Lesson:

Map One Shot-3

PW SKILLS

Pre-Requisites

- Maps in C++

List of Concepts Involved

- Important problems in STL

Problem 1: Can you make the strings equal?

Given an array of strings. You can move any number of characters from one string to any other string any number of times. You just have to make all of them equal.

Print "Yes" if you can make every string in the array equal by using any number of operations otherwise print "No".

Example 1:
Input: ["collegeee", "coll", "collegge"]
Output: Yes
Explanation: string at 1 index can take two 'e' from 0 index string and one 'g' from 2 index string. Thus, all strings will become equal.

Example 2:
Input: ["wall", "ah", "wallahah"]
Output: No
Explanation: Here we don't have enough number of characters to make all strings equal.

Solution Code: https://pastebin.com/aSMK2X47

Code Explanation: The trick here is that it is not necessary for us to transform the strings by performing operations on them, but we just need to check. This can be done by simply counting each character in all the strings. If each character is present in multiples of the size of the array of strings, then it is pretty clear that we can evenly divide it among all the strings thus making all the strings equal.

Time complexity: O(N), linear time complexity as we are traversing all the strings at least once, so here N is the sum of the size of all strings.

Space complexity: O(N), linear space, because we are using a map to store characters along with their count.

Code Output:

```
Enter number of strings: 3
Enter all the strings: collegeee coll collegge
Can you make the strings equal? Yes
```

```
Enter number of strings: 3
Enter all the strings: wall ah wallahah
Can you make the strings equal? No
```

Problem 2: Anagrams

Check whether two Strings are anagram of each other. Return true if they are else return false.
An anagram of a string is another string that contains the same characters, only the order of characters can be different. For example, “abcd” and “dabc” are an anagram of each other.

Input1:
triangle
integral
Output1:
true

Input2:
anagram
grams
Output2:
false

Explanation:

- We need to figure out the criteria of two strings to be anagrams.
- The first criteria is that length of 2 strings has to be the same for them to be anagrams.
- Secondly, the frequency of every character in a string has to be equal to the frequency of every character in the other string.
- No character to be extra or different in any of the strings.
- Thus, to main the count of the frequency for every character in the first string, we use a HashMap.
- Now, we can follow 2 approaches here: maintain a HashMap for both the strings and in the end just compare both the HashMaps, if they are equal we return true else false.
- This can be optimized by using only 1 HashMap. We store frequency for each character in any one of the strings. Then traverse over the other string, and for every character, first check if it is present in the HashMap or not. If it is not present, the strings cannot be anagrams anyways, so we return false. Else, we decrease its current frequency from the HashMap.
- Secondly, let’s consider if we can design a hash function which will give the same value for two strings if they are anagram of each other. In anagrams, only frequency matters, thus we can sort the strings through the hash function, i.e. hash(str) = sort(str). Now if hash(str1) = hash(str2) means str1 and str2 are anagrams.

Code: https://pastebin.com/E5MPWvYd

Output:

```
String-1 = triangle
String-2 = integral

The two strings are anagram of each other
```

```
String-1 = anagram
String-2 = gram

The two strings are NOT anagram of each other
```

Time Complexity: O(n) linear time complexity, where n represents the size of the two strings which is equal because only then we will be looping otherwise if size isn't same then we simply return false.

Space Complexity: O(n) linear space complexity, since we are creating an unordered_map here.

Problem 3: Dual mapping type questions

Check whether two Strings are isomorphic of each other. Return true if they are else return false.
Two strings are isomorphic of each other if there is a one-to-one mapping possible for every character of the first string to every character of second string and all occurrences of every character in first string maps to the same character in the second string.

Input1:
aab
xxy
Output1:
true

Input2:
abcdec
viouog
Output2:
false

Explanation:

- Two strings are considered to be isomorphic of each other, if there is a one to one mapping between all characters of the first to the second string.
- Consider three cases:
- aabb and ccdd: This is an example of isomorphic strings because both the a's in the first string are mapped to the same character, i.e. c in the second string. Also for b, both are mapped to d.
- aabb and ccde: This is not an example of isomorphic strings because both the a's in the first string are mapped to the same character, i.e. c in the second string but for b, one of them is mapped to d whereas the other is mapped to e.
- aabb and cccd: This is not an example of isomorphic strings because both the a's in the first string are mapped to the same character, i.e. c in the second string but for b, one of them is mapped to c which has already been mapped to a from the first string.
- To maintain one-to-one mapping, we will use an unordered_map here, where all characters from the first string will be the keys mapped with the corresponding character in the second string. The characters of the second string will be the values.
- If the length of both the strings is unequal, they cannot be isomorphic.
- We will traverse over the length of the strings, for every character in the first string, check if it is already present in the unordered_map or not. If already present, check if the value of this key is the same as the current character of the second string. If it is same, the match is correct and move to the next index, otherwise we have found a mismatch, and the strings are not isomorphic. Else, put the mapping to the unordered_map.

Code: https://pastebin.com/Ud6WBQYt

Output:

```
String - 1 = aab
String - 2 = xxy

The strings are Isomorphic? Yes
```

```
String - 1 = abcdec
String - 2 = viouog

The strings are Isomorphic? No
```

Time complexity: O(n) linear time complexity, where n represents the size of the two strings which is equal because only then we will be looping otherwise if size isn't same then we simply return false.

Space complexity: O(n) linear space complexity, since we are creating an unordered_map here.

Problem 4: Pair Sum

Given an array of length n and a target, return a pair whose sum is equal to the target. If there is no pair present, return -1.

Example 1:
Input:
n = 7
Elements = [1, 4, 5, 11, 13, 10, 2]
Target = 13
Output: [3, 6]
Explanation: Because a[3] + a[6] = 11 + 2 = 13

Example 2:
Input:
n = 5
Elements = [9, 10, 2, 3, 5]
Target = 15
Output: [1, 4]
Explanation: Because a[1] + a[4] = 10 + 5 = 15

Explanation:

- To find a pair and compare its sum to the target, we need to consider every element from the start and look for an element that completes its pair whose sum equals to the target.
- Instead of using nested loops, we can reduce the time complexity by using a map here.
- First store all the elements in the map, if there is any duplicate, we increment its value. So, we are storing element to its frequency as a pair.
- Traverse over the array and for every element, check if there is target-element present in the map. If yes, it means we have found a pair and return the pair.
- If after traversing over the entire array, we find no such pair, we return -1.

Code: https://pastebin.com/gTw2MVvu

Output:

```
Enter the number of elements: 7
Enter the elements of the array: 1 4 5 11 13 10 2
Enter the target: 13
The indices of 2 elements are: 3 6
```

```
Enter the number of elements: 5
Enter the elements of the array: 9 10 2 3 5
Enter the target: 15
The indices of 2 elements are: 1 4
```

Time complexity: O(N) since we are traversing all the elements at least once.
Space complexity: O(N) because we are creating a map and storing elements into it.

Problem 5: Finding Subarray questions

Given an array arr[] of length N, find the length of the longest subarray with a sum equal to 0.

Input1:
n = 8
arr[] = {15, -2, 2, -8, 1, 7, 10, 23}
Output1: 5
Explanation: The longest subarray with elements summing up-to 0 is {-2, 2, -8, 1, 7}

Input2:
n = 3
arr[] = {1, 2, 3}
Output2: 0

Explanation:

- Let's say their exist a subarray a[L...R] which has sum 0 thus

a[L] + a[L+1] + ... + a[R-1] + a[R] = 0 or (a[0] + a[1] + ... + a[R]) - (a[0] + a[1] + ... + a[L-1]) = 0
i.e prefix_sum(R) - prefix_sum(L-1) = 0 or prefix_sum(R) = prefix_sum(L-1).

- Now, if we take the prefix sum of the array then our question reduces to the maximum distance between two elements which have same value.
- Let's try to solve our problem using a map.
- The algorithm is the same, we will keep a track of the prefix sum for every index in the map. At every index, we basically need to check if there was any index prior that had the same prefix sum, and if it did we must have stored it in the map. This is where we will check the map if it contains the same prefix sum as that of the current index, if so, we have a subarray and just check if it is the longest yet by subtracting the indices. If not, we will store it in the map, to check for the further indices.
- Create a variable (sum), length (max_len), and a map (map) to store the sum-index pair as a key-value pair.
- map will store Integer, Integer pairs.
- Traverse the input array and for every index, update the value of sum = sum + array[i].
- Check every index, if the current sum is present in the map or not.
- If present, update the value of max_len to a maximum difference of two indices (current index and index in the map) and max_len.
- Else, put the value (sum) in the map, with the index as a key-value pair.
- Print the maximum length (max_len).

Code: https://pastebin.com/u7bB49de

Output:

```
Enter the size = 8
Enter the elements = 15 -2 2 -8 1 7 10 23

The length of the longest 0 sum subarray = 5
```

```
Enter the size = 3
Enter the elements = 1 2 3

The length of the longest 0 sum subarray = 0
```

Time complexity: O(N) since we are traversing all the elements at least once.
Space complexity: O(N) because we are creating an unordered_map.

Upcoming Class Teasers:

- Backtracking