Joshua Nelson
CSCE 146

Contained Words Lab Report

Problem

Given two words, return whether the second contains each of the letters of the first. If letters are duplicated, the second word should have at least the count of the first. Use only recursive methods.

Proposed Solution

Perform initial checks (not null, not the same string, two is longer or equal to one). If one is a single character, return whether it is present anywhere in one. Sort both strings. For the first character in the first string, find the first match in the (sorted) second. Compare the shortened strings once again, with the second string *starting* from the character after the previous match.

Joshua Nelson
CSCE 146

Tests and Results

i
hi
true
hi
i
false
asdf
faste
true
ddd
deed
false
a
a
true
hi
hi
true
turkey
jerkey
false
elf
shelf
True

Problems Encountered

This lab was a challenge! I misunderstood the lab instructions because I did not read closely enough and believed we were replicating the built-in 'contains' method. I had no idea how to keep track of the number of characters in a string; to avoid having to do, I sorted each string and only passed the relevant parts onto the next recursive call. I'm not sure how this would have been done without sorting.

Joshua Nelson
CSCE 146

Conclusions and Discussion

I'm not certain why the 'contains' method was prohibited; it had little relevance to the problem and (even in the original java source!) is equivalent to `return two.indexOf(one) > -1`. I enjoyed this lab considerably as it made me think outside the box. I would note that my solution was very inefficient as it created two new character arrays on almost every call, making it very expensive with respect to memory.

Additional Questions

1. Write some pseudo-code to solve the same problem but with loops instead.

   Iterative method:

   ```
   for (char c : one) {
           if ((two.count(c) < one.count(c)) { return false; }
   } return true;
   ```

   Pure method:
   ```
   return one.stream().all(c → two.count(c) >= one.count(c));
   ```

2. Between the recursive version and the iterative version you solved in the previous question, which one would run faster in real time? Why?

   Certainly the iterative method; it does not create arrays on every call. In general, recursive methods in languages which do not support tail recursion optimization are more inefficient since they have to add function calls to the stack frame. I would guess, however, that the pure method written above would have almost the same execution time.