Problem 1627: Graph Connectivity With Threshold

Problem Information

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

Problem Description

We have

n

cities labeled from

1

to

n

. Two different cities with labels

x

and

y

are directly connected by a bidirectional road if and only if

x

and

y

share a common divisor

strictly greater

than some

threshold

. More formally, cities with labels

x

and

y

have a road between them if there exists an integer

z

such that all of the following are true:

x % z == 0

,

y % z == 0

, and

z > threshold

.

Given the two integers,

n

and

threshold

, and an array of

queries

, you must determine for each

queries[i] = [a

i

, b

i

]

if cities

a

i

and

b

i

are connected directly or indirectly. (i.e. there is some path between them).

Return

an array

answer

, where

answer.length == queries.length

and

answer[i]

is

true

if for the

i

th

query, there is a path between

a

i

and

b

i

, or

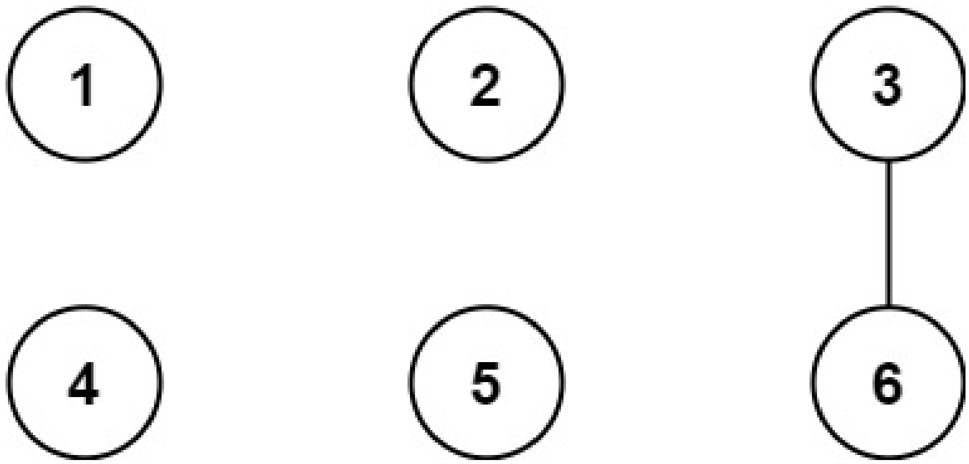
answer[i]

is

false

if there is no path.

Example 1:



Input:

n = 6, threshold = 2, queries = [[1,4],[2,5],[3,6]]

Output:

[false,false,true]

Explanation:

The divisors for each number: 1: 1 2: 1, 2 3: 1,

3

4: 1, 2,

4

5: 1,

5

6: 1, 2,

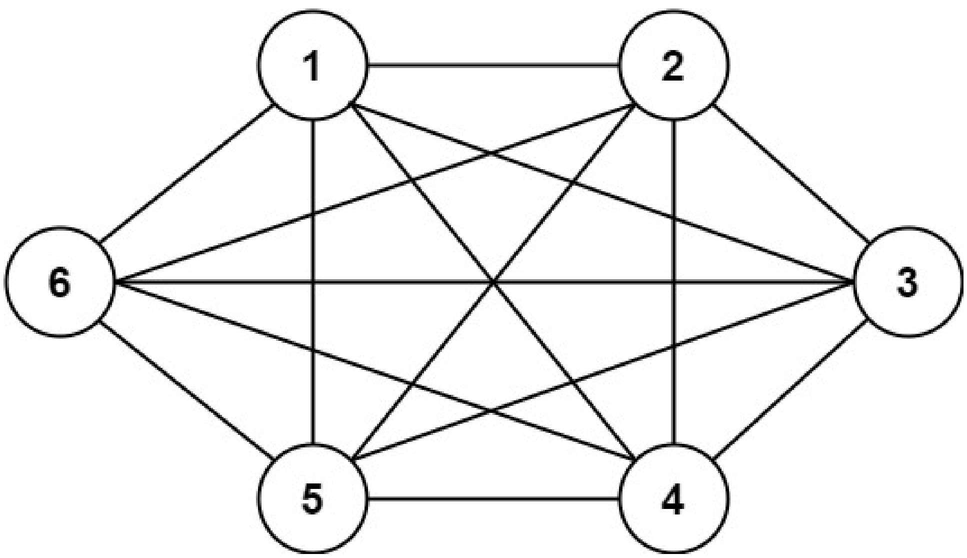
3

,

6

Using the underlined divisors above the threshold, only cities 3 and 6 share a common divisor, so they are the only ones directly connected. The result of each query: [1,4] 1 is not connected to 4 [2,5] 2 is not connected to 5 [3,6] 3 is connected to 6 through path 3--6

Example 2:



Input:

n = 6, threshold = 0, queries = [[4,5],[3,4],[3,2],[2,6],[1,3]]

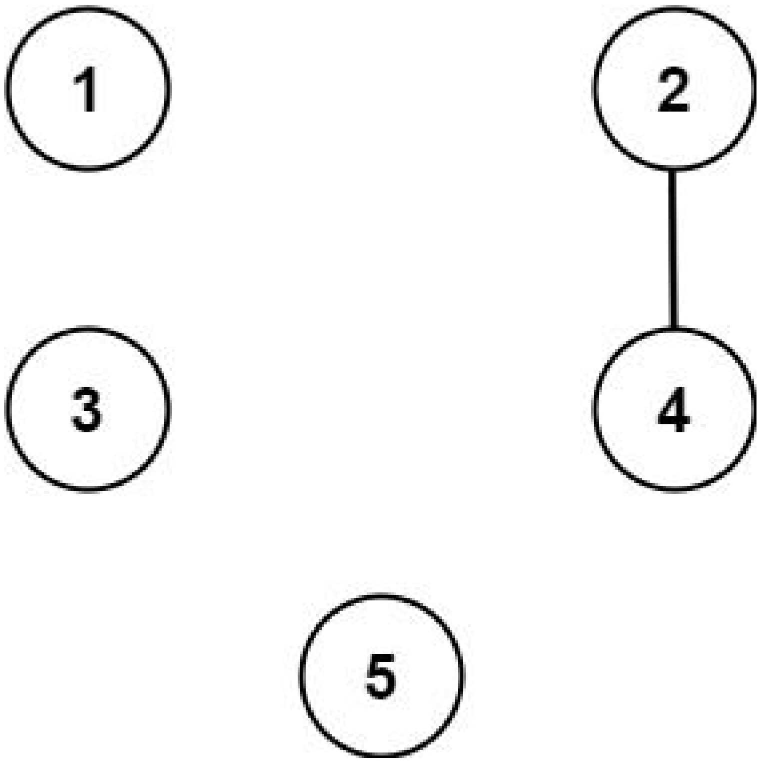
Output:

[true,true,true,true,true]

Explanation:

The divisors for each number are the same as the previous example. However, since the threshold is 0, all divisors can be used. Since all numbers share 1 as a divisor, all cities are connected.

Example 3:



Input:

n = 5, threshold = 1, queries = [[4,5],[4,5],[3,2],[2,3],[3,4]]

Output:

[false,false,false,false,false]

Explanation:

Only cities 2 and 4 share a common divisor 2 which is strictly greater than the threshold 1, so they are the only ones directly connected. Please notice that there can be multiple queries for the same pair of nodes [x, y], and that the query [x, y] is equivalent to the query [y, x].

Constraints:

2 <= n <= 10

0 <= threshold <= n

1 <= queries.length <= 10

5

queries[i].length == 2

1 <= a

i

, b

i

<= cities

a

i

!= b

i

Code Snippets

C++:

```
class Solution {
public:
vector<bool> areConnected(int n, int threshold, vector<vector<int>>& queries)
{

}
};
```

Java:

```
class Solution {
public List<Boolean> areConnected(int n, int threshold, int[][] queries) {

}
}
```

Python3:

```
class Solution:
def areConnected(self, n: int, threshold: int, queries: List[List[int]]) ->
List[bool]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function areConnected(n: number, threshold: number, queries: number[][]):
boolean[] {

};
```

C#:

```
public class Solution {
public IList<bool> AreConnected(int n, int threshold, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
bool* areConnected(int n, int threshold, int** queries, int queriesSize, int*
queriesColSize, int* returnSize) {

}
```

Go:

```
func areConnected(n int, threshold int, queries [][]int) []bool {

}
```

Kotlin:

```
class Solution {
fun areConnected(n: Int, threshold: Int, queries: Array<IntArray>):
List<Boolean> {

}
}
```

Swift:

```
class Solution {
func areConnected(_ n: Int, _ threshold: Int, _ queries: [[Int]]) -> [Bool] {

}
}
```

Rust:

```
impl Solution {
pub fn are_connected(n: i32, threshold: i32, queries: Vec<Vec<i32>>) ->
```

```
Vec<bool> {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} threshold
# @param {Integer[][]} queries
# @return {Boolean[]}
def are_connected(n, threshold, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $threshold
* @param Integer[][] $queries
* @return Boolean[]
*/
function areConnected($n, $threshold, $queries) {

}
}
```

Dart:

```
class Solution {
List<bool> areConnected(int n, int threshold, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def areConnected(n: Int, threshold: Int, queries: Array[Array[Int]]):
List[Boolean] = {
```

```
}
}
```

Elixir:

```
defmodule Solution do
@spec are_connected(n :: integer, threshold :: integer, queries ::
[[integer]]) :: [boolean]
def are_connected(n, threshold, queries) do

end
end
```

Erlang:

```
-spec are_connected(N :: integer(), Threshold :: integer(), Queries ::
[[integer()]]) -> [boolean()].
are_connected(N, Threshold, Queries) ->
.
```

Racket:

```
(define/contract (are-connected n threshold queries)
(-> exact-integer? exact-integer? (listof (listof exact-integer?)) (listof
boolean?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Graph Connectivity With Threshold
* Difficulty: Hard
* Tags: array, graph, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

class Solution {
public:
vector<bool> areConnected(int n, int threshold, vector<vector<int>>& queries)
{

}
};
```

Java Solution:

```
/**
* Problem: Graph Connectivity With Threshold
* Difficulty: Hard
* Tags: array, graph, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public List<Boolean> areConnected(int n, int threshold, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Graph Connectivity With Threshold
Difficulty: Hard
Tags: array, graph, math

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
```

```
def areConnected(self, n: int, threshold: int, queries: List[List[int]]) ->
List[bool]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Graph Connectivity With Threshold
* Difficulty: Hard
* Tags: array, graph, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @param {number} threshold
* @param {number[][]} queries
* @return {boolean[]}
*/
var areConnected = function(n, threshold, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Graph Connectivity With Threshold
* Difficulty: Hard
* Tags: array, graph, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function areConnected(n: number, threshold: number, queries: number[][]):
boolean[] {

};
```

C# Solution:

```
/*
* Problem: Graph Connectivity With Threshold
* Difficulty: Hard
* Tags: array, graph, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public IList<bool> AreConnected(int n, int threshold, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Graph Connectivity With Threshold
* Difficulty: Hard
* Tags: array, graph, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
bool* areConnected(int n, int threshold, int** queries, int queriesSize, int*
queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Graph Connectivity With Threshold
// Difficulty: Hard
// Tags: array, graph, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func areConnected(n int, threshold int, queries [][]int) []bool {

}
```

Kotlin Solution:

```
class Solution {
fun areConnected(n: Int, threshold: Int, queries: Array<IntArray>):
List<Boolean> {

}
}
```

Swift Solution:

```
class Solution {
func areConnected(_ n: Int, _ threshold: Int, _ queries: [[Int]]) -> [Bool] {

}
}
```

Rust Solution:

```
// Problem: Graph Connectivity With Threshold
// Difficulty: Hard
// Tags: array, graph, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn are_connected(n: i32, threshold: i32, queries: Vec<Vec<i32>>) ->
Vec<bool> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} threshold
# @param {Integer[][]} queries
# @return {Boolean[]}
def are_connected(n, threshold, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $threshold
* @param Integer[][] $queries
* @return Boolean[]
*/
function areConnected($n, $threshold, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<bool> areConnected(int n, int threshold, List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def areConnected(n: Int, threshold: Int, queries: Array[Array[Int]]):
List[Boolean] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec are_connected(n :: integer, threshold :: integer, queries ::
[[integer]]) :: [boolean]
def are_connected(n, threshold, queries) do

end
end
```

Erlang Solution:

```
-spec are_connected(N :: integer(), Threshold :: integer(), Queries ::
[[integer()]]) -> [boolean()].
are_connected(N, Threshold, Queries) ->
.
```

Racket Solution:

```
(define/contract (are-connected n threshold queries)
(-> exact-integer? exact-integer? (listof (listof exact-integer?)) (listof
boolean?))
)
```