Case study

Problem: When colour is applied to a part of a web page, what other parts of the page will obtain the same colour?

For example, if a section is coloured, the paragraphs and lists will get the same colour.
Basic definitions

A tree is formed of nodes connected by edges.

In a rooted tree, one node is designated as the root of the tree.

In a drawing where the root is at the top, an edge connects a parent to a child, where the parent is the node closer to the root.

Nodes that share a parent are siblings.

A node without children is a leaf; a node that is not a leaf is an internal node.

A node’s parent, its parent’s parent, and so on up to the root are its ancestors; a node’s children, children’s children, and so on are its descendants.

A node and all its descendants form the subtree rooted at that node.
Definitions for rooted trees

A tree is unordered if there is no order specified on the children of a node, and ordered otherwise.

A binary tree is a tree in which each parent has at most two children and each child is specified as either a left child or a right child. The subtree rooted at the left child is the left subtree and the subtree rooted at the right child is the right subtree.

The depth of a node is the number of ancestors of the node; a root is thus at depth 0.

The height of a node is determined by considering, in the subtree rooted at the node, the maximum number of edges passed going from a leaf to the node. A leaf is thus at height 0.

The height of a tree is the height of the root of the tree.
Tree traversals

A **tree traversal** is an ordering of the nodes in the tree.

- In a **postorder traversal**, each node appears after its children.
- In a **preorder traversal**, each node appears before its children.
- In a **level order traversal**, nodes appear in increasing order of depth.
- In an **inorder traversal** (only in a binary tree), for each node all nodes in the left subtree come before the node and all nodes in the right subtree come after the node.
Traversing Example
In a decision tree representing an algorithm, each node is labeled with a decision being made, the children of a node correspond to the different choices depending on the outcome of the decision, and each leaf corresponds to a possible outcome of the algorithm.
ADT Unordered Tree

Data: Nodes.

Operations:

- CreateUT()
- IsEmptyUT(U)
- RootUT(U)
- ParentUT(U, node)
- ChildrenUT(U, node)
- AddNodeUT(U, parent, label)
- DeleteNodeUT(U, node)
Completing the specification

Questions:

• If a node has no parent, what does ParentUT return?
• How are multiple children returned by ChildrenUT?
• How is AddNodeUT used to add the root?
• What tree results when a node is deleted?

Possible answers:

• The node itself.
• ADT Set.
• No parent is given.
• Restrict to leaves.

Note: We will drop UT when it is clear from context.
Other tree operations

- LeavesUT(U)
- PostOrderUT(U)
- PreOrderUT(U)
- LevelOrderUT(U)
- SwapSubtrees(U, node1, node2)
- DeleteSubtreeUT(U, node)
ADT Binary Tree

Data: Nodes.

Operations:

- CreateBT()
- IsEmptyBT(B)
- RootBT(B)
- ParentBT(B, node)
- LeftChildBT(B, node)
- RightChildBT(B, node)
- Sibling(B, node)
- AddNodeBT(B, parent, label, side)
- DeleteNodeBT(B, node)
Array implementation of a binary tree

```
3
5 10
8 13 11
14 18
3 5 10 8 13 11 14 18
0 1 2 3 4 5 6 7 8 9 10 11
```
More terminology for binary trees

In a **perfect** binary tree, each node has zero or two children and all leaves are at the same depth.

In a **complete** binary tree every level, except possibly the last, is completely filled, and all nodes on the last level are as far to the left as possible.
ADT Ordered Tree

Data: Nodes.

Operations:
- CreateOT()
- IsEmptyOT(O)
- RootOT(O)
- ParentOT(O, node)
- ChildrenOT(O, node)
- FirstChildOT(O, node)
- PrevSiblingOT(O, node)
- NextSiblingOT(O, node)
- AddNodeOT(O, parent, sibling, label)
- DeleteNodeOT(O, node)

Note: Use ADT List for ChildrenOT so that they are in order.
Alternate implementation of node

Node:

- label/item stored
- pointer to parent if first child, otherwise previous sibling
- pointer to first child
- pointer to next sibling
Pseudocode for ParentOT(O,x)

found ← False
current ← x
while not found
    y = parent or previous sibling of current
    if current is first child of y
        found ← True
    else
        current = y
return y