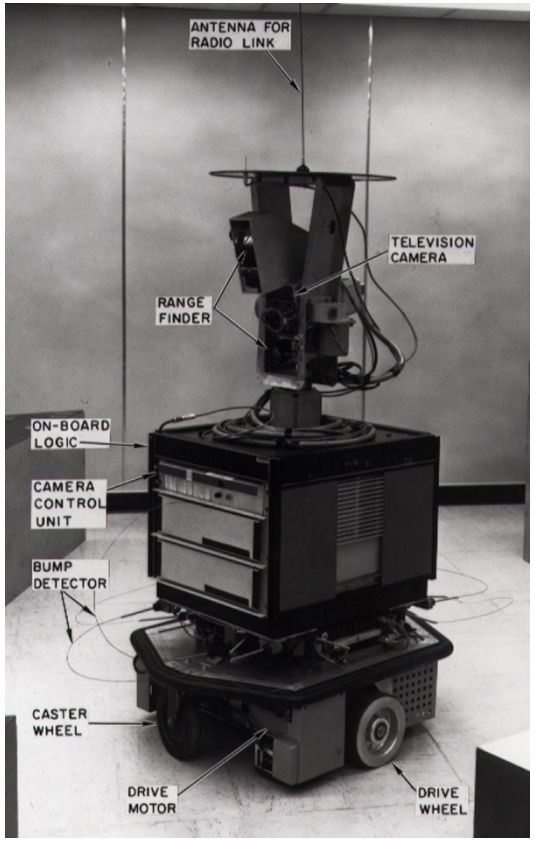
COMP 341 Intro to AI
Informed Search



Asst. Prof. Barış Akgün

Koç University

Search Algorithms

- Given a “frontier” (unexplored children nodes of explored nodes)
  - Select a state (if empty return failure, ignore if expanded (visited) before)
  - Return the solution if it’s a goal
  - Else, expand its successors (graph search: ignore if expanded before) and add to frontier
  - Repeat
- Selection strategy results in different algorithms
- Tree vs Graph!

Uninformed Search Algorithms

- DFS – Stack (recursive version exists as well)
- BFS – Queue

} Pros and Cons?

- ID-DFS: Iterative Deepening DFS

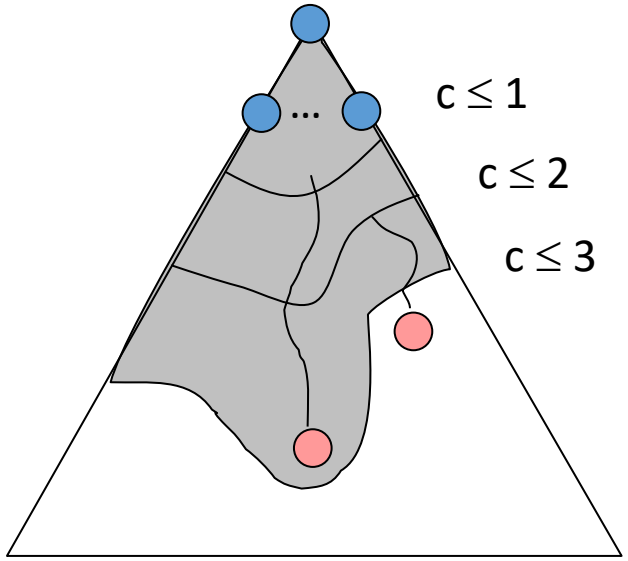
- Uniform Cost Search – Priority Queue

- Warning: No running away from worst-case exponential time complexity!

Uniform Cost Search

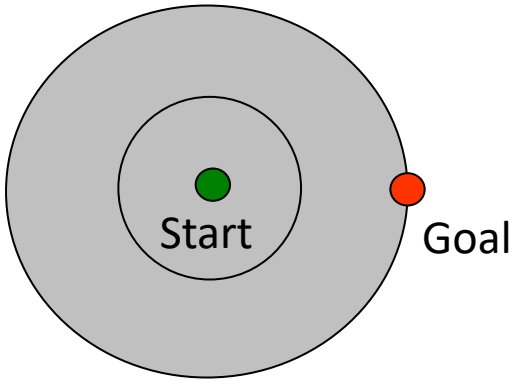
Idea: expand the lowest cost node first

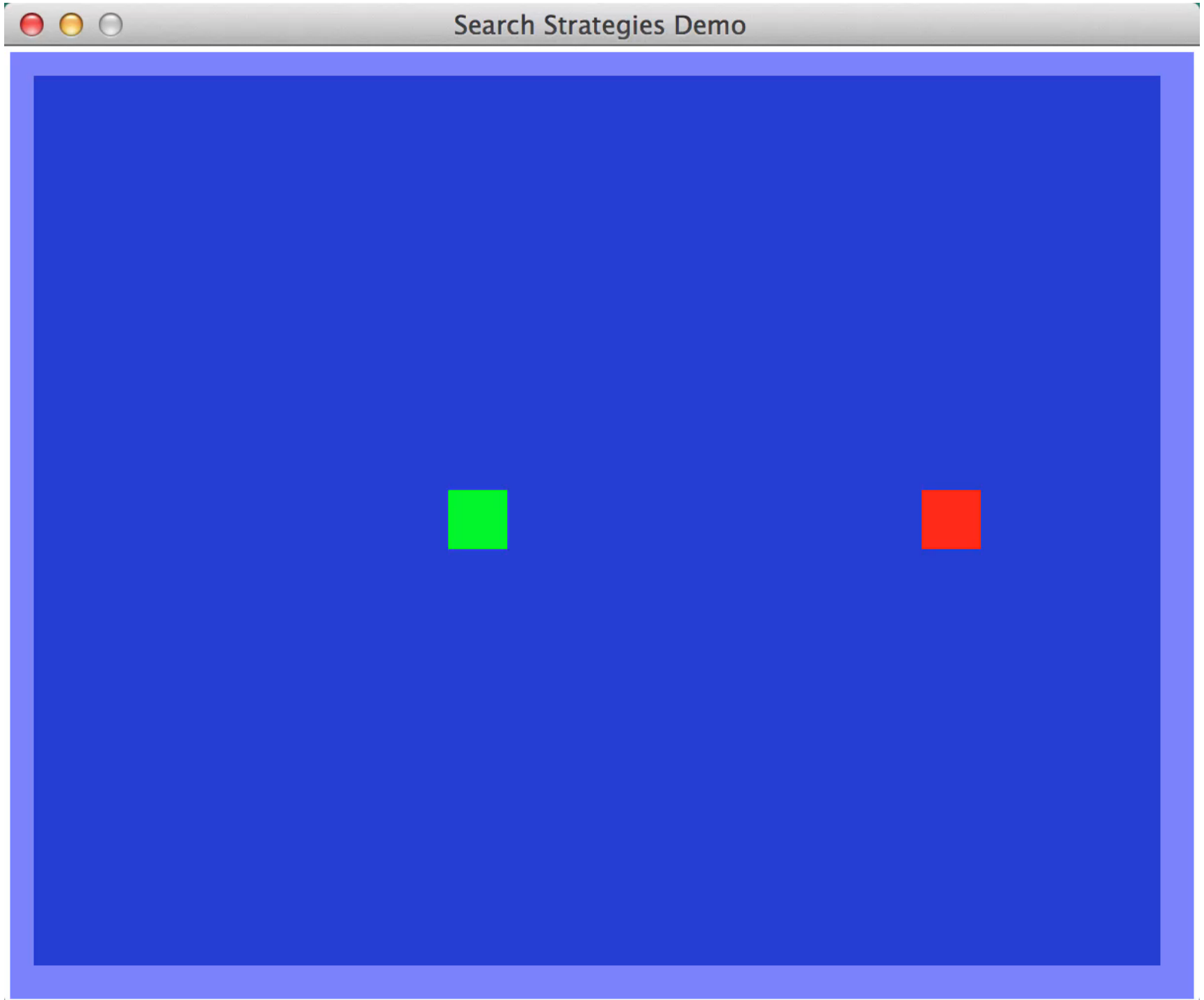
Implementation: Frontier is a priority que based on the cumulative cost (f(s) = g(s))

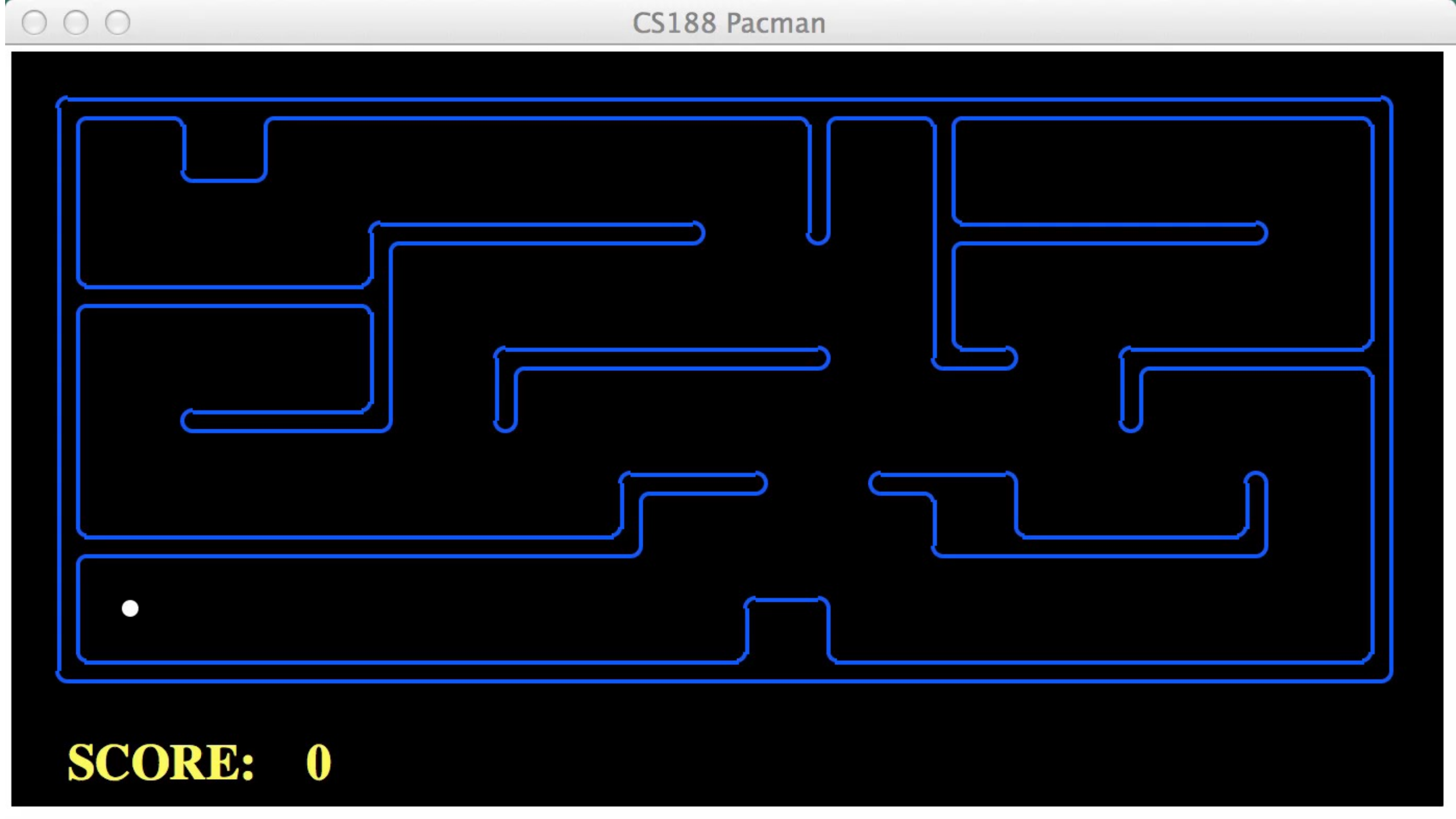


Good: Complete and Optimal

Bad: Explores in every direction (Uninformed)




Search Strategies Demo


CS188 Pacman
SCORE: 0

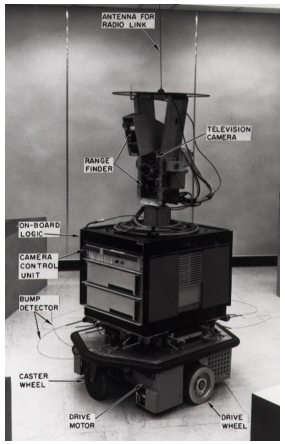
Informed Search

- What if know more?
  - Something about the given problem
  - Domain knowledge

- Look at promising nodes first
  - The ones that we think are *closer* to the goal!

- What is promising? Heuristics …
- Informed about a particular problem

Cost + Heuristic



Start

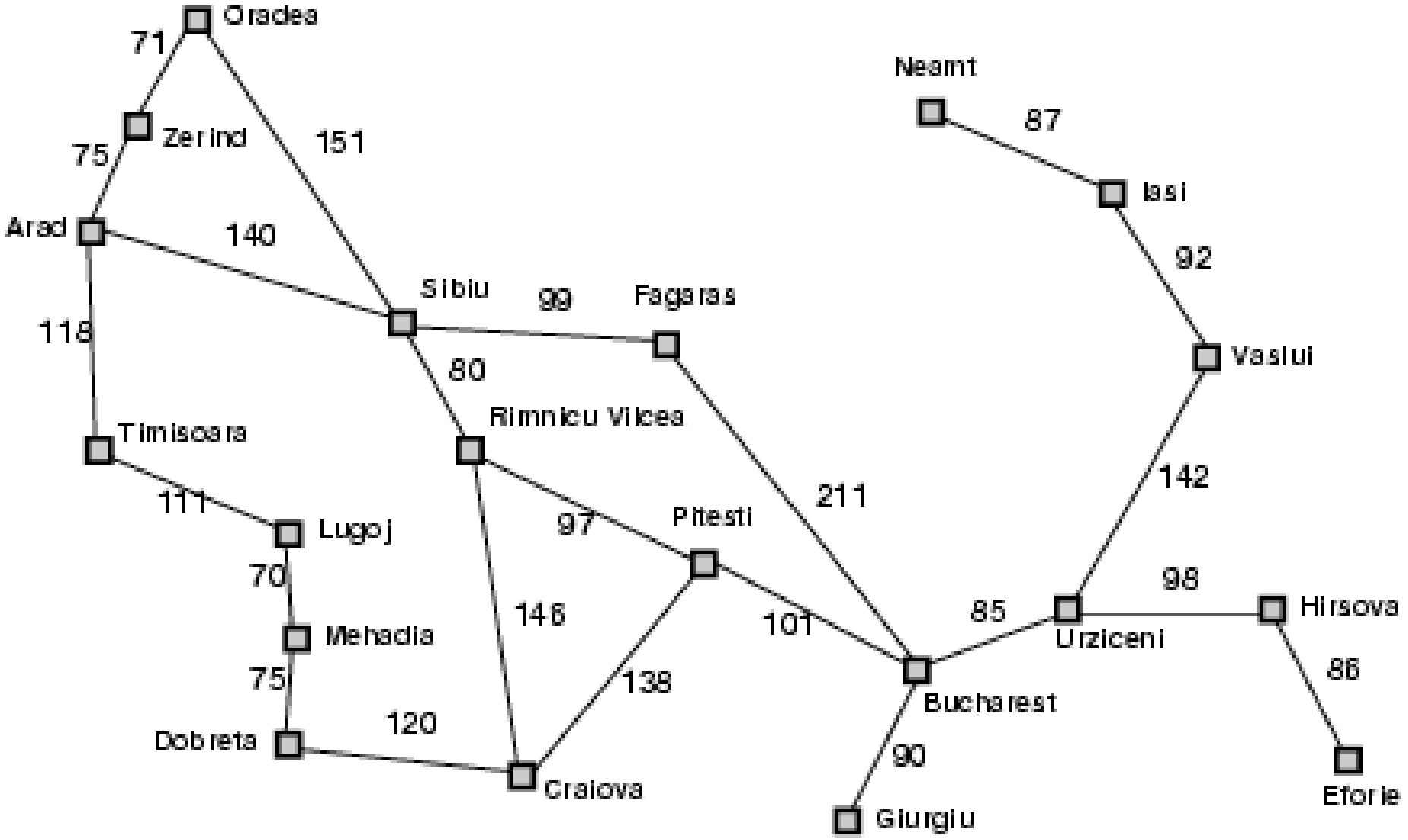
Current

Goal

Calculated Cost

Heuristic: Estimated Cost

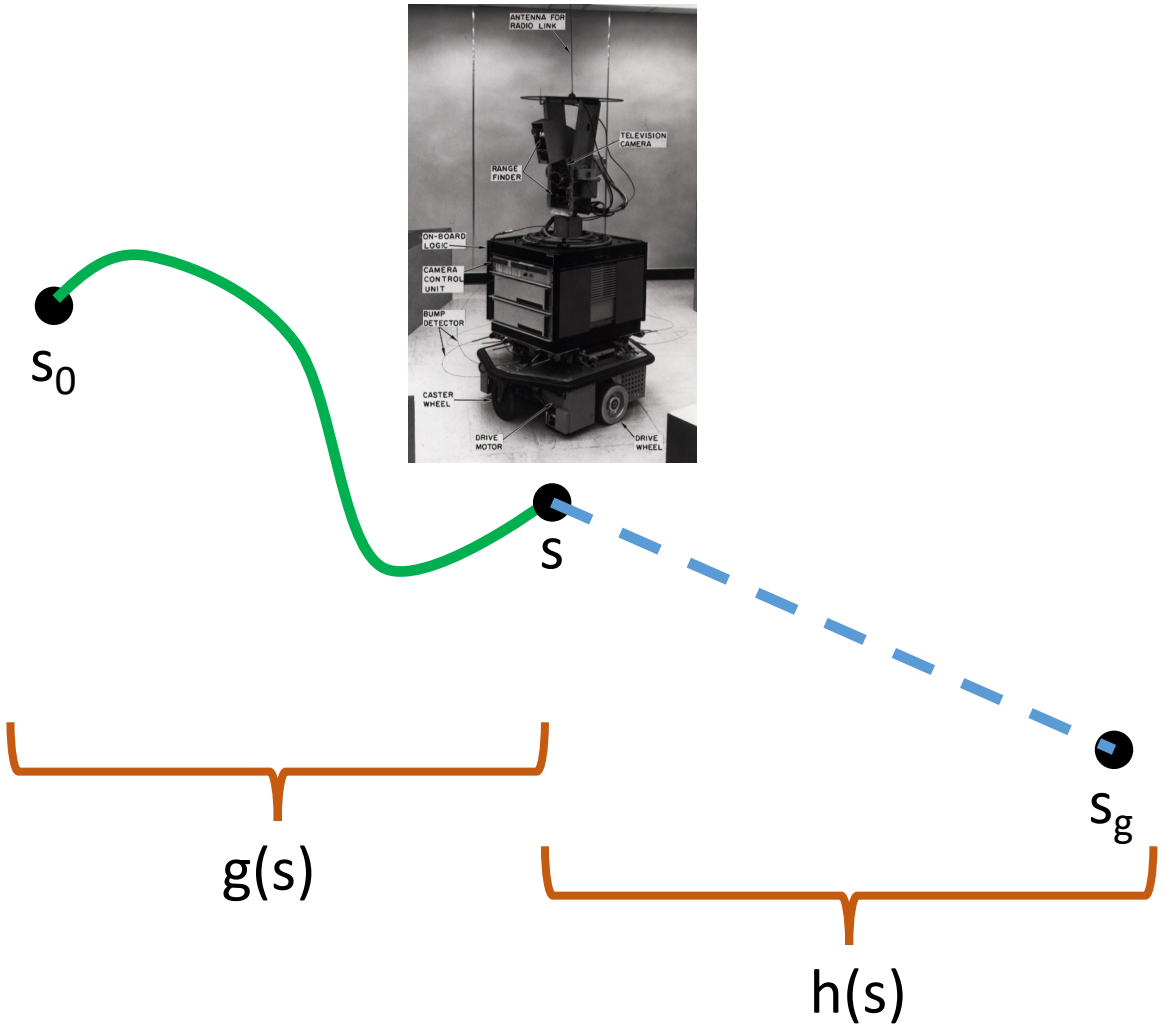
Example



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Notation

- Frontier: Priority queue
- f(s): Evaluation function for the queue
- g(s): Cumulative cost
- h(s): Desirability, closeness estimate etc. (heuristic!)
- Note: UCS priority queue f(s) = g(s)



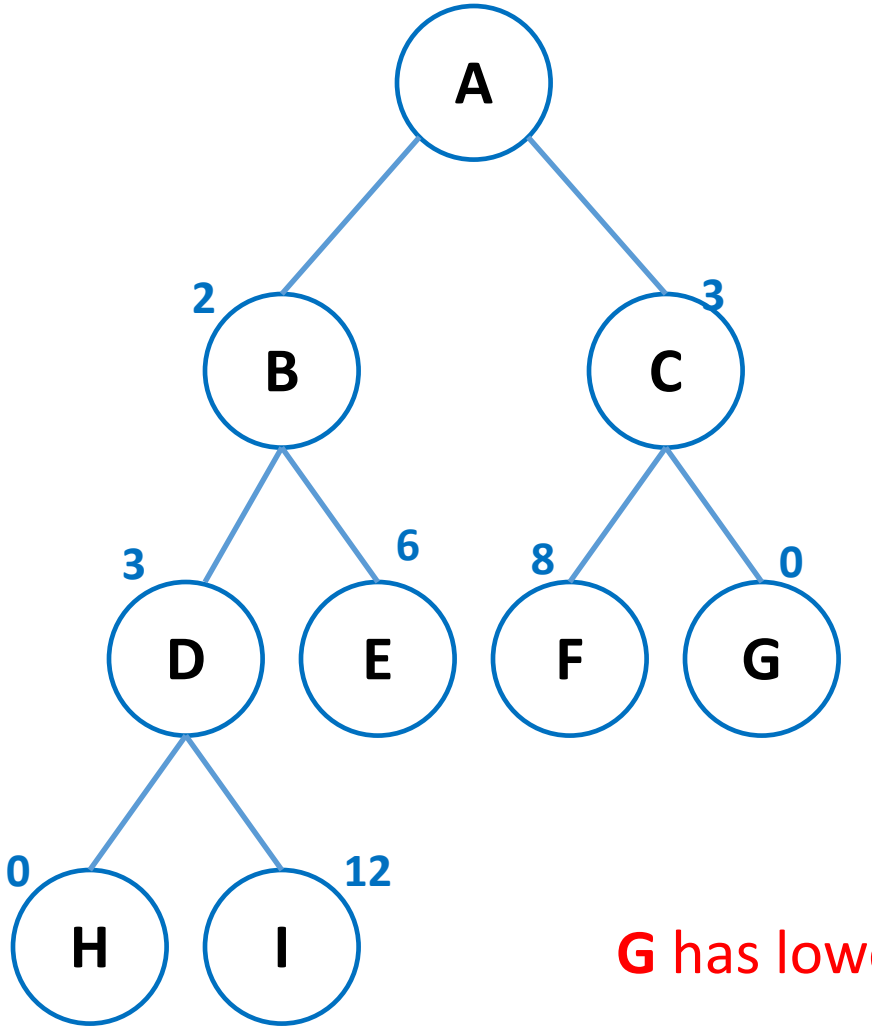
The figure shows a single goal state but this does not have to be the case! If there are multiple goals, heuristics will need to deal with them accordingly.

Greedy Best-First Search

Idea: expand the most promising node first

Implementation: Frontier is a priority que based only on the heuristic (f(s) = h(s))

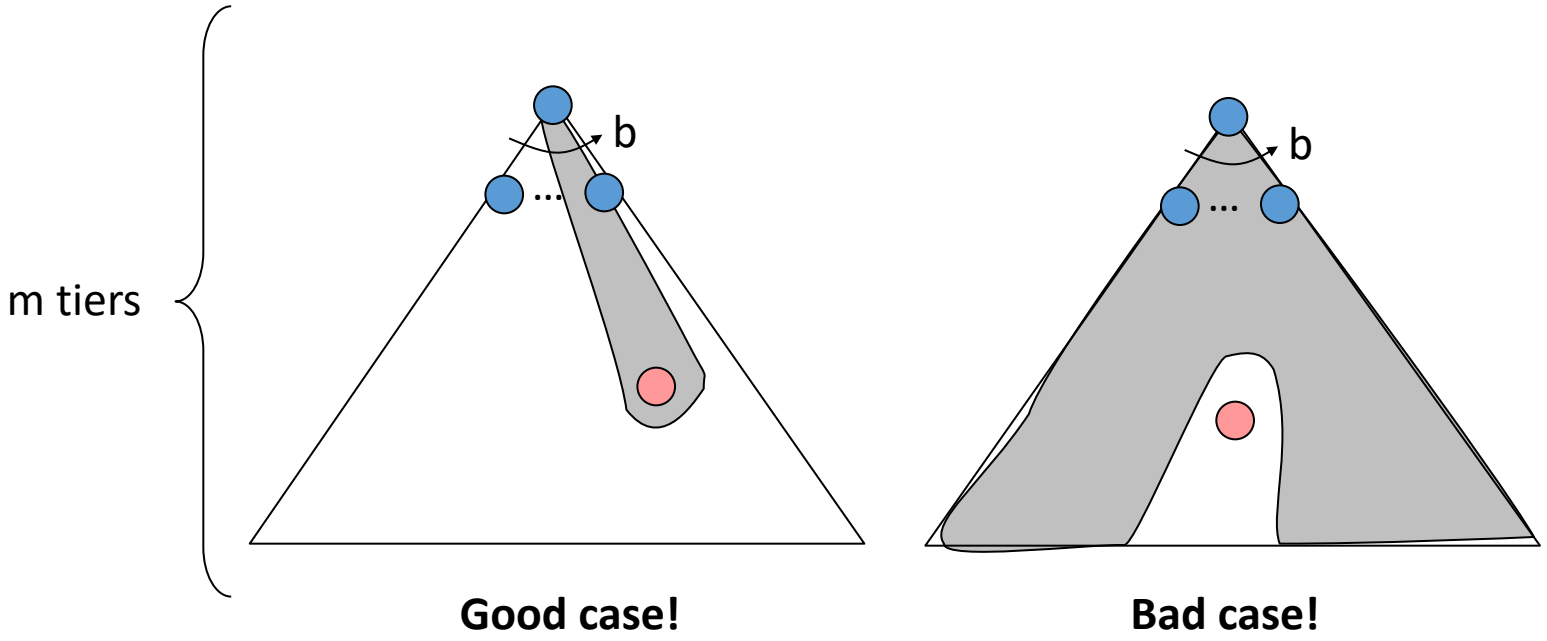
Sort of a guided DFS



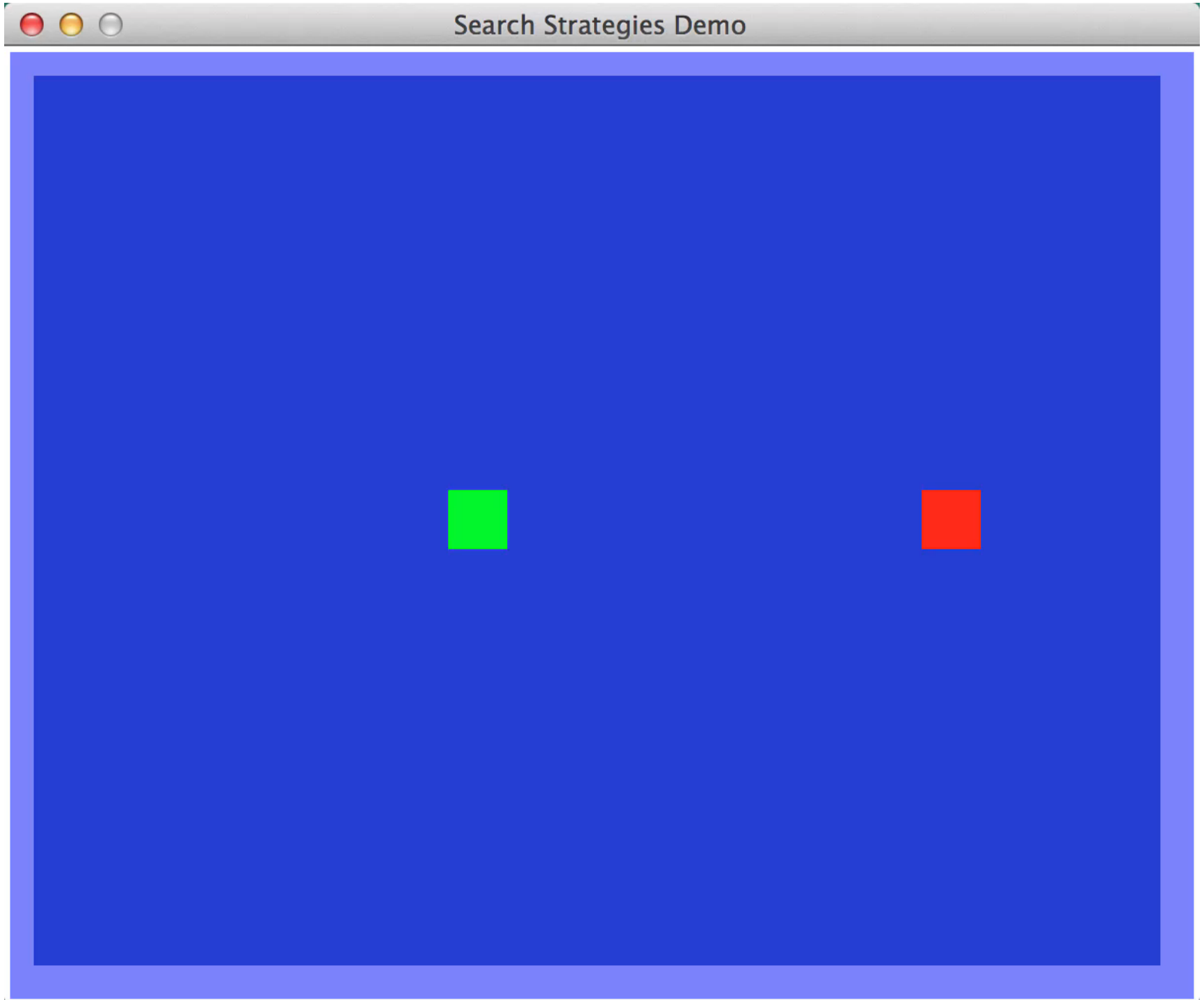
G has lower cost than **H**!

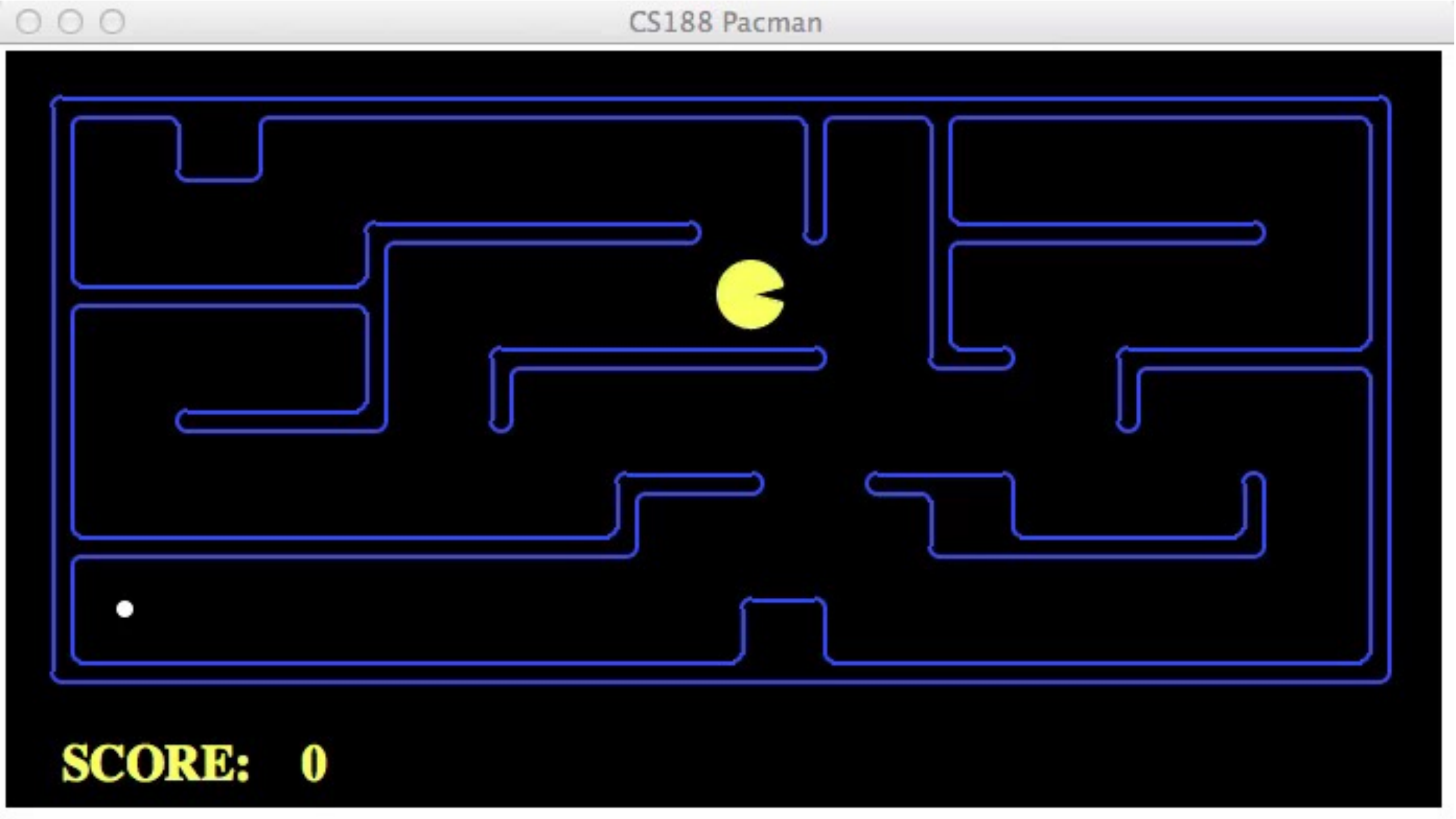
Greedy Best-First Search Properties

- **Completeness ?**
  - Only if cycles are prevented
- **Optimality ?**
  - **No**
- **Time Complexity ?**
  - $O(b^m)$
- **Space Complexity ?**
  - $O(b^m)$



These are ***worst case***. In practice, good heuristics can find reasonable solutions quickly!


Search Strategies Demo


CS188 Pacman
SCORE: 0

A* Search

+

=

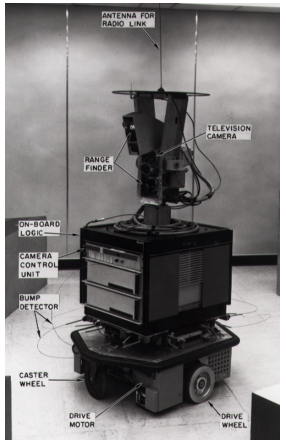
UCS
g(s)

Greedy
h(s)

A*
g(s) + h(s)

Brief History: Dijkstra (UCS variant) -> A1 -> A2, then A2 became A*

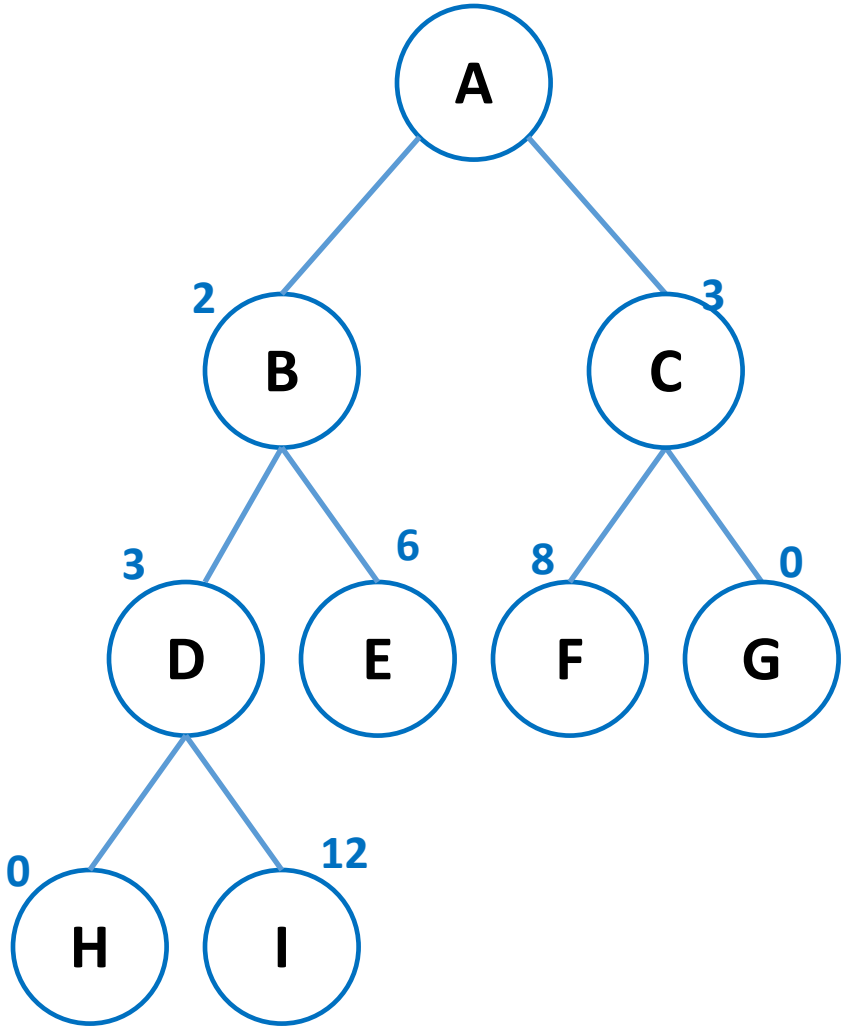
Developed for Shakey the Robot!



A* Search

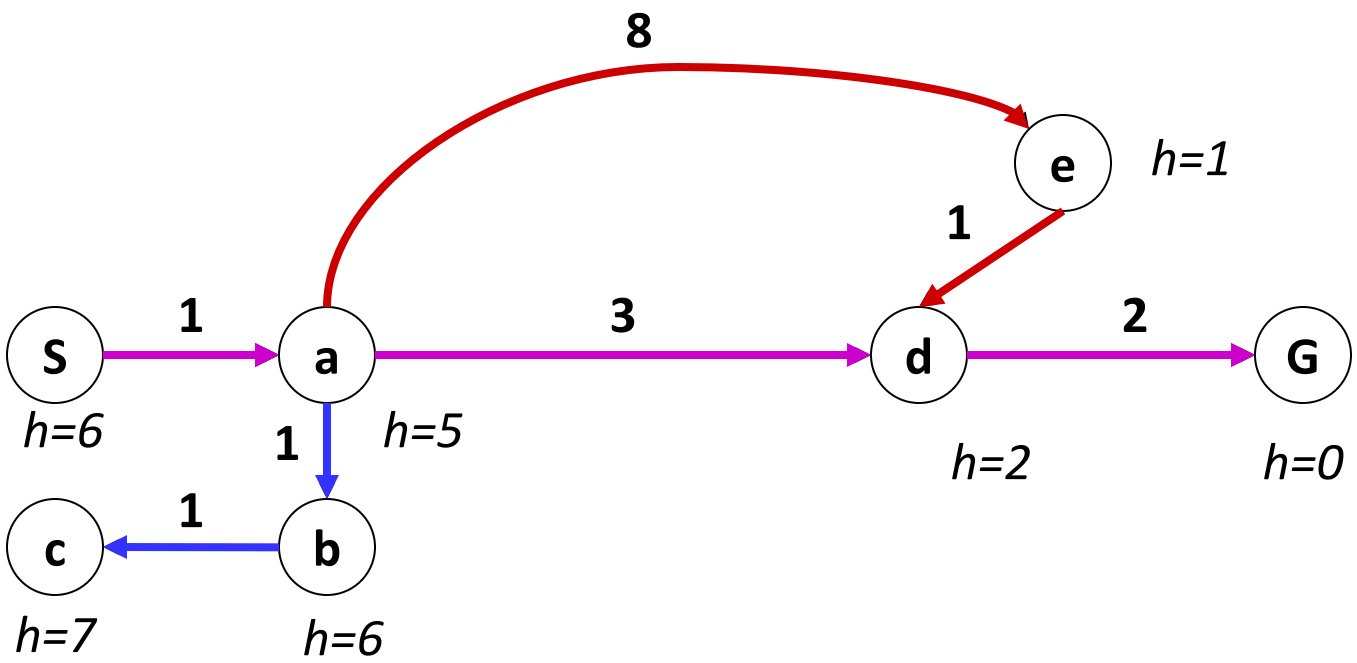
Idea: expand the most nodes that is both low cost **and** promising

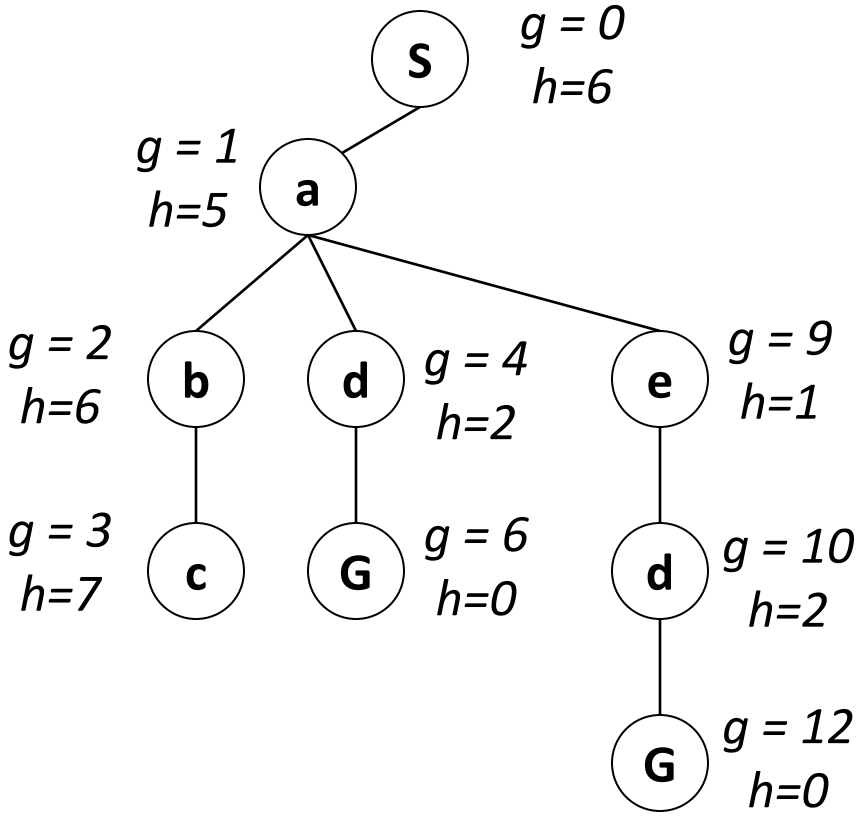
Implementation: Frontier is a priority que based on the cumulative cost + the heuristic (f(s) = g(s) + h(s))



Let's take another look

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



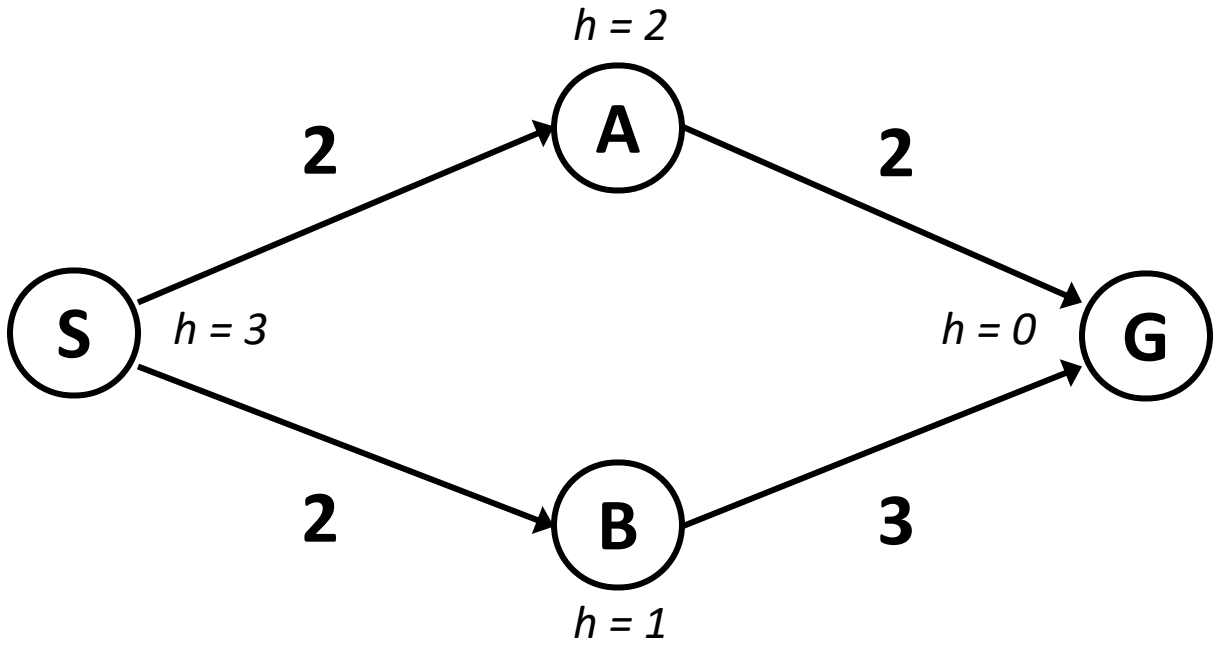


- A* Search orders by the sum: f(n) = g(n) + h(n)

Example: Teg Grenager

When should A* terminate?

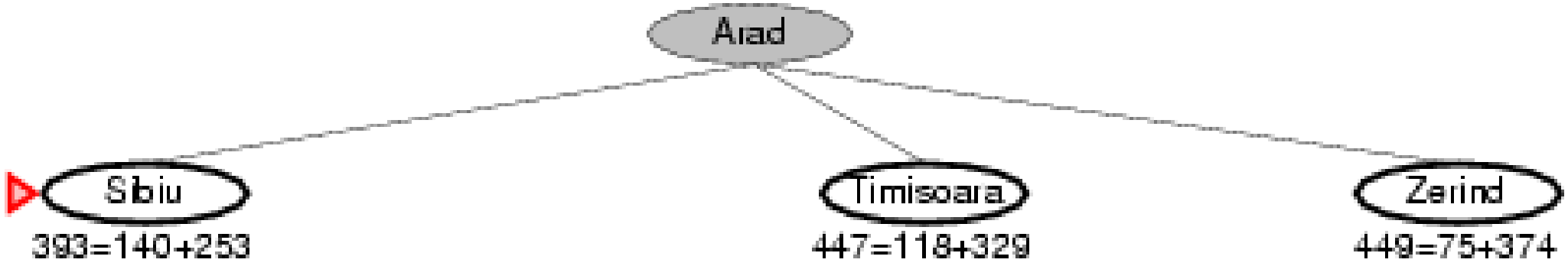
- Should we stop when we enqueue a goal?



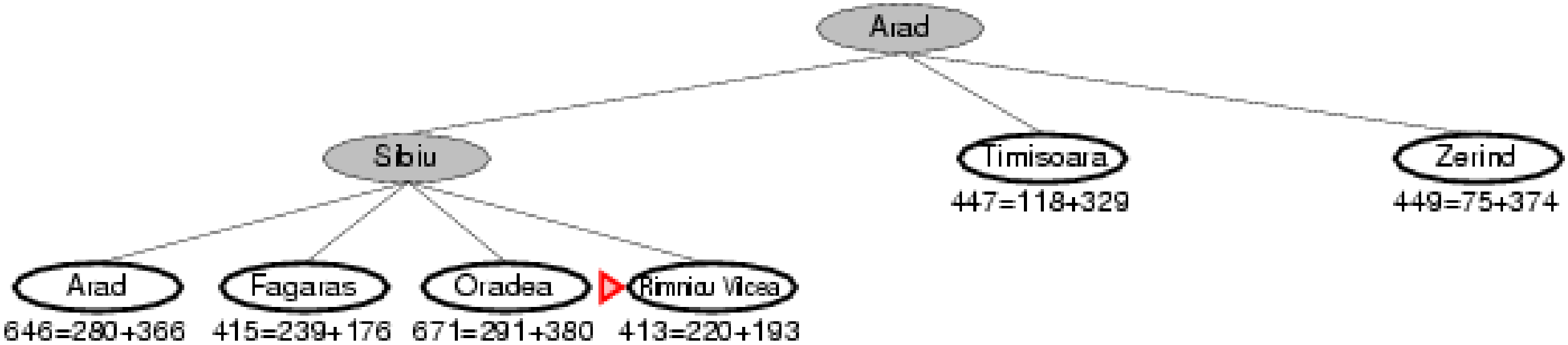
- No: only stop when we dequeue a goal

Yet Another Example

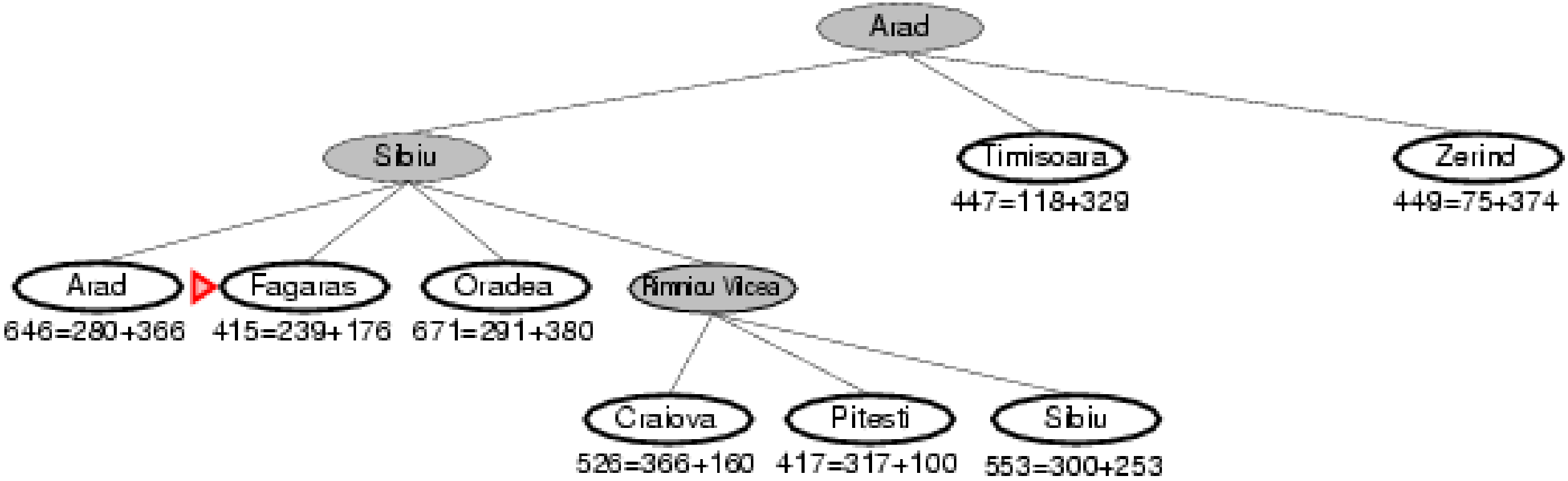
Yet Another Example



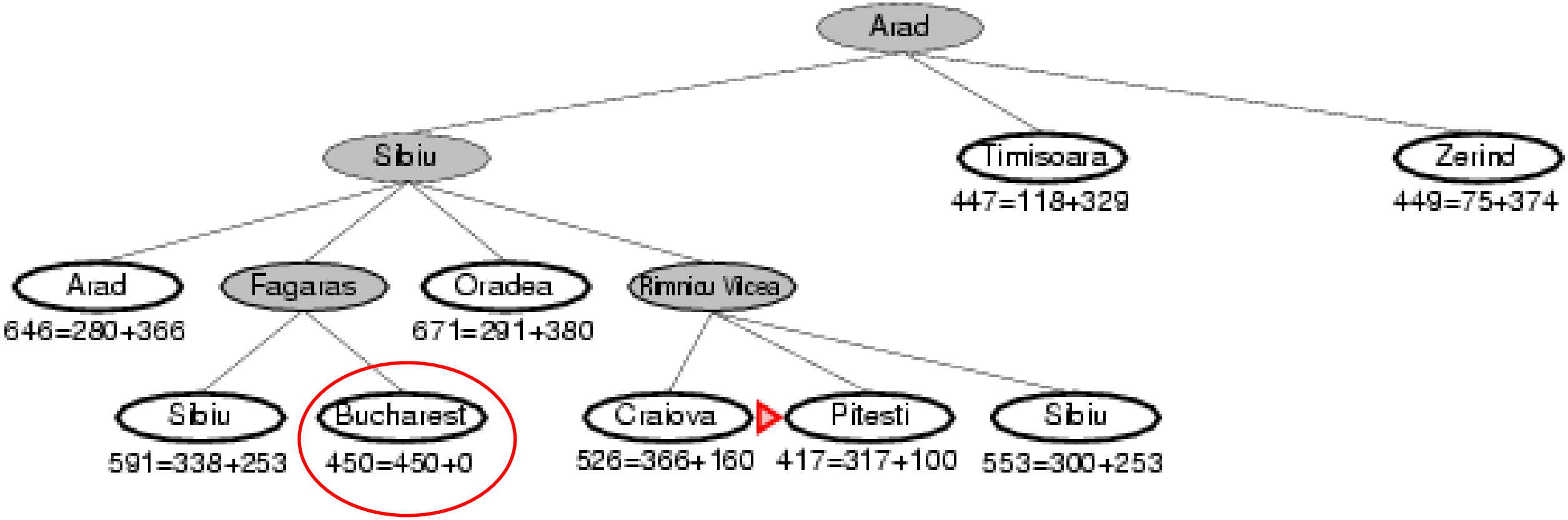
Yet Another Example



Yet Another Example

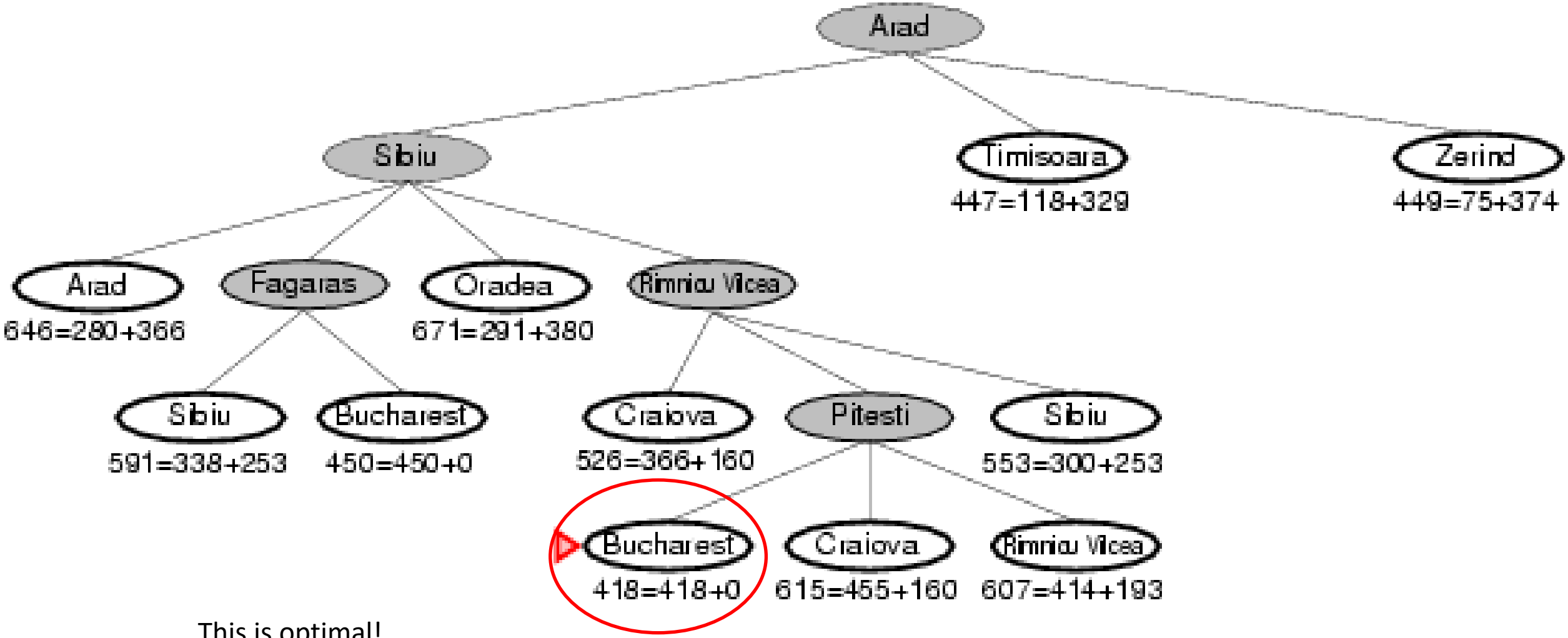


Yet Another Example



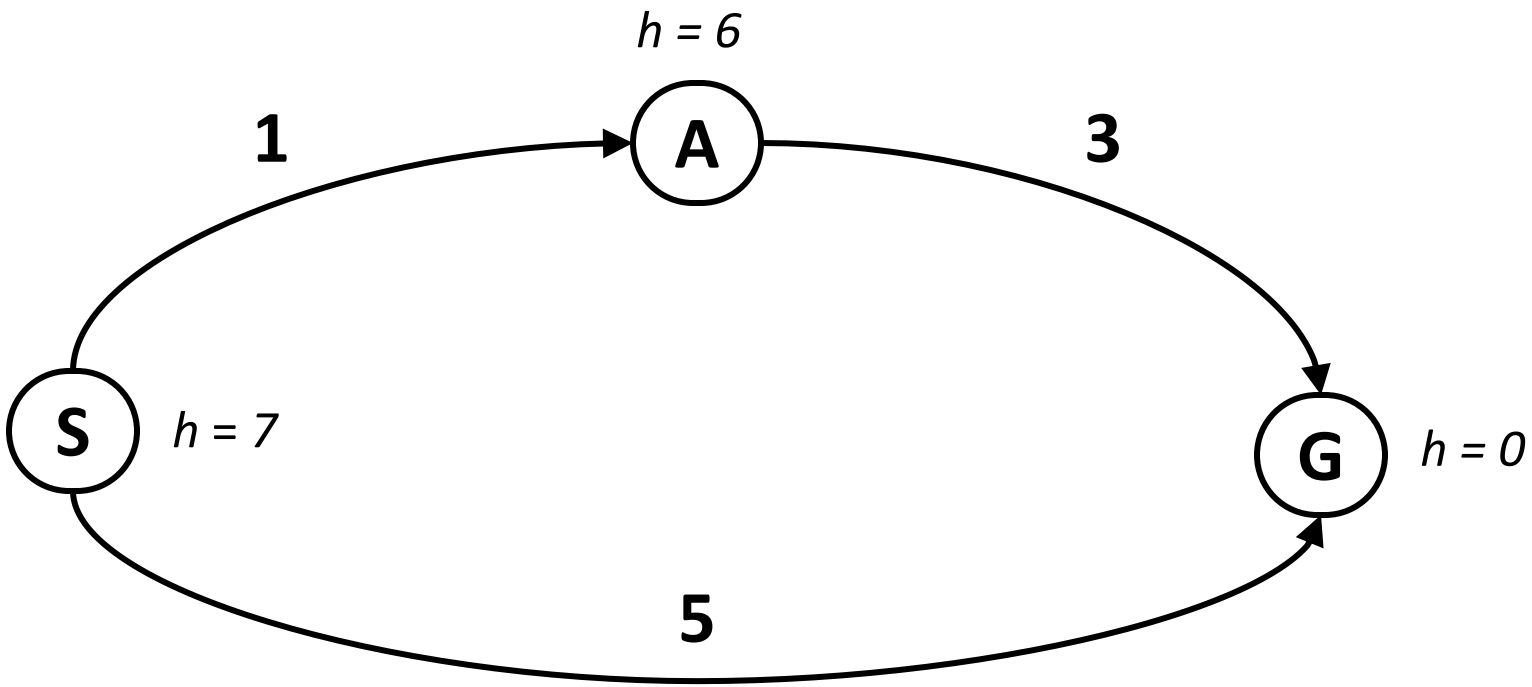
Is it optimal?

Yet Another Example



This is optimal!

Is A* Really Optimal?



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

Admissible Heuristic

- h(s):
  - **Under-estimate** the cost to goal
  - **Zero** for any goal state
  - Non-zero for all others

true cost to nearest goal

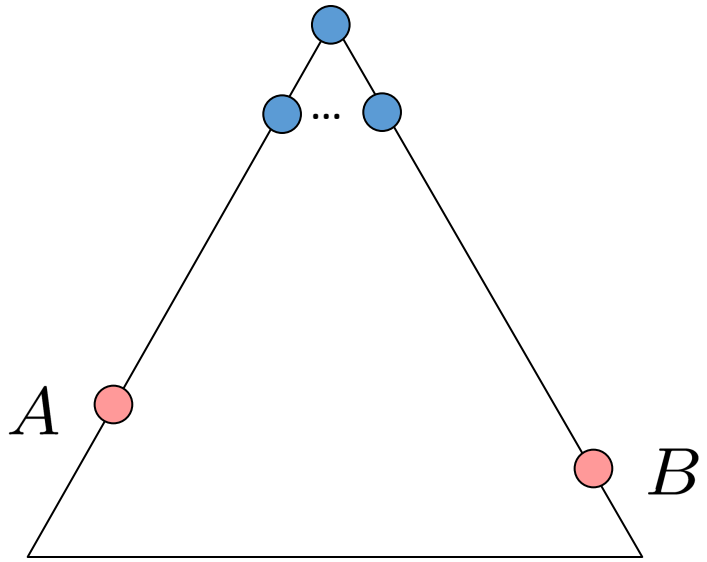
$$0 \leq h(s) \leq h^*(s)$$

- Makes A* tree-search **optimal** and **complete**!

- Art of A*: Finding a heuristic!

Optimality of A* Tree Search

- Path to goals are lowest cost (g(s) part)
- What if there are multiple goals? Will A* find the optimal one?



Assume:

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

Claim:

- A will exit the frontier data structure before B

Optimality of A* Tree Search: Blocking

Proof:

$f(s) = g(s) + h(s)$ definition

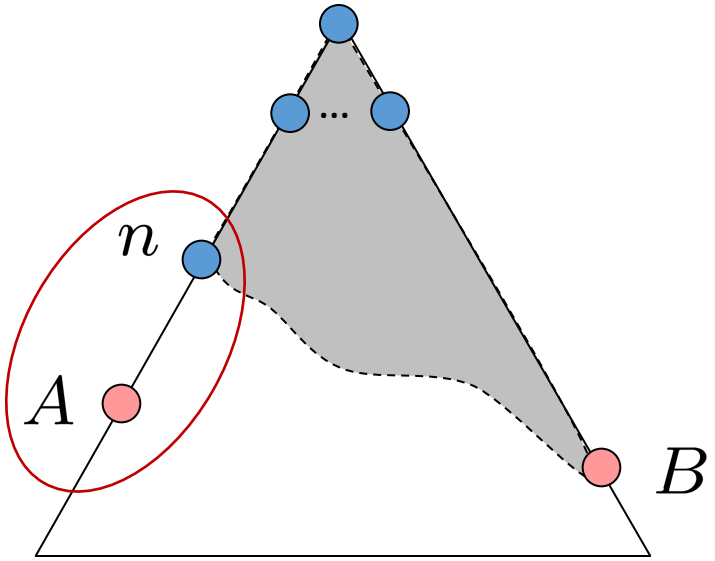
$f(B) = g(B)$ h$(B) = 0$ since B is a goal

$f(B) \geq g(A)$ B is suboptimal

$f(B) \geq f(n)$, since h is admissible

n will be expanded before B

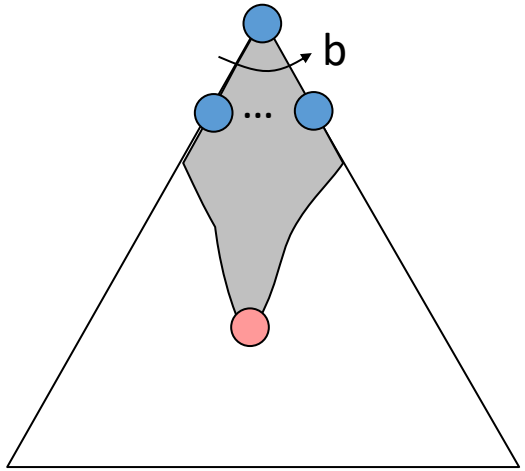
A* will never expand B!



Let n be an unexpanded node on a shortest path to A

A* Search Properties

- **Completeness ?**
  - Yes (some conditions)
- **Optimality ?**
  - Depends on h(s)!
- **Time Complexity ?**
  - Depends on h(s)!
- **Space Complexity ?**
  - Exponential

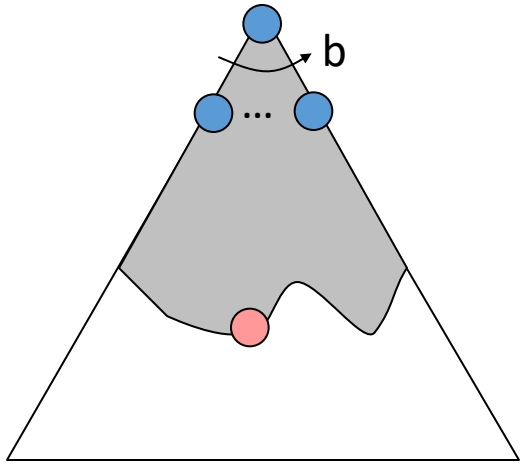


Is A*the be all, end all search algorithm?

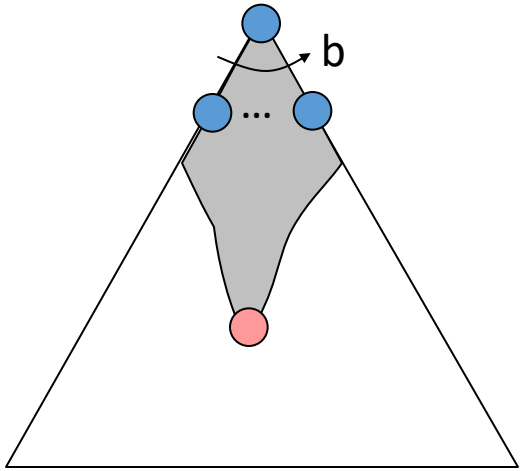
No! Unfortunately, we run out of space before we run out of time

A* vs UCS

Uniform-Cost

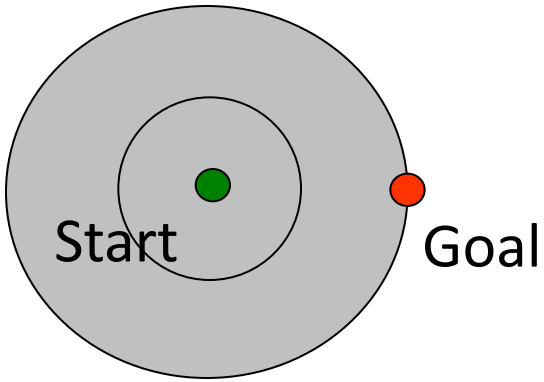


A*

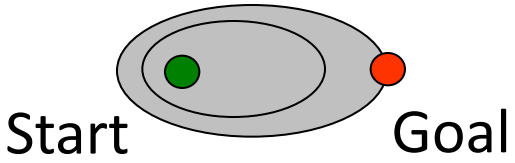


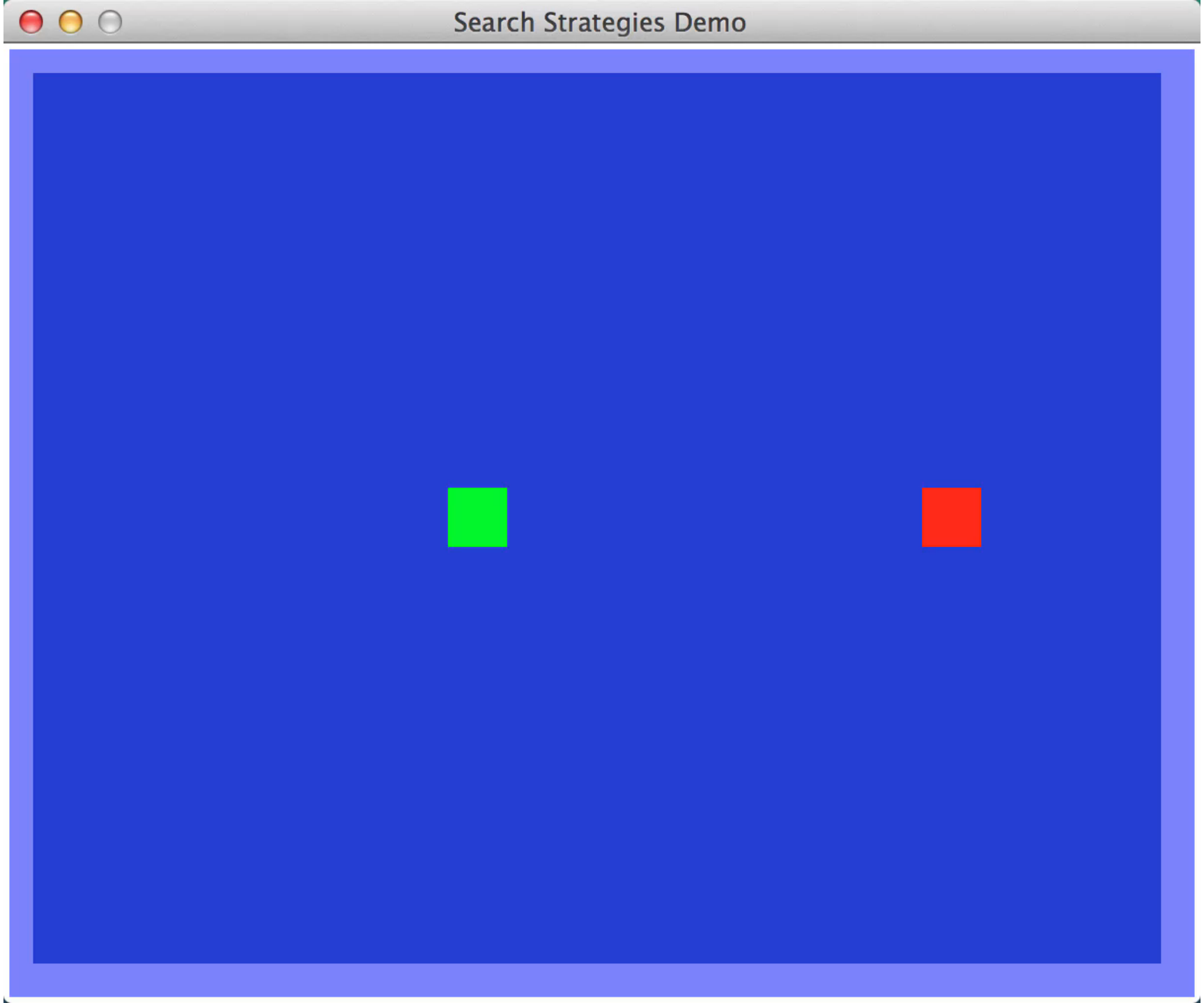
UCS vs A* Contours

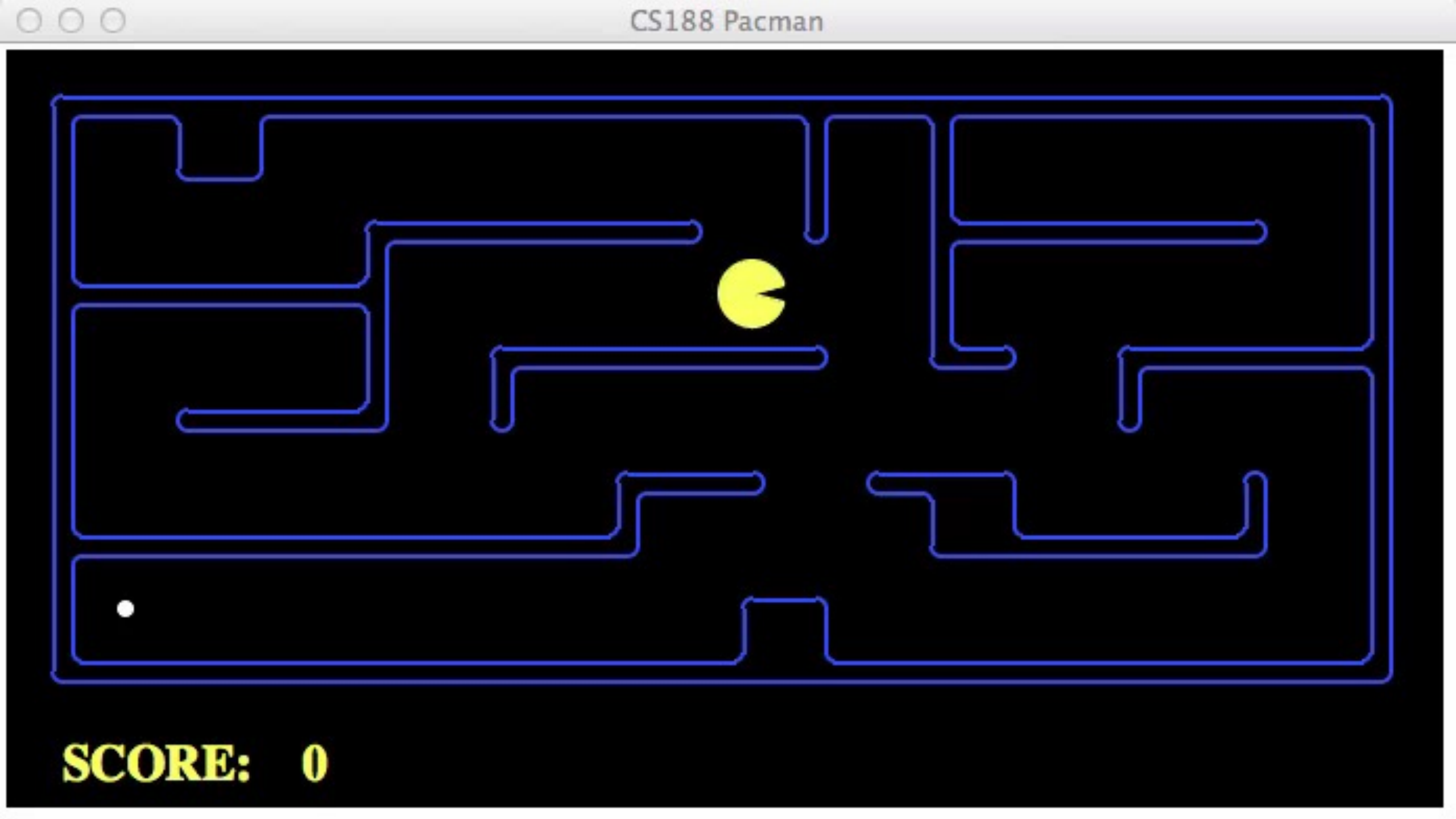
- Uniform-cost expands equally in all "directions"



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



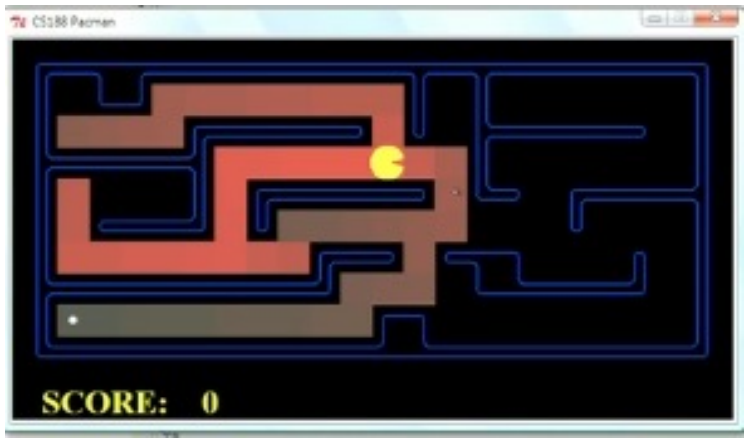

Search Strategies Demo


CS188 Pacman
SCORE: 0

UCS vs Greedy vs A*

Greedy

Uniform Cost



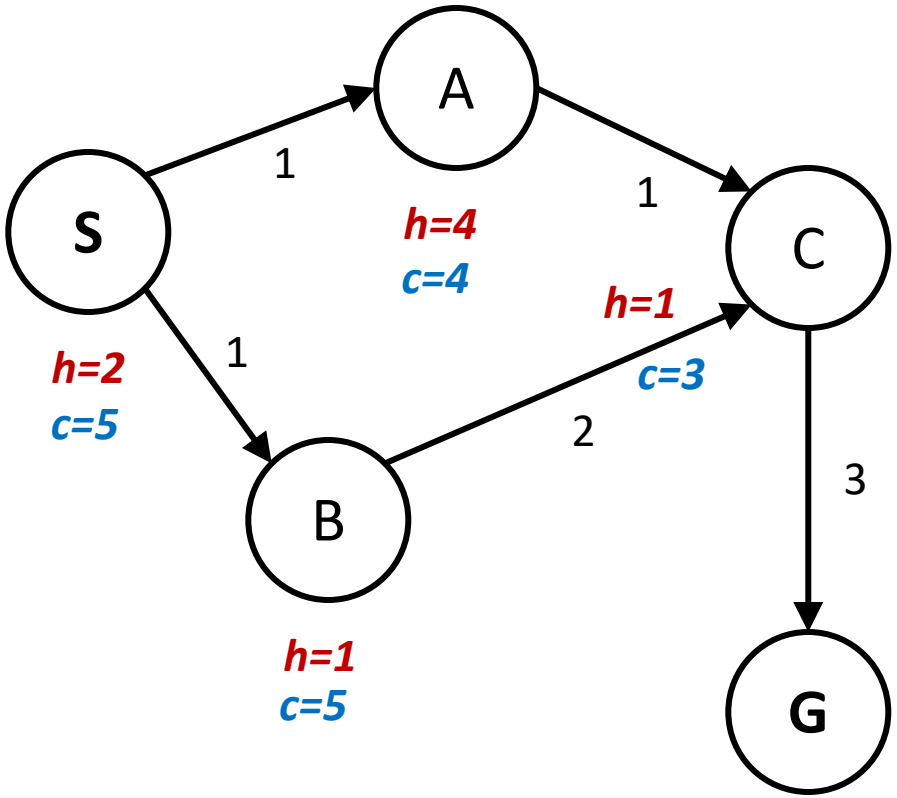
A*

A* Applications

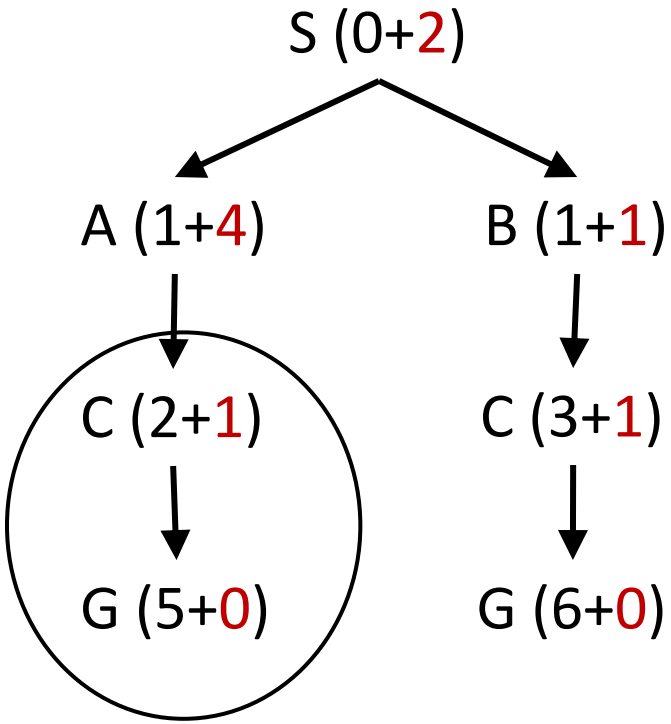
- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- …

What about A* Graph Search?

State space graph



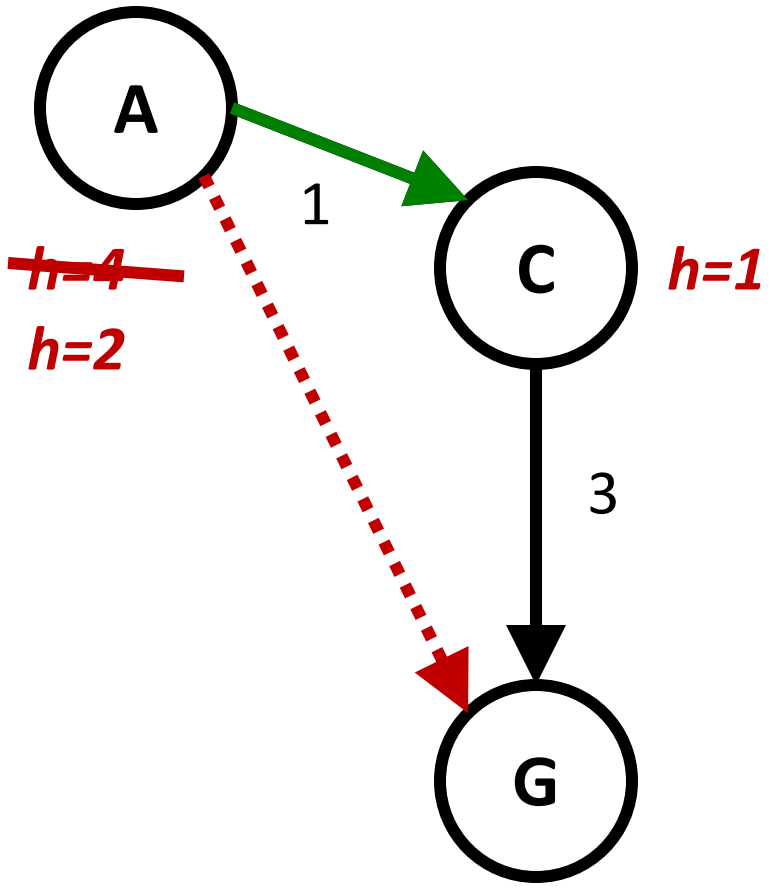
Search tree



Is ***h*** admissible?

This part would not be explored with A* Graph search since C is already expanded! (i.e. C has been put in the explored set)

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 "arc" cost ≤ actual cost for each arc

    h(A) – h(C) ≤ cost(A to C)

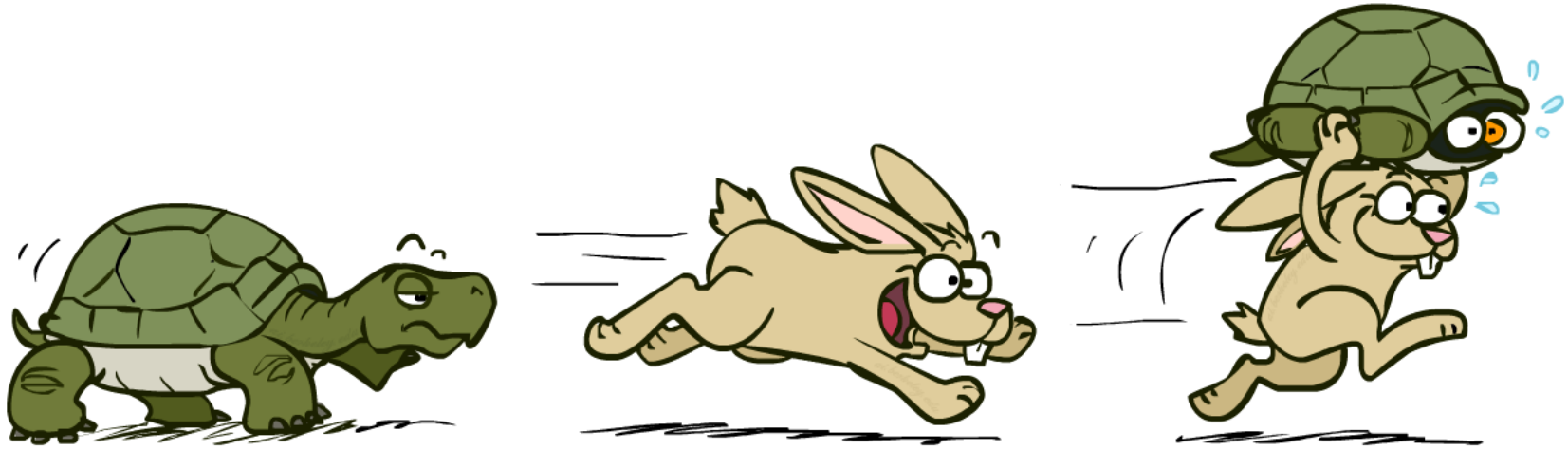
- Consequences of consistency:
  - The f value along a path never decreases

    h(A) ≤ cost(A to C) + h(C)
  - Makes A* graph search optimal

- Consistency implies admissibility!

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



Dominance

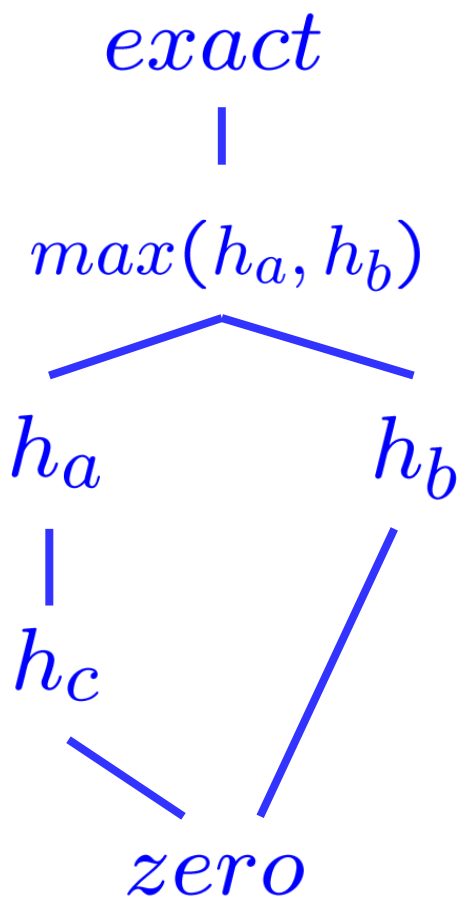
- Dominance: $h_a \geq h_c$ if

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

- Heuristics form a semi-lattice:
  - Max of admissible heuristics is admissible

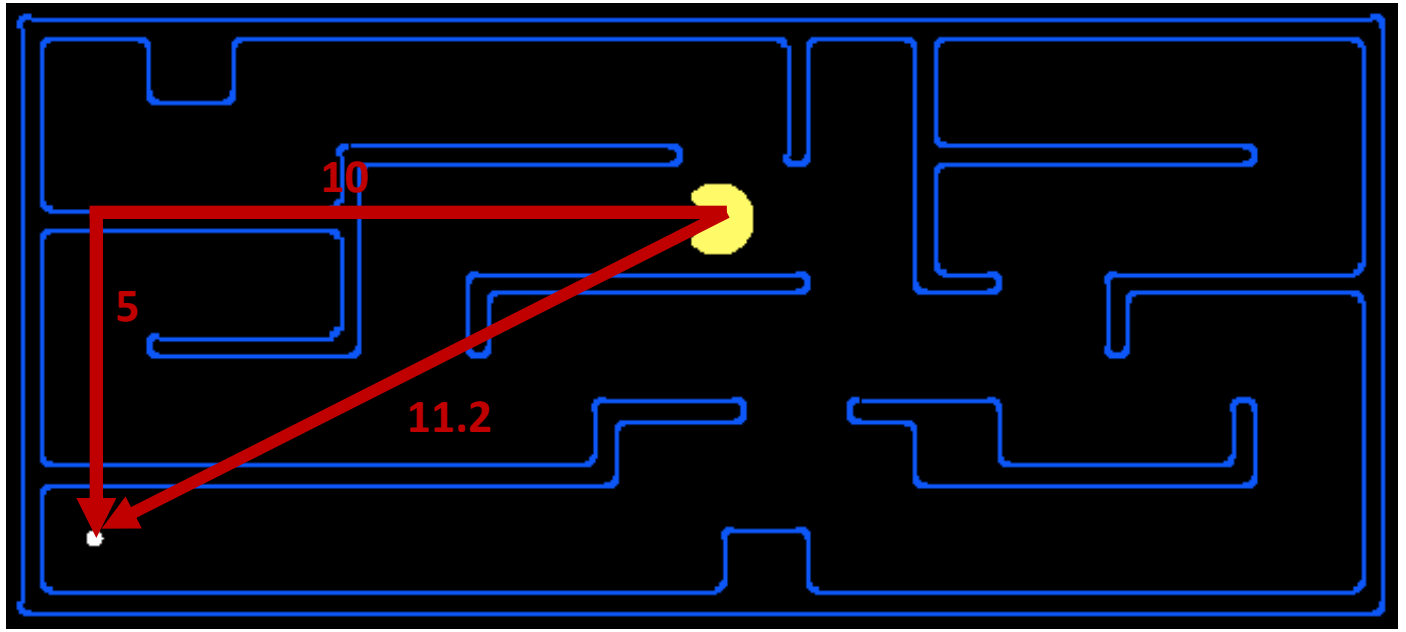
$$h(n) = max(h_a(n), h_b(n))$$

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



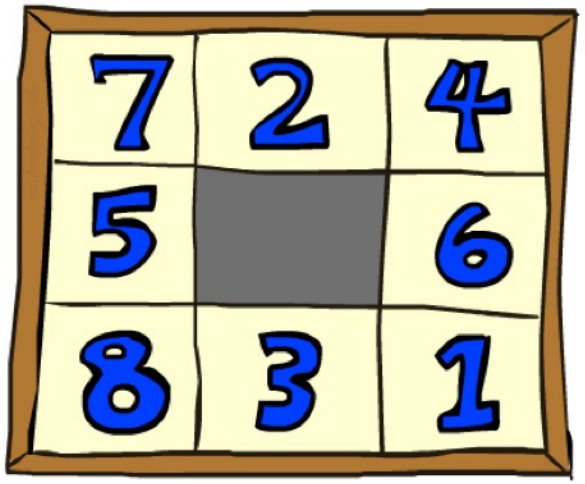
Creating Heuristics

- Some common ones for distances: Euclidean, Manhattan

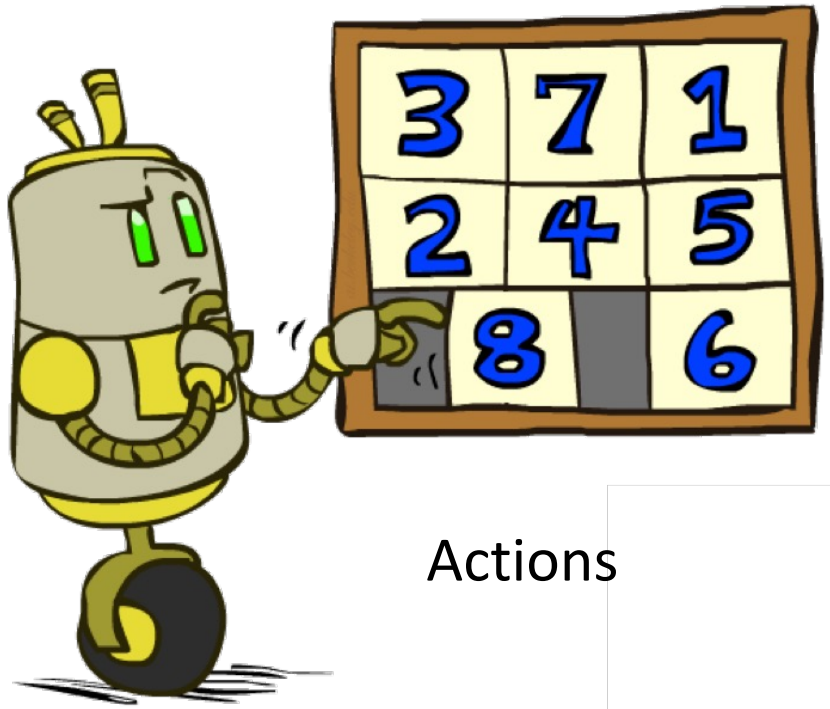


- Solutions to *relaxed* problem
- Useful inadmissible heuristics!

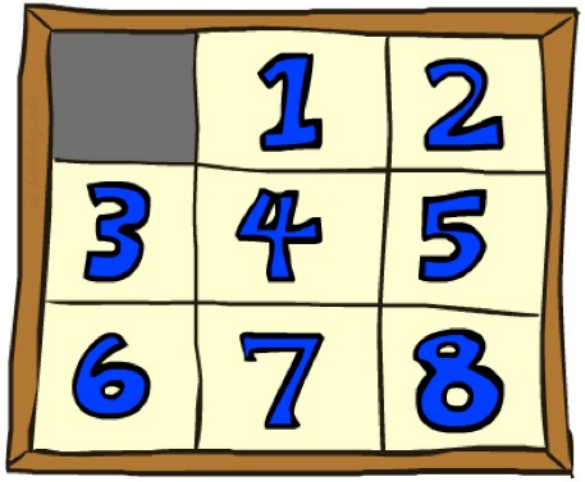
Example: 8 Puzzle



Start State



Actions



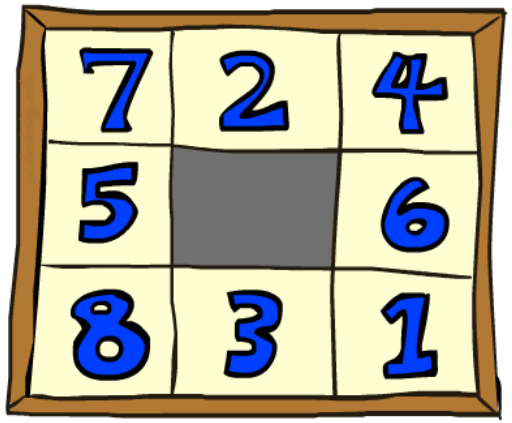
Goal State

Let's formulate 8 Puzzle

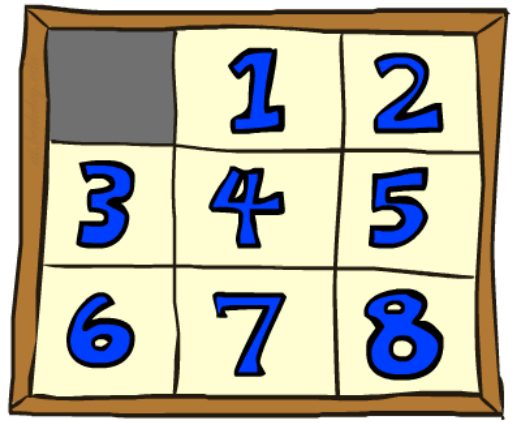
- State space?
- How many states?
- Action Space?
- Average branching factor?
- What should the costs be?

8 Puzzle I

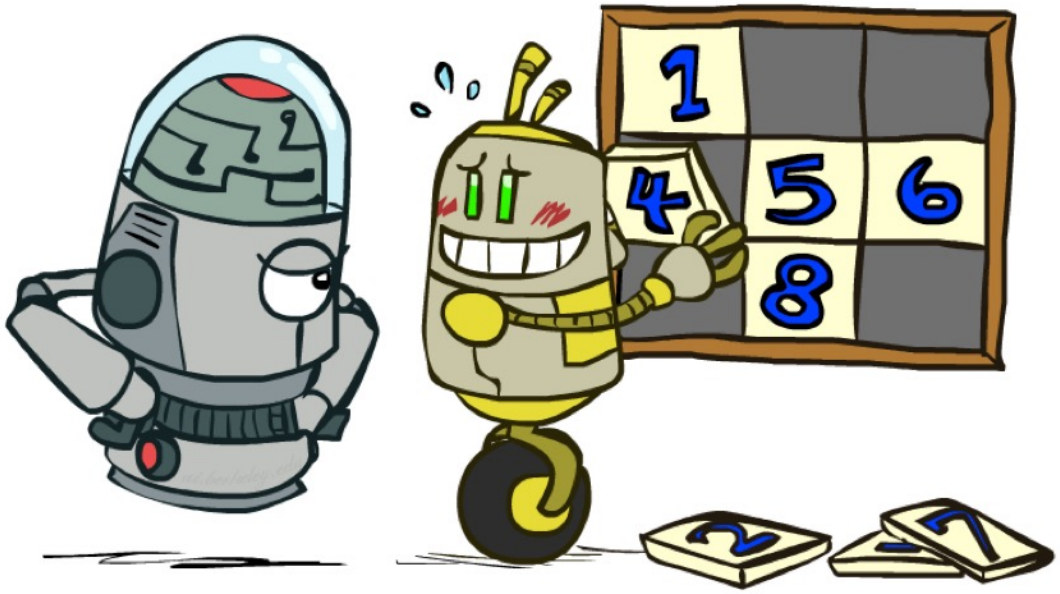
- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = 8
- This is a *relaxed-problem* heuristic



Start State



Goal State

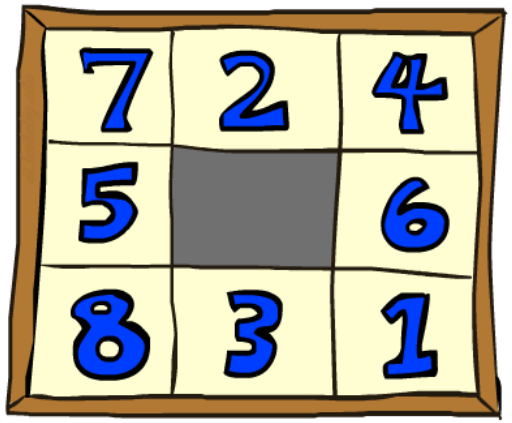


	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

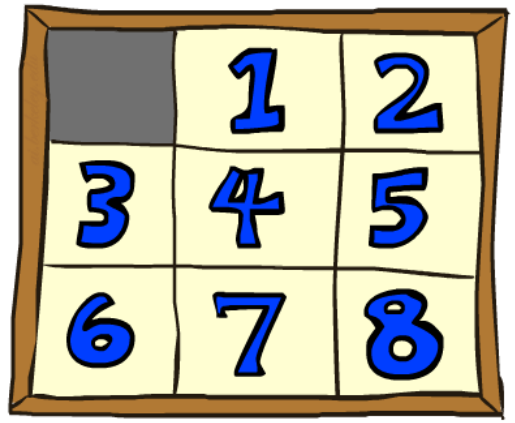
Statistics from Andrew Moore

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

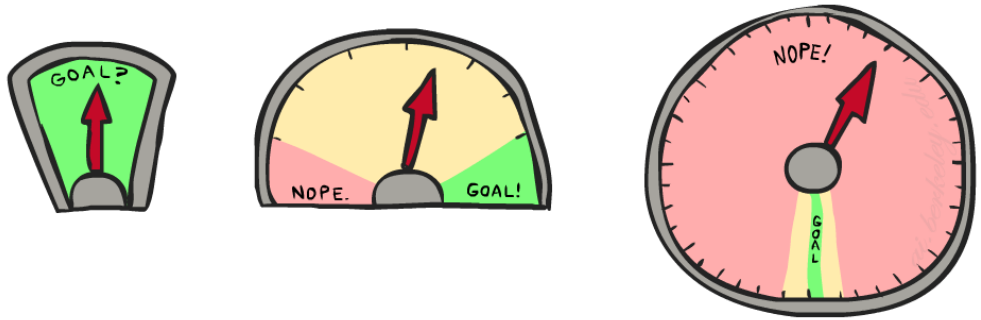


Goal State

	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
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?

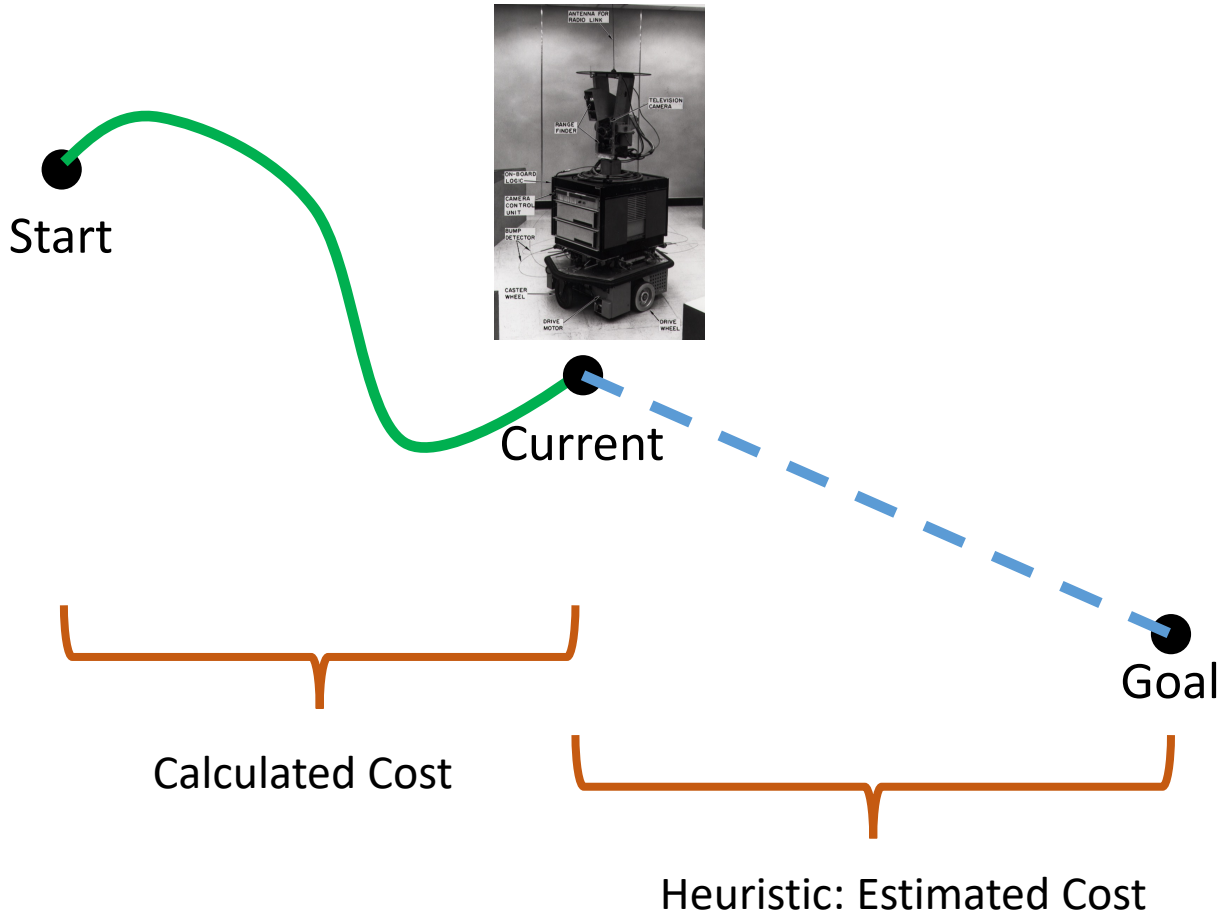


- With A*: a trade-off between quality of estimate and work per node
  - As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

Heuristics Recap

- How promising is a given state?
  - Admissibility/Consistency
  - Dominance

- Design:
  - Solution Costs to Relaxed Problems (e.g. 8 puzzle)
  - Geometric Limits (Euclidean/Manhattan)
  - Creativity ☺



Bi-directional A*

- If we have access to the goals:
  - Start another A* search with the start node as the goal node
  - Reverse the action transitions
  - Stop when two searches meet!

- Halves the state space complexity! (Still exponential)

Small Warning about Complexities

- You may be confused about Search having exponential complexity after learning about them in the Data Structures Course
  - Number of nodes |V|, number of edges |E|
  - Complexity of traversing a graph: O(|E|+|V|)
  - Complexity of finding the shortest path in a graph: O((|V|+|E|)log(|V|))

- In planning (AI), we look at the branching factor (b) and the depth (m). The problems we deal with induce very large state spaces which result in |E| and |V| to be exponential wrt b and m!

HW1

- Will be released soon!
- Graph search versions
- How to keep the path?
  - Either store the path to a state or store the parent
  - You need to know the action!
- You can solve all the problems now!

Beyond Vanilla A*

- Limited Space A* (aka Beam Search)
- Pre-compute costs for certain nodes and use a version of triangle equality
- Learning heuristics
- Online A*
- Dynamic Environment: D*

- There are others as well …