CS 4810 Artificial Intelligence
CS 6810 Topics in Artificial Intelligence

Review: Tree search

- Initialize the **frontier** using the **starting state**
- While the frontier is not empty
  - Choose a frontier node to expand according to **search strategy** and take it off the frontier
  - If the node contains the **goal state**, return solution
  - Else **expand** the node and add its children to the frontier

- To handle repeated states:
  - Keep an **explored set**; add each node to the explored set every time you expand it
  - Every time you add a node to the frontier, check whether it already exists in the frontier with a higher path cost, and if yes, replace that node with the new one

Review: Uninformed search strategies

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

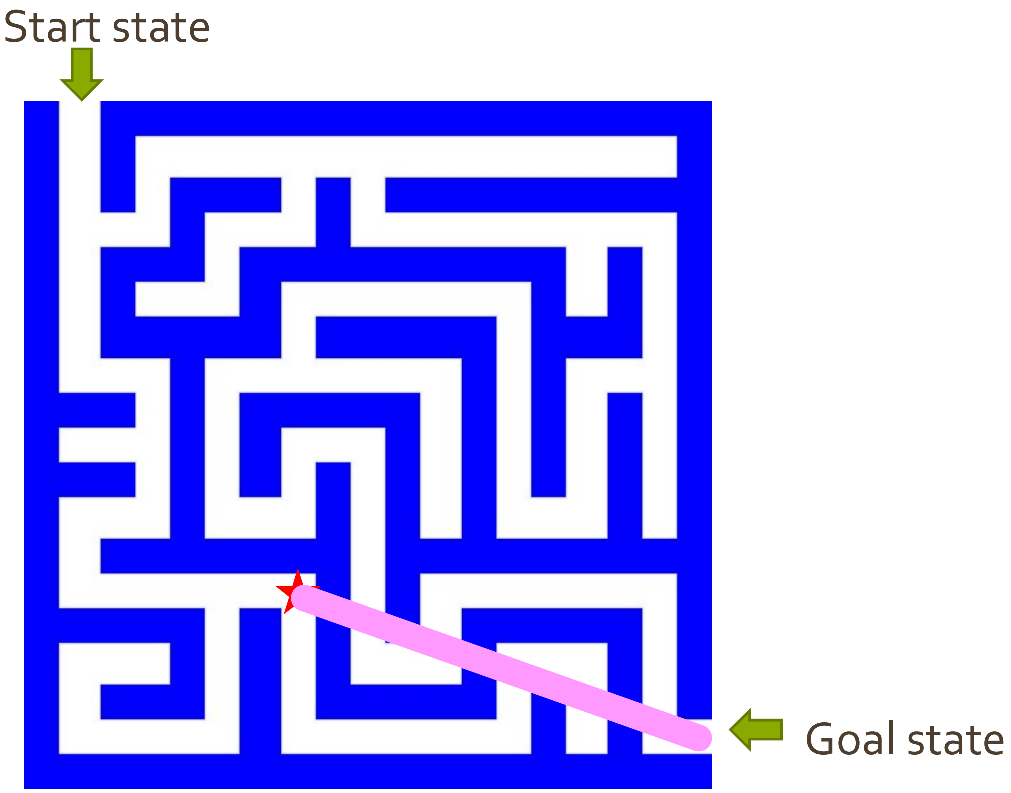
Informed search strategies (Sections 3.5-3.6)

- Idea: give the algorithm "hints" about the desirability of different states
  - Use an ***evaluation function*** to rank nodes and select the most promising one for expansion

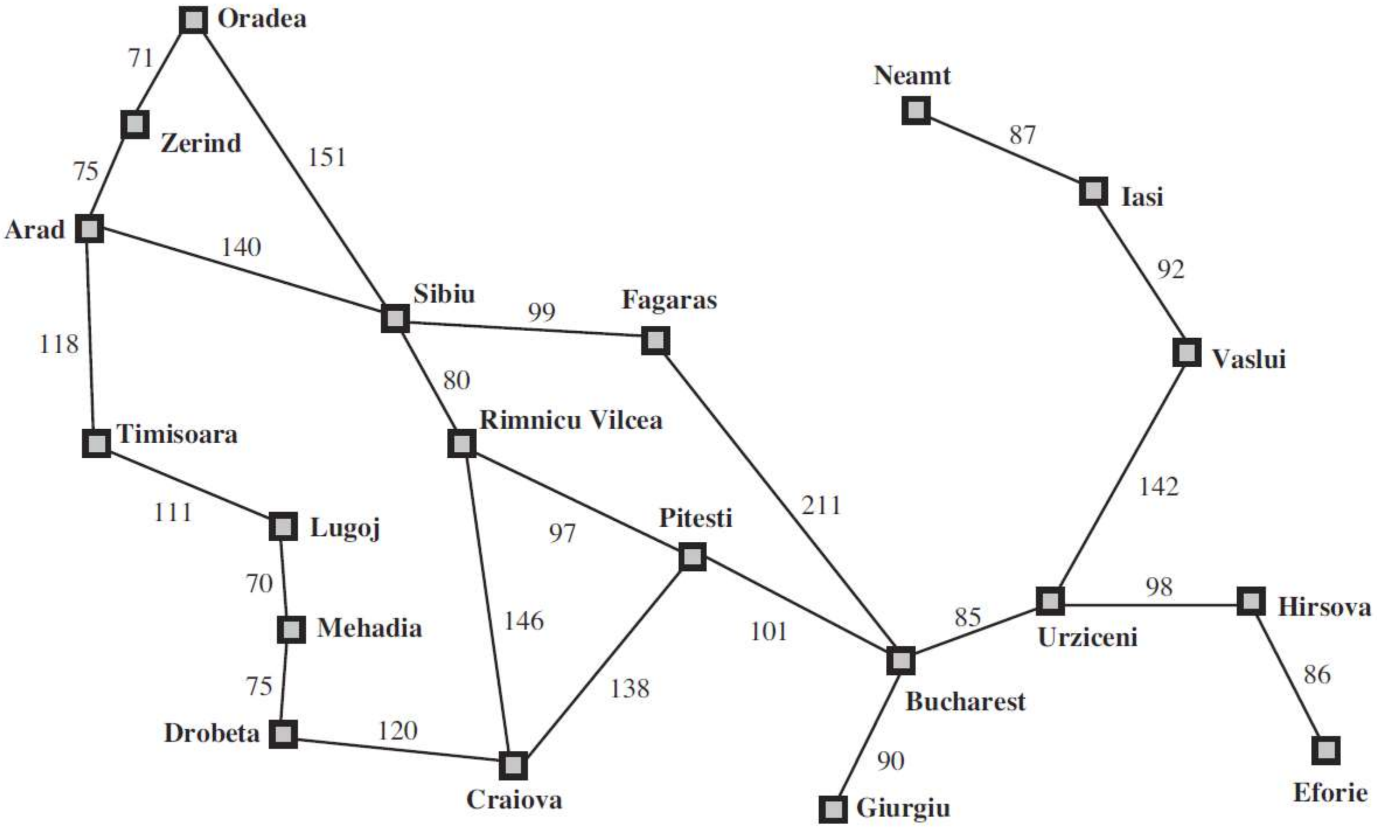
- Greedy best-first search
- A* search

Heuristic function

- **Heuristic function** *h*(*n*) estimates the cost of reaching goal from node *n*
- Example:



Heuristic for the Romania problem

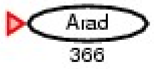


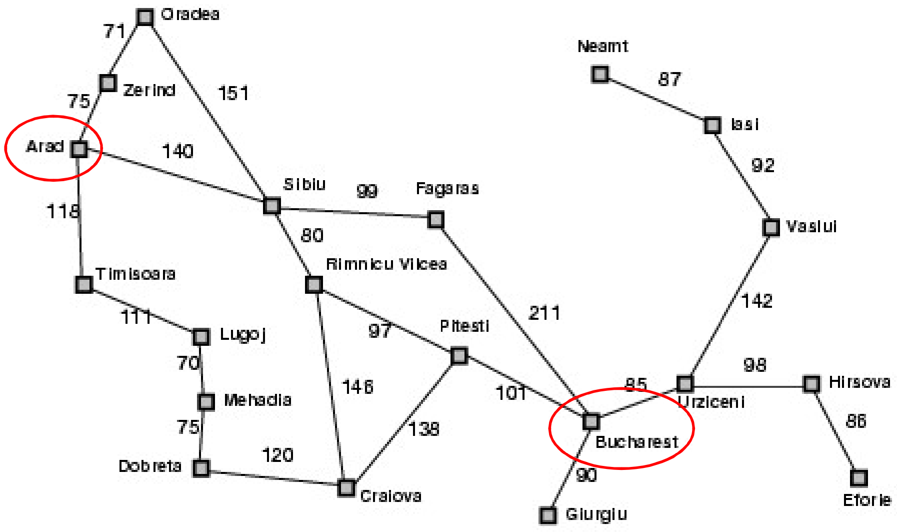
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search

- Expand the node that has the lowest value of the heuristic function *h*(*n*)

Greedy best-first search example

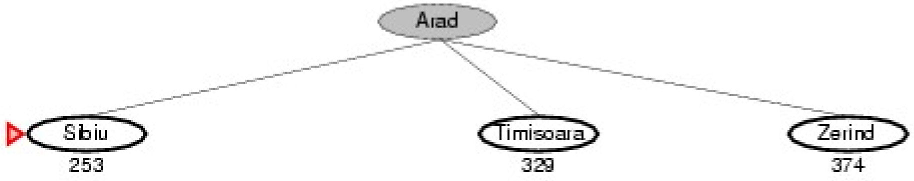


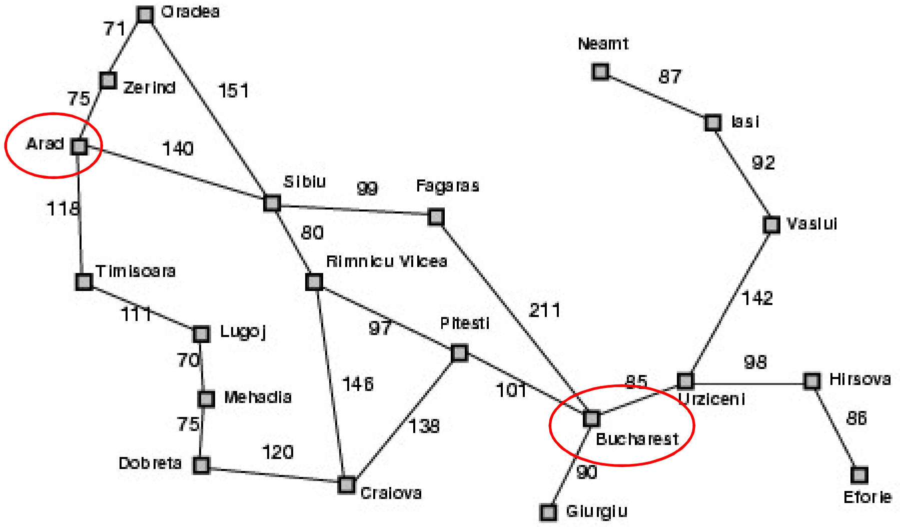


Straight−line distance to Bucharest

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

Greedy best-first search example

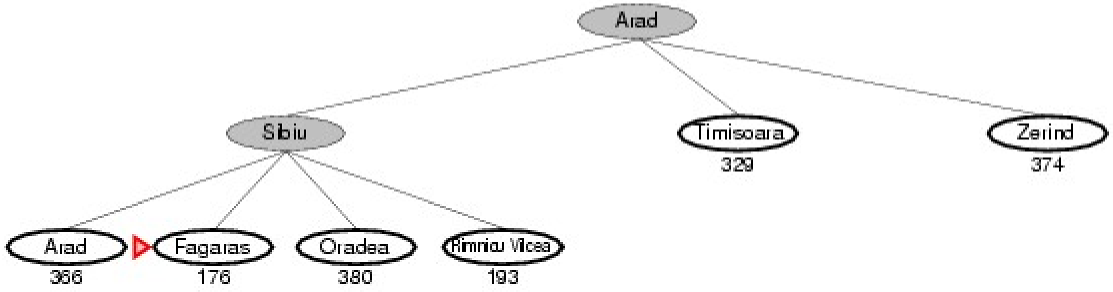


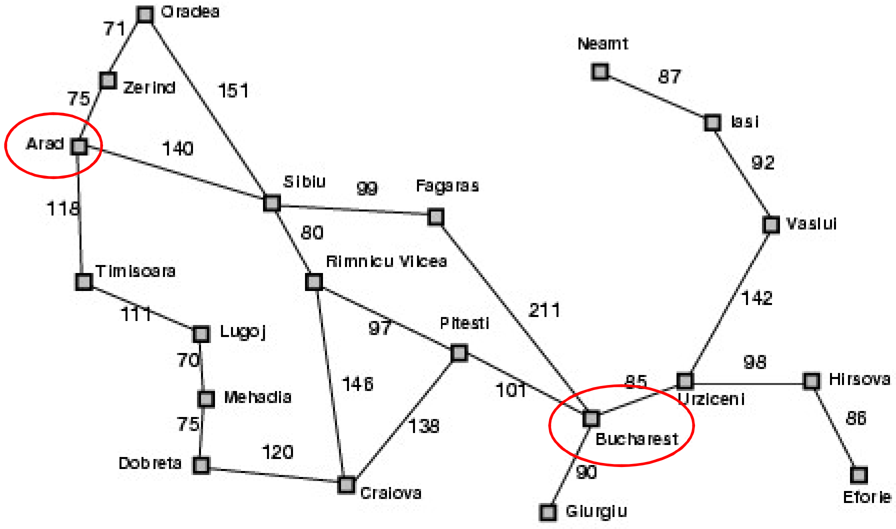


Straight–line distance
to Bucharest

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

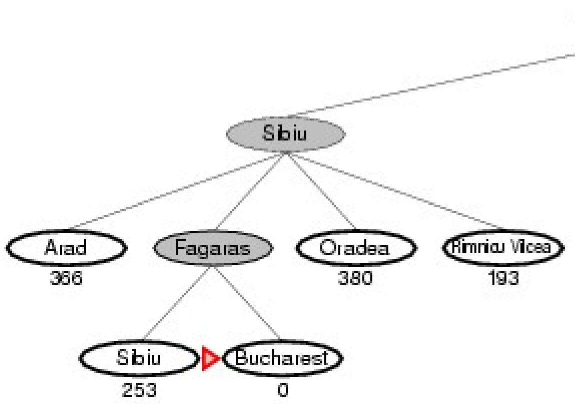
Greedy best-first search example

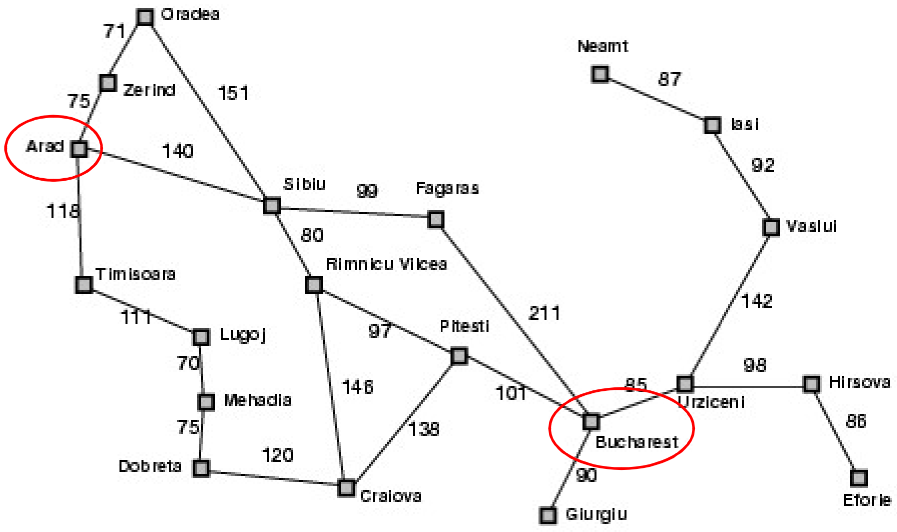




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search example





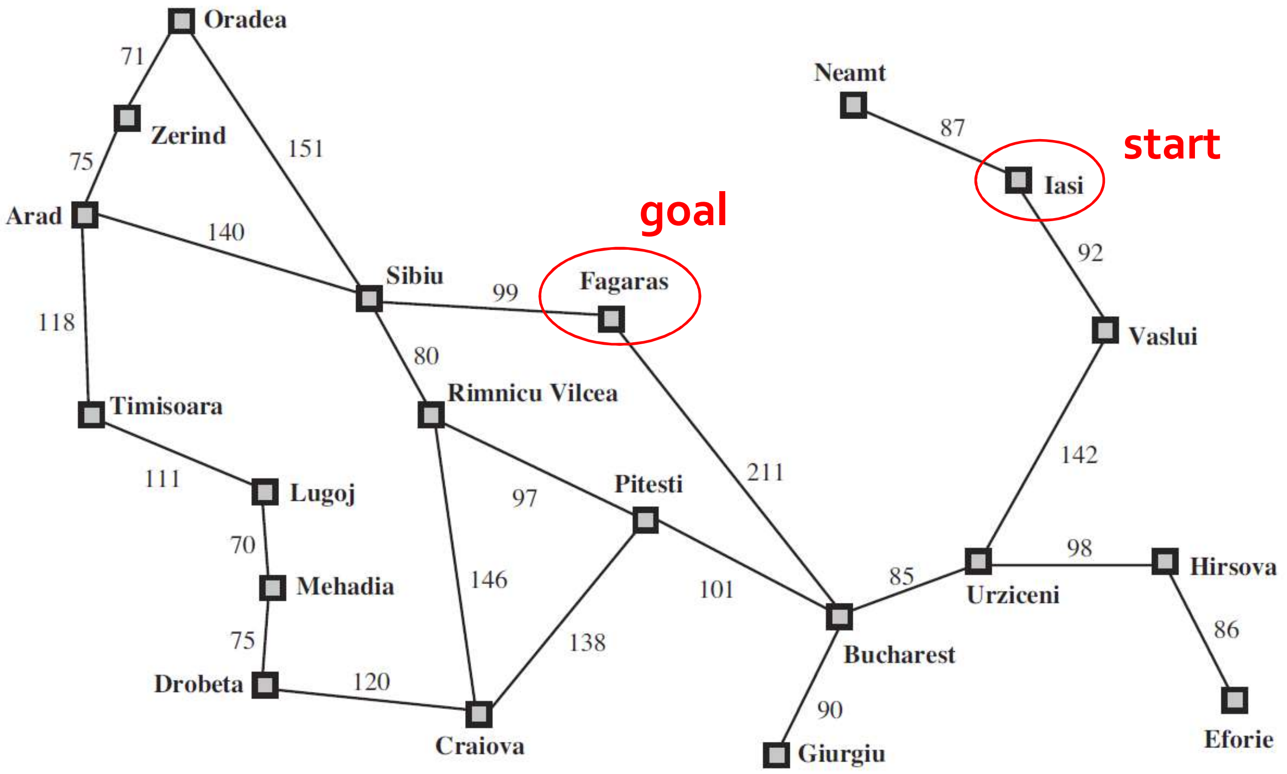
Straight–line distance
to Bucharest

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

Properties of greedy best-first search

- **Complete?**

  No – can get stuck in loops



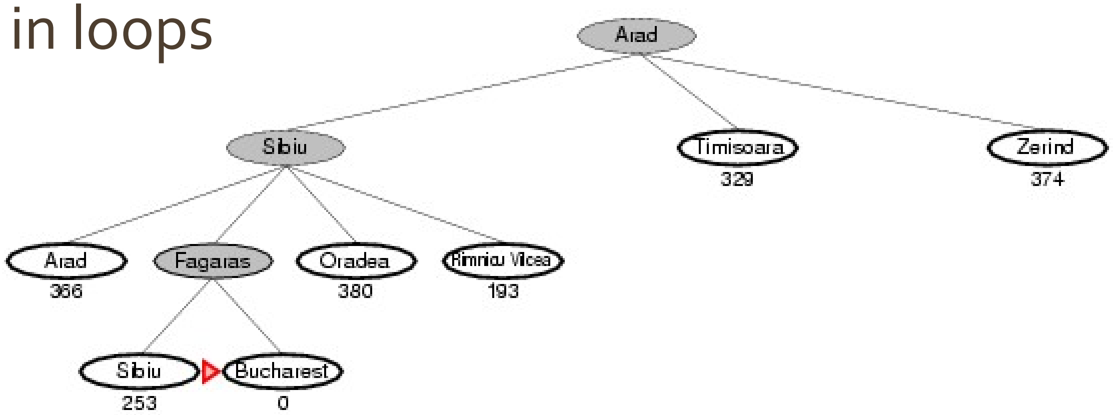
Properties of greedy best-first search

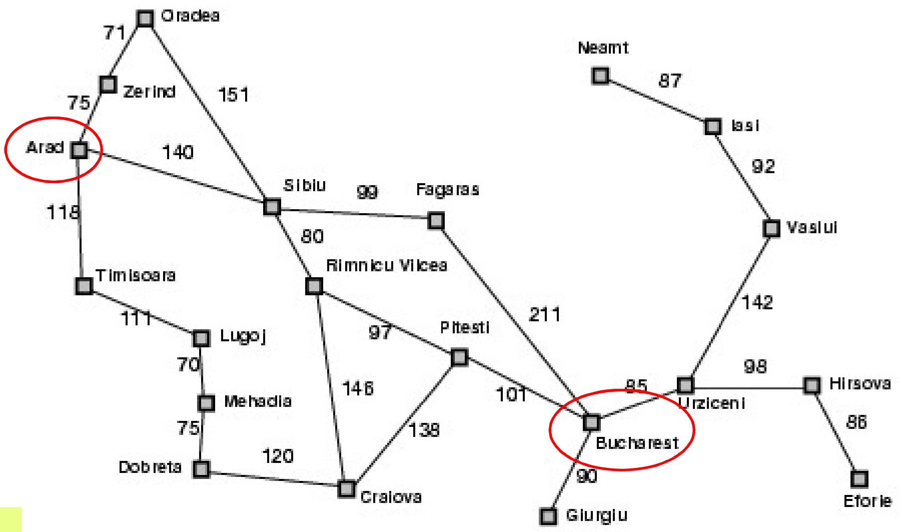
- **Complete?**

  No – can get stuck in loops

- **Optimal?**

  No





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

Properties of greedy best-first search

- **Complete?**

  No – can get stuck in loops

- **Optimal?**

  No

- **Time?**

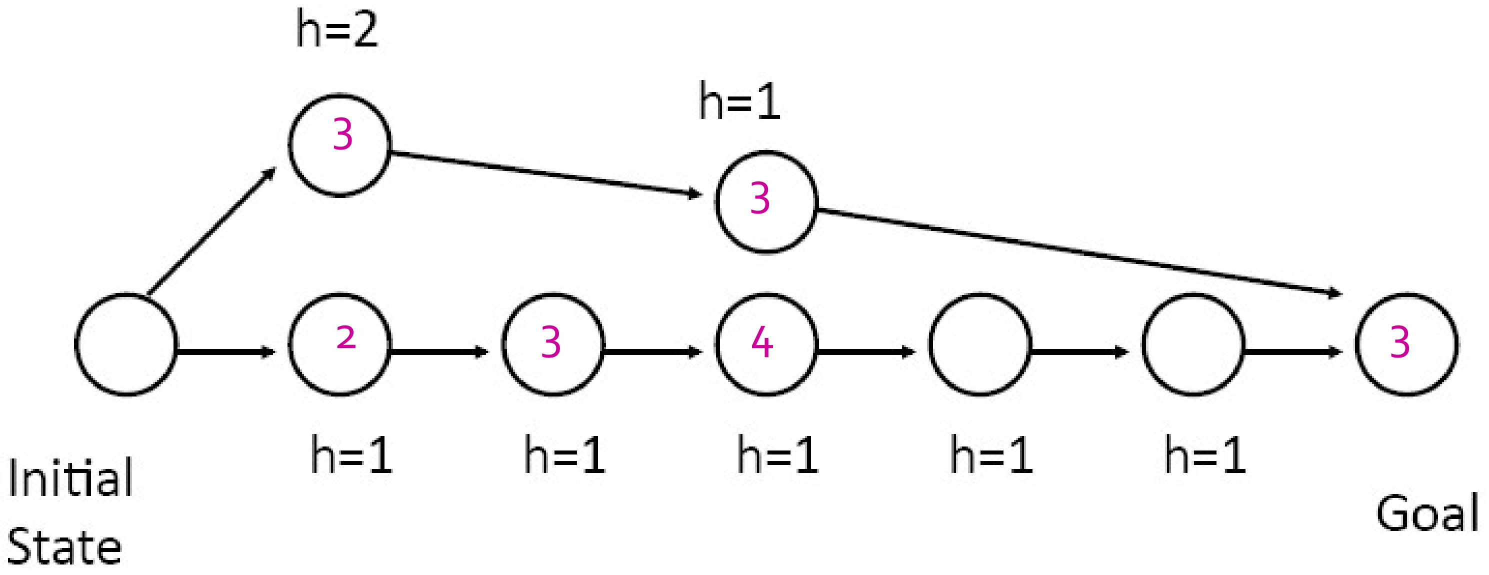
  Worst case: $O(b^m)$

  Can be much better with a good heuristic

- **Space?**

  Worst case: $O(b^m)$

How can we fix the greedy problem?



- How about keeping track of the distance already traveled in addition to the distance remaining?

A* search

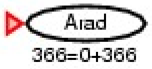
- Idea: avoid expanding paths that are already expensive
- The **evaluation function** $f(n)$ is the estimated total cost of the path through node *n* to the goal:

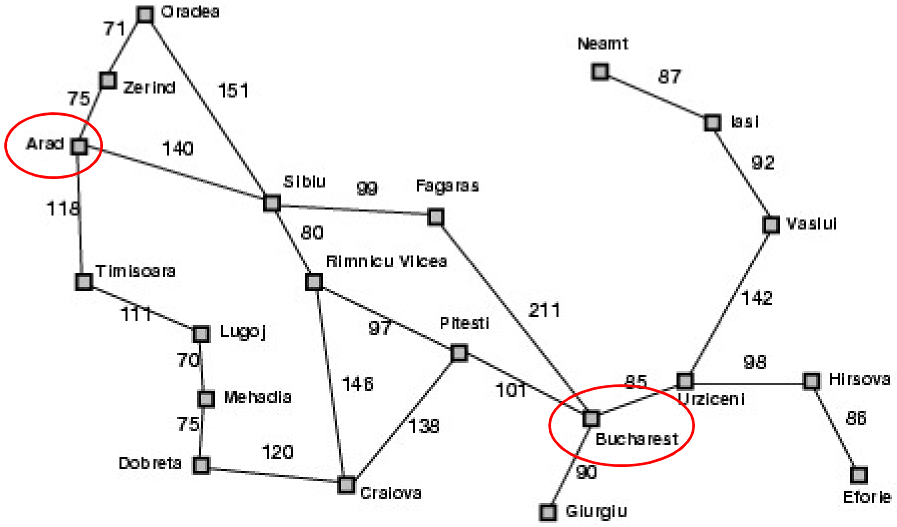
$$f(n) = g(n) + h(n)$$

$g(n)$: cost so far to reach *n* (path cost)

$h(n)$: estimated cost from *n* to goal (heuristic)

A* search example

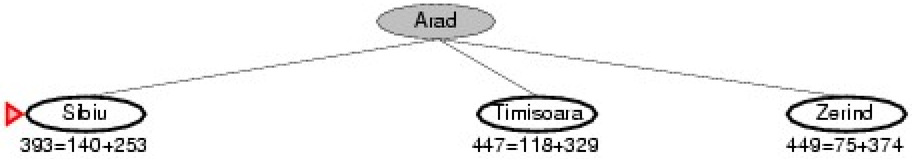


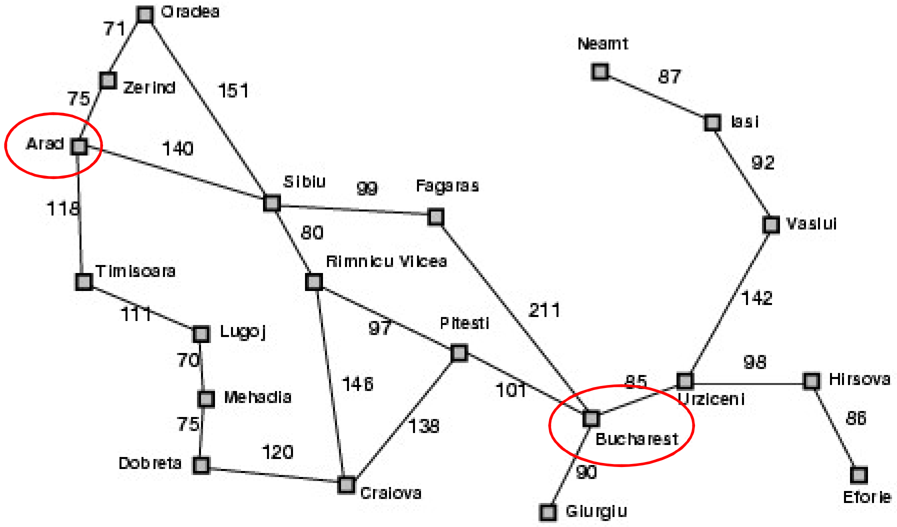


Straight–line distance to Bucharest

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

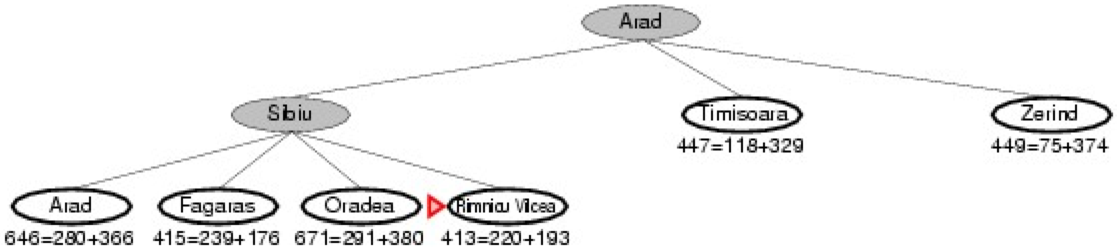
A* search example

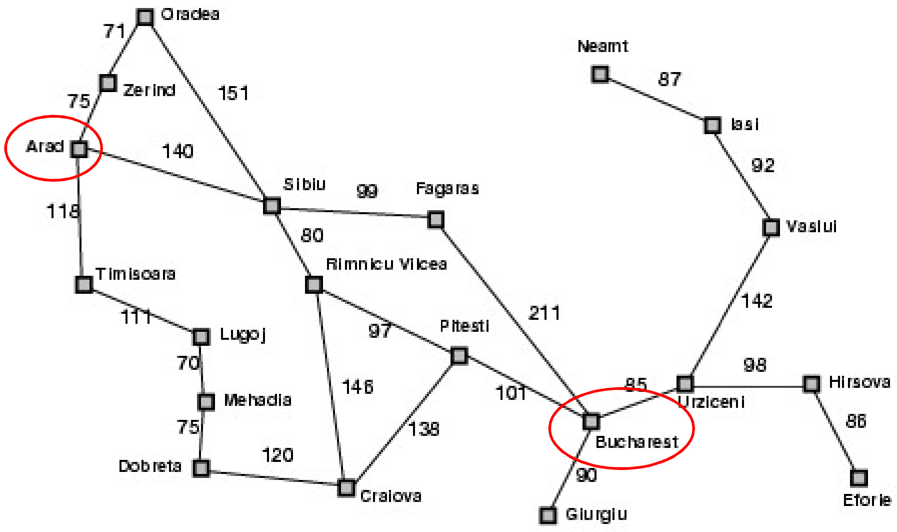




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

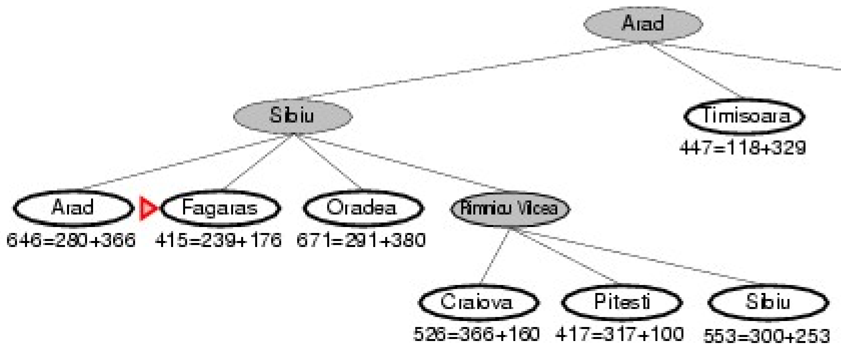


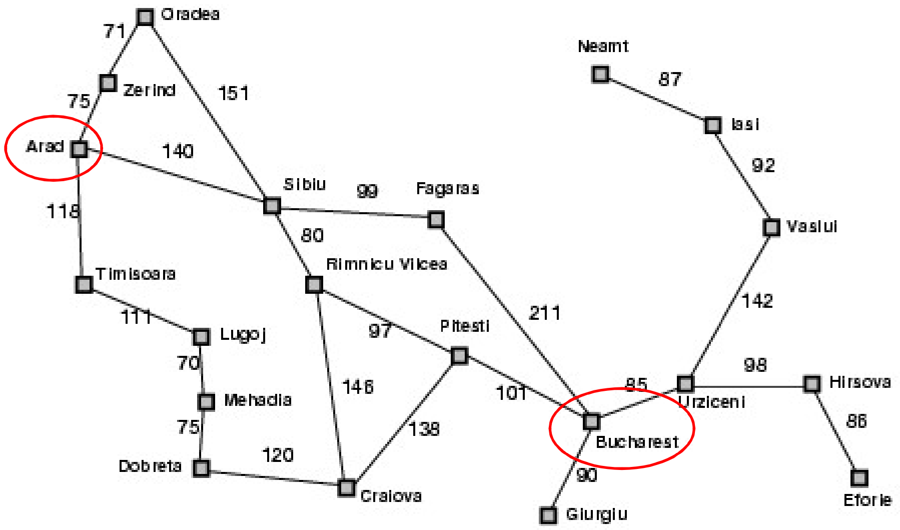


Straight–line distance
to Bucharest

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

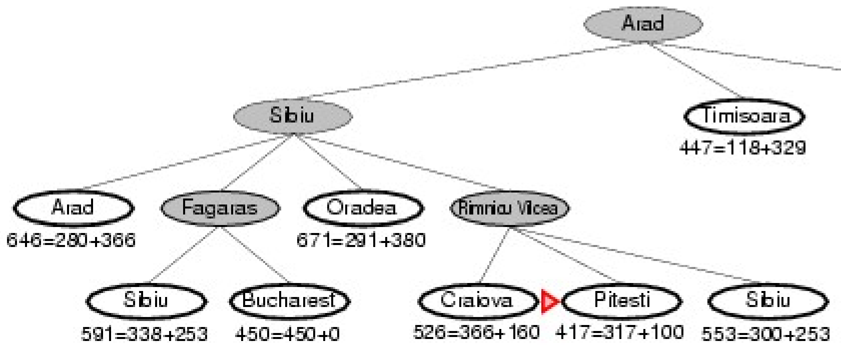
A* search example

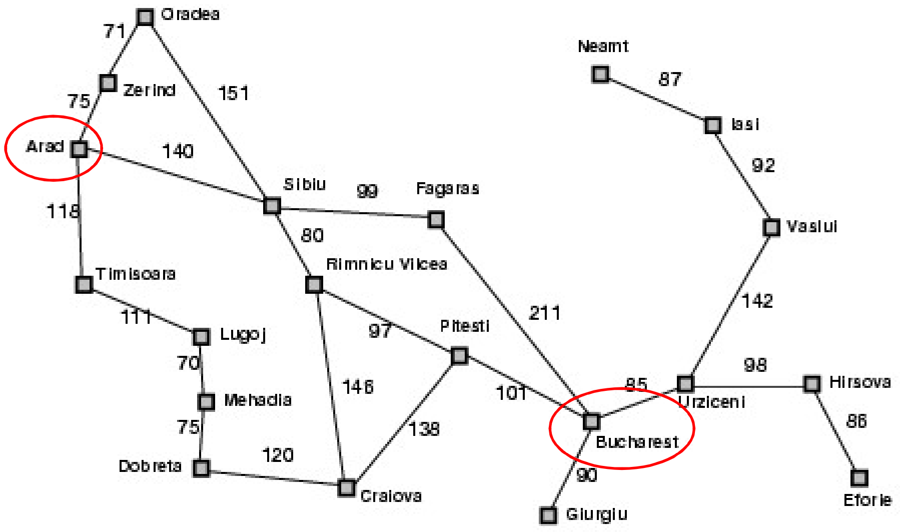




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

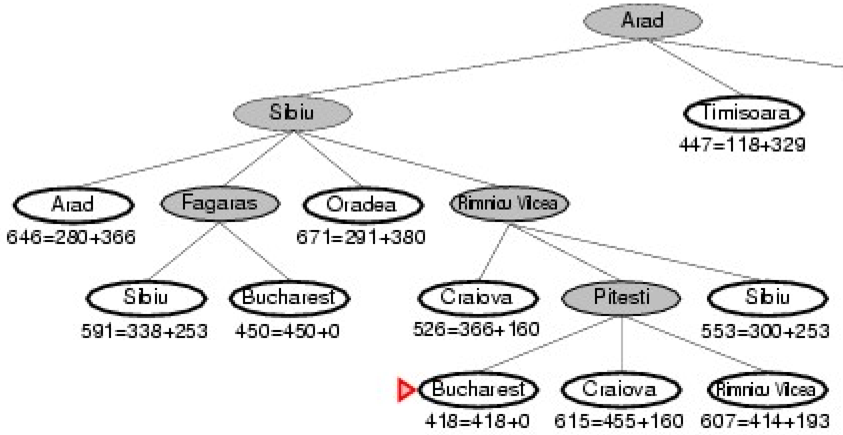


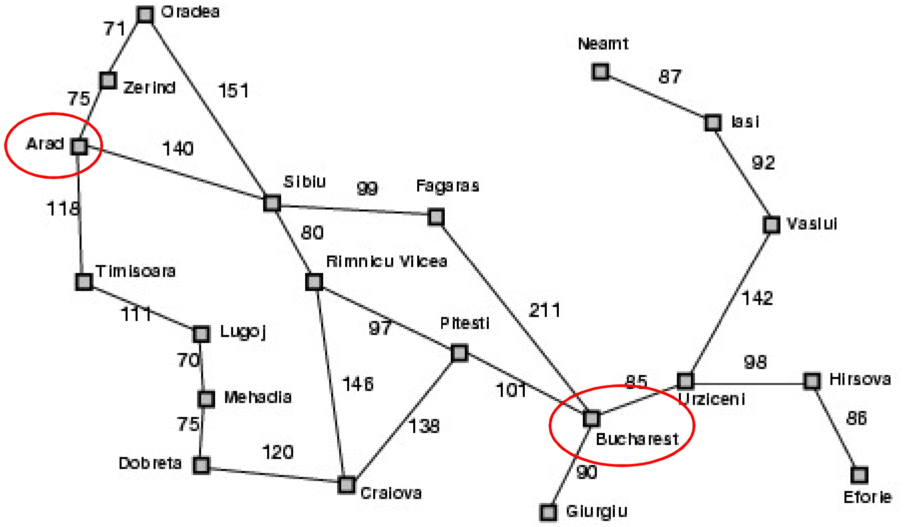


Straight–line distance
to Bucharest

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

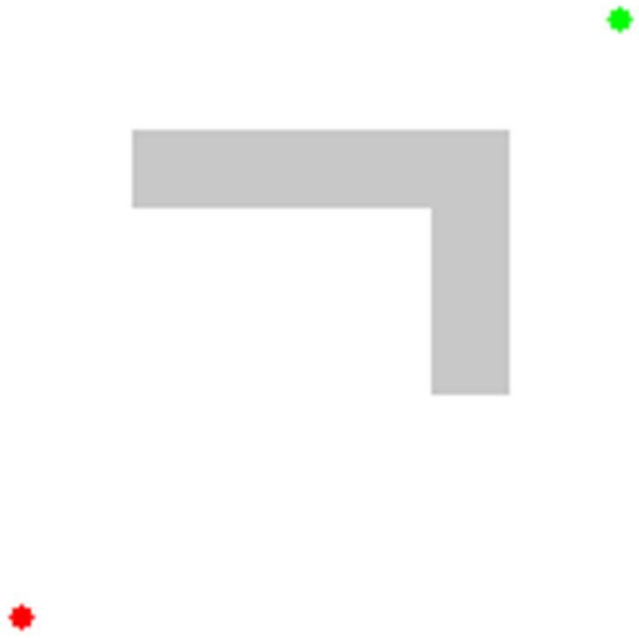
A* search example





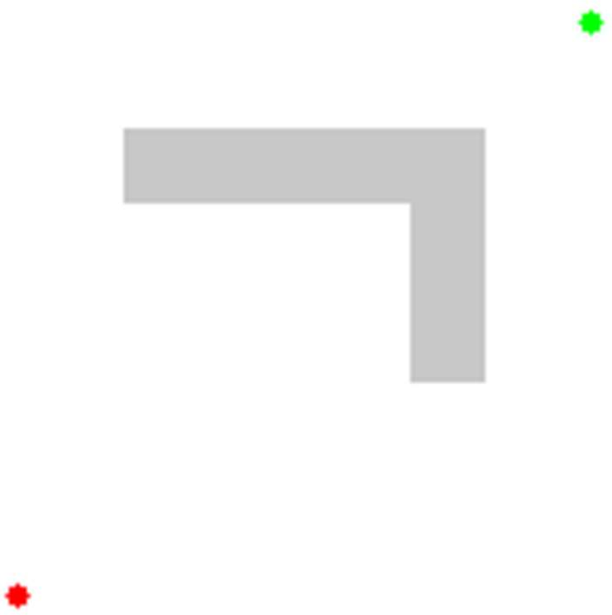
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Another example



Source: Wikipedia

Uniform cost search vs. A* search



Source: Wikipedia

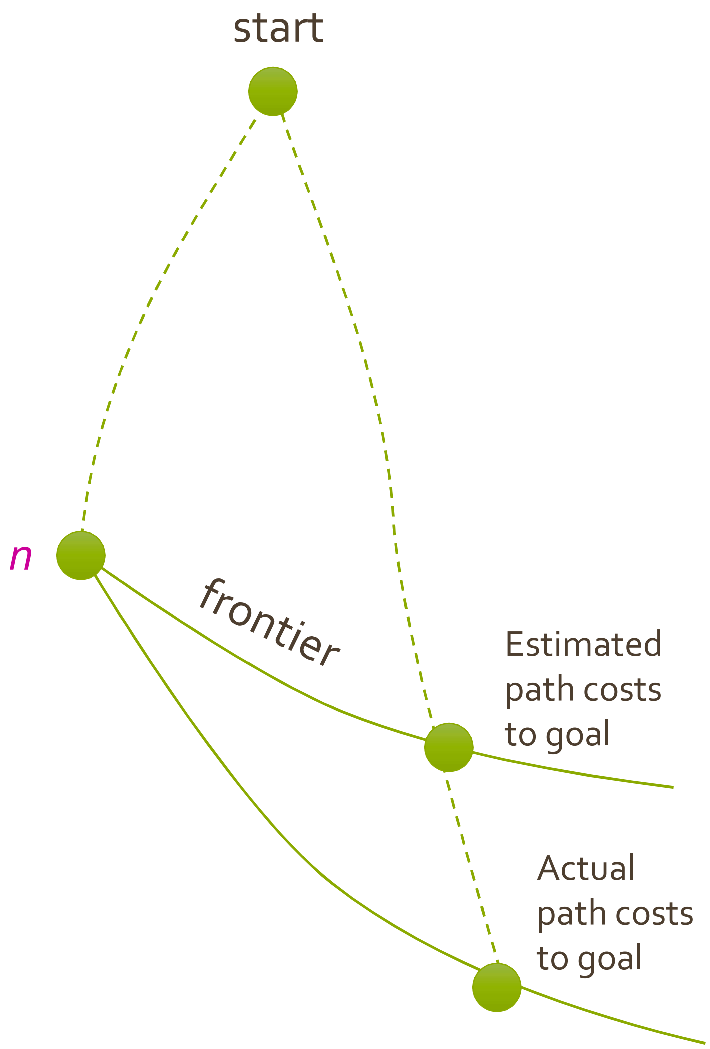
Admissible heuristics

- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- A heuristic $h(n)$ is admissible if for every node n, $h(n) \leq h^{*}(n)$, where $h^{*}(n)$ is the true cost to reach the goal state from n
- Example: straight line distance never overestimates the actual road distance
- **Theorem:** If $h(n)$ is admissible, A^{*} is optimal

Optimality of A*

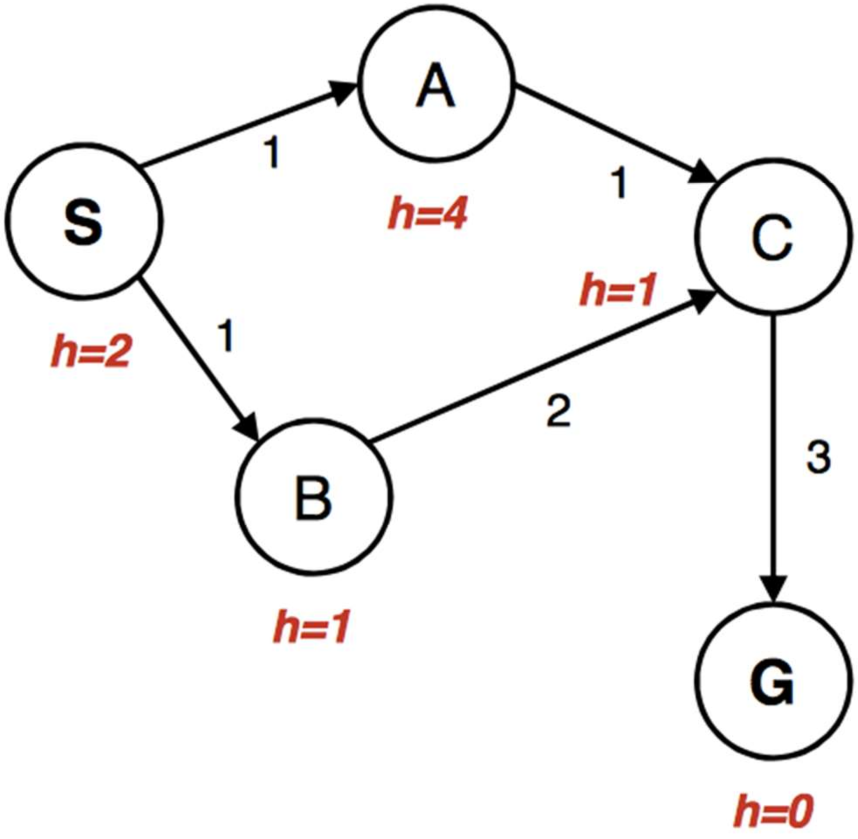
- **Theorem:** If the heuristic is admissible, A* *without repeated state detection* is optimal

- Proof sketch:
  - Let *C** be the evaluation function value (*actual* path cost) of the first goal node we select for expansion
  - Then all the other nodes on the frontier have *estimated* path costs to the goal that are at least as big as *C**
  - Because we are using an admissible heuristic, the *true* path costs to the goal for those nodes cannot be less than *C**

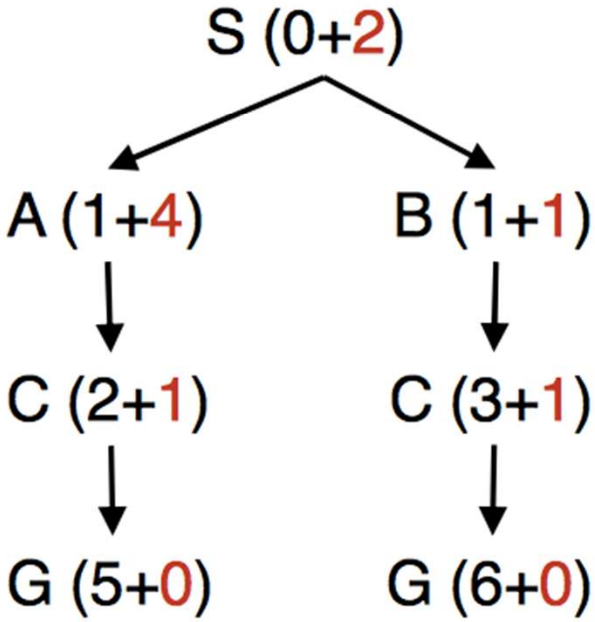


A* gone wrong?

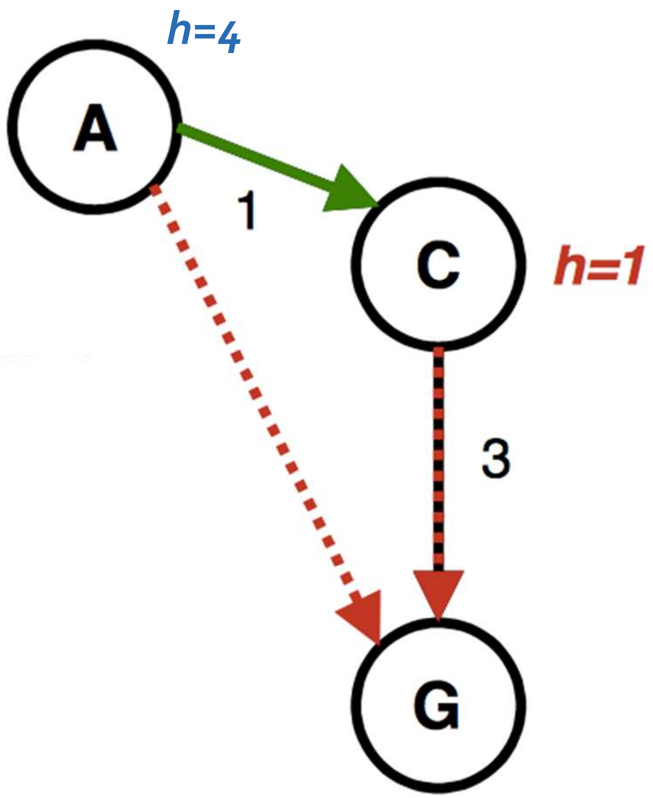
State space graph



Search tree



Consistency of heuristics



- Consistency: Stronger than admissibility
- Definition:

  cost(A to C) + h(C) ≥ h(A)

  cost(A to C) ≥ h(A) - h(C)

  real cost ≥ cost implied by heuristic

- Consequences:
  - The f value along a path never decreases
  - A* graph search is optimal

Source: Berkeley CS188x

Optimality of A*

- **Tree search** (i.e., search without repeated state detection):
  - A* is optimal if heuristic is ***admissible*** (and non-negative)
- **Graph search** (i.e., search with repeated state detection)
  - A* optimal if heuristic is ***consistent***
- Consistency implies admissibility
  - In general, most natural admissible heuristics tend to be consistent, especially if they come from relaxed problems

Source: Berkeley CS188x

Optimality of A*

- A* is *optimally efficient* – no other tree-based algorithm that uses the same heuristic can expand fewer nodes and still be guaranteed to find the optimal solution
    - A* expands all nodes for which $f(n) \leq C^*$. Any algorithm that does not risks missing the optimal solution

Properties of A*

- **Complete?**

  Yes – unless there are infinitely many nodes with *f*(*n*) ≤ *C**

- **Optimal?**

  Yes

- **Time?**

  Number of nodes for which *f*(*n*) ≤ *C** (exponential)

- **Space?**

  Exponential