Problem 2662: Minimum Cost of a Path With Special Roads

Problem Information

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

Problem Description

You are given an array

start

where

start = [startX, startY]

represents your initial position

(startX, startY)

in a 2D space. You are also given the array

target

where

target = [targetX, targetY]

represents your target position

(targetX, targetY)

.

The

cost

of going from a position

(x1, y1)

to any other position in the space

(x2, y2)

is

|x2 - x1| + |y2 - y1|

.

There are also some

special roads

. You are given a 2D array

specialRoads

where

specialRoads[i] = [x1

i

, y1

i

, x2

i

, y2

i

, cost

i

]

indicates that the

i

th

special road goes in

one direction

from

(x1

i

, y1

i

)

to

(x2

i

, y2

i

)

with a cost equal to

cost

i

. You can use each special road any number of times.

Return the

minimum

cost required to go from

(startX, startY)

to

(targetX, targetY)

.

Example 1:

Input:

start = [1,1], target = [4,5], specialRoads = [[1,2,3,3,2],[3,4,4,5,1]]

Output:

5

Explanation:

(1,1) to (1,2) with a cost of |1 - 1| + |2 - 1| = 1.

(1,2) to (3,3). Use

specialRoads[0]

with

the cost 2.

(3,3) to (3,4) with a cost of |3 - 3| + |4 - 3| = 1.

(3,4) to (4,5). Use

specialRoads[1]

with the cost

1.

So the total cost is 1 + 2 + 1 + 1 = 5.

Example 2:

Input:

start = [3,2], target = [5,7], specialRoads = [[5,7,3,2,1],[3,2,3,4,4],[3,3,5,5,5],[3,4,5,6,6]]

Output:

7

Explanation:

It is optimal not to use any special edges and go directly from the starting to the ending position with a cost |5 - 3| + |7 - 2| = 7.

Note that the

specialRoads[0]

is directed from (5,7) to (3,2).

Example 3:

Input:

start = [1,1], target = [10,4], specialRoads = [[4,2,1,1,3],[1,2,7,4,4],[10,3,6,1,2],[6,1,1,2,3]]

Output:

8

Explanation:

(1,1) to (1,2) with a cost of |1 - 1| + |2 - 1| = 1.

(1,2) to (7,4). Use

specialRoads[1]

with the cost

4.

(7,4) to (10,4) with a cost of |10 - 7| + |4 - 4| = 3.

Constraints:

start.length == target.length == 2

1 <= startX <= targetX <= 10

5

1 <= startY <= targetY <= 10

5

1 <= specialRoads.length <= 200

specialRoads[i].length == 5

startX <= x1

i

, x2

i

<= targetX

startY <= y1

i

, y2

i

<= targetY

1 <= cost

i

<= 10

5

Code Snippets

C++:

```
class Solution {
public:
int minimumCost(vector<int>& start, vector<int>& target, vector<vector<int>>&
specialRoads) {

}
};
```

Java:

```
class Solution {
public int minimumCost(int[] start, int[] target, int[][] specialRoads) {

}
}
```

Python3:

```
class Solution:
def minimumCost(self, start: List[int], target: List[int], specialRoads:
List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def minimumCost(self, start, target, specialRoads):
"""
:type start: List[int]
:type target: List[int]
:type specialRoads: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} start
* @param {number[]} target
* @param {number[][]} specialRoads
* @return {number}
*/
var minimumCost = function(start, target, specialRoads) {
```

```
};
```

TypeScript:

```
function minimumCost(start: number[], target: number[], specialRoads:
number[][]): number {

};
```

C#:

```
public class Solution {
public int MinimumCost(int[] start, int[] target, int[][] specialRoads) {

}
}
```

C:

```
int minimumCost(int* start, int startSize, int* target, int targetSize, int**
specialRoads, int specialRoadsSize, int* specialRoadsColSize) {

}
```

Go:

```
func minimumCost(start []int, target []int, specialRoads [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minimumCost(start: IntArray, target: IntArray, specialRoads:
Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func minimumCost(_ start: [Int], _ target: [Int], _ specialRoads: [[Int]]) ->
Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_cost(start: Vec<i32>, target: Vec<i32>, special_roads:
Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} start
# @param {Integer[]} target
# @param {Integer[][]} special_roads
# @return {Integer}
def minimum_cost(start, target, special_roads)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $start
* @param Integer[] $target
* @param Integer[][] $specialRoads
* @return Integer
*/
function minimumCost($start, $target, $specialRoads) {

}
}
```

Dart:

```
class Solution {
int minimumCost(List<int> start, List<int> target, List<List<int>>
specialRoads) {

}
}
```

Scala:

```
object Solution {
def minimumCost(start: Array[Int], target: Array[Int], specialRoads:
Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_cost(start :: [integer], target :: [integer], special_roads ::
[[integer]]) :: integer
def minimum_cost(start, target, special_roads) do

end
end
```

Erlang:

```
-spec minimum_cost(Start :: [integer()], Target :: [integer()], SpecialRoads
:: [[integer()]]) -> integer().
minimum_cost(Start, Target, SpecialRoads) ->
.
```

Racket:

```
(define/contract (minimum-cost start target specialRoads)
(-> (listof exact-integer?) (listof exact-integer?) (listof (listof
exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Cost of a Path With Special Roads
* 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 minimumCost(vector<int>& start, vector<int>& target, vector<vector<int>>&
specialRoads) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Cost of a Path With Special Roads
* 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 minimumCost(int[] start, int[] target, int[][] specialRoads) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Cost of a Path With Special Roads
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 minimumCost(self, start: List[int], target: List[int], specialRoads:
List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumCost(self, start, target, specialRoads):
"""
:type start: List[int]
:type target: List[int]
:type specialRoads: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Cost of a Path With Special Roads
* 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[]} start
* @param {number[]} target
* @param {number[][]} specialRoads
* @return {number}
```

```
*/
var minimumCost = function(start, target, specialRoads) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Cost of a Path With Special Roads
* 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 minimumCost(start: number[], target: number[], specialRoads:
number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Cost of a Path With Special Roads
* 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 MinimumCost(int[] start, int[] target, int[][] specialRoads) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Cost of a Path With Special Roads
* 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 minimumCost(int* start, int startSize, int* target, int targetSize, int**
specialRoads, int specialRoadsSize, int* specialRoadsColSize) {

}
```

Go Solution:

```
// Problem: Minimum Cost of a Path With Special Roads
// 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 minimumCost(start []int, target []int, specialRoads [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumCost(start: IntArray, target: IntArray, specialRoads:
Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumCost(_ start: [Int], _ target: [Int], _ specialRoads: [[Int]]) ->
```

```
Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Cost of a Path With Special Roads
// 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 minimum_cost(start: Vec<i32>, target: Vec<i32>, special_roads:
Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} start
# @param {Integer[]} target
# @param {Integer[][]} special_roads
# @return {Integer}
def minimum_cost(start, target, special_roads)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $start
* @param Integer[] $target
* @param Integer[][] $specialRoads
* @return Integer
*/
```

```
function minimumCost($start, $target, $specialRoads) {

}
}
```

Dart Solution:

```
class Solution {
int minimumCost(List<int> start, List<int> target, List<List<int>>
specialRoads) {

}
}
```

Scala Solution:

```
object Solution {
def minimumCost(start: Array[Int], target: Array[Int], specialRoads:
Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_cost(start :: [integer], target :: [integer], special_roads ::
[[integer]]) :: integer
def minimum_cost(start, target, special_roads) do

end
end
```

Erlang Solution:

```
-spec minimum_cost(Start :: [integer()], Target :: [integer()], SpecialRoads
:: [[integer()]]) -> integer().
minimum_cost(Start, Target, SpecialRoads) ->
.
```

Racket Solution:

```
(define/contract (minimum-cost start target specialRoads)
(-> (listof exact-integer?) (listof exact-integer?) (listof (listof
exact-integer?)) exact-integer?)
)
```