Problem 2277: Closest Node to Path in Tree

Problem Information

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

Problem Description

You are given a positive integer

n

representing the number of nodes in a tree, numbered from

0

to

n - 1

(

inclusive

). You are also given a 2D integer array

edges

of length

n - 1

, where

edges[i] = [node1

i

, node2

i

]

denotes that there is a

bidirectional

edge connecting

node1

i

and

node2

i

in the tree.

You are given a

0-indexed

integer array

query

of length

m

where

query[i] = [start

i

, end

i

, node

i

]

means that for the

i

th

query, you are tasked with finding the node on the path from

start

i

to

end

i

that is

closest

to

node

i

.

Return

an integer array

answer

of length

m

, where

answer[i]

is the answer to the

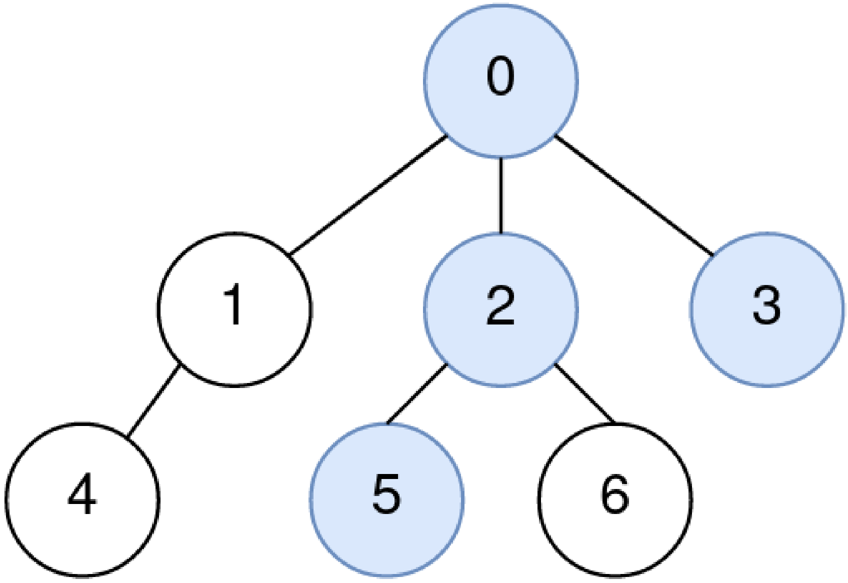
i

th

query

.

Example 1:



Input:

n = 7, edges = [[0,1],[0,2],[0,3],[1,4],[2,5],[2,6]], query = [[5,3,4],[5,3,6]]

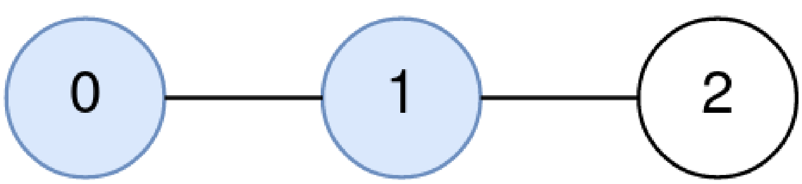
Output:

[0,2]

Explanation:

The path from node 5 to node 3 consists of the nodes 5, 2, 0, and 3. The distance between node 4 and node 0 is 2. Node 0 is the node on the path closest to node 4, so the answer to the first query is 0. The distance between node 6 and node 2 is 1. Node 2 is the node on the path closest to node 6, so the answer to the second query is 2.

Example 2:



Input:

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

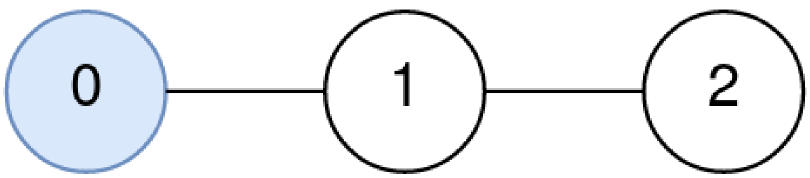
Output:

[1]

Explanation:

The path from node 0 to node 1 consists of the nodes 0, 1. The distance between node 2 and node 1 is 1. Node 1 is the node on the path closest to node 2, so the answer to the first query is 1.

Example 3:



Input:

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

Output:

[0]

Explanation:

The path from node 0 to node 0 consists of the node 0. Since 0 is the only node on the path, the answer to the first query is 0.

Constraints:

1 <= n <= 1000

edges.length == n - 1

edges[i].length == 2

0 <= node1

i

, node2

i

<= n - 1

node1

i

!= node2

i

1 <= query.length <= 1000

query[i].length == 3

0 <= start

i

, end

i

, node

i

<= n - 1

The graph is a tree.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int[] closestNode(int n, int[][] edges, int[][] query) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function closestNode(n: number, edges: number[][], query: number[][]):
number[] {

};
```

C#:

```
public class Solution {
public int[] ClosestNode(int n, int[][] edges, int[][] query) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* closestNode(int n, int** edges, int edgesSize, int* edgesColSize, int**
query, int querySize, int* queryColSize, int* returnSize) {

}
```

Go:

```
func closestNode(n int, edges [][]int, query [][]int) []int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
```

```
* @param Integer[][] $query
* @return Integer[]
*/
function closestNode($n, $edges, $query) {

}
}
```

Dart:

```
class Solution {
List<int> closestNode(int n, List<List<int>> edges, List<List<int>> query) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec closest_node(N :: integer(), Edges :: [[integer()]], Query ::
[[integer()]]) -> [integer()].
closest_node(N, Edges, Query) ->
.
```

Racket:

```
(define/contract (closest-node n edges query)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Closest Node to Path in 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> closestNode(int n, vector<vector<int>>& edges,
vector<vector<int>>& query) {

}
};
```

Java Solution:

```
/**
* Problem: Closest Node to Path in 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[] closestNode(int n, int[][] edges, int[][] query) {

}
}
```

Python3 Solution:

```
"""
Problem: Closest Node to Path in 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 closestNode(self, n: int, edges: List[List[int]], query: List[List[int]])
-> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Closest Node to Path in 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[][]} query
* @return {number[]}
*/
var closestNode = function(n, edges, query) {

};
```

TypeScript Solution:

```
/**
* Problem: Closest Node to Path in 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 closestNode(n: number, edges: number[][], query: number[][]):
number[] {

};
```

C# Solution:

```
/*
* Problem: Closest Node to Path in 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[] ClosestNode(int n, int[][] edges, int[][] query) {

}
}
```

C Solution:

```
/*
* Problem: Closest Node to Path in 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* closestNode(int n, int** edges, int edgesSize, int* edgesColSize, int**
query, int querySize, int* queryColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Closest Node to Path in 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 closestNode(n int, edges [][]int, query [][]int) []int {
```

```
}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Closest Node to Path in 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 closest_node(n: i32, edges: Vec<Vec<i32>>, query: Vec<Vec<i32>>) ->
Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
```

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
List<int> closestNode(int n, List<List<int>> edges, List<List<int>> query) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec closest_node(N :: integer(), Edges :: [[integer()]], Query ::
[[integer()]]) -> [integer()].
closest_node(N, Edges, Query) ->
.
```

Racket Solution:

```
(define/contract (closest-node n edges query)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) (listof exact-integer?))
)
```