27) Longest Common Subsequence

07 December 2024 18:46

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

From <https://leetcode.com/problems/longest-common-subsequence/description/>

```
int helper(int i, int j, string &s, string &t, vector<vector<int>> &dp){
    if(i < 0 || j < 0) return 0;
    if(dp[i][j] != -1) return dp[i][j];
    if(s[i] == t[j]) return 1 + helper(i-1, j-1, s, t, dp);
    return dp[i][j] = max(helper(i-1, j, s, t, dp), helper(i, j-1, s, t, dp));
}
int longestCommonSubsequence(string text1, string text2) {
    int n = text1.size();
    int m = text2.size();
    vector<vector<int>> dp(n, vector<int>(m, -1));
    return helper(n-1, m-1, text1, text2, dp);
}
```

28) Print all LCS sequences (based on prev problem) (code not given by striver)

07 December 2024 19:32

You are given two strings **s** and **t**. Now your task is to print all longest common sub-sequences in lexicographical order.

From <https://www.geeksforgeeks.org/problems/print-all-lcs-sequences3413/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=print-all-lcs-sequences>

```
  void findAllLCS(int i, int j, string &s, string &t, vector<vector<int>> &dp, string currentLCS,
set<string> &lcsSet) {
      if (i == 0 || j == 0) {
          reverse(currentLCS.begin(), currentLCS.end());
          lcsSet.insert(currentLCS);
          return;
      }

      if (s[i - 1] == t[j - 1]) {
          currentLCS.push_back(s[i - 1]);
          findAllLCS(i - 1, j - 1, s, t, dp, currentLCS, lcsSet);
      } else {
          if (dp[i - 1][j] == dp[i][j])
              findAllLCS(i - 1, j, s, t, dp, currentLCS, lcsSet);
          if (dp[i][j - 1] == dp[i][j])
              findAllLCS(i, j - 1, s, t, dp, currentLCS, lcsSet);
      }
  }
  vector<string> all_longest_common_subsequences(string &s, string &t) {
      int n = s.size(), m = t.size();
      vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));

      // Fill the DP table
      for (int i = 1; i <= n; i++) {
          for (int j = 1; j <= m; j++) {
              if (s[i - 1] == t[j - 1]) {
                  dp[i][j] = 1 + dp[i - 1][j - 1];
              } else {
                  dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
              }
          }
      }

      set<string> lcsSet;
      findAllLCS(n, m, s, t, dp, "", lcsSet);

      // Convert set to vector
      vector<string> result(lcsSet.begin(), lcsSet.end());
      return result;
  }
```

29) Longest Common Substring

08 December 2024 15:51

You are given two strings **s1** and **s2**. Your task is to find the length of the **longest common substring** among the given strings.

From <https://www.geeksforgeeks.org/problems/longest-common-substring1452/1>

//same as q no 27, just isme tabulation lga ke
Dp[i][j] = 0; in else case

```
 int longestCommonSubstr(string& s1, string& s2) {
    int n = s1.size();
    int m = s2.size();
    vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));
    int maxLen = 0;

    for (int i = 1; i <= n; i++) {
       for (int j = 1; j <= m; j++) {
          if (s1[i - 1] == s2[j - 1]) {
             dp[i][j] = dp[i - 1][j - 1] + 1;
             maxLen = max(maxLen, dp[i][j]);
          }
          else dp[i][j] = 0;
       }
    }
    return maxLen;
 }
```

//memoized sol

```
 int helper(int i, int j, string& text1, string& text2, vector<vector<int>>& dp) {
    if (i < 0 || j < 0) return 0;
    if (dp[i][j] != -1) return dp[i][j];

    if (text1[i] == text2[j]) return dp[i][j] = 1 + helper(i - 1, j - 1, text1, text2, dp);
    return dp[i][j] = 0;
 }

 int longestCommonSubstr(string& s1, string& s2) {
    int n = s1.size(), m = s2.size();
    vector<vector<int>> dp(n, vector<int>(m, -1));
    int maxLen = 0;

    for (int i = 0; i < n; i++) {
       for (int j = 0; j < m; j++) {
          helper(i, j, s1, s2, dp);
          maxLen = max(maxLen, dp[i][j]);
       }
    }
    return maxLen;
}
```

30) Longest palindromic subsequence (same as q no 27)

08 December 2024 16:02

Given a string s, find *the longest palindromic* ***subsequence****'s length in* s.
A **subsequence** is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements.

From <https://leetcode.com/problems/longest-palindromic-subsequence/description/>

```
int helper(int i, int j, string &s, string &t, vector<vector<int>> &dp){
    if(i < 0 || j < 0) return 0;
    if(dp[i][j] != -1) return dp[i][j];
    if(s[i] == t[j]) return 1 + helper(i-1, j-1, s, t, dp);
    return dp[i][j] = max(helper(i-1, j, s, t, dp), helper(i, j-1, s, t, dp));
}
int longestPalindromeSubseq(string s) {
    string s2 = s;
    reverse(s2.begin(), s2.end());
    int n = s.size();
    vector<vector<int>> dp(n, vector<int>(n, -1));
    return helper(n-1, n-1, s, s2, dp);
}
```

31) Minimum insertion steps to make a string palindrome (almost same as previous problem)

08 December 2024 16:12

Given a string s. In one step you can insert any character at any index of the string.
Return *the minimum number of steps* to make s palindrome.
A **Palindrome String** is one that reads the same backward as well as forward.

From <https://leetcode.com/problems/minimum-insertion-steps-to-make-a-string-palindrome/description/>

```
    int helper(int i, int j, string &s, string &t, vector<vector<int>> &dp){
        if(i < 0 || j < 0) return 0;
        if(dp[i][j] != -1) return dp[i][j];
        if(s[i] == t[j]) return 1 + helper(i-1, j-1, s, t, dp);
        return dp[i][j] = max(helper(i-1, j, s, t, dp), helper(i, j-1, s, t, dp));
    }
    int minInsertions(string s) {
        string s2 = s;
        reverse(s2.begin(), s2.end());
        int n = s.size();
        vector<vector<int>> dp(n, vector<int>(n, -1));
        int lps = helper(n-1, n-1, s, s2, dp);
        return n-lps;
    }
```

32) **Minimum number of deletions and insertions (**based on q no 27**)**

08 December 2024 16:20

Given two strings **s1** and **s2**. The task is to **remove or insert** the **minimum number** of characters from/in **s1** to transform it into **s2**. It could be possible that the same character needs to be removed from one point of **s1** and inserted into another point.

From <https://www.geeksforgeeks.org/problems/minimum-number-of-deletions-and-insertions0209/1?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=practice_card>

```
int helper(int i, int j, string &s, string &t, vector<vector<int>> &dp){
      if(i < 0 || j < 0) return 0;

      if(dp[i][j] != -1) return dp[i][j];
      if(s[i] == t[j]) return 1 + helper(i-1, j-1, s, t, dp);
      return dp[i][j] = max(helper(i-1, j, s, t, dp), helper(i, j-1, s, t, dp));
  }
  int minOperations(string &s1, string &s2) {
      int n = s1.size();
      int m = s2.size();
      vector<vector<int>> dp(n, vector<int>(m, -1));
      int lcs = helper(n-1, m-1, s1, s2, dp);
      return n+m- 2*(lcs);
  }
```

33) Delete operation for two strings (exactly same code as prev problem)

08 December 2024 16:24

Given two strings word1 and word2, return *the minimum number of* ***steps*** *required to make* word1 *and* word2 *the same*.
In one **step**, you can delete exactly one character in either string.

From <https://leetcode.com/problems/delete-operation-for-two-strings/description/>

```
    int helper(int i, int j, string &s, string &t, vector<vector<int>> &dp){
        if(i < 0 || j < 0) return 0;
        if(dp[i][j] != -1) return dp[i][j];
        if(s[i] == t[j]) return 1 + helper(i-1, j-1, s, t, dp);
        return dp[i][j] = max(helper(i-1, j, s, t, dp), helper(i, j-1, s, t, dp));
    }
    int minDistance(string s1, string s2){
        int n = s1.size();
        int m = s2.size();
        vector<vector<int>> dp(n, vector<int>(m, -1));
        int lcs = helper(n-1, m-1, s1, s2, dp);
        return n+m- 2*(lcs);
    }
```

34) Shortest common supersequence (based on q no 27 and concept of printing of longest common subsequence)

08 December 2024 17:21

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.

From <https://leetcode.com/problems/shortest-common-supersequence/description/>

```
    string shortestCommonSupersequence(string s1, string s2) {
        int n = s1.size();
        int m = s2.size();
        vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));
        for (int ind1 = 1; ind1 <= n; ind1++) {
            for (int ind2 = 1; ind2 <= m; ind2++) {
                if (s1[ind1 - 1] == s2[ind2 - 1])
                    dp[ind1][ind2] = 1 + dp[ind1 - 1][ind2 - 1];
                else
                    dp[ind1][ind2] = max(dp[ind1 - 1][ind2], dp[ind1][ind2 - 1]);
            }
        }
        int i = n, j = m;
        string ans = "";
        while (i > 0 && j > 0) {
            if (s1[i - 1] == s2[j - 1]) {
                ans += s1[i - 1];
                i--;
                j--;
            } else if (dp[i - 1][j] > dp[i][j - 1]) {
                ans += s1[i - 1];
                i--;
            } else {
                ans += s2[j - 1];
                j--;
            }
        }
        while (i > 0) {
            ans += s1[i - 1];
            i--;
        }
        while (j > 0) {
            ans += s2[j - 1];
            j--;
        }
        reverse(ans.begin(), ans.end());
        return ans;
    }
```

```
// memoization
```

```
    int helper(string& s1, string& s2, int i, int j, vector<vector<int>>& dp){
        if(i < 0 || j < 0) return 0;
        if(dp[i][j] != -1) return dp[i][j];
        if(s1[i] == s2[j]) return dp[i][j] = 1 + helper(s1, s2, i-1, j-1, dp);
        return dp[i][j] = max(helper(s1, s2, i-1, j, dp), helper(s1, s2, i, j-1,
dp));
    }
    string shortestCommonSupersequence(string s1, string s2) {
        int n = s1.length();
        int m = s2.length();
        vector<vector<int>> dp(n, vector<int>(m, -1));
        int len = helper(s1, s2, n-1, m-1, dp);
        string ans = "";
        int i = n-1; int j = m-1;
        while(i >= 0 && j >= 0){
            if(s1[i] == s2[j]){
                ans += s1[i];
                i--; j--;
            }
            else if (i > 0 && dp[i - 1][j] > (j > 0 ? dp[i][j - 1] : 0)){
                ans += s1[i];
                i--;
            }
            else{
                ans += s2[j];
                j--;
            }
        }
        while(i >= 0){
            ans += s1[i];
            i--;
        }
        while(j >= 0){
            ans += s2[j];
            j--;
        }
        reverse(ans.begin(), ans.end());
        return ans;
    }
```

35) Distinct Subsequence

09 December 2024 01:39

Given two strings s and t, return *the number of distinct* ***subsequences*** *of* s *which equals* t.
The test cases are generated so that the answer fits on a 32-bit signed integer.

From <https://leetcode.com/problems/distinct-subsequences/description/>

```
    int helper(int i, int j, string &s, string &t, vector<vector<int>> &dp){
        if(j < 0) return 1;
        if(i < 0) return 0;
        if(dp[i][j] != -1) return dp[i][j];
        if(s[i] == t[j]) return dp[i][j] = (helper(i-1, j, s, t, dp) + helper(i-1, 
j-1, s, t, dp));
        return dp[i][j] = helper(i-1, j, s, t, dp);
    }
    int numDistinct(string s, string t) {
        int m = s.length();
        int n = t.length();
        vector<vector<int>>dp(m, vector<int>(n, -1));
        return helper(m-1, n-1, s, t, dp);
    }
```

36) Edit distance

09 December 2024   02:00

Given two strings word1 and word2, return *the minimum number of operations required to convert word1 to word2.*
You have the following three operations permitted on a word:

- Insert a character
- Delete a character
- Replace a character

From <https://leetcode.com/problems/edit-distance/description/>

```
    int solve(int i, int j, string &s1, string &s2, vector<vector<int>> &dp){
        if(i < 0) return j+1;
        if(j < 0) return i+1;
        if(dp[i][j] != -1) return dp[i][j];
        if(s1[i] == s2[j]) return dp[i][j] = solve(i-1, j-1, s1, s2, dp);
        return dp[i][j] = 1 + min(solve(i-1, j-1, s1, s2, dp), min(solve(i-1, j, s1, 
s2, dp), solve(i, j-1, s1, s2, dp)));
    }
    int minDistance(string word1, string word2) {
        int m = word1.length();
        int n = word2.length();
        vector<vector<int>> dp(m, vector<int>(n, -1));
        return solve(m-1, n-1, word1, word2, dp);
    }
```

37) Wildcard matching

09 December 2024 02:31

Given an input string (s) and a pattern (p), implement wildcard pattern matching with support for '?' and '*' where:

- '?' Matches any single character.
- '*' Matches any sequence of characters (including the empty sequence).

The matching should cover the **entire** input string (not partial).

From <https://leetcode.com/problems/wildcard-matching/description/>

```
    bool helper(int i, int j, string &s, string &p, vector<vector<int>> &dp){
        if(i < 0 && j < 0) return true;
        if(i < 0 && j >= 0) return false;
        if(i >= 0 && j < 0){
            for(int k = 0; k <= i; k++){
                if(p[k] != '*') return false;
            }
            return true;
        }
        if(dp[i][j] != -1) return dp[i][j];
        if(p[i] == s[j] || p[i] == '?') return dp[i][j] = helper(i-1, j-1, s, p, 
dp);
        if(p[i] == '*') return dp[i][j] = helper(i-1, j, s, p, dp) || helper(i, j-1, 
s, p, dp);
        return dp[i][j] = false;
    }
    bool isMatch(string s, string p) {
        int n = s.length();
        int m = p.length();
        vector<vector<int>>dp(m, vector<int>(n, -1));
        return helper(m-1, n-1, s, p, dp);
    }
```

28)

17 February 2025 19:21

You are given two strings ***'s1'*** and ***'s2'***.

Return the longest common subsequence of these strings.

If there's no such string, return an empty string. If there are multiple possible answers, return any such string.

Note:
Longest common subsequence of string 's1' and 's2' is the longest subsequence of 's1' that is also a subsequence of 's2'. A 'subsequence' of 's1' is a string that can be formed by deleting one or more (possibly zero) characters from 's1'.

From <https://www.naukri.com/code360/problems/print-longest-common-subsequence_8416383?leftPanelTabValue=PROBLEM>

```
int solve(int n, int m, string &s1, string &s2, vector<vector<int>> &dp) {
    if (n < 0 || m < 0) {
        return 0;
    }
    if (dp[n][m] != -1) {
        return dp[n][m];
    }
    if (s1[n] == s2[m]) {
        return dp[n][m] = 1 + solve(n - 1, m - 1, s1, s2, dp);
    }
    return dp[n][m] = max(solve(n - 1, m, s1, s2, dp), solve(n, m - 1, s1, s2, dp));
}
string findLCS(int n, int m, string &s1, string &s2) {  // Accepts four arguments
    vector<vector<int>> dp(n, vector<int>(m, -1));
    // Get LCS length using the solve function
    int lcs_length = solve(n - 1, m - 1, s1, s2, dp);

    // Reconstruct the LCS string
    string lcs = "";
    int i = n - 1, j = m - 1;
    while (i >= 0 && j >= 0) {
        if (s1[i] == s2[j]) {
            lcs += s1[i];
            i--;
            j--;
        } else if (i > 0 && dp[i - 1][j] >= (j > 0 ? dp[i][j - 1] : 0)) {
            i--;
        } else {
            j--;
        }
    }
    // Reverse the LCS string since we built it from the end
    reverse(lcs.begin(), lcs.end());
    return lcs;
}
```