Rooted Tree Implementation and Traversal

CSCI2100
Tutorial 8
How to store a rooted tree in memory?
For each node, create a linked list of pointers (one per child).
In general, the space of storing $n$ nodes is $O(n)$. 
Rooted tree traversal

**Problem:** Given the root of a tree, count the number of nodes in the tree.

**Goal:** $O(n)$ time.

We will achieve the purpose by giving an algorithm to traverse the tree.
Rooted tree traversal

Given a5, how do we find the number of nodes in the tree?
A recursive view

- Recursively count the subtree of each child of the root.
A recursive view

• Recursively count the subtree of each child of the root.

\[
\text{count}(r): \\
\text{result} = 1 \\
\text{for each child } u \text{ of } r: \\
\text{result} = \text{result} + \text{count}(u) \\
\text{return result}
\]
Analysis – the smart way

• Intuitively, we visit each edge twice (descending once and ascending once).

• So the cost is

$$O(#\text{nodes} + # \text{edges}) = O(n)$$
Analysis – the “standard” way

Let \( f(n) \) denote the running time on a tree \( T \) of \( n \) nodes (we denote \(|T|\) as the number of nodes in \( T \)).

\[
\text{count}(r):
\begin{align*}
\text{result} &= 1 \\
\text{for each child } u \text{ of } r: & \quad \text{result} = \text{result} + \text{count}(u) \\
\text{return } \text{result}
\end{align*}
\]
Analysis – the “standard” way

For $n = 1$, we have:
$$f(1) = O(1)$$

For $n \geq 2$:
$$f(n) \leq f(|T_1|) + \cdots + f(|T_k|) + O(k + 1)$$

Where $T_1, T_2, \ldots, T_k$ are the subtrees at the child nodes of the root.

We can prove:
$$f(n) = O(n)$$

by the substitution method (left to you).
The recursive implementation may not work in today’s operating systems

• Every operating system today limits the depth of recursion
  • Typically at the order of hundreds.

• Our earlier program will crash if the tree is too tall.

• Next, we will see a non-recursive implementation based on a stack.
A stack-based implementation

```
a1 -> a1
a2 -> a1 -> a3
a3 -> a7 -> a4 -> a8
a4 ->
a5 -> a9
a6 -> a6 -> a2
a7 ->
a8 ->
a9 -> a10
a10 ->
a5
```
A stack-based implementation

*a3

a pointer remembering the child under processing
A stack-based implementation

```
a1 → a2 → a3 → a4 → a5 → a6 → a7 → a8 → a9 → a10
```

push the child into the stack
A stack-based implementation

pointer to the first child of node 9
A stack-based implementation

push the child into the stack
A stack-based implementation

node 10 has no children.
A stack-based implementation
A stack-based implementation

node 9 has no more children.

a10 has been popped
A stack-based implementation

A9 has been popped
A stack-based implementation

moving the pointer to the next child of node 5
A stack-based implementation

The algorithm then continues in the same fashion.
A stack-based implementation

- Running time = O(n) because every node in the linked lists is pushed once and popped once.