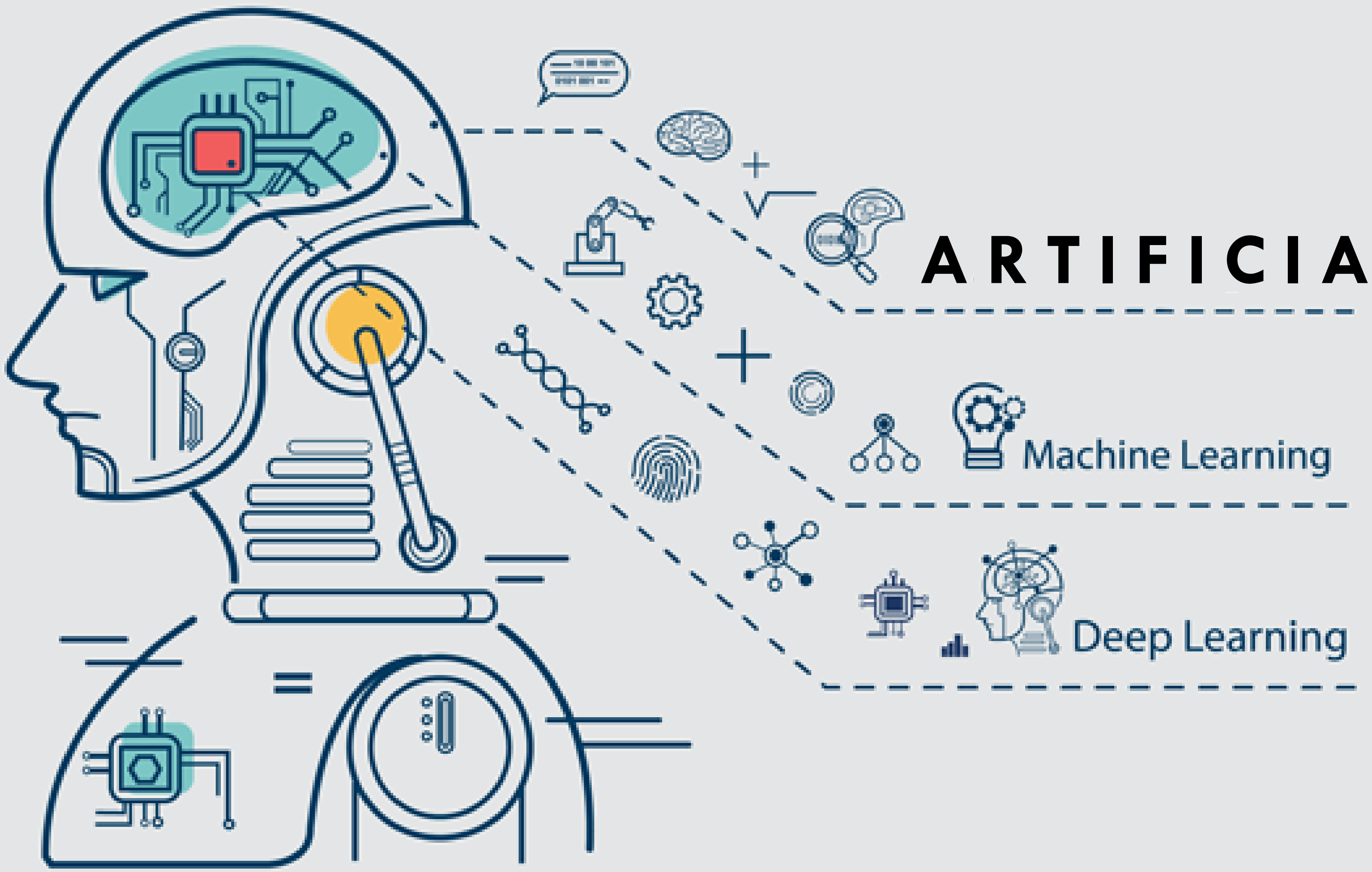
ARTIFICIAL INTELLIGENCE



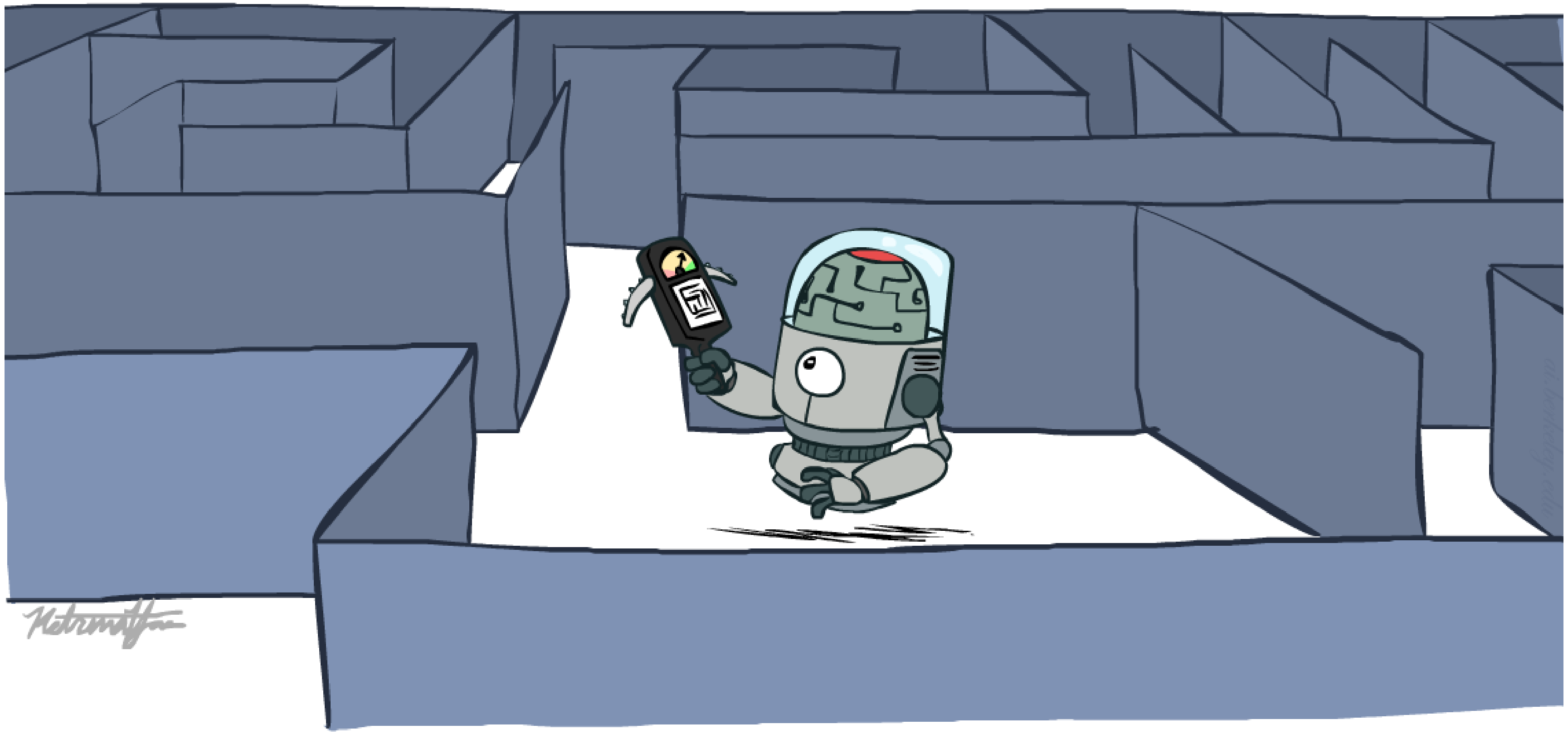
Dr. Seemab latif

Lecture 3

1 Oct 2024

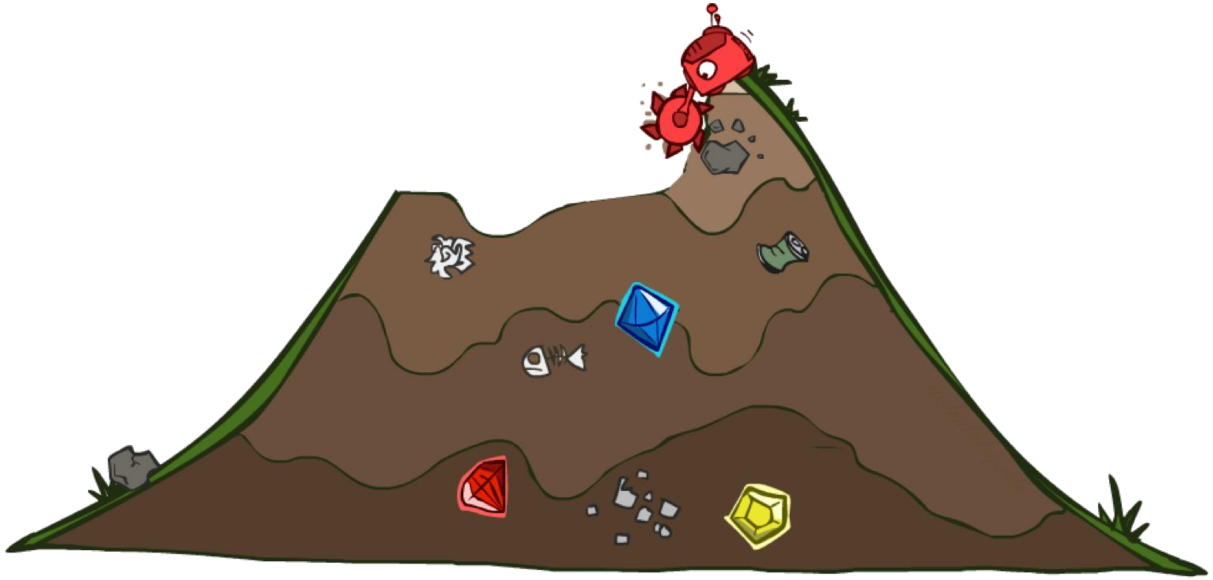
AI: Representation and Problem Solving

Informed Search



Slide credits: CMU AI, http://ai.berkeley.edu

Uninformed vs Informed Search



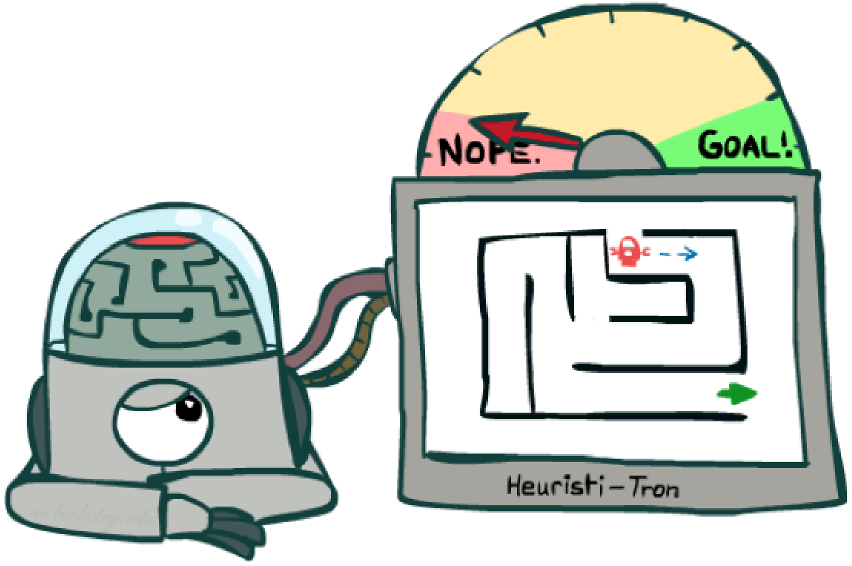
Today

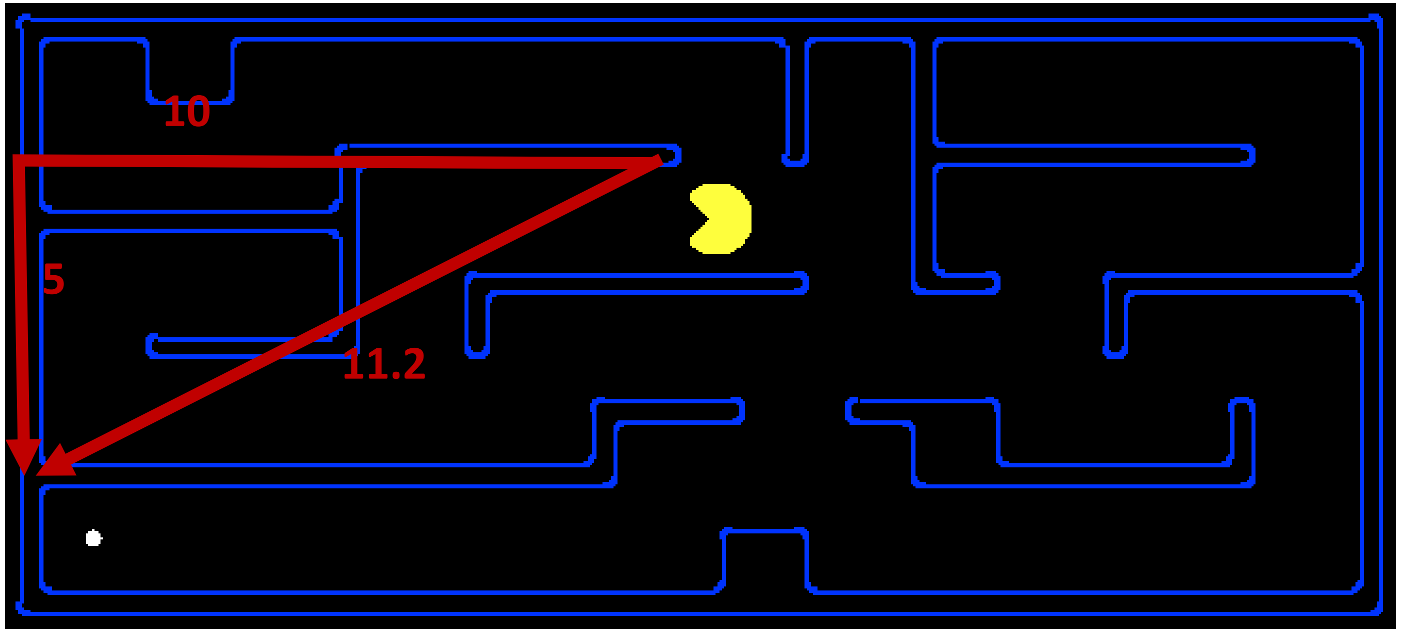
- Informed Search
  - Heuristics
  - Greedy Search
  - A* Search

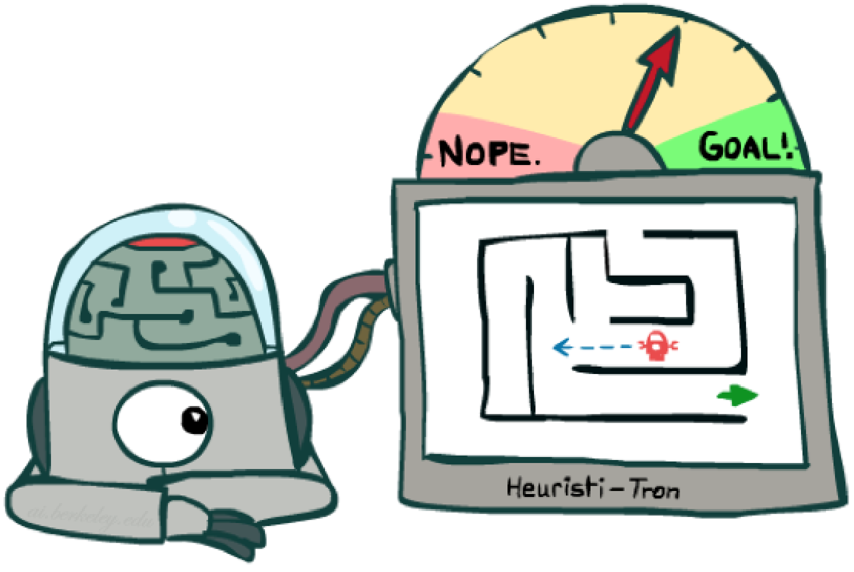
Search Heuristics

A heuristic is:

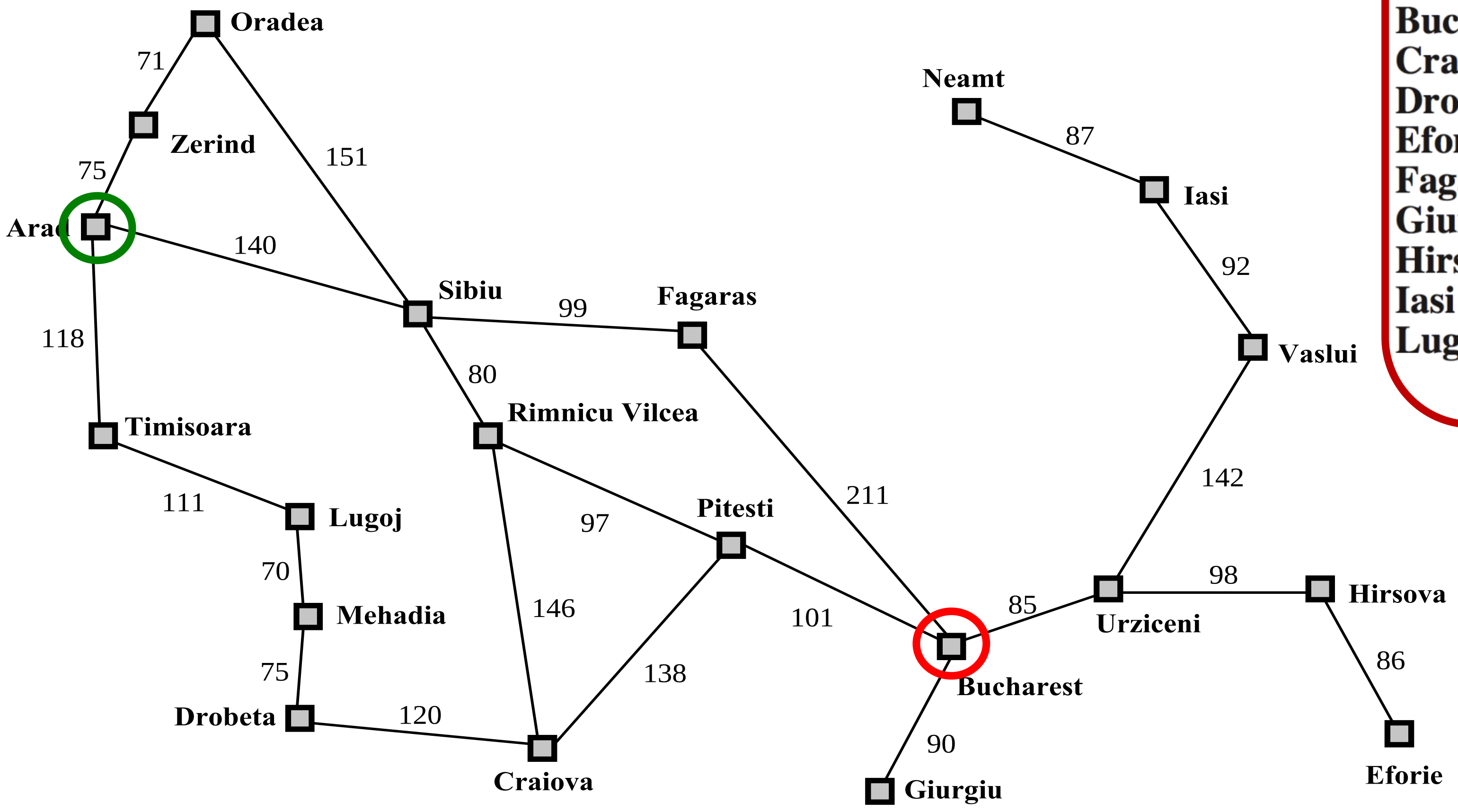
- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
- Examples: Manhattan distance, Euclidean distance for pathing







Example: Euclidean distance to Bucharest

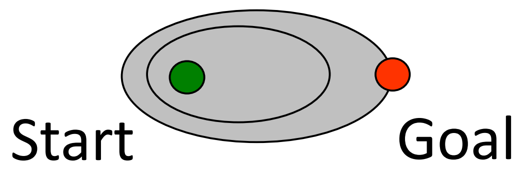


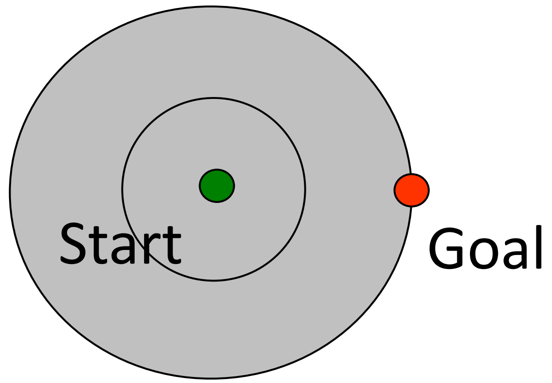
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

h(state) → value

Effect of heuristics

Guide search ***towards the goal*** instead of ***all over the place***





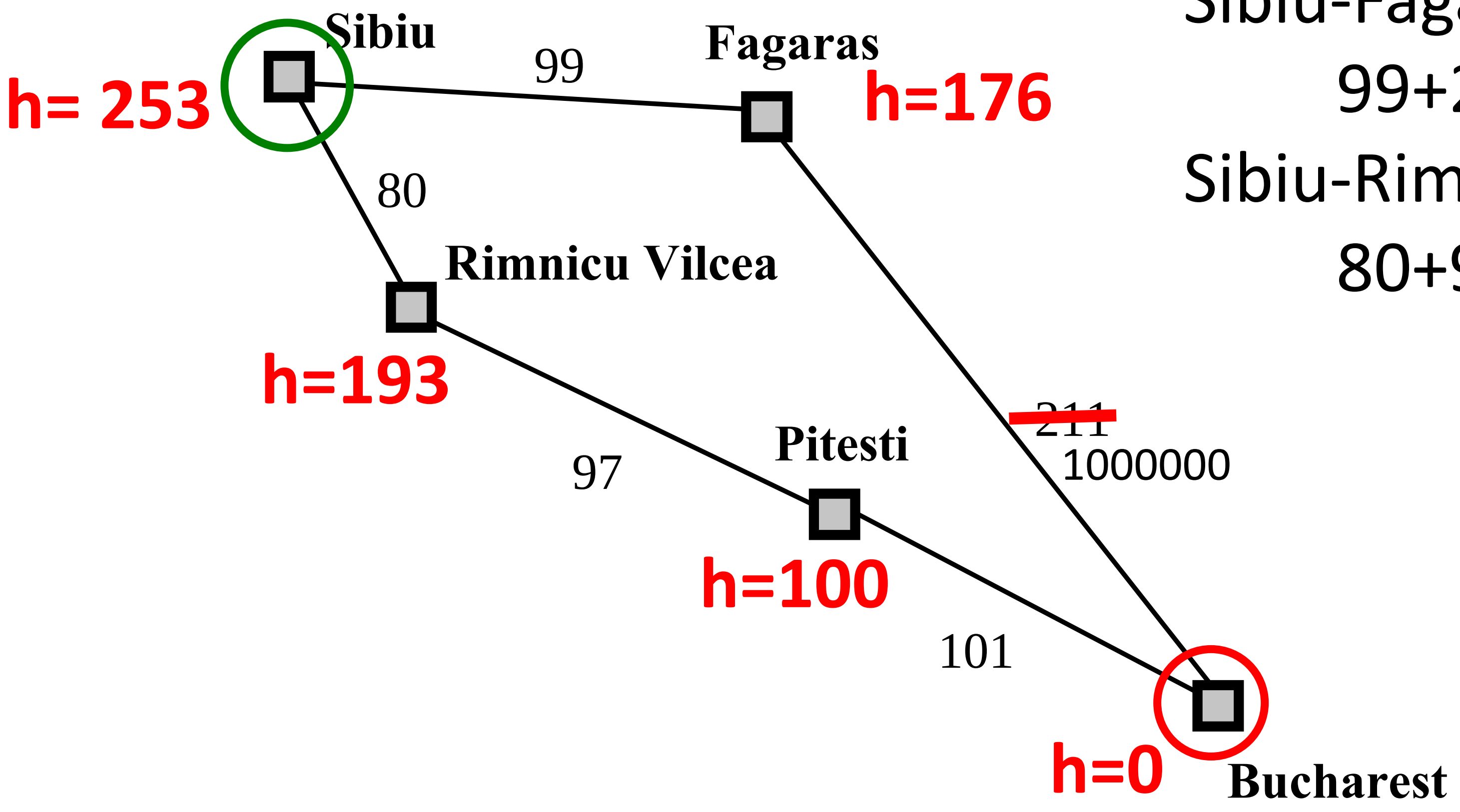
Informed

Uninformed

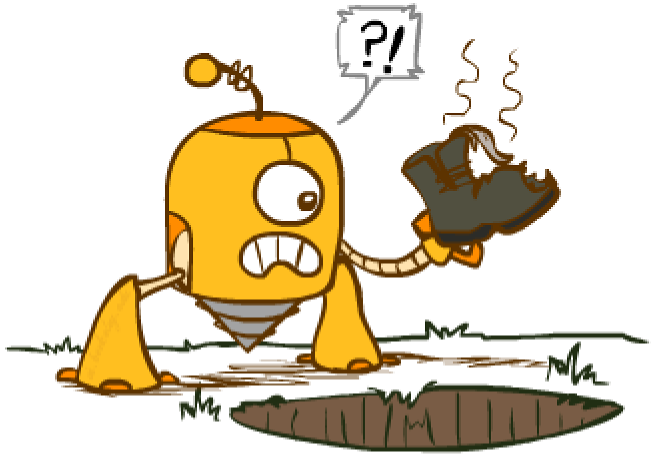
Greedy Search

Greedy Search

- Expand the node that seems closest…(order frontier by h)
- What can possibly go wrong?



Sibiu-Fagaras-Bucharest =
99+211 = **310**
Sibiu-Rimnicu Vilcea-Pitesti-Bucharest =
80+97+101 = **278**

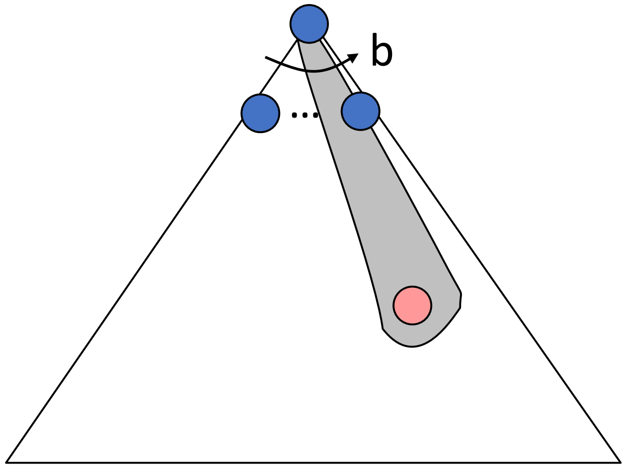


Greedy Search

- Strategy: expand a node that ***seems*** closest to a goal state, according to h

- Problem 1: it chooses a node even if it's at the end of a very long and winding road

- Problem 2: it takes h literally even if it's completely wrong



A* Search

A* Search

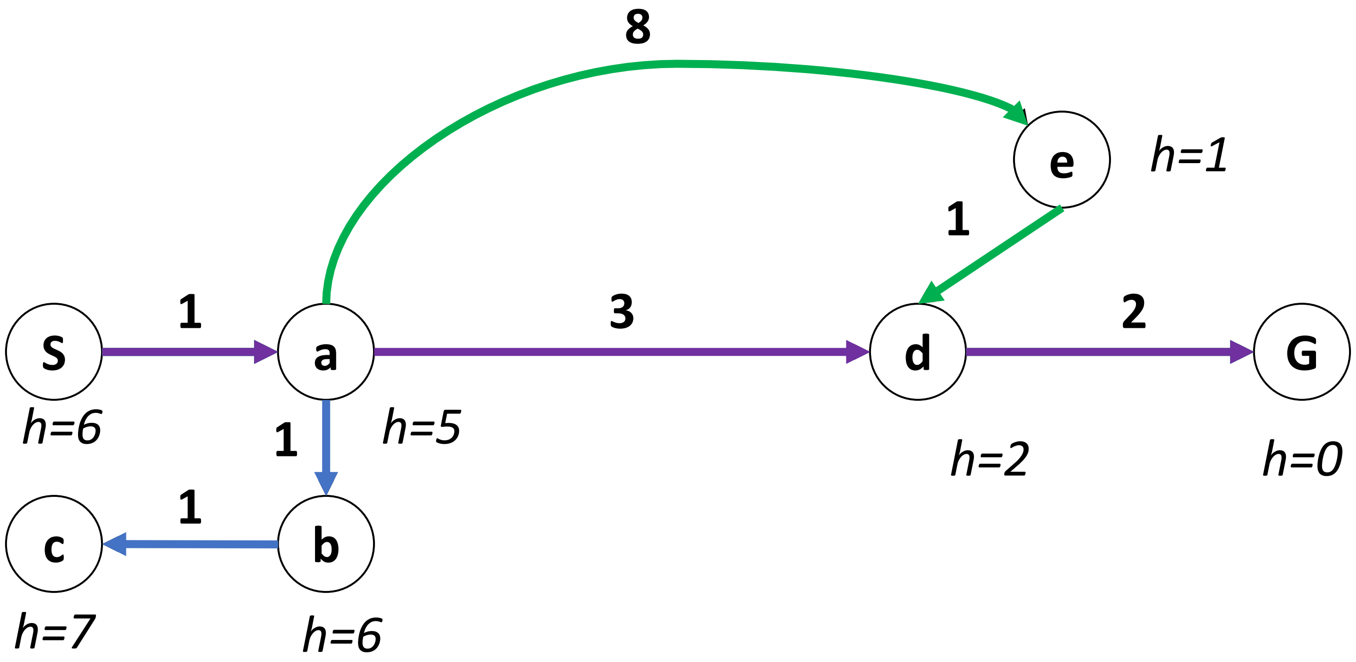
UCS

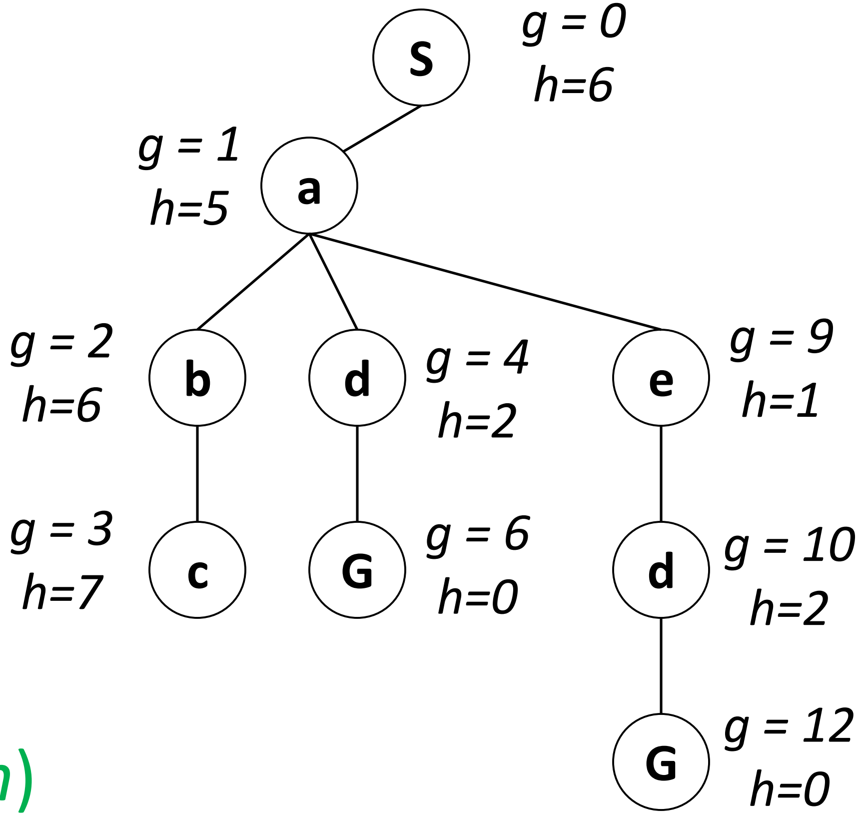
Greedy

A*

Combining UCS and Greedy

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

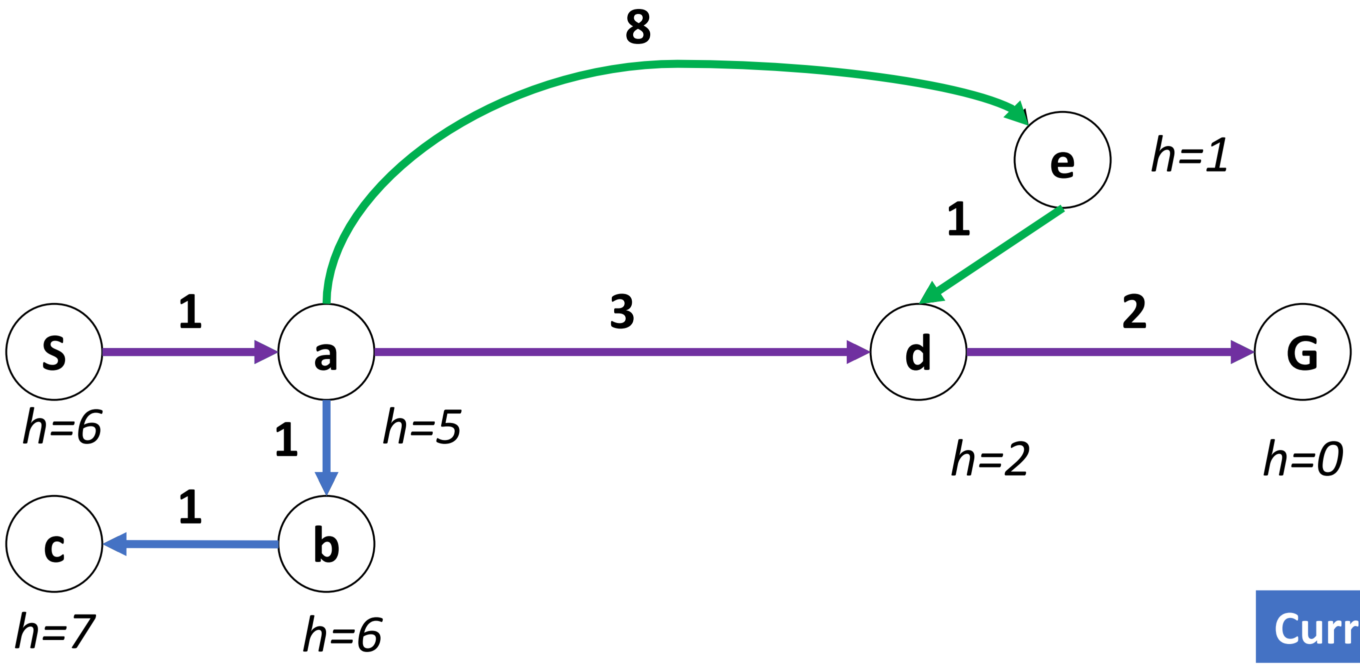




- A* Search orders by the sum: $f(n) = g(n) + h(n)$

Example: Teg Grenager

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

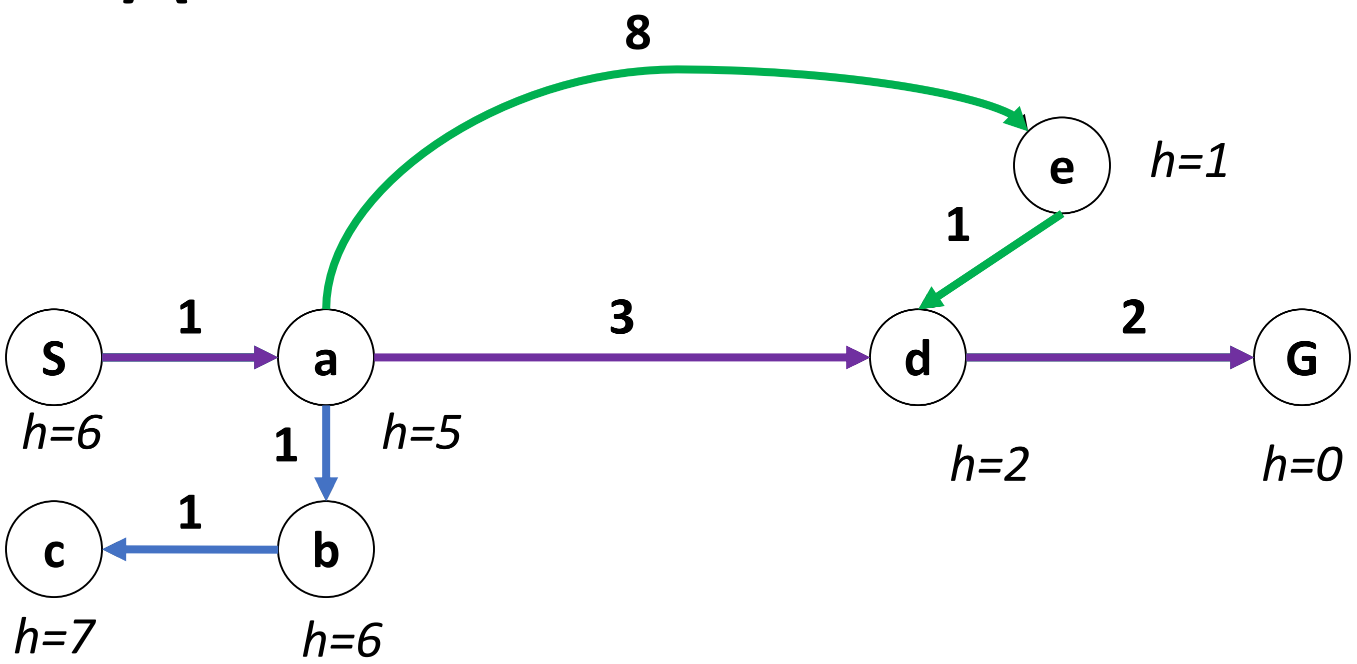


Current	Open list (Greedy)	Close
S(6)	A(5)	-
A(5)	E(1), D(2), B(6)	S(6)
E(1)	D(2), B(6)	S(6), A(5)
D(2)	G(0), B(6)	S(6), A(5), E(1)
G(0)	Goal	S(6), A(5), E(1), D(2)
Path =	S-A-E-D-G	
Cost =	1+8+1+2 = 12	

Current	Open list (A*)	Close
S(0+6=6)	A(1+5=6)	
A(6)	D(6), B(8), E(10)	S(6)
D(6)	G(6), B(8), E(10)	S(6), A(6)
G(6) = goal node	B(8), E(10)	S(6), A(6), D(6)
Path =	S-A-D-G	
Cost = 6		

Current	Open list (UCS)	Close
S(0)	A(1)	
A(1)	B(2), D(4), E(9)	S(0)
B(2)	C(3) , D(4), E(9)	S(0), A(1)
C(3)	D(4), E(9)	S(0), A(1), B(2)
D(4)	G(6), E(9)	S(0), A(1), B(2), C(3)
G(6) = goal node	E(9)	S(0), A(1), B(2), C(3), D(4)

A*



```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
    initialize the explored set to be empty
    initialize the frontier as a priority queue using g(n) as the priority
    add initial state of problem to frontier with priority g(S) = 0
    loop do
        if the frontier is empty then
            return failure
        choose a node and remove it from the frontier
        if the node contains a goal state then
            return the corresponding solution
        add the node state to the explored set
        for each resulting child from node
            if the child state is not already in the frontier or explored set then
                add child to the frontier
            else if the child is already in the frontier with higher g(n) then
                replace that frontier node with child
```

```
function A-STAR-SEARCH(problem) returns a solution, or failure
    initialize the explored set to be empty
    initialize the frontier as a priority queue using f(n) = g(n) + h(n) as the priority
    add initial state of problem to frontier with priority f(S) = 0 + h(S)
    loop do
        if the frontier is empty then
            return failure
        choose a node and remove it from the frontier
        if the node contains a goal state then
            return the corresponding solution
        add the node state to the explored set
        for each resulting child from node
            if the child state is not already in the frontier or explored set then
                add child to the frontier
            else if the child is already in the frontier with higher f(n) then
                replace that frontier node with child
```

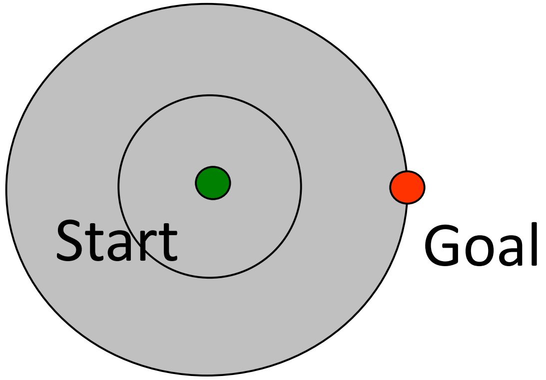
A* Search Algorithms

- A* Tree Search
  - Same tree search algorithm but with a frontier that is a priority queue using priority $f(n) = g(n) + h(n)$

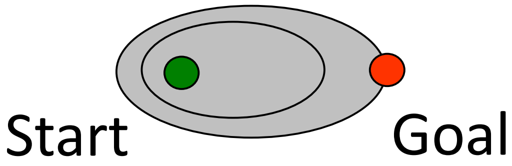
- A* Graph Search
  - Same as **UCS** graph search algorithm but with a frontier that is a priority queue using priority $f(n) = g(n) + h(n)$

UCS vs A* Contours

UCS vs A* Contours



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



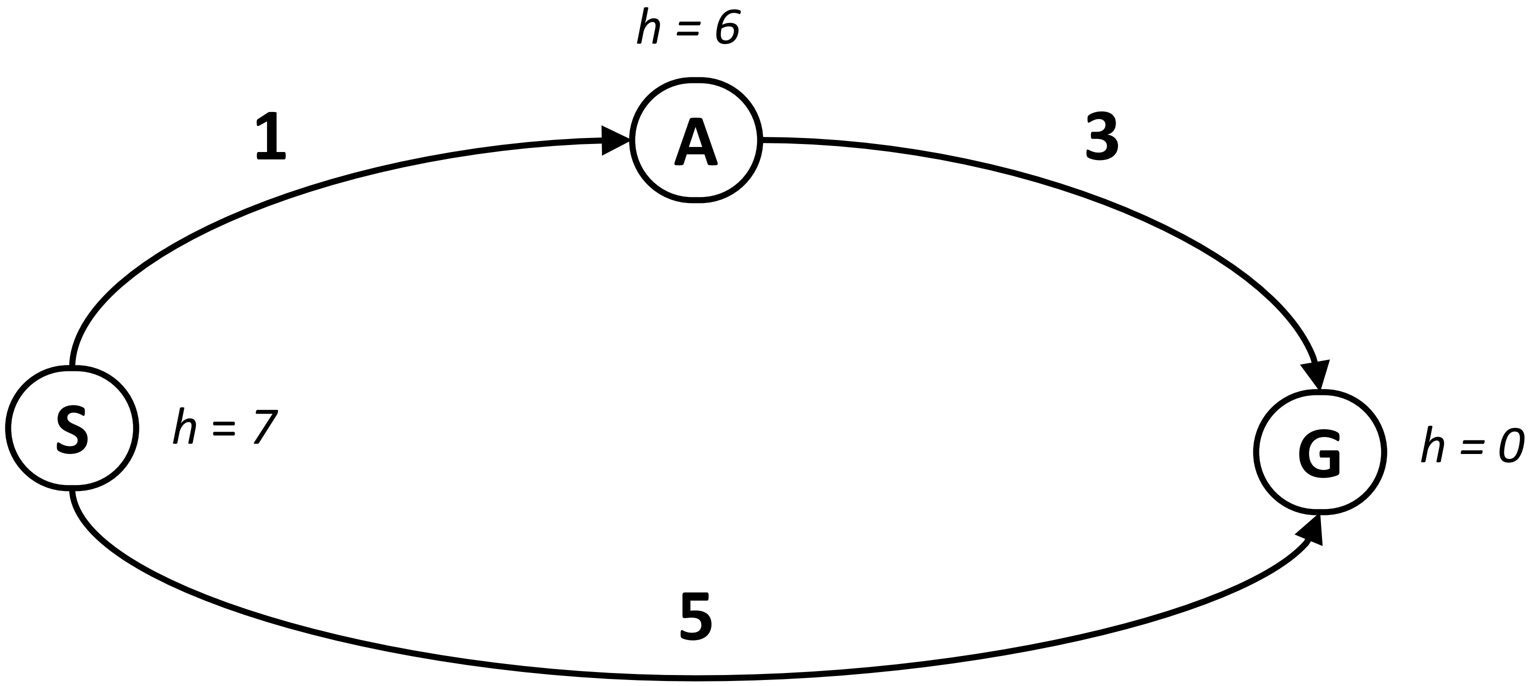
Comparison

Greedy

Uniform Cost

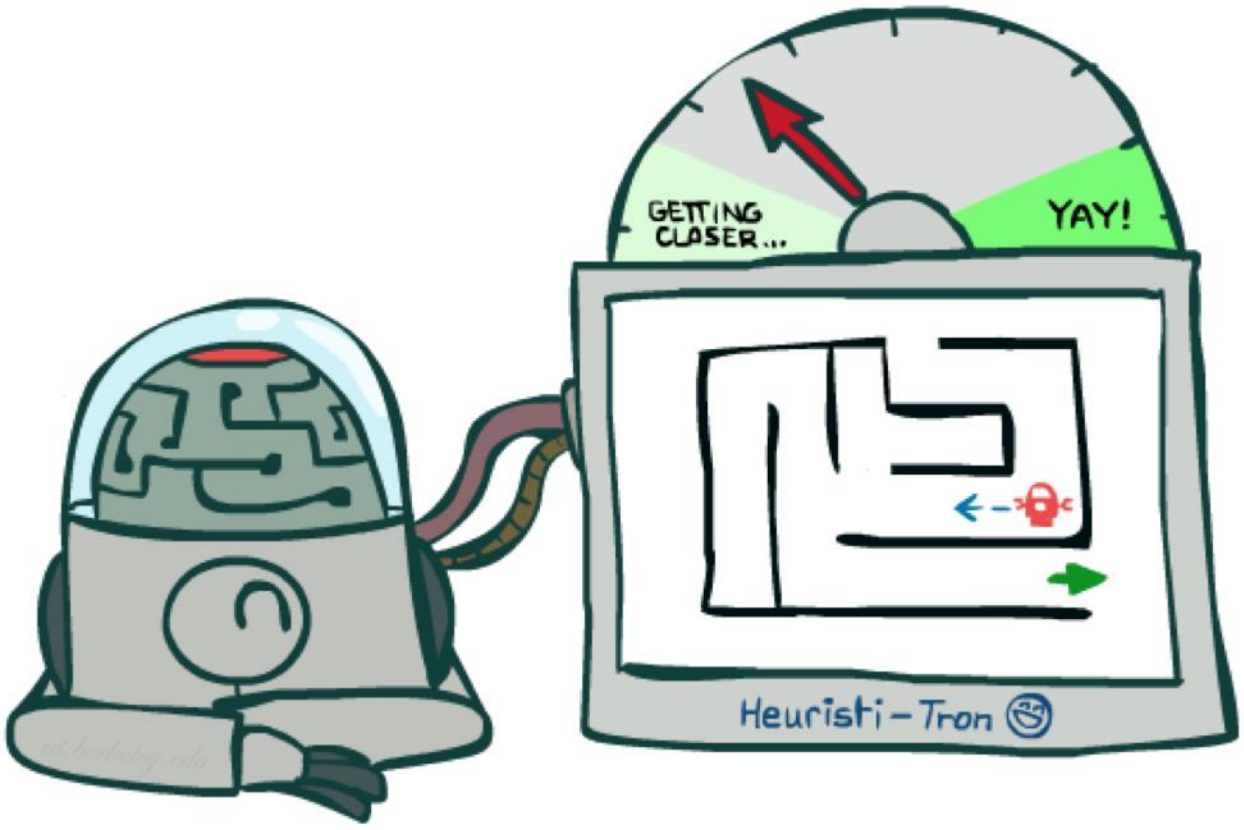
A*

Is A* Optimal?



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

Admissible Heuristics



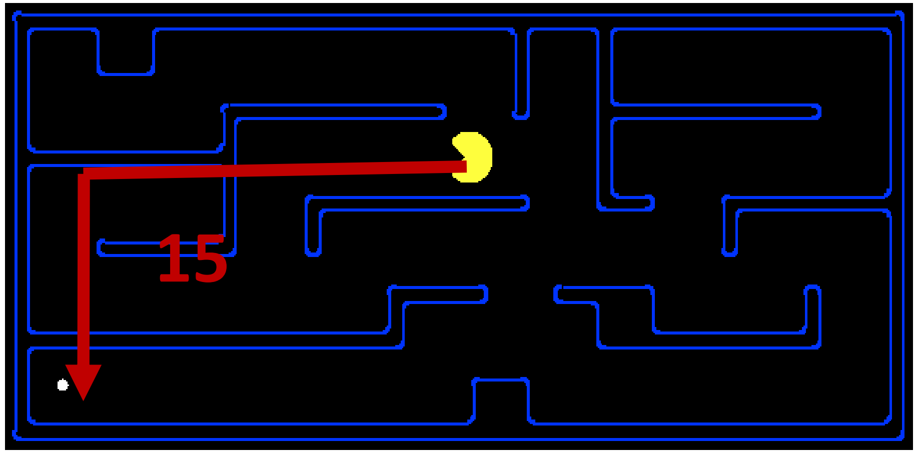
Admissible Heuristics

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

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

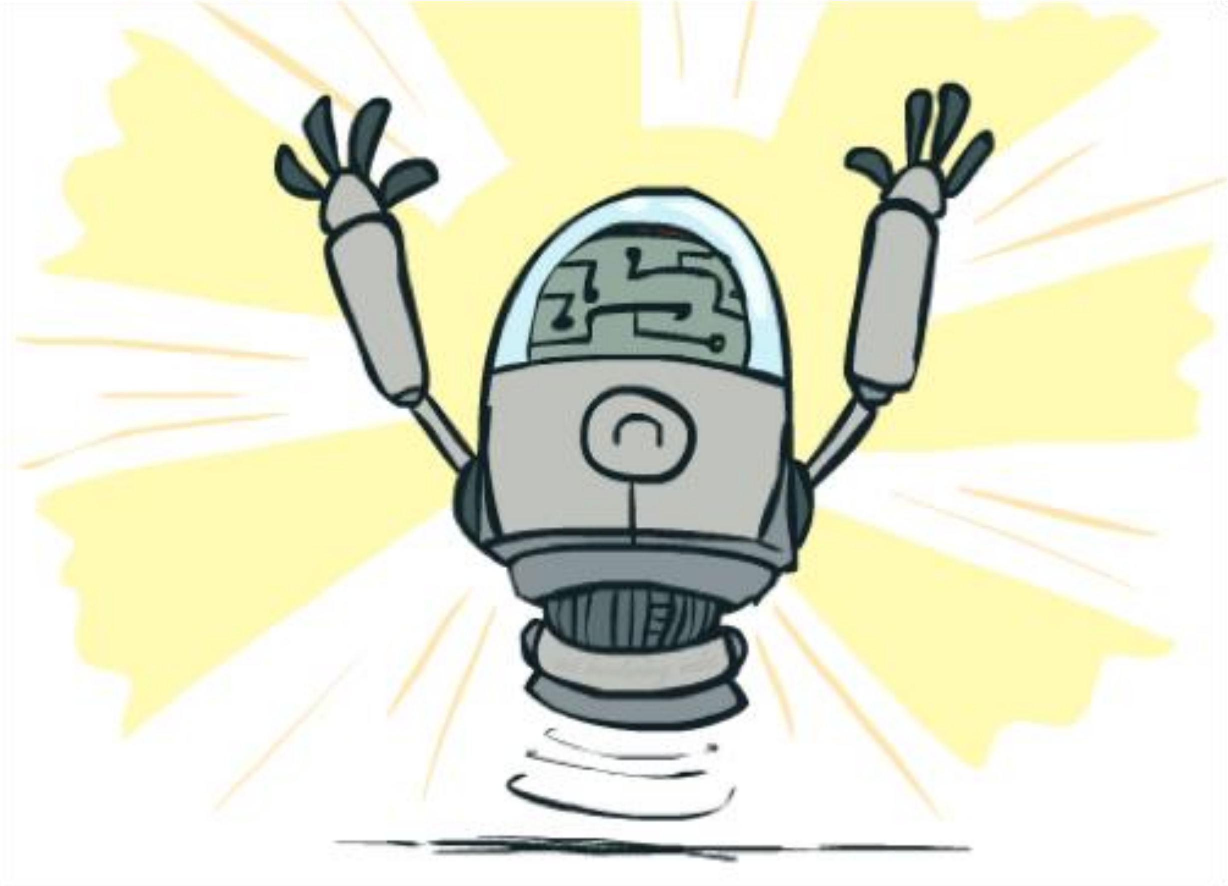
  where $h^*(n)$ is the true cost to a nearest goal

- Example:



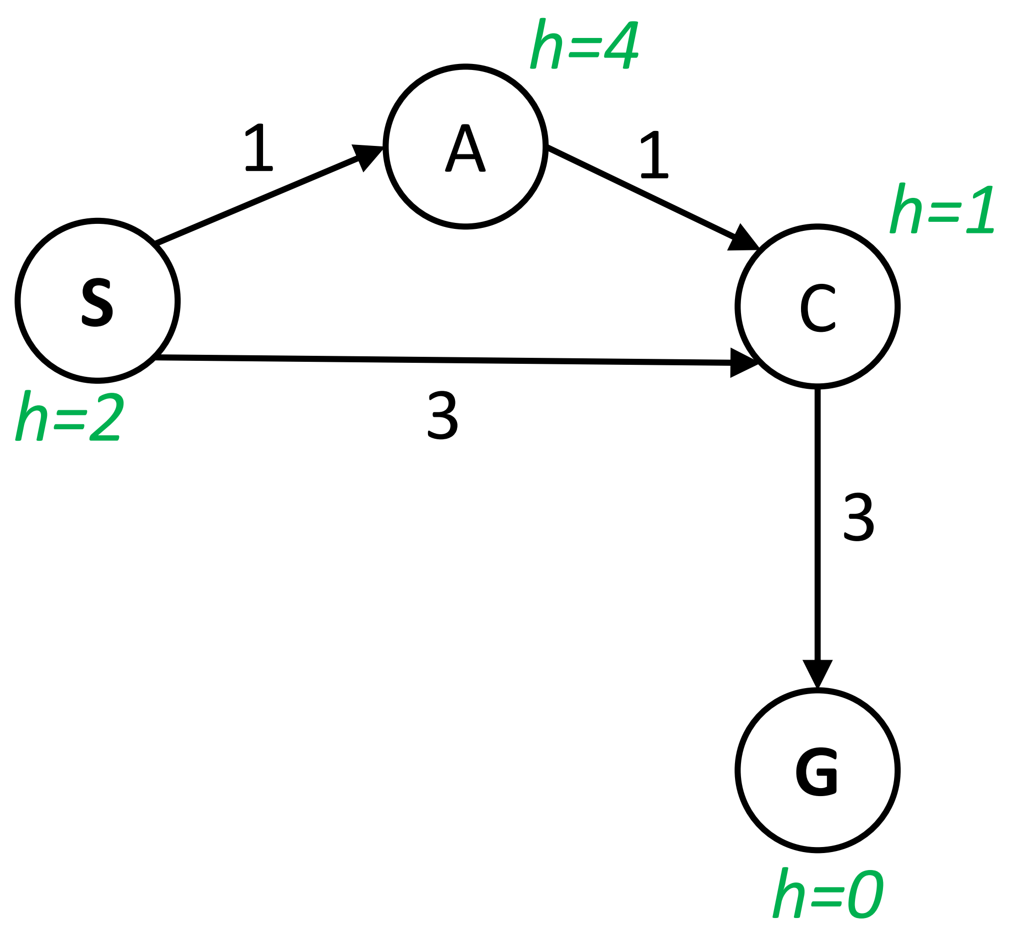
- Coming up with admissible heuristics is most of what's involved in using A* in practice.

Optimality of A* Tree Search

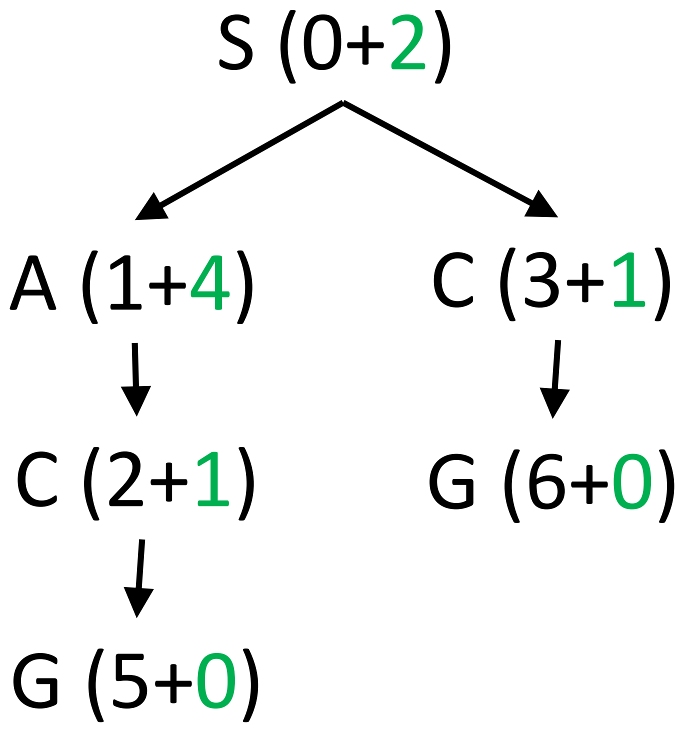


A* Tree Search

State space graph



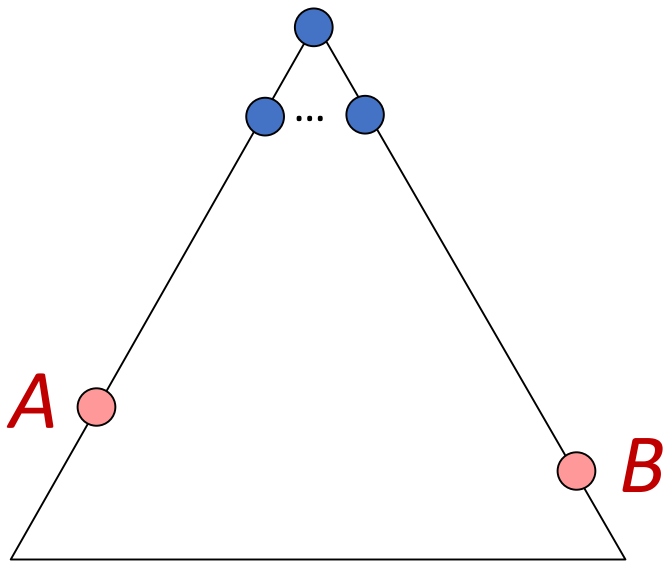
Search tree



Optimality of A* Tree Search

Assume:

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



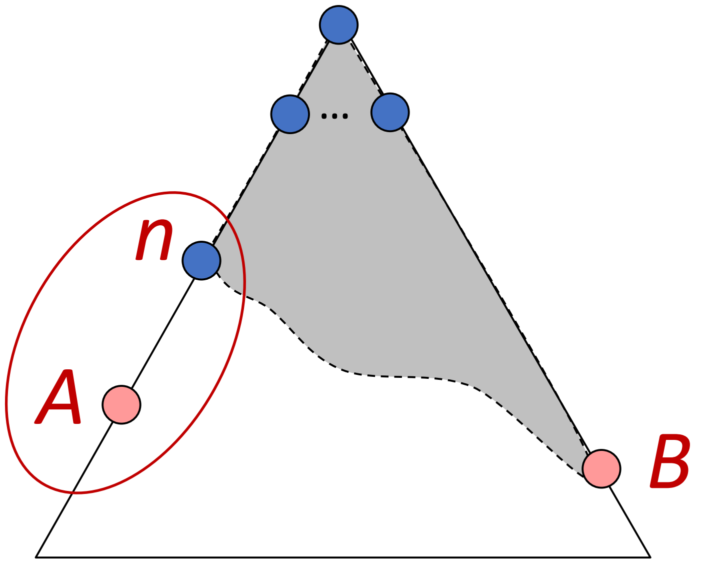
Claim:

- *A* will be chosen for exploration (popped off the frontier) before *B*

Optimality of A* Tree Search: Blocking

Proof:

- Imagine *B* is on the frontier
- Some ancestor *n* of *A* is on the frontier, too (Maybe the start state; maybe *A* itself!)
- Claim: *n* will be explored before *B*
  1. $f(n)$ is less than or equal to $f(A)$

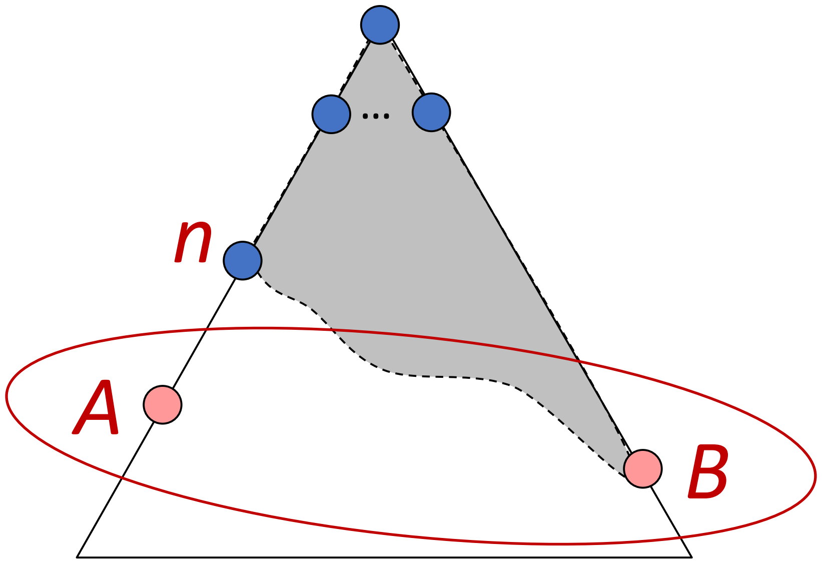


$f(n) = g(n) + h(n)$	Definition of f-cost
$f(n) \leq g(A)$	Admissibility of h
$g(A) = f(A)$	$h = 0$ at a goal

Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the frontier
- Some ancestor n of A is on the frontier, too (Maybe the start state; maybe A itself!)
- Claim: n will be explored before B
  1. $f(n)$ is less than or equal to $f(A)$
  2. $f(A)$ is less than $f(B)$



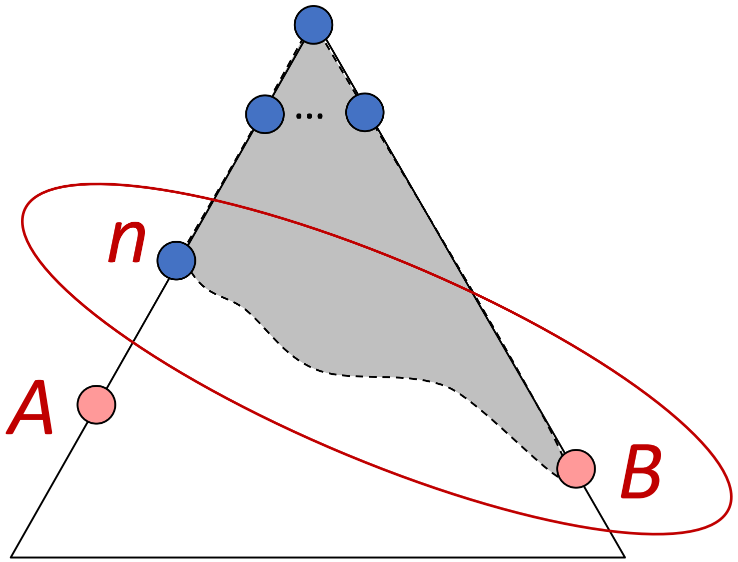
$g(A) < g(B)$ Suboptimality of B

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

Optimality of A* Tree Search: Blocking

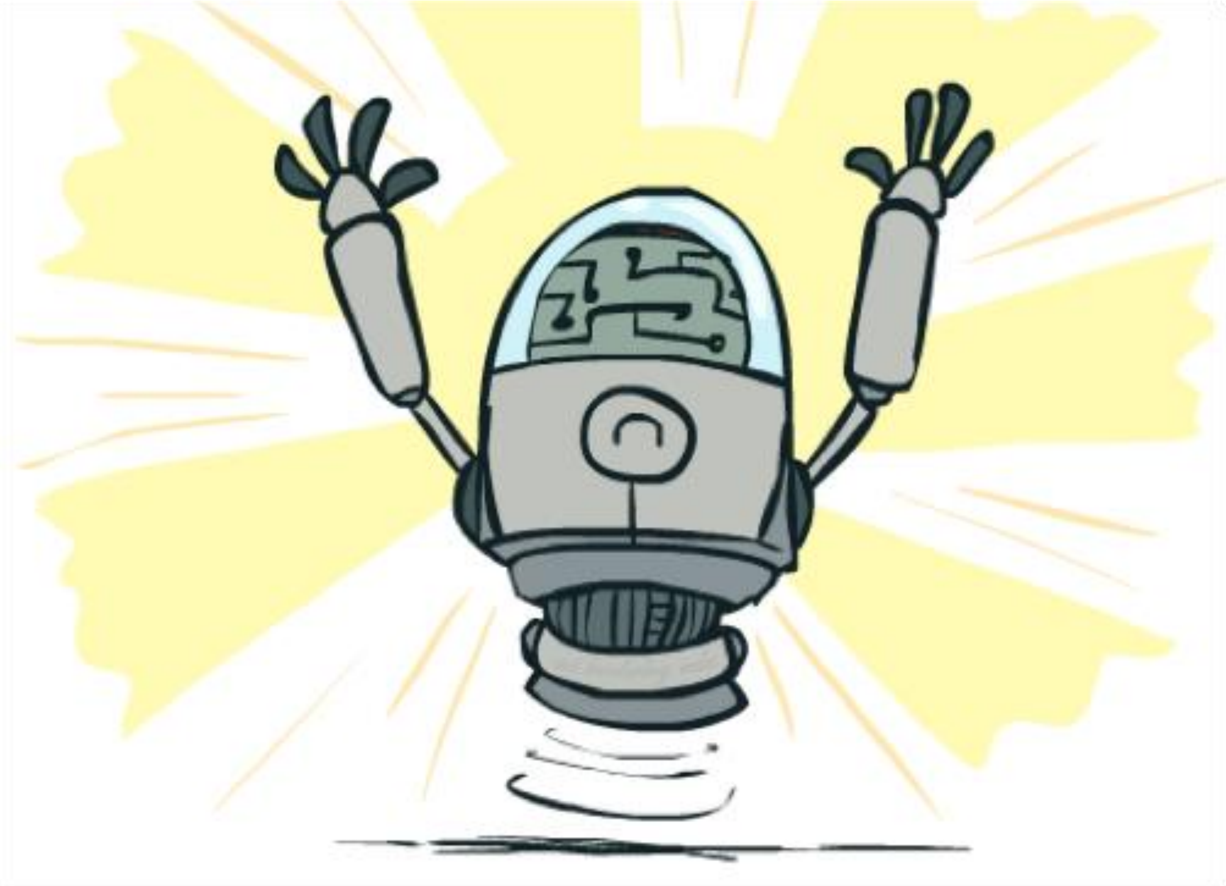
Proof:

- Imagine *B* is on the frontier
- Some ancestor *n* of *A* is on the frontier, too
  (Maybe the start state; maybe *A* itself!)
- Claim: *n* will be explored before *B*
  1. $f(n)$ is less than or equal to $f(A)$
  2. $f(A)$ is less than $f(B)$
  3. *n* is explored before *B*
- All ancestors of *A* are explored before *B*
- *A* is explored before *B*
- **A* search is optimal**



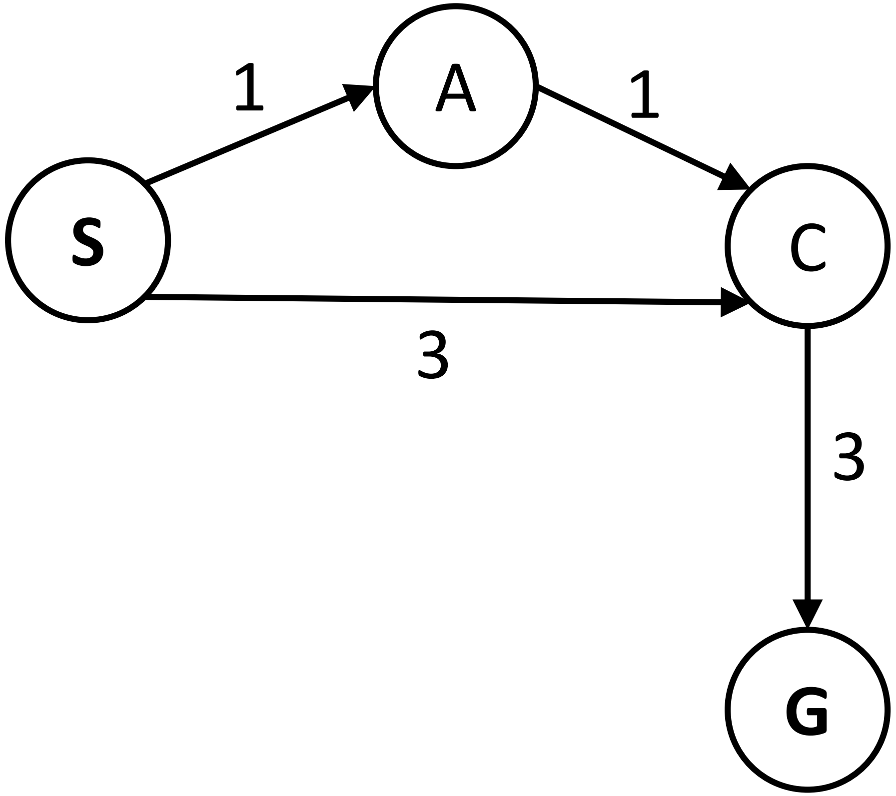
$f(n) \leq f(A) < f(B)$

Optimality of A* Graph Search



POLL 1

What paths does A* graph search consider during its search?



A) ~~S~~, ~~S-A~~, ~~S-C~~, S-C-G

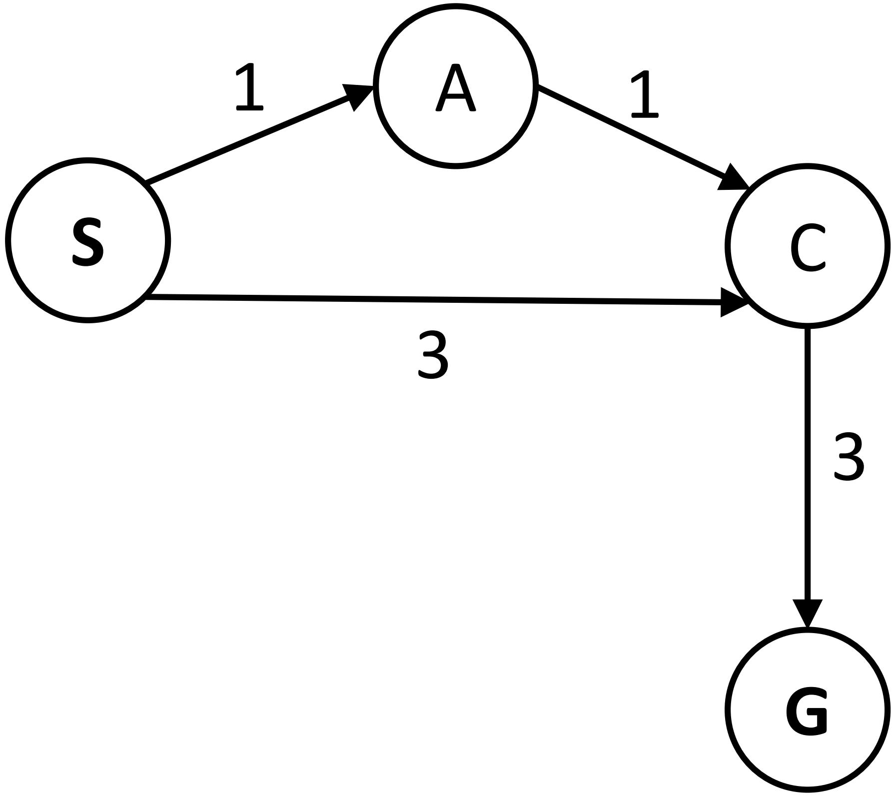
B) ~~S~~, ~~S-A~~, S-C, ~~S-A-C~~, S-C-G

C) ~~S~~, ~~S-A~~, ~~S-A-C~~, S-A-C-G

D) ~~S~~, ~~S-A~~, ~~S-C~~, ~~S-A-C~~, S-A-C-G

1-answer

What paths does A* graph search consider during its search?



A) ~~S~~, ~~S-A~~, ~~S-C~~, S-C-G

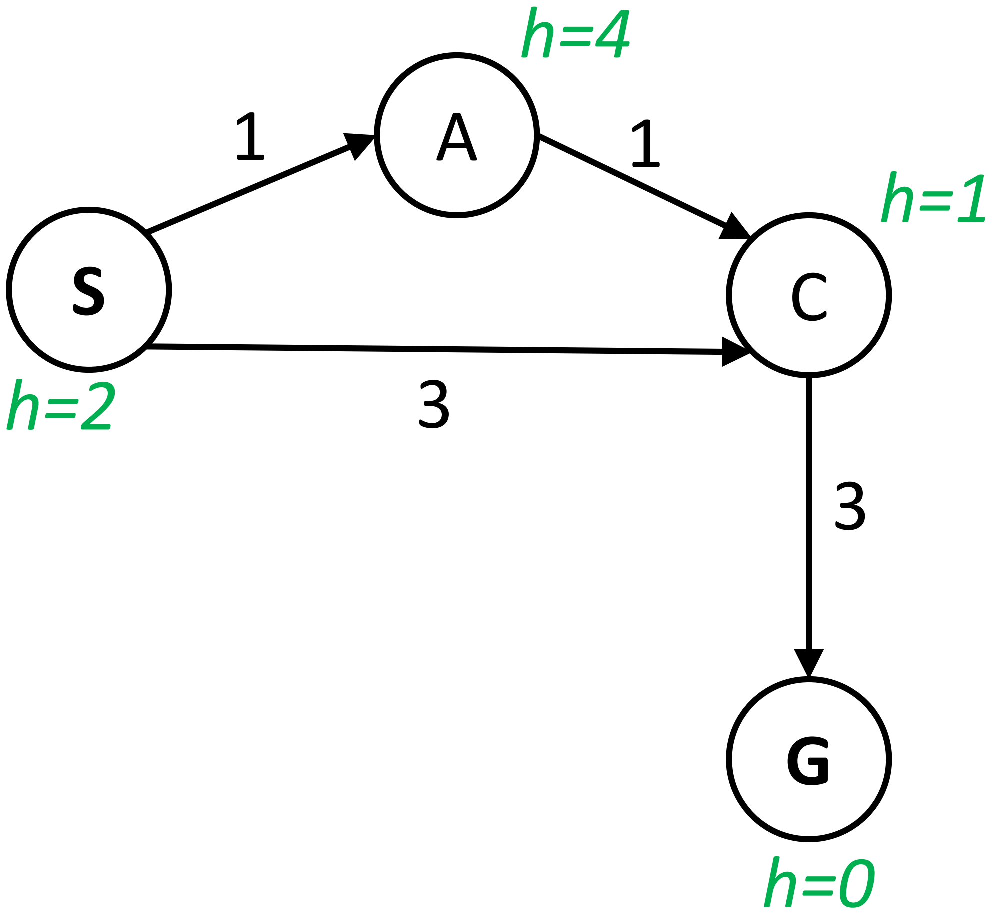
~~B) S, S-A, S-C, S-A-C, S-C-G~~

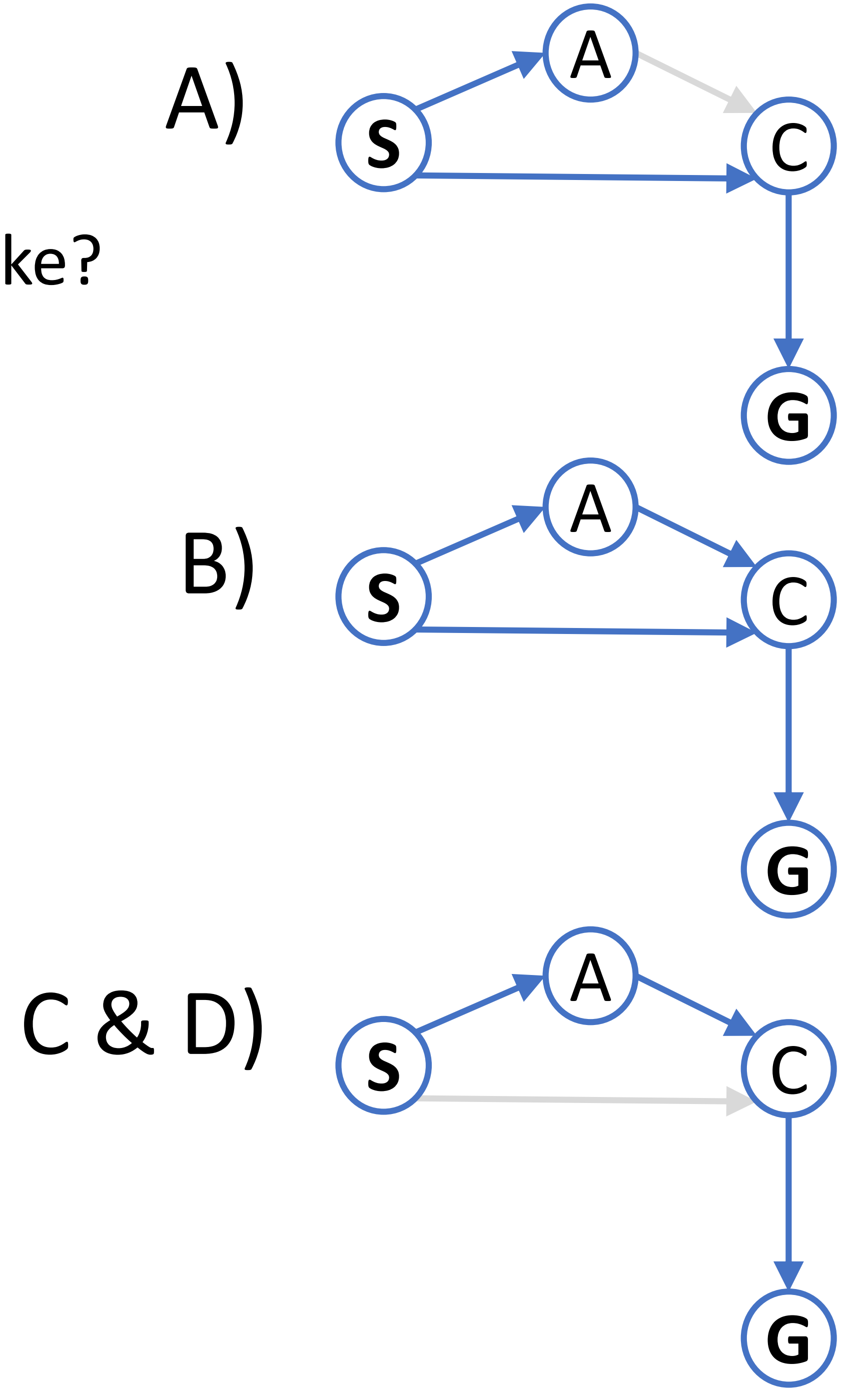
~~C) S, S-A, S-A-C, S-A-C-G~~

~~D) S, S-A, S-C, S-A-C, S-A-C-G~~

A* Graph Search

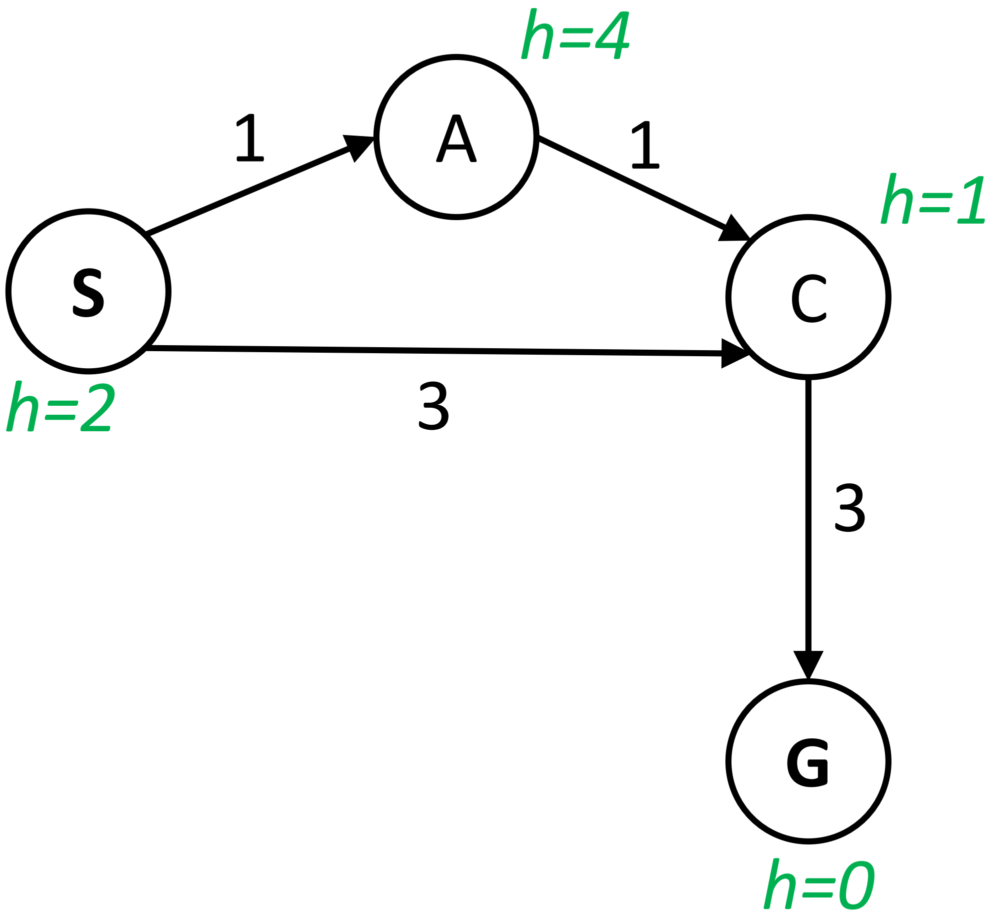
What does the resulting graph tree look like?



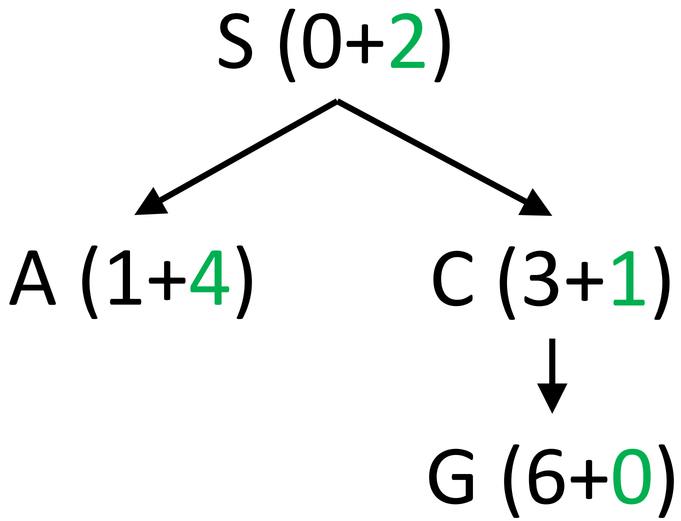


A* Graph Search Gone Wrong?

State space graph



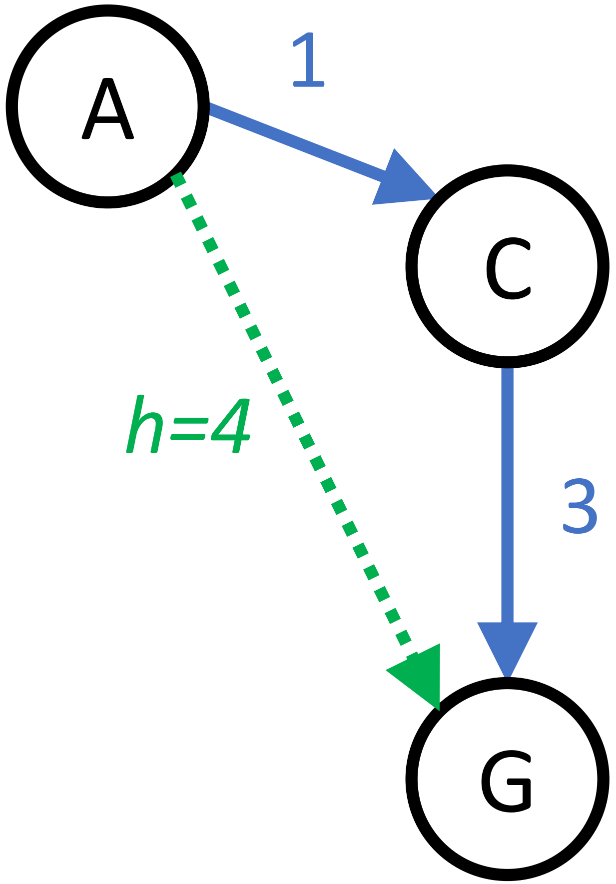
Search tree



Simple check against explored set blocks C

Fancy check allows new C if cheaper than old but requires recalculating C’s descendants

Admissibility of Heuristics

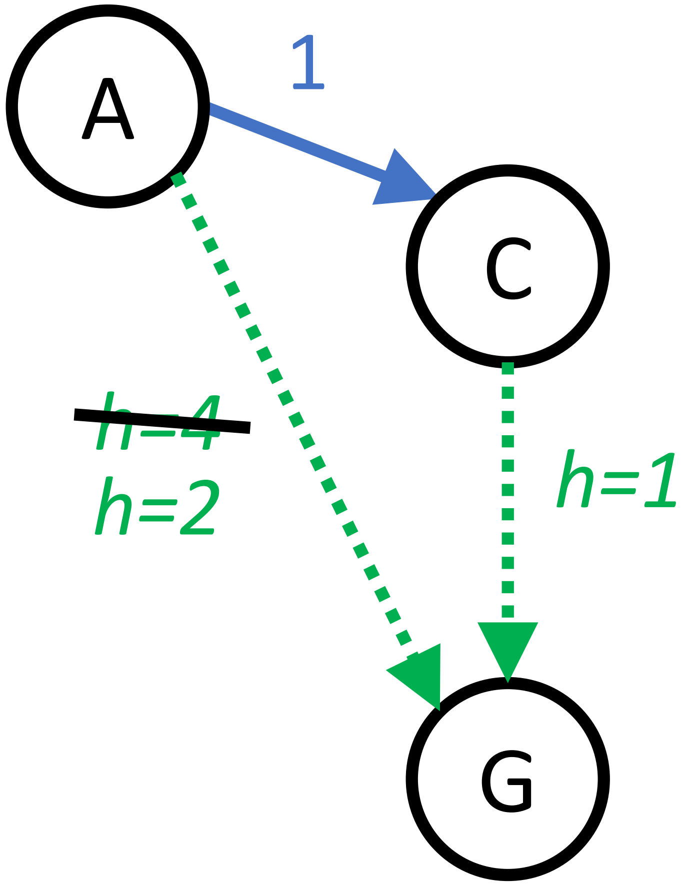


- Main idea: Estimated heuristic values ≤ actual costs
  - Admissibility:

    heuristic value ≤ actual cost to goal

    h(A) ≤ actual cost from A to G

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 step cost" ≤ actual cost for each step

    h(A) – h(C) ≤ cost(A to C)

    triangle inequality

    h(A) ≤ cost(A to C) + h(C)
- Consequences of consistency:
  - The f value along a path never decreases
  - A* graph search is optimal

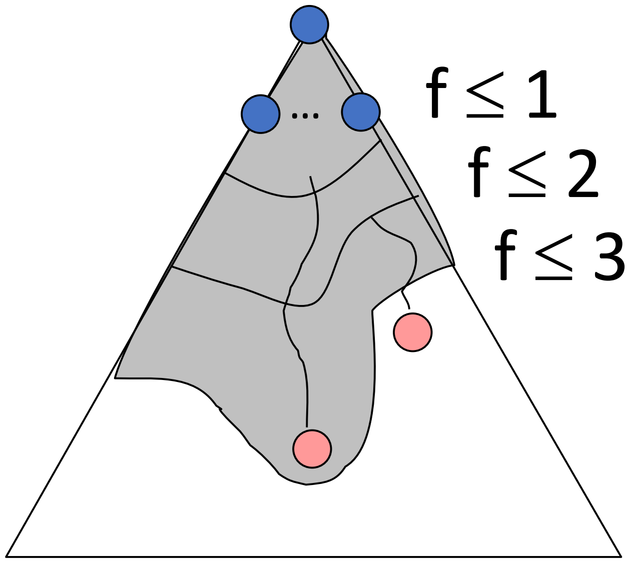
Optimality of A* Graph Search

- 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 explored before nodes that reach s suboptimally

  - Result: A* graph search is optimal



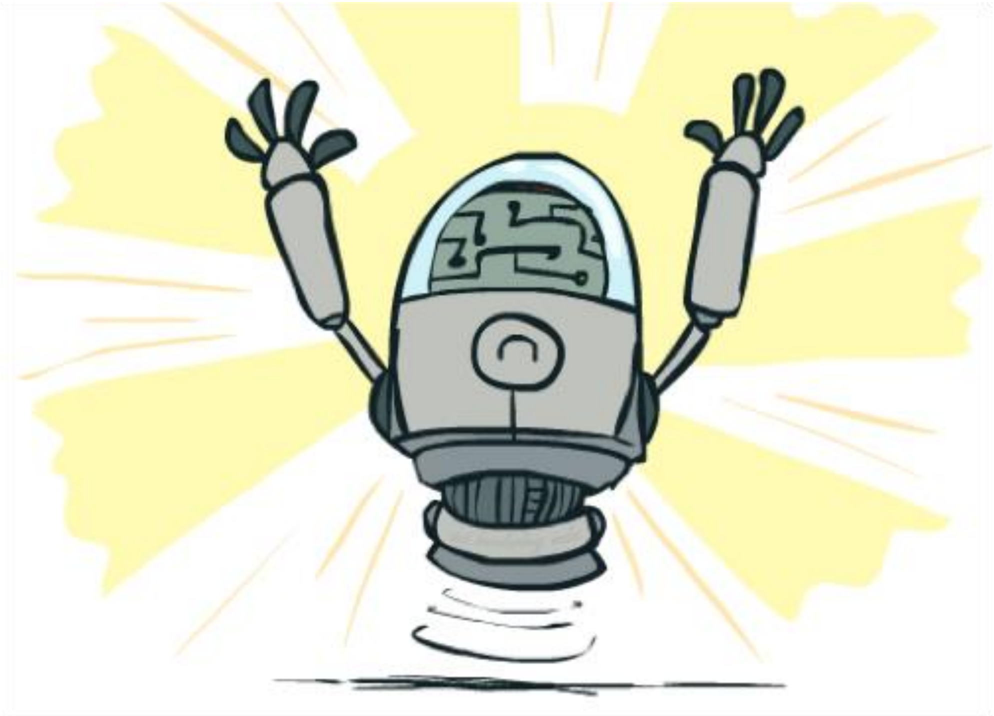
Optimality

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

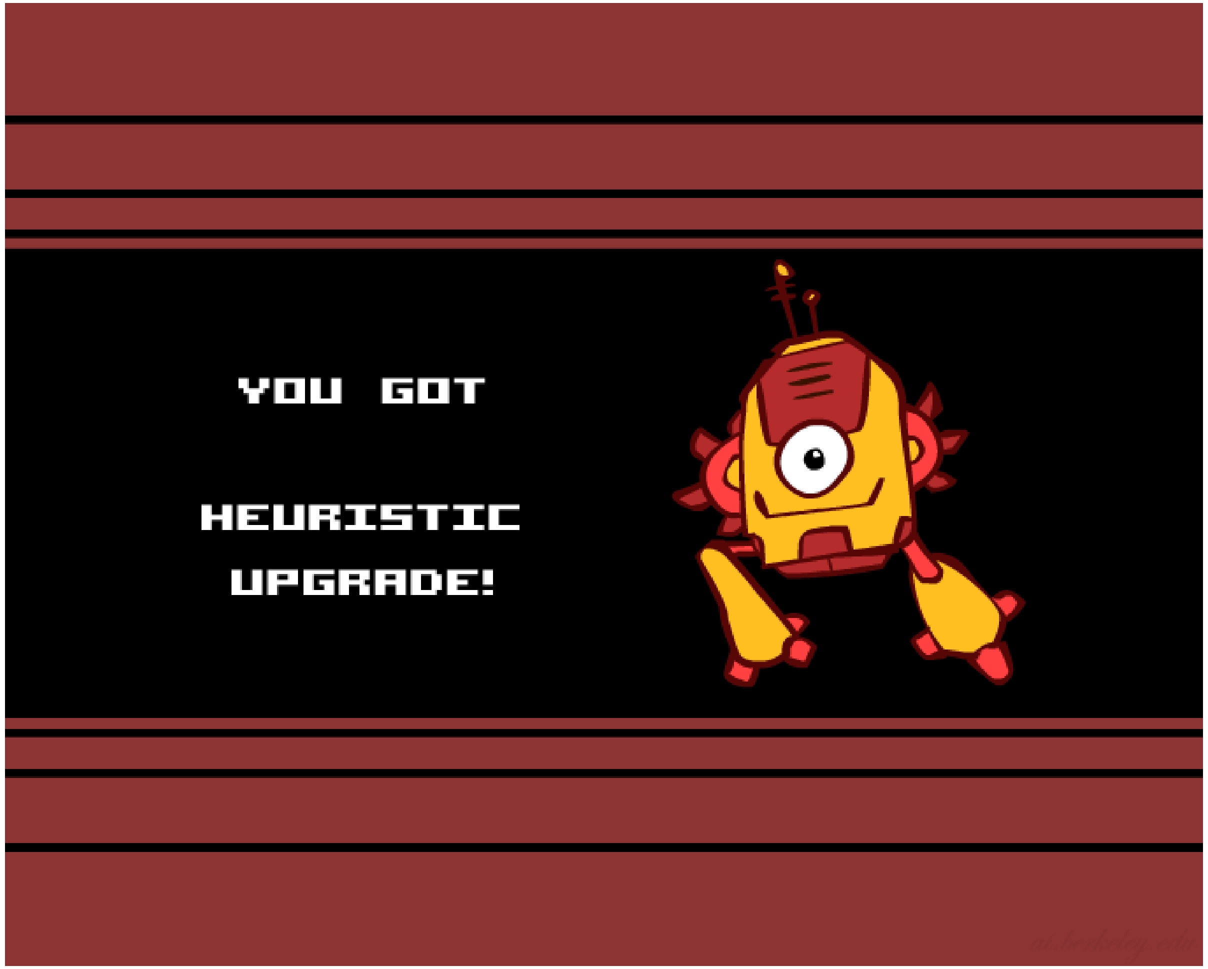
- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal (h = 0 is consistent)

- Consistency implies admissibility

- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems



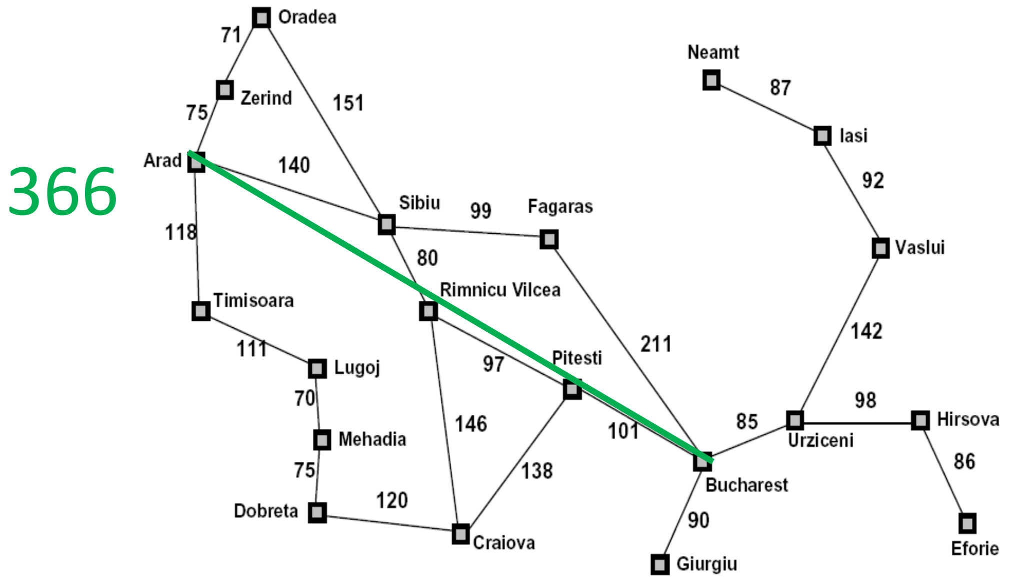
Creating Heuristics

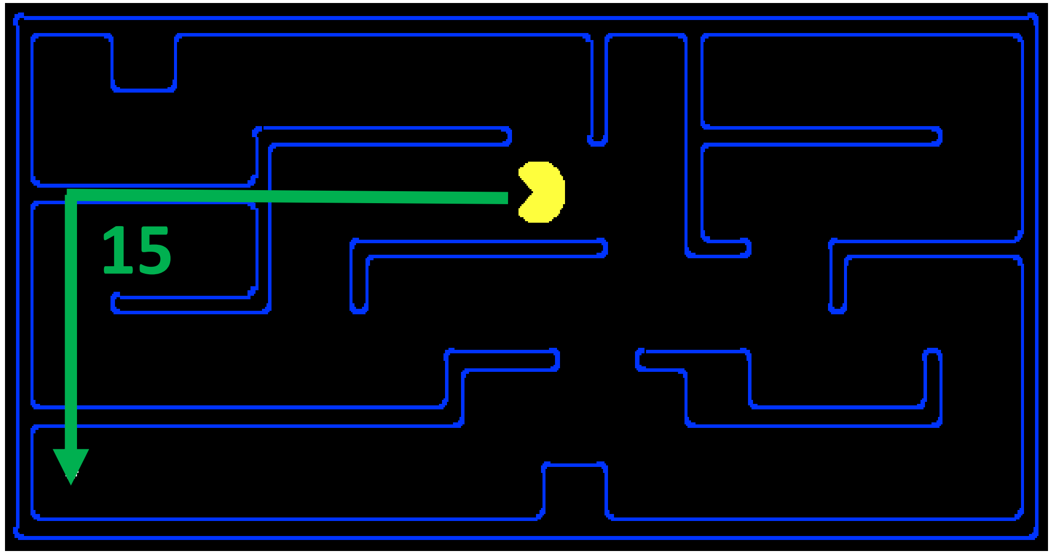


Creating Admissible Heuristics

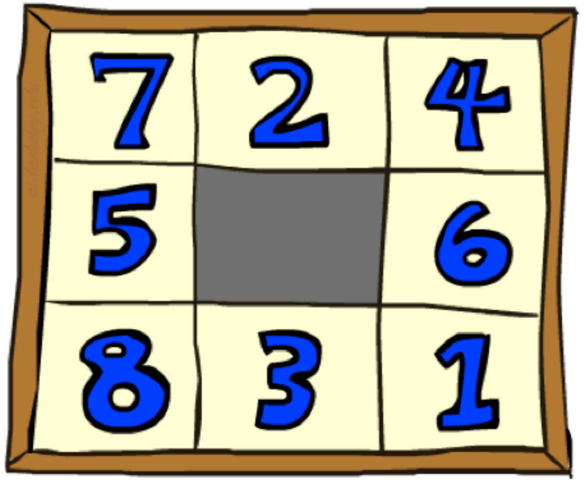
- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics

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

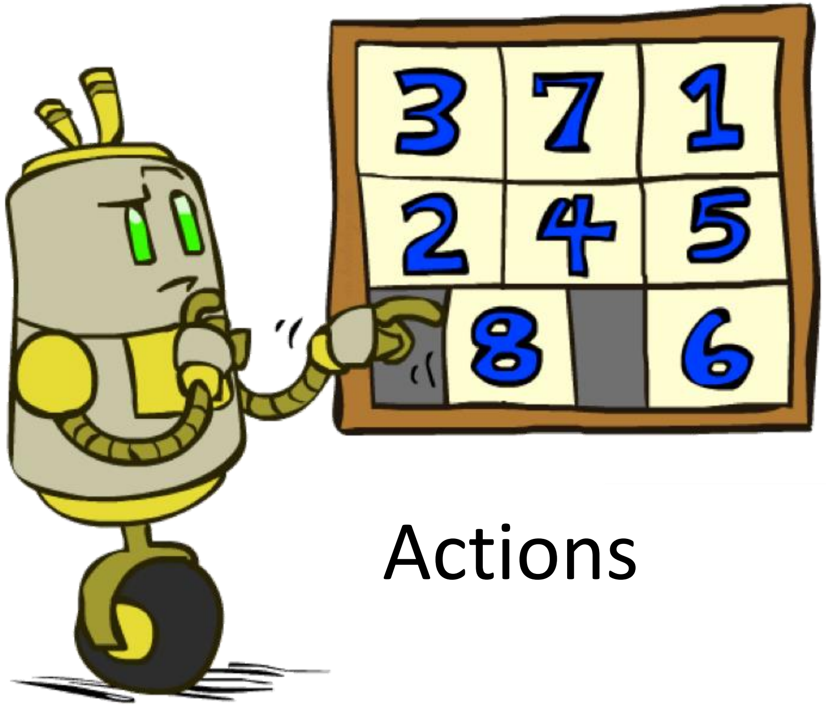




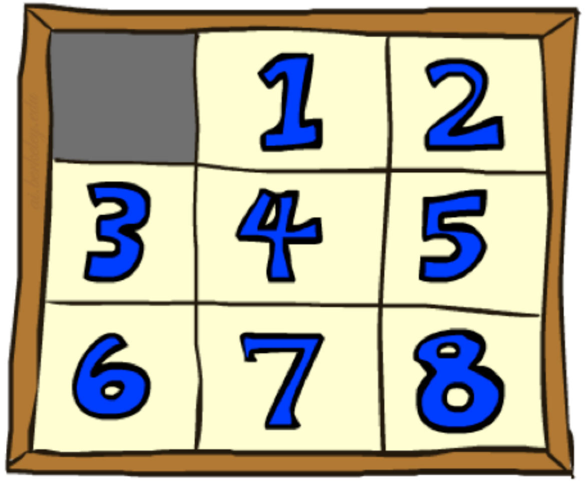
Example: 8 Puzzle



Start State



Actions

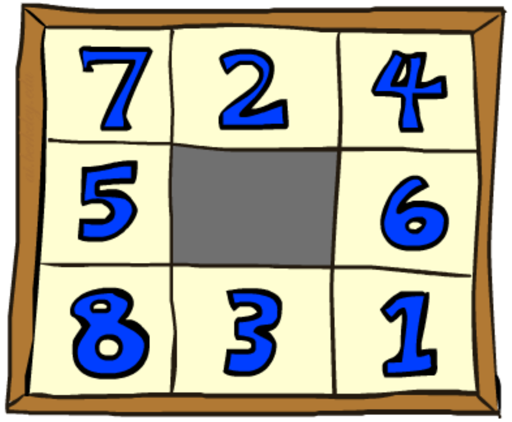


Goal State

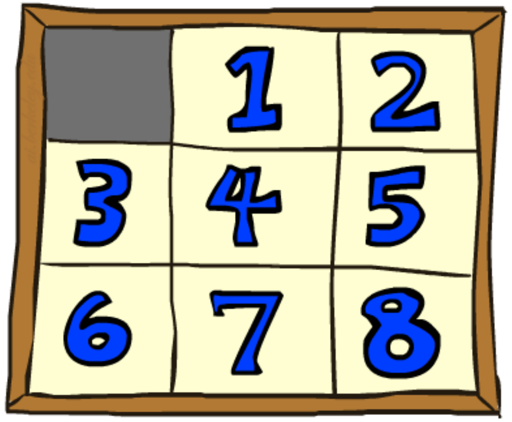
- What are the states?
- How many states?
- What are the actions?
- How many actions from the start state?
- What should the step costs be?

8 Puzzle I

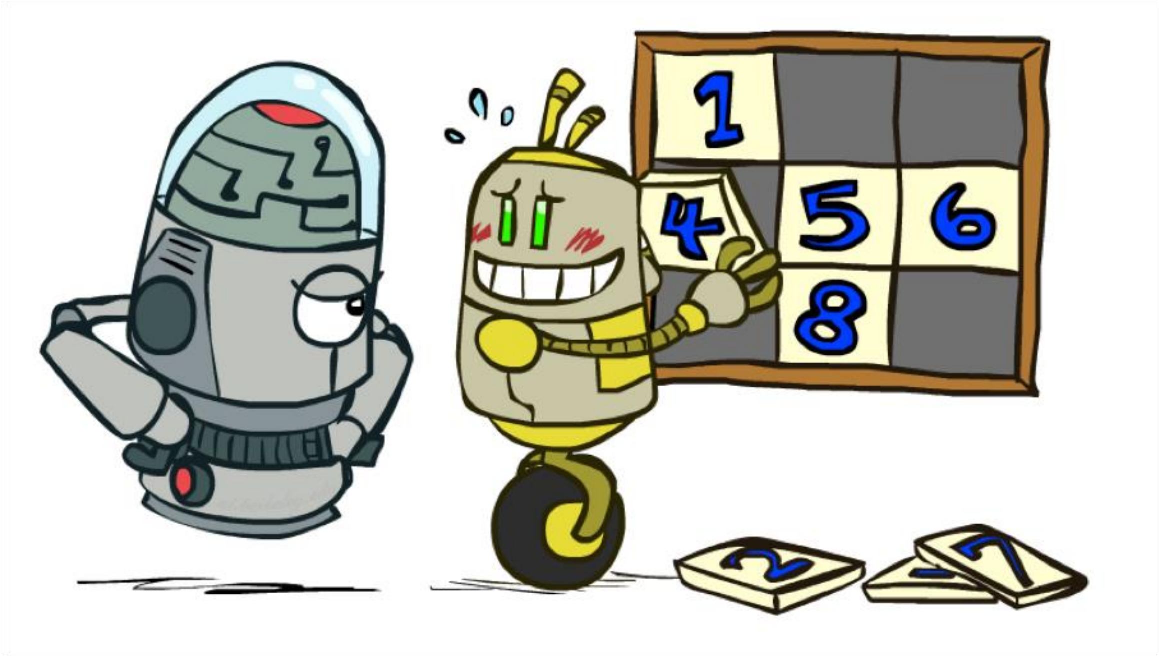
- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = 8
- This is a *relaxed-problem* heuristic



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
A*TILES	13	39	227

Statistics from Andrew Moore

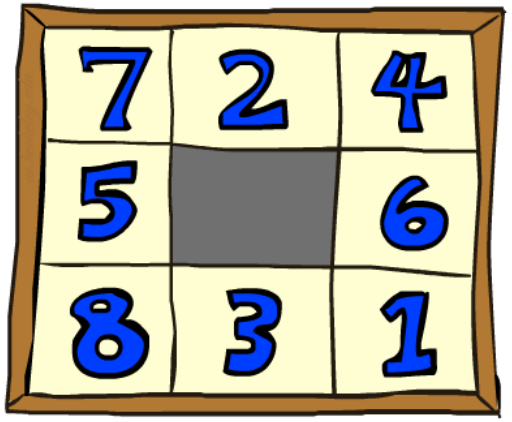
8 Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?

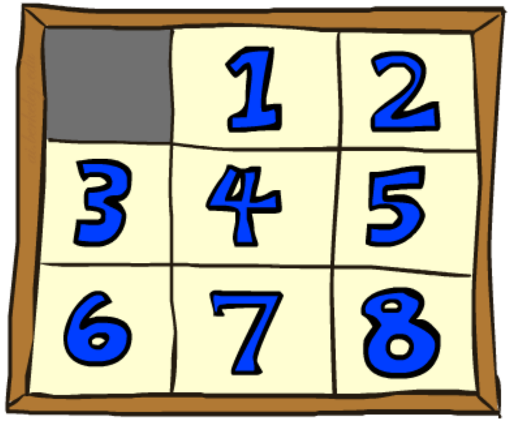
- Total *Manhattan* distance

- Why is it admissible?

- h(start) = 3 + 1 + 2 + … = 18



Start State



Goal State

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

Combining heuristics

- Dominance: $h_a \geq h_c$ if

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

  - Roughly speaking, larger is better as long as both are admissible
  - The zero heuristic is pretty bad (what does A* do with h=0?)
  - The exact heuristic is pretty good, but usually too expensive!

- What if we have two heuristics, neither dominates the other?
  - Form a new heuristic by taking the max of both:

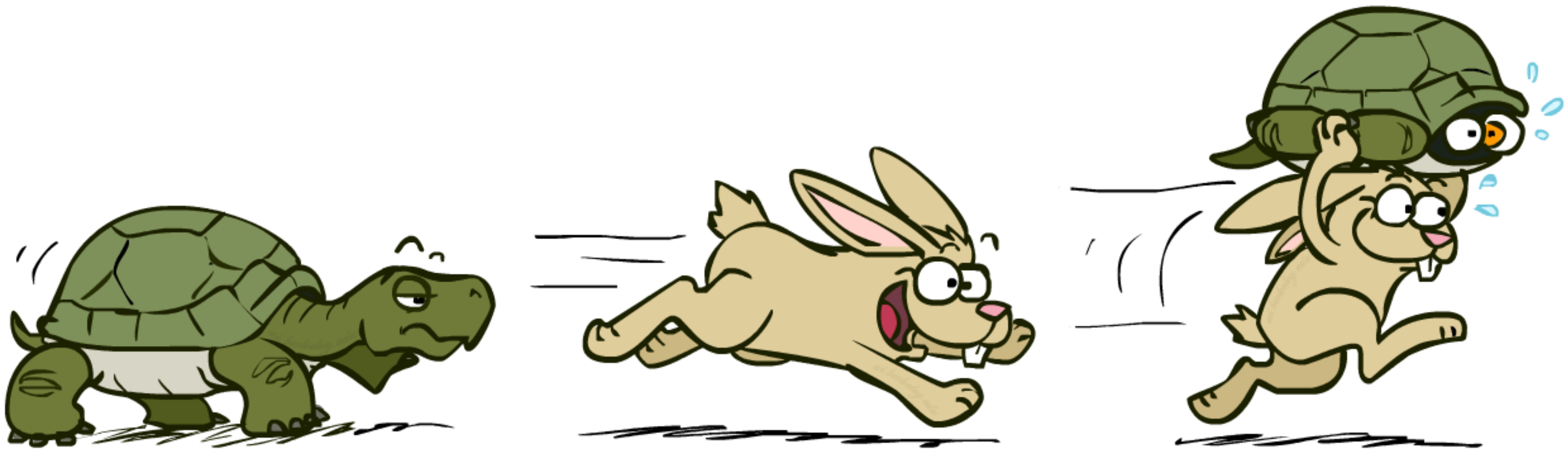
$$h(n) = \max(\, h_a(n), h_b(n) \,)$$

  - Max of admissible heuristics is admissible and dominates both!

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



In-Class Activity

- Q1: Practice creating heuristics and running Greedy and A* search
- Q2: Walk through Amazon Robot Example

Recommended Reading

- Notes added on LMS
- https://www.oreilly.com/library/view/graph-algorithms/9781492047674/ch04.html


QUESTIONS