Lists, trees, and recursive algorithms for them

Definition. $L$ is a list if either

- $L$ is NULL
- or $L$ is a pointer to a data structure with fields data and next, where next is a list.

Definition. $T$ is a tree if either

- $T$ is NULL
- or $T$ is a pointer to a data structure with fields data and children, where children is an array (alternatively, a list) of trees.

Lists

---

In List $L$

Out The sum of all data-values in $L$

function SumIterative(L)
    result = 0
    while $L \neq$ NULL do
        result = result + $L$.data
        $L$ = $L$.next
    end while
    return result
end function

---

In List $L$

Out The sum of all data-values in $L$

function SumRecursive(L)
    if $L$ == NULL then
        return 0
    end if
    return $L$.data + SumRecursive($L$.next)
end function
Trees

In Tree $T$

Out The sum of all data-values in $T$

\[
\text{function } \text{Sum}(T) \\
\quad \text{if } T == \text{NULL} \text{ then} \\
\quad \quad \text{return } 0 \\
\quad \text{end if} \\
\quad \text{result } = T.\text{data} \\
\quad \text{for } C \text{ in } T.\text{children} \text{ do} \\
\quad \quad \text{result } = \text{result } + \text{Sum}(C) \\
\quad \text{end for} \\
\quad \text{return result} \\
\text{end function}
\]

Note that this scheme can be easily adapted to compute min / max, product, and other similar functions. For example, see the following code for min.

In Tree $T$

Out The minimum of all data-values in $T$

\[
\text{function } \text{Min}(T) \\
\quad \text{if } T == \text{NULL} \text{ then} \\
\quad \quad \text{return } \infty \\
\quad \text{end if} \\
\quad \text{result } = T.\text{data} \\
\quad \text{for } C \text{ in } T.\text{children} \text{ do} \\
\quad \quad \text{result } = \min(\text{result}, \text{Min}(C)) \\
\quad \text{end for} \\
\quad \text{return result} \\
\text{end function}
\]

Sometimes, one has to compute more than needed during recursion in order to solve the problem. Consider the following problem

In Tree $T$

Out The second smallest data-value in $T$ (or $\infty$ if there are less than two vertices in $T$)

It can be solved using the following recursive algorithm
In Tree $T$

Out The pair $(a, b)$, where $a$ and $b$ are the smallest and the second smallest data-values in $T$, respectively

function TwoMin($T$)

if $T == NULL$ then

return $(\infty, \infty)$

end if

min = $T$. data

second_min = $T$. data

for $C$ in $T$. children do

$(C_{\text{min}}, C_{\text{second}}) = \text{TwoMin}(C)$

second_min = min(max(min, $C_{\text{min}}$), second_min, $C_{\text{second}}$)

min = min(min, $C_{\text{min}}$)

end for

return (min, second_min)

end function