Informed (heuristic) Search Strategies

Prof. G. Rama Mohan Babu
Professor & HoD, CSE(AI&ML)
RVR&JCCE, Guntur - 19

Heuristic Search

- Informed search strategy uses *problem-specific knowledge* to find solutions more efficiently than an uninformed search strategy.
- Best-first search is an instance of the general TREE/GRAPH SEARCH 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.
- Most best-first algorithms include as a component of *f* a heuristic function, denoted *h(n)*.
- *h(n) = estimated cost of the cheapest path from the state at node n to a goal state*. if n is a goal node, then *h(n) = 0*.
- Heuristic functions gives additional knowledge of the problem to the search algorithm.

3.5.1. Greedy Best First Search

- *The Greedy best-first search* tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly.
- It evaluates nodes by using just the heuristic function; i.e. $f(n) = h(n)$.
- The route-finding problems in Romania; we use the straight-line distance heuristic h_{SLD} . If the goal is Bucharest, we need to know the straight-line distances to Bucharest.

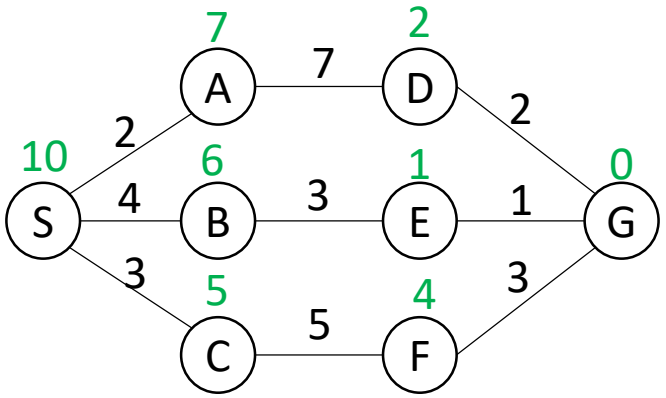
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

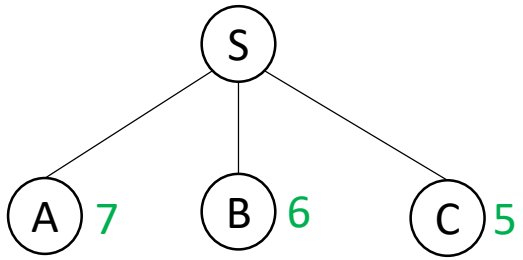
Figure 3.22 Values of h_{SLD}—straight-line distances to Bucharest.

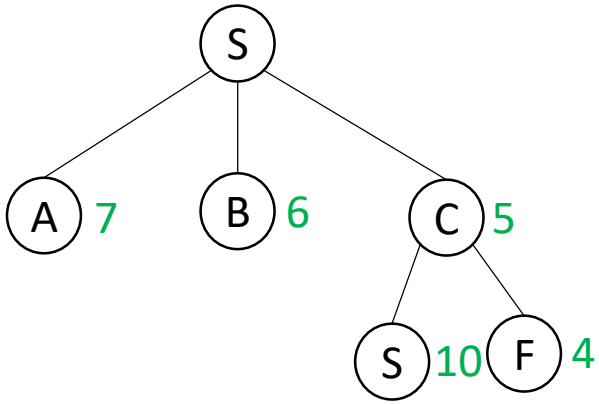
- **Greedy Best-First Algorithm:**
- **Initialize**:
  - Place the initial node in the priority queue.
- **Loop**:
  - Remove the node with the lowest heuristic value from the priority queue.
  - If this node is the goal, return the solution.
  - Otherwise, expand the node by generating its successors and add them to the priority queue.
- **Repeat** until the goal is found or the queue is empty.

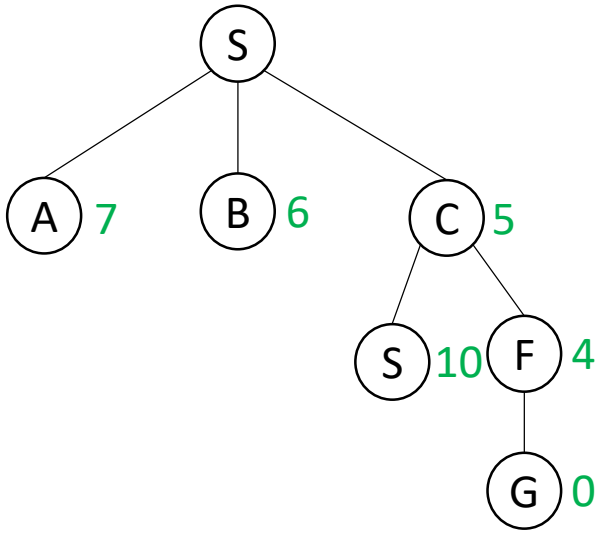
The Greedy Best-First Search algorithm is a useful heuristic search method that can quickly find solutions in many cases, but it may not always find the optimal solution.

Example 1: Simple Path Finding Problem



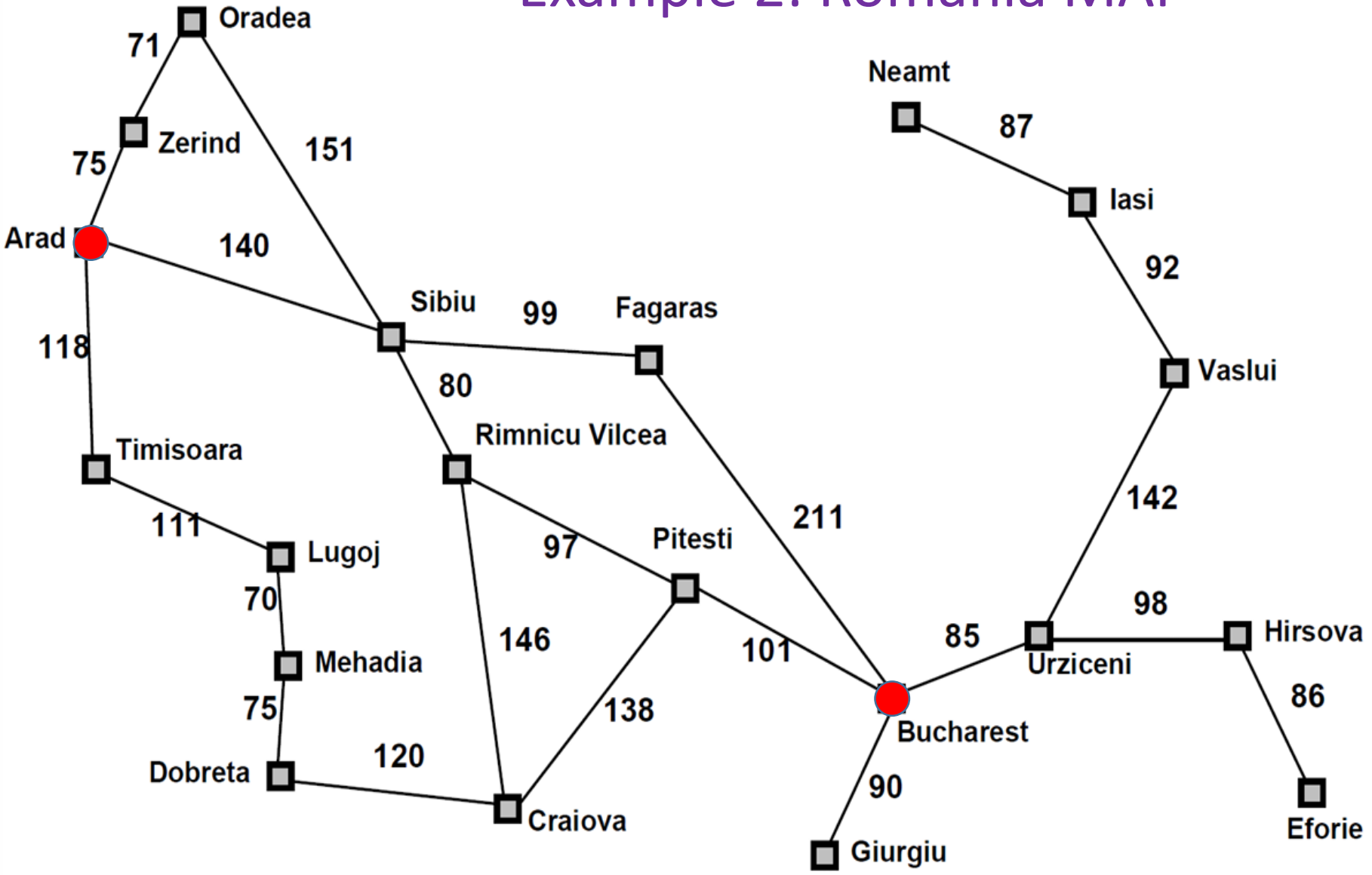






The path is S-C-F-G, and the cost is 11,but it is not optimal.
The optimal path is S-B-E-G and the cost is 8.
Hence the Greedy Best First Search may not give optimal solution.

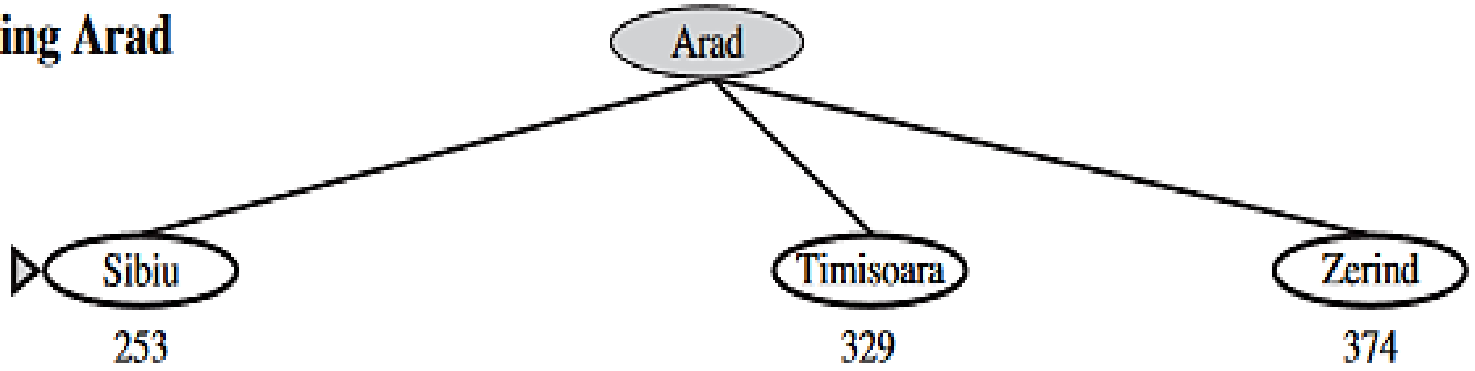
Example 2: Romania MAP



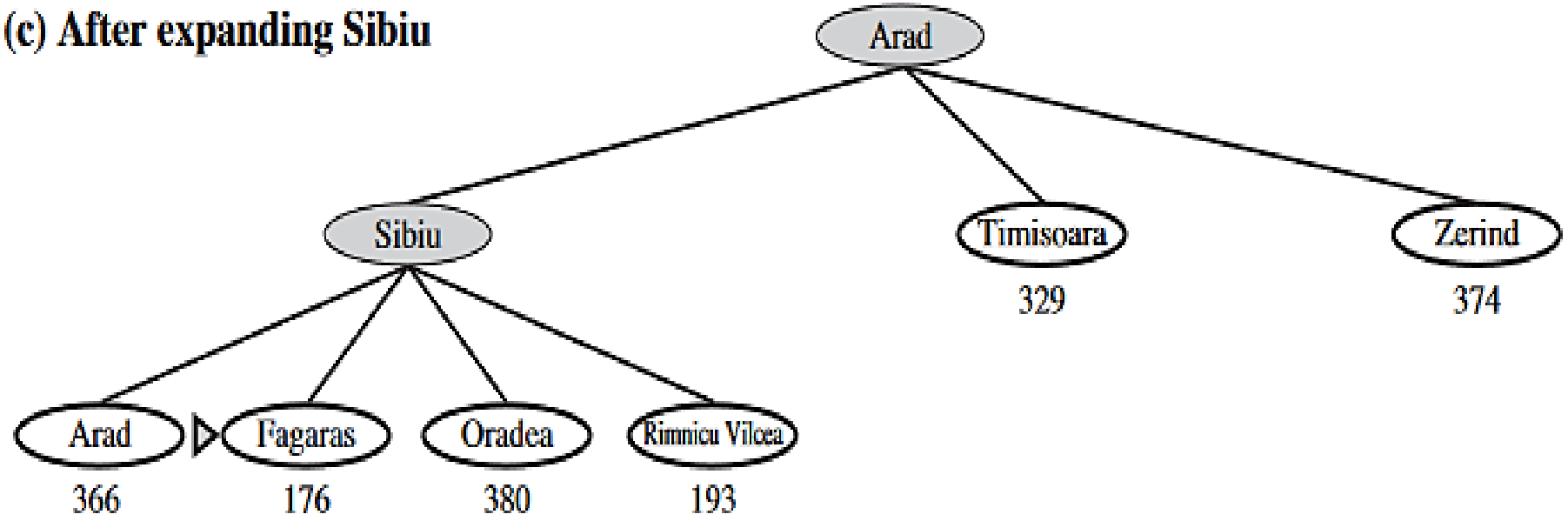
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

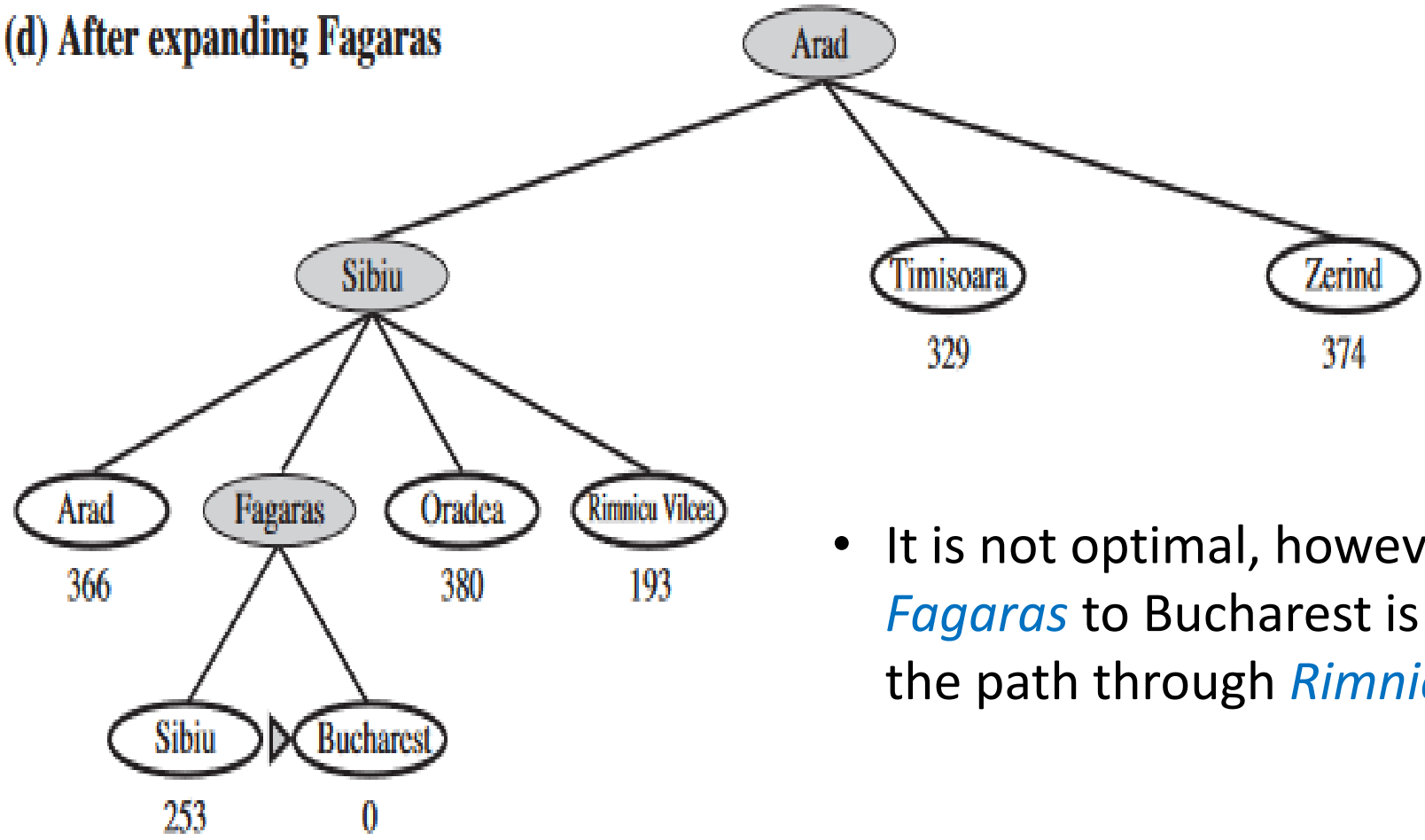
(a) The initial state

(b) After expanding Arad



(c) After expanding Sibiu





- It is not optimal, however, the path via *Sibiu* and *Fagaras* to Bucharest is 32 kilometers longer than the path through *Rimnicu Vilcea* and *Pitesti*.

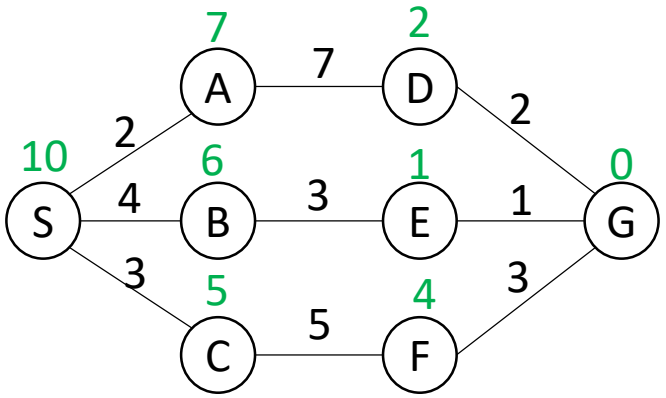
- Completeness: No, Can get stuck in loops;
  - Example: to find the path from *Iasi to Fagaras*: *The algorithm will never find this solution, however, because expanding Neamt puts Iasi back into the frontier, Iasi is closer to Fagaras than Vaslui, and Iasi will be expanded again, leading to an infinite loop.*
- Time & Space Complexity: $O(b^m)$ where m is the maximum depth of search space. With a good heuristic function, however, the complexity can be reduced substantially. The amount of the reduction depends on the particular problem and on the quality of the heuristic.
- Optimality: It is not always gives the optimal solution.

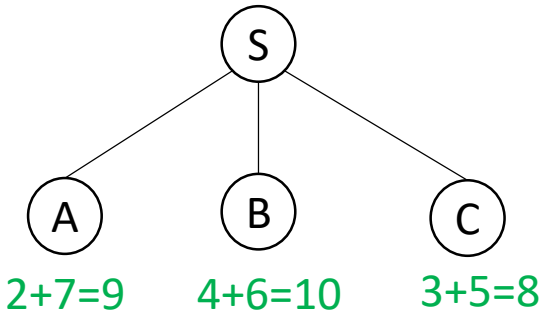
3.5.2. A* search: Minimizing the total estimated solution cost

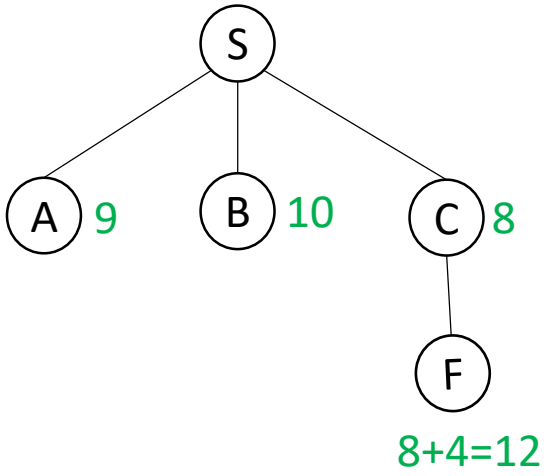
- The most widely known form of best-first search is called A∗ search.
- *Heuristic Function ($h(n)$):* Estimates the cost from node n to the goal. This function uses problem-specific knowledge to guide the search.
- *Cost Function ($g(n)$):* Represents the actual cost from the start node to node n.
- *Evaluation Function ($f(n)=g(n)+h(n)$):* Combines both the actual cost and the heuristic cost to estimate the total cost of the cheapest solution through node n.
- The algorithm is identical to UNIFORM-COST-SEARCH except that A∗ uses *g+h* instead of *g*.

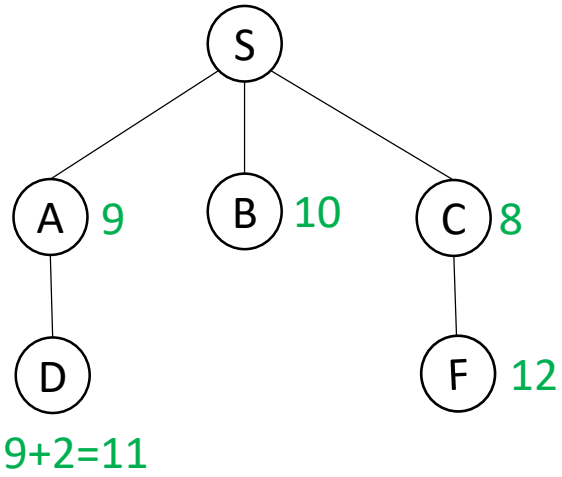
- **Initialize**:
  - Place the initial node in the priority queue with f(start)=g(start)+h(start).
  - Set g(start) = 0 since the cost to reach the start node is zero.
- **Loop**:
  - Remove the node with the lowest f(n) value from the priority queue (this is the current node).
  - If this node is the goal, return the solution path.
  - Otherwise, for each neighbor of the current node:
    - Calculate the tentative g score (the cost to reach the neighbor from the start node through the current node).
    - If this tentative g score is better than the previously known g score for the neighbor:
      - Update the neighbor's g score.
      - Calculate f(neighbor)=g(neighbor)+h(neighbor).
      - Add the neighbor to the priority queue if not already in it, or update its position in the queue.

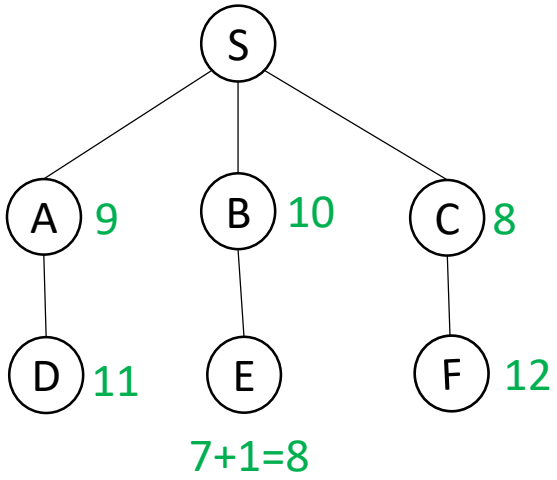
Example 1: Simple Path Finding Problem

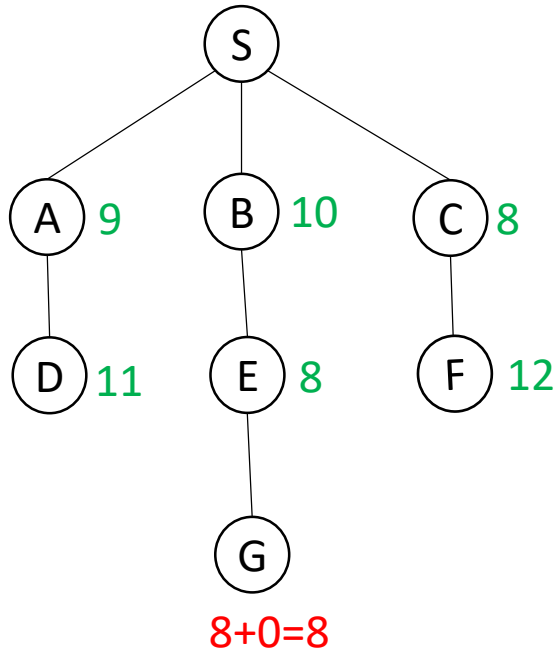






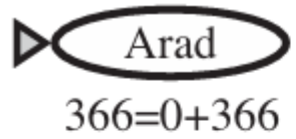




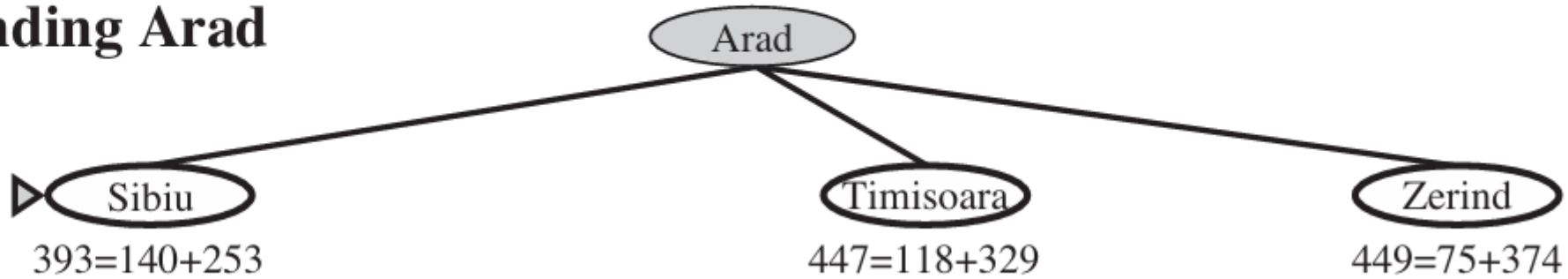


The Optimal Path is S-B-E-G and the cost is 8

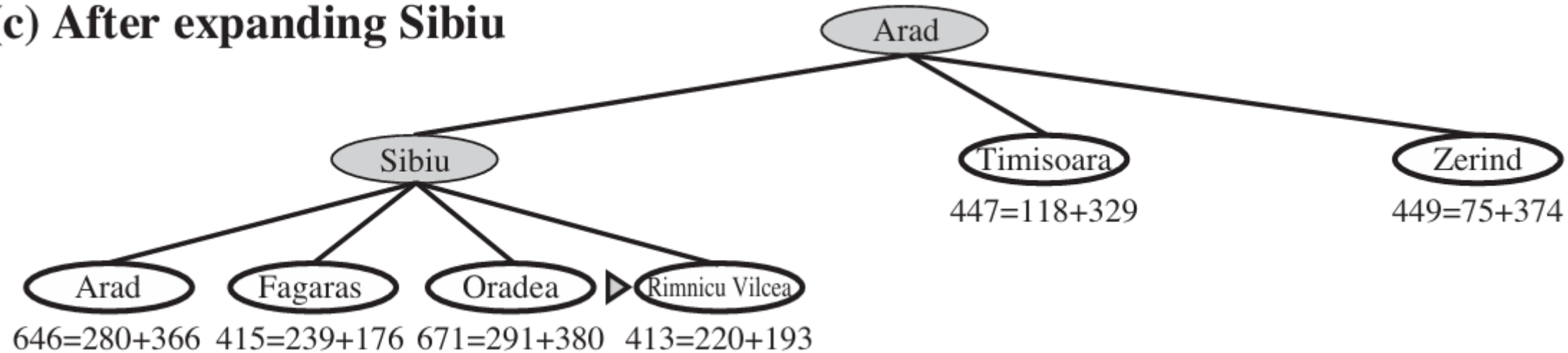
(a) The initial state



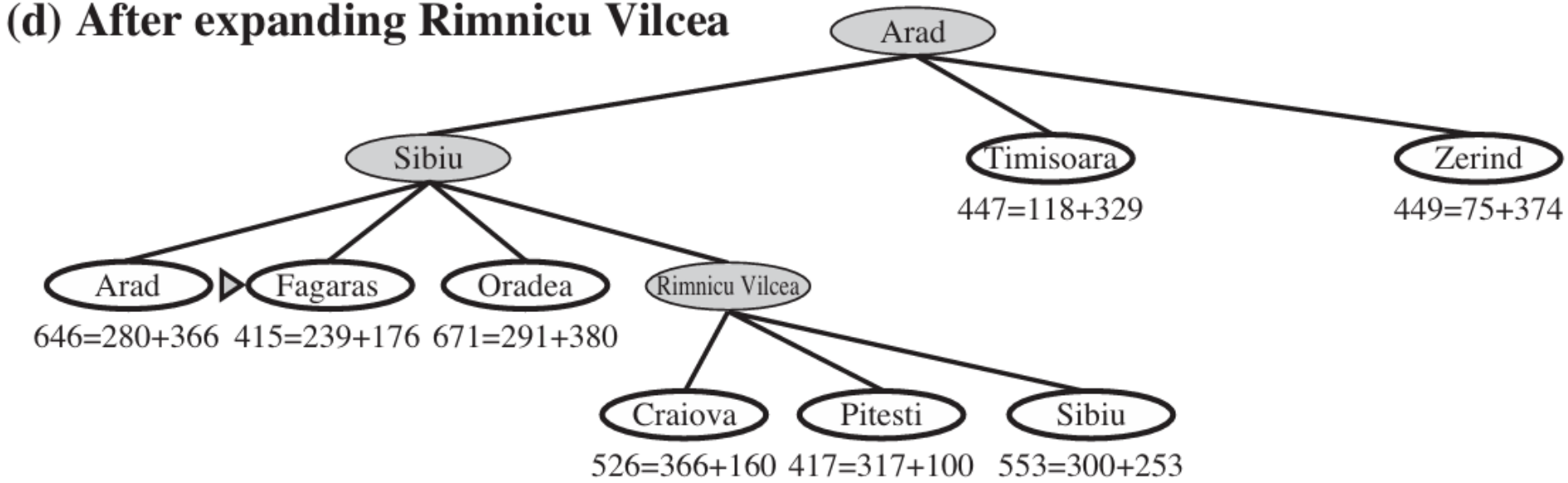
(b) After expanding Arad



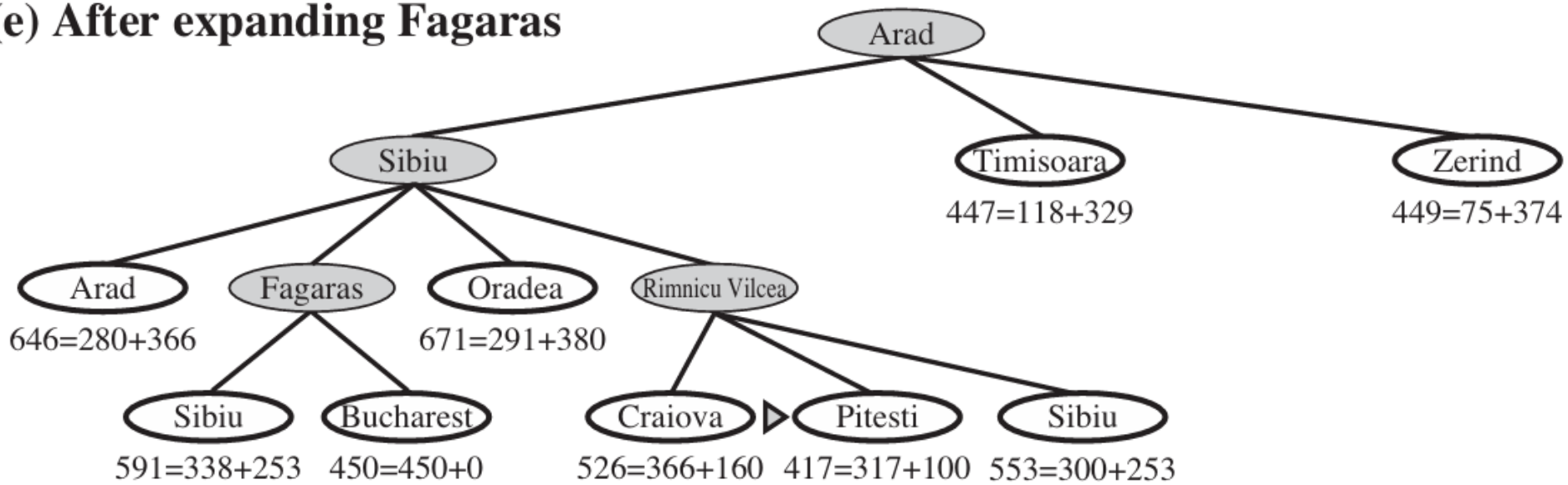
(c) After expanding Sibiu



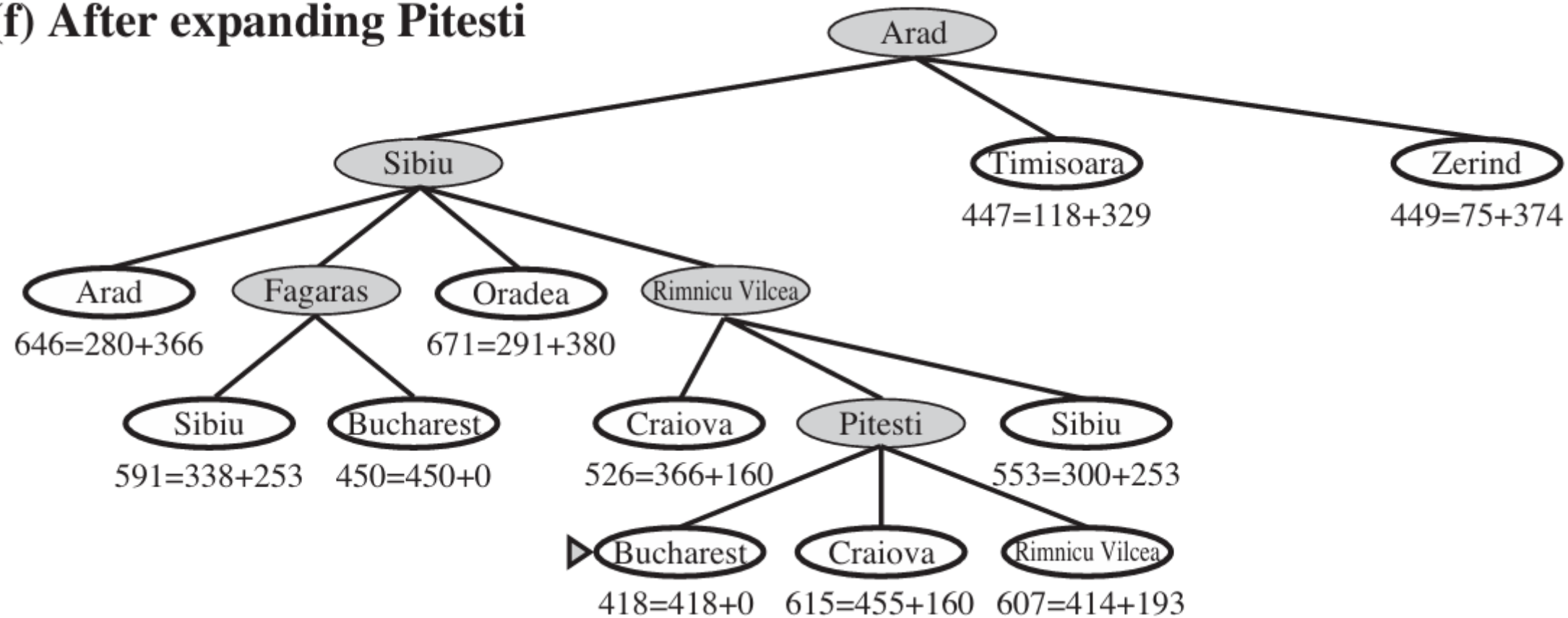
(d) After expanding Rimnicu Vilcea



(e) After expanding Fagaras



(f) After expanding Pitesti



Conditions for optimality of A*: Admissibility and consistency

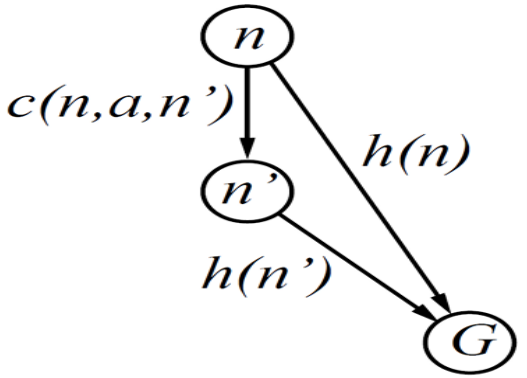
1. An *admissible heuristic* is one that never overestimates the cost to reach the goal (always less than or equal to the actual cost).

   Given that *g(n)* is the actual cost to reach node *n* along the current path, and *f(n)=g(n)+h(n)*, this means *f(n)* will not overestimate the true cost of reaching the goal through node *n*.

   The straight-line distance is admissible because the shortest path between any two points is a straight line so it will never over estimate.

2. A heuristic h(n) is *consistent (or monotonicity)* 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'*.

$$h(n) \leq C(n, a, n') + h(n')$$



This is a form of the general triangle inequality, which stipulates that each side of a triangle cannot be longer than the sum of the other two sides.

Optimality of A*

- The tree-search version of A* is optimal if *h(n)* is admissible, while the graph-search version is optimal if *h(n)* is *consistent*.
- If *h(n)* is consistent, then the values of *f(n)* along any path are nondecreasing.
- Whenever A* selects a node *n* for expansion, the optimal path to that node has been found.
- Timisoara is pruned; because h_{SLD} is *admissible*, the algorithm can safely ignore this subtree while still guaranteeing optimality.
- A* is optimally efficient for any given consistent heuristic. That is, no other optimal algorithm is guaranteed to expand fewer nodes than A*.
- A* usually runs out of space long before it runs out of time, making it impractical for many large-scale problems.

3.5.3. Memory-bounded heuristic search

- To reduce memory requirements for A*, the idea of iterative deepening can be adapted to heuristic search, resulting in the iterative-deepening A* (IDA*) algorithm.
- IDA* is a variant of heuristic search that uses the *f-cost(g+h)* as the cutoff, rather than depth.
- At each iteration, the cutoff is set to the smallest *f-cost* that exceeded the previous iteration's cutoff.
- IDA* is practical for problems with unit step costs and avoids the overhead of maintaining a sorted queue of nodes.
- However, it struggles with real-valued costs, similar to iterative uniform-cost search.

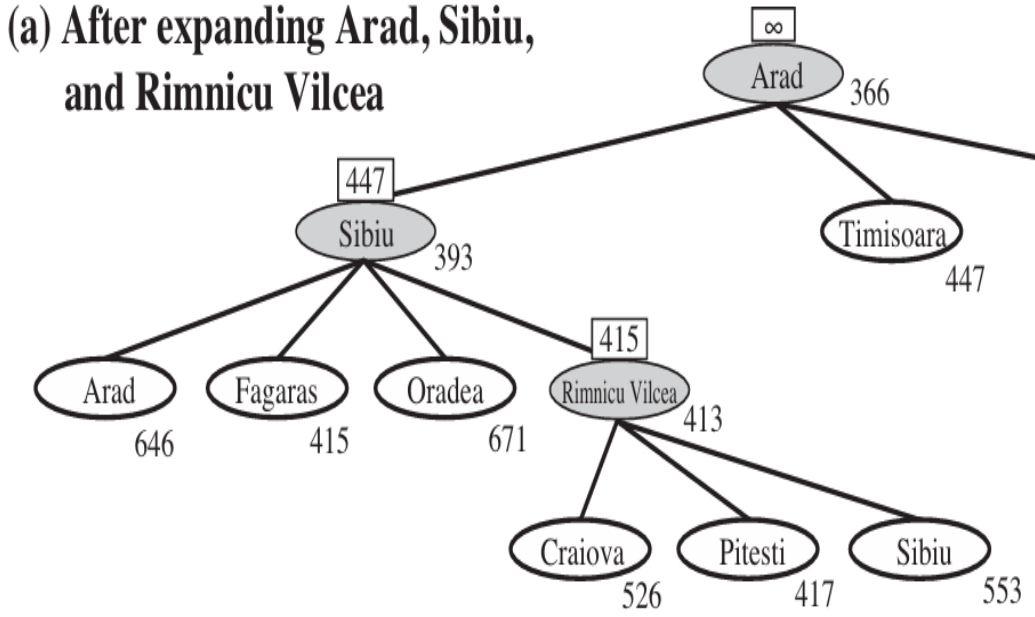
Recursive best-first search (RBFS)

- 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.
- Initialization:
  - RBFS starts from the initial node and recursively explores its successors.
- Recursive Expansion:
  - The algorithm recursively expands the most promising node (the one with the lowest f-value).
  - It keeps track of the best alternative path (second-best f-value) in case the current path leads to a dead end or a less optimal solution.
- Backtracking:
  - If a node's cost exceeds the current best alternative path, RBFS backtracks, updates the node's f-value, and explores the next most promising node.
  - This process continues until the goal is found or all nodes are explored.

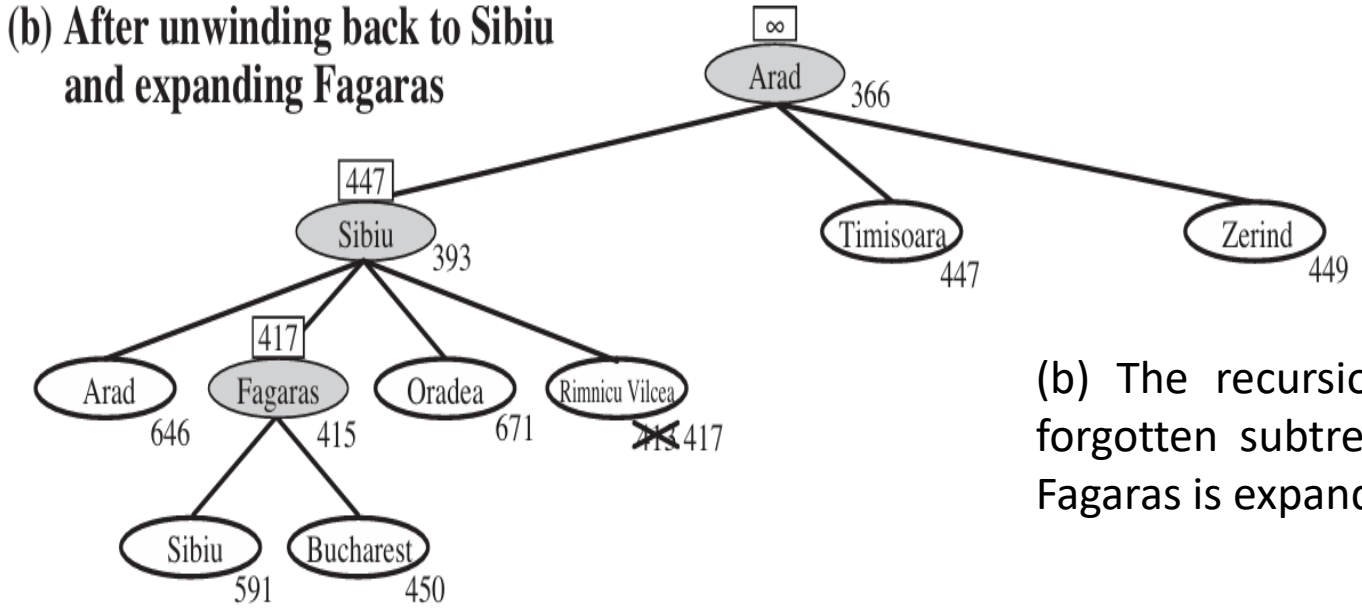
```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution, or failure
    return RBFS(problem, MAKE-NODE(problem.INITIAL-STATE), ∞)

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
```

Figure 3.26 The algorithm for recursive best-first search.

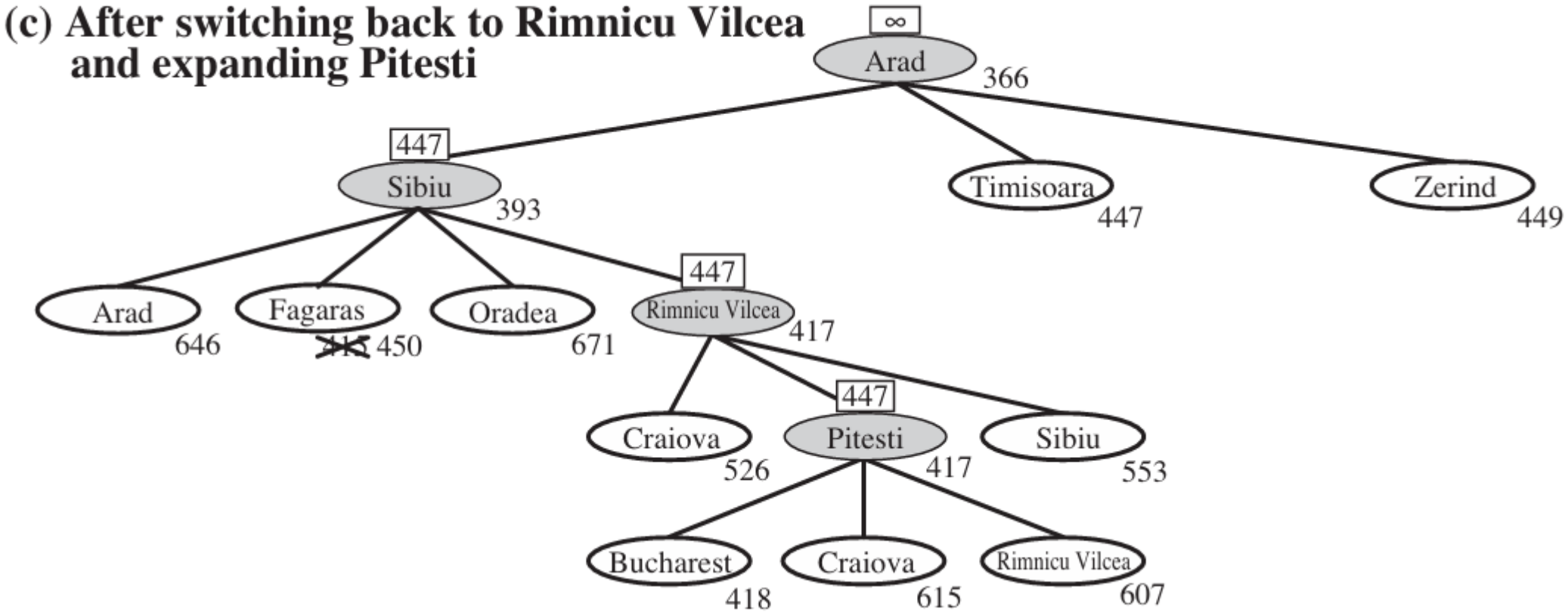
Stages in an RBFS search for the shortest route to Bucharest. The f-limit value for each recursive call is shown on top of each current node, and every node is labeled with its f-cost.



(a) The path via Rimnicu Vilcea is followed until the current best leaf(Pitesti) as a value that is worse than the best alternative path(Fagaras).



(b) The recursion unwinds and the best leaf value of the forgotten subtree(417) is backed up to Rimnicu Vilcea; then Fagaras is expanded, revealing a best leaf value of 450.



(c) The recursion unwinds and the best leaf value of the forgotten subtree (450) is backed up to Fagaras; then Rimnicu Vilcea is expanded. This time, because the best alternative path (through Timisoara) costs at least 447, the expansion continues to Bucharest.

3.5.4. Learning to Search Better

- Each state in a *metalevel state space* captures the internal (computational) state of a program that is searching in an *object-level state space* such as Romania.
- For example, the internal state of the A* algorithm consists of the current search tree. Each action in the *metalevel state space* is a computation step that alters the internal state; for example, each computation step in A* expands a leaf node and adds its successors to the tree.
- which shows a sequence of larger and larger search trees, can be seen as depicting a path in the *metalevel state space* where each state on the path is an *object-level search tree*.
- For harder problems, there will be many such missteps, and a *metalevel learning algorithm* can learn from these experiences to avoid exploring unpromising subtrees.