CSE 573: Artificial Intelligence
Winter 2019

Hanna Hajishirzi

Problem Spaces and Search

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

Logistics

- Feedback:
  - You can submit feedback through the Allen School's anonymous feedback tool https://www.cs.washington.edu/alumni/feedback
- Discussions and questions in class
- Check the schedule
- Remember, there are three penalty-free late day for the whole quarter (except final project)

Outline

- Agents that Plan Ahead
- Search Problems
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search
- Heuristic Search Methods
  - Best First / Greedy Search
  - A*

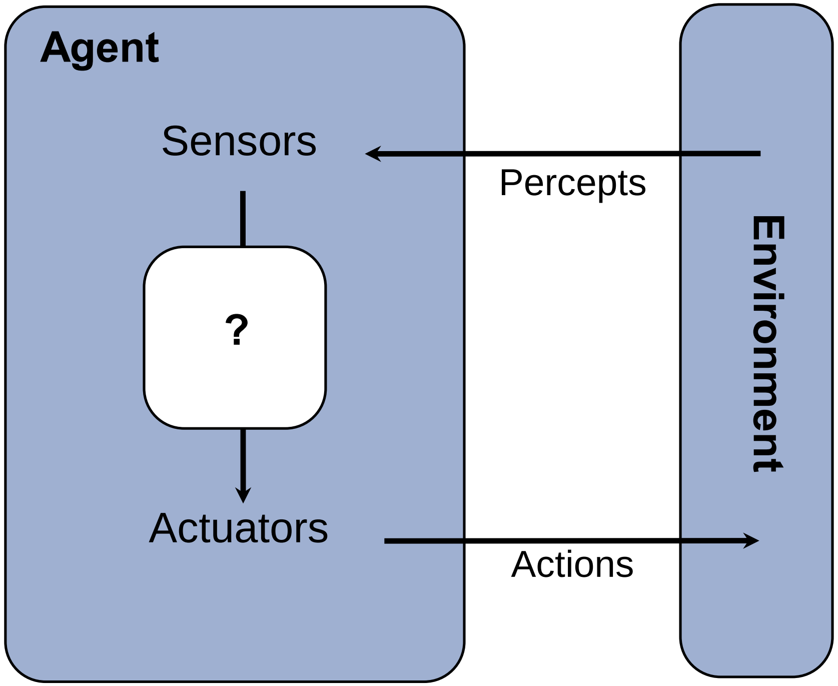
Review: Agents

An agent:

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

Environment:

- Input and output to the agent

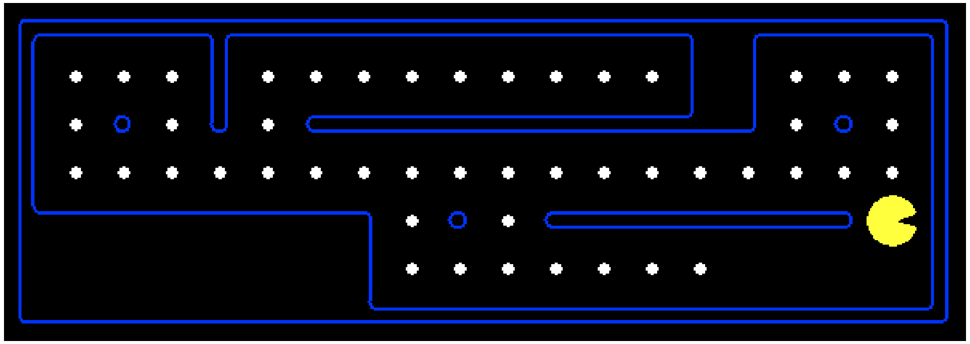


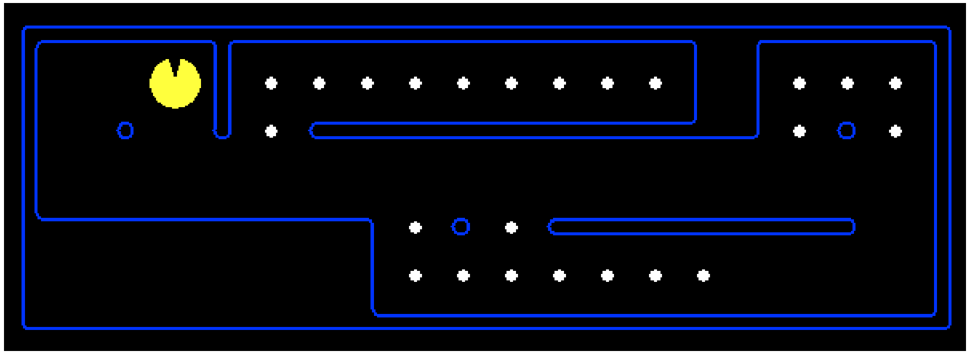
Search -- the environment is:

fully observable, single agent, deterministic, static, discrete

Reflex Agents

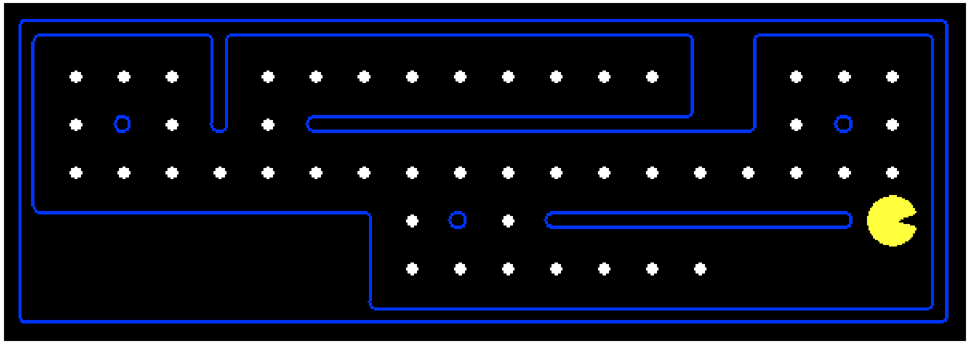
- Reflex agents:
  - Choose action based on current percept (and maybe memory)
  - Do not consider the future consequences of their actions
  - Act on how the world IS
- Can a reflex agent achieve goals?

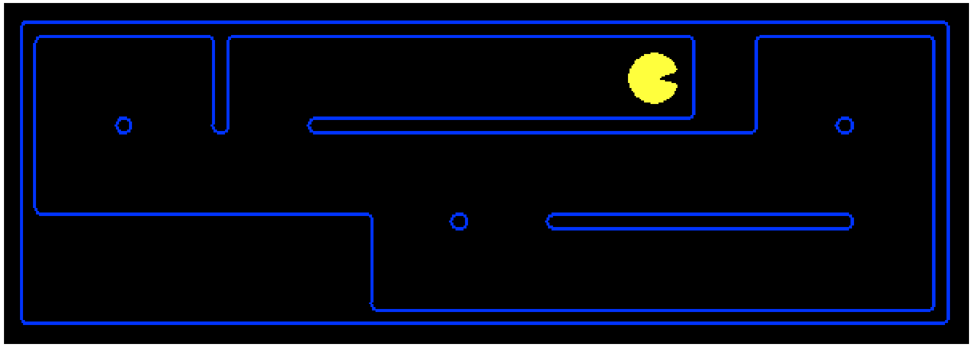




Goal Based Agents

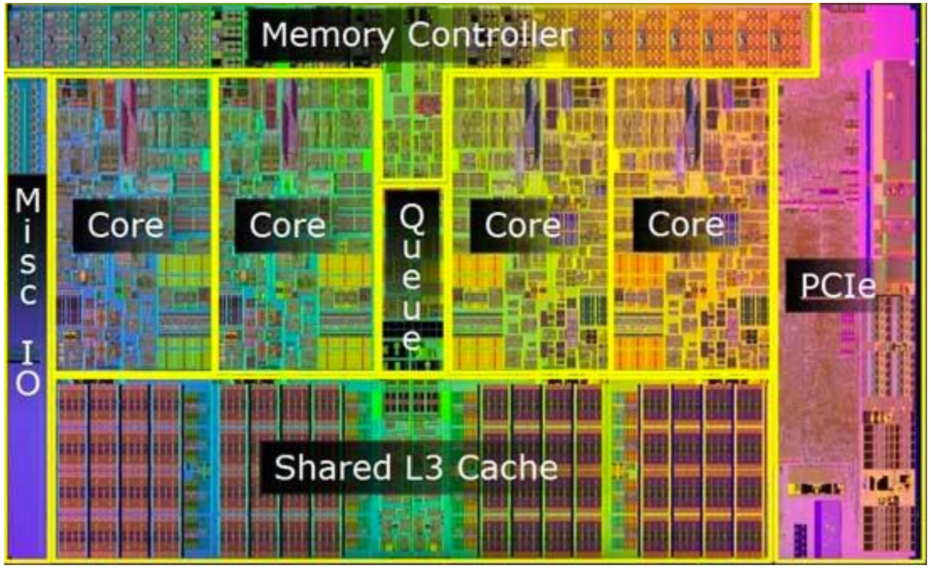
- Goal-based agents:
  - Plan ahead
  - Ask "what if"
  - Decisions based on (hypothesized) consequences of actions
  - Must have a model of how the world evolves in response to actions
  - Act on how the world WOULD BE





Search: it is not just for agents

Hardware verification



Planning optimal repair sequences

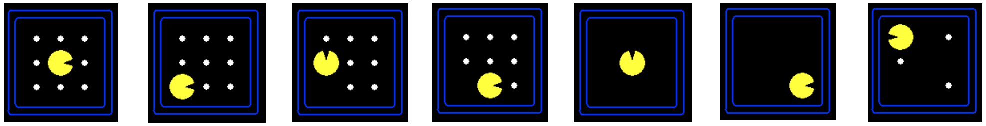
Search thru a
Problem Space / State Space

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

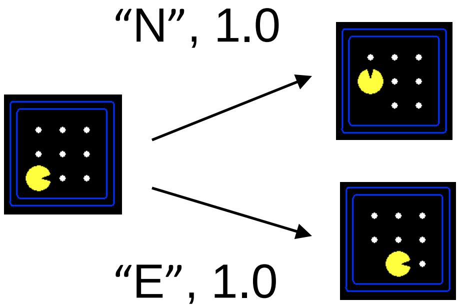
- Output:
  - Path: start $\Rightarrow$ a state satisfying goal test
  - [May require shortest path]
  - [Sometimes just need state passing test]

Example: Simplified Pac-Man

- Input:
  - A state space



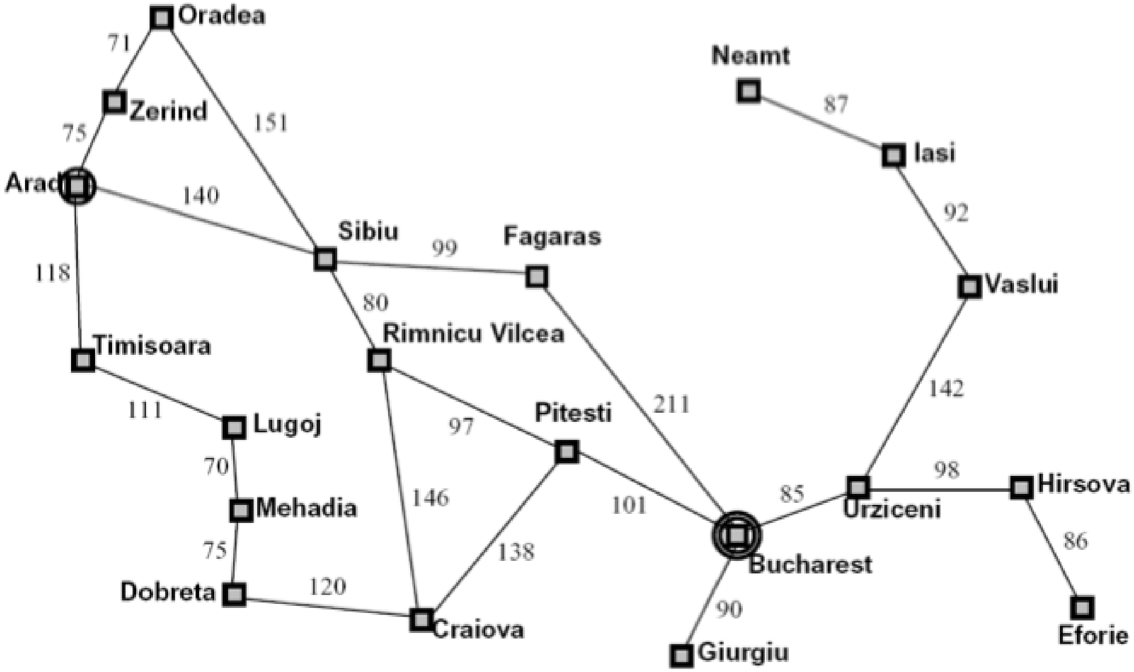
  - A successor function



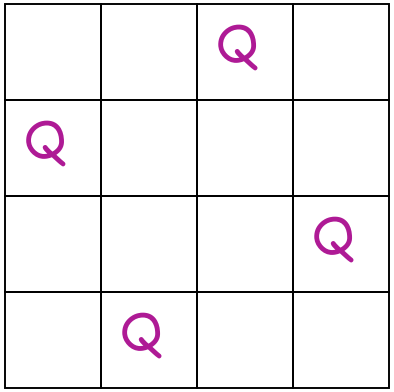
  - A start state
  - A goal test
- Output:

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

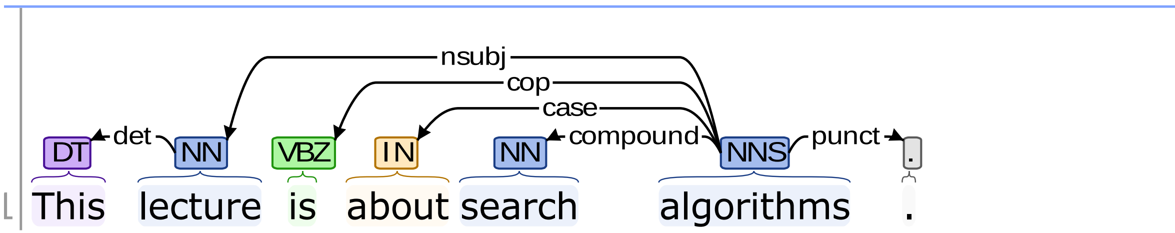
$$
\begin{aligned}
\partial_r^2 u &= -\left[E' - \frac{l(l+1)}{r^2} - r^2\right] u(r) \\
e^{-2s}\left(\partial_s^2 - \partial_s\right) u(s) &= -\left[E' - l(l+1)e^{-2s} - e^{2s}\right] u(s) \\
e^{-2s}\left[e^{\frac{1}{2}s}\left(e^{-\frac{1}{2}s}u(s)\right)'' - \frac{1}{4}u\right] &= -\left[E' - l(l+1)e^{-2s}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

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

Parsing Natural Language

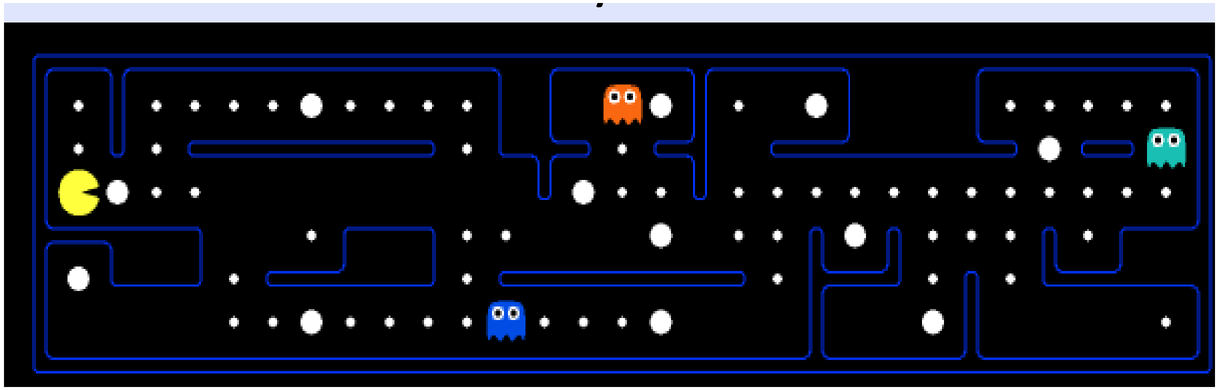
- Input:
  - Set of states
  - Operations
  - Start state
  - Goal state (test)
- Output:

This lecture is about search algorithms.



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

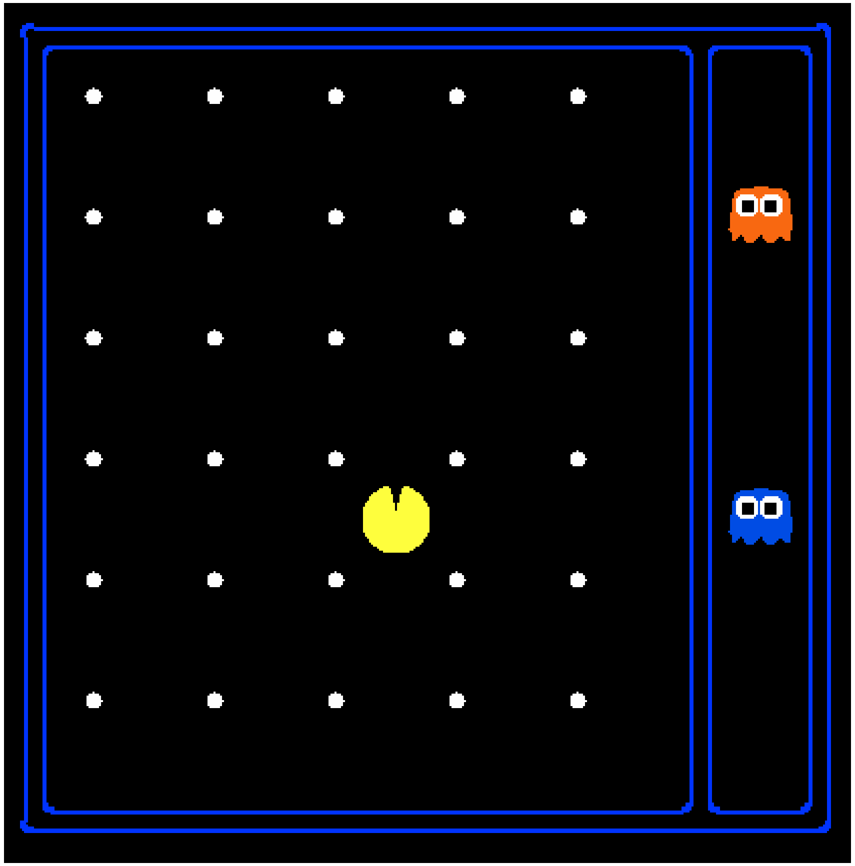
Successor: update location and dot boolean

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:

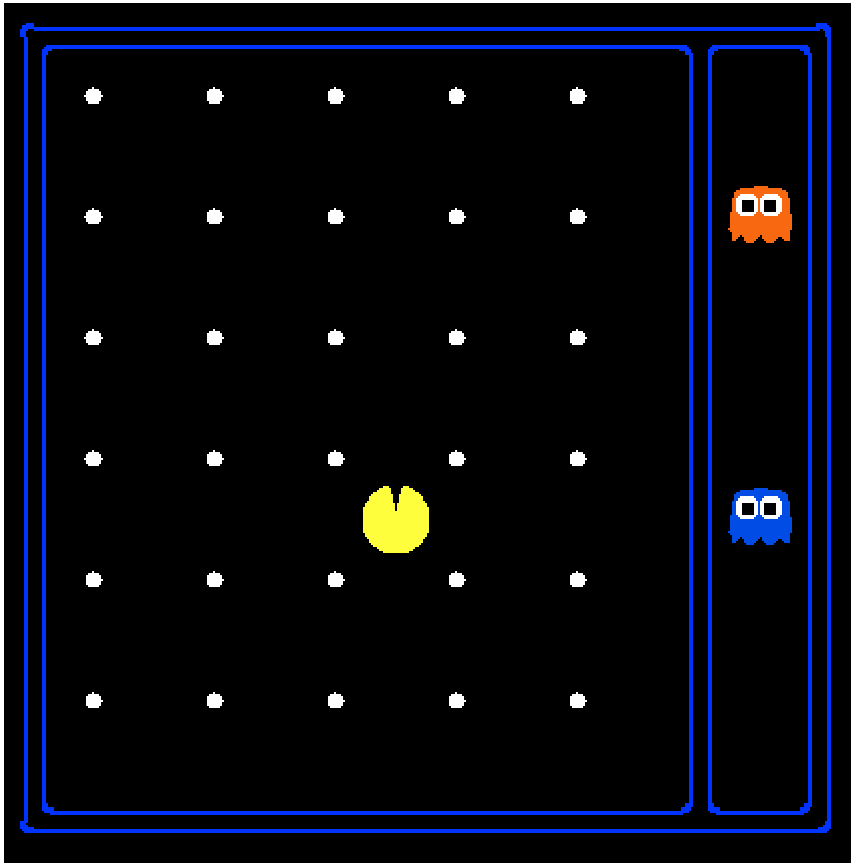
  $120*(2^{30})*(12^{2})*4$

- States for Pathing:

  120

- States for eat-all-dots:

  $120*(2^{30})$

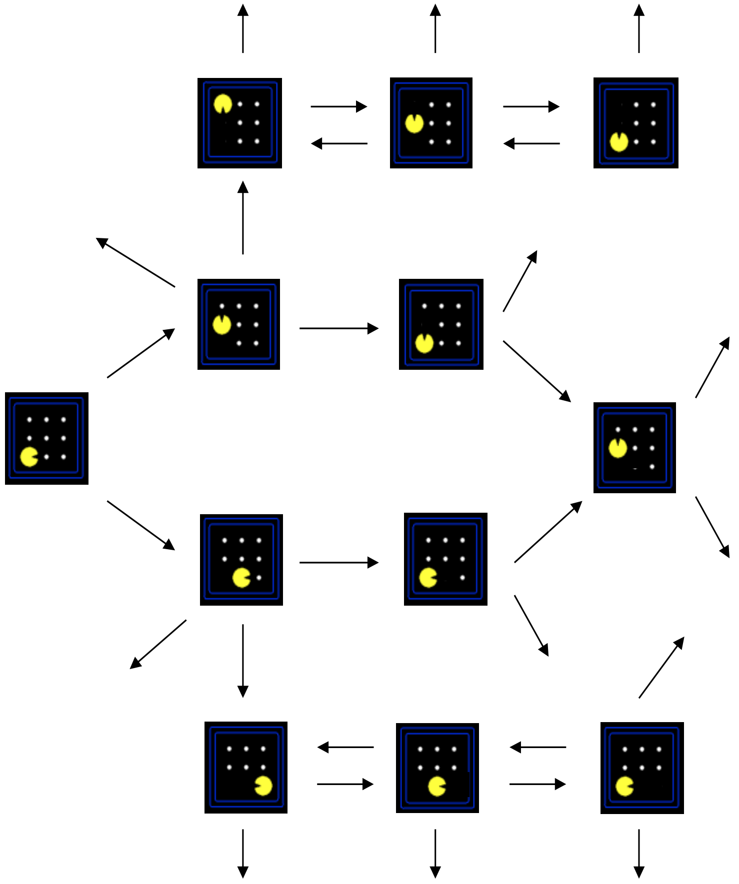


State Representation

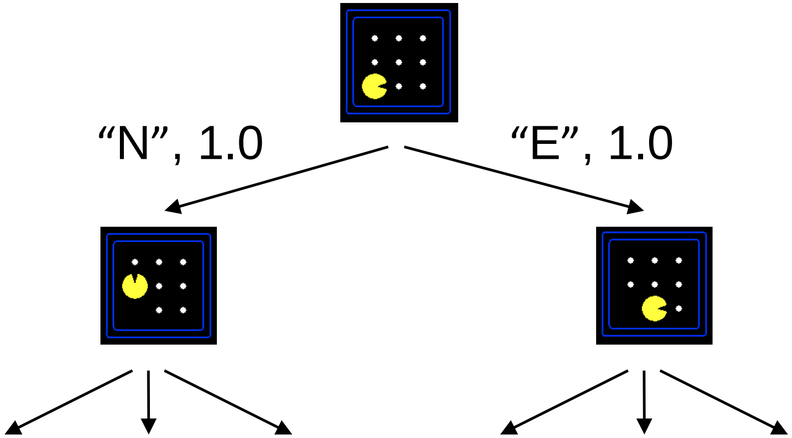
- Real-world applications:
  - Requires approximations and heuristics
  - Need to design state representation so that search is feasible
    - Only focus on important aspects of the state
    - E.g., Use features to represent world states

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)



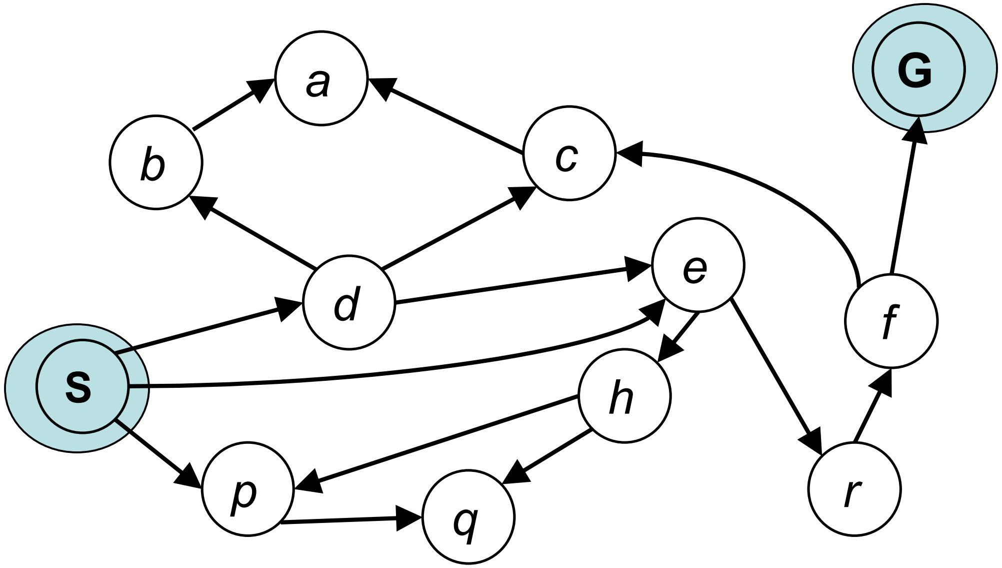
Search Trees



- 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

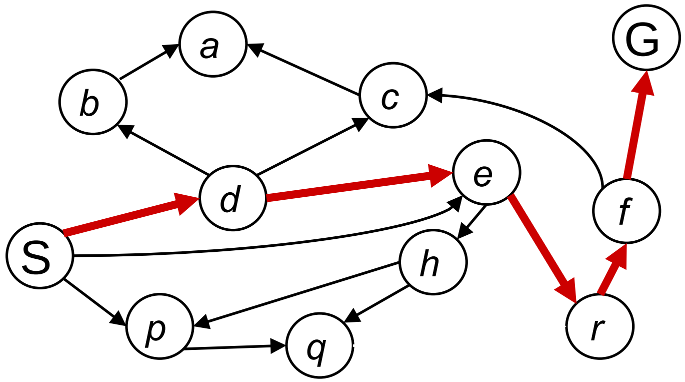
Example: Tree Search

State Graph:

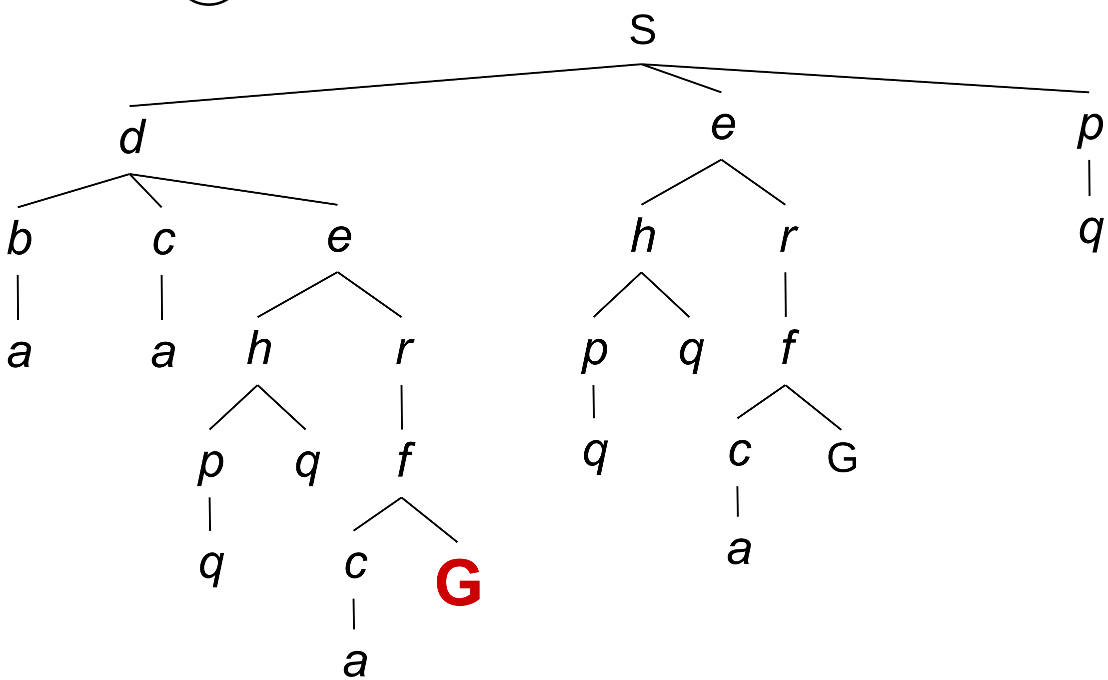


What is the search tree?

State Graphs vs. Search Trees



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



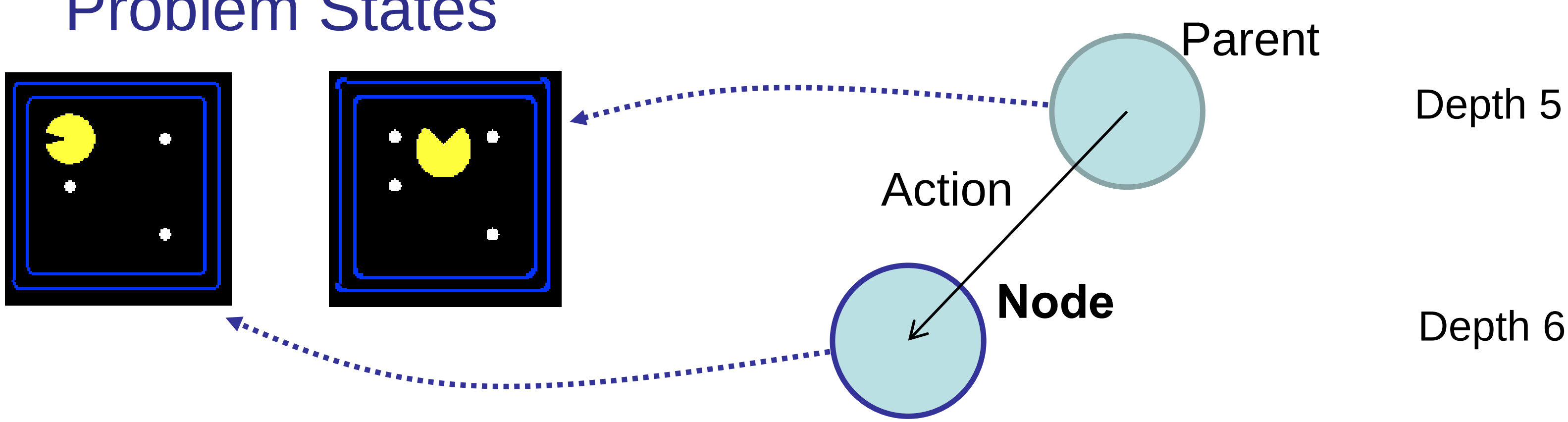
We construct both on demand – and we construct as little as possible.

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

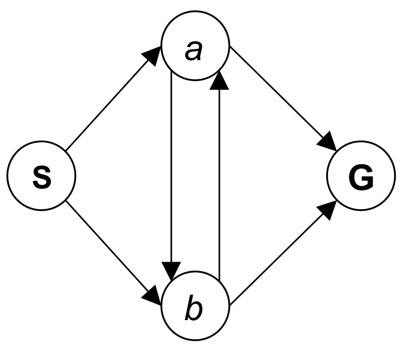
Search Nodes

Problem States



Quiz: State Graphs vs. Search Trees

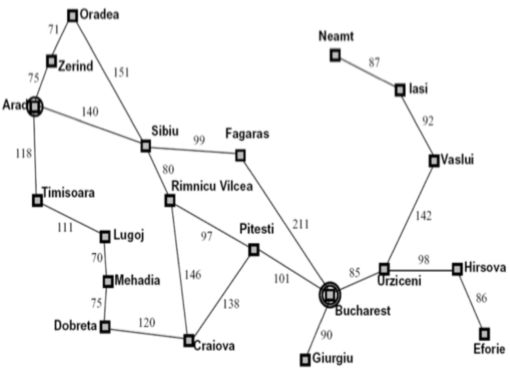
Consider this 4-state graph:

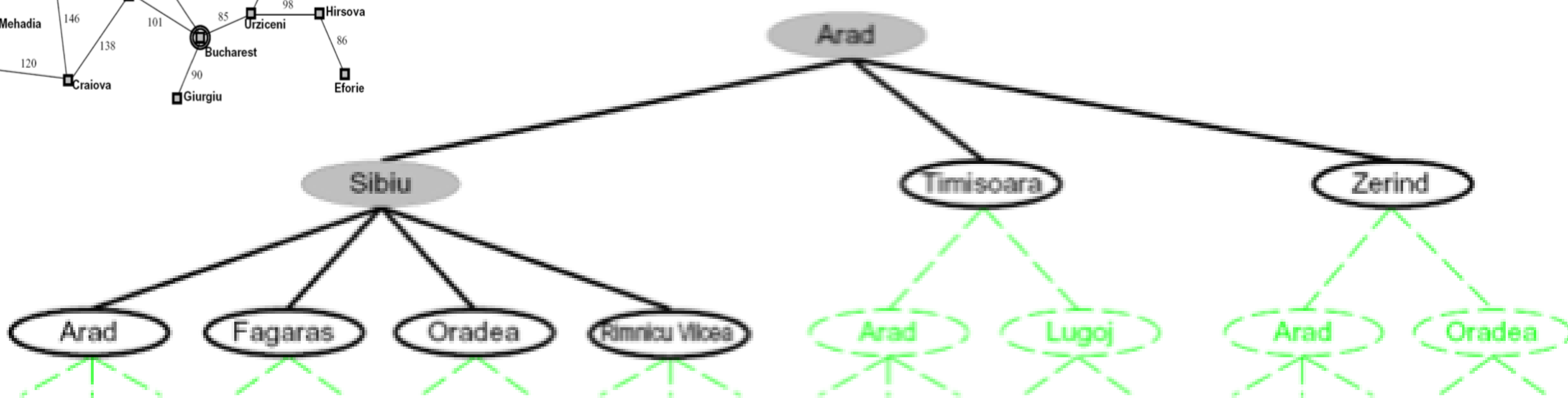


How big is its search tree (from S)?

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
```

Detailed pseudocode is in the book!

- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy

- Main question: which fringe nodes to explore?

Search Algorithms

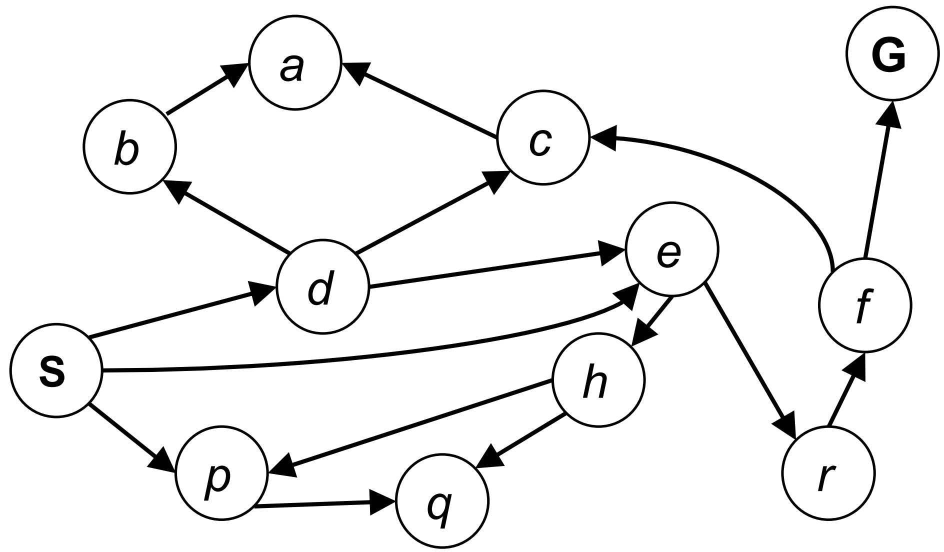
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search

- Heuristic Search Methods
  - Best First / Greedy Search
  - A*

Review: Depth First Search

***Strategy**: expand deepest node first*

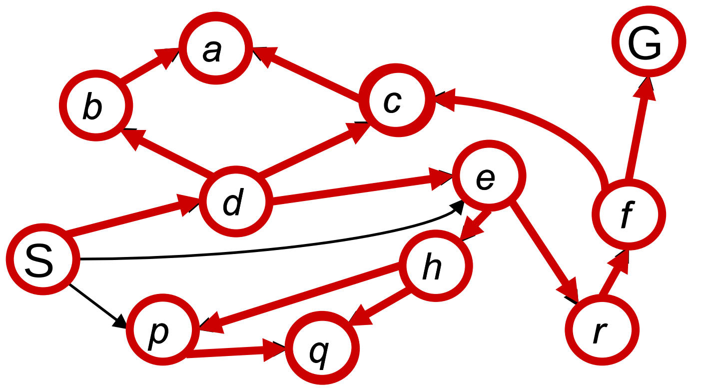
***Implementation**: Fringe is a LIFO queue (a stack)*

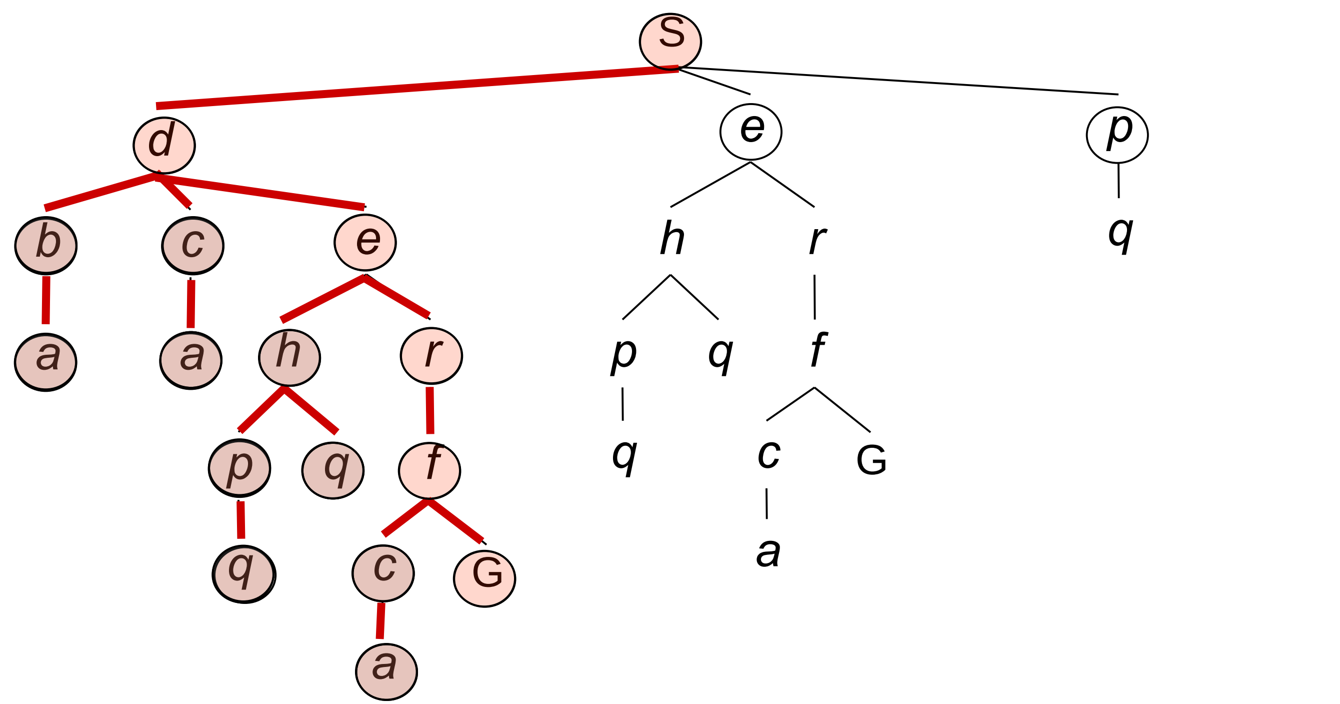


Review: Depth First Search

Expansion ordering:

(d,b,a,c,a,e,h,p,q,q,r,f,c,a,G)

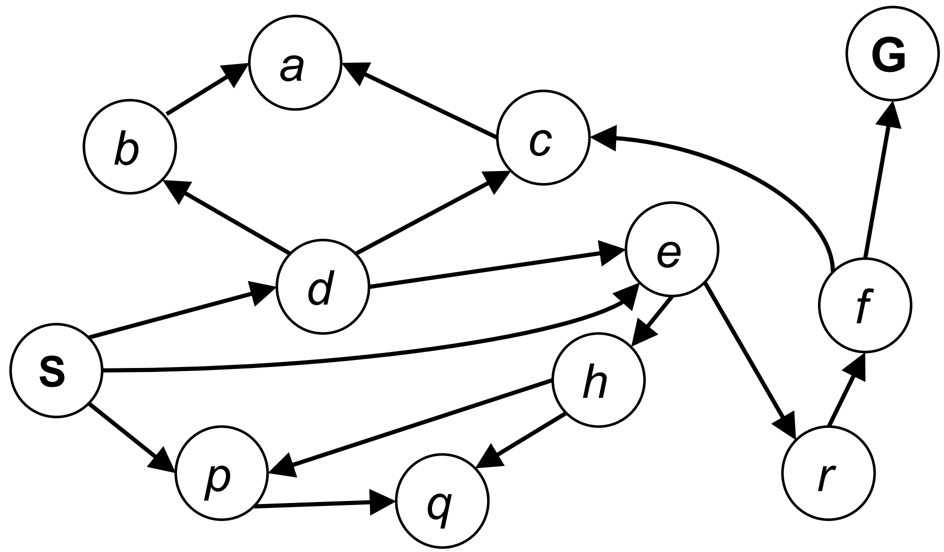




Review: Breadth First Search

***Strategy**: expand shallowest node first*

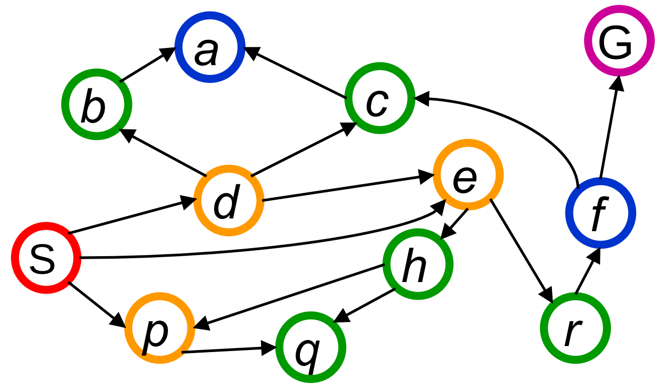
***Implementation**: Fringe is a FIFO queue*



Review: Breadth First Search

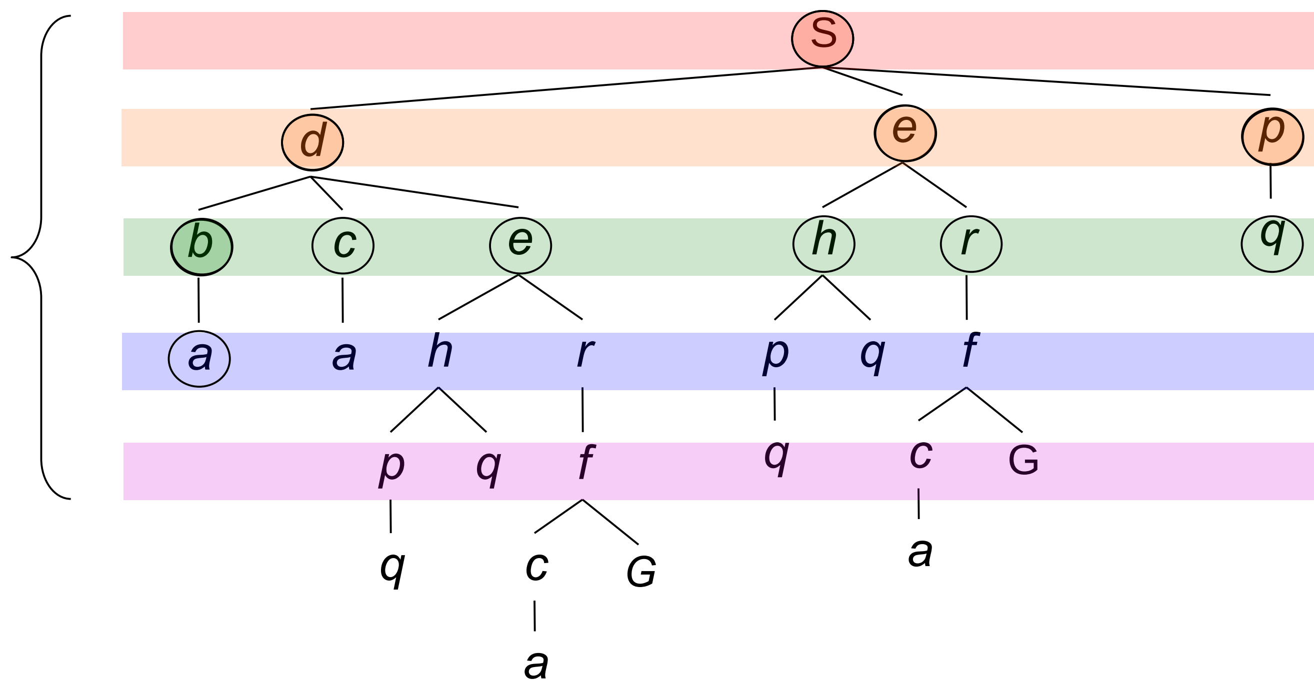
Expansion order:

(S,d,e,p,b,c,e,h,r,q,a,a ,h,r,p,q,f,p,q,f,q,c,G)



Search

Tiers



Search Algorithm Properties

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

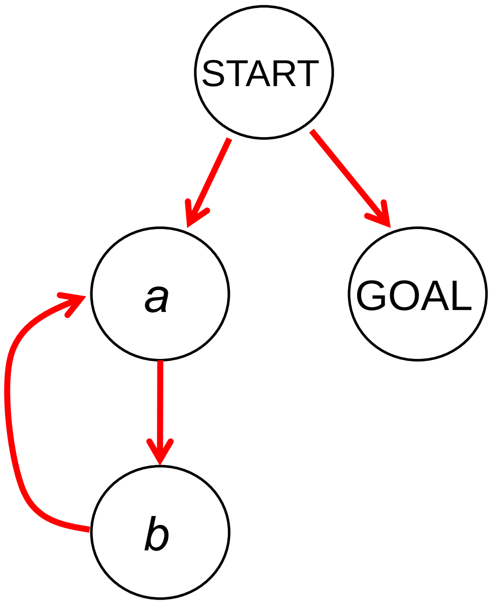
Variables:

n	Number of states in the problem
b	The maximum branching factor *B* (the maximum number of successors for a state)
C^*	Cost of least cost solution
d	Depth of the shallowest solution
m	Max depth of the search tree

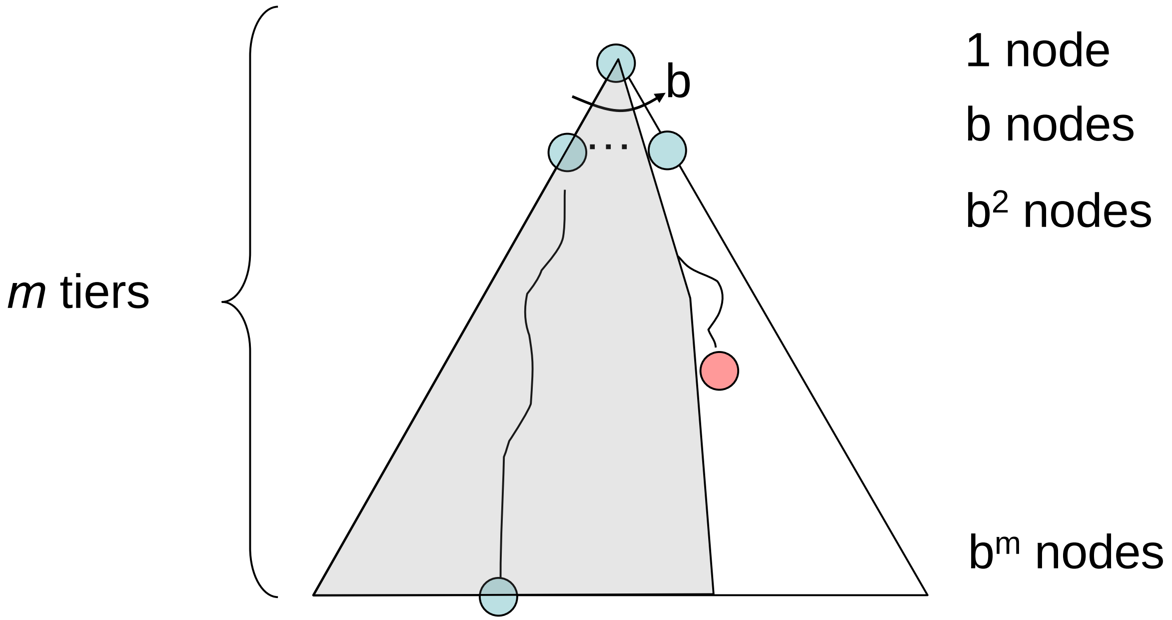
DFS

Algorithm		Complete	Optimal	Time	Space
DFS	Depth First Search	No	No	Infinite	Infinite

- Infinite paths make DFS incomplete…
  - How can we fix this?
  - Check new nodes against path from S
- Infinite search spaces still a problem
  - If the left subtree has unbounded depth



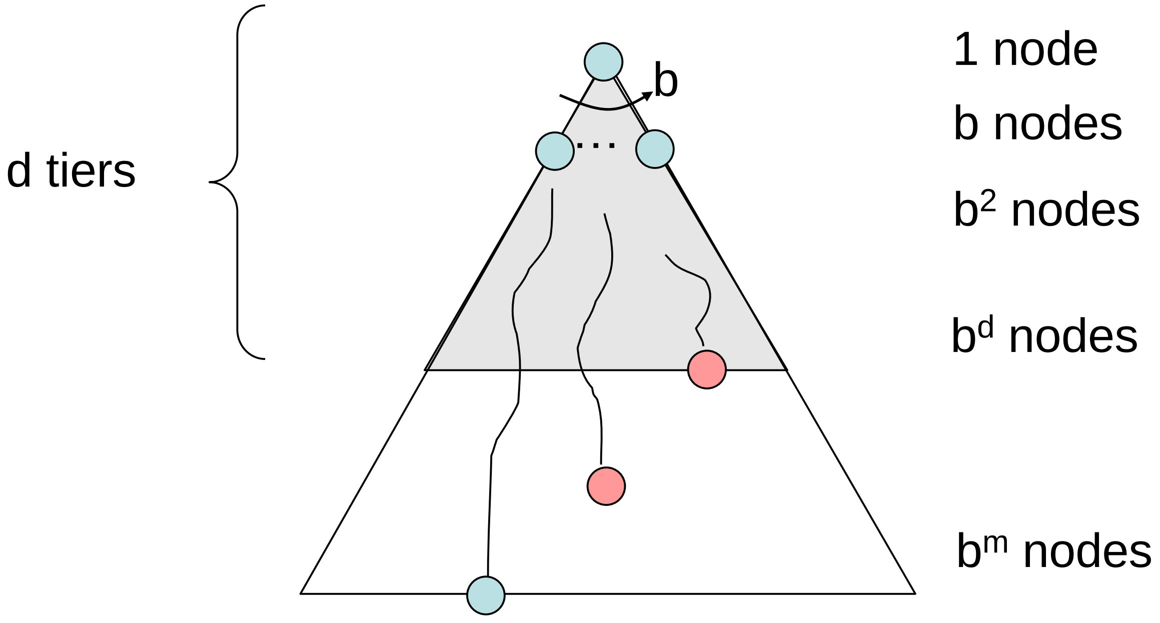
DFS



Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y if finite	N	$O(b^m)$	$O(bm)$

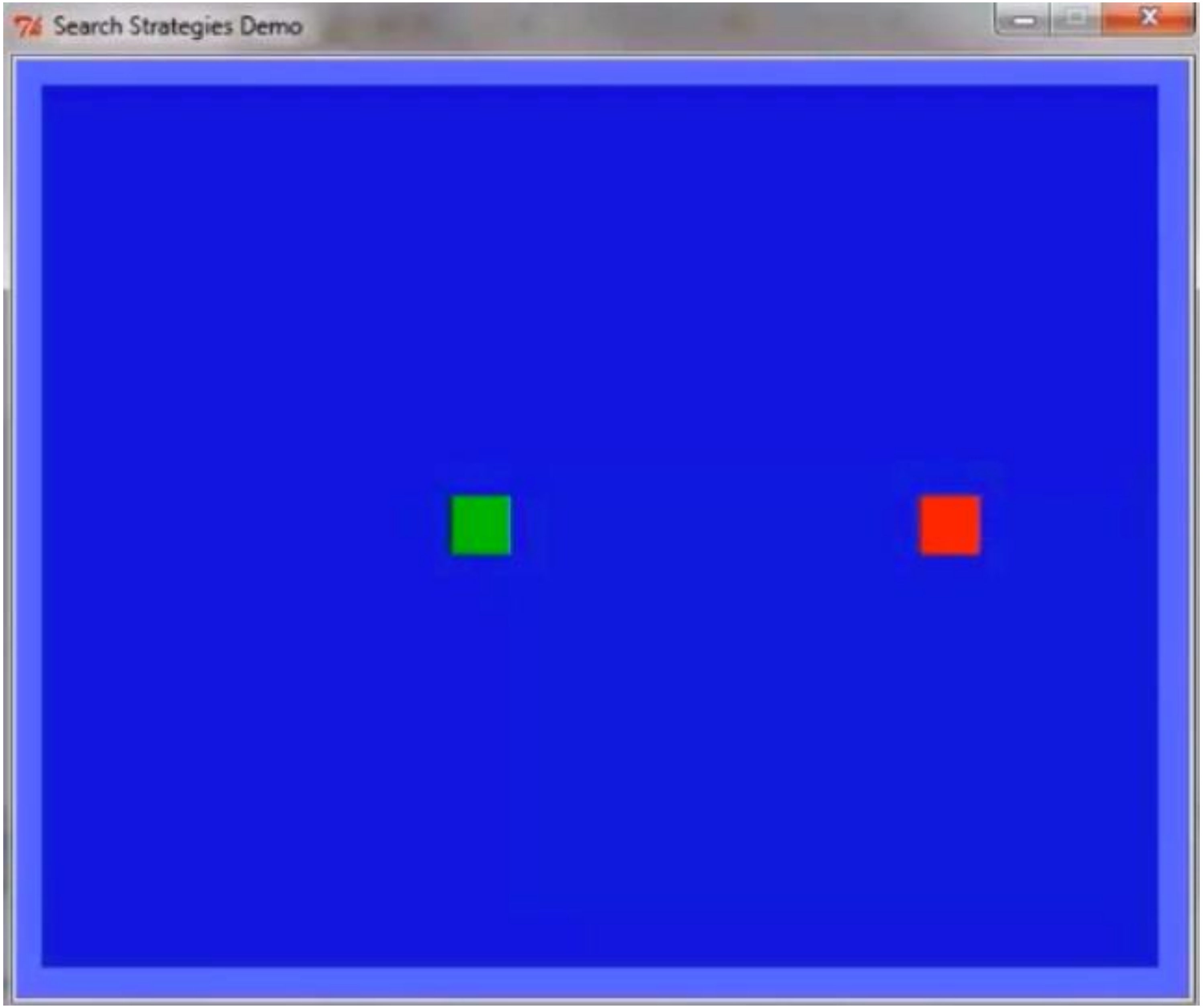
BFS

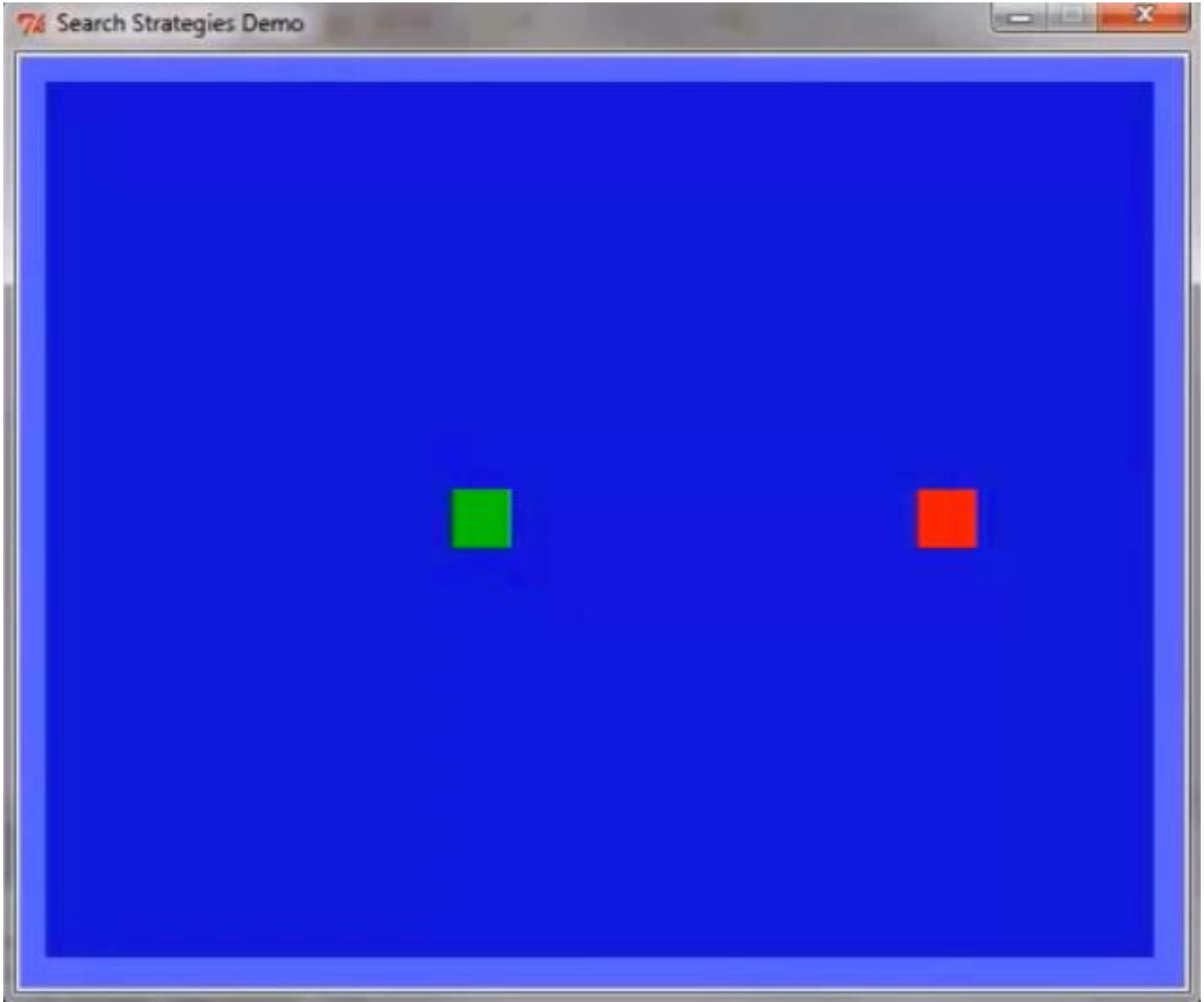
Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^m)$	$O(bm)$
BFS		Y	Y*	$O(b^d)$	$O(b^d)$



Comparisons

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


Search Strategies Demo

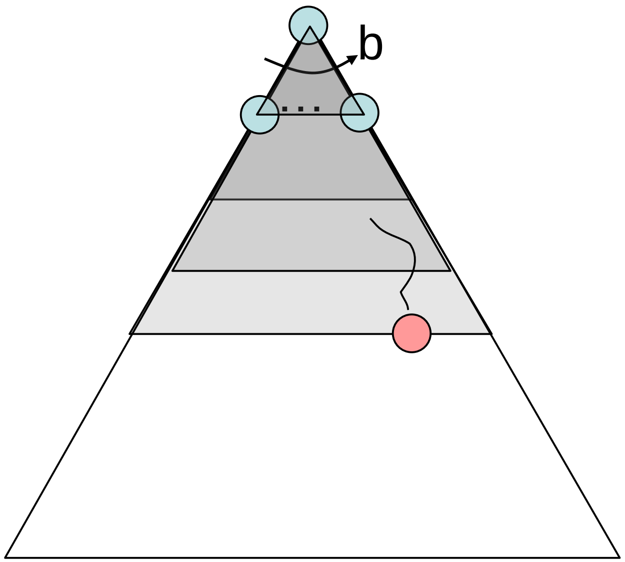

Search Strategies Demo

Iterative Deepening

Iterative deepening uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less.
2. If "1" failed, do a DFS which only searches paths of length 2 or less.
3. If "2" failed, do a DFS which only searches paths of length 3 or less.

….and so on.



Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^m)$	$O(bm)$
BFS		Y	Y*	$O(b^d)$	$O(b^d)$
ID		Y	Y*	$O(b^d)$	$O(bd)$

Search Methods

- Blind Search:
  - Depth First Search
  - Breadth First Search
  - Iterative Deepening Search

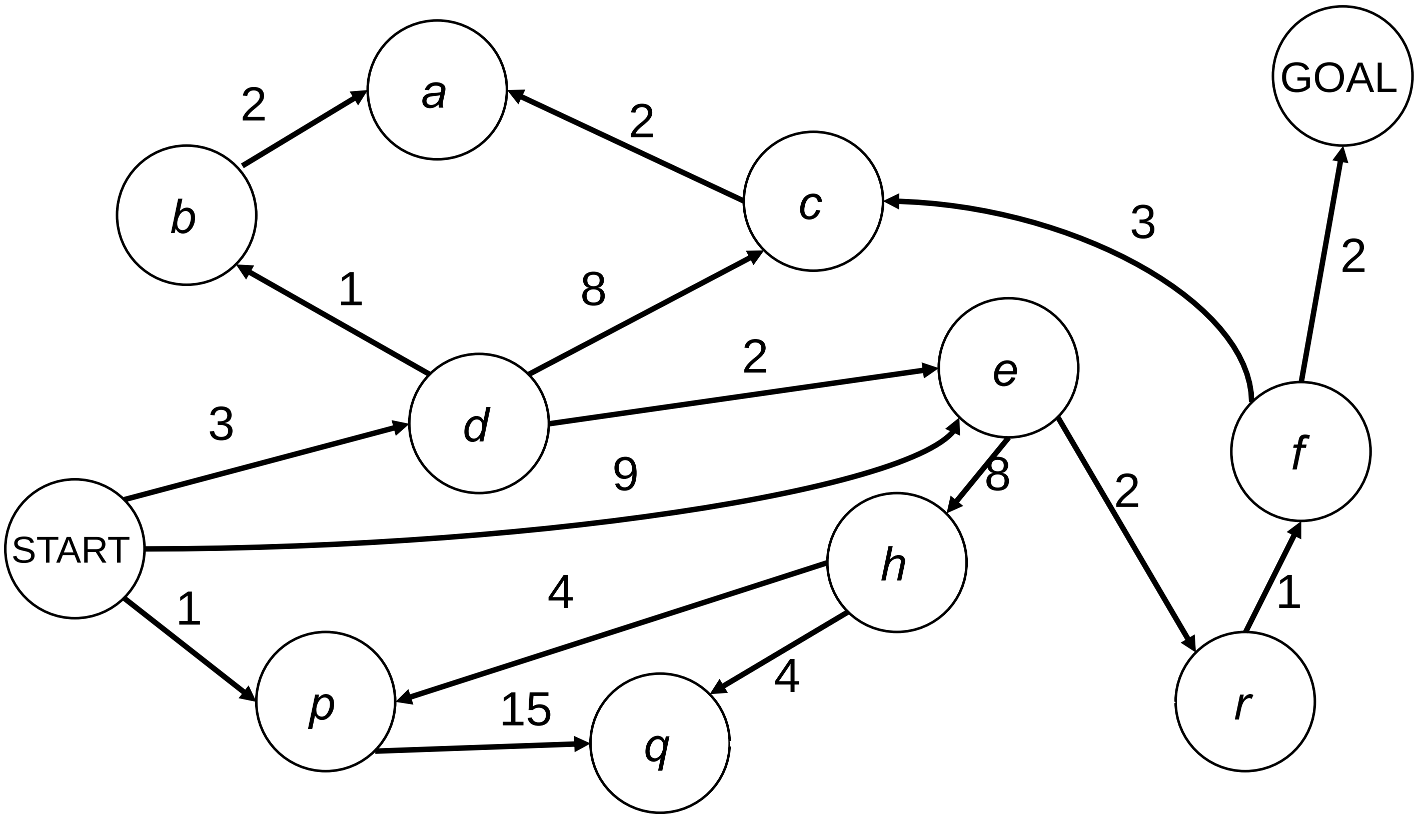
Search Methods

- Blind Search:
  - Depth First Search
  - Breadth First Search
  - Iterative Deepening Search
- Heuristic Search
  - Best First Search
  - Uniform Cost Search
  - Greedy Search
  - A*
  - Iterative Deepening A*
  - Beam Search
  - Hill Climbing

Blind Vs. Heuristic Search

- Cost of actions
- Heuristic guidance

Costs on Actions

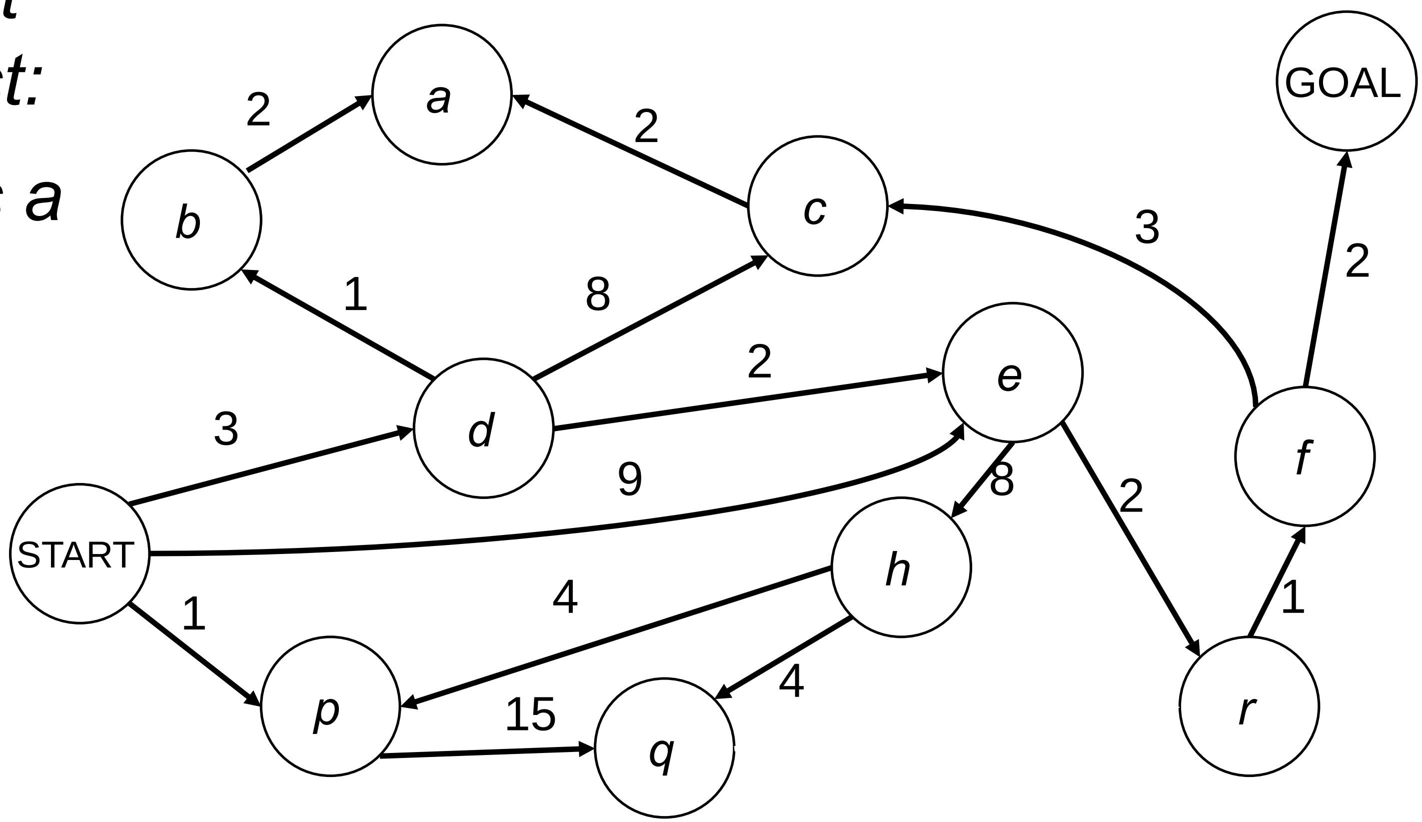


Notice that BFS finds the shortest path in terms of number of transitions. It does not find the least-cost path.

Uniform Cost Search

Expand cheapest node first:

Fringe is a priority queue

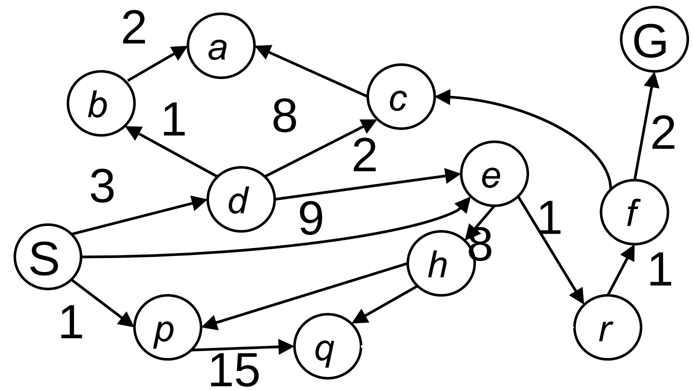


Uniform Cost Search

- Generalization of breadth-first search
- ***Priority*** queue of nodes to be explored
- Cost function f(n) applied to each node

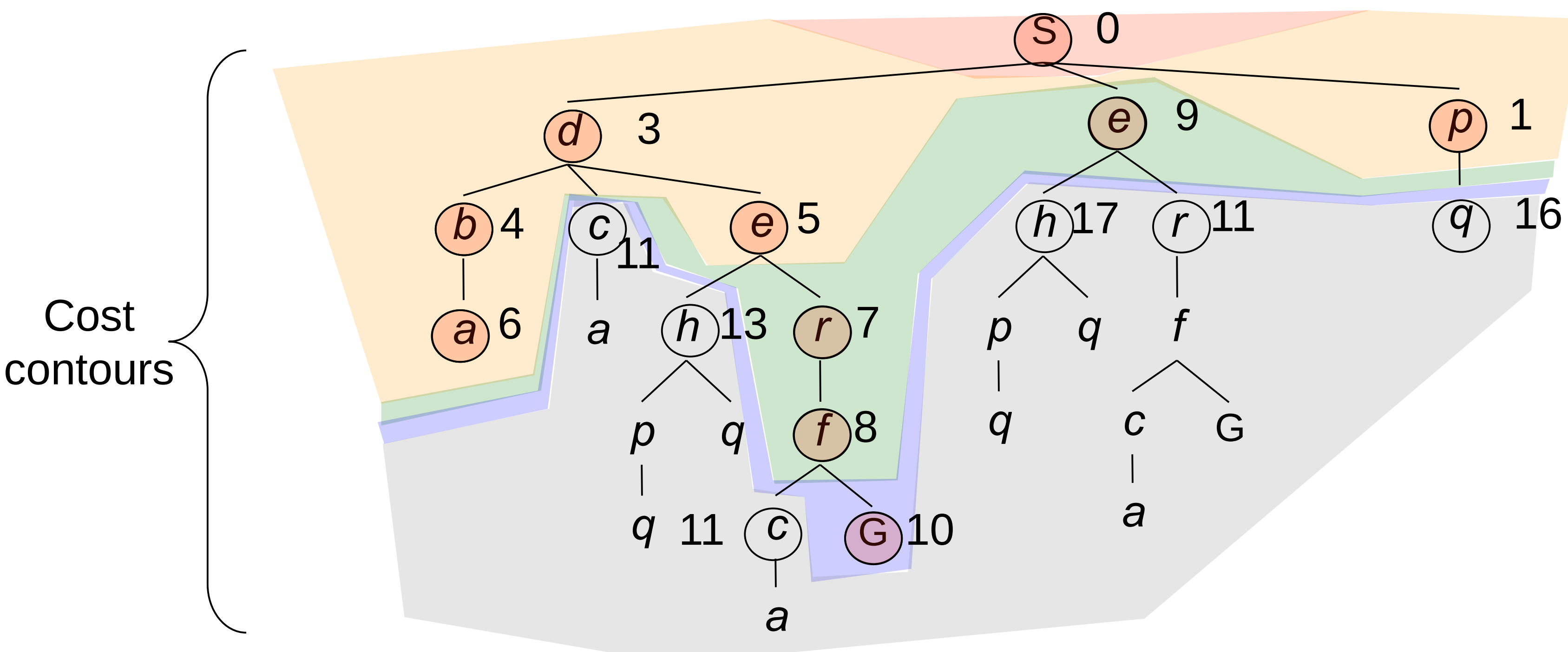
```
Add initial state to priority queue
While queue not empty
        Node = head(queue)
        If goal?(node) then return node
        Add children of node to queue
```

Uniform Cost Search



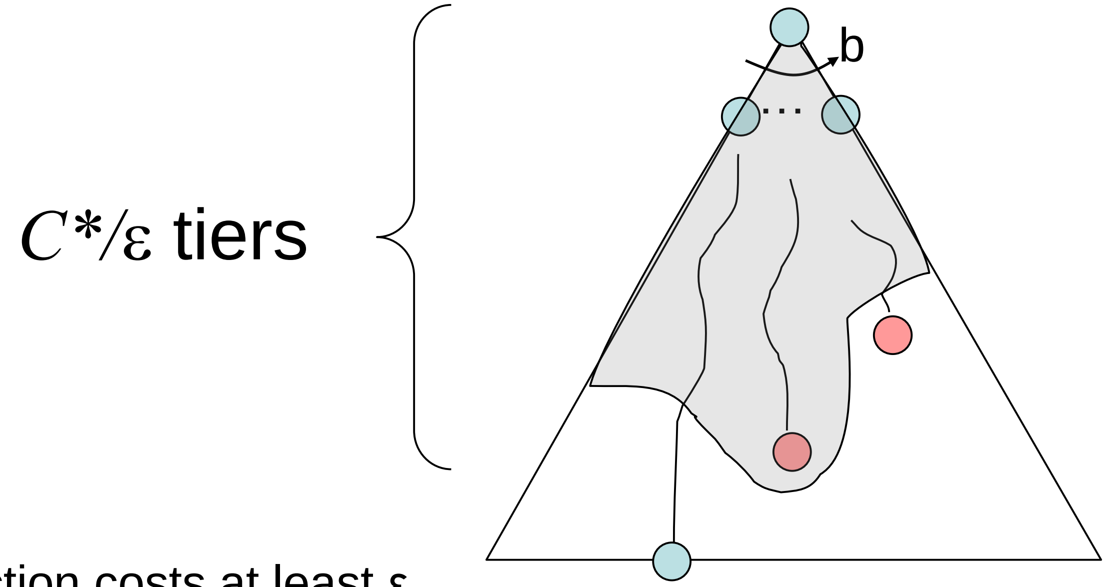
Expansion order:

(S,p,d,b,e,a,r,f,e,G)



Uniform Cost Search

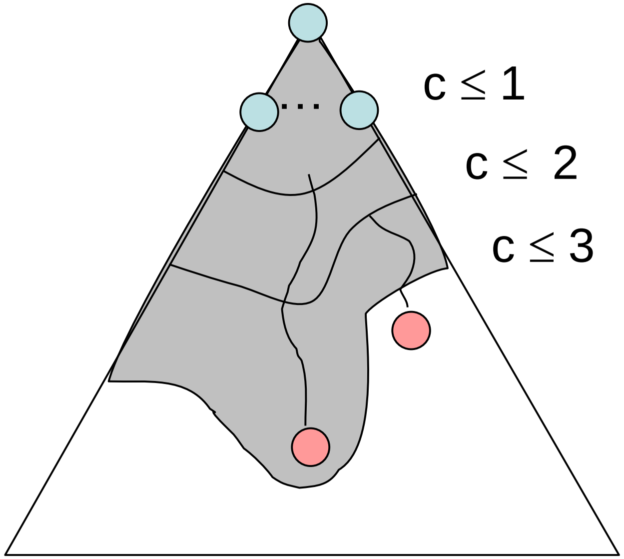
Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^m)$	$O(bm)$
BFS		Y	Y*	$O(b^d)$	$O(b^d)$
UCS		Y*	Y	$O(b^{C*/\varepsilon})$	$O(b^{C*/\varepsilon})$

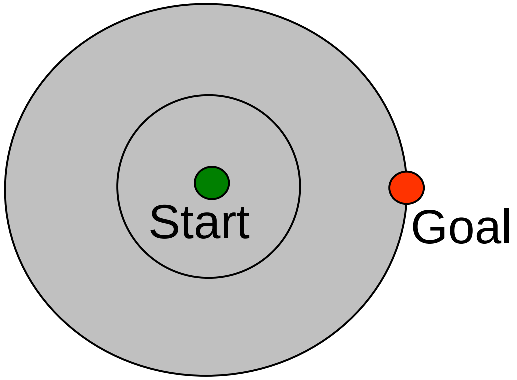


Every action costs at least ε

Uniform Cost Issues

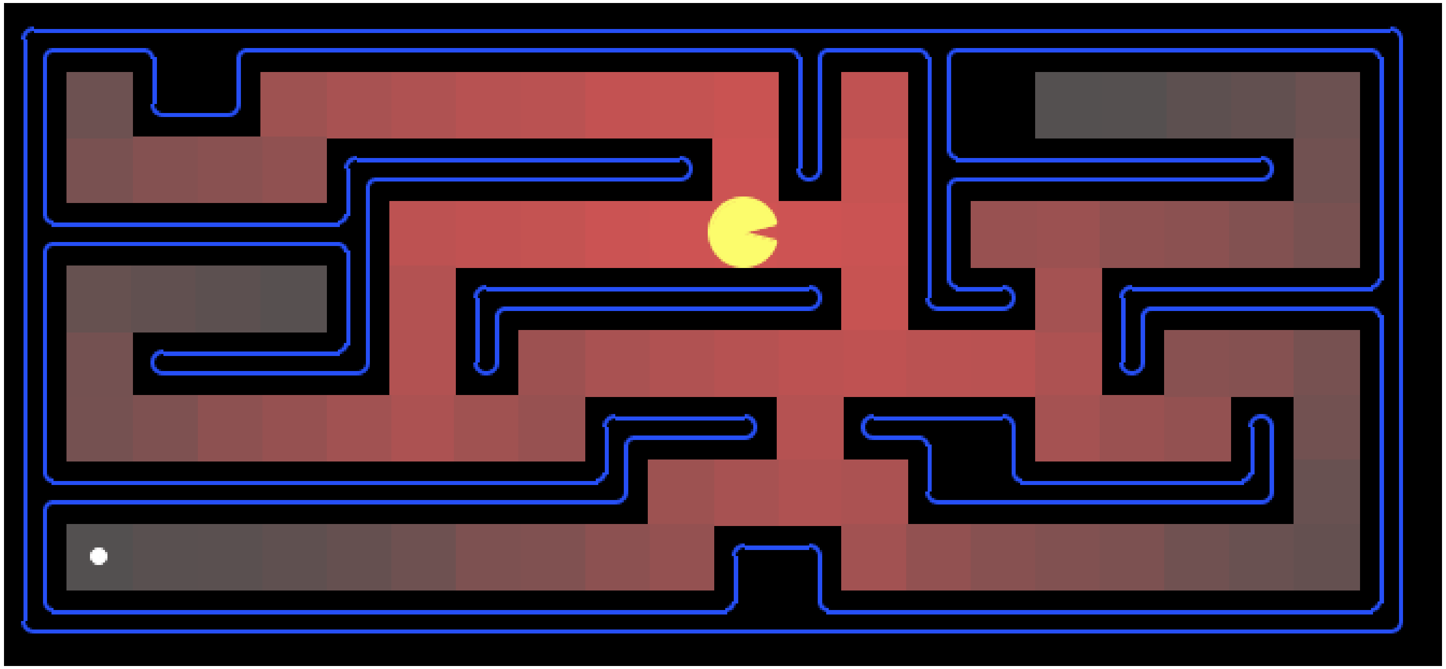
- Remember: explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every “direction”
  - No information about goal location





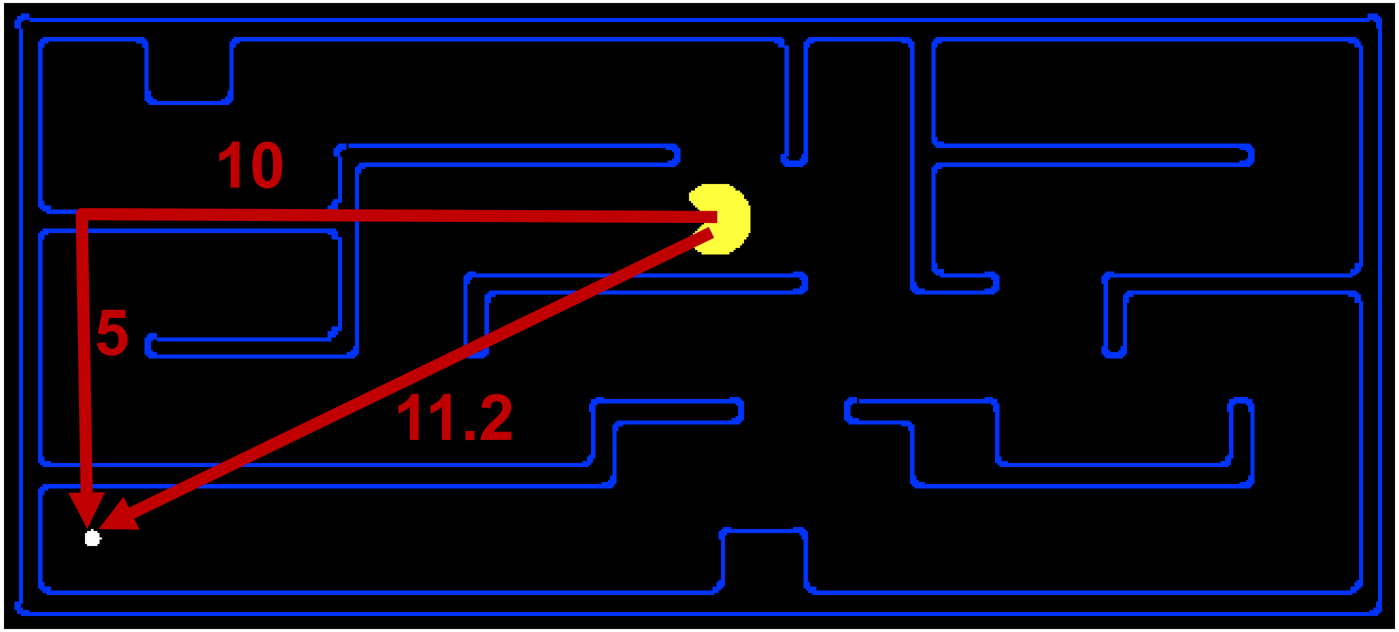
Uniform Cost: Pac-Man

- Cost of 1 for each action
- Explores all of the states, but one



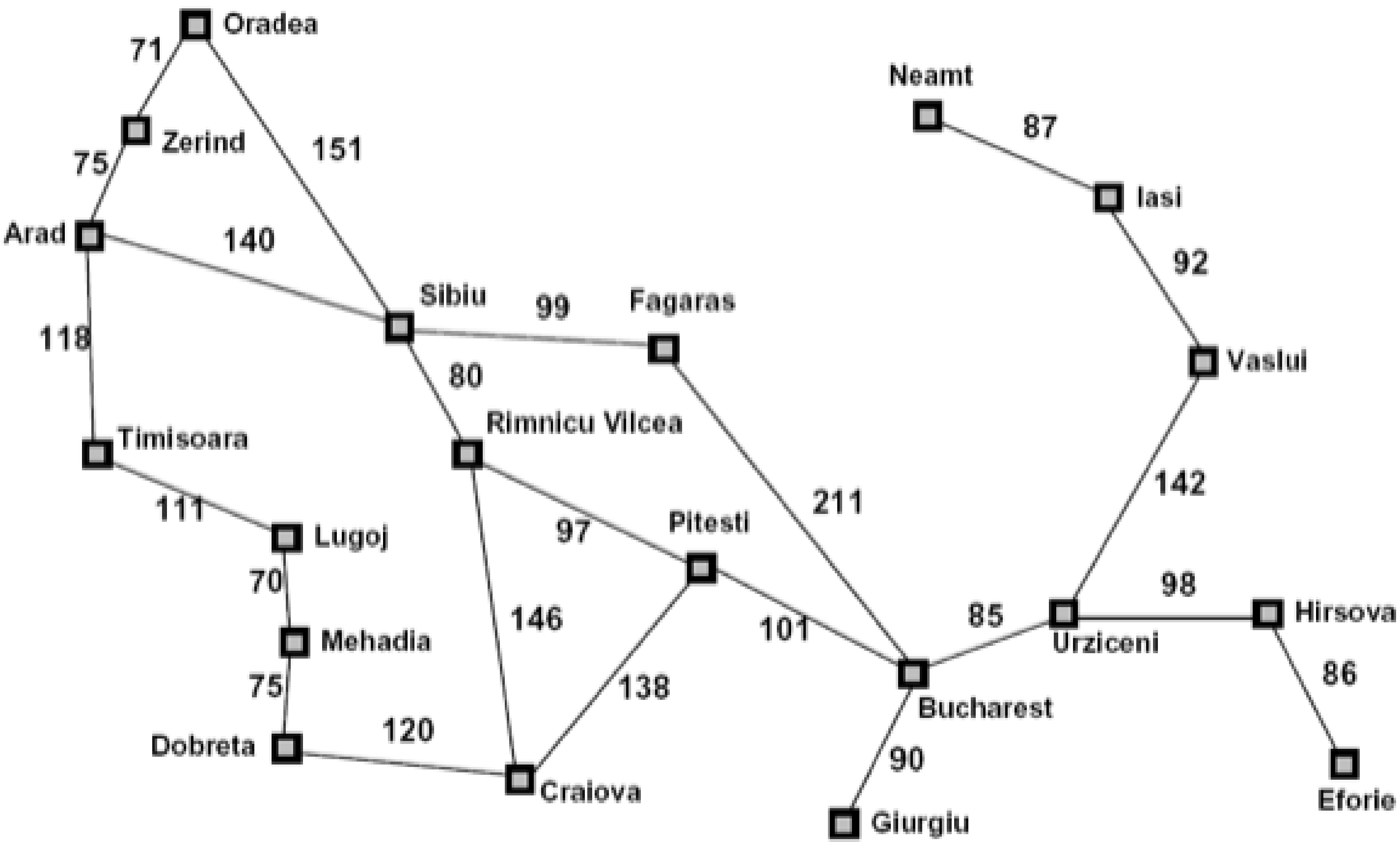
Search Heuristics

- Any *estimate* of how close a state is to a goal
- Designed for a particular search problem



- Examples: Manhattan distance, Euclidean distance

Heuristics



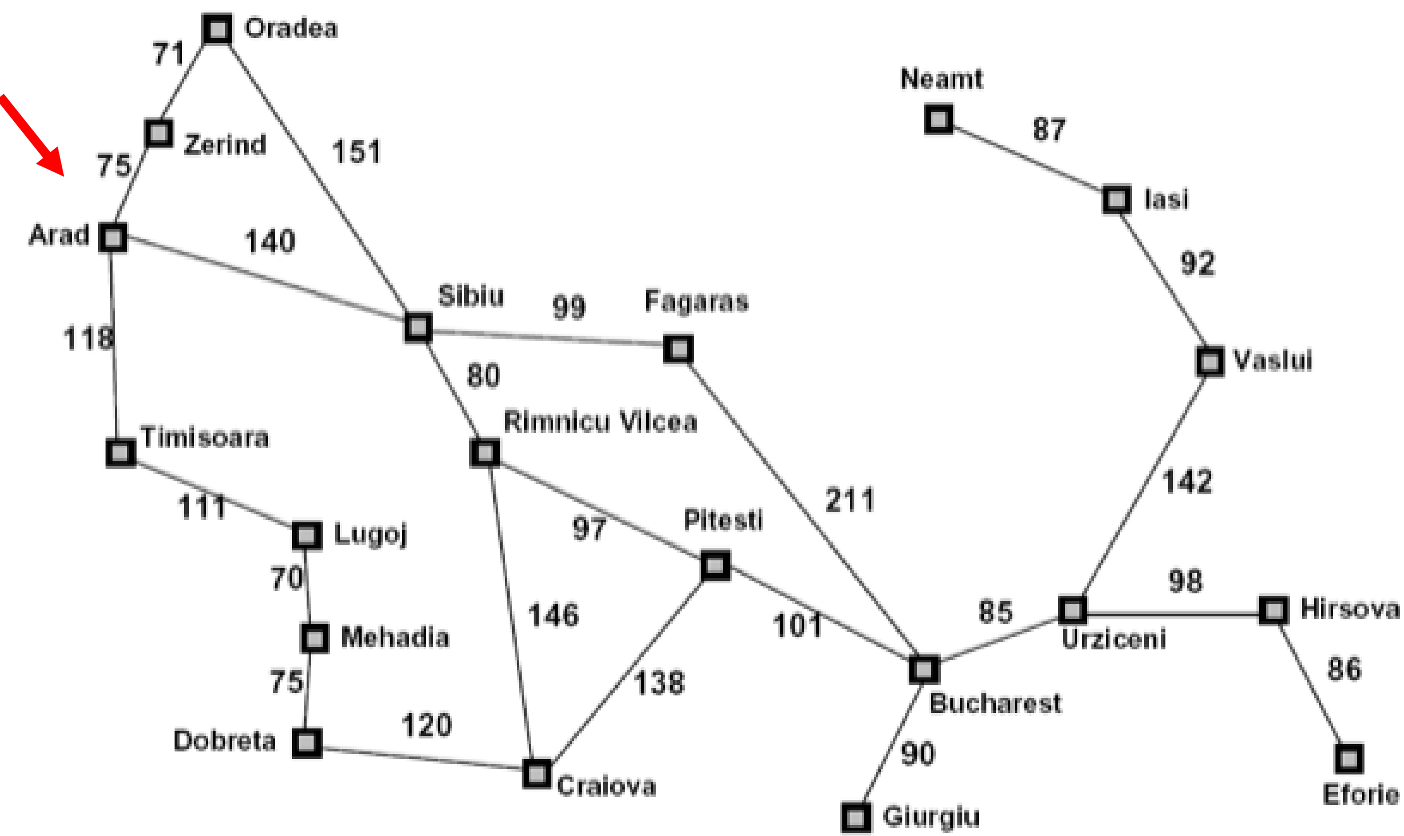
Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

H(x)

Best First / Greedy Search

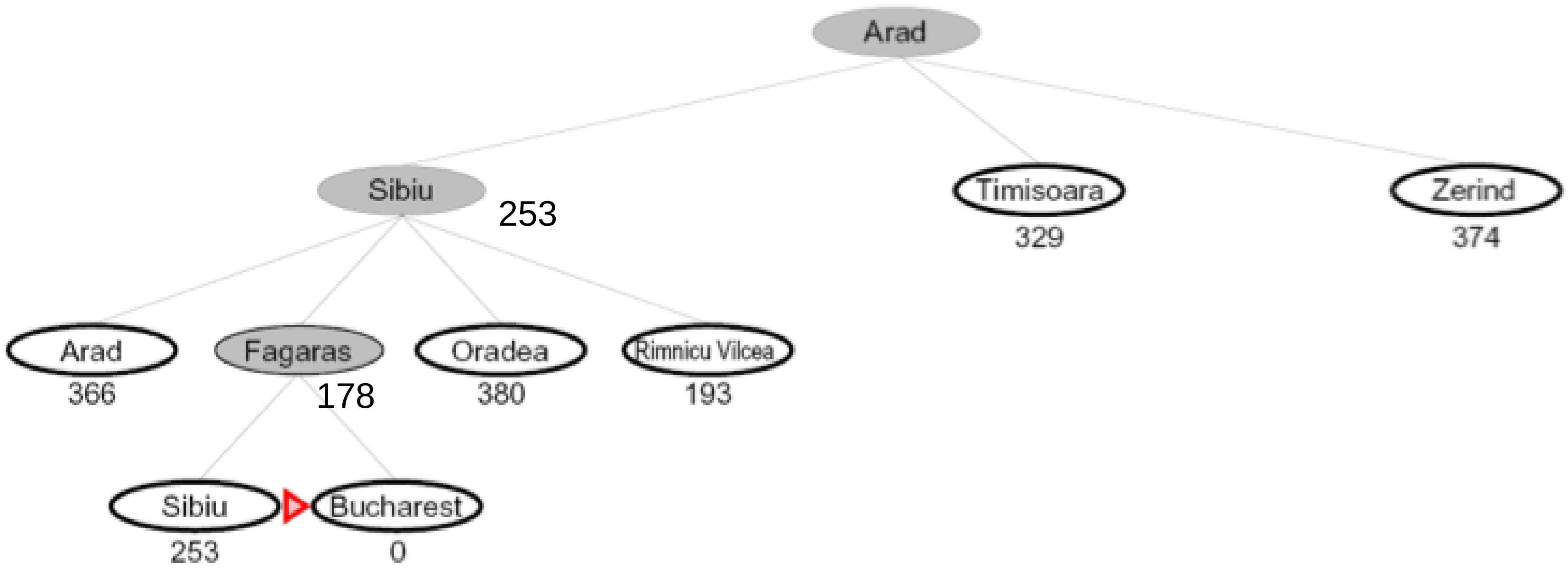
Best first with f(n) = heuristic estimate of distance to goal



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Best First / Greedy Search

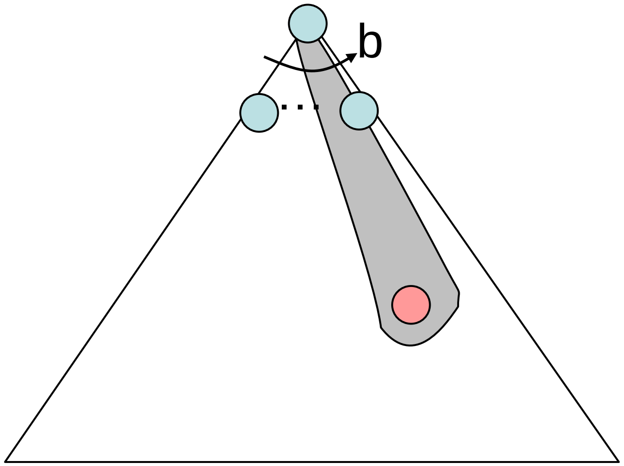
- Expand the node that seems closest…



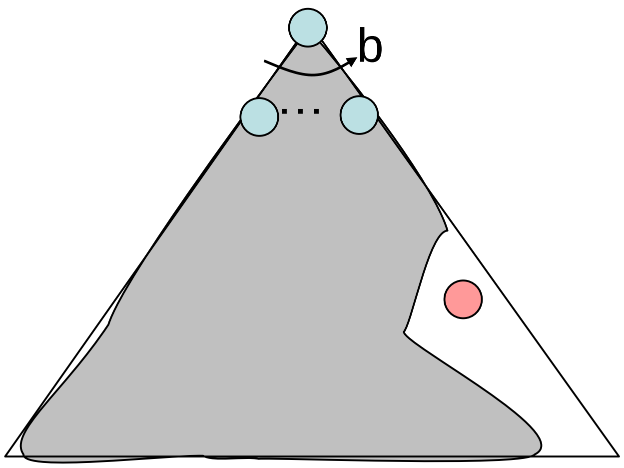
- What can go wrong?

Best First / Greedy Search

- A common case:
  - Best-first takes you straight to the (wrong) goal



- Worst-case: like a badly-guided DFS in the worst case
  - Can explore everything
  - Can get stuck in loops if no cycle checking



- Like DFS in completeness (finite states w/ cycle checking)

To Do:

- Look at the course website:
  - https://courses.cs.washington.edu/courses/cse573/19wi/
- Do the readings (Ch 3)
- Start PS1
  - START PS1 ASAP
- Try this visualization tool:
  - interactive search visualization