Administrative

- What to do if teammates drop?
  - Still have 3 or more? No problem... keep going.
  - Have two or fewer *and* want to be merged?
    - We’ll do what we can.

- Submitting problem sets
  
  **TO:**  
eecs492-staff@april.eecs.umich.edu

  **SUBJECT:**  
  PS1: <uniqname1>, <uniqname2>, <uniqname3>. challenge=<challenge score>

  **CC:**  
  <all of your team members>
General Tree Search

**Function** `Tree-search(problem)`
- **Returns** a solution, or failure

```plaintext
fringe = new Queue();
fringe.put(problem.initialState)

loop do
  if fringe.isEmpty() then return failure
  node ← fringe.get()
  if problem.isGoalState(node)
    then return node;
  fringe.putAll(problem.expand(node))
```

Which node in the fringe does `get()` return?

Breadth-First Search (BFS)

- General tree-search where queue is **first-in-first-out** (FIFO)
  - `get` operation returns oldest item on fringe
  - Corresponds to a **level-order** traversal of search tree
  - All nodes at level `d` expanded before any at `d+1`
BFS in Vacuum World

Actions:
L – move Left
R – move Right
S – Suck up the dirt

Starting in state 1, generate search tree.

Vacuum World BFS

Are we done?
Hurray!
Search Performance Criteria: BFS

- **Completeness**: Guaranteed to find a solution if it exists?
  - YES

- **Optimality**: Does the strategy find the minimum path cost solution?
  - YES if uniform action cost

- **Time Complexity**: How long does it take to find a solution?

- **Space Complexity**: How much memory is needed to perform the search?

BFS Time Complexity

- Assume uniform search space
  - Each node has same # successors
  - branching factor, $b$

- $d$: Depth of shallowest solution

- BFS generates
  - complete search trees at depth $\leq d$
  - Worst case: $b^{d+1}$ nodes at depth $d+1$
  - Total nodes:

$$b + b^2 + b^3 + ... + b^d + b^{d+1} = O(b^{d+1})$$
BFS Space Complexity

- Same as time complexity: $O(b^{d+1})$
- Must store entire fringe: all nodes at deepest level
- In typical computer configurations, will run out of space before running out of time

Uniform-Cost Search (UCS)

- General tree-search where queue is priority-first
  - get operation returns least-cost item on fringe
    - All nodes at cost less than $c$ expanded before any at cost $c$
      - Same as BFS if all actions have same cost, different otherwise
UCS for Route Planning

Are we done?

UCS Properties

- Complete?
  - Yes

- Time and space complexity
  - Uniform cost:
    - Same as BFS: $O(b^{d+1})$
  - In general, with minimal path cost $C^*$ and minimal action cost $\varepsilon$:
    - $O(b^{C^*/\varepsilon + 1})$

- Optimal as long as cost monotonic along path
  - Guaranteed by each edge cost $\geq 0$
Depth-First Search (DFS)

- General tree-search where queue is last-in-first-out (LIFO, aka stack)
  - get operation returns newest item on fringe
    - Often implemented recursively using function-call stack
  - Always expand deepest node on fringe

Vacuum World DFS

Uh oh!
Performance of Depth-First

- **Completeness**
  - Only for finite depth trees, and so in general: No

- **Optimality**
  - No

- **Time Complexity**
  - $O(b^m)$; where $m$ is the maximum depth of tree

- **Space Complexity**
  - $O(bm)$

Depth-Limited Search

- **Problem**: unbounded trees in DFS

- **Solution**:
  - Predetermine a depth limit $L$
  - Run DFS, cut off search at depth $L$

- **Complete iff exists solution within $L$ steps**

- **Optimal?**

- **Time complexity**: $O(b^L)$

- **Space complexity**: $O(bL)$

- **How do we choose $L$?**
Iterative Deepening DFS

function Iterative-deepening-search(problem)
    returns a solution, or failure

loop for depth from 0 to infinity
    if Depth-limited-search(problem, depth) succeeds
        then return its result
    end loop
return failure

IDS Properties

- Complete?
- Optimal?
- Time Complexity?
- Space Complexity?
IDS Complexity

- Time is same as running DLS for depth $= 1, \ldots, d$

$$\sum_{l=1}^{d} O(b^l) = O(b) + O(b^2) + \cdots + O(b^d) = O\left(b^d\right)$$

- Space is same as DLS for depth $d$: $O(bd)$

Analysis Summary

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Uniform-Cost</th>
<th>Depth-First</th>
<th>Depth-Limited</th>
<th>Iterative Deepening</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Time complexity</td>
<td>$O(b^{d+1})$</td>
<td>$O\left(b^{C^*+1}\right)$</td>
<td>$O(b^m)$</td>
<td>$O(b^L)$</td>
<td>$O\left(b^d\right)$</td>
</tr>
<tr>
<td>Space complexity</td>
<td>$O(b^{d+1})$</td>
<td>$O\left(b^{C^*+1}\right)$</td>
<td>$O(bm)$</td>
<td>$O(bL)$</td>
<td>$O(bd)$</td>
</tr>
</tbody>
</table>

Exponential in depth!  
Linear in depth!
Your turn!

- Show that:
  \[ b + b^2 + b^3 + \ldots + b^n = O(b^n) \]

- 7 Queens:
  - Goal: Place 7 queens on 7x7 chess board so that no two attack each other
  - Formulate the problem carefully
    - What state space?
    - What actions do you consider at each step?
  - Which search strategy to use?
    - Find a solution using your strategy.

7 queens, DFS
The importance of problem formulation

- Problem formulation has huge impact on complexity
  - State space
  - Actions

- Consider 7-queens problem:
  - Naïve state space: branching factor of roughly ~49
  - One queen per column: branching factor of 7.

Path Planning Example

- Consider an agent trying to find the best route from one place to another.
  - Actions = \{N, S, E, W\}
  - DFS is out. (Why?)

- The shortest path is 50 moves.
  - Complexity of BFS/IDS?

- How many distinct states are there?
  - $25^2 << 4^{50}$
  - What are we doing wrong?
    - Repeated states!
Avoiding repeated states

- Idea: don’t re-expand nodes that we’ve already expanded.
  - Closed list: set of all states previously visited
  - Memory usage?

General Tree Search

```javascript
function Tree-search(problem)
returns a solution, or failure

closed = new Set();
closed.add(problem.initialState);

fringe = new Queue();
fringe.put(problem.initialState);

loop do
  if fringe.isEmpty() then return failure
  node = fringe.get()
  if problem.isGoalState(node)
    then return node;
  for each child in problem.expand(node)
    if !closed.contains(child)
      closed.add(child)
      fringe.put(child)
```
General Tree Search: Analysis

- Time complexity?
  - O(|V| + |E|)
- Memory complexity?
  - O(|V|)
- Optimality?
  - BFS/IDS?
  - DFS?

- What about DFS’s infinite loop problem?
  - Fixed for finite worlds!

- Can we guarantee optimality, regardless of the policy?
  - For finite worlds, if we replace old bad paths with new good paths rather than discarding the new good paths.
  - Some bookkeeping...

Graph Search: Summary

- Graph Search
  - Maintains list of states already visited
  - Terminates searches that revisit the same states.

- When do we want to use graph search?
  - When repeated states are likely
  - We can afford the memory
Preview: Informed Search

- Employ substantive criteria about node states in deciding which to expand

- Questions:
  - What criteria?
  - How to exploit?
  - Properties?