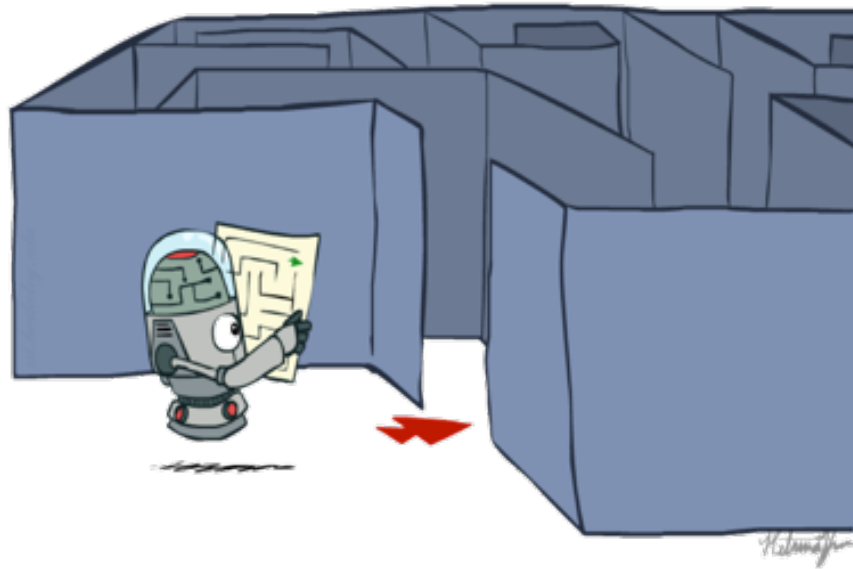


Artificial Intelligence

Search



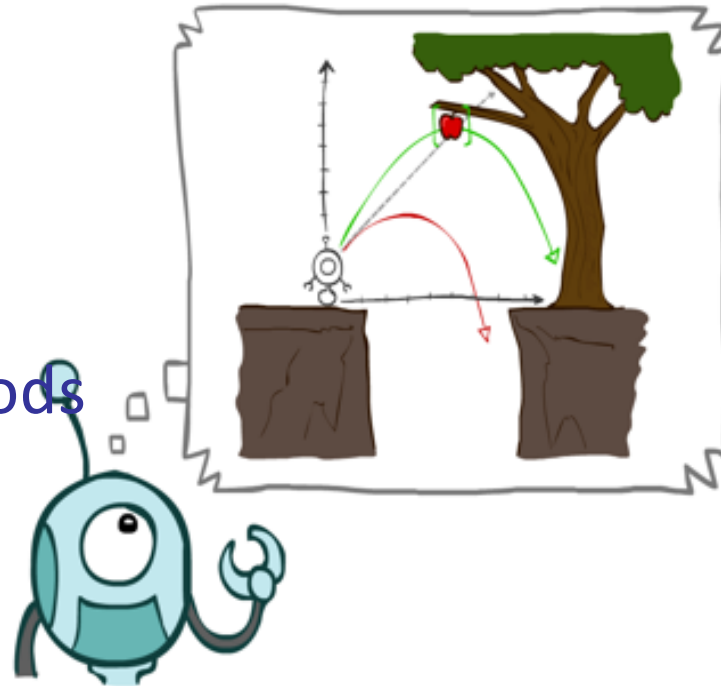
Instructor: David Suter

Course Delivered for Xidian University

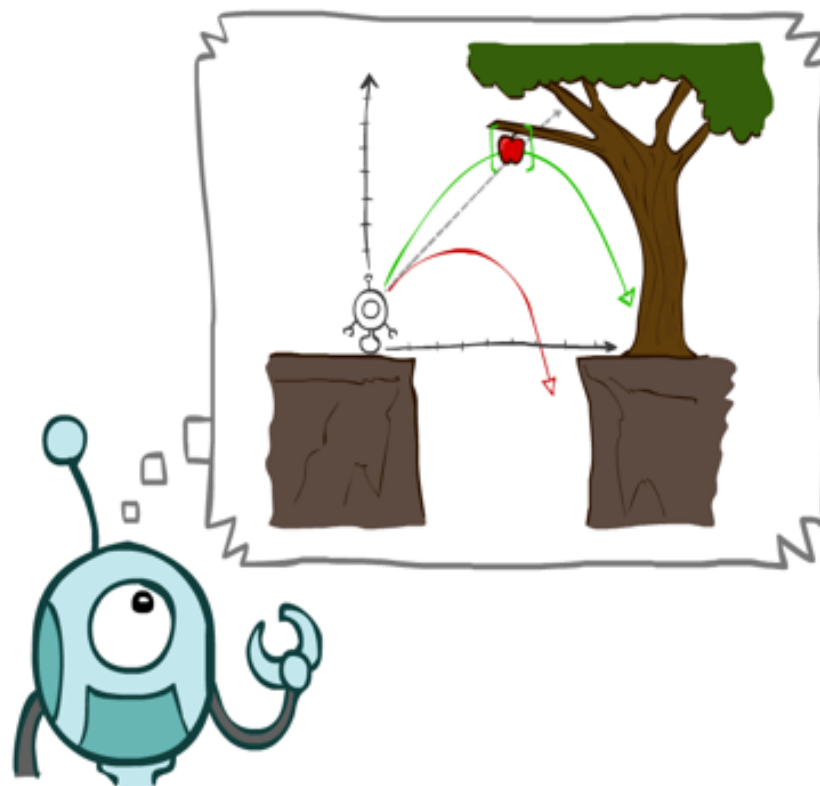
[Many slides adapted from those created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. Some others from colleagues at Adelaide University.]

Topics

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

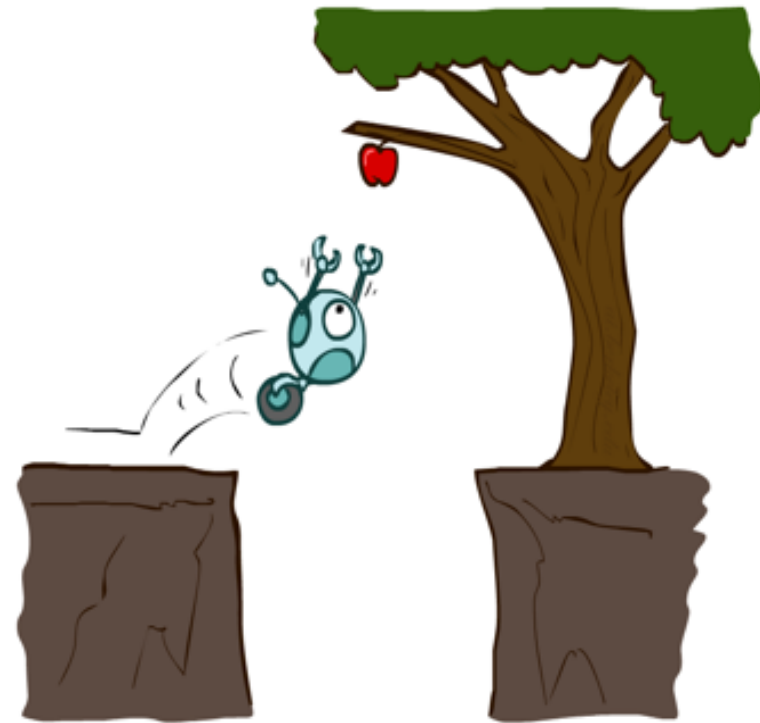


Agents that Plan



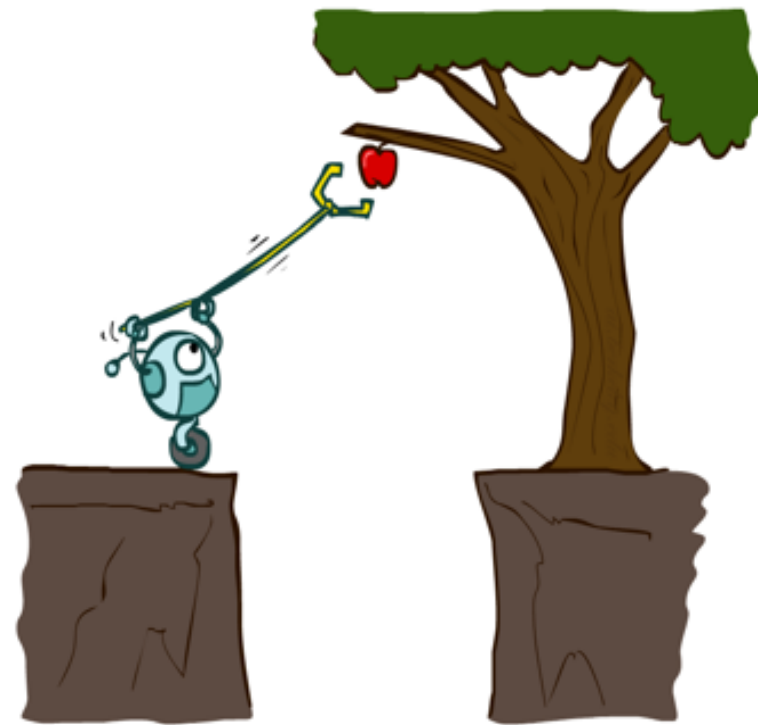
Reflex Agents

- 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?



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

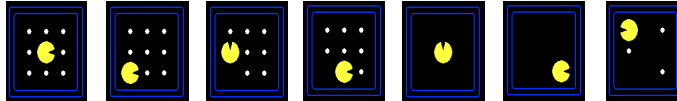


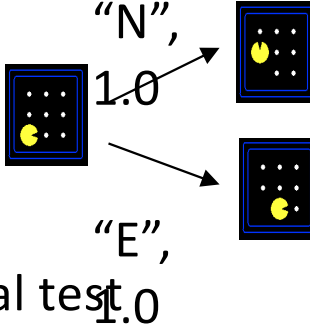
Search Problems



Search Problems

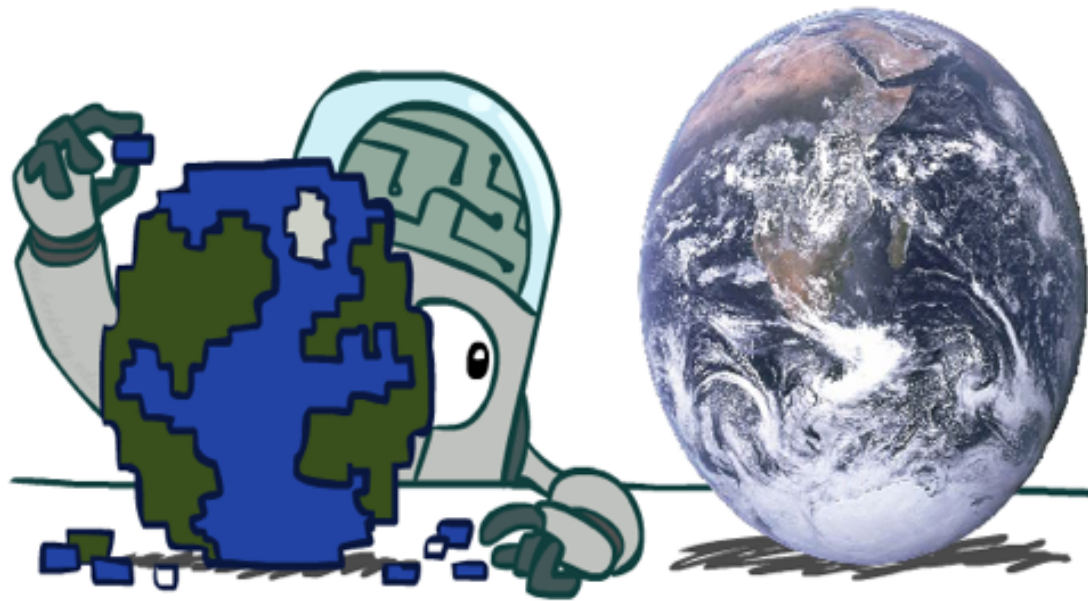
- A **search problem** consists of:

- A state space 

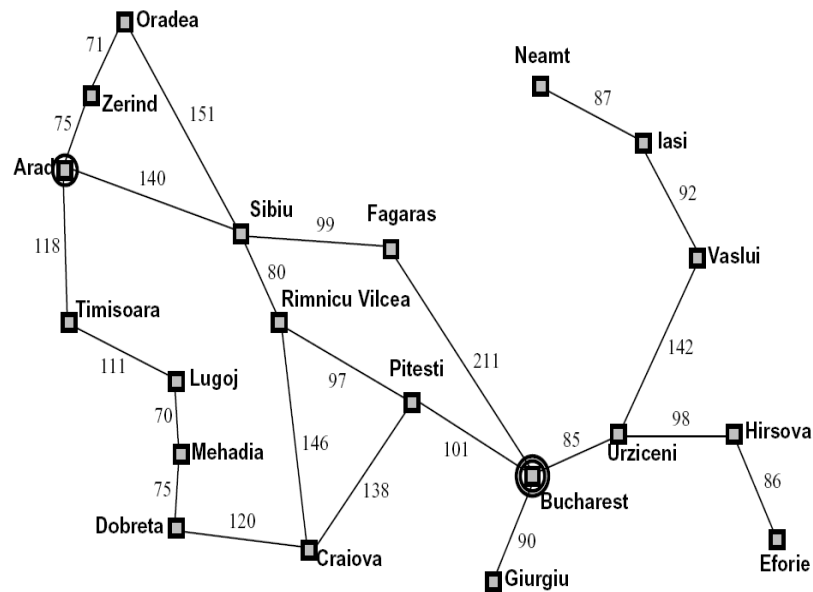
- A successor function (with actions, costs)


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

Search Problems Are Models



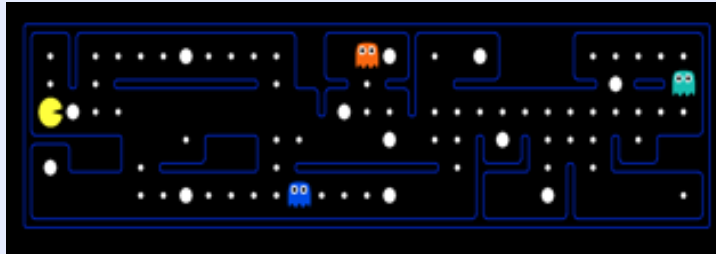
Example: Traveling in Romania



- 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: Path Finding

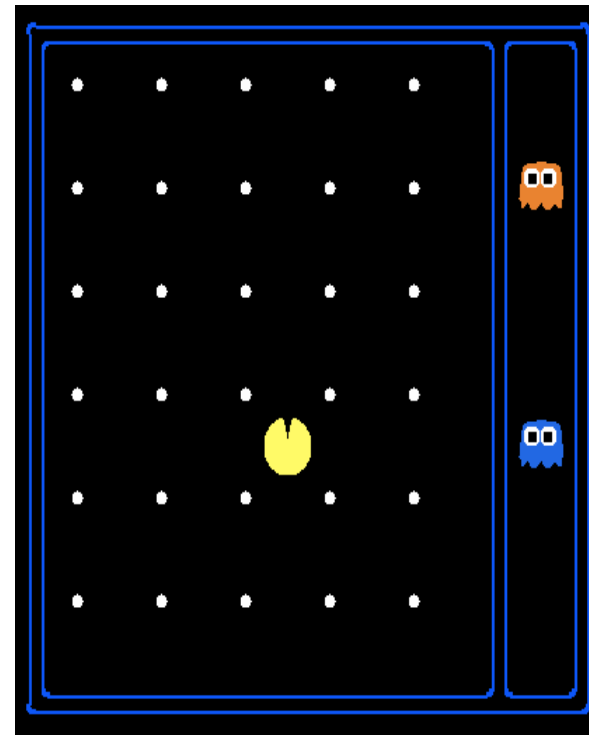
- States: (x,y) location
- Actions: NSEW
- Successor: update location only
- Goal test: is $(x,y)=\text{END}$

■ Problem: Eat-All-Dots

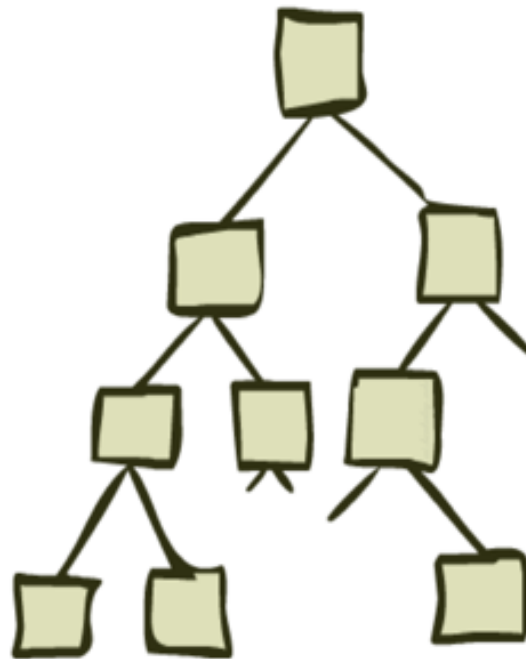
- States: $\{(x,y), \text{dot booleans}\}$
- Actions: NSEW
- Successor: update location and possibly a dot boolean
- Goal test: dots all

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?
 $120 \times (2^{30})$

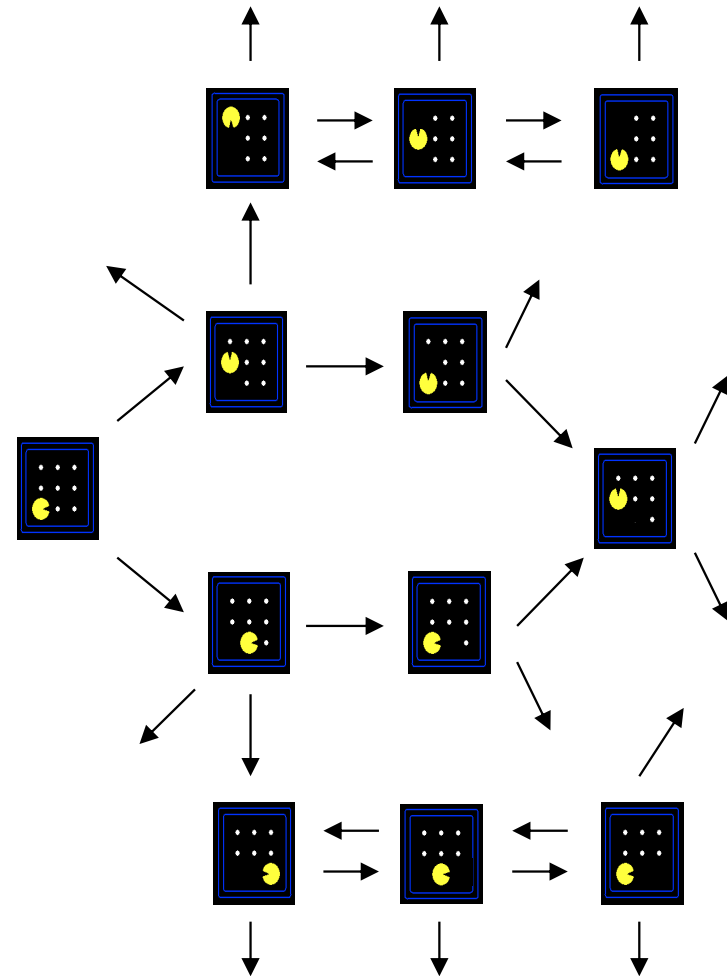


State Space Graphs and Search Trees



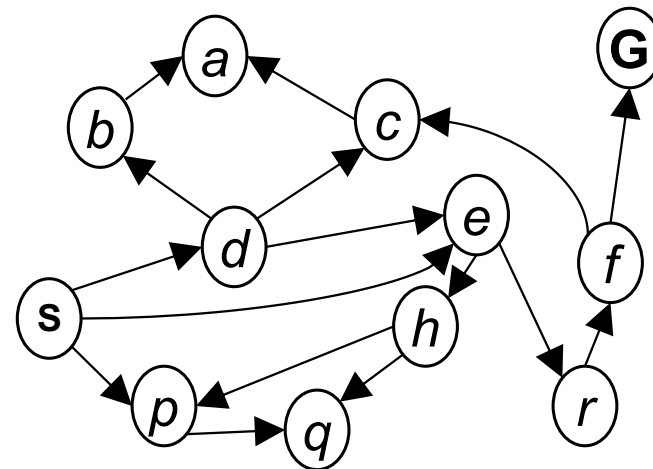
State Space Graphs

- 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



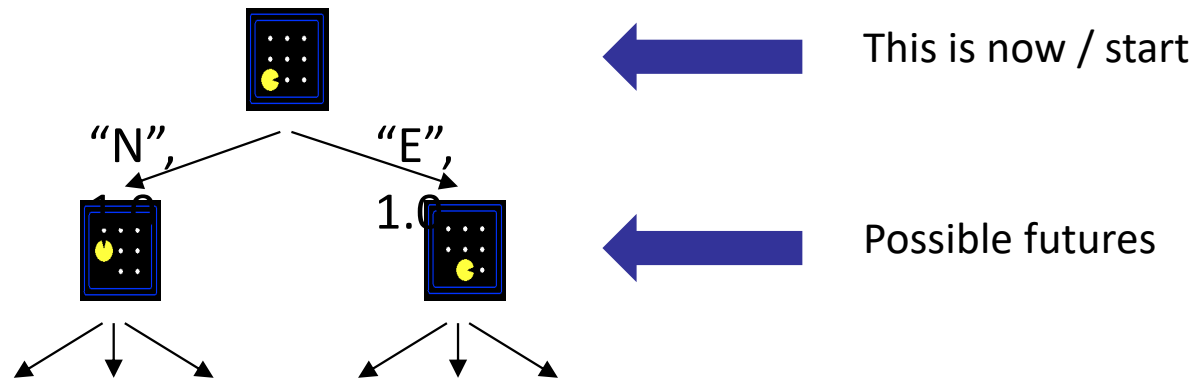
State Space Graphs

- 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 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*

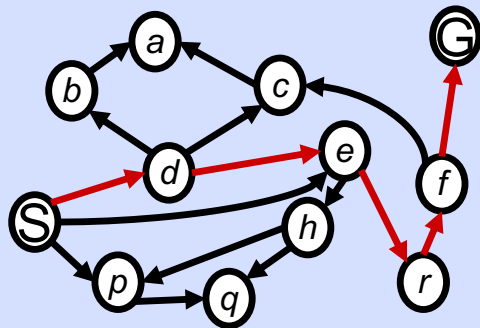
Search Trees



- 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 that achieve those states
 - For most problems, we can never actually build the whole tree

State Space Graphs vs. Search Trees

State Space Graph

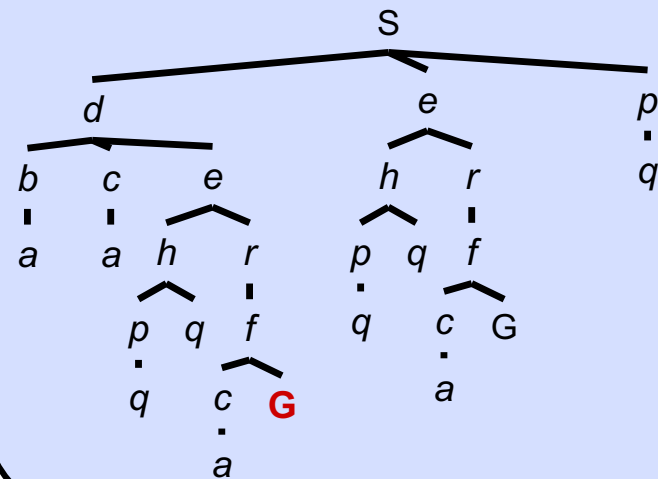


Each NODE
in the
search tree
is an entire
PATH in the
state space
graph.

we
construct
both on
demand –
and we
construct as
little as
little as

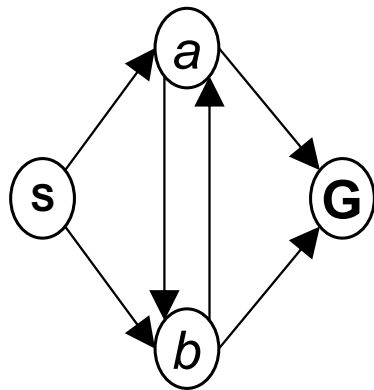
...

Search Tree



Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:

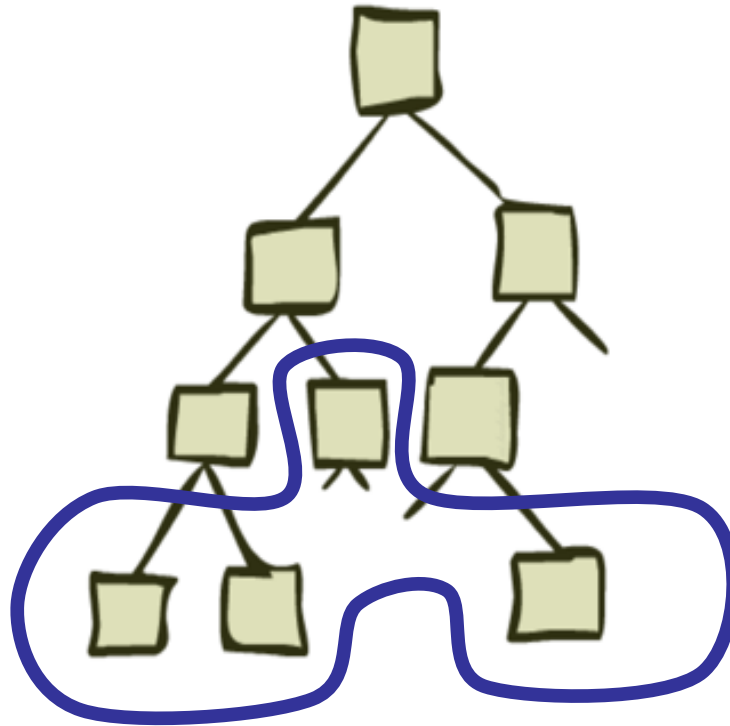


How big is its search tree (from S)?

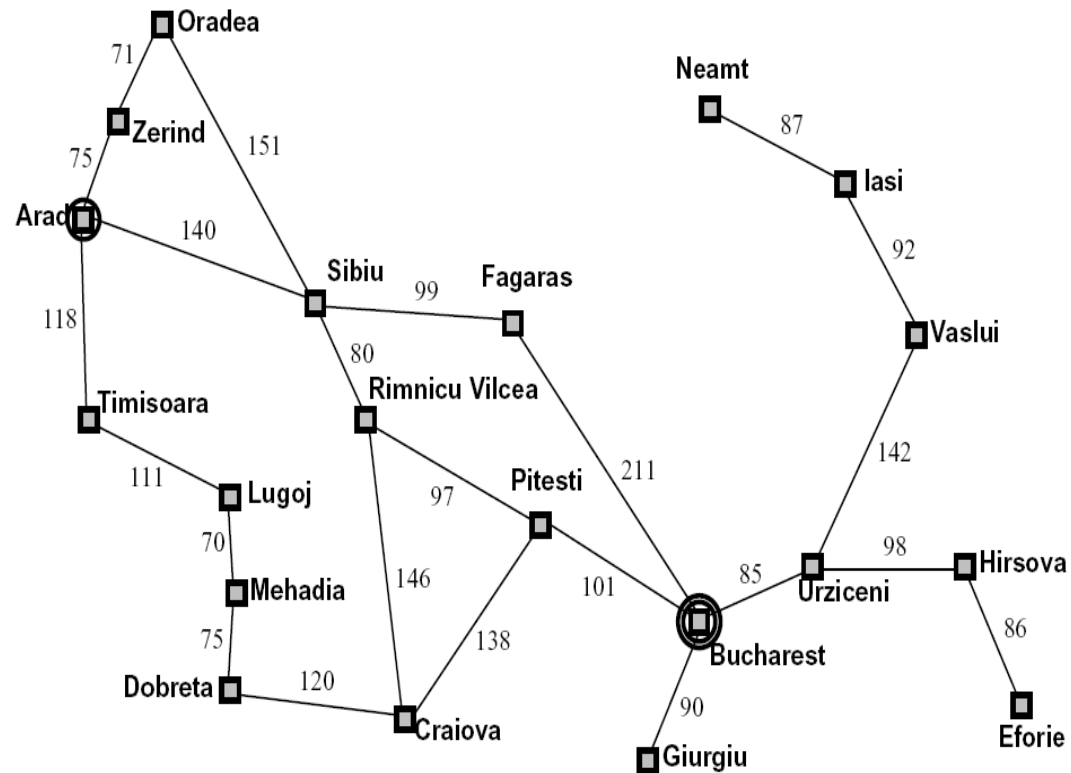


Important: Lots of repeated structure in the search tree!

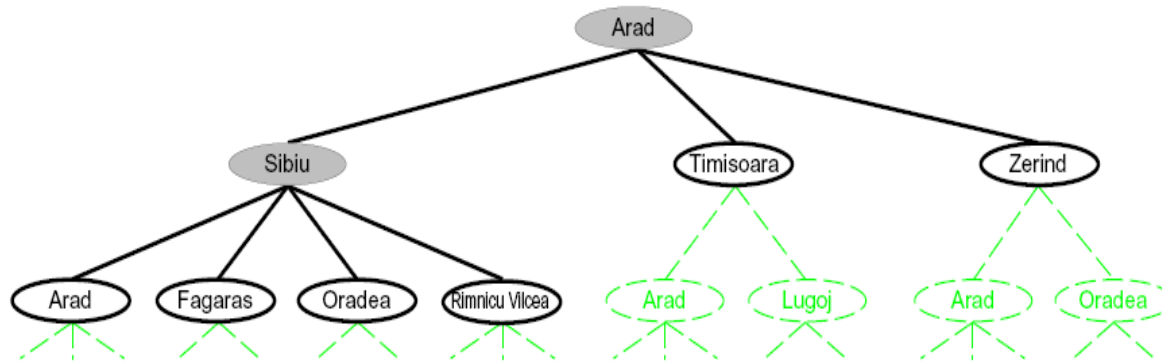
Tree Search



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
```

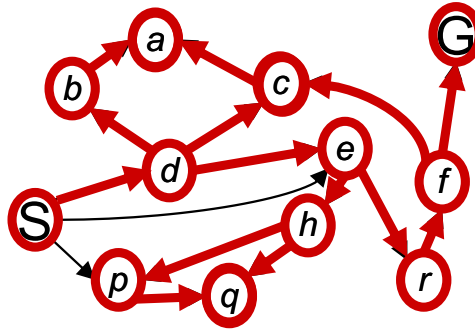
- Important ideas:
 - Fringe
 - Expansion
 - Exploration strategy
- Main question: which fringe nodes to explore?

Depth-First Search

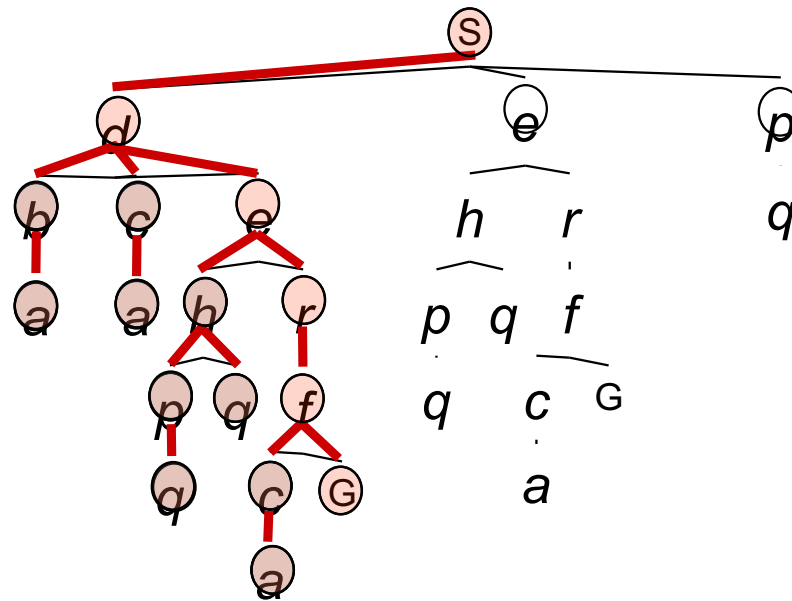


Depth-First Search

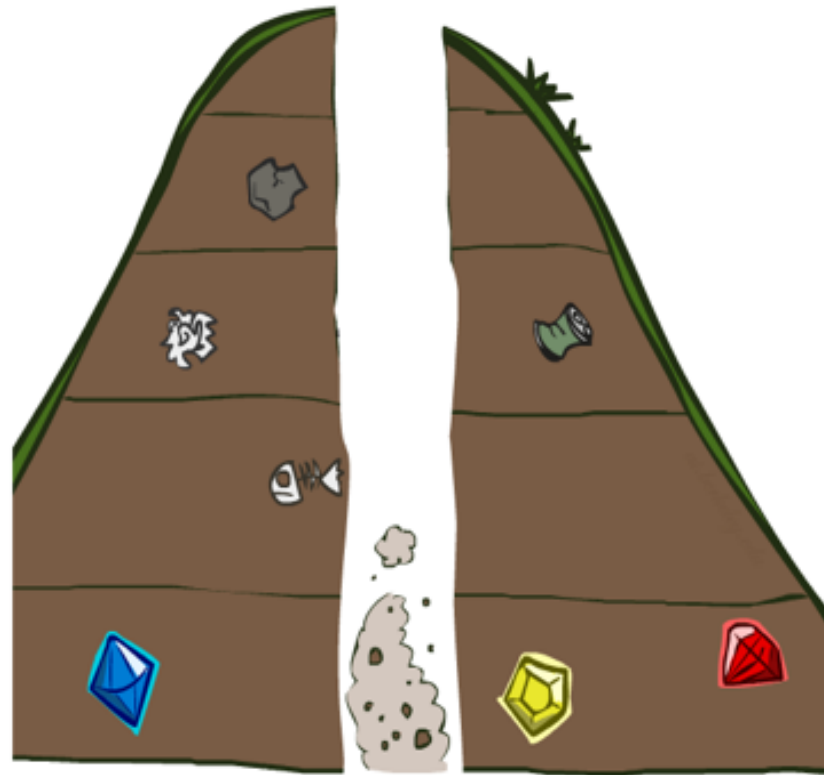
Strategy:
expand a
deepest node
first



Implementation: Fringe is
a LIFO stack



Search Algorithm Properties



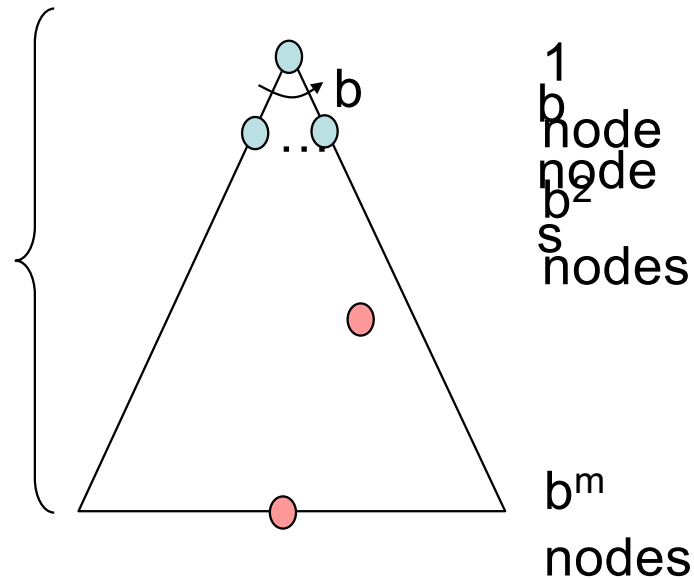
Search Algorithm Properties

- 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

m
tiers



- Number of nodes in entire tree?

- $1 + b + b^2 + \dots + b^m = O(b^m)$