Homework 5

Robbie McKinstry, Jack McQuown, Cyrus Ramavarapu

12 September 2016

Greedy Problems

Problem 7:

Problem 17:

Dynamic Programming

Problem 2:

The longest common subsequence between the three can be initially defined recursively by considering the last letter in common between the three strings and then seeing if this letter is in the longest common subsequence of one letter shorter.

$$S_A = A_1, A_2, A_3, \ldots, A_{i-1}, A_i$$
$$S_B = B_1, B_2, B_3, \ldots, B_{j-1}, B_j$$
$$S_C = C_1, C_2, C_3, \ldots, C_{k-1}, C_k$$

Reading right to left, the first letter common to each string will be the last letter in the longest common subsequence. As a result, once this letter is found, problem can be redefined in terms of the shorter substrings formed by ignoring the first common point and all suceeding letters.

This analysis leads to the following recursive algorithm:

Function: *LCS*
Input: *int i, int j, int k*
if $i \equiv j \equiv k \equiv 0$ **then**
| *return* 0
end
if $A_i \equiv B_j \equiv C_k$ **then**
| $LCS(i-1, j-1, k-1) + A_i$
else if $(A_i \equiv B_j) \neq C_k$ **then**
| $\max(LCS(i-1,\ j-1,\ k),\ LCS(i,\ j,\ k-1))$
else if $(A_i \equiv C_k) \neq B_j$ **then**
| $\max(LCS(i-1,\ j,\ k-1),\ LCS(i, j-1, k))$
else if $A_i \neq (B_j \equiv C_k)$ **then**
| $\max(LCS(i,\ j-1,\ k-1),\ LCS(i-1,\ j,\ k))$
else
| $\max(LCS(i,\ j-1,\ k-1), LCS(i-1,\ j,\ k-1), LCS(i-1,\ j-1,\ k))$
end

Although this algorithm produces the longest common subsequence for three given strings, it runs in exponential time due to the numerous recursive calls that operate on a problem of only 1 letter smaller.

By moving to an array based solution and changing the recursive calls to array look-ups, a polynomial runtime algorithm can be developed.

Function: *Array LCS*
Input: *string A, String B, String C*
Initialization:
Array[ ][ ][ ] *LCS*
for $i \leftarrow 0$ *to len*(A) **do**
  $LCS[i][0][0] = 0$
end
for $j \leftarrow 0$ *to len*(B) **do**
  $LCS[0][j][0] = 0$
end
for $k \leftarrow 0$ *to len*(C) **do**
  $LCS[0][0][k] = 0$
end
Array Calculations:
for $i \leftarrow 0$ *to len*(A) **do**
  for $j \leftarrow 0$ *to len*(B) **do**
    for $k \leftarrow 0$ *to len*(C) **do**
      if $A_i \equiv B_j \equiv C_k$ **then**
        $LCS[i][j][k] = LCS[i-1][j-1][k-1] + 1$
      else if $(A_i \equiv B_j) \neq C_k$ **then**
        $LCS[i][j][k] = \max(LCS[i-1][j-1][k],\ LCS[i][j][k-1])$
      else if $(A_i \equiv C_k) \neq B_j$ **then**
        $\max(LCS[i][j][k] = LCS[i-1][j][k-1],\ LCS[i][j-1][k])$
      else if $A_i \neq (B_j \equiv C_k)$ **then**
        $LCS[i][j][k] = \max(LCS[i][j-1][k-1],\ LCS[i-1][j],[k])$
      else
        $LCS[i][j][k] = \max(LCS[i][j-1][k-1],$
        $LCS[i-1][j][k-1],\ LCS[i-1][j-1][k])$
      end
    end
  end
end