FirstName LastName
StudentID acct@ucsc.edu

CMPS 102 — Winter 2019 – Homework 5

Two problems, 20 points, due 11:50 pm Wednesday March 13th, read the *Homework Guidelines*

1. (10 pts) Consider the problem of detecting similar papers in a history course. If the assignment was on a particular topic, then one would expect that many of the key terms related to the topic would appear in most of the essays, but perhaps not in the same order. Let the *sequence similarity* of two essays be the length of the longest sequence of words such that both essays contain the sequence of words in the same order.

   For example, if one paper was "four score and seven years ago our fathers brought forth upon this continent, a new nation conceived in liberty, and dedicated to the proposition that all men are created equal." and another paper was "four years and seven days ago, fathers helped created a new nation with liberty and justice for all on the continent."

   One sequence I found that seems to have the most words in both essays in right order is: "four and seven ago fathers a new nation and all" (10 words). Replacing "all" with "the" gives another 10 words occurring in both sequences in order. Although "continent" and "years" also occur in both papers, they cannot be added to the sequence while preserving the sequence's order of occurrence.

   For this problem you are to create a dynamic programming algorithm that takes a papers P and Q as a lists/arrays of words n and m words respectively (possibly with repeats) and determines a longest sequence of words that occur in both papers in the same order.

   (a) (4 pts) First focus on the length of the longest sequence of words occurring in both papers in order. Derive a recurrence for this length by considering what can happen with the last words in the papers. What are the boundary conditions for your recurrence? Give a brief rational why the recurrence gives the right values.

   Let $M[i,j]$ be the length of the longest sequence of words occurring in the same order in both the first i words of P and the first j words of Q. Thus $M[n,m]$ is the length of the longest sequence of words occurring in the same order in both P and Q. Let S be a longest sequence of words occurring in the same order in both P and Q, and let $|S|$ be the number of words in S, so $|S| = M[n,m]$.

   Case1: If the last word in S is the same as both $P[n]$ and $P[m]$, then $M[n,m] = |S| = 1 + M[n-1, m-1]$. This is true because:

   - $|S|-1 \leq M[n-1, m-1]$ since the first $|S|-1$ words of S appear in order in both $P[1..n-1]$ and $Q[1..m-1]$.
   - $|S|-1 \not< M[n-1]$, since otherwise some sequence of words S' with $|S'| > |S|-1$ occurs in the same order in both $P[1..n-1]$ and $Q[1..m-1]$, and S' followed by $P[n] = Q[m]$ would contradict the optimality of S.

   Case2: If $P[n]$ is not the last word in S, then S must appear in order in $P[1..n-1]$, and $M[n-1,m] \geq M[n,m]$. Since any sequence of words occurring in $P[1..n-1]$ also occurs in P every sequence of words occurring in the same order in both $P[1..n-1]$ and Q must also occur in the same order in both P and Q, so $M[n-1,m] \not> M[n,m]$, so $M[n,m] = M[n-1,m]$.

   Case 3: By the same reasoning as Case 2, if $Q[m]$ is not the last word in S, then $M[n,m] = M[n,m-1]$.

Thus, when $m, n \geq 1$:

$$M[n,m] = 1 + M[n-1, m-1] \quad \text{if } P[n] = Q[m],$$
$$M[n,m] = \max\{M[n-1,m]; M[n,m-1]\} \quad \text{otherwise.}$$

For boundary conditions we have that $M[0,j] = M[i,0] = 0$.

(b) (3 pts) Give a bottom-up dynamic programming algorithm based on your recurrence that calculates the value of the optimal solution (i.e. the length of the longest sequence of words occurring in both lists in the same order). This algorithm should fill in a table.

I will call my table M, defined as in the previous part.

```
1: for i=0..n do
2:     M[i,0] ← 0
3: end for
4: for j=0..m do
5:     M[0,j] ← 0
6: end for
7: for i=1..n do
8:     for j=1..m do
9:         if P[i] = Q[j] then
10:            M[i,j] ← 1 + M[i-1,j-1]
11:        else
12:            M[i,j] ← max{M[n-1,m]; M[n,m-1]}
13:        end if
14:    end for
15: end for
```

(c) (1 pt) What is the running time of your dynamic programming algorithm? (use asymptotic notation)

The running time $\Theta(mn)$.

(d) (2 pts) Describe how to output an actual longest sequence of words of words occurring in both lists in the same order. What is the (asymptotic) worst-case running time of your output method (assuming the table from part (b) has already been calculated).

Traceback through the array $M[]$ starting at cell n, m can use a stack as follows:
When at cell i, j:

i. if $i = 0$ or $j = 0$ return the sequence of words popped off the stack and stop
ii. else if $P[i] = Q[j]$ push word $P[i]$ onto a stack and move to cell $i-1, j-1$
iii. else if $M[i-1, j] > M[i, j-1]$ move to cell $i-1, j$
iv. else move to cell $i, j-1$.

The worst-case running time of this procedure is $O(m+n)$. It starts in cell n, m, the loop takes constant time (except for the 0,0 location), and reduces at least one of i or j by one each iteration.

2. (10 pts) Many scientific and engineering applications involve performing long sequences of matrix multiplications, and often the matrices are not square. Matrices M_1 and M_2 can be multiplied if the dimensions of M_1 are $a \times b$ and the dimensions of M_2 are $b \times c$, (so M_1 has the same number of

columns as M_2 has rows) and the product will have dimensions $a \times c$. In this problem we will consider only the standard way to multiply two matrices, so multiplying an $a \times b$ matrix with a $b \times c$ matrix takes time abc.

A sequence of matrix multiplications $M_1 M_2 \cdots M_n$ is defined if each M_i has dimensions $d_{i-1} \times d_i$, so each matrix has a number of columns equal to the next matrix's number of rows. The resulting product will be $d_0 \times d_n$.

Matrix multiplication is associative, so the sequence of matrices can be parenthesized in any way (but the order of the matrices cannot be changed). Parenthesizing in a clever way can greatly reduce the time required to multiply the matrices. For example consider the following product with $d_0 = 8$, $d_1 = 4$, $d_2 = 10$, and $d_3 = 3$:

$$\underbrace{M_1}_{8\times 4} \cdot \underbrace{M_2}_{4\times 10} \cdot \underbrace{M_3}_{10\times 3}$$

In this case, parenthesizing as $(M_1 M_2)M_3$ takes time $(8 \cdot 4 \cdot 10) + (8 \cdot 10 \cdot 3) = 560$ while parenthesizing as $M_1(M_2 M_3)$ takes time $(4 \cdot 10 \cdot 3) + (8 \cdot 4 \cdot 3) = 216$, and is more than twice as fast. The savings can easily be even more dramatic. You may want to experiment with other sequences of 3 or 4 matrices with varying dimensions. Note that the times to multiply a sequence of n matrices does not depend on the values in the matrices, only on the sequence of dimensions $d_0, d_1, \ldots, d_n$. Furthermore, as each matrix multiplication is done, one of the dimension from the sequence of dimensions is effectively cancelled. The cost of the multiplication is the product of three dimensions: the cancelled dimension and the two adjacent remaining dimensions.

For this problem you are to create a dynamic programming problem for determining the cost of the best way to parenthesize a sequence of matrix multiplications.

(a) (4 pts) By considering the last multiplication done, create a recurrence for the minimum cost to multiply a sequence of matrices with dimensions $d_0, \ldots, d_n$ in terms of the minimum costs needed to multiply subsequences of the matrices.

Let $M[i, j]$ be the cost of the best way to multiply matrices $M_i, M_{i+1}, \ldots, M_j$, so $M[1, n]$ is the cost of the best way to multiply the entire sequence.

When the parenthesization first multiplies $M_i, \ldots, M_k$ and $M_{k+1}, \ldots M_j$, the final multiplication takes $d_{i-1} d_k d_j$ time, and $M[i, j] = d_{i-1} d_k d_j + M[i, k] + M[k+1, j]$. Note that $i \leq k$ and $k \leq j$ for the sequences of matrices to be non-empty. The best parenthesization will use the best choice of k, so:

$$M[i, j] = \min_{i \leq k < j} \{d_{i-1} d_k d_j + M[i, k] + M[k+1, j]\}.$$

For the boundary conditions we have $M[i, i] = 0$ for all $1 \leq i \leq n$.

(b) (1 pt) What is "smaller" about the subproblems that must be solved?

The subproblems that must be solved multiply a sequence of fewer matrices.

(c) (4 pts) Give a bottom-up table-filling algorithm for computing the minimum cost to multiply a sequence of matrices from the dimensions $d_0, \ldots, d_n$. Make sure that the values needed to fill in a cell are computed before being used.

I will call my table M, defined as in the previous part.

```
1: for i = 1..n do
2:     M[i, i] ← 0
3: end for
4: for δ = 1..n − 1 do
5:     for i = 1..n − δ do
6:         j ← i + δ
7:         M[i, j] ← min_{i≤k<j} {d_{i−1} d_k d_j + M[i, k] + M[k + 1, j]}
8:     end for
9: end for
```

(d) (1 pt) Briefly describe how the table's values can be used to determine which products should be computed before the last multiplication (i.e. between which M_i and M_{i+1} are separated by the outermost parentheses) in a best way to parenthesize the product. (you need only describe how to determine the "last" multiplication, and need not recursively compute the entire ordering).

The value $M[1, n]$ is the best way to multiply the entire sequence. Some k value (there may be more than one) minimizes $d_0 d_k d_n + M[1, k] + M[k + 1, n]$. The last multiplication in the optimal solution associated with this minimizing k multiplies matrix $A = M_1 M_2 \cdots M_k$ with matrix $B = M_{k+1} \cdots M_n$.