CSC3100 Assignment 3

Xue Zhongkai 122090636

Problem 1

This problem is rather simple and could be solved with basic Python conditional statements. Traverse through the letters of the word, accumulate the times one letter appears and we could execute corresponding outputs.
The complexity of the problem is O(N).
Here is the accumulation:

```
a = b = 0
for ch in s:
    if ch == 'A':
        a += 1
    elif ch == 'B':
        b += 1
    else:
        pass
```

Problem 2

For this kind of the problem, it is natural to derive an expression before the numerical calculation. Instead, it would cause O(N^2) complexity if we compute and compare r1, r2 respectively.
By the inequality V1 > V2, we could derive d2 = r2 – (r2 ^3 – 3V1 / 4pi) ^ (1/3). The calculation only costs O(1) each, which combines to make an O(N) complexity.

```
def find_d2(r1, r2, d1):
    d2 = r2 - pow(r2**3 - (r1**3 - (r1 - d1)**3), 1/3)
    return d2
```

Problem 3

This problem is a classical tree problem calculating the weighted sum of one node to all other nodes. Of course dfs is needed when traversing through all the nodes, but a single dfs with the starting point updated every time will cause a nearly O(N^2) complexity. Hence we should absorb the spirits of flatting the task to multiple O(N) and to some extent, try to save the information of the path already visited.
In order to adapt to the updated information, I keep a *class* of BlackWhiteTree, which is basically a list of *struct* Nodes storing neighbouring node and the corresponding weight. This design would be good for self-recursion as parameters could be updated in place.

```
struct Node {
    vector<pair<int, int> > adj;
};

class BlackWhiteTree {
private:
    vector<Node> tree;
    vector<int> color;
    vector<long long> count, dis;
    vector<long long> ans;
    long long total = 0;

public:
    BlackWhiteTree(int node_num) : tree(node_num), color(node_num),
                        count(node_num),dis(node_num), ans(node_num) {}
```

The core part is composed of two dfs. The initial one traversing all the neighbours, and calculate *count* for the sub-trees recursively, which is for distinguishing between white and black nodes. The current *count* is equal to itself added by that of sub-trees.
The later dfs adjusts the distances of current node's neighbours, and accmulates by the node to neighbour through root and without passing through root.

```
void dfsHelper(int node, int parent) {
        count[node] = color[node];
        for (auto& adj_weight : tree[node].adj) {
            int adj = adj_weight.first;
            int weight = adj_weight.second;
            if (adj != parent) {
```

```
                dfsHelper(adj, node);
                count[node] += count[adj];
                dis[node] += dis[adj] + count[adj] * weight;
            }
        }
    }

    void dfs(int node, int parent) {
        ans[node] = dis[node];
        for (auto& adj_weight : tree[node].adj) {
            int adj = adj_weight.first;
            int weight = adj_weight.second;
            if (adj != parent) {
                long long node_to_adj_root = dis[node] - (dis[adj] + count[adj] *
weight);
                long long node_to_adj = (total - count[adj]) * weight;
                dis[adj] += node_to_adj_root + node_to_adj;
                dfs(adj, node);
            }
        }
    }
```

After that, we initialize a tree and assign the attributes, after which we execute the main function and get the results.

```
int main() {
    int n;
    cin >> n;
    BlackWhiteTree tree(n);

    vector<int> colors(n);
    for (int i = 0; i < n; ++i) {
        cin >> colors[i];
    }
    tree.saveColor(colors);

    for (int i = 0; i < n - 1; ++i) {
        int u, v, w;
        cin >> u >> v >> w;
        tree.addEdge(u - 1, v - 1, w);
    }

    const vector<long long>& answers = tree.answer();
    for (int i = 0; i < n; ++i) {
        cout << (answers[i] % MOD) << endl;
    }

    return 0;
}
```