need to right rotating the values K times. If you observe carefully then you will find that a level with X nodes will return back to its original position after every X rotation. So, we would require only K%X rotations. Now instead of performing these rotations we can efficiently find the middle element in constant time as if K>= index of middle element then, the index of middle element after K rotations would be (Number of nodes at that level – K) + (index of middle element -1) otherwise the index would be (index of middle element – K – 1).

Time Complexity:

BFS traversal: O(n)

To find middle element, O(mlog m)), and there is one call per level. Since it's a binary tree, there are log n levels in the worst case.

However, since the nodes at each level are processed separately, in the worst case scenario, the time complexity can be approximated as O(n log n).

Space Complexity:

Space complexity would be O(N).

Implementation:

The code reads the number of nodes `n` and constructs an adjacency list `adj` to represent the tree by reading `n-1` edges. It then reads the value `K` and performs a breadth-first search (BFS) starting from the root node (1), using a queue to manage the traversal. As it traverses the tree, it collects nodes at each level in a vector `temp`. When it encounters nodes at a new level, it calculates a middle value using the `temp` vector and `K` through the `calculate` function in an effective manner without actually rotating the array, adds this value to the cumulative answer `ans`, clears the `temp` vector, and continues. After the BFS completes, it makes a final calculation with the remaining nodes in `temp` and outputs the total answer `ans`.

Code:

```
#include <bits/stdc++.h>

using namespace std;
long int calculate(vector<long int>& temp, long int K){
   sort(temp.begin(),temp.end());
   int middle= temp.size()/2 + 1;
   int size = temp.size();
   K%=size;
   int index;
   if (K >= middle)
     index = (size - K) + (middle - 1);
   else
     index = (middle - K - 1);
   return temp[index];
```

```
}
int main()
{
    long int n,u,v,ans=0,i,K;
    cin>>n;
    vector<long int>adj[n+1];
    for(i=0;i<n-1;i++){
        cin>>u>>v;
        adj[u].push_back(v);
    }
    cin>>K;
    queue<vector<long int>>q;
    q.push({1, 0});
    int cur=0;
    vector<long int>temp;
    while(!q.empty()){
        long int node = q.front()[0];
        long int level = q.front()[1];
        q.pop();
        if(level==cur)temp.push_back(node);
        else{
            ans+= calculate(temp, K);
            temp.clear();
            cur=level;
            temp.push_back(node);
        }
        for(auto it:adj[node]){
            q.push({it, level+1});
        }

    }
    ans+= calculate(temp, K);
    cout<<ans<<endl;

    return 0;
}
```