CMSC 471
Fall 2012

Class #4

Tue 9/11/11
Uninformed Search

Kevin Winner, winnerk1@umbc.edu

Today's Class

- Specific algorithms
  - Breadth-first search
  - Depth-first search
  - Uniform cost search
  - Iterative deepening search
- Example problems revisited
- Lisp search demo

Uninformed Search

Section 3.4

Some material adopted from notes by Charles R. Dyer, University of Wisconsin-Madison

Key Procedures to be Defined

- EXPAND
  - Generate all successor nodes of a given node
- GOAL-TEST
  - Test if state satisfies all goal conditions
- QUEUEING-FUNCTION
  - Used to maintain a ranked list of nodes that are candidates for expansion

Evaluating Search Strategies

- **Completeness**
  - Guarantees finding a solution whenever one exists
- **Time complexity**
  - How long (worst or average case) does it take to find a solution? Usually measured in terms of the number of nodes expanded
- **Space complexity**
  - How much space is used by the algorithm? Usually measured in terms of the maximum size of the "nodes" list during the search
- **Optimality/Admissibility**
  - If a solution is found, is it guaranteed to be an optimal one? That is, is it the one with minimum cost?

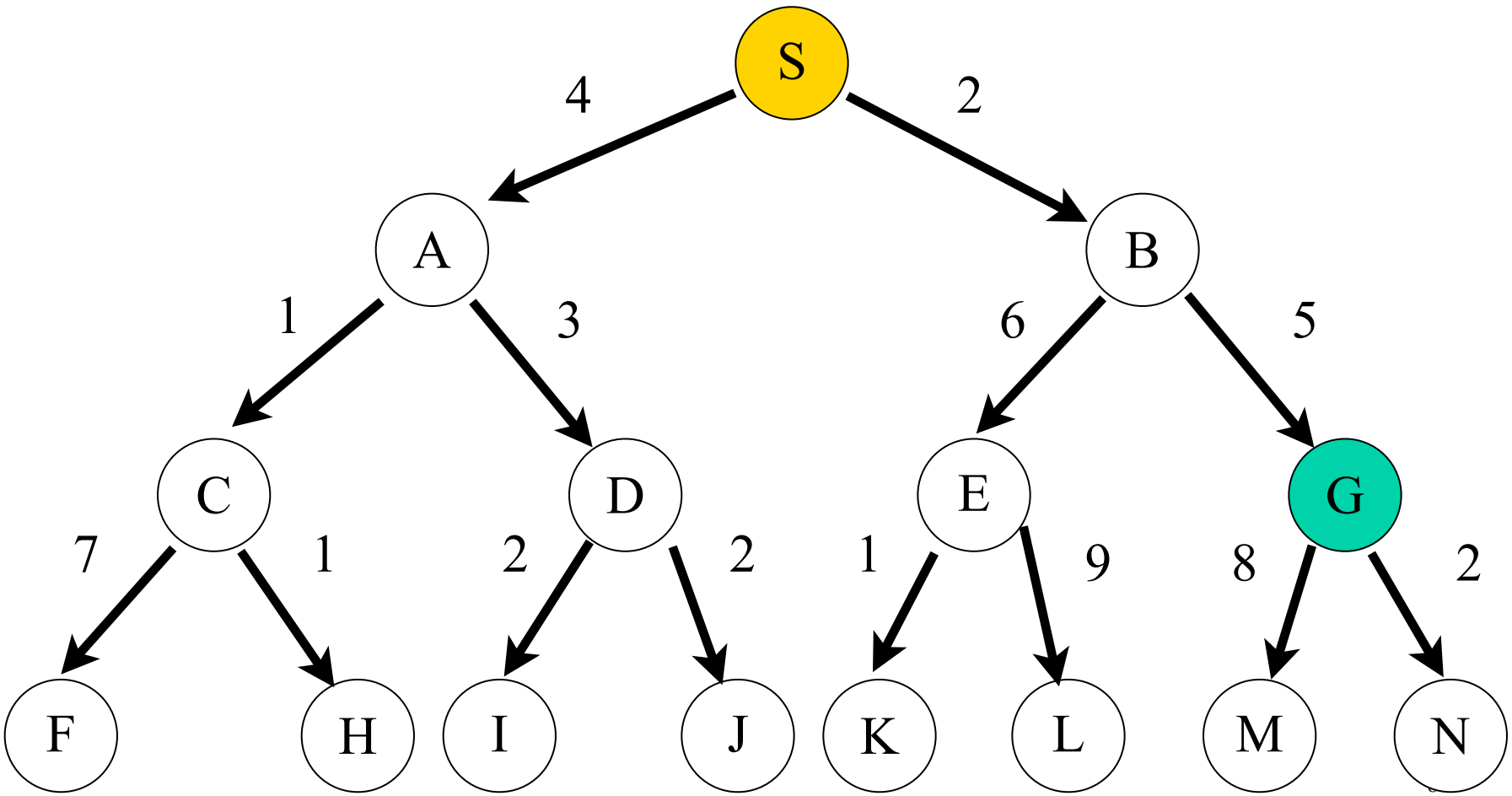
Other stuff

- Search process constructs a search tree, where
  - **root** is the initial state and
  - **leaf nodes** are nodes having no successors
  - the **frontier** of the graph are the nodes which have not been expanded yet but which are in our queue
- Uninformed search strategies may not be **complete**
  - Infinite state spaces
  - Cycles in the graph

Uninformed vs. Informed Search

- **Uninformed search strategies**
  - Also known as "blind search," uninformed search strategies use no information about the likely "direction" of the goal node(s)
  - Uninformed search methods: Breadth-first, depth-first, depth-limited, depth-first iterative deepening, uniform-cost, bidirectional*
- **Informed search strategies (next class...)**
  - Also known as "heuristic search," informed search strategies use information about the domain to (try to) (usually) head in the general direction of the goal node(s)
  - Informed search methods: Hill climbing, best-first, greedy search, beam search, A, A*
- **Uninformed search is not useless!**
  - Heuristics are not always available

Example Problem Graph



Uninformed Search Methods

Breadth-First Search (BFS)

- Enqueue nodes in **FIFO** (first-in, first-out) order.
- **Complete**
- **Optimal** (i.e., admissible) if all operators have the same cost. Otherwise, not optimal but finds solution with shortest path length.
- **Exponential time and space complexity**, $O(b^d)$, where d is the depth of the solution and b is the branching factor (i.e., number of children) at each node
- Will take a **long time to find solutions** with a large number of steps because BFS must look at all shorter length possibilities first
  - A complete search tree of depth d where each non-leaf node has b children, has a total of $1 + b + b^2 + ... + b^d = (b^{(d+1)} - 1)/(b-1)$ nodes
  - For a complete search tree of depth 12, where every node at depths 0, ..., 11 has 10 children and every node at depth 12 has 0 children, there are $1 + 10 + 100 + 1000 + ... + 10^{12} = (10^{13} - 1)/9 = O(10^{12})$ nodes in the complete search tree. If BFS expands 1000 nodes/sec and each node uses 100 bytes of storage, then BFS will take 35 years to run in the worst case, and it will use 111 terabytes of memory!

Depth-First Search (DFS)

- Enqueue nodes on nodes in **LIFO** (last-in, first-out) order. That is, nodes used as a stack data structure to order nodes.
- **Not complete** (gets stuck in cycles and cannot handle infinite state spaces)
- **Not optimal** (always returns the first solution found)
- **Exponential time**, $O(b^d)$, but only **linear space**, $O(bd)$
- Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)
- When search hits a deadend, can only back up one level at a time even if the "problem" occurs because of a bad action choice near the top of the tree.

Depth-First Search Variants

- **Goal test at generation**
- **Backtracking search**
  - Only generate one successor at a time while expanding
  - Add more bookkeeeping data in order to resume expansion later
    - Effectively does away with the nodes list
    - Space complexity becomes *O(d)*
- **Depth-limited search**
  - Cutoff expansion of states beyond some depth *d*
  - Handles infinite state spaces (and cycles)
  - No longer complete if the depth of the shallowest goal is > than *d*
    - Means we must choose *d* very carefully (typically with domain knowledge)

Iterative Deepening Search (IDS)

- First do DFS to depth 0 (i.e., treat start node as having no successors), then, if no solution found, do DFS to depth 1, etc.

  until solution found do

  DFS with depth cutoff c

  c = c+1

- **Complete**
- **Optimal/Admissible** if all operators have the same cost. Otherwise, not optimal but guarantees finding solution of shortest length (like BFS).
- Time complexity is a little worse than BFS or DFS because nodes near the top of the search tree are generated multiple times, but because almost all of the nodes are near the bottom of a tree, the worst case time complexity is still exponential, $O(b^d)$

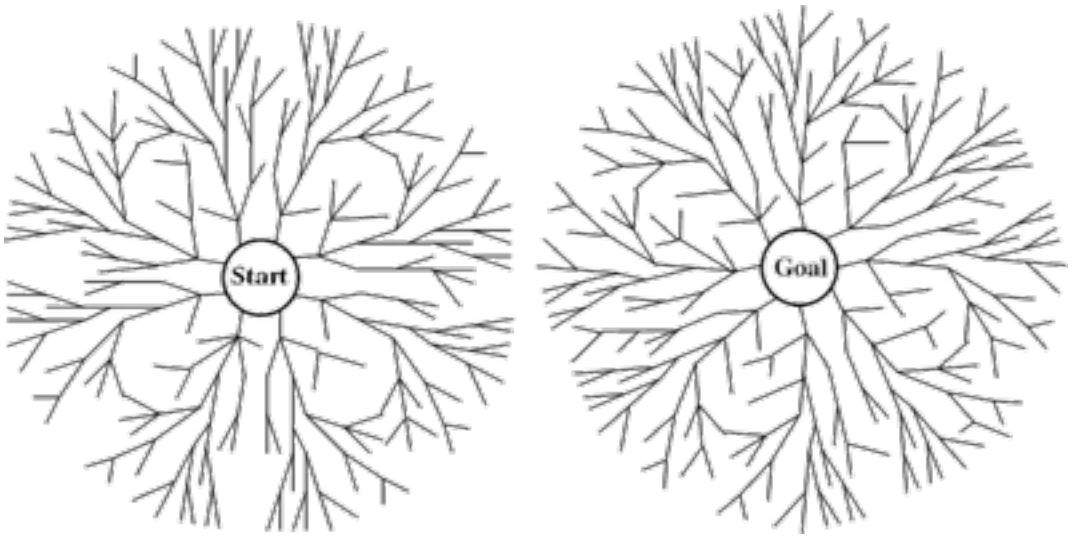
Iterative Deepening Search

- If branching factor is b and solution is at depth d, then nodes at depth d are generated once, nodes at depth d-1 are generated twice, etc.
  - Hence $N(\text{IDS}) = (d)\text{b} + (d\text{-}1)b^2 + \ldots + (1)b^d = O(b^d)$.
  - If b=10 and d=5, then:
    - $N(\text{IDS}) = 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}450$
    - $N(\text{BFS}) = 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 = 111{,}110$
- Advantages from DFS
  - **Linear space complexity**, O(bd)
  - Can find long solutions quickly
- Advantages from BFS
  - **Optimal** and **complete**
- Generally preferred for **large state spaces** where **solution depth is unknown**

Uniform-Cost (UCS)

- Enqueue nodes by **path cost**. That is, let g(n) = cost of the path from the start node to the current node n. Sort nodes by increasing value of g.
  - Identical to breadth-first search if all operators have equal cost
- Called *"Dijkstra's Algorithm"* in the algorithms literature and similar to *"Branch and Bound Algorithm"* in operations research literature
- **Complete (*)**
  - Complete iff there are no zero-cost cycles
- **Optimal/Admissible** (*)
  - Admissibility depends on the goal test being applied *when a node is removed from the nodes list*, not when its parent node is expanded and the node is first generated
- **Exponential time and space complexity**, $O(b^d)$

Bi-directional Search



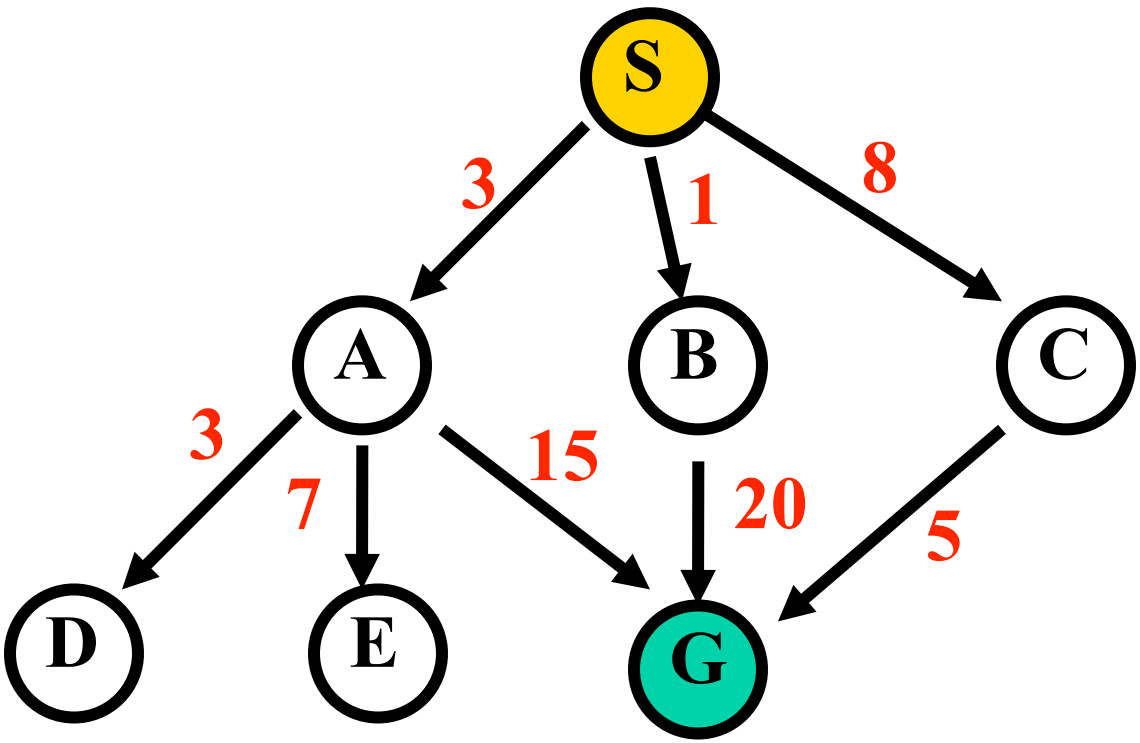
- Alternate searching from the start state toward the goal and from the goal state toward the start.
- Stop when the frontiers intersect.
- Works well only when there are unique start and goal states.
- Requires the ability to generate “predecessor” states.
- Can (sometimes) lead to finding a solution more quickly.

Avoiding Repeated States

- In increasing order of effectiveness in reducing size of state space and with increasing computational costs:
  1. Do not return to the state you just came from.
  2. Do not create paths with cycles in them.
  3. Do not generate any state that was ever created before.
- Net effect depends on frequency of "loops" in state space.

Uninformed Search Examples

Example for Illustrating Search Strategies



Depth-First Search

Expanded node	Nodes list
	{ S^0 }
S^0	{ A^3 B^1 C^8 }
A^3	{ D^6 E^{10} G^{18} B^1 C^8 }
D^6	{ E^{10} G^{18} B^1 C^8 }
E^{10}	{ G^{18} B^1 C^8 }
G^{18}	{ B^1 C^8 }

Solution path found is S A G, cost 18

Number of nodes expanded (including goal node) = 5

Breadth-First Search

Breadth-First Search

Expanded node	Nodes list
	{ S^0 }
S^0	{ A^3 B^1 C^8 }
A^3	{ B^1 C^8 D^6 E^{10} G^{18} }
B^1	{ C^8 D^6 E^{10} G^{18} G^{21} }
C^8	{ D^6 E^{10} G^{18} G^{21} G^{13} }
D^6	{ E^{10} G^{18} G^{21} G^{13} }
E^{10}	{ G^{18} G^{21} G^{13} }
G^{18}	{ G^{21} G^{13} }

Solution path found is S A G , cost 18

Number of nodes expanded (including goal node) = 7

Uniform-Cost Search

Uniform-Cost Search

Expanded node	Nodes list
	{ S^0 }
S^0	{ B^1 A^3 C^8 }
B^1	{ A^3 C^8 G^{21} }
A^3	{ D^6 C^8 E^{10} G^{18} G^{21} }
D^6	{ C^8 E^{10} G^{18} G^1 }
C^8	{ E^{10} G^{13} G^{18} G^{21} }
E^{10}	{ G^{13} G^{18} G^{21} }
G^{13}	{ G^{18} G^{21} }

Solution path found is S B G, cost 13

Number of nodes expanded (including goal node) = 7

Iterative Deepening Search

Iterative Deepening Search

d	Expanded node	Nodes list
		{ S^0 }
0	S^0	{}
		{ S^0 }
1	S^0	{A^3 B^1 C^8}
	A^3	{}
	B^1	{}
	C^8	{}
		{ S^0 }
2	S^0	{A^3 B^1 C^8}
	A^3	{ D^6 E^{10} G^{18} B^1 C^8 }
	D^6	{ E^{10} G^{18} B^1 C^8 }
	E^{10}	{ G^{18} B^1 C^8 }
	G^{18}	{ B^1 C^8 }

Solution path found is S A G , cost 18

Number of nodes expanded (including goal node) = 10

8-Puzzle Revisited

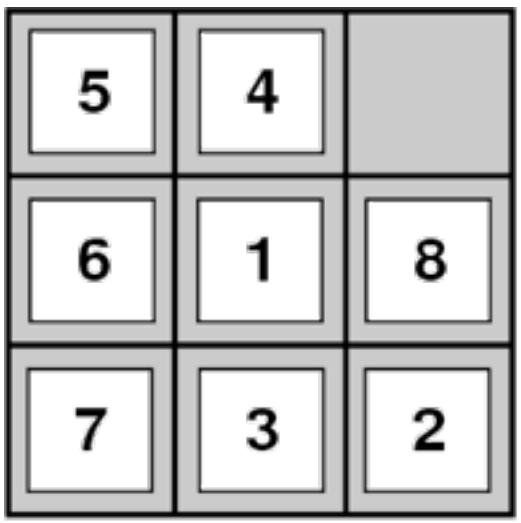
Depth-first search?

Breadth-first search?

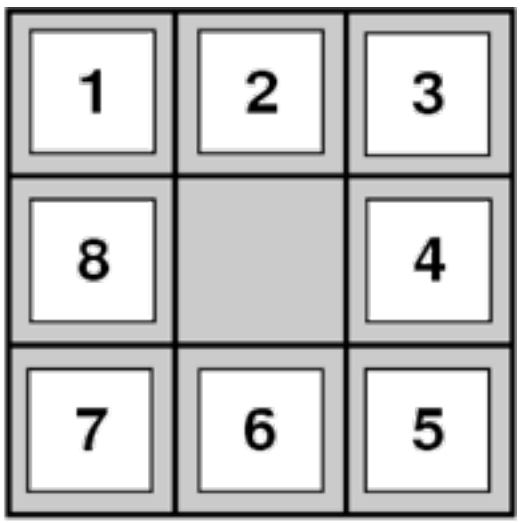
Uniform-cost?

Iterative deepening?

Vote!



Start State



Goal State

8-Puzzle Revisited

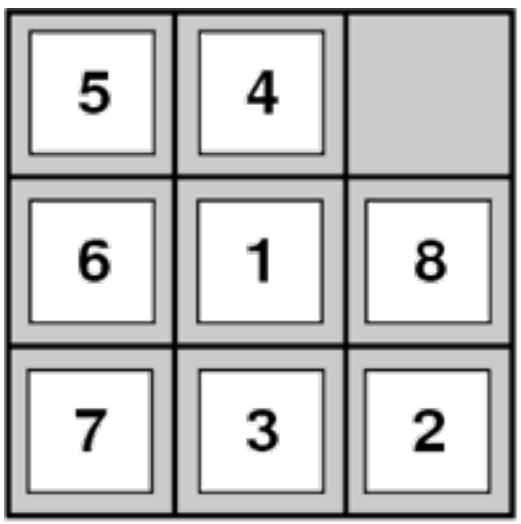
Depth-first search?

Breadth-first search?

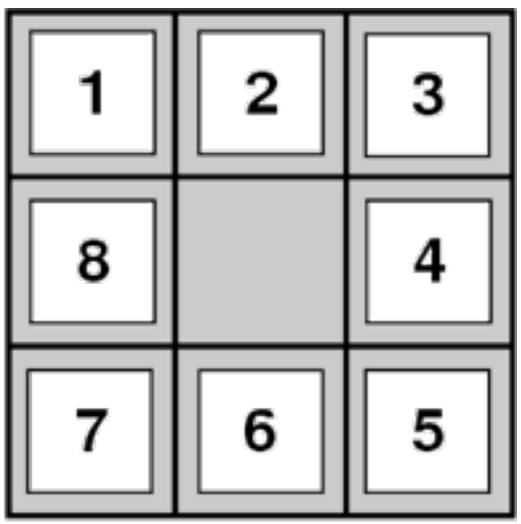
Uniform-cost?

Iterative deepening?

Vote!



Start State



Goal State

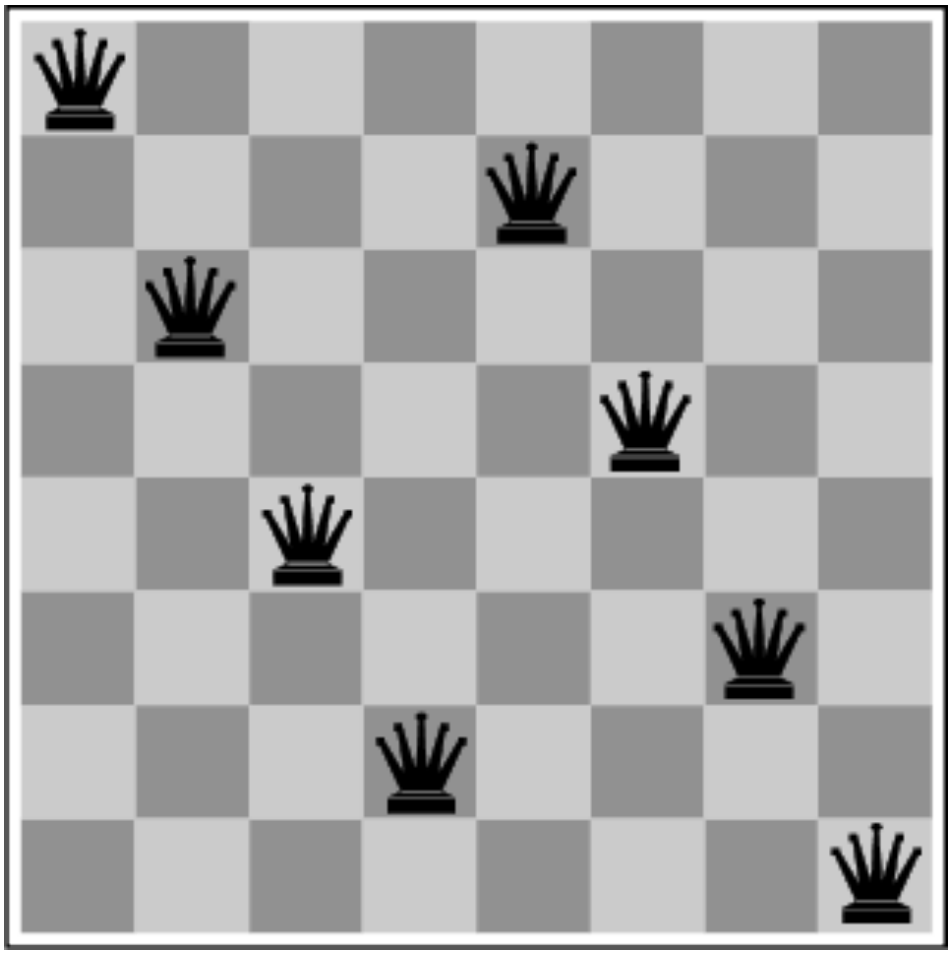
The 8-Queens Problem Revisited

Depth-first search?

Breadth-first search?

Uniform-cost?

Iterative deepening?



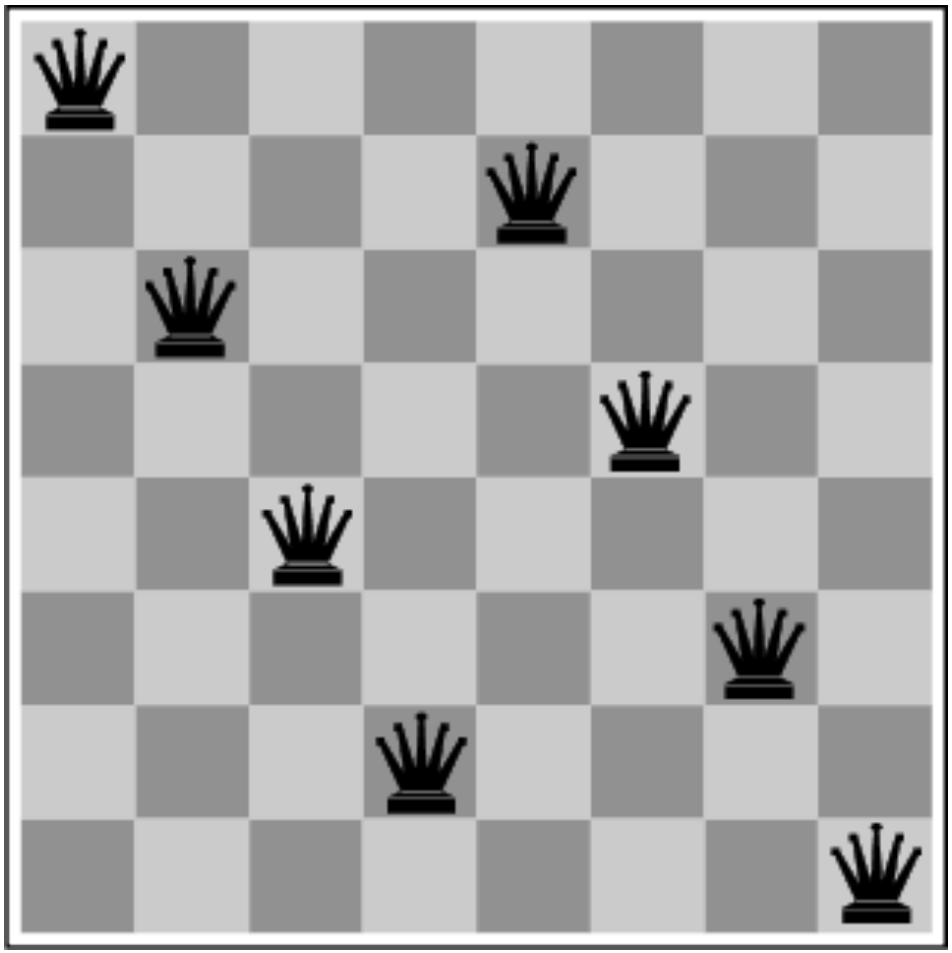
The 8-Queens Problem Revisited

Depth-first search?

Breadth-first search?

Uniform-cost?

Iterative deepening?



Missionaries and Cannibals

Depth-first search?

Breadth-first search?

Uniform-cost?

Iterative deepening?

Missionaries and Cannibals

Depth-first search?

Breadth-first search?

Uniform-cost?

Iterative deepening?

Missionaries and Cannibals

Depth-first search?

Breadth-first search?

Uniform-cost?

Iterative deepening?

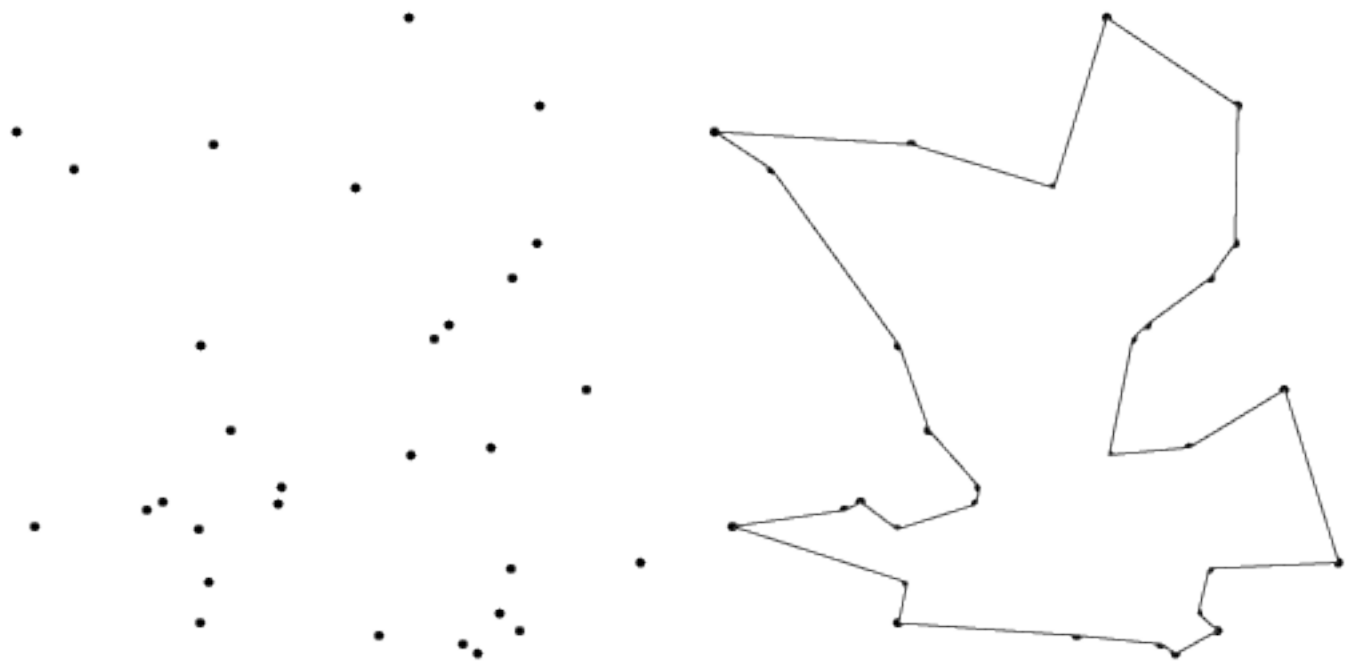
Traveling Salesman

Depth-first search?

Breadth-first search?

Uniform-cost?

Iterative deepening?



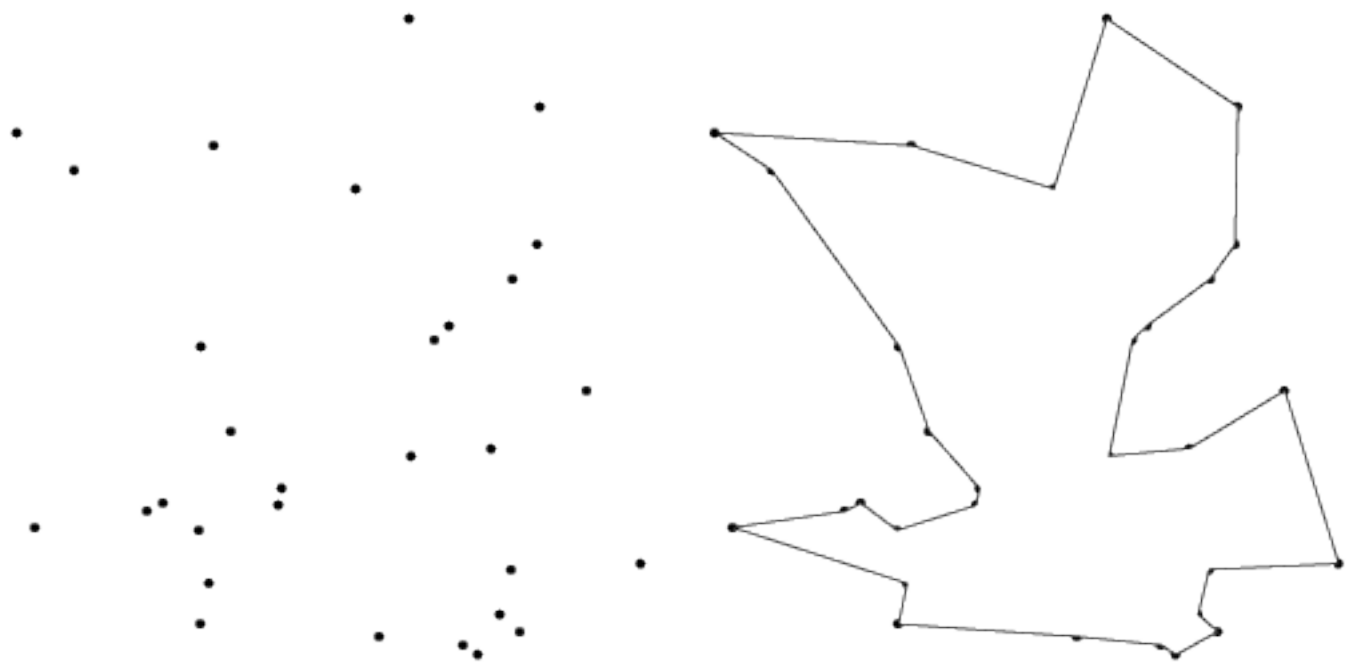
Traveling Salesman

Depth-first search?

Breadth-first search?

Uniform-cost?

Iterative deepening?



That's all!

- Lisp Search demo (time permitting)
- Next class
  - Informed Search
  - **Homework #1 is due!!**