Problem 1724: Checking Existence of Edge Length Limited Paths II

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, and the graph may not be connected.

Implement the

DistanceLimitedPathsExist

class:

DistanceLimitedPathsExist(int n, int[][] edgeList)

Initializes the class with an undirected graph.

boolean query(int p, int q, int limit)

Returns

true

if there exists a path from

p

to

q

such that each edge on the path has a distance

strictly less than

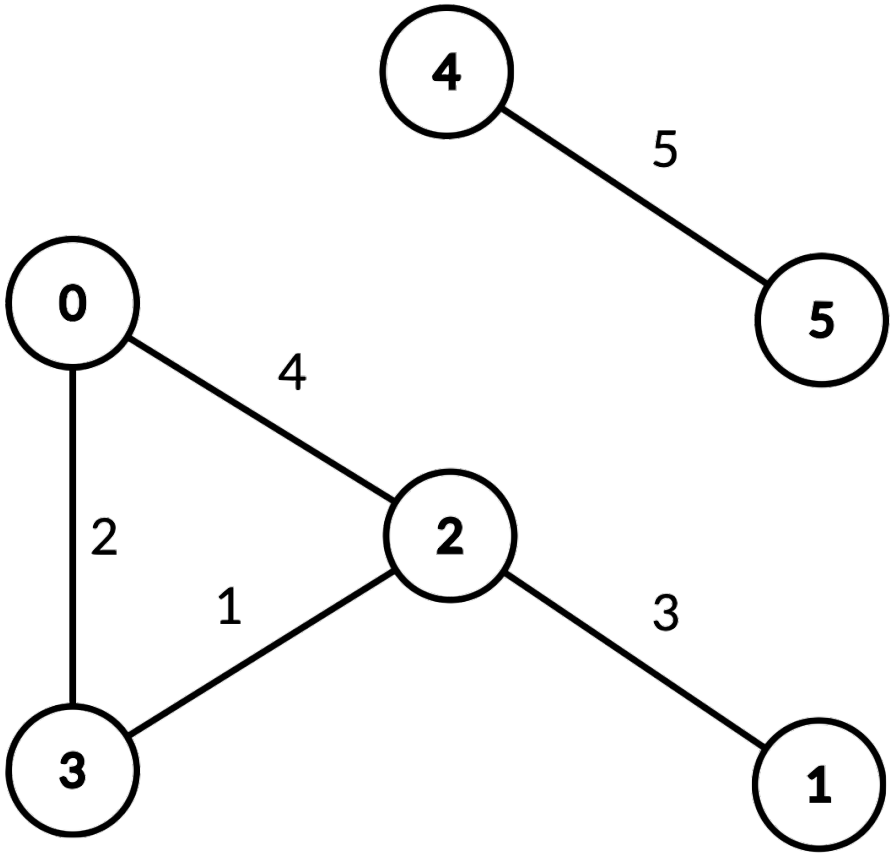
limit

, and otherwise

false

.

Example 1:



Input

["DistanceLimitedPathsExist", "query", "query", "query", "query"] [[6, [[0, 2, 4], [0, 3, 2], [1, 2, 3], [2, 3, 1], [4, 5, 5]]], [2, 3, 2], [1, 3, 3], [2, 0, 3], [0, 5, 6]]

Output

[null, true, false, true, false]

Explanation

DistanceLimitedPathsExist distanceLimitedPathsExist = new DistanceLimitedPathsExist(6, [[0, 2, 4], [0, 3, 2], [1, 2, 3], [2, 3, 1], [4, 5, 5]]); distanceLimitedPathsExist.query(2, 3, 2); // return true. There is an edge from 2 to 3 of distance 1, which is less than 2. distanceLimitedPathsExist.query(1, 3, 3); // return false. There is no way to go from 1 to 3 with distances

strictly

less than 3. distanceLimitedPathsExist.query(2, 0, 3); // return true. There is a way to go from 2 to 0 with distance < 3: travel from 2 to 3 to 0. distanceLimitedPathsExist.query(0, 5, 6); // return false. There are no paths from 0 to 5.

Constraints:

2 <= n <= 10

4

0 <= edgeList.length <= 10

4

edgeList[i].length == 3

0 <= u

i

, v

i

, p, q <= n-1

u

i

!= v

i

p != q

1 <= dis

i

, limit <= 10

9

At most

10

4

calls will be made to

query

.

Code Snippets

C++:

```
class DistanceLimitedPathsExist {
public:
DistanceLimitedPathsExist(int n, vector<vector<int>>& edgeList) {

}

bool query(int p, int q, int limit) {

}
};

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* DistanceLimitedPathsExist* obj = new DistanceLimitedPathsExist(n,
edgeList);
* bool param_1 = obj->query(p,q,limit);
*/
```

Java:

```
class DistanceLimitedPathsExist {

public DistanceLimitedPathsExist(int n, int[][] edgeList) {

}

public boolean query(int p, int q, int limit) {

}
}

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* DistanceLimitedPathsExist obj = new DistanceLimitedPathsExist(n, edgeList);
* boolean param_1 = obj.query(p,q,limit);
*/
```

Python3:

```
class DistanceLimitedPathsExist:

def __init__(self, n: int, edgeList: List[List[int]]):


def query(self, p: int, q: int, limit: int) -> bool:




# Your DistanceLimitedPathsExist object will be instantiated and called as
such:
# obj = DistanceLimitedPathsExist(n, edgeList)
# param_1 = obj.query(p,q,limit)
```

Python:

```
class DistanceLimitedPathsExist(object):

def __init__(self, n, edgeList):
"""
:type n: int
```

```
:type edgeList: List[List[int]]
"""


def query(self, p, q, limit):
"""
:type p: int
:type q: int
:type limit: int
:rtype: bool
"""



# Your DistanceLimitedPathsExist object will be instantiated and called as
such:
# obj = DistanceLimitedPathsExist(n, edgeList)
# param_1 = obj.query(p,q,limit)
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} edgeList
*/
var DistanceLimitedPathsExist = function(n, edgeList) {

};

/**
* @param {number} p
* @param {number} q
* @param {number} limit
* @return {boolean}
*/
DistanceLimitedPathsExist.prototype.query = function(p, q, limit) {

};

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
```

```
* var obj = new DistanceLimitedPathsExist(n, edgeList)
* var param_1 = obj.query(p,q,limit)
*/
```

TypeScript:

```
class DistanceLimitedPathsExist {
constructor(n: number, edgeList: number[][]) {

}

query(p: number, q: number, limit: number): boolean {

}
}

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* var obj = new DistanceLimitedPathsExist(n, edgeList)
* var param_1 = obj.query(p,q,limit)
*/
```

C#:

```
public class DistanceLimitedPathsExist {

public DistanceLimitedPathsExist(int n, int[][] edgeList) {

}

public bool Query(int p, int q, int limit) {

}
}

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* DistanceLimitedPathsExist obj = new DistanceLimitedPathsExist(n, edgeList);
* bool param_1 = obj.Query(p,q,limit);
*/
```

C:

```
typedef struct {

} DistanceLimitedPathsExist;


DistanceLimitedPathsExist* distanceLimitedPathsExistCreate(int n, int**
edgeList, int edgeListSize, int* edgeListColSize) {

}

bool distanceLimitedPathsExistQuery(DistanceLimitedPathsExist* obj, int p,
int q, int limit) {

}

void distanceLimitedPathsExistFree(DistanceLimitedPathsExist* obj) {

}

/**
* Your DistanceLimitedPathsExist struct will be instantiated and called as
such:
* DistanceLimitedPathsExist* obj = distanceLimitedPathsExistCreate(n,
edgeList, edgeListSize, edgeListColSize);
* bool param_1 = distanceLimitedPathsExistQuery(obj, p, q, limit);

* distanceLimitedPathsExistFree(obj);
*/
```

Go:

```
type DistanceLimitedPathsExist struct {

}


func Constructor(n int, edgeList [][]int) DistanceLimitedPathsExist {
```

```
}


func (this *DistanceLimitedPathsExist) Query(p int, q int, limit int) bool {

}


/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* obj := Constructor(n, edgeList);
* param_1 := obj.Query(p,q,limit);
*/
```

Kotlin:

```
class DistanceLimitedPathsExist(n: Int, edgeList: Array<IntArray>) {

fun query(p: Int, q: Int, limit: Int): Boolean {

}

}

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* var obj = DistanceLimitedPathsExist(n, edgeList)
* var param_1 = obj.query(p,q,limit)
*/
```

Swift:

```
class DistanceLimitedPathsExist {

init(_ n: Int, _ edgeList: [[Int]]) {

}
```

```
func query(_ p: Int, _ q: Int, _ limit: Int) -> Bool {

}
}

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* let obj = DistanceLimitedPathsExist(n, edgeList)
* let ret_1: Bool = obj.query(p, q, limit)
*/
```

Rust:

```
struct DistanceLimitedPathsExist {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl DistanceLimitedPathsExist {

fn new(n: i32, edgeList: Vec<Vec<i32>>) -> Self {

}

fn query(&self, p: i32, q: i32, limit: i32) -> bool {

}
}

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* let obj = DistanceLimitedPathsExist::new(n, edgeList);
* let ret_1: bool = obj.query(p, q, limit);
*/
```

Ruby:

```
class DistanceLimitedPathsExist

=begin
:type n: Integer
:type edge_list: Integer[][]
=end
def initialize(n, edge_list)

end



=begin
:type p: Integer
:type q: Integer
:type limit: Integer
:rtype: Boolean
=end
def query(p, q, limit)

end



end

# Your DistanceLimitedPathsExist object will be instantiated and called as
such:
# obj = DistanceLimitedPathsExist.new(n, edge_list)
# param_1 = obj.query(p, q, limit)
```

PHP:

```
class DistanceLimitedPathsExist {
/**
* @param Integer $n
* @param Integer[][] $edgeList
*/
function __construct($n, $edgeList) {

}

/**
```

```
* @param Integer $p
* @param Integer $q
* @param Integer $limit
* @return Boolean
*/
function query($p, $q, $limit) {

}
}

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* $obj = DistanceLimitedPathsExist($n, $edgeList);
* $ret_1 = $obj->query($p, $q, $limit);
*/
```

Dart:

```
class DistanceLimitedPathsExist {

DistanceLimitedPathsExist(int n, List<List<int>> edgeList) {

}

bool query(int p, int q, int limit) {

}
}

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* DistanceLimitedPathsExist obj = DistanceLimitedPathsExist(n, edgeList);
* bool param1 = obj.query(p,q,limit);
*/
```

Scala:

```
class DistanceLimitedPathsExist(_n: Int, _edgeList: Array[Array[Int]]) {

def query(p: Int, q: Int, limit: Int): Boolean = {
```

```
}

}

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* val obj = new DistanceLimitedPathsExist(n, edgeList)
* val param_1 = obj.query(p,q,limit)
*/
```

Elixir:

```
defmodule DistanceLimitedPathsExist do
@spec init_(n :: integer, edge_list :: [[integer]]) :: any
def init_(n, edge_list) do

end

@spec query(p :: integer, q :: integer, limit :: integer) :: boolean
def query(p, q, limit) do

end
end

# Your functions will be called as such:
# DistanceLimitedPathsExist.init_(n, edge_list)
# param_1 = DistanceLimitedPathsExist.query(p, q, limit)

# DistanceLimitedPathsExist.init_ will be called before every test case, in
which you can do some necessary initializations.
```

Erlang:

```
-spec distance_limited_paths_exist_init_(N :: integer(), EdgeList ::
[[integer()]]) -> any().
distance_limited_paths_exist_init_(N, EdgeList) ->
.

-spec distance_limited_paths_exist_query(P :: integer(), Q :: integer(),
Limit :: integer()) -> boolean().
```

```
distance_limited_paths_exist_query(P, Q, Limit) ->
.


%% Your functions will be called as such:
%% distance_limited_paths_exist_init_(N, EdgeList),
%% Param_1 = distance_limited_paths_exist_query(P, Q, Limit),

%% distance_limited_paths_exist_init_ will be called before every test case,
in which you can do some necessary initializations.
```

Racket:

```
(define distance-limited-paths-exist%
(class object%
(super-new)

; n : exact-integer?
; edge-list : (listof (listof exact-integer?))
(init-field
n
edge-list)

; query : exact-integer? exact-integer? exact-integer? -> boolean?
(define/public (query p q limit)
)))

;; Your distance-limited-paths-exist% object will be instantiated and called
as such:
;; (define obj (new distance-limited-paths-exist% [n n] [edge-list
edge-list]))
;; (define param_1 (send obj query p q limit))
```

Solutions

C++ Solution:

```
/*
* Problem: Checking Existence of Edge Length Limited Paths II
* Difficulty: Hard
```

```
* Tags: tree, graph, dp
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

class DistanceLimitedPathsExist {
public:
DistanceLimitedPathsExist(int n, vector<vector<int>>& edgeList) {

}

bool query(int p, int q, int limit) {

}
};

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* DistanceLimitedPathsExist* obj = new DistanceLimitedPathsExist(n,
edgeList);
* bool param_1 = obj->query(p,q,limit);
*/
```

Java Solution:

```
/**
* Problem: Checking Existence of Edge Length Limited Paths II
* Difficulty: Hard
* Tags: tree, graph, dp
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

class DistanceLimitedPathsExist {

public DistanceLimitedPathsExist(int n, int[][] edgeList) {
```

```
}

public boolean query(int p, int q, int limit) {

}
}

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* DistanceLimitedPathsExist obj = new DistanceLimitedPathsExist(n, edgeList);
* boolean param_1 = obj.query(p,q,limit);
*/
```

Python3 Solution:

```
"""
Problem: Checking Existence of Edge Length Limited Paths II
Difficulty: Hard
Tags: tree, graph, dp

Approach: DFS or BFS traversal
Time Complexity: O(n) where n is number of nodes
Space Complexity: O(n) or O(n * m) for DP table
"""

class DistanceLimitedPathsExist:

def __init__(self, n: int, edgeList: List[List[int]]):


def query(self, p: int, q: int, limit: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class DistanceLimitedPathsExist(object):

def __init__(self, n, edgeList):
```

```
        """
        :type n: int
        :type edgeList: List[List[int]]
        """


    def query(self, p, q, limit):
        """
        :type p: int
        :type q: int
        :type limit: int
        :rtype: bool
        """



# Your DistanceLimitedPathsExist object will be instantiated and called as
such:
# obj = DistanceLimitedPathsExist(n, edgeList)
# param_1 = obj.query(p,q,limit)
```

JavaScript Solution:

```
/**
* Problem: Checking Existence of Edge Length Limited Paths II
* Difficulty: Hard
* Tags: tree, graph, dp
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} n
* @param {number[][]} edgeList
*/
var DistanceLimitedPathsExist = function(n, edgeList) {

};
```

```
/**
* @param {number} p
* @param {number} q
* @param {number} limit
* @return {boolean}
*/
DistanceLimitedPathsExist.prototype.query = function(p, q, limit) {

};

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* var obj = new DistanceLimitedPathsExist(n, edgeList)
* var param_1 = obj.query(p,q,limit)
*/
```

TypeScript Solution:

```
/**
* Problem: Checking Existence of Edge Length Limited Paths II
* Difficulty: Hard
* Tags: tree, graph, dp
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

class DistanceLimitedPathsExist {
constructor(n: number, edgeList: number[][]) {

}

query(p: number, q: number, limit: number): boolean {

}
}

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
```

```
such:
* var obj = new DistanceLimitedPathsExist(n, edgeList)
* var param_1 = obj.query(p,q,limit)
*/
```

C# Solution:

```
/*
* Problem: Checking Existence of Edge Length Limited Paths II
* Difficulty: Hard
* Tags: tree, graph, dp
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class DistanceLimitedPathsExist {

public DistanceLimitedPathsExist(int n, int[][] edgeList) {

}

public bool Query(int p, int q, int limit) {

}
}

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* DistanceLimitedPathsExist obj = new DistanceLimitedPathsExist(n, edgeList);
* bool param_1 = obj.Query(p,q,limit);
*/
```

C Solution:

```
/*
* Problem: Checking Existence of Edge Length Limited Paths II
* Difficulty: Hard
* Tags: tree, graph, dp
*
```

```
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(n) or O(n * m) for DP table
*/




typedef struct {

} DistanceLimitedPathsExist;


DistanceLimitedPathsExist* distanceLimitedPathsExistCreate(int n, int**
edgeList, int edgeListSize, int* edgeListColSize) {

}

bool distanceLimitedPathsExistQuery(DistanceLimitedPathsExist* obj, int p,
int q, int limit) {

}

void distanceLimitedPathsExistFree(DistanceLimitedPathsExist* obj) {

}

/**
* Your DistanceLimitedPathsExist struct will be instantiated and called as
such:
* DistanceLimitedPathsExist* obj = distanceLimitedPathsExistCreate(n,
edgeList, edgeListSize, edgeListColSize);
* bool param_1 = distanceLimitedPathsExistQuery(obj, p, q, limit);

* distanceLimitedPathsExistFree(obj);
*/
```

Go Solution:

```
// Problem: Checking Existence of Edge Length Limited Paths II
// Difficulty: Hard
```

```
// Tags: tree, graph, dp
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(n) or O(n * m) for DP table

type DistanceLimitedPathsExist struct {

}


func Constructor(n int, edgeList [][]int) DistanceLimitedPathsExist {

}


func (this *DistanceLimitedPathsExist) Query(p int, q int, limit int) bool {

}


/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* obj := Constructor(n, edgeList);
* param_1 := obj.Query(p,q,limit);
*/
```

Kotlin Solution:

```
class DistanceLimitedPathsExist(n: Int, edgeList: Array<IntArray>) {

fun query(p: Int, q: Int, limit: Int): Boolean {

}

}

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
```

```
* var obj = DistanceLimitedPathsExist(n, edgeList)
* var param_1 = obj.query(p,q,limit)
*/
```

Swift Solution:

```
class DistanceLimitedPathsExist {

init(_ n: Int, _ edgeList: [[Int]]) {

}

func query(_ p: Int, _ q: Int, _ limit: Int) -> Bool {

}
}

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* let obj = DistanceLimitedPathsExist(n, edgeList)
* let ret_1: Bool = obj.query(p, q, limit)
*/
```

Rust Solution:

```
// Problem: Checking Existence of Edge Length Limited Paths II
// Difficulty: Hard
// Tags: tree, graph, dp
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(n) or O(n * m) for DP table

struct DistanceLimitedPathsExist {

}


/**
```

```
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl DistanceLimitedPathsExist {

fn new(n: i32, edgeList: Vec<Vec<i32>>) -> Self {

}

fn query(&self, p: i32, q: i32, limit: i32) -> bool {

}
}

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* let obj = DistanceLimitedPathsExist::new(n, edgeList);
* let ret_1: bool = obj.query(p, q, limit);
*/
```

Ruby Solution:

```
class DistanceLimitedPathsExist

=begin
:type n: Integer
:type edge_list: Integer[][]
=end
def initialize(n, edge_list)

end


=begin
:type p: Integer
:type q: Integer
:type limit: Integer
:rtype: Boolean
=end
def query(p, q, limit)
```

```
end


end

# Your DistanceLimitedPathsExist object will be instantiated and called as
such:
# obj = DistanceLimitedPathsExist.new(n, edge_list)
# param_1 = obj.query(p, q, limit)
```

PHP Solution:

```
class DistanceLimitedPathsExist {
/**
* @param Integer $n
* @param Integer[][] $edgeList
*/
function __construct($n, $edgeList) {

}

/**
* @param Integer $p
* @param Integer $q
* @param Integer $limit
* @return Boolean
*/
function query($p, $q, $limit) {

}
}

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* $obj = DistanceLimitedPathsExist($n, $edgeList);
* $ret_1 = $obj->query($p, $q, $limit);
*/
```

Dart Solution:

```
class DistanceLimitedPathsExist {

DistanceLimitedPathsExist(int n, List<List<int>> edgeList) {

}

bool query(int p, int q, int limit) {

}
}

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* DistanceLimitedPathsExist obj = DistanceLimitedPathsExist(n, edgeList);
* bool param1 = obj.query(p,q,limit);
*/
```

Scala Solution:

```
class DistanceLimitedPathsExist(_n: Int, _edgeList: Array[Array[Int]]) {

def query(p: Int, q: Int, limit: Int): Boolean = {

}

}

/**
* Your DistanceLimitedPathsExist object will be instantiated and called as
such:
* val obj = new DistanceLimitedPathsExist(n, edgeList)
* val param_1 = obj.query(p,q,limit)
*/
```

Elixir Solution:

```
defmodule DistanceLimitedPathsExist do
@spec init_(n :: integer, edge_list :: [[integer]]) :: any
def init_(n, edge_list) do

end
```

```
@spec query(p :: integer, q :: integer, limit :: integer) :: boolean
def query(p, q, limit) do

end
end

# Your functions will be called as such:
# DistanceLimitedPathsExist.init_(n, edge_list)
# param_1 = DistanceLimitedPathsExist.query(p, q, limit)

# DistanceLimitedPathsExist.init_ will be called before every test case, in
which you can do some necessary initializations.
```

Erlang Solution:

```
-spec distance_limited_paths_exist_init_(N :: integer(), EdgeList ::
[[integer()]]) -> any().
distance_limited_paths_exist_init_(N, EdgeList) ->
.

-spec distance_limited_paths_exist_query(P :: integer(), Q :: integer(),
Limit :: integer()) -> boolean().
distance_limited_paths_exist_query(P, Q, Limit) ->
.


%% Your functions will be called as such:
%% distance_limited_paths_exist_init_(N, EdgeList),
%% Param_1 = distance_limited_paths_exist_query(P, Q, Limit),

%% distance_limited_paths_exist_init_ will be called before every test case,
in which you can do some necessary initializations.
```

Racket Solution:

```
(define distance-limited-paths-exist%
(class object%
(super-new)

; n : exact-integer?
```

```
; edge-list : (listof (listof exact-integer?))
(init-field
n
edge-list)

; query : exact-integer? exact-integer? exact-integer? -> boolean?
(define/public (query p q limit)
)))

;; Your distance-limited-paths-exist% object will be instantiated and called
as such:
;; (define obj (new distance-limited-paths-exist% [n n] [edge-list
edge-list]))
;; (define param_1 (send obj query p q limit))
```