Lecture 3

CS 6260 Automated Planning and Learning

Informed Search

Department of Computer Science and Engineering

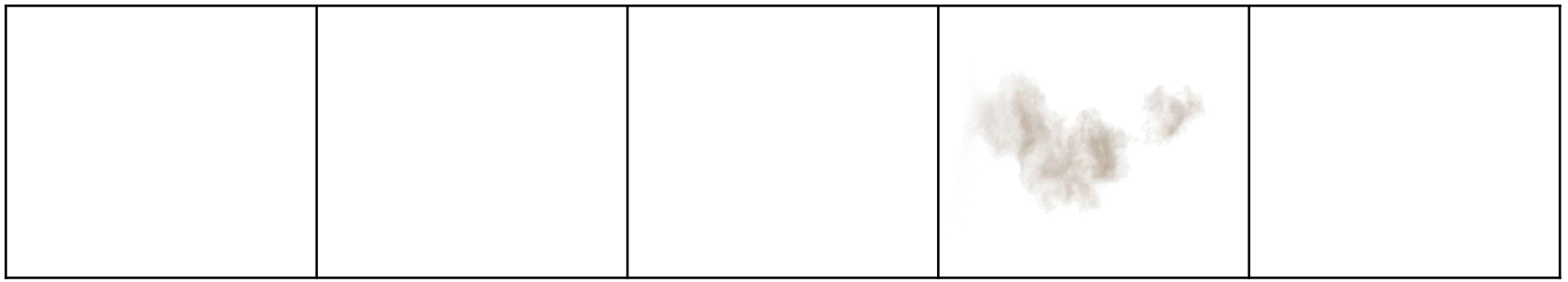
Indian Institute of Technology Madras

Some material based on lecture slides for ASU's *CSE571 Artificial Intelligence by Siddharth Srivastava*.

Recap:
Non-Observable vs Partially Observable

Forms of Solutions

- Deterministic, non-observable (No sensor)
  - "Conformant planning": solution is a plan, but often no solutions exist



???

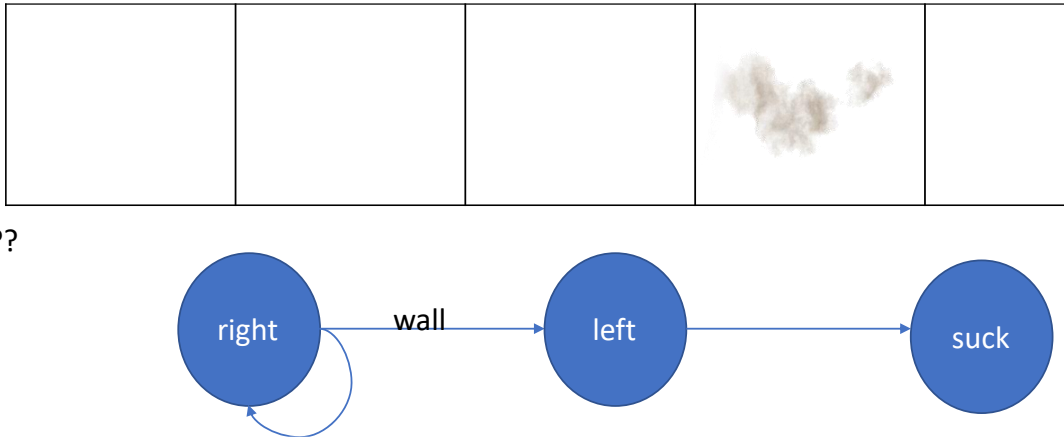
Right, Right, Right, Right, Suck, Left, Suck, Left, Suck, Left, Suck, Left, Suck

Exactly 4 rights, because it knows actions are deterministic

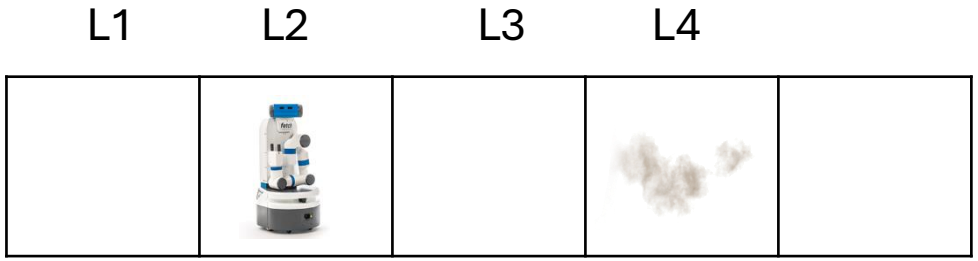
Forms of Solutions

- Partially observable
  - Sensors report noisy, incomplete information about the current state
  - Solution is a finite state machine

???



Forms of Solutions

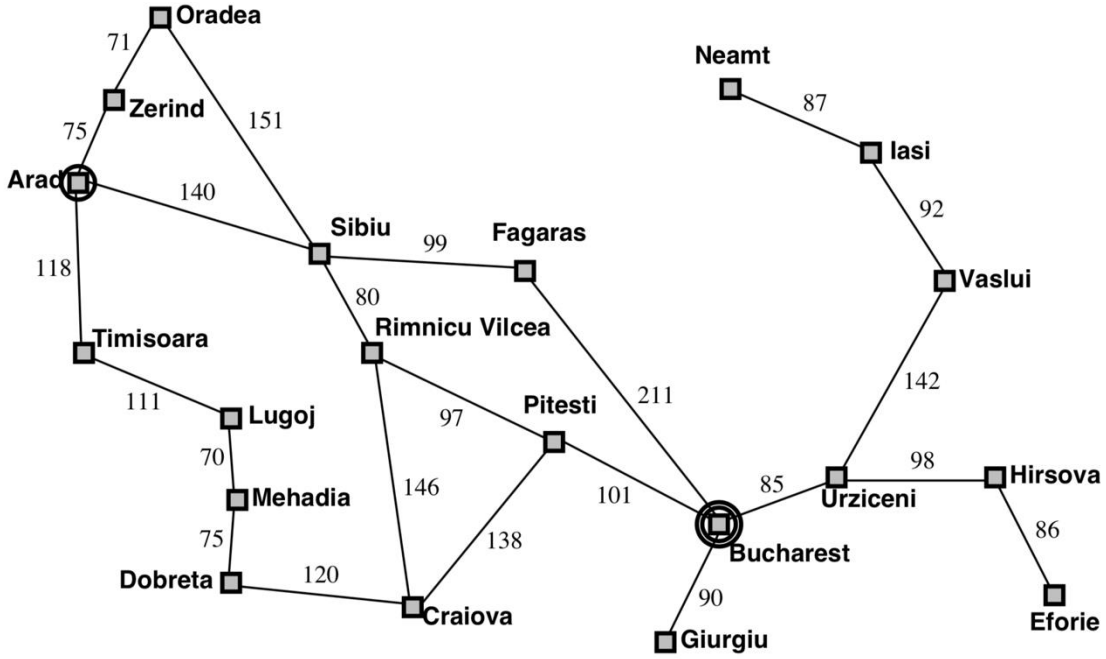


- State:

Recap: General Form of Search Problems

Search Problem Defined Over an Implicit Graph

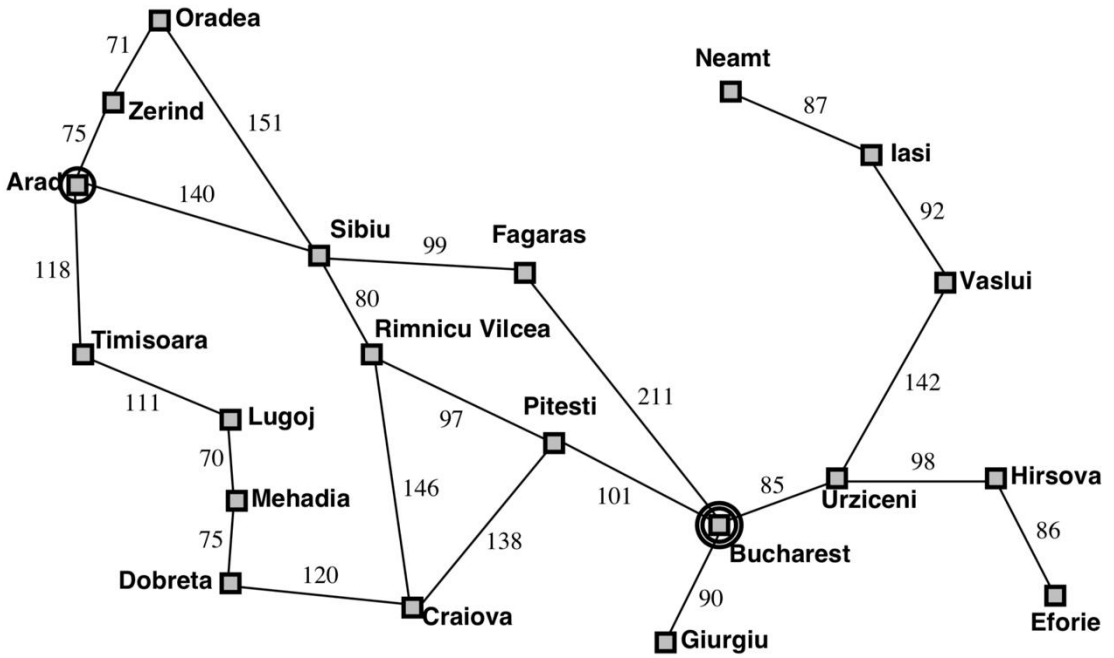
- A search problem is defined by
  - S set of possible states
    - Defined, but not enumerated a priori
  - $s_0 \in S$ start state (~vertex)
  - A set of actions
  - $N: S \rightarrow S$ successor function
    - $T: S \times A \rightarrow S$ deterministic transition function
    - $N(s) = \{s' : \exists a \, T(s, a) = s'\}$
  - $G: S \rightarrow \{True, False\}$ goal test
  - $C: S \times A \times S \rightarrow \mathbb{R}^+$ path cost



A **solution** to a search problem is a sequence of actions leading from s_o to a state satisfying the goal test

Role of Abstraction in Modeling Search Problems

- Search problem formulations are models
  - Most models are abstractions
  - Driving from one city to another involves
    - Traffic, one-way streets, fuel considerations, rest stops, …
- Abstract state = set of real states
- Abstract action = set of procedures

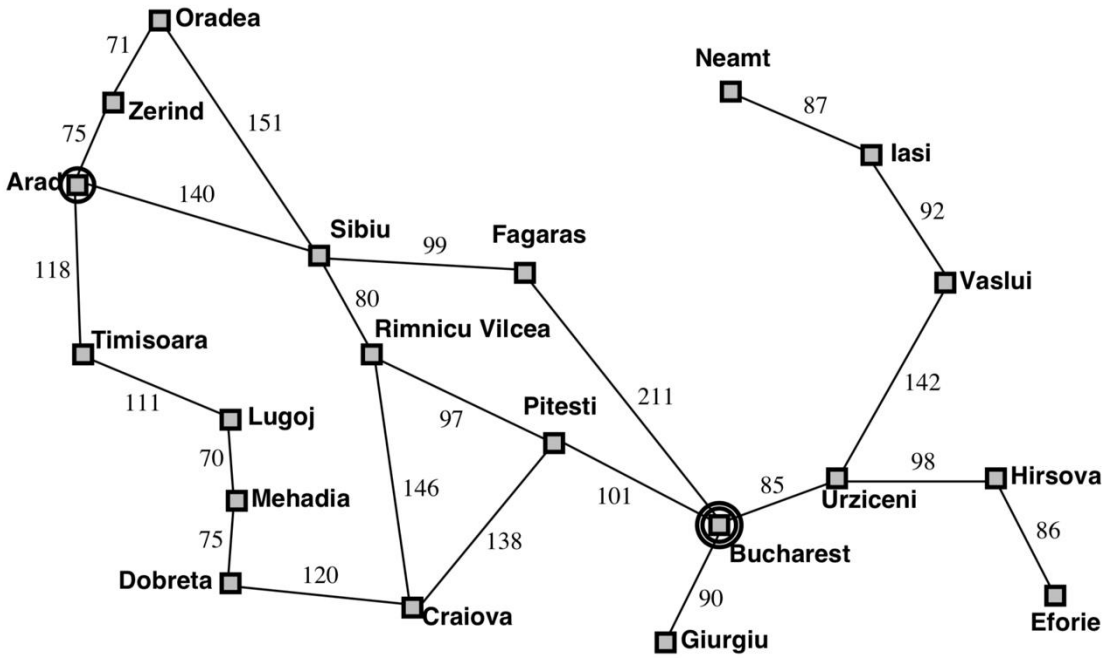


Abstraction is a common and important aspect of modeling

Abstract problem should be easier than the real problem!

Role of Abstraction in Modeling Search Problems

- For the abstract solution to be usable we need:
  - $\forall s, a, s'$ such that $T(s, a) = s'$
    - Every real state in s gets to some real state in s' by following a procedure defined by a
    - "Downward refinability"
  - There should be a way to go from all real locations "in Arad" to some real location "in Zerind"

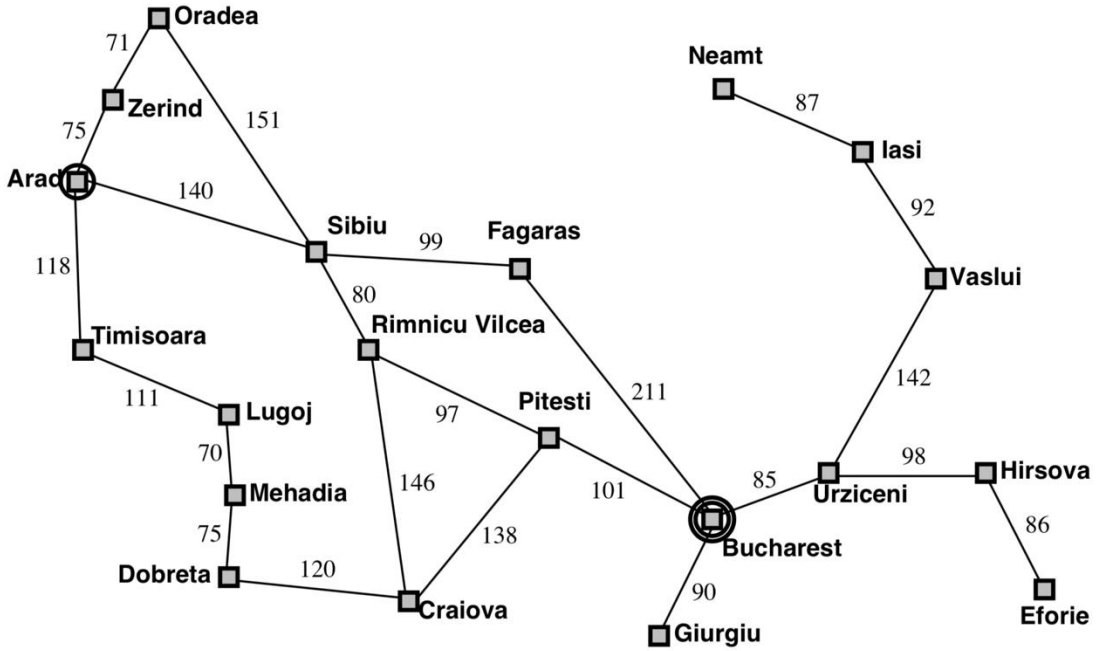


Abstract problem should be easier than the real problem!

Solving Search Problems

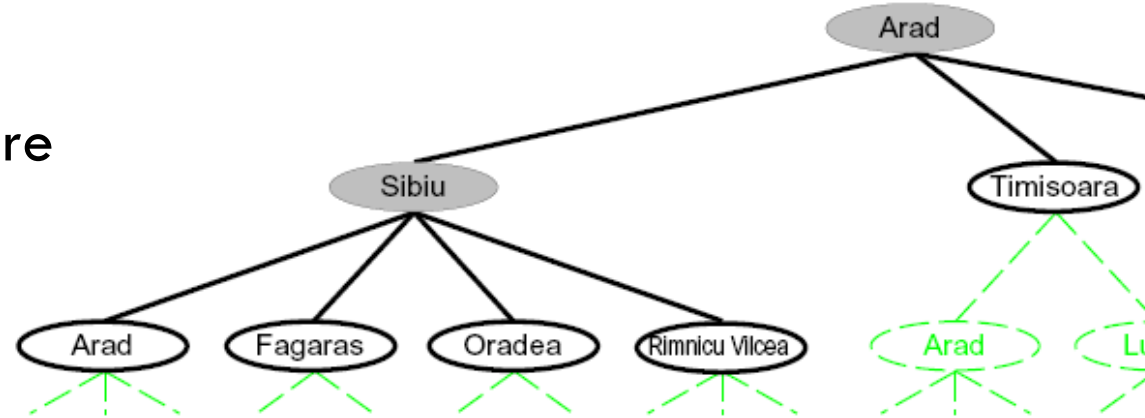
Evaluating Search Algorithms

- Properties of algorithms for computing paths (agent programs):
  - Completeness
    - The algorithm will find a solution if there exists one
  - Soundness
    - If the algorithm returns a solution, it will be a valid path in the graph
  - Space complexity
  - Time complexity
  - Optimality
    - Algorithm returns solutions of *minimal* cost



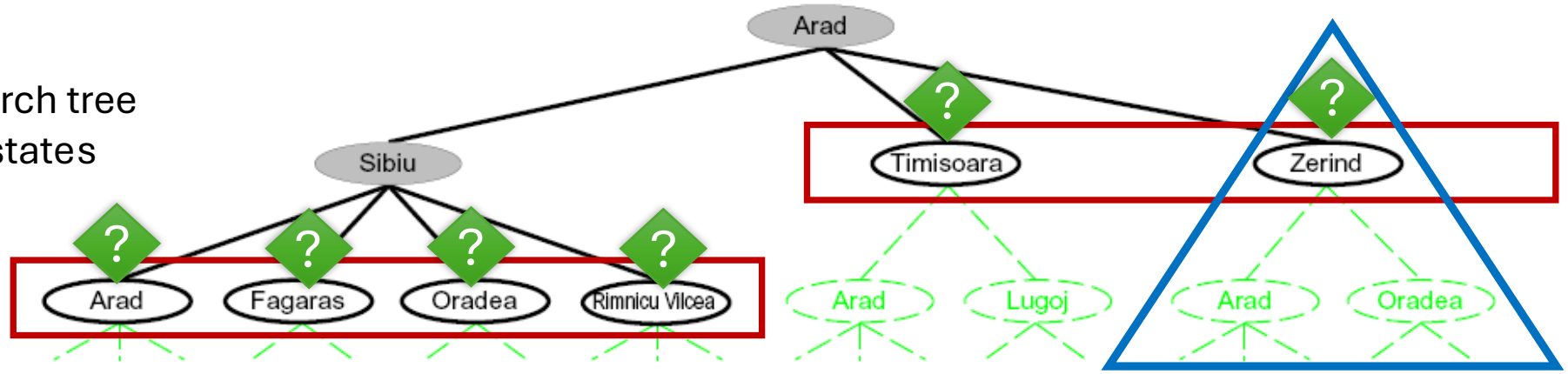
Search Algorithms: Fundamentals

- A search tree is used to evaluate possible courses of action in an offline simulation
  - Use the successor function to incrementally explore neighbors of explored states
  - ➢ "Expand" states
- Node in the search tree vs state
  - A state is a configuration of the agent, environment
  - A node is a data structure used in the search-tree
    - State, parent, children, depth, incurred path cost $g(x)$
    - Multiple nodes may include the same state (not desirable)
    - Each node corresponds to a partial plan
  - The root node includes the start state



General Notions of Search

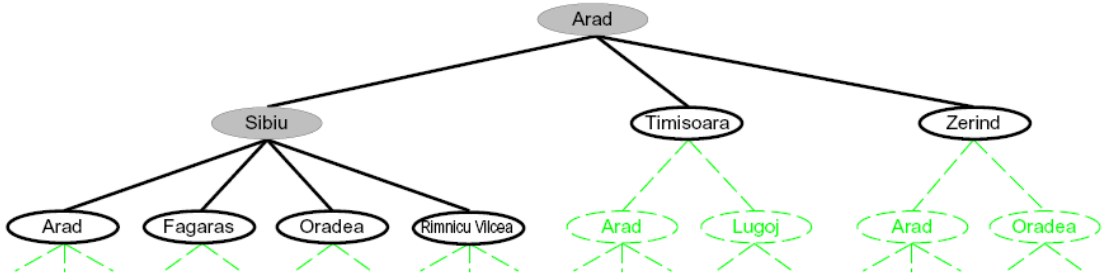
Partial search tree
Nodes vs states



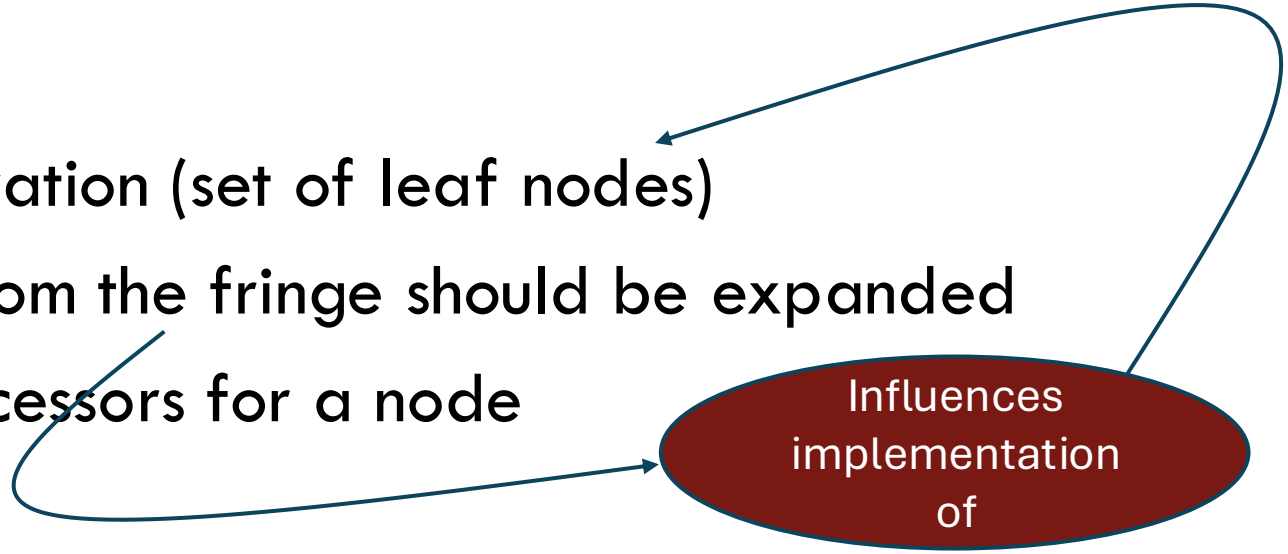
- Fringe: partial plans under consideration
- Exploration strategy: which node from the fringe should be expanded
- Node expansion: generation of successors for a fringe node

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
```

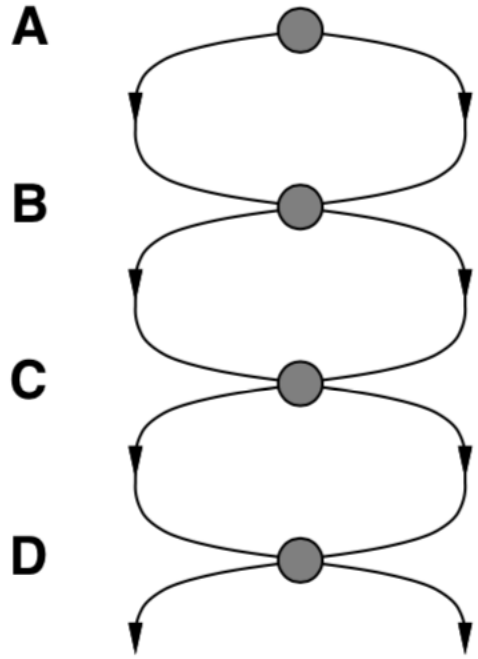


- Fringe: partial plans under consideration (set of leaf nodes)
- Exploration strategy: which node from the fringe should be expanded
- Node expansion: generation of successors for a node



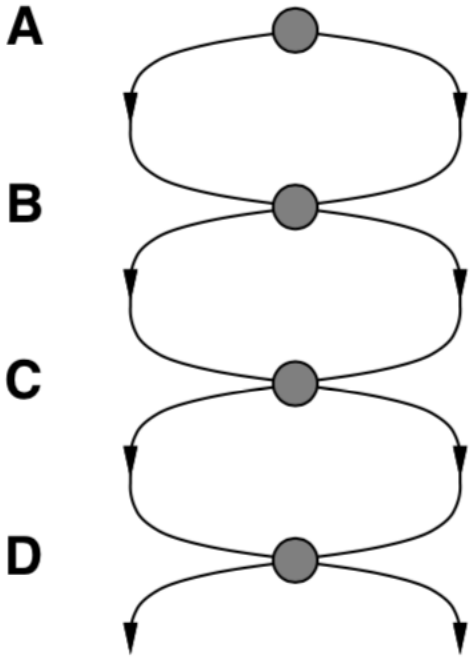
Problem with Tree Search

Doesn't detect repeated states



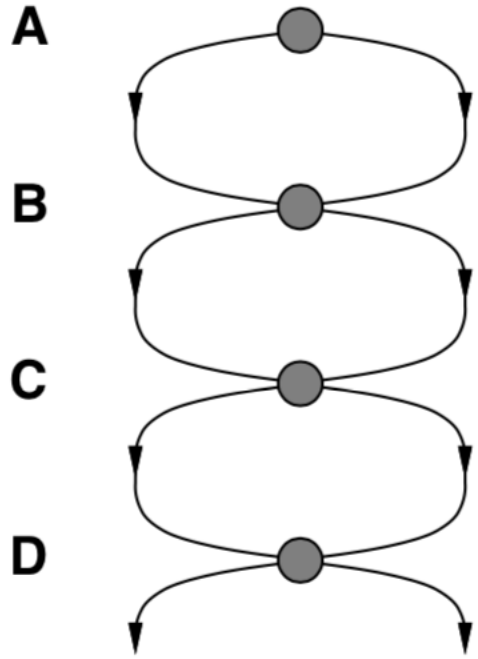
Problem with Tree Search

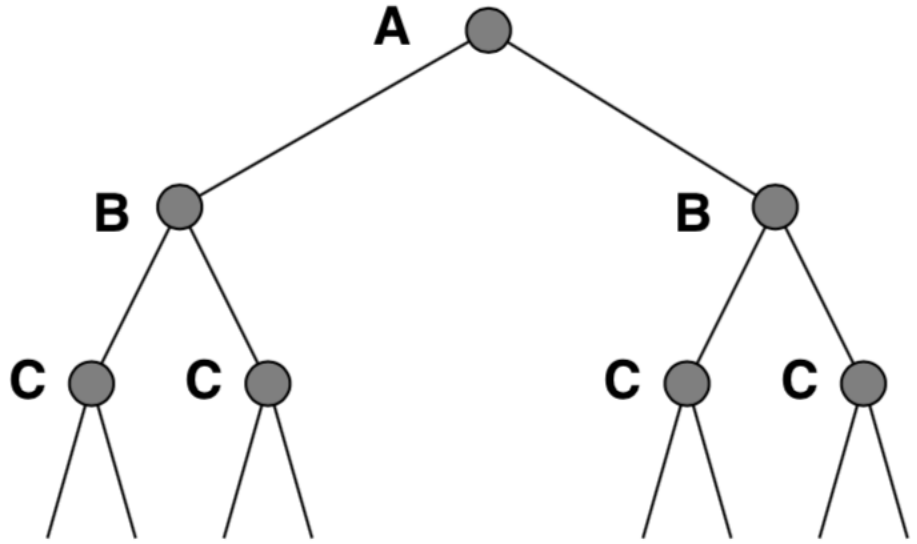
Doesn't detect repeated states



Problem with Tree Search

Doesn't detect repeated states





Easy Fix: Maintain a Closed List

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE(node)) then return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
    end
```

Uniformed Search

- Uninformed, or undirected search algorithms operate without any consideration of how far a state is from a goal
  - Good for situations where distance to a goal cannot be estimated

- Examples:
  - Depth first search
  - Breadth first search
  - Iterative deepening search
  - Uniform cost search

- https://movingai.com/SAS/

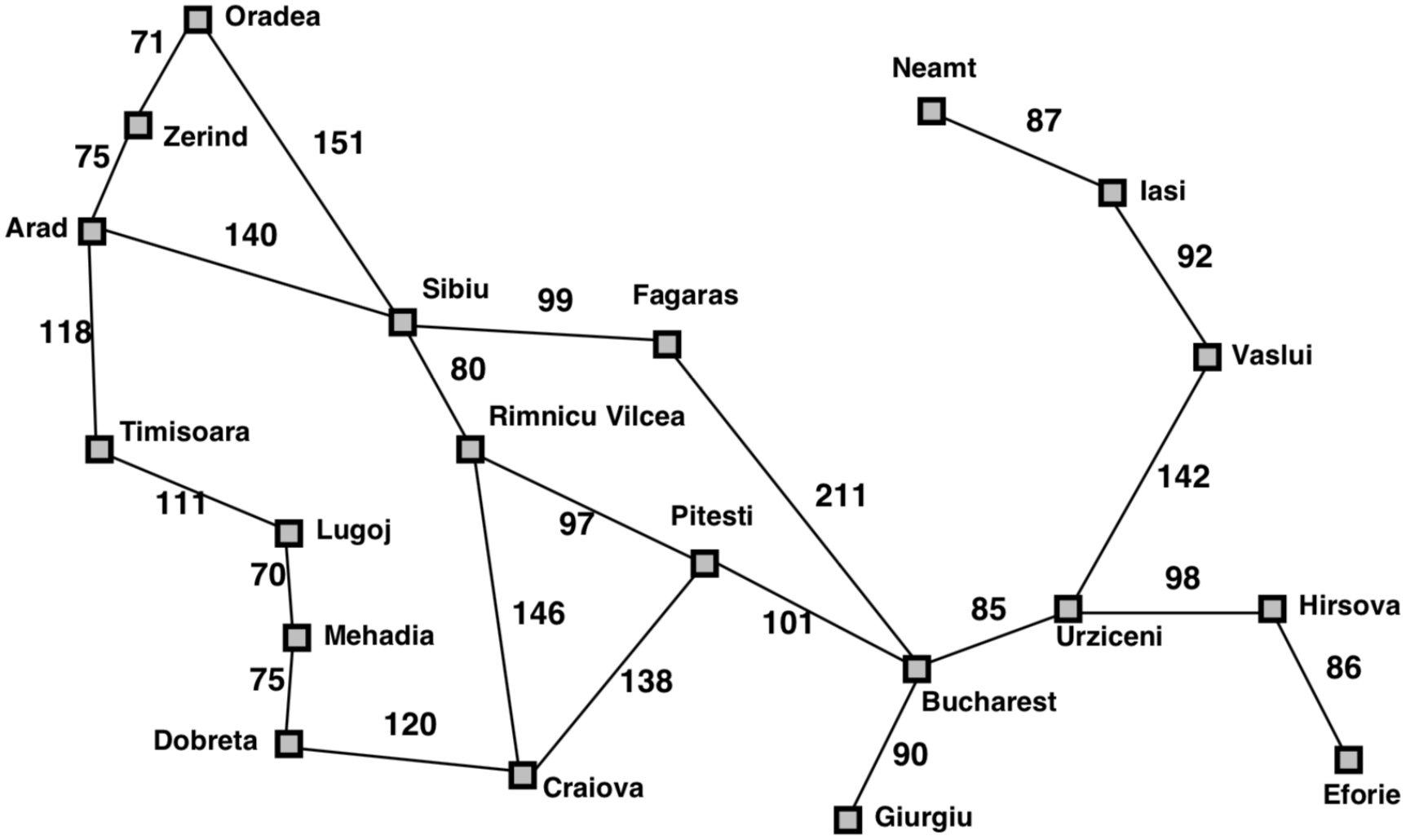
Informed Search

Informed Search

- Can we use goal-cost estimates?
  - This cost estimate is known as a heuristic function
  - Cost estimates amount to measuring the desirability of a state

- Informed search algorithms use the heuristic to guide node expansion
  - Fringe is implemented as a priority queue
  - Key for the queue is computed *using the heuristic function*

Heuristic: Example



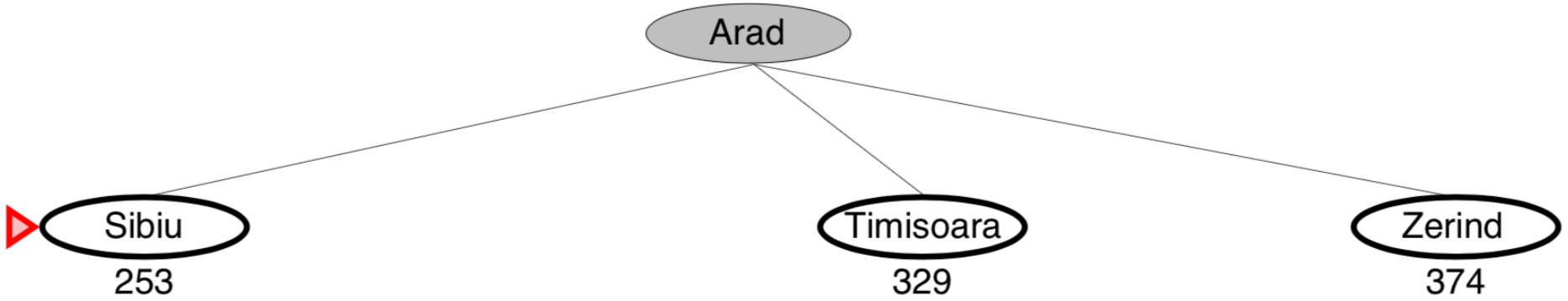
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

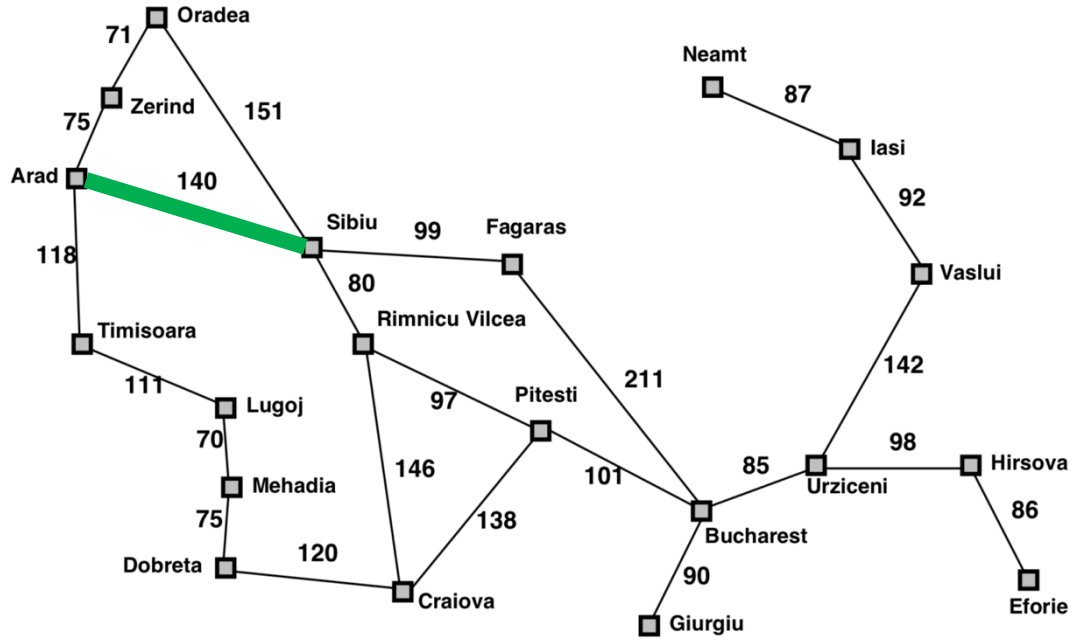
Greedy Best First Search

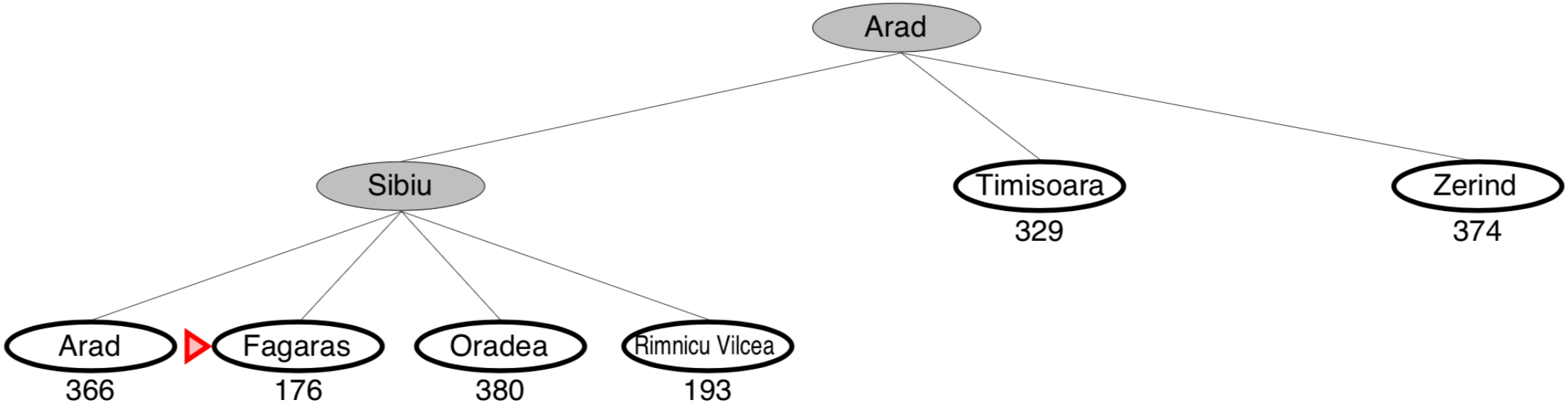
- Implement TreeSearch with following expansion strategy:
  - Expand the node with the least h value
  - ↪ appears to be the closest to the goal

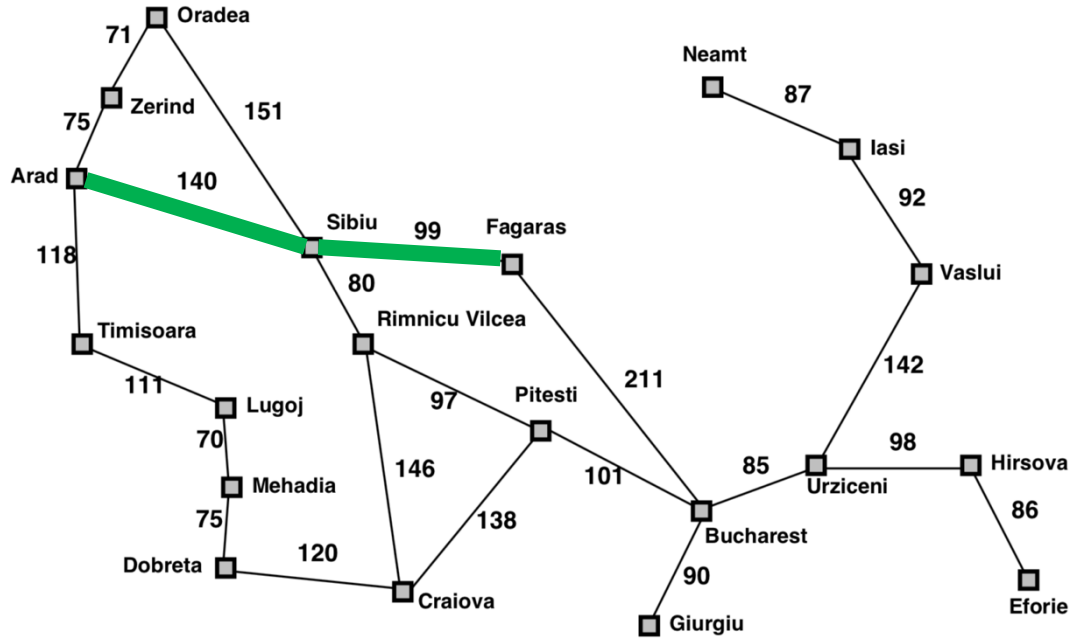
Will this be an optimal algorithm?

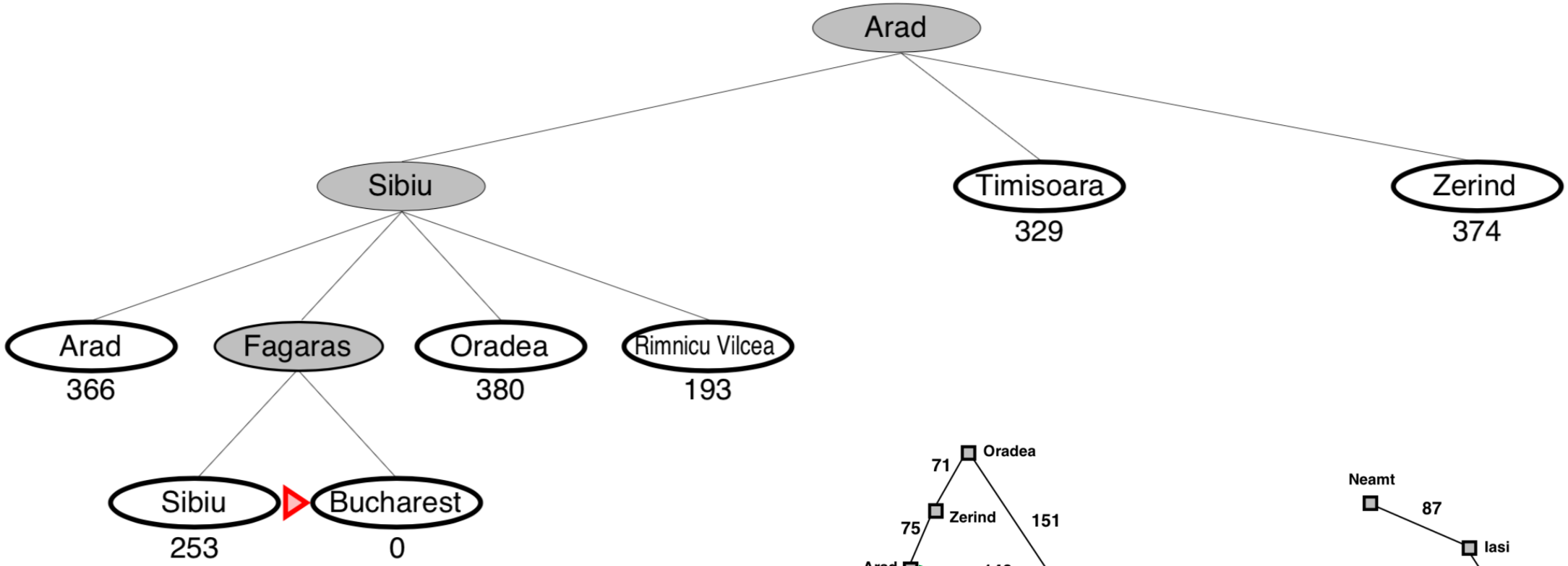

Arad
366

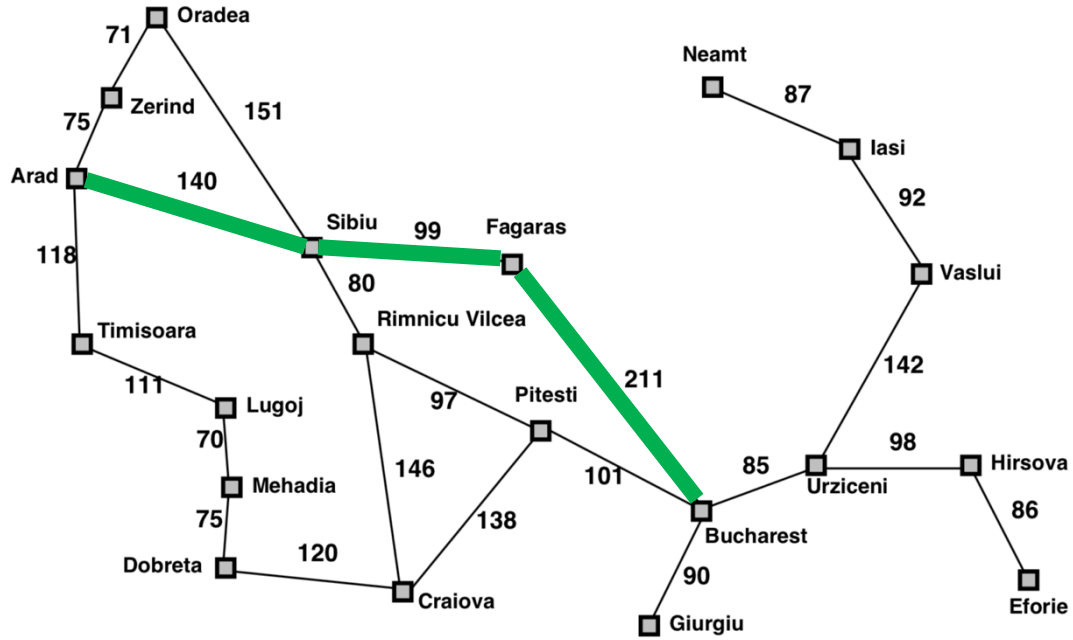

Arad
Sibiu
253
Timisoara
329
Zerind
374


Oradea
71
Neamt
87
Zerind
151
75
Iasi
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
211
111
97
Pitesti
Lugoj
98
70
146
85
Hirsova
Mehadia
101
Urziceni
75
138
86
Bucharest
Dobreta
120
90
Craiova
Eforie
Giurgiu


Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
176
Oradea
380
Rimnicu Vilcea
193


Oradea
71
Neamt
87
Zerind
151
75
Iasi
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
211
97
Pitesti
Lugoj
98
70
85
Hirsova
146
101
Mehadia
Urziceni
86
75
138
Bucharest
Dobreta
120
90
Craiova
Eforie
Giurgiu


Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
Oradea
380
Rimnicu Vilcea
193
Sibiu
253
Bucharest
0


Oradea
71
Neamt
87
Zerind
75
151
Iasi
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
211
Lugoj
97
Pitesti
70
98
Hirsova
146
85
Mehadia
101
Urziceni
75
138
86
Bucharest
Dobreta
120
90
Craiova
Eforie
Giurgiu

Greedy Best First Search Properties

- Complete in finite graphs with closed list
- Time
- Space
- Optimal?

Greedy Best First Search Properties

- Complete in finite graphs with closed list
- Time $O(b^m)$
- Space $O(b^m)$
- Optimal? No

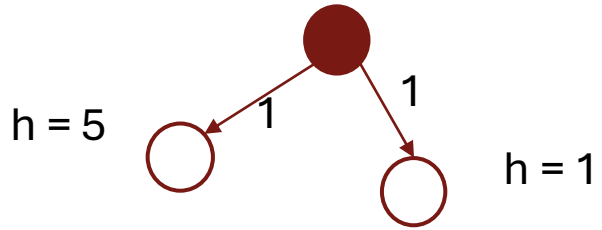
What if the heuristic is not very good?

Greedy Search: Negative Result

- In the worst case, greedy search can expand all other nodes in the graph (even those that are arbitrarily further than the goal) before reaching the goal

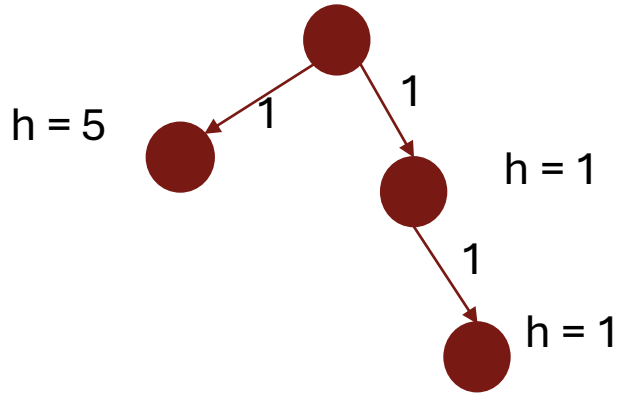
Greedy Search: Negative Result

- In the worst case, greedy search can expand all other nodes in the graph (even those that are arbitrarily further than the goal) before reaching the goal



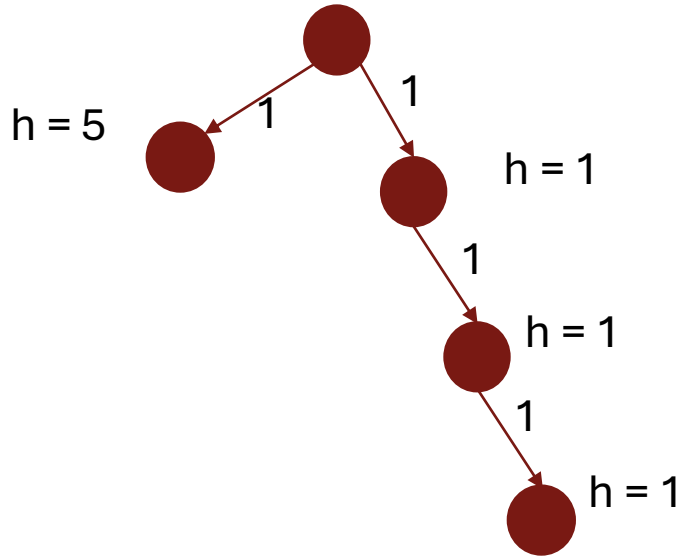
Greedy Search: Negative Result

- In the worst case, greedy search can expand all other nodes in the graph (even those that are arbitrarily further than the goal) before reaching the goal



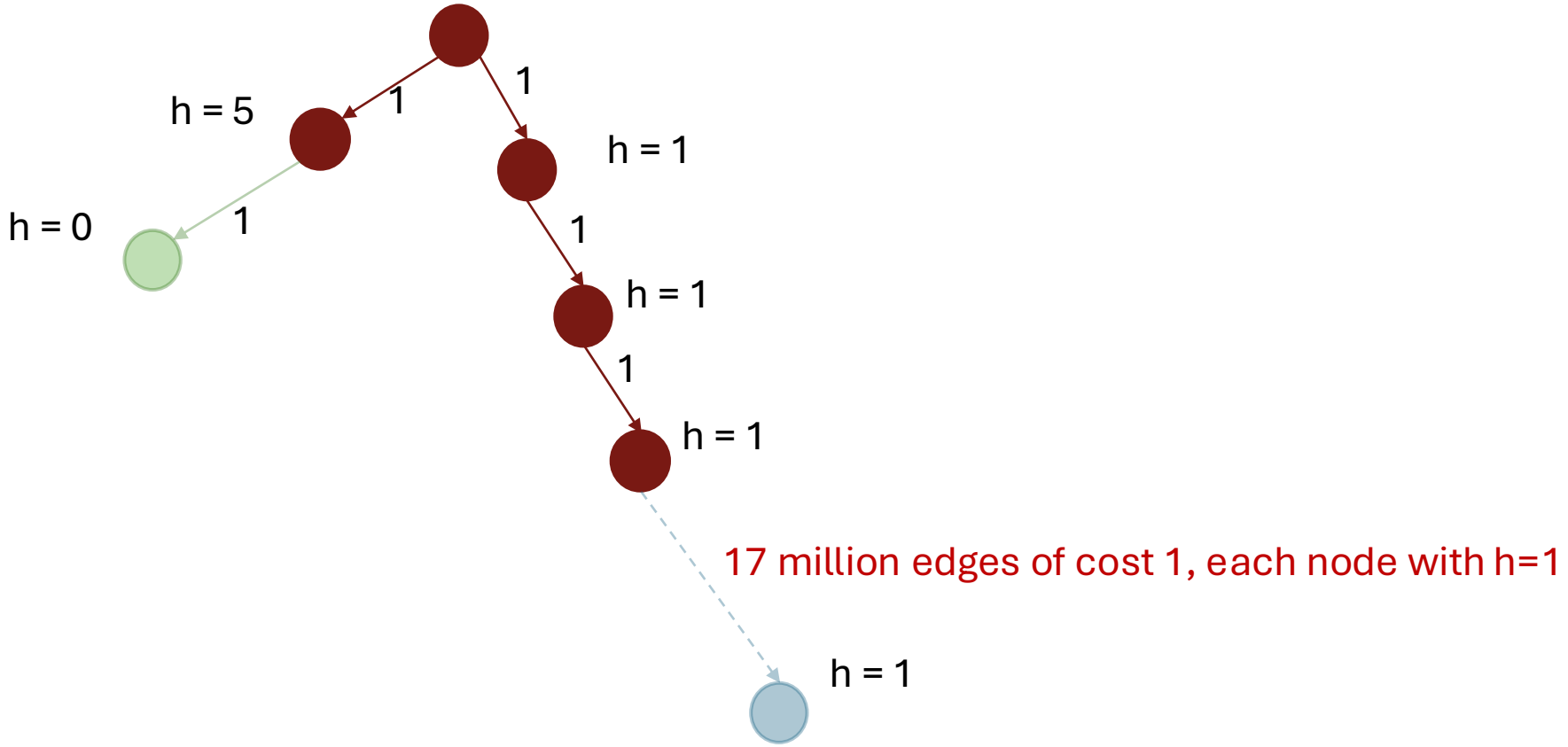
Greedy Search: Negative Result

- In the worst case, greedy search can expand all other nodes in the graph (even those that are arbitrarily further than the goal) before reaching the goal



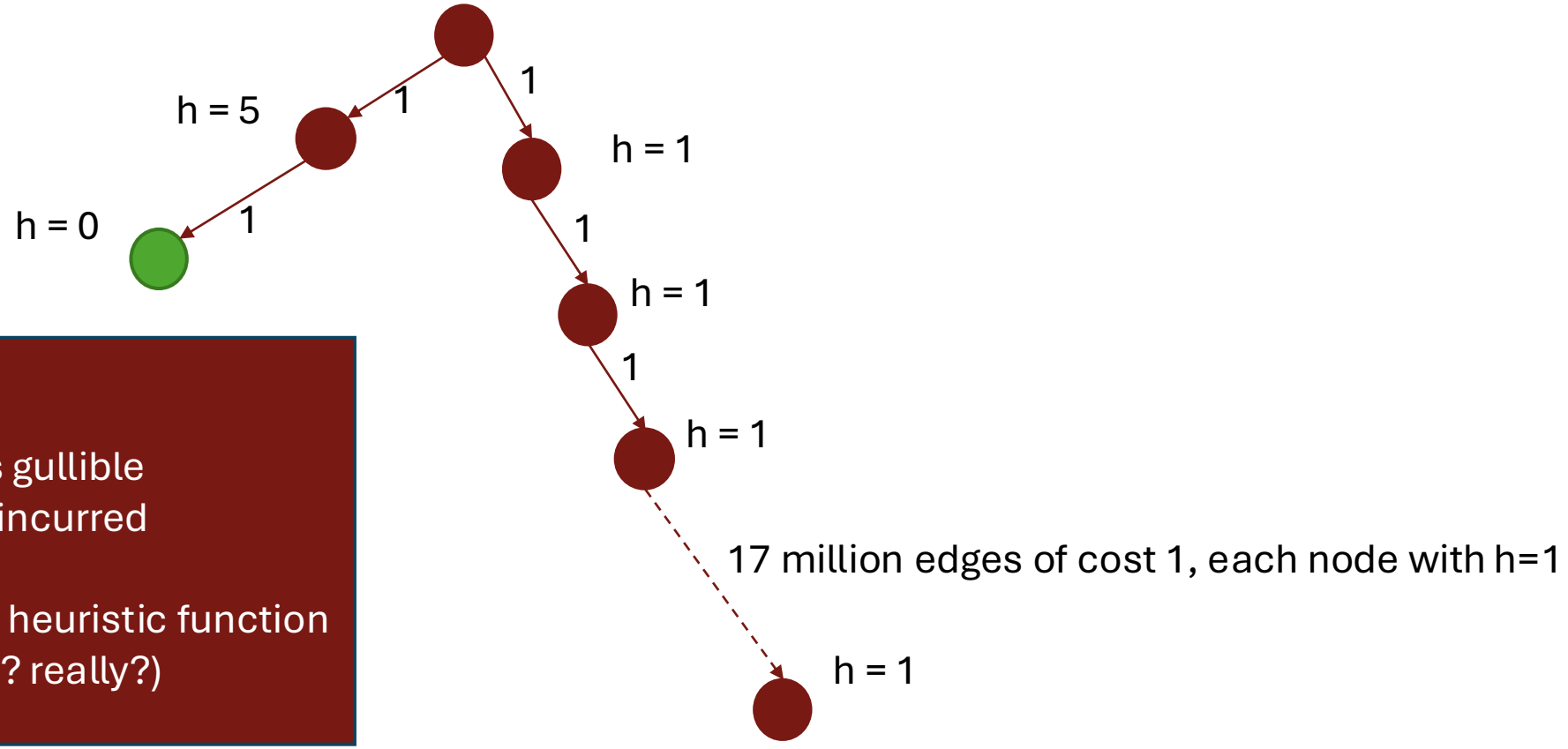
Greedy Search: Negative Result

- In the worst case, greedy search can expand all other nodes in the graph (even those that are arbitrarily further than the goal) before reaching the goal



Greedy Search: Negative Result

- In the worst case, greedy search can expand all other nodes in the graph (even those that are arbitrarily further than the goal) before reaching the goal

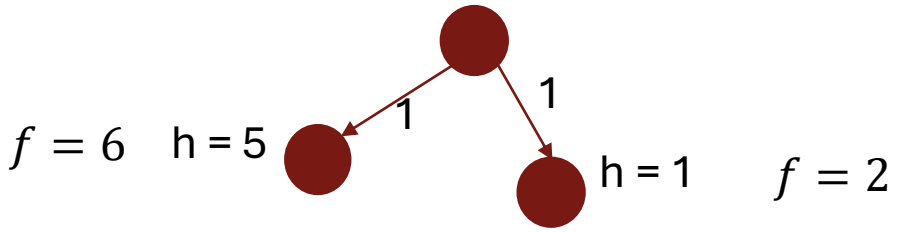


Oops!

- Algorithm could be made less gullible
  - Pay attention to the cost incurred
    ~ gambler's ruin
- Our stated requirements for a heuristic function were too loose (anything goes? really?)

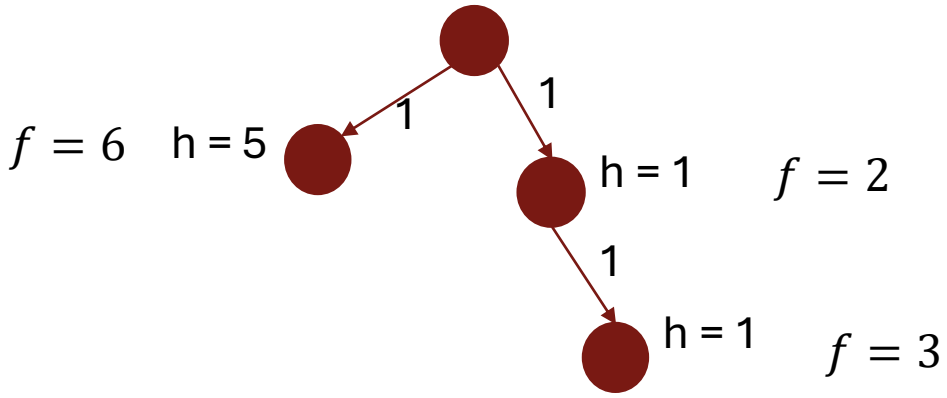
A* Search

- A* directly addresses those limitations
- $g(n)$ = cost required to get to the node from the start
- $h(n)$ = estimated cost to the goal from n (heuristic, as before)
- $f(n) = g(n) + h(n)$
- Expansion strategy:
  - expand the node with the least f value



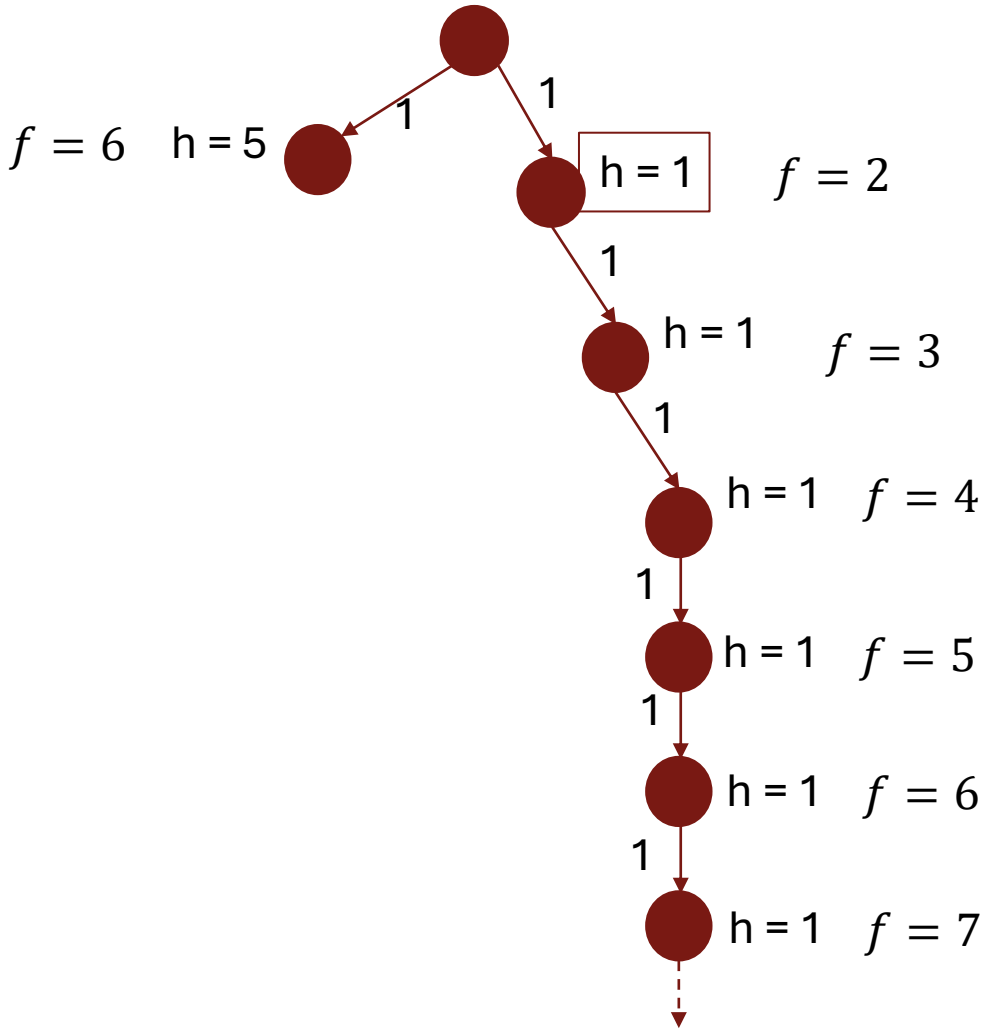
A* Search

- A* directly addresses those limitations
- $g(n)$ = cost required to get to the node from the start
- $h(n)$ = estimated cost to the goal from n (heuristic, as before)
- $f(n) = g(n) + h(n)$
- Expansion strategy:
  - expand the node with the least f value



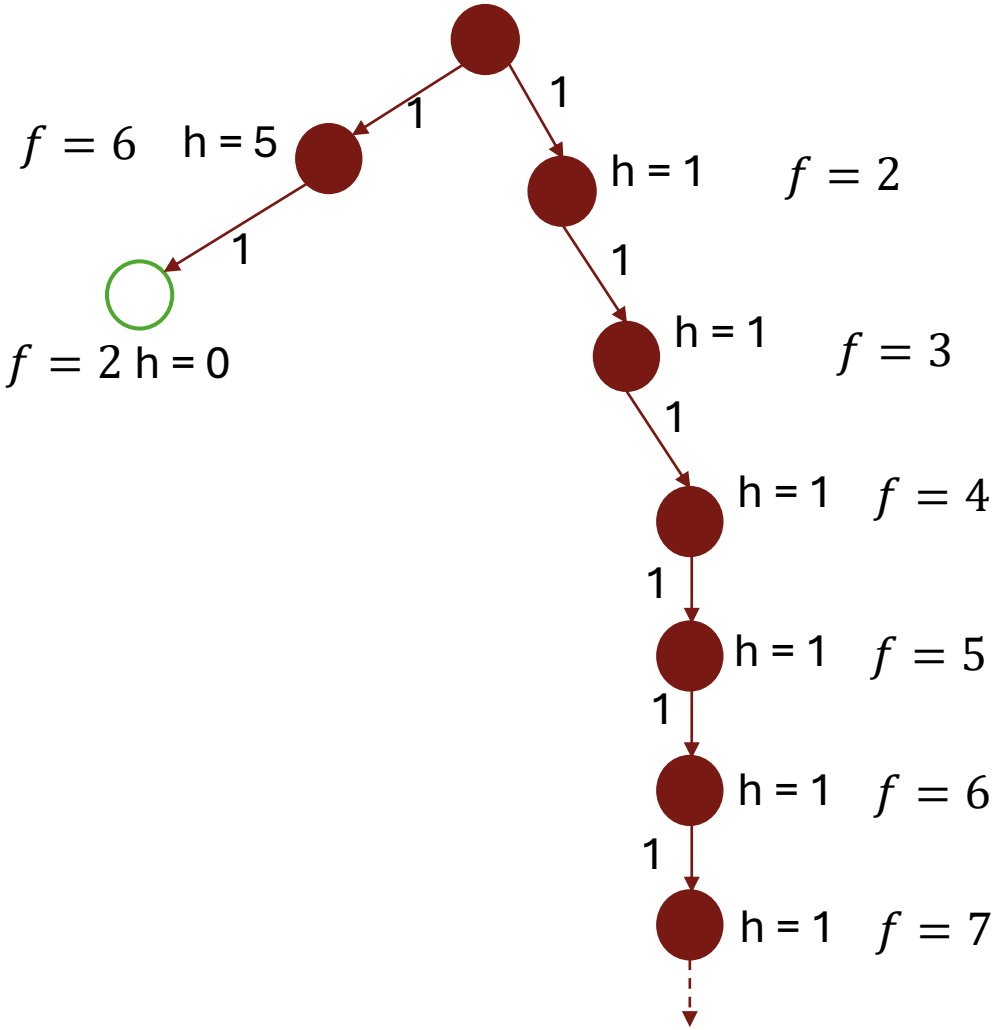
A* Search

- A* directly addresses those limitations
- $g(n)$ = cost required to get to the node from the start
- $h(n)$ = estimated cost to the goal from n (heuristic, as before)
- $f(n) = g(n) + h(n)$
- Expansion strategy:
  - expand the node with the least f value



A* Search

- A* directly addresses those limitations
- $g(n)$ = cost required to get to the node from the start
- $h(n)$ = estimated cost to the goal from n (heuristic, as before)
- $f(n) = g(n) + h(n)$
- Expansion strategy:
  - expand the node with the least f value



A* Search

- A* directly addresses those limitations
- $g(n)$ = cost required to get to the node from the start
- $h(n)$ = estimated cost to the goal from n (heuristic, as before)
- $f(n) = g(n) + h(n)$
- Expansion strategy:
  - expand the node with the least f value

Not immune to bad h
But won't get fooled forever