409. Longest Palindrome

We will count each character's number and get each character's even number. At the same time, if there is an odd number. We will set single = 1. In the end, we will return even * 2 + single. TC is O(n)

```
from collections import defaultdict
class Solution:
    def longestPalindrome(self, s: str) -> int:
        memo = defaultdict(int)
        count = 0
        single = 0
        for c in s:
          memo[c] += 1
        for k, v in memo.items():
          if v % 2 == 1:
            single = 1
          count += v // 2
        return count * 2 + single
```

5. Longest Palindromic Substring

We will iterate through string and start from current character until longest length. We will compare each time and record the longest one. O(n**2)

```
class Solution:
    def longestPalindrome(self, s: str) -> str:
        max_length = 0
        result = ""
        for i in range(len(s)):
            longest_palindrome = self.helper(s, i, i)
            if len(longest_palindrome) > max_length:
                result = longest_palindrome
                max_length = len(longest_palindrome)
            longest_palindrome = self.helper(s, i, i + 1)
            if len(longest_palindrome) > max_length:
                result = longest_palindrome
                max_length = len(longest_palindrome)
        return result

    def helper(self, s, start, end):
        while start >= 0 and end < len(s) and s[start] == s[end]:
            start -= 1
            end += 1
        return s[start + 1:end]
```

127. Word Ladder

We will use dfs to dive in to next layer if there is word existing in the wordlist. If we add transformed word to our next_ite. We will keep looking up until we find endword. Then we will return current layer, which represents looking time.

```
from collections import defaultdict
class Solution:
    def ladderLength(self, beginWord: str, endWord: str, wordList: List[str]) -> int:
        visited = set()
        letters = 'abcdefghijklmnopqrstuvwxyz'
        words = set(wordList)

        def dfs(cur_words, visited, time):
            next_ite = []
            for w in cur_words:
                for i in range(len(w)):
                    for l in letters:
                        new_word = w[:i] + l + w[i + 1:]
                        if new_word in words and new_word not in visited:
                            if new_word == endWord:
                                return time + 1
                            next_ite.append(new_word)
                            visited.add(new_word)
            if next_ite:
                return dfs(next_ite, visited, time + 1)
            else:
                return 0

        return dfs([beginWord], visited, 1)
```

64. Minimum Path Sum

We will use dp. We will add the previous one when it's first row or first column. Otherwise, we will add min(left, right). Then we will return the last cell's number. TC is O(mn)

```
class Solution:
    def minPathSum(self, grid: List[List[int]]) -> int:
        m = len(grid)
        n = len(grid[0])
        for i in range(m):
            for j in range(n):
                if i == 0:
                    if j == 0:
                        continue
                    else:
                        grid[i][j] += grid[i][j - 1]
                else:
```

```
            if j == 0:
                grid[i][j] += grid[i - 1][j]
            else:
                grid[i][j] += min(grid[i][j - 1], grid[i - 1][j])
        return grid[-1][-1]
```

734. Sentence Similarity

I will transform each array in pairs as tuple and add this tuple to a set. Then we will go through words1 and words2 at the same time. If they are not equal and (words1[i], words2[i])(words2[i], words1[i]) not in pairs_set, we will return False. Or in the end, we will return True. TC is O(n)

```
class Solution:
    def areSentencesSimilar(self, words1: List[str], words2: List[str], pairs: List[List[str]]) -> bool:
        if len(words1) != len(words2):
          return False
        words_pair = set()
        for pair in pairs:
          words_pair.add(tuple(pair))
        for i in range(len(words1)):
          if (words1[i], words2[i]) not in words_pair and (words2[i], words1[i]) not in words_pair and words2[i] != words1[i]:
            return False
        return True
```