CS-E3190 Principles of Algorithmic Techniques

02. Recursive Algorithms – Tutorial Exercise

1. **Matrix multiplication.** Let $A, B \in \mathbb{R}^{2\times 2}$ and $C = AB$ s.t.

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}, C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}.$$

Following the naive approach, C is computed as follows,

$$\begin{aligned} c_{11} &= a_{11}b_{11} + a_{12}b_{21} \\ c_{12} &= a_{11}b_{12} + a_{12}b_{22} \\ c_{21} &= a_{21}b_{11} + a_{22}b_{21} \\ c_{22} &= a_{21}b_{12} + a_{22}b_{22} \end{aligned}$$

For this naive approach, 8 multiplications and 4 additions are needed.

The Strassen's algorithm enables to compute C with only 7 multiplications as follows,

$$\begin{aligned} P_1 &= (a_{11} + a_{22})(b_{11} + b_{22}) \\ P_2 &= (a_{21} + a_{22})b_{11} \\ P_3 &= a_{11}(b_{12} - b_{22}) \\ P_4 &= a_{22}(b_{21} - b_{11}) \\ P_5 &= (a_{11} + a_{12})b_{22} \\ P_6 &= (a_{21} - a_{11})(b_{11} + b_{12}) \\ P_7 &= (a_{12} - a_{22})(b_{21} + b_{22}) \end{aligned}
\qquad
\begin{aligned} c_{11} &= P_1 + P_4 - P_5 + P_7 \\ c_{12} &= P_3 + P_5 \\ c_{21} &= P_2 + P_4 \\ c_{22} &= P_1 - P_2 + P_3 + P_6 \end{aligned}$$

(a) Let $A, B \in \mathbb{R}^{n\times n}$ and $C = AB$. Write a recursive algorithm based on Strassen's design for computing C.

Hint: you can assume n is a power of two, since the matrices can be padded when implementing the algorithm.

Solution. First we split A and B into 4 blocks of size $(\frac{n}{2} \times \frac{n}{2})$, we use the following notations:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Then the algorithm is given as follows,

Algorithm 1: *Product*(A, B, n)

if $n = 1$ **then**
  $c_{11} = a_{11} \cdot b_{11}$
else
  $m \leftarrow \frac{n}{2}$
  $A_{11}, A_{12}, A_{21}, A_{22} \leftarrow Split(A)$
  $B_{11}, B_{12}, B_{21}, B_{22} \leftarrow Split(B)$
  $P_1 = \textbf{Product}((A_{11} + A_{22}), (B_{11} + B_{22}), m)$
  $P_2 = \textbf{Product}((A_{21} + A_{22}), B_{11}, m)$
  $P_3 = \textbf{Product}(A_{11}, (B_{12} - B_{22}), m)$
  $P_4 = \textbf{Product}(A_{22}, (B_{21} - B_{11}), m)$
  $P_5 = \textbf{Product}((A_{11} + A_{12}), B_{22}, m)$
  $P_6 = \textbf{Product}((A_{21} - A_{11}), (B_{11} + B_{12}), m)$
  $P_7 = \textbf{Product}((A_{12} - A_{22}), (B_{21} + B_{22}), m)$
  $C_{11} = P_1 + P_4 - P_5 + P_7$
  $C_{12} = P_3 + P_5$
  $C_{21} = P_2 + P_4$
  $C_{22} = P_1 - P_2 + P_3 + P_6$
end

Where the *Split* function, takes a matrix of size $n \times n$ and divides it into four matrices of size $(\frac{n}{2} \times \frac{n}{2})$.

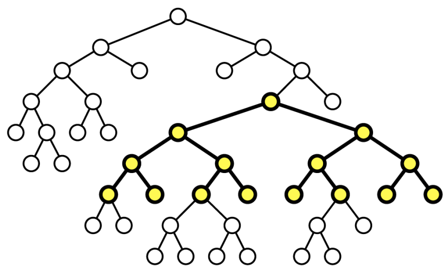
(b) Analyze the running time of your algorithm.

Solution. Note that four blocks of C may be computed independently from each other. For computing $Product(A, B, n)$, we first need to compute P_1 to P_7 recursively for the matrices of size $((\frac{n}{2} \times \frac{n}{2})$. Hence, we may come up with the following recurrence relation,

$$T(n) = 7T\left(\frac{n}{2}\right) + O(n^2)$$

We need $O(n^2)$ arithmetic operations to combine the sub-problems P_i , for $i \in \{1, 2, \dots 7\}$. Since $\log_2 7 \geq 2$, we conclude using the Master theorem that $T(n) = O(n^{\log_2 7}) \ll O(n^3)$.

2. **Complete sub-trees.** A binary tree is complete if every internal node has two children, and every leaf has exactly the same depth.



Describe and analyze a recursive algorithm that computes the largest complete sub-tree of a given binary tree. Your algorithm should return both the root and the depth of this sub-tree.

Solution. We will describe a recursive algorithm $DCS(v)$ that labels each node v of the tree with the depth of its largest complete sub-tree.

Let us consider a binary tree rooted at v. This node has two children v_l and v_r. If we delete the edge between v_l and v, we have two connected sub-graphs. We denote the sub-graph rooted at v_l by $left(v)$. Similarly we denote the sub-gragh rooted at v_r with $right(v)$.

DCS(v):

(i) If $left(v)$ or $right(v)$ is $Null$ then

$$DCS(v) \leftarrow 0.$$

(ii) Else:

$$DCS(v) \leftarrow min\{DCS(v_l), DCS(v_r)\} + 1.$$

Correctness: By induction this algorithm outputs a correct answer. Indeed,

- Let us consider a leaf node v_0, our algorithm returns 0. This proves the base case.
- Now consider an arbitrary node v with its children v_l and v_r either already correctly labeled by $DCS(v)$ or null (induction hypotheses).

  If $left(v)$ or $right(v)$ are $Null$, then $DCS(v)$ correctly returns 0, as v can only be a leaf in a complete binary sub-tree. If $DCS(v_l)$ and $DCS(v_r)$ are m and n respectively then depth of the largest sub-tree rooted at v equals the minimum of m and n plus one. Since v has two children and can be considered as part of the complete sub-tree, this will increase the depth of the largest sub-tree by one. Hence, if a step of the algorithm is correct then the next one is as well.

We now describe an algorithm $LCS(T)$, which returns the depth of the largest sub-tree of a binary tree T and its root node. It simply consists in going through all nodes' labels and picking the maximum.

LCS(T):

(i) $maxDepth \leftarrow -\infty$.

(ii) Run $DCS(v_0)$ where v_0 is the root of T (now all the nodes in T are labeled)

(iii) $\forall v \in T$, if $DCS(v) > maxDepth$, then

$$maxDepth \leftarrow DCS(v)$$
$$maxRoot \leftarrow v.$$

(iv) return $maxDepth$ and $maxRoot$.

The algorithm $DCS(v)$ requires $O(1)$ operations (one comparison and one addition) for each node and thus in total runs in $O(n)$.

The time complexity needed for $LCS(T)$ is the complexity of $DCS(v)$ plus the complexity of the scan of tree which is also in $O(n)$.