CS 3510 Algorithms 10/12/2023

Lecture 13: Chain Matrix Multiplication

Lecturer: Abrahim Ladha *Scribe(s): Tejas Pradeep*

1 Problem Statement

Suppose we are given a sequence of matrix dimensions, compute the product of the matrices. We are not given the matricies, but the dimensions. We want to determine which order to multiply the matricies in to minimize the number of operations. So say you have the following matrices, $A = (50 \times 20)$, $B = (20 \times 1)$, $C = (1 \times 10)$, $D = (10 \times 100)$. Suppose we want to compute the product $ABCD$. Note that we can do this for rectangular matricies since the dimension of the rows of A is the dimension of the column of B and so on. Matrix multiplication is associative, but not communitive. It is true that $(AB)C = A(BC)$, but not true that $AB = BA$.

Given two matricies of dimensions $(d_0 \times d_1)$ and $(d_1 \times d_2)$, we may ballpark the cost of multiplying these two matricies together as $d_0 d_1 d_2$ fixed point operations. We are not concerned with actually multiplying the matricies together, just computing which order to best multiply them according to this cost function. In the previous example, consider the following parantheticalizations

- $(A((BC)D)) = 20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100 = 120200$
- $((A(BC))D) = 20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100 = 60200$
- $((AB)(CD)) = 50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100 = 7000$

This difference allows us to try and optimize. The number of possible associations grows exponentially, making the search space non trivial. The number of parantheticalizations follows the Catalan numbers, which is just negligiblly less than $O(4^n)$. If we greedily split through the one, we get the smallest cost set of associations, but the greedy approach doesn't actually yield the best associations in other series. We will approach the problem with dynamic programming.

2 Algorithm

We can trivially create trees for these associations where the children are the left and right terms and the parent is the expression.

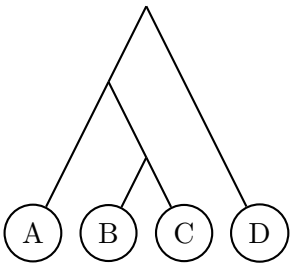

Figure 1: A tree representing an association for the matrices; each sub tree represents a multiplication.

We can not however choose one binary tree from an algorithm; instead we need to consider all binary trees. If you've noticed though, each binary tree is made of other binary trees, meaning if we could store these in our dynamic programming structure, we can compute a solution by combining its minimal subproblems.

We'll define a dynamic programming table of dimensions $T[1..n][1..n]$ where

$$T[i,j] = \text{minimum cost to multiply } A_i, A_{i+1}, ..., A_j$$

Then to find the minimum cost over the whole array, we index into the table at $T[1,n]$. Our base case also comes from this equation, $\forall i T[i,i] = 0$ because we're doing no multiplications in that case.

At each level we're going to have some partition of the overall series, say $A_i...A_j$. We're going to have to find some middle point to multiply two sub matrices at, say $(A_i...A_k)(A_{k+1}...A_j)$. The cost for the overall level is the cost for the smaller two multiplications $A_i...A_k$ and $A_{k+1}...A_j$ plus the cost for the current multiplication.

With every dynamic programming algorithm, consider all possibilities, then take the min or the max or the sum over those. For some given subproblem? What is our last operation?

$$\begin{aligned}
&(A_1)(A_2A_3A_4...A_n)\\
&(A_1A_2)(A_3A_4...A_n)\\
&(A_1A_2A_3)(A_4...A_n)\\
&(A_1A_2A_3A_4)(...A_n)\\
&...\\
&(A_1A_2A_3A_4...)(A_n)
\end{aligned}$$

The very last step is where we split the sequence of matricies into two products to be computed separately and then combined. Of course we're looking for the minimum here, so we need to take the minimum k over all the costs, we do this by making the table

$$T[i,j] = \min_{i \le k \le j} T[i,k] + T[k+1,j] + d_{i-1}d_kd_j$$

The k for which this is minimum corresponds to the costs to compute $(A_i...A_k), (A_{k+1}, ..., A_j)$, plus the cost to multiply those together.

3 Pseudocode

We're going to end up ignoring half the table because the recurrence only fills in the top half since $i \leq j$. This effects which cells in the table make sense, but also where we'll find our solution.

```
def chainmatrix(d_0, d_1, ..., d_n):
    intialize dp table of size n, n to all zeroes 0
    for i in 1..n:
        dp[i,i] = 0

    for s in 1..n
        for i in 1..n-s
            j = i + s
            dp[i, j] = min_{i <= k < j} {dp[i, k] + dp[k + 1,
  j] + d_{i - 1}d_kd_j}
```

Figure 2: Algorithm to find the optimal way to multiply a sequence of matrices.

We have weird loops because we need to fill in the table in a specific way.
s is the size of the series at some level and i is the starting index of the partition.
Runtime: In the algorithm, we fill in $n^2/2$ cells, each taking n time, so $O(n^3)$ time. This may seem slow but is much better than the brute force approach, which takes $O(4^n)$ time.

4 Example

Our original set of matrices had sizes $[50, 20, 1, 10, 100]$. We can fill in our base case below where the indices are equal to each other.

0	-	-	-
-	0	-	-
-	-	0	-
-	-	-	0

We'll do $T[1,2] = m_0m_1m_2 = 1000$, $T[2,3] = m_1m_2m_3 = 200$, $T[3,4] = m_2m_3m_4 = 1000$. This makes our table the following

0	1000	-	-
-	0	200	-
-	-	0	1000
-	-	-	0

13: Chain Matrix Multiplication-3

Now we can do $T[1,3] = \min(T[1,1] + T[2,3] + m_0m_1m_3 = 1500, T[1,2] + T[3,3] + m_0m_2m_3 = 1500) = 1500$, $T[2,4] = \min(T[2,2] + T[3,4] + m_1m_2m_4 = 3000, T[2,3] + T[4,4] + m_1m_3m_4 = 3000) = 20200) = 3000$.

0	1000	1500	-
-	0	200	3000
-	-	0	1000
-	-	-	0

Now for our final recurrence we'll be doing the whole array.

$$dp[1,4] = \min \left\{ \begin{array}{ll} dp[1,1] + dp[2,4] + m_0m_1m_4, & \text{if } k = 1 \\ dp[1,2] + dp[3,4] + m_0m_2m_4, & \text{if } k = 2 \\ dp[1,3] + dp[4,4] + m_0m_3m_4, & \text{if } k = 3 \end{array} \right\}$$

$$dp[1,4] = \min \left\{ \begin{array}{ll} 0 + 3000 + 100000, & \text{if } k = 1 \\ 1000 + 1000 + 5000, & \text{if } k = 2 \\ 1500 + 0 + 50000, & \text{if } k = 3 \end{array} \right\} = 7000$$

Finally this would make our table the following, and we can get the cost for the whole series by looking at the cell in the top right.

0	1000	1500	7000
-	0	200	3000
-	-	0	1000
-	-	-	0

We haven't stored enough information to actually find the associations though; instead we only have the cost. If you wanted this information, you would store the indices where you split (the ks), and reconstruct the answer using those.