Shri Vile Parle Kelavani Mandal's

# INSTITUTE OF TECHNOLOGY

**DHULE (M.S.)**

**DEPARMENT OF COMPUTER ENGINEERING**

**Subject :** Competitive Programming Lab **Subject Code :** BTCOL606

**Name of student: Pagariya Sakshi Manojkumar** **Roll No : 28**

**Class:** T. Y. Comp. Engg. **Batch : T2**

**Semester :** VI **Academic Year :** 2022-2023

**Assignment No :**

**Date of Performance**: / / 2023 **Date of Submission**: / / 2023

| **Marks Split upto** | **Maximum Marks** | **Marks Obtained** |
|---|---|---|
| **Performance Conduction** | **3** | |
| **Report Writing** | **3** | |
| **Attendance** | **2** | |
| **Viva/Oral** | **2** | |
| **Total Marks** | **10** | |
| **Signature of Subject Teacher** | | |

**Title :** *Sort Anagrams*

| | |
|---|---|
| **Problem Staement :** | Given an array of strings strs, group the anagrams together. You can return the answer in any order. An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once. |
| **Platform Used:** | Leetcode |
| **Theory :** | The code aims to group anagrams together from a given list of strings. Anagrams are words or phrases that can be formed by rearranging the letters of another word or phrase. To solve this problem, the code utilizes the concept of anagrams being strings with the same set of characters, but possibly in different orders. By sorting the characters of each string, we can obtain a unique representation for each set of anagrams. The algorithm starts by initializing an empty dictionary, `strs_table`, which will serve as a data structure to store the groups of anagrams. The keys of this dictionary are the sorted representations of the strings, and the values are lists of the original strings. Next, the algorithm iterates over each string in the input list. For each string, it creates a sorted representation by sorting the characters and joining them back together. This |

| | |
|---|---|
| | sorted string serves as the key in `strs_table`<br><br>If the sorted string is not already present as a key in `strs_table`, the algorithm creates a new entry with the sorted string as the key and an empty list as the value. This ensures that each unique set of anagrams has its own entry in the dictionary.<br><br>Finally, the original string is appended to the list associated with the sorted string key in `strs_table`. This way, all the anagrams of a particular set are grouped together under the same key.<br><br>In the end, the algorithm returns the values of `strs_table` as a list. Each value in the list represents a group of anagrams.The underlying theory of this code revolves around recognizing the common property of anagrams and leveraging the sorted representation to efficiently group them together. By utilizing a dictionary as a data structure, the algorithm achieves an efficient grouping of anagrams in linear time complexity. |
| **Algorithm:** | 1. Initialize an empty dictionary, `strs_table`.<br>2. Iterate over each string `s` in the input list `strs`.<br>   - Create a sorted representation of `s` by sorting its characters and joining them back together.<br>   - Check if the sorted string is already a key in `strs_table`.<br>   - If it is not, create a new entry in `strs_table` with the sorted string as the key and an empty list as the value.<br>   - If it is already a key, proceed to the next step.<br>   - Append `s` to the list associated with the sorted string key in `strs_table`.<br>3. Convert the values of `strs_table` to a list and return it. Each value represents a group of anagrams. |
| **Programming Language:** | Python |
| **Code:** | `class Solution(object):`<br>`    def groupAnagrams(self, strs):`<br>`        """`<br>`        :type strs: List[str]`<br>`        :rtype: List[List[str]]`<br>`        """`<br>`        strs_table = {}`<br><br>`        for s in strs:`<br>`            sorted_string = ''.join(sorted(s))`<br><br>`            if sorted_string not in strs_table:`<br>`                strs_table[sorted_string] = []`<br><br>`            strs_table[sorted_string].append(s)`<br><br>`        return list(strs_table.values())` |

| | |
|---|---|
| **Output:** | <pre>["eat","tea","tan","ate","nat","bat"]<br>[["tan","nat"],["bat"],["eat","tea","ate"]]<br><br>[""]<br>[[""]]</pre> |