A problem solving agent is one which decides what actions and states to consider in completing a goal.

Examples:
- Finding the shortest path from one city to another
- 8-puzzle
Example: The 8-puzzle

Start State

Goal State
Problem Solving Agents

- 8-Puzzle
  - Action: Move blank square up, down, left, or right
  - State: arrangement of squares including blank
  - Goal Test: Is it in correct order?
  - Cost: 1 for each move it took
Problem Statement

- How do I get from initial state to goal?
  - Use a search algorithm
- From initial state, generate more states
- Choose one of the adjacent states, and expand the state from there
- Continue until you find solution
- Search strategy: Which states do you expand first?
- Picture of 8-puzzle search tree
Distinction between state and node

- **State**: configuration of the world
  - What is the arrangement of tiles in the puzzle?

- **Node**: data structure within the search tree.

- # nodes > # states

- Much of the time, the tree is “virtual”. 
Uninformed Search Strategies

- Uninformed: No knowledge of whether one state is better than another, except for goal
  - How could knowledge help? (Rubik’s cube?)
  - Informed search uses heuristics

- Interested in following:
  - Completeness: Solution guaranteed if it exists
  - Time complexity: How long?
  - Space complexity: How much memory?
  - Optimality: Does it find best solution (lowest cost)? Does it find it first?
Breadth-first search

- Overview
  - Expand root node
  - Expand all children of root node
  - Expand all grandchildren, etc.

- In general
  - Expand all nodes at depth $d$ before expanding nodes at depth $d+1$
Breadth-First Analysis

- Completeness?
- Optimality?
- Time and space?
  - Let \( b \) = branching factor: maximum number of branches a given node can yield
  - What is branching factor for 8-puzzle?
Breadth-First Search

- **Time complexity**: How many nodes do I expand? If solution at depth \( d \), approx

\[
1 + b + b^2 + b^3 + b^4 + \ldots + b^d = \frac{b^{d+1} - 1}{b - 1} = O(b^d)
\]

- **Space complexity**: \( O(b^d) \)
  - Same thing: Need to maintain all prior nodes so you know path
  - Usually implemented with a queue
Uniform Cost Search

- Similar to Breadth-First search, but expand cheapest path so far
- Example: Finding shortest distance to a city
Uniform Cost Search

- Completeness:
- Complexity:
- Optimality:
Depth-First Search

- Expand root node
- Expand node at deepest level of tree
- Repeat
Depth
First
Depth-First Search

- **Space complexity:**
  - Must store all nodes on current path
  - Must store all unexplored sibling nodes on path
  - At depth $m$, required to store $1+bm$ nodes
    - $(1+m$ if can each node remembers what to expand next)
  - $O(bd)$: Much better than $O(b^d)$

- **Time complexity:**
  - Still need to explore all nodes: $O(b^d)$
  - Depth-first can get lucky and find long path quickly
  - Depth-first can get “lost” down a really long path
Depth-First Search

- **Complete**
  - No – if tree is infinite, could spend forever exploring one branch even if there is a finite solution somewhere

- **Optimality**
  - Might never find any solutions

- Usually implemented via recursion
Depth-Limited Search

- Depth-First search, but limit maximum depth allowed
- Complete:
- Optimality:
- Time complexity:
- Space complexity:
Iterative Deepening Search

- Depth-limited search, with
  - depth = 0
  - then again with depth = 1
  - then again with depth = 2
  - ... until you find a solution
Iterative Deepening

Depth Limit = 0

Depth = 0
Iterative Deepening

Depth = 0

Depth Limit = 1
Iterative Deepening

Depth Limit = 2

Depth = 0
Depth = 1
Depth = 2
Iterative Deepening Search

- Why iterative deepening search?
  - Complete: eventually will find solution

- Why not use BFS?
  - BFS traverses in same order, apart from repeats.
  - Aren’t repeats inefficient?
Iterative Deepening Search

- Memory requirements are same as those as DFS: $O(bd)$ instead of $O(b^d)$
- Can think of it as BFS where store less info, and rediscover it when you need it
  - Completeness and optimality the same as for BFS
- How much time do you lose due to repeats?
  - It’s not so bad, since you don’t repeat the bottom levels as much (the big ones)
- See book or CS 227 (Alg II) for details
Start searching forward from initial state and backwards from goal, and try to meet in the middle

Should reduce from $O(b^d)$ to $O(2b^{d/2}) = O(b^{d/2})$

- Hash table to track where you’ve been
- Can you search backwards? Depends on goal (is goal unique?)
Avoiding Repeated States

- How do you make sure you don’t cycle?
- Need to store all the states you have already been to: lots of memory! \(O(b^d)\)
- Checking would only pay off if space has lots of cycles
- Hash table usually used to make lookup efficient
Searching in Partially Observable Environments

- What if I have no sensors at all, apart from “at goal”?
  - I have a belief state, which is a set of all states I might be in.
    - {at home, on dirt, etc.}
  - Search through all possible belief states, until you reach one that contains only goals