Letter Combination of Phone Number LeetCode

Given a string containing digits, the task is to generate all possible letter combinations that can be formed using the given digits. Each digit corresponds to a set of letters as follows:

- Digit 2 corresponds to 'abc'
- Digit 3 corresponds to 'def'
- Digit 4 corresponds to 'ghi'
- Digit 5 corresponds to 'jkl'
- Digit 6 corresponds to 'mno'
- Digit 7 corresponds to 'pqrs'
- Digit 8 corresponds to 'tuv'
- Digit 9 corresponds to 'wxyz'

Write a program to find and print all possible letter combinations of the given digits.

Approach 1: Function to find all possible letter combinations for the given digits using Backtracking approach

The program aims to solve the problem using a backtracking approach. Backtracking is a technique where the program systematically explores all possible solutions by making choices and then undoing those choices if they do not lead to a valid solution. The given digits are translated into a mapping of corresponding strings of letters. The idea is to generate all possible combinations by recursively exploring each digit's corresponding letters.

1. Initialize an empty vector **ans** to store the generated combinations.
2. Define a recursive function **solve** that takes the following parameters:
    - **digits**: The input string of digits.
    - **output**: The current letter combination being formed.
    - **index**: The index of the current digit being processed.
    - **mapping**: An array mapping each digit to its corresponding letters.
    - **ans**: A reference to the vector storing the final answer.
3. In the **solve** function:
    - If **index** is greater than or equal to the length of **digits**, it means all digits have been considered, so the current **output** is a valid combination and should be added to **ans**.

- Otherwise, get the integer value of the current digit.
- Get the corresponding string of letters for the current digit.
- Iterate through each letter of the current digit's letters:
    - Include the current letter in the **output**.
    - Recursively call **solve** for the next digit's combination.
    - Remove the last letter from the **output** to backtrack.

4. In the **letterCombinations** function:
    - If the input **digits** string is empty, return an empty vector.
    - Initialize an empty string **output** to store the current combination being formed.
    - Create a mapping array **mapping** of digit-to-letters.
    - Start the recursive process by calling the **solve** function with initial values of **digits**, **output**, **index**, **mapping**, and **ans**.
5. In the **main** function:
    - Initialize the input string **digits**.
    - Call the **letterCombinations** function to get the final answer.
    - Print all the generated combinations.

Time Complexity: The backtracking approach explores all possible combinations of letters for the given digits. In the worst case, each digit can correspond to four letters. Therefore, the total number of recursive calls will be O(4^n), where 'n' is the number of digits in the input string. Since each recursive call involves constant time operations, the overall time complexity is O(4^n).

Space Complexity: The space complexity is primarily determined by the space used for the recursive call stack and the space used to store the final combinations. The maximum depth of the recursive call stack is 'n', the number of digits. Additionally, the space used to store the combinations in the vector is O(4^n) since there can be that many possible combinations. Thus, the space complexity is O(n + 4^n), considering the call stack space and the output vector space.