2467. Most Profitable Path in a Tree

There is an undirected tree with `n` nodes labeled from `0` to `n - 1`, rooted at node `0`. You are given a 2D integer array `edges` of length `n - 1` where `edges[i] = [ai, bi]` indicates that there is an edge between nodes `ai` and `bi` in the tree.

At every node `i`, there is a gate. You are also given an array of even integers `amount`, where `amount[i]` represents:

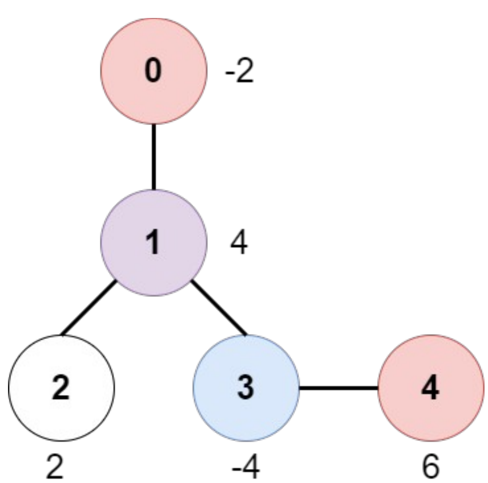
- the price needed to open the gate at node `i`, if `amount[i]` is negative, or,
- the cash reward obtained on opening the gate at node `i`, otherwise.

The game goes on as follows:

- Initially, Alice is at node `0` and Bob is at node `bob`.
- At every second, Alice and Bob **each** move to an adjacent node. Alice moves towards some **leaf node**, while Bob moves towards node `0`.
- For **every** node along their path, Alice and Bob either spend money to open the gate at that node, or accept the reward. Note that:
  - If the gate is **already open**, no price will be required, nor will there be any cash reward.
  - If Alice and Bob reach the node **simultaneously**, they share the price/reward for opening the gate there. In other words, if the price to open the gate is `c`, then both Alice and Bob pay `c / 2` each. Similarly, if the reward at the gate is `c`, both of them receive `c / 2` each.
- If Alice reaches a leaf node, she stops moving. Similarly, if Bob reaches node `0`, he stops moving. Note that these events are **independent** of each other.

Return *the **maximum** net income Alice can have if she travels towards the optimal leaf node.*

Example 1:



```
Input: edges = [[0,1],[1,2],[1,3],[3,4]], bob = 3, amount = [-2,4,2,-4,6]
Output: 6
Explanation:
The above diagram represents the given tree. The game goes as follows:
- Alice is initially on node 0, Bob on node 3. They open the gates of their
respective nodes.
  Alice's net income is now -2.
- Both Alice and Bob move to node 1.
  Since they reach here simultaneously, they open the gate together and share the
reward.
  Alice's net income becomes -2 + (4 / 2) = 0.
- Alice moves on to node 3. Since Bob already opened its gate, Alice's income
remains unchanged.
  Bob moves on to node 0, and stops moving.
- Alice moves on to node 4 and opens the gate there. Her net income becomes 0 + 6 =
6.
Now, neither Alice nor Bob can make any further moves, and the game ends.
It is not possible for Alice to get a higher net income.
```

Example 2:


```
Input: edges = [[0,1]], bob = 1, amount = [-7280,2350]
Output: -7280
Explanation:
Alice follows the path 0->1 whereas Bob follows the path 1->0.
Thus, Alice opens the gate at node 0 only. Hence, her net income is -7280.
```

Constraints:

- `2 <= n <=` 10^5
- `edges.length == n - 1`
- `edges[i].length == 2`
- `0 <= ai, bi < n`
- `ai != bi`
- `edges` represents a valid tree.
- `1 <= bob < n`
- `amount.length == n`
- `amount[i]` is an **even** integer in the range $[-10^4, 10^4]$.

2467. Most Profitable Path in a Tree

Overview

We are given a tree with `n` nodes, where `n - 1`edges define its structure. The tree is rooted at node `0`. Additionally, we are provided with an array `amount` of size `n`, where each element represents the value of a node. All values in `amount` are even integers. Finally, we are given an integer `bob`, which indicates the starting node for Bob.

The two players Alice and Bob, traverse the tree simultaneously under the following conditions:

1. Alice starts at node `0` and moves towards a leaf node (a node with only one connection).
2. Bob starts at node `bob` and moves towards node `0` along the shortest path.

For each node visited, the income calculations follow these rules:

- If a player reaches a node first, they collect the full value of that node.
- If both players arrive at the same node at the same time, they split the value equally.
- If a node was previously visited by the other player, no income is collected.

Our goal is to find the largest (maximum) income Alice can collect by choosing an optimal path toward a leaf node.

Let's look at an example of finding the maximum income that Alice can achieve:

2467. Most Profitable Path in a Tree

```
/*
    Double Pass DFS
    Time complexity: O(6n)
    Space complexity: O(6n)
*/
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
class Solution{
    private:
        vvi graph;
        int n;
        vi parent,depth;
    public:
        void build_graph(vvi& edges){
            graph.resize(n);
            for(auto& edge: edges){
                int u=edge[0];
                int v=edge[1];
                graph[u].push_back(v);
                graph[v].push_back(u);
            }
        }

        int mostProfitablePath(vvi& edges, int bob, vi& amount){
            // Function DFS having double role:
            // 1st call: Compute the depth and the parent of each node while Alice move
            // 2nd call: Compute the max income of Alice, While both of them move
            auto dfs=[&](int u,int par,int d,auto& self)->int{
                int alice_income=amount[u];
                int max_path=INT_MIN;
                depth[u]=d;
                parent[u]=par;
                for(auto& v: graph[u]){
                    if(v!=par) max_path=std::max(max_path,self(v,u,d+1,self));
                }

                return max_path==INT_MIN?alice_income:alice_income+max_path;
            };
```

```
        n=edges.size()+1;

        build_graph(edges);

        // Phase I: Depths from Alice's view
        // Starting from 0(Alice start node), get depths and parent of each node

        depth.resize(n,0); // Store each node's depth while Alice moves
        parent.resize(n); // Store each node's parent while Alice moves

        // While Alice moves: Compute each node's depth and each node's parent
        dfs(0,-1,0,dfs);

        // Phase II: Depth of each node in Alice's path from Bob's view
        // Starting from bob's start node, determine the depth of each node

        int current=bob,bob_depth=0;
        // While parent of Alice's start node (0) not reached
        while(current!=-1){
           // If Alice's node's depth>Bob's node's depth, means bob moves in that node first
           if(depth[current]>bob_depth) amount[current]=0;
           // If Alice's node's depth==Bob's node's depth,
           // means both Alice and Bob moves in that node IN SAME TIME
           else if(depth[current]==bob_depth) amount[current]/=2;

           // Otherwise, Alice moves first to that node, so she will take all


           current=parent[current]; // Got next node
           bob_depth++; // Bob moves to next node
        }

        // At this point, Bob update all nodes in Alice's path,
        // Simulate the moves of Alice and Bob and compute Alice's max income
        return dfs(0,-1,0,dfs);
    }
};
```

2467. Most Profitable Path in a Tree

```
/*
    One Pass DFS
    Time complexity: O(3n)
    Space compelxity: O(4n)
*/
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
class Solution{
    private:
        vvi graph;
        int n;
        vi depth; // Bob's depth
    public:
        void build_graph(vvi& edges){
            graph.resize(n);
            for(auto& edge: edges){
                int u=edge[0];
                int v=edge[1];
                graph[u].push_back(v);
                graph[v].push_back(u);
            }
        }
```

```
int mostProfitablePath(vvi& edges, int bob, vi& amount){

        auto dfs=[&](int u,int par,int alice_depth,int& alice_score,auto& self)->void{
            // Initialize Alice's income to 0
            int alice_take=0;

            // Initialize Bob's depth
            if(u==bob) depth[u]=0;
            else depth[u]=INT_MAX/2;

            // Maximum of each path to a leaf node
            int max_path=INT_MIN;

            for(auto& v: graph[u]){
                if(v!=par){
                    self(v,u,alice_depth+1,max_path,self);

                    // Update Bob's depth while we identifying his path
                    depth[u]=std::min(depth[u],depth[v]+1);
                }
            }
            // If Bob's depth > Alice's, means Alice visit that node first
            // So, Alice will take all
            if(depth[u]>alice_depth) alice_take=amount[u];
            // If both depths are equal, they wil share the value at that node
            else if(depth[u]==alice_depth) alice_take=amount[u]/2;

            alice_score=std::max(alice_score,max_path==INT_MIN?alice_take:alice_take+max_path);
        };

        n=edges.size()+1;
        build_graph(edges);
        depth.resize(n,0);
        int alice_score=INT_MIN;
        dfs(0,-1,0,alice_score,dfs);

        return alice_score;
    }
};
```