49. Group Anagrams

Group Anagrams

Medium

Given an array of strings `strs`, group all *anagrams* together into sublists. You may return the output in any order.

An anagram is a string that contains the exact same characters as another string, but the order of the characters can be different.

Example 1:

```
Input: strs = ["act","pots","tops","cat","stop","hat"]

Output: [["hat"],["act", "cat"],["stop", "pots", "tops"]]
```

Copy

Example 2:

```
Input: strs = ["x"]

Output: [["x"]]
```

Copy

Example 3:

```
Input: strs = [""]

Output: [[""]]
```

Copy

Constraints:

`1 <= strs.length <= 1000`. `0 <= strs[i].length <= 100strs[i]` is made up of lowercase English letters.

- `1 <= strs.length <= 1000`.
- `0 <= strs[i].length <= 100`
- `strs[i]` is made up of lowercase English letters.

My Initial Thought :

Anagrams can be detected using the frequency of each word and the length of the word, we can define different anagrams by matching similar character frequencies. To group sublists of anagrams we can use the maximum length and minimum length of existing words, use a hash map to sort commonly frequently words from most to last, we can narrow down the lists iteratively as we go, so using monotonic properties of the strings.

So as we go down the list of most to least commonly frequently words, we can narrow down groups of anagrams,

Revising this logic, it is best if we can instead sort the individuals words instead to be arranged alphabetically, this way we can map character frequency

We first initialise a dictionary to store the grouped anagrams

- Count character frequency using a dictionary
- Convert the frequency dictionary to a sorted tuple of (char, count)

```
key = tuple(sorted(char_frequency.items()))  # Sort by charact
```

.items() is used to convert char_frequency into a dictionary

which is then sorted by character and is converted into a tuple to be used as a dictionary key

(remember that a tuple is hashable)

```
  groups[key].append(word)
```

Then we group words with the same frequency key, adds current word to the list corresponding to its key

```
  return list(groups.values())
```

```
from collections import defaultdict

class Solution:
    def groupAnagrams(self, strs: List[str]) -> List[List[str]]
        groups = defaultdict(list)  # To store grouped anagrams

        for word in strs:

         char_frequency = {}
         for char in word:
            char_frequency[char] = char_frequency.get(char, 0) +

        # Convert the frequency dictionary to a sorted tuple of
         key = tuple(sorted(char_frequency.items()))  # Sort by

        # Group words with the same frequency key
         groups[key].append(word)
```

```
        return list(groups.values())
```

Why this works

- We sort the given characters by its character frequency and alphabetical order, you'll find that words will belong in the same group if they are a matching anagram. This is their unique identifier
- Now we need to create a key (which is done by converting the dictionary of character frequency into a tuple)
- After converting the key, we can group words with the same frequency