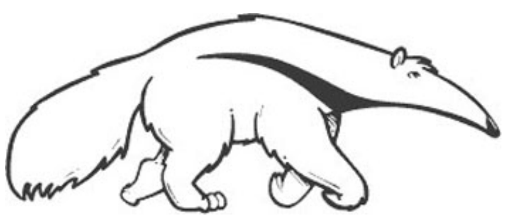
Heuristic search, A*

CS171, Fall 2016

Introduction to Artificial Intelligence

Prof. Alexander Ihler



Reading: R&N 3.5-3.7

BREN:ICS
INFORMATION AND COMPUTER SCIENCES

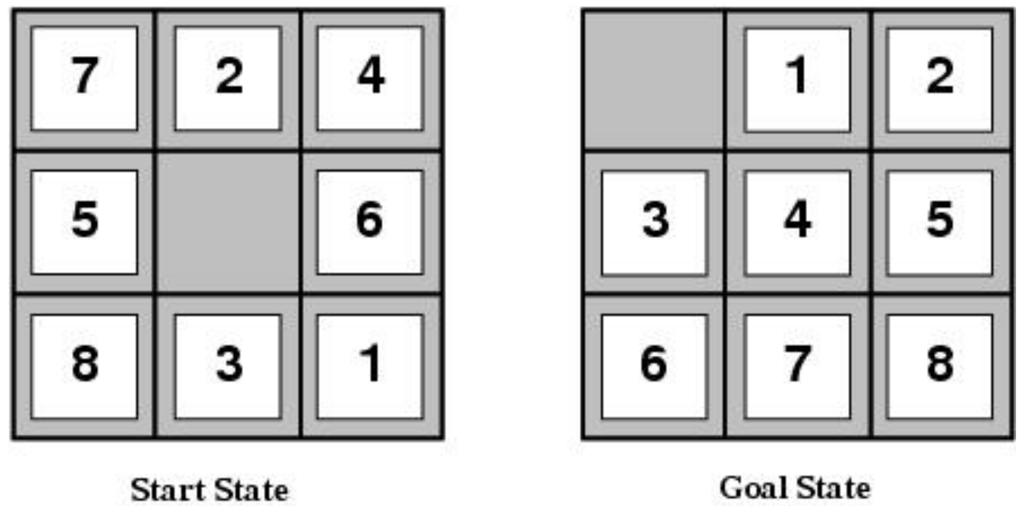
UNIVERSITY of CALIFORNIA IRVINE

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

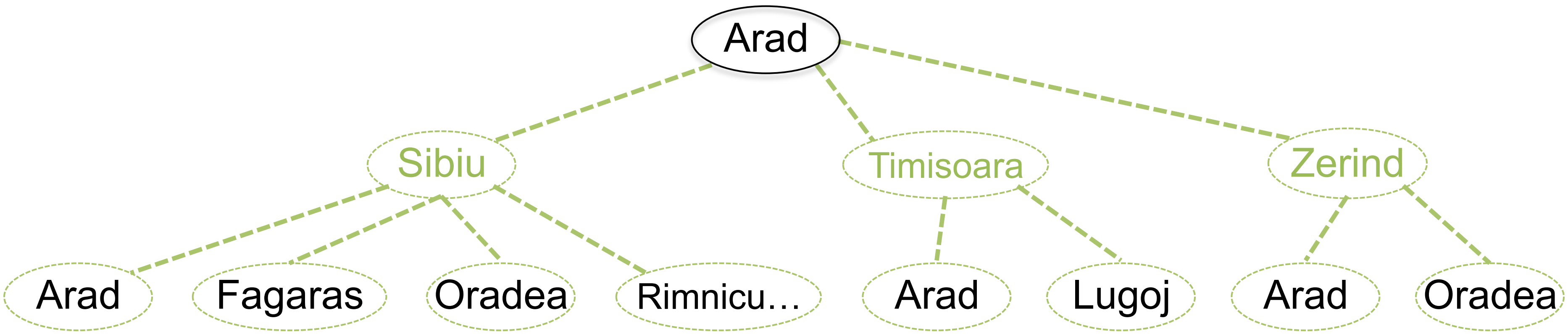
In AI, "NP-Complete" means "Formally interesting"

Limitations of uninformed search



- Search space size makes search tedious
  - Combinatorial explosion
- Ex: 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
    initialize the search tree using the initial state of *pr*
    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 on heuristic quality!
  - The better your heuristic, the faster your search!

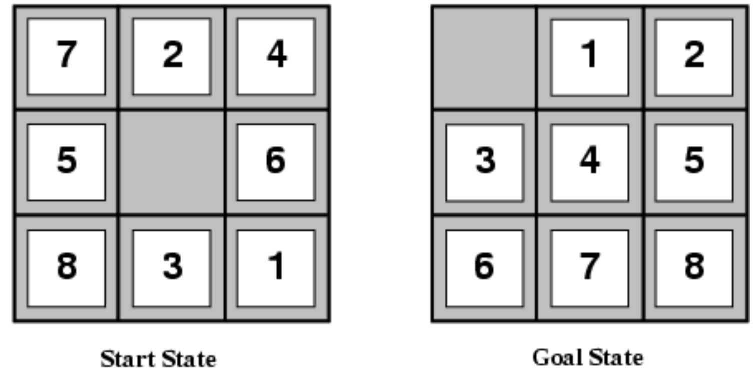
Heuristic function

- Heuristic
  - Def'n: a commonsense rule or rules intended to increase the probability of solving some problem
  - Same linguistic root as "Eureka" = "I have found it"
  - Using rules of thumb to find answers

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

- Provides problem-specific knowledge to the search algorithm

Ex: 8-Puzzle



- 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

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

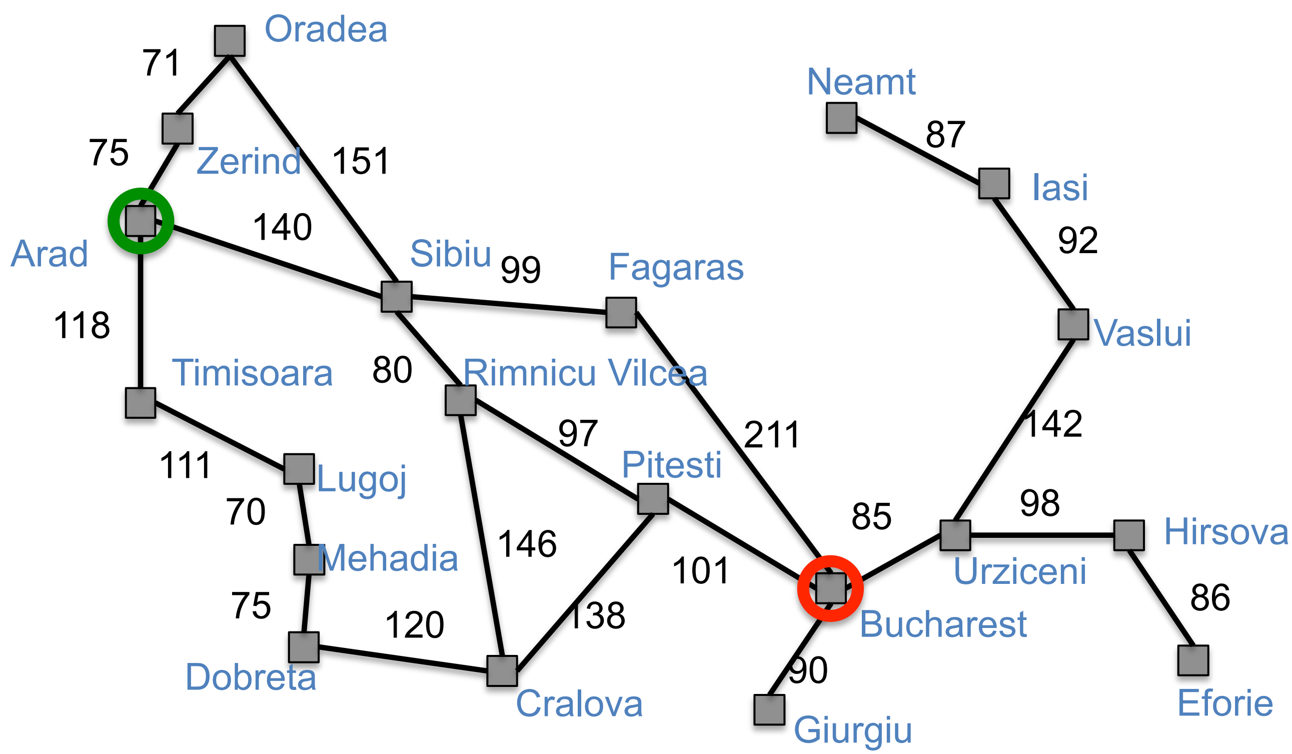
    $h_1(s) = 8$

  - h_2: sum of the distances of the tiles from their goal

    $h_2(s) = 3+1+2+2+2+3+3+2$ ("Manhattan distance")

    $= 18$

Ex: 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

Relationship of search algorithms

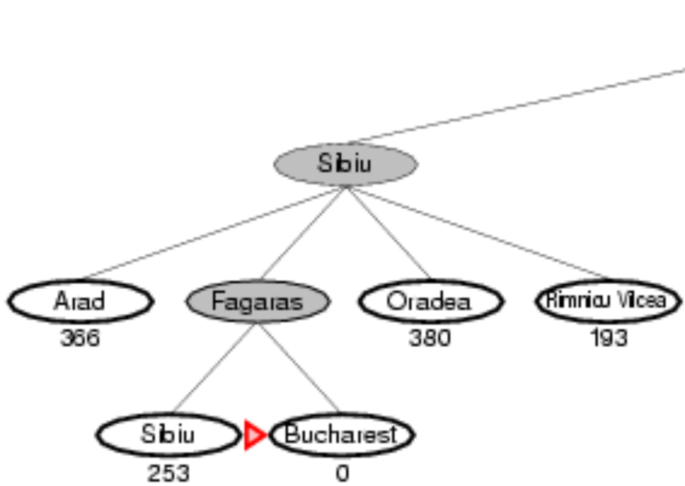
- Notation
  - g(n) = known cost so far to reach n
  - h(n) = estimated (optimal) cost from n to goal
  - f(n) = g(n)+h(n) = estimated (optimal) total cost through n
- Uniform cost search: sort frontier by g(n)
- Greedy best-first search: sort frontier by h(n)
- A* search: sort frontier by f(n)
  - Optimal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

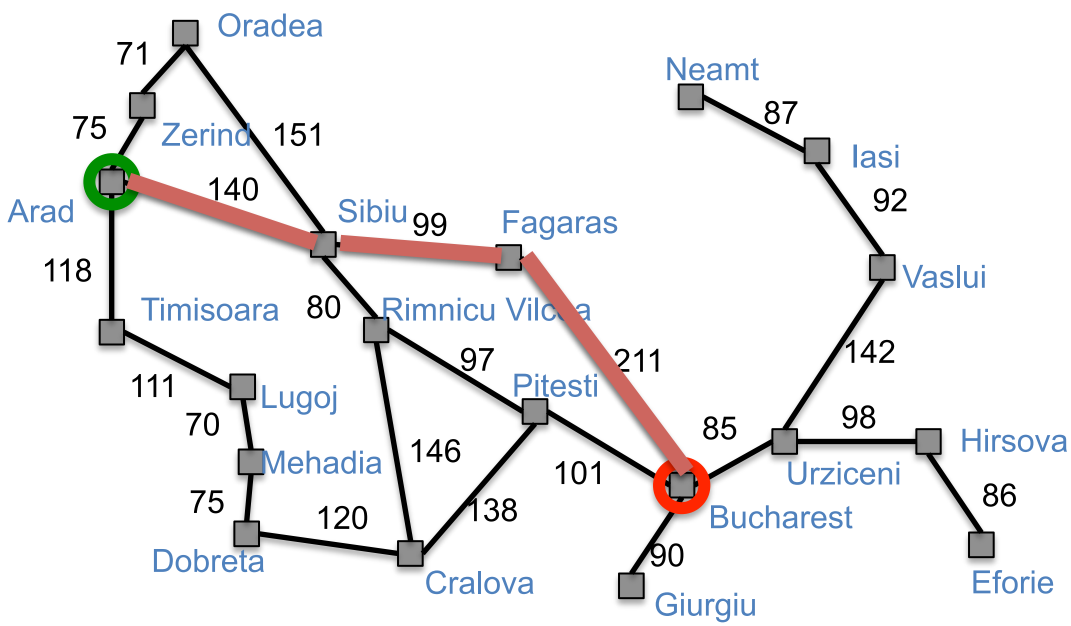
Greedy best-first search

(sometimes just called "best-first")

- h(n) = estimate of cost from n to goal
  - Ex: 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)

Ex: GBFS for Romania



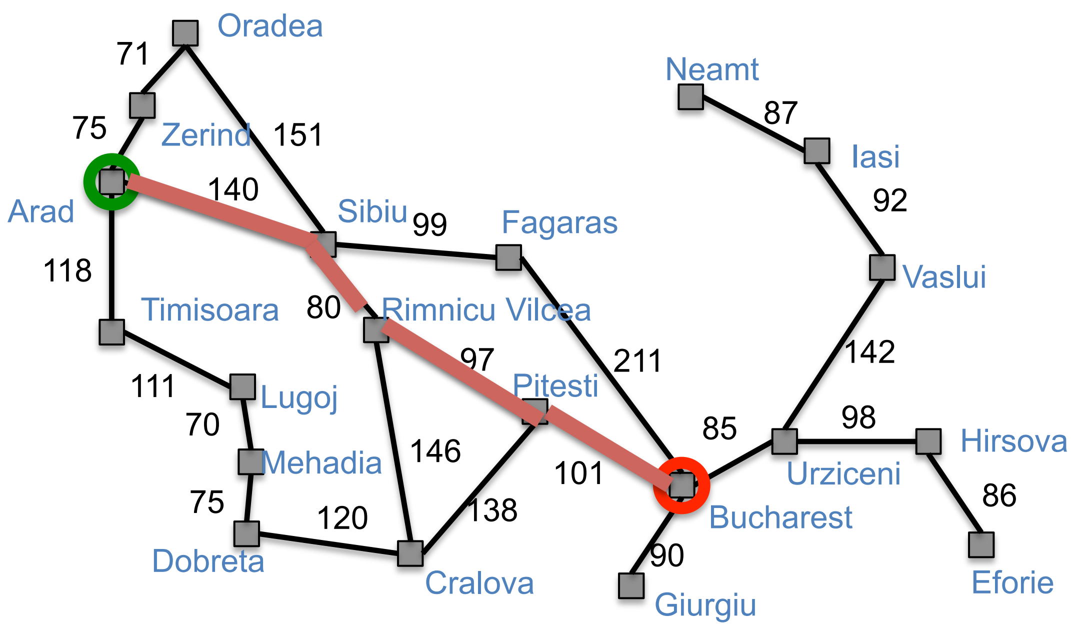


Straight-line dist to goal	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Ex: GBFS for Romania

GBFS: 450km

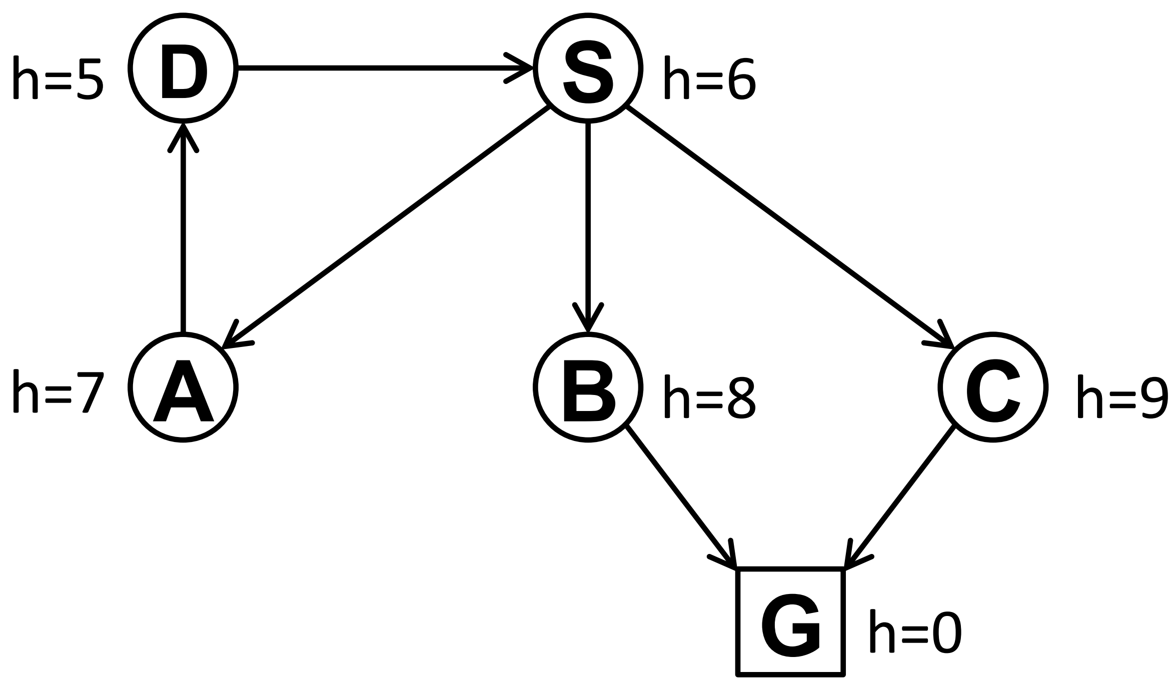
Optimal path: 418 km



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
  - Graph version is complete in finite spaces
- Time? $O(b^m)$
  - A good heuristic can give dramatic improvement
- Space? $O(b^m)$
  - Keeps all nodes in memory
- Optimal? No
  - Ex: Arad – Sibiu – Rimnicu Vilcea – Pitesti – Bucharest 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)

- 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

- Priority queue sort function = $f(n)$

Admissible heuristics

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

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

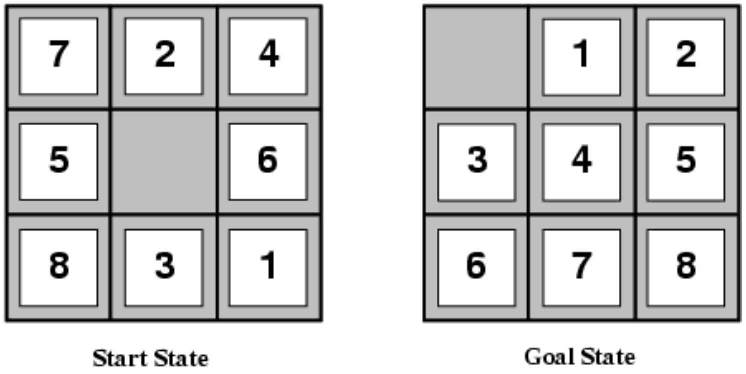
  $h^*(n)$ = the true cost to reach the goal state from n

- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic (or, never pessimistic)
  - Ex: straight-line distance never overestimates road distance

- Theorem:

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

Admissible heuristics



- 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

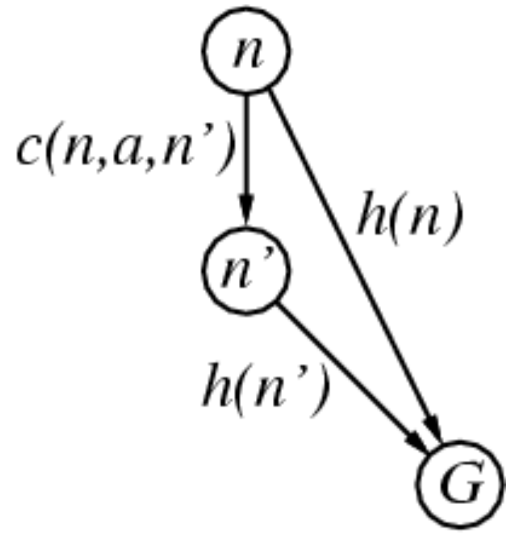
    $h_2(s) = 3+1+2+2+2+3+3+2$ ("Manhattan distance")

    $= 18$

Consistent heuristics

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

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



(Triangle inequality)

- If h is consistent, we have

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

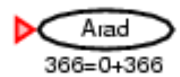
i.e., $f(n)$ is non-decreasing along any path.

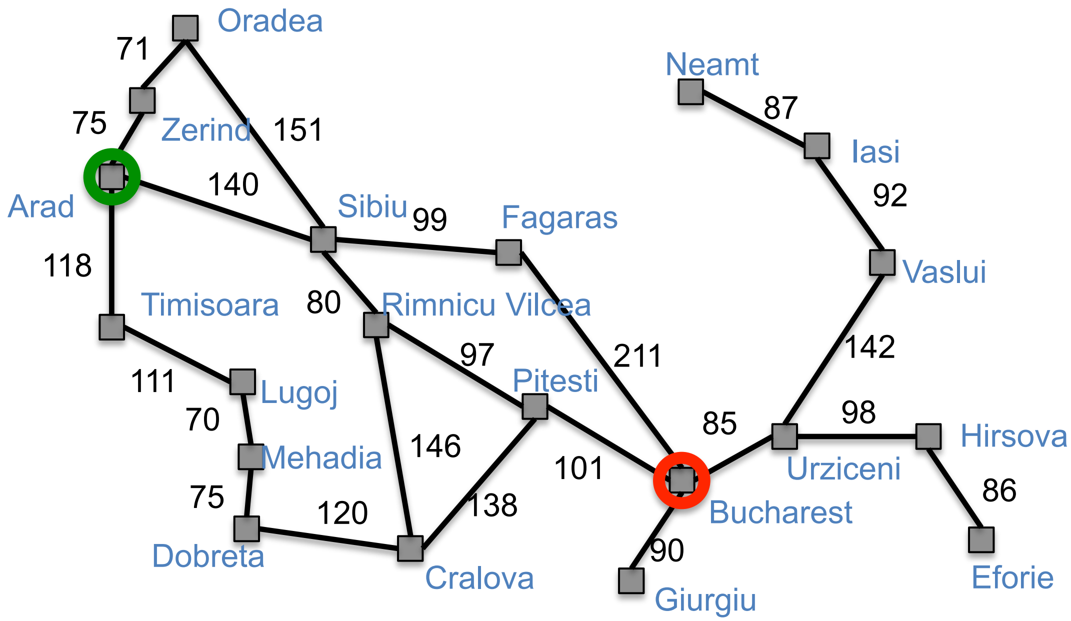
- Consistent $\Rightarrow$ admissible (stronger condition)
- Theorem: If $h(n)$ is consistent, A* using Graph-Search is optimal

Optimality conditions

- Tree search optimal if admissible
- Graph search optimal if 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 are also consistent
- Could we do optimal graph search with an admissible heuristic?
  - Yes, but you would have to do additional work to update descendants when a cheaper path to a node is found
  - A consistent heuristic avoids this problem

Ex: A* 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

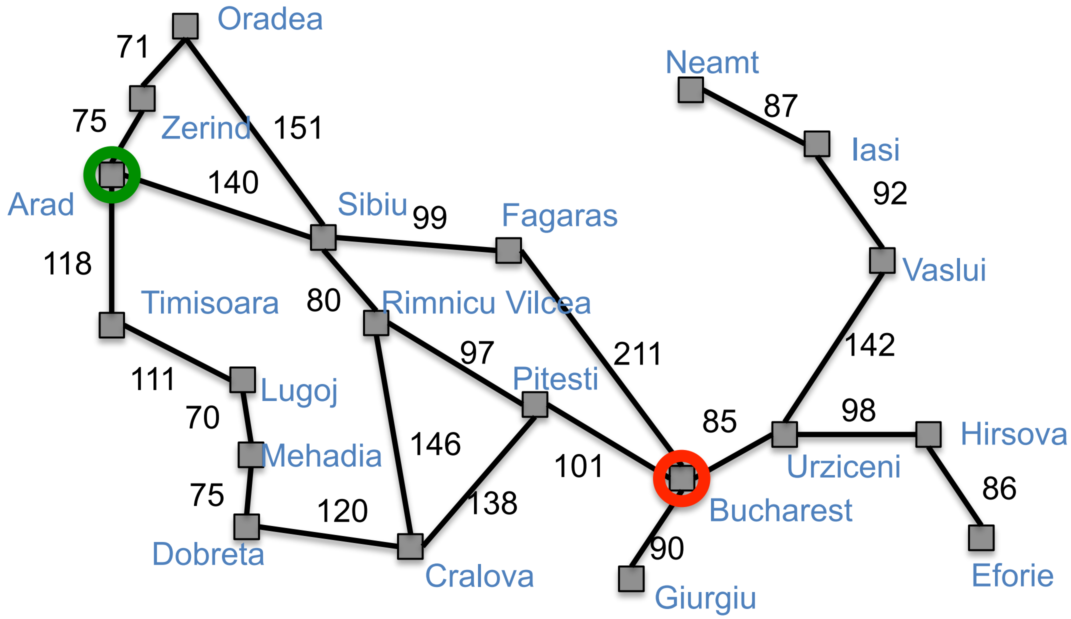
Ex: A* for Romania

(Simulated queue)

Expanded: None

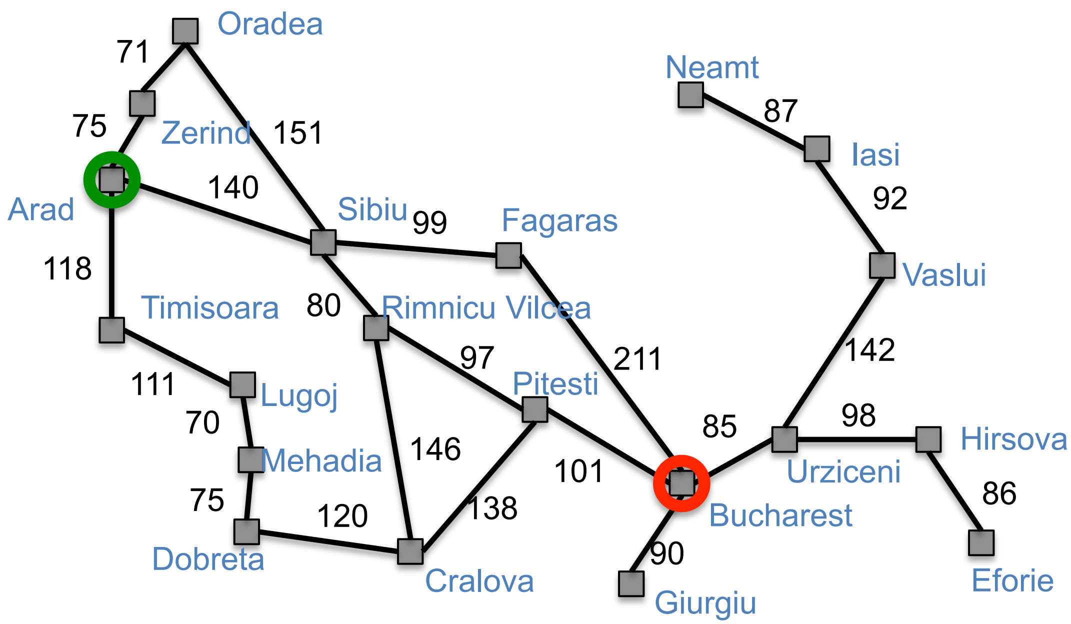
Children: None

Frontier: Arad/366 (0+366),



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

Ex: A* 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

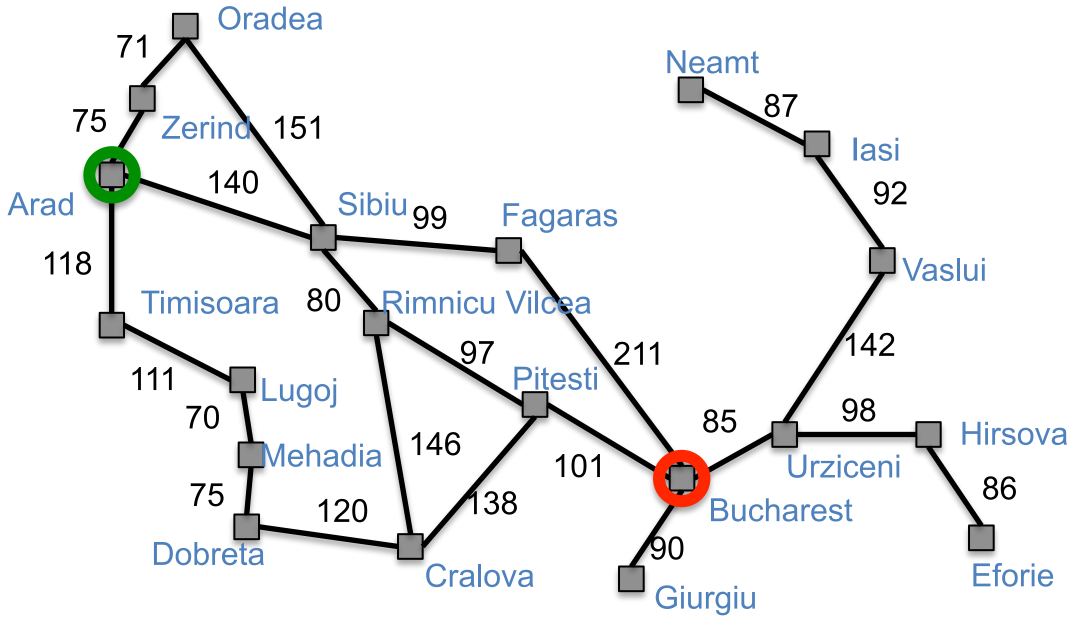
(Simulated queue)

Ex: A* for Romania

Expanded: Arad/366 (0+366),

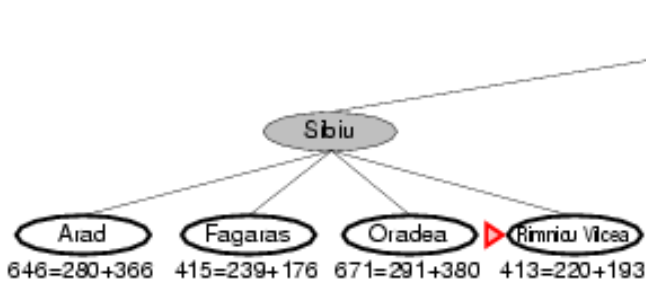
Children: Sibiu/393 (140+253), Timisoara/447 (118+329), Zerind/449 (75+374),

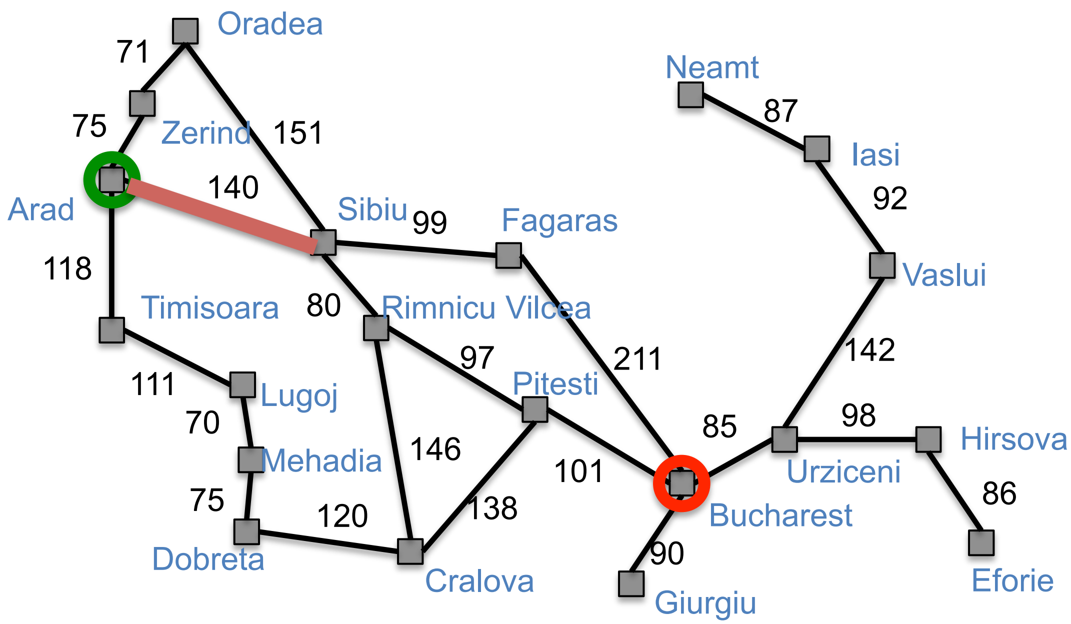
Frontier: Arad/366 (0+366), Sibiu/393 (140+253), Timisoara/447 (118+329), Zerind/449 (75+374),



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

Ex: A* 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

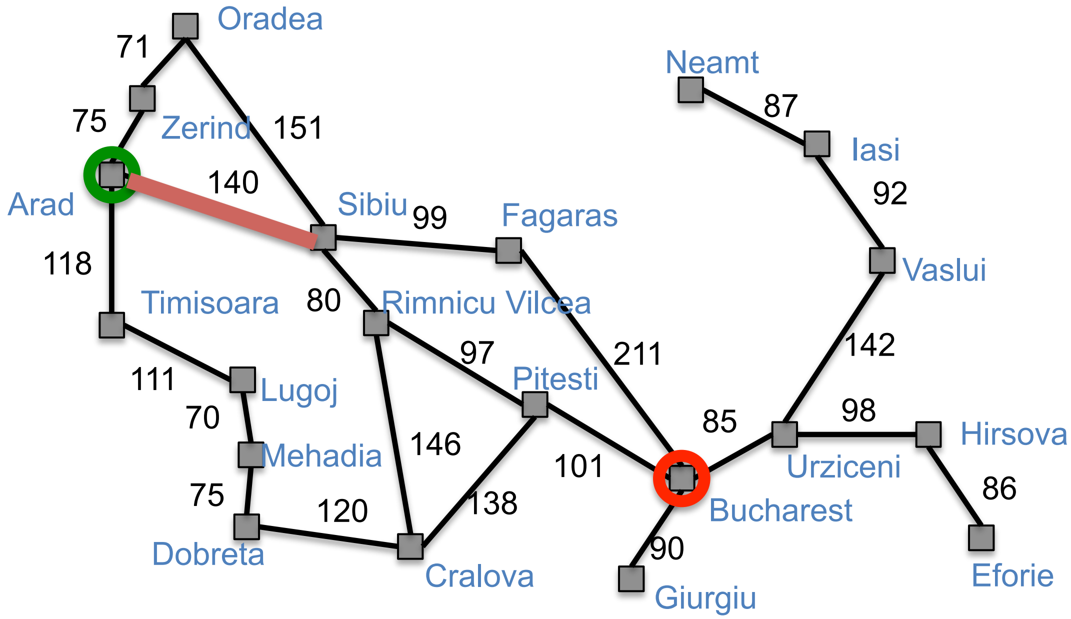
(Simulated queue)

Ex: A* for Romania

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

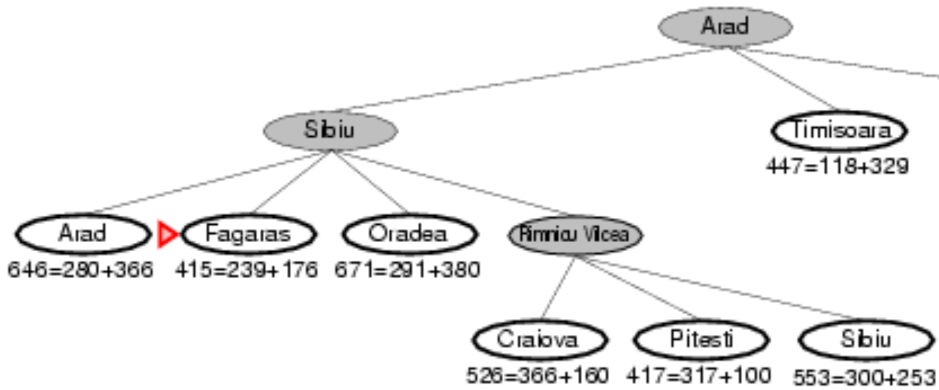
Children: Arad/646 (280+366), Fagaras/415 (239+176), Oradea/671 (291+380), RimnicuVilcea/413 (220+193),

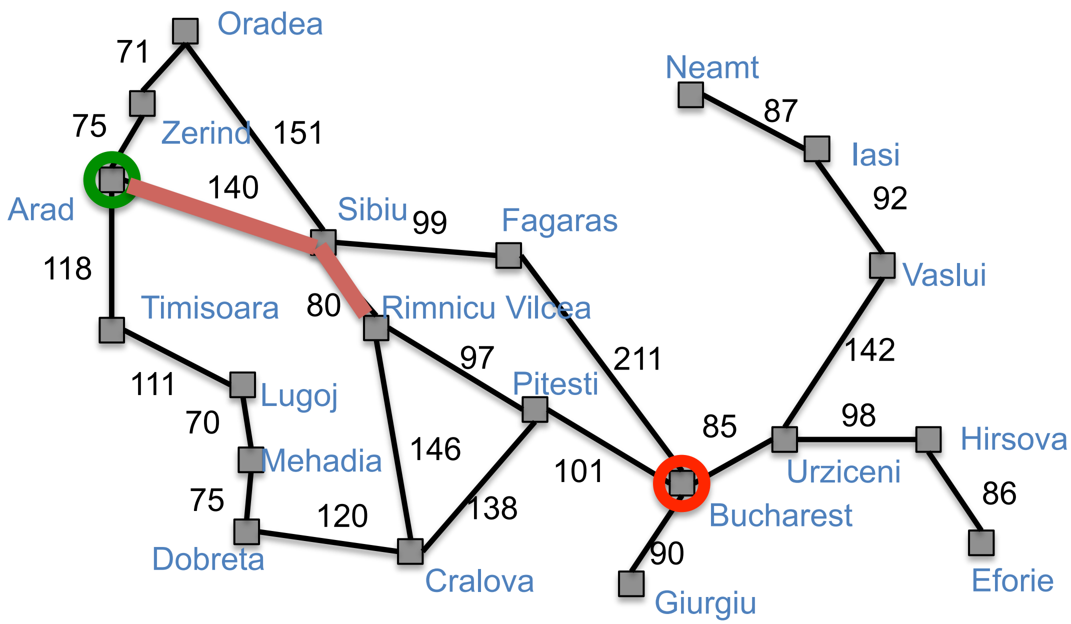
Frontier: Arad/366 (0+366), Sibiu/393 (140+253), Timisoara/447 (118+329), Zerind/449 (75+374), Arad/646 (280+366), Fagaras/415 (239+176), Oradea/671 (291+380), RimnicuVilcea/413 (220+193),



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

Ex: A* 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

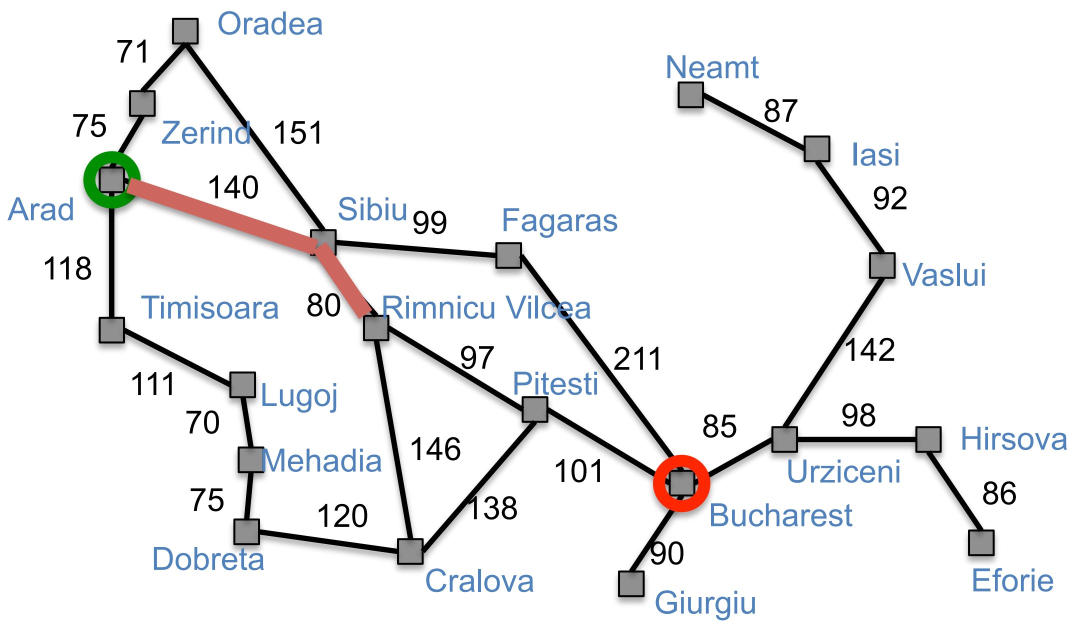
Ex: A* for Romania

(Simulated queue)

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

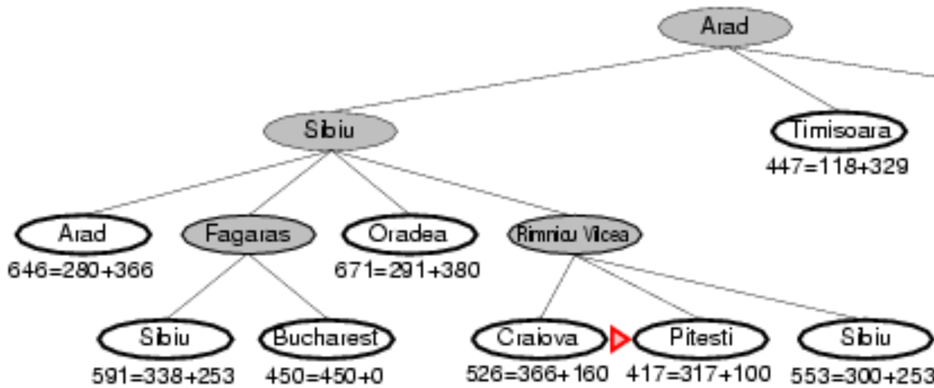
Children: Craiova/526 (366+160), Pitesti/417 (317+100), Sibiu/553 (300+253),

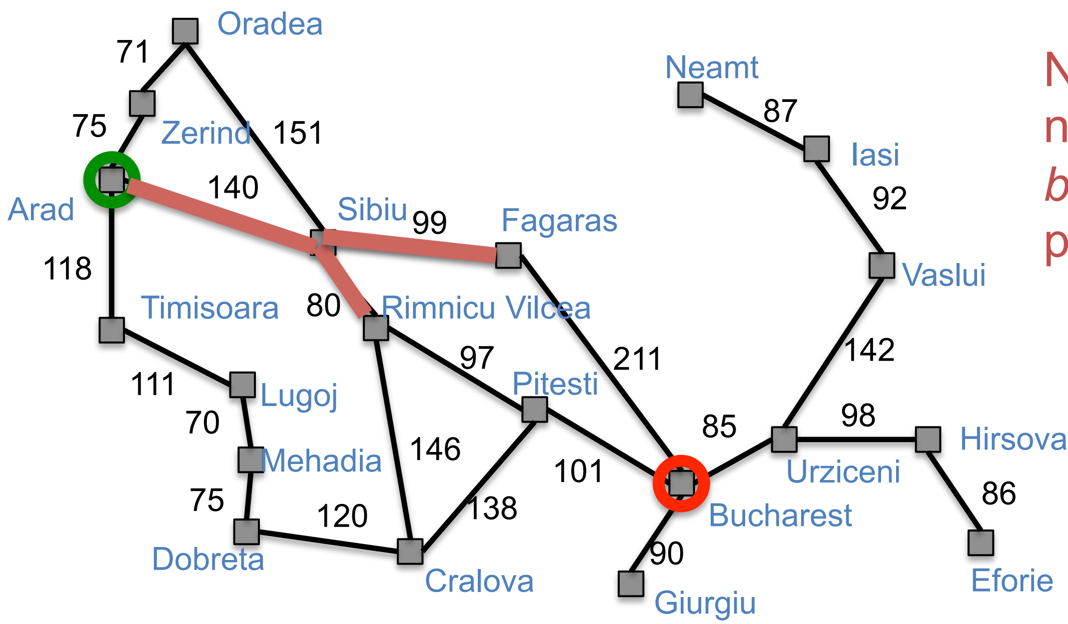
Frontier: Arad/366 (0+366), Sibiu/393 (140+253), Timisoara/447 (118+329), Zerind/449 (75+374), Arad/646 (280+366), Fagaras/415 (239+176), Oradea/671 (291+380), RimnicuVilcea/413 (220+193), Craiova/526 (366+160), Pitesti/417 (317+100), Sibiu/553 (300+253),



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

Ex: A* for Romania





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

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

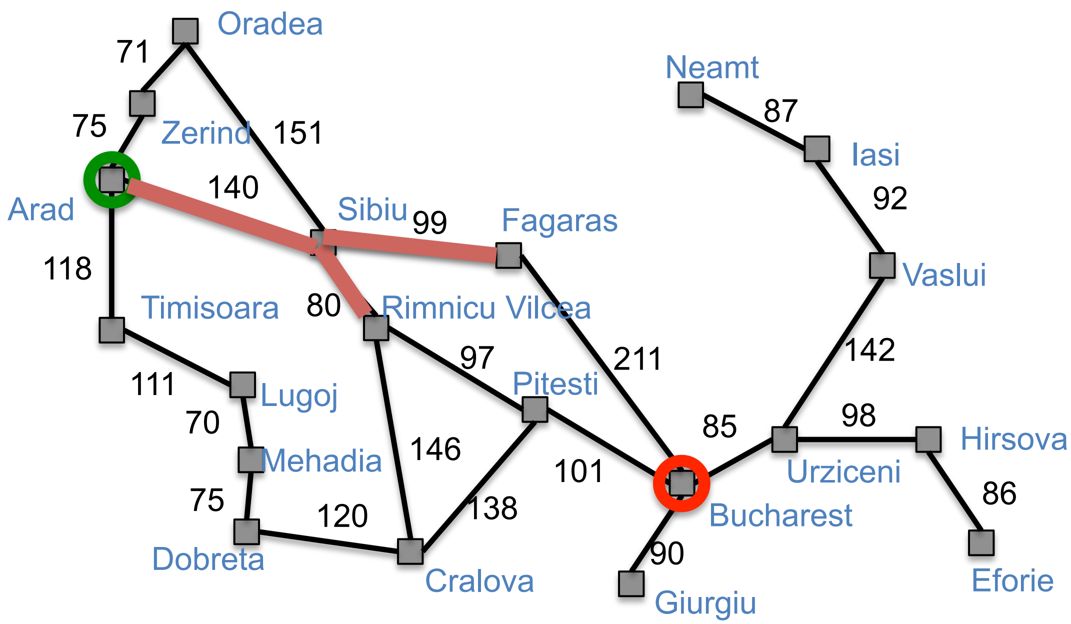
Ex: A* for Romania

(Simulated queue)

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

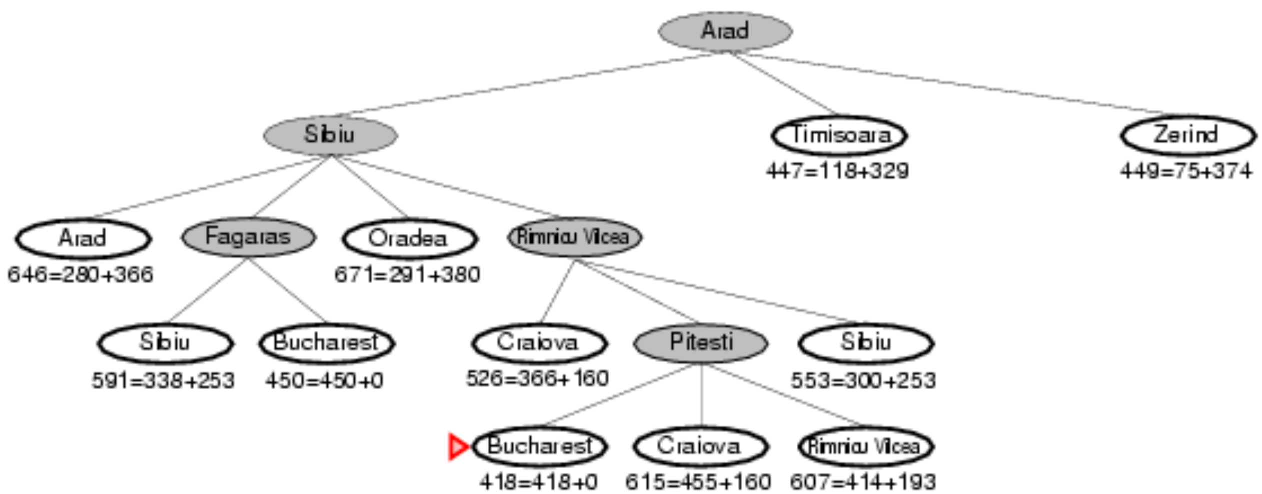
Children: Bucharest/450 (450+0), Sibiu/591 (338+253),

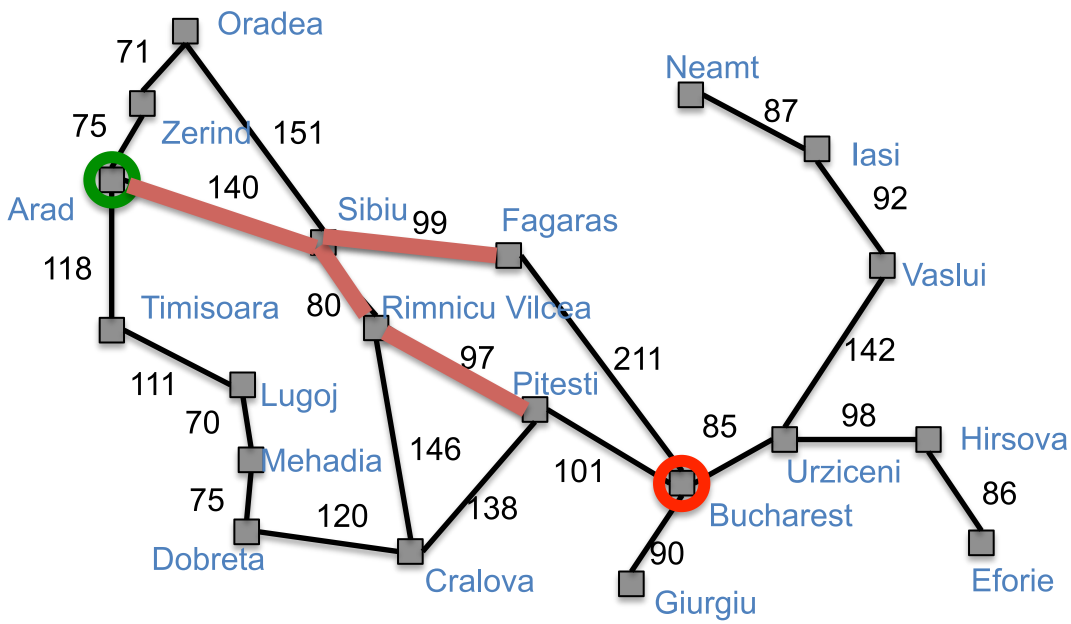
Frontier: Arad/366 (0+366), Sibiu/393 (140+253), Timisoara/447 (118+329), Zerind/449 (75+374), Arad/646 (280+366), Fagaras/415 (239+176), Oradea/671 (291+380), RimnicuVilcea/413 (220+193), Craiova/526 (366+160), Pitesti/417 (317+100), Sibiu/553 (300+253), Bucharest/450 (450+0), Sibiu/591 (338+253),



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

Ex: A* 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

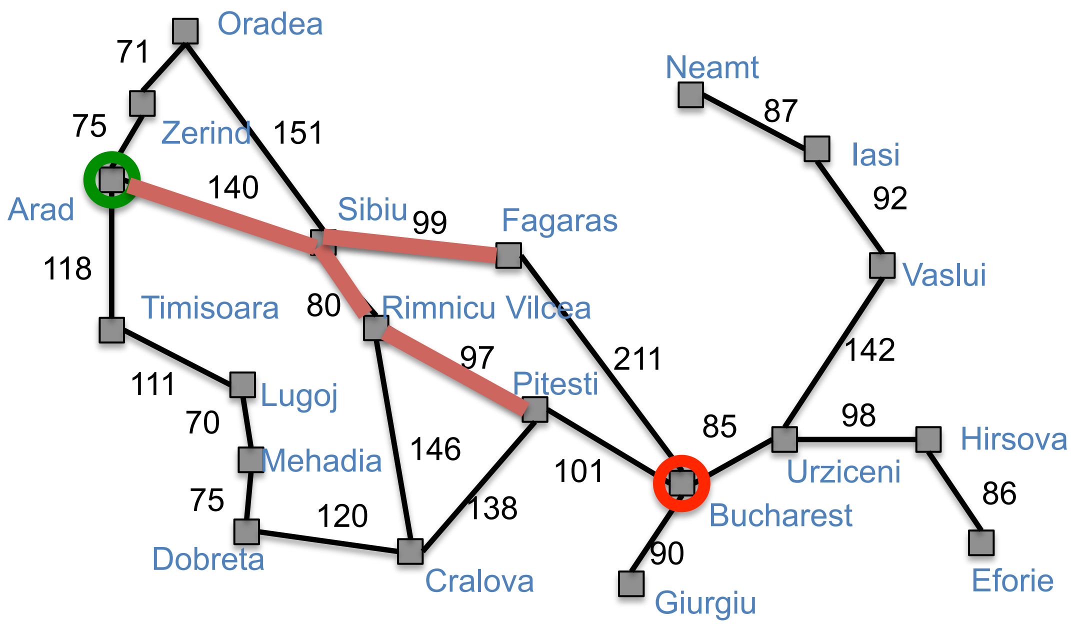
(Simulated queue)

Ex: A* for Romania

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

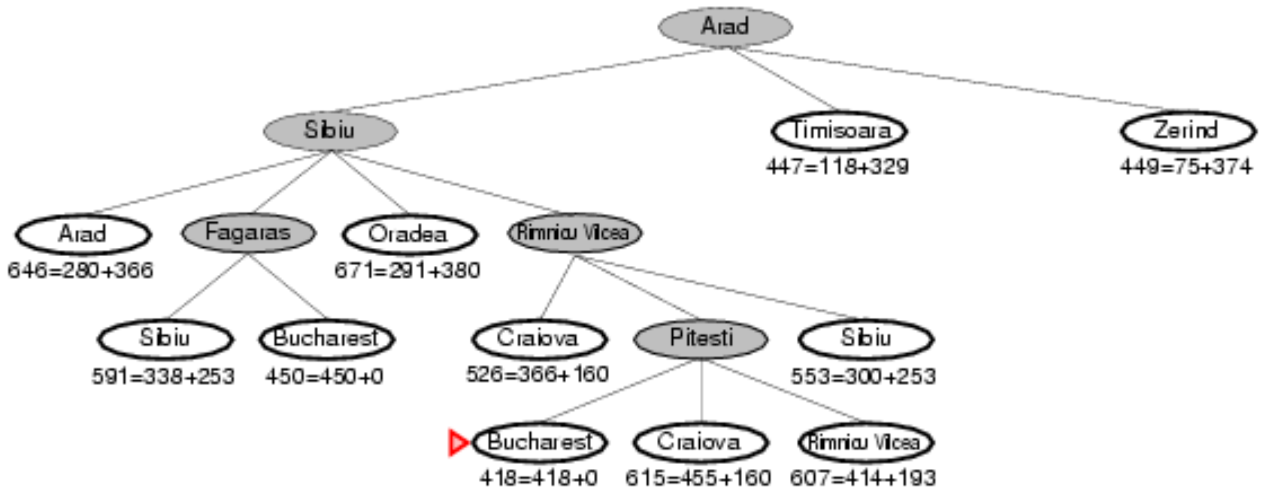
Children: Bucharest/418 (418+0), Craiova/615 (455+160), RimnicuVilcea/607 (414+193),

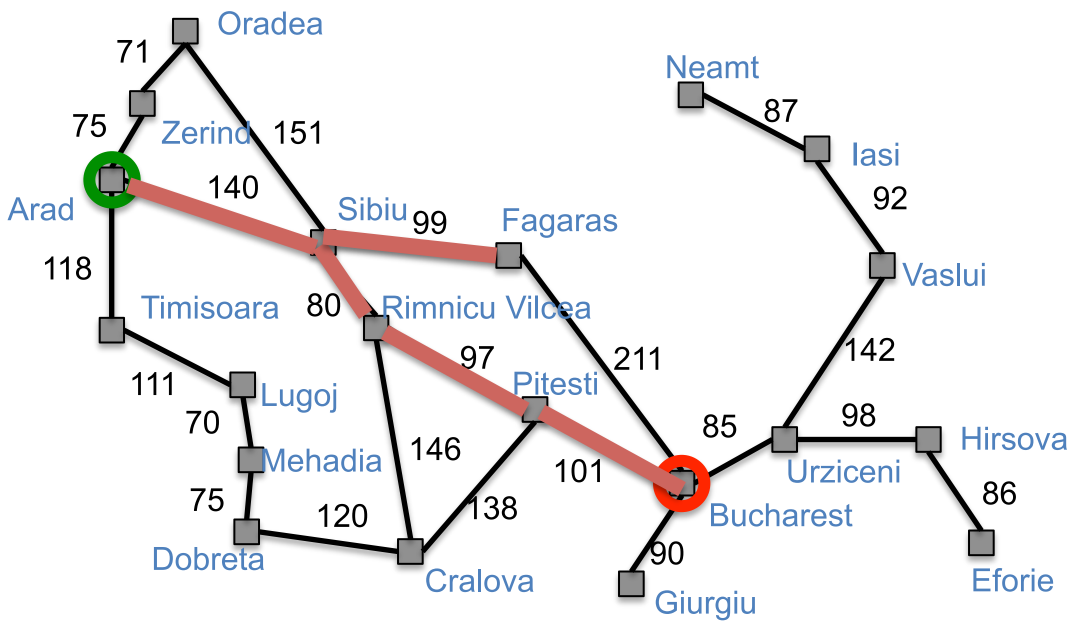
Frontier: Arad/366 (0+366), Sibiu/393 (140+253), Timisoara/447 (118+329), Zerind/449 (75+374), Arad/646 (280+366), Fagaras/415 (239+176), Oradea/671 (291+380), RimnicuVilcea/413 (220+193), Craiova/526 (366+160), Pitesti/417 (317+100), Sibiu/553 (300+253), Bucharest/450 (450+0), Sibiu/591 (338+253), Bucharest/418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



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

Ex: A* 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

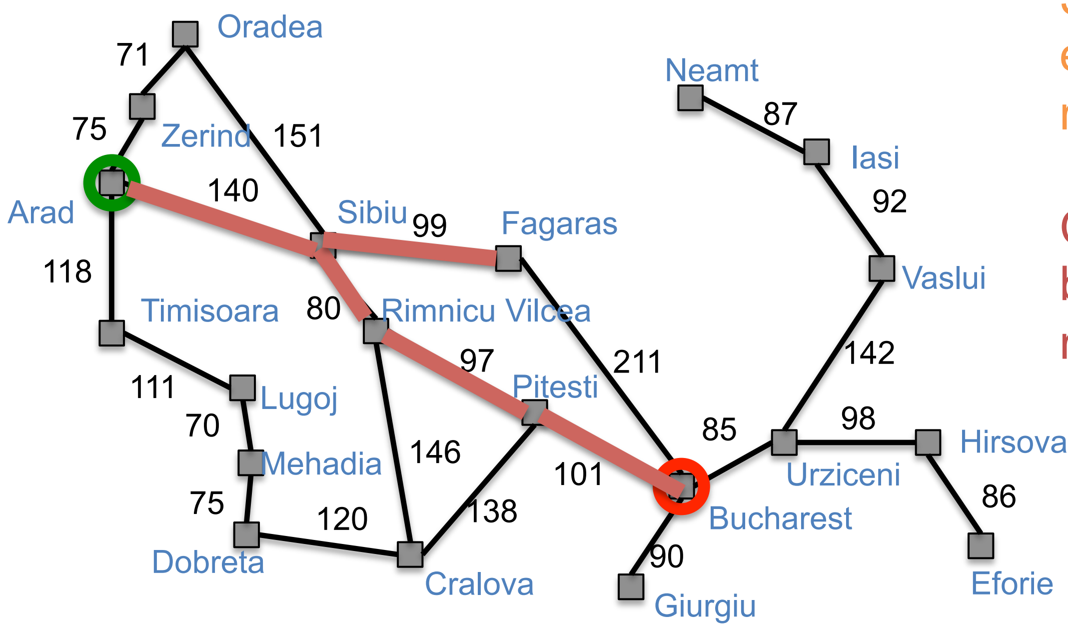
Ex: A* for Romania

(Simulated queue)

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)



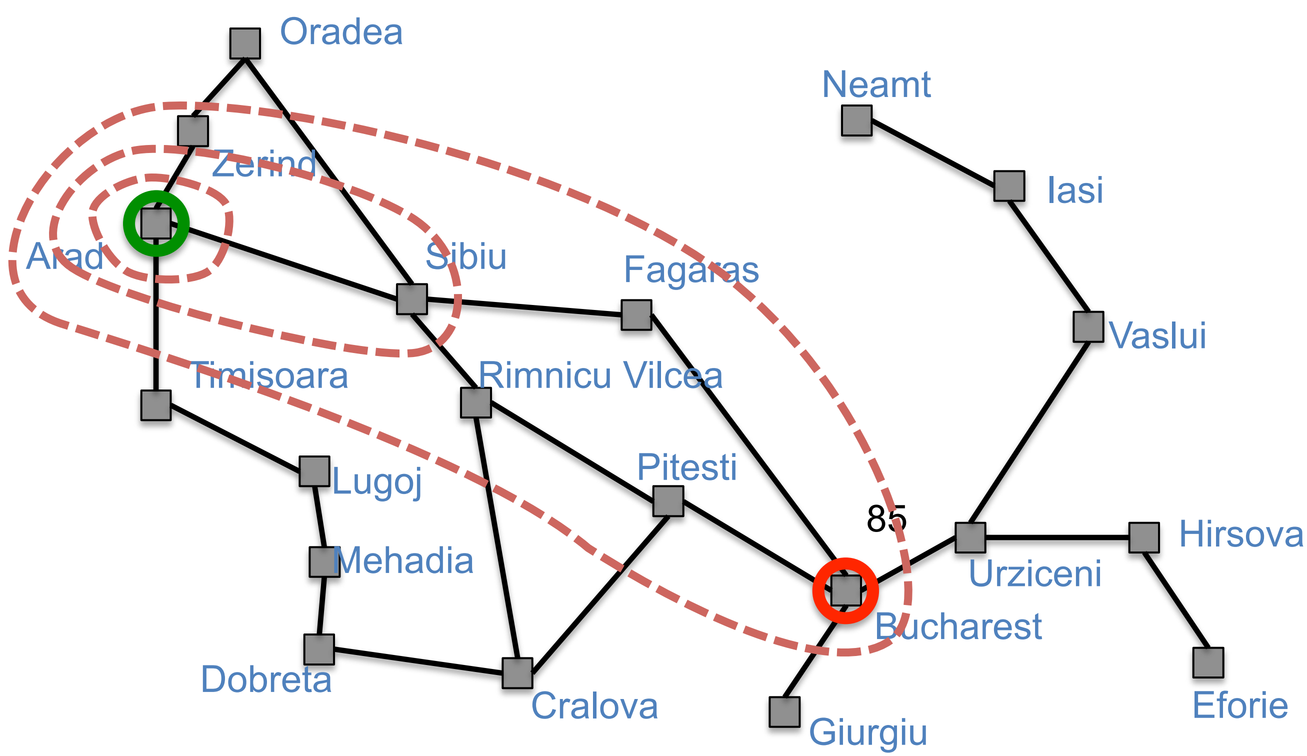
Shorter, more expensive path remains on queue

Cheaper path will be found & 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 $f=f_i$, where $f_i < f_{i+1}$



Properties of A* search

- Complete? Yes
  - Unless infinitely many nodes with f < f(G)
  - Cannot happen if step-cost $\geq \epsilon > 0$
- Time/Space? $O(b^m)$
  - Except if $|h(n) - h^*(n)| \leq O(\log h^*(n))$
- 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

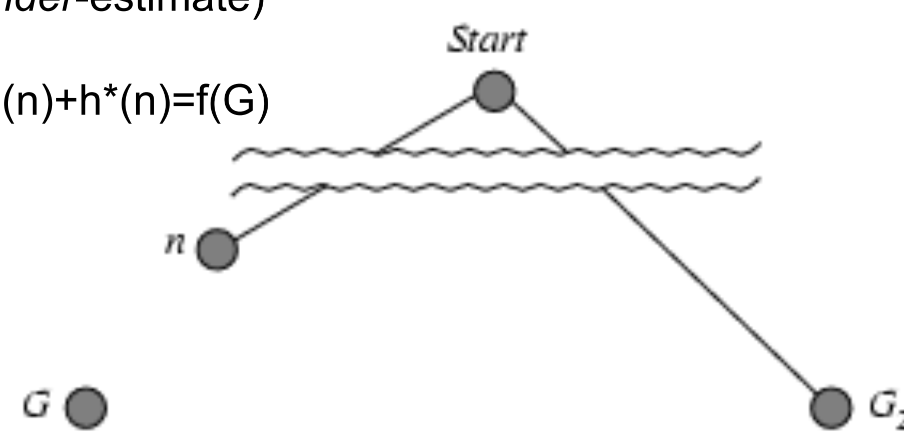
Optimality of A*

- 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



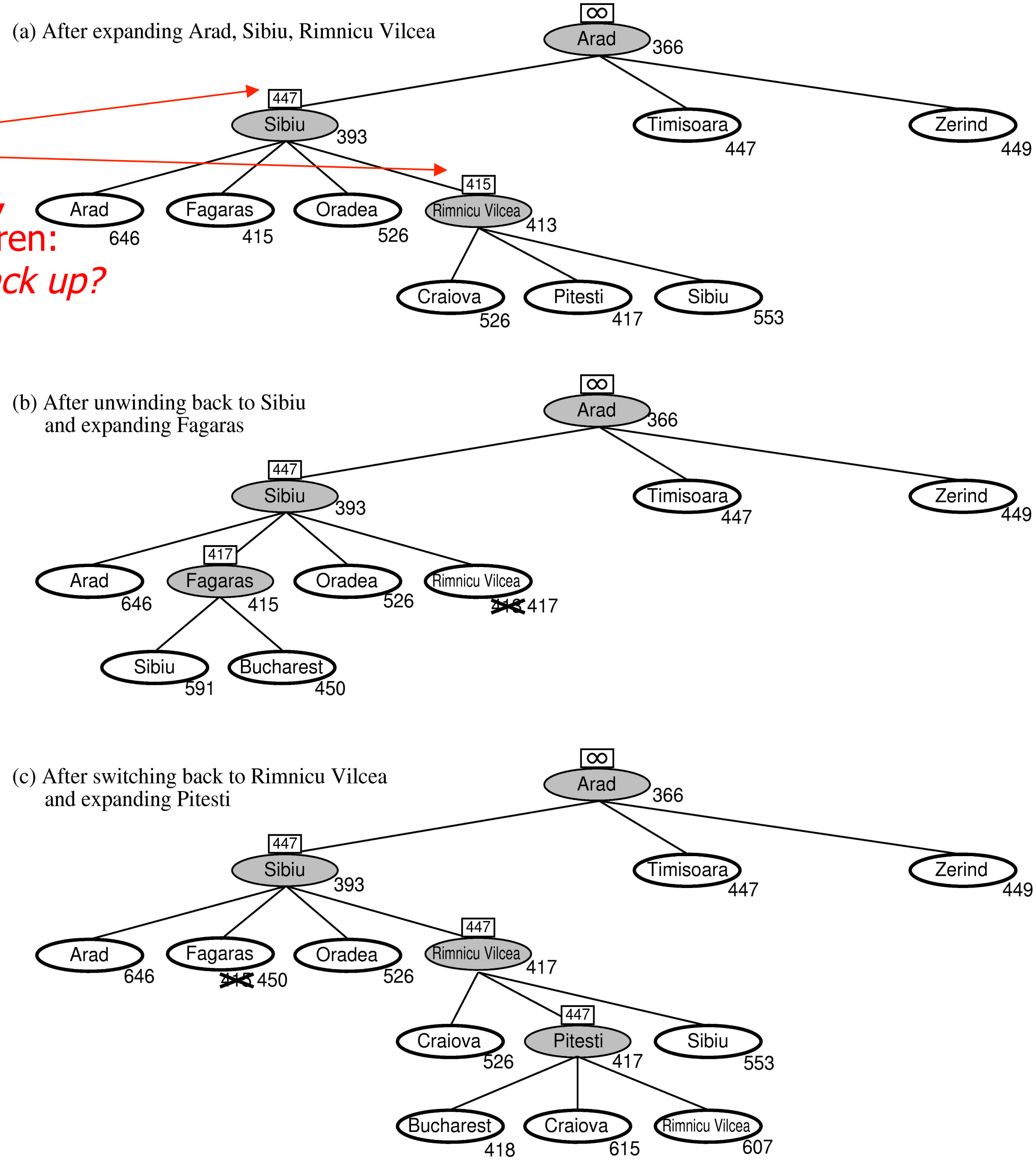
Memory-bounded heuristic search

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

- Idea: recursive best-first search (RBFS)
  - Try something like depth-first search, but don't forget everything about the branches we have partially explored
  - Remember the best f(n) value we have found so far in the branch we're deleting

RBFS:

best alternative
over frontier nodes,
which are not children:
i.e. do I want to back up?



RBFS changes its mind very often in practice.

This is because the f=g+h become more accurate (less optimistic) as we approach the goal. Hence, higher level nodes have smaller f-values and will be explored first.

Problem: We should keep in memory whatever we can.

Simple Memory Bounded A* (SMA*)

- Memory limited, but uses available memory well:
  - Like A*, but if memory full: delete the worst node (largest f-val)
  - Like RBFS, remember the best descendent in deleted branch
  - If there is a tie (equal f-values) we delete the oldest nodes first.
  - SMA* finds the optimal *reachable* solution given memory constraint.
  - Time can still be exponential.

A solution is not reachable if a single path from root to goal does not fit in memory

- Best of search algorithms we've seen
  - Using memory avoids double work; heuristic guides exploration
  - If memory is not a problem, basic A* is easy to code & performs well

SMA* Pseudocode

Note: not in 2nd edition of R&N

```
function SMA*(problem) returns a solution sequence
  inputs: problem, a problem
  static: Queue, a queue of nodes ordered by f-cost

  Queue ← MAKE-QUEUE({MAKE-NODE(INITIAL-STATE[problem])})
  loop do
      if Queue is empty then return failure
      n ← deepest least-f-cost node in Queue
      if GOAL-TEST(n) then return success
      s ← NEXT-SUCCESSOR(n)
      if s is not a goal and is at maximum depth then
         f(s) ← ∞
      else
         f(s) ← MAX(f(n),g(s)+h(s))
      if all of n's successors have been generated then
         update n's f-cost and those of its ancestors if necessary
      if SUCCESSORS(n) all in memory then remove n from Queue
      if memory is full then
         delete shallowest, highest-f-cost node in Queue
         remove it from its parent's successor list
         insert its parent on Queue if necessary
      insert s in Queue
  end
```

Simple memory-bounded A* (SMA*)

(Example with 3-node memory)

Progress of SMA*. Each node is labeled with its *current* *f*-cost.
Values in parentheses show the value of the best forgotten descendant.

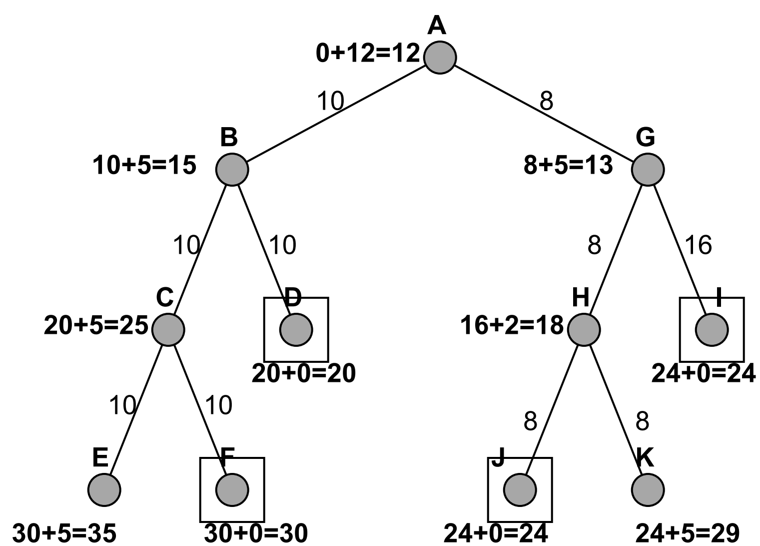
maximal depth is 3, since memory limit is 3. This branch is now useless.

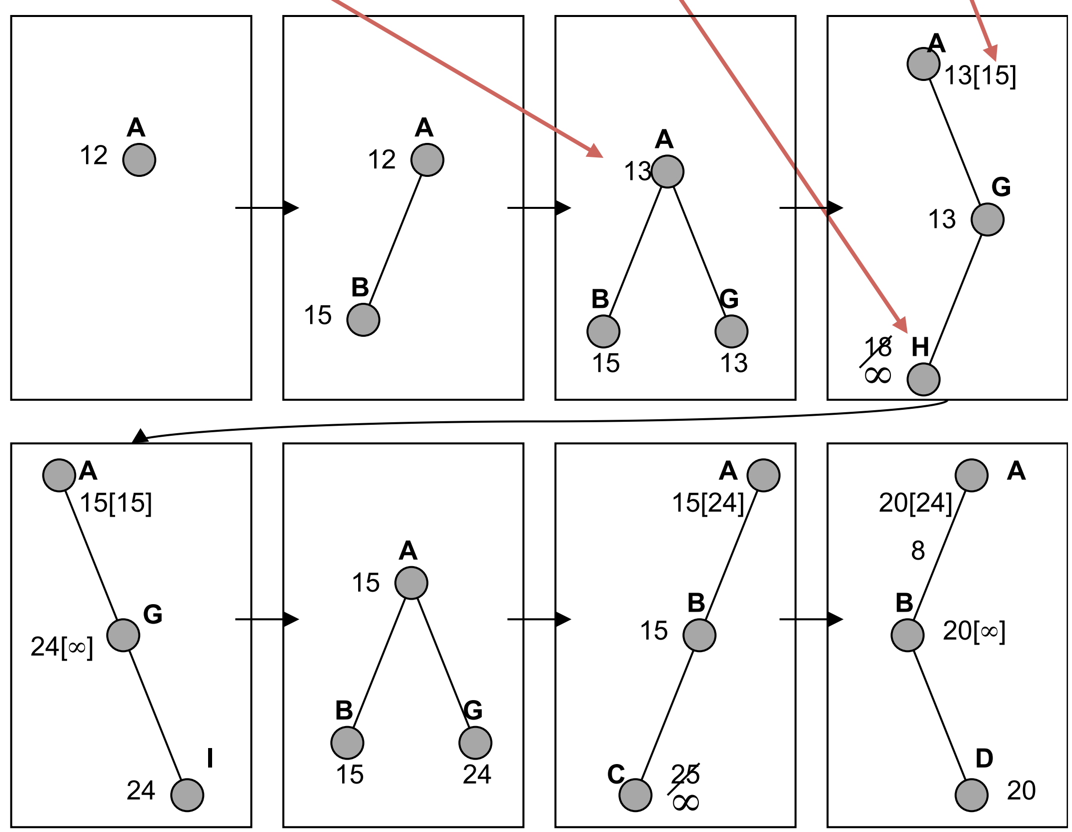
best forgotten node

best estimated solution so far for that node

Search space

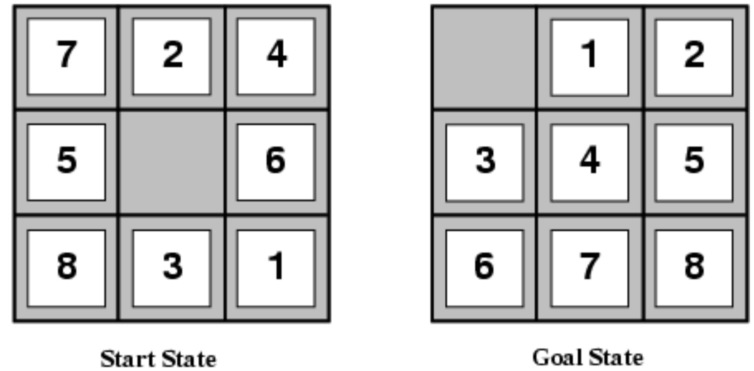
g+h = f □ = goal





Algorithm can tell you when best solution found within memory constraint is optimal or not.

Heuristic functions



- 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

    $h_2(s) = 3+1+2+2+2+3+3+2$ ("Manhattan distance")

    $= 18$

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

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

Ex: 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:

$$
\begin{aligned}
N+1 &= 1 + b^* + (b^*)^2 + \ldots + (b^*)^d \\
&= ((b^*)^d - 1)/(b^* - 1)
\end{aligned}
$$

$$N \approx (b^*)^d \quad \Rightarrow \quad b^* \approx \sqrt[d]{N}$$

  - 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

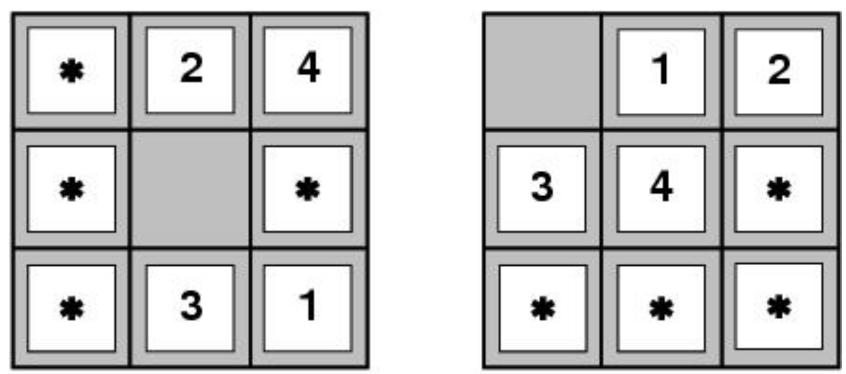
Designing heuristics

- 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
- Ex: 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
  - Ex: 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
  - Ex: 8-Puzzle, dramatic speed-up
  - Still worst-case exponential time complexity (NP-complete)

- Next: local search techniques (hill climbing, GAs, annealing…)
  - Read R&N Ch 4 before next lecture

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