TREES

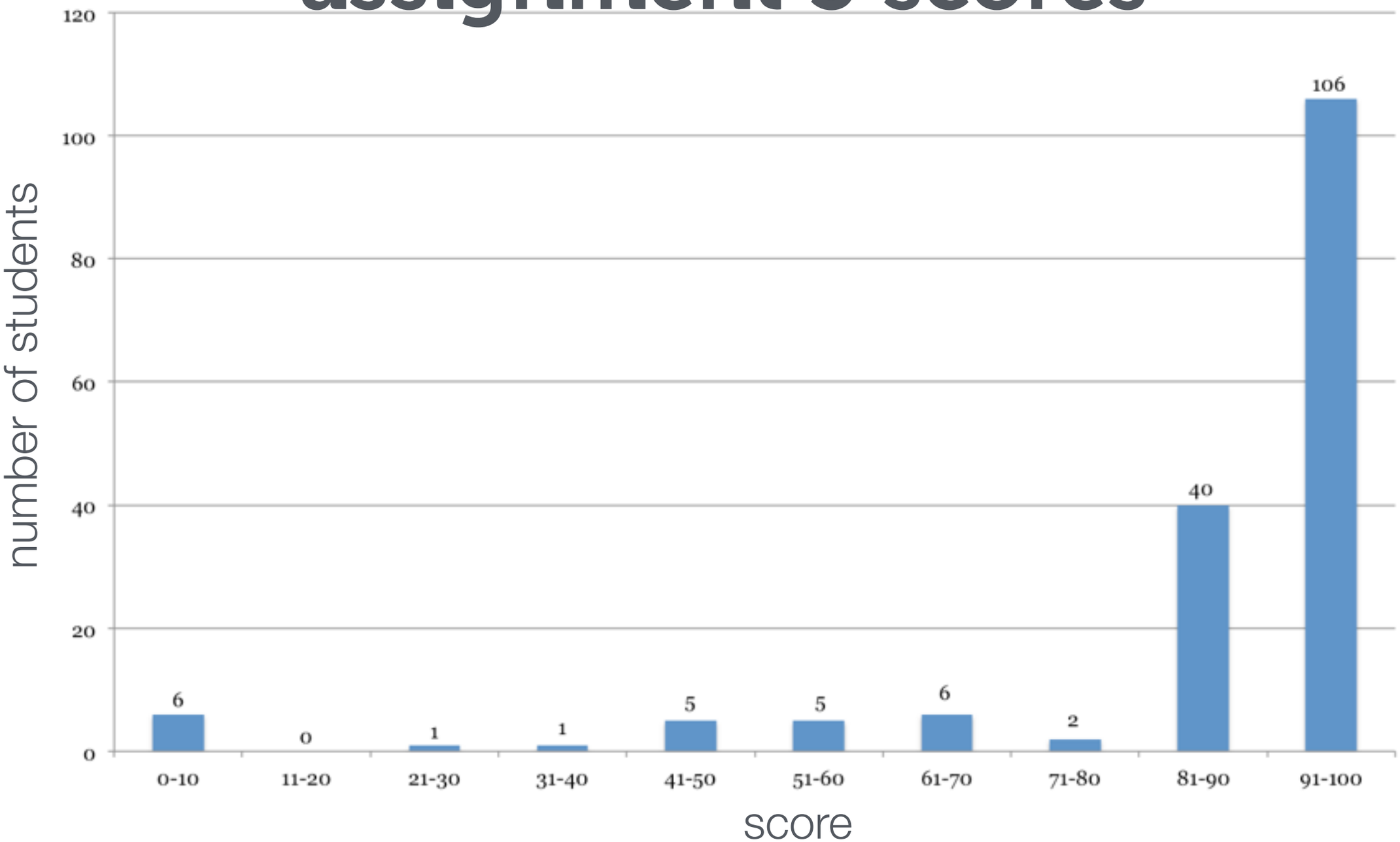
cs2420 | Introduction

administrivia…

-assignment 7 due Thursday at midnight

-asking for regrades through assignment 5 and midterm must be complete by Friday

assignment 5 scores



last time…

-a **queue** is a FIRST-IN, FIRST-OUT data structure

-FIFO

-insert on the back, remove from the front

-operations:

-*enqueue*… adds an item to the back of the queue

-*dequeue*… removes and returns the item at the front

-like a stack, all operations are **O(1)**

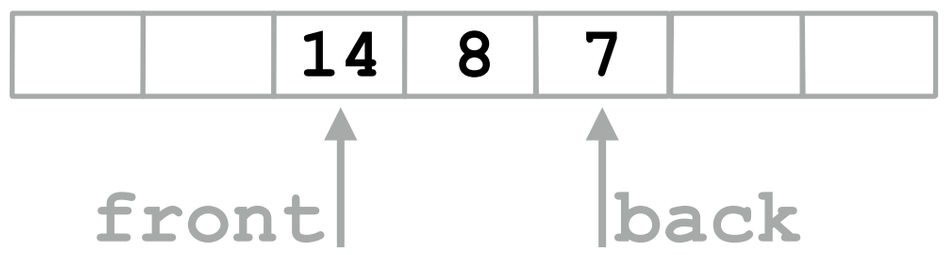
as an array…

-keep track of `front` and `back` indices

-`front` and `back` advance through the array

  -*enqueueing* advances `back`

  -*dequeueing* advance `front`



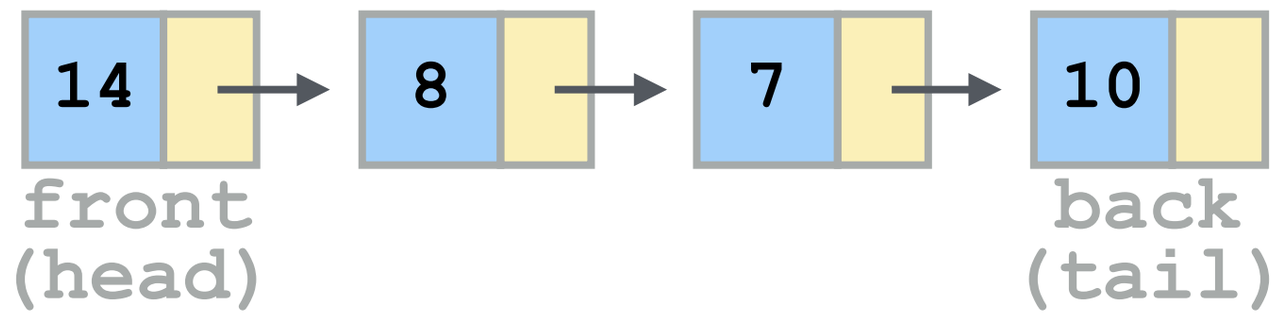
-what happens when `back` reaches the end of the array?

as a linked list…

-remember, inserting and deleting to the head and tail of a linked list is automatically **O(1)**

-`front` is analogous to `head`
-`back` is analogous to `tail`

-no messy wrap-around, or growth issues



-which linked list operations are analogous to *enqueue* and *dequeue*?

summary

-linked lists and wrap-around arrays are both **O(1)** for queue implementations

-BUT, arrays are much more complicated to code

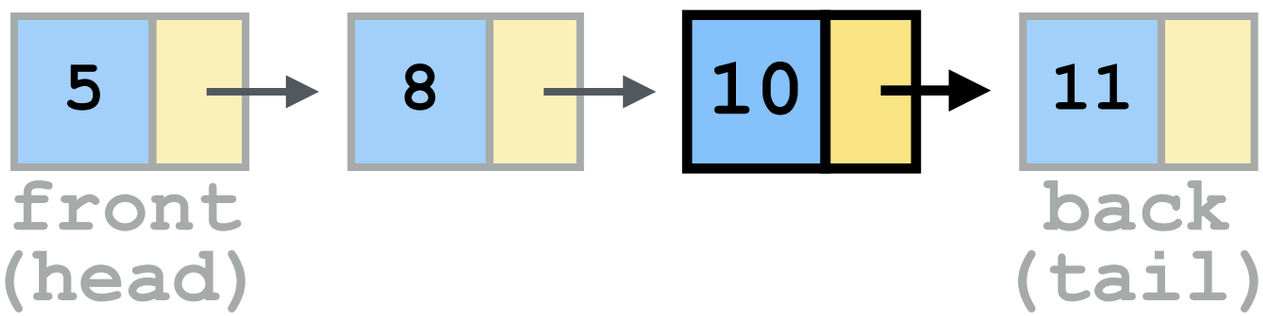
-both queues and stacks require very little code on top of a good linked list implementation

priority queues

using a linked list…

-always add items in correct, sorted spot

`enqueue(10)`



-dequeue will return smallest item **O(1)**

-what is the cost of *enqueue*?

-we will study a more advanced priority queue later…

today…

-trees

-terminology

-binary trees

-traversing a tree

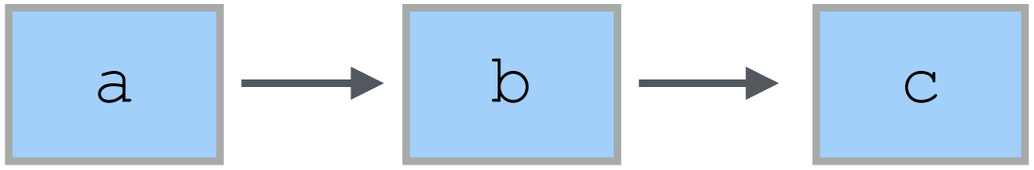
-EXAMPLE: expression trees

-DOT format

trees

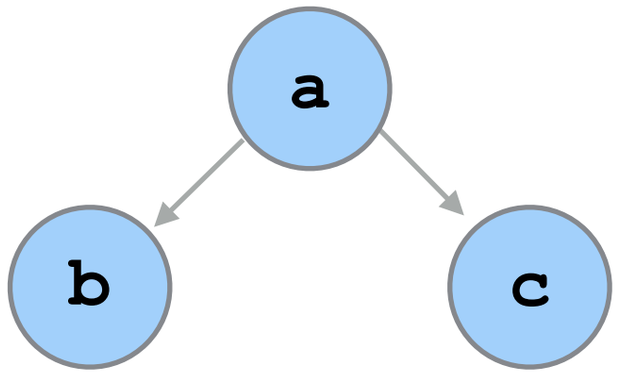
-**trees** are a linked data structure with a hierarchical formation

-recall that a linked list has a reference to a next (and sometimes previous) node



-trees can have multiple links, called branches

THERE ARE MULTIPLE DIRECTIONS
YOU CAN TAKE AT ANY GIVEN NODE



-trees have a **hierarchical structure**

-meaning, any node is a subtree of some larger tree

-except the very top node!

-in CS, trees are usually represented with the root at the top

-trees are recursive in nature

-any given node is itself a tree

-a tree consists of:

-a data element…

-…and more subtrees

-there is a strict parent-to-child relationship among nodes

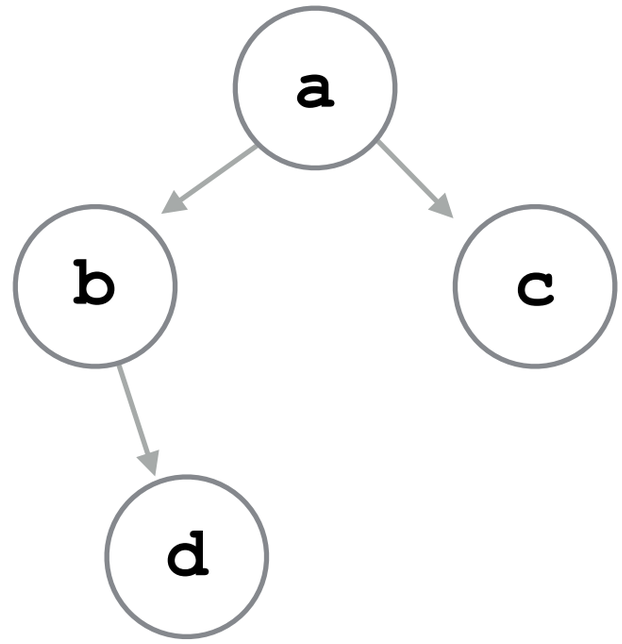
-links *only* go from parent to child

-not from child to parent

-not from sibling to sibling

-every node has exactly one parent, except for the **root**, which has none

-there is exactly one path from the root to any other node



terminology

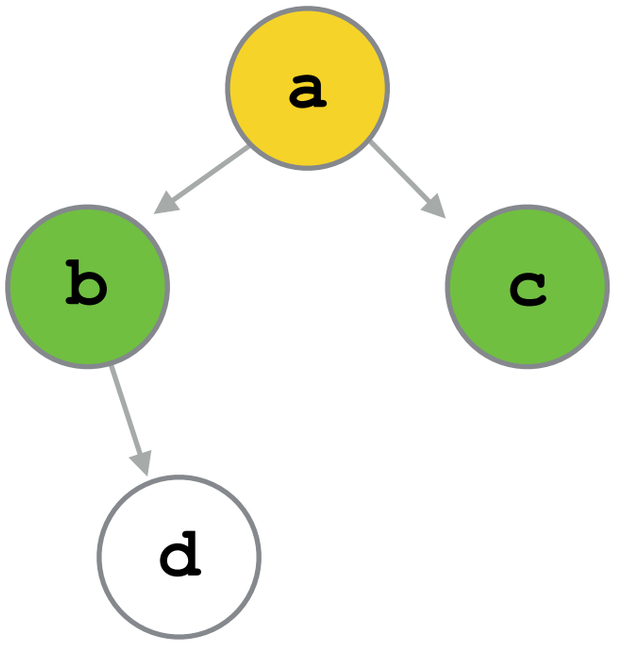
-**root node:** the single node in a tree that has no parents

-**parent:** a node's parent has a direct reference to it

-nodes have AT MOST one parent

-**child:** a node `B` is a child of node `A` if `A` has a direct reference to `B`

-**sibling:** two nodes are siblings if they have the same parent

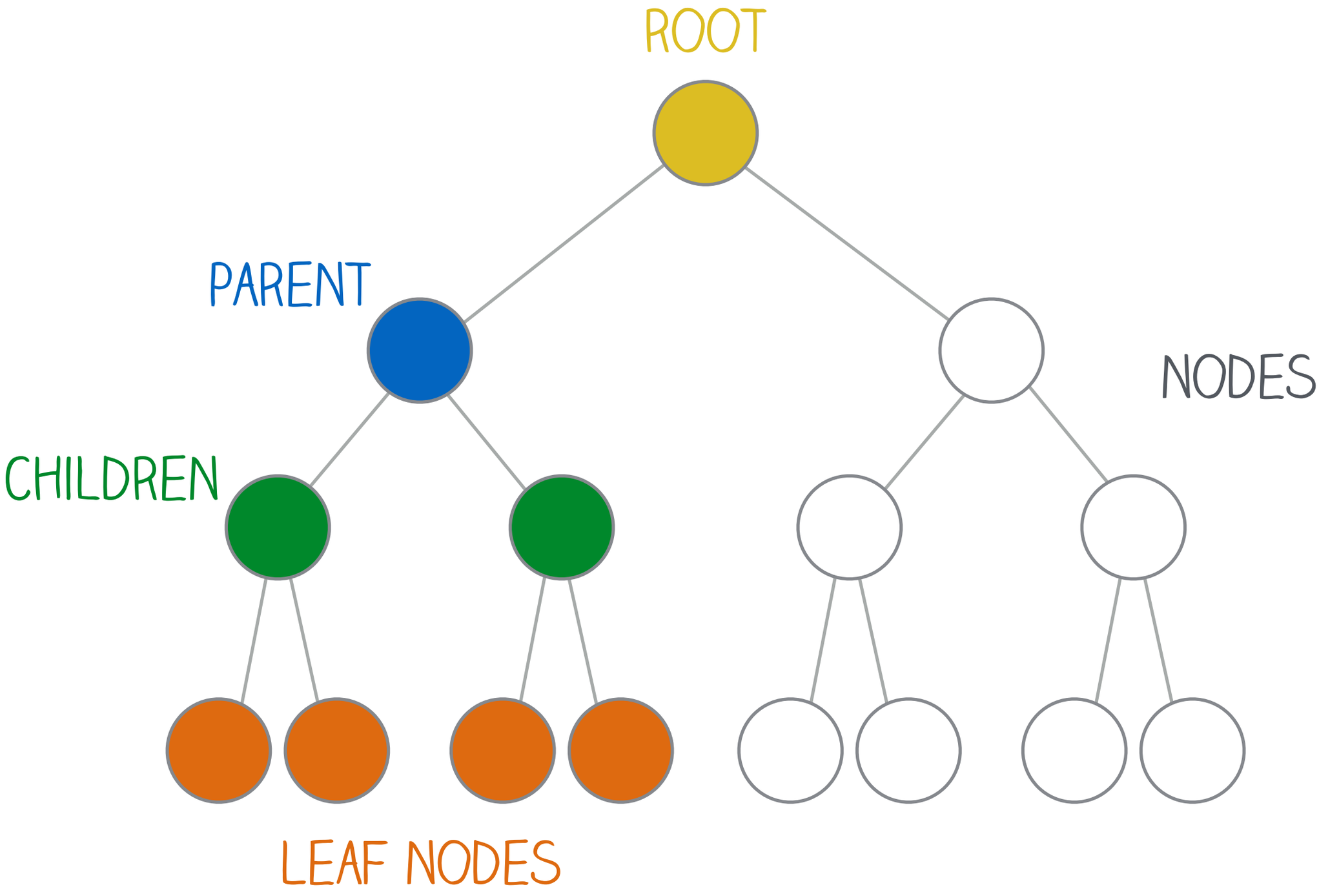


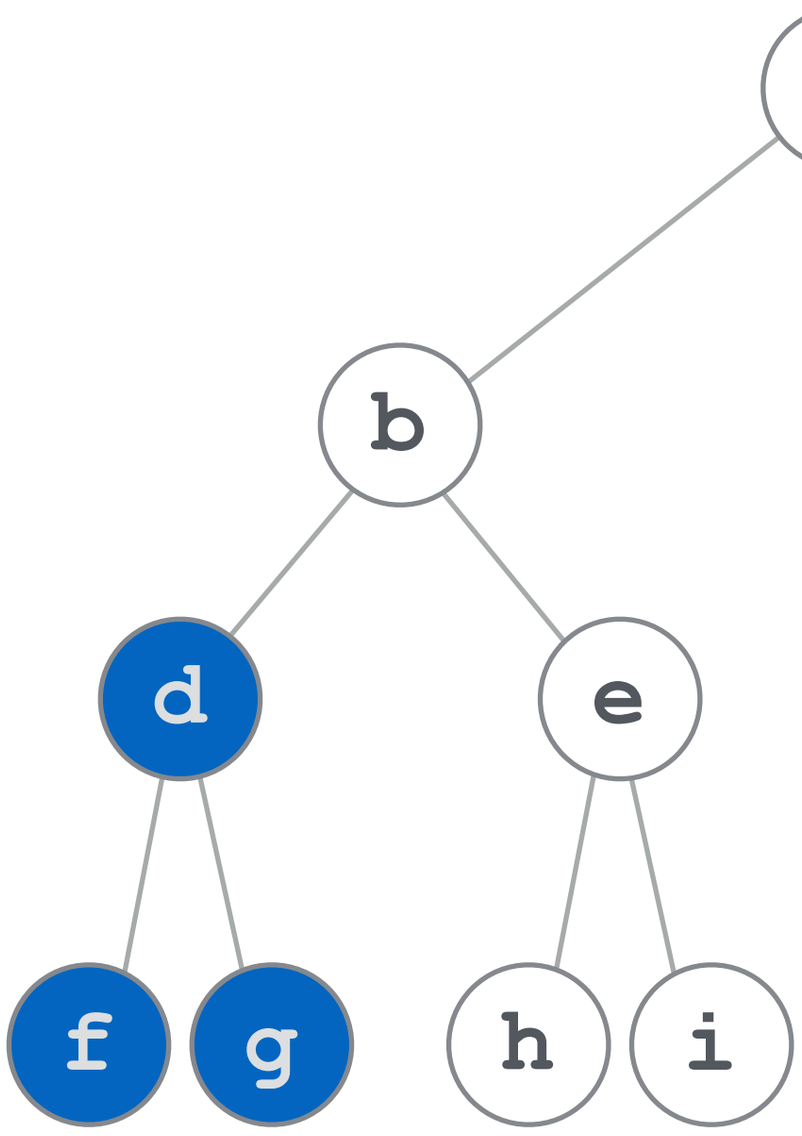
-**leaf node:** a node with no children

-**inner node:** a node with at least one child

-**depth:** the number of ancestors a node has
  -ie. how many steps to the root
  -children are exactly one level deeper than their parents
  -a root node has depth 0

-**height:** the depth of a tree's deepest leaf node


ROOT
PARENT
NODES
CHILDREN
LEAF NODES



SUBTREE ROOTED AT NODE **c**
(LEAF NODES ARE TREES TOO!)

SUBTREE ROOTED AT NODE **d**

example

The root is ___.

The height is ___.

The parent of **`v3`** is ___.

The depth of **`v3`** is ___.

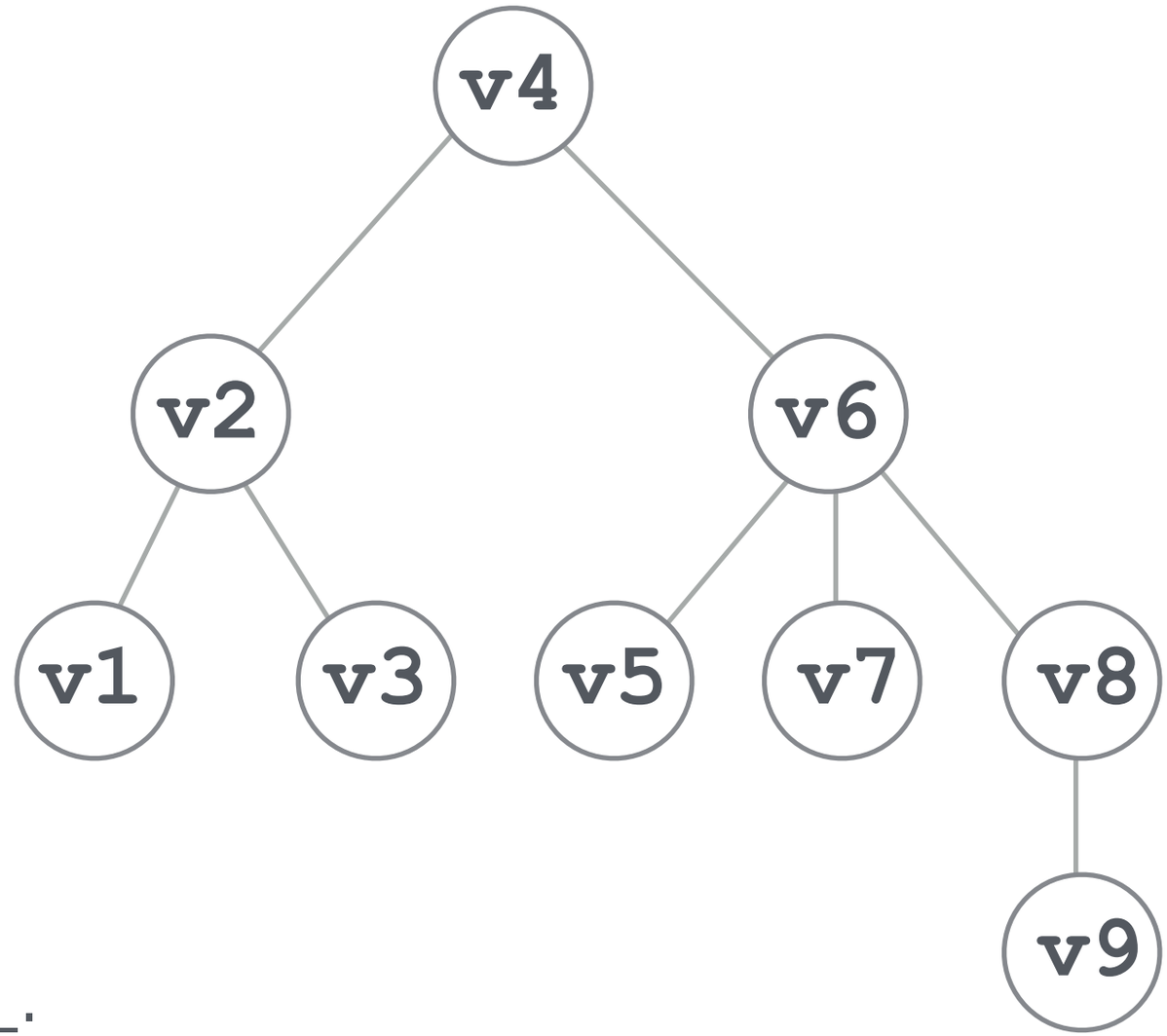
The children of **`v6`** are ___.

The ancestors of **`v1`** are ___.

The descendants of **`v6`** are ___.

The leaves are ___.

Every node other than ___ is the root of a subtree.



binary trees

-**binary trees** are a special case of a tree in which a node can have AT MOST two children

-these nodes are designated *left* and *right*

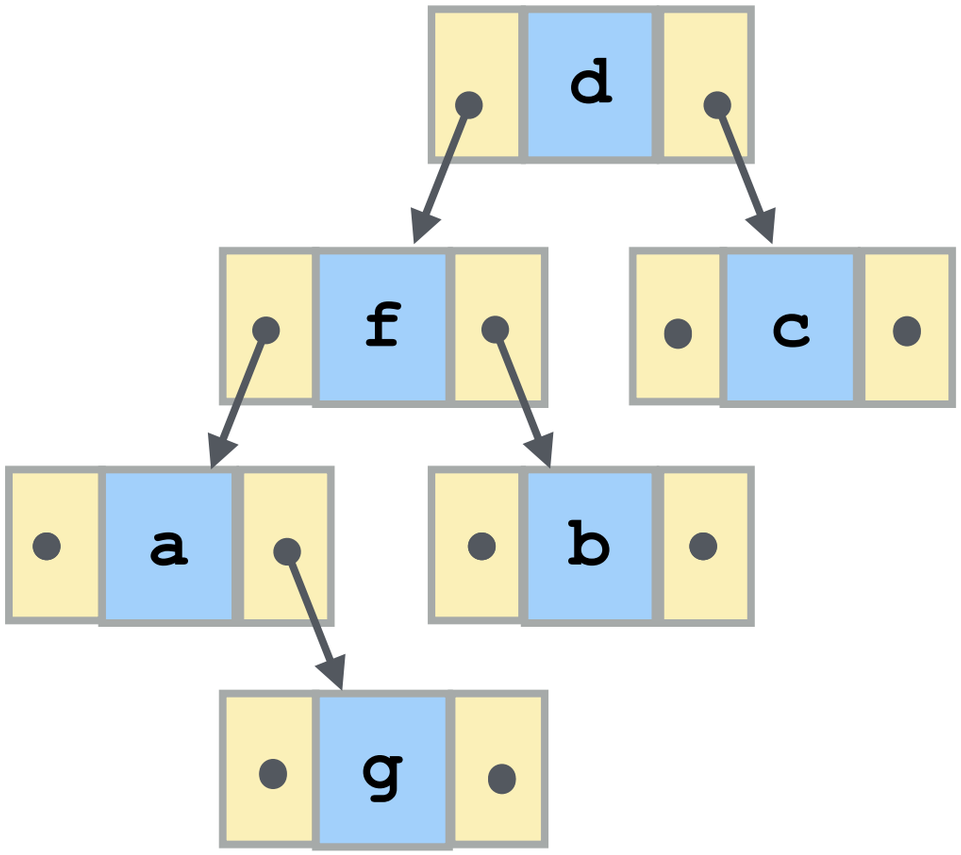
-in this class we will mostly concentrate on binary trees

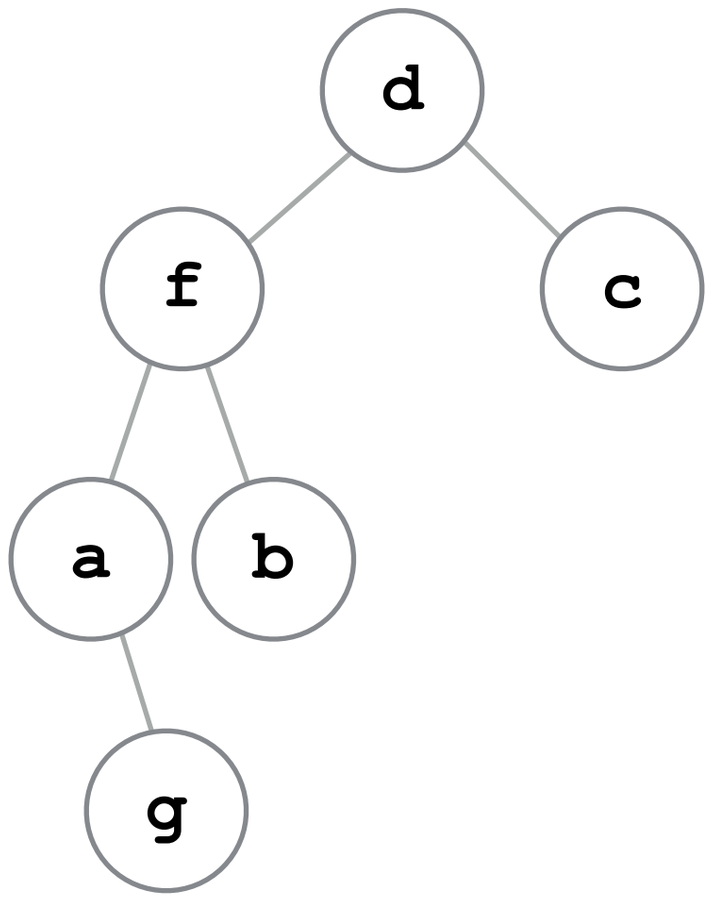
WHAT SHOULD THE IMPLEMENTATION OF A BINARY TREE LOOK LIKE?
WHAT ABOUT A BINARY TREE NODE?

-each node has two reference variables

-one for each of the two children

-if there is no child, the reference is set to `null`





```
class BinaryNode<E>
{
  E data;
  BinaryNode left;
  BinaryNode right;
}
```

-what are the values of `left` and `right` for a leaf node?

-this is the just the `Node` class!

-the `BinaryTree` class would contain what?

traversing a tree

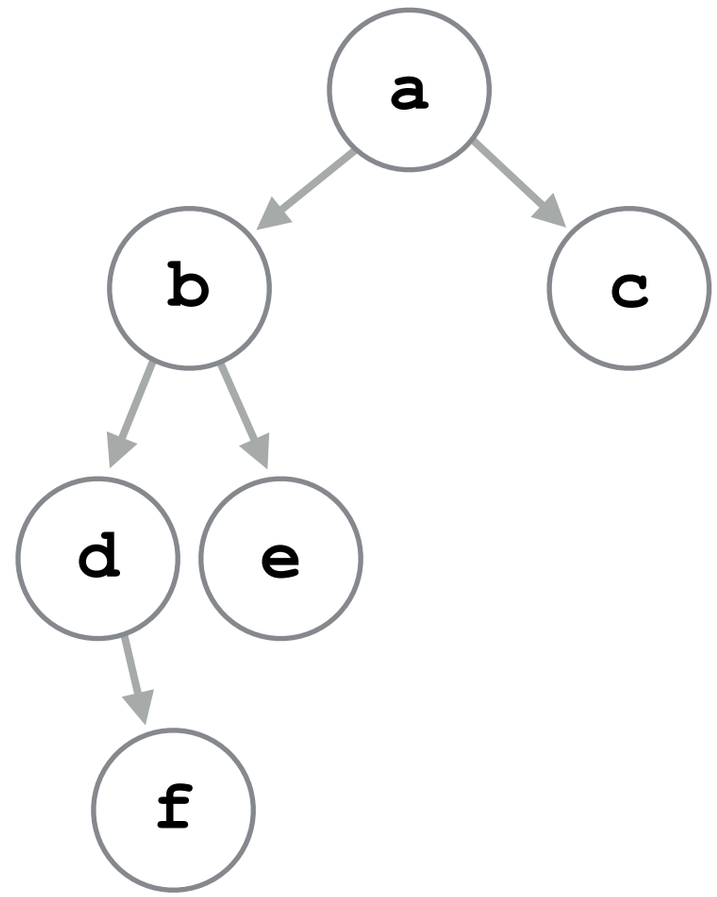
-traversing a *linked list* is simple

-there is only one way to go!

-how do we traverse a binary tree if we want to visit every node?

-eg. we want to print out the data at every node

-how do we decide which direction to take at each node?



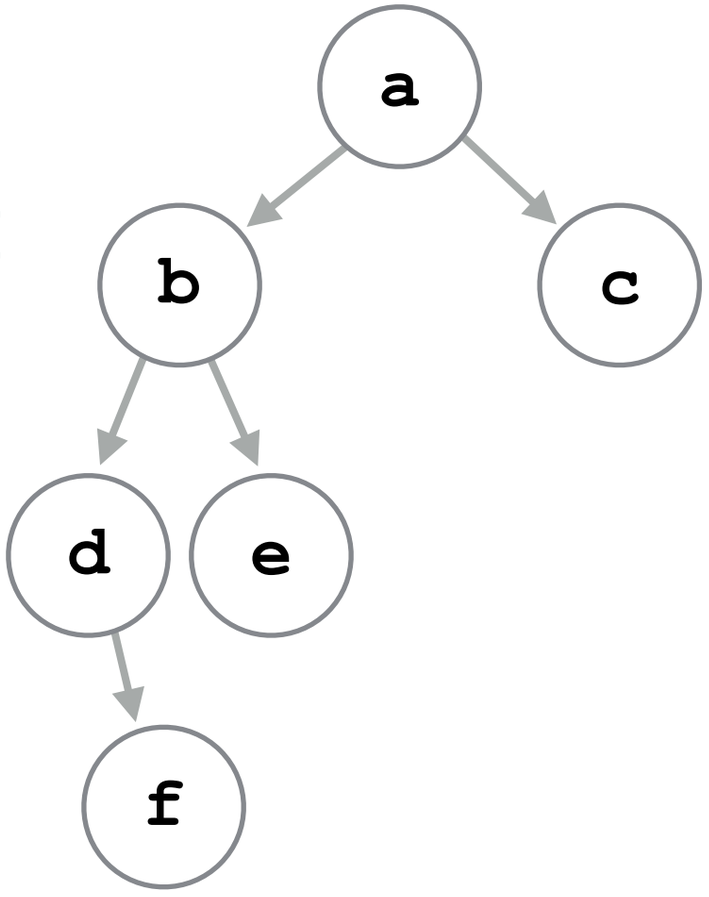
depth-first traversal

-to visit every node, go both directions at each node

-trees are recursive in nature

-start at root, recursively traverse the left subtree, then the right subtree

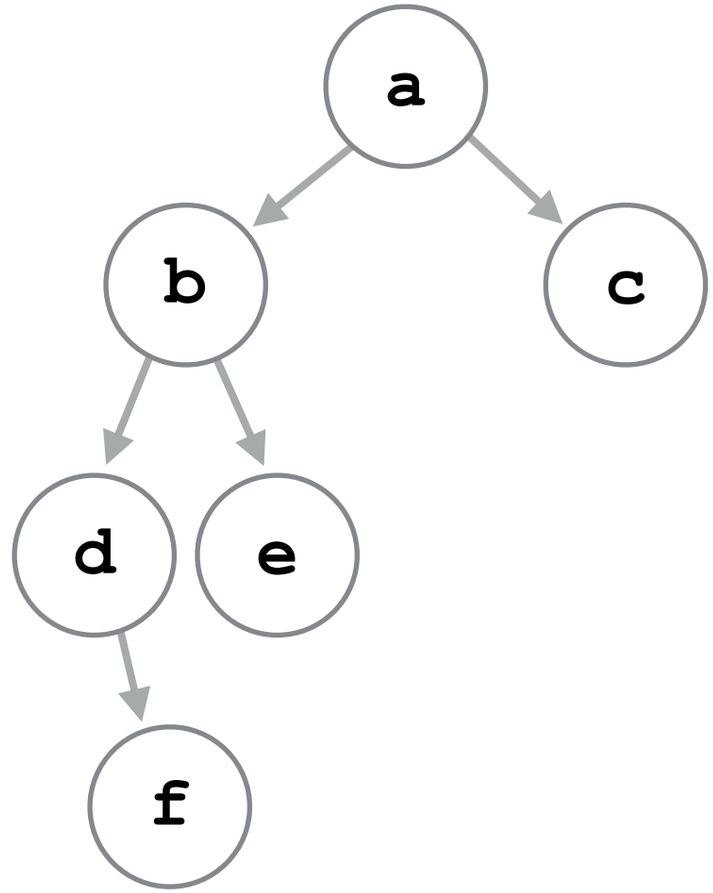
-if the subtree is null, stop (return)



```
public static void DFT(BinaryNode N)
{
  if(N == null)
    return;

  System.out.println(N.data);
  DFT(N.left);
  DFT(N.right);
}
```

WHAT DOES THIS PRINT OUT?



traversal orders

- **pre-order:** use the node before traversing its children

- **in-order:** traverse left child, use node, traverse right child

- **post-order:** use node after traversing both children

- **pre-order:**

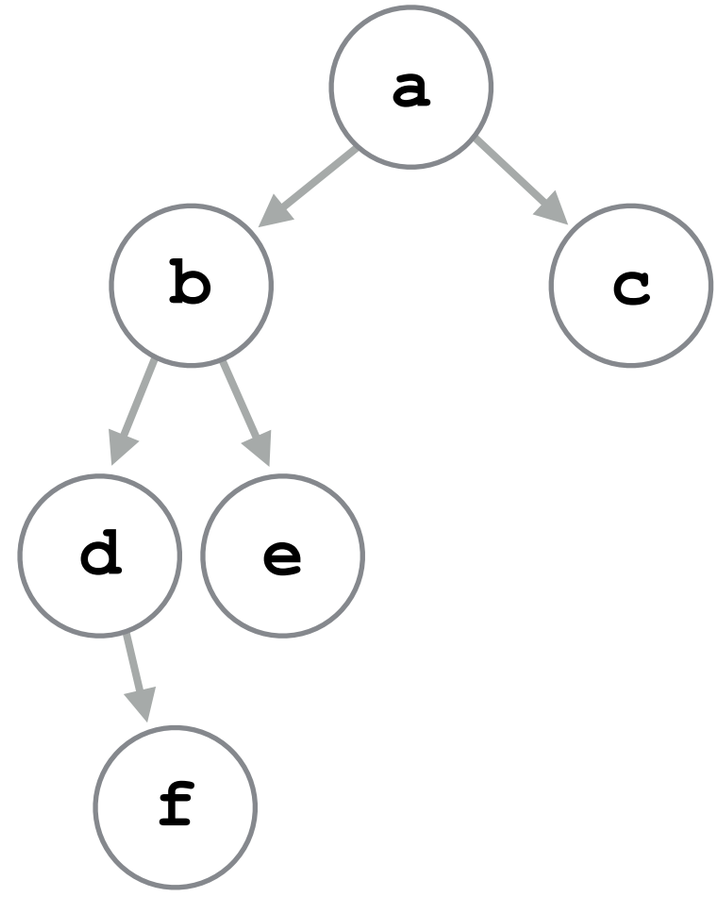
```
use N    // eg. print N
DFT(N.left);
DFT(N.right);
```

- **in-order:**

```
DFT(N.left);
use N    // eg. print N
DFT(N.right);
```

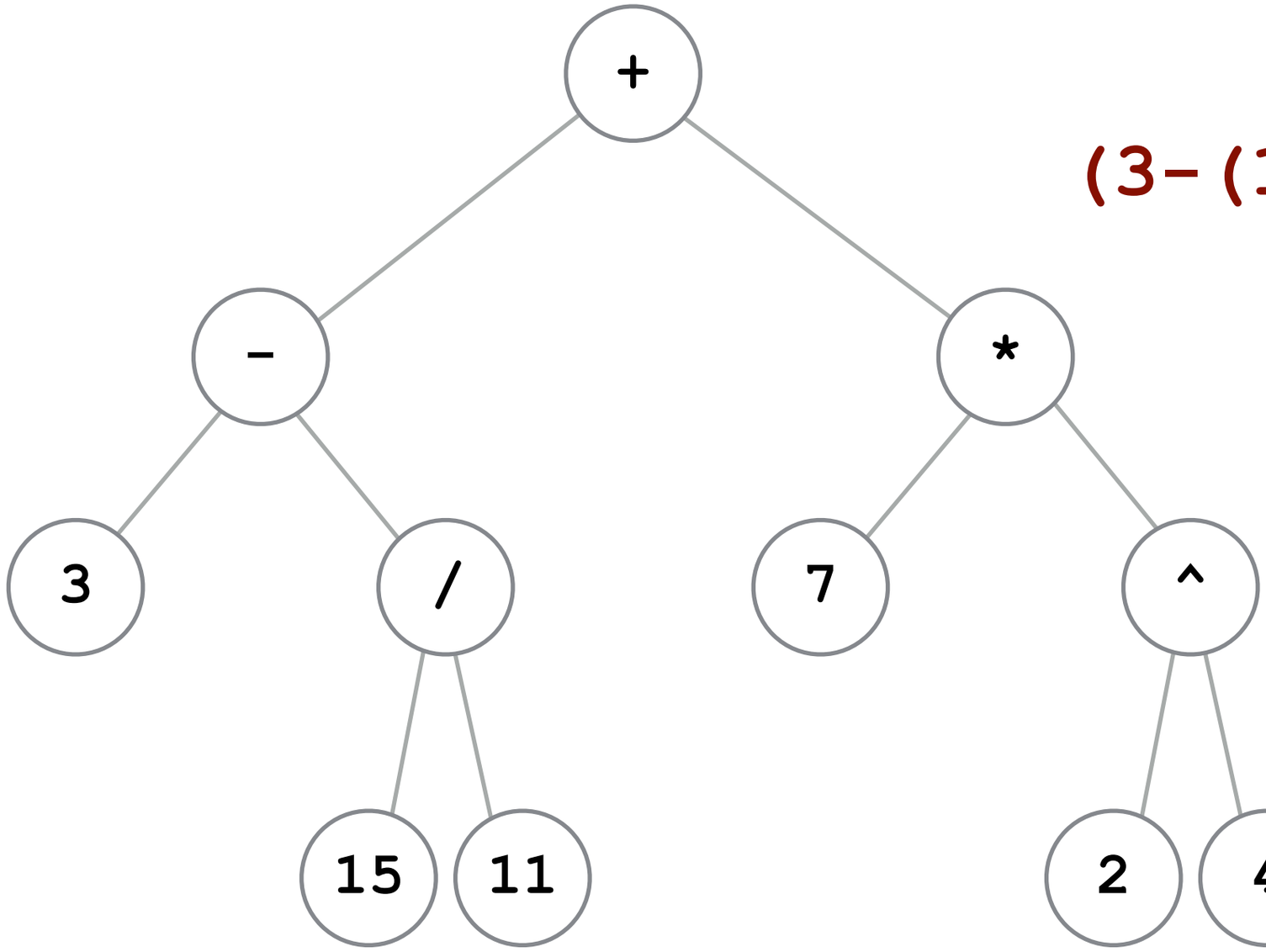
- **post-order:**

```
DFT(N.left);
DFT(N.right);
use N    // eg. print N
```



NOTE: NODES ARE STILL TRAVERSED IN THE SAME ORDER, BUT "USED" (PRINTED) IN A DIFFERENT ORDER

EXAMPLE: expression trees



$(3-(15/11))+(7*2^4)$

HOW CAN WE TRAVERSE THIS TREE TO EVALUATE THE EXPRESSION?

```
public static double evaluate(Node n)
{
  if(n.isLeaf())
    return n.value;

  double leftVal = evaluate(n.left);
  double rightVal = evaluate(n.right);

  switch(n.operator){
   case '+':
     return leftVal + rightVal;
   case '-':
     return leftVal - rightVal;
   …
  }
}
```

```
public static double evaluate(Node n)
{
  if(n.isLeaf())
    return n.value;

  double leftVal = evaluate(n.left);
  double rightVal = evaluate(n.right);
```

```
  switch(n.operator){
   case '+':
     return leftVal + rightVal;
   case '-':
     return leftVal - rightVal;
   …
  }
}
```

Node CLASS HAS THESE FIELDS AND METHOD!

DOT format

- DOT is a tool for tree (and graph) visualization
  - it is part of the GraphViz software
  - http://www.graphviz.org
  - installed on the CADE machines

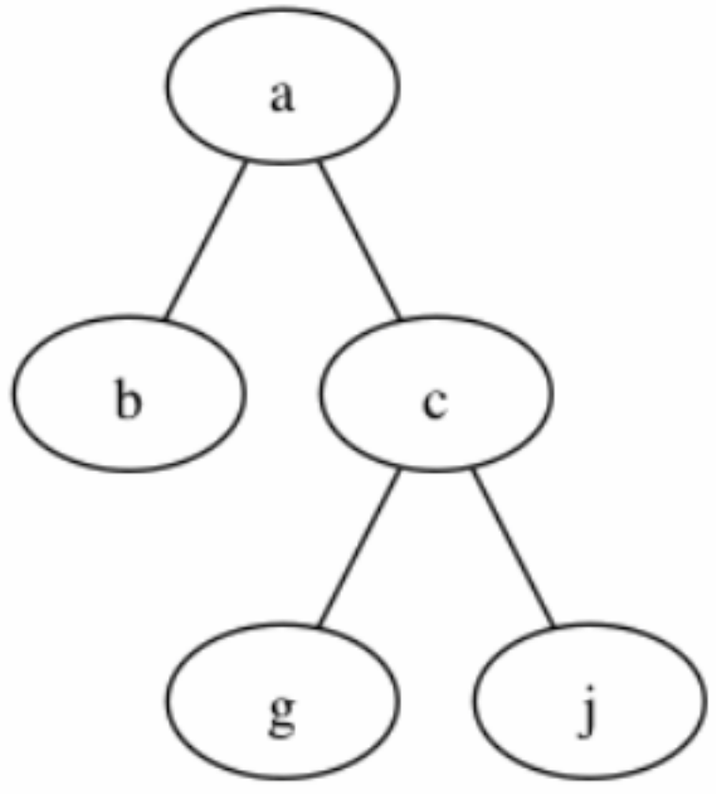
- DOT is also a file format for trees (and graphs)
  - we can (and will!) write Java code to read them as input to construct a tree, as well as output them from an existing tree for debugging purposes

(simplified) DOT format

-the DOT language as *many* features for specifying the layout of a tree (and graph)

-the simplest format looks like this:

```
graph myGraph{
“a” -- “b”
“a” -- “c”
“c” -- “g”
“c” -- “j”
}
```



DOT tool

-the CADE Linux machines have the command-line DOT tool installed

```
dot -Tgif input.dot -o output.gif
```

-“`-Tgif`” means create a .gif file as the result

-“`-o`” means specify the name of the output file

next time…

-reading

-chapters 8 and 19 in book

-chapter 6

-http://opendatastructures.org/ods-java/

-homework

-assignment 7 due Thursday