1092. Shortest Common Supersequence

Given two strings `str1` and `str2`, return *the shortest string that has both* `str1` *and* `str2` *as* ***subsequences***. If there are multiple valid strings, return **any** of them.

A string `s` is a **subsequence** of string `t` if deleting some number of characters from `t` (possibly `0`) results in the string `s`.

Example 1:

```
Input: str1 = "abac", str2 = "cab"
Output: "cabac"
Explanation:
str1 = "abac" is a subsequence of "cabac" because we can delete the first "c".
str2 = "cab" is a subsequence of "cabac" because we can delete the last "ac".
The answer provided is the shortest such string that satisfies these properties.
```

Example 2:

```
Input: str1 = "aaaaaaaa", str2 = "aaaaaaaa"
Output: "aaaaaaaa"
```

Constraints:

- `1 <= str1.length, str2.length <= 1000`
- `str1` and `str2` consist of lowercase English letters.

Overview

We are given two strings, `str1` and `str2`, and our goal is to construct the shortest string that contains both as subsequences. If multiple valid solutions exist, we can return any of them.

A supersequence of a string is a sequence that includes the original string as a subsequence. This means we can derive the original string by removing certain characters without altering the relative order of the remaining ones.

> The Shortest Common Supersequence (SCS) is the smallest string that contains both `str1` and `str2` as subsequences.

This problem is closely linked to the Longest Common Subsequence (LCS). A strong understanding of LCS allows us to efficiently construct the SCS. If this concept is unfamiliar, it is highly recommended to first solve the following problems:

- 1143. Longest Common Subsequence
- 516. Longest Palindromic Subsequence
- 1062. Longest Repeating Substring

> Note: The LCS represents the longest sequence of characters that appear in both strings in the same order. To form the SCS, we preserve the LCS while inserting the remaining characters from both strings around it, ensuring that the final sequence maintains the relative order of all characters.

1143. Longest Common Subsequence

Given two strings `text1` and `text2`, return *the length of their longest* ***common subsequence***. If there is no **common subsequence**, return `0`.

A **subsequence** of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

- For example, `"ace"` is a subsequence of `"abcde"`.

A **common subsequence** of two strings is a subsequence that is common to both strings.

Example 1:

```
Input: text1 = "abcde", text2 = "ace"
Output: 3
Explanation: The longest common subsequence is "ace" and its length is 3.
```

Example 2:

```
Input: text1 = "abc", text2 = "abc"
Output: 3
Explanation: The longest common subsequence is "abc" and its length is 3.
```

Example 3:

```
Input: text1 = "abc", text2 = "def"
Output: 0
Explanation: There is no such common subsequence, so the result is 0.
```

Constraints:

- `1 <= text1.length, text2.length <= 1000`
- `text1` and `text2` consist of only lowercase English characters.

1143. Longest Common Subsequence

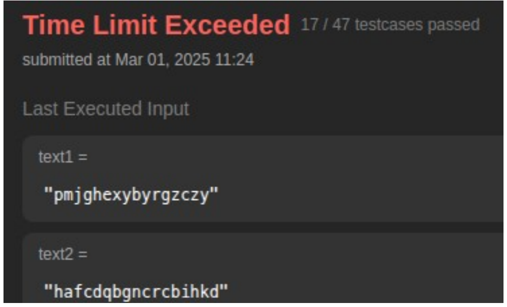
/*

Recursion

Time complexity: $O\left(2^{n+m}\right)$

Space complexity:O(max(n,m))

*/



Time Limit Exceeded 17 / 47 testcases passed

submitted at Mar 01, 2025 11:24

Last Executed Input

text1 =

"pmjghexybyrgzczy"

text2 =

"hafcdqbgncrcbihkd"

```
class Solution {
   public:
      int longestCommonSubsequence(std::string text1, std::string text2) {
         int n=text1.size(),m=text2.size();
         auto lcs=[&](int i,int j,auto& self)->int{
            if(i==n || j==m) return 0;
            if(text1[i]==text2[j]) return 1+self(i+1,j+1,self);
            return std::max(self(i+1,j,self),self(i,j+1,self));
         };
         return lcs(0,0,lcs);
      }
};
```

1143. Longest Common Subsequence

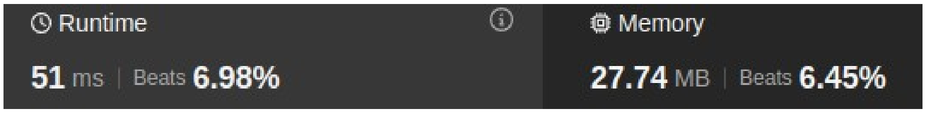
/*

Top-Down

Time complexity: O(nm)

Space complexity:O(max(n,m))

*/



Runtime

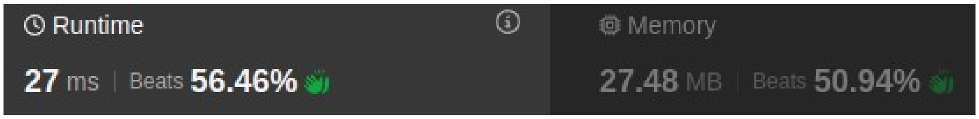
51 ms | Beats 6.98%

Memory

27.74 MB | Beats 6.45%

```
class Solution {
public:
   int longestCommonSubsequence(std::string text1, std::string text2) {
      int n=text1.size(),m=text2.size();
      std::vector<std::vector<int>> memo(n+1,std::vector<int>(m+1,-1));
      auto lcs=[&](int i,int j,auto& self)->int{
         if(i==n || j==m) return memo[i][j]=0;
         if (memo[i][j]!=-1) return memo[i][j];
         if(text1[i]==text2[j]) return memo[i][j]=1+self(i+1,j+1,self);
         return memo[i][j]=std::max(self(i+1,j,self),self(i,j+1,self));
      };
      return lcs(0,0,lcs);
   }
};
```

1143. Longest Common Subsequence



Runtime

27 ms | Beats 56.46%

Memory

27.48 MB | Beats 50.94%

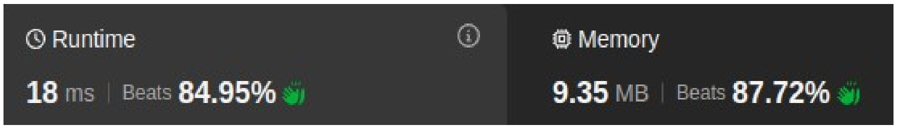
```
/*
    Bottom Up
    Time complexity: O(nm)
    Space compelxity:O(nm)
*/
class Solution {
public:
    int longestCommonSubsequence(std::string text1, std::string text2) {
        int n=text1.size(),m=text2.size();
        std::vector<std::vector<int>> dp(n+1,std::vector<int>(m+1,-1));
        auto lcs=[&]()->int{
          for(int i=0;i<=n;++i) dp[i][m]=0;
          for(int j=0;j<=m;++j) dp[n][j]=0;
          for(int i=n-1;i>=0;--i){
            for(int j=m-1;j>=0;--j){
                if(text1[i]==text2[j]) dp[i][j]=1+dp[i+1][j+1];
                else dp[i][j]=std::max(dp[i+1][j],dp[i][j+1]);
            }
          }
          return dp[0][0];
        };
        return lcs();
    }
};
```

1143. Longest Common Subsequence

Restore the LCS

```
auto get_lcs=[&]()->std::string{
      int i=0,j=0;
      std::string ans;
      while(i<n&&j<m){
        if(text1[i]==text2[j]){
          ans.push_back(text1[i]);
          i++;
          j++;
        }
        else{
          if(dp[i+1][j]>dp[i][j+1]) i++;
          else j++;
        }
      }
      return ans;
    };
```

1143. Longest Common Subsequence



Runtime

18 ms | Beats 84.95%

Memory

9.35 MB | Beats 87.72%

```
/*
   Bottom Up: Space optimization
   Time complexity: O(nm)
   Space complexity:O(2*min(n,m))
*/
class Solution {
public:
   int longestCommonSubsequence(std::string text1, std::string text2) {
      int n=text1.size(),m=text2.size();
      if(n<m){
         std::swap(n,m);
         std::swap(text1,text2);
      }
      std::vector<std::vector<int>> dp(2,std::vector<int>(m+1,0));
      auto lcs=[&]()->int{
         for(int i=n-1;i>=0;--i){
            for(int j=m-1;j>=0;--j){
               if(text1[i]==text2[j]) dp[0][j]=1+dp[1][j+1];
               else dp[0][j]=std::max(dp[1][j],dp[0][j+1]);
            }
            dp[1]=dp[0];
         }

         return dp[0][0];
      };

      return lcs();
   }
};
```

1092. Shortest Common Supersequence

```
/*
    Phase#1: Recursion Top-Down: build DP table to find the longest common subsequence(LCS)
    Phase#2: Use DP table to build the answer.
    Time complexity: O(nm+(n+m))
    Space complexity: O(nm)
*/
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;

class Solution {
    public:
        std::string shortestCommonSupersequence(std::string str1, std::string str2){
            int n=str1.size();
            int m=str2.size();

            // Phase #1: Determine Longest common subsequence

            // Memoize answers
            vvi memo(n+1,vi(m+1,0));

            // Recursive function to compute longest common subsequence
            auto lcs=[&](int i,int j,auto& self)->int{
                if(i==n || j==m) return 0;
                if (memo[i][j] != 0) return memo[i][j];
                if(str1[i]==str2[j]) return memo[i][j]=1+self(i+1,j+1,self);
                return memo[i][j]=std::max(self(i+1,j,self),self(i,j+1,self));
            };
            // Phase #2: Get the shortest common supersequence

            // Using the DP memo array of phase #1 to build the answer
            auto get_lcs=[&]()->std::string{
                std::string ans="";
                int i=0,j=0;
                // While both string not terminated
                while(i<n&&j<m){
                    // If letters are not equal
                    if(str1[i]!=str2[j]){
                        // If the result in memo[i][j] came from the down cell
                        if(memo[i+1][j]>memo[i][j+1]){
                            ans+=str1[i];
                            i++;
                        }
                        // If the result in memo[i][j] came from the right cell
                        else{
                            ans+=str2[j];
                            j++;
                        }
                    }
                    // If letters are equal
                    else{
                        ans+=str1[i];
                        i++;
                        j++;
                    }
                }

                // Add remaining letters to answer
                while(i<n) ans+=str1[i++];
                while(j<m) ans+=str2[j++];

                return ans;
            };

            lcs(0,0,lcs);

            return get_lcs();
        }
};
```