Informed search methods

Uninformed vs. Informed Search

- Uninformed search strategies
  - Find solutions to problems by systematically generating new states and testing for goal
  - Most are incredibly inefficient

Uninformed vs. Informed Search

- Uninformed search strategies
  - Find solutions to problems by systematically generating new states and testing for goal
  - Most are incredibly inefficient
- Informed search strategies
  - Use problem-specific knowledge
  - Find solutions more efficiently

Informed Search

- Informed search strategy
  - One that uses problem-specific knowledge beyond the problem definition itself
- General approach is called "Best First" search
  - Node selected for expansion based on an evaluation function, f(n)
    - Measuring a distance to goal
  - Node with lowest evaluation is selected

Techniques

- Best-first search
  - Expand “minimum cost” nodes first
  - Greedy search
  - A* search

Best-First Search

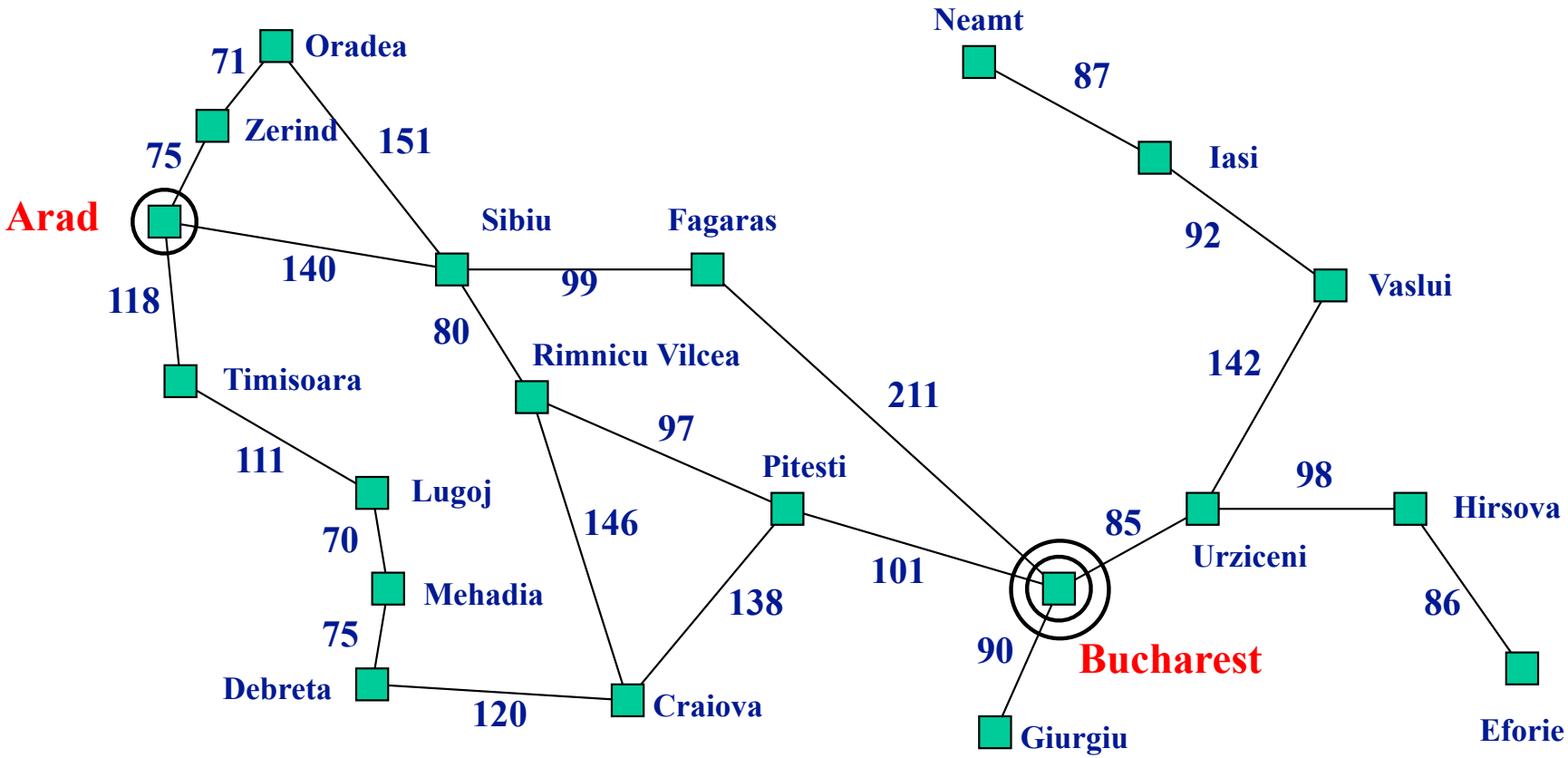
- Expand “minimum cost” nodes first
  - Greedy search
  - A* search
- Open set
  - Priority queue implementation
  - Insert expanded nodes in decreasing order of desirability (most desirable first)

Greedy Search

- Simplest best-first search strategy
- Minimize estimated cost to reach goal
- Heuristic function *h(n)*
  - **Estimate of cost from node *n* to *goal***
    - Require *h(n)* = 0 if *n* is goal

- Greedy search expands the node that *appears* to be closest to the goal
  - Each step tries to get as close to goal as possible
  - Though, will not consider if its action will be the best in the long run

Romania Step Costs in km



Straight-Line Distance to Bucharest

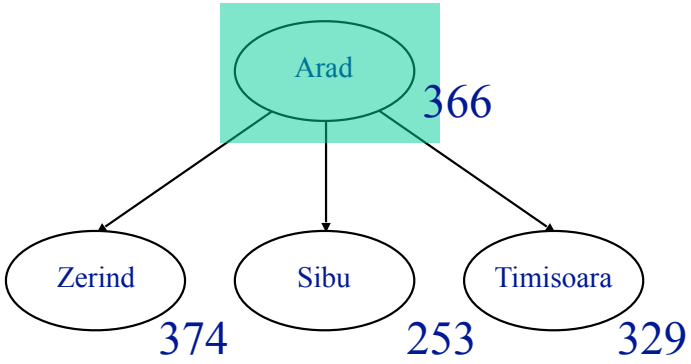
H_{SLD}

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Greedy Search Example: Romania

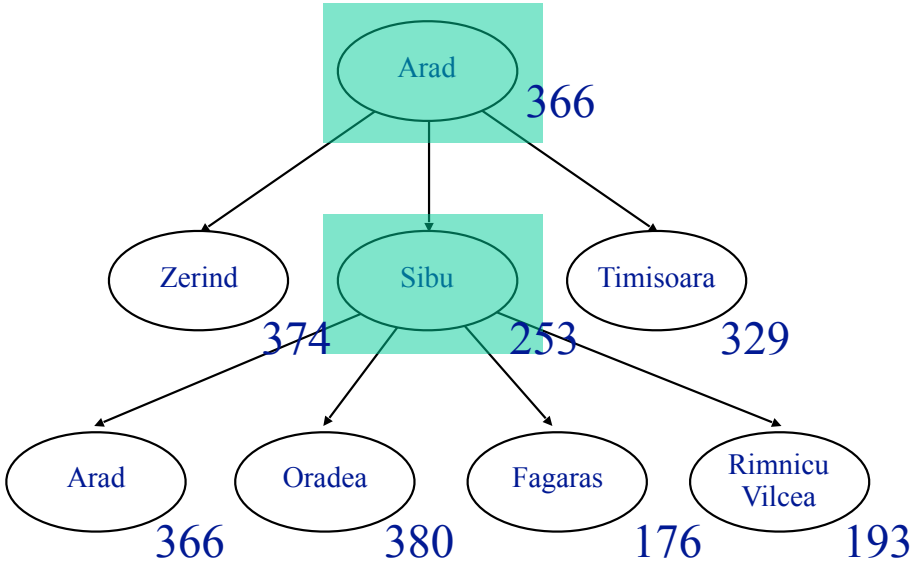
Open = { Arad(366) }

Greedy Search Example: Romania



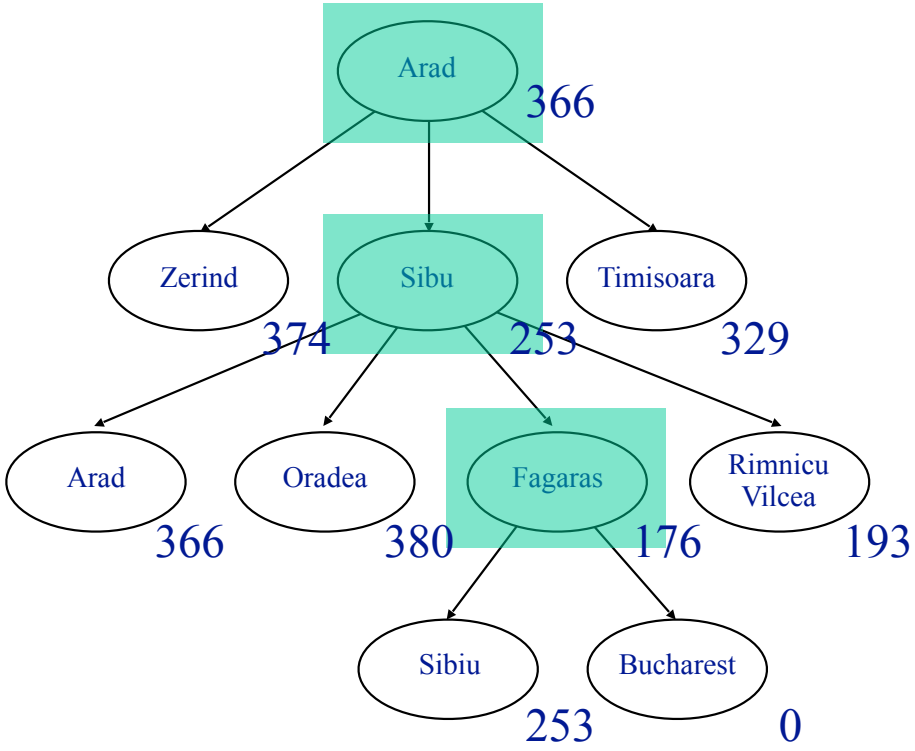
Open = { Sibu(253), Timisoara(329), Zerind(374) }

Greedy Search Example: Romania



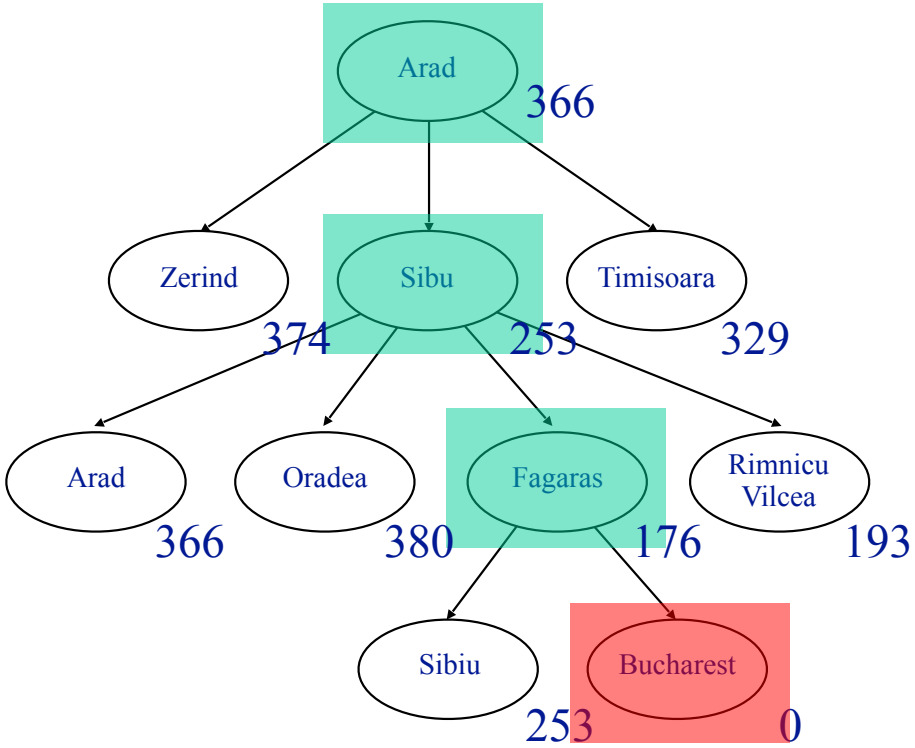
Open = { Fagaras(176), RimnicuVilcea(193), Timisoara(329), Arad(366), Zerind(374), Oradea(380) }

Greedy Search Example: Romania

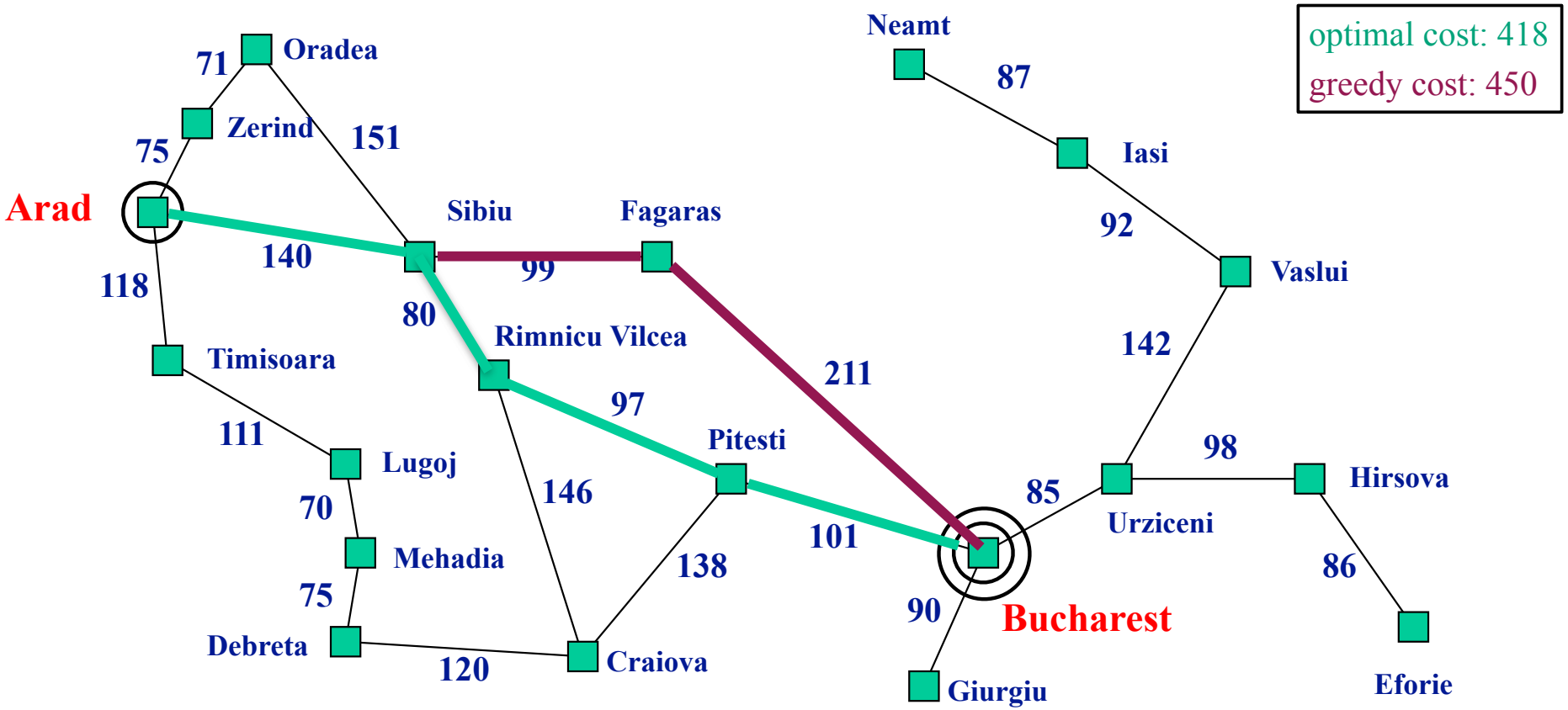


Open = { Bucharest(0), Fagaras(176), RimnicuVilcea(193), Sibu(253), Timisoara(329), Arad(366), Zerind(374), Oradea(380) }

Greedy Search Example: Romania



Romania Step Costs in km



Properties of Greedy Search

- Not complete (can be bad)
  - Can get stuck in loops
  - Complete if check for repeated states
- Time (can be bad)
  - $O(b^m)$, m=maximum depth (worst case, like DFS)
- Space (can be bad)
  - $O(b^m)$, keeps all nodes in memory (worst case)
- Not optimal (can be bad)
  - Heuristic is an estimate

But a good heuristic can give dramatic improvement!

A* Search

- Minimizes total estimated path cost
- Avoids expanding paths already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - Actual cost to reach node n so far → $\boldsymbol{g(n)}$
  - Estimated cost from n to goal → $\boldsymbol{h(n)}$
  - Estimated total cost through n to goal

    → $\boldsymbol{f(n)} = g(n) + h(n)$

- A* search uses *admissible* heuristic
  - i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is true cost from n to goal
  - e.g., $h_{SLD}(n)$ never overestimates actual distance

A* Search Example: Romania

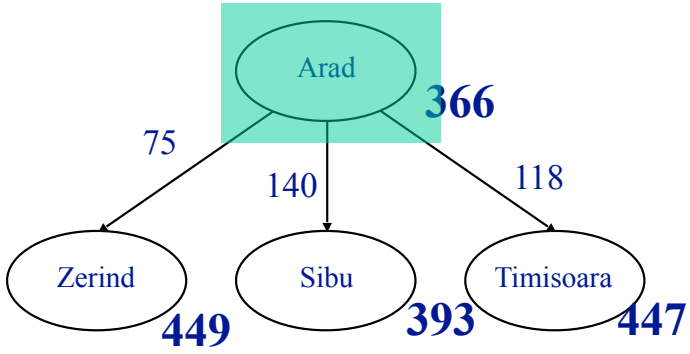
Open = { Arad(366) }

366

g(n): cost so far

f(n)=*g(n)*+*h(n)* = cost so far + estimated cost to goal

A* Search Example: Romania



Open = { Sibu(393), Timisoara(447), Zerind(449) }

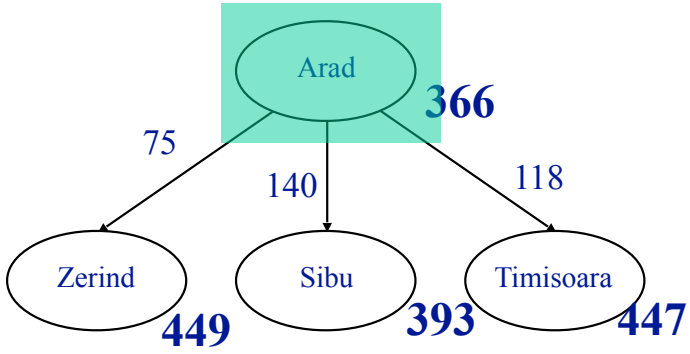
f(Zerid)=75+ 374= 449

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

g(n): cost so far

f(n)=*g(n)*+*h(n)* = cost so far + estimated cost to goal

A* Search Example: Romania



Open = { Sibu(393), Timisoara(447), Zerind(449) }

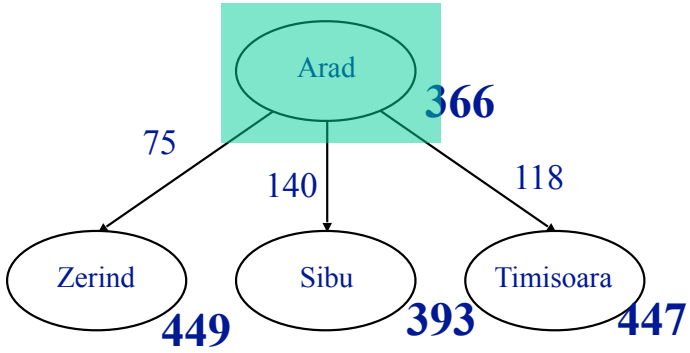
f(Zerid)=75+ 374= 449
f(Sibu)=140+ 253=393

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

g(n): cost so far

f(n)=*g(n)*+*h(n)* = cost so far + estimated cost to goal

A* Search Example: Romania



Open = { Sibu(393), Timisoara(447), Zerind(449) }

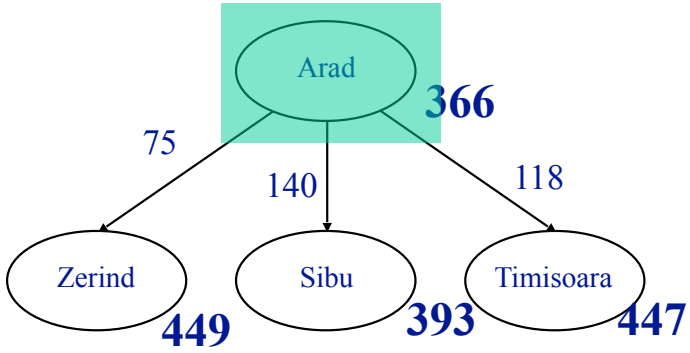
f(Zerid)=75+ 374= 449
f(Sibu)=140+ 253=393
f(Timisoara)=118 + 329 =447

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

g(n): cost so far

f(n)=*g(n)*+*h(n)* = cost so far + estimated cost to goal

A* Search Example: Romania

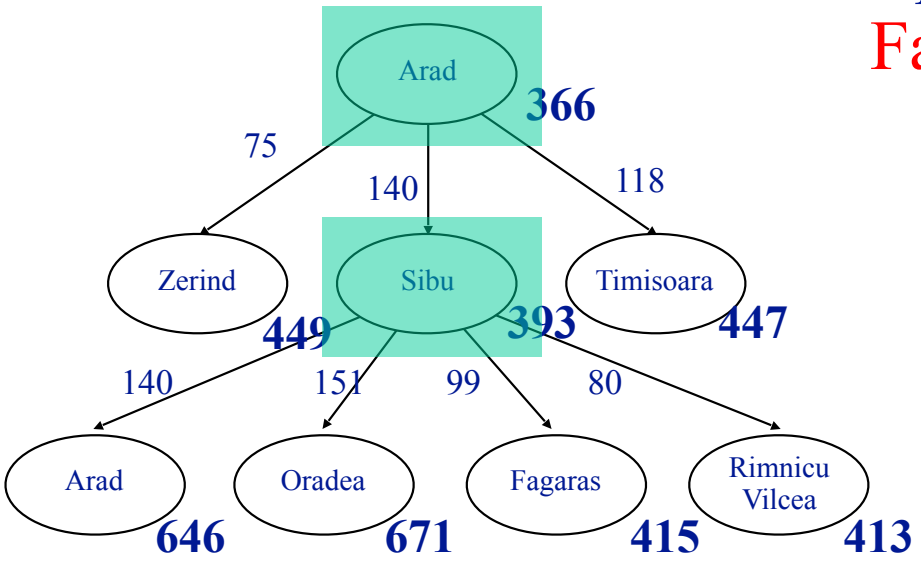


Open = { Sibu(393),
Timisoara(447), Zerind(449) }

g(n): cost so far

f(n)=*g(n)*+*h(n)* = cost so far + estimated cost to goal

A* Search Example: Romania



Open = { RimnicuVilcea(413), Fagaras(415), Timisoara(447), Zerind(449), Arad(646), Oradea(671) }

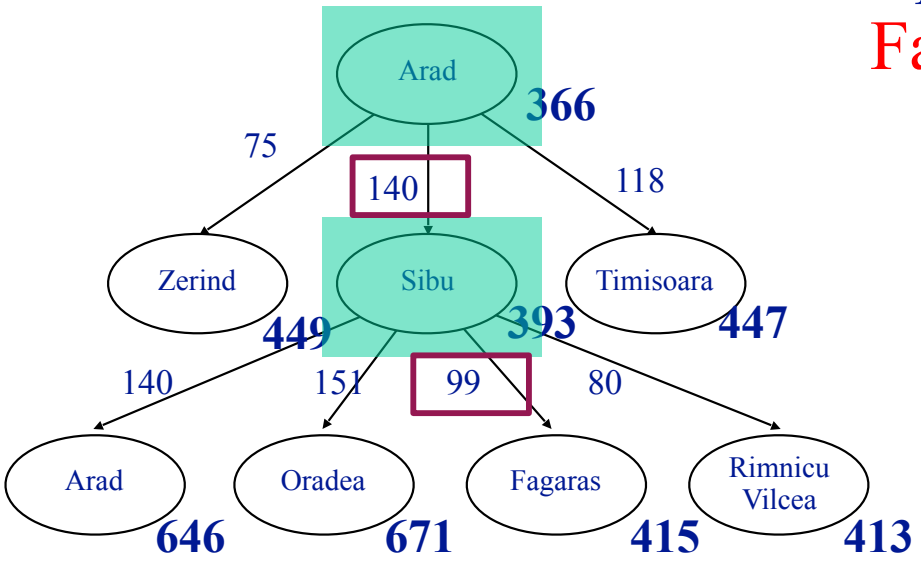
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

f(Fagaras)=(140+99) + 176=415

$g(n)$: cost so far

$\boldsymbol{f(n)}=g(n)+h(n)$ = cost so far + estimated cost to goal

A* Search Example: Romania



Open = { RimnicuVilcea(413), Fagaras(415), Timisoara(447), Zerind(449), Arad(646), Oradea(671) }

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

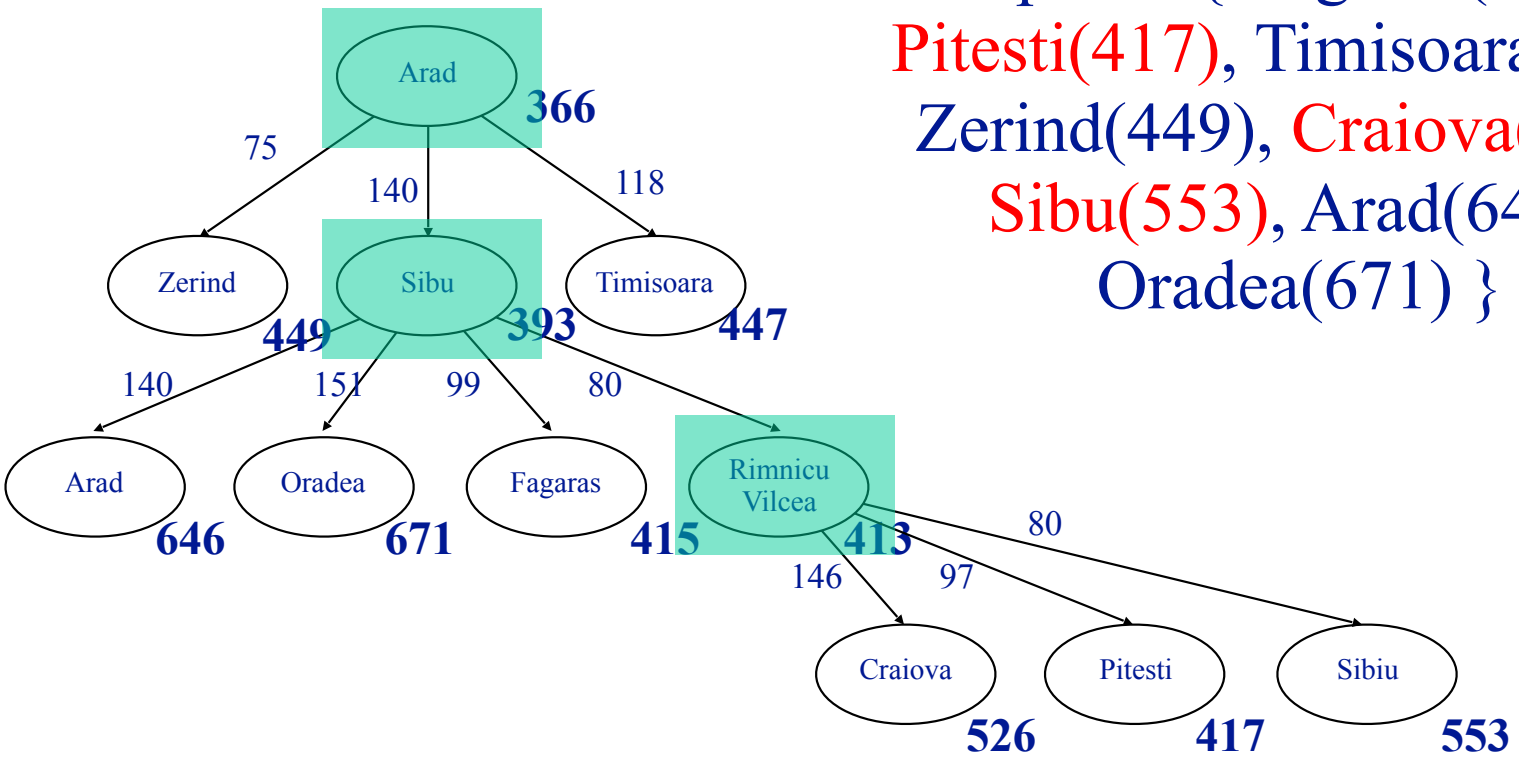
f(Fagaras)=(140+99) + 176=415

$g(n)$: cost so far

$\boldsymbol{f(n)}=g(n)+h(n)$ = cost so far + estimated cost to goal

A* Search Example: Romania

Open = { Fagaras(415), Pitesti(417), Timisoara(447), Zerind(449), Craiova(526), Sibu(553), Arad(646), Oradea(671) }

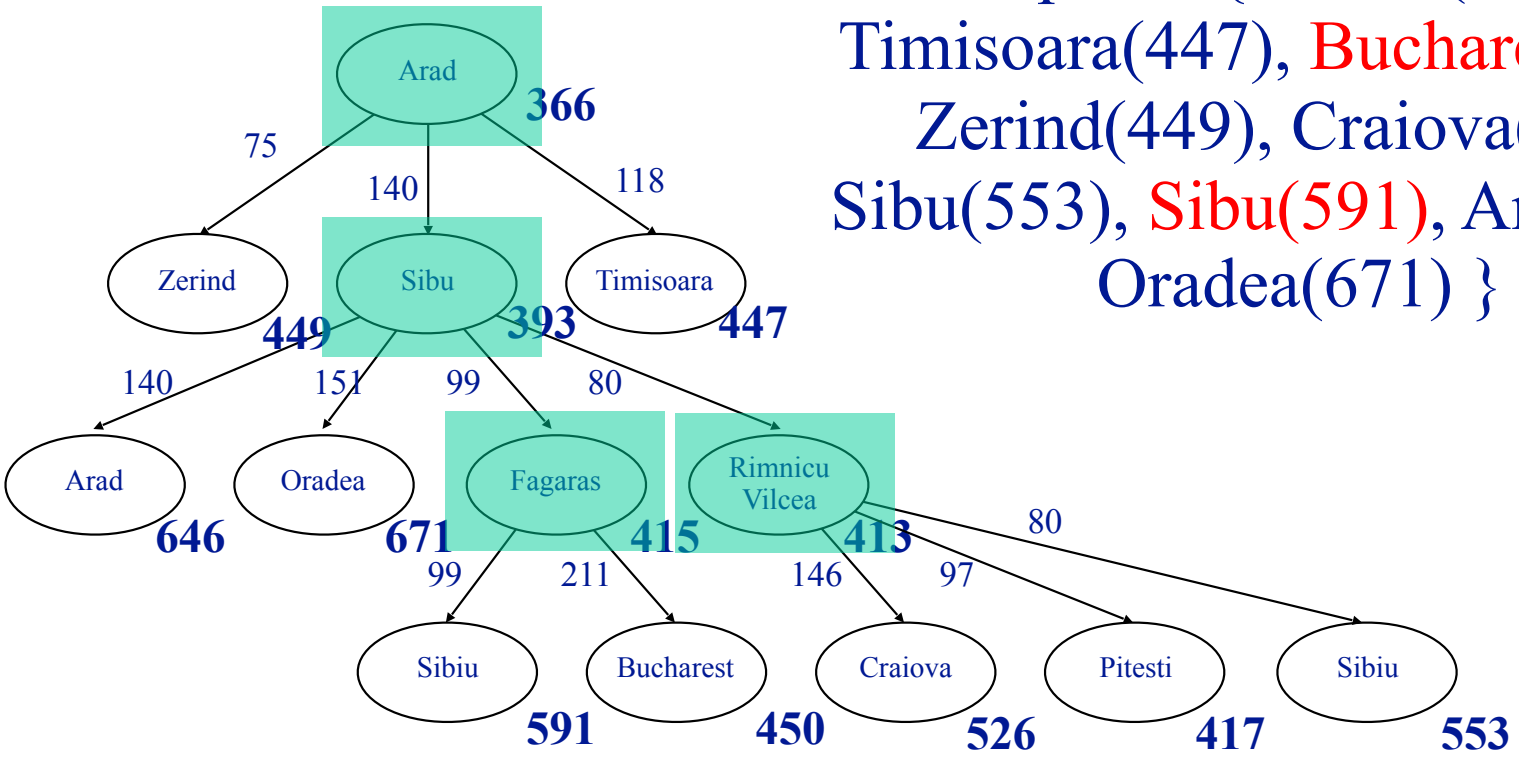


g(n): cost so far

f(n)=*g(n)*+*h(n)* = cost so far + estimated cost to goal

A* Search Example: Romania

Open = { Pitesti(417), Timisoara(447), Bucharest(450), Zerind(449), Craiova(526), Sibu(553), Sibu(591), Arad(646), Oradea(671) }

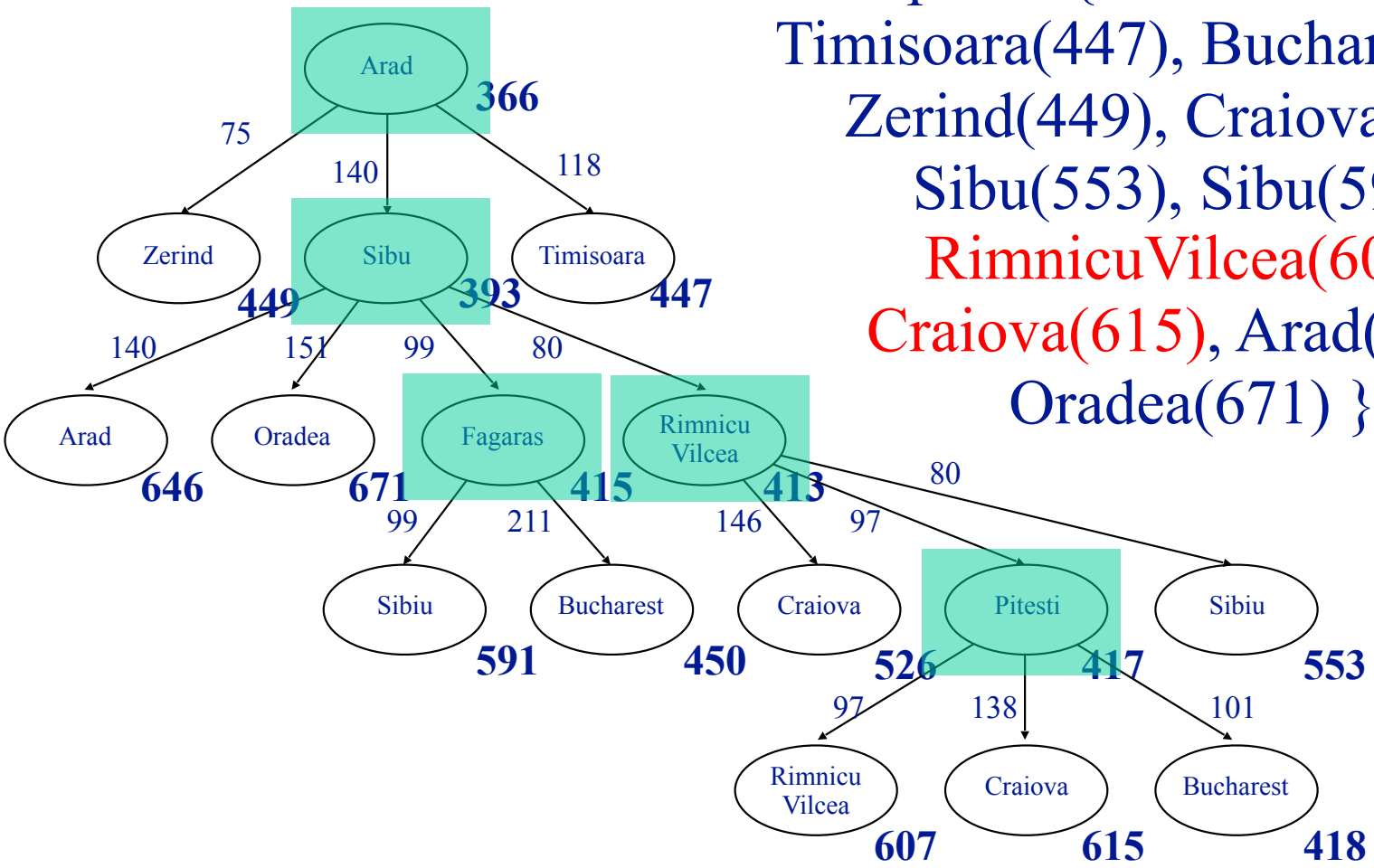


g(n): cost so far

f(n)=*g(n)*+*h(n)* = cost so far + estimated cost to goal

A* Search Example: Romania

Open = { Bucharest(418), Timisoara(447), Bucharest(450), Zerind(449), Craiova(526), Sibu(553), Sibu(591), RimnicuVilcea(607), Craiova(615), Arad(646), Oradea(671) }

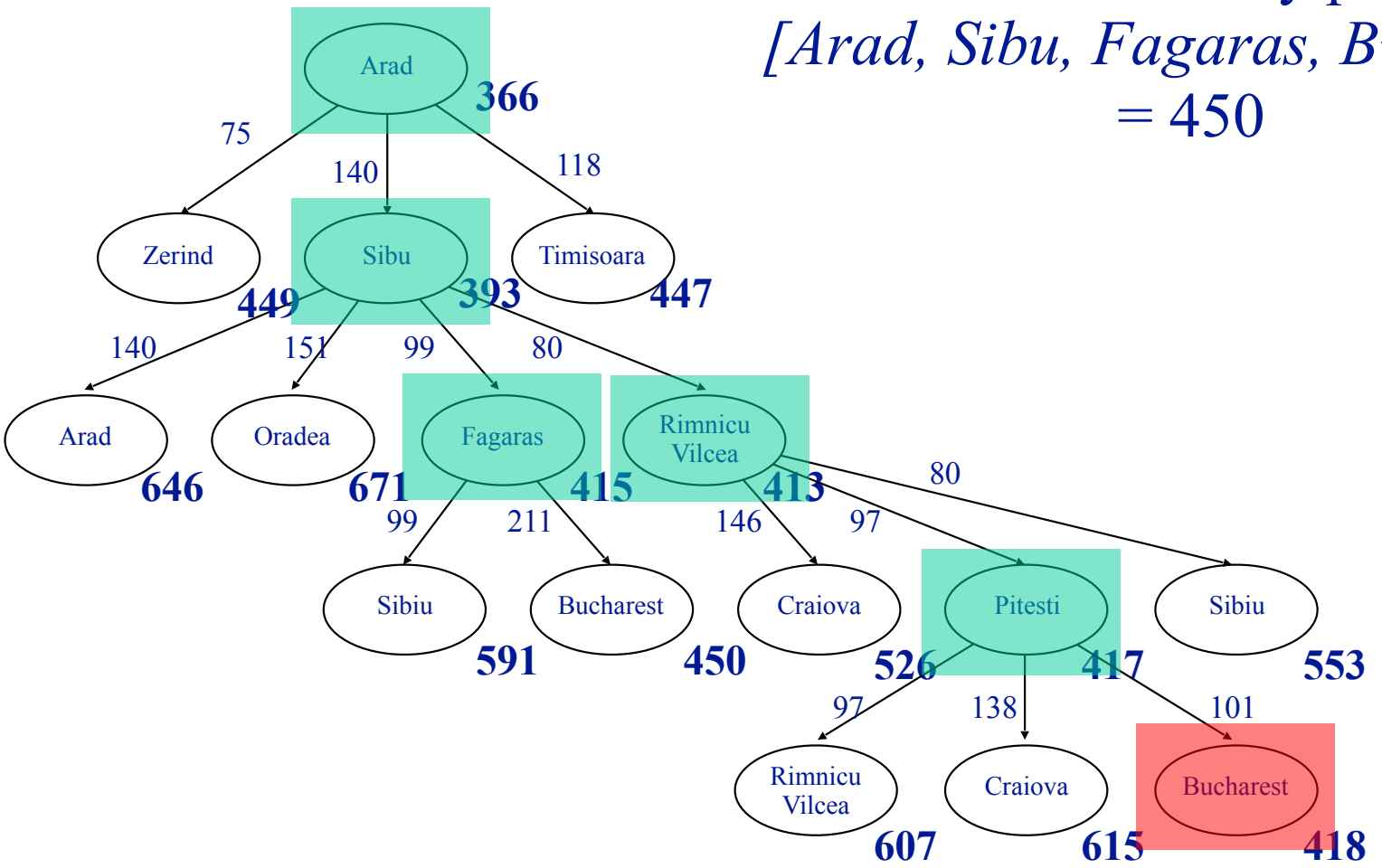


g(n): cost so far

f(n)$=g(n)+h(n)$ = cost so far + estimated cost to goal

A* Search Example: Romania

Recall that Greedy path was
[Arad, Sibu, Fagaras, Bucharest]
= 450

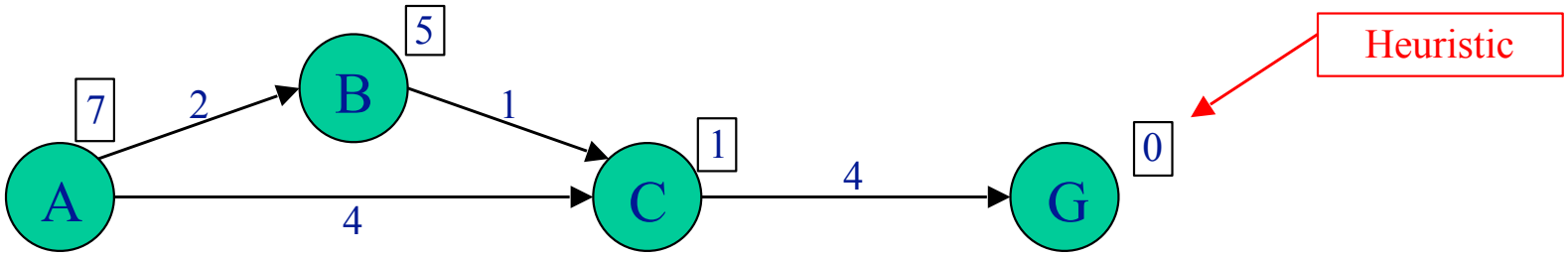


g(n): cost so far

f(n)=*g(n)*+*h(n)* = cost so far + estimated cost to goal

Repeated states in A*

- Contrary to greedy search algorithms, avoiding repeated states can actually harm A*
- Consider:



A(0+7=7) B(2+5=7) C(4+1=5) G(4+4+0=8)

- Goal is discovered before finding shortest path to it!
- Depending on algorithm construction, could return sub-optimal path
  - At best, devolves to BFS-like performance

Consistent Heuristics

– Consistency: The difference in heuristic values between two connected states is always smaller than (or equal to) the actual cost:

$$cost(x, y) \geq \left| h(x) - h(y) \right|$$

For two connected states x and y

Consistent Heuristics

- Consistency: The difference in heuristic values between two connected states is always smaller than (or equal to) the actual cost:

$$cost(x, y) \geq \left| h(x) - h(y) \right|$$

For two connected states x and y

- Consistency ensures that the total estimated path cost always increases (or stays the same) as we get closer to a goal state
  - For this reason, sometimes also called a monotonic heuristic
  - Guarantees that shortest path is always expanded first

Properties of A*

- Complete (good)
  - Unless infinitely many nodes
- Time and space (not good)
  - Still exponential in worst case (keeps all nodes in memory)
  - Strongly depends on choice of heuristic
- Optimal (good)
  - Expands fewest nodes