CS310: Advanced Data Structures and Algorithmns

Fall 2014 Programming Assignment 3

Due: Tuesday, October 24, 2014 at midnight

Goals

This assignment aims to help you:

- Learn about dynamic programming.
- Try out a recursive search.

In this assignment we begin our systematic study of algorithms and algorithm patterns.

Reading

Read Weiss Chap. 7, esp. 7.6.

Advice

Before writing the code try to run a small example on paper. Think what you would do if you were given the set of instructions or hints and had to do it without a computer. Then start programming.

Questions

1. Weiss, Exercise 7.31, except return the set of strings from permute:

   public static Set<String> permute (String str)

   Use recursion, that is, call permute on a substring of str, a string of letters, and then build up the new set based on the set returned from the recursive call. Use StringBuffer to put strings together. Usage: java cs310.Permute str prints out the permuted strings. All the letters of str are different.

2. The maximum increasing subsequence problem is similar to the contiguous subsequence problem introduced in class. It is the problem of, given a sequence, finding an increasing subsequence of maximum length contains in it. In this case the subsequence does not have to be contiguous. For example, if the input is:

   0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15

   A maximum contiguous subsequence is:

   0, 2, 6, 9, 13, 15

   Notice that the solution is not necessarily unique, so other subsequences of size 6 may exist.

   Write a **dynamic programming** program to solve the problem. It works in a similar way to the maximum contiguous subsequence problem discussed in class. That is – given an input sequence a[0]..a[n-1] each index j either extends a previous maximum length subsequence or starts a new one if a[j] is not larger than any previous element. Notice that since the sequence is not necessarily contiguous, it's not sufficient to only check a[j-1] for each a[j] (as we did in class for the maximum contiguous subsequence), but possibly all the elements left of a[j], which makes the algorithm slower. You can assume that no two elements in your input array are equal. In addition to your input array keep also an array p[] where p[j] is the predecessor of a[j] in the maximum increasing subsequence ending in a[j], and an array s[] where s[j] is the size of the maximum subsequence ending in j. These arrays will later be used to backtrack on the output and print the sequence.

3. Weiss 7.46 a. Greedy recursive search. For the class setup, use the attached Grid.java. The attached file contains a main, the grid (as appears on page 346) and the Spot class. You will have to write only the groupSize method and a helper functions. Here's one idea: set up a local Set of Spots to hold spots you've found so far in the cluster. From the current spot-position, greedily add as many direct neighbor spots as possible. For each neighbor spot that is actually newly-found, call recursively from that position. See attached MakeChange.java for an example of a recursion helper method – you need a recursion helper here to fill the set. Once the set is filled with spots in the group, the top-level method just returns its size.

   Usage: "java cs310.Grid 3 7" for example, to search from (3,7), the top occupied square of the L-shaped group. This case should print out "4".

Delivery

- Permute.java: usage "java cs310.Permute str", to print out the permutations of the letters in string str.
- DynamicSubsequence.java, usage: "java cs310. DynamicSubsequence arr", where arr is the one shown in question 2. Any of the possible maximum subsequences will be accepted.
- Grid.java, usage: "java cs310.Grid 3 7".