To DFS / BFS – Deprived

● If you do not know what DFS or BFS are, and Ch. 3 is not enough of a review for you, please see the TAs ASAP
  ○ CS173 (Prereq.) covers BFS, DFS in detail
  ○ Alternative classes in ECE should have covered those (let me know if not)
Problem-Solving Agent
Problem-Solving Agent

- Formulate Goal
- Formulate Problem
  - States
  - Actions
- Find Solution
Example Problem

Start Street

Street with Parking
Looking for Parking

- Going home; need to find street parking
- Formulate Goal:
  - Car is parked
- Formulate Problem:
  - States: street with parking and car at that street
  - Actions: drive between street segments
- Find solution:
  - Sequence of street segments, ending with a street with parking
Problem Formulation

Start Street

Street with Parking

Search Path
Search Problem

● State space
  ○ each state is an abstract representation of the environment
  ○ the state space is discrete

● Initial state

● Successor function

● Goal test

● Path cost
Search Problem

- State space
- **Initial state:**
  - usually the current state
  - sometimes one or several hypothetical states (“what if …”)
- Successor function
- Goal test
- Path cost
Search Problem

- State space
- Initial state
- **Successor function:**
  - [state ⊆ subset of states]
  - an abstract representation of the possible actions
- Goal test
- Path cost
Search Problem

- State space
- Initial state
- Successor function
- **Goal test:**
  - usually a condition
  - sometimes the description of a state
- Path cost
Search Problem

- State space
- Initial state
- Successor function
- Goal test

**Path cost:**
- [path △ positive number]
- usually, path cost = sum of step costs
- e.g., number of moves of the empty tile
Assumptions in Basic Search

- The environment is static
- The environment is discretizable
- The environment is observable
- The actions are deterministic
Search Space Size

● Unlike Search in CS225, AI encounters search spaces that are too large
● AI Search typically does not realize the entire search graph or state space
● Examples
  o Scheduling CS classes such that every student in every program of study can take every class they wish
  o Search for shortest path that covers all streets (Travelling Salesman Problem)
Search Space Size

- Scheduling CS classes such that every student in every program of study can take every class they wish

- States = ?
- State Space Size = ?
- Search Time = ?
Search Space Size

- Search for shortest path that covers all streets (Travelling Salesman Problem)

- State = ?
- State Space Size = ?
- Search Time = ?
Simple Agent Algorithm

Problem-Solving-Agent

1. initial-state □ sense/read state
2. goal □ select/read goal
3. successor □ select/read action models
4. problem □ (initial-state, goal, successor)
5. solution □ search(problem)
6. perform(solution)
Basic Search Concepts

- Search tree
- Search node
- Node expansion
- **Search strategy**: At each stage it determines which node to expand
Node Data Structure

- STATE
- PARENT
- ACTION
- COST
- DEPTH

If a state is too large, it may be preferable to only represent the initial state and (re-)generate the other states when needed.
Fringe

- Set of search nodes that have not been expanded yet
- Implemented as a queue FRINGE
  - INSERT(node,FRINGE)
  - REMOVE(FRINGE)
- The ordering of the nodes in FRINGE defines the search strategy
Search Algorithm

1. If GOAL?(initial-state) then return initial-state
2. INSERT(initial-node,FRINGE)
3. Repeat:
   - If FRINGE is empty then return failure
   - n □ REMOVE(FRINGE)
   - s □ STATE(n)
   - For every state s’ in SUCCESSORS(s)
     ■ Create a node n’
     ■ If GOAL?(s’) then return path or goal state
     ■ INSERT(n’,FRINGE)
Search Strategies

- A strategy is defined by picking the order of node expansion

- **Performance Measures:**
  - Completeness – does it always find a solution if one exists?
  - Time complexity – number of nodes generated/expanded
  - Space complexity – maximum number of nodes in memory
  - Optimality – does it always find a least-cost solution

- Time and space complexity are measured in terms of
  - \( b \) – maximum branching factor of the search tree
  - \( d \) – depth of the least-cost solution
  - \( m \) – maximum depth of the state space (may be \( \infty \))
Remark

- Some problems formulated as search problems are NP-hard problems. We cannot expect to solve such a problem in less than exponential time in the worst-case.
- But we can nevertheless strive to solve as many instances of the problem as possible.
Blind vs. Heuristic Strategies

- **Blind** (or uninformed) strategies do not exploit any of the information contained in a state.
- **Heuristic** (or informed) strategies exploits such information to assess that one node is “more promising” than another.
Blind Strategies

- **Breadth-first**
  - Bidirectional

- **Depth-first**
  - Depth-limited
  - Iterative deepening

- **Uniform-Cost**

  - Step cost = 1

  - Step cost = $c(\text{action}) > 0$
Breadth-First Strategy

New nodes are inserted at the end of FRINGE

FRINGE = (1)
Breadth-First Strategy

New nodes are inserted at the end of FRINGE

FRINGE = (2, 3)
Breadth-First Strategy

New nodes are inserted at the end of FRINGE

FRINGE = (3, 4, 5)
Breadth-First Strategy

New nodes are inserted at the end of FRINGE

FRINGE = (4, 5, 6, 7)
Evaluation

- **b**: branching factor
- **d**: depth of shallowest goal node
- Complete
- Optimal if step cost is 1
- Number of nodes generated:
  \[1 + b + b^2 + \ldots + b^d + b(b^d-1) = \mathcal{O}(b^{d+1})\]
- Time and space complexity is \(\mathcal{O}(b^{d+1})\)
# Time and Memory Requirements

<table>
<thead>
<tr>
<th>d</th>
<th>#Nodes</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>111</td>
<td>0.01 msec</td>
<td>11 Kbytes</td>
</tr>
<tr>
<td>4</td>
<td>11,111</td>
<td>1 msec</td>
<td>1 Mbyte</td>
</tr>
<tr>
<td>6</td>
<td>$\sim10^6$</td>
<td>1 sec</td>
<td>100 Mb</td>
</tr>
<tr>
<td>8</td>
<td>$\sim10^8$</td>
<td>100 sec</td>
<td>10 Gbytes</td>
</tr>
<tr>
<td>10</td>
<td>$\sim10^{10}$</td>
<td>2.8 hours</td>
<td>1 Tbyte</td>
</tr>
<tr>
<td>12</td>
<td>$\sim10^{12}$</td>
<td>11.6 days</td>
<td>100 Tbytes</td>
</tr>
<tr>
<td>14</td>
<td>$\sim10^{14}$</td>
<td>3.2 years</td>
<td>10,000 Tb</td>
</tr>
</tbody>
</table>

Assumptions: $b = 10$; 1,000,000 nodes/sec; 100 bytes/node
## Time and Memory Requirements

<table>
<thead>
<tr>
<th>$d$</th>
<th>#Nodes</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>111</td>
<td>0.01 msec</td>
<td>11 Kbytes</td>
</tr>
<tr>
<td>4</td>
<td>11,111</td>
<td>1 msec</td>
<td>1 Mbyte</td>
</tr>
<tr>
<td>6</td>
<td>$\sim 10^6$</td>
<td>1 sec</td>
<td>100 Mb</td>
</tr>
<tr>
<td>8</td>
<td>$\sim 10^8$</td>
<td>100 sec</td>
<td>10 Gbytes</td>
</tr>
<tr>
<td>10</td>
<td>$\sim 10^{10}$</td>
<td>2.8 hours</td>
<td>1 Tbyte</td>
</tr>
<tr>
<td>12</td>
<td>$\sim 10^{12}$</td>
<td>11.6 days</td>
<td>100 Tbytes</td>
</tr>
<tr>
<td>14</td>
<td>$\sim 10^{14}$</td>
<td>3.2 years</td>
<td>10,000 Tb</td>
</tr>
</tbody>
</table>

Assumptions: $b = 10$; 1,000,000 nodes/sec; 100 bytes/node
Bidirectional Strategy

2 fringe queues: FRINGE1 and FRINGE2

Time and space complexity = $O(b^{d/2}) << O(b^d)$
Depth-First Strategy

New nodes are inserted at the front of FRINGE

FRINGE = (1)

1

2

3

4

5

(1)
Depth-First Strategy

New nodes are inserted at the front of FRINGE

FRINGE = (2, 3)
Depth-First Strategy

New nodes are inserted at the front of FRINGE

FRINGE = (4, 5, 3)
Depth-First Strategy

New nodes are inserted at the front of FRINGE
Depth-First Strategy

New nodes are inserted at the front of FRINGE
Depth-First Strategy

New nodes are inserted at the front of FRINGE
Depth-First Strategy

New nodes are inserted at the front of FRINGE
Depth-First Strategy

New nodes are inserted at the front of FRINGE
Depth-First Strategy

New nodes are inserted at the front of FRINGE
Depth-First Strategy

New nodes are inserted at the front of FRINGE
Depth-First Strategy

New nodes are inserted at the front of FRINGE
Evaluation

- **b**: branching factor
- **d**: depth of shallowest goal node
- **m**: maximal depth of a leaf node
- Complete only for finite search tree
- Not optimal
- Number of nodes generated:
  \[1 + b + b^2 + \ldots + b^m = O(b^m)\]
- Time complexity is \(O(b^m)\)
- Space complexity is \(O(b^m)\)
Depth-Limited Strategy

- Depth-first with depth cutoff $k$ (maximal depth below which nodes are not expanded)

- Three possible outcomes:
  - Solution
  - Failure (no solution)
  - Cutoff (no solution within cutoff)
Iterative Deepening Strategy

Repeat for \( k = 0, 1, 2, \ldots \):
- Perform depth-first with depth cutoff \( k \)
  - Complete
  - Optimal if step cost =1
  - Time complexity is:
    \[
    (d+1)(1) + db + (d-1)b^2 + \ldots + (1) b^d = O(b^d)
    \]
  - Space complexity is: \( O(bd) \)
Comparison of Strategies

- Breadth-first is complete and optimal, but has high space complexity.
- Depth-first is space efficient, but neither complete nor optimal.
- Iterative deepening is asymptotically optimal.
Repeated States

- Queens
- Assembly planning
- Man
- Few

Search tree is finite

Search tree is infinite

8-puzzle and robot navigation
Avoiding Repeated States

- Requires comparing state descriptions
- Breadth-first strategy:
  - Keep track of all generated states
  - If the state of a new node already exists, then discard the node
Avoiding Repeated States

● Depth-first strategy:
  o Solution 1:
    ■ Keep track of all states associated with nodes in current tree
    ■ If the state of a new node already exists, then discard the node
    □ Avoids loops
  o Solution 2:
    ■ Keep track of all states generated so far
    ■ If the state of a new node has already been generated, then discard the node
    □ Space complexity of breadth-first
Detecting Identical States

- Use explicit representation of state space

- Use hash-code or similar representation
Uniform-Cost Strategy

- Each step has some cost $> 0$.
- The cost of the path to each fringe node $N$ is $g(N) =$ costs of all steps.
- The goal is to generate a solution path of minimal cost.
- The queue FRINGE is sorted in increasing cost.
Modified Search Algorithm

1. INSERT(initial-node,FRINGE)

2. Repeat:
   - If FRINGE is empty then return failure
   - n □ REMOVE(FRINGE)
   - s □ STATE(n)
   - If GOAL?(s) then return path or goal state
   - For every state s’ in SUCCESSORS(s)
     ■ Create a node n’
     ■ INSERT(n’,FRINGE)
Branch and Bound

- If search involves optimization (e.g. shortest path for covering all streets)
  - Can record the best path so far, and bound the search when the current path becomes longer than current best path.
Summary

- Search tree state space
- Search strategies: breadth-first, depth-first, and variants
- Evaluation of strategies: completeness, optimality, time and space complexity
- Avoiding repeated states
- Optimal search with variable step costs