# Tutorial (week 13)

All the answers can be given with pseudocode or with Python. Remember that most of the time, many solutions are possible.

1. What is the best way to multiply a chain of matrices with dimensions that are $10 \times 5$, $5 \times 2$, $2 \times 20$, $20 \times 12$ and $12 \times 4$ ? Show your work.

   **Solution:**
   We apply the MatrixChain algorithm with input sequence $d_0 = 10$, $d_1 = 5$, $d_2 = 2$,$d_3 = 20$, $d_4 = 12$, $d_5 = 4$. First, we initialise all the $N_{i,i}$ value to 0 for $i \in [0, 4]$, that is:

   $$N_{0,0} = N_{1,1} = N_{2,2} = N_{3,3} = N_{4,4} = 0.$$

   Then, we will compute the following values:

   $$\begin{aligned} N_{0,1} &= N_{0,0} + N_{1,1} + d_0 d_1 d_2 = 100 \\ N_{1,2} &= N_{1,1} + N_{2,2} + d_1 d_2 d_3 = 200 \\ N_{2,3} &= N_{2,2} + N_{3,3} + d_2 d_3 d_4 = 480 \\ N_{3,4} &= N_{3,3} + N_{4,4} + d_3 d_4 d_5 = 960 \end{aligned}$$

   $$\begin{aligned} N_{0,2} &= \min\{N_{0,0} + N_{1,2} + d_0 d_1 d_3, N_{0,1} + N_{2,2} + d_0 d_2 d_3\} = 500 \\ N_{1,3} &= \min\{N_{1,1} + N_{2,3} + d_1 d_2 d_4, N_{1,2} + N_{3,3} + d_1 d_3 d_4\} = 600 \\ N_{2,4} &= \min\{N_{2,2} + N_{3,4} + d_2 d_3 d_5, N_{2,3} + N_{4,4} + d_2 d_4 d_5\} = 576 \end{aligned}$$

   $$\begin{aligned} N_{0,3} = \quad & \min\{N_{0,0} + N_{1,3} + d_0 d_1 d_4, N_{0,1} + N_{2,3} + d_0 d_2 d_4, N_{0,2} + N_{3,3} + d_0 d_3 d_4\} = 820 \\ N_{1,4} = \quad & \min\{N_{1,1} + N_{2,4} + d_1 d_2 d_5, N_{1,2} + N_{3,4} + d_1 d_3 d_5, N_{1,3} + N_{4,4} + d_1 d_4 d_5\} = 616 \end{aligned}$$

   $$N_{0,4} = \quad \min\{N_{0,0} + N_{1,4} + d_0 d_1 d_5, N_{0,1} + N_{2,4} + d_0 d_2 d_5, N_{0,2} + N_{3,4} + d_0 d_3 d_5, N_{0,3} + N_{4,4} + d_0 d_4 d_5\} = 756$$

2. Design an efficient algorithm for the matrix chain multiplication problem that outputs a fully parenthesized expression for how to multiply the matrices in the chain using the minimum number of operations.

   **Solution:**
   We perform the same algorithm as the MatrixChain algorithm, but we store more information than just the $N_{i,j}$ values: we also store which of the $k$ value was used to compute the minimum.

In the case of the previous exercise example, lets call $A_0$, $A_1$, $A_2$, $A_3$, $A_4$ the five input matrices. One can check that when computing $N_{0,4}$, the minimum was obtained when $k = 1$ using $N_{0,1}$ and $N_{2,4}$ (and thus by multiplying $(A_0 \times A_1)$ with $(A_2 \times A_3 \times A_4)$). One can see that no other parenthesis can be added in $(A_0 \times A_1)$, thus only remains to check $(A_2 \times A_3 \times A_4)$ via the term $N_{2,4}$. One can check that when computing $N_{2,4}$, the minimum was obtained when $k = 3$ using $N_{2,3}$ and $N_{4,4}$ (and thus by multiplying $(A_2 \times A_3)$ with $A_4$). All in all, one should use the following parenthesis to minimize the amount of multiplications: $(A_0 \times A_1) \times ((A_2 \times A_3) \times A_4)$.

3. Suppose we define a probabilistic event so that $V(i, 0) = 1$ and $V(0, j) = 0$, for all $i$ and $j$, and $V(i, j)$, for $i, j \geq 1$, is defined as $V(i, j) = 0.5V(i-1, j) + 0.5V(i, j-1)$. What is the probability of the event, $V(2, 2)$ ?

   **Solution:**

$$\begin{aligned} V(2,2) &= 0.5V(1,2) + 0.5V(2,1) \\ &= 0.25V(0,2) + 0.25V(1,1) + 0.25V(1,1) + 0.25V(2,0) \\ &= 0 + 0.5V(1,1) + 0.25 \\ &= 0.25V(0,1) + 0.25V(1,0) + 0.25 \\ &= 0.5 \end{aligned}$$

4. Show the longest common subsequence table, $L$, for the following two strings:

$$X = \texttt{"skullandbones"}$$
$$Y = \texttt{"lullabybabies"}.$$

   What is a longest common subsequence between these strings?

   **Solution:**
   Strings $X$ and $Y$ have 13 characters each. We can create an empty $L$ table and first set that $L[-1, i] = L[j, -1] = 0$ (i.e. first row and column are all zeros). Then, one can fill the table row per row.

| | | | $s$ | $k$ | $u$ | $l$ | $l$ | $a$ | $n$ | $d$ | $b$ | $o$ | $n$ | $e$ | $s$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $L$ | $-1$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| | $-1$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $l$ | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $u$ | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $l$ | 2 | 0 | 0 | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| $l$ | 3 | 0 | 0 | 0 | 1 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| $a$ | 4 | 0 | 0 | 0 | 1 | 2 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| $b$ | 5 | 0 | 0 | 0 | 1 | 2 | 3 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 |
| $y$ | 6 | 0 | 0 | 0 | 1 | 2 | 3 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 |
| $b$ | 7 | 0 | 0 | 0 | 1 | 2 | 3 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 |
| $a$ | 8 | 0 | 0 | 0 | 1 | 2 | 3 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 |
| $b$ | 9 | 0 | 0 | 0 | 1 | 2 | 3 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 |
| $i$ | 10 | 0 | 0 | 0 | 1 | 2 | 3 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 |
| $e$ | 11 | 0 | 0 | 0 | 1 | 2 | 3 | 4 | 4 | 4 | 5 | 5 | 5 | 6 | 6 |
| $s$ | 12 | 0 | 1 | 1 | 1 | 2 | 3 | 4 | 4 | 4 | 5 | 5 | 5 | 6 | 7 |

Eventually, we see that the longest common subsequence between $X$ and $Y$ is 7: `ullabes`

5. Binomial coefficients are a family of positive integers that have a number of useful properties and they can be defined in several ways. One way to define them is as an indexed recursive function, $C(n, k)$, where the $C$ stands for "choice" or "combinations". In this case, the definition is as follows:

$$C(n, 0) = 1,$$
$$C(n, n) = 1,$$

and, for $0 < k < n$,

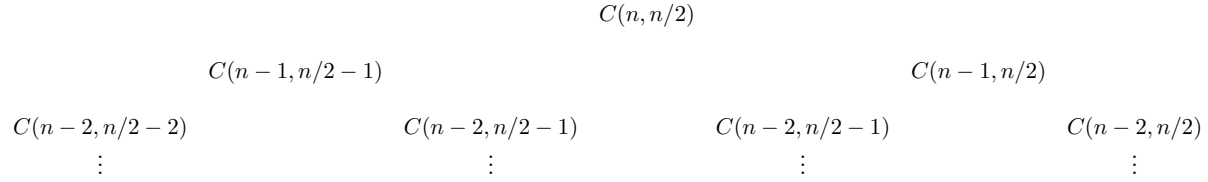
$$C(n, k) = C(n - 1, k - 1) + C(n - 1, k).$$

a) Show that, if we don't use memoization, and $n$ is even, then the running time for computing $C(n, n/2)$ is at least $2^{n/2}$.

b) Describe a scheme for computing $C(n, k)$ using memoization. Give a big-oh characterization of the number of arithmetic operations needed for computing $C(n, \lceil n/2 \rceil)$ in this case.

**Solution:**

a) If we don't use memoization, then every application of the recursive equation doubles the number of calls that we make, until we hit a boundary condition. This is easy to see with a binary tree representation of the successive computations, where the tree height will be at least $n/2$ (since we will start hitting the boundary conditions only from depth $n/2$).




b) If we build Pascal's triangle, then we can apply memoization to this problem. The total number of arithmetic operations needed to compute $C(n, k)$ is $O(n^2)$ in this case, since the Pascal triangle's final height will be $n$ so as to obtain the value of $C(n, \lceil n/2 \rceil)$.

$$\binom{0}{0}$$
$$\binom{1}{0} \quad \binom{1}{1}$$
$$\binom{2}{0} \quad \binom{2}{1} \quad \binom{2}{2}$$
$$\binom{3}{0} \quad \binom{3}{1} \quad \binom{3}{2} \quad \binom{3}{3}$$
$$\binom{4}{0} \quad \binom{4}{1} \quad \binom{4}{2} \quad \binom{4}{3} \quad \binom{4}{4}$$

6. Given a sequence $S = (x_0, x_1, x_2, \ldots, x_{n-1})$ of numbers, describe an $O(n^2)$-time algorithm for finding a longest subsequence $T = (x_{i_0}, x_{i_1}, x_{i_2}, \ldots, x_{i_{k-1}})$ of numbers, such that $i_j < i_{j+1}$ and $x_{i_j} > x_{i_{j+1}}$. That is, $T$ is a longest decreasing subsequence of $S$.

**Solution:**
We can use dynamic programming to solve that problem since it has optimal substructure. Indeed, assume that $L[i]$ is the length of the longest decreasing subsequence of $T[0, \ldots, i]$ that ends with $T[i]$, then $L[i+1]$ can be computed by taking the maximum of $L[j]+1$ for $j < i+1$ when $T[j] < T[i+1]$.

Eventually the longest decreasing subsequence of $S$ is simply the maximum value of $L$.

The time complexity of the algorithm is $O(n^2)$ as we have two nested for loops ranging in $O(n)$ (computing the maximum of $L$ is done independently in $O(n)$).

```
# lis returns length of the longest increasing subsequence
# in arr of size n
def longest_decreasing_subsequence(my_array):
    '''
    This function will compute the length of the longest decreasing
    subsequences of an array of numbers
    INPUT: an array of numbers
    OUTPUT: the length of the longest decreasing subsequence of
    the input array
    '''
    n = len(my_array)

    # Initialise the longest decreasing subsequences to 1
    lds = [1]*n

    # Compute the longest decreasing subsequences values with memoization
    for i in range (1 , n):
        for j in range(0 , i):
            if my_array[i] < my_array[j] and lds[i]< lds[j] + 1 :
                lds[i] = lds[j]+1

    # Find and return the maximal value
    maximum = 0
    for i in range(n): maximum = max(maximum , lds[i])
    return maximum
```

## Hints

- **Question 3: Just work through the recurrence equation.**
- **Question 4: Review the LCS algorithm.**
- **Question 5: Consider how Pascal's triangle applies to this problem.**
- **Question 6: Review the LCS algorithm.**