Problem 3532: Path Existence Queries in a Graph I

Problem Information

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

Problem Description

You are given an integer

n

representing the number of nodes in a graph, labeled from 0 to

n - 1

.

You are also given an integer array

nums

of length

n

sorted in

non-decreasing

order, and an integer

maxDiff

.

An undirected edge exists between nodes i and j if the absolute difference between nums[i] and nums[j] is at most maxDiff (i.e., |nums[i] - nums[j]| <= maxDiff).

You are also given a 2D integer array

queries

. For each

queries[i] = [u

i

, v

i

]

, determine whether there exists a path between nodes

u

i

and

v

i

.

Return a boolean array

answer

, where

answer[i]

is

true

if there exists a path between

u

i

and

v

i

in the

i

th

query and

false

otherwise.

Example 1:

Input:

n = 2, nums = [1,3], maxDiff = 1, queries = [[0,0],[0,1]]

Output:

[true,false]

Explanation:

Query

[0,0]

: Node 0 has a trivial path to itself.

Query

[0,1]

: There is no edge between Node 0 and Node 1 because

|nums[0] - nums[1]| = |1 - 3| = 2

, which is greater than

maxDiff

.

Thus, the final answer after processing all the queries is

[true, false]

.

Example 2:

Input:

n = 4, nums = [2,5,6,8], maxDiff = 2, queries = [[0,1],[0,2],[1,3],[2,3]]

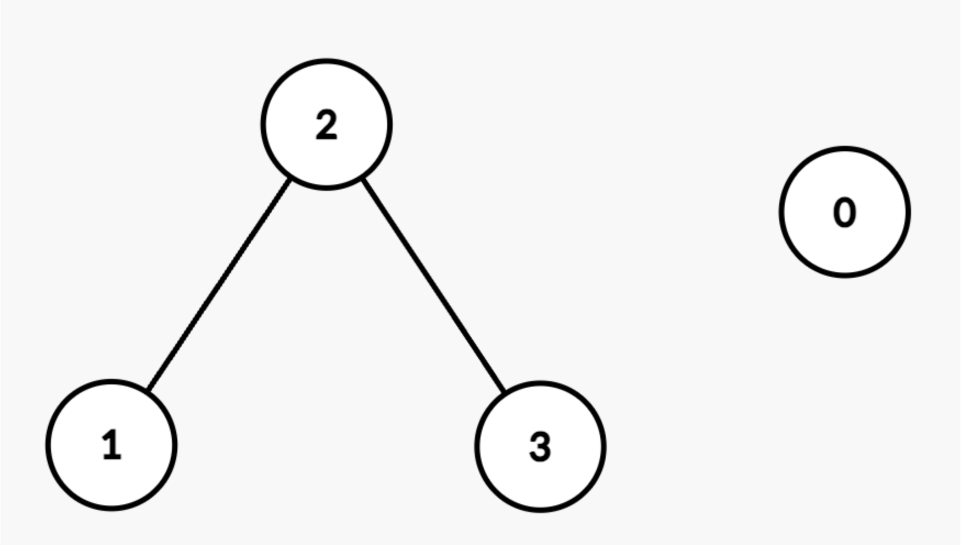
Output:

[false,false,true,true]

Explanation:

The resulting graph is:

,



Query

[0,1]

: There is no edge between Node 0 and Node 1 because

|nums[0] - nums[1]| = |2 - 5| = 3

, which is greater than

maxDiff

.

Query

[0,2]

: There is no edge between Node 0 and Node 2 because

|nums[0] - nums[2]| = |2 - 6| = 4

, which is greater than

maxDiff

.

Query

[1,3]

: There is a path between Node 1 and Node 3 through Node 2 since

|nums[1] - nums[2]| = |5 - 6| = 1

and

|nums[2] - nums[3]| = |6 - 8| = 2

, both of which are within

maxDiff

.

Query

[2,3]

: There is an edge between Node 2 and Node 3 because

|nums[2] - nums[3]| = |6 - 8| = 2

, which is equal to

maxDiff

.

Thus, the final answer after processing all the queries is

[false, false, true, true]

.

Constraints:

1 <= n == nums.length <= 10

5

0 <= nums[i] <= 10

5

nums

is sorted in

non-decreasing

order.

0 <= maxDiff <= 10

5

1 <= queries.length <= 10

5

queries[i] == [u

i

, v

i

]

0 <= u

i

, v

i

< n

Code Snippets

C++:

```
class Solution {
public:
vector<bool> pathExistenceQueries(int n, vector<int>& nums, int maxDiff,
vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public boolean[] pathExistenceQueries(int n, int[] nums, int maxDiff, int[][]
queries) {

}
}
```

Python3:

```
class Solution:
def pathExistenceQueries(self, n: int, nums: List[int], maxDiff: int,
queries: List[List[int]]) -> List[bool]:
```

Python:

```
class Solution(object):
def pathExistenceQueries(self, n, nums, maxDiff, queries):
"""
:type n: int
```

```
:type nums: List[int]
:type maxDiff: int
:type queries: List[List[int]]
:rtype: List[bool]
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[]} nums
* @param {number} maxDiff
* @param {number[][]} queries
* @return {boolean[]}
*/
var pathExistenceQueries = function(n, nums, maxDiff, queries) {

};
```

TypeScript:

```
function pathExistenceQueries(n: number, nums: number[], maxDiff: number,
queries: number[][]): boolean[] {

};
```

C#:

```
public class Solution {
public bool[] PathExistenceQueries(int n, int[] nums, int maxDiff, int[][]
queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
bool* pathExistenceQueries(int n, int* nums, int numsSize, int maxDiff, int**
queries, int queriesSize, int* queriesColSize, int* returnSize) {
}
```

Go:

```
func pathExistenceQueries(n int, nums []int, maxDiff int, queries [][]int) []bool {

}
```

Kotlin:

```
class Solution {
fun pathExistenceQueries(n: Int, nums: IntArray, maxDiff: Int, queries: Array<IntArray>): BooleanArray {

}
}
```

Swift:

```
class Solution {
func pathExistenceQueries(_ n: Int, _ nums: [Int], _ maxDiff: Int, _ queries: [[Int]]) -> [Bool] {

}
}
```

Rust:

```
impl Solution {
pub fn path_existence_queries(n: i32, nums: Vec<i32>, max_diff: i32, queries: Vec<Vec<i32>>) -> Vec<bool> {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[]} nums
# @param {Integer} max_diff
# @param {Integer[][]} queries
# @return {Boolean[]}
def path_existence_queries(n, nums, max_diff, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $nums
* @param Integer $maxDiff
* @param Integer[][] $queries
* @return Boolean[]
*/
function pathExistenceQueries($n, $nums, $maxDiff, $queries) {

}
}
```

Dart:

```
class Solution {
List<bool> pathExistenceQueries(int n, List<int> nums, int maxDiff,
List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def pathExistenceQueries(n: Int, nums: Array[Int], maxDiff: Int, queries:
Array[Array[Int]]): Array[Boolean] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec path_existence_queries(n :: integer, nums :: [integer], max_diff ::
integer, queries :: [[integer]]) :: [boolean]
def path_existence_queries(n, nums, max_diff, queries) do

end
end
```

Erlang:

```
-spec path_existence_queries(N :: integer(), Nums :: [integer()], MaxDiff ::
integer(), Queries :: [[integer()]]) -> [boolean()].
path_existence_queries(N, Nums, MaxDiff, Queries) ->
.
```

Racket:

```
(define/contract (path-existence-queries n nums maxDiff queries)
(-> exact-integer? (listof exact-integer?) exact-integer? (listof (listof
exact-integer?)) (listof boolean?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Path Existence Queries in a Graph I
* Difficulty: Medium
* Tags: array, graph, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<bool> pathExistenceQueries(int n, vector<int>& nums, int maxDiff,
vector<vector<int>>& queries) {
}
};
```

Java Solution:

```
/**
* Problem: Path Existence Queries in a Graph I
* Difficulty: Medium
* Tags: array, graph, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public boolean[] pathExistenceQueries(int n, int[] nums, int maxDiff, int[][]
queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Path Existence Queries in a Graph I
Difficulty: Medium
Tags: array, graph, hash, sort, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def pathExistenceQueries(self, n: int, nums: List[int], maxDiff: int,
queries: List[List[int]]) -> List[bool]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def pathExistenceQueries(self, n, nums, maxDiff, queries):
"""
:type n: int
:type nums: List[int]
:type maxDiff: int
:type queries: List[List[int]]
:rtype: List[bool]
"""
```

JavaScript Solution:

```
/**
* Problem: Path Existence Queries in a Graph I
* Difficulty: Medium
* Tags: array, graph, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number} n
* @param {number[]} nums
* @param {number} maxDiff
* @param {number[][]} queries
* @return {boolean[]}
*/
var pathExistenceQueries = function(n, nums, maxDiff, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Path Existence Queries in a Graph I
* Difficulty: Medium
* Tags: array, graph, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function pathExistenceQueries(n: number, nums: number[], maxDiff: number,
queries: number[][]): boolean[] {

};
```

C# Solution:

```
/*
* Problem: Path Existence Queries in a Graph I
* Difficulty: Medium
* Tags: array, graph, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public bool[] PathExistenceQueries(int n, int[] nums, int maxDiff, int[][]
queries) {

}
}
```

C Solution:

```
/*
* Problem: Path Existence Queries in a Graph I
* Difficulty: Medium
* Tags: array, graph, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
```

```
*/
bool* pathExistenceQueries(int n, int* nums, int numsSize, int maxDiff, int** queries, int queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Path Existence Queries in a Graph I
// Difficulty: Medium
// Tags: array, graph, hash, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func pathExistenceQueries(n int, nums []int, maxDiff int, queries [][]int) []bool {

}
```

Kotlin Solution:

```
class Solution {
fun pathExistenceQueries(n: Int, nums: IntArray, maxDiff: Int, queries: Array<IntArray>): BooleanArray {

}
}
```

Swift Solution:

```
class Solution {
func pathExistenceQueries(_ n: Int, _ nums: [Int], _ maxDiff: Int, _ queries: [[Int]]) -> [Bool] {

}
}
```

Rust Solution:

```
// Problem: Path Existence Queries in a Graph I
// Difficulty: Medium
// Tags: array, graph, hash, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn path_existence_queries(n: i32, nums: Vec<i32>, max_diff: i32, queries:
Vec<Vec<i32>>) -> Vec<bool> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[]} nums
# @param {Integer} max_diff
# @param {Integer[][]} queries
# @return {Boolean[]}
def path_existence_queries(n, nums, max_diff, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $nums
* @param Integer $maxDiff
* @param Integer[][] $queries
* @return Boolean[]
*/
function pathExistenceQueries($n, $nums, $maxDiff, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<bool> pathExistenceQueries(int n, List<int> nums, int maxDiff,
List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def pathExistenceQueries(n: Int, nums: Array[Int], maxDiff: Int, queries:
Array[Array[Int]]): Array[Boolean] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec path_existence_queries(n :: integer, nums :: [integer], max_diff ::
integer, queries :: [[integer]]) :: [boolean]
def path_existence_queries(n, nums, max_diff, queries) do

end
end
```

Erlang Solution:

```
-spec path_existence_queries(N :: integer(), Nums :: [integer()], MaxDiff ::
integer(), Queries :: [[integer()]]) -> [boolean()].
path_existence_queries(N, Nums, MaxDiff, Queries) ->
.
```

Racket Solution:

```
(define/contract (path-existence-queries n nums maxDiff queries)
(-> exact-integer? (listof exact-integer?) exact-integer? (listof (listof
exact-integer?)) (listof boolean?))
)
```