Problem 2646: Minimize the Total Price of the Trips

Problem Information

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

Problem Description

There exists an undirected and 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.

Additionally, you are given a 2D integer array

trips

, where

trips[i] = [start

i

, end

i

]

indicates that you start the

i

th

trip from the node

start

i

and travel to the node

end

i

by any path you like.

Before performing your first trip, you can choose some

non-adjacent

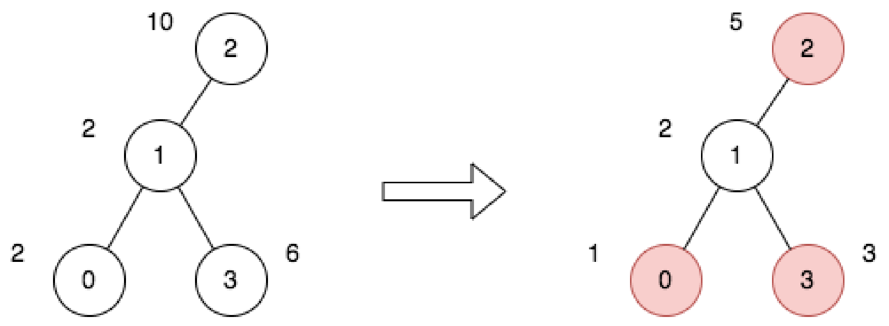
nodes and halve the prices.

Return

the minimum total price sum to perform all the given trips

.

Example 1:



Input:

n = 4, edges = [[0,1],[1,2],[1,3]], price = [2,2,10,6], trips = [[0,3],[2,1],[2,3]]

Output:

23

Explanation:

The diagram above denotes the tree after rooting it at node 2. The first part shows the initial tree and the second part shows the tree after choosing nodes 0, 2, and 3, and making their price half. For the 1 st

trip, we choose path [0,1,3]. The price sum of that path is 1 + 2 + 3 = 6. For the 2

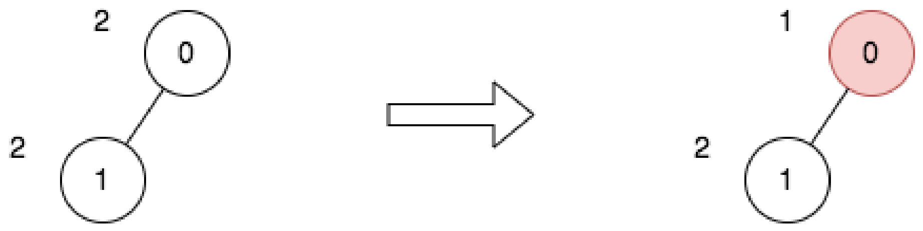
nd

trip, we choose path [2,1]. The price sum of that path is 2 + 5 = 7. For the 3

rd

trip, we choose path [2,1,3]. The price sum of that path is 5 + 2 + 3 = 10. The total price sum of all trips is 6 + 7 + 10 = 23. It can be proven, that 23 is the minimum answer that we can achieve.

Example 2:



Input:

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

Output:

1

Explanation:

The diagram above denotes the tree after rooting it at node 0. The first part shows the initial tree and the second part shows the tree after choosing node 0, and making its price half. For the 1

st

trip, we choose path [0]. The price sum of that path is 1. The total price sum of all trips is 1. It can be proven, that 1 is the minimum answer that we can achieve.

Constraints:

1 <= n <= 50

edges.length == n - 1

0 <= a

i

, b

i

<= n - 1

edges

represents a valid tree.

price.length == n

price[i]

is an even integer.

1 <= price[i] <= 1000

1 <= trips.length <= 100

0 <= start

i

, end

i

<= n - 1

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int minimumTotalPrice(int n, int[][] edges, int[] price, int[][]
trips) {

}
}
```

Python3:

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

Python:

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

```
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int MinimumTotalPrice(int n, int[][] edges, int[] price, int[][]
trips) {

}
}
```

C:

```
int minimumTotalPrice(int n, int** edges, int edgesSize, int* edgesColSize,
int* price, int priceSize, int** trips, int tripsSize, int* tripsColSize) {

}
```

Go:

```
func minimumTotalPrice(n int, edges [][]int, price []int, trips [][]int) int
{
}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (minimum-total-price n edges price trips)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
(listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimize the Total Price of the Trips
* Difficulty: Hard
* Tags: array, tree, graph, 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:
int minimumTotalPrice(int n, vector<vector<int>>& edges, vector<int>& price,
vector<vector<int>>& trips) {

}
};
```

Java Solution:

```
/**
* Problem: Minimize the Total Price of the Trips
* Difficulty: Hard
* Tags: array, tree, graph, 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 int minimumTotalPrice(int n, int[][] edges, int[] price, int[][]
trips) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimize the Total Price of the Trips
Difficulty: Hard
Tags: array, tree, graph, 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 minimumTotalPrice(self, n: int, edges: List[List[int]], price: List[int],
trips: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimize the Total Price of the Trips
* Difficulty: Hard
* Tags: array, tree, graph, 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
* @param {number[][]} trips
* @return {number}
*/
var minimumTotalPrice = function(n, edges, price, trips) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimize the Total Price of the Trips
* Difficulty: Hard
* Tags: array, tree, graph, 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 minimumTotalPrice(n: number, edges: number[][], price: number[],
trips: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimize the Total Price of the Trips
* Difficulty: Hard
* Tags: array, tree, graph, 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 int MinimumTotalPrice(int n, int[][] edges, int[] price, int[][]
trips) {

}
}
```

C Solution:

```
/*
* Problem: Minimize the Total Price of the Trips
* Difficulty: Hard
* Tags: array, tree, graph, 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
*/

int minimumTotalPrice(int n, int** edges, int edgesSize, int* edgesColSize,
int* price, int priceSize, int** trips, int tripsSize, int* tripsColSize) {

}
```

Go Solution:

```
// Problem: Minimize the Total Price of the Trips
// Difficulty: Hard
// Tags: array, tree, graph, 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 minimumTotalPrice(n int, edges [][]int, price []int, trips [][]int) int
{

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimize the Total Price of the Trips
// Difficulty: Hard
// Tags: array, tree, graph, 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 minimum_total_price(n: i32, edges: Vec<Vec<i32>>, price: Vec<i32>,
trips: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

```
object Solution {
def minimumTotalPrice(n: Int, edges: Array[Array[Int]], price: Array[Int],
trips: Array[Array[Int]]): Int = {
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (minimum-total-price n edges price trips)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
(listof (listof exact-integer?)) exact-integer?)
)
```