Python very detailed solution with explanation and walkthrough step by step.

hxuanhung ★ 371 Last Edit: November 25, 2019 12:32 AM 3.8K VIEWS

65

Why might we want to solve the longest common subsequence problem?

> File comparison. The Unix program "diff" is used to compare two different versions of the same file, to d changes have been made to the file. It works by finding a longest common subsequence of the lines of t line in the subsequence has not been changed, so what it displays is the remaining set of lines that have instance of the problem we should think of each line of a file as being a single complicated character in

Solution

1. Recursive solution

```
class Solution:
    def longestCommonSubsequence(self, s1: str, s2: str) -> int:
        return self.helper(s1, s2, 0, 0)

    def helper(self, s1, s2, i, j):
        if i == len(s1) or j == len(s2):
            return 0
        if s1[i] == s2[j]:
            return 1 + self.helper(s1, s2, i + 1, j + 1)
        else:
            return max(self.helper(s1, s2, i+1, j), self.helper(s1, s2, i, j + 1))
```

If the two strings have no matching characters, so the last line always gets executed, the the time bounds ar coefficients, which (if m=n) are close to 2^n.

```
                        lcs("AXYT", "AYZX")
                       /                   \
          lcs("AXY", "AYZX")                lcs("AXYT", "AYZ")
          /          \                      /                \
lcs("AX", "AYZX") lcs("AXY", "AYZ")   lcs("AXY", "AYZ") lcs("AXYT", "AY")
```

2. Recursive solution with Memoization

```
class Solution:
    def longestCommonSubsequence(self, s1: str, s2: str) -> int:
        m = len(s1)
        n = len(s2)
        memo = [[-1 for _ in range(n + 1)] for _ in range(m + 1)]
        return self.helper(s1, s2, 0, 0, memo)
```