Course Name:	Competitive Programming Laboratory (216U01L401)	Semester:	IV
Date of Performance:	_11__ / 04___ / ___25___	DIV/ Batch No:	E-2
Student Name:	Shreyans Tatiya	Roll No:	16010123325

Experiment No: 6

Title: To use string algorithms to solve competitive programming problems.

Aim and Objective of the Experiment:

1. **Understand** the concepts of string algorithm.
2. **Apply** the concepts to solve the problem.
3. **Implement** the solution to given problem statement.
4. **Create** test cases for testing the solution.
5. **Analyze** the result for efficiency of the solution.

COs to be achieved:

CO4: Apply geometry, game theory, and string algorithms in competitive programming.

Books/ Journals/ Websites referred:

1. https://leetcode.com/problems/group-anagrams/description/
2. Competitive Programmer's Handbook (CPH) - Antti Laaksonen

Theory:

To group anagrams from a list of strings, we rely on the observation that anagrams contain the exact same characters with the same frequencies, just arranged differently. By sorting the characters of each string, we obtain a common key for all its anagram variants. This sorted form is used as a key in a hash map , where each key maps to a list of words that are anagrams of each other. A **hash map is a data structure** that stores key-value pairs and allows for **fast lookups, insertions, and deletions** on average in constant time, O(1). Internally, it uses a hash function to convert the key into an index in an underlying array. This makes it perfect for grouping problems where we want to categorize elements efficiently. By using the sorted string as a hash key, we classify words based on their character composition without repeated comparisons. This approach reduces the problem from a potentially quadratic time complexity to a more optimal one, typically O(n * k log k), where n is the number of strings and k is the maximum length of a string. This solution elegantly combines string manipulation, sorting, and hashing demonstrating the power of fundamental data structures and algorithmic patterns.

Problem statement

Given an array of strings strs, group the anagrams together. You can return the answer in any order.

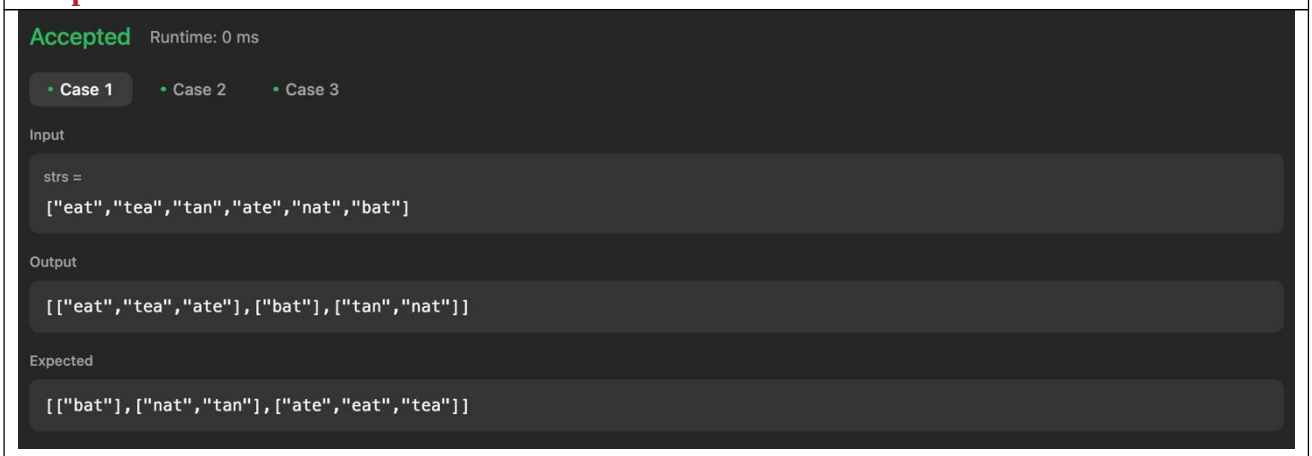
Example 1:
Input: strs = ["eat","tea","tan","ate","nat","bat"]
Output: [["bat"],["nat","tan"],["ate","eat","tea"]]

Explanation:
There is no string in strs that can be rearranged to form "bat".
The strings "nat" and "tan" are anagrams as they can be rearranged to form each other.
The strings "ate", "eat", and "tea" are anagrams as they can be rearranged to form each other.

Code :

```
class Solution {
    public:
        vector<vector<string>> groupAnagrams(vector<string>& strs) {
            unordered_map<string, vector<string>> mp;
            for (auto x: strs) {
                string word = x;
                sort(word.begin(), word.end());
                mp[word].push_back(x);
            }
            vector<vector<string>> ans;
            for (auto x: mp) {
                ans.push_back(x.second);
            }
            return ans;
        }
    };
```

Output:



Post Lab Subjective/Objective type Questions:

1. Explain KMP Algorithm?

The **KMP (Knuth-Morris-Pratt) algorithm** is an efficient string matching technique used to find occurrences of a pattern within a given text. Unlike the naive approach, which can re-check characters unnecessarily, KMP improves the process by preprocessing the pattern to build an auxiliary array called the **LPS (Longest Prefix Suffix)** array. This array stores the length of the longest prefix that is also a suffix for every position in the pattern. Using this LPS array, the algorithm can skip unnecessary comparisons when a mismatch occurs, thus avoiding backtracking in the text. The overall time complexity of the KMP algorithm is **O(n + m)**, where `n` is the length of the text and `m` is the length of the pattern. This makes it particularly efficient for searching large texts or when repeated searches are required with the same pattern.

Conclusion:
The experiment successfully demonstrates how anagrams can be efficiently grouped using hashing and string sorting. By leveraging a hash map with sorted strings as keys, we achieve optimized performance and avoid unnecessary comparisons. This highlights the practical use of data structures like hash maps for solving real-world string classification problems.