

## Formula for Matrix Multiplication

Given a sequence  $(A_1, \dots, A_n)$  of matrices, the product  $A_1 \cdots A_n$  can be calculated provided that for each  $i$ , the matrix  $A_i$  has shape  $p_{i-1} \times p_i$  for a sequence of positive integers  $(p_0, \dots, p_n)$ .

To calculate the product  $A \cdot B$ , where  $A$  is of shape  $p \times q$  and  $B$  is of shape  $q \times r$ , one uses  $p \cdot q \cdot r$  entry-wise multiplications.

The product  $A_1 \cdots A_n$  is calculated by decomposing this into sub-problems of the form  $A(1, k) = A_1 \cdots A_k$  and  $A(k+1, n) = A_{k+1} \cdots A_n$  and then obtaining the answer as  $A(1, k) \cdot A(k+1, n)$ .

More generally, for  $i \leq j$ , let  $m(i, j)$  denote the cost in terms of entry-wise multiplications of calculating the product  $A(i, j) = A_i \cdots A_j$ . When  $i = j$ , no calculation is required, so  $m(i, i) = 0$ . When  $i < j$ , one way to complete this task is to choose  $k$  in the range  $[i, j-1]$ , calculating  $A(i, k)$  and  $A(k+1, j)$  and then calculating the product  $A(i, k) \cdot A(k+1, j)$ . Thus, we obtain the inequality

$$m(i, j) \leq \mu(i, k, j) = m(i, k) + m(k+1, j) + p_{i-1} \cdot p_k \cdot p_j$$

In fact, if the only operation that we can use to calculate the product  $A(i, j)$  is pairwise matrix multiplication, then

$$m(i, j) = \min_{i \leq k < j} \mu(i, k, j)$$

Let us define  $k(i, j)$  to be the largest  $k$  in  $[i, j-1]$  such that it achieves this minimum.

$$k(i, j) = \max\{k : k \in [i, j-1] \text{ and } \mu(i, k, j) = m(i, j)\}$$

It is reasonably clear that if we know  $k(i, j)$  for all  $i, j$  in  $[1, n]$ , then we can make a “formula” for doing this multiplication at cost  $m(1, n)$ . We first start by cutting the expression  $A_1 \cdots A_n$  at  $p_0 = r(1, n)$ , then cutting the expression  $A_1 \cdots A_{p_0}$  at  $p_1 = k(1, p_0)$  and the expression  $A_{p_0+1} \cdots A_n$  at  $p_2 = k(p_0+1, n)$ , and so on.

Secondly, it is clear that in order to make such a formula, the *entries* of  $A_i$  are not utilised. We only need the sequence  $(p_0, \dots, p_n)$  of positive integers.

Thus, we want an algorithm to calculate  $m(i, j)$  and  $k(i, j)$  given the sequence  $(p_0, \dots, p_n)$ . This will use a dynamic programming approach since the same  $m(i, j)$  and  $k(i, j)$  appear in the calculation for  $m(p, q)$  and  $k(p, q)$  for many  $p$  and  $q$ . In other words, the sub-problems *overlap*.

In the following Python code the variable `soln` is a mapping which takes pairs  $(i, j)$  to the pair  $(m(i, j), f(i, j))$  where  $f(i, j)$  is the string representing the optimal formula for calculating  $A(i, j) = A_i \cdots A_j$ . The code builds up this mapping “from the bottom up” as is often the case in dynamic programming.

**Exercise:** Try to understand, from the point of view of dynamic programming, the meaning of the variable `r` in the following program, and why the outermost `for` loop takes increasing successive values of `r`.

```
def matmul(p):
    n = len(p)-1
    soln = {}
    for i in range(1,n+1):
        soln[(i,i)] = (0,"A["+str(i)+"]")
    for r in range(1,n):
        for i in range(1,n-r+1):
            oval = float("inf")
            j = i+r
            for k in range(i,j):
                left = soln[(i,k)]
                right = soln[(k+1,j)]
                val = left[0]+right[0]+p[j-1]*p[i]*p[j+k]
                if val < oval:
                    oval = val
                    oterm = "("+left[1]+"*"+right[1]+")"
            soln[(i,j)]=(oval,oterm)
    ans = soln[(1,n)]
    print("An optimal calculation makes", ans[0], \
          " multiplications using the expression:")
    print(ans[1])
```

Given the input  $p = (33, 35, 23, 20, 34, 31)$ , this program says that an optimal expression that makes 80740 multiplications uses the formula

$$((A_1 \cdot (A_2 \cdot A_3)) \cdot (A_4 \cdot A_5))$$

Of course, the above program is easily modified to also give the values of  $k(i, j)$  for  $1 \leq i \leq j \leq n$ .