Artificial Intelligence

Kushal Shah @ Sitare

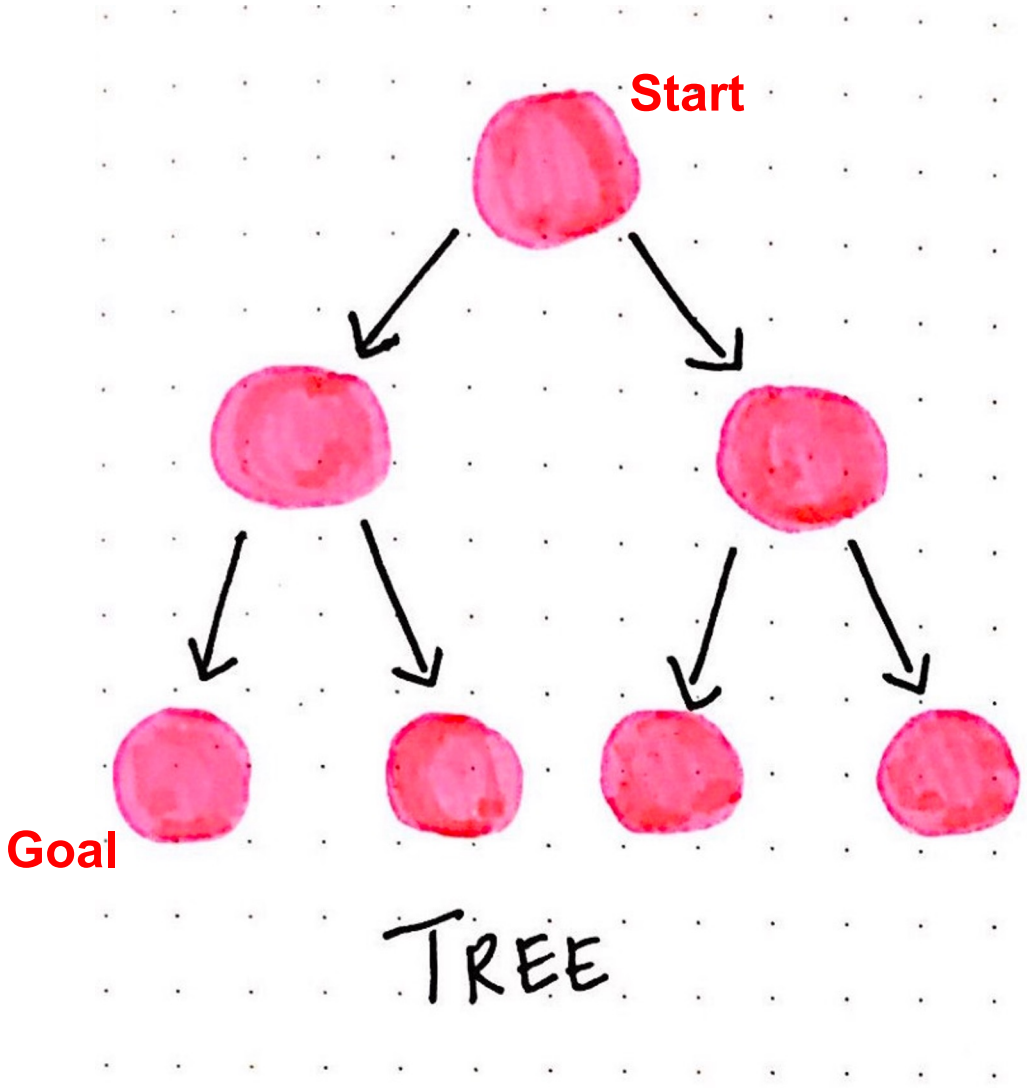
Uninformed (blind) search

- Breadth First Search [BFS], Depth First Search [DFS], etc

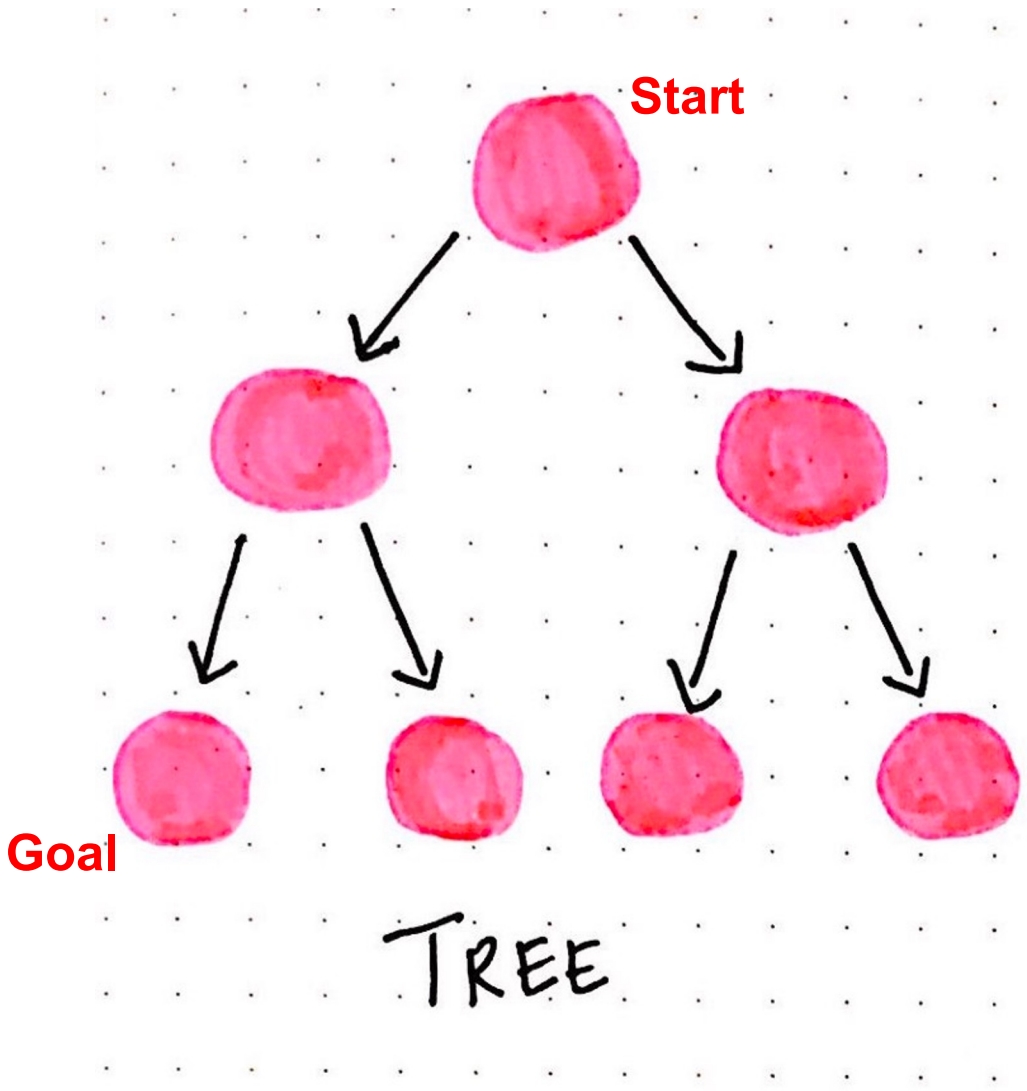
- Blind strategies have no additional information about states beyond that provided in the problem definition.

- All they can do is generate successors and distinguish a goal state from a non-goal state.

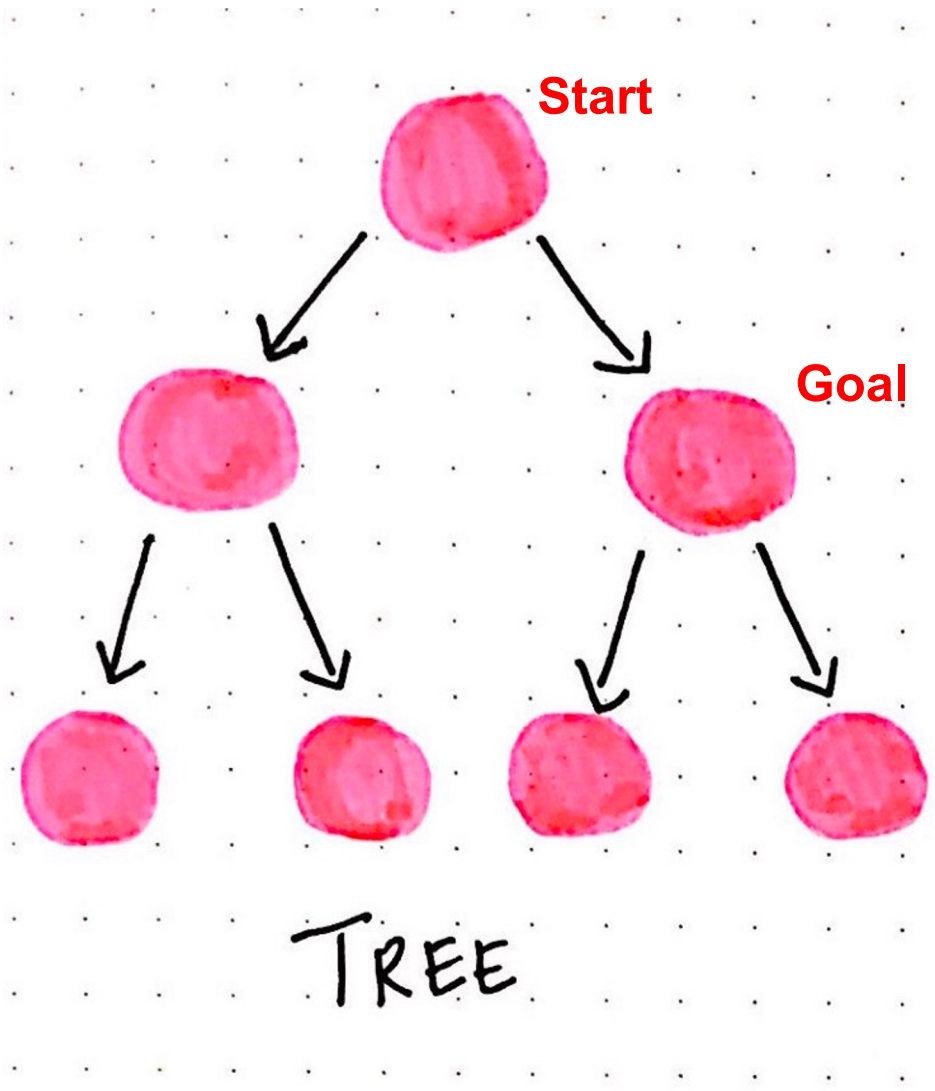
- All search strategies are distinguished by the order in which nodes are expanded.



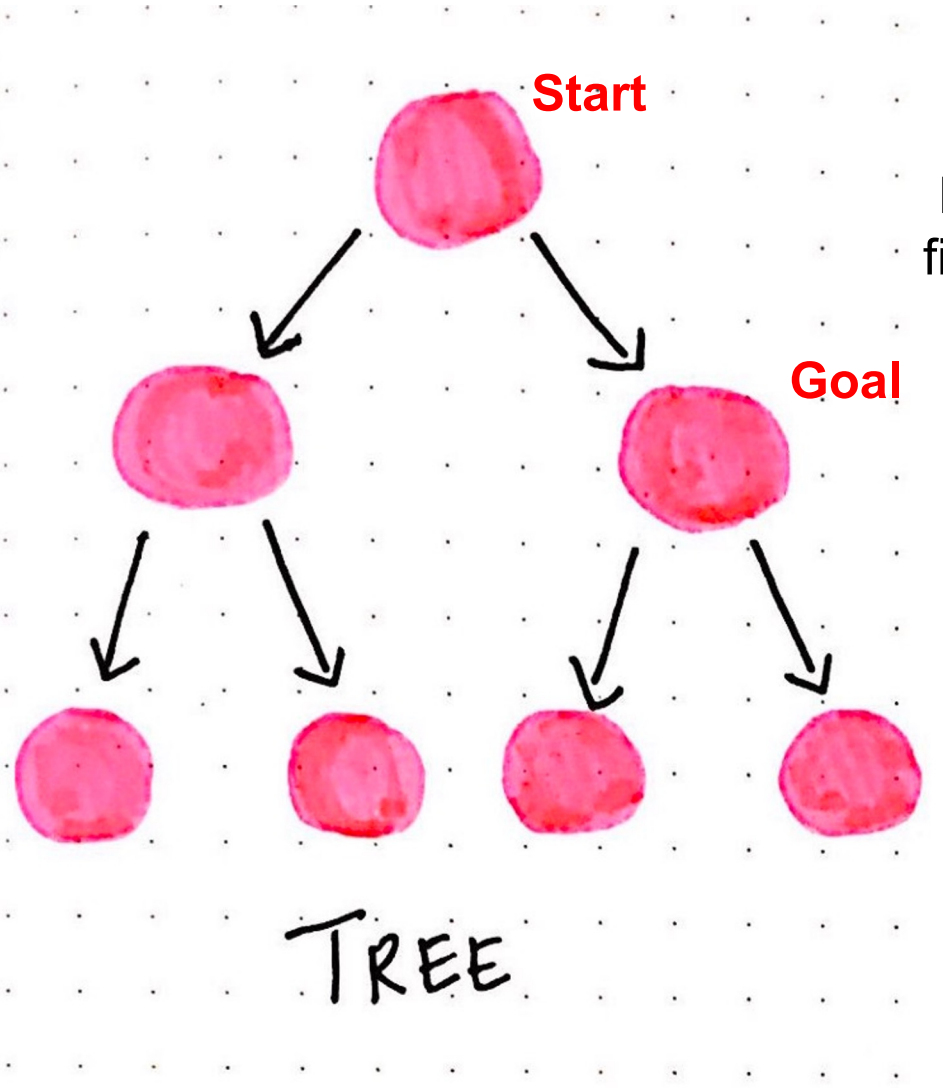
Which algo will find the optimal solution?



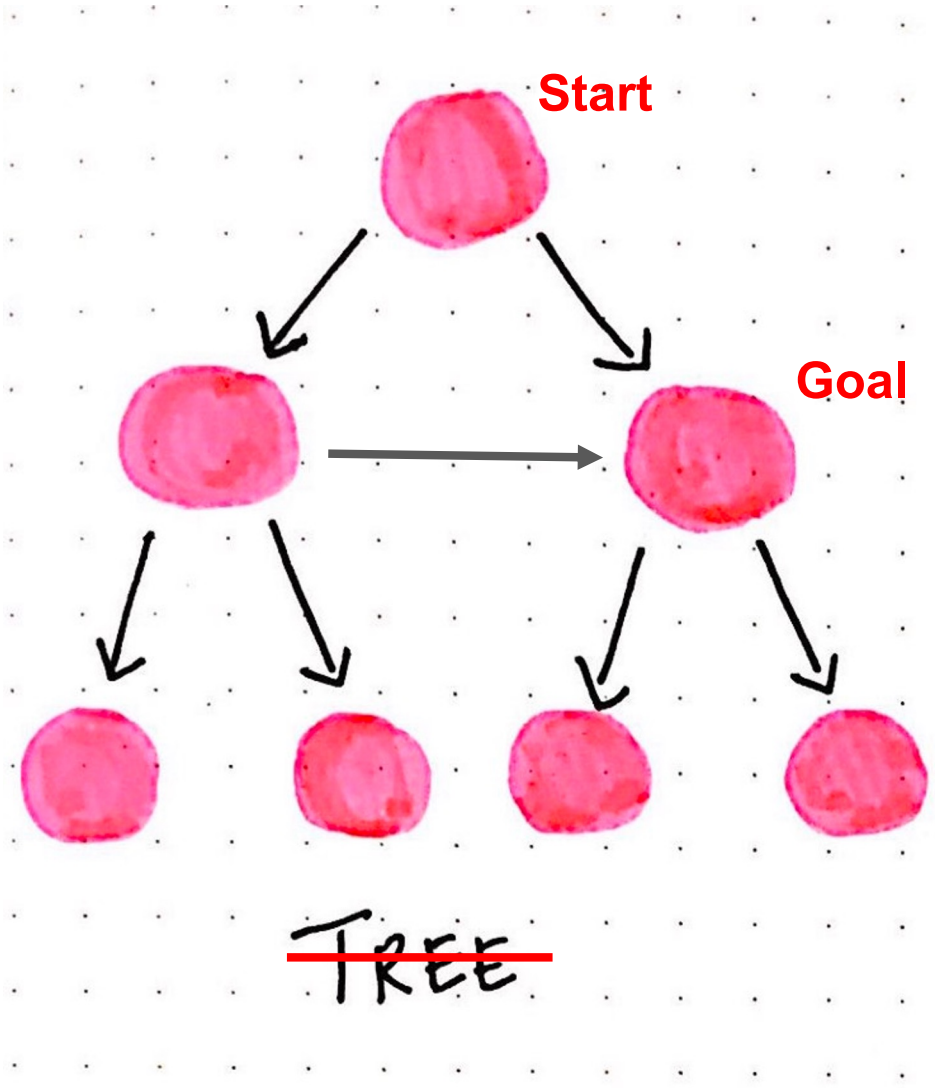
Both BFS and DFS find optimal solution.



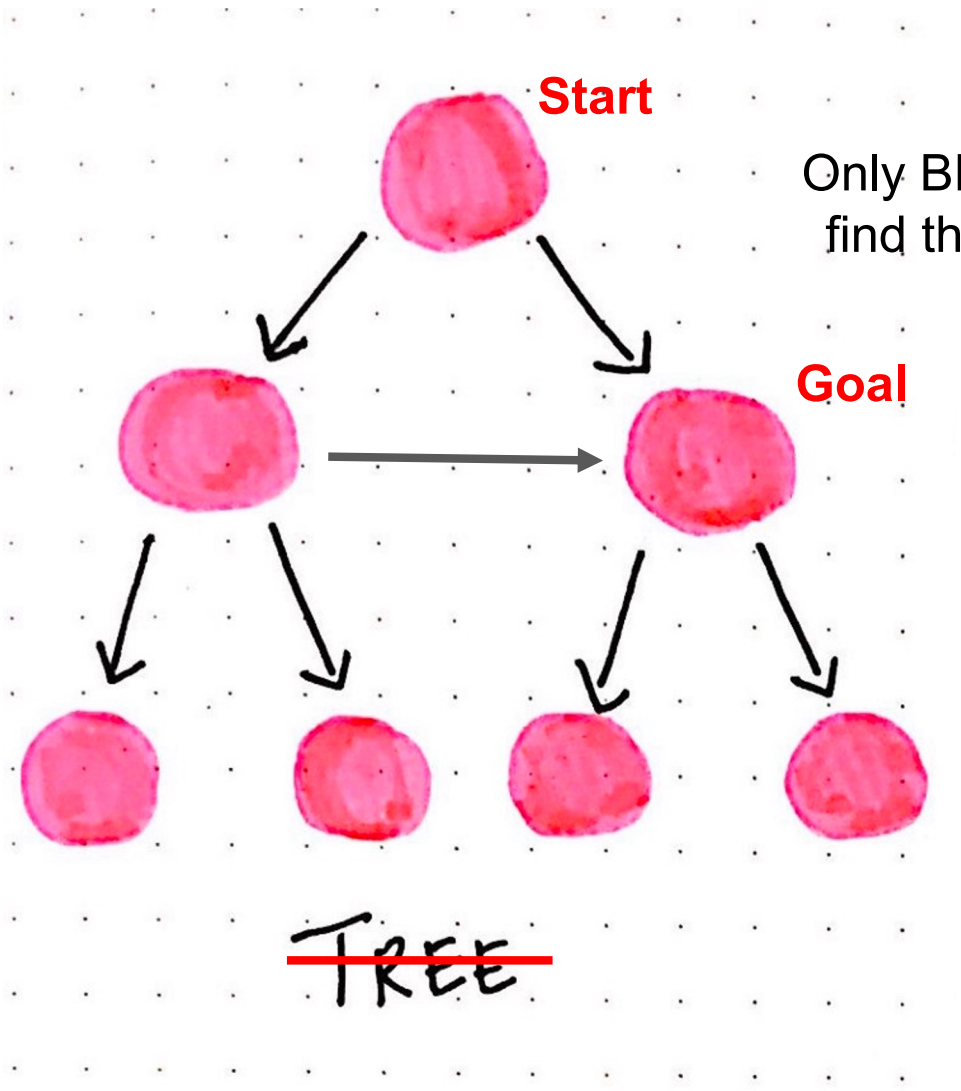
Which algo will find the optimal solution?



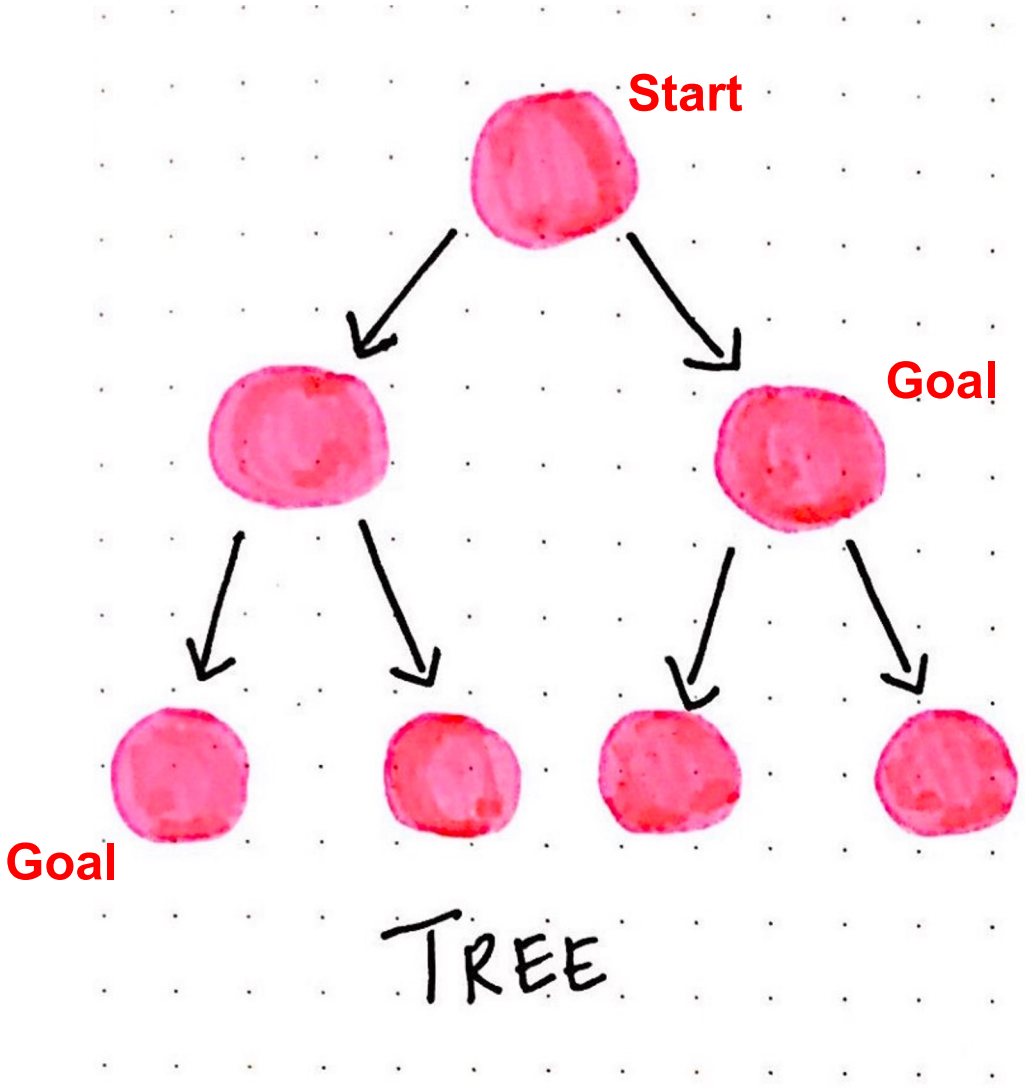
Both BFS and DFS find optimal solution.



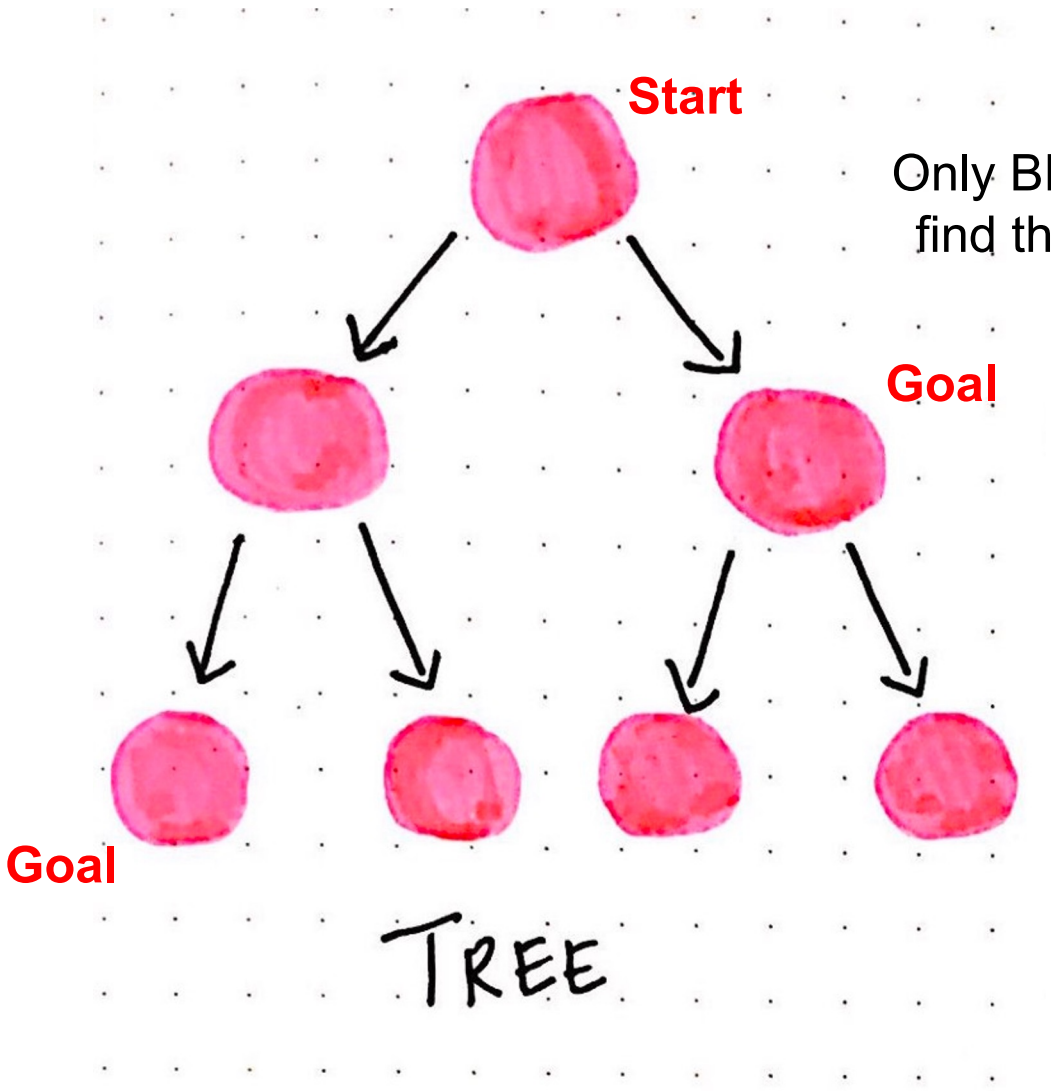
Which algo will find the optimal solution?



Only BFS is guaranteed to find the optimal solution.



Which algo will find the optimal solution?



Only BFS is guaranteed to find the optimal solution.

Informed (heuristic) Search

- Greedy best-first search [Section 3.5.1]

- A* algorithm and its Optimality [Section 3.5.2]

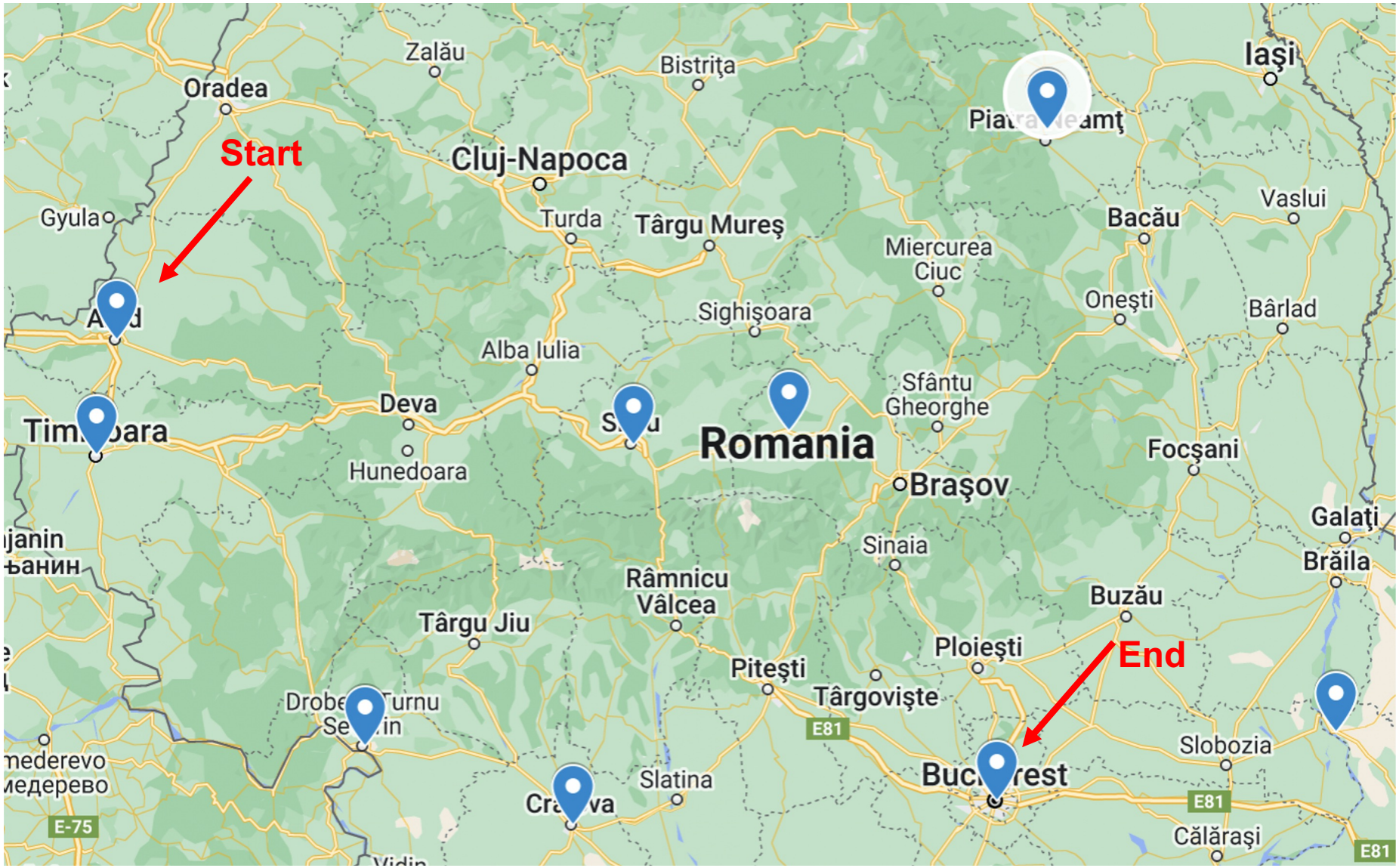
- Heuristic Function – admissible and dominating heuristic [Section 3.6]

Informed (heuristic) search

- Best-first search :
  a node is selected for expansion based on an evaluation function, f(n)

- The evaluation function is construed as a cost estimate
  and its choice determines the search strategy

- The node with the lowest evaluation function value is expanded first.


Start
End
Zalău
Bistrița
Iași
Oradea
Cluj-Napoca
Gyula
Turda
Târgu Mureș
Bacău
Vaslui
Miercurea Ciuc
Onești
Bârlad
Sighișoara
Alba Iulia
Deva
Sfântu Gheorghe
Romania
Hunedoara
Focșani
Brașov
Galați
Brăila
Sinaia
Râmnicu Vâlcea
Buzău
Târgu Jiu
Ploiești
Pitești
Târgoviște
E81
Slobozia
Slatina
E-75
Călărași

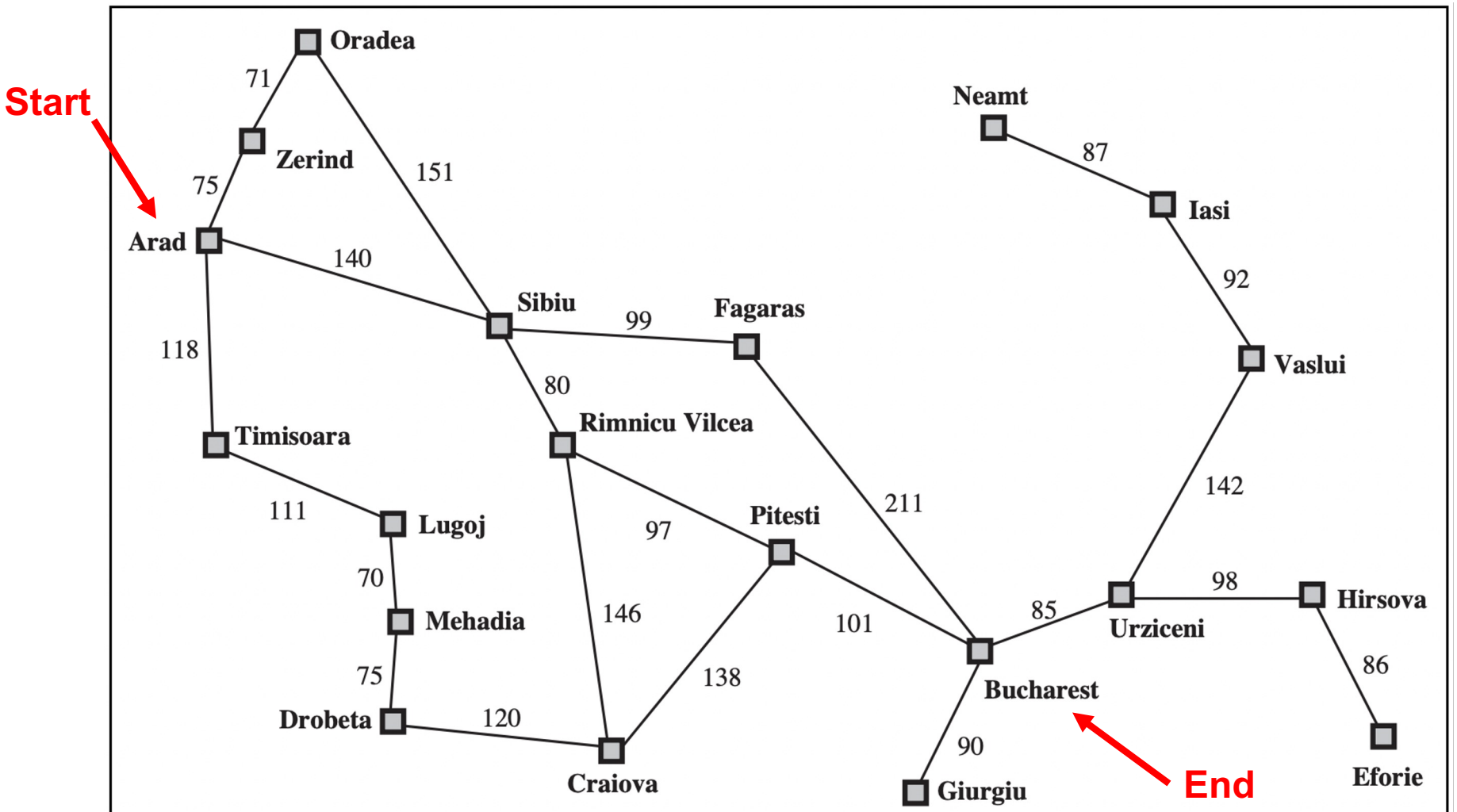
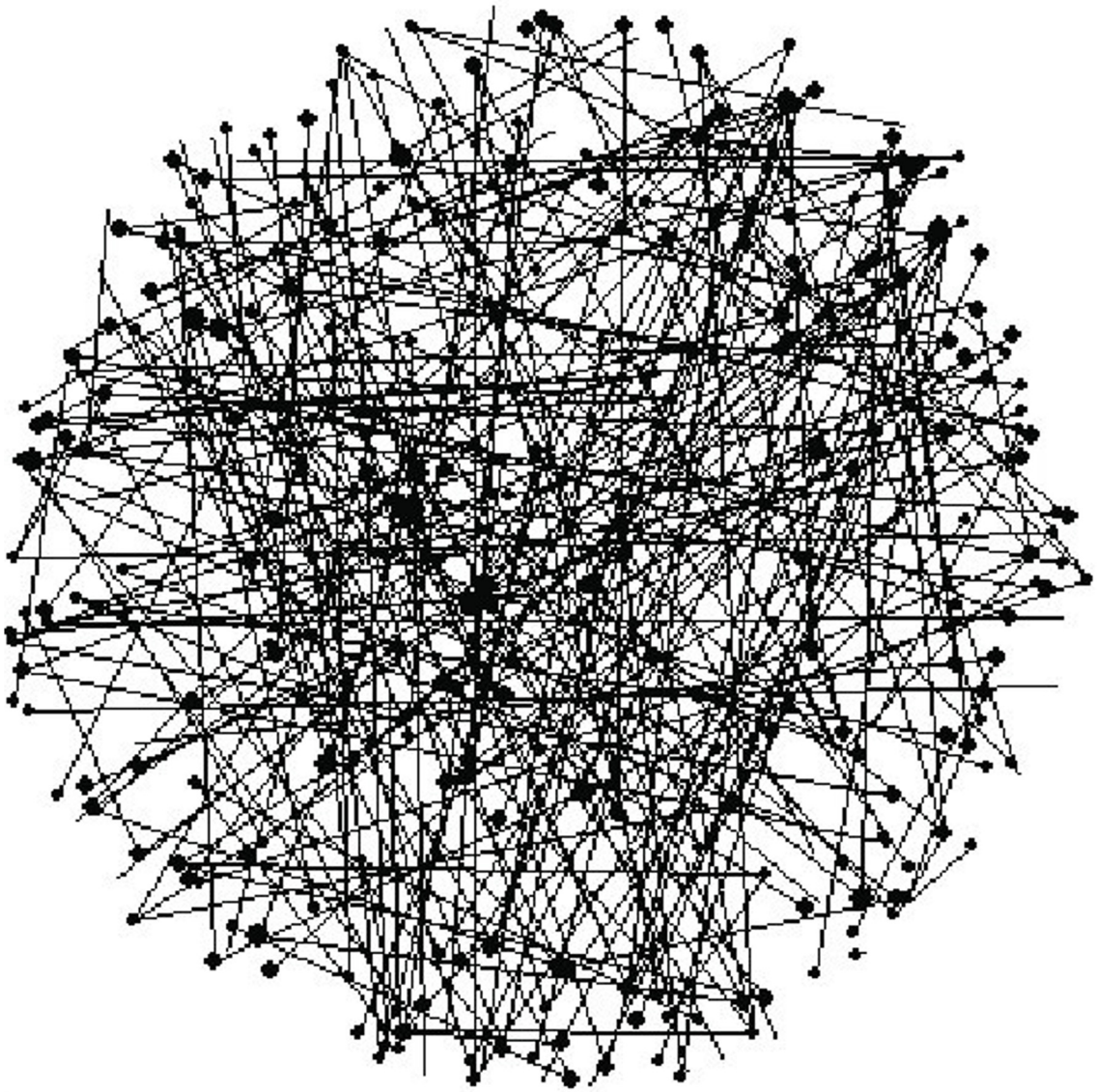


Figure 3.2 A simplified road map of part of Romania. (distances in kilometers)



Greedy best-first search

Greedy best-first search

- Expand the node that is closest to the goal as determined by a heuristic function

Greedy best-first search

- Expand the node that is closest to the goal as determined by a heuristic function
- Assumes that this is likely to lead to a solution quickly

Greedy best-first search

- Expand the node that is closest to the goal as determined by a heuristic function
- Assumes that this is likely to lead to a solution quickly

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.

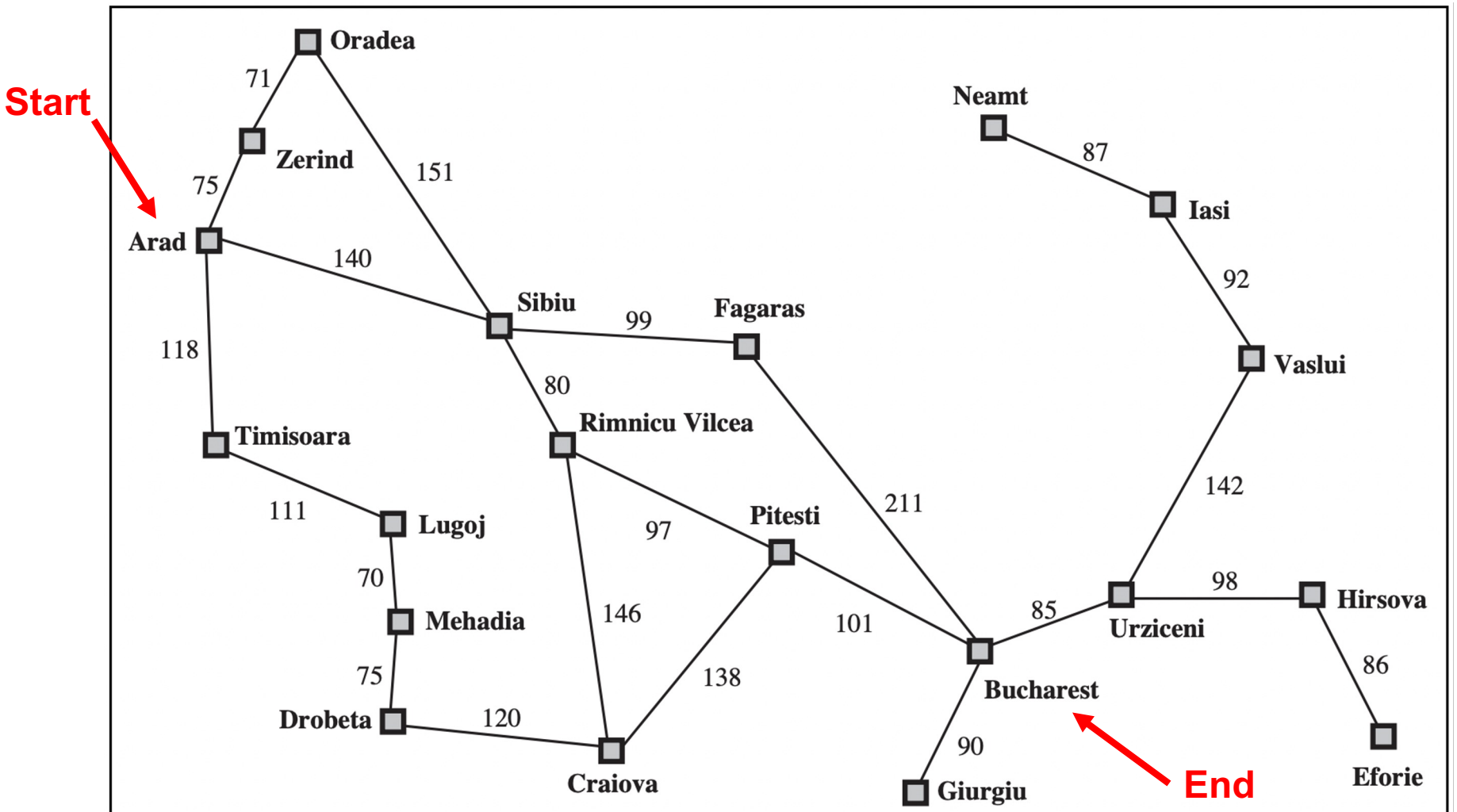


Figure 3.2 A simplified road map of part of Romania. (distances in kilometers)

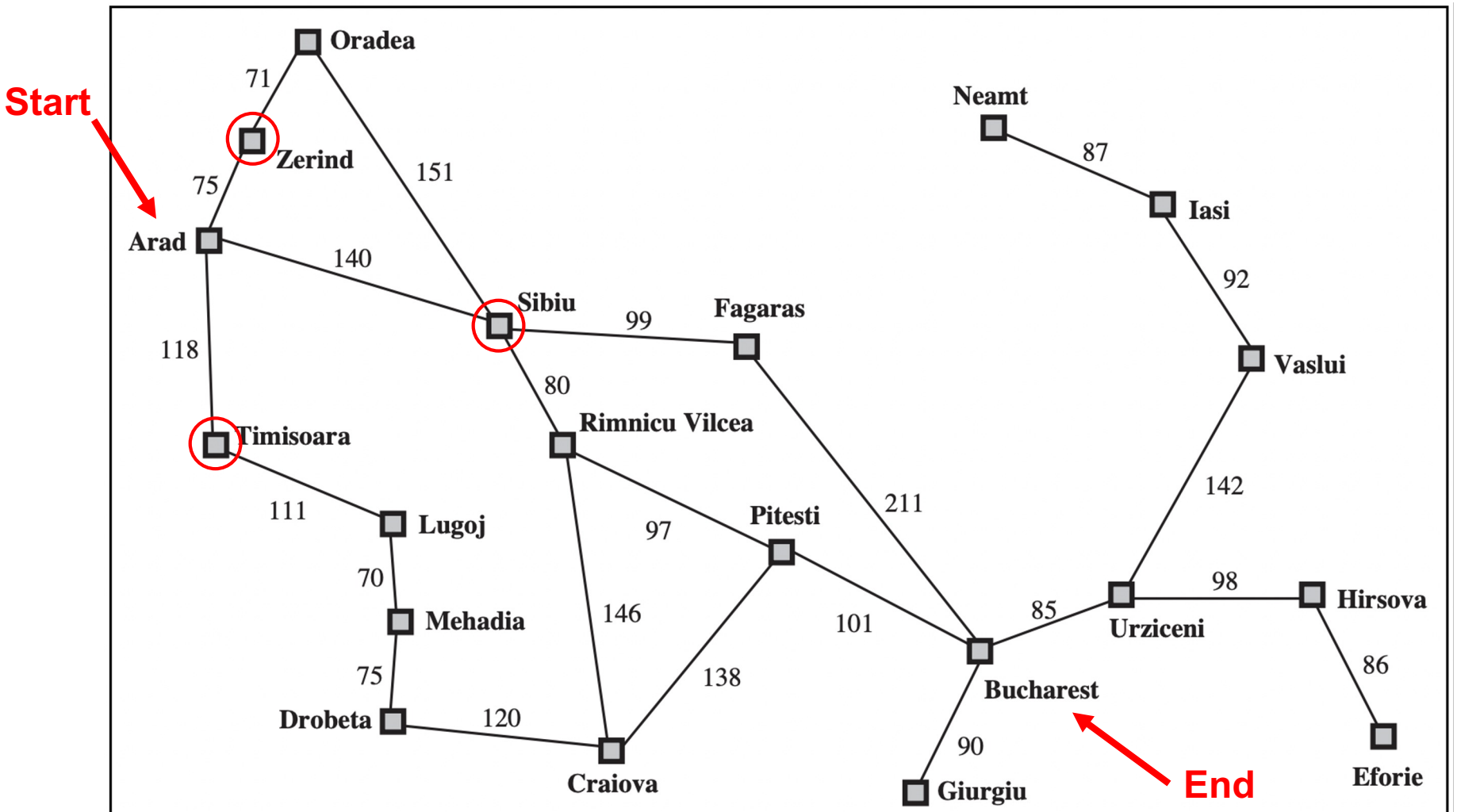


Figure 3.2 A simplified road map of part of Romania. (distances in kilometers)

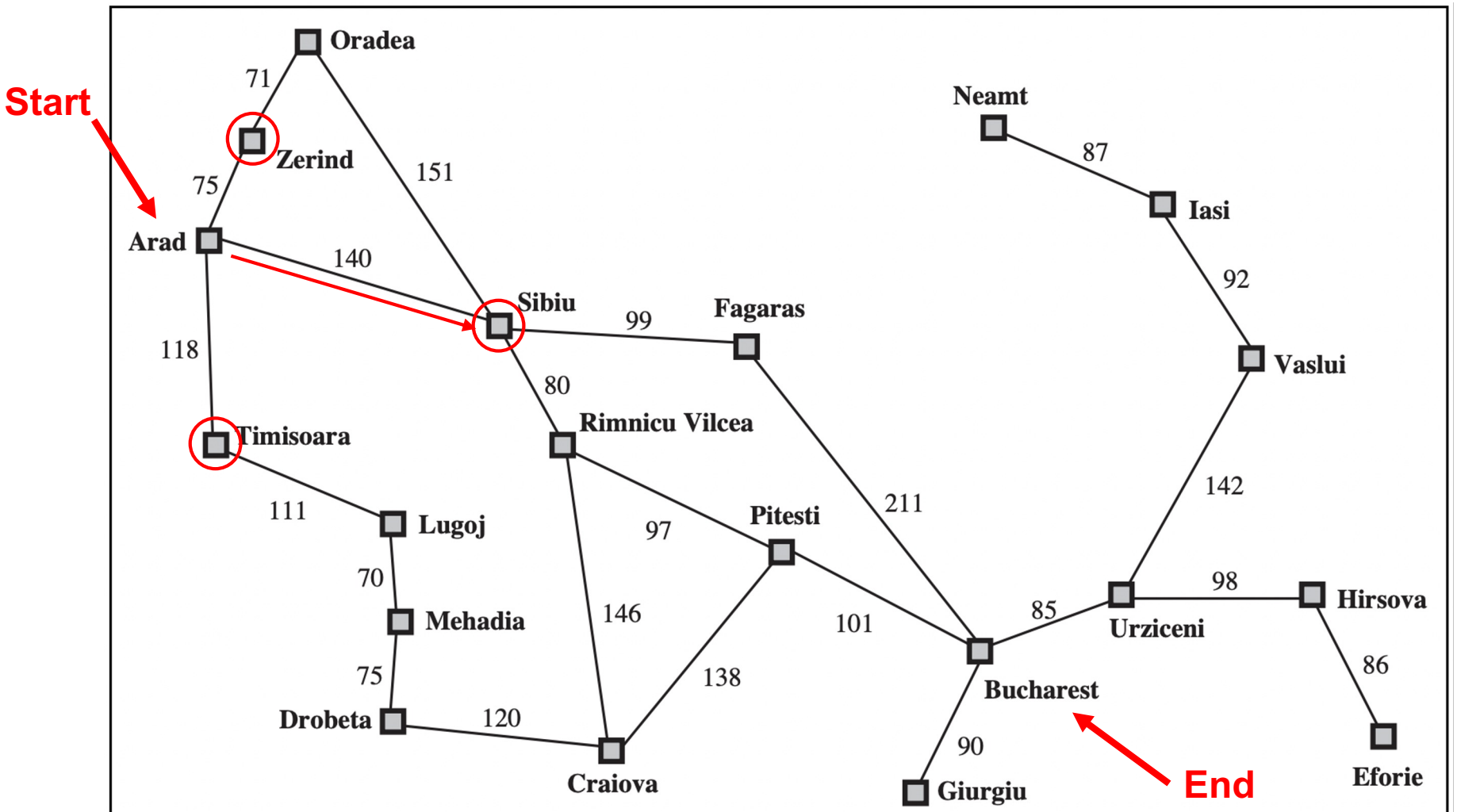


Figure 3.2 A simplified road map of part of Romania. (distances in kilometers)

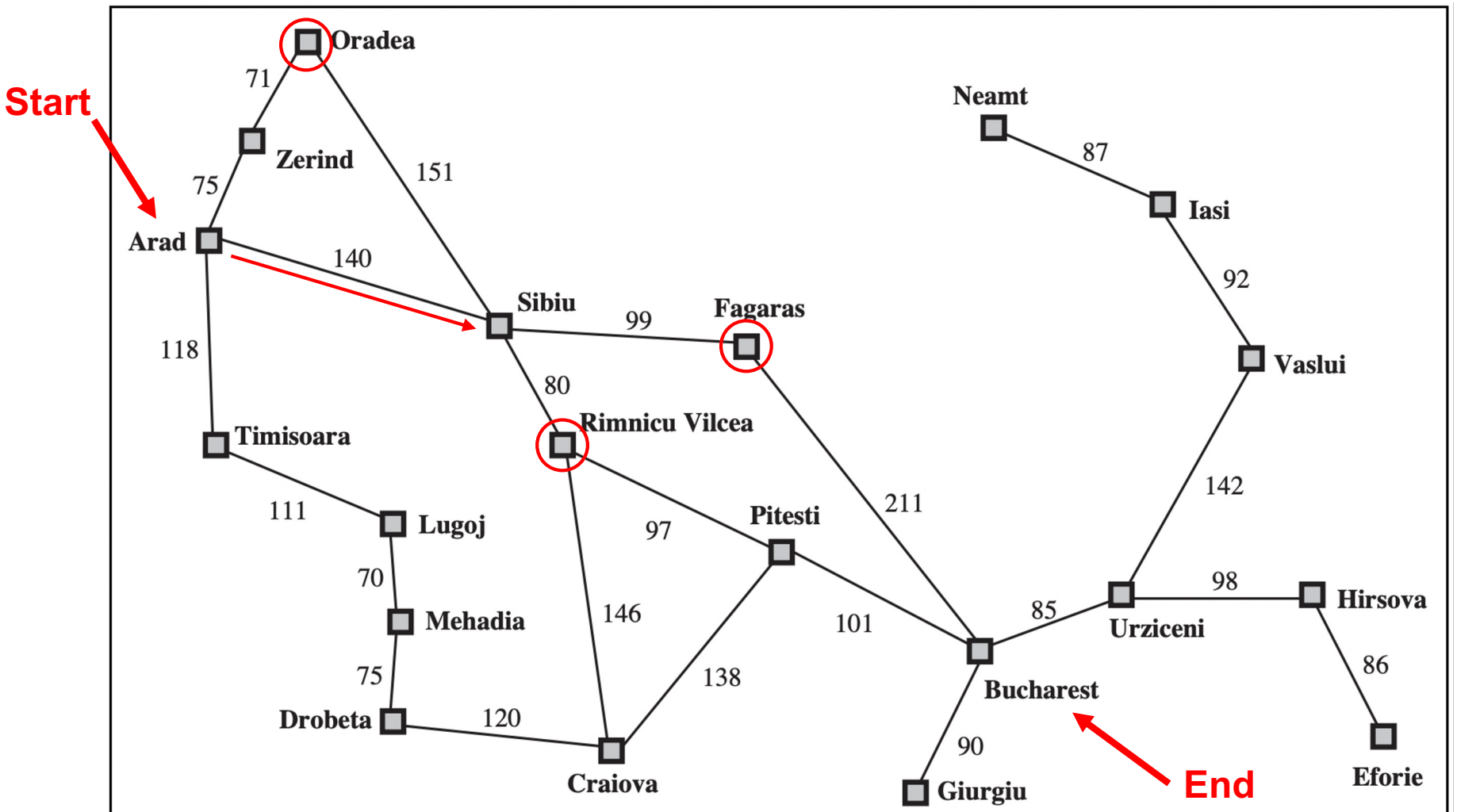


Figure 3.2 A simplified road map of part of Romania. (distances in kilometers)

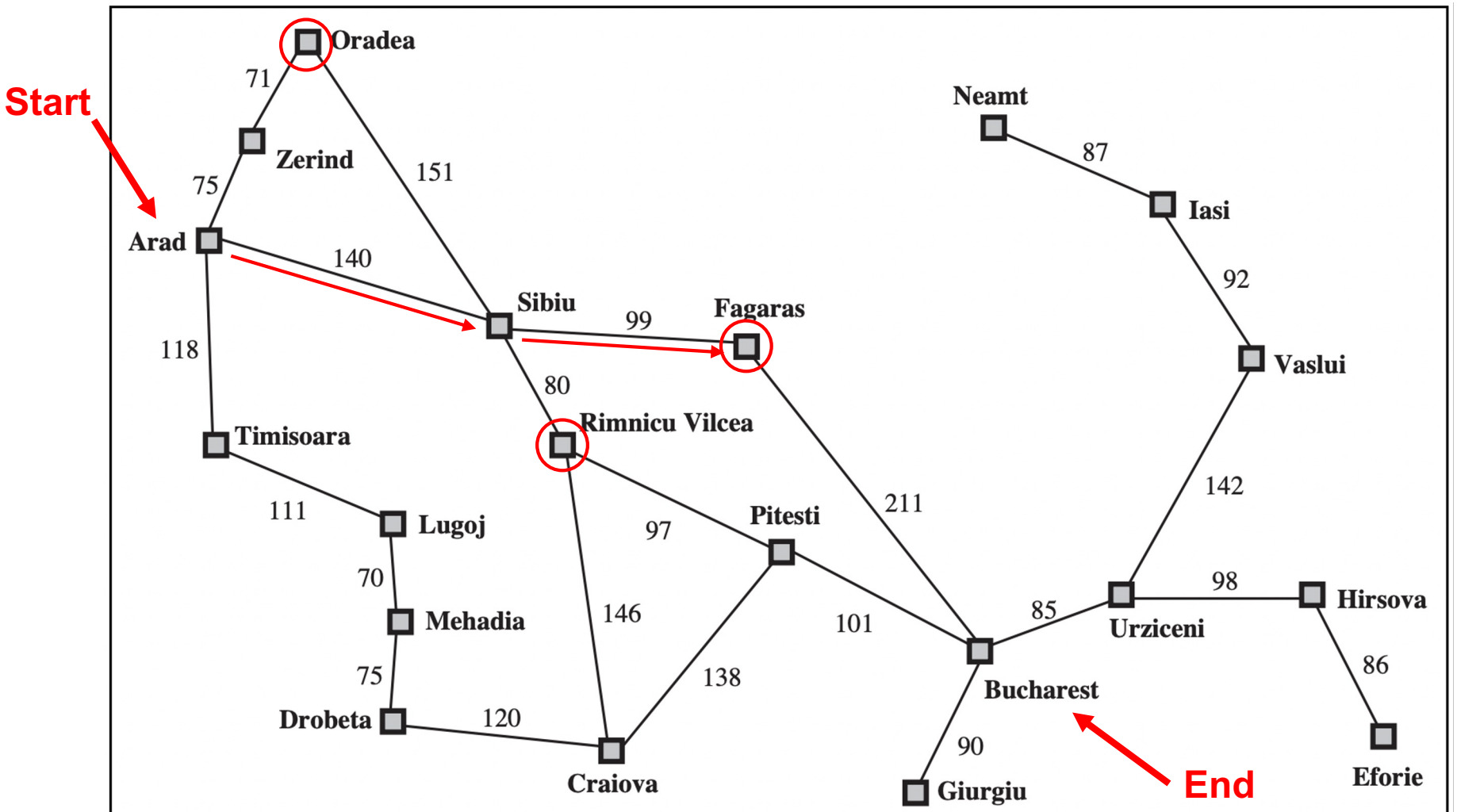


Figure 3.2 A simplified road map of part of Romania. (distances in kilometers)

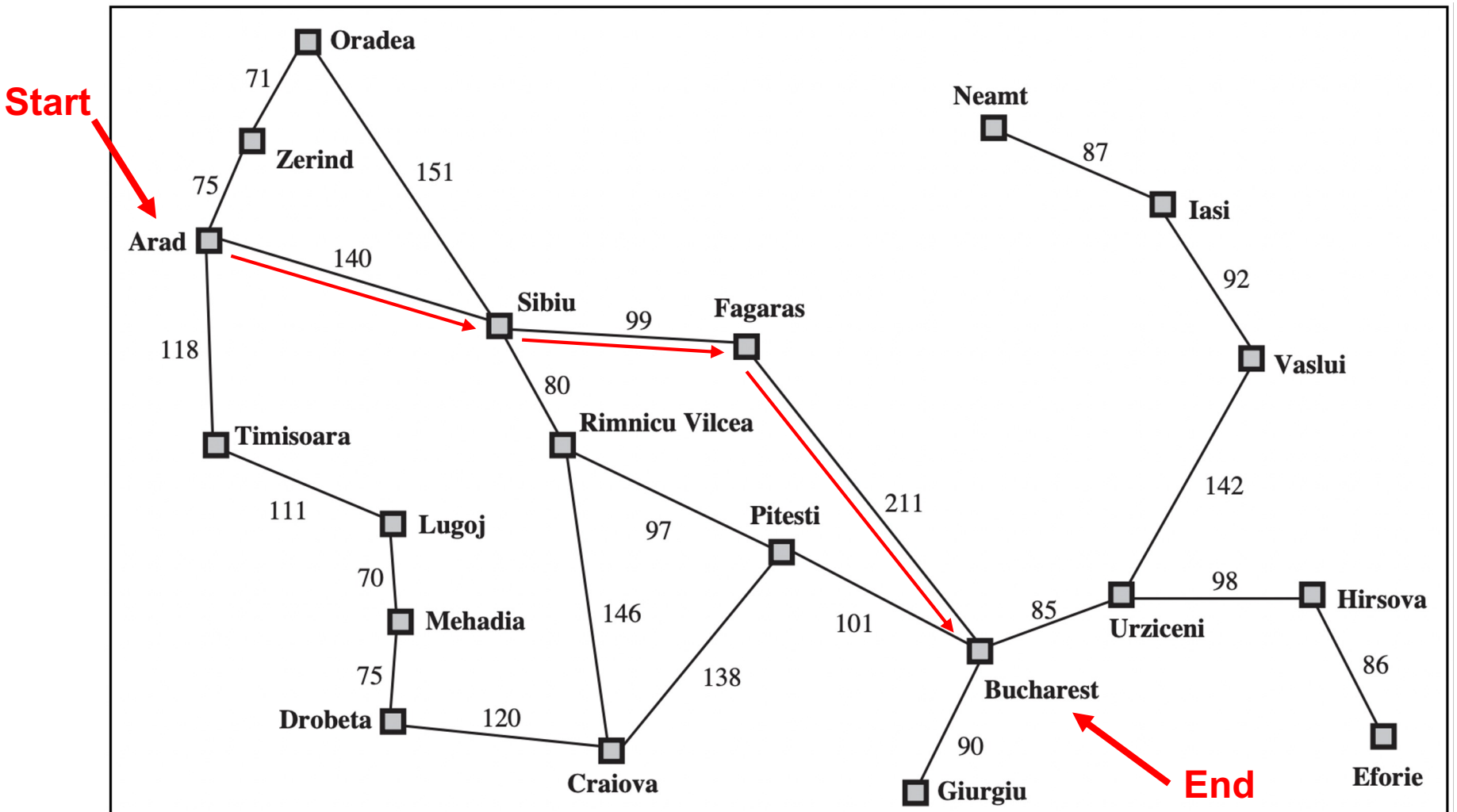
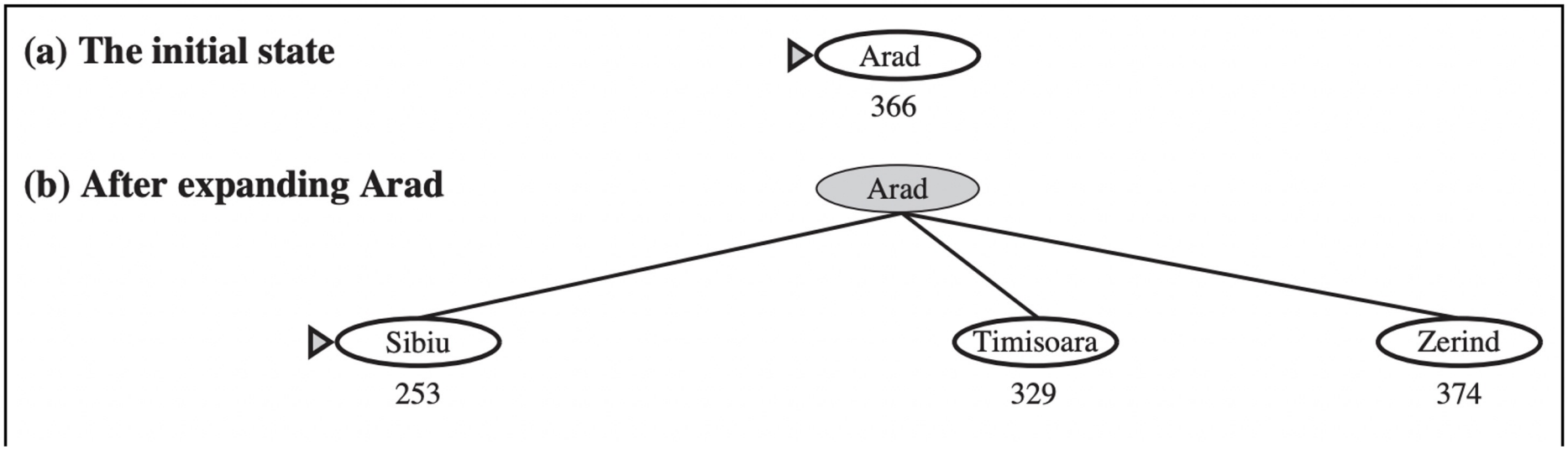
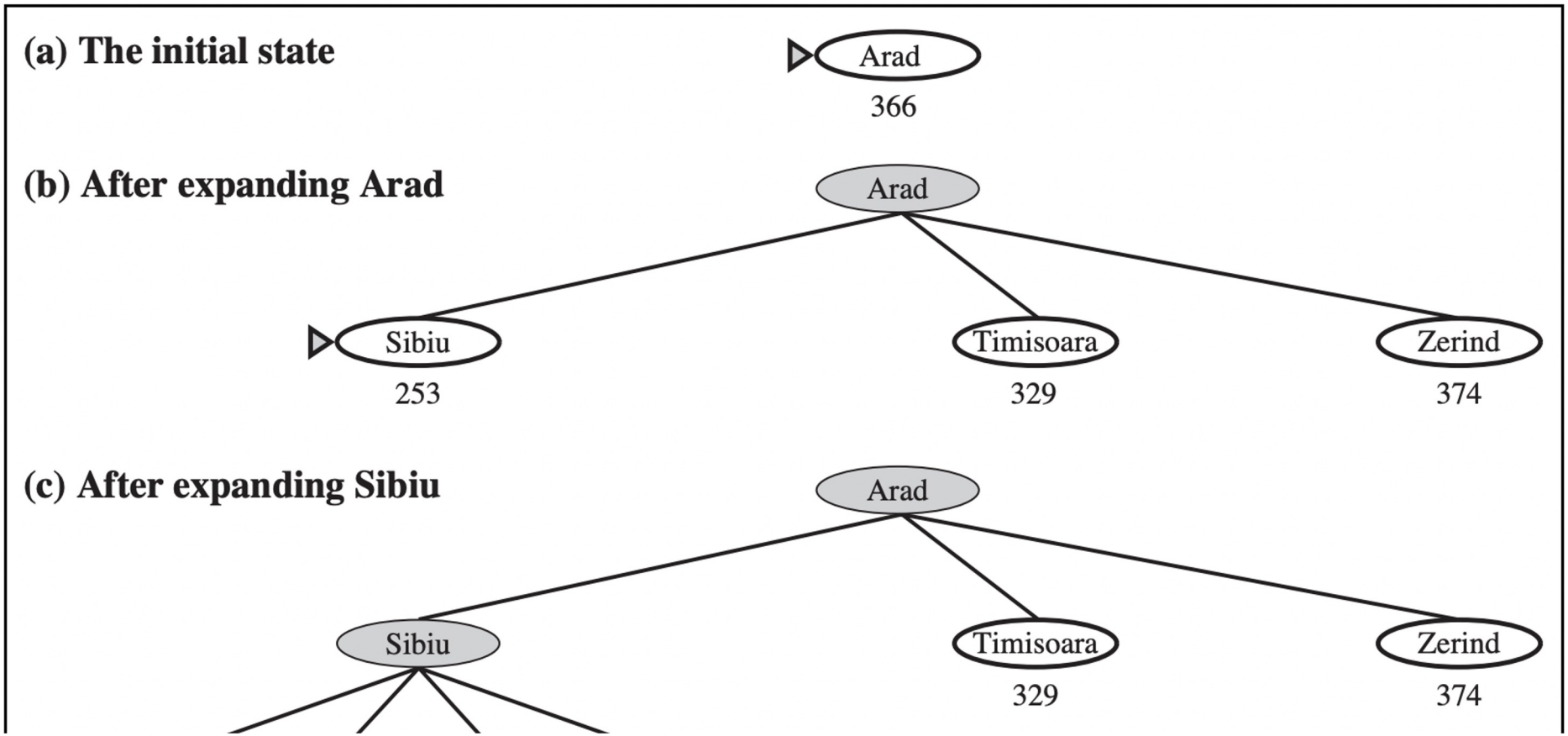
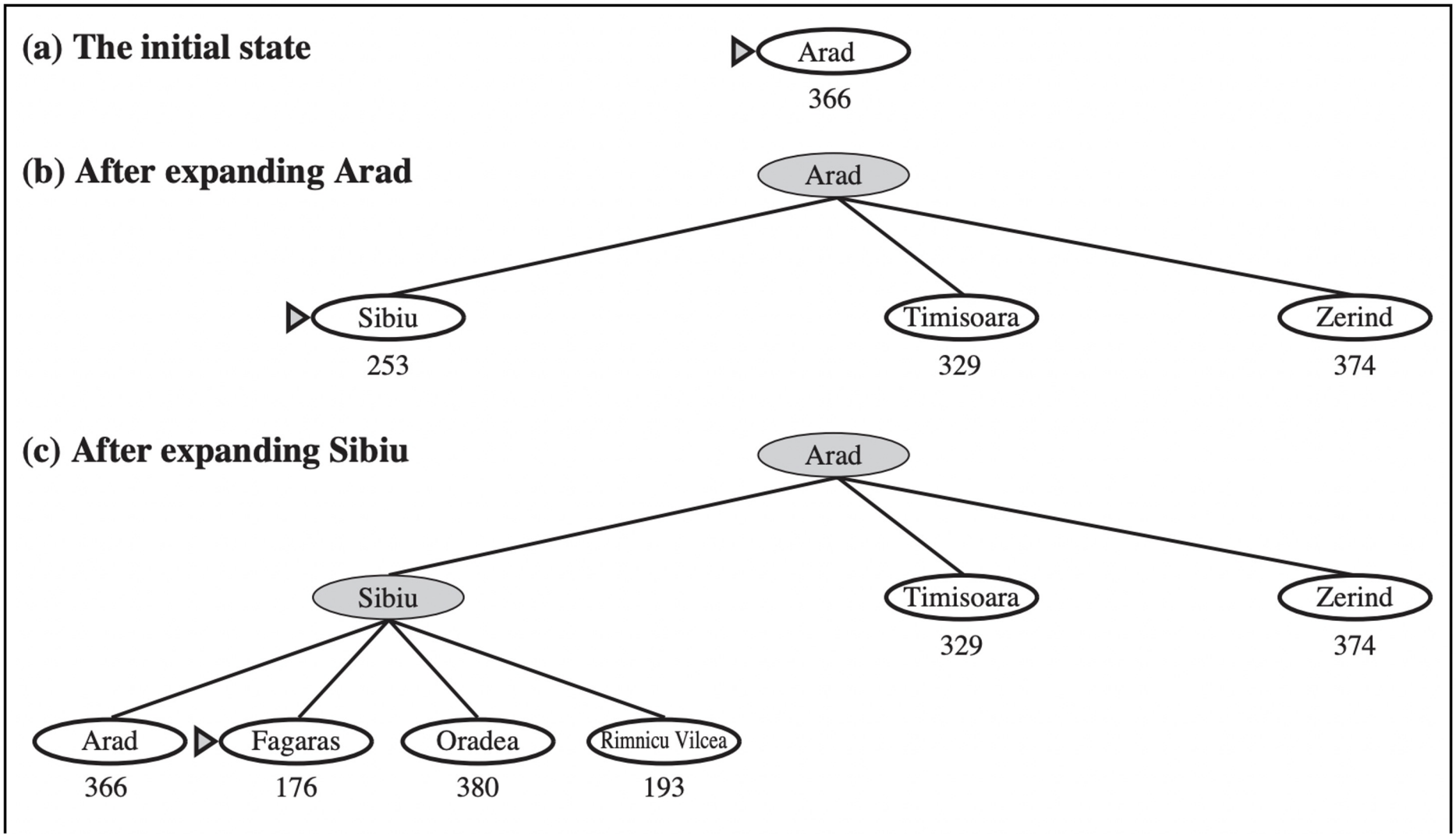


Figure 3.2 A simplified road map of part of Romania. (distances in kilometers)

(a) The initial state


(a) The initial state
Arad
366
(b) After expanding Arad
Arad
Sibiu
253
Timisoara
329
Zerind
374


(a) The initial state
Arad
366
(b) After expanding Arad
Arad
Sibiu
Timisoara
Zerind
253
329
374
(c) After expanding Sibiu
Arad
Sibiu
Timisoara
Zerind
329
374


(a) The initial state
Arad
366
(b) After expanding Arad
Arad
Sibiu
Timisoara
Zerind
253
329
374
(c) After expanding Sibiu
Arad
Sibiu
Timisoara
Zerind
329
374
Arad
Fagaras
Oradea
Rimnicu Vilcea
366
176
380
193

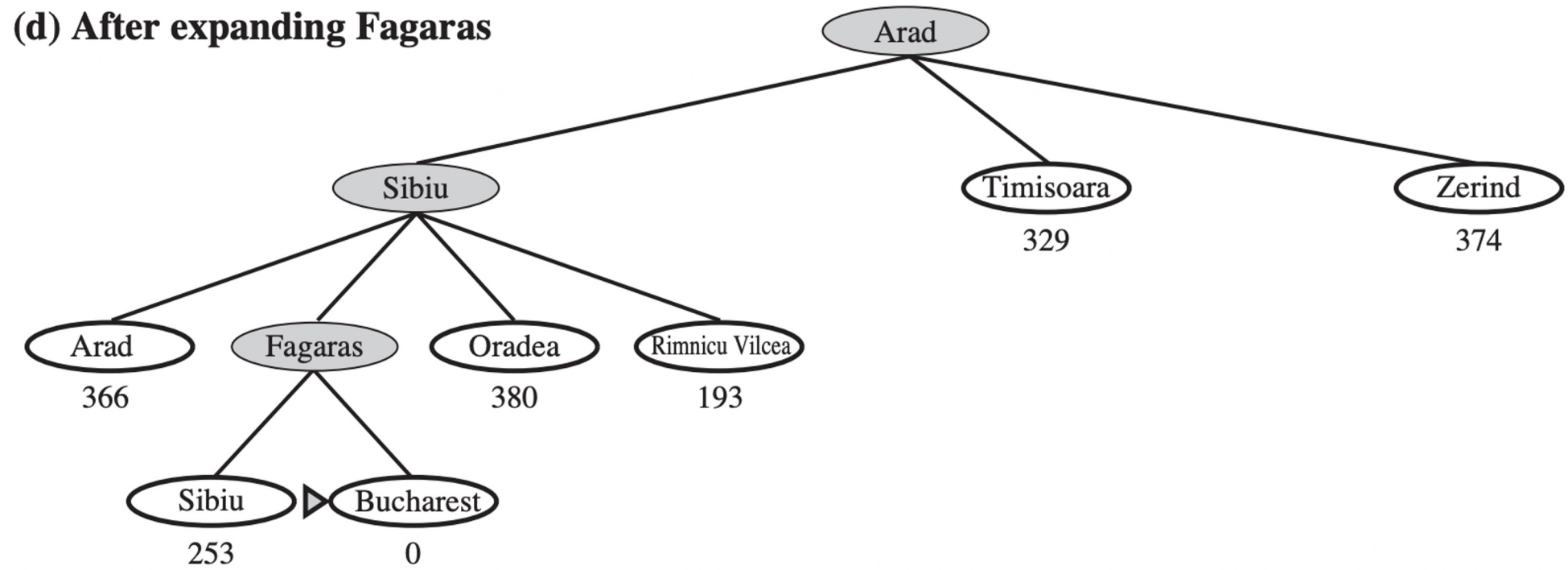


Figure 3.23 Stages in a greedy best-first tree search for Bucharest with the straight-line distance heuristic h_{SLD}. Nodes are labeled with their h-values.

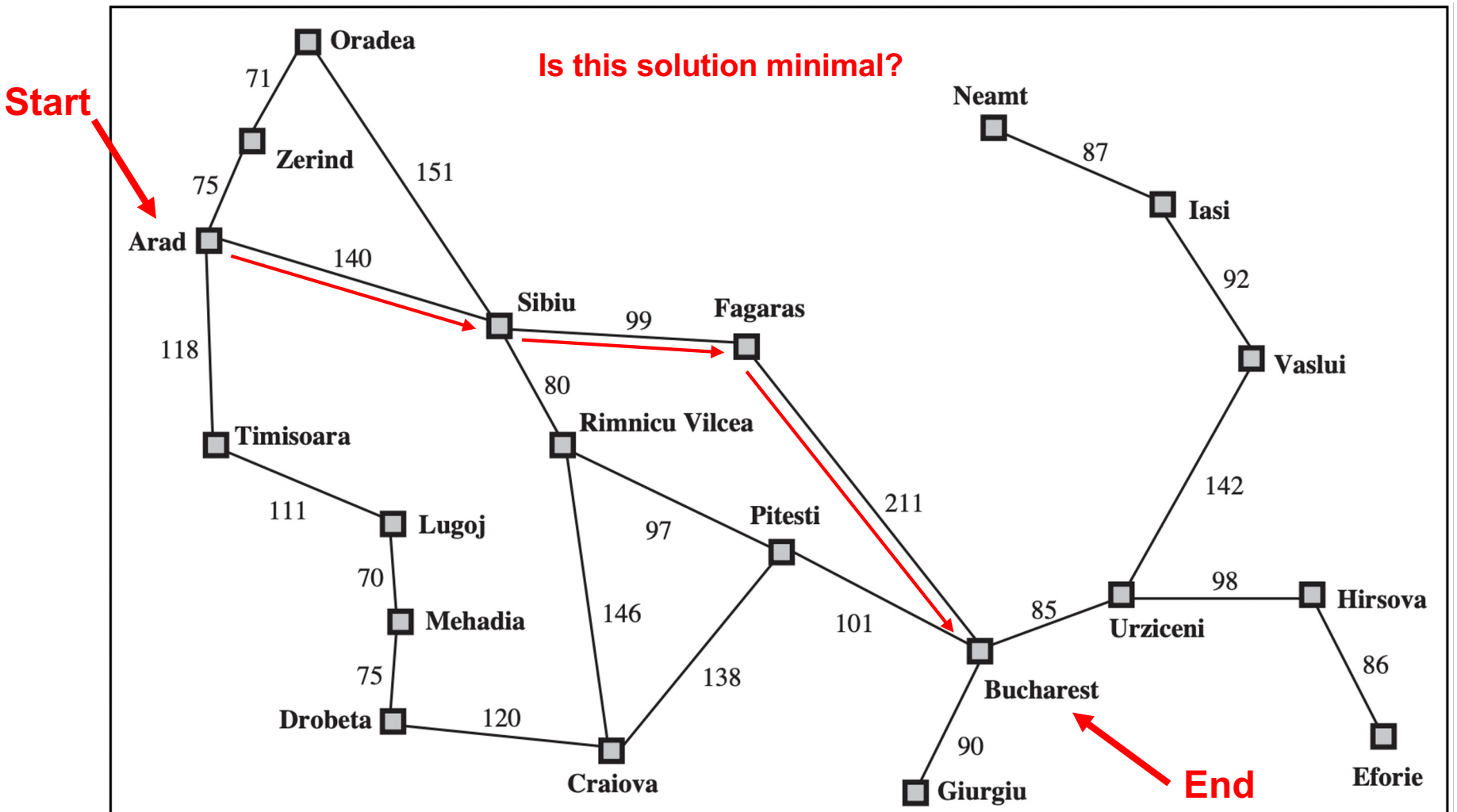


Figure 3.2 A simplified road map of part of Romania. (distances in kilometers)

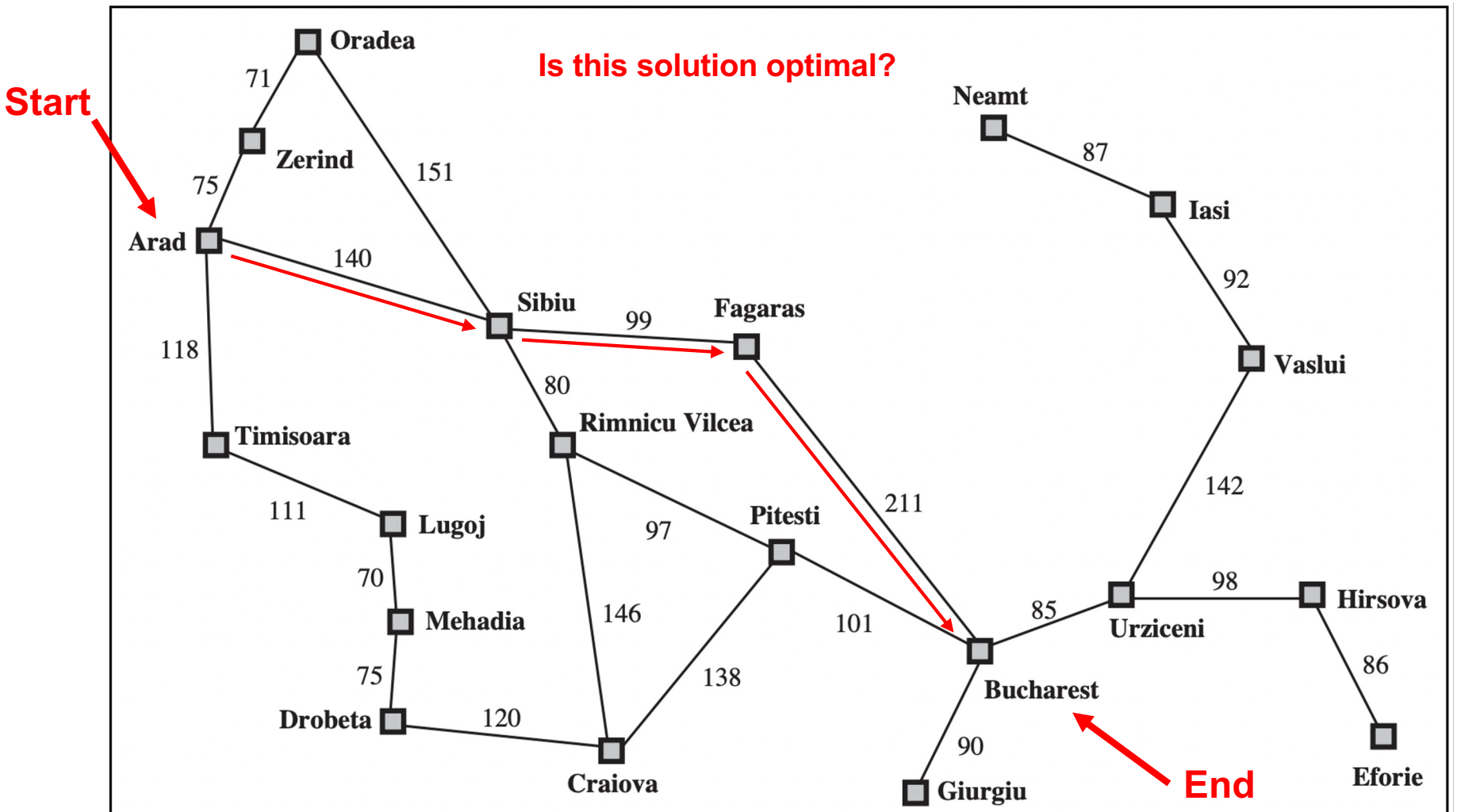


Figure 3.2 A simplified road map of part of Romania. (distances in kilometers)

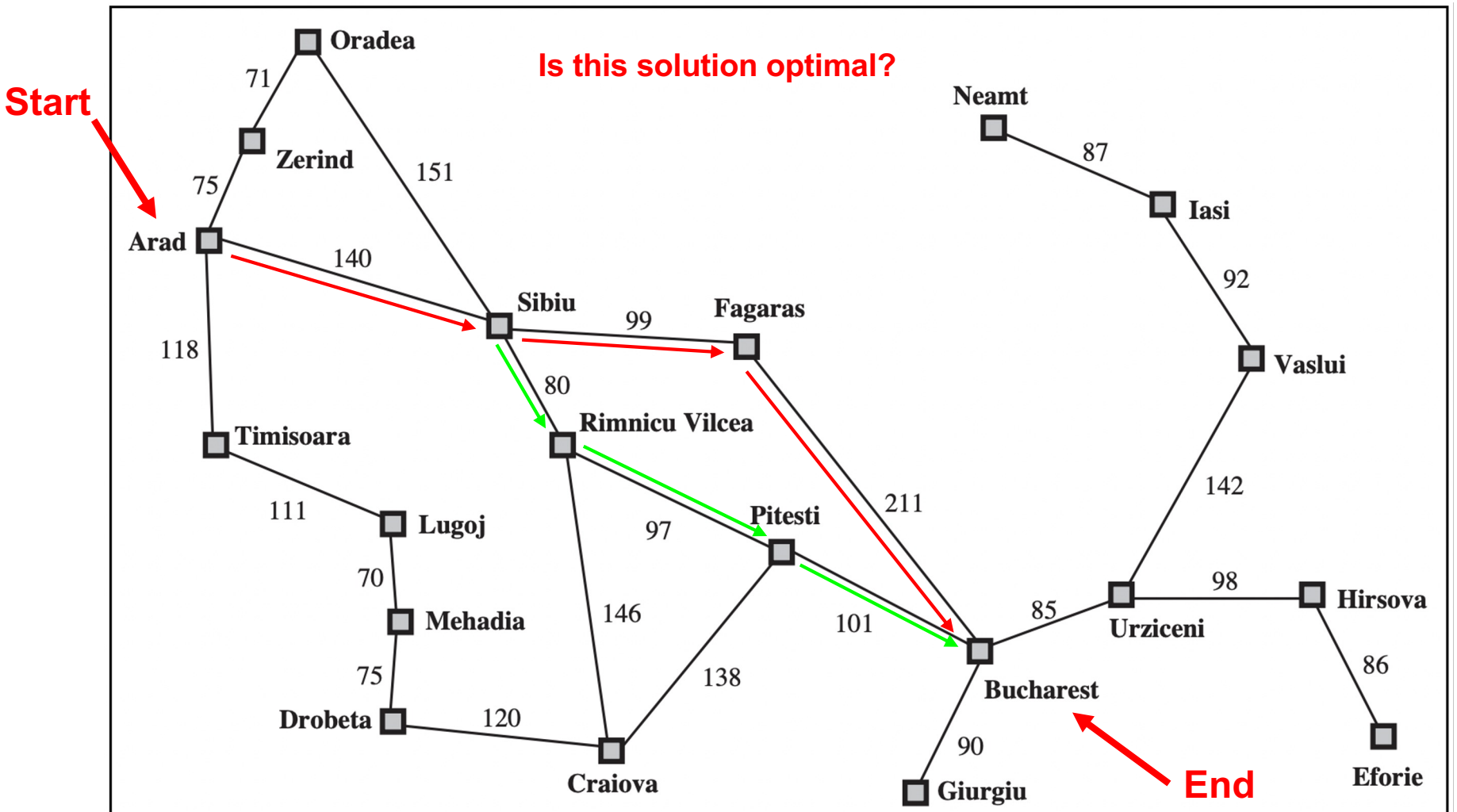
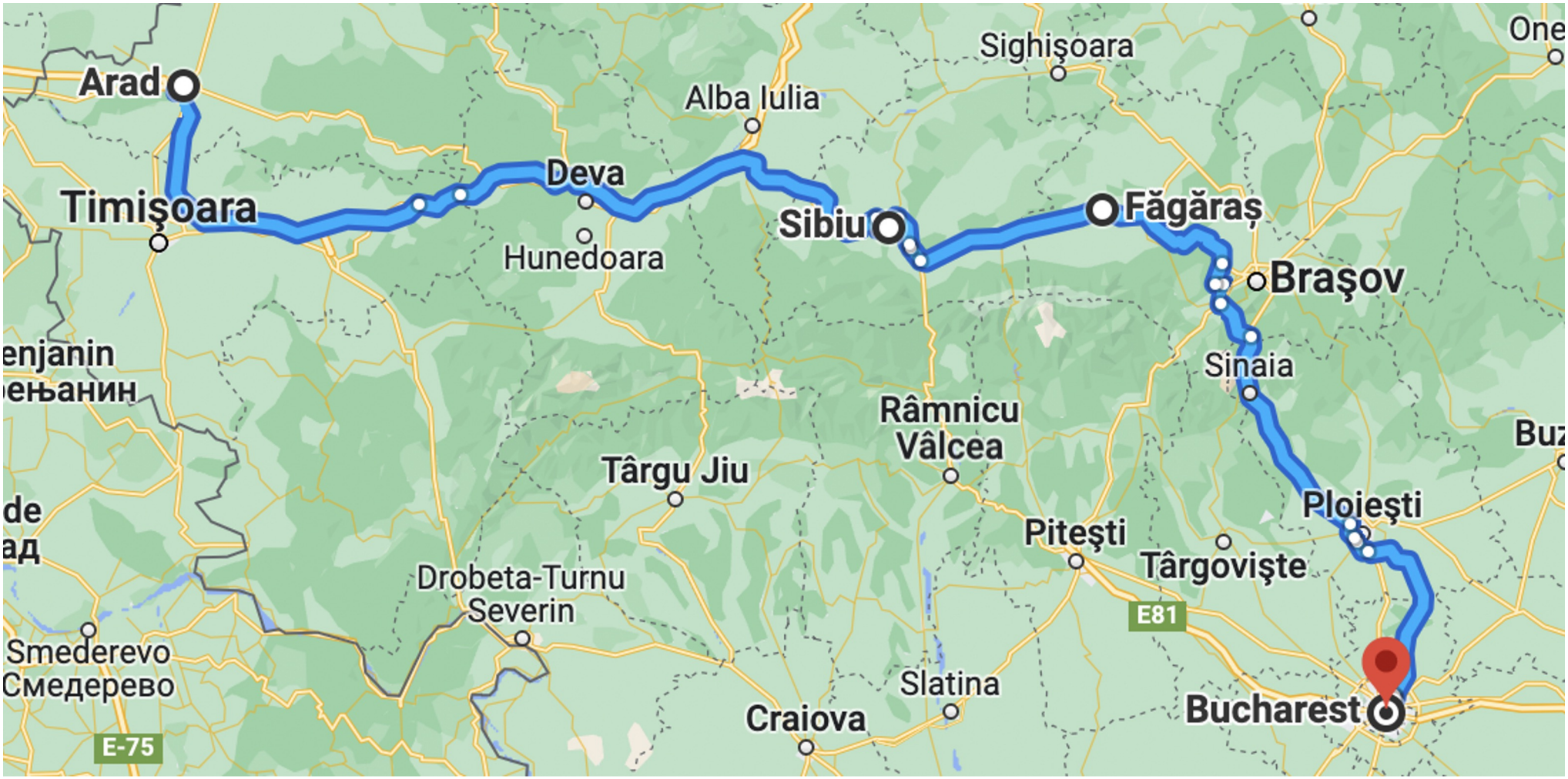


Figure 3.2 A simplified road map of part of Romania. (distances in kilometers)


Arad
Timişoara
Deva
Hunedoara
Alba Iulia
Sighişoara
Sibiu
Făgăraş
Braşov
Sinaia
Ploieşti
Târgovişte
Piteşti
Râmnicu
Vâlcea
Târgu Jiu
Drobeta-Turnu
Severin
Craiova
Slatina
Bucharest
Smederevo
Смедерево
E-75
E81

Greedy best-first search

Greedy best-first search

- Expand the node that is closest to the goal as determined by a heuristic function

Greedy best-first search

- Expand the node that is closest to the goal as determined by a heuristic function
- Assumes that this is likely to lead to a solution quickly

Greedy best-first search

- Expand the node that is closest to the goal as determined by a heuristic function
- Assumes that this is likely to lead to a solution quickly
- The solution from greedy best-first search may not be optimal since a shorter path may exist

Greedy best-first search

- Expand the node that is closest to the goal as determined by a heuristic function
- Assumes that this is likely to lead to a solution quickly
- The solution from greedy best-first search may not be optimal
  since a shorter path may exist
- Search cost is minimal since solution found
  without expanding a node that is not on the solution path

Greedy best-first search

- Expand the node that is closest to the goal as determined by a heuristic function
- Assumes that this is likely to lead to a solution quickly
- The solution from greedy best-first search may not be optimal since a shorter path may exist
- Search cost is minimal since solution found without expanding a node that is not on the solution path
- This algorithm is called “greedy” since at each step it tries to get as close to the goal as it can

Greedy best-first search

- Expand the node that is closest to the goal as determined by a heuristic function
- Assumes that this is likely to lead to a solution quickly
- The solution from greedy best-first search may not be optimal since a shorter path may exist
- Search cost is minimal since solution found without expanding a node that is not on the solution path
- This algorithm is called “greedy” since at each step it tries to get as close to the goal as it can
- Minimal, but not complete since it can lead to dead end

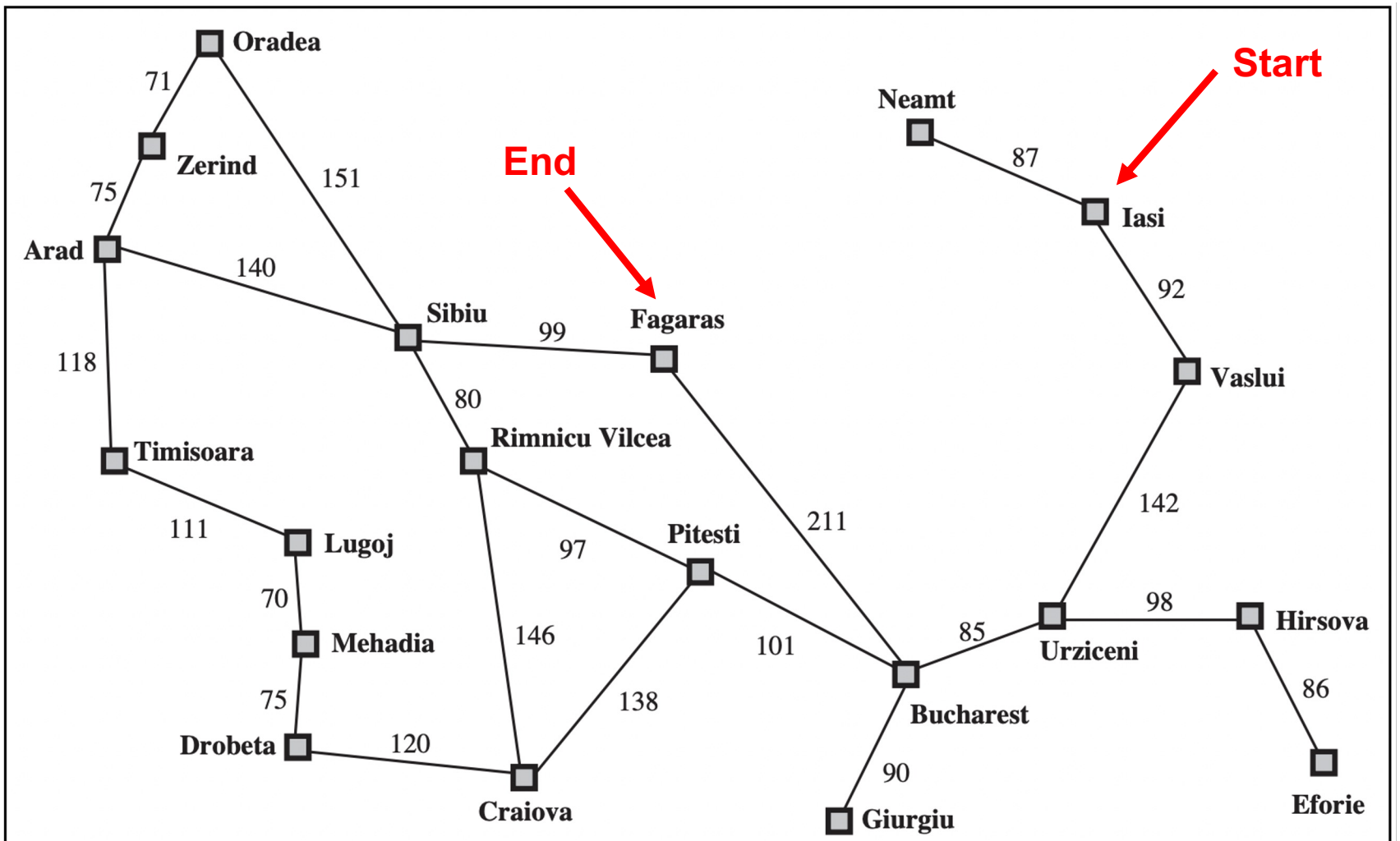


Figure 3.2 A simplified road map of part of Romania.

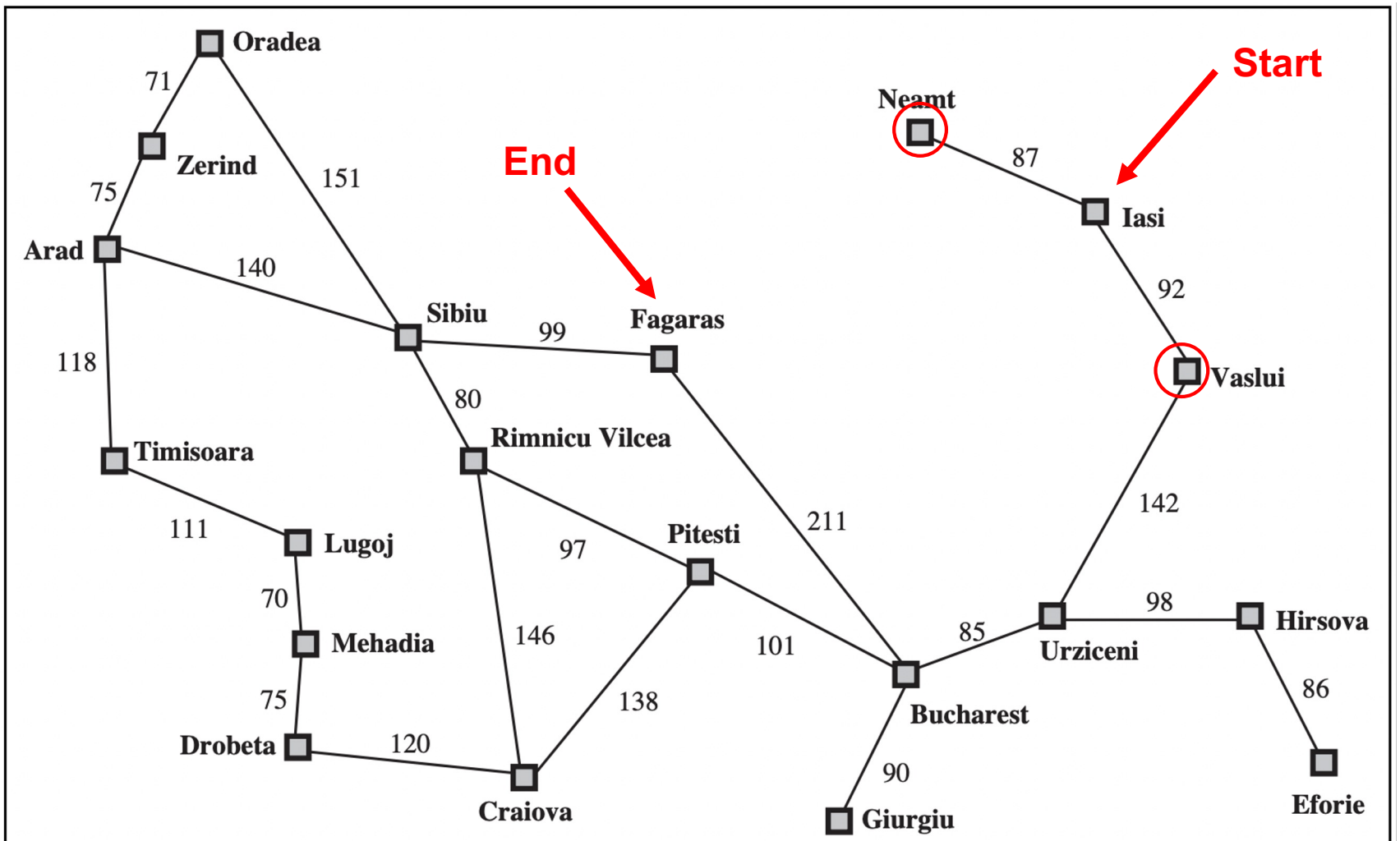


Figure 3.2 A simplified road map of part of Romania.

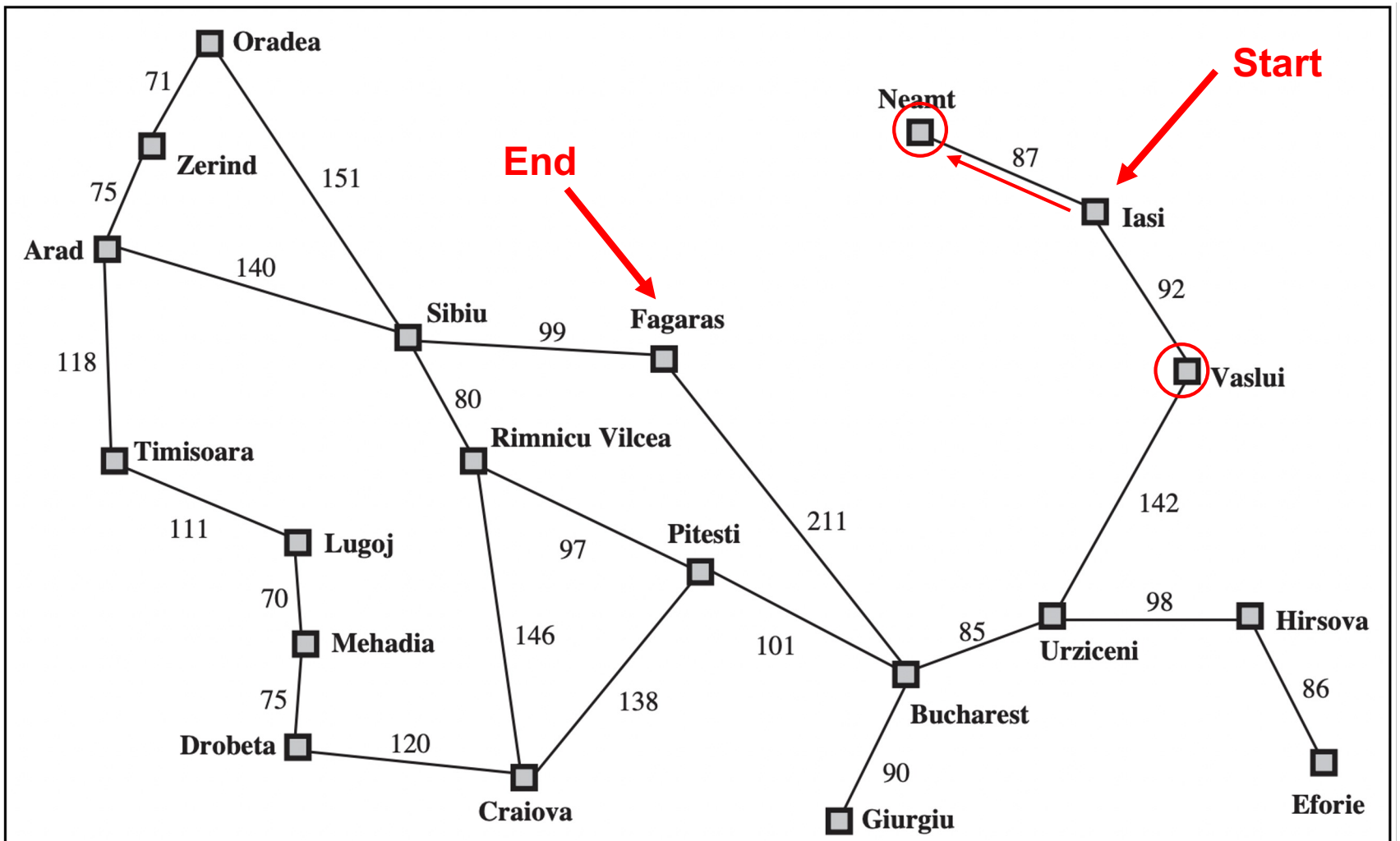


Figure 3.2 A simplified road map of part of Romania.

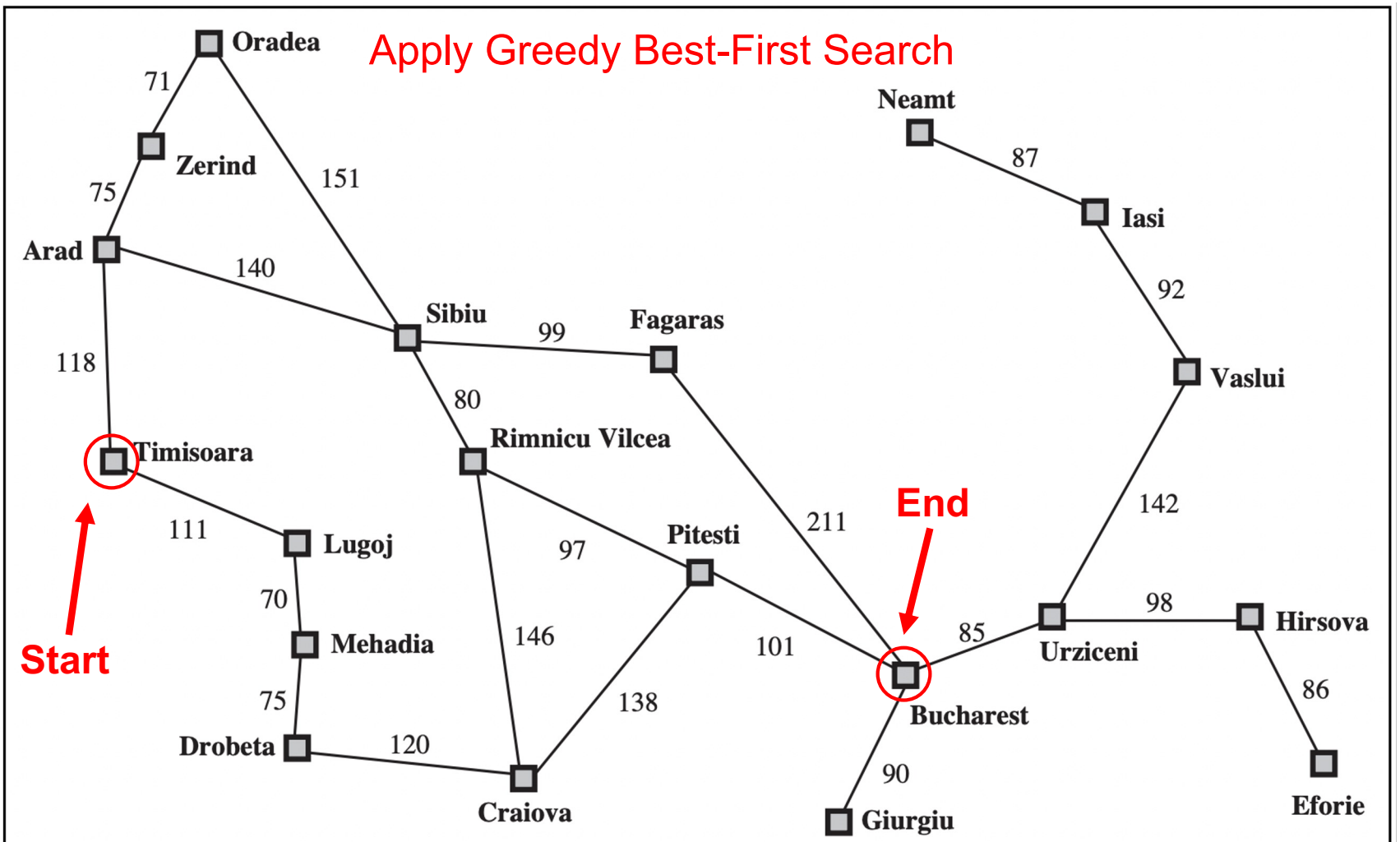


Figure 3.2 A simplified road map of part of Romania.

A* SEARCH

A* search

A* search

- It evaluates nodes by combining g(n), the exact cost to reach the node ‘n’ from the starting point, and h(n), the approximate (heuristic) cost to get from the node ‘n’ to the goal:

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

A* search

- It evaluates nodes by combining g(n), the exact cost to reach the node ‘n’ from the starting point, and h(n), the approximate (heuristic) cost to get from the node ‘n’ to the goal:

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

- g(n) : the exact path cost from the start node to node n

A* search

- It evaluates nodes by combining g(n), the exact cost to reach the node ‘n’ from the starting point, and h(n), the approximate (heuristic) cost to get from the node ‘n’ to the goal:

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

- g(n) : the exact path cost from the start node to node n

- h(n) : the estimated (heuristic) cost of the cheapest path from n to the goal

A* search

- It evaluates nodes by combining g(n), the exact cost to reach the node ‘n’ from the starting point, and h(n), the approximate (heuristic) cost to get from the node ‘n’ to the goal:

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

- g(n) : the exact path cost from the start node to node n

- h(n) : the estimated (heuristic) cost of the cheapest path from n to the goal

- f(n) : estimated cost of the cheapest solution from the starting point to the goal state through node ‘n’

A* search

- It evaluates nodes by combining g(n), the exact cost to reach the node ‘n’ from the starting point, and h(n), the approximate (heuristic) cost to get from the node ‘n’ to the goal:

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

- g(n) : the exact path cost from the start node to node n
- h(n) : the estimated (heuristic) cost of the cheapest path from n to the goal
- f(n) : estimated cost of the cheapest solution from the starting point to the goal state through node ‘n’
- Provided that the heuristic function h(n) satisfies certain conditions, $A*$ search is both complete and optimal

A* search

- It evaluates nodes by combining g(n), the exact cost to reach the node ‘n’ from the starting point, and h(n), the approximate (heuristic) cost to get from the node ‘n’ to the goal:

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

- g(n) : the exact path cost from the start node to node n
- h(n) : the estimated (heuristic) cost of the cheapest path from n to the goal
- f(n) : estimated cost of the cheapest solution from the starting point to the goal state through node ‘n’
- Provided that the heuristic function h(n) satisfies certain conditions, $A*$ search is both complete and optimal
  - h(n) should be an admissible heuristic, i.e. it never overestimates the cost to reach the goal.

A* search

- It evaluates nodes by combining g(n), the exact cost to reach the node ‘n’ from the starting point, and h(n), the approximate (heuristic) cost to get from the node ‘n’ to the goal:

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

- g(n) : the exact path cost from the start node to node n

- h(n) : the estimated (heuristic) cost of the cheapest path from n to the goal

- f(n) : estimated cost of the cheapest solution from the starting point to the goal state through node ‘n’

- Provided that the heuristic function h(n) satisfies certain conditions, A∗ search is both complete and optimal

  - h(n) should be an admissible heuristic, i.e. it never overestimates the cost to reach the goal. (h_SLD is a good example since straight line distance will always be lower than actual road distance)

A* search

- It evaluates nodes by combining g(n), the exact cost to reach the node ‘n’ from the starting point, and h(n), the approximate (heuristic) cost to get from the node ‘n’ to the goal:

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

- g(n) : the exact path cost from the start node to node n

- h(n) : the estimated (heuristic) cost of the cheapest path from n to the goal

- f(n) : estimated cost of the cheapest solution from the starting point to the goal state through node ‘n’

- Provided that the heuristic function h(n) satisfies certain conditions, A∗ search is both complete and optimal

  - h(n) should be an admissible heuristic, i.e. it never overestimates the cost to reach the goal. (h_SLD is a good example since straight line distance will always be lower than actual road distance)

  - h(n) must be consistent, i.e. it satisfies the triangle inequality (required for graphs)

A* search

- It evaluates nodes by combining g(n), the exact cost to reach the node ‘n’ from the starting point, and h(n), the approximate (heuristic) cost to get from the node ‘n’ to the goal:

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

- g(n) : the exact path cost from the start node to node n

- h(n) : the estimated (heuristic) cost of the cheapest path from n to the goal

- f(n) : estimated cost of the cheapest solution from the starting point to the goal state through node ‘n’

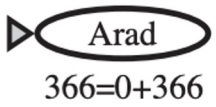
- Provided that the heuristic function h(n) satisfies certain conditions, A∗ search is both complete and optimal

  - h(n) should be an admissible heuristic, i.e. it never overestimates the cost to reach the goal. (h_SLD is a good example since straight line distance will always be lower than actual road distance)

  - h(n) must be consistent, i.e. it satisfies the triangle inequality (required for graphs) [All consistent heuristics are admissible, but reverse is not true]

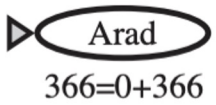
f(n) = g(n) + h(n)

(a) The initial state

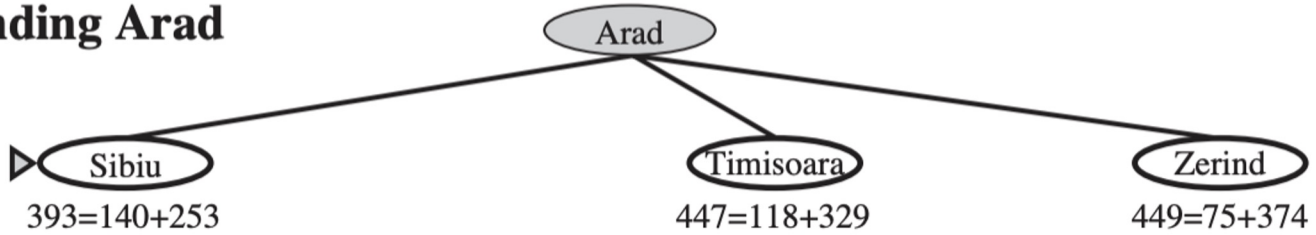


f(n) = g(n) + h(n)

(a) The initial state

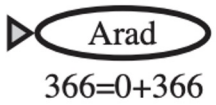


(b) After expanding Arad

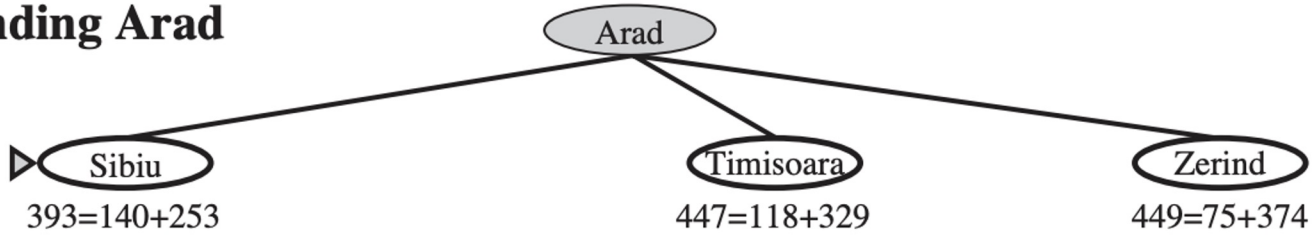


f(n) = g(n) + h(n)

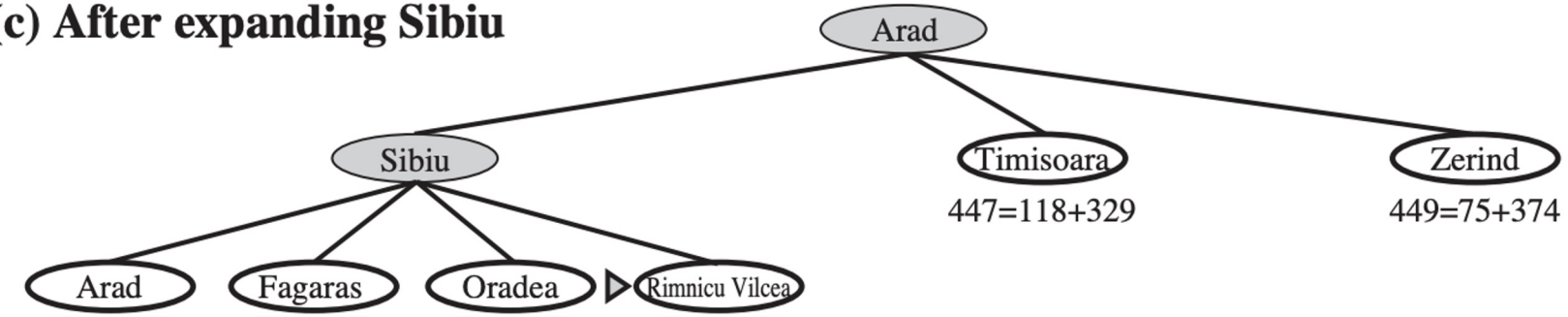
(a) The initial state



(b) After expanding Arad

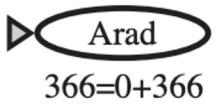


(c) After expanding Sibiu

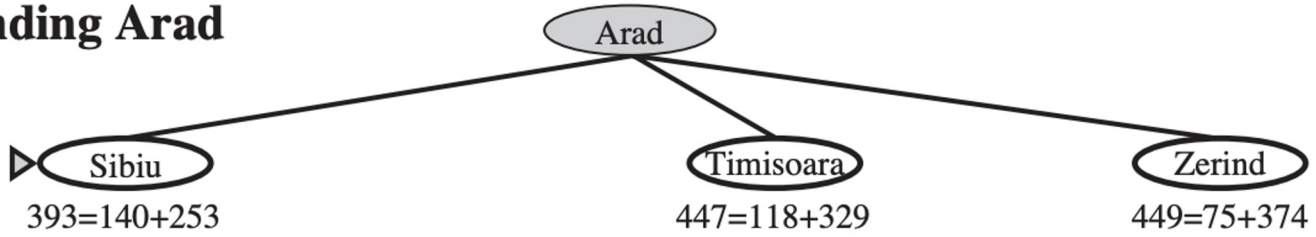


f(n) = g(n) + h(n)

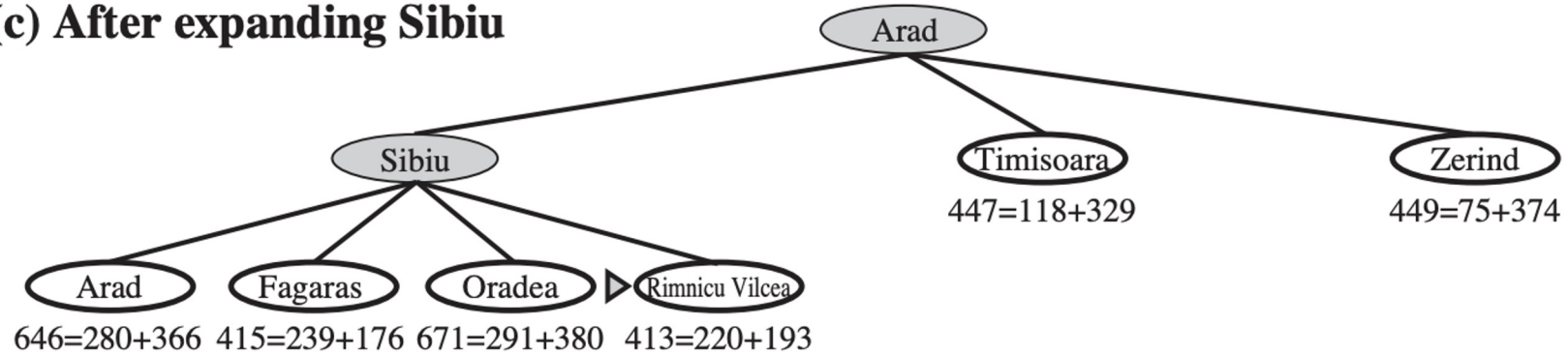
(a) The initial state



(b) After expanding Arad

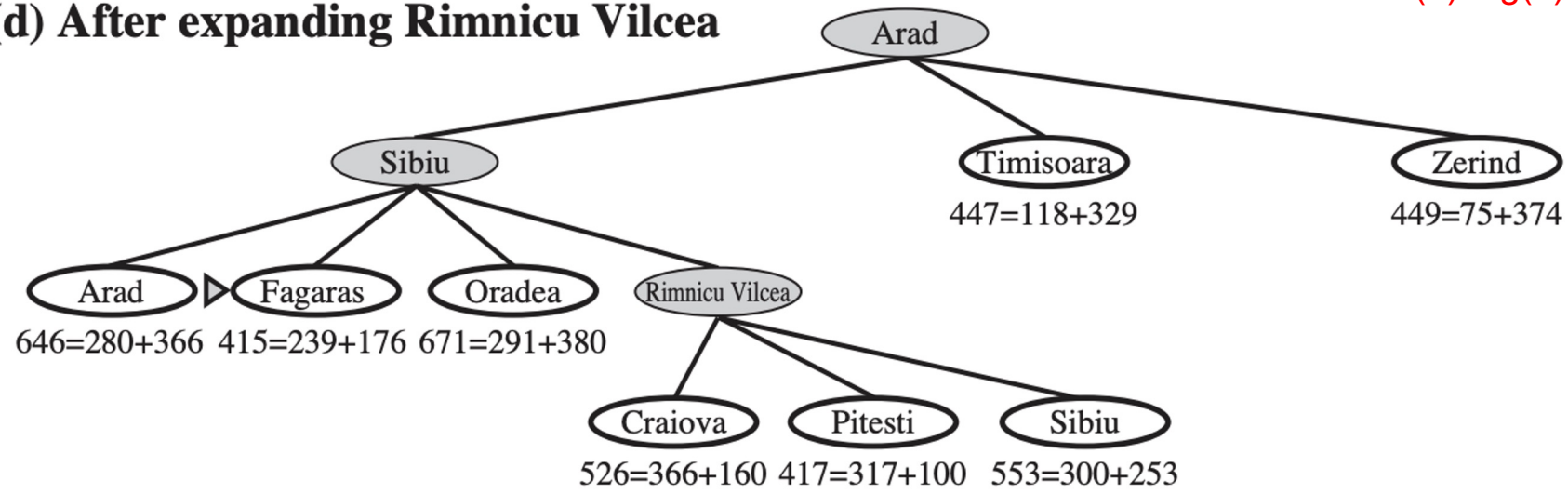


(c) After expanding Sibiu



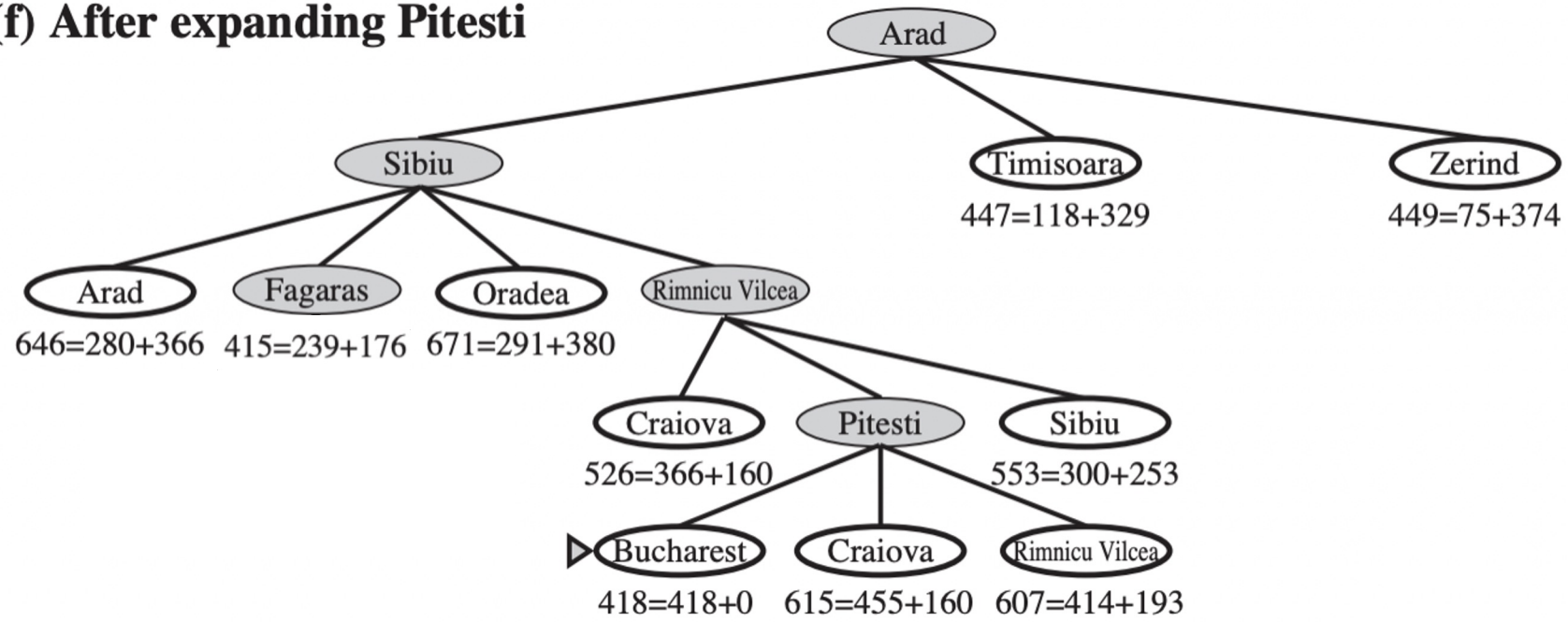
f(n) = g(n) + h(n)

(d) After expanding Rimnicu Vilcea



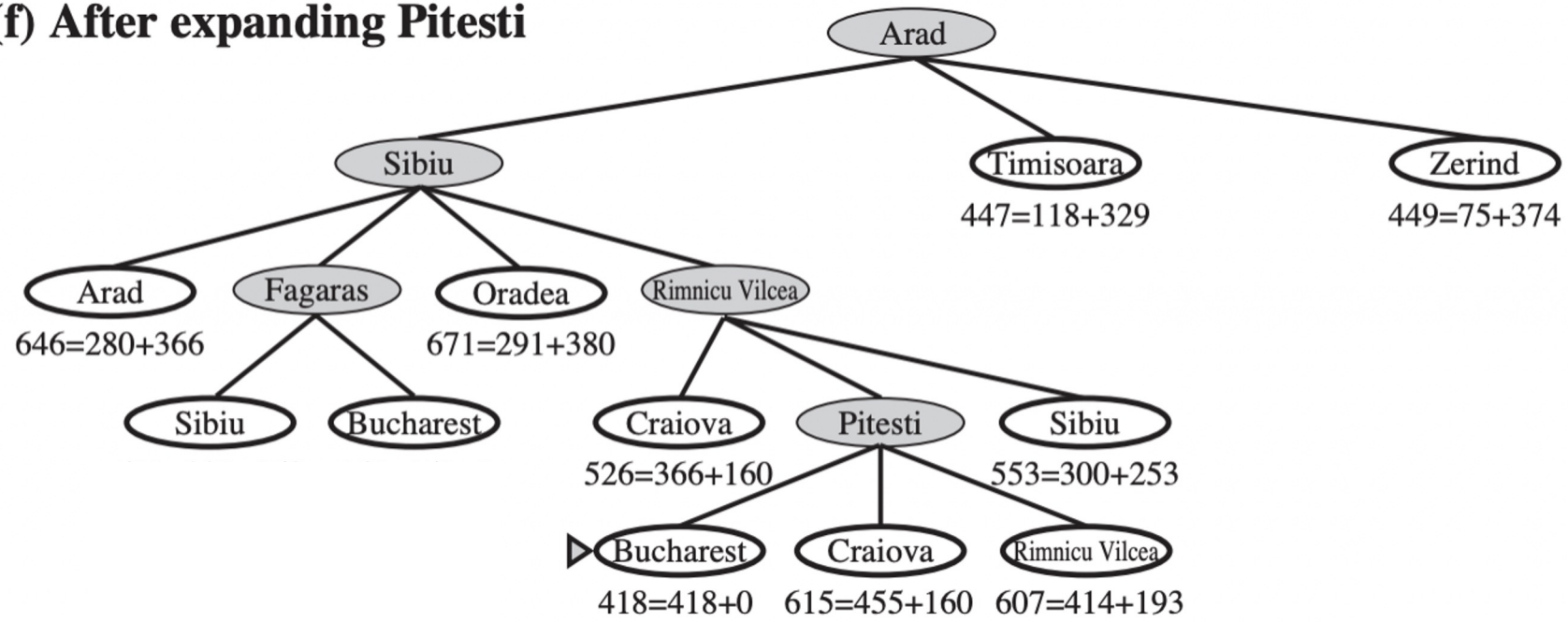
f(n) = g(n) + h(n)

(f) After expanding Pitesti



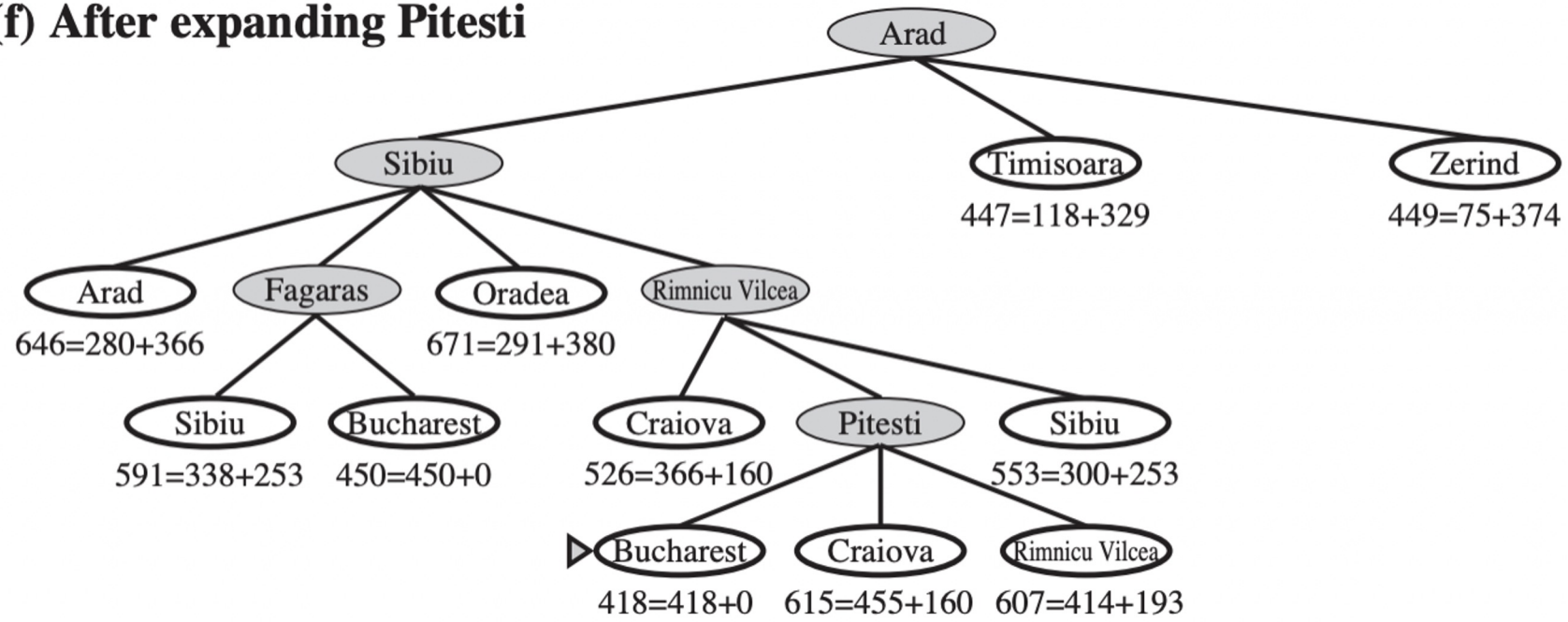
f(n) = g(n) + h(n)

(f) After expanding Pitesti



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

(f) After expanding Pitesti



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

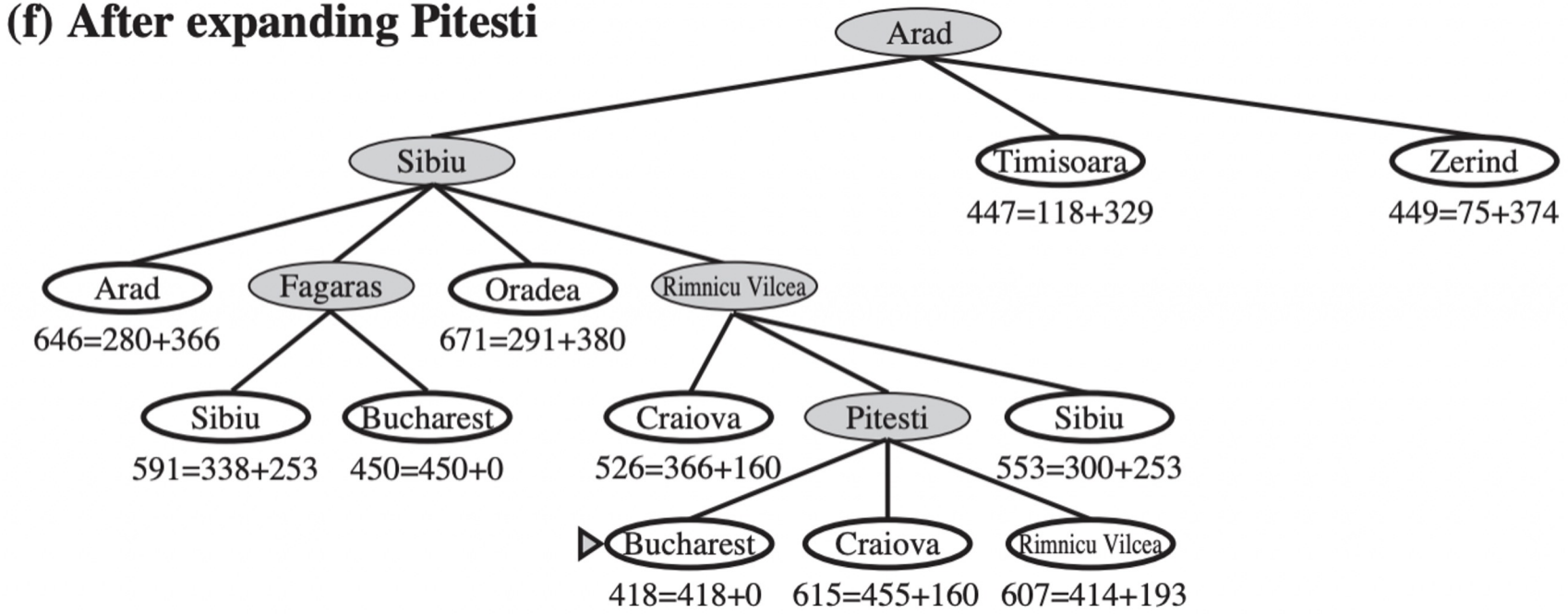


Figure 3.24 Stages in an A* search for Bucharest. Nodes are labeled with $f = g + h$. The h values are the straight-line distances to Bucharest taken from Figure 3.22.

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

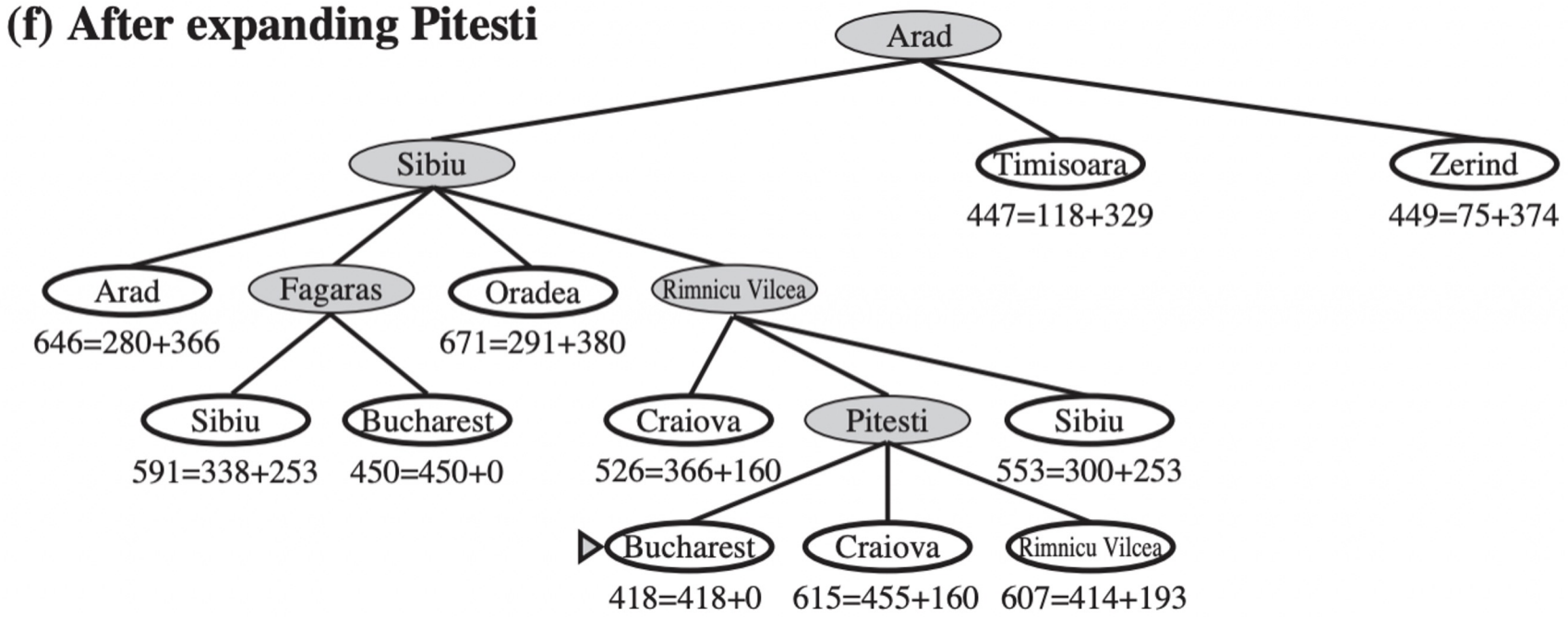
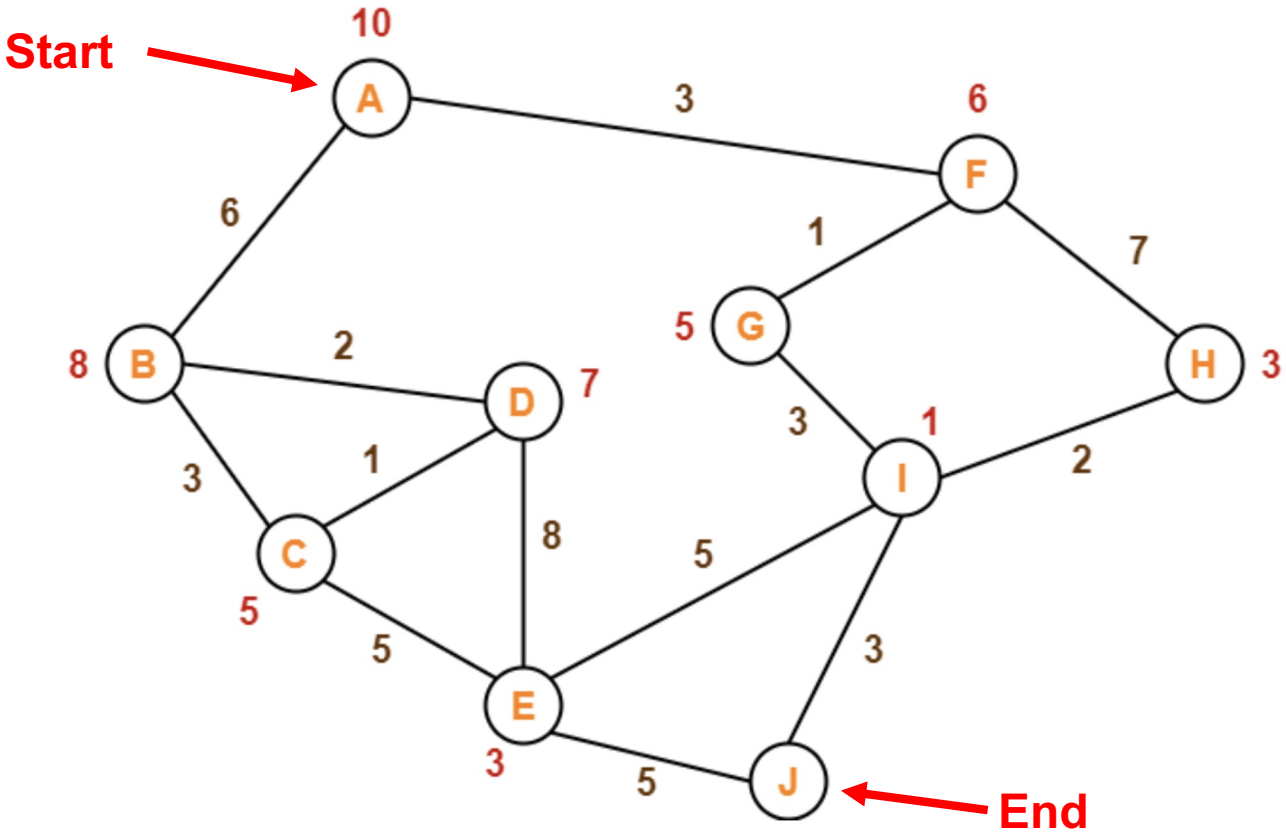


Figure 3.24 Stages in an A* search for Bucharest. Nodes are labeled with $f = g + h$. The h values are the straight-line distances to Bucharest taken from Figure 3.22.

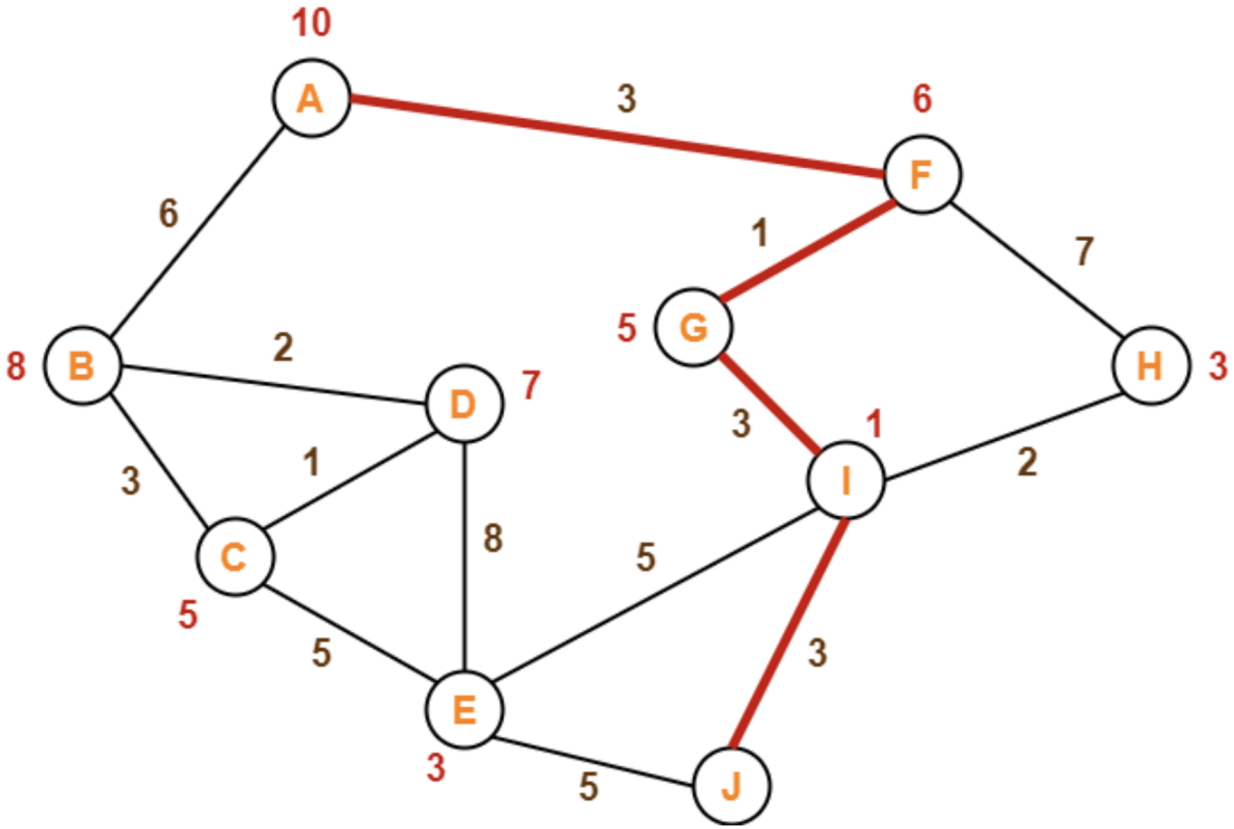
A* is complete and optimal,
but has very high space complexity!



The numbers written on edges represent the distance between the nodes.

The numbers written on nodes represent the heuristic value.

Find the most cost-effective path to reach from start state A to final state J using A* Algorithm.


10
A
3
6
F
6
1
7
5
G
H
3
8
B
2
D
7
3
1
3
1
I
2
C
8
5
5
5
3
E
3
5
J

Given an initial state of a 8-puzzle problem and final state to be reached-

Initial State

Final State

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

Find the most cost-effective path to reach the final state from initial state using A* Algorithm.

Given an initial state of a 8-puzzle problem and final state to be reached-

Initial State

Final State

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

Find the most cost-effective path to reach the final state from initial state using A* Algorithm.

Consider g(n) = Depth of node

Given an initial state of a 8-puzzle problem and final state to be reached-

Initial State

Final State

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

Find the most cost-effective path to reach the final state from initial state using A* Algorithm.

Consider g(n) = Depth of node and h(n) = Number of misplaced tiles.

1	2	3
8		4
7	6	5

Final State

Initial State

2	8	3
1	6	4
7		5

1	2	3
8		4
7	6	5

Final State

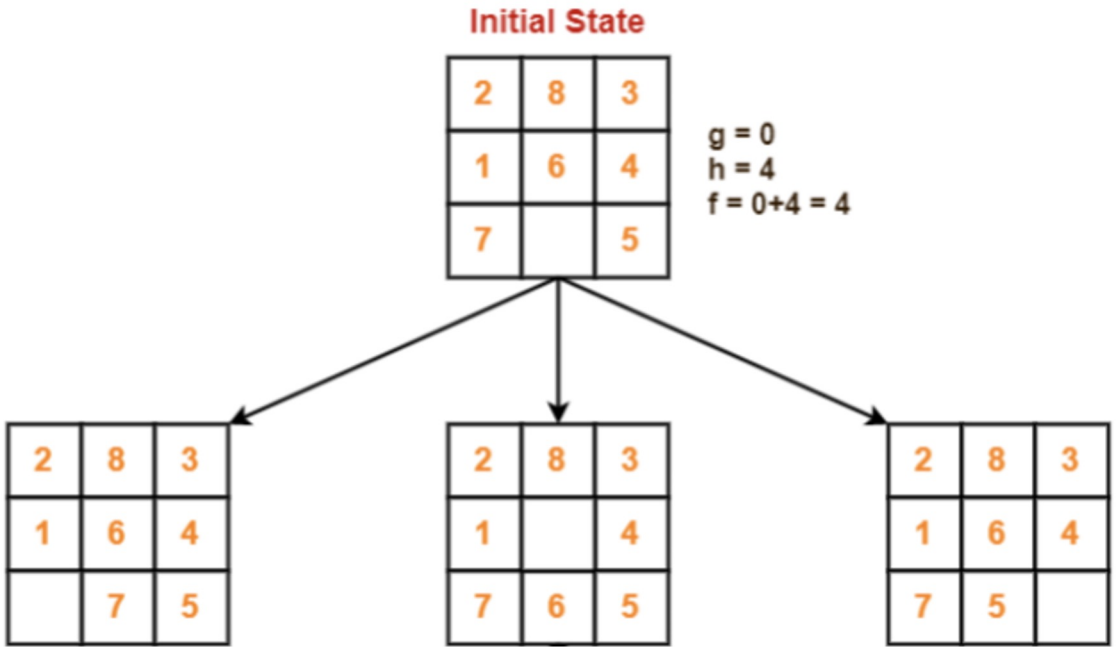
Initial State

2	8	3
1	6	4
7		5

g = 0
h = 4
f = 0+4 = 4

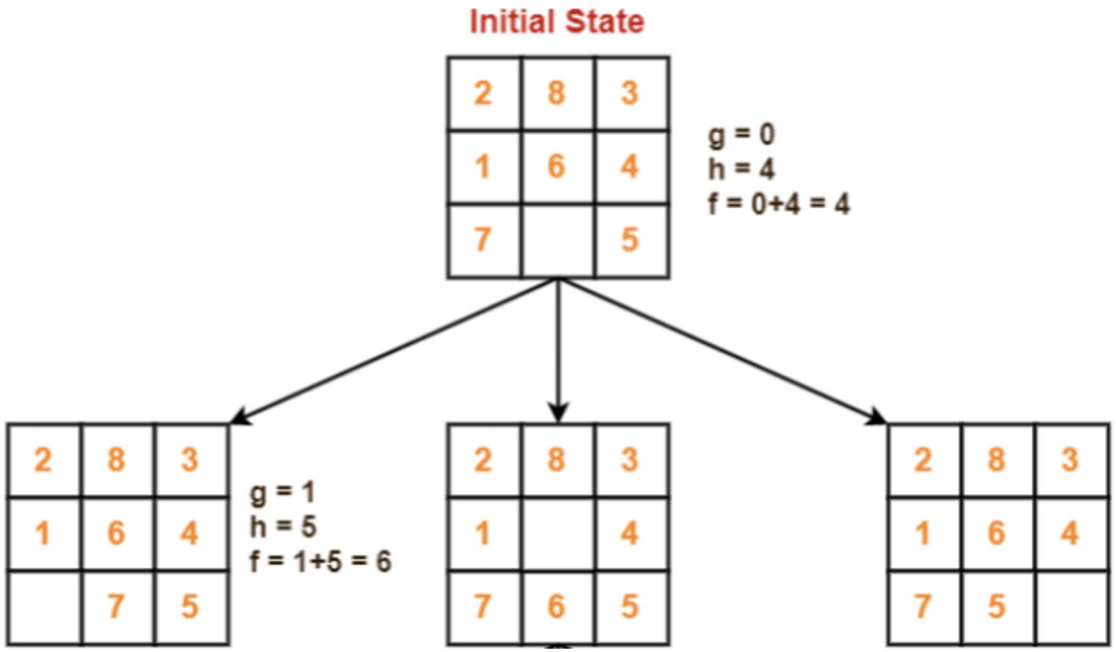

1 2 3
8 4
7 6 5

Final State


Initial State
2 8 3
1 6 4
7 5
g = 0
h = 4
f = 0+4 = 4
2 8 3
1 6 4
7 5
2 8 3
1 4
7 6 5
2 8 3
1 6 4
7 5

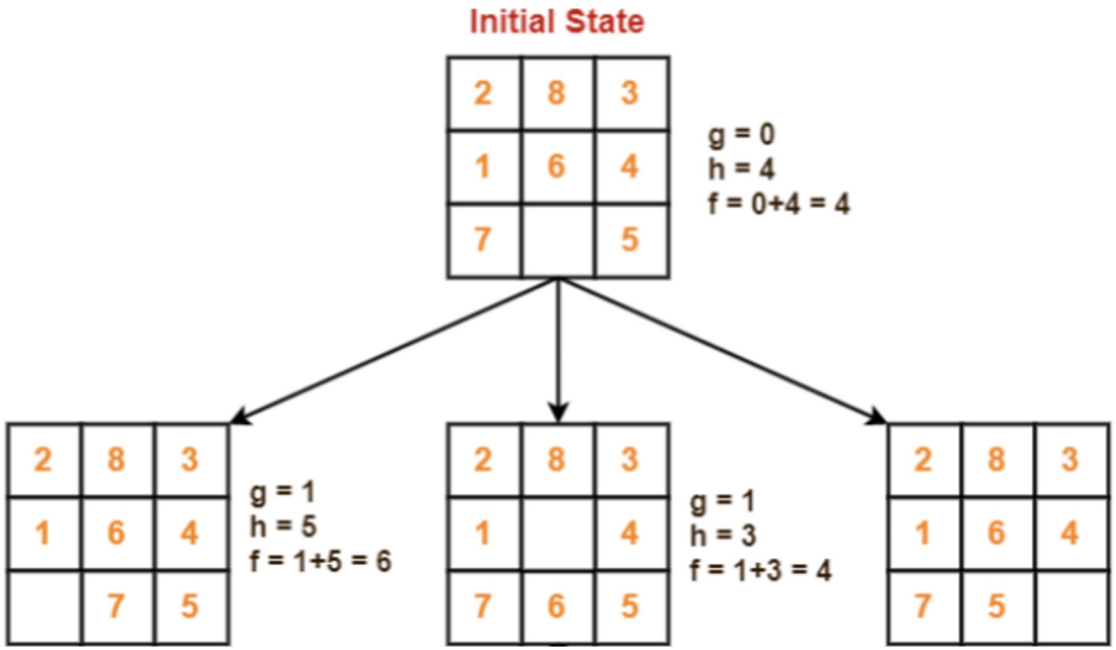
1	2	3
8		4
7	6	5

Final State



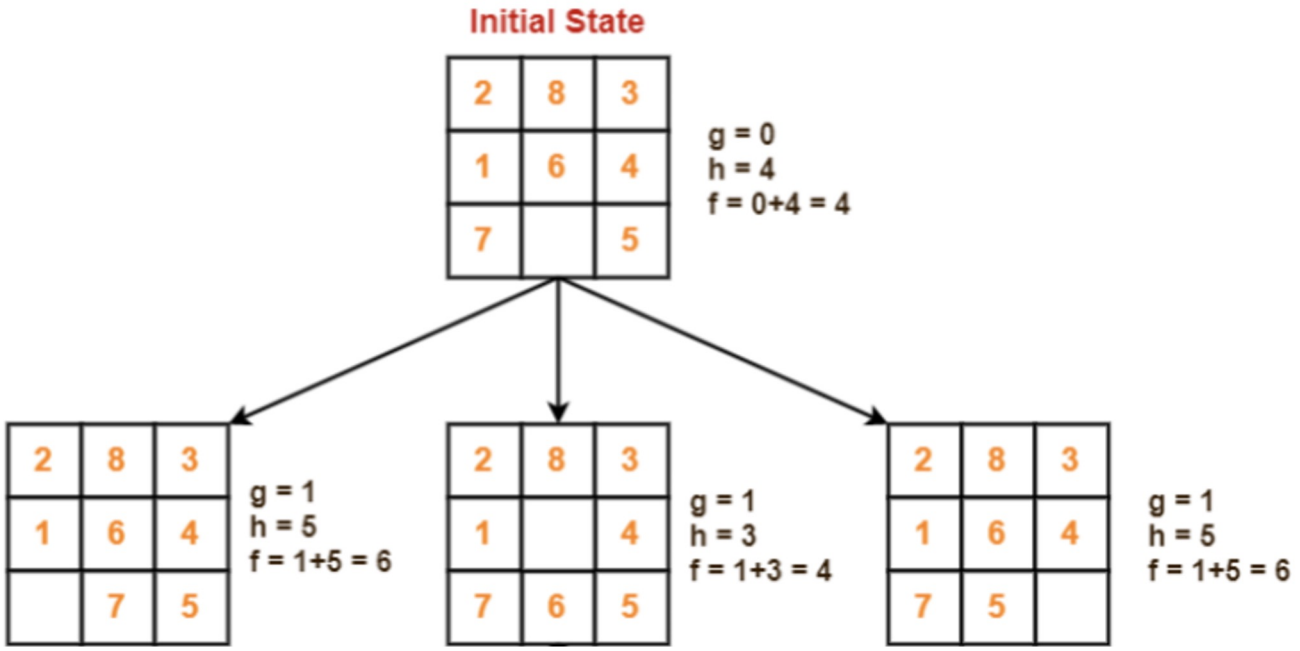
1	2	3
8		4
7	6	5

Final State



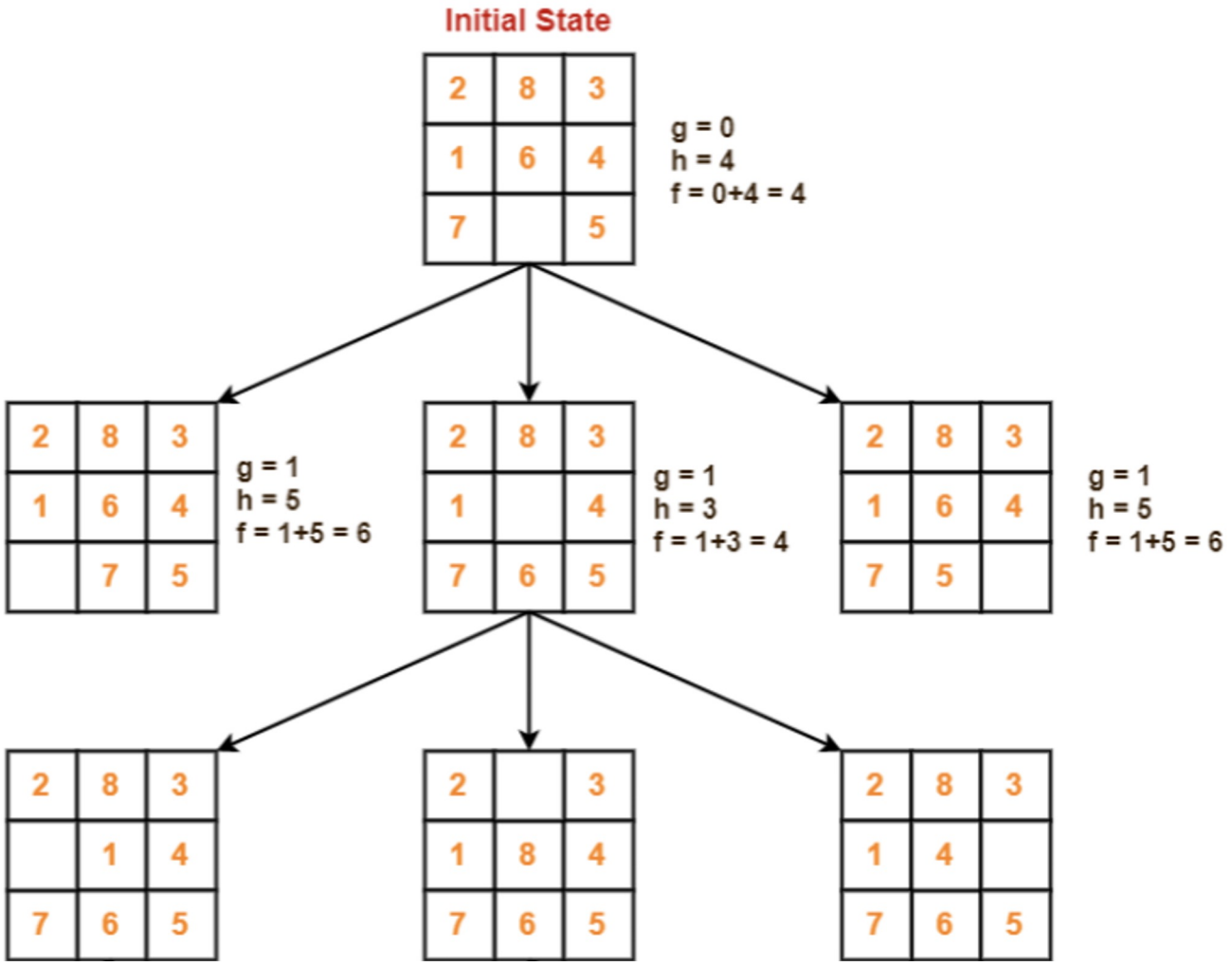

1 2 3
8 4
7 6 5

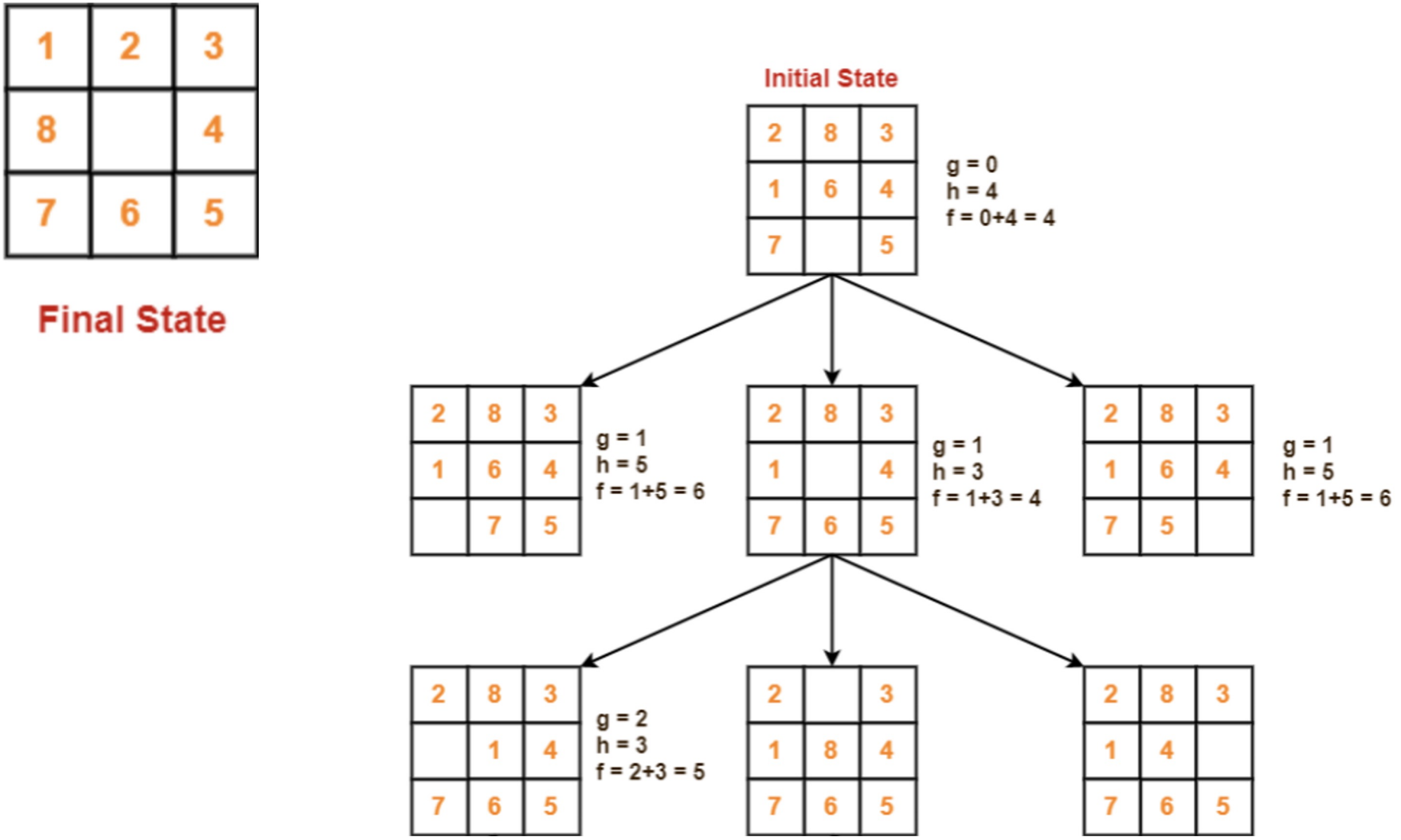
Final State

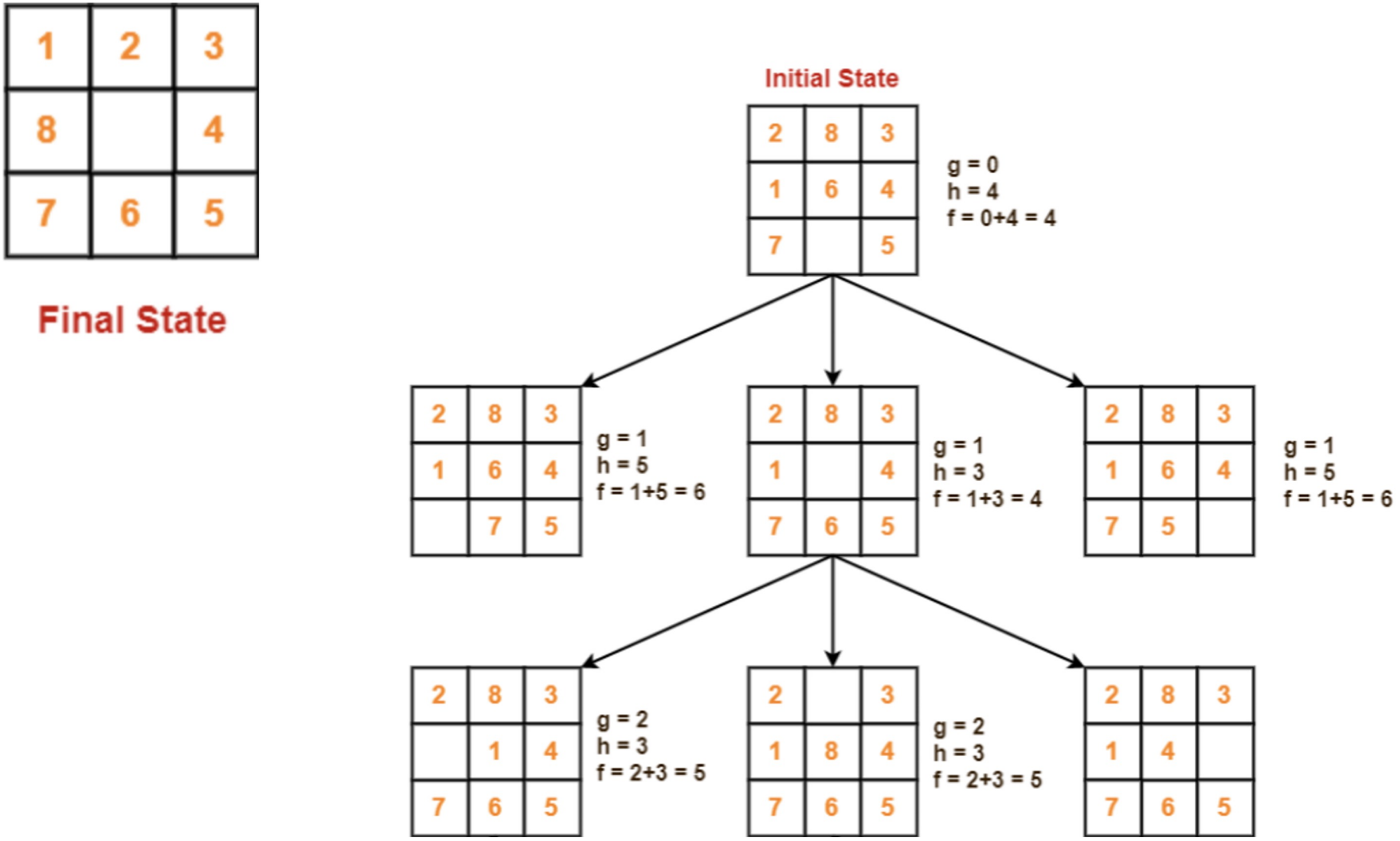

Initial State
2 8 3
1 6 4
7 5
g = 0
h = 4
f = 0+4 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 1
h = 3
f = 1+3 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6

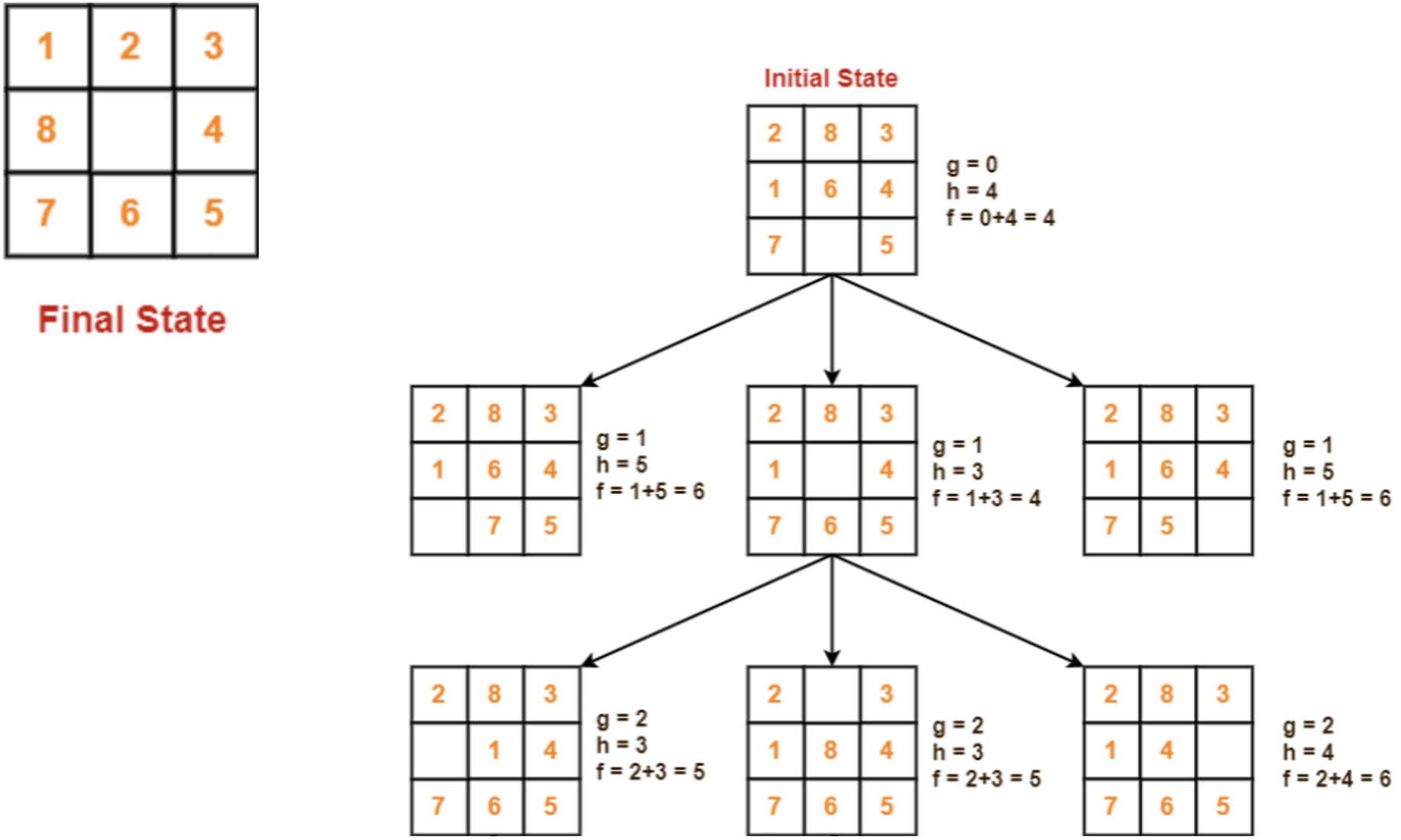

1 2 3
8 4
7 6 5

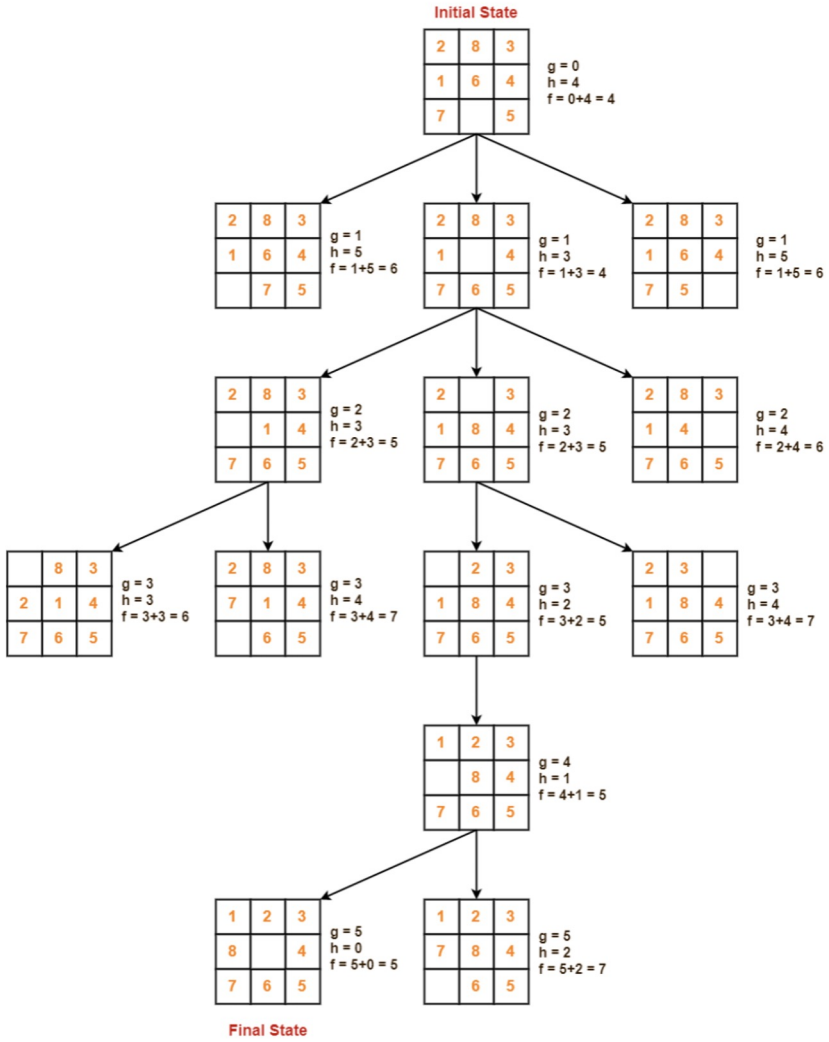
Final State


Initial State
2 8 3
1 6 4
7 5
g = 0
h = 4
f = 0+4 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 1
h = 3
f = 1+3 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
2 3
1 8 4
7 6 5
2 8 3
1 4
7 6 5


1 2 3
8 4
7 6 5
Final State
Initial State
2 8 3
1 6 4
7 5
g = 0
h = 4
f = 0+4 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 1
h = 3
f = 1+3 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 2
h = 3
f = 2+3 = 5
2 3
1 8 4
7 6 5
2 8 3
1 4
7 6 5


1 2 3
8 4
7 6 5
Final State
Initial State
2 8 3
1 6 4
7 5
g = 0
h = 4
f = 0+4 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 1
h = 3
f = 1+3 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 2
h = 3
f = 2+3 = 5
2 3
1 8 4
7 6 5
g = 2
h = 3
f = 2+3 = 5
2 8 3
1 4
7 6 5


1 2 3
8 4
7 6 5
Final State
Initial State
2 8 3
1 6 4
7 5
g = 0
h = 4
f = 0+4 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 1
h = 3
f = 1+3 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 2
h = 3
f = 2+3 = 5
2 3
1 8 4
7 6 5
g = 2
h = 3
f = 2+3 = 5
2 8 3
1 4
7 6 5
g = 2
h = 4
f = 2+4 = 6


Initial State
2 8 3
1 6 4
7 5
g = 0
h = 4
f = 0+4 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 1
h = 3
f = 1+3 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 2
h = 3
f = 2+3 = 5
2 3
1 8 4
7 6 5
g = 2
h = 3
f = 2+3 = 5
2 8 3
1 4
7 6 5
g = 2
h = 4
f = 2+4 = 6
8 3
2 1 4
7 6 5
g = 3
h = 3
f = 3+3 = 6
2 8 3
7 1 4
6 5
g = 3
h = 4
f = 3+4 = 7
2 3
1 8 4
7 6 5
g = 3
h = 2
f = 3+2 = 5
2 3
1 8 4
7 6 5
g = 3
h = 4
f = 3+4 = 7
1 2 3
8 4
7 6 5
g = 4
h = 1
f = 4+1 = 5
1 2 3
8 4
7 6 5
g = 5
h = 0
f = 5+0 = 5
1 2 3
7 8 4
6 5
g = 5
h = 2
f = 5+2 = 7
Final State

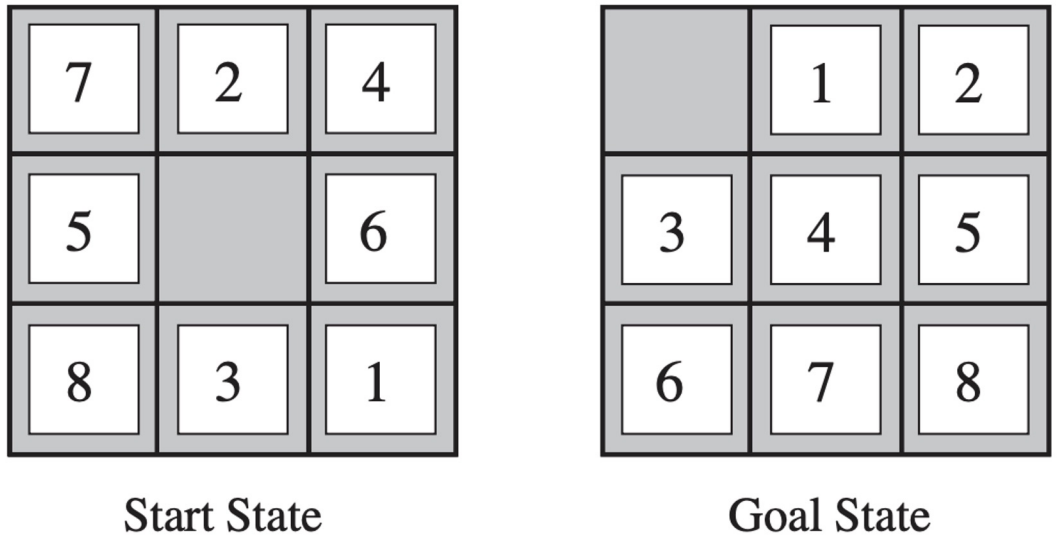


Figure 3.28 A typical instance of the 8-puzzle. The solution is 26 steps long.