**Problem - 3**

1. **Aim of the Experiment :**

You are given a tree with N vertices (numbered 1 through N) and a sequence of integers A1​,A2​,…,AN​. You may choose an arbitrary permutation p1​,p2​,…,pN​ of the integers 1 through N. For each valid i, you should assign the value Api​​to vertex i.

1. **Objective of the Experiment :**

To choose a vertex of the tree and K−1 times, we have to choose an arbitrary permutation p1​,p2​,…,pN​ of the integers 1 through N. For each valid i, you should assign the value Api​​to vertex i

1. **Explanation:**

The dfs function performs a depth-first search on the tree, calculating the distance from the root to each node (d[x]). It also finds the minimum sum of distances (mn) needed to cover the tree.

The solve function reads input, builds the tree, finds the root, calculates the necessary values, and then computes the final result based on the problem statement.

The main function reads the number of test cases (tt) and calls the solve function for each test case.

1. **Algorithm :**

function dfs(x, p = -1):

**Step 1:** Define a DFS function that traverses the tree, calculates the distance from the root to each node (d[x]), and finds the minimum sum of distances (mn).

**Step 2:** If the current node is a leaf (size of its adjacency list is 1) and it's not the root, update the minimum sum (mn).

**Step 3:** Recursively explore the adjacent nodes, keeping track of the two smallest distances (m1 and m2).

function solve():

**Step 1:** Read the number of nodes n and the target sum k.

**Step 2:** Read the array a representing the values associated with each node.

**Step 3:** Build the tree by reading the edges between nodes and populating the adjacency list.

**Step 4:** Find the root of the tree by identifying the node with only one adjacency (leaf node).

**Step 5:** Initialize the minimum distance variable (mn) and calculate distances using DFS.

**Step 6:** Sort the array a in descending order.

**Step 7:** Calculate the sum s of the top mn values in the sorted array.

**Step 8:** Adjust k based on the relationship between k and mn.

**Step 9:** Calculate the final answer (ans) by summing the appropriate values from the array.

**Step 10:** Output the result.

1. **Code :**

#include <bits/stdc++.h>

using namespace std;

using ll = long long;

using ld = long double;

#define ft first

#define sd second

constexpr ll N = ll(3e5) + 5;

constexpr int MOD = int(1e9) + 7;

constexpr int inf = 0x3f3f3f3f;

int a[N], d[N], mn;

vector<int> v[N];

int dfs(int x, int p = -1){

if(v[x].size() == 1 && p != -1){

mn = min(mn, d[x]);

return 1;

}

int m1 = inf, m2 = inf;

for(auto& i : v[x]){

if(i == p) continue;

d[i] = 1 + d[x];

int k = dfs(i, x);

if(k < m1){

m2 = m1;

m1 = k;

}

else if(k < m2) m2 = k;

}

if(m2 != inf) mn = min(mn, m1 + m2);

return m1 + 1;

}

void solve(){

int n, k;

cin >> n >> k;

for(int i = 0; i < n; i++) cin >> a[i];

for(int i = 1; i < n; i++){

int x, y;

cin >> x >> y;

v[x].push\_back(y);

v[y].push\_back(x);

}

int r = 1;

for(int i = 1; i <= n; i++){

if(v[i].size() == 1) r = i;

}

mn = inf;

d[r] = 0;

dfs(r);

mn++;

sort(a, a + n, greater<int>());

ll ans = 0, s = 0;

for(int i = 0; i < mn; i++) s += a[i];

if((k / mn) & 1){

ans = (k / mn - 1) \* s;

k = k % mn + mn;

}

else{

ans = k / mn \* s;

k = k % mn;

}

if(k < 3){

ans += a[0] \* (k >= 1);

ans += a[1] \* (k == 2);

}

else for(int i = 0; i < mn && k > 0; i++){

ans += min(k, 2) \* a[i];

k -= 2;

}

cout << ans << '\n';

for(int i = 1; i <= n; i++) v[i].clear();

}

int main() {

ios::sync\_with\_stdio(false);

cin.tie(nullptr);

int tt = 1;

cin >> tt;

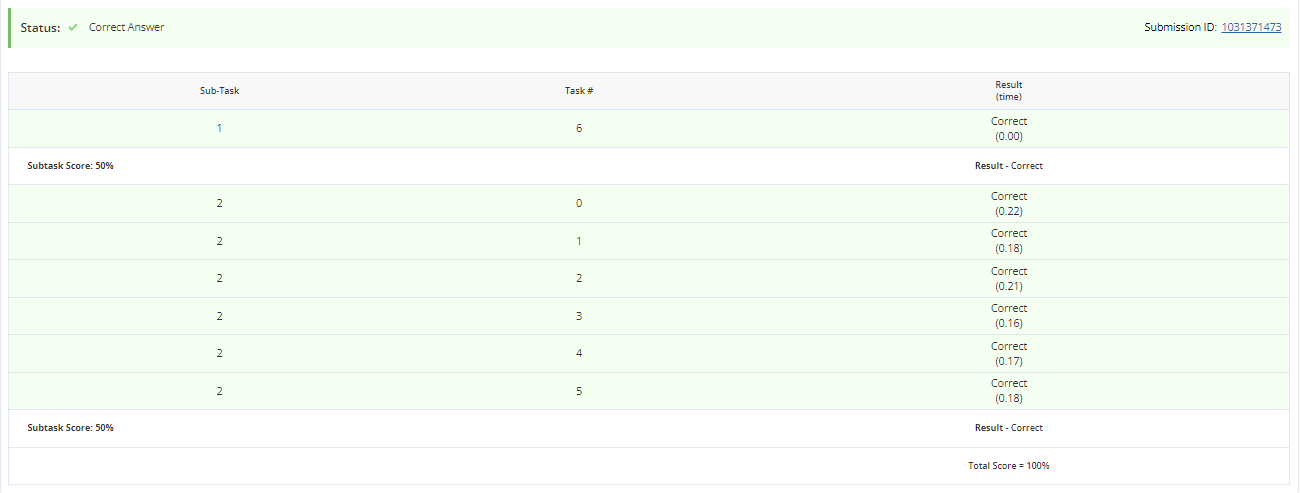
for(int i = 0; i < tt; i++) {

solve();

}

}

1. **Output:**

****

**Learning outcomes (What I have learnt):**

1. **Learned how to use the vectors in C++.**
2. **Learned how to do competitive coding.**
3. **Learned about the permutation and combination in terms of coding language.**