Problem 1575: Count All Possible Routes

Problem Information

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

Problem Description

You are given an array of

distinct

positive integers locations where

locations[i]

represents the position of city

i

. You are also given integers

start

,

finish

and

fuel

representing the starting city, ending city, and the initial amount of fuel you have, respectively.

At each step, if you are at city

i

, you can pick any city

j

such that

j != i

and

0 <= j < locations.length

and move to city

j

. Moving from city

i

to city

j

reduces the amount of fuel you have by

|locations[i] - locations[j]|

. Please notice that

|x|

denotes the absolute value of

x

.

Notice that

fuel

cannot

become negative at any point in time, and that you are

allowed

to visit any city more than once (including

start

and

finish

).

Return

the count of all possible routes from

start

to

finish

. Since the answer may be too large, return it modulo

10

9

+ 7

.

Example 1:

Input:

locations = [2,3,6,8,4], start = 1, finish = 3, fuel = 5

Output:

4

Explanation:

The following are all possible routes, each uses 5 units of fuel: 1 -> 3 1 -> 2 -> 3 1 -> 4 -> 3 1 -> 4 -> 2 -> 3

Example 2:

Input:

locations = [4,3,1], start = 1, finish = 0, fuel = 6

Output:

5

Explanation:

The following are all possible routes: 1 -> 0, used fuel = 1 1 -> 2 -> 0, used fuel = 5 1 -> 2 -> 1 -> 0, used fuel = 5 1 -> 0 -> 1 -> 0, used fuel = 3 1 -> 0 -> 1 -> 0 -> 1 -> 0, used fuel = 5

Example 3:

Input:

locations = [5,2,1], start = 0, finish = 2, fuel = 3

Output:

0

Explanation:

It is impossible to get from 0 to 2 using only 3 units of fuel since the shortest route needs 4 units of fuel.

Constraints:

2 <= locations.length <= 100

1 <= locations[i] <= 10

9

All integers in

locations

are

distinct

.

0 <= start, finish < locations.length

1 <= fuel <= 200

Code Snippets

C++:

```
class Solution {
public:
int countRoutes(vector<int>& locations, int start, int finish, int fuel) {
}
};
```

Java:

```
class Solution {
public int countRoutes(int[] locations, int start, int finish, int fuel) {

}
}
```

Python3:

```
class Solution:
def countRoutes(self, locations: List[int], start: int, finish: int, fuel:
int) -> int:
```

Python:

```
class Solution(object):
def countRoutes(self, locations, start, finish, fuel):
"""
:type locations: List[int]
:type start: int
:type finish: int
:type fuel: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} locations
* @param {number} start
* @param {number} finish
* @param {number} fuel
* @return {number}
*/
var countRoutes = function(locations, start, finish, fuel) {

};
```

TypeScript:

```
function countRoutes(locations: number[], start: number, finish: number,
fuel: number): number {

};
```

C#:

```
public class Solution {
public int CountRoutes(int[] locations, int start, int finish, int fuel) {

}
}
```

C:

```
int countRoutes(int* locations, int locationsSize, int start, int finish, int
fuel) {

}
```

Go:

```
func countRoutes(locations []int, start int, finish int, fuel int) int {

}
```

Kotlin:

```
class Solution {
fun countRoutes(locations: IntArray, start: Int, finish: Int, fuel: Int): Int
{

}
}
```

Swift:

```
class Solution {
func countRoutes(_ locations: [Int], _ start: Int, _ finish: Int, _ fuel:
Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_routes(locations: Vec<i32>, start: i32, finish: i32, fuel: i32)
-> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} locations
# @param {Integer} start
# @param {Integer} finish
# @param {Integer} fuel
# @return {Integer}
def count_routes(locations, start, finish, fuel)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $locations
* @param Integer $start
* @param Integer $finish
* @param Integer $fuel
* @return Integer
*/
function countRoutes($locations, $start, $finish, $fuel) {

}
}
```

Dart:

```
class Solution {
int countRoutes(List<int> locations, int start, int finish, int fuel) {

}
}
```

Scala:

```
object Solution {
def countRoutes(locations: Array[Int], start: Int, finish: Int, fuel: Int):
Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_routes(locations :: [integer], start :: integer, finish ::
integer, fuel :: integer) :: integer
def count_routes(locations, start, finish, fuel) do

end
end
```

Erlang:

```
-spec count_routes(Locations :: [integer()], Start :: integer(), Finish ::
integer(), Fuel :: integer()) -> integer().
count_routes(Locations, Start, Finish, Fuel) ->
.
```

Racket:

```
(define/contract (count-routes locations start finish fuel)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count All Possible Routes
* Difficulty: Hard
* Tags: array, dp
*
* 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 countRoutes(vector<int>& locations, int start, int finish, int fuel) {

}
};
```

Java Solution:

```
/**
* Problem: Count All Possible Routes
* Difficulty: Hard
* Tags: array, dp
*
* 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 countRoutes(int[] locations, int start, int finish, int fuel) {

}
}
```

Python3 Solution:

```
"""
Problem: Count All Possible Routes
Difficulty: Hard
Tags: array, dp
```

```
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 countRoutes(self, locations: List[int], start: int, finish: int, fuel:
int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countRoutes(self, locations, start, finish, fuel):
"""
:type locations: List[int]
:type start: int
:type finish: int
:type fuel: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count All Possible Routes
* Difficulty: Hard
* Tags: array, dp
*
* 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[]} locations
* @param {number} start
* @param {number} finish
* @param {number} fuel
* @return {number}
*/
var countRoutes = function(locations, start, finish, fuel) {

};
```

TypeScript Solution:

```
/**
* Problem: Count All Possible Routes
* Difficulty: Hard
* Tags: array, dp
*
* 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 countRoutes(locations: number[], start: number, finish: number,
fuel: number): number {

};
```

C# Solution:

```
/*
* Problem: Count All Possible Routes
* Difficulty: Hard
* Tags: array, dp
*
* 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 CountRoutes(int[] locations, int start, int finish, int fuel) {

}
}
```

C Solution:

```
/*
* Problem: Count All Possible Routes
* Difficulty: Hard
* Tags: array, dp
*
* 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 countRoutes(int* locations, int locationsSize, int start, int finish, int
fuel) {

}
```

Go Solution:

```
// Problem: Count All Possible Routes
// Difficulty: Hard
// Tags: array, dp
//
// 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 countRoutes(locations []int, start int, finish int, fuel int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countRoutes(locations: IntArray, start: Int, finish: Int, fuel: Int): Int
{

}
}
```

Swift Solution:

```
class Solution {
func countRoutes(_ locations: [Int], _ start: Int, _ finish: Int, _ fuel:
Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count All Possible Routes
// Difficulty: Hard
// Tags: array, dp
//
// 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 count_routes(locations: Vec<i32>, start: i32, finish: i32, fuel: i32)
-> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} locations
# @param {Integer} start
# @param {Integer} finish
# @param {Integer} fuel
# @return {Integer}
def count_routes(locations, start, finish, fuel)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $locations
* @param Integer $start
* @param Integer $finish
* @param Integer $fuel
* @return Integer
*/
function countRoutes($locations, $start, $finish, $fuel) {

}
}
```

Dart Solution:

```
class Solution {
int countRoutes(List<int> locations, int start, int finish, int fuel) {

}
}
```

Scala Solution:

```
object Solution {
def countRoutes(locations: Array[Int], start: Int, finish: Int, fuel: Int):
Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_routes(locations :: [integer], start :: integer, finish ::
integer, fuel :: integer) :: integer
def count_routes(locations, start, finish, fuel) do

end
end
```

Erlang Solution:

```
-spec count_routes(Locations :: [integer()], Start :: integer(), Finish ::
integer(), Fuel :: integer()) -> integer().
count_routes(Locations, Start, Finish, Fuel) ->
```

```
.
```

Racket Solution:

```
(define/contract (count-routes locations start finish fuel)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```