Problem 3515: Shortest Path in a Weighted Tree

Problem Information

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

Problem Description

You are given an integer

n

and an undirected, weighted tree rooted at node 1 with

n

nodes numbered from 1 to

n

. This is represented by a 2D array

edges

of length

n - 1

, where

edges[i] = [u

i

, v

i

, w

i

]

indicates an undirected edge from node

u

i

to

v

i

with weight

w

i

.

You are also given a 2D integer array

queries

of length

q

, where each

queries[i]

is either:

[1, u, v, w']

–

Update

the weight of the edge between nodes

u

and

v

to

w'

, where

(u, v)

is guaranteed to be an edge present in

edges

.

[2, x]

–

Compute

the

shortest

path distance from the root node 1 to node

x

.

Return an integer array

answer

, where

answer[i]

is the

shortest

path distance from node 1 to

x

for the

i

th

query of

[2, x]

.

Example 1:

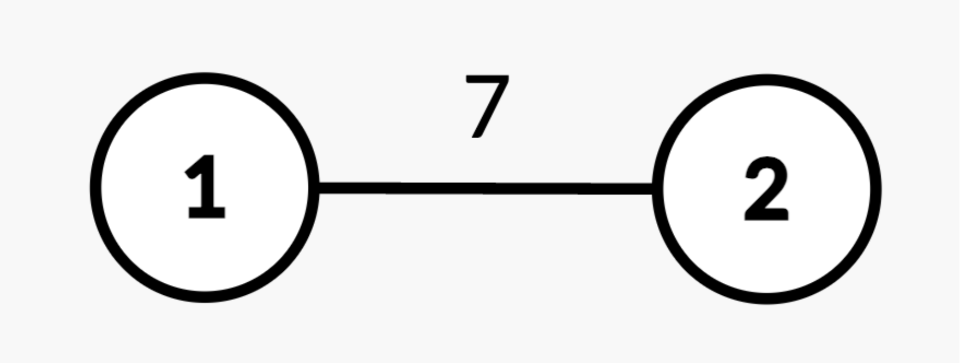
Input:

n = 2, edges = [[1,2,7]], queries = [[2,2],[1,1,2,4],[2,2]]

Output:

[7,4]

Explanation:



Query

[2,2]

: The shortest path from root node 1 to node 2 is 7.

Query

[1,1,2,4]

: The weight of edge

(1,2)

changes from 7 to 4.

Query

[2,2]

: The shortest path from root node 1 to node 2 is 4.

Example 2:

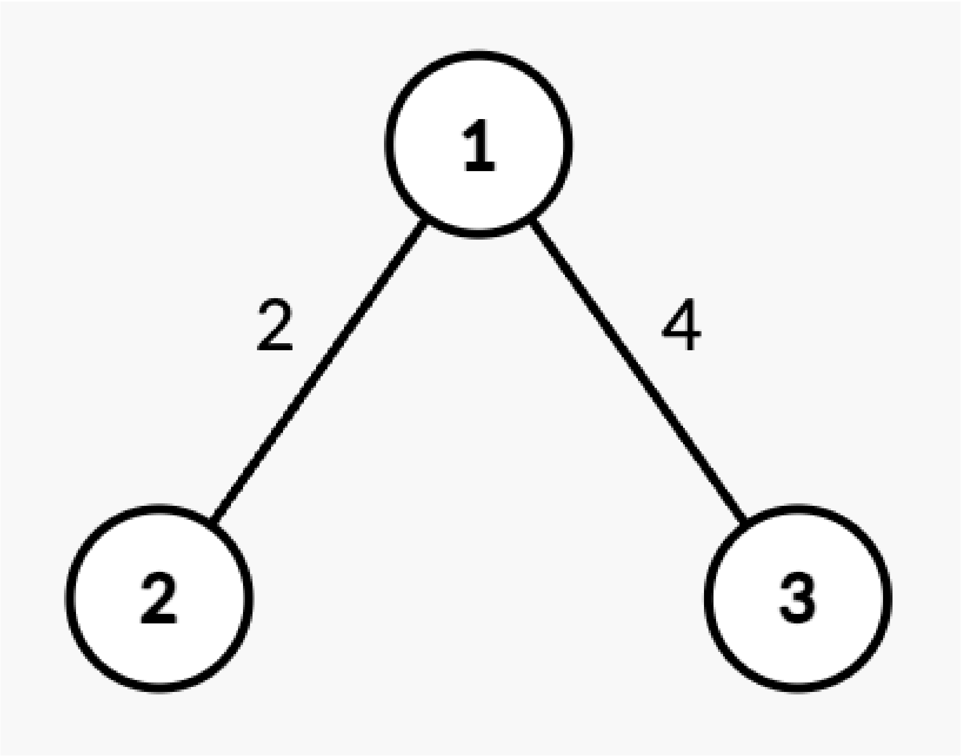
Input:

n = 3, edges = [[1,2,2],[1,3,4]], queries = [[2,1],[2,3],[1,1,3,7],[2,2],[2,3]]

Output:

[0,4,2,7]

Explanation:



Query

[2,1]

: The shortest path from root node 1 to node 1 is 0.

Query

[2,3]

: The shortest path from root node 1 to node 3 is 4.

Query

[1,1,3,7]

: The weight of edge

(1,3)

changes from 4 to 7.

Query

[2,2]

: The shortest path from root node 1 to node 2 is 2.

Query

[2,3]

: The shortest path from root node 1 to node 3 is 7.

Example 3:

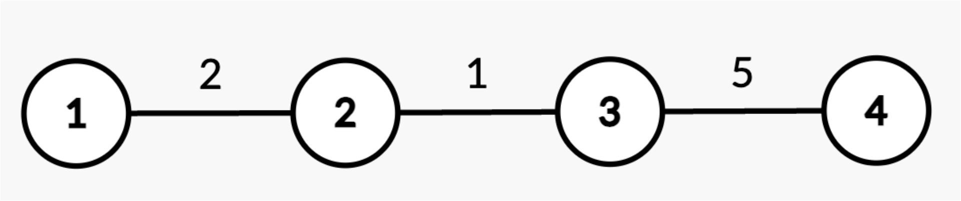
Input:

n = 4, edges = [[1,2,2],[2,3,1],[3,4,5]], queries = [[2,4],[2,3],[1,2,3,3],[2,2],[2,3]]

Output:

[8,3,2,5]

Explanation:



Query

[2,4]

: The shortest path from root node 1 to node 4 consists of edges

(1,2)

,

(2,3)

, and

(3,4)

with weights

2 + 1 + 5 = 8

.

Query

[2,3]

: The shortest path from root node 1 to node 3 consists of edges

(1,2)

and

(2,3)

with weights

2 + 1 = 3

.

Query

[1,2,3,3]

: The weight of edge

(2,3)

changes from 1 to 3.

Query

[2,2]

: The shortest path from root node 1 to node 2 is 2.

Query

[2,3]

: The shortest path from root node 1 to node 3 consists of edges

(1,2)

and

(2,3)

with updated weights

2 + 3 = 5

.

Constraints:

1 <= n <= 10

5

edges.length == n - 1

edges[i] == [u

i

, v

i

, w

i

]

1 <= u

i

, v

i

<= n

1 <= w

i

<= 10

4

The input is generated such that

edges

represents a valid tree.

1 <= queries.length == q <= 10

5

queries[i].length == 2

or

4

queries[i] == [1, u, v, w']

or,

queries[i] == [2, x]

1 <= u, v, x <= n

(u, v)

is always an edge from

edges

.

1 <= w' <= 10

4

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

```
func treeQueries(n int, edges [][]int, queries [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun treeQueries(n: Int, edges: Array<IntArray>, queries: Array<IntArray>):
IntArray {


}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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


}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (tree-queries n edges queries)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest Path in a Weighted Tree
* Difficulty: Hard
* Tags: array, tree, graph, 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> treeQueries(int n, vector<vector<int>>& edges,
vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Shortest Path in a Weighted Tree
* Difficulty: Hard
* Tags: array, tree, graph, 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[] treeQueries(int n, int[][] edges, int[][] queries) {
}
}
```

Python3 Solution:

```
"""
Problem: Shortest Path in a Weighted Tree
Difficulty: Hard
Tags: array, tree, graph, 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 treeQueries(self, n: int, edges: List[List[int]], queries:
List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Shortest Path in a Weighted Tree
* Difficulty: Hard
* Tags: array, tree, graph, 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} n
* @param {number[][]} edges
* @param {number[][]} queries
* @return {number[]}
*/
var treeQueries = function(n, edges, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Shortest Path in a Weighted Tree
* Difficulty: Hard
* Tags: array, tree, graph, 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 treeQueries(n: number, edges: number[][], queries: number[][]):
number[] {

};
```

C# Solution:

```
/*
* Problem: Shortest Path in a Weighted Tree
* Difficulty: Hard
* Tags: array, tree, graph, 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[] TreeQueries(int n, int[][] edges, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Shortest Path in a Weighted Tree
* Difficulty: Hard
* Tags: array, tree, graph, 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* treeQueries(int n, int** edges, int edgesSize, int* edgesColSize, int**
queries, int queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Shortest Path in a Weighted Tree
// Difficulty: Hard
// Tags: array, tree, graph, 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 treeQueries(n int, edges [][]int, queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun treeQueries(n: Int, edges: Array<IntArray>, queries: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Shortest Path in a Weighted Tree
// Difficulty: Hard
// Tags: array, tree, graph, 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 tree_queries(n: i32, edges: Vec<Vec<i32>>, queries: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (tree-queries n edges queries)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) (listof exact-integer?))
)
```