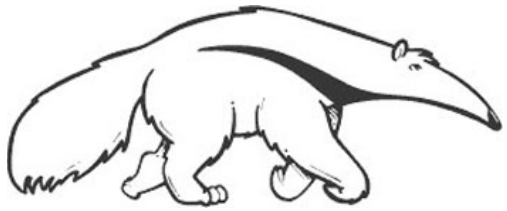
Heuristic search: GBFS, A*

CS171, Winter Quarter, 2019
Introduction to Artificial Intelligence
Prof. Richard Lathrop



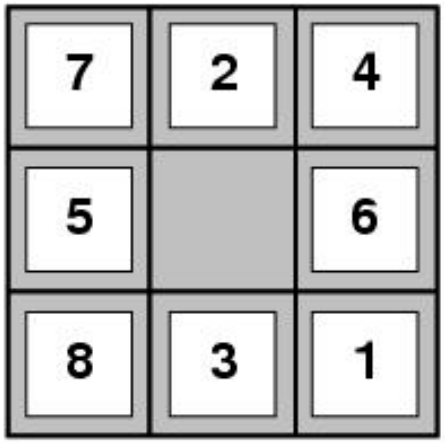
Read Beforehand: R&N 3.5-3.7

Outline

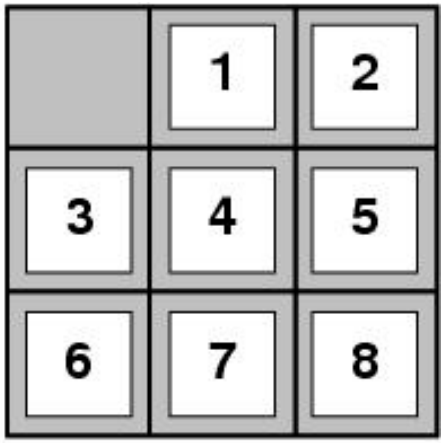
- Review limitations of uninformed search methods
- **Informed (or heuristic) search**
- **Problem-specific heuristics to improve efficiency**
  - Best-first, A* (and if needed for memory limits, RBFS, SMA*)
  - Techniques for generating heuristics
  - A* is optimal with admissible (tree)/consistent (graph) heuristics
  - A* is quick and easy to code, and often works ***very*** well
- **Heuristics**
  - A structured way to add "smarts" to your solution
  - Provide ***significant*** speed-ups in practice
  - Still have worst-case exponential time complexity

In AI, "NP-Complete" means "Formally interesting"

Limitations of uninformed search



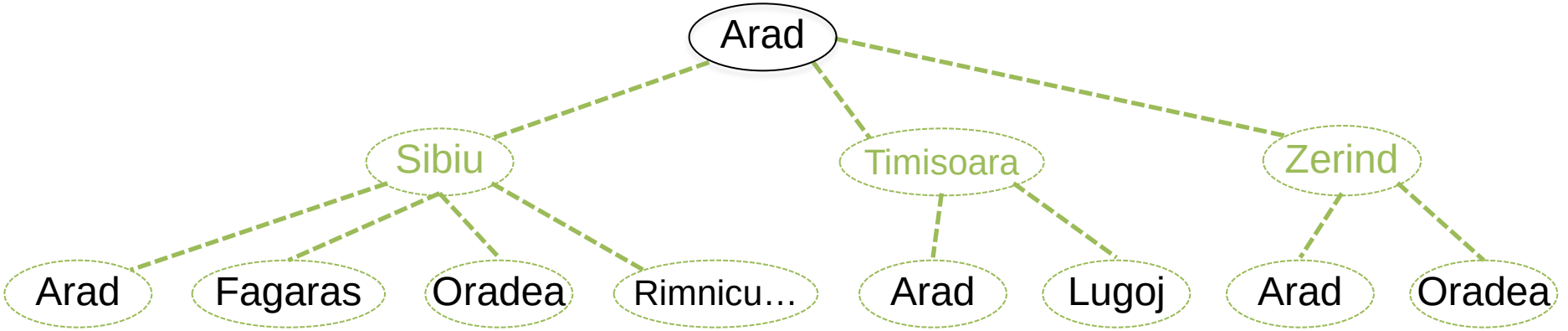
Start State



Goal State

- Search space size makes search tedious
  - Combinatorial explosion
- Example: 8-Puzzle
  - Average solution cost is ~ 22 steps
  - Branching factor ~ 3
  - Exhaustive search to depth 22: 3.1×10^{10} states
  - 24-Puzzle: 10^{24} states (much worse!)

Recall: tree search



function TREE-SEARCH (*problem*, *strategy*) : returns a so
    initialize the search tree using the initial state of *prob*
    while (true):
        if no candidates for expansion: return failure
        choose a leaf node for expansion according to *strategy*
        if the node contains a goal state: return the corresponding solution
        else: expand the node and add the resulting nodes to the search tree

This "strategy" is what differentiates different search algorithms

Heuristic function

- Idea: use a heuristic function h(n) for each node
  - *g(n)* = known path cost so far to node n
  - *h(n)* = estimate of optimal cost to goal from node n
  - *f(n) = g(n)+h(n)* = estimate of total cost to goal through n
  - *f(n)* provides an estimate for the total cost

- "Best first" search implementation
  - Order the nodes in frontier by an evaluation function
  - Greedy Best-First: order by *h(n)*
  - A* search: order by *f(n)*

- Search efficiency depends heavily on heuristic quality!
  - **The better your heuristic, the faster your search!**

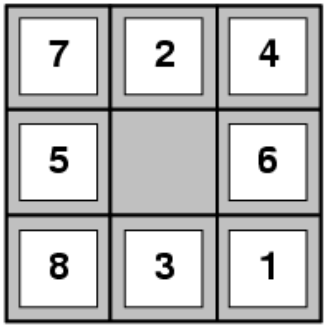
Heuristic function

- Heuristic
  - Definition: a commonsense rule or rules intended to increase the probability of solving some problem
  - Same linguistic root as ancient Greek "Eureka" = "I have found it"
    - "Eureka" = motto of the state of California
  - Using "rules of thumb" (= guesstimates) to find answers

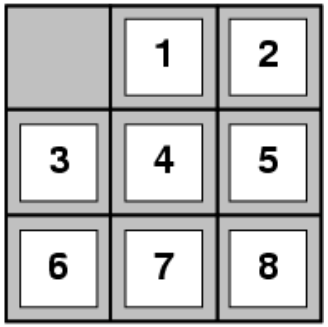
- Heuristic function h(n)
  - Estimate of (optimal) remaining cost from *n* to goal
  - Defined using only the state of node *n*
  - **NOTE: *h(n) = 0* if *n* is a goal node**
  - Example: straight line distance from *n* to Bucharest
    - Not true state space distance, just an estimate! Actual distance can be higher

- Provides problem-specific knowledge to the search algorithm

Example: 8-Puzzle



Start State



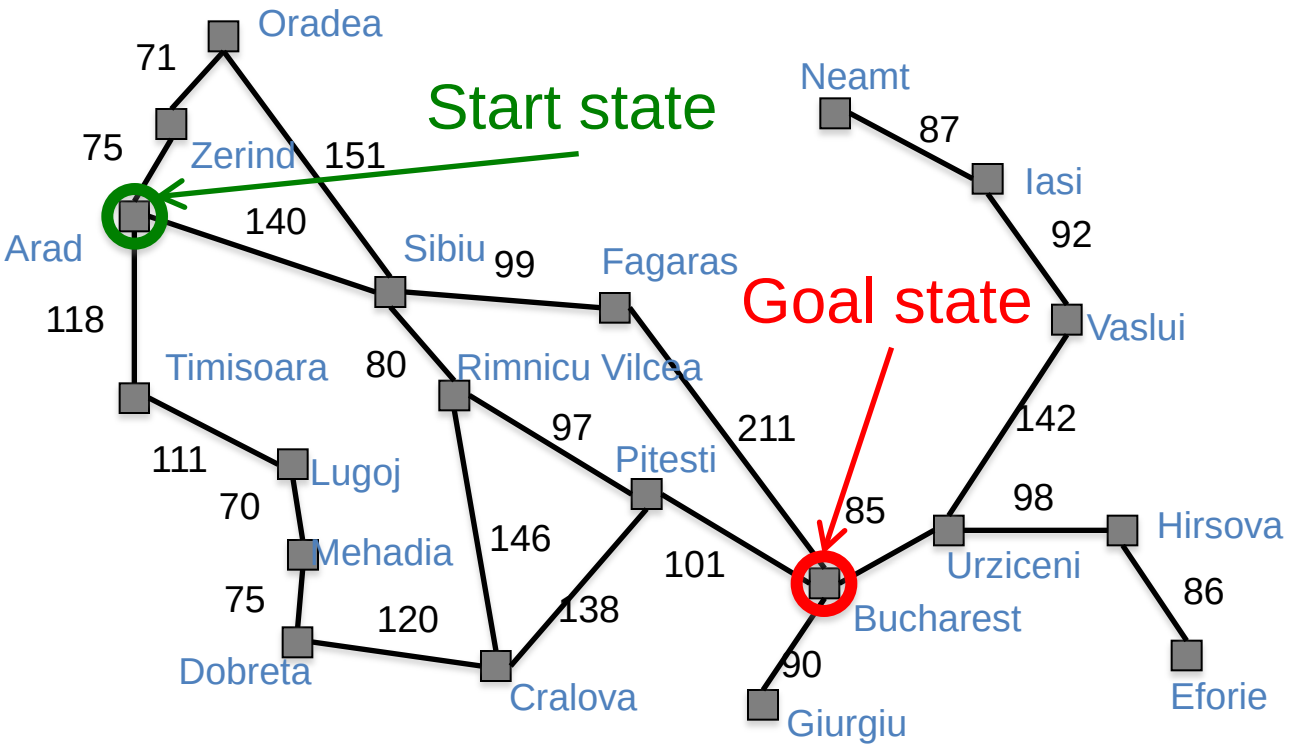
Goal State

- 8-Puzzle
  - Avg solution cost is about 22 steps
  - Branching factor ~ 3
  - Exhaustive search to depth 22 = 3.1 x 10^10 states
  - A good heuristic function can reduce the search process
- Two commonly used heuristics
  - h_1: the number of misplaced tiles

    $h_1(s) = 8$
  - h_2: sum of the distances of the tiles from their goal ("Manhattan distance")

    $h_2(s) = 3+1+2+2+2+3+3+2$
    $= 18$

Example: Romania, straight-line distance



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

How to implement straight-line distance as a heuristic function *h(n)*?

- Heuristic function *h(n)* is called with node *n*.
- For example (your implementation may vary)
  - *h(n)* has a static hash table that maps city string names to straight-line distances shown in the table
  - Node *n* contains the city string name of its state
  - *h(n)* uses the state city string name as a key into the hash table to retrieve its straight-line distance
  - That straight-line distance is the return value of *h(n)*

Relationship of search algorithms

- Notation:
  - *g(n)* = known cost so far to reach *n*
  - *h(n)* = estimated optimal cost from *n* to goal
  - *h*(n)* = true optimal cost from *n* to goal (unknown to agent)
  - *f(n) = g(n)+h(n)* = estimated optimal total cost through *n*
- Uniform cost search: sort frontier by *g(n)*
- Greedy best-first search: sort frontier by *h(n)*
- A* search: sort frontier by *f(n) = g(n) + h(n)*
  - Optimal for admissible / consistent heuristics
  - Generally the preferred heuristic search framework
  - Memory-efficient versions of A* are available: RBFS, SMA*

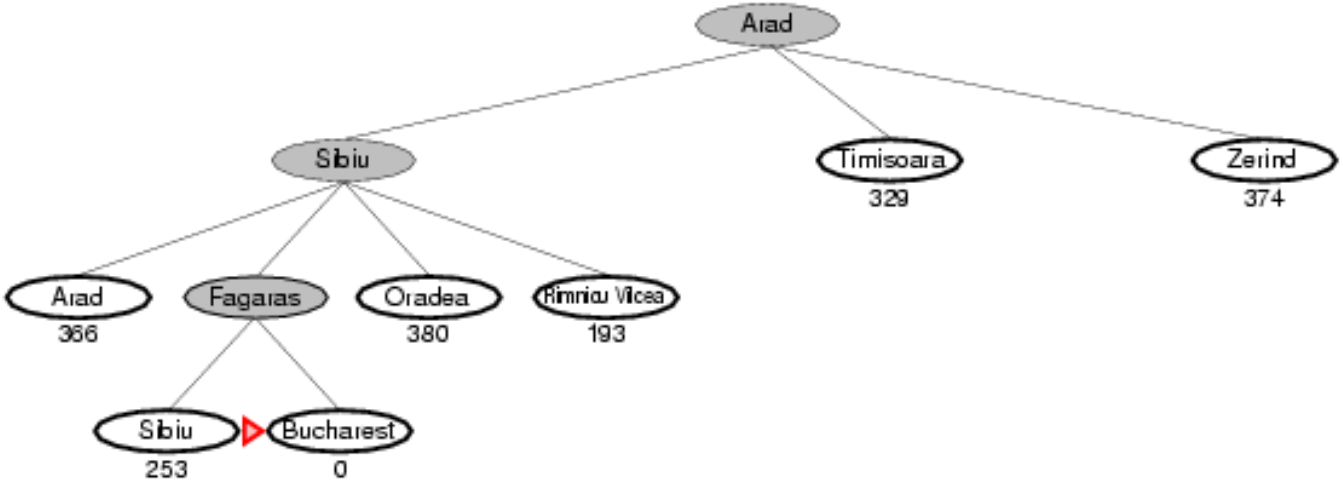
Greedy best-first search

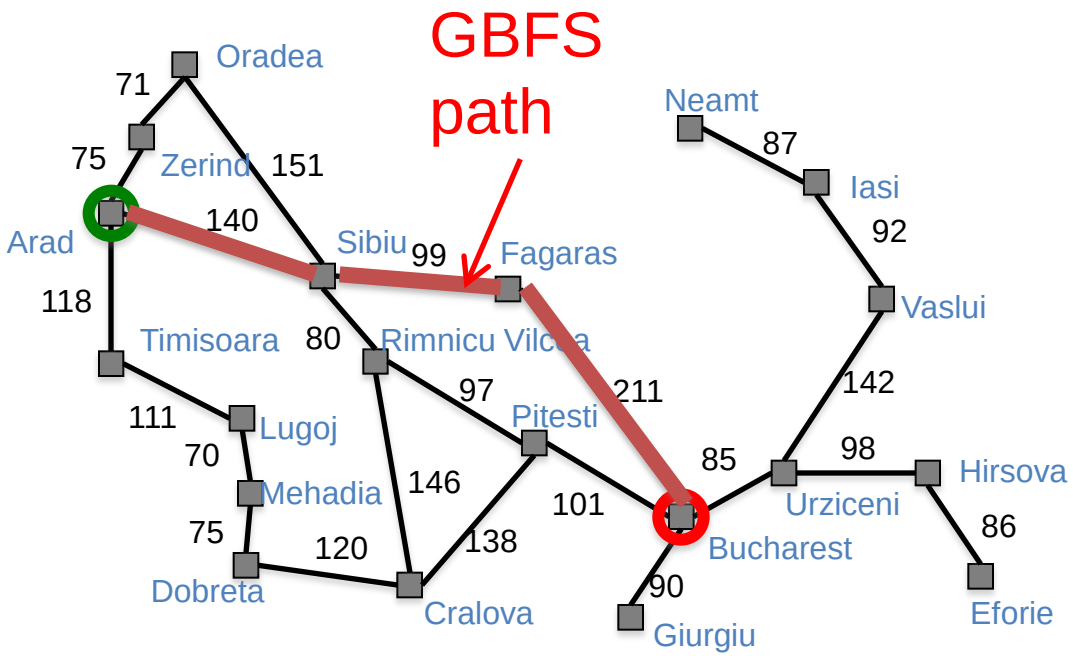
(sometimes just called “best-first”)

- *h(n)* = estimate of cost from *n* to goal
  - Example: *h(n)* = straight line distance from n to Bucharest

- Greedy best-first search expands the node that appears to be closest to goal
  - Priority queue sort function = *h(n)*

Example: GBFS for Romania



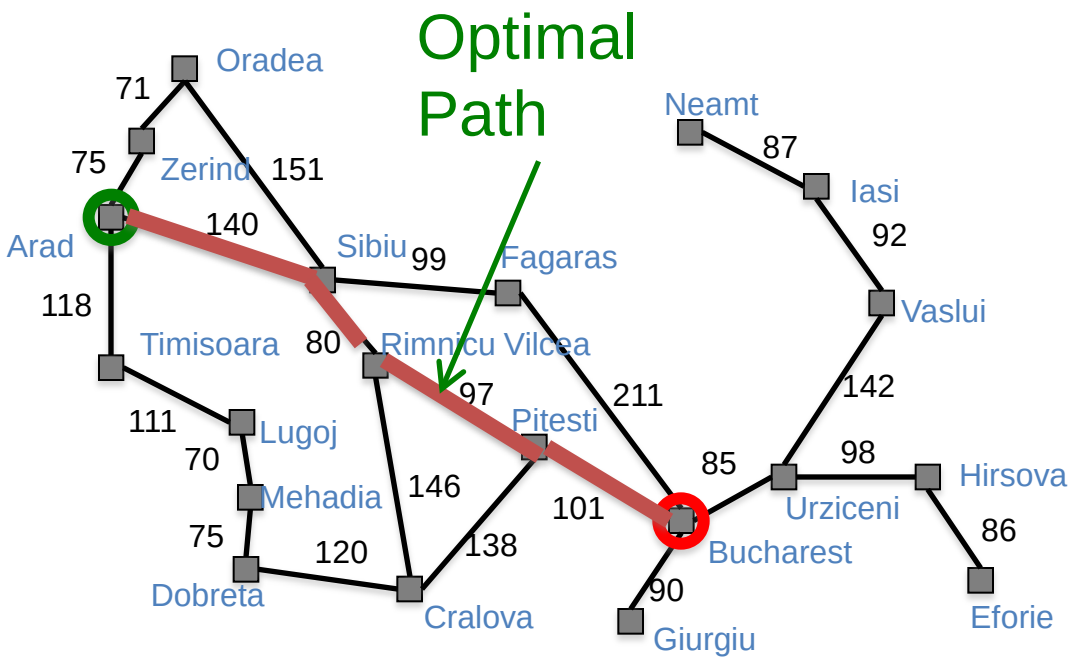


Straight-line dist to goal	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Example: GBFS for Romania

GBFS path: 450km

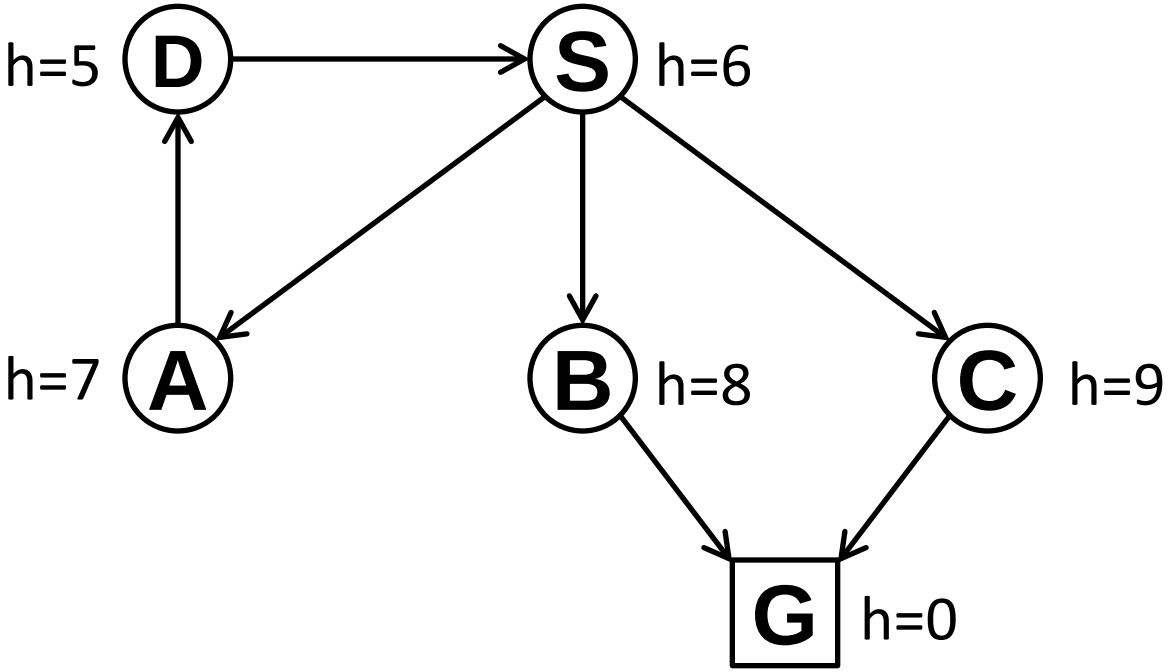
Optimal path: 418 km



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

Greedy best-first search

- With tree-search, will become stuck in this loop:
  - Order of node expansion: S A D S A D S A D …
  - Path found: none
  - Cost of path found: none



Properties of greedy best-first search

- Complete?
  - Tree version can get stuck in loops (moderately easy to prevent)
  - Graph version is complete in finite spaces
  - Neither Tree nor Graph version is complete in infinite spaces
    - Easy for a malicious demon to lead it astray with evil heuristic values
- Time? $O(b^m)$
  - A good heuristic can give dramatic improvement
- Space? $O(b^m)$
  - Keeps all nodes in memory
- Optimal? No
  - Example:

    Arad – Sibiu – Rimnicu Vilcea – Pitesti – Bucharest is shorter!

A* search

- Idea: avoid expanding paths that are already expensive
  - Generally the preferred (simple) heuristic search
  - Optimal if heuristic is:
    - admissible (tree search) / consistent (graph search)
  - Always terminates with a solution path (even if heuristic is not admissible) if step costs $\geq \varepsilon > 0$ and branching factor is finite
    - proof by Hart, Nilsson, and Raphael (1968)
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ = cost so far to reach n
  - $h(n)$ = estimated cost from n to goal
  - $f(n) = g(n)+h(n) =$ estimated total cost of path through n to goal

- A* algorithm is identical to UCS except that the priority queue sort function is $f(n) = g(n) + h(n)$

Admissible heuristics

- A heuristic *h(n)* is admissible if, for every node n,

  $h(n) \leq h^*(n)$

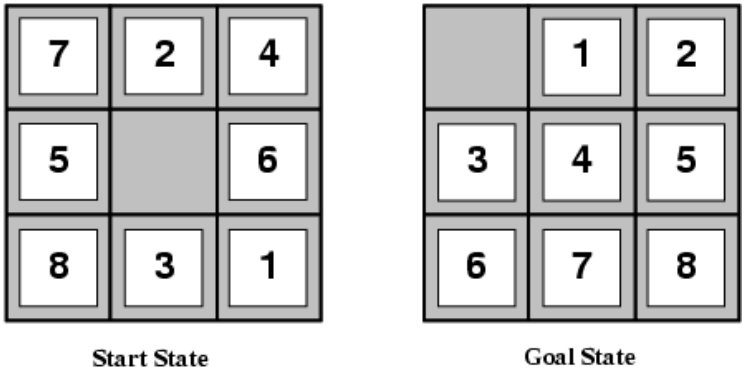
  $h^*(n)$ = the true cost to reach the goal state from *n*

- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is never pessimistic
  - Example: straight-line distance never overestimates the actual road distance

- **Theorem:**

  if *h(n)* is admissible, A* using Tree-Search is optimal

Example: Admissible heuristics



- Two commonly used admissible heuristics
  - h_1: the number of misplaced tiles

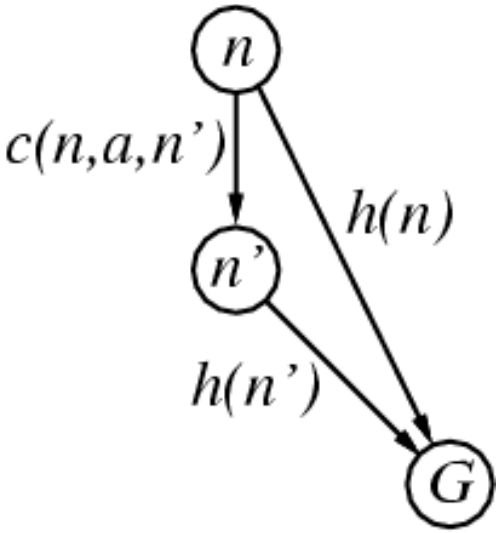
    $h_1(s) = 8$
  - h_2: sum of the distances of the tiles from their goal ("Manhattan distance")

    $h_2(s) = 3+1+2+2+2+3+3+2$
    $= 18$

Consistent heuristics

- A heuristic is consistent (or monotone) if for every node *n*, every successor *n'* of *n* generated by any action *a*,

$$c(n,a,n') + h(n') \geq h(n)$$



(Triangle inequality)

- Thus, if *h(n)* is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n,a,n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

  i.e., *f(n)* is non-decreasing along any path.

- Consistent $\Rightarrow$ admissible (consistency is a stronger condition)

- **Theorem:** If *h(n)* is consistent, A* using Graph-Search is optimal

Consistent $\Rightarrow$ Admissible

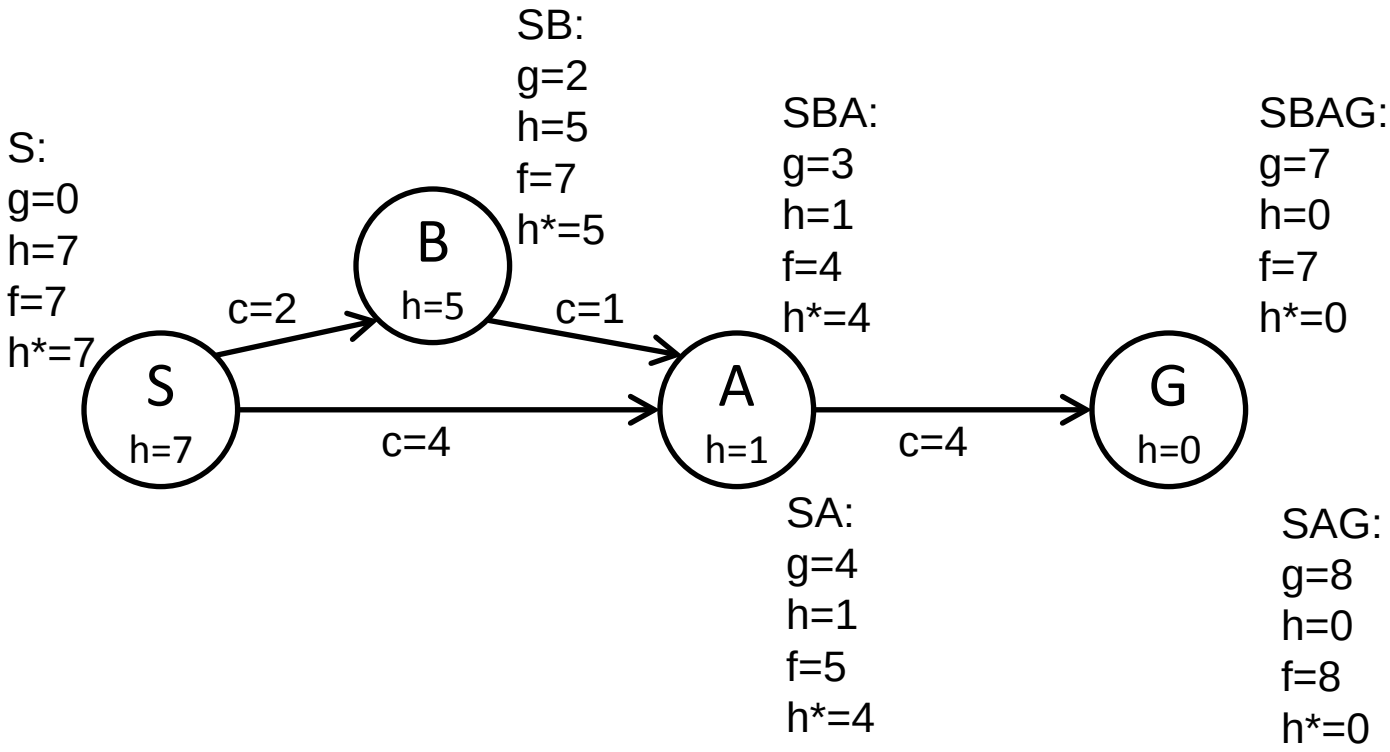
- **<u>Any Consistent heuristic is also Admissible.</u>**
  - Let n be a node, n' be a successor of n, and the function h() be a consistent heuristic
  - By definition of consistent, $c(n,a,n') \geq h(n) - h(n')$
  - Basis step: let n' be a goal node G, then $h(n') = 0$
    - Then $h^*(n) = C(n,a,G) \geq h(n) - 0 = h(n)$
    - Thus, $h^*(n) \geq h(n)$ so h() is admissible at n
  - Induction step: let n' be such that $h^*(n') \geq h(n')$
    - Then $h(n) \leq c(n,a,n') + h(n') \leq c(n,a,n') + h^*(n') = h^*(n)$
    - Thus, $h^*(n) \geq h(n)$ so h() is admissible at n
  - Consequently, h() is admissible at every node

Admissible $\not\Rightarrow$ Consistent

- **Most Admissible heuristics are Consistent**
  - "one has to work quite hard to concoct heuristics that are admissible but not consistent." — R&N p. 95
- 8-puzzle: Admissible, but not Consistent, heuristic
  - $h_1(n)$ = number of misplaced tiles
  - $h_2(n)$ = sum of (Manhattan) distances of tiles to goal
  - $h_{ex}(n)$ = Choose_randomly($h_1(n)$, $h_2(n)$)
    - $h_{ex}(n)$ is admissible but not (necessarily) consistent
    - $h_{ex}(n)$ is (probably) not non-decreasing along all paths
    - $h_1(n)$ and $h_2(n)$ are not necessarily related to each other
    - Random combination may not satisfy triangle inequality

Example adapted from "Inconsistent Heuristics in Theory and Practice" by Felner, Zahavi, Holte, Schaeffer, Sturtevant, & Zhang

Example: Admissible, but not Consistent, heuristic

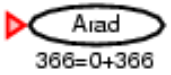


- *h(n)* is admissible because $h(n) \leq h^*(n)$ for all n
- *h(n)* is NOT consistent because paths SA and BA are not monotone (i.e., *f(.)* is NOT non-decreasing)

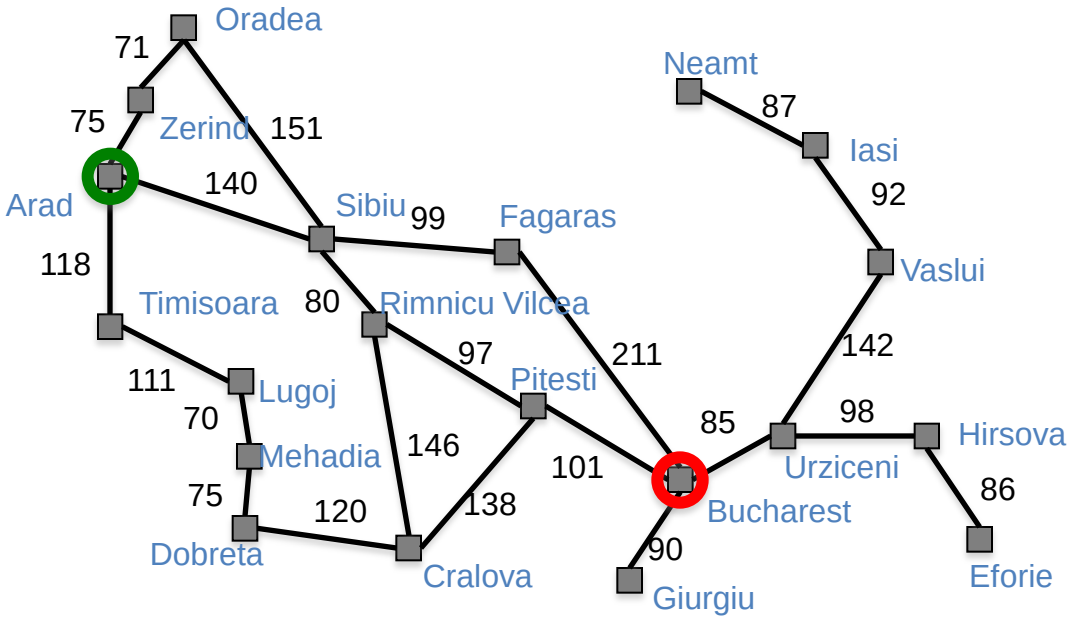
Optimality conditions

- A* Tree Search is optimal if heuristic is admissible
- A* Graph Search is optimal if heuristic is consistent
- Why two different conditions?
  - In graph search you often find a long cheap path to a node after a short expensive one, so you might have to update all of its descendants to use the new cheaper path cost so far
  - A consistent heuristic avoids this problem (it can't happen)
  - Consistent is slightly stronger than admissible
  - Almost all admissible heuristics also are consistent
- Could we do optimal A* Graph Search with an admissible heuristic?
  - Yes, but we would have to do additional work to update descendants when a cheaper path to a node is found
  - A consistent heuristic avoids this problem

Example: A* Tree Search for Romania



Red triangle:
Node to expand next



Straight-line dist to goal	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Fagaras	176
Lugoj	244
Mehadia	241
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Zerind	374

(Simulated queue)

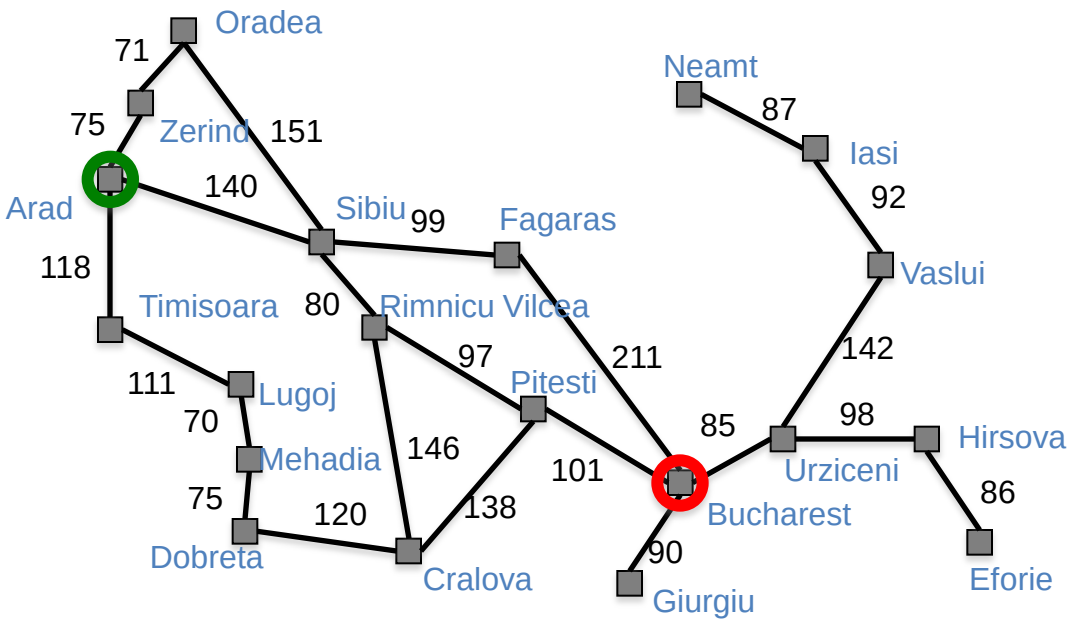
Example: A* Tree Search for Romania

Expanded: None

Children: None

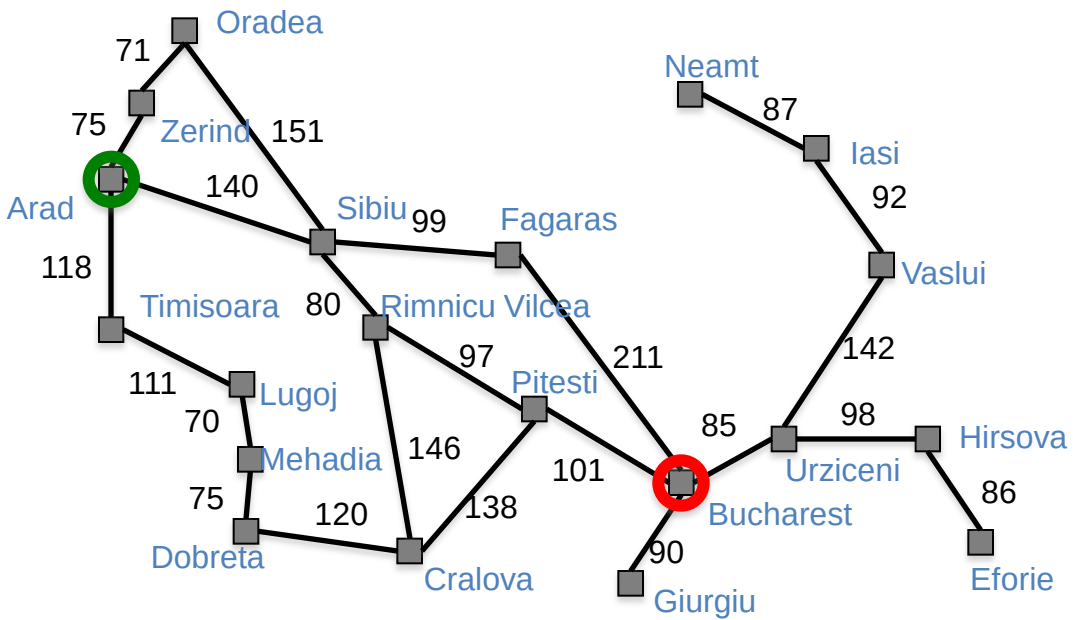
Frontier: Arad/366 (0+366),

Red name:
Node to expand next



Straight-line dist to goal	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Fagaras	176
Lugoj	244
Mehadia	241
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Zerind	374

Example: A* Tree Search for Romania



Straight-line dist to goal	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Fagaras	176
Lugoj	244
Mehadia	241
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Zerind	374

(Simulated queue)

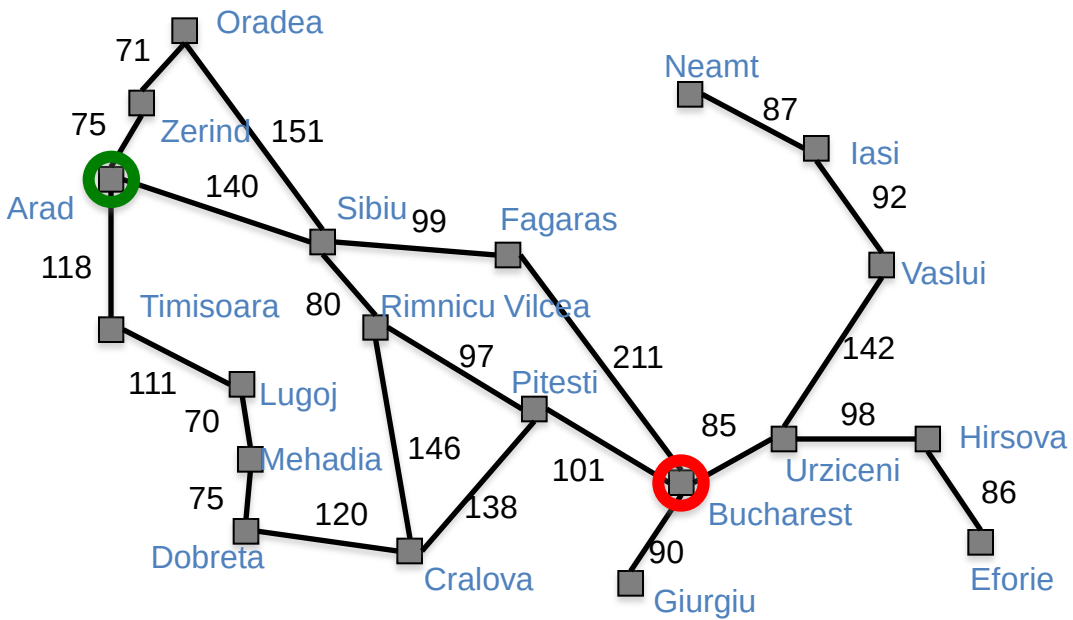
Example: A* Tree Search for Romania

Expanded: Arad/366 (0+366)

Underlined node:
Last node expanded

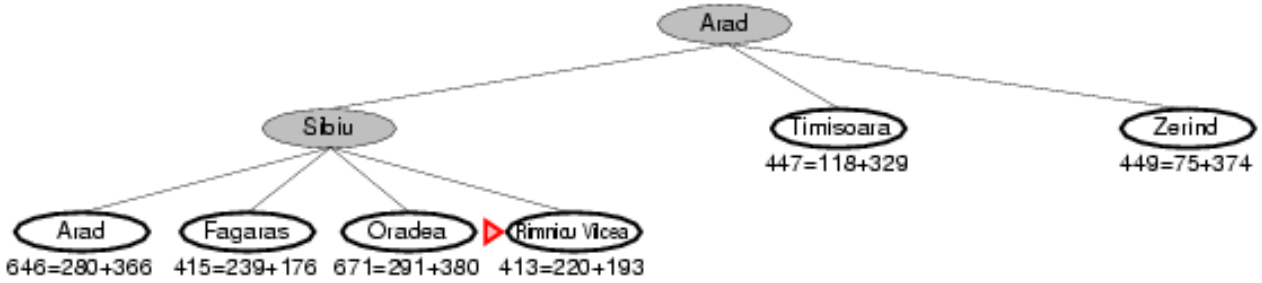
Children: Sibiu/393 (140+253), Timisoara/447 (118+329), Zerind/449 (75+374)

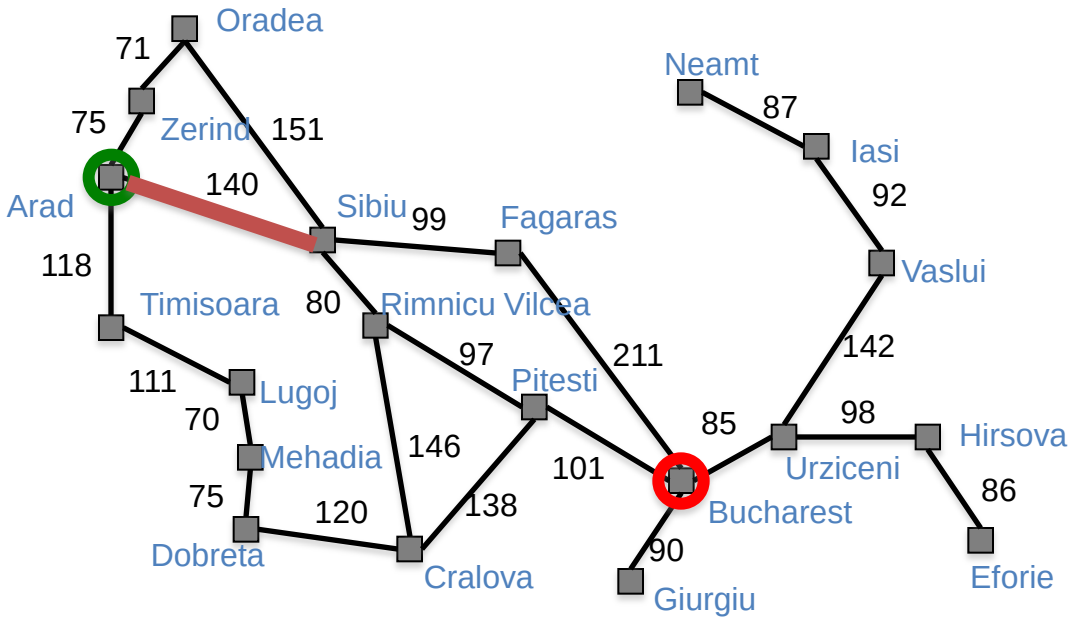
Frontier: ~~Arad/366 (0+366)~~, Sibiu/393 (140+253), Timisoara/447 (118+329), Zerind/449 (75+374)



Straight-line dist to goal	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Fagaras	176
Lugoj	244
Mehadia	241
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Zerind	374

Example: A* Tree Search for Romania





Straight-line dist to goal	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Fagaras	176
Lugoj	244
Mehadia	241
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Zerind	374

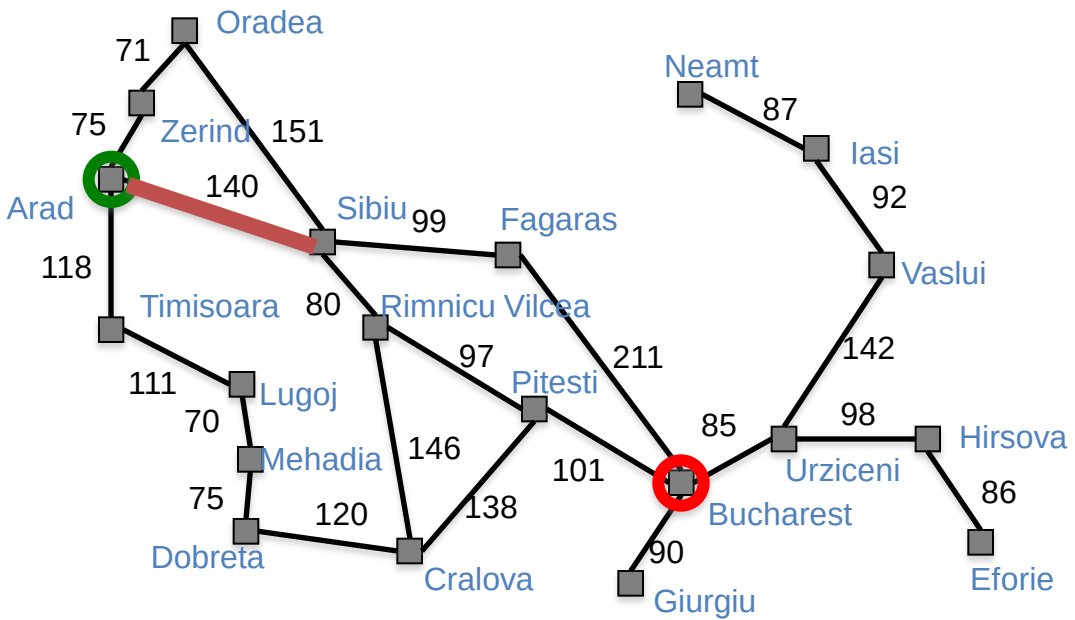
(Simulated queue)

Example: A* Tree Search for Romania

Expanded: Arad/366 (0+366), Sibiu/393 (140+253)

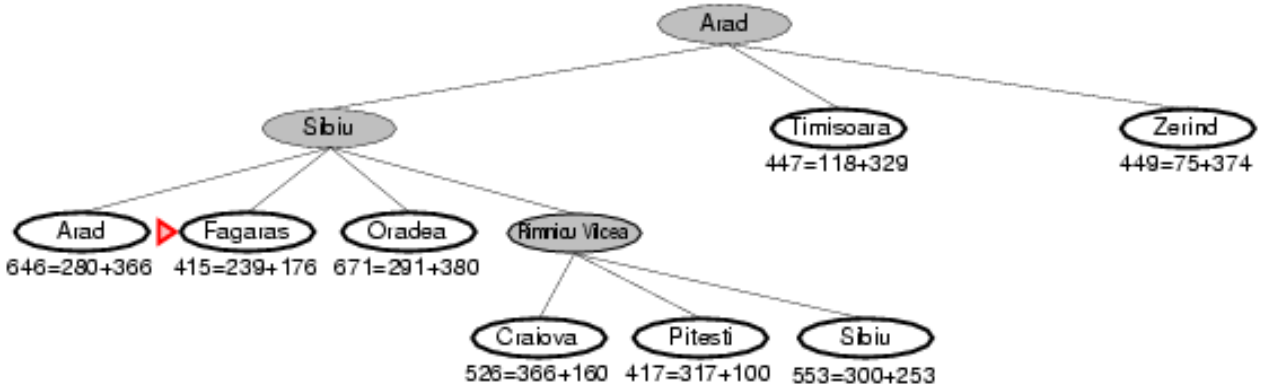
Children: Arad/646 (280+366), Fagaras/415 (239+176), Oradea/671 (291+380), RimnicuVilcea/413 (220+193)

Frontier: ~~Arad/366 (0+366)~~, ~~Sibiu/393 (140+253)~~, Timisoara/447 (118+329), Zerind/449 (75+374), Arad/646 (280+366), Fagaras/415 (239+176), Oradea/671 (291+380), RimnicuVilcea/413 (220+193)

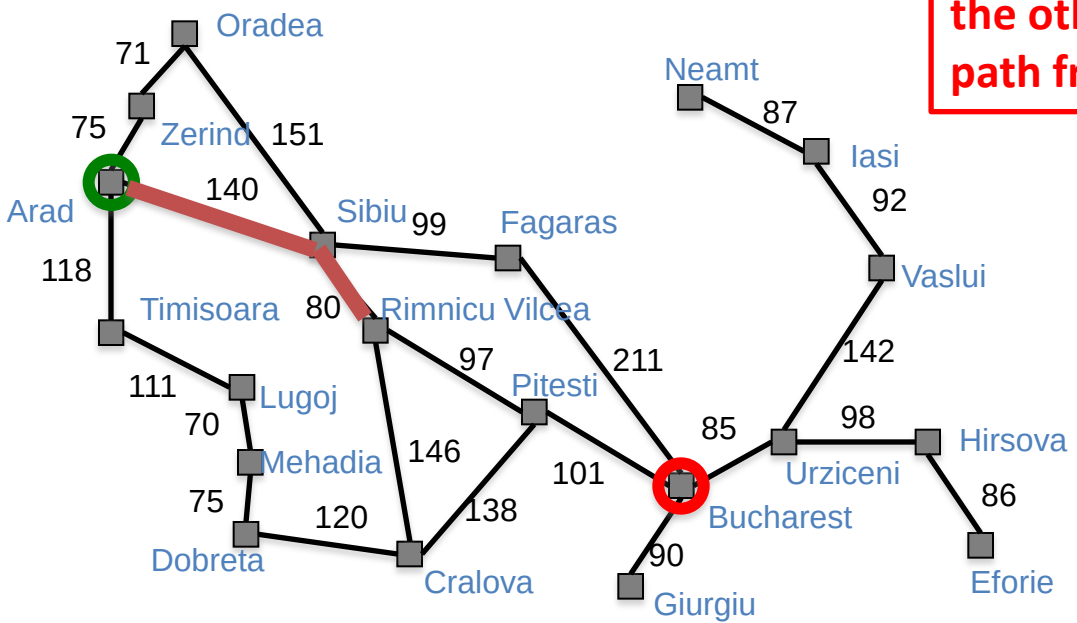


Straight-line dist to goal	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Fagaras	176
Lugoj	244
Mehadia	241
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Zerind	374

Example: A* Tree Search for Romania



The loop at Sibiu could be detected by noticing the other Sibiu on the path from child to root.



Straight-line dist to goal	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Fagaras	176
Lugoj	244
Mehadia	241
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Zerind	374

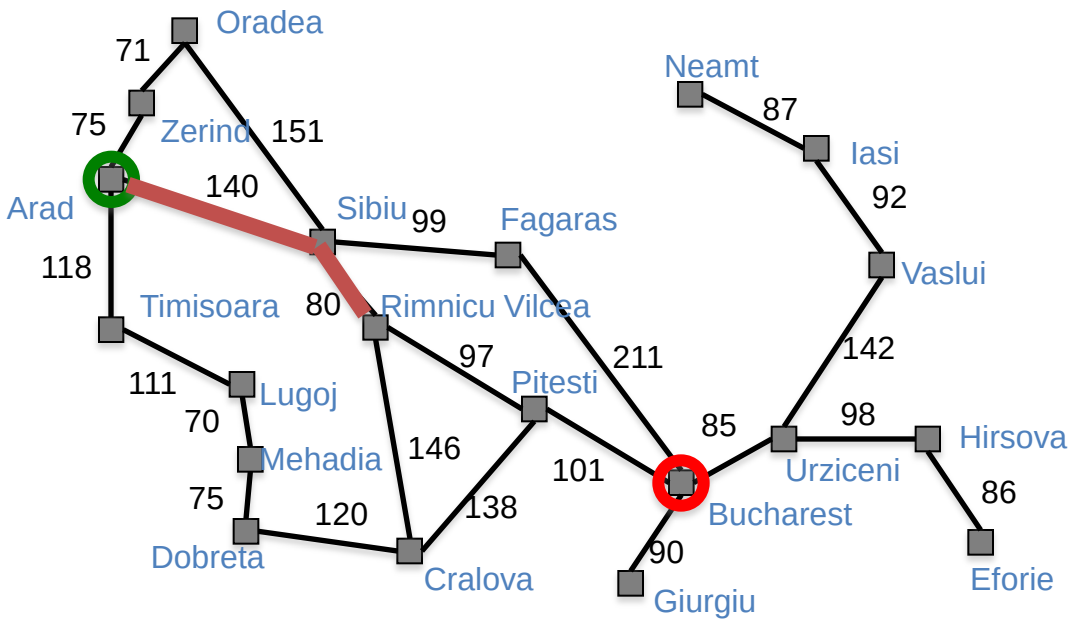
(Simulated queue)

Example: A* Tree Search for Romania

Expanded: Arad/366 (0+366), Sibiu/393 (140+253), RimnicuVilcea/413 (220+193)

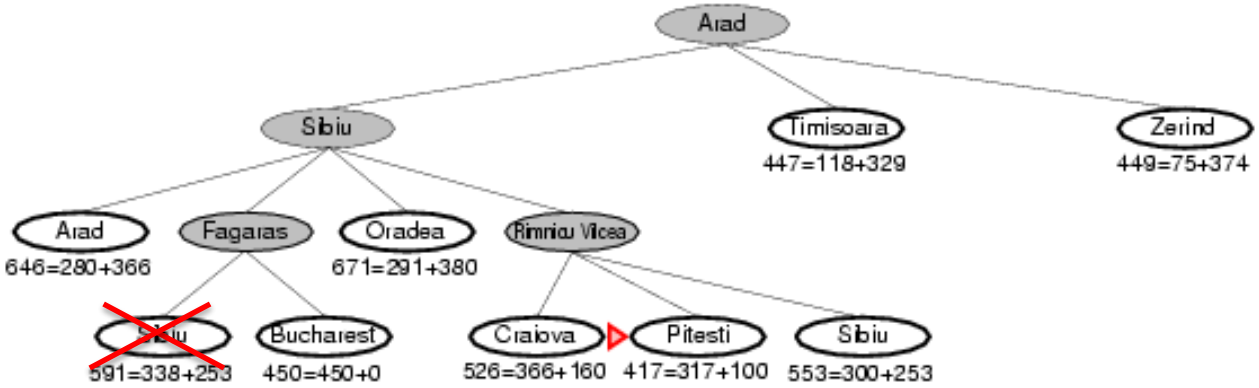
Children: Craiova/526 (366+160), Pitesti/417 (317+100), Sibiu/553 (300+253)

Frontier: ~~Arad/366 (0+366)~~, ~~Sibiu/393 (140+253)~~, Timisoara/447 (118+329), Zerind/449 (75+374), Arad/646 (280+366), Fagaras/415 (239+176), Oradea/671 (291+380), ~~RimnicuVilcea/413 (220+193)~~, Craiova/526 (366+160), Pitesti/417 (317+100), Sibiu/553 (300+253)

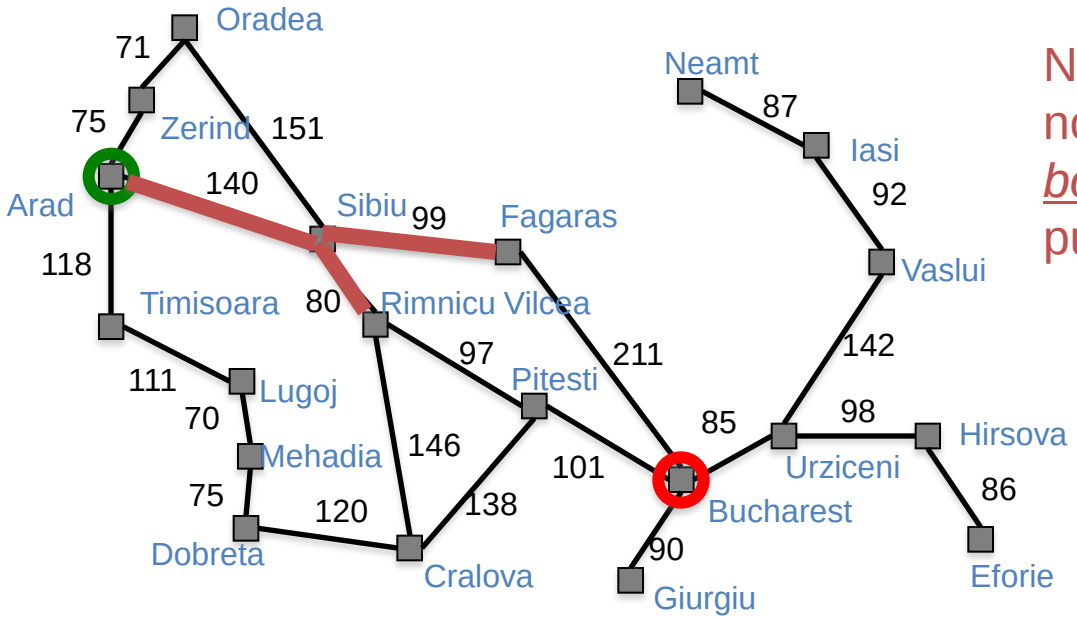


Straight-line dist to goal	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Fagaras	176
Lugoj	244
Mehadia	241
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Zerind	374

Example: A* Tree Search for Romania



Remove the higher-cost of identical nodes.



Note: search does not "backtrack"; *both* routes are pursued at once.

Straight-line dist to goal	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Fagaras	176
Lugoj	244
Mehadia	241
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Zerind	374

(Simulated queue)

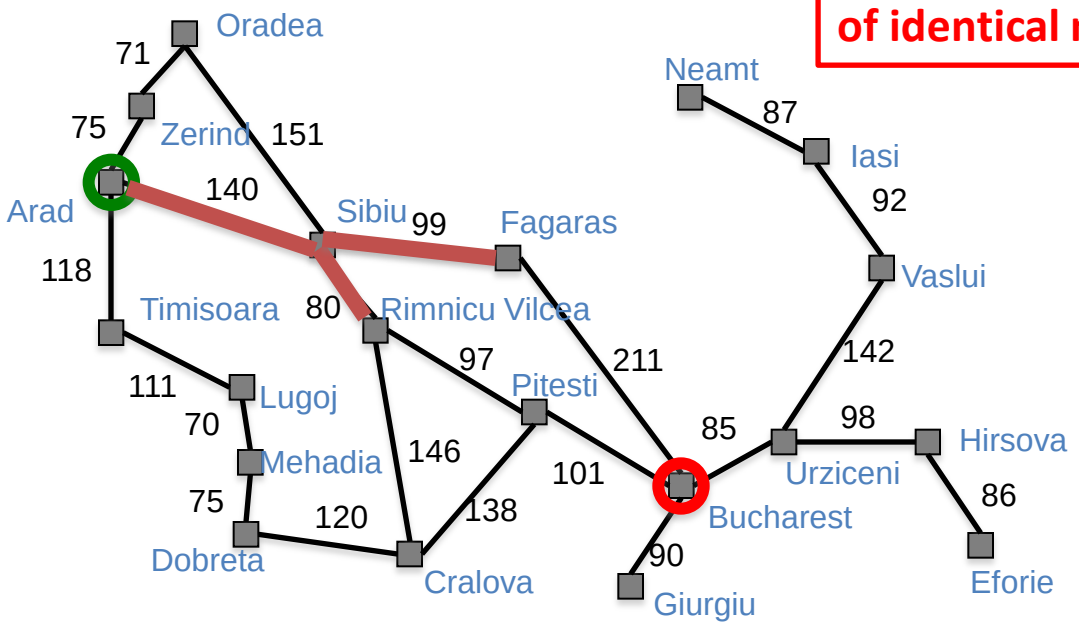
Example: A* Tree Search for Romania

Expanded: Arad/366 (0+366), Sibiu/393 (140+253), RimnicuVilcea/413 (220+193), Fagaras/415 (239+176)

Children: Bucharest/450 (450+0), Sibiu/591 (338+253)

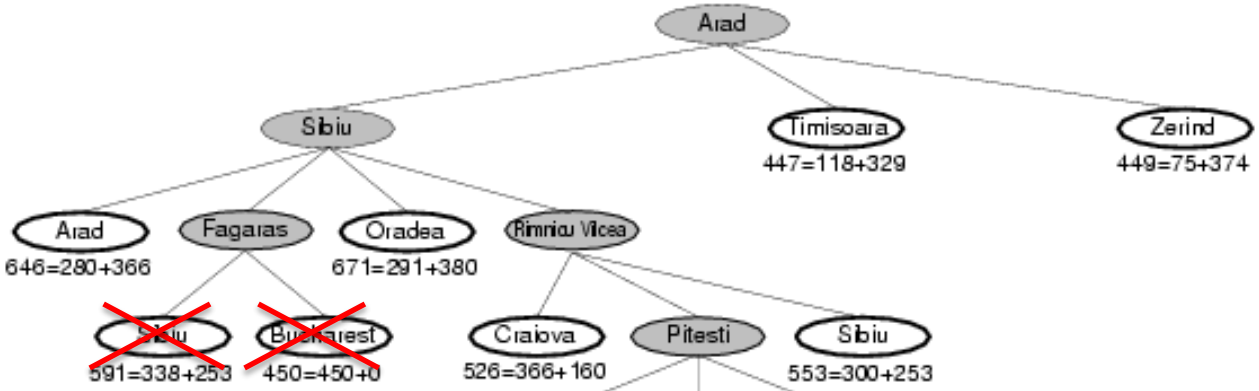
Frontier: ~~Arad/366 (0+366)~~, ~~Sibiu/393 (140+253)~~, Timisoara/447 (118+329), Zerind/449 (75+374), Arad/646 (280+366), ~~Fagaras/415 (239+176)~~, Oradea/671 (291+380), ~~RimnicuVilcea/413 (220+193)~~, Craiova/526 (366+160), Pitesti/417 (317+100), Sibiu/553 (300+253), Bucharest/450 (450+0), ~~Sibiu/591 (338+253)~~

Remove the higher-cost of identical nodes.

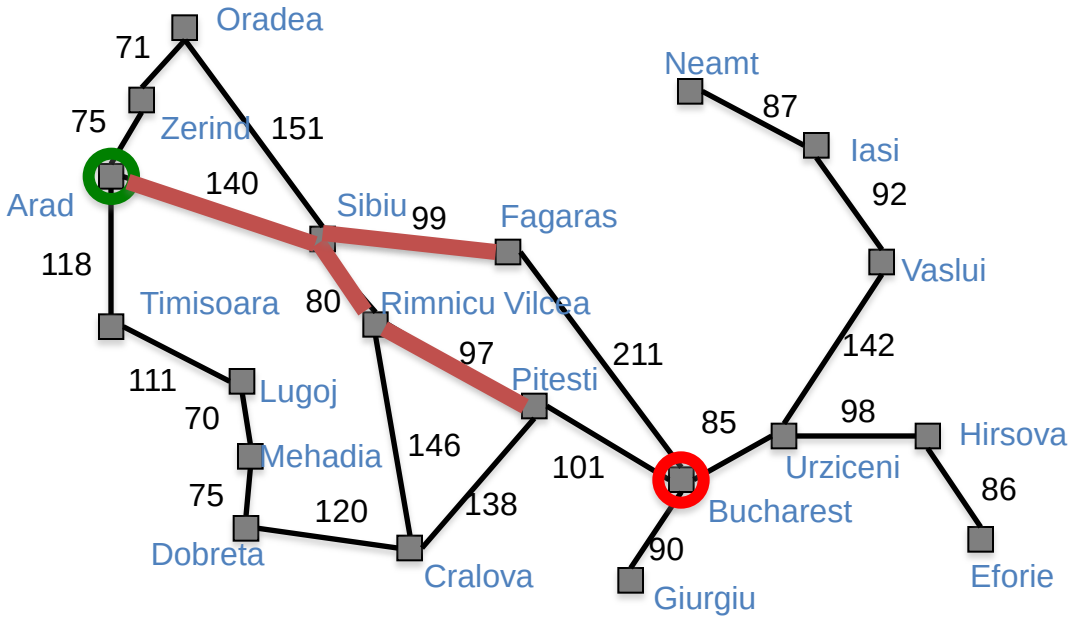


Straight-line dist to goal	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Fagaras	176
Lugoj	244
Mehadia	241
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Zerind	374

Example: A* Tree Search for Romania



Remove the higher-cost of identical nodes.



Straight-line dist to goal	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Fagaras	176
Lugoj	244
Mehadia	241
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Zerind	374

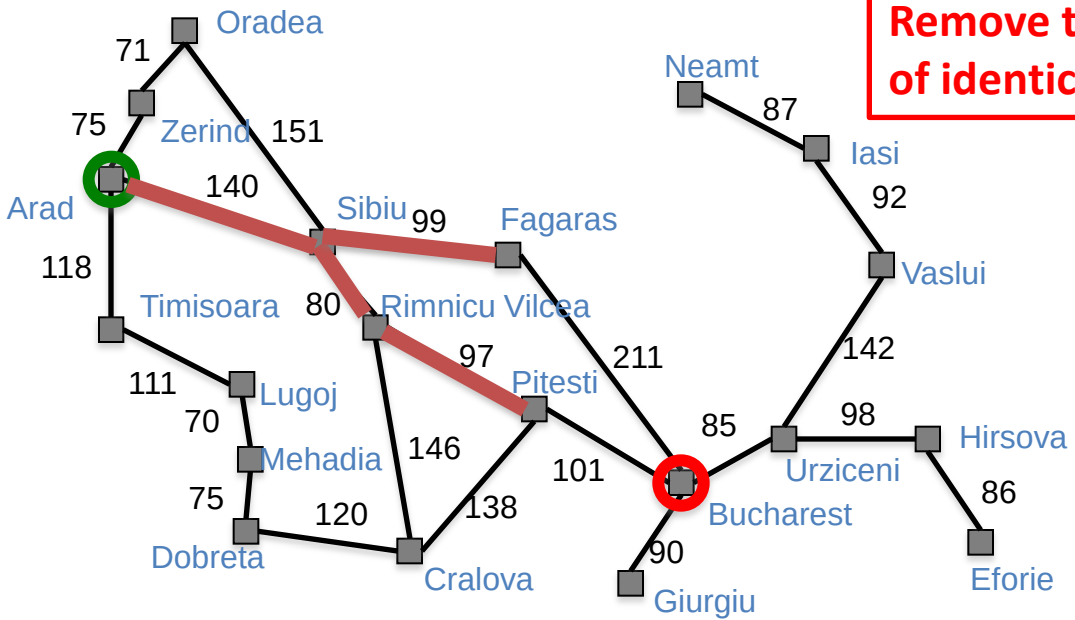
(Simulated queue)

Example: A* Tree Search for Romania

Expanded: Arad/366 (0+366), Sibiu/393 (140+253), RimnicuVilcea/413 (220+193), Fagaras/415 (239+176), Pitesti/417 (317+100),

Children: Bucharest/418 (418+0), Craiova/615 (455+160), RimnicuVilcea/607 (414+193),

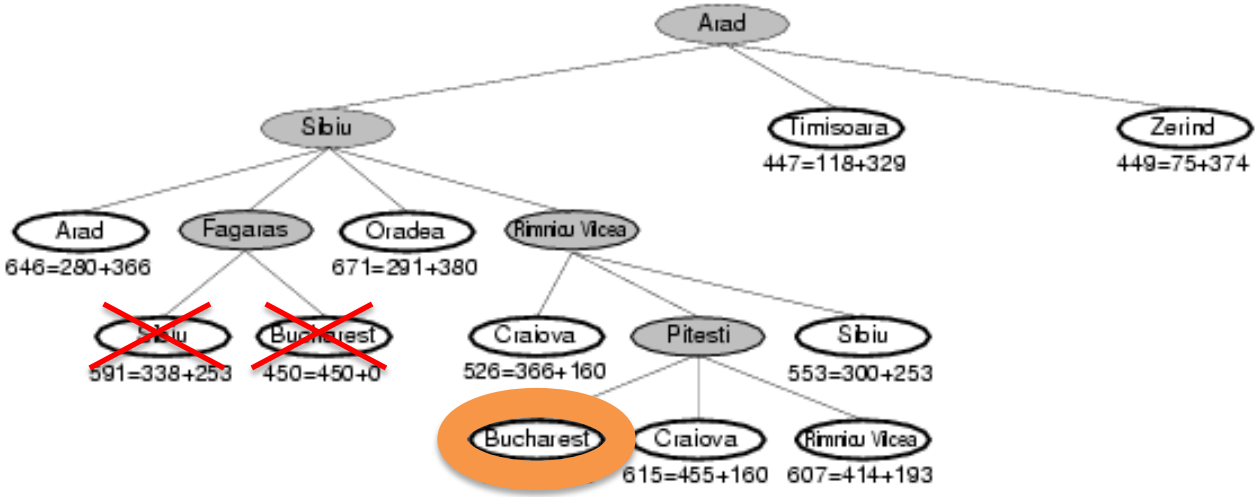
Frontier: ~~Arad/366 (0+366)~~, ~~Sibiu/393 (140+253)~~, Timisoara/447 (118+329), Zerind/449 (75+374), Arad/646 (280+366), ~~Fagaras/415 (239+176)~~, Oradea/671 (291+380), ~~RimnicuVilcea/413 (220+193)~~, Craiova/526 (366+160), ~~Pitesti/417 (317+100)~~, Sibiu/553 (300+253), ~~Bucharest/450 (450+0)~~, ~~Sibiu/591 (338+253)~~, Bucharest/418 (418+0), Craiova/615 (455+160), RimnicuVilcea/607 (414+193)

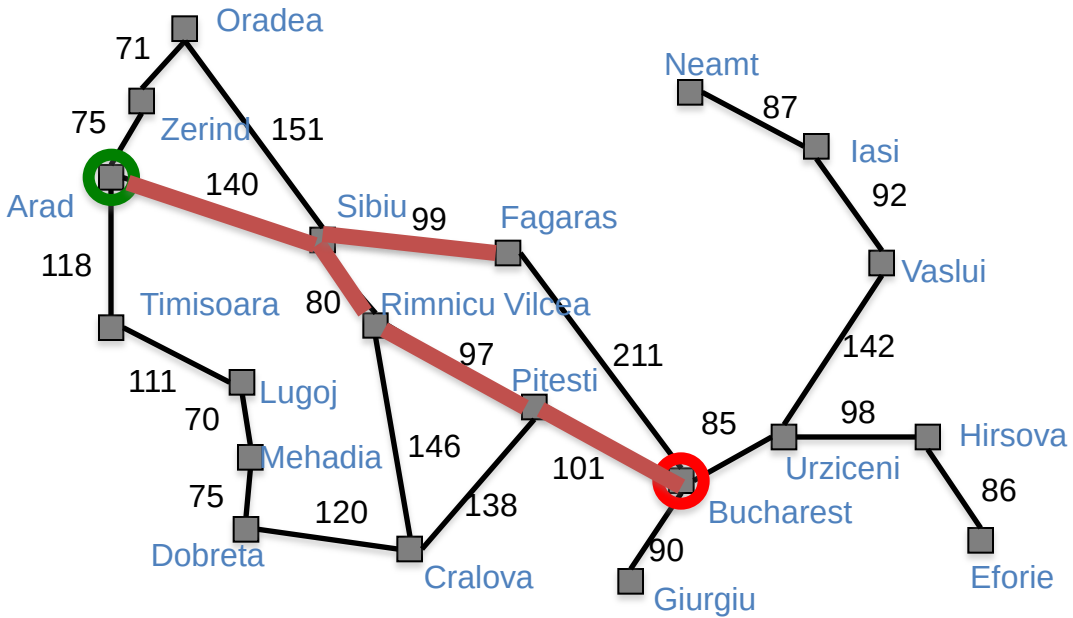


Remove the higher-cost of identical nodes.

Straight-line dist to goal	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Fagaras	176
Lugoj	244
Mehadia	241
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Zerind	374

Example: A* Tree Search for Romania





Straight-line dist to goal	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Fagaras	176
Lugoj	244
Mehadia	241
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Zerind	374

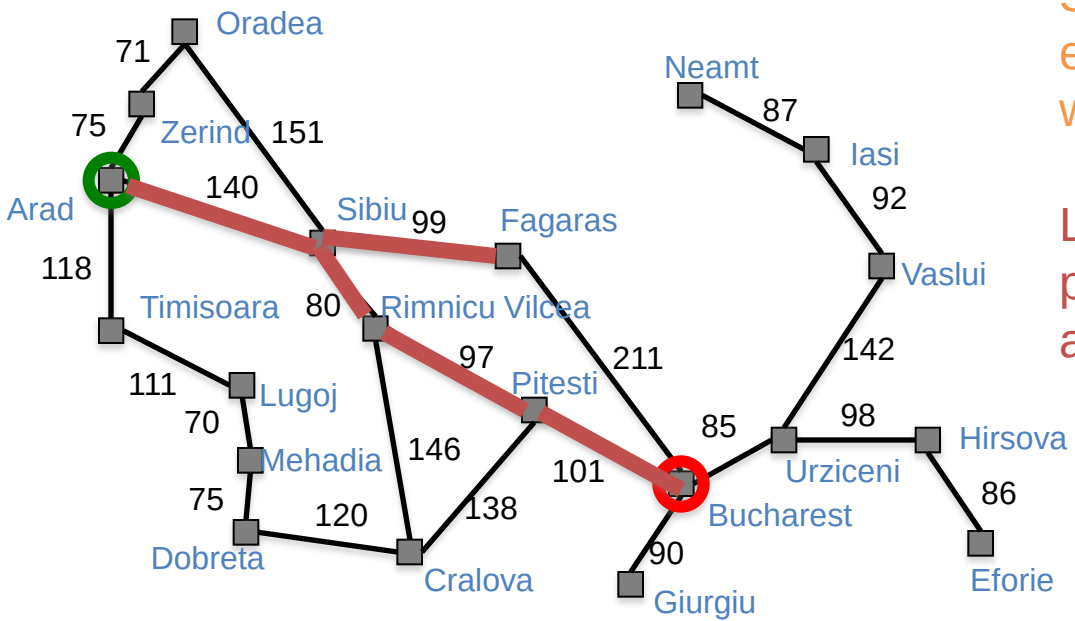
(Simulated queue)

Example: A* Tree Search for Romania

Expanded: Arad/366 (0+366), Sibiu/393 (140+253), RimnicuVilcea/413 (220+193), Fagaras/415 (239+176), Pitesti/417 (317+100), Bucharest/418 (418+0)

Children: None *(goal test succeeds)*

Frontier: ~~Arad/366 (0+366)~~, ~~Sibiu/393 (140+253)~~, Timisoara/447 (118+329), Zerind/449 (75+374), Arad/646 (280+366), ~~Fagaras/415 (239+176)~~, Oradea/671 (291+380), ~~RimnicuVilcea/413 (220+193)~~, Craiova/526 (366+160), ~~Pitesti/417 (317+100)~~, Sibiu/553 (300+253), ~~Bucharest/450 (450+0)~~, ~~Sibiu/591 (338+253)~~, ~~Bucharest/418 (418+0)~~, Craiova/615 (455+160), RimnicuVilcea/607 (414+193)



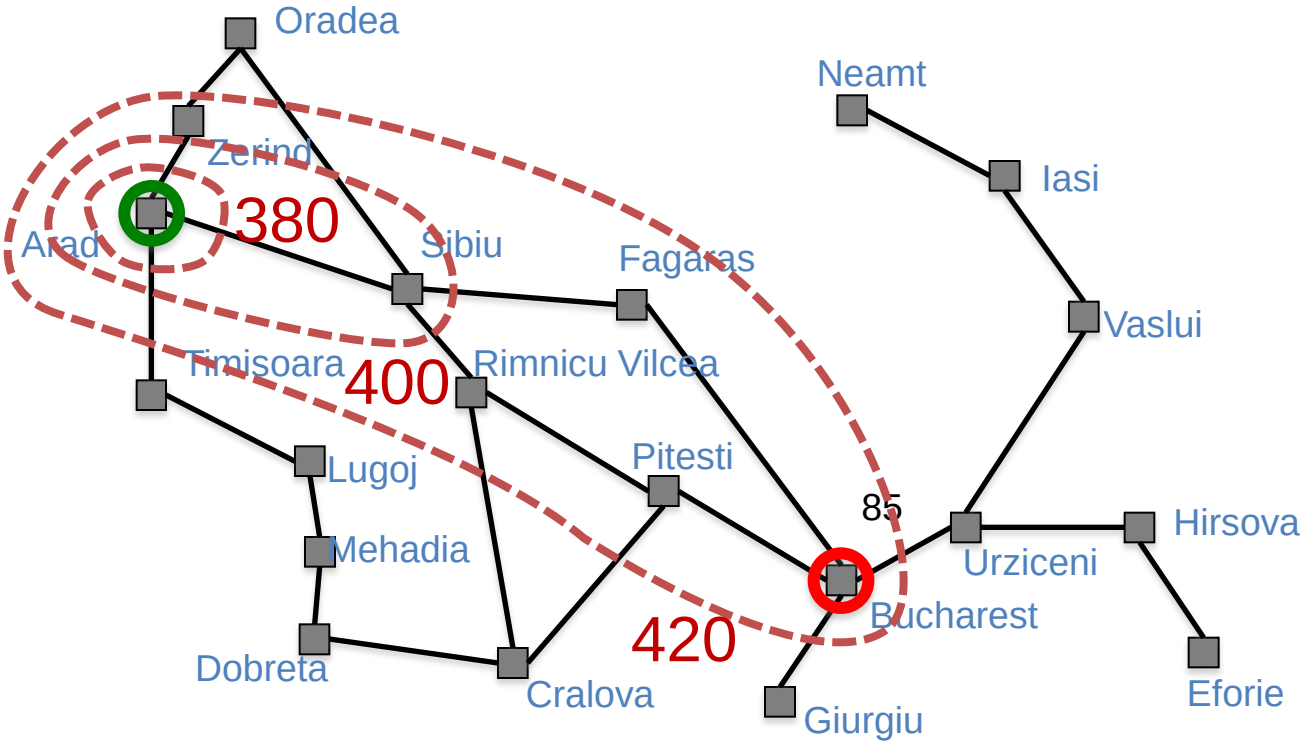
Shorter, more expensive path was removed

Longer, cheaper path will be found and returned

Straight-line dist to goal	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Fagaras	176
Lugoj	244
Mehadia	241
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Zerind	374

Contours of A* search

- For consistent heuristic, A* expands in order of increasing *f* value
- Gradually adds "*f*-contours" of nodes
- Contour *i* has all nodes with $f(n) \leq f_i$, where $f_i < f_{i+1}$
- Contours at *f* = 380, 400, & 420 starting at Arad. Nodes inside a given contour have *f*-costs ≤ the contour value.



Properties of A* search

- Complete? Yes
  - Unless infinitely many nodes with $f < f(G)$
  - Cannot happen if step-cost $\geq \varepsilon > 0$
- Time/Space? Depends on heuristic. Informally, $O(b^m)$.
  - Except if $|h(n) - h^*(n)| \leq O(\log h^*(n))$
    - Unlikely to have such an excellent heuristic function
  - More formally, it depends on the heuristic function (see R&N p. 98)
    - In simplest cases, complexity is $O(b^{h^*-h}) \approx O((b^{(h^*-h)/h^*})^d)$
    - Thus, in simplest cases, $b^{(h^*-h)/h^*)}$ is the effective branching factor (discussed below)
- Optimal? Yes
  - With: Tree-Search, admissible heuristic; Graph-Search, consistent heuristic
- Optimally efficient? Yes
  - No optimal algorithm with same heuristic is guaranteed to expand fewer nodes

Optimality of A* tree search with an admissible heuristic

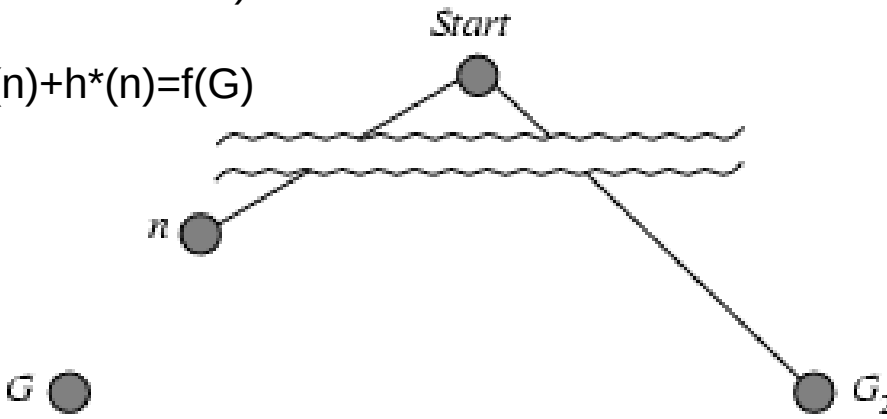
- Proof:
  - Suppose some suboptimal goal G_2 has been generated & is on the frontier. Let n be an unexpanded node on the path to an optimal goal G
  - Show: $f(n) < f(G_2)$ (so, n is expanded before G_2)

$f(G_2) = g(G_2)$ since $h(G_2) = 0$
$f(G) = g(G)$ since $h(G) = 0$
$g(G_2) > g(G)$ since G_2 is suboptimal

$f(G_2) > f(G)$ from above, with h=0

$h(n) \leq h^*(n)$ since h is admissible (*under*-estimate)
$g(n) + h(n) \leq g(n) + h^*(n)$ from above
$f(n) \leq f(G)$ since $g(n)+h(n)=f(n)$ & $g(n)+h^*(n)=f(G)$
$f(n) < f(G2)$ from above

R&N pp. 95-98 proves the optimality of A* graph search with a consistent heuristic

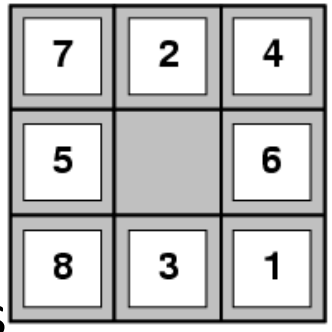


Memory-bounded heuristic search

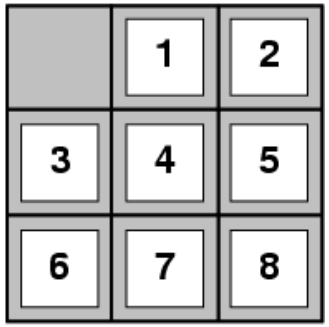
- Memory is a major limitation of A*
  - Usually run out of memory before run out of time
- How can we solve the memory problem?

- R&N section 3.5.3, pp. 99-102, gives methods for A* that use less memory (but more time & book-keeping)
  - Iterative-deepening A* (IDA*), recursive best-first search (RBFS), memory-bounded A* (MA*), simplified MA* (SMA*)
- I no longer teach these methods because I believe they add little that is conceptually new
  - Be aware that they exist if you need them

Heuristic functions



Start State



Goal State

- 8-Puzzle
  - Avg solution cost is about 22 steps
  - Branching factor ~ 3
  - Exhaustive search to depth 22 = 3.1 x 10^10 states
  - A good heuristic f'n can reduce the search process
  - True cost for this start & goal: 26

- Two commonly used heuristics
  - h_1: the number of misplaced tiles

    $h_1(s) = 8$
  - h_2: sum of the distances of the tiles from their goal ("Manhattan distance")

    $h_2(s) = 3+1+2+2+2+3+3+2$

    $= 18$

Dominance

- Definition:

  If $h_2(n) \geq h_1(n)$ for all n

  then h_2 dominates h_1
  - h_2 is almost always better for search than h_1
  - h_2 is guaranteed to expand no more nodes than h_1
  - h_2 almost always expands fewer nodes than h_1
  - Not useful unless are h_1 , h_2 are admissible / consistent

- Example: 8-Puzzle / sliding tiles
  - h_1: the number of misplaced tiles
  - h_2: sum of the distances of the tiles from their goal
  - h_2 dominates h_1

Example: 8-Puzzle

Average number of nodes expanded

d	IDS	A*(h1)	A*(h2)
2	10	6	6
4	112	13	12
8	6384	39	25
12	364404	227	73
14	3473941	539	113
20	----------	7276	676
24	----------	39135	1641

Average over 100 randomly generated 8-puzzle problems
h1 = number of tiles in the wrong position
h2 = sum of Manhattan distances

Effective branching factor, b*

- Let A* generate N nodes to find a goal at depth d
  - Effective branching b* is the branching factor a uniform tree of depth d would have in order to contain N+1 nodes:

$$N + 1 = 1 + b^* + (b^*)^2 + (b^*)^3 + \ldots + (b^*)^d$$
$$= ((b^*)^{d+1} - 1) / (b^* - 1)$$
$$N \approx (b^*)^d \quad \Rightarrow \quad b^* \approx \sqrt[d]{N}$$

  - For sufficiently hard problems, b* is often fairly constant across different problem instances
- A good guide to the heuristic's overall usefulness
- A good way to compare different heuristics

(Optional Mathematical Details)

- Let $\Phi = 1 + b^* + (b^*)^2 + (b^*)^3 + \ldots$

- Then $(b^*)\Phi = b^* + (b^*)^2 + (b^*)^3 + \ldots$
  - So $\Phi - (b^*)\Phi = \Phi(1 - (b^*)) = 1$
  - So $\Phi = 1 / (1 - (b^*))$
- Then $(b^*)^{d+1}\Phi = (b^*)^{d+1} + (b^*)^{d+2} + (b^*)^{d+3} + \ldots$
  - So $1 + b^* + (b^*)^2 + (b^*)^3 + \ldots + (b^*)^d$
    $= \Phi - (b^*)^{d+1}\Phi = \Phi(1 - (b^*)^{d+1})$
  - So $1 + b^* + (b^*)^2 + (b^*)^3 + \ldots + (b^*)^d$
    $= (1 - (b^*)^{d+1}) / (1 - (b^*)) = ((b^*)^{d+1} - 1) / (b^* - 1)$
    $\approx (b^*)^d$
- Note: Usually b* ≥ 1 and so the infinite sum Φ does not converge. Thus, the analysis above is more an explanation of a useful result than it is a formal proof. R&N discuss effective branching factor but give no formula (p. 103, but see p. 111).

Designing heuristics

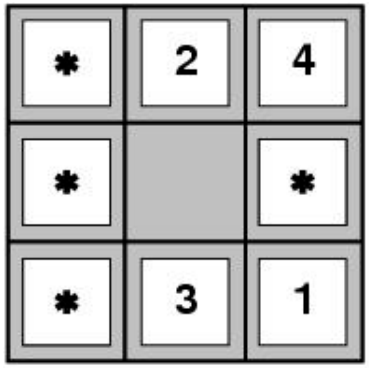
- Often constructed via problem relaxations
  - A problem with fewer restrictions on actions
  - Cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- Example: 8-Puzzle
  - Relax rules so a tile can move anywhere: $h_1(n)$
  - Relax rules so tile can move to any adjacent square: $h_2(n)$
- A useful way to generate heuristics
  - Example: ABSOLVER (Prieditis 1993) discovered the first useful heuristic for the Rubik's cube

More on heuristics

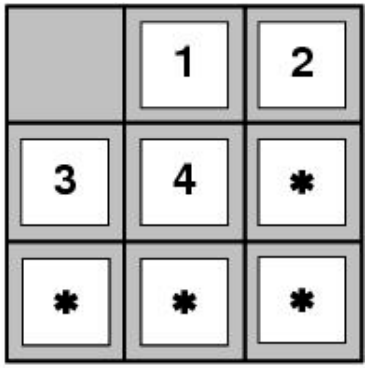
- Combining heuristics
  - H(n) = max { h1(n), h2(n), … , hk(n) }
  - "max" chooses the least optimistic heuristic at each node

- Pattern databases
  - Solve a subproblem of the true problem

    (= a lower bound on the cost of the true problem)
  - Store the exact solution for each possible subproblem



Start State



Goal State

Summary

- Uninformed search has uses but also severe limitations
- Heuristics are a structured way to make search smarter

- Informed (or heuristic) search uses problem-specific heuristics to improve efficiency
  - Best-first, A* (and if needed for memory, RBFS, SMA*)
  - Techniques for generating heuristics
  - A* is optimal with admissible (tree) / consistent (graph heuristics

- Can provide significant speed-ups in practice
  - Example: 8-Puzzle, dramatic speed-up
  - Still worst-case exponential time complexity (NP-complete)

You should know…

- evaluation function *f(n)* and heuristic function *h(n)* for each node *n*
  - g(n) = known path cost so far to node n.
  - h(n) = estimate of (optimal) cost to goal from node n.
  - f(n) = g(n)+h(n) = estimate of total cost to goal through node n.

- Heuristic searches: Greedy-best-first, A*
  - A* is optimal with admissible (tree)/consistent (graph) heuristics
  - Prove that A* is optimal with admissible heuristic for tree search
  - Recognize when a heuristic is admissible or consistent

- h_2 dominates h_1 iff $h_2(n) \geq h_1(n)$ for all n
- Effective branching factor: b*
- Inventing heuristics: relaxed problems; max or convex combination