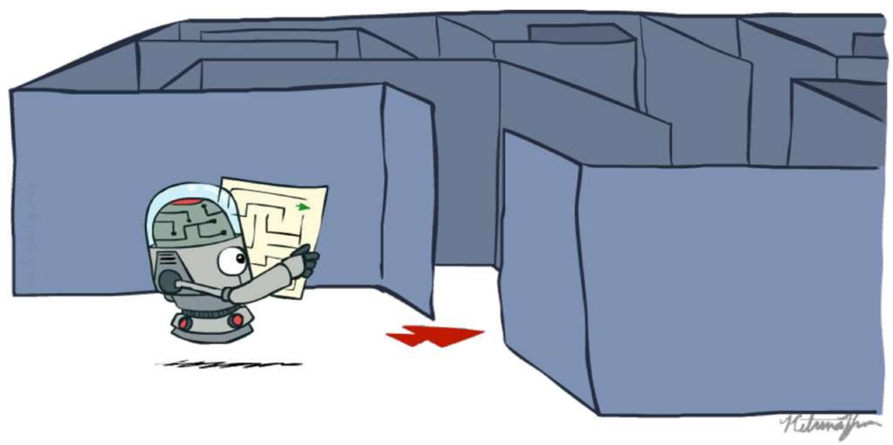
Advanced Artificial Intelligence

Search



Fall 402
Fatemeh Mansoori
University of Isfahan

[These slides are based of the slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley (ai.berkeley.edu).]

Agent

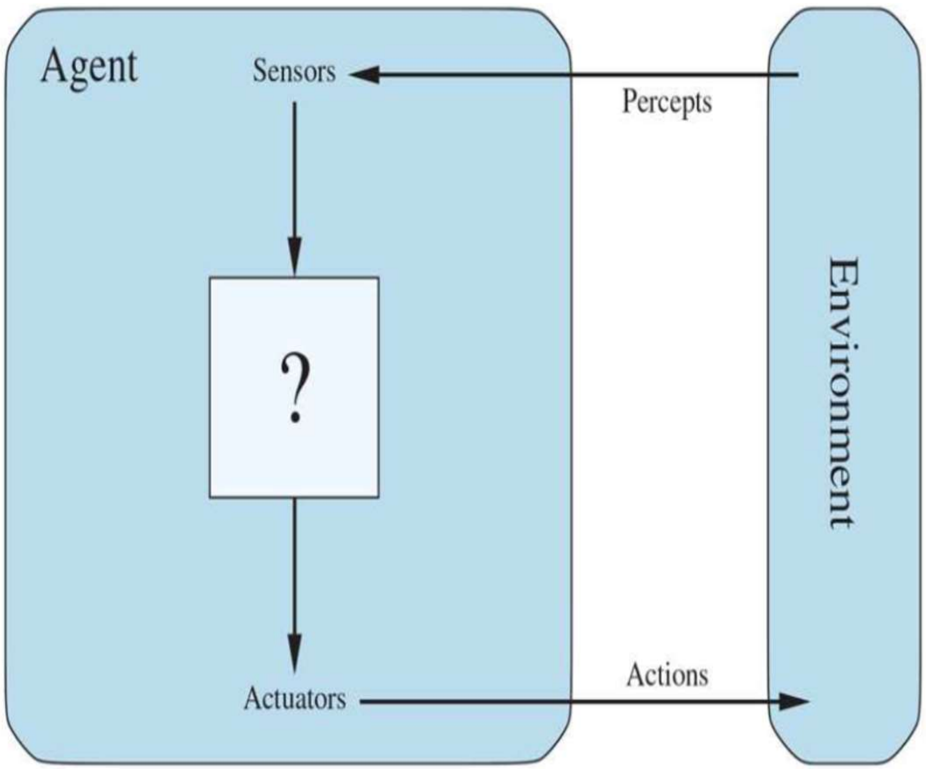
An **agent** is anything that can be viewed as **perceiving its environment** through **sensors** and **acting upon that environment** through **actuators**

- Human agent
- Robot agent
- Software agent

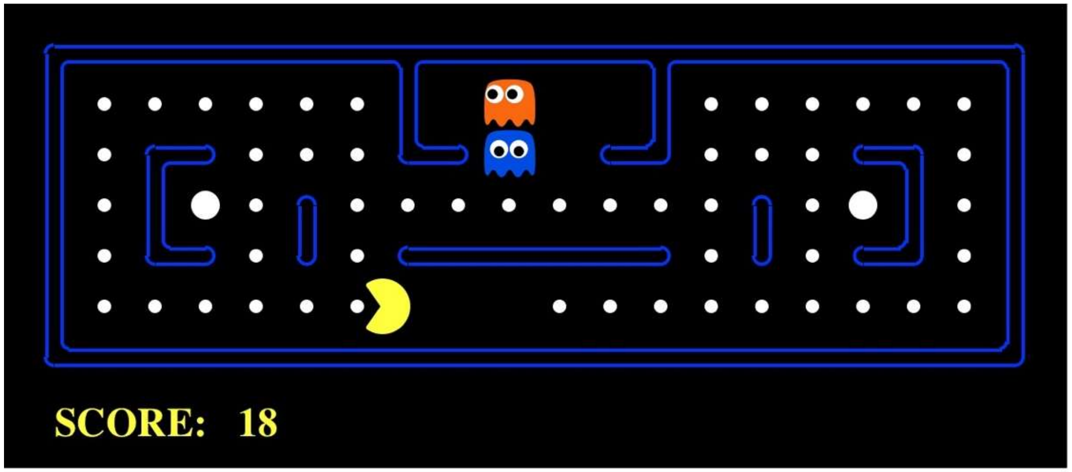
Environment could be everything. Maybe the entire universe!

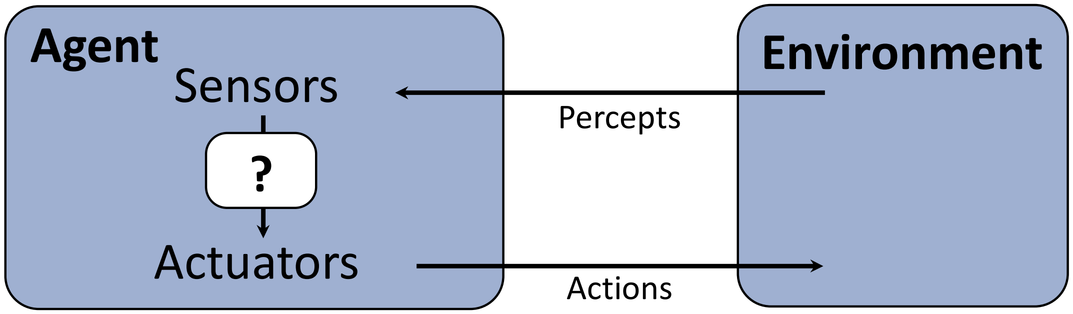
In practice, that part of the universe whose states we care about

- what the agent perceives
- What is affected by the agents's actions.



Pac-Man as an Agent





Pac-Man is a registered trademark of Namco-Bandai Games, used here for educational purposes

Demo1: pacman-l1.mp4 or L1D2

Agent

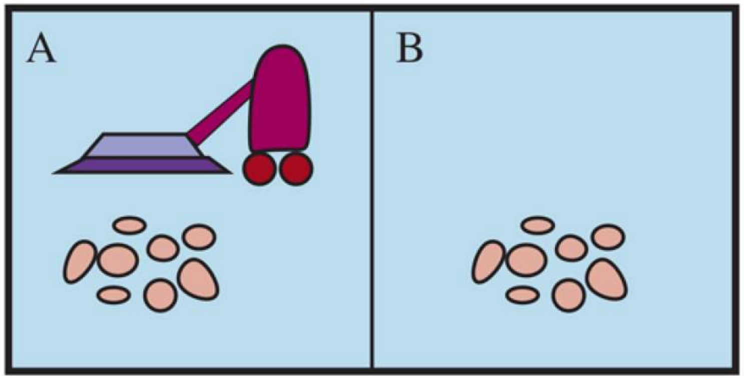
- **Percept**
  - content an agent’s sensors are perceiving
- **Percept sequence**
  - complete history of everything the agent has ever perceived

an agent’s choice of action at any given instant can depend on its built-in knowledge and on the entire percept sequence observed to date, but not on anything it hasn’t perceived

agent’s behavior is described by the **agent function** that maps any given percept sequence to an action.

agent function for an artificial agent will be implemented by an **agent program**.

A vacuum-cleaner



Available Actions: move to right, move to left, suck, do noting

very simple agent function: if the current square is dirty, then suck; otherwise, move to the other square

Percept sequence	Action
[A, *Clean*]	*Right*
[A, *Dirty*]	*Suck*
[B, *Clean*]	*Left*
[B, *Dirty*]	*Suck*
[A, *Clean*], [A, *Clean*]	*Right*
[A, *Clean*], [A, *Dirty*]	*Suck*
⋮	⋮
[A, *Clean*], [A, *Clean*], [A, *Clean*]	*Right*
[A, *Clean*], [A, *Clean*], [A, *Dirty*]	*Suck*
⋮	⋮

Rational Agents

- An **agent** is an entity that *perceives* and *acts*.

- Abstractly, an agent is a function from percept histories to actions :

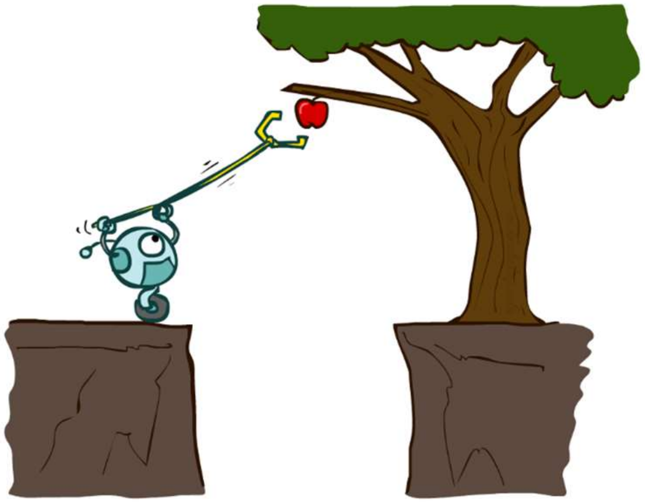
$$f : \mathcal{P}^* \rightarrow \mathcal{A}$$

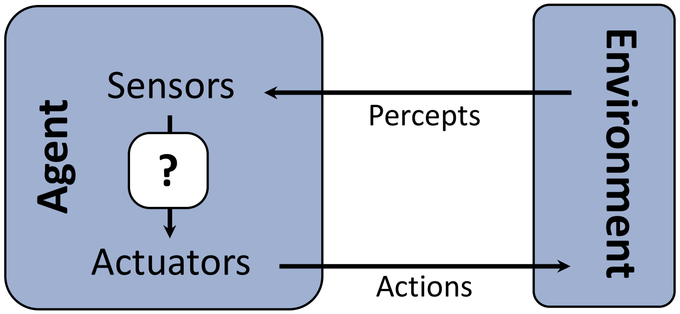
- For any given class of environments and tasks, we seek the agent (or class of agents) with the best (expected) performance (or utility)

- A **rational agent** selects actions that maximize its (expected) **utility**.

Caveat: computational limitations make perfect rationality unachievable

design best program for given machine resources





Rationality

- A **rational agent** is one that does the right thing
- evaluate an agent's behavior by its consequences
- Agent generates a sequence of actions according to the percepts it receives
- The sequence of actions causes the environment to go through a sequence of states
  - If the sequence is desirable, then the agent has performed well.
  - desirability is captured by a **performance measure** that evaluates any given sequence of environment states.

Performance measure

- Humans have desires and preferences of their own
- Machines, do *not* have desires and preferences of their own; the performance measure is, initially at least, in the mind of the designer of the machine, or the users the machine

- E.g. vacuum-cleaner agent :
  - amount of dirt cleaned up in a single eight-hour
  - It is good ?

  - One point could be awarded for each clean square at each time step
  - Perhaps with a penalty for electricity consumed and noise generated

As a general rule, it is better to design performance measures according to what one actually wants to be achieved in the environment, rather than according to how one thinks the agent should behave.

Rationality

What is rational at any given time depends on four things:

- The performance measure that defines the criterion of success
- The agent's prior knowledge of the environment.
- The actions that the agent can perform.
- The agent's percept sequence to date.

a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has

Is it rational ?

vacuum-cleaner agent that cleans a square if it is dirty and moves to the other square

Task Environment

- In rationality of the simple vacuum-cleaner age, we had to specify
  - the performance measure
  - the environment
  - the agent's actuators and sensors.
- We group all these under the heading of the **task environment**

- we call this the **PEAS** (**P**erformance, **E**nvironment, **A**ctuators, **S**ensors)
- In designing an agent, the first step must always be to specify the task environment as fully as possible.

PEAS

we had to specify the performance measure, the environment, and the agent's actuators and sensors

PEAS (**P**erformance, **E**nvironment, **A**ctuators, **S**ensors)

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits, minimize impact on other road users	Roads, other traffic, police, pedestrians, customers, weather	Steering, accelerator, brake, signal, horn, display, speech	Cameras, radar, speedometer, GPS, engine sensors, accelerometer, microphones, touchscreen

PEAS description of the task environment for an automated taxi driver.

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments	Touchscreen/voice entry of symptoms and findings
Satellite image analysis system	Correct categorization of objects, terrain	Orbiting satellite, downlink, weather	Display of scene categorization	High-resolution digital camera
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, tactile and joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, raw materials, operators	Valves, pumps, heaters, stirrers, displays	Temperature, pressure, flow, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, feedback, speech	Keyboard entry, voice

Examples of agent types and their PEAS descriptions

Properties of task environment

- **FULLY OBSERVABLE VS. PARTIALLY OBSERVABLE**
- **Fully observable**
  - agent's sensors give it access to the complete state of the environment at each point in time
- Effectively fully observable
  - sensors detect all aspects that are *relevant* to the choice of action
- Fully observable environments are convenient because the agent need not maintain any internal state to keep track of the world

- **partially observable**
  - noisy and inaccurate sensors
  - parts of the state are simply missing from the sensor data
    - Vacuum agent with only a local dirt sensor
- **Unobservable**
  - the agent has no sensors at all

vs.

Properties of task environment

- **SINGLE-AGENT VS. MULTIAGENT**
- agent solving a crossword puzzle
  - e.g. by itself is clearly in a single-agent environment
  - E.g. agent playing chess is in a two-agent environment.

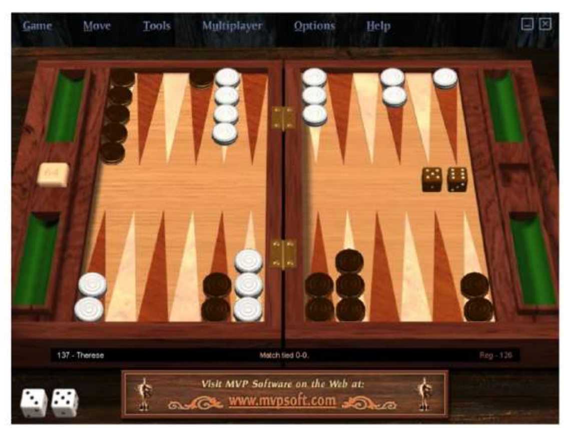
  - Competitive vs **cooperative**

vs.

Properties of task environment

- **Deterministic vs. nondeterministic**
    - Deterministic : next state of the environment is completely determined by the current state and the action executed by the agent
- an agent need not worry about uncertainty in a fully observable, deterministic environment
- the word **stochastic** is used by some as a synonym for “nondeterministic”

vs.



Properties of task environment

- **EPISODIC VS. SEQUENTIAL :**
  - episodic : the agent's experience is divided into atomic episodes.
  - In each episode the agent receives a percept and then performs a single action
  - the next episode does not depend on the actions taken in previous episodes
  - Many classification tasks are episodic : an agent that has to spot defective parts on an assembly line bases each decision on the current part
- In sequential environments, on the other hand, the current decision could affect

all future decisions

- Episodic environments are much simpler than sequential environments because the agent does not need to think ahead

vs.

Properties of task environment

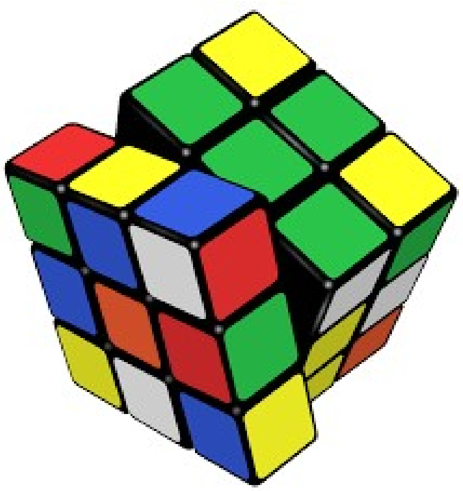
DISCRETE VS. CONTINUOUS :

- discrete/continuous distinction applies to the *state* of the environment, to the way *time* is handled and to the *percepts* and *actions* of the agent.
- chess environment has a finite number of distinct states (excluding the clock). Chess also has a discrete set of percepts and actions
- Taxi driving is a continuous-state and continuous-time problem

vs.

Properties of task environment

- **STATIC VS. DYNAMIC:**
  - Dynamic :
    - environment can change while an agent is thinking
  - Semidynamic :
    - the environment does not change with the passage of time but the agent’s performance score does



VS

- **KNOWN VS. UNKNOWN :**
  - is not strictly a property of the environment
  - In a known environment, the outcomes (or outcome probabilities if the environment is nondeterministic) for all actions are given
  - if the environment is unknown, the agent will have to learn how it works in order to make good decisions
  - distinction between known and unknown environments is not the same as the one between fully and partially observable environments.
  - A *known* environment can be *partially* observable :
    - E.g. solitaire game I know the rules but am still unable to see the cards that have not yet been turned over
  - An *unknown* environment can be *fully* observable :
    - in a new video game, the screen may show
    - the entire game state but I still don’t know what the buttons do until I try them.

Properties of task environment

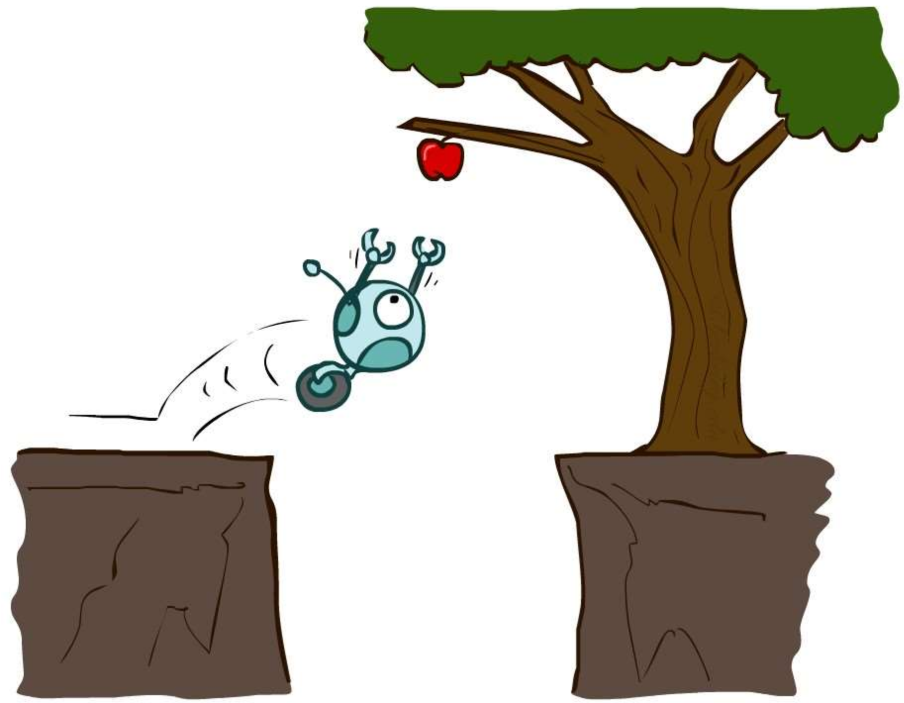
Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

Examples of task environments and their characteristics.

AGENTS

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?

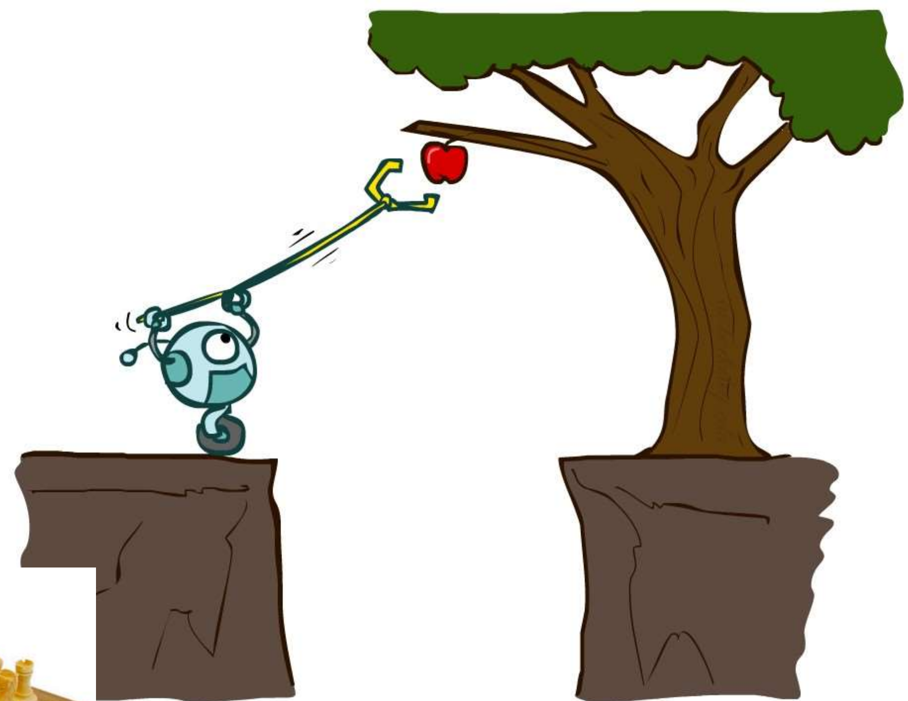


[Demo: reflex optimal (L2D1)]
[Demo: reflex optimal (L2D2)]

Goal based 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



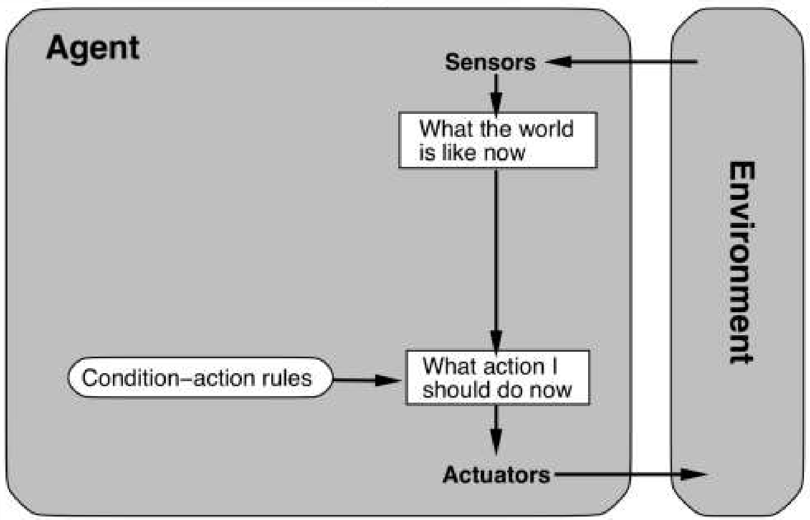
[Demo: re-planning (L2D3)]
[Demo: mastermind (L2D4)]

Utility based agent

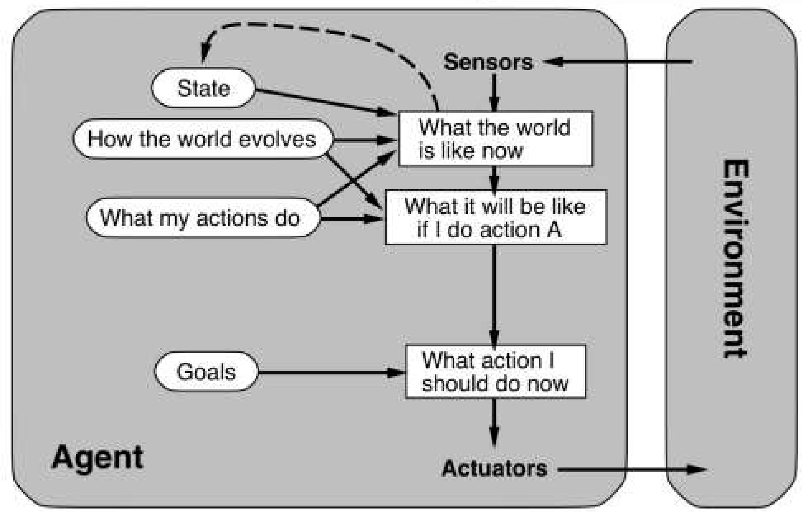
Utility Based Agents:

- Like goal-based, but
- Trade off multiple goals
- Reason about probabilities of outcomes
- Act on how the world will **LIKELY** be

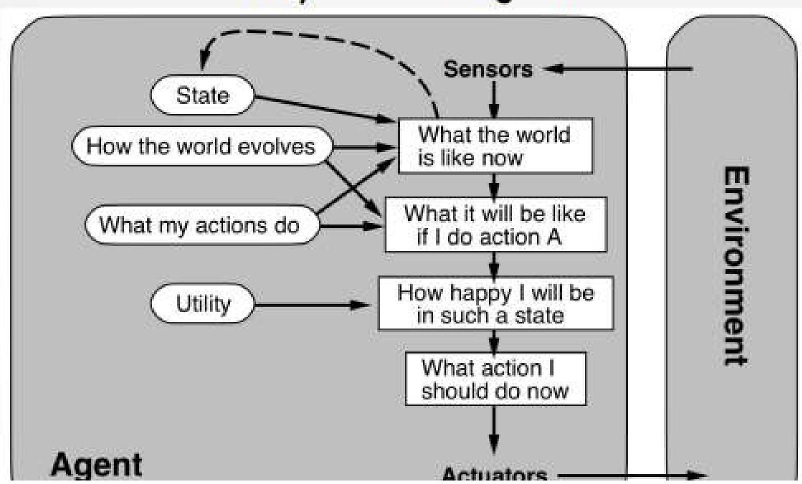
Reflex Agent



Goal-based Agent



Utility-based Agent

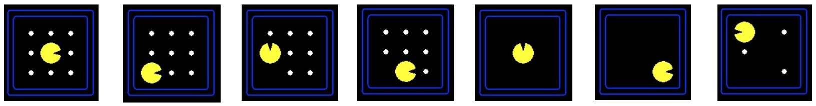


We need to make one important refinement to the standard model to account for the fact that perfect rationality—always taking the exactly optimal action—is not feasible in complex environments. The computational demands are just too high. Chapters 5 and 17 deal with the issue of **limited rationality**—acting appropriately when there is not enough time to do all the computations one might like. However, perfect rationality often remains a good starting point for theoretical analysis.

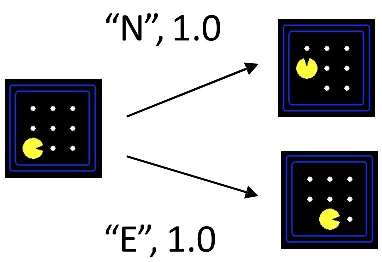
Search Problems

Search Problems

- A search problem consists of:
  - A state space



  - A successor function (with actions, costs)

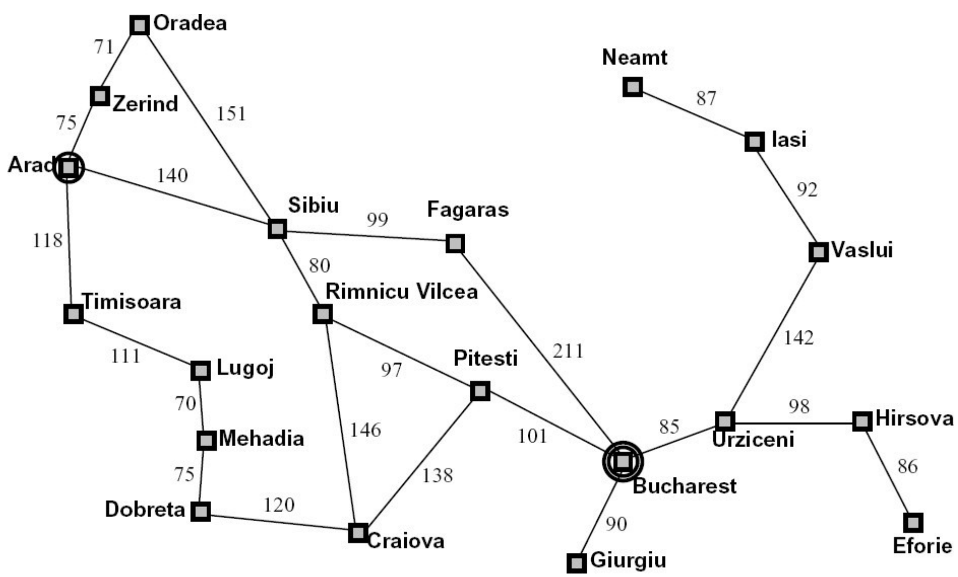


  - A start state and a goal test

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

Search Problems Are Models

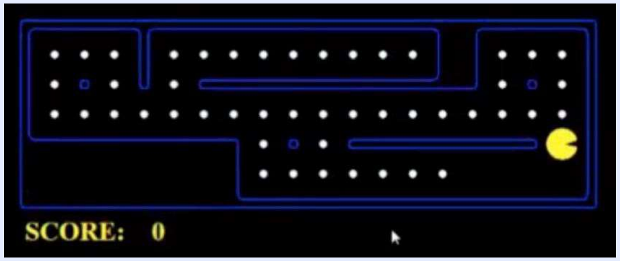
Example: Traveling in Romania



- 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



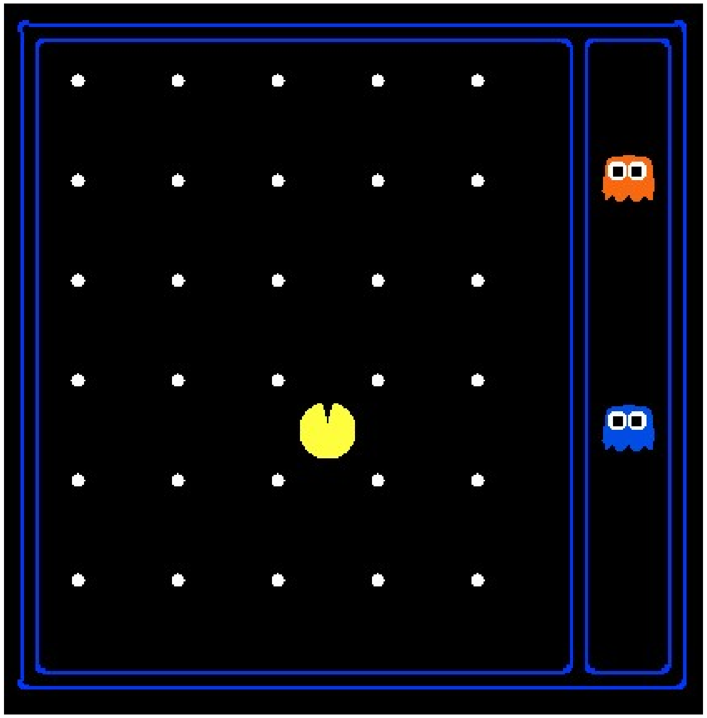
A search state keeps only the details needed for planning (abstraction)

- Problem: Pathing
  - 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

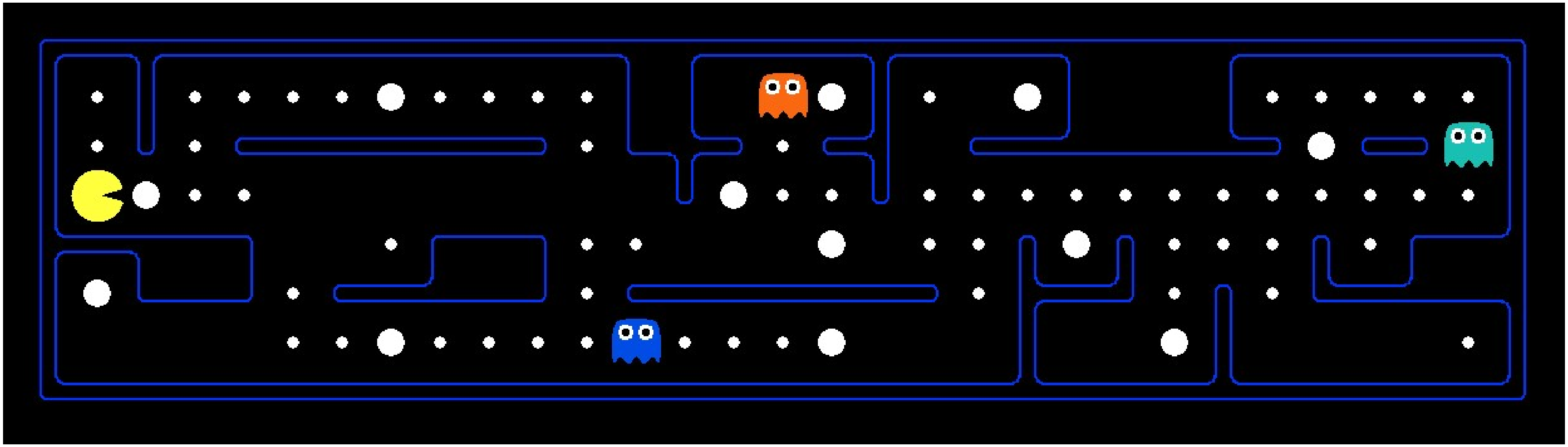
State Space Sizes?

- 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 pathing?
    120
  - States for eat-all-dots?
    $120 \times (2^{30})$

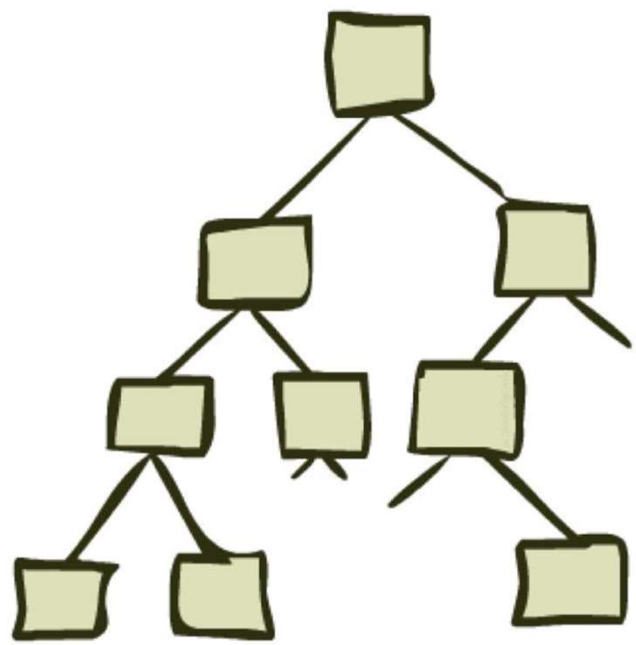


Quiz: Safe Passage



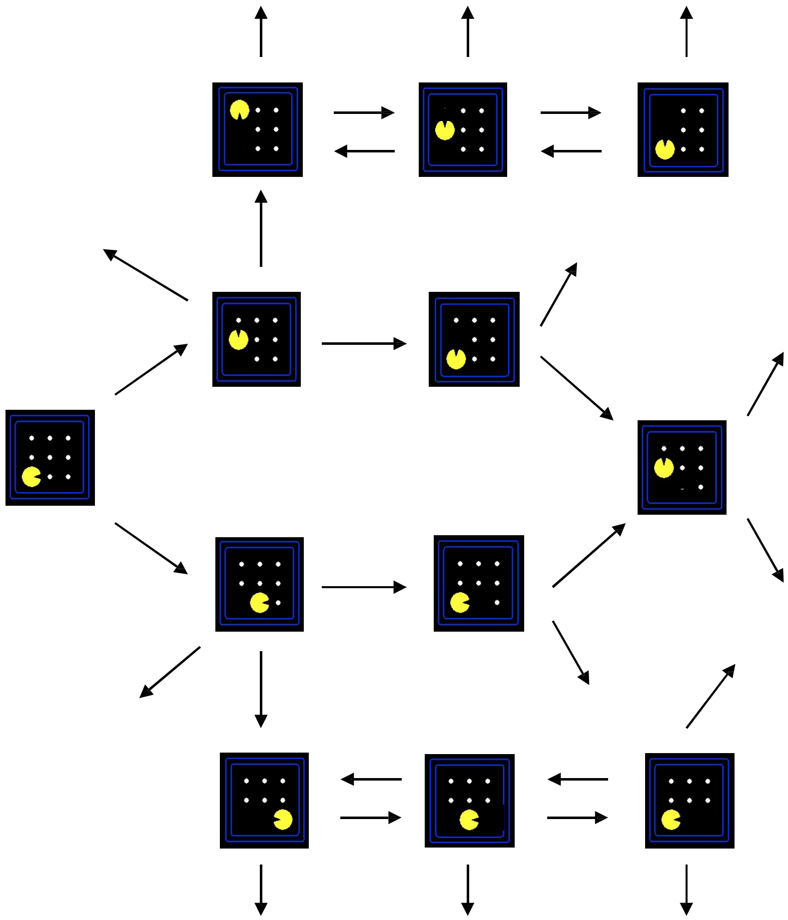
- Problem: eat all dots while keeping the ghosts perma-scared
- What does the state space have to specify?
  - (agent position, dot booleans, power pellet booleans, remaining scared time)

State Space Graphs and Search Trees



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

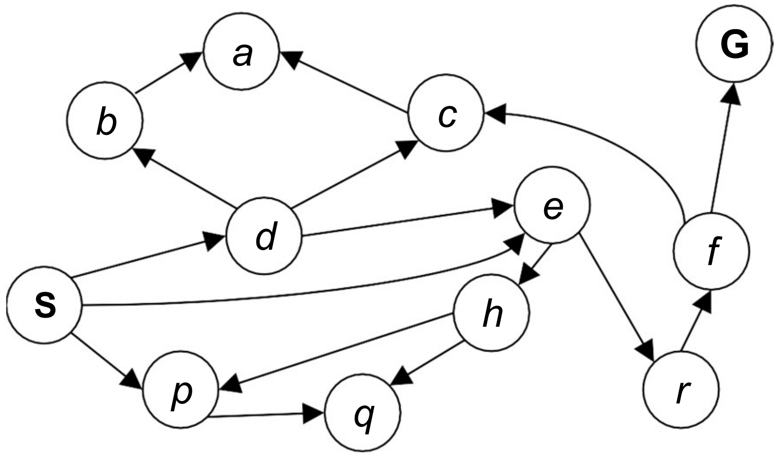


State Space Graphs

- 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)

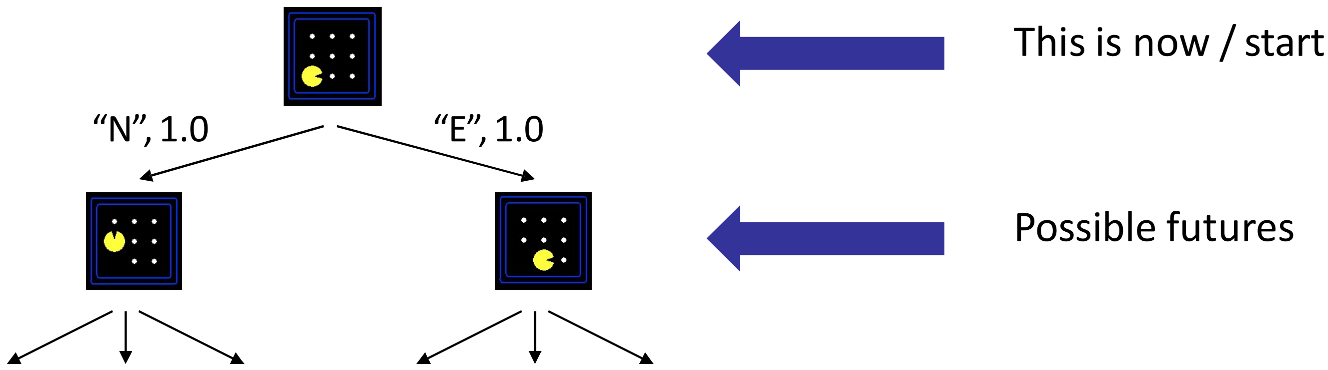
- In a state space graph, each state occurs only once!

- 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

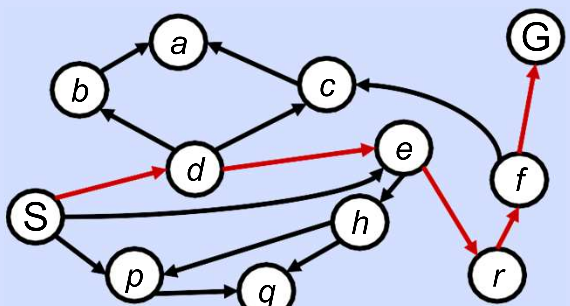
Search Trees



- 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

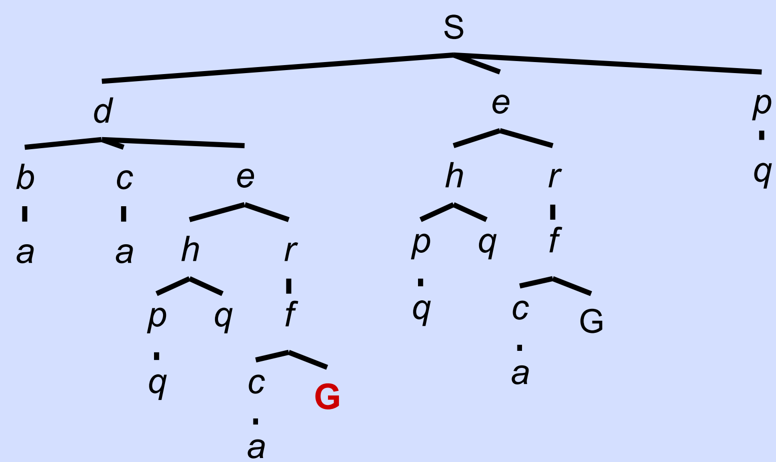
State Space Graph



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

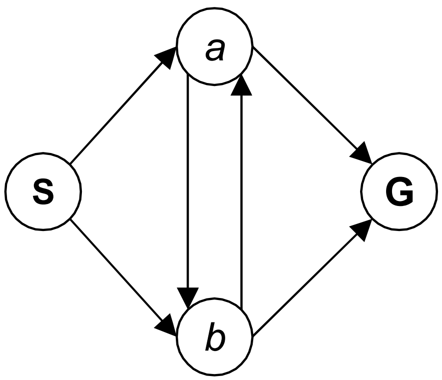
We construct both on demand – and we construct as little as possible.

Search Tree

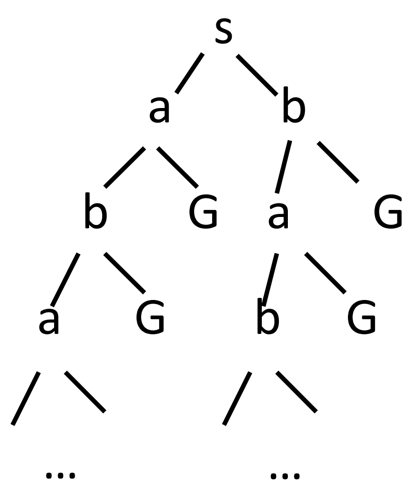


Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



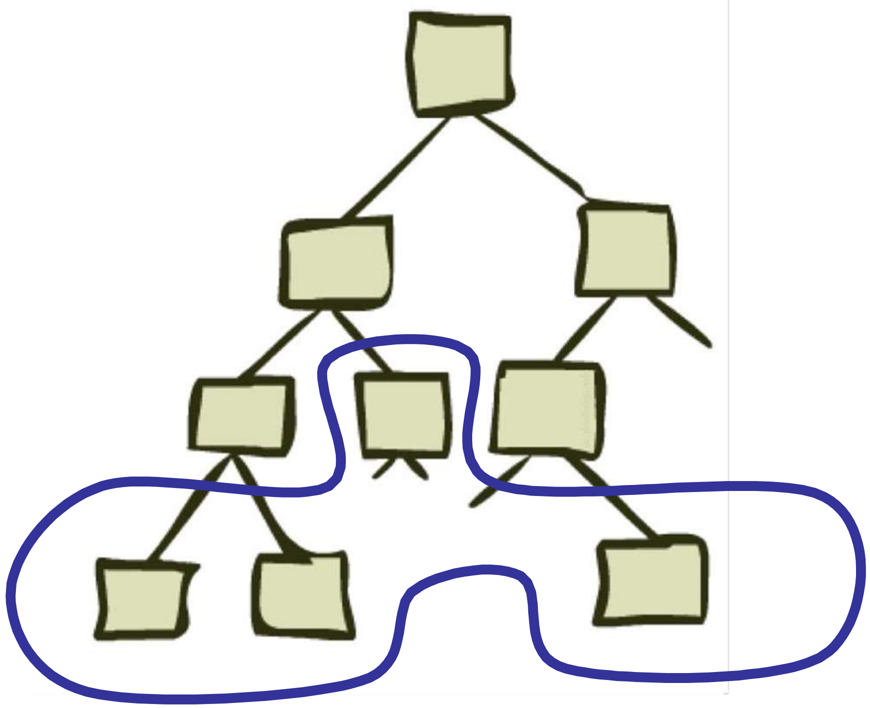
How big is its search tree (from S)?



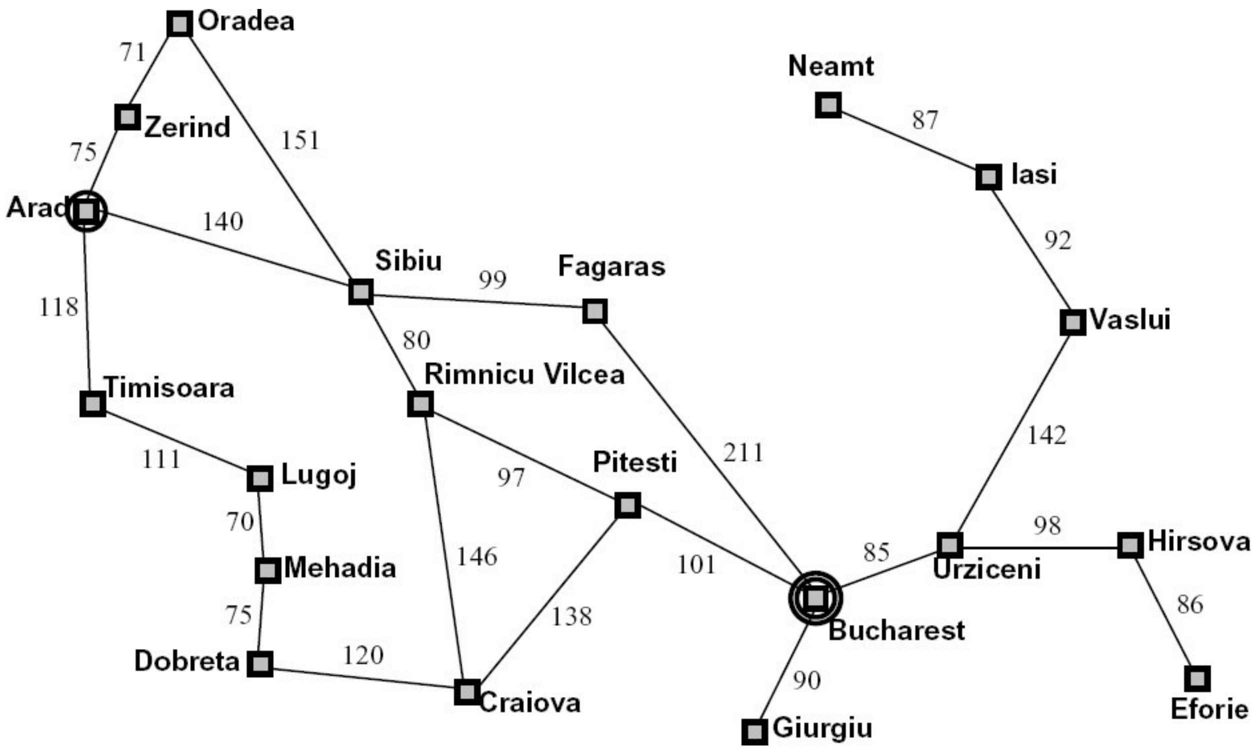
∞

Important: Lots of repeated structure in the search tree!

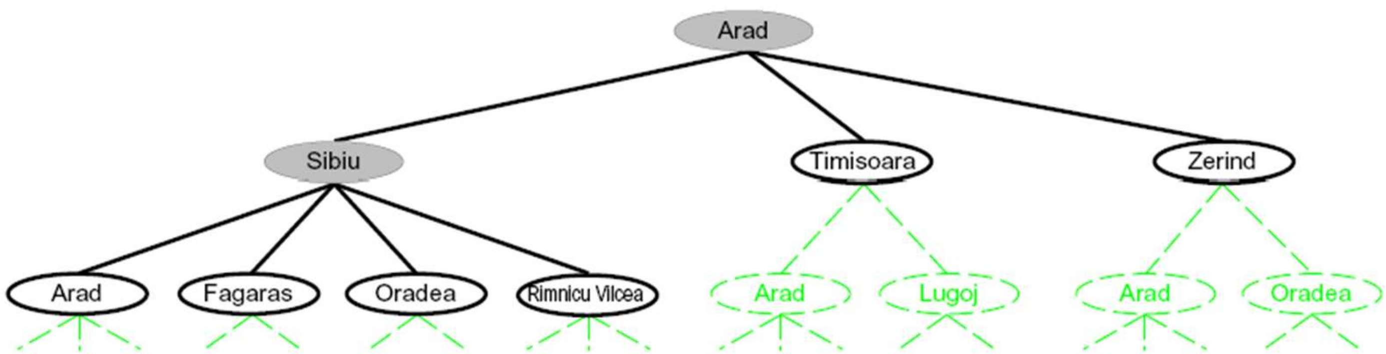
Tree Search



Search Example: Romania



Searching with a Search Tree



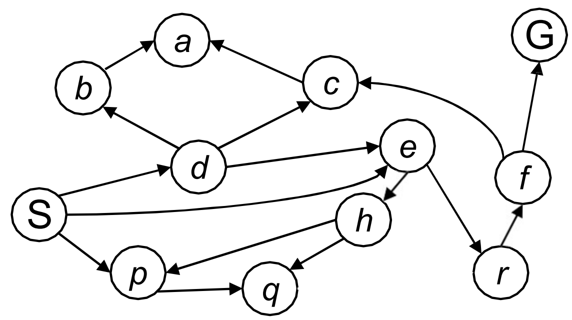
- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a fringe of partial plans under consideration
  - Try to expand as few tree nodes as possible

General Tree Search

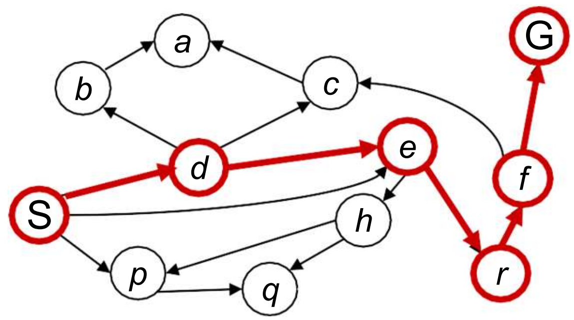
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the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

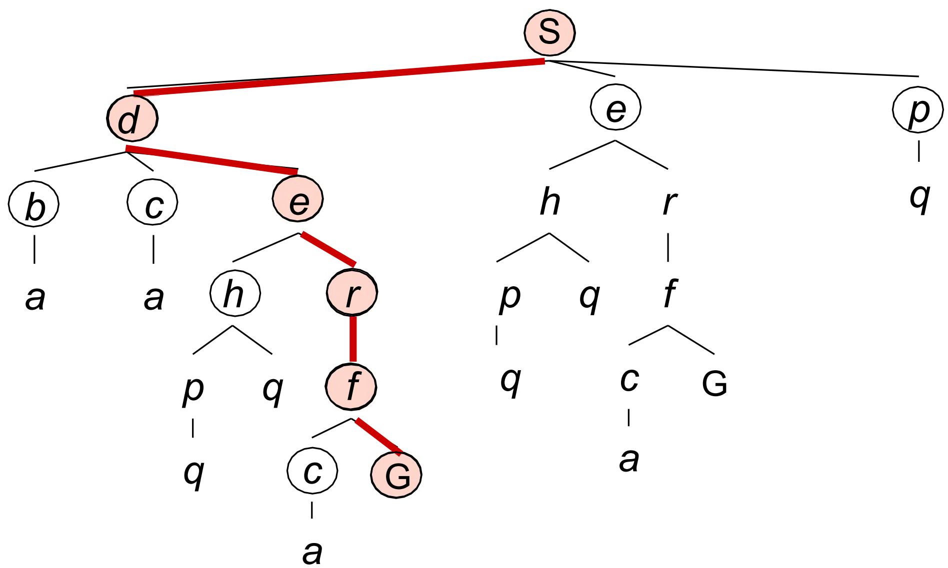
- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy
- Main question: which fringe nodes to explore?

Example: Tree Search



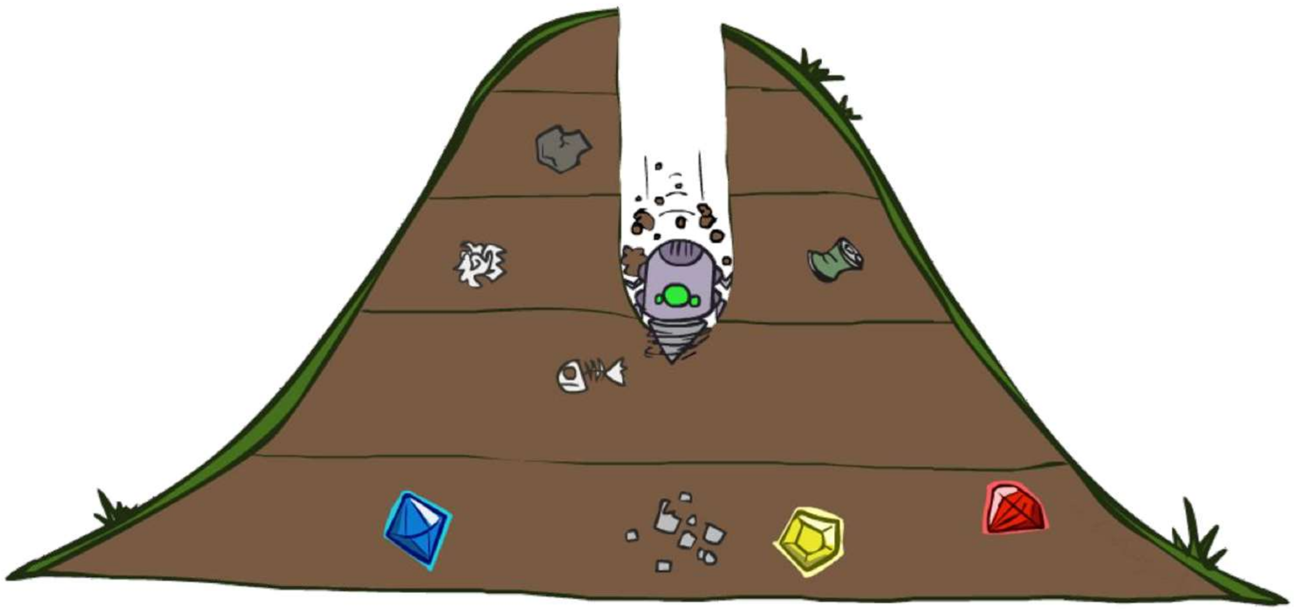
Example: Tree Search





~~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~~

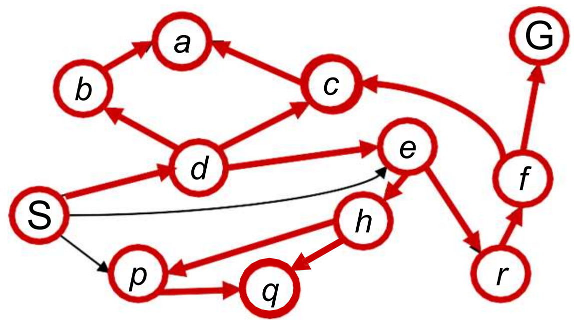
Depth-First Search

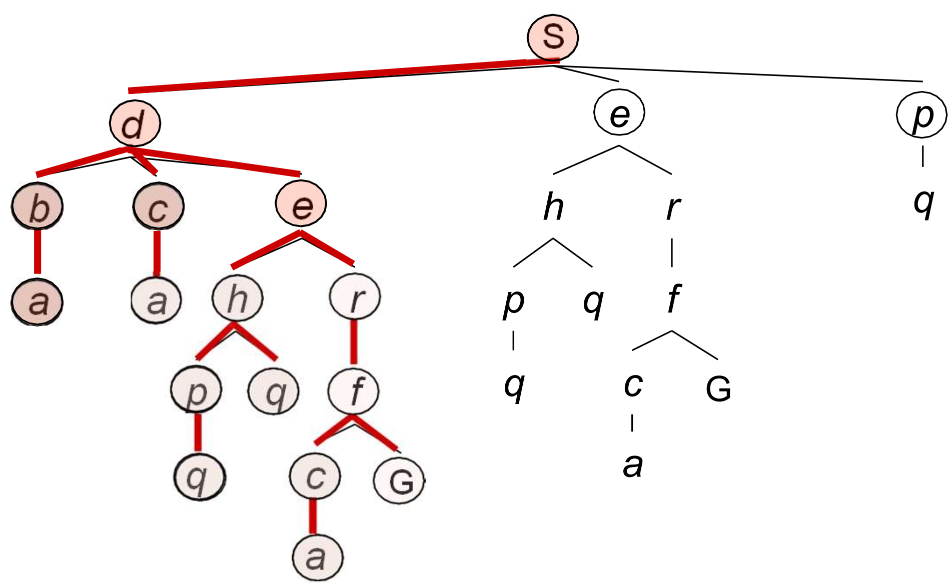


Depth-First Search

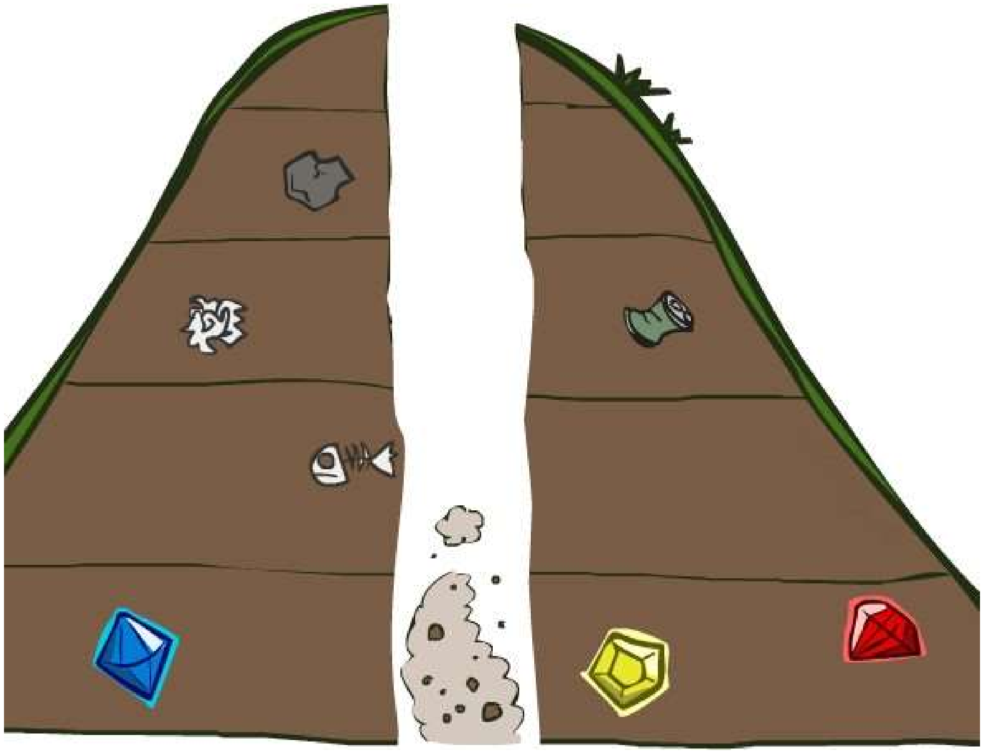
Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack





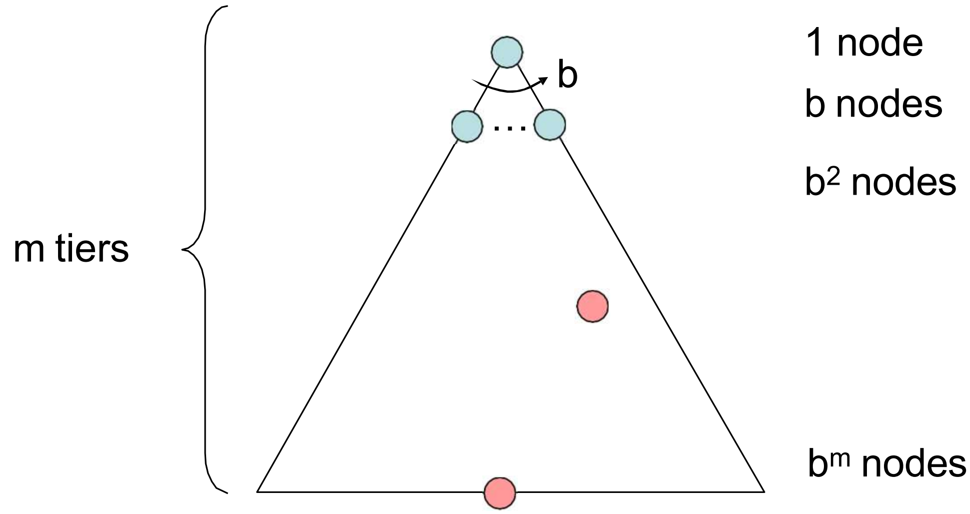
Search Algorithm Properties



Search Algorithm Properties

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

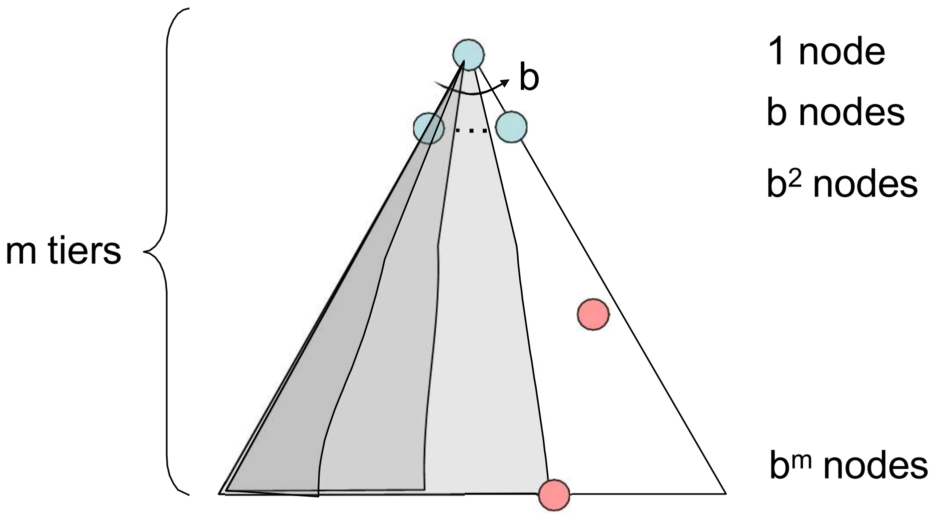
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths



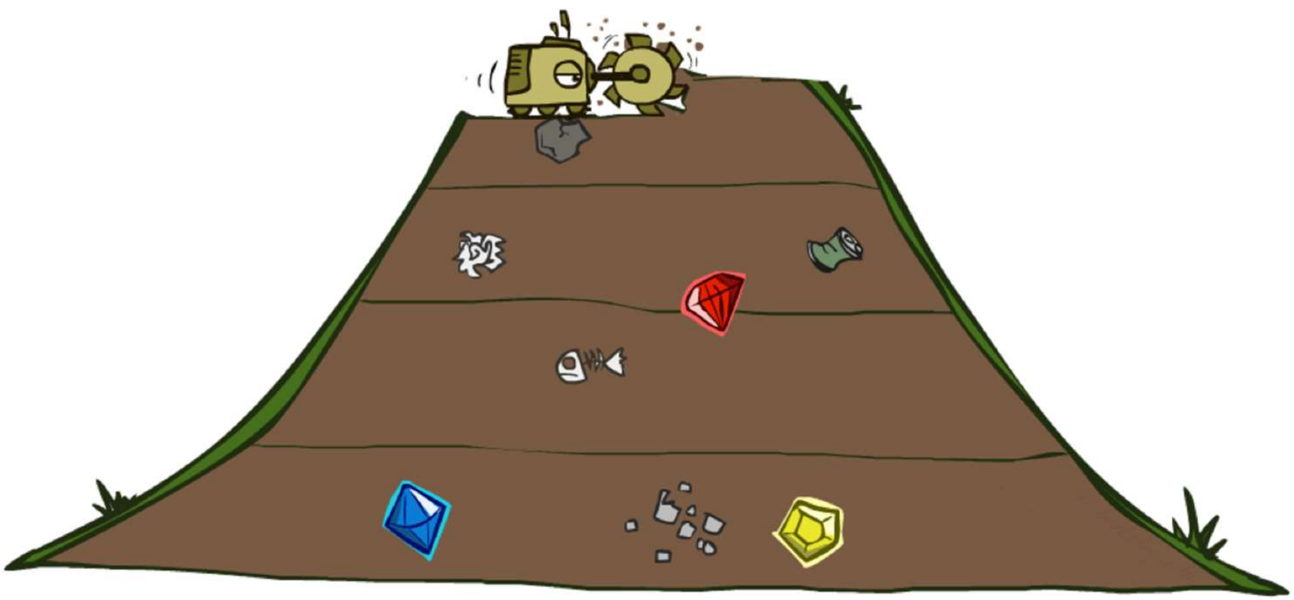
- Number of nodes in entire tree?
  - $1 + b + b^2 + \ldots. b^m = O(b^m)$

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



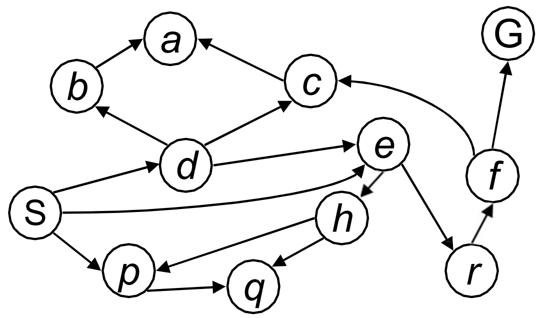
Breadth-First Search

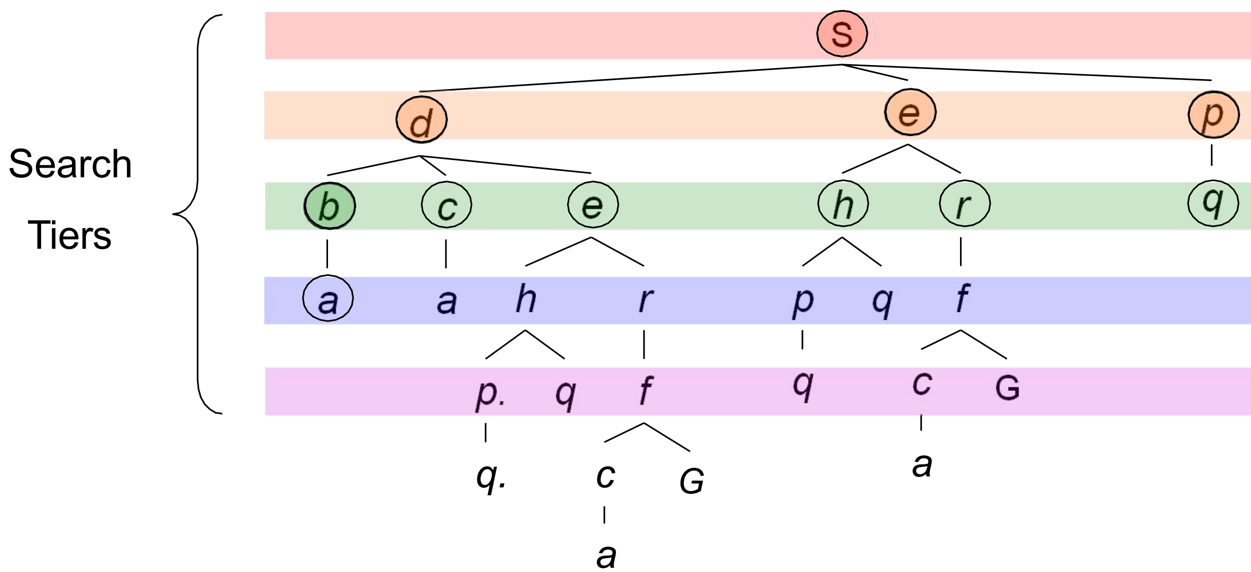


Breadth-First Search

Strategy: expand a shallowest node first

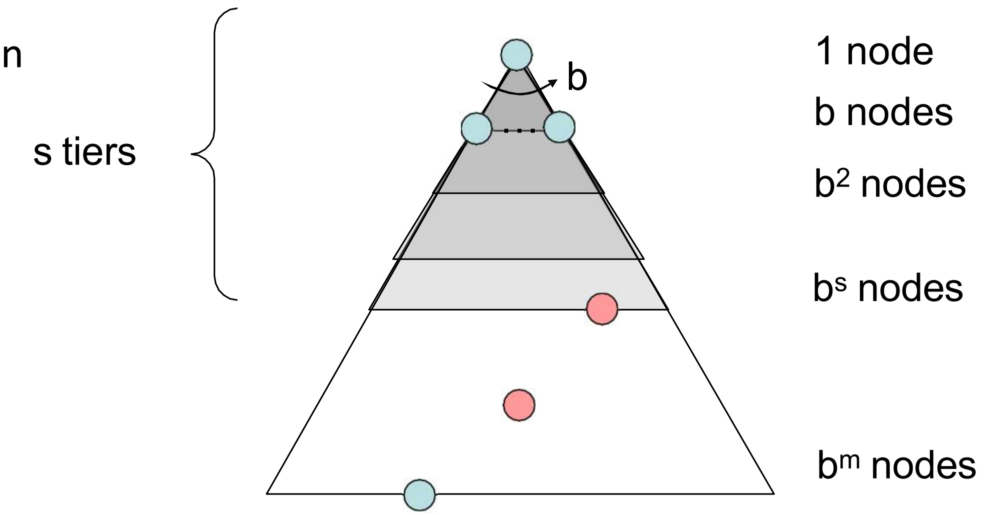
Implementation: Fringe is a FIFO queue



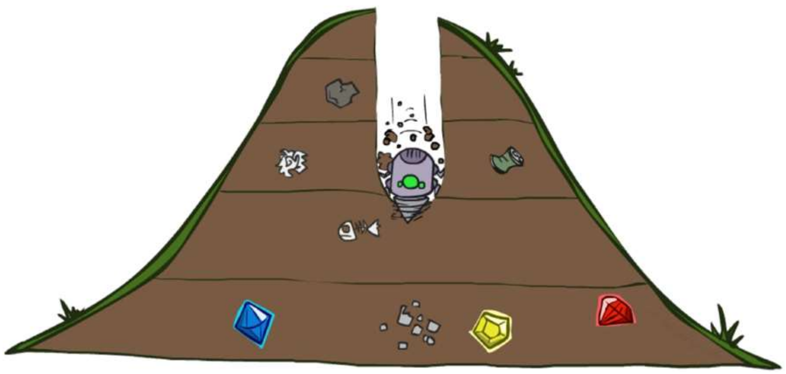


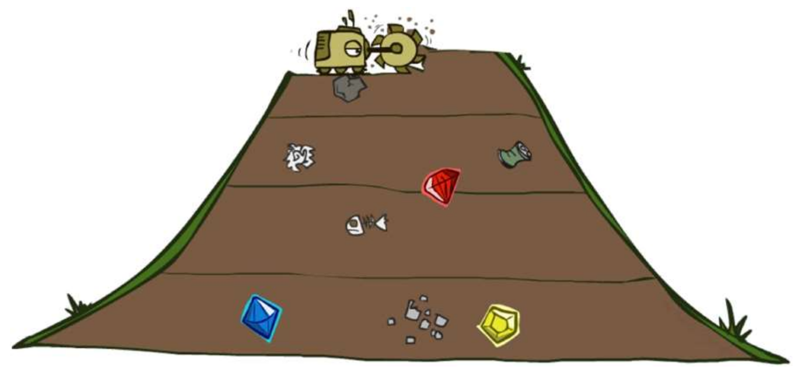
Breadth-First Search (BFS) Properties

- 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)



Quiz: DFS vs BFS





Quiz: DFS vs BFS

- When will BFS outperform DFS?

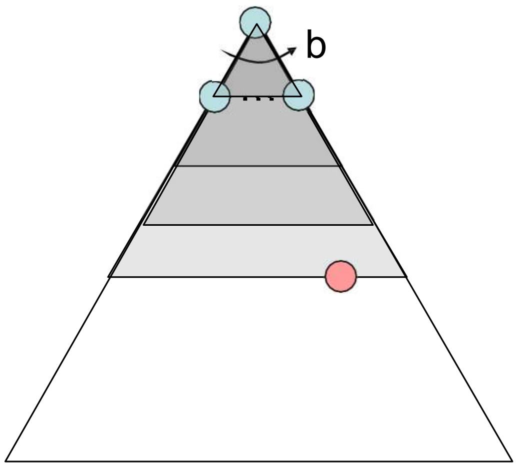
- When will DFS outperform BFS?

[Demo: dfs/bfs maze water (L2D6)]

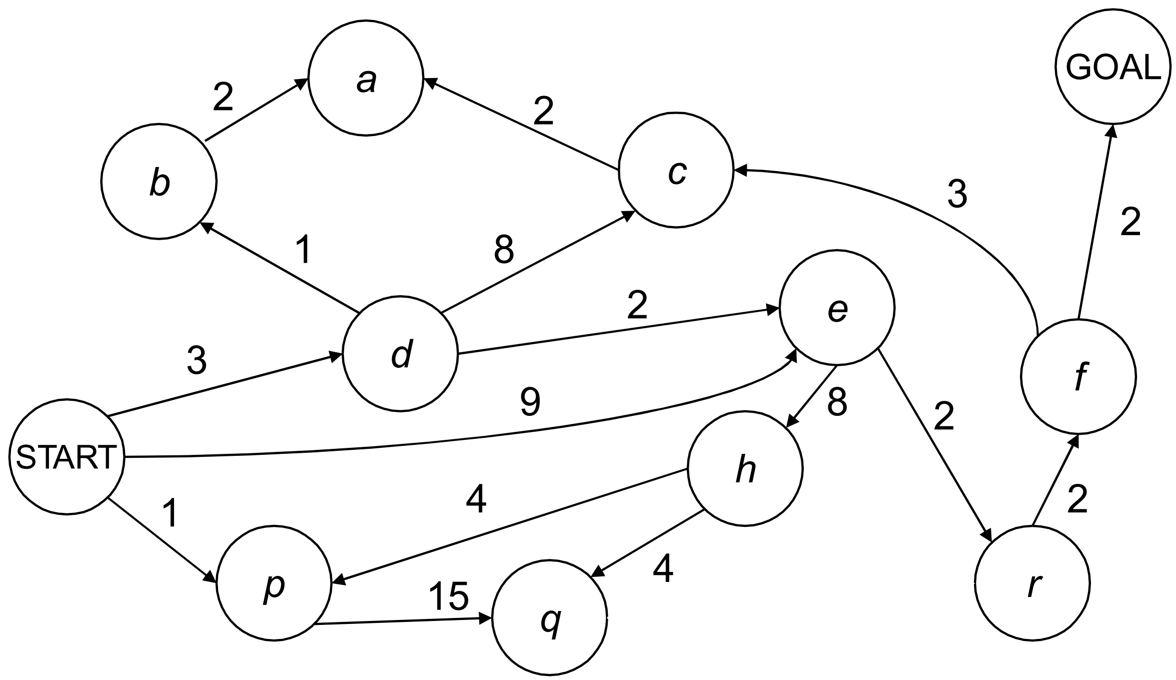
Iterative Deepening

- 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!

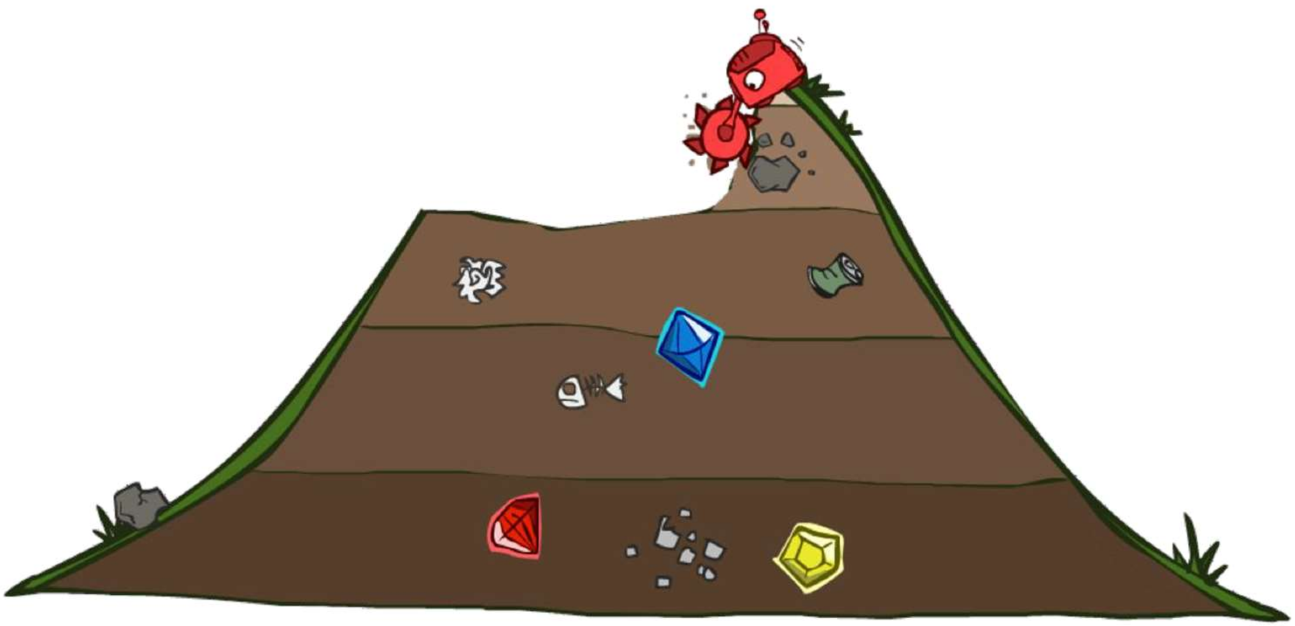


Cost-Sensitive Search



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.

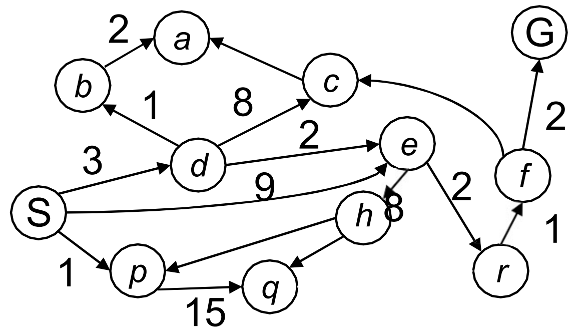
Uniform Cost Search



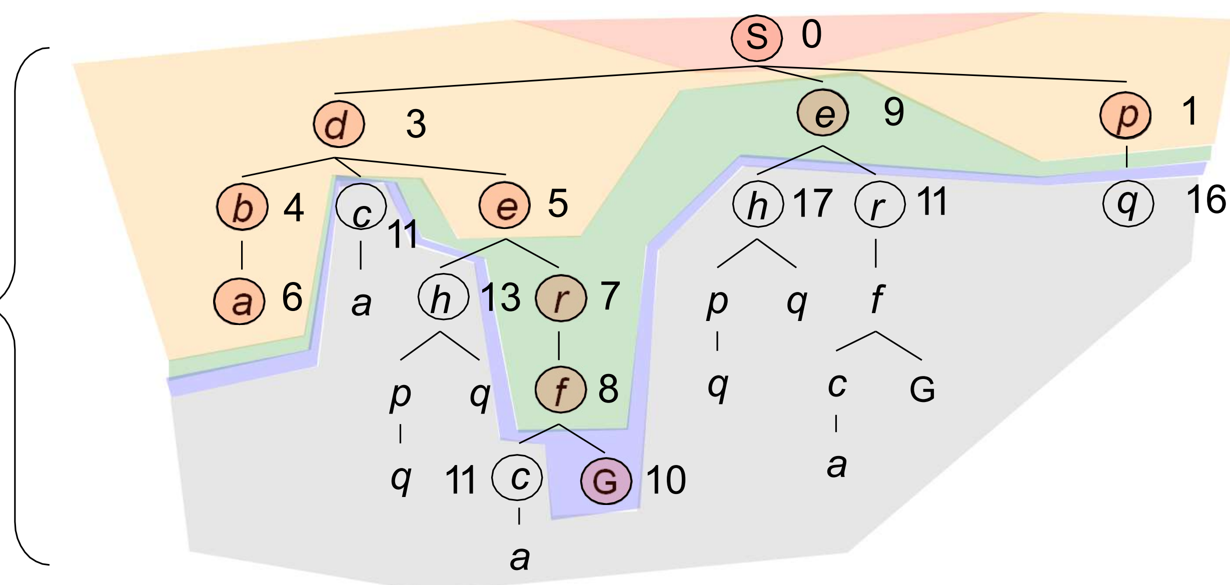
Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)

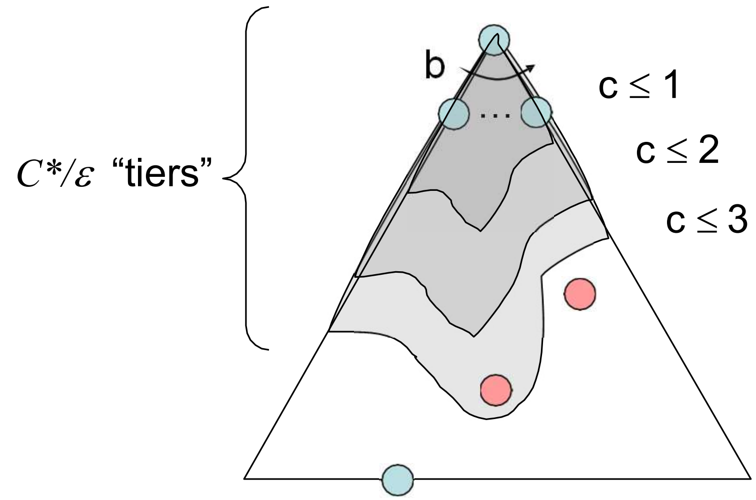


Cost contours



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*)



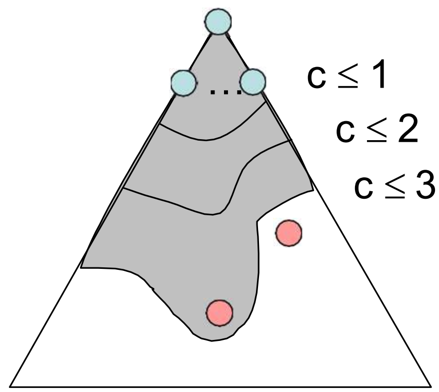
Uniform Cost Issues

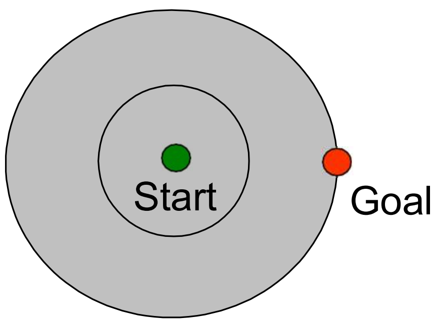
- Remember: UCS explores increasing cost contours

- The good: UCS is complete and optimal!

- The bad:
  - Explores options in every “direction”
  - No information about goal location

- We’ll fix that soon!





[Demo: empty grid UCS (L2D5)]
[Demo: maze with deep/shallow water DFS/BFS/UCS (L2D7)]

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