Lab 12: Dynamic Programming

CS2110 Programming Lab

November 3, 2014

Introduction

Please solve the following problems using dynamic programming.

1 Longest Common Subsequence

1.1 Background:

Evolutionary biologists scourge for patterns in protein sequences of animals to find out evolutionary patterns. One measure for evolutionary similarity is the length of the longest common subsequences between proteins from different organisms. There are many proteins and many organisms, clearly this is a herculean task. You are going to help these scientists by applying dynamic programming for the LCS problem on the protein dataset that is given.

1.2 Input

Your program must take an input file that contains a pair of strings (separated by $). The input strings will only have capital alphabets in them. The protein sequences can be of varying lengths. A sequence is split into multiple lines with each line containing at most 60 amino acids. For example, consider the protein from a bull:

MEKTELIQKAKLAEQAERYDDMATCMKAVTEQGAELSNEERNLLSVAYKNVVGGRRSAWR
VISSIEQKTDTSDKKLQLIKDYREKVESELRSICTTVLELLDKYLIANATNPESKVFYLK
MKGDYFRYLAEVACGDDRKQTIDNSQGAYQEAFDISKKEMQPTHPIRLGLALNFSVFYYE
ILNNPELACTLAKTAFDEAIAELDTLNEDSYKDSTLIMQLLRDNLTLWTSDSAGEECDAA
EGAEN

The protein has 245 amino acids split into 5 lines. The first four lines contain 60 amino acids each and the last one contains 5 amino acids. The 245 character string is the protein.

1.3 Output

Your program must print a similarity score for every pair of proteins given to you. The similarity score must be measured as the length of the longest common subsequence. Your program should also print the longest common subsequence itself. In case there are many, print any one of them.

Output Format:

For each pair, print the length of the sequence followed by the actual longest subsequence. For example:
12 MKMRCTVGPADE

For any further details, refer to the README file in the template.

2 Floyd Warshall Shortest Path

The Floyd Warshall algorithm is used for solving the All Pairs Shortest Path problem. The problem is to find shortest distances between every pair of vertices in a given edge weighted directed Graph.

The algorithm has been described in the tutorial provided. Complete details of the algorithm can be found in Chapter 25, section 2 of the book "Introduction to Algorithms" 3rd edition, by Cormen et al, the e-copy of which is available online.

2.1 Input

Your input will have a number 'N' indicating the number of nodes in the graph. This will be followed by N lines containing the weighted adjacency matrix of the graph where each entry indicates the edge weight for an edge between two nodes. Each number is TAB separated from each other in a line. An entry "INF" indicates that there is no edge between the two nodes.

Example:

4

0 5 INF 10
INF 0 3 INF
INF INF 0 1
INF INF INF 0

2.2 Output

Your output should be the final adjacency matrix containing the shortest path length between every two nodes. Each number on a line should be TAB separated.

Example:

0 5 8 9
INF 0 3 4
INF INF 0 1
INF INF INF 0

For more details, refer to the README file in the template provided.

3 Box Stacking

You are given a set of n types of rectangular 3-D boxes, where the i^{th} box has height h(i), width w(i) and depth d(i) (all integers). You need to create a stack of boxes which is as tall as possible, but you can only stack a box on top of another box if the dimensions of the 2-D base of the lower box are each STRICTLY larger than those of the 2-D base of the higher box. Of course, you can rotate a box so that any side functions as its base. It is also allowable to use multiple instances of the same type of box.

Also, note that all numbers in the input and the output will be integers. You need not care about really large inputs which go beyond the range of an integer.

3.1 Example

Input:

4
4 6 7
1 2 3
4 5 6
10 12 32

Output:

60