2391. Minimum Amount of Time to Collect Garbage

Medium **Array** String **Prefix Sum** 

# **Problem Description**

each house is provided to us in an array garbage where each string consists of letters 'M', 'P', and 'G', representing the types of garbage at that house. We also have information on travel times between houses, given by the travel array. travel[i] tells us the time it takes to travel from house i to house i+1.

In this problem, we are managing waste collection for a city with a uniquely efficient garbage collection system. There is a row of

houses numbered from 0, and at each house, there's a certain amount of metal (M), paper (P), and glass (G) waste. The waste at

The city has three garbage trucks, each designated to collect one type of garbage. All trucks start collecting from house 0, but

not every truck has to visit every house. The trucks move in order of the house numbers, but a truck only needs to visit a house if there is garbage for it to collect there. At any point in time, only one truck can be active, meaning the other two can't collect or move while one is in operation.

The goal is to determine the minimum total time required to collect all the garbage from all the houses. Intuition

To solve this problem, an efficient approach involves considering two key components. Firstly, we need to calculate the total time spent collecting garbage, which is the sum of garbage units since each unit takes one minute to be collected. Secondly, we must

## track the furthest each truck travels since the trucks only need to travel as far as the last house that has garbage for them.

• We start by initializing a variable ans to zero, which will accumulate the total time. We create a dictionary last to record the index of the last house that contains each type of garbage. • We iterate over the garbage array, adding the length of each string to ans, which corresponds to the collection time at that house. During this iteration, we also update the last dictionary for each type of garbage encountered.

• The travel time for each truck will only need to include the distance to the furthest house for its type of garbage. To simplify the calculation, we

create a cumulative sum array s of the travel array times. • We add to ans the cumulative travel times of the furthest house for each garbage type by summing the corresponding travel times from the s

The solution is implemented in the following steps:

- array.
- Finally, we return the resulting ans, which now represents the minimum number of minutes needed to collect all the garbage.

be the indices of the last house that contains the corresponding type of garbage.

travel time for each truck to reach the last house where its service is needed.

- The implementation of the solution can be broken down into multiple parts, aligning with the intuition detailed above. Here's a
- step-by-step approach to how the solution works: **Initialization of variables:**
- We initialize an answer variable ans to 0. This will accumulate the total time spent collecting garbage and traveling. A dictionary last is created to keep track of the last occurrence of each garbage type. The keys will be 'M', 'P', and 'G', and the values will

This means for each type of garbage, we're keeping the index of the last house where it has to be collected.

from the travel array. This accumulation includes the time it takes to travel from house 0 to every other house.

Solution Approach

 We iterate over the garbage array with enumerate() to have both the index i and the garbage string s. • We add the length of s to ans, since each garbage unit takes one minute to collect. • Within the same loop, we iterate through each character c in the string s. For each character, we update the last dictionary: last[c] = i.

We then add the total travel time for each garbage truck by summing the travel time (from the array s) up to the last house that requires its

• Finally, we return the accumulated answer ans, which now contains the total number of minutes spent collecting all the garbage and the

This solution takes advantage of simple iterations through the garbage array to determine the total collection time, and the use of

• We use Python's accumulate function from the itertools module with an initial parameter set to 0, to create a cumulative sum array s

**Calculating travel time:** 

Iterating through garbage array:

Let's illustrate the solution with a small example:

House 0 has "MPG" (Metal, Paper, Glass)

House 1 has "PP" (Paper)

House 3 has "M" (Metal)

House 2 has "GP" (Glass, Paper)

Following the solution approach:

**Initialization of variables:** 

ans is initialized to 0.

service: sum(s[i] for i in last.values()). This is added to the ans. **Return the total time:** 

Suppose we have a row of four houses numbered from 0 to 3, and the contents of their garbage bins are as follows:

a cumulative sum array to efficiently calculate the travel times required for each type of garbage truck. This approach ensures that all garbage is collected in the minimum amount of time possible by optimizing travel for each garbage truck. **Example Walkthrough** 

The garbage array for these houses would be: ["MPG", "PP", "GP", "M"]. The travel time between the houses is given by the travel array: [2, 4, 3] which indicates it takes 2 minutes to travel from House 0 to House 1, 4 minutes from House 1 to House 2, and 3 minutes from House 2 to House 3.

### Iterating through garbage array:

minutes).

The ans after adding travel times is ans += 3 + 6 + 6.

ans is initially 8, and after adding travel times, it becomes 23.

• last is initialized to {}, which will be used to store the last occurrences of 'M', 'P', and 'G'.

At House 0, "MPG" adds 3 minutes to ans, and last is updated to {'M': 0, 'P': 0, 'G': 0}.

At House 1, "PP" adds 2 minutes to ans, and last is updated to {'M': 0, 'P': 1, 'G': 0}.

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

The minimum amount of time needed to collect all the garbage.

# Calculate the total time to pick up all the garbage

# and track the last occurrence of each type of garbage

# Add the time to collect garbage at this index

# Dictionary to store the last index where 'G', 'P', or 'M' appears

# Update the last seen index for each garbage type in the bags

# Add the travel time to the last occurrence of each garbage type

int garbageCollection(vector<string>& garbage, vector<int>& travel) {

// Initialize total time taken to collect garbage to 0.

int garbageCount = garbage.size();

for (int i = 0; i < garbageCount; ++i) {</pre>

totalTime += garbage[i].size();

lastPosition[c - 'A'] = i;

for (char& c : garbage[i]) {

int cumulativeTravel[travelCount + 1];

for (int i = 1; i <= travelCount; ++i) {</pre>

cumulativeTravel[0] = 0;

int travelCount = travel.size();

int lastPosition[26] = {};

int totalTime = 0;

// Get the number of garbage bags and the number of travel times.

total\_time += sum(prefix\_sum\_travel[i] for i in last\_index.values())

# Initialize the total collection time

for i, bags in enumerate(garbage):

total\_time += len(bags)

for type\_garbage in bags:

 At House 2, "GP" adds 2 minutes to ans, and last is updated to {'M': 0, 'P': 2, 'G': 2}. At House 3, "M" adds 1 minute to ans, and last is updated to {'M': 3, 'P': 2, 'G': 2}. Calculating travel time:

• The cumulative sum array s from travel plus an initial 0 is calculated: [0, 2, 6, 9] (using the itertools.accumulate function).

○ We add the travel times for each truck to ans. For 'M', we add s[3] (3 minutes). For 'P', we add s[2] (6 minutes). For 'G', we add s[2] (6

Hence, the minimum total time required to collect all the garbage from all the houses is 23 minutes.

**Python** 

class Solution:

Returns:

total\_time = 0

last\_index = {}

Solution Implementation

**Return the total time:** 

Calculate the minimum amount of time to collect garbage. Args: garbage: A list of strings representing the content of garbage collected on each day. travel: A list of integers representing the amount of time needed to travel between adjacent houses.

#### last\_index[type\_garbage] = i # Calculate the prefix sum of travel time for quick lookup prefix\_sum\_travel = list(accumulate(travel, initial=0))

return total\_time

```
Java
class Solution {
    public int garbageCollection(String[] garbage, int[] travel) {
       // lastPositionOf holds the last position where each type of garbage appears
       int[] lastPositionOf = new int[26];
       int numberOfHouses = garbage.length;
        int totalTime = 0;
       // Calculate the total amount of garbage and update the last position for each garbage type
        for (int i = 0; i < numberOfHouses; ++i) {</pre>
            int garbageAtHouse = garbage[i].length();
            totalTime += garbageAtHouse; // Collect garbage at current house
            // Update the last position for each type of garbage in current house
            for (int j = 0; j < garbageAtHouse; ++j) {</pre>
                lastPositionOf[garbage[i].charAt(j) - 'A'] = i;
       // Calculate the prefix sum of travel time between the houses
       int[] prefixSumOfTravel = new int[travel.length + 1];
        for (int i = 0; i < travel.length; ++i) {</pre>
            prefixSumOfTravel[i + 1] = prefixSumOfTravel[i] + travel[i];
       // Add the travel time for each garbage truck to reach the last house of its type
        for (int lastPosition : lastPositionOf) {
            if(lastPosition != 0) { // Only add travel time if the garbage type was present
                totalTime += prefixSumOfTravel[lastPosition];
       // Return the total time to collect all garbage and return
       return totalTime;
```

// Initialize an array to keep track of the last occurrence position of each type of garbage ('A' to 'Z').

// Iterate through all the piles of garbage to perform initial collection and record the last occurrence.

// Add the size of the current pile to the total time (as it represents direct collection time).

if (i != 0) { // Non-zero position means the garbage type was present and needs to be collected.

// Update the last occurrence position for each type of garbage found in the current pile.

// Initialize a prefix sum array to store the cumulative travel time to reach each house.

cumulativeTravel[i] = cumulativeTravel[i - 1] + travel[i - 1];

```
// Add the necessary travel time for each type of garbage to the total time.
for (int i : lastPosition) {
```

class Solution:

Args:

Returns:

total\_time = 0

last\_index = {}

C++

public:

class Solution {

```
totalTime += cumulativeTravel[i];
       // Return the total time taken for garbage collection.
       return totalTime;
TypeScript
function garbageCollection(garbage: string[], travel: number[]): number {
   // The number of houses
   const numHouses = garbage.length;
   // The number of travel segments
   const numTravelSegments = travel.length;
   // Initialize the total time to 0
   let totalTime = 0;
   // Array to store the last house visited for each kind of garbage (26 represents 26 letters)
   const lastVisited = new Array(26).fill(0);
   // Traverse through garbage bins of each house
   for (let i = 0; i < numHouses; ++i) {</pre>
       // Add the garbage collection time which is equal to the garbage amount
       totalTime += garbage[i].length;
       // Record the last house visited for each kind of garbage
        for (const character of garbage[i]) {
            lastVisited[character.charCodeAt(0) - 'A'.charCodeAt(0)] = i;
   // Prefix sum array to store cumulative travel time between houses
   const cumulativeTravelTime = new Array(numTravelSegments + 1).fill(0);
   for (let i = 1; i <= numTravelSegments; ++i) {</pre>
        cumulativeTravelTime[i] = cumulativeTravelTime[i - 1] + travel[i - 1];
   // Calculate the extra travel time for each kind of garbage based on the last visited house
    for (const houseIndex of lastVisited) {
        totalTime += cumulativeTravelTime[houseIndex];
   // Return the total time spent
   return totalTime;
```

```
return total_time
Time and Space Complexity
```

**Time Complexity** 

#### garbage. Within this loop, there is another loop that iterates over each character in strings of garbage. In the worst case, the length of all garbage strings combined is M (assuming M to be the sum of lengths of all strings in garbage), so this part runs in O(M) time.

The time complexity of the given code can be analyzed based on its two main operations.

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

garbage: A list of strings representing the content of garbage collected on each day.

travel: A list of integers representing the amount of time needed to travel between adjacent houses.

Calculate the minimum amount of time to collect garbage.

The minimum amount of time needed to collect all the garbage.

# Calculate the total time to pick up all the garbage

# and track the last occurrence of each type of garbage

# Add the time to collect garbage at this index

# Calculate the prefix sum of travel time for quick lookup

# Add the travel time to the last occurrence of each garbage type

total\_time += sum(prefix\_sum\_travel[i] for i in last\_index.values())

prefix\_sum\_travel = list(accumulate(travel, initial=0))

# Dictionary to store the last index where 'G', 'P', or 'M' appears

# Update the last seen index for each garbage type in the bags

# Initialize the total collection time

for i, bags in enumerate(garbage):

total\_time += len(bags)

for type\_garbage in bags:

last\_index[type\_garbage] = i

- The second part of the code uses the built-in function accumulate() to create list s from travel, which runs in O(T) time where T is the number of elements in travel. The summing up of s[i] for i in last values() has a worst case of O(K) where K is the number of unique characters (i.e., 'G',
- Overall, the time complexity of the function is O(N + M + T + K) which simplifies to O(N + M + T) since K is small and constant.

'M', 'P' in the garbage collection problem). However, since K is small and constant (at most 3 in this specific scenario), this can

The first loop iterates over the list garbage once, making its time complexity O(N) where N is the number of elements in

**Space Complexity** 

## The last dictionary potentially holds last indices for each type of garbage. Since the input problem is limited to three types of garbage at most, this dictionary's size is O(K), which is considered O(1) because K is small and constant.

As for space complexity:

be considered 0(1).

- The list s is created using the accumulate() function on the travel list, so its space is O(T) where T is the number of elements in travel.
- Therefore, the space complexity of the code is O(T) because the constant space for last is negligible.