Part 1:
Solving Problems with Search

Lecture plan

- Python for AI
- Project 0: questions, comments?
- Agents that Plan
- Planning as Search
- Specific search algorithms
  - Some review, some new ones

Why Python for AI?

- Object oriented, but without the annoyances of Java and C++
- Easy and fun to write code
  https://www.learnpython.org/

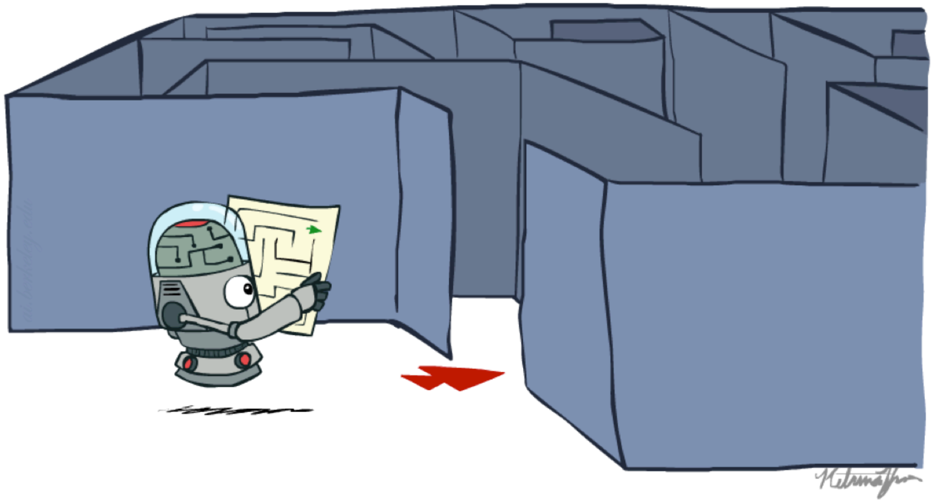
- Community and textbook support
  https://wiki.python.org/moin/PythonForArtificialIntelligence
  http://aima.cs.berkeley.edu/python/readme.html

- Advanced language features adapted from from LISP, Prolog:
  - functions as first-class citizens
  - List processing / lambda operators (from LISP)
    http://www.u.arizona.edu/~erdmann/mse350/topics/list_comprehensions.html

Project 0: Python 101. Due Friday 1/20

- Goal: gentle introduction to Python language, common data structures
- Algorithm:
    1. Download files
    2. Follow tutorial (step-by-step)
    3. Read provided starting code
    4. Fill in function bodies
    5. Run autograder
    6. If all tests passed – submit. Else: goto 3.
- Questions, stuck? Piazza!
- **Python clinic:** Wednesday 1/18, Lab. (times TBD).

Search

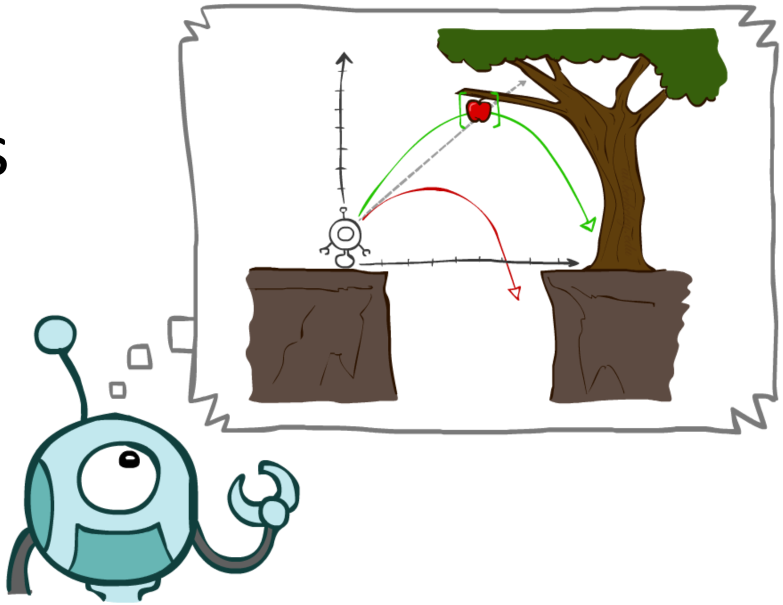


[Slides adapted from Dan Klein and Pieter Abbeel]

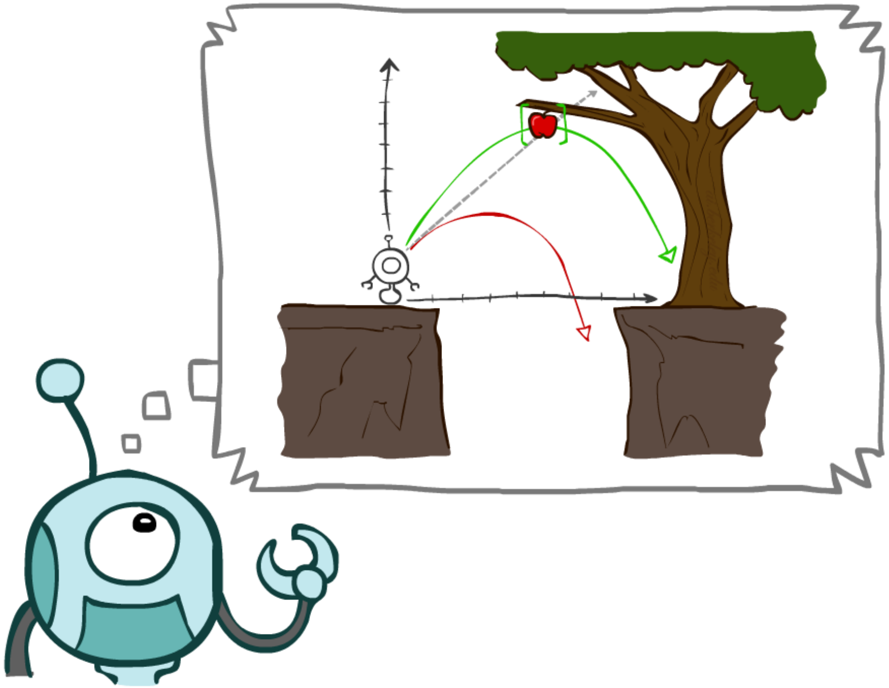
http://ai.berkeley.edu.]

Today

- Agents that Plan Ahead
- Search Problems
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Iterative Deepening



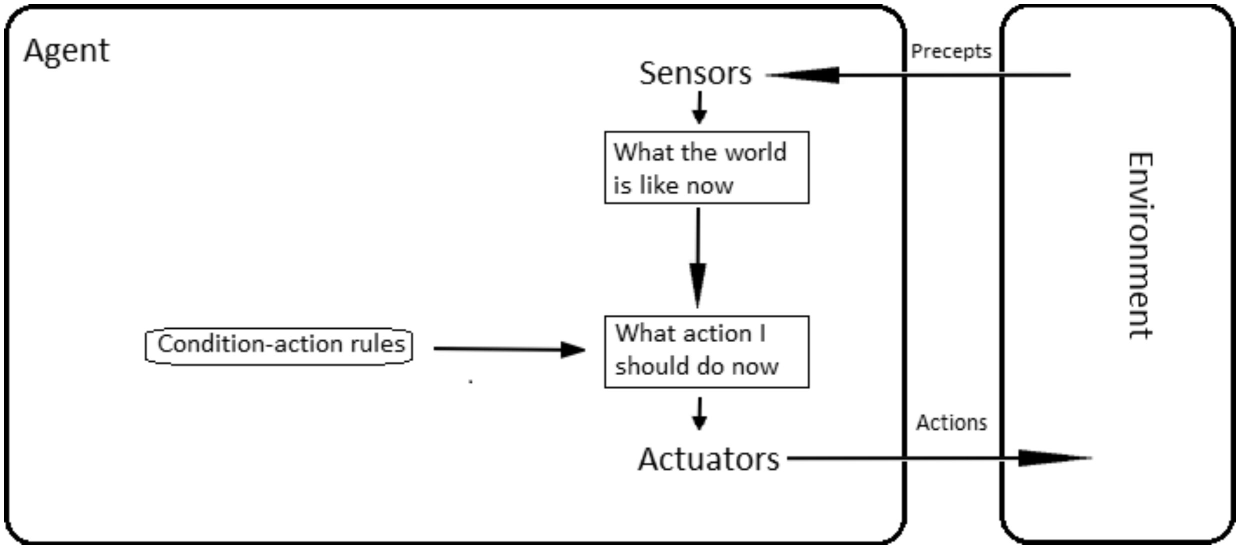
Agents that Plan



Agent types

- Five basic types in order of increasing generality:
  - Table Driven agents
  - Simple reflex agents
  - Model-based reflex agents
  - Goal-based agents
    - Problem-solving agents
  - Utility-based agents
    - Can distinguish between different goals
  - Learning agents

Reflex Agents



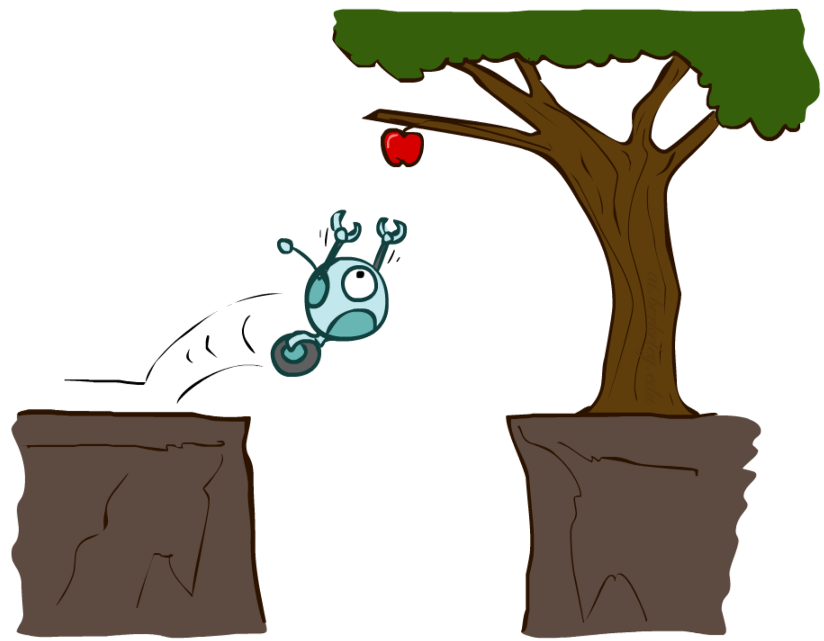
- Act only on the basis of the current percept.
- Based on the *condition-action rules*:

  if condition

  then action.
- Infinite loops are often unavoidable (can be fixed with some randomization)

Reflex Agents

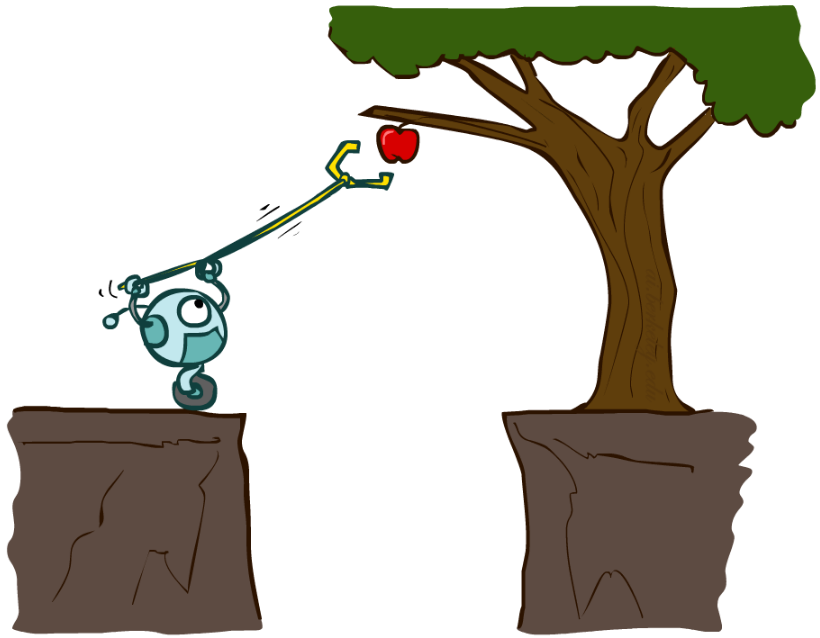
- Reflex agents:
  - Choose action based on current percept (and maybe memory)
  - May have memory or a model of the world's current state
  - Do not consider the future consequences of their actions
  - Consider how the world IS
- Can a reflex agent be rational?



Demo of reflex agent (optimal)
Demo of reflex agent (stuck)

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

Rational agents

- Performance measure: An objective criterion for success of an agent's behavior, e.g.,
  - Robot driver?
  - Chess-playing program?
  - Spam email classifier?

- Rational Agent: selects actions that is *expected* to **maximize its performance measure,**
  - given percept sequence
  - given agent's built-in knowledge
  - consider: how to maximize expected <u>future</u> performance, with only <u>historical</u> data?

Task Environment

- Before we design an intelligent agent, we must specify its “task environment”:

PEAS:

Performance measure

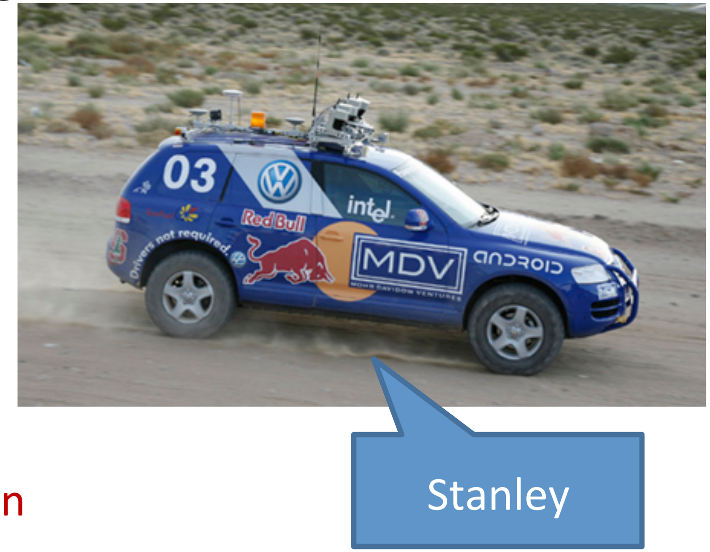
Environment

Actuators

Sensors

PEAS: Example

- Example: Agent = robot driver in DARPA Challenge



  - Performance measure:
    - Time to complete course
  - Environment:
    - Road, obstacles
  - Actuators:
    - Steering wheel, accelerator, brake, signal, horn
  - Sensors:
    - Optical cameras, lasers, sonar, accelerometer, speedometer, GPS, odometer, engine sensors, …

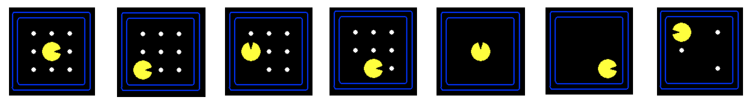
Goal-Based Agents: Search

- Five basic types in order of increasing generality:
  - Table Driven agents
  - Simple reflex agents
  - Model-based reflex agents
  - Goal-based agents
    - Problem-solving agents: solve problems by <u>searching</u> for solution
  - Utility-based agents
    - Can distinguish between different goals
  - Learning agents

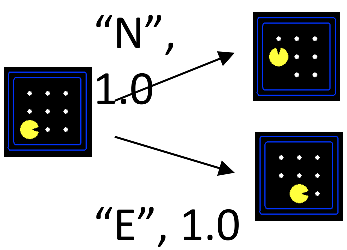
Search Problems

Search Problems

- A search problem consists of:
  - A state space



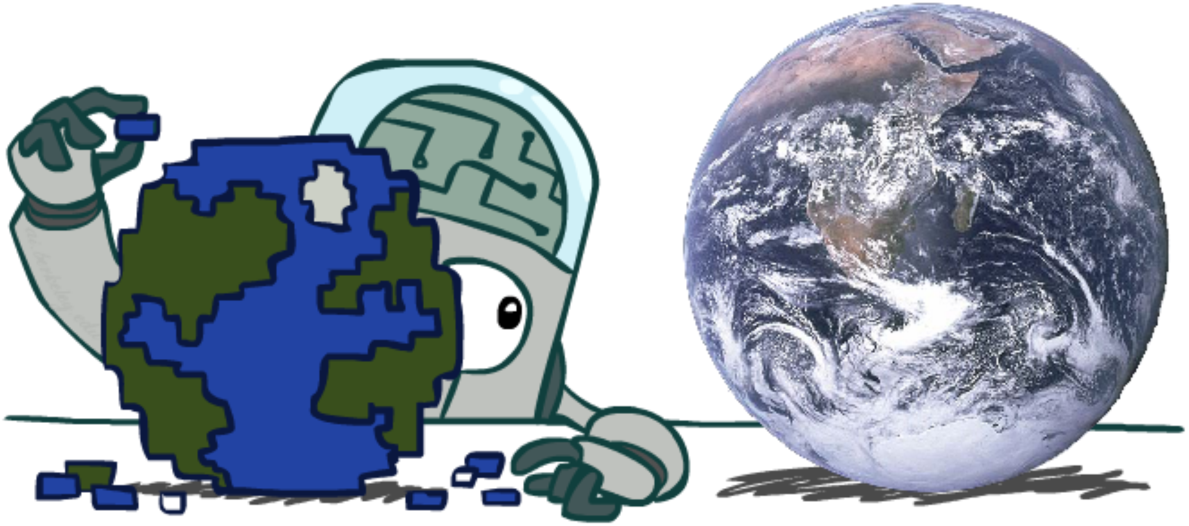
  - A successor function (with actions, costs)



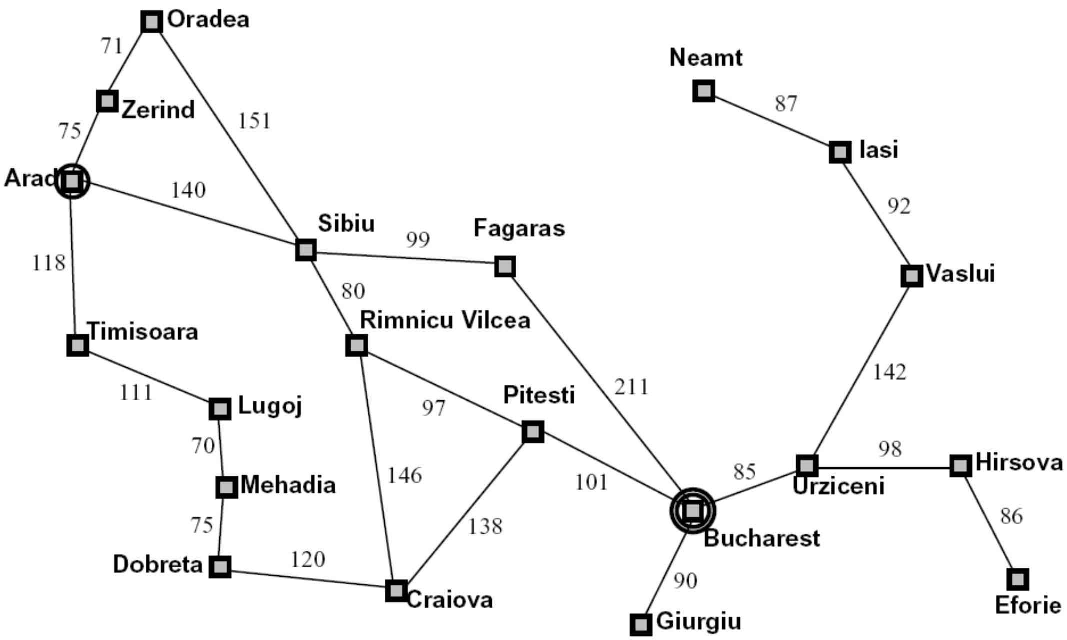
  - A start state and a goal test

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

Search Problems Are Models



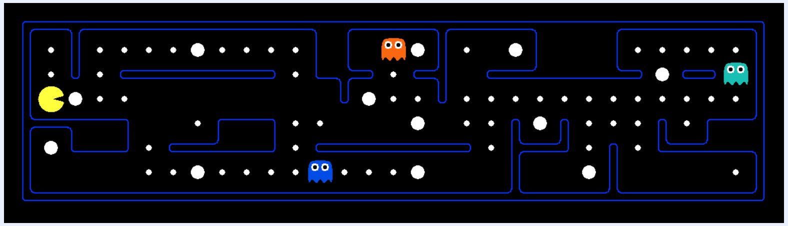
Example: Traveling in Romania



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

What's in a State Space?

The world state includes every last detail of the environment



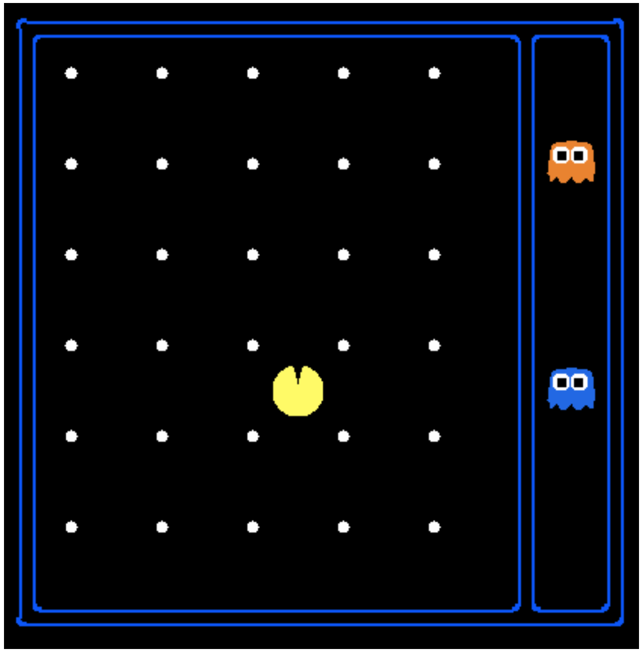
A search state keeps **only** the details needed for planning (abstraction).
Note: *Static info (e.g., walls) do not need to be stored for each state*

- Problem: Path-Finding
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location only
  - Goal test: is (x,y)=END

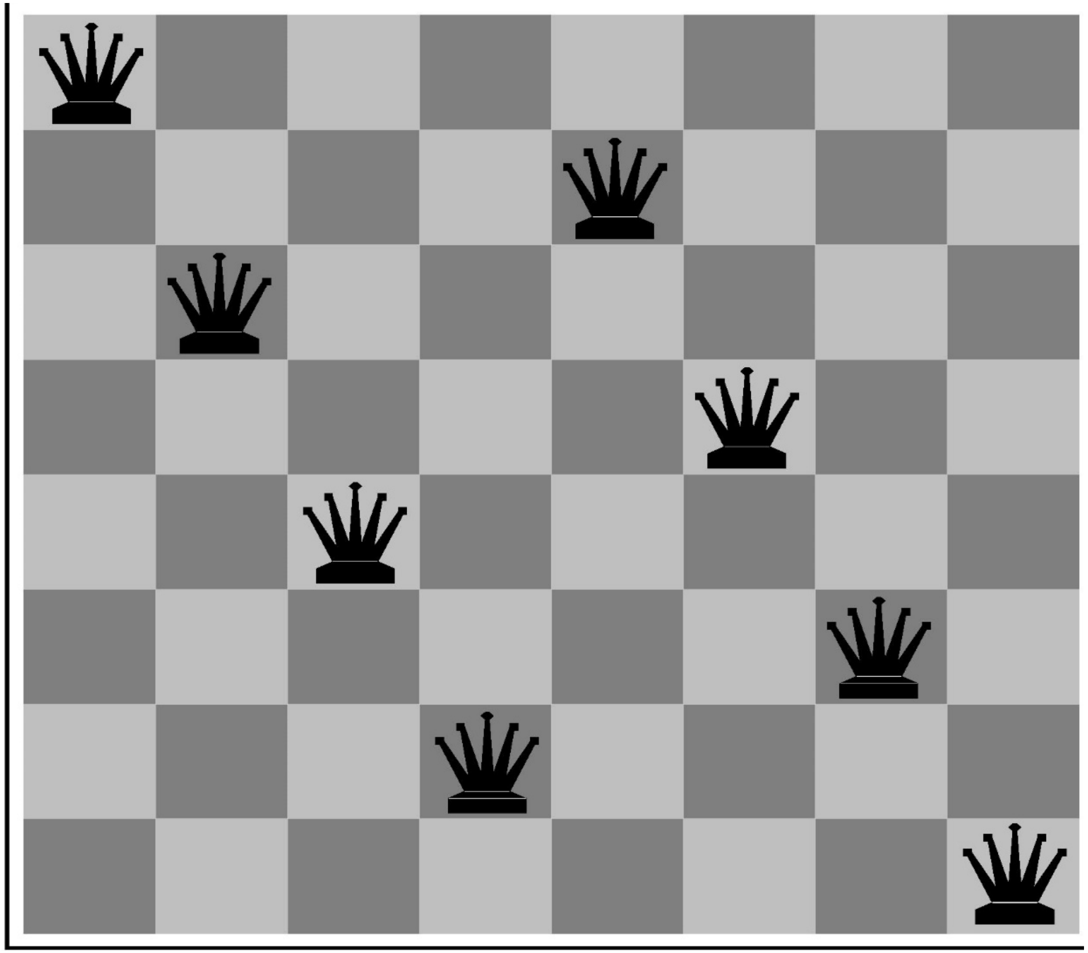
- Problem: Eat-All-Dots
  - States: {(x,y), dot booleans}
  - Actions: NSEW
  - Successor: update location and possibly a dot boolean
  - Goal test: dots all false

State Space Sizes?

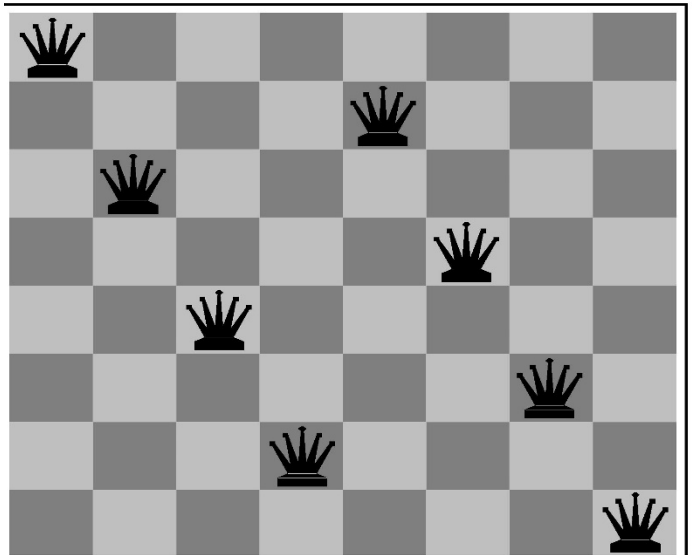
- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW
- How many
  - World states?
    $120 \times (2^{30}) \times (12^2) \times 4$
  - States for path-finding?
    120
  - States for eat-all-dots?
    $120 \times (2^{30})$



Example: 8-queens problem



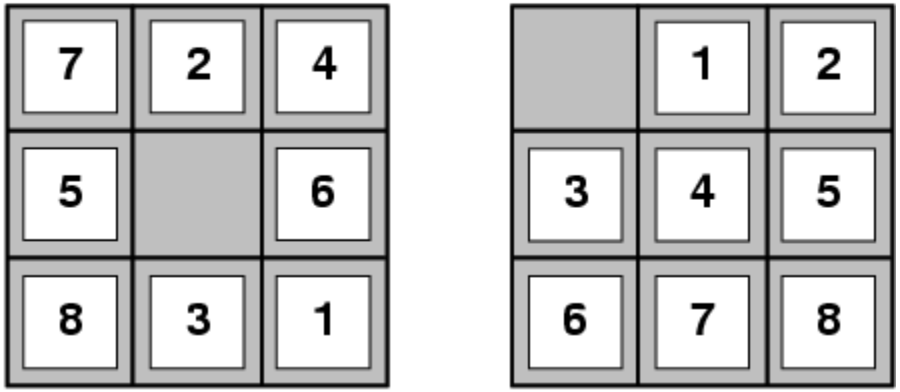
State-Space problem formulation



- states? -any arrangement of n<=8 queens
  -*or* arrangements of n<=8 queens in leftmost n columns, 1 per column, such that no queen attacks any other.
- initial state? no queens on the board
- actions? -add queen to any empty square
  -*or* add queen to leftmost empty square such that it is not attacked by other queens.
- goal test? 8 queens on the board, none attacked.
- path cost? 1 per move

Example: 8-puzzle

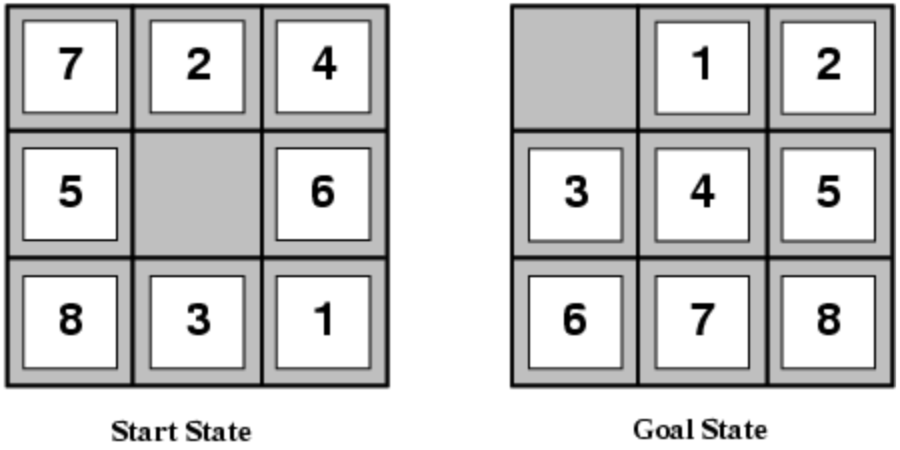
- states?
- initial state?
- actions?
- goal test?
- path cost?



Start State

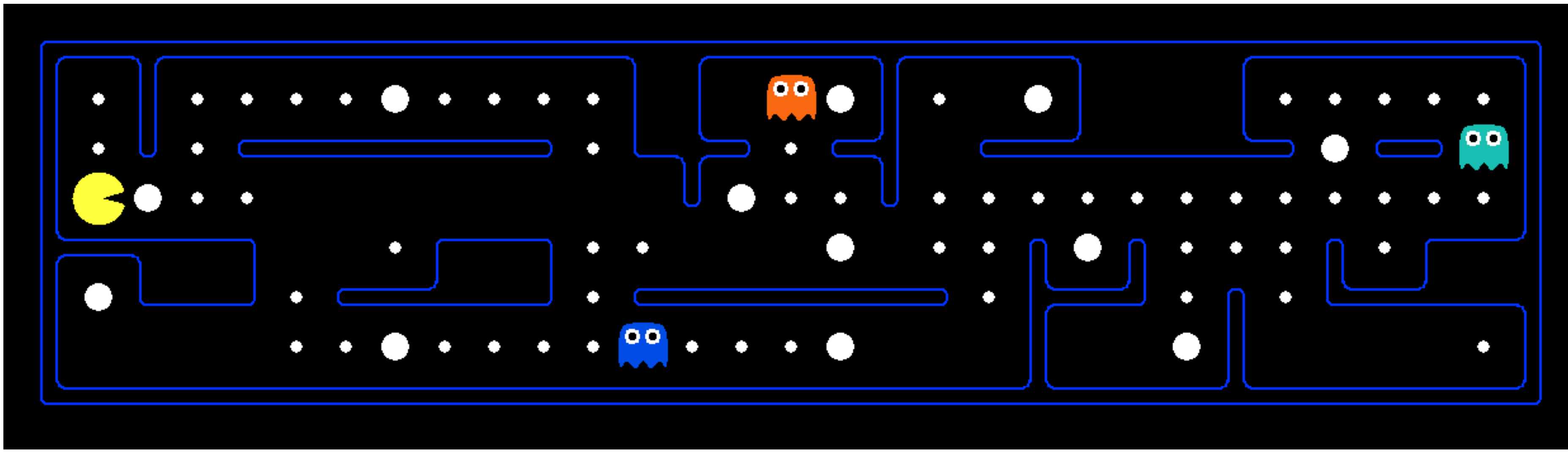
Goal State

State Space Problem Formulation



- states? locations of tiles
- initial state? given
- actions? move blank left, right, up, down
- goal test? goal state (given)
- path cost? 1 per move

Quiz: Safe Passage



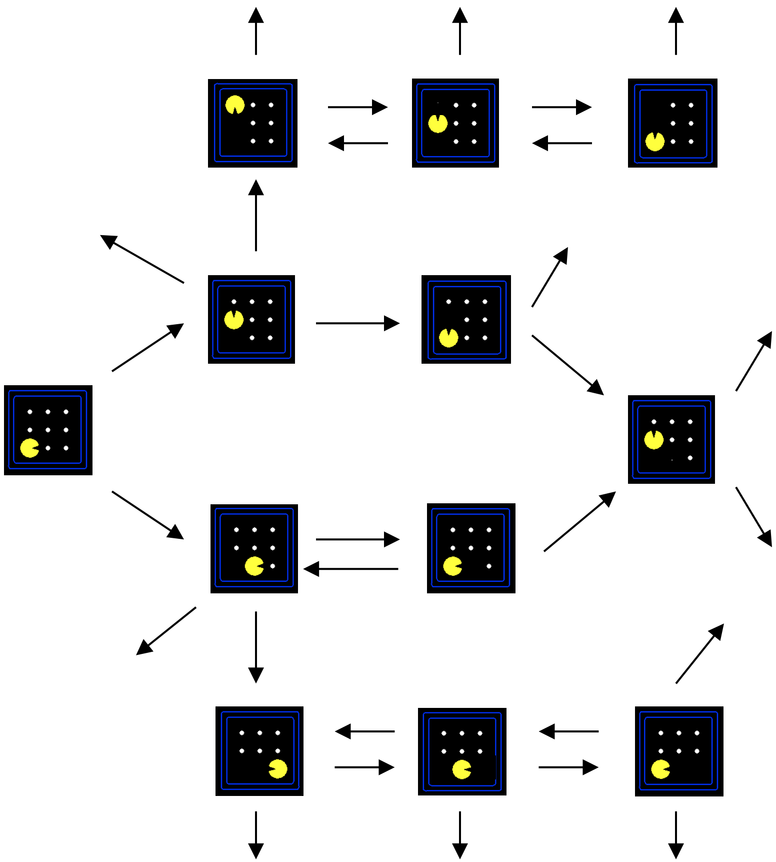
- Problem: eat all dots while keeping the ghosts scared
- What does the state space have to specify?
  - (agent position, dot booleans, power pellet booleans, remaining scared time)

State Space Graphs: 1

- **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)

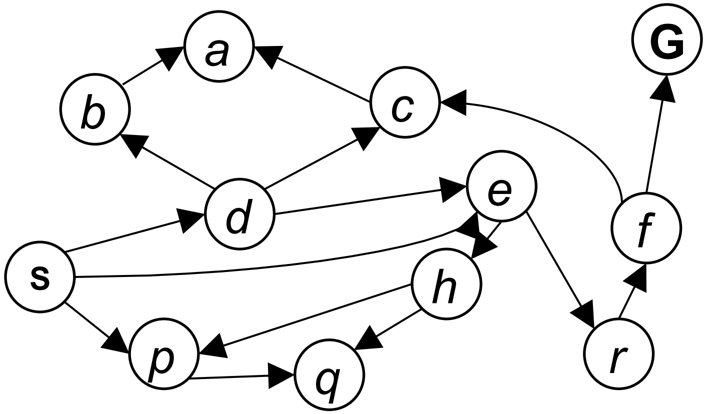
- In a state space graph, each state occurs only once!
  - Careful: if there are loops in this graph, keep track of visited states

- We can rarely build this full graph in memory (it's too big), but it's a useful idea



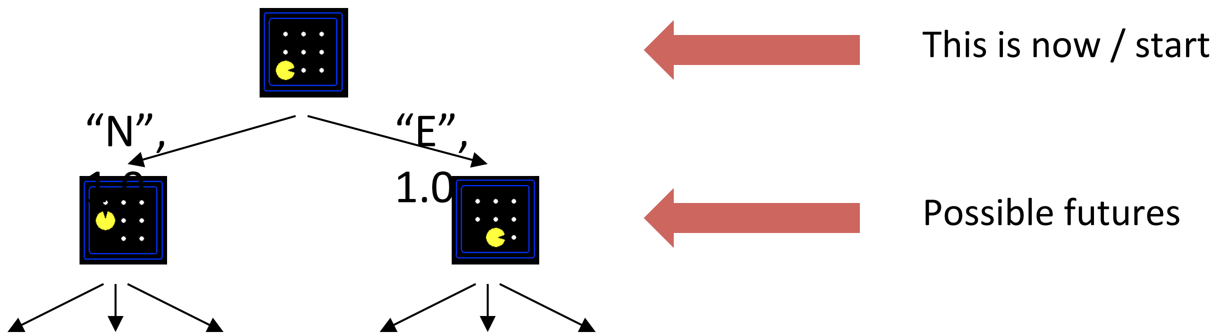
State Space Graphs: 2

- 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 search 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



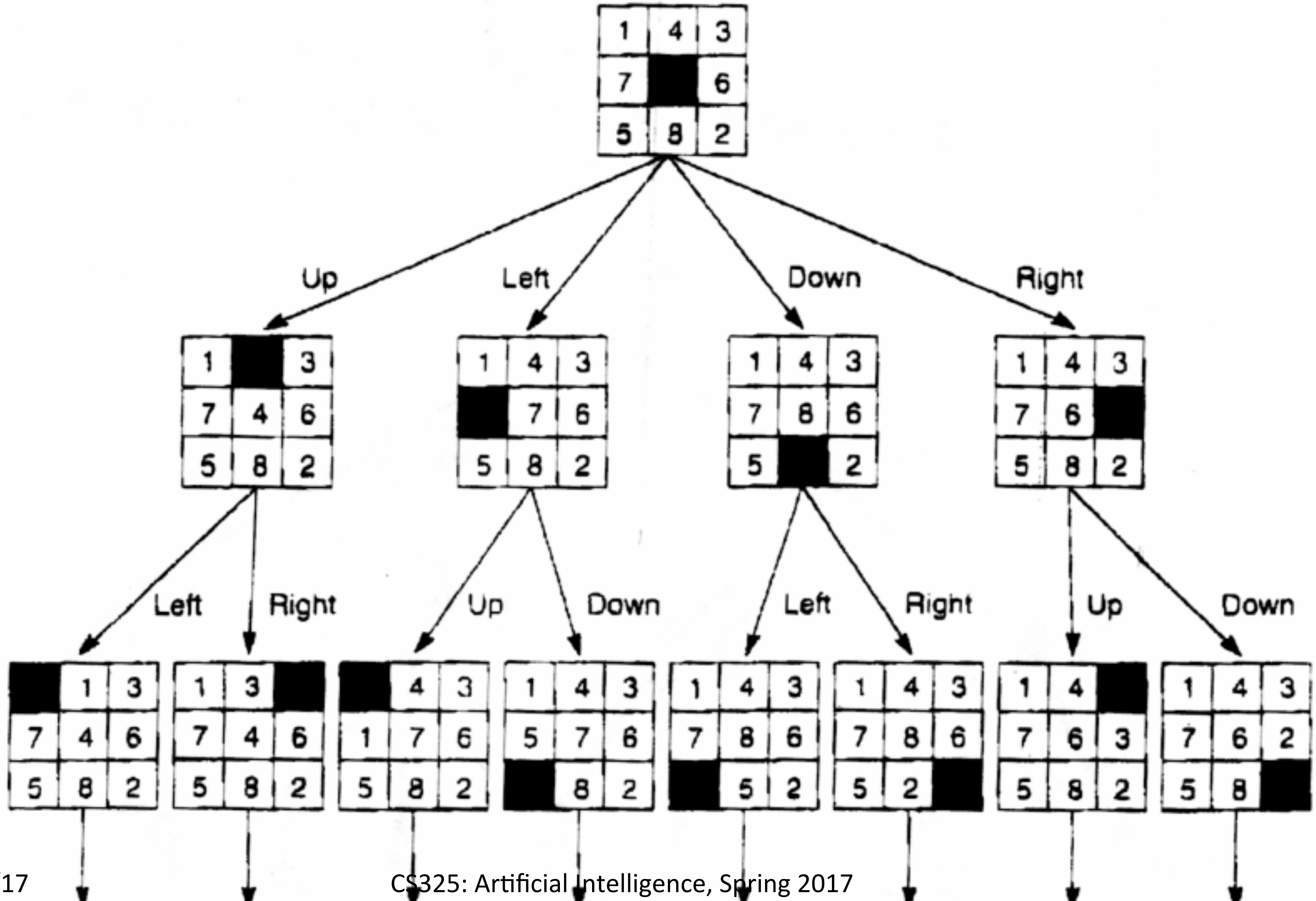
Tiny search graph for a tiny search problem

Search Trees

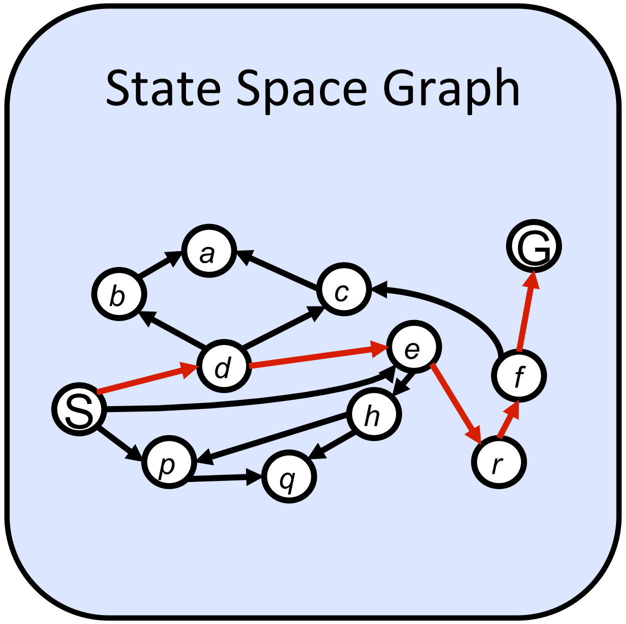


- A search tree:
  - A “what if” tree of plans and their outcomes
  - The start state is the root node
  - Children correspond to successors
  - Nodes show states, but correspond to PLANS (paths from root to the state)
  - For most problems, we can never actually build the whole tree (too large)

Search Tree for 8 puzzle problem

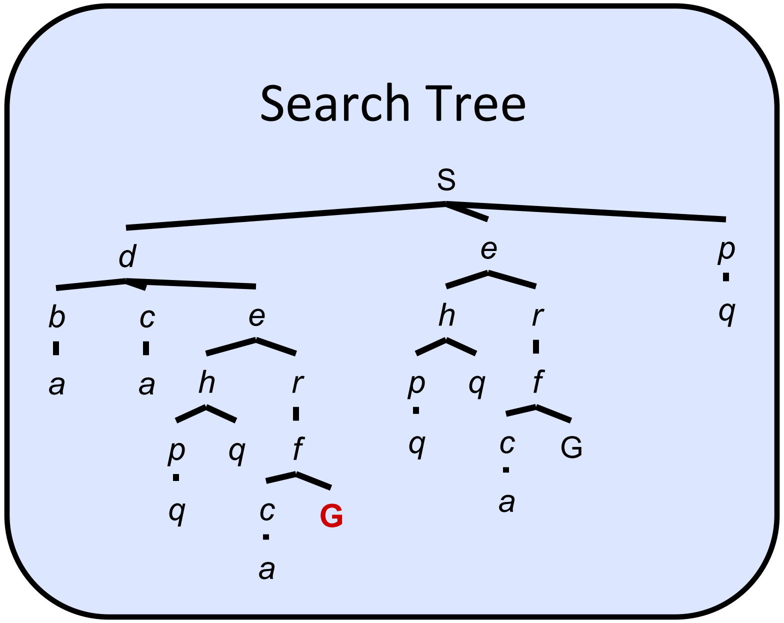


State Space Graphs vs. Search Trees



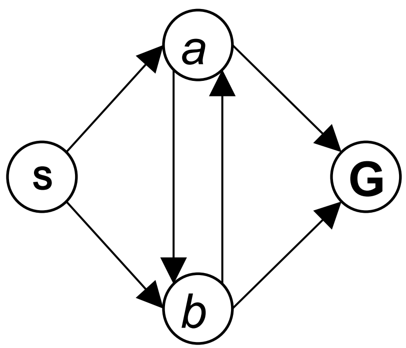
Each NODE in in the search tree is an entire PATH in the state space graph.

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



Quiz: State Space Graphs vs. Search Trees

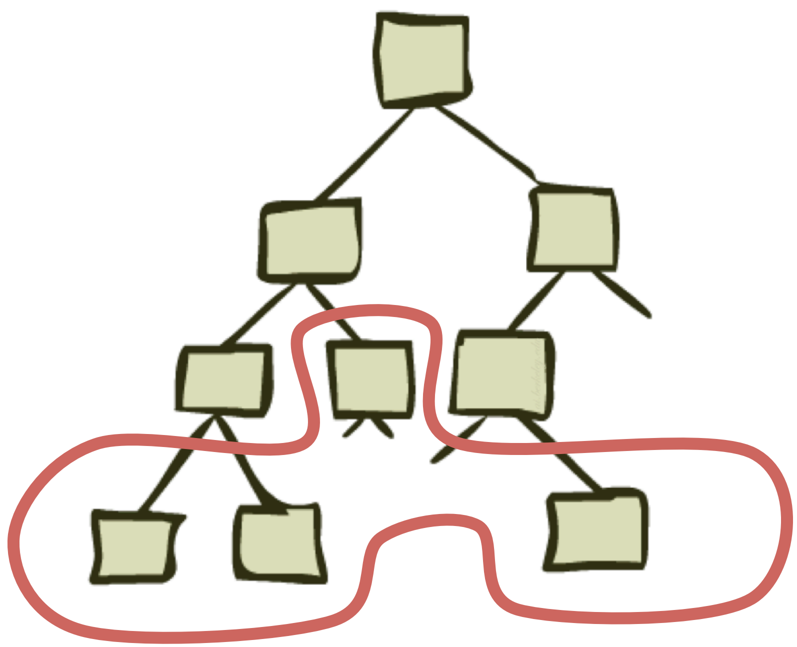
Consider this 4-state graph:



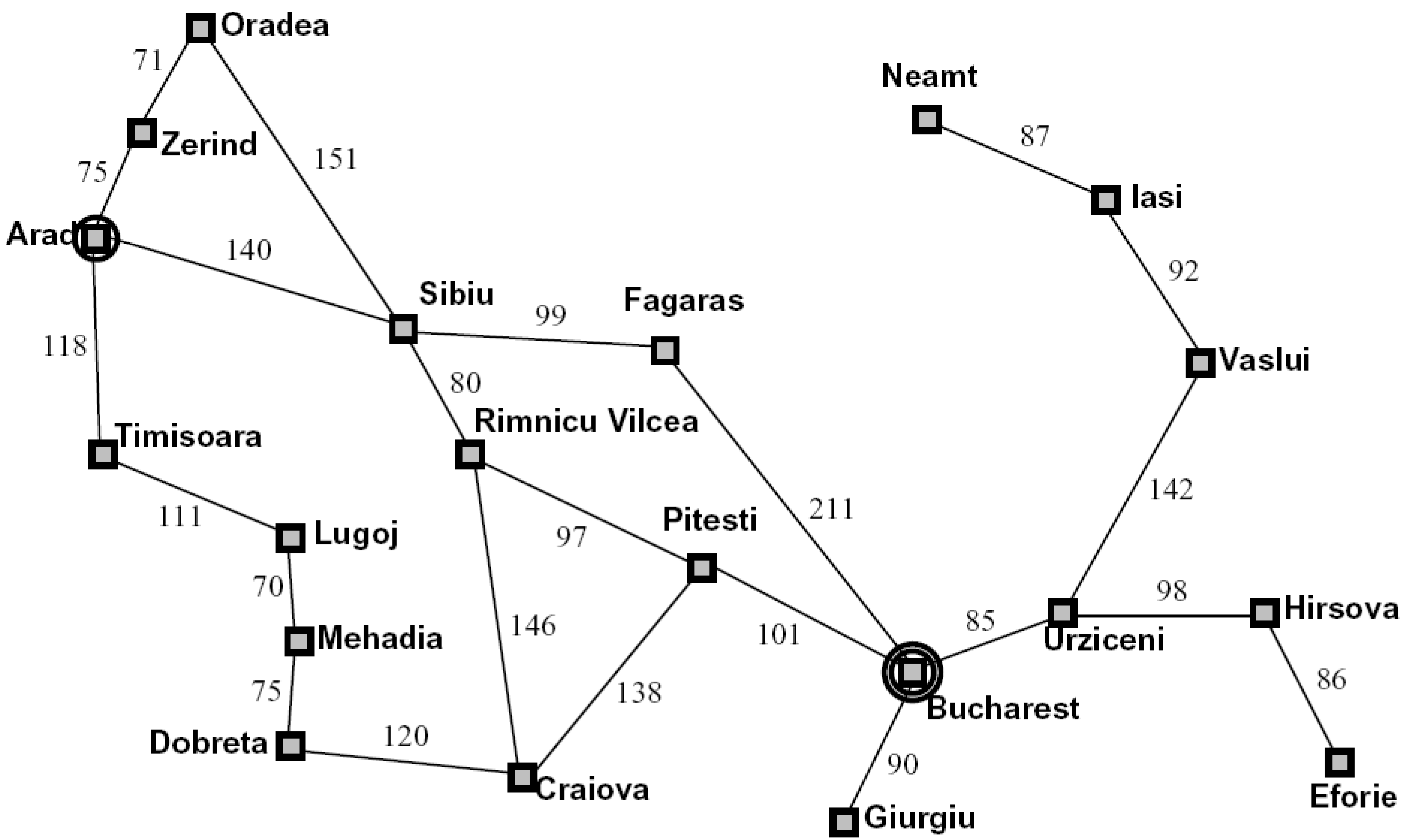
How big is its search tree (from S)?

<u>Problem</u>: Lots of repeated structure in the search tree!

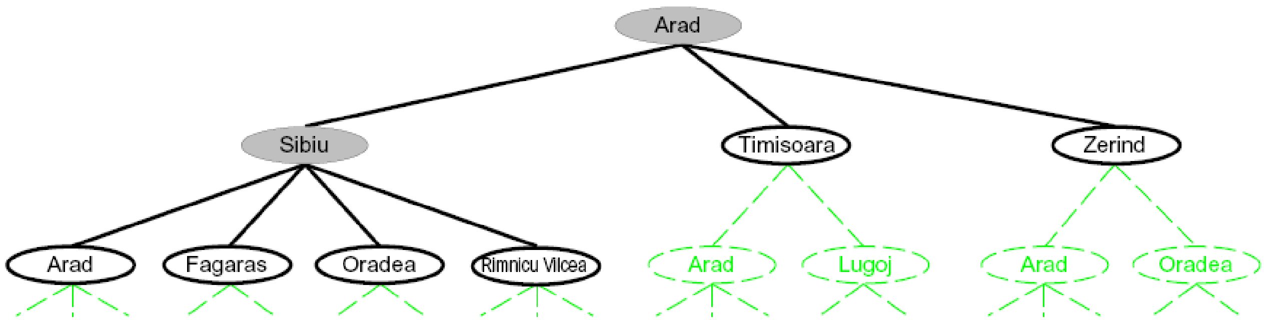
Tree Search



Search Example: Romania



Searching with a Search Tree



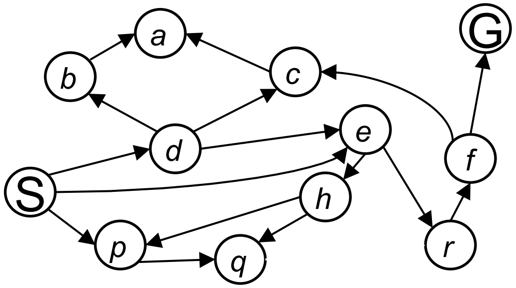
- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a fringe of partial plans under consideration
  - Try to expand as few tree nodes as possible

General Tree Search

```
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
- Main question: which fringe nodes to explore?

Example: Tree Search



[Example: on board]

Why Search can be hard

Assuming b=10, 1000 nodes/sec, 100 bytes/node

Depth of Solution	Nodes to Expand	Time	Memory
0	1	1 millisecond	100 bytes
2	111	0.1 seconds	11 kbytes
4	11,111	11 seconds	1 megabyte
8	10^8	31 hours	11 giabytes
12	10^{12}	35 years	111 terabytes

$$P(40) \approx \frac{64!}{32!\,(8!)^2\,(2!)^6} \approx 10^{43}.$$

Sidebar: Search vs. Intuition


Tania Sachdev

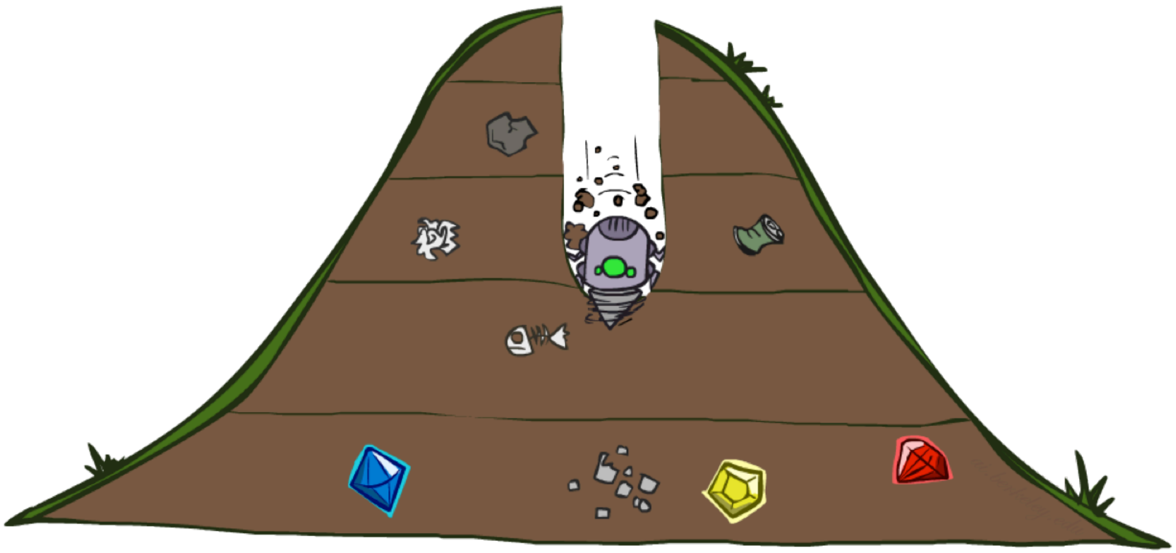
- Human chess grandmasters think “only” 3-5 moves ahead (Kasparov occasionally 12-14) but rely on patterns, intuition

- Deep blue and others: exhaustive search for optimal state/ solution. Evaluates 100M positions/sec, vs. Kasparov 3 positions/sec

Search Strategies

- A search strategy is defined by picking the order of node expansion (fringe exploration)
- Strategies are evaluated along the following dimensions:
  - **completeness**: does it always find a solution if one exists?
  - **time complexity**: number of nodes generated
  - **space complexity**: maximum number of nodes in memory
  - **optimality**: does it always find a least-cost solution?
- Time and space complexity are measured in terms of
  - ***b:*** maximum branching factor of the search tree
  - ***d:*** depth of the least-cost solution
  - ***m***: maximum depth of the state space (may be ∞)

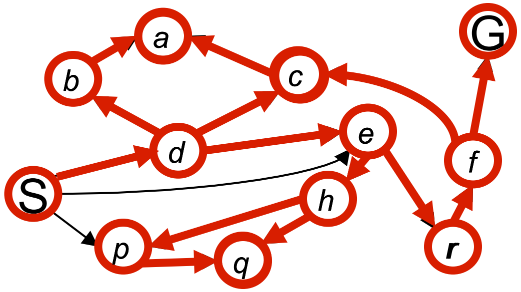
Depth-First Search

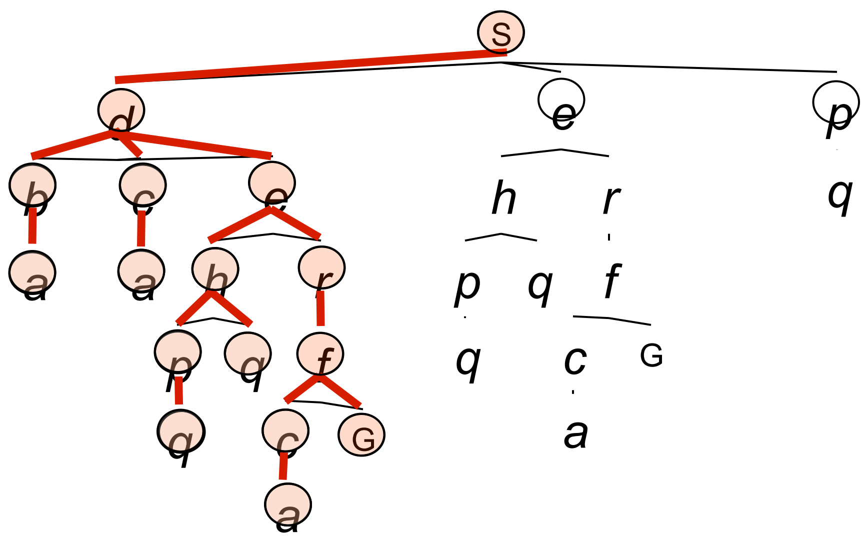


Depth-First Search

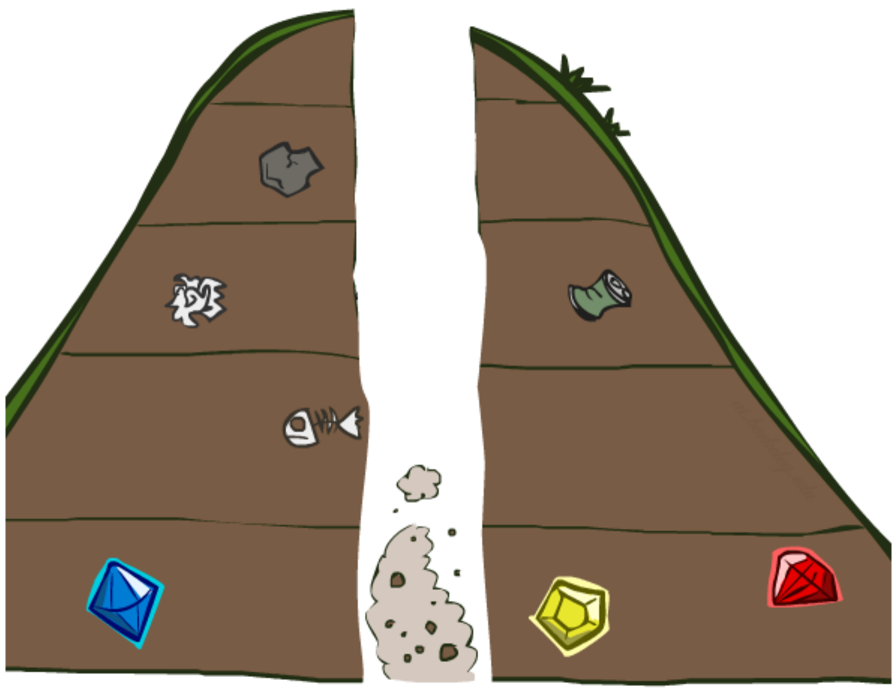
Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack





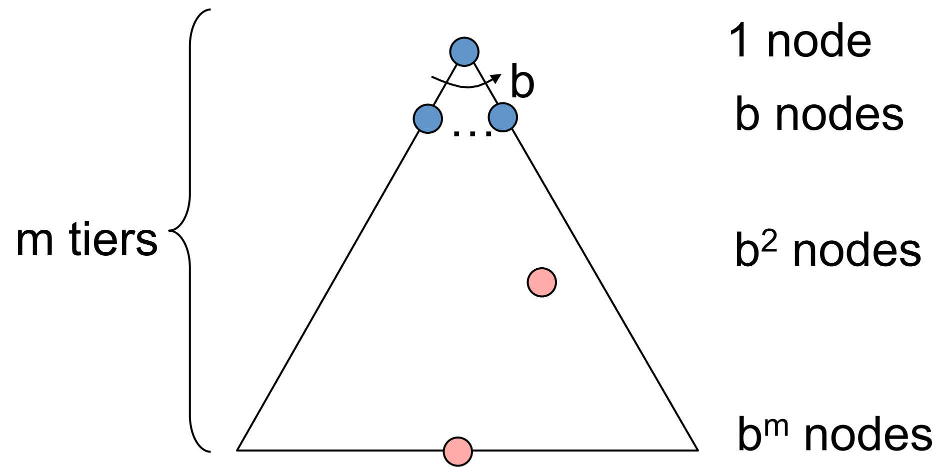
Search Algorithm Properties



Search Algorithm Properties

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

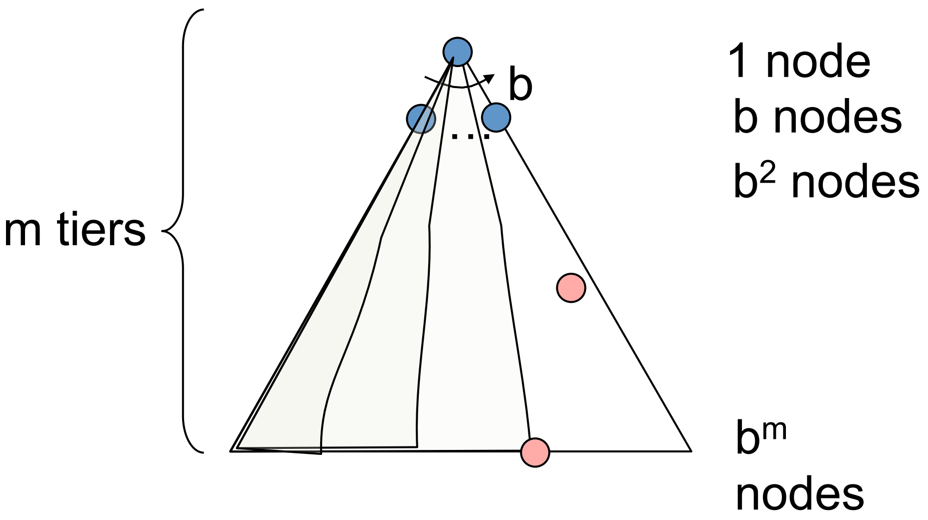
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths



- 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



To Do:

- Enroll in Piazza, check for updates!

- Do the readings (Chapter 3 in R&N)
- Finish Project 0 (posted on website and Piazza)

- Next week: Project 1 will be assigned. hardness: 7
  - **Implication**: go through Python tutorials and do project 0 as soon as possible.