CS2223, Project 5
A Dynamic Programming Exercise and Bellman-Ford[*]

Due Monday, 10 Dec 2012

In this project, you will do two different exercises. One is the Bellman-Ford algorithm for finding lowest cost ("shortest") paths in a graph with weights. One can regard it as a dynamic algorithm, though maybe it's not the most typical. The other is far more typical. It's about how to find a maximum subsequence of a given sequence of numbers, subject to the constraint that you can't take any two adjacent numbers.

Maximum Non-adjacent Subsequence. You will be given an array of integers. You would like to select a subsequence of the array so as to maximize the value of the integers you select. However, the constraint is that you must not take any two adjacent numbers.

Suppose that you want to choose a subsequence of the sequence

$$\langle 17, 12, 42, 16, 18, 56, 32 \rangle.$$

Clearly, you want to take the 56 in slot 6, and the 42 in slot 3. That does mean that you have to sacrifice the 16, 18, and 32 in slots 4, 5, and 7.

Optimizing this—getting the most you can get, subject to the non-adjacency condition—is clearly a job for dynamic programming.

In max_nonadj.lua you will find some code—including the transformer recursively_memoize_oper—and some comments. The goal of this activity is to solve this problem.

You must select a subsequence of the array a to maximize the selected values, subject to the constraint that a[i] and a[i+1] are never both selected.

We write MNAS as short for "maximal non-adjacent sequence."

[*]Joshua Guttman, FL 137, `mailto:guttman@wpi.edu`. Include [cs2223] in the Subject: header of email messages. Due midnight at the end of 10 Dec.

1. First write out the recurrence that defines the optimal choice from the first i elements of the array in terms of the optimal choice on subsequences. Include this in a block comment in max_nonadj.lua near the top.

2. Next write the code for the MNAS operator. It should be modeled directly on the recurrence relation that you defined in part (1).

3. Write the code for the MNAS sequence, which uses

   ```
   local f = recursively_memoize_oper(mnas_oper(s))
   ```

   to translate from the optimal sums to determine the actual entries in a maximal sequence. You may find the section "Improving the code" in the LCS section of Chap 15 useful for this. You can also model your code on the example in dynamic_lcs.lua (lcs_max_substring).

4. Test your code on the sequences s1 through s5, and on other sequences that you invent. Include the commands and output in a block comment in the submitted file.

5. Now transform your recursive solution using recursively_memoize_oper into an iterative solution that allocates an array, inserting maxima for successive indexes directly into the array. You may use the textbook and dynamic_lcs.lua (lcs_iter) as examples. Return the array as result.

   Test your code on s1–s4 and other examples. Include the test commands and results as a block comment in your submission.

6. Use your iterative solution directly to determine the actual slots chosen in a subsequence with the maximum sum. Print them out. You may use the textbook and lcs_iter_str as guides. Test your code and include the test commands and results in comments.

Bellman-Ford. Reread *CLRS*, pp. 643–55. Using the template given in bellman_ford.lua, implement the Bellman-Ford algorithm. Construct a number of graphs using graphs.random(), and use Bellman-Ford to build a number of shortest distance tables. Also print out the paths built up for these examples.

More detailed instructions are in the starter file.