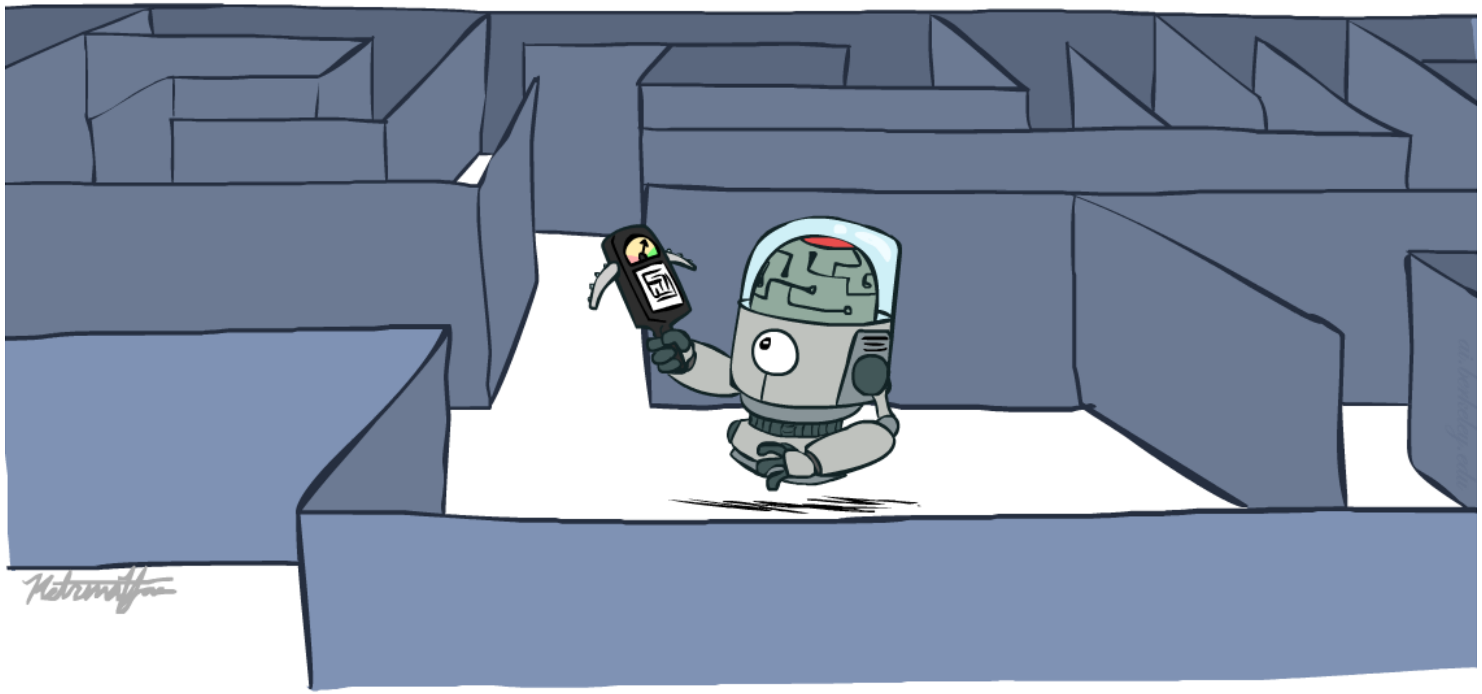
CS 188: Artificial Intelligence

Search Continued

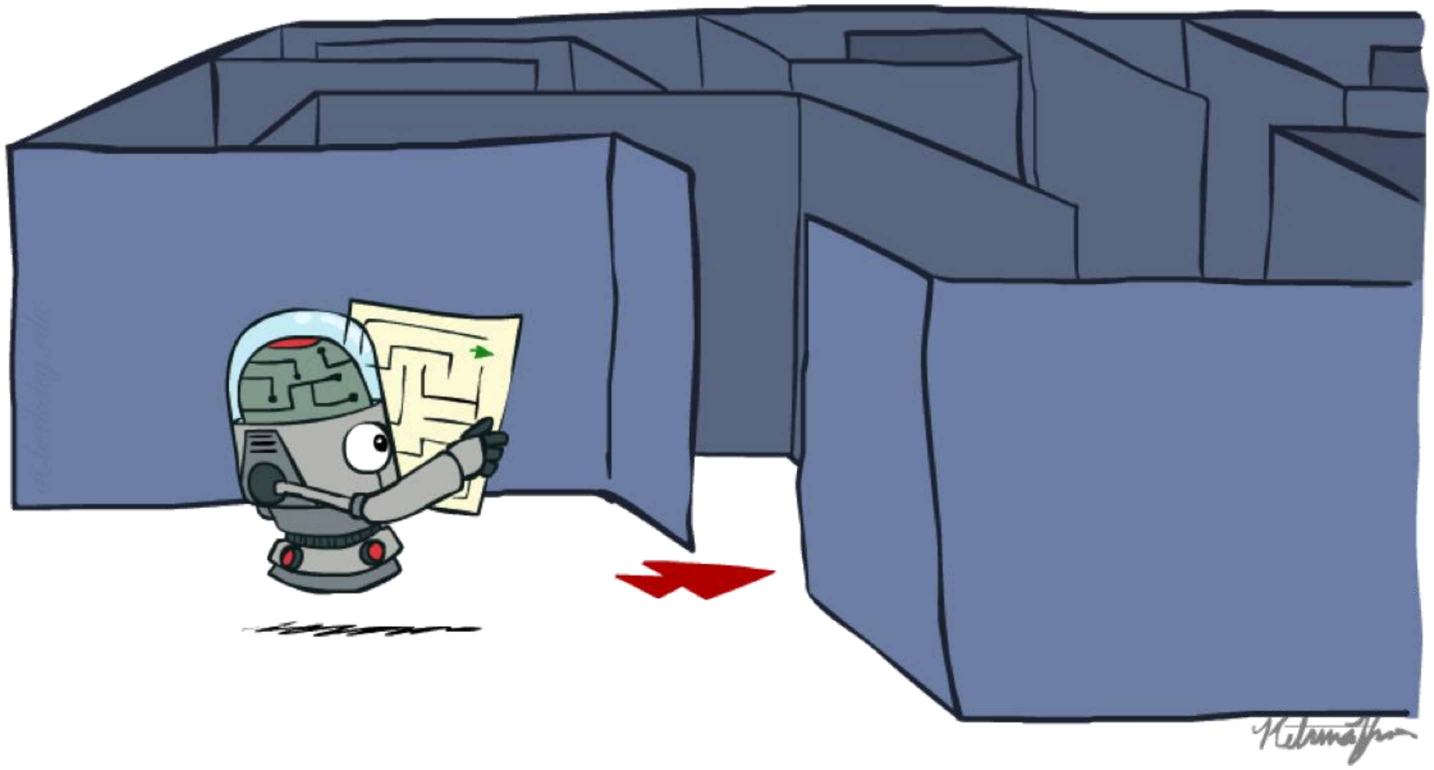


Instructors: Anca Dragan

University of California, Berkeley

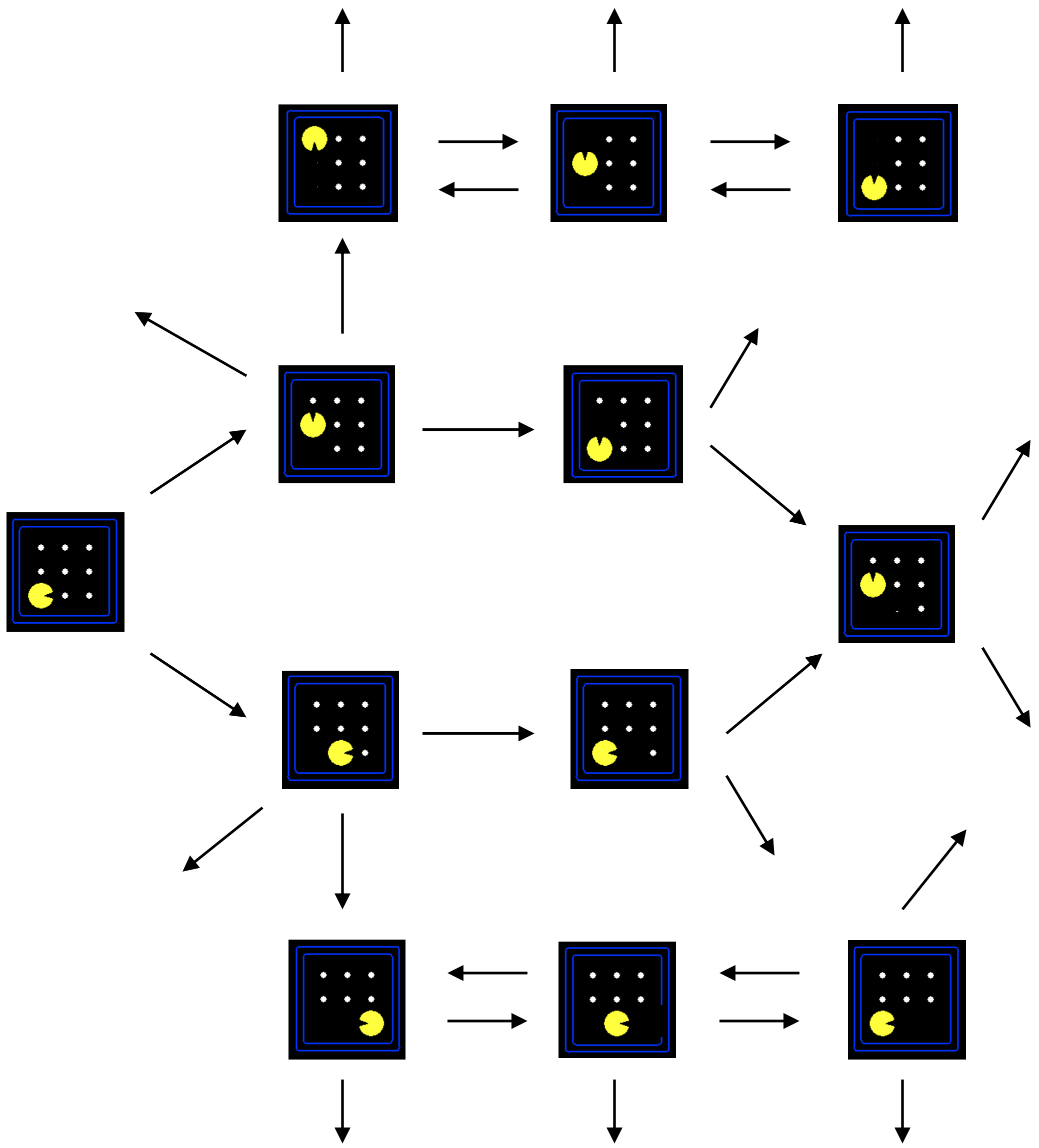
[These slides adapted from Dan Klein and Pieter Abbeel; ai.berkeley.edu]

Recap: Search

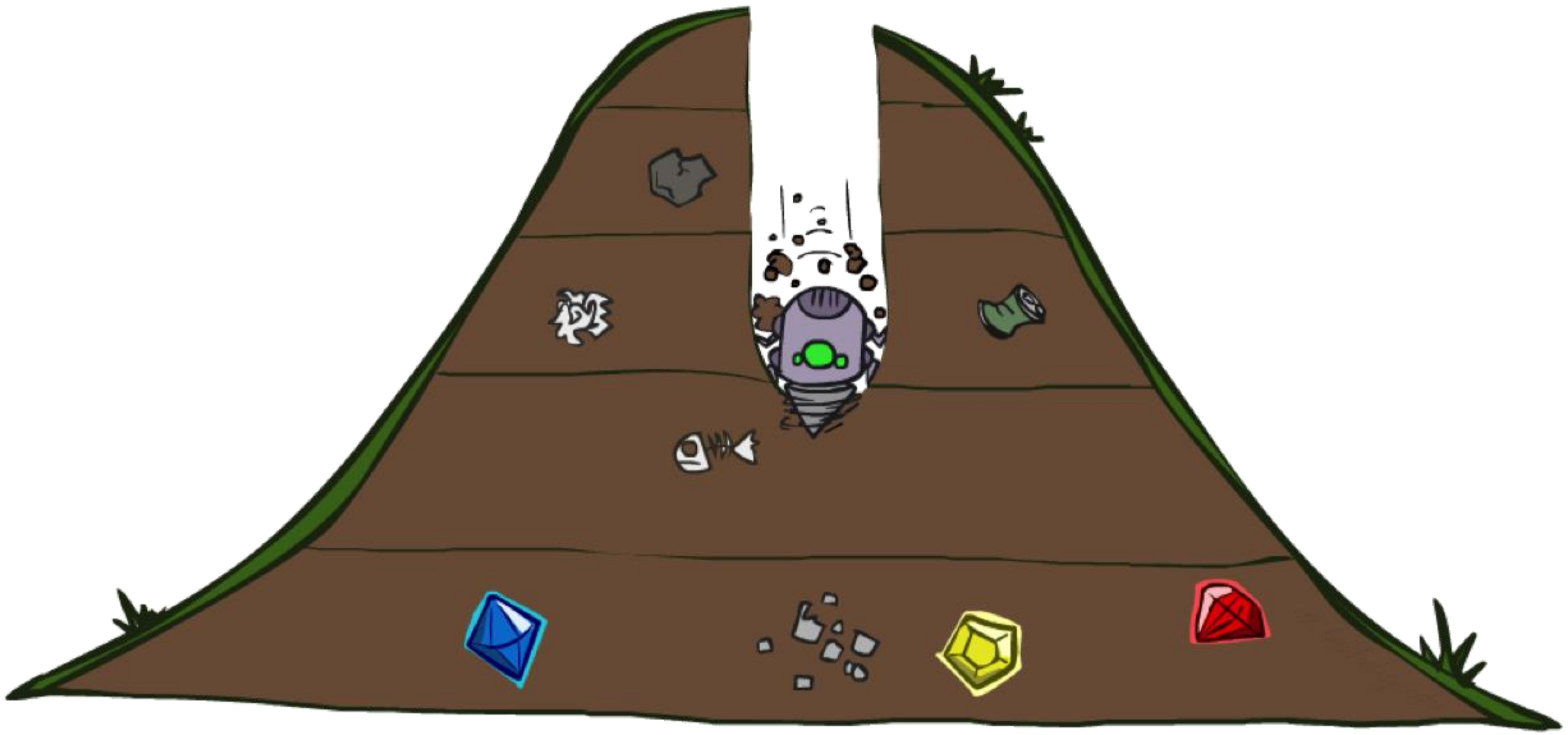


Search

- Search problem:
  - States (abstraction of the world)
  - Actions (and costs)
  - Successor function (world dynamics):
    - {s’ |s,a->s’ }
  - Start state and goal test



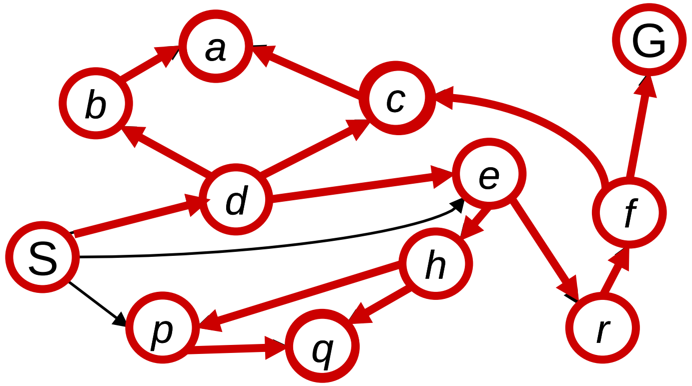
Depth-First Search

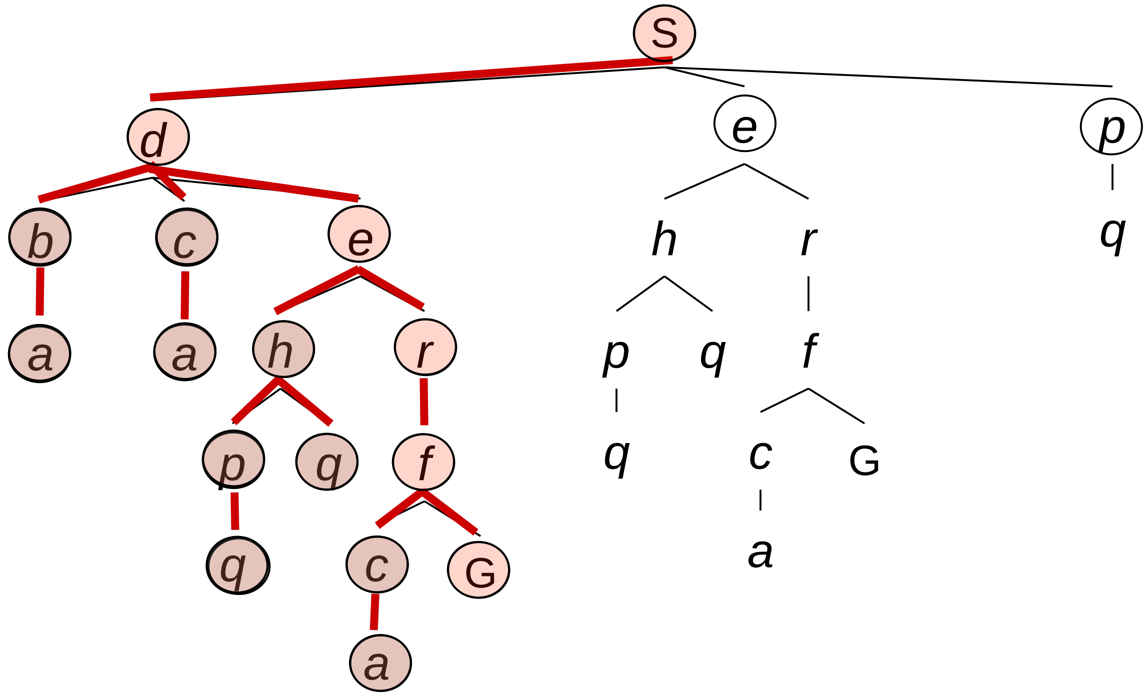


Depth-First Search

Strategy: expand a deepest node first

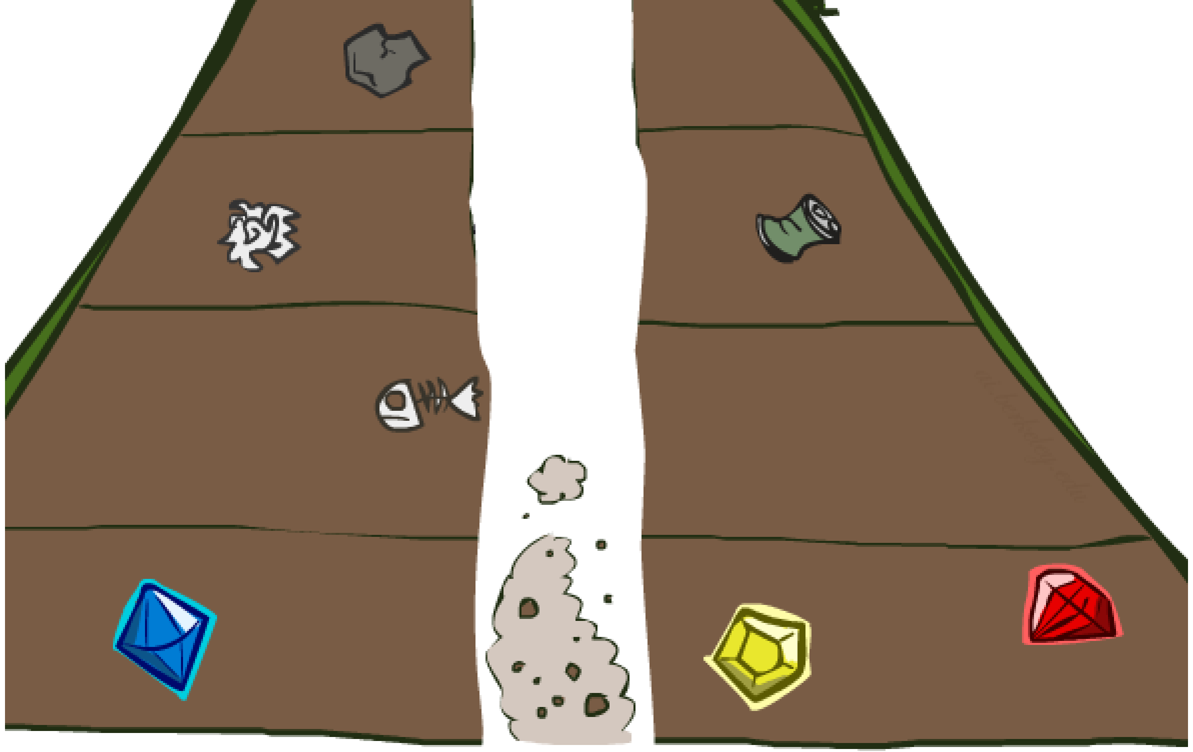
Implementation: Fringe is a LIFO stack





Search Algorithm Properties

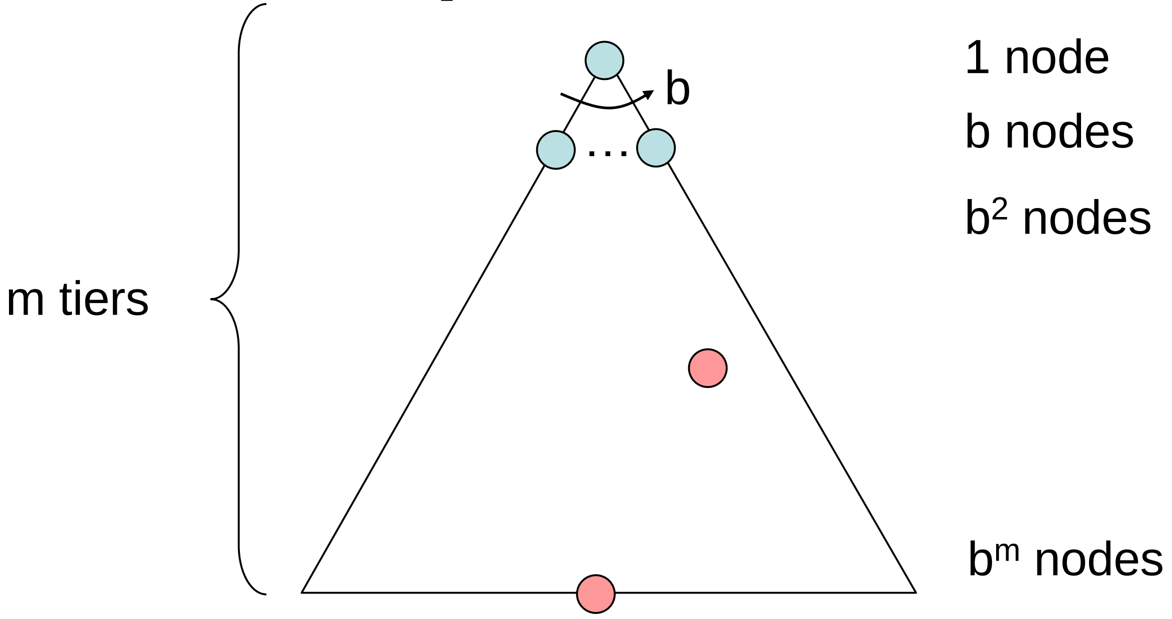
- DFS’ s advantage is that is save the space, for the fact that we don’ t need to store so many candidates.



Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
  - Return in finite time if not?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

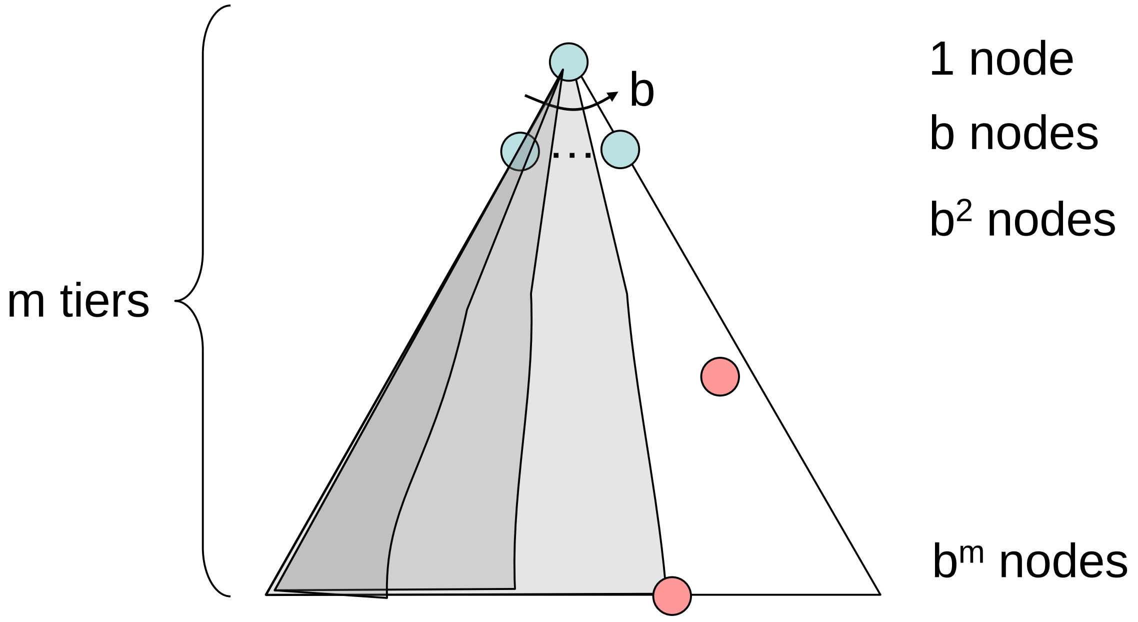
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths



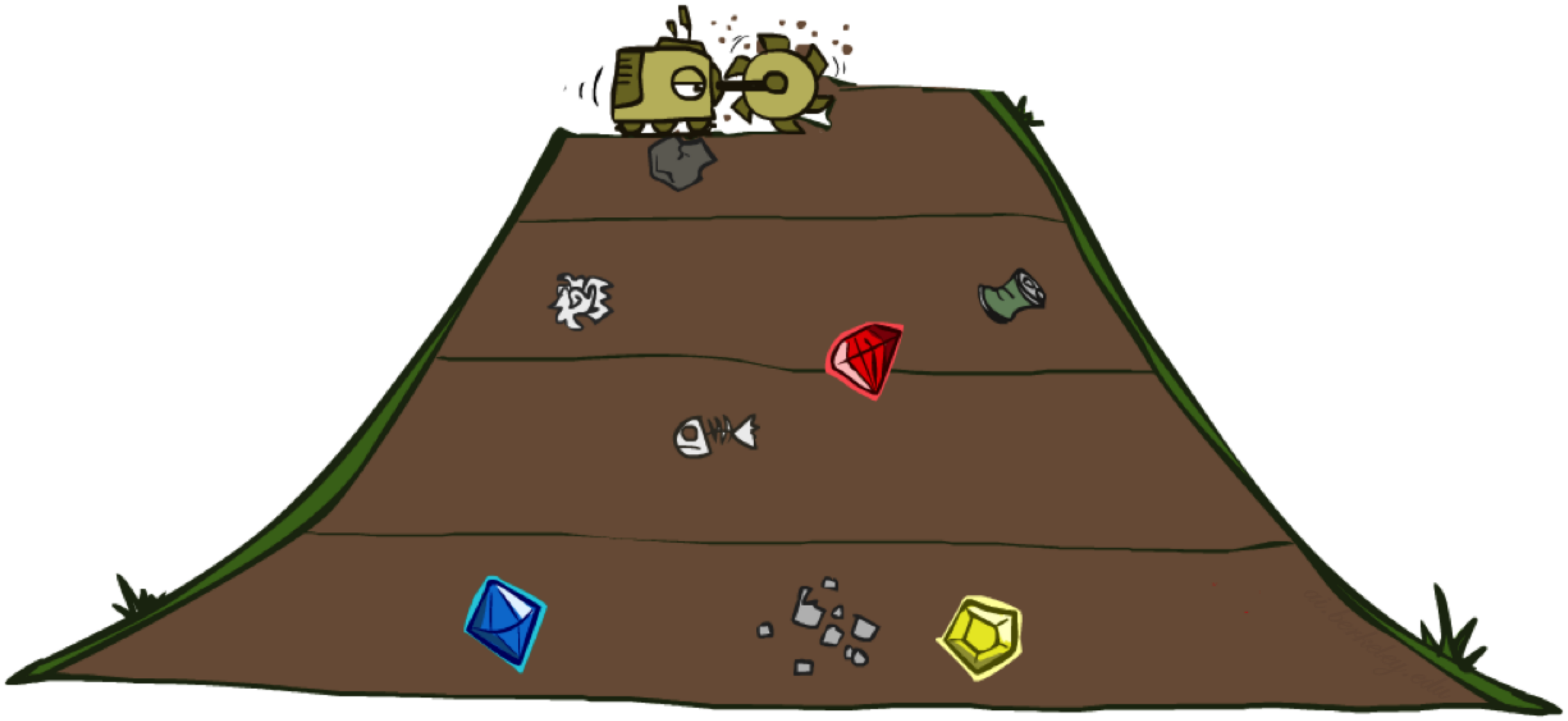
- Number of nodes in entire tree?
  - $1 + b + b^2 + \cdots.\ b^m = O(b^m)$

Depth-First Search (DFS) Properties

- What nodes DFS expand?
  - Some left prefix of the tree.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the fringe take?
  - Only has **siblings on path to root**, so $O(bm)$
- Is it complete?
  - m could be infinite, so only if we prevent cycles (more later)
- Is it optimal?
  - No, it finds the "leftmost" solution, regardless of depth or cost



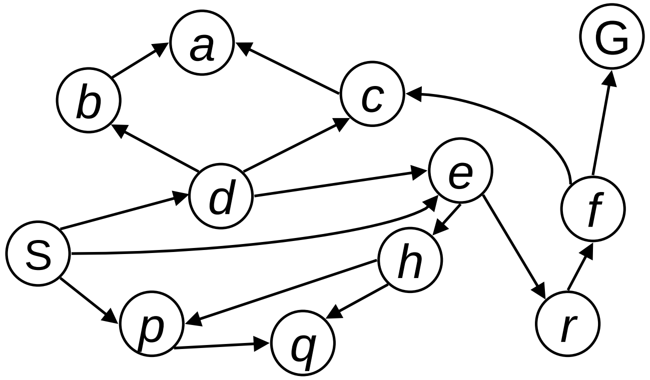
Breadth-First Search



Breadth-First Search

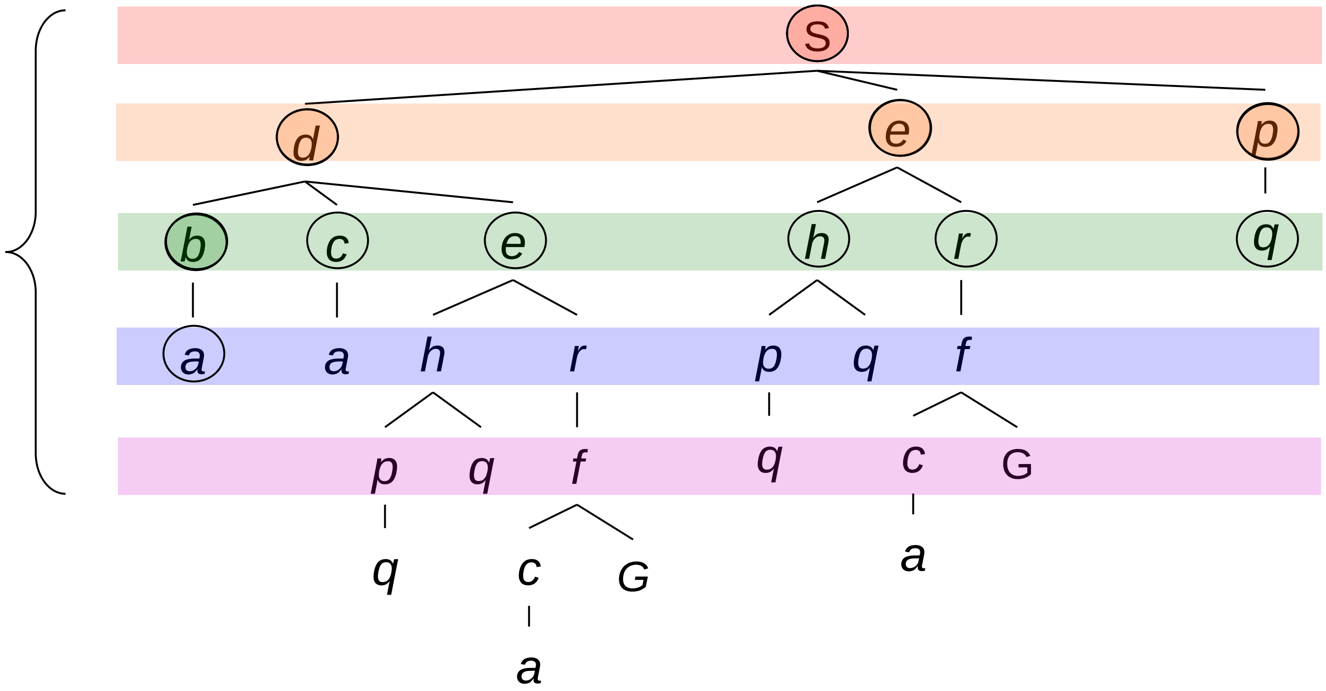
Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue



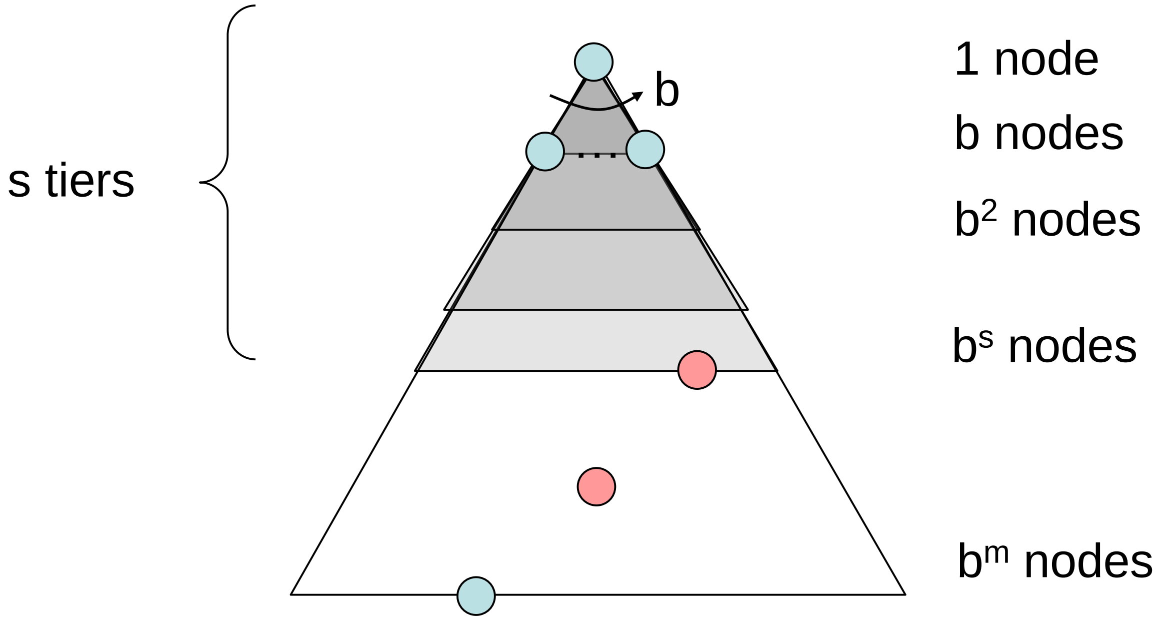
Search

Tiers

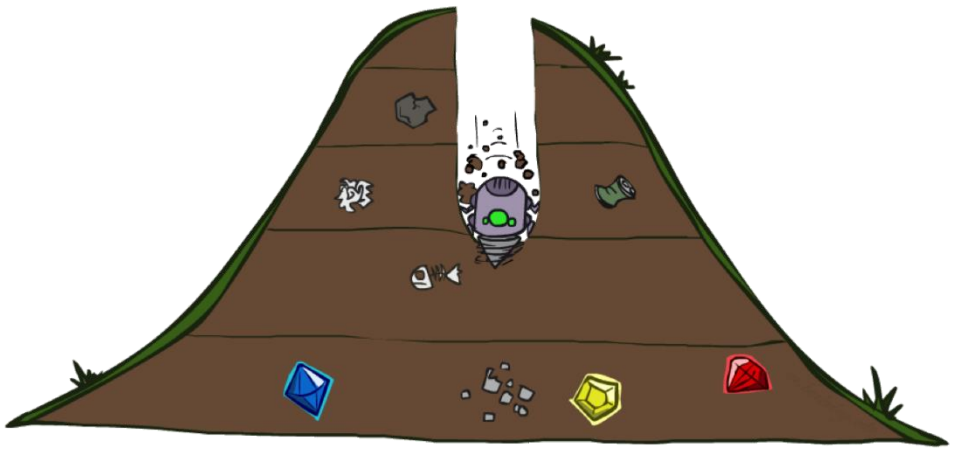


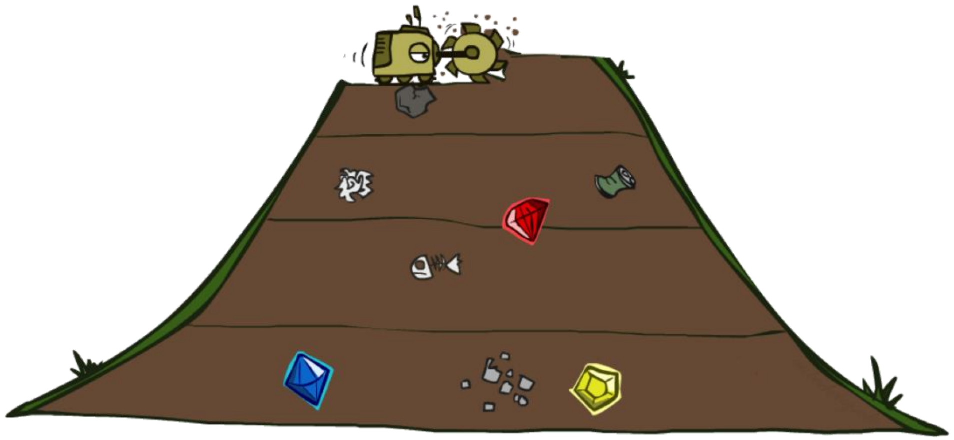
Breadth-First Search (BFS) Properties

- 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! (if no solution, still need depth != ∞)
- Is it optimal?
  - Only if costs are all 1 (more on costs later)



Quiz: DFS vs BFS





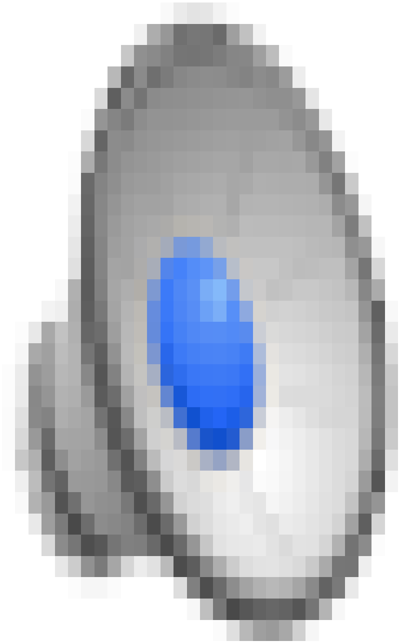
DFS vs BFS

- When will BFS outperform DFS?

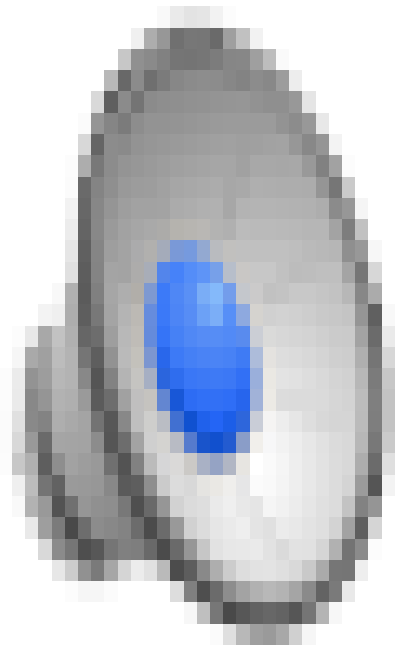
- When will DFS outperform BFS?

[Demo: dfs/bfs maze water (L2D6)]

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

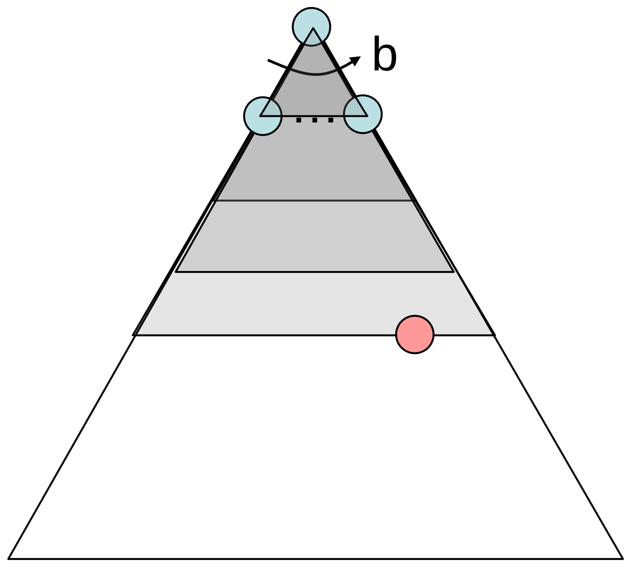


Video of Demo Maze Water DFS/BFS (part 2)



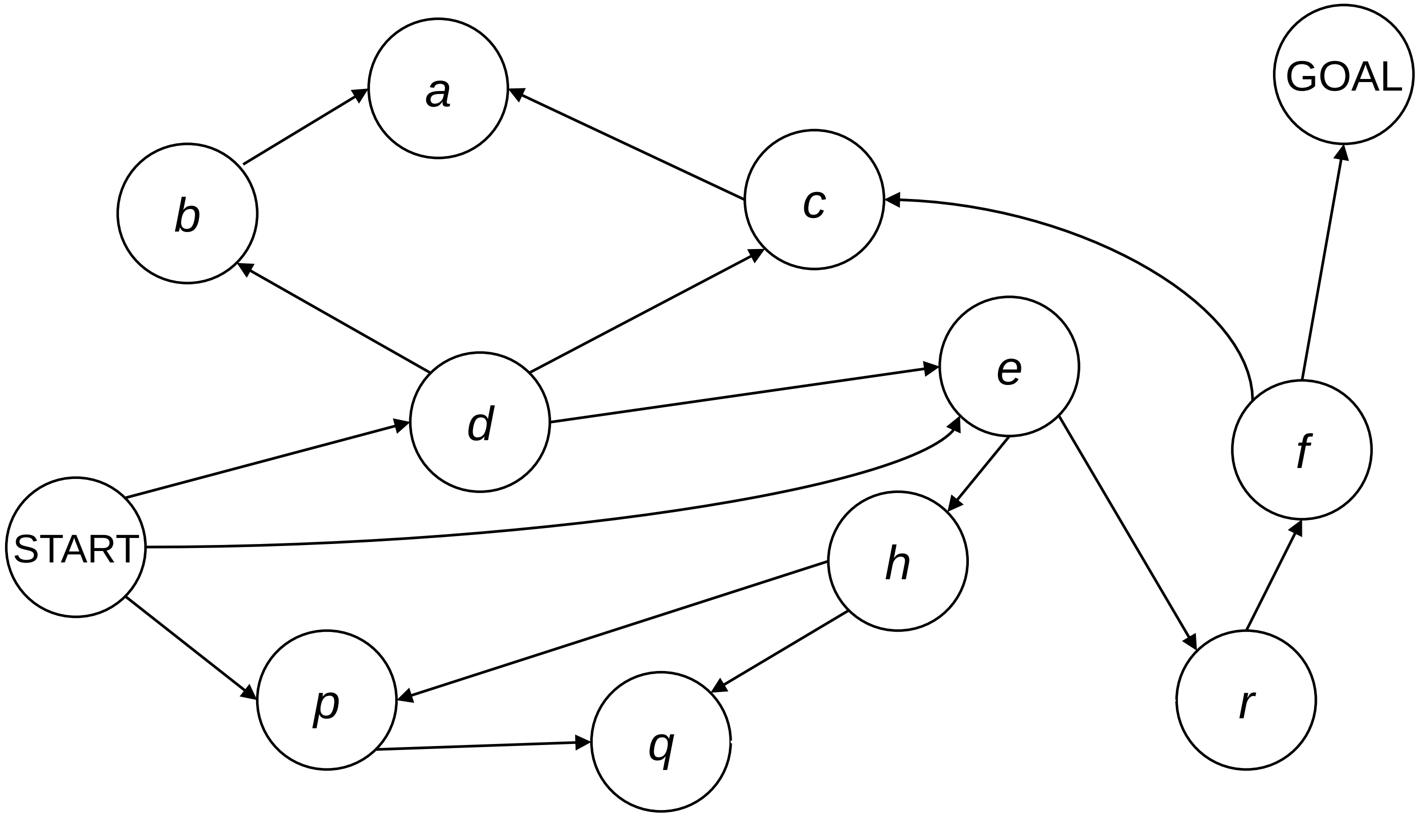
Iterative Deepening

- Idea: get DFS' s space advantage with BFS' s time / shallow-solution advantages
  - 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. …..

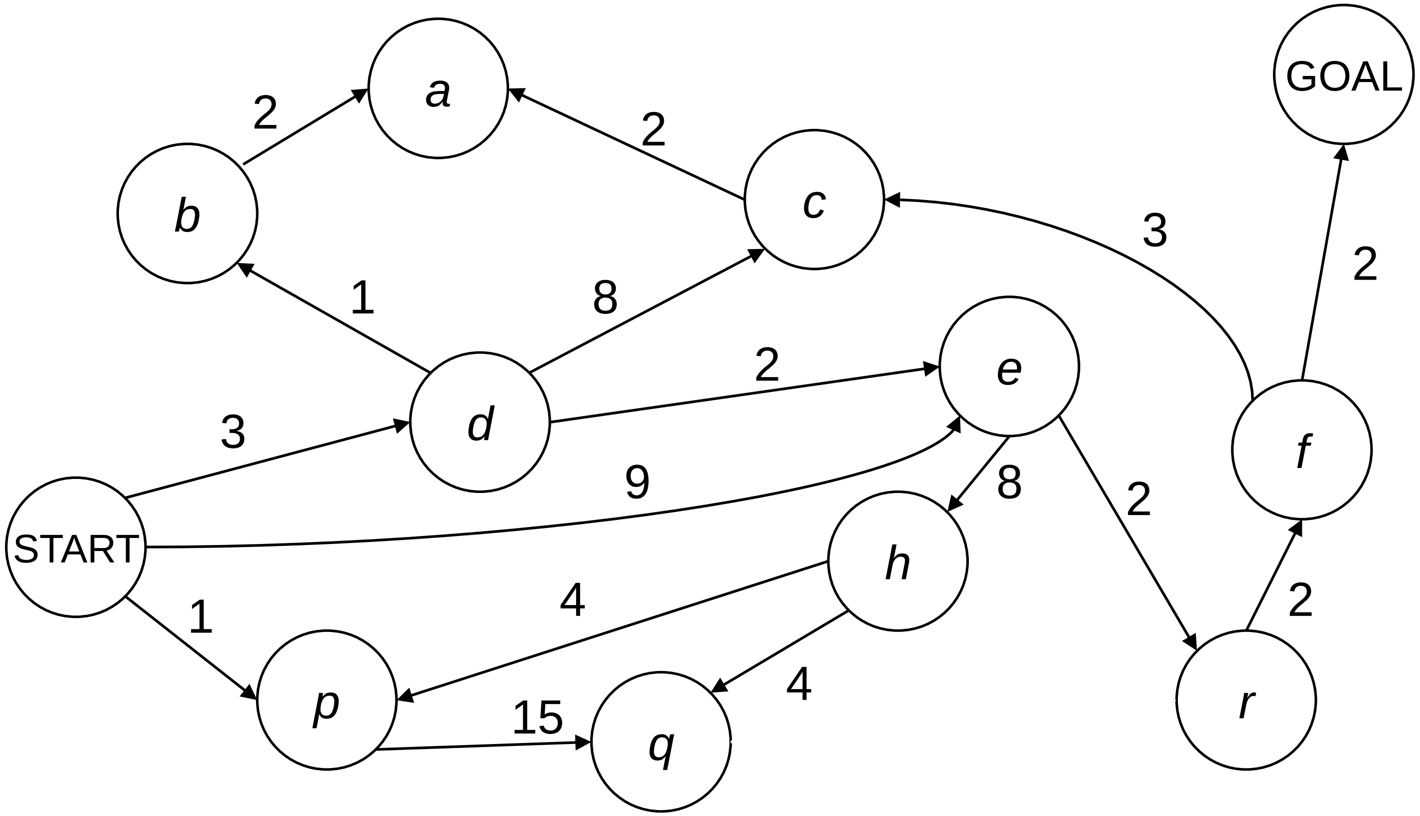


- Isn' t that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!

Cost-Sensitive Search



Cost-Sensitive Search

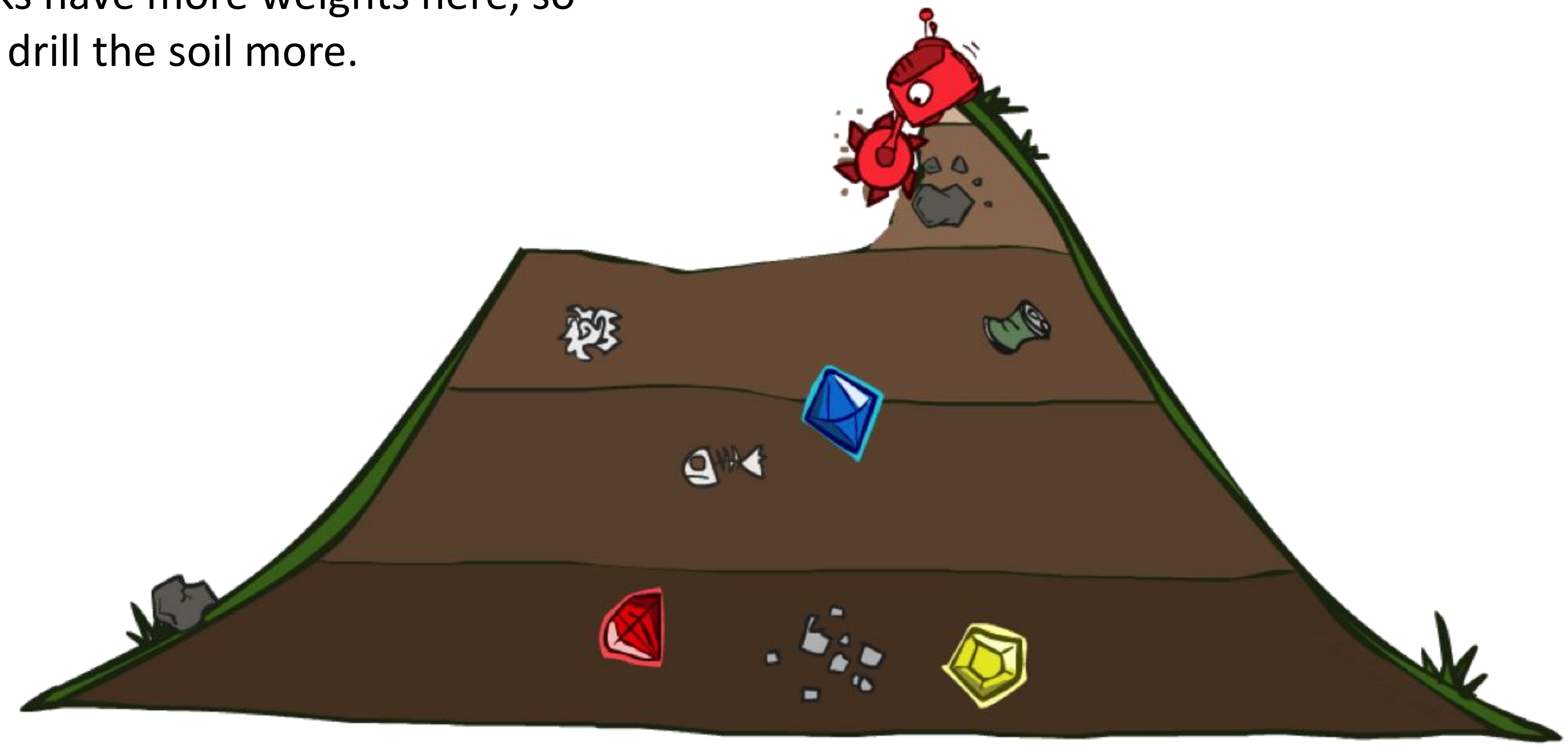


BFS finds the shortest path in terms of number of actions. It **does not find the** least-cost **path.** We will now cover a similar algorithm which does find the least-cost path.->UCS

How?

Uniform Cost Search

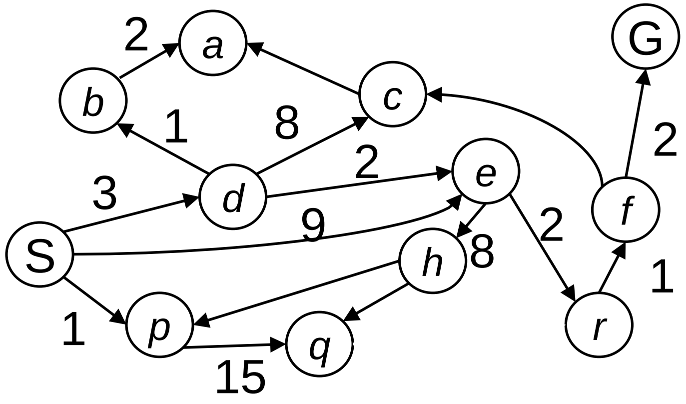
The rocks have more weights here, so
that we drill the soil more.



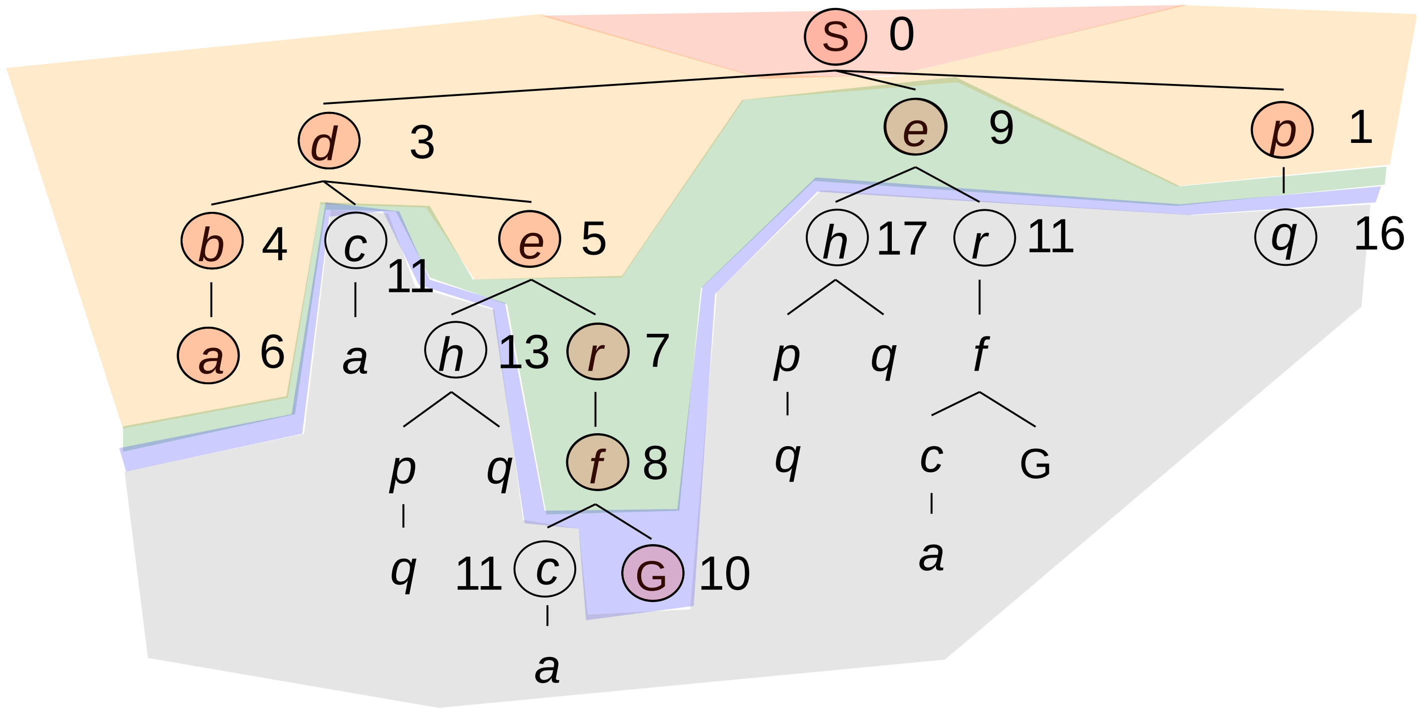
Uniform Cost Search

Strategy: expand a *cheapest node* *first:*

Fringe is a priority queue (priority: cumulative cost)

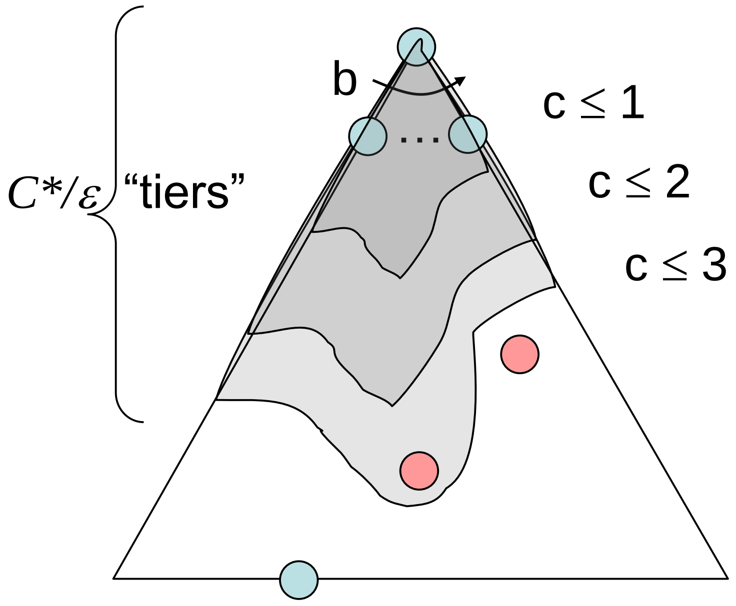


Cost contours



Uniform Cost Search (UCS) Properties

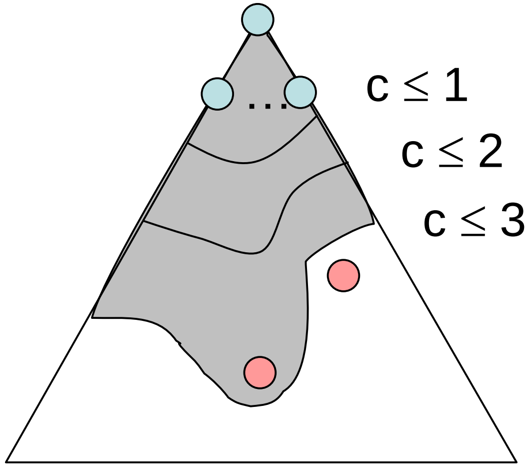
- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs $C*$ and arcs cost **at least** ε, then the "**effective depth**" is roughly $C*/\varepsilon$
  - Takes time $O(b^{C*/\varepsilon})$ (exponential in effective depth)
- How much space 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! (if no solution, still need depth != ∞)
- Is it optimal?
  - Yes! (Proof via A*)

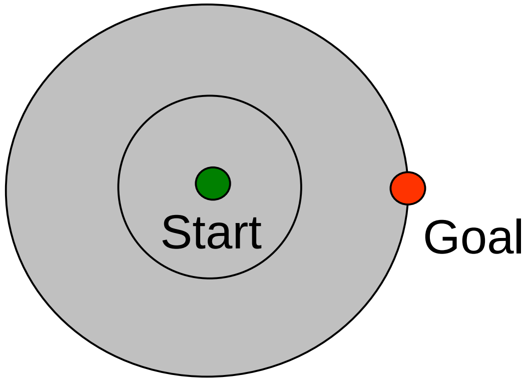


Uniform Cost Issues

- Remember: UCS explores increasing cost contours
- The good: **UCS is complete and optimal!**

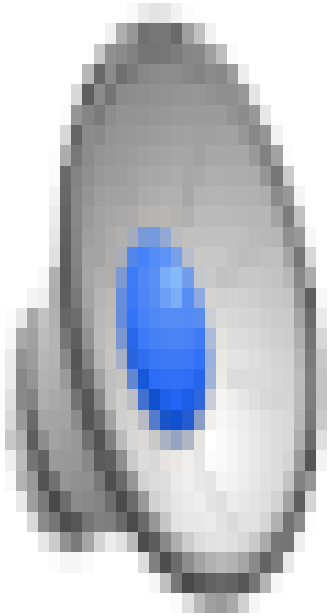
  DFS is not complete and not optimal, BFS is optimal when all the cost are 1(shortest path , but not cheapest path) but it still can ensure that it’ s complete.
- The bad:
  - Explores options in every “direction”
  - **No information about goal location**
- We’ ll fix that soon!



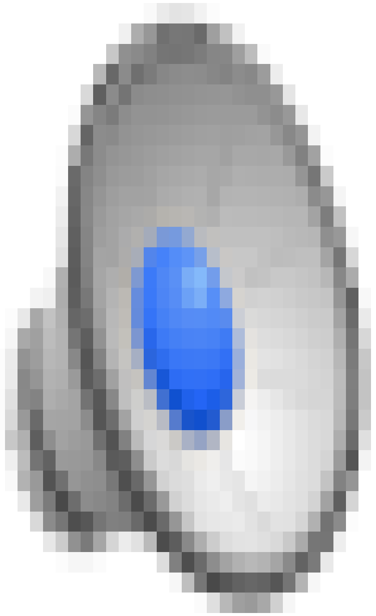


[Demo: empty grid UCS (L2D5)]
[Demo: maze with deep/shallow water DFS/BFS/UCS (L2D7)]

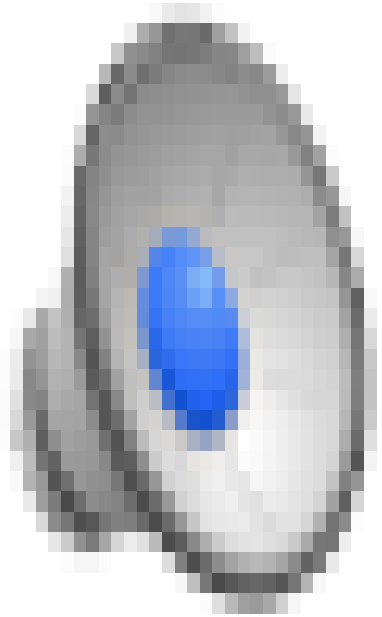
Video of Demo Empty UCS



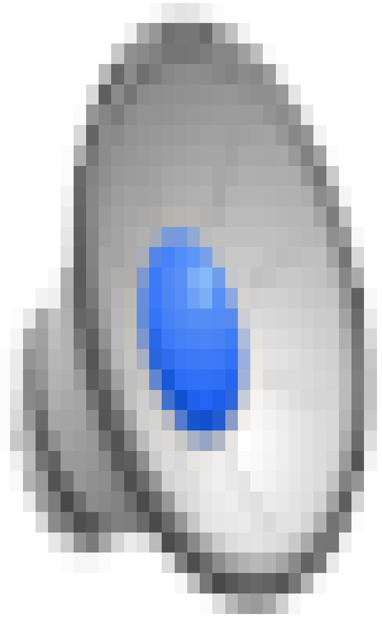
Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 1) BFS



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

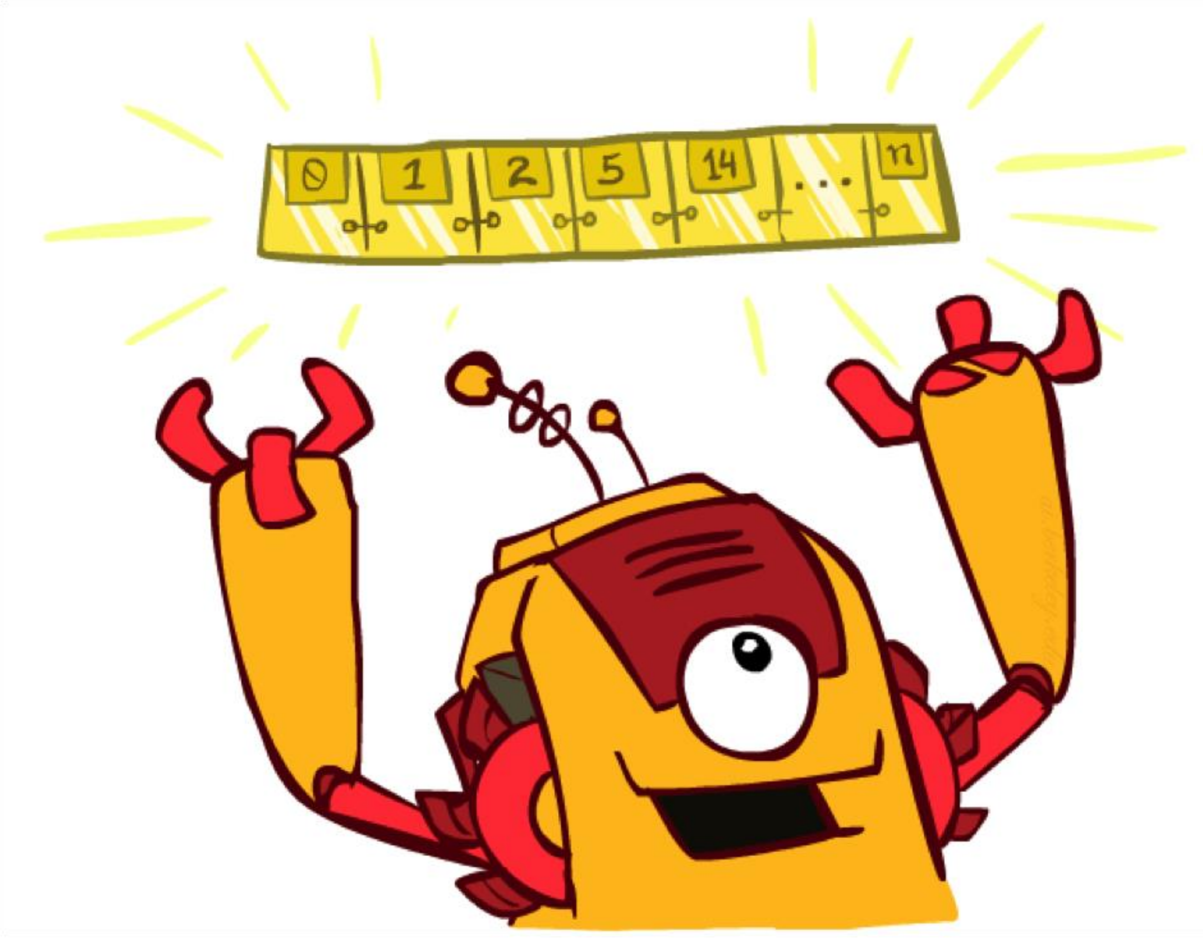


Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 3) DFS



The One Queue

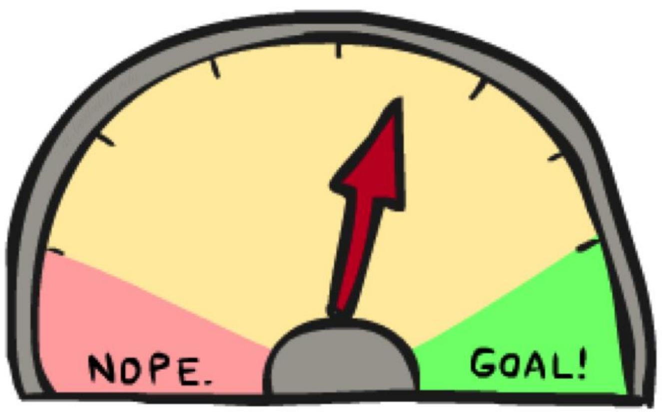
- All these search algorithms are the same except for fringe strategies
  - Conceptually, all fringes are **priority queues** (i.e. collections of nodes with attached priorities)
  - Practically, for DFS and BFS, you can avoid the log(n) overhead from an actual priority queue, by using stacks and queues
  - Can even code one implementation that takes a variable queuing object



Up next: Informed Search

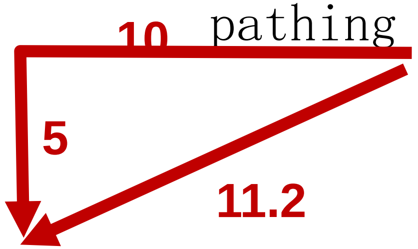
- Uninformed Search
  - DFS
  - BFS
  - UCS

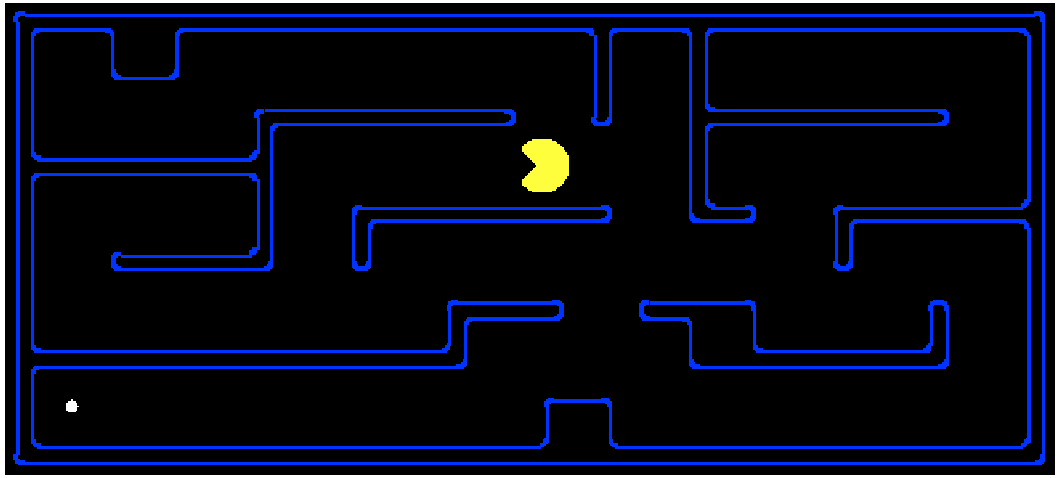
- Informed Search
  - Heuristics
  - Greedy Search
  - A* Search
  - Graph Search

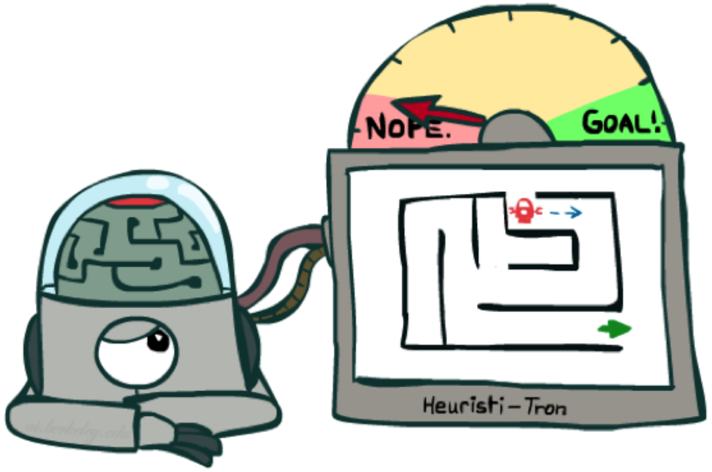


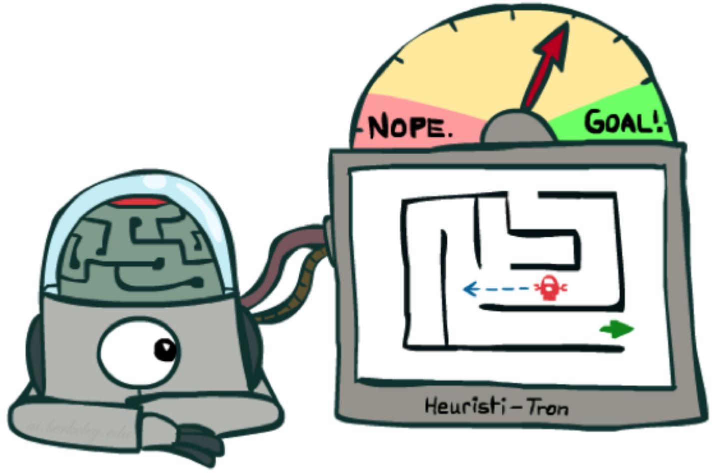
Search Heuristics

- A heuristic is:
  - A **function** that ***estimates*** **how close** a state is to a goal(If the current state is far from the goal, them the heuristic is large.)
  - Designed for a particular search problem
  - Pathing?
  - Examples: Manhattan distance, Euclidean distance for pathing

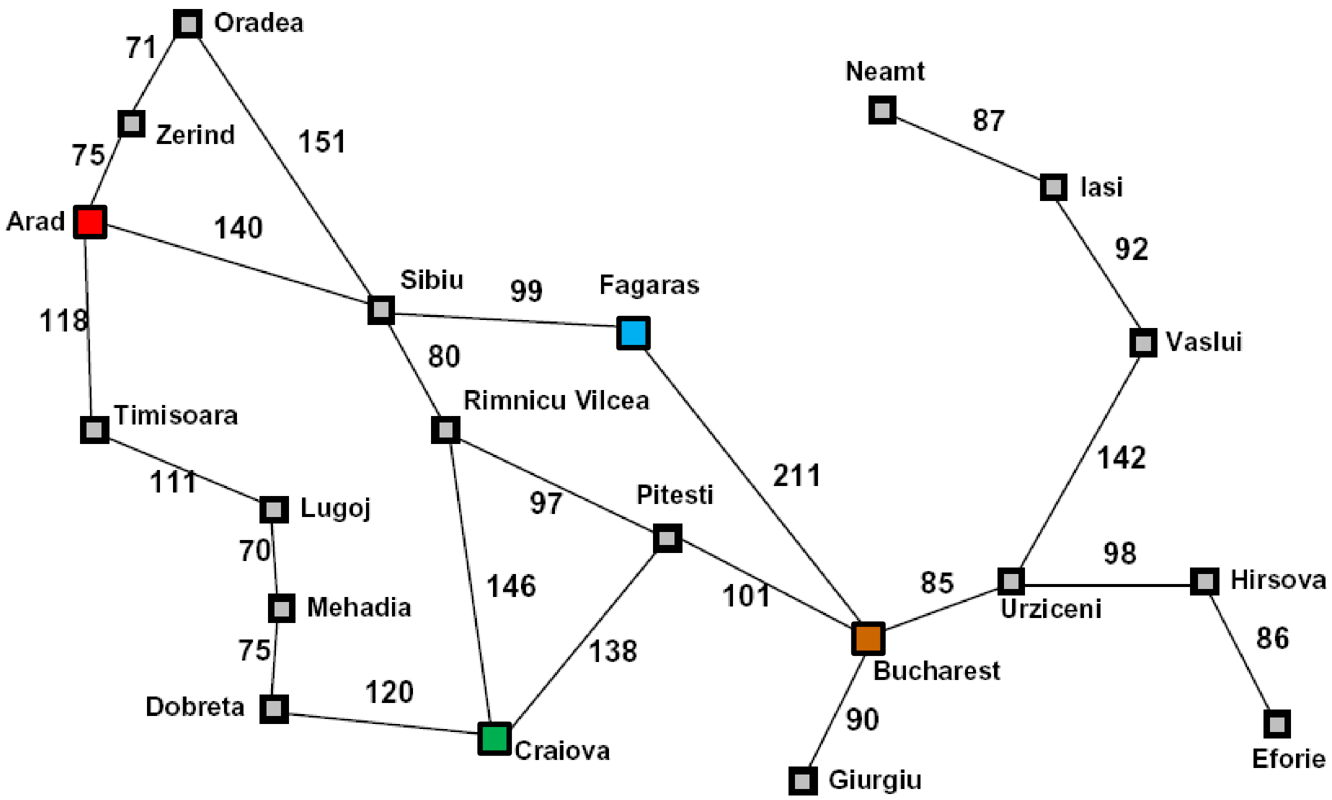








Example: Heuristic Function



Straight−line distance to Bucharest

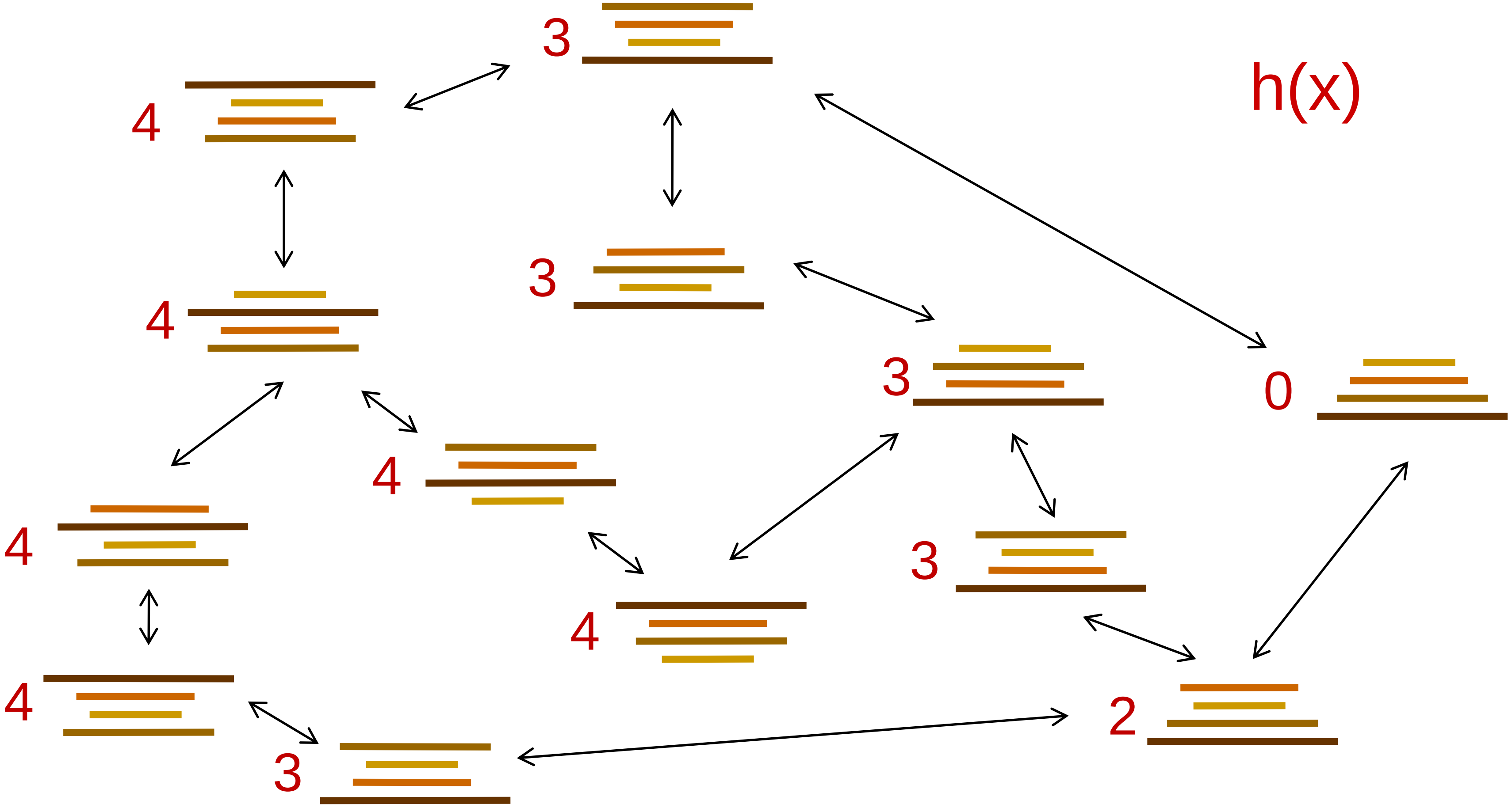
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

Example: Heuristic Function

Heuristic?

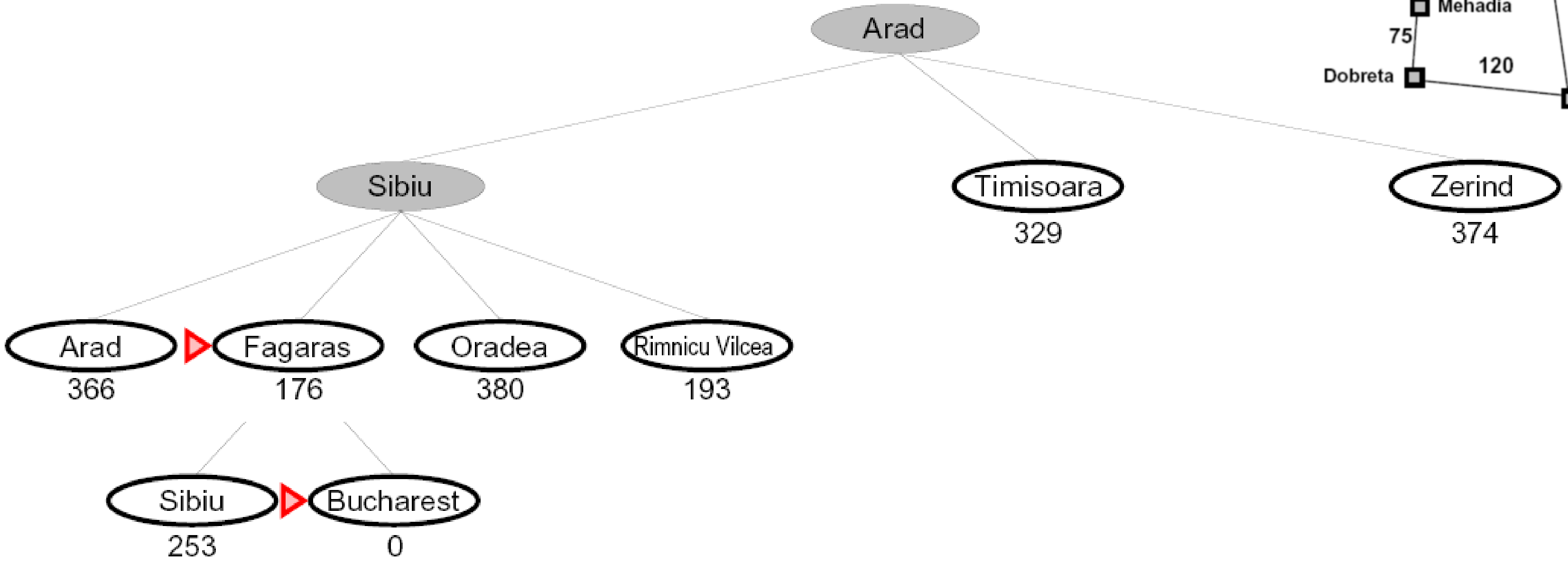
E.g. the number of the largest pancake that is still out of place

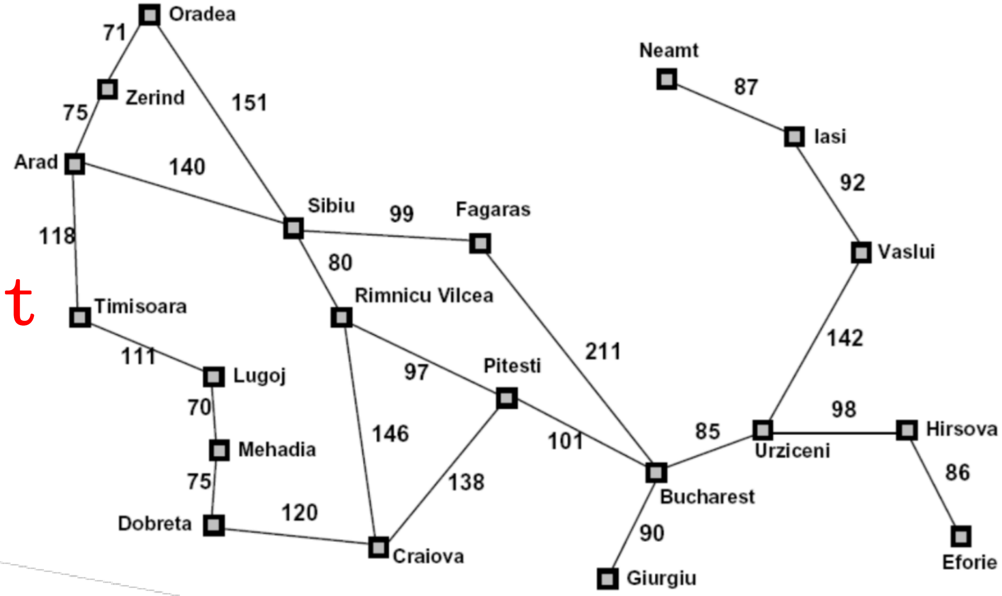


Greedy Search

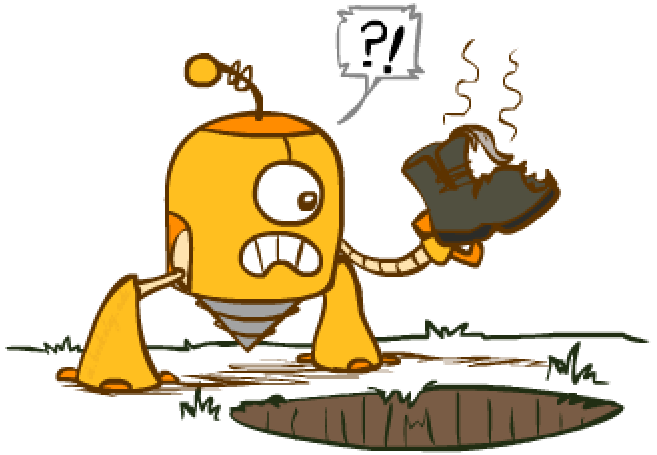
Greedy Search

- Expand the node that seems closest
- (The heuristic is the smallest)



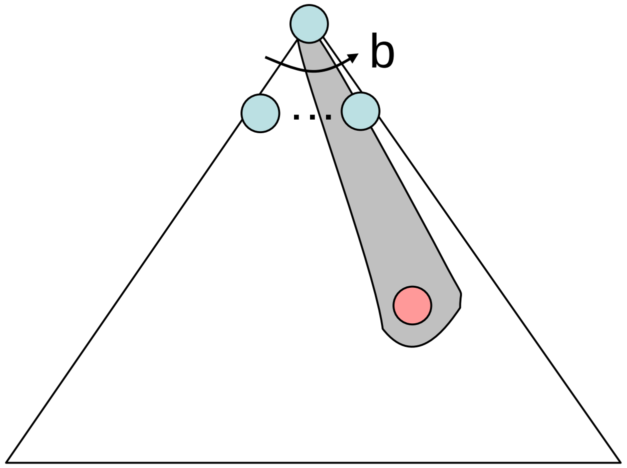


- Is it optimal?
  - No. Resulting path to Bucharest is not the shortest!

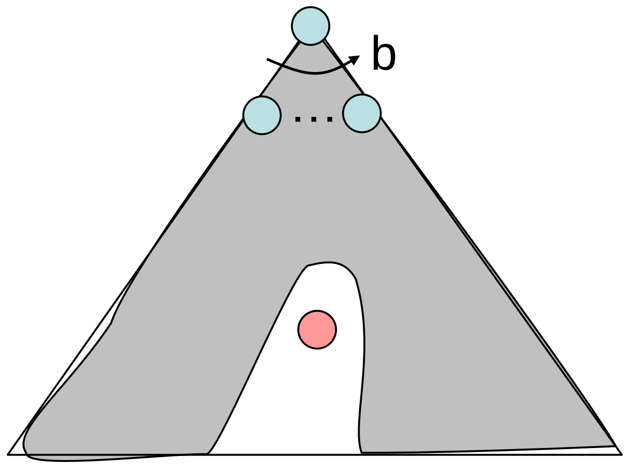


Greedy Search

- Strategy: expand a node that you think is closest to a goal state
  - Heuristic: estimate of distance to nearest goal for each state(Note that here the h(s) value is just the estimate of the distance to the nearest goal, not exactly the value. And here we don’t **constrain the heuristic** to be admissible or consistent, so that the value could be **larger** than d or **smaller** than d)



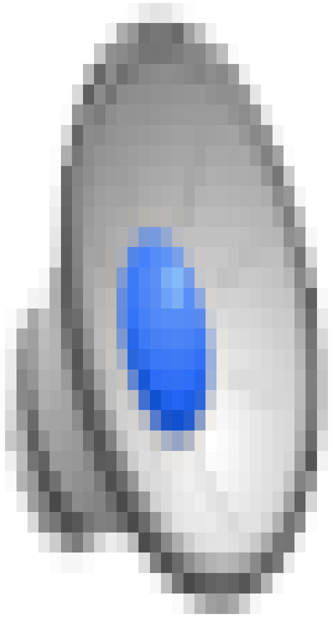
- A common case:
  - **Best-first takes you straight to the (wrong) goal**



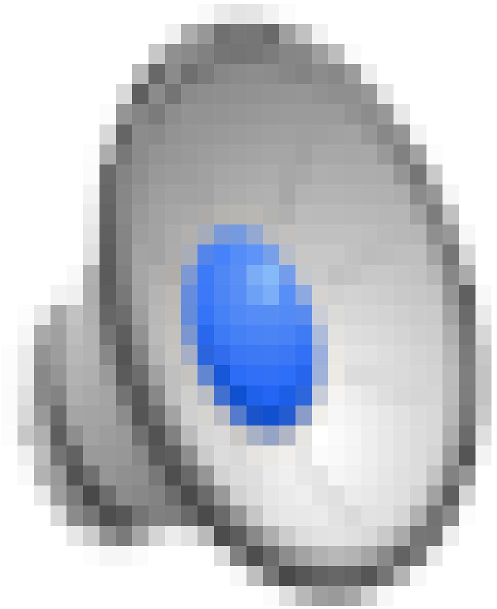
- Worst-case: like a badly-guided DFS

[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)

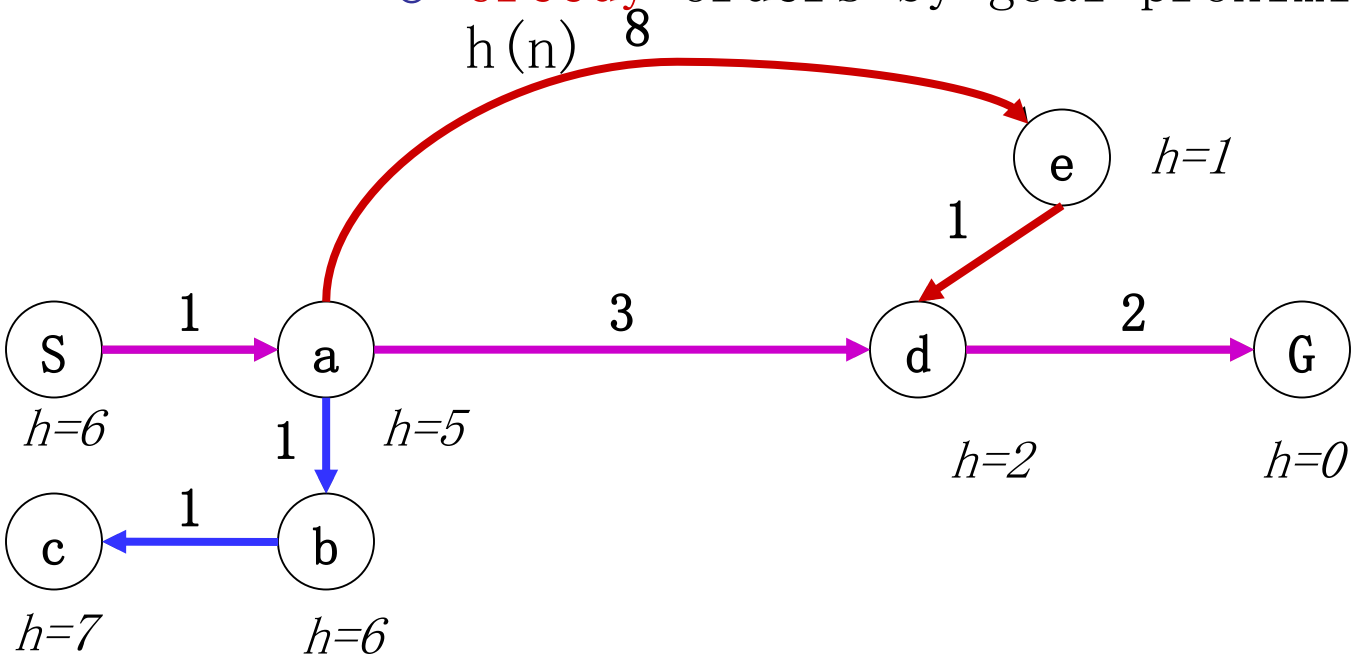


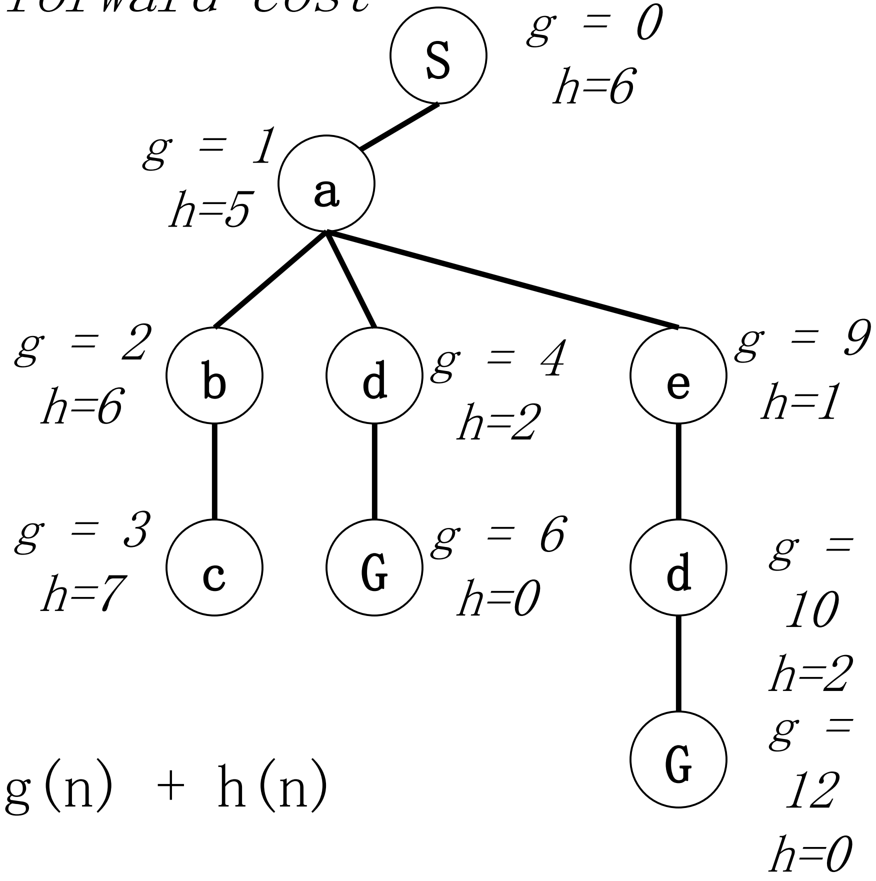
A* Search

A* Search

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)



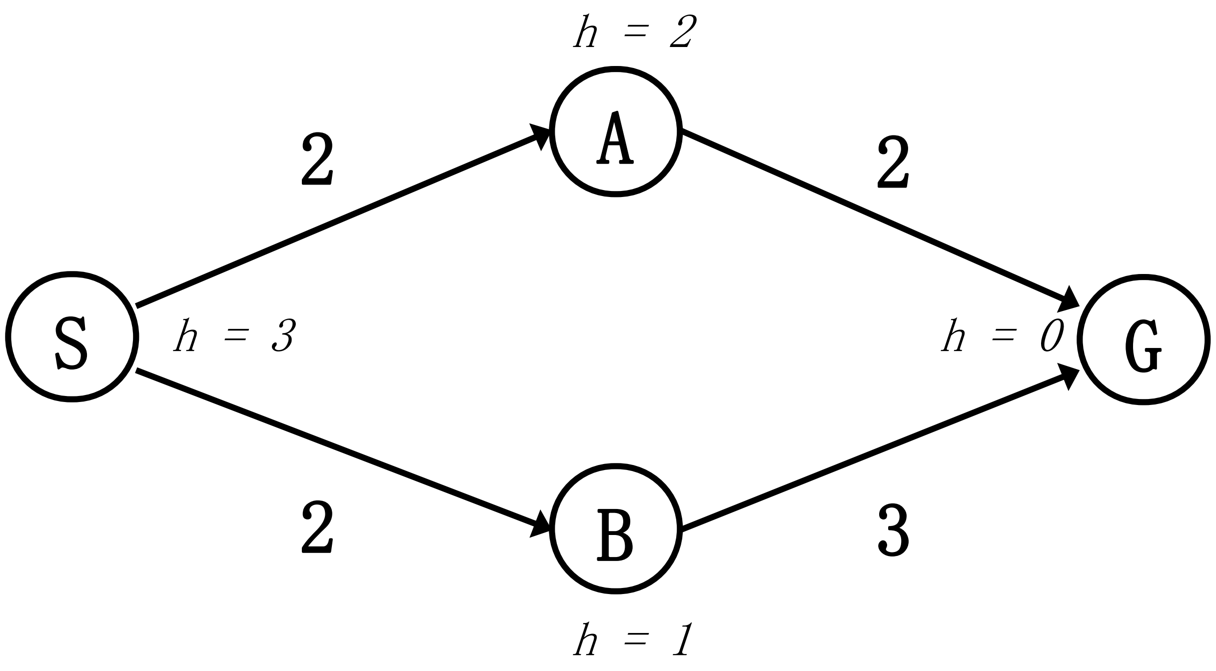


- A* Search orders by the sum: f(n) = g(n) + h(n)

Example: Teg

When should A* terminate?

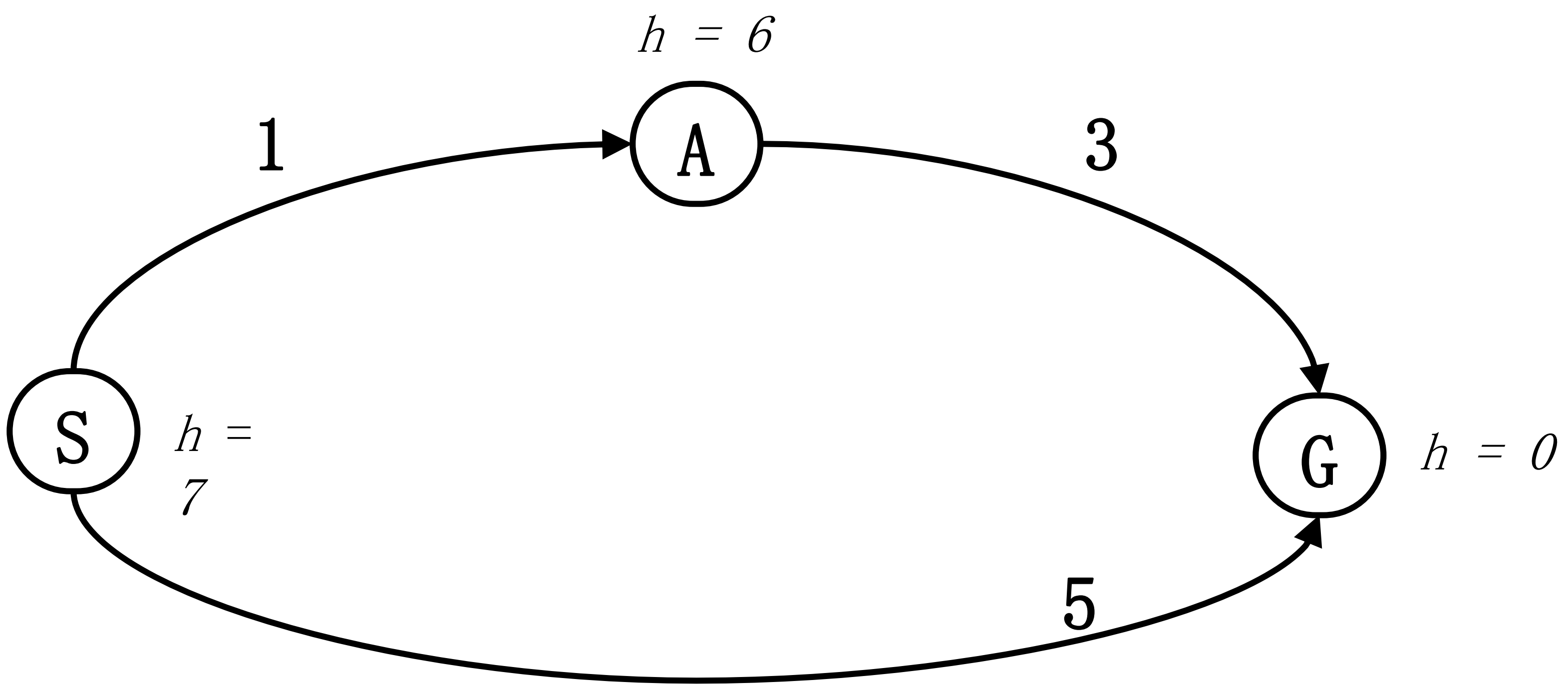
- Should we stop when we enqueue a goal?



	g	h	+
~~S~~	~~0~~	~~3~~	~~3~~
~~S->A~~	~~2~~	~~2~~	~~4~~
~~S->B~~	~~2~~	~~1~~	~~3~~
S->B->G	5	0	5
$\boxed{\text{S->A->G 4 0 4}}$			

- No: only stop when we dequeue a goal
- When we enqueue a goal, we don’t stop

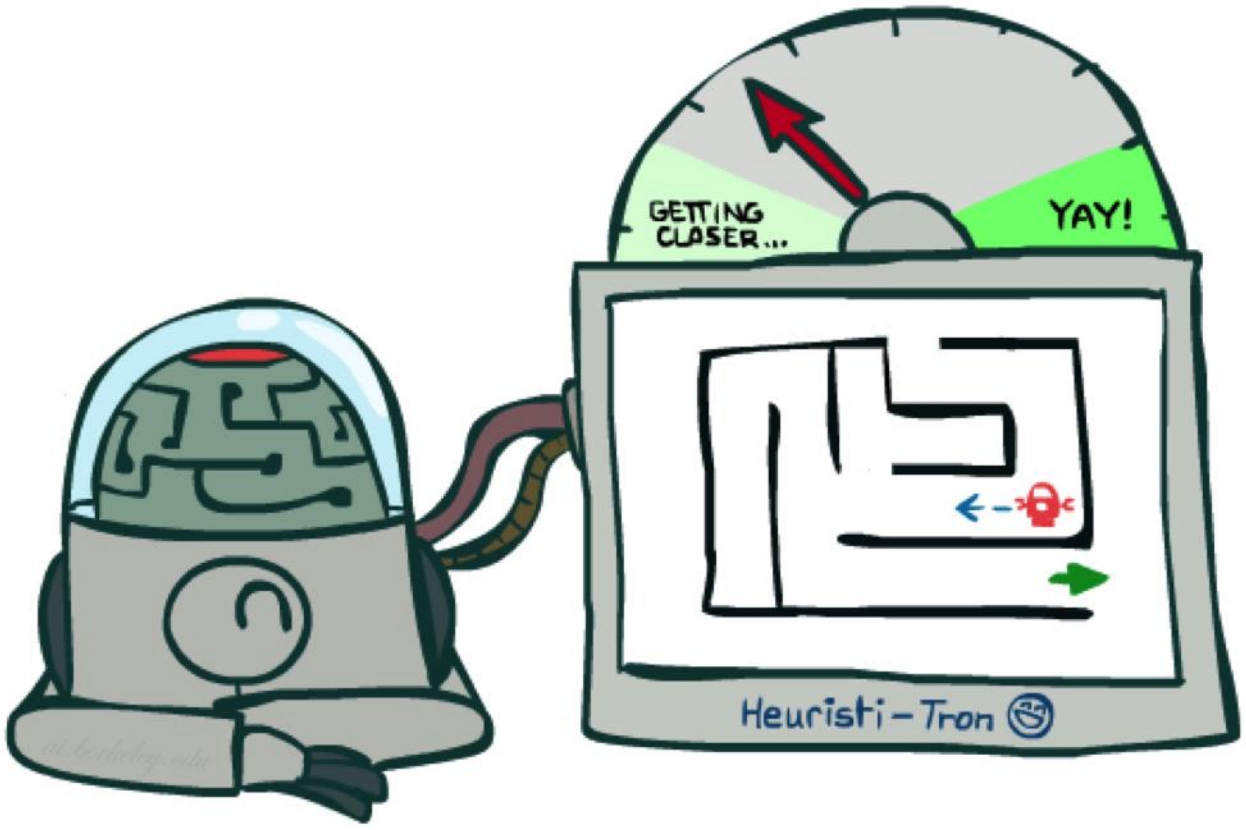
Is A* Optimal?



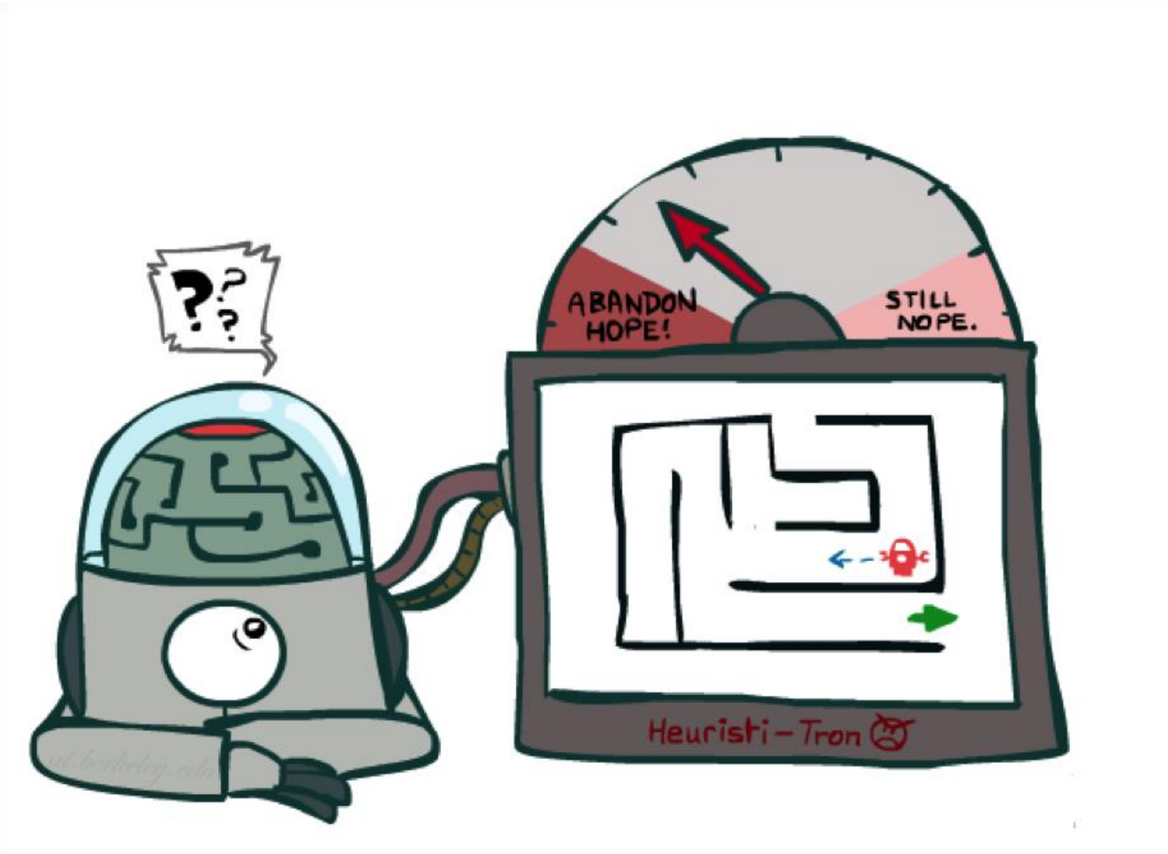
	g	h	+
~~S~~	~~0~~	~~7~~	~~7~~
S->A	1	6	7
S->G	5	0	5

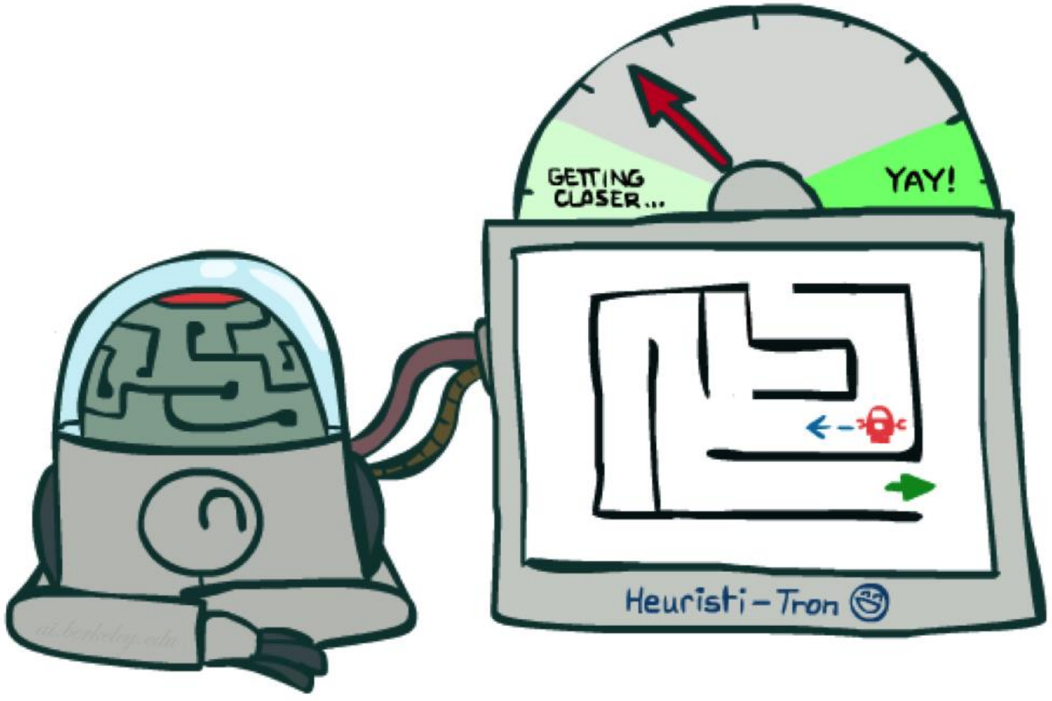
- What went wrong?
- **Actual bad goal cost(实际距离)** < estimated good goal cost-> 例如， A-G的实际距离为3，但是我们往大了预估，预估为6.
- **We need estimates to be less than actual costs! That is to say, h(s) value is small than the actual distance between current point and the goal test.**

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(永远不大于实际距离)

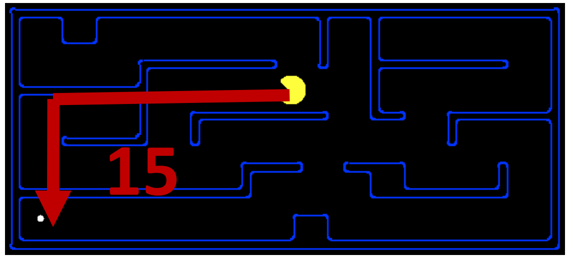
Admissible Heuristics

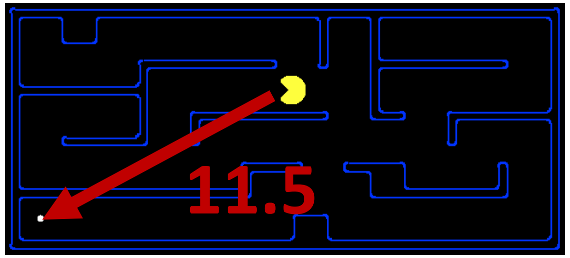
- A heuristic *h* is *admissible* (optimistic) if:

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

where $h^*(n)$ is the true cost to a nearest goal(Note that here we can set the heuristic to 0, which makes the A* algorithm back to the UCS, but we never make the h(n)-> infinity, because the greedy algorithm is not optimal.)

- Examples:

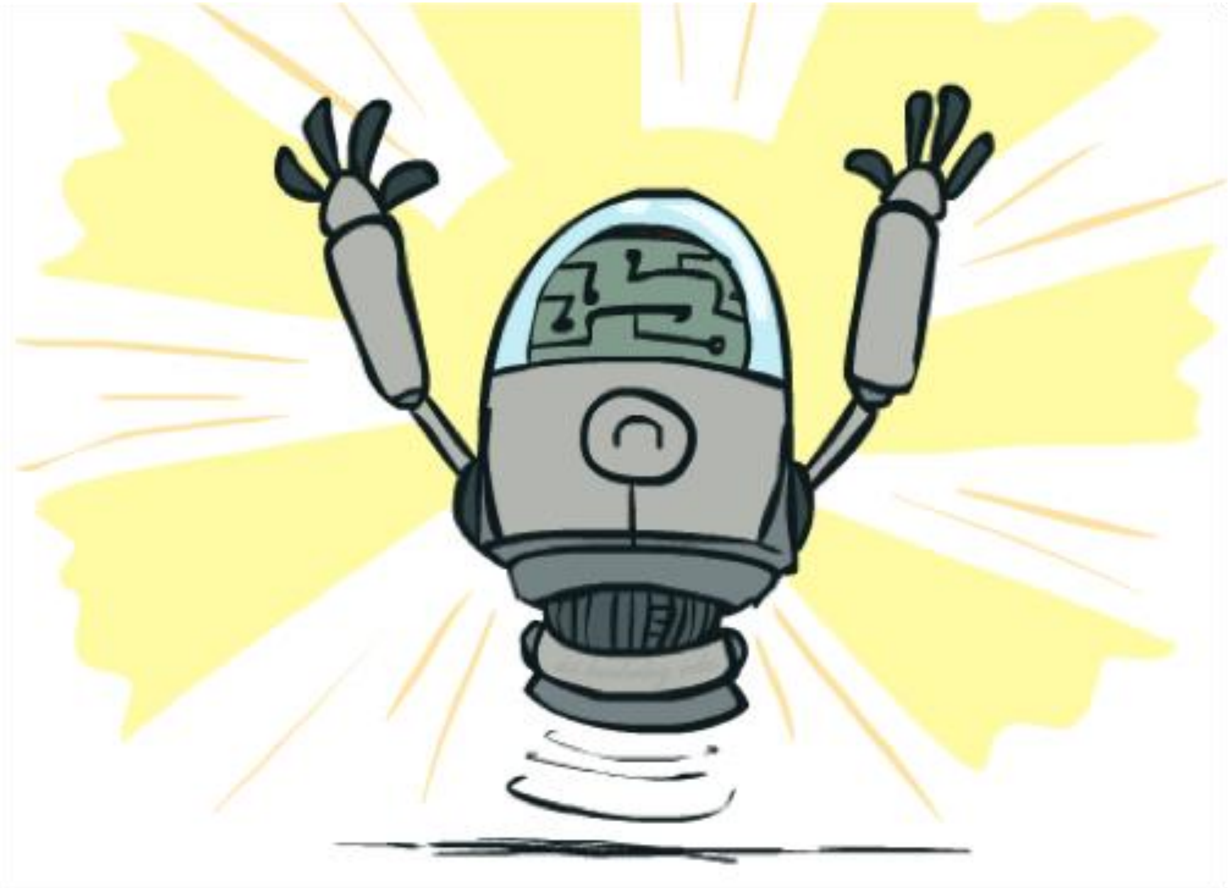




0.0

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

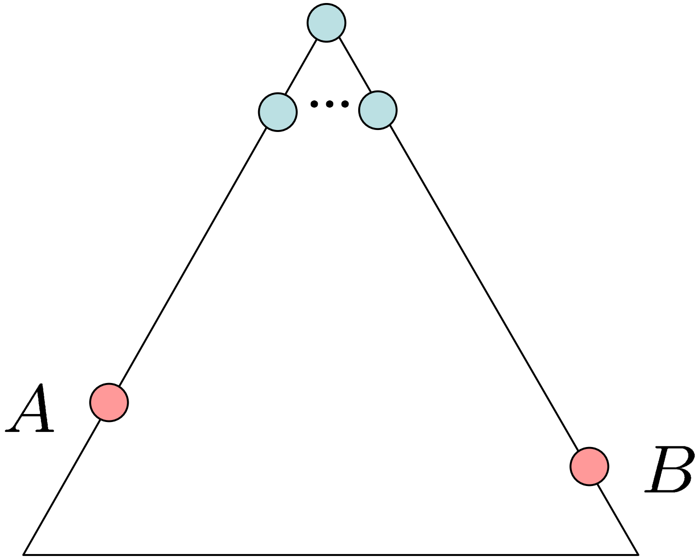
Optimality of A* Tree Search



Optimality of A* Tree Search

Assume:

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



Claim:

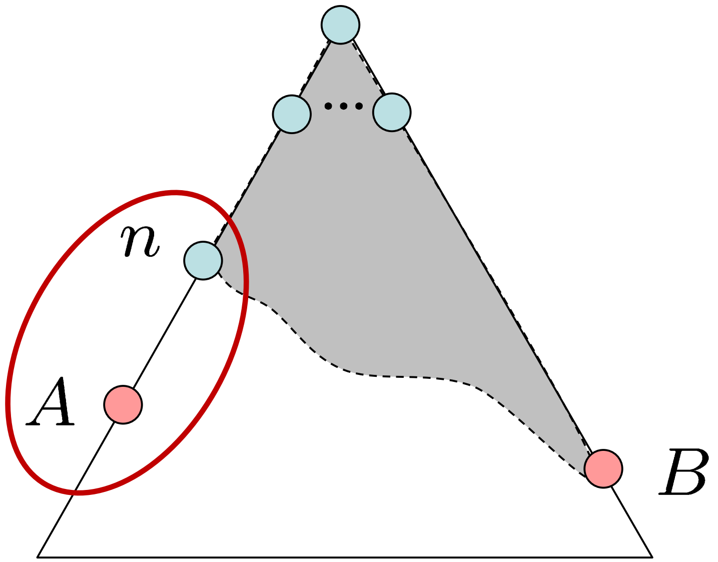
- **A will exit the fringe before B**

Admissible Heuristics implies optimality of A* tree search. (Not ensure for graph search, we will talk about it soon)

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)



Admissibility of h,
h(n)<= g(n -> A
),g(A)=g(n)+g(n->A)

$$f(n) = g(n) + h(n)$$ ->Definition of f-cost

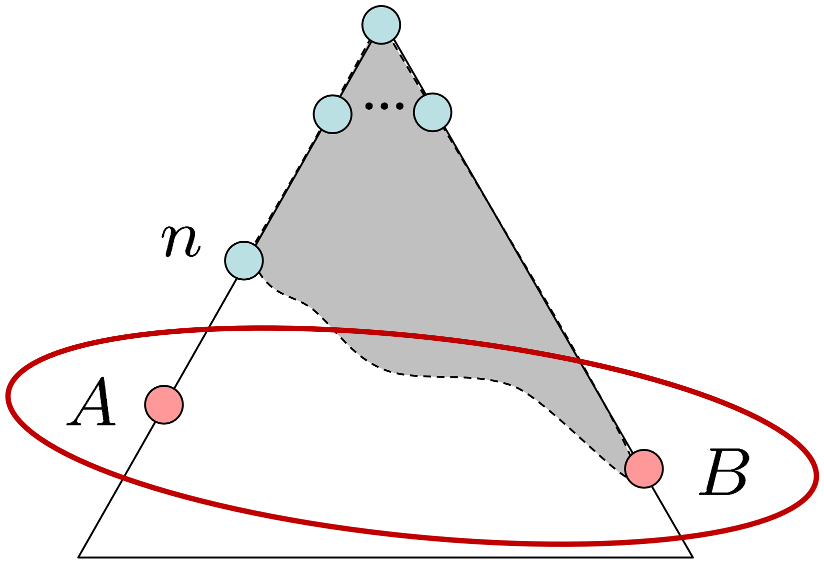
$$f(n) \leq g(A)$$

$$g(A) = f(A)$$ -> 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)



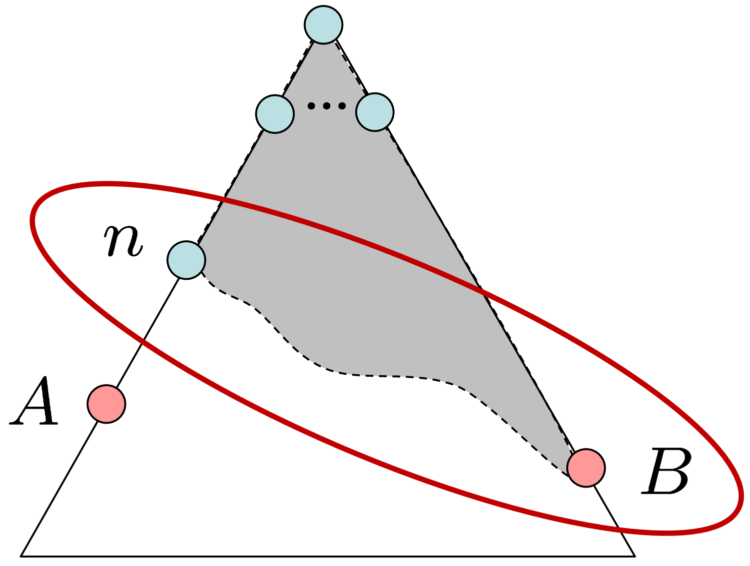
$g(A) < g(B)$ B is suboptimal

$f(A) < f(B)$ h = 0 at a goal

Optimality of A* Tree Search: Blocking

Proof:

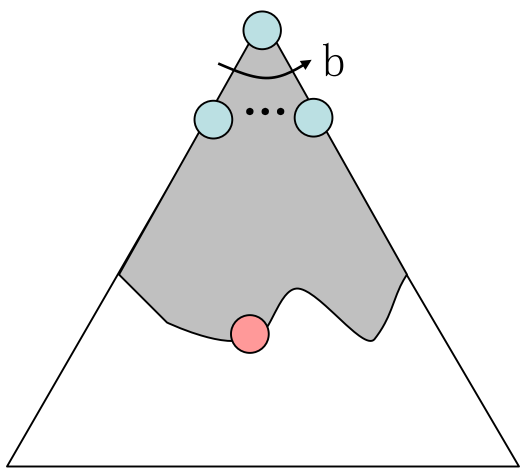
- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will be expanded before B
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)
  3. n expands before B
- All ancestors of A expand before B
- A expands before B
- A* search is optimal



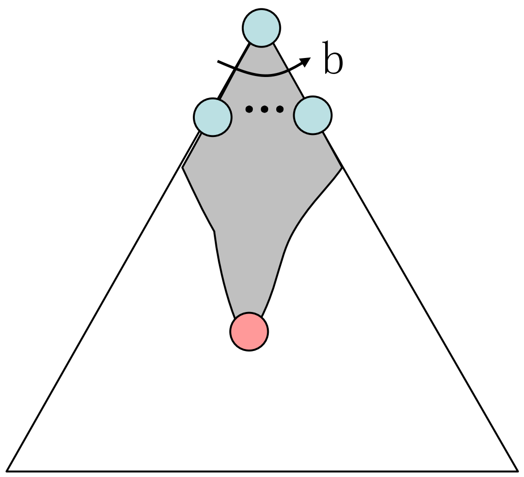
$$f(n) \leq f(A) < f(B)$$

Properties of A*

Uniform-Cost

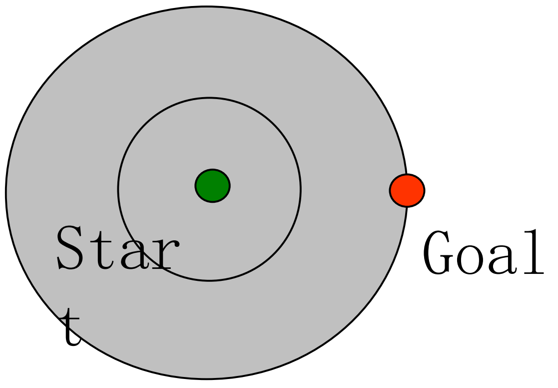


A*

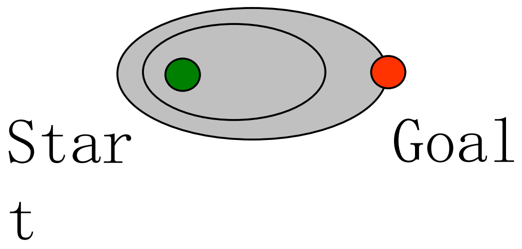


UCS vs A* Contours

- Uniform-cost expands **equally** in all "directions"

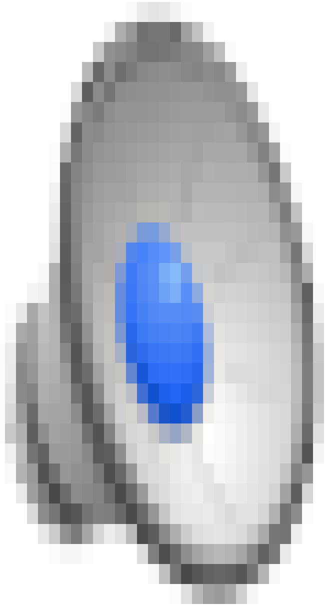


- A* expands **mainly** toward the goal, but does hedge its bets to ensure optimality

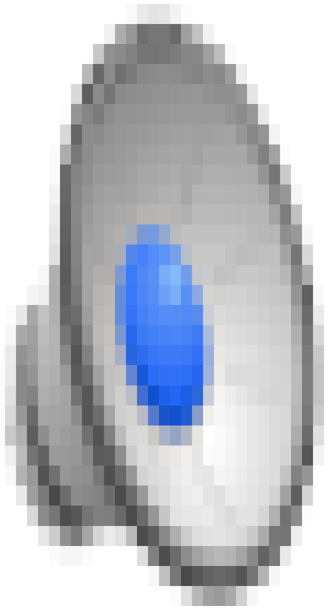


[Demo: contours UCS / greedy / A* empty (L3D1)]
[Demo: contours A* pacman small maze (L3D5)]

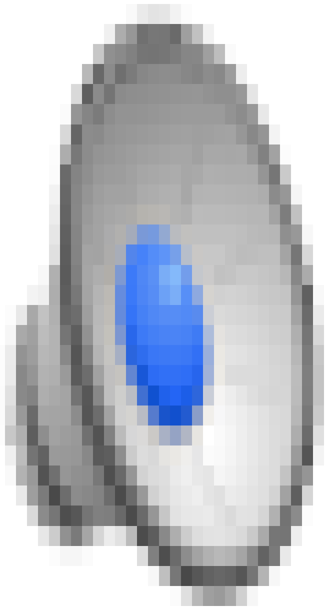
Video of Demo Contours (Empty) -- UCS



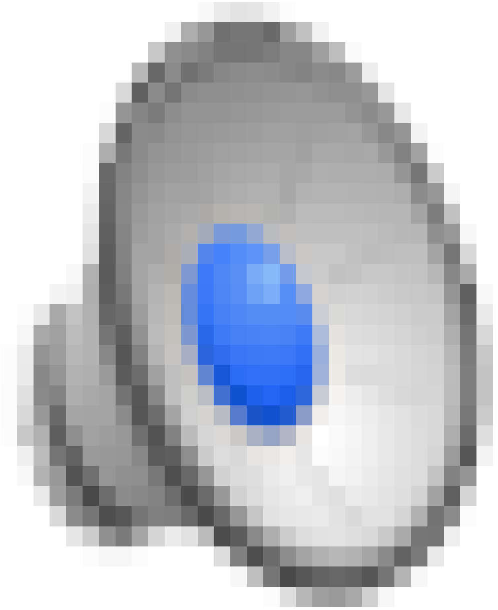
Video of Demo Contours (Empty) -- Greedy



Video of Demo Contours (Empty) – A*



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



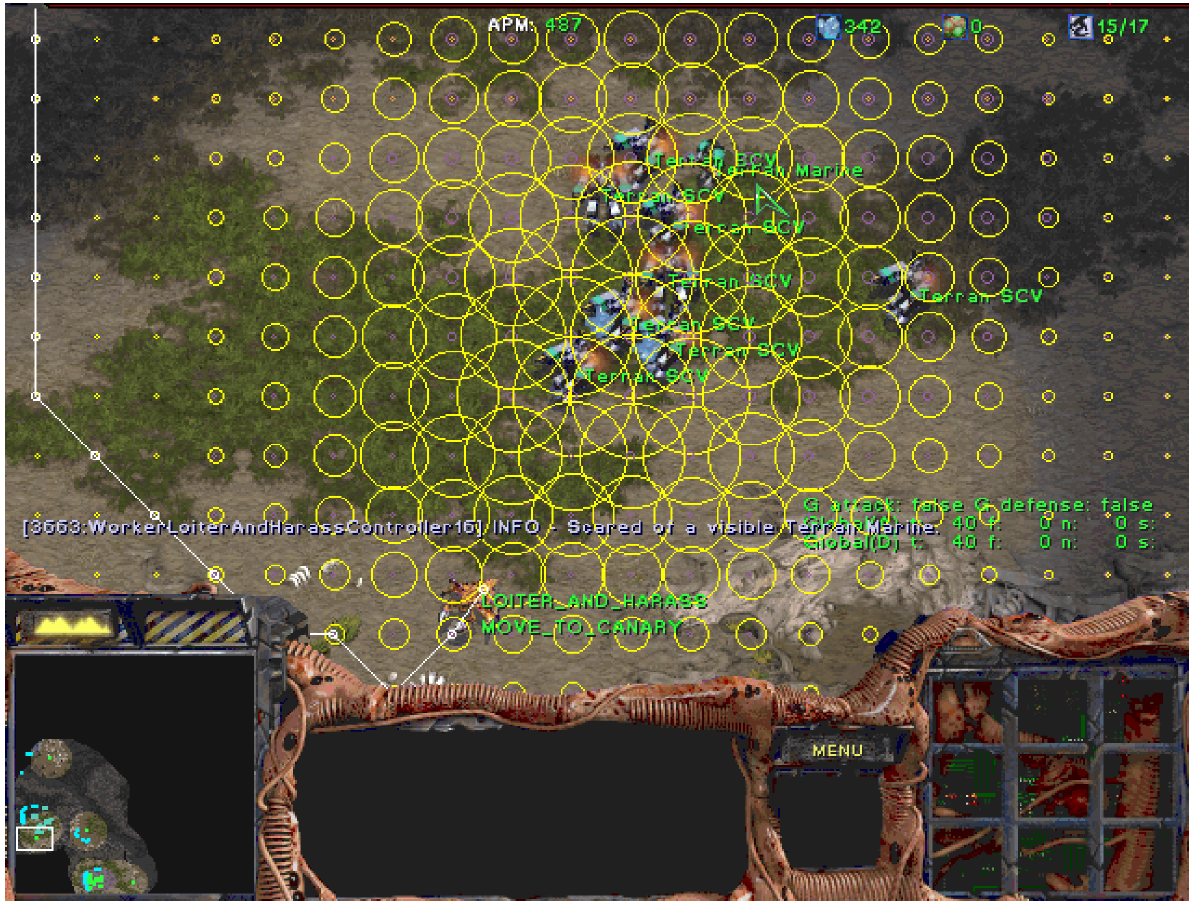
Comparison

Greedy

Uniform Cost

A*

A* Applications

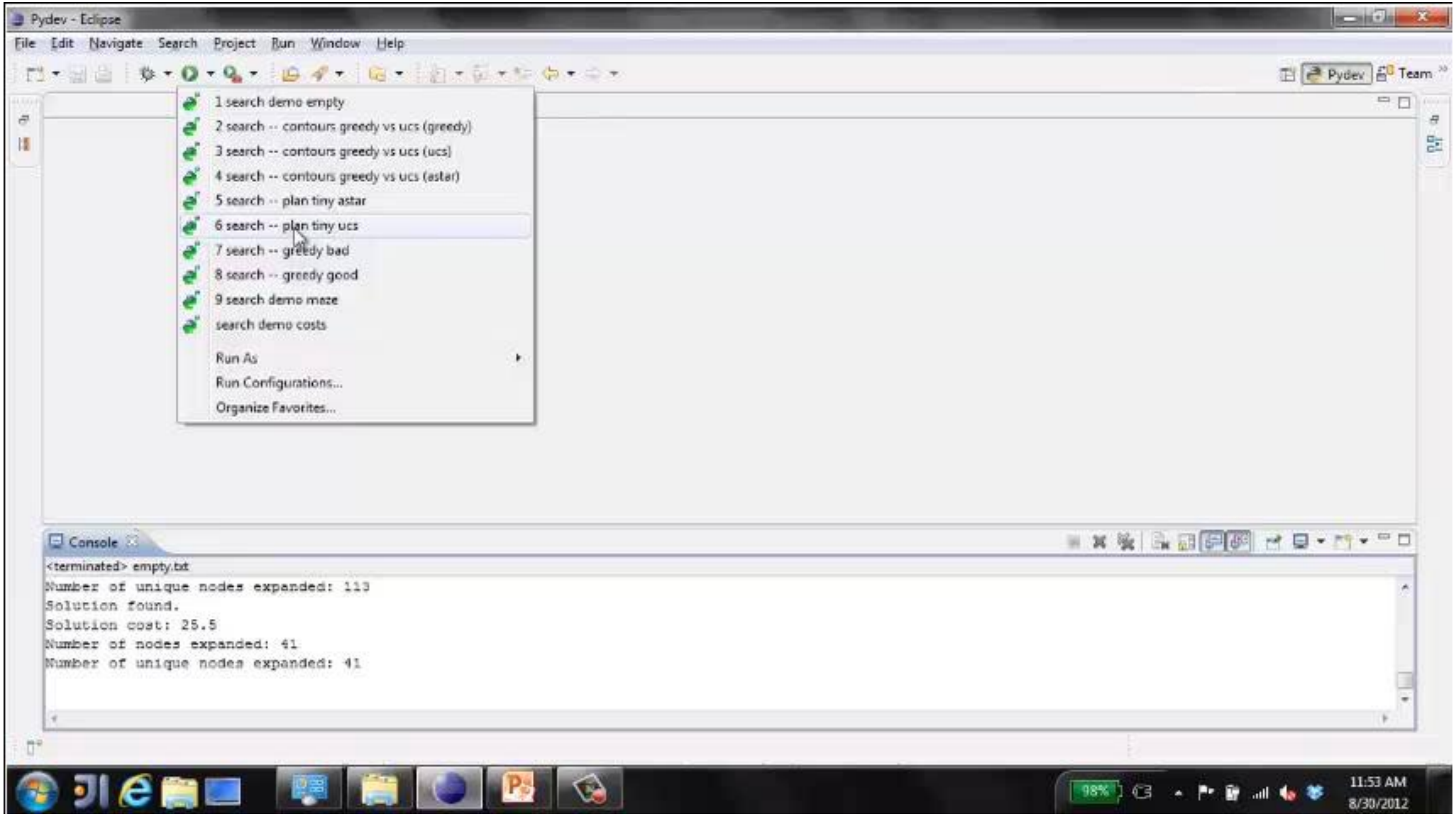


A* Applications

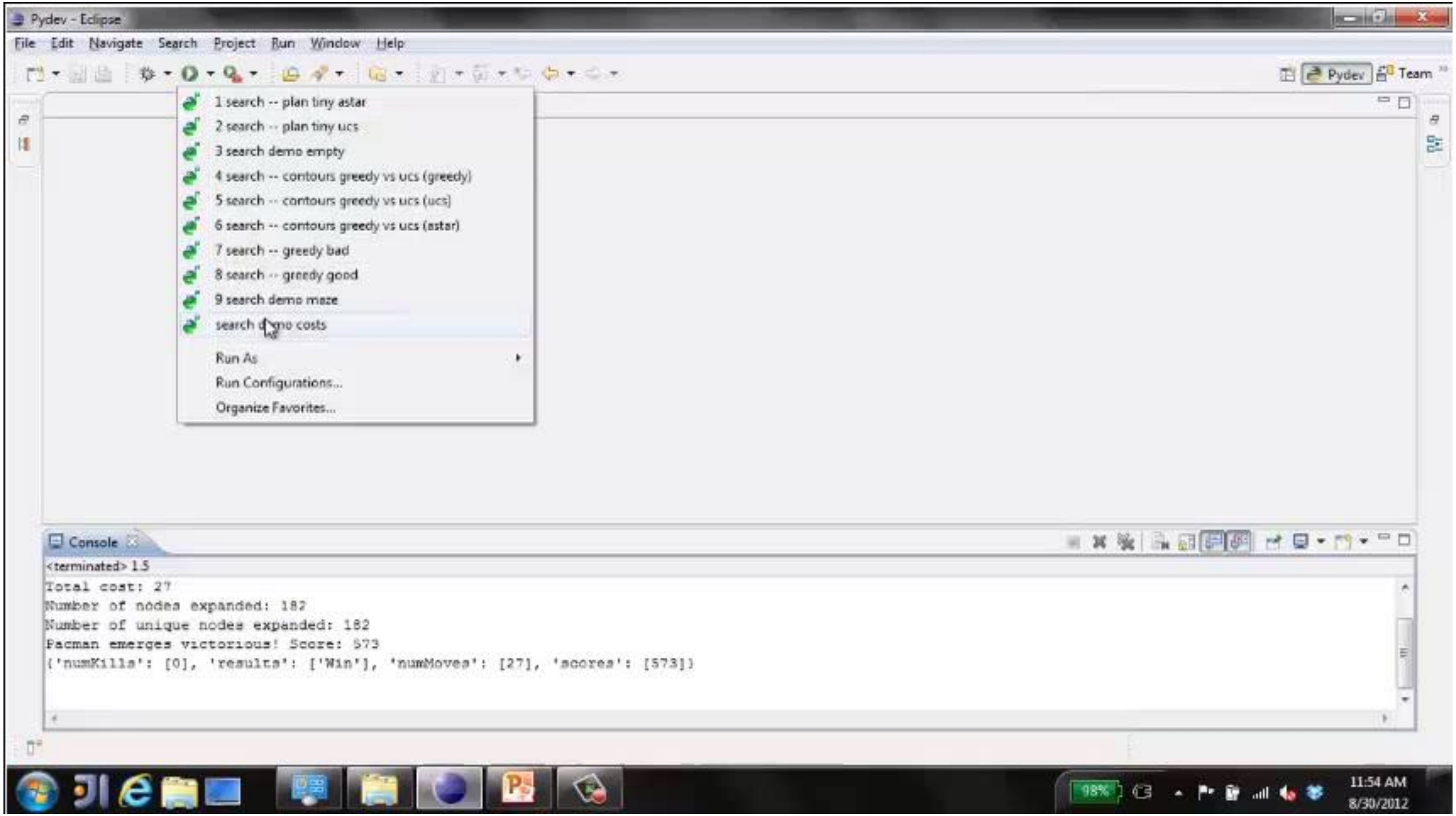
- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- …

[Demo: UCS / A* pacman tiny maze (L3D6,L3D7)]
[Demo: guess algorithm Empty Shallow/Deep (L3D8)]

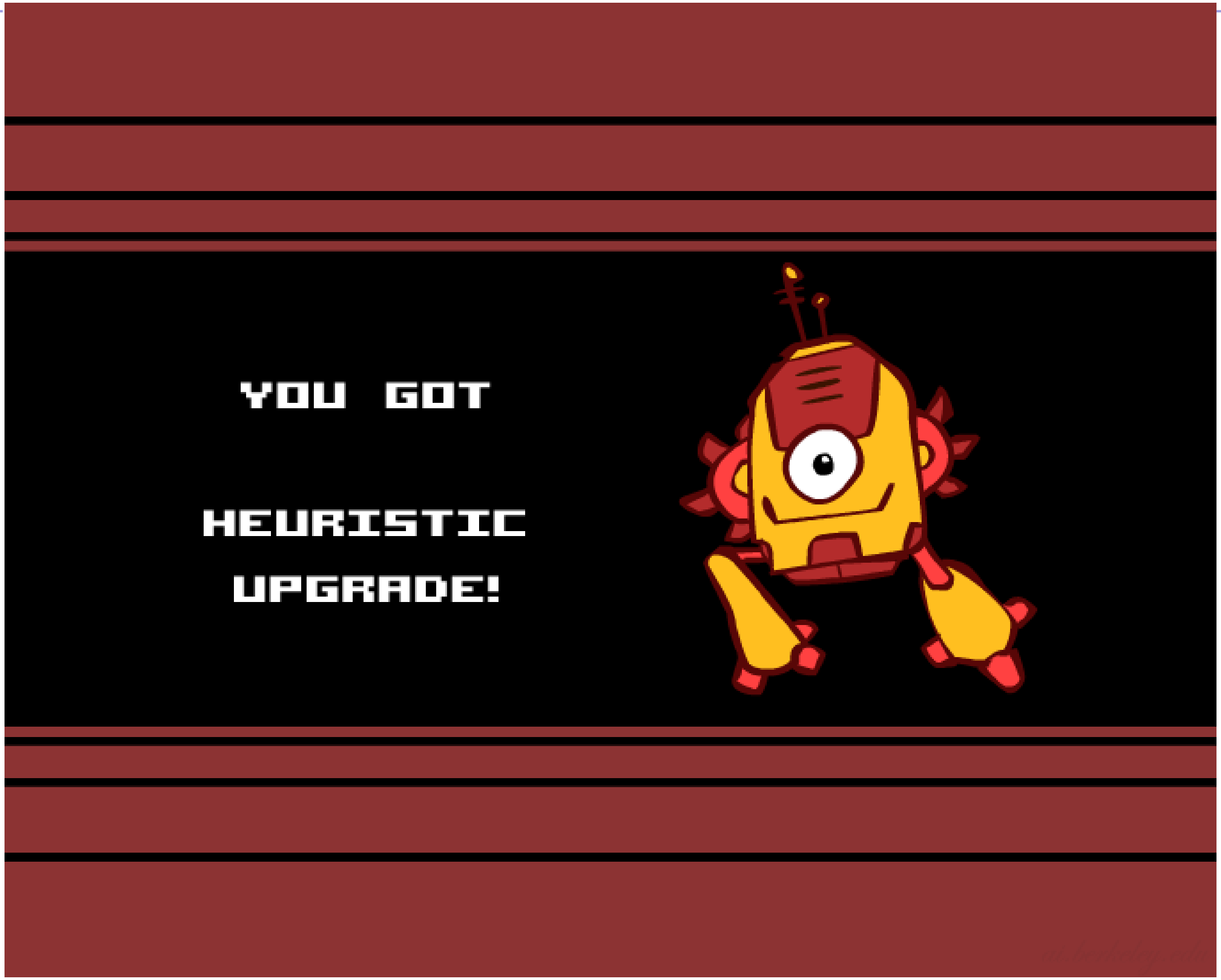
Video of Demo Pacman (Tiny Maze) – UCS / A*



Video of Demo Empty Water Shallow/Deep - Guess Algorithm

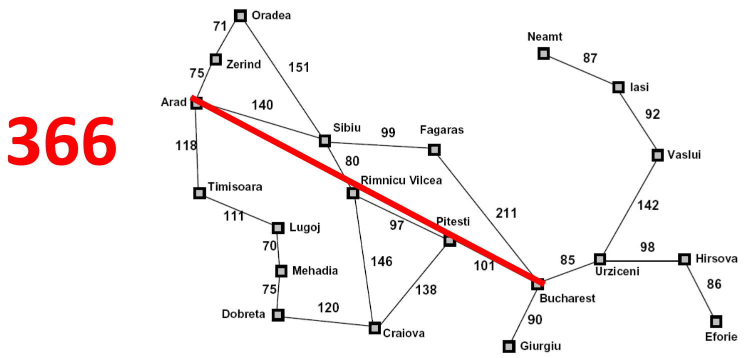


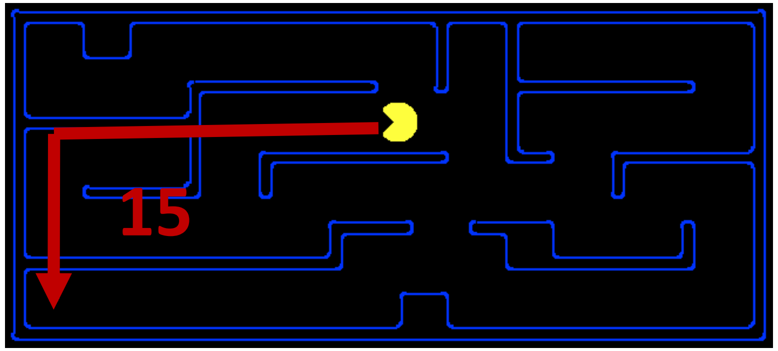
Creating Heuristics



Creating Admissible Heuristics

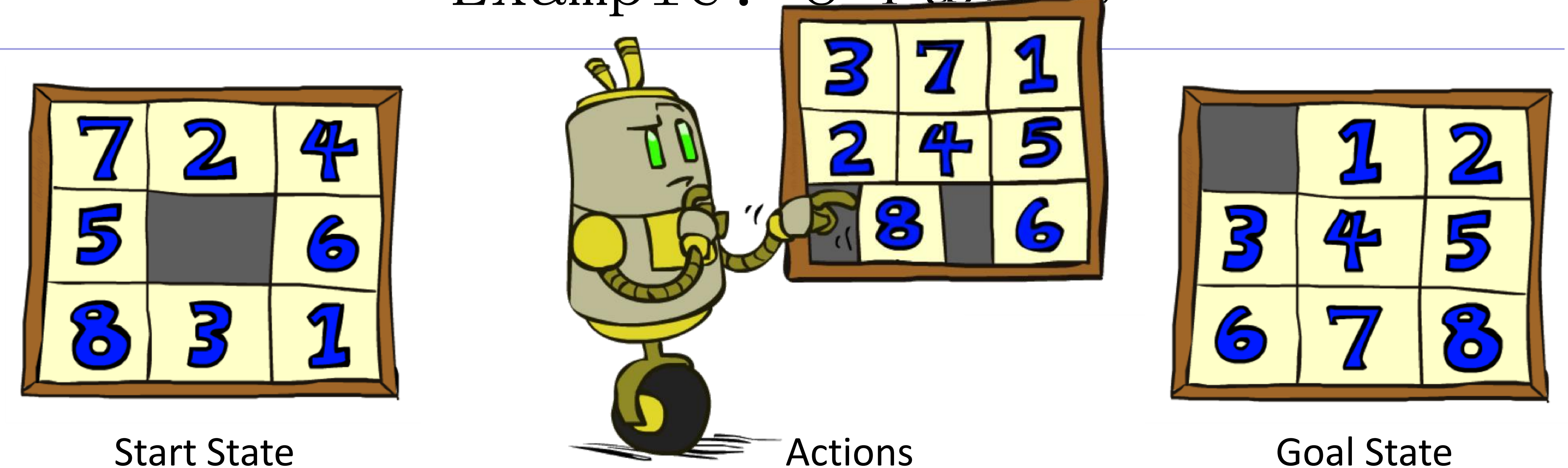
- Most of the work in solving hard search problems optimally is in **coming up with admissible heuristics**
- Often, admissible heuristics are solutions to *relaxed problems (we ignore all the obstacles, all the conditions, like the straight line we consider below)*, where new actions are available





- Inadmissible heuristics are often useful too! -> Perhaps the result path is not optimal, but is’ s not far from the optimal one!

Example: 8 Puzzle

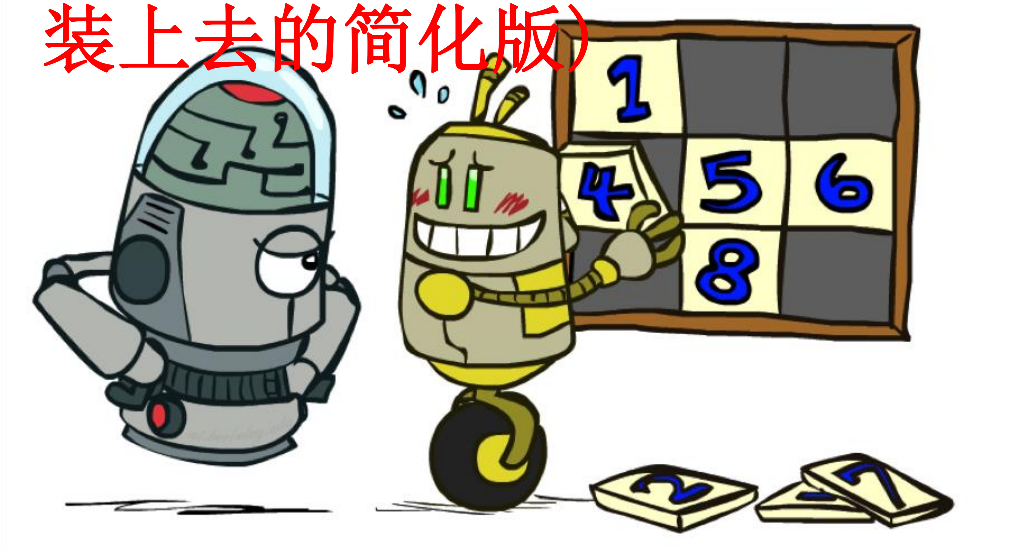


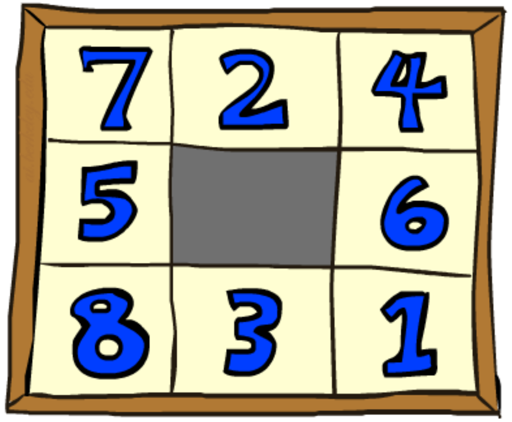
- What are the states? -> Grids with different numbers.
- How many states? -> 9!
- What are the actions? NESW.
- How many successors from the start state?
- 4
- What should the costs be?
- Move one step, cost 1.

Admissible heuristics ?

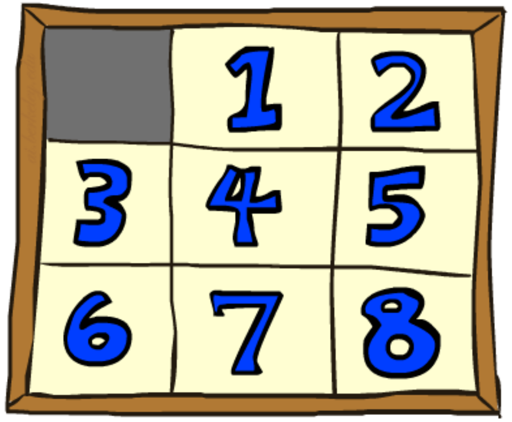
8 Puzzle I

- Heuristic: Number of tiles misplaced 8
- Why is it admissible?
- h(start) =
- This is a *relaxed-problem* heuristic(即将它转成可以拆下来再装上去的简化版)





Start State



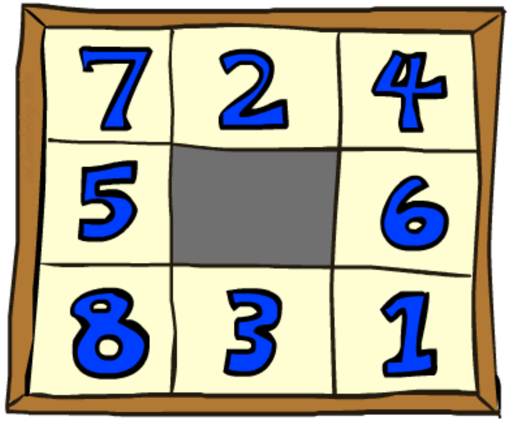
Goal State

	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

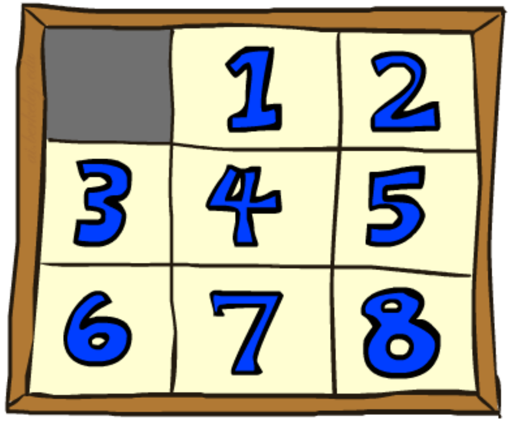
Statistics from Andrew Moore

8 Puzzle II

- What if we had an **easier 8-puzzle（此为另一个简化版本，简化程度没有上一个高，要求必须8个tiles同时在板）** where any tile could slide any direction at any time, ignoring other tiles?



Start State



Goal State

- Total *Manhattan* distance
- ->h (#misplace)<h(Manhattan)<h*
- Why is it admissible?

  ->At each time we only move one tile, we don’t change any other tiles.

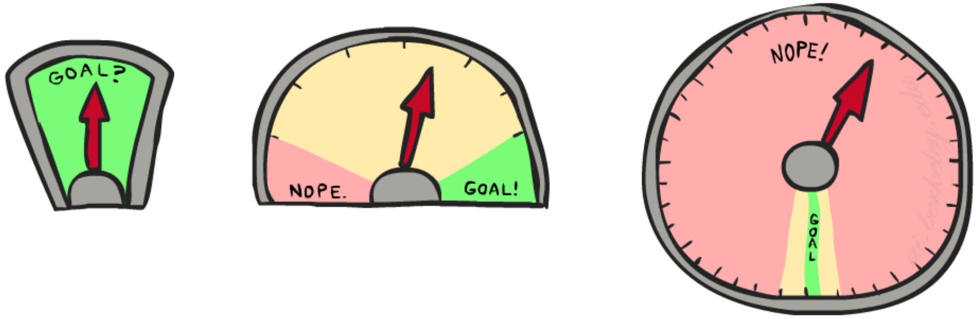
$3 + 1 + 2 + \ldots = 18$

- h(start) =

	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 (the quality of estimation is high) but usually do more work per node to compute the heuristic itself

Semi-Lattice of Heuristics

Trivial Heuristics, Dominance

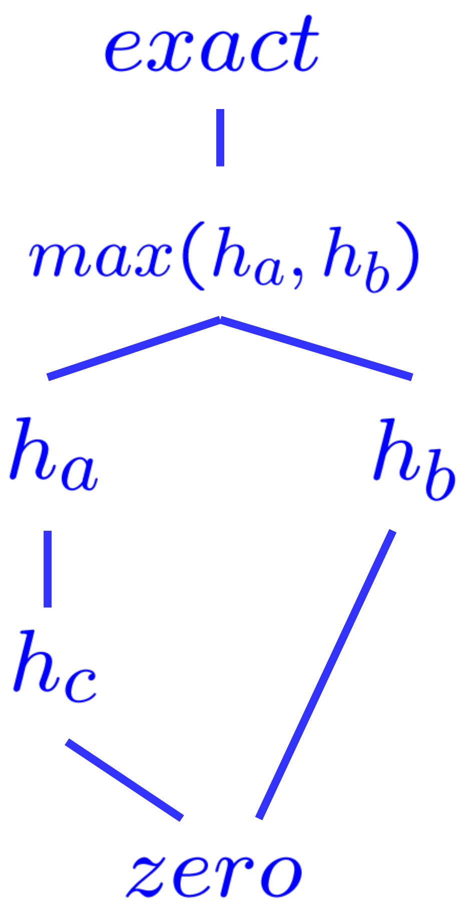
- Dominance: $h_a \geqslant h_c$ if

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

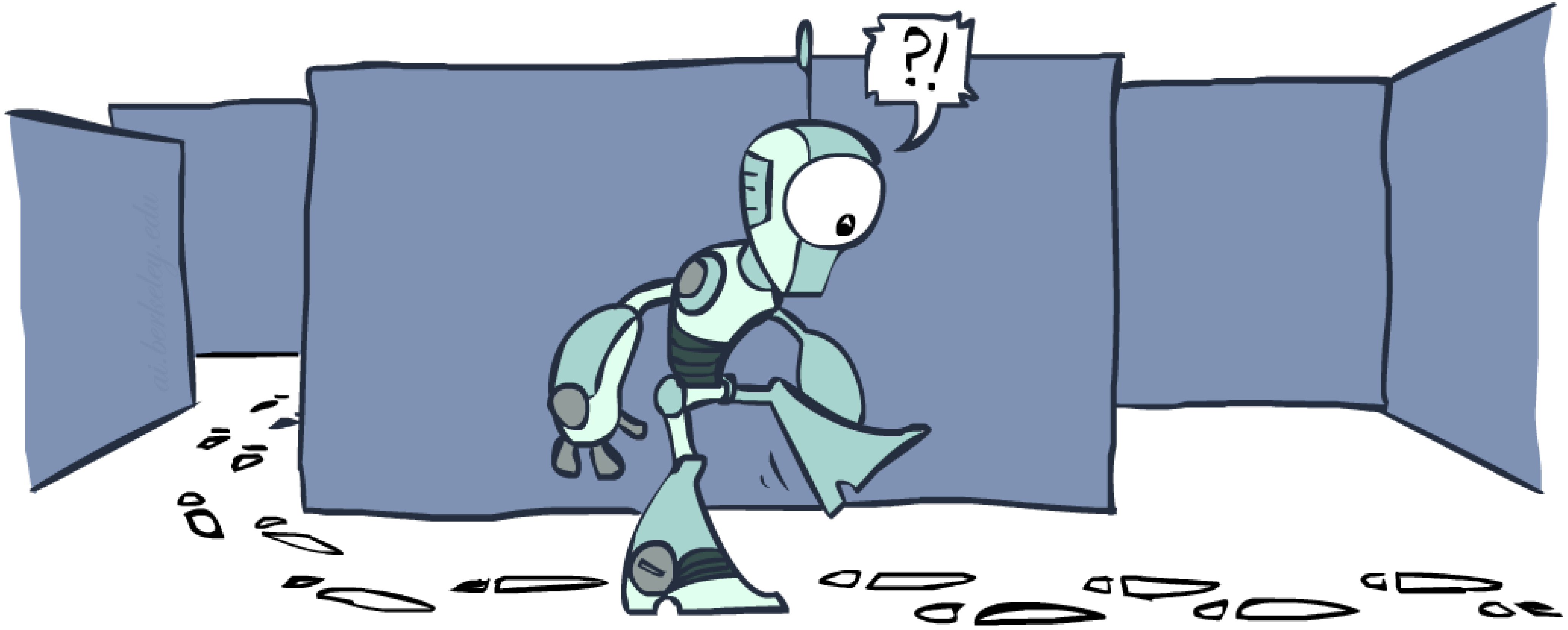
- Heuristics form a semi-lattice:
  - **Max of admissible heuristics** is admissible

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

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

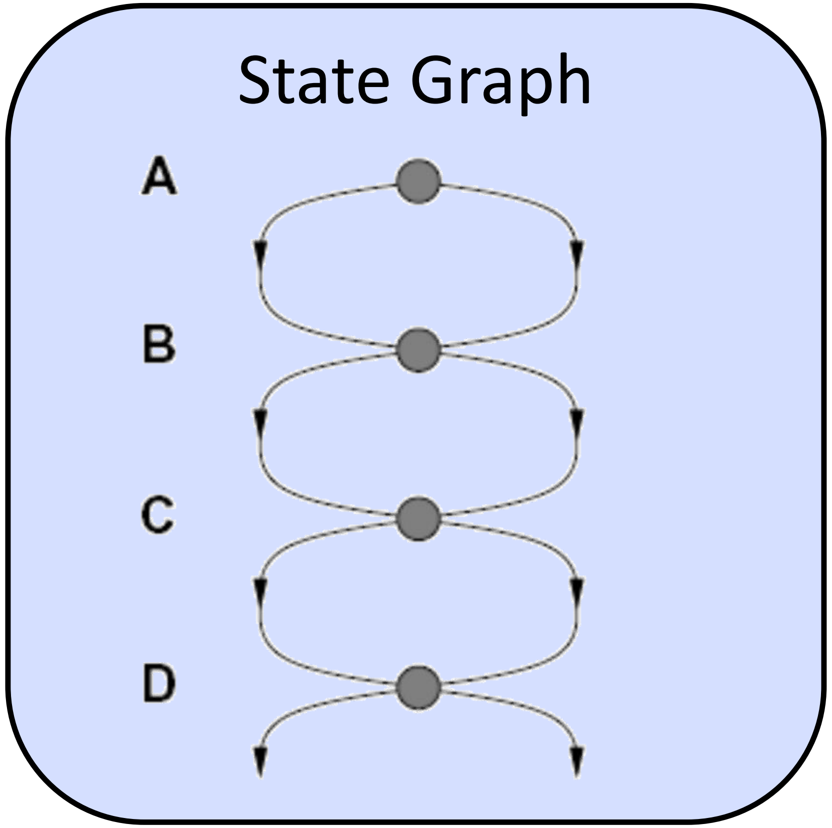


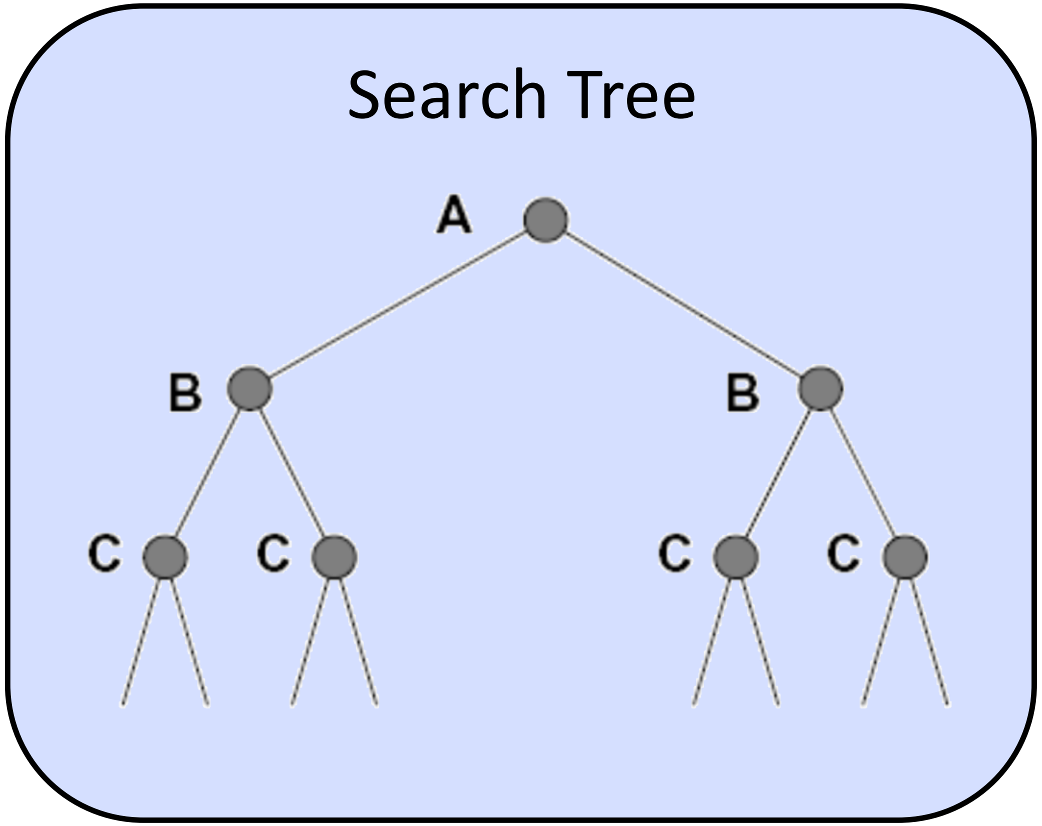
Graph Search



Tree Search: Extra Work!

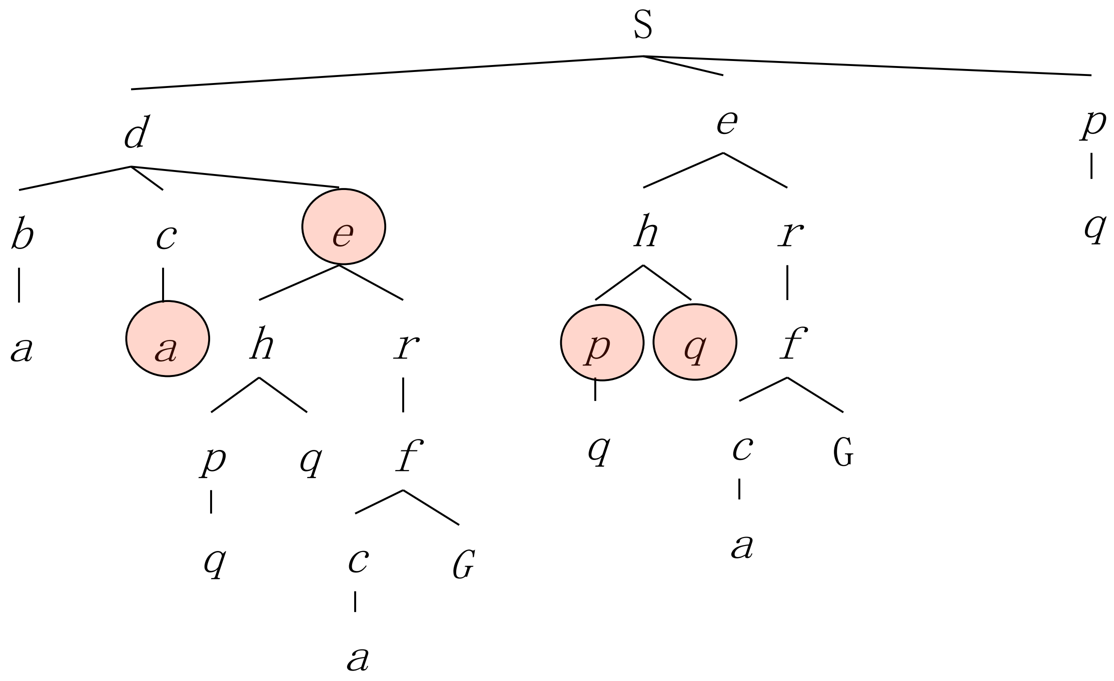
- Failure to **detect repeated states** can cause **exponentially more work**.





Graph Search

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

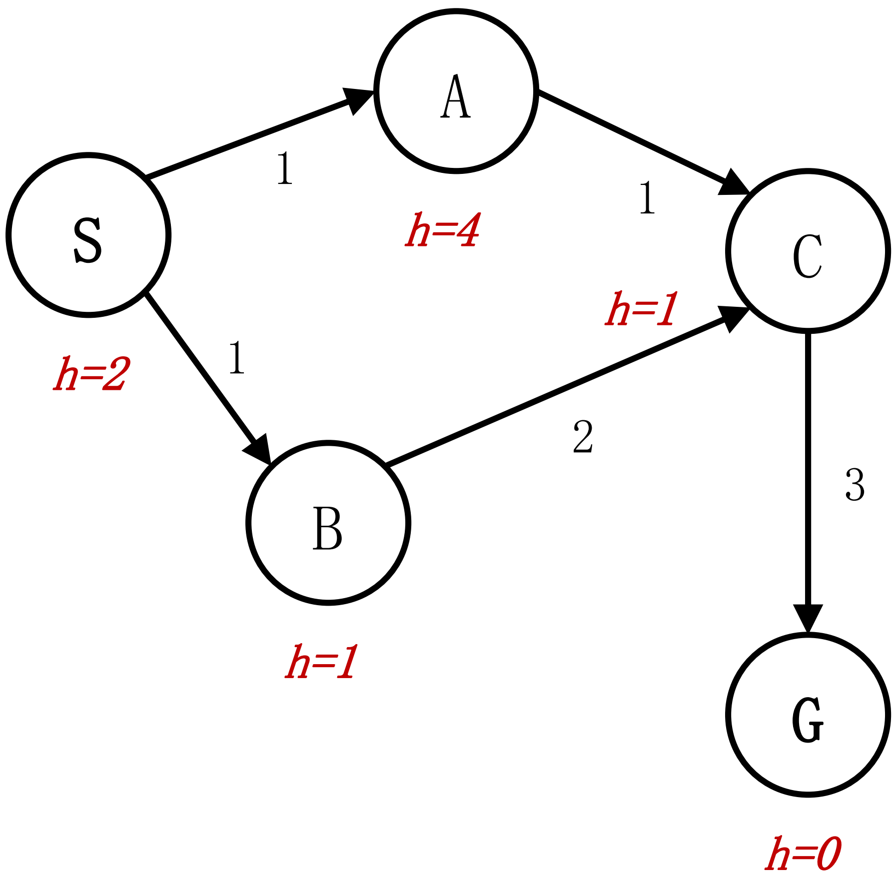


Graph Search

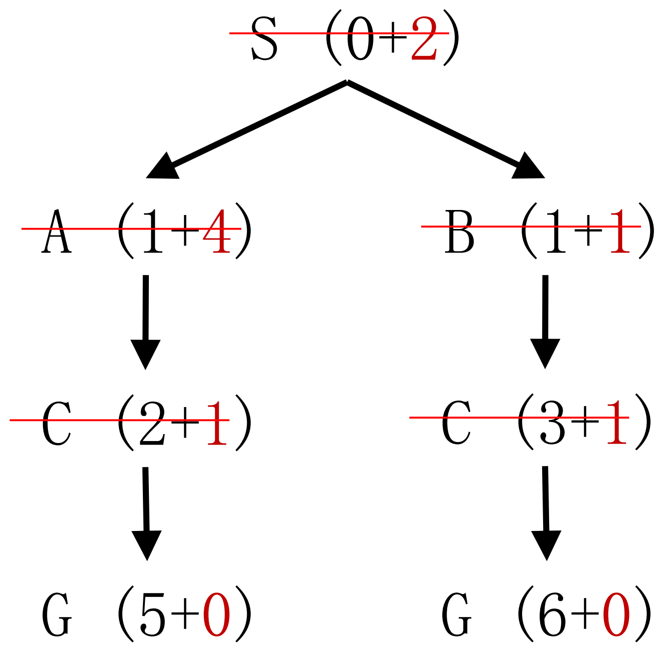
- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states (“closed set”)
  - Expand the search tree node-by-node, but…
  - **Before expanding a node, check to make sure its state has never been expanded before**
  - If not new, skip it, if new add to closed set
- 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

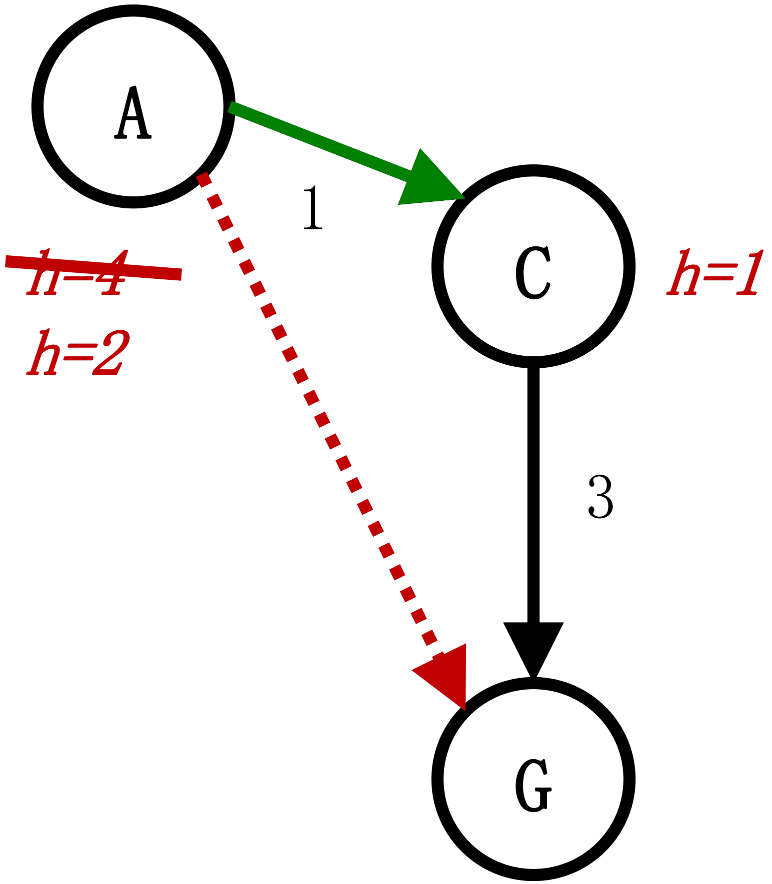


Search tree



Closed Set: S B C A

Consistency of Heuristics



- Main idea: estimated heuristic costs ⩽ actual costs
  - Admissibility: heuristic cost ⩽ actual cost to goal

    h(A) ⩽ actual cost from A to G
  - Consistency: heuristic "arc" cost ⩽ actual cost for each arc

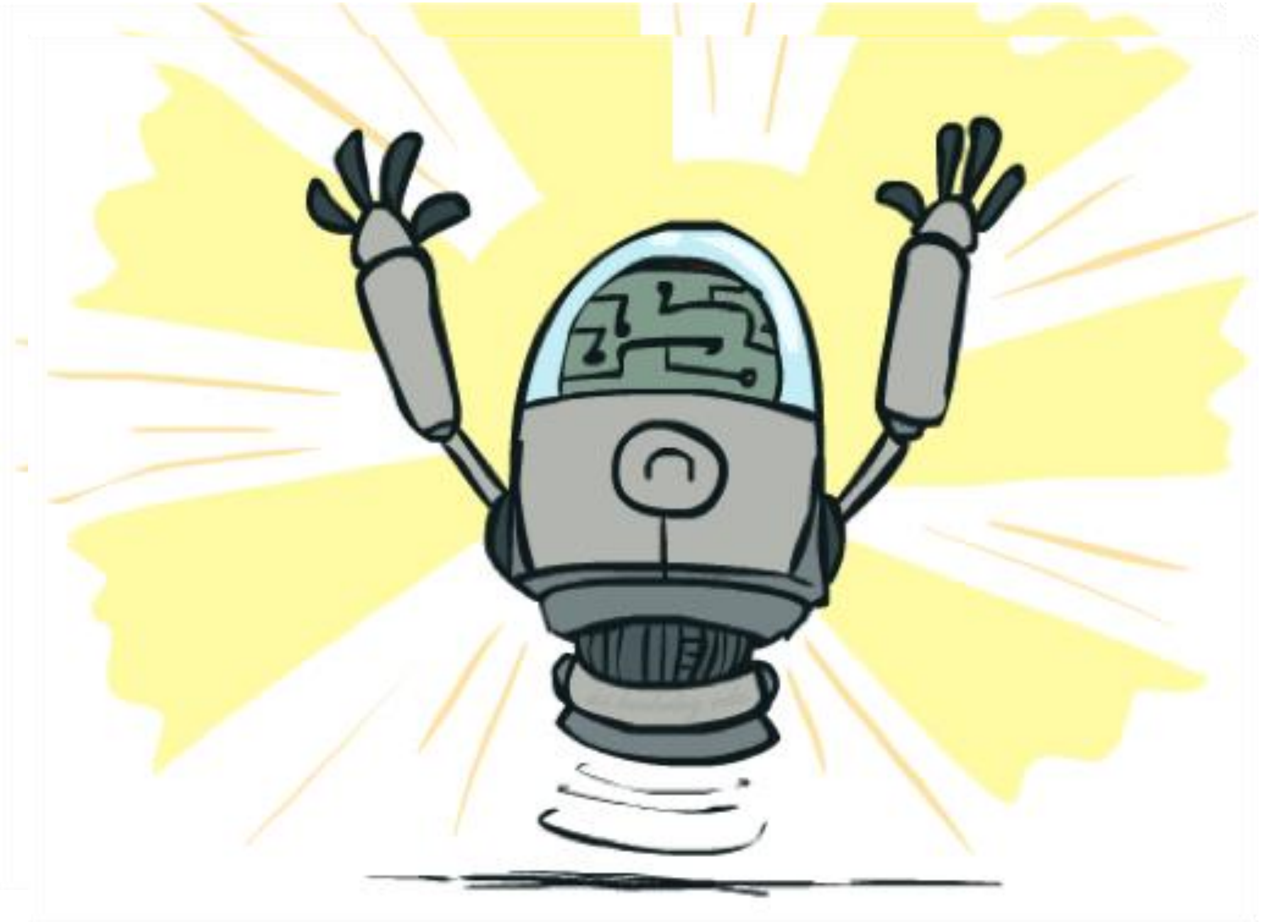
    h(A) - h(C) ⩽ cost(A to C)
- Consequences of consistency:
  - The f value along a path never decreases ->consistency can imply the admissity.

    h(A) ⩽ cost(A to C) + h(C)
  - A* graph search is optimal

Optimality of A* Search

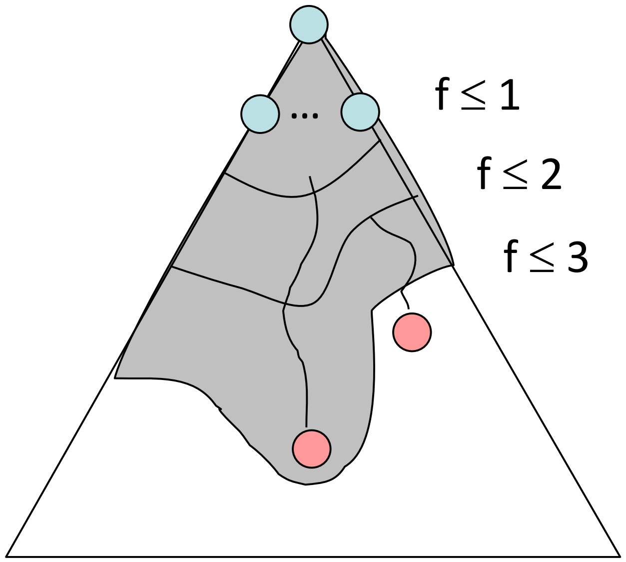
- With a admissible heuristic, Tree A* is optimal.
- With a consistent heuristic, Graph A* is optimal.
  - See slides, also video lecture from past years for details.
- With h=0, the same proof shows that UCS is optimal.

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)-> this is derived by consistency definition.
  - 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

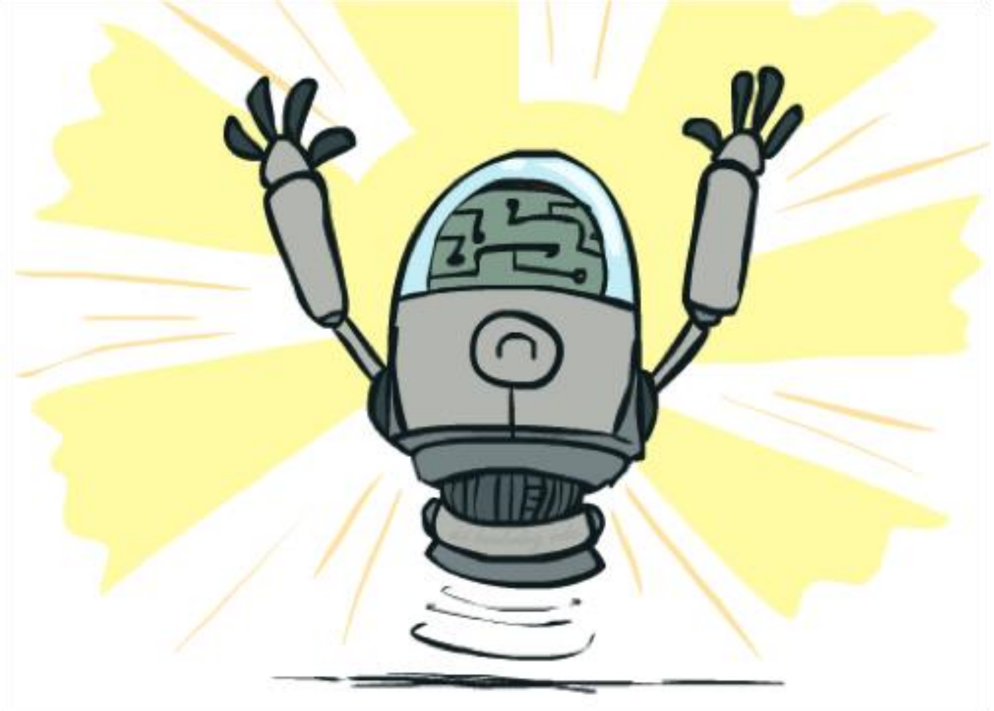


Optimality

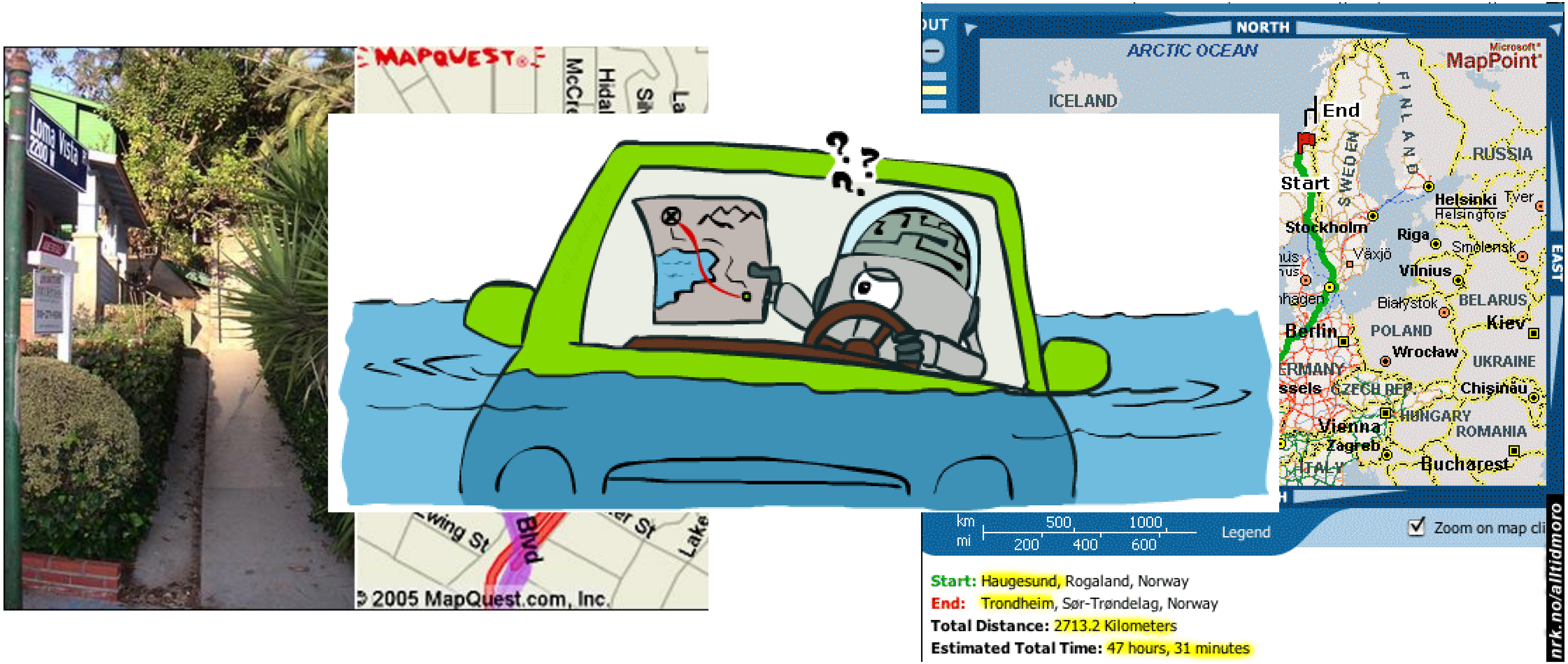
- Tree search:
  - A* is optimal if heuristic is admissible
  - UCS is a special case (h = 0)
- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal (h = 0 is consistent)

- **Consistency implies admissibility**

- In general, most *natural* admissible heuristics tend to be consistent, especially if from relaxed problems ->
  Thus, if we want to find a consistent heuristics(this is also admissible), we just need to find the solutions for the relaxed problems



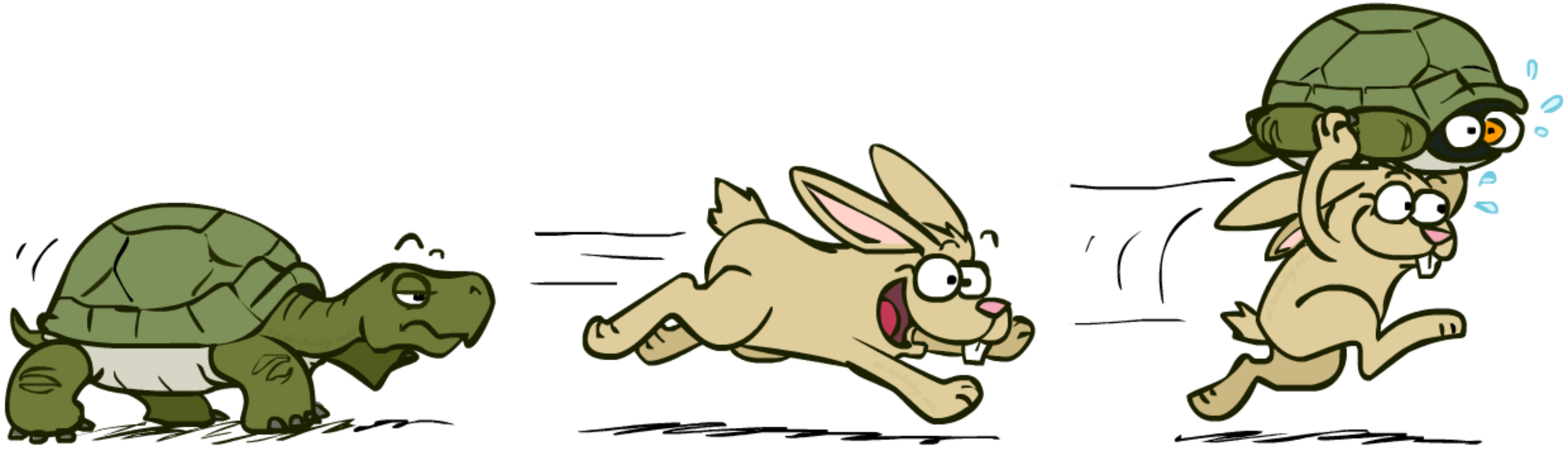
Search Gone Wrong?



A*: Summary

A*: Summary

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



Tree Search Pseudo-Code

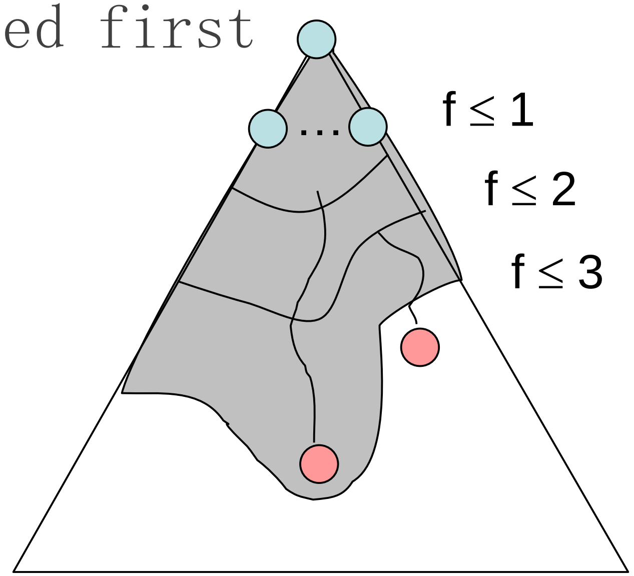
```
function TREE-SEARCH(problem, fringe) return a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        for child-node in EXPAND(STATE[node], problem) do
            fringe ← 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: $f(n) = g(n) + h(n)$ = cost to n + heuristic
  - Proof idea: the optimal goal(s) have the lowest f value, so it must get expanded first

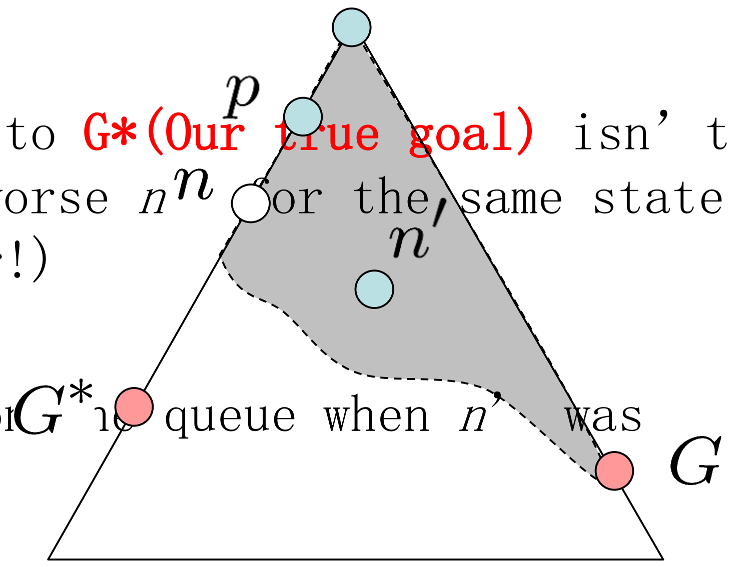


There's a problem with this argument. What are we assuming is true?

Optimality of A* Graph Search

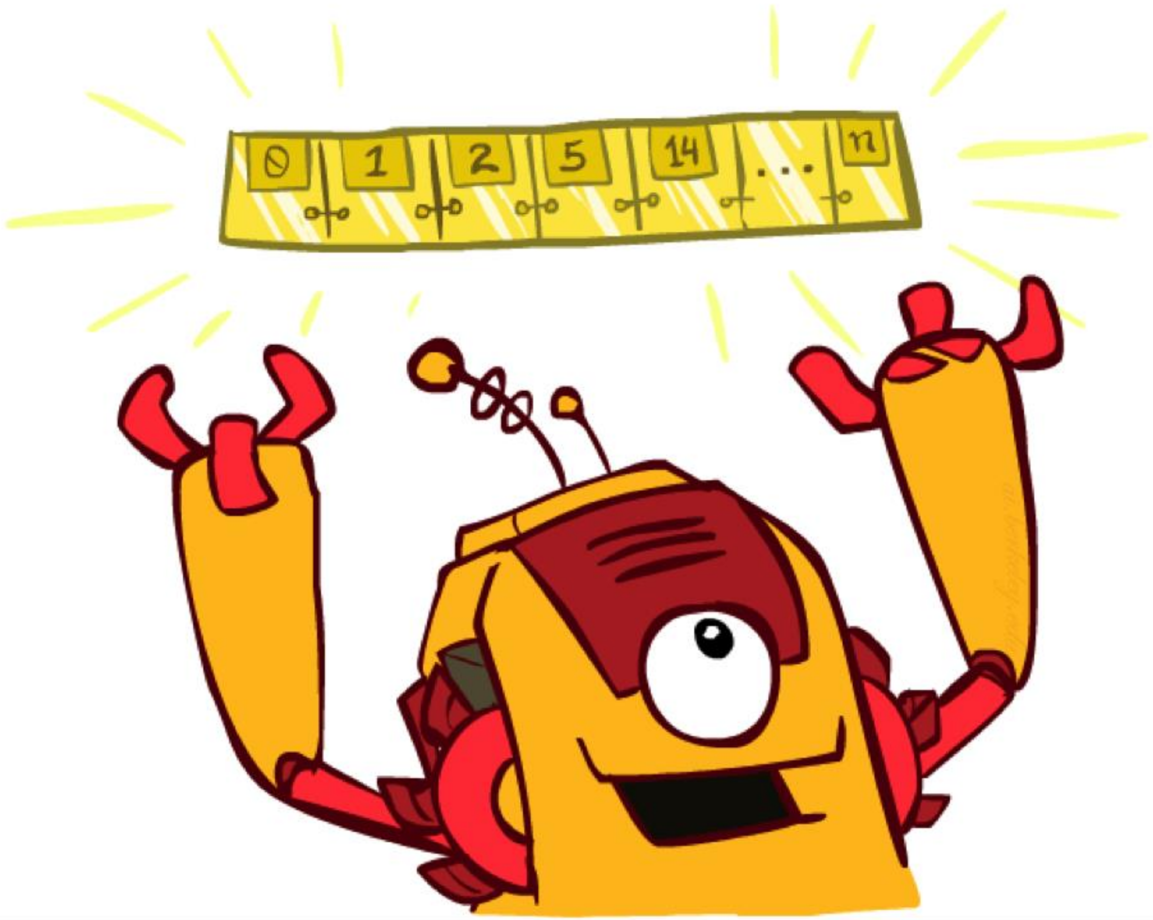
Proof:

- New possible problem: some *n* on path to G*(Our true goal) 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 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!-> It can happen that G’ is reached due to we’ve reached n’ and G* is not reached due to we haven’t reached n. Because p must be reached before n’ is reached, and n should be reached before n’ because p is reached.



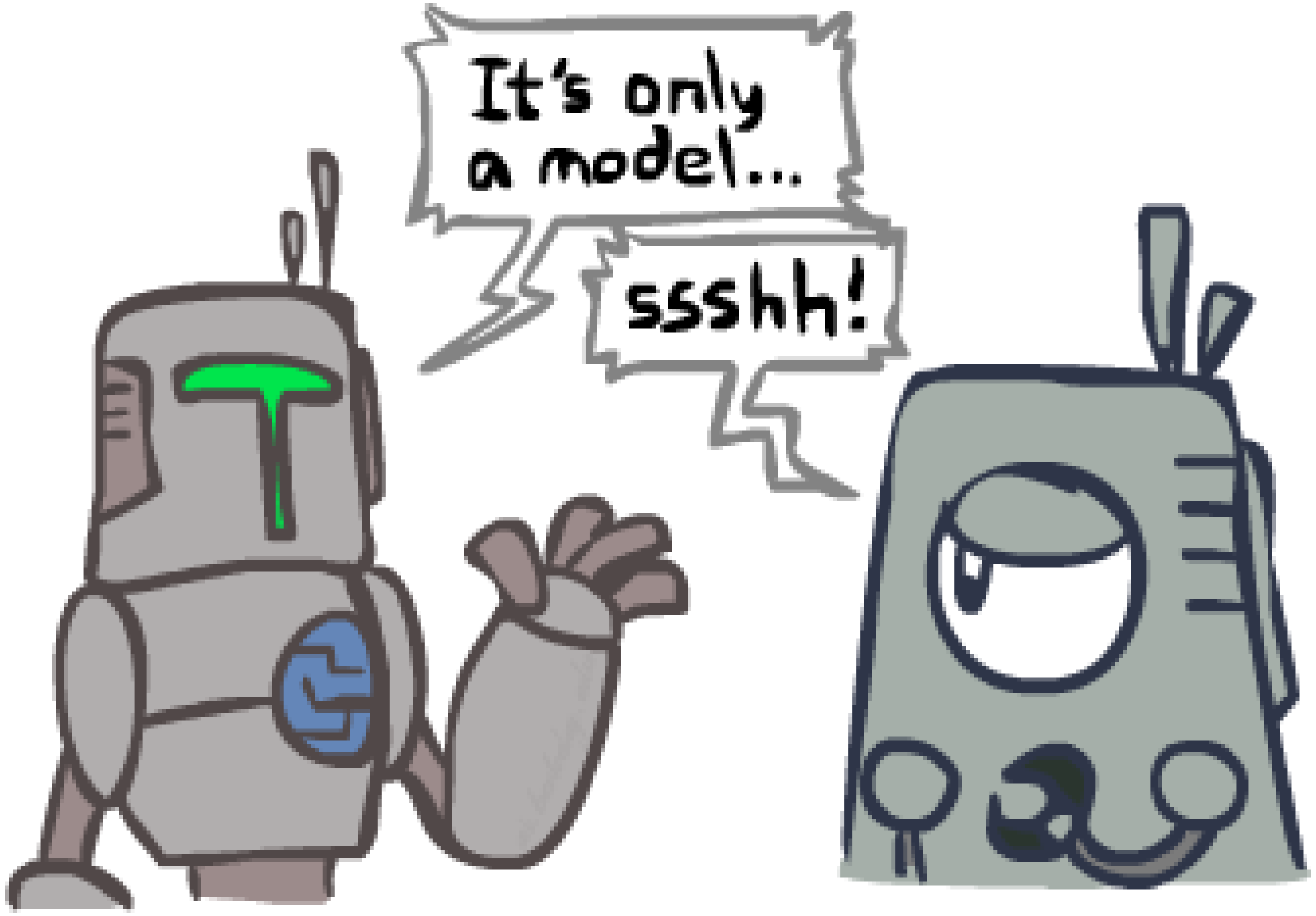
The One Queue

- 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



Search and Models

- Search operates over models of the world
  - The agent doesn't actually try all the plans out in the real world!
  - Planning is all "in simulation"
  - Your search is only as good as your models…



Search Gone Wrong?