Problem 3620: Network Recovery Pathways

Problem Information

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

Problem Description

You are given a directed acyclic graph of

n

nodes numbered from 0 to

n – 1

. This is represented by a 2D array

edges

of length

m

, where

edges[i] = [u

i

, v

i

, $cost_i$] indicates a one■way communication from node u_i to node v_i with a recovery cost of $cost_i$.

Some nodes may be offline. You are given a boolean array online where online[i] = true means node i is online. Nodes 0 and n – 1

are always online.

A path from 0 to

n – 1

is

valid

if:

All intermediate nodes on the path are online.

The total recovery cost of all edges on the path does not exceed

k

.

For each valid path, define its

score

as the minimum edge■cost along that path.

Return the

maximum

path score (i.e., the largest

minimum

-edge cost) among all valid paths. If no valid path exists, return -1.

Example 1:

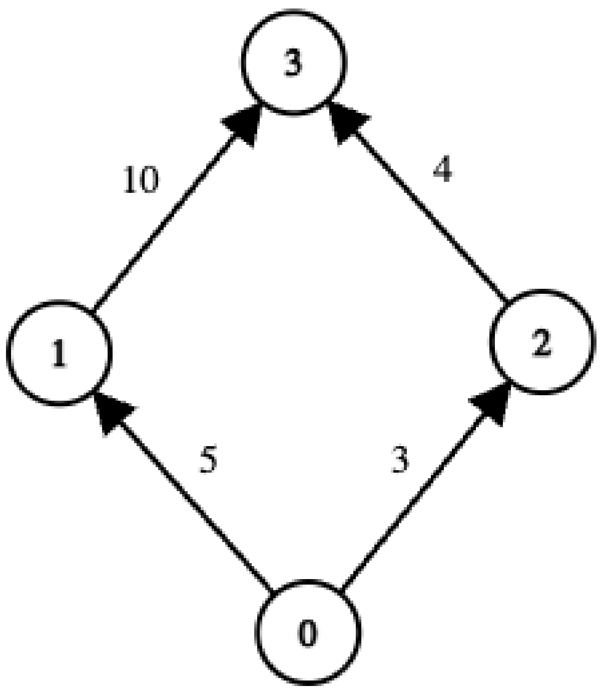
Input:

edges = [[0,1,5],[1,3,10],[0,2,3],[2,3,4]], online = [true,true,true,true], k = 10

Output:

3

Explanation:



The graph has two possible routes from node 0 to node 3:

Path

$0 \rightarrow 1 \rightarrow 3$

Total cost =

5 + 10 = 15

, which exceeds k (

$15 > 10$

), so this path is invalid.

Path

$0 \rightarrow 2 \rightarrow 3$

Total cost =

$3 + 4 = 7 <= k$

, so this path is valid.

The minimum edge■cost along this path is

$\min(3, 4) = 3$

.

There are no other valid paths. Hence, the maximum among all valid path■scores is 3.

Example 2:

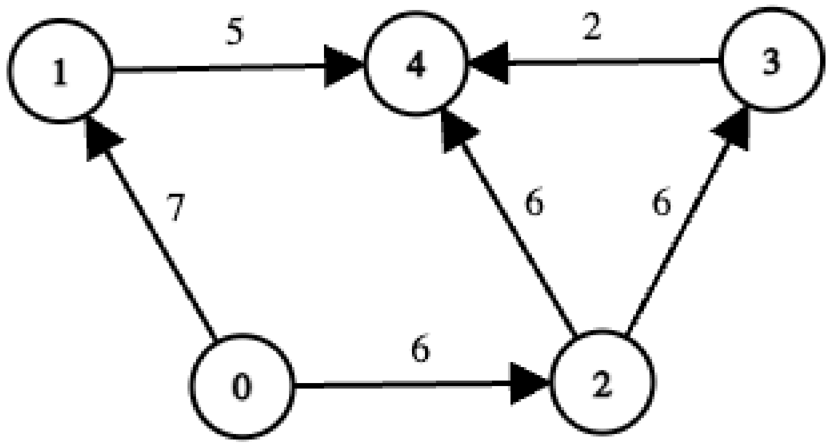
Input:

edges = [[0,1,7],[1,4,5],[0,2,6],[2,3,6],[3,4,2],[2,4,6]], online = [true,true,true,false,true], k = 12

Output:

6

Explanation:



Node 3 is offline, so any path passing through 3 is invalid.

Consider the remaining routes from 0 to 4:

Path

$0 \rightarrow 1 \rightarrow 4$

Total cost =

$7 + 5 = 12 <= k$

, so this path is valid.

The minimum edge■cost along this path is

$\min(7, 5) = 5$

.

Path

$0 \rightarrow 2 \rightarrow 3 \rightarrow 4$

Node 3 is offline, so this path is invalid regardless of cost.

Path

0 → 2 → 4

Total cost =

6 + 6 = 12 <= k

, so this path is valid.

The minimum edge■cost along this path is

min(6, 6) = 6

.

Among the two valid paths, their scores are 5 and 6. Therefore, the answer is 6.

Constraints:

n == online.length

2 <= n <= 5 * 10

4

0 <= m == edges.length <=

min(10

5

, n * (n - 1) / 2)

edges[i] = [u

i

, v

i

, cost

i

]

0 <= u

i

, v

i

< n

u

i

!= v

i

0 <= cost

i

<= 10

9

0 <= k <= 5 * 10

13

online[i]

is either

true

or

false

, and both

online[0]

and

online[n – 1]

are

true

.

The given graph is a directed acyclic graph.

Code Snippets

C++:

```
class Solution {
public:
int findMaxPathScore(vector<vector<int>>& edges, vector<bool>& online, long
long k) {

}
};
```

Java:

```
class Solution {
public int findMaxPathScore(int[][] edges, boolean[] online, long k) {

}
}
```

Python3:

```
class Solution:
def findMaxPathScore(self, edges: List[List[int]], online: List[bool], k:
int) -> int:
```

Python:

```
class Solution(object):
def findMaxPathScore(self, edges, online, k):
"""
:type edges: List[List[int]]
:type online: List[bool]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} edges
* @param {boolean[]} online
* @param {number} k
* @return {number}
*/
var findMaxPathScore = function(edges, online, k) {

};
```

TypeScript:

```
function findMaxPathScore(edges: number[][], online: boolean[], k: number):
number {

};
```

C#:

```
public class Solution {
public int FindMaxPathScore(int[][] edges, bool[] online, long k) {

}
}
```

C:

```
int findMaxPathScore(int** edges, int edgesSize, int* edgesColSize, bool*
online, int onlineSize, long long k) {

}
```

Go:

```
func findMaxPathScore(edges [][]int, online []bool, k int64) int {

}
```

Kotlin:

```
class Solution {
fun findMaxPathScore(edges: Array<IntArray>, online: BooleanArray, k: Long):
Int {

}
}
```

Swift:

```
class Solution {
func findMaxPathScore(_ edges: [[Int]], _ online: [Bool], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_max_path_score(edges: Vec<Vec<i32>>, online: Vec<bool>, k: i64)
-> i32 {

}
```

```
}
```

Ruby:

```
# @param {Integer[][]} edges
# @param {Boolean[]} online
# @param {Integer} k
# @return {Integer}
def find_max_path_score(edges, online, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Boolean[] $online
* @param Integer $k
* @return Integer
*/
function findMaxPathScore($edges, $online, $k) {

}
}
```

Dart:

```
class Solution {
int findMaxPathScore(List<List<int>> edges, List<bool> online, int k) {

}
}
```

Scala:

```
object Solution {
def findMaxPathScore(edges: Array[Array[Int]], online: Array[Boolean], k:
Long): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec find_max_path_score(edges :: [[integer]], online :: [boolean], k ::
integer) :: integer
def find_max_path_score(edges, online, k) do

end
end
```

Erlang:

```
-spec find_max_path_score(Edges :: [[integer()]], Online :: [boolean()], K ::
integer()) -> integer().
find_max_path_score(Edges, Online, K) ->
.
```

Racket:

```
(define/contract (find-max-path-score edges online k)
(-> (listof (listof exact-integer?)) (listof boolean?) exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Network Recovery Pathways
* Difficulty: Hard
* Tags: array, graph, dp, sort, search, queue, heap
*
* 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 findMaxPathScore(vector<vector<int>>& edges, vector<bool>& online, long
long k) {

}
};
```

Java Solution:

```
/**
* Problem: Network Recovery Pathways
* Difficulty: Hard
* Tags: array, graph, dp, sort, search, queue, heap
*
* 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 findMaxPathScore(int[][] edges, boolean[] online, long k) {

}
}
```

Python3 Solution:

```
"""
Problem: Network Recovery Pathways
Difficulty: Hard
Tags: array, graph, dp, sort, search, queue, heap

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 findMaxPathScore(self, edges: List[List[int]], online: List[bool], k:
int) -> int:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findMaxPathScore(self, edges, online, k):
"""
:type edges: List[List[int]]
:type online: List[bool]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Network Recovery Pathways
* Difficulty: Hard
* Tags: array, graph, dp, sort, search, queue, heap
*
* 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[][]} edges
* @param {boolean[]} online
* @param {number} k
* @return {number}
*/
var findMaxPathScore = function(edges, online, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Network Recovery Pathways
* Difficulty: Hard
```

```
* Tags: array, graph, dp, sort, search, queue, heap
*
* 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 findMaxPathScore(edges: number[][], online: boolean[], k: number):
number {

};
```

C# Solution:

```
/*
* Problem: Network Recovery Pathways
* Difficulty: Hard
* Tags: array, graph, dp, sort, search, queue, heap
*
* 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 FindMaxPathScore(int[][] edges, bool[] online, long k) {

}
}
```

C Solution:

```
/*
* Problem: Network Recovery Pathways
* Difficulty: Hard
* Tags: array, graph, dp, sort, search, queue, heap
*
* 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
*/
```

```
int findMaxPathScore(int** edges, int edgesSize, int* edgesColSize, bool*
online, int onlineSize, long long k) {

}
```

Go Solution:

```
// Problem: Network Recovery Pathways
// Difficulty: Hard
// Tags: array, graph, dp, sort, search, queue, heap
//
// 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 findMaxPathScore(edges [][]int, online []bool, k int64) int {

}
```

Kotlin Solution:

```
class Solution {
fun findMaxPathScore(edges: Array<IntArray>, online: BooleanArray, k: Long):
Int {

}
}
```

Swift Solution:

```
class Solution {
func findMaxPathScore(_ edges: [[Int]], _ online: [Bool], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Network Recovery Pathways
// Difficulty: Hard
```

```
// Tags: array, graph, dp, sort, search, queue, heap
//
// 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 find_max_path_score(edges: Vec<Vec<i32>>, online: Vec<bool>, k: i64)
-> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} edges
# @param {Boolean[]} online
# @param {Integer} k
# @return {Integer}
def find_max_path_score(edges, online, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $edges
* @param Boolean[] $online
* @param Integer $k
* @return Integer
*/
function findMaxPathScore($edges, $online, $k) {

}
}
```

Dart Solution:

```
class Solution {
int findMaxPathScore(List<List<int>> edges, List<bool> online, int k) {

}
}
```

Scala Solution:

```
object Solution {
def findMaxPathScore(edges: Array[Array[Int]], online: Array[Boolean], k:
Long): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_max_path_score(edges :: [[integer]], online :: [boolean], k ::
integer) :: integer
def find_max_path_score(edges, online, k) do

end
end
```

Erlang Solution:

```
-spec find_max_path_score(Edges :: [[integer()]], Online :: [boolean()], K ::
integer()) -> integer().
find_max_path_score(Edges, Online, K) ->
.
```

Racket Solution:

```
(define/contract (find-max-path-score edges online k)
(-> (listof (listof exact-integer?)) (listof boolean?) exact-integer?
exact-integer?)
)
```