49. Group Anagrams

```
from collections import defaultdict
class Solution:
    def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
        mem = defaultdict(list)
        for s in strs:
            mem[''.join(sorted(s))].append(s)
        return mem.values()
```

76. Minimum Window Substring

We will use slide window and hashmap. TC is O(n), SC is O(n).

```
from collections import defaultdict
class Solution:
    def minWindow(self, s: str, t: str) -> str:
        mem = defaultdict(int)
        counter = 0
        result = ''
        min_length = float('inf')
        l, r = 0, 0
        for c in t:
            if mem[c] == 0:
                counter += 1
            mem[c] += 1

        while r < len(s):
            if s[r] in mem:
                mem[s[r]] -= 1
                if mem[s[r]] == 0:
                    counter -= 1
            r += 1
            while counter == 0:
                if s[l] in mem:
                    if r - l < min_length:
                        min_length = r - l
                        result = s[l:r]
                    if mem[s[l]] == 0:
                        counter += 1
                    mem[s[l]] += 1
                l += 1
        return result
```

340. Longest Substring with At Most K Distinct Characters

```
from collections import defaultdict
class Solution:
    def lengthOfLongestSubstringKDistinct(self, s: str, k: int) -> int:
        mem = defaultdict(int)
        counter = 0
        max_length = 0
        l, r = 0, 0
        if k == 0:
            return 0
        while r < len(s):
            if mem[s[r]] == 0:
                k -= 1
            mem[s[r]] += 1
            r += 1
            while k == -1 and l < r:
                if r - l - 1 > max_length:
                    max_length = r - l - 1
                mem[s[l]] -= 1
                if mem[s[l]] == 0:
                    k += 1
                l += 1
        if r - l > max_length:
            max_length = r - l

        return len(s) if max_length == 0 else max_length
```

Simple version:

```
from collections import defaultdict
class Solution:
    def lengthOfLongestSubstringKDistinct(self, s: str, k: int) -> int:
        mem = defaultdict(int)
        counter = 0
        max_length = 0
        l, r = 0, 0
        while r < len(s):
            if mem[s[r]] == 0:
                k -= 1
            mem[s[r]] += 1
            r += 1
            while k < 0:
                mem[s[l]] -= 1
                if mem[s[l]] == 0:
```

```
                k += 1
            l += 1
        if r - l > max_length:
            max_length = r - l



    return max_length
```

395. Longest Substring with At Least K Repeating Characters

```
class Solution:
    def longestSubstring(self, s: str, k: int) -> int:
        if len(s) < k:
            return 0
        c = min(set(s), key=s.count)
        if s.count(c) >= k:
            return len(s)
        return max([self.longestSubstring(splited_s, k) for splited_s in s.split(c)])
```

3. Longest Substring Without Repeating Characters

Use slice window. TC is O(n), SC is O(1)

```
from collections import defaultdict
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        mem = defaultdict(int)
        counter = 0
        max_length = 0
        l, r = 0, 0
        while r < len(s):
            mem[s[r]] += 1
            r += 1
            while mem[s[r - 1]] > 1:
                mem[s[l]] -= 1
                l += 1
            if r - l > max_length:
                max_length = r - l
        return max_length
```