Fundamentals of Artificial Intelligence

Informed Searches

Informed (Heuristic) Search Strategies

- An **informed search strategy** uses problem specific knowledge beyond the definition of the problem itself and it can find solutions more efficiently than can an ***uninformed strategy***.
- **Best-first search** is an algorithm in which a node is selected for expansion based on an **evaluation function, f(n)**.
- The evaluation function is construed as a cost estimate, so the node with the lowest evaluation is expanded first.
- The implementation of best-first graph search is identical to that for **uniform-cost search** except for the use of ***evaluation function f*** instead of ***lowest path cost g*** to order the priority queue.
- The choice of ***evaluation function*** determines the search strategy.
- Most **best-first algorithms** include a **heuristic function, h(n)** as a component of *evaluation function.*

Uniform-Cost Search: Review

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

We have use evaluation function f(n) instead of PATH-COST for Best-first search.

Heuristic Function

- A **heuristic function h(n)** is the estimated cost of the *cheapest path* from the state at node n to a goal state.
- **Heuristic functions** are the most common form of additional knowledge of the problem for ***informed (heuristic) search algorithms***.
- **Heuristic functions** are nonnegative, problem-specific functions, with one constraint: if n is a goal node, then h(n)=0.
- When we use a **heuristic function** to guide our search, we perform **informed (heuristic") search**.

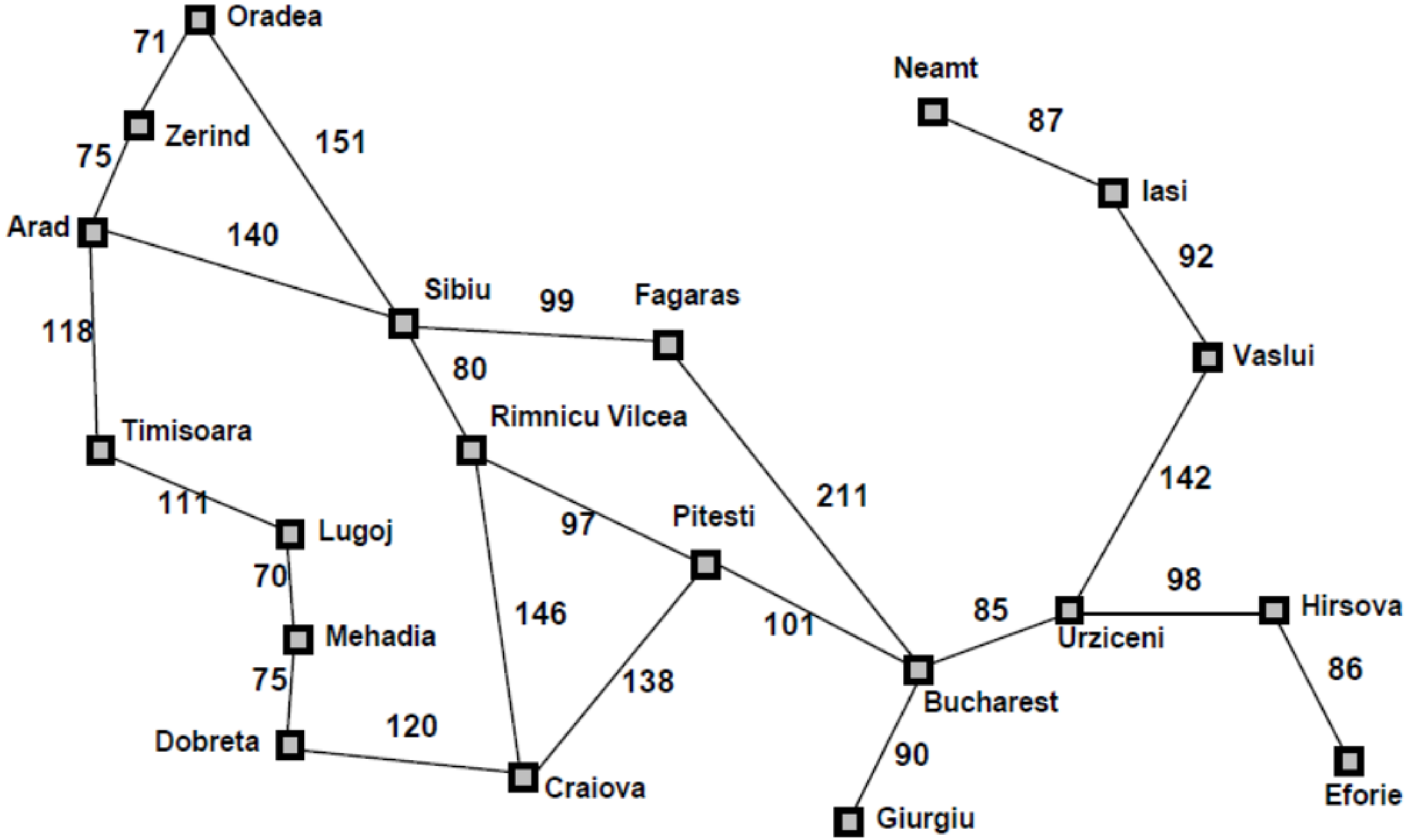
- Some informed (heuristic) searches:
  - Greedy best-first search
  - A*
  - Recursive best-first search (a memory-bounded heuristic search)

Heuristic Function Example
straight line distance heuristic

- For route-finding problems in Romania, we can use the **straight line distance heuristic h_{SLD}**.
- If the goal is Bucharest, we need to know the ***straight-line distances to Bucharest***
- The values of h_{SLD} cannot be computed from the problem description itself.
- Since h_{SLD} is correlated with actual road distances and it is a useful heuristic.

$h_{SLD}(n)$ = straight-line distance from node n to Bucharest

Heuristic Function Example
straight line distance heuristic



straight-line distances to Bucharest

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

- **Greedy best-first search** expands the node that is ***closest to the goal***, on the grounds that this is likely to lead to a solution quickly.
- Greedy search expands the node that appears to be closest to goal

- **Greedy best-first search** evaluates nodes by using just ***the heuristic function***.
- This means that it uses *heuristic function h(n)* as the *evaluation function f(n)* (that is **f(n) = h(n)**).

- For Romania problem, the heuristic function

  $h_{SLD}(n)$ = straight-line distance from n to Bucharest

  as evaluation function

Greedy search example

Node labels are h_{SLD} values

Arad is the initial state.

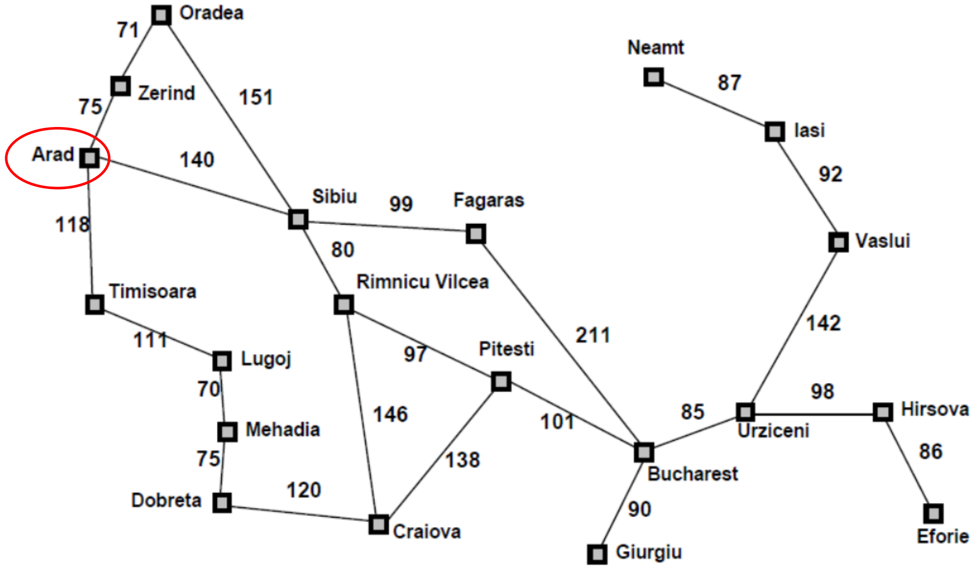
straight-line distances to Bucharest

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

Frontier

Arad 366

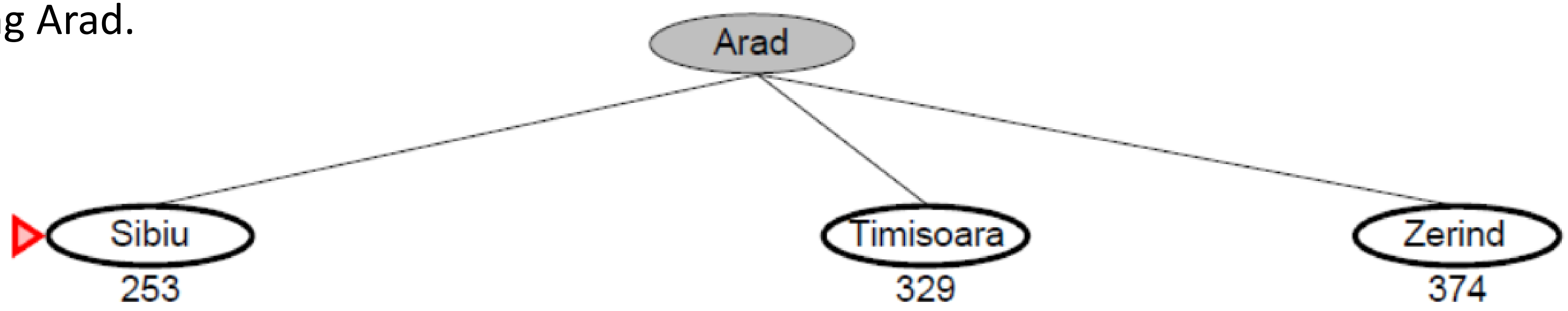
Explored



Greedy search example

Node labels are h_{SLD} values

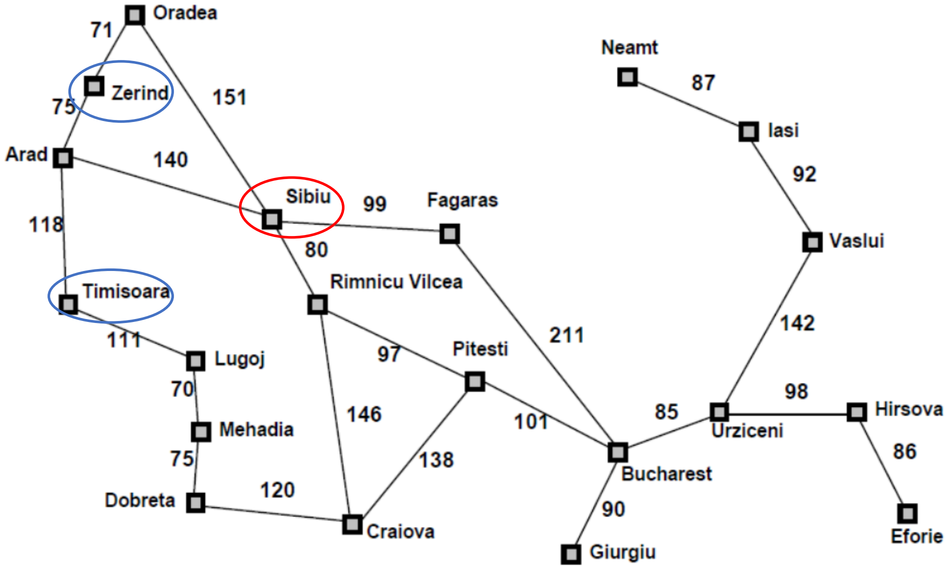
After expanding Arad.



straight-line distances to Bucharest

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

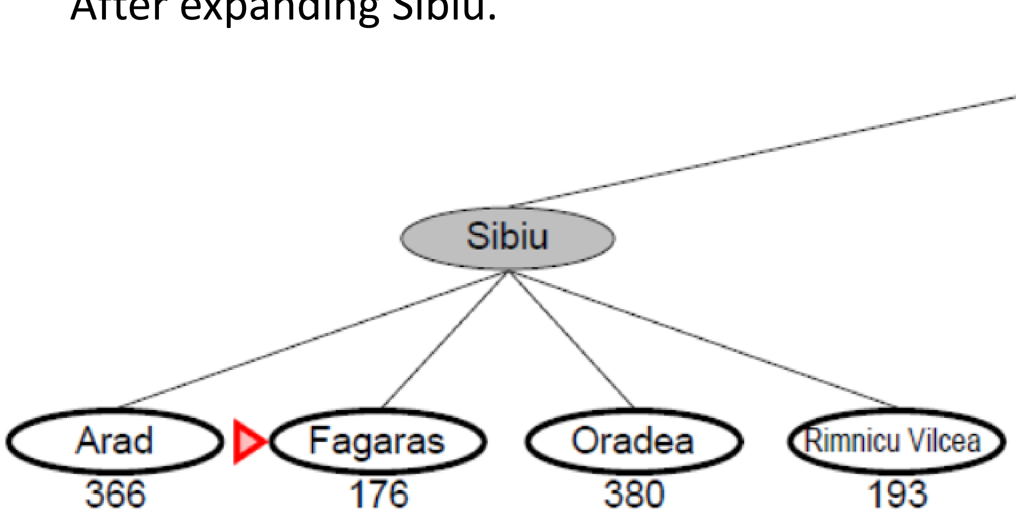
Frontier		Explored
Sibiu	253	Arad
Timisoara	329	
Zerind	374	



Greedy search example

Node labels are h_{SLD} values

After expanding Sibiu.



straight-line distances to Bucharest

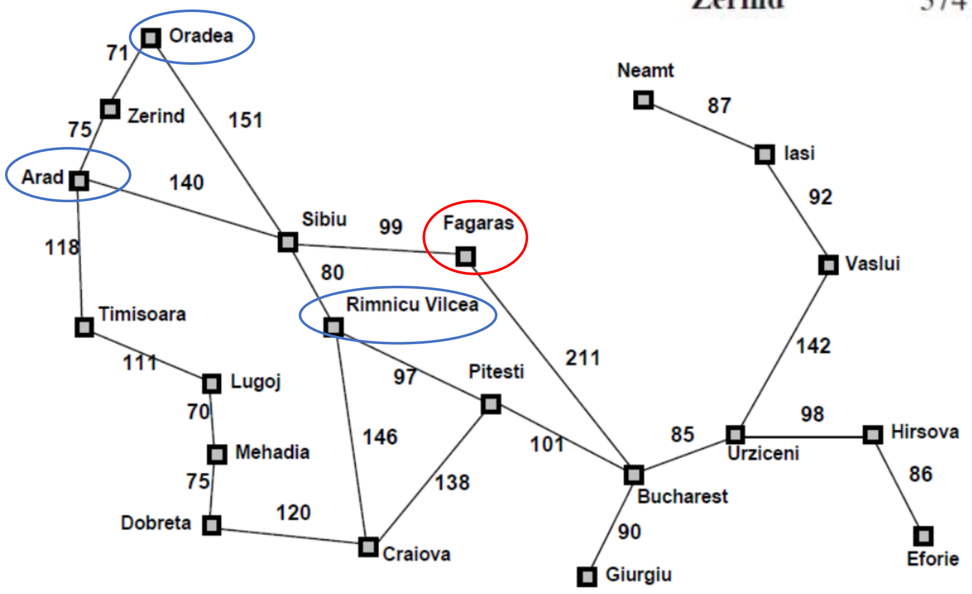
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Frontier

Fagaras	176
RimnicuVil	193
Timisoara	329
Zerind	374
Oradea	380

Explored

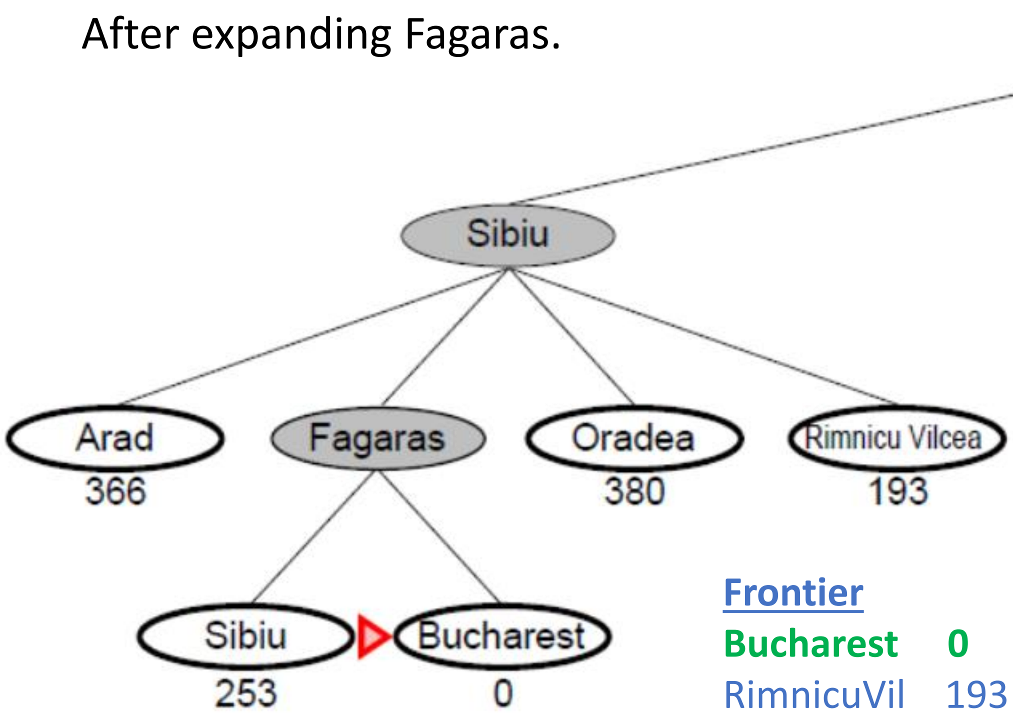
Arad
Sibiu



Greedy search example

Node labels are h_{SLD} values

After expanding Fagaras.



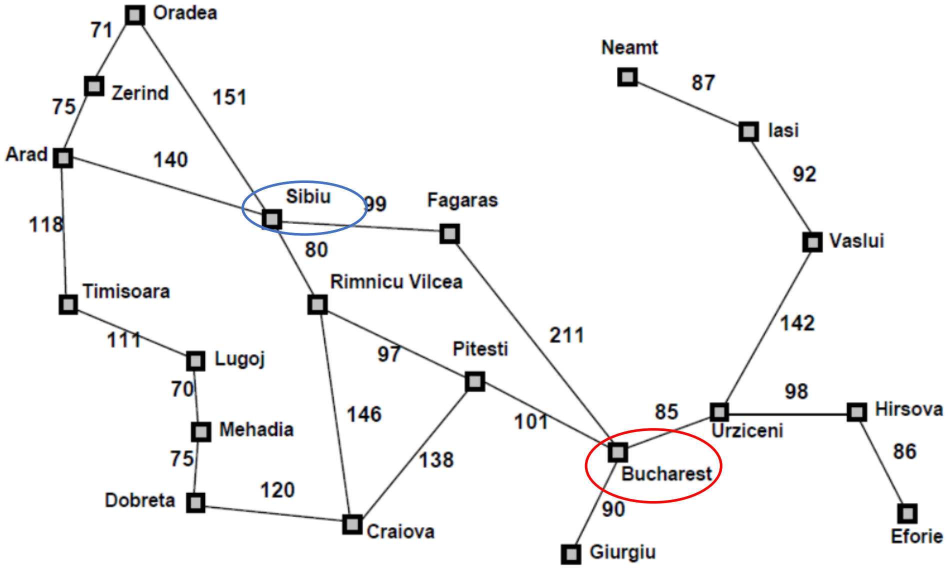
Frontier	
Bucharest	**0**
RimnicuVil	193
Timisoara	329
Zerind	374
Oradea	380

Explored

Arad
Sibiu
Fagaras

straight-line distances to Bucharest

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

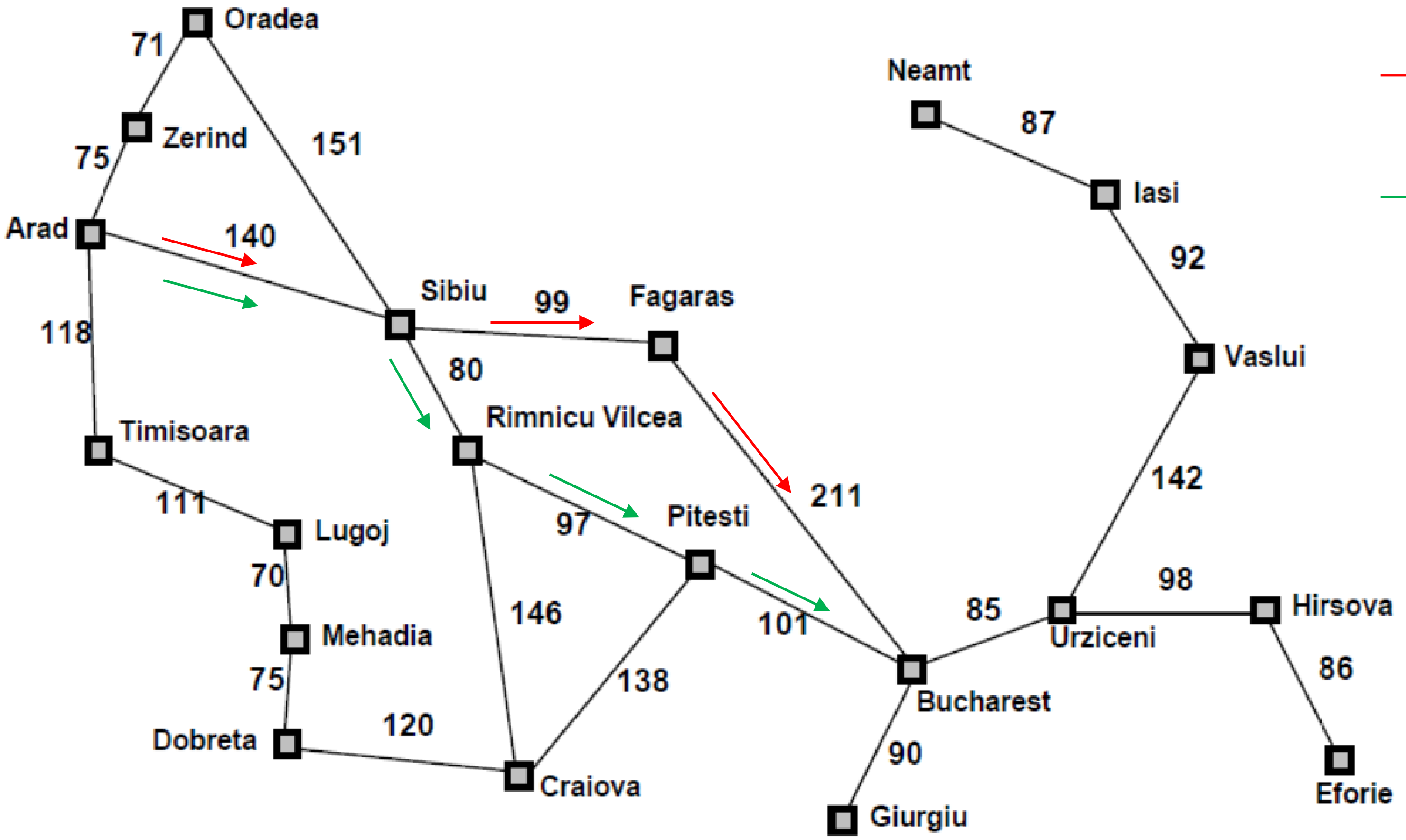


Properties of greedy search

Complete?	NO	It can get stuck in loops without repeated-state checking
	Complete in finite space with repeated-state checking	
Time?	$O(b^m)$	where m is the maximum depth of the search tree.
	but a good heuristic can give dramatic improvement	
Space?	$O(b^m)$	keeps all nodes in memory
Optimal?	NO	nodes expanded in increasing order of path cost

Properties of greedy search

Optimal? NO the path via Sibiu and Fagaras to Bucharest is 32 kilometers longer than the path through Rimnicu Vilcea and Pitesti.



path found by greedy search

optimum path

A* Search

Minimizing the total estimated solution cost

Idea: Avoid expanding paths that are already expensive

Evaluation function f(n) = g(n) + h(n)

g(n) = cost so far to reach node n

h(n) = estimated cost to goal from node n

f(n) = estimated total cost of path through node n to goal

- Since g(n) gives the path cost from the start node to node n, and ***h(n) is the estimated cost of the cheapest path from n to the goal***,

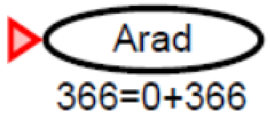
    f(n) = estimated cost of the cheapest solution through n

- ***If heuristic function h(n) satisfies certain conditions***, **A∗ search is both complete and optimal**.

A* Search Example

Node labels are $f(n) = g(n) + h_{SLD}(n)$

Arad is the initial state.



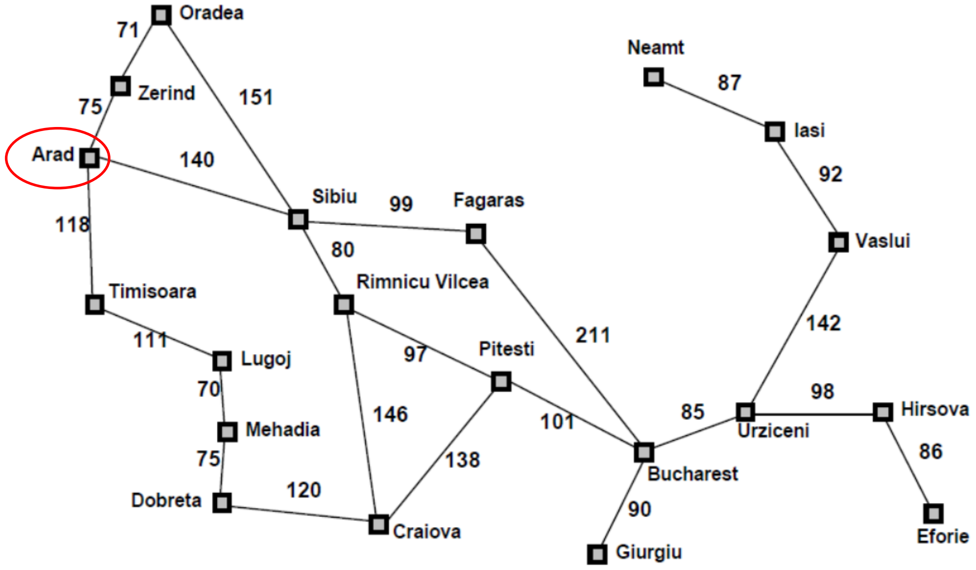
straight-line distances to Bucharest

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

Frontier

Arad 366

Explored



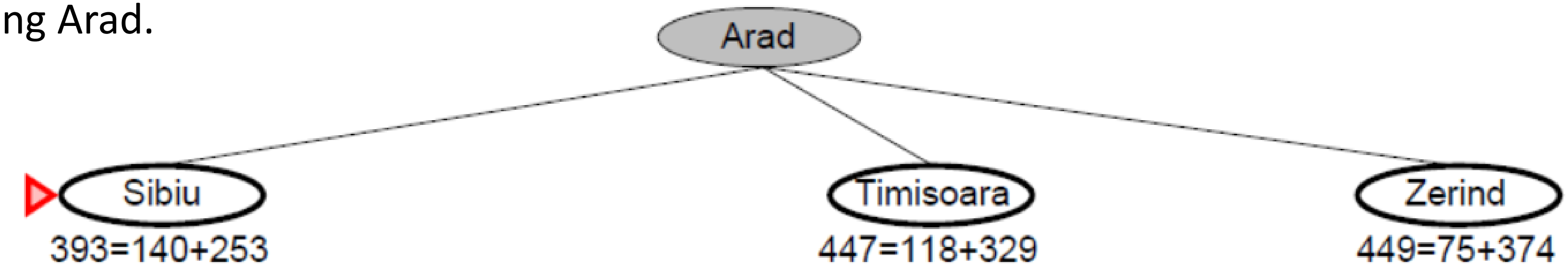
A* Search Example

Node labels are $f(n) = g(n) + h_{SLD}(n)$

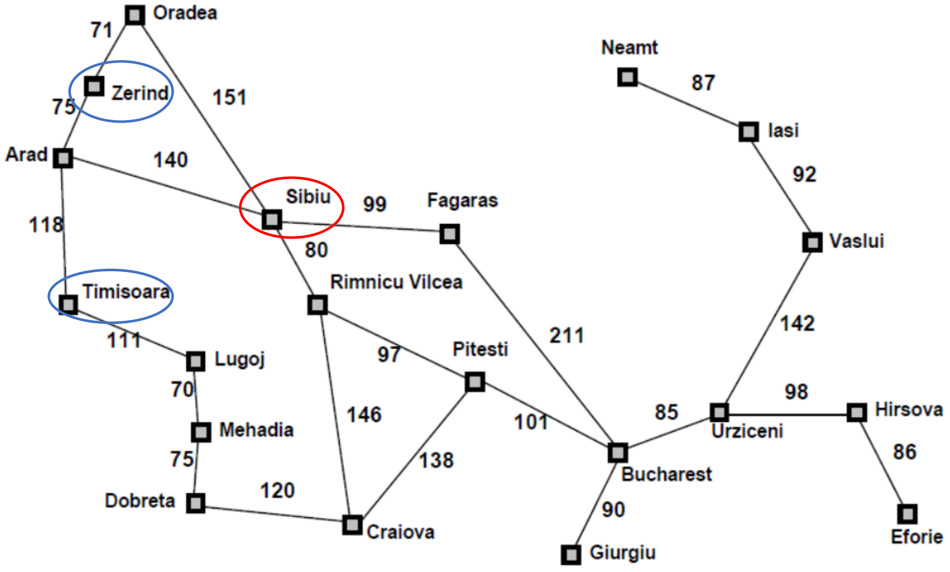
straight-line distances to Bucharest

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

After expanding Arad.



Frontier		**Explored**
Sibiu	393	Arad
Timisoara	447	
Zerind	449	



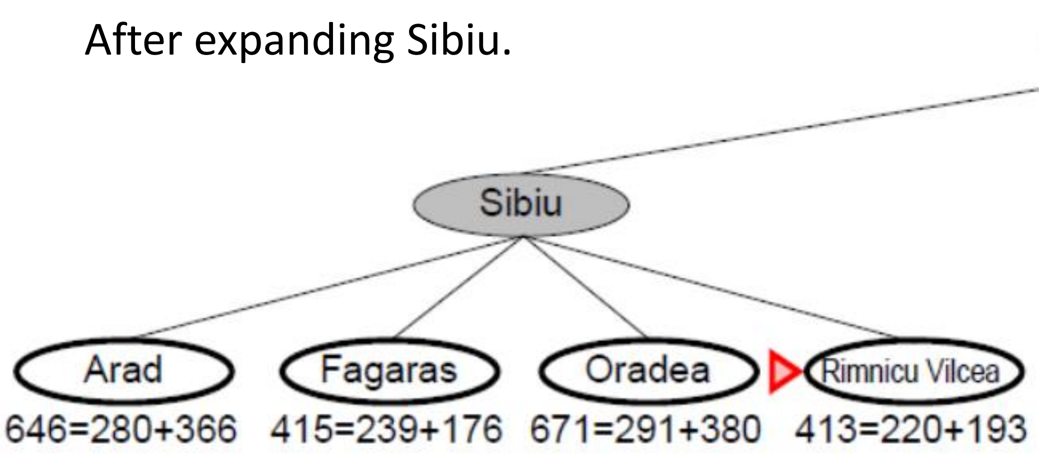
A* Search Example

Node labels are $f(n) = g(n) + h_{SLD}(n)$

straight-line distances to Bucharest

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

After expanding Sibiu.

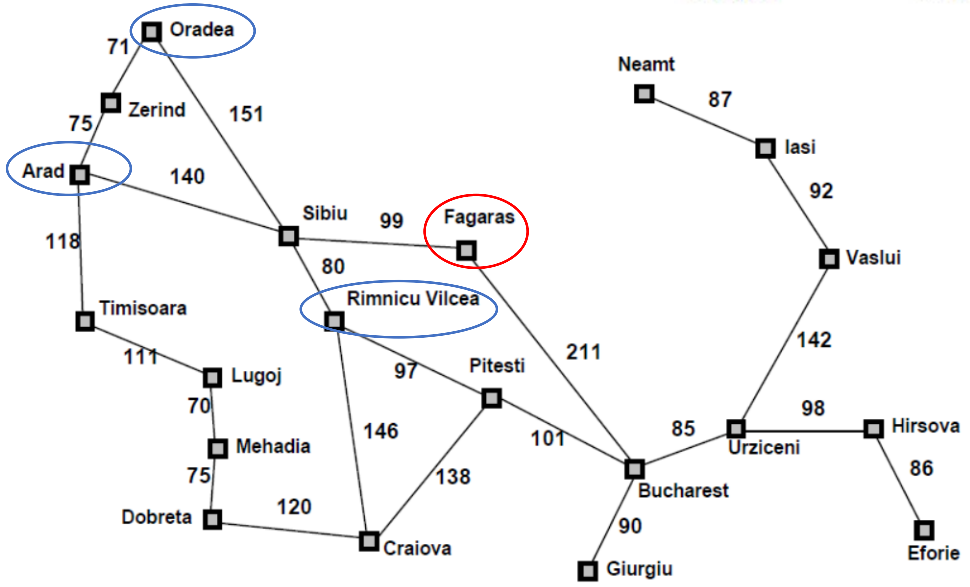


Frontier

RimnicuVil	413
Fagaras	415
Timisoara	447
Zerind	449
Oradea	671

Explored

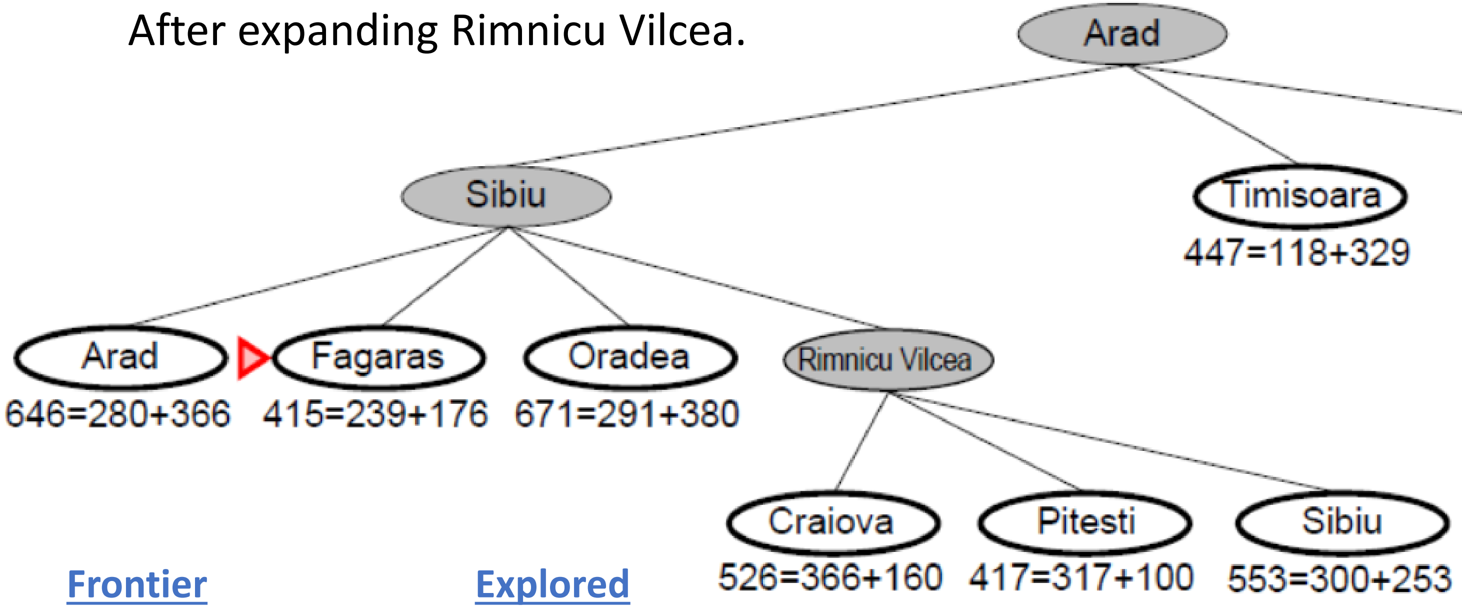
Arad
Sibiu



A* Search Example

Node labels are $f(n) = g(n) + h_{SLD}(n)$

After expanding Rimnicu Vilcea.



straight-line distances to Bucharest

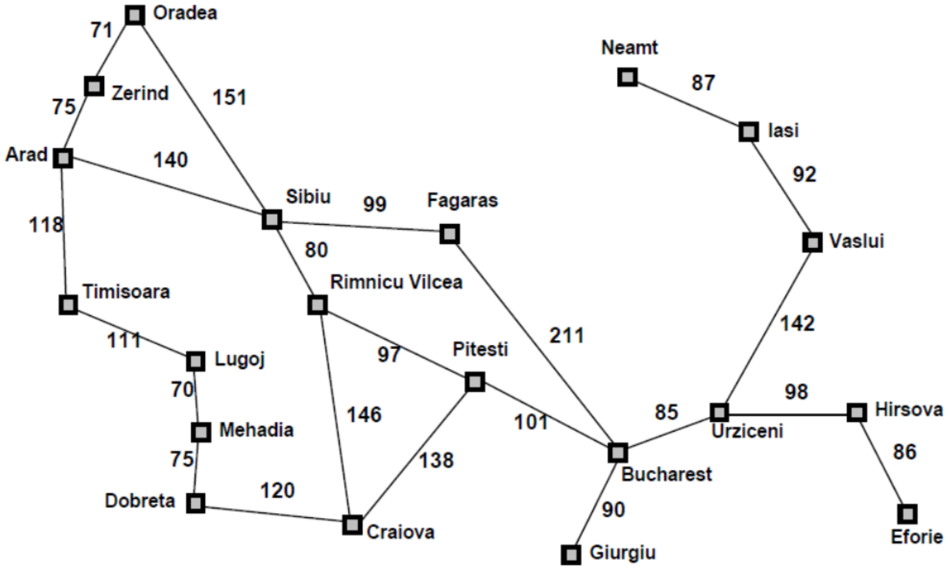
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Frontier

Fagaras	415
Pitesti	417
Timisoara	447
Zerind	449
Craiova	526
Oradea	671

Explored

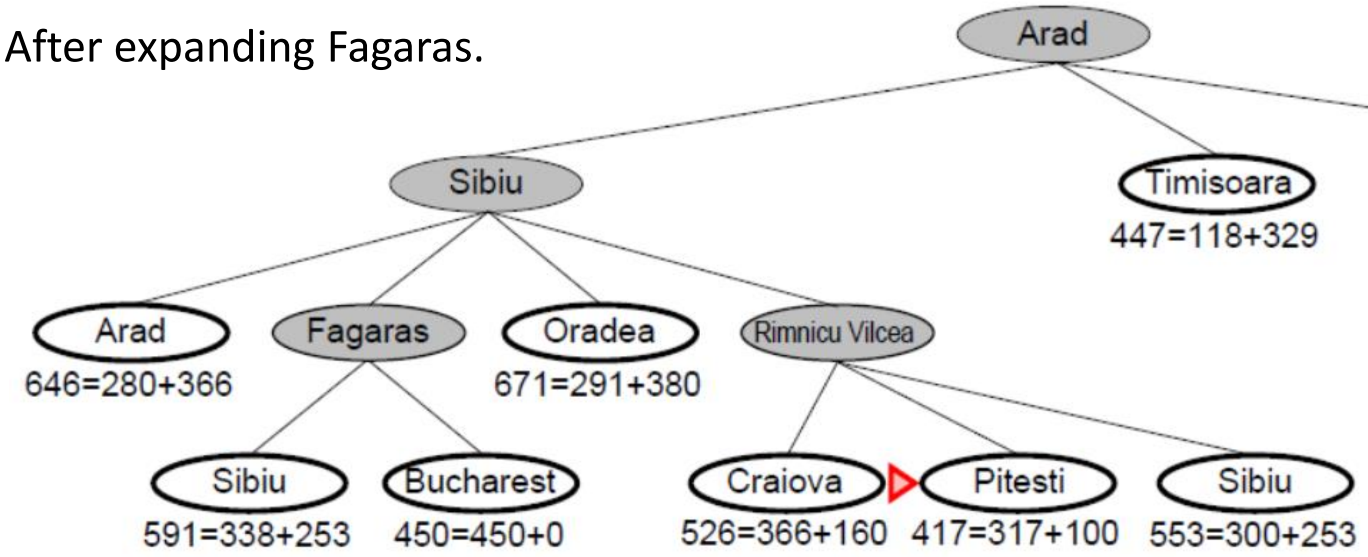
Arad
Sibiu
RimnicuVil



A* Search Example

Node labels are f(n) = g(n) + h_{SLD}(n)

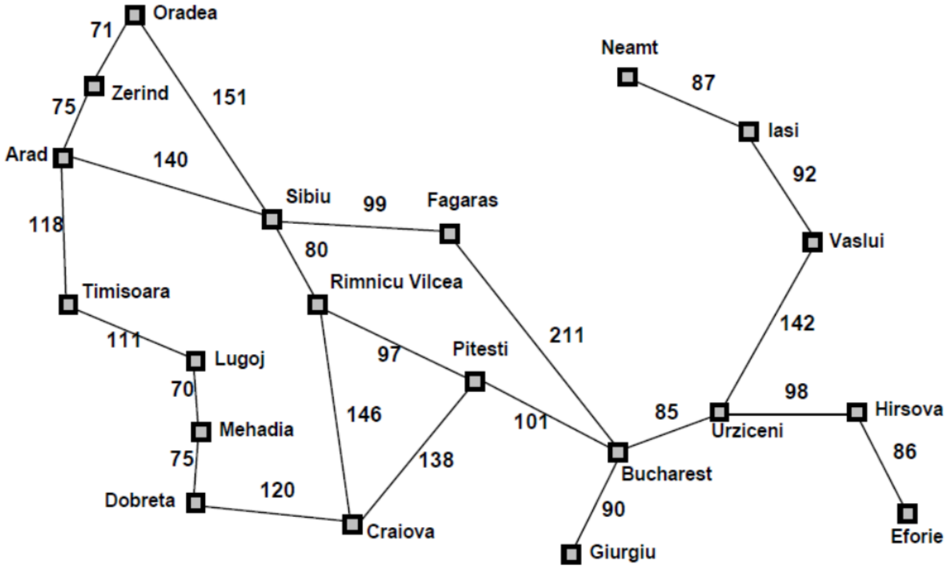
After expanding Fagaras.



straight-line distances to Bucharest

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

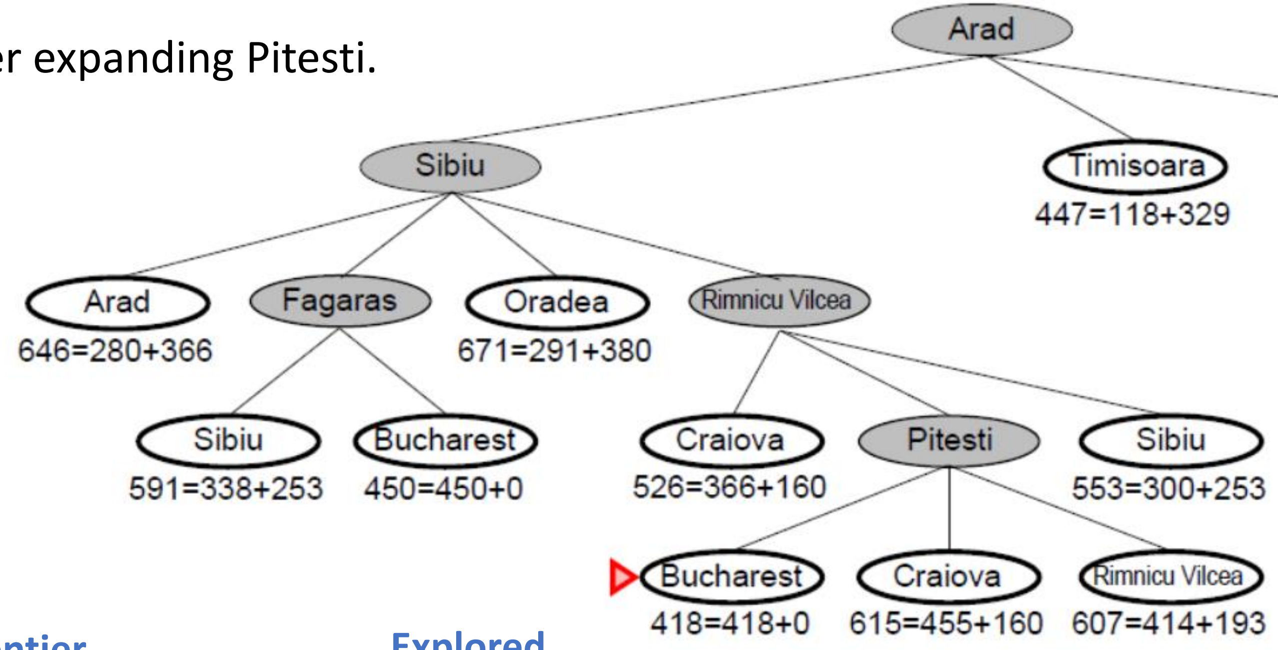
Frontier		Explored
Pitesti	417	Arad
Timisoara	447	Sibiu
Zerind	449	RimnicuVil
Bucharest	450	Fagaras
Craiova	526	
Oradea	671	



A* Search Example

Node labels are $f(n) = g(n) + h_{SLD}(n)$

After expanding Pitesti.



straight-line distances to Bucharest

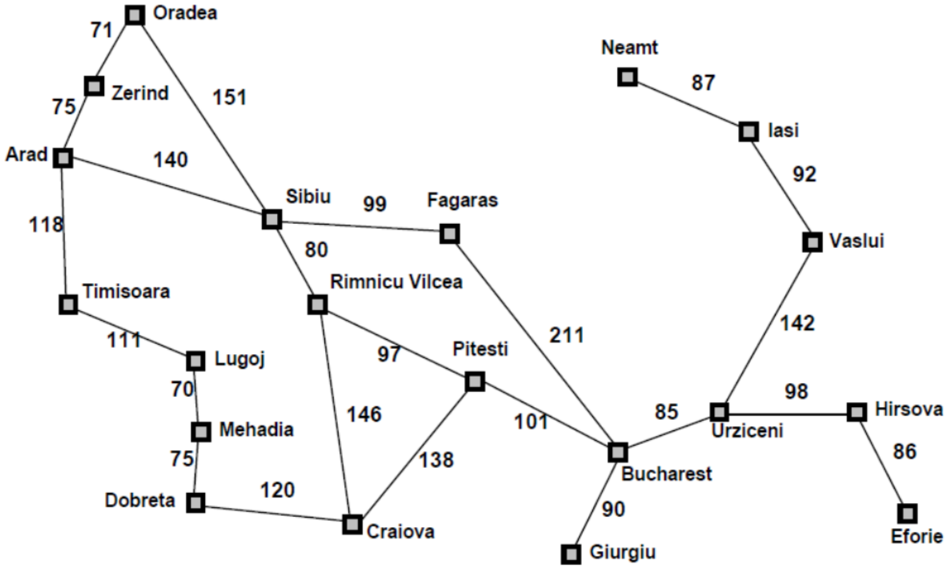
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Frontier

Bucharest	~~450~~ 418
Timisoara	447
Zerind	449
Craiova	526
Oradea	671

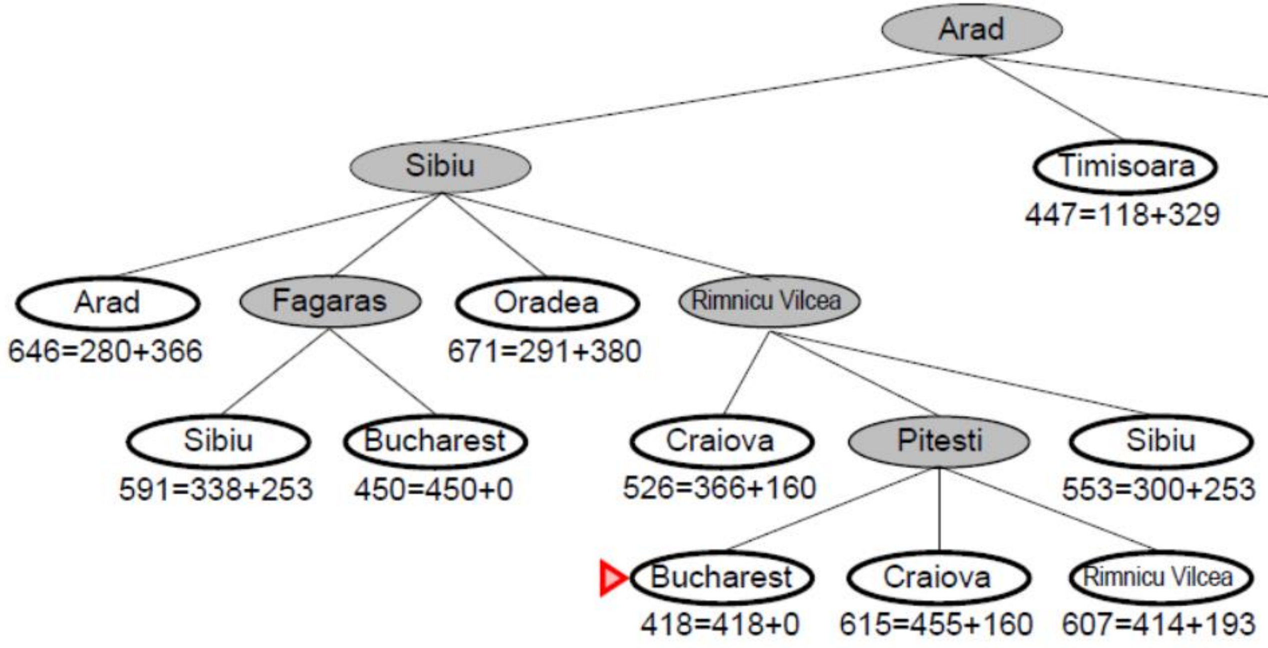
Explored

Arad
Sibiu
RimnicuVil
Fagaras
Pitesti



A* Search Example

Node labels are $f(n) = g(n) + h_{SLD}(n)$



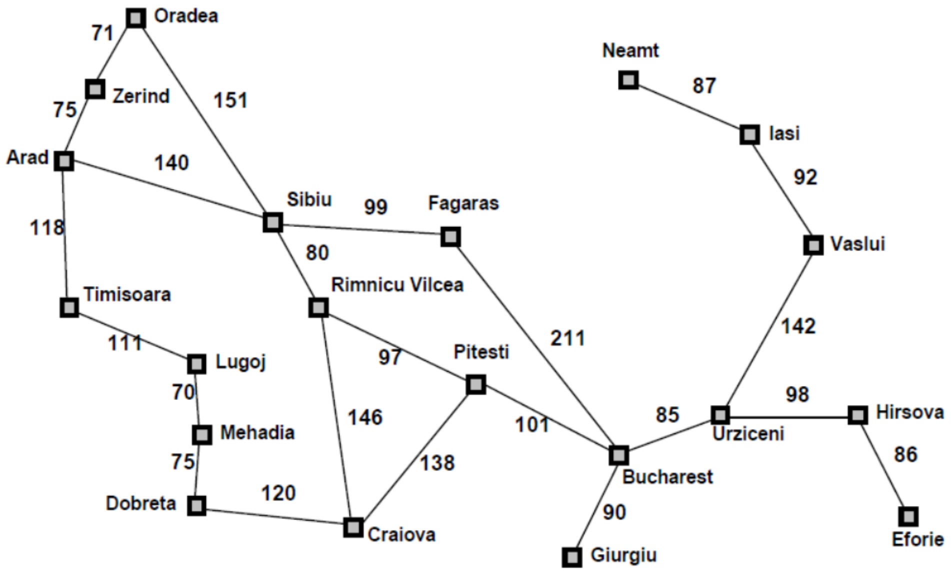
straight-line distances to Bucharest

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

Path found by A*: Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest
A* Path Cost: 140+80+97+101 = 418

Optimum Path Cost: 418

A* finds an optimum path.



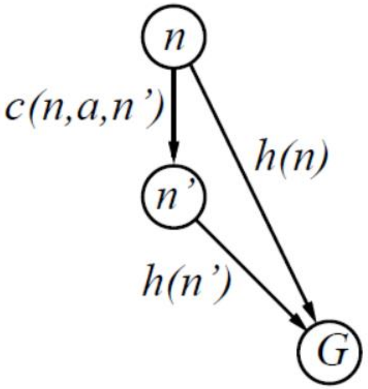
Conditions for Optimality: Admissibility and Consistency

- The first *condition for optimality* is that heuristic function h(n) must be an **admissible heuristic**.
- **An admissible heuristic is one that never overestimates the cost to reach the goal (optimistic).**
- **A heuristic h(n) is admissible if for every node n, h(n)≤ C(n) where C(n) is the true cost to reach the goal state from the state of node n.**
  - **Straight-line distance heuristic h_{SLD}(n)** is **admissible** because the shortest path between any two points is a straight line. **h_{SLD}(n)** never overestimates actual distance.
  - If h(n) is admissible, f(n) never overestimates the cost to reach the goal because f(n)=g(n)+h(n) and g(n) is the actual cost to reach node n.
- A **heuristic h(n) is consistent** if, for every node n and every successor n′ of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n′ plus the estimated cost of reaching the goal from n′:

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

- h_{SLD}(n) is consistent.
- **If h(n) is admissible and consistent, then A* is complete and optimal.**

Optimality of A* (proof)



- **If h(n) is consistent ($h(n) \leq c(n, a, n') + h(n')$), then the values of f(n) along any path are non-decreasing.**
  - The proof follows directly from the definition of consistency.
  - Suppose n′ is a successor of n; then $g(n')=g(n) + c(n, a, n')$ for some action a, and we have

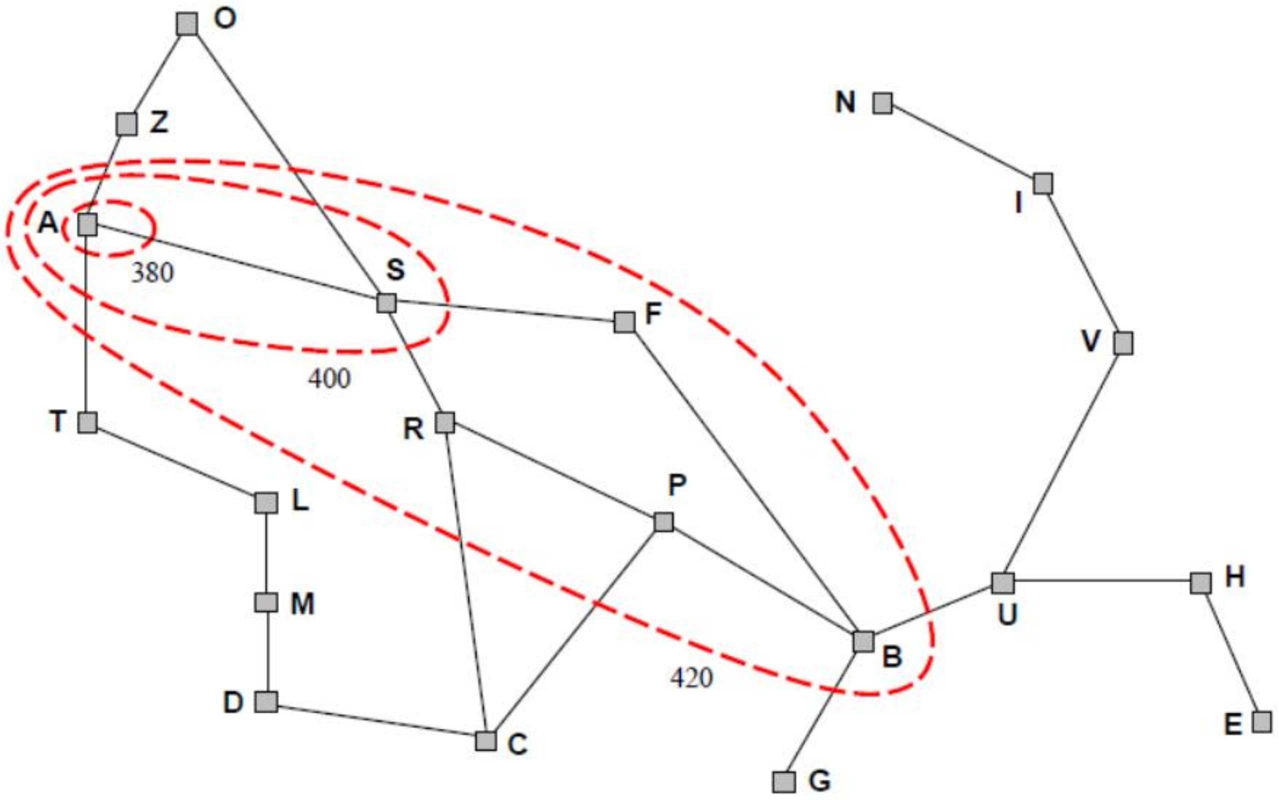
$$f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n) .$$

- **Whenever A∗ selects a node n for expansion, the optimal path to that node has been found.**
  - If this is not the case, there would have to be another frontier node n′ on the optimal path from the start node to n, because f is non-decreasing along any path, n′ would have lower f-cost than n and would have been selected first.

- **From these two preceding observations, it follows that the sequence of nodes expanded by A∗ is in non-decreasing order of f(n).**
- **Hence, the first goal node selected for expansion must be an optimal solution because f is the true cost for goal nodes (h(Goal)=0) and all later goal nodes will be at least as expensive.**

Optimality of A*

- A* expands nodes 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*

Complete?	YES	unless there are infinitely many nodes with $f \leq f(Goal)$
Time?	Exponential (depends on h(n))	
Space?	$O(b^m)$	keeps all nodes in memory
Optimal?	YES	A* cannot expand f_{i+1} until f_i is finished.

- A* expands all nodes with $f(n) < C^*$ where C^* is the optimal cost
- A* expands some nodes with $f(n) = C^*$
- A* expands no nodes with $f(n) > C^*$

Recursive best-first search
Memory-bounded heuristic search

- **Recursive best-first search (RBFS)** is a simple recursive algorithm that attempts to mimic the operation of *standard best-first search*, but using only ***linear space***.
- Its structure is similar to that of a ***recursive depth-first search***, but rather than continuing indefinitely down the current path, it uses the f-limit variable to keep track of the f-value of the best alternative path available from any ancestor of the current node.
- If the current node exceeds this limit, the recursion unwinds back to the alternative path.
- As the recursion unwinds, RBFS replaces the f-value of each node along the path with a backed-up value (the best f-value of its children).
- In this way, RBFS remembers the f-value of the best leaf in the forgotten subtree and can therefore decide whether it's worth

Recursive best-first search

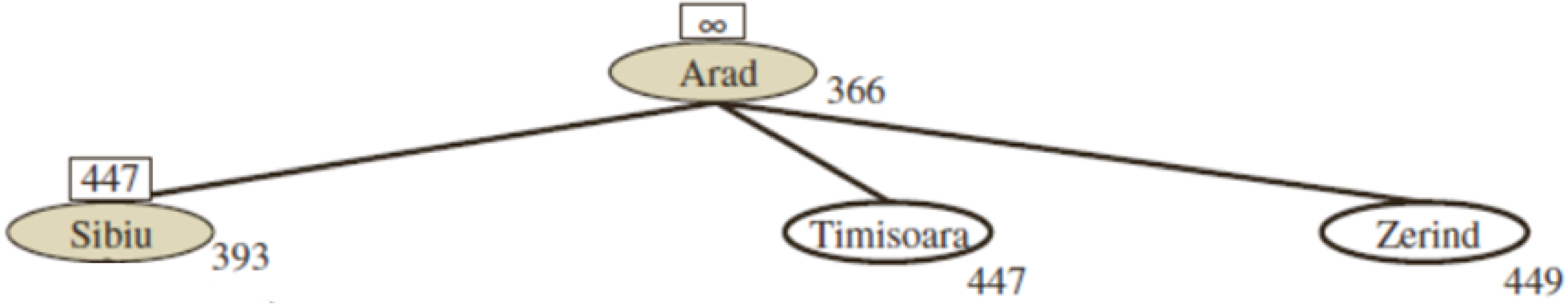
```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution, or failure
   return RBFS(problem, MAKE-NODE(problem.INITIAL-STATE), ∞)

function RBFS(problem, node, f_limit) returns a solution, or failure and a new f-cost limit
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  successors ← [ ]
  for each action in problem.ACTIONS(node.STATE) do
      add CHILD-NODE(problem, node, action) into successors
  if successors is empty then return failure, ∞
  for each s in successors do  /* update f with value from previous search, if any */
      s.f ← max(s.g + s.h, node.f))
  loop do
      best ← the lowest f-value node in successors
      if best.f > f_limit then return failure, best.f
      alternative ← the second-lowest f-value among successors
      result, best.f ← RBFS(problem, best, min(f_limit, alternative))
      if result ≠ failure then return result
```

Stages in an RBFS search for the shortest route to Bucharest.

Arad will be expanded.

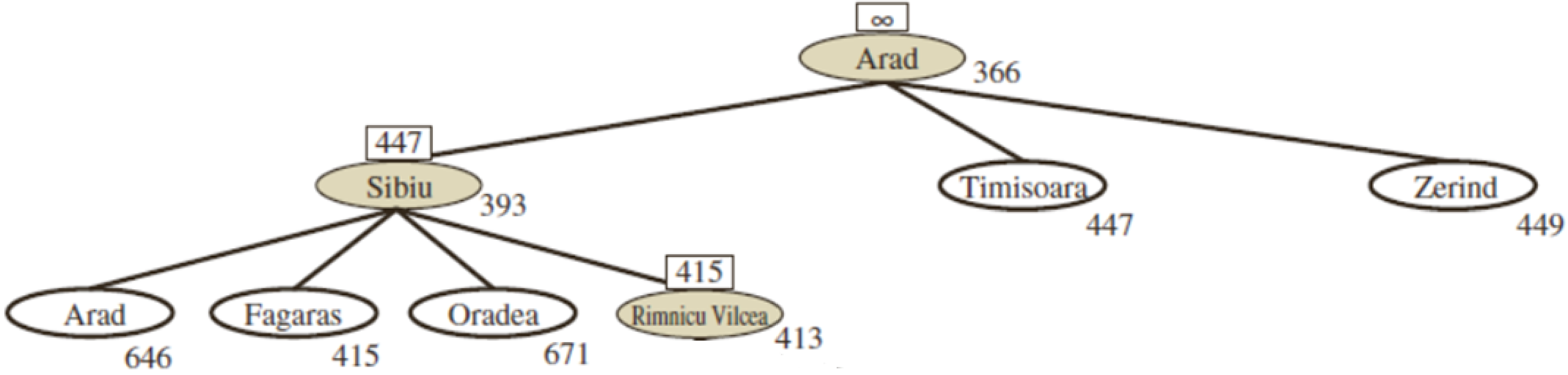
Stages in an RBFS search for the shortest route to Bucharest.

Sibiu will be expanded



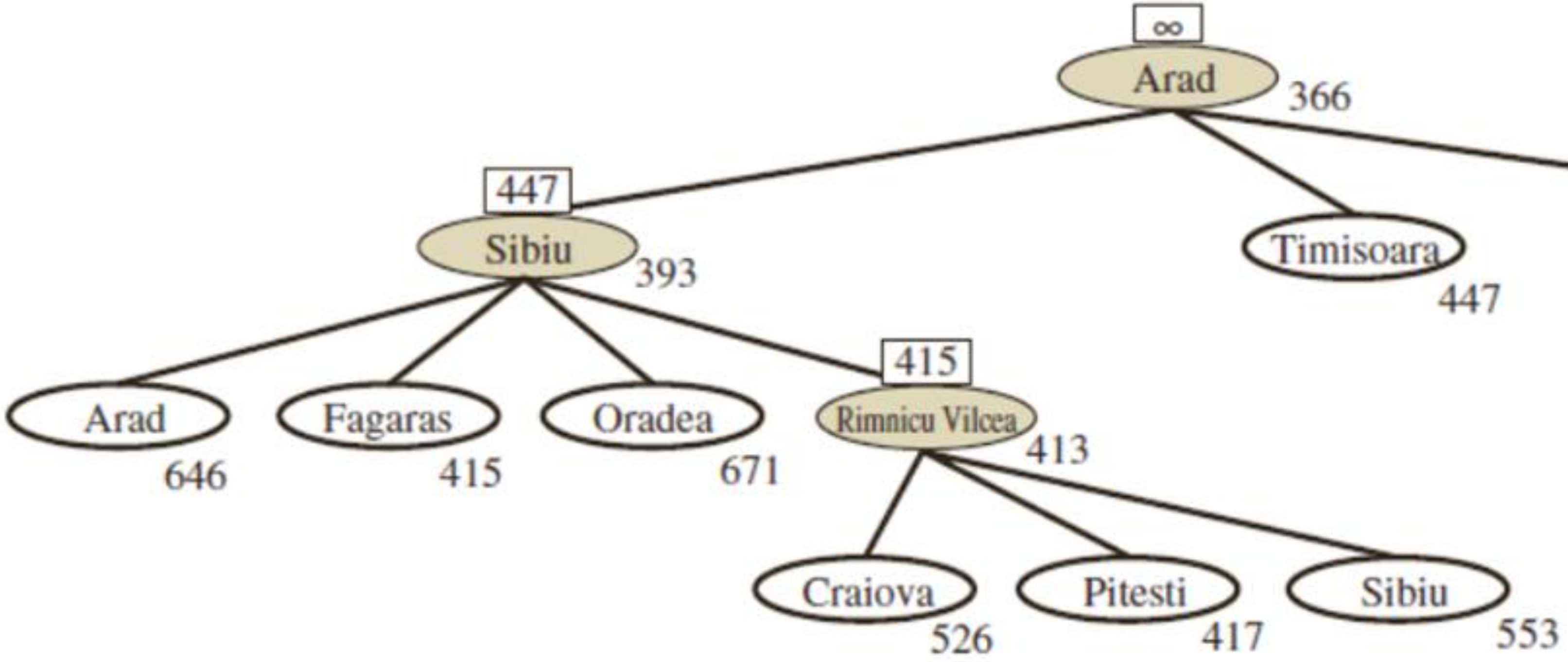
Stages in an RBFS search for the shortest route to Bucharest.

Rimnicu Vilcea will be expanded



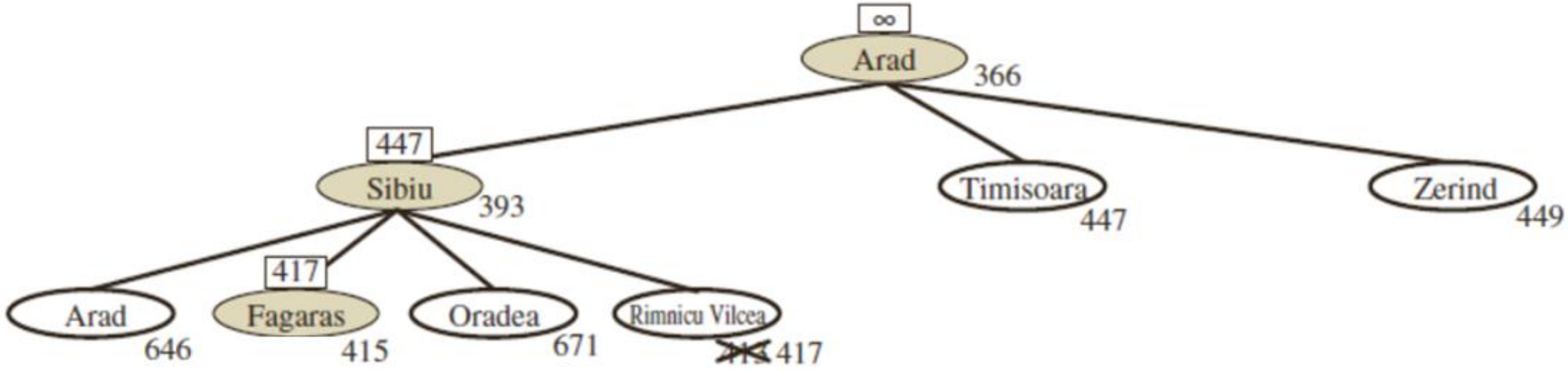
Stages in an RBFS search for the shortest route to Bucharest.

Unwind to Sibiu.



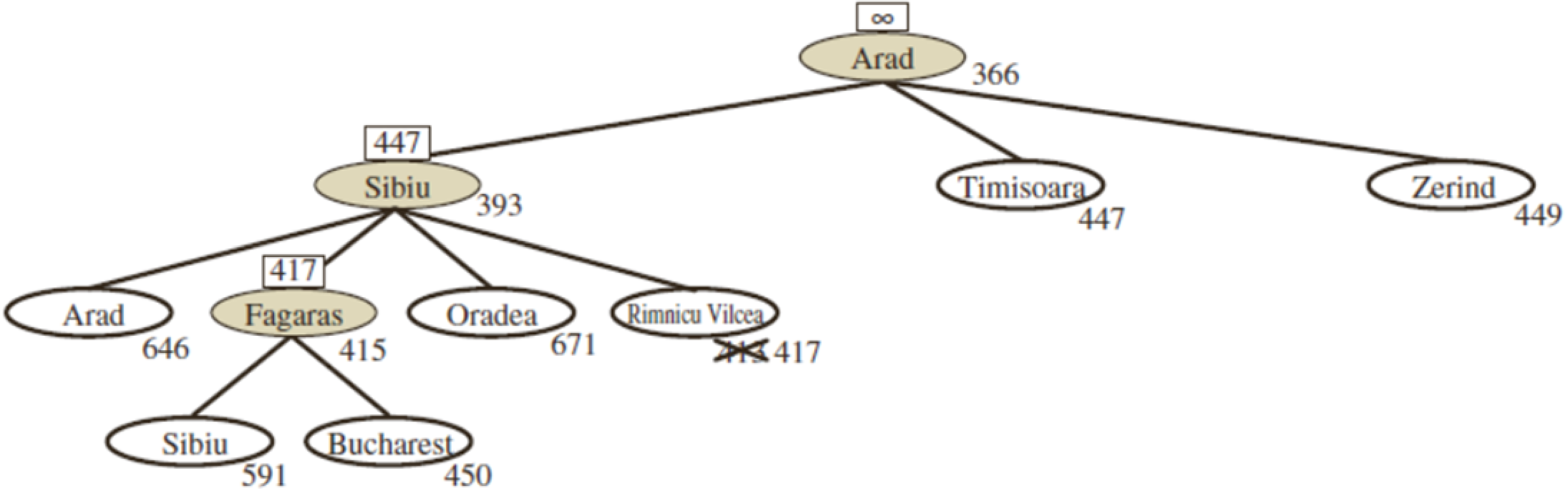
Stages in an RBFS search for the shortest route to Bucharest.

After unwinding to Sibiu. Fagaras will be expanded



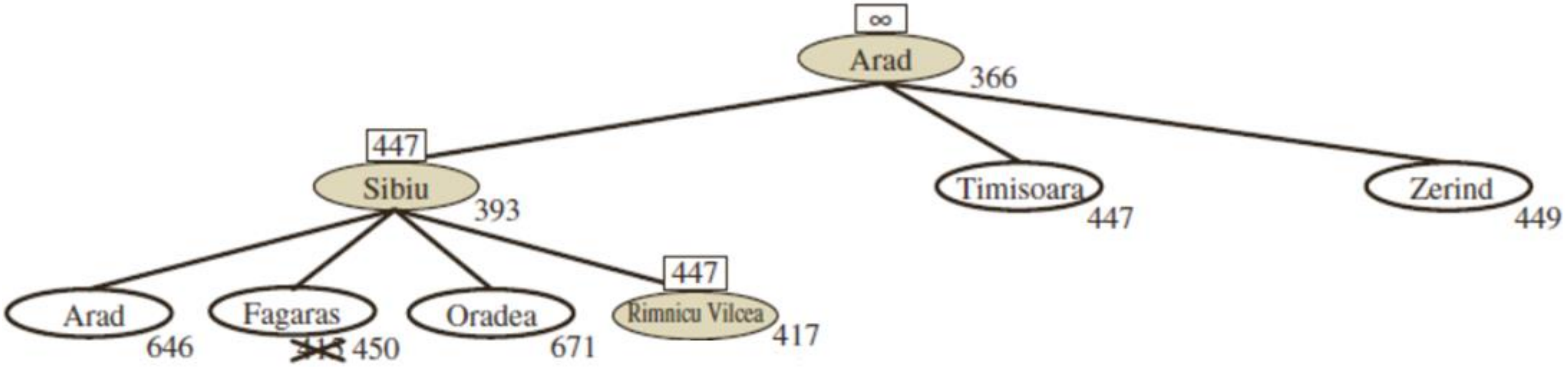
Stages in an RBFS search for the shortest route to Bucharest.

Unwind to Sibiu again.



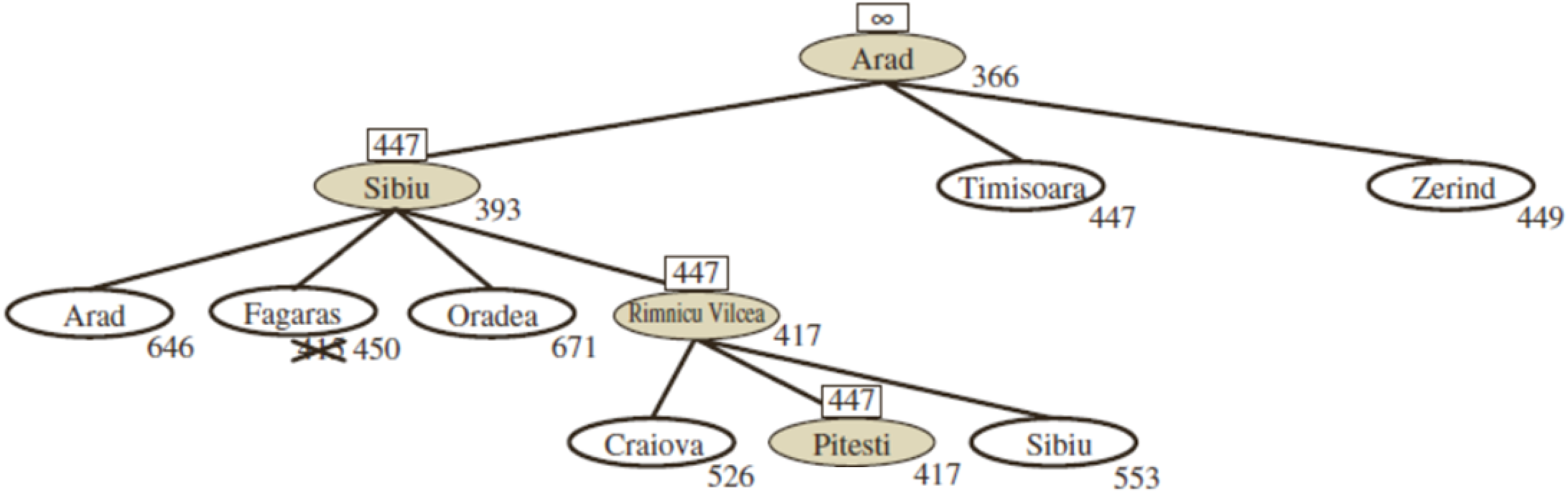
Stages in an RBFS search for the shortest route to Bucharest.

After Unwinding to Sibiu again. Rimnicu Vilcea will be re-expanded.



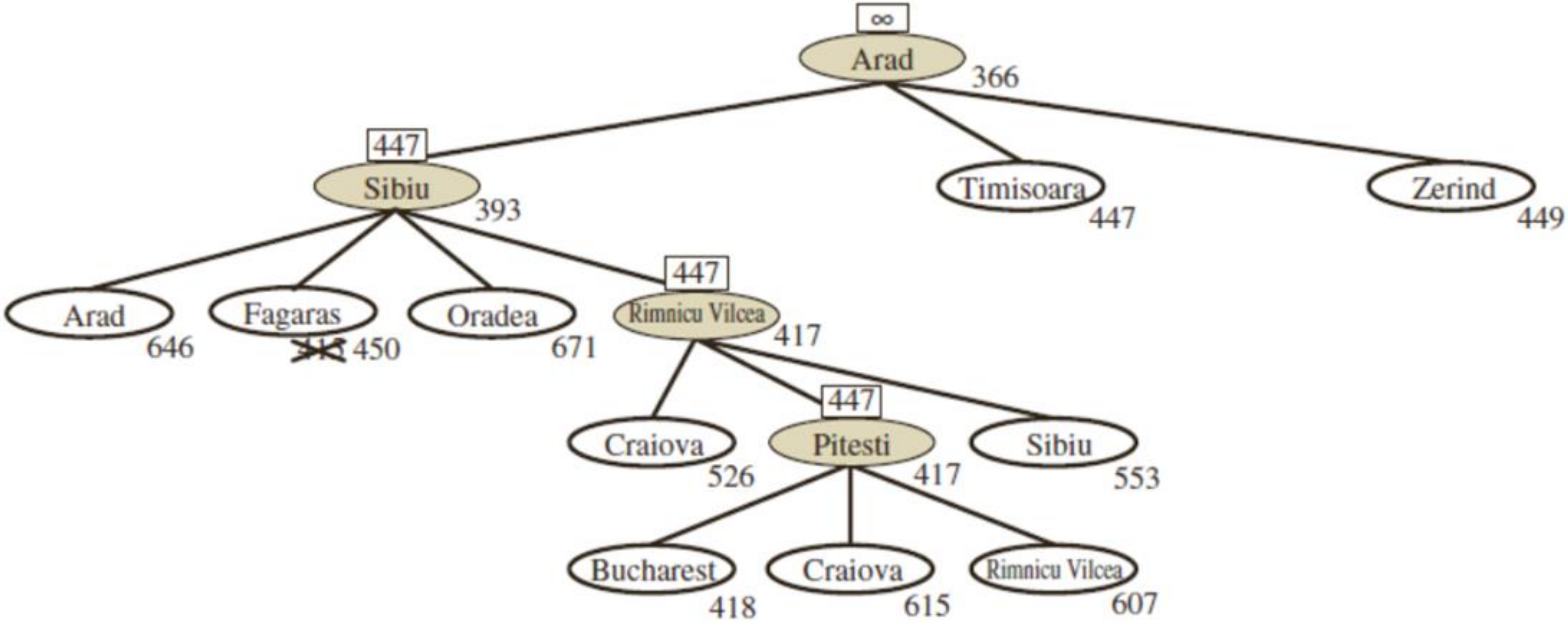
Stages in an RBFS search for the shortest route to Bucharest.

Pitesti will be expanded.



Stages in an RBFS search for the shortest route to Bucharest.

After expanding Pitesti, the best successor is Bucharest. RBFS will be called with Bucharest and Goal is reached.



Heuristic Functions

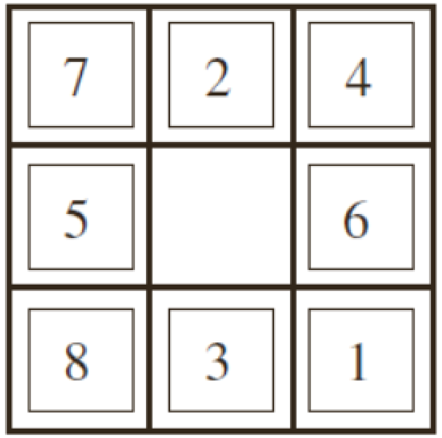
- The 8-puzzle was one of the earliest heuristic search problems.
- The average solution cost for a randomly generated 8-puzzle instance is about 22 steps.
- The branching factor is about 3.
  - in the middle ➔four moves
  - in a corner ➔ two moves
  - along an edge ➔ three moves
- A search algorithm may look at 170,000 states to solve a random 8-puzzle instance, because 9!/2=181,400 distinct states are reachable.
- A search algorithm may look at 10^{13} s.tates to solve a random 15-puzzle instance

➔ We need a good heuristic function.

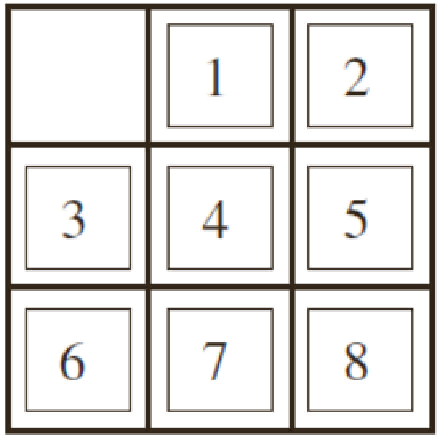
➔ If we want to find the shortest solutions by using A∗,
we need a heuristic function that never overestimates the number of steps to the goal.

Heuristic Functions

- A typical instance of the 8-puzzle. The solution is 26 steps long.



Start State



Goal State

Heuristic Functions

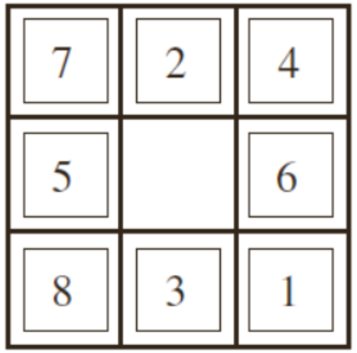
Two admissible heuristics for 8-puzzle.

h1(n) = number of misplaced tiles

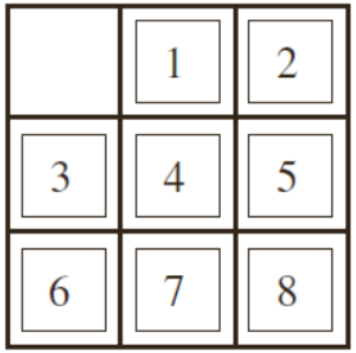
- **h1 is an admissible heuristic** because any tile that is out of place must be moved at least once.

h2(n) = total Manhattan distance (the sum of the distances of the tiles from their goal positions)

- Because tiles cannot move along diagonals, the distance is the sum of the horizontal and vertical distances**.**
- **h2 is also admissible** because all any move can do is move one tile one step closer to the goal.



Start State



Goal State

h1(start) = 8

h2(start) = 3+1+2+2+2+3+3+2 = 18
summation Manhattan Distances of tiles 1 to 8

Neither of these overestimates the true solution cost, which is 26.

The effect of heuristic accuracy on performance

- The quality of a heuristic can be measured by its **effective branching factor b∗**.
- If the total number of nodes generated by A∗ for a particular problem is N and the solution depth is d, then b∗ is the branching factor that a uniform tree of depth d would have to have in order to contain N + 1 nodes.

$$N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d .$$

- If A∗ finds a solution at depth 5 using 52 nodes, then the effective branching factor is 1.92.
- Experimental measurements of b∗ on a small set of problems can provide a good guide to the heuristic's overall usefulness.
- A well-designed heuristic would have a value of b∗ close to 1.

The effect of heuristic accuracy on performance

- To test the heuristic functions h1 and h2, 1200 random problems are generated with solution lengths from 2 to 24 (100 for each even number) and solved with iterative deepening search and with A∗ tree search using both h1 and h2.
- *The results suggest that h2 is better than h1*, and it is far better than using iterative deepening search.

	Search Cost (nodes generated)			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Dominance

- If $h2(n) \geq h1(n)$ for all n (both h1 and h2 are admissible) then **h2 dominates h1** and h2 is better than h1 for search.
- Domination translates directly into efficiency.
- A∗ using h2 will never expand more nodes than A∗ using h1.
- It is generally better to use a ***heuristic function with higher values***, provided it is consistent.

Given any admissible heuristics h_a, h_b,

$$h(n) = \max(h_a(n), h_b(n))$$

is also admissible and h(n) dominates both h_a and h_b

Generating Admissible Heuristics from Relaxed Problems

- h1 (misplaced tiles) and h2 (Manhattan distance) are fairly good heuristics for the 8-puzzle and that h2 is better.
- A ***problem with fewer restrictions on the actions*** is called a **relaxed problem**.
- **Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem.**
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then h1(n) gives the shortest solution.
- If the rules are relaxed so that a tile can move to any adjacent square, then h2(n) gives the shortest solution.

- ***Key point:*** the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem

Summary

- Heuristic functions estimate costs of shortest paths
- Good heuristics can dramatically reduce search cost
- Greedy best-first search expands lowest h
  - incomplete and not always optimal
- A* search expands lowest g + h
  - complete and optimal
  - also optimally efficient
- Admissible heuristics can be derived from exact solution of relaxed problems