Tutorial 6: Trees and Algorithms

CAB301 - Algorithms and Complexity

School of Computer Science, Faculty of Science

Tree

A **tree** is a collection of nodes connected by directed edges.

<div class="flexbox"> <div style="flex: 0.5">

Various implementations, but in CAB301, each node:

- Has a piece of data (`Key`)
- Reference to its `FirstChild`
- Reference to its `FirstSibling`

</div> <div style="flex: 0.5">

```
public class Node
{
    public int Key;
    public Node FirstChild
```

Breath First Traversal

Visit all nodes at a given depth before moving to the next depth. Use a `Queue` to store sibling nodes. Nearer siblings (first in) are visited first (first out).

<div class="flexbox"> <small style="font-size: 22px; flex: 0.5">

```
ALGORITHM $BreadthFirstTraversal(root)$
$q \leftarrow \emptyset$ // Empty queue
$q.\mathrm{enqueue}(root)$
while $q \neq \emptyset$ do
    $r \leftarrow q.\mathrm{dequeue}()$
    $\mathrm{visit}\ r$
    $r \leftarrow r.\mathrm{FirstChild}$
     while $r \neq \mathrm{null}$ do
```

Depth First Traversal

Visit all nodes in a branch before moving to the next branch. Use a `Stack` to store sibling nodes. Deeper siblings (last in) are visited first (first out).

<div class="flexbox"> <small style="font-size: 22px; flex: 0.5">

```
**ALGORITHM** $DepthFirstTraversal(root)$
$s \leftarrow \emptyset$ // Empty stack
$s.\mathrm{push}(root)$
**while** $s \neq \emptyset$ **do**
        $r \leftarrow s.\mathrm{pop}()$
         **while** $r \neq \mathrm{null}$ **do**
                $\mathrm{visit}\ r$
                **if** $r.\mathrm{FirstSibling} \neq \mathrm{null}$
```