Informed Search

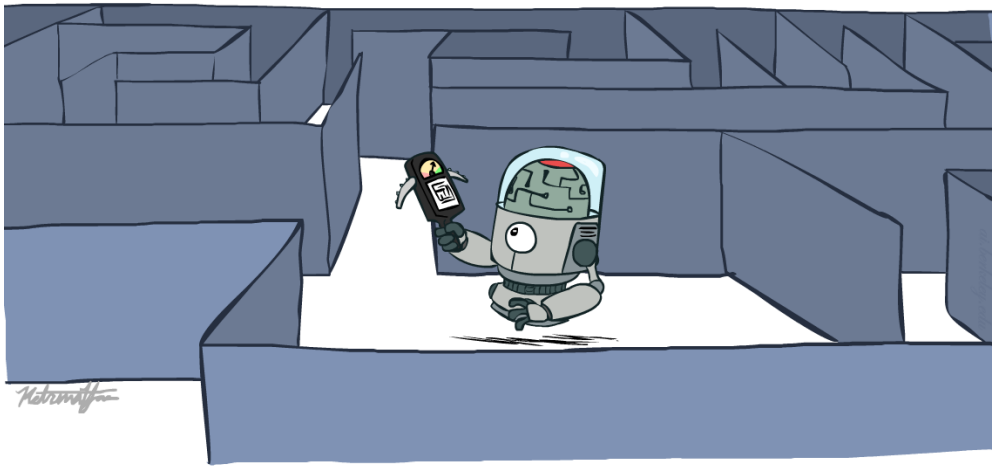
Lecture 5

How can we exploit problem-specific knowledge to find solutions more efficiently?

Where does this knowledge come from and under what conditions is it useful?

Agenda

- Review: Uninformed Search
- Informing Search via Heuristics
- Best-First Search: Greedy, A*
  - Properties: Admissibility, Consistency
- Heuristic Origins



Search Problem Formalism

Defined via the following components:

- The **initial state** the agent starts in
- A **successor/transition function**
  - *S*(x) = {action+cost->state}
- A **goal test**, which determines whether a given state is a goal state
- A **path cost** that assigns a numeric cost to each path

A **solution** is a sequence of actions leading from initial state to a goal state. (**Optimal** = lowest path cost.)

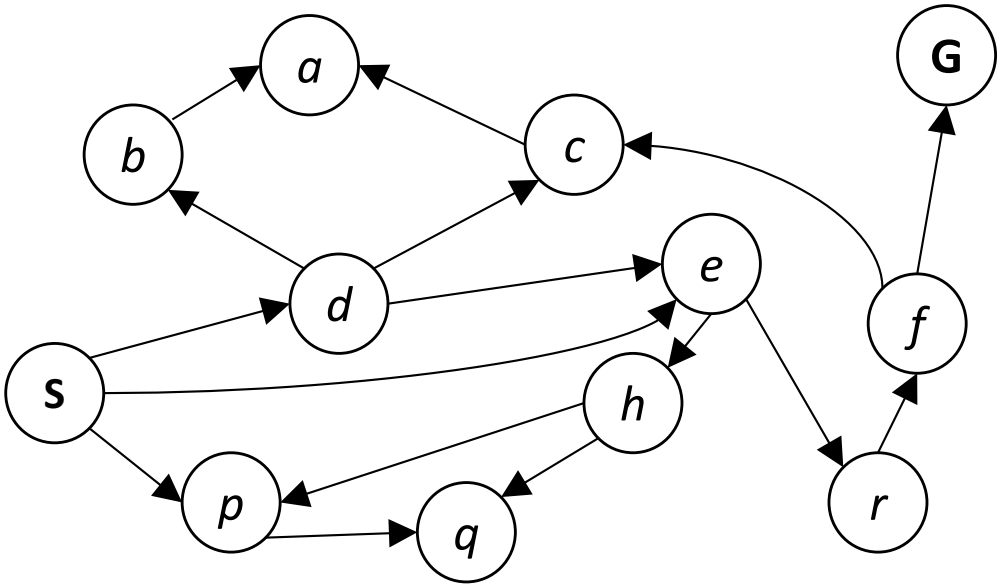
Together the initial state and successor function implicitly define the **state space**, the set of all reachable states

State Space Graph

- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent successors (action results)
  - The goal test is a set of goal node(s)

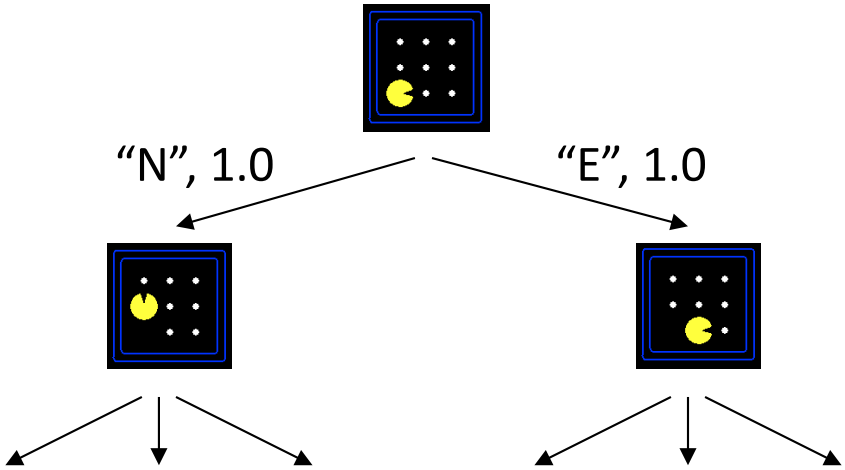
- In a search graph, each state occurs only once!

- **We can rarely build this full graph in memory (it's too big), but it's a useful idea**



Search Tree

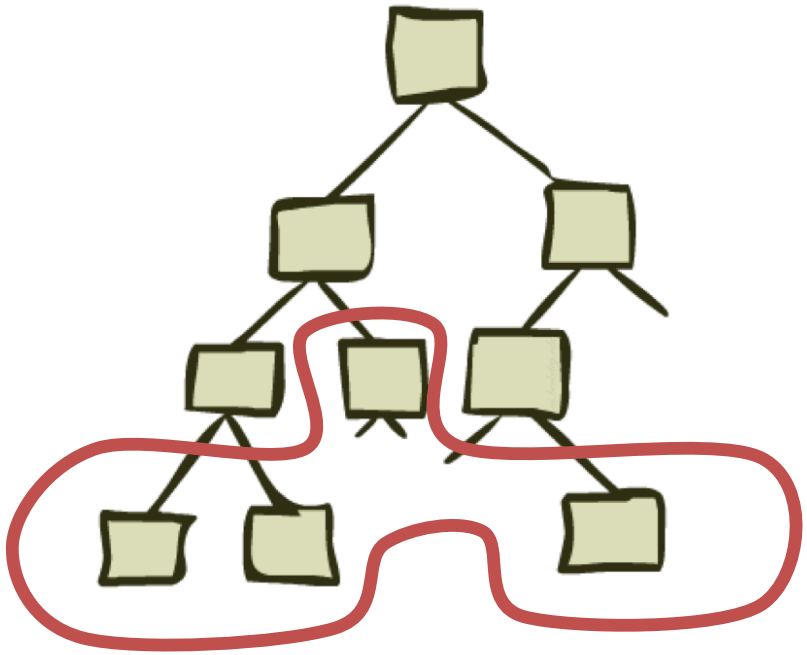
- A "what if" tree of plans and their outcomes
- The start state is the root node
- Children correspond to successors
- Nodes show states, but correspond to PLANS that achieve those states
- **For most problems, we can never actually build the whole tree**



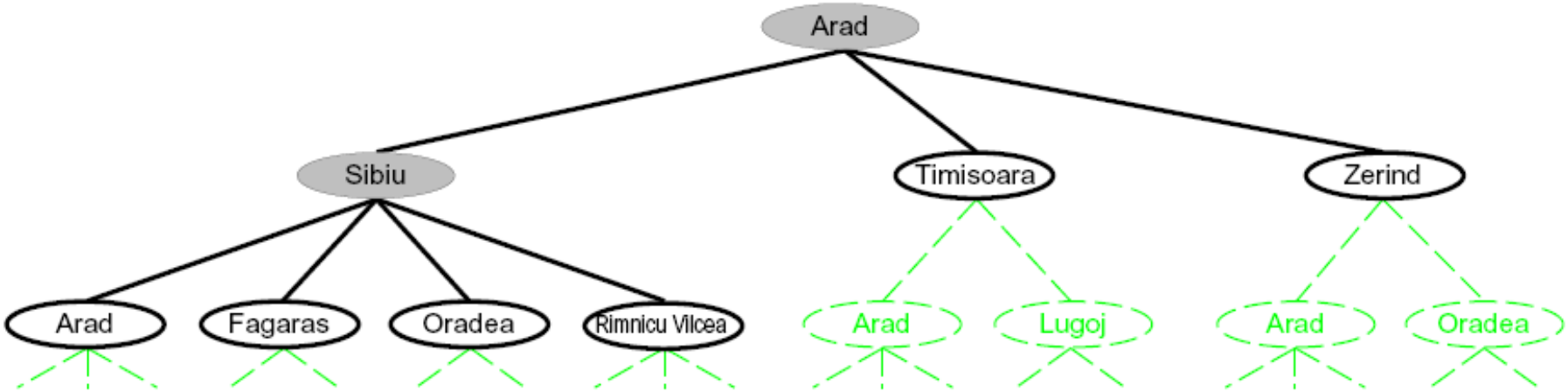
Searching for Solutions

Basic idea: incrementally build a search tree until a goal state is found

- Root = initial state
- Expand via transition function to create new nodes
- Nodes that haven't been expanded are **leaf nodes** and form the **frontier** (**open list**)
- Different **search strategies** (next lecture) choose next node to expand (as few as possible!)
- Use a **closed list** to prevent expanding the same state more than once

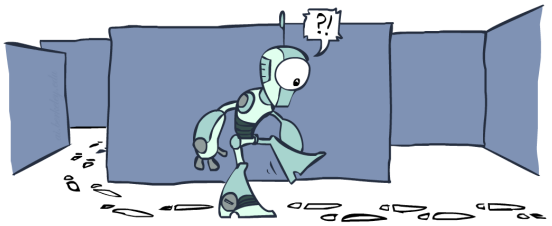


General Algorithm




```
function GRAPH-SEARCH( problem, fringe) returns 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
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
    end
```

Queue (FIFO)
Stack (LIFO)
Priority Queue
- **f(n)**



Evaluating a Search Strategy

Solution

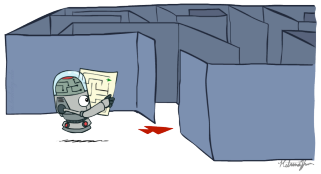
- **Completeness**: does it always find *a* solution if one exists?
- **Optimality**: does it always find a least-cost solution?

Efficiency

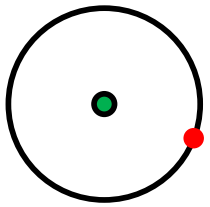
- **Time Complexity**: number of nodes generated/expanded
- **Space Complexity**: maximum number of nodes in memory

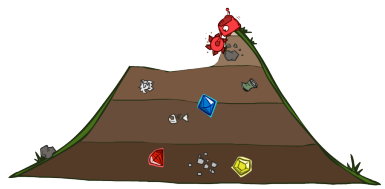
Uninformed Search

Search given **only** the problem definition

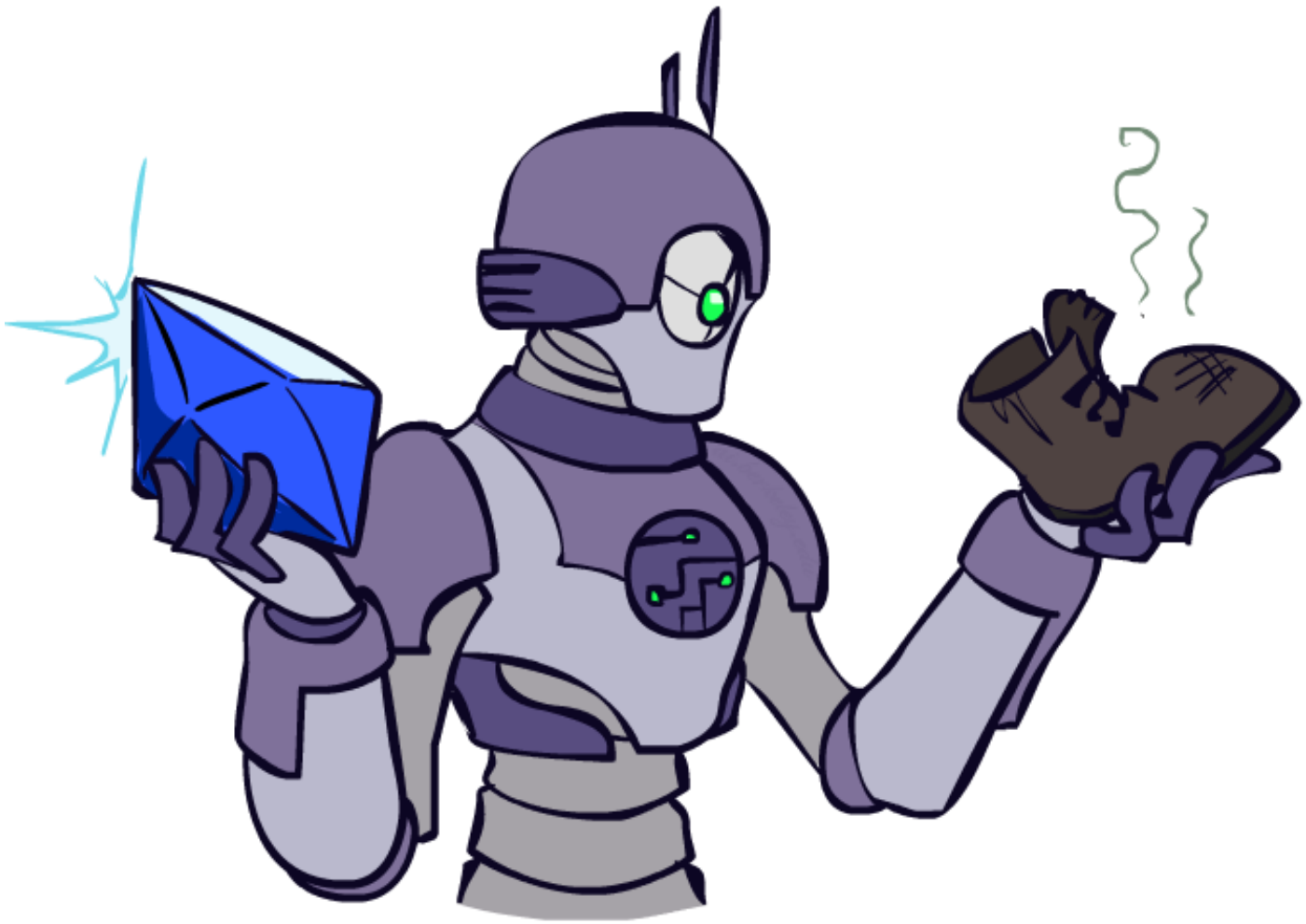


	DFS	BFS	UCS
Fringe	LIFO (stack)	FIFO (queue)	PQ (path cost)
Complete		X	X
Optimal			X
Time	$\mathcal{O}(b^m)$	$\mathcal{O}(b^s)$	$\mathcal{O}(b^{C^*/\varepsilon})$
Space	$\mathcal{O}(bm)$	$\mathcal{O}(b^s)$	$\mathcal{O}(b^{C^*/\varepsilon})$
Assumptions: potentially infinite depth, arbitrary positive action costs			





Let's **Inform** the Search

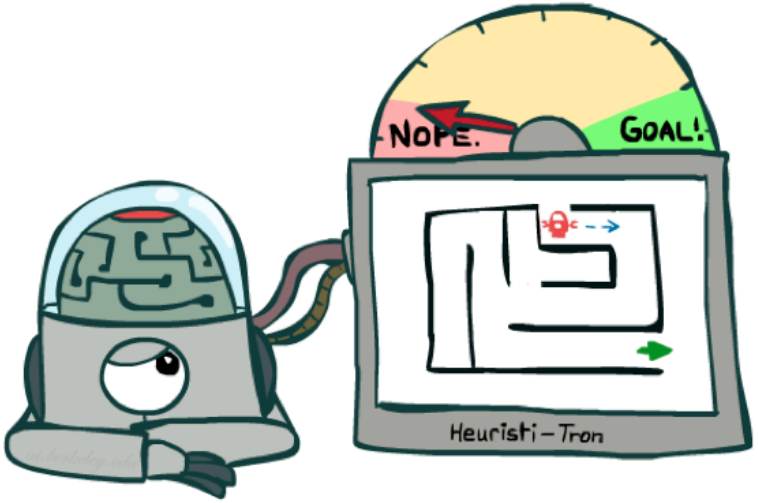


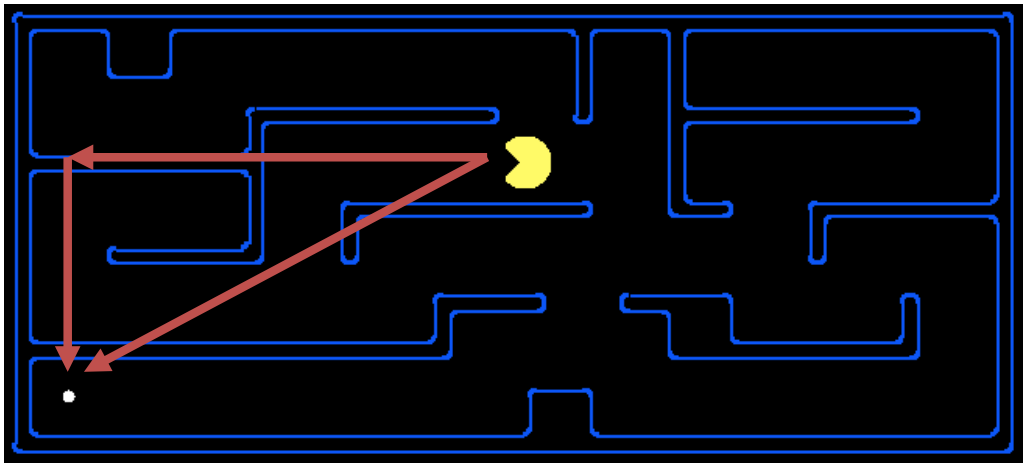
Search Heuristic

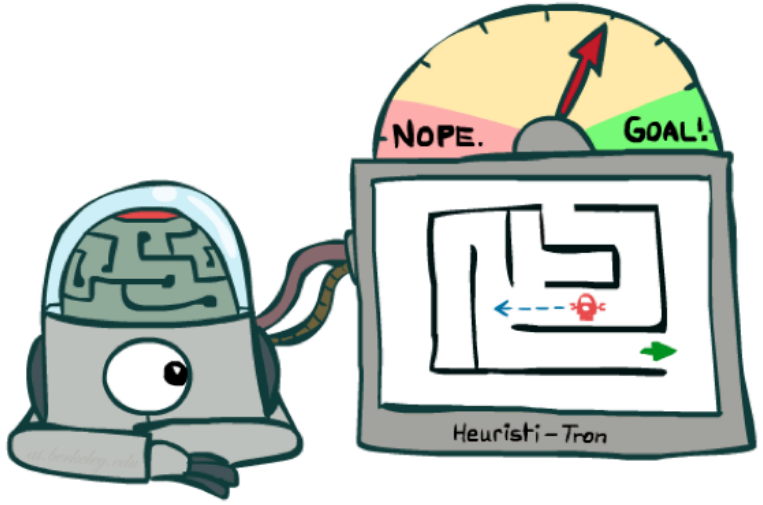
A[n] **heuristic** is…

- a function, $h(n)$, that estimates how close the input state is from a goal state
- designed for a particular problem

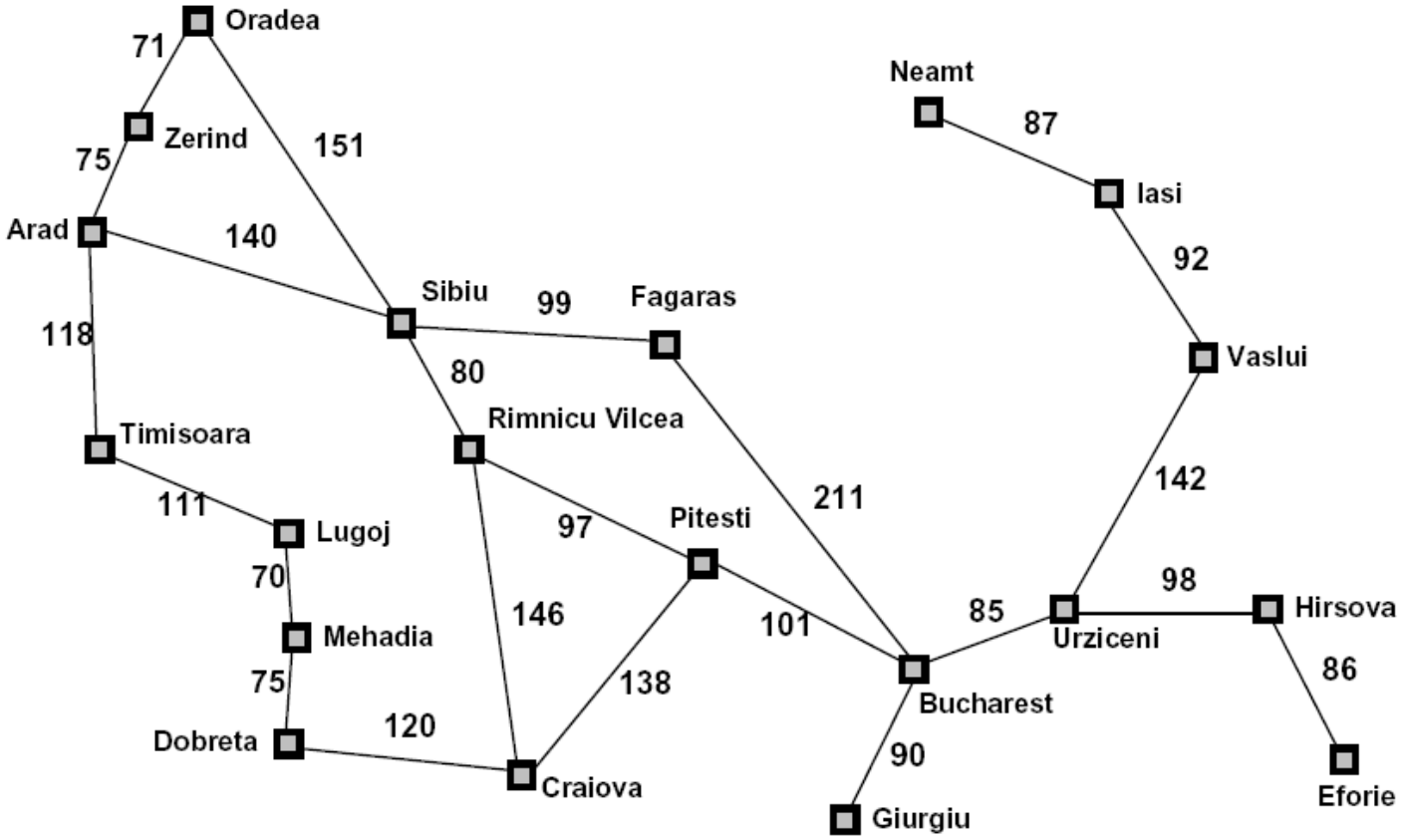
Examples: distance
(Manhattan, Euclidean)







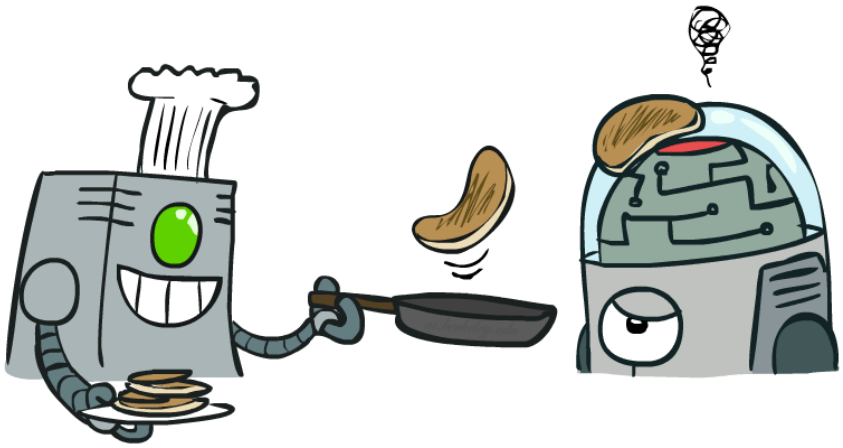
Example Search Heuristic



h(x)

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

The 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

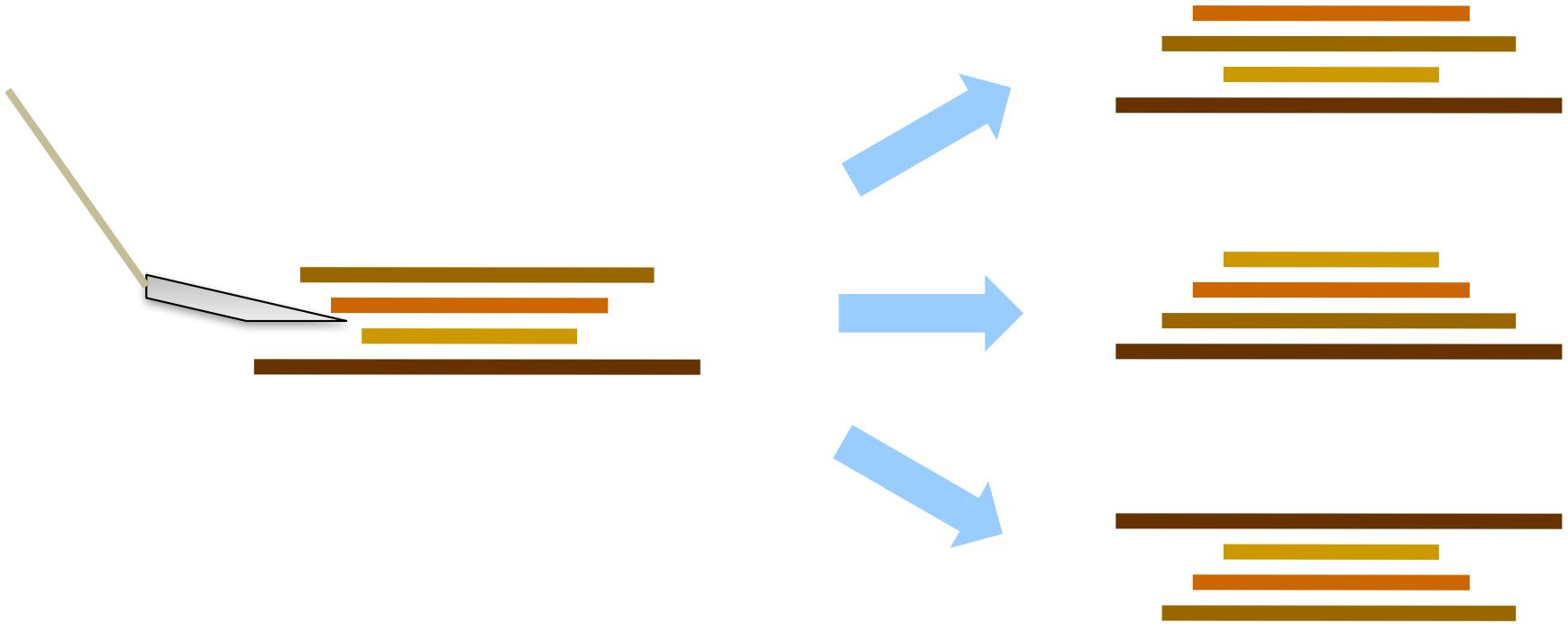
For a permutation σ of the integers from 1 to n, let $f(\sigma)$ be the smallest number of prefix reversals that will transform σ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all σ in (the symmetric group) S_n. We show that $f(n) \leqslant (5n+5)/3$, and that $f(n) \geqslant 17n/16$ 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 $3n/2-1 \leqslant g(n) \leqslant 2n+3$.

1. Introduction

We introduce our problem by the following quotation from [1]

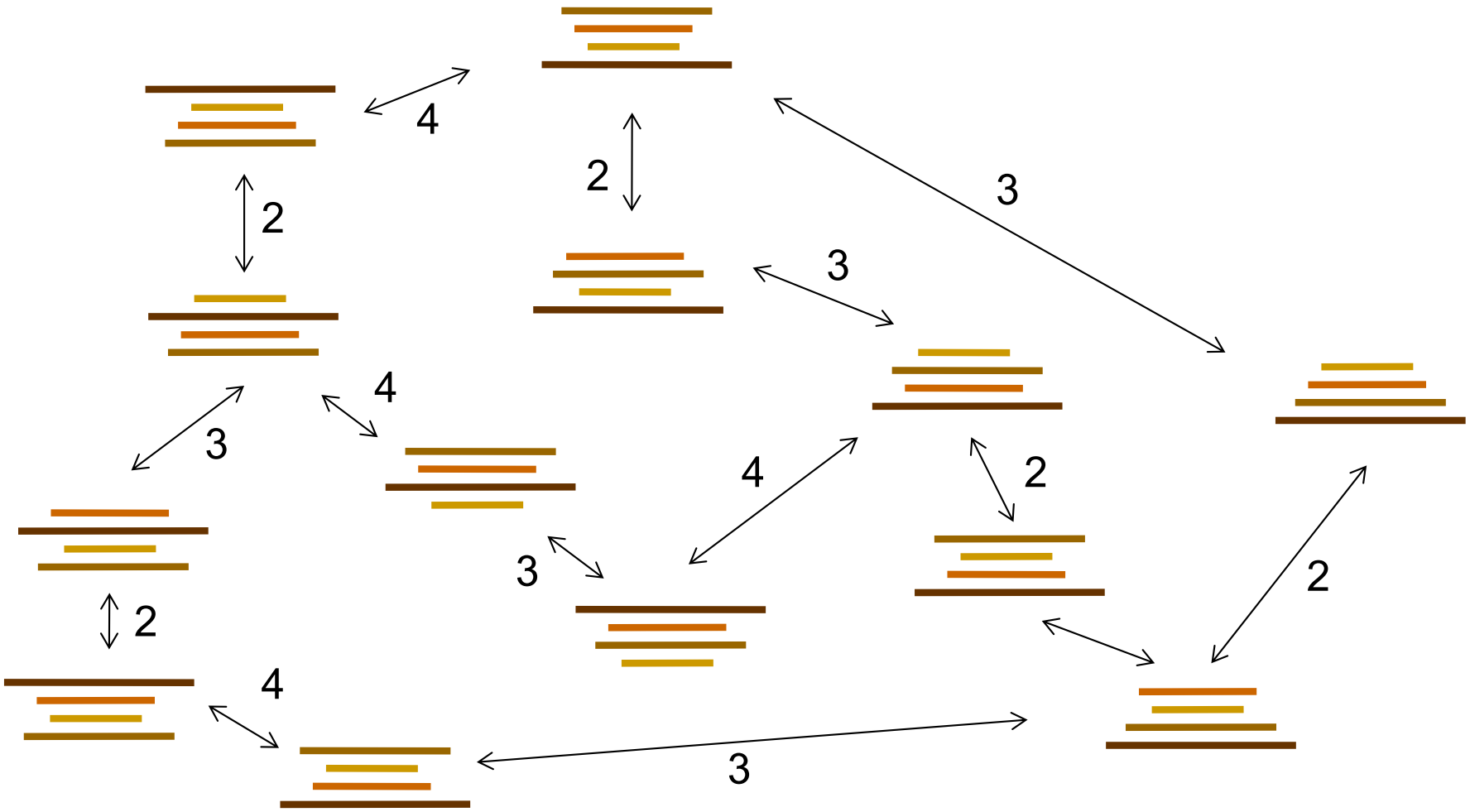
> The chef in our place is sloppy, and when he prepares a stack of pancakes they come out all different sizes. Therefore, when I deliver them to a customer, on the way to the table I rearrange them (so that the smallest winds up on top, and so on, down to the largest at the bottom) by grabbing several from the top and flipping them over, repeating this (varying the number I flip) as many times as necessary. If there are n pancakes, what is the maximum number of flips (as a function $f(n)$ of n) that I will ever have to use to rearrange them?

Pancake Problem

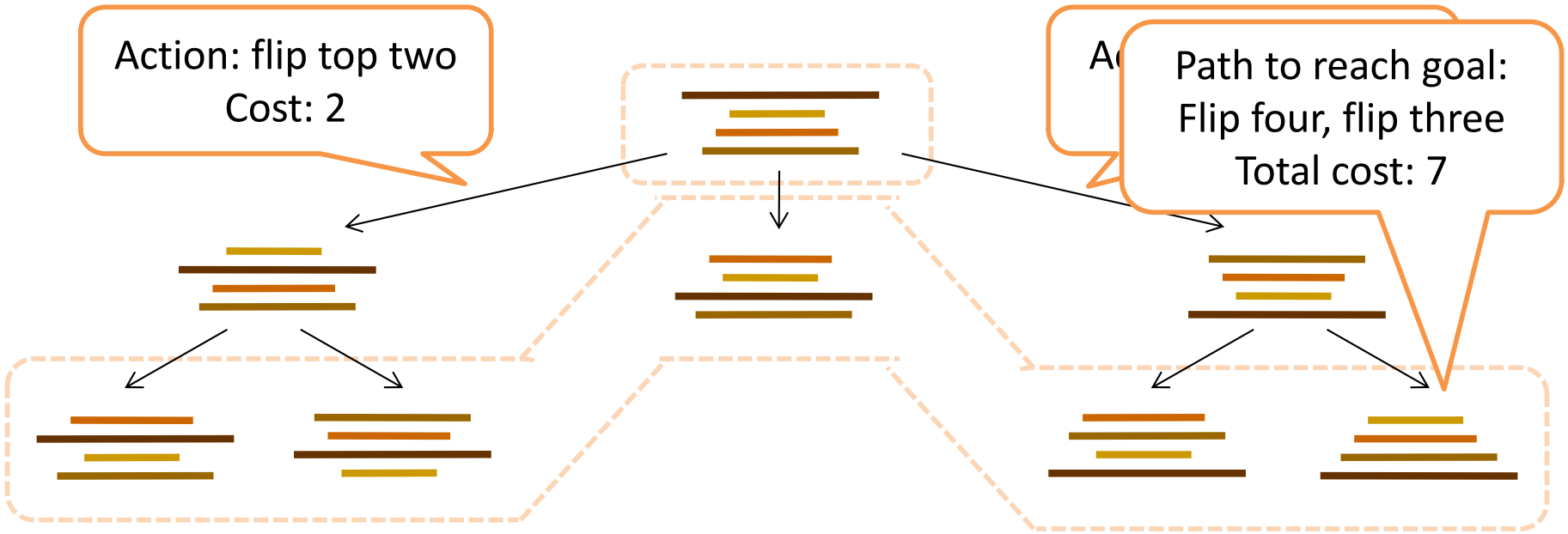


Cost: number of pancakes flipped

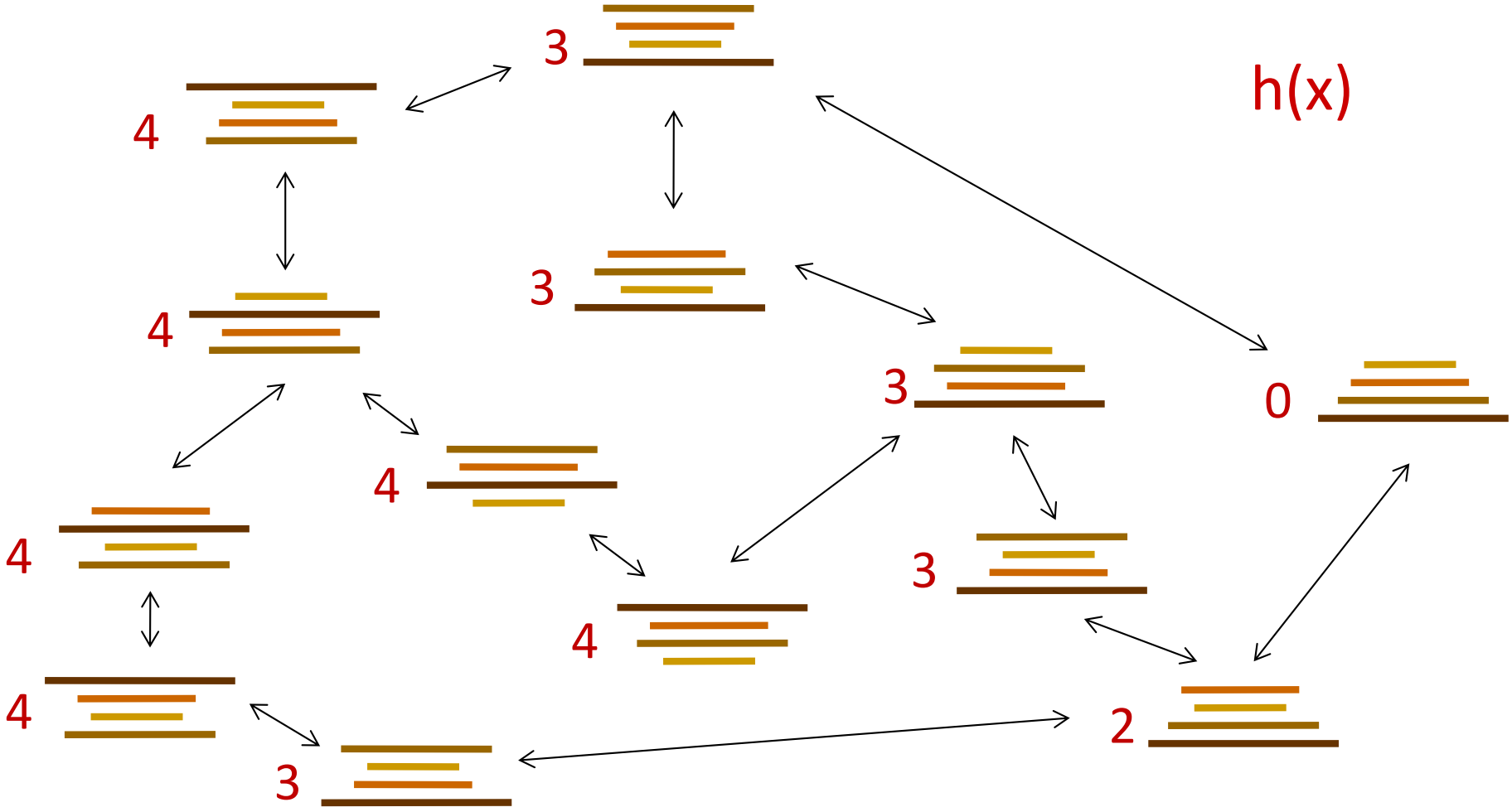
[Partial] State Space Graph



Example Graph Search



Example Search Heuristic (2)



Heuristic: number of the largest pancake out of place

Best-First Search

- In Graph Search, apply $f(n)$, an **evaluation function**, for each node
  - Estimates "desirability"

- Typical decomposition of $f(n)$
  - $g(n)$: cost to reach n from initial state
  - $h(n)$: heuristic function
    - Estimate of distance to goal

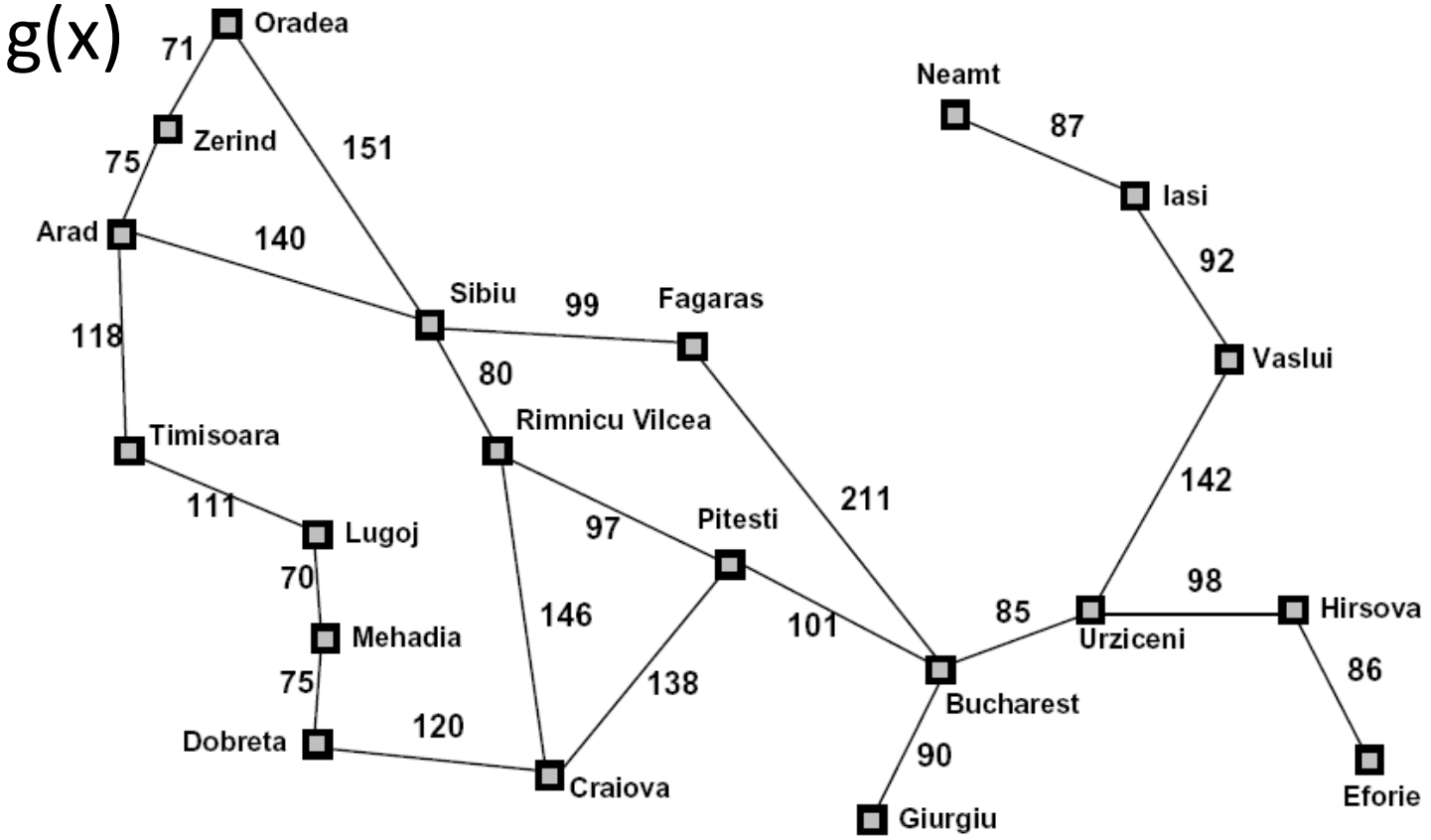
Special cases

- Uniform-Cost Search (**UCS**): $f(n) = g(n)$
- **Greedy** Best-First Search: $f(n) = h(n)$
- **A*** Search: $f(n) = g(n) + h(n)$

Greedy Best-First Search

Target Problem

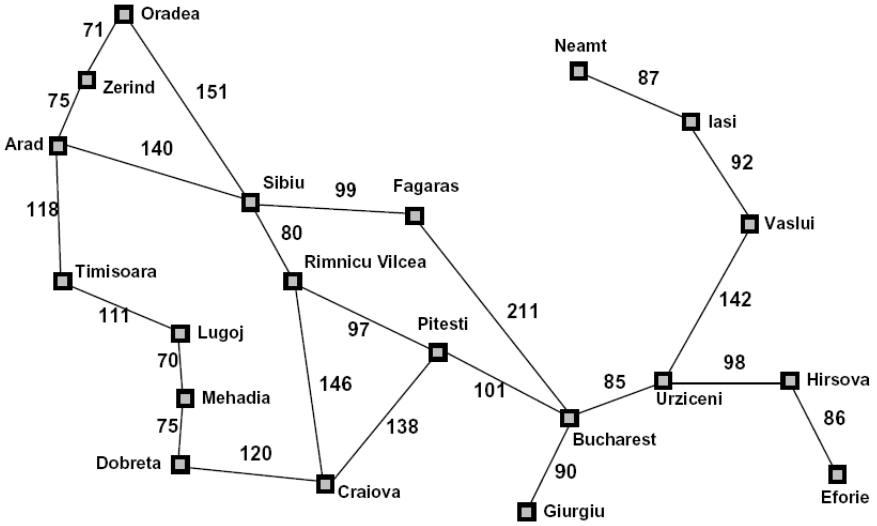
g(x)



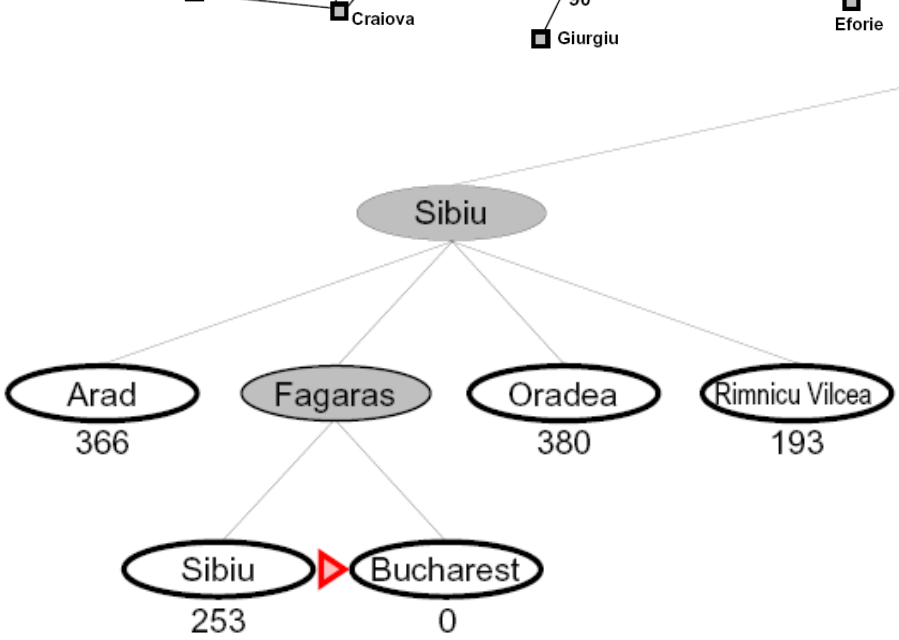
h(x)

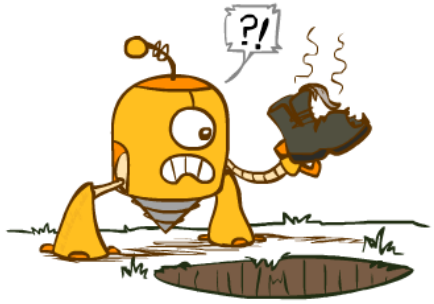
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Best-First Search



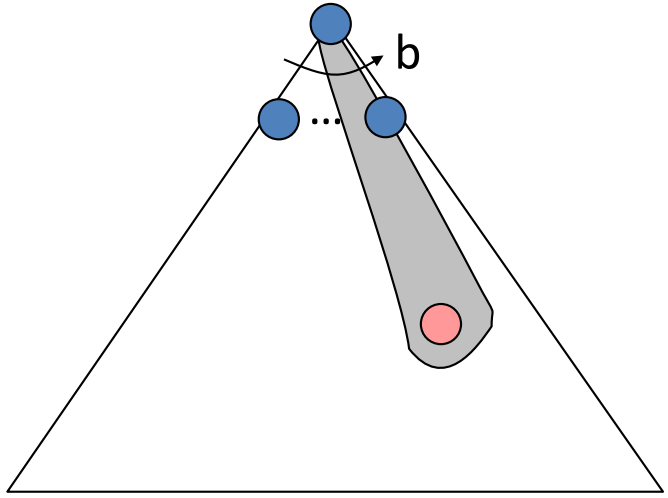
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

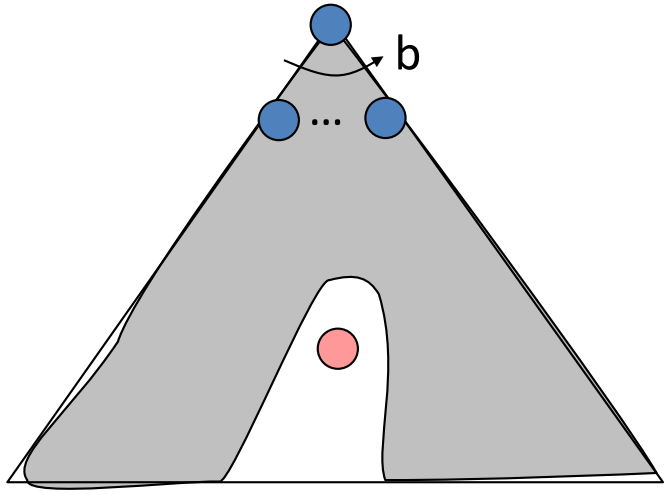




Greedy Analysis

- Best case
  - Take you directly to the nearest goal
- Common case
  - Takes you [less-than-optimally] to a goal
- Worst case
  - Like badly guided DFS

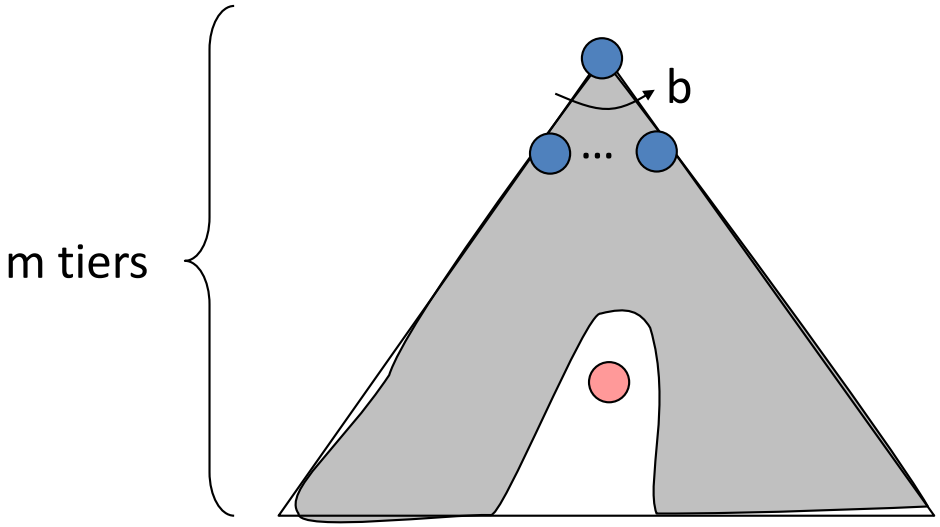




Greedy Evaluation

Time

- $\mathcal{O}(b^m)$
  - Can improve dramatically with a good heuristic



Space

- $\mathcal{O}(b^m)$

Complete

- Only if finite

Optimal

- No

Empty-Greedy

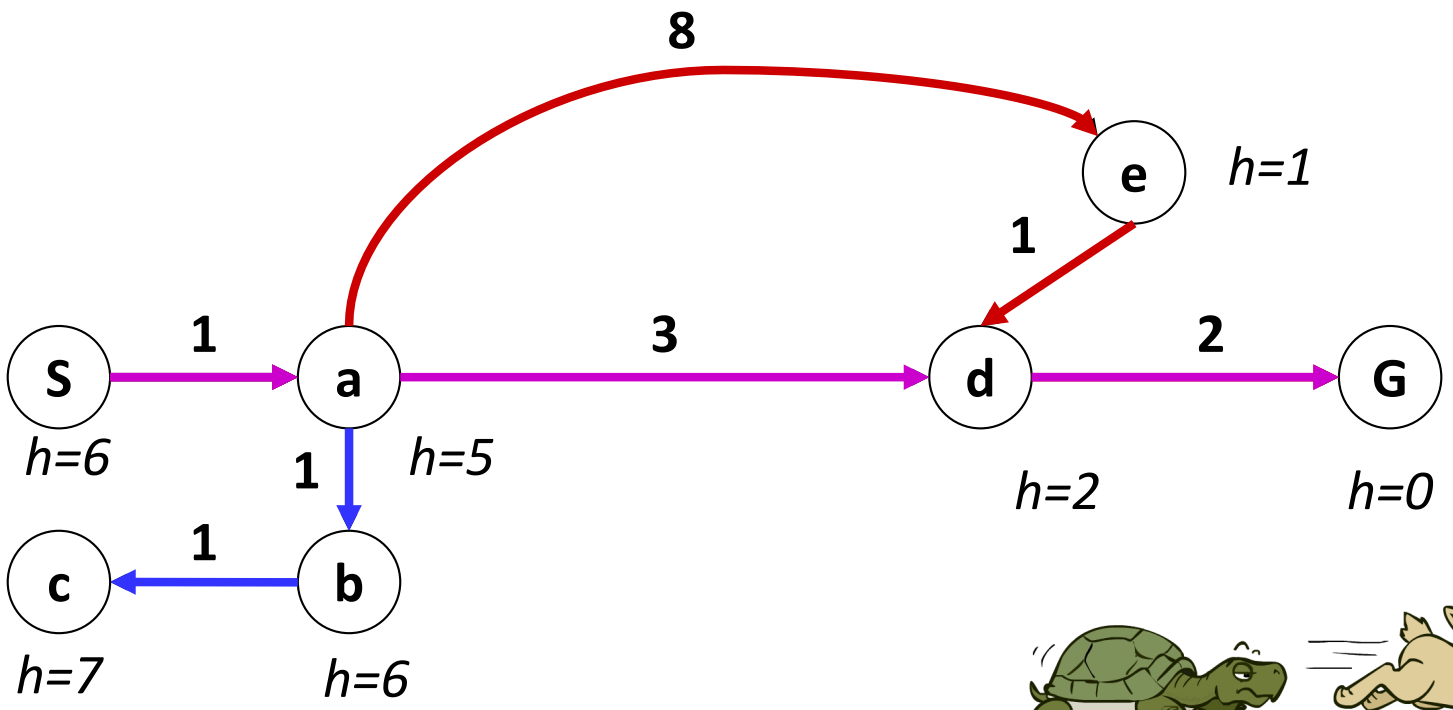
Maze-Greedy

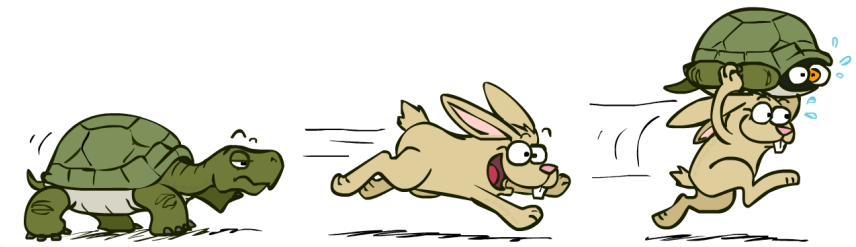
PacMan-Greedy

A* Search

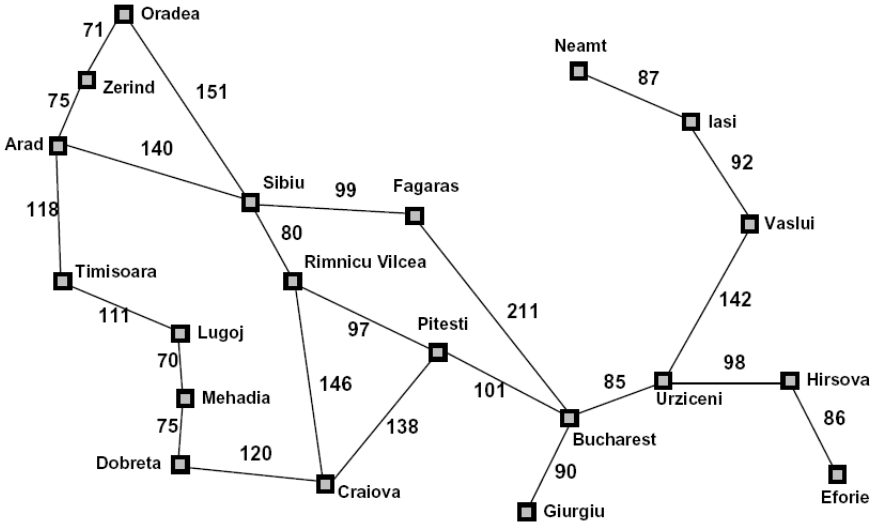
Combining UCS and Greedy

- UCS: orders by path, or backward, cost, $g(n)$
- Greedy: orders by goal proximity, or forward cost, $h(n)$
- A*: orders by the sum: $f(n) = g(n) + h(n)$

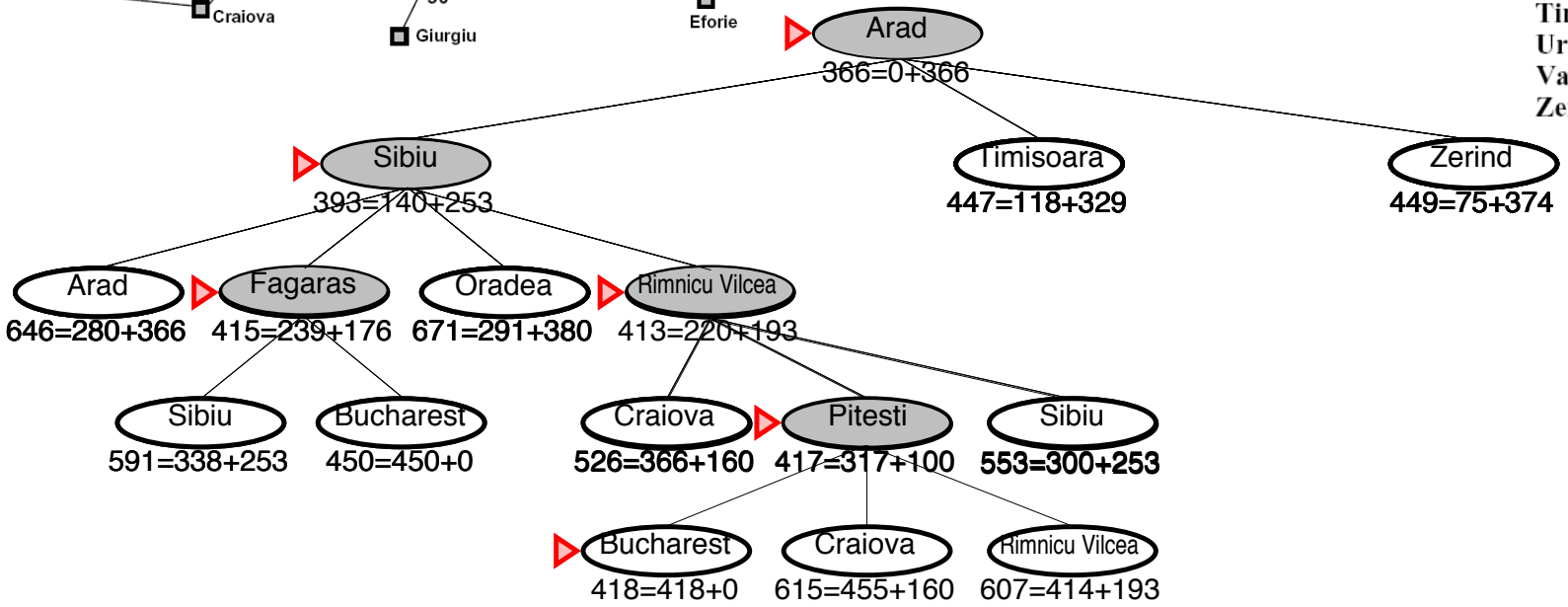




A* Search

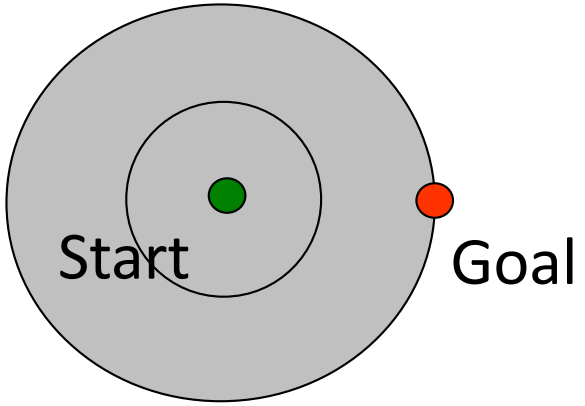


Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

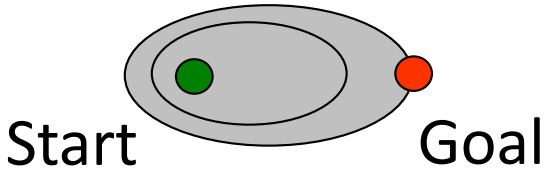


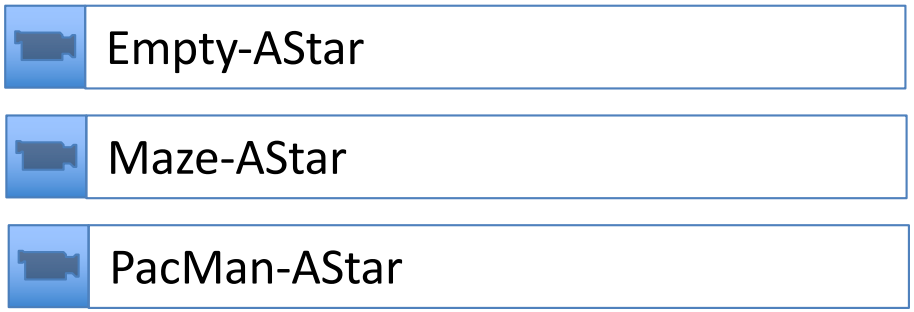
UCS vs. A*

- Uniform-cost expands equally in all "directions"

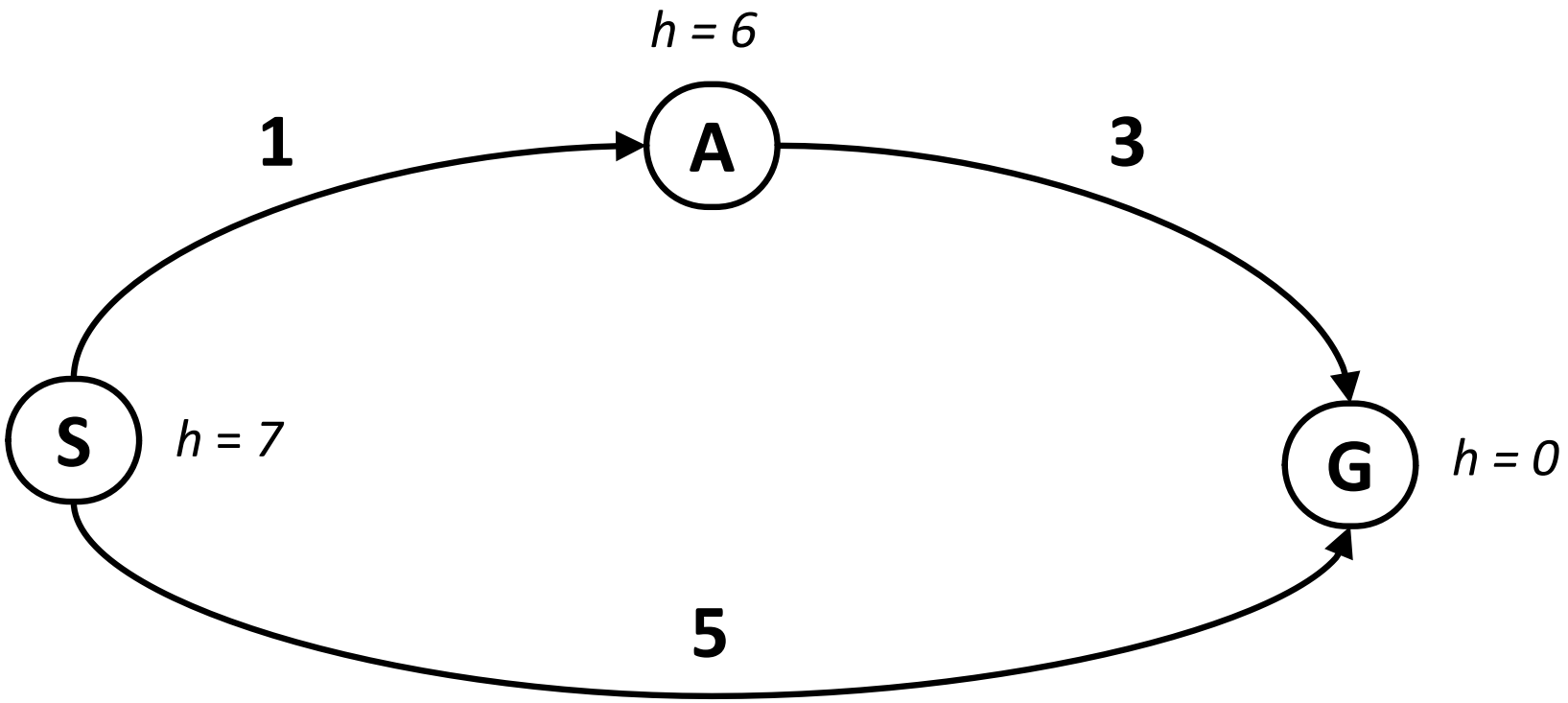


- A* expands mainly toward the goal, but does hedge its bets to ensure optimality





Is A* Optimal?



To ensure optimality, we need certain *properties* to hold on the heuristic function

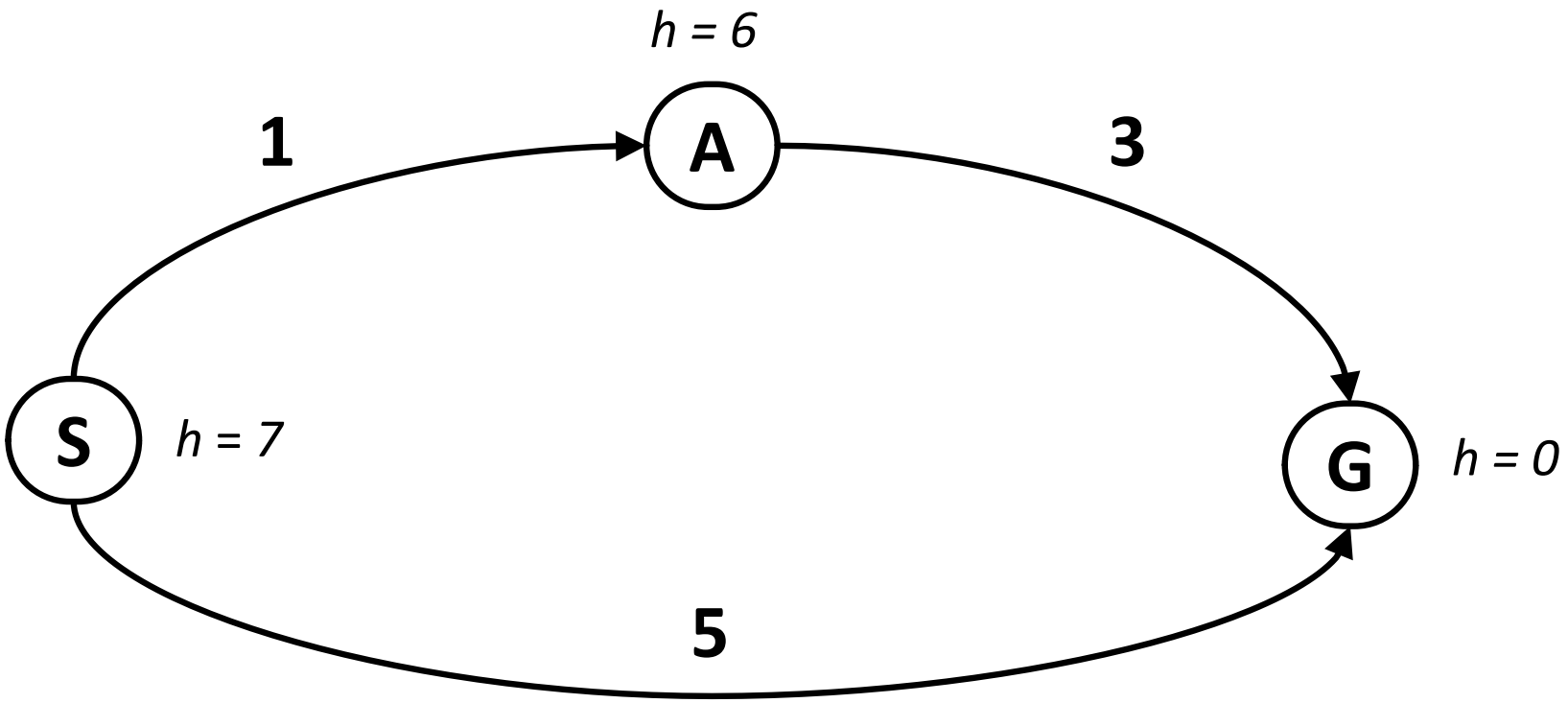
Property #1: Admissibility

- An **admissible** heuristic never overestimates cost to the goal

- Consequence: $f(n)$ never overestimates
  - $f(n) = g(n) + h(n)$
  - $g(n)$ reflects *actual* path cost

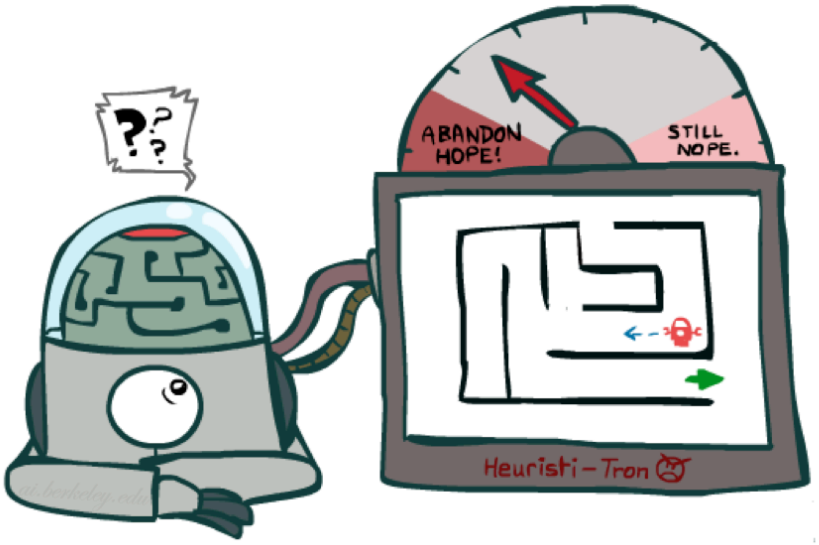
- Example: straight-line distance (SLD)

Issue Revisited

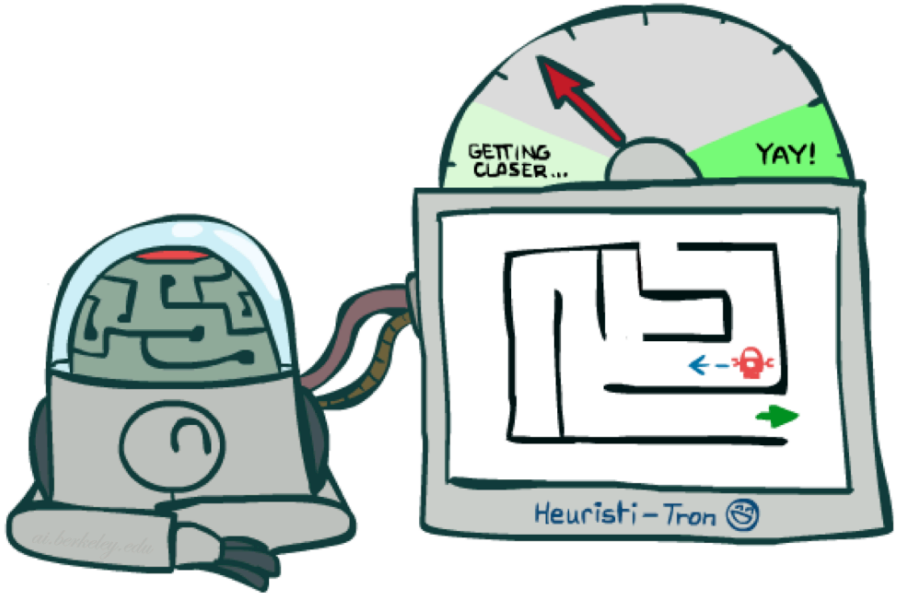


- Is $\mathrm{h}(n)$ admissible?

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

Checkup

For an admissible heuristic, what must the value of $h(g)$ be, if g is a goal?

In general…

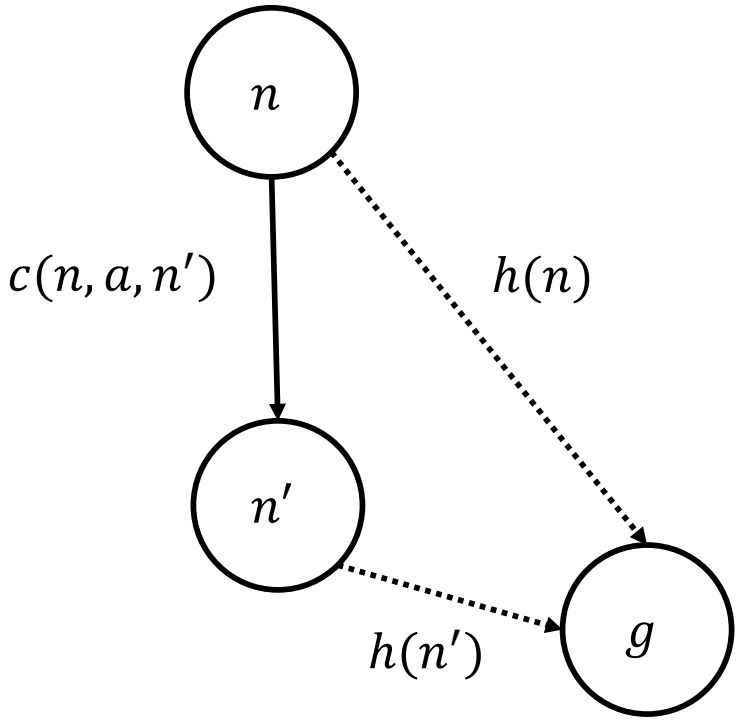
$$0 \leq h(n) \leq h^*(n)$$

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

Property #2: Consistency/Monotonicity

A[n] heuristic is **consistent** iff…

- for every node n,
- and every successor n' of n generated via action a,
- the estimated cost of reaching the goal from n is no greater than the step cost of getting to n plus the estimated cost of reaching the goal from n'

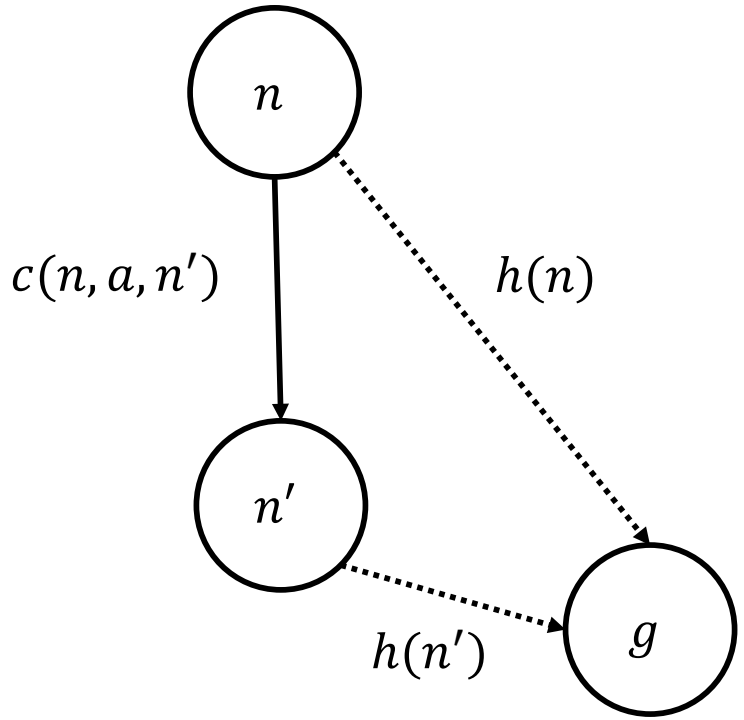


$$h(n) \leq c(n, a, n') + h(n')$$

Implication of Consistency

$f(n)$ Monotonicity

If a[n] heuristic is consistent, $f(n)$ **never decreases along a path**



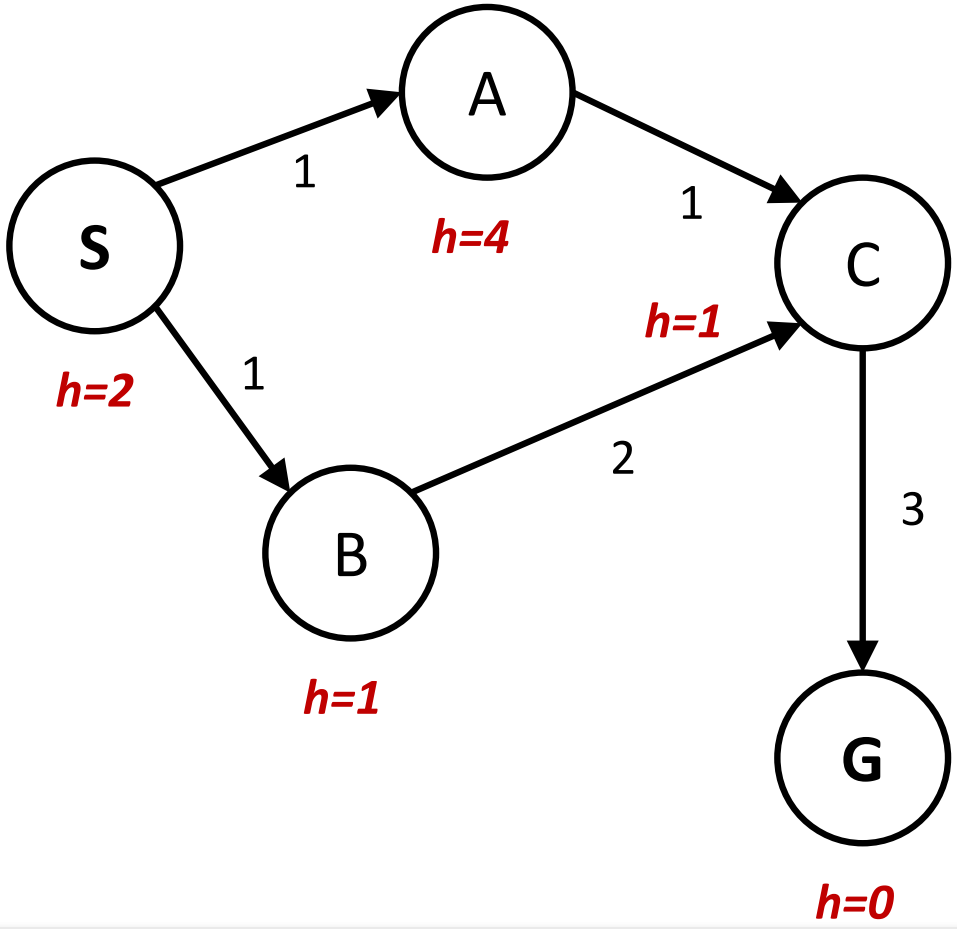
Proof… $f(n')$

$$= g(n') + h(n')$$
$$= g(n) + c(n,a,n') + h(n')$$
$$\geq g(n) + h(n) = f(n)$$

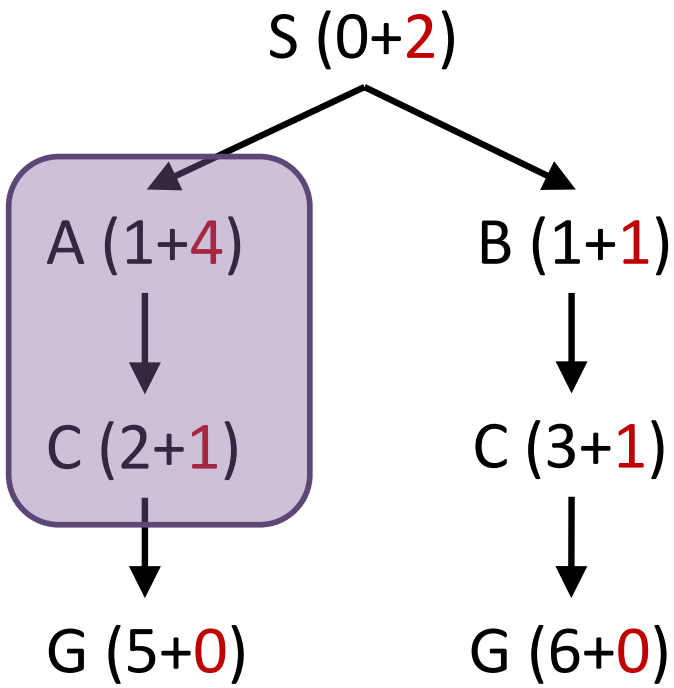
$$h(n) \leq c(n,a,n') + h(n')$$

Checkup

State Space Graph

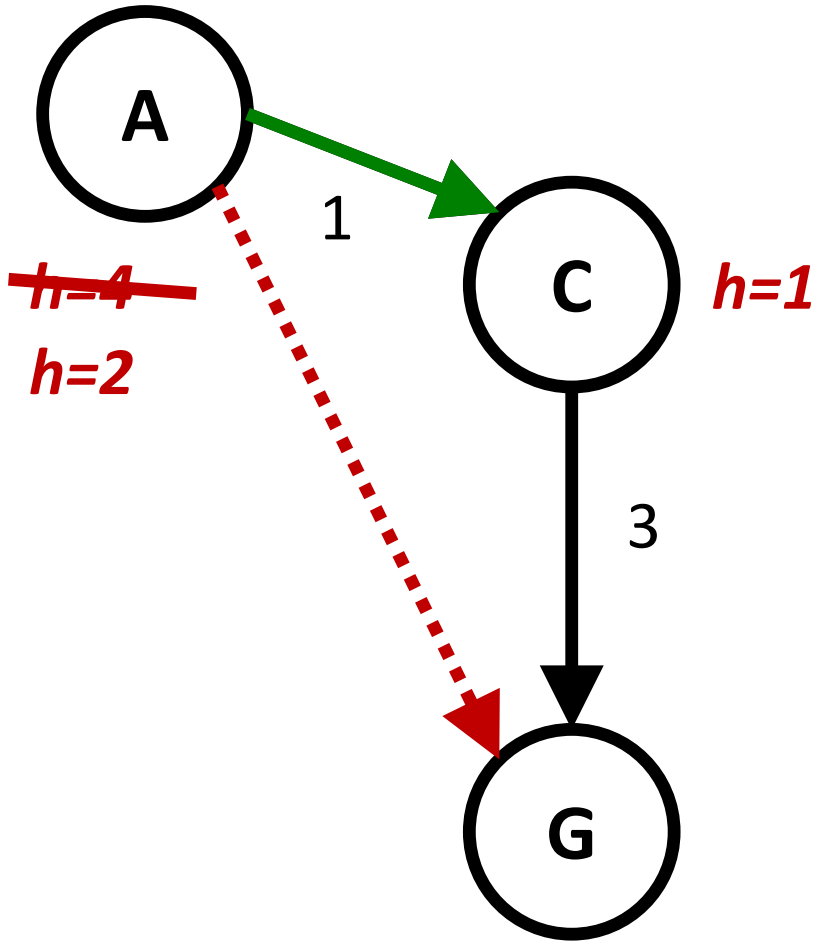


Search Tree



- Admissible?
- Consistent?

Consistency of Heuristics



Admissibility

- Heuristic ≤ actual cost

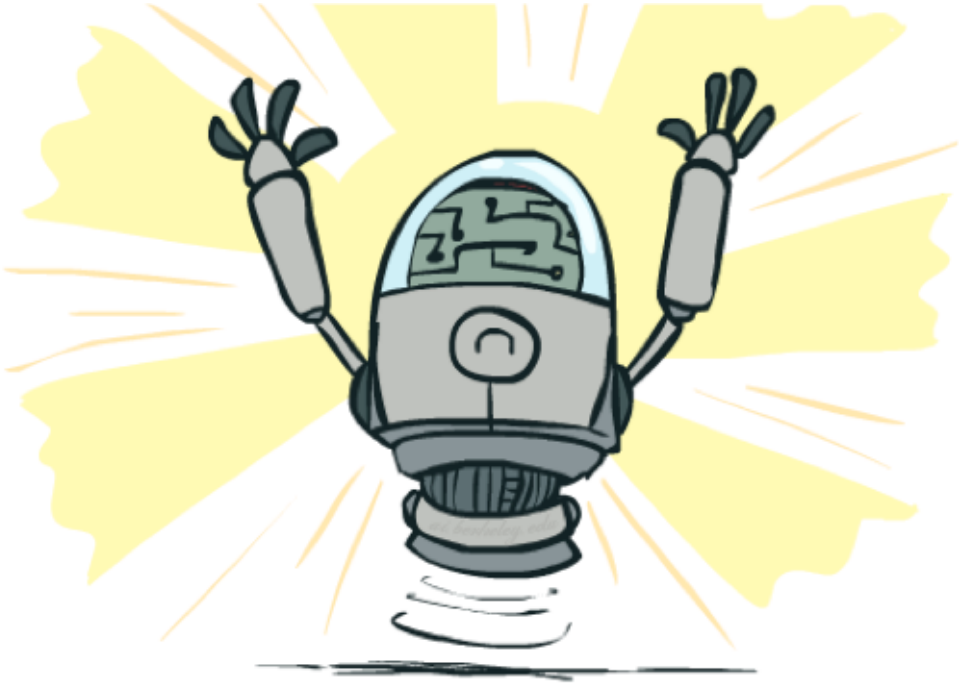
$$h(A) \leq c(A, G)$$

Consistency

- Heuristic ≤ actual arc

$$h(A) - h(C) \leq c(A, C)$$

Optimality of A* Graph Search

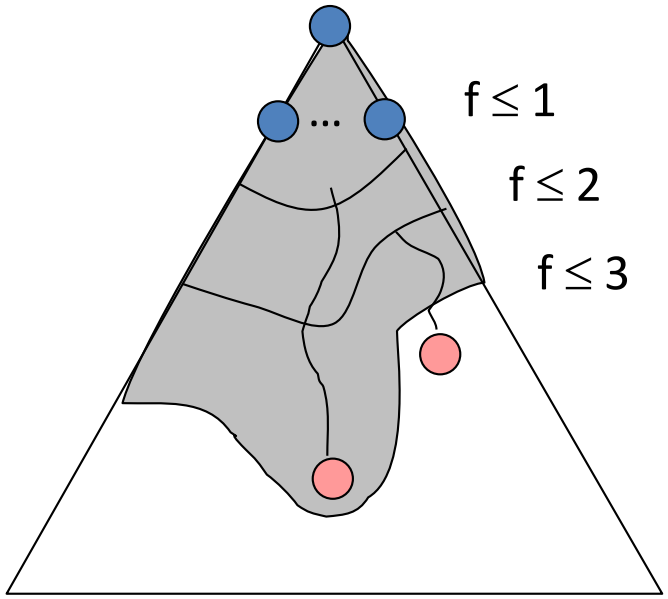


- If $h(n)$ is consistent, A* Graph Search is **optimal**
- All consistent heuristics are admissible
  - Most admissible heuristics are also consistent
  - Most of your time applying A* is finding the right $h(n)$

Proof Ideas

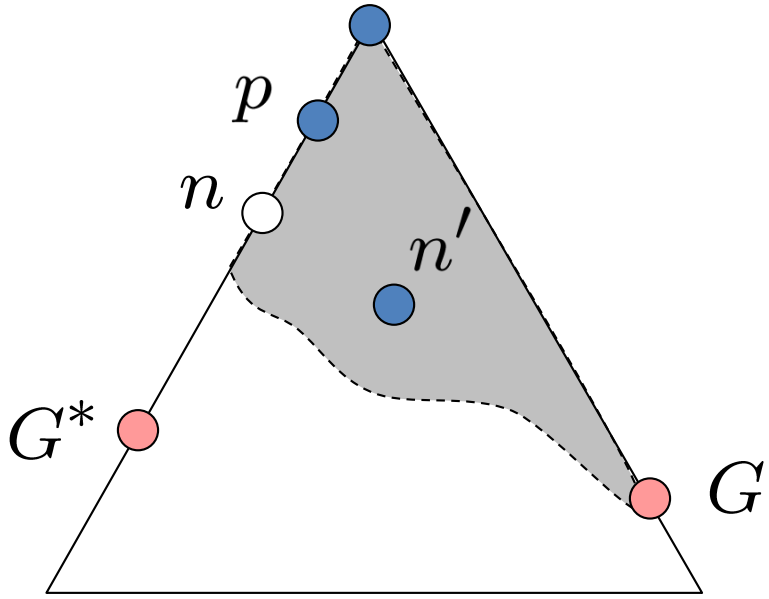
Let's assume positive action costs

1. A* expands nodes in increasing total f value (*f-contours*)
   - Lowest cost gets popped from the fringe
   - Consistency implies path monotonicity
2. The optimal goal(s) have the lowest f value, so it must get expanded first



A* Optimality Proof

1. Assume some n on path to G^* isn't in the fringe when we need it, b/c some worse n' for the same state popped and expanded first
2. Let p be the ancestor of n that was on the queue when n' was popped
3. $f(p) < f(n)$: consistency
4. $f(n) < f(n')$: n' suboptimality
5. $\therefore p$ would have been expanded before n': **contradiction!**



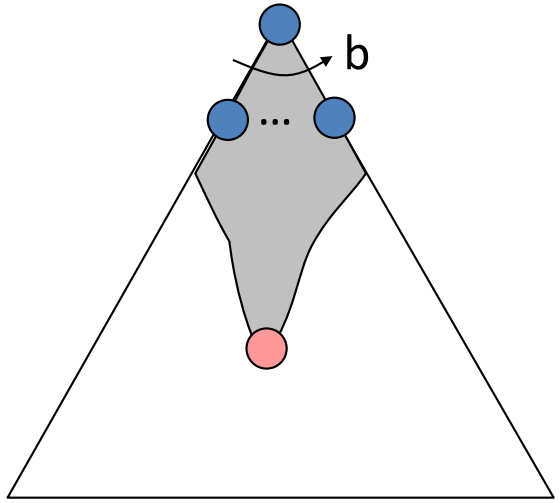
A* Evaluation

Time
- Strongly related to the problem and heuristic; exponential w.r.t. error/solution length

Space
- Whole tree (see book for ID-A, RBFS, SMA*)

Complete
- Yes, assuming positive action costs

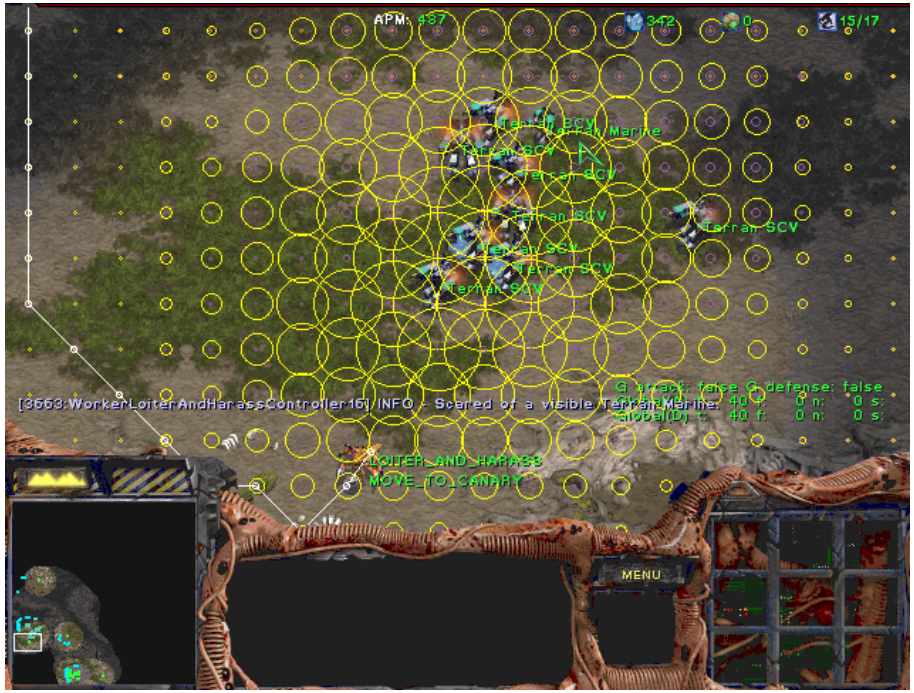
Optimal
- Yes, assuming consistent heuristic



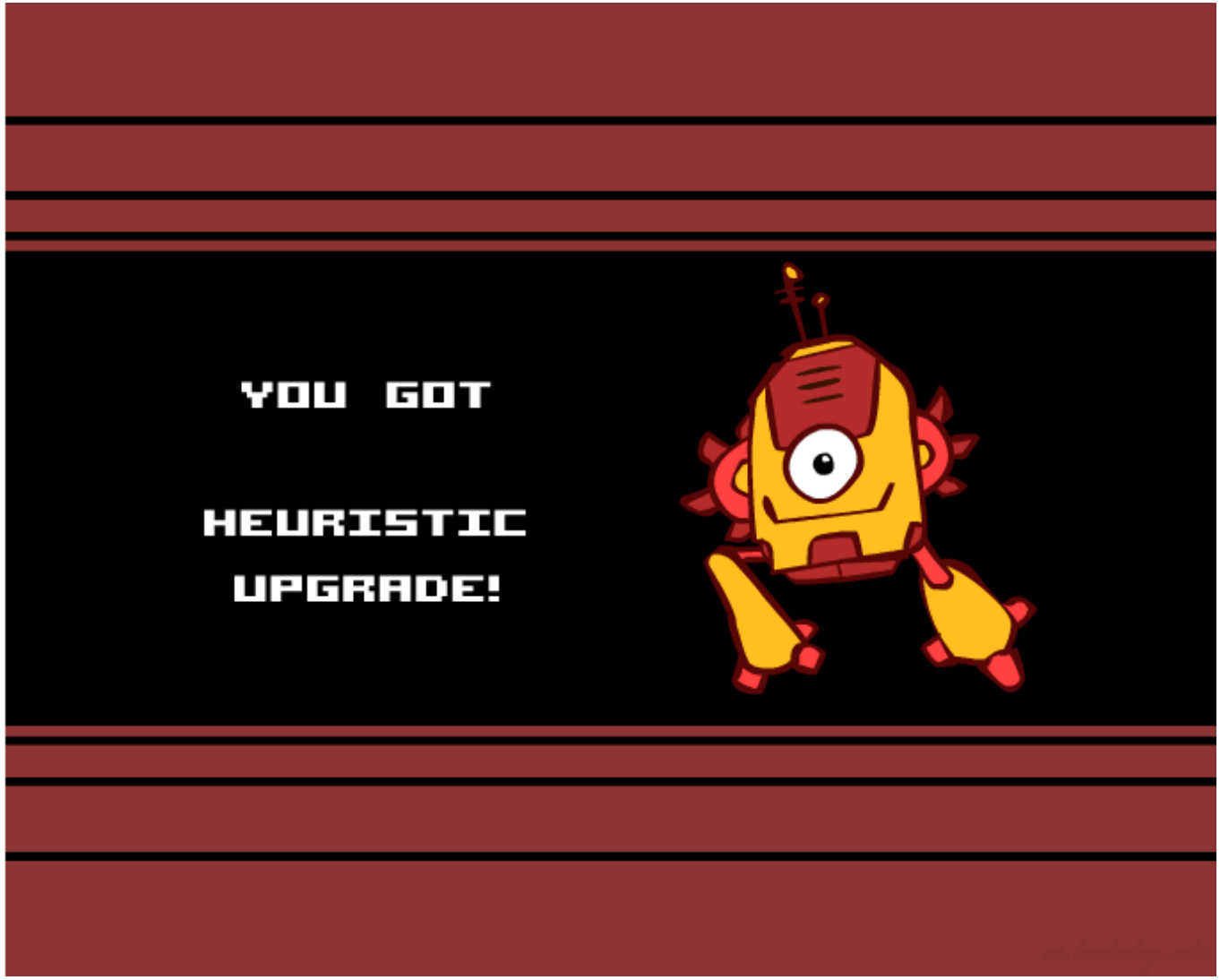
- A* expands all nodes with $f(n) < C^*$
- A* expands some nodes with $f(n) = C^*$
- A* expands **no** nodes with $f(n) > C^*$

A* Applications

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

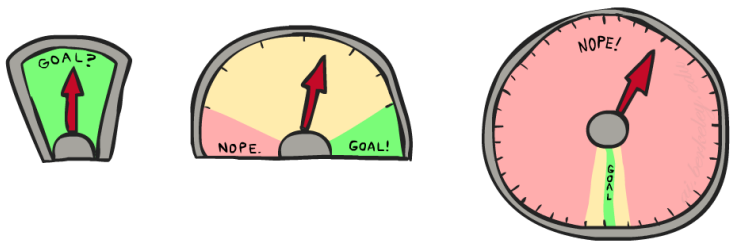


Creating Heuristics

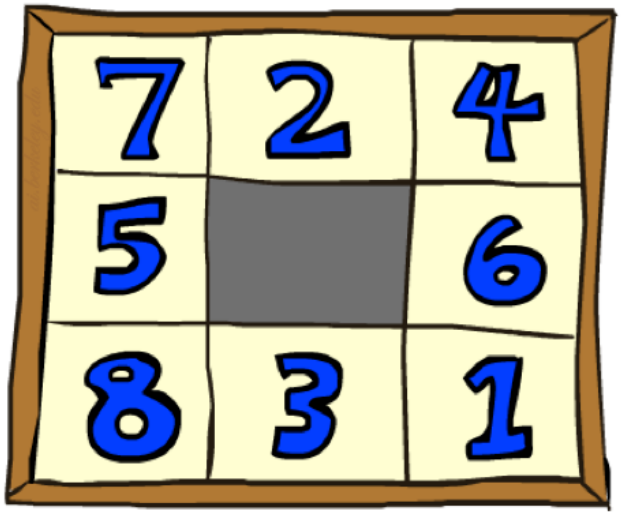


Heuristics Tradeoff

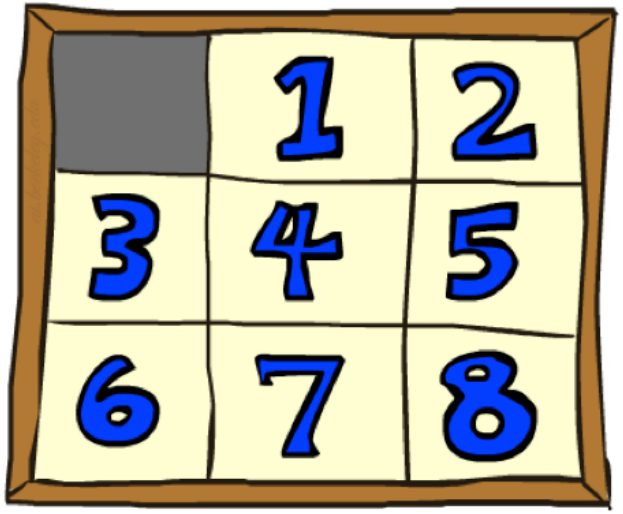
- There is a trade-off between quality of the goal estimate and the expended work per node to generate the estimate
- As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself
  - As an extreme, consider using true cost
- Often, heuristics are solutions to **relaxed** (i.e. easier) problems, where new actions are available
  - Key point: the optimal solution cost of the relaxed problem must not over-estimate the optimal solution cost of the real problem



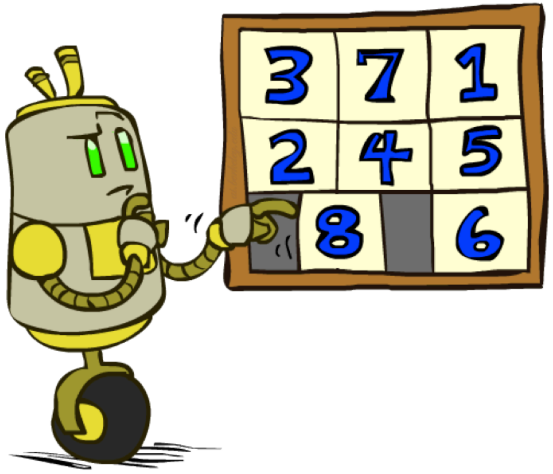
Example: 8 Puzzle



Start State

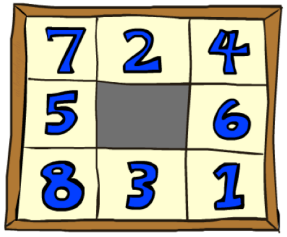


Goal State

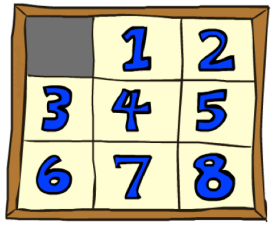


Relaxed Variant #1

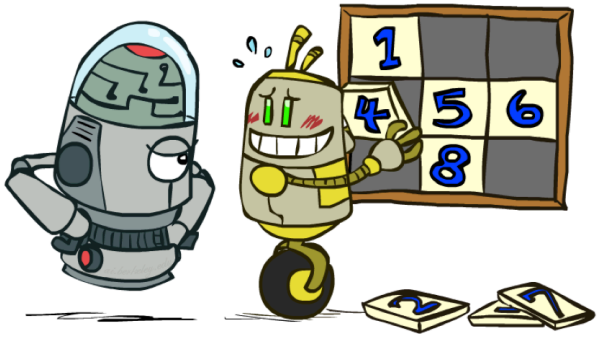
- Assume you can move tiles anywhere
- Heuristic: number of tiles misplaced
  - h(start) = 8
- Argue for consistency



Start State



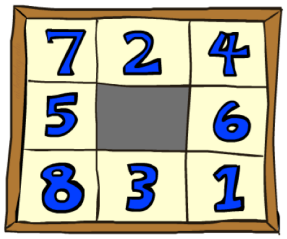
Goal State



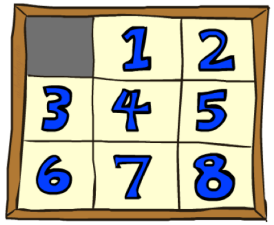
	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

Relaxed Variant #2

- Assume you can move tiles in any direction independent of other tiles
- Heuristic: total Manhattan distance
  - h(start)=3+1+…=18
- Argue for consistency



Start State



Goal State

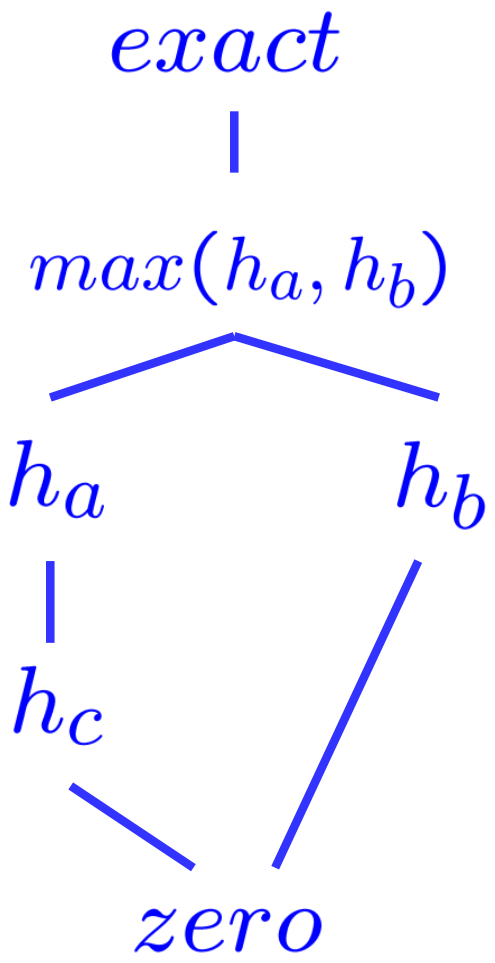
	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
TILES	13	39	227
MANHATTAN	12	25	73

Heuristic Dominance

- Assume <u>consistent</u> heuristics: $h_1, h_2, \ldots$
- $\forall n\ h_1(n) \geq h_2(n) \equiv h_1$ **dominates** h_2
  - h_1 is better for search
- If no heuristic dominates in all states, consider $h_{max}(n) = \max\{h_1(n), h_2(n), \ldots\}$
  - h_{max} is consistent
  - h_{max} is a dominant heuristic over the set

Heuristic Hierarchy

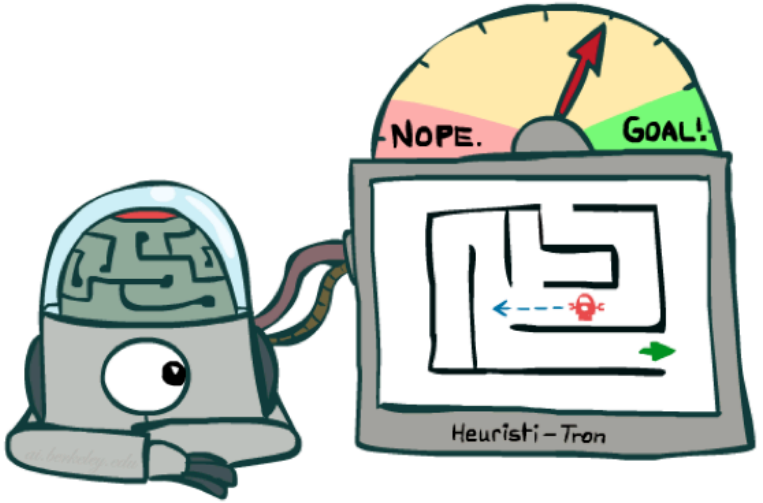
- Bottom is the **trivial**, or zero, heuristic
  - $\forall n\ h(n) = 0$
- Top is the exact solution



Summary (1)

- Heuristics are problem-specific functions that estimate how close the input state is from a goal state

- Good heuristics can *dramatically* reduce search cost



Summary (2)

- Best-first search uses an evaluation function
  - UCS: $g(n)$
  - Greedy: $h(n)$
  - A*: $g(n) + h(n)$
- A* is complete and optimal given a consistent heuristic
- Consistent heuristics can be derived from solutions to relaxed problems
  - Exploiting dominance can lead to even better heuristics