Blind 75

Bhanu Prasanna Koppolu

2023-11-04

Table of contents

Notations

- The first approach is the idea popped from my mind when I looked at the problem.

Day - 9

After going through lot of things I have come across NeetCode. I have decided to solve all the 150 questions provided by NeetCode.

Contains Duplicate - 217 - LeetCode - Easy - Array & Hashing

Approach - 1

The First Approach that came to my mind was to write a Nested For Loop to check if it Contains Duplicate.

```
class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        for (auto i = nums.begin(); i != nums.end(); i++) {
            for (auto j = i + 1; j != nums.end(); j++) {
                if (*i == *j) return true;
            }
        }

        return false;
    }
};
```

Output

The First Approach has Time Complexity of O(N^2) and Space Complexity of O(1).

```
Time Limit Exceeded
70 / 75 testcases passed
```

Approach - 2

Another Approach to this problem is using Sorting then Checking if it Contains Duplicate.

```
class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        sort(nums.begin(), nums.end());

        for (int i = 0; i < nums.size() - 1; i++) {
            if (nums[i] == nums[i+1]) {
                return true;
            }
        }
        return false;
    }
};
```

Output

The Second Approach has Time Complexity of O(N log(N)) and Space Complexity of O(1).

```
Accepted
```

Approach - 3

Using unordered_set to check if it Contains Duplicate. The Time Complexity for Basic operations in unordered_set is O(1) and for set it is O(log(N)).

```
class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        unordered_set<int> us;

        for (auto i = nums.begin(); i != nums.end(); i++) {
            if (us.find(*i) != us.end()) {
                return true;
            }
            us.insert(*i);
        }

        return false;
    }
};
```

Output

The Third Approach has Time Complexity of O(N) and Space Complexity of O(N). ***The optimal solution.***

```
Accepted
```

Valid Anagram - 242 - LeetCode - Easy - Array & Hashing

Approach - 1

Sort the characters of the string of `t` and `s` then check if both are same or not.

```
class Solution {
public:
    bool isAnagram(string s, string t) {
        sort(s.begin(), s.end());
        sort(t.begin(), t.end());

        if (s == t) {
            return true;
        }
        return false;
    }
};
```

Output

The First Approach has Time Complexity of O(N log(N)) and Space Complexity of O(1).

```
Accepted
```

Approach - 2

Using unordered_map which has principles derived from Hash Map. So basic operations are O(1).

```
class Solution {
public:
    bool isAnagram(string s, string t) {

        if (s.size() != t.size()) {
            return false;
        }

        unordered_map<char, int> ums;
        unordered_map<char, int> umt;

        for (int i = 0; i < s.size(); i++) {
            ums[s[i]]++;
            umt[t[i]]++;
        }

        for (int i = 0; i < s.size(); i++) {
            if (ums[s[i]] != umt[s[i]]) {
                return false;
            }
        }

        return true;
    }
};
```

Output

The Second Approach has Time Complexity of O(S + T) and Space Complexity of O(S + T).

```
Accepted
```

Two Sum - 1 - LeetCode - Easy - Array & Hashing

Approach - 1

The general approach to this problem is Brute Force Approach.

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        int l = nums.size();

        for (int i = 0; i < l; i++) {
            for (int j = 0; j < l && i != j; j ++) {
                if (nums[i] + nums[j] == target) {
                    return {i, j};
                }
            }
        }

        return {};
    }
};
```

Output

The First Approach has Time Complexity of O(N^2) and Space Complexity of O(1).

```
Accepted
```

Approach - 2

Using unordered_map which has principles derived from Hash Map. So basic operations are O(1).

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map<int, int> um;

        for (int i = 0; i < nums.size(); i++) {
            if (auto search = um.find(target - nums[i]); search != um.end()) {
                return {search->second, i};
            }

            um[nums[i]] = i;
        }

        return {};
    }
};
```

Output

The Second Approach has Time Complexity of O(N) and Space Complexity of O(N).

```
Accepted
```

Day - 10

Group Anagrams - 49 - LeetCode - Medium - Array & Hashing

Approach - 1

Sorting and Matching Strings.

```
class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        vector<string> st = strs;
        vector<vector<string>> fs;
        unordered_set<int> us;

        for (int i = 0; i < st.size(); i++) {
            sort(st[i].begin(), st[i].end());
        }

        for (int i = 0; i < st.size(); i++) {
            vector<string> vs;

            for (int j = i; j < st.size(); j++) {
                if (us.find(j) == us.end()) {
                    if (st[i] == st[j]) {
                        vs.push_back(strs[j]);
                        us.insert(j);
                    }
                }
            }
            if (vs.size() != 0) {
                fs.push_back(vs);
            }
        }
```

```
        return fs;
    }
};
```

Output

The First Approach has Time Complexity of O(mn log(n)) and Space Complexity of O(n).

```
Accepted
```

Approach - 2

Used Hash Map to reduce the Time Complexity.

```
class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        unordered_map<string, vector<string>> m;

        for (int i = 0; i < strs.size(); i++) {
            string key = getkey(strs[i]);
            m[key].push_back(strs[i]);
        }

        vector<vector<string>> fs;

        for (auto i = m.begin(); i != m.end(); i++) {
            fs.push_back(i->second);
        }

        return fs;
    }
private:
    string getkey (string val) {
        string s = "";
        vector<int> c(26);

        for (int i = 0; i < val.size(); i++) {
            c[val[i] - 'a'] += 1;
```

```
        }

        for (int i = 0; i < 26; i++) {
            s = s + to_string(c[i]) + '#';
        }

        return s;
    }
};
```

Output

The Second Approach has Time Complexity of O(mn) and Space Complexity of O(mn).

```
Accepted
```