1. Find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is \((5, 10, 12, 5, 50, 6)\).

**Answer:**

Basically this question is to show how to iterate the dynamic programming Matrix-chain algorithm given in lecture 9. We have 5 matrices \(A_1, \ldots, A_5\), hence we need a 5-by-5 table/array which we call \(m\). Our first step is to set \(m[i, i] = 0\) for every \(1 \leq i \leq 5\) (also we black out the bottom left-hand half of the array, since cells in that part of the array represent sequences \(A_i \ldots A_j\) for \(i > j\), which doesn’t make sense).

In this solution, I don’t actually draw out the \(s\) matrix. The entries of the \(s\) matrix only matter for sequences of \(\geq 3\) matrices (as there is only one possible parenthesisation for sequences of length 1 or 2). However, I do mention the values given to \(s\) in the description below for the cases of \(\ell = 3\) (\(A_1 A_2 A_3, A_2 A_3 A_4\) and \(A_3 A_4 A_5\)), of \(\ell = 4\) (\(A_1 \ldots A_4\) and \(A_2 \ldots A_5\)) and \(\ell = 5\).

Initialising the main matrix \(m\), we get:

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 0 &  \\
2 & - & 0 &  \\
3 & - & - & 0 &  \\
4 & - & - & - & 0 &  \\
5 & - & - & - & - & 0 \\
\end{array}
\]

Now consider all “sequence windows” of length 2 (\(\ell = 2\) in terms of line 4 of \textsc{MatrixChainOrder}). In this case there is only ever one possible split (taking one matrix on each side), hence there is no choice to be made - eg, for cell [1, 2], we have \(m[1, 2] = 5 \times 10 \times 12 = 600\).

Doing the same operation for \(m[2, 3], m[3, 4], m[4, 5]\), we get:

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 0 & 600 &  \\
2 & - & 0 & 600 &  \\
3 & - & - & 0 & 3000 &  \\
4 & - & - & - & 0 & 1500 &  \\
5 & - & - & - & - & 0 &  \\
\end{array}
\]

Next we consider windows of length 3 (\(\ell = 3\) in the Algorithm).
We must fill-in \(m[1, 3], m[2, 4], m[3, 5]\). I’ll do \(m[1, 3]\) in full:
For \( m[1,3] \), we can choose \( k = 1 \) or \( k = 2 \) (\( k \leftarrow i \) to \( j - 1 \), line 8. of algorithm \textsc{MatrixChainOrder}). If we take \( k = 1 \), our cost is

\[
m[1,1] + m[2,3] + p_0p_1p_3 = 0 + 600 + 5 \times 10 \times 5 = 850.
\]

If we take \( k = 2 \), our cost is

\[
m[1,2] + m[3,3] + p_0p_2p_3 = 600 + 0 + 5 \times 12 \times 5 = 900.
\]

Hence we set \( m[1,3] = 850 \), \( s[1,3] = 1 \) (remember \( s[i,j] \) stores the top-level split for the optimum parenthesization). After doing \( m[2,4], m[3,5] \) similarly, we get the new table:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>600</td>
<td>850</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>0</td>
<td>600</td>
<td>3100</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>3000</td>
<td>1860</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>1500</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

We also have \( s[2,4] = 3 \) and \( s[3,5] = 3 \).

Next we do windows of length 4 - there are just two, \([1,4]\) and \([2,5]\). Doing those (I'm not giving details), we get

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>600</td>
<td>850</td>
<td>2100</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>0</td>
<td>600</td>
<td>3100</td>
<td>2400</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>3000</td>
<td>1860</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>1500</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

We also have \( s[1,4] = 3 \) and \( s[2,5] = 3 \).

Finally we must calculate \( m[1,5] \). There are 4 possibilities for top-level parentheses, namely \( k = 1,2,3,4 \). We have

\[
\begin{align*}
m[1,1] + m[2,5] + p_0p_1p_5 & = 0 + 2400 + 5 \times 10 \times 6 = 2700 \\
m[1,2] + m[3,5] + p_0p_2p_5 & = 600 + 1860 + 5 \times 12 \times 6 = 2820 \\
m[1,3] + m[4,5] + p_0p_3p_5 & = 850 + 1500 + 5 \times 5 \times 6 = 2500 \\
m[1,4] + m[5,5] + p_0p_4p_5 & = 2100 + 0 + 5 \times 50 \times 6 = 3600
\end{align*}
\]

Hence we have \( m[1,5] = 2500 \) and \( s[1,5] = 3 \).

You might want to trace back the \( s \) values to find the parenthesization.

2
2. Consider the following typesetting problem. The input is a sequence of $n$ words containing $l_1, l_2, \ldots, l_n$ characters, respectively. Each line can hold at most $P$ characters, the text is left-aligned, and words cannot be split between lines. If a line contains words from $i$ to $j$ (inclusive) then the number of spaces at the end of the line is $s = P - \sum_{k=i}^{j} l_k - (j - i)$ (because the words are separated by white spaces). We aim to typeset the text so as to avoid large white spaces at the end of lines, i.e., we would like to minimize the sum over all lines of the square of the number of white spaces at the end of the line.

(a) Give an efficient algorithm for this problem.

**Answer:** We use dynamic programming. Let $A[j]$ be the optimal cost (minimal sum of squares of white spaces at the end of the line over all lines) one can achieve by typesetting only the words $1, \ldots, j$ and ignoring the remaining words. Then

$$A[j] = \min_{i<j: T[j] - T[i] - (j - (i + 1)) \leq P} \left( A[i] + (P - (T[j] - T[i]) - (j - (i + 1)))^2 \right)$$

where $T[j] = \sum_{i=1}^{j} l_i$.

I.e., we first optimally typeset words up to $i$ and then place the remaining words from $(i + 1)$ to $j$ in the last line. (There are $(j - (i + 1))$ white spaces in the last line that are not at the end of the line.)

We can compute a table of the values $T[1 \ldots n]$, and use dynamic programming to compute each value $A[j]$ in sequence. (Return fail if any word longer than $P$.)

At the end $A[n]$ contains the value of the optimal solution. We can reconstruct the optimal solution itself by maintaining backpointers (as usual in dynamic programming) to record the optimal splitting of the words between lines. One can also use an auxiliary data structure to record that.

(b) Formally prove the running time of the algorithm (matching upper and lower bounds).

**Answer:** We can compute a table of the values $T[1 \ldots n]$ in $O(n)$ steps. Each of the $n$ steps takes $O(n)$ time to compute (trying different values of $i < j$), and for $j \geq n/2$ also $\Omega(n)$, so we get $\Theta(n^2)$.

**Note:** It is also possible to get a bound in terms of $P$, i.e., $O(n \times \min(n, P))$, by using the fact that a line can hold at most $\lfloor P/2 \rfloor$ words (plus the separating spaces). However, generally, if one interprets $n$ as the full size of the problem instance and encodes numbers in binary, then one could have $P = \Omega(2^n)$. The other hand, if one explicitly considers $P$ as a fixed constant, then the above would even yield $O(n \times \min(n, P)) = O(n)$. 

---

3