Trees

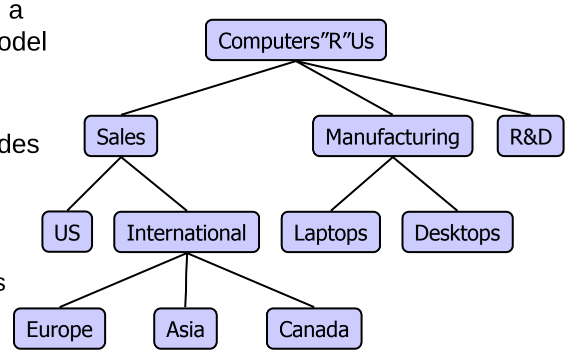
CSE 2011
Fall 2009

Trees

- Linear access time of linked lists is prohibitive
  - Does there exist any simple data structure for which the running time of most operations (search, insert, delete) is O(log N)?
- Trees
  - Basic concepts
  - Tree traversal
  - Binary trees
  - Binary search trees and operations

What is a Tree?

- In computer science, a tree is an abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation
- Applications:
  - Organization charts
  - File systems
  - Programming environments



Recursive Definition

- A tree is a collection of nodes.
  - The collection can be empty.
  - Otherwise, a tree consists of a distinguished node *r* (the *root*), and zero or more nonempty *subtrees* T_1, T_2, . . . , T_k, each of whose roots is connected by a directed *edge* from *r.*

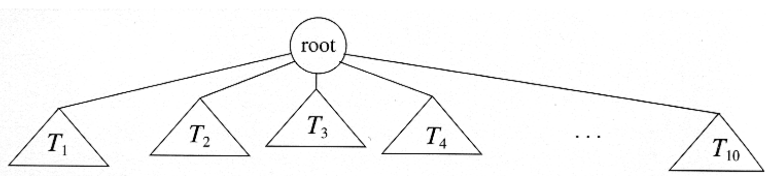


Figure 4.1 Generic tree

Terminologies

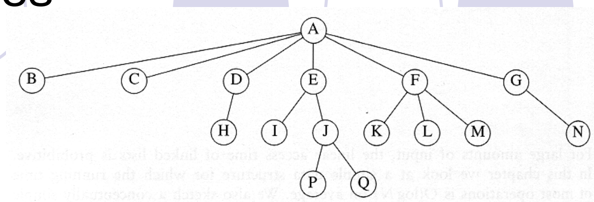


Figure 4.2 A tree

- *Child* and *Parent*
  - Every node except the root has one parent
  - A node can have zero or more children
- *Leaves*
  - Leaves are nodes with no children
- *Sibling*
  - Nodes with the same parent
- *Ancestor* and *descendant*
  - If there is a path from n1 to n2 then
    - n1 is an ancestor of n2
    - n2 is a descendant of n1
  - *Proper ancestor* and *proper descendant* if n1 ≠ n2

Terminologies (2)

- *Path*
  - a sequence of edges
- *Length of a path*
  - number of edges on the path
- *Depth* of a node
  - length of the unique path from the root to that node

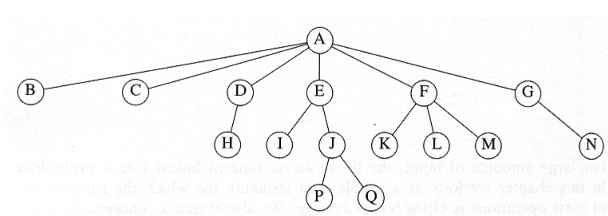


Figure 4.2 A tree

Terminologies (3)

- *Height* of a node
  - length of the longest path from that node to a leaf
  - all leaves are at height 0
- The height of a tree = the height of the root
  = the depth of the deepest leaf

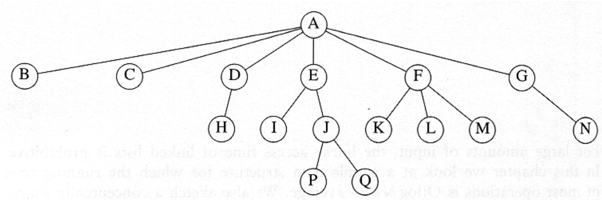


Figure 4.2 A tree

Example: UNIX Directory

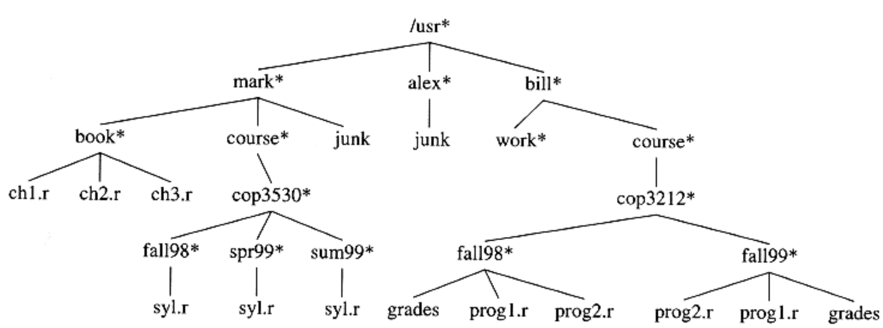


Figure 4.5 UNIX directory

Example: Expression Trees

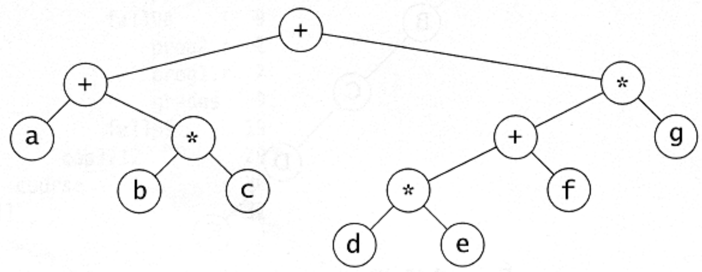


Figure 4.14 Expression tree for (a + b * c) + ((d * e + f) * g)

- Leaves are operands (constants or variables)
- The internal nodes contain operators

Tree ADT

- We use positions to abstract nodes (position ≡ node)
- Generic methods:
  - integer size()
  - boolean isEmpty()
  - Iterator elements()
  - Iterator positions()
- Accessor methods:
  - position root()
  - position parent(p)
  - positionIterator children(p)

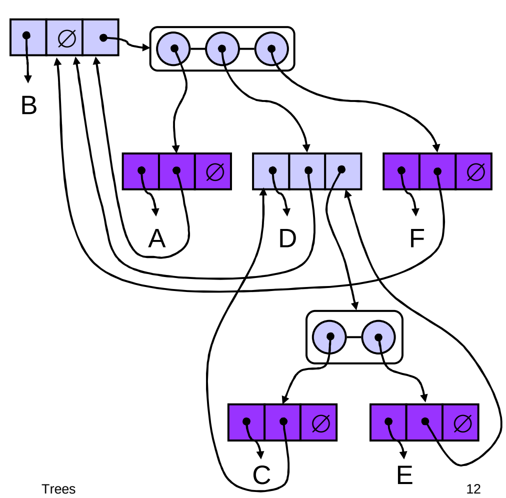
- Query methods:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)
- Update method:
  - object replace (p, e): replace with e and return element stored at node p
- Additional update methods may be defined by data structures implementing the Tree ADT

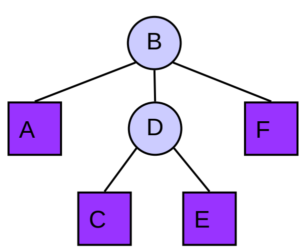
Implementing Trees

- Arrays ?
- Linked “lists” (pointers) ?

Linked Structure for Trees

- A node is represented by an object storing
  - Element
  - Parent node
  - Sequence of children nodes

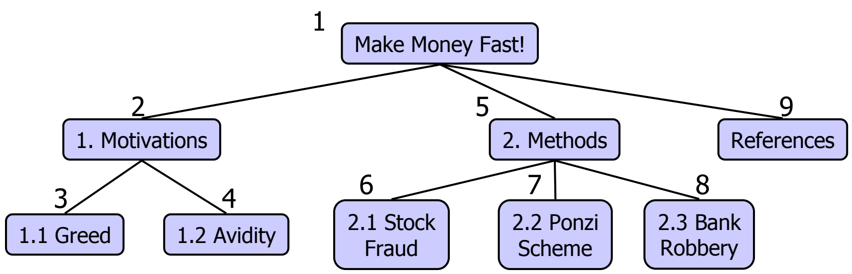




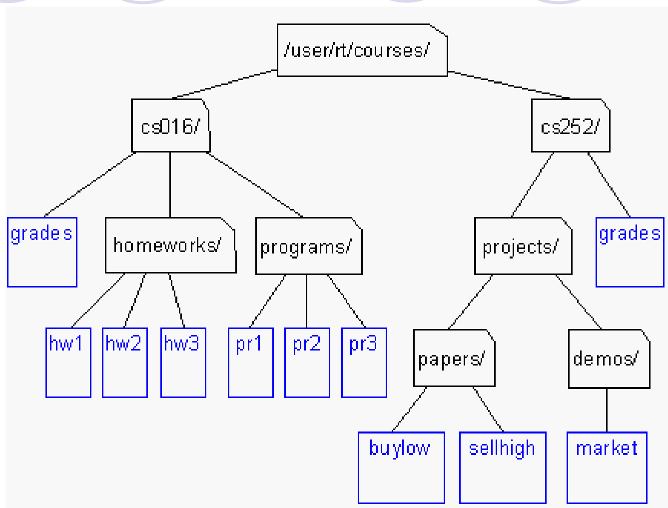
Preorder Traversal

- A traversal visits the nodes of a tree in a systematic manner
- In a preorder traversal, a node is visited before its descendants
- Application: print a structured document

Algorithm ***preOrder***(*v*)
    visit(*v*)
    for each child ***w*** of ***v***
        preOrder (***w***)



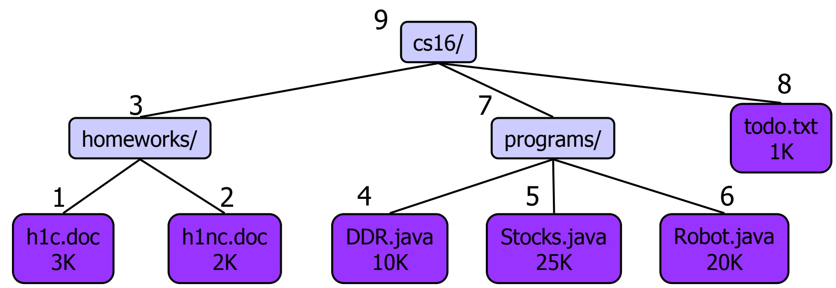
An Example



Postorder Traversal

- In a postorder traversal, a node is visited after its descendants
- Application: compute space used by files in a directory and its subdirectories

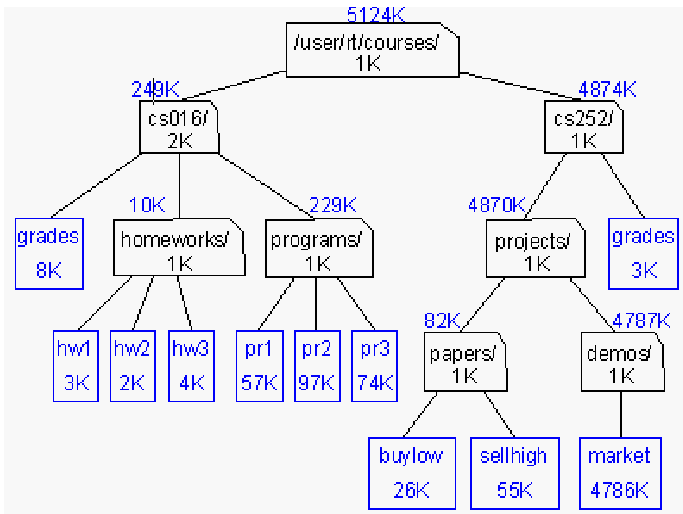
Algorithm ***postOrder***(***v***)
    for each child ***w*** of ***v***
        postOrder (***w***)
    visit(***v***)



Applications

- Either preorder traversal or postorder traversal can be used when the order of computation is not important.
  Example: printing the contents of a tree (in any order)
- Preorder traversal is required when we must perform a computation for each node ***before*** performing any computations for its descendents.
  Example: Printing the headings of chapters, sections, sub-sections of a book.
- Postorder traversal is needed when the computation for a node *v* requires the computations for *v*'s children to be done first.
  Example: Given a file system, compute the disk space used by a directory.

Example: Computing Disk Space



Example: UNIX Directory Traversal

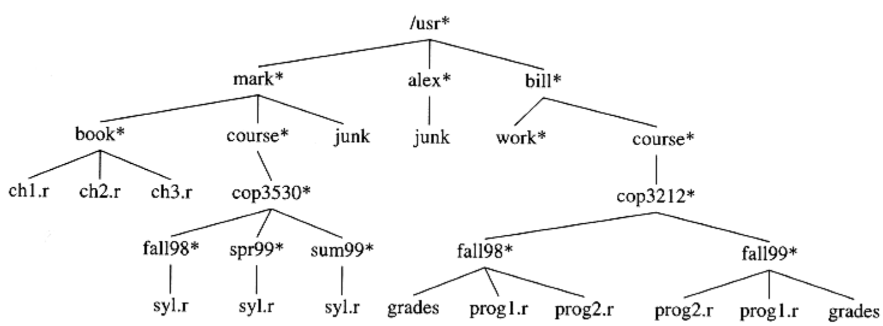


Figure 4.5 UNIX directory

Example: Unix Directory Traversal

Preorder

```
/usr
    mark
        book
            ch1.r
            ch2.r
            ch3.r
        course
            cop3530
                fall98
                    syl.r
                spr99
                    syl.r
                sum99
                    syl.r
        junk
    alex
        junk
    bill
        work
        course
            cop3212
                fall98
                    grades
                    prog1.r
                    prog2.r
                fall99
                    prog2.r
                    prog1.r
                    grades
```

Postorder

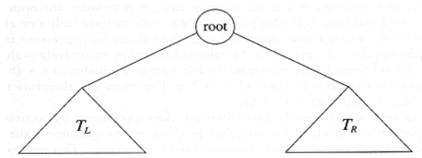
```
                ch1.r            3
                ch2.r            2
                ch3.r            4
            book                10
                        syl.r    1
                    fall98       2
                        syl.r    5
                    spr99        6
                        syl.r    2
                    sum99        3
                cop3530         12
            course              13
            junk                 6
        mark                    30
            junk                 8
        alex                     9
            work                 1
                        grades   3
                        prog1.r  4
                        prog2.r  1
                    fall98       9
                        prog2.r  2
                        prog1.r  7
                        grades   9
                    fall99      19
                cop3212         29
            course              30
        bill                    32
    /usr                        72
```


Binary Trees

CSE 2011

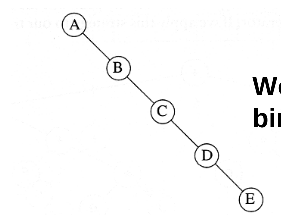
Binary Trees

- A tree in which each node can have at most two children.



Generic binary tree

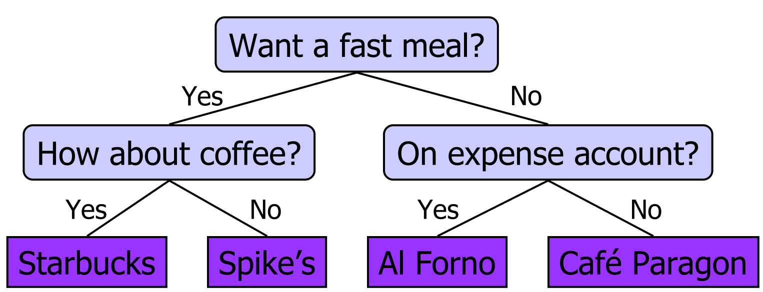
- The depth of an “average” binary tree is considerably smaller than N. In the worst case, the depth can be as large as N – 1.



Worst-case binary tree

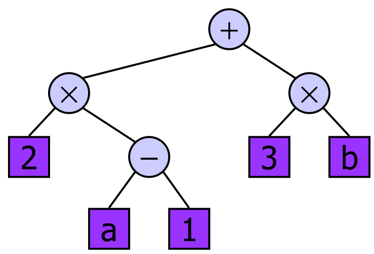
Decision Tree

- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision



Arithmetic Expression Tree

- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands
- Example: arithmetic expression tree for the expression $(2 \times (a - 1) + (3 \times b))$



BinaryTree ADT

- The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT
- Additional methods:
  - position left(p)
  - position right(p)
  - boolean hasLeft(p)
  - boolean hasRight(p)
- Update methods may be defined by data structures implementing the BinaryTree ADT

Implementing Trees

- Arrays?
  - Homework: How can trees be implemented using arrays? Analyze the storage requirements.
- Linked structure?

Linked Structure of Binary Trees

```
class BinaryNode {
    Object          element
    BinaryNode      left;
    BinaryNode      right;
    BinaryNode      parent;
}
```

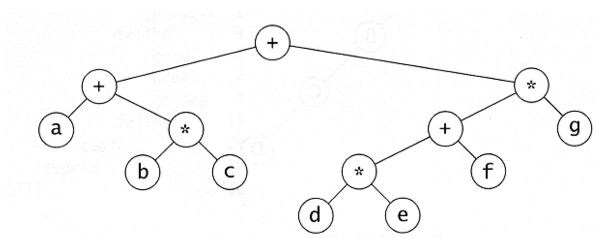
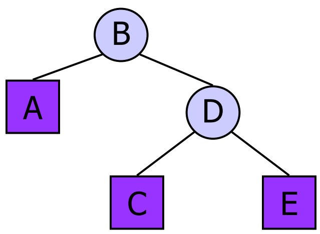
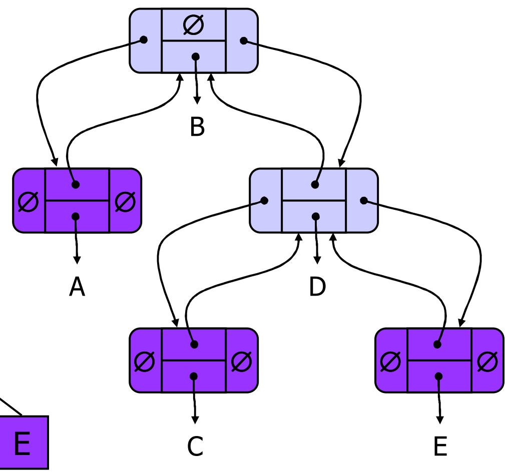


Figure 4.14 Expression tree for (a + b * c) + ((d * e + f) * g)

Linked Structure of Binary Trees (2)

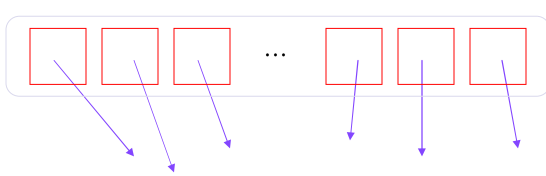
- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node

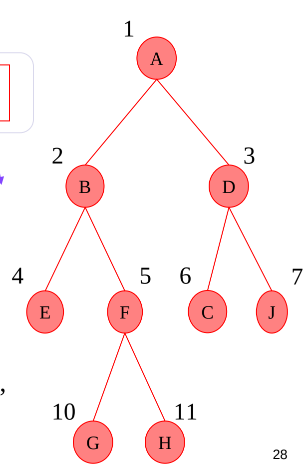




Array-Based Implementation

- Nodes are stored in an array.





- Let rank(node) be defined as follows:
  - rank(root) = 1
  - if node is the left child of parent(node), rank(node) = 2*rank(parent(node))
  - if node is the right child of parent(node), rank(node) = 2*rank(parent(node))+1

Binary Tree Traversal

- Preorder (node, left, right)
- Postorder (left, right, node)
- Inorder (left, node, right)

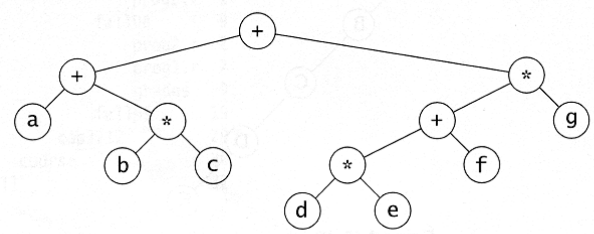


Figure 4.14 Expression tree for (a + b * c) + ((d * e + f) * g)

Preorder Traversal: Example

- Preorder traversal
  - node, left, right
  - prefix expression
    - + + a * b c * + * d e f g

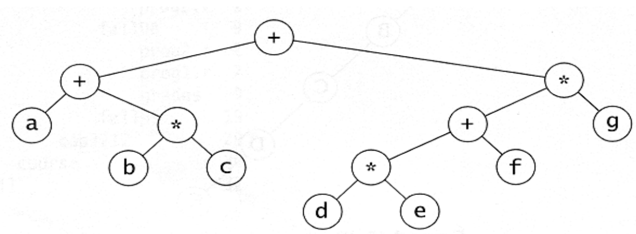


Figure 4.14 Expression tree for (a + b * c) + ((d * e + f) * g)

Postorder Traversal: Example

- Postorder traversal
  - left, right, node
  - postfix expression
    - a b c * + d e * f + g * +

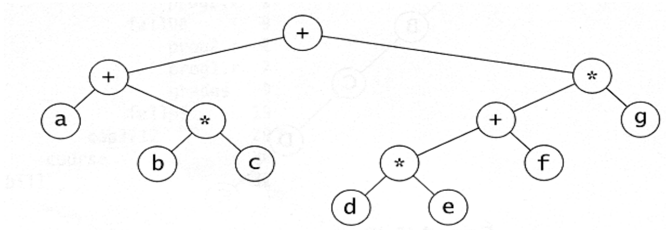


Figure 4.14 Expression tree for (a + b * c) + ((d * e + f) * g)

Inorder Traversal: Example

- Inorder traversal
  - left, node, right
  - infix expression
    - a + b * c + d * e + f * g

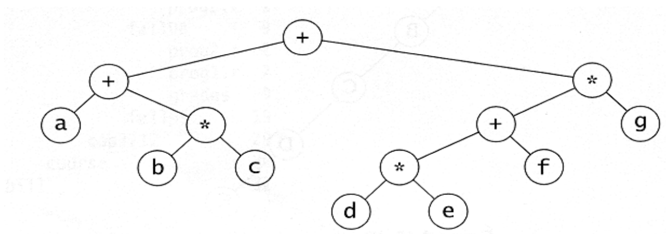


Figure 4.14 Expression tree for (a + b * c) + ((d * e + f) * g)

Pseudo-code for Binary Tree Traversal

Algorithm *Preorder*(x)
Input: x is the root of a subtree.
1. **if** $x \neq$ NULL
2. **then** output key(x);
3. *Preorder*(left(x));
4. *Preorder*(right(x));

Algorithm *Postorder*(x)
Input: x is the root of a subtree.
1. **if** $x \neq$ NULL
2. **then** *Postorder*(left(x));
3. *Postorder*(right(x));
4. output key(x);

Algorithm *Inorder*(x)
Input: x is the root of a subtree.
1. **if** $x \neq$ NULL
2. **then** *Inorder*(left(x));
3. output key(x);
4. *Inorder*(right(x));

Properties of Proper Binary Trees

- A binary trees is ***proper*** if each node has either zero or two children.
- Level: depth
  - The root is at level 0
  - Level d has at most 2^d nodes
- Notation:
  - n number of nodes
  - e number of external (leaf) nodes
  - i number of internal nodes
  - h height

$$n = e + i$$
$$e = i + 1$$
$$h+1 \leq e \leq 2^h$$

$$n = 2e - 1$$
$$h \leq i \leq 2^h - 1$$
$$2h+1 \leq n \leq 2^{h+1} - 1$$

$$\log_2 e \leq h \leq e - 1$$
$$\log_2 (i + 1) \leq h \leq i$$
$$\log_2 (n + 1) - 1 \leq h \leq (n - 1)/2$$

Properties of (General) Binary Trees

- Level: depth

  The root is at level 0

  Level *d* has at most 2^{d} nodes
- Notation:
  - n number of nodes
  - e number of external (leaf) nodes
  - i number of internal nodes
  - h height

$$h+1 \le n \le 2^{h+1} - 1$$

$$1 \le e \le 2^{h}$$

$$h \le i \le 2^{h} - 1$$

$$\log_2 (n+1) - 1 \le h \le n - 1$$

Next time …

- Binary Search Trees