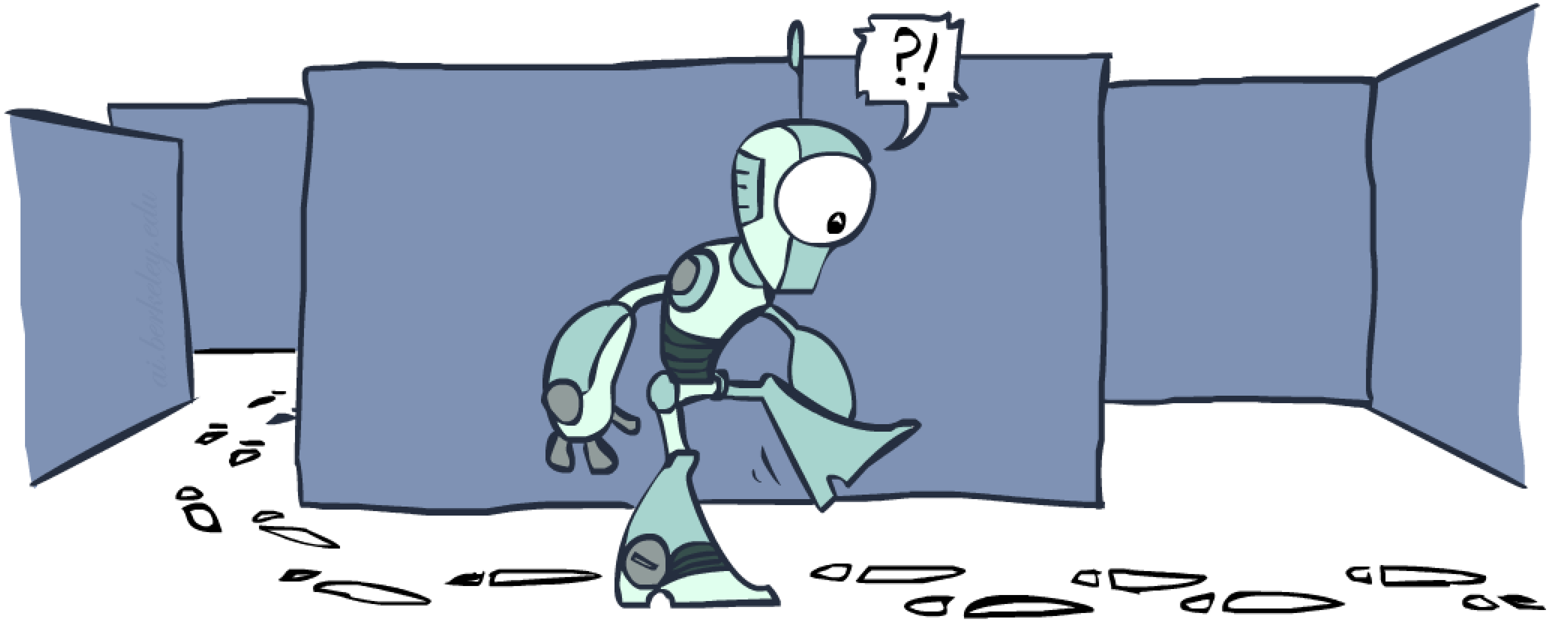
Solving Problems with Search: 4

Today's Plan

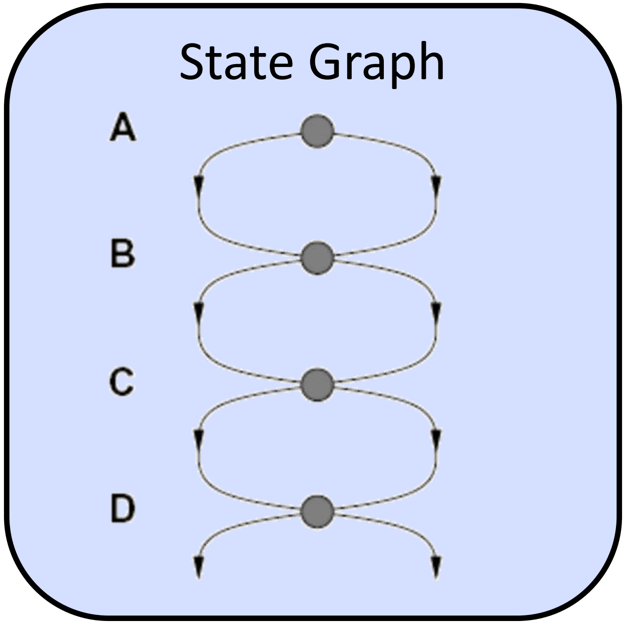
- Project 1 comments and Q&A
- Review: A*
- Properties of A* algorithm and heuristics

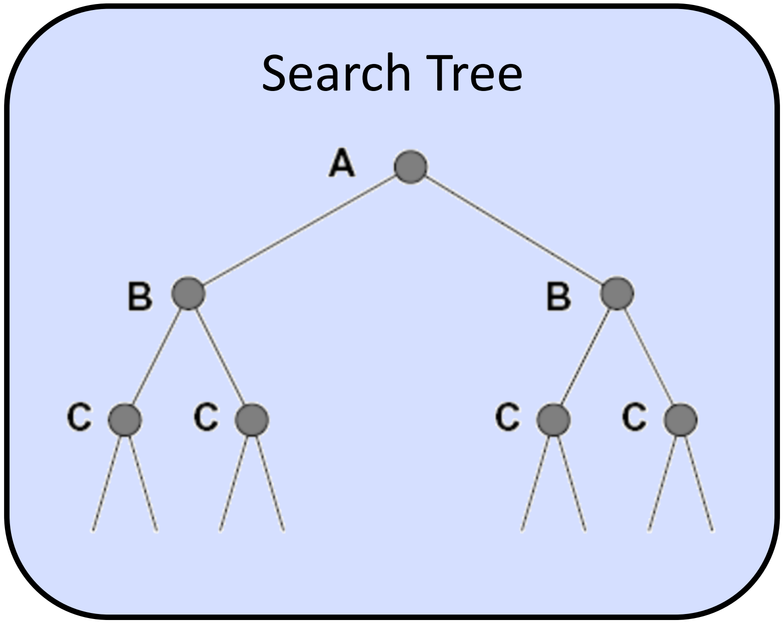
Graph Search



Tree Search: Extra Work!

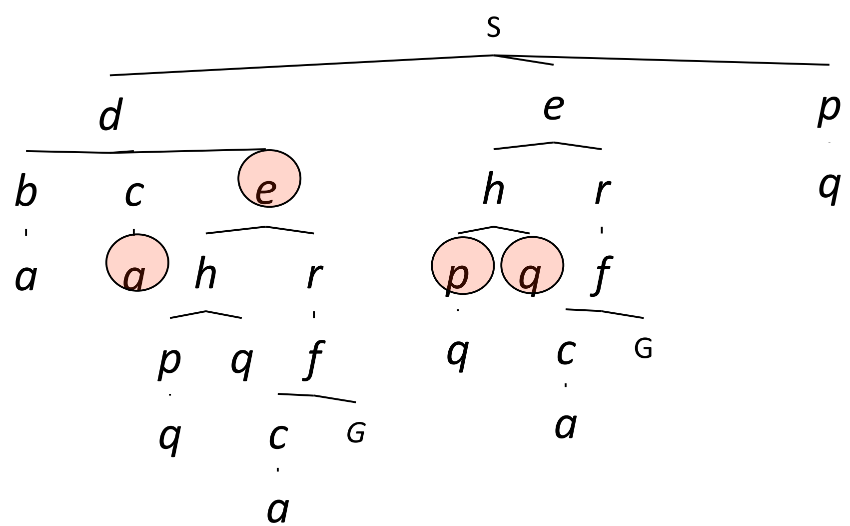
- Failure to detect repeated states can cause exponentially more work.





Graph Search

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



Graph Search: **Implementation**

- 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 expanded: skip it, if new: add to closed set
- **Efficiency tip**: store the closed set as a set, not a list (why?)

Graph Search Pseudo-Code

Why do you need graph search for Pacman?

```
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
```

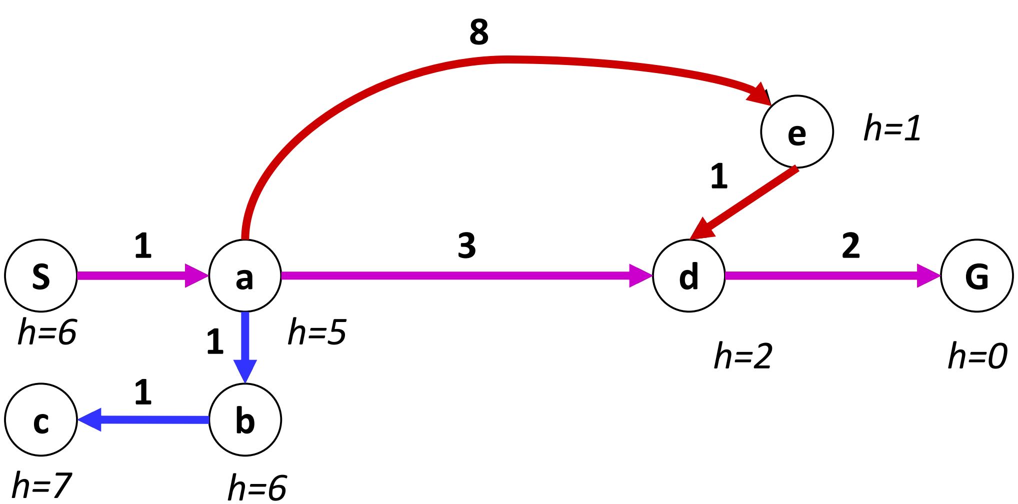
Need this check to handle loops

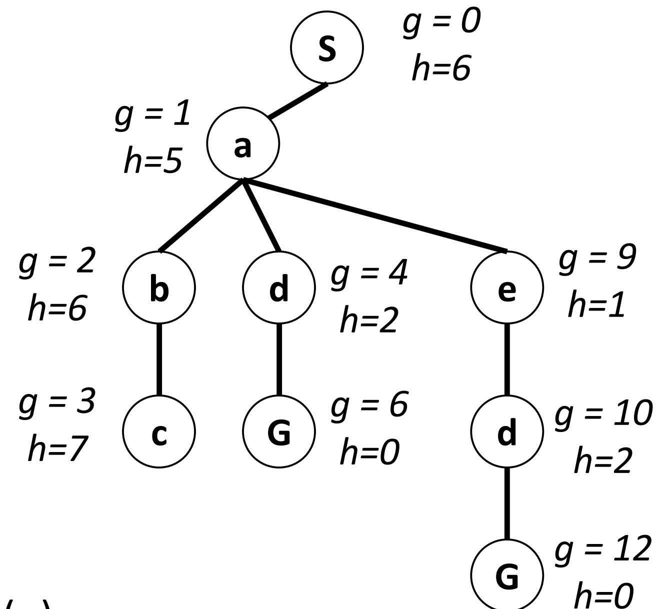
Project 1 Questions & Tips

- Use Piazza, read FAQ before posting questions: https://piazza.com/emory/spring2017/cs325/
- **Questions 1-4**: if you develop a correct solution for DFS, the rest will be very easy modifications
- Do not use shortcuts: use Node class or similar: https://piazza.com/class/ixql4613j9k223
- Questions 5-8: more fun/creative. Leave enough time.

A* Review: f(n) = UCS + Heuristic

- 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

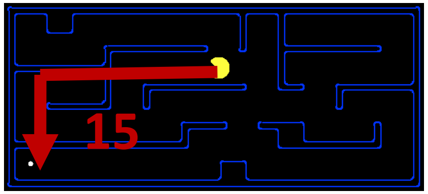
Admissible Heuristics

- 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:

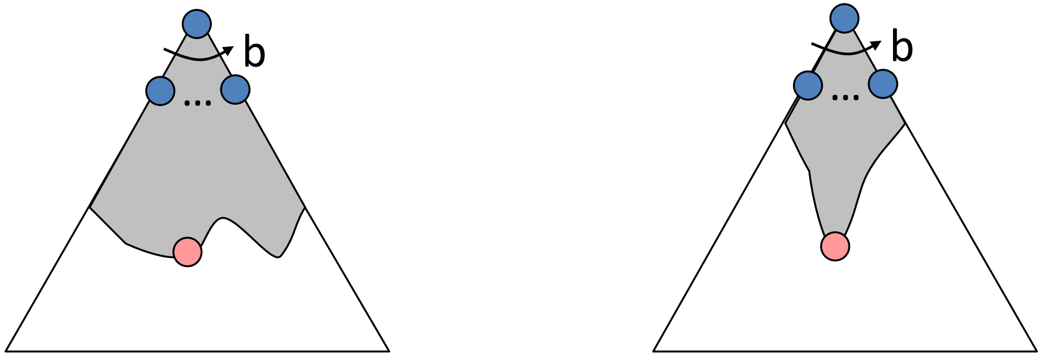


- Coming up with admissible heuristics is most of what's involved in using A* in practice.

Properties of A*

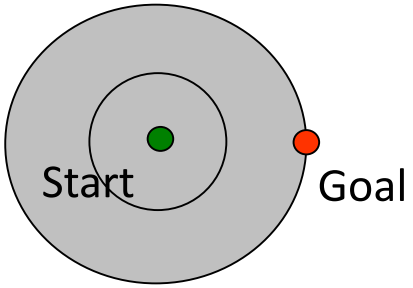
Properties of A*

Uniform-Cost | A*

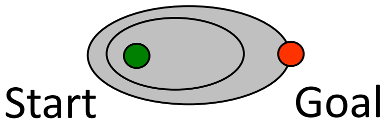


UCS vs A* Contours

- Uniform-cost expands equally in all “directions”

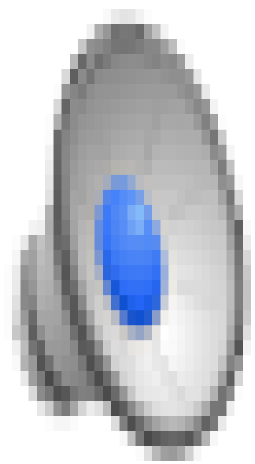


- A* expands mainly toward the goal, but does hedge its bets to ensure optimality

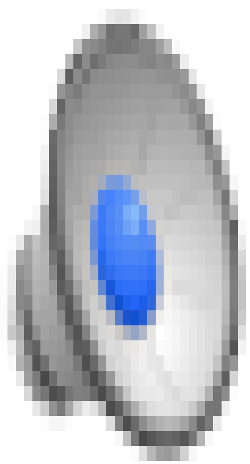


[Demo: contours UCS / greedy / A* empty (L3D1)]
[Demo: contours A* pacman small maze (L3D5)]

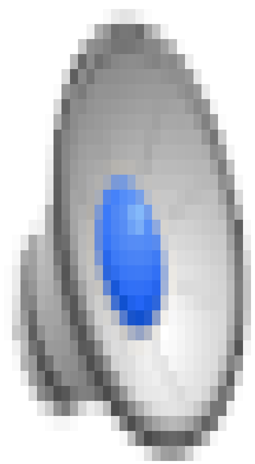
Video of Demo Contours (Empty) -- UCS



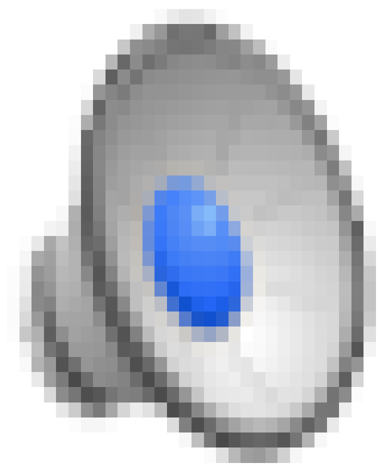
Video of Demo Contours (Empty) -- Greedy



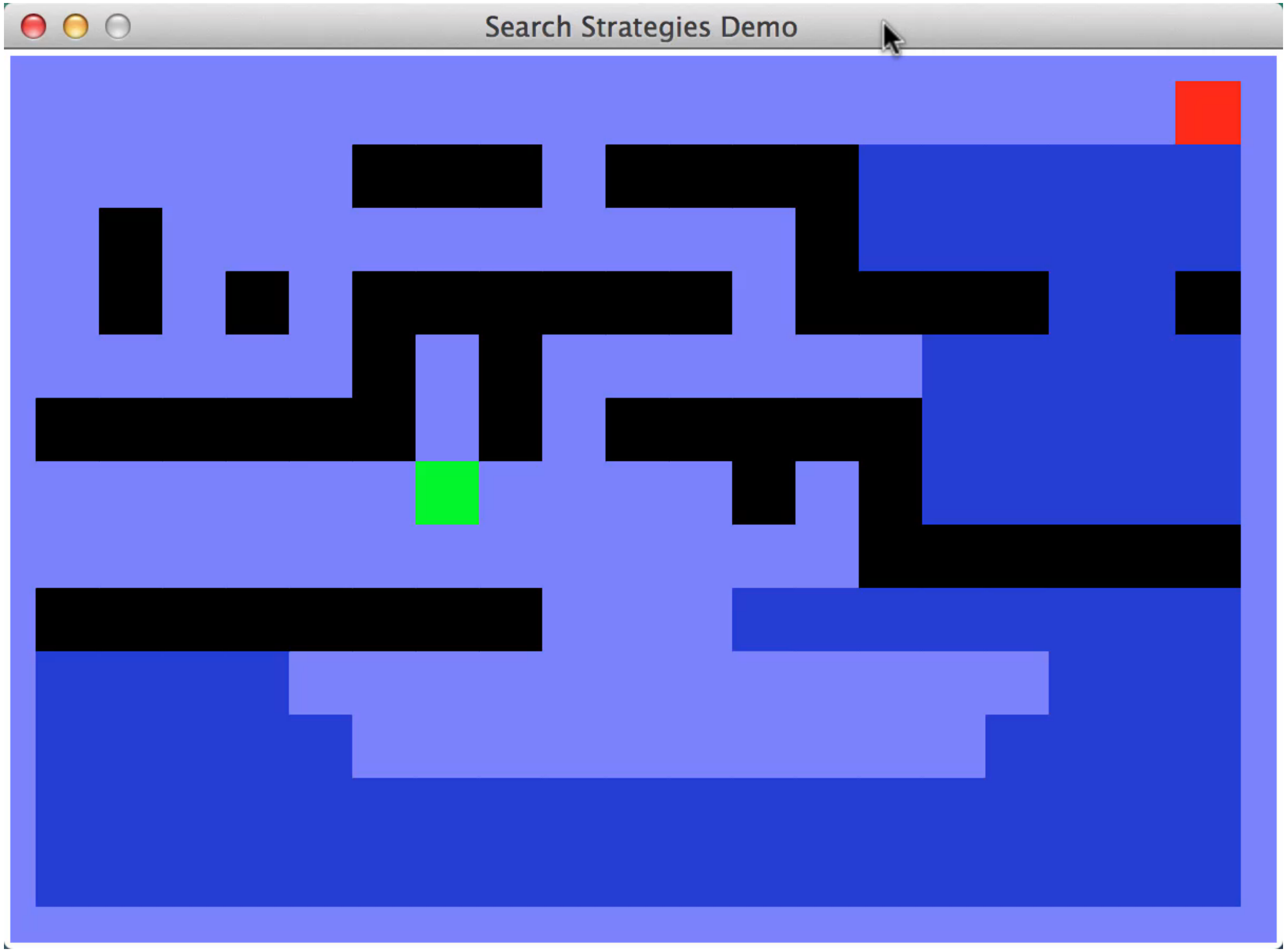
Video of Demo Contours (Empty) – A*



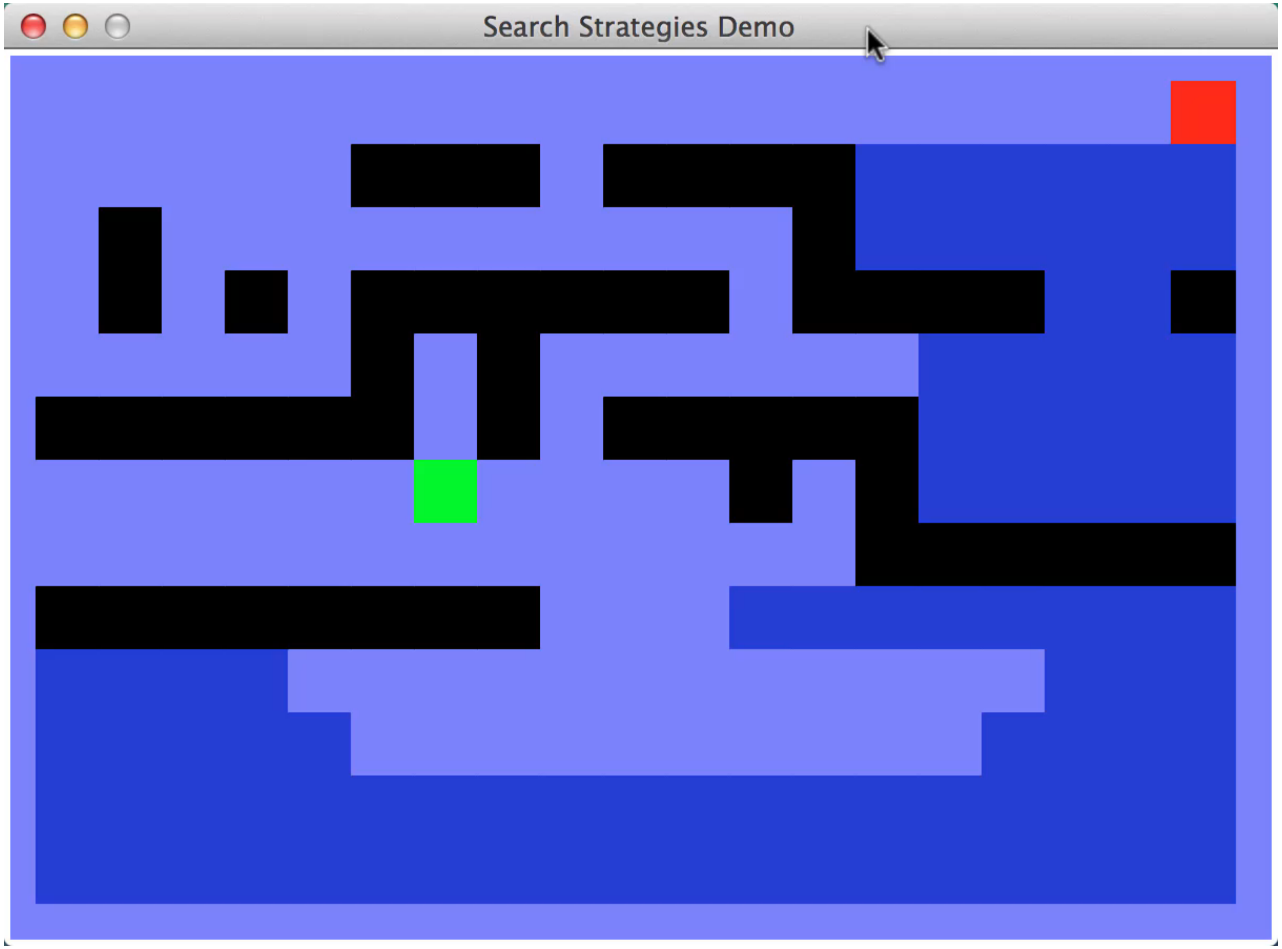
Video of Demo Contours (Pacman Small Maze) – A*



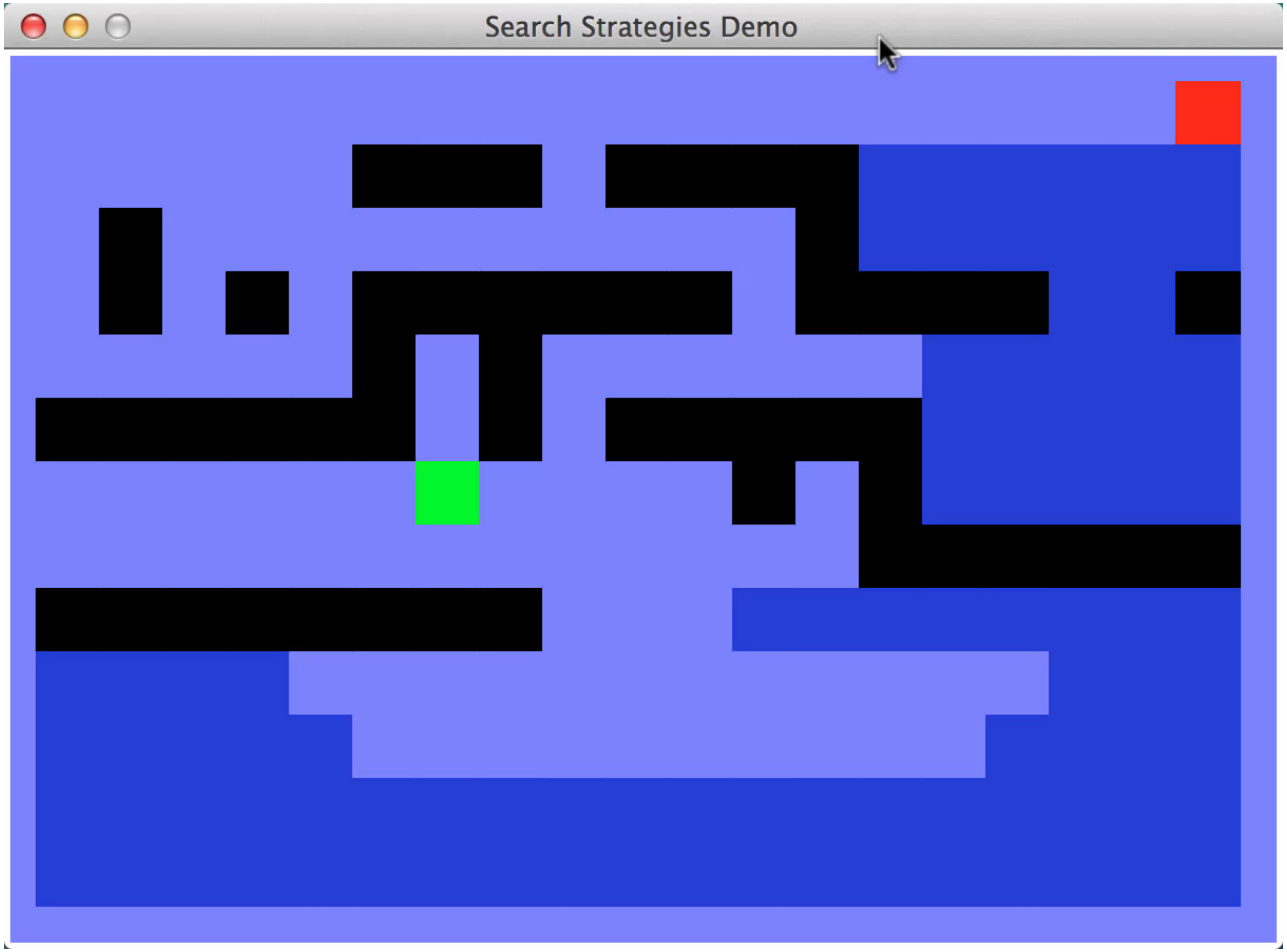
Which Algorithm (1)?



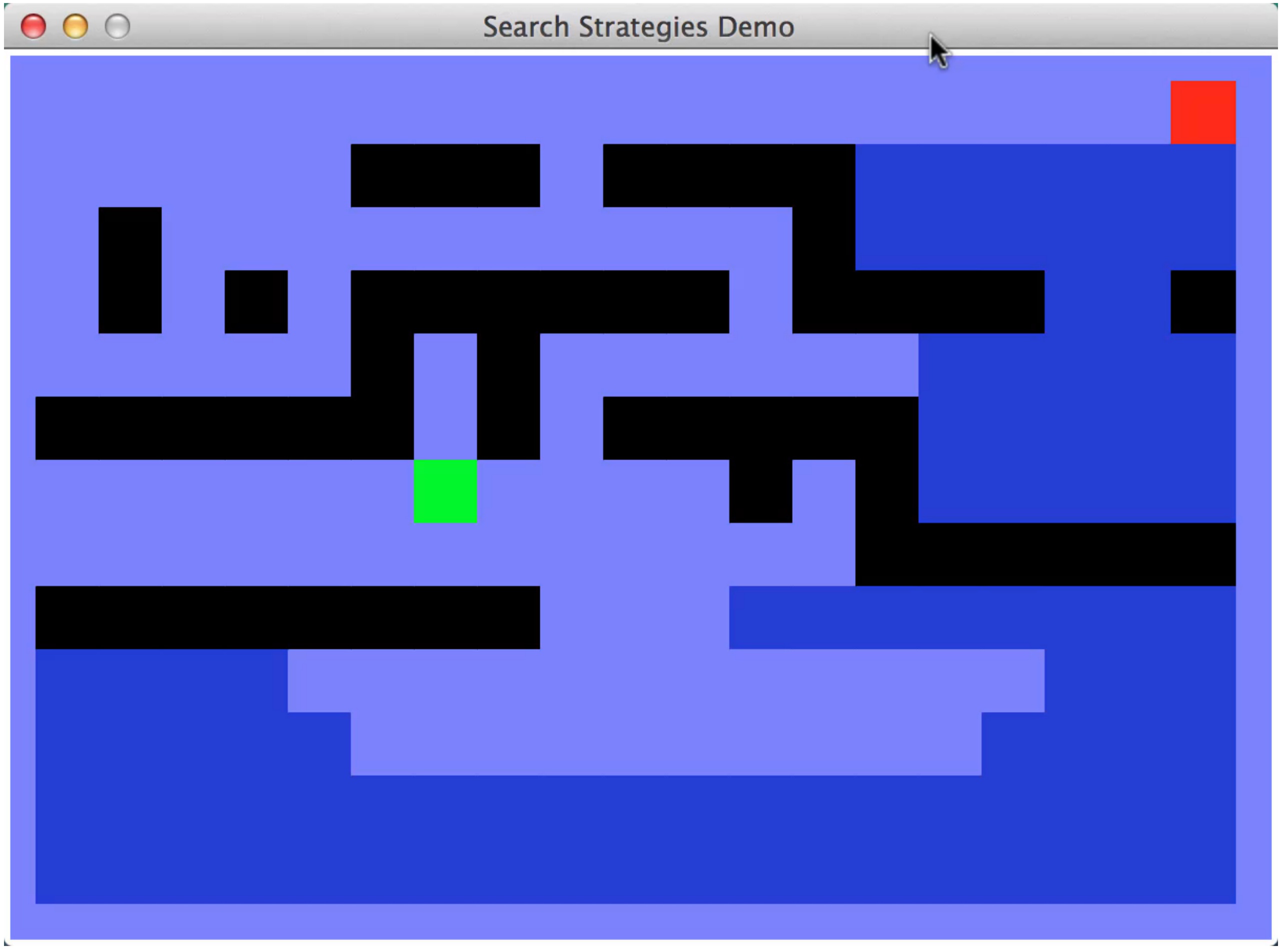
Which Algorithm (2)



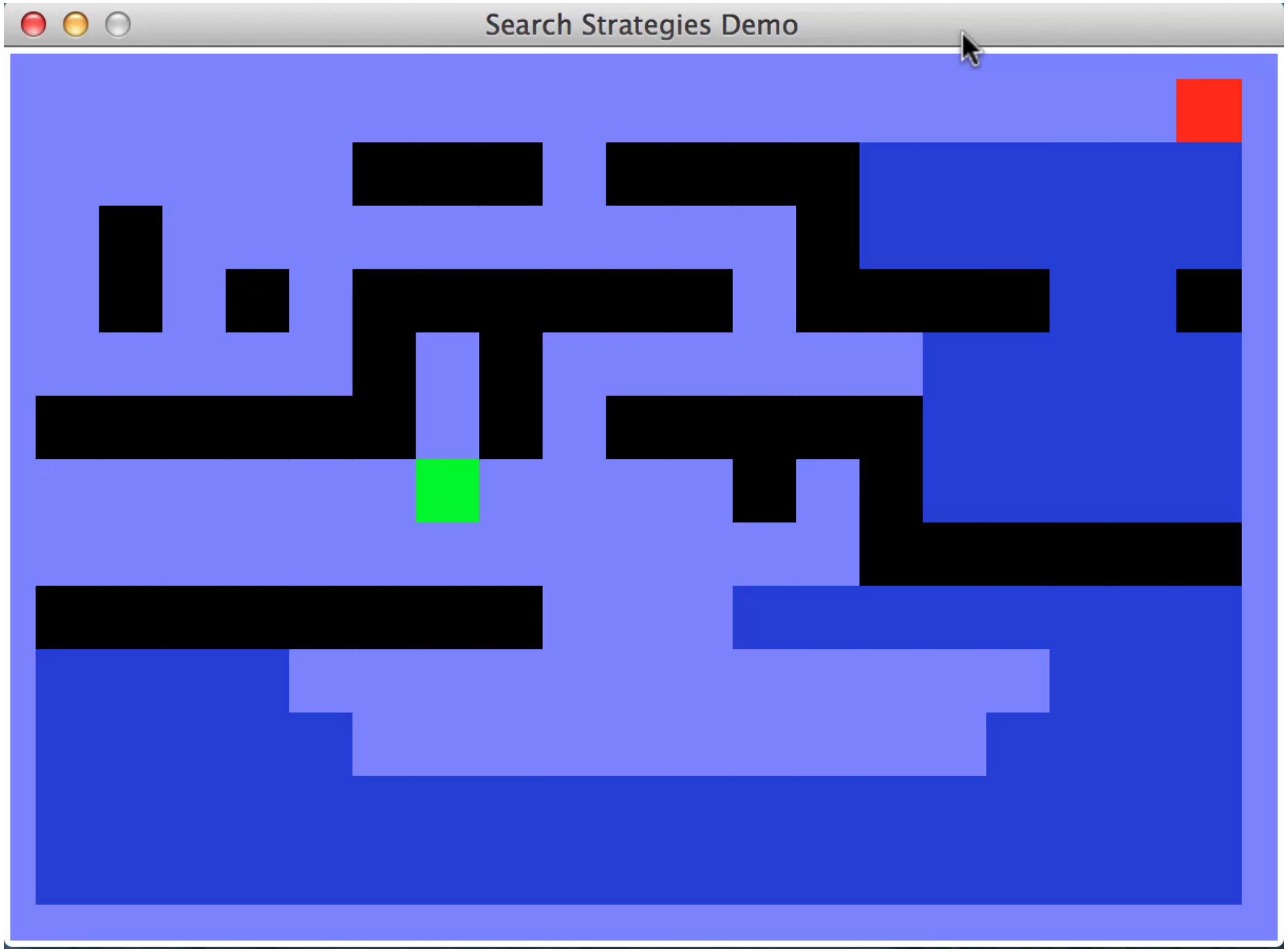
Which Algorithm (3)



Which Algorithm (4)?

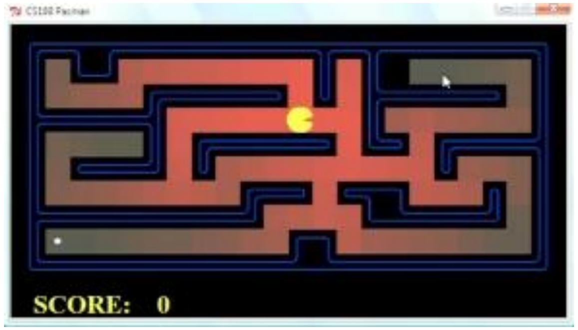


Which Algorithm (5)



Comparison: Summary

Greedy

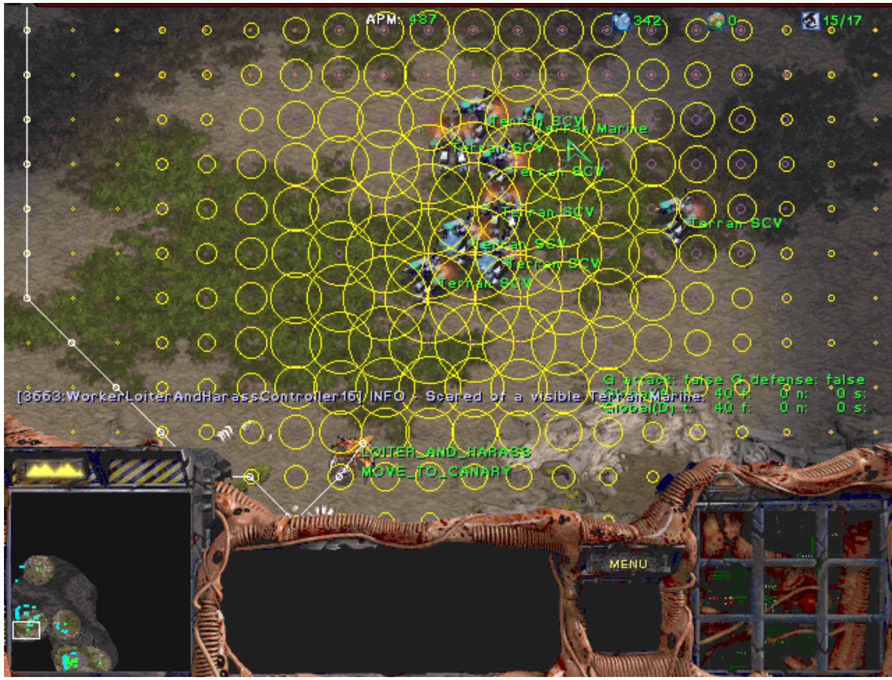


Uniform Cost

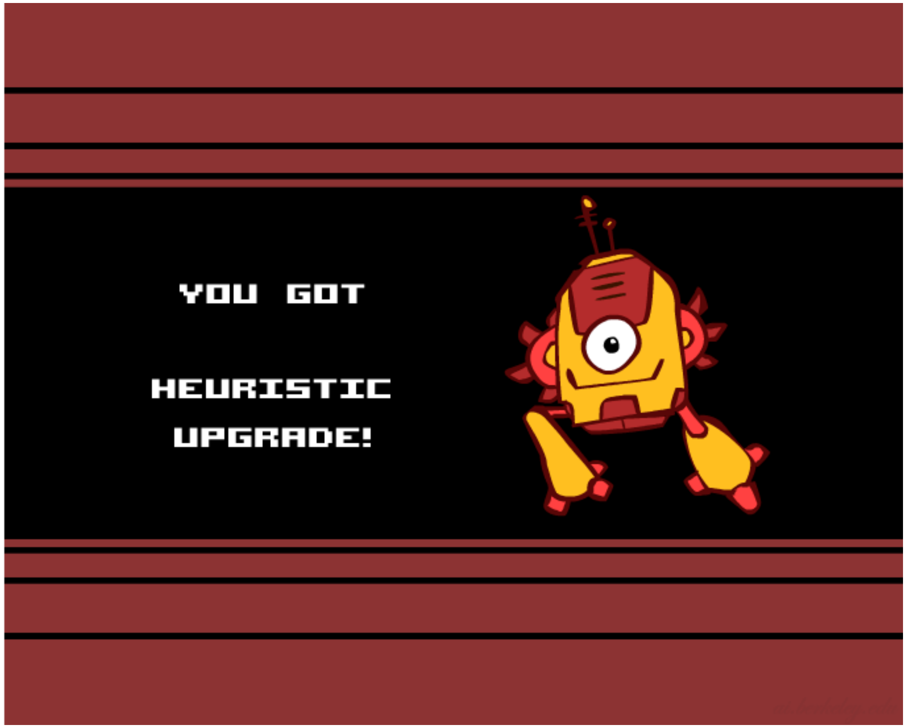
A*

A* Applications

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



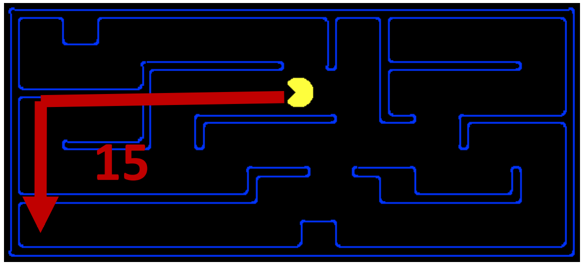
Creating Heuristics



Creating Admissible Heuristics

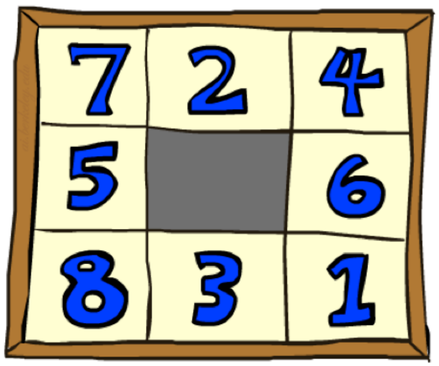
- **Most of the work in solving hard search problems optimally is in coming up with admissible heuristics**

- Often, admissible heuristics are solutions to *relaxed problems,* where new actions are available

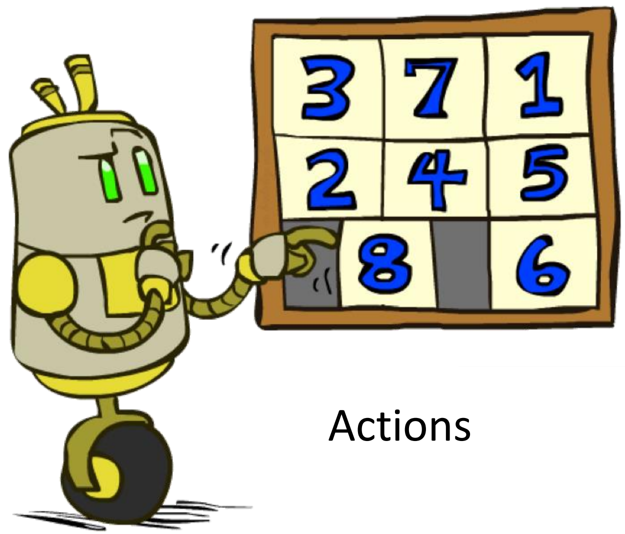


- Inadmissible heuristics are often useful too!

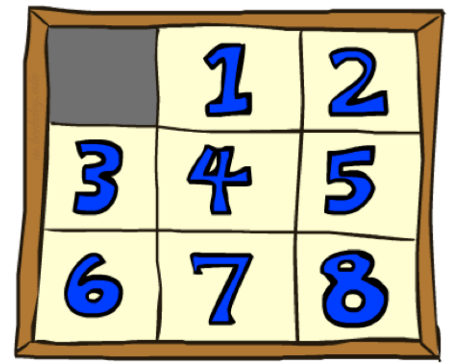
Example: 8 Puzzle



Start State



Actions

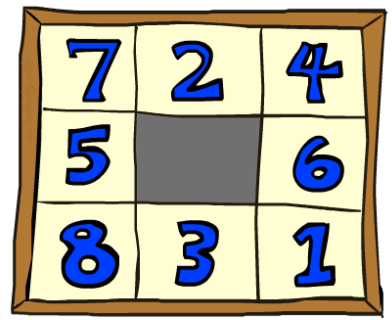


Goal State

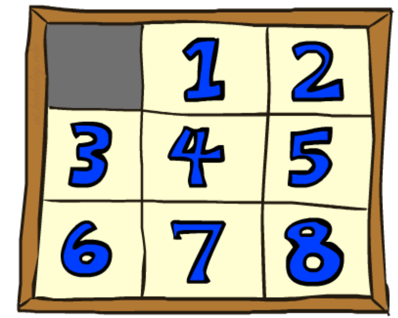
- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

8 Puzzle I

- Heuristic:
- Is it admissible?
- h(start) =
- h(goal) =



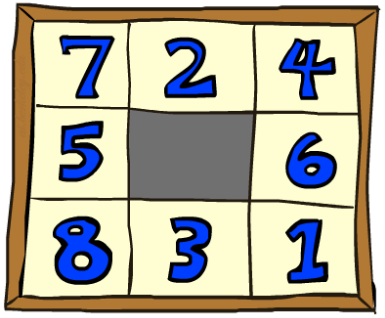
Start State



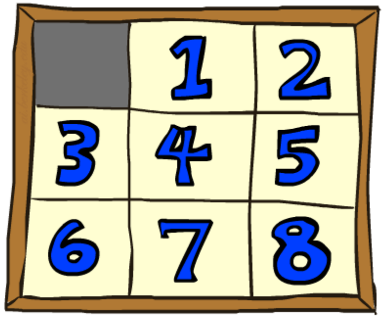
Goal State

8 Puzzle: **Tiles** heuristic

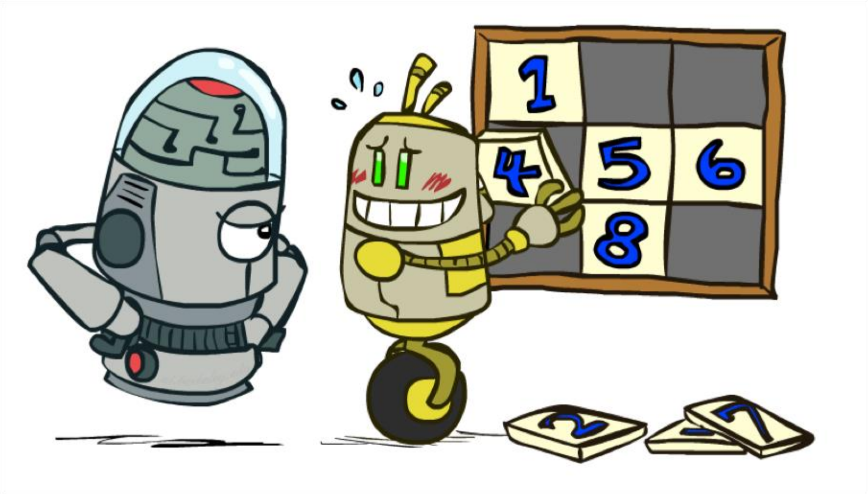
- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) =8
- This is a *relaxed-problem* heuristic



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

Statistics from Andrew Moore

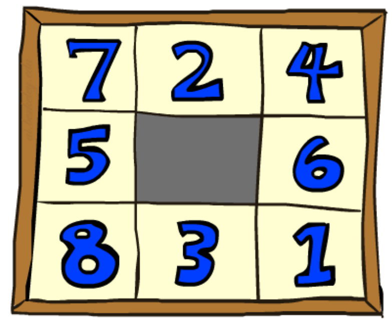
8 Puzzle II: **Manhattan** heuristic

- **Relaxation**: easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?

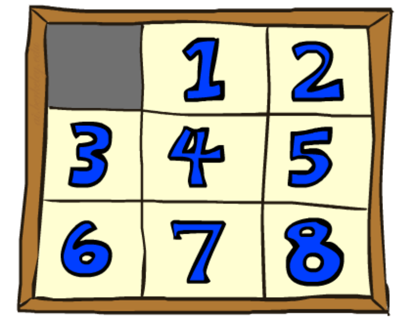
- Total *Manhattan* distance from correct location

- Is it admissible?

- h(start) = 3 + 1 + 2 + … = 18



Start State

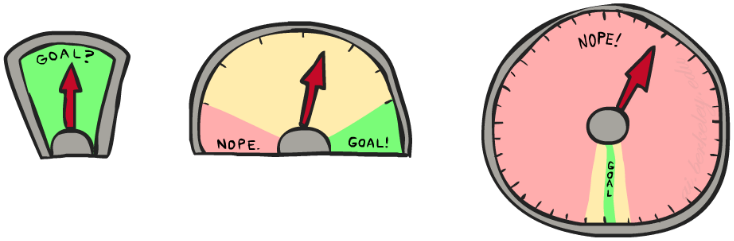


Goal State

	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: **Oracle** heuristic

- 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

Recap: **Problem Relaxation**

- A problem with fewer restrictions on the actions is called a relaxed problem

- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem

- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution

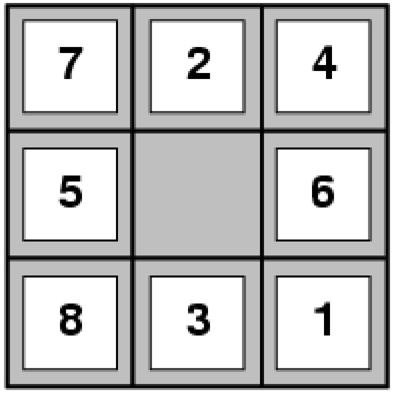
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution

Designing heuristics (cont'd)

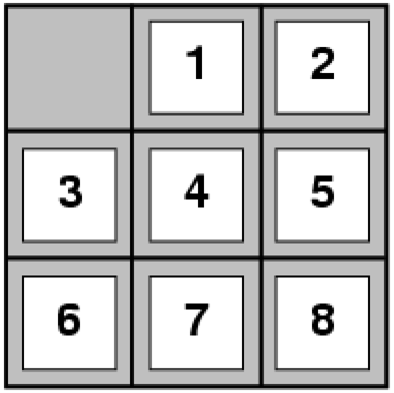
E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



Start State



Goal State

- <u>$h_1(S)$ = ?</u>
- <u>$h_2(S)$ = ?</u>

Heuristics: cont'd

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- $h_1(S) = ?$ 8
- $h_2(S) = ?$ 3+1+2+2+2+3+3+2 = 18

Which is "better" – h1 or h2?

Idea: Heuristic **dominance**

- If $h_2(n) \geq h_1(n)$ for all n (both admissible, i.e., < true cost)
  then h_2 dominates h_1

  h_2 is better for search

- Typical search costs (average number of nodes expanded):

- *d=12* IDS = 3,644,035 nodes
  $A^*(h_1)$ = 227 nodes
  $A^*(h_2)$ = 73 nodes
- *d=24* IDS = too many nodes
  $A^*(h_1)$ = 39,135 nodes
  $A^*(h_2)$ = 1,641 nodes

Example: Heuristics for Chess

- To select next move, must evaluate expected benefit of successor position:
  - **Value of the pieces** (count value of your pieces – value of opponents pieces)
  - **Space**: threatened/controlled space by you – space controlled by opponent
  - **Pawn** structure
  - …
- Examples:
  - https://www.quora.com/What-are-some-heuristics-for-quickly-evaluating-chess-positions
  - https://chessprogramming.wikispaces.com/Killer+Heuristic

Example: Heuristics for Motion Planning

- Robot motion: many moving (body) parts
- What's the most efficient way to accomplish goal?

https://www.youtube.com/watch?v=dSwDZmvtGZY

Example: Machine Translation

- 1. Translate words from source to target
- 2. Choose the "more likely" translation among candidates
- h(t) = count of phrase seen in target language
- What could go wrong…

Example: Machine Translation

- h(t) = count of phrase seen in target language

Designing Heuristics

- A good heuristic is:
  - ✓ Admissible (optimistic)
  - ➢ Consistent (non-decreasing)
  - ✓ “Accurate”

Trivial Heuristics, Dominance

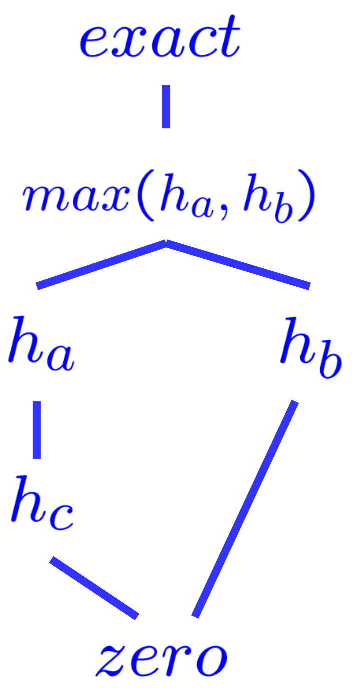
- Dominance: $h_a \geq h_c$ if

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

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

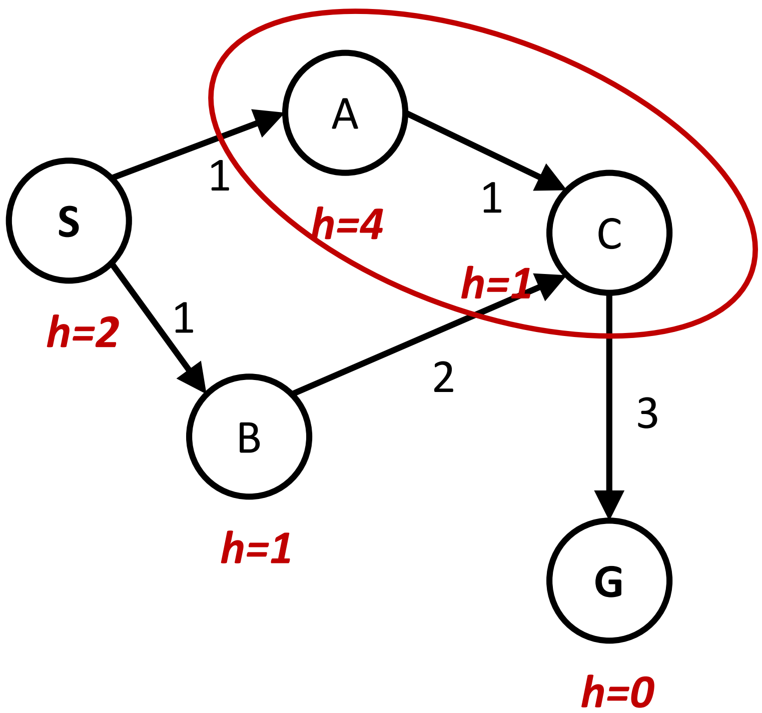
$$h(n) = max(h_a(n), h_b(n))$$

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

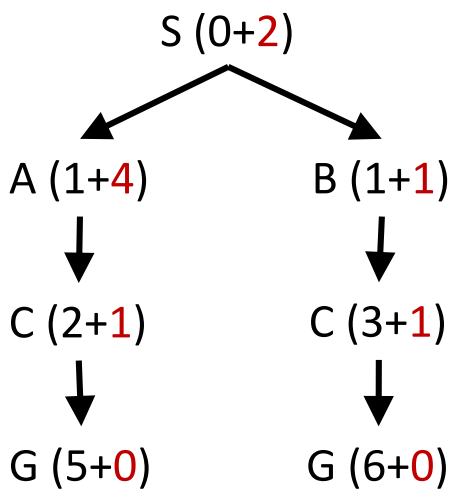


A* Graph Search Gone Wrong?

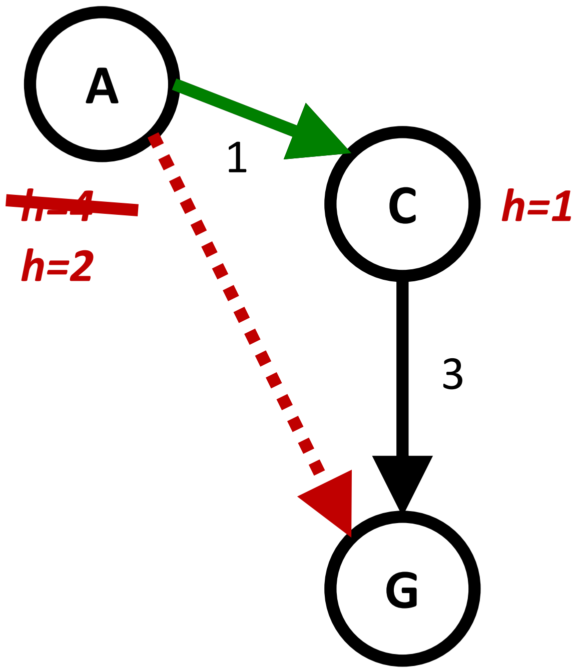
State space graph



Search tree



Consistency of Heuristics



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

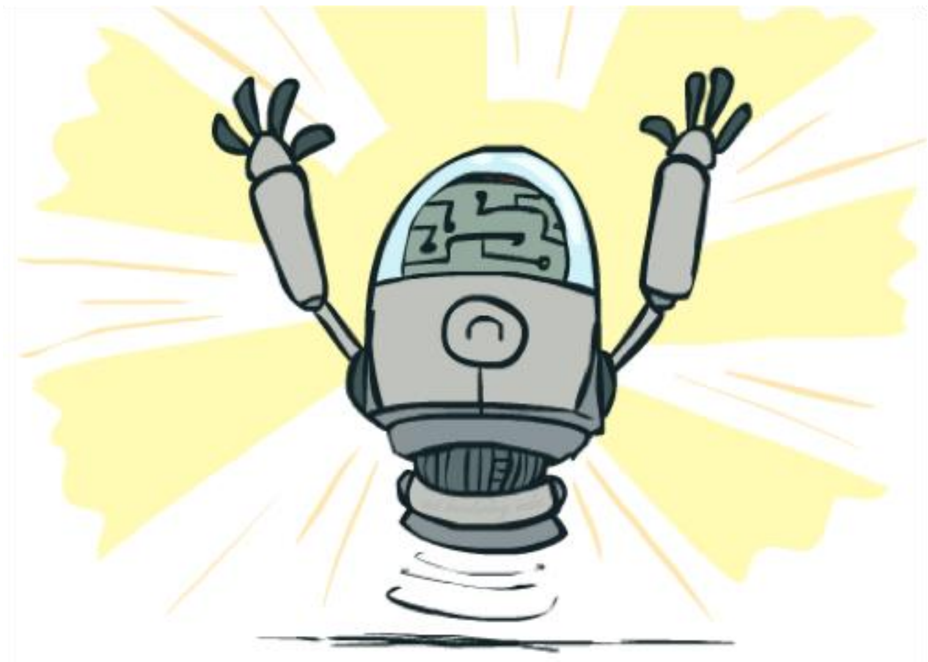
    h(A) ≤ actual cost from A to G
  - Consistency: heuristic "arc" cost ≤ actual cost for each arc

    h(A) – h(C) ≤ cost(A to C)

- Consequences of consistency:
  - The f value along a path never decreases

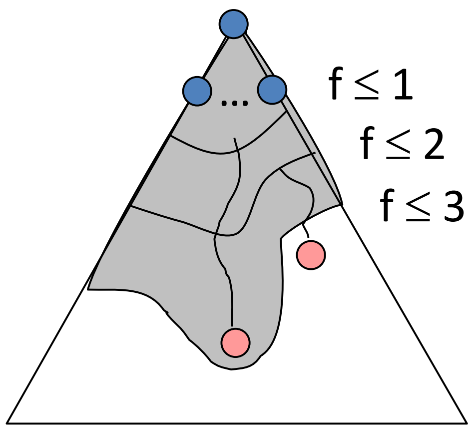
    h(A) ≤ cost(A to C) + h(C)
  - A* graph search is optimal

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)
  - **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



??? Uhm... we already proved that for **admissible** heuristics???

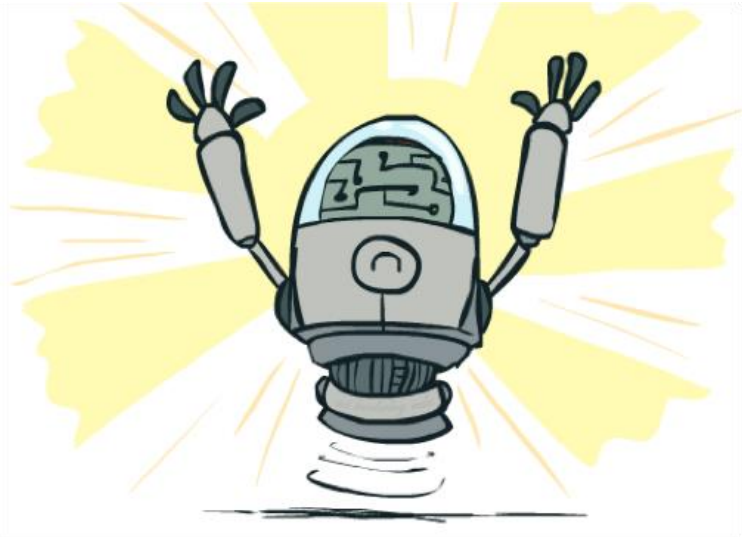
Optimality (2): Tree vs. Graph Search

- **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



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