Algorithms Homework 6

Liam Dillingham

November 2, 2018

1 Question 15.2-2

Give a recursive algorithm MATRIX-CHAIN-MULTIPLY(A, s, i, j) that actually performs the optimal matrix-chain multiplication, given the sequence of matrices $\langle A_1, A_2, ..., A_n \rangle$, the s table computed by MATRIX-CHAIN-ORDER, and the indices, i and j. (The initial call would be MATRIX-CHAIN-MULTIPLY(A, s, 1, n).).

The initial call MATRIX-CHAIN-MULTIPLY(A, s, 1, n) is because the initial problem is parenthesizing the initial problem: the entire list of matrices. Then, we can start working on the subproblem(s). The first thing to handle would be the base case. In matrix-chain-multiply, we parenthesize, or group, matrices per the indices. So for two indices, i and j, we multiply, from left to right, All matrices with indices $\geq i$ up to and including all matrices with indices $\leq j$.

It should be noted that $s[i, j]$ contains a value which specifies the upper limit of parenthesization. So the call to the algorithm should look like this:

MATRIX-CHAIN-MULTIPLY(A, s, i, $s[i, j]$);

Where i is initially 1, and $s[i, j]$ is $s[1, n]$, which will give us the parenthesization for the entire list of matrices. Then we will want to call MATRIX-CHAIN-MULTIPLY(A, s, i, $s[i, j]$) again on the left and right of the parenthesization "split" to see what an optimal parenthesization is on each subproblem. So we call the function again on the left and right intervals, and multiply their product like so:

MATRIX-CHAIN-MULTIPLY(A, s, i, $s[i, j]$) $*$ MATRIX-CHAIN-MULTIPLY(A, s, $s[i, j] +$ 1, j)

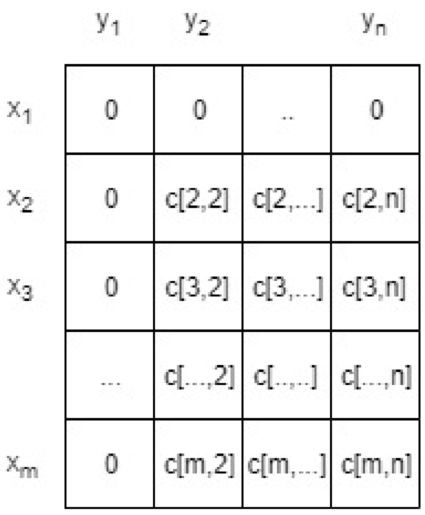
We should also be aware that the cost of multiplying a matrix by itself is 0. Or rather, we do not multiply a matrix by itself. In the case where $i == j$, or that the left bounding parenthesization equals the right, we simply return the matrix at that index. Therefore, the pseudocode looks as follows:

```
MATRIX-CHAIN-MULTIPLY(A, s, i, j)
   if i == j
      return A[i] // A is an array of matrices
   return MATRIX-CHAIN-MULTIPLY(A, s, i, s[i, j]) * MATRIX-CHAIN-MULTIPLY(A, s, s[i, j] + 1, j)
```

2 Question 15.4-2

Give pseudocode to reconstruct an LCS from the completed c table and the original sequences $X = \langle x_1, x_2, ..., x_m \rangle$ and $Y = \langle y_1, y_2, ..., y_n \rangle$ in $\mathcal{O}(m + n)$ time, without using the b table.

A completed C-table can be seen below:



	y1	y2		yn
x1	0	0	..	0
x2	0	c[2,2]	c[2,...]	c[2,n]
x3	0	c[3,2]	c[3,...]	c[3,n]
	...	c[...,2]	c[..,..]	c[...,n]
xm	0	c[m,2]	c[m,...]	c[m,n]

(a) Completed C-table

Where each cell corresponds to the current LCS at that point in each string. That is, at $c[3, 2]$, suppose the value is 2. That means that as of the 3rd character in X and the 2nd character in Y, we have a subsequence of length 2. Thus, when we are at $c[m, n]$, we have walked the entire length of both strings. While there are base cases where one strings and the other doesn't, this is the actual terminal state for the entire algorithm. Thus, when we reach $c[m, n]$, we can walk backwards towards $c[0, 0]$ to compute the reversed LCS, and then reverse that to obtain the LCS. The pseudocode for this can be shown below:

```
CONSTRUCT-LCS(x, y, c)
   output = "" // Let output be an empty string to push onto
   m = x.length // length of string x
   n = y.length // length of string y

   i = m
   j = n
   for k = m + n to 1
      max = c[i - 1, j - 1]
      if c[i - 1, j] > max
         max = c[i - 1, j]
         i = i - 1
         output.push(x[i - 1])
      elseif c[i, j - 1] > max
         max = c[i, j - 1]
         j = j - 1
         output.push(y[j - 1])
      else
         i = i - 1
         j = j - 1
         output.push(x[i - 1])

   output.reverse()
   return output
```

the loop runs $m + n$ times. and since the length of the LCS is either m or n then reversing the string either takes m or n times. Then, the runtime is either $\mathcal{O}(2m+n)$ or $\mathcal{O}(m+2n)$ which, asymptotically is $\mathcal{O}(m + n)$.