Problem 2538: Difference Between Maximum and Minimum Price Sum

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

There exists an undirected and initially unrooted tree with

n

nodes indexed from

0

to

n - 1

. You are given the integer

n

and a 2D integer array

edges

of length

n - 1

, where

edges[i] = [a

i

, b

i

]

indicates that there is an edge between nodes

a

i

and

b

i

in the tree.

Each node has an associated price. You are given an integer array

price

, where

price[i]

is the price of the

i

th

node.

The

price sum

of a given path is the sum of the prices of all nodes lying on that path.

The tree can be rooted at any node

root

of your choice. The incurred

cost

after choosing

root

is the difference between the maximum and minimum

price sum

amongst all paths starting at

root

.

Return

the

maximum

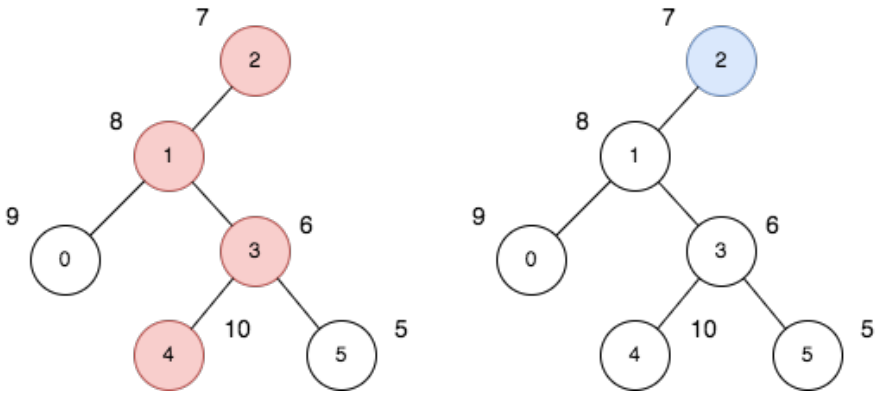
possible

cost

amongst all possible root choices

.

Example 1:



Input:

n = 6, edges = [[0,1],[1,2],[1,3],[3,4],[3,5]], price = [9,8,7,6,10,5]

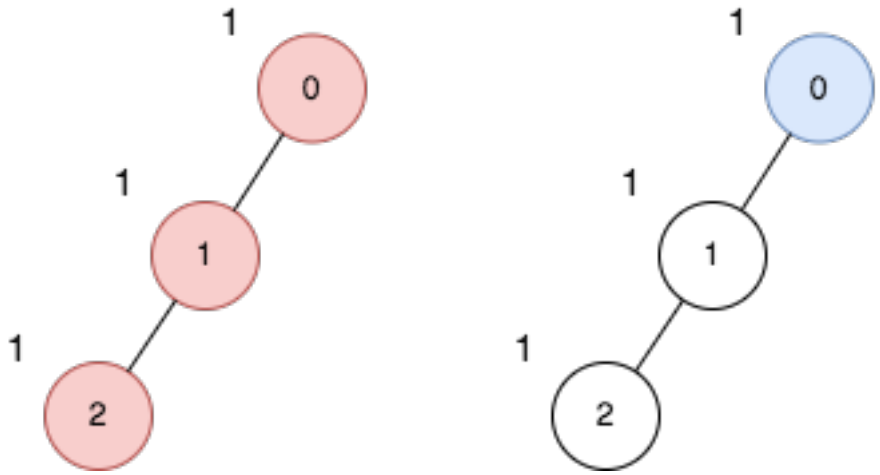
Output:

24

Explanation:

The diagram above denotes the tree after rooting it at node 2. The first part (colored in red) shows the path with the maximum price sum. The second part (colored in blue) shows the path with the minimum price sum. - The first path contains nodes [2,1,3,4]: the prices are [7,8,6,10], and the sum of the prices is 31. - The second path contains the node [2] with the price [7]. The difference between the maximum and minimum price sum is 24. It can be proved that 24 is the maximum cost.

Example 2:



Input:

n = 3, edges = [[0,1],[1,2]], price = [1,1,1]

Output:

2

Explanation:

The diagram above denotes the tree after rooting it at node 0. The first part (colored in red) shows the path with the maximum price sum. The second part (colored in blue) shows the path with the minimum price sum. - The first path contains nodes [0,1,2]: the prices are [1,1,1], and the sum of the prices is 3. - The second path contains node [0] with a price [1]. The difference between the maximum and minimum price sum is 2. It can be proved that 2 is the maximum cost.

Constraints:

1 <= n <= 10

5

edges.length == n - 1

0 <= a

i

, b

i

<= n - 1

edges

represents a valid tree.

price.length == n

1 <= price[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
long long maxOutput(int n, vector<vector<int>>& edges, vector<int>& price) {

}
};
```

Java:

```
class Solution {
public long maxOutput(int n, int[][] edges, int[] price) {

}
}
```

Python3:

```
class Solution:
def maxOutput(self, n: int, edges: List[List[int]], price: List[int]) -> int:
```

Python:

```
class Solution(object):
def maxOutput(self, n, edges, price):
"""
:type n: int
:type edges: List[List[int]]
:type price: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} edges
* @param {number[]} price
* @return {number}
*/
var maxOutput = function(n, edges, price) {

};
```

TypeScript:

```
function maxOutput(n: number, edges: number[][], price: number[]): number {

};
```

C#:

```
public class Solution {
public long MaxOutput(int n, int[][] edges, int[] price) {

}
}
```

C:

```
long long maxOutput(int n, int** edges, int edgesSize, int* edgesColSize,
int* price, int priceSize) {

}
```

Go:

```
func maxOutput(n int, edges [][]int, price []int) int64 {

}
```

Kotlin:

```
class Solution {
fun maxOutput(n: Int, edges: Array<IntArray>, price: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func maxOutput(_ n: Int, _ edges: [[Int]], _ price: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_output(n: i32, edges: Vec<Vec<i32>>, price: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer[]} price
# @return {Integer}
def max_output(n, edges, price)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer[] $price
* @return Integer
*/
function maxOutput($n, $edges, $price) {

}
}
```

Dart:

```
class Solution {
int maxOutput(int n, List<List<int>> edges, List<int> price) {

}
}
```

Scala:

```
object Solution {
def maxOutput(n: Int, edges: Array[Array[Int]], price: Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_output(n :: integer, edges :: [[integer]], price :: [integer]) ::
integer
def max_output(n, edges, price) do

end
end
```

Erlang:

```
-spec max_output(N :: integer(), Edges :: [[integer()]], Price ::
[integer()]) -> integer().
```

```
max_output(N, Edges, Price) ->
.
```

Racket:

```
(define/contract (max-output n edges price)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Difference Between Maximum and Minimum Price Sum
* Difficulty: Hard
* Tags: array, tree, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
long long maxOutput(int n, vector<vector<int>>& edges, vector<int>& price) {

}
};
```

Java Solution:

```
/**
* Problem: Difference Between Maximum and Minimum Price Sum
* Difficulty: Hard
* Tags: array, tree, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public long maxOutput(int n, int[][] edges, int[] price) {

}
}
```

Python3 Solution:

```
"""
Problem: Difference Between Maximum and Minimum Price Sum
Difficulty: Hard
Tags: array, tree, dp, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def maxOutput(self, n: int, edges: List[List[int]], price: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxOutput(self, n, edges, price):
"""
:type n: int
:type edges: List[List[int]]
:type price: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Difference Between Maximum and Minimum Price Sum
```

```
* Difficulty: Hard
* Tags: array, tree, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} n
* @param {number[][]} edges
* @param {number[]} price
* @return {number}
*/
var maxOutput = function(n, edges, price) {

};
```

TypeScript Solution:

```
/**
* Problem: Difference Between Maximum and Minimum Price Sum
* Difficulty: Hard
* Tags: array, tree, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxOutput(n: number, edges: number[][], price: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Difference Between Maximum and Minimum Price Sum
* Difficulty: Hard
* Tags: array, tree, dp, search
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public long MaxOutput(int n, int[][] edges, int[] price) {

}
}
```

C Solution:

```
/*
* Problem: Difference Between Maximum and Minimum Price Sum
* Difficulty: Hard
* Tags: array, tree, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

long long maxOutput(int n, int** edges, int edgesSize, int* edgesColSize,
int* price, int priceSize) {

}
```

Go Solution:

```
// Problem: Difference Between Maximum and Minimum Price Sum
// Difficulty: Hard
// Tags: array, tree, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxOutput(n int, edges [][]int, price []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxOutput(n: Int, edges: Array<IntArray>, price: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func maxOutput(_ n: Int, _ edges: [[Int]], _ price: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Difference Between Maximum and Minimum Price Sum
// Difficulty: Hard
// Tags: array, tree, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_output(n: i32, edges: Vec<Vec<i32>>, price: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer[]} price
# @return {Integer}
def max_output(n, edges, price)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer[] $price
* @return Integer
*/
function maxOutput($n, $edges, $price) {

}
}
```

Dart Solution:

```
class Solution {
int maxOutput(int n, List<List<int>> edges, List<int> price) {

}
}
```

Scala Solution:

```
object Solution {
def maxOutput(n: Int, edges: Array[Array[Int]], price: Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_output(n :: integer, edges :: [[integer]], price :: [integer]) ::
integer
def max_output(n, edges, price) do

end
end
```

Erlang Solution:

```
-spec max_output(N :: integer(), Edges :: [[integer()]], Price ::
[integer()]) -> integer().
max_output(N, Edges, Price) ->
.
```

Racket Solution:

```
(define/contract (max-output n edges price)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
exact-integer?)
)
```