Fundamentals of Artificial Intelligence

Lecture-3
Informed Search

Debela Desalegn

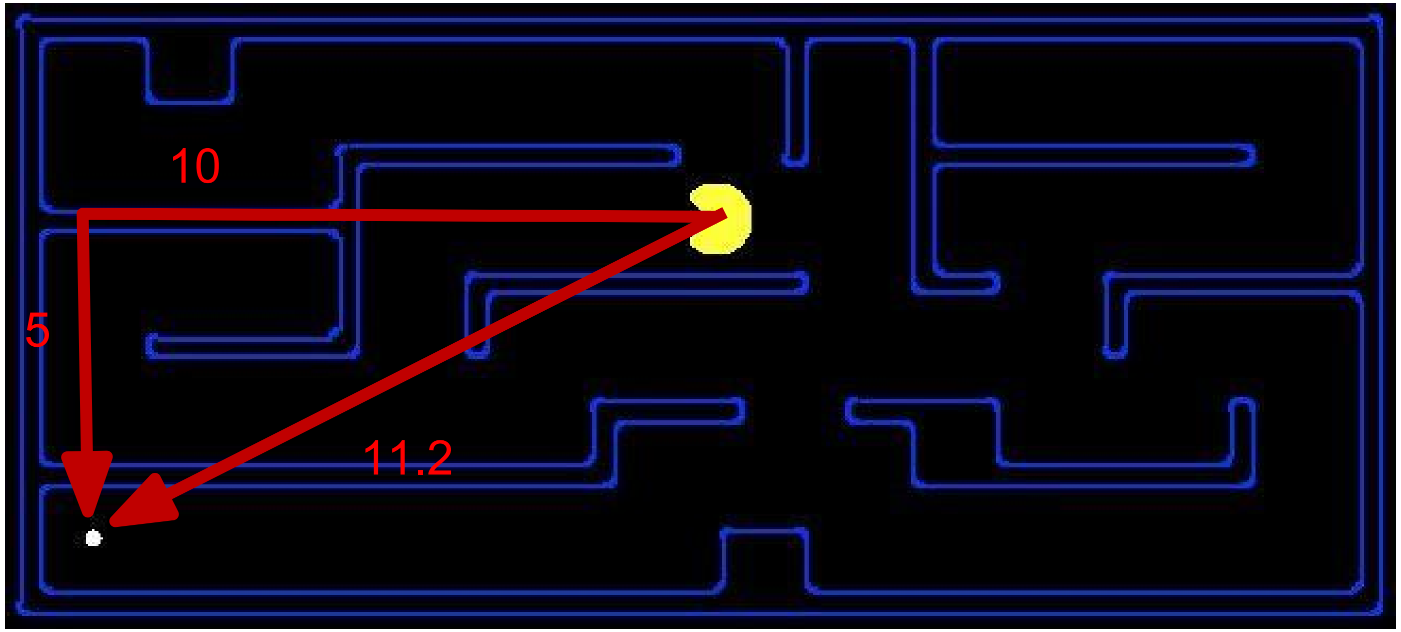
April 04, 2025

Informed Search

Heuristics:

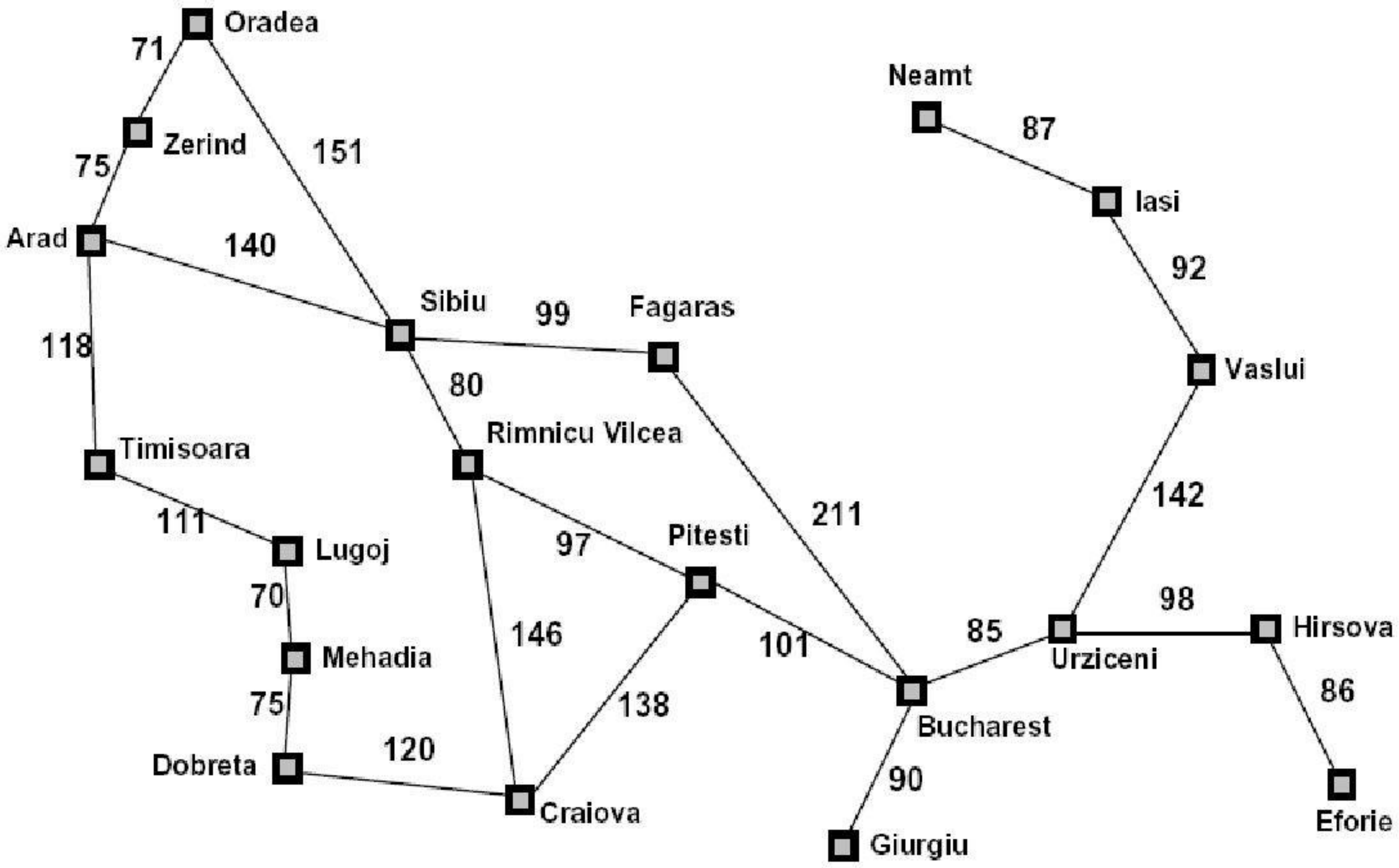
- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem

Examples: Manhattan distance, Euclidean distance for pathing



Informed Search

Heuristics:

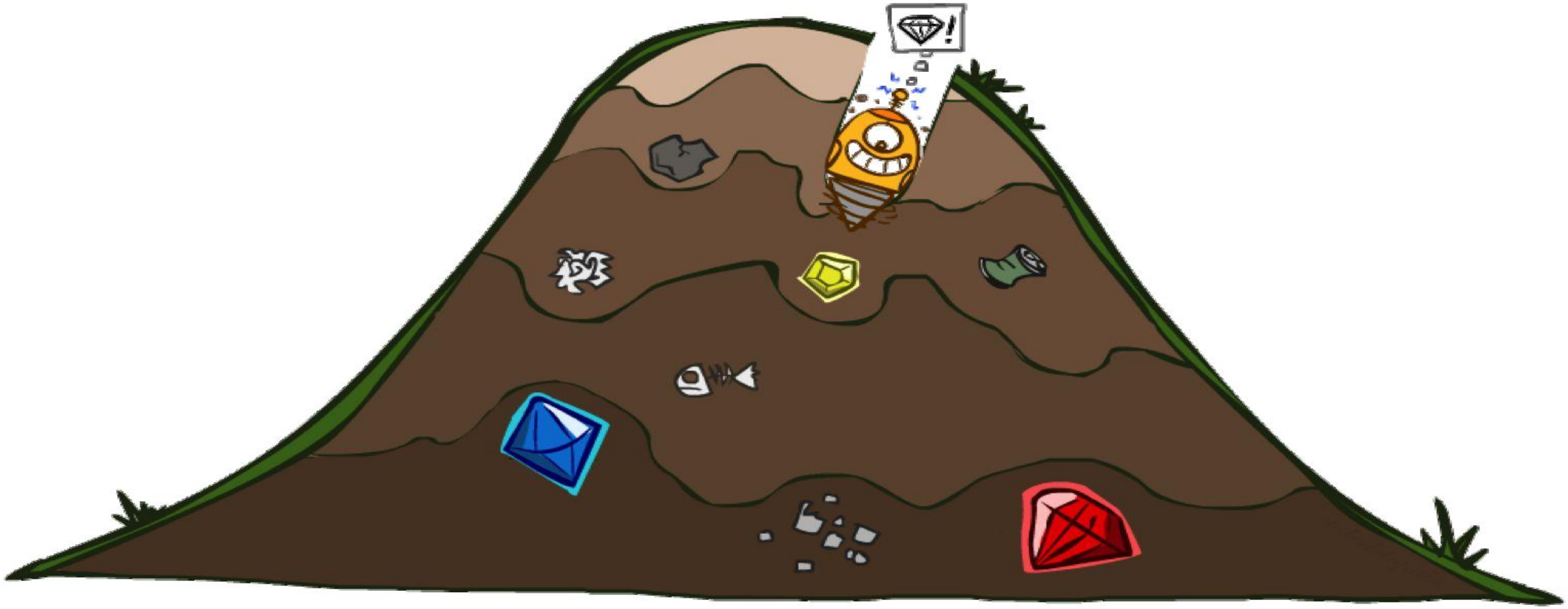


Straight−line distance
to Bucharest

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

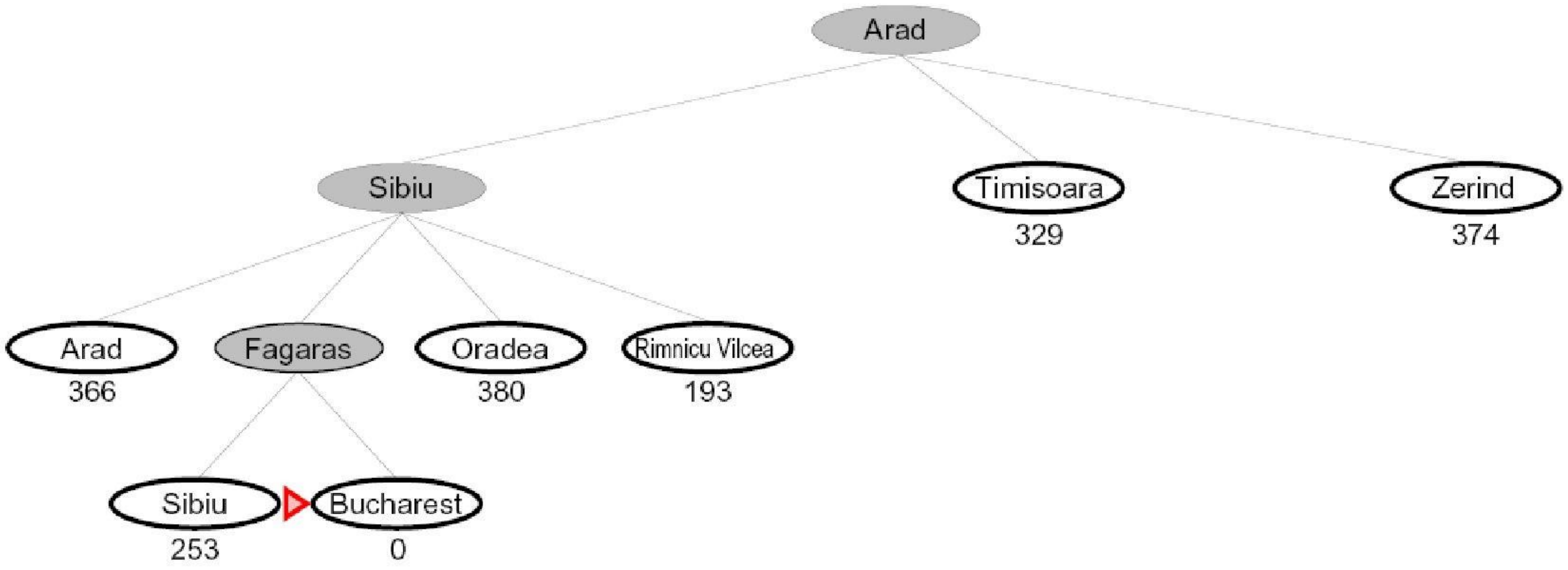
h(x)

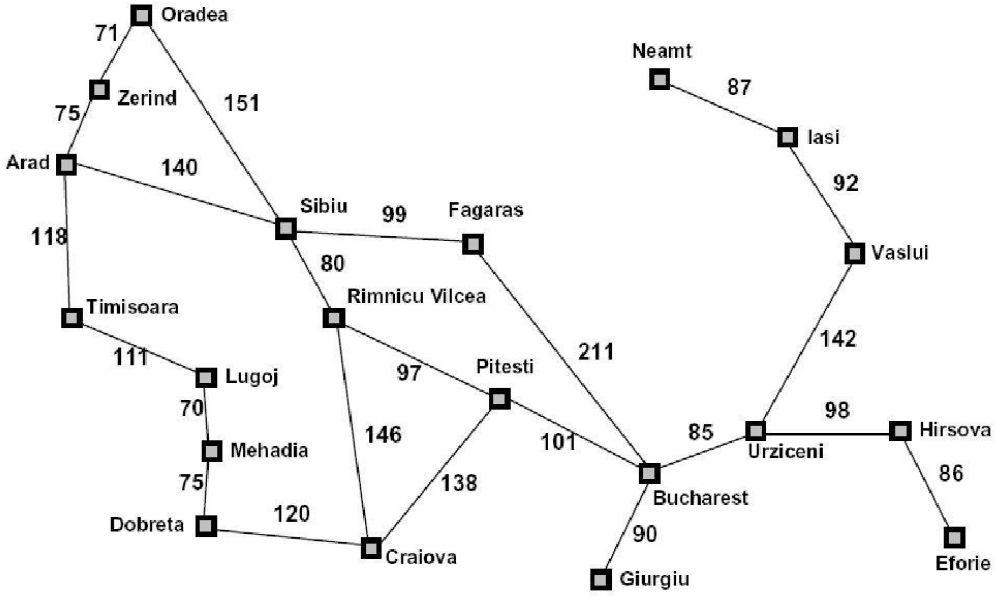
Greedy Search

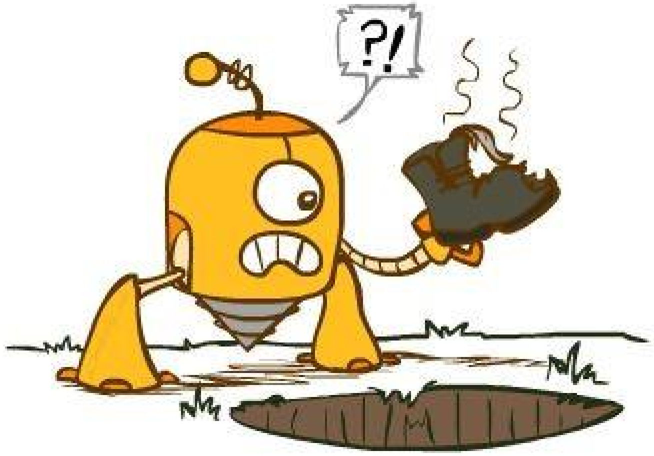


Greedy Search

- Expand the node that seems closest…





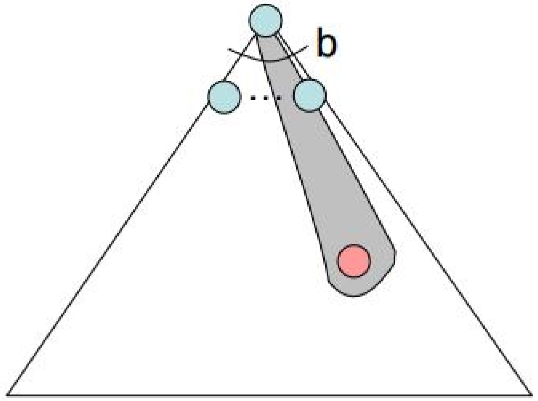


- What can go wrong?

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

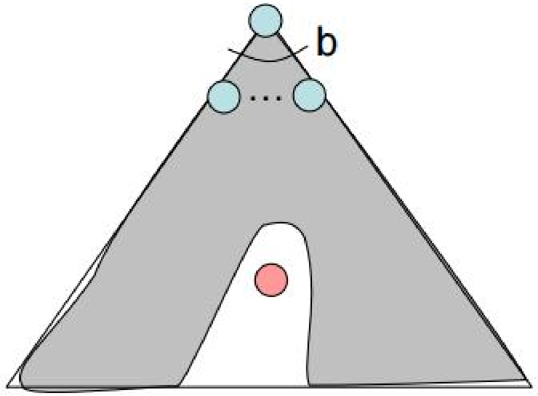


A common case: Best-first takes you straight to the (wrong) goal

Worst-case: like a badly-guided DFS

Time and Space Complexity: O(bm)

Not Optimal: Global Optimum?

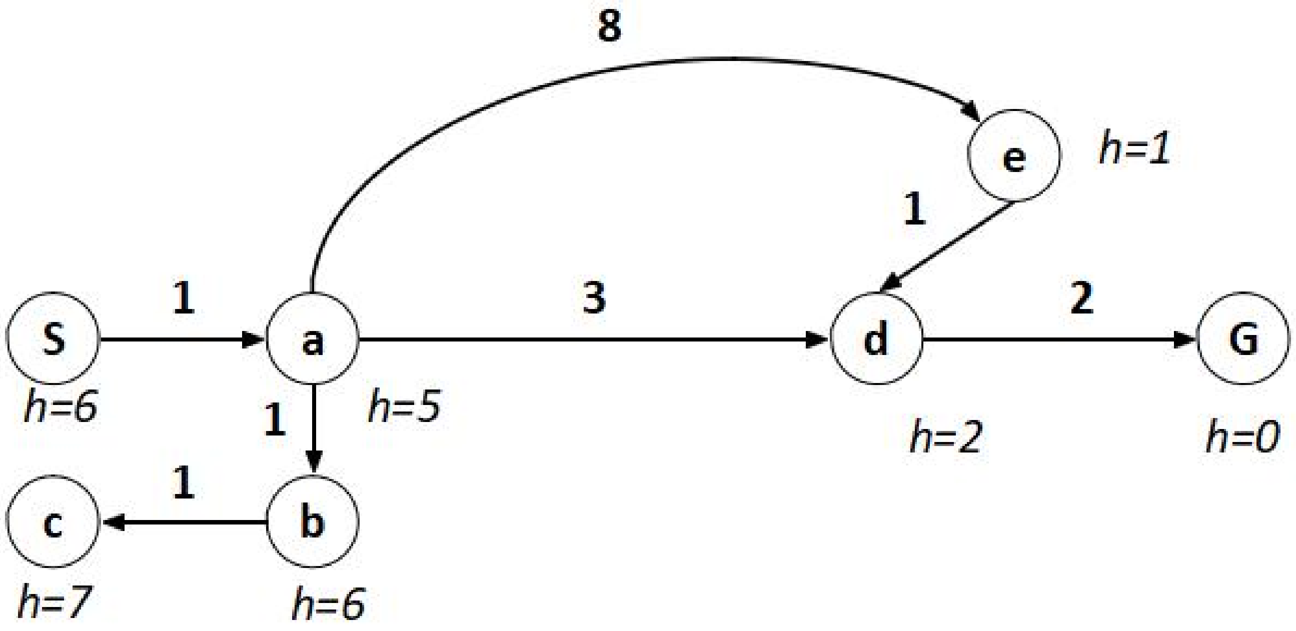


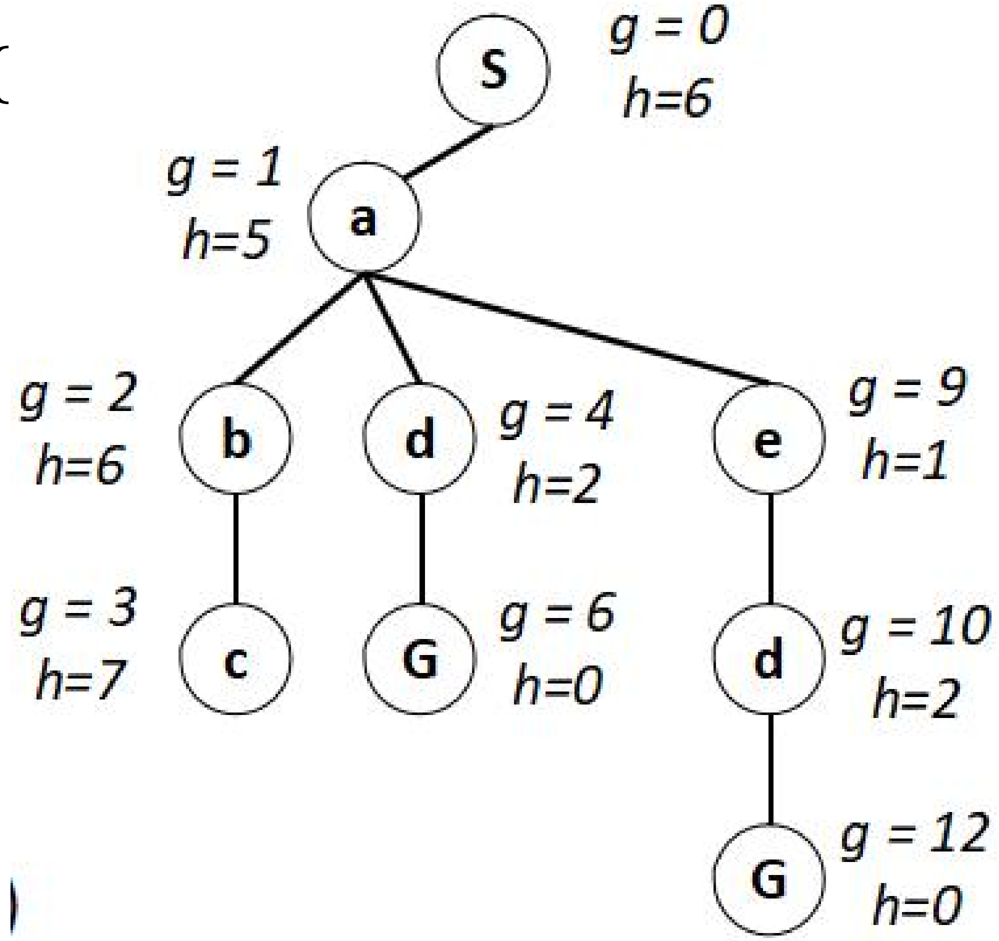
A* Search

A* Search

Combining UCS and Greedy

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

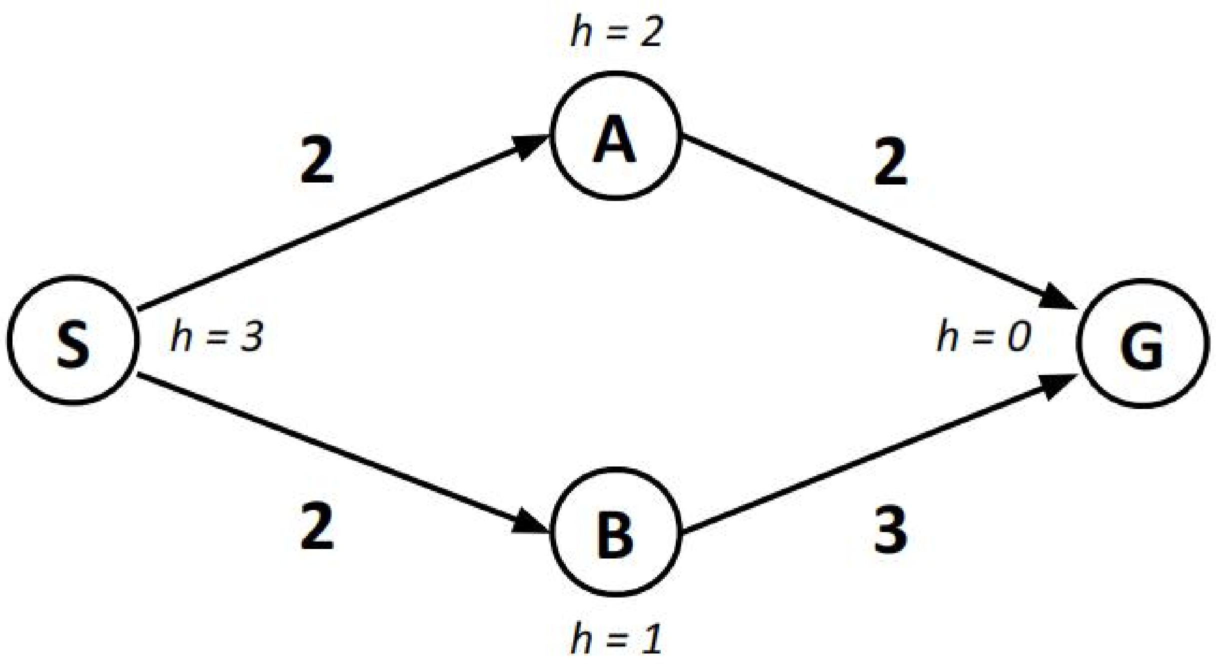




A* Search orders by the sum: **f(n) = g(n) + h(n)**

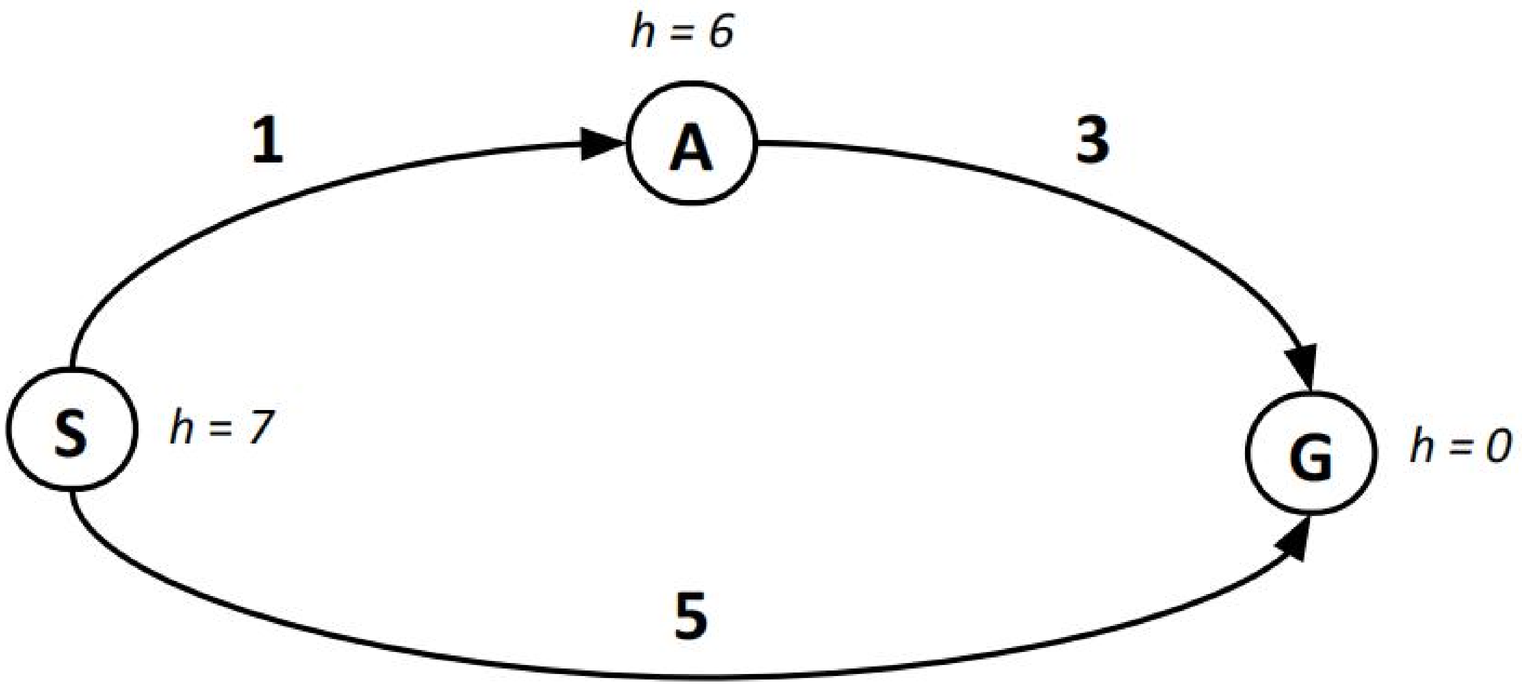
When should A* terminate?

Should we stop when we enqueue a goal?



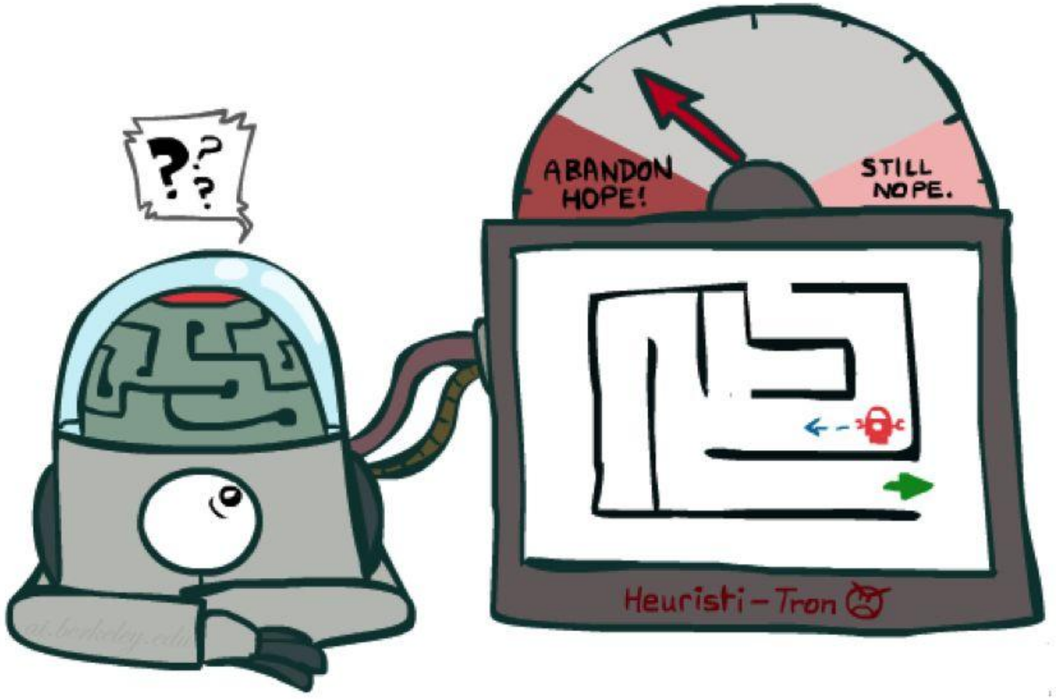
No: only stop when we dequeue a goal.

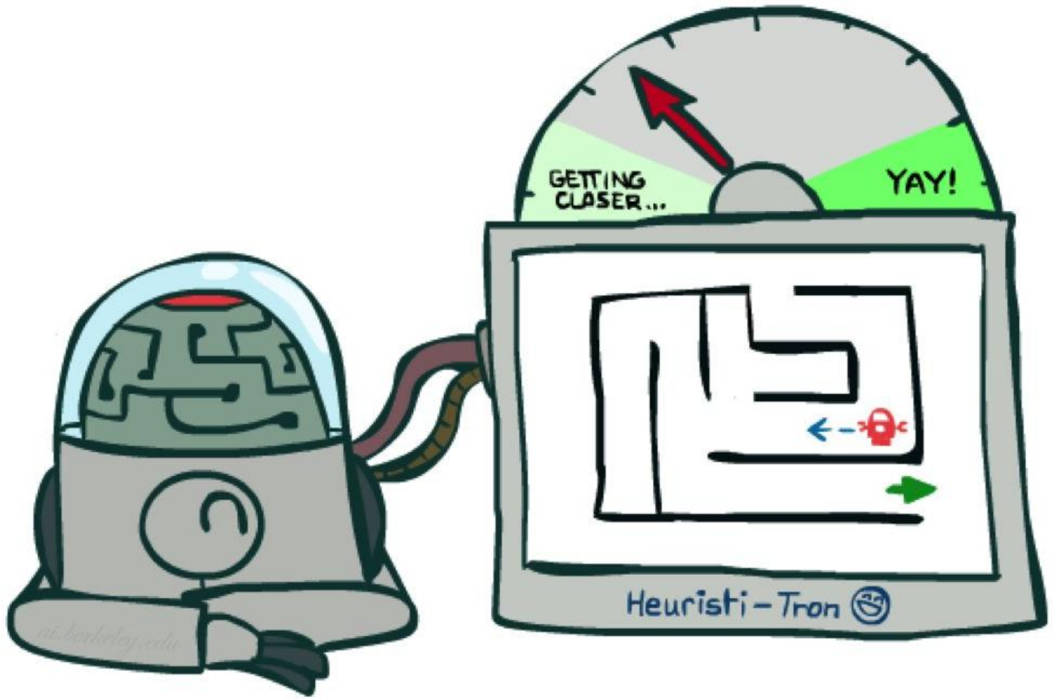
Is A* Optimal?



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

Admissible Heuristics





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

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.

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

Consistency? **H(n) <= c(n,a,n') + h(n')**

H(n)-H(n') <= C(n,a,n')

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.

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

Consistency? **H(n) <= c(n,a,n') + h(n')**

H(n)-H(n') <= C(n,a,n')

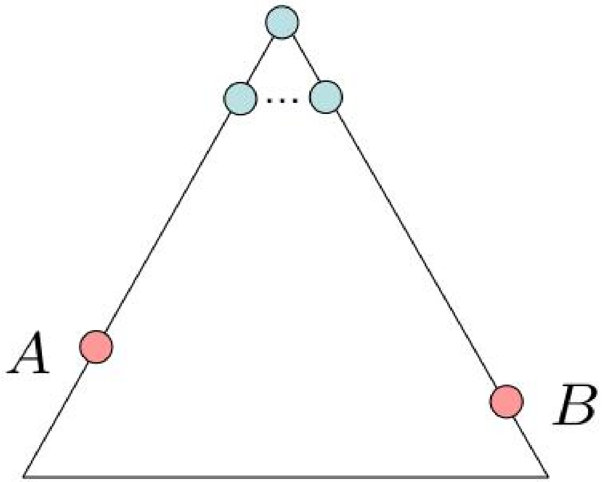
Optimality of A* Tree Search

Assume:

A is an optimal goal node

B is a suboptimal goal node

h is admissible



Claim:

A will exit the fringe before **B**

Optimality of A* Tree Search

- **A*** is cost-optimal, which we can be shown with a proof by contradiction.
- Suppose the optimal path has cost **C***, but the algorithm returns a path with cost **C >C***. Then there must be some node **n**which is on the optimal path and is unexpanded (because if all the nodes on the optimal path had been expanded, then we would have returned that optimal solution).

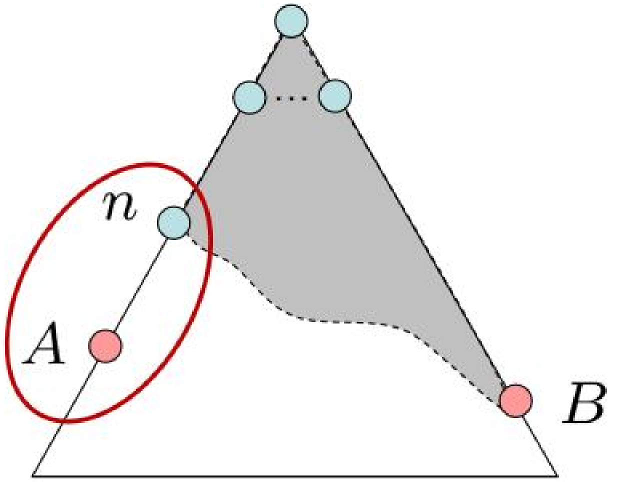
$$
\begin{aligned}
f(n) &> C^* \quad \text{(otherwise } n \text{ would have been expanded)} \\
f(n) &= g(n) + h(n) \quad \text{(by definition)} \\
f(n) &= g^*(n) + h(n) \quad \text{(because } n \text{ is on an optimal path)} \\
f(n) &\leq g^*(n) + h^*(n) \quad \text{(because of admissibility, } h(n) \leq h^*(n)\text{)} \\
f(n) &\leq C^* \quad \text{(by definition, } C^* = g^*(n) + h^*(n)\text{)}
\end{aligned}
$$

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).**



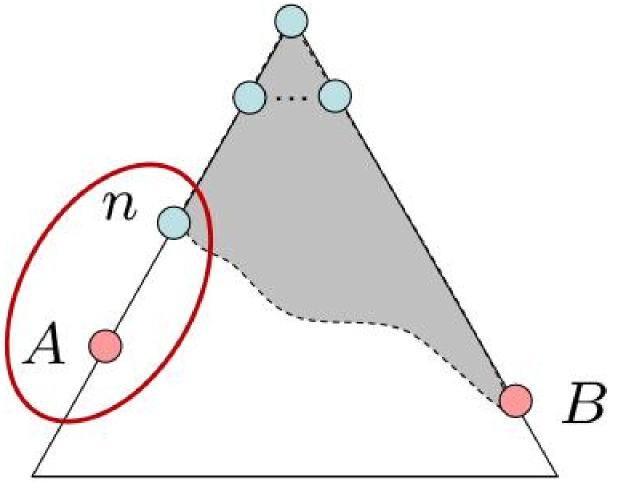
$f(n) = g(n) + h(n)$	Definition of f-cost
$f(n) \leq g(A)$	Admissibility of h
$g(A) = f(A)$	h = 0 at a goal

Optimality of A* Tree Search: Blocking

Proof:

- Imagine **B** is on the 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)**



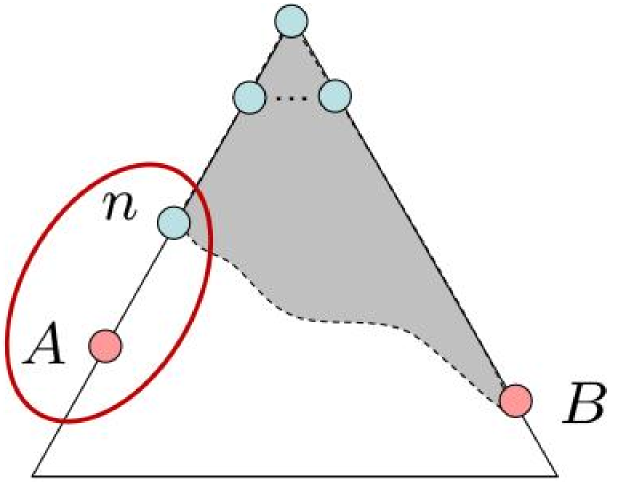
$g(A) < g(B)$ B is suboptimal

$f(A) < f(B)$ 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.**



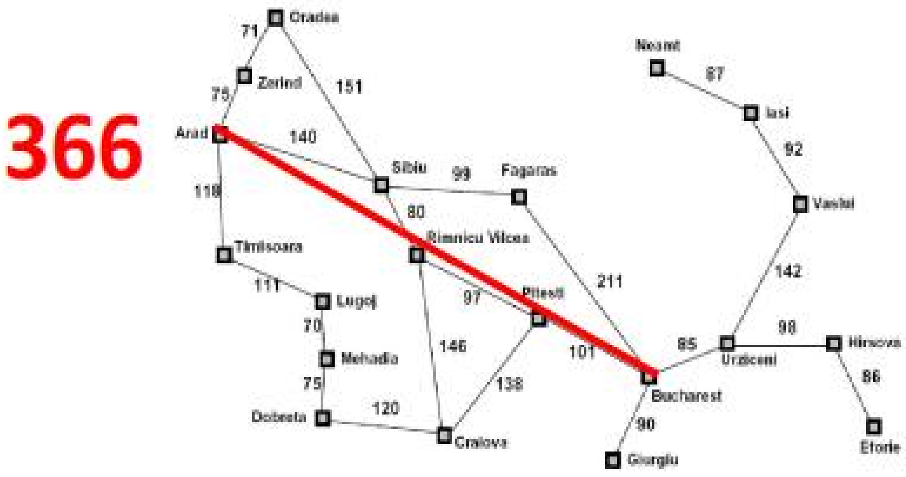
1. **f(n)** is less or equal to **f(A)**
2. **f(A)** is less than **f(B)**
3. ***n*** expands before **B**

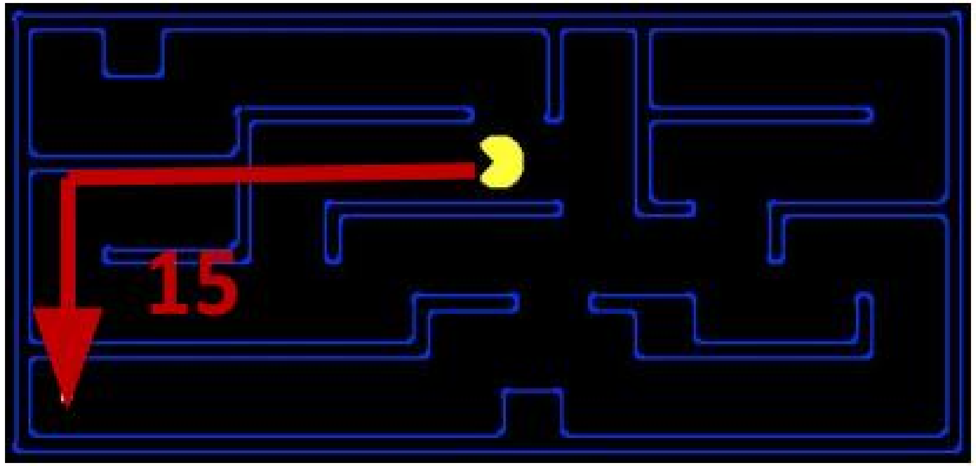
$$f(n) \leq f(A) < f(B)$$

- All ancestors of **A** expand before **B**
- **A** expands before **B**
- **A*** search is optimal

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.

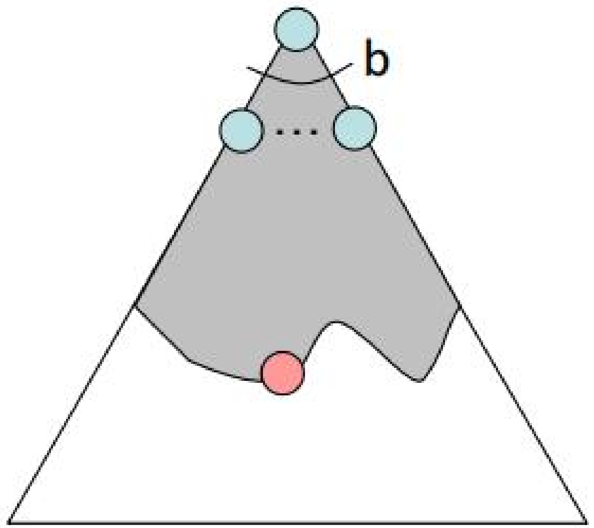




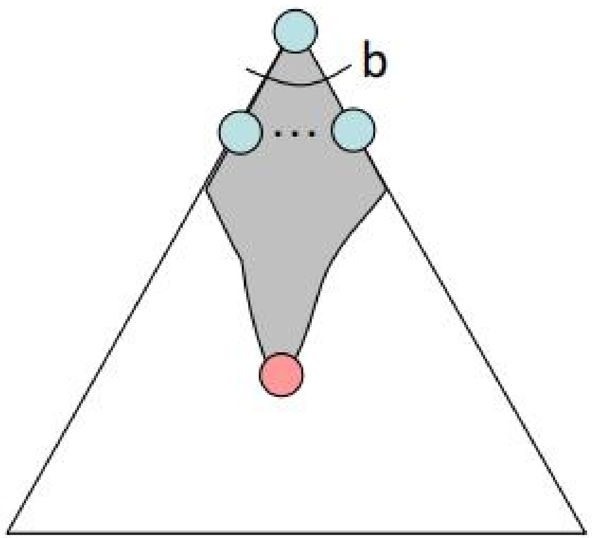
✓ **Inadmissible heuristics are often useful too**

Properties of A*

Uniform-Cost

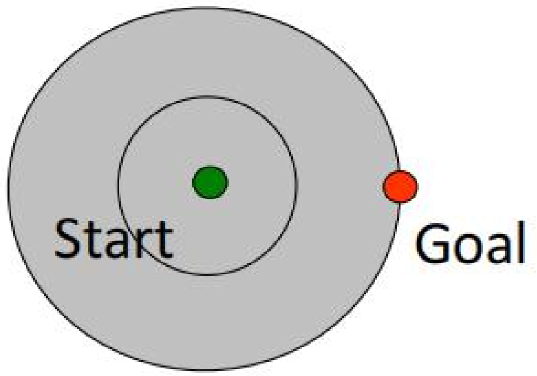


A*

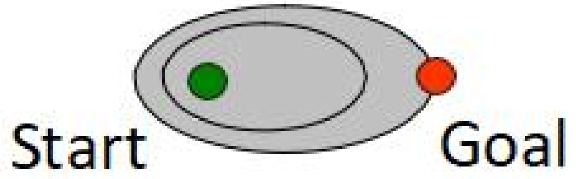


Properties of A*

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



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



A* Applications

- Video games
- Pathing / Routing problems
- Resource planning problems
- Robot motion planning
- Language analysis

Semi-Lattice of Heuristics

Dominance: $\mathbf{h_a \geq h_c}$ if

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

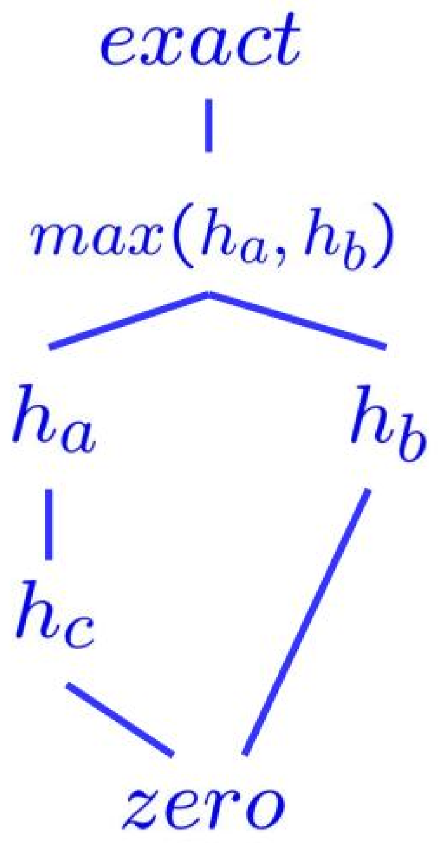
Heuristics form a semi-lattice:

- Max of admissible heuristics is admissible

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

Trivial heuristics

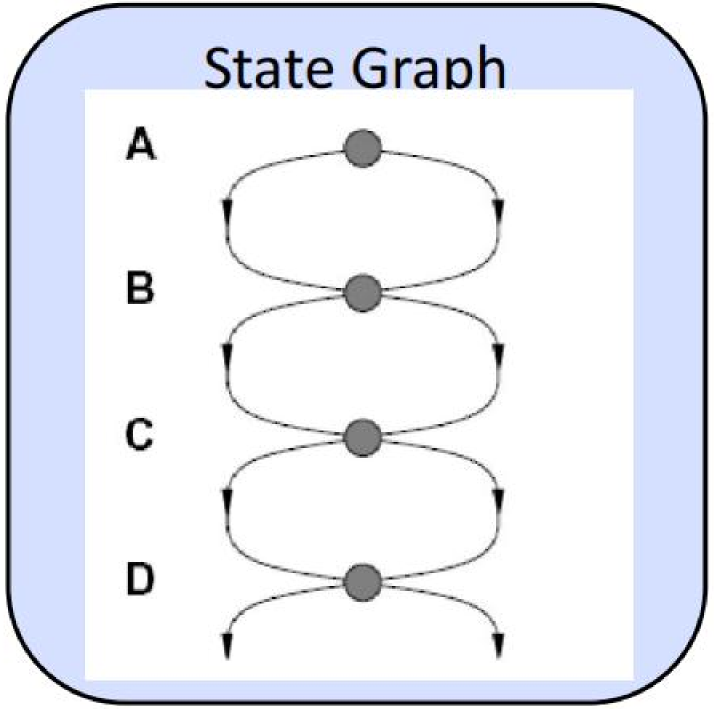
- Bottom of lattice is the zero heuristic (what does this give us?)
- Top of lattice is the exact heuristic

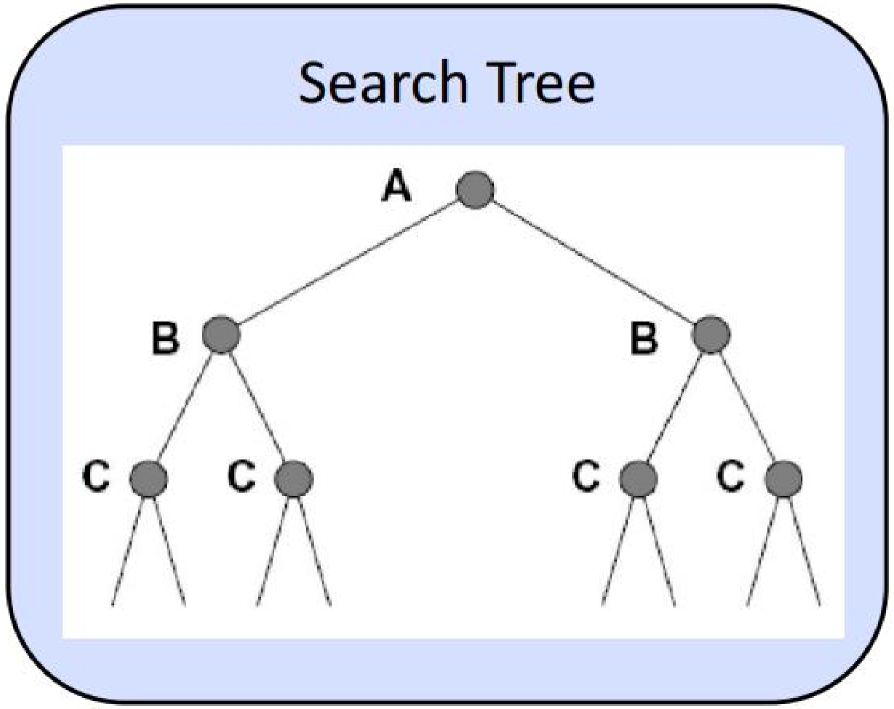


Graph Search

Tree Search: Extra Work!

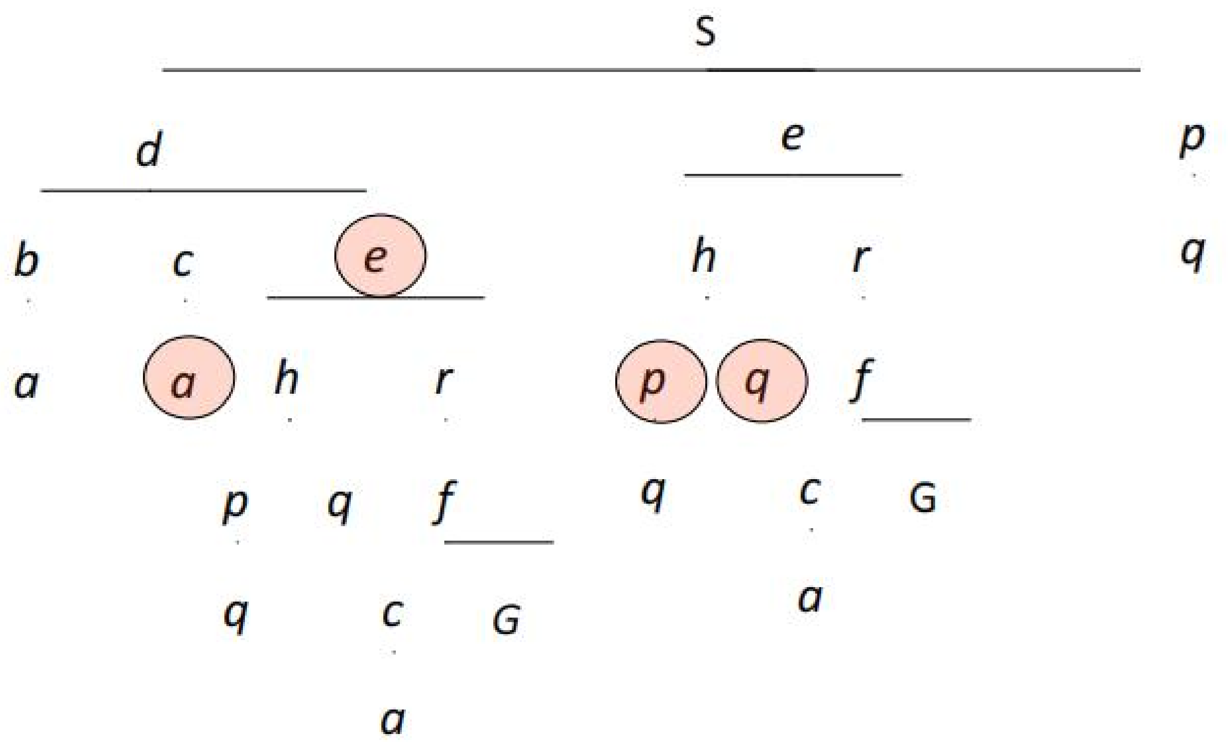
- Failure to detect repeated states can cause exponentially more work.





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; reducing overhead.

Can graph search wreck completeness? Why/why not?

How about optimality?

A* Graph Search Gone Wrong?

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; reducing overhead.

Can graph search wreck completeness? Why/why not?

How about optimality?