SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with ‘A++’ Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF INFORMATION TECHNOLOGY

19CSE303 - ARTIFICIAL INTELLIGENCE
III YEAR IV SEM

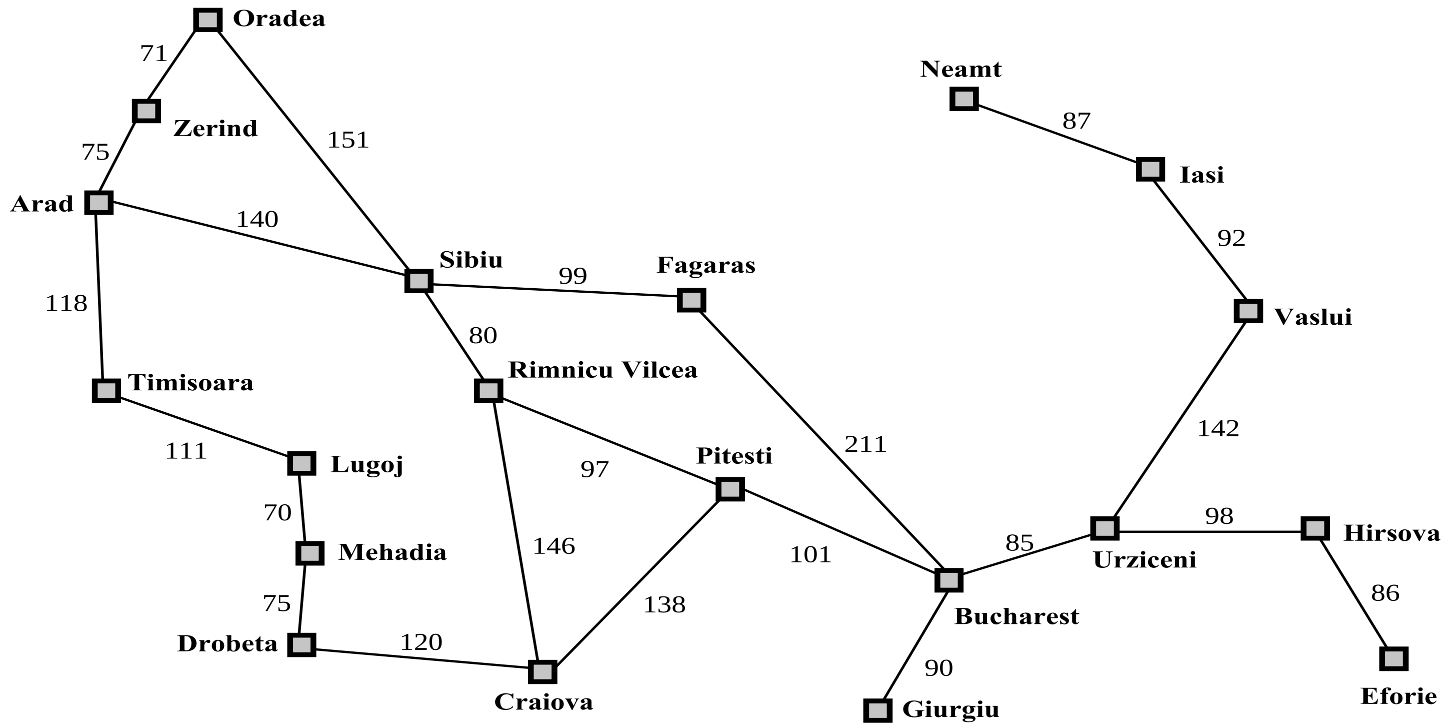
UNIT I – PROBLEM SOLVING

TOPIC – Problem Formulation

Setup

- Perception/action cycle [board]

- Goal-based agent: find a sequence of actions to achieve a goal
  - Search, then execute
- The methods in this chapter are appropriate for problems for which the environment is observable, discrete, determininistic.[which means?]

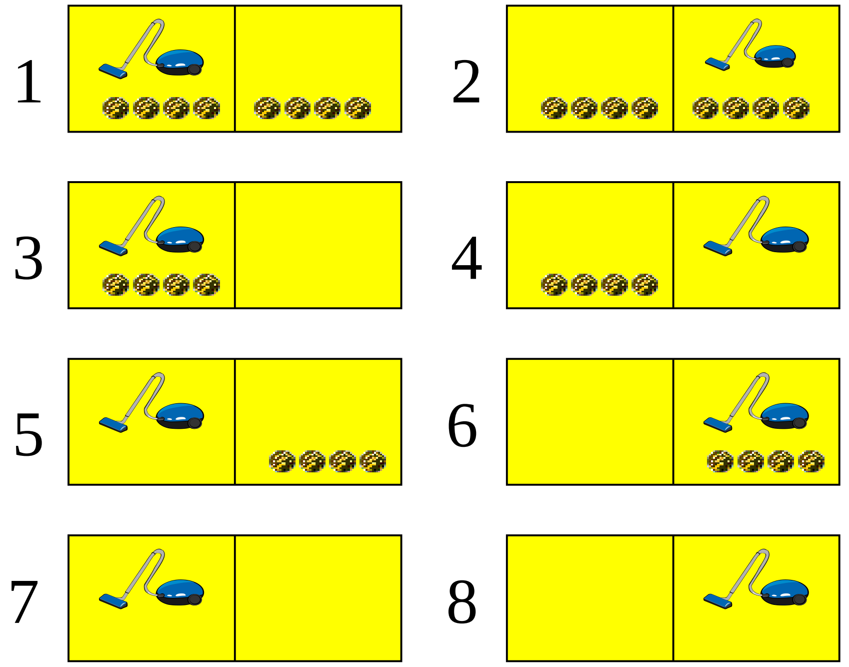

Oradea
71
Zerind
75
151
Arad
140
Sibiu
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
75
138
Drobeta
120
Craiova
Neamt
87
Iasi
92
Vaslui
142
98
Hirsova
85
Urziceni
Bucharest
86
90
Giurgiu
Eforie

Example Problems

- Toy problems
  - Illustrate/test various problem-solving methods
  - Concise, exact description
  - Can be used to compare performance
  - *Examples*: 8-puzzle, 8-queens problem, Cryptarithmetic, Vacuum world, Missionaries and cannibals, simple route finding
- Real-world problem
  - More difficult
  - No single, agreed-upon specification (state, successor function, edgecost)
  - *Examples*: Route finding, VLSI layout, Robot navigation, Assembly sequencing (read 3.2.2 about complexities)

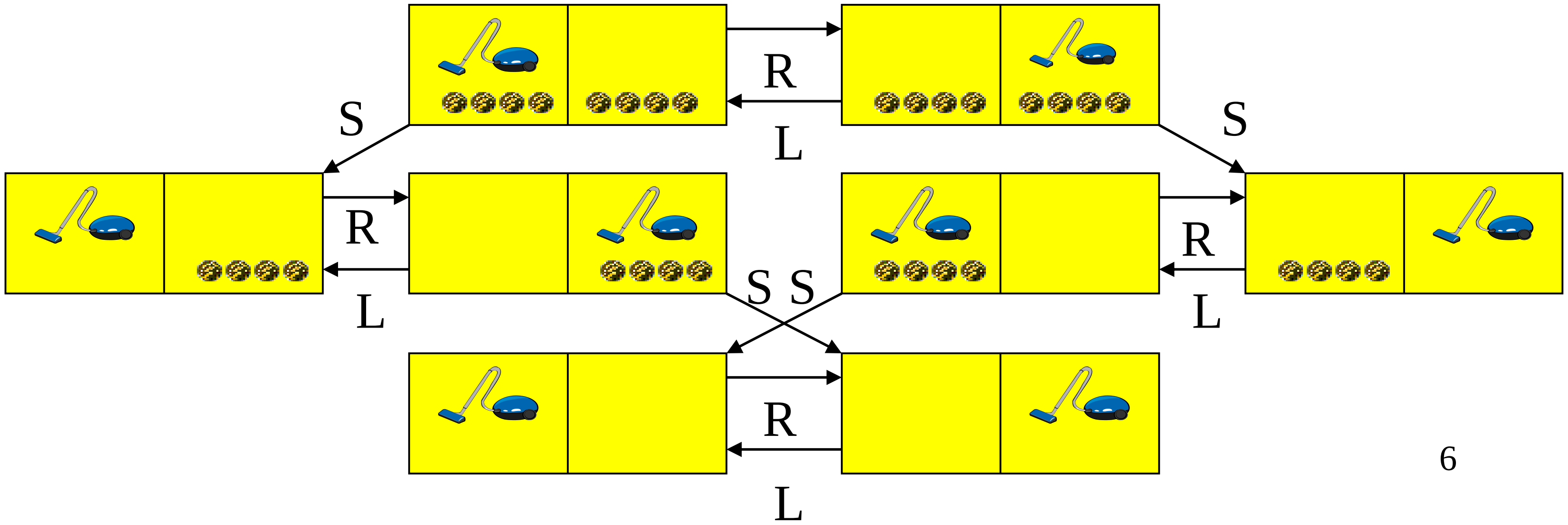
Toy Problems
The vacuum world

- The vacuum world
  - The world has only two *locations*
  - Each location may or may not contain *dirt*
  - The agent may be in one location or the other
  - 8 possible *world states*
  - Three possible actions: *Left, Right, Suck*
  - *Goal*: clean up all the dirt



Toy Problems
The vacuum world

- *States*: one of the 8 states given earlier
- *Operators*: move left, move right, suck
- *Goal test*: no dirt left in any square
- *Path cost*: each action costs one



Missionaries and cannibals

- Missionaries and cannibals
  - Three missionaries and three cannibals want to cross a river
  - There is a boat that can hold two people
  - Cross the river, but make sure that the missionaries are not outnumbered by the cannibals on either bank
- Lots of *abstraction*
  - Crocodiles in the river, the weather and so on
  - Only the endpoints of the crossing are important, etc.

Missionaries and cannibals

- http://www.novelgames.com/gametips/details.php?id=29
- [Problem formulation]

Real-world problems

- Route finding
  - Defined in terms of locations and transitions along links between them
  - *Applications*: routing in computer networks, automated travel advisory systems, airline travel planning systems
- Touring and traveling salesperson problems
  - "Visit every city on the map at least once and end in Bucharest"
  - Needs information about the visited cities
  - *Goal*: Find the shortest tour that visits all cities
  - *NP-hard*, but a lot of effort has been spent on improving the capabilities of TSP algorithms
  - *Applications*: planning movements of automatic circuit board drills

Real-world problems

- VLSI layout
  - Place cells on a chip so they don't overlap and there is room for connecting wires to be placed between the cells
- Robot navigation
  - Generalization of the route finding problem
    - No discrete set of routes
    - Robot can move in a continuous space
    - Infinite set of possible actions and states
- Assembly sequencing
  - Automatic assembly of complex objects
  - The problem is to find an order in which to assemble the parts of some object

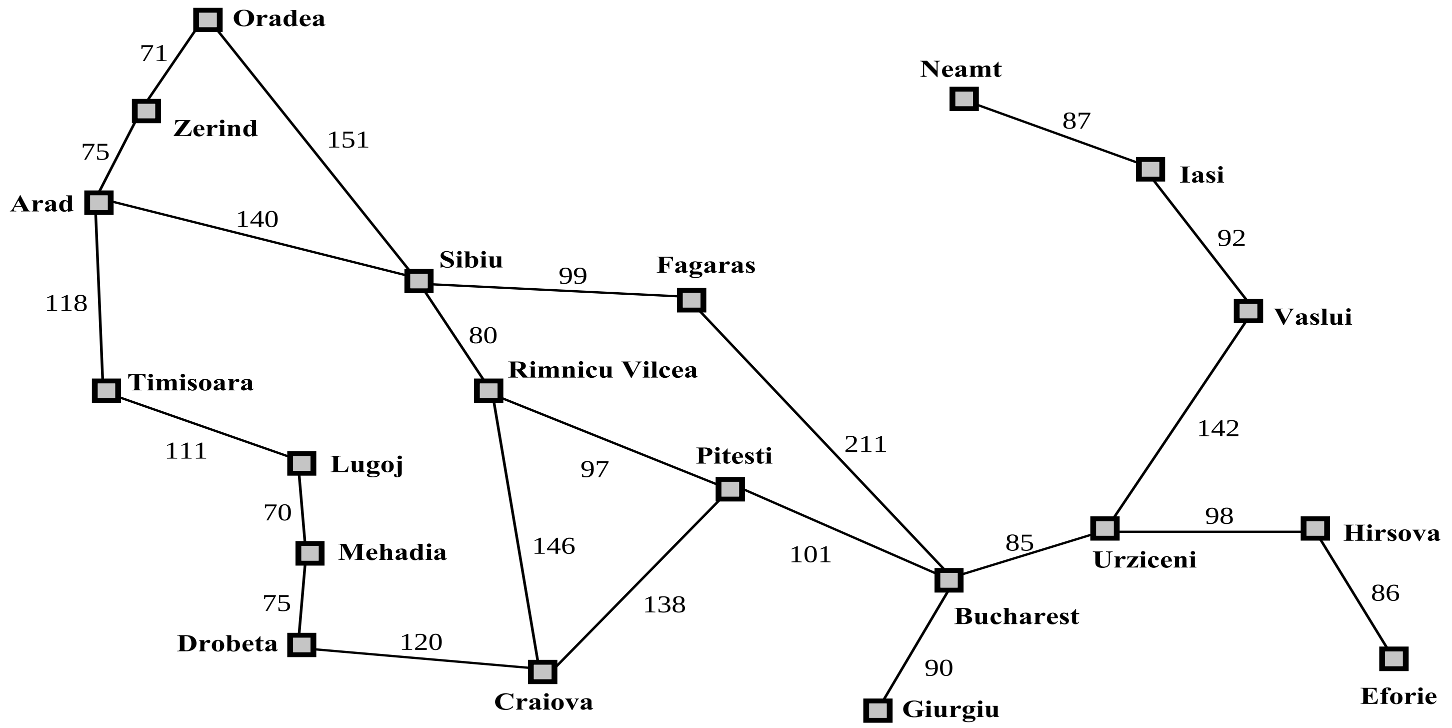
What is a Solution?

- A sequence of actions (a plan) which leads from the initial state into a goal state (e.g., the sequence of actions that gets the missionaries safely across the river)
- Or sometimes just the goal state (e.g., infer molecular structure from mass spectrographic data)

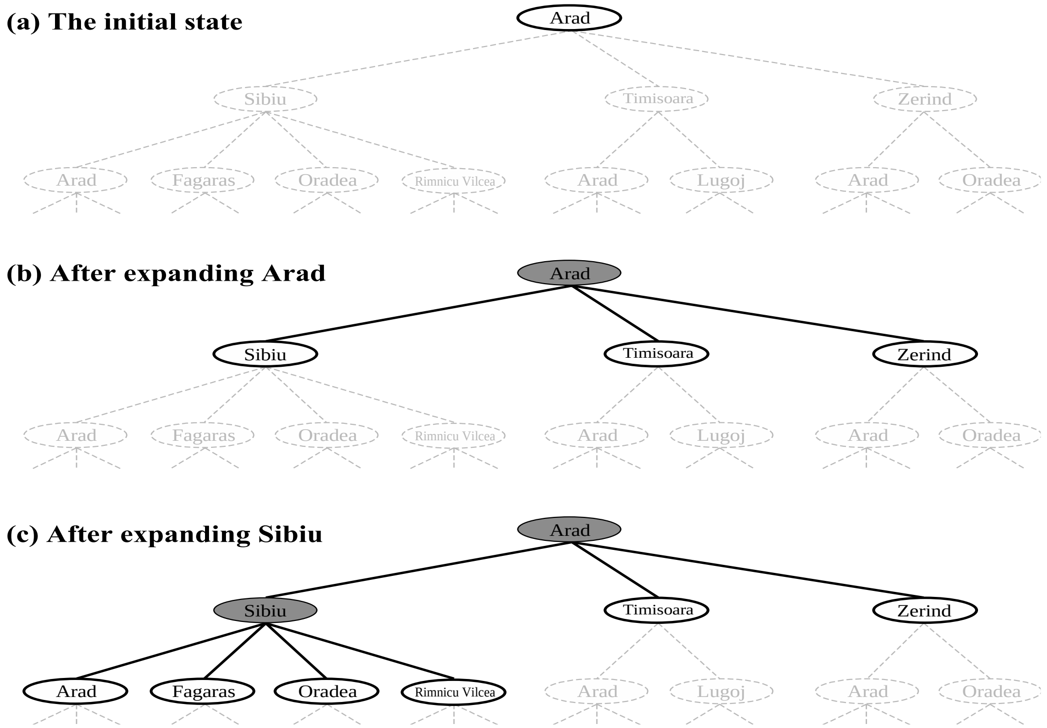
Our Current Framework

- Backtracking state-space search
- Others:
  - Constraint-based search
  - Optimization search
  - Adversarial search

State Space


Oradea
71
Zerind
151
75
Arad
140
118
Sibiu
99
Fagaras
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
75
Drobeta
120
138
Craiova
Neamt
87
Iasi
92
Vaslui
142
98
Hirsova
Urziceni
85
Bucharest
90
Giurgiu
86
Eforie

Search Trees



States vs. Nodes

- []

Generalized Search

```
Start by adding the initial state to a
    list, called fringe
Loop
    If there are no states left then fail
    Otherwise remove a node from fringe, cur
    If it's a goal state return it
    Otherwise expand it and add the resulting nodes to fringe
```

Expand a node = generate its successors

General Tree Search

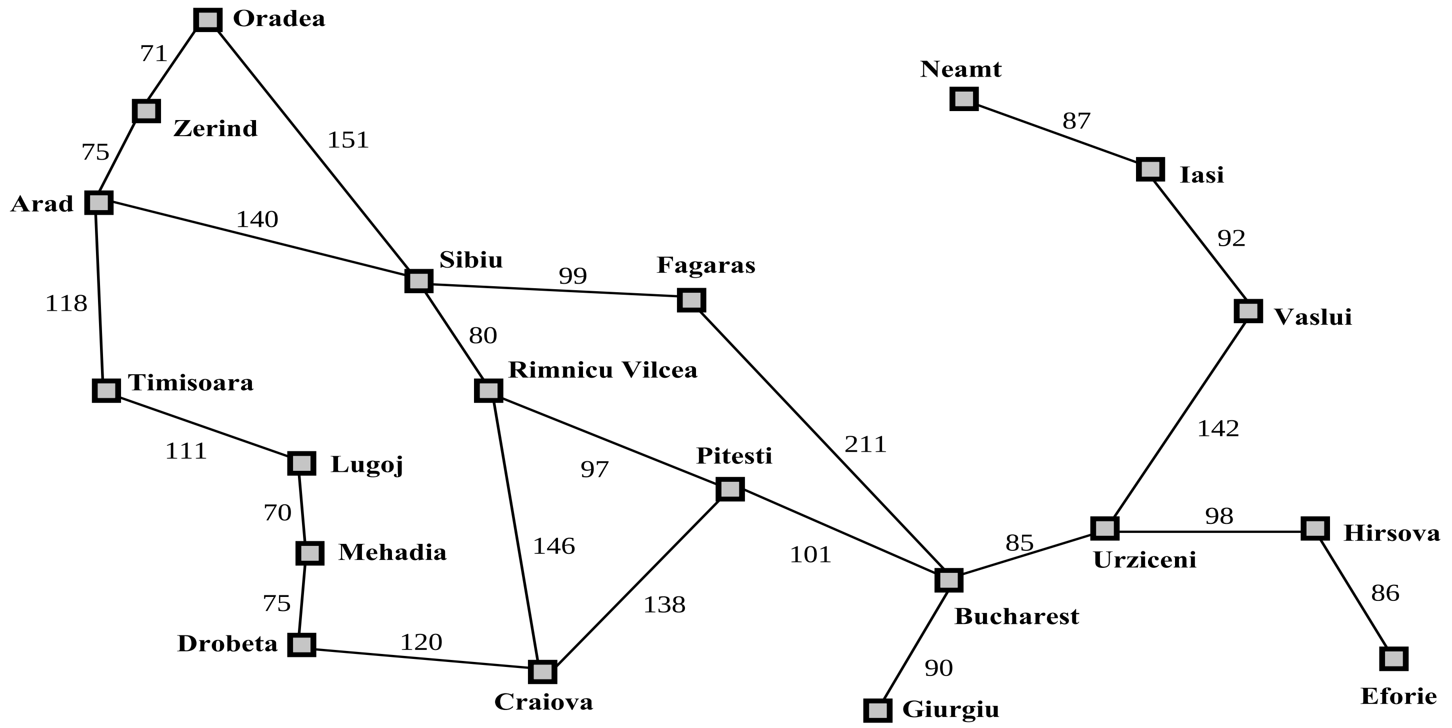
- Code on course webpage
- Style of code: simple, for class
  - Two versions; one simpler than the other.   These slides: the simpler one.
    - The simpler one requires input for the successors, and is only good for comparing depthfirst and breadthfirst search, and treesearch vs. graphsearch.
    - Look at it; if it isn't trivial to you, get up to speed before the next class (there is a link to a Python tutorial in the syllabus)
- The AIMA website has fully object-oriented code (Python, Java, etc)

```
def treesearch (qfun,fringe):
    while len(fringe) > 0:
        cur = fringe[0]
        fringe = fringe[1:]
        if goalp(cur): return cur
        fringe = qfun(makeNodes(cur,successors(cur)),fringe)
    return []
```

Blind Search Strategies

[breadth-first, uniform-cost, and depth-first search; evaluation criteria on next 3 slides; then iterative deepening search.]

State Space


Oradea
71
Zerind
75
151
Arad
140
Sibiu
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
70
Mehadia
75
Drobeta
120
Craiova
146
97
Pitesti
138
101
211
Bucharest
90
Giurgiu
85
Urziceni
98
Hirsova
86
Eforie
142
Vaslui
92
Iasi
87
Neamt

Evaluation Criteria

- Space
  - Maximum number of nodes in memory at one time
- Optimality
  - Does it always find a least-cost solution?
  - Cost considered: sum of the edgecosts of the path to the goal (the g-val)

Evaluation Criteria

- Completeness
  - Does it find a solution when one exists
- Time
  - The number of nodes generated during search

Time and Space Complexity Measured in Terms of

- b – maximum branching factor of the
  search tree; we will assume b is finite
- d – depth of the shallowest goal
- m – maximum depth of any path in the state space (may be infinite)

```
def graphsearch (qfun,fringe):
    expanded = { }
    while len(fringe) > 0:
        cur = fringe[0]
        fringe = fringe[1:]
        if goalp(cur):  return cur
        if not (expanded.has_key(cur.state) and\
                expanded[cur.state].gval <= cur.gval):
            expanded[cur.state] = cur
            fringe = qfun(makeNodes(cur,successors(cur)),fringe)
    return []
```

```
def depthLimSearch (fringe,depthlim):
    while len(fringe) > 0:
        cur = fringe[0]
        fringe = fringe[1:]
        if goalp(cur): return cur
        if cur.depth <= depthlim:
            fringe = makeNodes(cur,successors(cur)) + fringe
    return []
```

```
def iterativeDeepening(start):
   result = []
   depthlim = 1
   startnode = Node(start)
   while not result:
      result = depthLimSearch([startnode],depthlim)
      depthlim = depthlim + 1
   return result
```

```
def depthLimSearch (fringe,depthlim):
     while len(fringe) > 0:
         cur = fringe[0]
         fringe = fringe[1:]
         if goalp(cur): return cur
         if cur.depth <= depthlim:
             fringe = makeNodes(cur,successors(cur)) + fringe
     return []
```

Wrap Up

- Chapter 3.1-4
- Code under resources on the course webpage: simplesearch.py
- Notes (in response to questions asked in the past)
  - The book writes separate code for breadth-first search, i.e., they don't call treesearch as in the class code. Their version applies the goal test to a node when it is first generated, before it is added to the fringe. This can save time and space: In the worst case, when the goal is on the right frontier of the search tree, my version of breadth-first search generates, and adds to the fringe, an extra level of nodes than their version does. In figure 3.21 (time and space), breadth-first search is O(b^d) and uniform-cost search is O(b^(d+1)) if all edge-costs are equal.
  - For the exam, I won't be concerned with this complexity distinction. We are focusing on larger differences, such as whether the complexity is linear versus exponential and whether the exponent is d or m (which may be quite substantially different).
  - However (please see the following slide) …

Wrap Up

- However, we are focusing on the behavior of the algorithms
- Possible exam questions are:
  - Suppose we change tree search (or graph search) so that it applies the goal test to a node when it is first generated, before it is added to the fringe, and return immediately if the test is positive.
  - Is breadth-first search, uniform-cost search, (or another algorithm) still optimal? If so, explain why and list any conditions. If not, give a (small) counter example.

- As far as depth-first-search, there is a difference in the implementation from simplesearch.py and search.py: simplesearch.py (and these lecture notes) adds the results of the successor function to the fringe in one step; search.py adds them one by one. Specifically for depth-first search, this will affect whether the search proceeds down the left or the right of the tree. For the exam, and in class, I'll be clear about this difference if it comes up.
- From a theoretical point of view, in this framework for AI problem solving, the order of the successors is ignored, and not part of the distinctions between the algorithms. That is, there is no metric by which one successor can be judged better than another one.

THANK YOU