Due Monday,11/28/16

Dynamic Programming Homework

On paper: #4 (for the Instructor)
On paper: #1, 2, 3 (for the TA)
Extra Credit: submit your .cpp file via Canvas

(5 pts) 1. For the matrix multiplication example problem,

A_1: 30×1
A_2: 1×40
A_3: 40×10
A_4: 10×25

and the 2-D 4×4 table looks like this. Fill in the blank squares. You can do this by hand if you are careful (and it's instructive to do this for a small problem such as this one), but you should write a computer program that will compute the diagonal table results for the product of any number of matrices up to and including 10; you will need this for the next two problems anyway.

i \ j	1	2	3	4
1	0	1200	700	1400
2	X	0	400	11000
3	X	X	0	10000
4	X	X	X	0

(10 pts) 2. The algorithm given for matrix multiplication will tell the minimum work to do the multiplication, but it does not tell what the parenthetical ordering is, so you still don't know the order in which to do the multiplications. Modify your program so that it gives not only the minimum work value but also displays the matrix multiplication in parenthesized form that produces this minimum work value.

Hint: In addition to the $n \times n$ M[i,j] table, maintain a second $n \times n$ table that stores, for each i,j value entered into the M table, the value of k, $i \le k < j$, that produced the minimum value. Remember that the k-value associated with M[i,j] is the top-level split-point, i.e., it represents the last multiplication (set of parentheses) done for the product A_i through A_j. This suggests using recursion to walk back from the top-level split down through the splits in the smaller and smaller sub-products. So at the end of your program add a function/method to recursively write out the parenthesized expression for multiplying A_i through A_j, then invoke that for i = 1, j = n.

Submit output from your program (just copy the output and securely staple it or tape it on your paper) for the Example case of 4 matrices. This should show 3 things: the M[i,j] table values along the diagonals, the final minimal work required, and the parenthesized expression. You know what all the output should look like.

(10 pts) 3. Use your program to find the minimum cost to multiply the following matrices and the parenthesized expression for that multiplication.

A1: 10×17
A2: 17×12
A3: 12×25
A4: 25×14
A5: 14×30
A6: 30×15
A7: 15×9

(20 pts) 4. The Longest Increasing Subsequence problem. Given a sequence of positive integers

$X = x_1, x_2, \ldots, x_n$

you want to find the longest increasing subsequence of X. For example, in

X = 6, 2, 9, 6, 5, 8, 12, 15, 7

the longest increasing subsequence (each value is larger than the previous value) is

2, 6, 8, 12, 15

with length 5. Such a subsequence might not be unique (here 2, 5, 8, 12, 15 also works), but the length is unique.

a) Describe a completely brute-force solution to the problem for a sequence of length n. Give an approximate order of magnitude for the work involved. (explain your answer)

b) To identify the subtasks, suppose that

$x_{i1}, x_{i2}, \ldots x_{ik}$

is an increasing subsequence of X. Then

$x_{i1} < x_{i2}, < \ \ldots < x_{ik}$

To extend this increasing subsequence, x_{ik+1} must satisfy the inequality $x_{ik} < x_{ik+1}$. Hence you will need to know all the "predecessors" for any given x_j (all the values before it that are smaller) in order to tell which subsequences can be extended by adding x_j.

A subtask will be to compute L[j], the length of the longest increasing subsequence that ends with index j. This would require that x_j can be added as the last node on a previously identified longest subsequence, and the new subsequence ending in x_j will be 1 longer. Thinking of storing the results of subtasks in a table, you can use a linear table similar to the one for the Fibonacci problem, and walk through the table left to right:

	i	1	2	3	4	5	6	7	8	9
L[i]										

Any element with no predecessors must be the start of an increasing sequence and the length of that sequence is 1. This is a "base case" for filling in the table. Other than this special case, the value of the next L[j] is obtained from looking at all the L[i] for which i is a predecessor of j and computing the maximum such value. The table for the example problem above would begin with

	i	1	2	3	4	5	6	7	8	9
L[i]		1	1	2						

Both x_1 and x_2 have no predecessors but both are predecessors of x_3. Hence to compute L[3] requires knowledge of both L[1] and L[2]. Moving on,

	i	1	2	3	4	5	6	7	8	9
L[i]		1	1	2	2					

the computation of L[4] uses L[2] but not L[1] or L[3] because x_1 and x_3 are not predecessors of x_4.

Complete the table for the example problem.

c) Unlike our previous examples, when we complete the table, the last entry, L[n], is not yet the solution to our problem. Recall that L[j] equals the length of the longest subsequence ending at index j. There is no reason why the longest subsequence necessarily ends with the last entry in the sequence. What is the final step needed to solve the problem of finding the maximum length?

d) Write pseudocode for the heart of the dynamic programming algorithm to find the length of the longest increasing subsequence.

e) Let the work units for this algorithm be comparisons of X-values and comparisons of L-values. Describe the kind of sequence of length n that results in the best case. Describe the kind of sequence of length n that results in the worst case. Give a fairly precise order-of-magnitude for the work in each case. Explain your answers.

(20 pts) 5. (**Extra credit** – submit your LIS.cpp file).

a) Write a computer program to solve the Longest Increasing Subsequence problem. Your program should read input from a text file (one value per line; name the text file *incseq.txt*). The first value in the file should be the number of elements in the sequence, maximum number of elements 15. Your program should write out the input sequence and the length of the longest subsequence.

b) Add to your program so that the output includes the actual values in the longest increasing subsequence, written in increasing order.