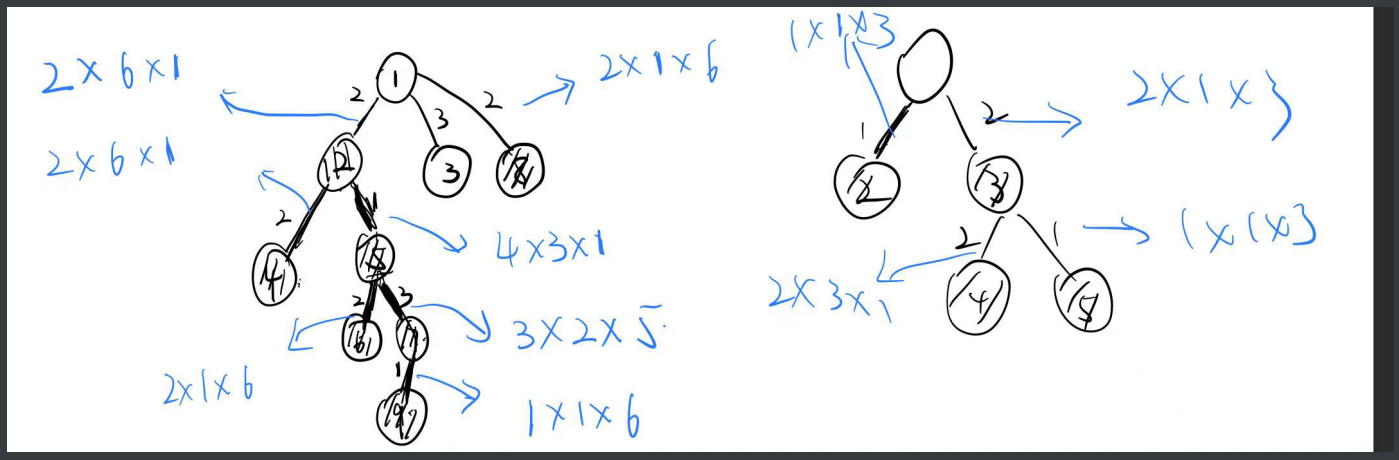
CSC3100 Assignment 3 Report 122090837

**P1**

1. Define a TreeNode Class with three attributes, color, weight(edge with its father), and child array list.
2. Then we construct a tree and initialize all n nodes with corresponding colors.
3. Inputting $n-1$ lines from the second node to the n-th node, and assign the node to their father's child array list.
4. By observation, I find that the counting of each edge has a relationship with the number of the black nodes of  the child node and its subtree, that is, take the edge $1->2$ (no direction, but to show it more clearly with an arrow) as an example of the figure in the left. The edge is passed through 6 times since the eighth node should go through it to all the other black nodes. By applying this to other edges, we can easily find that the distance which counting each edge should be $weight \times black\ nodes\ of\ the\ subtree \times the\ rest\ of\ the\ other\ black\ nodes$ , which can be expressed as $weight \times subtreeBlackSize \times (blackCounts - subtreeBlackNodes)$




5. Now we have the tree and we find the pattern. To traversal the tree, we can use Depth First Search(DFS).

```
public static int dfs(TreeNode node) {
        int subtreeBlackSize = node.color;// if is black, count itself
        for(TreeNode child : node.children){
            subtreeBlackSize += dfs(child);//find subtreeBlackSize by
recursion
        }
        //update the total distance, since we have counted all the
situation of this edge
        totalDistance += (long)node.weight * (blackCount -
subtreeBlackSize) * subtreeBlackSize;
        return subtreeBlackSize;
    }
```

**P2**

1. To maintain a stream of statistics, we'd better use a linked list rather than an array.
2. While adding the number to the linked list, we can keep updating the minimum absolute difference between adjacent prices
3. We also maintain a Tree Map whose key represents the number contained in the linked list and whose value represents the counting of each number. The reason why I use TreeMap is that TreeMap can not only have a sorted array of the number as its key, but it can also record the count of each number as its key.

```
LinkedList<Long> storage = new LinkedList<>();
        TreeMap<Long,Integer> counts = new TreeMap<>();
        Long minADJDifference = Long.MAX_VALUE;
        Long minDifference = Long.MAX_VALUE;
        for (int i = 0; i < n; i++) {
            long a = sc.nextLong();
            storage.add(a);
            if(counts.containsKey(a)){
                counts.put(a,counts.get(a) + 1);
            }else{
                counts.put(a,1);
            }
```

```
            if(i > 0){
                minADJDifference = Math.min(minADJDifference,
Math.abs(storage.get(i) - storage.get(i - 1)));
                minDifference =
findMInDifference(counts,a,minDifference);
            }
        }
```

4. The rest of the operation is quite easy.
   BUY: add the element to the storage(The Linked List) and update the minADJDifference

   check if the key is contained in the counts(The Tree Map), if yes update the value to value + 1, if not, add the element to a new key with value 1.

   Then use the method findMInDifference to update minDifference.

```
 private static long findMInDifference(TreeMap<Long,Integer> map,long
a,long minDifference) {
        if(map.get(a) > 1) return 0;
        Long higherKey = map.higherKey(a);
            if(higherKey != null){
                minDifference =
Math.min(minDifference,Math.abs(higherKey - a));
            }
            Long lowerKey = map.lowerKey(a);
            if(lowerKey != null){
                minDifference =
Math.min(minDifference,Math.abs(lowerKey - a));
            }
        return minDifference;
    }
```

5. Other operations just take and print the corresponding results.