

Announcements

- Homework 1:

- stanford.ai-class.com
- To be done individually; due Sunday
- Review section: Tomorrow 11:00-12:00, Skilling Auditorium

- Project 1: Search

- cs221.stanford.edu
- Start early and ask questions
- You can work in groups of up to 3 people; due Thursday

- Contact us

- cs221.11.12@gmail.com

CS 221: Artificial Intelligence

Fall 2011

Lecture 2: Search

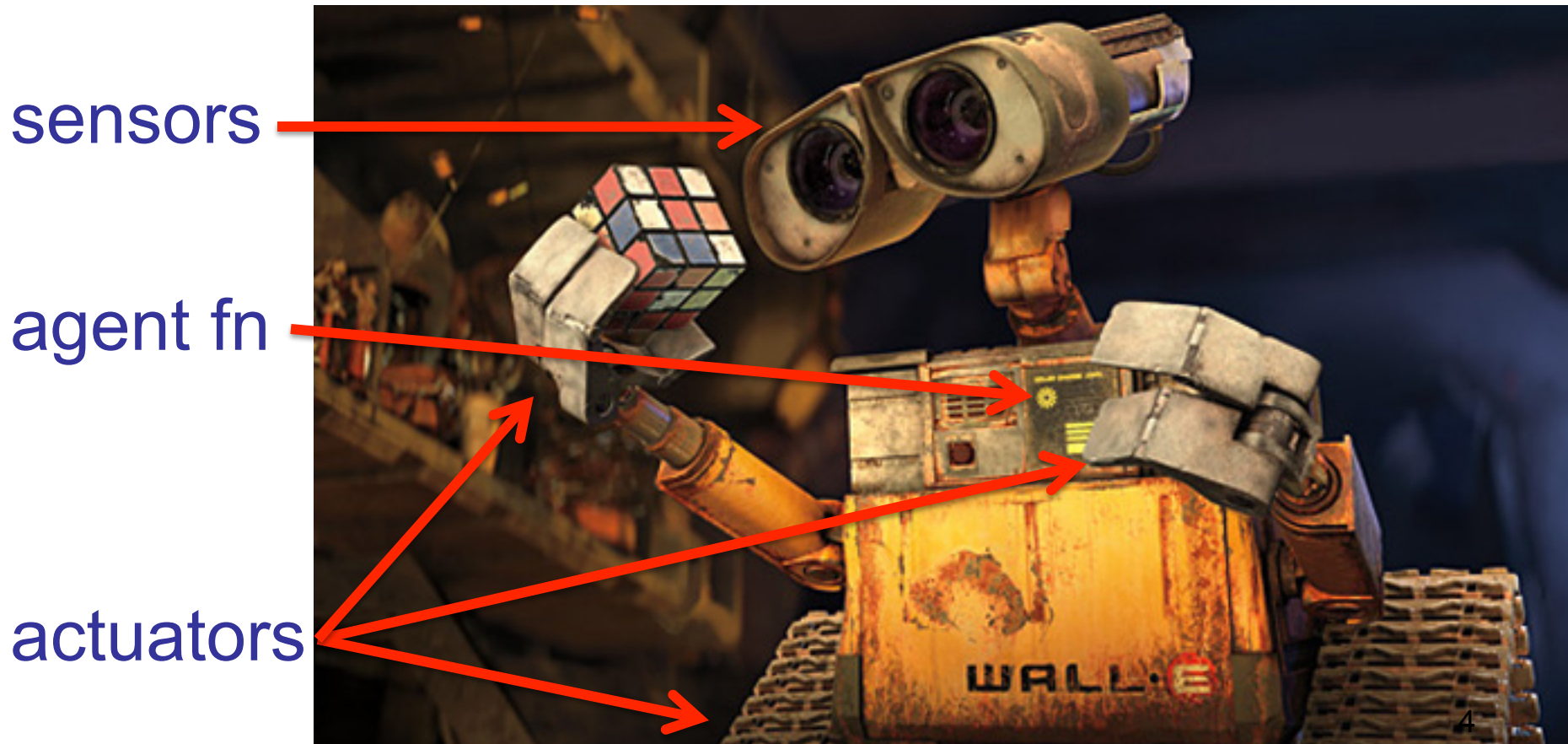
(Slides from Dan Klein,
with help from Stuart Russell, Andrew Moore, Teg Grenager,
Peter Norvig)

Outline

- Problem-solving agents (Book: 3.1)
 - Problem types and problem formulation
- Search trees and state space graphs (3.3)
- Uninformed search (3.4)
 - Depth-first, Breadth-first, Uniform cost
 - Search graphs
- Informed search (3.5)
 - Greedy search, A* search
 - Heuristics, admissibility

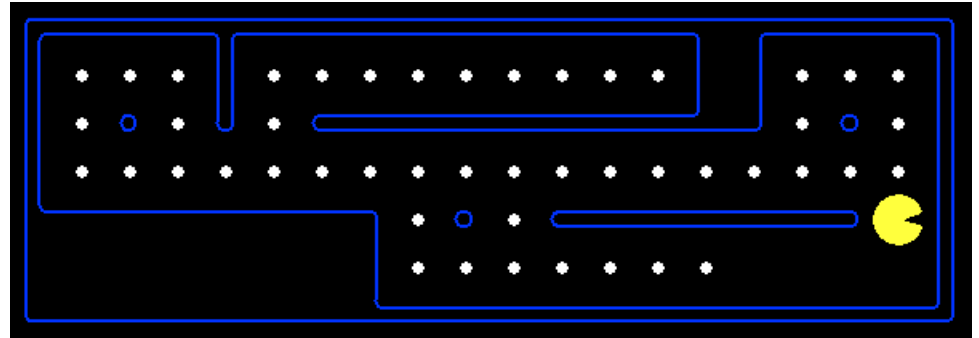
Agents

$$act = AgentFn(percept)$$

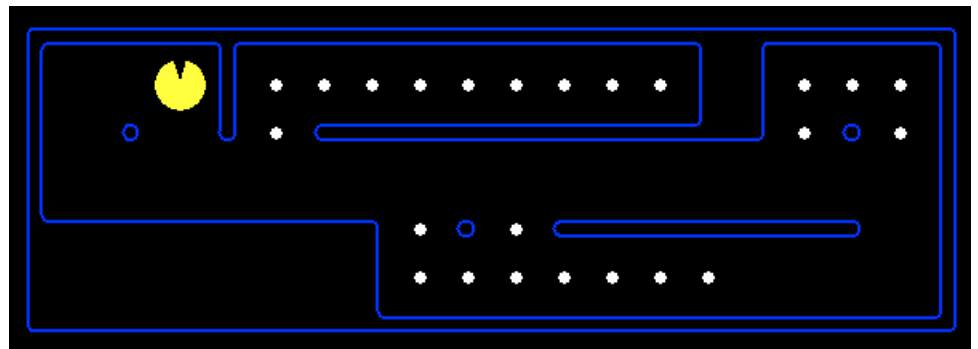


Reflex Agents

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

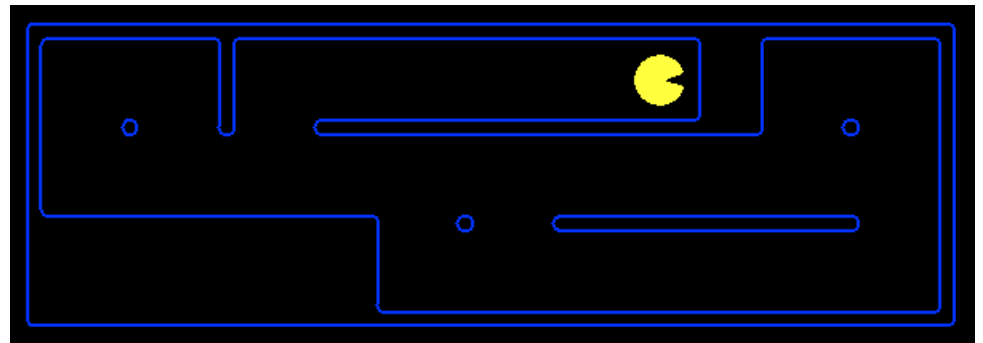
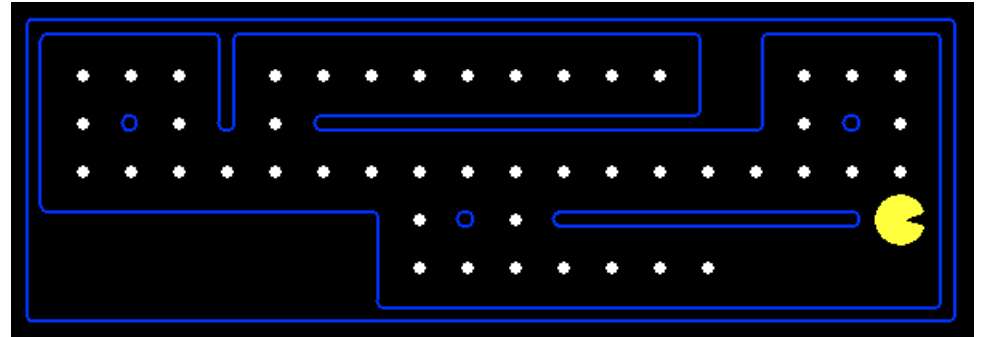


*Strategy: Dots: WNES
No Dots: NESW*



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**



WWGBAD?



WWGBAD?



Problem types

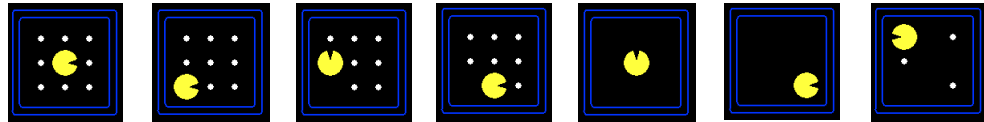
- Fully observable, deterministic
 - single-belief-state problem
- Non-observable
 - sensorless (conformant) problem
- Partially observable/non-deterministic
 - contingency problem
 - interleave search and execution
- Unknown state space
 - exploration problem
 - execution first



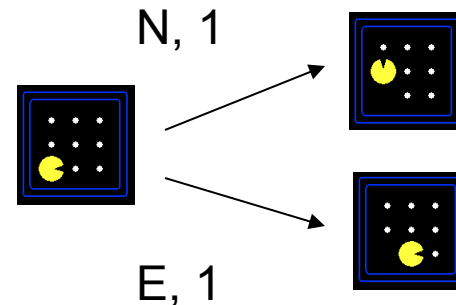
Search Problems

- A **search problem** consists of:

- A state space



- A transition model



- A start state, goal test, and path cost function
- A **solution** is a sequence of actions (a plan) which transforms the start state to a goal state

Transition Models

- Successor function

- $\text{Successors}(\text{state}) = \{(N, 1, \text{state}_N), (E, 1, \text{state}_E)\}$

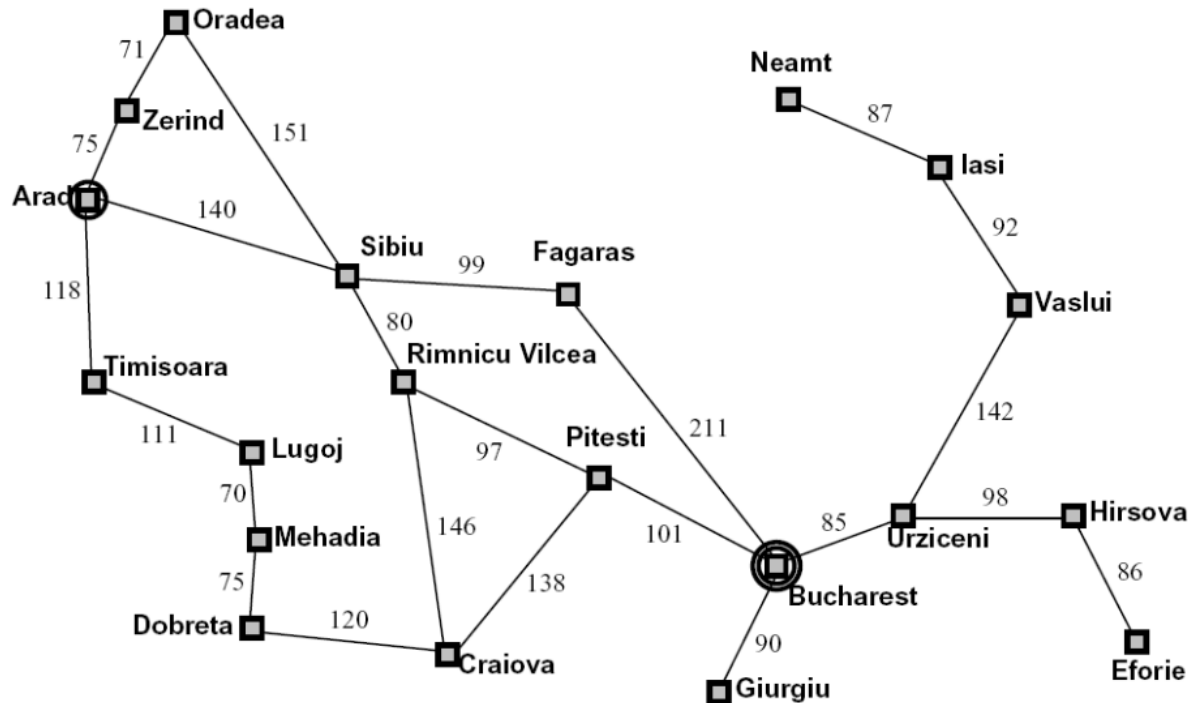
- Actions and Results

- $\text{Actions}(\text{state}) = \{N, E\}$

- $\text{Result}(\text{state}, N) = \text{state}_N$; $\text{Result}(\text{state}, E) = \text{state}_E$

- $\text{Cost}(\text{state}, N, \text{state}_N) = 1$; $\text{Cost}(\text{state}, E, \text{state}_E) = 1$

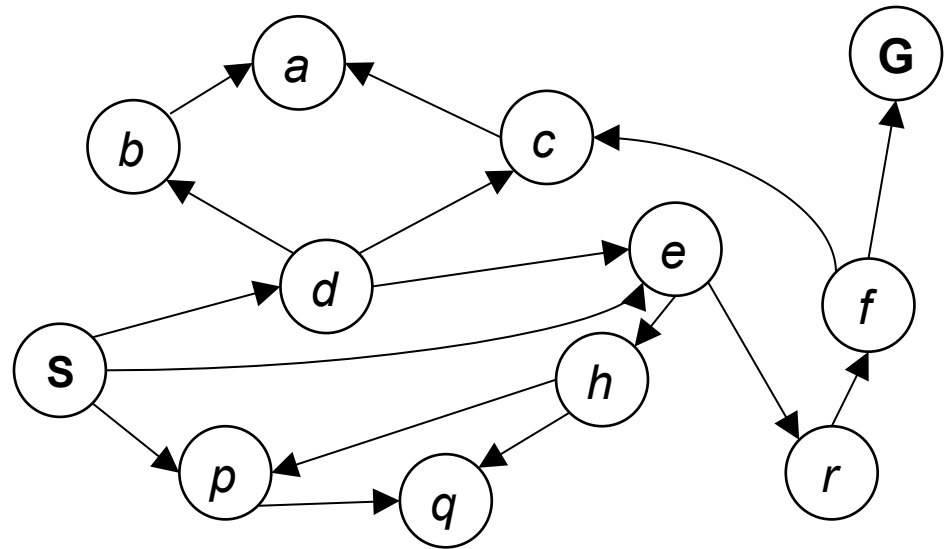
Example: Romania



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

State Space Graphs

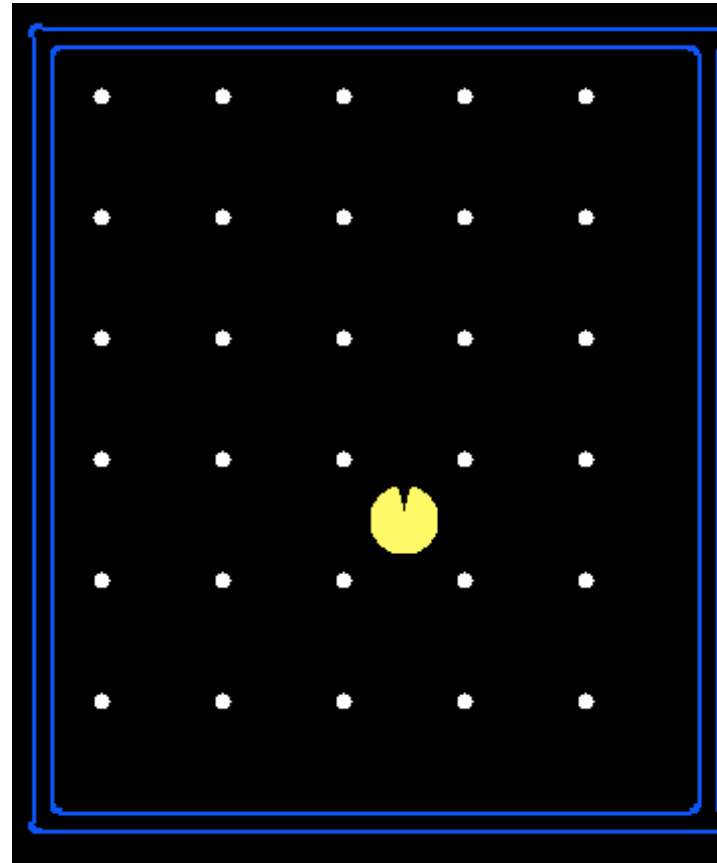
- State space graph: A mathematical representation of a search problem
 - For every search problem, there's a corresponding state space graph
 - The successor function is represented by arcs
- This can be large or infinite, so we won't create it in memory



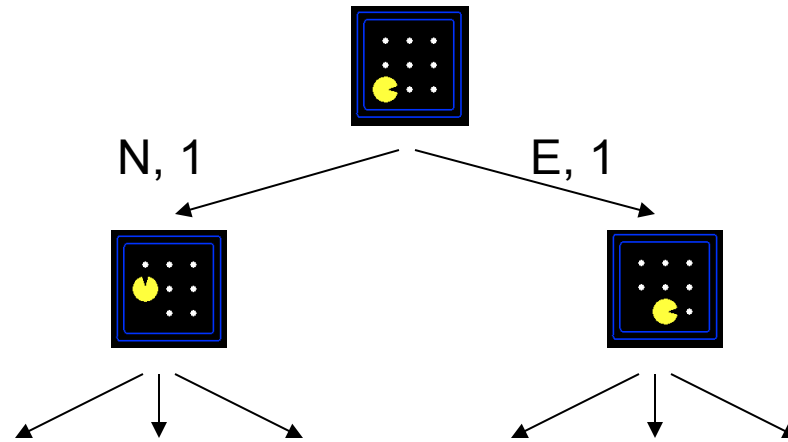
*Ridiculously tiny search graph
for a tiny search problem*

Exponential State Space Sizes

- Search Problem:
Eat all of the food
- Pacman positions:
 $10 \times 12 = 120$
- Food count: 30

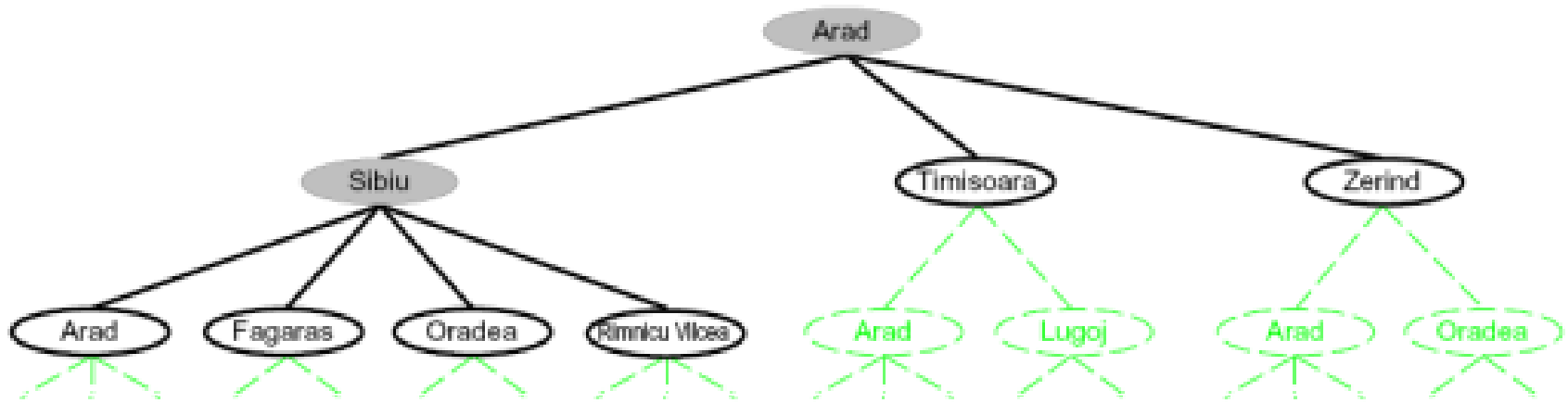


Search Trees



- A search tree:
 - This is a “what if” tree of plans and outcomes
 - Start state at the root node
 - Children correspond to successors
 - Nodes contain states, correspond to **paths** to those states
 - For most problems, we can never actually build the whole tree

Another Search Tree



■ Search:

- Expand out possible plans
- Maintain a **frontier** 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
```

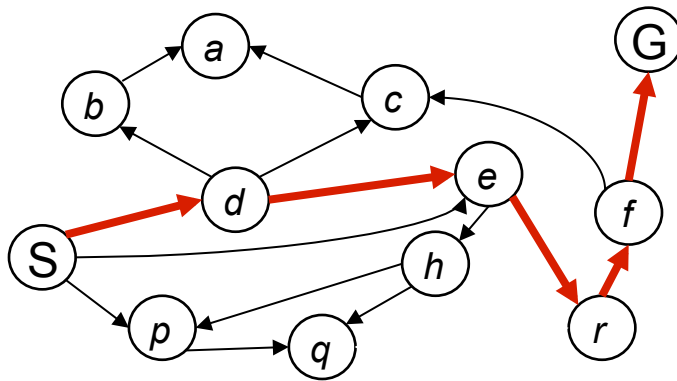
- Important ideas:

- Frontier (aka fringe)
- Expansion
- Exploration strategy

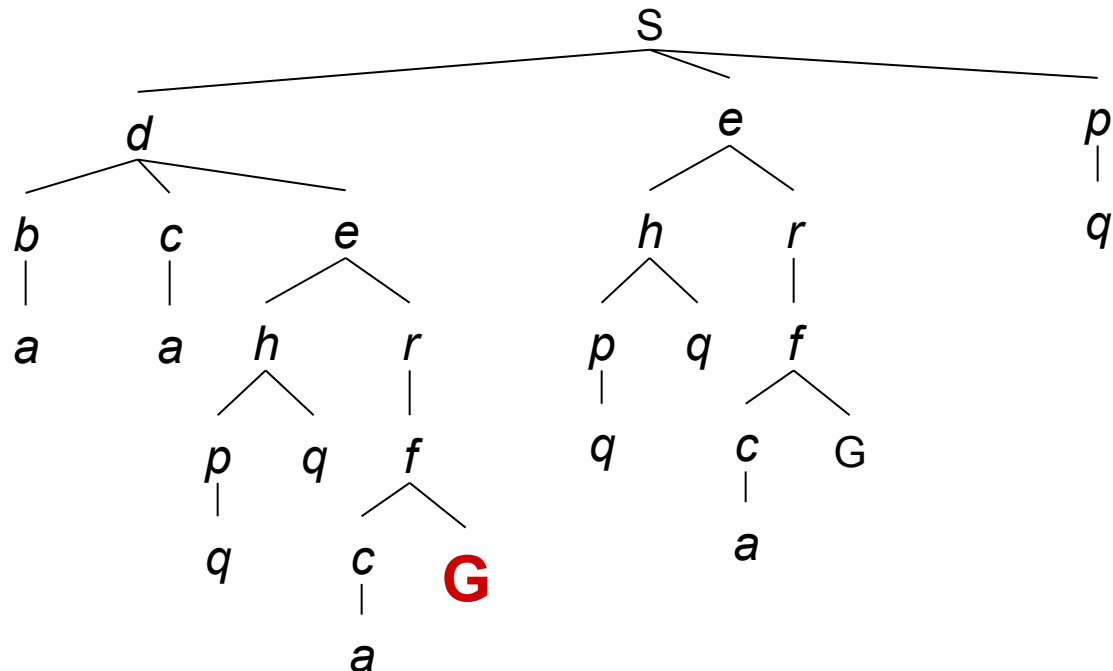
*Detailed pseudocode
is in the book!*

- Main question: which frontier nodes to explore?

State Space vs. Search Tree



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



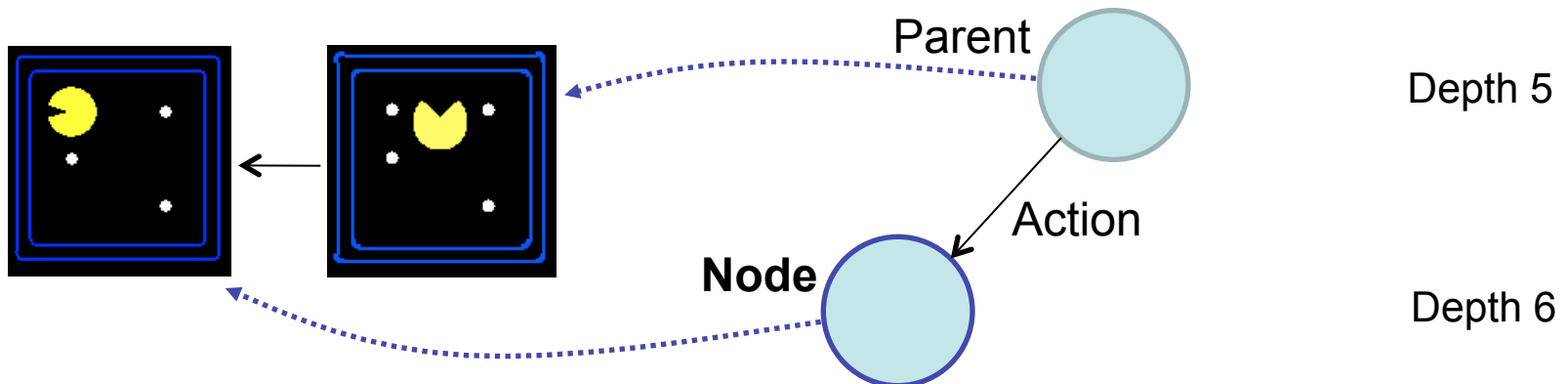
We construct both on demand – and we construct as little as possible.

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 paths
 - Represent a path (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**

State Space Graph

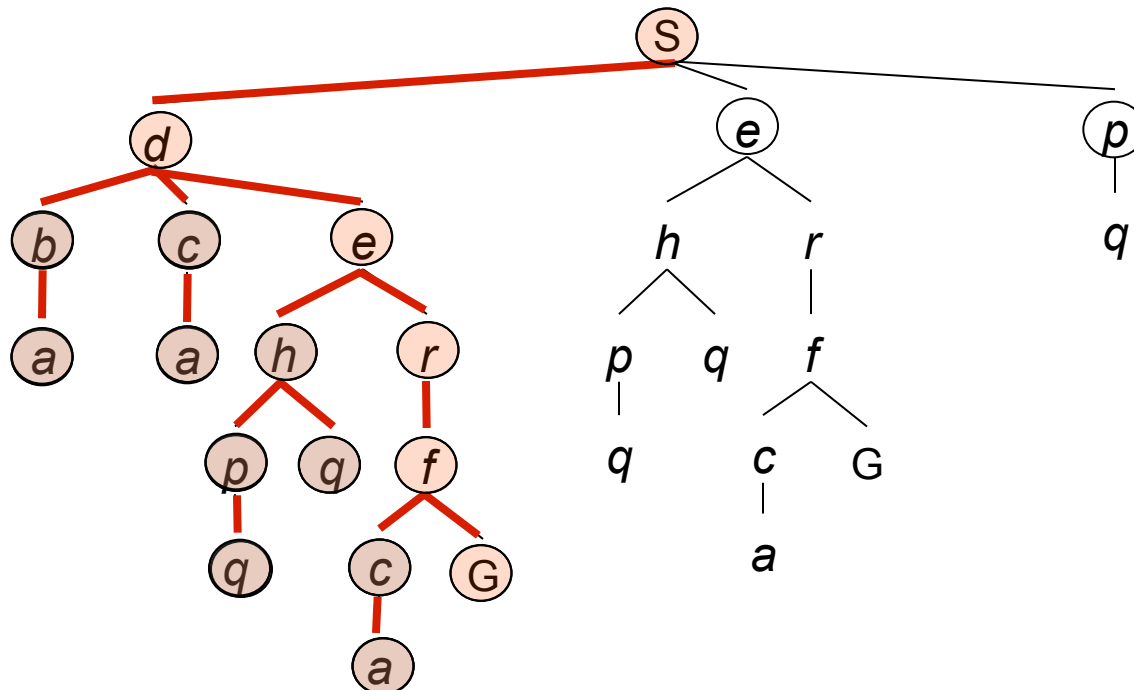
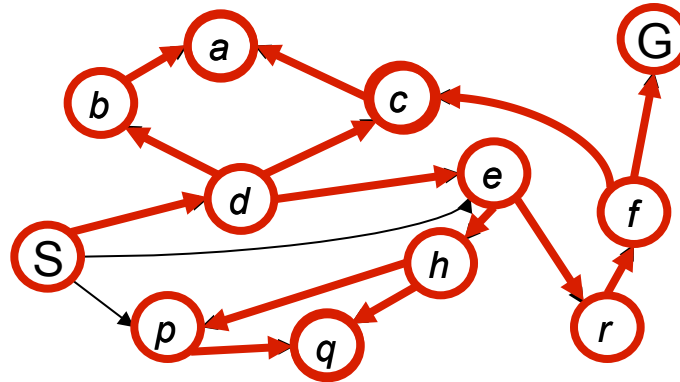
Search Tree



Depth First Search

Strategy: expand deepest node first

*Implementation:
Frontier is a LIFO stack*

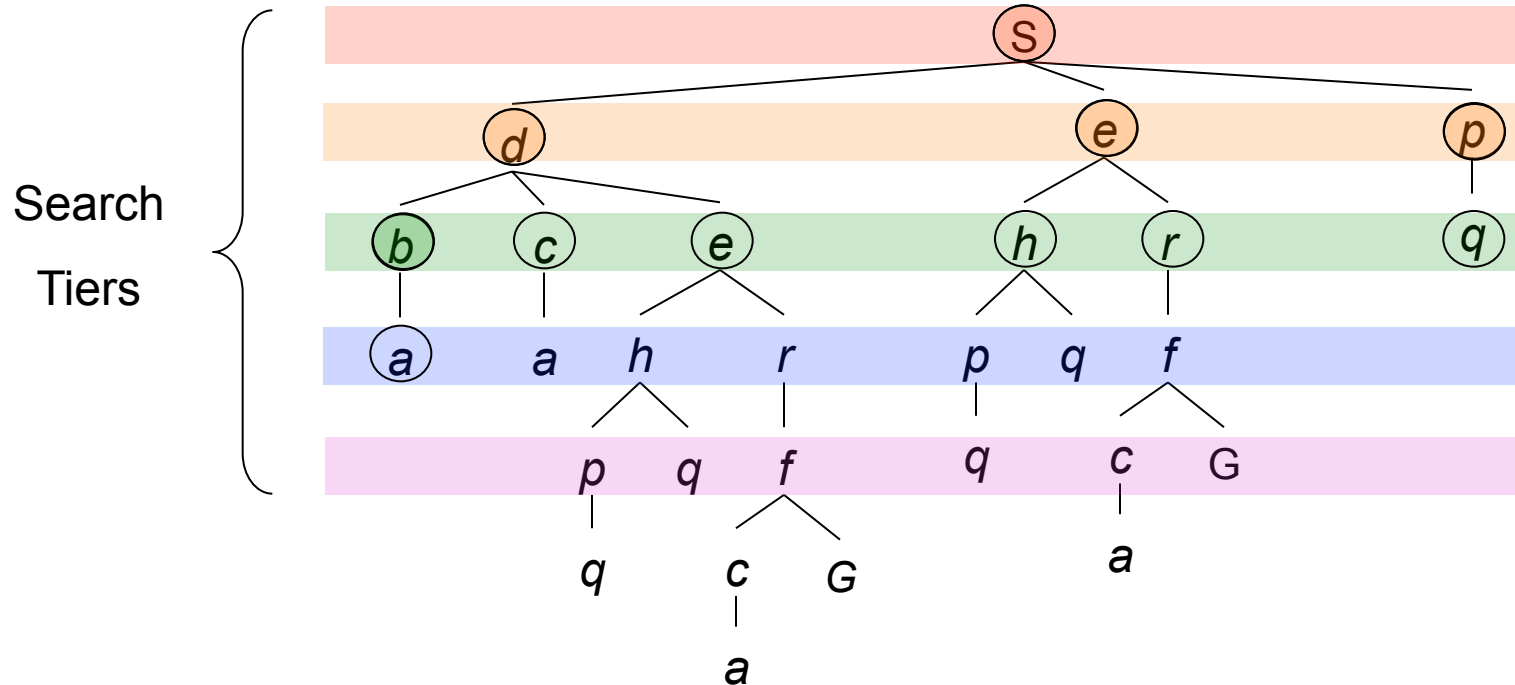
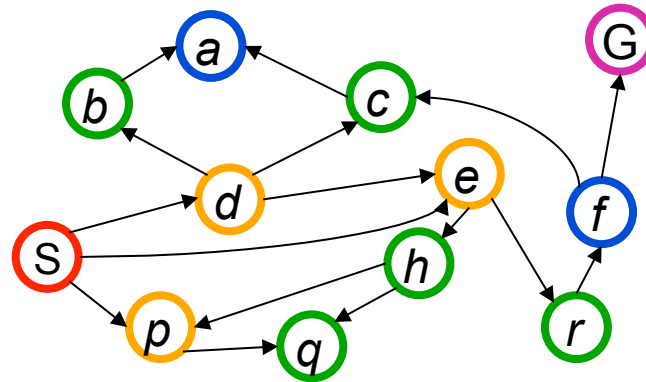


[demo: dfs]

Breadth First Search

Strategy: expand shallowest node first

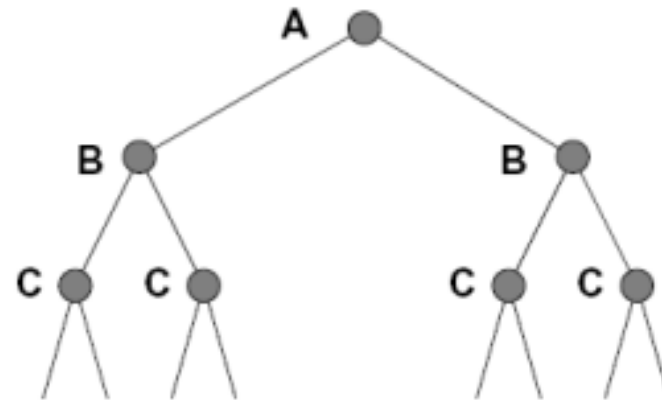
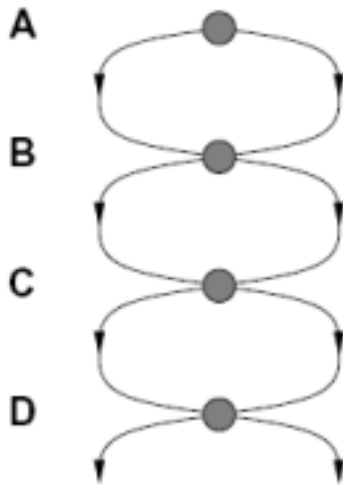
*Implementation:
Fringe is a FIFO queue*



[demo: bfs]

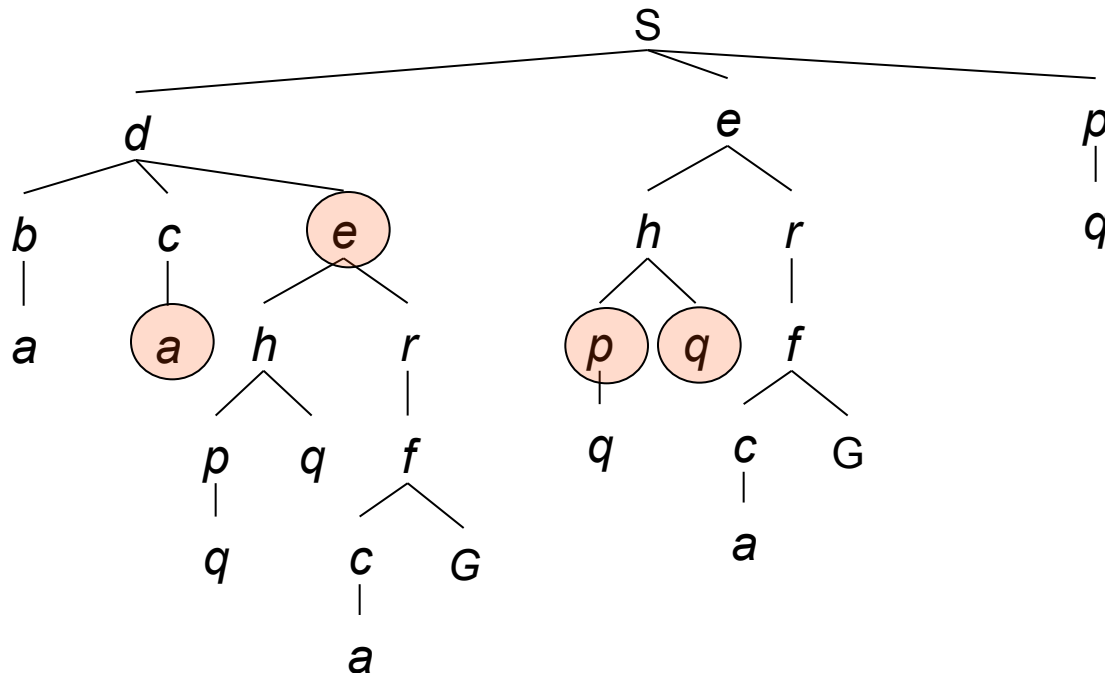
Santayana's Warning

- *“Those who cannot remember the past are condemned to repeat it.”* – George Santayana
- Failure to detect repeated states can cause exponentially more work (why?)



Graph Search

- In BFS, for example, we shouldn't bother expanding the circled nodes (why?)



Graph Search

- Very simple fix: never expand a state twice

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure

 initialize the frontier using the initial state of *problem*

initialize the explored set to be empty

loop do

if the frontier is empty **then return** failure

 choose a leaf node and remove it from the frontier

if the node contains a goal state **then return** the corresponding solution

add the node to the explored set

 expand the chosen node, adding the resulting nodes to the frontier

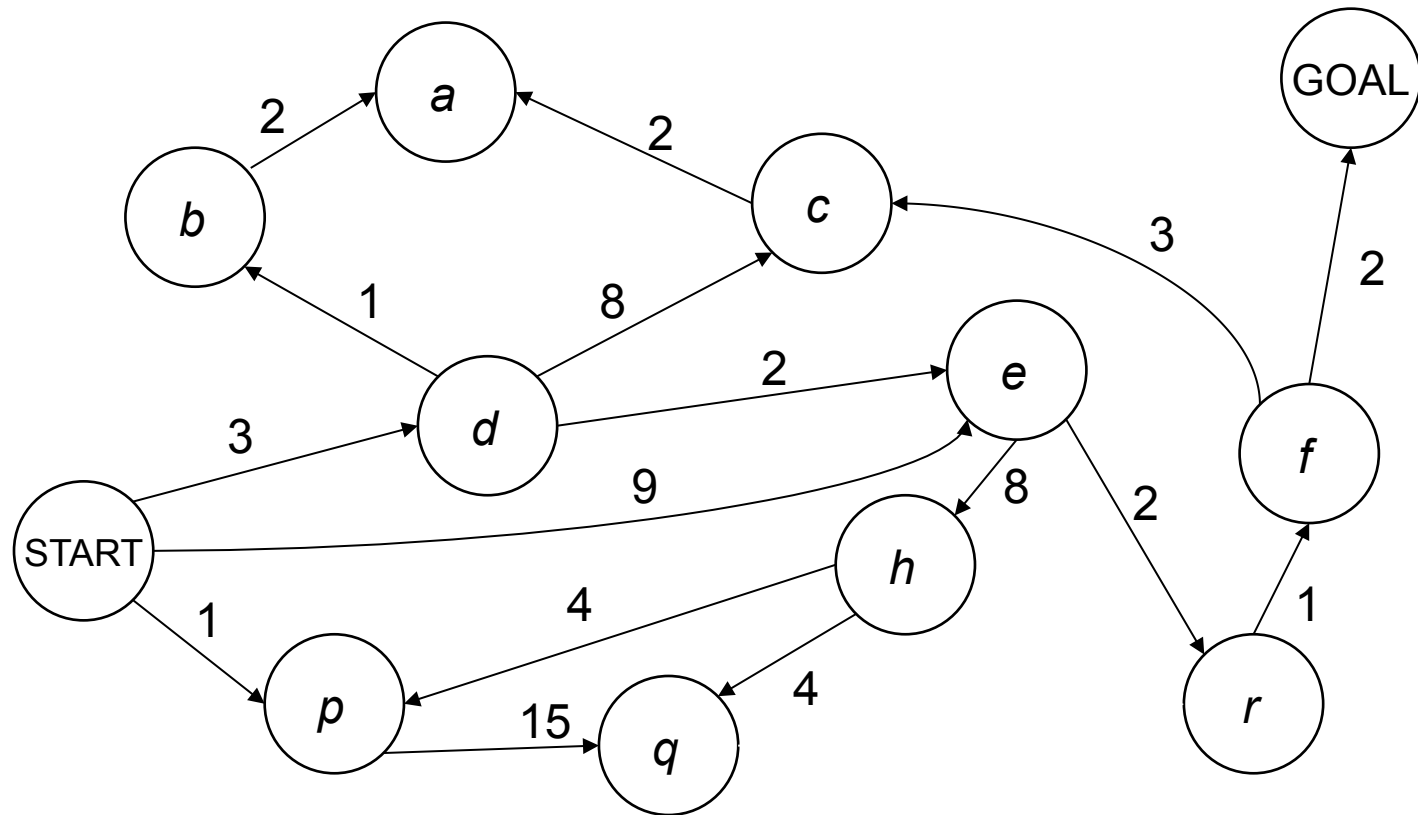
only if not in the frontier or explored set

- Can this wreck completeness? Lowest cost?

Graph Search Hints

- Graph search is almost always better than tree search (when not?)
- Implement explored as a dict or set
- Implement frontier as priority Q *and* set

Costs on Actions



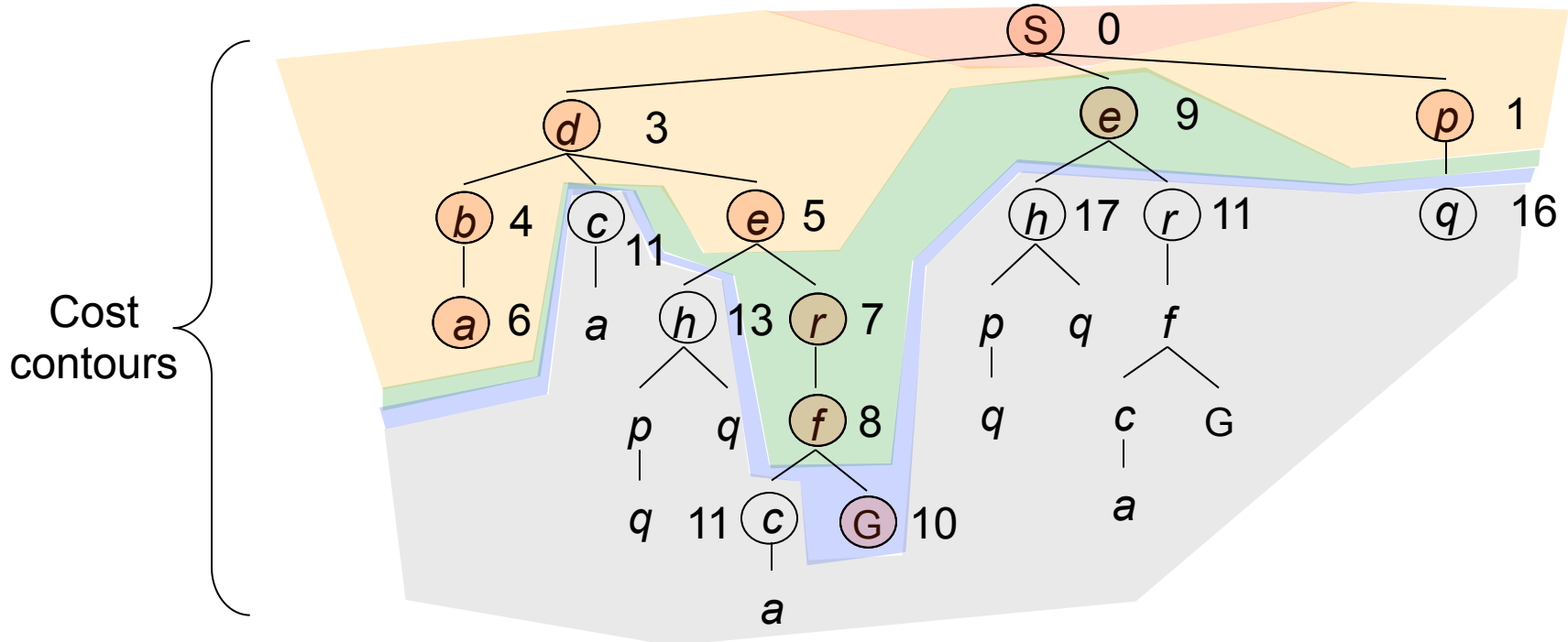
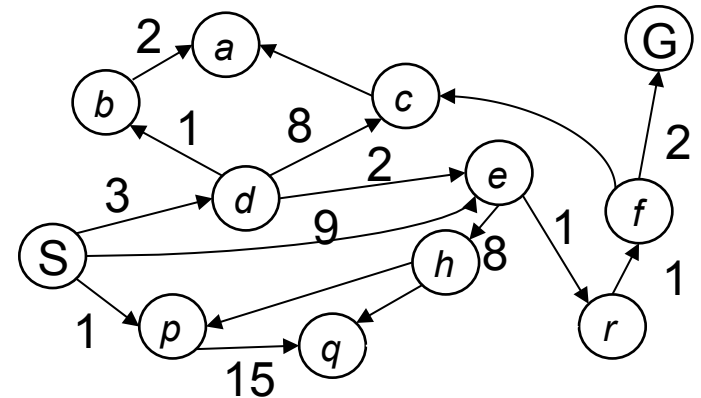
Notice that BFS finds the shortest path in terms of number of transitions. It does not find the least-cost path.

We will quickly cover an algorithm which does find the least-cost path.

Uniform Cost Search

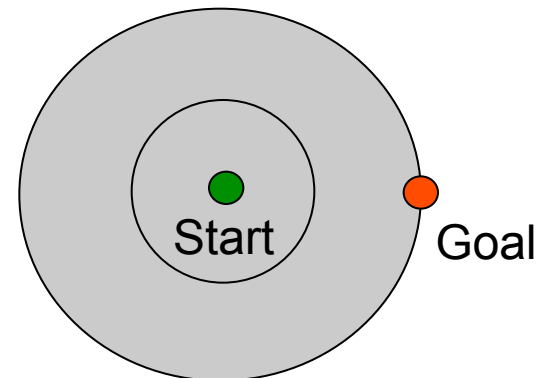
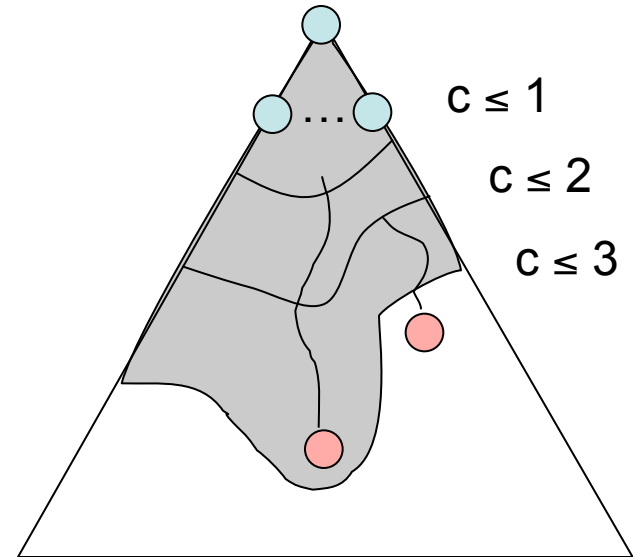
Expand cheapest node first:

Frontier is a priority queue



Uniform Cost Issues

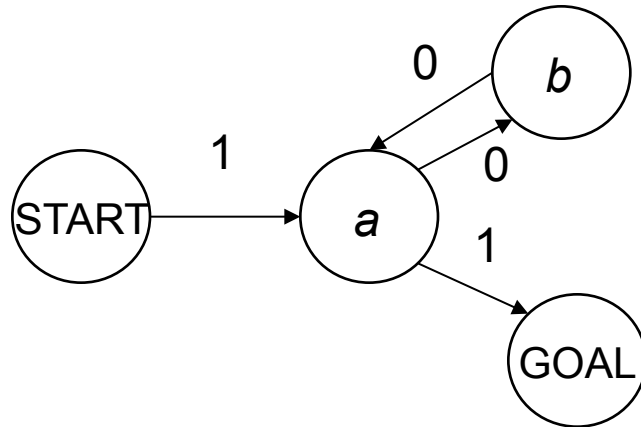
- Remember: explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
 - Explores options in every “direction”
 - No information about goal location



[demos: ucs, ucs2]

Uniform Cost Search

- What will UCS do for this graph?



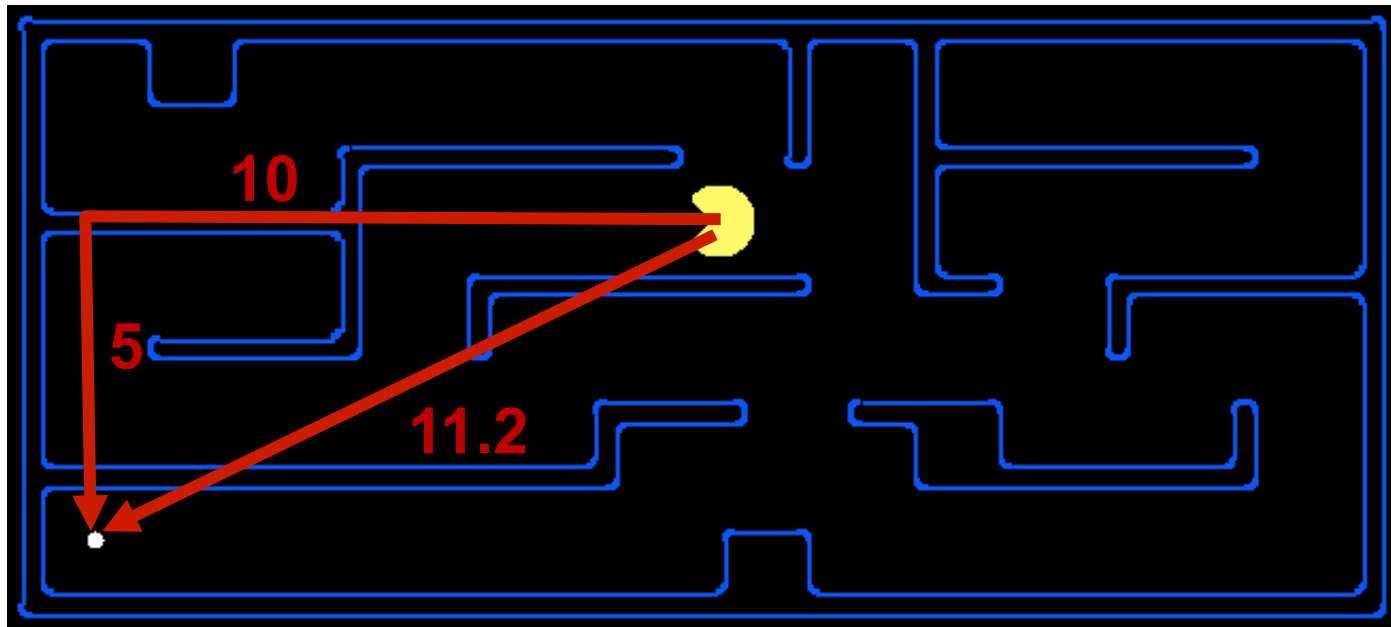
- What does this mean for completeness?

AI Lesson

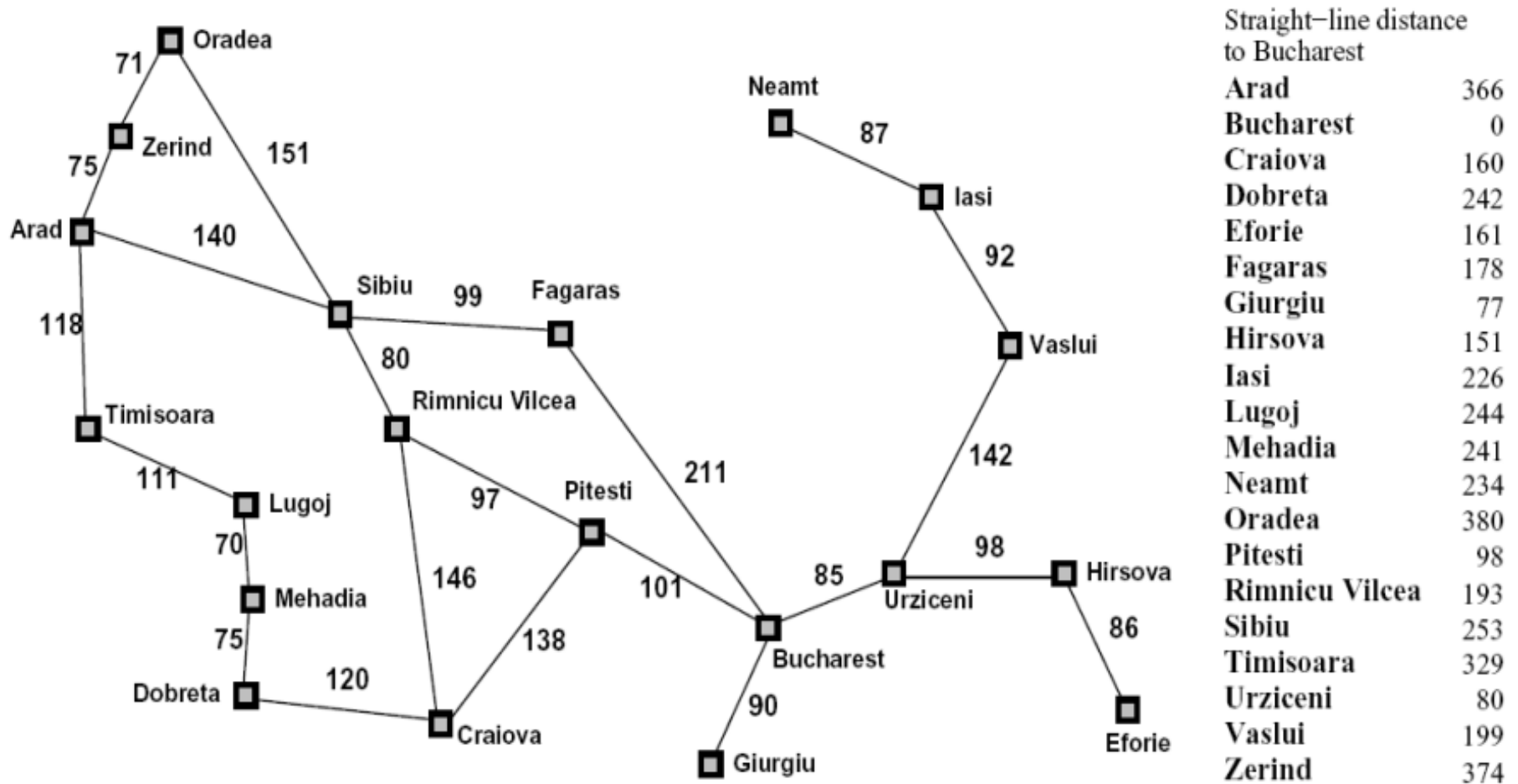
*To do more,
Know more*

Search Heuristics

- Any *estimate* of how close a state is to a goal
- Designed for a particular search problem
- Examples: Manhattan distance, Euclidean distance

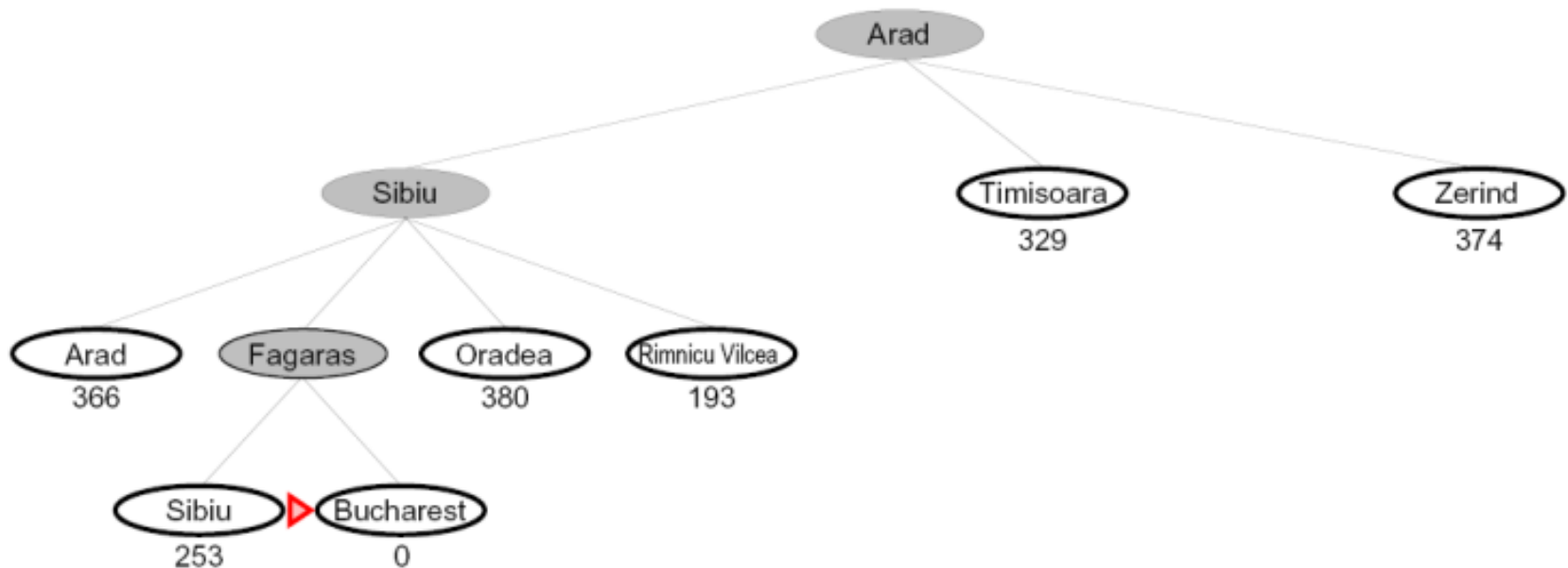


Heuristics



Greedy Best First Search

- Expand the node that *seems* closest to goal...



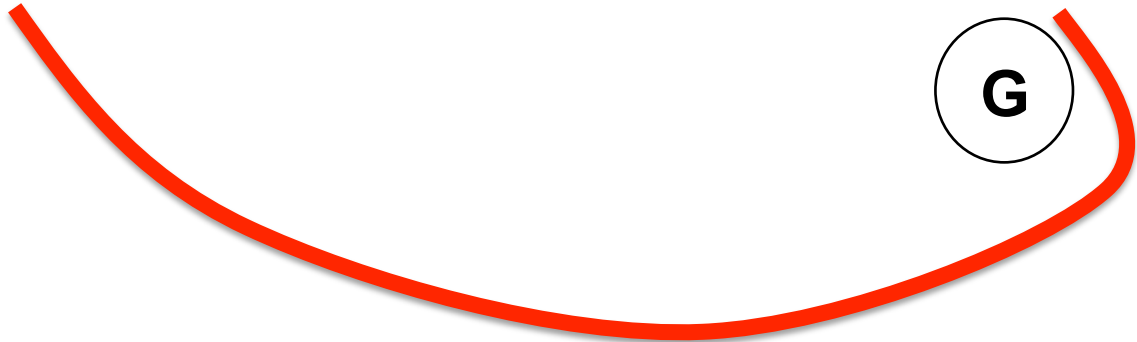
- What can go wrong?

[demos: gbf1, gbf2]

Greedy goes wrong

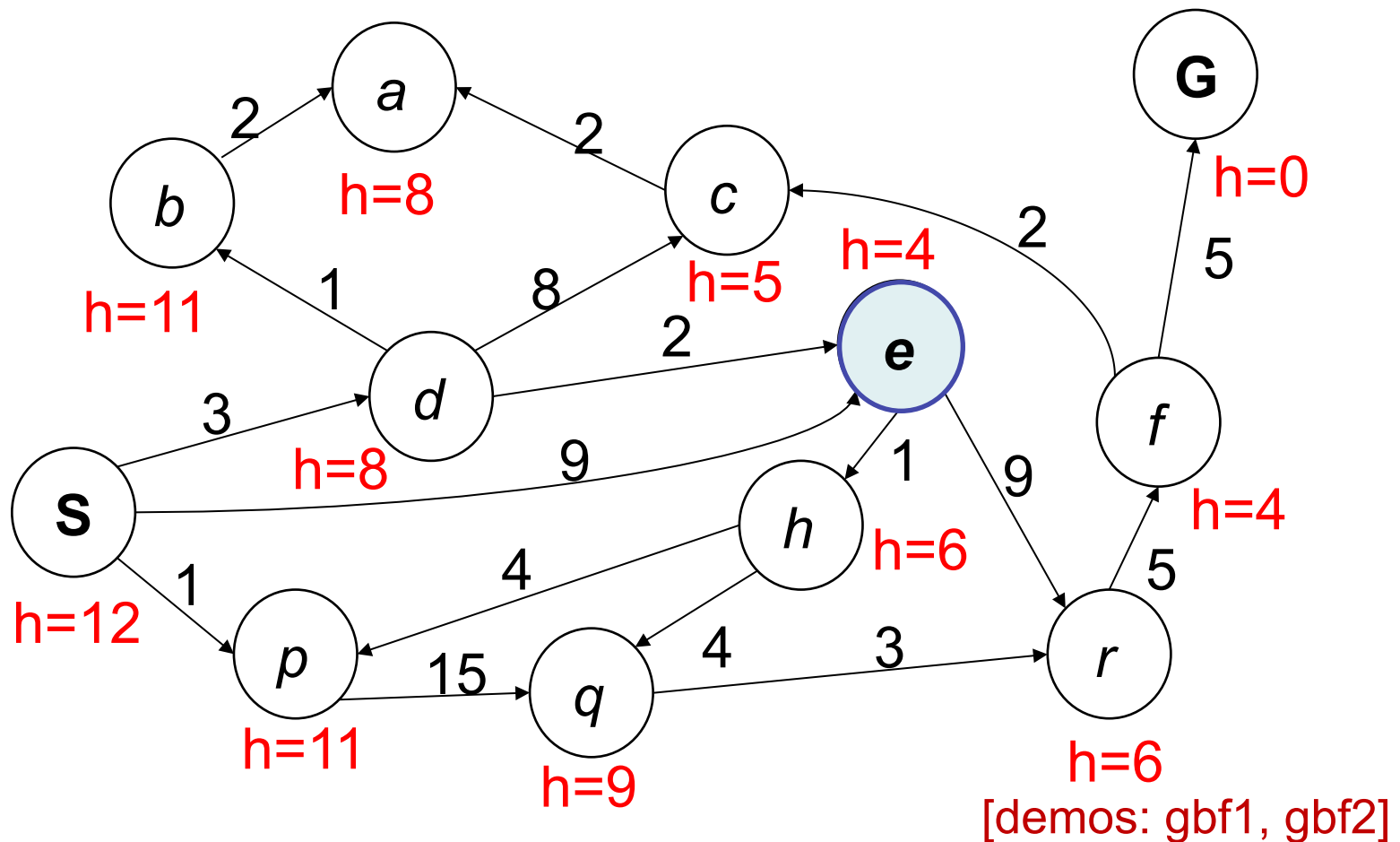
S

G



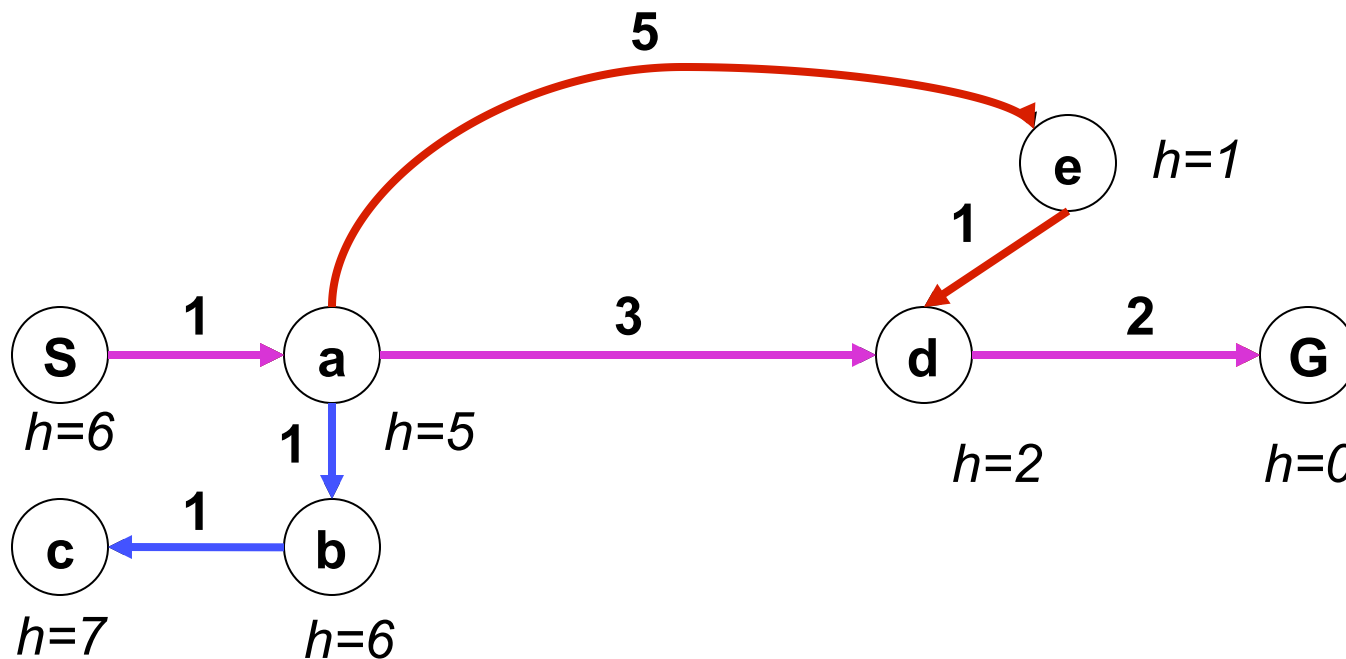
Best First / Greedy Search

- Strategy: expand the closest node to the goal



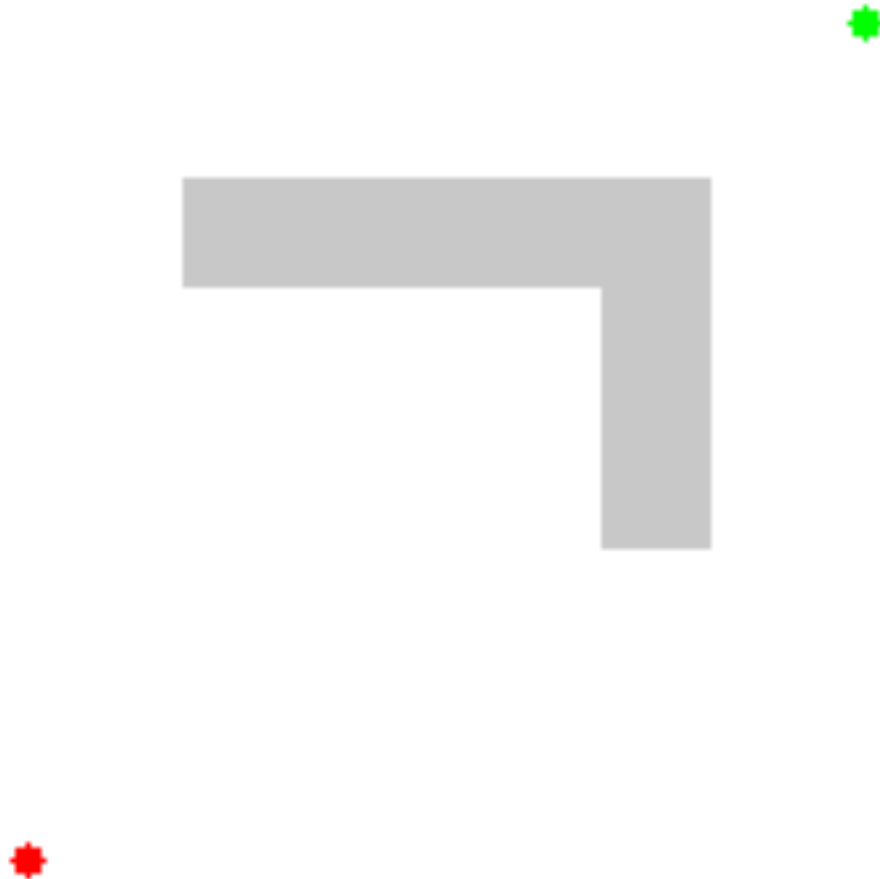
Combining UCS and Greedy

- **Uniform-cost** orders by path cost, or *backward cost* $g(n)$
- **Best-first** orders by distance to goal, or *forward cost* $h(n)$



- **A* Search** orders by the sum: $f(n) = g(n) + h(n)$

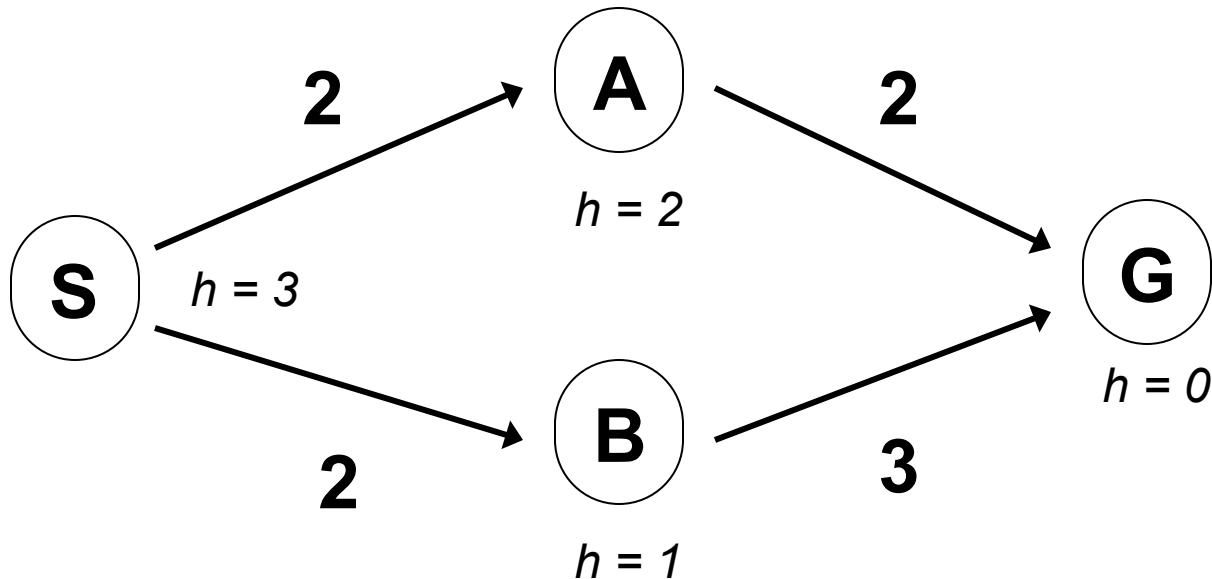
A* Search Progress



source: wikipedia page for A* Algorithm; by Subh83

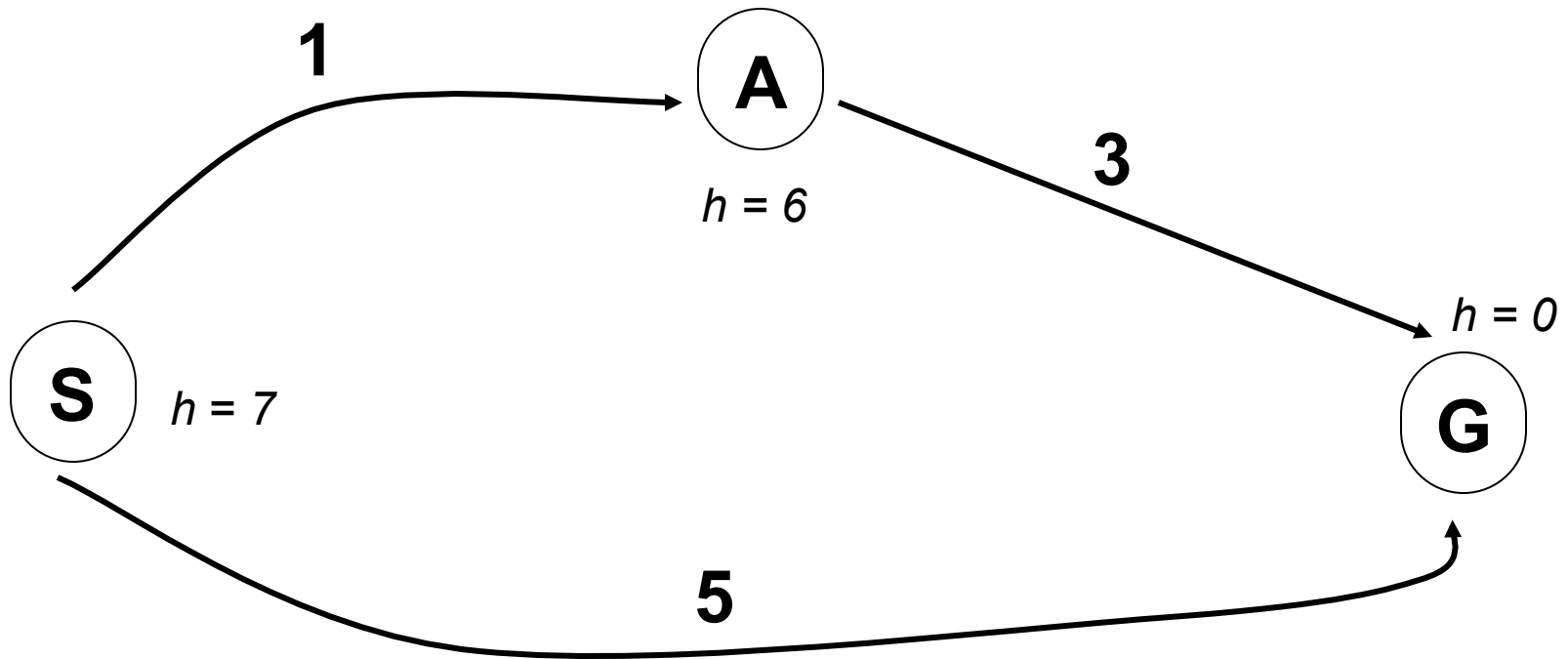
When should A* terminate?

- Should we stop when we enqueue a goal?



- No: only stop when we dequeue a goal

Is A* Optimal?



- What went wrong?
- Actual bad path cost (5) < estimate good path cost (1+6)
- We need estimates ($h=7$) to be less than actual (5) costs!

Admissible Heuristics

- A heuristic h is *admissible* (optimistic) if:

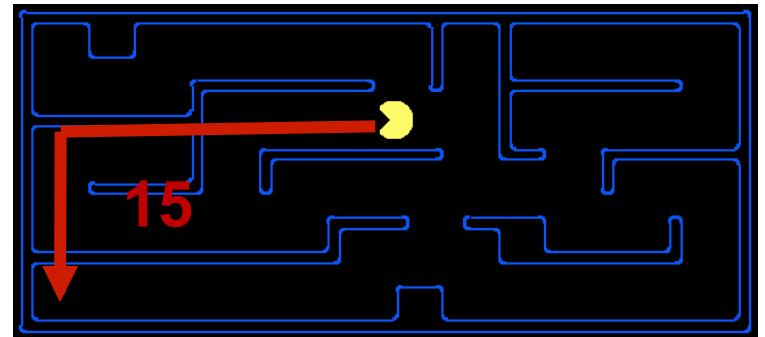
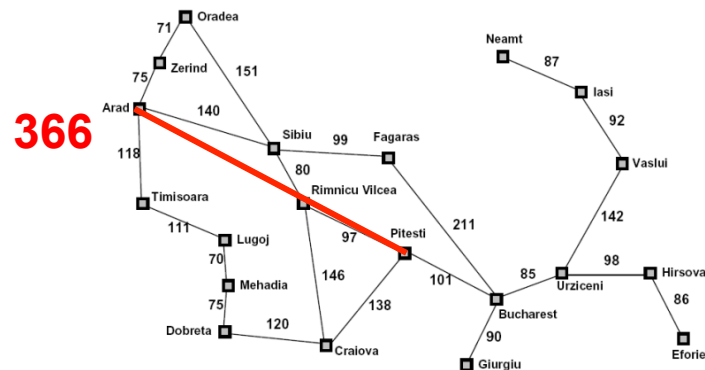
$$h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal

Never overestimate!

Creating Admissible Heuristics

- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics
- Often, admissible heuristics are solutions to *relaxed problems*, where new actions are available

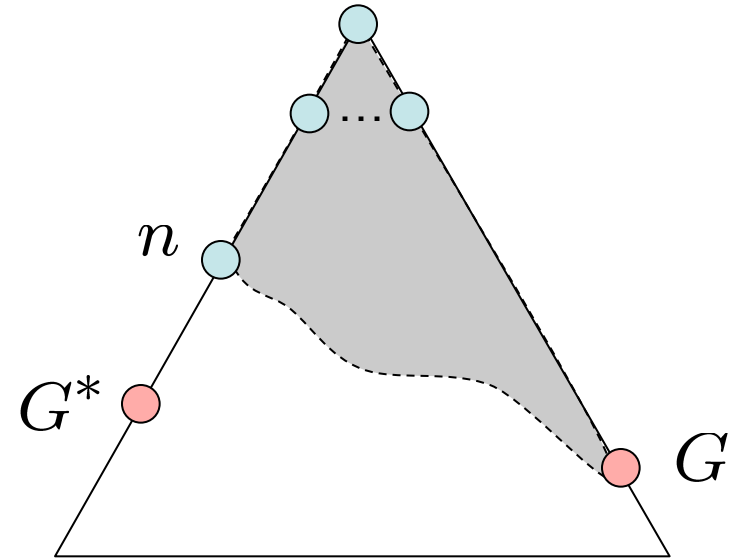


- Inadmissible heuristics are often useful too (why?)

Optimality of A*: Blocking

Notation:

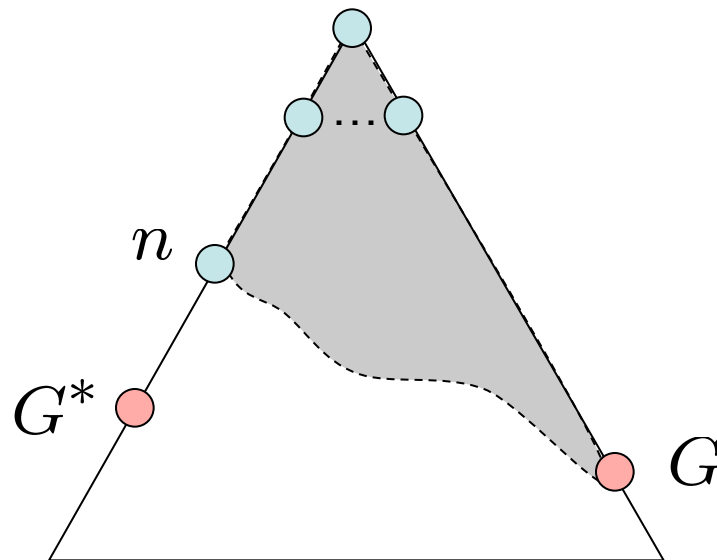
- $g(n)$ = cost to node n
- $h(n)$ = estimated cost from n to the nearest goal (heuristic)
- $f(n) = g(n) + h(n)$ = estimated total cost via n
- G^* : a lowest cost goal node
- G : another goal node



Optimality of A*: Blocking

Proof:

- What could go wrong?
- We'd have to have to pop a suboptimal goal G off the frontier before G^*
- This can't happen:
 - Imagine a suboptimal goal G is on the queue
 - Some node n which is a subpath of G^* must also be on the frontier (why?)
 - n will be popped before G



$$f(n) = g(n) + h(n)$$

$$g(n) + h(n) \leq g(G^*)$$

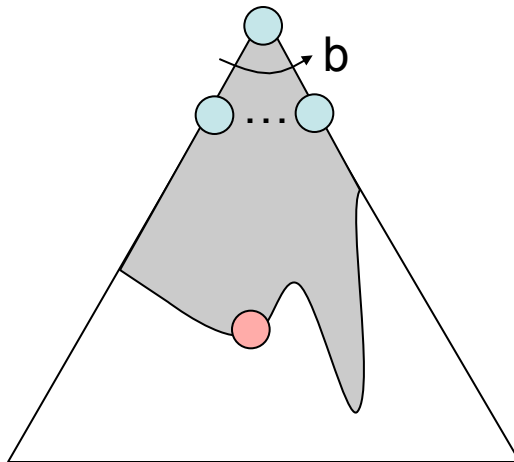
$$g(G^*) < g(G)$$

$$g(G) = f(G)$$

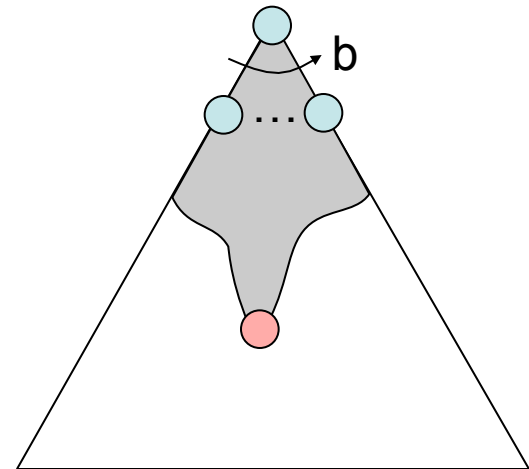
$$f(n) < f(G)$$

Properties of A^*

Uniform-Cost

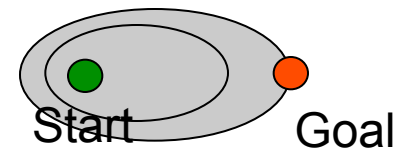
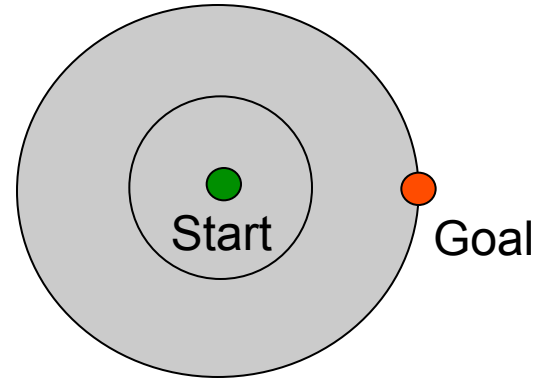


A^*



UCS vs A* Contours

- Uniform-cost expanded in all directions
- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



Example: 8 Puzzle

| | | |
|---|---|---|
| 7 | 2 | 4 |
| 5 | | 6 |
| 8 | 3 | 1 |

Start State

| | | |
|---|---|---|
| | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

Goal State

- What are the states?
- How many states?
- What are the actions?
- What states can I reach from the start state?
- What should the costs be?

8 Puzzle

- Heuristic: Number tiles misplaced
- Why is it admissible?

| | | |
|---|---|---|
| 7 | 2 | 4 |
| 5 | | 6 |
| 8 | 3 | 1 |

Start State

| | | |
|---|---|---|
| | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

Goal State

- $h(\text{start}) =$
- 8
- This is a **relaxed-problem** heuristic:

| Average nodes expanded when optimal path has length... | | | |
|--|------------|------------|-------------------|
| | ...4 steps | ...8 steps | ...12 steps |
| UCS | 112 | 6,300 | 3.6×10^6 |
| TILES | 13 | 39 | 227 |

Move **A** to **B** if ~~adjacent(A,B)~~ and ~~empty(B)~~

8 Puzzle

- What if we had an easier 8-puzzle where any tile could slide one step at any time, ignoring other tiles?
- Total *Manhattan* distance
- Why admissible?

| | | |
|---|---|---|
| 7 | 2 | 4 |
| 5 | | 6 |
| 8 | 3 | 1 |

Start State

| | | |
|---|---|---|
| | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

Goal State

- $h(\text{start}) =$
- $3 + 1 + 2 + \dots = 18$

- Relaxed problem:

| | Average nodes expanded when optimal path has length... | | |
|-----------|--|------------|-------------|
| | ...4 steps | ...8 steps | ...12 steps |
| TILES | 13 | 39 | 227 |
| MANHATTAN | 12 | 25 | 73 |

Move **A** to **B** if adjacent(**A**,**B**) and ~~empty(**B**)~~

Trivial Heuristics, Dominance

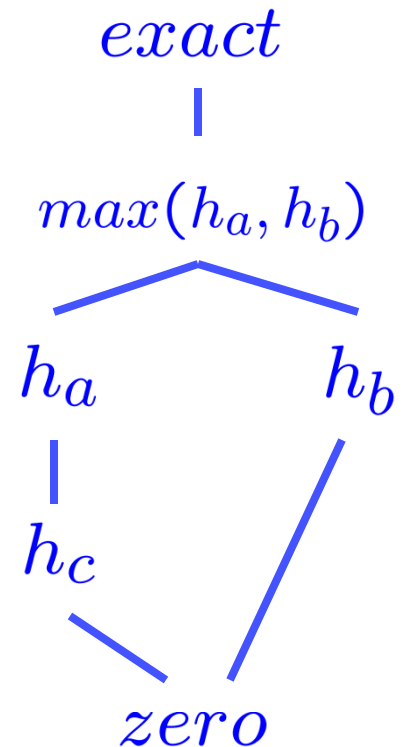
- Dominance: $h_a \geq h_c$ if

$$\forall n : h_a(n) \geq h_c(n)$$

- Heuristics form a semi-lattice:
 - Max of admissible heuristics is admissible

$$h(n) = \max(h_a(n), h_b(n))$$

- Trivial heuristics
 - Bottom of lattice is the zero heuristic (what does this give us?)
 - Top of lattice is the exact heuristic



Other A* Applications

- Path finding / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- ...

Summary: A^*

- A^* uses both backward costs, $g(n)$, and (estimates of) forward costs, $h(n)$
- A^* is optimal with admissible heuristics
- Heuristic design is key: often use relaxed problems
- A^* is not the final word in search algorithms (but it does get the final word for today)