Problem 983: Minimum Cost For Tickets

Problem Information

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

Problem Description

You have planned some train traveling one year in advance. The days of the year in which you will travel are given as an integer array

days

. Each day is an integer from

1

to

365

.

Train tickets are sold in

three different ways

:

a

1-day

pass is sold for

costs[0]

dollars,

a

7-day

pass is sold for

costs[1]

dollars, and

a

30-day

pass is sold for

costs[2]

dollars.

The passes allow that many days of consecutive travel.

For example, if we get a

7-day

pass on day

2

, then we can travel for

7

days:

2

,

3

,

4

,

5

,

6

,

7

, and

8

.

Return

the minimum number of dollars you need to travel every day in the given list of days

.

Example 1:

Input:

days = [1,4,6,7,8,20], costs = [2,7,15]

Output:

11

Explanation:

For example, here is one way to buy passes that lets you travel your travel plan: On day 1, you bought a 1-day pass for costs[0] = $2, which covered day 1. On day 3, you bought a 7-day pass for costs[1] = $7, which covered days 3, 4, ..., 9. On day 20, you bought a 1-day pass for costs[0] = $2, which covered day 20. In total, you spent $11 and covered all the days of your travel.

Example 2:

Input:

days = [1,2,3,4,5,6,7,8,9,10,30,31], costs = [2,7,15]

Output:

17

Explanation:

For example, here is one way to buy passes that lets you travel your travel plan: On day 1, you bought a 30-day pass for costs[2] = $15 which covered days 1, 2, ..., 30. On day 31, you bought a 1-day pass for costs[0] = $2 which covered day 31. In total, you spent $17 and covered all the days of your travel.

Constraints:

1 <= days.length <= 365

1 <= days[i] <= 365

days

is in strictly increasing order.

costs.length == 3

1 <= costs[i] <= 1000

Code Snippets

C++:

```
class Solution {
public:
int mincostTickets(vector<int>& days, vector<int>& costs) {

}
};
```

Java:

```
class Solution {
public int mincostTickets(int[] days, int[] costs) {

}
}
```

Python3:

```
class Solution:
def mincostTickets(self, days: List[int], costs: List[int]) -> int:
```

Python:

```
class Solution(object):
def mincostTickets(self, days, costs):
"""
:type days: List[int]
:type costs: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} days
* @param {number[]} costs
* @return {number}
*/
var mincostTickets = function(days, costs) {

};
```

TypeScript:

```
function mincostTickets(days: number[], costs: number[]): number {

};
```

C#:

```
public class Solution {
public int MincostTickets(int[] days, int[] costs) {

}
}
```

C:

```
int mincostTickets(int* days, int daysSize, int* costs, int costsSize) {

}
```

Go:

```
func mincostTickets(days []int, costs []int) int {

}
```

Kotlin:

```
class Solution {
fun mincostTickets(days: IntArray, costs: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func mincostTickets(_ days: [Int], _ costs: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn mincost_tickets(days: Vec<i32>, costs: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} days
# @param {Integer[]} costs
# @return {Integer}
def mincost_tickets(days, costs)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $days
* @param Integer[] $costs
* @return Integer
*/
function mincostTickets($days, $costs) {

}
}
```

Dart:

```
class Solution {
int mincostTickets(List<int> days, List<int> costs) {
```

```
}
}
```

Scala:

```
object Solution {
def mincostTickets(days: Array[Int], costs: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec mincost_tickets(days :: [integer], costs :: [integer]) :: integer
def mincost_tickets(days, costs) do

end
end
```

Erlang:

```
-spec mincost_tickets(Days :: [integer()], Costs :: [integer()]) ->
integer().
mincost_tickets(Days, Costs) ->
.
```

Racket:

```
(define/contract (mincost-tickets days costs)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Cost For Tickets
```

```
* Difficulty: Medium
* 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 mincostTickets(vector<int>& days, vector<int>& costs) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Cost For Tickets
* Difficulty: Medium
* 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 mincostTickets(int[] days, int[] costs) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Cost For Tickets
Difficulty: Medium
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 mincostTickets(self, days: List[int], costs: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def mincostTickets(self, days, costs):
"""
:type days: List[int]
:type costs: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Cost For Tickets
* Difficulty: Medium
* 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[]} days
* @param {number[]} costs
* @return {number}
*/
var mincostTickets = function(days, costs) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Cost For Tickets
* Difficulty: Medium
* 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 mincostTickets(days: number[], costs: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Cost For Tickets
* Difficulty: Medium
* 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 MincostTickets(int[] days, int[] costs) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Cost For Tickets
* Difficulty: Medium
* 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 mincostTickets(int* days, int daysSize, int* costs, int costsSize) {

}
```

Go Solution:

```
// Problem: Minimum Cost For Tickets
// Difficulty: Medium
// 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 mincostTickets(days []int, costs []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun mincostTickets(days: IntArray, costs: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func mincostTickets(_ days: [Int], _ costs: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Cost For Tickets
// Difficulty: Medium
// 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 mincost_tickets(days: Vec<i32>, costs: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} days
# @param {Integer[]} costs
# @return {Integer}
def mincost_tickets(days, costs)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $days
* @param Integer[] $costs
* @return Integer
*/
function mincostTickets($days, $costs) {

}
}
```

Dart Solution:

```
class Solution {
int mincostTickets(List<int> days, List<int> costs) {

}
}
```

Scala Solution:

```
object Solution {
def mincostTickets(days: Array[Int], costs: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec mincost_tickets(days :: [integer], costs :: [integer]) :: integer
def mincost_tickets(days, costs) do

end
end
```

Erlang Solution:

```
-spec mincost_tickets(Days :: [integer()], Costs :: [integer()]) ->
integer().
mincost_tickets(Days, Costs) ->
.
```

Racket Solution:

```
(define/contract (mincost-tickets days costs)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```