Trees – Part 2

This lesson is borrowed from the following:

Reference

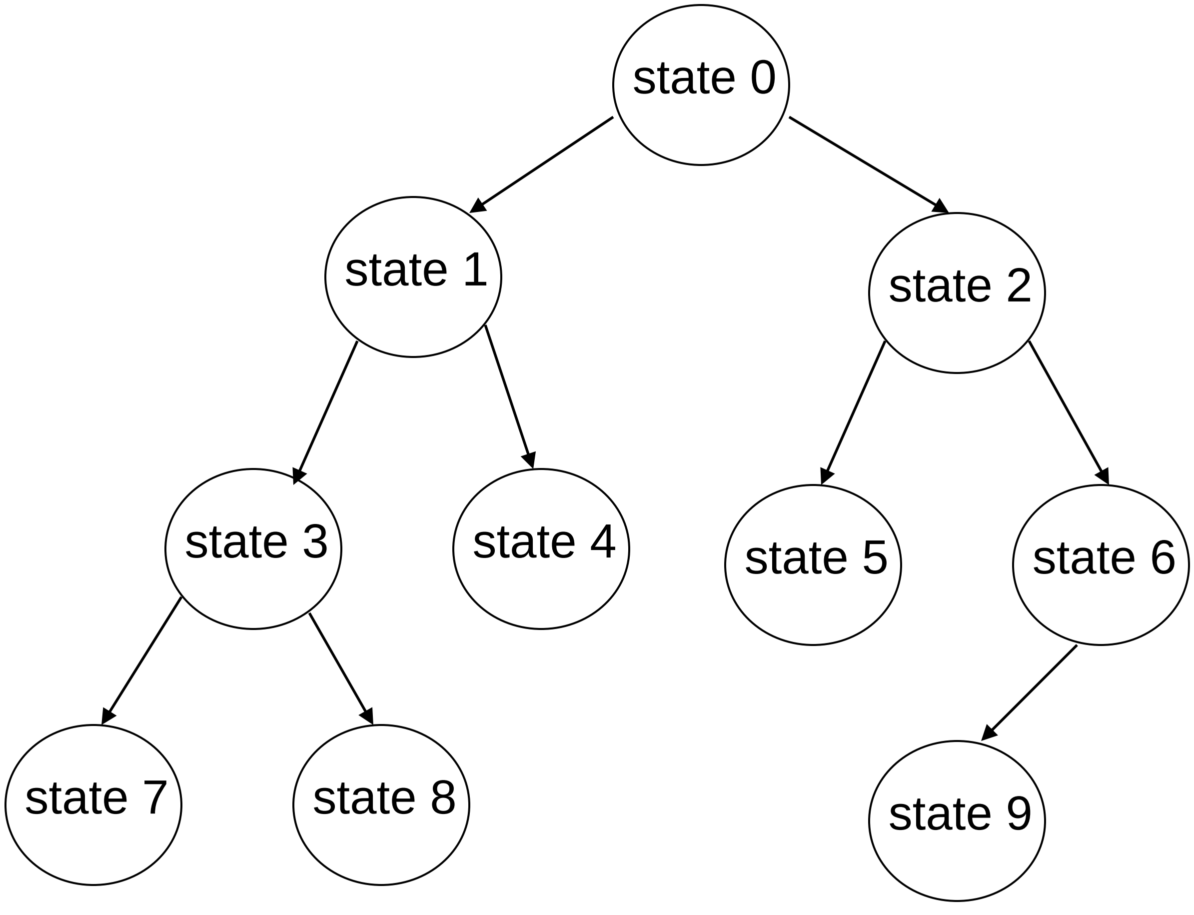
CS 367 – Introduction to Data Structures

http://pages.cs.wisc.edu/~mattmcc/cs367/notes/Trees-II.ppt

Tree Traversal

- Sometimes necessary to scan entire tree
  - imagine a system that has no keys, just states
    - AI algorithms are a great example
  - to find the best state, must search all nodes
- Two ways to search an entire tree
  - breadth first
    - search all nodes at one level, and then go to next level
  - depth first
    - go all the way down one branch and then the next and so on

Breadth First Search



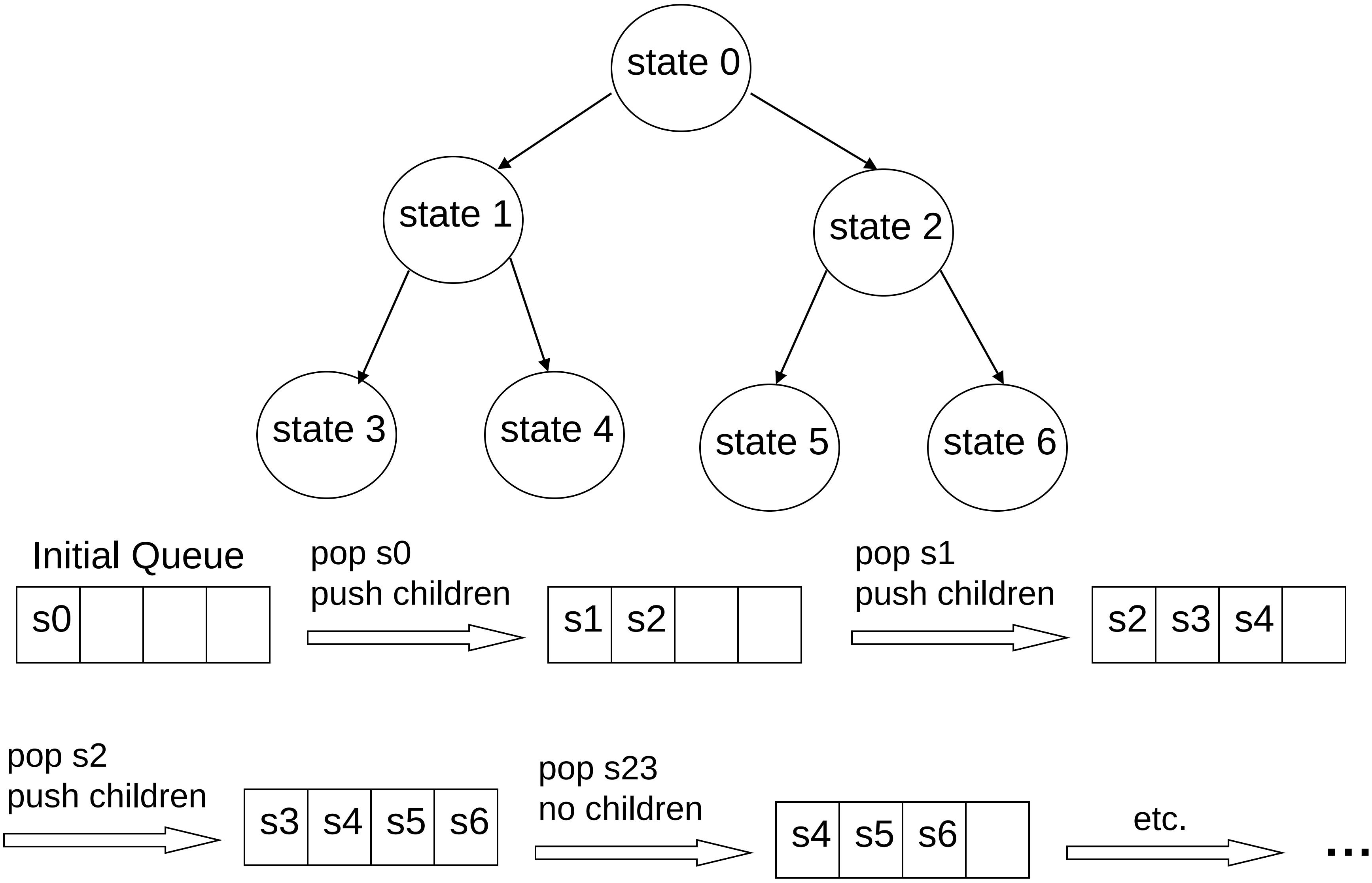
Search Order

state 0
state 1
state 2
state 3
state 4
state 5
state 6
state 7
state 8
state 9

Breadth First Search

- Best way to implement a breadth first search is with a queue
  - visit a node
  - enqueue all of the nodes children
  - dequeue the next item from the queue
  - repeat until the queue is empty

Breadth First Search



Breadth First Search

```
public void printTree-Breadth() {
    if(root == null) {
        System.out.println("Empty tree.");
        return;
    }
    Queue queue = new QueueList();
    queue.enqueue(root);
    while(!queue.isEmpty()) {
        TreeNode tmp = queue.dequeue();
        System.out.println(tmp.data.toString());
        if(tmp.left != null) { queue.enqueue(tmp.left); }
        if(tmp.right != null) { queue.enqueue(tmp.right); }
    }
}
```

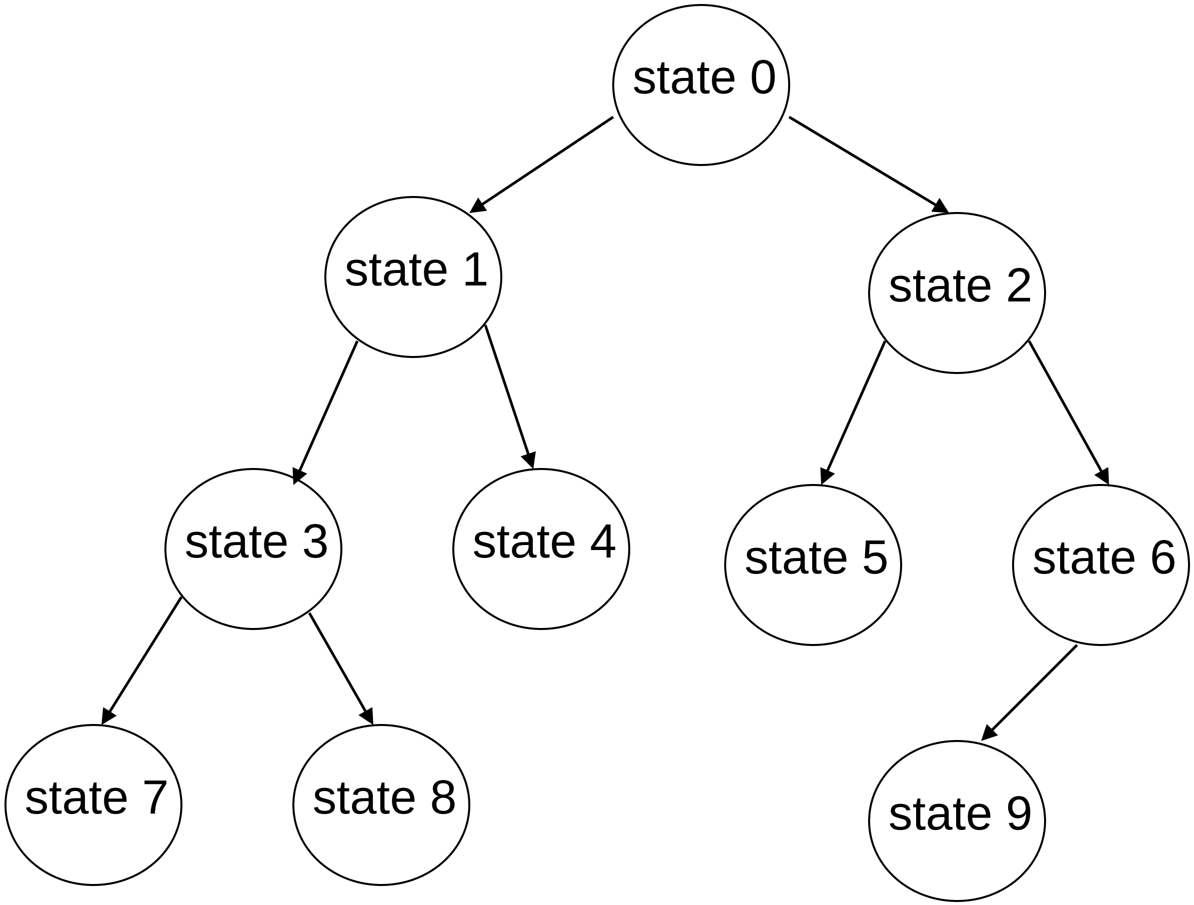
Breadth First Search

- Advantage
  - guaranteed to find the wanted state
    - if it exists
- Disadvantage
  - excessive memory requirements
  - $M_{needed} = 2^{level - 1}$

Depth First Search

- Three possible orderings for depth first
  - preorder
    - visit node, then its left child, then its right child
  - inorder
    - visit left child, then the node, then right child
  - postorder
    - visit left child, then right child, then the node
- All depth first searches are easy to implement with recursion

Depth First - Preorder



Search Order

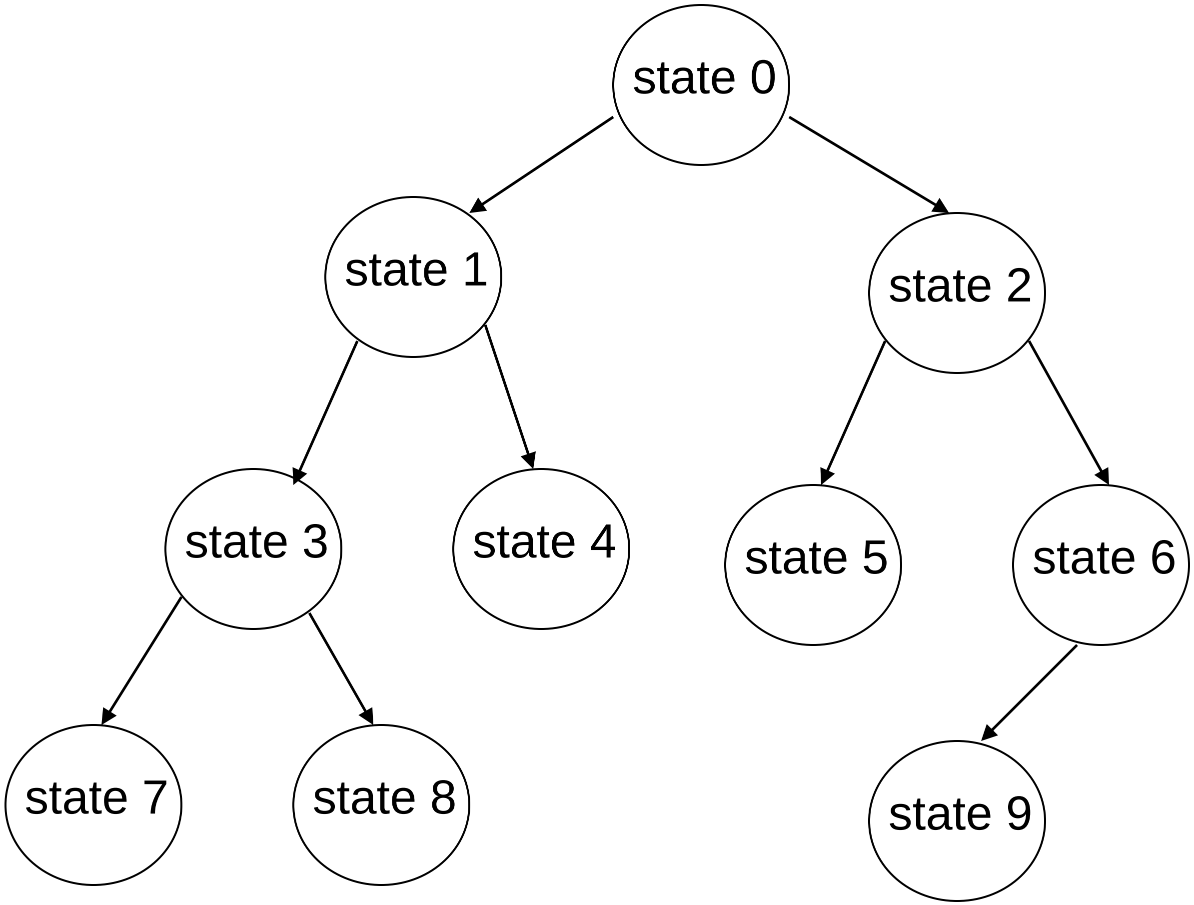
state 0
state 1
state 3
state 7
state 8
state 4
state 2
state 5
state 6
state 7

Depth First - Preorder

```
public void printTree-Preorder() {
    if(root == null) { System.out.println(“Empty tree”); }
    else { printTree-Preorder(root); }
}

private void printTree-Preorder(Node node) {
    if(node == null)
        return;
    System.out.println(node.data.toString());
    printTree-Preorder(node.left);
    printTree-Preorder(node.right);
}
```

Depth First - Inorder



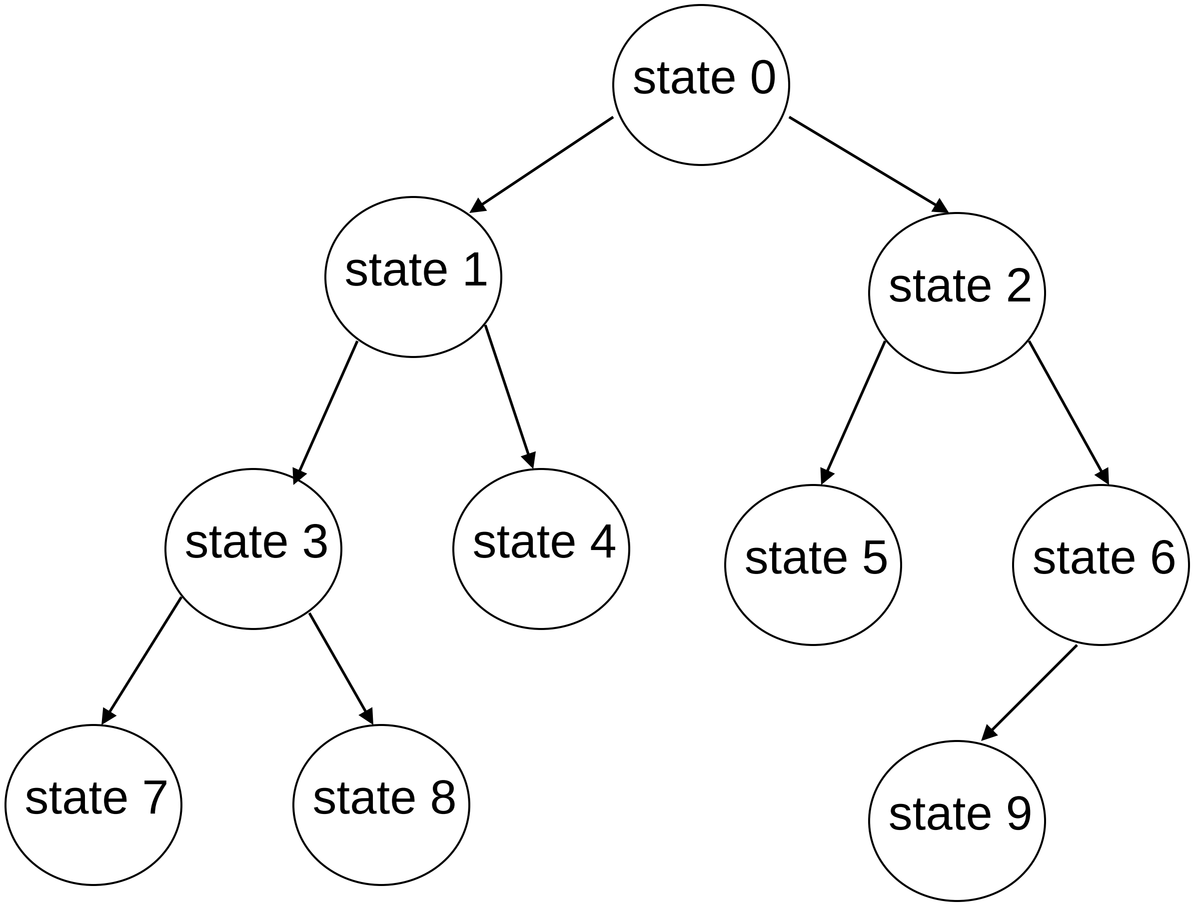
Search Order
state 7
state 3
state 8
state 1
state 4
state 0
state 5
state 2
state 9
state 6

Depth First - Inorder

```
public void printTree-Inorder() {
    if(root == null) { System.out.println(“Empty tree”); }
    else { printTree-Inorder(root); }
}

private void printTree-Inorder(Node node) {
    if(node == null)
        return;
    printTree-Inorder(node.left);
    System.out.println(node.data.toString());
    printTree-Inorder(node.right);
}
```

Depth First – Postorder



Search Order

state 7
state 8
state 3
state 4
state 1
state 5
state 9
state 6
state 2
state 0

Depth First - Postorder

```
public void printTree-Postorder() {
    if(root == null) { System.out.println("Empty tree"); }
    else { printTree-Postorder(root); }
}

private void printTree-Postorder(Node node) {
    if(node == null)
        return;
    printTree-Postorder(node.left);
    printTree-Postorder(node.right);
    System.out.println(node.data.toString());
}
```

Depth First Search

- Advantages
  - requires much less memory than breadth first
    - M_{needed} = level
- Disadvantage
  - may never find the solution
    - some search spaces have an infinite number of states (or very nearly infinite)
    - this means a single “branch” is infinite
    - we’ll never search other branches

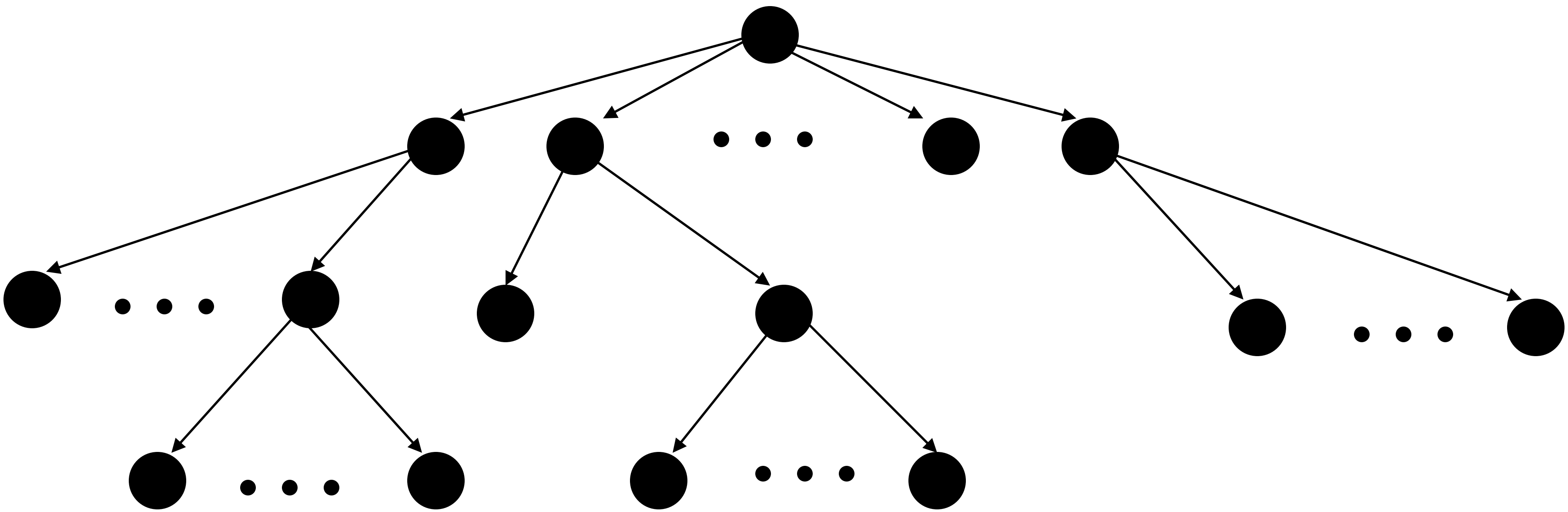
Application of Tree Traversal

- We’ve already shown how tree traversal can be used to print out all of the nodes
  - this is good for debugging, but not needed for much else
- Consider a computer chess game
  - each node represents a state of the game
    - where each piece is on the board
  - for the computer to decide it’s next move, it would like to pick a node that will lead to the best state

Chess Game

- We’ll say a node contains the following
  - the position of each piece on the board
  - a value indicating how favorable the current positions are
    - high value means it’s good
      - check mating opponent would be the highest value
    - a low value means it’s bad
      - being in check mate will be the lowest value
  - each node will have from 0 to 16 children
    - why?

Chess Game



There are millions of more states but they won’t all fit on the slide.
☺

Chess Game

- Consider the first possible move
  - can move any one of 8 pawns or 2 knights
    - that means we have 10 successor states
  - opponent will then have 10 possible moves
  - obviously, the number of states available at just the second level is immense
    - the overall search space is virtually infinite
- Two possible solutions
  - use a breadth first search and only go to a certain level
    - possibly monumental memory requirements
  - use a depth first search but only go so far along each branch
    - limits the memory requirements