Problem 3604: Minimum Time to Reach Destination in Directed Graph

Problem Information

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

Problem Description

You are given an integer

n

and a

directed

graph with

n

nodes labeled from 0 to

n - 1

. This is represented by a 2D array

edges

, where

edges[i] = [u

i

, v

i

, start

i

, end

i

]

indicates an edge from node

u

i

to

v

i

that can

only

be used at any integer time

t

such that

start

i

<= t <= end

i

.

You start at node 0 at time 0.

In one unit of time, you can either:

Wait at your current node without moving, or

Travel along an outgoing edge from your current node if the current time

t

satisfies

start

i

<= t <= end

i

.

Return the

minimum

time required to reach node

n - 1

. If it is impossible, return

-1

.

Example 1:

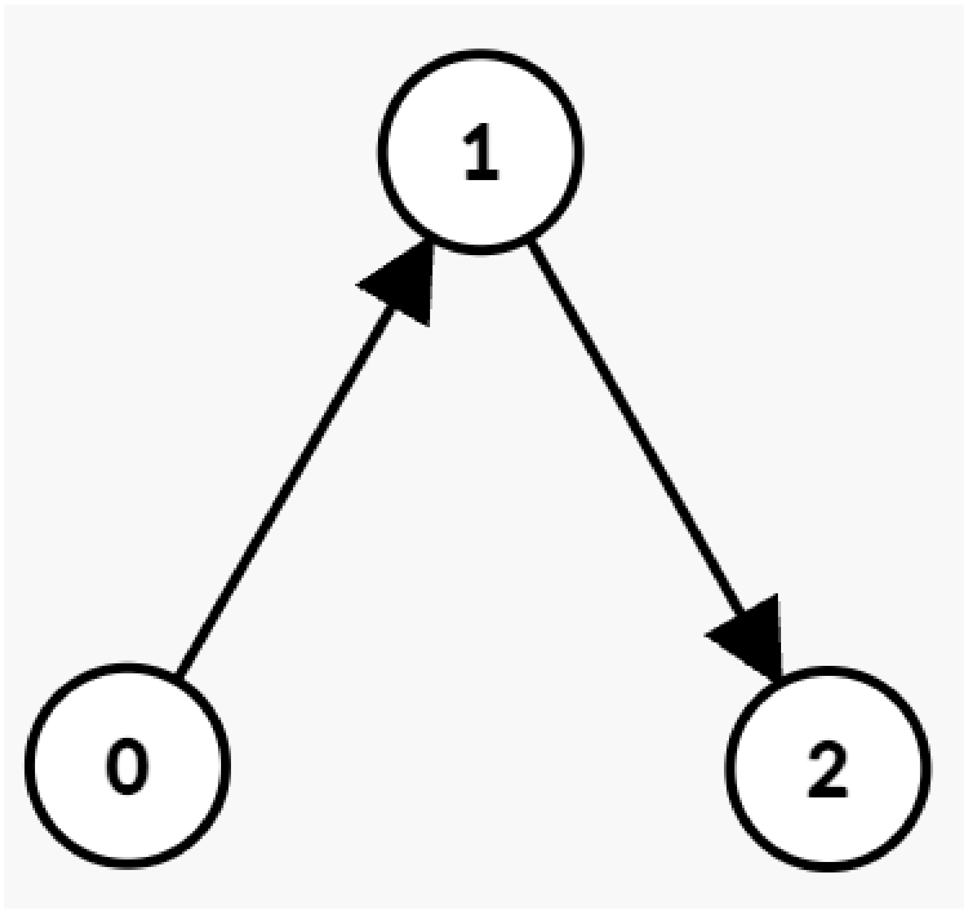
Input:

n = 3, edges = [[0,1,0,1],[1,2,2,5]]

Output:

3

Explanation:



The optimal path is:

At time

$t = 0$

, take the edge

$(0 \to 1)$

which is available from 0 to 1. You arrive at node 1 at time

$t = 1$

, then wait until

$t = 2$

.

At time

t =

2

, take the edge

$(1 \to 2)$

which is available from 2 to 5. You arrive at node 2 at time 3.

Hence, the minimum time to reach node 2 is 3.

Example 2:

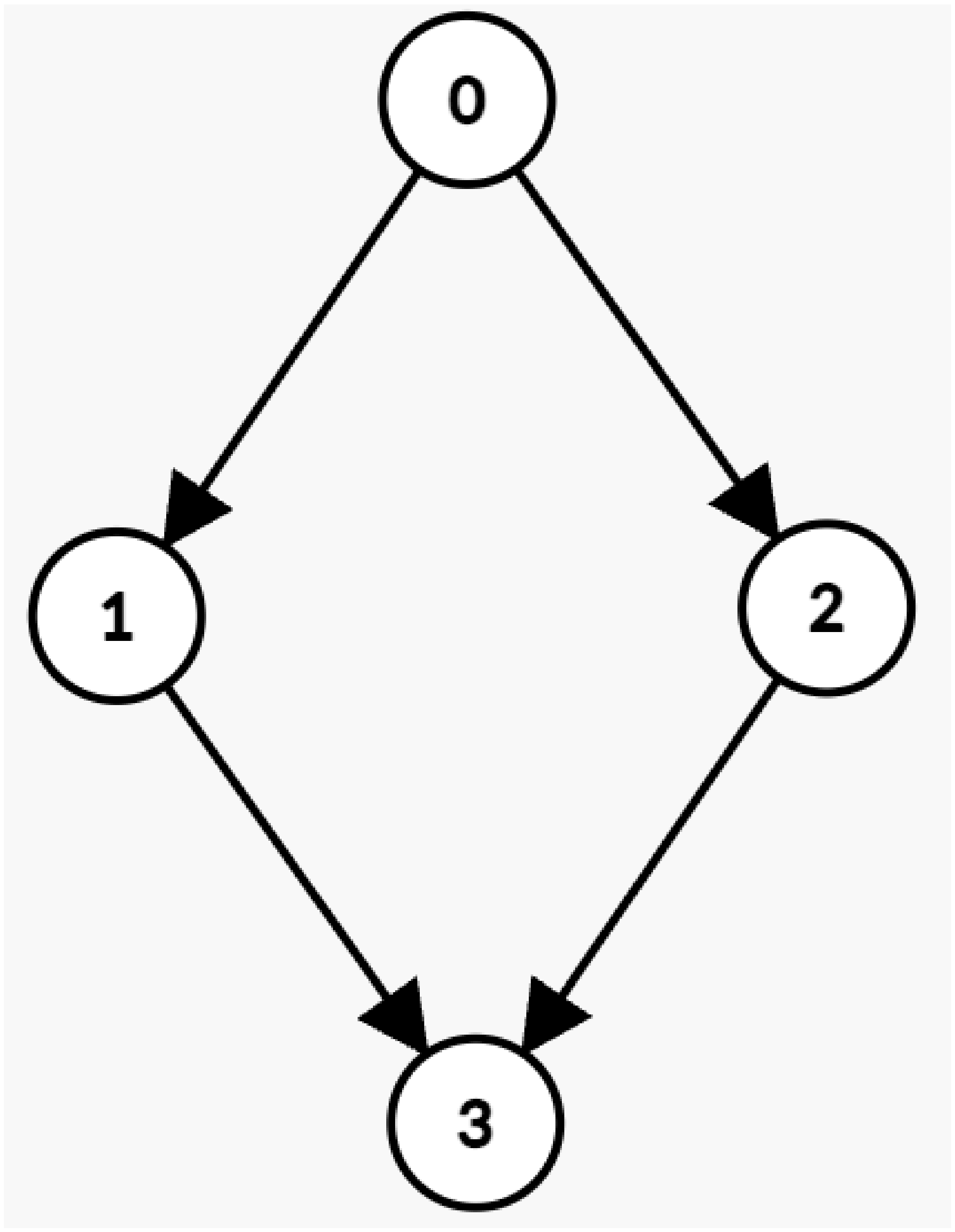
Input:

n = 4, edges = [[0,1,0,3],[1,3,7,8],[0,2,1,5],[2,3,4,7]]

Output:

5

Explanation:



The optimal path is:

Wait at node 0 until time

t = 1

, then take the edge

$(0 \to 2)$

which is available from 1 to 5. You arrive at node 2 at

$t = 2$

.

Wait at node 2 until time

$t = 4$

, then take the edge

$(2 \to 3)$

which is available from 4 to 7. You arrive at node 3 at

$t = 5$

.

Hence, the minimum time to reach node 3 is 5.

Example 3:

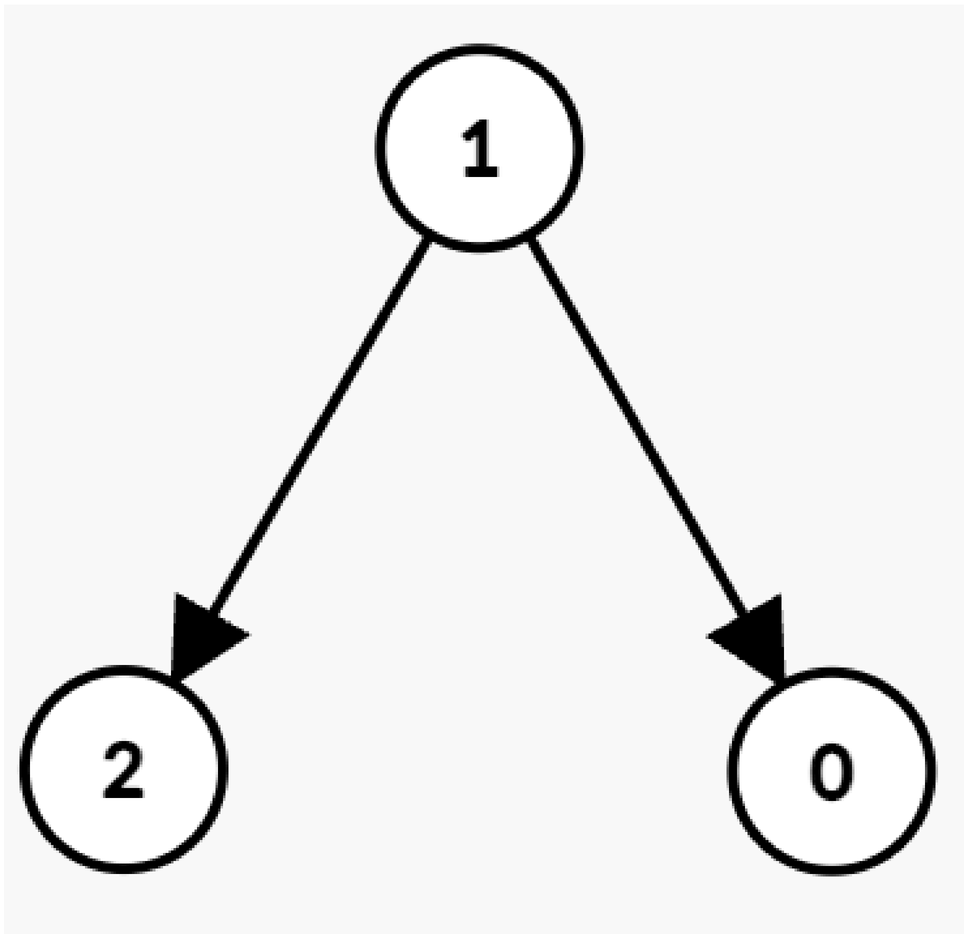
Input:

n = 3, edges = [[1,0,1,3],[1,2,3,5]]

Output:

-1

Explanation:



Since there is no outgoing edge from node 0, it is impossible to reach node 2. Hence, the output is -1.

Constraints:

1 <= n <= 10

5

0 <= edges.length <= 10

5

edges[i] == [u

i

, v

i

, start

i

, end

i

]

0 <= u

i

, v

i

<= n - 1

u

i

!= v

i

0 <= start

i

<= end

i

<= 10

9

Code Snippets

C++:

```
class Solution {
public:
int minTime(int n, vector<vector<int>>& edges) {

}
};
```

Java:

```
class Solution {
public int minTime(int n, int[][] edges) {

}
}
```

Python3:

```
class Solution:
def minTime(self, n: int, edges: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def minTime(self, n, edges):
"""
:type n: int
:type edges: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} edges
* @return {number}
*/
var minTime = function(n, edges) {

};
```

TypeScript:

```
function minTime(n: number, edges: number[][]): number {

};
```

C#:

```
public class Solution {
public int MinTime(int n, int[][] edges) {

}
}
```

C:

```
int minTime(int n, int** edges, int edgesSize, int* edgesColSize) {

}
```

Go:

```
func minTime(n int, edges [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minTime(n: Int, edges: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func minTime(_ n: Int, _ edges: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_time(n: i32, edges: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @return {Integer}
def min_time(n, edges)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @return Integer
*/
function minTime($n, $edges) {

}
}
```

Dart:

```
class Solution {
int minTime(int n, List<List<int>> edges) {



}
}
```

Scala:

```
object Solution {
def minTime(n: Int, edges: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_time(n :: integer, edges :: [[integer]]) :: integer
def min_time(n, edges) do

end
end
```

Erlang:

```
-spec min_time(N :: integer(), Edges :: [[integer()]]) -> integer().
min_time(N, Edges) ->
.
```

Racket:

```
(define/contract (min-time n edges)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Time to Reach Destination in Directed Graph
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* 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:
int minTime(int n, vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Time to Reach Destination in Directed Graph
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* 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 int minTime(int n, int[][] edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Time to Reach Destination in Directed Graph
Difficulty: Medium
Tags: array, graph, queue, heap

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 minTime(self, n: int, edges: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minTime(self, n, edges):
"""
:type n: int
:type edges: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Time to Reach Destination in Directed Graph
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* 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[][]} edges
* @return {number}
*/
var minTime = function(n, edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Time to Reach Destination in Directed Graph
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* 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 minTime(n: number, edges: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Time to Reach Destination in Directed Graph
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* 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 int MinTime(int n, int[][] edges) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Time to Reach Destination in Directed Graph
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* 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
```

```
*/

int minTime(int n, int** edges, int edgesSize, int* edgesColSize) {

}
```

Go Solution:

```
// Problem: Minimum Time to Reach Destination in Directed Graph
// Difficulty: Medium
// Tags: array, graph, queue, heap
//
// 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 minTime(n int, edges [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minTime(n: Int, edges: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minTime(_ n: Int, _ edges: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Time to Reach Destination in Directed Graph
// Difficulty: Medium
// Tags: array, graph, queue, heap
```

```
//
// 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 min_time(n: i32, edges: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @return {Integer}
def min_time(n, edges)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @return Integer
*/
function minTime($n, $edges) {

}
}
```

Dart Solution:

```
class Solution {
int minTime(int n, List<List<int>> edges) {

}
}
```

Scala Solution:

```
object Solution {
def minTime(n: Int, edges: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_time(n :: integer, edges :: [[integer]]) :: integer
def min_time(n, edges) do

end
end
```

Erlang Solution:

```
-spec min_time(N :: integer(), Edges :: [[integer()]]) -> integer().
min_time(N, Edges) ->
.
```

Racket Solution:

```
(define/contract (min-time n edges)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```