Pseudocode for Search Algorithms

The following slides contain pseudocode for various search algorithms:

- Search (which can be modified for BFS, DFS, UCS, etc) using appropriate ordering of the OPEN list
- Depth Limited Search
- Iterative Deepening Search
- Search with Path Checking
- Cycle checking ensuring optimality

Algorithm for Search

```
Search(open, successors, goal? ):
    open.insert(<start>)
    while not open.empty():
        n = open.extract()   #remove node from OPEN
        state = n.end_state()
        if (goal?(state)):
            return n              #n is solution
        for succ in sucessors(state):
            open.insert(<n,succ>)
                                  #open could grow or shrink
    return false
```

Depth Limited Search

```
DL_Search(open, successors, goal?, maxd):
    open.insert(<start>)          #OPEN MUST BE A STACK FOR DFS
    cutoff = false
    while not open.empty():
        n = open.extract()        #remove node from OPEN
        state = n.end_state()
        if (goal?(state)):
            return (n,cutoff)     #n is solution
        if depth(n) < maxd:       #Only successors if depth(n) < maxd
            for succ in sucessors(state):
                open.insert(<n,succ>)
        else:
                cutoff= true.     #some node was not
                                  #expanded because of depth
                                  #limit.
    return  (false, cutoff)
```

Iterative Deeping Search

```
ID_Search(open, successors, goal?):
    maxd = 0
    while true:
        (n, cutoff) = DL_Search(open, successors, goal?, maxd)
        if n:
            return n
        elif not cutoff:          #no nodes at deeper levels exit
            return fail
        else:
            maxd = maxd + 1
```

Search with Path Checking

```
Search(open, successors, goal? ):
    open.insert(<start>)
    while not open.empty():
        n = open.extract()        #remove node from OPEN
        state = n.end_state()
        if (goal?(state)):
            return n              #n is solution
        for succ in sucessors(state):
            if not succ in <n>:   #put on OPEN if not path cycle
                open.insert(<n,succ>)
    return false
```

Cycle Checking—ensuring optimality

```
Search(open, successors, goal? ):
    open.insert(<start>)
    seen = {start : 0}                  #seen is dict storing min cost
    while not open.empty():
        n = open.extract()
        state = n.end_state()
        if cost(n) <= seen[state]:  #only expand if cheapest path
            if (goal?(state)):
                return n
            for succ in sucessors(state):
                if not succ in seen or cost(<n,succ>) < seen[succ]:
                    open.insert(<n,succ>)
                    seen[succ] = cost(<n,succ>)
    return false
```