Homework 6

Robbie McKinstry, Jack McQuown, Cyrus Ramavarapu

13 September 2016

Problem 3:

The shortest common super-sequence between A and B can be defined recursively by considering the last letter of A and B and concatenating one of the letters onto a new string C depending on if the letters are equal or not. The algorithm will read A and B from right to left, and if the letters are equal it will concatenate that letter to the string C. If the letters are not equal, it will recursively form the String C using both letters and choose the string C that has the shortest length.

This can be expressed in the following algorithm.

```
Function: super
/* String A, B as globals */
Input: int i, int j
if $A[i] \equiv B[j]$ then
    return $super(i-1, j-1, A, B)$
else
    return $\min(len(super(i-1, j) \cdot A[i]), len(super(i, j-1) \cdot B[j]))$
end
```

Although this algorithm produces the shortest common super-sequence for the given strings A and B, it runs in exponential time because of the recursive calls that decrease the problem size by 1.

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

```
Function: super
/* String A, B as globals */
Input: int i, int j
Initialization:
Array[ ][ ][ ] super
for $i \leftarrow 0$ to $len(A)$ do
    $super[i][0] = i$
end
for $j \leftarrow 0$ to $len(B)$ do
    $super[0][j] = j$
end
Array Calculations:
for $i \leftarrow 0$ to $len(A)$ do
    for $j \leftarrow 0$ to $len(B)$ do
        if $A[i] \equiv B[j]$ then
            $super[i][j] = super[i-1][j-1] + 1$
        else
            $super[i][j] = \min(super[i-1][j], super[i][j-1]) + 1$
        end
    end
end
```

(a)

Below is the table that the SCS algorithm constructs for inputs A = zxyyzz, and B = zzyxzy

Table 1: SCS Table

	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	1	2	3	4	5	6
2	2	2	3	4	5	5	6
3	3	3	4	4	5	6	7
4	4	4	4	5	6	7	8
5	5	5	5	6	7	7	8
6	6	6	6	6	7	8	9

(b)

The length of the shortest common super-sequence can be found in the bottom right of the table, or index (6,6)

(c)

Table 2: SCS Table

	0	1	2	3	4	5	6
0							
1							
2		↖	←				
3				↖			
4				↑	←		
5						↖	
6						↑	←

The way to compute the shortest common super-sequence from the table is to follow the arrows above and record the letter at the index that the arrow points to. When A = zxyyzz and B = zzyxzy, the first letter to record is C = z from A_6. Then record the letter y from B_6, so that C = yz. Then move to (5,5) in the table where we have a match, $A_5 \equiv B_5$, and record the letter z, C = zyz. Then move to (4,4) in the table where no match occurs so we record the letter x from B_4 and y from A_4, where now C = yxzyz. Then move to (3,3) in the table where we have another match, $A_3 \equiv B_3$, and record the letter y, C = yyxzyz. Then continue following the arrows and recording the appropriate letters from A and B, and then adding them to C.

Problem 4:

In order to find the cheapest total cost to convert a string A to a string B, we can define a recursive solution that will attempt each option and choose the string of options with the minimum total cost. This recursive solution will read the strings A and B from right to left, and move farther down the string according to the action that is performed.

This is expressed in the following algorithm

```
Function: diff
Input: int i, int j, String A, String B
if $A[i] \equiv B[j]$ then
    return $diff(i-1, j-1)$
else
    $opt_1 = diff(i-1, j, A, B) + 3$
    $opt_2 = diff(i, j-1, A, B) + 4$
    $opt_3 = diff(i-1, j-1, A, B) + 5$
    return $\min(opt_1, opt_2, opt_3)$
end
```

This algorithm will produce the shortest common super-sequence for two given strings, however it runs in exponential time due to the recursive calls that operate on a problem of only 1 or two letters smaller.

Moving to an array based, iterative solution and changing the recursive function calls to array look-ups can change this to a polynomial time algorithm.

```
**Function:** *Array diff*
**Input:** *int i, int j, String A, String B*
**Initialization:**
**Array**[ ][ ][ ] *diff*
for $i \leftarrow 0$ to $len(A)$ do
    $diff[i][0] = 0$
end
for $j \leftarrow 0$ to $len(B)$ do
    $diff[0][j] = 0$
end
**Array Calculations:**
for $i \leftarrow 0$ to $len(A)$ do
    for $j \leftarrow 0$ to $len(B)$ do
        if $A[i] \equiv B[j]$ then
            $diff[i][j] = diff[i-1][j-1]$
        else
            $diff[i][j] = \min(diff[ji-1][j] + 3,$
                              $diff[i][j-1] + 4,$
                              $diff[i-1][j-1] + 5)$
        end
    end
end
```