Introduction to Artificial Intelligence

Chapter 2: Solving Problems by Searching (3) Informed (Heuristic) Search

cuu duong than cong . com

Nguyễn Hải Minh, Ph.D

nhminh@fit.hcmus.edu.vn

Outline

1. Informed Search Strategies
2. Best-first search
3. Greedy best-first search
4. A* search
5. Memory-bounded heuristic search

Informed (Heuristic) Search Strategies

- Informed Search Strategies:
  - Use the information *beyond the definition of the problem* itself to improve **efficiency**
  - Can provide *significant speed-up* in practice

What is a heuristic?

- How to include additional information to make a search more efficient?
  - Using a "Heuristic"
- What is a heuristic?
  - "Heuristic" means "serving to aid discovery"
  - A *rule of thumb* to find answers
  - A *shortcuts* to make a decision
  - It helps estimate the quality or potential of partial solutions
  - It helps when no algorithmic solution is available

Example of daily life heuristics

- Judgement heuristics:
  - Buying products: The more expensive, the better
  - Writing reports: The longer, the better
- Availability heuristics
- Anchoring
  - "Limit 12 per customer"
  - 2 watches: $5000 and $500

...

QUIZ

Think of a heuristic that you use everyday to judge something or to make a decision. Briefly explain your answer. In your opinion, is it a good heuristic? (can it help you to make the correct decision?)

*Do not use the ones that have been discussed on the slide

**Go to Moodles from 18:00 to 20:00 today (May 23th) to submit your answer (bonus credit)

Informed search strategies

- Algorithms:
  - Best-first search
  - Greedy best-first search
  - A*
  - RBFS
  - SMA*
  - Memory-bounded heuristic search
  - Techniques for generating heuristics

Informed search strategies

- Best-first search
  - An instance of the general Tree-search or Graph-search algorithm
  - A node is selected for expansion based on an evaluation function, $f(n)$
    - Node with lowest $f(n)$ is expanded first
    - f is only an **ESTIMATE** of node quality
  - The choice of f determines the search strategy
  - More accurate name: "seemingly best-first search"

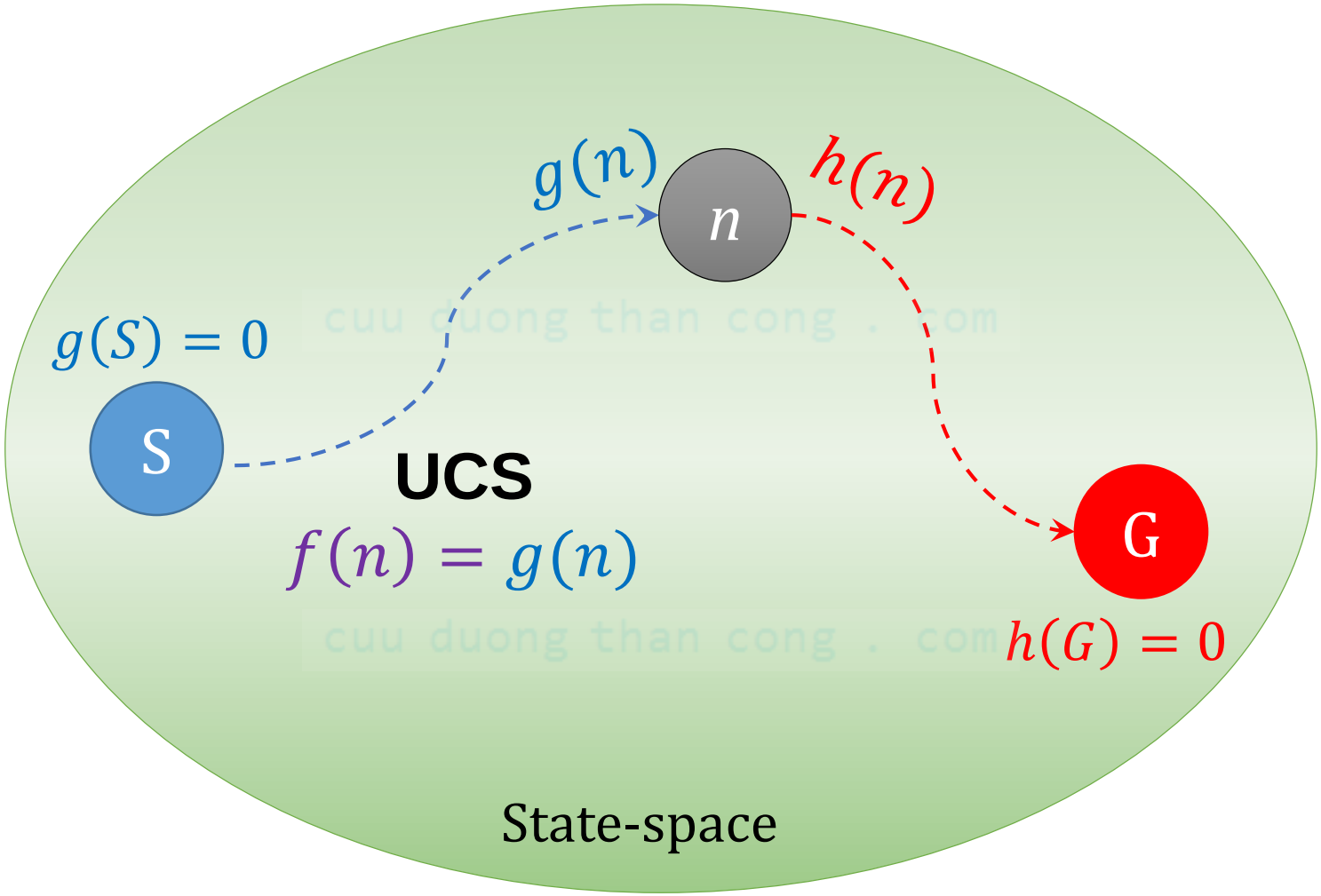
Informed search strategies

- Best-first search algorithms
  - Uniform cost search: $f(n) = g(n)$=path cost to n
  - Greedy best-first search
  - A* search
  - …
- Most best-first search algorithms include a heuristic function

Informed search strategies

- Heuristic function $h(n)$
  - Estimate of cheapest cost from n to goal
  - When n is goal: $h(n) = 0$
  - Example:
    - Straight line distance from n to Bucharest
  - Provide problem-specific knowledge to the search algorithm

Cost function vs Heuristic function



Greedy best-first search (GBS)

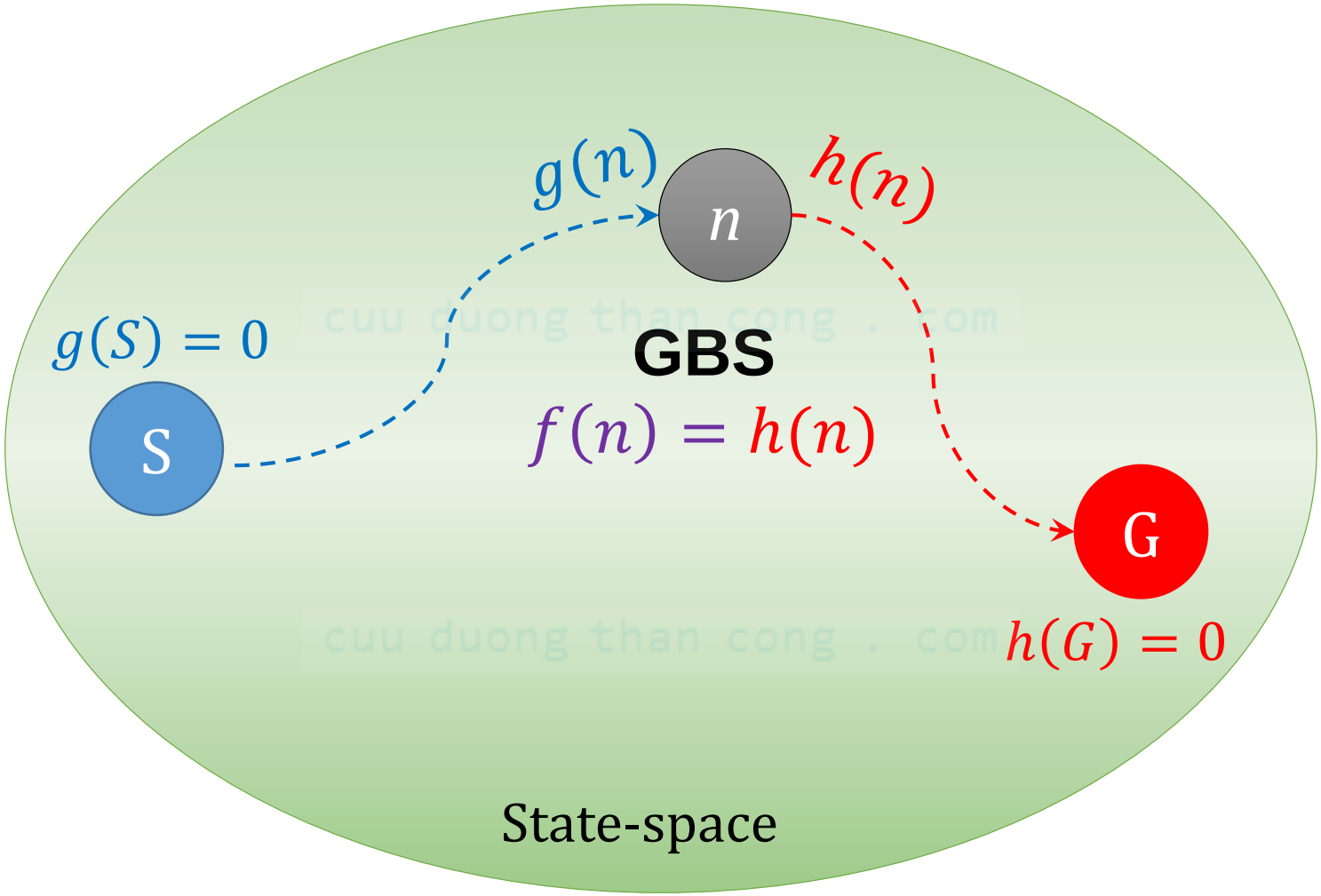
Greedy best-first search

- Idea:
  - GBS expands the node that appears to be closest to goal

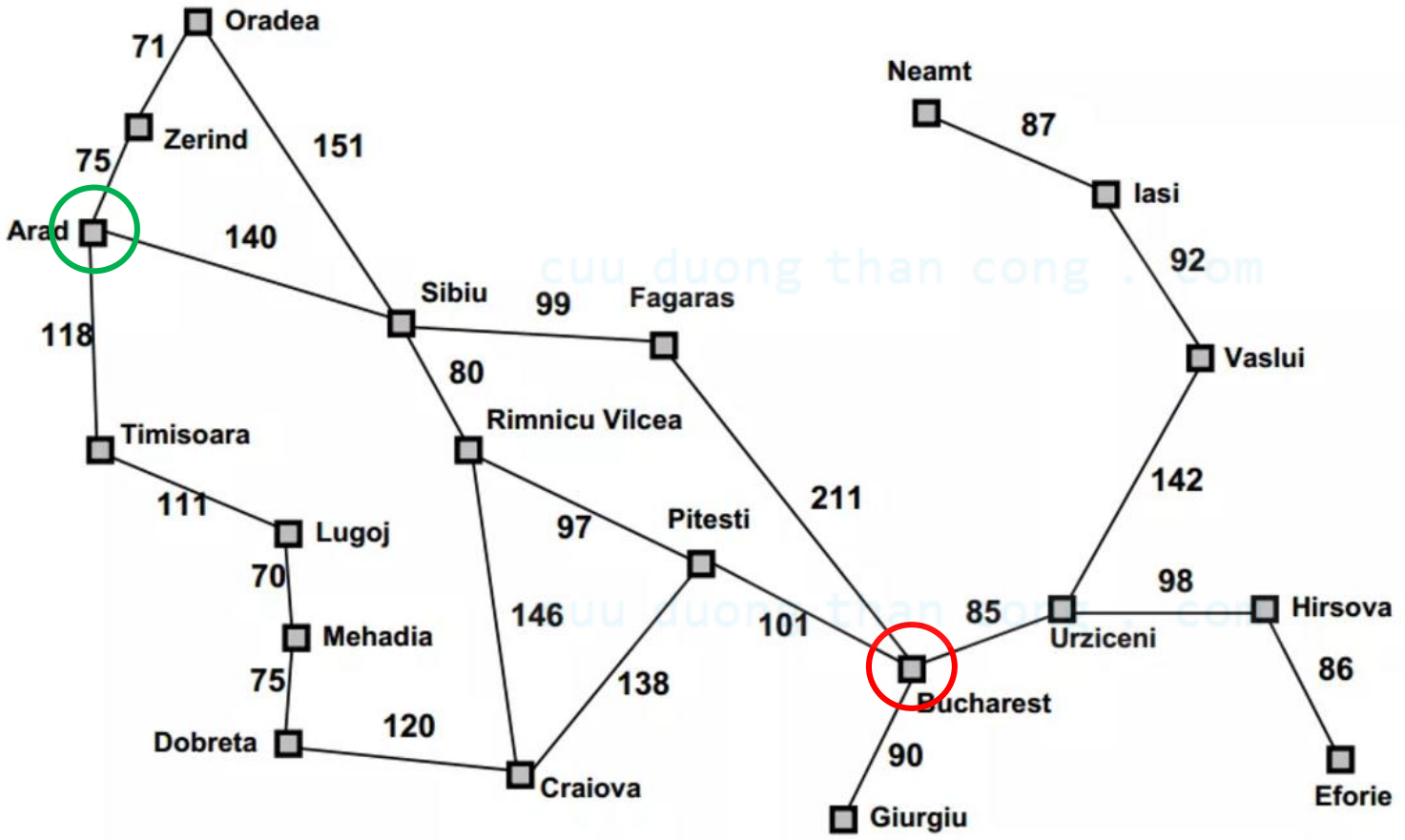
- Evaluation function

  $f(n) = h(n)$ = estimate of cost from n to *goal*
  - e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest

Greedy best-first search



Romania with step costs in km



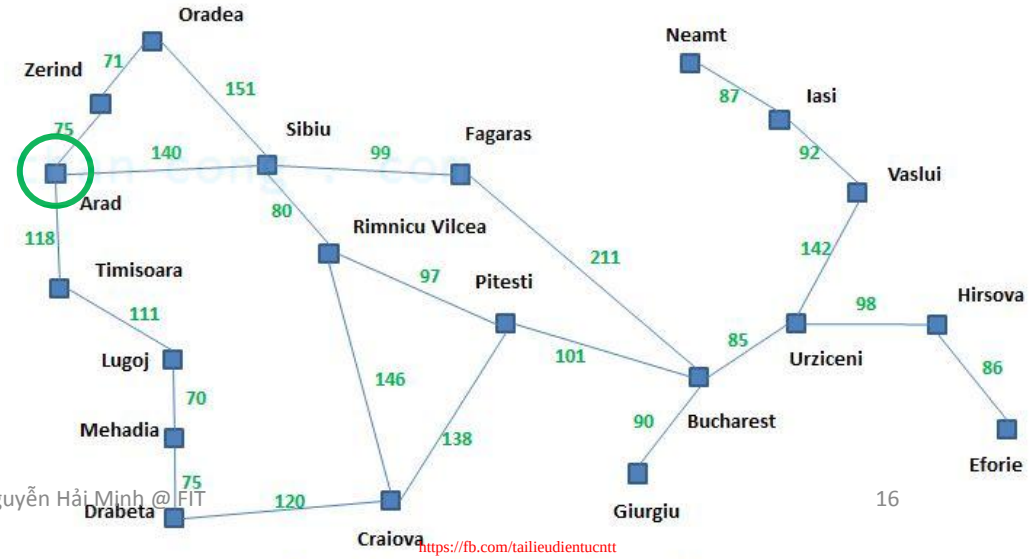
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search example

(a) The initial state

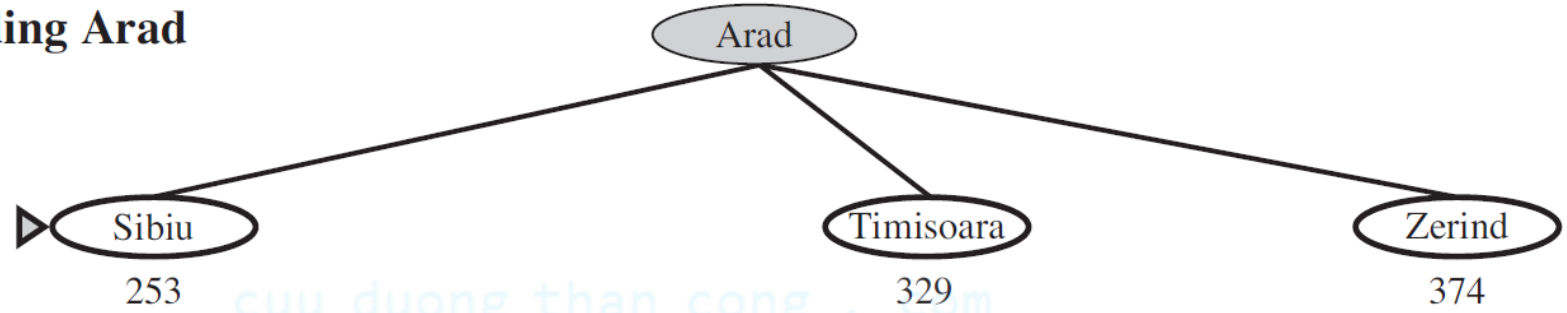
$h_{SLD}(\text{Arad})$

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

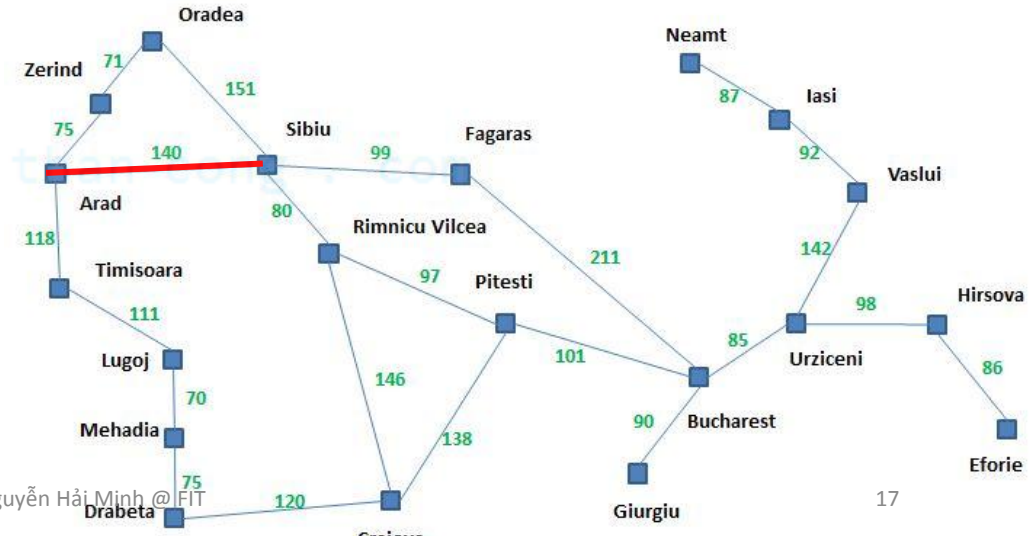


Greedy best-first search example

(b) After expanding Arad

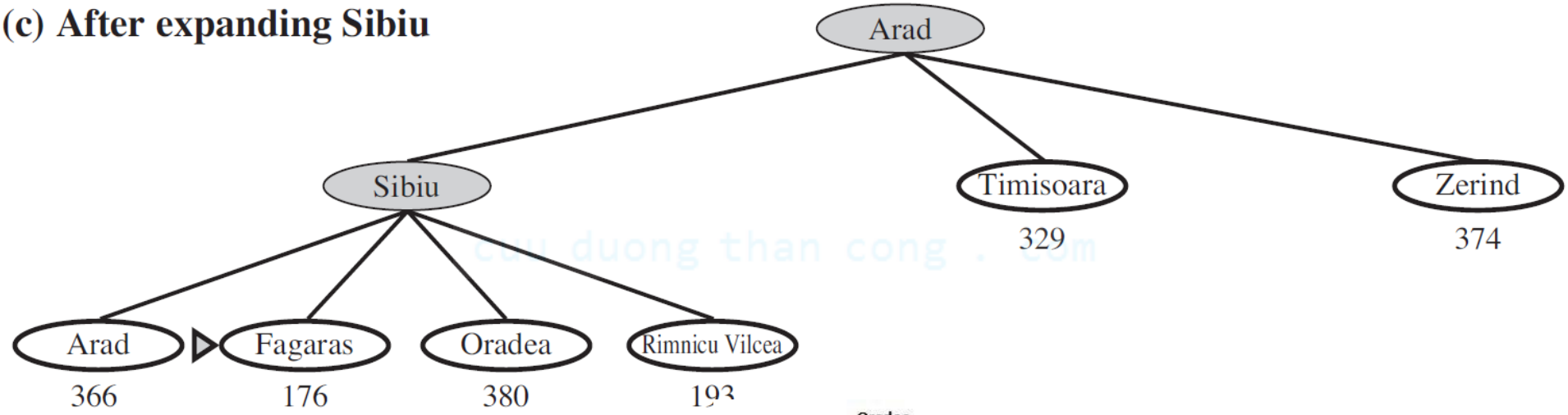


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

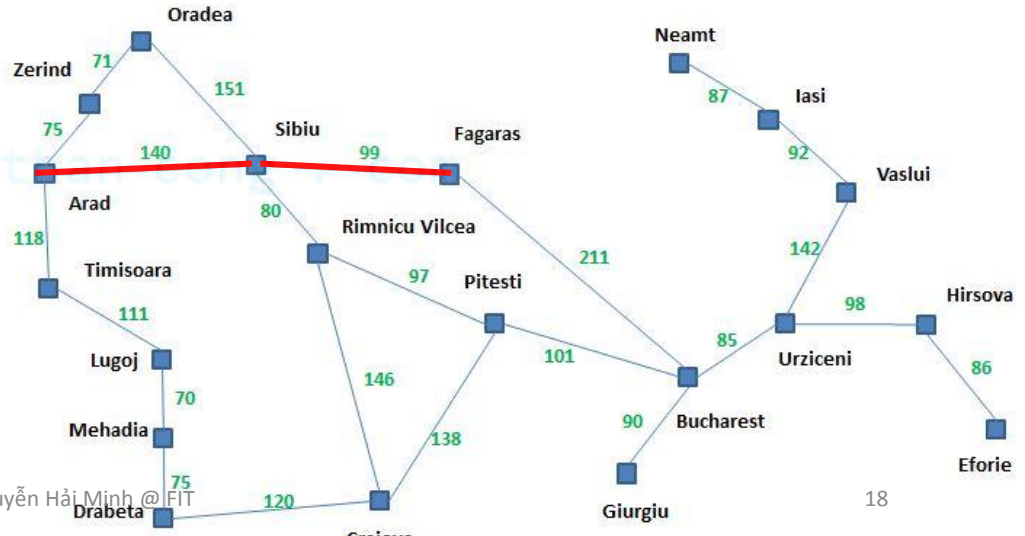


Greedy best-first search example

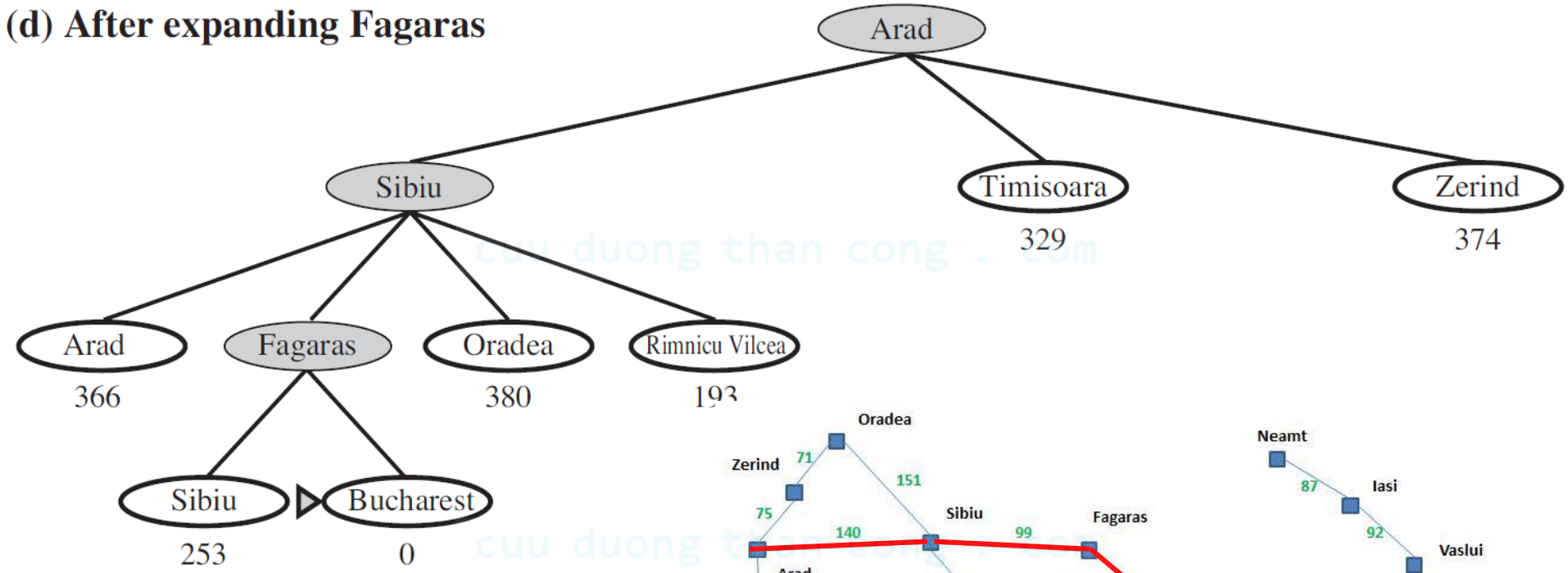
(c) After expanding Sibiu



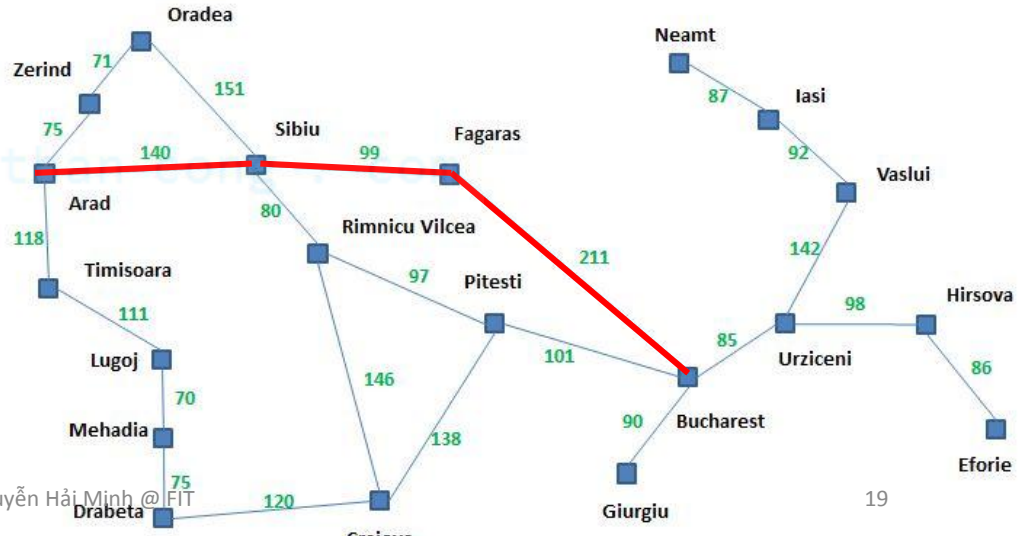
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374



Greedy best-first search example



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



Properties of greedy best-first search

- Completeness
  - No – can get stuck in loops
  - e.g., Iasi → Neamt → Iasi → Neamt →
- Time complexity
  - $O(b^m)$

  →A good heuristic can give dramatic improvement
- Space complexity
  - $O(b^m)$ -- keeps all nodes in memory
- Optimality
  - No

A* search

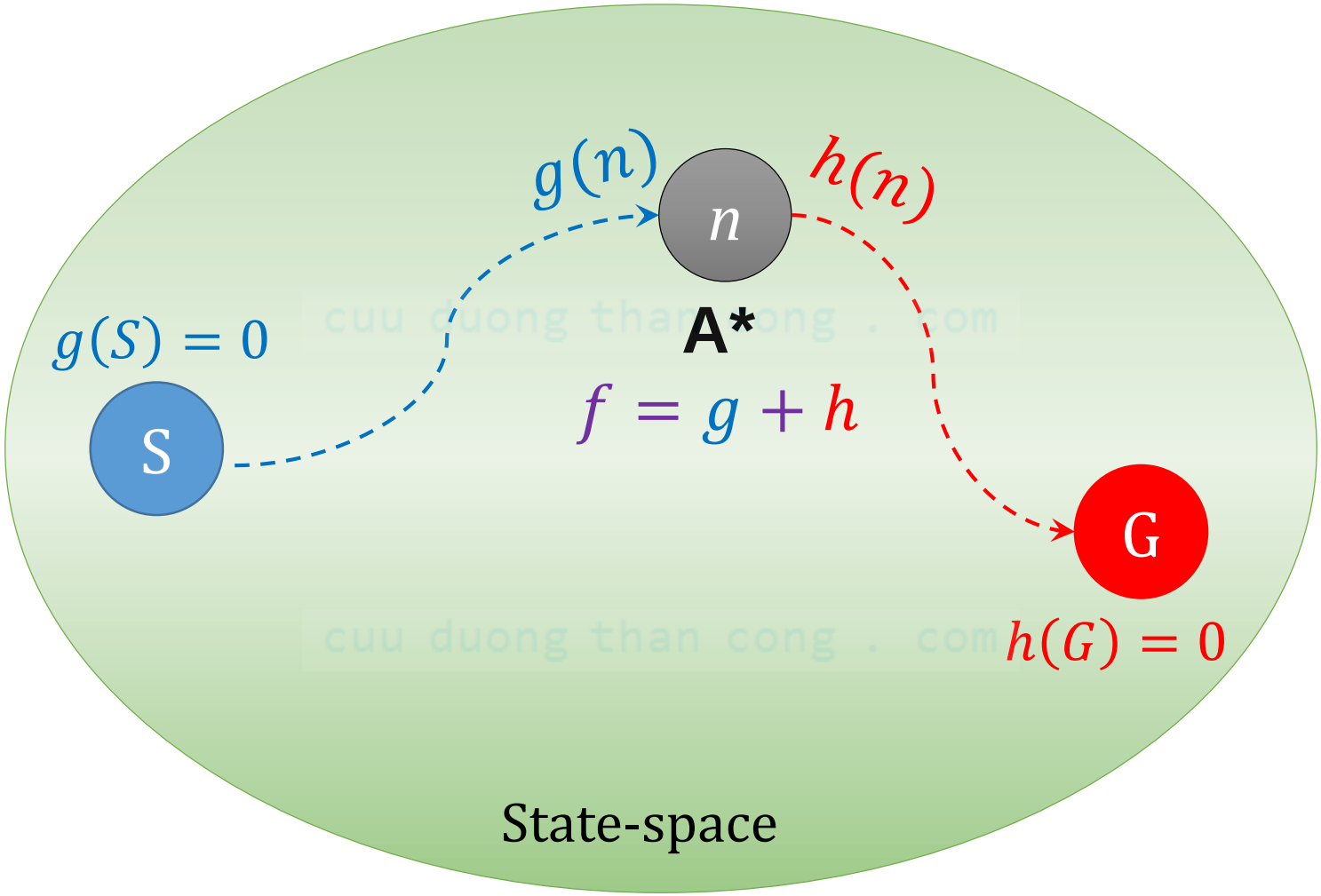
A* search

- Idea:
  - Use heuristic to guide search
  - Avoid expanding paths that are already expensive
  - Ensure to compute a path with minimum cost
- Evaluation function

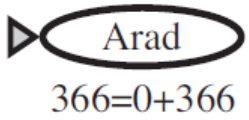
$$f(n) = g(n) + h(n)$$

  - $g(n)$ = cost from the starting node to reach n
  - $h(n)$ = estimated cost of the cheapest path from n to goal
  - $f(n)$ = estimated total cost of path through n to goal

A* search



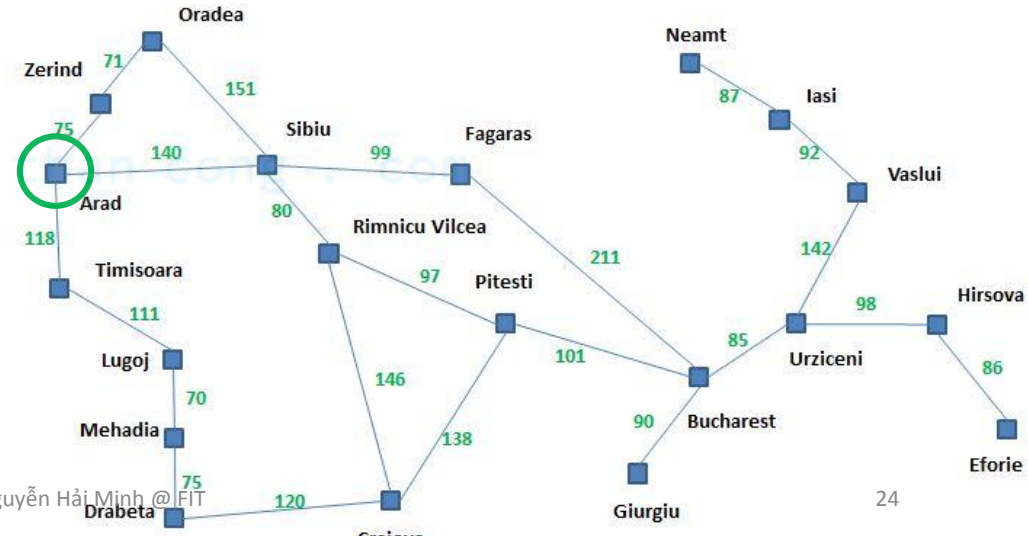
(a) The initial state



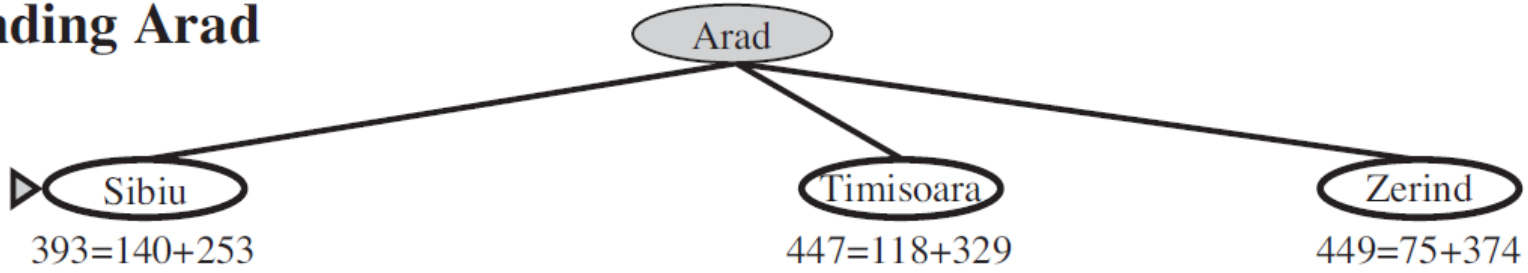
$f = g + n$

A* search example

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

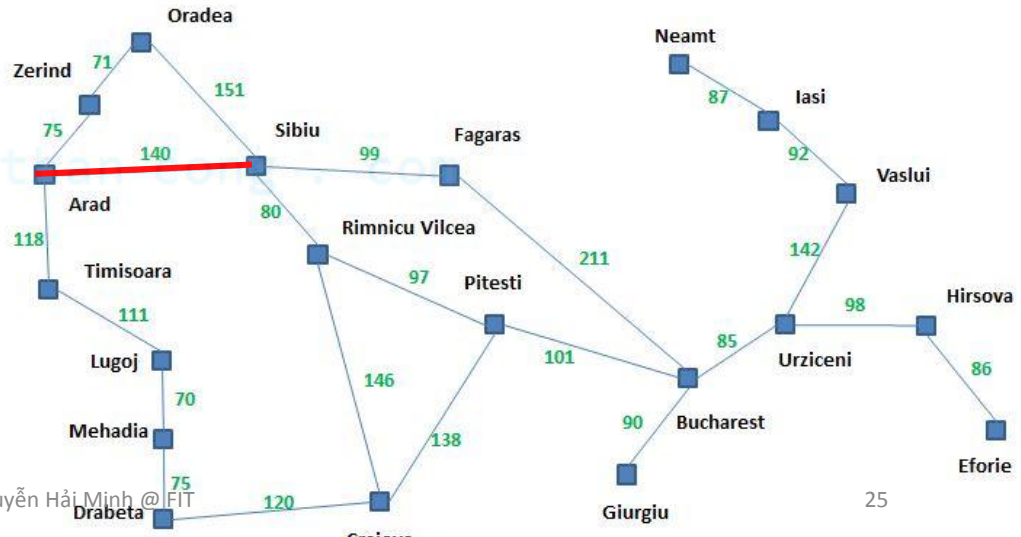


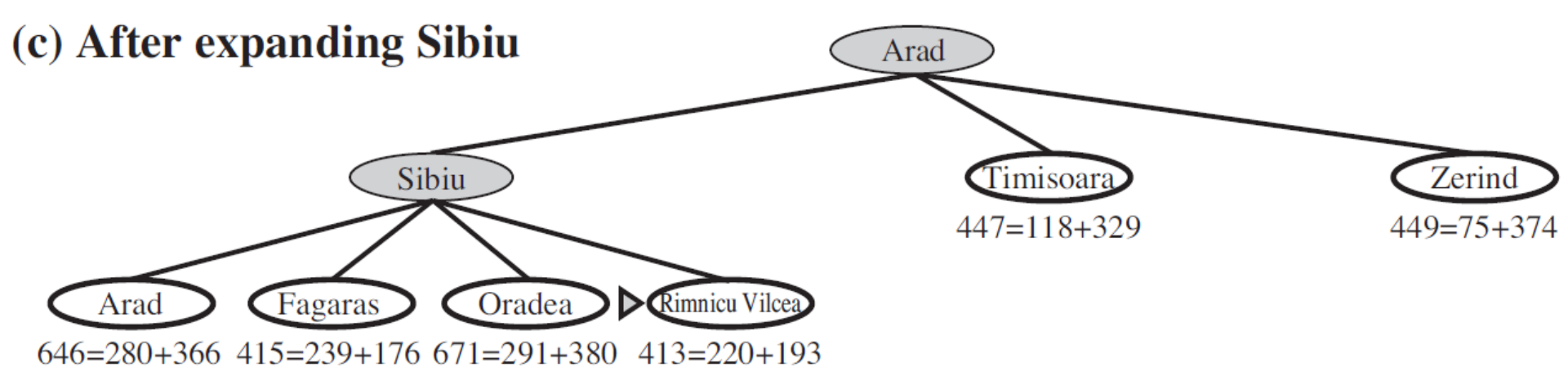
(b) After expanding Arad



A* search example

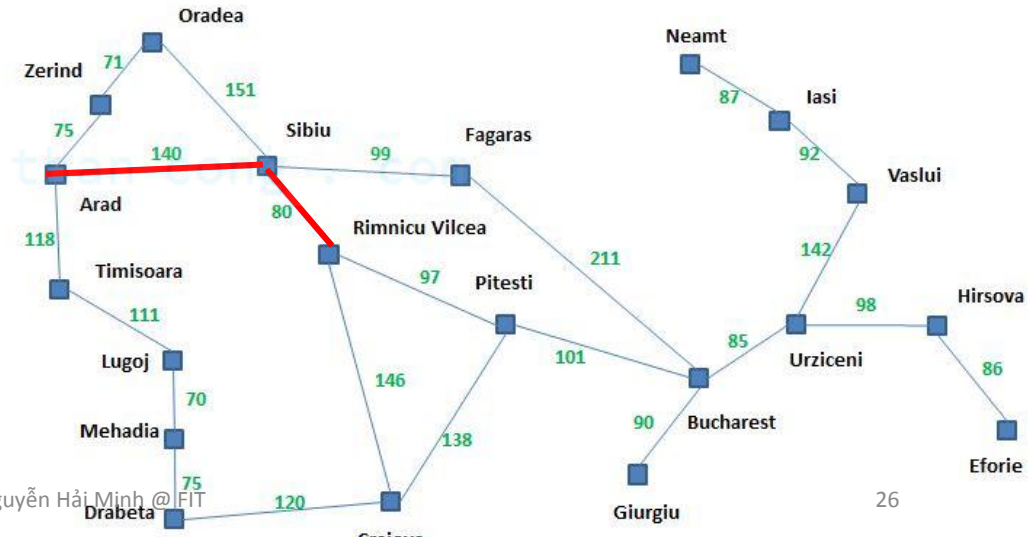
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

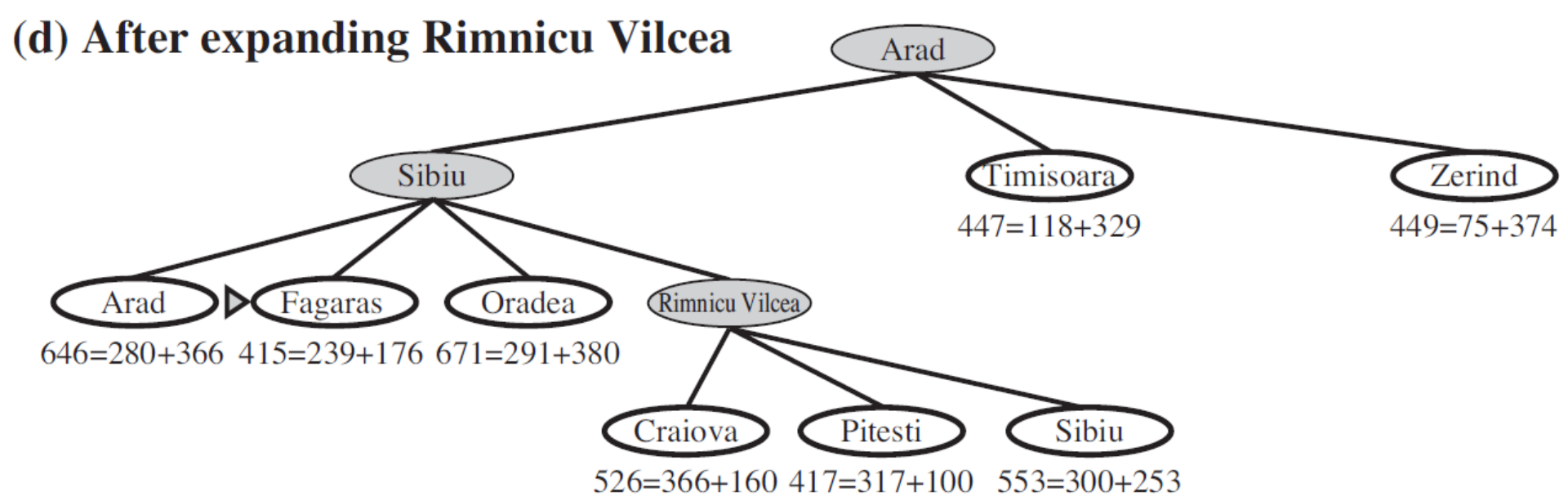




A* search example

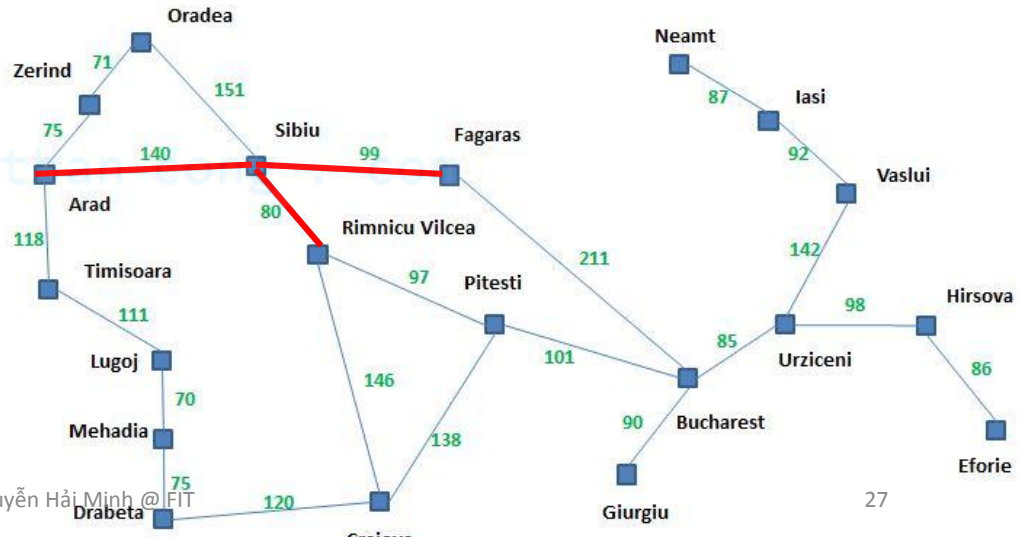
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

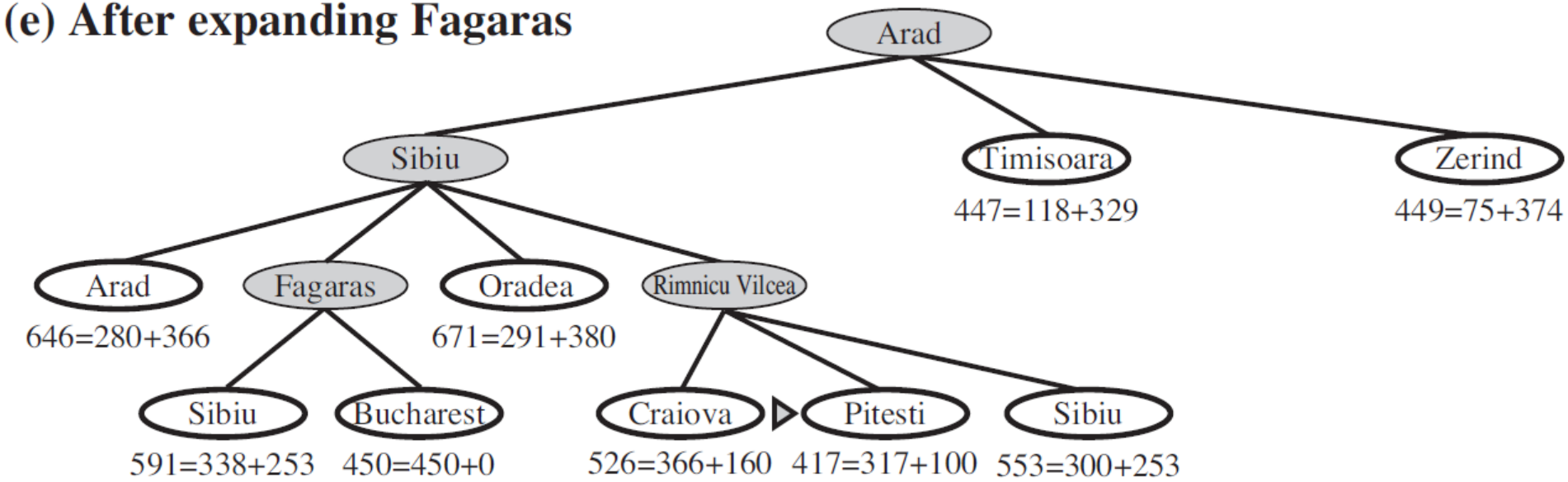




A* search example

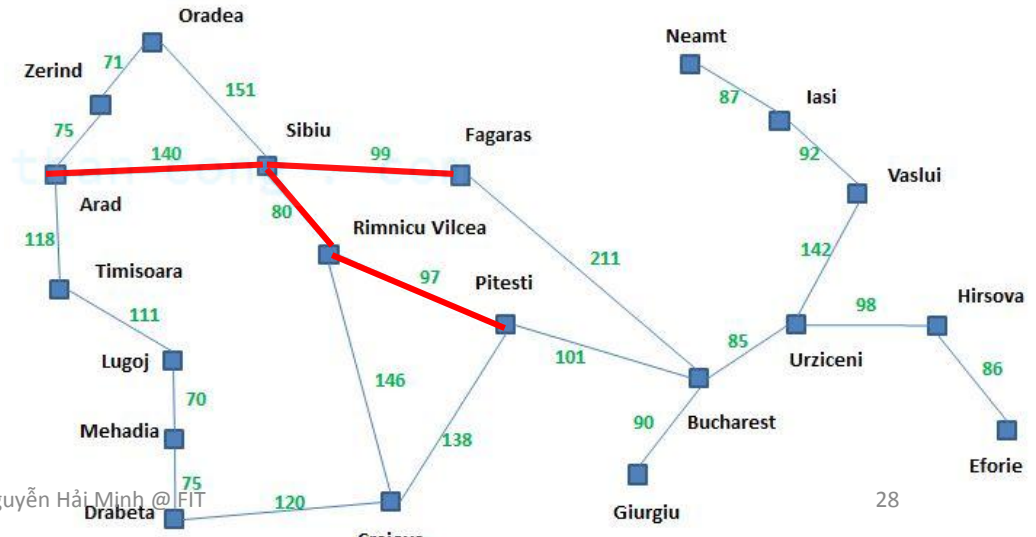
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

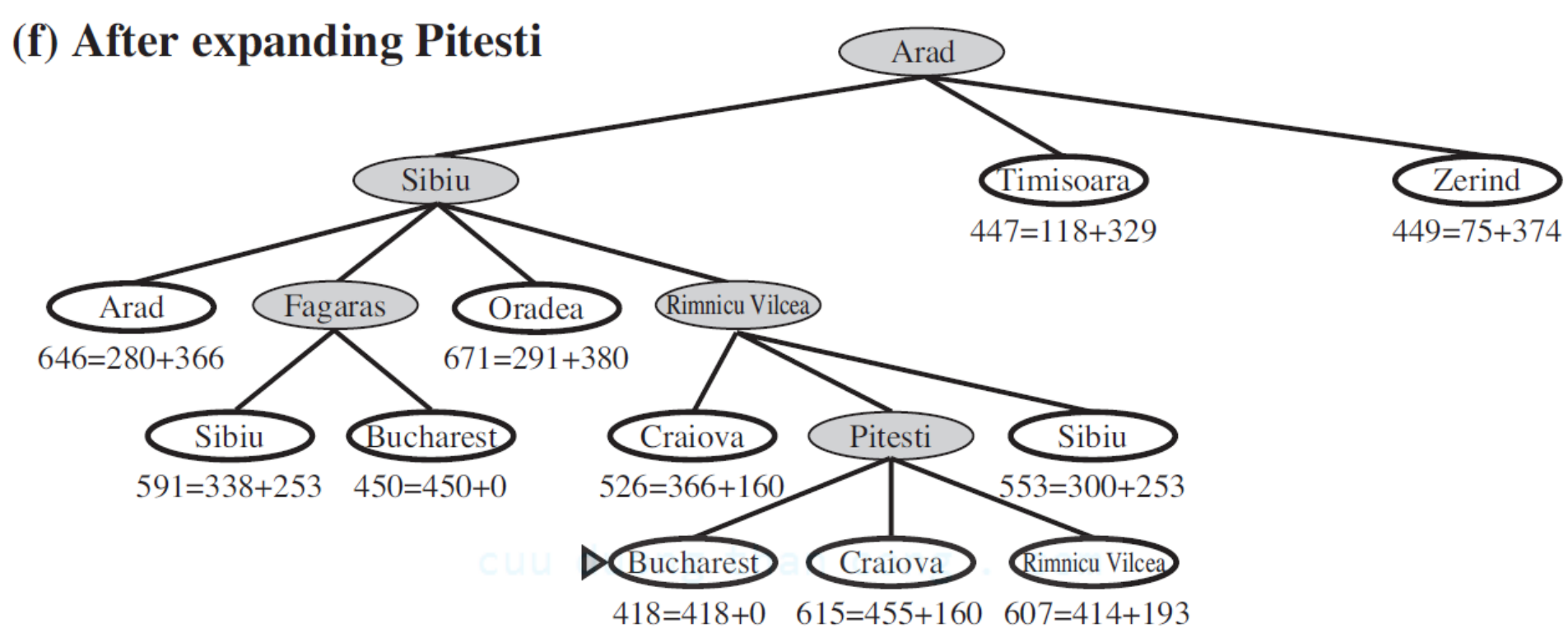




A* search example

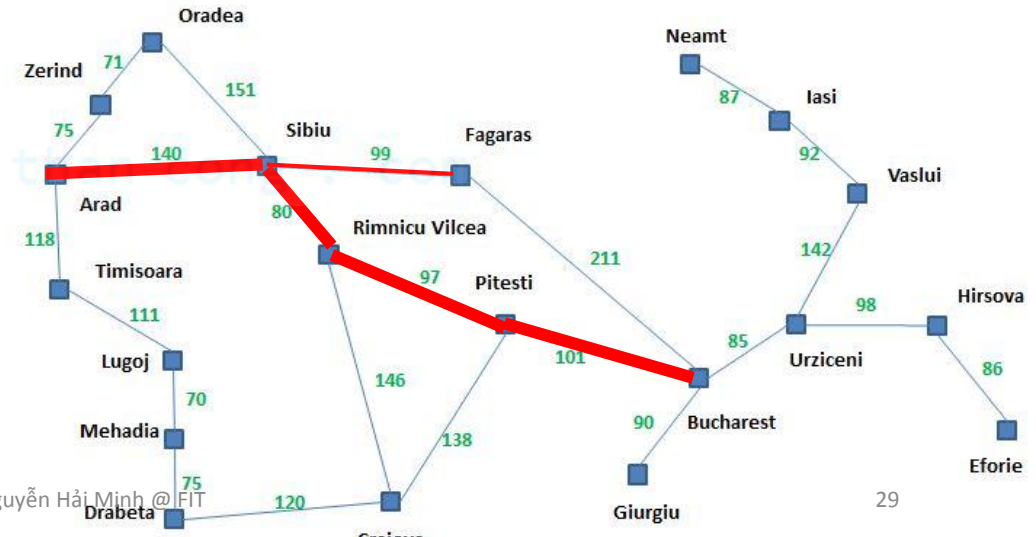
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374





A* search example

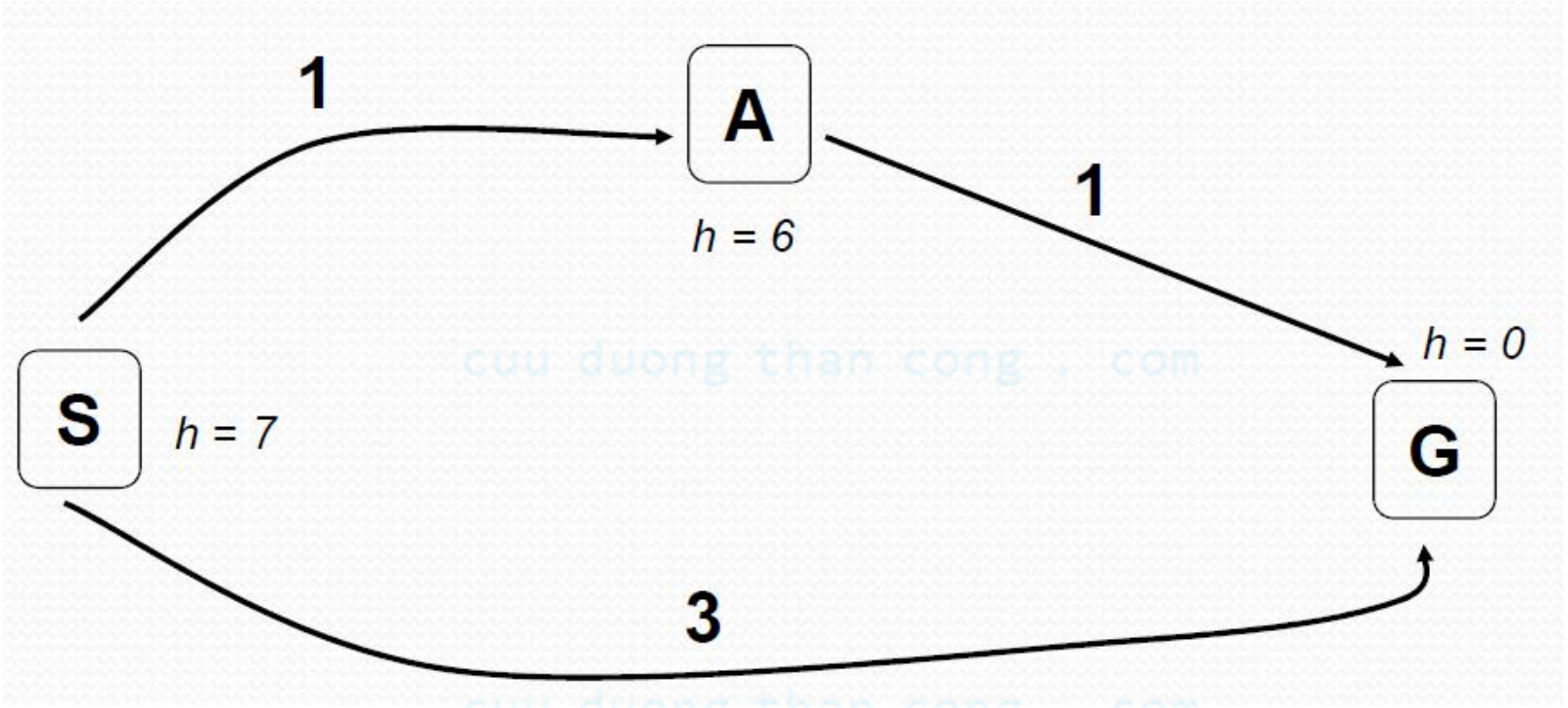
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374



Properties of A* search

- Completeness
  - Yes – with conditions
  - *Review: condition for completeness of UCS: g(n)>0*
- Optimality
  - Yes – with conditions
- Time complexity
  - Exponential
- Space complexity
  - Exponential (Keeps all nodes in memory)

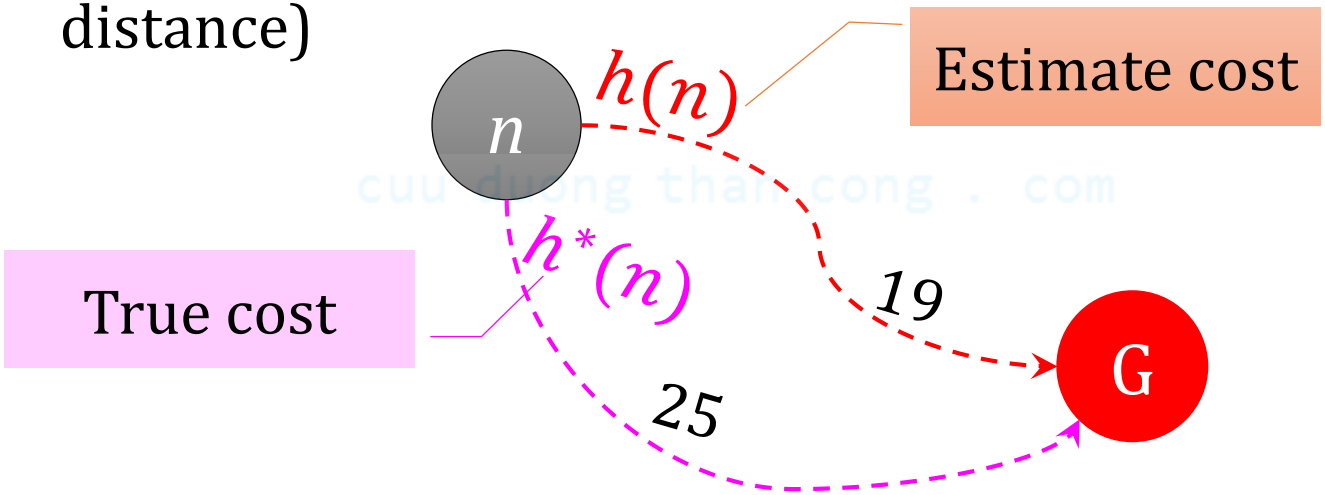
A* is not always optimal...



In what conditions, A* is optimal?

Optimality for TREE-SEARCH: Admissible heuristics

- A heuristic $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$ where $h^*(n)$ is 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
  - Example: $h_{SLD}(n)$ (never overestimates the actual road distance)

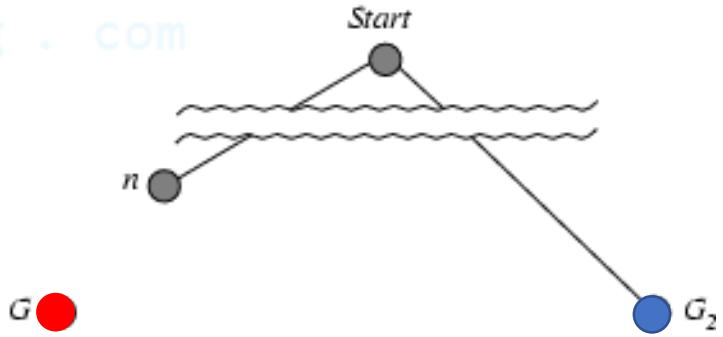


Optimality for TREE-SEARCH: Admissible heuristics

- Theorem: If *h(n)* is admissible, A* using **TREE-SEARCH** is optimal

Optimality of A* for TREE-SEARCH Proof

- Suppose some suboptimal goal G_2 has been generated and is in the frontier.
- Let n be an unexpanded node in the frontier such that n is on a shortest path to an optimal goal G.



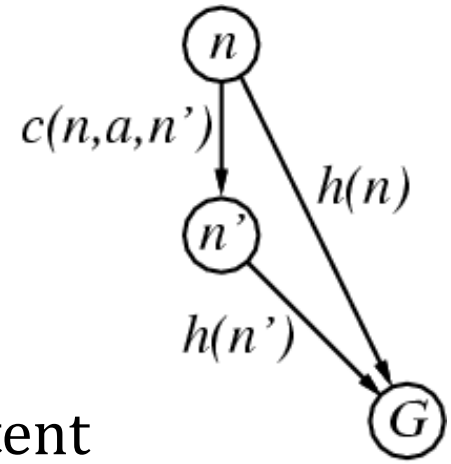
- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$
- $f(G_2) > f(G)$ (1)
- $h(n) \leq h^*(n)$ since h is admissible
- $g(n) + h(n) \leq g(n) + h^*(n)$
- $f(n) \leq f(G)$ (2)

From (1), (2): $f(G_2) > f(n)$, and A* **will never select G_2** for expansion

Optimality for GRAPH-SEARCH: Consistent heuristics

- In graph search, the optimal path to a repeated state could be discard if it is not the first one selected
  - → Admissibility is not sufficient for graph search
  - → This problem is fix by consistency property of $h(n)$
- A heuristic is consistent if for every node n, every successor n' of n generated by any action a,

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



(aka "monotonic")

- Admissible heuristics are generally consistent

Optimality for GRAPH-SEARCH: Consistent heuristics

➢If h is consistent, we have

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

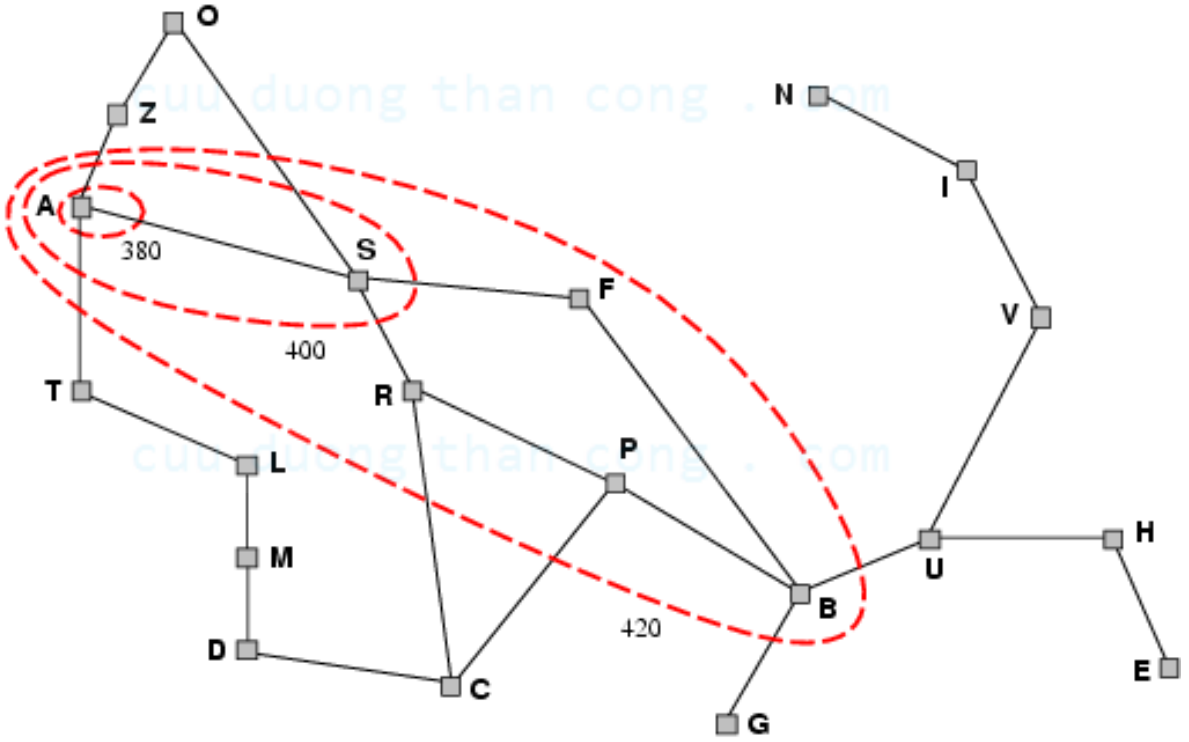
i.e., $f(n)$ is **non-decreasing** along any path.

Thus, first goal-state selected for expansion must be optimal

❑Theorem: If $h(n)$ is consistent, A* using **GRAPH-SEARCH** is optimal

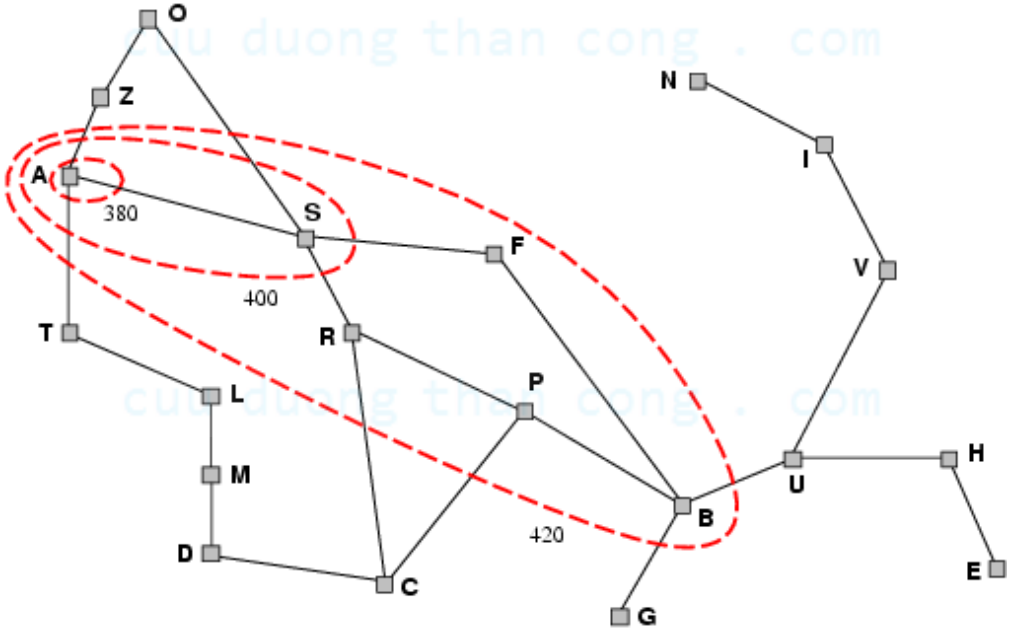
Contours of A* search

- 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 $f_i < f_{i+1}$



Contours of A* search

- With uniform-cost ($h(n) = 0$), contours will be circular
- With good heuristics, contours will be focused around optimal path
- A* will expand all nodes with cost $f(n) < C^*$



Comments on A*: The good

- A* never expand nodes with $f(n) > C^*$
  - All nodes like these are **PRUNED** while stile guaranteeing optimality
- A* is optimally efficient for any given consistent heuristic
  - No other optimal algorithm is guaranteed to expand fewer nodes than A*

Comments on A*: The bad

- A* expands all nodes with $f(n) < C^*$
  - This can still be exponentially large
  - A* usually runs out of space before it runs out of time
- Exponential growth will occur unless error in $h(n)$ grows no faster than log(true path cost)
  - In practice, error is usually proportional to true path cost (not log)
  - So exponential growth is common

→ Not practical for many large-scale problems

Memory-bound heuristic search

- Iterative-deepening A* (IDA*)
- Recursive best-first search (RBFS)
- Memory-bound A* (MA*)
- Simplified MA* (SMA*)

Memory-bound heuristic search

- In practice, A* runs out of memory before it runs out of time
  - How can we solve the memory problem for A* search?
- Idea:
  - Try something like DFS, but not forget everything about the branches we have partially explored

Iterative-deepening A* (IDA*)

- The main difference with IDS:
  - A* use f-cost (g+h) as the cutoff rather than the depth
  - At each iteration, the cutoff value is the smallest f-cost of any node that exceeded the cutoff on the previous iteration
- Difficulties:
  - Only practical for unit step costs
  - Difficult with real valued costs like UCS (see homework **3.17**)

Recursive best-first search (RBFS)

- Similar to DFS, but keeps track of the *f*-value of the best alternative path available from any ancestor of the current node

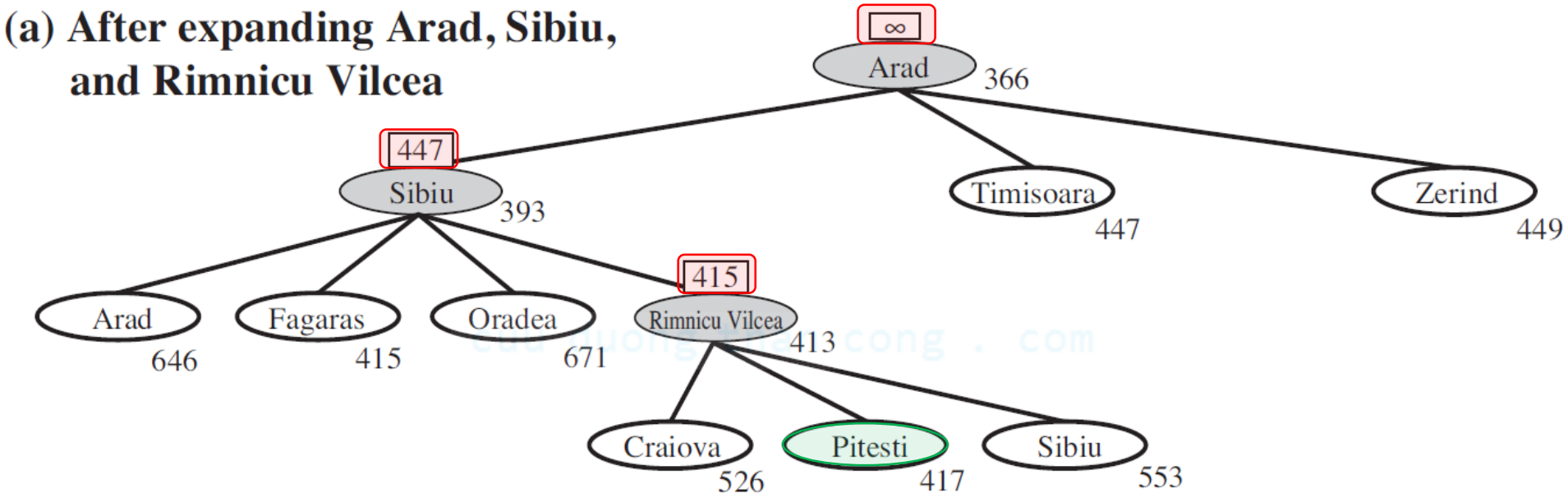
- If current node exceeds *f*-limit -> backtrack to alternative path

- As it backtracks, replace *f*-value of each node along the path with the best $f(n)$ value of its children
  - This allows it to return to this subtree, if it turns out to look better than alternatives

Recursive best-first search (RBFS)

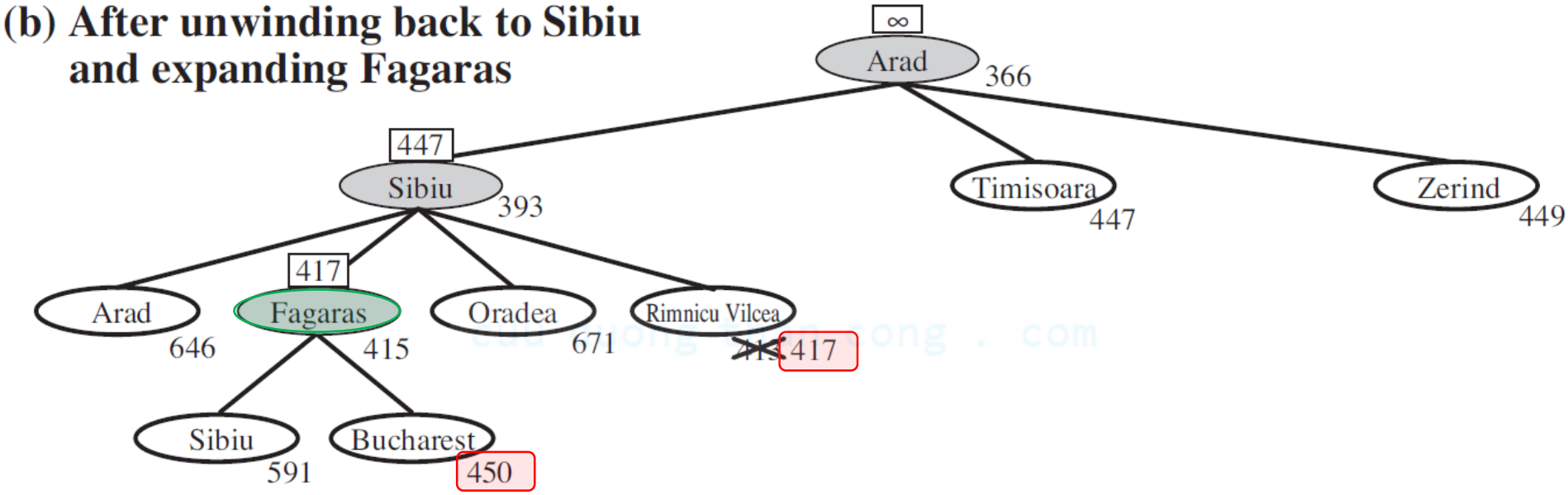
```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution, or failure
    return RBFS(problem, MAKE-NODE(problem.INITIAL-STATE), ∞)

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
```

Recursive best-first search (RBFS)



- Path until Rumnicu Vilcea is already expanded
- Above node: *f*-limit for every recursive call is shown on top.
- Below node: *f(n)*
- The path is followed until Pitesti which has a *f*-value worse than the *f-limit*.

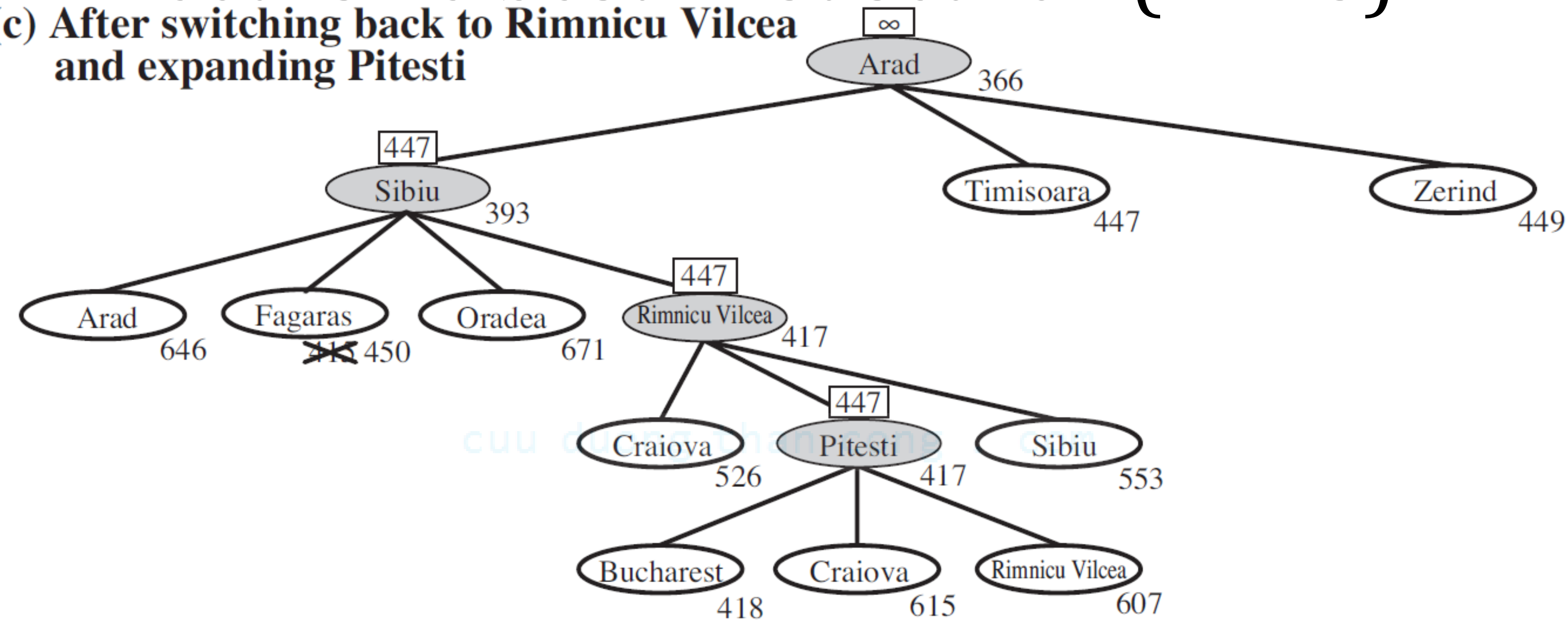
Recursive best-first search (RBFS)



- Unwind recursion and store best *f*-value for current best leaf Rimnicu Vilcea
  - *result*, *f* [*best*] ← RBFS(*problem*, *best*, min(*f_limit*, *alternative*))
- *best* is now Fagaras. Call RBFS for new *best*
  - *best* value is now 450

Recursive best-first search (RBFS)

(c) After switching back to Rimnicu Vilcea and expanding Pitesti



- ❑Unwind recursion and store best *f*-value for current best leaf Fagaras
  - ○ *result, f* [*best*] ← RBFS*(problem, best,* min(*f_limit, alternative*))
- ❑*best* is now Rimnicu Viclea (again). Call RBFS for new *best*
  - ○ Subtree is again expanded
  - ○ Best alternative subtree is now through Timisoara
- ❑Solution is found since because 447 > 418.

Properties of RBFS

- Optimality
  - Like A*, optimal if *h(n)* is admissible
- Time complexity difficult to characterize
  - Depends on accuracy if h(n) and how often best path changes.
  - Can end up "switching" back and forth
- Space complexity is
  - Linear time: *O(bd)*
  - Other extreme to A* - uses ***too little*** memory

(Simplified) Memory-bound A* - (S)MA*

- This is like A*, but when memory is full we delete the worst node (largest *f*-value).
- Like RBFS, SMA∗ backs up the value of the forgotten node to its parent. If there is a tie (equal f-values) we delete the oldest nodes first.
- Simplified-MA* finds the optimal *reachable* solution given the memory constraint.
- Time can still be exponential.

Next class

- Individual Assignment 2
- Chapter 2: Solving Problems by Searching (cont.)
  - Heuristic Functions