Problem 1:

You are given a tree with n nodes, where each edge in the tree has a corresponding weight denoting the length of each edge. The nodes in the tree are colored either black or white. Your task is to calculate the sum of distances between every pair of black nodes in the tree. Let B = {b1, b2, ...} a set of black nodes, then the answer is formulated as:

where |B| denotes the number of the black nodes in the tree, and dist(bi, bj) is the length of the simple path from the i-th to j-th black node.

Write a program to calculate the sum of distances on the tree between every pair of black nodes Ans in the given tree.

Solution:

First, we can use depth-first search (DFS) to traverse the tree. We can start at the root node, traverse the tree, and calculate the distance from each node to all the black nodes. We can use an array to record the distance from each node to all the black nodes. We can then use this array to calculate the distance between each pair of black nodes. Finally, we can add all the distances together to get the answer. The time complexity of this algorithm is O(n^2), where n is the number of nodes in the tree. If you want a more efficient algorithm, you can try using dynamic programming or other more advanced algorithms. In this algorithm, we can use dynamic programming to optimize the time complexity. We can use two arrays to record, respectively, the sum of distances from each node to all black nodes in the subtree and the number of black nodes in the subtree. We can then use these arrays to calculate the distance from each node to all the black nodes. The time complexity of this algorithm is O(n), where n is the number of nodes in the tree. This algorithm is more efficient than using DFS because it avoids double calculations.

Problem 2:

Description

Mario bought n math books and he recorded their prices. The prices are all integers, and the price sequence is a = {a0, a2, ...ai, ..., an − 1} of length n (n ⩽ 100000). Please help him to manage this price sequence. There are three types of operations:

- BUY x: buy a new book with price x, thus x is added at the end of a.
- CLOSEST ADJ PRICE: output the minimum absolute difference between adjacent prices.
- CLOSEST PRICE: output the absolute difference between the two closest prices in the entire sequence.

A total of m operations are performed (1 ⩽ m ⩽ 100000). Each operation is one of the three mentioned types. You need to write a program to perform given operations. For operations "CLOSEST ADJ PRICE" and "CLOSEST PRICE" you need to output the corresponding answers

Solution:

We can use an array to store the price sequence of a book. For the BUY x operation, we can add x to the end of the array. For the operation CLOSEST ADJ PRICE, we need to find the smallest absolute difference between two adjacent prices. We can use a loop to go through the set of numbers, calculate the absolute difference between two adjacent prices, and find the minimum. For the operation CLOSEST PRICE, we need to find the smallest absolute difference between the two prices in the entire sequence. We can use a sorting algorithm to solve this problem. Specifically, we can use a quicksort algorithm to sort a price sequence, and then traverse the sorted sequence, calculating the absolute difference between two adjacent prices, and finding the minimum value. The time complexity of this algorithm is O(nlogn), where n is the number of elements in the array. If you want a more efficient algorithm, you can try using other more advanced algorithms such as radix sort, heapsort, merge sort, etc.