2055. Plates Between Candles String] Medium Array Binary Search Prefix Sum Leetcode Link

Problem Description In this problem, we are given a string s representing a table where '*' characters signify plates and '||' characters represent

[left_i, right_i]. This pair indicates a specific substring of s — from index left_i to right_i, inclusive. The task is to determine how many plates are situated between candles within these substrings. It's important to note that a plate is only counted if it has at least one candle to its left and one candle to its right within the boundary of the substring defined by the

candles placed on the table. Alongside the string, we receive an array of queries queries, with each query given as a pair of indices

query. Our goal is to return an array of integers where each element corresponds to the count of plates between candles for each query in queries.

Intuition The brute force approach to solving this would entail scanning the substring for each query to count the plates between candles,

locate the nearest candles for any given plate quickly.

Step 2: Leftmost and Rightmost Candle Arrays

To handle this problem efficiently, we employ a prefix sum array and two additional arrays to track the positions of the leftmost and rightmost candles relative to each position.

which could lead to a significant amount of repeated work, especially with overlapping or similar queries.

The prefix sum array, presum, gets constructed such that presum[i] stores the sum of plates encountered from the start of the string up to index 1 - 1. This allows us to calculate the number of plates between any two candle positions quickly by subtracting the

appropriate prefix sums. However, before we can use this array, we need to know where the boundaries are — the closest candles to the left and right of

each plate. For that, we introduce two arrays: left and right. As we iterate through the string, we update left[i] with the last seen

candle position on the left of index i and right[i] with the upcoming candle position on the right of index i. These arrays help us

With these arrays ready, we can now process each query. We take the given query range [1, r] and use the right array to find the first candle to the right of 1 and the left array to find the last candle to the left of r. If these exist and are properly positioned (the right of 1 is less than the left of r), we've identified the boundaries of the relevant candle-plate-candle sequence. Then we can use

the differences of the prefix sums to find the number of plates between these two candles. This approach allows us to process queries efficiently since we reduced the problem to constant-time look-ups and a single subtraction operation per query, after the initial setup of the prefix sum and left/right boundary arrays.

Step 1: Prefix Sum Array The first step is constructing the presum array. It accumulates the count of plates ('*') as we iterate through the string s. presum[i] is computed as presum[i-1] + (s[i-1] == '*'). This setup will later enable us to quickly calculate the number of plates

The solution approach breaks down into several key steps corresponding to the following algorithmic patterns and data structures:

last encountered candle. • Conversely, scanning from right to left, right[i] is set to i if s[i] is a candle. Otherwise, it retains the position of the next

candle to the right.

between any two indices.

Solution Approach

As we scan the string from left to right, each left[i] is updated to i if s[i] is a candle. If not, it carries over the position of the

We initialize two arrays, left and right, to record the positions of the nearest candles on the left and right sides of each index in s.

The purpose of these arrays is to determine, for each plate, the positions of the closest candles that potentially enclose it in a valid

sequence.

queries.

Step 3: Query Processing For each query [1, r] in queries:

We find the closest candle to the right of 1 using right[1] and the closest candle to the left of r using left[r]. Let's refer to

We then check if i and j form a valid boundary — that is, if i is before j. If not, no plates can be counted for this query.

The algorithm leverages prefix sums, array scanning, and good use of boundary tracking to optimize the repeated calculations

involved in answering each query. This results in an efficient solution that only requires a linear scan of the input string and the

This result is stored in the ans array at the position corresponding to the current query.

The final output is the ans array, which contains the count of plates between candles for each query.

If they do form a valid boundary, the number of plates between these candles is the difference between the presum just after the left candle and just before the right candle: presum[j] - presum[i + 1].

Example Walkthrough

candles ('|') for each query range.

these positions as i and j, respectively.

Step 1: Prefix Sum Array We create the presum array as follows:

Step 2: Leftmost and Rightmost Candle Arrays

Next, we construct the left and right arrays:

• left = [-1, -1, 2, 2, 2, 2, 6, 6, 6, 6]

Step 3: Query Processing

Now we answer each query:

2. For the query [0, 9], we check:

positions 2 and 6.

Let's take an example to illustrate the solution approach:

· s = " * ** " • presum = [0, 0, 0, 1, 2, 3, 3, 4, 5, 5] Each element in presum indicates the total number of plates to the left of it.

Given a string s = " /**/", and queries queries = [[2, 5], [0, 9]]. We want to count the number of plates (') that are located between

- right = [2, 2, 2, 6, 6, 6, 6, 6, 9, 9, 9]The left array is constructed by scanning from left to right and marking the latest position of a candle. Similarly, right is constructed by scanning from right to left.
- 1. For the query [2, 5], we check:

Nearest right candle from 0: right[0] = 2

We have a valid boundary because 2 < 6.

Get the length of the string 's'

 $prefix_sum = [0] * (n + 1)$

for i, char in enumerate(s):

closest_left_candle = [0] * n

for i, char in enumerate(s):

if char == '|':

if s[i] == '|':

answer = [0] * len(queries)

Process each query

return answer

closest_right_candle = [0] * n

for i in range(n - 1, -1, -1):

left_candle_index = right_candle_index = -1

left_candle_index = i

right_candle_index = i

Update the index when a candle is found

closest_left_candle[i] = left_candle_index

Populate the closest_right_candle array in reverse

closest_right_candle[i] = right_candle_index

for index, (left, right) in enumerate(queries):

int[] nearestLeftCandle = new int[length];

if (s.charAt(i) == '|') {

if (s.charAt(i) == '|') {

// Array to hold the results of queries

int[] answer = new int[queries.length];

for (int k = 0; k < queries.length; ++k) {</pre>

int[] nearestRightCandle = new int[length];

// Calculate the nearest left candle positions

nearestLeftCandle[i] = lastCandleIndex;

// Calculate the nearest right candle positions

nearestRightCandle[i] = nextCandleIndex;

for (int i = 0, lastCandleIndex = -1; i < length; ++i) {

lastCandleIndex = i; // Update last seen candle

for (int i = length - 1, nextCandleIndex = -1; i >= 0; --i) {

int startIndex = nearestRightCandle[queries[k][0]];

int endIndex = nearestLeftCandle[queries[k][1]];

// Function to calculate the number of plates between candles

prefixSum[i + 1] = prefixSum[i] + (s[i] == '*');

// Find the nearest left candle for each position

if (s[i] == '|') lastCandleIndex = i;

if (s[i] == '|') nextCandleIndex = i;

for (int k = 0; k < queries.size(); ++k) {

// Function to calculate the number of plates between candles

const prefixSum: number[] = new Array(n + 1).fill(0);

// Calculating the prefix sum for the plates

const nearestLeftCandle: number[] = new Array(n);

// Find the nearest left candle for each position

if (s[i] === '|') lastCandleIndex = i;

nearestLeftCandle[i] = lastCandleIndex;

if (s[i] === '|') nextCandleIndex = i;

for (let k = 0; k < queries.length; ++k) {</pre>

nearestRightCandle[i] = nextCandleIndex;

// Answer array to store the result for each query

const answer: number[] = new Array(queries.length);

// Find the nearest right candle from the start index of the query

// Find the nearest left candle from the end index of the query

const endCandleIndex: number = nearestLeftCandle[queries[k][1]];

const startCandleIndex: number = nearestRightCandle[queries[k][0]];

// If both candles are valid and the start candle is to the left of the end candle

answer[k] = prefixSum[endCandleIndex] - prefixSum[startCandleIndex + 1];

// In case there is no valid segment of plates between candles, the result is 0

// Calculate the number of plates between candles and store it in the answer

if (startCandleIndex !== -1 && endCandleIndex !== -1 && startCandleIndex < endCandleIndex) {</pre>

// Find the nearest right candle for each position

const nearestRightCandle: number[] = new Array(n);

function platesBetweenCandles(s: string, queries: number[][]): number[] {

// Create a prefix sum array to keep track of the number of plates

prefixSum[i + 1] = prefixSum[i] + (s[i] === '*' ? 1 : 0);

// Arrays to store the nearest left and right candle positions

nearestRightCandle[i] = nextCandleIndex;

// Answer vector to store the result for each query

nearestLeftCandle[i] = lastCandleIndex;

for (int i = 0, lastCandleIndex = -1; i < n; ++i) {

// Find the nearest right candle for each position

for (int i = n - 1, nextCandleIndex = -1; $i \ge 0$; --i) {

// Find the nearest right candle from the start index of the query

// If both candles are valid and the start candle is to the left of the end candle

answer[k] = prefixSum[endCandleIndex] - prefixSum[startCandleIndex + 1];

// Calculate the number of plates between candles and store it in the answer

if (startCandleIndex >= 0 && endCandleIndex >= 0 && startCandleIndex < endCandleIndex) {</pre>

// Find the nearest left candle from the end index of the query

int startCandleIndex = nearestRightCandle[queries[k][0]];

int endCandleIndex = nearestLeftCandle[queries[k][1]];

// Calculating the prefix sum of the plates

vector<int> platesBetweenCandles(string s, vector<vector<int>>& queries) {

// Create a prefix sum array to keep track of number of plates

// Arrays to store the nearest left and right candle positions

nextCandleIndex = i; // Update next seen candle

// Iterate over each query to determine the number of plates between candles

// Find the nearest right candle index from the start of the current query

// If valid candle indices are found and the start index is before the end index

// Find the nearest left candle index from the end of the current query

// Calculate the number of plates by subtracting the prefix sums

answer[k] = prefixSum[endIndex] - prefixSum[startIndex + 1];

if (startIndex >= 0 && endIndex >= 0 && startIndex < endIndex) {</pre>

Initialize the answer array, one entry for each query

Find the closest right candle for the left index

Calculate the number of plates between the candles

Return the result list containing the number of plates for each query

// Arrays to hold the index of the nearest left and right candles to each position

And the closest left candle for the right index

candle_to_the_right = closest_right_candle[left]

After processing both queries, our ans array, which holds the resulting counts, is [0, 2].

def platesBetweenCandles(self, s: str, queries: List[List[int]]) -> List[int]:

Nearest left candle from 9: left[9] = 6

 Nearest right candle from 2 (inclusive): right [2] = 2 Nearest left candle from 5 (inclusive): left[5] = 2

Since right[2] equals left[5], there are no valid boundaries within this substring, so the result for this query is 0.

○ The number of plates is presum[6] - presum[2 + 1] which equals 3 - 1 = 2. There are 2 plates between the candles at

This example completes the walk-through of the solution approach using an example string and a couple of queries. The algorithm

efficiently computes the desired count using prefix sums, and nearest left/right candle tracking to avoid repeated calculations for each query.

Python Solution

class Solution:

8

9

10

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

43

44

45

46

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

42

43

44

45

46

54

6

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

8

9

10

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

42

43

44

45

46

47

48

49

50

51

53

54

56

55 }

from typing import List

n = len(s)

Build the prefix sum array 11 prefix_sum[i + 1] = prefix_sum[i] + (char == '*') 12 13 14 # Initialize arrays to store the index of the closest candle to the left and right

Initialize a prefix sum array that will count the number of plates ('*') up to index 'i'

38 candle_to_the_left = closest_left_candle[right] 39 40 # If valid candle indexes are found and they do not overlap if candle_to_the_right >= 0 and candle_to_the_left >= 0 and candle_to_the_right < candle_to_the_left:</pre> 41 42

```
47
Java Solution
    class Solution {
         // Method to calculate the number of plates between candles based on a set of queries
  3
         public int[] platesBetweenCandles(String s, int[][] queries) {
             int length = s.length(); // Length of the string
             // Array to hold the prefix sum of plates up to each position
             int[] prefixSum = new int[length + 1];
  8
             // Calculate the prefix sum of plates
  9
             for (int i = 0; i < length; ++i) {</pre>
 10
 11
                 prefixSum[i + 1] = prefixSum[i] + (s.charAt(i) == '*' ? 1 : 0);
```

answer[index] = prefix_sum[candle_to_the_left] - prefix_sum[candle_to_the_right + 1]

49 50 51 return answer; 52 53

C++ Solution

1 #include <vector>

#include <string>

using std::vector;

using std::string;

int n = s.size();

vector<int> prefixSum(n + 1);

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

vector<int> nearestLeftCandle(n);

vector<int> nearestRightCandle(n);

vector<int> answer(queries.size());

// Process each query

const n: number = s.length;

for (let i = 0; i < n; ++i) {

let lastCandleIndex: number = -1;

let nextCandleIndex: number = -1;

// Process each query

else {

Let's break it down step by step:

for (let i = n - 1; i >= 0; ---i) {

for (let i = 0; i < n; ++i) {

class Solution {

public:

```
45
 46
 47
            return answer;
 48
 49
    };
 50
Typescript Solution
  1 // Importing necessary libraries
  2 import { Vector, String } from "prelude-ts";
```

return answer;

Time and Space Complexity

answer[k] = 0;

1. We first compute the prefix sum array presum, which takes O(n) time, where n is the length of the string s. 2. Then we create two arrays left and right that store the index of the nearest left and right candle for each position in the string. Building these arrays takes O(n) time each since we iterate over all elements from left to right and right to left, respectively.

3. Finally, we iterate over all the queries. For each query, it takes 0(1) time to compute the answer using the prefix sums and the

left and right arrays. Since there are q queries, this step takes O(q) time in total. Therefore, the overall time complexity of the code is O(n) + O(n) + O(q) = O(n + q).

- Space Complexity: 1. We use O(n) space for the prefix sum array presum of size n + 1.
- In conclusion, the time complexity of the code is 0(n + q), and the space complexity is 0(n + q), where n is the length of the string and q is the number of queries.

queries. **Time Complexity:**

The given Python code defines a method platesBetweenCandles to calculate the number of plates between candles for a series of

2. We also use O(n) space for each of the left and right arrays.

3. Finally, the space for output ans is O(q) where q is the number of queries.

Therefore, the total space used is O(n) + O(n) + O(n) + O(q) = O(n + q).