***class Node:*** *"""AIMA: A node in a search tree. Contains a pointer to the parent (the node hat this is a successor of) and to the actual state for this node. Note that if a state is arrived at by two paths, then there are two nodes with he same state. Also includes the action that got us to this state, and the total path\_cost (also known as g) to reach the node. Other functions may add an f and h value; see best\_first\_graph\_search and astar\_search for an explanation of how the f and h values are handled. You will not need to subclass this class. """*

**def** \_\_init\_\_(self, state, parent=None, action=None, path\_cost=0):  
 *"Create a search tree Node, derived from a parent by an action."* self.state = state  
 self.parent = parent  
 self.action = action  
 **if** parent:  
 self.path\_cost = parent.path\_cost + path\_cost  
 self.depth = parent.depth + 1  
 **else**:  
 self.path\_cost = path\_cost  
 self.depth = 0  
  
 **def** \_\_repr\_\_(self):  
 **return "<Node %s>"** % (self.state,)  
  
 **def** nodePath(self):  
 *"Create a list of nodes from the root to this node."* x, result = self, [self]  
 **while** x.parent:  
 result.append(x.parent)  
 x = x.parent  
 result.reverse()  
 **return** result  
  
 **def** expand(self, problem):  
 *"""  
 Return a list of nodes reachable from this node.  
 """* **return** [Node(next, self, act, cost)  
 **for** (next, act, cost) **in** problem.getSuccessors(self.state)]