Tree Definitions & Types of Trees
On Pointers

- Pointers represent relationships between objects
  - In a singly linked list they show the successor (next) relationship

```
+---------+----------+--------+
| data    | successor|
|         | next     |
+---------+----------+--------+
```

- In a doubly linked list one pointer shows the successor relationship and the other pointer shows the predecessor (prev) relationship

```
+---------+----------+--------+
| data    | successor|
|         | next     |
+---------+----------+--------+
```

prev
data
successor
next

© Gunnar Gotshalks
• Pointers in singly and doubly linked lists show an ordering relationship

• A pointer can be used to show any binary relationship

   [Diagram of Alice pointing to Bob]

The above can represent

» Alice is the mother of Bob
» Alice is the mentor of Bob
» Alice telephones Bob
» Alice emails Bob
» etc.
Objects can have the same relationship with more than one other object

A is the mother of B, C and D
A is the mentor of B, C and D
A telephones B, C and D
A emails B, C and D etc.
Trees

- Trees show a relationship among a collection of objects (nodes), where relationships are one way and only one object (node) is at the head of every arrow.

- Example of a tree structure with nodes labeled A, B, C, D, E, F, G, H. Relationships include supervises, mother of, contains, owns, and et cetera.
Terminology

- **Parent** – node at the tail of an arrow
- **Child** – node at the head of an arrow
- **Siblings** – nodes with the same parent

![Diagram of tree terminology]

- A is the parent of D
- B, C & D are siblings
- D is a child of A
• **Ancestor** – the node itself, parent, parent of parent, etc.

• **Descendent** – the node itself, child, child of child, etc.
Terminology – 3

- **Root** – node with no parent
- **Leaf** – node with no children – also called **external** (all other nodes are **internal**)

![Tree Diagram]

© Gunnar Gotshalks
Tree Definition

- An empty tree is a tree
- A node of type T with a finite number of children where each child is a disjoint tree of base type T, called a subtrees

A subtree

All subtrees are empty, leaf node
Terminology – 4

- **Path** – the set of edges from the root to a node
- **Path length** – the number of edges in a path

Path from A to H is 
\(<A, B> <B, F> <F, H>\)

Length is 3
**Terminology – 5**

- **Node level & node depth** – the path length from the root
  - The root is level 0 and depth 0
  - Other nodes depth is 1 + depth of parent

![Diagram of a tree with node levels and depths labeled]

- Depth = Level = 0
- Depth = Level = 1
- Depth = Level = 2
- Depth = Level = 3
• **Height of a tree** – The longest path length from the root to a leaf.
  
  > Non empty tree: Height = max depth

![Diagram of a tree with labeled heights and depths]
**Terminology – 7**

- **Degree** – the maximum number of possible children for every node in the tree

- **Proper tree (full tree)** – Nodes have all children non-void, or all children are void

  > A full tree of height 2 and degree 3

  ![Diagram of a full tree of height 2 and degree 3]

  > If some of the nodes have fewer actual children the tree is still of degree 3
• **Complete** – All levels are full except for the deepest level, which is partially filled from the left

> A complete binary tree of degree 2
**Terminology – 9**

- **Balanced** – Different definitions depending upon type of tree
  - Having $1/N$ of the nodes in each of $N$ children
  - Height of all subtrees within constant $K$
    - In a binary tree
      - $\text{Height(left\_subtree)} - \text{Height(right\_subtree)} \leq K$
  - $\max\_\text{level(leafNode)} - \min\_\text{level(leafNode)} \leq K$
    - For a complete tree $K=1$

- **Balance** – Redistribute the nodes to restore balance constraint while maintaining the ordering property
Types of Trees

- General tree
  - Every node can have any number of sub-trees, there is no maximum
  - Different number is possible of each node

- N’ary tree
  - Every node has at most N sub-trees
    - Special case N= 2 is a binary tree
    - Sub-trees may be empty – pointer is void
Ordered Trees

• Can be general or N’ary but have one additional constraint
  » An ordering is imposed between the parent and its children
  » Each node has one or more keys that are in an order relationship ( < ≤ ≥ > ) with the keys in its children
    > The same relationship pattern is used throughout for all nodes in the tree
Ordered Trees – N’ary case

• Each note contains 1 to N–1 keys

\[ \text{key}_1 < \text{key}_2 < \ldots < \text{key}_{n-1} \]
Ordered Trees – Binary Search Tree

- Special case N’ary with N=2
  - Complementary relationship of the parent with two children
    - \text{key\_left\_child} < \text{key\_parent} < \text{key\_right\_child}
Ordered Trees – Heap

• Special case N’ary with N=2 and a complete binary tree
  » Same relationship between the parent and each child
    \[ \text{key}_{\text{left}_\text{child}} < \text{key}_{\text{parent}} > \text{key}_{\text{right}_\text{child}} \]
N’ary Tree Nodes

- Unordered & ordered trees – for small N
  
  » Data + specific names for pointer fields
N’ary Tree Nodes – 2

• Unordered trees – large N
  » Data + array of pointers

• Ordered trees – large N
  » Array of keys and an array of pointers that are logically interspersed
General Tree Nodes

- Use list of pointers – for any number of children
Array Representation of N’ary Trees

- If N’ary trees are complete, then can use arrays to store the data.
  - Pointers are indices to the array (addresses relative to the start of the array scaled by the size of a pointer)
  - Use arithmetic to compute where the children are

- Binary trees are a special case
  - Heaps are usually implemented using arrays to represent a complete binary tree
Array Representation – 2

- Mapping between a complete binary tree and an array

left = 2 * parent

right = 2 * parent + 1
Array Representation – 3

• In general for an N’ary tree the following set of relationships holds – the root must have index 0

  \[
  \begin{align*}
  \text{child}_1 &= N \times \text{parent} + 1 \\
  \text{child}_2 &= N \times \text{parent} + 2 \\
  \text{child}_3 &= N \times \text{parent} + 3 \\
  \text{...} \\
  \text{child}_N &= N \times \text{parent} + N
  \end{align*}
  \]

  Nodes are numbered in breadth traversal order

• The binary case is an exception where the root can be index 1 because \(2 \times 1 = 2\), the index adjacent to the root

  » This gives the pair \(2 \times \text{parent} \& 2 \times \text{parent} + 1\), which is less arithmetic than the above, and the inverse to find the parent is easier to compute.
Representing General Trees as Binary Trees

• Binary trees are all that are logically necessary

  » Lisp programming language for artificial intelligence applications has binary trees as its fundamental (and in theory only) data structure.
An Example

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
       /\   /
      H  E  F
       \  /  \
        G  H
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```

```
A
  /\   \\
  B  C  D
  /   \   /
 E    F  G
```