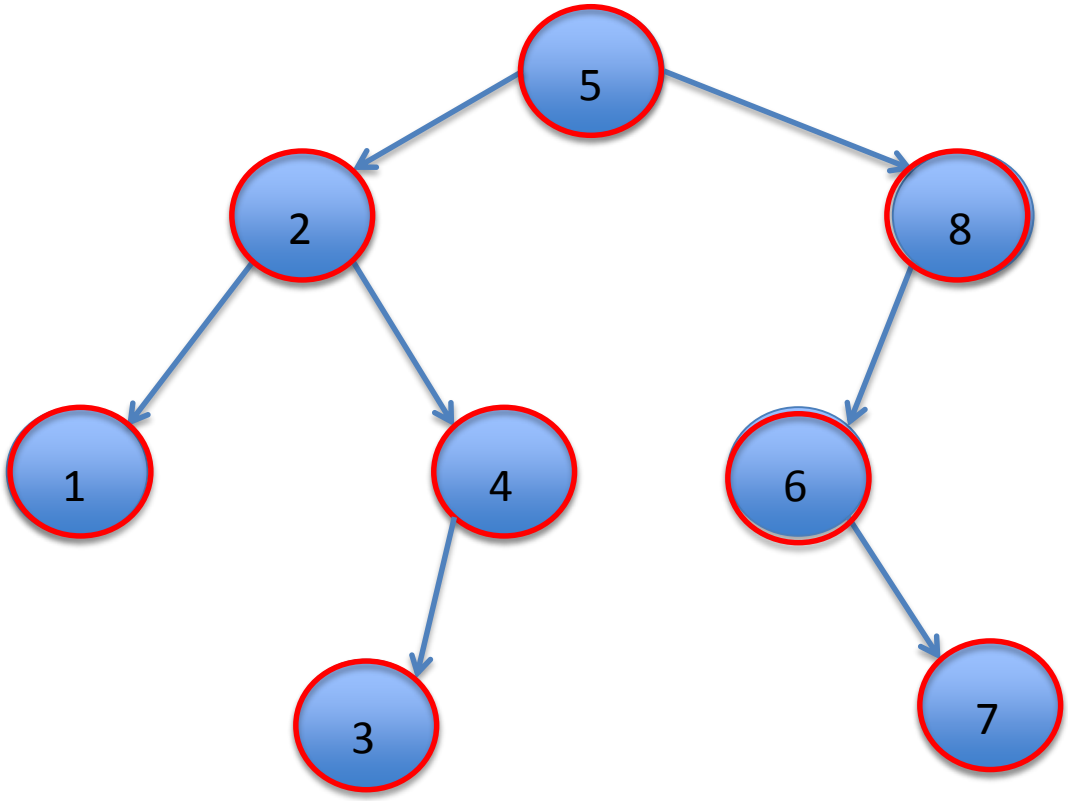
Searching a tree

- Imagine we want to examine a tree
  - To determine if an element is present
  - To find a path to a solution point
  - To make a series of decisions to reach some objective

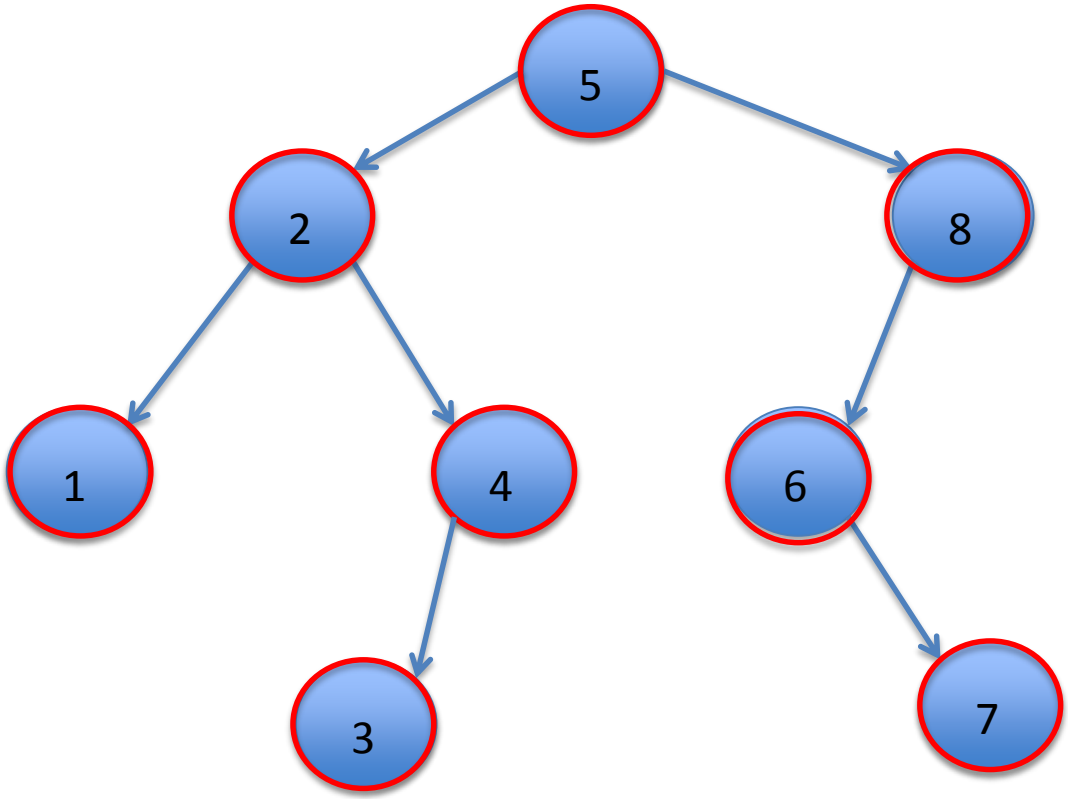
Searching a tree

- Depth first search
  - Start with the root
  - At any node, if we haven't reached our objective, take the left branch first
  - When get to a leaf, backtrack to the first decision point and take the right branch
- Breadth first search
  - Start with the root
  - Then proceed to each child at the next level, in order
  - Continue until reach objective

Depth first search



Breadth first search



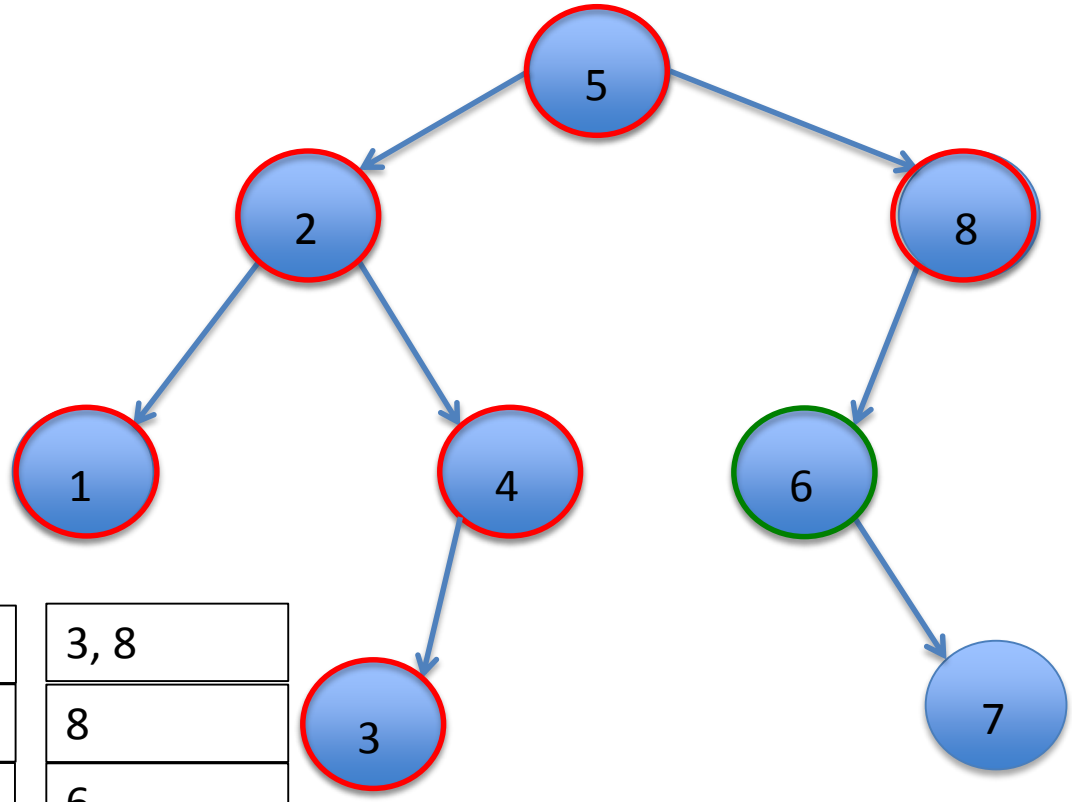
Depth first search for containment

- Idea is to keep a data structure (called a stack) that holds nodes still to be explored
- Use an evaluation function to determine when reach objective (i.e. for containment, whether value of node is equal to desired value)
- Start with the root node
- Then add children, if any, to front of data structure, with left branch first
- Continue in this manner

DFS code

```
def DFSBinary(root, fcn):
    stack= [root]
    while len(stack) > 0:
        if fcn(stack[0]):
            return True
        else:
            temp = stack.pop(0)
            if temp.getRightBranch():
                stack.insert(0,
temp.getRightBranch())
            if temp.getLeftBranch():
                stack.insert(0,
temp.getLeftBranch())
    return False
```

Depth first search



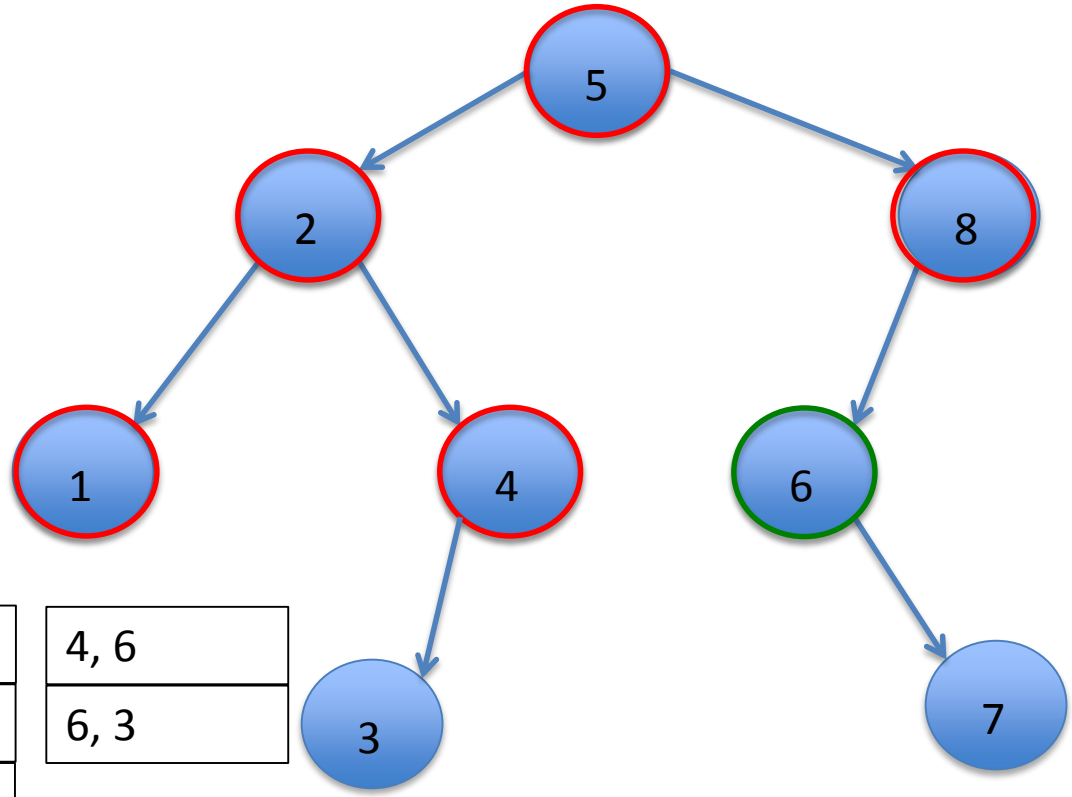
5
2, 8
1, 4, 8
4, 8

3, 8
8
6

BFS code

```
Def BFSBinary(root, fcn):
    queue = [root]
    while len(queue) > 0:
        if fcn(queue[0]):
            return True
        else:
            temp = queue.pop(0)
            if temp.getLeftBranch():
                queue.append(temp.getLeftBranch())
            if temp.getRightBranch():
                queue.append(temp.getRightBranch())
    return False
```

Breadth first search



5
2, 8
8, 1, 4
1, 4, 6

4, 6
6, 3

Depth first search for path

- Suppose we want to find the actual path from root node to desired node
- A simple change in the code lets us trace back up the tree, once we find the desired node

DFS code

```
def DFSBinaryPath(root, fcn):
    stack= [root]
    while len(stack) > 0:
        if fcn(stack[0]):
            return TracePath(stack[0])
        else:
            temp = stack.pop(0)
            if temp.getRightBranch():
                stack.insert(0, temp.getRightBranch())
            if temp.getLeftBranch():
                stack.insert(0, temp.getLeftBranch())
    return False

def TracePath(node):
    if not node.getParent():
        return [node]
    else:
        return [node] + TracePath(node.getParent())
```

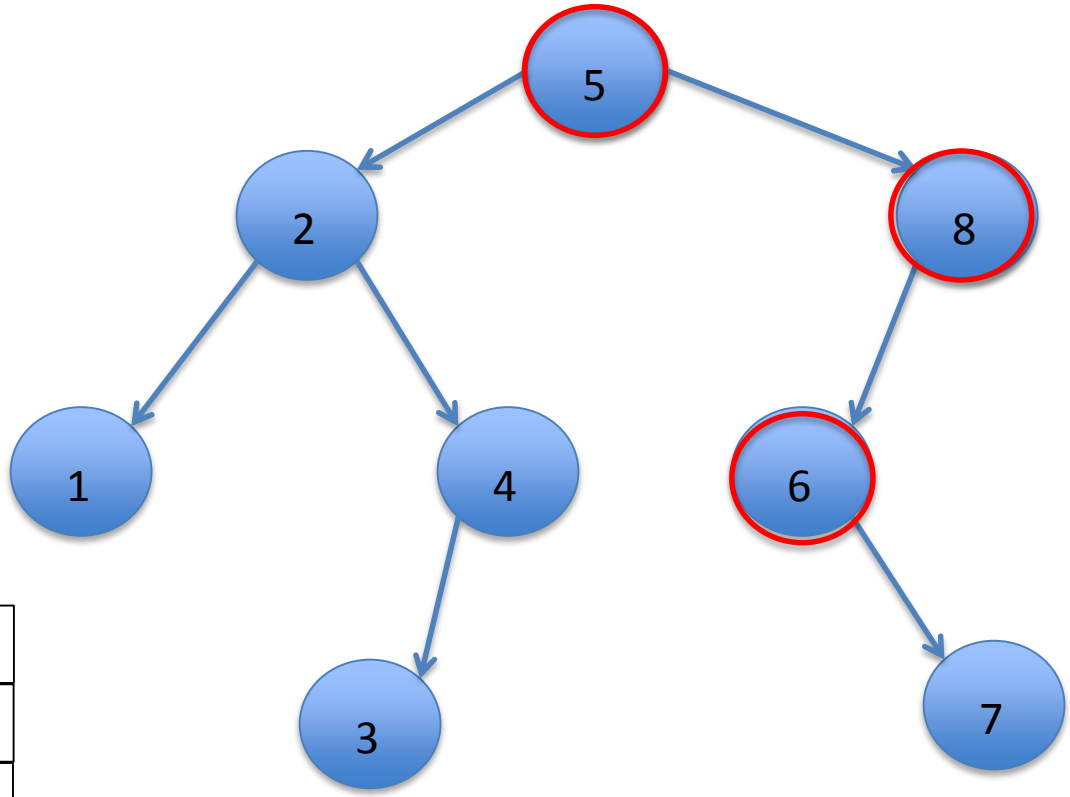
Ordered search

- Suppose we know that the tree is ordered, meaning that for any node, all the nodes to the “left” are less than that node’s value, and all the nodes to the “right” are greater than that node’s value

DFS code

```
def DFSBinaryOrdered(root, fcn, ltFcn):
    stack= [root]
    while len(stack) > 0:
        if fcn(stack[0]):
            return True
        elif ltFcn(stack[0]):
            temp = stack.pop(0)
            if temp.getLeftBranch():
                stack.insert(0,
temp.getLeftBranch())
        else:
            if temp.getRightBranch():
                stack.insert(0,
temp.getRightBranch())
    return False
```

Depth first ordered search



5
8
6