Problem 2391: Minimum Amount of Time to Collect Garbage

Problem Information

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

Problem Description

You are given a

0-indexed

array of strings

garbage

where

garbage[i]

represents the assortment of garbage at the

i

th

house.

garbage[i]

consists only of the characters

'M'

,

'P'

and

'G'

representing one unit of metal, paper and glass garbage respectively. Picking up

one

unit of any type of garbage takes

1

minute.

You are also given a

0-indexed

integer array

travel

where

travel[i]

is the number of minutes needed to go from house

i

to house

i + 1

.

There are three garbage trucks in the city, each responsible for picking up one type of garbage. Each garbage truck starts at house

0

and must visit each house

in order

; however, they do

not

need to visit every house.

Only

one

garbage truck may be used at any given moment. While one truck is driving or picking up garbage, the other two trucks

cannot

do anything.

Return

the

minimum

number of minutes needed to pick up all the garbage.

Example 1:

Input:

garbage = ["G","P","GP","GG"], travel = [2,4,3]

Output:

21

Explanation:

The paper garbage truck: 1. Travels from house 0 to house 1 2. Collects the paper garbage at house 1 3. Travels from house 1 to house 2 4. Collects the paper garbage at house 2 Altogether, it takes 8 minutes to pick up all the paper garbage. The glass garbage truck: 1. Collects the glass garbage at house 0 2. Travels from house 0 to house 1 3. Travels from house 1 to house 2 4. Collects the glass garbage at house 2 5. Travels from house 2 to house 3 6. Collects the glass garbage at house 3 Altogether, it takes 13 minutes to pick up all the glass garbage. Since there is no metal garbage, we do not need to consider the metal garbage truck. Therefore, it takes a total of 8 + 13 = 21 minutes to collect all the garbage.

Example 2:

Input:

garbage = ["MMM","PGM","GP"], travel = [3,10]

Output:

37

Explanation:

The metal garbage truck takes 7 minutes to pick up all the metal garbage. The paper garbage truck takes 15 minutes to pick up all the paper garbage. The glass garbage truck takes 15 minutes to pick up all the glass garbage. It takes a total of 7 + 15 + 15 = 37 minutes to collect all the garbage.

Constraints:

2 <= garbage.length <= 10

5

garbage[i]

consists of only the letters

'M'

,

'P'

, and

'G'

.

1 <= garbage[i].length <= 10

travel.length == garbage.length - 1

1 <= travel[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
int garbageCollection(vector<string>& garbage, vector<int>& travel) {

}
};
```

Java:

```
class Solution {
public int garbageCollection(String[] garbage, int[] travel) {

}
```

```
}
```

Python3:

```
class Solution:
def garbageCollection(self, garbage: List[str], travel: List[int]) -> int:
```

Python:

```
class Solution(object):
def garbageCollection(self, garbage, travel):
"""
:type garbage: List[str]
:type travel: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string[]} garbage
* @param {number[]} travel
* @return {number}
*/
var garbageCollection = function(garbage, travel) {

};
```

TypeScript:

```
function garbageCollection(garbage: string[], travel: number[]): number {

};
```

C#:

```
public class Solution {
public int GarbageCollection(string[] garbage, int[] travel) {

}
}
```

C:

```
int garbageCollection(char** garbage, int garbageSize, int* travel, int
travelSize) {

}
```

Go:

```
func garbageCollection(garbage []string, travel []int) int {

}
```

Kotlin:

```
class Solution {
fun garbageCollection(garbage: Array<String>, travel: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func garbageCollection(_ garbage: [String], _ travel: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn garbage_collection(garbage: Vec<String>, travel: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} garbage
# @param {Integer[]} travel
# @return {Integer}
def garbage_collection(garbage, travel)
```

```
end
```

PHP:

```
class Solution {

/**
* @param String[] $garbage
* @param Integer[] $travel
* @return Integer
*/
function garbageCollection($garbage, $travel) {

}
}
```

Dart:

```
class Solution {
int garbageCollection(List<String> garbage, List<int> travel) {

}
}
```

Scala:

```
object Solution {
def garbageCollection(garbage: Array[String], travel: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec garbage_collection(garbage :: [String.t], travel :: [integer]) ::
integer
def garbage_collection(garbage, travel) do

end
end
```

Erlang:

```
-spec garbage_collection(Garbage :: [unicode:unicode_binary()], Travel ::
[integer()]) -> integer().
garbage_collection(Garbage, Travel) ->
.
```

Racket:

```
(define/contract (garbage-collection garbage travel)
(-> (listof string?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Amount of Time to Collect Garbage
* Difficulty: Medium
* Tags: array, string, sort
*
* 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 garbageCollection(vector<string>& garbage, vector<int>& travel) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Amount of Time to Collect Garbage
* Difficulty: Medium
* Tags: array, string, sort
*
```

```
* 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 garbageCollection(String[] garbage, int[] travel) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Amount of Time to Collect Garbage
Difficulty: Medium
Tags: array, string, sort

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 garbageCollection(self, garbage: List[str], travel: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def garbageCollection(self, garbage, travel):
"""
:type garbage: List[str]
:type travel: List[int]
:rtype: int
"""
```

JavaScript Solution:

```javascript
/**
* Problem: Minimum Amount of Time to Collect Garbage
* Difficulty: Medium
* Tags: array, string, sort
*
* 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 {string[]} garbage
* @param {number[]} travel
* @return {number}
*/
var garbageCollection = function(garbage, travel) {

};
```

TypeScript Solution:

```typescript
/**
* Problem: Minimum Amount of Time to Collect Garbage
* Difficulty: Medium
* Tags: array, string, sort
*
* 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 garbageCollection(garbage: string[], travel: number[]): number {

};
```

C# Solution:

```csharp
/*
* Problem: Minimum Amount of Time to Collect Garbage
* Difficulty: Medium
* Tags: array, string, sort
*
```

```
* 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 GarbageCollection(string[] garbage, int[] travel) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Amount of Time to Collect Garbage
* Difficulty: Medium
* Tags: array, string, sort
*
* 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 garbageCollection(char** garbage, int garbageSize, int* travel, int
travelSize) {

}
```

Go Solution:

```
// Problem: Minimum Amount of Time to Collect Garbage
// Difficulty: Medium
// Tags: array, string, sort
//
// 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 garbageCollection(garbage []string, travel []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun garbageCollection(garbage: Array<String>, travel: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func garbageCollection(_ garbage: [String], _ travel: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Amount of Time to Collect Garbage
// Difficulty: Medium
// Tags: array, string, sort
//
// 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 garbage_collection(garbage: Vec<String>, travel: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} garbage
# @param {Integer[]} travel
# @return {Integer}
def garbage_collection(garbage, travel)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $garbage
* @param Integer[] $travel
* @return Integer
*/
function garbageCollection($garbage, $travel) {

}
}
```

Dart Solution:

```
class Solution {
int garbageCollection(List<String> garbage, List<int> travel) {

}
}
```

Scala Solution:

```
object Solution {
def garbageCollection(garbage: Array[String], travel: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec garbage_collection(garbage :: [String.t], travel :: [integer]) ::
integer
def garbage_collection(garbage, travel) do

end
end
```

Erlang Solution:

```
-spec garbage_collection(Garbage :: [unicode:unicode_binary()], Travel ::
[integer()]) -> integer().
garbage_collection(Garbage, Travel) ->
.
```

Racket Solution:

```
(define/contract (garbage-collection garbage travel)
(-> (listof string?) (listof exact-integer?) exact-integer?)
)
```