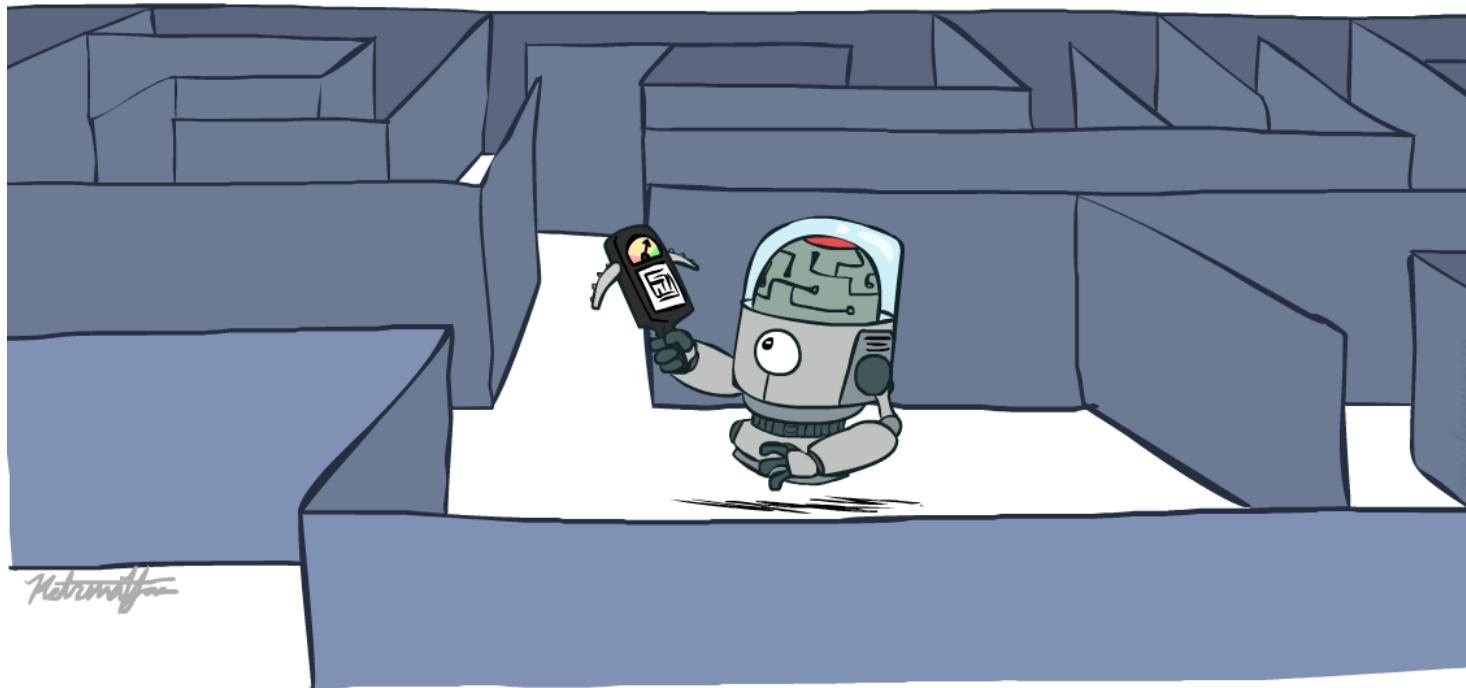


Announcements

- Oops! We didn't finish last lecture.
 - Mea culpa! I'm used to smaller tutorials and talks where you can schedule more time and the goal is to ensure that everyone understands everything.
- Class has been expanded by 50. Please do not sit on steps!
- Homework 1: Search
 - On edX. Due Tue 9/6 at 11:59pm. Instant grading, many submits allowed.
- Project 1: Search
 - Released today! Due Friday 9/16 at 5pm.
 - Start early and ask questions. It's longer than most!
- Week 1 notes released Friday
- Sections
 - You can go to any. Exam practice session begin Monday 9/5.
- Contest!

CS 188: Artificial Intelligence

Uninformed and Informed Search



Instructors: Josh Hug & Adam Janin

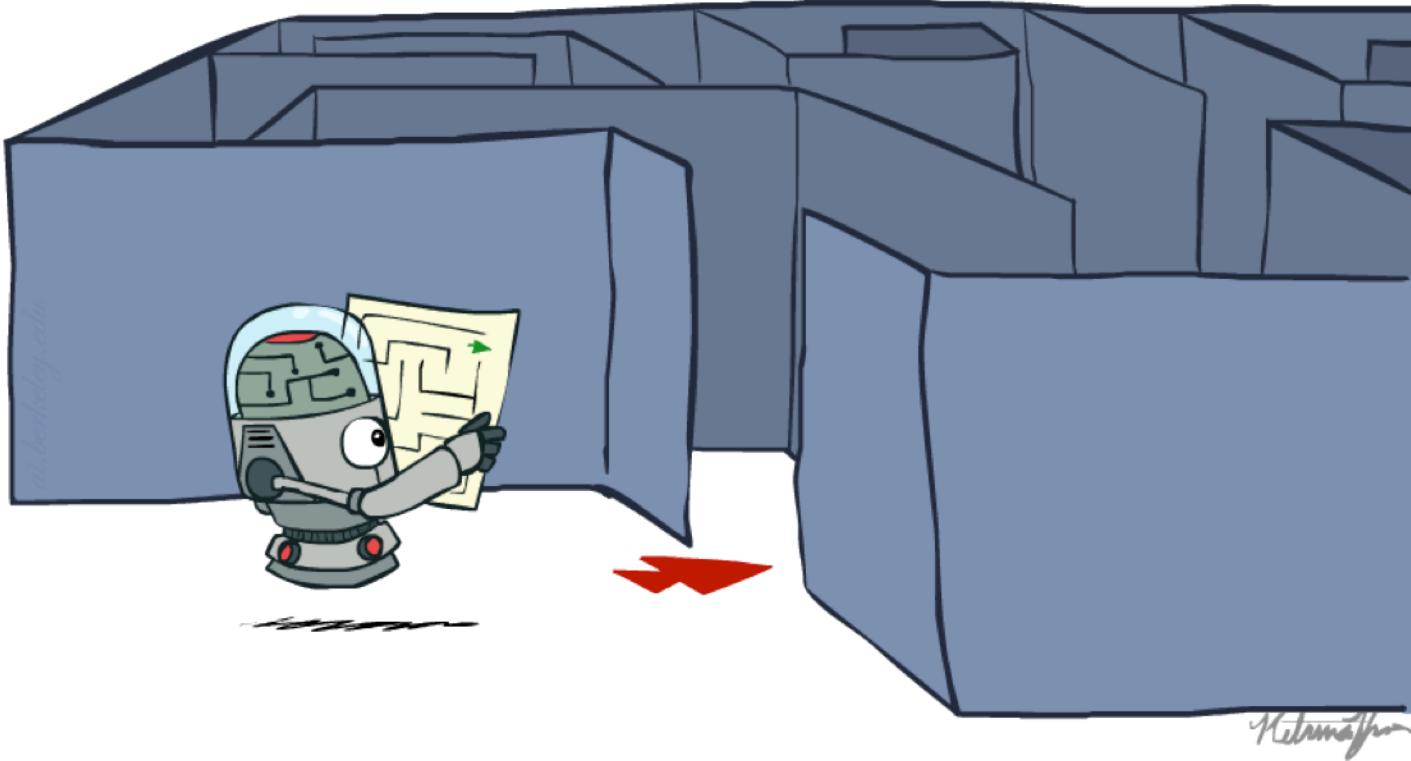
University of California, Berkeley

Today

- Finish up Uninformed Search
- Informed Search
 - Heuristics
 - Greedy Search
 - A* Search
- Graph Search



Recap: Search



Recap: Search

- **Search problem:**

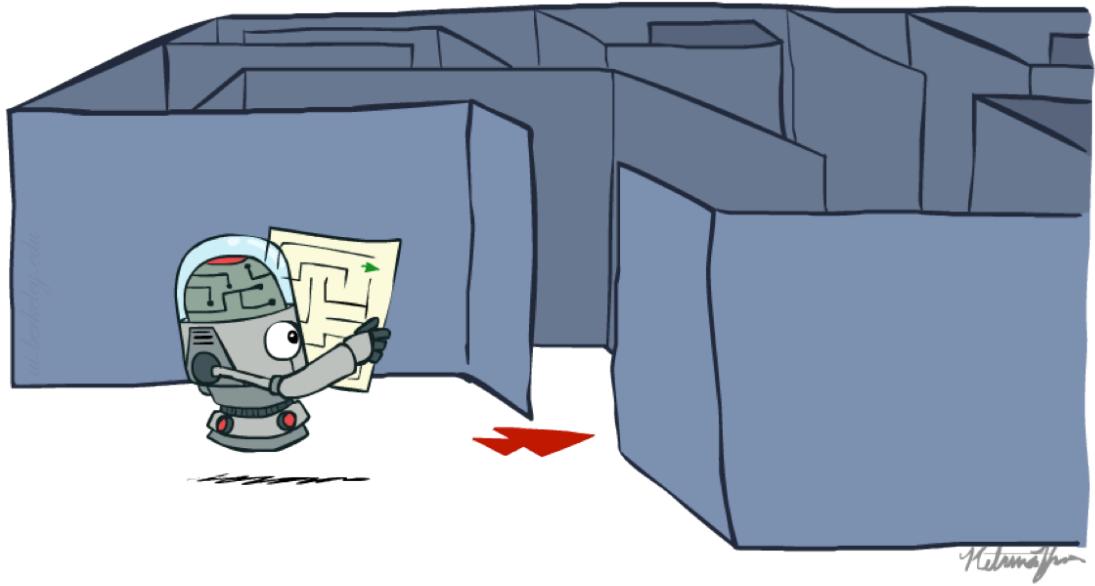
- States (configurations of the world)
- Actions and costs
- Successor function (world dynamics)
- Start state and goal test

- **Search tree:**

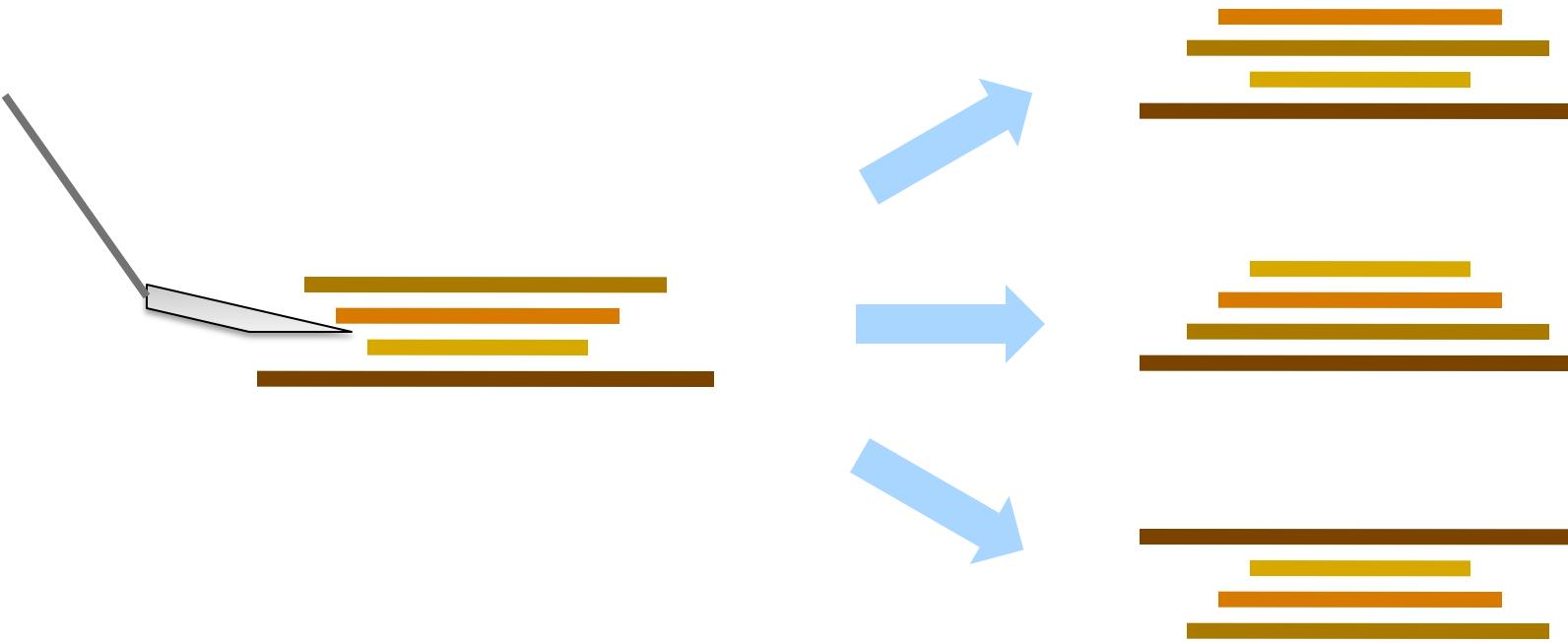
- Nodes: represent plans for reaching states
- Plans have costs (sum of action costs)

- **Search algorithm:**

- Systematically builds a search tree
- Chooses an ordering of the fringe (unexplored nodes)
- Optimal: finds least-cost plans



Example: Pancake Problem



Cost: Number of pancakes flipped

Example: Pancake Problem

BOUNDS FOR SORTING BY PREFIX REVERSAL

William H. GATES

Microsoft, Albuquerque, New Mexico

Christos H. PAPADIMITRIOU*†

Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.

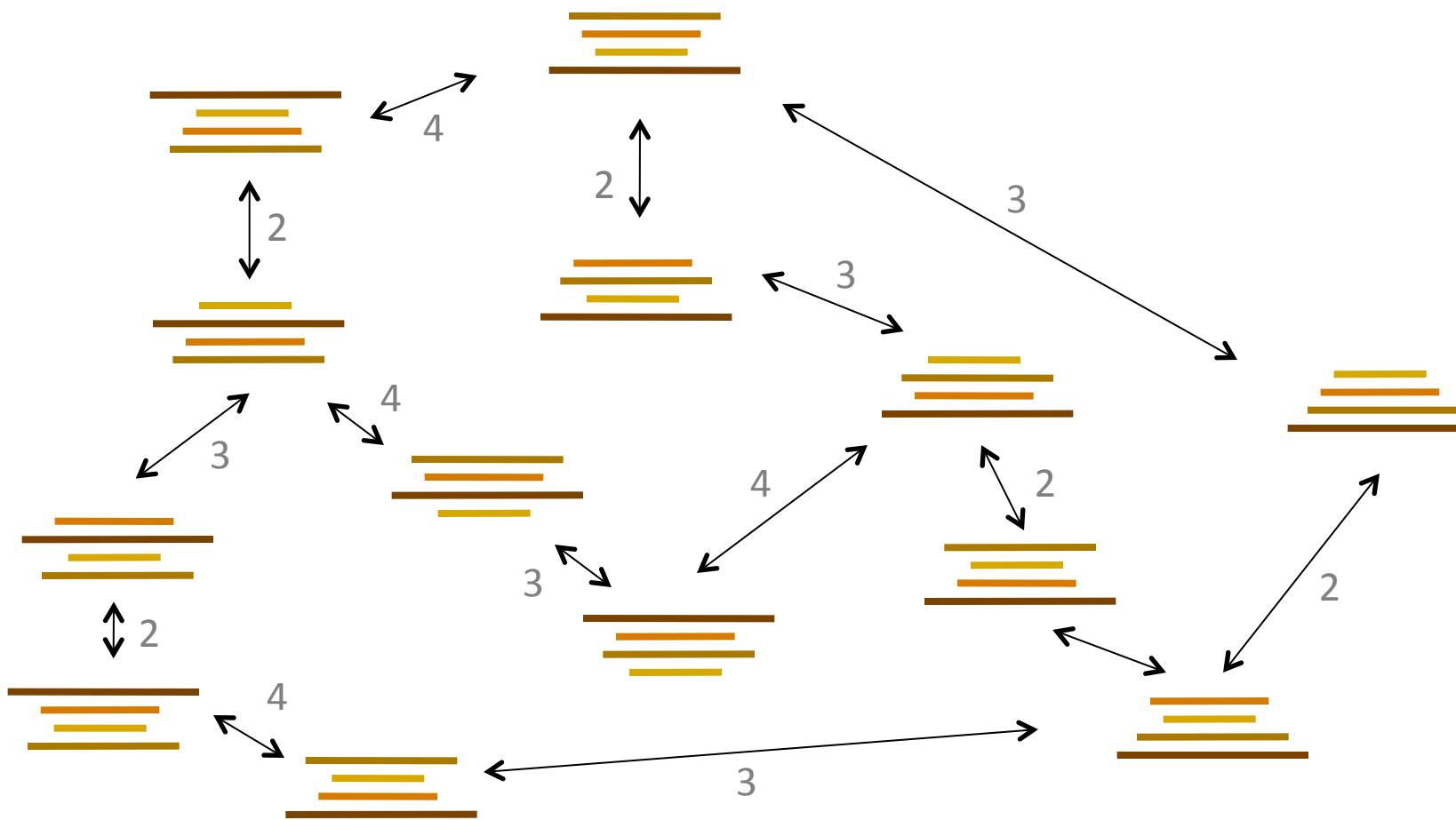
Received 18 January 1978

Revised 28 August 1978

For a permutation σ of the integers from 1 to n , let $f(\sigma)$ be the smallest number of prefix reversals that will transform σ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all σ in (the symmetric group) S_n . We show that $f(n) \leq (5n + 5)/3$, and that $f(n) \geq 17n/16$ for n a multiple of 16. If, furthermore, each integer is required to participate in an even number of reversed prefixes, the corresponding function $g(n)$ is shown to obey $3n/2 - 1 \leq g(n) \leq 2n + 3$.

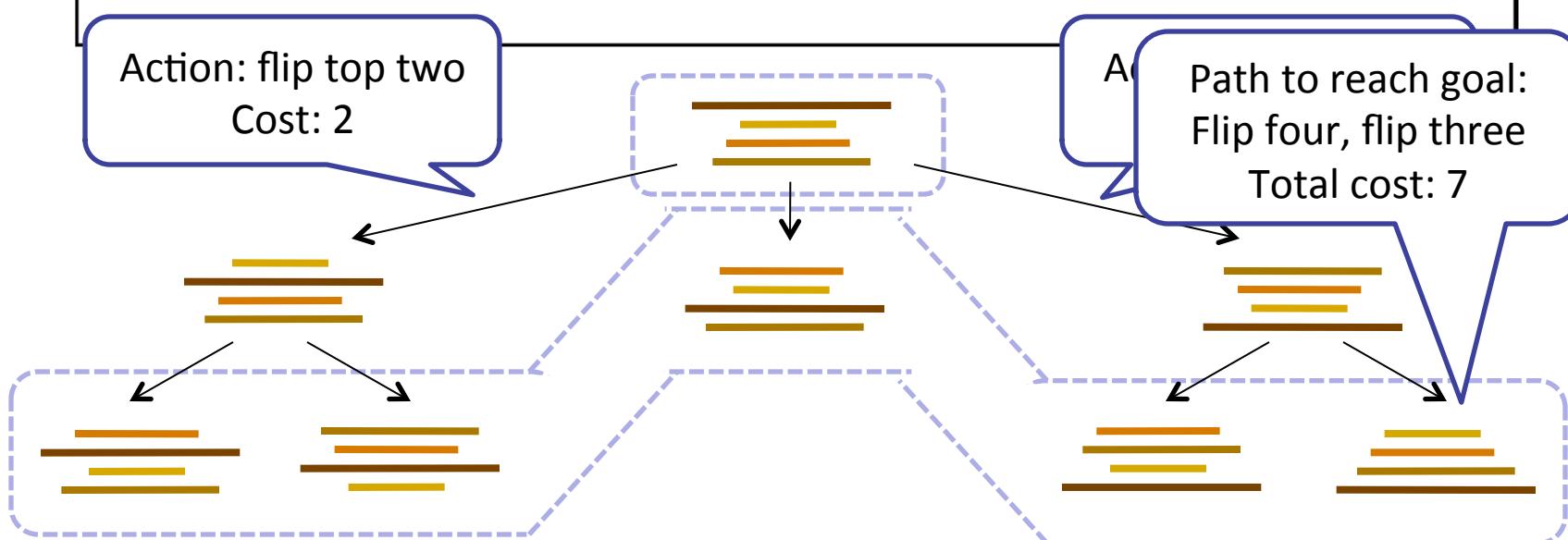
Example: Pancake Problem

State space graph with costs as weights



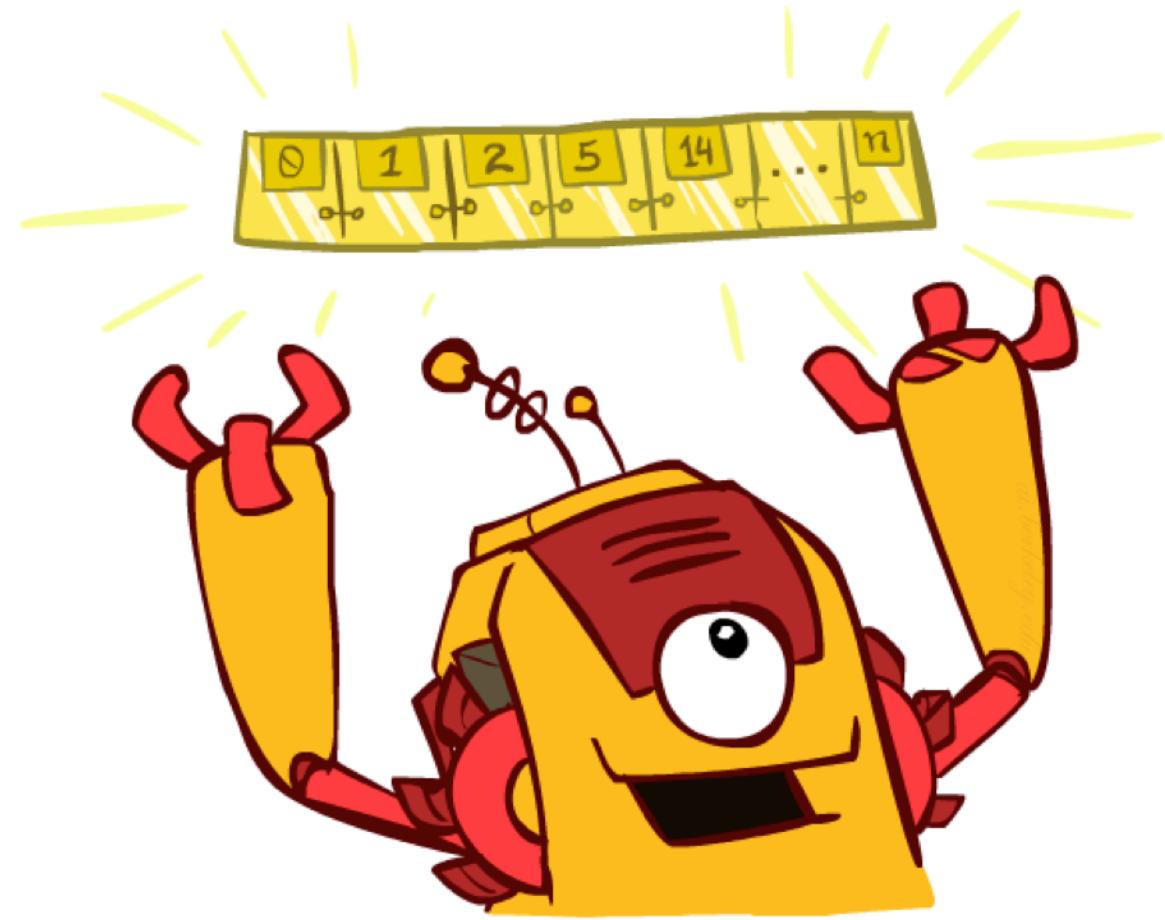
General Tree Search

```
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
```

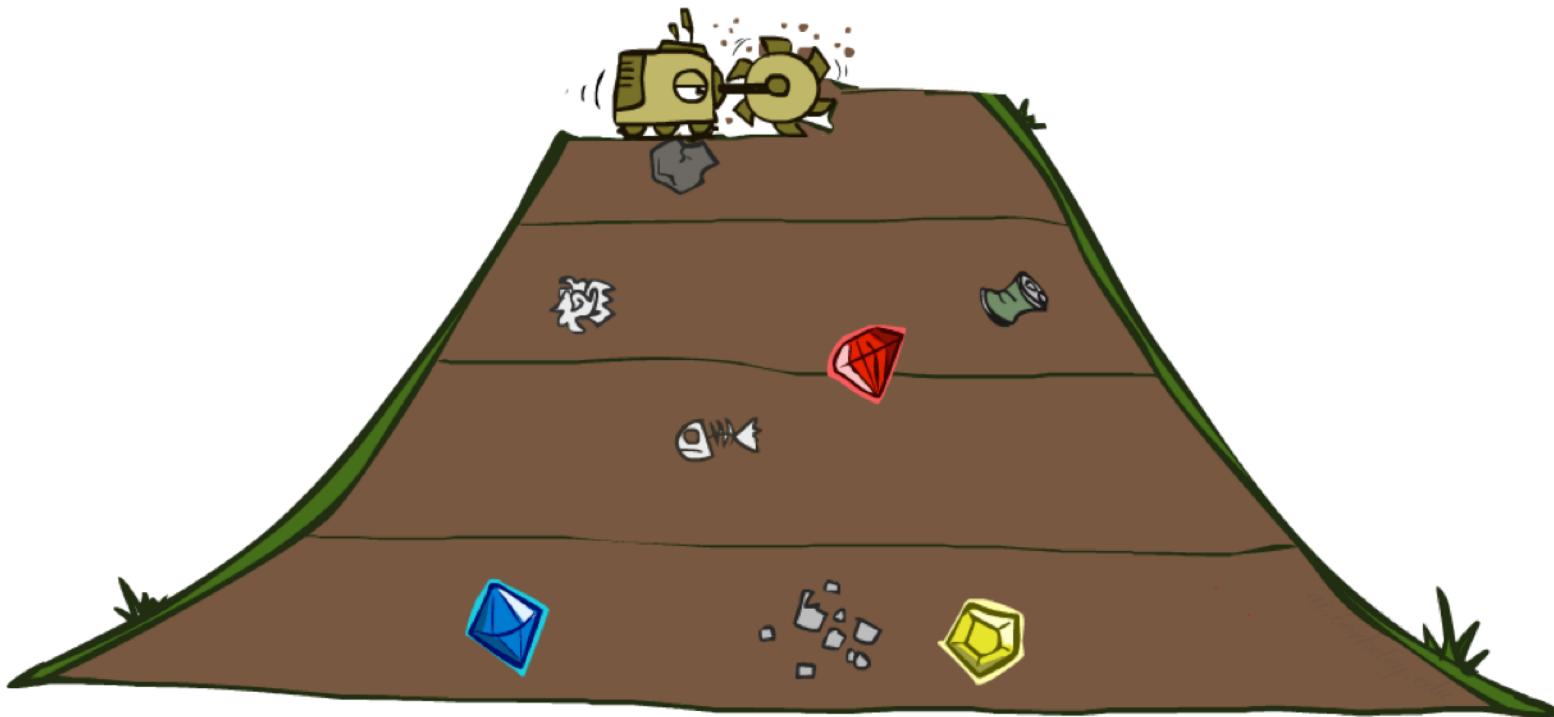


The One Queue

- 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



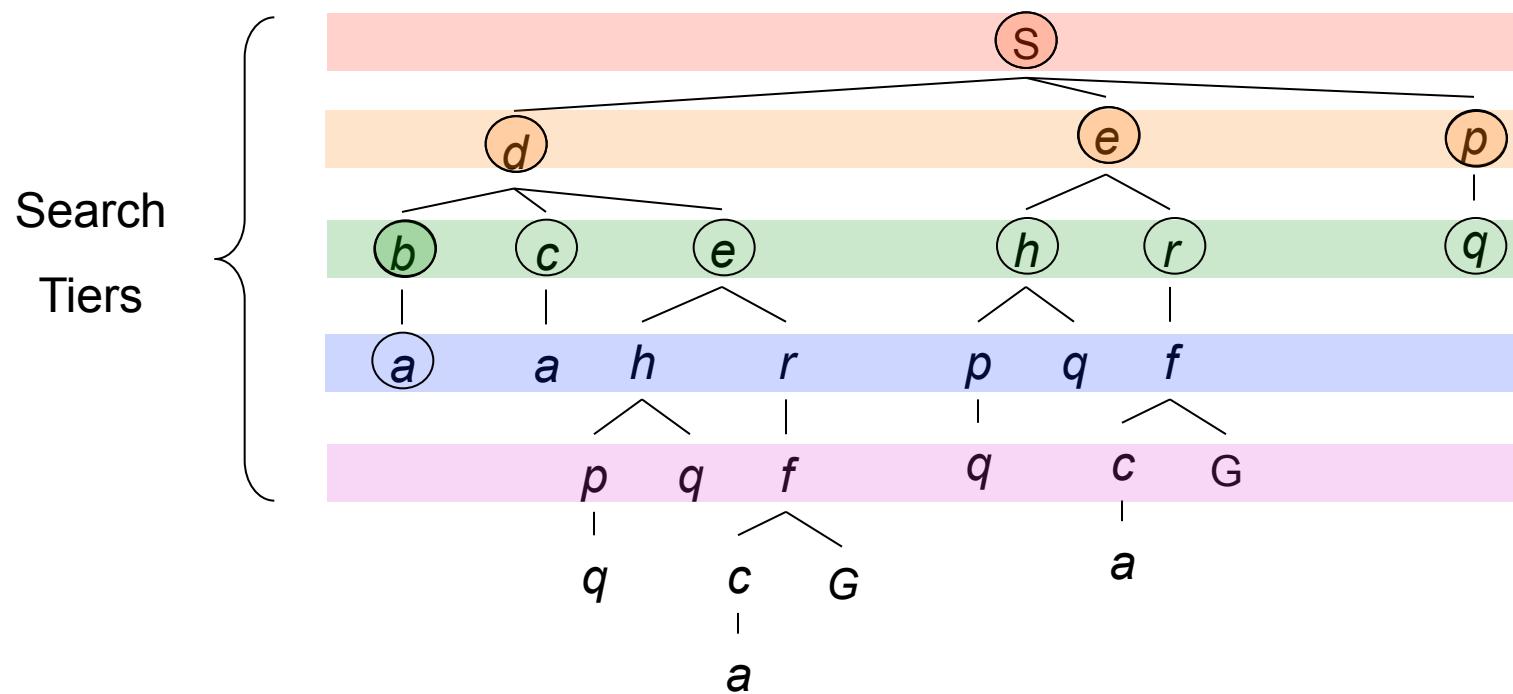
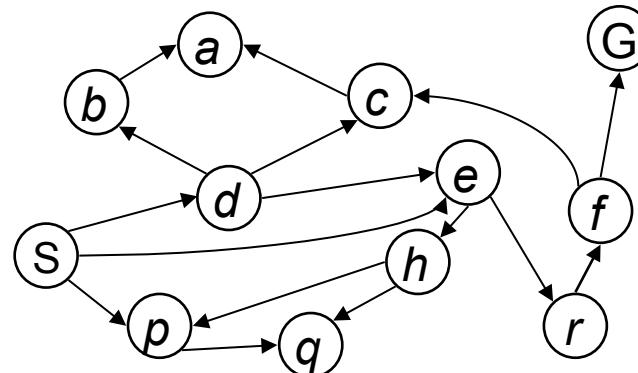
Breadth-First Search



Breadth-First Search

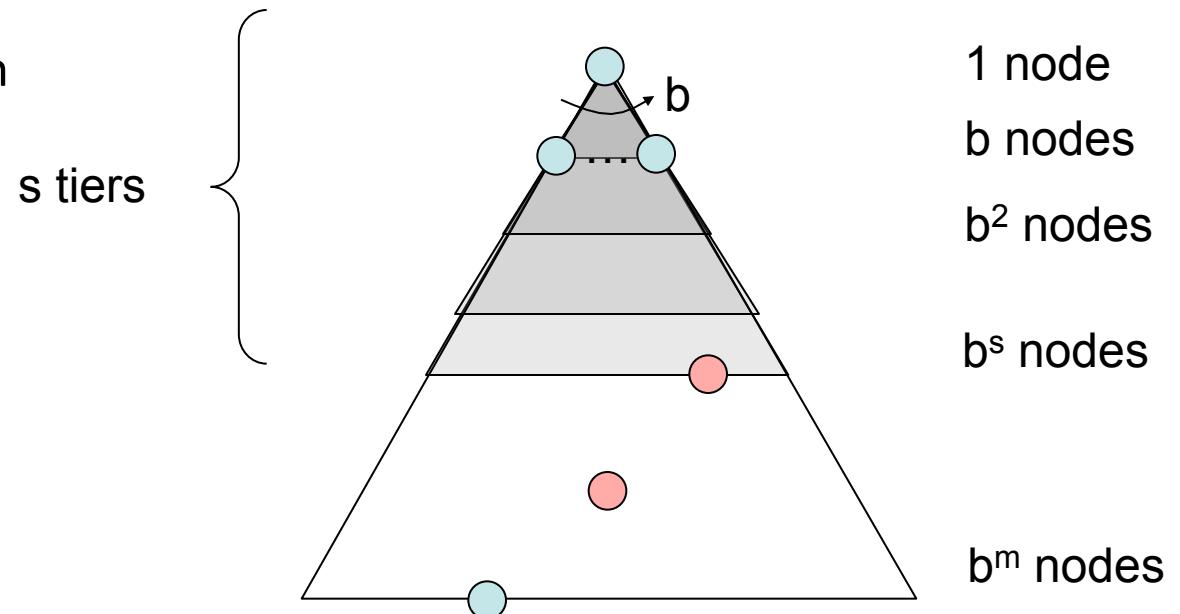
Strategy: expand a shallowest node first

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

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

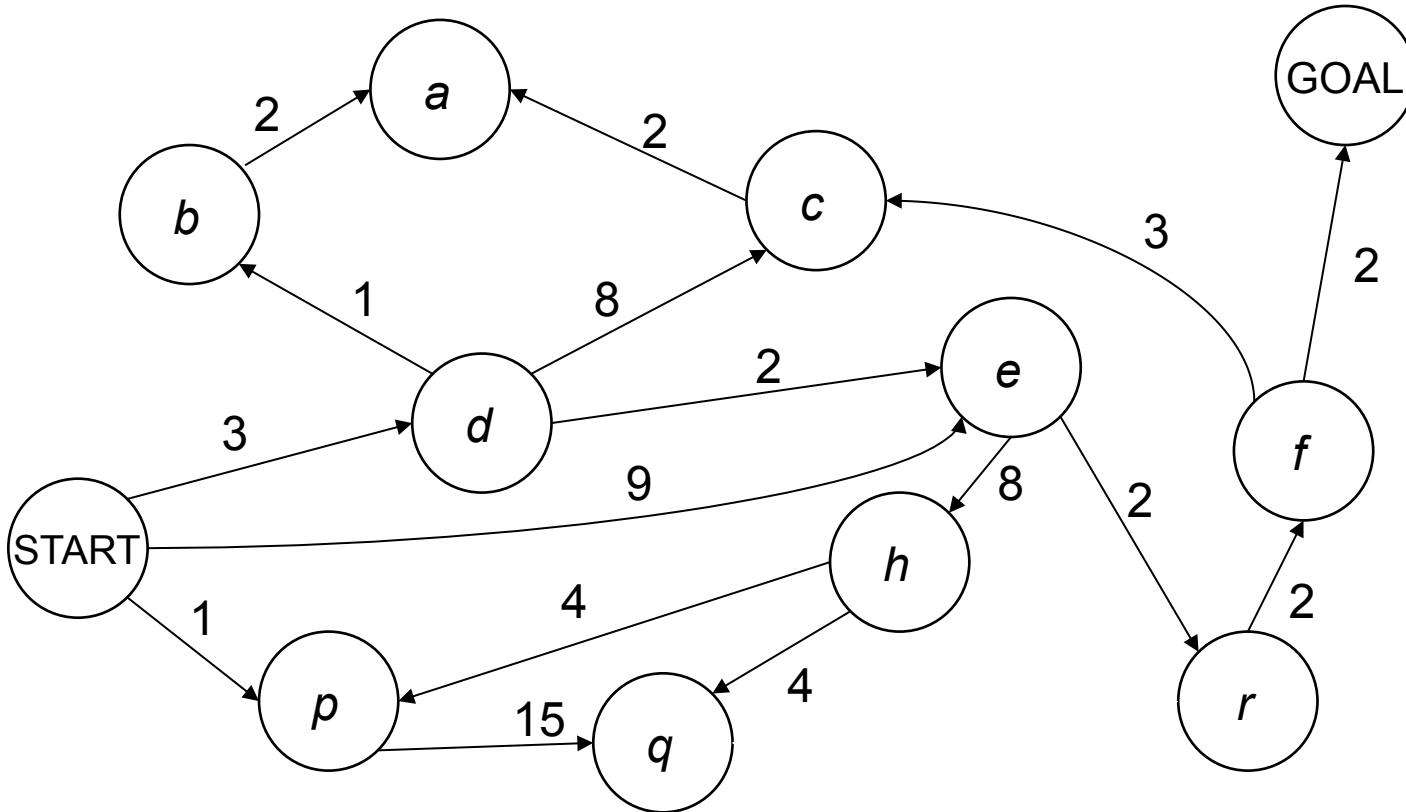
Video of Demo Maze Water DFS/BFS (part 1)



Video of Demo Maze Water DFS/BFS (part 2)



Cost-Sensitive Search

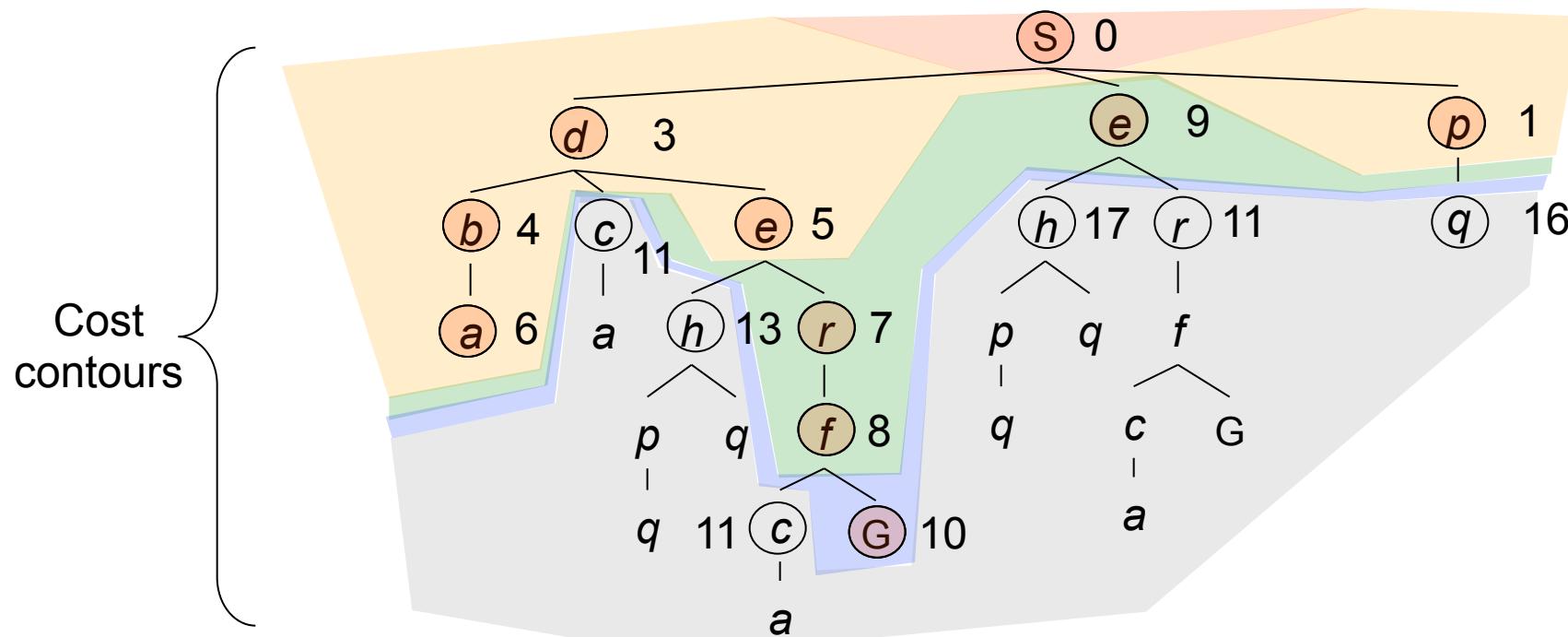
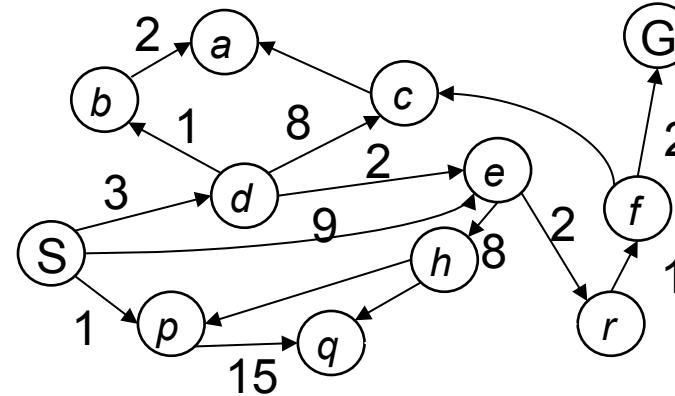


BFS finds the shortest path in terms of number of actions.
It does not find the least-cost path. We will now cover
a similar algorithm which does find the least-cost path.

Uniform Cost Search

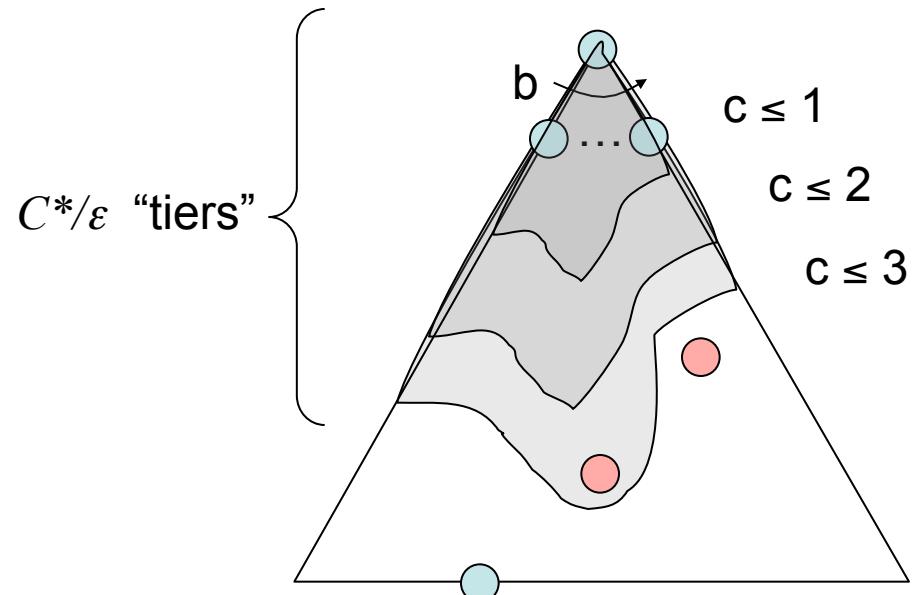
Strategy: expand a cheapest node first:

Fringe is a priority queue
(priority: cumulative cost)



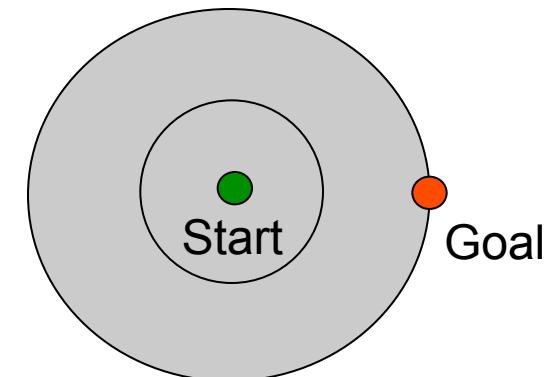
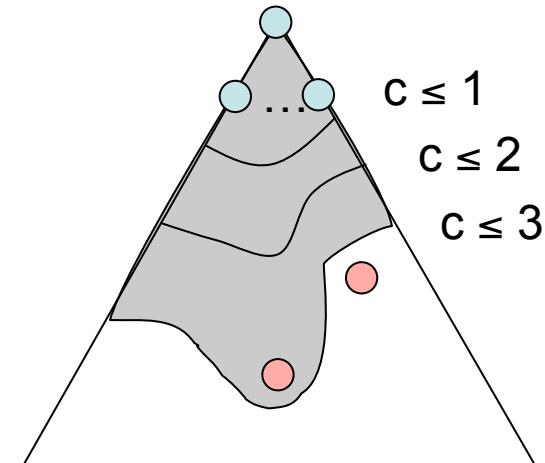
Uniform Cost Search (UCS) Properties

- 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 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!

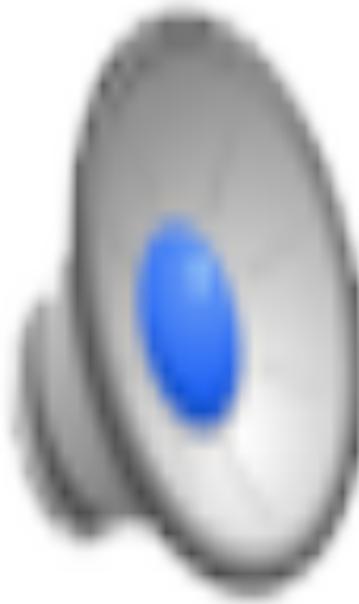


[Demo: empty grid UCS (L2D5)]
[Demo: maze with deep/shallow water DFS/BFS/UCS (L2D7)]

Video of Demo Contours UCS Empty



Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 2)

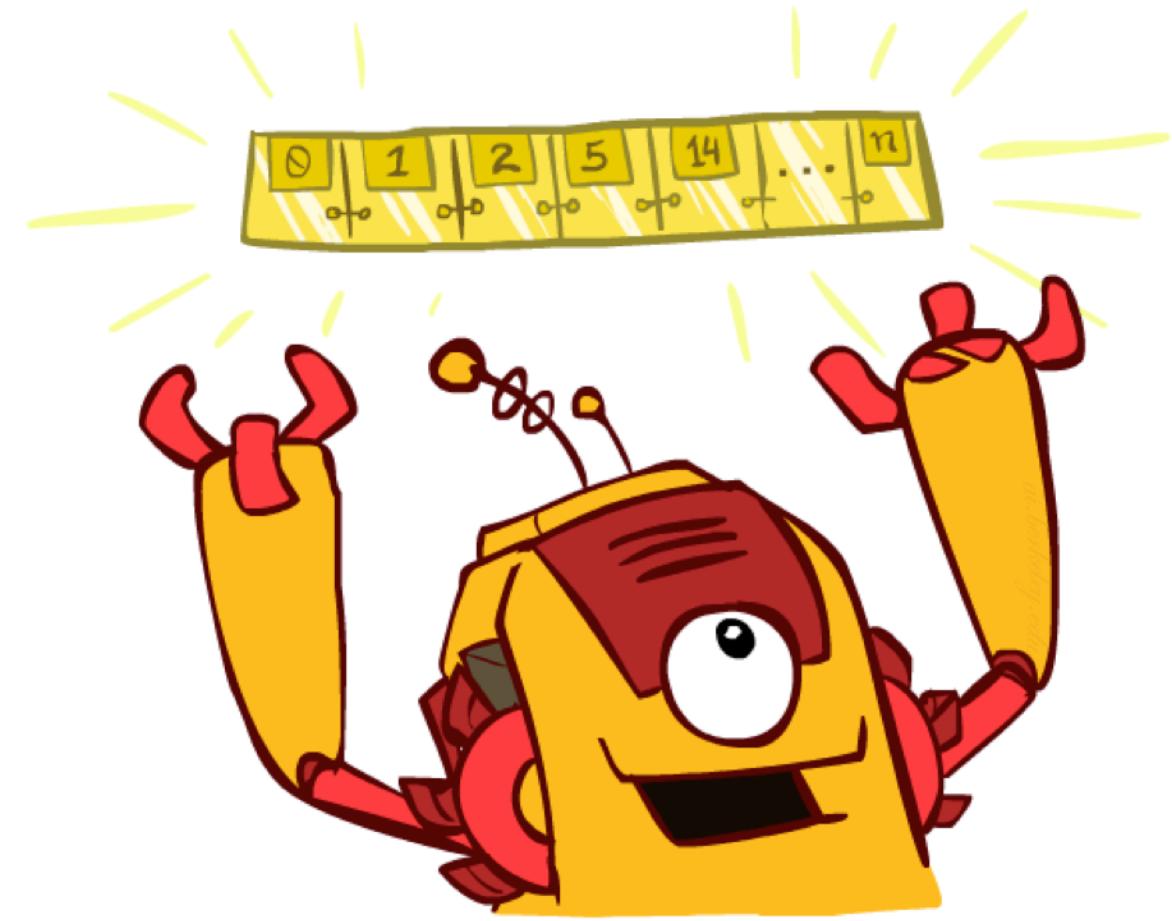


Video of Demo Contours UCS Pacman Small Maze



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. This will be the preferred approach on project 1.



Up next: Informed Search

- Uninformed Search

- DFS
- BFS
- UCS



- Informed Search

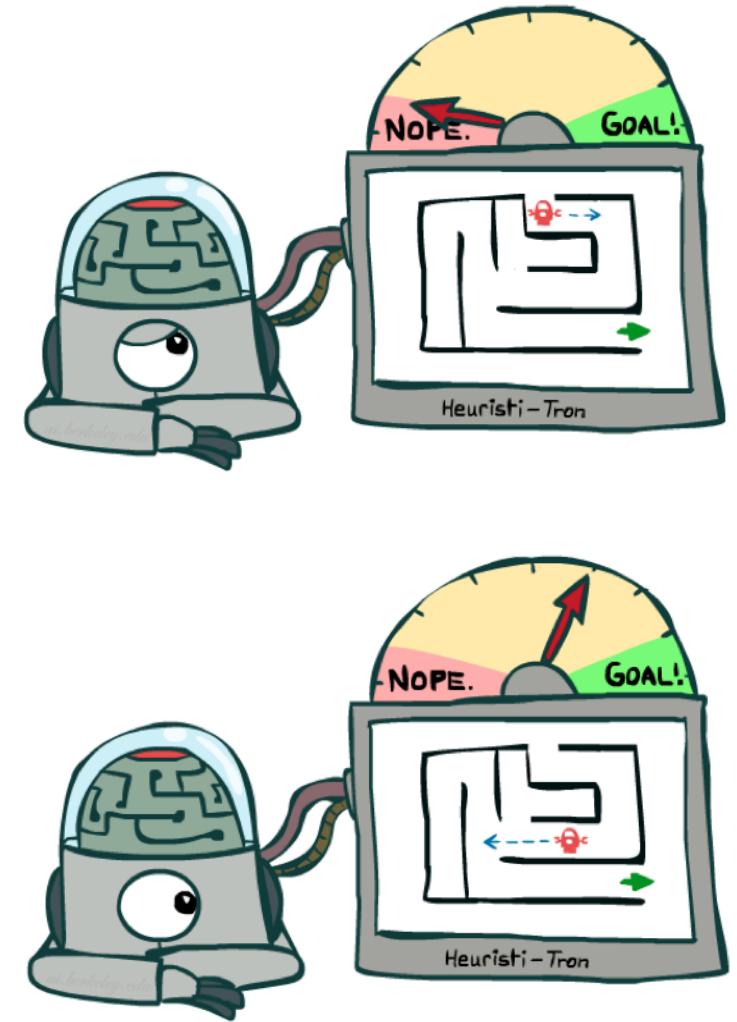
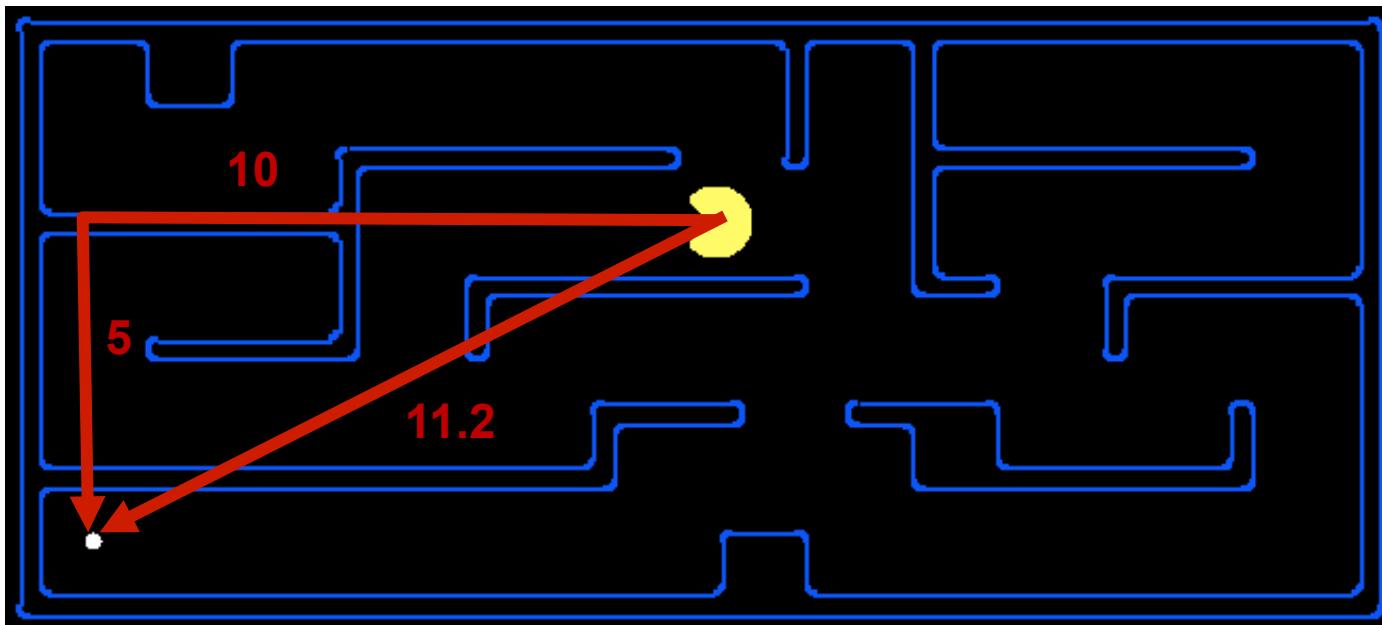
- Heuristics
- Greedy Search
- A* Search
- Graph Search



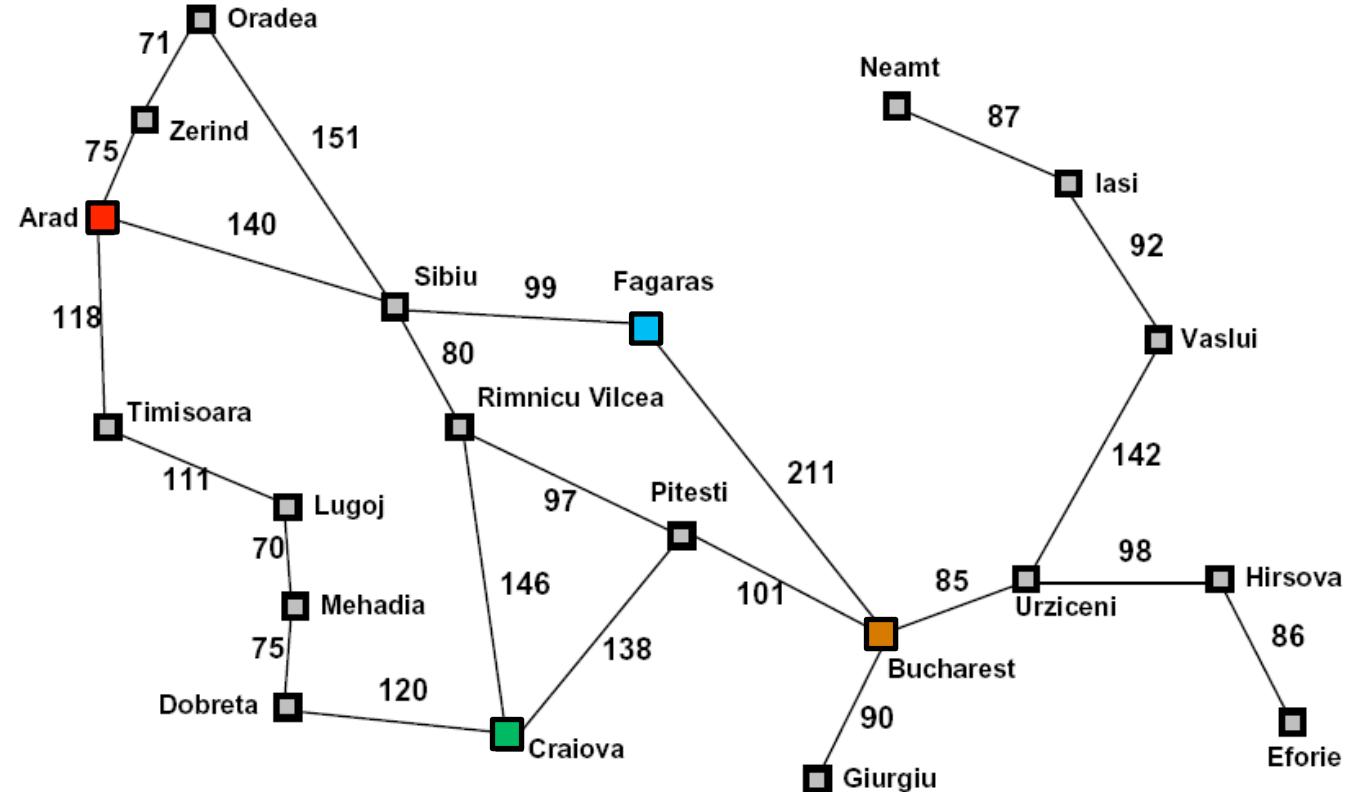
Search Heuristics

- A heuristic is:

- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
- Examples: Manhattan distance, Euclidean distance for pathing

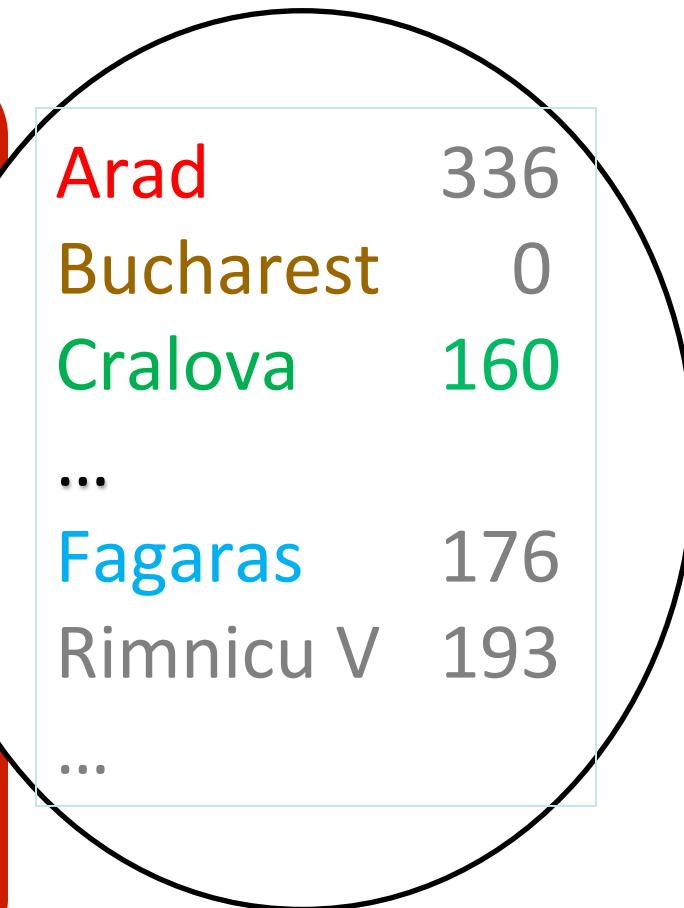


Example: Heuristic Function



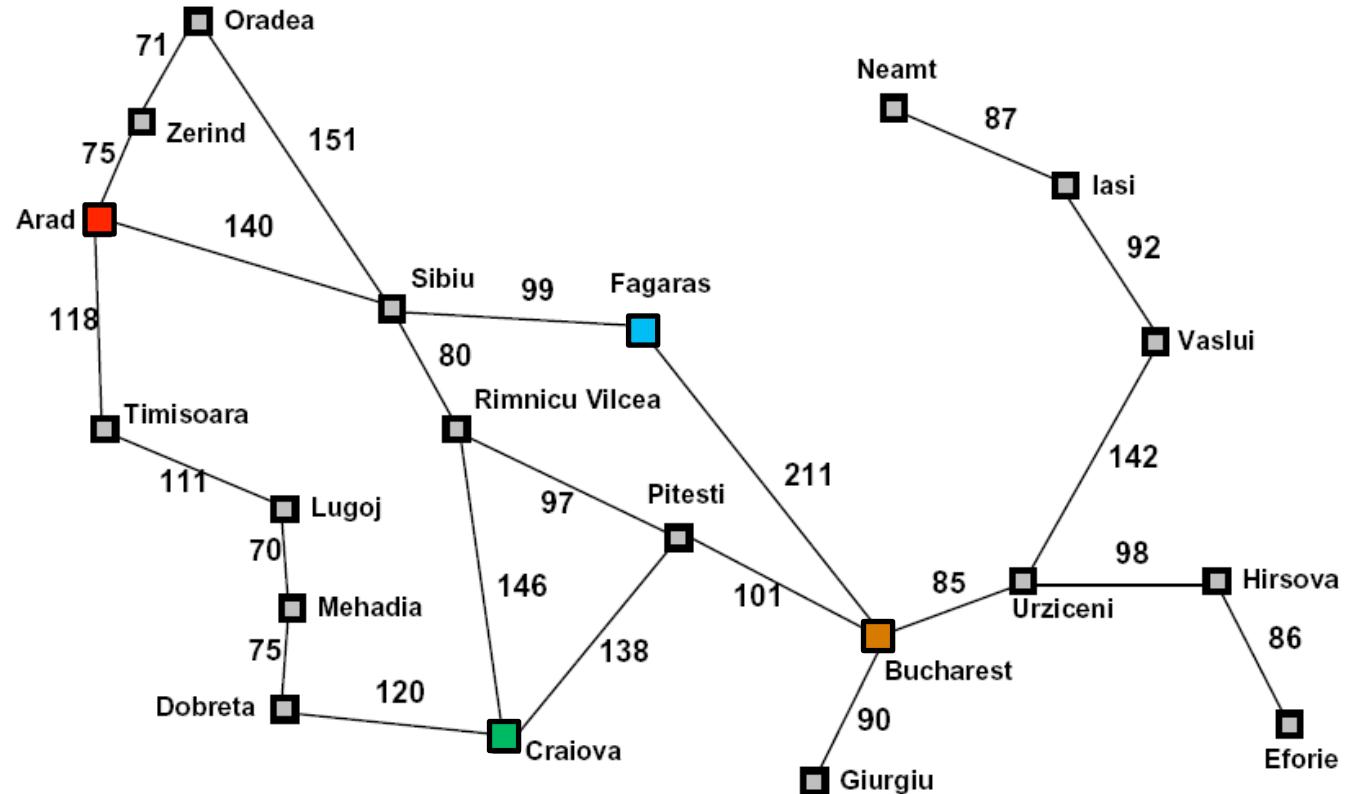
Straight-line distance
to Bucharest

Arad 366
Bucharest 0
Craiova 160
Dobrete 24
Eforie 1
Fagaras 176
Giurgiu 38
Hirsova 98
Iasi 193
Lugoj 253
Mehadia 329
Neamt 80
Oradea 199
Pitesti 374
Rimnicu Vilcea 193
Sibiu 223
Timisoara 23
Urziceni 86
Vaslui 199
Zerind 374



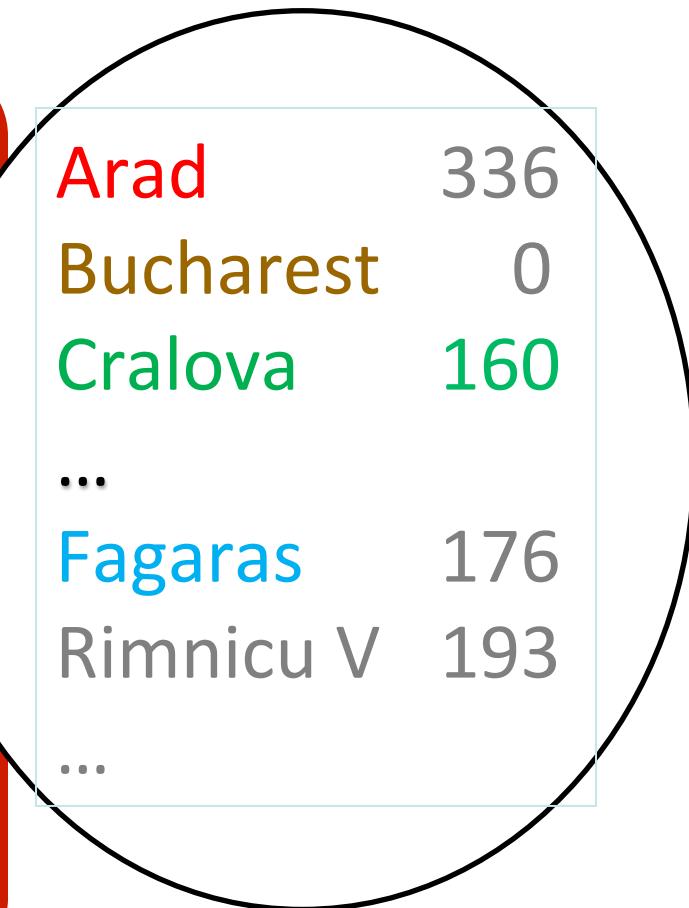
$h(x)$

Greedy Search



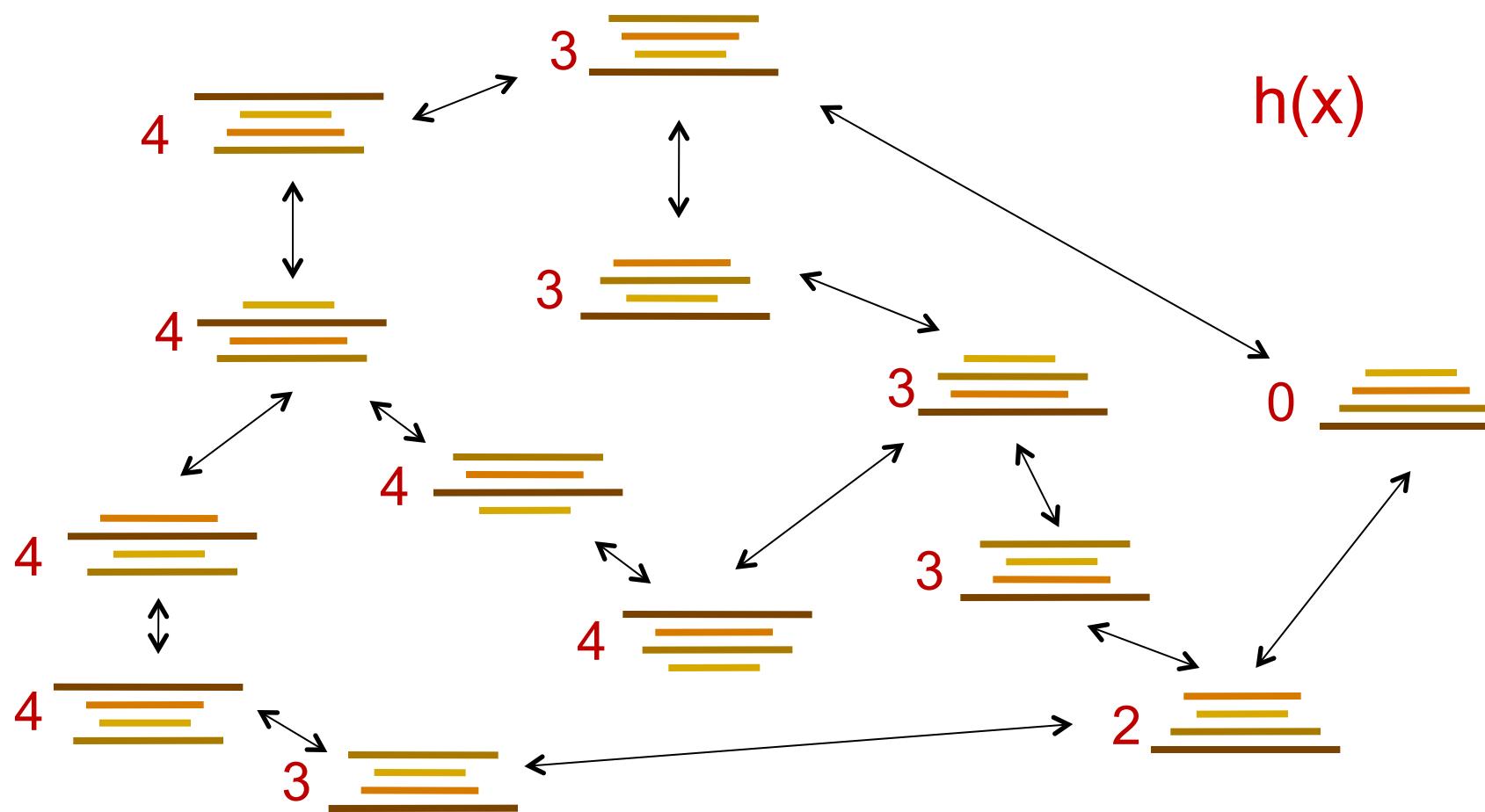
Straight-line distance to Bucharest
Arad
Bucharest
Craiova
Dobreta
Eforie
Fagaras
Giurgiu
Hirsova
Iasi
Lugoj
Mehadia
Neamt
Oradea
Pitesti
Rimnicu Vilcea
Sibiu
Timisoara
Urziceni
Vaslui
Zerind

$h(x)$



Example: Heuristic Function

Heuristic: the number of the largest pancake that is still out of place

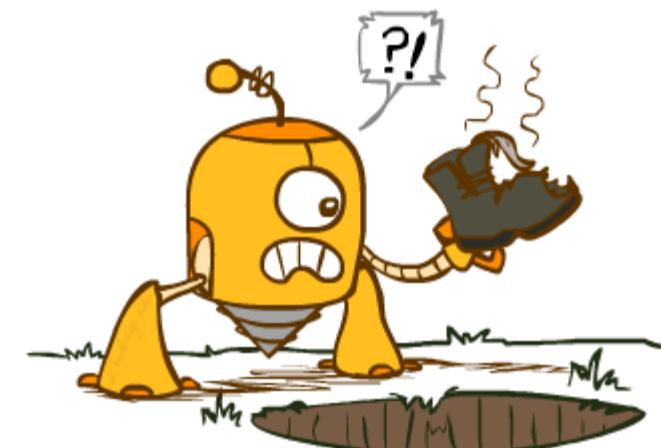
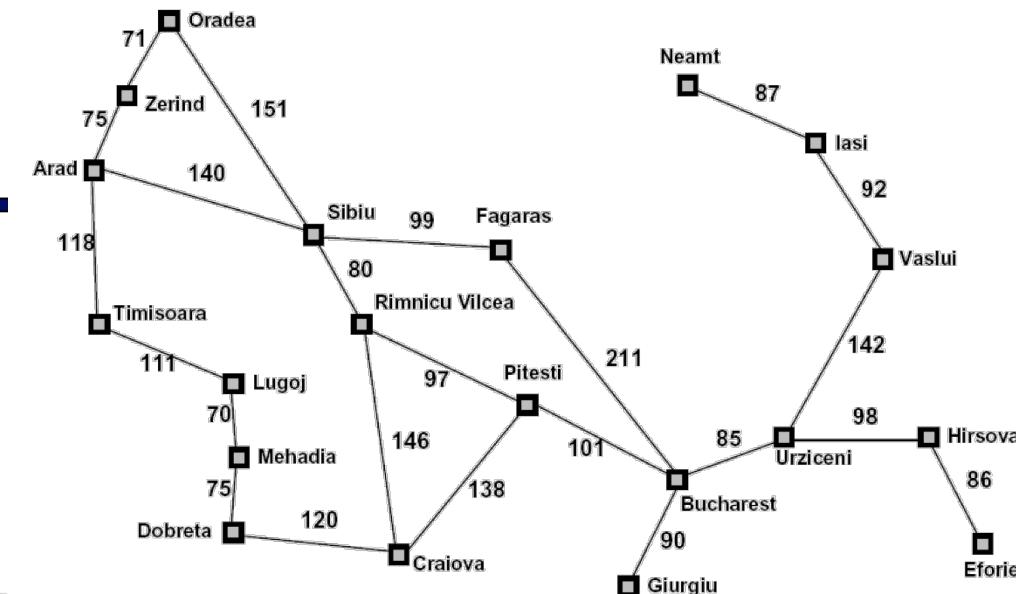
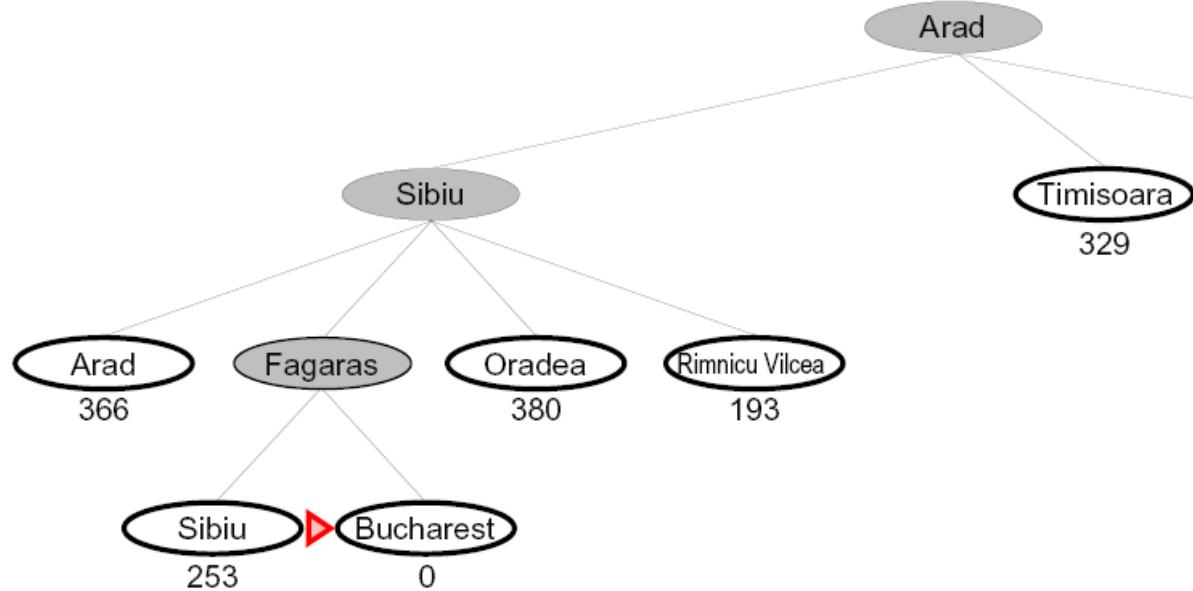


Greedy Search



Greedy Search

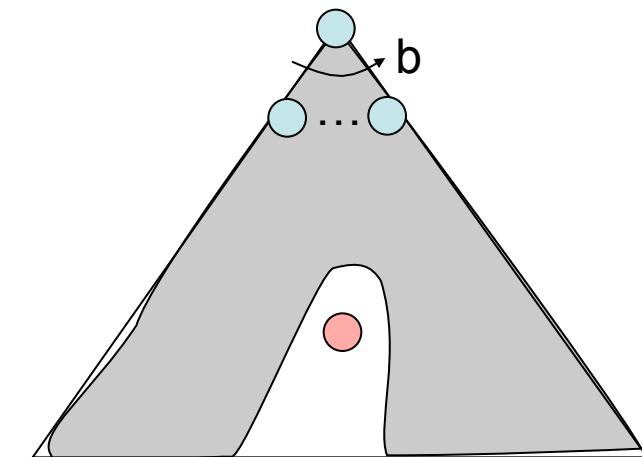
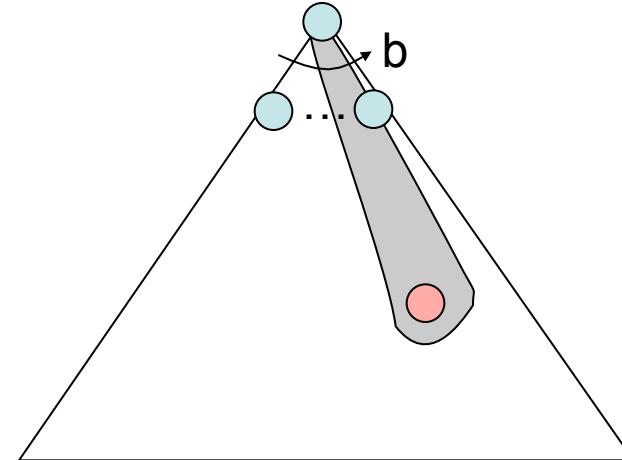
- Expand the node that seems closest...



- What can go wrong?
 - Resulting path to Bucharest is not the shortest!

Greedy Search

- Strategy: expand a node that you think is closest to a goal state
 - Heuristic: estimate of distance to nearest goal for each state
- A common case:
 - Best-first takes you straight to the (wrong) goal
- Worst-case: like a badly-guided DFS



[Demo: contours greedy empty (L3D1)]

[Demo: contours greedy pacman small maze (L3D4)]

Video of Demo Contours Greedy (Empty)



Video of Demo Contours Greedy (Pacman Small Maze)



A* Search

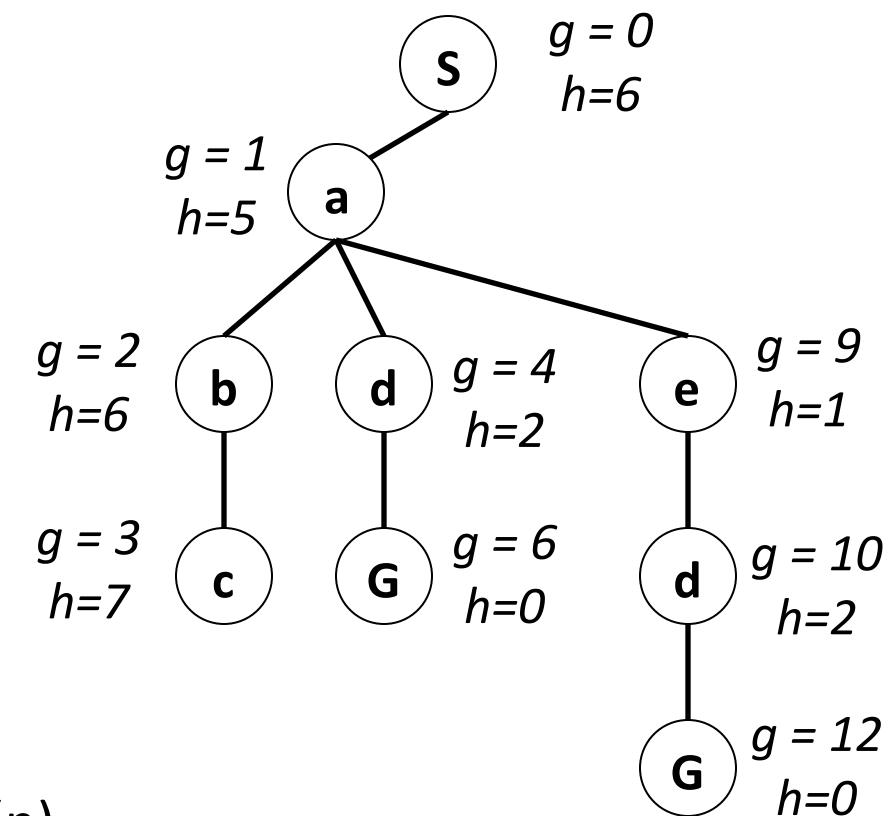
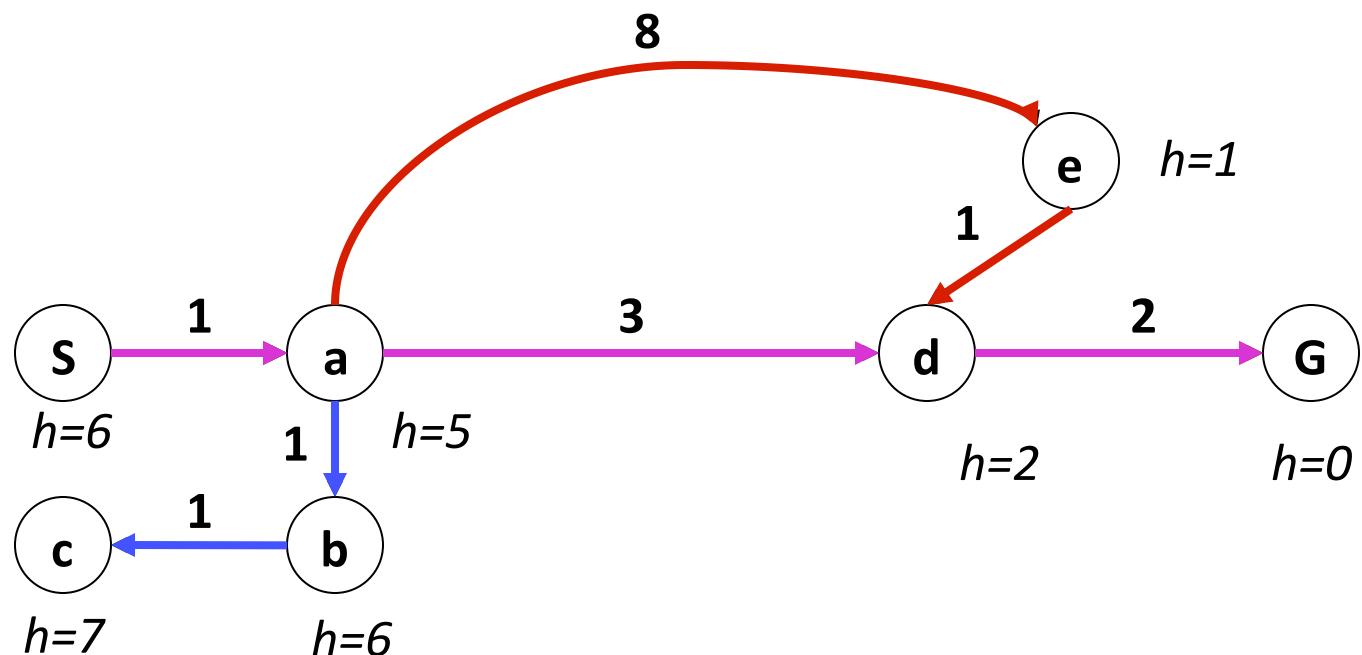


A* Search

1

Combining UCS and Greedy

- Uniform-cost orders by path cost, or *backward cost* $g(n)$
- Greedy orders by goal proximity, or *forward cost* $h(n)$

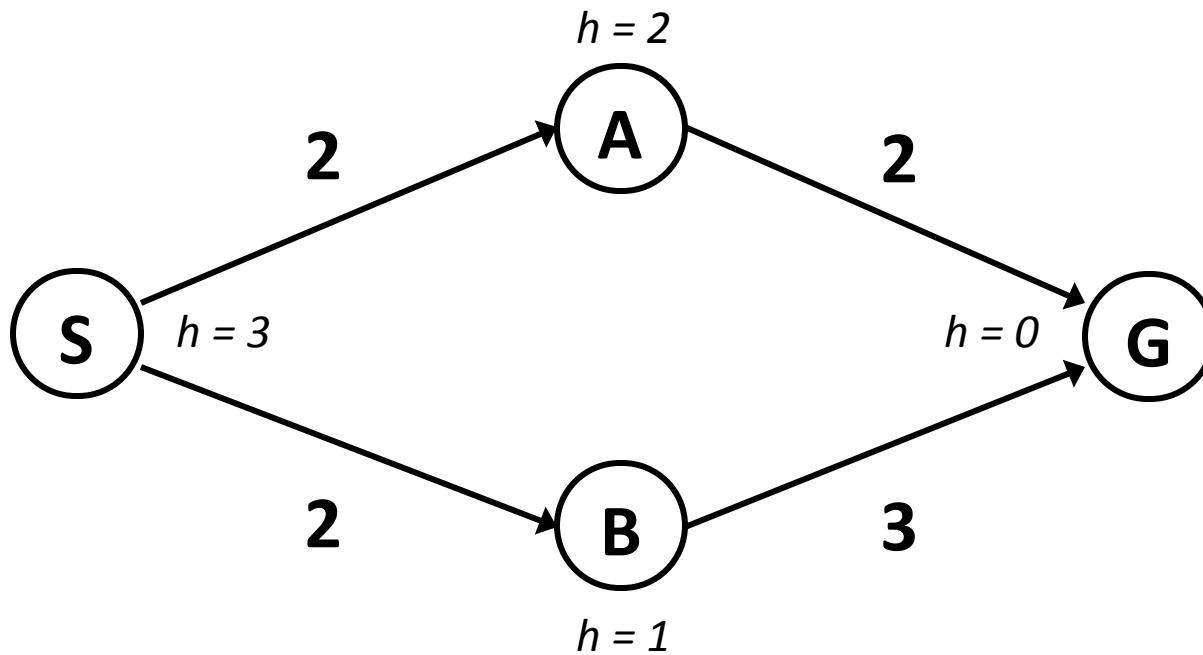


- A* Search orders by the sum: $f(n) = g(n) + h(n)$

Example: Teg Grenager

When should A* terminate?

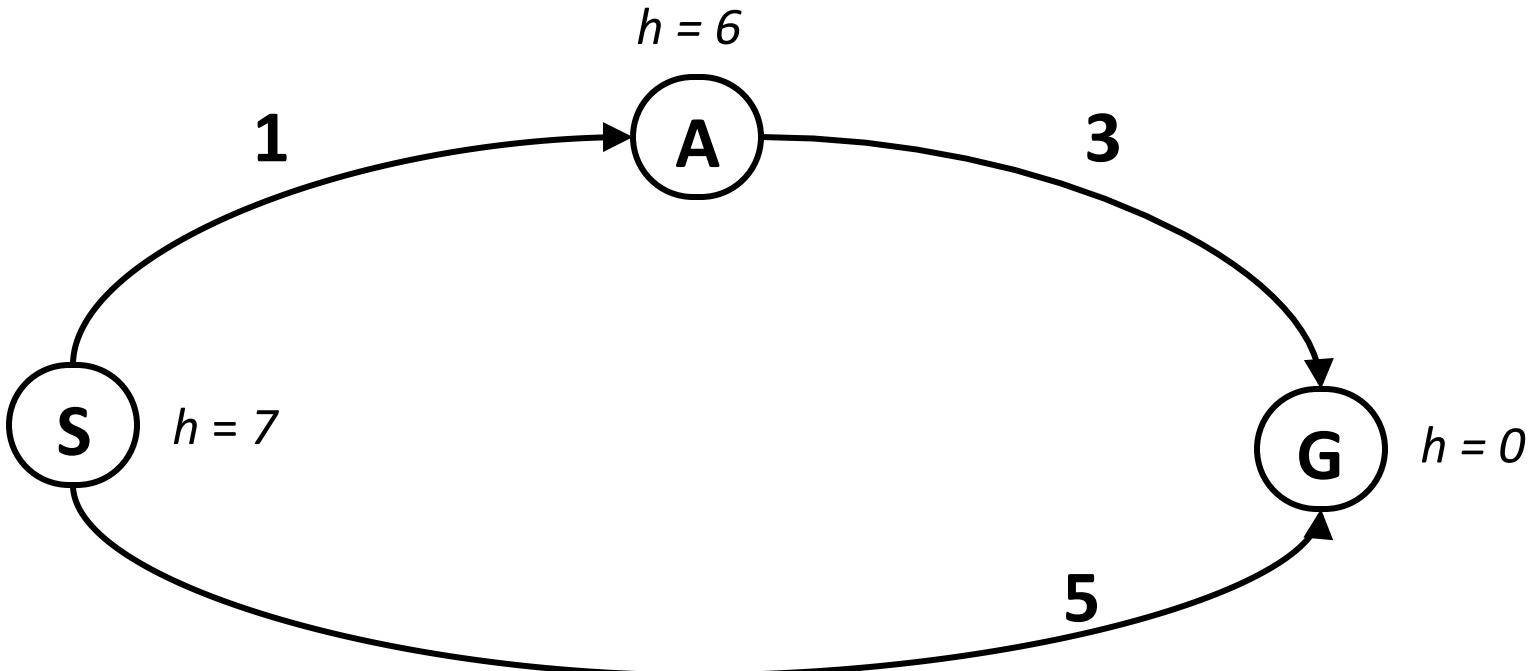
- Should we stop when we enqueue a goal?



	g	h	$+$
S	0	3	3
S->A	2	2	4
S->B	2	1	3
S->B->G	5	0	5
S->A->G	4	0	4

- No: only stop when we dequeue a goal

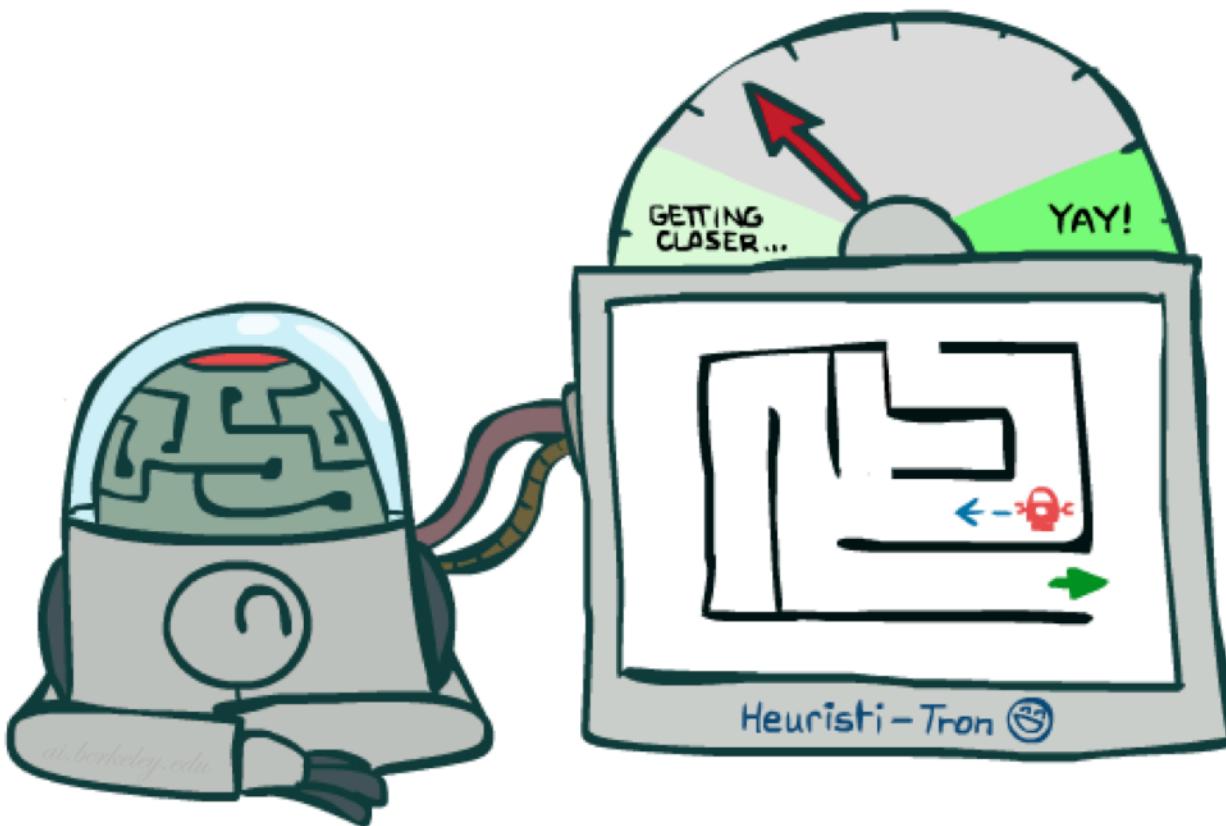
Is A* Optimal?



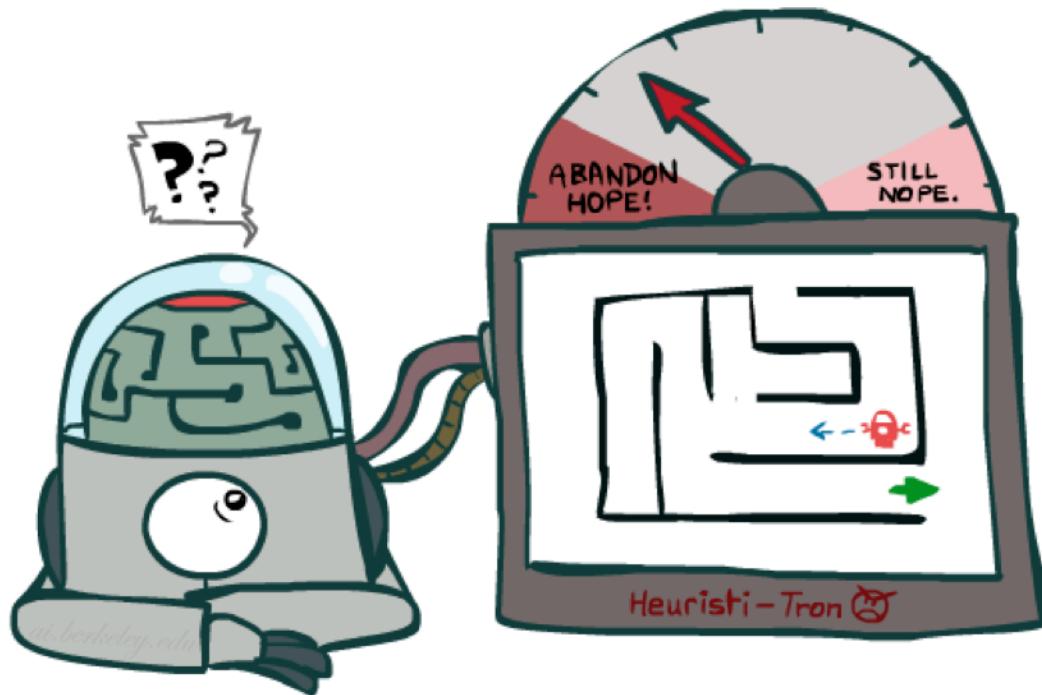
	g	h	$+$
S	0	7	7
S->A	1	6	7
S->G	5	0	5

- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!

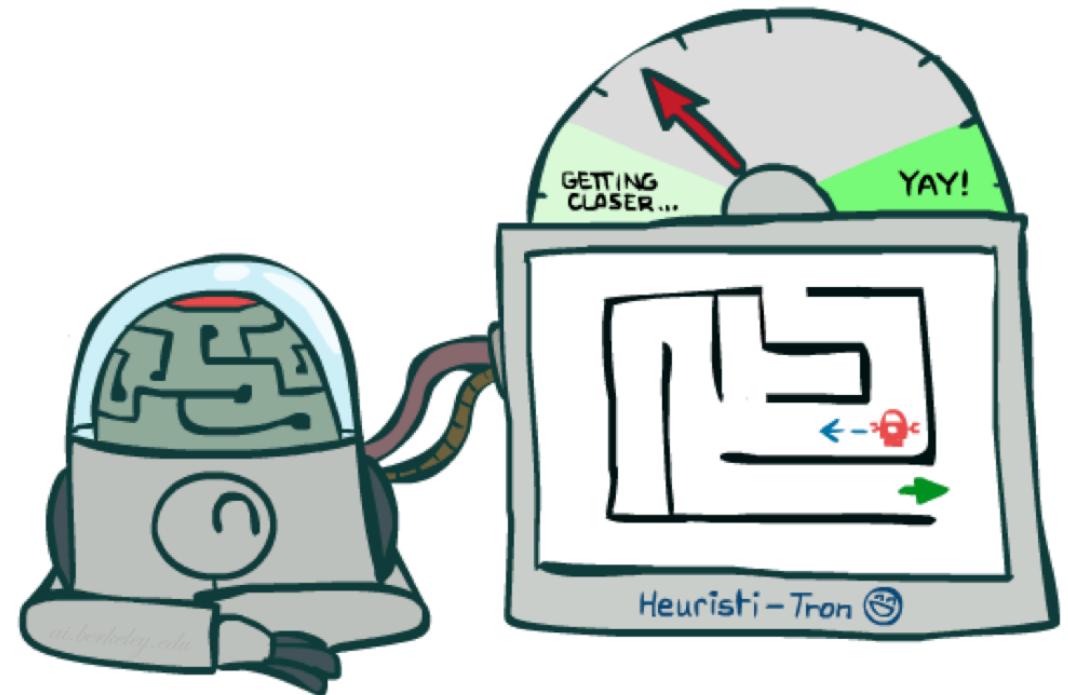
Admissible Heuristics



Idea: Admissibility



Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the fringe



Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs

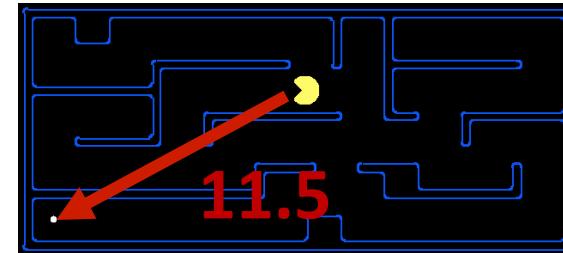
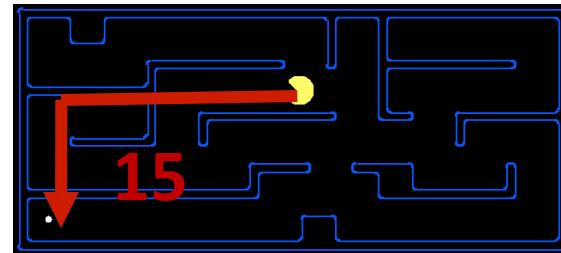
Admissible Heuristics

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

$$0 \leq h(n) \leq h^*(n)$$

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

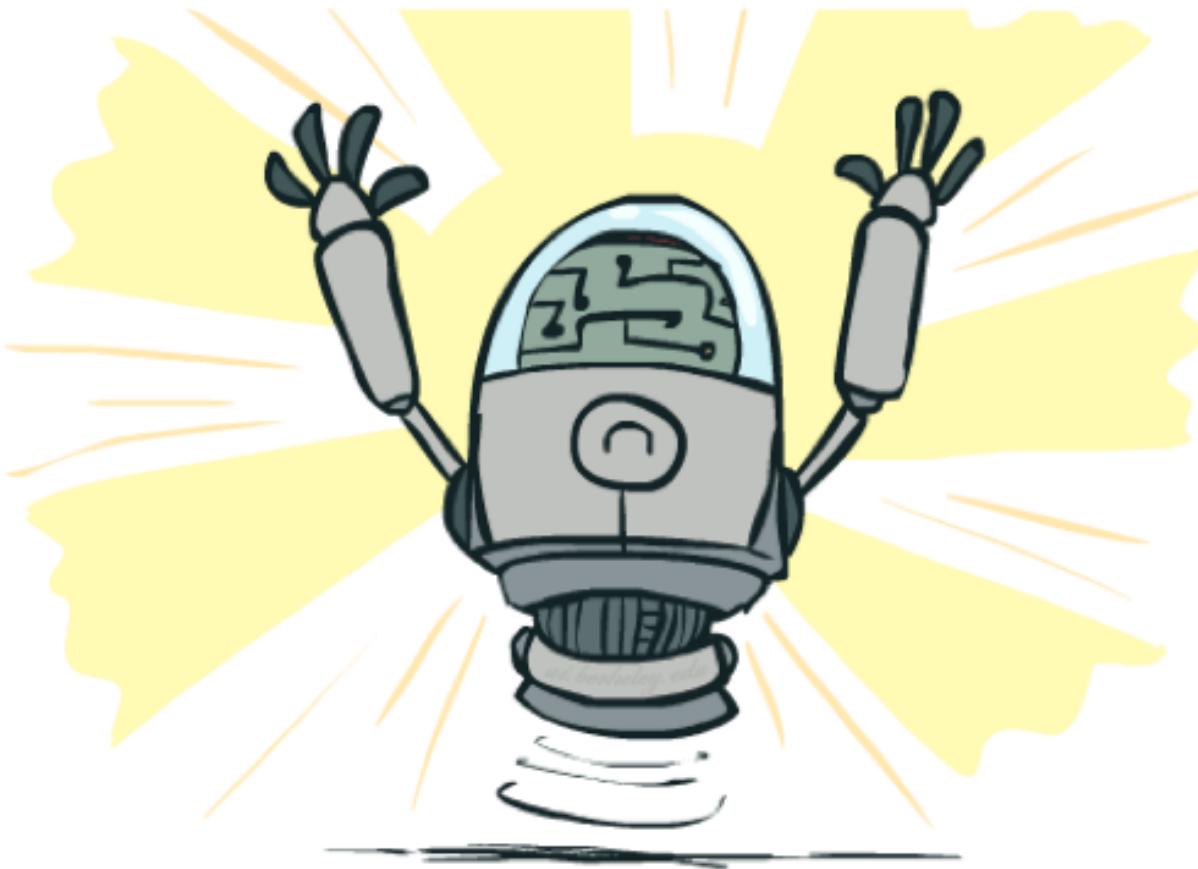
- Examples:



0.0

- Coming up with admissible heuristics is most of what's involved in using A* in practice.

Optimality of A* Tree Search



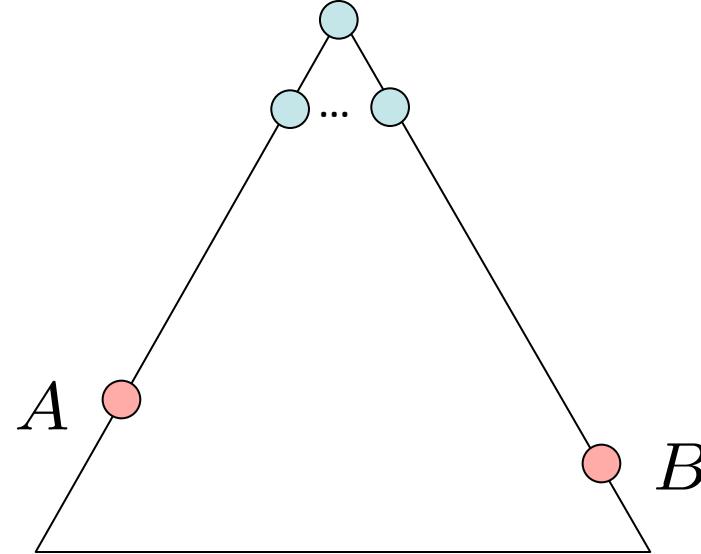
Optimality of A* Tree Search

Assume:

- A is an optimal goal node
- B is a suboptimal goal node
- h is admissible

Claim:

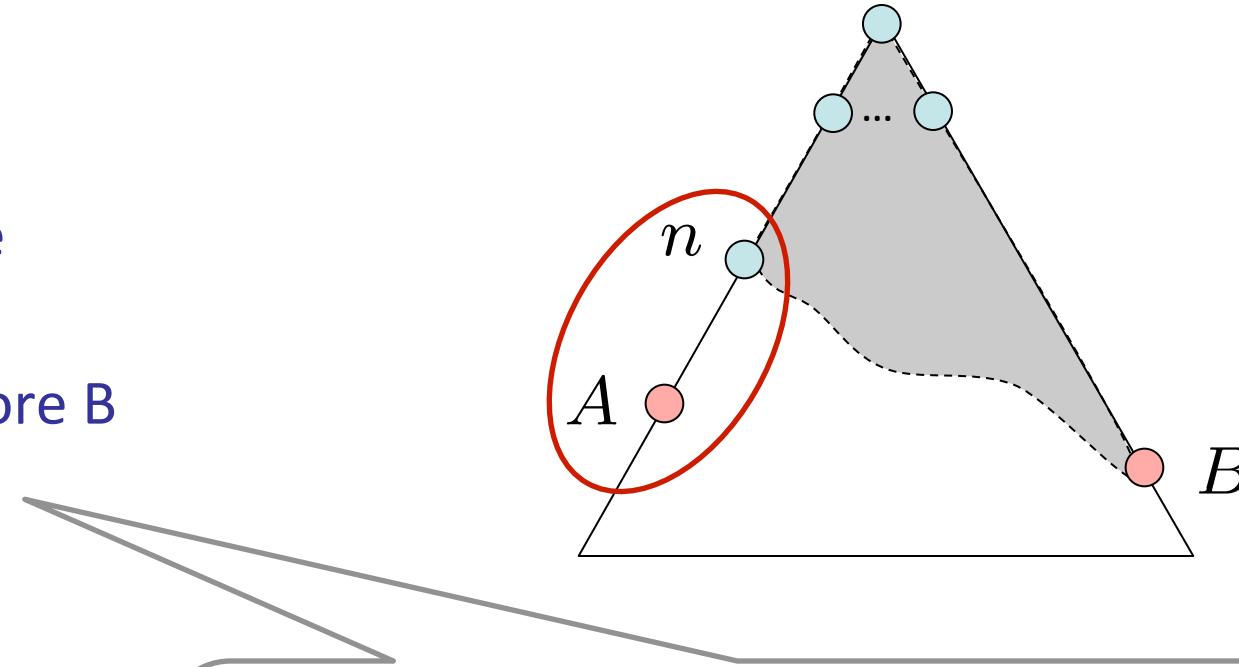
- A will exit the fringe before B



Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will be expanded before B
 - 1. $f(n)$ is less or equal to $f(A)$



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

$$f(n) \leq g(A)$$

$$g(A) = f(A)$$

Definition of f-cost

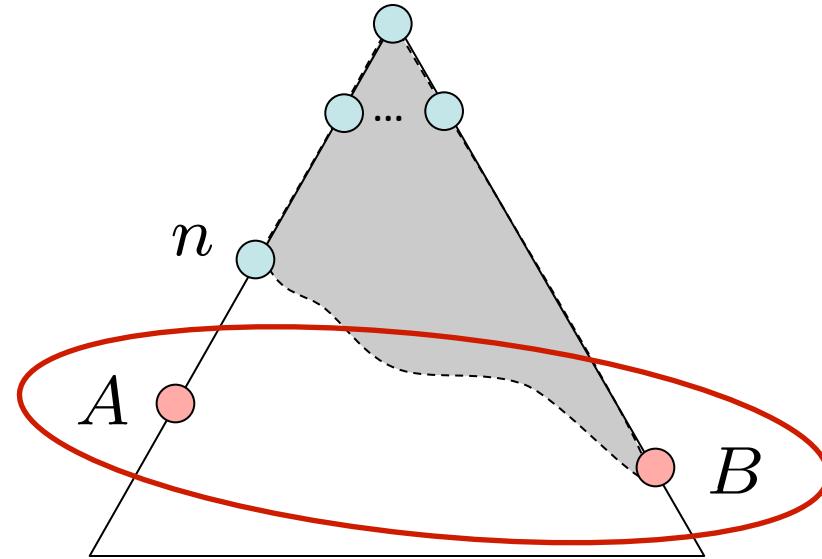
Admissibility of h

$h = 0$ at a goal

Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will be expanded before B
 1. $f(n)$ is less or equal to $f(A)$
 2. $f(A)$ is less than $f(B)$



$$g(A) < g(B)$$

$$f(A) < f(B)$$

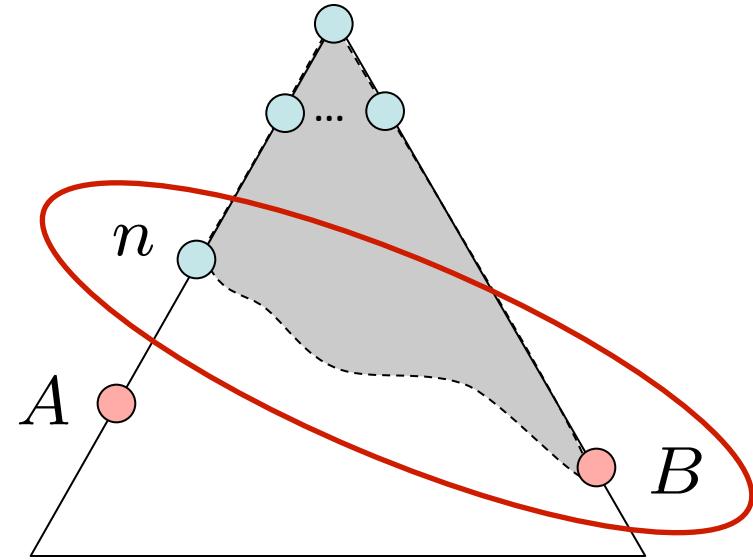
B is suboptimal

$h = 0$ at a goal

Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will be expanded before B
 1. $f(n)$ is less or equal to $f(A)$
 2. $f(A)$ is less than $f(B)$
 3. n expands before B
- All ancestors of A expand before B
- A expands before B
- A* search is optimal

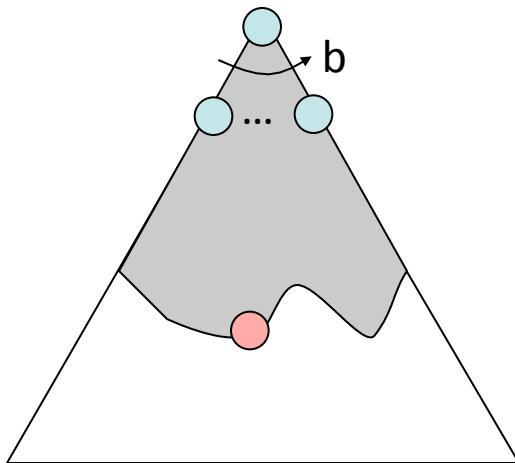


$$f(n) \leq f(A) < f(B)$$

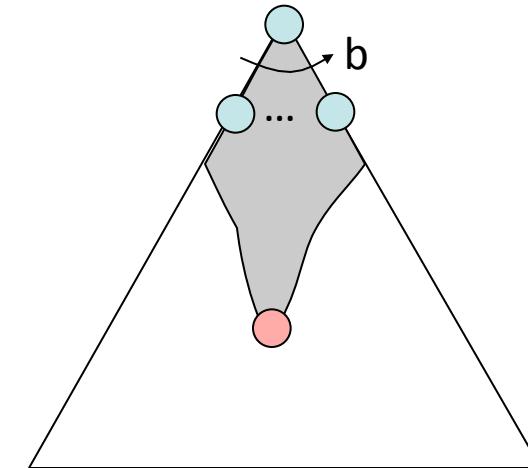
Properties of A*

Properties of A*

Uniform-Cost

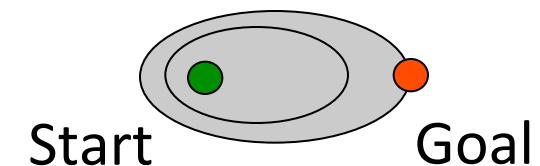
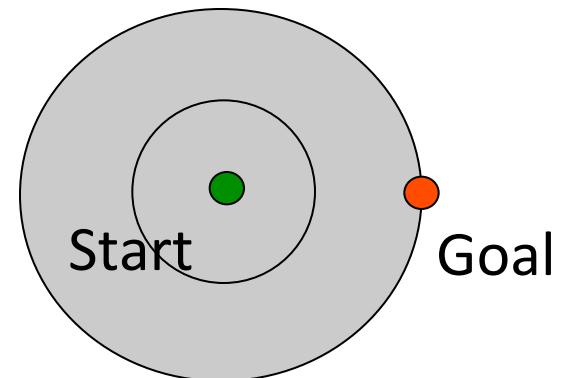


A*



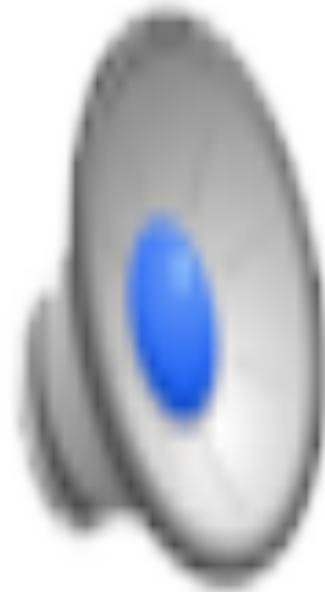
UCS vs A* Contours

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



[Demo: contours UCS / greedy / A* empty (L3D1)]
[Demo: contours A* pacman small maze (L3D5)]

Video of Demo Contours (Empty) -- UCS



Video of Demo Contours (Empty) -- Greedy



Video of Demo Contours (Empty) – A*



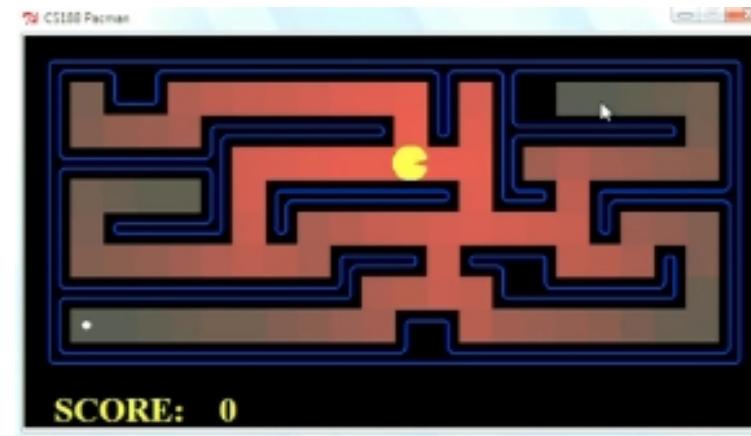
Video of Demo Contours (Pacman Small Maze) – A*



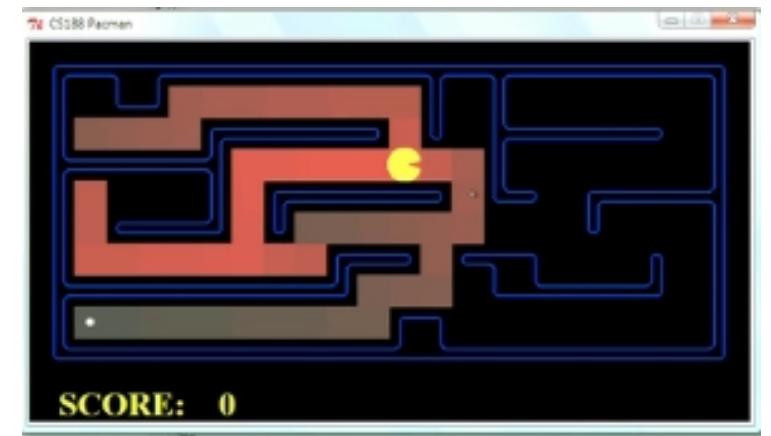
Comparison



Greedy



Uniform Cost



A*

A* Applications



A* Applications

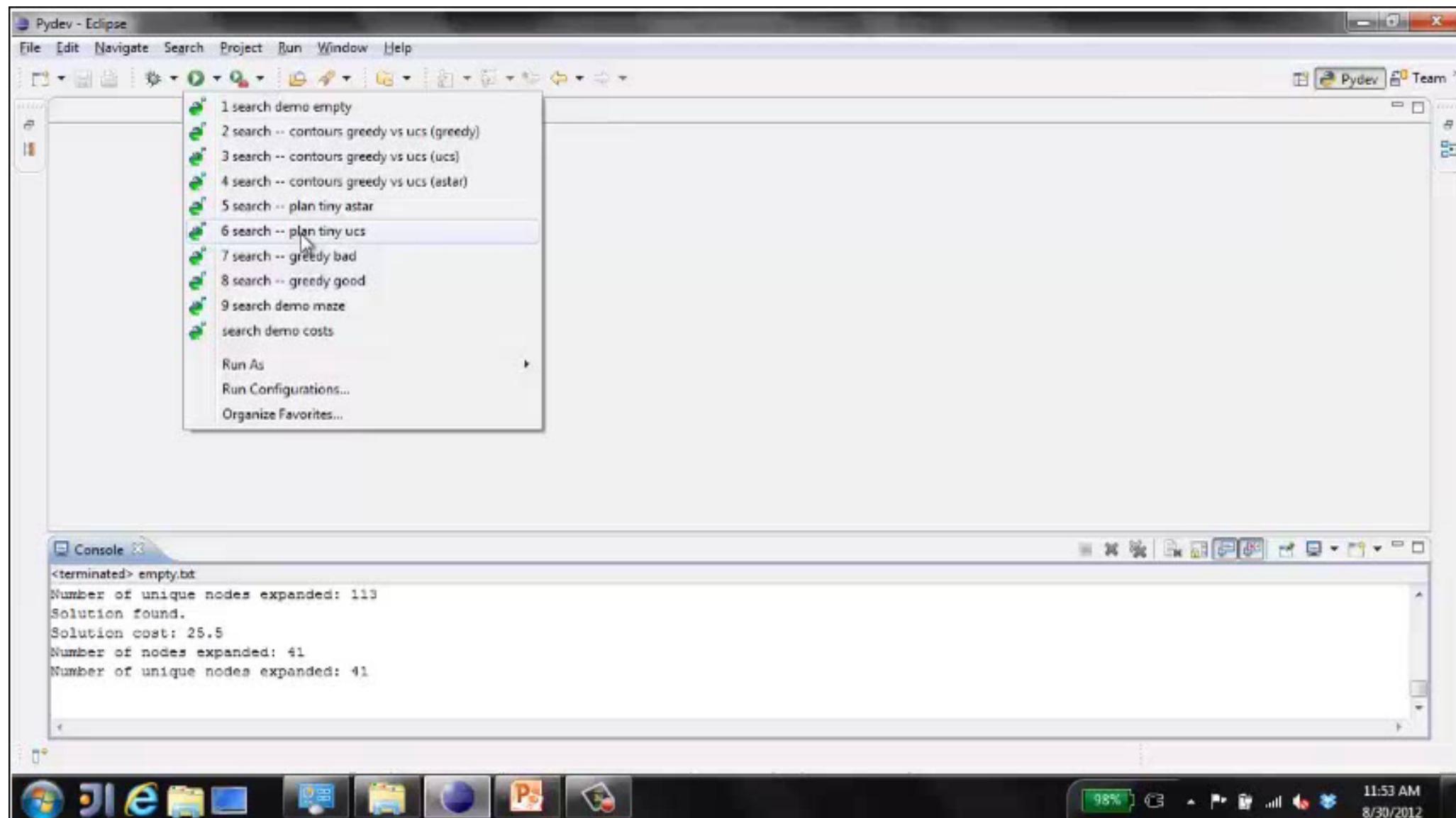
- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- ...



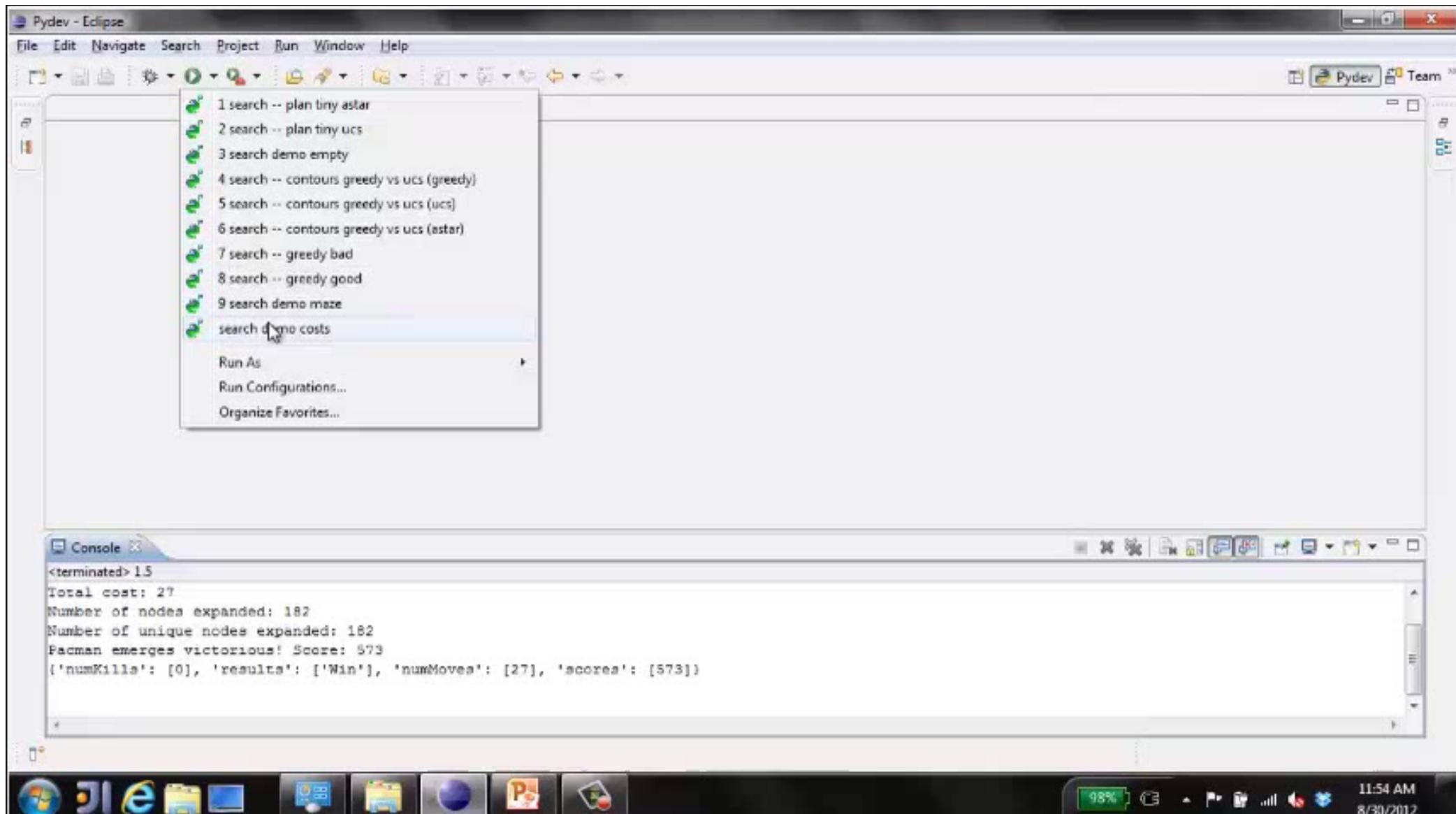
[Demo: UCS / A* pacman tiny maze (L3D6,L3D7)]

[Demo: guess algorithm Empty Shallow/Deep (L3D8)]

Video of Demo Pacman (Tiny Maze) – UCS / A*



Video of Demo Empty Water Shallow/Deep – Guess Algorithm

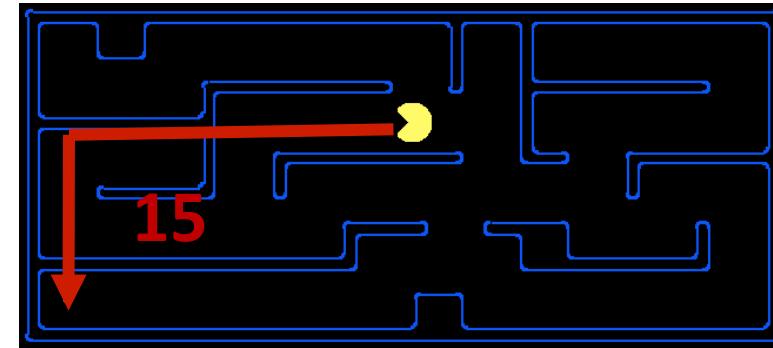
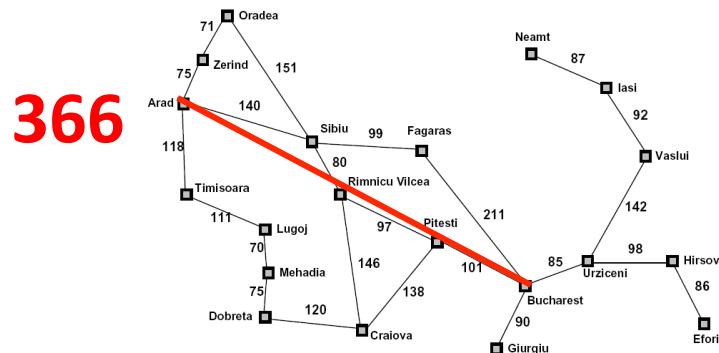


Creating Heuristics



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

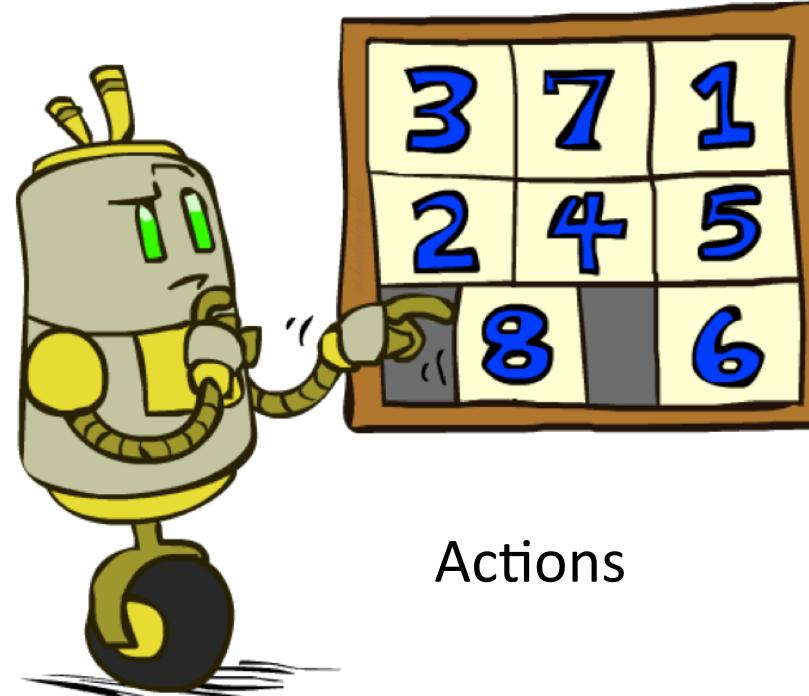


- Inadmissible heuristics are often useful too

Example: 8 Puzzle

7	2	4
5		6
8	3	1

Start State



Actions

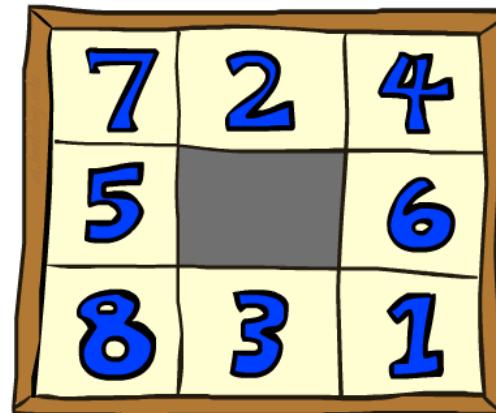
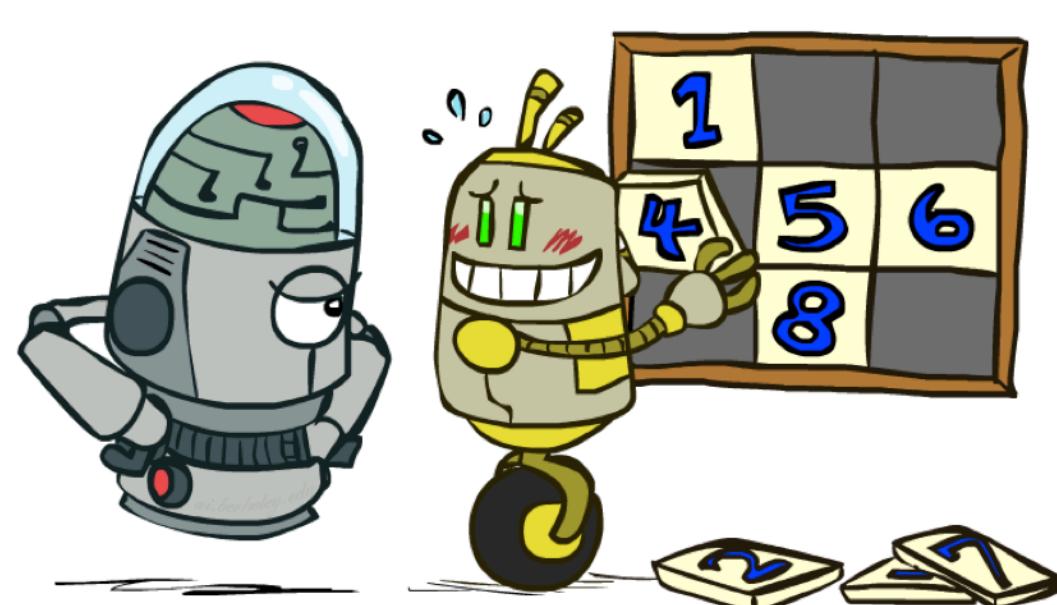
	1	2
3	4	5
6	7	8

Goal State

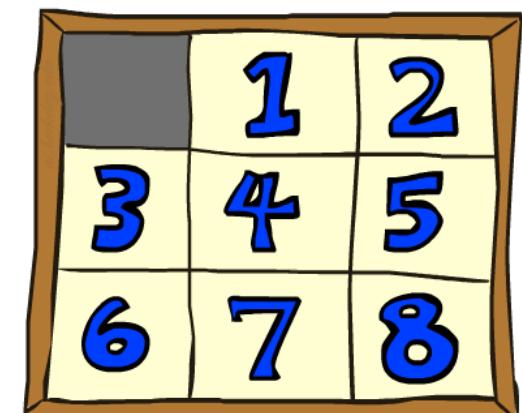
- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

8 Puzzle I

- Heuristic: Number of tiles misplaced
- Why is it admissible?
- $h(\text{start}) = 8$
- This is a *relaxed-problem* heuristic



Start State



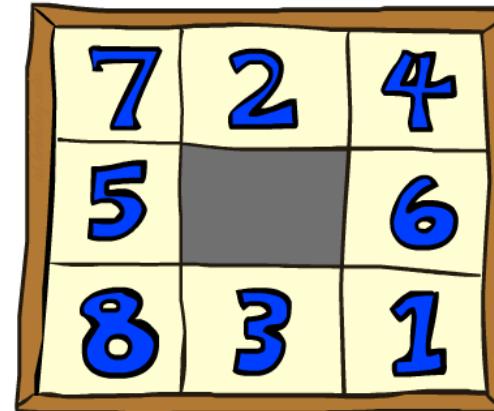
Goal State

Average nodes expanded
when the optimal path has...

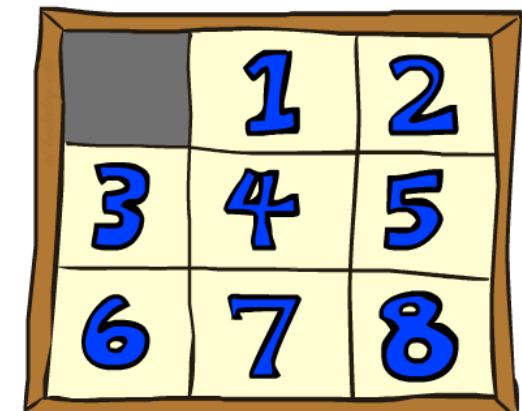
	...4 steps	...8 steps	...12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

8 Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?
- Total *Manhattan* distance
- Why is it admissible?
- $h(\text{start}) = 3 + 1 + 2 + \dots = 18$



Start State



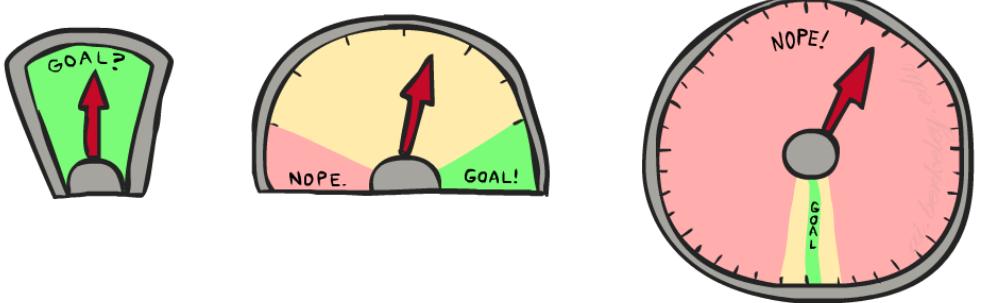
Goal State

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

8 Puzzle III

- How about using the *actual cost* as a heuristic?

- Would it be admissible?
- Would we save on nodes expanded?
- What's wrong with it?



- With A*: a trade-off between quality of estimate and work per node
 - As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

Semi-Lattice of Heuristics

Trivial Heuristics, Dominance

- Dominance: $h_a \geq h_c$ if

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

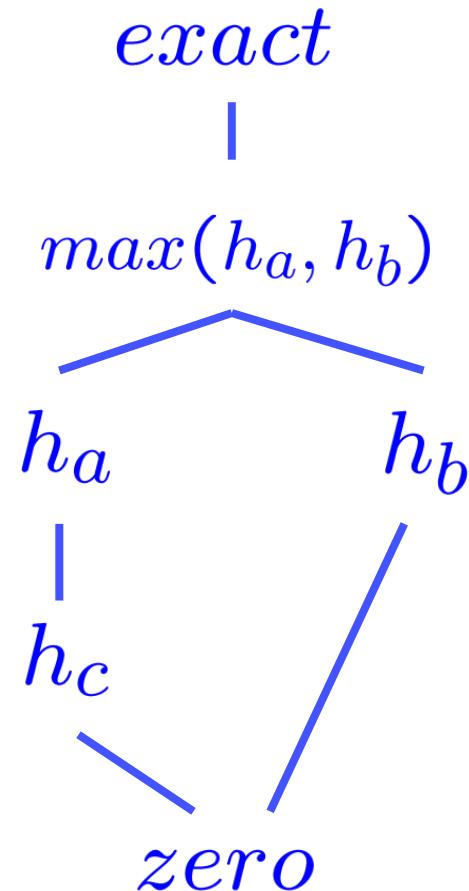
- Heuristics form a semi-lattice:

- Max of admissible heuristics is admissible

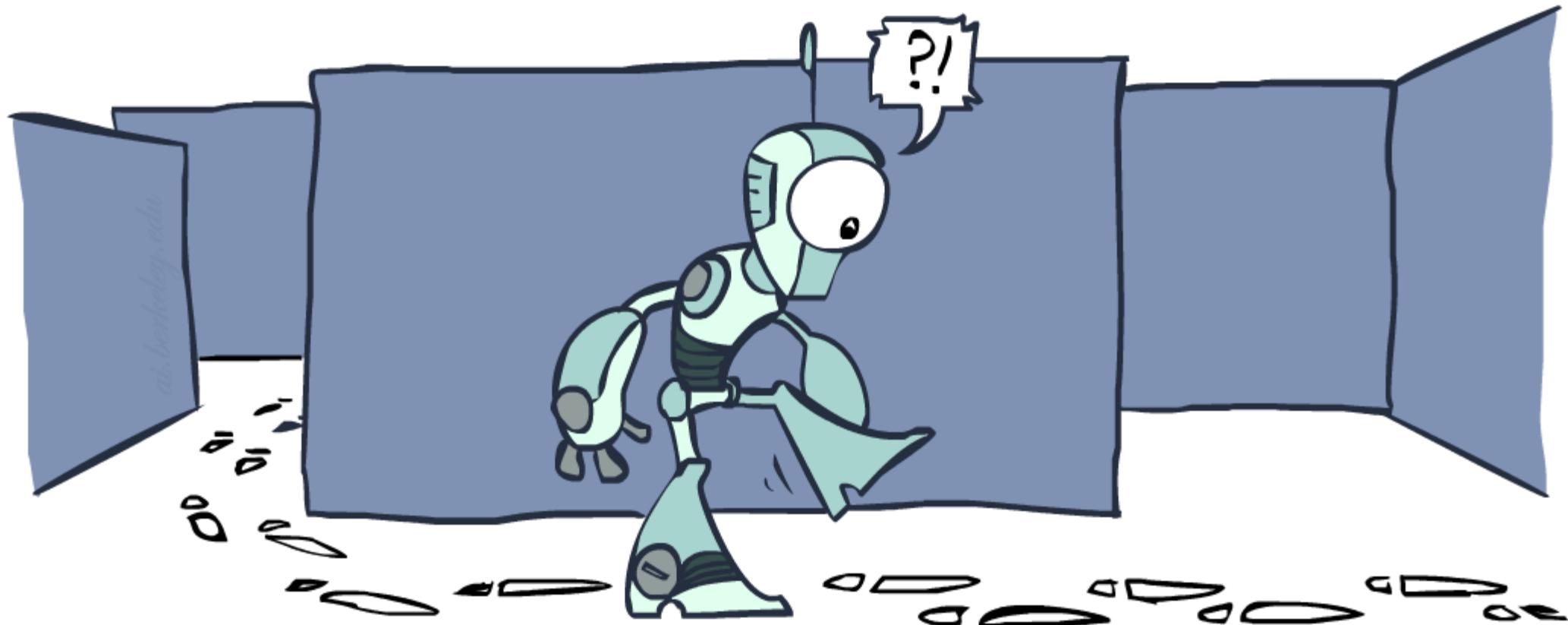
$$h(n) = \max(h_a(n), h_b(n))$$

- Trivial heuristics

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

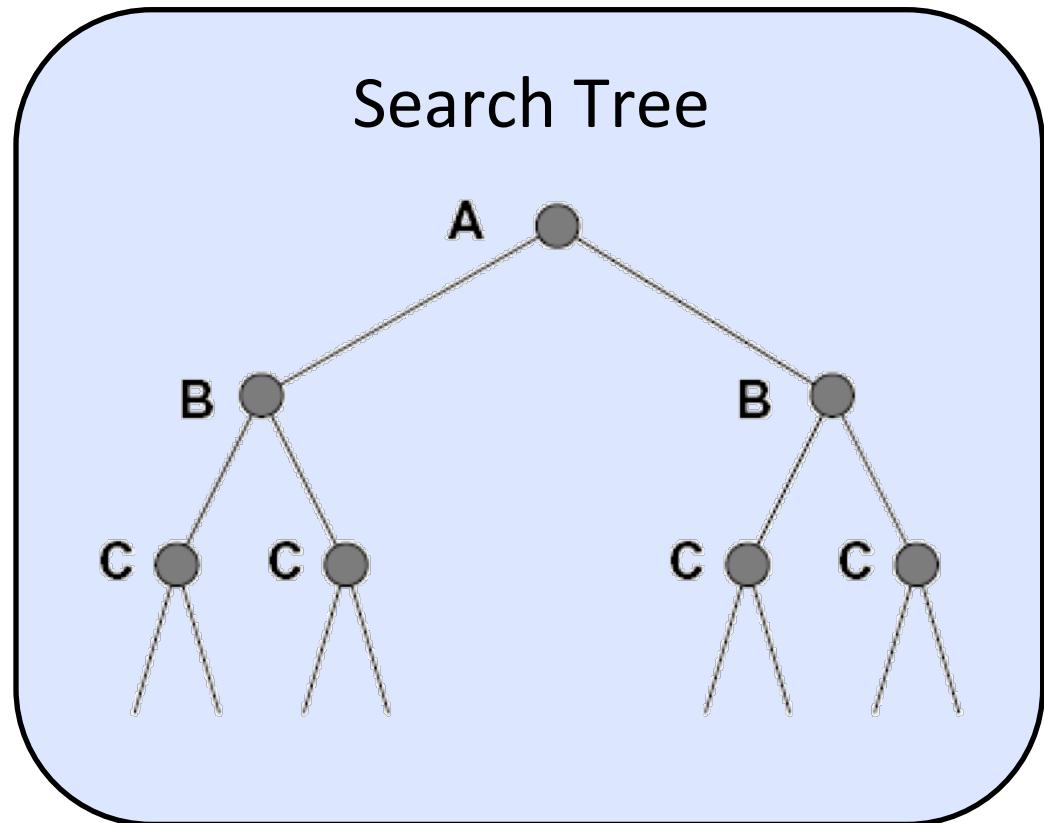
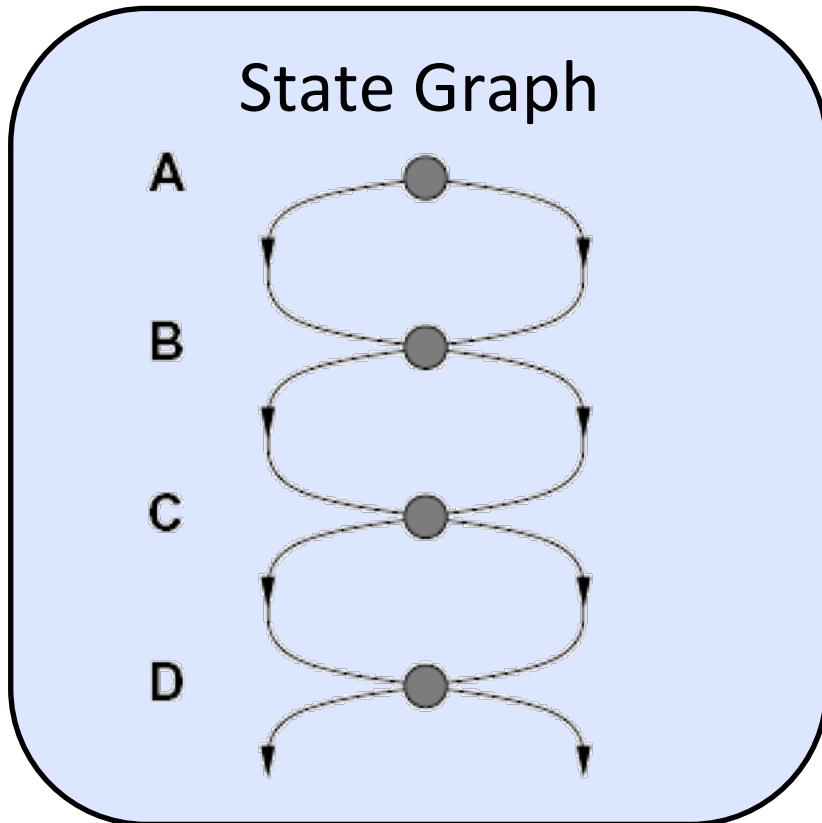


Graph Search



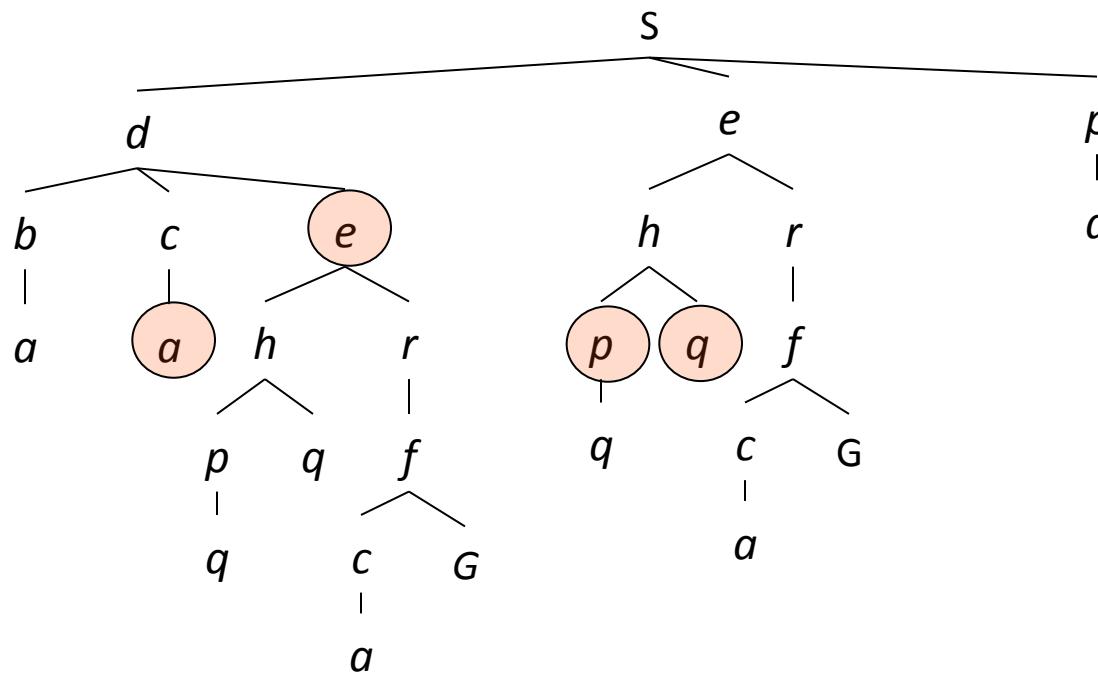
Tree Search: Extra Work!

- Failure to detect repeated states can cause exponentially more work.



Graph Search

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

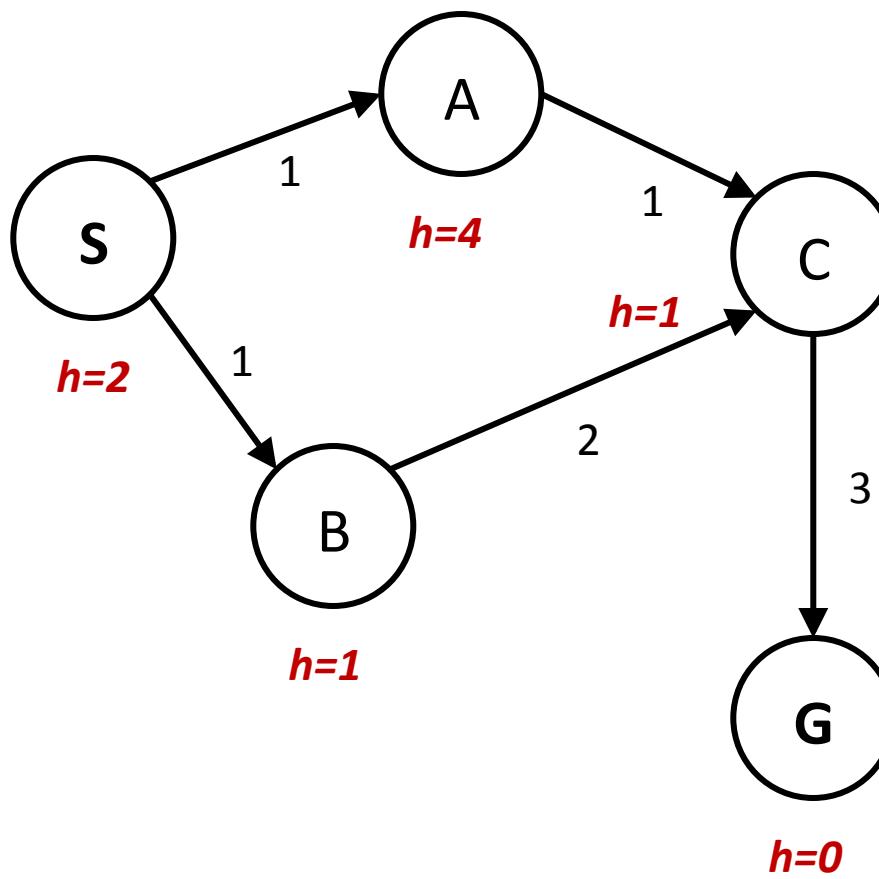


Graph Search

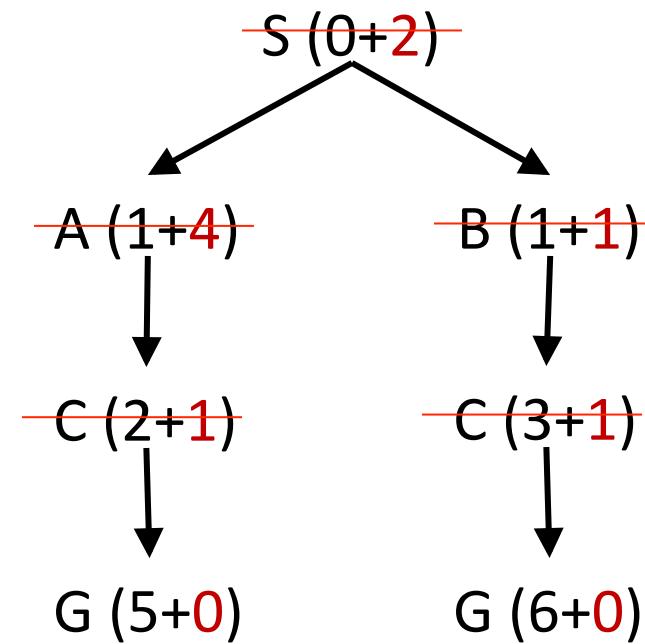
- Idea: never **expand** a state twice
- How to implement:
 - Tree search + set of expanded states (“closed set”)
 - Expand the search tree node-by-node, but...
 - Before expanding a node, check to make sure its state has never been expanded before
 - If not new, skip it, if new add to closed set
- Important: **store the closed set as a set**, not a list
- Can graph search wreck completeness? Why/why not?
- How about optimality?

A* Graph Search Gone Wrong?

State space graph

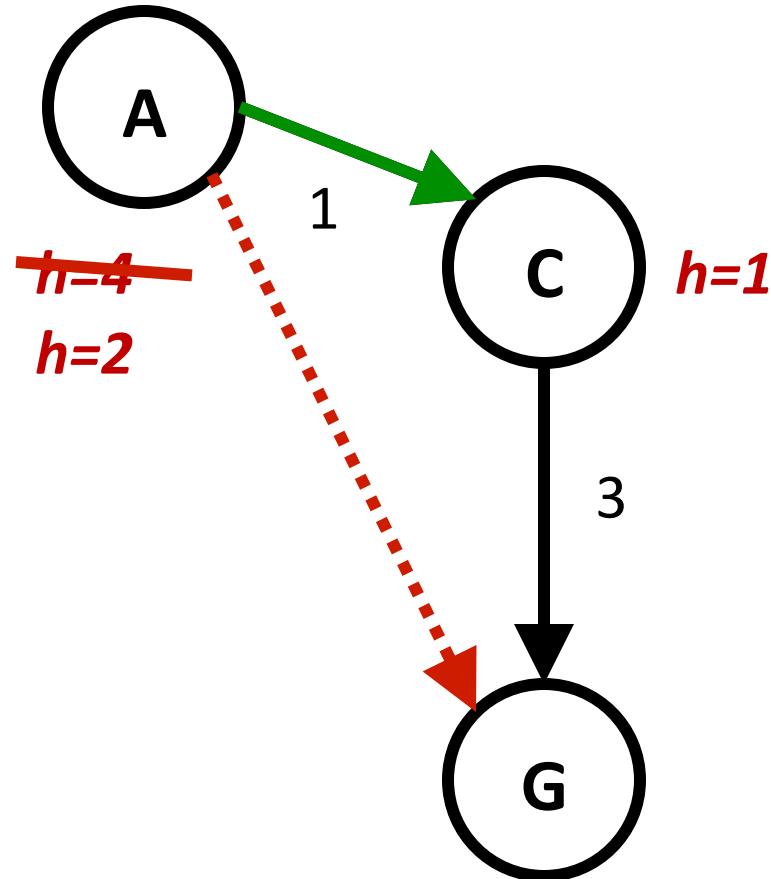


Search tree



Closed Set: S B C A

Consistency of Heuristics

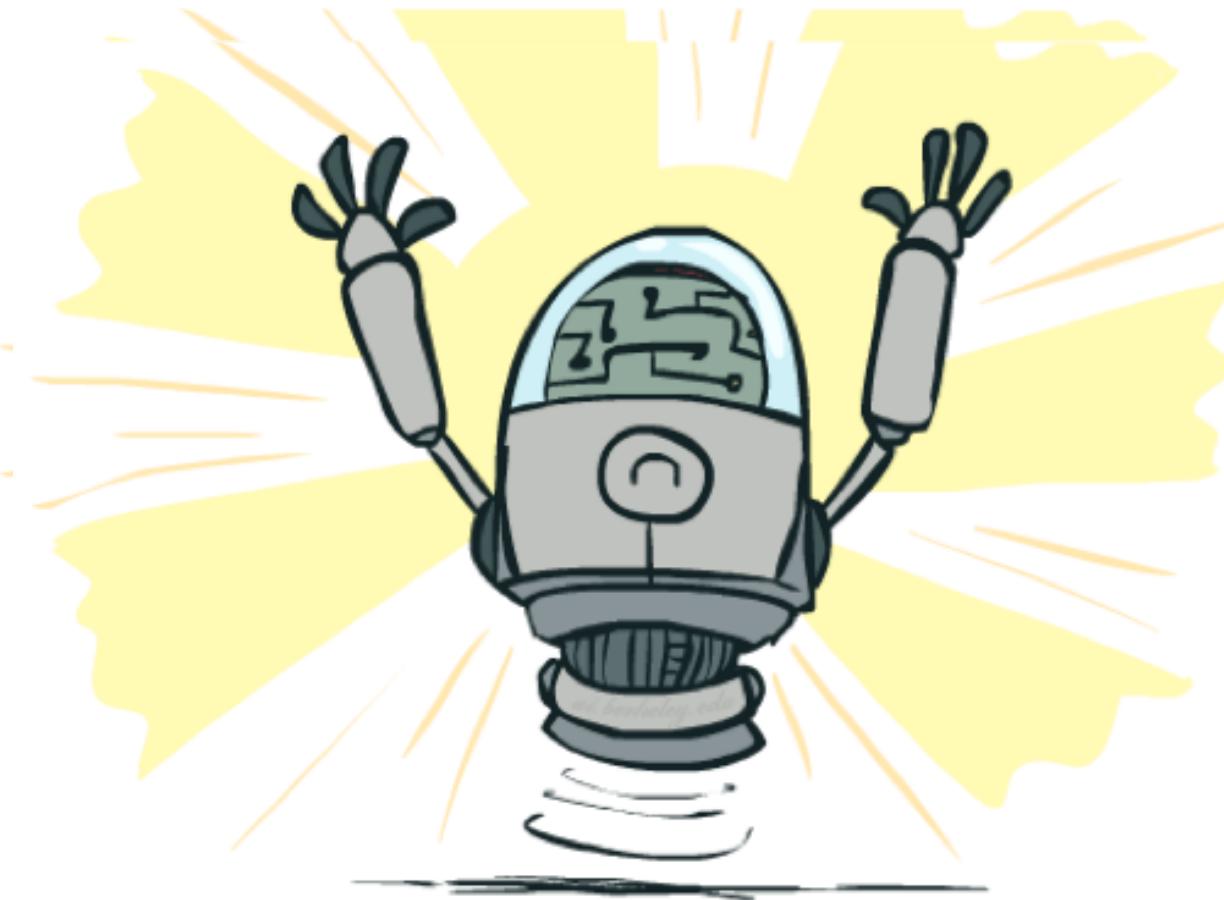


- Main idea: estimated heuristic costs \leq actual costs
 - Admissibility: heuristic cost \leq actual cost to goal
$$h(A) \leq \text{actual cost from } A \text{ to } G$$
 - Consistency: heuristic “arc” cost \leq actual cost for each arc
$$h(A) - h(C) \leq \text{cost}(A \text{ to } C)$$
- Consequences of consistency:
 - The f value along a path never decreases
$$h(A) \leq \text{cost}(A \text{ to } C) + h(C)$$
 - A* graph search is optimal

Optimality of A* Search

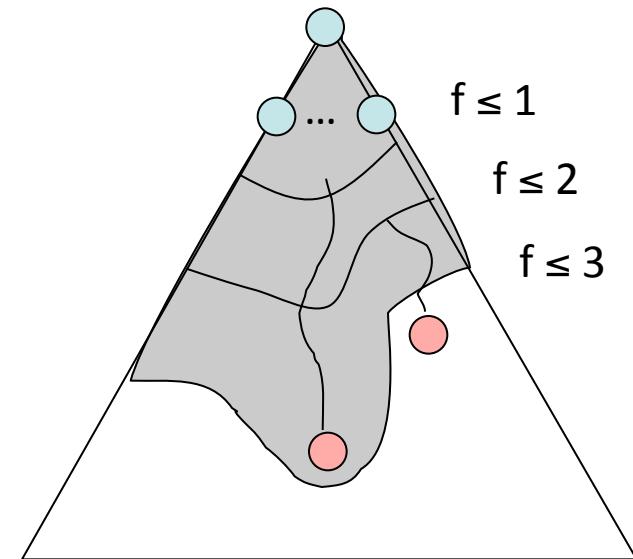
- With a admissible heuristic, Tree A* is optimal.
- With a consistent heuristic, Graph A* is optimal.
- With $h=0$, the same proofs shows that UCS is optimal.
- Skipping proof because of time constraints. Available from:
 - Lecture notes on edge.edx.org.
 - Previous year's lectures on edge.edx.org, edx.org or YouTube.
 - Any book on the topic (e.g. page 97 in Russell & Norvig).

Optimality of A* Graph Search



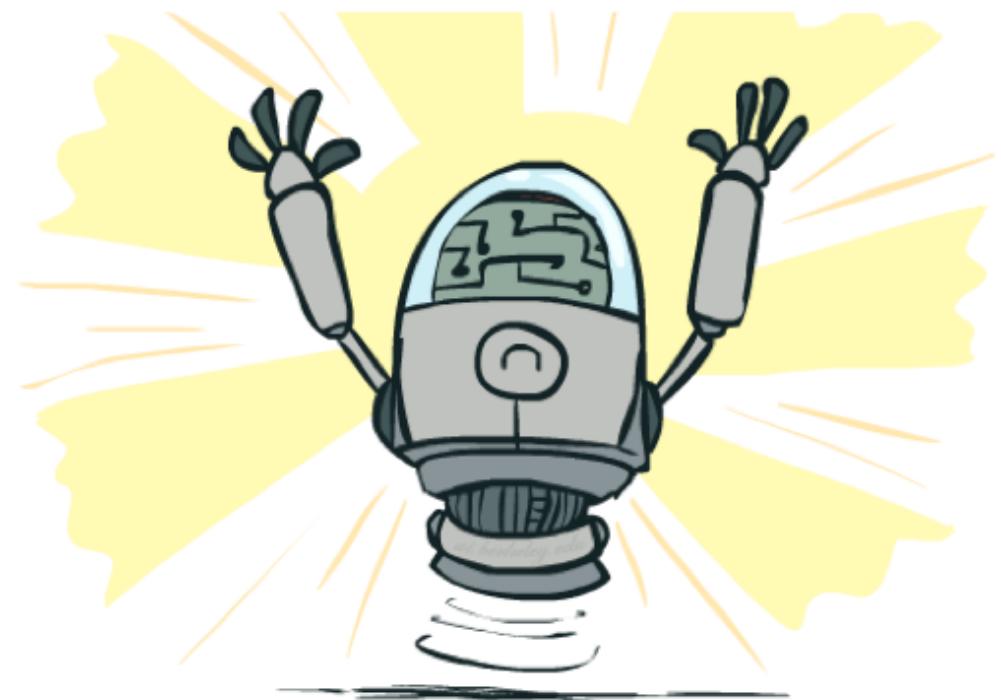
Optimality of A* Graph Search

- Sketch: consider what A* does with a consistent heuristic:
 - Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)
 - Fact 2: For every state s, nodes that reach s optimally are expanded before nodes that reach s suboptimally
- Result: A* graph search is optimal

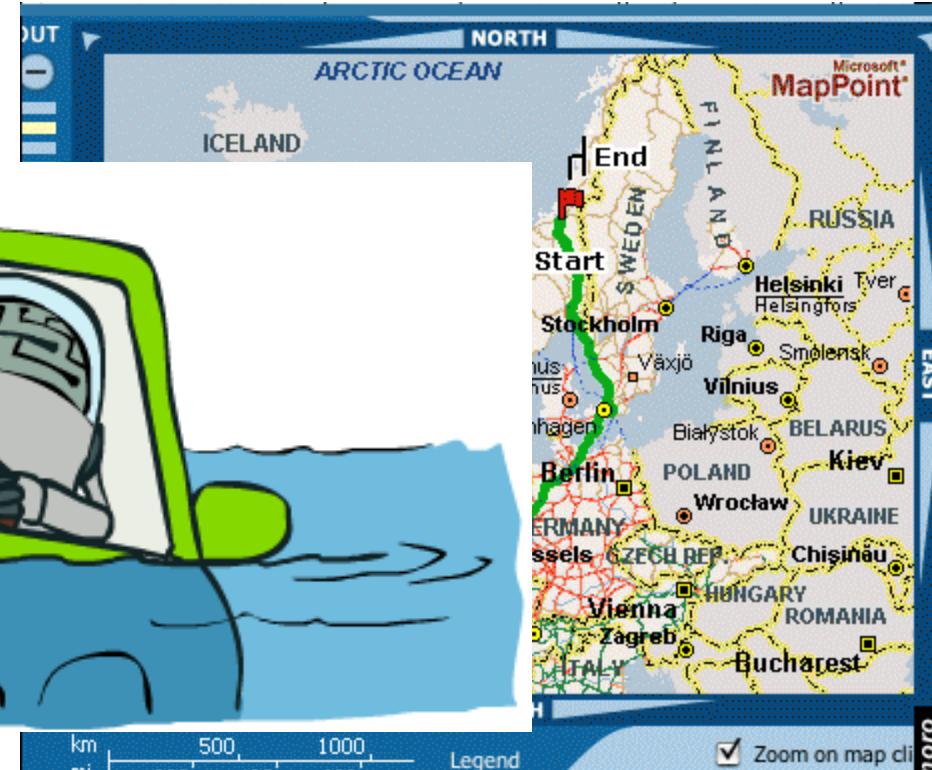
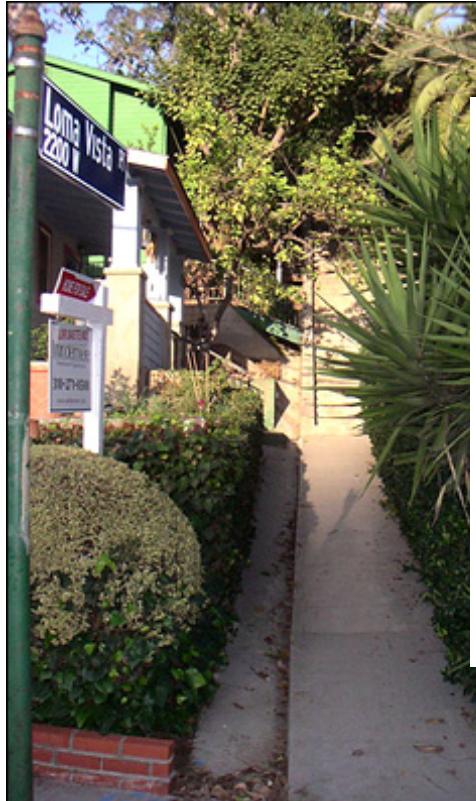


Optimality

- Tree search:
 - A* is optimal if heuristic is admissible
 - UCS is a special case ($h = 0$)
- Graph search:
 - A* optimal if heuristic is consistent
 - UCS optimal ($h = 0$ is consistent)
- Consistency implies admissibility
- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems



Search Gone Wrong?



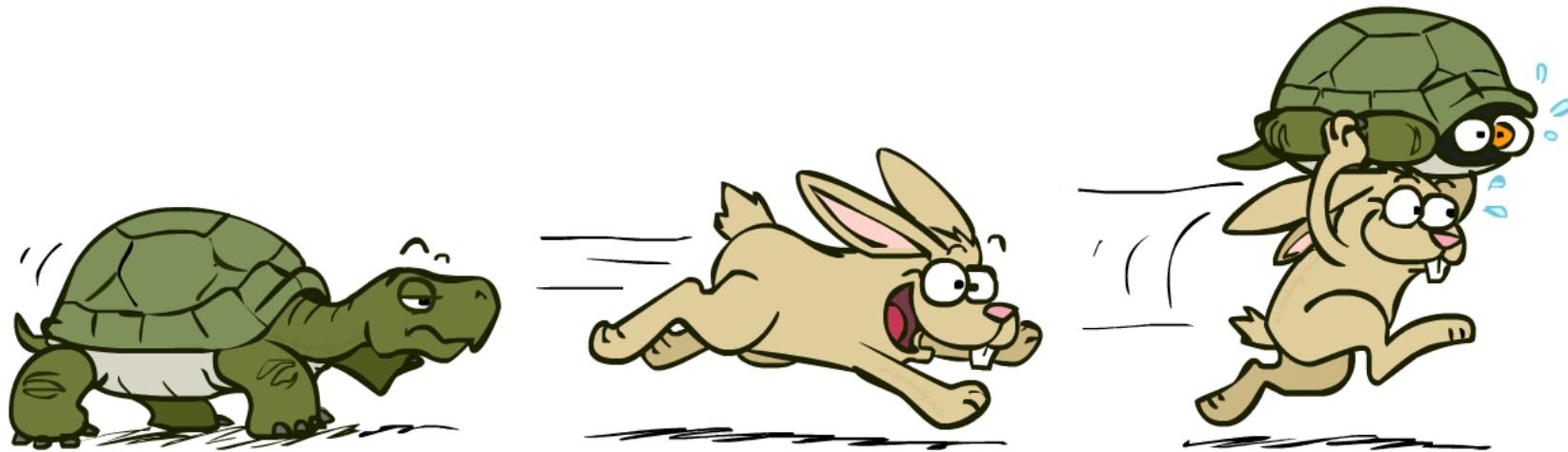
Start: Haugesund, Rogaland, Norway
End: Trondheim, Sør-Trøndelag, Norway
Total Distance: 2713.2 Kilometers
Estimated Total Time: 47 hours, 31 minutes

A*: Summary



A*: Summary

- A* uses both backward costs and (estimates of) forward costs
- A* is optimal with admissible / consistent heuristics
- Heuristic design is key: often use relaxed problems



Tree Search Pseudo-Code

```
function TREE-SEARCH(problem, fringe) return a solution, or failure
  fringe  $\leftarrow$  INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node  $\leftarrow$  REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then return node
    for child-node in EXPAND(STATE[node], problem) do
      fringe  $\leftarrow$  INSERT(child-node, fringe)
    end
  end
```

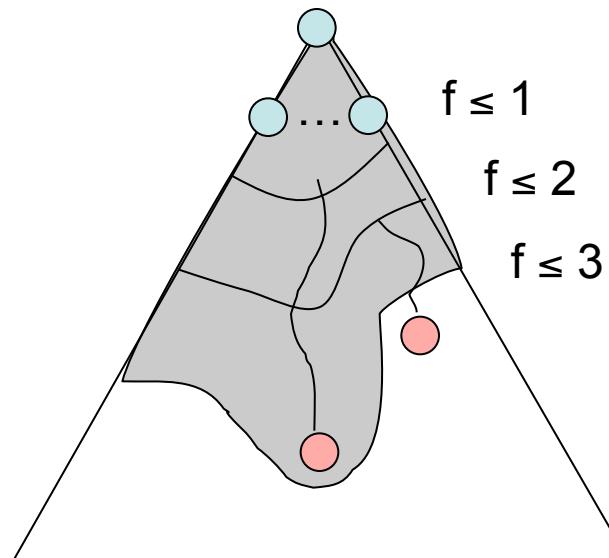
Graph Search Pseudo-Code

```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
  closed ← an empty set
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then return node
    if STATE[node] is not in closed then
      add STATE[node] to closed
      for child-node in EXPAND(STATE[node], problem) do
        fringe ← INSERT(child-node, fringe)
    end
  end
```

Optimality of A* Graph Search

- Consider what A* does:
 - Expands nodes in increasing total f value (f-contours)
Reminder: $f(n) = g(n) + h(n) = \text{cost to } n + \text{heuristic}$
 - Proof idea: the optimal goal(s) have the lowest f value, so it must get expanded first

There's a problem with this argument. What are we assuming is true?



Optimality of A* Graph Search

Proof:

- New possible problem: some n on path to G^* isn't in queue when we need it, because some worse n' for the same state dequeued and expanded first (disaster!)
- Take the highest such n in tree
- Let p be the ancestor of n that was on the queue when n' was popped
- $f(p) < f(n)$ because of consistency
- $f(n) < f(n')$ because n' is suboptimal
- p would have been expanded before n'
- Contradiction!

