49. Group Anagrams

ex

i/p → strs = ["eat", "tea", "tan", "ate", "nat", "bat"]

o/p → [["bat"], ["nat", "tan"], ["ate", "eat", "tea"]];

Approch-1

Rearrange the letters of the string such that if two strings are same the we can check easily

↳ The most obvious rearrangement to make all characters string same is

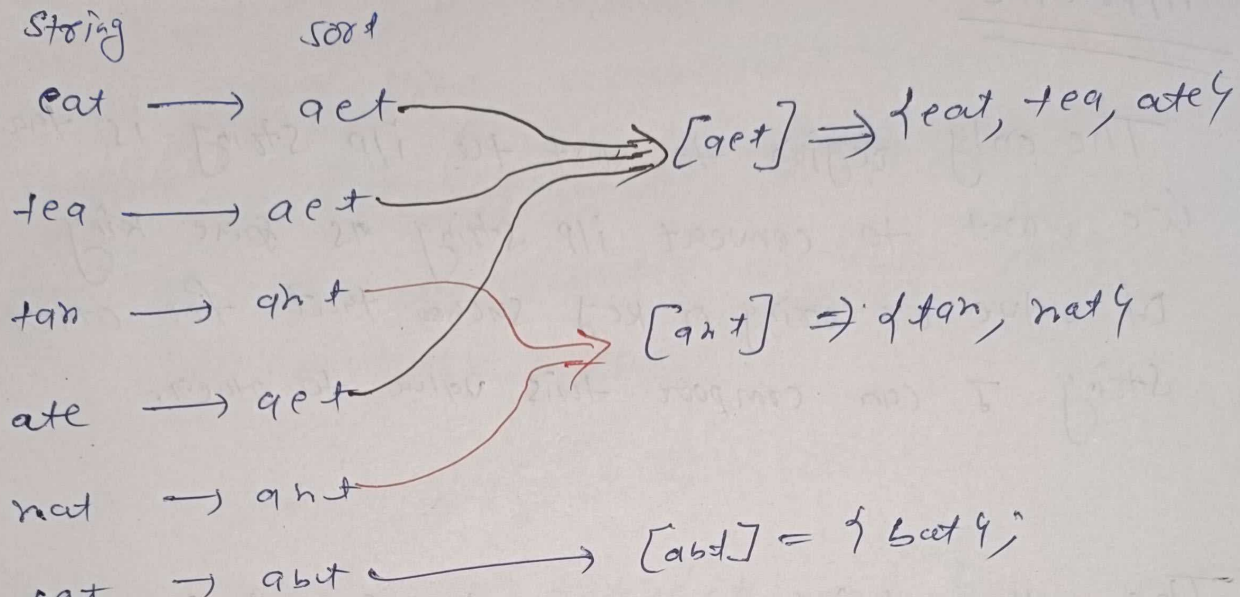
↳ sort the string → characters will be in sorted order.

ate → sort → aet

tea → sort → aet

att → sort → att

These two are anagram



Code

```
vector<vector<string>> groupAnagrams(vector<string>& strs)
{
    unordered_map<string, vector<string>> mp;

    for(auto s: strs)
    {
        string original_str = s;
        sort(s.begin(), s.end());
        mp[s].push_back(original_str);
    }

    vector<vector<string>> anagrams;

    for(auto it : mp)
    {
        anagrams.push_back(it.second);
    }

    return anagrams;
}
```

loop sorting

T.C. = O(n klog(k))

S.C. = O(nk)

Approch-2

The only region to sort the i/p string is that we want to convert i/p string as some king of value or string or key such that for other string I can compare this value to other.

The other thing we can do is we can get the freqencies of characters from i/p string & store it in

vector< int > freq;

< 1, 0, 0, 0, 1 --- >

& for other string if two string have same frequency of vector then they have to be anagram.

Note: If we want to make a vector<int> as our key then we have to use: a map → T.C. = O(log n) ← for insert & update.

or if we use an unordered-map then we have to explicitly write the hash fn for the vector<int>;

Imp?:-

Rather using vector<int> frequency we convert the frequency vector to a string so that we can directly use this string as key in unordered_map

<1, 0, 0, 1, ---->

"1#0#0#0#1#-----"

↳ why we are using #

This # is act as delimeter bcz in the original string if frequency of character like freq[a] = 12 then we can

12#0#1#

easily separate two characters's frequency.

T.C. = O(n * k)

S.C. = O(n * k)