ADS Tutorial 8 Solutions

Instructor: Aris Filos-Ratsikas TA: Kat Molinet

November 25, 2024

Problem 1

Consider the Matrix Chain Multiplication problem in which we wish to compute the product $A_1 \cdot A_2 \cdot A_3 \cdot A_4$ where A_1, A_2, A_3, A_4 are rectangular matrices with dimensions $5 \times 10, 10 \times 5, 5 \times 3, 3 \times 9$ respectively. Assume that the time required to multiply two matrices of dimensions $p \times q$ and $q \times r$ is pqr.

Apply the dynamic programming algorithm MATRIX-CHAIN-ORDER to compute the optimal parenthesization on this input. Show the table M (that contains the costs of the optimal solutions to subproblems) and the table S (that contains the optimal splits) at the end of the execution of the algorithm. Your solution should include writing the recurrence relation for computing $M[i, j]$ for $i < j$.

Solution

Our first step is to set up the 4×4 dynamic programming tables M and S. Each entry $M(i, j)$ represents the minimum number of multiplications needed to compute $A_i \cdot \cdots \cdot A_j$, while each entry $S(i, j)$ in S represents the optimal "splitting" value k, allowing us to actually reconstruct the optimal parenthesization once we've filled in our dynamic programming table M.

To initialize M, our first observation is that for any $i > j$, the value $M(i, j)$ is undefined since it doesn't make sense to consider a "backwards" sequence of matrices. What about when $i = j$? Well, when $i = i$, the value $M(i, i)$ represents the number of multiplications needed to simplify the single-matrix sequence A_i. But this is already simplified, and so no multiplications are required. Thus, our diagonal entries are all initialized to zero in M.

Initializing table M:

	j:	1	2	3	4
	1	0			
i:	2	-	0		
	3	-	-	0	
	4	-	-	-	0

To fill out the remainder of our table M, we use the formula from lecture for $M(i, j)$ when $i < j$:

$$M(i, j) = \min_{i \le k < j} \left\{ M(i, k) + M(k + 1, j) + p_{i-1} \cdot p_k \cdot p_j \right\}.$$

Where does this formula come from? Informally, we can think of k as our "splitting point" in a sequence of matrix multiplications – all the matrices before the splitting point are grouped together associatively and multiplied out into a single matrix of size $p_{i-1} \times p_k$, and all the matrices after the splitting point are similarly multiplied together into a single size $p_k \times p_j$ matrix. These two groups of matrix products can then be multiplied together in $p_{i-1} \cdot p_k \cdot p_j$ multiplications. In a solution which is optimal (in the sense that it parenthesizes the sequence in a way which minimizes the number of multiplications needed), the groups before and after the splitting point will be grouped/parenthesized in an optimal way – otherwise, the overall solution could be made better. The number of mutliplications needed to optimally multiply the matrices before the split is, by definition, $M(i,k)$; and similarly, $M(k+1,j)$ for the matrices after the split. Thus, the number of multiplications needed for a split at k is $M(i,k)+M(k+1,j)+p_{i-1} \cdot p_k \cdot p_j$. Finding the splitting point k which minimizes this value gives us the general recursive formula for $M(i,j)$.

Now all we need to do is apply this formula to our specific example, noting in table S the splitting point / k-value which minimizes our value $M(i,j)$ at each entry (i,j) in M. For example, for the entries to the immediate right of the diagonal row of zeroes in M, we have:

$$M(1,2) = \min_{1\le k<2} \{M(1,k)+M(k+1,2)+p_0 \cdot p_k \cdot p_2\} = M(1,1)+M(2,2)+5\cdot 10\cdot 5 = 250$$
$$M(2,3) = \min_{2\le k<3} \{M(2,k)+M(k+1,3)+p_1 \cdot p_k \cdot p_3\} = M(2,2)+M(3,3)+10\cdot 5\cdot 3 = 150$$
$$M(3,4) = \min_{3\le k<4} \{M(3,k)+M(k+1,4)+p_2 \cdot p_k \cdot p_4\} = M(3,3)+M(4,4)+5\cdot 3\cdot 9 = 135,$$

using the fact that, according to our initialized table M, $M(i,i)=0$. This gives us the following updated tables M and S:

Updating tables M (left) and S (right):

	j:	1	2	3	4
	1	0	250		
i:	2	-	0	150	
	3	-	-	0	135
	4	-	-	-	0

	j:	1	2	3	4
	1	-	1		
i:	2	-	-	2	
	3	-	-	-	3
	4	-	-	-	-

Continuing to work away from the diagonal, we compute $M(1,3)$ and $M(2,4)$:

$$\begin{aligned} M(1,3) &= \min_{1\le k<4} \{M(1,k)+M(k+1,3)+p_0 \cdot p_k \cdot p_3\} \\ &= \min\{M(1,1)+M(2,3)+5\cdot 10\cdot 3,\ \ M(1,2)+M(3,3)+5\cdot 5\cdot 3\} \\ &= \min\{0+150+150=300,\ \ 250+0+75=325\} \\ &= 300,\ \text{with } k=1 \end{aligned}$$

And similarly, we can compute that for $(2,4)$, we have $M=585$ for $k=2$ and $M=420$ for $k=3$. Thus, we conclude that splitting on $k=3$ to get $(A_2A_3)A_4$ in 420 multiplications is the best way of grouping the subsequence $A_2A_3A_4$ (as opposed to the other option $A_2(A_3A_4)$, when $k=2$).

Finally, we can compute the final cell, $(1,4)$, which tells us the minimal number of multiplications needed to multiply out the entire sequence of matrices $A_1A_2A_3A_4$. Once again applying our recursive formula for $M(i,j)$, we get that $M=870$ when $k=1$, 610 when $k=2$, and 435 when $k=3$. Since 435 is the best, we record in in our table M, and update table S to note that we split on $k=3$. At the end of the process, we get the following dynamic programming tables M and S:

Updating tables M (left) and S (right):

	j:	1	2	3	4
	1	0	250	300	435
i:	2	-	0	150	420
	3	-	-	0	135
	4	-	-	-	0

	j:	1	2	3	4
	1	-	1	1	3
i:	2	-	-	2	3
	3	-	-	-	3
	4	-	-	-	-

So we know that we can compute the matrix product $A_1A_2A_3A_4$ using 435 multiplications. But how should we parenthisize our sequence to accomplish this? This is where the table S comes in. Looking at entry $(1,4)$ of S, we see that we split our sequence $A_1A_2A_3A_4$ at $k = 3$; i.e., in the optimal solution, we will have $(A_1A_2A_3)A_4$. But in what order should we multiply $A_1A_2A_3$? Once again, we turn to the table S to find that for $(1,3)$, the best splitting place is at $k = 1$; i.e., we should multiply in the following order: $A_1(A_2A_2)$. Putting everything together, we have $(A_1(A_2A_3))A_4$ as our optimal parenthesization of our matrices, requiring 435 multiplications.

Problem 2

A contiguous subsequence of length k a sequence S is a subsequence which consists of k consecutive elements of S. For instance, if S is $1, 2, 3, -11, 10, 6, -10, 11, -5$, then $3, -11, 10$ is a contiguous subsequence of S of length 3. Give an algorithm based on dynamic programming that, given a sequence S of n numbers as input, runs in linear time and outputs the contiguous subsequence of S of maximum sum. Assume that a subsequence of length 0 has sum 0. For the example above, the answer of the algorithm would be $10, 6.-10, 11$ with a sum of 17.

Solution

Let $a_1a_2 \dots a_n$ be the sequence S. We will use dynamic programming to design an algorithm that solves the contiguous subsequence problem. Let $M(j)$ be the optimal solution (the length of the subsequence of maximum sum) *ending at position* j. By definition, we have that $M(0) = 0$. We have the following relation:

$$M[j+1] = \max\{M[j] + a_{j+1}, 0\},$$

with $M[1] = \max\{a_1, 0\}$. To find the contiguous subsequence S^* of maximum sum, we operate as follows. First, we find the element $i*$ for which $M[i^*]$ is maximised. This can be done in polynomial time, by computing the partial sums and storing them in an array (similarly to the approach in the weighted interval scheduling problem). S^* will end at i^*. The beginning of S^* will be the largest $j \leq i^*$ for which $M[j-1] = 0$, as extending the subsequence to start before j will only decrease the sum. If there is no such j, then S^* starts at the beginning of S.