<https://leetcode.com/problems/group-anagrams/>

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.

**Example 1:**

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

Output: [["bat"],["nat","tan"],["ate","eat","tea"]]

**Example 2:**

Input: strs = [""]

Output: [[""]]

**Example 3:**

Input: strs = ["a"]

Output: [["a"]]

**Constraints:**

* 1 <= strs.length <= 104
* 0 <= strs[i].length <= 100
* strs[i] consists of lowercase English letters.

**Attempt 1: 2023-02-01**

**Solution 1:  Hash Table + Instead of sorting we build the key string in this way (30 min)**

class Solution {

public List<List<String>> groupAnagrams(String[] strs) {

Map<String, List<String>> map = new HashMap<String, List<String>>();

for(String str : strs) {

int[] encrypted = new int[26];

for(char c : str.toCharArray()) {

encrypted[c - 'a']++;

}

String key = Arrays.toString(encrypted);

map.putIfAbsent(key, new ArrayList<String>());

map.get(key).add(str);

}

return new ArrayList<>(map.values());

}

}

Time Complexity: O(NK), where N is the length of strs, and K is the maximum length of a string in strs. Counting each string is linear in the size of the string, and we count every string.

Space Complexity: O(NK), the total information content stored in ans.

**Refer to**

<https://leetcode.com/problems/group-anagrams/solutions/566674/java-hashmap-solution-and-no-sorting-required/>

class Solution {

public List<List<String>> groupAnagrams(String[] strs) {

Map<String, List<String>> map = new HashMap();

for (String s : strs) {

int[] m = new int[26];

for (char c : s.toCharArray()) m[c-'a']++;

String key = Arrays.toString(m);

map.putIfAbsent(key, new ArrayList());

map.get(key).add(s);

}

return new ArrayList(map.values());

}

}

<https://leetcode.com/problems/group-anagrams/solutions/127405/group-anagrams/>

#### **Approach 2: Categorize by Count**

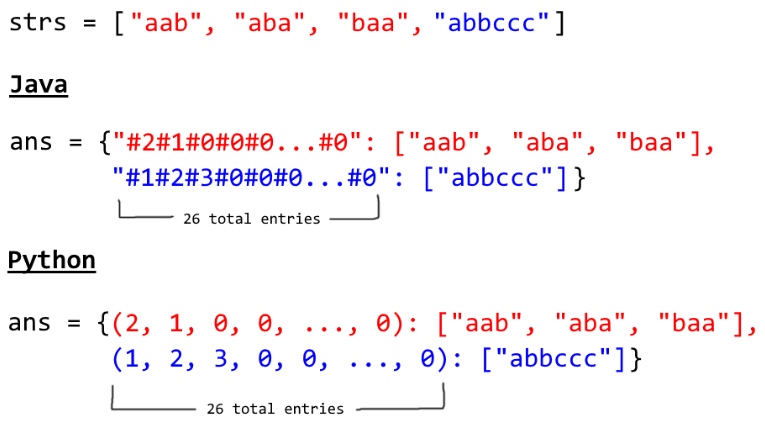
**Intuition**

Two strings are anagrams if and only if their character counts (respective number of occurrences of each character) are the same.

**Algorithm**

We can transform each string s into a character count consisting of 26 non-negative integers representing the number of a's, b's, c's, etc. We use these counts as the basis for our hash map.

In Java, the hashtable representation of our count will be a string delimited with '**#**' characters. For example, abbccc will be #1#2#3#0#0#0...#0where there are 26 entries total. In python, the representation will be a tuple of the counts. For example, abbccc will be (1, 2, 3, 0, 0, ..., 0), where again there are 26 entries total.



class Solution {

public List<List<String>> groupAnagrams(String[] strs) {

if (strs.length == 0) return new ArrayList();

Map<String, List> ans = new HashMap<String, List>();

int[] count = new int[26];

for (String s : strs) {

Arrays.fill(count, 0);

for (char c : s.toCharArray()) count[c - 'a']++;

StringBuilder sb = new StringBuilder("");

for (int i = 0; i < 26; i++) {

sb.append('#');

sb.append(count[i]);

}

String key = sb.toString();

if (!ans.containsKey(key)) ans.put(key, new ArrayList());

ans.get(key).add(s);

}

return new ArrayList(ans.values());

}

}

**Complexity Analysis**

* Time Complexity: O(NK), where N is the length of strs, and K is the maximum length of a string in strs. Counting each string is linear in the size of the string, and we count every string.
* Space Complexity: O(NK), the total information content stored in ans.

**Solution 2:  Hash Table + Sorting (10 min)**

class Solution {

public List<List<String>> groupAnagrams(String[] strs) {

Map<String, List<String>> map = new HashMap<String, List<String>>();

for(String str : strs) {

char[] chars = str.toCharArray();

Arrays.sort(chars);

String key = String.valueOf(chars);

map.putIfAbsent(key, new ArrayList<String>());

map.get(key).add(str);

}

return new ArrayList<>(map.values());

}

}

Time Complexity: O(NKlog⁡K), where N is the length of strs, and K is the maximum length of a string in strs. The outer loop has complexity O(N) as we iterate through each string. Then, we sort each string in O(Klog⁡K) time.

Space Complexity: O(NK), the total information content stored in ans.

**Refer to**

<https://leetcode.com/problems/group-anagrams/solutions/127405/group-anagrams/>

#### **Approach 1: Categorize by Sorted String**

**Intuition**

Two strings are anagrams if and only if their sorted strings are equal.

**Algorithm**

Maintain a map ans : {String -> List}where each key K\text{K}Kis a sorted string, and each value is the list of strings from the initial input that when sorted, are equal to K\text{K}K.

In Java, we will store the key as a string, eg. code. In Python, we will store the key as a hashable tuple, eg. ('c', 'o', 'd', 'e').



class Solution {

public List<List<String>> groupAnagrams(String[] strs) {

if (strs.length == 0) return new ArrayList();

Map<String, List> ans = new HashMap<String, List>();

for (String s : strs) {

char[] ca = s.toCharArray();

Arrays.sort(ca);

String key = String.valueOf(ca);

if (!ans.containsKey(key)) ans.put(key, new ArrayList());

ans.get(key).add(s);

}

return new ArrayList(ans.values());

}

}

**Complexity Analysis**

* Time Complexity: O(NKlog⁡K), where N is the length of strs, and K is the maximum length of a string in strs. The outer loop has complexity O(N) as we iterate through each string. Then, we sort each string in O(Klog⁡K) time.
* Space Complexity: O(NK), the total information content stored in ans