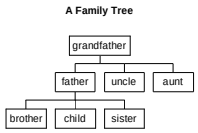
Trees

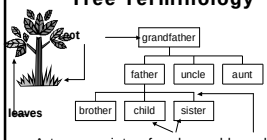
Linear lists
- e.g.,arrays, stacks, queues, and linked lists
- have a unique first and last element
- each interior item has a unique successor

Tree is a nonlinear structure
- a member may have multiple successors

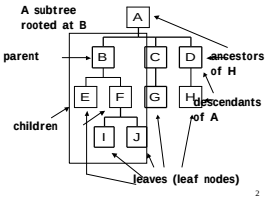


1

Tree Terminology

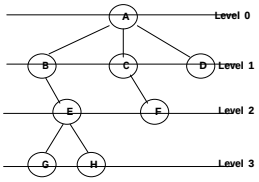


- A tree consists of nodes and branches.



2

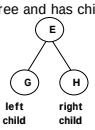
- The level of a node is the length of the path from the root to the node.
- The depth of a tree is the maximum level of any node in the tree.



3

Binary Trees

- Each node may have 0, 1, or 2 children.
- A binary tree has a recursive structure: each node is the root of its own subtree and has children.



- At any level n, a binary tree may contain from 1 to 2^n nodes.
- Density is a measure of the size of a tree (number of nodes) relative to the depth of the tree.

4

Various Tree Types

Degenerate tree
- there is a single leaf node and each nonleaf node has only one child
  - this is equivalent to a linked list

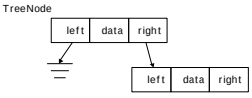
Complete binary tree
- N is a tree in which each level 0 to N-1 has a full set of nodes and all leaf nodes at level N occupy the leftmost positions in the tree.

Full tree
- A complete binary tree that contains 2^N nodes at level N.

5

Binary Tree Structure

- A binary tree structure is built with nodes.
- A tree node contains a data field and two pointer fields:
  - left pointer (left)
  - right pointer (right)
- The root node defines an entry point into the binary tree
- A pointer field specifies a node at the next level in the tree.
- A leaf node has a NULL left and right pointer.



6

TreeNode Class

```
#ifndef TREENODE_CLASS
#define TREENODE_CLASS

#ifndef NULL
const int NULL = 0;
#endif // NULL

// BinSTree depends on TreeNode
template <class T>
class BinSTree;

// declares a tree node object for a binary tree
template <class T>
class TreeNode
{
   . . .
}
#endif // TREENODE_CLASS
```

7

```
// TreeNode applications, protected and private are equivalent
template <class T>
class TreeNode
{
 protected:
    // points to the left and right children of the node
    TreeNode<T> *left;
    TreeNode<T> *right;
  public:
    // public member allowing the client to update its value
    T data;

    // constructor
    TreeNode (const T& item,
              TreeNode<T> *lptr = NULL,
              TreeNode<T> *rptr = NULL);
    // virtual destructor.
    virtual ~TreeNode(void);

    // access methods for the pointer fields
    TreeNode<T>* Left(void) const;
    TreeNode<T>* Right(void) const;

    // BinTree needs access to left and right
    friend class BinSTree<T>;
};
```

8

```
// constructor. initialize the data and pointer fields.
// the pointer NULL assigns an empty tree
template <class T>
TreeNode<T>::TreeNode (const T& item,
                       TreeNode<T> *lptr,
                       TreeNode<T> *rptr)
   : data(item), left(lptr), right(rptr)
{}

// method Left allows the user to reference the left child
template <class T>
TreeNode<T>* TreeNode<T>::Left(void) const
{
  // return the private member value left
  return left;
}

// method Left allows the user to reference the right child
template <class T>
TreeNode<T>* TreeNode<T>::Right(void) const
{
  // return the private member value right
  return right;
}
```

9

```
// does nothing. exists so nodes derived from it will be
// destroyed properly by delete. used in Chapter 13 for
// AVL trees
template <class T>
TreeNode<T>::~TreeNode(void)
{}
```

10