Lecture 5

CS 6380 Artificial Intelligence

Informed Search

Department of Computer Science and Engineering

Indian Institute of Technology Madras (IITM), India

Some material based on lecture slides for CSE 571: Artificial Intelligence by Siddharth Srivastava, ASU.

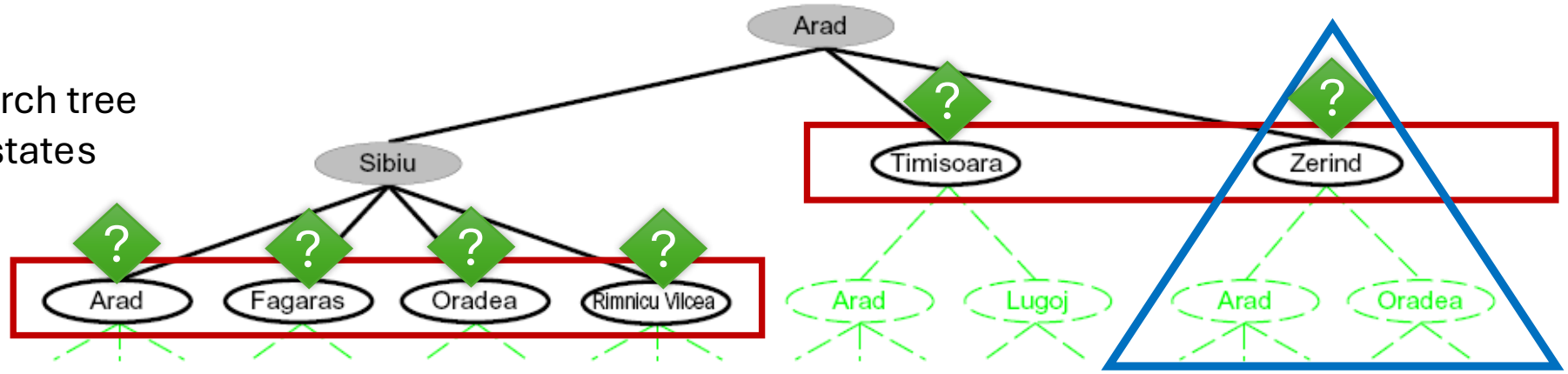
Logistics and Recap

In-class and HW Quizzes

- Evaluation through gradescope.com
- You should've received an enrollment email.

General Notions of Search

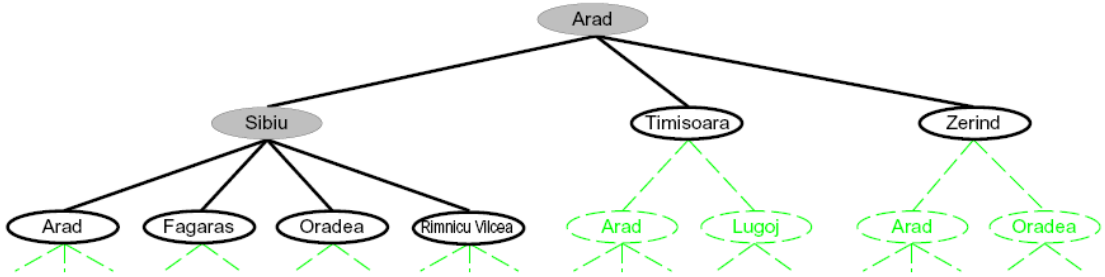
Partial search tree
Nodes vs states



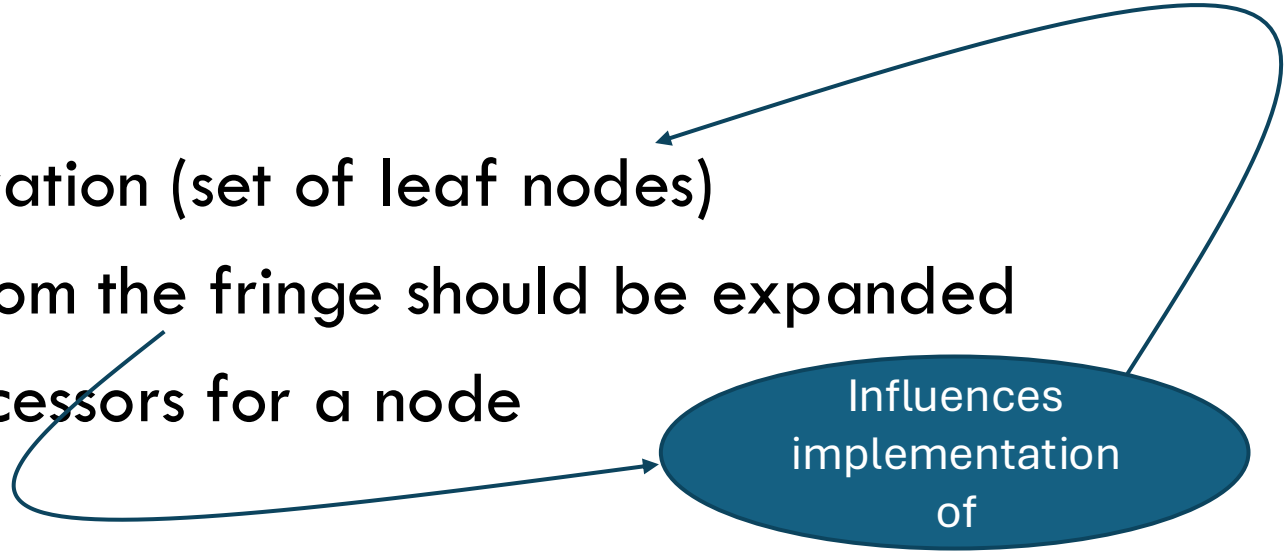
- Fringe: partial plans under consideration
- Exploration strategy: which node from the fringe should be expanded
- Node expansion: generation of successors for a fringe node

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

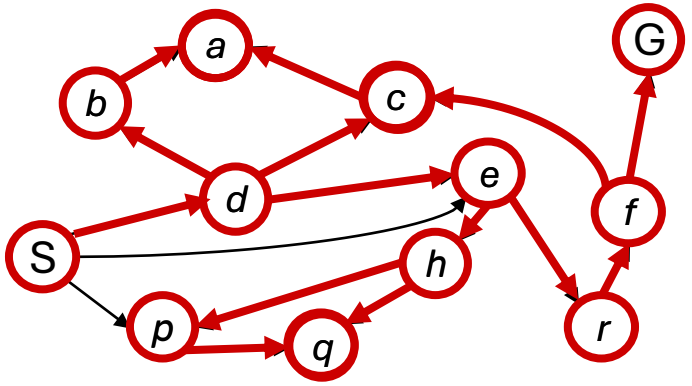


Depth-First Search

Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack

(successors go on top)



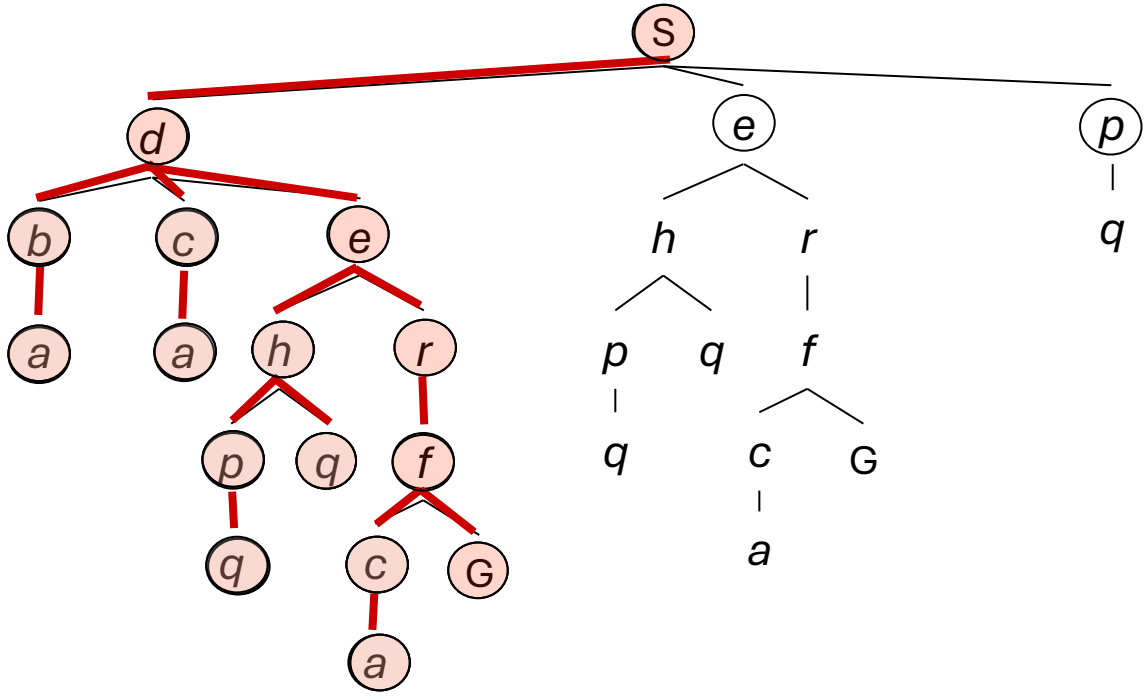


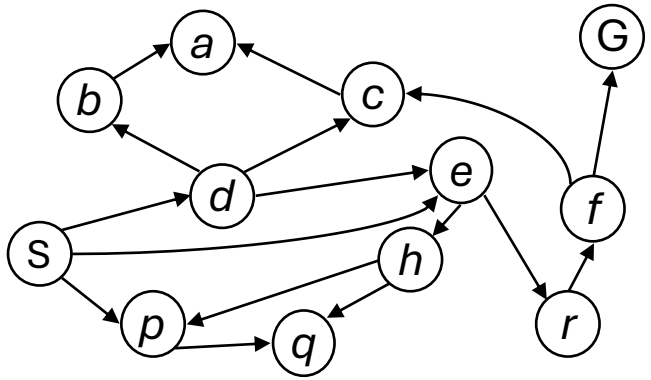
Illustration courtesy Abeel and Kleir

Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue

(successors go at end)



Search Tiers

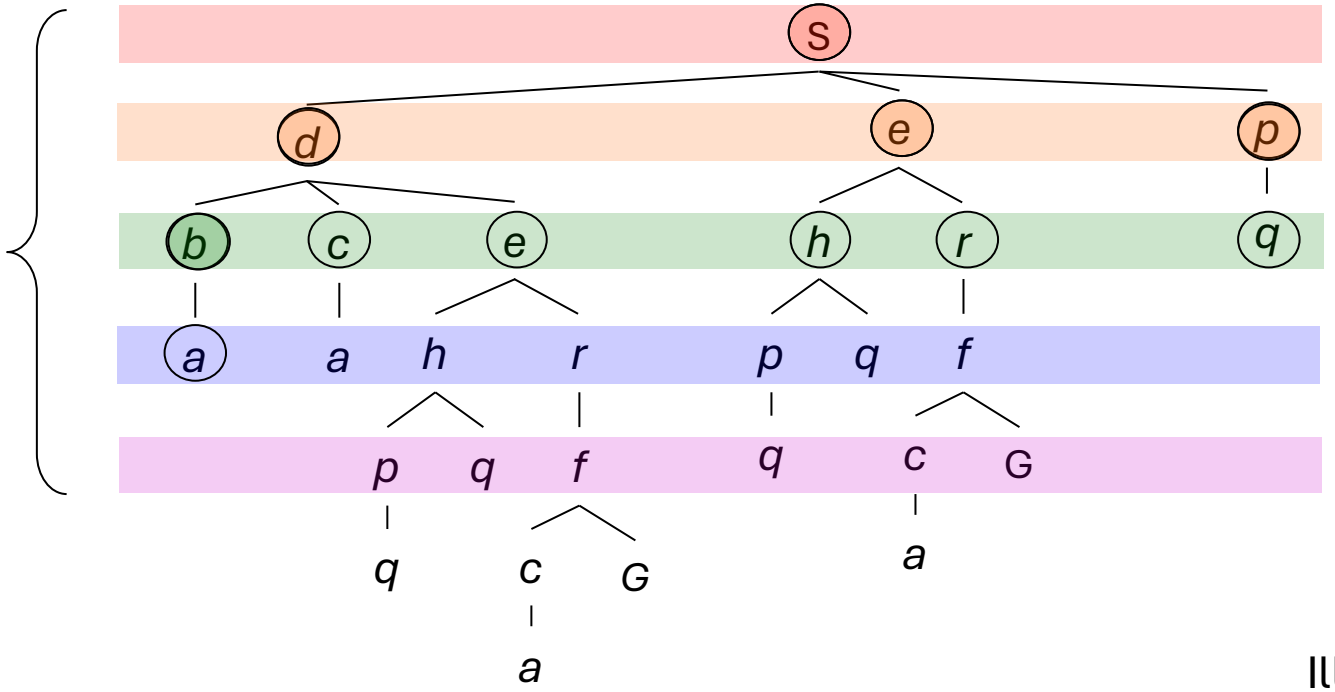


Illustration courtesy Abeel and Kleir

Iterative Deepening

- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - Run DFS with depth limit 1. If no solution…
  - Run DFS with depth limit 2. If no solution…
  - Run DFS with depth limit 3. …..
- Why is this a good idea?
- BFS generates successor nodes at the next level (can be fixed: how?)
- Space usage $O(bs)$ vs $O(b^s)$

$$b = 10, m = 5$$

$$N(IDS) = 5 \cdot 10 + 4 \cdot 100 + 3 \cdot 10^3 + 2 \cdot 10^4 + 10^5$$

$$N(BFS) = 10 + 100 + 10^3 + 10^4 + 10^5 + 999{,}990$$

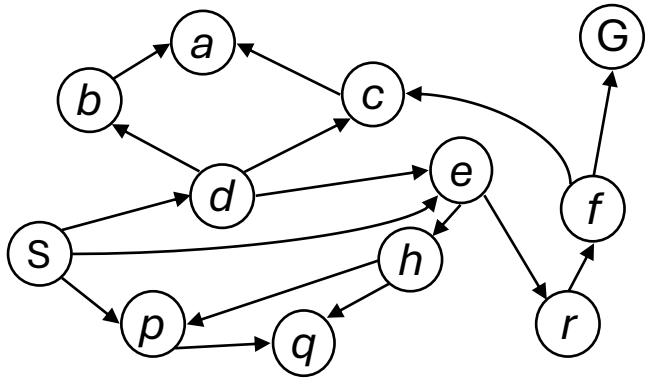
$$(s+1) \cdot 1 + s \cdot b + (s-1) \cdot b^2 + \cdots + 1 \cdot b^s \sim O(b^s)$$

Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue

(successors go at end)



Search Tiers

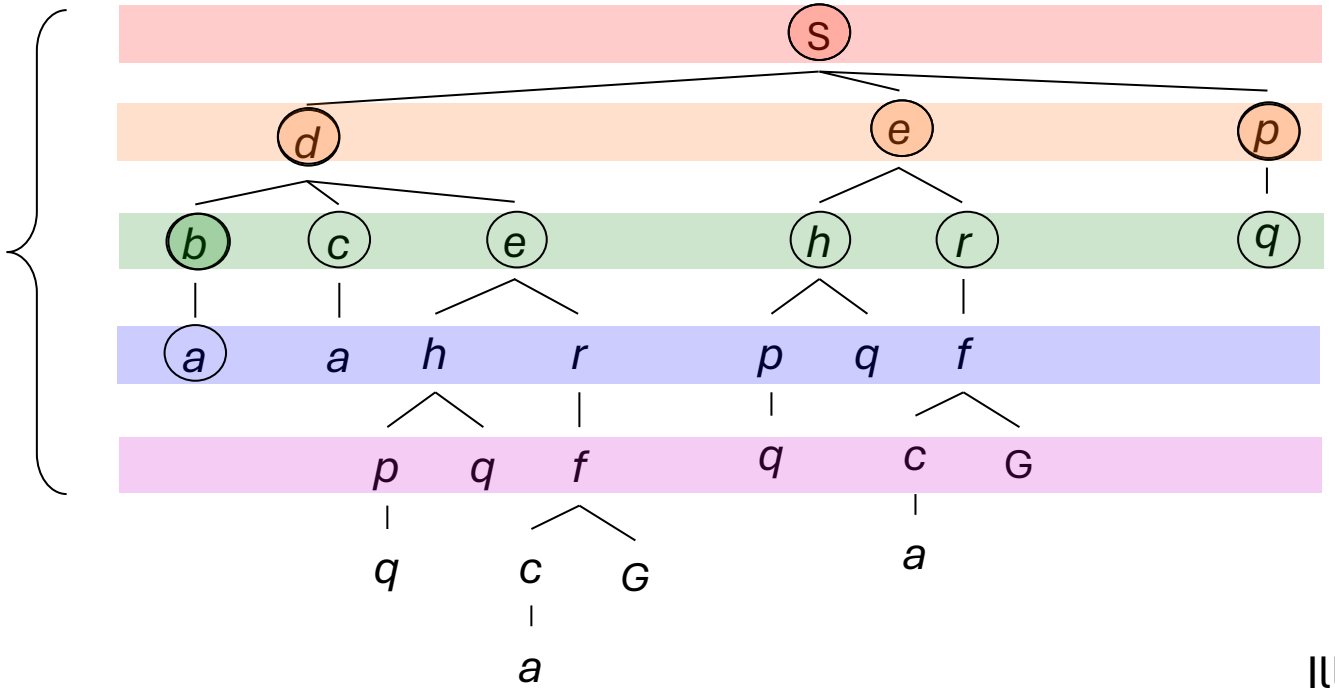
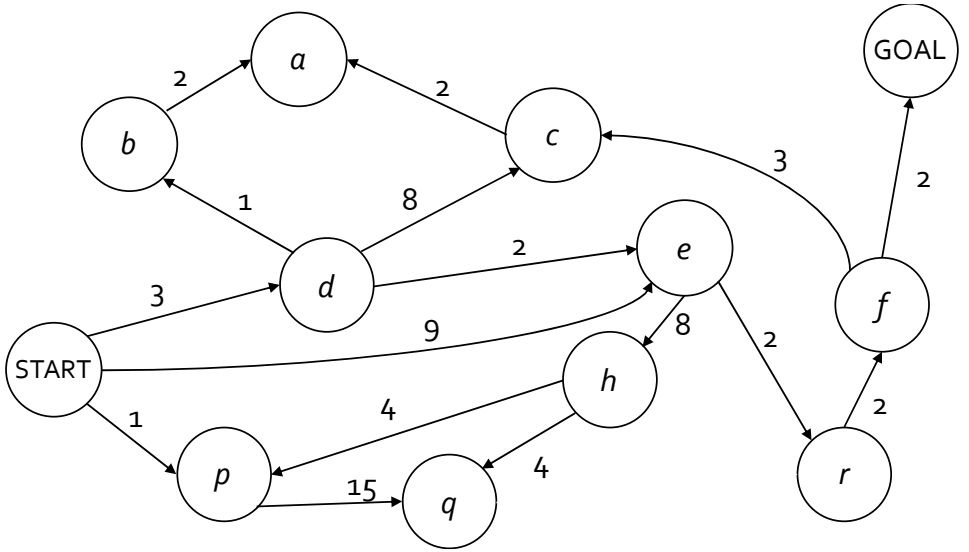


Illustration courtesy Abeel and Kleir

Cost-Sensitive Search

- BFS expands the graph “uniformly” in terms of depth
- All nodes at a lower depth are expanded before nodes at a greater depth
- Often, actions (edges) have heterogeneous costs, so expanding lower depths may yield expensive solutions
- Can BFS be extended to expand lower-cost nodes first?

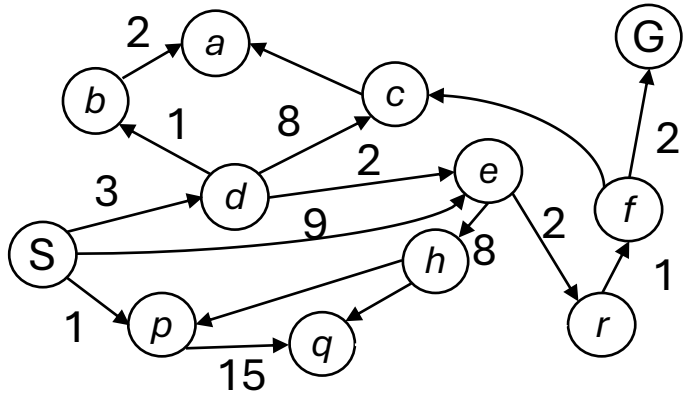
Yes!



Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)



Cost contours

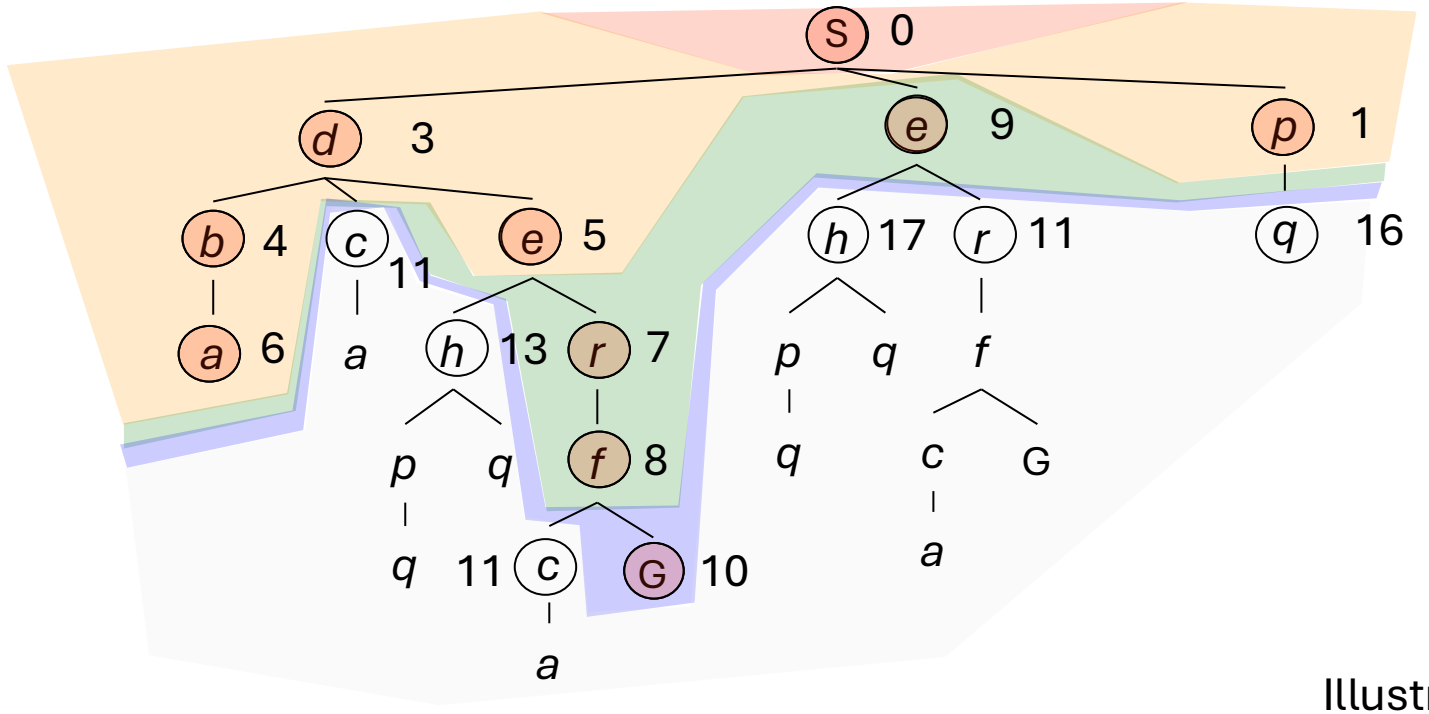


Illustration courtesy Abeel and Kleir

Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)
- How much space does the fringe take?
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$
- Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!
- Is it optimal?
  - Yes! (Proof next lecture via A*)

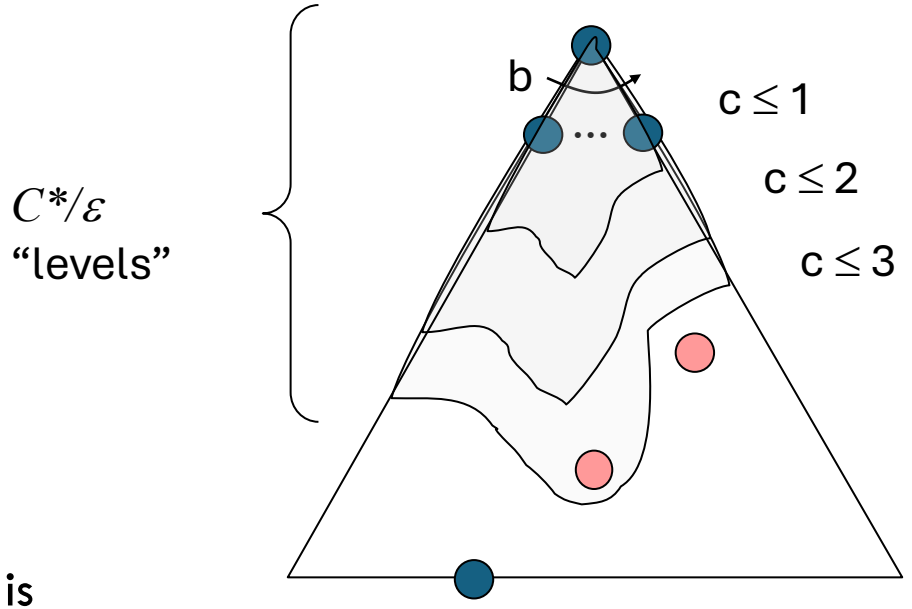
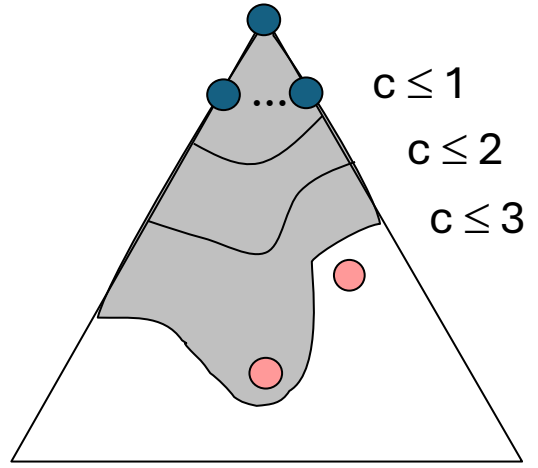


Illustration courtesy Abeel and Kleir

Uniform Cost Issues

- UCS is complete and optimal!
  - Why?
- However, UCS inherits disadvantages of undirected search like all other algorithms discussed today
- Not goal directed!



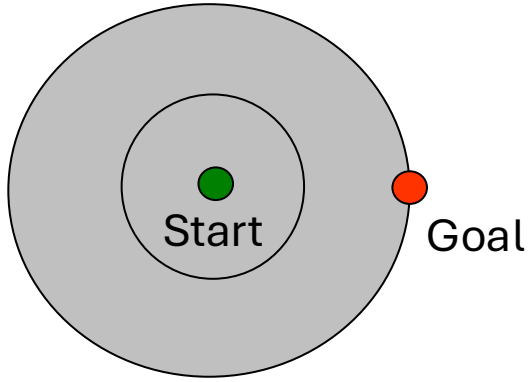


Illustration courtesy Abeel and Kleir

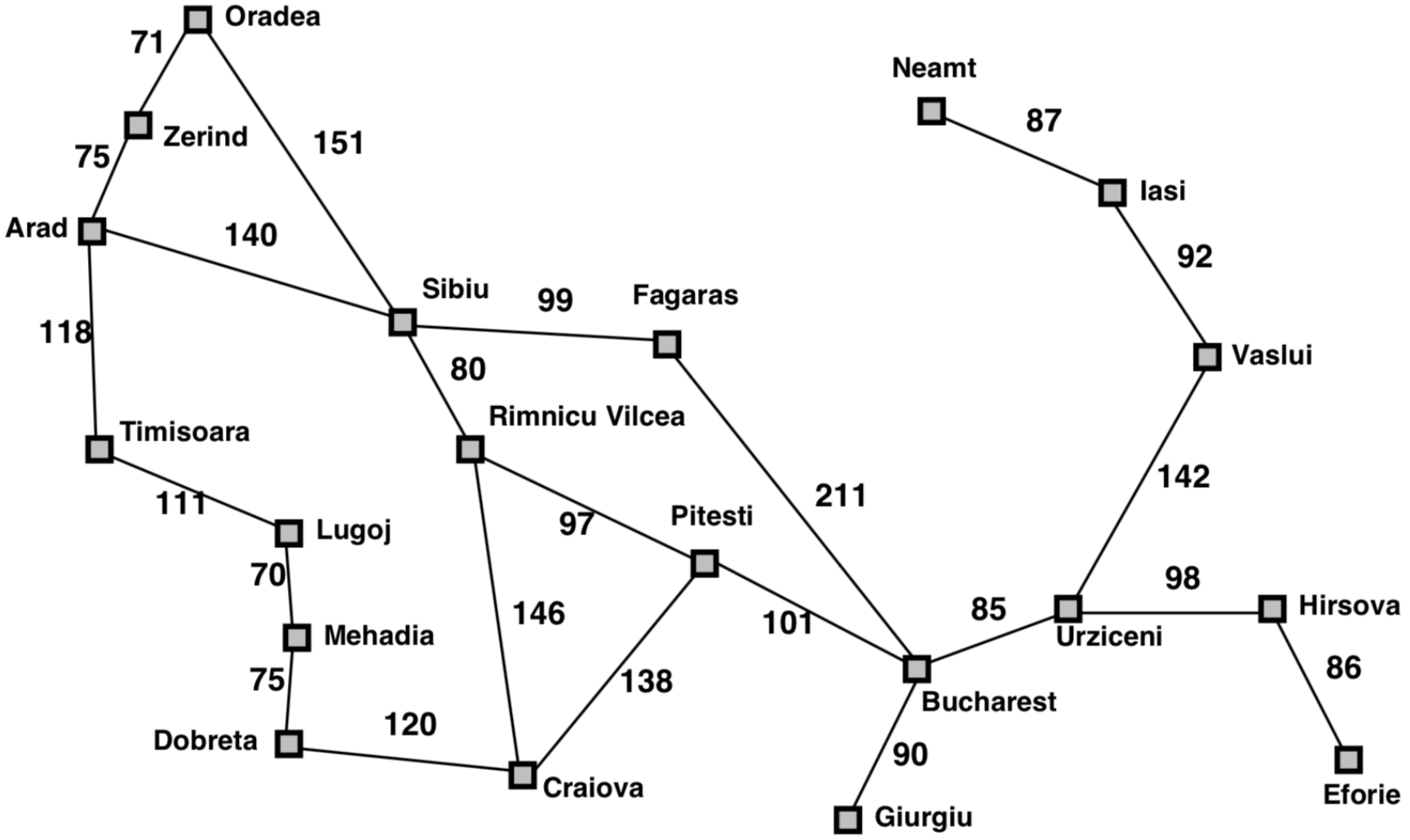
Informed Search

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



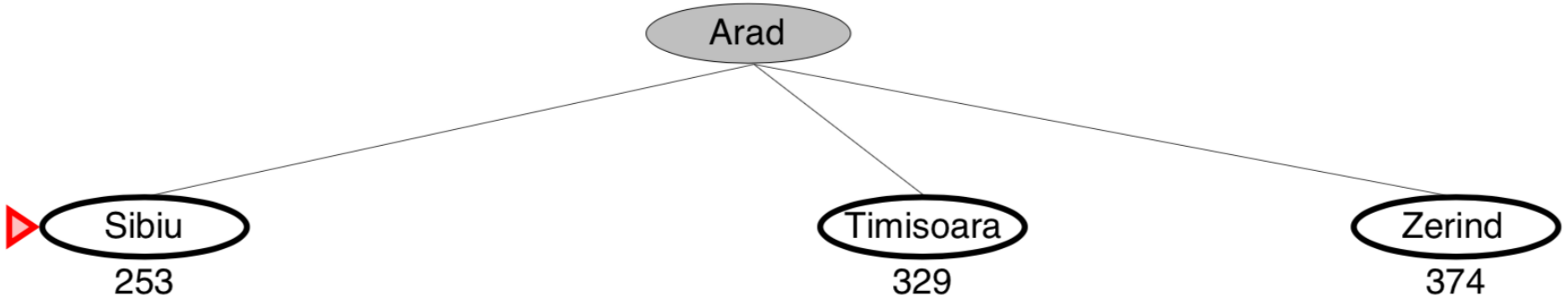
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

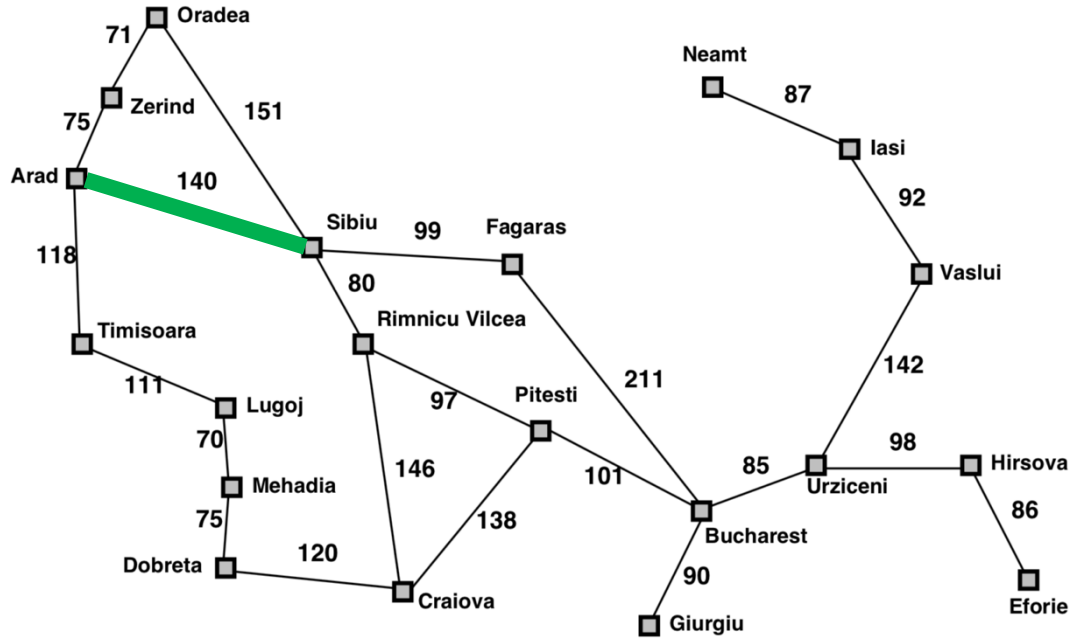
Greedy Best First Search

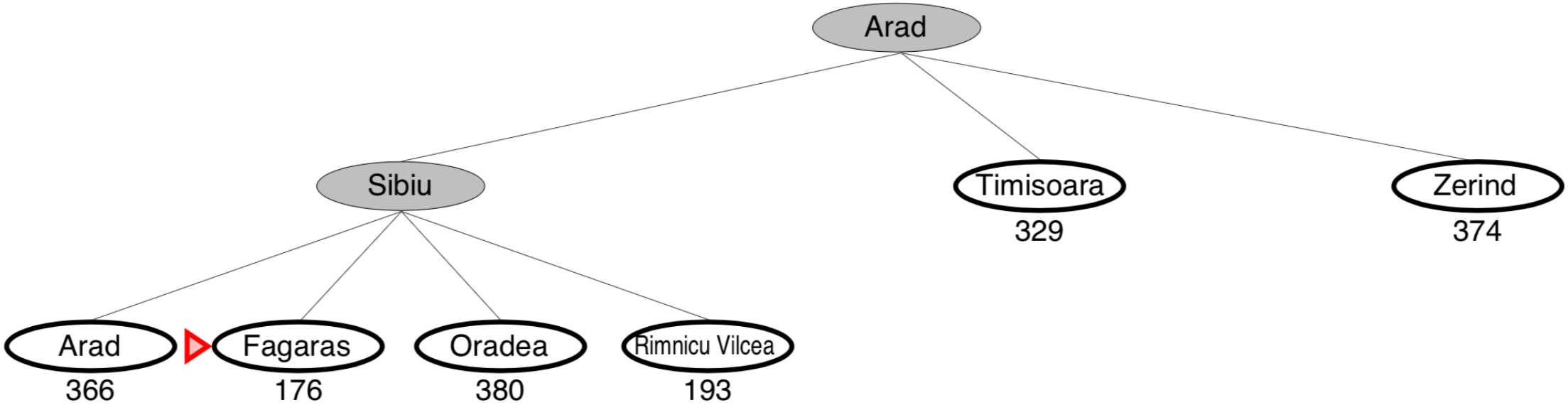
- Implement TreeSearch with following expansion strategy:
  - Expand the node with the least h value
  - ↪ appears to be the closest to the goal

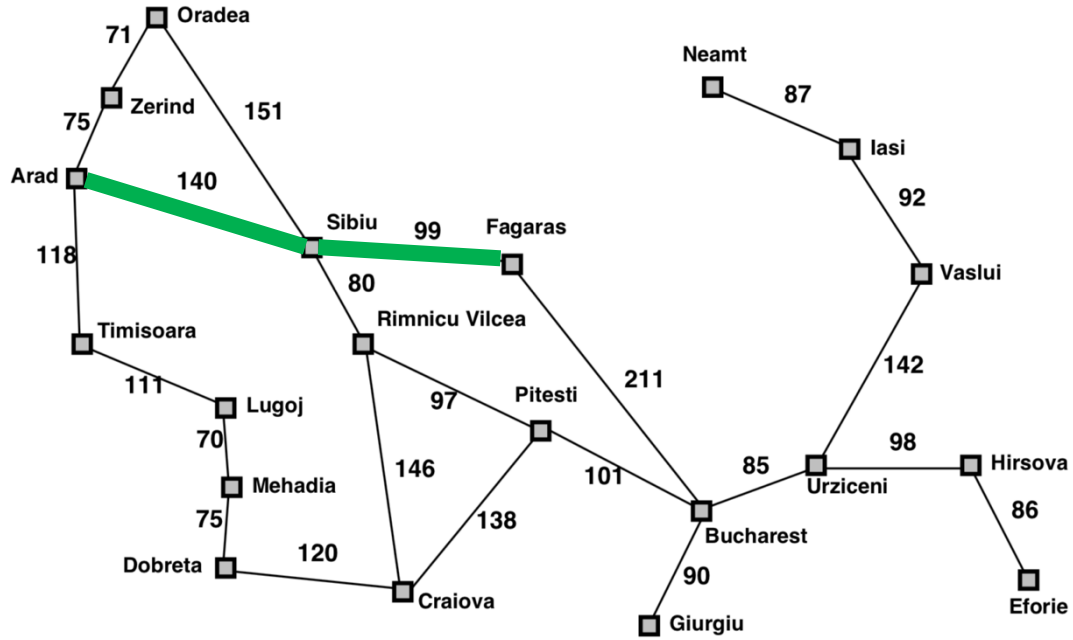
Will this be an optimal algorithm?

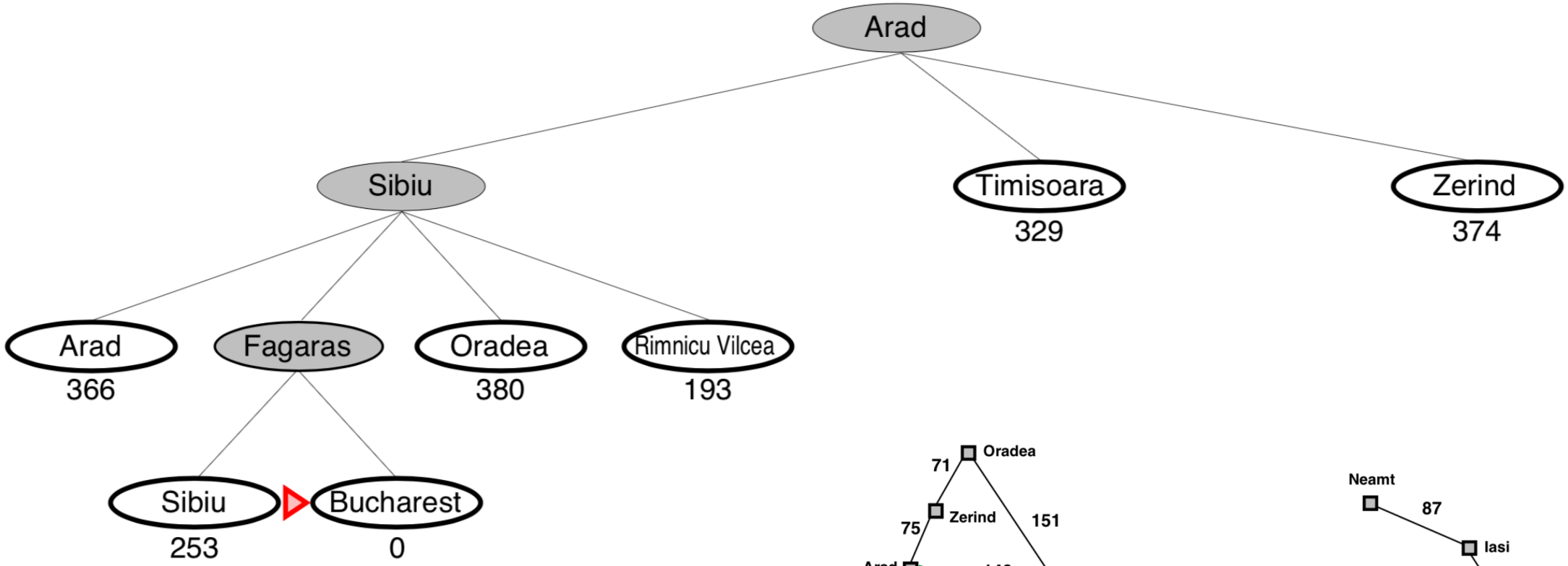

Arad
366

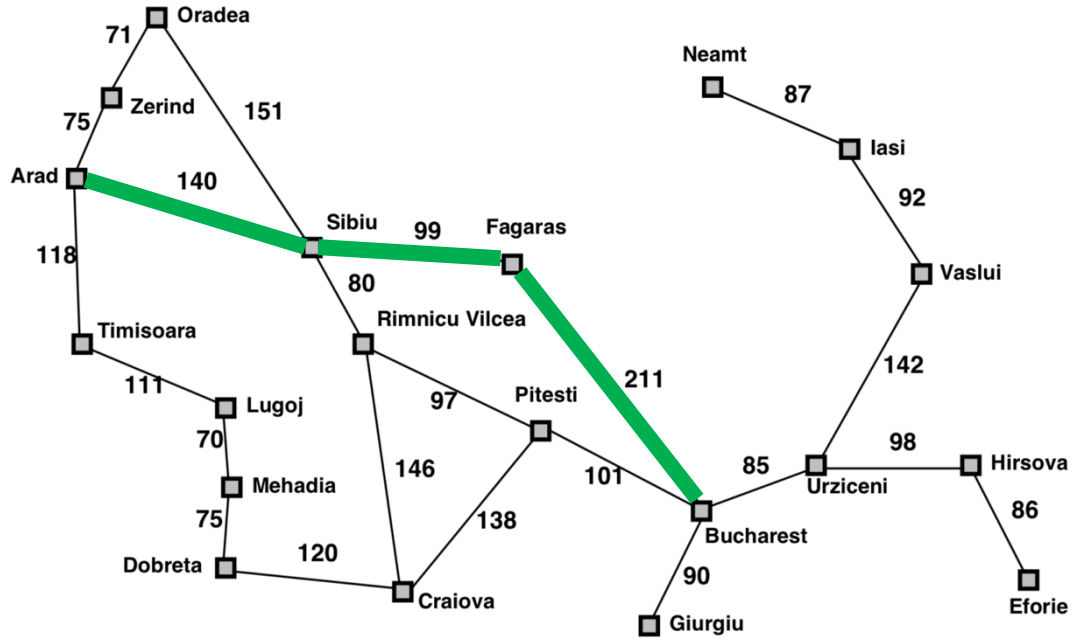

Arad
Sibiu
253
Timisoara
329
Zerind
374


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

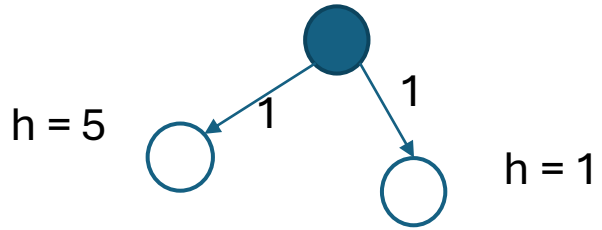
What if the heuristic is not very good?

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

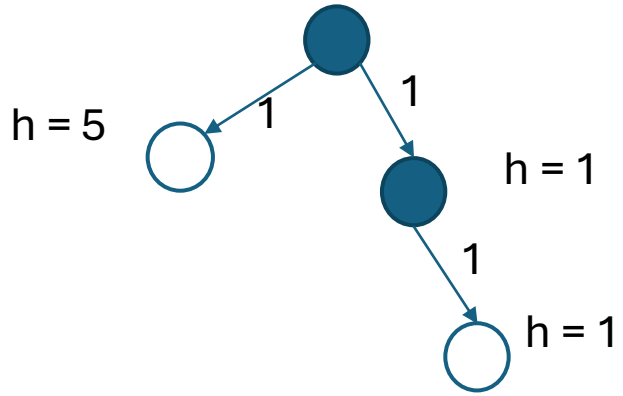
Greedy Search: Negative Result

- 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



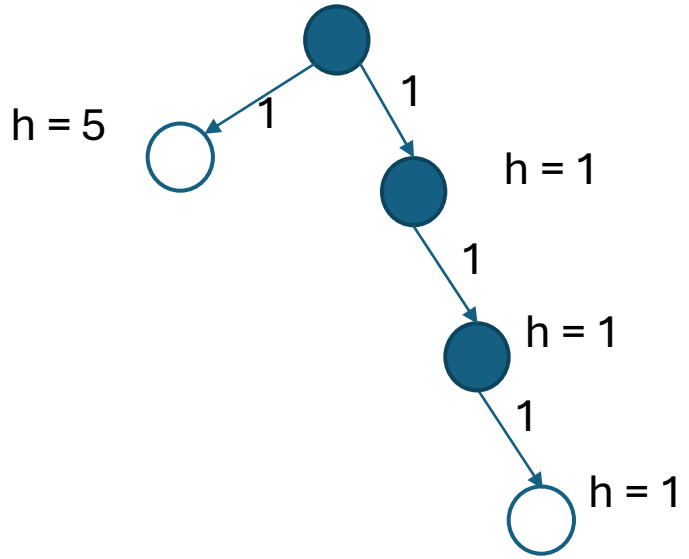
Greedy Search: Negative Result

- 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



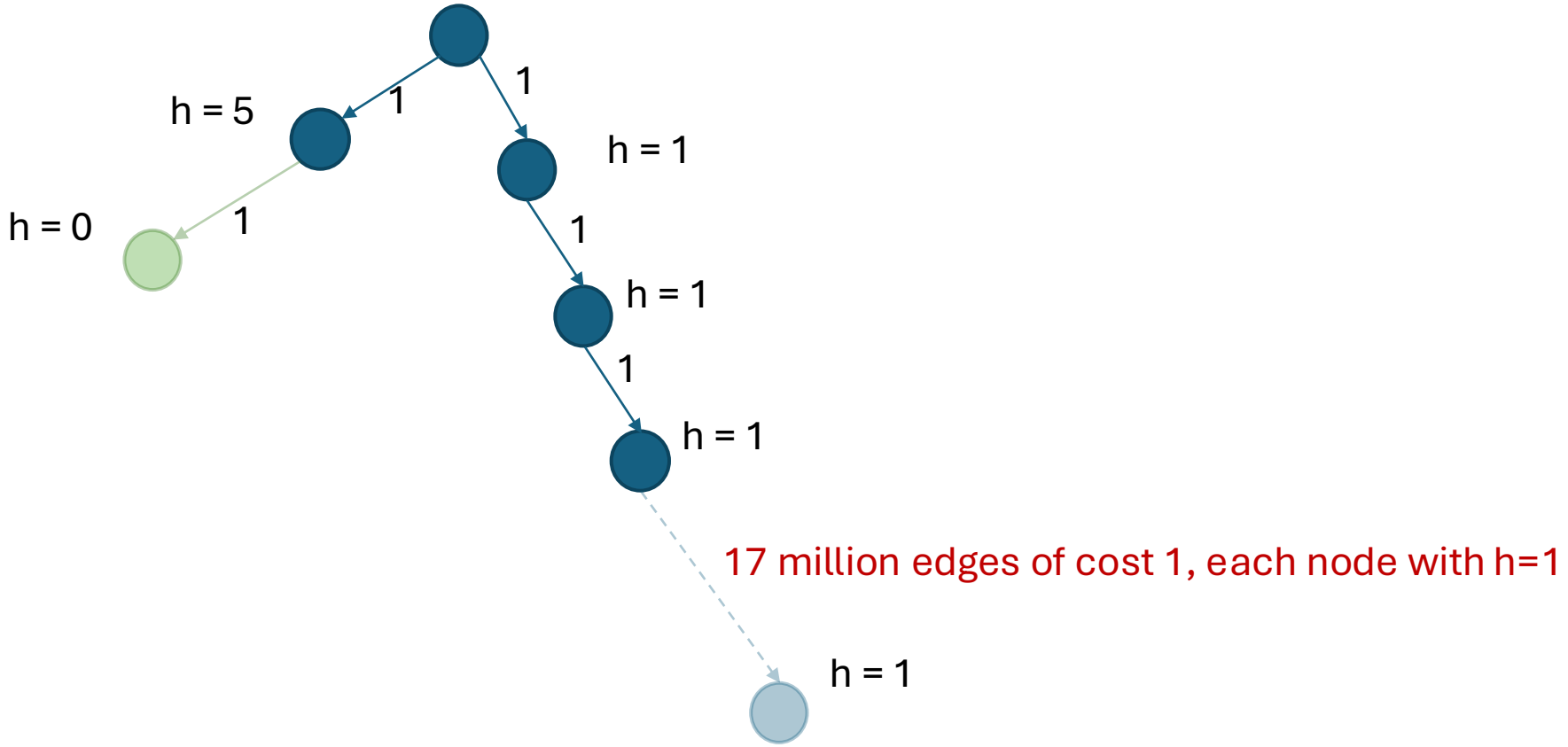
Greedy Search: Negative Result

- 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



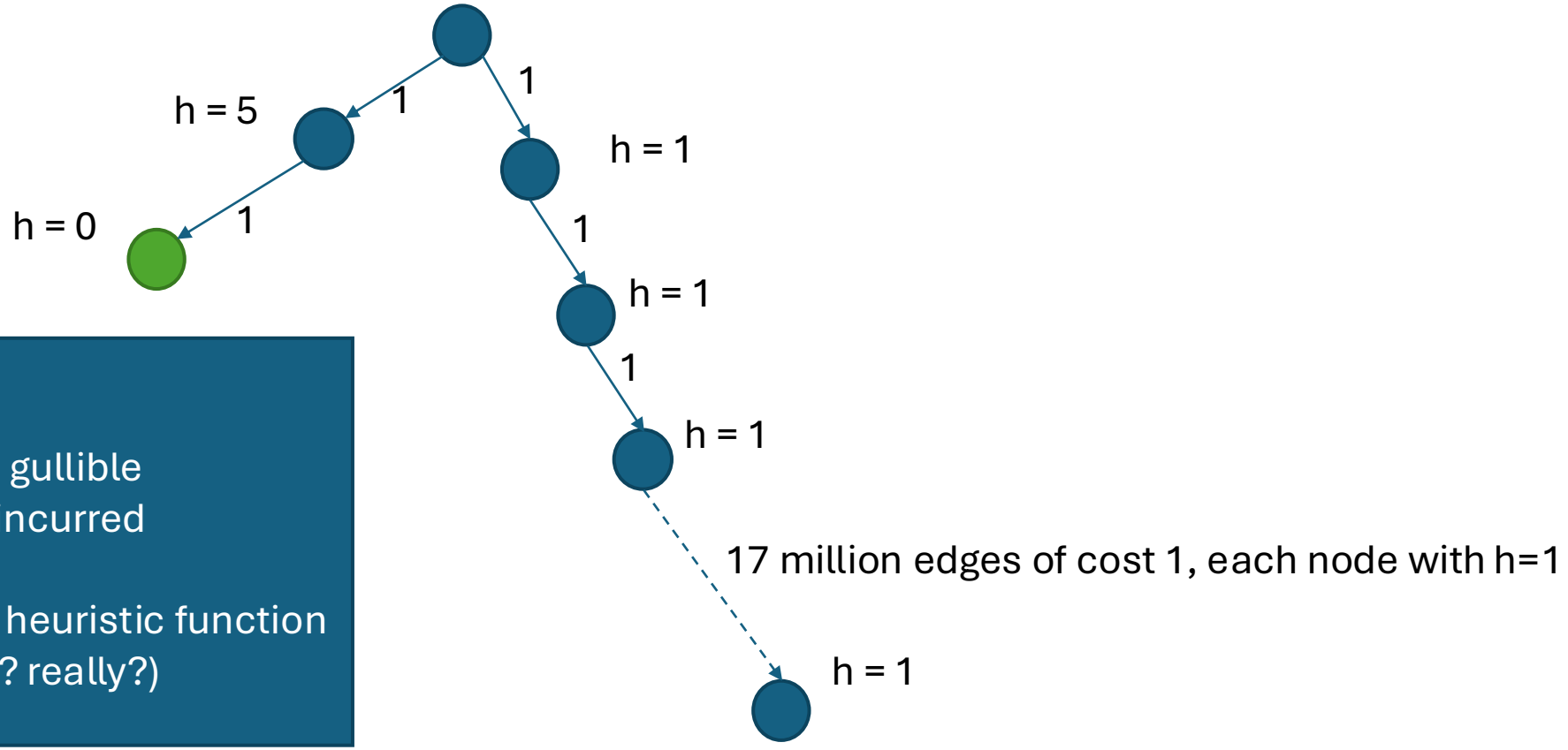
Greedy Search: Negative Result

- 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

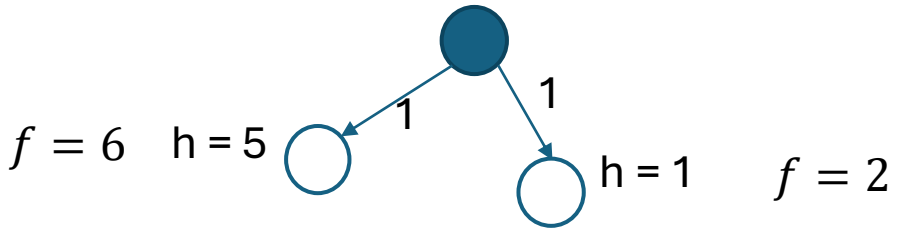


Oops!

- 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?)

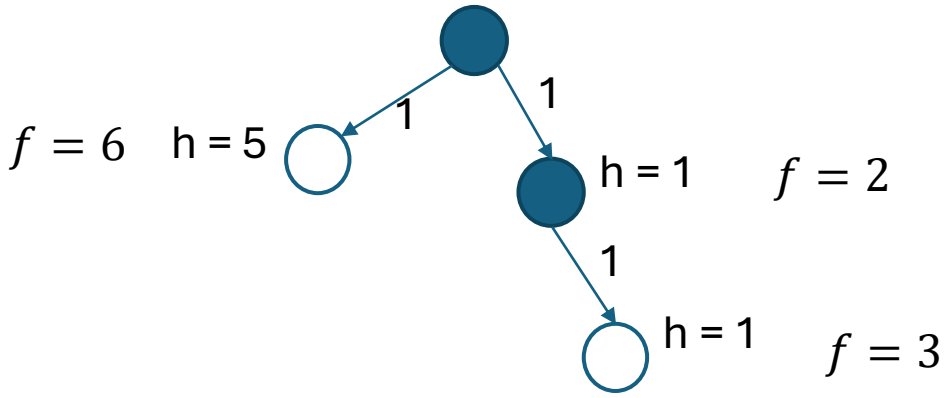
A* Search

- 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



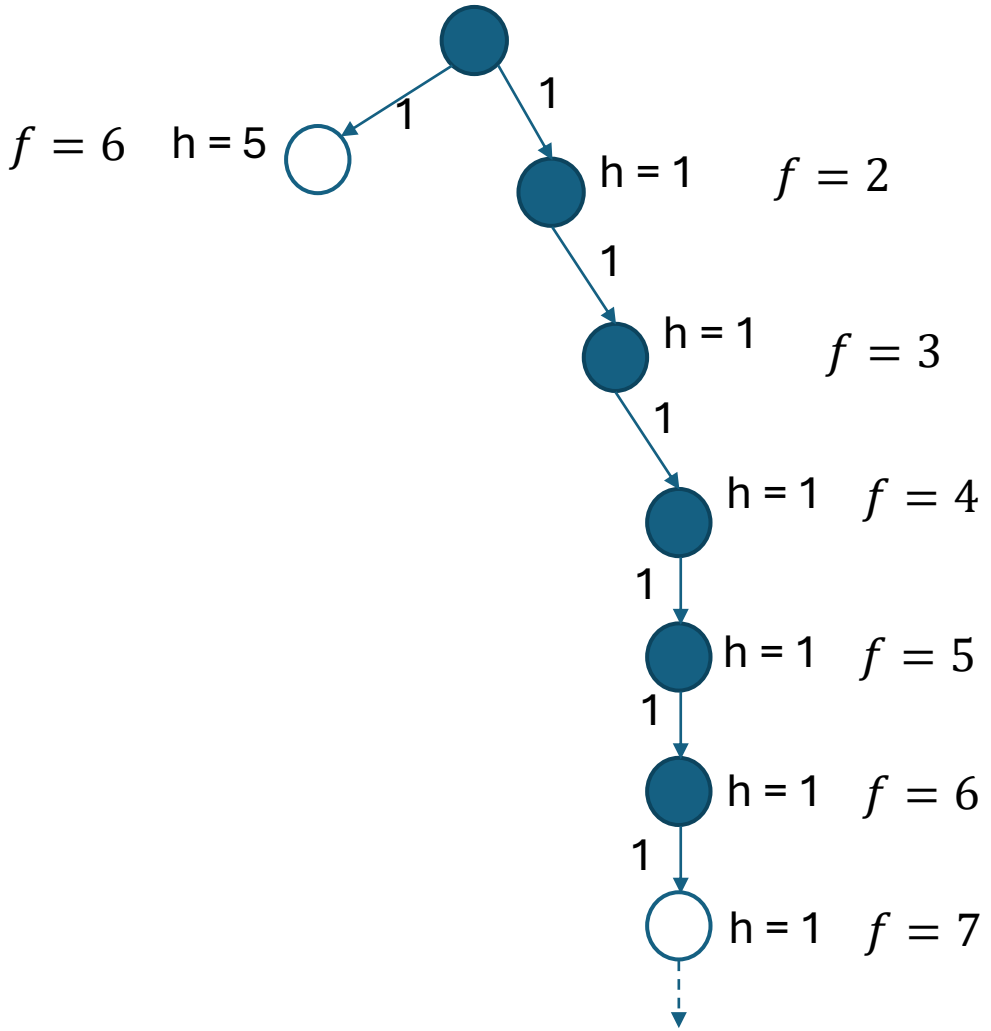
A* Search

- 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



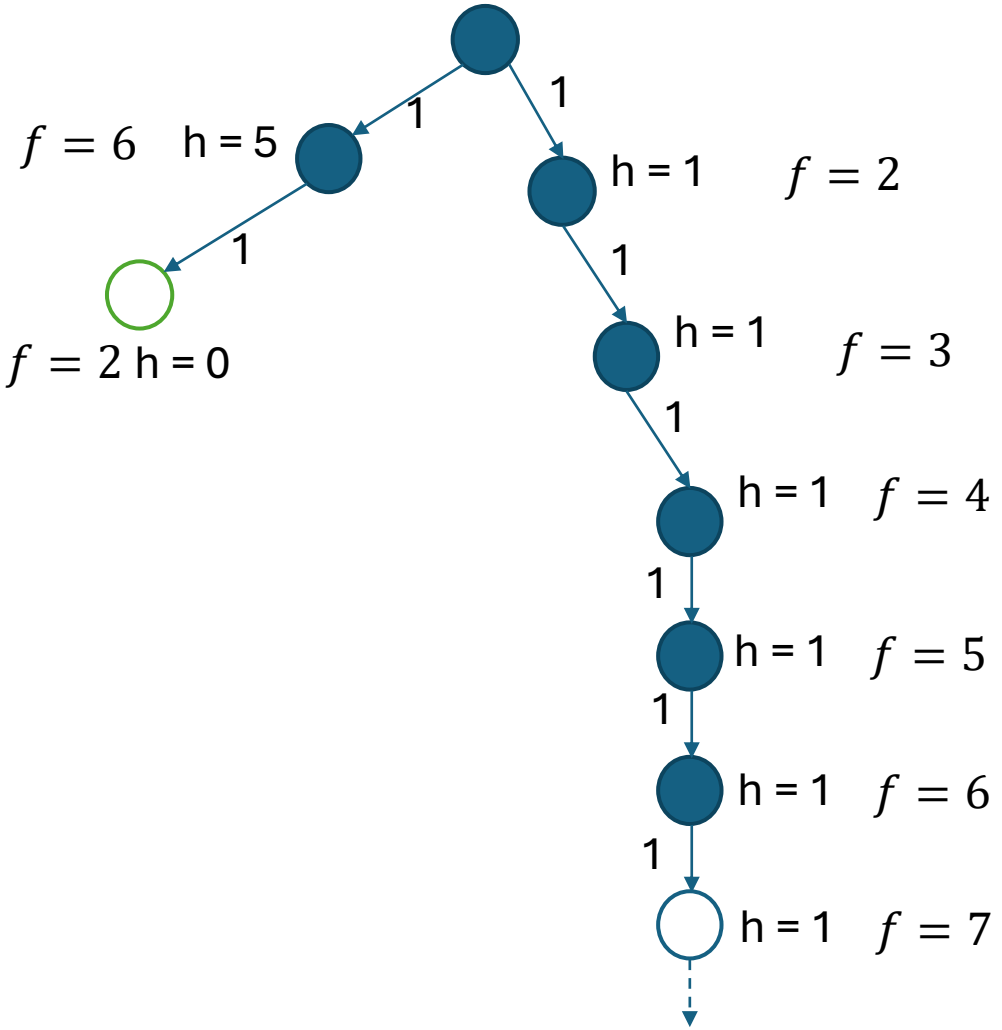
A* Search

- 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

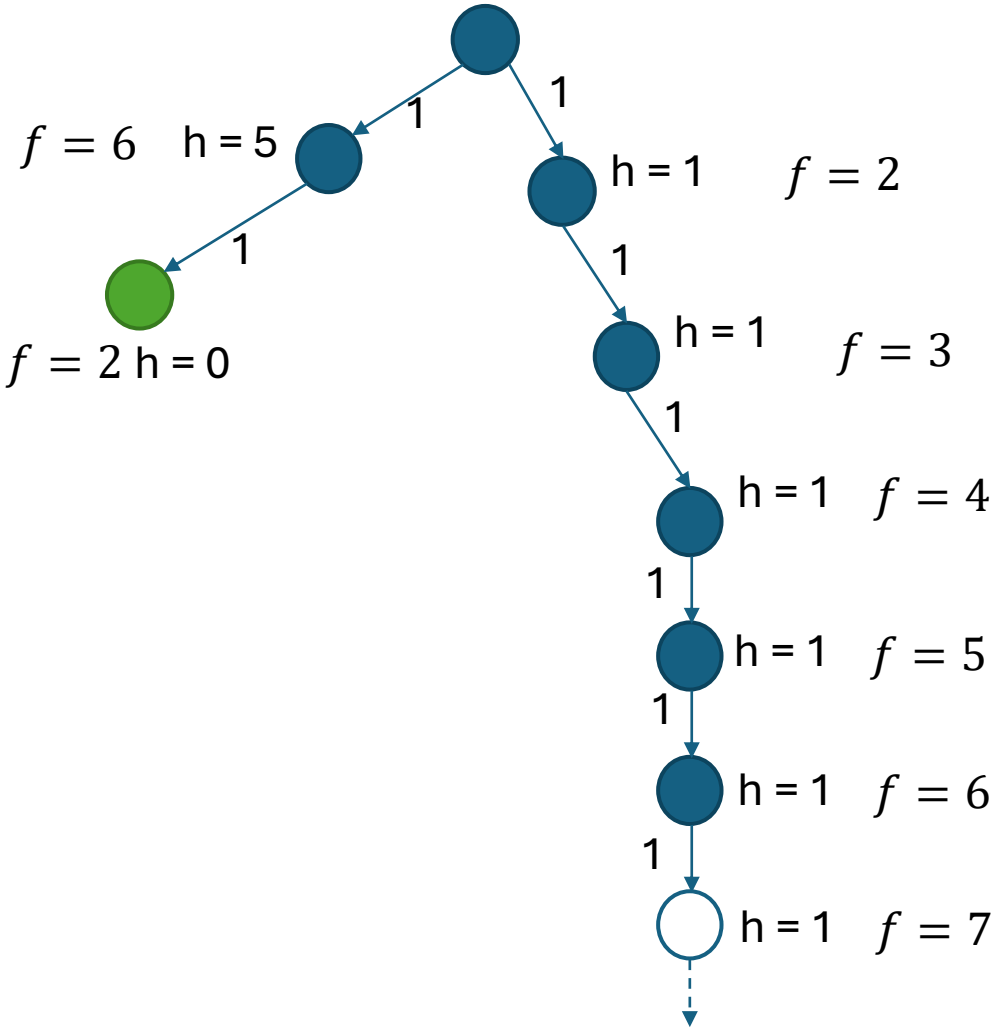
- A* directly addresses those limitations
- $g(n)$ = cost required to get to the node from the start
- $h(n)$ = estimated cost to the goal from n (heuristic, as before)
- $f(n) = g(n) + h(n)$
- Expansion strategy:
  - expand the node with the least f value



A* Search

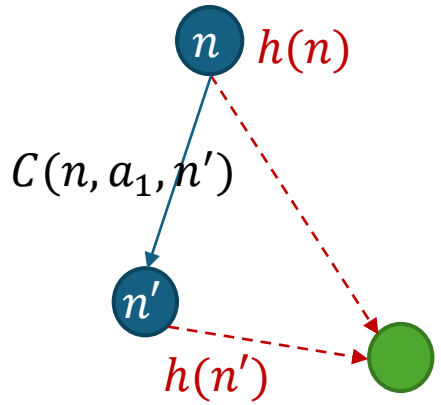
- A* directly addresses those limitations
- $g(n)$ = cost required to get to the node from the start
- $h(n)$ = estimated cost to the goal from n (heuristic, as before)
- $f(n) = g(n) + h(n)$
- Expansion strategy:
  - expand the node with the least f value

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



Optimality of A*

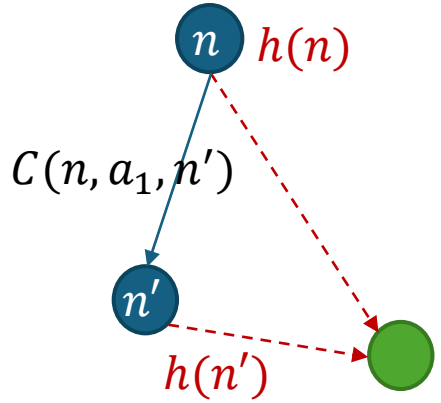
- A heuristic is admissible if
  - $h(n) \leq h^*(n)$, where $h^*(n)$ is the optimal cost of reaching the goal from n
  - $h(g) = 0$
  - Admissible heuristics are optimistic about the cost to the goal
- A heuristic is consistent if for every a
  - $h(n) - h(n') \leq C(n, a, n')$
  - Consistent heuristics are optimistic about the cost of each edge
  - This is a more specific constraint



What is the relationship between admissible and consistent heuristics?

Optimality of A*

- A heuristic is admissible if
  - $h(n) \leq h^*(n)$, where $h^*(n)$ is the optimal cost of reaching the goal from n
  - $h(g) = 0$
  - Admissible heuristics are optimistic about the cost to the goal
- A heuristic is consistent if for every a
  - $h(n) - h(n') \leq C(n, a, n')$
  - Consistent heuristics are optimistic about the cost of each edge
  - This is a more specific constraint



- When using a consistent heuristic, graph search and tree search versions of A* are optimal
- When using an admissible but inconsistent heuristic,
  - tree search version of A* is optimal
  - graph search version is optimal when closed list is managed carefully