Problem 3553: Minimum Weighted Subgraph With the Required Paths II

Problem Information

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

Problem Description

You are given an

undirected weighted

tree with

n

nodes, numbered from

0

to

n - 1

. It is represented by a 2D integer array

edges

of length

n - 1

, where

edges[i] = [u

i

, v

i

, w

i

]

indicates that there is an edge between nodes

u

i

and

v

i

with weight

w

i

.

Additionally, you are given a 2D integer array

queries

, where

queries[j] = [src1

j

, src2

j

, dest

j

]

.

Return an array

answer

of length equal to

queries.length

, where

answer[j]

is the

minimum total weight

of a subtree such that it is possible to reach

dest

j

from both

src1

j

and

src2

j

using edges in this subtree.

A

subtree

here is any connected subset of nodes and edges of the original tree forming a valid tree.

Example 1:

Input:

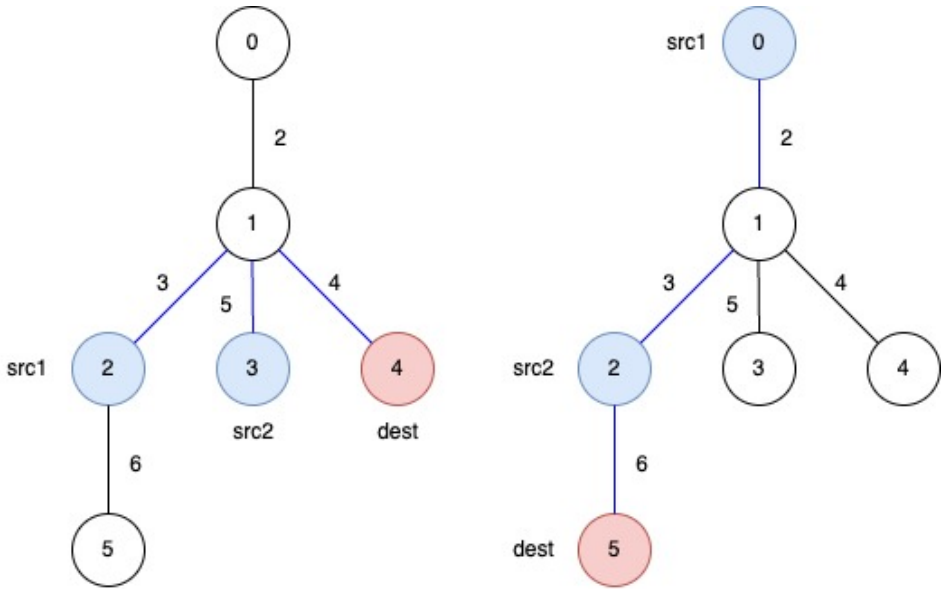
edges = [[0,1,2],[1,2,3],[1,3,5],[1,4,4],[2,5,6]], queries = [[2,3,4],[0,2,5]]

Output:

[12,11]

Explanation:

The blue edges represent one of the subtrees that yield the optimal answer.



answer[0]

: The total weight of the selected subtree that ensures a path from

src1 = 2

and

src2 = 3

to

dest = 4

is

3 + 5 + 4 = 12

.

answer[1]

: The total weight of the selected subtree that ensures a path from

src1 = 0

and

src2 = 2

to

dest = 5

is

2 + 3 + 6 = 11

.

Example 2:

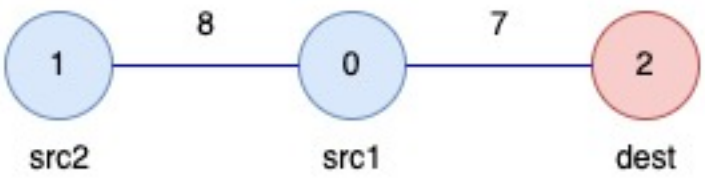
Input:

edges = [[1,0,8],[0,2,7]], queries = [[0,1,2]]

Output:

[15]

Explanation:



answer[0]

: The total weight of the selected subtree that ensures a path from

src1 = 0

and

src2 = 1

to

dest = 2

is

8 + 7 = 15

.

Constraints:

3 <= n <= 10

5

edges.length == n - 1

edges[i].length == 3

0 <= u

i

, v

i

< n

1 <= w

i

<= 10

4

1 <= queries.length <= 10

5

queries[j].length == 3

0 <= src1

j

, src2

j

, dest

j

< n

src1

j

,

src2

j

, and

dest

j

are pairwise distinct.

The input is generated such that

edges

represents a valid tree.

Code Snippets

C++:

```
class Solution {
public:
vector<int> minimumWeight(vector<vector<int>>& edges, vector<vector<int>>&
queries) {

}
};
```

Java:

```
class Solution {
public int[] minimumWeight(int[][] edges, int[][] queries) {

}
}
```

Python3:

```
class Solution:
def minimumWeight(self, edges: List[List[int]], queries: List[List[int]]) ->
List[int]:
```

Python:

```
class Solution(object):
def minimumWeight(self, edges, queries):
"""
:type edges: List[List[int]]
:type queries: List[List[int]]
:rtype: List[int]
```

```
"""
```

JavaScript:

```
/**
* @param {number[][]} edges
* @param {number[][]} queries
* @return {number[]}
*/
var minimumWeight = function(edges, queries) {

};
```

TypeScript:

```
function minimumWeight(edges: number[][], queries: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] MinimumWeight(int[][] edges, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* minimumWeight(int** edges, int edgesSize, int* edgesColSize, int**
queries, int queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go:

```
func minimumWeight(edges [][]int, queries [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun minimumWeight(edges: Array<IntArray>, queries: Array<IntArray>): IntArray
{

}
}
```

Swift:

```
class Solution {
func minimumWeight(_ edges: [[Int]], _ queries: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_weight(edges: Vec<Vec<i32>>, queries: Vec<Vec<i32>>) ->
Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} edges
# @param {Integer[][]} queries
# @return {Integer[]}
def minimum_weight(edges, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Integer[][] $queries
* @return Integer[]
```

```
*/
function minimumWeight($edges, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> minimumWeight(List<List<int>> edges, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def minimumWeight(edges: Array[Array[Int]], queries: Array[Array[Int]]):
Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_weight(edges :: [[integer]], queries :: [[integer]]) ::
[integer]
def minimum_weight(edges, queries) do

end
end
```

Erlang:

```
-spec minimum_weight(Edges :: [[integer()]], Queries :: [[integer()]]) ->
[integer()].
minimum_weight(Edges, Queries) ->
.
```

Racket:

```
(define/contract (minimum-weight edges queries)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Weighted Subgraph With the Required Paths II
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
vector<int> minimumWeight(vector<vector<int>>& edges, vector<vector<int>>&
queries) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Weighted Subgraph With the Required Paths II
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int[] minimumWeight(int[][] edges, int[][] queries) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Minimum Weighted Subgraph With the Required Paths II
Difficulty: Hard
Tags: array, tree, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

```
class Solution(object):
def minimumWeight(self, edges, queries):
"""
:type edges: List[List[int]]
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Weighted Subgraph With the Required Paths II
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number[][]} edges
* @param {number[][]} queries
* @return {number[]}
*/
var minimumWeight = function(edges, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Weighted Subgraph With the Required Paths II
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function minimumWeight(edges: number[][], queries: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Minimum Weighted Subgraph With the Required Paths II
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
```

```
public int[] MinimumWeight(int[][] edges, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Weighted Subgraph With the Required Paths II
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* minimumWeight(int** edges, int edgesSize, int* edgesColSize, int**
queries, int queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Minimum Weighted Subgraph With the Required Paths II
// Difficulty: Hard
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func minimumWeight(edges [][]int, queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumWeight(edges: Array<IntArray>, queries: Array<IntArray>): IntArray
{

}
}
```

Swift Solution:

```
class Solution {
func minimumWeight(_ edges: [[Int]], _ queries: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Minimum Weighted Subgraph With the Required Paths II
// Difficulty: Hard
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn minimum_weight(edges: Vec<Vec<i32>>, queries: Vec<Vec<i32>>) ->
Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} edges
# @param {Integer[][]} queries
# @return {Integer[]}
def minimum_weight(edges, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Integer[][] $queries
* @return Integer[]
*/
function minimumWeight($edges, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> minimumWeight(List<List<int>> edges, List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def minimumWeight(edges: Array[Array[Int]], queries: Array[Array[Int]]):
Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_weight(edges :: [[integer]], queries :: [[integer]]) ::
[integer]
def minimum_weight(edges, queries) do

end
end
```

Erlang Solution:

```
-spec minimum_weight(Edges :: [[integer()]], Queries :: [[integer()]]) ->
[integer()].
minimum_weight(Edges, Queries) ->
.
```

Racket Solution:

```
(define/contract (minimum-weight edges queries)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```