Problem 2476: Closest Nodes Queries in a Binary Search Tree

Problem Information

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

Problem Description

You are given the

root

of a

binary search tree

and an array

queries

of size

n

consisting of positive integers.

Find a

2D

array

answer

of size

n

where

answer[i] = [min

i

, max

i

]

:

min

i

is the

largest

value in the tree that is smaller than or equal to

queries[i]

. If a such value does not exist, add

-1

instead.

max

i

is the

smallest

value in the tree that is greater than or equal to

queries[i]

. If a such value does not exist, add

-1

instead.

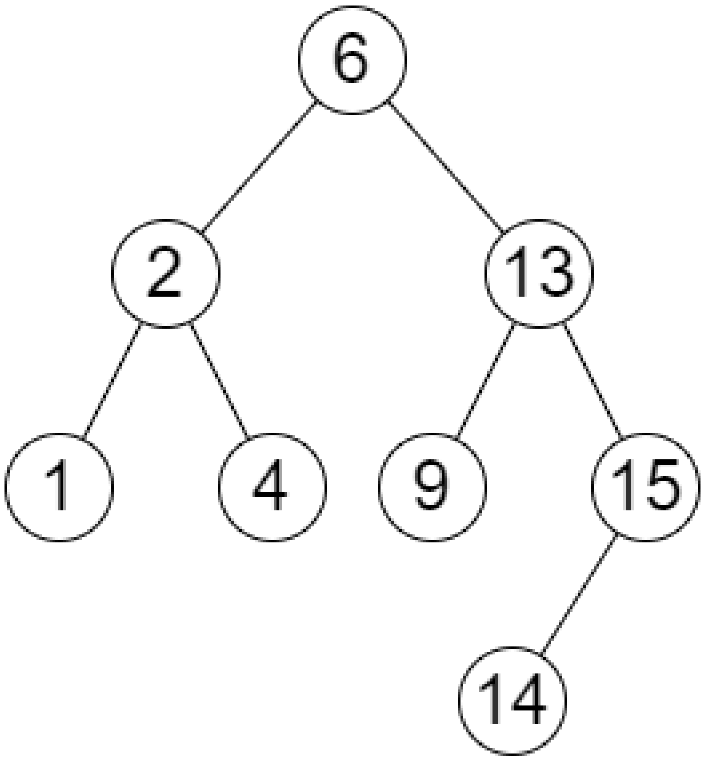
Return

the array

answer

.

Example 1:



Input:

root = [6,2,13,1,4,9,15,null,null,null,null,null,null,14], queries = [2,5,16]

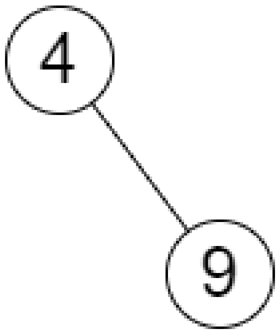
Output:

[[2,2],[4,6],[15,-1]]

Explanation:

We answer the queries in the following way: - The largest number that is smaller or equal than 2 in the tree is 2, and the smallest number that is greater or equal than 2 is still 2. So the answer for the first query is [2,2]. - The largest number that is smaller or equal than 5 in the tree is 4, and the smallest number that is greater or equal than 5 is 6. So the answer for the second query is [4,6]. - The largest number that is smaller or equal than 16 in the tree is 15, and the smallest number that is greater or equal than 16 does not exist. So the answer for the third query is [15,-1].

Example 2:



Input:

root = [4,null,9], queries = [3]

Output:

[[-1,4]]

Explanation:

The largest number that is smaller or equal to 3 in the tree does not exist, and the smallest number that is greater or equal to 3 is 4. So the answer for the query is [-1,4].

Constraints:

The number of nodes in the tree is in the range

[2, 10

5

]

.

1 <= Node.val <= 10

6

n == queries.length

1 <= n <= 10

5

1 <= queries[i] <= 10

6

Code Snippets

C++:

```
/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* TreeNode *left;
* TreeNode *right;
* TreeNode() : val(0), left(nullptr), right(nullptr) {}
* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
* TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
* };
*/
class Solution {
public:
vector<vector<int>> closestNodes(TreeNode* root, vector<int>& queries) {

}
};
```

Java:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* int val;
* TreeNode left;
* TreeNode right;
* TreeNode() {}
* TreeNode(int val) { this.val = val; }
* TreeNode(int val, TreeNode left, TreeNode right) {
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
class Solution {
public List<List<Integer>> closestNodes(TreeNode root, List<Integer> queries)
{

}
}
```

Python3:

```
# Definition for a binary tree node.
# class TreeNode:
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
# self.right = right
class Solution:
def closestNodes(self, root: Optional[TreeNode], queries: List[int]) ->
List[List[int]]:
```

Python:

```
# Definition for a binary tree node.
# class TreeNode(object):
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
# self.right = right
class Solution(object):
def closestNodes(self, root, queries):
"""
:type root: Optional[TreeNode]
:type queries: List[int]
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* Definition for a binary tree node.
* function TreeNode(val, left, right) {
* this.val = (val===undefined ? 0 : val)
* this.left = (left===undefined ? null : left)
* this.right = (right===undefined ? null : right)
* }
*/
/**
* @param {TreeNode} root
* @param {number[]} queries
* @return {number[][]}
*/
var closestNodes = function(root, queries) {

};
```

TypeScript:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* val: number
* left: TreeNode | null
* right: TreeNode | null
* constructor(val?: number, left?: TreeNode | null, right?: TreeNode | null)
{
* this.val = (val===undefined ? 0 : val)
* this.left = (left===undefined ? null : left)
* this.right = (right===undefined ? null : right)
* }
* }
*/

function closestNodes(root: TreeNode | null, queries: number[]): number[][] {

};
```

C#:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* public int val;
* public TreeNode left;
* public TreeNode right;
* public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
public class Solution {
public IList<IList<int>> ClosestNodes(TreeNode root, IList<int> queries) {

}
}
```

C:

```
/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* struct TreeNode *left;
* struct TreeNode *right;
* };
*/
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** closestNodes(struct TreeNode* root, int* queries, int queriesSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go:

```
/**
* Definition for a binary tree node.
* type TreeNode struct {
* Val int
* Left *TreeNode
* Right *TreeNode
* }
*/
func closestNodes(root *TreeNode, queries []int) [][]int {

}
```

Kotlin:

```
/**
* Example:
* var ti = TreeNode(5)
* var v = ti.`val`
* Definition for a binary tree node.
* class TreeNode(var `val`: Int) {
* var left: TreeNode? = null
* var right: TreeNode? = null
* }
*/
class Solution {
fun closestNodes(root: TreeNode?, queries: List<Int>): List<List<Int>> {

}
}
```

Swift:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* public var val: Int
* public var left: TreeNode?
* public var right: TreeNode?
* public init() { self.val = 0; self.left = nil; self.right = nil; }
* public init(_ val: Int) { self.val = val; self.left = nil; self.right =
nil; }
* public init(_ val: Int, _ left: TreeNode?, _ right: TreeNode?) {
* self.val = val
* self.left = left
* self.right = right
* }
* }
*/
class Solution {
func closestNodes(_ root: TreeNode?, _ queries: [Int]) -> [[Int]] {

}
}
```

Rust:

```
// Definition for a binary tree node.
// #[derive(Debug, PartialEq, Eq)]
// pub struct TreeNode {
// pub val: i32,
// pub left: Option<Rc<RefCell<TreeNode>>>,
// pub right: Option<Rc<RefCell<TreeNode>>>,
// }
//
// impl TreeNode {
// #[inline]
// pub fn new(val: i32) -> Self {
// TreeNode {
// val,
// left: None,
// right: None
// }
// }
// }
use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
pub fn closest_nodes(root: Option<Rc<RefCell<TreeNode>>>, queries: Vec<i32>)
-> Vec<Vec<i32>> {

}
}
```

Ruby:

```ruby
# Definition for a binary tree node.
# class TreeNode
# attr_accessor :val, :left, :right
# def initialize(val = 0, left = nil, right = nil)
# @val = val
# @left = left
# @right = right
# end
# end
# @param {TreeNode} root
# @param {Integer[]} queries
# @return {Integer[][]}
def closest_nodes(root, queries)

end
```

PHP:

```php
/**
* Definition for a binary tree node.
* class TreeNode {
* public $val = null;
* public $left = null;
* public $right = null;
* function __construct($val = 0, $left = null, $right = null) {
* $this->val = $val;
* $this->left = $left;
* $this->right = $right;
* }
* }
*/
class Solution {

/**
* @param TreeNode $root
* @param Integer[] $queries
* @return Integer[][]
*/
function closestNodes($root, $queries) {

}
}
```

Dart:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* int val;
* TreeNode? left;
* TreeNode? right;
* TreeNode([this.val = 0, this.left, this.right]);
* }
*/
class Solution {
List<List<int>> closestNodes(TreeNode? root, List<int> queries) {

}
}
```

Scala:

```
/**
* Definition for a binary tree node.
* class TreeNode(_value: Int = 0, _left: TreeNode = null, _right: TreeNode =
null) {
* var value: Int = _value
* var left: TreeNode = _left
* var right: TreeNode = _right
* }
*/
object Solution {
def closestNodes(root: TreeNode, queries: List[Int]): List[List[Int]] = {

}
}
```

Elixir:

```
# Definition for a binary tree node.
#
# defmodule TreeNode do
# @type t :: %__MODULE__{
# val: integer,
# left: TreeNode.t() | nil,
# right: TreeNode.t() | nil
# }
# defstruct val: 0, left: nil, right: nil
# end

defmodule Solution do
@spec closest_nodes(root :: TreeNode.t | nil, queries :: [integer]) ::
[[integer]]
def closest_nodes(root, queries) do

end
end
```

Erlang:

```
%% Definition for a binary tree node.
%%
%% -record(tree_node, {val = 0 :: integer(),
%% left = null :: 'null' | #tree_node{},
%% right = null :: 'null' | #tree_node{}}).

-spec closest_nodes(Root :: #tree_node{} | null, Queries :: [integer()]) ->
[[integer()]].
closest_nodes(Root, Queries) ->
.
```

Racket:

```
; Definition for a binary tree node.
#|

; val : integer?
; left : (or/c tree-node? #f)
; right : (or/c tree-node? #f)
(struct tree-node
(val left right) #:mutable #:transparent)

; constructor
(define (make-tree-node [val 0])
(tree-node val #f #f))

|#
```

```
(define/contract (closest-nodes root queries)
(-> (or/c tree-node? #f) (listof exact-integer?) (listof (listof
exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Closest Nodes Queries in a Binary Search Tree
* Difficulty: Medium
* 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
*/

/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* TreeNode *left;
* TreeNode *right;
* TreeNode() : val(0), left(nullptr), right(nullptr) {}
* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
* TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
* };
*/
class Solution {
public:
vector<vector<int>> closestNodes(TreeNode* root, vector<int>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Closest Nodes Queries in a Binary Search Tree
* Difficulty: Medium
* 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
*/

/**
* Definition for a binary tree node.
* public class TreeNode {
* int val;
* TreeNode left;
* TreeNode right;
* TreeNode() {}
* TreeNode(int val) { this.val = val; }
* TreeNode(int val, TreeNode left, TreeNode right) {
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
class Solution {
public List<List<Integer>> closestNodes(TreeNode root, List<Integer> queries)
{

}
}
```

Python3 Solution:

```
"""
Problem: Closest Nodes Queries in a Binary Search Tree
Difficulty: Medium
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
```

```
"""

# Definition for a binary tree node.
# class TreeNode:
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
# self.right = right
class Solution:
def closestNodes(self, root: Optional[TreeNode], queries: List[int]) ->
List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
# Definition for a binary tree node.
# class TreeNode(object):
# def __init__(self, val=0, left=None, right=None):
# self.val = val
# self.left = left
# self.right = right
class Solution(object):
def closestNodes(self, root, queries):
"""
:type root: Optional[TreeNode]
:type queries: List[int]
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Closest Nodes Queries in a Binary Search Tree
* Difficulty: Medium
* 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
*/
```

```
/**
* Definition for a binary tree node.
* function TreeNode(val, left, right) {
* this.val = (val===undefined ? 0 : val)
* this.left = (left===undefined ? null : left)
* this.right = (right===undefined ? null : right)
* }
*/
/**
* @param {TreeNode} root
* @param {number[]} queries
* @return {number[][]}
*/
var closestNodes = function(root, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Closest Nodes Queries in a Binary Search Tree
* Difficulty: Medium
* 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
*/

/**
* Definition for a binary tree node.
* class TreeNode {
* val: number
* left: TreeNode | null
* right: TreeNode | null
* constructor(val?: number, left?: TreeNode | null, right?: TreeNode | null)
{
* this.val = (val===undefined ? 0 : val)
* this.left = (left===undefined ? null : left)
* this.right = (right===undefined ? null : right)
* }
* }
*/

function closestNodes(root: TreeNode | null, queries: number[]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Closest Nodes Queries in a Binary Search Tree
* Difficulty: Medium
* 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
*/

/**
* Definition for a binary tree node.
* public class TreeNode {
* public int val;
* public TreeNode left;
* public TreeNode right;
* public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
public class Solution {
public IList<IList<int>> ClosestNodes(TreeNode root, IList<int> queries) {

}
}
```

C Solution:

```
/*
* Problem: Closest Nodes Queries in a Binary Search Tree
* Difficulty: Medium
* 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
*/

/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* struct TreeNode *left;
* struct TreeNode *right;
* };
*/
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** closestNodes(struct TreeNode* root, int* queries, int queriesSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Closest Nodes Queries in a Binary Search Tree
// Difficulty: Medium
// 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

/**
* Definition for a binary tree node.
* type TreeNode struct {
```

```
* Val int
* Left *TreeNode
* Right *TreeNode
* }
*/
func closestNodes(root *TreeNode, queries []int) [][]int {

}
```

Kotlin Solution:

```
/**
* Example:
* var ti = TreeNode(5)
* var v = ti.`val`
* Definition for a binary tree node.
* class TreeNode(var `val`: Int) {
* var left: TreeNode? = null
* var right: TreeNode? = null
* }
*/
class Solution {
fun closestNodes(root: TreeNode?, queries: List<Int>): List<List<Int>> {

}
}
```

Swift Solution:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* public var val: Int
* public var left: TreeNode?
* public var right: TreeNode?
* public init() { self.val = 0; self.left = nil; self.right = nil; }
* public init(_ val: Int) { self.val = val; self.left = nil; self.right =
nil; }
* public init(_ val: Int, _ left: TreeNode?, _ right: TreeNode?) {
* self.val = val
* self.left = left
* self.right = right
* }
* }
*/
class Solution {
func closestNodes(_ root: TreeNode?, _ queries: [Int]) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Closest Nodes Queries in a Binary Search Tree
// Difficulty: Medium
// 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

// Definition for a binary tree node.
// #[derive(Debug, PartialEq, Eq)]
// pub struct TreeNode {
// pub val: i32,
// pub left: Option<Rc<RefCell<TreeNode>>>,
// pub right: Option<Rc<RefCell<TreeNode>>>,
// }
//
// impl TreeNode {
// #[inline]
// pub fn new(val: i32) -> Self {
// TreeNode {
// val,
// left: None,
// right: None
// }
// }
// }
use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
```

```
pub fn closest_nodes(root: Option<Rc<RefCell<TreeNode>>>, queries: Vec<i32>)
-> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# Definition for a binary tree node.
# class TreeNode
# attr_accessor :val, :left, :right
# def initialize(val = 0, left = nil, right = nil)
# @val = val
# @left = left
# @right = right
# end
# end
# @param {TreeNode} root
# @param {Integer[]} queries
# @return {Integer[][]}
def closest_nodes(root, queries)

end
```

PHP Solution:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* public $val = null;
* public $left = null;
* public $right = null;
* function __construct($val = 0, $left = null, $right = null) {
* $this->val = $val;
* $this->left = $left;
* $this->right = $right;
* }
* }
*/
class Solution {
```

```
/**
* @param TreeNode $root
* @param Integer[] $queries
* @return Integer[][]
*/
function closestNodes($root, $queries) {

}
}
```

Dart Solution:

```
/**
* Definition for a binary tree node.
* class TreeNode {
* int val;
* TreeNode? left;
* TreeNode? right;
* TreeNode([this.val = 0, this.left, this.right]);
* }
*/
class Solution {
List<List<int>> closestNodes(TreeNode? root, List<int> queries) {

}
}
```

Scala Solution:

```
/**
* Definition for a binary tree node.
* class TreeNode(_value: Int = 0, _left: TreeNode = null, _right: TreeNode =
null) {
* var value: Int = _value
* var left: TreeNode = _left
* var right: TreeNode = _right
* }
*/
object Solution {
def closestNodes(root: TreeNode, queries: List[Int]): List[List[Int]] = {
}
}
```

Elixir Solution:

```
# Definition for a binary tree node.
#
# defmodule TreeNode do
# @type t :: %__MODULE__{
# val: integer,
# left: TreeNode.t() | nil,
# right: TreeNode.t() | nil
# }
# defstruct val: 0, left: nil, right: nil
# end

defmodule Solution do
@spec closest_nodes(root :: TreeNode.t | nil, queries :: [integer]) ::
[[integer]]
def closest_nodes(root, queries) do

end
end
```

Erlang Solution:

```
%% Definition for a binary tree node.
%%
%% -record(tree_node, {val = 0 :: integer(),
%% left = null :: 'null' | #tree_node{},
%% right = null :: 'null' | #tree_node{}}).

-spec closest_nodes(Root :: #tree_node{} | null, Queries :: [integer()]) ->
[[integer()]].
closest_nodes(Root, Queries) ->
.
```

Racket Solution:

```
; Definition for a binary tree node.
#|
; val : integer?
; left : (or/c tree-node? #f)
; right : (or/c tree-node? #f)
(struct tree-node
(val left right) #:mutable #:transparent)

; constructor
(define (make-tree-node [val 0])
(tree-node val #f #f))

|#

(define/contract (closest-nodes root queries)
(-> (or/c tree-node? #f) (listof exact-integer?) (listof (listof
exact-integer?)))
)
```