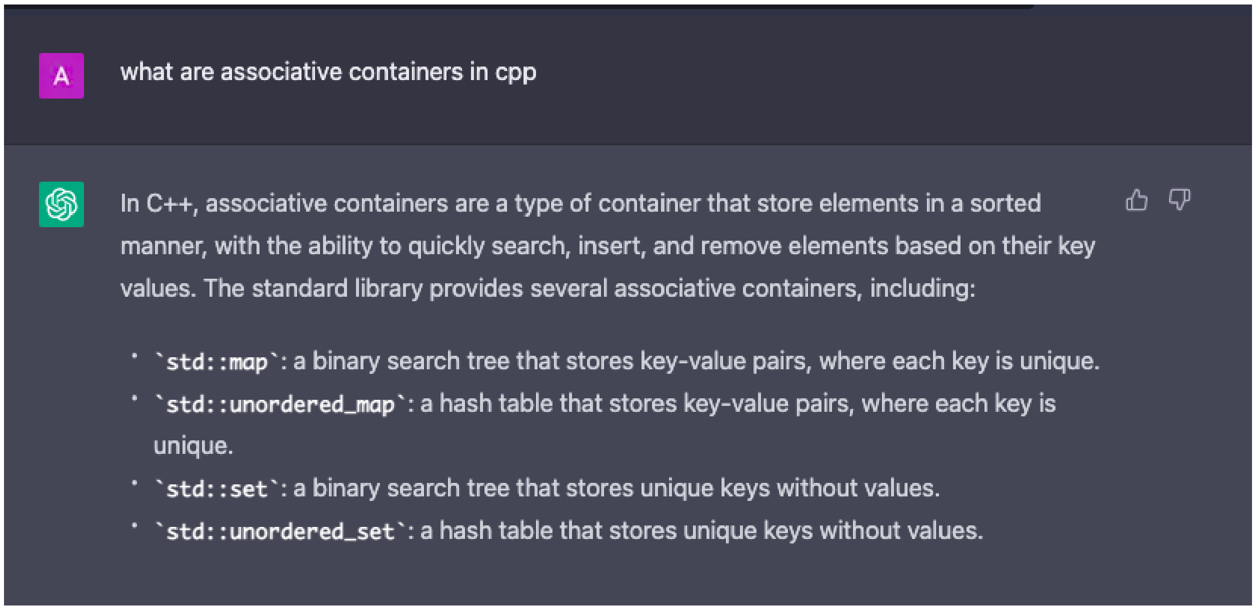
Hashing

What is a map in cpp?

Map in cpp is used to store key-value pairs where the key is unique. They are associative containers. Keys can't be modified in the map while modification can be made to value.



A what are associative containers in cpp

In C++, associative containers are a type of container that store elements in a sorted manner, with the ability to quickly search, insert, and remove elements based on their key values. The standard library provides several associative containers, including:

- `std::map`: a binary search tree that stores key-value pairs, where each key is unique.
- `std::unordered_map`: a hash table that stores key-value pairs, where each key is unique.
- `std::set`: a binary search tree that stores unique keys without values.
- `std::unordered_set`: a hash table that stores unique keys without values.

Map in STL

In unordered_map, the keys are not sorted and the time taken for insertion is O(1)
In ordered map, the keys are sorted and the time taken for insertion is O(logn)

Hashing?

Hashing technique is useful for searching purposes. Lots of space is wasted if a naive method of just storing the values at the index equal to their value is used. To improvise this, some mathematical functions are used.
In the naive method, h(x) = x function is used which is a one - one function.
If something like h(x) = x%10 is used then this is a many - one function. This will restrict the size to 10. But it will lead to a **collision**.

Methods to avoid collision

1) **Open Hashing**
    a) **Chaining -** At indexes, instead of storing the values, the starting of chains are going to be stored. Now if 3 and 13 are present in the key space. Then they will want to go to the same index 3. So the head of the linked list storing these two values will be stored in the hash table index 3.
2) **Closed Hashing**

a) **Linear Probing -** If the index at the value stored according to the mathematical function is also filled with some other value, then the next empty index will be found and the value will be stored there.
Mathematical function - h’(x) = [h(x) + f(i)] % size, f(i) = i where i = 1,2,3,4

b) **Quadratic Probing -** h’(x) = [h(x) + f(i)] % size, f(i) = i^2 where i = 1,2,3,4

Problems

1) Maximum distance between two occurrences
Intuition - We can keep storing the first occurrence of each number and checking if the distance is maximum or not.
TC - O(n)
SC - O(n)

2) First Unique Character
Code
Intuition - Maintain a variable unique index and a map. If after updating the frequency in the map. The character at unique_index is not unique now. Move until you find a unique index.
Time Complexity - O(2*n) (Reason is the nested loop can run at max n times. Unique index can at max go to s.length()).
Space Complexity - O(n)

3) Find common characters between strings
Code
Intuition - The frequency of the common character is going to be the minimum frequency of the character in a string. So calculate the minimum frequencies.
Time Complexity - O(words.length * words[i].length)
Space Complexity - O(26) => O(1)

4) Longest consecutive subsequence
Code
Intuition - Maintain two maps, one for checking while all elements are present in the array, the other for checking if it is part of a subsequence. Now loop all the elements and check for the start of a consequence. For an element to be the start of a subsequence two conditions should be matched - firstly element - 1 should not be present in the array (else that would start of the subsequence), secondly element should not be visited already(The visited array is to avoid duplicates as there can be many 1’s or any other number on the array which can be start of a subsequence)
Time complexity- O(N)

Space complexity - O(N)

5) Largest subarray with sum 0
Intuition - Maintain a map which will store {prefix_sum, first_occurence}. Now maintain a sum variable and at every index add the current variable to the sum. If the sum is already present in the map. Then the sum of values between the first occurrence and current occurrence would be zero. Hence check if i - mp[sum] is greater than local ans and update it.

6) Contiguous Array
Code
Intuition - To find the largest subarray with equal 0 and 1. We can consider 0 as -1 and then whenever sum will be 0, we will be having a subarray with equal number of 1's and -1's (0's)

7) Continuous Subarray Sum
Code
Intuition - 0 —-----> l —------> r
SumR = Sum(L-1) + Sum(l, r)
SumR%mod = Sum(L-1)%mod + Sum(l, r)%mod
Sum(l, r)%mod = 0
Hence SumR%mod = Sum(L-1)%mod

8) Brick Wall
Code
Intuition - Maintain a map and store the {distance, crack} value for each, later just check the number of cracks at every distance and update the answer with total rows - number of cracks.

9) Subarray Sum Equals K
Code
Intuition - Maintain a map to store {sum, frequency}. Maintain a sum variable and check how many times sum - k is present add that to the answer.

10) Subarray with given xor
Intuition - 0 —-------------> l —------------------>r
XORr = XORl ^ XOR(l, r)
XOR(l, r) = k
XORr = XORl ^ k => XORl = XORr ^ k
if(XORr ^ k) is present => count += mp[XORr^k]

11) Find the length of smallest subarray with sum k

Intuition - Maintain a map {sum , index}. Maintain a variable current sum, if the current sum - k is present in the map. Then from the index stored in map to the current index will have sum k. So compare the ans with i - mp[sum-k]. Update the mp[sum] in every iteration.

Follow up - Print the subarray.

Intuition for follow up - Store the range as well in a pair<int, int>.

12) Largest subarray with sum k

Intuition - There are multiple solutions to this problem(sliding window, hashing). **Hashing is used when the array contains a negative integer. Sliding window can't be used if the arrays contain negative integers.** Similar to the previous solution, the only thing different here is we don't push {sum, i} if it's only present in the map.

Note - Add sum only when it's not present, i.e., mp.count(sum)==0

13) Longest Palindrome by Contacting Two Letter Words

Code

Intuition - First add all the strings in a map. Then iterate over the map. There are mainly two type of strings ->

a) String with it[0] not equal to it[1] (example - lc)
   For this type of string we can add min(current, reverse) to the final string.
   Lc - 3, cl - 2
   We can add only two lc in left because only two cl is present for right.
   This would increase the length of the final palindromic string by min(current, reverse) * 4.
b) String with it[0] equal to it[1] (example - gg)
   For these types of string you will again have two types, length of string is even, add all elements. If string length is odd, then only for one of these odd length equal strings you can add odd length(one instance will go in the middle), rest of the strings should be added len-1 times.
   Even length || first odd length string -> len*2
   other odd length string -> (len-1)*2

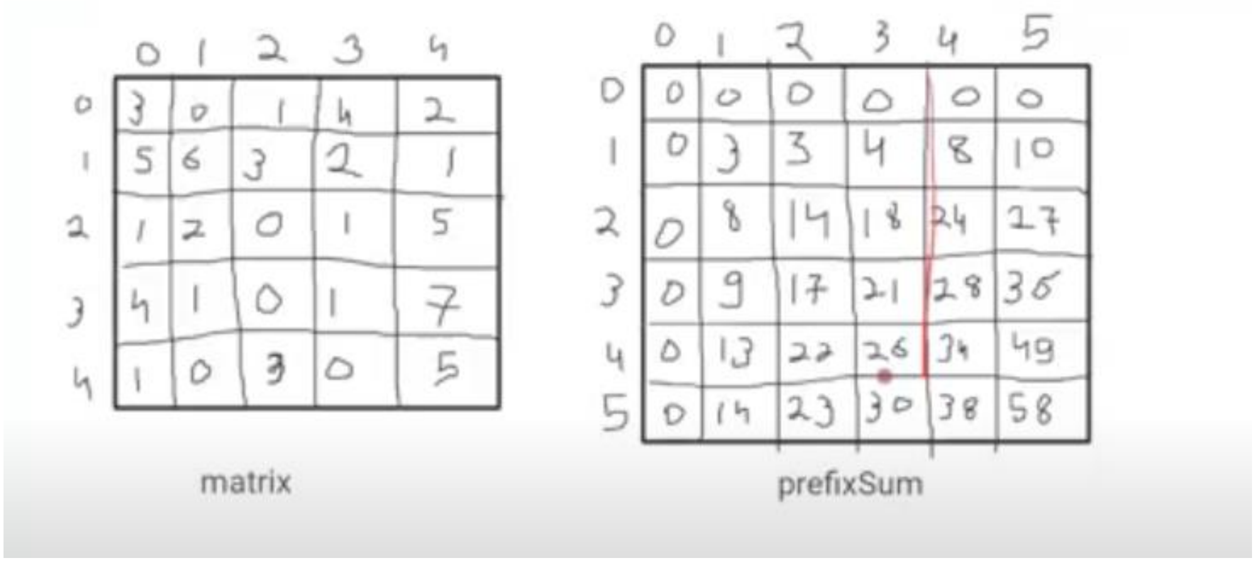
14) Concat Palindrome

Code

Intuition - Here the point to think is that we can, according to the elements of the smaller string, arrange the characters in the larger string to generate the answer. The answer will always be possible if the frequency of the characters in the smaller string is equal to the larger string. What if the number of characters are more in a larger string? In that case the number of extra characters can be arranged in the larger string to make it a palindrome only when the number of characters is even. If the number of characters is odd, we will put one element in the mid and the rest characters in a similar way as how we were doing for the even ones.

15) Range Sum Query 2D - Immutable
Code
Intuition - Create a prefix sum array of size [n+1][m+1]. The first row and first column is going to store zeros (this will help us in calculation later as we don't have check boundaries then). Now prefixsum[i][j] will represent sum of all the elements from (0, 0) to (i-1, j-1). (i-1, j-1) because we are having 1 based indexing in prefix sum but 0 based in matrix.
To calculate prefix[i][j] = prefix[i-1][j] + prefix[i][j-1] - prefix[i-1][j-1] + matrix[i-1][j-1]
prefix[i-1][j] = top
Prefix[i][j-1] = left
Prefix[i-1][j-1] = diagonal -> added twice with top and left
Matrix[i-1][j-1] -> 0 based indexing in matrix

To calculate sum from (r1, c1) to (r2, c2)
Sum = prefix[r2, c2] - prefix[r1-1][c2] - prefix[r2][c1-1] prefix[r1-1][c1-1]



	0	1	2	3	4
0	3	0	1	4	2
1	5	6	3	2	1
2	1	2	0	1	5
3	4	1	0	1	7
4	1	0	3	0	5

matrix

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	3	3	4	8	10
2	0	8	14	18	24	27
3	0	9	17	21	28	36
4	0	13	22	26	34	49
5	0	14	23	30	38	58

prefixSum

16) Max Points on a Line
Code
Intuition - Iterate through all points and maintain a map for each point, calculate slope of each point with every other point and store {slope, count}. After doing so,

the max count + 1 will be the answer. To handle the case of points lying in vertical axes, maintain a variable named vertical points separately and then the answer can be max(max_count+1, vertical_points+1).
Follow up - What if the values are not unique
Intuition - Maintain another counter for repeated values and then answer will be repeated points + max(max_count, vertical_points).

17) Number of pairs with Concatenation equal to target
Code
Intuition - Generate the required string and then increment the ans by frequency[required] and if the required string is equal to current string then ans will be incremented by frequency[required]-1.

18) Find a triplet that sums to a given value
Intuition - Iterate over all the elements and then apply two sum techniques by iteration from i+1.
Time Complexity - O(n2)

19) Check if array pairs are divisible by k
Code
Intuition - We can store the frequencies remainders inside a map {remainder, count}. As numbers can be negative, the remainder will be negative and it will cause problems later on, so instead of storing just arr[i]%k , store ((arr[i]%k)+k)%k. Then just check if for every it.first in map, there's a mp[k-it.second] is present with equal number or not. One edge case is when the remainder is 0, in that case just check if the remaining frequency is even or odd.

20) Count Distinct element in every window
Intuition - Maintain a map {element, frequency}. Now when the left element is removed from the window and its frequency is updated in the map, check if the frequency has gone down to zero because in that case the number of distinct elements will decrease by. Similarly when the right element is added into the map check if it's frequency is 1, then it is increasing the number of distinct elements.

21) Subarray sum divisible by k
Code
Intuition - Maintain a map {sum, frequency}. Add the current sum%k to the map. As numbers can be negative, sum can also be negative and we are playing with remainder, so add ((current sum%k) + k) %k.

22) Group Anagrams

Code

Intuition - Two ways to check if two strings are anagram of each other, we have two methods - 1) equate frequency map of both 2) equate sorted strings. As the string only contains lower case characters, we can first sort it using count sort (O(n)) complexity and then all the strings matching this sorted string will go to the mp[sorted_string].

23) Count numbers of substrings with exactly k distinct characters

Code

Intuition - Number of substrings with exactly k distinct characters = Number of substrings with k - Number of substrings with k-1. Use two pointers, keep adding characters using the right pointer and if unique characters exceed k, then move left and remove mp[left]. Increment counter by number of subarrays i.e, right-left+1.

24) Equal 0,1 and 2

Intuition - Maintain a map with a key as a pair of {count(0) - count(1), count(0) - count(2)}. Now if at any point (i) these values are zo and z2, then if there exists a point j such that j < i, at that point also values are z0 and z2. Then the subarray (j, i) contains an equal number of 0s, 1s and 2s as the difference is constant. **Map supports pair as key while unordered_map doesn't**.

25) Count of substrings having at least k unique characters

Intuition - Count of substrings having at least k unique characters = total substrings - count of substrings having at most k -1.

26) Largest substring with at least k unique characters

Intuition - Complete string would be the answer if there are k unique characters in it.

27) Maximum sum of distinct subarrays with length k

Code

Intuition - Maintain a window of k length and if the map size is equal to k, then update the answer.