Visual Explanation of the 'Group Anagrams' Solution

Problem Statement

Given a list of strings, group anagrams together. Two strings are anagrams if they have the same characters with the same frequencies, regardless of order.

Key Code Used:

```
from collections import defaultdict

def group_anagrams(strs):
    anagram_map = defaultdict(list)

    for word in strs:
        char_count = [0] * 26
        for char in word:
            char_count[ord(char) - ord('a')] += 1

        anagram_map[tuple(char_count)].append(word)

    return list(anagram_map.values())
```

Step-by-Step Breakdown

1. Input:

Example Input:

["act", "pots", "tops", "cat", "stop", "hat"]

2. Character Counting:

For each word in the input list, count the occurrences of each character and represent the count as a list of 26 integers (corresponding to each letter of the alphabet).

Word	Character Count Array
act	[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
pots	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0]
tops	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0]
cat	[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
stop	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0]

hat	[1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

3. Using the Character Count Array as a Key:

Convert the list of character counts into an immutable tuple so it can be used as a dictionary key. Then group all words with the same key together in the dictionary.

Key (Character Count Tuple)	Words
(1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0)	["act", "cat"]
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0)	["pots", "tops", "stop"]
(1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0)	["hat"]

4. Extracting Values:

Finally, take all the grouped anagrams (values in the dictionary) and return them as a list of lists.

Final Output:

[["act", "cat"], ["pots", "tops", "stop"], ["hat"]]

Why This Solution Works:

1. Efficiency:

- Counting characters: O(N * M), where N = number of words, M = average word length.

- Grouping words using a dictionary: O(1) insertion for each word.

- Overall: O(N * M).

2. Simplicity:

- Using `defaultdict` makes appending to groups straightforward.

3. Scalability:

- Handles a large number of words and diverse inputs effectively.