02/09/25

classmate
Date 2/9/25
Page 74

DSA

Hashing Introduction

★ **Dictionary**: Stores key value pair.

★ **Set**: Stores only keys, don't store duplicates

★ **Functions**: 1> size() : no. of keys
2> add(key): adds a key to hs, if 'key' is already present, it won't do anything
3> contains(key): check if a key is present or not.
4> remove(key): If key is present, it'll remove it.

• **Notes**: - keys can't be duplicate
- order of insertion is not maintained
- TC(): add, contains, remove → O(1)
→ O(1) (String len)

Q) Given an arr[N] elements & Q queries, for each query find the freq. of each element.

	0	1	2	3	4	5	6	7	8	9
ex: arr[10]:	2	6	3	8	7	2	2	3	8	10

Queries	Freq
2	3
8	2
3	2
5	0

Idea 1: For every Query[i] Find Q[i] in arr, if found freq+1

TC: O(Q*N)

★ Idea 2: dict [key, value]
element freq.

- Steps: 1> Create a dict: if arr[i] is not present in dict → add (arr[i], 1)

2) Otherwise → update (arr[i], freq+1)

TC: O(N)

3) Search for query[i] in dict. & return the value if found else 0
TC: O(Q)

dict	
[2, 1] 2,3	[7, 1]
[6, 1]	[10, 1]
[3, 1] 2	
[8, 1] 2	

Overall TC: O(N+Q) SC: O(N)

```
def freq_query(queries, arr):
    freq_map = { }
    # Step 1: Create the dict
    for element in ~~freq_map~~ arr:
        if element in freq_map:
            freq_map[element] += 1
        else:
            freq_map[element] = 1

    # Step 2: Handle queries
    result = []
    for q in queries:
        if q in freq_map:
            result.append(freq_map[q])
        else:
            result.append(0)
    return result
```

Q) Given arr[N] elements, find the count of all distint elements

```
                0  1  2  3  4
arr[5]:         3  5  6  5  4
```

| 3 | 5 | 6 | 4 | set

size(set) = 4

ans = 4

```
                0  1  2  3  4
arr[5]:         1  1  1  2  2
```

| 1 | 2 | set

size(set) = 2

Idea: 1. Create set
2. Get length of set

```
def count_distinct(arr):
    unique = set(arr)
    return size(unique)
```

TC: O(N)
SC: O(N)

Q) Given a string s, find the length of the longest substring without repeating characters

abcabcbb — (3)
bbbb — (1)
pwwkew — (3)

☆ Brute force approach:

1) Generate all substrings : O(N²)
2) Check uniqueness : O(N)
Total : TC : O(N³)
SC : O(1)

★ Observations for optimization :

Get unique chars - set
Find substring - sliding window

```
0 1 2 3 4 5 6 7
a b c a b c b b
```

s=0, e=0, len=1	a
s=0, e=1, len=2	ab
s=0, e=2, len=(3)	abc
s=1, e=3, len=3	bca
s=2, e=4, len=3	cab
s=3, e=5, len=3	abc
s=5, e=6, len=2	cb
s=7, e=7, len=1	b

```
def longest_substring(s):
    start = 0
    max_len = 0
    char_set = set()

    for e in range(len(s)):
        while s[e] in char_set:
            char_set.remove(s[start])
            start += 1

        char_set.add(s[e])
        max_len = max(max_len,
                      e-start+1)

    return max_len
```

TC: O(N) SL: N - len of str, M - len of sub-str O(min(N, M))