Informed Search

Lecture 4, CMSC 170

John Roy Daradal / Instructor

Previously on CMSC 170

- Search Problems
- Tree Search
- Uninformed Search
  - Depth-First Search
  - Breadth-First Search
  - Uniform Cost Search

Today's Topics

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

Review: Search Problem

- States
- Actions and costs
- Successor function
- Start state
- Goal test

Review: Search Tree

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

Review: Search Algorithm

- Systematically builds a search tree
- Chooses **ordering** of the *fringe*
- **DFS** (stack), **BFS** (queue), **UCS** (PQ)
- **Optimal** = finds least-cost plans

Review: Uninformed Search

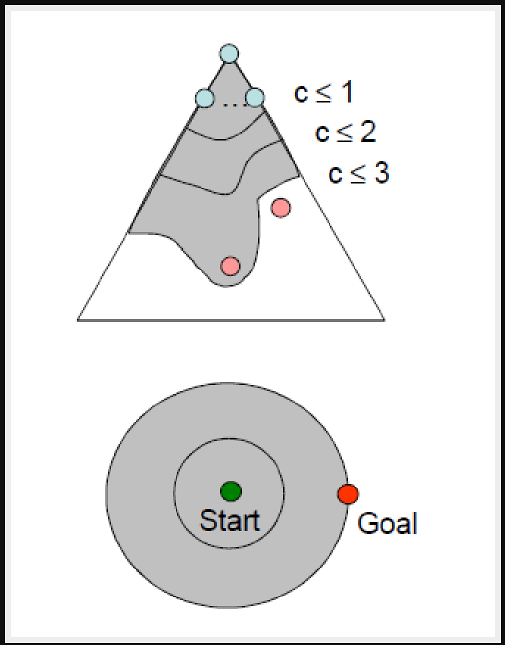
- Explore tree in **systematic** way
- **DFS** (leftmost, deepest), **BFS** (per-layer), **UCS** (least-costly)
- **Limitation**: *unaware* where the **goals** are

Review: Uniform Cost Search

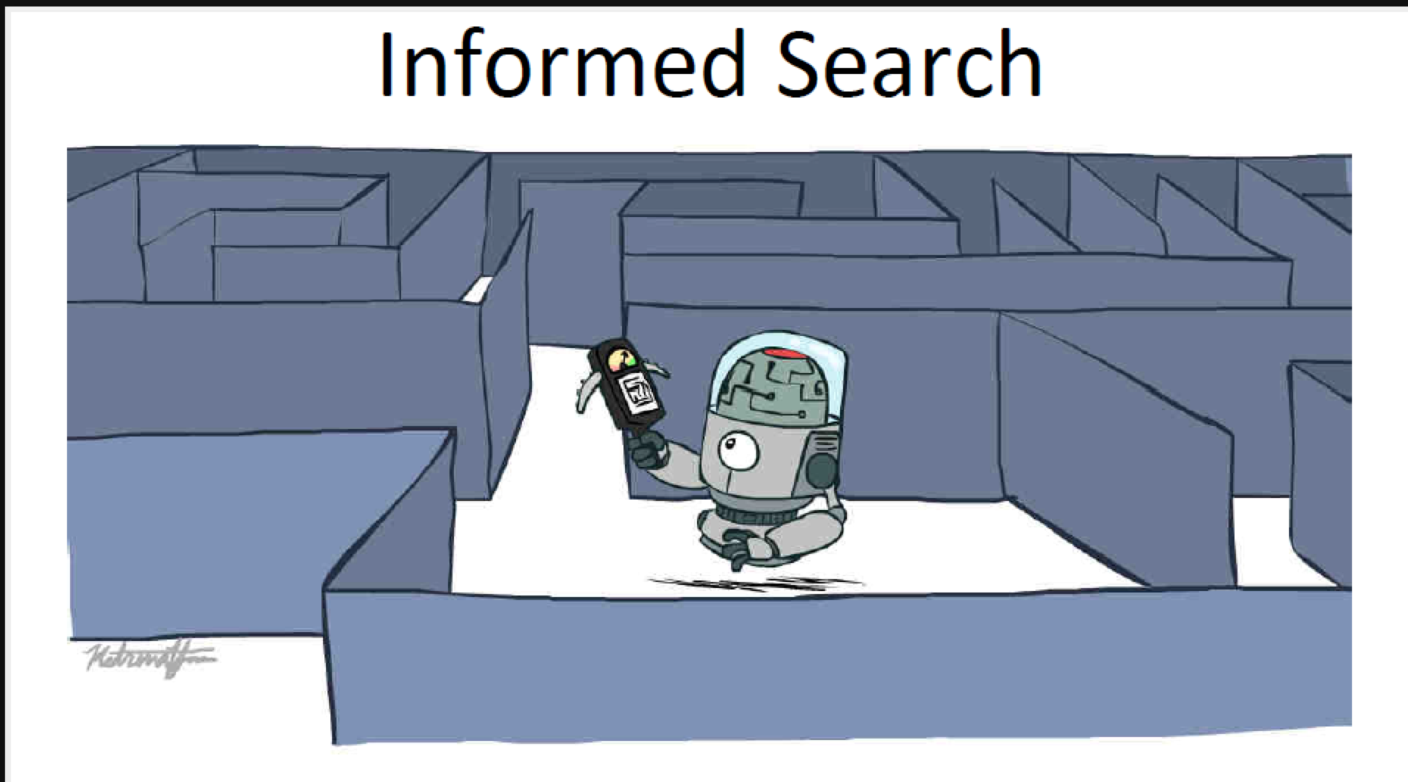
- **Strategy**: Expand node with **lowest path cost**
- **Good**: UCS is **optimal** and **complete**

Review: Uniform Cost Search

- **Bad**: explores options in **every direction**
- **Bad**: no info about **goal location**

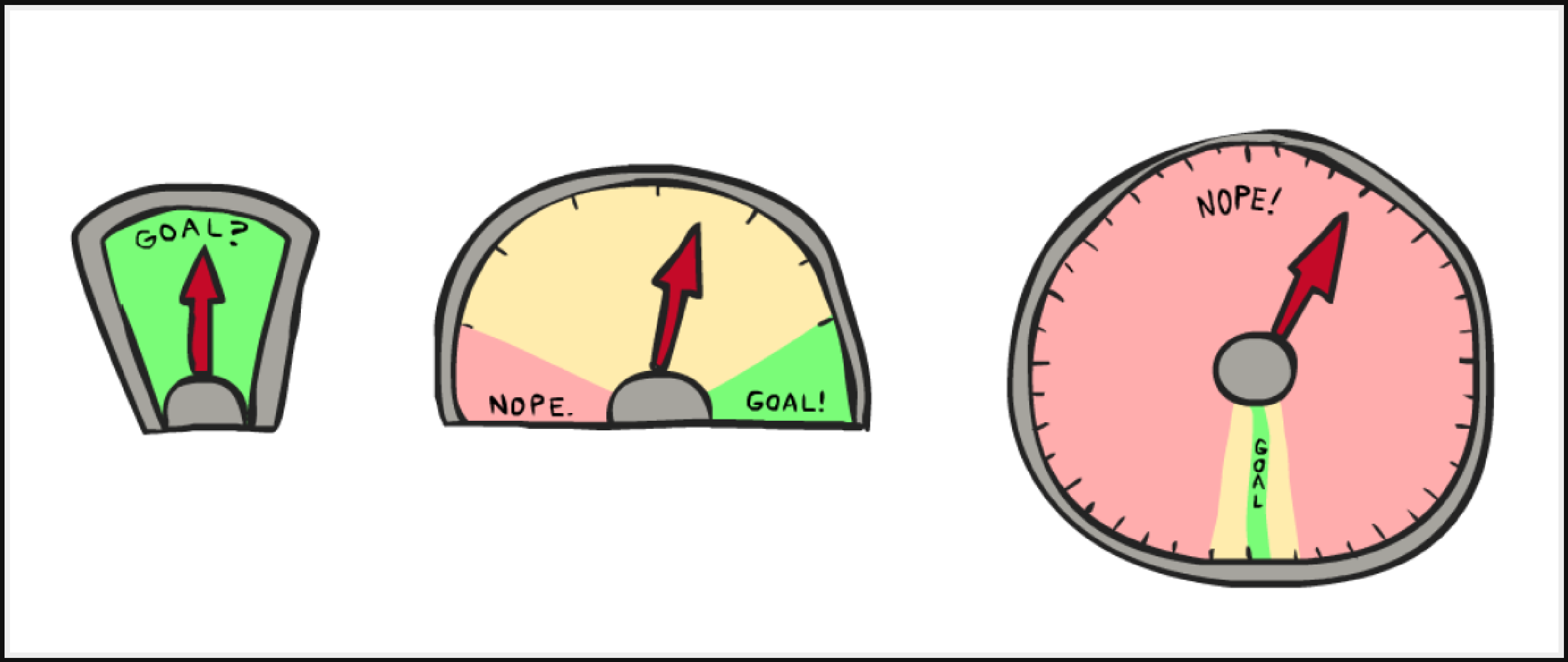


Informed Search



Informed Search

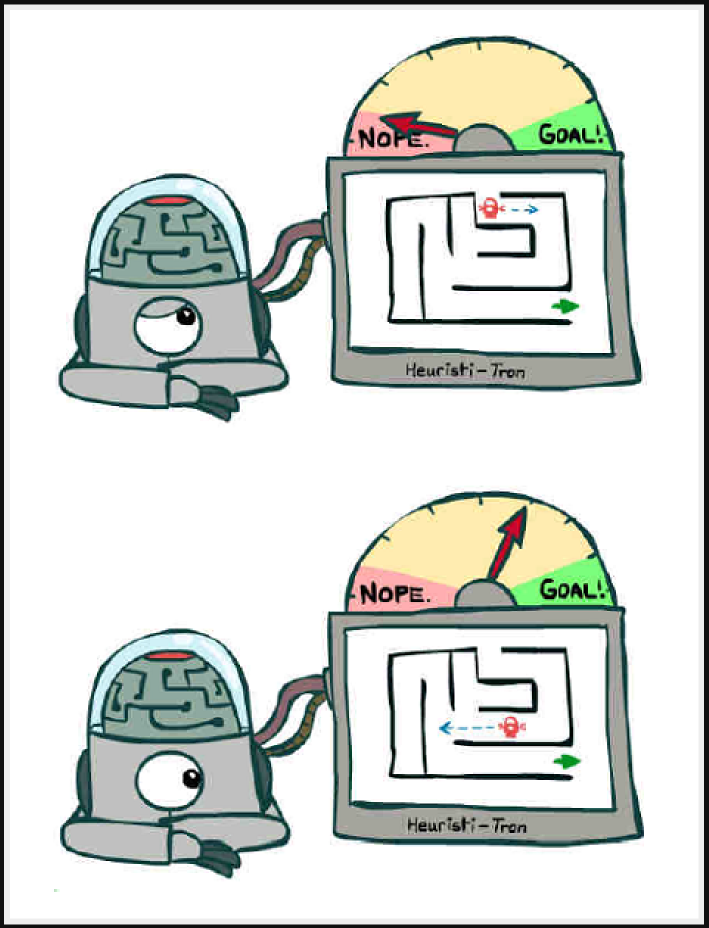
- *Key Idea*: Use a **heuristic** function
- **Compass**: look at a state and *evaluate* if that state gets you **closer** to the *goal*



Search Heuristics

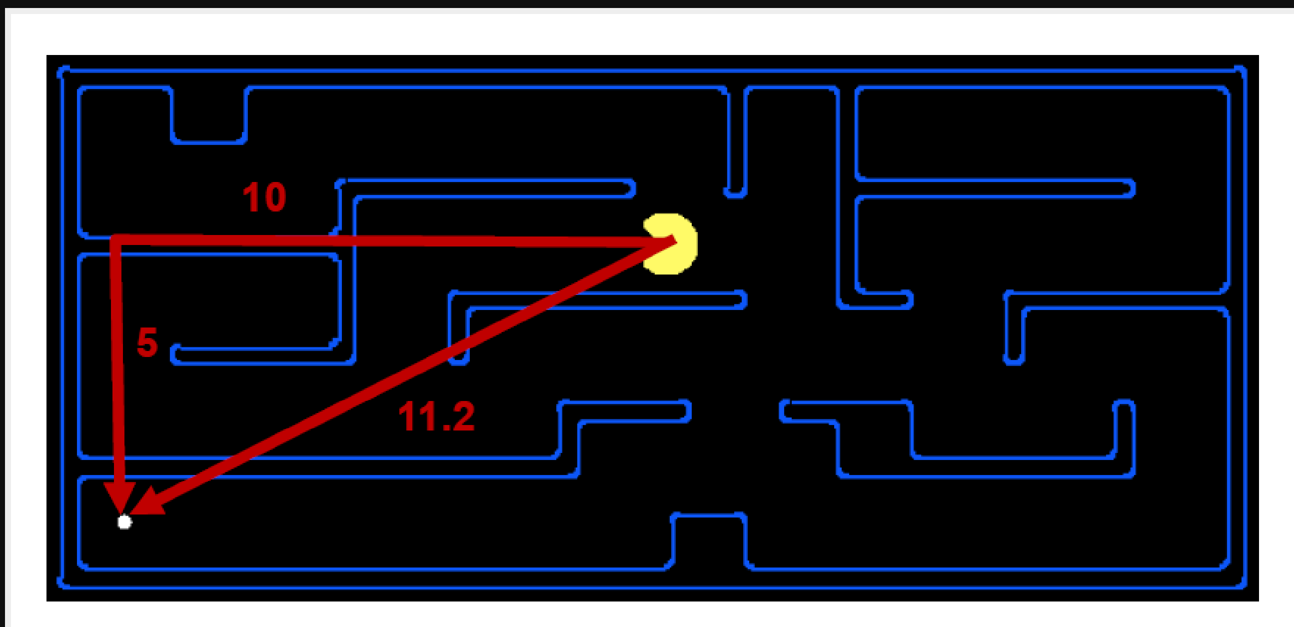
- A *function* that **estimates** how close a state is to a goal
- *Input*: state, *output*: score
- Heuristic design **varies** for different search problems

Search Heuristic



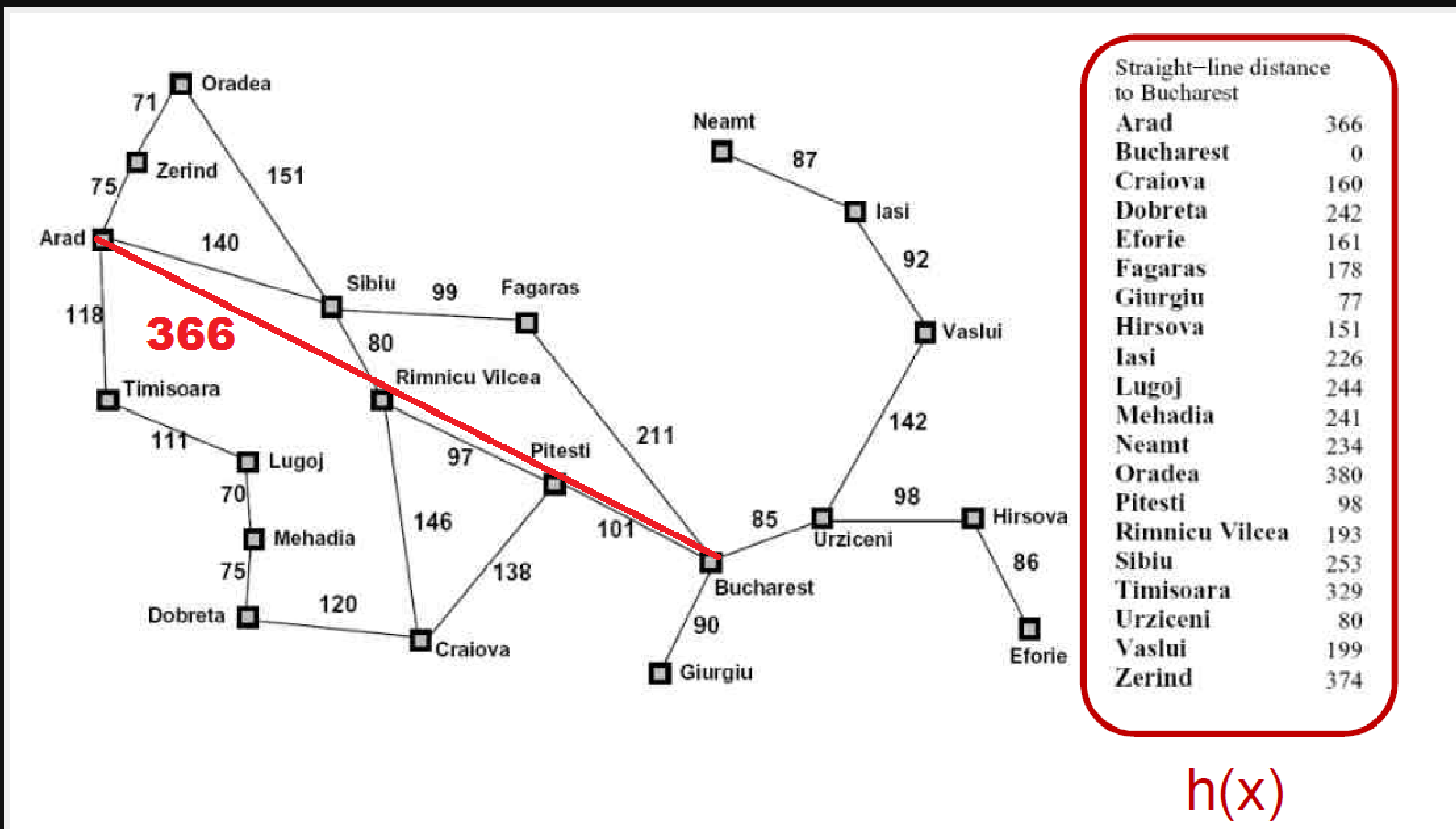
Example: Pacman

Manhattan distance, Euclidean distance



Example: Romania Vacation

Straight line distance from city to Bucharest



Greedy Search

Greedy Search

- aka *Best-First Search*
- **Strategy**: expand node that **seems closest** to *goal state*, according to **heuristic**

Greedy Search

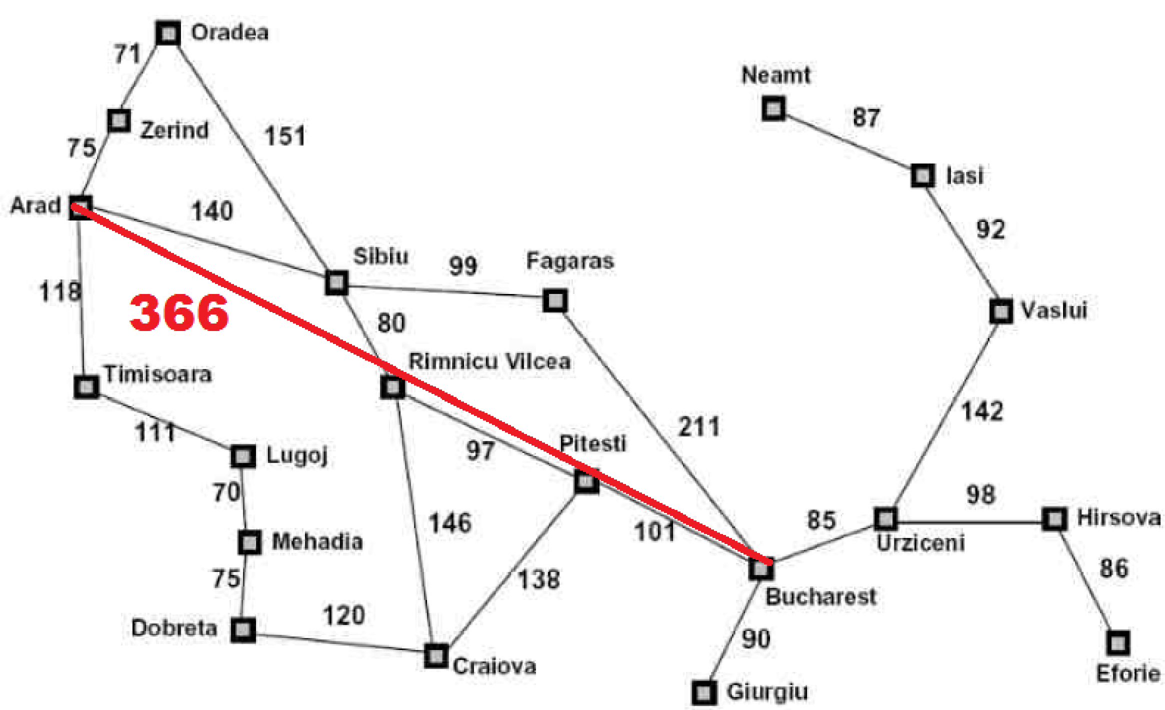
- **Heuristic**: estimates distance to nearest goal for each state
- **Implementation**: fringe = **priority queue** (priority: *heuristic value*)

UCS vs Greedy Search

- *UCS*: **cumulative** cost (sum of action costs)
- *Greedy*: heuristics are **not cumulative**

Example: Romania Vacation

UCS vs Greedy Search



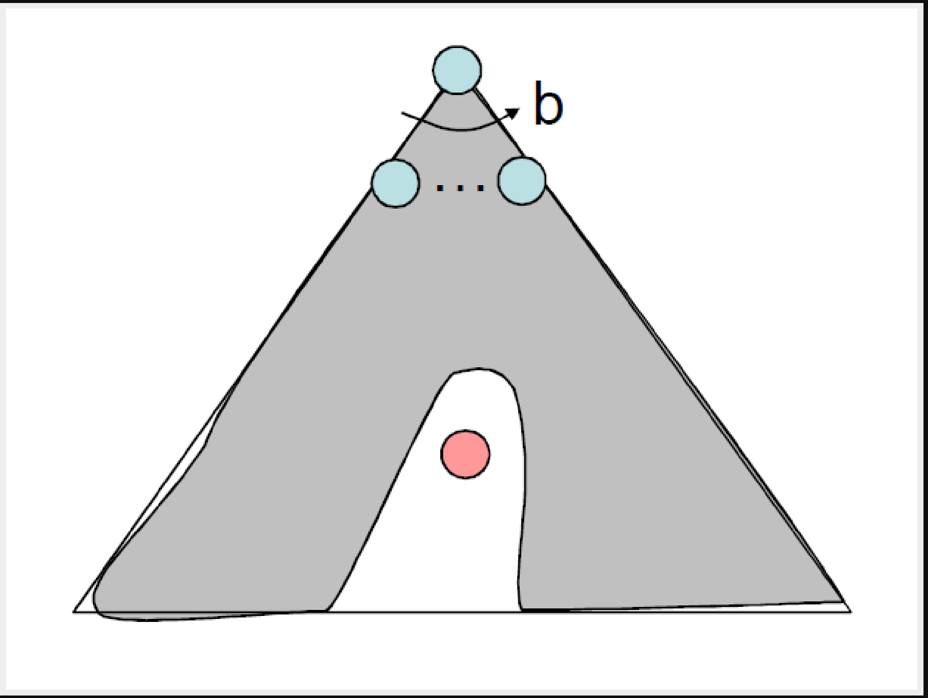
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

Greedy Search

What can go wrong?

- **Worst case**: like a badly-guided DFS



Greedy Search

- Like a **guided DFS**
- *Problem*: when optimal solution needs to do something backwards from heuristic briefly in order to proceed

Greedy Search

- **Common case**: greedy search takes you *straight* to goal, but with a **suboptimal** plan
- *Search quality* is only as good as **heuristic**
- In general, heuristics are **not perfect**

Greedy Search Properties

- **Complete**? Yes, if no loops
- **Optimal**? No
- **Time**? $O(b^m)$, but good heuristic improves this
- **Space**? $O(b^m)$, keeps all nodes in memory

UCS vs Greedy Search

- *UCS*: **optimal** solution, **no guide** (expands in all directions)
- *Greedy*: **guided** search toward goal, mostly **suboptimal** solution

A* Search

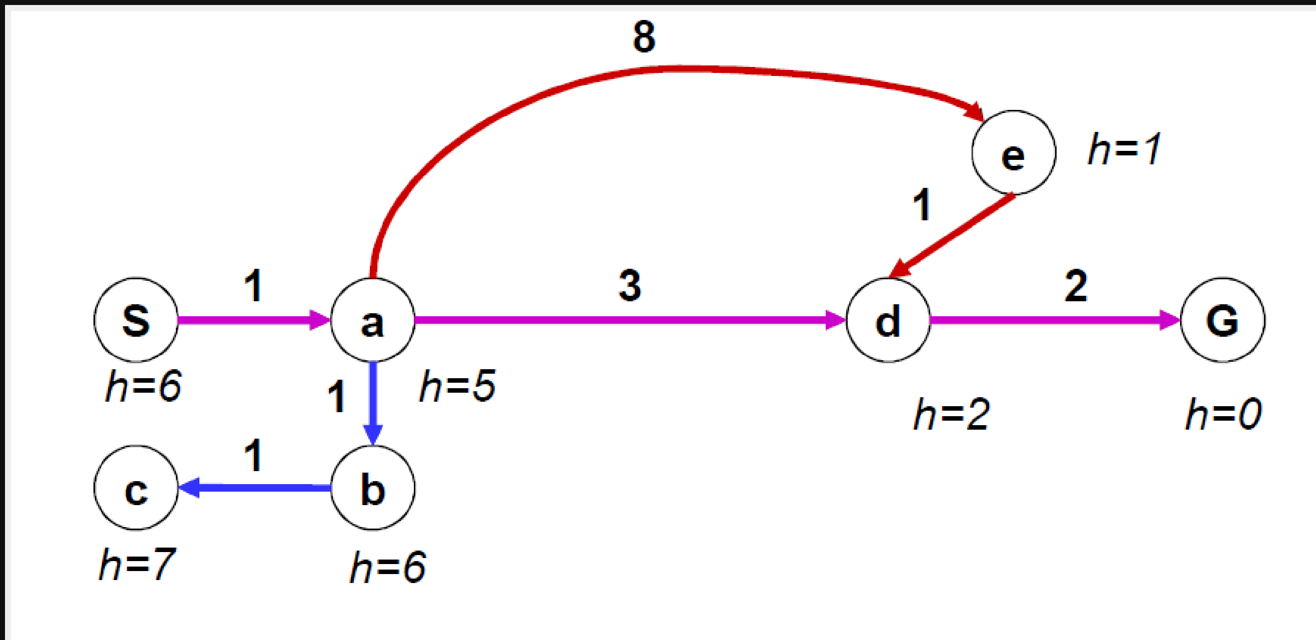
Combining UCS and Greedy

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

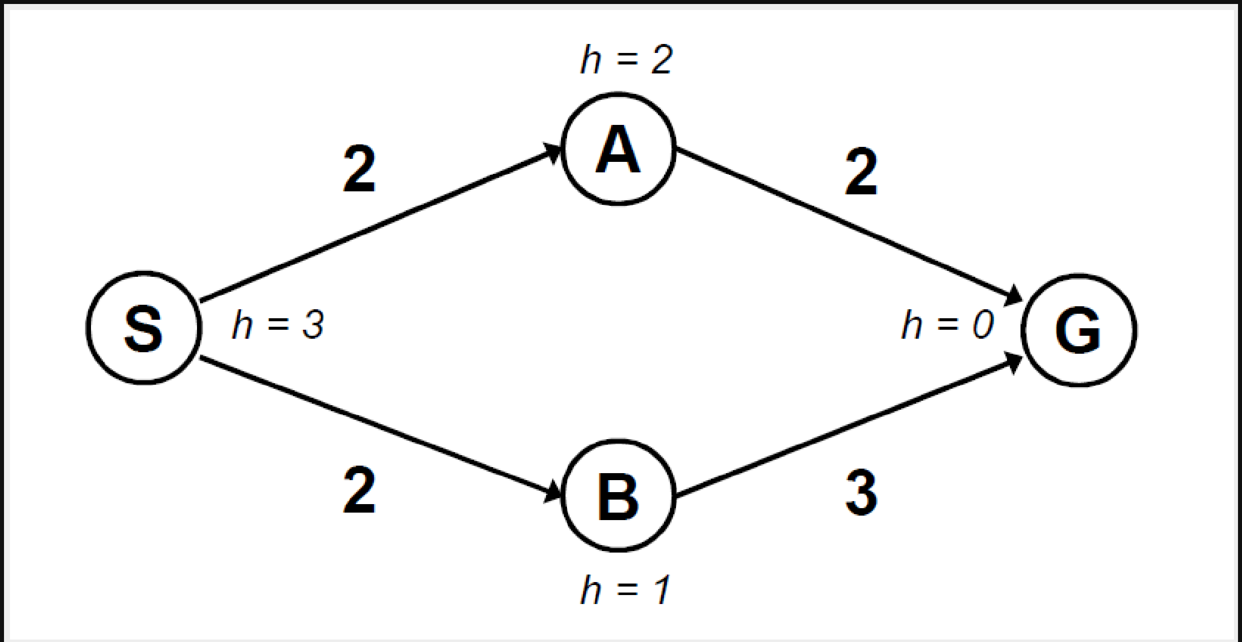
Combining UCS and Greedy

- **A* Search**: orders by the sum $f(n) = g(n) + h(n)$
- *Priority* = **actual path cost** + **heuristic**

Example: A* Search



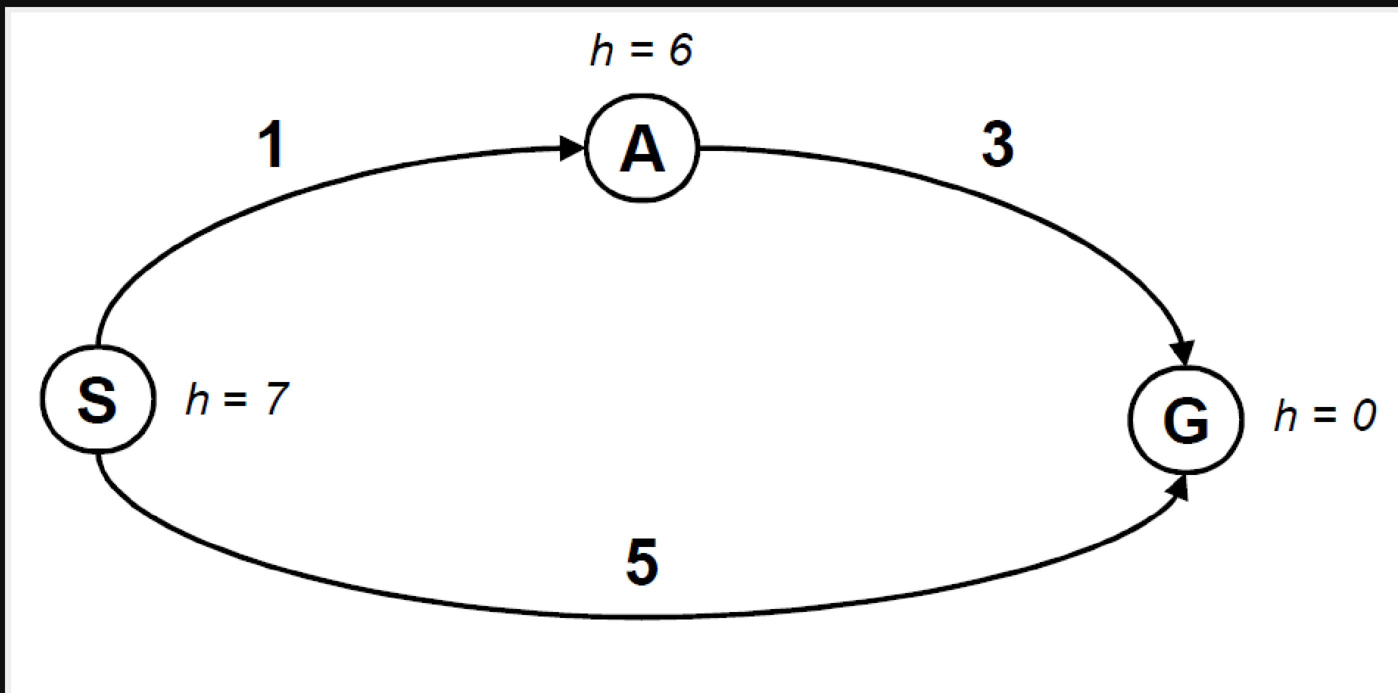
Example: A* Search



- Should A* stop when we *enqueue* a goal?
- No, stop when we **dequeue** a goal

Example: A* Search

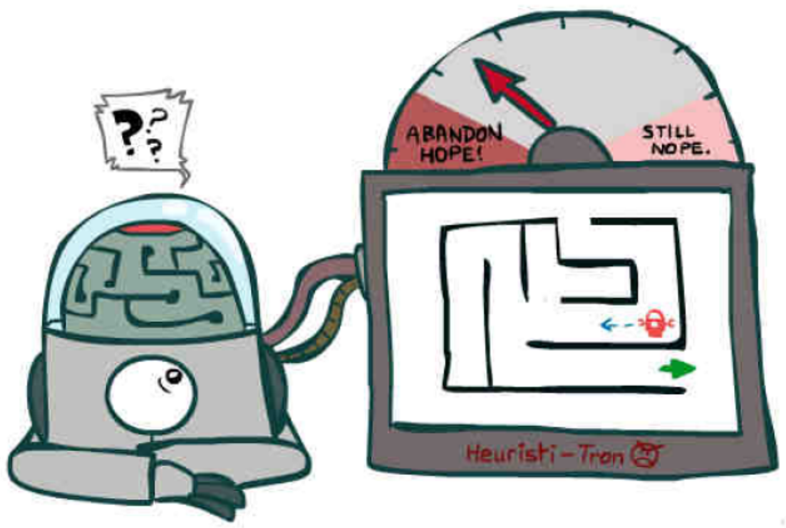
Is A* search optimal?



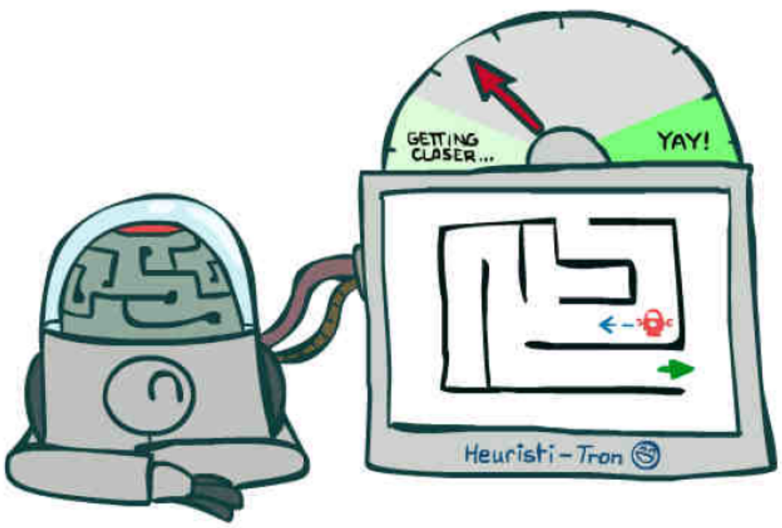
What went wrong?

- **Good path** = S-A-G, cost = 4
- **Bad path** = S-G, cost = 5
- **Heuristic** value for A = 6
- actual bad goal cost < estimated good goal cost
- *Estimates* should be **less** than *actual costs*!

Admissibility



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



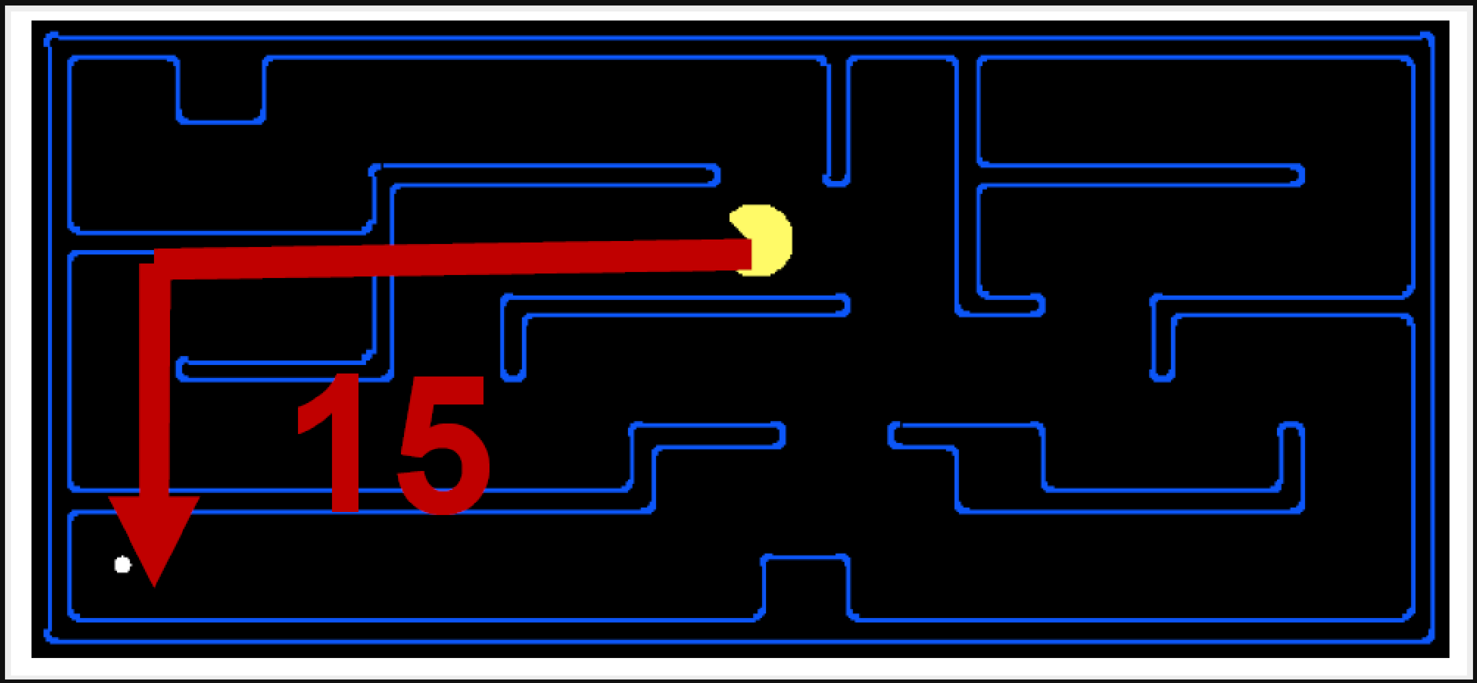
Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs

Admissible Heuristics

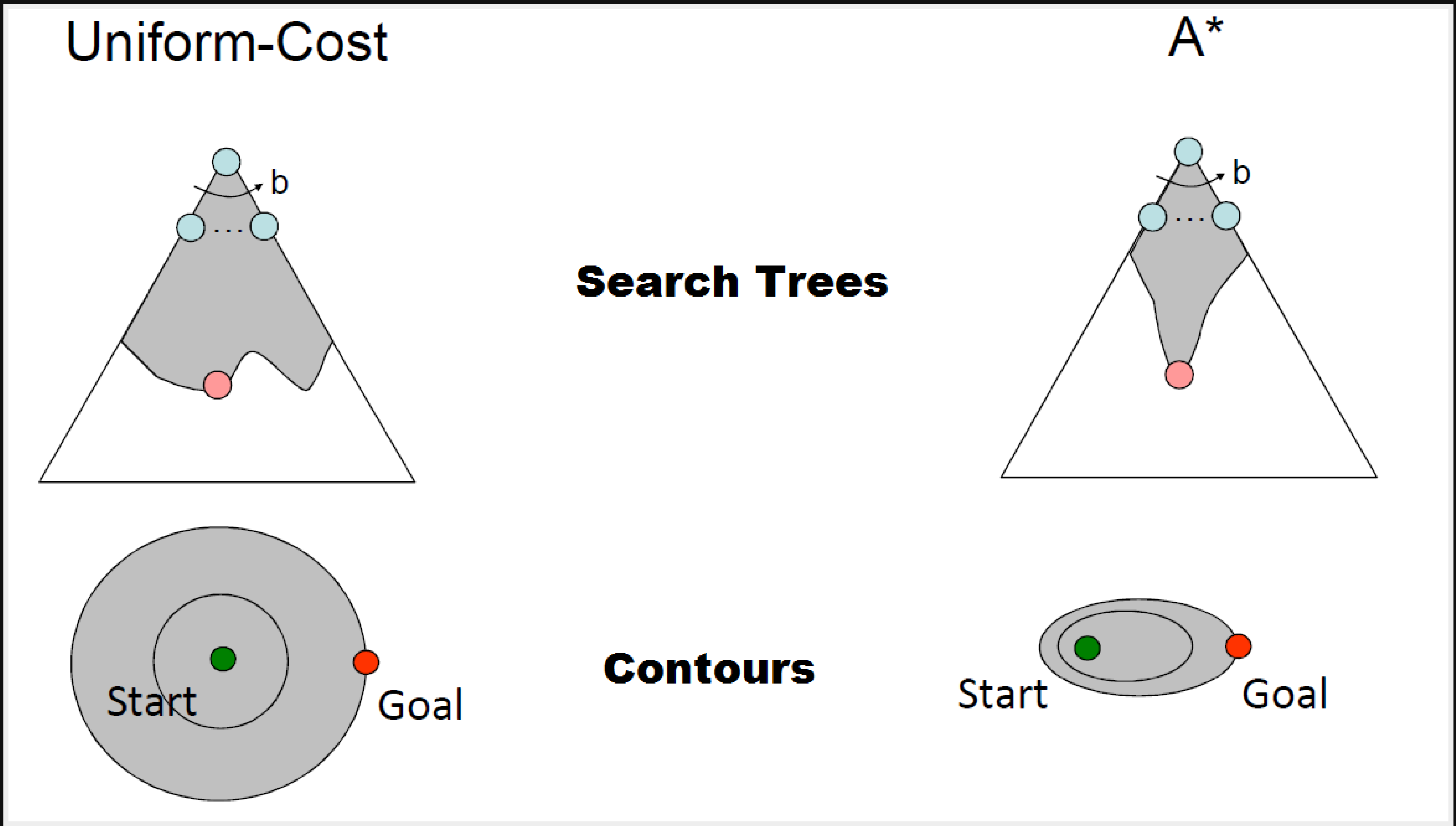
- Let $h^*(n)$ = true cost to nearest goal
- Heuristic h is **admissible** (*optimistic*) if
  $0 \leq h(n) \leq h^*(n)$
- estimate ≤ true cost

Example

Admissible heuristic: Manhattan distance



UCS vs A*


Uniform-Cost
A*
b
b
Search Trees
Contours
Start
Goal
Start
Goal

UCS vs A* Contours

- **UCS**: expands equally in all "directions"
- **A***: expands mainly toward goal, but trims its bets to ensure optimality

UCS and A*

- UCS is A* search using a *zero heuristic*
- **Zero heuristic** = no clue where the goal is, **worst** kind of heuristic

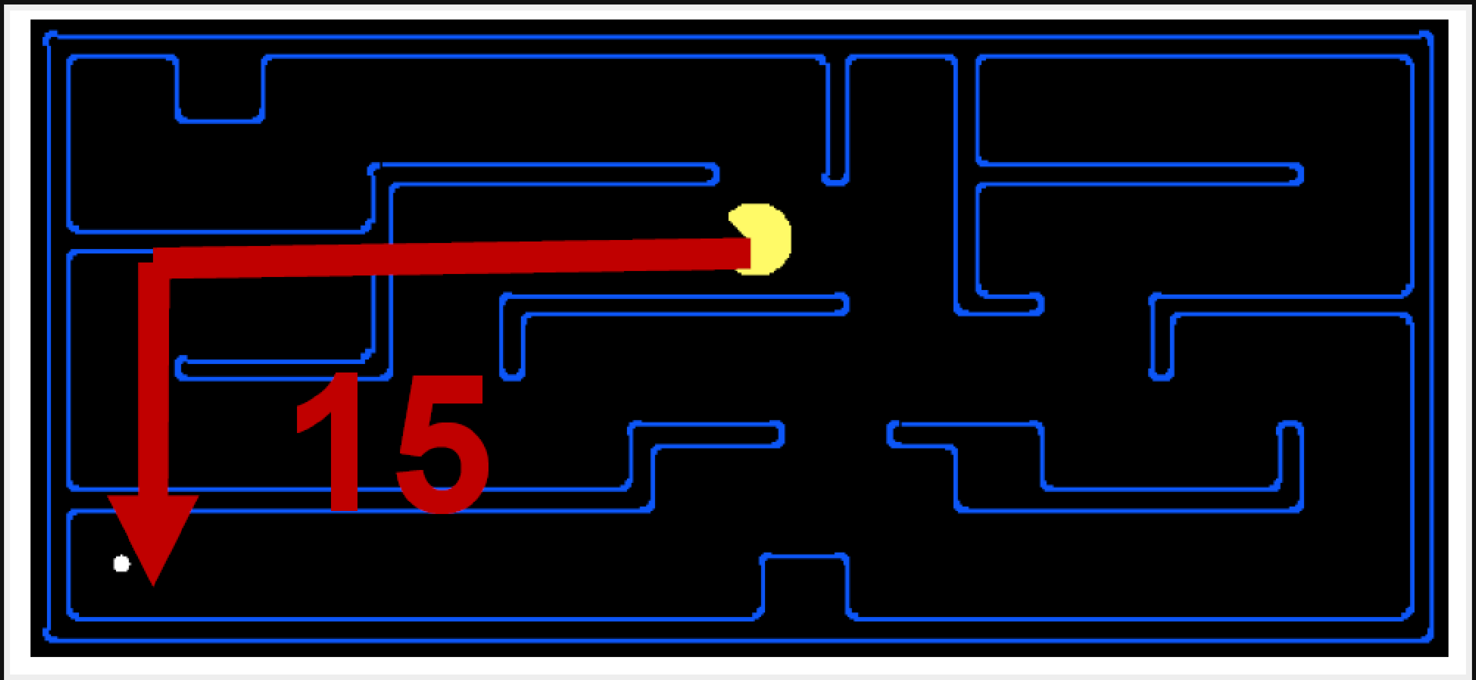
Heuristic Quality

"The worse the heuristic, the lower it is as a lower bound, the more work you're gonna do in order to guarantee optimality."

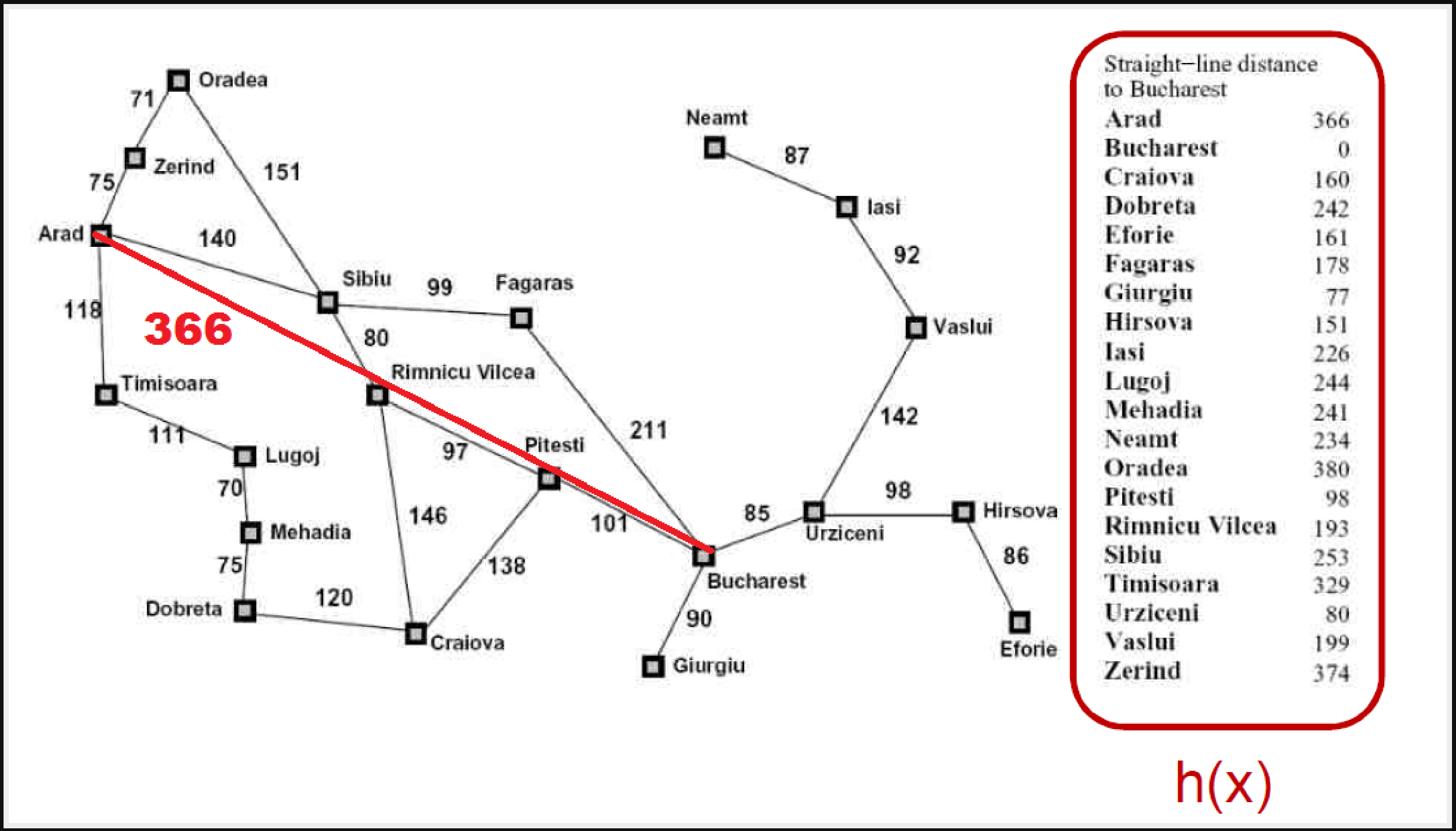
Admissible Heuristics

- *In practice*, designing an **admissible heuristic** is most of the work in using A* search, to ensure **optimal** results
- *Usually*: solutions to **relaxed problems**, where *new actions* are available

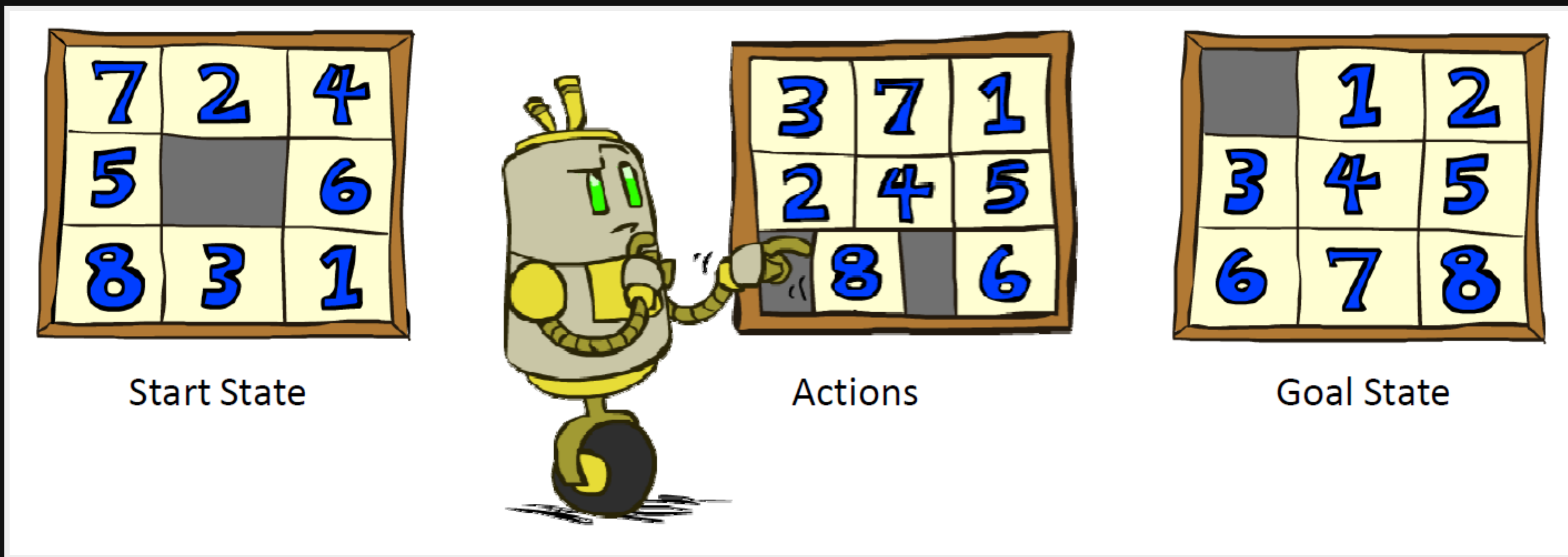
Creating Heuristics



Creating Heuristics



Example: 8 Puzzle

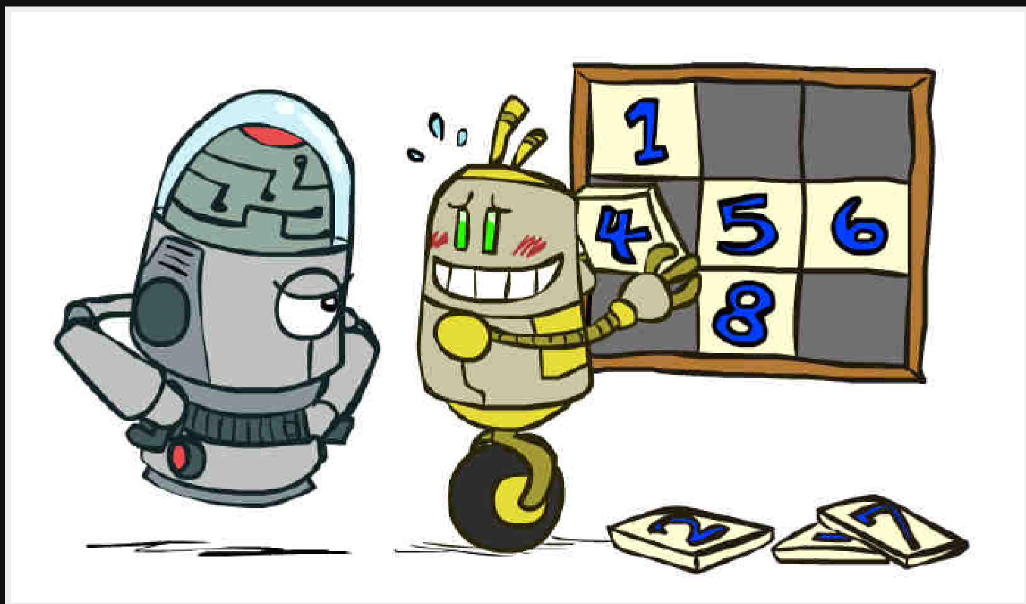


What are possible heuristics for 8 Puzzle?

Heuristic 1

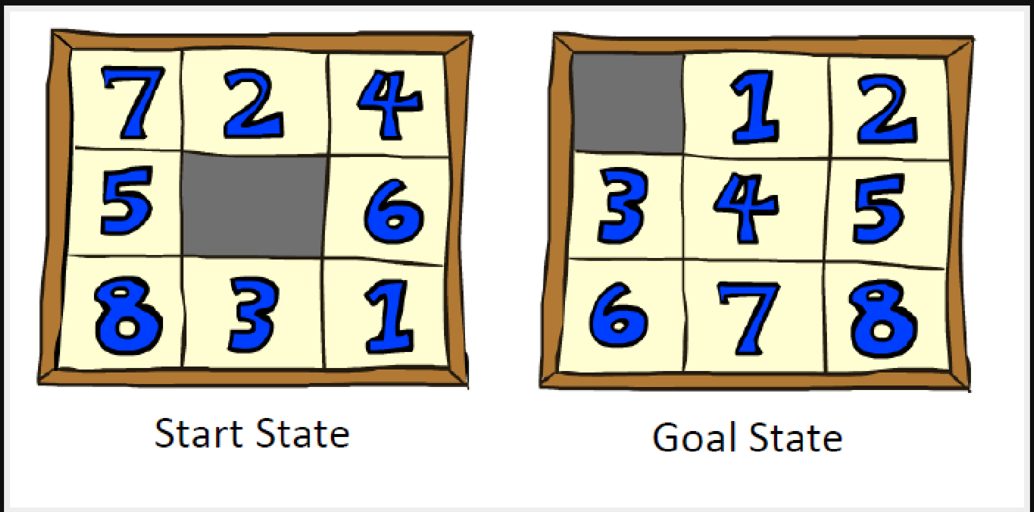
- Number of misplaced tiles
- Loose, extreme relaxation of problem

Heuristic 1



Heuristic 1

h(start) = 8



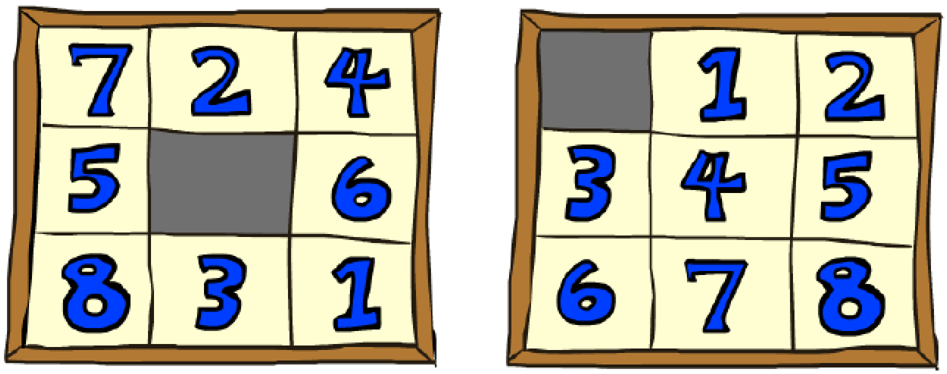
Note: Don't count the free space

Heuristic 2

- If we can make any tile slide in any direction (NEWS) at any time, **ignoring other tiles**
- Total Manhattan distance

Heuristic 2

h(start) = 18



Start State

Goal State

3 + 1 + 2 + 2 + 2 + 3 + 3 + 2

Relaxed Problems

- **8 Puzzle**: a block can move from A to B if **(1) A is adjacent to B**, and **(2) B is blank**
- Relax condition 2 = heuristic 2
- Relax condition 1,2 = heuristic 1
- Both heuristics are **admissible**

Heuristic 3

- How about **actual cost** as heuristic?
- Would it be admissible? Yes
- Would we save on nodes expanded? Yes
- What's wrong with it? **Too long** to compute

A* Search

- **Tradeoff** between **quality of estimate** and **work per node**
- As heuristic gets **closer** to *true cost*: **expand fewer** nodes, but do **more work** per node to compute heuristic

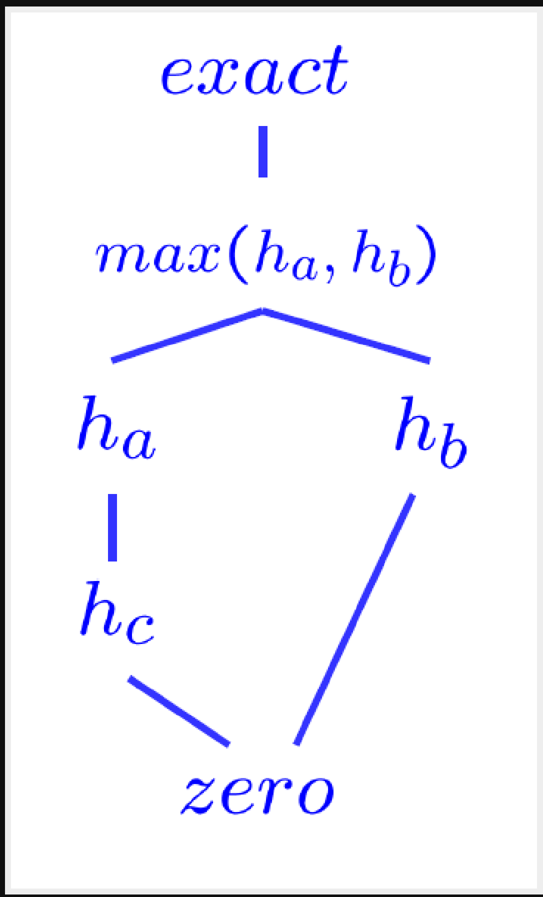
Dominance

- $h_a \geq h_c$ if $\forall$ n: $h_a(n) \geq h_c(n)$
- *Example*: 8-puzzle, heuristic 2 ≥ heuristic 1

Heuristics

- Max of two or more admissible heuristics is admissible
- $h(n) = \max(h_a(n), h_b(n))$

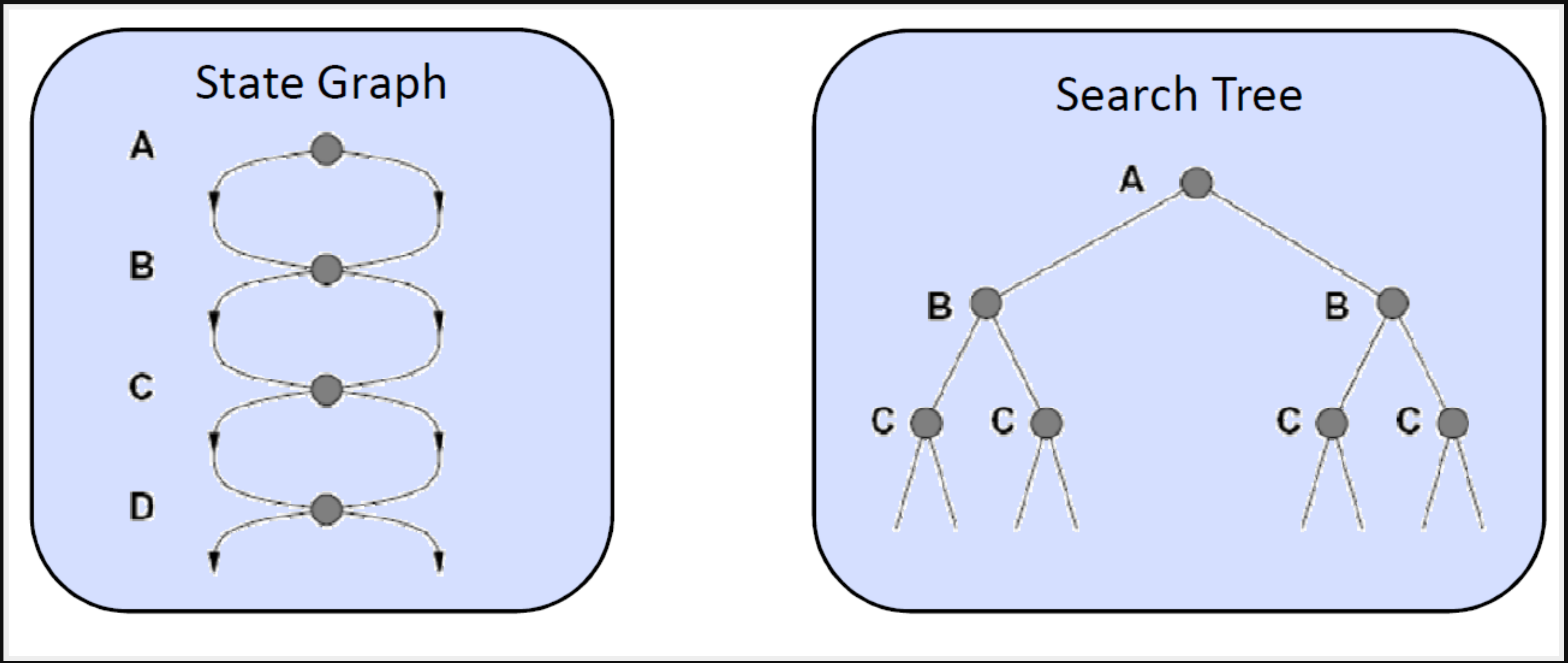
Heuristics



Recall: 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
```

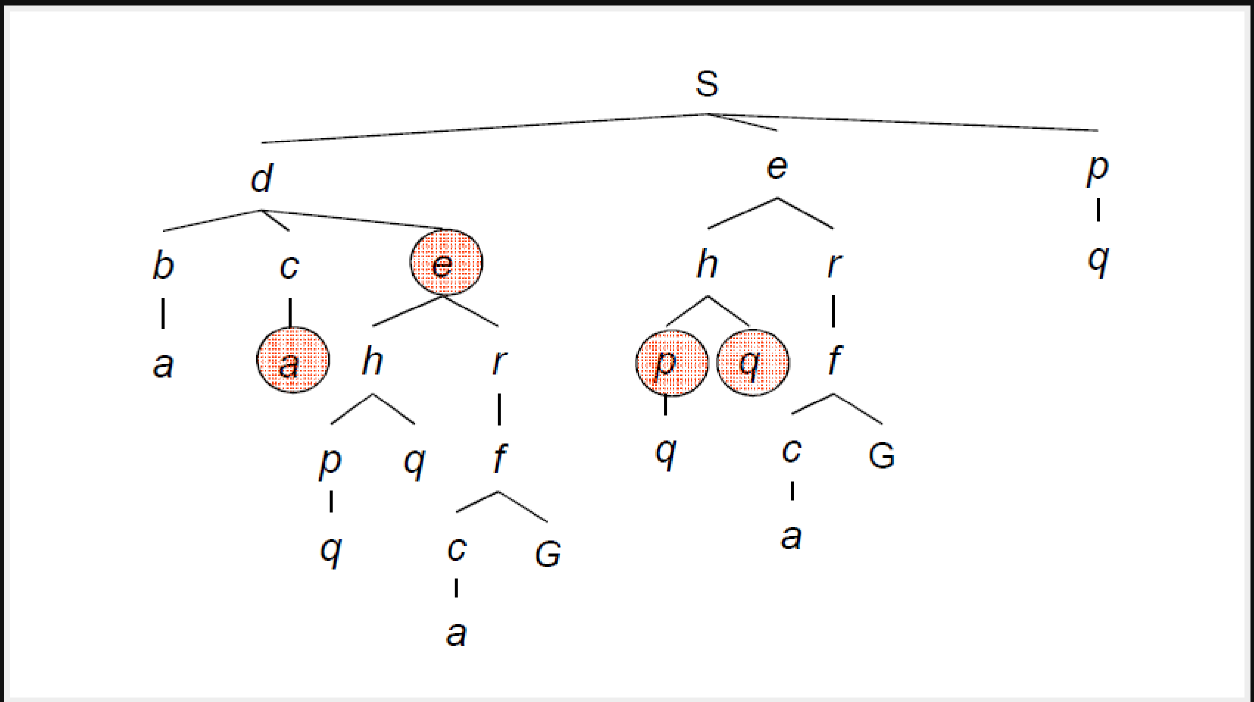
Recall: Tree Search



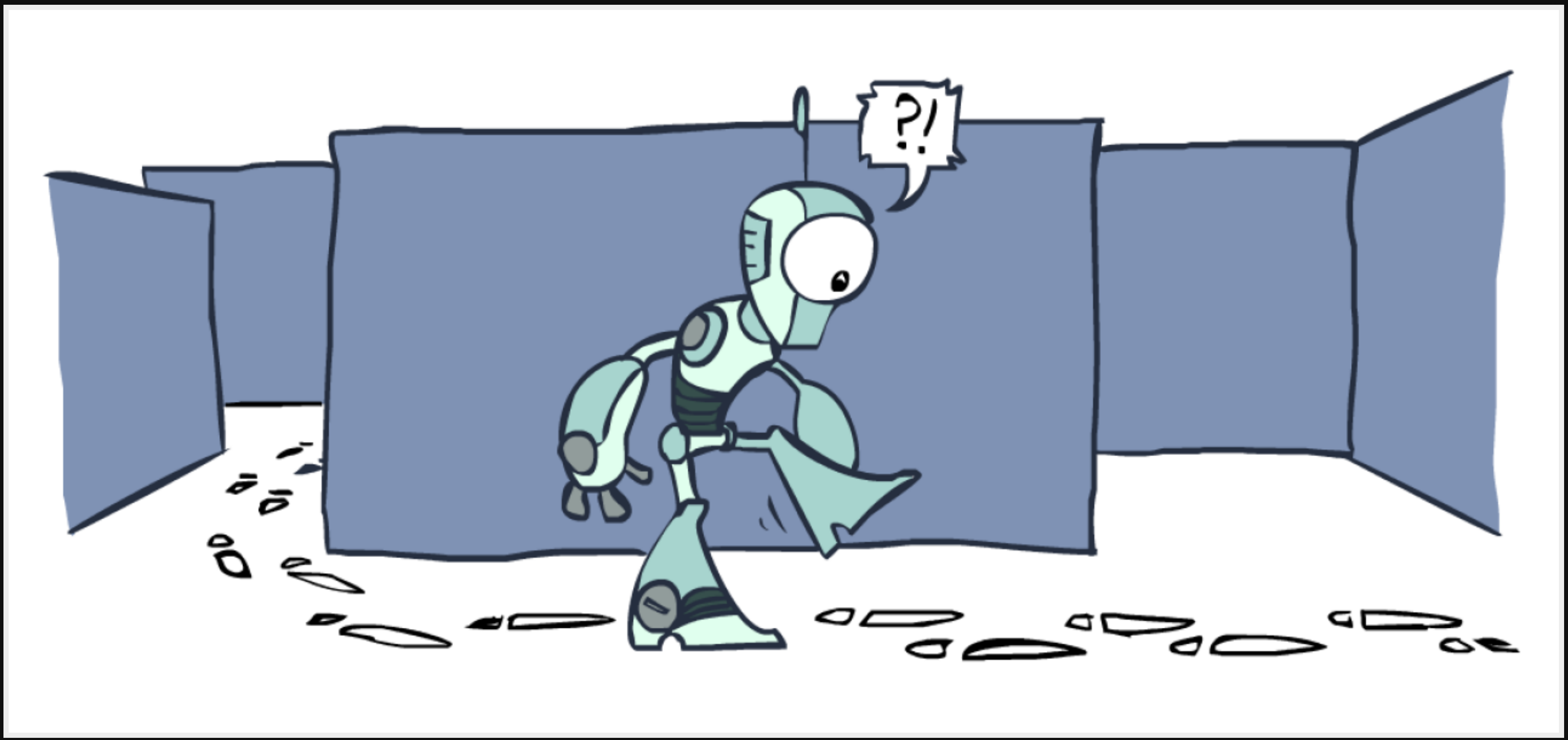
Failure to **detect repeated states** can cause **exponentially** more work

Example

Why shouldn't we bother expanding red nodes?



Graph Search



Graph Search

- *Idea*: never expand a state twice
- Tree search + set of expanded states

Graph Search

- Expand search tree node by node
- Before expanding node, check that it hasn't been expanded before
- If not new, skip it
- If new, expand and add to set

Graph Search

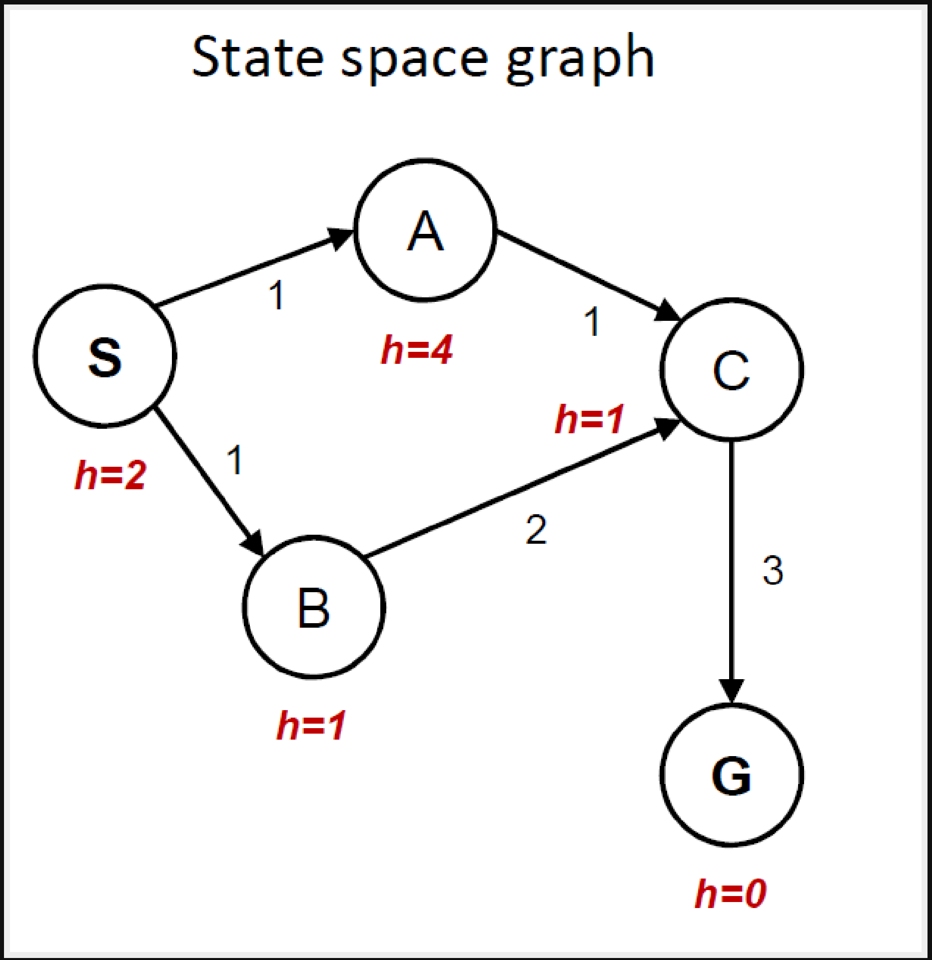
```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure

   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
```

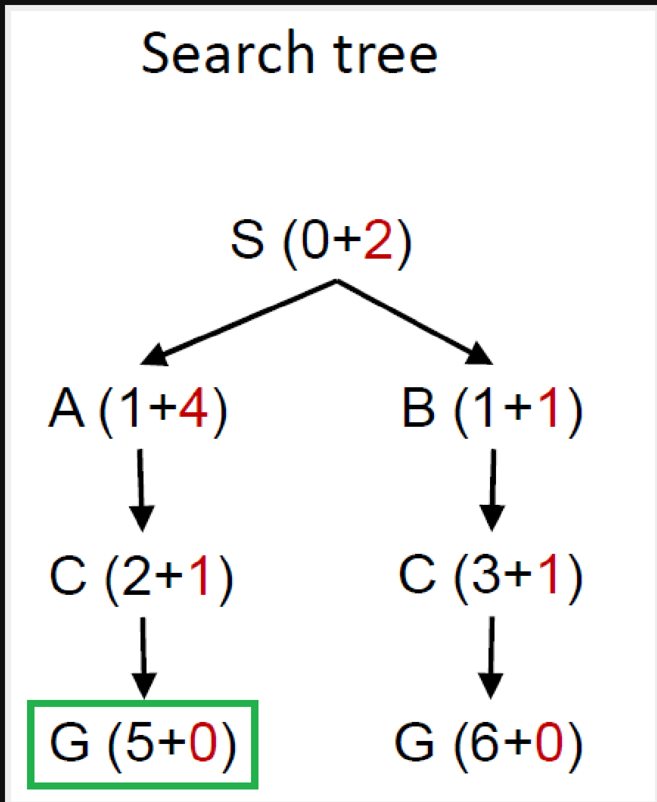
Question

- Can graph search destroy completeness? No
- How about optimality? Yes

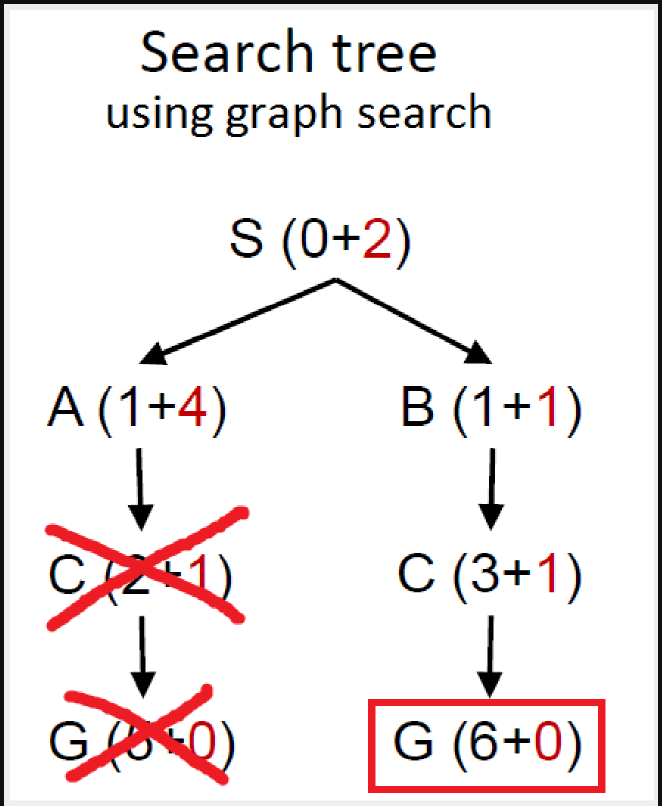
Example



Example: Tree Search



Example: Graph Search



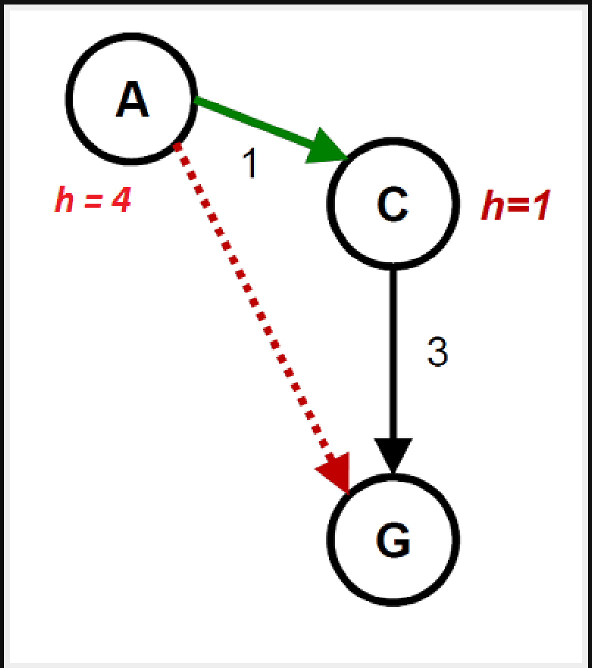
Admissible Heuristics

- heuristic cost ≤ actual cost of **path to goal**
- h(A) ≤ actual cost from A to G
- **A* tree search** is *optimal*

Consistent Heuristics

- heuristic cost ≤ actual cost, **for each arc**
- h(A) - h(C) ≤ actual cost from A to C,
  for each arc A-C

Example



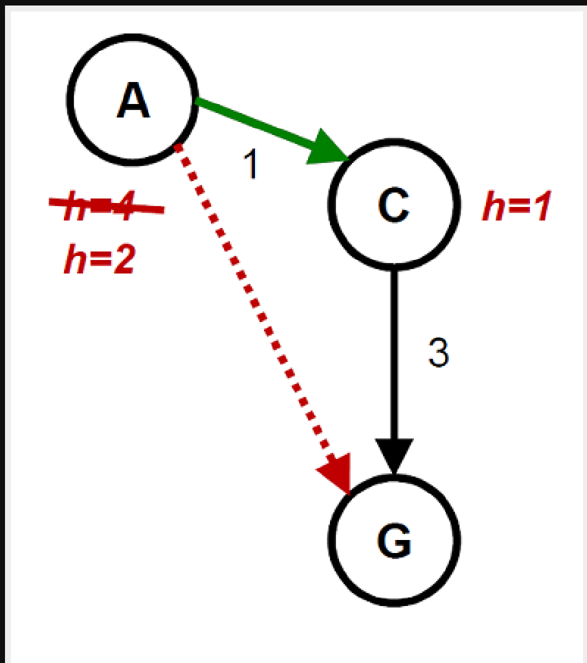
Example

Estimate of cost from:

- A to G = **h(A)** = 4
- C to G = **h(C)** = 1
- A to C = **h(A) - h(C)** = 4 - 1 = 3
- Estimate of cost from A to C **exceeds** actual cost = 1

Consistent Heuristics

Consistent heuristic if h(A) = 2



- h(A-C) = 2-1 = 1 ≤ actual(A-C) = 1
- h(C-G) = 1-0 = 1 ≤ actual(C-G) = 3

Consequences of Consistency

- f value along a path never decreases
- $h(A) \leq \text{cost}(A \text{ to } C) + h(C)$
- **A* graph search** is *optimal*

A* Search Properties

- Complete? Yes
- Optimal? Depends on the heuristic

A* Search Optimality

Optimal if:

- **A* tree search**: heuristic is **admissible**
- **A* graph search**: heuristic is **consistent**

Heuristics

- Consistency **implies** admissibility
- In general, most **admissible** heuristics tend to be **consistent**, especially if from relaxed problems

A* Summary

- Uses both **backward costs** + **estimates of forward cost**
- **Optimal** with admissible / consistent heuristics
- **Heuristic design** is *key*: often use **relaxed** problems

Informed Search

- Add information about where the goal is
- Find optimal solutions while exploring less of search tree

Demo: UCS vs Greedy vs A*

Search

- Search and Planning
- Completeness vs Optimality
- Uninformed Search (BFS, DFS, ID, UCS)
- Informed Search (Greedy, A*)
- Admissible vs Consistent Heuristics

Announcements

- **Assignment 2** available at Courseware
- *Tuesday*: **MP1 - Pacman Search** (Python 2)
- *Next Topic*: Constraint Satisfaction Problems

References

- *Artificial Intelligence: A Modern Approach, 3rd Edition*, S. Russell and P. Norvig, 2010
- CS 188 Lec 3 slides, Dan Klein, UC Berkeley

Questions?