Problem 3650: Minimum Cost Path with Edge Reversals

Problem Information

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

Problem Description

You are given a directed, weighted graph with

n

nodes labeled from 0 to

n - 1

, and an array

edges

where

edges[i] = [u

i

, v

i

, w

i

]

represents a directed edge from node

u

i

to node

v

i

with cost

w

i

.

Each node

u

i

has a switch that can be used

at most once

: when you arrive at

u

i

and have not yet used its switch, you may activate it on one of its incoming edges

v

i

→ u

i

reverse that edge to

u

i

→ v

i

and

immediately

traverse it.

The reversal is only valid for that single move, and using a reversed edge costs

2 * w

i

.

Return the

minimum

total cost to travel from node 0 to node

n - 1

. If it is not possible, return -1.

Example 1:

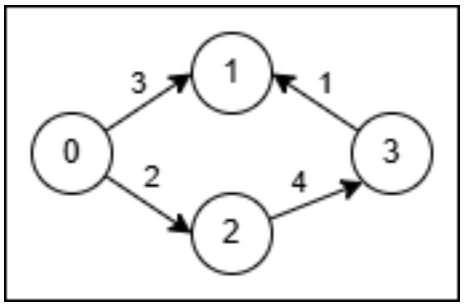
Input:

n = 4, edges = [[0,1,3],[3,1,1],[2,3,4],[0,2,2]]

Output:

5

Explanation:



Use the path

$0 \rightarrow 1$

(cost 3).

At node 1 reverse the original edge

$3 \rightarrow 1$

into

$1 \rightarrow 3$

and traverse it at cost

2 * 1 = 2

.

Total cost is

3 + 2 = 5

.

Example 2:

Input:

n = 4, edges = [[0,2,1],[2,1,1],[1,3,1],[2,3,3]]

Output:

3

Explanation:

No reversal is needed. Take the path

$0 \rightarrow 2$

(cost 1), then

$2 \rightarrow 1$

(cost 1), then

$1 \rightarrow 3$

(cost 1).

Total cost is

1 + 1 + 1 = 3

.

Constraints:

2 <= n <= 5 * 10

4

1 <= edges.length <= 10

5

edges[i] = [u

i

, v

i

, w

i

]

0 <= u

i

, v

i

<= n - 1

1 <= w

i

<= 1000

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

```
}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Cost Path with Edge Reversals
* 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 minCost(int n, vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Cost Path with Edge Reversals
* 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 minCost(int n, int[][] edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Cost Path with Edge Reversals
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 minCost(self, n: int, edges: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Cost Path with Edge Reversals
* 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 minCost = function(n, edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Cost Path with Edge Reversals
* 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 minCost(n: number, edges: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Cost Path with Edge Reversals
* 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 MinCost(int n, int[][] edges) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Cost Path with Edge Reversals
* 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 minCost(int n, int** edges, int edgesSize, int* edgesColSize) {

}
```

Go Solution:

```
// Problem: Minimum Cost Path with Edge Reversals
// 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 minCost(n int, edges [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Cost Path with Edge Reversals
// 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_cost(n: i32, edges: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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