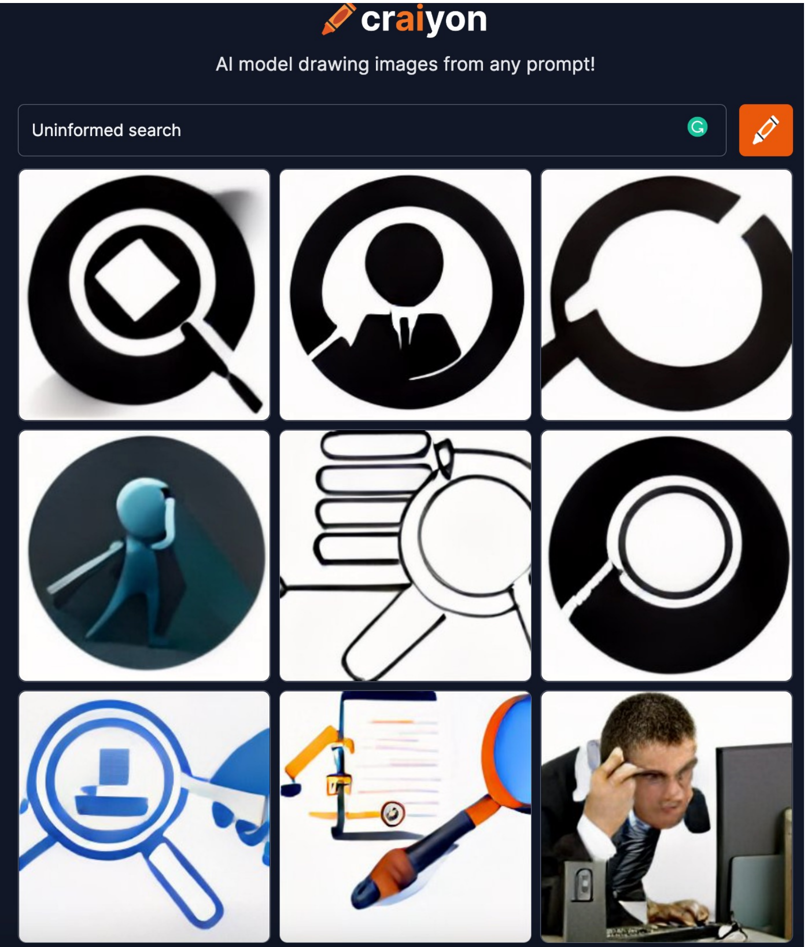
Search algorithms and uninformed search



CS B551
Fall 2022

Announcements

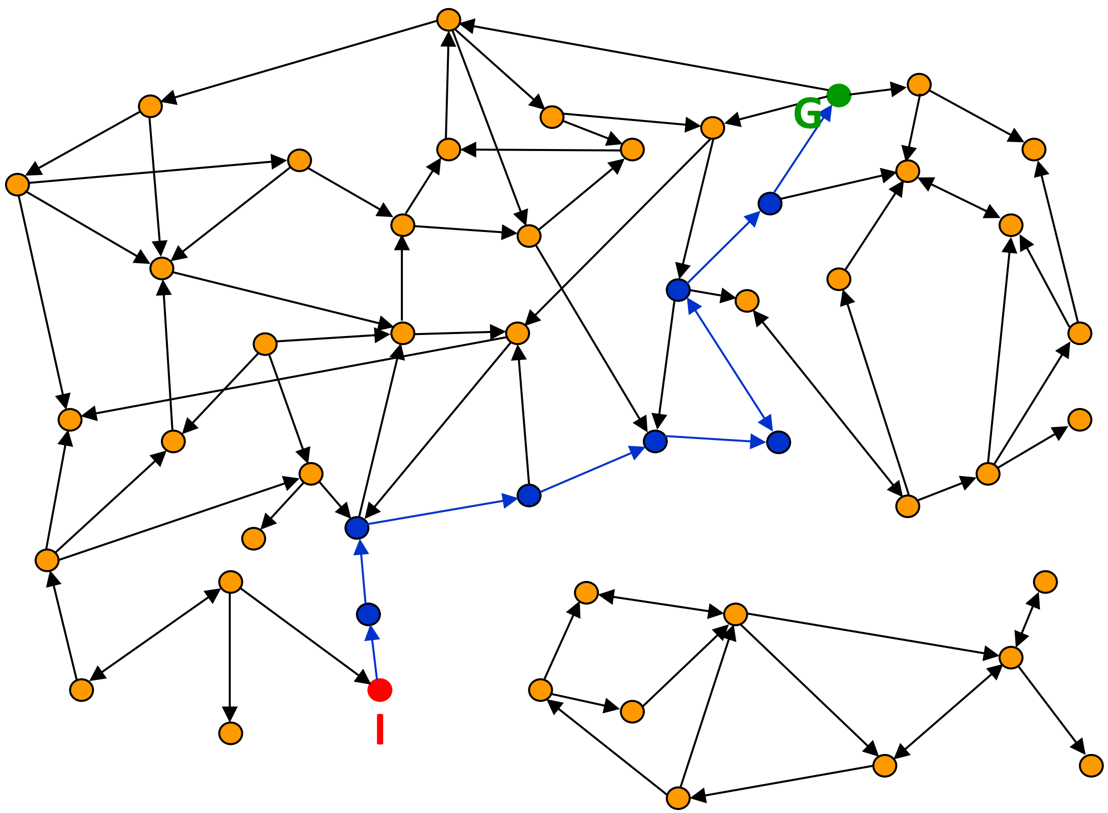
- AI's & Office hours
- Canvas, Q&A Community, Slack, etc. for Syllabus, videos, questions, slides, discussions, etc.
- Assignment 0 coming soon! (Wednesday)
  - Practice with searching, and with Python.
  - Lots of online resources to learn Python: Google Code, CodeAcademy, many, many tutorials, etc.

These abstractions have 5 parts:

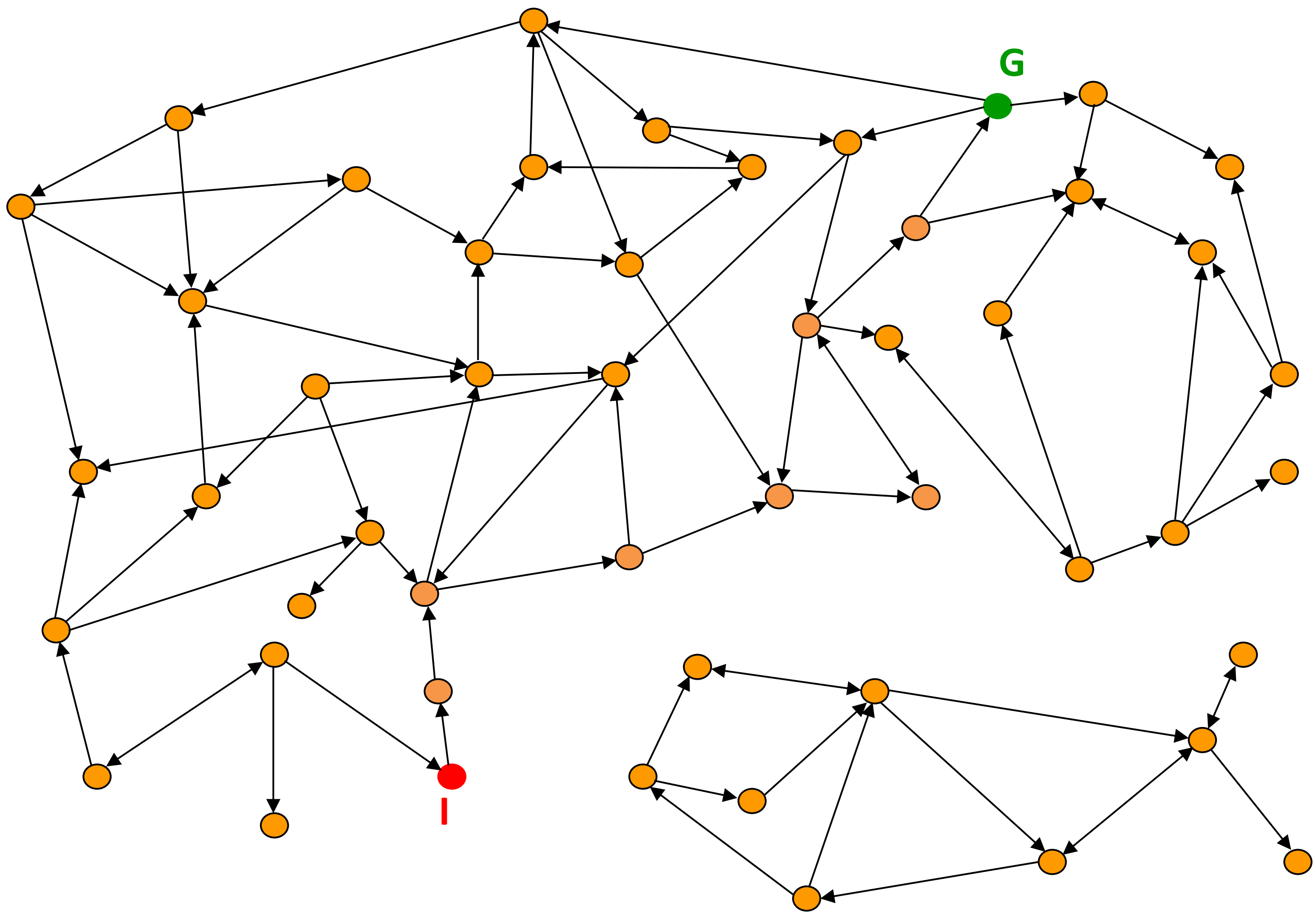
1. Set of states S
2. Initial state s_0
3. A function SUCC: $S \rightarrow 2^S$ that encodes possible transitions of the system
4. Set of goal states
5. A cost function that calculates how "expensive" a given set of moves is

Recall: Abstracting AI problems with graphs

- Nodes are states
- Start state, goal state(s)
- Edges encode SUCC
- Cost function
- A solution is a path from initial node to a goal
- The cost of a path is the sum of its edge costs
- An optimal solution is a path of minimum cost



Graph search



8-, 15-, 24-Puzzles

8-puzzle → 362,880 states

0.036 sec

15-puzzle → 2.09 x 10^{13} states

~ 55 hours

24-puzzle → 10^{25} states

> 10^9 years

100 million states/sec

Tractability of search hinges on the ability to explore only a tiny portion of the state graph!

Intractability

- Constructing the full state graph is intractable for most interesting problems
- n-puzzle: $(n+1)!$ states

Example of a search tree

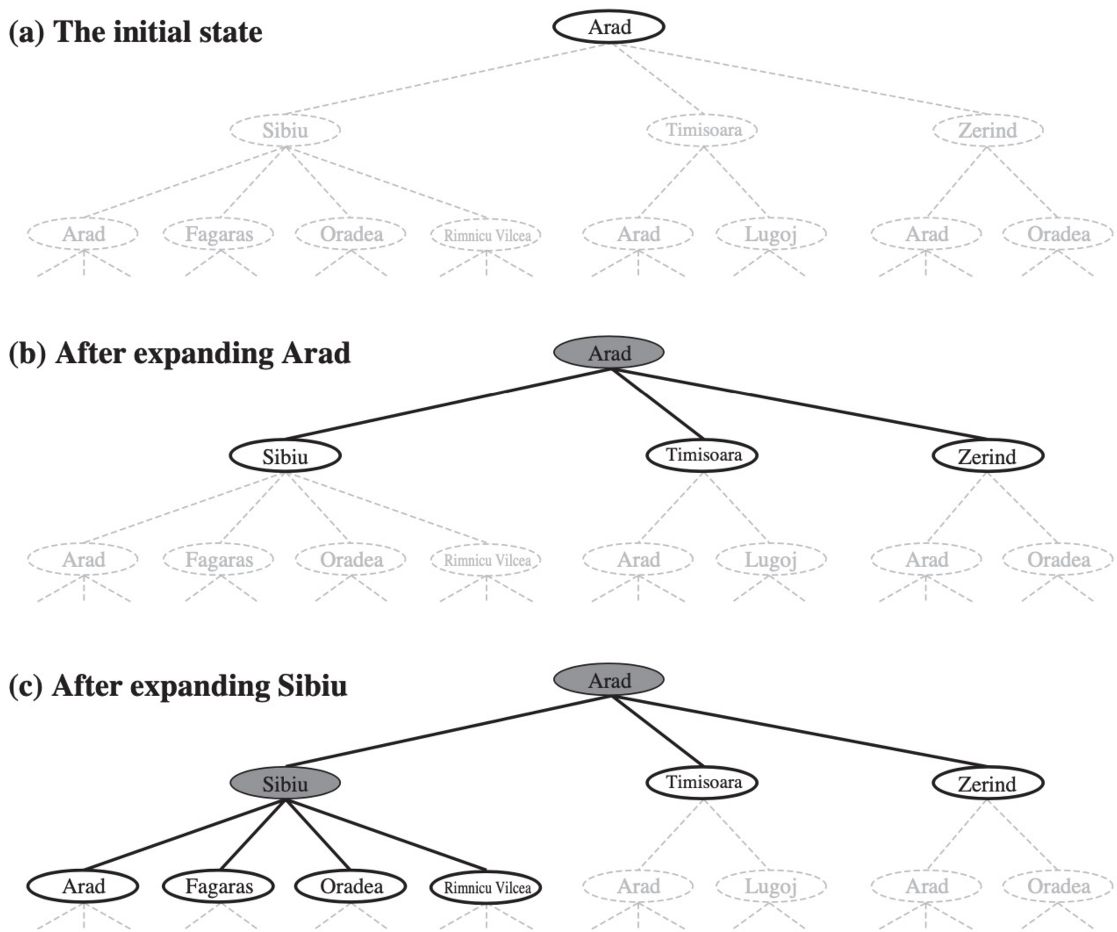


Figure 3.6 Partial search trees for finding a route from Arad to Bucharest. Nodes that have been expanded are shaded; nodes that have been generated but not yet expanded are outlined in bold; nodes that have not yet been generated are shown in faint dashed lines.

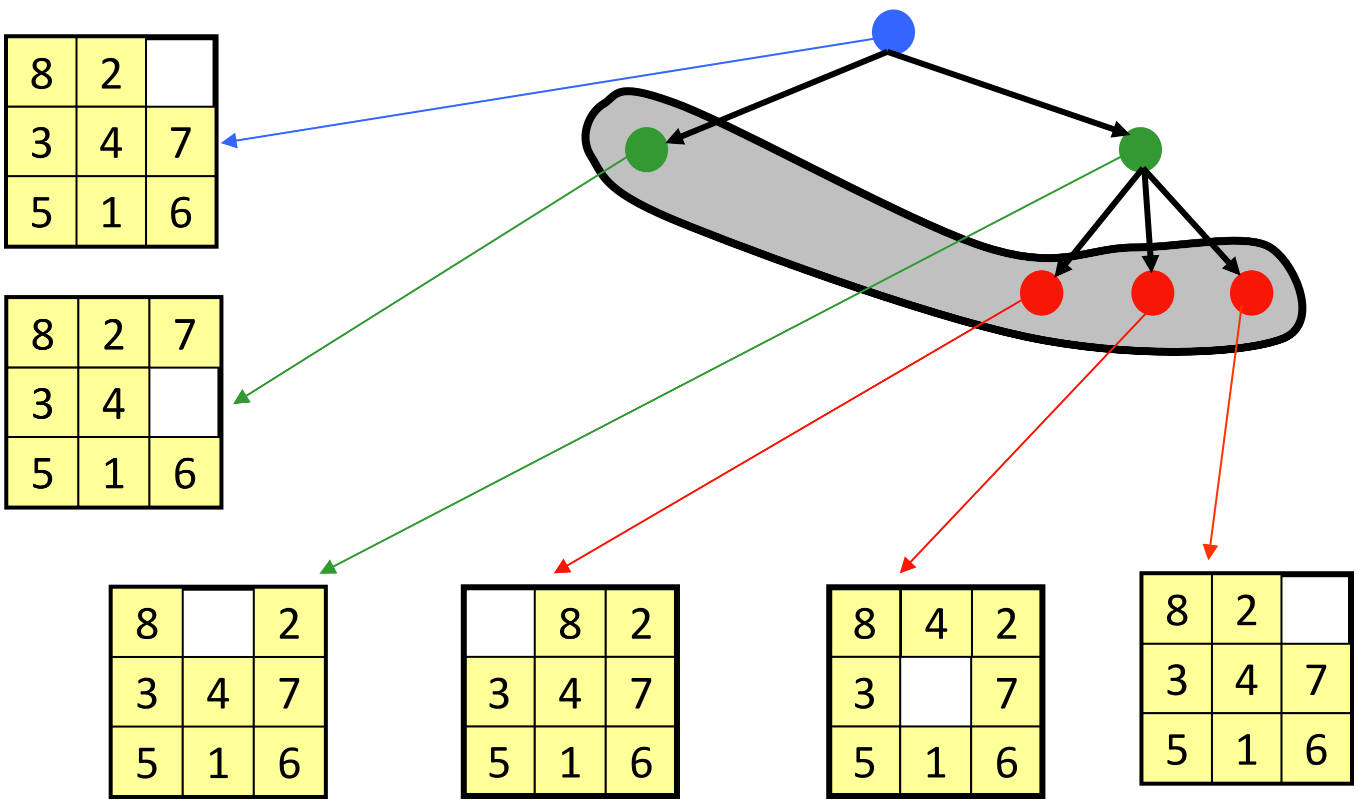
The root node of the tree corresponds to the initial state, *In(Arad)*.

We **expand** the current state; that is, applying each legal action to the current state, thereby **generating** a new set of states. In this case, we add three branches from the **parent node** *In(Arad)* leading to three new **child nodes**: *In(Sibiu)*, *In(Timisoara)*, and *In(Zerind)*.

What to do now?

The set of all leaf nodes available for expansion at any given point is called the **frontier**.

Fringe or Frontier



Informal description of a search algorithm

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 the 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

Figure 3.7 An informal description of the general tree-search and graph-search algorithms. The parts of GRAPH-SEARCH marked in bold italic are the additions needed to handle repeated states.

Evaluation criteria for search algorithms

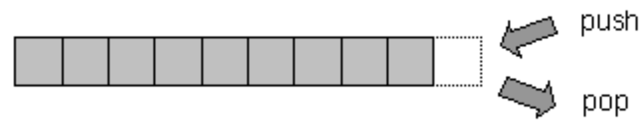
- We will consider different algorithms and evaluate them in terms of:
  - **Completeness**: Is the algorithm guaranteed to find a solution when there is one?
  - **Optimality**: Does the strategy find the optimal solution (minimum cost path)?
  - **Time complexity**: How long does it take to find a solution?
  - **Space complexity**: How much memory is needed to perform the search?
    - size of the state space graph, V + E, where V is the set of vertices (nodes) of the graph and E is the set of edges (links).

Note about complexity

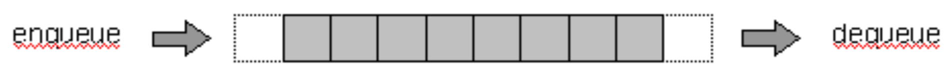
- Complexity is usually expressed in terms of three quantities:
  - b, the **branching factor** or maximum number of successors of any node;
  - d, the **depth** of the shallowest goal node (i.e., the number of steps along the path from the root);
  - m, the maximum length of any path in the state space.
- Time is often measured in terms of the number of nodes generated during the search, and space in terms of the maximum number of nodes stored in memory.

Stacks, Queues, PQs, Oh my!

- Stack



- Queue



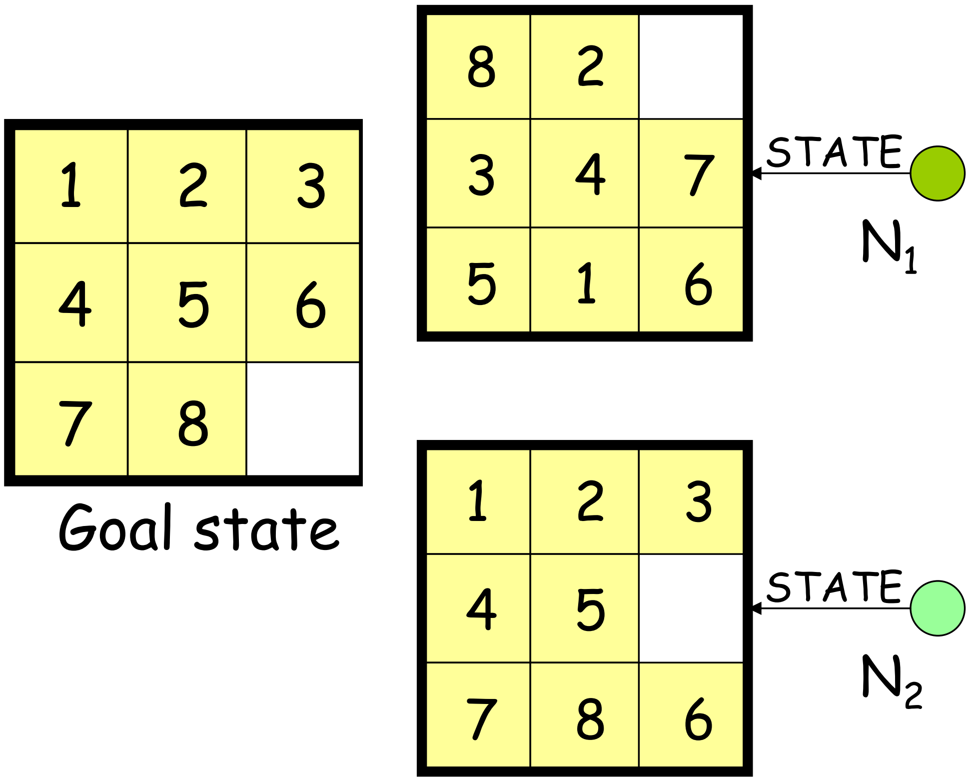
- Priority Queue
  - You put (item, priority) pairs into queue
  - You remove the highest-priority item from the queue

Blind vs. Heuristic Strategies

- Blind (or un-informed) strategies do not use properties of states to order FRINGE. All states are treated the same.

- Heuristic (or informed) strategies order FRINGE so that more “promising” states are considered first

Example

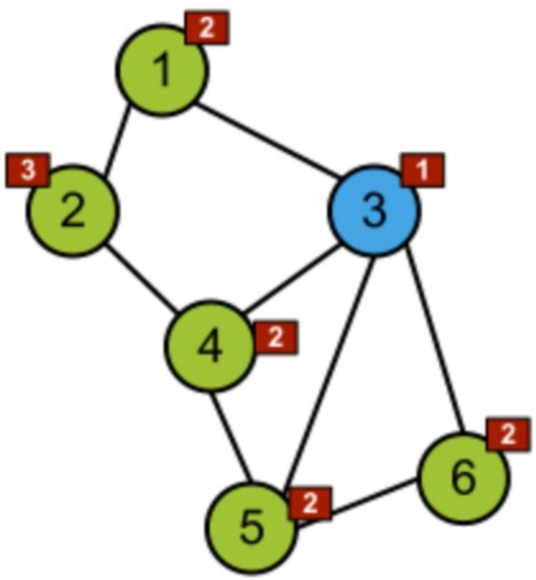


For a blind strategy, N_1 and N_2 are just two nodes (at some position in the search tree)

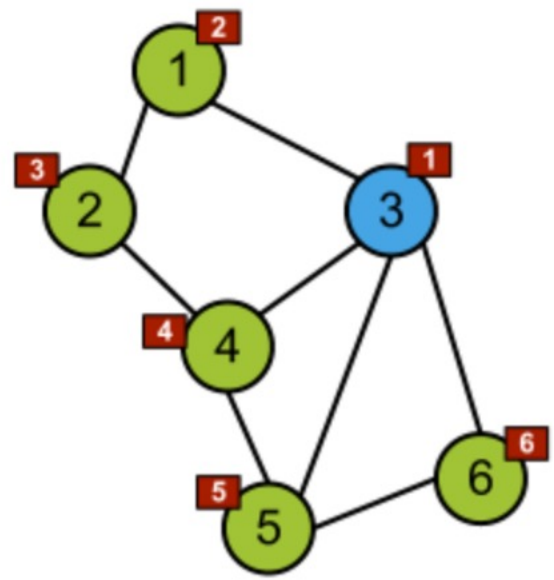
For a heuristic strategy, N_2 seems more promising than N_1 (fewer misplaced tiles)

Breadth-first search and Depth-first search

- We will cover two uninformed (blind) search algorithms:



Breadth-first search



Depth-first search

Breadth-first search

Breadth-first search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then *their* successors, and so on. In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.

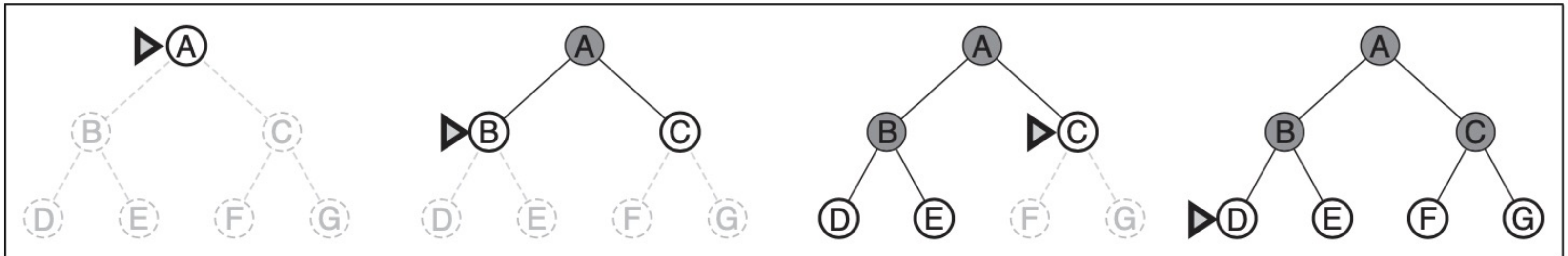


Figure 3.12 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker.

https://opendsa-server.cs.vt.edu/OpenDSA/Books/CS3/html/GraphTraversal.html#breadth-first-search

Time and Memory Requirements

d	# Nodes	Time	Memory
2	111	.01 msec	11 Kbytes
4	11,111	1 msec	1 Mbyte
6	$\sim 10^{6}$	1 sec	100 Mb
8	$\sim 10^{8}$	100 sec	10 Gbytes
10	$\sim 10^{10}$	2.8 hours	1 Tbyte
12	$\sim 10^{12}$	11.6 days	100 Tbytes
14	$\sim 10^{14}$	3.2 years	10,000 Tbytes

Assumptions: b = 10; 1,000,000 nodes/sec; 100bytes/node

Depth-first search

Depth-first search always expands the *deepest* node in the current frontier of the search tree.

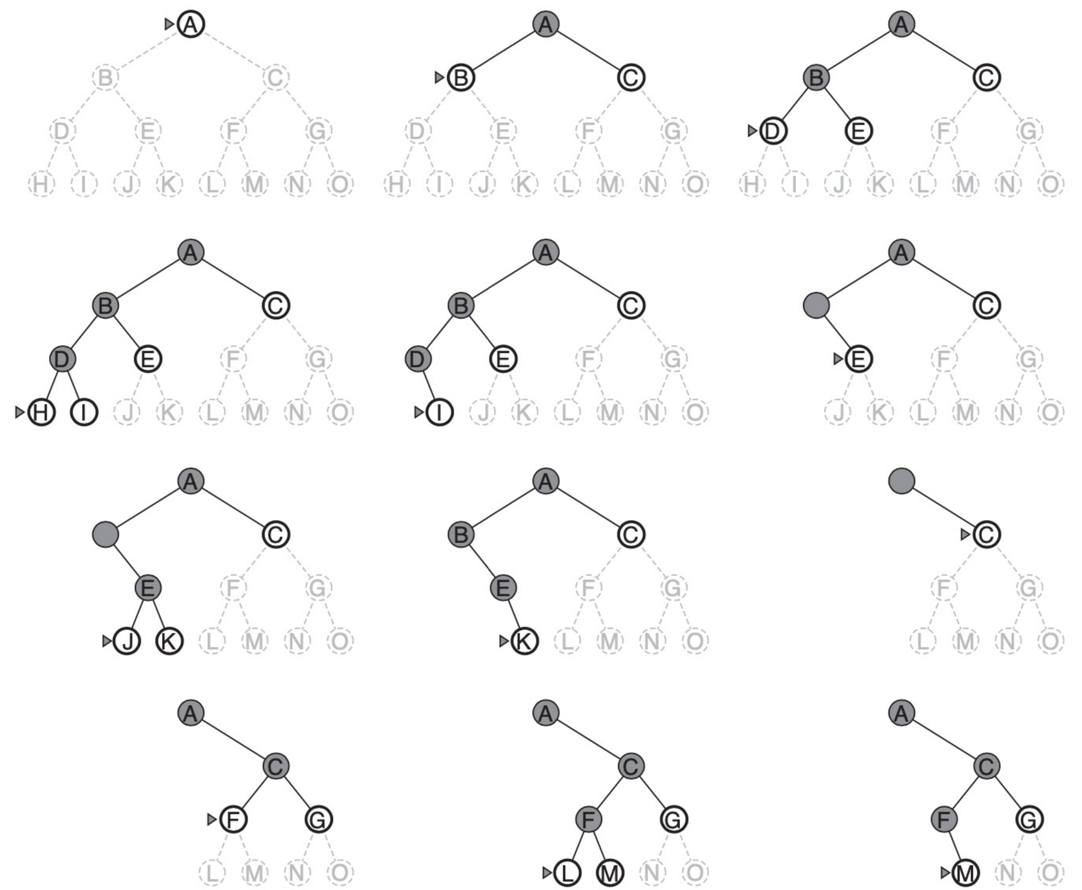


Figure 3.16 Depth-first search on a binary tree. The unexplored region is shown in light gray. Explored nodes with no descendants in the frontier are removed from memory. Nodes at depth 3 have no successors and M is the only goal node.

https://opendsa-server.cs.vt.edu/OpenDSA/Books/CS3/html/GraphTraversal.html#depth-first-search

Comparing uninformed search strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^a	Yes^{a,b}	No	No	Yes^a	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^c	Yes	No	No	Yes^c	Yes^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.

Next class

- Heuristic (or informed) search