CS 343H: Artificial Intelligence

Lecture 4: Informed Search

1/23/2014

Slides courtesy of Dan Klein at UC-Berkeley
Unless otherwise noted

Today

- Informed search
  - Heuristics
  - Greedy search
  - A* search

- Graph search

Recap: Search

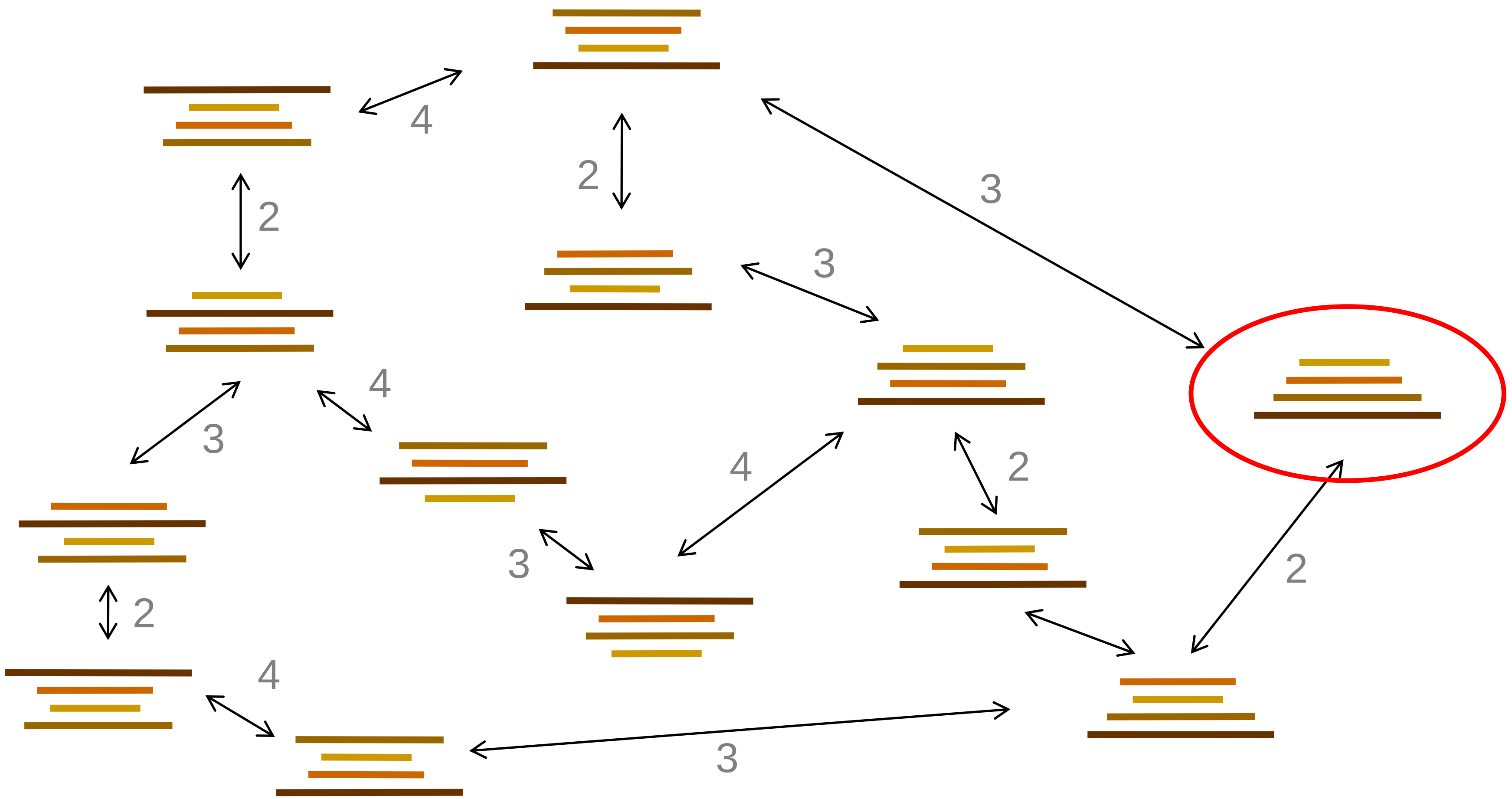
- Search problem:
  - States (configurations of the world)
  - Actions and costs
  - Successor function: a function from states to lists of (state, action, cost) triples (world dynamics)
  - Start state and goal test
- Search tree:
  - Nodes: represent plans for reaching states
  - Plans have costs (sum of action costs)
- Search algorithm:
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)
  - Optimal: finds least-cost plans

Example: Pancake Problem

Cost: Number of pancakes flipped

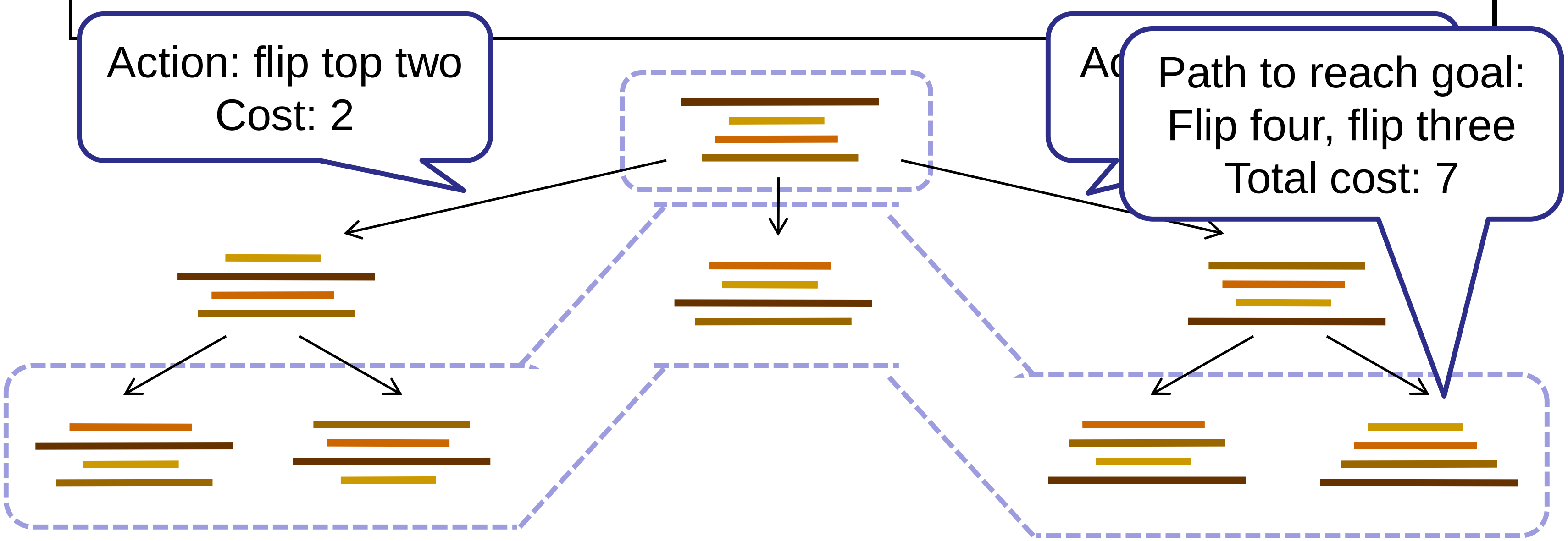
Example: Pancake Problem

State space graph with costs as weights



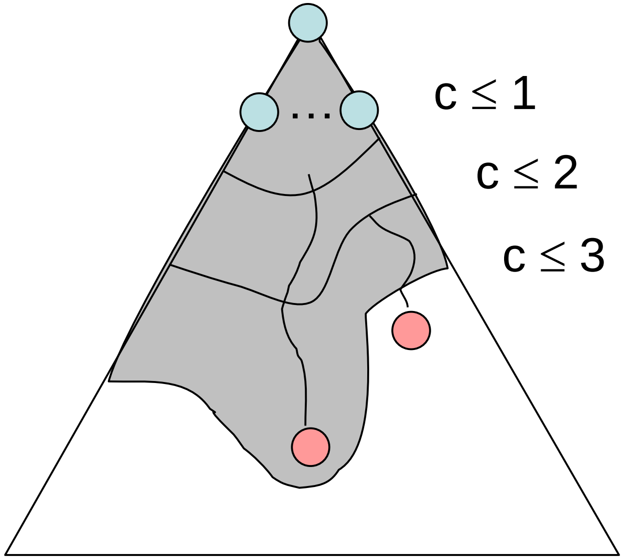
General Tree Search

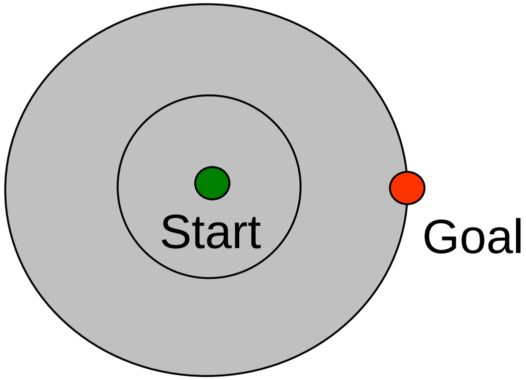
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the 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: Uniform Cost Search

- Strategy: expand lowest path cost
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every "direction"
  - No information about goal location

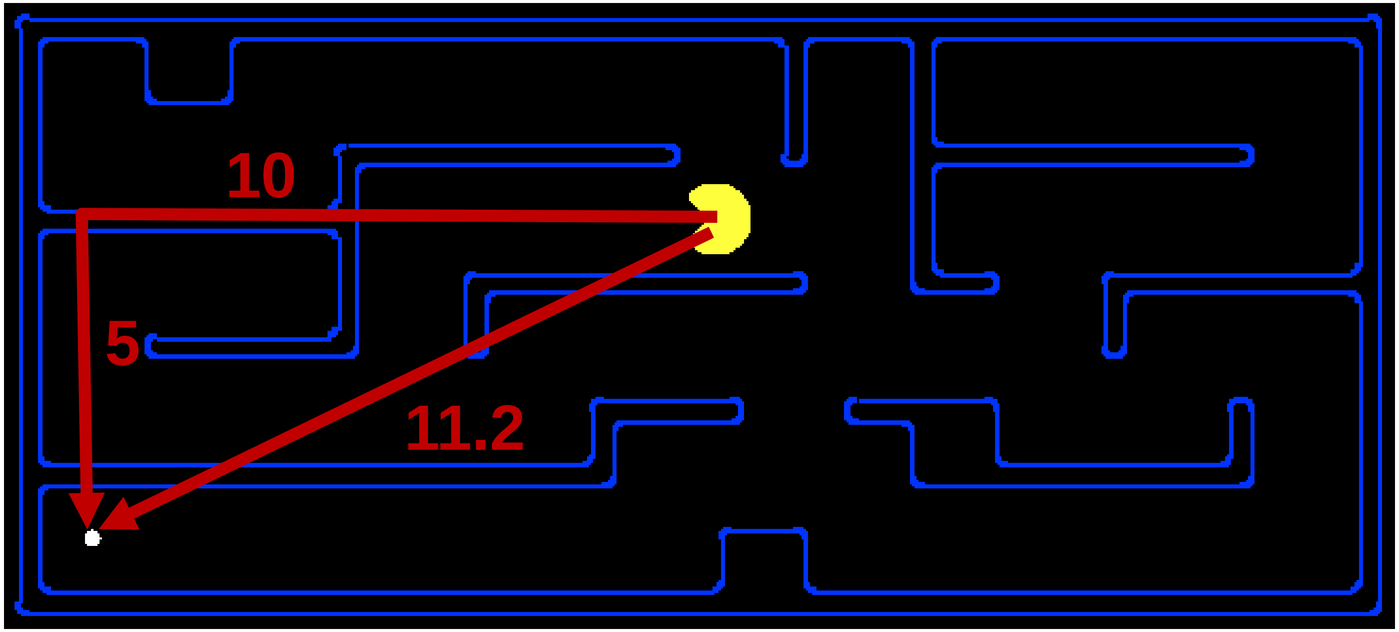




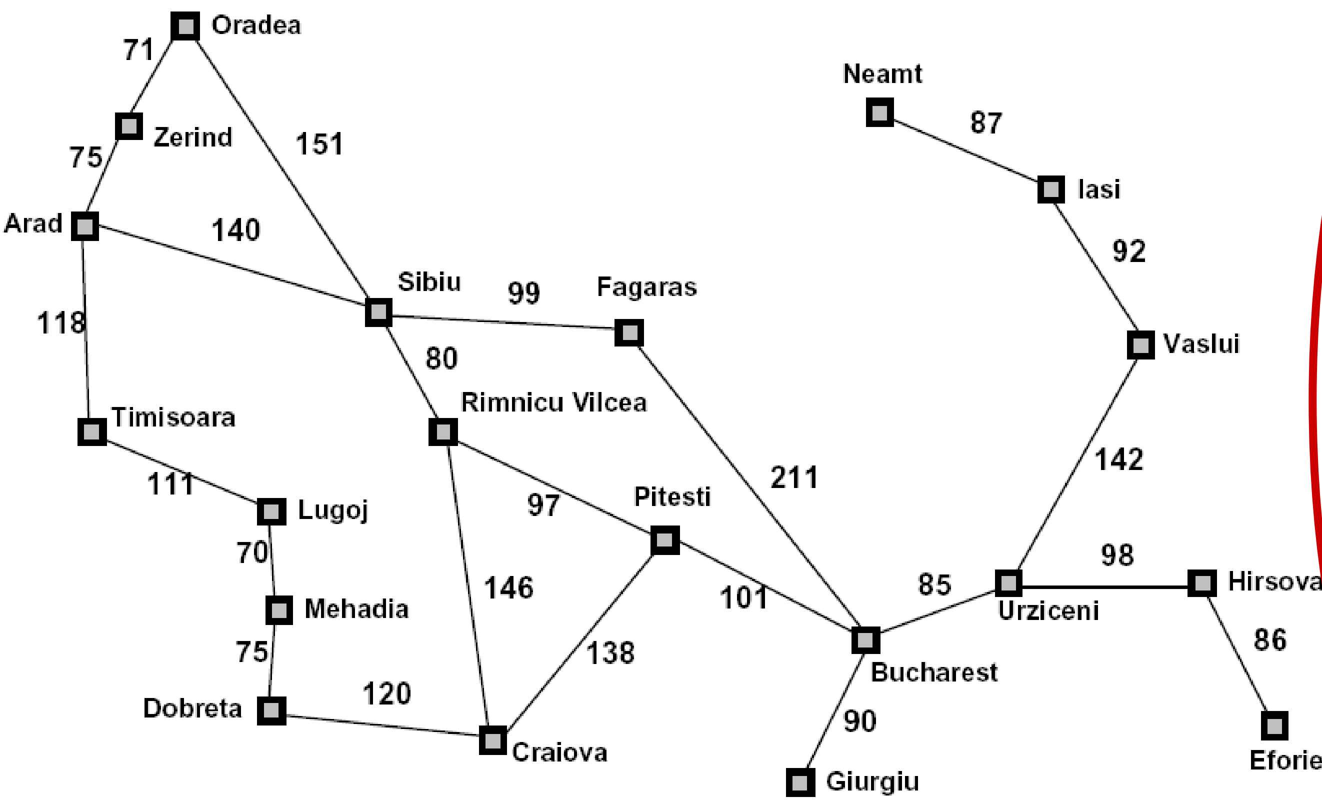
[demo: countours UCS]

Search heuristic

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



Example: Heuristic Function

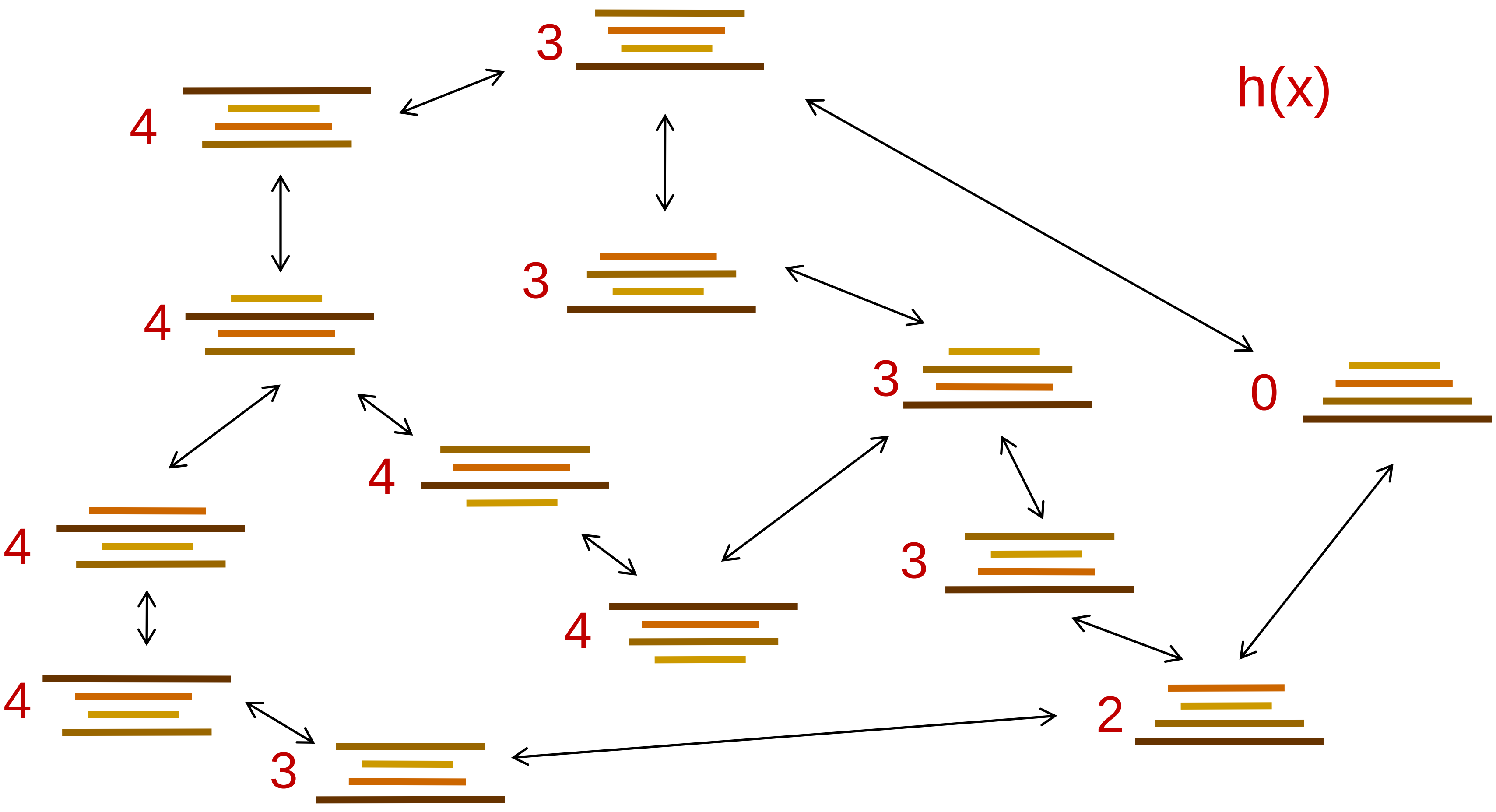


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

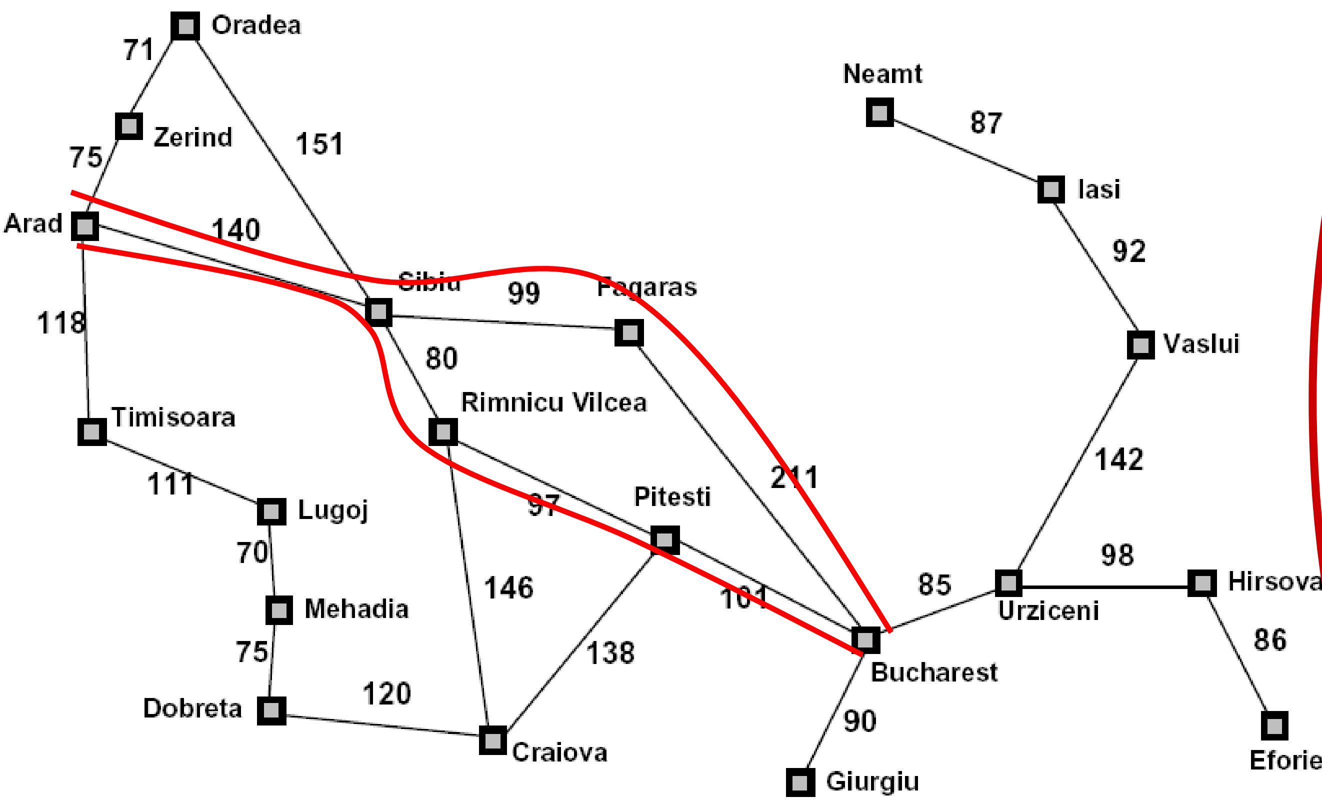
Example: Heuristic Function

Heuristic: the largest pancake that is still out of place



- How to use the heuristic?
- What about following the “arrow” of the heuristic?.... Greedy search

Example: Heuristic Function



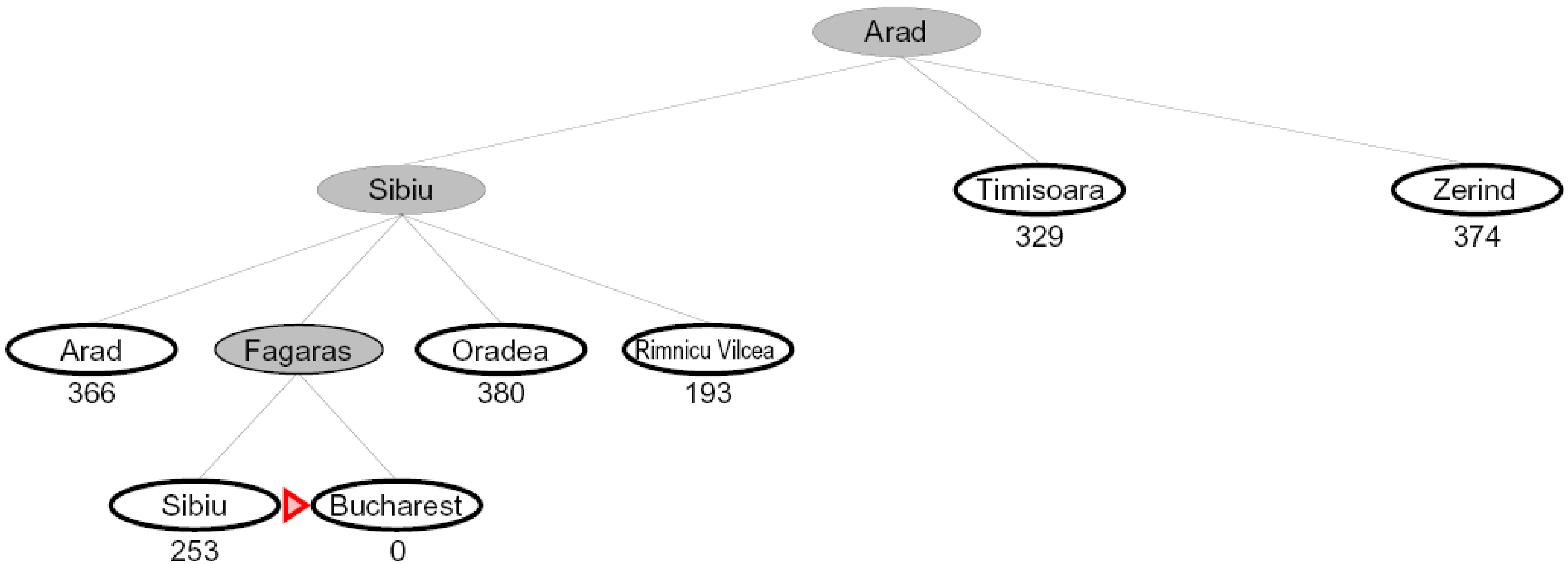
Straight−line distance to Bucharest

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)

Best First / Greedy Search

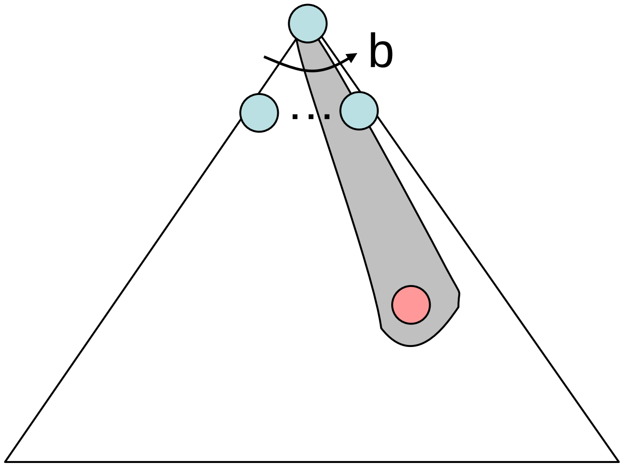
- Expand the node that seems closest…

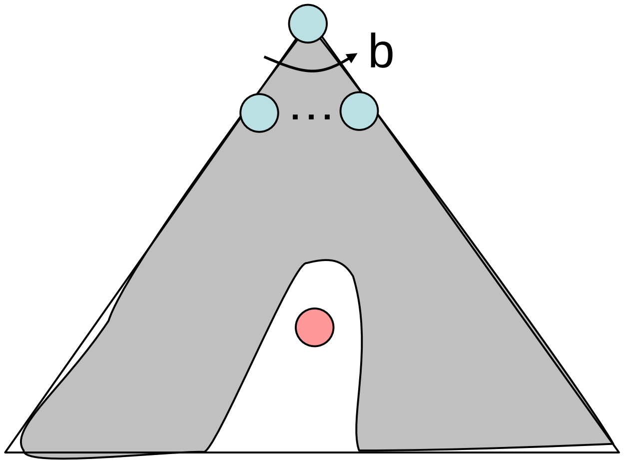


- What can go wrong?

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



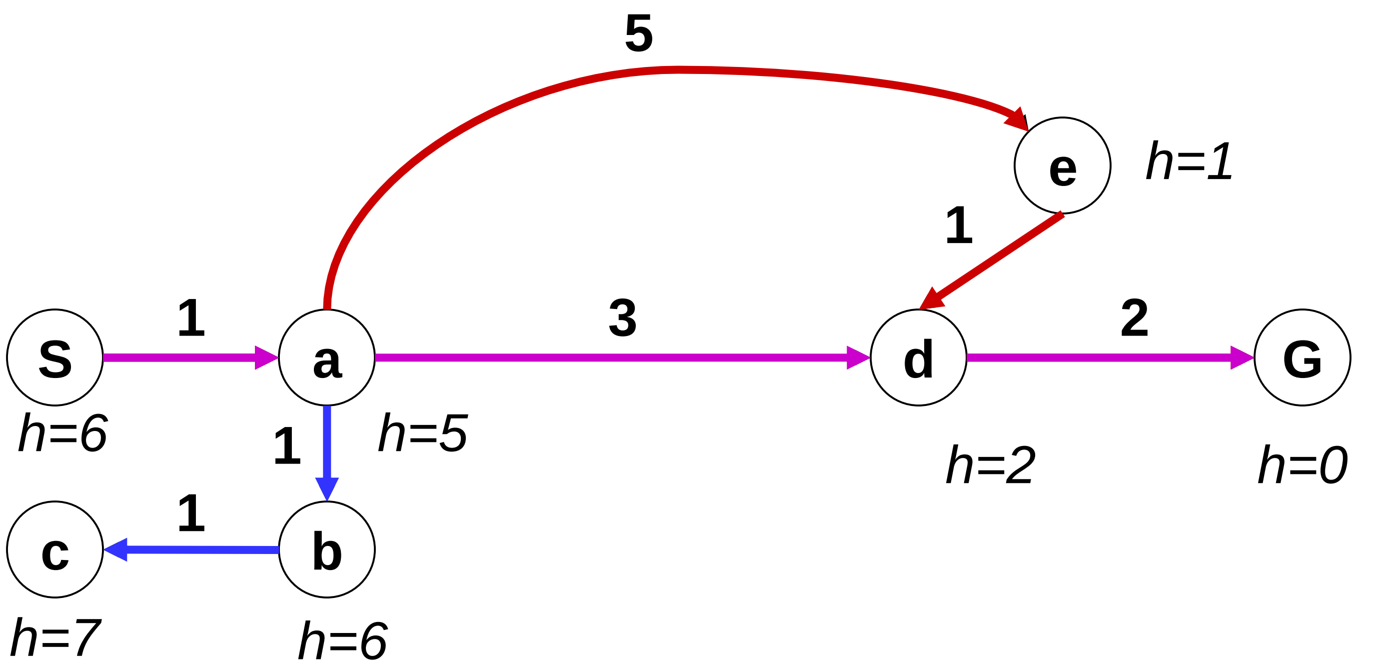


[demo: countours greedy]

Enter: A* search

Combining UCS and Greedy

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

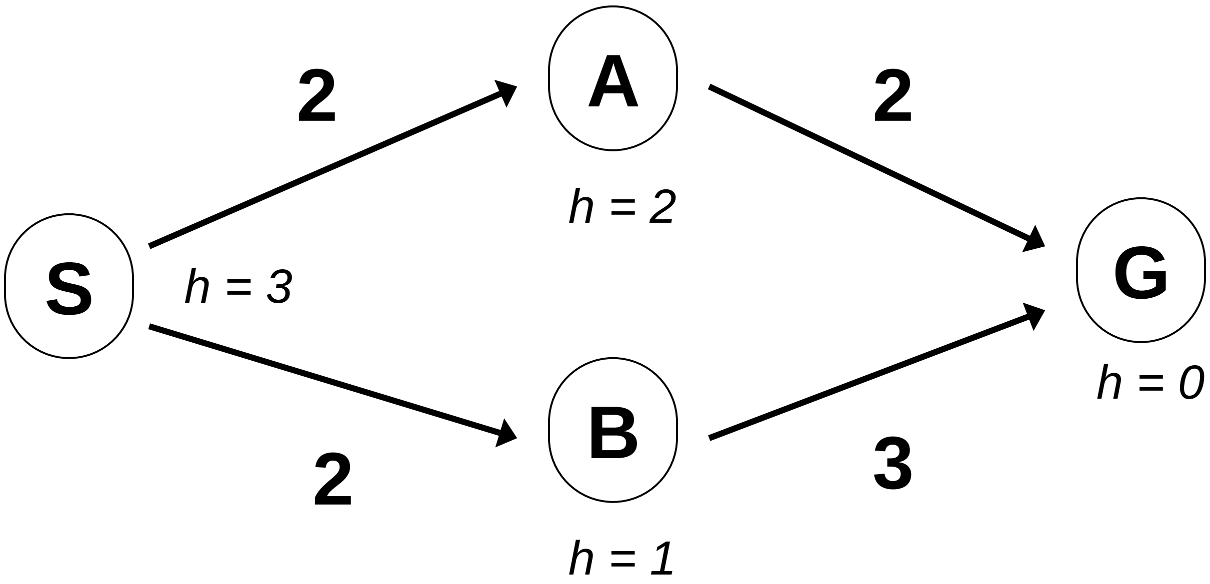


- A* Search orders by the sum: f(n) = g(n) + h(n)

Example: Teg Grenager

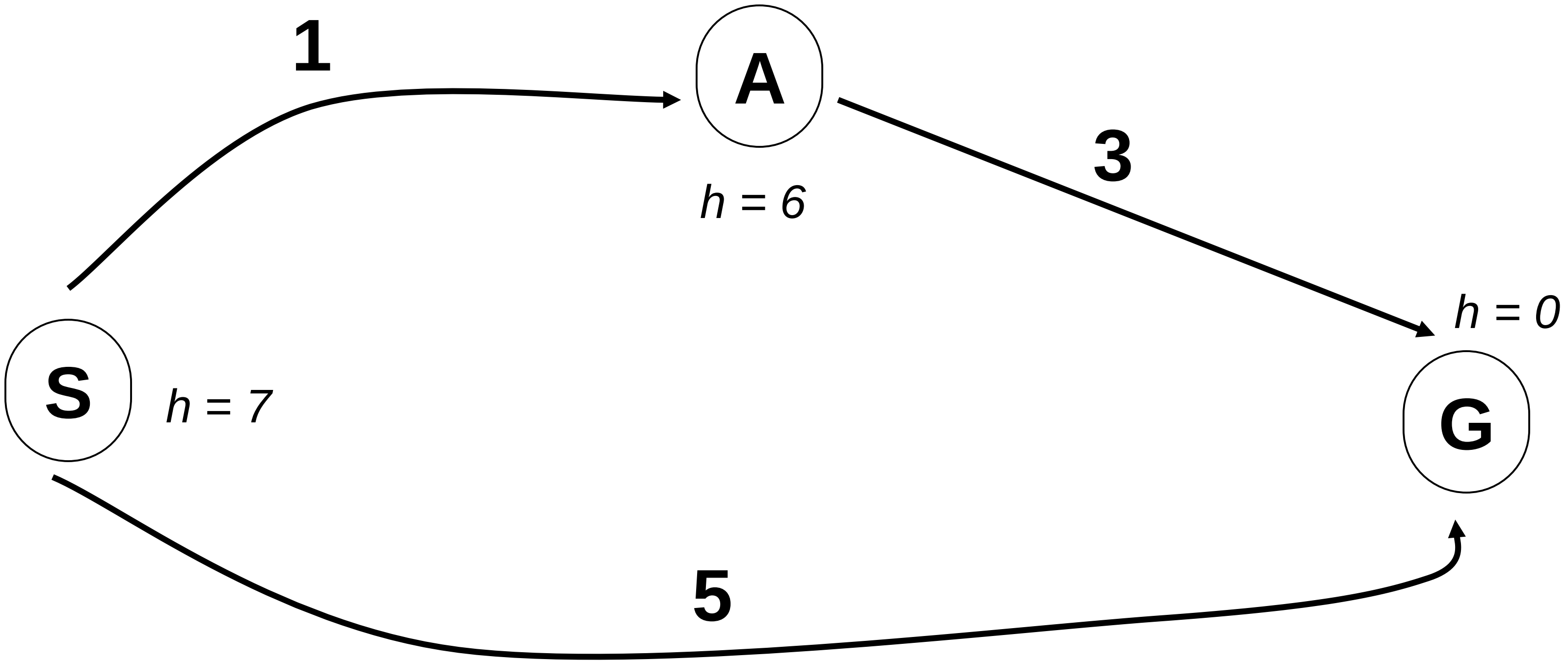
When should A* terminate?

- 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 <u>less</u> than actual costs!

Idea: admissibility

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

Admissible (optimistic):
slows down bad plans but
never outweigh true costs

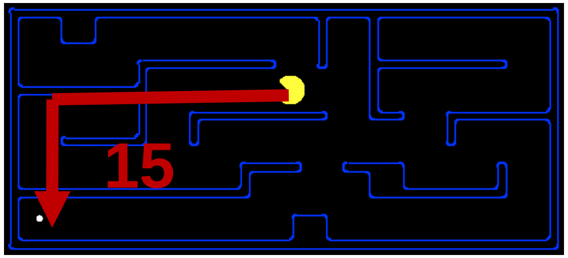
Admissible Heuristics

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

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

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.

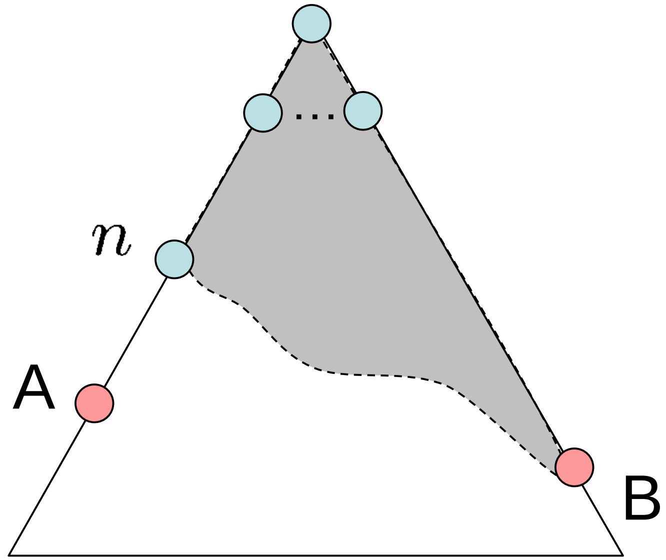
Optimality of A*

Notation:

- g(n) = cost to node n
- h(n) = estimated cost from n

  to the nearest goal (heuristic)
- f(n) = g(n) + h(n) =

  estimated total cost via n
- A: a lowest cost goal node
- B: another goal node



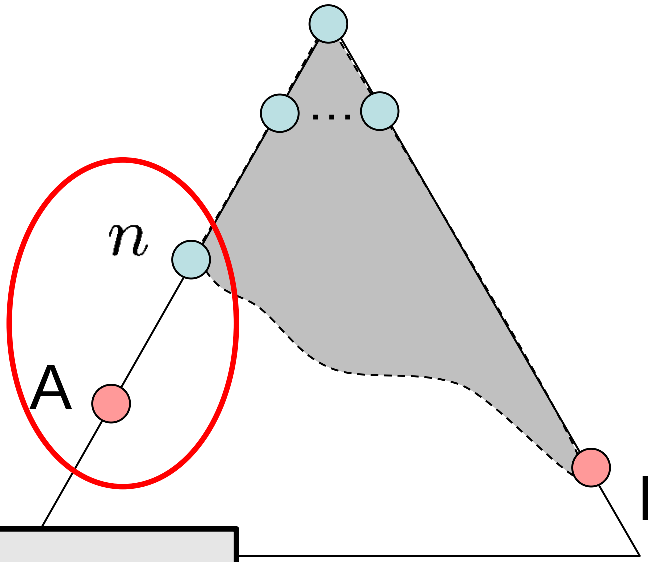
Claim: A will exit the fringe before B.

Claim: A will exit the fringe before B.

Optimality of A*

- Imagine B is on the fringe.
- Some ancestor n of A must be on the fringe too (maybe n is A)
- Claim: n will be expanded before B.

1. f(n) <= f(A)



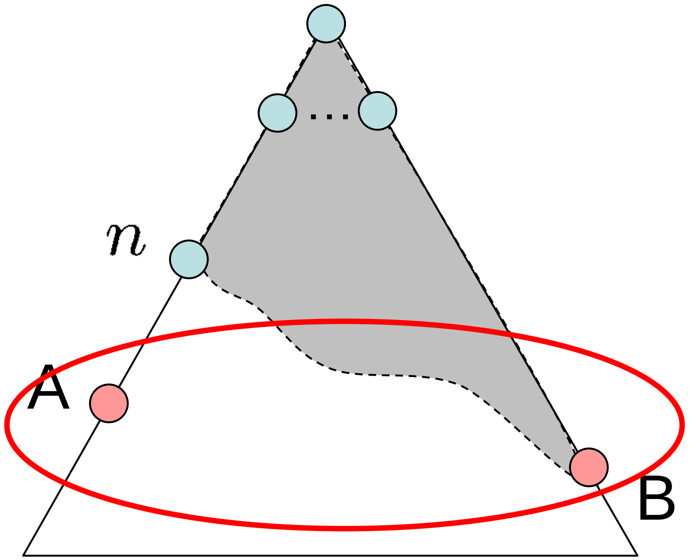
- f(n) = g(n) + h(n) // by definition
- f(n) <= g(A) // by admissibility of h
- g(A) = f(A) // because h=0 at goal

Claim: A will exit the fringe before B.

Optimality of A*

- Imagine B is on the fringe.
- Some ancestor n of A must be on the fringe too (maybe n is A)
- Claim: n will be expanded before B.

1. f(n) <= f(A)
2. f(A) < f(B)



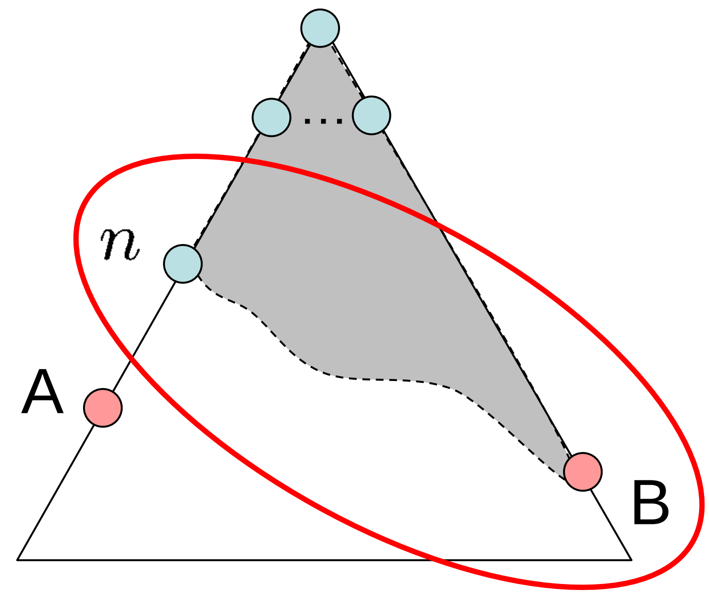
- g(A) < g(B) // B is suboptimal
- f(A) < f(B) // h=0 at goals

Claim: A will exit the fringe before B.

Optimality of A*

- Imagine B is on the fringe.
- Some ancestor n of A must be on the fringe too (maybe n is A)
- Claim: n will be expanded before B.

1. f(n) <= f(A)
2. f(A) < f(B)
3. n will expand before B



- f(n) <= f(A) < f(B) // from above
- f(n) < f(B)

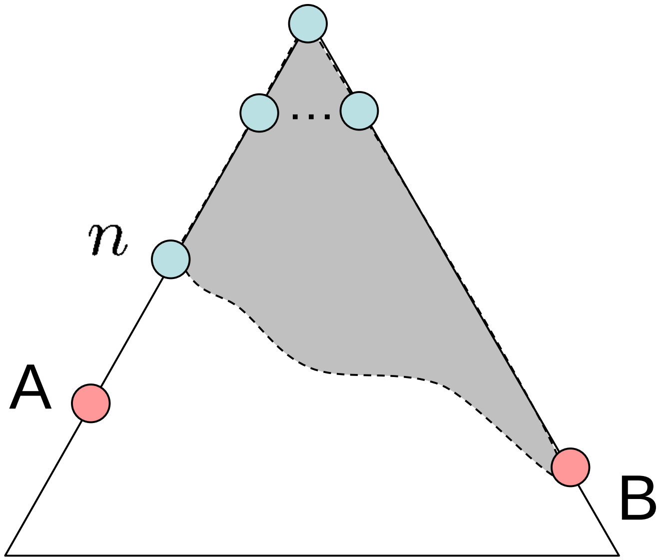
Claim: A will exit the fringe before B.

Optimality of A*

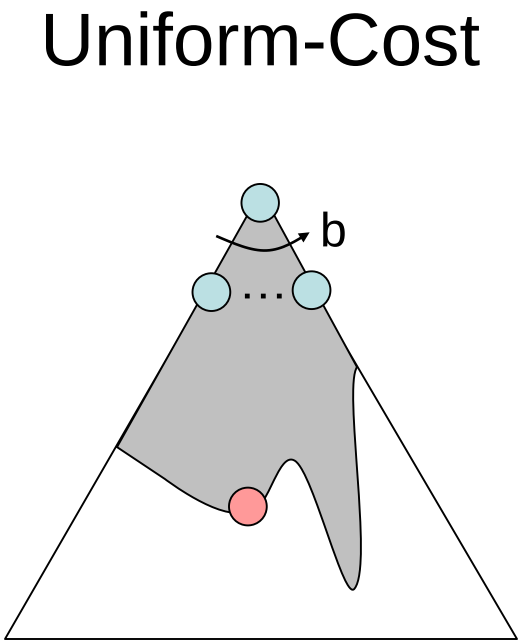
- Imagine B is on the fringe.
- Some ancestor n of A must be on the fringe too (maybe n is A)
- Claim: n will be expanded before B.

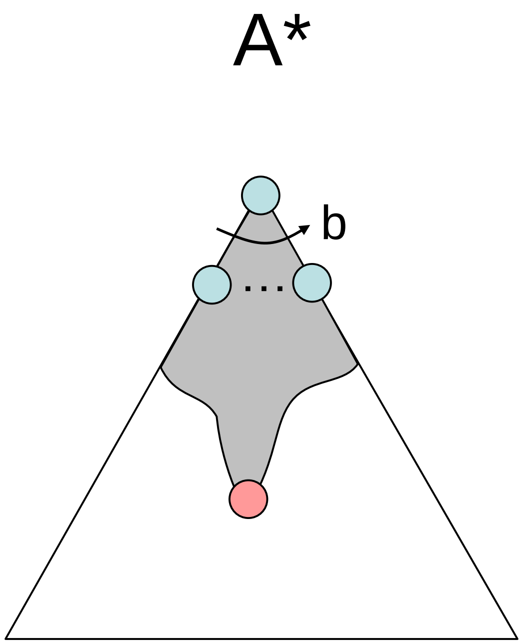
1. f(n) <= f(A)
2. f(A) < f(B)
3. n will expand before B

- All ancestors of A expand before B
- A expands before B



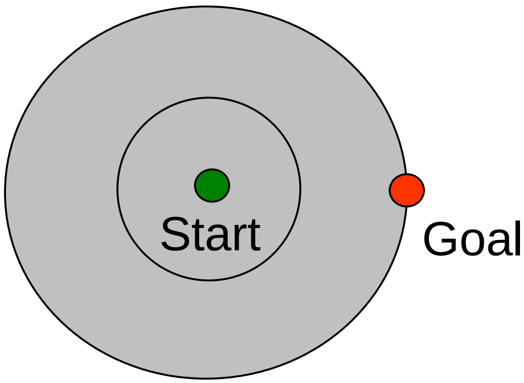
Properties of A*



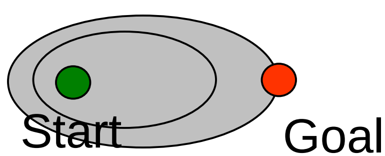


UCS vs A* Contours

- Uniform-cost expands equally in all directions



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



[demo: countours UCS / A*]

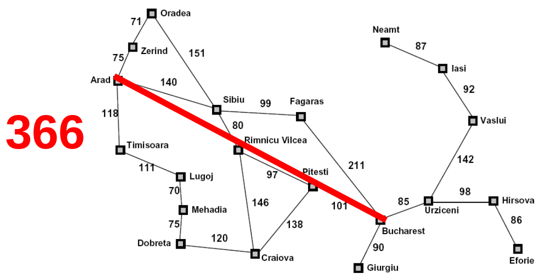
A* applications

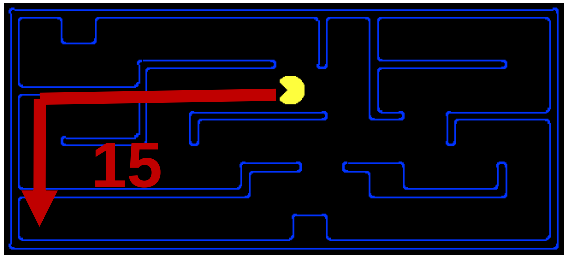
- Pathing / routing problems
- Video games
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- …

Creating Admissible Heuristics

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

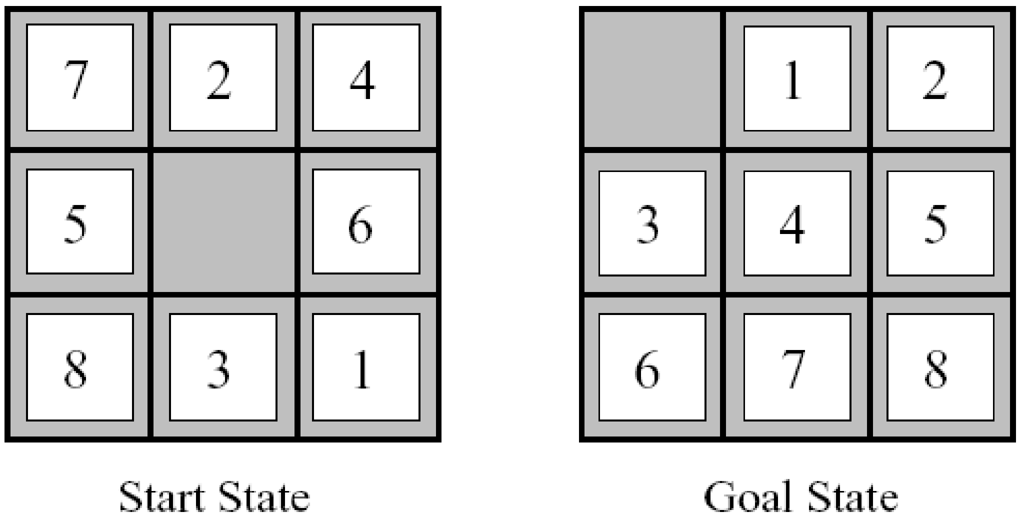
- Often, admissible heuristics are solutions to *relaxed problems*, where new actions are available





- Inadmissible heuristics are often useful too (why?)

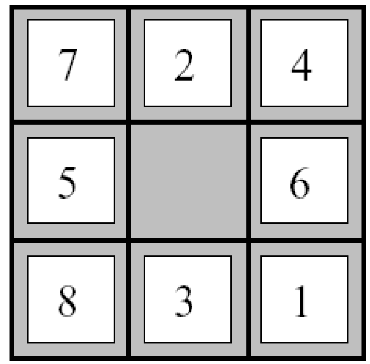
Example: 8 Puzzle



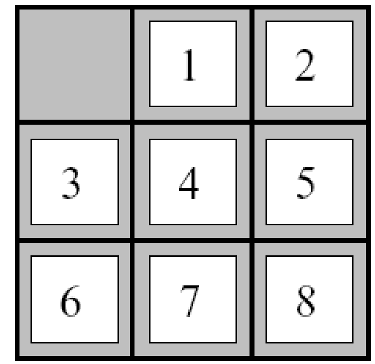
- What are the states?
- How many states?
- What are the actions?
- What states can I reach from the start state?
- What should the costs be?

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 optimal path has length…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

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 admissible?

- h(start) =

  3 + 1 + 2 + …

  = 18

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

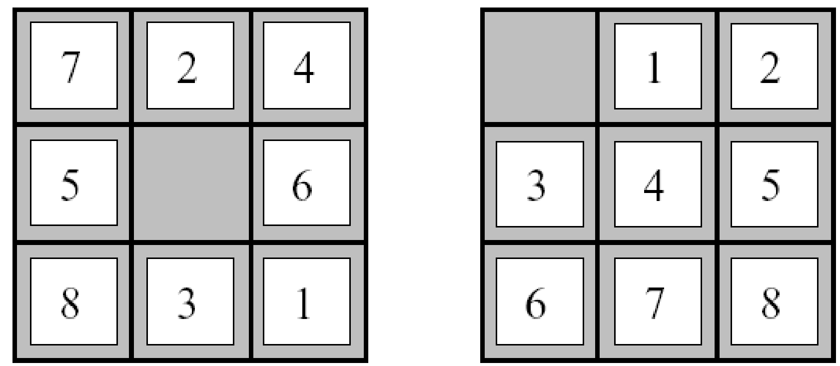
	Average nodes expanded when optimal path has length…		
	…4 steps	…8 steps	…12 steps
TILES	13	39	227
MANHATTAN	12	25	73

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 admissible?
- h(start) =

  3 + 1 + 2 + …

  = 18



Start State Goal State

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

8 Puzzle III

- How about using the *actual cost* as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What's wrong with it?

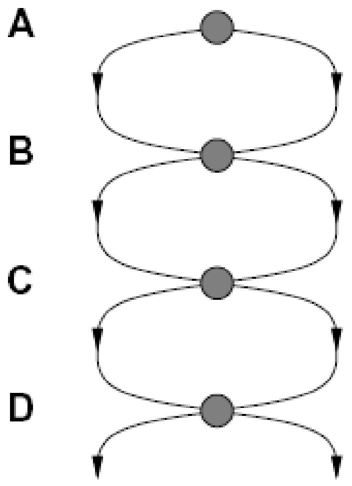
- With A*: a trade-off between quality of estimate and work per node!

Today

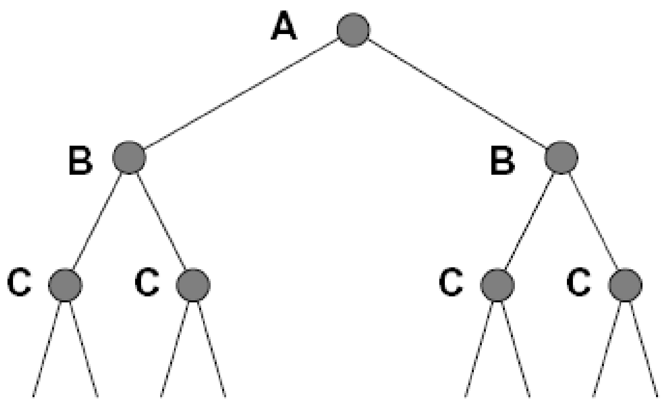
- Informed search
  - Heuristics
  - Greedy search
  - A* search
- Graph search

Tree Search: Extra Work!

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



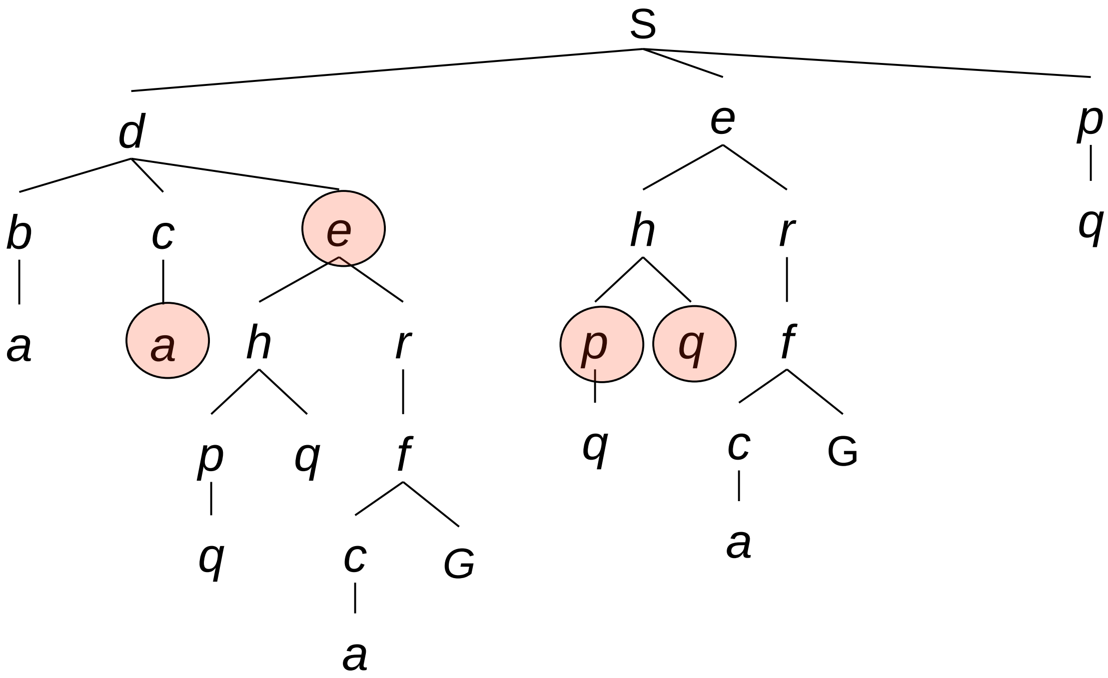
State graph



Search tree

Graph Search

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



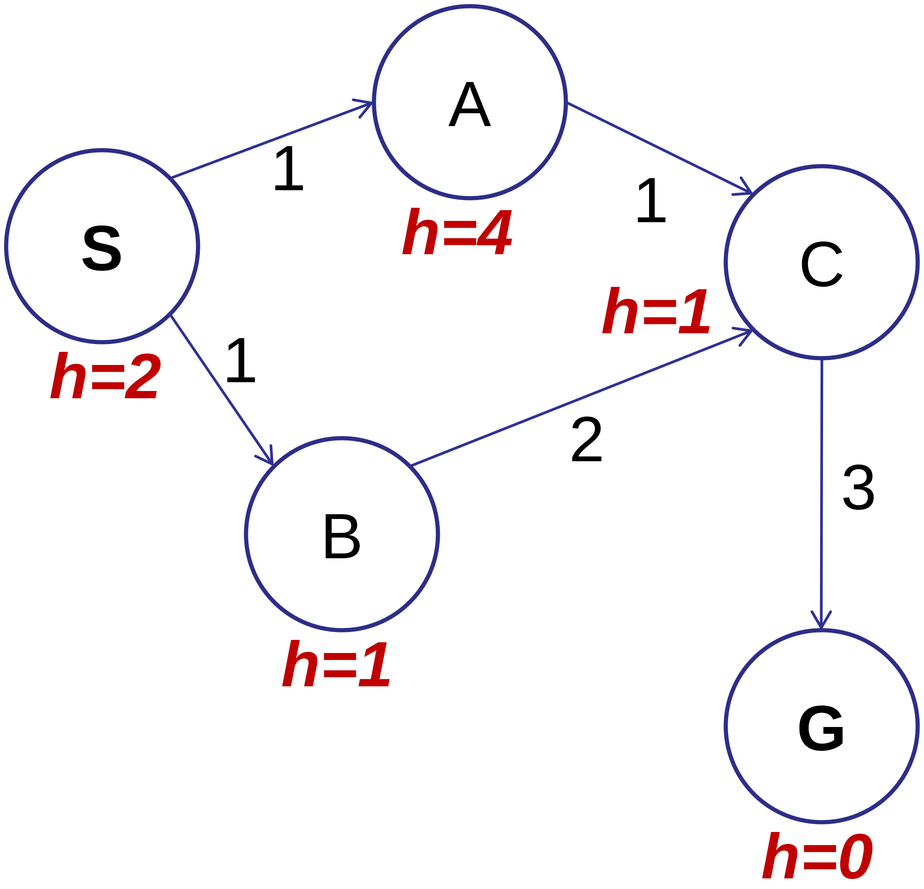
Graph Search

- 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 is new
  - If not new, skip it
- Important: store the closed set as a set, not a list
- Can graph search wreck completeness? Why/why not?
- How about optimality?

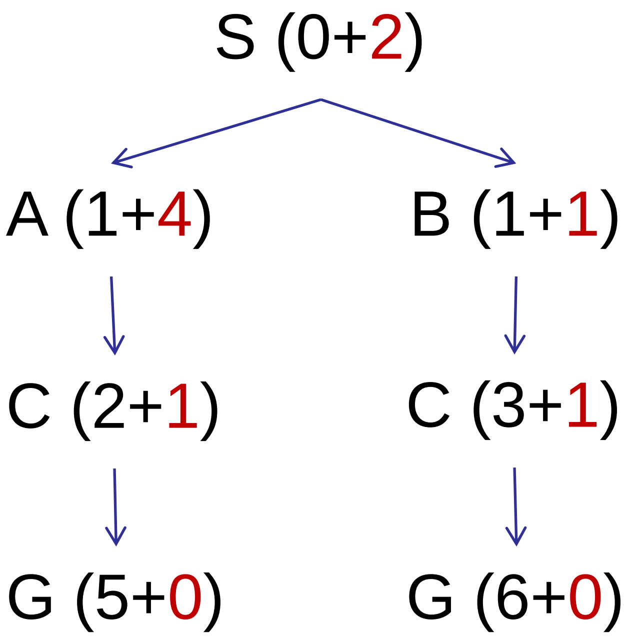
Warning: 3e book has a more complex, but also correct, variant

A* Graph Search Gone Wrong?

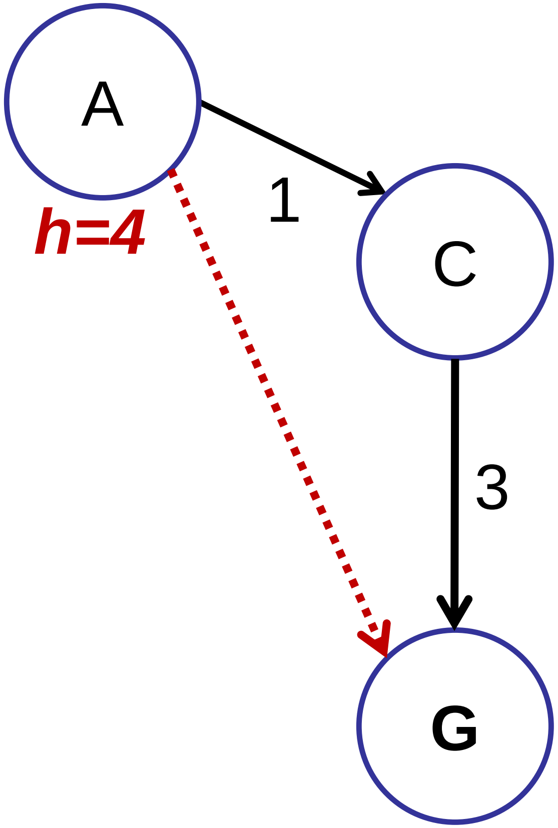
State space graph



Search tree

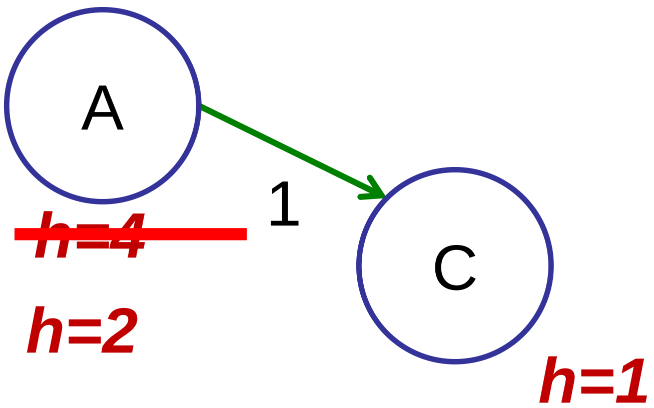


Consistency of Heuristics



- Admissibility: heuristic cost <= actual cost to goal
  - h(A) <= actual cost from A to G

Consistency of Heuristics



- Stronger than admissibility
- Definition:
- heuristic cost <= actual cost *per arc*
- h(A) - h(C) <= cost(A to C)
- Consequences:
  - The f value along a path never decreases
  - A* graph search is optimal

Optimality

- Tree search:
  - A* is optimal if heuristic is admissible (and non-negative)
  - 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

Trivial Heuristics, Dominance

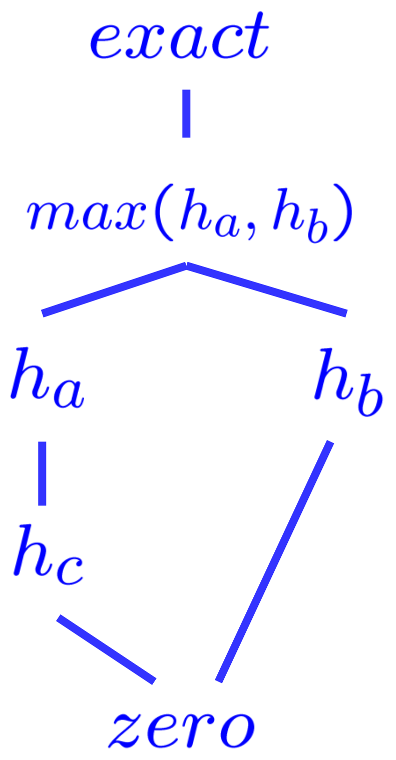
- Dominance: $h_a \geq h_c$ if

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

- 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



Summary: A*

- 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