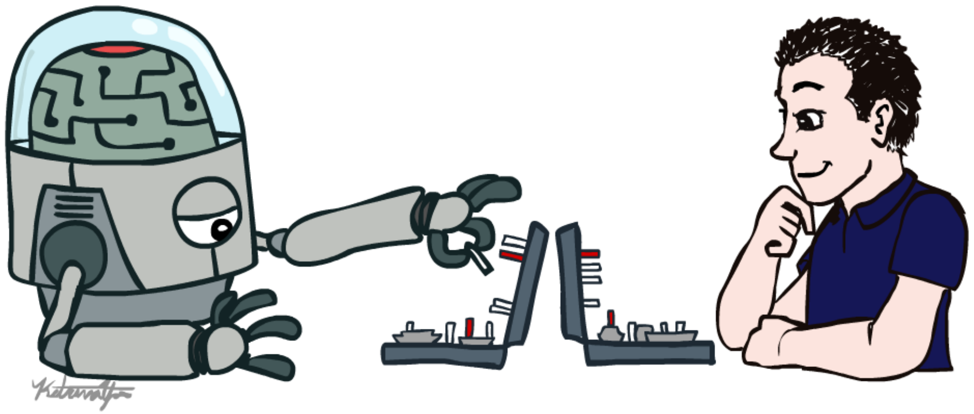
# Lecture 03



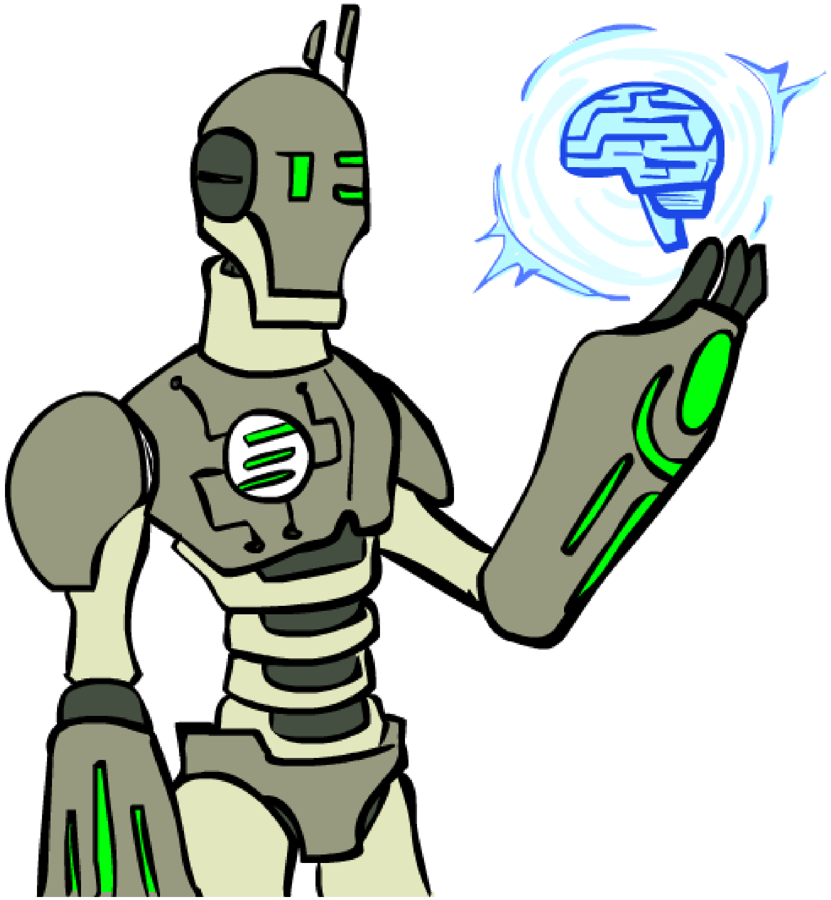
**Ashis Kumar Chanda**
chanda@rowan.edu

# Search Algorithms

- Blind search – BFS, DFS, uniform cost
  - No concept of the “right direction”
  - can only recognize goal once it’s achieved

- Heuristic search – we have rough idea of how good various states are, and use this knowledge to guide our search

# Today

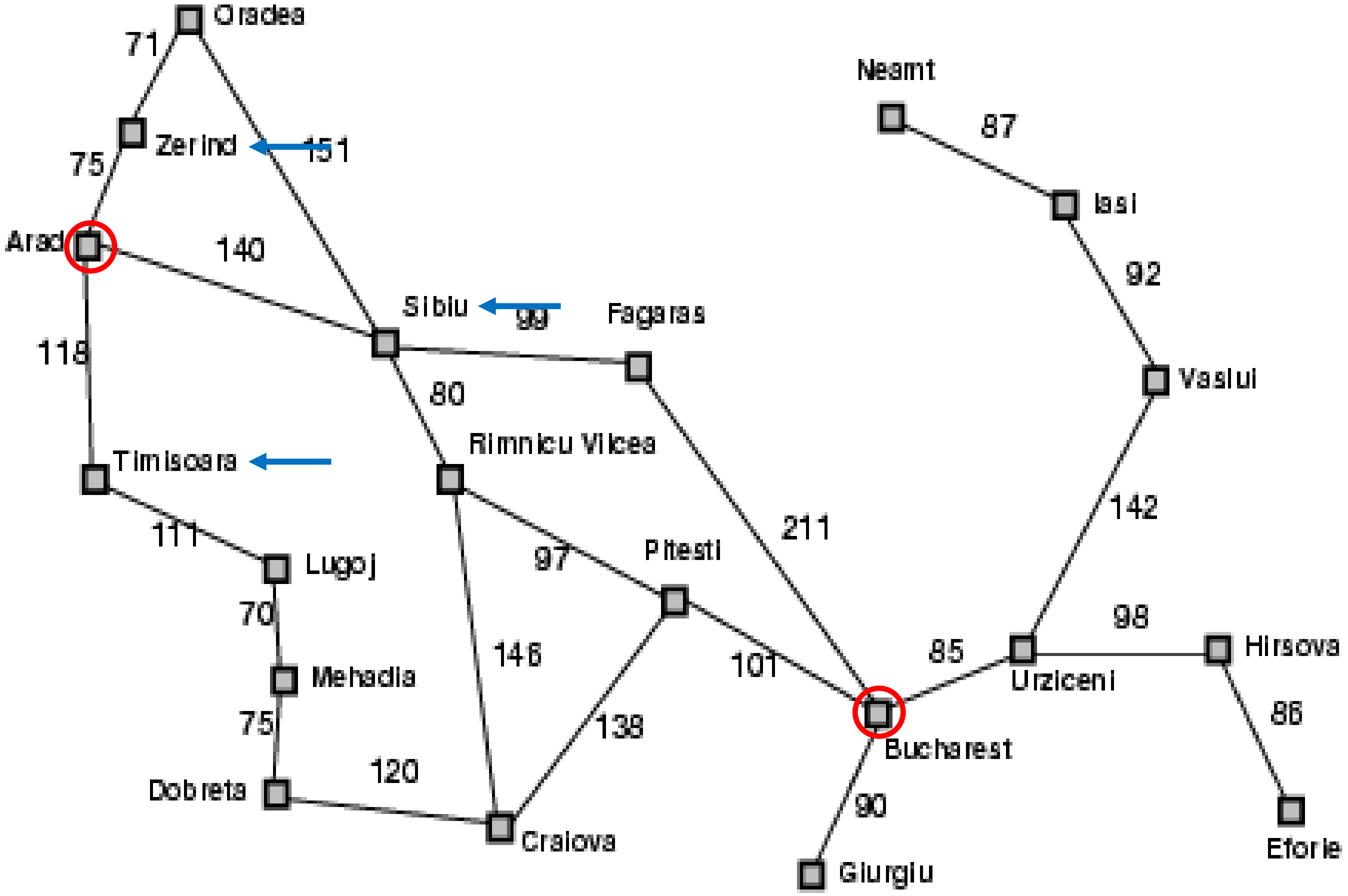


- Best-first search
- Greedy best-first search
- $A^*$ search

# Best-first search

- Idea: use an evaluation function *f(n)* for each node
  - Estimate of "desirability"
  - Expand most desirable unexpanded node
  - Use domain-specific hints about the location of goals
- Implementation:
  - Order the nodes in fringe in decreasing order of desirability

# Romania with step costs in km




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 |

Showing straight line distances to Bucharest

# Greedy best-first search

- Greedy best first search is a form of best-first search that expands first the node with the lowest **h(n)** value (node that appears to be closest to the goal).
- Evaluation function $f(n) = h(n)$ (heuristic)

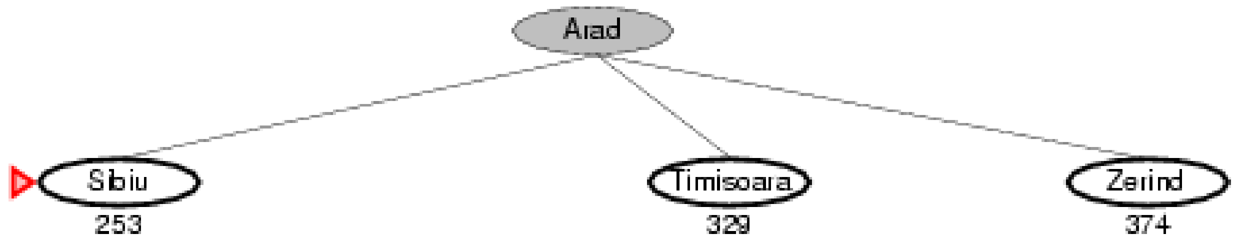
  = estimate of cost from $n$ to *goal*

Example:

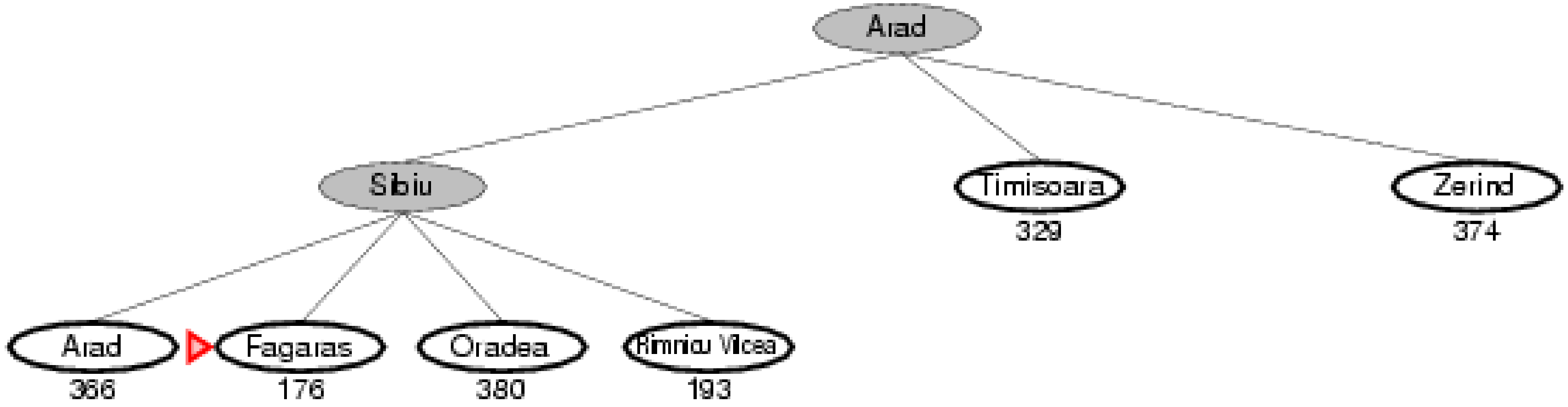
$h_{SLD}(n)$ = straight-line distance from $n$ to Bucharest

# Greedy best-first search example

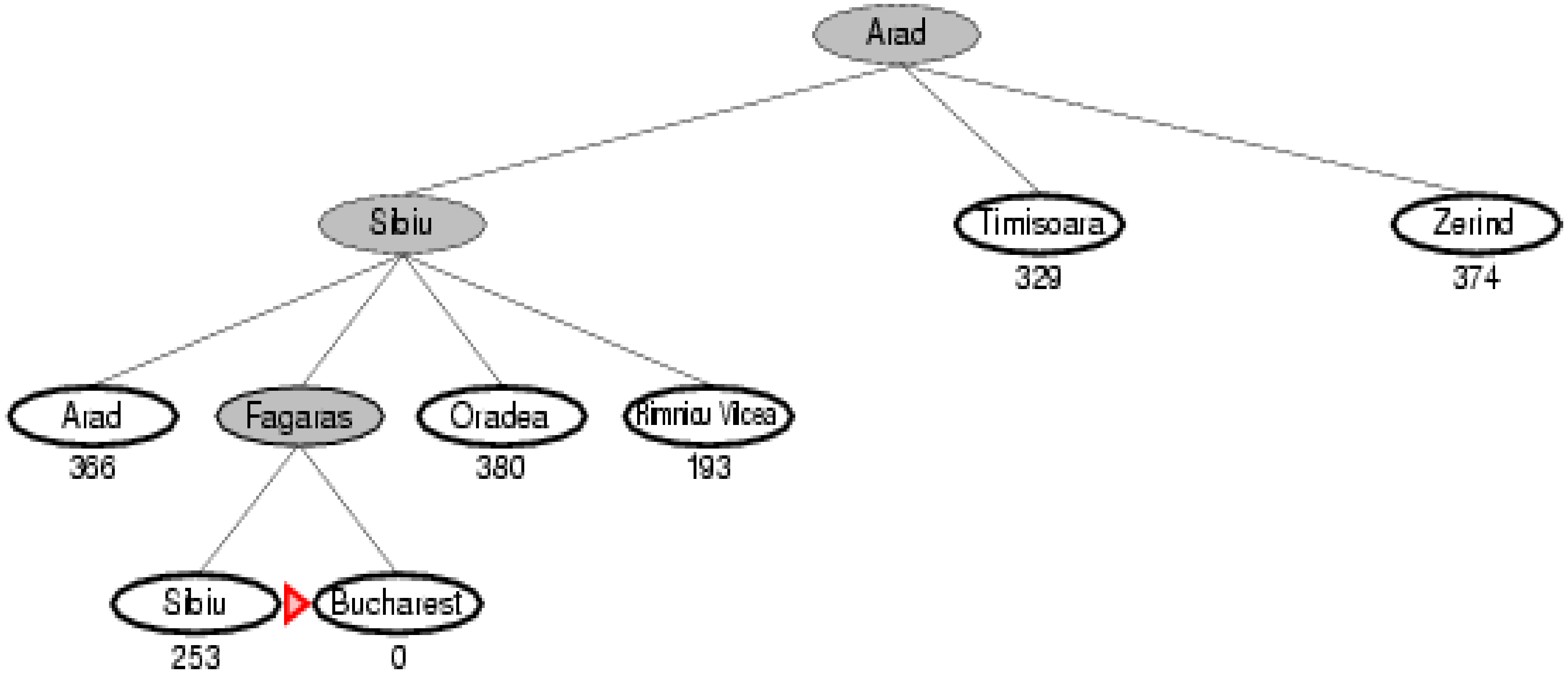
# Greedy best-first search example



# Greedy best-first search example

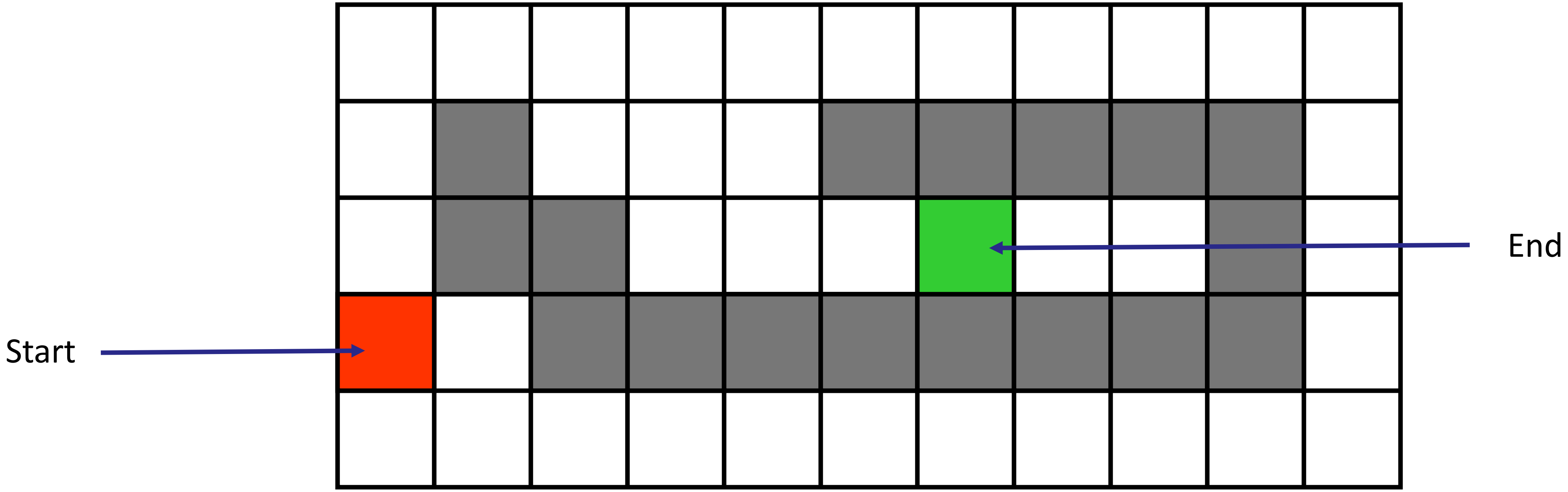


# Greedy best-first search example



# Robot Navigation

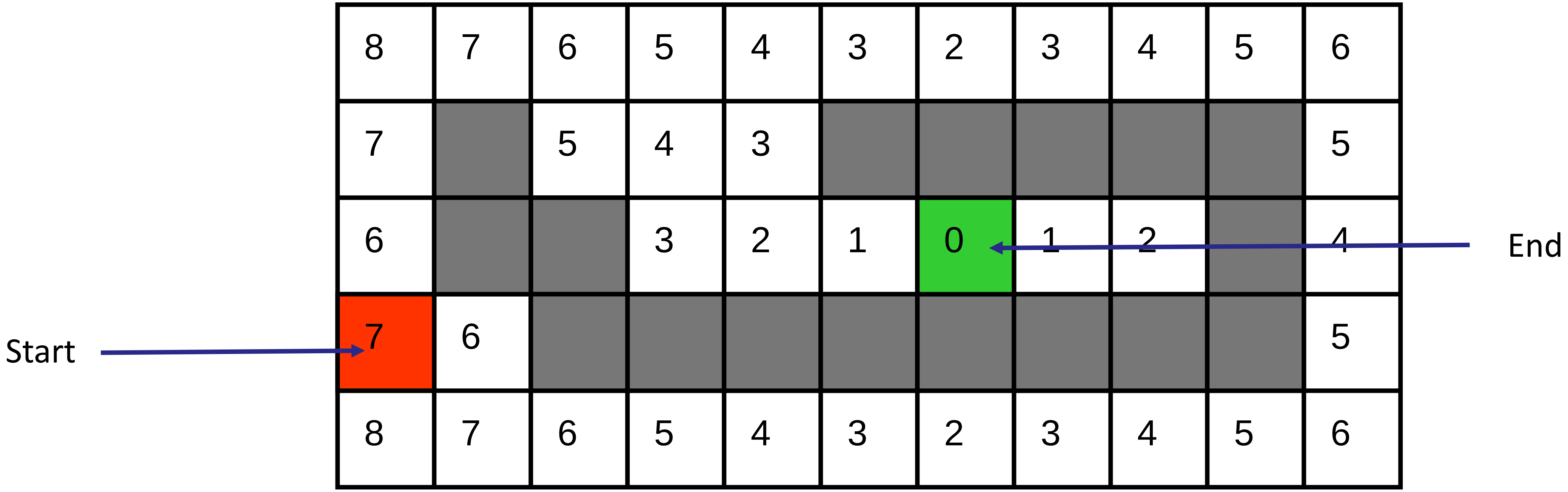
We can consider a 2D Grid having several obstacles and we start from a source cell (colored red below) to reach towards a goal cell (colored green below).



# Robot Navigation

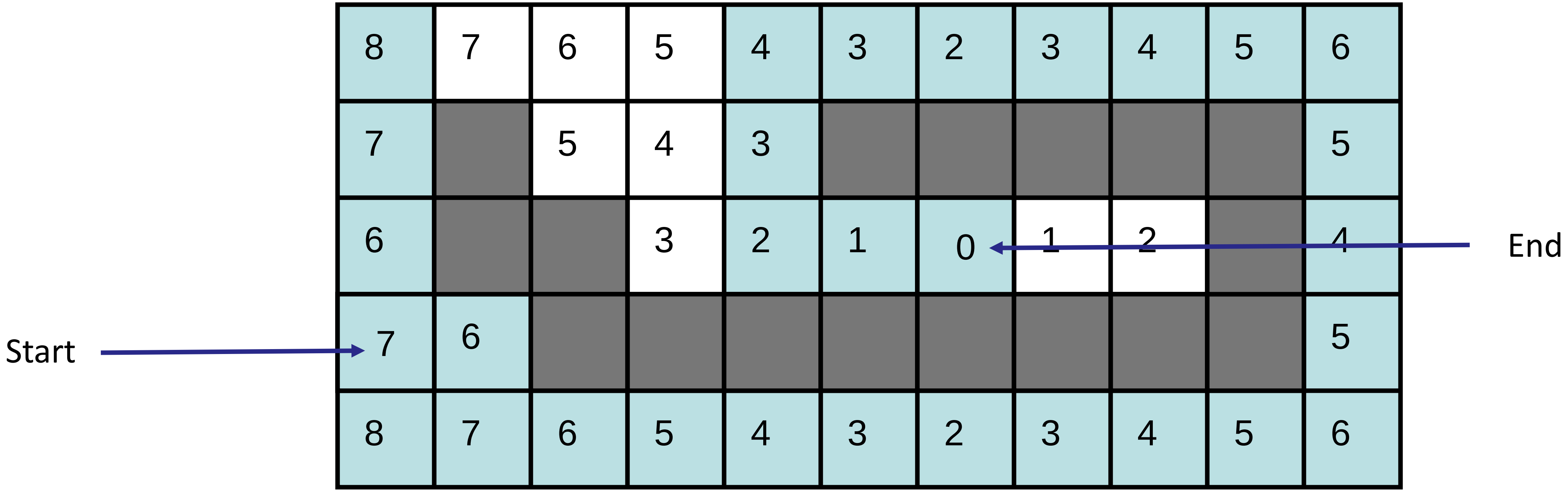
| | | | | | | |
|---|---|---|---|---|---|---|
| | 5 | 4 | 3 | 4 | 5 | |
| 5 | 4 | 3 | 2 | 3 | 4 | 5 |
| 4 | 3 | 2 | 1 | 2 | 3 | 4 |
| 3 | 2 | 1 | 0 | 1 | 2 | 3 |
| 4 | 3 | 2 | 1 | 2 | 3 | 4 |
| 5 | 4 | 3 | 2 | 3 | 4 | 5 |
| | 5 | 4 | 3 | 4 | 5 | |

f(N) = h(N), with h(N) = Manhattan distance to the goal



# Robot Navigation

f(N) = h(N), with h(N) = Manhattan distance to the goal



# Properties of greedy best-first search

- **Complete**? If we eliminate endless loops, yes
- **Time**? $O(b^m)$, but a good heuristic can give dramatic improvement
- **Space**? $O(b^m)$ - keeps all nodes in memory
- **Optimal**? No
    - *b:* maximum branching factor of the search tree
    - *m*: maximum depth of the state space (may be ∞)

# More informed search

- We kept looking at nodes **closer and closer to the goal**, but were accumulating costs as we got further from the initial state.
- Our goal is not to **minimize the distance from the current head of our path to the goal**, we want to minimize the ***overall*** length of the path to the goal!
- Let **g(n)** be the cost of the best path found so far between the initial node and n

- $f(n) = g(n) + h(n)$

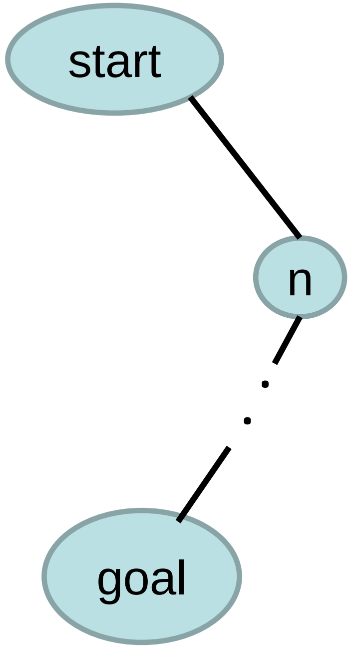
# A* search (Or, A star search)

- Idea: avoid expanding paths that are already expensive.

- Evaluation function $f(n) = g(n) + h(n)$

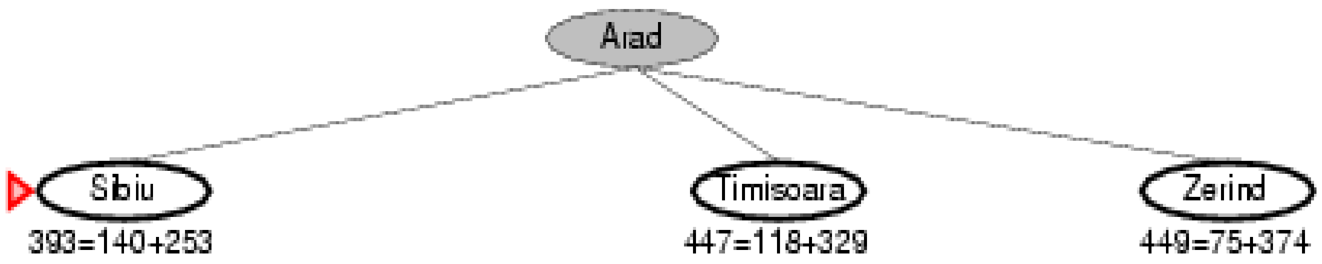
- $g(n)$ = cost so far to reach $n$
- $h(n)$ = estimated cost from $n$ to goal (admissible heuristic)
- $f(n)$ = estimated total cost of path through $n$ to goal

- Then, best-first search with this evaluation function is called A* search

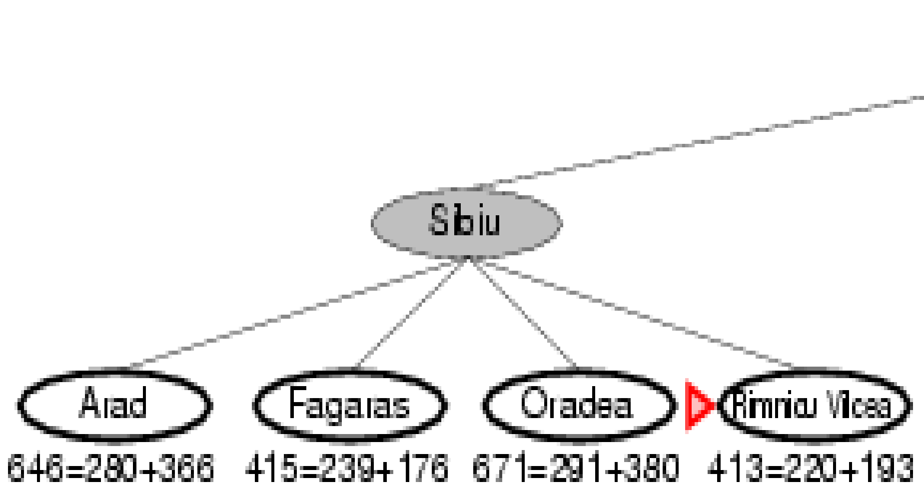


# $A^*$ search example

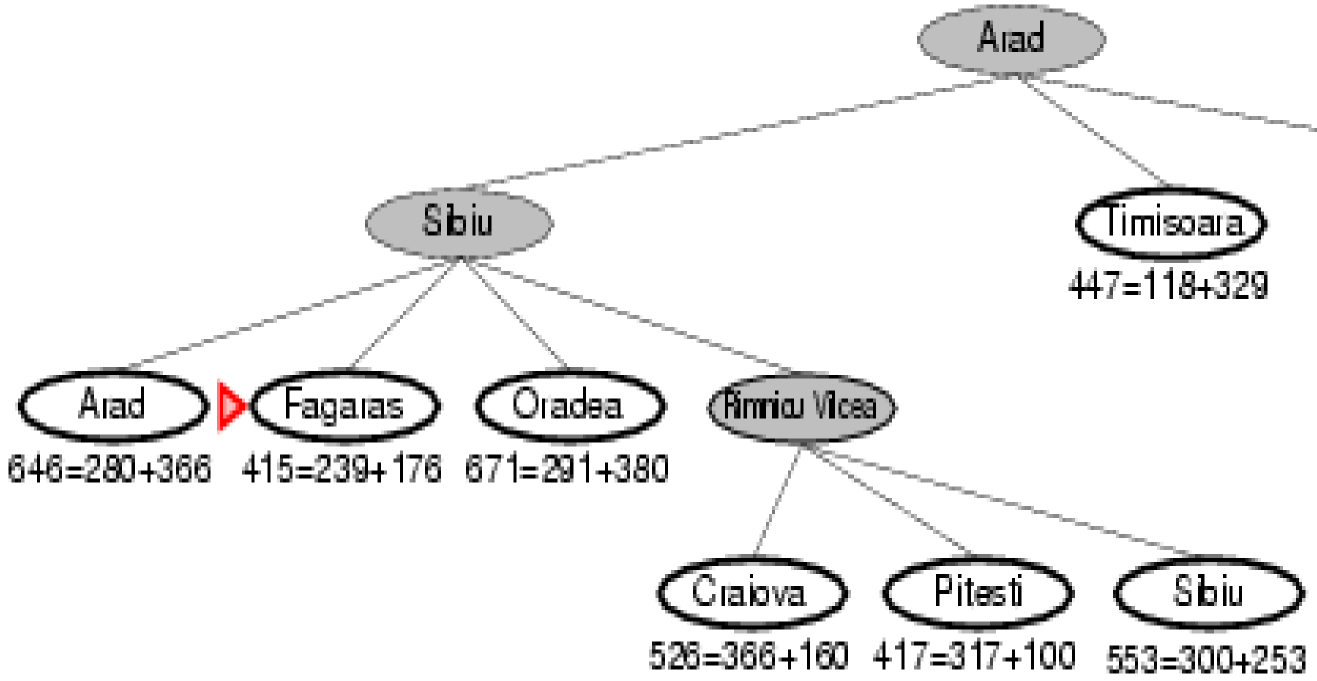
# A* search example



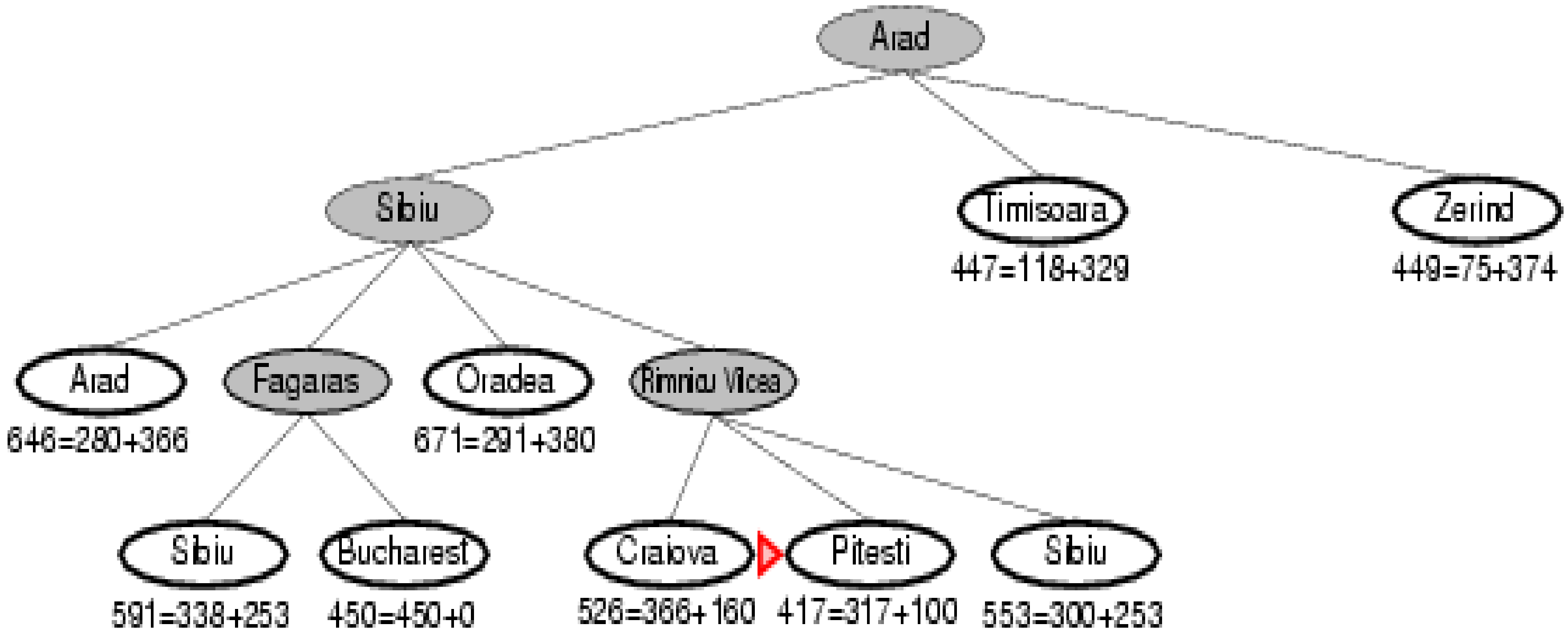
# A* search example



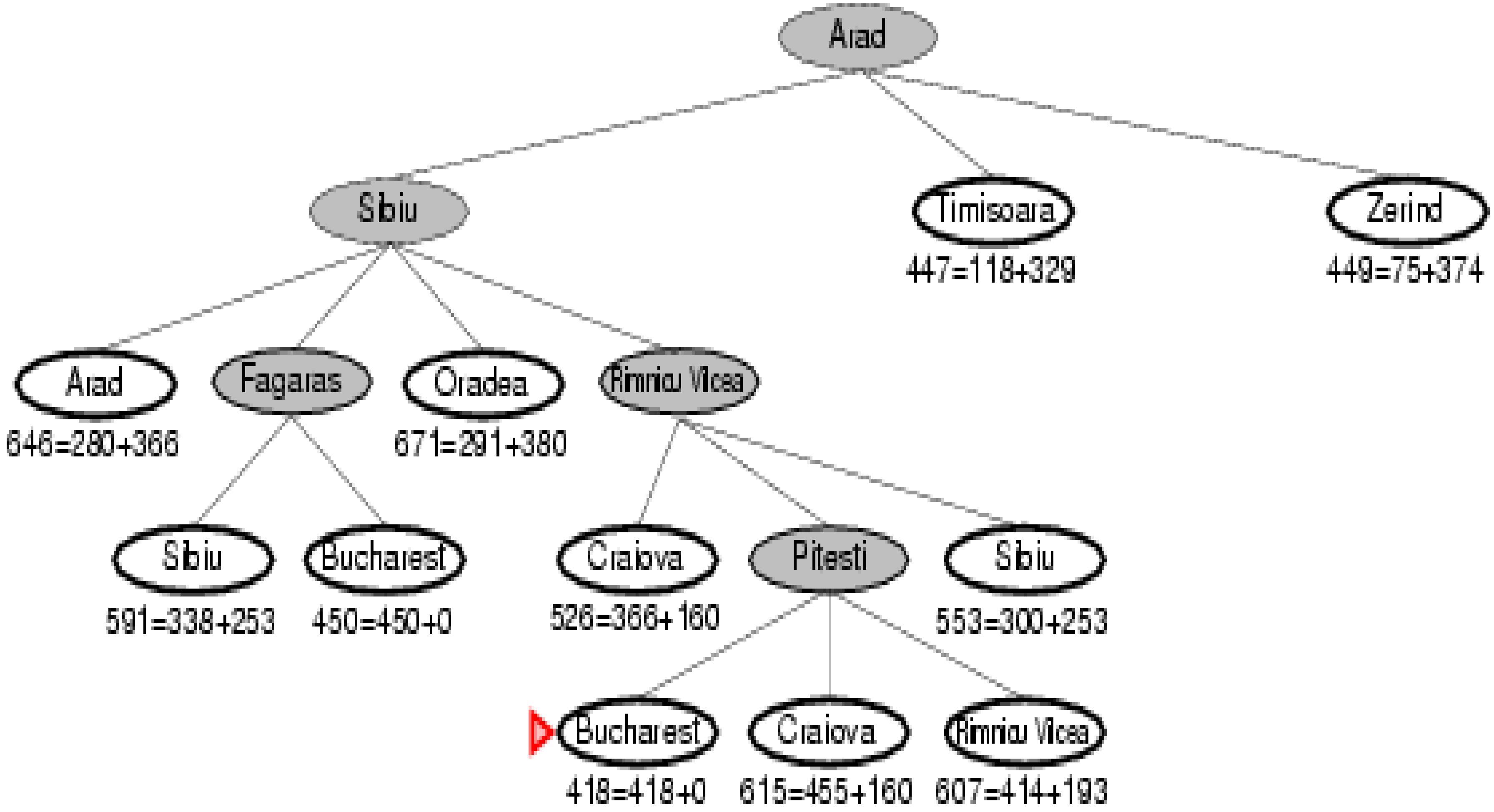
# A* search example



# A* search example



# A* search example



# Robot Navigation

| | | | | | | |
|---|---|---|---|---|---|---|
| | 5 | 4 | 3 | 4 | 5 | |
| 5 | 4 | 3 | 2 | 3 | 4 | 5 |
| 4 | 3 | 2 | 1 | 2 | 3 | 4 |
| 3 | 2 | 1 | 0 | 1 | 2 | 3 |
| 4 | 3 | 2 | 1 | 2 | 3 | 4 |
| 5 | 4 | 3 | 2 | 3 | 4 | 5 |
| | 5 | 4 | 3 | 4 | 5 | |

What should be f(n) for the robot navigation problem?

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 3 | 4 | 5 | 6 |
| 7 | | 5 | 4 | 3 | | | | | | 5 |
| 6 | | | 3 | 2 | 1 | 0 | 1 | 2 | | 4 |
| 7 | 6 | | | | | | | | | 5 |
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 3 | 4 | 5 | 6 |

End

Start

# Robot Navigation

f(N) = g(N)+h(N), with h(N) = Manhattan distance to goal

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 8+3 | 7+4 | 6+3 | 5+6 | 4+7 | 3+8 | 2+9 | 3+10 | 4 | 5 | 6 |
| 7+2 | | 5+6 | 4+7 | 3+8 | | | | | | 5 |
| 6+1 | | | 3 | 2+9 | 1+10 | 0+11 | 1 | 2 | | 4 |
| 7+0 | 6+1 | | | | | | | | | 5 |
| 8+1 | 7+2 | 6+3 | 5+4 | 4+5 | 3+6 | 2+7 | 3+8 | 4 | 5 | 6 |

# Completeness and Optimality of A*

- **Claim 1**: If there is a path from the initial to a goal node, A* (with no removal of repeated states) terminates by finding the best path, hence is:
  - complete
  - optimal
- requirements:
  - Each node has a finite number of successors

# Completeness of A*

- Theorem: If there is a finite path from the initial state to a goal node, A* will find it.

# Optimality of A*

- Theorem: If h(n) is admissible, then A* is optimal.

- Admissible: An admissible heuristic is one that never overestimates the cost to reach a goal.

# Admissible Heuristic

- Let $h^*(n)$ be the true cost of the optimal path from n to a goal node

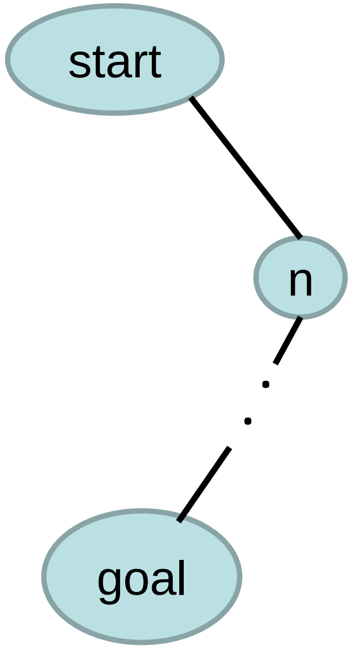
- Heuristic $h(n)$ is admissible if:

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

- An admissible heuristic is always optimistic.

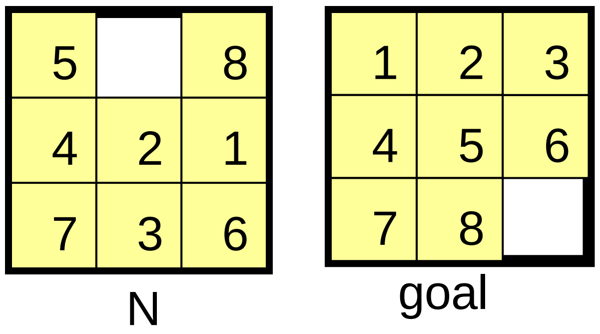
# Proof of Optimality of A*

- Suppose, the optimal path has cost C*
- But the algorithm returns a path with cost C , and C>C*
- g*(n): the cost of optimal path from start to n
- h*(n): the cost of optimal path from n to goal
  - C > C*
  - f(n) > C*
  - We know that f(n) = g(n) + h(n) = g*(n) + h(n) [because, n is optimal path]
  - f(n) <= g*(n) + h*(n) [because, h(n)<= h*(n)]
  - f(n) <= C*, but it is a contradiction



# Heuristic Function

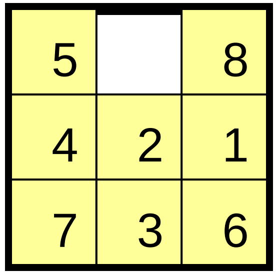
- Function h(N) that estimates the cost of the cheapest path from node N to goal node.
- Example: 8-puzzle



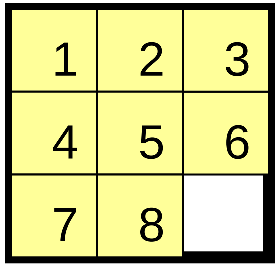

h(N) = number of misplaced tiles
= 6

# Heuristic Function

- Function h(N) that estimates the cost of the cheapest path from node N to goal node.
- Example: 8-puzzle




N




goal

h(N) = sum of the distances of every tile to its goal position

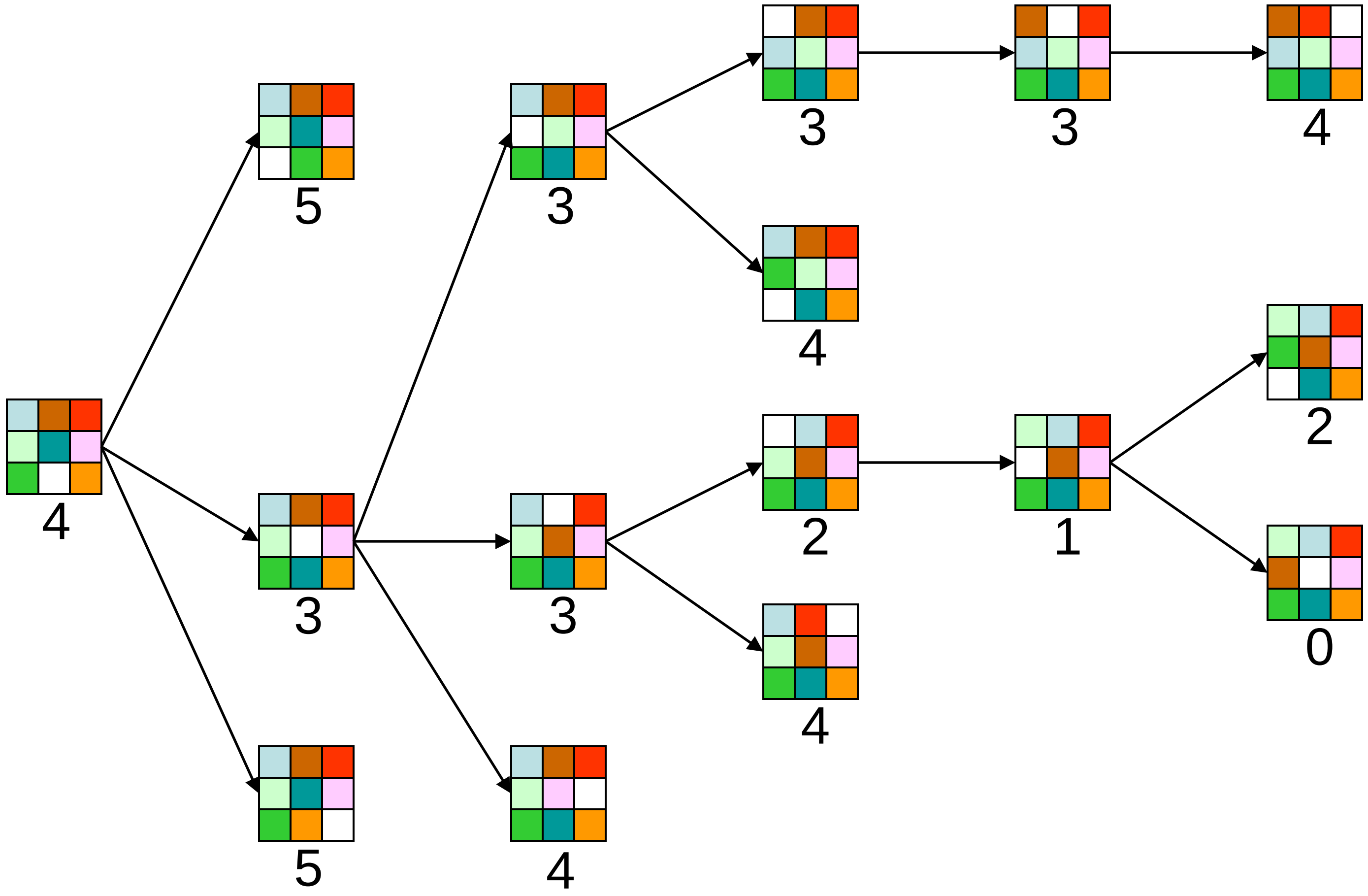
= 3 + 1 + 3 + 0 + 2 + 1 + 0 +3

[Here, four and seven are in right place]

= 13

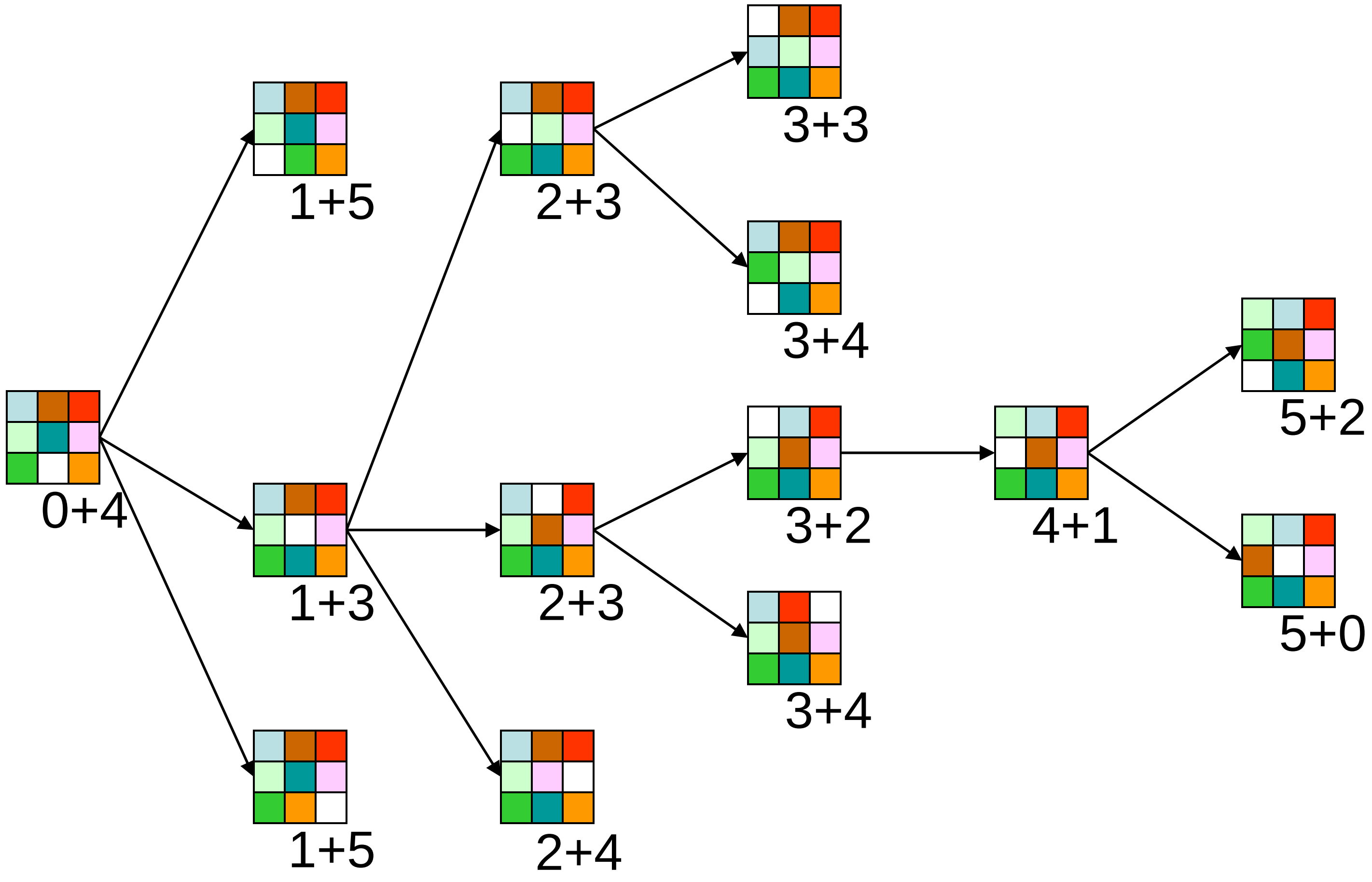
# 8-puzzle

f(N) = h(N) = number of misplaced tiles



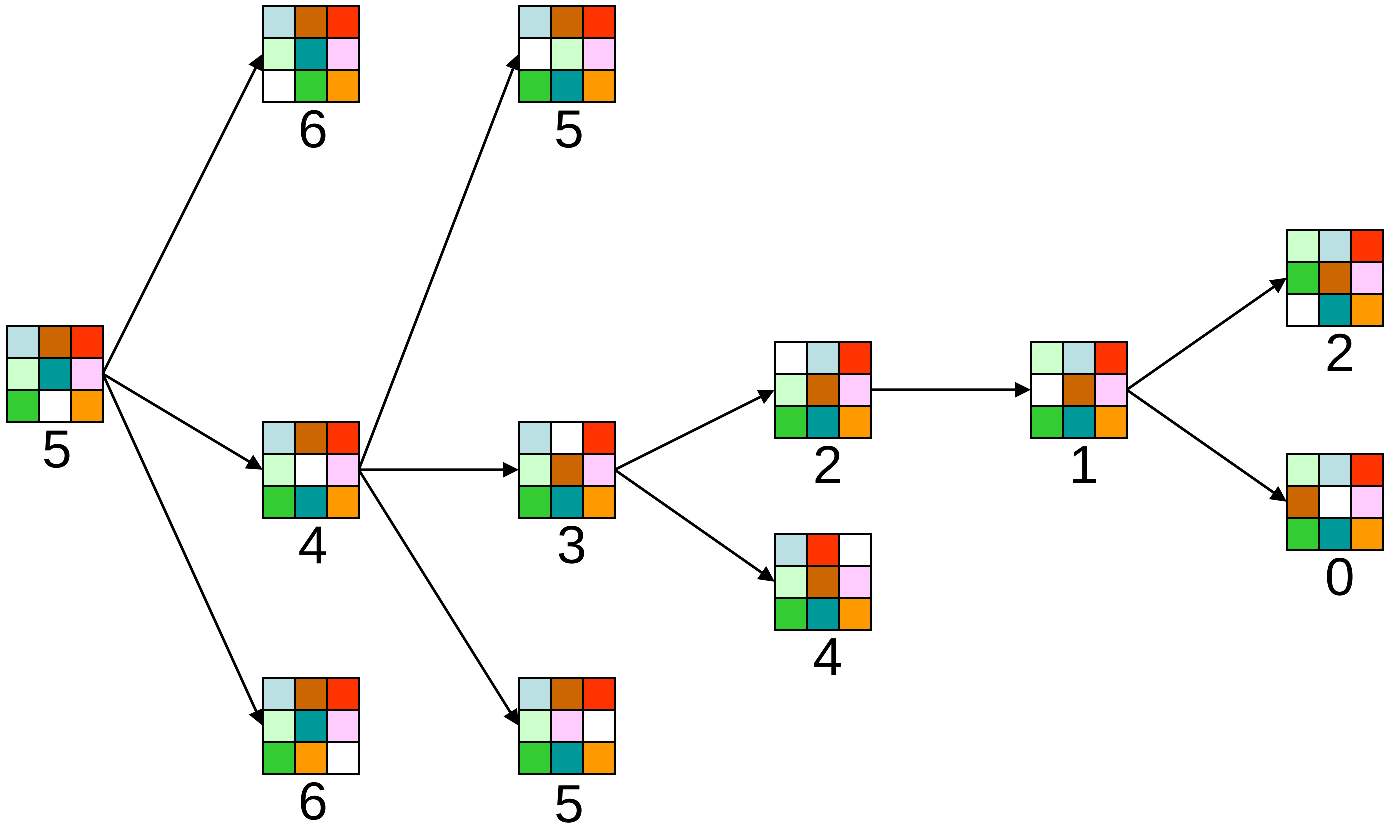
# 8-puzzle

f(N) = g(N) + h(N)
with h(N) = number of misplaced tiles



# 8-puzzle

$f(N) = h(N) = \Sigma$ distances of tiles to goal



# 8-puzzle

| | | |
|---|---|---|
| 5 | | 8 |
| 4 | 2 | 1 |
| 7 | 3 | 6 |

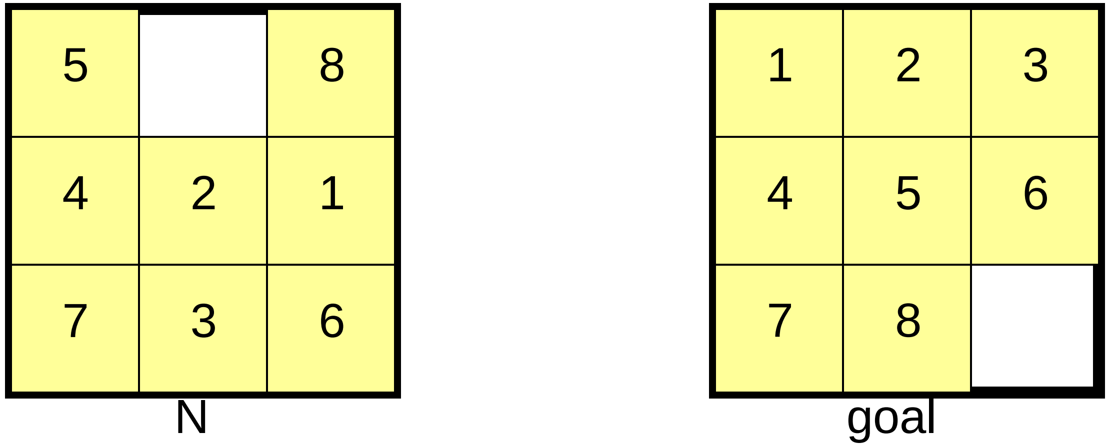
N

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

goal

- h1(N) = number of misplaced tiles = 6 is admissible
- h2(N) = sum of distances of each tile to goal = 13
  is admissible

# 8-puzzle




- h1(N) = number of misplaced tiles
- h2(N) = sum of distances of each tile to goal

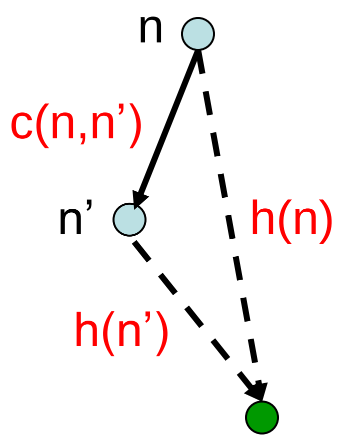
are both consistent

# Consistent Heuristic

- The admissible heuristic h is consistent (or satisfies the monotone restriction) if for every node n and every successor n′ of n:

  $$h(n) \leq c(n,n') + h(n')$$

  This is a form of triangular inequality.

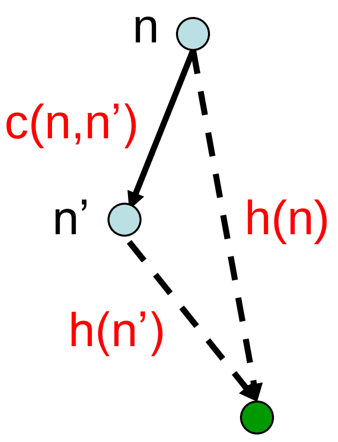



- A side of a triangle cannot be longer than the sum of the other two sides.

# Claims

- If h is consistent, then the function f along any path is non-decreasing:

$$f(n) = g(n) + h(n)$$
$$f(n') = g(n) + c(n,n') + h(n')$$

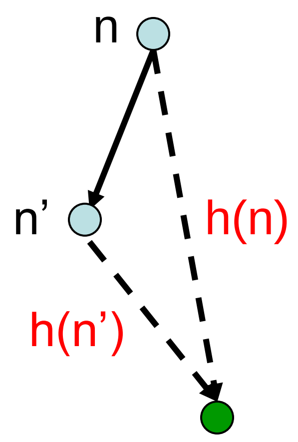


# Claims

- If h is consistent, then the function f along any path is non-decreasing:

$$f(n) = g(n) + h(n)$$
$$f(n') = g(n) + c(n,n') + h(n')$$

$$h(n) \le c(n,n') + h(n')$$
$$f(n) \le f(n')$$

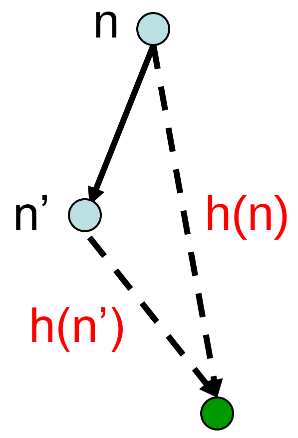


# Claims

- If h is consistent, then the function f along any path is non-decreasing:

$$f(n) = g(n) + h(n)$$
$$f(n') = g(n) + c(n,n') + h(n')$$

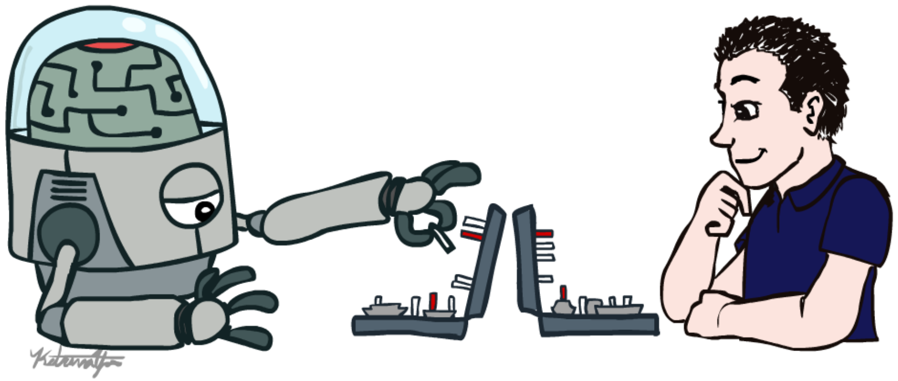
$$h(n) \leq c(n,n') + h(n')$$
$$f(n) \leq f(n')$$




- If h is consistent, then whenever A* expands a node it has already found an optimal path to the state associated with this node

# Next class?

- Local search algorithm
- Hill climbing problem



# Thanks!