Problem 1697: Checking Existence of Edge Length Limited Paths

Problem Information

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

Problem Description

An undirected graph of

n

nodes is defined by

edgeList

, where

edgeList[i] = [u

i

, v

i

, dis

i

]

denotes an edge between nodes

u

i

and

v

i

with distance

dis

i

. Note that there may be

multiple

edges between two nodes.

Given an array

queries

, where

queries[j] = [p

j

, q

j

, limit

j

]

, your task is to determine for each

queries[j]

whether there is a path between

p

j

and

q

j

such that each edge on the path has a distance

strictly less than

limit

j

.

Return

a

boolean array

answer

, where

answer.length == queries.length

and the

j

th

value of

answer

is

true

if there is a path for

queries[j]

is

true

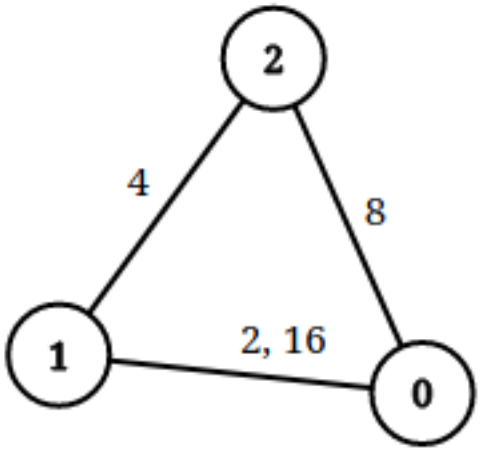
, and

false

otherwise

.

Example 1:



Input:

n = 3, edgeList = [[0,1,2],[1,2,4],[2,0,8],[1,0,16]], queries = [[0,1,2],[0,2,5]]

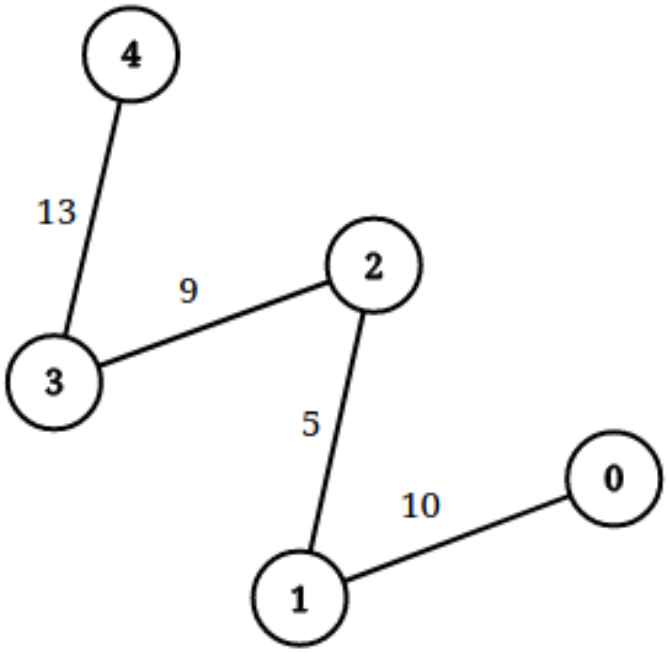
Output:

[false,true]

Explanation:

The above figure shows the given graph. Note that there are two overlapping edges between 0 and 1 with distances 2 and 16. For the first query, between 0 and 1 there is no path where each distance is less than 2, thus we return false for this query. For the second query, there is a path (0 -> 1 -> 2) of two edges with distances less than 5, thus we return true for this query.

Example 2:



Input:

n = 5, edgeList = [[0,1,10],[1,2,5],[2,3,9],[3,4,13]], queries = [[0,4,14],[1,4,13]]

Output:

[true,false]

Explanation:

The above figure shows the given graph.

Constraints:

2 <= n <= 10

5

1 <= edgeList.length, queries.length <= 10

5

edgeList[i].length == 3

queries[j].length == 3

0 <= u

i

, v

i

, p

j

, q

j

<= n - 1

u

i

!= v

i

p

j

!= q

j

1 <= dis

i

, limit

j

<= 10

9

There may be

multiple

edges between two nodes.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean[] distanceLimitedPathsExist(int n, int[][] edgeList, int[][]
queries) {

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function distanceLimitedPathsExist(n: number, edgeList: number[][], queries:
number[][]): boolean[] {

};
```

C#:

```
public class Solution {
public bool[] DistanceLimitedPathsExist(int n, int[][] edgeList, int[][]
```

```
queries) {

}
}
```

C:

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

}
```

Go:

```
func distanceLimitedPathsExist(n int, edgeList [][]int, queries [][]int)
[]bool {

}
```

Kotlin:

```
class Solution {
fun distanceLimitedPathsExist(n: Int, edgeList: Array<IntArray>, queries:
Array<IntArray>): BooleanArray {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn distance_limited_paths_exist(n: i32, edge_list: Vec<Vec<i32>>,
queries: Vec<Vec<i32>>) -> Vec<bool> {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<bool> distanceLimitedPathsExist(int n, List<List<int>> edgeList,
List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def distanceLimitedPathsExist(n: Int, edgeList: Array[Array[Int]], queries:
Array[Array[Int]]): Array[Boolean] = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec distance_limited_paths_exist(N :: integer(), EdgeList :: [[integer()]],
Queries :: [[integer()]]) -> [boolean()].
distance_limited_paths_exist(N, EdgeList, Queries) ->
.
```

Racket:

```
(define/contract (distance-limited-paths-exist n edgeList queries)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) (listof boolean?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Checking Existence of Edge Length Limited Paths
* Difficulty: Hard
* Tags: array, graph, sort
*
* 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> distanceLimitedPathsExist(int n, vector<vector<int>>& edgeList,
vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Checking Existence of Edge Length Limited Paths
* Difficulty: Hard
* Tags: array, graph, sort
*
* 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 boolean[] distanceLimitedPathsExist(int n, int[][] edgeList, int[][]
queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Checking Existence of Edge Length Limited Paths
Difficulty: Hard
Tags: array, graph, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Checking Existence of Edge Length Limited Paths
* Difficulty: Hard
* Tags: array, graph, sort
*
* 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[][]} edgeList
* @param {number[][]} queries
* @return {boolean[]}
*/
var distanceLimitedPathsExist = function(n, edgeList, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Checking Existence of Edge Length Limited Paths
* Difficulty: Hard
* Tags: array, graph, sort
*
* 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 distanceLimitedPathsExist(n: number, edgeList: number[][], queries:
number[][]): boolean[] {

};
```

C# Solution:

```
/*
* Problem: Checking Existence of Edge Length Limited Paths
* Difficulty: Hard
* Tags: array, graph, sort
*
* 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 bool[] DistanceLimitedPathsExist(int n, int[][] edgeList, int[][]
queries) {

}
}
```

C Solution:

```
/*
* Problem: Checking Existence of Edge Length Limited Paths
* Difficulty: Hard
* Tags: array, graph, sort
*
```

```
* 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* distanceLimitedPathsExist(int n, int** edgeList, int edgeListSize, int*
edgeListColSize, int** queries, int queriesSize, int* queriesColSize, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Checking Existence of Edge Length Limited Paths
// Difficulty: Hard
// Tags: array, graph, sort
//
// 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 distanceLimitedPathsExist(n int, edgeList [][]int, queries [][]int)
[]bool {

}
```

Kotlin Solution:

```
class Solution {
fun distanceLimitedPathsExist(n: Int, edgeList: Array<IntArray>, queries:
Array<IntArray>): BooleanArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Checking Existence of Edge Length Limited Paths
// Difficulty: Hard
// Tags: array, graph, sort
//
// 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 distance_limited_paths_exist(n: i32, edge_list: Vec<Vec<i32>>,
queries: Vec<Vec<i32>>) -> Vec<bool> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edgeList
* @param Integer[][] $queries
```

```
* @return Boolean[]
*/
function distanceLimitedPathsExist($n, $edgeList, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<bool> distanceLimitedPathsExist(int n, List<List<int>> edgeList,
List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def distanceLimitedPathsExist(n: Int, edgeList: Array[Array[Int]], queries:
Array[Array[Int]]): Array[Boolean] = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec distance_limited_paths_exist(N :: integer(), EdgeList :: [[integer()]],
Queries :: [[integer()]]) -> [boolean()].
distance_limited_paths_exist(N, EdgeList, Queries) ->
.
```

Racket Solution:

```
(define/contract (distance-limited-paths-exist n edgeList queries)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) (listof boolean?))
)
```