Problem 2361: Minimum Costs Using the Train Line

Problem Information

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

Problem Description

A train line going through a city has two routes, the regular route and the express route. Both routes go through the

same

n + 1

stops labeled from

0

to

n

. Initially, you start on the regular route at stop

0

.

You are given two

1-indexed

integer arrays

regular

and

express

, both of length

n

.

regular[i]

describes the cost it takes to go from stop

i - 1

to stop

i

using the regular route, and

express[i]

describes the cost it takes to go from stop

i - 1

to stop

i

using the express route.

You are also given an integer

expressCost

which represents the cost to transfer from the regular route to the express route.

Note that:

There is no cost to transfer from the express route back to the regular route.

You pay

expressCost

every

time you transfer from the regular route to the express route.

There is no extra cost to stay on the express route.

Return

a

1-indexed

array

costs

of length

n

, where

costs[i]

is the

minimum

cost to reach stop

i

from stop

0

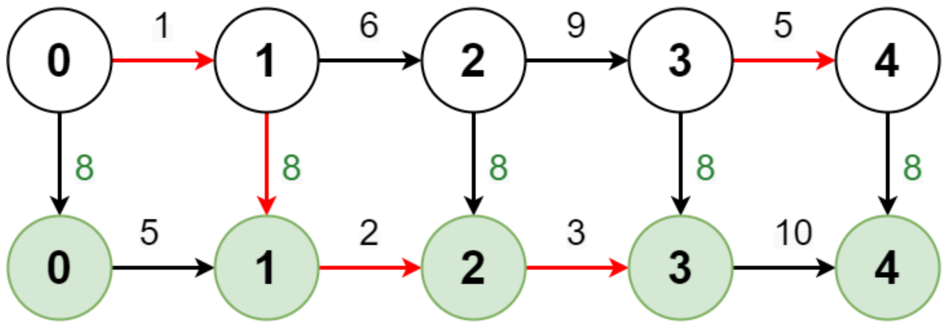
.

Note that a stop can be counted as

reached

from either route.

Example 1:



Input:

regular = [1,6,9,5], express = [5,2,3,10], expressCost = 8

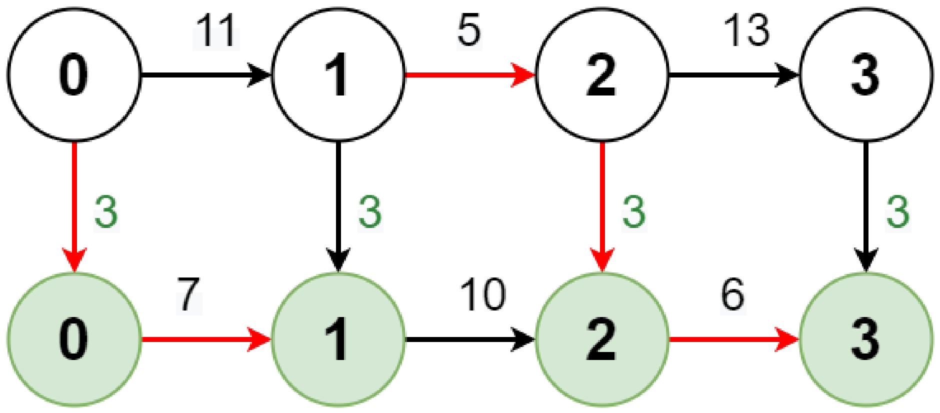
Output:

[1,7,14,19]

Explanation:

The diagram above shows how to reach stop 4 from stop 0 with minimum cost. - Take the regular route from stop 0 to stop 1, costing 1. - Take the express route from stop 1 to stop 2, costing 8 + 2 = 10. - Take the express route from stop 2 to stop 3, costing 3. - Take the regular route from stop 3 to stop 4, costing 5. The total cost is 1 + 10 + 3 + 5 = 19. Note that a different route could be taken to reach the other stops with minimum cost.

Example 2:



Input:

regular = [11,5,13], express = [7,10,6], expressCost = 3

Output:

[10,15,24]

Explanation:

The diagram above shows how to reach stop 3 from stop 0 with minimum cost. - Take the express route from stop 0 to stop 1, costing 3 + 7 = 10. - Take the regular route from stop 1 to stop 2, costing 5. - Take the express route from stop 2 to stop 3, costing 3 + 6 = 9. The total cost is 10 + 5 + 9 = 24. Note that the expressCost is paid again to transfer back to the express route.

Constraints:

n == regular.length == express.length

1 <= n <= 10

5

1 <= regular[i], express[i], expressCost <= 10

5

Code Snippets

C++:

```
class Solution {
public:
vector<long long> minimumCosts(vector<int>& regular, vector<int>& express,
int expressCost) {

}
};
```

Java:

```
class Solution {
public long[] minimumCosts(int[] regular, int[] express, int expressCost) {

}
}
```

Python3:

```
class Solution:
def minimumCosts(self, regular: List[int], express: List[int], expressCost:
int) -> List[int]:
```

Python:

```
class Solution(object):
def minimumCosts(self, regular, express, expressCost):
"""
:type regular: List[int]
:type express: List[int]
:type expressCost: int
:rtype: List[int]
```

```
"""
```

JavaScript:

```
/**
* @param {number[]} regular
* @param {number[]} express
* @param {number} expressCost
* @return {number[]}
*/
var minimumCosts = function(regular, express, expressCost) {

};
```

TypeScript:

```
function minimumCosts(regular: number[], express: number[], expressCost:
number): number[] {

};
```

C#:

```
public class Solution {
public long[] MinimumCosts(int[] regular, int[] express, int expressCost) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* minimumCosts(int* regular, int regularSize, int* express, int
expressSize, int expressCost, int* returnSize) {

}
```

Go:

```
func minimumCosts(regular []int, express []int, expressCost int) []int64 {

}
```

Kotlin:

```
class Solution {
fun minimumCosts(regular: IntArray, express: IntArray, expressCost: Int):
LongArray {

}
}
```

Swift:

```
class Solution {
func minimumCosts(_ regular: [Int], _ express: [Int], _ expressCost: Int) ->
[Int] {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_costs(regular: Vec<i32>, express: Vec<i32>, express_cost: i32)
-> Vec<i64> {

}
}
```

Ruby:

```
# @param {Integer[]} regular
# @param {Integer[]} express
# @param {Integer} express_cost
# @return {Integer[]}
def minimum_costs(regular, express, express_cost)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $regular
* @param Integer[] $express
* @param Integer $expressCost
* @return Integer[]
*/
function minimumCosts($regular, $express, $expressCost) {

}
}
```

Dart:

```
class Solution {
List<int> minimumCosts(List<int> regular, List<int> express, int expressCost)
{

}
}
```

Scala:

```
object Solution {
def minimumCosts(regular: Array[Int], express: Array[Int], expressCost: Int):
Array[Long] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_costs(regular :: [integer], express :: [integer], express_cost
:: integer) :: [integer]
def minimum_costs(regular, express, express_cost) do

end
end
```

Erlang:

```
-spec minimum_costs(Regular :: [integer()], Express :: [integer()],
ExpressCost :: integer()) -> [integer()].
minimum_costs(Regular, Express, ExpressCost) ->
.
```

Racket:

```
(define/contract (minimum-costs regular express expressCost)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer? (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Costs Using the Train Line
* 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:
vector<long long> minimumCosts(vector<int>& regular, vector<int>& express,
int expressCost) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Costs Using the Train Line
* 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 long[] minimumCosts(int[] regular, int[] express, int expressCost) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Costs Using the Train Line
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 minimumCosts(self, regular: List[int], express: List[int], expressCost:
int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumCosts(self, regular, express, expressCost):
"""
:type regular: List[int]
:type express: List[int]
:type expressCost: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Costs Using the Train Line
* 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[]} regular
* @param {number[]} express
* @param {number} expressCost
* @return {number[]}
*/
var minimumCosts = function(regular, express, expressCost) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Costs Using the Train Line
* 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 minimumCosts(regular: number[], express: number[], expressCost:
number): number[] {

};
```

C# Solution:

```
/*
* Problem: Minimum Costs Using the Train Line
* 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 long[] MinimumCosts(int[] regular, int[] express, int expressCost) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Costs Using the Train Line
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* minimumCosts(int* regular, int regularSize, int* express, int
expressSize, int expressCost, int* returnSize) {

}
```

Go Solution:

```
// Problem: Minimum Costs Using the Train Line
// 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 minimumCosts(regular []int, express []int, expressCost int) []int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minimumCosts(regular: IntArray, express: IntArray, expressCost: Int):
LongArray {

}
}
```

Swift Solution:

```
class Solution {
func minimumCosts(_ regular: [Int], _ express: [Int], _ expressCost: Int) ->
[Int] {

}
}
```

Rust Solution:

```
// Problem: Minimum Costs Using the Train Line
// 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 minimum_costs(regular: Vec<i32>, express: Vec<i32>, express_cost: i32)
-> Vec<i64> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} regular
# @param {Integer[]} express
# @param {Integer} express_cost
# @return {Integer[]}
def minimum_costs(regular, express, express_cost)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $regular
* @param Integer[] $express
* @param Integer $expressCost
* @return Integer[]
*/
function minimumCosts($regular, $express, $expressCost) {

}
}
```

Dart Solution:

```
class Solution {
List<int> minimumCosts(List<int> regular, List<int> express, int expressCost)
{

}
}
```

Scala Solution:

```
object Solution {
def minimumCosts(regular: Array[Int], express: Array[Int], expressCost: Int):
Array[Long] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_costs(regular :: [integer], express :: [integer], express_cost
:: integer) :: [integer]
def minimum_costs(regular, express, express_cost) do

end
end
```

Erlang Solution:

```
-spec minimum_costs(Regular :: [integer()], Express :: [integer()],
ExpressCost :: integer()) -> [integer()].
minimum_costs(Regular, Express, ExpressCost) ->
.
```

Racket Solution:

```
(define/contract (minimum-costs regular express expressCost)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer? (listof
exact-integer?))
)
```