INTRODUCTION TO ARTIFICIAL INTELLIGENCE

EPISODE 2
TODAY’S MENU

1. SEARCH

2. BFS, DFS

3. INFORMED SEARCH: A*
SEARCH
BREADTH-FIRST SEARCH

SHORTEST PATH: BACKTRACK FROM THE GOAL
BREADTH-FIRST SEARCH

queue: [A] (green nodes)

active →

A − B − C − D − E − F
BREADTH-FIRST SEARCH

queue: □ (green nodes)

active

(visited = orange)
BREADTH-FIRST SEARCH

queue: [B, C, D] (green nodes)
BREADTH-FIRST SEARCH

queue: \[ C, D \] (green nodes)
BREADTH-FIRST SEARCH

queue: [D] (green nodes)
BREADTH-FIRST SEARCH

queue: \([D,F]\) (green nodes)
BREADTH-FIRST SEARCH

queue: [F] (green nodes)
BREADTH-FIRST SEARCH

queue: [ ] (green nodes)
BREADTH-FIRST SEARCH

queue: [E] (green nodes)
BREADTH-FIRST SEARCH

queue  (green nodes)
DEPTH-FIRST VS BREADTH-FIRST

• In breadth-first search (BFS), the node list is a **queue**: first-in, first-out

• In depth-first search (DFS), the node list is a **stack**: last-in, first-out

• Other search algorithms can be obtained by replacing the list with yet other structures

• "Best-first search": Node list is a **priority queue**: highest-value gets out first
PROBLEM-SOLVING & SEARCH

• So far it's just been algorithms and data structures

• Where's the AI?

• Many cool AI applications can be built by formalizing a problem as a search problem
  – define the set of possible states (state space)
  – define allowed transitions between states (possibly with associated costs)
  – find a path from the initial state to the goal
PROBLEM-SOLVING & SEARCH

Missionaries and Cannibals

START
High Scores
more games

© Copyright, 2007, Novel Games Limited. All Rights Reserved.
A simple but working sudoku algorithm:

1. start at the top-left corner
2. if cell is given, move to the next cell
3. set value to '0'
4. increment by one
5. if value ≤ 9 and no conflicts detected:
   move to the next cell, and goto 2
6. if value > 9:
   back up to the previous not given cell
7. goto 4
SUDOKU
SUDOKU

1 3
2 1 1
1 2 3
SUDOKU

1 3 3
1 2 1
1 2 3
SUDOKU
SUDOKU
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>4</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>-----</td>
<td>-----</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SUDOKU
SUDOKU

2 3 1 4
1 4 3 2
3 2 1 1
1 2 3
SUDOKU

2 3 1 4
1 4 3 2
3 2 4 1
1 2 3
SUDOKU

IT'S DEPTH-FIRST SEARCH!
BEST-FIRST SEARCH

• We can order the node list any way we like using a priority queue.

• Often different costs are associated with different transitions.

• If objects stored in the priority queue are (node, cost) pairs, where cost is the cost incurred by the path to the node, we can take the cost into account (it is not a function of node).

• Ordering based on the cost:
  – Dijkstra's algorithm: expand the node with minimum accumulated cost.

• Play around with the PathFinding applet.
HEURISTICS

• To direct the search not only towards short paths, but also towards the goal, we can use heuristics.

• A heuristic is an estimate of the *remaining cost* from a node to the goal.

• For example, when finding a route, the distance as the crow flies (Euclidean or geodesic distance) is a lower-bound.
A*:

- A*: best-first search with the priority defined by minimizing the estimated total cost

\[ f(\text{node}, \text{cost}) = \text{cost} + h(\text{node}), \]

where \( h(\text{node}) \) is the heuristic, i.e., an estimate of the remaining cost from the node to the goal

- NB: A node may appear in the priority queue multiple times with different costs (from different paths to the node)
**A**

- Intuition: expand the paths that appear to be optimal first (both low incurred cost, and low estimated remaining cost)

  - Theorem: If the heuristic $h(\text{node})$ never over-estimates the remaining cost, the optimal path is returned

  - No proof given on this course:
    - Intuition is that a *(node, cost)* pair where node = goal and cost is greater than the optimal cost, can never be first in the priority queue

  - For example, Euclidean distance never over-estimates the actual path length (barring worm-holes)
Omat reitit
Ei omia reittejä (ohje omien reittien tallentamiseen).

Omat paikat
Ei omia paikkoja (ohje omien paikkojen tallentamiseen).
Exercise 1.4 "TravelPlanner (2p)"
- state: \((\text{stop, path})\)
- allowed transitions: neighboring stops
- task: find the path with the fewest stops (on the path)
- method: BFS

- Stops and routes available as a json file
  - Java package reads the file and provides a method for listing neighboring stops
  - in python: import json

- Next week: minimize travel time in minutes, A*