Hunter Caskey, (716) 720-2564
Professor John Boon
Foundations of Algorithms
JHU Whiting School of Engineering
24 November 2019

Programming Assignment #3

[50 points] Give an efficient algorithm that takes strings s, x, and y and decides if s is an interleaving of x and y. Derive the computational complexity of your algorithm.

In order to decide if s is an interleaving of x and y we will construct an algorithm consisting of two procedures: isInterleave and recurseInterleave, with isInterleave serving as a wrapper for the recurseInterleave procedure. For the purposes of analysis, let S be the length of the string s, let X be the length of string x, and let Y be the length of string Y.

The recurseInterleave procedure is fairly straightforward: it will return true if and only if some repetitions of the substrings x and y are contained within a string s. One caveat to this functionality is that it assumes that the first symbol of s is either a symbol of x or y, and any presence of 'noise' or meaningless characters attached to the beginning or the end of s causes the procedure to return false. The operation of recursiveInterleave is fairly straightforward in that on every call is shrinks the search space of s by exactly 1 character and either the length of x or y by exactly 1 character. One of the characters for x and y have been found for the first time a boolean value is set to true and passed along in the recursive calls. The function terminates is x and y have both been found in s and all of the characters in s have been processed. On each recursive call, a constant amount of comparisons are made, and the recursion ends in the worst case when all symbols of s have been examined. Thus the running time of recurseInterleave is O(S).

The isInterleave continuously calls recurseInterleave with different permutations of s in the attempt to cancel out any 'noise' that may be present that would interfere with the analysis of the two substring symbols. To do this, a double loop is used, the outer iterating over x and the inner loop iterating over y. Each execution of the double-loop will result in the construction of either 2 or 3 substrings of s, each different permutations. One substring removes characters from the fronts of s, one removes them from the end of s, and one removes them from both the front and end. The loop only executes as much as it needs to in order to derive a single viable interleaving, then terminates.

Pseudocode:

```
IS-INTERLEAVE(s, x, y):
for i=0 to x.length
    for j=0 to y.length
        substring1 = s_i...s_n
        substring2 = s_0...s_{n-j}
        substring3 = s_i...s_{n-j}
        if RECURSE-INTERLEAVE(substring1, x, y, x, y) and
           RECURSE-INTERLEAVE(substring2, x, y, x, y) and
           RECURSE-INTERLEAVE(substring3, x, y, x, y)
              return true
return false

RECURSE-INTERLEAVE(s, x', y', x, y, foundX, foundY):
if x'.length == 0
    foundX = true
    x'=x
if y'.length == 0
    foundY = true
    y'=y
if s.length == 0 and foundX and found Y
    return true
if s.length == 0
    return false
return (s_0 == x'_0 and RECURSE-INTERLEAVE(s_1...s_n, x'_1...x'_X, y', x, y, foundX, foundY))
    or (s_0 == y'_0 and RECURSE-INTERLEAVE(s_1...s_n, x', y'_1...y'_Y, x, y, foundX, foundY))
```

Analysis:

The running time for this algorithm is $O(3 \cdot XY) \cdot O(S) = 3 \cdot O(XYS) = O(XYS)$. The factor of 3 represents the 3 different permutations of s created on each execution of the double-loop, the factors of XY directly correspond to the execution of the double-loop, and the factor of S corresponds to the blackbox recurseInterleave call that is made during each execution of the double-loop. Hence, the running time of the algorithm is linear in nature and thus is efficient.

[50 points] Implement your algorithm and test its run time to verify your complexity analysis. Remember that CPU time is not a valid measure for testing run time. You must use something such as the number of comparisons.

See the included source code and programming3_tests.log for test results.

An analysis of the results of the algorithm shows that when presented with strings that have a low amount of noise on the front and end or for strings with no noise at all the algorithm behaves rather efficiently - on the order of < 10 comparisons per combined character input of s, x, and y. However, the requirement of having to rerun the main portion of the algorithm with many permutations of the input string s adds considerable to the comparisons per ratio. The worst example of such scaling can be found in the 5th test with a ratio of approximately 100 comparisons per character.