Prefix Sum

are from each other, summing up these distances for each unique value.

Hash Table

Problem Description

Medium Array

calculate the sum of distances (intervals) between arr[i] and all other elements in arr that have the same value as arr[i]. In other words, for every element that matches arr[i], we sum up the absolute differences in their indices. The output should be an array intervals of the same length as arr where each intervals[i] contains the computed sum for arr[i]. To put it simply, if an element appears multiple times in the array, we are looking to find out how far apart all of those occurrences

The problem provides us with an array arr where each element may occur more than once. For each element arr[i], we want to

Intuition

this dictionary are the unique values, and the values are lists containing the indices where the key occurs.

The overarching idea is to use these lists to calculate the intervals in an efficient manner. A neat observation is that when calculating the sum of intervals for a particular number in arr, we notice that if the indices are sorted, each index v[i] will contribute (i * v[i]) -((m-i)*v[i]) to the total sum, where i is the current position within the occurrences of the number in arr, and m is the total

number of occurrences. Put differently, for any given occurrence, the distances from it to all previous occurrences will each increase

The solution approach makes use of a dictionary (or equivalent) to store occurrences of each unique value in the given array. Keys in

by v[i] - v[i - 1] when moving to the next occurrence, and the distances to all next occurrences will each decrease by the same amount. The algorithm's efficiency comes from leveraging this pattern to update the sum iteratively while traversing through the list of occurrences of each unique value, thereby avoiding the need to recompute the sum from scratch for every element. Here is a step-by-step breakdown of the solution:

1. Create a dictionary (d) to group elements of arr by their values, mapping each value to a list of indices at which it occurs in arr.

2. Initialize an answer array (ans) of size n with zeros to hold the sum of intervals for each corresponding element in arr. 3. Iterate through the grouped values in the dictionary: For the current group, calculate the initial interval sum for the first occurrence.

solution achieves efficient computation of the intervals.

Iterate through the indices in the current group:

 Calculate the difference between the current index and the previous one. Update the running total (val), adding the interval increments for indices before the current one and subtracting for

By organizing the elements in this way and cleverly calculating the contributions of each element's occurrences to the sum, the

- indices after. Store the running total in the corresponding position in the ans array. 4. Return the ans array, which now contains the sum of intervals for each element in arr.
- Solution Approach

be used to store the computed sum of intervals for each element within arr.

The solution approach can be broken down into the following steps:

the corresponding value is a list of indices at which this key appears. The enumerate function is used to get both the index i and the value v as we iterate through arr. This step essentially groups the occurrences of each element for easy access.

1. Group Elements by Value: The defaultdict(list) is used to create a dictionary d where each key is a value in the array arr, and

```
d[v].append(i)
2. Prepare the Answer Array: An array ans of size n (where n is the length of the input array arr) is initialized with zeros. This will
```

1 ans = [0] * n

1 d = defaultdict(list)

2 for i, v in enumerate(arr):

3. Iterate Over the Groups: Then, we iterate through all lists of indices (all values) present in d. For each list v, we calculate the sum

going backwards in the list of occurrences.

distances in this manner.

1 arr = [1, 3, 1, 3, 2, 1]

We iterate over the entries in d:

5]) - (0 * 3) = 7.

Within the group:

The final ans array will be:

1 ans = [0, 2, 5, 2, 0, 8]

 $1 d = {$

2 1: [0, 2, 5],

1 for v in d.values(): m = len(v)val = sum(v) - v[0] * m4. Compute the Intervals Using a Running Total: As we iterate over the indices in v:

• We incrementally update val by adding the cumulative distance going forward and subtracting the cumulative distance

of intervals for the first element in the list as val. Here, m refers to the total number of occurrences in the list v.

• The final sum for each occurrence p of the value is then saved in ans [p]. 1 for i, p in enumerate(v): delta = v[i] - v[i - 1] if i >= 1 else 0 val += i * delta - (m - i) * deltaans[p] = val

5. Return the Result: The ans array is returned, which holds the sum of intervals for each element of the original array arr.

The solution approach combines dictionary mapping with efficient handling of sequential updates, thereby avoiding redundant

calculations. This reduces the complexity considerably compared to a naive approach that might involve nested loops. The use of a

The delta represents the difference between the current index and the previous one, which affects the cumulative

running total (incremental adjustment of the val) to calculate the distances as we iterate through each list of occurrences is key to the efficiency of the solution. Example Walkthrough

First, we want to group the elements by value. We will get a dictionary d that looks like this:

3: [1, 3], 2: [4]

Let's consider a small example to illustrate the solution approach. Suppose we have the following array arr:

Here, the keys of the dictionary are the unique values from arr, and the values are lists containing the indices at which the corresponding key occurs in arr. The answer array ans will initially be all zeros:

• For the group with key 1, v is [0, 2, 5]. The length of v (m) is 3. The initial interval sum val is calculated as val = sum([0, 2,

○ The difference (delta) between the first and second occurrence of 1 is 2 - 0 = 2. Update val to val + (1 * 2) - (2 * 2)

○ The difference between the second and third occurrence is 5 - 2 = 3. Update val to val + (2 * 3) - (1 * 3) = 5 + 6 -

Within the group:

3 = 8. ans [5] is set to 8.

= 7 + 2 - 4 = 5. ans [2] is set to 5.

1 ans = [0, 0, 0, 0, 0, 0] # since there are 6 elements in arr

 \circ The difference (delta) between the first and second occurrence of 3 is 3 - 1 = 2. Update val to val + (1 * 2) - (1 * 2) = 2 + 2 - 2 = 2. ans [1] and ans [3] are both set to 2 since this is the total distance for both occurrences.

• Finally, for the group with key 2, v is [4]. Because there is only one occurrence, the distance is zero. ans [4] remains 0.

• Moving on to the group with key 3, v is [1, 3]. Here, m is 2 and initial val is val = sum([1, 3]) - (1 * 2) = 4 - 2 = 2.

the problem by using a dictionary to map values to indices and by calculating the sum incrementally. The solution leverages the spatial ordering of index occurrences, making use of a running total instead of performing redundant

In this final array, each ans [i] is the sum of intervals between the occurrences of arr[i] in the original array. This efficiently solves

calculations, which would happen if doing this for each element individually without considering the previously calculated distances.

Length of the array n = len(arr)# Populate the dictionary with the indices for each value 10 for i, value in enumerate(arr): 11

Python Solution

class Solution:

12

14

15

16

17

18

10

11

12

13

14

15

16 17

18

19

20

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

42

41 }

from collections import defaultdict

answer = [0] * n

index_map = defaultdict(list)

index_map[value].append(i)

Initialize the answer list with zeros

def getDistances(self, arr: List[int]) -> List[int]:

Dictionary to store the indices of each unique value in the array

Loop through the dictionary to calculate the distances for each value

// A map to store the indices of each unique value in the array

indexMap.computeIfAbsent(arr[i], k -> new ArrayList<>()).add(i);

Map<Integer, List<Integer>> indexMap = new HashMap<>();

// Iterate over each list of indices for all unique values

// Calculate the sum of all indices for the current value

// Initialize the sum for the first location of this value

// Calculate the total distance for each index occurrence

sumIndices += i * delta - (size - i) * delta;

result[indices.get(i)] = sumIndices;

// Difference between the current and previous index

int delta = i >= 1 ? indices.get(i) - indices.get(i - 1) : 0;

// Update sumIndices to include distances of current index from others

unordered_map<int, vector<int>> indexMap; // Maps each unique value to the indices it appears at

// Set the total distance for the current index in the result array

// Populate the map with indices for each value

for (List<Integer> indices : indexMap.values()) {

sumIndices -= (size * indices.get(∅));

for (int i = 0; i < size; ++i) {

vector<long long> getDistances(vector<int>& arr) {

// Index each element in the map

for (int i = 0; i < size; ++i) {

// Index each element in the map

for (let i = 0; i < size; ++i) {</pre>

for (let itemKey in indexMap) {

let sum: bigint = BigInt(0);

for (let index of indices) {

sum += BigInt(index);

answer[indices[i]] = sum;

for (let i = 0; i < indices.length; ++i) {</pre>

if (!indexMap[arr[i]]) {

indexMap[arr[i]] = [];

indexMap[arr[i]].push_back(i);

// Initialize the answer array with the size of the input array,

let answer: bigint[] = new Array<bigint>(size).fill(BigInt(0));

// Iterate over each element in the map to compute the sum of distances

// Calculate the initial sum of distances from the first occurrence

let delta: number = i >= 1 ? indices[i] - indices[i - 1] : 0;

// and subtract it for the remaining occurrences

sum += BigInt(i * delta - (count - i) * delta);

// Update the sum: add the difference for the previous occurrences,

// Assign the sum to the index of the current occurrence in the answer array

// filled with BigInts representing zero for each element

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

int size = arr.size(); // Size of the input array

int length = arr.length;

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

long[] result = new long[length];

int size = indices.size();

for (int index : indices) {

sumIndices += index;

// Array to store the result

long sumIndices = 0;

```
for indices in index_map.values():
               # Calculate the initial total distance for the first occurrence
19
20
               m = len(indices)
               total_distance = sum(indices) - indices[0] * m
21
22
23
               # Calculate the distances for all other occurrences of the same value
24
               for i, index in enumerate(indices):
25
                   # Calculate the change in distance when moving from one occurrence to the next
                   delta = indices[i] - indices[i - 1] if i >= 1 else 0
26
27
                   # Update the total distance considering the number of elements to the left and right
28
                   total_distance += i * delta - (m - i) * delta
29
                   # Store the total distance at the current index in the answer list
30
                   answer[index] = total_distance
31
32
           return answer
33
Java Solution
   class Solution {
       public long[] getDistances(int[] arr) {
```

C++ Solution 1 class Solution { 2 public:

return result;

```
indexMap[arr[i]].push_back(i);
10
11
12
           vector<long long> answer(size); // Initialize the answer vector with the size of the input array
13
           // Iterate over each element in the map to compute the sum of distances
           for (auto& item : indexMap) {
14
15
               auto& indices = item.second; // References the vector of indices for this element value
               int count = indices.size(); // Number of occurrences of this element value
16
               long long sum = 0;
17
18
               // Calculate the initial sum of distances from the first occurrence
19
               for (int index : indices) {
20
21
                   sum += index;
22
23
               sum -= count * indices[0]; // Subtract distances as though all elements were at the first occurrence
24
25
               // Loop over indices to distribute the sum to the appropriate position in the answer vector
               for (int i = 0; i < indices.size(); ++i) {</pre>
26
27
                   // Calculate the difference between the positions of the current and previous occurrences
28
                   int delta = i >= 1 ? indices[i] - indices[i - 1] : 0;
29
                   // Update the sum: add the difference for the previous occurrences,
30
                   // and subtract it for the remaining occurrences
                   sum += i * delta - (count - i) * delta;
31
                   // Assign the sum to the index of the current occurrence in the answer vector
                   answer[indices[i]] = sum;
33
34
35
36
           return answer; // Return the computed sum of distances vector
37
38 };
39
Typescript Solution
  1 // Define the type for mapping each unique value to the indices it appears at
  2 let indexMap: Record<number, number[]> = {};
  4 // Define the method signature with the appropriate TypeScript types
    function getDistances(arr: number[]): bigint[] {
         indexMap = {}; // Reset the indexMap for each call
         const size: number = arr.length; // Size of the input array
  8
```

42 43 return answer; // Return the computed sum of distances array 44 45 } 46

Time and Space Complexity

distances for each unique element in the array.

Time Complexity: 1. Constructing the dictionary d where each value is a list of indices has a time complexity of O(n), where n is the length of the array arr. This is because we iterate over the array once.

2. The outer loop for v in d.values(): has at most n iterations in the worst-case scenario where all array elements are unique.

3. Inside this loop, we calculate the initial val for each group of identical elements. The summation sum(v) is 0(m), where m is the

number of times a distinct element appears in the array. The term v[0] * m is 0(m) since it involves m multiplications. As such,

The given Python code involves creating a dictionary to group indices of identical elements and then computing the sum of

let indices: number[] = indexMap[itemKey]; // References the array of indices for this element value

sum -= BigInt(count * indices[0]); // Subtract distances as though all elements were at the first occurrence

let count: number = indices.length; // Number of occurrences of this element value

// Loop over indices to distribute the sum to the appropriate position in the answer array

// Calculate the difference between the positions of the current and previous occurrences

this part has a time complexity of O(m) for each unique element.

- 4. The inner loop for i, p in enumerate(v): iterates m times (where m is the size of the list v). The calculation within this loop is 0(1), as it consists of simple arithmetic operations. Therefore, the time complexity for the inner loop is 0(m). Combining these observations, the overall time complexity of the algorithm depends on the distribution of elements in the array. In
- average and expected case, where elements are repeated, the time complexity would be 0(n + k * m) where k is the number of unique elements. However, since the sum of all m for each unique element is n, the average-case time complexity remains linear, or 0(n).

the worst-case scenario, where all elements are unique, the time complexity tends to O(n * m) which simplifies to O(n). In the

Space Complexity:

1. The space complexity of the dictionary d can be up to O(n) in the case where all elements are distinct since we store an index

- for each element. 2. The ans list also takes up O(n) space since it stores the result for each element in the arr.
- 3. Auxiliary space complexity is 0(1) as there are no other data structures that grow with input size; only a fixed number of variables are used.

Therefore, the space complexity of the code is O(n) accounting for the space required for the dictionary d and the answer array ans.