Problem 3585: Find Weighted Median Node in 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 0 with

n

nodes numbered from 0 to

n - 1

. This is represented by a 2D array

edges

of length

n - 1

, where

edges[i] = [u

i

, v

i

, w

i

]

indicates an edge from node

u

i

to

v

i

with weight

w

i

.

The

weighted median node

is defined as the

first

node

x

on the path from

u

i

to

v

i

such that the sum of edge weights from

u

i

to

x

is

greater than or equal to half

of the total path weight.

You are given a 2D integer array

queries

. For each

queries[j] = [u

j

, v

j

]

, determine the weighted median node along the path from

u

j

to

v

j

.

Return an array

ans

, where

ans[j]

is the node index of the weighted median for

queries[j]

.

Example 1:

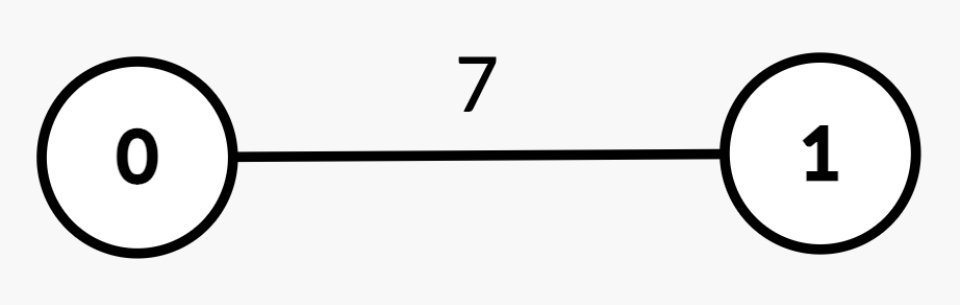
Input:

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

Output:

[0,1]

Explanation:



Query

Path

Edge

Weights

Total

Path

Weight

Half

Explanation

Answer

[1, 0]

$1 \rightarrow 0$

[7]

7

3.5

Sum from

$1 \rightarrow 0 = 7 >= 3.5$

, median is node 0.

0

[0, 1]

$0 \rightarrow 1$

[7]

7

3.5

Sum from

$0 \rightarrow 1 = 7 >= 3.5$

, median is node 1.

1

Example 2:

Input:

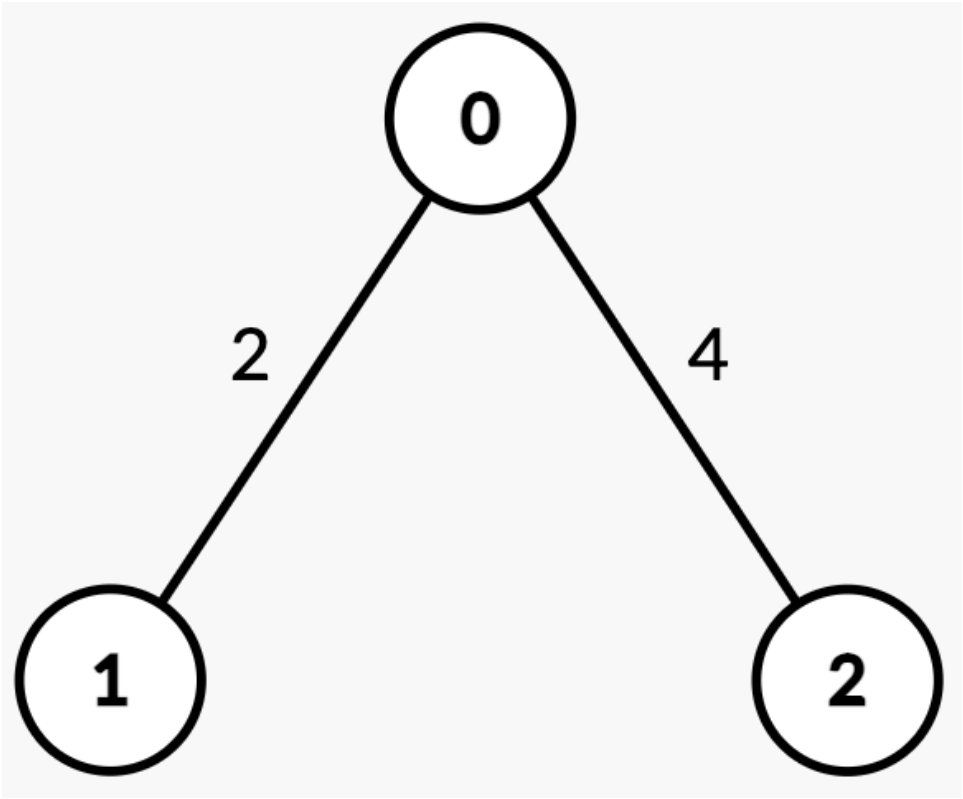
n = 3, edges = [[0,1,2],[2,0,4]], queries = [[0,1],[2,0],[1,2]]

Output:

[1,0,2]

E

xplanation:



Query

Path

Edge

Weights

Total

Path

Weight

Half

Explanation

Answer

[0, 1]

$0 \rightarrow 1$

[2]

2

1

Sum from

$0 \rightarrow 1 = 2 >= 1$

, median is node 1.

1

[2, 0]

$2 \rightarrow 0$

[4]

4

2

Sum from

$2 \rightarrow 0 = 4 >= 2$

, median is node 0.

0

[1, 2]

$1 \rightarrow 0 \rightarrow 2$

[2, 4]

6

3

Sum from

$1 \rightarrow 0 = 2 < 3$

.

Sum from

$1 \rightarrow 2 = 2 + 4 = 6 >= 3$

, median is node 2.

2

Example 3:

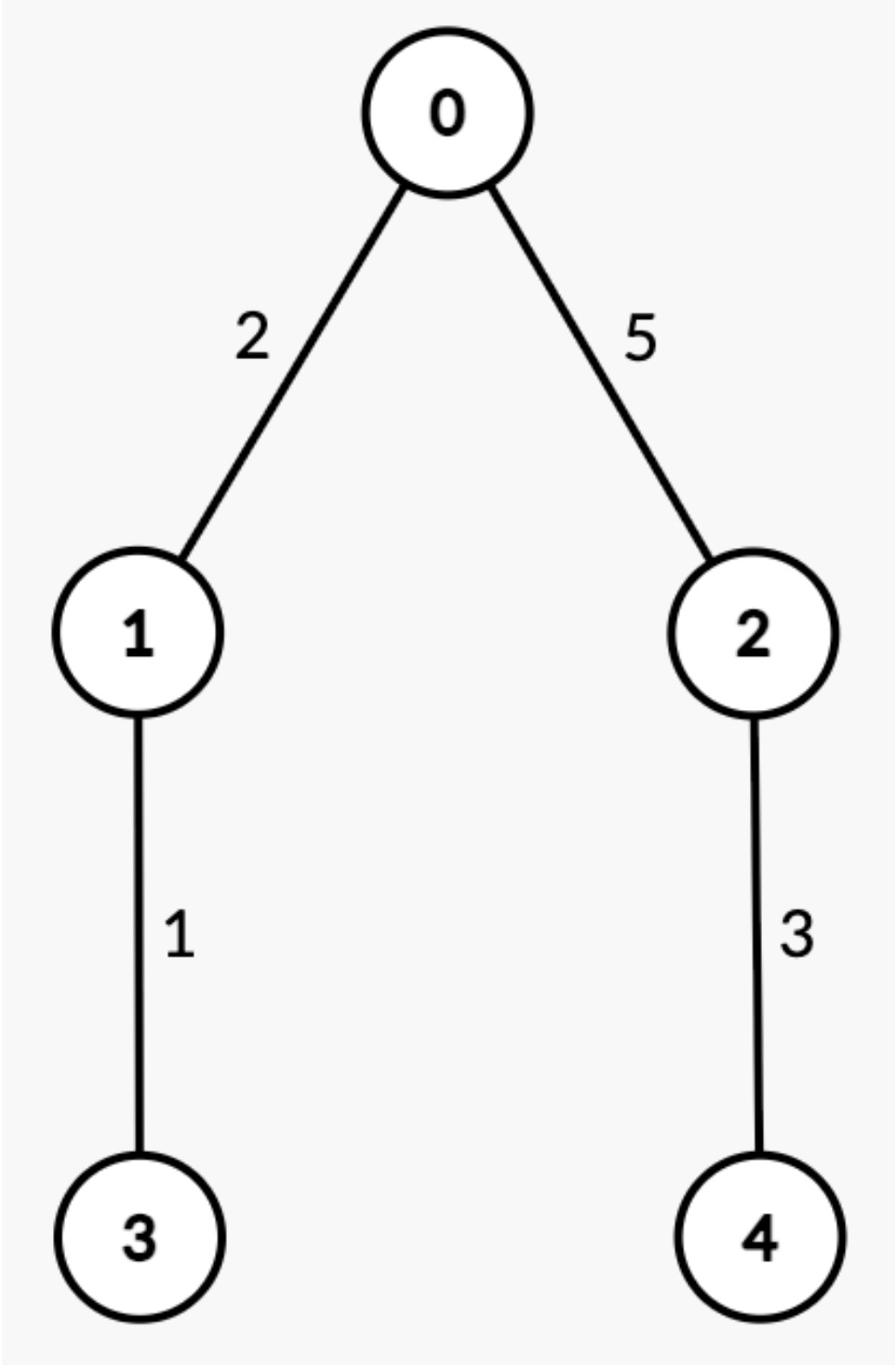
Input:

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

Output:

[2,2]

Explanation:



Query

Path

Edge

Weights

Total

Path

Weight

Half

Explanation

Answer

[3, 4]

$3 \rightarrow 1 \rightarrow 0 \rightarrow 2 \rightarrow 4$

[1, 2, 5, 3]

11

5.5

Sum from

$3 \rightarrow 1 = 1 < 5.5$

.

Sum from

$3 \rightarrow 0 = 1 + 2 = 3 < 5.5$

.

Sum from

$3 \rightarrow 2 = 1 + 2 + 5 = 8 >= 5.5$

, median is node 2.

2

[1, 2]

$1 \rightarrow 0 \rightarrow 2$

[2, 5]

7

3.5

Sum from

$1 \rightarrow 0 = 2 < 3.5$

.

Sum from

$1 \rightarrow 2 = 2 + 5 = 7 >= 3.5$

, median is node 2.

2

Constraints:

2 <= n <= 10

5

edges.length == n - 1

edges[i] == [u

i

, v

i

, w

i

]

0 <= u

i

, v

i

< n

1 <= w

i

<= 10

9

1 <= queries.length <= 10

5

queries[j] == [u

j

, v

j

]

0 <= u

j

, v

j

< n

The input is generated such that

edges

represents a valid tree.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def findMedian(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 findMedian = function(n, edges, queries) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn find_median(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 find_median(n, edges, queries)

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Find Weighted Median Node in Tree
* 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:
vector<int> findMedian(int n, vector<vector<int>>& edges,
vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Find Weighted Median Node in Tree
* 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 int[] findMedian(int n, int[][] edges, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Weighted Median Node in Tree
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 findMedian(self, n: int, edges: List[List[int]], queries:
List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Weighted Median Node in Tree
* 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[][]} queries
* @return {number[]}
*/
var findMedian = function(n, edges, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Weighted Median Node in Tree
* 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 findMedian(n: number, edges: number[][], queries: number[][]):
number[] {

};
```

C# Solution:

```
/*
* Problem: Find Weighted Median Node in Tree
* 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 int[] FindMedian(int n, int[][] edges, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Find Weighted Median Node in Tree
* 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
*/

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

}
```

Go Solution:

```
// Problem: Find Weighted Median Node in Tree
// 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 findMedian(n int, edges [][]int, queries [][]int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find Weighted Median Node in Tree
// 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 find_median(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 find_median(n, edges, queries)

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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