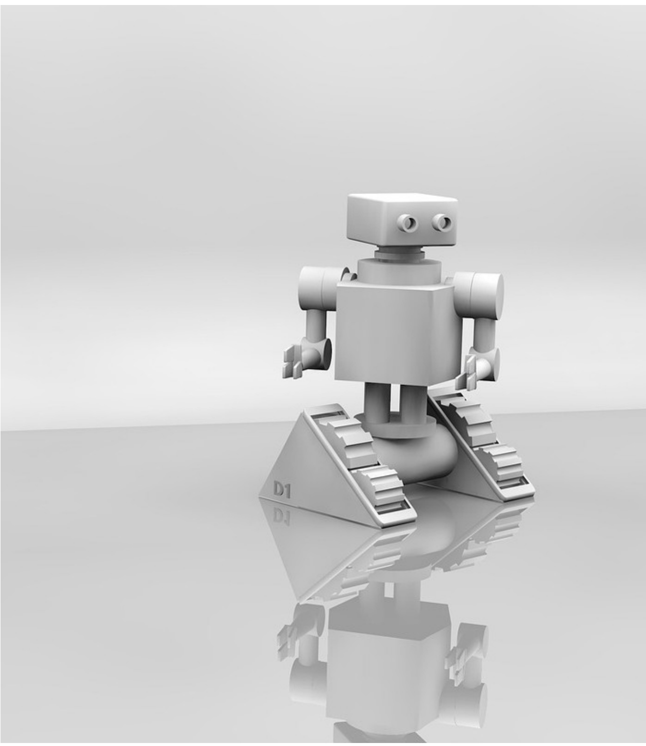


# CPSC 481
# Artificial Intelligence

Dr. Mira Kim
Mira.kim@fullerton.edu

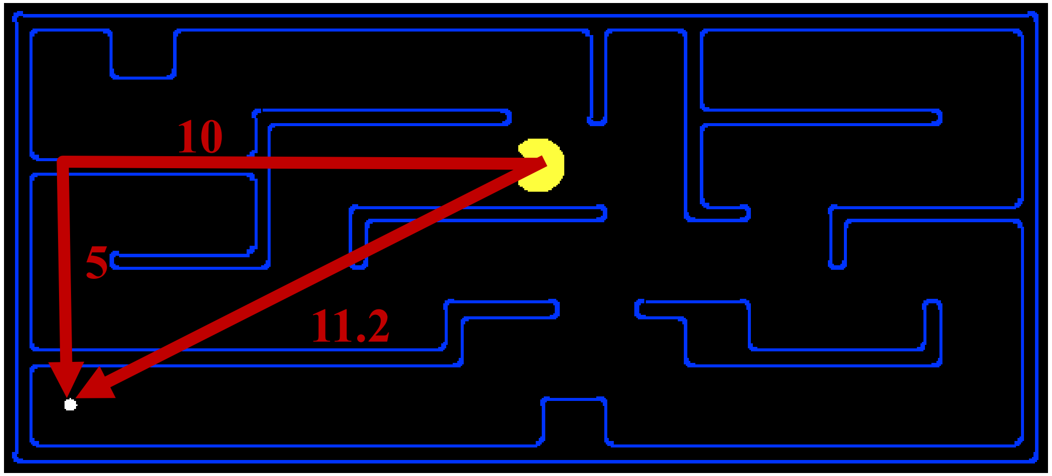
# What we will cover today

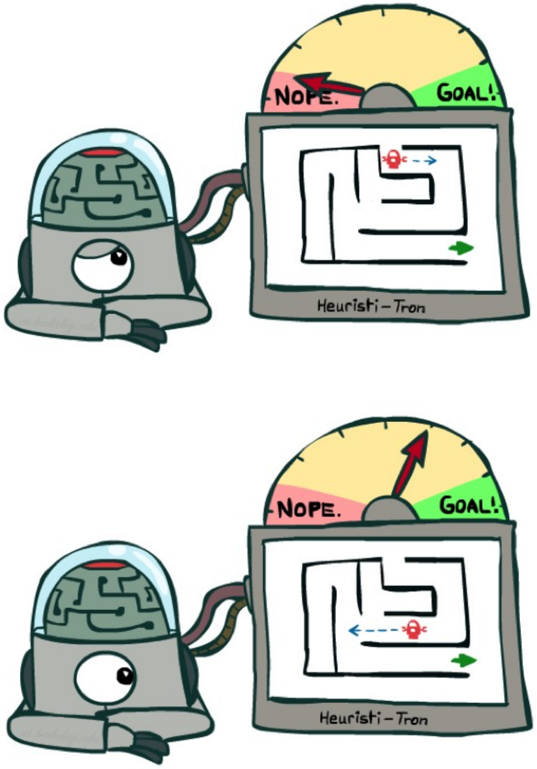
- Informed search
  - Heuristics
  - Greedy best-first search
  - A*

Textbook: Chapter 3.5

# Search Heuristics

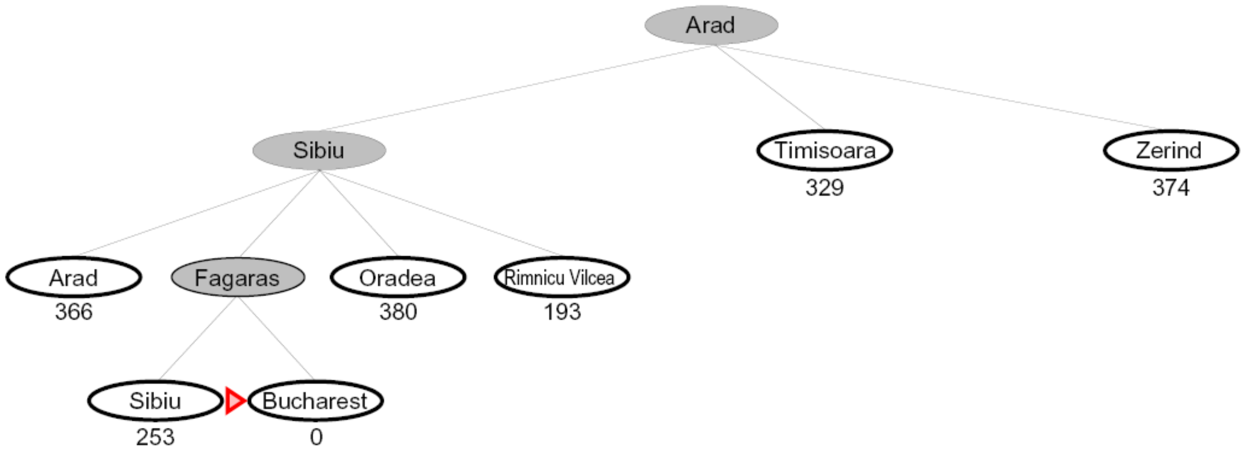

- A heuristic is:
  - A function that *estimates* how close a state is to a goal
  - Designed for a particular search problem



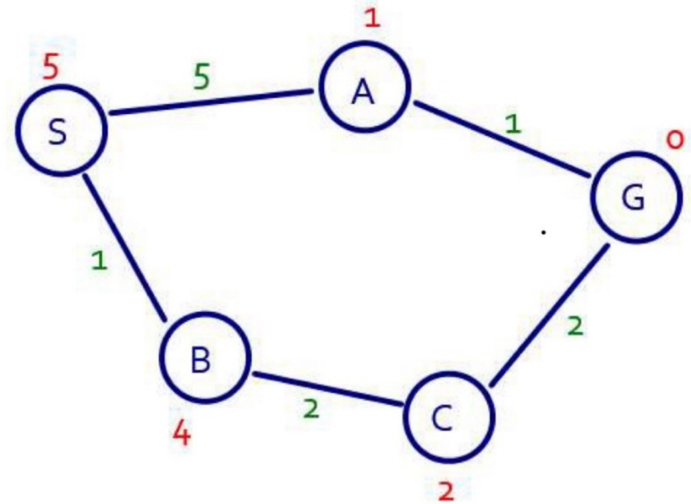
# Greedy Best-First Search

- Expand the node that seems closest according to heuristic
- Heuristic: estimate of distance to nearest goal for each state
- Also called “Pure heuristic search”

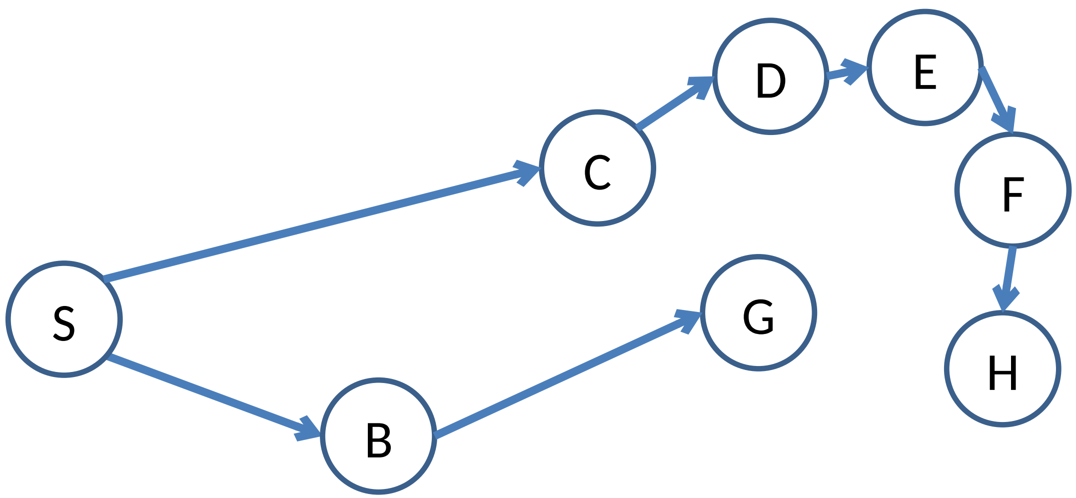


# Greedy Best-first search

- What can go wrong?
- Not optimal
- Can get stuck in dead ends
  – Not complete

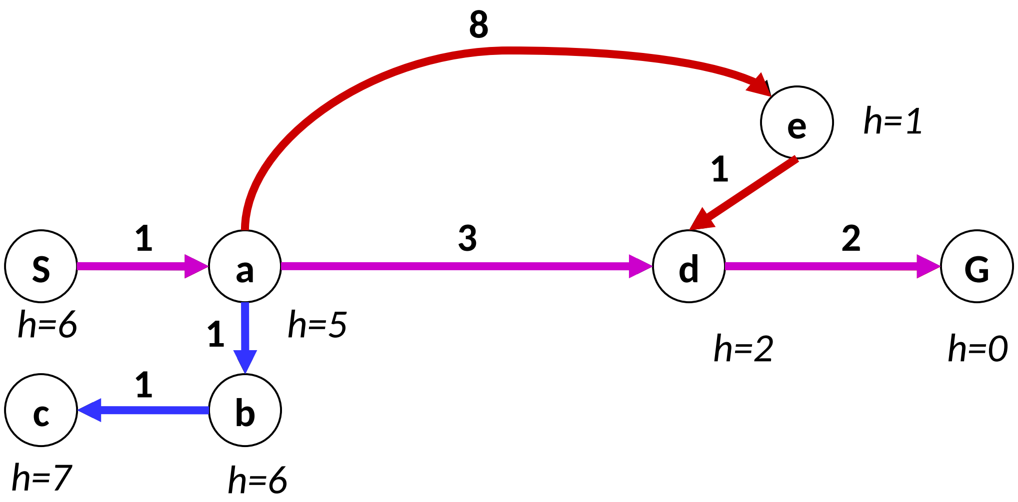


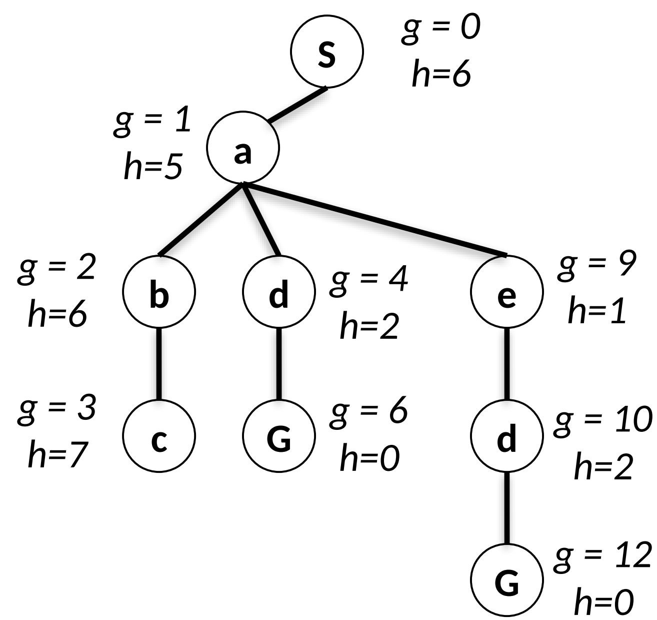

Heuristic cost (perfectly accurate!)
Actual cost



# A* Search –
# 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)
- A* is pronounced "A star"

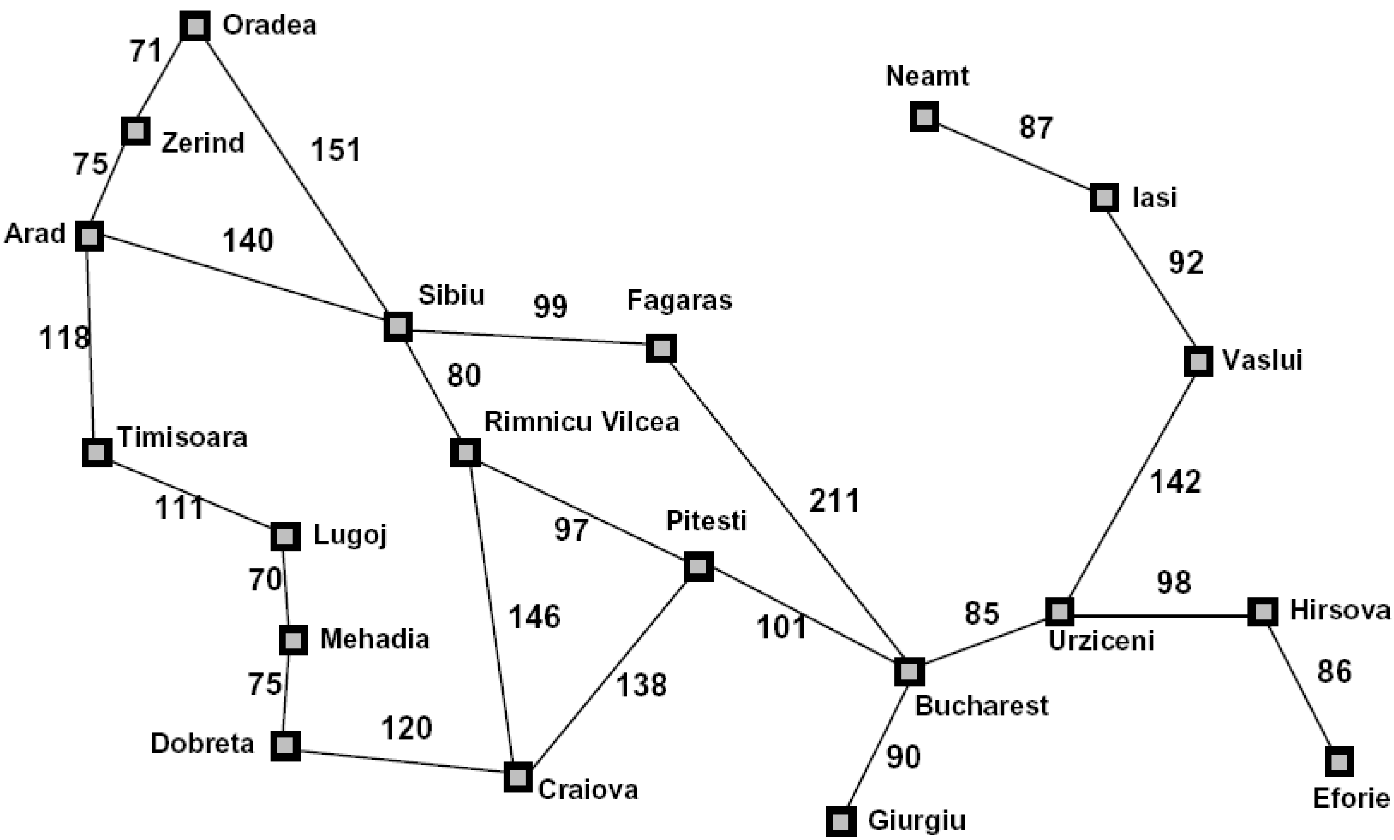
Example: Teg Grenager

# A* search

- Input: A problem
- Data structures:
  - Frontier (also called "open")
    - **Priority Queue, ordered by path cost + heuristic cost**
  - Explored (also called "closed")
    - Set, for efficiency

```
Initialize frontier with initial state
Initialize explored to empty
Loop do
   IF the frontier is empty RETURN FAILURE
   Choose lowest cost node from frontier and remove it
   IF lowest cost node is goal RETURN SUCCESS
   Add node to explored
   FOR every child node of node
     IF child not already on frontier or explored
       insert child node to frontier
     ELSE IF child is in frontier with higher path cost
       replace existing frontier node with child node
```

# Example: Heuristic Function

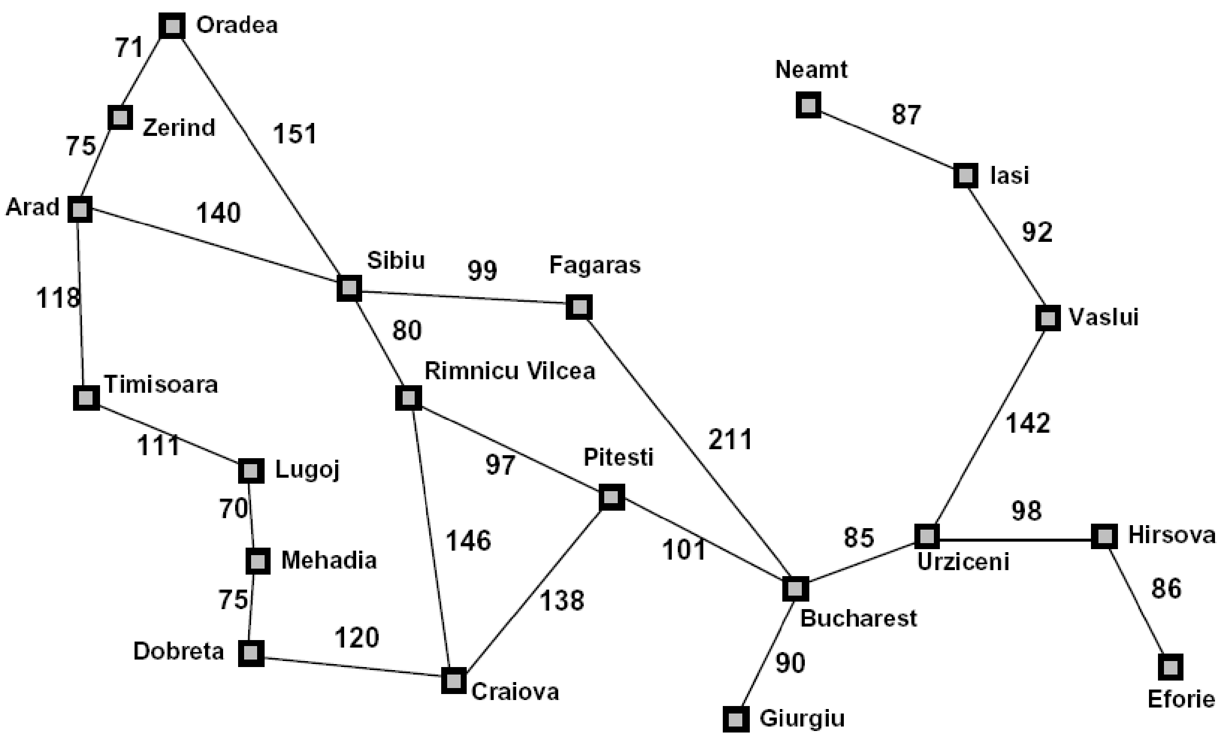



Straight−line distance to Bucharest

| | |
|---|---|
| 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(n)

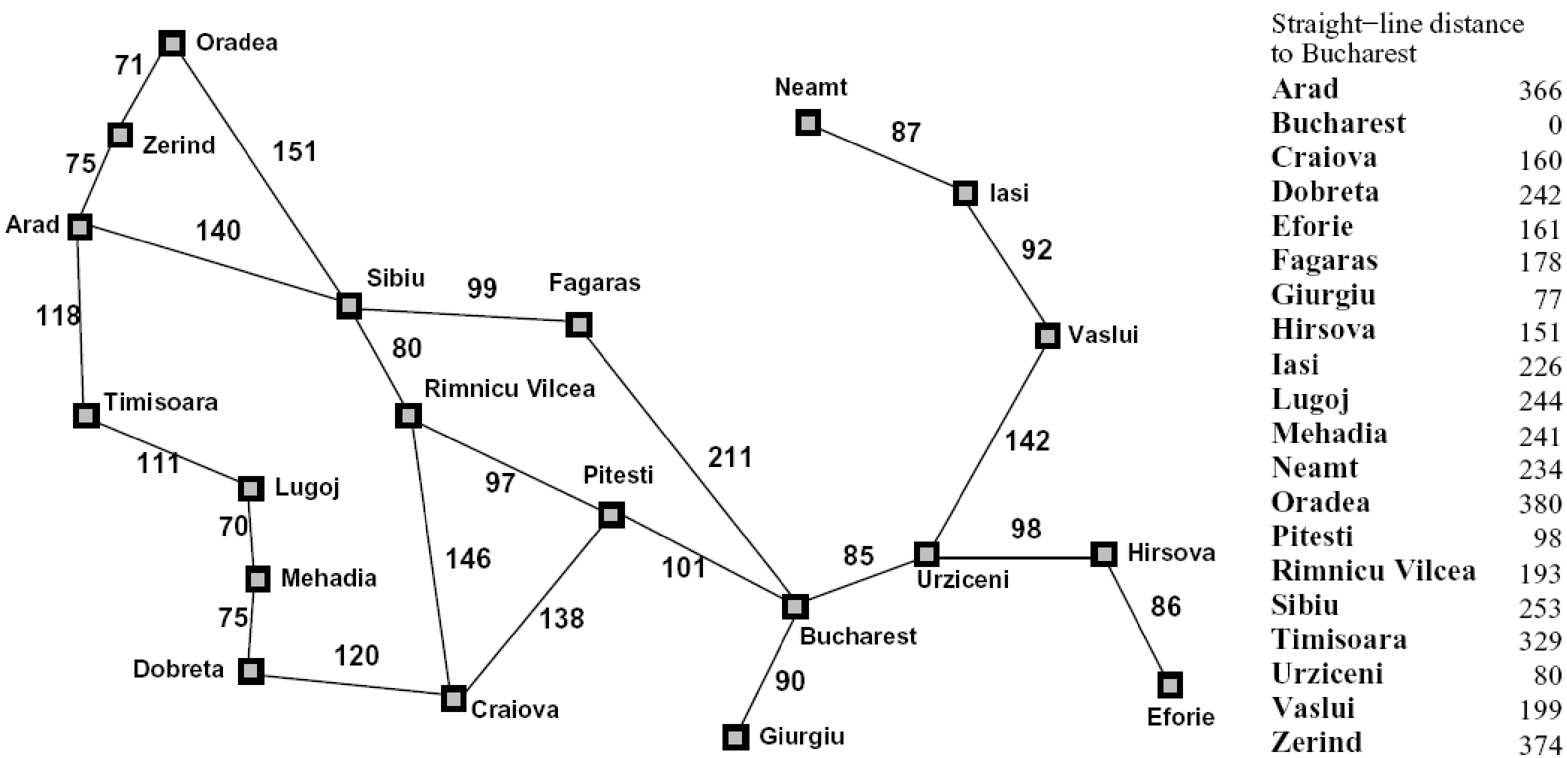
| Explored (set) | Frontier (priority queue) |
|---|---|
| A(0+366=366) | {} |
| | |
| | |



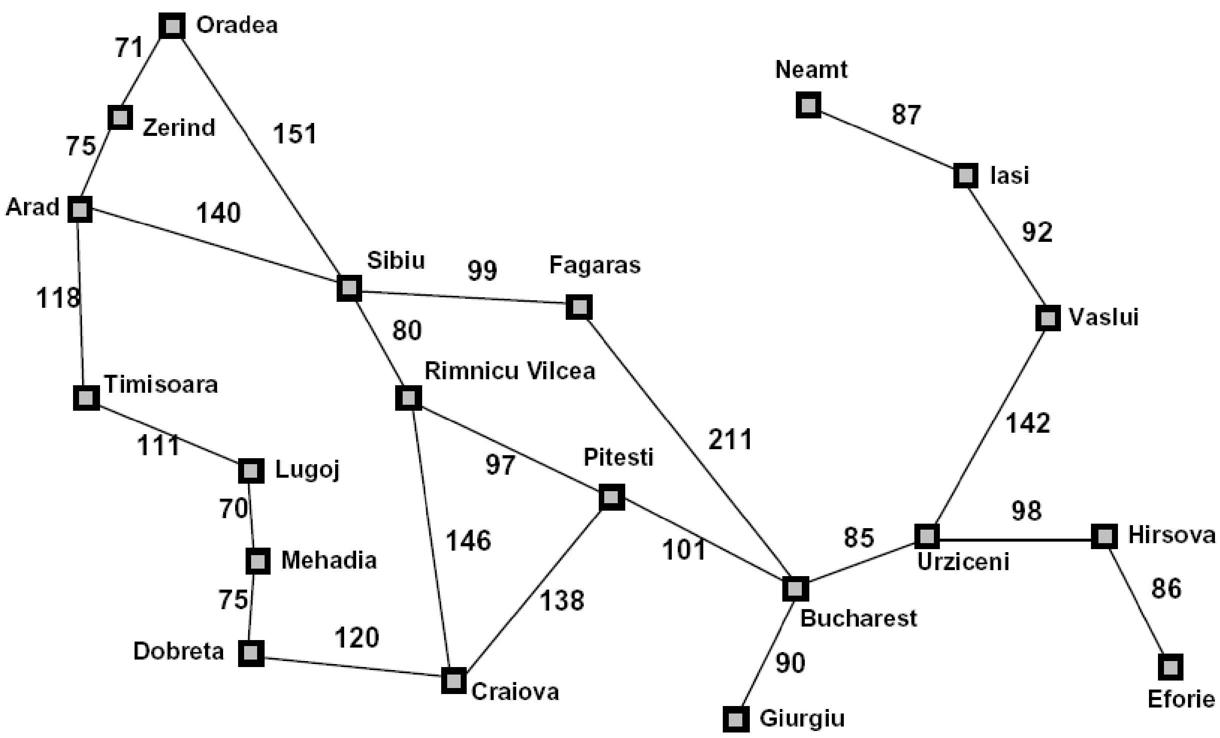

Straight−line distance to Bucharest

| | |
|---|---|
| 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 |

| Explored (set) | Frontier (priority queue) |
|---|---|
| {} | A(0+366=366) |
| {A} | Z(75+374=449), S(140+253=393), T(118+329=447) |
| | |



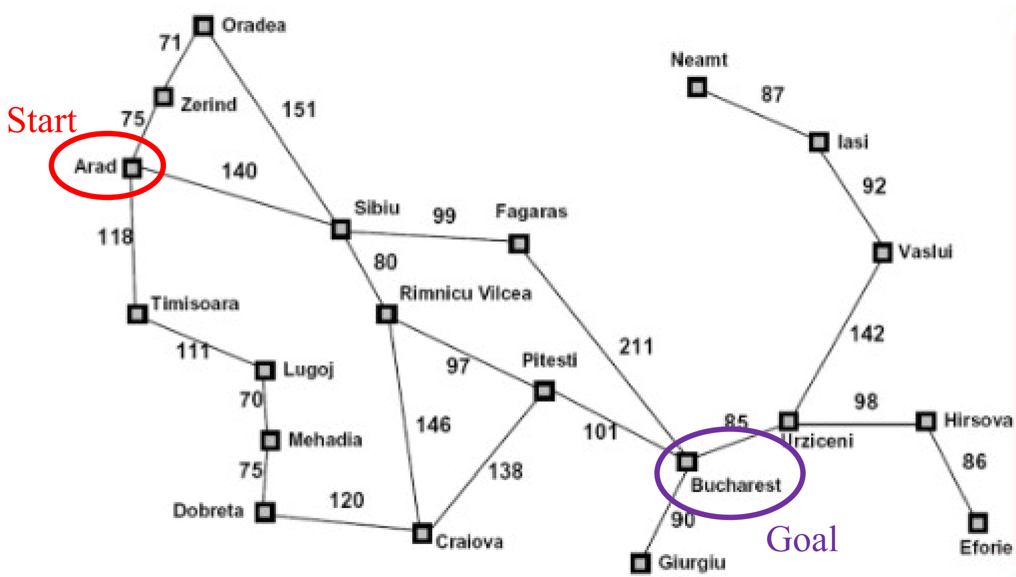
| Explored (set) | Frontier (priority queue) |
|---|---|
| {} | A(0+366=366) |
| {A} | Z(75+374=449), S(140+253=393), T(118+329=447) |
| {A, S} | Z(449), T(447), O (140+151+380 = 671), F (140+99+178 = 417), RV (140+80+193 =413) |




| 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 |

# In-class exercise

- Work on the rest of the iterations for going from Arad to Bucharest
  - Show frontier (priority queue) and explored node



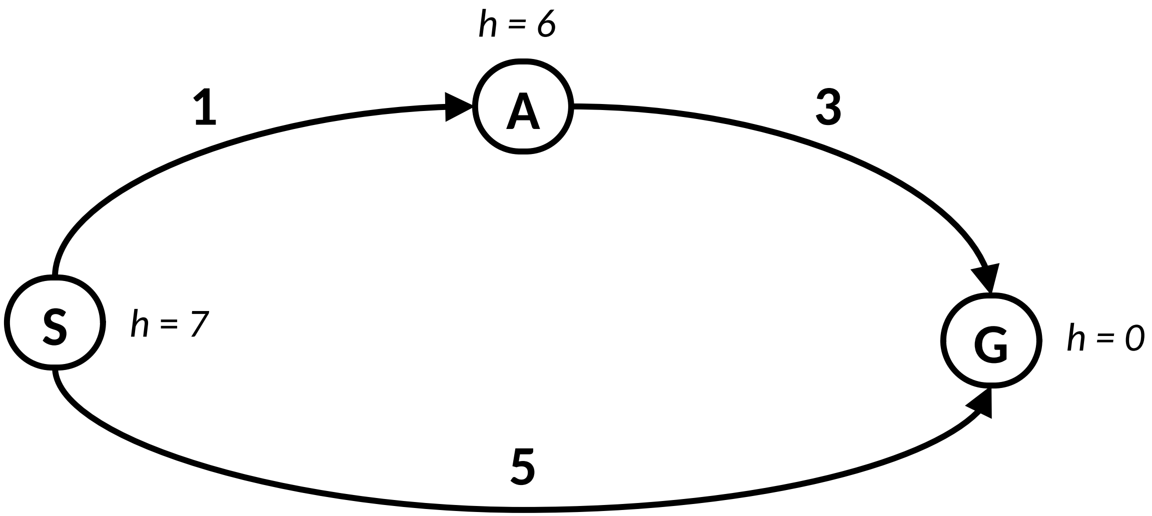

| Straight−line distance to Bucharest | |
|---|---|
| Arad | 366 |
| Bucharest | 0 |
| Craiova | 160 |
| Dobreta | 242 |
| Eforie | 161 |
| Fagaras | 178 |
| Giurgiu | 77 |
| Hirsova | 151 |
| Iasi | 226 |
| Lugoj | 244 |
| Mehadia | 241 |
| Neamt | 234 |
| Oradea | 380 |
| Pitesti | 98 |
| Rimnicu Vilcea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

h(n)

A* SEARCH

# PROPERTIES OF A*

# Is A* Optimal?




- What went wrong?
- Estimated goal cost > actual goal cost
- We need estimates to be less than actual costs!

# Conditions for optimality of A*

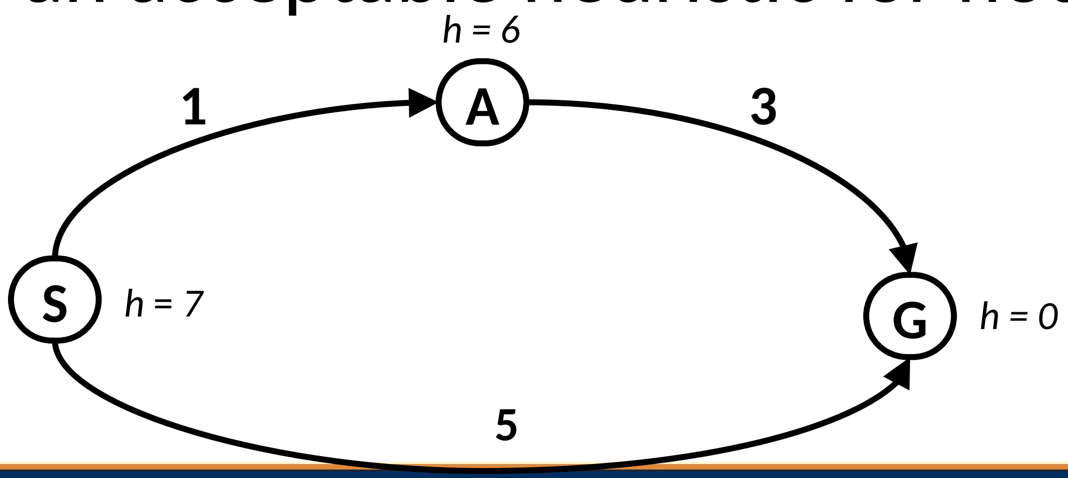
1. Heuristic must be *admissible*
   – Must never overestimate true cost
   – Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs

# Admissible Heuristics

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

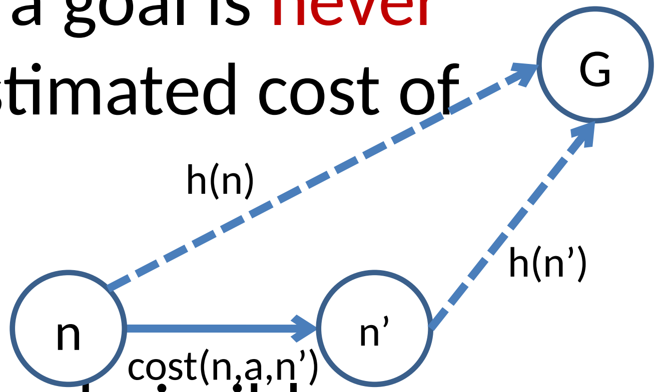
  where   is the true cost to nearest goal

- What is an acceptable heuristic for node A?



# Conditions for optimality of A*

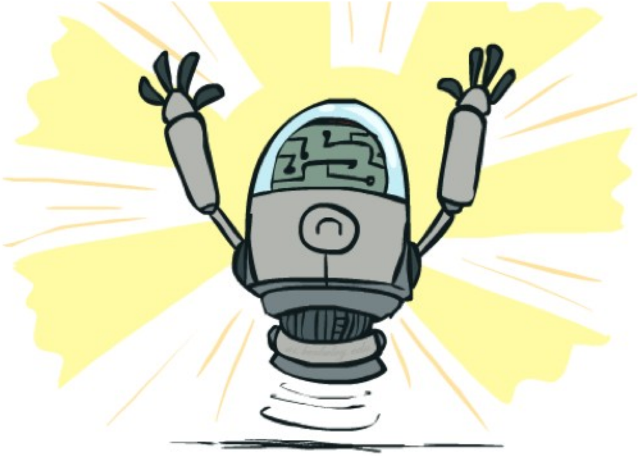
2. Heuristic must be *consistent* (also called *monotonic*)
   - The estimated cost of getting to a goal is never more than cost of an action + estimated cost of next state




   - Every consistent heuristic is also admissible
   - In general, most natural admissible heuristics tend to be consistent

# Optimality of A* Search

- Complete
  - Terminates even on infinite state spaces if a reachable goal exists
- Optimal
  - Always finds the closest goal
- Optimally efficient
  - No other optimal algorithm with the same heuristics expands fewer nodes
- Note: Uniform cost search is a special case of A* (h = 0)

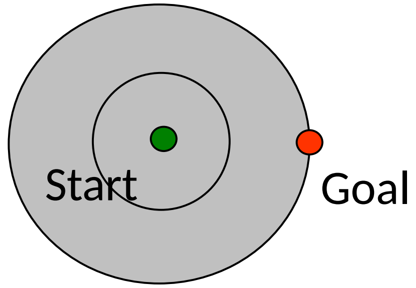


# Complexity of A* Search

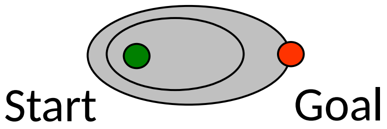
- Time complexity:
  - Unfortunately, number of states still exponential in the length of the solution
  - Number of states depends on the “error” in the heuristic
    - Error =
- Space complexity:
  - Keeps all generated nodes in memory, so exponential

# UCS vs A* Contours

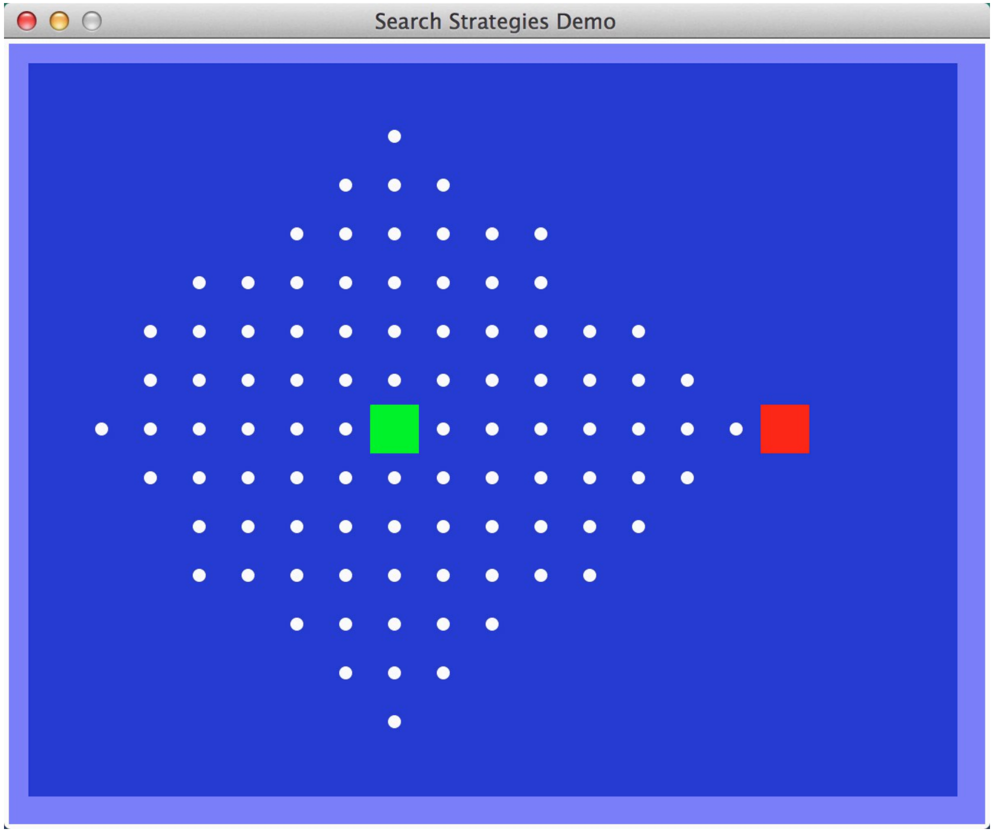
- Uniform-cost expands equally in all "directions"



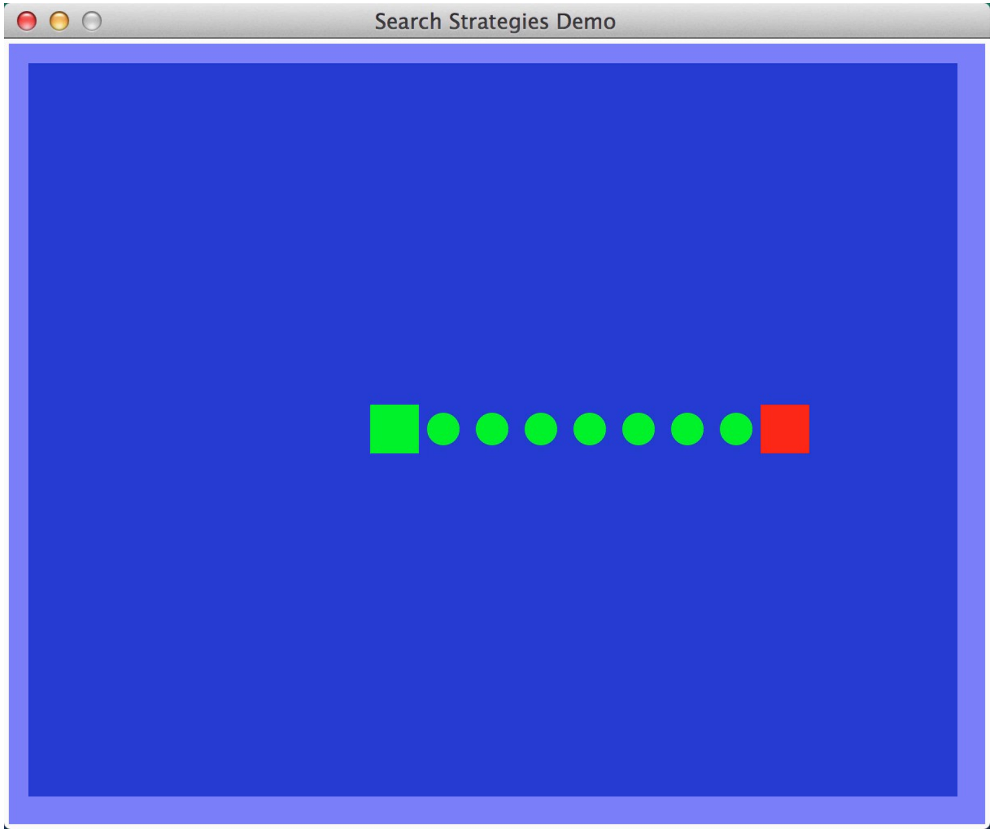

- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



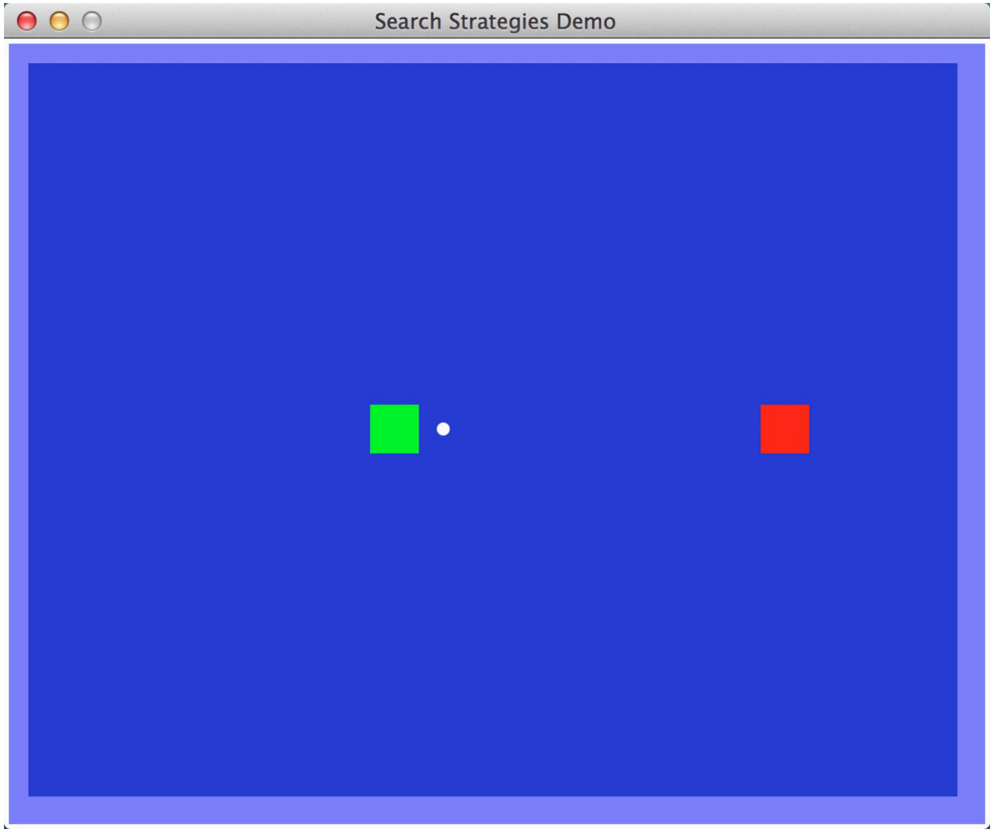
# Video of Demo Contours (Empty) -- UCS



# Video of Demo Contours (Empty) -- Greedy



# Video of Demo Contours (Empty) – A*



# A* Applications

- Video games
- Path finding
- Robot motion planning
- Resource allocation

A* SEARCH

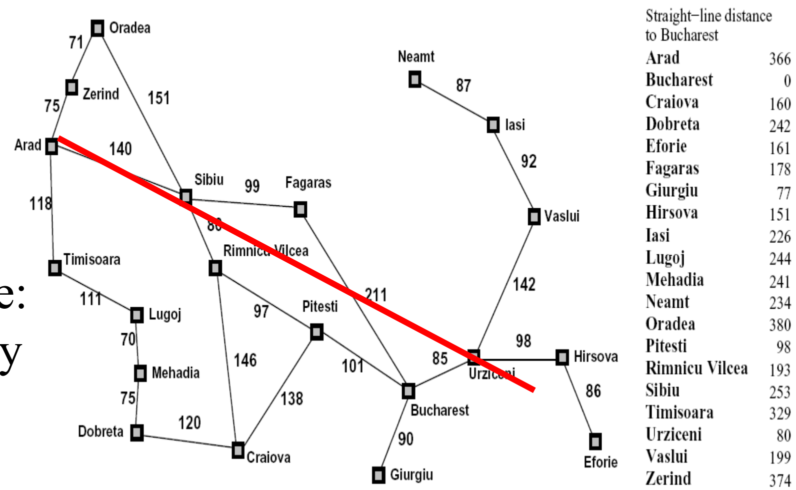
# CREATING HEURISTICS

# Creating Admissible Heuristics

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

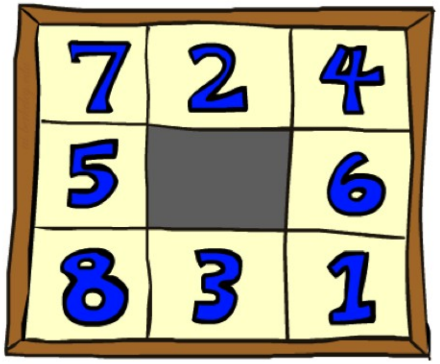
- Often, admissible heuristics are solutions to *relaxed problems*, where new actions are available

Euclidean distance:
assume you can fly

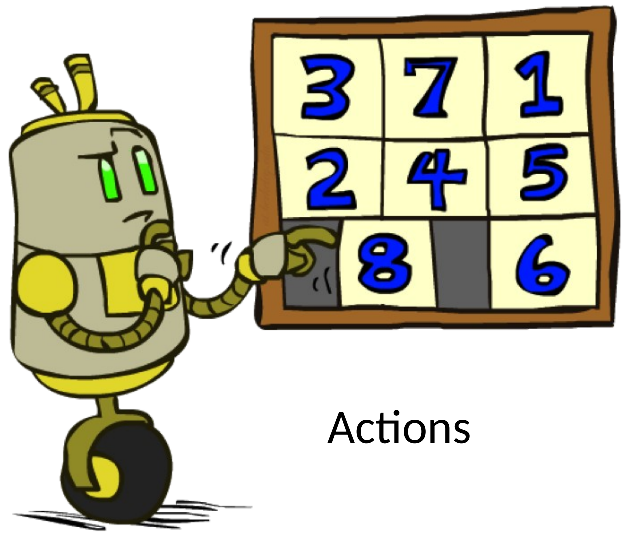



- Inadmissible heuristics are often useful too

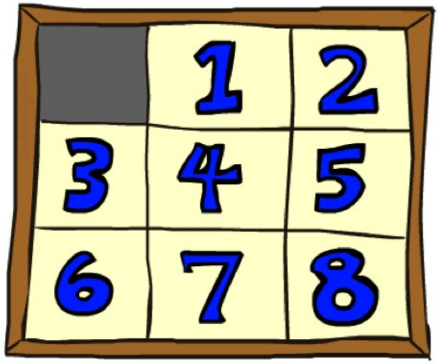
# Example: 8 Puzzle




Start State




Actions

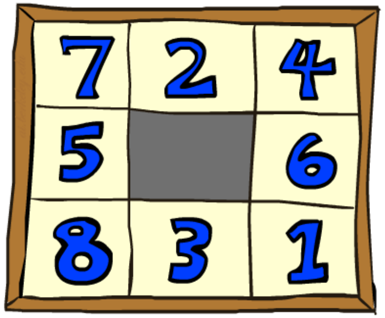



Goal State

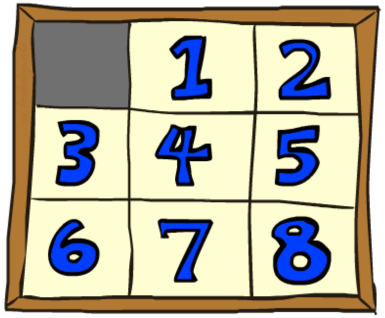
- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

# 8 Puzzle I

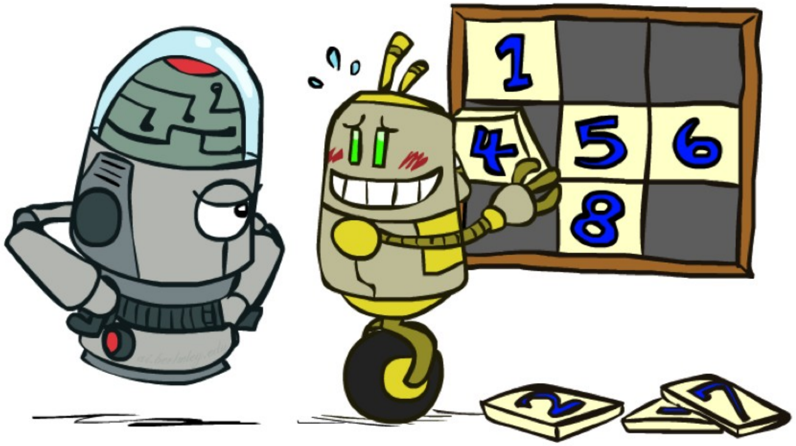
- Heuristic: Number of tiles misplaced
- h(start) = 8
- This is a *relaxed-problem* heuristic
- Why is it admissible?



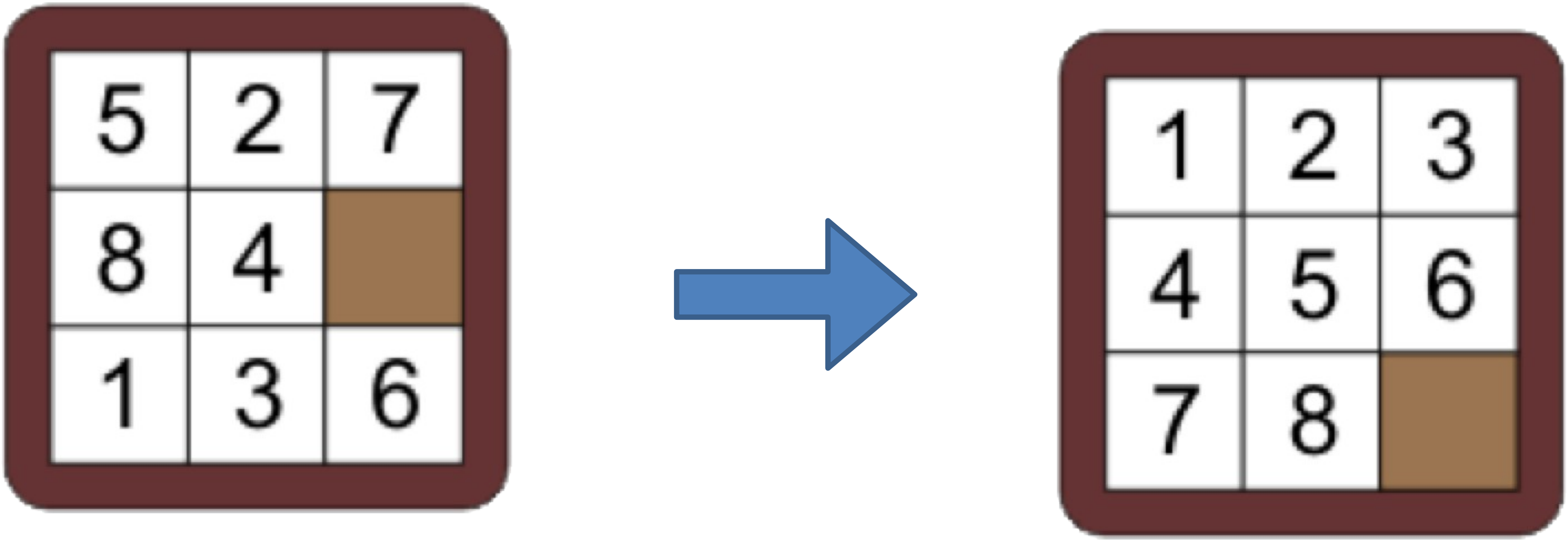
Start State



Goal State



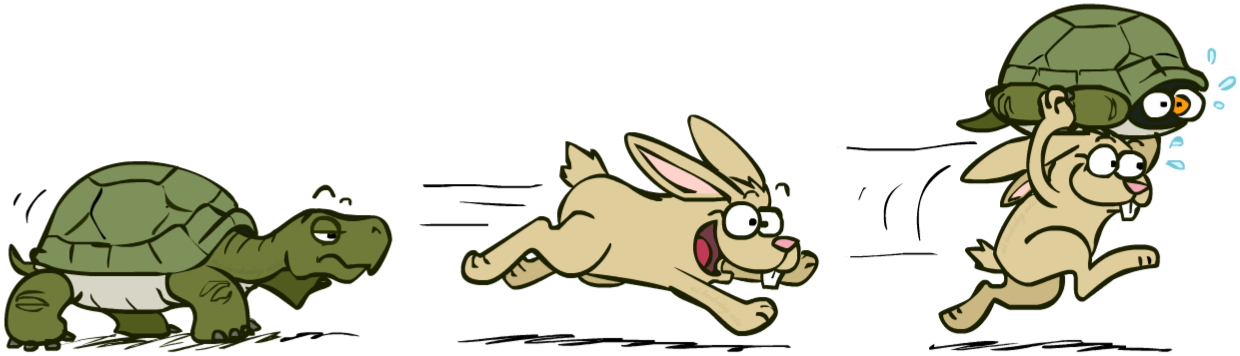
# Sliding tile puzzle: misplaced tiles heuristic




h(n) = ?

# 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



# Acknowledgement

- https://inst.eecs.berkeley.edu/~cs188/su20/

# References


- Russel and Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Prentice Hall, 2010.