Blind 75

Bhanu Prasanna Koppolu

2024-02-09

Table of contents

Blind 75

I want to improve my problem solving skills. I am starting with Blind 75.

Language Used - C++

1 - Two Sum - Easy

Notes

The optimal solution uses a Unordered Map which is internally built using Hash Table concepts.

Usage of Hash Table enables the least cost of operations like search, delete, and insert.

Program

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

        for (int i = 0; i < nums.size(); i++) {
            int com = target - nums[i];
            if (s.find(com) != s.end()) {
                return {s[com], i};
            }
            s.insert({nums[i], i});
        }
        return {};
    }
};
```

2 - Contains Duplicate - Easy

Notes

The optimal solution uses a Unordered Set which is internally built using Hash Table concepts.

Usage of Hash Table enables the least cost of operations like search, delete, and insert.

Program

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

        for (int i : nums) {
            if (us.find(i) != us.end()) {
                return true;
            }
            us.insert(i);
        }

        return false;
    }
};
```

3 - Valid Anagram - Easy

Notes

The optimal solution is to count the number of characters in each string for the 26 alphabets.

Program

```
class Solution {
public:
    bool isAnagram(string s, string t) {
        if (s.size() != t.size()) {
            return false;
        }

        int c1[26] = {0};
        int c2[26] = {0};

        for (int i = 0; i < s.size(); i++) {
            c1[(s[i] - 97)]++;
            c2[(t[i] - 97)]++;
        }

        for (int i = 0; i < 26; i++) {
            if (c1[i] != c2[i]) {
                return false;
            }
        }

        return true;
    }
};
```

4 - Group Anagrams - Medium

Notes

The optimal solution uses the same 26 alphabet count method but additionally uses a map to store the same type anagrams.

Program

```
class Solution {
private:
    string getKey (string str) {
        vector<int> v(26,0);

        for (int i = 0; i < str.size(); i++) {
            v[str[i] - 'a']++;
        }

        string s = "";

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

        return s;
    }
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        unordered_map<string, vector<string>> um;

        for (int i = 0; i < strs.size(); i++) {
            string k = getKey(strs[i]);
            um[k].push_back(strs[i]);
        }

        vector<vector<string>> v;

        for (auto i = um.begin(); i != um.end(); i++) {
            v.push_back(i->second);
        }
        return v;
    }
};
```

5 - Top K Frequent Elements - Medium

Notes

Usage of modified Bucket Sort approach which provides this optimal solution. This solution can also be done using Max Heap.

Program

```
class Solution {
public:
    vector<int> topKFrequent(vector<int>& nums, int k) {
        unordered_map<int, int> um;
        for (int i = 0; i < nums.size(); i++) {
            um[nums[i]]++;
        }

        vector<vector<int>> v(nums.size() + 1);
        for (auto it = um.begin(); it != um.end(); it++) {
            v[it->second].push_back(it->first);
        }

        vector<int> result;

        for (int i = nums.size(); i >= 0; i--) {
            if (result.size() >= k) {
                return result;
            }
            if (v[i].size() != 0) {
                result.insert(result.end(), v[i].begin(), v[i].end());
            }
        }

        return {};
    }
};
```

6 - Product of Array Except Itself - Medium

Notes

The optimal solution is using a prefix and a postfix to parse through the array from both sides and multiplying.

Program

```cpp
class Solution {
public:
    vector<int> productExceptSelf(vector<int>& nums) {
        vector<int> result(nums.size(), 1);
        int pre = 1;
        int post = 1;

        for (int i = 0; i < nums.size(); i++) {
            result[i] = result[i] * pre;
            pre = pre * nums[i];
        }

        for (int i = nums.size() - 1; i >= 0; i--) {
            result[i] = result[i] * post;
            post = post * nums[i];
        }

        return result;
    }
};
```