Intelligent Agents

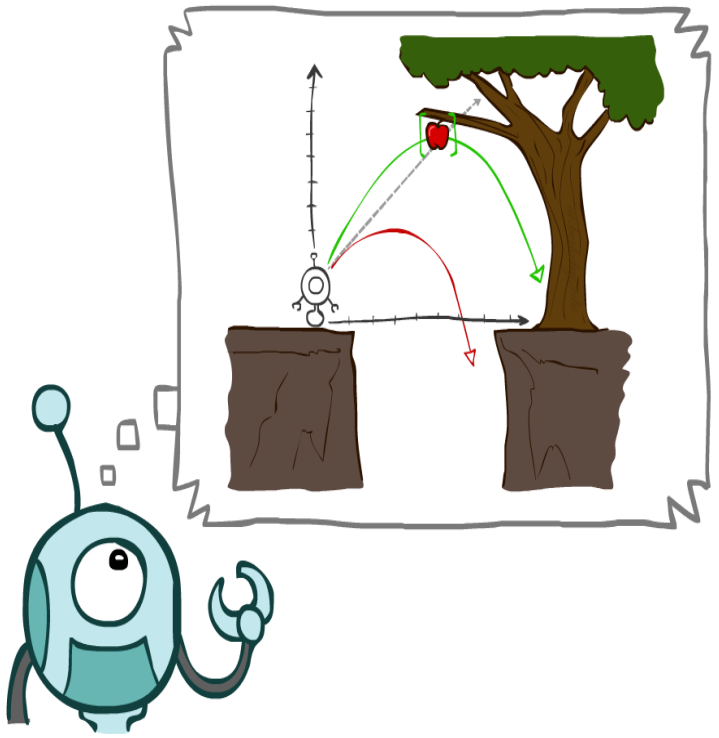
Artificial Intelligence

By

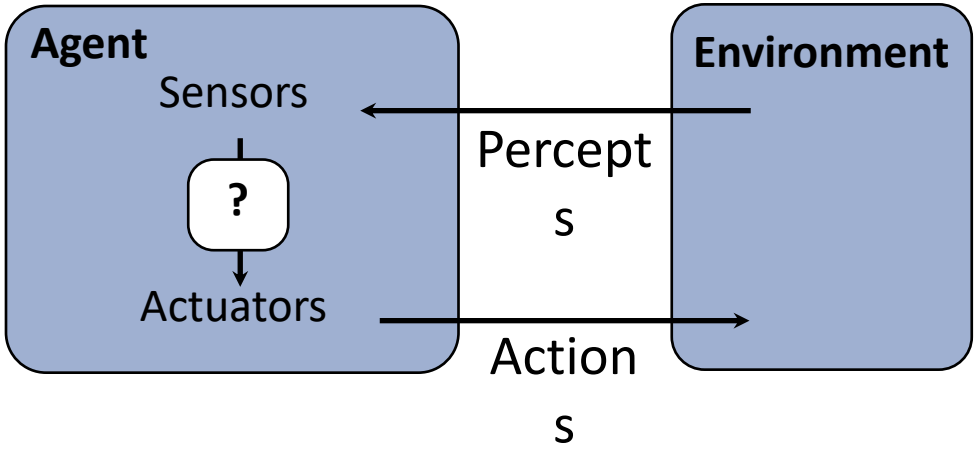
Ayub Othman Abdulrahman

Today

- Agents that Plan Ahead
- Search Problems
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search



Agents and environments



- An agent ***perceives*** its environment through ***sensors*** and ***acts*** upon it through ***actuators***

Rationality

- A ***rational agent*** chooses actions maximize the ***expected*** utility
  - Today: agents that have a goal, and a cost
    - E.g., reach goal with lowest cost
  - Later: agents that have numerical utilities, rewards, etc.
    - E.g., take actions that maximize total reward over time (e.g., largest profit in $)

Specifying the task environment

Agent Type	Performance Measure	Environment	Actuator	Sensor
Taxi driver	Safe, fast legal, comfortable trip, maximize profits.	Roads, other traffics, pedestrians, customer	Steering, Accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, engine sensors.

Agent design

- **The environment type largely determines the agent design**
  - ***Fully/partially observable*** => agent requires ***memory*** (internal state)
  - ***Discrete/continuous*** => agent may not be able to enumerate ***all states***
  - ***Stochastic/deterministic*** => agent may have to prepare for ***contingencies***
  - ***Single-agent/multi-agent*** => agent may need to behave ***randomly***
  - ***Episodic vs. sequential: Chess and taxi driver***
  - ***Static vs. dynamic: deliberating***
  - ***Known vs. unknown: "law of physics)***

The structure of agents

- The job of AI is to design an agent program that implements the agent function the mapping from percept to actions.

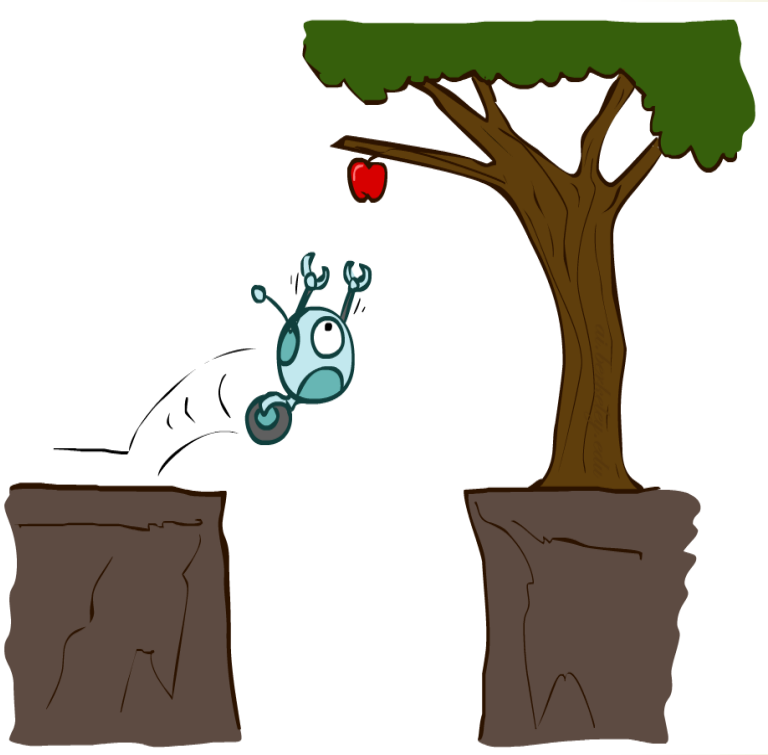
- Agent= architecture + program.

Agent Program

- Simple reflex agents
- Model-based reflex agents
- Goal-based agents
- Utility-based agents

Simple Reflex Agents

- Reflex agents:
  - Choose action based on current percept (and maybe memory)
  - May have memory or a model of the world's current state
  - Do not consider the future consequences of their actions
  - Consider how the world IS
- Can a reflex agent be rational?



Model-based reflex agents

- The agents should maintain some sort of internal state that depends on the percept history and thereby reflects some of the unobserved aspect if the current state.
- For braking problem, the internal state is not to extensive.
- Just previous frame from the camera, allowing the agent to detect when two red lights at the edge of the vehicle go on or off simultaneously.

Goal-based agents

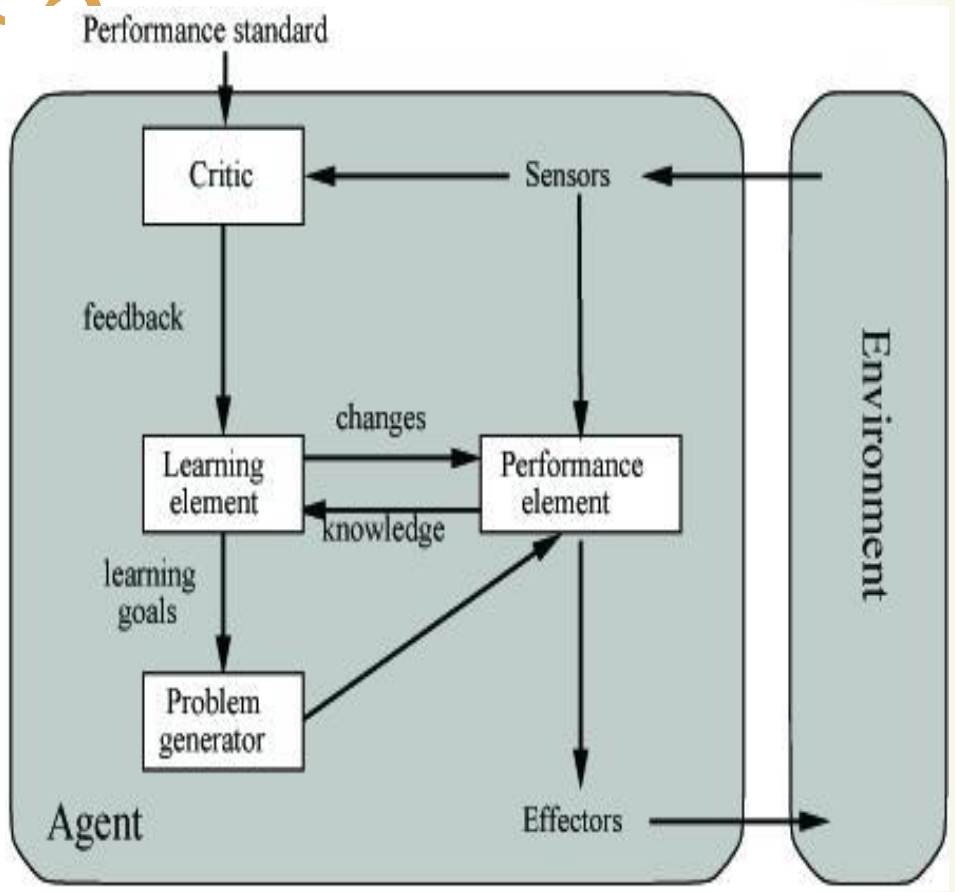
- Knowing something about the current state of the environment is not always enough to decide what to do.
- For example, at a road junction, the taxi can turn left, turn right, go straight on.
- The correct decision depends on where the taxi trying to get to.

Utility-based agents

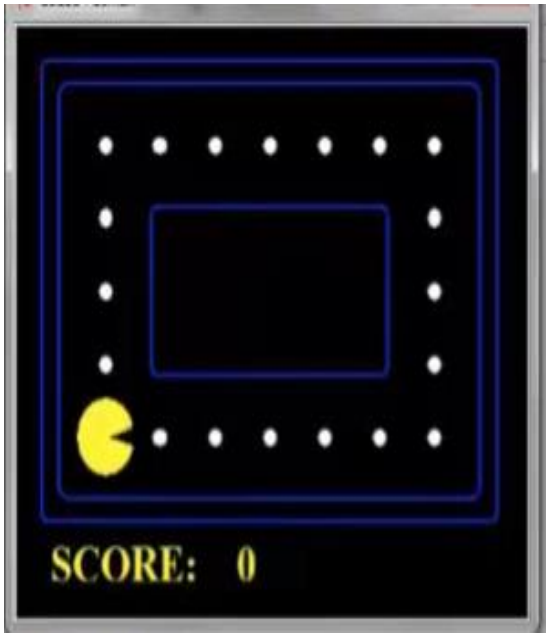
- Goals alone are not enough to generate high-quality behavior in most environments.
- For example, many actions sequences will get the taxi to its destination but
- Some are quicker, safer, more reliable or cheaper than others.

Learning agents

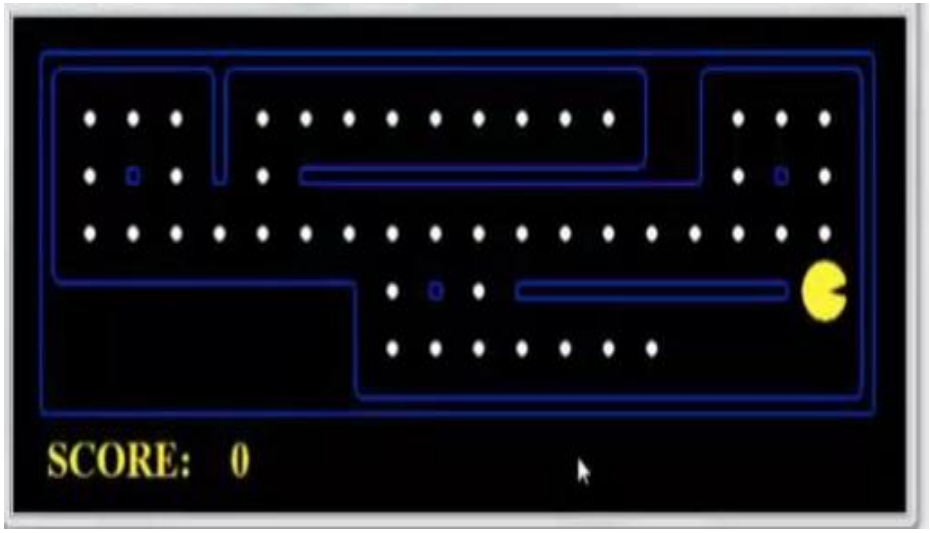
- Turing method is "proposing method to build learning machine and then to teach them"



Video of Demo Reflex Optimal

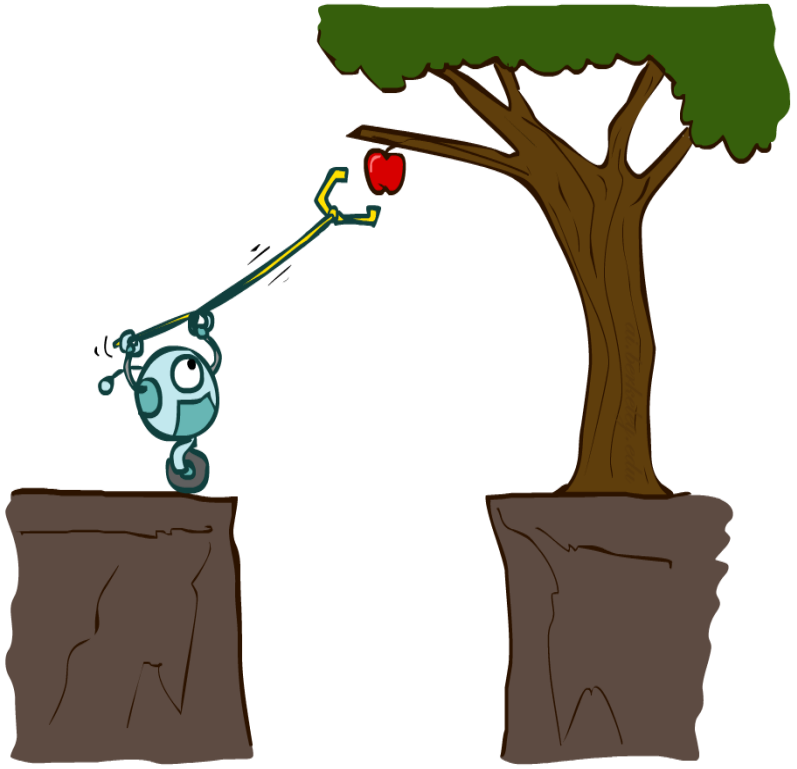


Video of Demo Reflex Odd

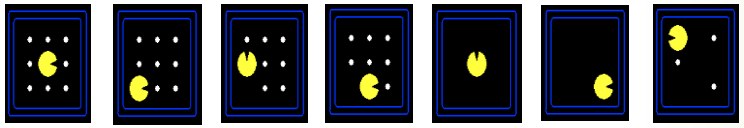


Planning Agents

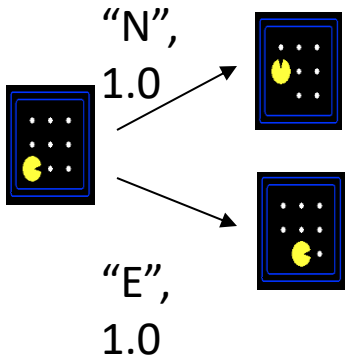
- Planning agents:
  - Ask "what if"
  - Decisions based on (hypothesized) consequences of actions
  - Must have a model of how the world evolves in response to actions
  - Must formulate a goal (test)
  - Consider how the world WOULD BE
- Optimal vs. complete planning
- Planning vs. replanning



Search Problems



- A search problem consists of:
  - A state space
  - A successor function (with actions, costs)
  - 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

Well-defined Problems and Solutions

- Initial state
- Action available to the agent
- Transition model: a description of what each action does.
- State space: initial state, actions and transition model define the state space.
- Goal test: it reaches the goal or not!
- Path cost: assign numeric cost to each path.

Toy problem: vacuum world

- States: both agent location and dirt location (2 location)
- 2*2^2=8 possible states.(n.2^n)= states.
- Initial state: any state can be designed as initial state.
- Actions: three actions: left, right, and suck.
- Transition model: actions have their expected affect.
- Goal test: checks whether all the squares are clean.
- Path cost: each step costs 1.

Vacuum world

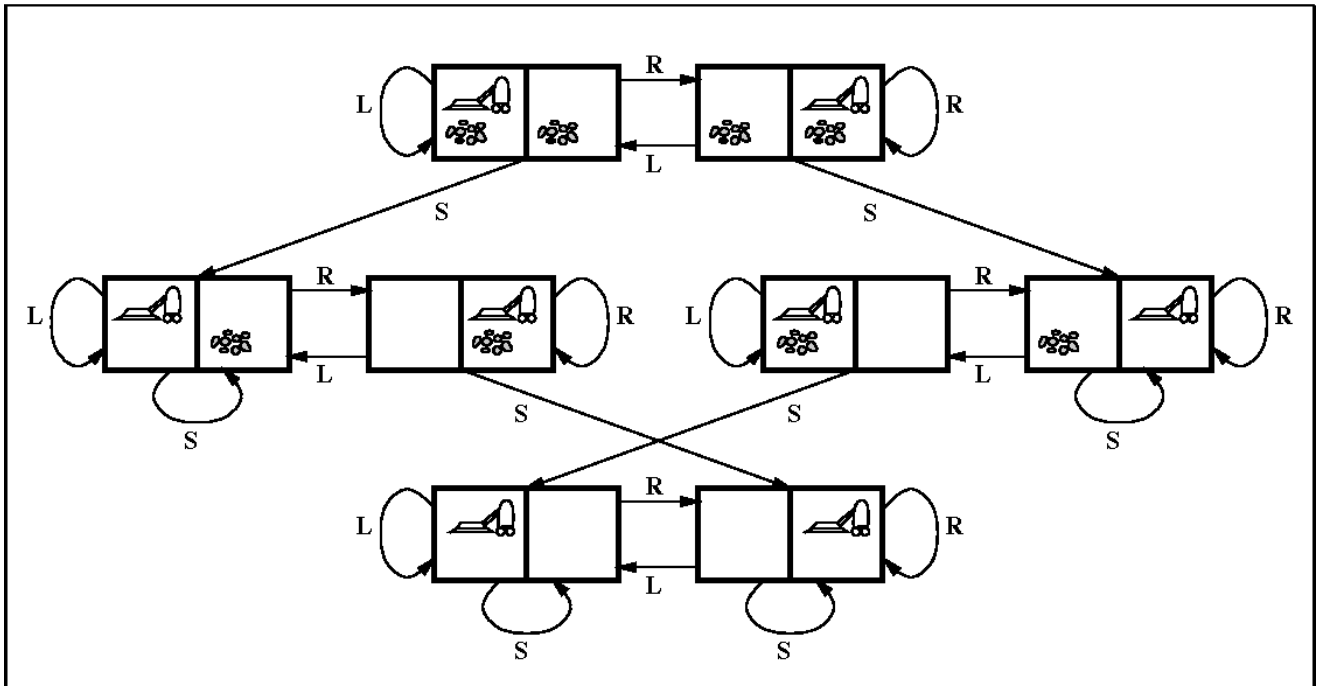
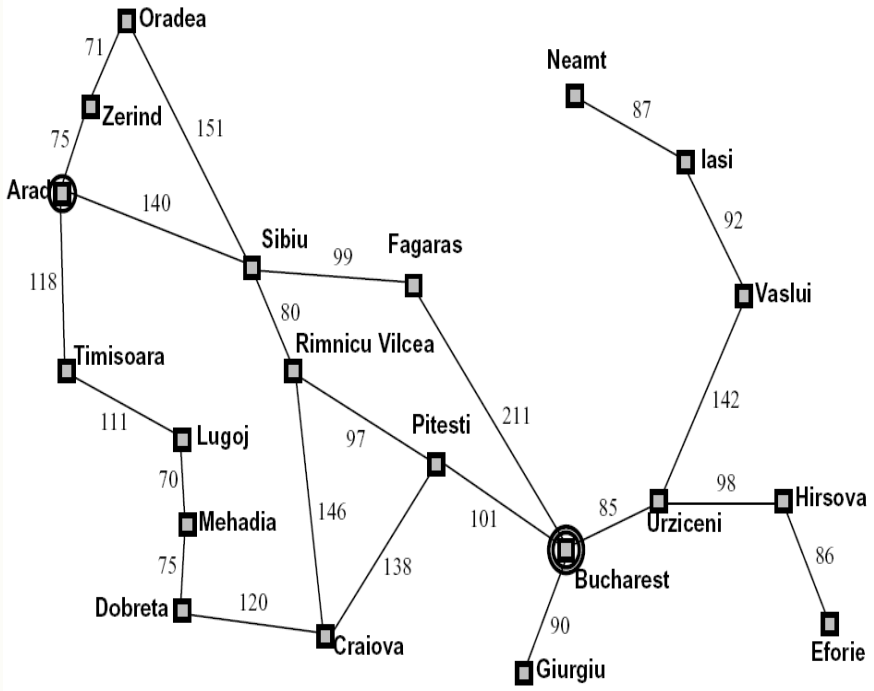


Figure 3.3 The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

Example: Traveling in Romania

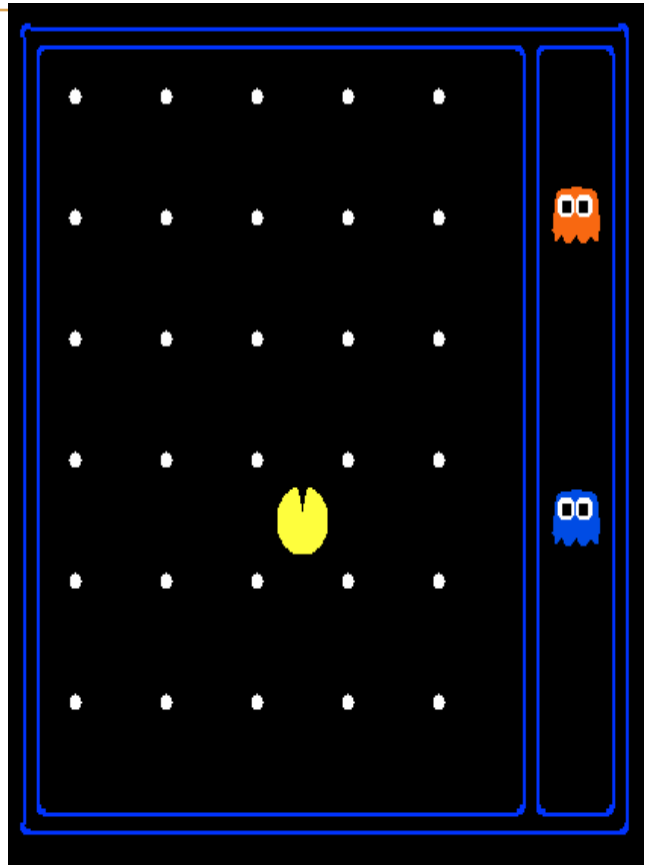


- State space:
  - Cities
- Successor function:
  - Roads: Go to adjacent city with cost = distance
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?
- Solution?

State Space Sizes?

- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW

- How many
  - World states?
    $120 \times (2^{30}) \times (12^2) \times 4$
  - States for pathing? 120
  - States for eat-all-dots? $120 \times (2^{30})$

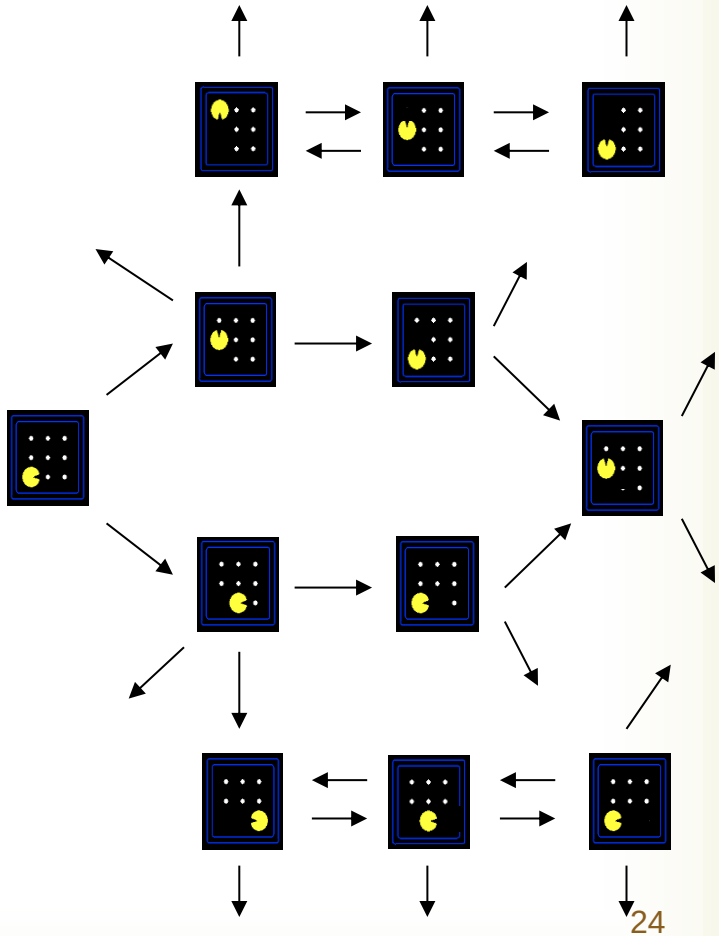


Measuring Problem-solving performance

- Completeness: is algorithm guaranteed to find a solution when there is one?
- Optimality: does the strategy find the optimal solution?
- Time complexity: how long it takes to find a solution?
- Space complexity: how much memory is needed to perform the search?

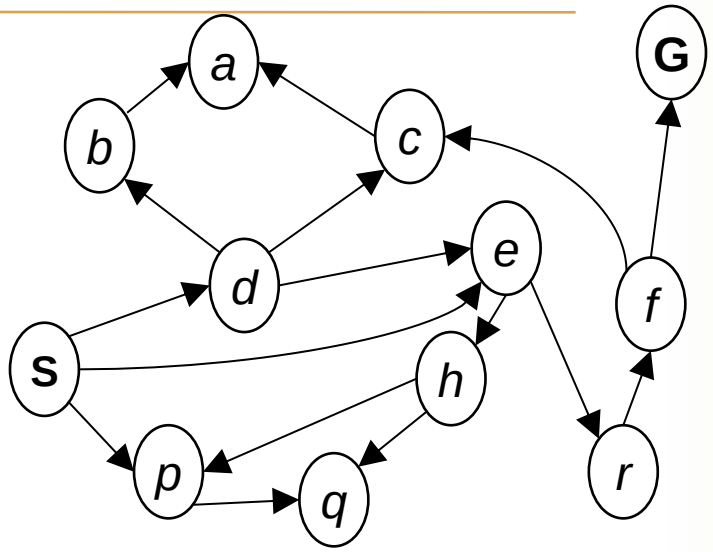
State Space Graphs

- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent successors (action results)
  - The goal test is a set of goal nodes (maybe only one)
- In a state space graph, each state occurs only once!
- We can rarely build this full graph in memory (it's too big), but it's a useful idea



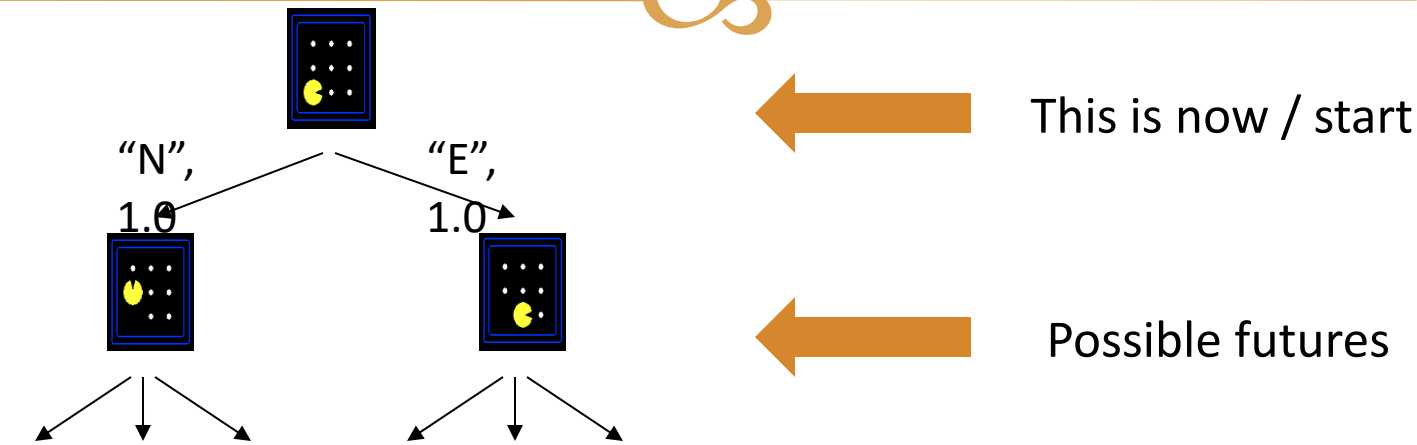
State Space Graphs

- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent successors (action results)
  - The goal test is a set of goal nodes (maybe only one)
- In a state space graph, each state occurs only once!
- We can rarely build this full graph in memory (it's too big), but it's a useful idea



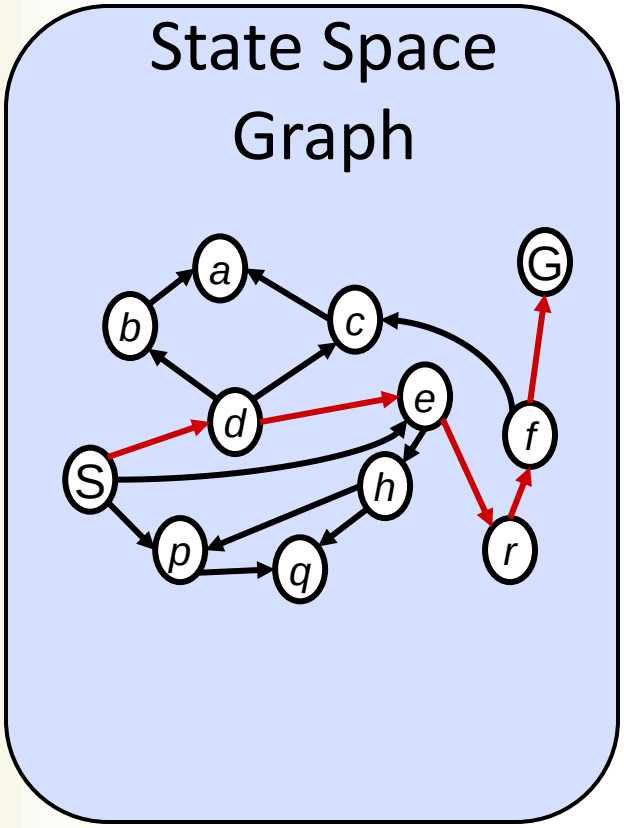
Tiny state space graph for a tiny search problem

Search Trees



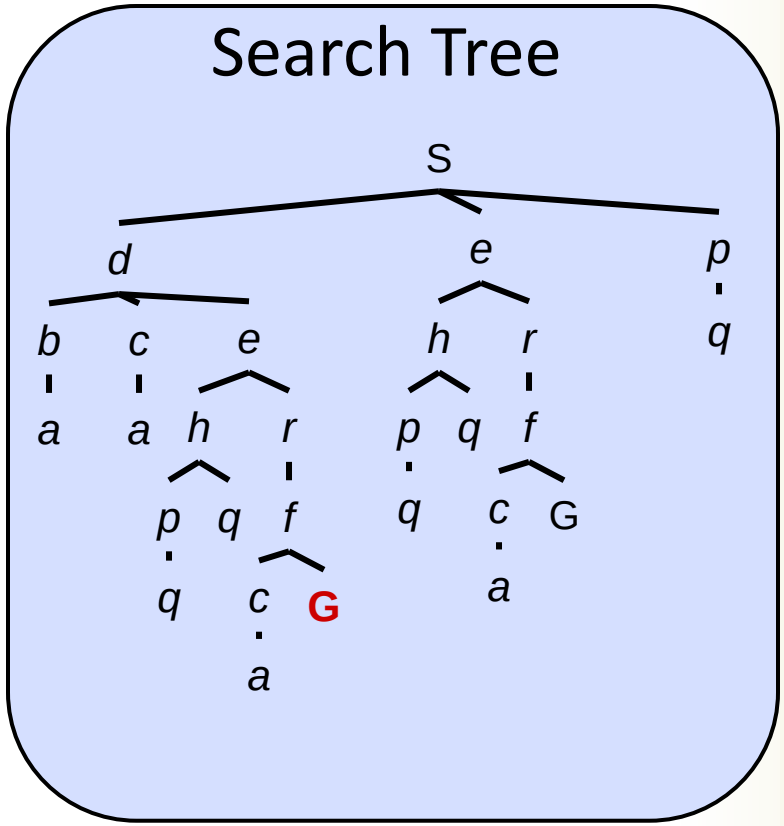
- A search tree:
  - A "what if" tree of plans and their outcomes
  - The start state is the root node
  - Children correspond to successors
  - Nodes show states, but correspond to PLANS that achieve those states
  - For most problems, we can never actually build the whole tree

State Space Graphs vs. Search Trees

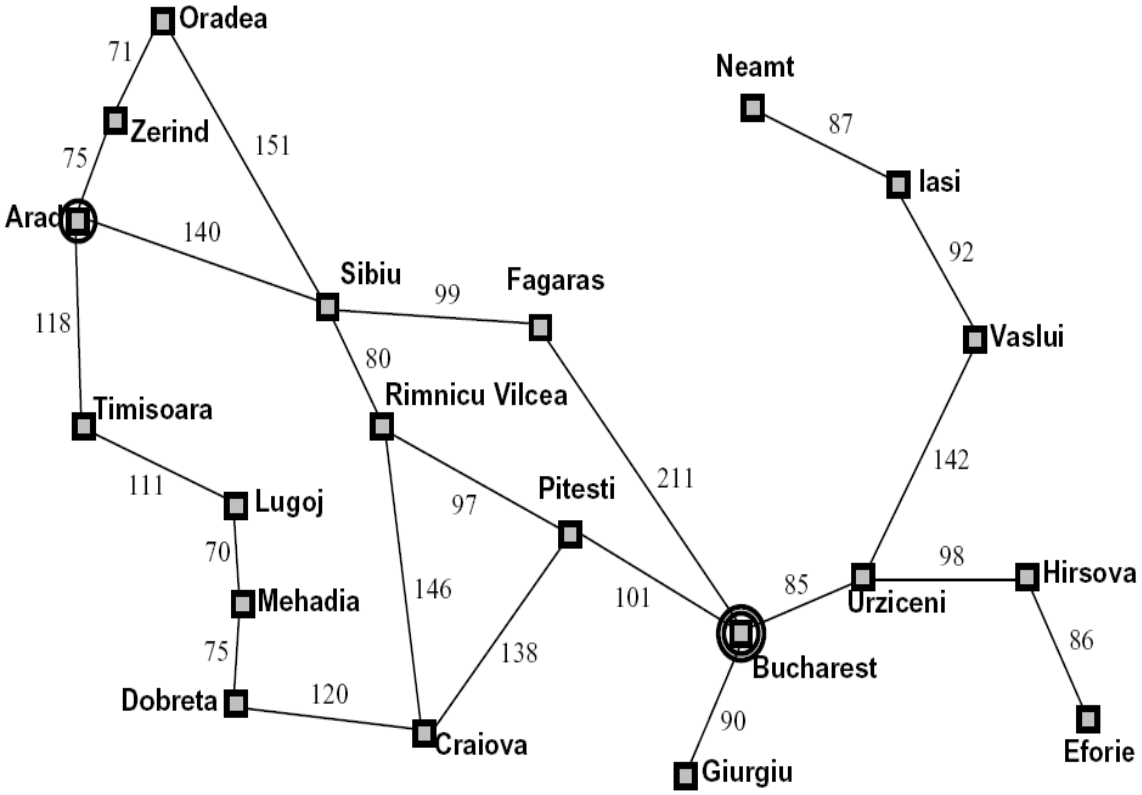


Each NODE in in the search tree is an entire PATH in the state space graph.

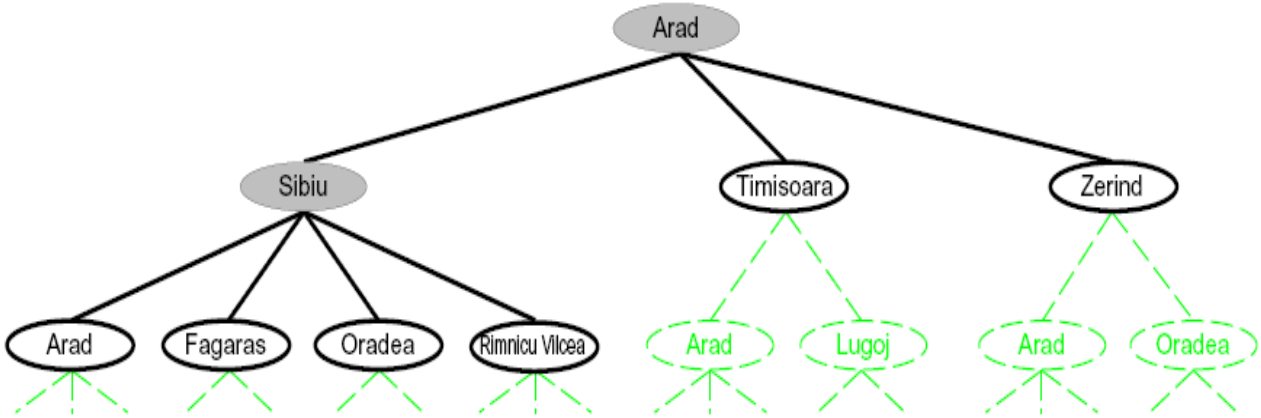
We construct both on demand – and we construct as little as possible.



Search Example: Romania



Searching with a Search Tree

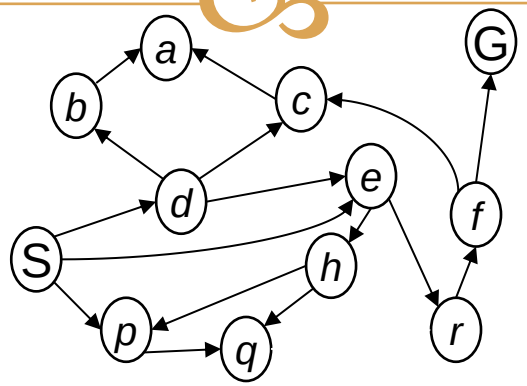


- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a fringe of partial plans under consideration
  - Try to expand as few tree nodes as possible

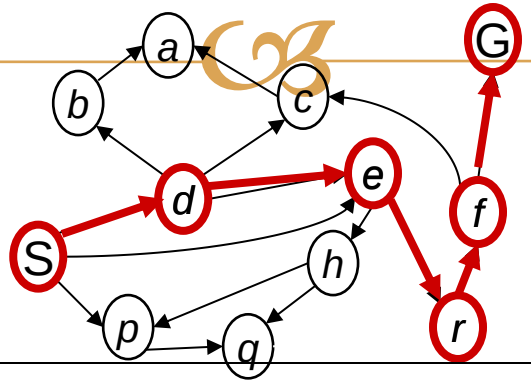
General Tree Search

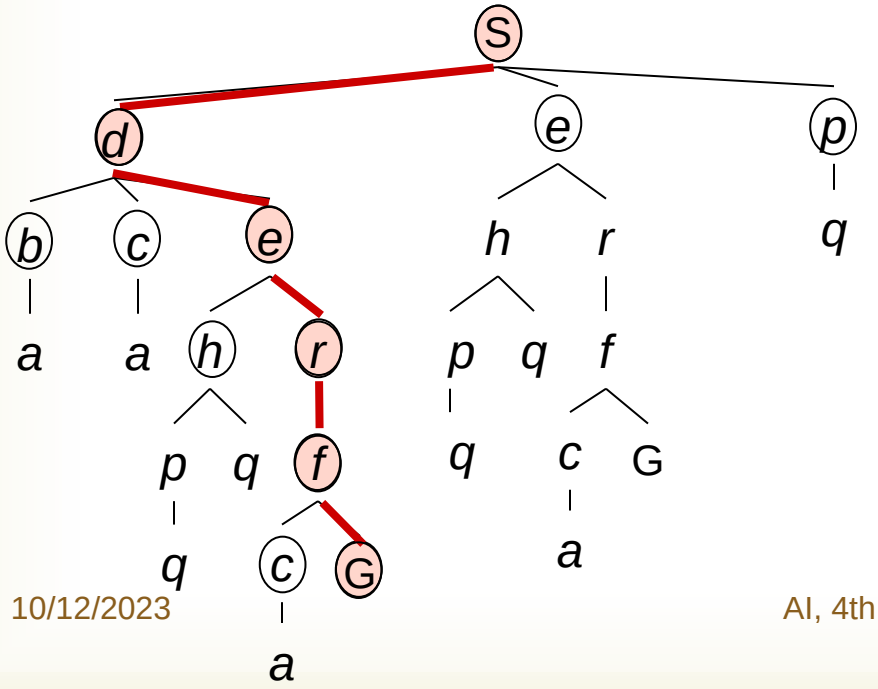
- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy
- Main question: which fringe nodes to explore?

Example: Tree Search



Example: Tree Search





~~s~~

~~s → d~~

s → e

s → p

s → d → b

s → d → c

~~s → d →~~ e

s → d → e → h

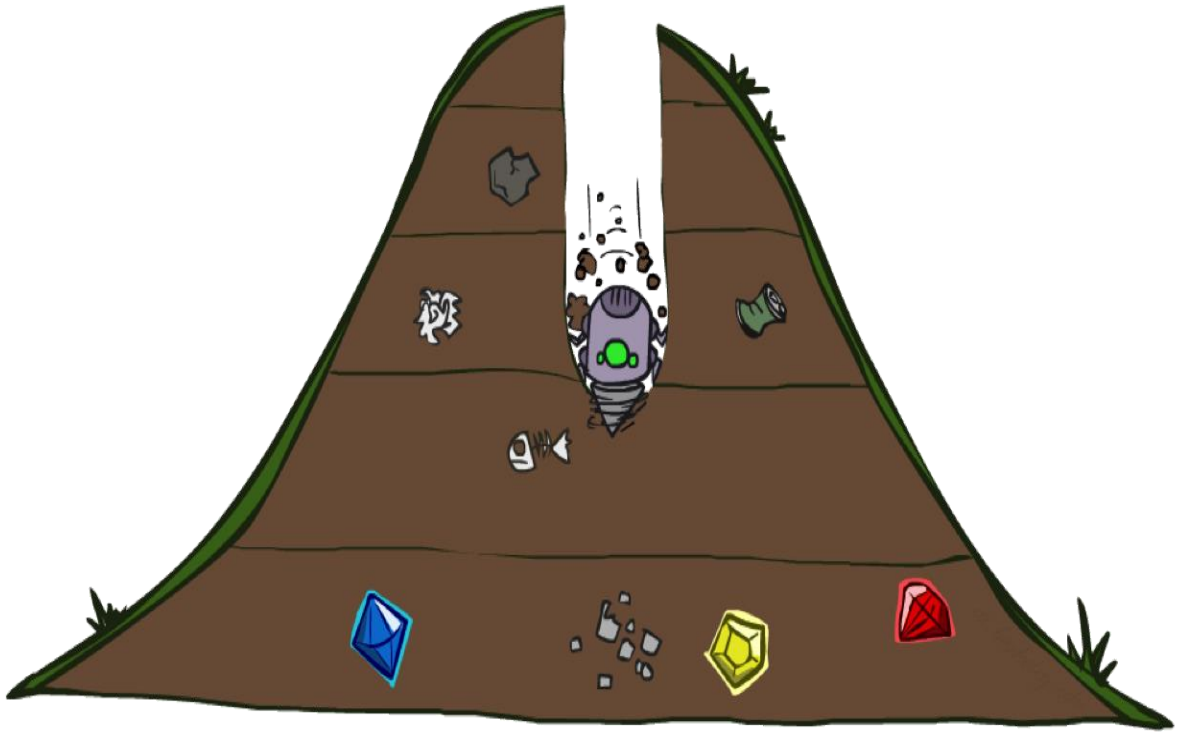
~~s → d → e →~~ r

~~s → d → e → r~~ → f

s → d → e → r → f → c

~~s → d → e → r →~~ f → G

Depth-First Search



Depth-First Search

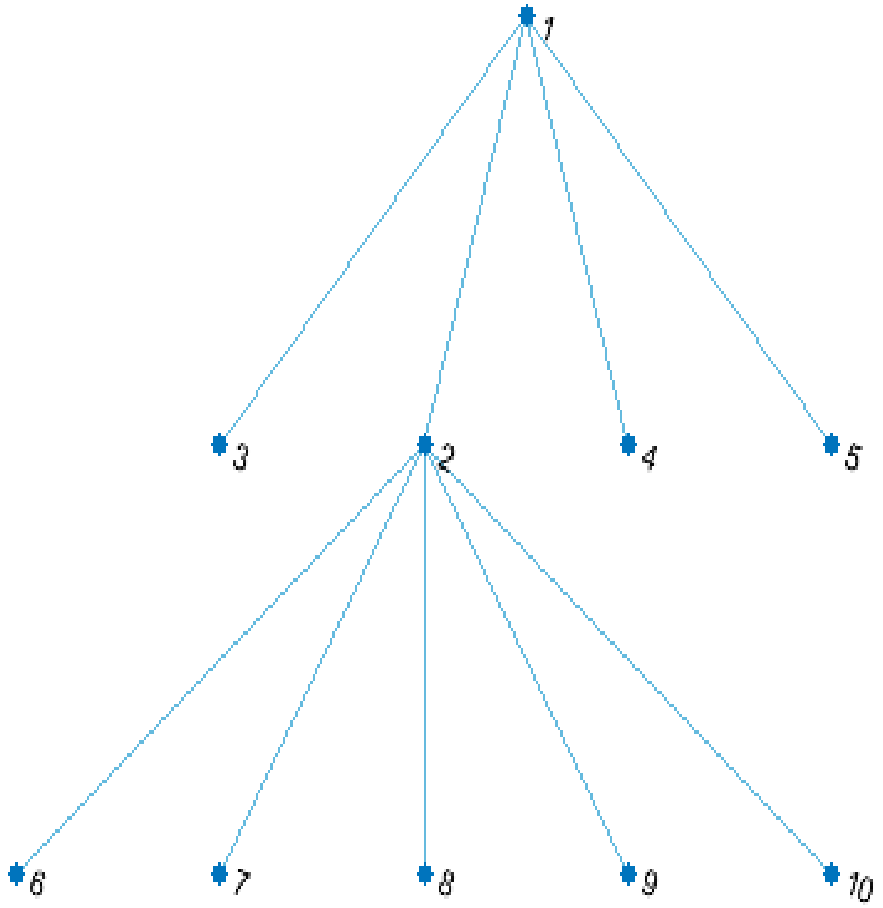
- The DFS algorithm is the search algorithm which begins the searching from the root node and goes down till the leaf of a branch at a time looking for a particular key.
- If the key is not found, the searching retraces its steps back to the point from where the other branch was left unexplored and the same procedure is repeated for that branch.

Depth-First Search

- DFS follows the following 3 steps:
  - Visit a node “S”.
  - Mark “S” as visited.
  - Recursively visit every unvisited node attached to “S”.
- Since DFS is of **recursive** nature, this can be implemented using **stacks**.
- Use this link: https://www.interviewbit.com/tutorial/depth-first-search/

Depth-First Search

```
s = [1 1 1 1 2 2 2 2 2];
t = [3 5 4 2 6 10 7 9 8];
G = graph(s,t);
plot(G)
```



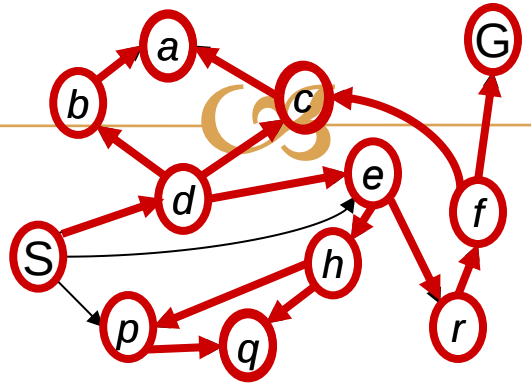
Depth-First Search

- Perform a depth-first search of the graph start
- at node 7. The result indicates the order of no
  discovery.

```
v = dfsearch(G,7)
```

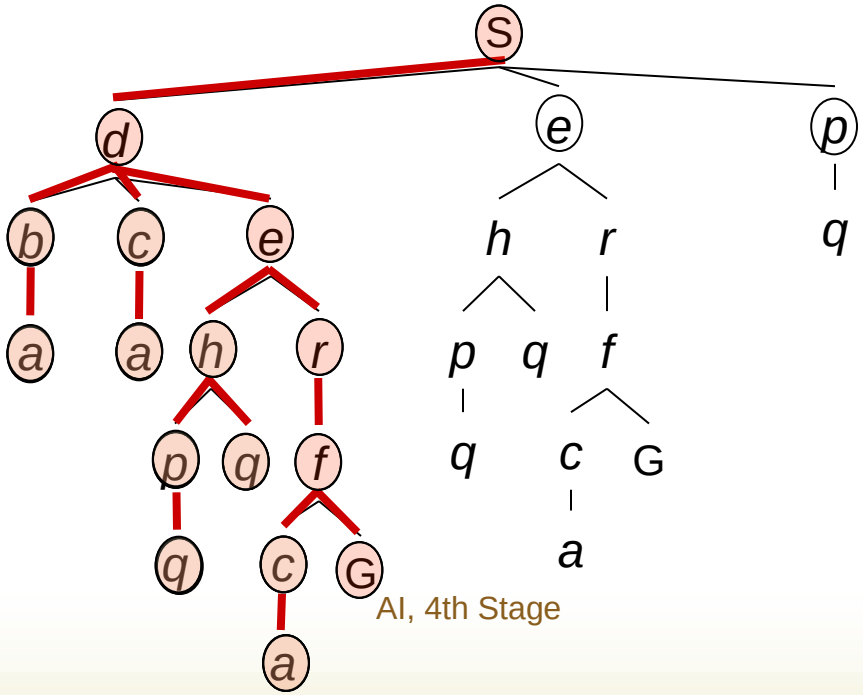
```
v = 10×1

     7
     2
     1
     3
     4
     5
     6
     8
     9
    10
```

Depth-First Search



Strategy: expand a deepest node first

Implementation : Fringe is a LIFO stack

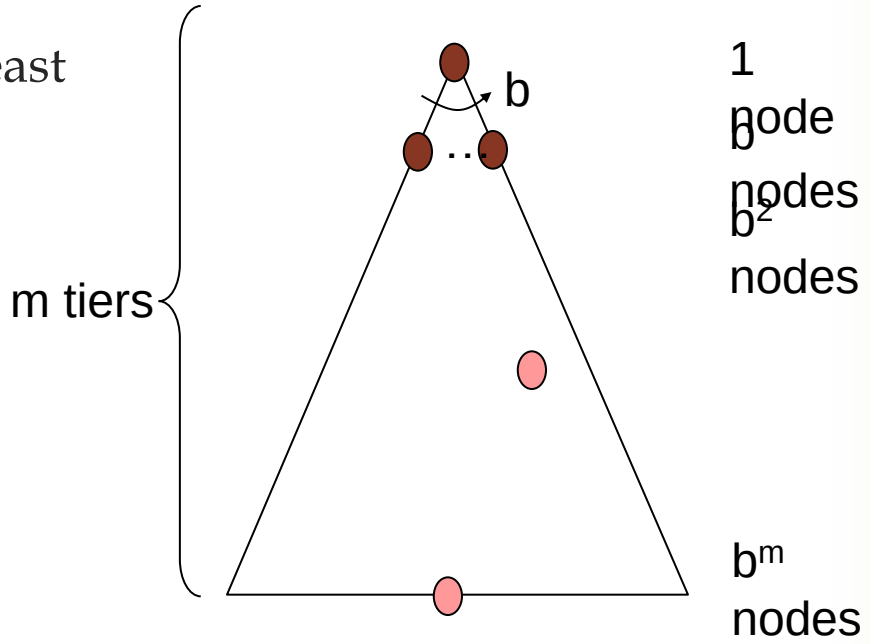


Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

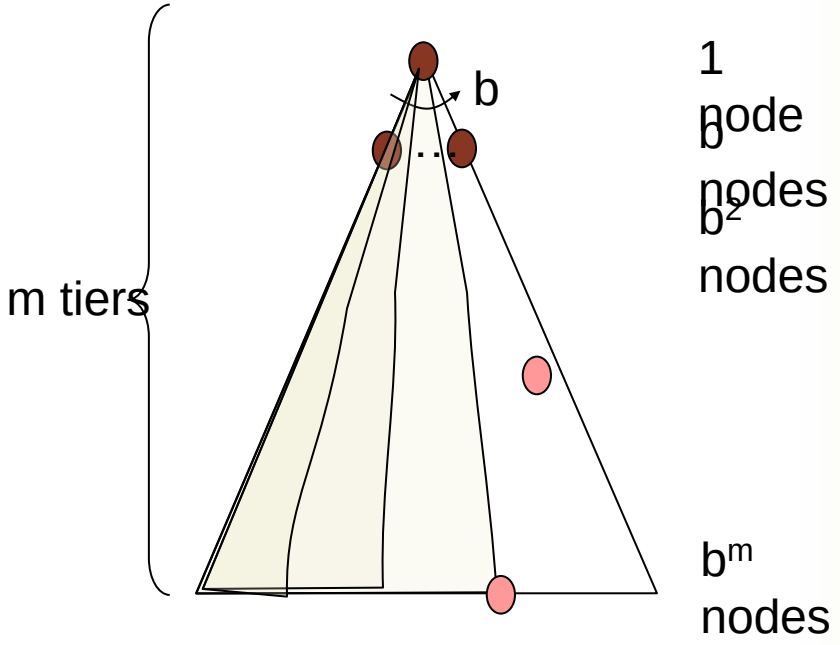
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths

- Number of nodes in entire tree?
  - $1 + b + b^2 + \ldots. b^m = O(b^m)$

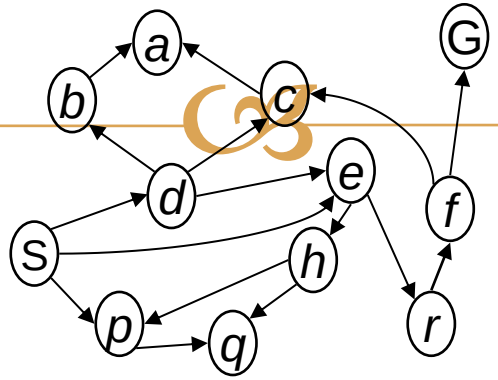


Depth-First Search (DFS) Properties

- What nodes DFS expand?
  - Some left prefix of the tree.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the fringe take?
  - Only has siblings on path to root, so $O(bm)$
- Is it complete?
  - m could be infinite, so only if we prevent cycles (more later)
- Is it optimal?
  - No, it finds the "leftmost" solution, regardless of depth or cost



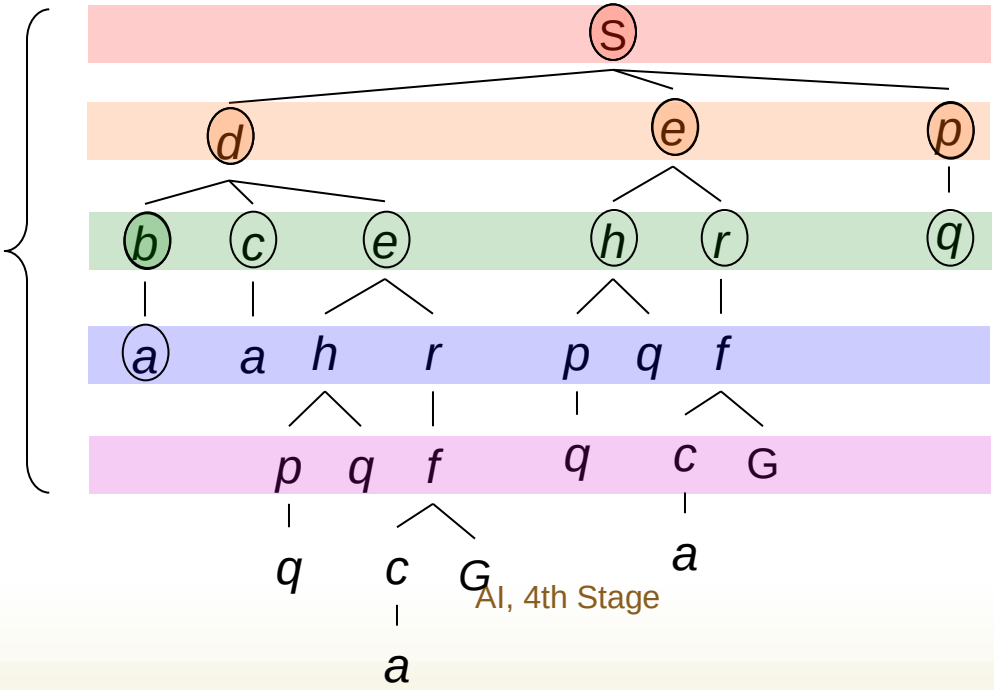
Breadth-First Search



Strategy: expand a shallowest node first

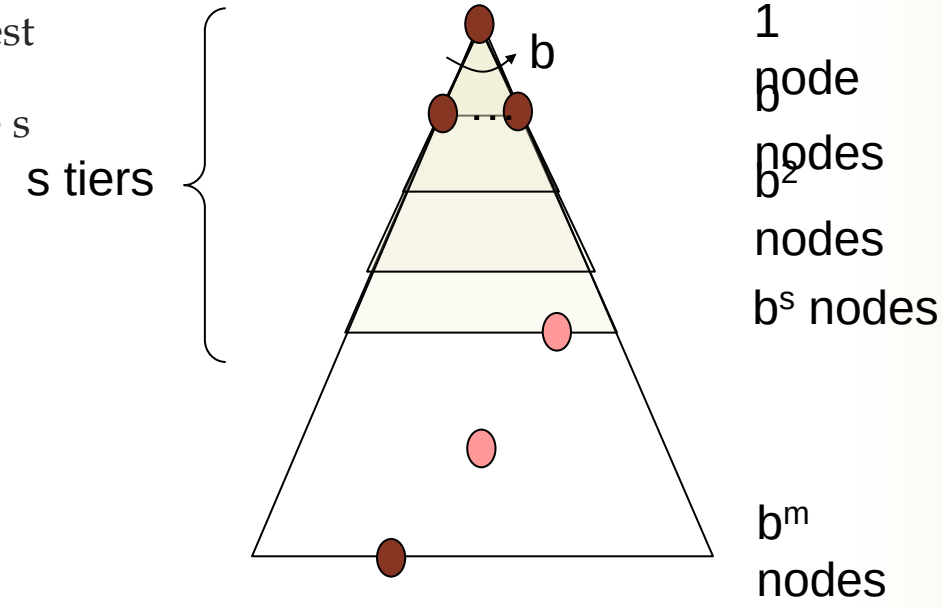
Implementation: Fringe is a FIFO queue

Search Tiers



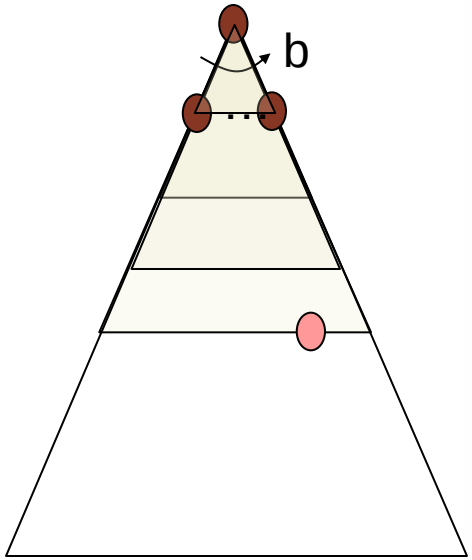
Breadth-First Search (BFS) Properties

- 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 fringe take?
  - Has roughly the last tier, so $O(b^s)$
- Is it complete?
  - s must be finite if a solution exists, so yes!
- Is it optimal?
  - Only if costs are all 1 (more on costs later)

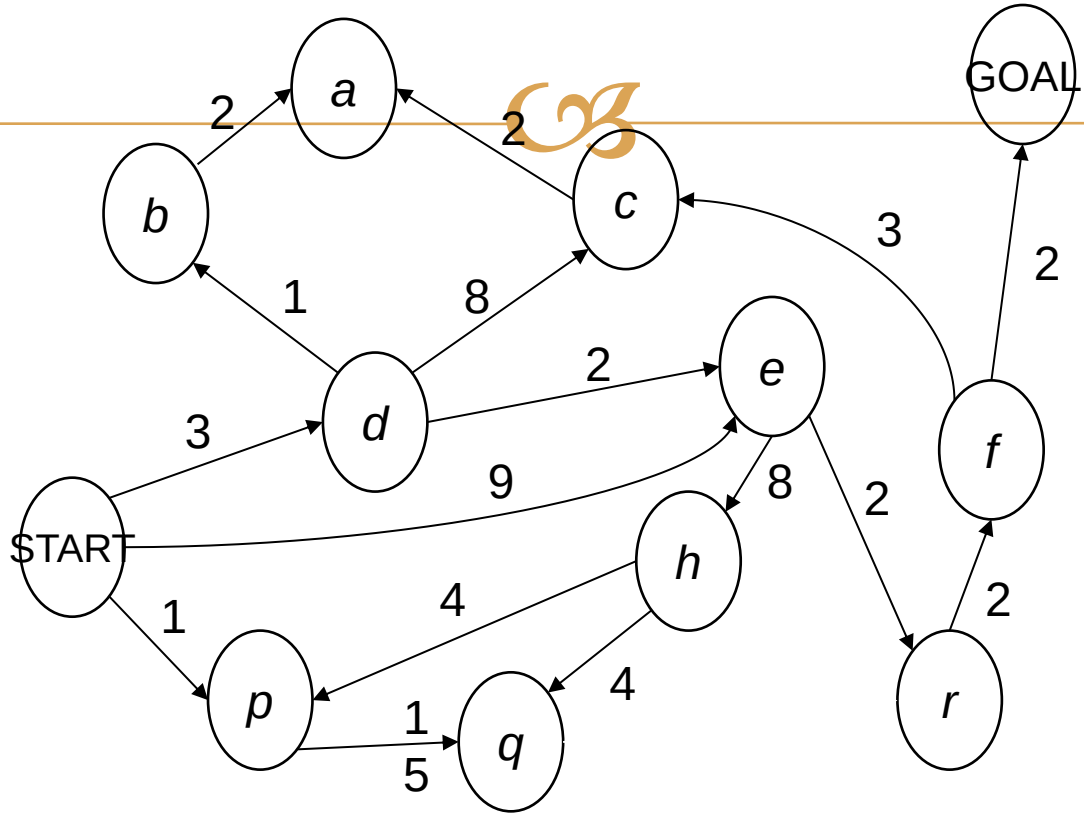


Iterative Deepening

- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..
- Isn't that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!

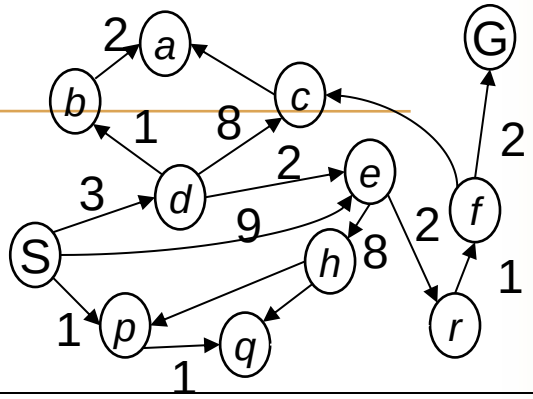


Cost-Sensitive Search



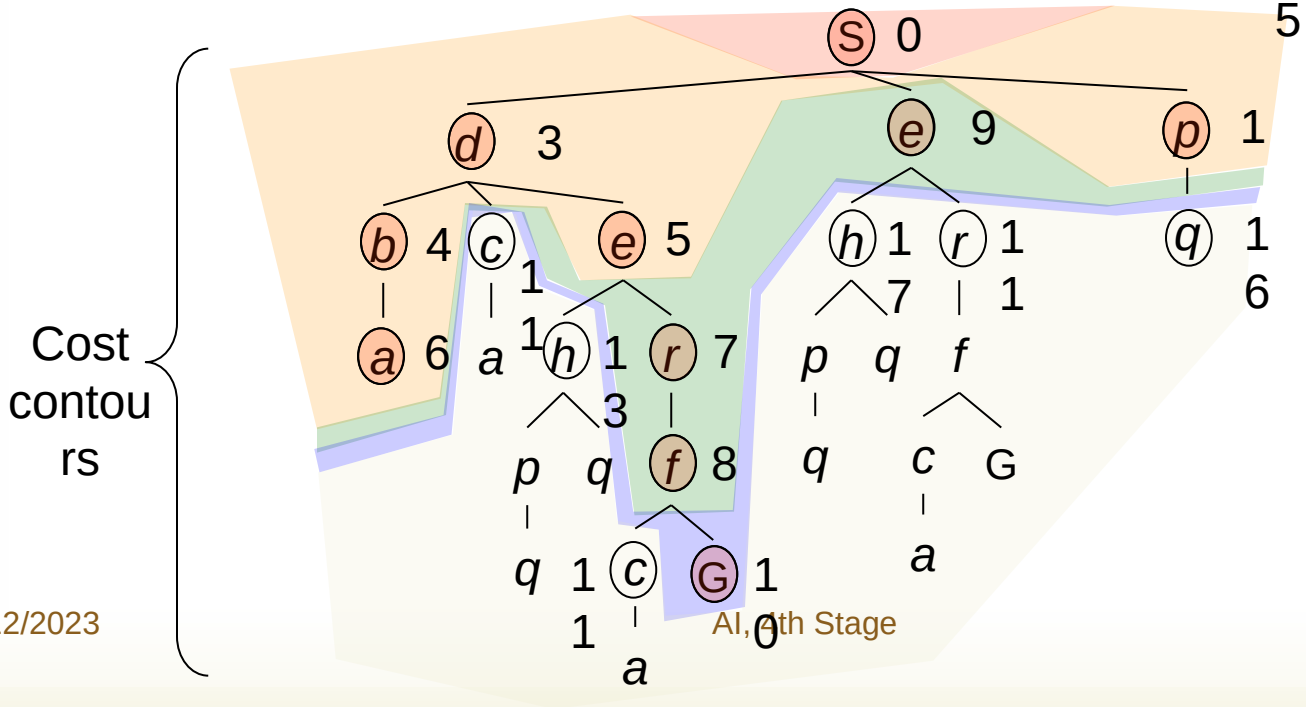
BFS finds the shortest path in terms of number of actions. It does not find the least-cost path. We will now cover a similar algorithm which does find the least-cost path.

Uniform Cost Search



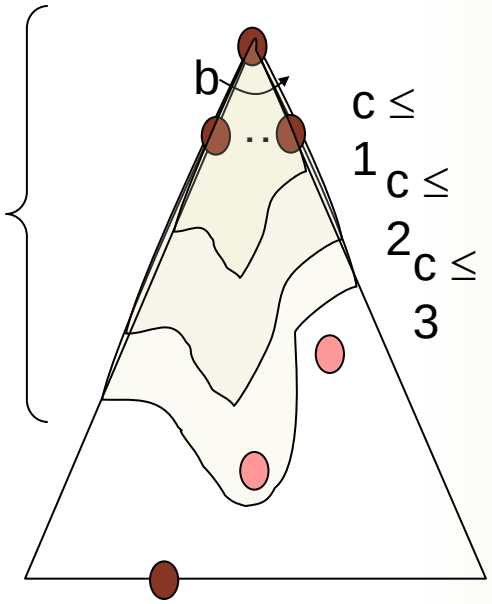
Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)



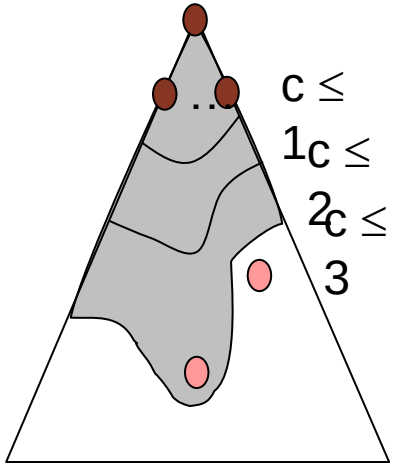
Uniform Cost Search (UCS) Properties

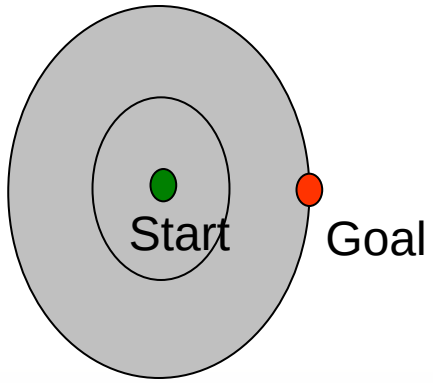
- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)
- How much space does the fringe take?
  - 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! (Proof next lecture via A*)



Uniform Cost Issues

- Remember: UCS explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every "direction"
  - No information about goal location
- We'll fix that soon!





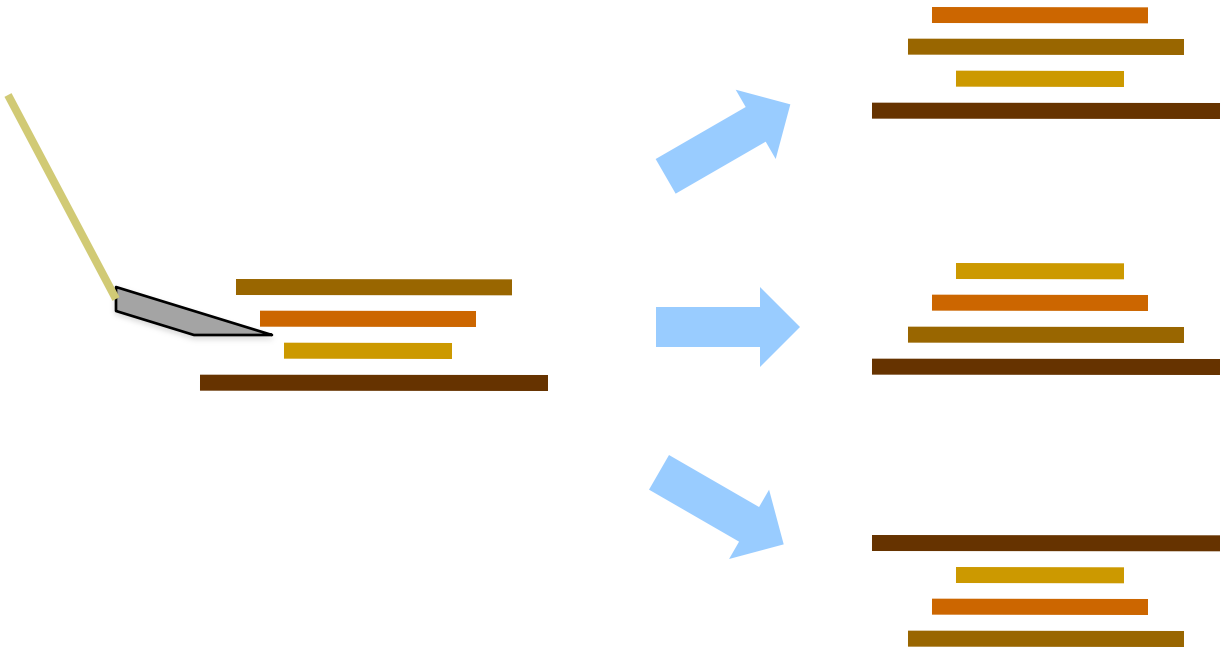
The One Queue

- All these search algorithms are the same except for fringe strategies
  - Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)
  - Practically, for DFS and BFS, you can avoid the log(n) overhead from an actual priority queue, by using stacks and queues
  - Can even code one implementation that takes a variable queuing object

Search and Models

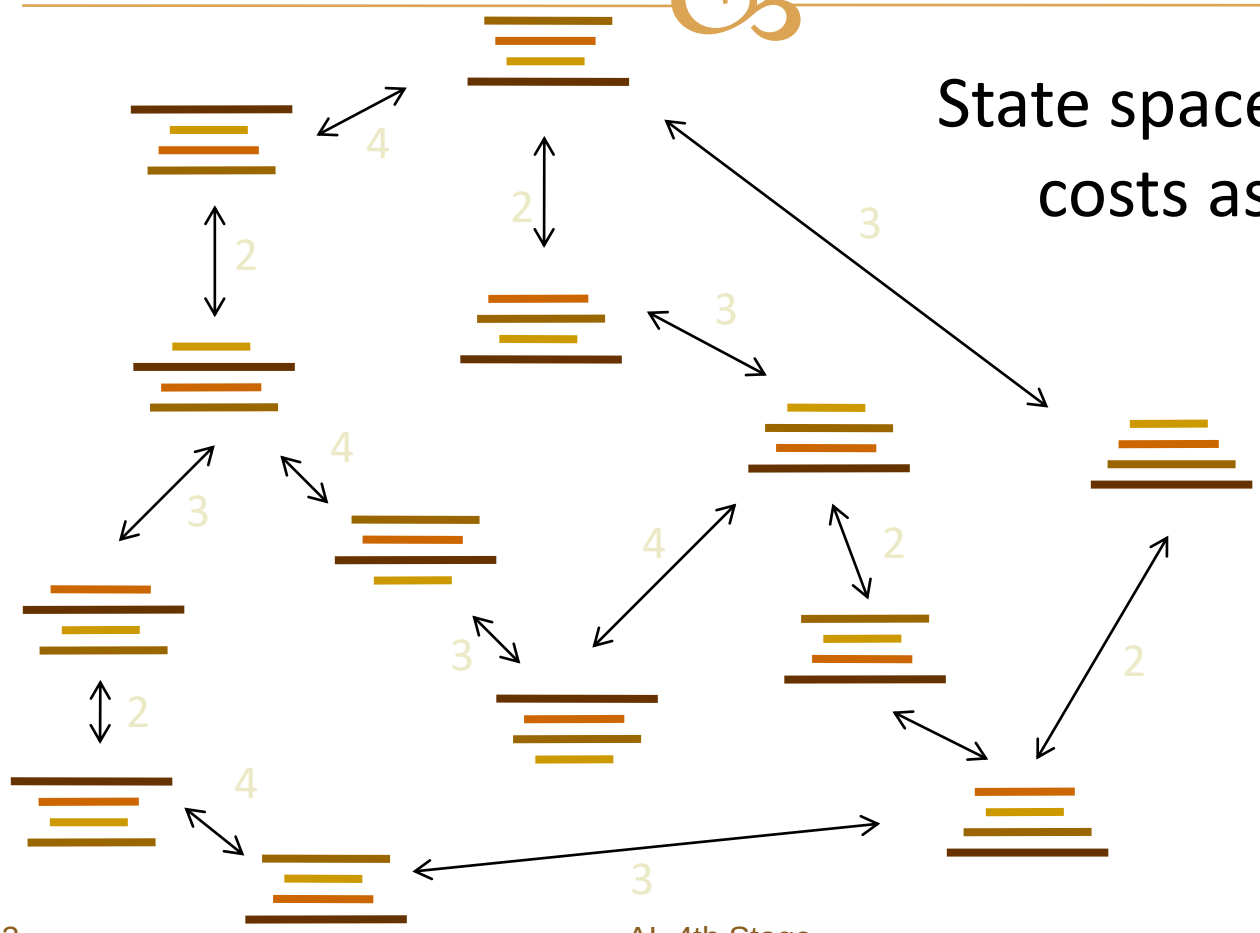
- Search operates over models of the world
  - The agent doesn't actually try all the plans out in the real world!
  - Planning is all "in simulation"
  - Your search is only as good as your models…

Example: Pancake Problem



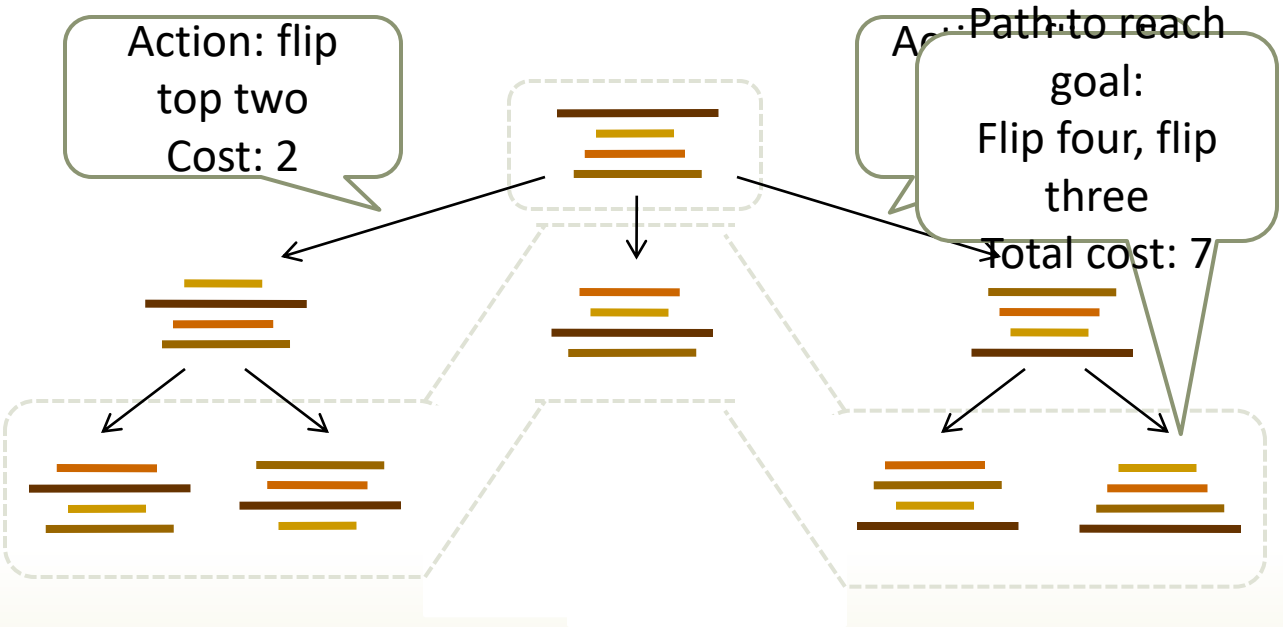
Cost: Number of pancakes flipped

Example: Pancake Problem



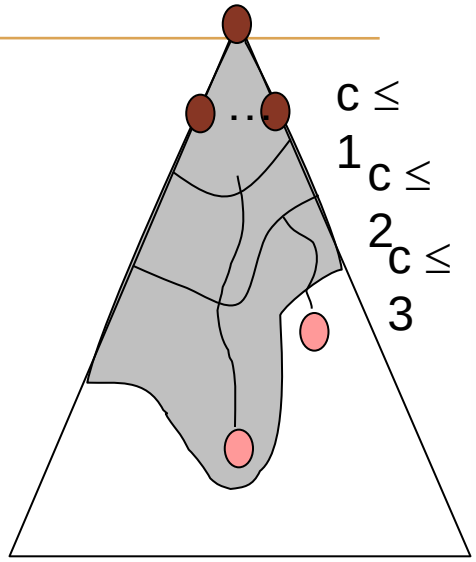
State space graph with costs as weights

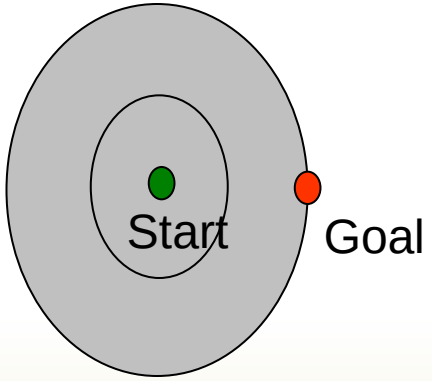
General Tree Search



Uniform Cost Search

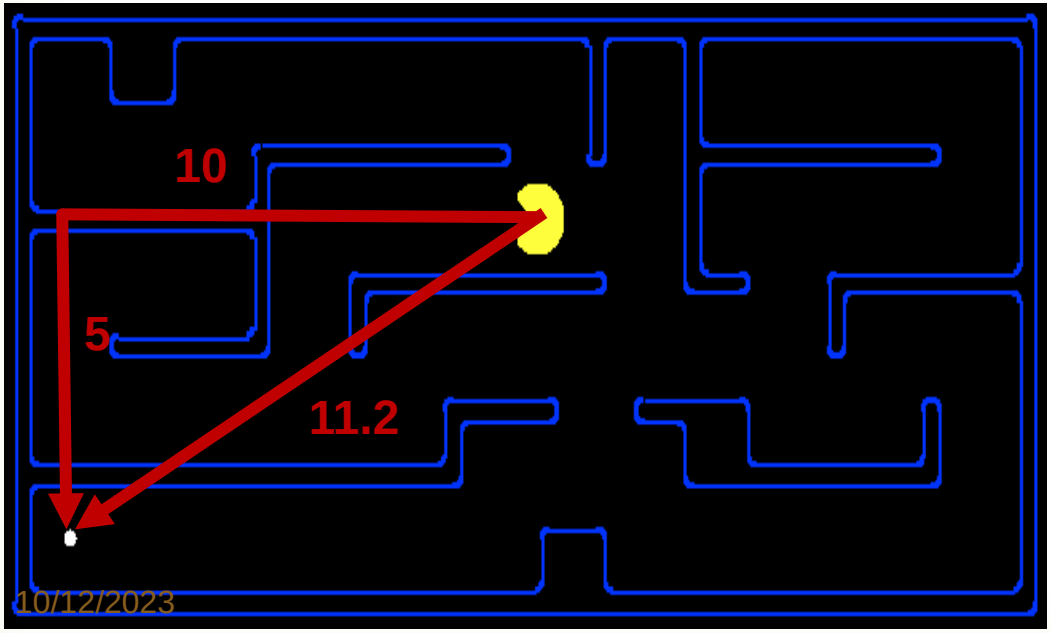
- Strategy: expand lowest path cost
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every "direction"
  - No information about goal location



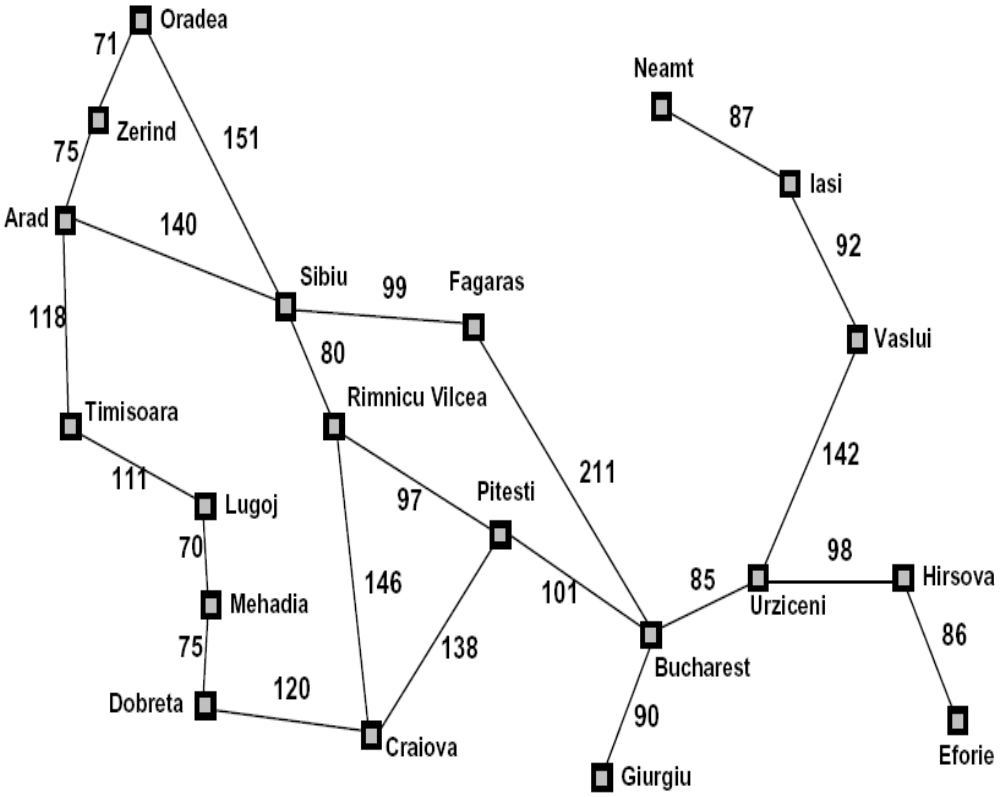


Search Heuristics

- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
- Examples: Manhattan distance, Euclidean distance for path



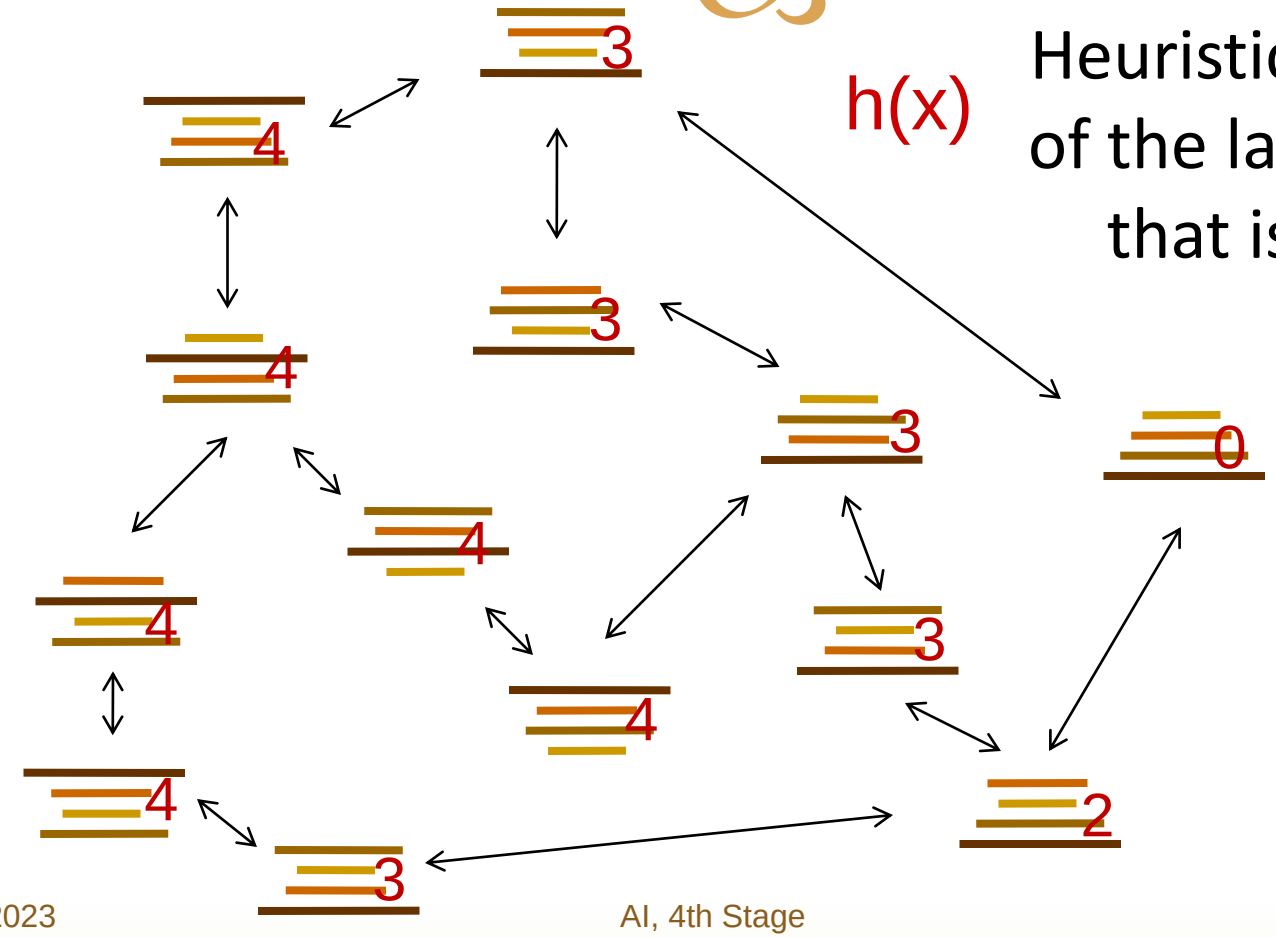
Example: Heuristic Function



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

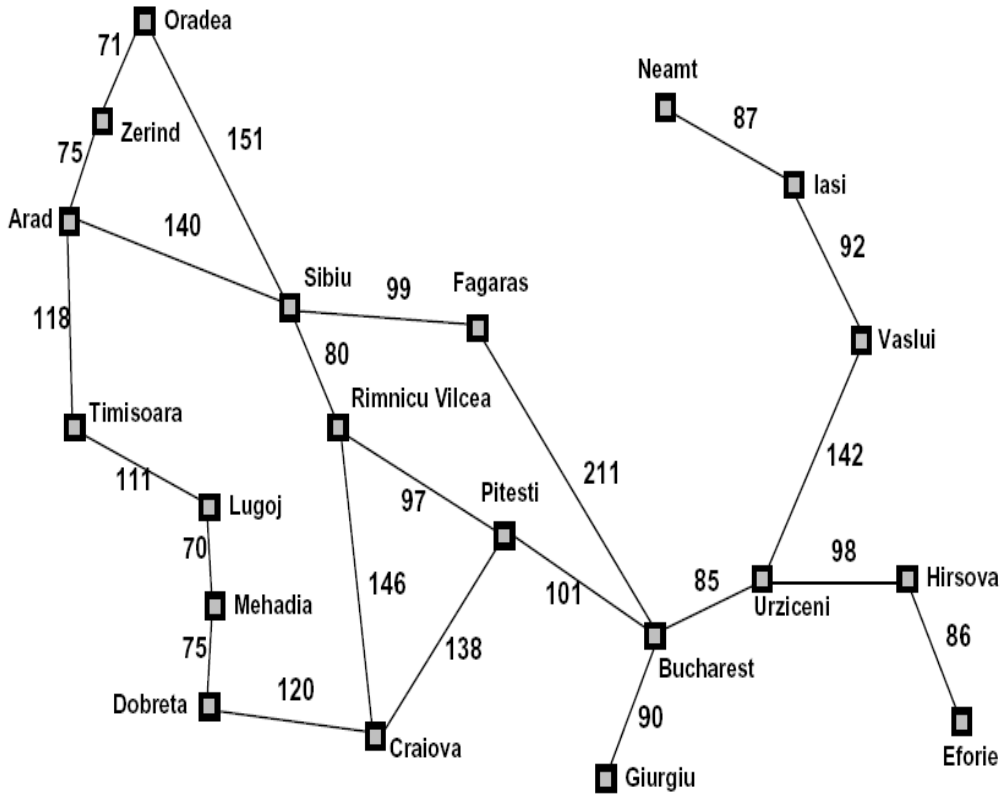
Example: Heuristic Function



Heuristic: the number of the largest pancake that is still out of place

Greedy Search

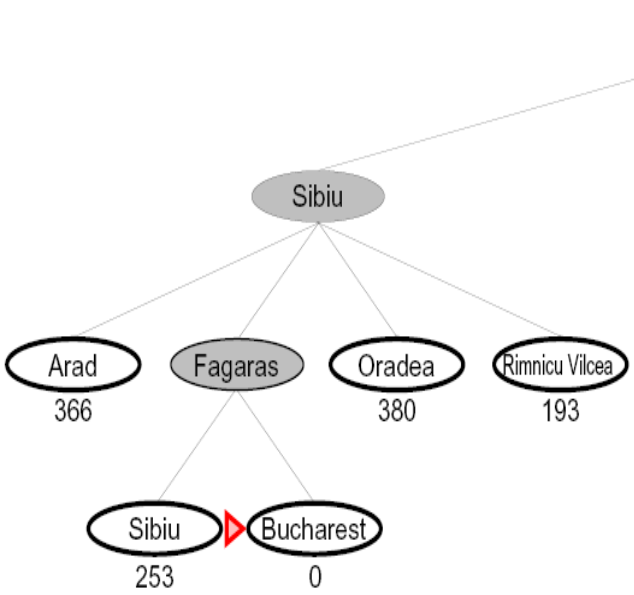
Example: Heuristic Function

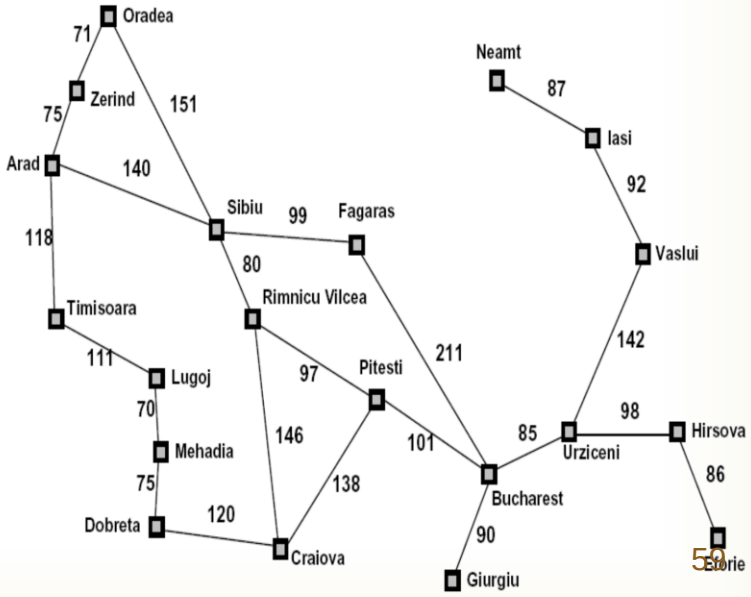


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

Greedy Search

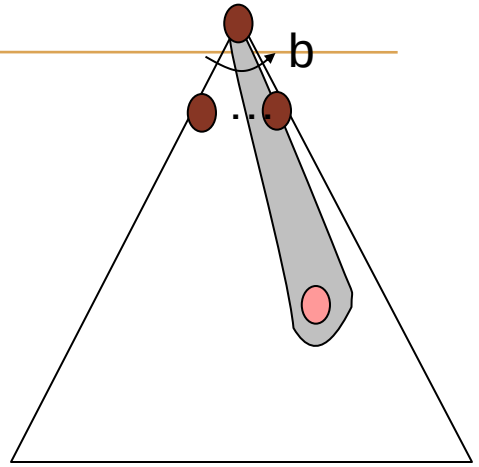


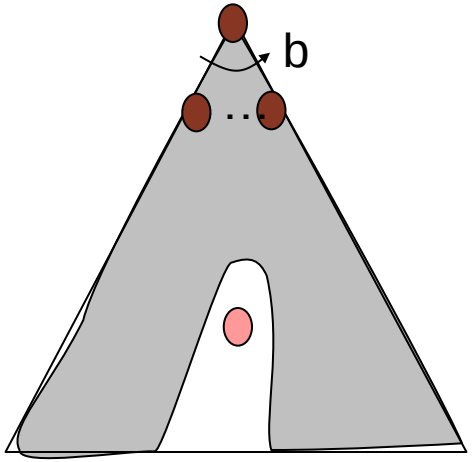


- What can go wrong?

Greedy Search

- Strategy: expand a node that you think is closest to a goal state
  - Heuristic: estimate of distance to nearest goal for each state
- A common case:
  - Best-first takes you straight to the (wrong) goal
- Worst-case: like a badly-guided DFS





End

- Any Question