Artificial Intelligence
CE-417, Group 1
Computer Eng. Department
Sharif University of Technology

Spring 2023

By Mohammad Hossein Rohban, Ph.D.

Courtesy: Most slides are adopted from CSE-573 (Washington U.), original slides for the textbook, and CS-188 (UC. Berkeley).

Problem Space and Uninformed Search

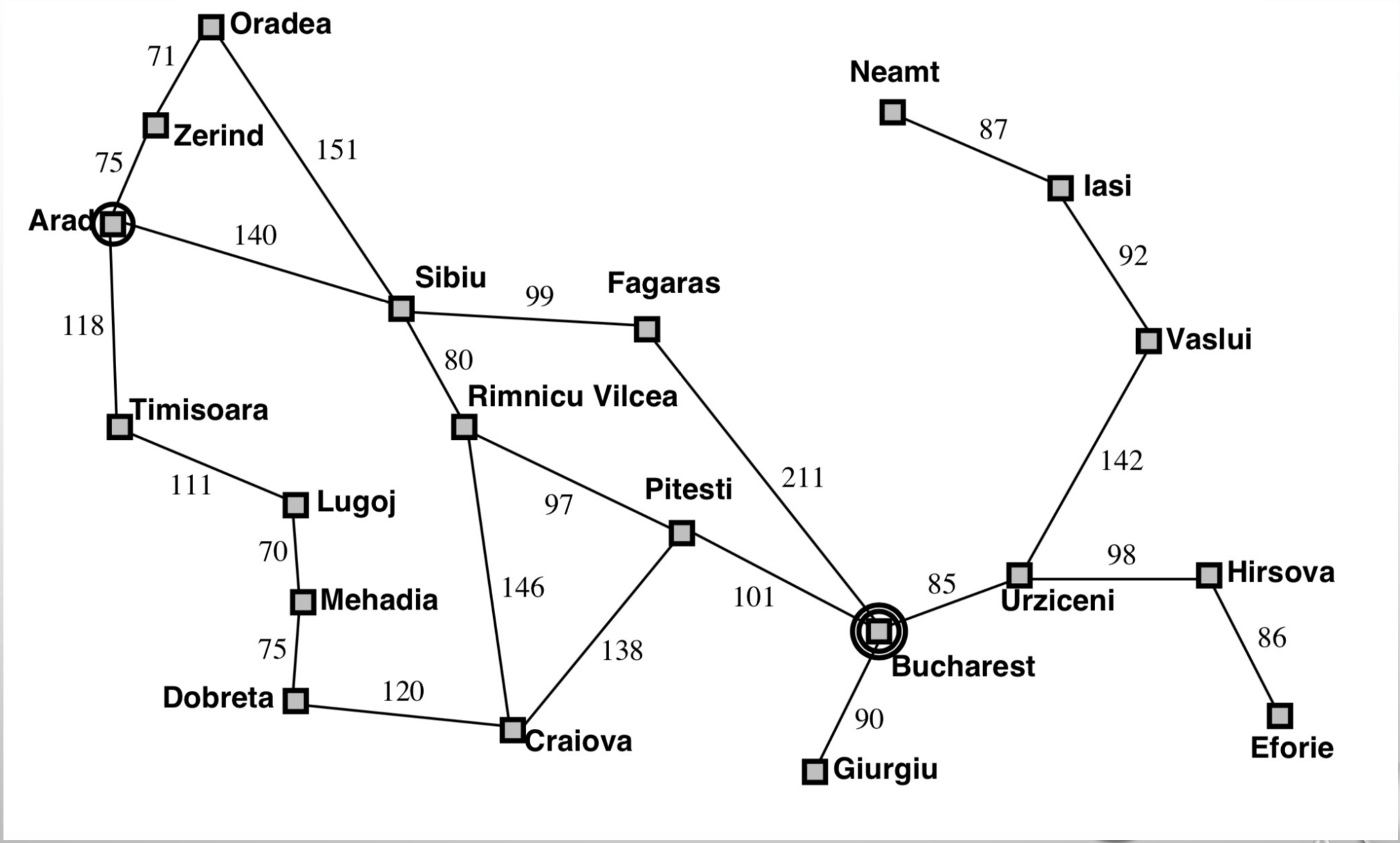
Problem solving agents

- On holiday in Romania; currently in Arad.

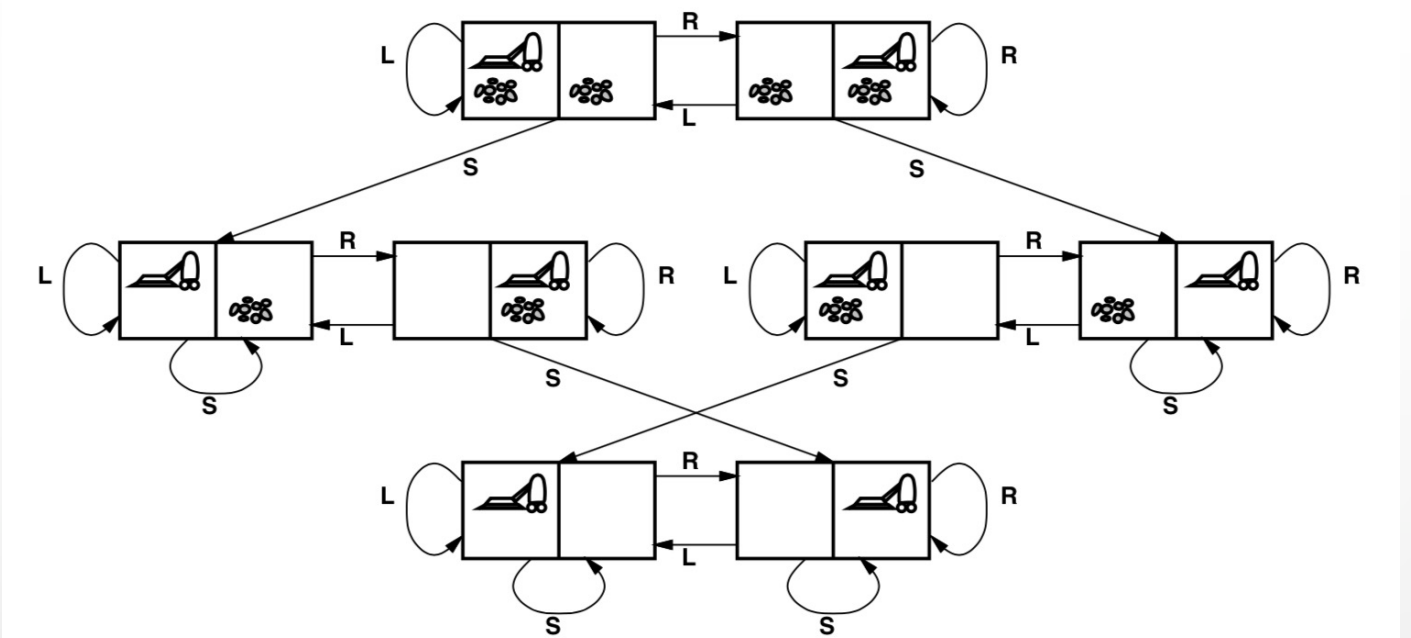
  Flight leaves tomorrow from Bucharest

- Formulate goal: **be in Bucharest**
- Formulate problem:
  - states: **various cities**
  - actions: **drive between cities**
- Find solution: **sequence of cities**, e.g., Arad, Sibiu, Fagaras, Bucharest

Problem solving agents (cont.)

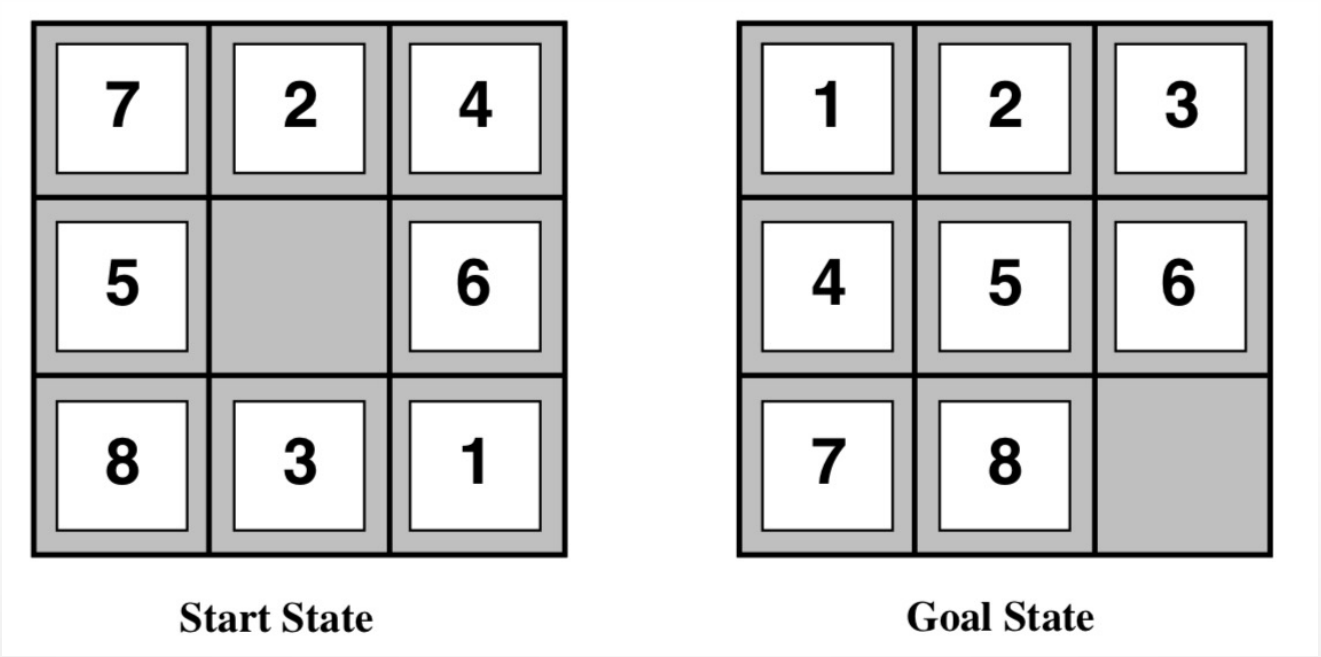


Another example: vacuum world



- States: **integer dirt and robot locations** (ignore dirt amounts etc.)
- Actions: **Left**, **Right**, **Suck**, **NoOp**
- Goal test: **no dirt**
- Path cost: **1 per action (0 for NoOp)**

Another example: The 8-puzzle



- States: **integer locations of tiles** (ignore intermediate positions)
- Actions: **move blank left, right, up, down** (ignore unjamming etc.)
- Goal test: **= goal state** (given)
- Path cost: **1 per move**
- [Note: optimal solution of n-Puzzle family is NP-hard]

Tree Search Algorithms

- Basic idea:

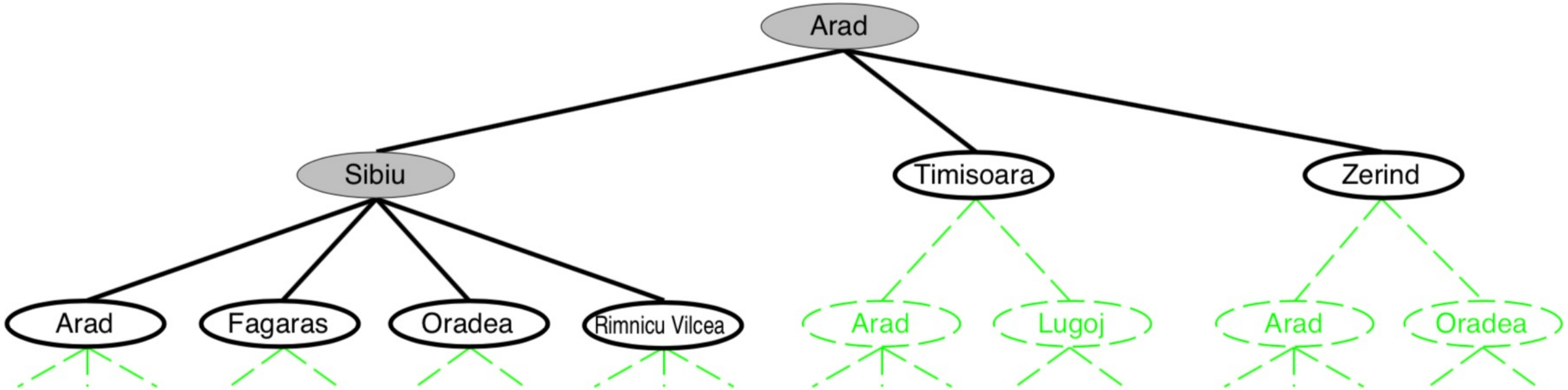
  offline, simulated exploration of state space

  by generating successors of already-explored states

(a.k.a. expanding states)

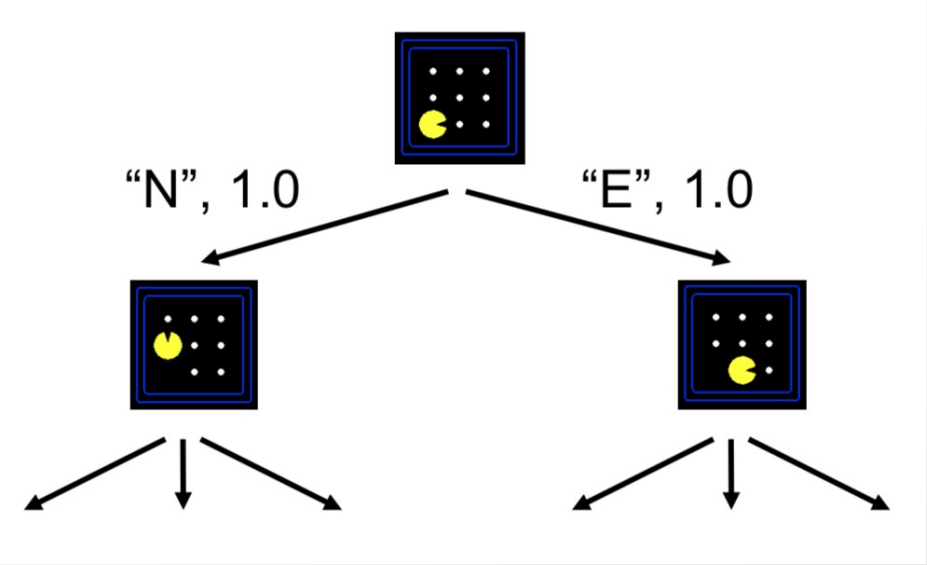
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

Search Tree Example



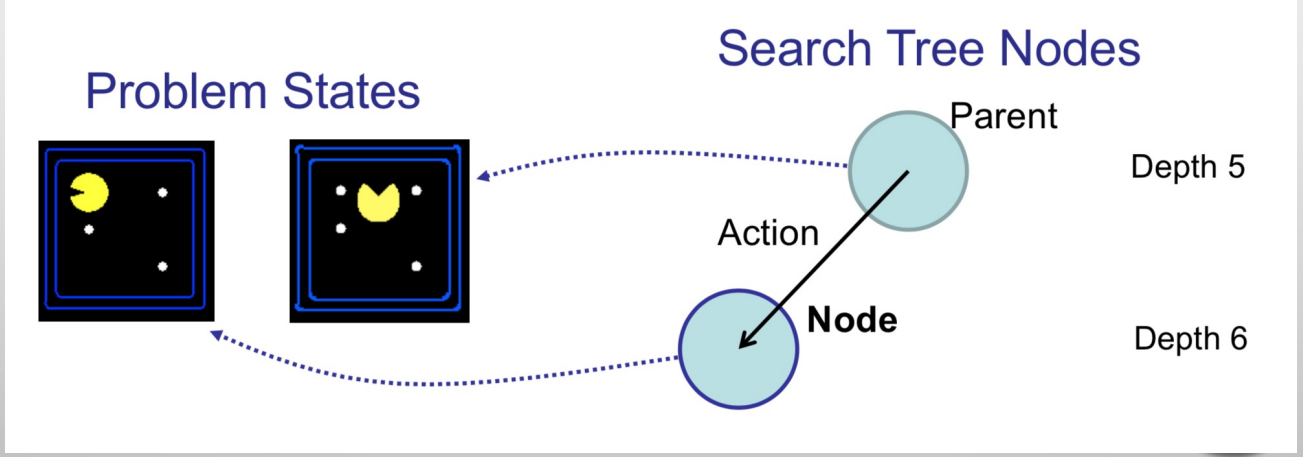
Search Tree

- A search tree:
  - Start state at the root node
  - Children correspond to successors
  - Nodes ***contain*** states, correspond to PLANS to those states
  - Edges are labeled with actions and costs
  - For most problems, we can never actually build the whole tree



States vs. Nodes

- Vertices in state space graphs are problem states
- Represent an abstracted state of the world
- Have successors, can be goal / non-goal, have multiple predecessors
- Vertices in search trees ("Nodes") are plans
- Contain a **problem state** and one parent, a path length, a depth, and a cost
- Represent a plan (sequence of actions) which results in the node's state
- **The same problem state may be achieved by multiple search tree nodes**



Search Strategies

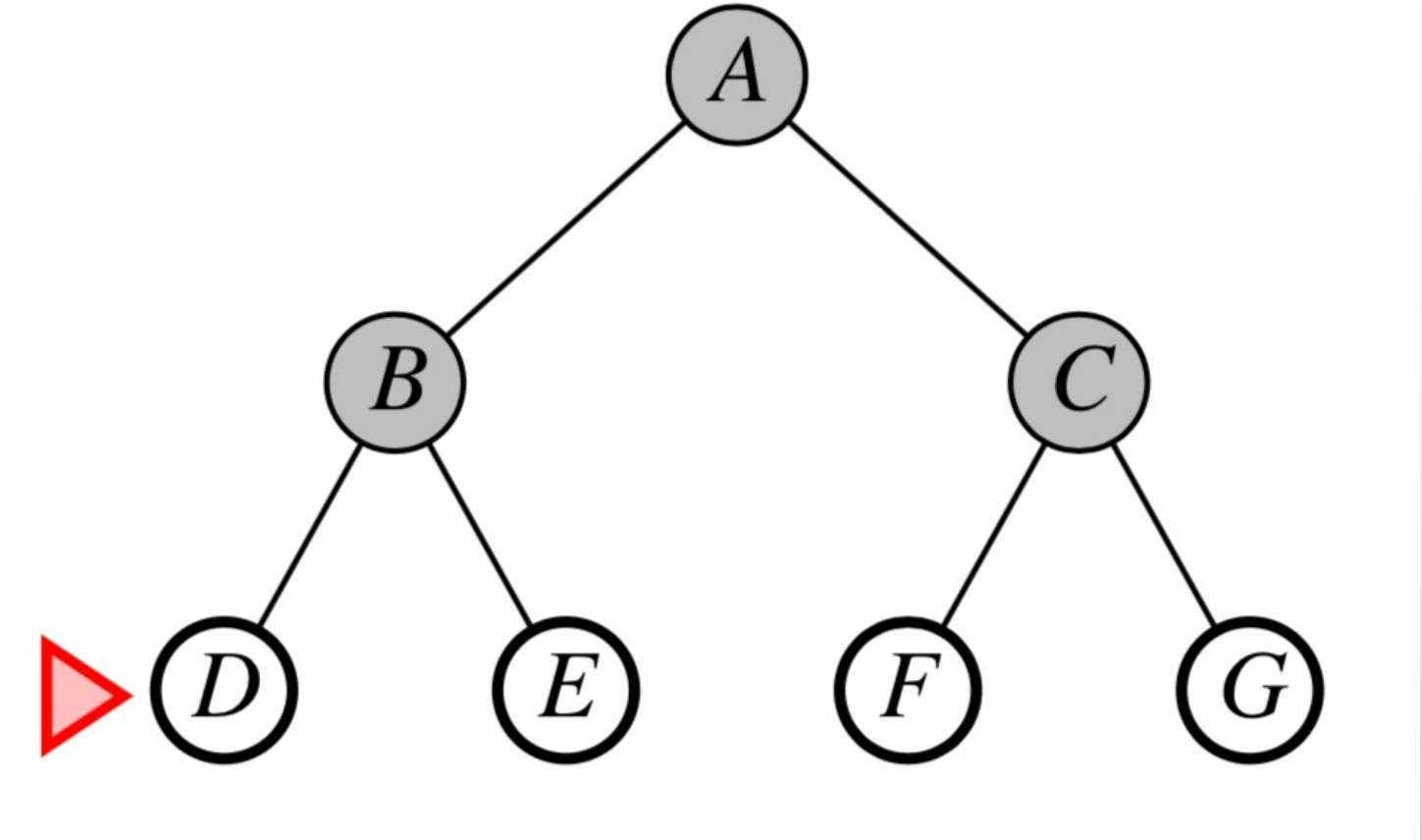
- A strategy is defined by picking **the order of node expansion**
- Strategies are evaluated along the following dimensions:
  - **completeness:** does it always find a solution if one exists?
  - **time complexity:** number of nodes generated/expanded
  - **space complexity:** maximum number of nodes in memory
  - **optimality:** does it always find a least-cost solution?
- Time and space complexity are measured in terms of
  - ***b***: maximum branching factor of the search tree
  - ***d***: depth of the least-cost solution
  - ***m***: maximum depth of the state space (may be ∞)

Search Strategies

- **Uninformed** strategies use only the information available in the problem definition
- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search

Breadth-first search

- Expand shallowest unexpanded node



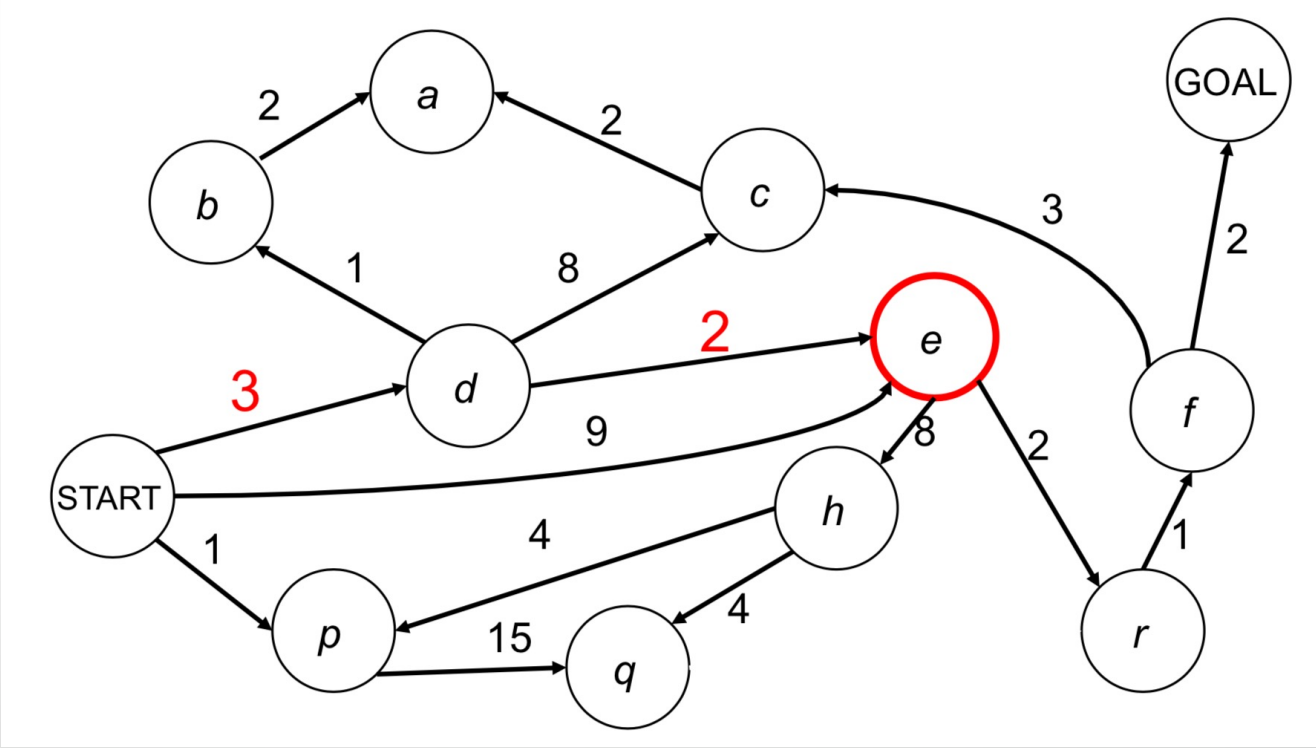
Best-first search

- Generalization of breadth-first search
- Cost function f(n) applied to each node
  - Breadth-first search : f(n) = depth(n)
  - Dijkstra's Algorithm (Uniform cost) : f(n) = the sum of edge costs from start to n

Uniform Cost Search

- Best first, where

  f(n) = “cost from **start** to **n**”



aka “Dijkstra’s Algorithm”

Depth-limited search

= depth-first search **with depth limit l**, i.e., nodes at depth l have no successors

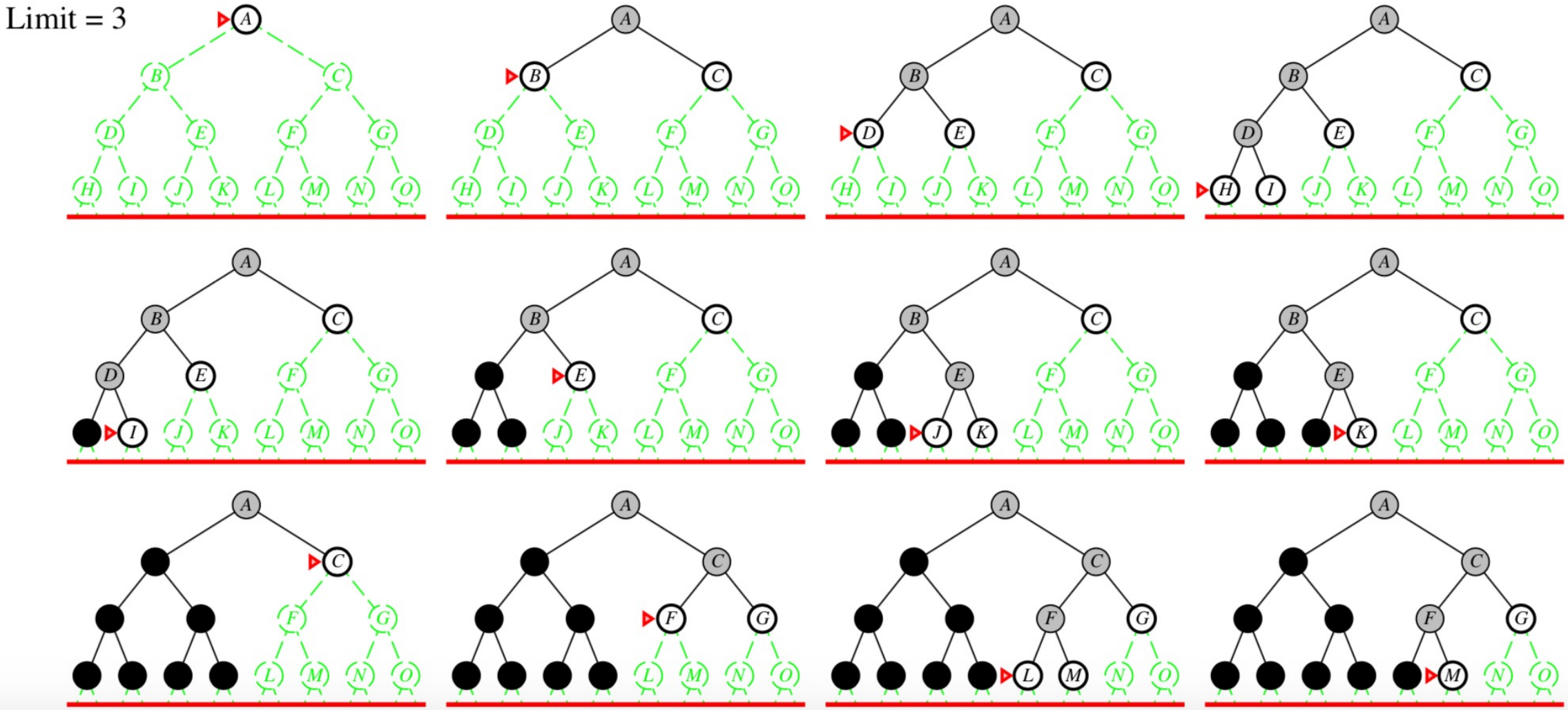
```
function DEPTH-LIMITED-SEARCH(problem, limit) returns soln/fail/cutoff
    RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
    cutoff-occurred? ← false
    if GOAL-TEST(problem, STATE[node]) then return node
    else if DEPTH[node] = limit then return cutoff
    else for each successor in EXPAND(node, problem) do
        result ← RECURSIVE-DLS(successor, problem, limit)
        if result = cutoff then cutoff-occurred? ← true
        else if result ≠ failure then return result
    if cutoff-occurred? then return cutoff else return failure
```

Iterative deepening search (cont.)

- Gradually increasing the limit in depth-limited search, until the solution is found:

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
    end
```

Iterative deepening search (cont.)



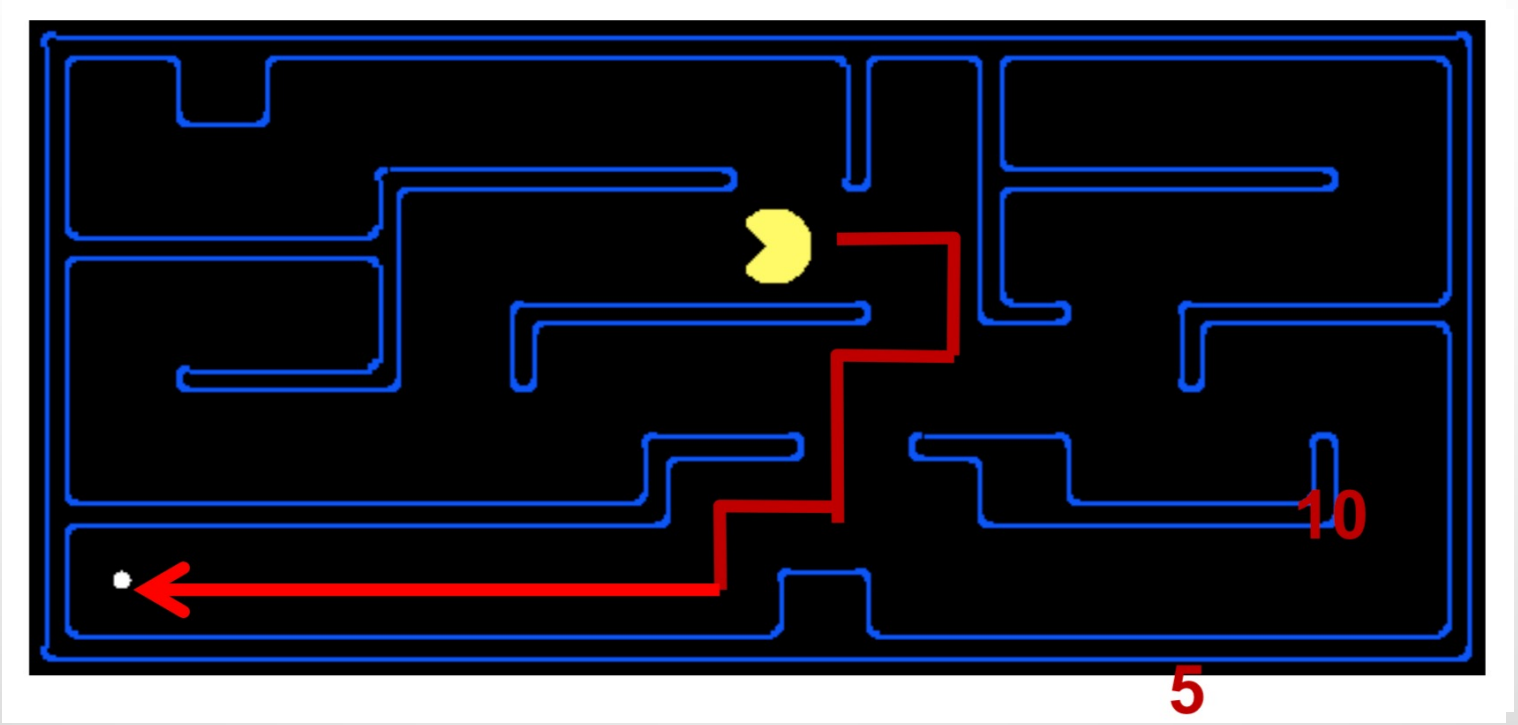
Informed search

Blind vs. Heuristic Search

- Blind:
  - Search in all directions systematically
- Heuristic Guidance:
  - How far is the goal state from a given state approximately?

What is a "Heuristic"?

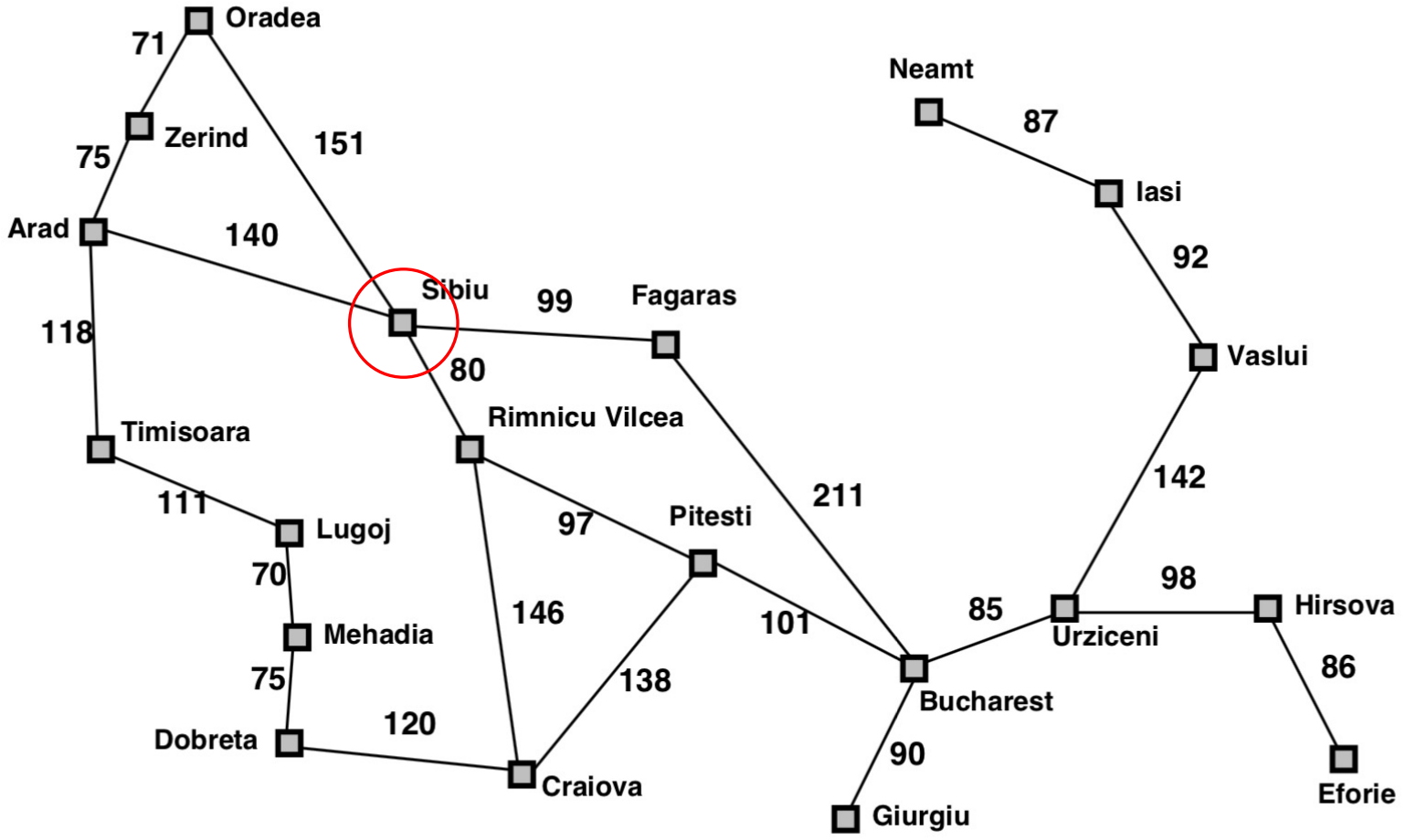
- An ***estimate*** of how close a state is to a goal
- Designed for a particular search problem



- Examples: Manhattan distance: 10+5 = 15; Euclidean distance: 11.2
- Actual distance to goal: 2+4+2+1+8= 17

Greedy Search

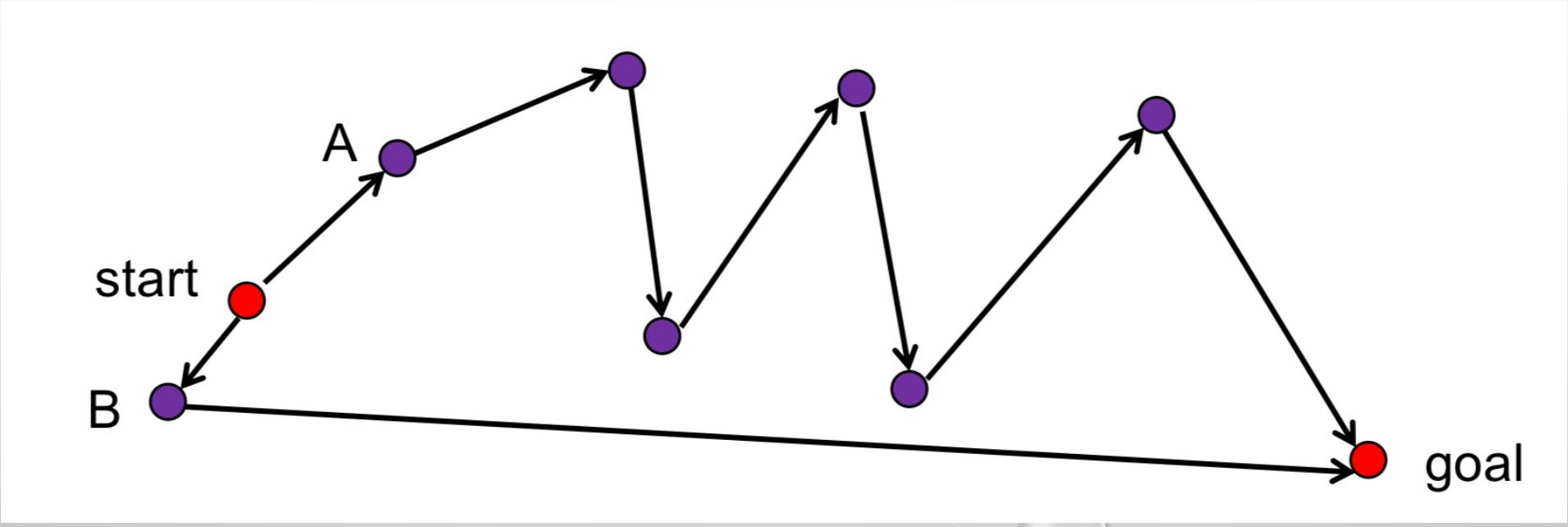
- *Best first with f(n) = heuristic estimate of distance to goal*



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

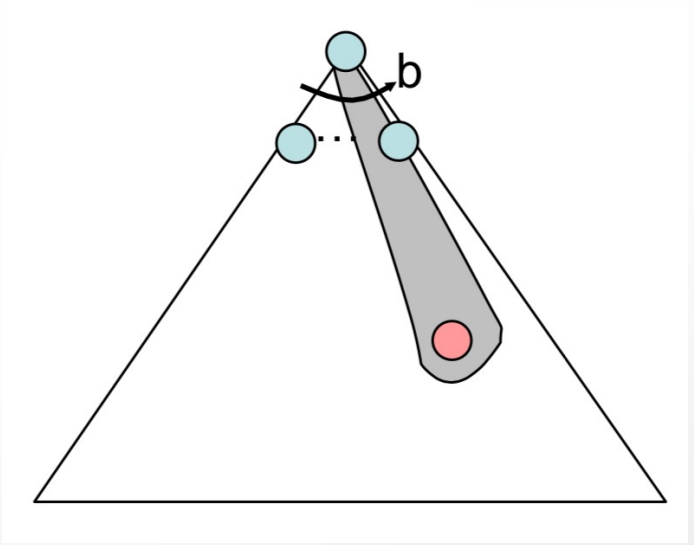
How can it go wrong?

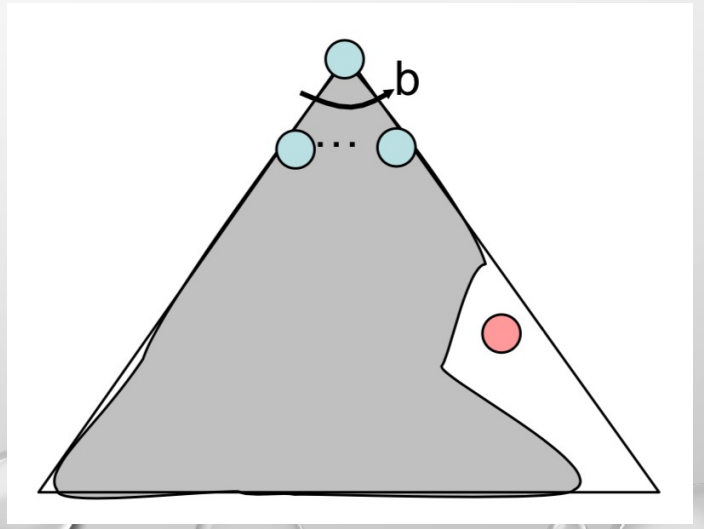
- Expand the node that seems closest…



Problems with the Greedy Search

- **Common case**:
  - Best-first takes you straight to a (suboptimal) goal
- **Worst-case**: like a badly-guided DFS
  - Can explore everything
  - Can get stuck in loops if no cycle checking
- Like DFS in completeness
  - Complete w/ cycle checking
  - *If* finite # states

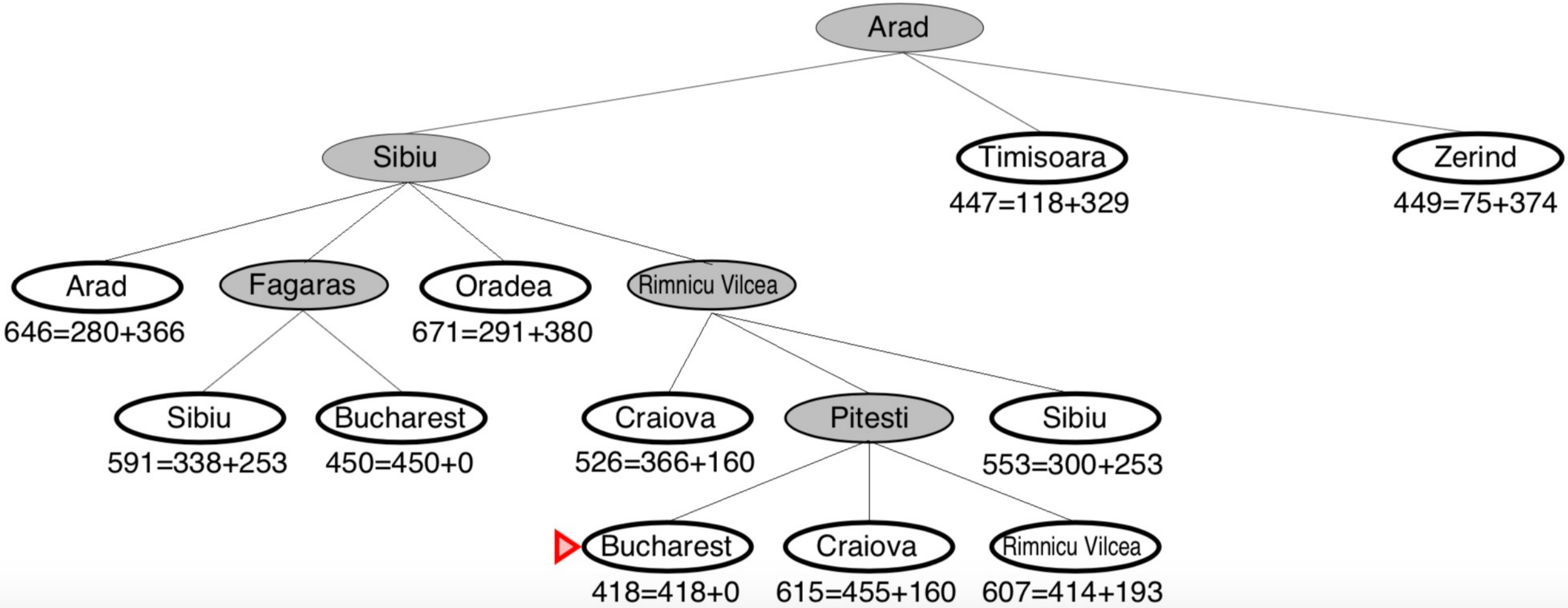




A* Search

- Hart, Nilsson & Rafael 1968
- Best first search with $f(n) = g(n) + h(n)$
  - $g(n)$ = sum of costs from start to n
  - $h(n)$ = estimate of lowest cost path n → goal
- $h(goal) = 0$
- Can view as cross-breed:
  - g(n) ~ uniform cost search
  - h(n) ~ greedy search
- Best of both worlds...

A* example


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

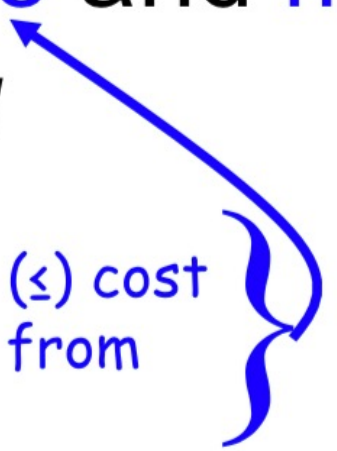
A* optimality (tree-search)?

Theorem: If h(n) is admissible then A* is optimal in tree search.

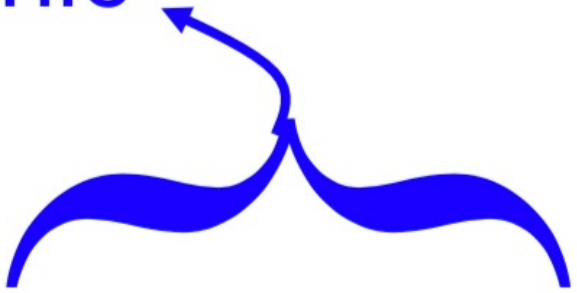
A* optimality (graph-search)?

If h(n) is admissible and monotonic

then A* is ***optimal***

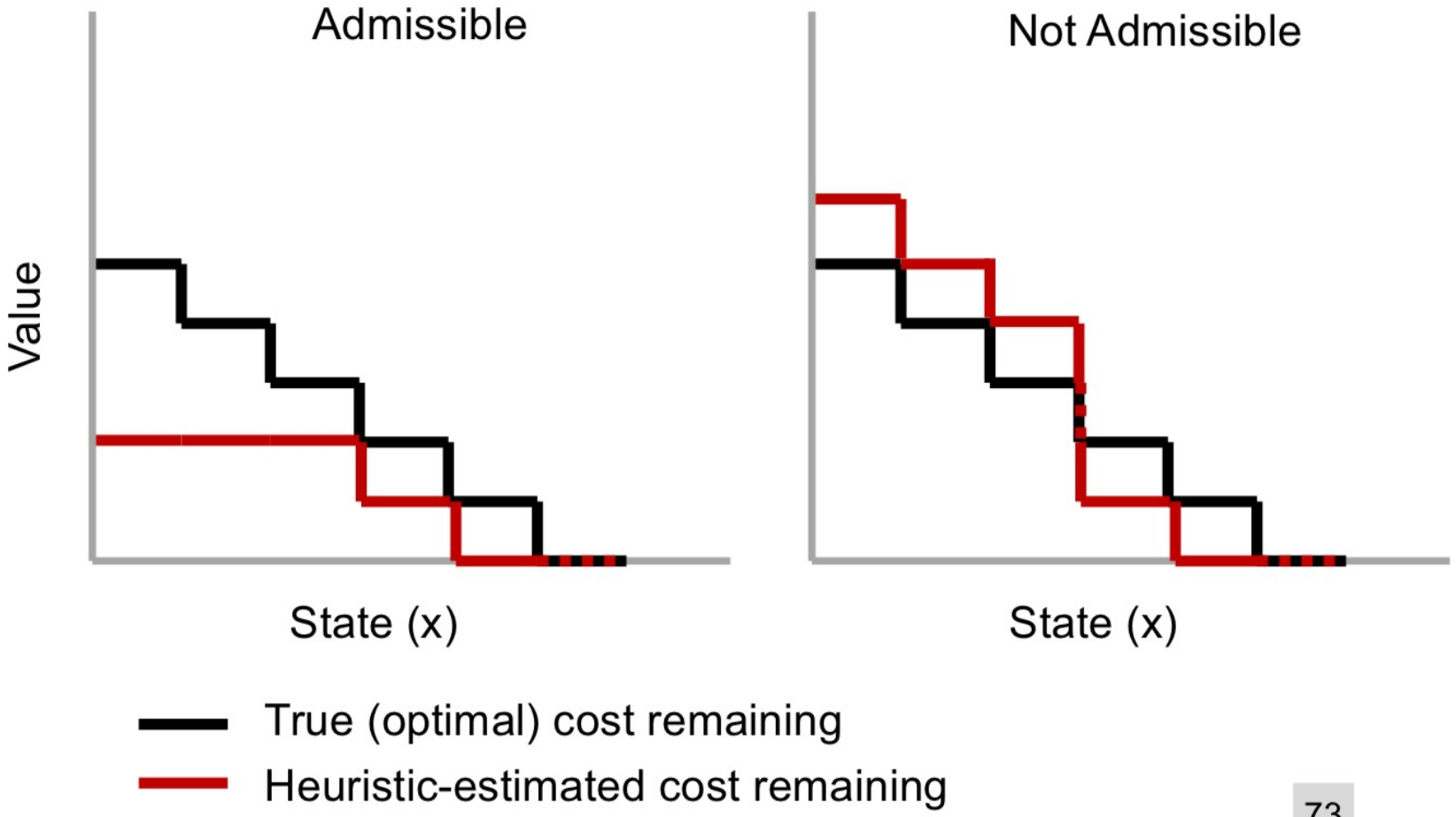


Underestimates (≤) cost of reaching goal from node



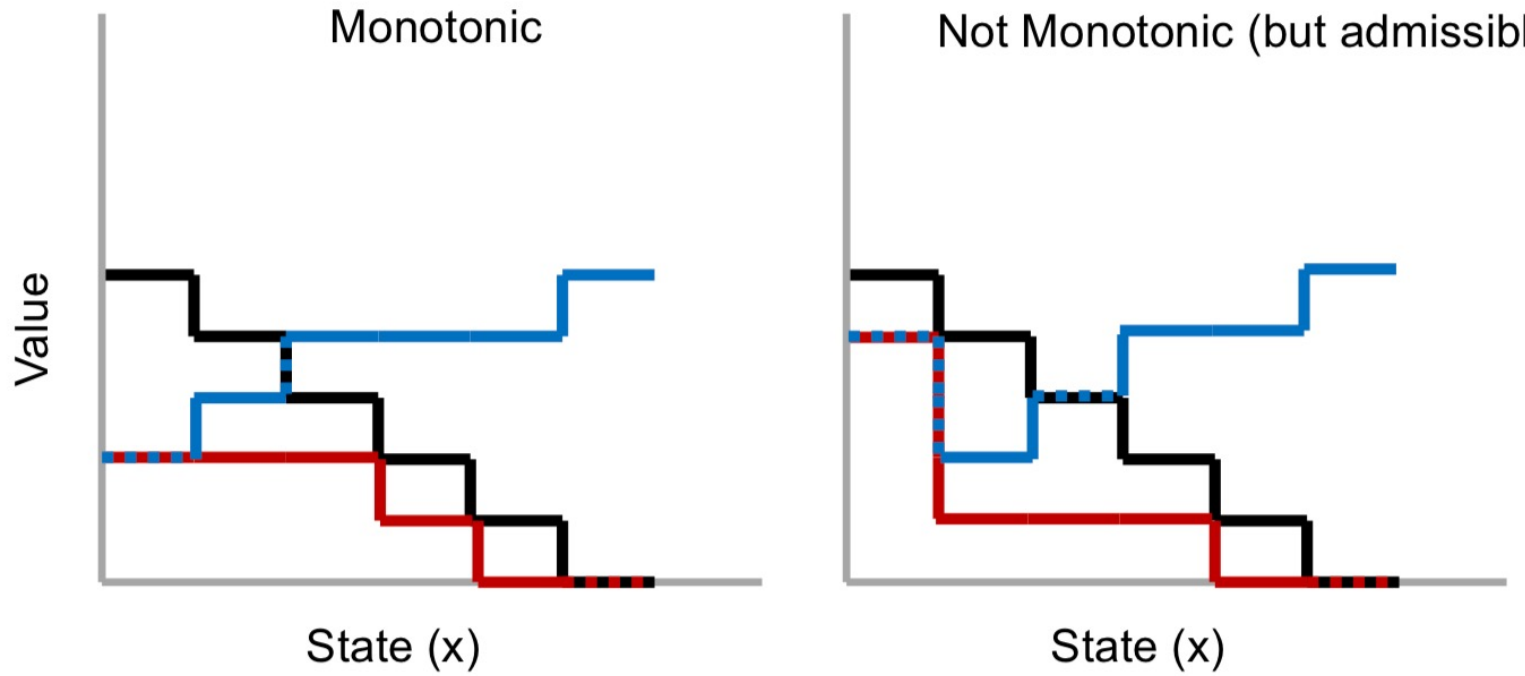
f values never decrease From node to descendants (triangle inequality)

Admissible Heuristics



Slide credit: Travis Mandel

Monotonic (or Consistent) Heuristics



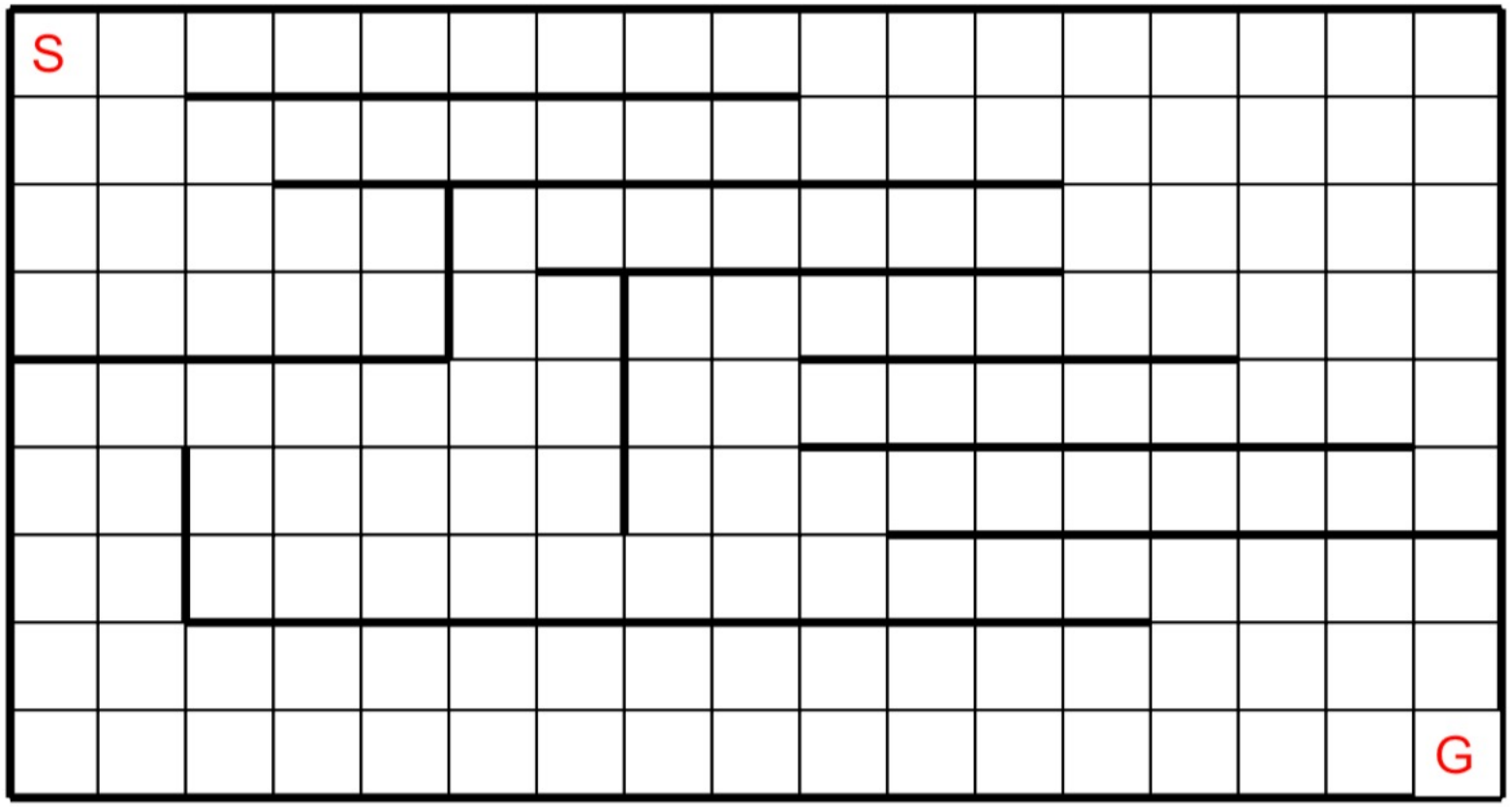
75

Slide credit: Travis Mandel

Example: Maze

- Is Manhattan distance
  - Admissible
  - Monotonic

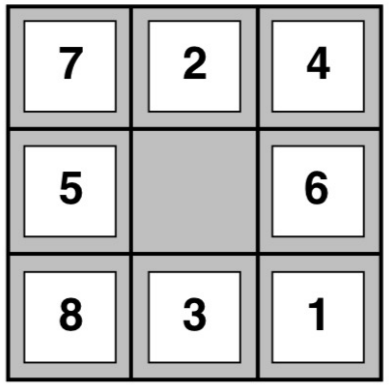
for Maze?



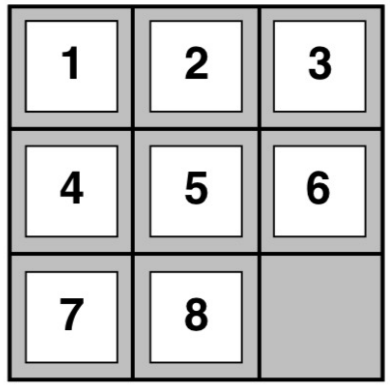
Another example: the 8-puzzle

E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles
$h_2(n)$ = total Manhattan distance
(i.e., no. of squares from desired location of each tile)



Start State



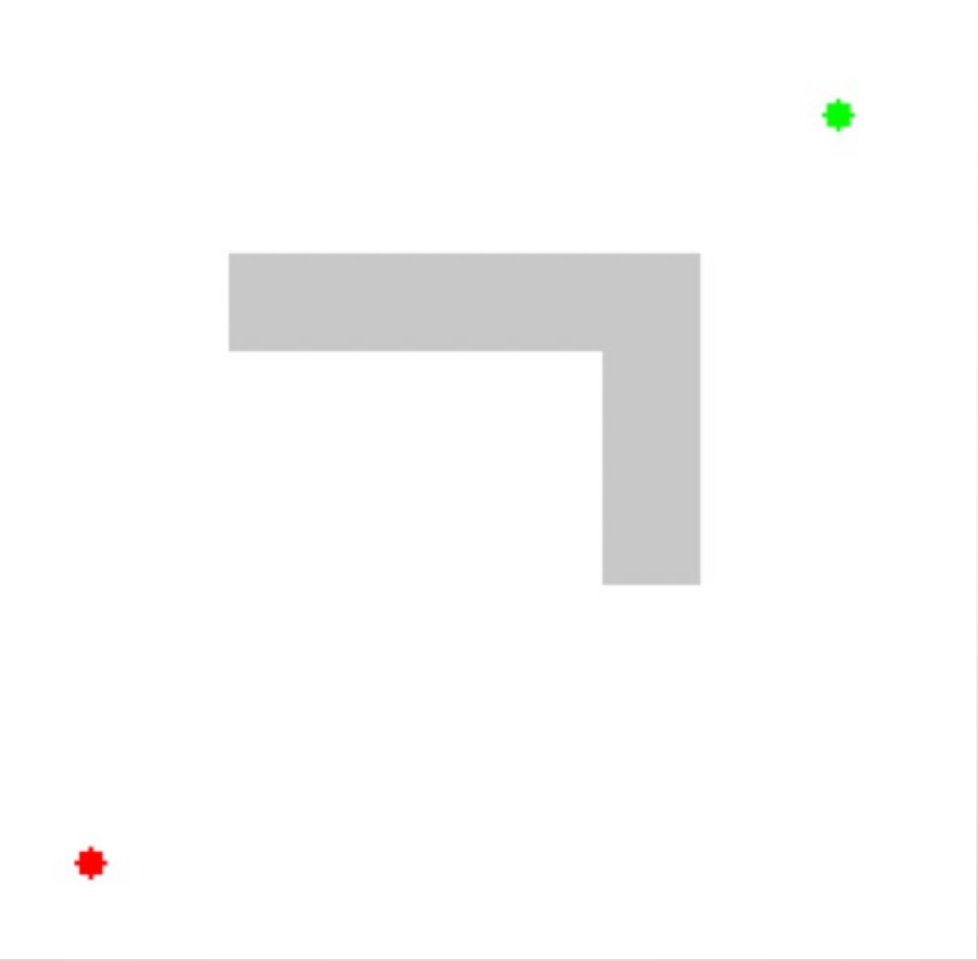
Goal State

$h_1(S) =$?? 6
$h_2(S) =$?? 4+0+3+3+1+0+2+1 = 14

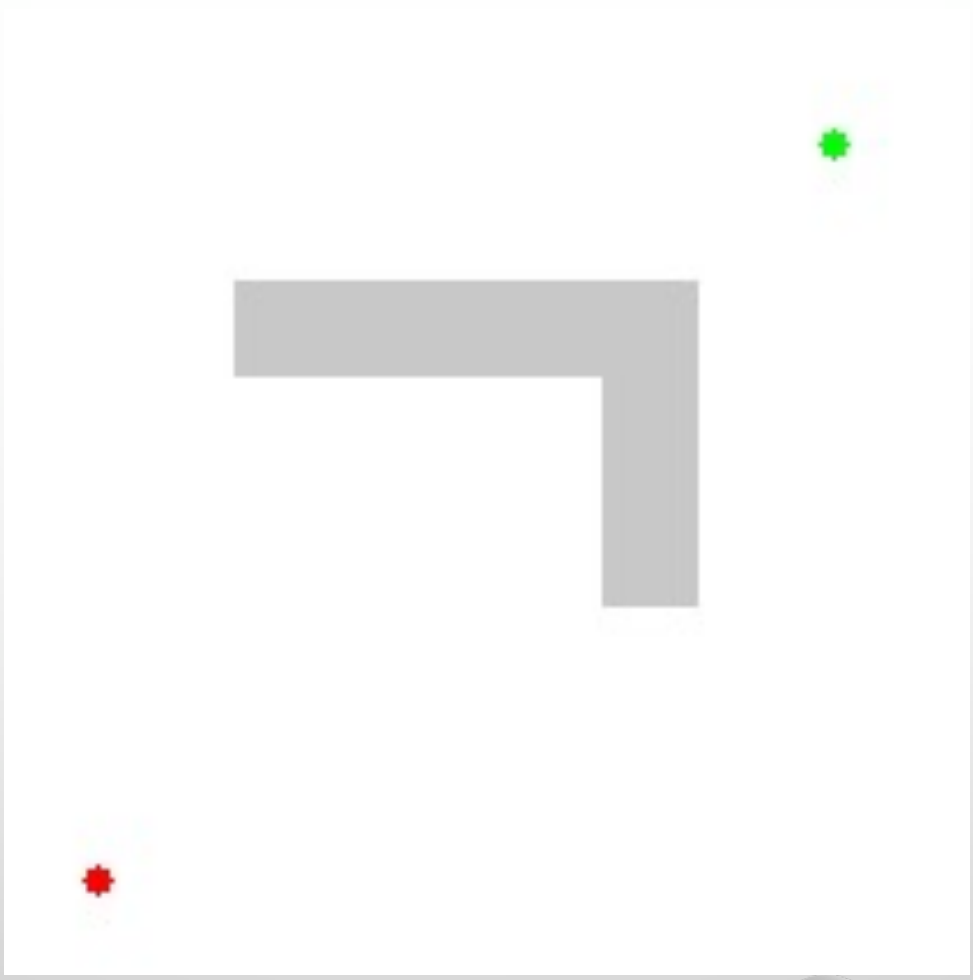
Heuristics Dominance

- If $h_2(n) \geq h_1(n)$ for all n (both admissible) then h_2 dominates h_1 and is better for search
- Typical search costs for n-puzzle:
  - d = 14
    - IDS = 3,473,941 nodes
    - $A^*(h_1)$ = 539 nodes
    - $A^*(h_2)$ = 113 nodes
  - d = 24
    - IDS ≈ 54,000,000,000 nodes
    - $A^*(h_1)$ = 39,135 nodes
    - $A^*(h_2)$ = 1,641 nodes
- Given any admissible heuristics h_a, h_b, $h(n) = \max(h_a(n), h_b(n))$ is also admissible and dominates h_a, h_b

A* demo



A* demo



A* Summary

- **Pros**
  - Produces optimal cost solution!
  - Does so quite quickly (focused)
    - A* is **optimally efficient** for any given heuristics function.
- **Cons**
  - Maintains priority queue…
  - Which can get exponentially big
  - Theorem: Exponential growth will occur unless $|h(n) - h^*(n)| \leq O(\log h^*(n))$.