Subsequences Of String CodeStudio

This C++ program generates all possible subsequences of a given string using three different approaches: backtracking, an alternative backtracking implementation, and a bitwise approach.

Approach 1: Function to find all subsets of a given string using backtracking approach

- The **solve** function uses recursive backtracking to generate all possible combinations of characters from the input string **str**.
- The base case is when the **index** becomes greater than or equal to the length of **str**. In this case, if the **output** is not empty, it's added to the answer.
- It explores two possibilities at each step: excluding the current character and including it in the subset.
- The main function **subsequences** initializes the **output** as an empty string and starts the backtracking process.
- **Time Complexity: Exponential, O(2^n), where n is the length of the input string str. In the worst case, it generates 2^n subsets.**
- **Space Complexity: O(n), as the maximum depth of the recursive call stack is n.**

Approach 2: Function to find all subsequences of a given string using backtracking approach alternative implementation

- The **findSubsequences** function is an alternative implementation of backtracking.
- It iterates through the input string starting from the given **index**.
- At each step, it appends the character at the current index to the **output**, then recursively explores the next characters.
- After exploring all possibilities with the current character included, the last character is removed from the **output**.
- **Time Complexity: Exponential, O(2^n), similar to the first approach.**
- **Space Complexity: O(n), same as the first approach.**

Approach 3: Function to find the power set of a string using the Bitwise approach

- The **subsequencesBitwise** function generates subsequences using a bitwise approach.

- It uses bit manipulation to represent whether a character from the string is included or not in each subset.
- The outer loop iterates through all possible subsets (2^n iterations) using a bitmask **i**.
- The inner loop checks the bits of the **i** bitmask and adds the corresponding character to the **subsets** string.
- If **subsets** is not empty after the inner loop, it's added to the answer.
- **Time Complexity: O(n * 2^n), as there are 2^n subsets, and in each subset, characters are checked for inclusion in O(n) time.**
- **Space Complexity: O(n * 2^n), as there are 2^n subsets and the average length of each subset is n.**