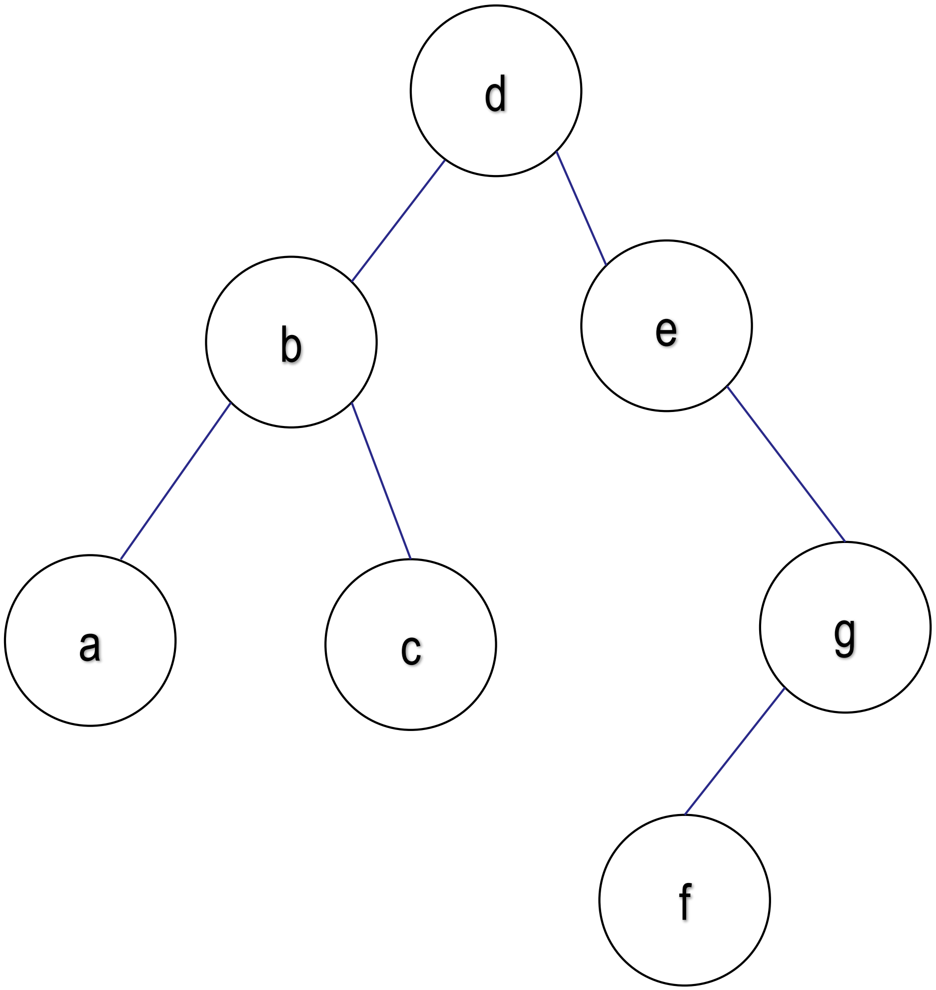
Binary tree traversal

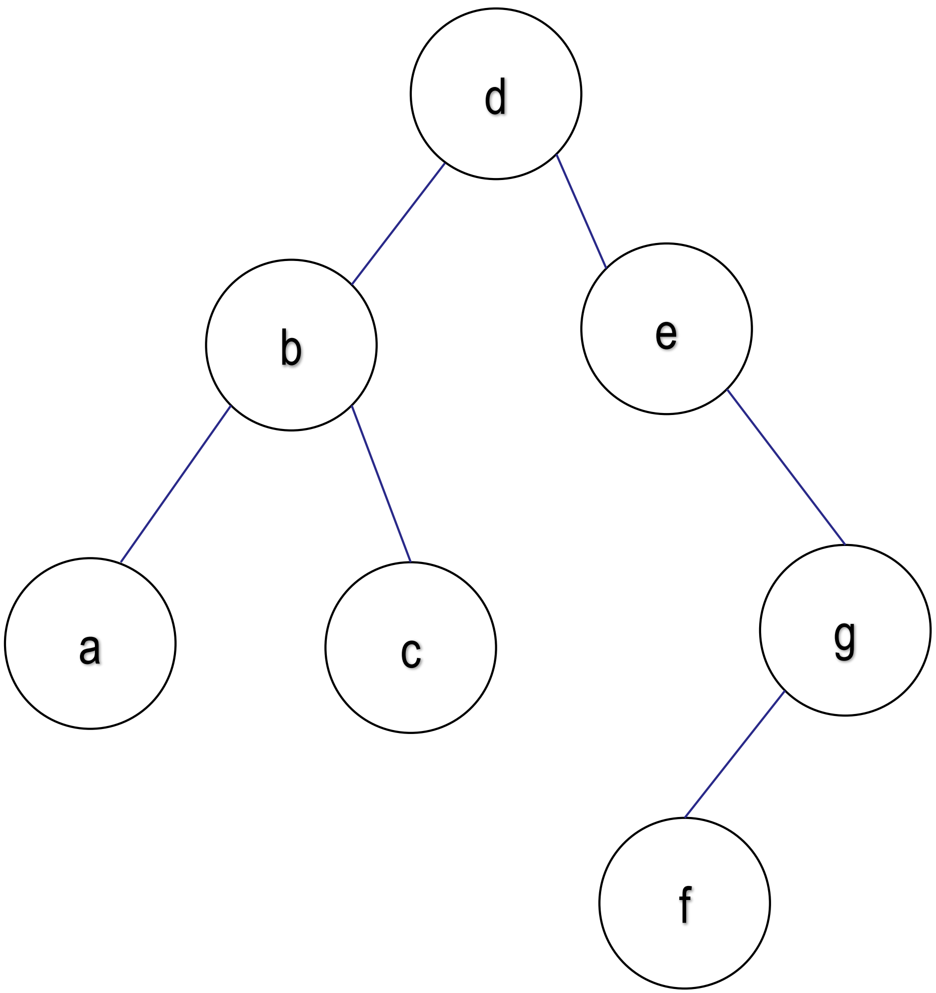


Traversal means navigating through all nodes in the tree.

3 types:
- Pre-order traversal
- In-order traversal
- Post-order traversal

Differ in whether you process a node before its children, between its two children, or after its children

Binary tree pre-order traversal



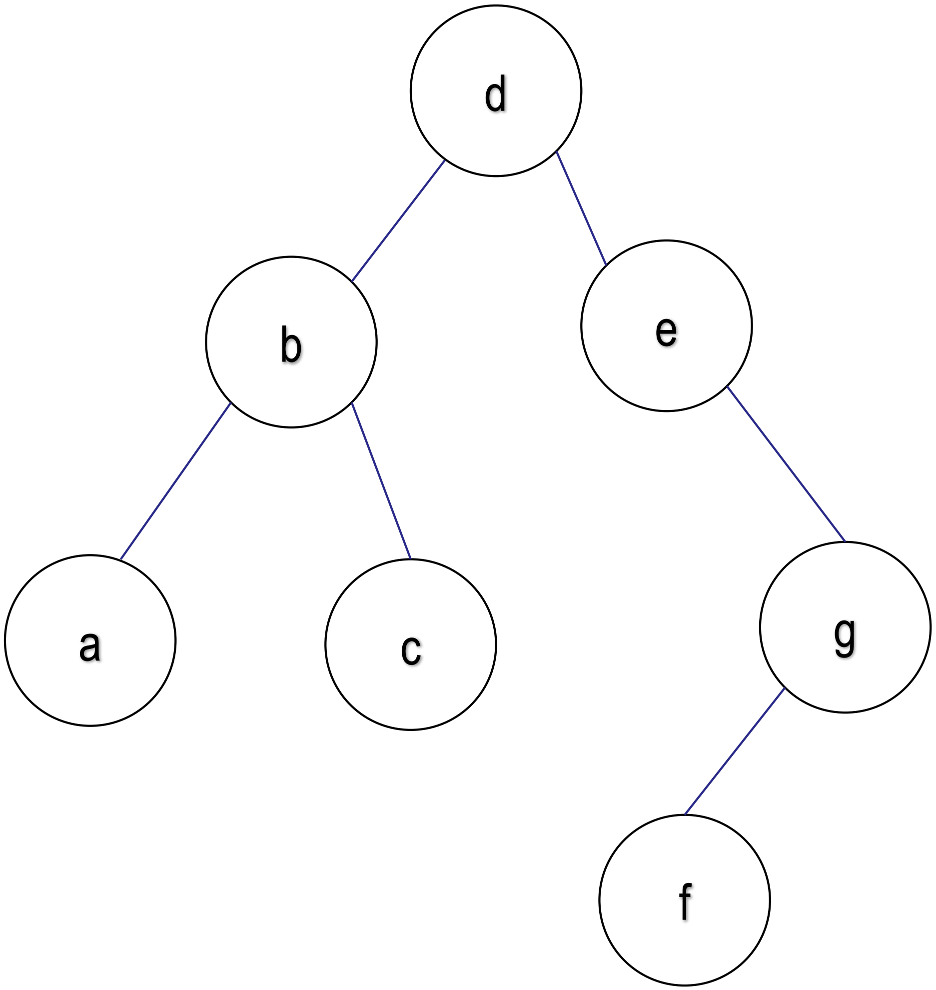
Process a node before its children.

```
Pre-order( Node n ) {
        if (n != null) {
                process n
                Pre-order (n.left);
                Pre-order (n.right);
        }
}
```

Order on this tree:
d b a c e g f

DALHOUSIE UNIVERSITY
Inspiring Minds

Binary tree in-order traversal

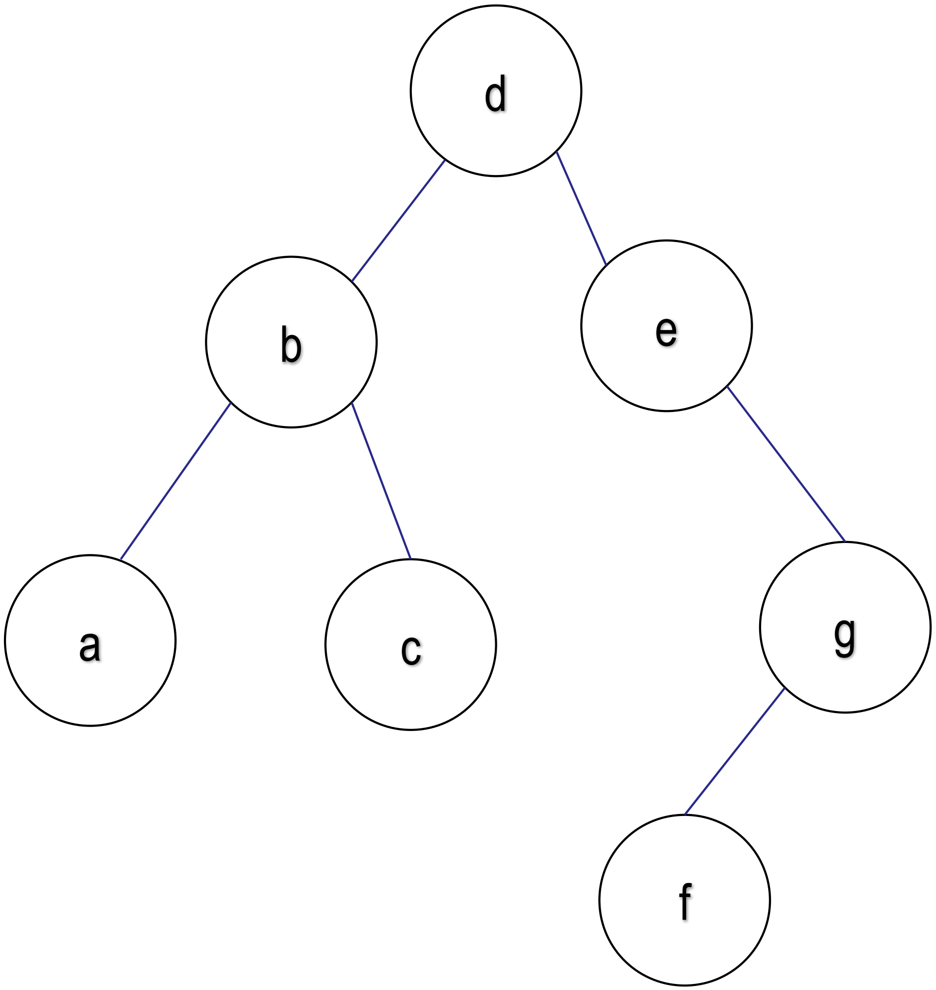


Process a node between its children.

```
In-order( Node n ) {
        if (n != null) {
                In-order (n.left);
                process n
                In-order (n.right);
        }
}
```

Order on this tree:
a b c d e f g

Binary tree post-order traversal

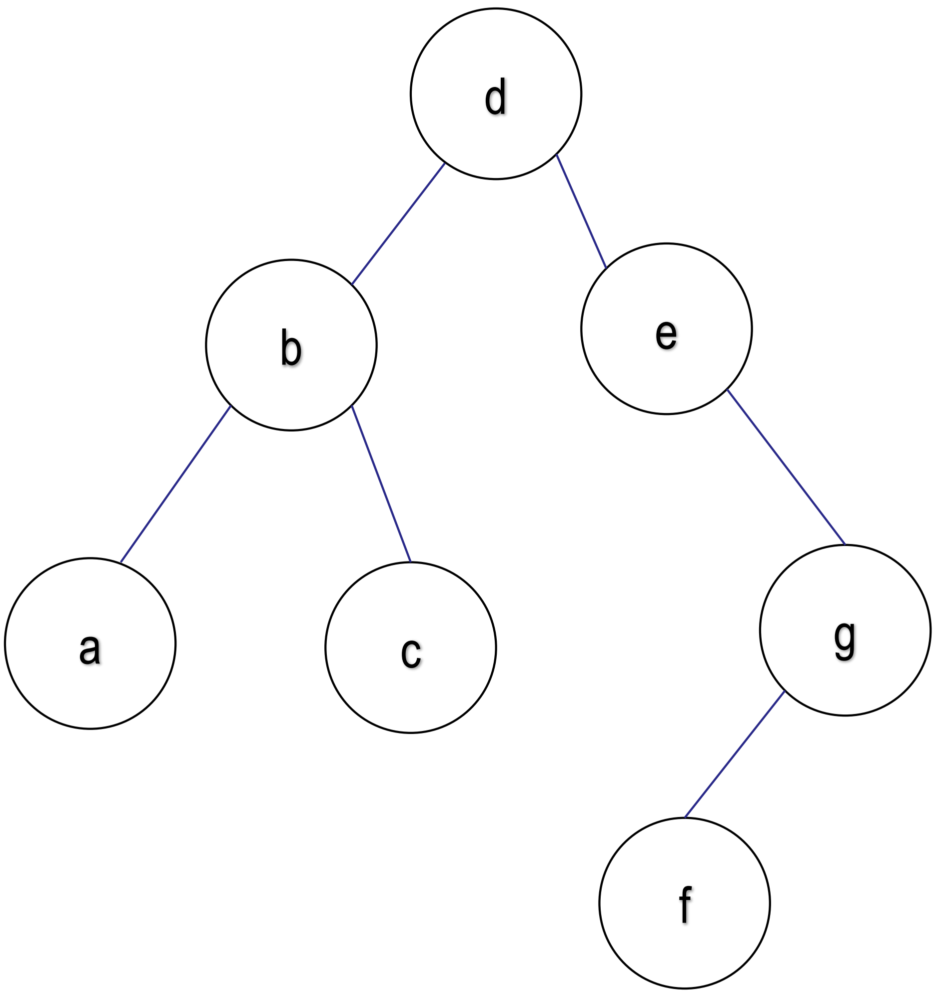


Process a node after its children.

```
Post-order( Node n ) {
        if (n != null) {
                Post-order (n.left);
                Post-order (n.right);
                process n
        }
}
```

Order on this tree:
a c b f g e d

Binary tree breadth-first traversal



Process a node before its children.

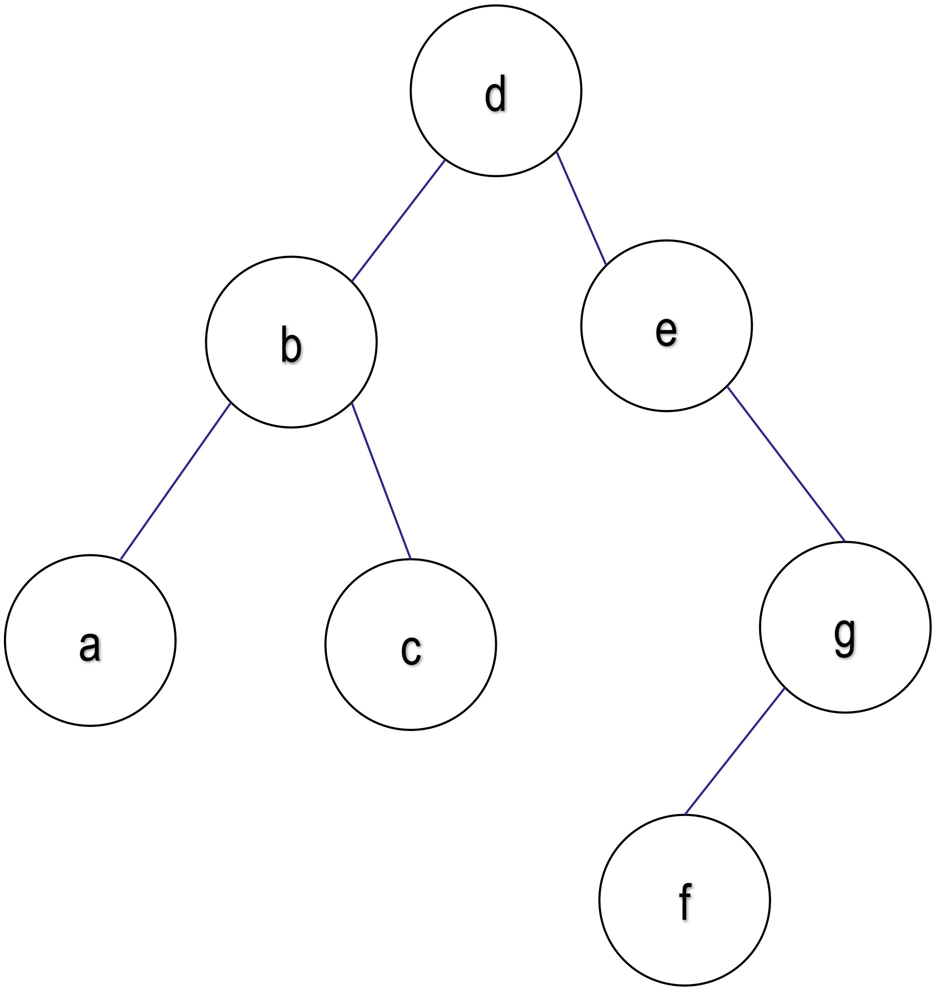
```
BFT( Node n, Queue q ) {
        if (n != null) {
                process n
                q.add (n.left);
                q.add (n.right);
                BFT( q.remove(),q
);
        }
}
```

Order on this tree:
d b e a c g f

Example of tail recursion

DALHOUSIE UNIVERSITY *Inspiring Minds*

Binary tree breadth-first traversal
(non-recursive version)



Process a node before its children.

```
BFT( Node n ) {
        Queue q;
        while (n != null) {
                process n
                q.add (n.left);
                q.add (n.right);
                n = q.remove();
        }
}
```

DALHOUSIE UNIVERSITY
Inspiring Minds

Recursion

- Pro
  - Code can look simpler
  - Fewer lines of code
  - The call stack manages data to remember
  - Naturally fits some problems
- Con
  - Can consume lots of stack space
  - Typically less time-efficient than iterative solutions
  - Can inadvertently solve the same sub-problem multiple times
    - Consider memoization

DALHOUSIE UNIVERSITY
Inspiring Minds

Keys to recursion

- Have all of the stopping cases
- Ensure that each recursive call does provide a smaller problem instance

- Practice

Defensive Programming

Defensive Programming

- **It's about a programming style that buffers your implementation from errors in how other parts of the program may use your code or methods.**

DALHOUSIE UNIVERSITY
Inspiring Minds

Defensive Programming for Robustness

- Robustness: Ensure that your program as a whole continues to run no matter what bad information comes its way
- Correctness: Ensure that your program never returns an inaccurate result
- The two concepts are different!

Defensive Programming

- Defensive programming comes at a cost
  - Run time cycles to check for odd cases
  - Memory if adding check information to data structures
  - Maintenance of defensive programming code
  - Potential for errors in the defense code

- Find the degree of defensive programming that matches your context

How can others influence your code?

- User input
- Parameter values
- Resource permissions
- Environment variables
- Data read in
  - Files
  - Database
  - Network

Input Validation

- Decide on a <u>consistent</u> model on how to handle bad input data
  - Pretend the method succeeded in a “vacuous” manner?
  - Have the method fail automatically?
  - Throw an exception?
  - Return an error code?

DALHOUSIE UNIVERSITY
Inspiring Minds