ANNOUNCEMENTS

- Homework Programming Assignment Out Today
- Start thinking about Projects
  - 3 person teams (2 acceptable)
  - Projects should exploit course techniques
  - I am happy to help with formulating Ideas
- In class assignment today
  - On paper

UNINFORMED SEARCH

LECTURE 2

CS5100

DRAWING ON MATERIAL AND IMAGES FROM RUSSELL & NORVIG,

AND MANY OTHER SOURCES

(THE WEB, ROB PLATT, MAGY SEIF EL-NASR, BERKELEY CS188)

SEARCH/PLANNING AGENTS

Agents that plan:

- Decisions based on (hypothesized) consequences of actions
  - How the world would be after some action(s)
- Model-based
  - Use a model of how the world evolves in response to actions

- What are the implicit assumptions here??

SEARCH/PLANNING AGENTS

Agents that plan:

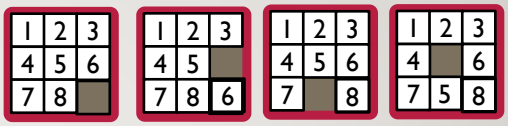
- Ask "what if"
- Decisions based on (hypothesized) consequences of actions
  - How the world would be after some action(s)
- Model-based
  - Use a model of how the world evolves in response to actions

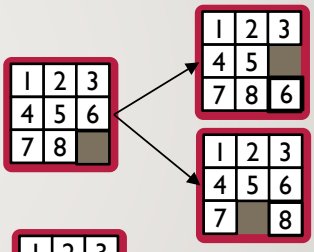
For now , we will look at search/planning agents that plan a solution to the goal.
Later on we will relax that as we look at ways agents that agents can use planning more flexibly, in ways more responsive to uncertainties

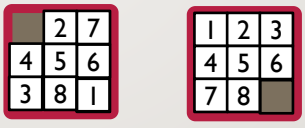
USING SEARCH TO PLAN OUT A SOLUTION

SEARCH PROBLEMS

- A search problem consists of:
  - A state space
  - A transition function
    (with actions, possibly costs)
    AKA successor, transition model
  - A start state and a goal test

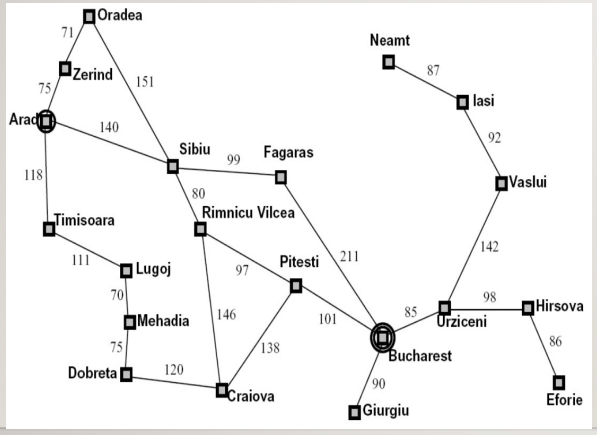






- A solution is a sequence of actions (a plan) which transforms the start state to a goal state

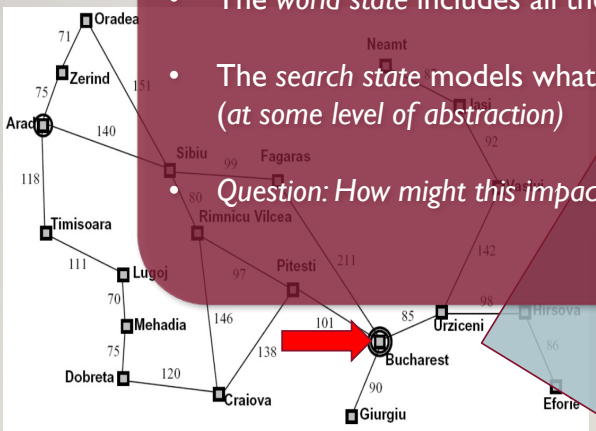
EXAMPLE: PLANNING A ROUTE IN ROMANIA



- State space:
  - Cities
- Transition function:
  - Roads: Go to adjacent city with cost = distance
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?
- Solution?

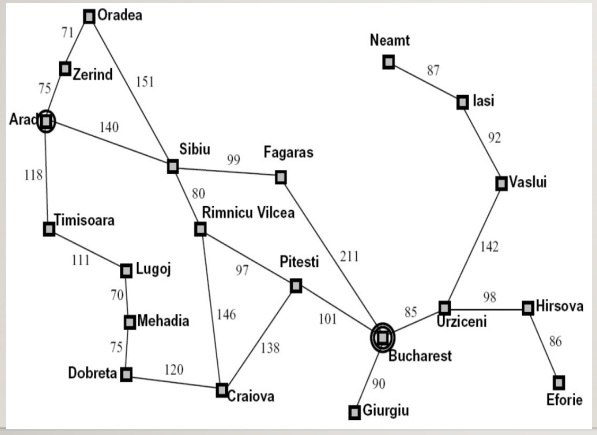
SEARCH PROBLEMS FORMULATE (ABSTRACT) MODELS

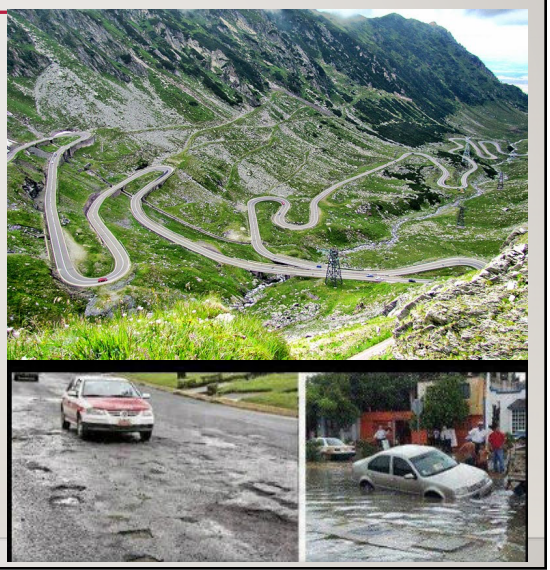
- Example: Bucharest, Romania is a single state (node) in the state space
- Abstracts away detail
- The *world state* includes all the details about the world
- The *search state* models what is necessary to search for a solution (*at some level of abstraction*)
- *Question: How might this impact the execution of the solution?*



SEARCH PROBLEMS FORMULATE MODELS

- What detail to abstract?
- And when to abstract it?



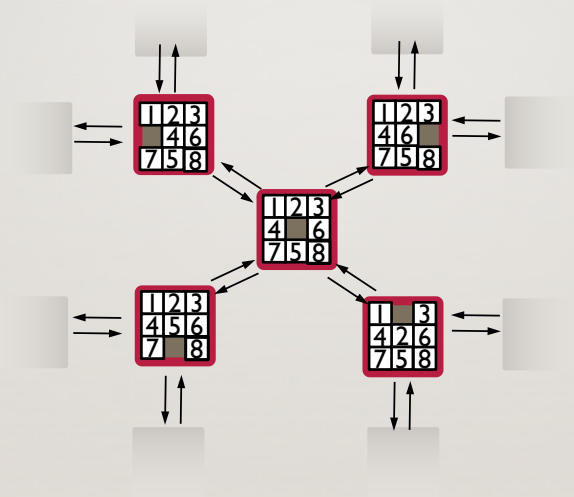


STATE SPACE GRAPHS AND *SEARCH TREES*

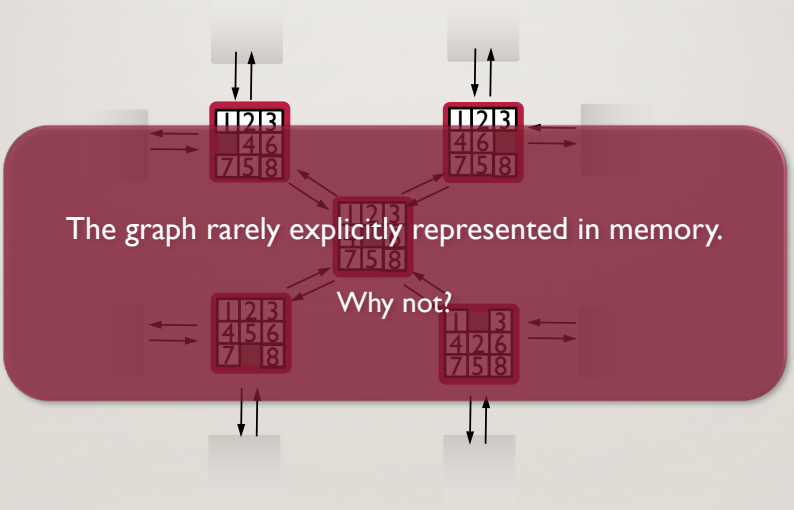
STATE SPACE GRAPH

- State space graph is a representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs/edges represent transitions (action results)
  - The goal test is a set of goal nodes (maybe only one)
- Each state occurs only once.

PORTION OF STATE SPACE GRAPH FOR 8-PUZZLE



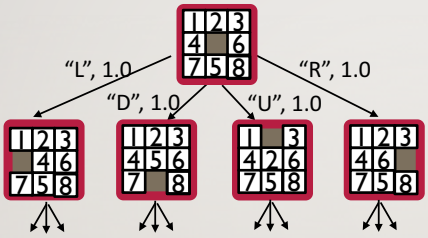
PORTION OF STATE SPACE GRAPH FOR 8-PUZZLE



STATE SPACE SIZES

- 8 puzzle
  - 9! = 362880 if all states reachable
  - Actually (9!/2) = 181440
- Go:
  - Est. 10^{79}
- To put that number into perspective
  - What is the estimate of the number of atoms in the *known* universe?
  - Est. 10^{78} to 10^{82}

SEARCH TREES



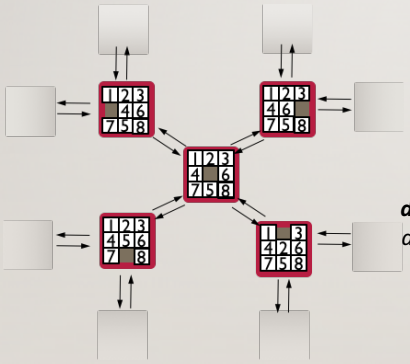
Start State

Possible Next States

- A search tree:
  - A "what if" tree of action sequences and their outcomes
  - The start state is the root node
  - Children correspond to the outcomes of actions (successors)
  - Nodes show states, but correspond to action sequences that achieve those states

STATE SPACE GRAPHS VS. SEARCH TREES

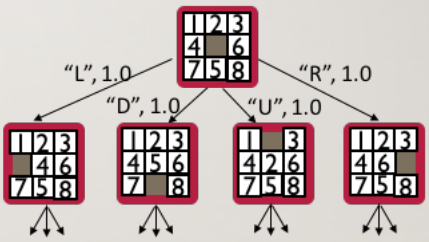
State Space Graph



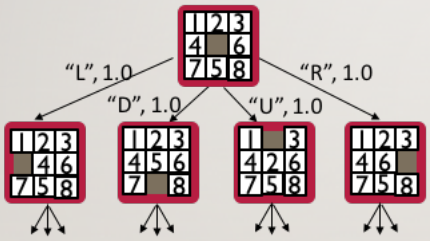
Each NODE in in the search tree represents an entire PATH in the state space graph.

***Constructed on demand** –construct as little as possible.*

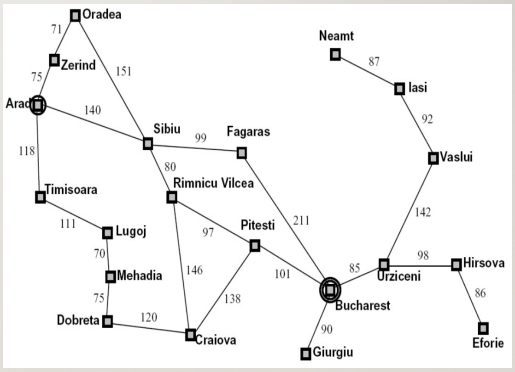
Search Tree



HOW BIG IS THE SEARCH TREE FOR THESE PROBLEMS



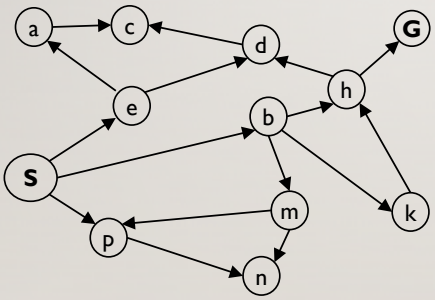
- What are the next states that result from these 4 states?



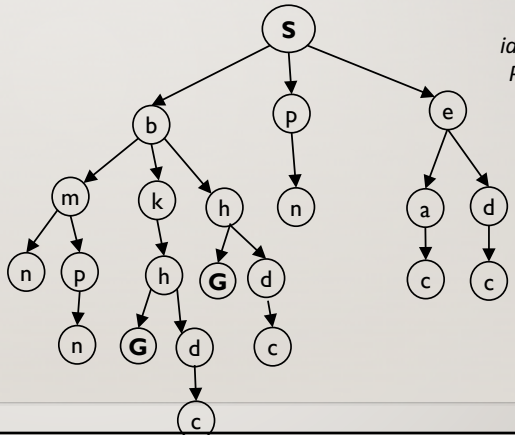
- What are the next states that result from these 4 states?

STATE SPACE GRAPHS VS. SEARCH TREES

State Space Graph



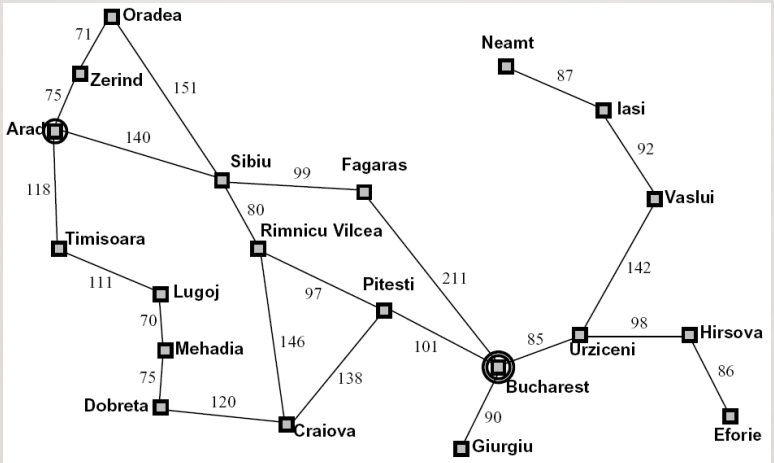
Search Tree



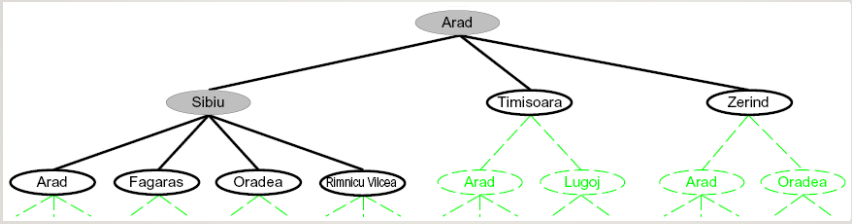
A NODE in in the search tree identifies an entire PATH in the state space graph.

TREE SEARCH

SEARCH EXAMPLE: ROMANIA



SEARCHING WITH A SEARCH TREE



- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a frontier of partial plans (search paths) under consideration
  - Try to expand as few tree nodes as possible

GENERAL TREE 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

- Main question: which frontier node to choose to expand?
- Different Search methods have different strategies for choosing the node to expand

TREE SEARCH

VERSUS

GRAPH SEARCH

KEEP IN MIND WHEN YOU DO THE PROBLEM SET

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

TERMINOLOGY

- STATE SPACE GRAPH
  - Representation of a search problem showing states of the problems and transitions between them
- SEARCH TREE
  - Representation of possible sequences of actions (transitions) from some start state
- TREE SEARCH
  - Algorithm for actually searching for a solution
- GRAPH SEARCH
  - Algorithm for searching for a solution **that does not re-visit already explored nodes**

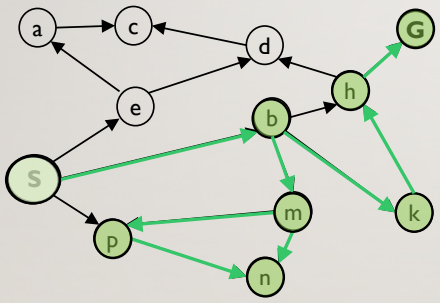
DEPTH-FIRST TREE SEARCH

DEPTH-FIRST TREE SEARCH

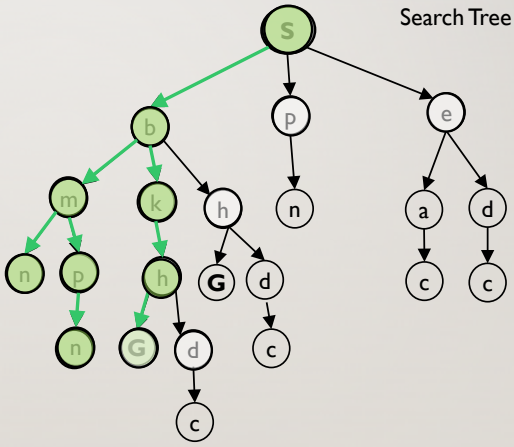
- DEPTH FIRST strategies expand the deepest node in the search tree
- Equivalent to expanding the node most recently added to the frontier

DEPTH-FIRST SEARCH

State Space Graph



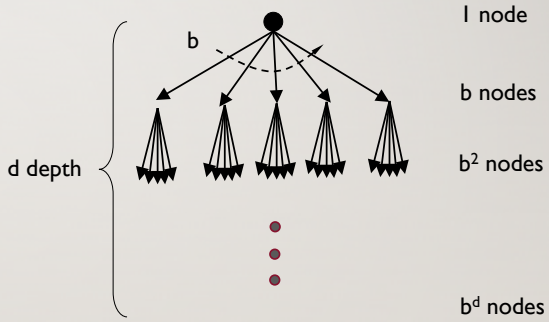
Search Tree



SEARCH ALGORITHM PROPERTIES

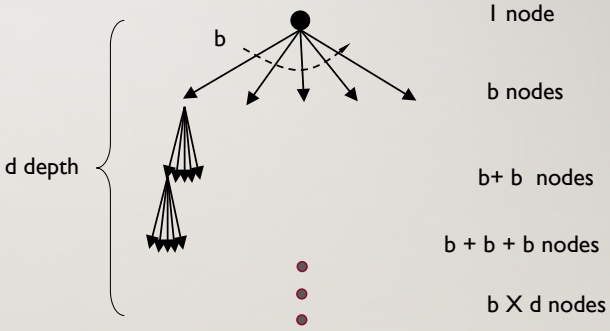
SEARCH ALGORITHM PROPERTIES

- Completeness?
  - Does it find a solution if one exists?
- Optimal?
  - Does it find the lowest cost path?
- Time complexity?
  - How Lon does it take?
- Space complexity?
  - How much memory is needed
- Search tree properties:
  - b is the (max) branching factor
  - d is the maximum depth
  - solutions at various depths
- Number of nodes in entire tree?
  - $1 + b + b^2 + \ldots. b^d = O(b^d)$



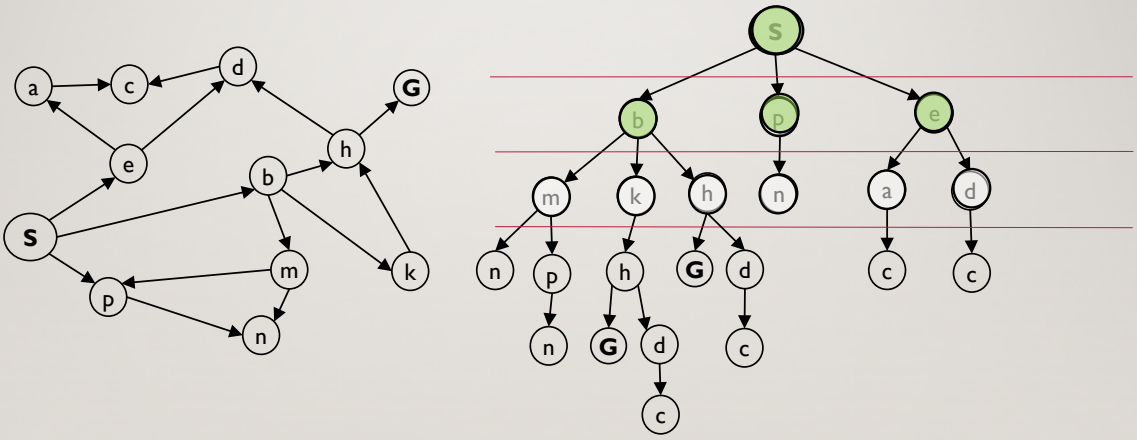
DEPTH-FIRST SEARCH (DFS) PROPERTIES

- What nodes DFS expand?
  - Some left prefix of the tree.
  - Could end up generating the whole tree!
  - If d (depth) is finite, takes time $O(b^d)$
- How much space does the frontier take?
  - Needs to keep track of siblings on the path to root,
  - So O(bd)
- Complete?
  - d could be infinite, so only if cycles are prevented
- Optimal?
  - No, finds the "leftmost" solution, regardless of depth



BREADTH-FIRST SEARCH

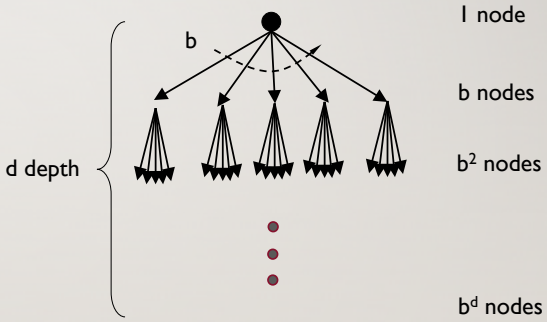
BREADTH-FIRST SEARCH



All nodes are expanded at a given depth before any nodes are expanded at the next level

BREADTH-FIRST SEARCH (BFS) PROPERTIES

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
  - Search takes time $O(b^s)$
- How much space does the frontier take?
  - Has roughly the last level, so $O(b^s)$
- Is it complete?
  - s must be finite if a solution exists, so yes if branching is finite
- Is it optimal?
  - Only if costs are all 1 (more on costs later)



DIFFERENCE BETWEEN DFS AND BFS ?

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

- DFS: LIFO Queue (Stack)
  - Can use a recursive function that calls itself on each child node in turn
- BFS: FIFO Queue

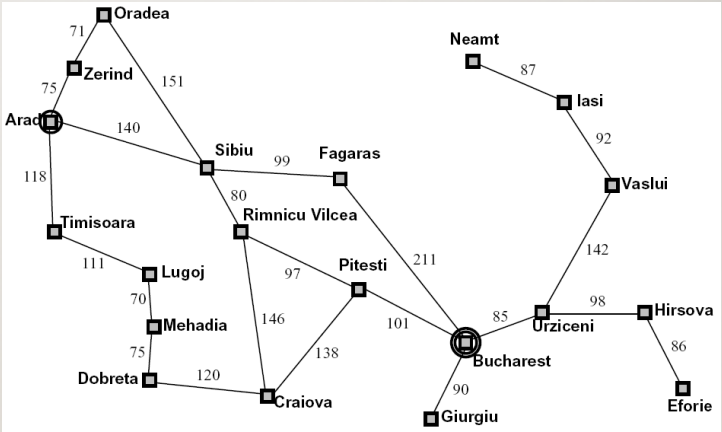
QUESTION: DFS VS BFS

- When will BFS outperform DFS?
- When will DFS outperform BFS?

COST-SENSITIVE SEARCH

BFS finds shortest path in terms of number of actions.

- But actions may have costs.
- Romania map example
  - Miles traveled
  - Condition of roads
  - Traffic
- BFS only finds the least-cost path if actions have same costs



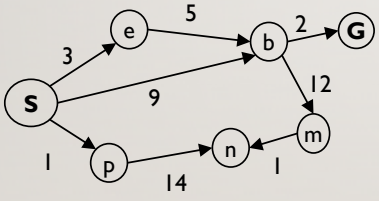
UNIFORM COST SEARCH

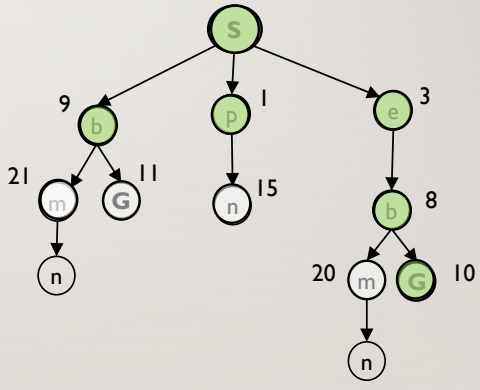
UNIFORM COST SEARCH

Strategy: expand a cheapest node (path) first:

Frontier maintained as priority queue

(cumulative cost along the path)





UNIFORM COST SEARCH PROPERTIES

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)
- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$
- Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!
- Is it optimal?
  - Yes! (more later)

UNIFORM COST ISSUES

- The bad:
  - Like BFS and DFS doesn't take into account where it is going
  - No information about goal location

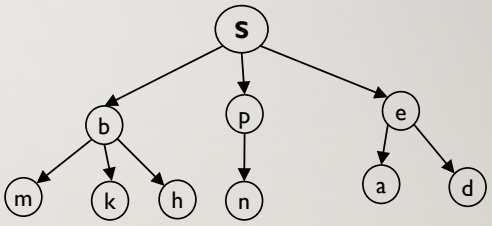
OTHER VARIANTS OF UNINFORMED SEARCH

ITERATIVE DEEPENING DEPTH FIRST SEARCH

- Function IDDFS (problem)
  - For D = 0 to ∞ do
    - result = do DFS to depth D
    - If result ≠ FAIL then return result
- Why may this be a good idea?
  - What good features of DFS and BFS does this capture?
- Why does this seem to be a bad idea?

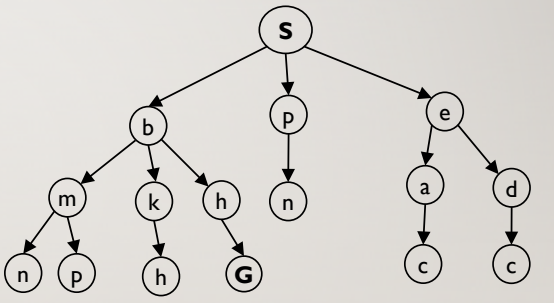
ITERATIVE DEEPENING DEPTH FIRST SEARCH

- DFS's space advantage with BFS's time / shallow solution advantages
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. If no solution…
  - Run a DFS with depth limit 4 …..
- Seems wasteful, but often not wasteful
  - Most work happens in the lowest level searched, so…
  - The upper levels are not [illegible] in large search spaces.



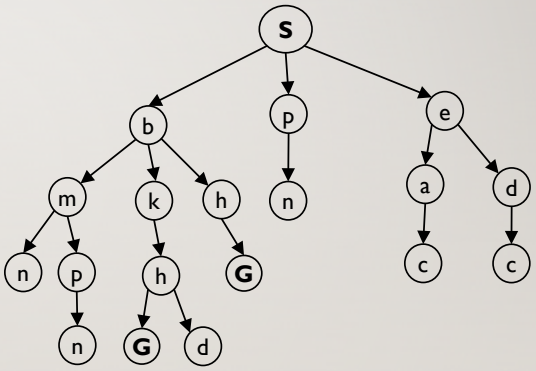
ITERATIVE DEEPENING DEPTH FIRST SEARCH

- DFS's space advantage with BFS's time / shallow solution advantages
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. If no solution…
  - Run a DFS with depth limit 4 …..
- Seems wasteful, but often not wasteful
  - Most work happens in the lowest level searched, so…
  - The upper levels are not [illegible] in large search spaces.

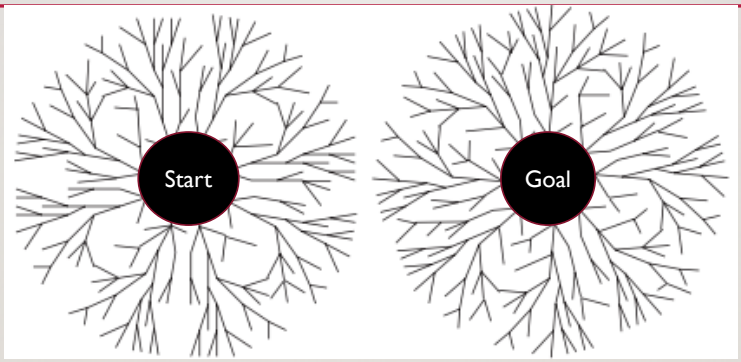


ITERATIVE DEEPENING DEPTH FIRST SEARCH

- DFS's space advantage with BFS's time / shallow solution advantages
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. If no solution…
  - Run a DFS with depth limit 4 …..
- Seems insane but often not wasteful
  - Most work happens in the lowest level searched, so not so bad!
  - The uninformed search preferred in large search spaces (early uses came out of chess work)



BIDIRECTIONAL SEARCH

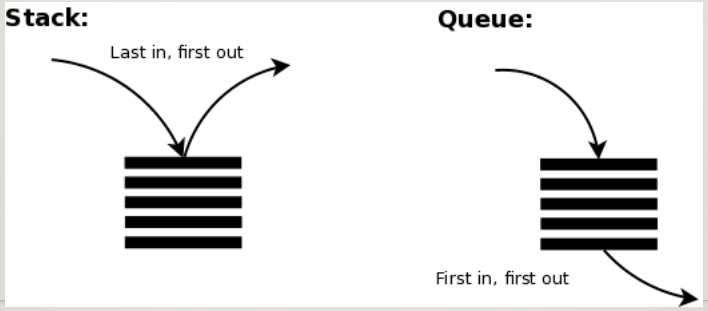


- How does this change the criteria for success?
- Why may Bidirectional Search be a good idea?
- What does it presume?

ADDENDUM AND COMMENTS

KEY DIFFERENCE IN THESE SEARCH METHODS IS HOW THE QUEUE IS MAINTAINED

- All frontiers are priority queues (i.e. collections of nodes with attached priorities)
  - For DFS and BFS, you can use stacks and queues



WHEN IS THE GOAL TEST APPLIED?

- When the goal test is applied differs among algorithms
- When could you apply the goal test for BFS?
  - When it is generated
- When should you apply the goal test for UCS?
  - When it is selected for expansion

PROPERTIES DIFFER DEPENDING ON WHETHER USING GRAPH OR TREE SEARCH

- DFS using graph search
  - Avoids repeated states and redundant paths
  - Complete? Yes in finite spaces
  - Space complexity governed by maintaining the explored list
- DFS using tree search
  - Not complete even in finite spaces
  - Space Complexity: Need only store path to root along with unexpanded siblings
- Space complexity is often the key concern

SEARCH USES ABSTRACT MODELS

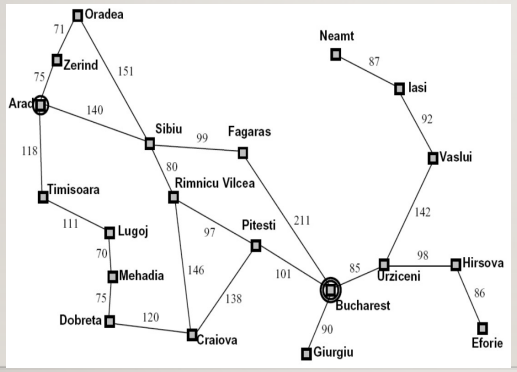
- Search operates over models of the world
  - The agent doesn't actually try all the plans out in the real world!
  - Planning is all "in simulation"
  - The search is only as good as your models…

SEARCH DEPENDS ON ACCURATE MODEL OF THE WORLD

- Your model may be wrong from the start
  - In November 2010, Nicaraguan troops unknowingly crossed the border to Costa Rica, removed that country's flag and replaced it with their own. The reason: Google Maps told the troops' commander the territory belonged to Nicaragua.
- The mapping between the representation of the solution and world may not be obvious
  - Is this the third or second turn off the rotary?

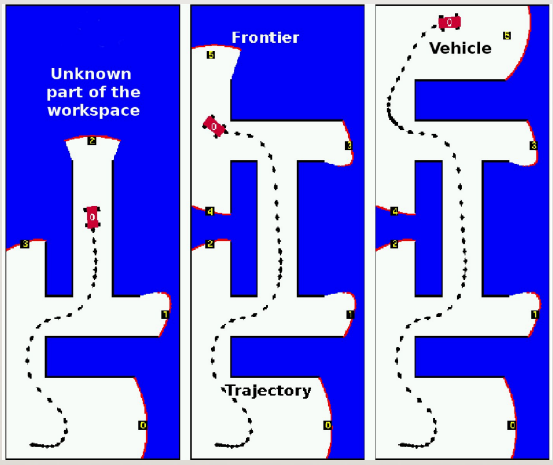
ALSO THE WORLD CAN CHANGE INVALIDATING THE MODEL

- The world can change
  - while searching for the solution
  - while executing the solution



PLANNING OUT A SOLUTION TAKES TIME

PLANNING OUT A COMPLETE SOLUTION ALSO REQUIRES A MODEL



Pracsys Lab
Rutgers

ALSO THE ABOVE DISCUSSION OF SEARCH ASSUMES COMPLETE SOLUTIONS

Would/COULD you plan complete solutions to these problems?
Why not?

7	2	4
5		6
8	3	1

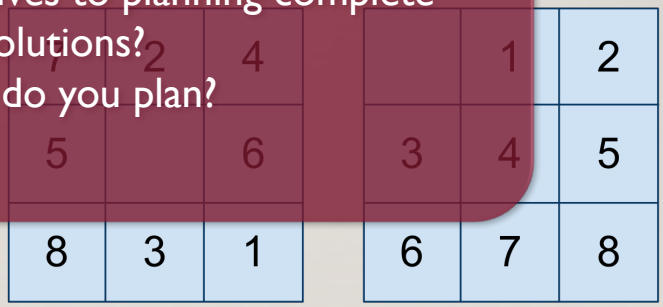
Start State

	1	2
3	4	5
6	7	8

Goal State

WOULD YOU PLAN COMPLETE SOLUTIONS?

What are alternatives to planning complete solutions?
How do you plan?



Start State Goal State