Algorithm Theory - Assignment 4

Téo Bouvard

March 12, 2020

Problem 1

a) We compute the m-table and the s-table according to the MATRIX-CHAIN-ORDER procedure, with a slight modification : rather than replacing a cell value if the computational cost is lower, we replace it when the cost is higher. By doing this, we find the parenthesization which maximizes the number of scalar multiplications. Note that all indices have been fixed to start at 0 for consistency. This means that matrices are A_0, A_1, A_2, A_3, A_4.

Table 1: m-table

0	15750	18000	21000	43875
	0	2625	6000	17625
		0	750	4500
			0	1250
				0

Table 2: s-table

	0	1	1	0
		1	1	1
			2	3
				3

By reconstructing the optimal solution, we find that the optimal parenthesization is $(A_0(A_1((A_2A_3)A_4)))$ for a total cost of 43875 multiplications.

b) The result have been checked with the python code attached. The only modifications (apart from the zero-index fix) are :

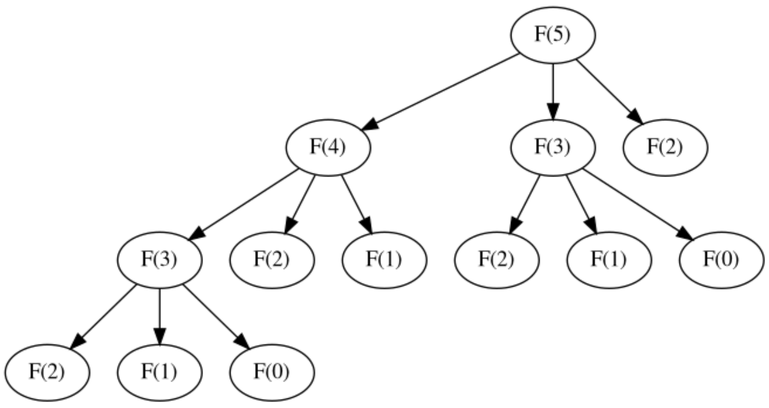
- Replacing the initialization of a cell value from infinity to zero. $m[i][j] = \infty \rightarrow m[i][j] = 0$.
- Replacing the operator in the update condition. $q < m[i][j] \rightarrow q > m[i][j]$.

Problem 2

a) A simple recursive algorithm could be used to compute the values of the series. A possible implementation is given in the function F-DAQ below.

```
F-DAQ($n$)
    if $n < 3$
        return $n$
    return F-DAQ($n-1$) $*$ F-DAQ($n-2$) $+$ $(n-3)$ $*$ F-DAQ($n-3$)
```

b) The subproblem graph for this algorithm is a ternary tree of height $n - 2$.

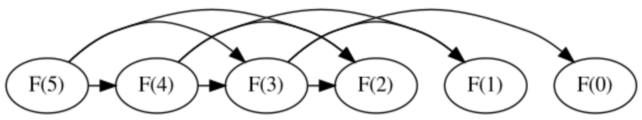


We can see that this approach is highly inefficent as subproblems are solved multiple times in the recursive tree. The time complexity can be defined recursively as $T(n) = T(n-1) + T(n-2) + T(n-3) + \mathcal{O}(1)$. We can see in the subproblem graph that the recursive tree has 3 branches at each level and has a height of $n-2$. This results in a time complexity of $\mathcal{O}(3^{n-2})$.

c) We can solve this problem dynamically, by using the algorithm F-DYN

```
F-DYN($n$)
    let $v[0..n]$ be an array
    **for** $i = 0$ **to** $n$
        **if** $i < 3$
            $v[i] = i$
        **else**
            $v[i] = v[i-1] * v[i-2] + (i-3) * v[i-3]$
    **return** $v[n]$
```

d) In this case, the complexity is linear because we only do a single pass over the array of intermediate values. Thus, the time complexity of F-DYN is $\mathcal{O}(n)$.

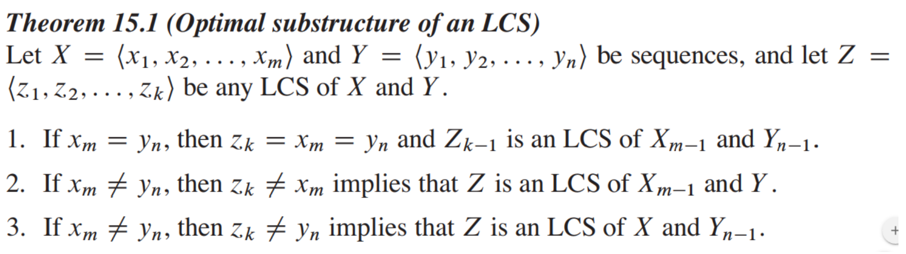


Problem 3

a) Dynamic programming is efficient on problems having

- Optimal substructure, meaning that the optimal solution to the original problem contains the optimal solutions to its subproblems.
- Overlapping subproblems, meaning that the original problem can be split into subproblems with some subproblems being identical to one another.

Theorem 15.1 of Introduction to Algorithms proves the optimal substructure of LCS.



Theorem 15.1 (Optimal substructure of an LCS)
Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y.

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

To find the LCS of X and Y, we have to find the LCS of X and Y_{n-1} as well as the LCS of X_{m-1} and Y. Both of these subproblems contain the subproblem of finding the LCS of X_{n-1} and Y_{m-1}. This shows why the LCS problem has overlapping subproblems.

b) When developing an algorithm based on dynamic programming, the main steps are the following.

- Characterize the structure of an optimal solution
- Recursively define the value of an optimal solution
- Compute the value of an optimal solution
- Optionally, construct the optimal solution from the computed information

c) The longest common subsequence of CADACA and CACAQ is CACA.

		C	A	D	A	C	A
	0	0	0	0	0	0	0
C	0	↖ 1	← 1	← 1	← 1	↖ 1	← 1
A	0	↑ 1	↖ 2	← 2	↖ 2	← 2	↖ 2
C	0	↖ 1	↑ 2	↑ 2	↑ 2	↖ 3	← 3
A	0	↑ 1	↖ 2	↑ 2	↖ 3	↑ 3	↖ 4
Q	0	↑ 1	↑ 2	↑ 2	↑ 3	↑ 3	↑ 4