Search

Lecture 3, CMSC 170

John Roy Daradal / Instructor

Previously on CMSC 170

- **Rational agent**: maximize expected utility
- Reflex vs Planning Agent
- Types of Environments

Today's Topics

- Search Problems
- Tree Search
- Uninformed Search
  - Depth-First Search
  - Breadth-First Search
  - Uniform Cost Search

Brain: Decision-Making

- Planning vs Learning
- Simulation vs Memory

Planning Agents

- Agents that **plan ahead** to solve problems
- Thinks about **consequences** of its actions by performing **simulations**

Planning Agents

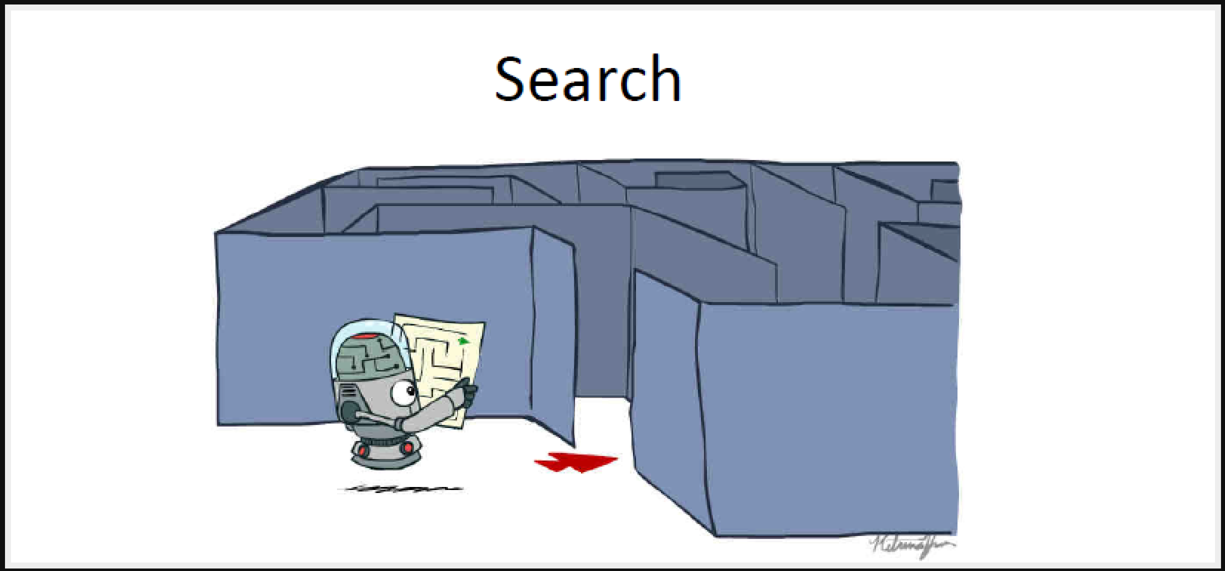
Goal-Based

- finds a solution that satisfies goals

Utility-Based

- find best possible solution

Planning and Search



Search Problem

- How to reach **goal** state from **start** state?
- **Solution**: *sequence of actions* (**plan**); start state → goal state
- **Complexity** comes from having many possible states (**large search space**)

Search Problem

State

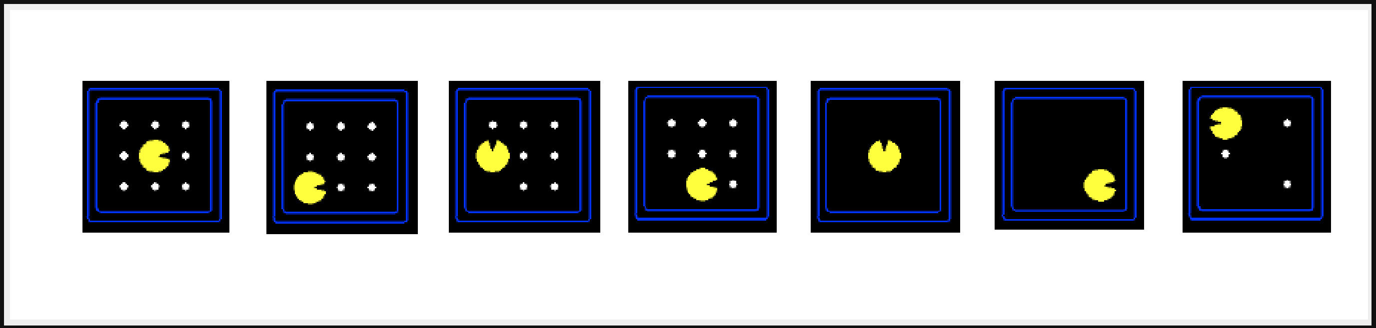
- encodes how the world is at a certain point

Start State

- initial configuration of the world

Search Problem

States



Search Problem

Actions

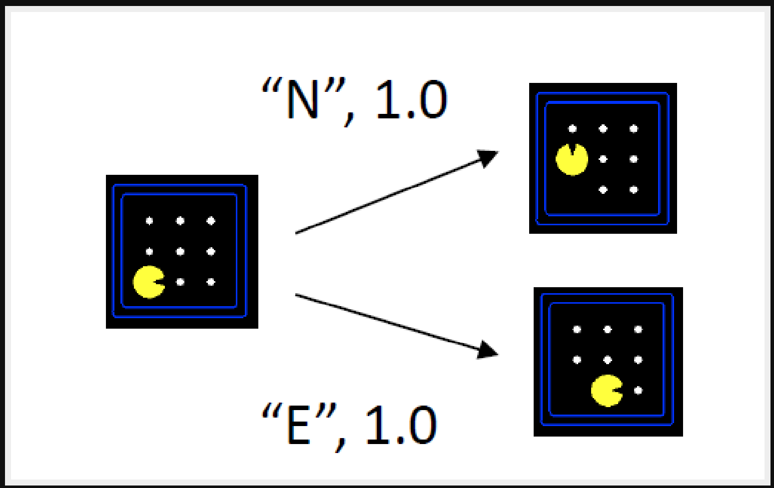
- valid steps the agent can perform

Successor Function

- models how the world works
- state $\longrightarrow$ (action,cost) $\longrightarrow$ state

Search Problem

Successor Function

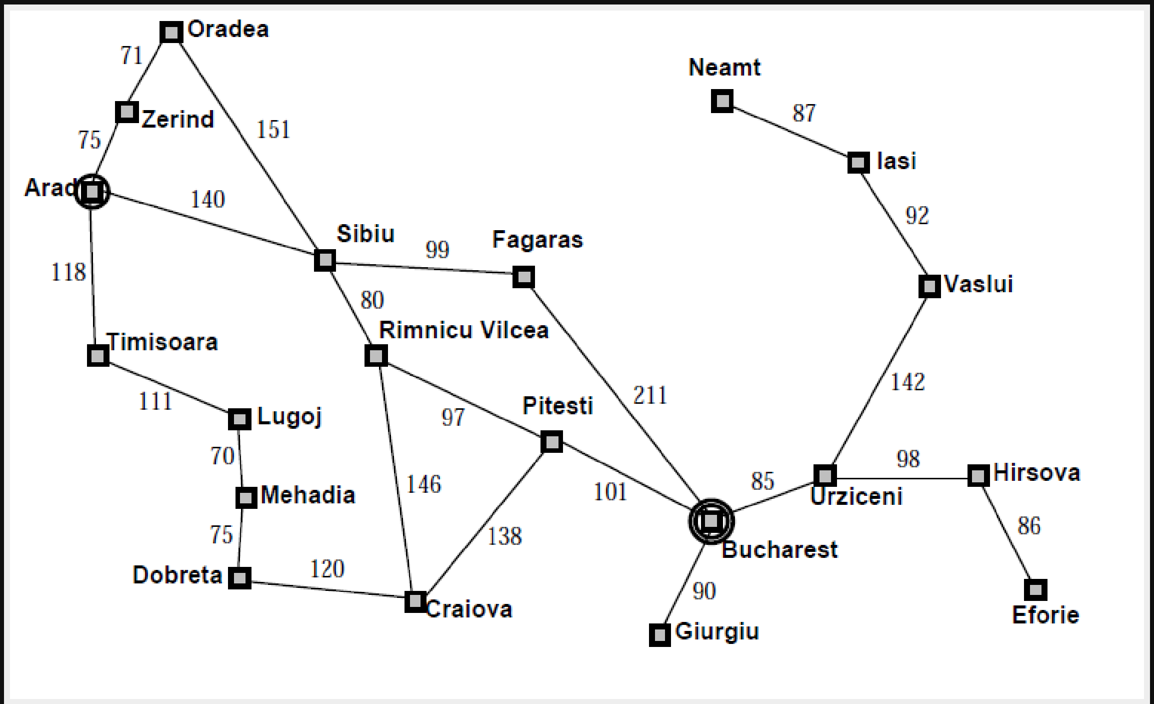


Search Problem

Goal Test

- checks if your **goal** has been *reached*
- not necessarily a goal state, could be a **description**

Romania Vacation



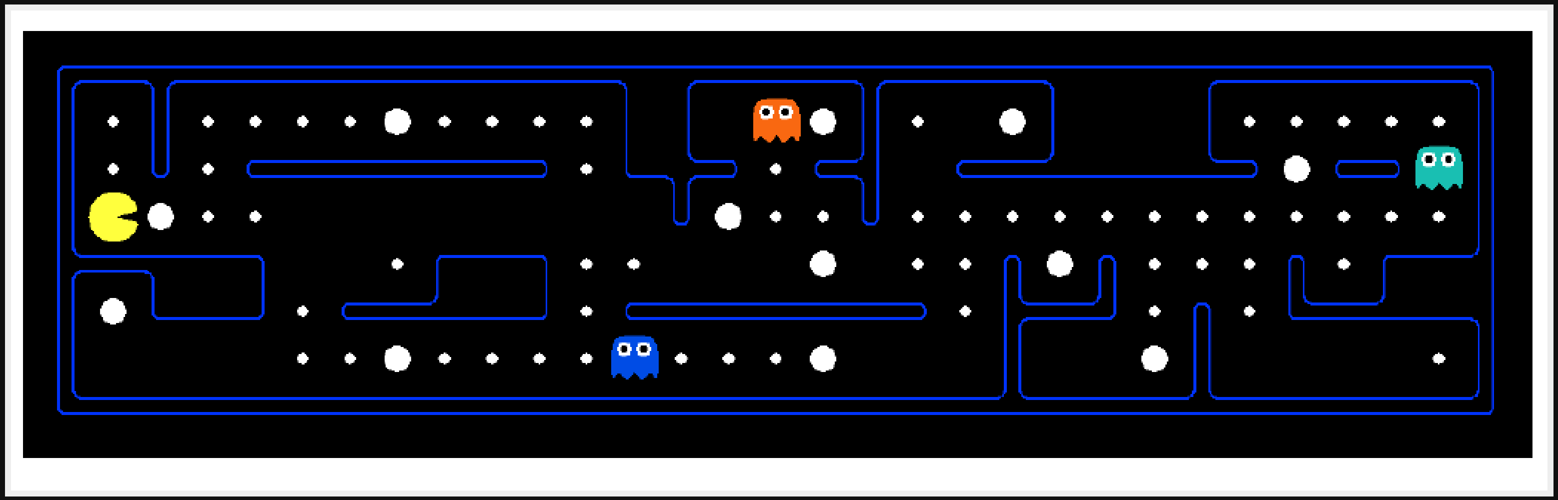
Romania Vacation

- **States**: cities
- **Successor Function**: roads
  (go to adjacent city, cost = distance)

Romania Vacation

- **Start state**: Arad
- **Goal test**: is state == Bucharest?
- **Solution**: path from Arad to Bucharest

Pacman



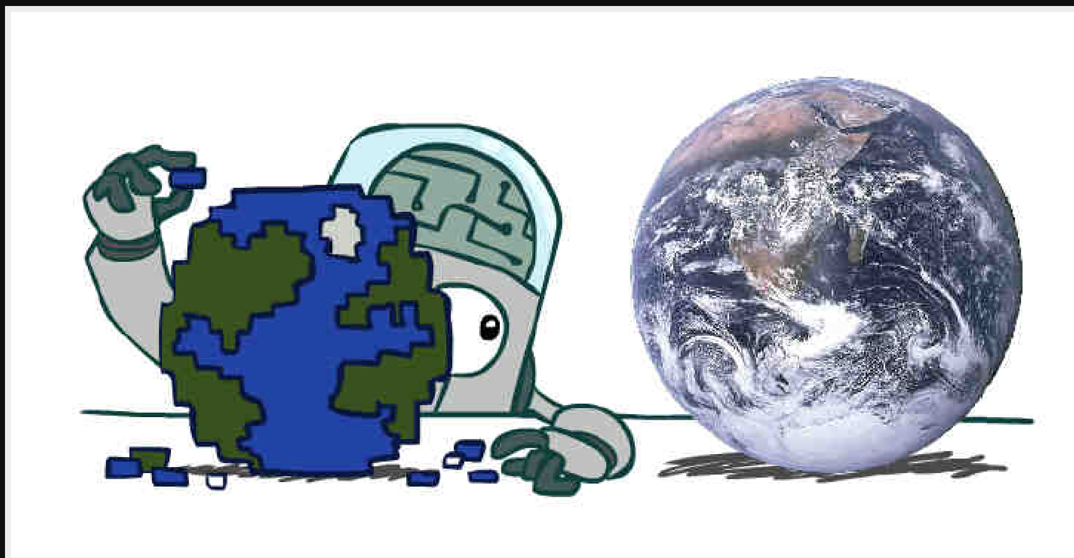
Pacman: Pathing

- **States**: (x,y) location
- **Actions**: NEWS
- **Successor**: update location
- **Goal test**: is (x,y) == END?

Pacman: Eat-All-Dots

- **States**: {(x,y), dot booleans}
- **Actions**: NEWS
- **Successor**: Update location, dot boolean
- **Goal test**: dots all false?

Search Problems are Models



Search Problems are Models

- Planning agent uses a **model** of the world
- **Search quality** (correctness, time, memory) is dependent on **model quality**

Model Quality

- **Too abstract**: not enough details, can't solve the problem
- **Too detailed**: deal with all complexities of the world, search will take too long

World State vs. Search State

World State

- includes all details of the environment
- don't model over this, usually very large

World State vs. Search State

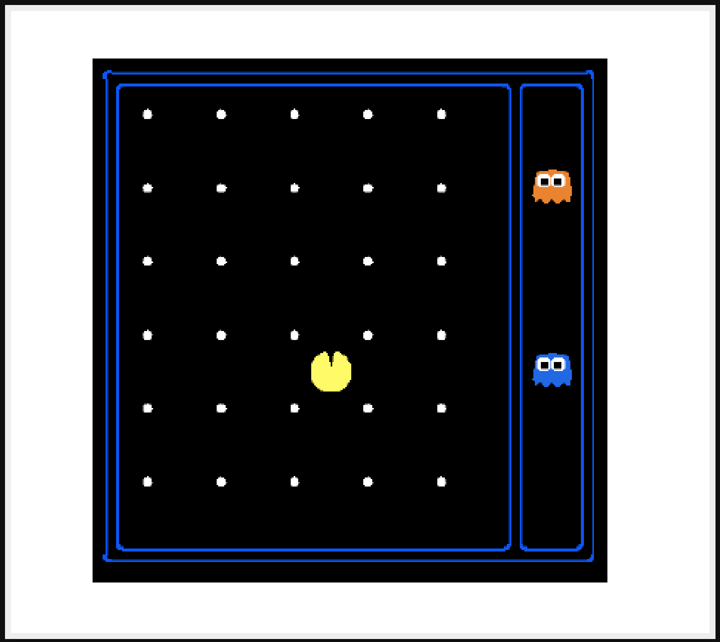
Search State

- only keeps details needed for planning (*abstraction*)
- depends on your search problem

Search Problem

- **State space**: start state + actions + successor function
- **Complexity** of search problem depends on **size** of state space

State Space Sizes



State Space Sizes

World state:

- Agent positions: 10 x 12 = 120
- Food count: 30
- Ghost positions: 12
- Agent Facing: 4 (NEWS)

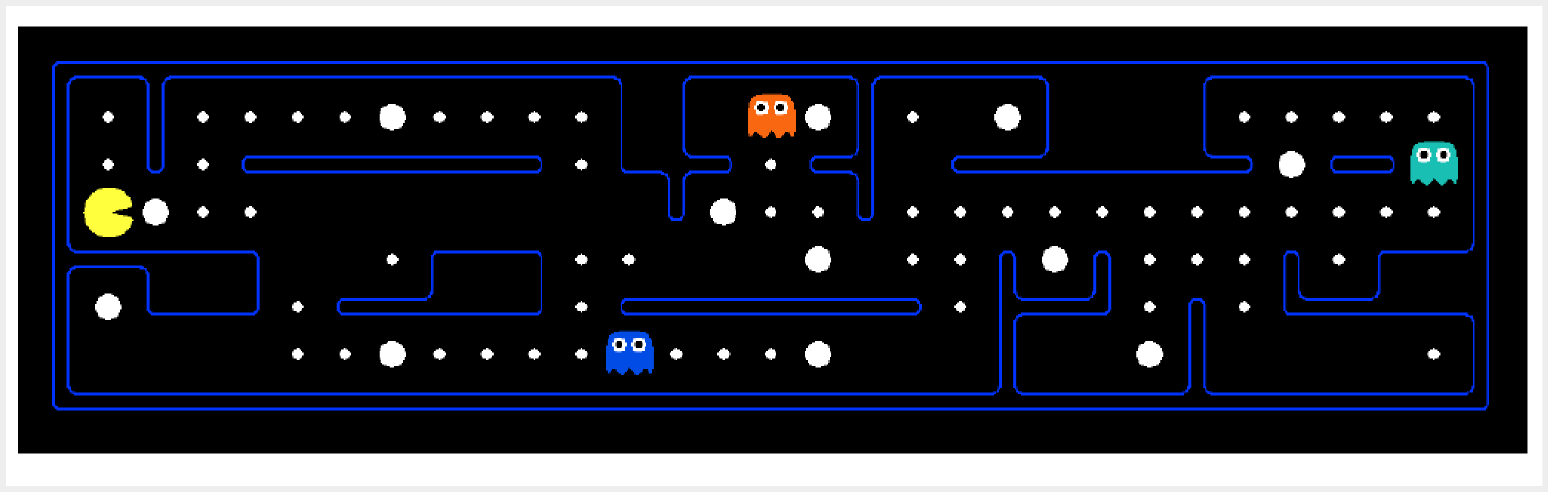
State Space Sizes

- World States = $120 * (2^{30}) * (12^{2}) * 4$
- States for pathing = 120
- States for eat-all-dots = $120 * (2^{30})$

State Space Sizes

"In general, search space is so large that you will never be able to enumerate it."

Exercise



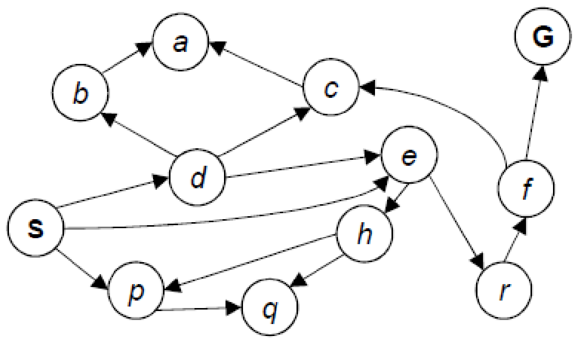
Exercise

- *Problem*: Eat all dots while keeping ghosts scared
- *Question*: What does the state space have to specify?

Answer

- Agent Position (x,y)
- Dot booleans
- Power pellet booleans
- Remaining scared time

State Space Graphs



Tiny search graph for a tiny search problem

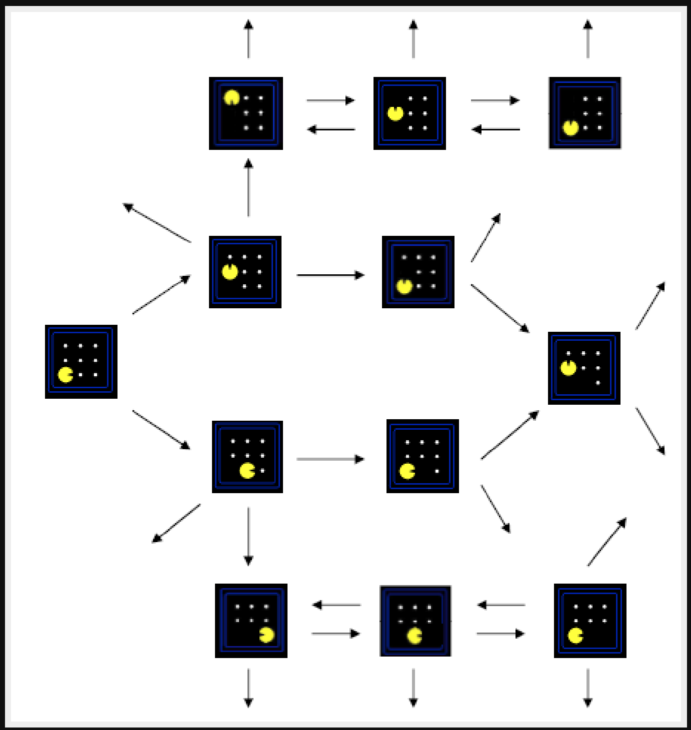
State Space Graphs

- **Nodes**: *abstracted* world configurations (*states*)
- The more you can **abstract**, the more **efficient** your search will be

State Space Graphs

- **Edges**: represent successors (action results)
- **Goal test**: set of *goal nodes* (maybe one)

State Space Graphs



State Space Graphs

- Each state occurs only *once*
- We can rarely build *full graph* in memory (*too big*)
- Most parts are **unreachable** during search

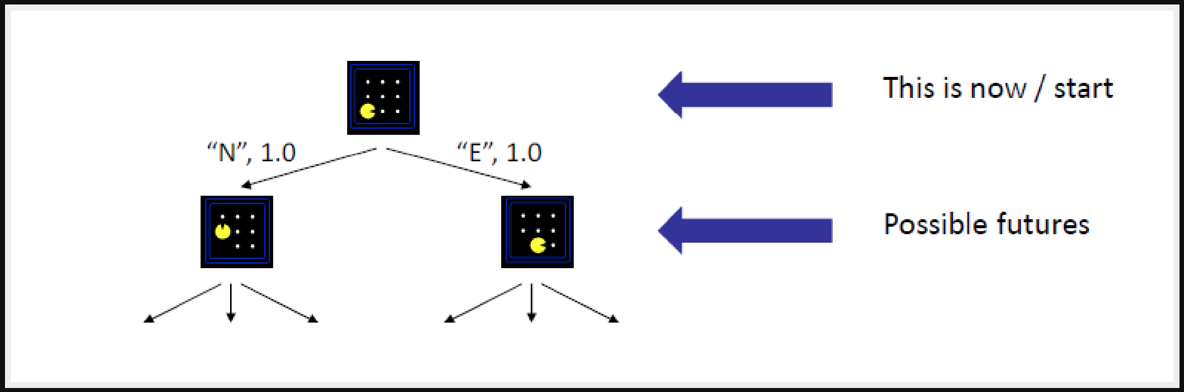
Search Trees

- Contains **start state** and things that can happen from it
- A "what if" tree of **plans** and their **outcomes**

Search Trees

- **Start state** is the *root node*
- *Children* correspond to **successors**

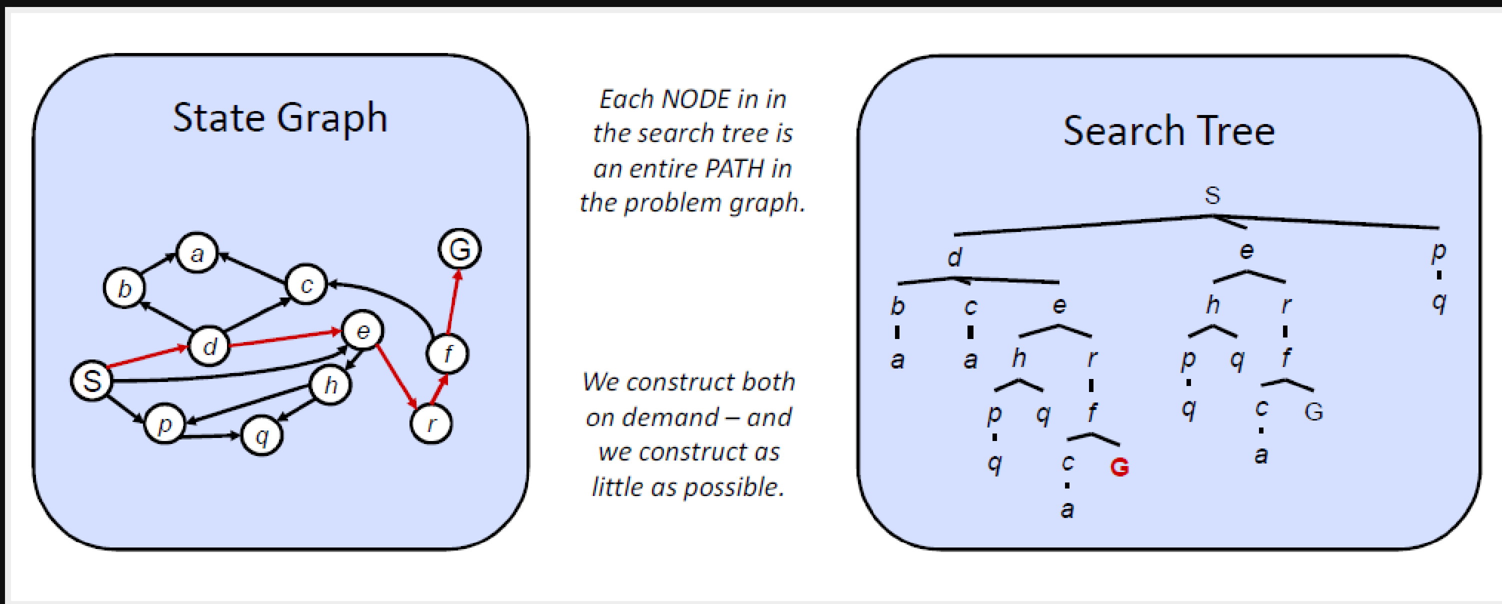
Search Trees



Search Trees

- Nodes show states, but correspond to **plans** that achieve those states
- For most problems, we can never actually build the *whole tree*

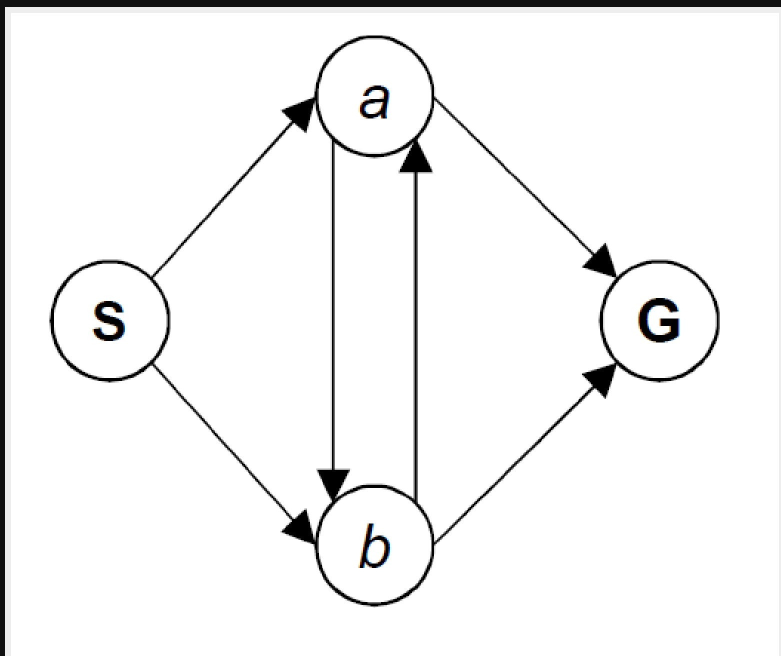
State Graphs vs. Search Trees



Exercise

State graph = 4 nodes

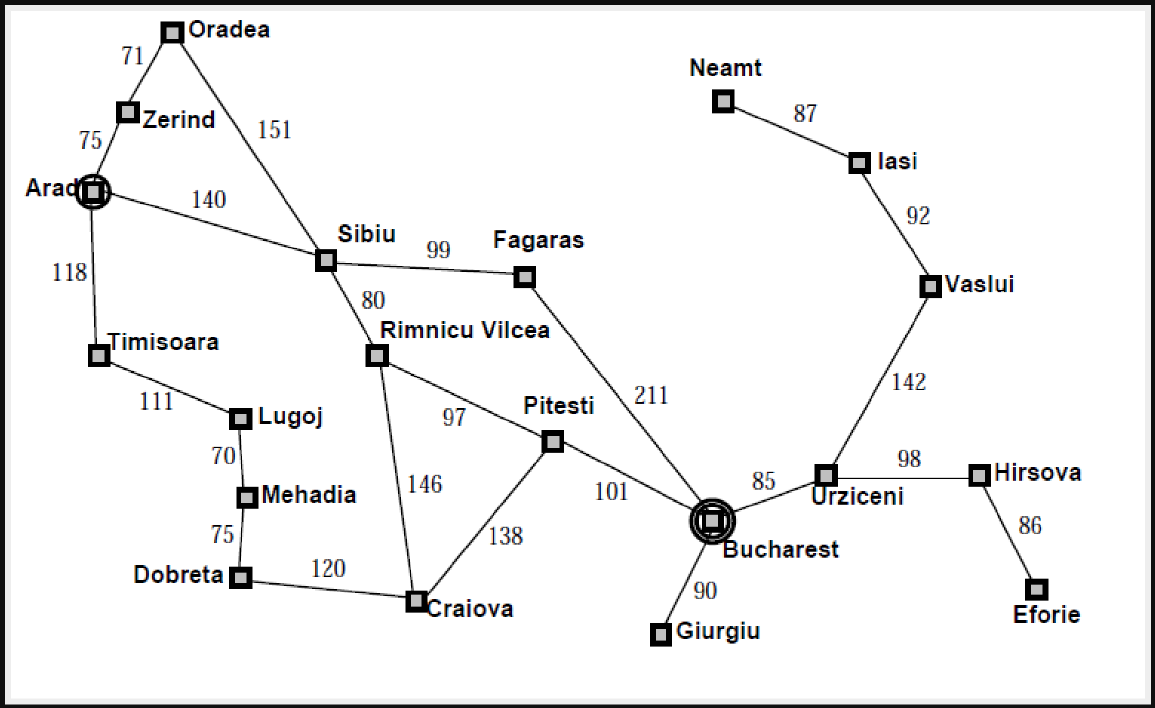
How big is the **search tree** (from S)?



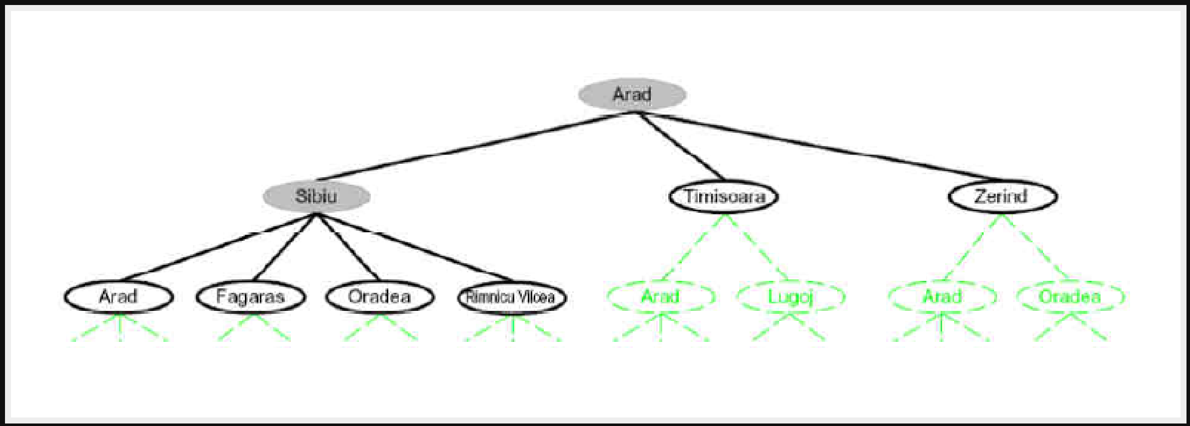
Answer

- ∞
- *Important*: Lots of **repeated** structure in the search tree

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

Basic Idea:

- **Offline, simulated** exploration of state space
- Generating **successors** of already-explored states (*expanding*)

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
```

General Tree Search

Important ideas:

- **Fringe**: all of the plans that may yet to work
- **Expansion**: picking something out of fringe and expanding
- **Exploration strategy**: which fringe nodes to explore next?

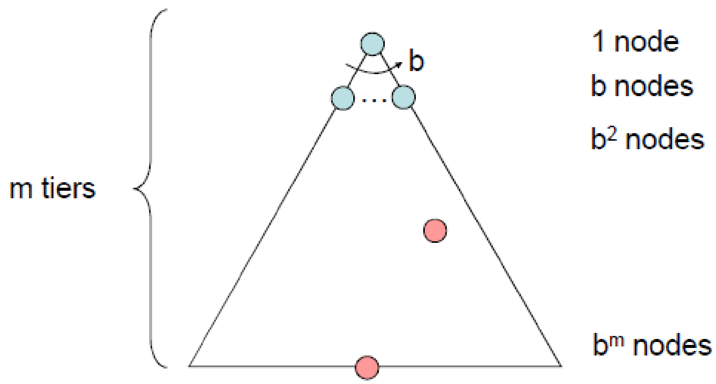
Uninformed Search

- Depth-First Search
- Breadth-First Search
- Uniform-Cost Search

Questions

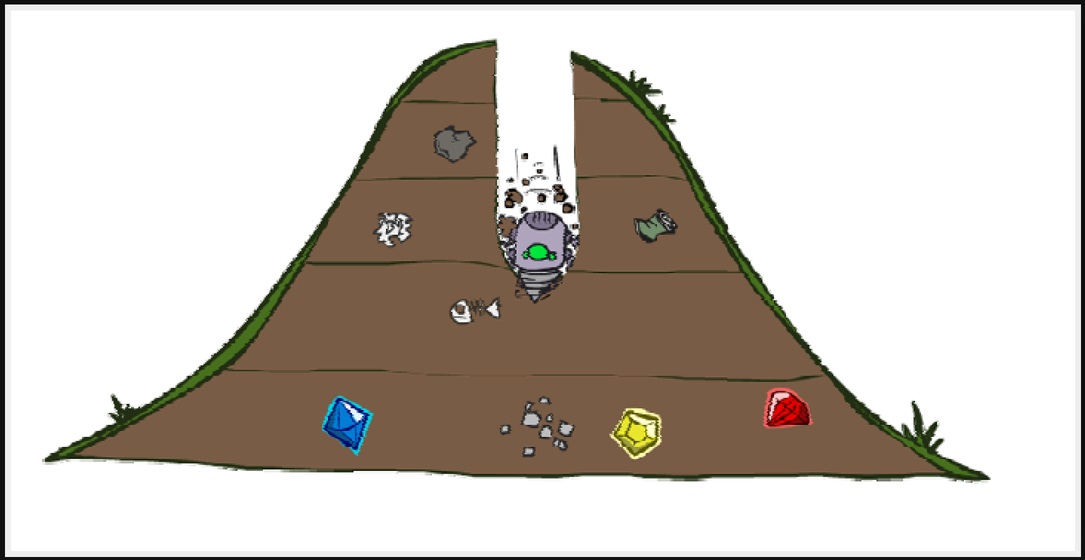
- **Complete**? Can it find solution?
- **Optimal**? Best solution?
- **Time** complexity?
- **Space** complexity?

Search Tree



- search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths
- Number of nodes in entire tree?
  - $1 + b + b^2 + ... b^m = O(b^m)$

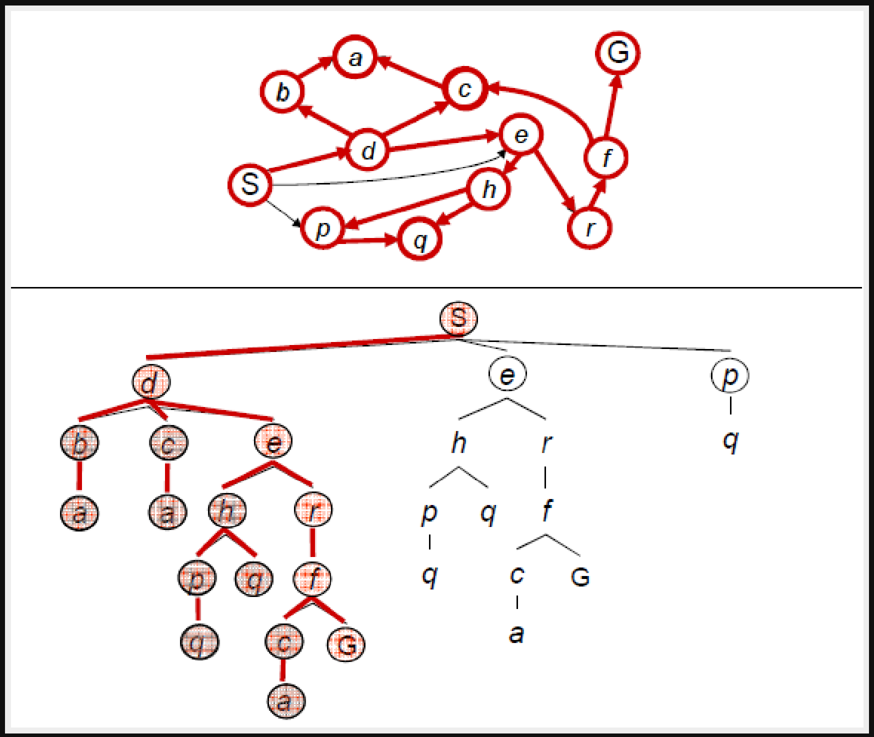
Depth-First Search



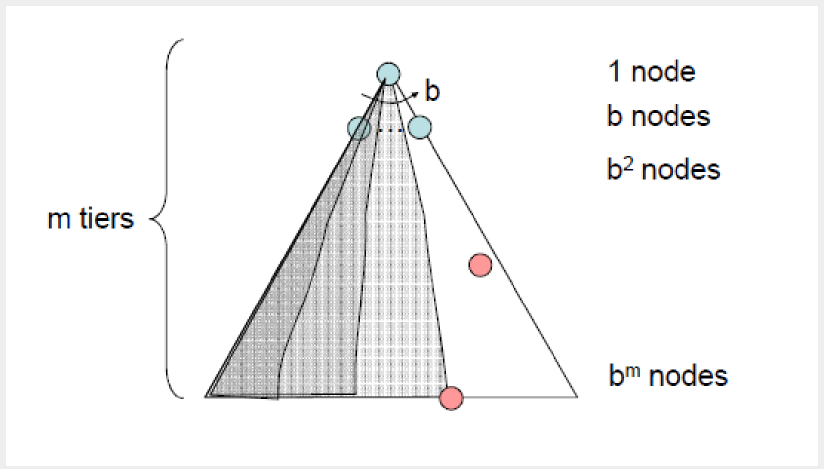
Depth-First Search

- **Strategy**: Expand a *deepest* node first
- **Implementation**: Fringe is a LIFO **stack**

Depth-First Search



DFS Properties



DFS Properties

What nodes does DFS **expand**?

- Some **left prefix** of the tree
- Could process the *whole* tree!
- If m is finite, takes **$O(b^m)$** *time* (*exponential*)

DFS Properties

How much **space** does *fringe* take?

- Only keeps *siblings* on path to root
- Takes **O(bm)** *space* (*polynomial*)

DFS Properties

Completeness

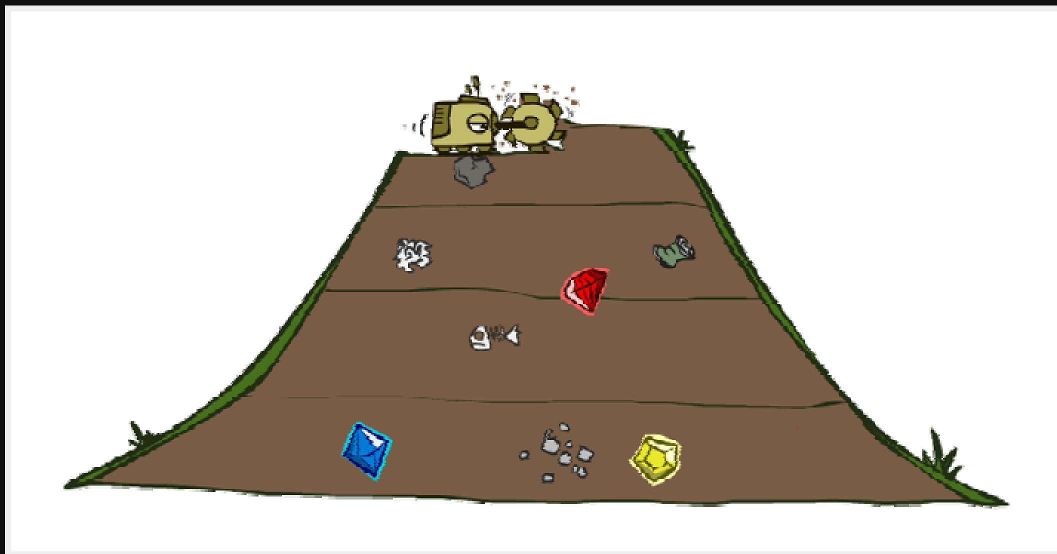
- m could be infinite (*cycles*)
- DFS is *complete* only if we **prevent cycles**

DFS Properties

Optimal

- not optimal
- finds the **leftmost** solution regardless of *depth* or *cost*

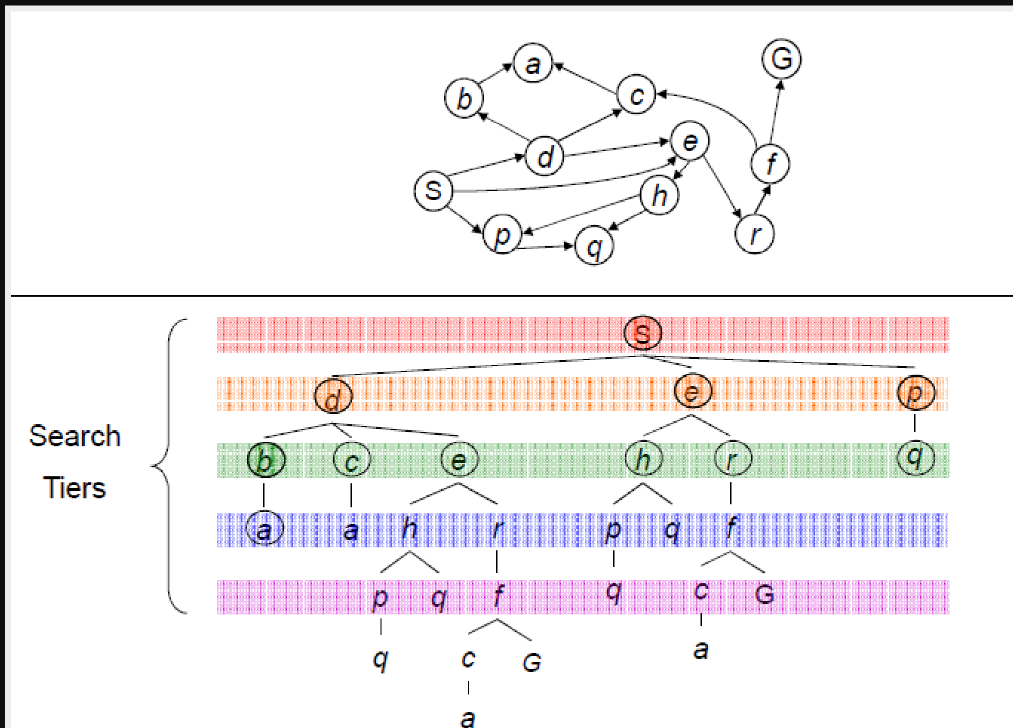
Breadth-First Search



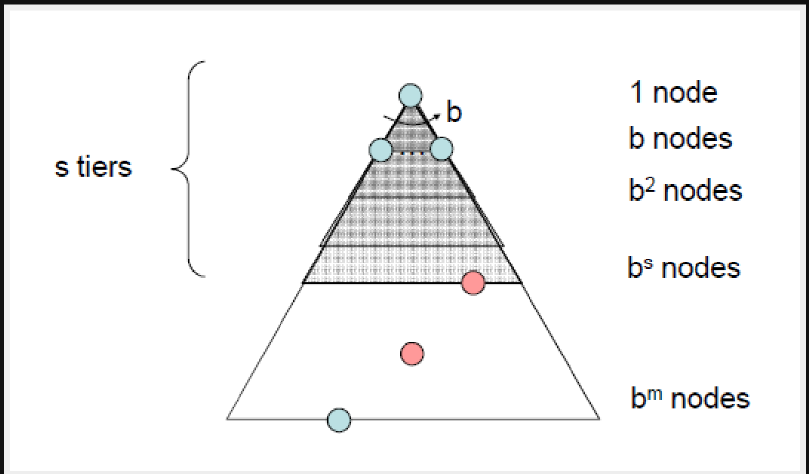
Breadth-First Search

- **Strategy**: Expand a *shallowest* node first
- **Implementation**: Fringe is a FIFO **queue**
- Search the tree in tiers / **layers**

Breadth-First Search



BFS Properties



BFS Properties

What nodes does BFS **expand**?

- Processes **all nodes** above *shallowest* solution
- Let depth of shallowest solution = s
- Search takes **$O(b^s)$** time (*exponential*)

BFS Properties

How much **space** does *fringe* take?

- Keeps *all nodes* from the last layer
- Takes **$O(b^s)$** *space* (exponential)

BFS Properties

Completeness

- s must be finite if a solution exists, so yes

Optimal

- only if costs are all 1
- solution deeper into the tree might cost less than shallowest solution

BFS

- **Space** is the big problem
- Can easily generate nodes at 100MB/sec
- 24 hrs = 8640GB

Question

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

Answer

- If solutions are relatively *shallow*, **BFS**
- If solutions are down at the *bottom*, **DFS**

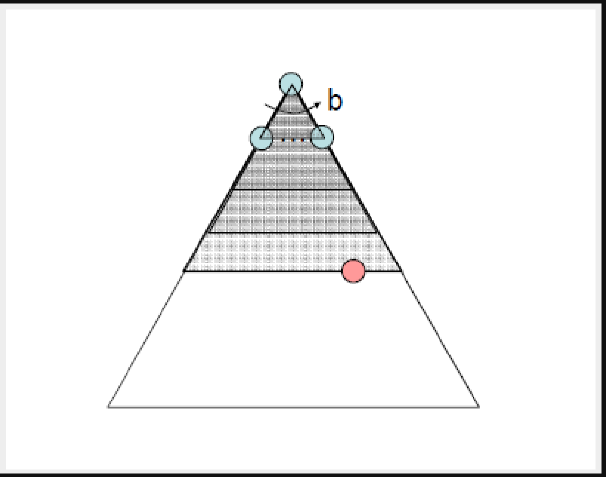
Iterative Deepening

- Best of both worlds!
- *Idea*: Combine DFS' **space** advantage with BFS' **time** / shallow-solution advantages

Iterative Deepening

- 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
- And so on..

Iterative Deepening



Iterative Deepening

- **RT**: $O(b^s)$
- **Space**: $O(bs)$
- **Complete**? Yes
- **Optimal**? Yes, if step cost = 1 (like BFS)

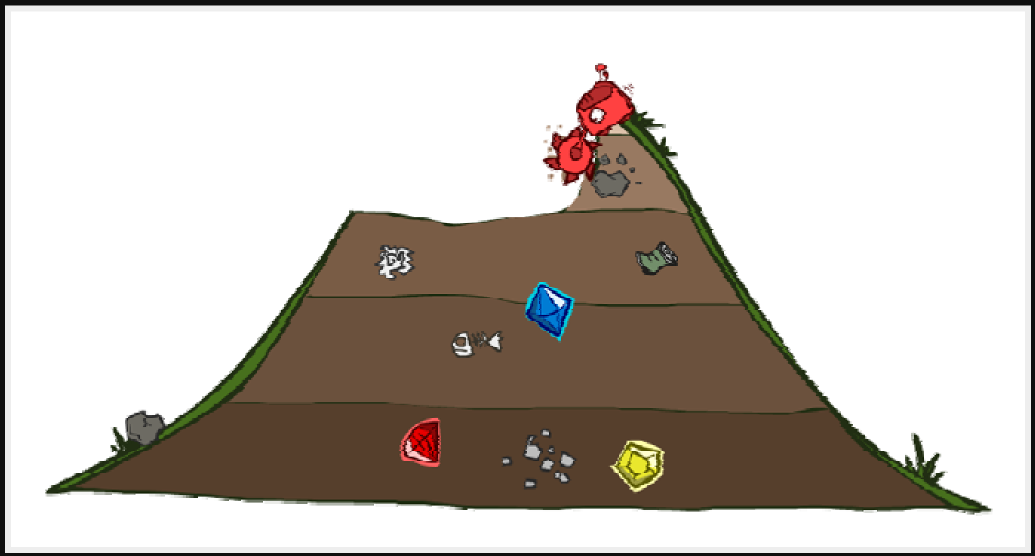
Iterative Deepening

- Doing DFS lots of times
- Isn't this wastefully redundant?
- Generally, most work happens in the lowest level searched, so **not so bad**!

Cost-Sensitive Search

- BFS finds *shortest path* in terms of **number of actions**
- It doesn't find the **least-cost path**

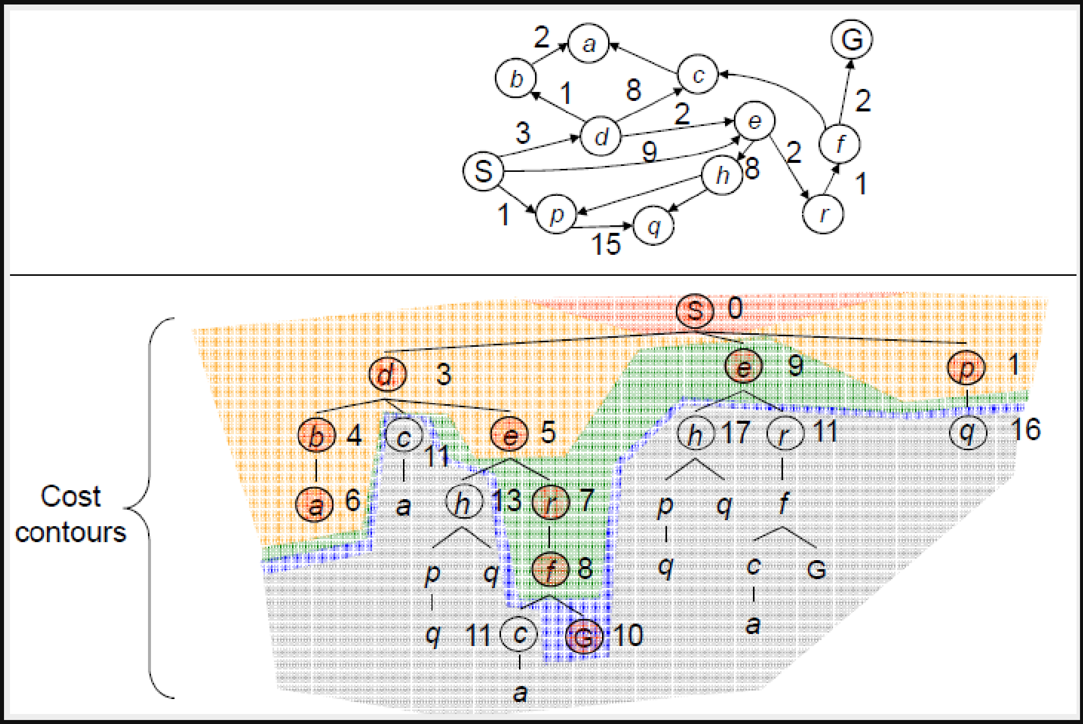
Uniform Cost Search



Uniform Cost Search

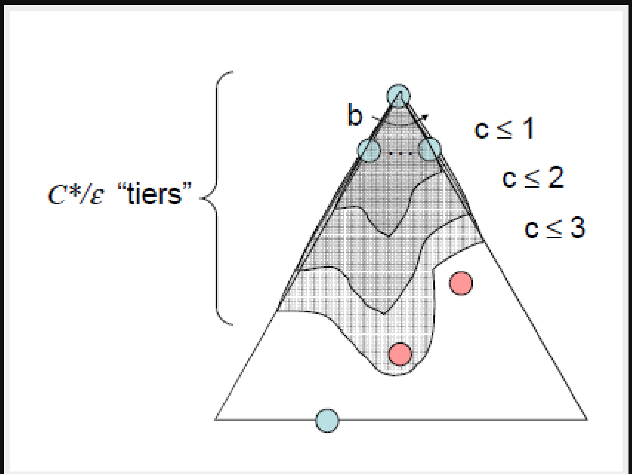
- **Strategy**: Expand a *cheapest* node first
- **Implementation**: Fringe = **priority queue** (priority: *cumulative cost*)

Uniform Cost Search



UCS Properties

C* = least-cost solution



UCS Properties

What nodes does UCS **expand**?

- Processes *all nodes* with cost less than cheapest solution
- Let solution cost = C*, arcs cost at least ε,
- Effective depth ~ roughly **C*/ε**
- **$O(b^{C^*/\varepsilon})$** *time* (*exponential* in effective depth)

UCS Properties

How much **space** does *fringe* take?

- Has roughly the last layer
- Takes $\mathbf{O(b^{C^*/\varepsilon})}$ *space* (exponential)

UCS Properties

Completeness

- Assuming best solution has finite cost and minimum arc cost is positive, yes!

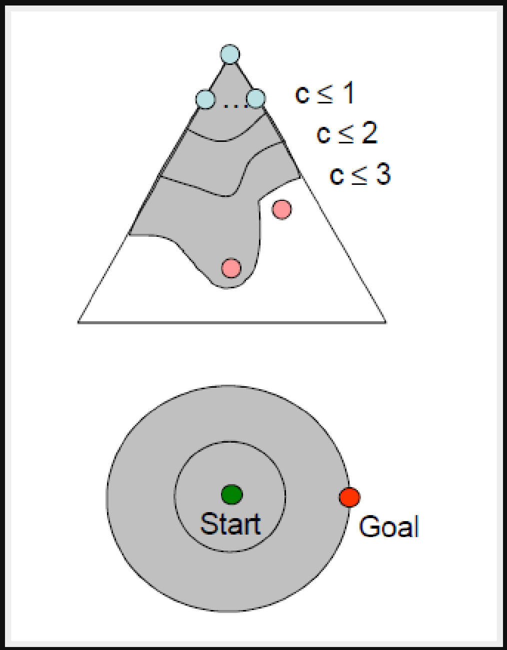
Optimal

- Yes!

UCS Issues

- UCS explores increasing cost contours
- **Good**: UCS is **complete** and **optimal**
- **Bad**: It explores options in *every direction*
- **No information** about *goal location* (**uninformed**)

UCS Issues



Search Algorithms

- Essentially the same except for **fringe** strategies
- **DFS**: Stack
- **BFS**: Queue
- **UCS**: Priority Queue

Comparison

	Finds	Time	Space	Fringe
DFS	Leftmost solution	$O(b^m)$	$O(bm)$	Stack
BFS	Shallowest solution	$O(b^s)$	$O(b^s)$	Queue
UCS	Least-cost solution	$O(b^{c^*/\varepsilon})$	$O(b^{c^*/\varepsilon})$	Priority Queue

Demo

- DFS
- BFS
- UCS

Search and Models

- Search operates over **models** of the world
- Agent doesn't actually try plans in real world
- Planning is all in **simulation**
- Search is *only as good as* your model

Search

Works when environment is:

- Fully observable
- Deterministic
- Discrete
- Benign
- Static

Summary

- **Search Problems**: states, actions, successor function, start state, goal test
- Search quality depends on **model quality**
- **Tree Search Algorithms**: DFS, BFS, UCS

Next Meeting

- Informed Search
- Heuristics
- Greedy Search
- A* Search
- Graph Search

Announcements

- *Assignment 2*: **Search**, next meeting
- *MP 1*: **Pacman Search**, next week

References

- *Artificial Intelligence: A Modern Approach, 3rd Edition*, S. Russell and P. Norvig, 2010
- CS 188 Lec 2 slides, Dan Klein, UC Berkeley

Questions?