Uninformed search strategies (Section 3.4)
Uninformed search strategies

- A **search strategy** is defined by picking the order of node expansion
- **Uninformed** search strategies use only the information available in the problem definition
  - Breadth-first search
  - Depth-first search
  - Iterative deepening search
  - Uniform-cost search
**Breadth-first search**

- Expand shallowest unexpanded node
- Implementation: *frontier* is a FIFO queue

Example state space graph for a tiny search problem

Example from P. Abbeel and D. Klein
Breadth-first search

- Expansion order: 
  \((S,d,e,p,b,c,e,h,r,q,a,a, h,r,p,q,f,p,q,f,q,c,G)\)
Depth-first search

• Expand deepest unexpanded node
• Implementation: \textit{frontier} is a LIFO queue
Depth-first search

- Expansion order:
  \[(d, b, a, c, a, e, h, p, q, q, r, f, c, a, G)\]
**Preparing for a Date:**

What situations might I prepare for?
1) Medical emergency
2) Dancing
3) Food too expensive

**Okay, what kinds of emergencies can happen?**

1) Snakebite
2) Lightning strike
3) Fall from chair

**Hmm... which snakes are dangerous? Let's see...**

1) a) Corn snake
2) Garter snake
3) Copperhead

**The research comparing snake venoms is scattered and inconsistent. I'll make a spreadsheet to organize it.**

**I'm here to pick you up. You're not dressed?**

**By L&, the inland taipan has the deadliest venom of any snake!**

I really need to stop using depth-first searches.
BFS vs. DFS
BFS vs. DFS
BFS vs. DFS
BFS vs. DFS
BFS vs. DFS
BFS vs. DFS
BFS vs. DFS
BFS vs. DFS
BFS vs. DFS
BFS vs. DFS
BFS vs. DFS
BFS vs. DFS
BFS vs. DFS
BFS vs. DFS
BFS vs. DFS
BFS vs. DFS
BFS vs. DFS
BFS vs. DFS
Analysis of search strategies

• Strategies are evaluated along the following criteria:
  – **Completeness**: does it always find a solution if one exists?
  – **Optimality**: does it always find a least-cost solution?
  – **Time complexity**: number of nodes generated
  – **Space complexity**: maximum number of nodes in memory

• Time and space complexity are measured in terms of
  – \( b \): maximum branching factor of the search tree
  – \( d \): depth of the optimal solution
  – \( m \): maximum length of any path in the state space (may be infinite)
Properties of breadth-first search

- **Complete?**
  Yes (if branching factor $b$ is finite)

- **Optimal?**
  Yes – if cost = 1 per step

- **Time?**
  Number of nodes in a $b$-ary tree of depth $d$: $O(b^d)$
  ($d$ is the depth of the optimal solution)

- **Space?**
  $O(b^d)$

- Space is the bigger problem (more than time)
Properties of depth-first search

• **Complete?**
  
  Fails in infinite-depth spaces, spaces with loops
  Modify to avoid repeated states along path
  → complete in finite spaces

• **Optimal?**
  
  No – returns the first solution it finds

• **Time?**
  
  Could be the time to reach a solution at maximum depth \( m: O(b^m) \)
  Terrible if \( m \) is much larger than \( d \)
  But if there are lots of solutions, may be much faster than BFS

• **Space?**
  
  \( O(bm) \), i.e., linear space!
Iterative deepening search

• Use DFS as a subroutine

  1. Check the root
  2. Do a DFS searching for a path of length 1
  3. If there is no path of length 1, do a DFS searching for a path of length 2
  4. If there is no path of length 2, do a DFS searching for a path of length 3...
Iterative deepening search

Limit = 0
Iterative deepening search

Limit = 1
Iterative deepening search
Iterative deepening search

Limit = 3
Properties of iterative deepening search

• **Complete?**
  
  Yes

• **Optimal?**
  
  Yes, if step cost = 1

• **Time?**
  
  \((d+1)b^0 + d b^1 + (d-1)b^2 + \ldots + b^d\)

• **Space?**

  \(O(bd)\)
Search with varying step costs

- BFS finds the path with the fewest steps, but does not always find the cheapest path
Uniform-cost search

- For each frontier node, save the total cost of the path from the initial state to that node
- Expand the frontier node with the lowest path cost
- Implementation: \textit{frontier} is a priority queue ordered by path cost
- Equivalent to BFS if step costs all equal
- Equivalent to Dijkstra’s algorithm in general
Uniform-cost search example

- Expansion order: (S,p,d,b,e,a,r,f,e,G)
Properties of uniform-cost search

• **Complete?**
  Yes, if step cost is greater than some positive constant $\varepsilon$ (we don’t want infinite sequences of steps that have a finite total cost)

• **Optimal?**
  Yes
Optimality of uniform-cost search

- **Graph separation property**: every path from the initial state to an unexplored state has to pass through a state on the frontier
  - Proved inductively

- Optimality of UCS: proof by contradiction
  - Suppose UCS terminates at goal state $n$ with path cost $g(n)$ but there exists another goal state $n'$ with $g(n') < g(n)$
  - By the graph separation property, there must exist a node $n''$ on the frontier that is on the optimal path to $n'$
  - But because $g(n'') \leq g(n') < g(n)$, $n''$ should have been expanded first!
Properties of uniform-cost search

• **Complete?**
  Yes, if step cost is greater than some positive constant \( \varepsilon \) (we don’t want infinite sequences of steps that have a finite total cost)

• **Optimal?**
  Yes – nodes expanded in increasing order of path cost

• **Time?**
  Number of nodes with path cost \( \leq \) cost of optimal solution \((C^*)\), \(O(b^{C^*}/\varepsilon)\)
  This can be greater than \(O(b^d)\): the search can explore long paths consisting of small steps before exploring shorter paths consisting of larger steps

• **Space?**
  \(O(b^{C^*}/\varepsilon)\)
## Review: Uninformed search strategies

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete?</th>
<th>Optimal?</th>
<th>Time complexity</th>
<th>Space complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>Yes</td>
<td>If all step costs are equal</td>
<td>O(b^d)</td>
<td>O(b^d)</td>
</tr>
<tr>
<td>DFS</td>
<td>No</td>
<td>No</td>
<td>O(b^m)</td>
<td>O(bm)</td>
</tr>
<tr>
<td>IDS</td>
<td>Yes</td>
<td>If all step costs are equal</td>
<td>O(b^d)</td>
<td>O(bd)</td>
</tr>
<tr>
<td>UCS</td>
<td>Yes</td>
<td>Yes</td>
<td>Number of nodes with g(n) ≤ C*</td>
<td></td>
</tr>
</tbody>
</table>

- **b**: maximum branching factor of the search tree
- **d**: depth of the optimal solution
- **m**: maximum length of any path in the state space
- **C\***: cost of optimal solution
- **g(n)**: cost of path from start state to node n
Attribution

Slides developed by Svetlana Lazebnik based on content from Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, 3rd edition