Problem 3534: Path Existence Queries in a Graph II

Problem Information

Difficulty: Hard
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

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

]

, find the

minimum

distance between nodes

u

i

and

v

i

.

If no path exists between the two nodes, return -1 for that query.

Return an array

answer

, where

answer[i]

is the result of the

i

th

query.

Note:

The edges between the nodes are unweighted.

Example 1:

Input:

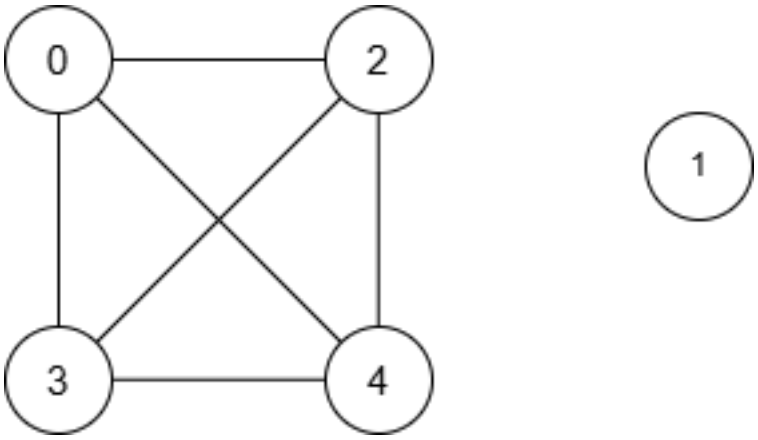
n = 5, nums = [1,8,3,4,2], maxDiff = 3, queries = [[0,3],[2,4]]

Output:

[1,1]

Explanation:

The resulting graph is:



Query

Shortest Path

Minimum Distance

[0, 3]

$0 \rightarrow 3$

1

[2, 4]

$2 \rightarrow 4$

1

Thus, the output is

[1, 1]

.

Example 2:

Input:

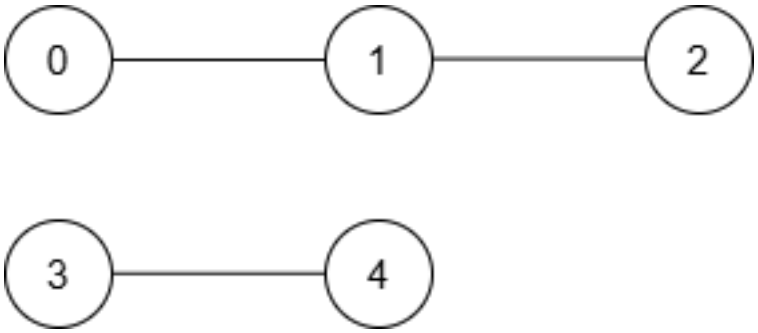
n = 5, nums = [5,3,1,9,10], maxDiff = 2, queries = [[0,1],[0,2],[2,3],[4,3]]

Output:

[1,2,-1,1]

Explanation:

The resulting graph is:



Query

Shortest Path

Minimum Distance

[0, 1]

$0 \rightarrow 1$

1

[0, 2]

$0 \rightarrow 1 \rightarrow 2$

2

[2, 3]

None

-1

[4, 3]

$3 \rightarrow 4$

1

Thus, the output is

[1, 2, -1, 1]

.

Example 3:

Input:

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

Output:

[0,-1,-1]

Explanation:

There are no edges between any two nodes because:

Nodes 0 and 1:

|nums[0] - nums[1]| = |3 - 6| = 3 > 1

Nodes 0 and 2:

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

Nodes 1 and 2:

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

Thus, no node can reach any other node, and the output is

[0, -1, -1]

.

Constraints:

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

5

0 <= nums[i] <= 10

5

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<int> pathExistenceQueries(int n, vector<int>& nums, int maxDiff,
vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public int[] 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[int]:
```

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[int]
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[]} nums
* @param {number} maxDiff
```

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

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

}
}
```

Dart:

```
class Solution {
List<int> 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[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec path_existence_queries(n :: integer, nums :: [integer], max_diff ::
integer, queries :: [[integer]]) :: [integer]
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()]]) -> [integer()].
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 exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Path Existence Queries in a Graph II
* Difficulty: Hard
* Tags: array, graph, dp, greedy, sort, 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> pathExistenceQueries(int n, vector<int>& nums, int maxDiff,
vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Path Existence Queries in a Graph II
* Difficulty: Hard
* Tags: array, graph, dp, greedy, sort, 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[] pathExistenceQueries(int n, int[] nums, int maxDiff, int[][]
queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Path Existence Queries in a Graph II
Difficulty: Hard
Tags: array, graph, dp, greedy, sort, 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 pathExistenceQueries(self, n: int, nums: List[int], maxDiff: int,
queries: List[List[int]]) -> List[int]:
# 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[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Path Existence Queries in a Graph II
* Difficulty: Hard
* Tags: array, graph, dp, greedy, sort, 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[]} nums
* @param {number} maxDiff
* @param {number[][]} queries
* @return {number[]}
*/
var pathExistenceQueries = function(n, nums, maxDiff, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Path Existence Queries in a Graph II
* Difficulty: Hard
* Tags: array, graph, dp, greedy, sort, 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 pathExistenceQueries(n: number, nums: number[], maxDiff: number,
queries: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Path Existence Queries in a Graph II
```

```
* Difficulty: Hard
* Tags: array, graph, dp, greedy, sort, 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[] PathExistenceQueries(int n, int[] nums, int maxDiff, int[][]
queries) {

}
}
```

C Solution:

```
/*
* Problem: Path Existence Queries in a Graph II
* Difficulty: Hard
* Tags: array, graph, dp, greedy, sort, 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* 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 II
// Difficulty: Hard
// Tags: array, graph, dp, greedy, sort, 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 pathExistenceQueries(n int, nums []int, maxDiff int, queries [][]int)
[]int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Path Existence Queries in a Graph II
// Difficulty: Hard
// Tags: array, graph, dp, greedy, sort, 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 path_existence_queries(n: i32, nums: Vec<i32>, max_diff: i32, queries:
Vec<Vec<i32>>) -> Vec<i32> {
```

```
}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[]} nums
# @param {Integer} max_diff
# @param {Integer[][]} queries
# @return {Integer[]}
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[][] $queries
* @return Integer[]
*/
function pathExistenceQueries($n, $nums, $maxDiff, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> 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[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec path_existence_queries(n :: integer, nums :: [integer], max_diff ::
integer, queries :: [[integer]]) :: [integer]
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()]]) -> [integer()].
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 exact-integer?))
)
```