Reminders

✦ Homework 3 will be released today
✦ Lyra has help hours Wednesday
✦ My help hours canceled due to conference
✦ Next class is REMOTE (on Zoom)
Q1: What is 1 thing that interested you in the reading?

Q2: What is 1 thing that you found confusing or concerning?
Q1: What is 1 thing that interested you in the reading?

- Image Generation - cool!
- Sandwiches - hidden layers
- Generative Adversarial Networks (GANs)

Q2: What is 1 thing that you found confusing or concerning?

- Data sparsity / class imbalance
Recap
Search Tree

Root node = start state

Expanded nodes

Frontier

Choose leaf node from frontier for expansion according to the search strategy

Determines the search process
Search Strategies

Review: *Strategy* = order of tree expansion
  - Implemented by different queue structures (LIFO, FIFO, priority)

Dimensions for evaluation
  - *Completeness* - always find the solution?
  - *Optimality* - finds a least cost solution (lowest path cost) first?
  - *Time complexity* - # of nodes generated *(worst case)*
  - *Space complexity* - # of nodes simultaneously in memory *(worst case)*

Time/space complexity variables
  - \( b, \) maximum branching factor of search tree
  - \( d, \) depth of the shallowest goal node
  - \( m, \) maximum length of any path in the state space (potentially \( \infty \))
Graph Search vs Tree Search

**function** `TREE-SEARCH(problem)` *returns* a solution, or failure
initialize the frontier using the initial state of `problem`

loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

**function** `GRAPH-SEARCH(problem)` returns a solution, or failure
initialize the frontier using the initial state of `problem`

*initialize the explored set to be empty*
loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    *add node to the explored set*
    expand the chosen node, adding the resulting nodes to the frontier
    only if not in the frontier or explored set
Depth-limited search: A building block

Depth-First search but with depth limit \( l \).
- i.e. nodes at depth \( l \) have no successors.
- No infinite-path problem!

If \( l = d \) (by luck!), then optimal
- But:
  - If \( l < d \) then incomplete 😞
  - If \( l > d \) then not optimal 😞

Time complexity: \( O(b^l) \)
Space complexity: \( O(bl) 😊 \)
## Summary of algorithms

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Depth-First</th>
<th>Depth-limited</th>
<th>Iterative deepening</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td><strong>YES</strong></td>
<td><strong>NO</strong></td>
<td><strong>NO</strong></td>
<td><strong>YES</strong></td>
</tr>
<tr>
<td>Time</td>
<td>(b^d)</td>
<td>(b^m)</td>
<td>(b^l)</td>
<td>(b^d)</td>
</tr>
<tr>
<td>Space</td>
<td>(b^d)</td>
<td>(b^m)</td>
<td>(b^l)</td>
<td>(b^d)</td>
</tr>
<tr>
<td>Optimal?</td>
<td><strong>YES</strong></td>
<td><strong>NO</strong></td>
<td><strong>NO</strong></td>
<td><strong>YES</strong></td>
</tr>
</tbody>
</table>

\(d\) = depth of shallowest goal

\(m\) = maximum depth (depth of longest path)
Informed Search
Informed Search

An informed search strategy uses domain-specific information about the location of the goals in order to find a solution more efficiently than uninformed search.

Hints will come as part of a heuristic function denoted h(n).

One of the most famous informed search algorithms is A* which was developed for robot navigation.

Shakey the robot was developed at the Stanford Research Institute from 1966 to 1972.
Motivation: Map Navigation Problems

All our search methods so far assume \textit{step-cost} = 1

This is only true for some problems
g(N): the path cost function

- Our assumption so far: All moves equal in cost
  - Cost = # of nodes in path-1
  - $g(N) = \text{depth}(N)$ in the search tree

Assign a unique cost to each step:

$$C(i, j) \quad \text{Cost from } N_i \text{ to } N_j$$

$$g(N_3) = C(0, 1) + (1, 2) + (2, 3)$$
Uniform-cost search (UCS)

Extend BFS w/ cost

Frontier: priority queue ordered by g(h)

Test if a node is a goal only when we expand it

Update a node on the frontier if we find a cheaper path to the same state
Shape of Search

- **Breadth First Search** explores equally in all directions. Its frontier is implemented as a FIFO queue. This results in smooth contours or “plys”.

- **Uniform Cost Search** lets us prioritize which paths to explore. Instead of exploring all possible paths equally, it favors lower cost paths. Its frontier is a priority queue. This results in “cost contours”.

Slides adapted from Chris Callison-Burch
A Better Idea...

- Node expansion based on *an estimate* which *includes distance to the goal*
- General approach of informed search:
  - *Best-first search*: node selected for expansion based on an *evaluation function* $f(n)$
    - $f(n)$ includes *estimate* of distance to goal *(new idea!)*
- Implementation: Sort frontier queue by this new $f(n)$.
  - Special cases: *greedy search*, and *A* search

$$g(n) = \text{cost function that tells us cost from start to } n$$

$$f(n) = \text{heuristic that estimates cost from } n \text{ to goal}$$

Slides adapted from Chris Callison-Burch
Simple, useful estimate heuristic: straight-line distances
Greedy Best-First Search
Greedy best-first search: \( f(n) = h(n) \)

Expands the node that \textit{is estimated} to be closest to goal

Completely ignores \( g(n) \): the cost to get to \( n \)

In our Romanian map, \( h(n) = h_{SLD}(n) = \) straight-line distance from \( n \) to Bucharest

In a grid, the heuristic distance can be calculated using the “Manhattan distance”:

```python
def heuristic(a, b):
    # Manhattan distance on a square grid
    return abs(a.x - b.x) + abs(a.y - b.y)
```
Greedy best-first search example

Frontier queue:

- Initial State = Arad
- Goal State = Bucharest
Properties of greedy best-first search

**Optimal?**

- No!
  - Found: *Arad → Sibiu → Fagaras → Bucharest (450km)*
  - Shorter: *Arad → Sibiu → Rimnicu Vilcea → Pitesti → Bucharest (418km)*
A* Search
A* search

Avoid paths that are expensive but prioritize promising paths.

\[
f(n) = g(n) + h(n)
\]

- \( g(n) \): actual cost from start to node
- \( h(n) \): estimated cost from node to goal
- \( f(n) \): estimated total cost of path from start to goal
Idea: Admissibility

Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the frontier

Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs
UCS vs A* Contours

Uniform-cost expands equally in all “directions”

A* expands mainly toward the goal, but does hedge its bets to ensure optimality
A* Applications

Pathing / routing problems (A* is in your GPS!)
Video games
Robot motion planning
Resource planning problems

...
Heuristics
Heuristic Functions

For the 8-puzzle

- Avg. solution cost is about 22 steps
  - (branching factor ≤ 3)
  - (branching factor ≤ 3)
- A good heuristic function can reduce the search process
Admissible Heuristics

For the 8-puzzle:

\[ h_{oop}(n) = \text{number of out of place tiles} \]

\[ h_{md}(n) = \text{total Manhattan distance (i.e., \# of moves from desired location of each tile)} \]

\[ h_{oop}(S) = 8 \]

\[ h_{md}(S) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18 \]
Key: Admissibility

Inadmissible (pessimistic) heuristics break optimality by pushing good plans too far back on the frontier, which means they may never get expanded.

Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs. That means that the true best plan will always be expanded.