CSE 473: Artificial Intelligence
Spring 2014

Hanna Hajishirzi

Problem Spaces and Search

slides from
Dan Klein, Stuart Russell, Andrew Moore, Dan Weld, Pieter Abbeel, Luke Zettelmoyer

Outline

- Agents that Plan Ahead
- Search Problems
- Uninformed Search Methods (part review for some)
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search
- Heuristic Search Methods (new for all)
  - Best First / Greedy Search

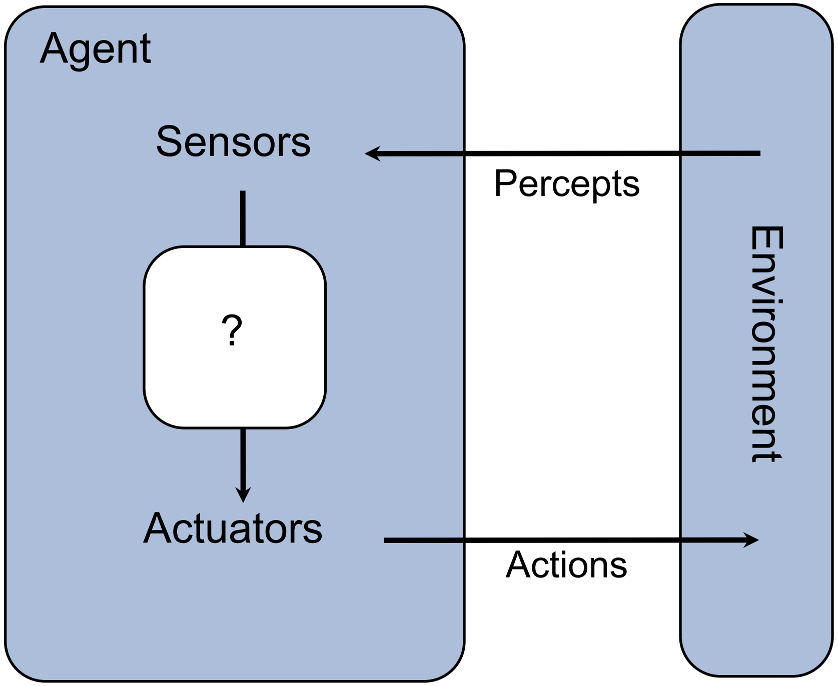
Review: Agents

An agent:

- Perceives and acts
- Selects actions that maximize its utility function
- Has a goal

Environment:

- Input and output to the agent

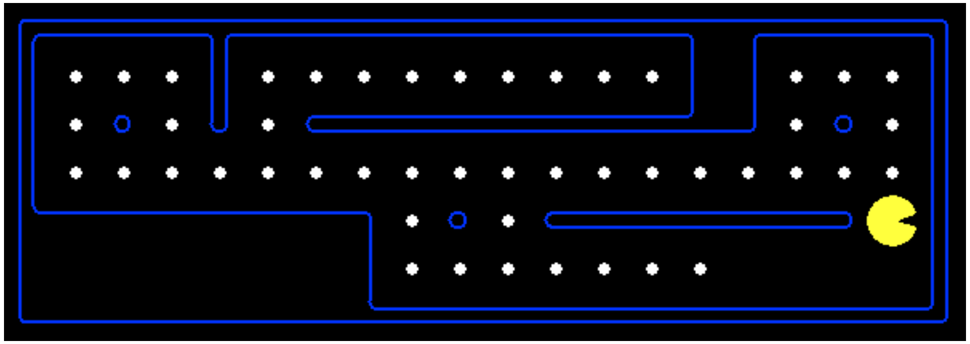


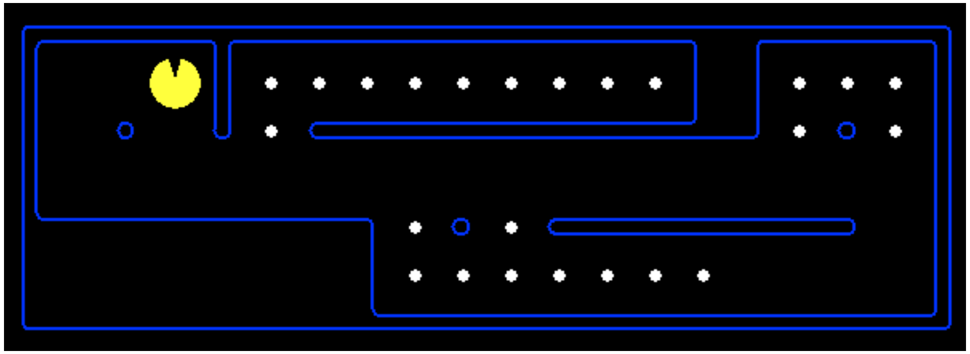
Search -- the environment is:

fully observable, single agent, deterministic, static, discrete

Reflex Agents

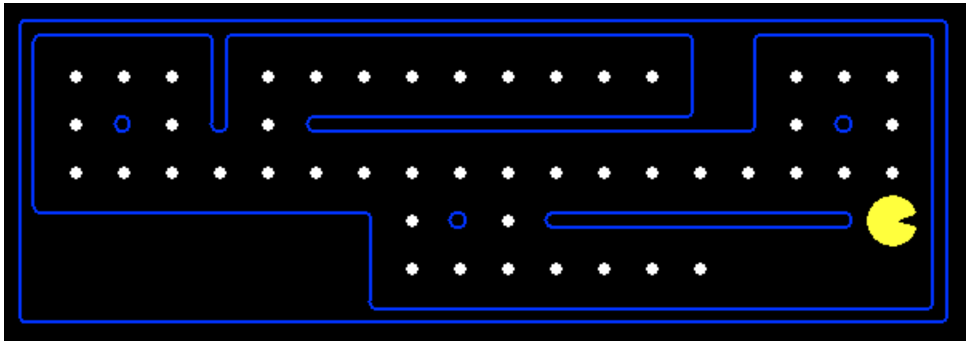
- Reflex agents:
  - Choose action based on current percept (and maybe memory)
  - Do not consider the future consequences of their actions
  - Act on how the world IS
- Can a reflex agent achieve goals?

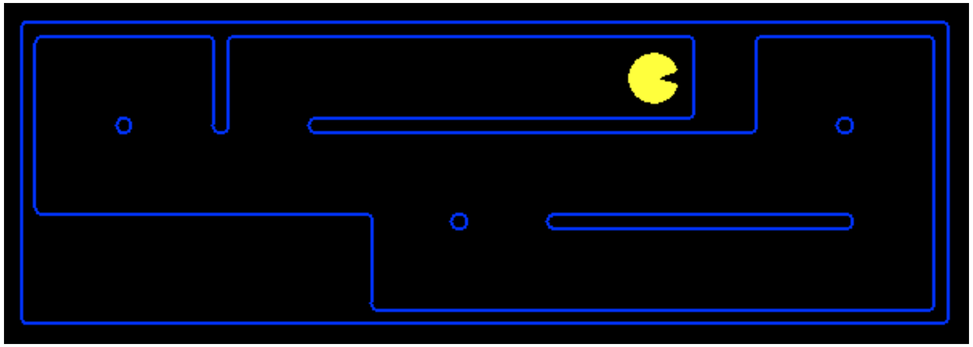




Goal Based Agents

- Goal-based agents:
  - Plan ahead
  - Ask “what if”
  - Decisions based on (hypothesized) consequences of actions
  - Must have a model of how the world evolves in response to actions
  - Act on how the world WOULD BE





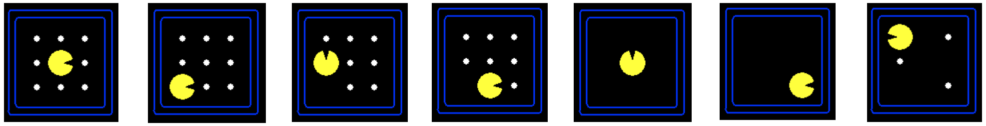
Search thru a Problem Space / State Space

- Input:
  - Set of states
  - Successor Function [and costs - default to 1.0]
  - Start state
  - Goal state [test]

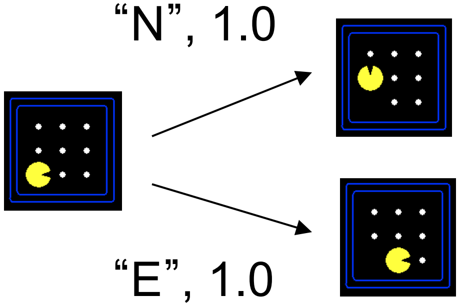
- Output:
  - Path: start $\Rightarrow$ a state satisfying goal test
  - [May require shortest path]
  - [Sometimes just need state passing test]

Example: Simplified Pac-Man

- Input:
  - A state space



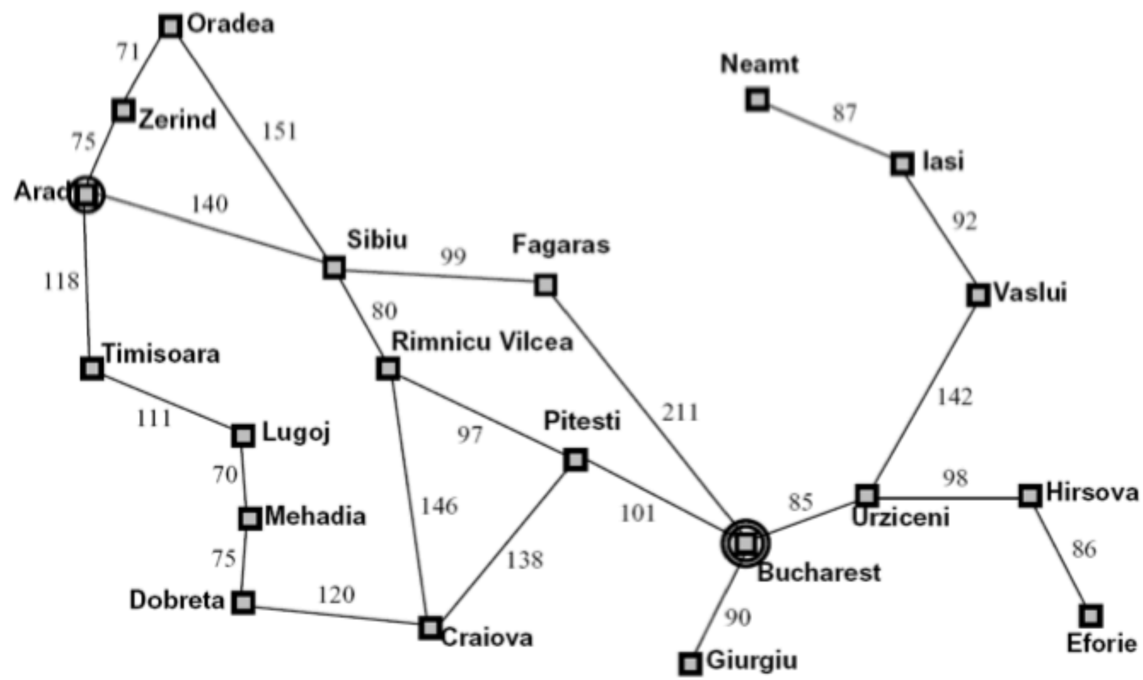
  - A successor function



  - A start state
  - A goal test
- Output:

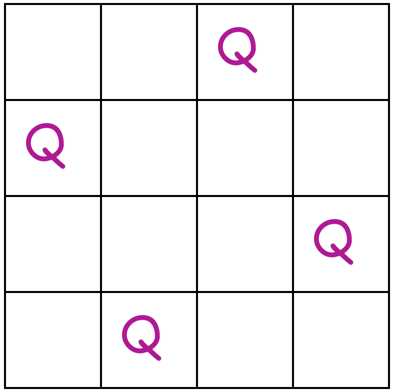
Ex: Route Planning: Romania → Bucharest

- Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state (test)
- Output:



Example: N Queens

- Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state (test)
- Output



Introducing MATHEMATICA 5
Παρουσιάζουμε το
Featuring a new generation of advanced algorithms with unparalleled speed, scope, and scalability

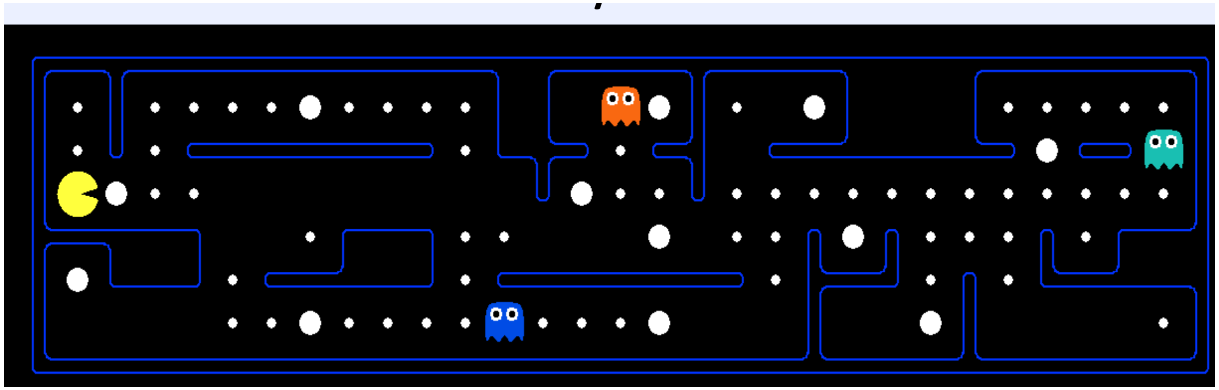
Algebraic Simplification

$$
\begin{aligned}
\partial_r^2 u &= -\left[E' - \frac{l(l+1)}{r^2} - r^2\right] u(r) \\
e^{-2s}\left(\partial_s^2 - \partial_s\right) u(s) &= -\left[E' - l(l+1)e^{-2s} - e^{2s}\right] u(s) \\
e^{-2s}\left[e^{\frac{1}{2}s}\left(e^{-\frac{1}{2}s}u(s)\right)'' - \frac{1}{4}u\right] &= -\left[E' - l(l+1)e^{-2s} - e^{2s}\right] u(s) \\
e^{-2s}\left[e^{\frac{1}{2}s}\left(e^{-\frac{1}{2}s}u(s)\right)''\right] &= -\left[E' - \left(l+\frac{1}{2}\right)^2 e^{-2s} - e^{2s}\right] u(s) \\
v'' &= -e^{2s}\left[E' - \left(l+\frac{1}{2}\right)^2 e^{-2s} - e^{2s}\right] v
\end{aligned}
$$

- Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state (test)
- Output:

What is in State Space?

- A world state includes every details of the environment



- A search state includes only details needed for planning

Problem: Pathing

States: {x,y} locations

Actions: NSEW moves

Successor: update location

Goal: is (x,y) End?

Problem: Eat-all-dots

States: {(x,y), dot booleans}

Actions: NSEW moves

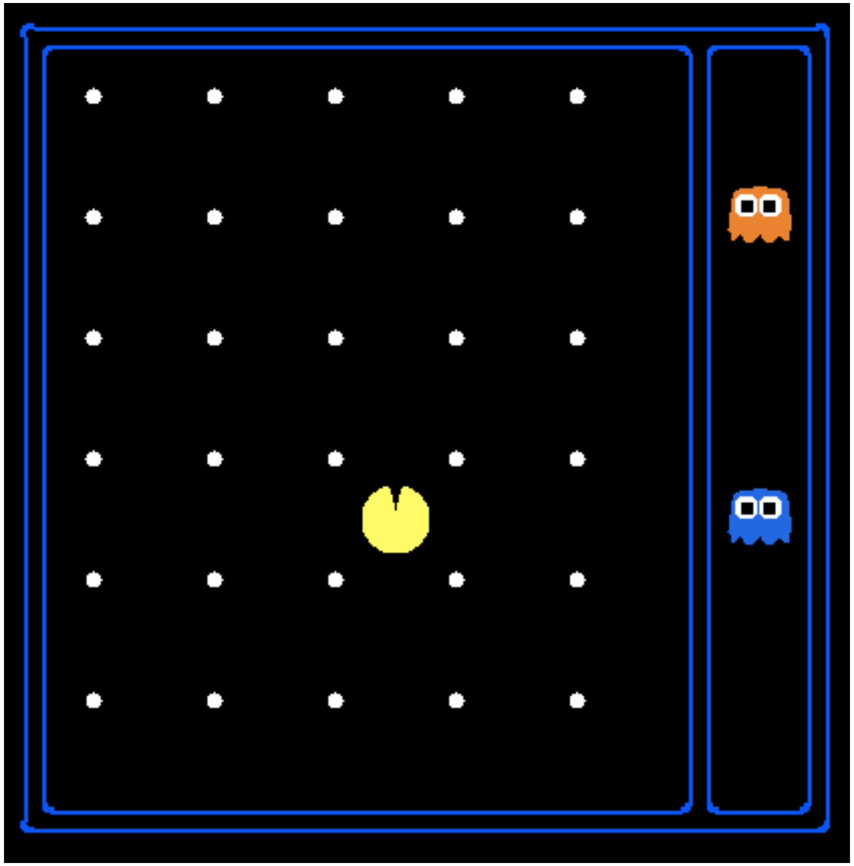
Successor: update location and dot boolean

Goal: dots all false?

State Space Sizes?

- World states:

- Pacman positions:
  10 x 12 = 120
- Pacman facing:
  up, down, left, right
- Food Count: 30
- Ghost positions: 12



State Space Sizes?

- How many?
- World State:

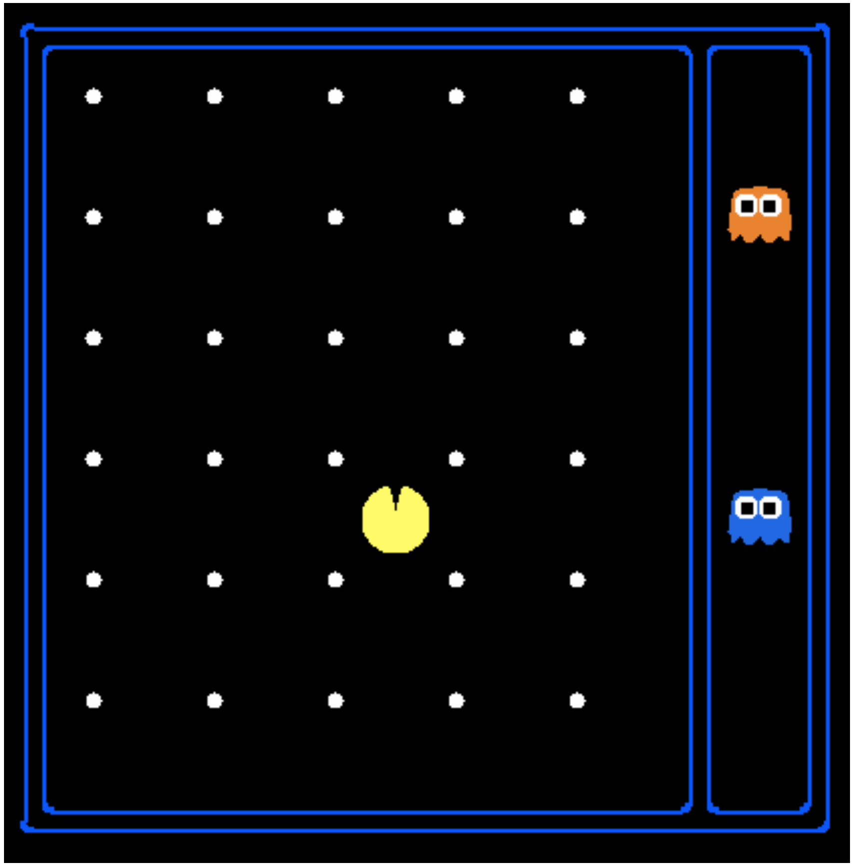
  $120*(2^{30})*(12^{2})*4$

- States for Pathing:

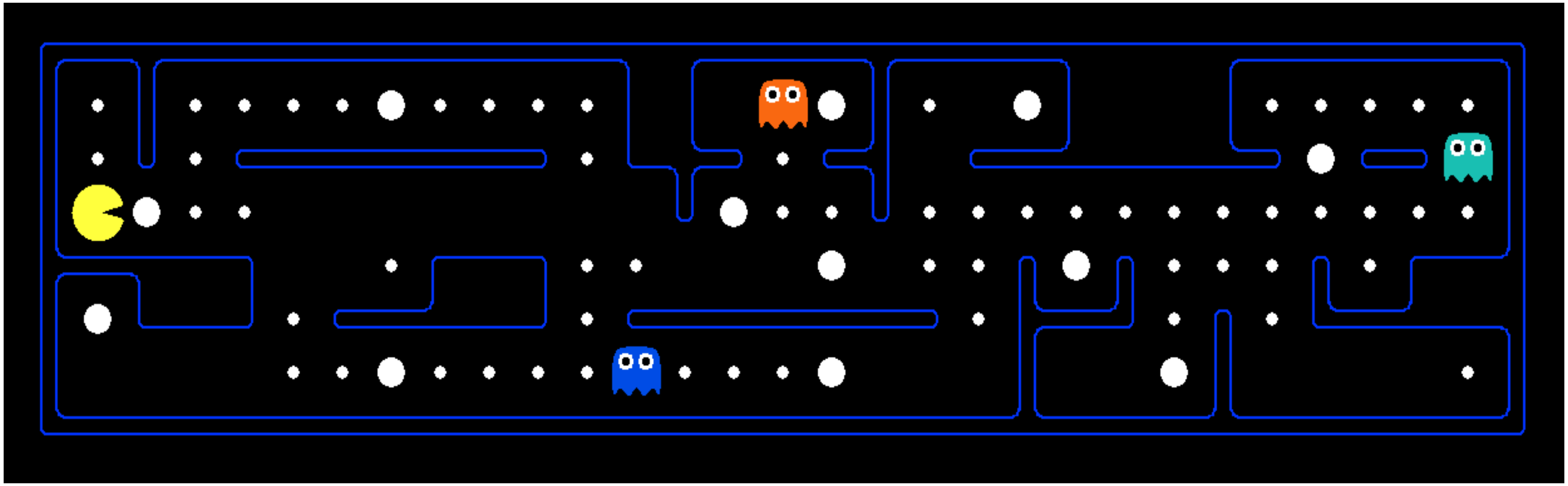
  120

- States for eat-all-dots:

  $120*(2^{30})$



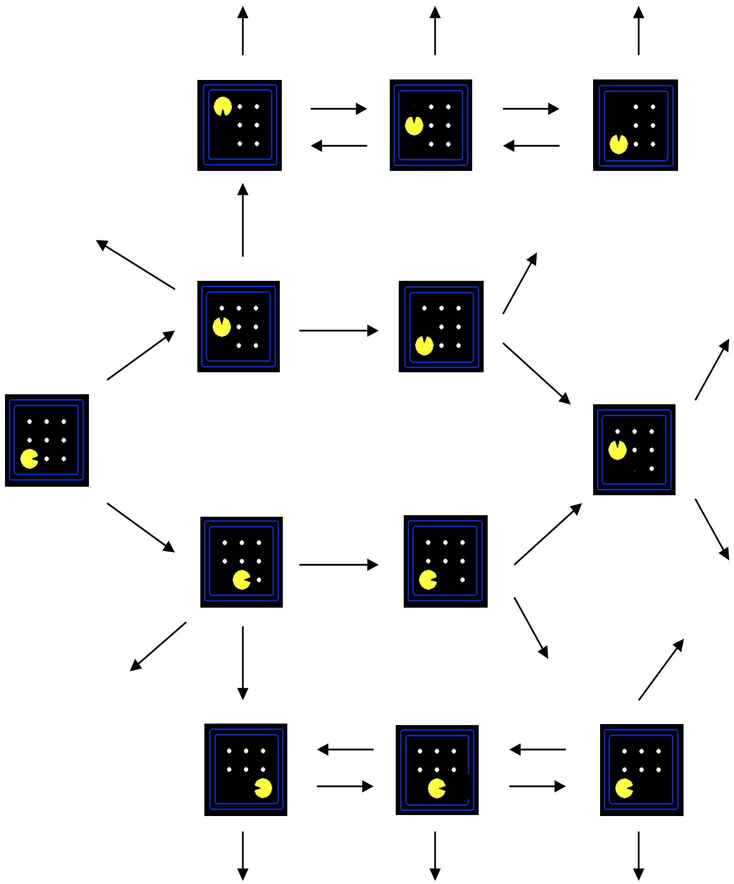
Quiz: Safe Passage



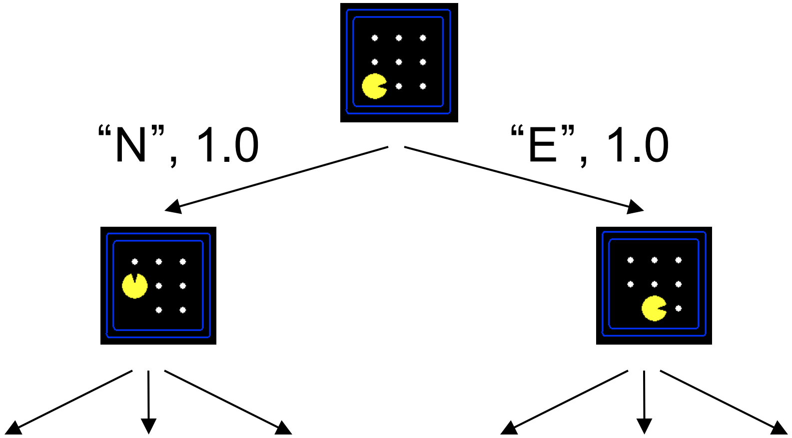
- Problem: eat all dots while keeping the ghosts perma-scared
- What does the state space have to specify?

State Space Graphs

- State space graph:
  - Each node is a state
  - The successor function is represented by arcs
  - Edges may be labeled with costs
- We can rarely build this graph in memory (so we don't)



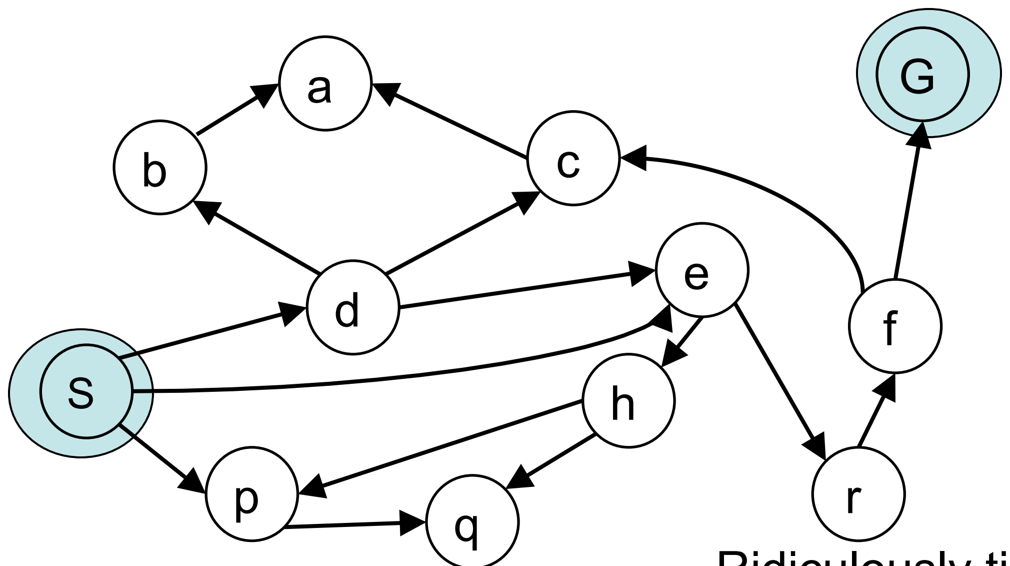
Search Trees



- A search tree:
  - Start state at the root node
  - Children correspond to successors
  - Nodes contain states, correspond to PLANS to those states
  - Edges are labeled with actions and costs
  - For most problems, we can never actually build the whole tree

Example: Tree Search

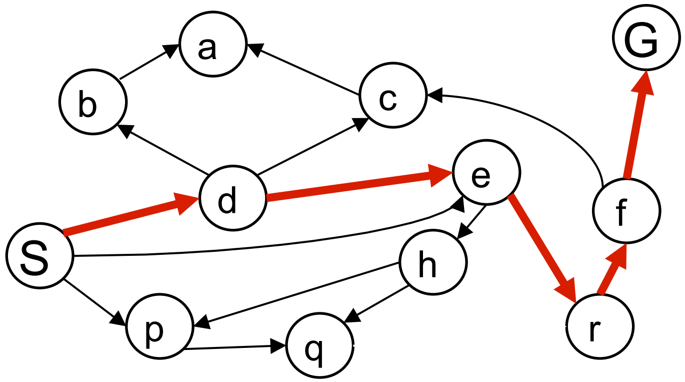
State Graph:



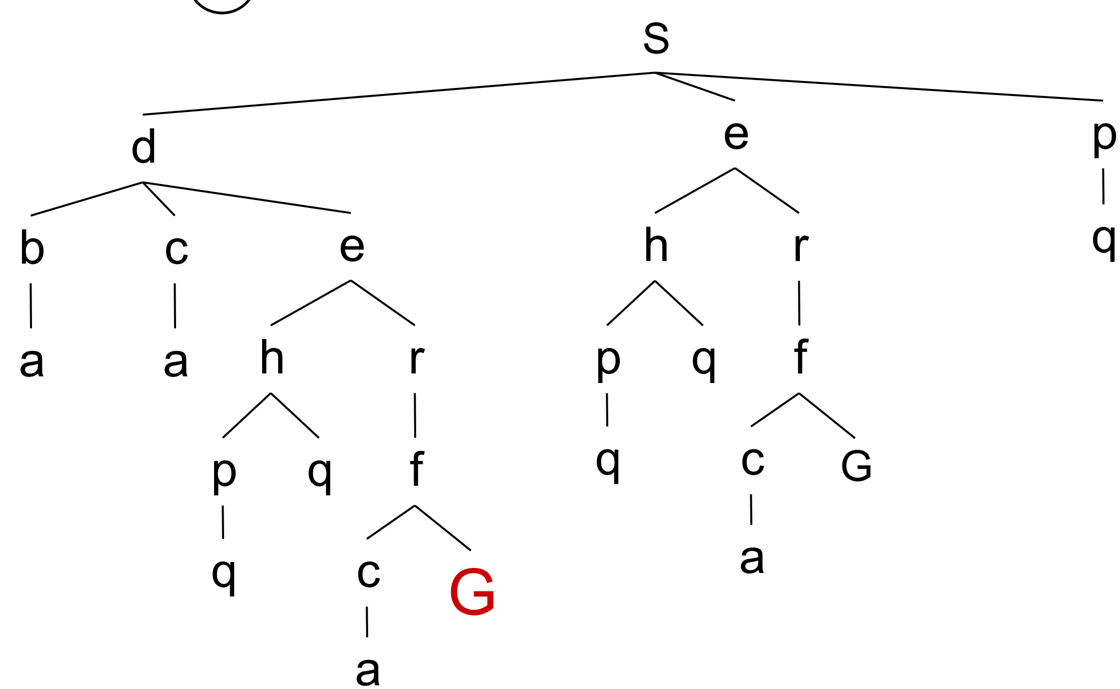
Ridiculously tiny search graph
for a tiny search problem

What is the search tree?

State Graphs vs. Search Trees



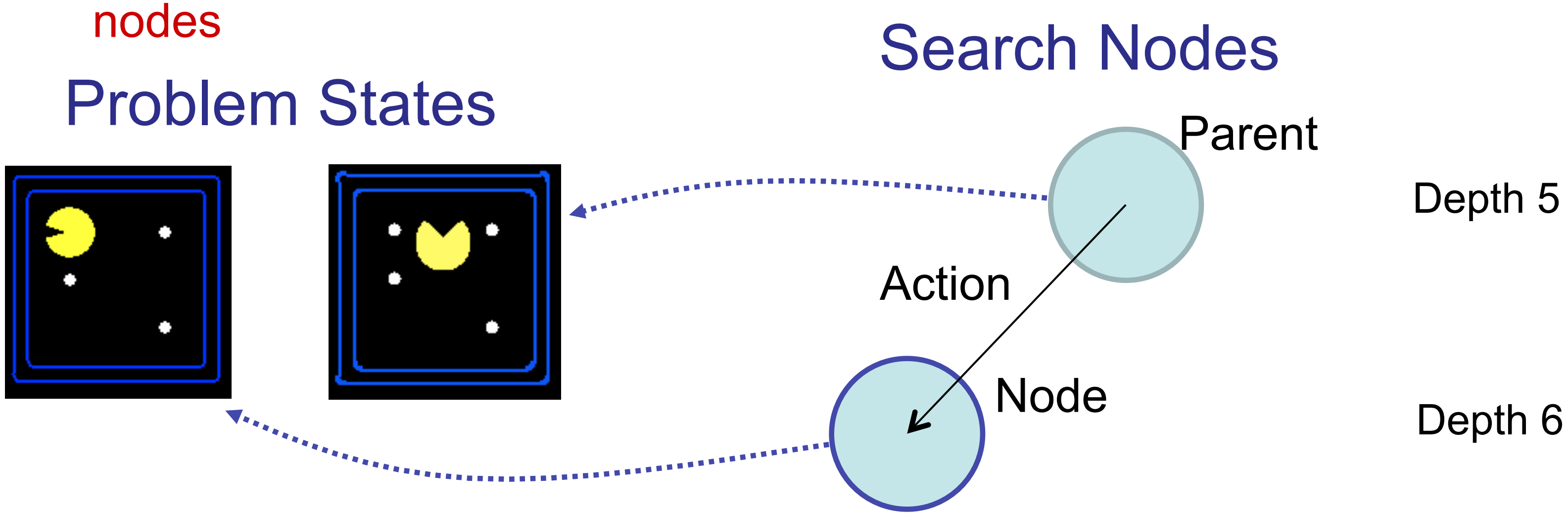
Each NODE in in the search tree is an entire PATH in the problem graph.



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

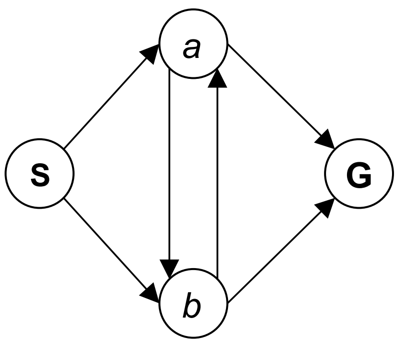
States vs. Nodes

- Nodes in state space graphs are problem states
  - Represent an abstracted state of the world
  - Have successors, can be goal / non-goal, have multiple predecessors
- Nodes in search trees are plans
  - Represent a plan (sequence of actions) which results in the node's state
  - Have a problem state and one parent, a path length, a depth & a cost
  - The same problem state may be achieved by multiple search tree nodes



Quiz: State Graphs vs. Search Trees

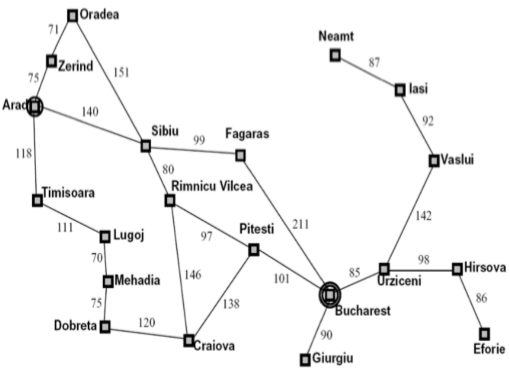
Consider this 4-state graph:



How big is its search tree (from S)?

Important: Lots of repeated structure in the search tree!

Building Search Trees



- Search:
  - Expand out possible plans
  - Maintain a fringe of unexpanded plans
  - Try to expand as few tree nodes as possible

General Tree Search

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy

Detailed pseudocode is in the book!

- Main question: which fringe nodes to explore?