Uninformed Search

Reading: Chapter 4 (Tuesday, 2/5)
HW#1 due next Tuesday
Uninformed

- Search through the space of possible solutions
- Use no knowledge about which path is likely to be best
- Exception: uniform cost
  - Each path is given a cost
Before actually doing something to solve a puzzle, an intelligent agent explores possibilities “in its head”

- Human vs. rational?
- For what games, do we do this?

Search = “mental exploration of possibilities”

Making a good decision requires exploring several possibilities

Execute the solution once it’s found
Formulating Problems as Search

Given an initial state and a goal, find the sequence of actions leading through a sequence of states to the final goal state.

Terms:

- **Successor function**: given action and state, returns \{action, successors\}
- **State space**: the set of all states reachable from the initial state
- **Path**: a sequence of states connected by actions
- **Goal test**: is a given state the goal state?
- **Path cost**: function assigning a numeric cost to each path
- **Solution**: a path from initial state to goal state
Example: the 8-puzzle

- How would you use AI techniques to solve the 8-puzzle problem?
8-puzzle URLs

- http://www.permadi.com/java/puzzle8
8 Puzzle

- **States**: integer locations of tiles
  - (0 1 2 3 4 5 6 7 8)
  - (0 1 2)(3 4 5) (6 7 8)
- **Action**: left, right, up, down
- **Goal test**: is current state = (0 1 2 3 4 5 6 7 8)?
- **Path cost**: same for all paths
- **Successor function**: given {up, (5 2 3 0 1 8 4 7 6)} -> ?
- What would the state space be for this problem?
What are we searching?

- **State space vs. search space**
  - State represents a *physical* configuration
  - Search space represents a tree/graph of *possible solutions… an abstract configuration*

- **Nodes**
  - Abstract data structure in search space
  - Parent, children, depth, path cost, associated state

- **Expand**
  - A function that given a node, creates all children nodes, using *successor function*
Uninformed Search Strategies

The strategy gives the order in which the search space is searched

- Breadth first
- Depth first
- Depth limited search
- Iterative deepening
- Uniform cost
**Implementation: general tree search**

```plaintext
function TREE-SEARCH(problem, fringe) returns a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST[problem] applied to STATE(node) succeeds return node
    fringe ← INSERTALL(EXPAND(node, problem), fringe)

function EXPAND(node, problem) returns a set of nodes
  successors ← the empty set
  for each action, result in SUCCESSOR-Fn[problem](STATE[node]) do
    s ← a new NODE
    PARENT-NODE[s] ← node; ACTION[s] ← action; STATE[s] ← result
    PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
    DEPTH[s] ← DEPTH[node] + 1
    add s to successors
  return successors
```
Breadth-first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end
Breadth-first search

Expand shallowest unexpanded node

Implementation:

\( \text{fringe} \) is a FIFO queue, i.e., new successors go at end
Breadth-first search

Expand shallowest unexpanded node

Implementation:

*fringe* is a FIFO queue, i.e., new successors go at end
Breadth-first search

Expand shallowest unexpanded node

Implementation:

*fringe* is a FIFO queue, i.e., new successors go at end
A different view of the same problem for visualization: 

**Breadth first**

- OPEN = start node; CLOSED = empty
- While OPEN is not empty do
  - Remove leftmost state from OPEN, call it X
  - If X = goal state, return success
  - Put X on CLOSED
  - SUCCESSORS = Successor function (X)
  - Remove any successors on OPEN or CLOSED
  - Put remaining successors on right end of OPEN
- End while
Visualization
A different view of the same problem for visualization: Depth-first

- OPEN = start node; CLOSED = empty
- While OPEN is not empty do
  - Remove leftmost state from OPEN, call it X
  - If X = goal state, return success
  - Put X on CLOSED
  - SUCCESSORS = Successor function (X)
  - Remove any successors on OPEN or CLOSED
  - Put remaining successors on left end of OPEN

- End while
Depth-first search

Expand deepest unexpanded node

Implementation:

\( fringe = \text{LIFO queue, i.e., put successors at front} \)
Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front
Depth-first search

Expand deepest unexpanded node

Implementation:

\[ fringe = \text{LIFO queue, i.e., put successors at front} \]
Depth-first search

Expand deepest unexpanded node

Implementation:

\[ fringe = \text{LIFO queue, i.e., put successors at front} \]
**Depth-first search**

Expand deepest unexpanded node

**Implementation:**

$fringe = \text{LIFO queue, i.e., put successors at front}$
Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe = \text{LIFO queue, i.e., put successors at front}$
Depth-first search

Expand deepest unexpanded node

Implementation:

\[ \text{fringe} = \text{LIFO queue}, \text{i.e., put successors at front} \]
Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front
Complexity Analysis

- **Completeness**: is the algorithm guaranteed to find a solution when there is one?
- **Optimality**: Does the strategy find the optimal solution?
- **Time**: How long does it take to find a solution?
- **Space**: How much memory is needed to perform the search?
**Cost variables**

- **Time**: number of nodes generated
- **Space**: maximum number of nodes stored in memory
- **Branching factor**: $b$
  - Maximum number of successors of any node
- **Depth**: $d$
  - Depth of shallowest goal node
- **Path length**: $m$
  - Maximum length of any path in the state space
### Complexity

**BFS vs. DFS**

- **Optimal**
  - Time = $1 + b + b^2 + b^3 + \ldots + b^d + (b^{d+1} - 1)$
  - $= O(b^{d+1})$

- **Space** = $O(b^{d+1})$
  - Space is the big problem

- **Complete if $b$ is finite**

- **Not optimal**
  - Time = $1 + b + b^2 + b^3 + \ldots + b^m$
  - $= O(b^m)$

- **Space** = $O(b^m)$
  - Space is linear

- **Complete in finite spaces; fails in infinite depth**
Can we combine benefits of both?

- Depth limited
  - Select some limit in depth to explore the problem using DFS
  - How do we select the limit?

- Iterative deepening
  - DFS with depth 1
  - DFS with depth 2 up to depth d
Iterative deepening search $l = 1$

Limit = 1

Diagram showing the iterative deepening search process.
Iterative deepening search $l = 2$

Limit = 2
Iterative deepening search $l = 3$

Limit = 3
Three types of incompleteness

- Sensorless problems
- Contingency problems
  - Adversarial problems
- Exploration problems