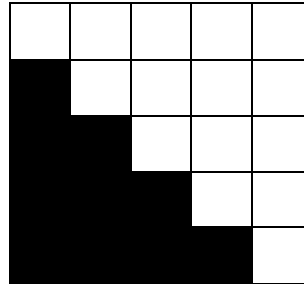


## The Algorithm

For our dynamic programming algorithm to solve this problem, we will need two  $m \times m$  2-dimensional arrays, of which we will only use one half.



One of them is called  $M$ , which stores the minimum runtimes; the other is called  $S$ , which stores the multiplication orders.

Each matrix sequence multiplication can be broken down into a multiplication of two sequences. For example,

$$A_1 ((A_2 A_3) A_4)$$

can be broken down into the multiplication of the length one sequence  $A_1$  and the length 3 sequence  $((A_2 A_3) A_4)$ .  $((A_2 A_3) A_4)$  can in turn be turned into the multiplication of  $(A_2 A_3)$  and  $A_4$ .  $(A_2 A_3)$  at last will be the multiplication of two length one sequences  $A_2$  and  $A_3$ .

We formulate our problem of multiplying a sequence of matrices starting at matrix  $i$  and ending at matrix  $j$  as

$$\text{Opt}(i, j)$$

.  $\text{Opt}(3, 6)$  means the optimal runtime and corresponding order of multiplying the sequence from matrix 3 to matrix 6. We want  $\text{Opt}(1, m)$  for this problem.

The rows of our two 2-dimensional arrays specify the starting matrix, the column specifies the ending matrix (e.g.  $M[3, 6]$  is the least runtime possible for multiplying the sequence starting at matrix 3 and ending at matrix 6;  $S[3, 6]$  is the corresponding optimal order of multiplication). We want  $M[1, m]$  and  $S[1, m]$  for this problem.

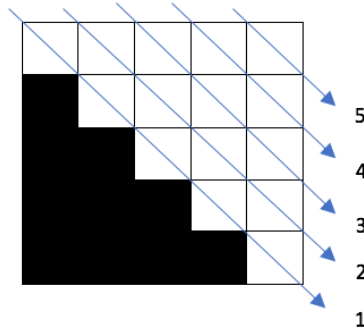
## Pseudocode

```
1  Opt-Matrix-Multiplication:
2      M[][] = new matrix[m][m]
3      S[][] = new matrix[m][m]
4
5      int diff = 0
6      for i = m, m-1, ... 1
7          for j = 1, 2, ... i
8              starting_matrix = j
9              ending_matrix = j+diff
10             if (diff == 0) # on the diagonal of the matrices
11                 M[starting_matrix][ending_matrix] = 0
12                 S[starting_matrix][ending_matrix] = null
13             else
14                 middle_matrix = a number between starting_matrix (inclusive)
15                                 and ending_matrix (exclusive)
16                 such that
17
18                 new_opt = (M[starting_matrix][middle_matrix] +
19                             M[middle_matrix + 1][ending_matrix] +
20                             (number of rows of starting_matrix *
21                             number of columns of middle_matrix *
22                             number of columns of ending_matrix))
23
24                 is minimized.
25
26                 M[starting_matrix][ending_matrix] = new_opt
27                 S[starting_matrix][ending_matrix] = join(S[starting_matrix][middle_matrix],
28                                                         S[middle_matrix+1][ending_matrix])
29                                                         .add(middle_matrix)
30             endif
31         endfor
32         diff++
33     endfor
34     minimum_runtime = M[1][m]
35     order = S[1][m]
--
```

This part

```
for i = m, m-1, ... 1
    for j = 1, 2, ... i
        .....
```

graphically represented is this:



## Proof of Correctness

Observe that the optimal multiplication ordering of a sequence of matrices of length  $> 1$  is broken into two sequences at an optimal spot. Call the last matrix of the first sequence the middle matrix. A sequence of length three can be broken down into

$$A_1 (A_2 A_3)$$

or

$$(A_1 A_2) A_3$$

.

And  $(A_1 A_2)$  is broken down into the multiplication of single matrices  $A_1$  and  $A_2$ . Opt(a single matrix  $A_1$ ) =  $A_1$ , which is the base case. Each of the two optimally broken sequences can in turn be broken into their own optimal two sequences. Done recursively, this looks like a tree. The runtime of multiplying the two sequences is the sum of the runtimes from each sequence plus the runtime of multiplying these two sequences, which is the product of the number of rows of the starting matrix and the number of columns of the middle matrix and the number of columns of the ending matrix. The recursion can be summarized as

$$\text{Opt}(\text{matrix}_{\text{start}}, \text{matrix}_{\text{end}}) = \begin{cases} 0, & \text{if } \text{matrix}_{\text{start}} = \text{matrix}_{\text{end}} \\ \min_{\text{matrix}_{\text{start}} \leq \text{matrix}_{\text{middle}} < \text{matrix}_{\text{end}}} \left( \begin{aligned} &\text{Opt}(\text{matrix}_{\text{start}}, \text{matrix}_{\text{middle}}) + \\ &\text{Opt}(\text{matrix}_{\text{middle}} + 1, \text{matrix}_{\text{end}}) + \\ &\text{number of rows of } \text{matrix}_{\text{start}} * \\ &\text{number of columns of } \text{matrix}_{\text{middle}} * \\ &\text{number of columns of } \text{matrix}_{\text{end}} \end{aligned} \right) & \text{otherwise} \end{cases}$$

And the order of multiplications is the combined orders of the two sequences. We could store the index of the matrix that breaks up each sequence for each subproblem. So

$$A_1 (A_2 A_3)$$

Would have the break order as

$$[1,2]$$

Because you break after  $A_1$  at first, making the two sequences  $A_1$  and  $(A_2 A_3)$ . Then you break after  $A_2$ , making the right sequence  $((A_1) (A_2))$ .

This is formulated as

$$\text{Opt} - \text{Order}(\text{matrix}_{\text{start}}, \text{matrix}_{\text{end}}) = \begin{cases} \text{null,} & \text{if } \text{matrix}_{\text{end}} = \text{matrix}_{\text{end}} \\ \text{join}(\text{Opt} - \text{Order}(\text{matrix}_{\text{start}}, \text{matrix}_{\text{middle}}), \\ \text{Opt} - \text{Order}(\text{matrix}_{\text{middle}}, \text{matrix}_{\text{end}})) \\ \text{.add - to - the - front}(\text{matrix}_{\text{middle}}) \\ \text{otherwise} \\ \text{(using the optimal } \text{matrix}_{\text{middle}} \text{ as shown} \\ \text{above in Opt)} \end{cases}$$

We still now prove by strong induction that the above returns the optimal runtime.

### Base Case

We want to prove  $M[1][1]$  is the optimal runtime and  $S[1][1]$  describes the optimal ordering if there were only one matrix. The optimal runtime for one matrix should be 0 and the optimal ordering should be null because there is no multiplication going on. This is all true by definition.

### Inductive Case

We need to prove  $M[i][j] = \text{Opt}(i, j)$  and  $S[i][j]$  describes the corresponding multiplication order. Since we're using strong induction, we can assume for all  $a < i$  and for all  $b < j$ ,  $M[a][b]$  is the optimal runtime for the sequence starting at matrix  $a$  and ending at matrix  $b$ , and  $S[a][b]$  describes the optimal ordering of said sequence. Thus,

$$\begin{aligned} M[i][j] = & M[i][\text{middle\_matrix}] + \\ & M[\text{middle\_matrix} + 1][j] + \\ & (\text{number of rows of } i * \\ & \text{number of columns of middle\_matrix} * \\ & \text{number of columns of } j) \end{aligned}$$

*(with optimal middle\_matrix)*

$$\begin{aligned} &= \text{Opt}(i, \text{middle\_matrix}) + \\ &\quad \text{Opt}(\text{middle\_matrix} + 1, j) + \\ &\quad (\text{number of rows of } i * \\ &\quad \text{number of columns of middle\_matrix} * \\ &\quad \text{number of columns of } j) \end{aligned}$$

Which was our recursive definition given above.

S keeps track of the order by recording the optimal breaks in each sequence. When you have a sequence from matrix  $i$  to matrix  $j$  broken up by  $\text{middle\_matrix}$ , the order of the optimal orderings of the left and right sequences stay the same, so you combine their breaks. The break that happened at the current level also needs to be accounted for, so it needs to be added to the front because it is the most recent break.

## Runtime Analysis

Since we update each of the upper half of our 2-dimensional array once, that would be  $O(n^2)$ . Getting the optimal  $\text{middle\_matrix}$  that breaks the sequence for each of those cells is  $O(n)$  because you could do a linear search for a  $\text{middle\_matrix}$  that minimizes the recurrence. Altogether, it is  **$O(n^3)$** .