UNIVERSITY of WASHINGTON

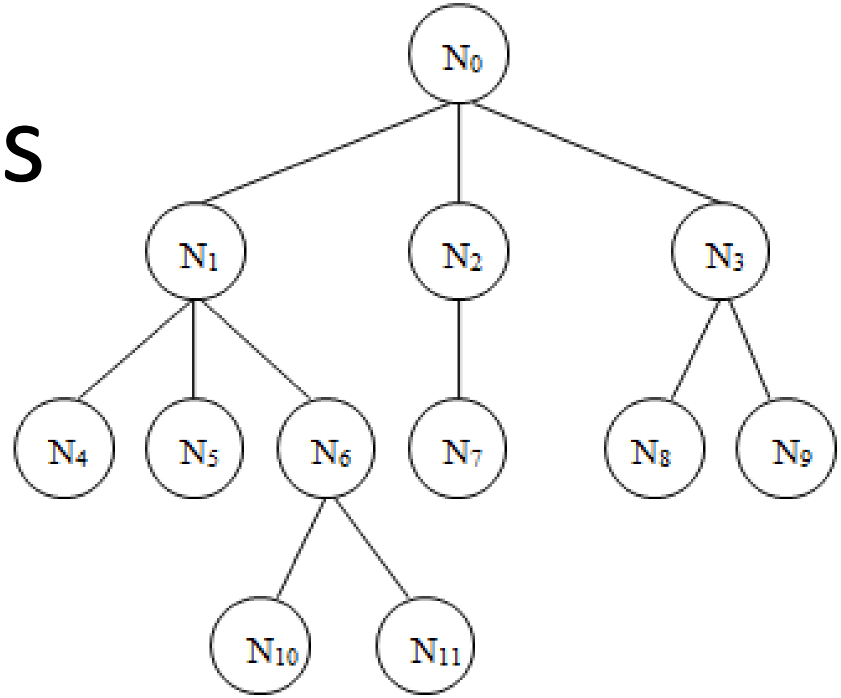
CSS 343: Data Structures, Algorithms, and Discrete Mathematics II

Tree Introduction

Version 1

Wooyoung Kim

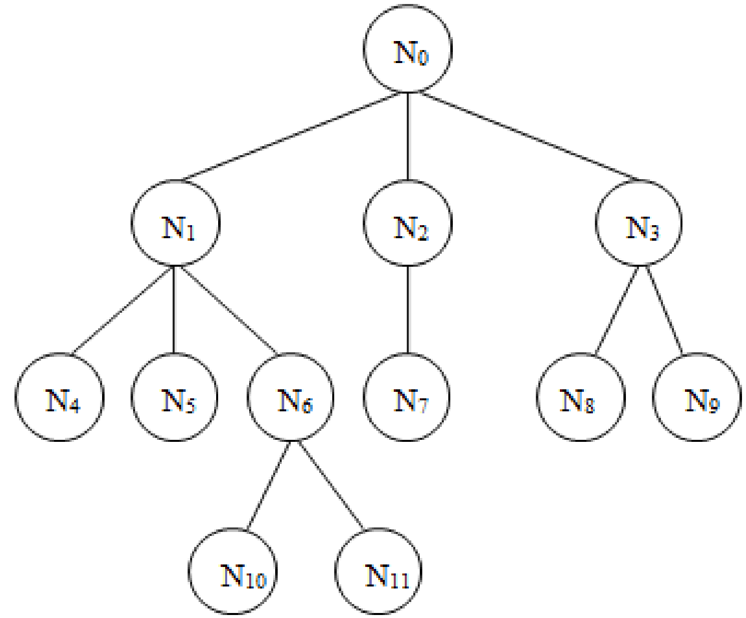
Terms

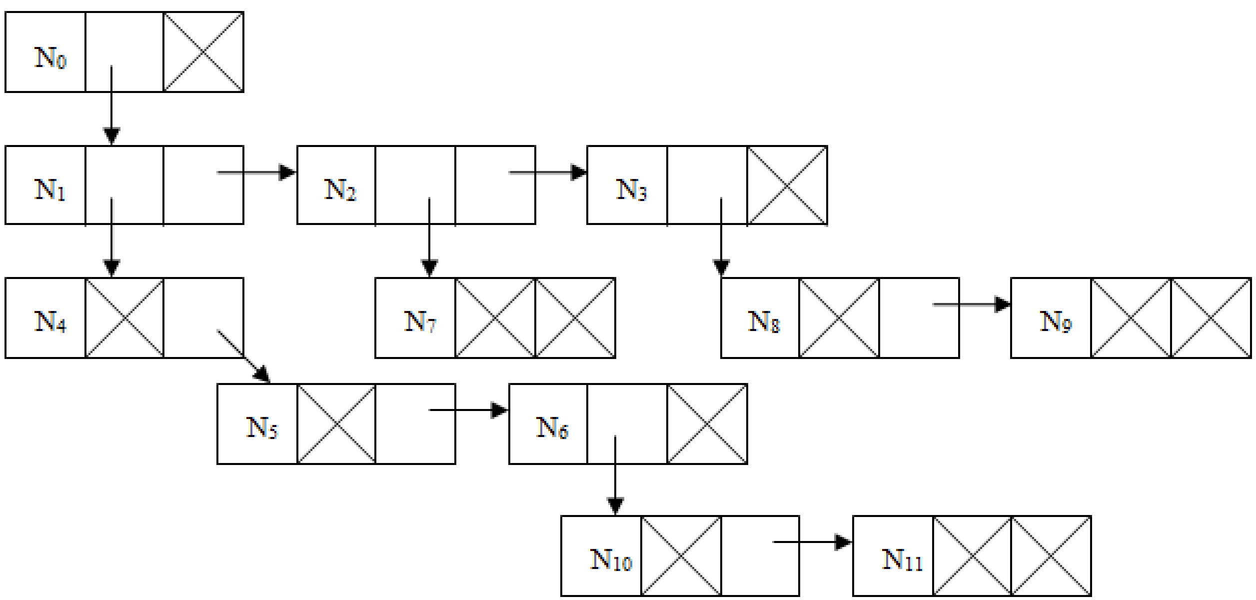


- Node
- Branch (edge)
- Parent, child, siblings
- Ancestor, descendant
- Height (many definitions): maximal level of any node
  - If root as level 0, height of this tree: 3
  - If root as level 1, height of this tree: 4
- Can be defined using a **recursive definition**.
  - T is a tree if:
    - T is empty, or
    - T has a node and zero or more non-empty subtrees (each of which is also a tree.)

General Tree Implementation

- Each node can have **any number of children**
- Use the **first-child, next-sibling** representation
- Any application using general tree?





Binary Tree

- At most 2 children

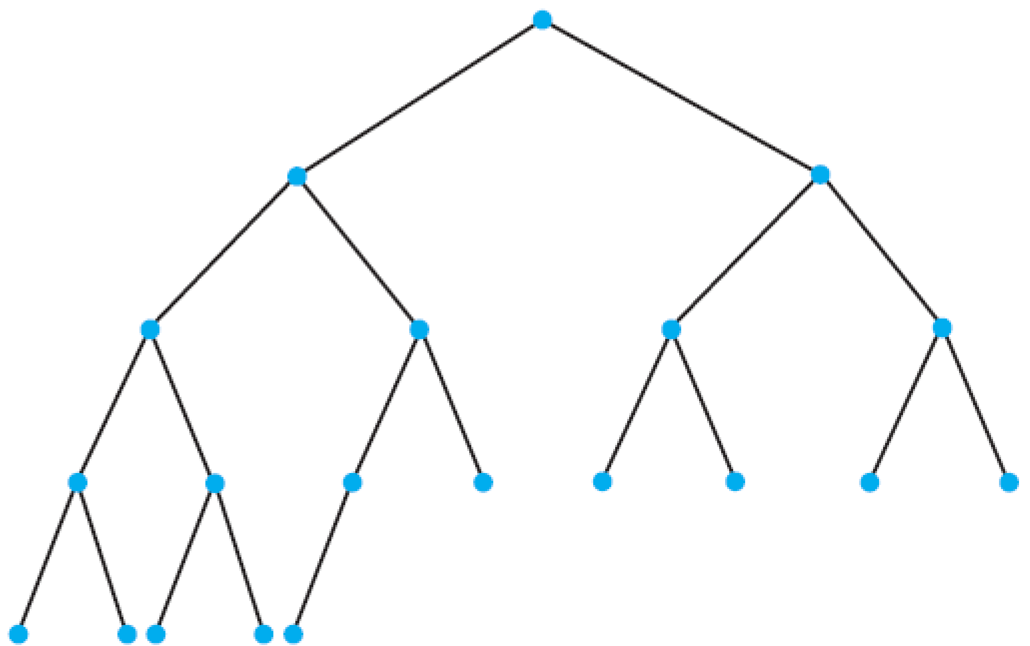
```
struct BinaryTreeNode {
    Object *item;
    BinaryTreeNode *leftChild;
    BinaryTreeNode *rightChild;
};
```

- Many uses
  - Expression tree
  - Huffman coding algorithm
  - Binary search tree
  - etc.
- Will include a binary tree implementation in assignment 2

Full, Complete, and Balanced Binary Trees (1)

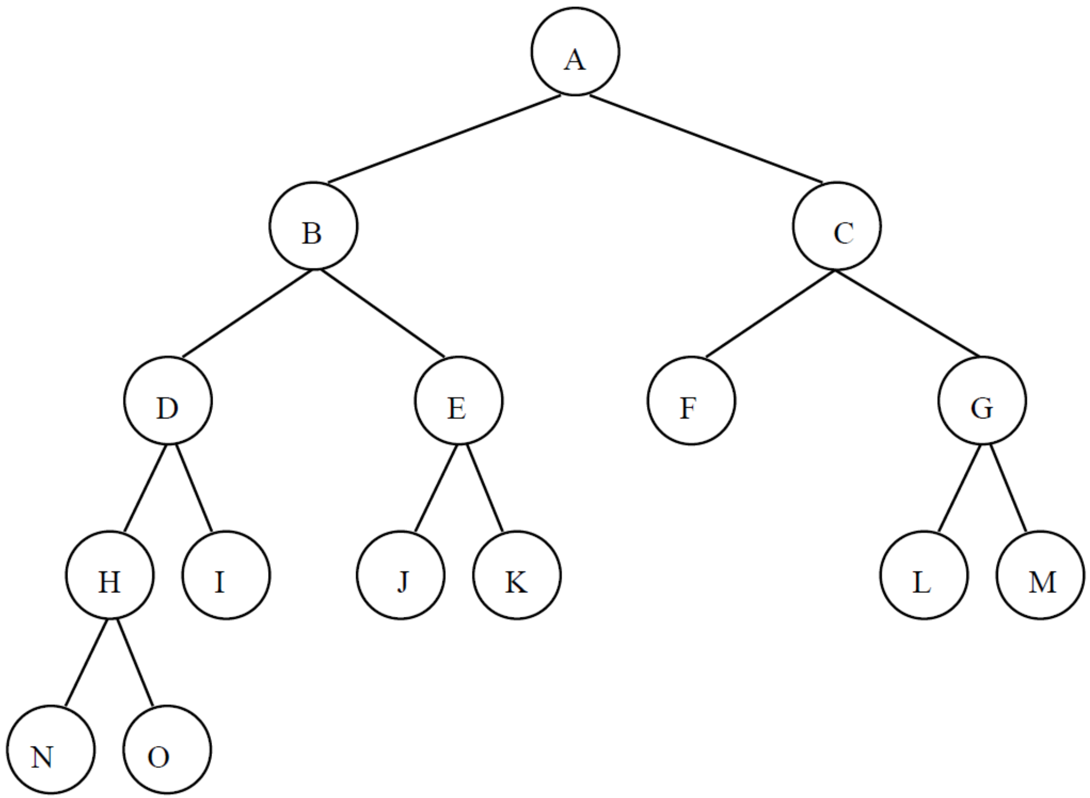
- **Full binary tree** has no nodes with one child; they all have zero or two.
- **Perfect binary tree:** full binary tree where all of the leaves are at the same level
- **Complete binary tree** has every level, except possibly the deepest, is completely filled. The nodes in the last level are as far left as possible
- **Balanced binary tree:** where the left and right subtrees of any node have height difference by at most 1

Full, Complete, and Balanced Binary Trees (2)



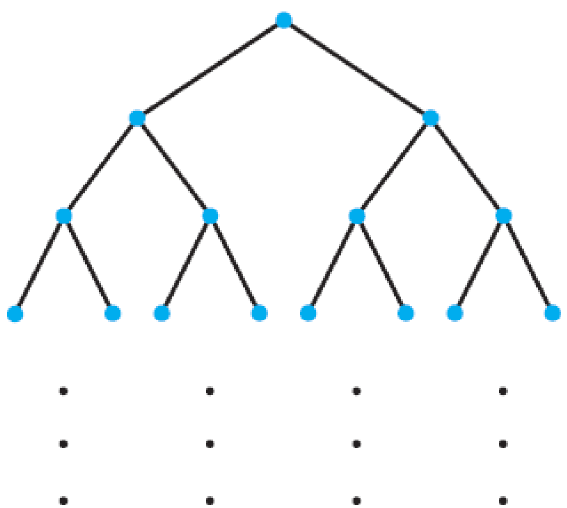
- Is it full? Complete? Balanced?

Full, Complete, and Balanced Binary Trees (3)



- Is it full? Complete? Balanced?

The Maximum and Minimum Heights of a Binary Tree

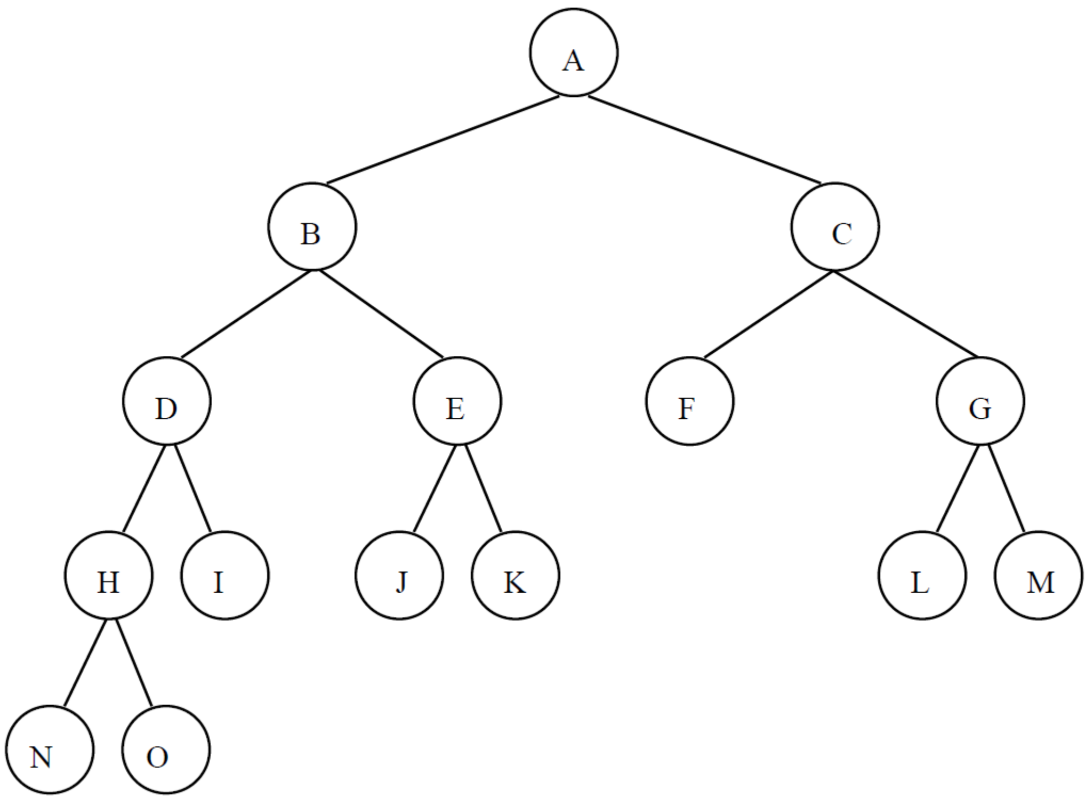


Level	Number of nodes at this level	Total number of nodes at this level and all previous levels
1	$1 = 2^0$	$1 = 2^1 - 1$
2	$2 = 2^1$	$3 = 2^2 - 1$
3	$4 = 2^2$	$7 = 2^3 - 1$
4	$8 = 2^3$	$15 = 2^4 - 1$
⋮	⋮	⋮
h	2^{h-1}	$2^h - 1$

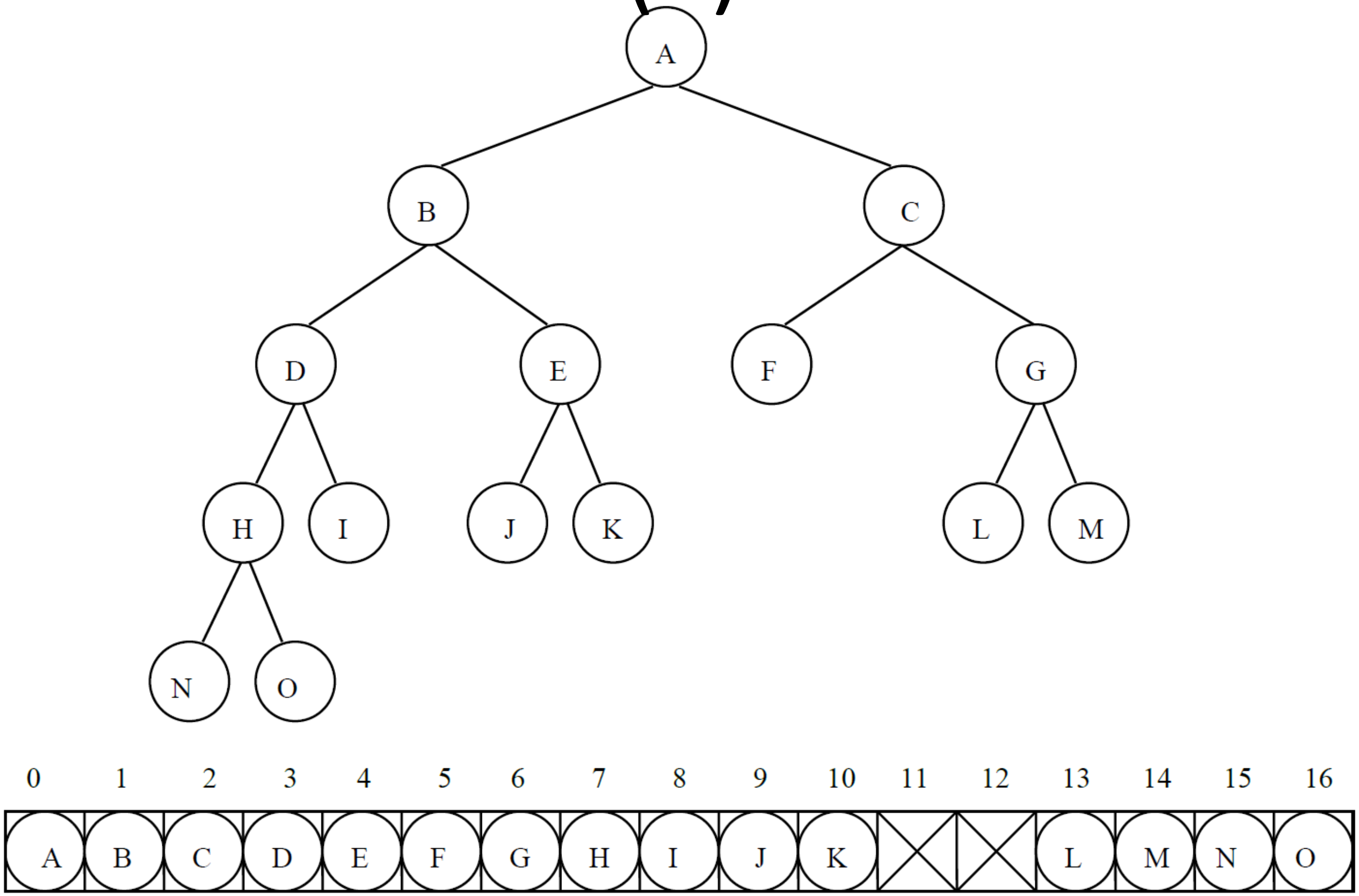
Binary tree – array implementation(1)

Array size of a tree with height h: 2^h

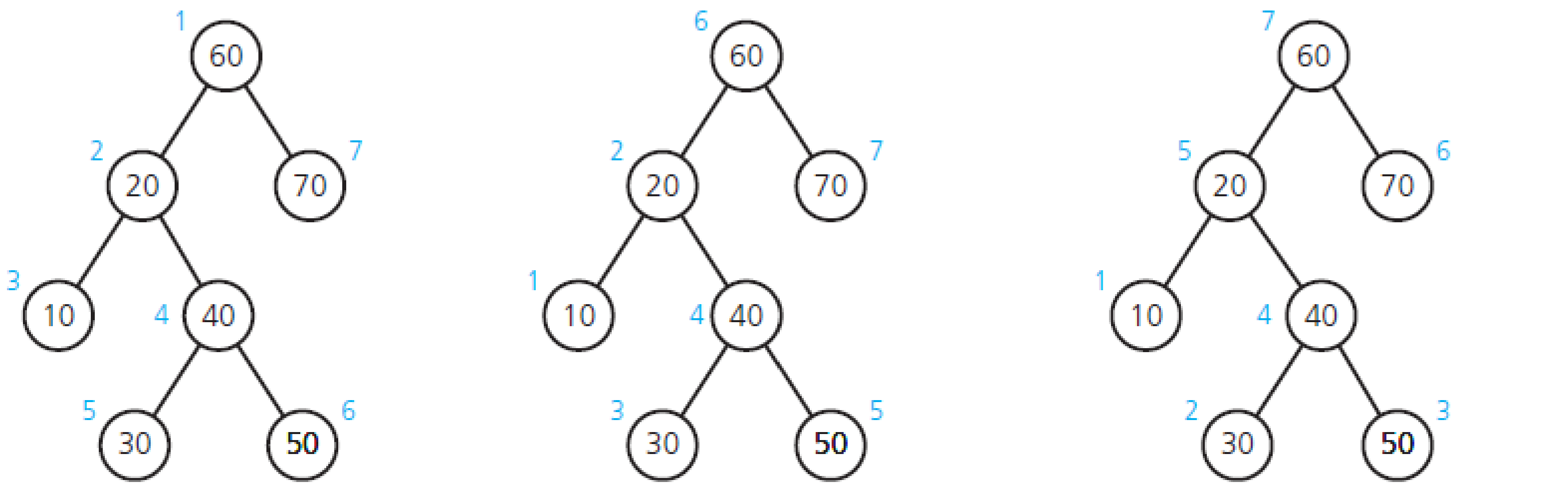
1) Store the root at index 0
2) Children of a node at position *i* at positions *2i+1* and *2i+2*.



Binary tree – array implementation (2)



Traversals of a Binary Tree



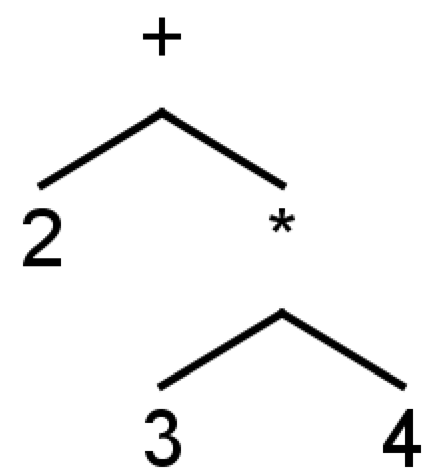
(a) Preorder: 60, 20, 10, 40, 30, 50, 70 (b) Inorder: 10, 20, 30, 40, 50, 60, 70 (c) Postorder: 10, 30, 50, 40, 20, 70, 60

(Numbers beside nodes indicate traversal order.)

- Can you implement?

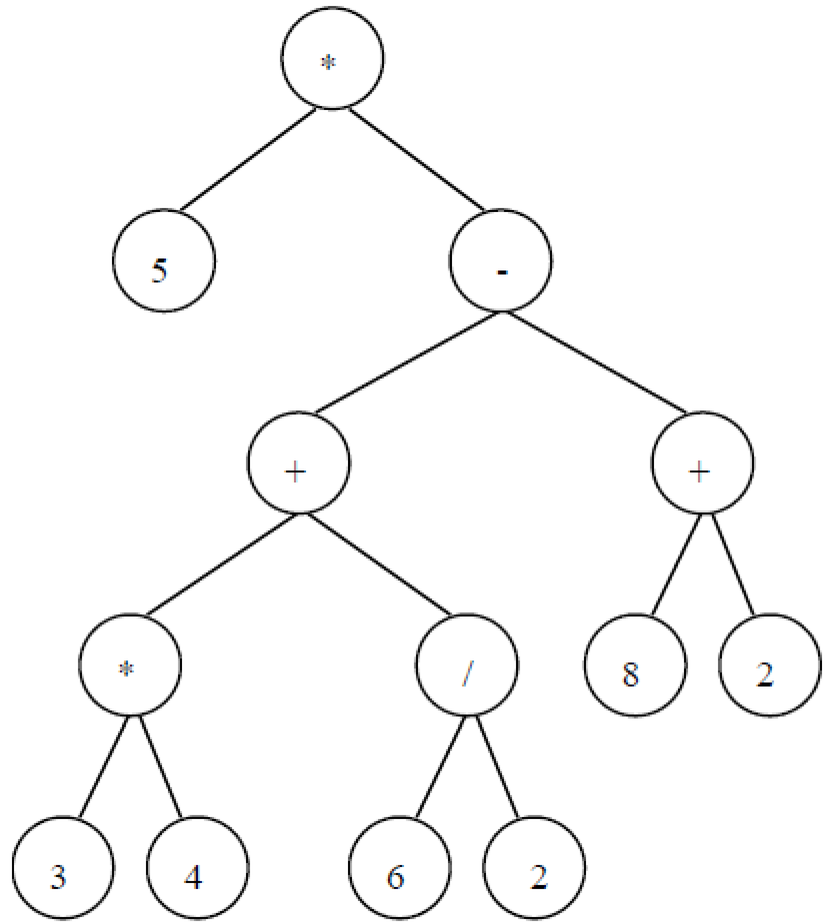
Expression tree

- Solving binary (mathematical) expression
  – Binary (unary) operator: internal node
  – Operands: leaf node
  – E.g., compiler does it
- Example
  – 2 + 3 * 4
  – How about (2+3) * 4
    - Idea? Substitute: X*4
  – How about - Practice
    - 5 * ((3 * 4) + (6 / 2) – (8 + 2))



Expression tree

- Prefix:
- Postfix:
- Infix:



Exercise Expression tree

$$A - (B + C * D) / E$$

1. Build an expression tree

2. Prefix expression

3. Postfix expression

4. Infix expression

Evaluate Postfix

A – (B + C * D) / E = A B C D * + E / -
Evaluate it when A=8, B=7, C=4, D=5, E = 9

Infix: 8-(7+4*5)/9 = 5

PostFix: 8 7 4 5 * + 9 / -

c	Operation	stack
8		
7		
4		
5		
*		
+		
9		
/		
-		

Evaluate Postfix

A – (B + C * D) / E = A B C D * + E / -
Evaluate it when A=8, B=7, C=4, D=5, E = 9

Infix: 8-(7+4*5)/9 = 5

PostFix: 8 7 4 5 * + 9 / -

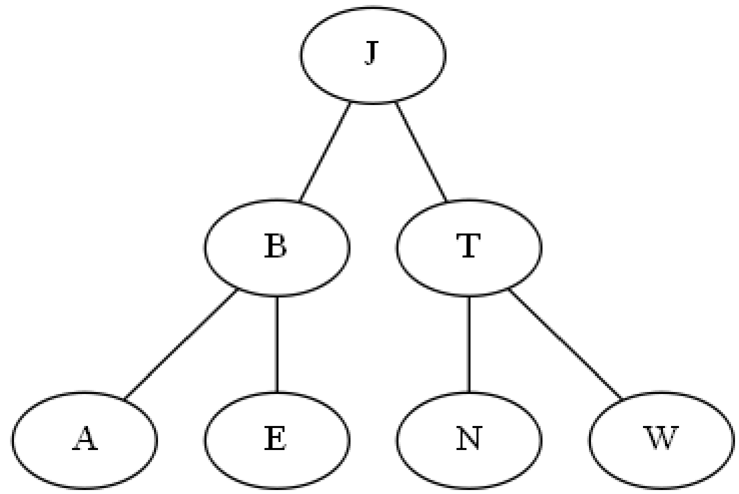
c	Operation	stack
8	Push	8
7	Push	8 7
4	Push	8 7 4
5	Push	8 7 4 5
*	Pop 5 and 4, push 4*5	8 7 20
+	Pop 20, 7, push 7+20	8 27
9	Push	8 27 9
/	Pop 9, 27, push 27/9	8 3
-	Pop 3, 8, push 8-3 = 5	5

Exercise

- Carrano, Chapter 15:

  Exercises: #22

```
Put the root of the tree in C
while (C is not empty){
      Remove a node N from C
      Visit N
      if (N has a left child)
          put the child in C
      if (N has a right child)
          put the child in C
      }
```



Algorithm 1: Remove the newest node from C

Algorithm 2: Remove the oldest node from C

Question: What is the order of visits from each algorithm?

Hint: Use C and Visit as separate storages

More Exercise

- Carrano, Chapter 15:
  Exercises: #4, #22

- Write a method to traverse a tree (inorder, preorder, postorder)
- Write a method to sum the elements in a binary tree
- Write a method to count the number of nodes with exactly one child
- Trace "countLeaves" in lecture note 1, using box-tracing method

Reminder

- Program 1 due by July 4^{th} 11:59 pm
  - posted Valgrind info on Canvas (inside Files/additionalMaterials/)
  - Compile & run as before (just need to use valgrind step to verify if there's memory leak)
  - Points will be deducted for memory leak
  - Points will be deducted if your program cannot be run on Linux but in other platform

Huffman coding

- Used in compression (e.g., text document)
- Observations
  - Some characters (e.g., ‘e’, ‘o’,) appear more often than others (‘q’, ‘z’)
- Ideas?
  - Fewer bits for symbols appearing frequently
  - Possibility of ambiguity if you don’t do it right
  - So Huffman coding to the rescue!

https://www.techiedelight.com/huffman-coding/

https://www.cs.usfca.edu/~galles/visualization/java/download.html (visualization)

Algorithm (one version)

Each letter is a small tree with a single node (and has an associated weight - the frequency)

Repeat until all nodes form a single tree (normally put smaller on left):

- Select the two trees with the smallest weights.
- Merge these trees into a new tree by adding a node that is the parent of both.
- The weight of the new tree is the sum of the weights of the two previous trees.
- Assign a *0* to one branch (normally left) of the tree and a *1* to the other branch (normally right) of the tree.

E.g. e (22); a (20); r (15); s (18); p (19)

- Document (input): eeeeeeeeeeeeeeeeeeeeeeaaaaaaaaaaaaaaaaaaaarrr rrrrrrrrrrrrsssssssssssssssssspppppppppppppppppppp

Get frequencies of each letter

- e (22); a (20); r (15); s (18); p (19)

Tree:

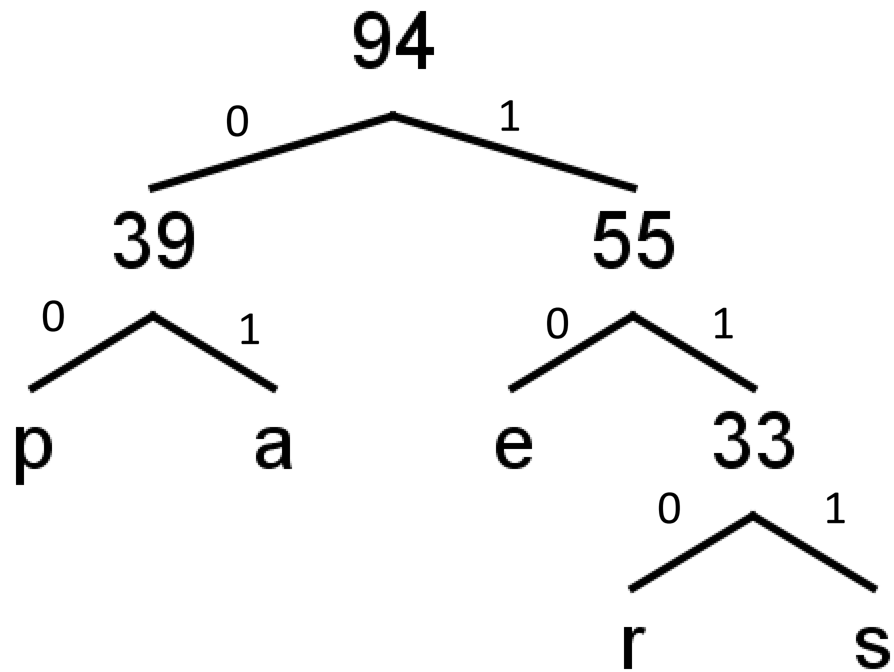
Strategy:

1. Sort
2. Remove two letters of smallest frequency, put them as children. Add new letter (combined) with the sum of frequencies
3. Repeat 1 & 2 until all letters are removed

Note: When put them in a tree, put small frequency as a left child

- e (22); a (20); r (15); s (18); p (19)

Tree:



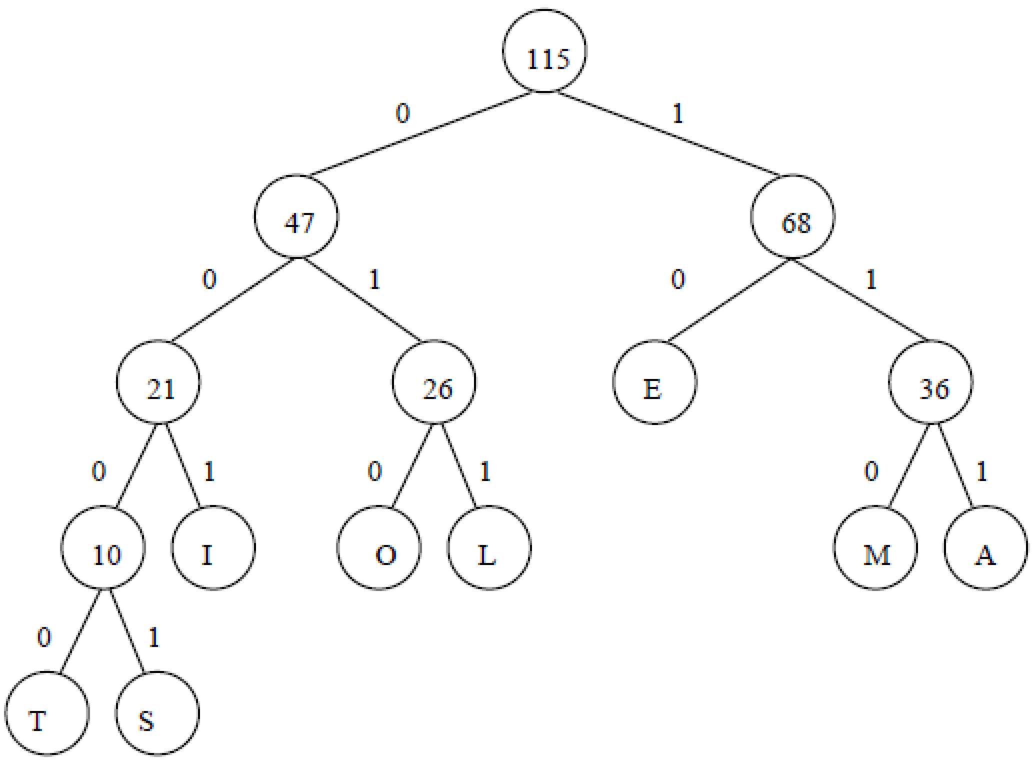
What is 1111001

Exercise

- A (20); E (32); L (14); I (11); M (16); O (12); S (7); T (3) – draw tree and then list code for each character

Exercise

- A (20); E (32); L (14); I (11); M (16); O (12); S (7); T (3) – draw tree and then list code for each character



Huffman encoding efficiency

Take home question.

Hint: The time will depend on what data structure you will use when implementing the algorithm

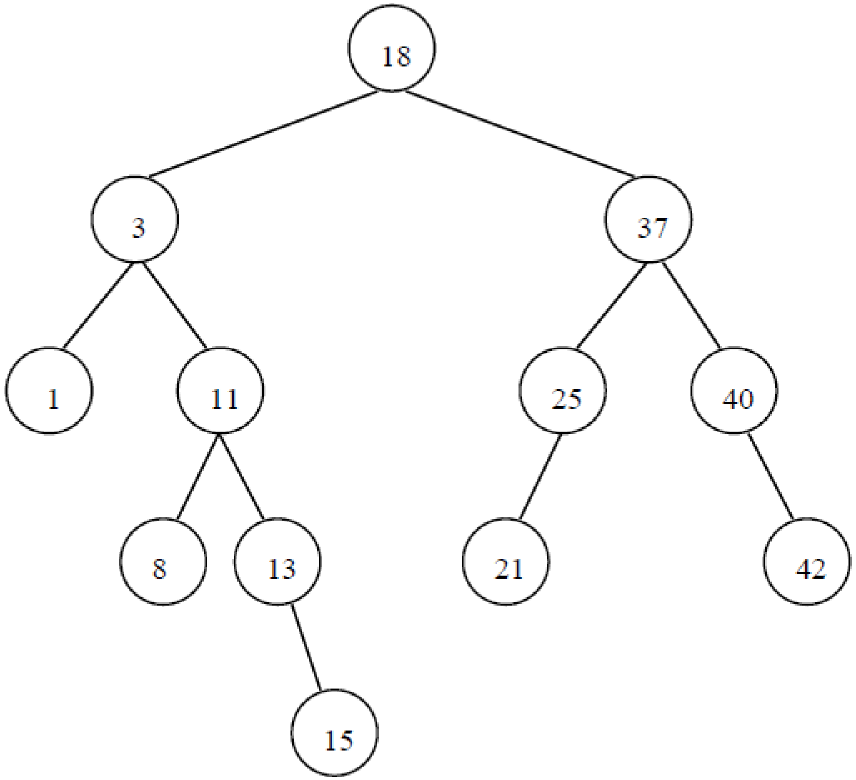
Binary Search Tree

- Ordered collection of items
  - Greater than all of the values in its left subtree and less than all of the values in its right subtree,
  - equal can be put on the left or right or discarded
- Build a binary search tree
  - 18 3 37 11 25 21 40 8 13 1 42 15

https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

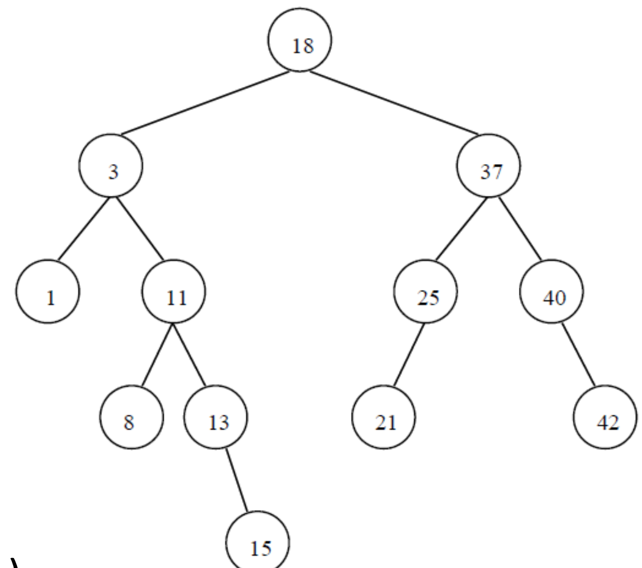
Binary Search Tree

- Ordered collection of items
  - Greater than all of the values in its left subtree and less than all of the values in its right subtree,
  - equal can be put on the left or right or discarded
- Build a binary search tree
  - 18 3 37 11 25 21 40 8 13 1 42 15



Programming a BST – Retrieve, Insert

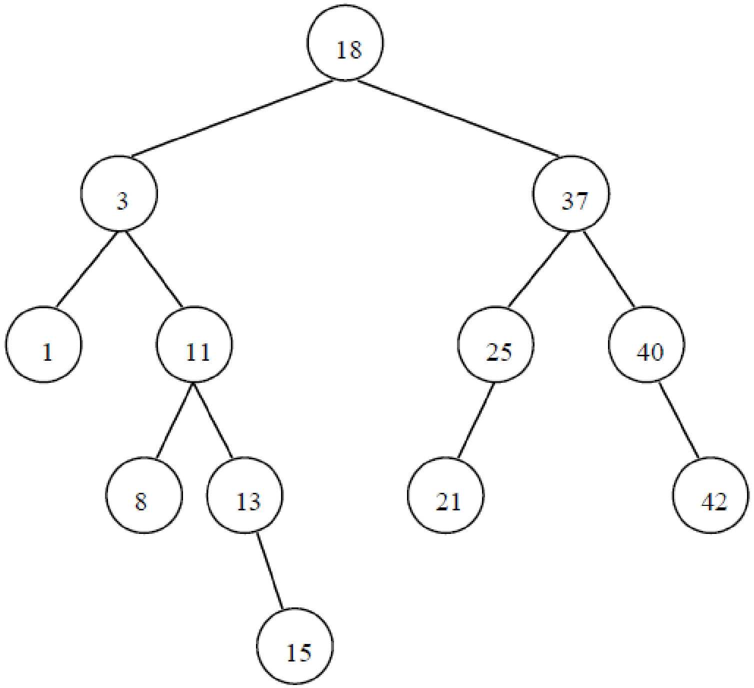
```
const Object * retrieve(BinaryTreeNode *root, const Object &key) {
    if (root == NULL) return NULL;
    else if (key == *root->item) return root->item;
    else if (key < *root->item) return retrieve(root->leftChild, key);
    else return retrieve(root->rightChild, key); }

void insert(BinaryTreeNode *&root, Object *item) {
    if (root == NULL) {
        root = new BinaryTreeNode;
        root->item = item;
        root->leftChild = NULL;
        root->rightChild = NULL; }
    else if (*item < *root->item) insert(root->leftChild, item);
    else insert(root->rightChild, item); }
```



Assumption: drop duplicate values

How about delete p – group practice

- First find, then delete
- Possible node position
  - Leaf, no children (e.g., 1)
  - Only left child (e.g., 25)
  - Only right child (e.g., 13)
  - Two children - A little trickier (e.g., 18)
    - Need to find a replacement for it
    - Either largest descendant of the left child
    - OR smallest descendant of the right child (USE THIS in your practice)



How about delete p – group practice

- write pseudo-code

```
bool deleteNode(BinaryTreeNode *&root, const TreeData &item)
{
        if (root == NULL)  return false;
        else if (item == *root->item) {
                deleteRoot(root);
                return true;
        }
        else if (item < *root->item)  return deleteNode(root->leftChild, item);
        else return deleteNode(root->rightChild, item);
}
```

BST efficiency

1. If not a balanced tree with n nodes
   - Insertion:
   - Deletion:
   - Retrieval:

2. If complete and balanced tree with n nodes
   - Insertion:
   - Deletion:
   - Retrieval:

BST efficiency

1. If not a balanced tree with n nodes
   - Insertion:
   - Deletion:
   - Retrieval:

O(n)

2. If complete and balanced tree with n nodes
   - Insertion:
   - Deletion:
   - Retrieval:

O(logn)