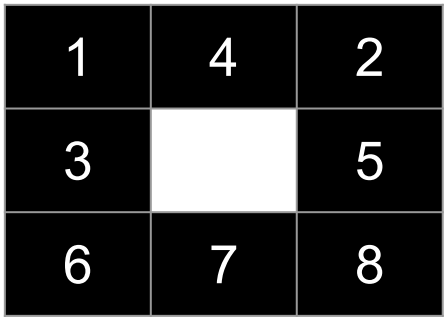
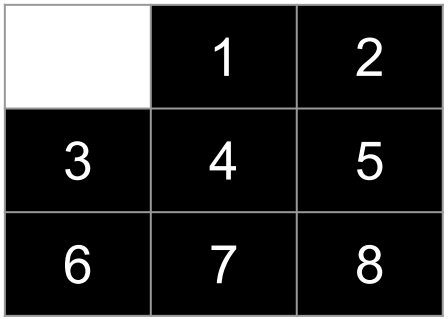
Example: 8-Puzzle Problem

- Solve an 8-Puzzle problem using BFS and DFS

  (Searching process and total steps?)



Start

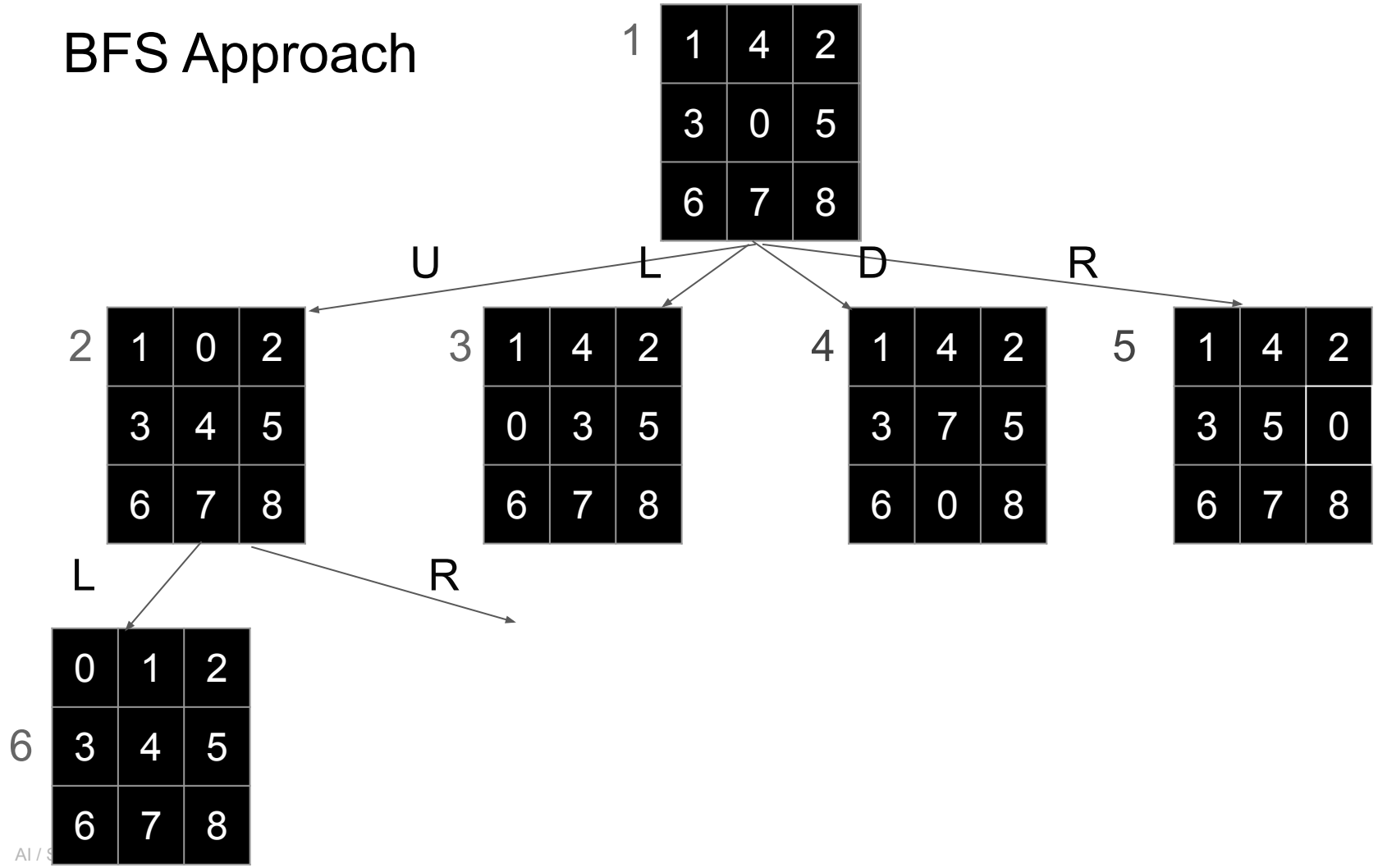


Goal

Breadth-First Search Algorithm

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

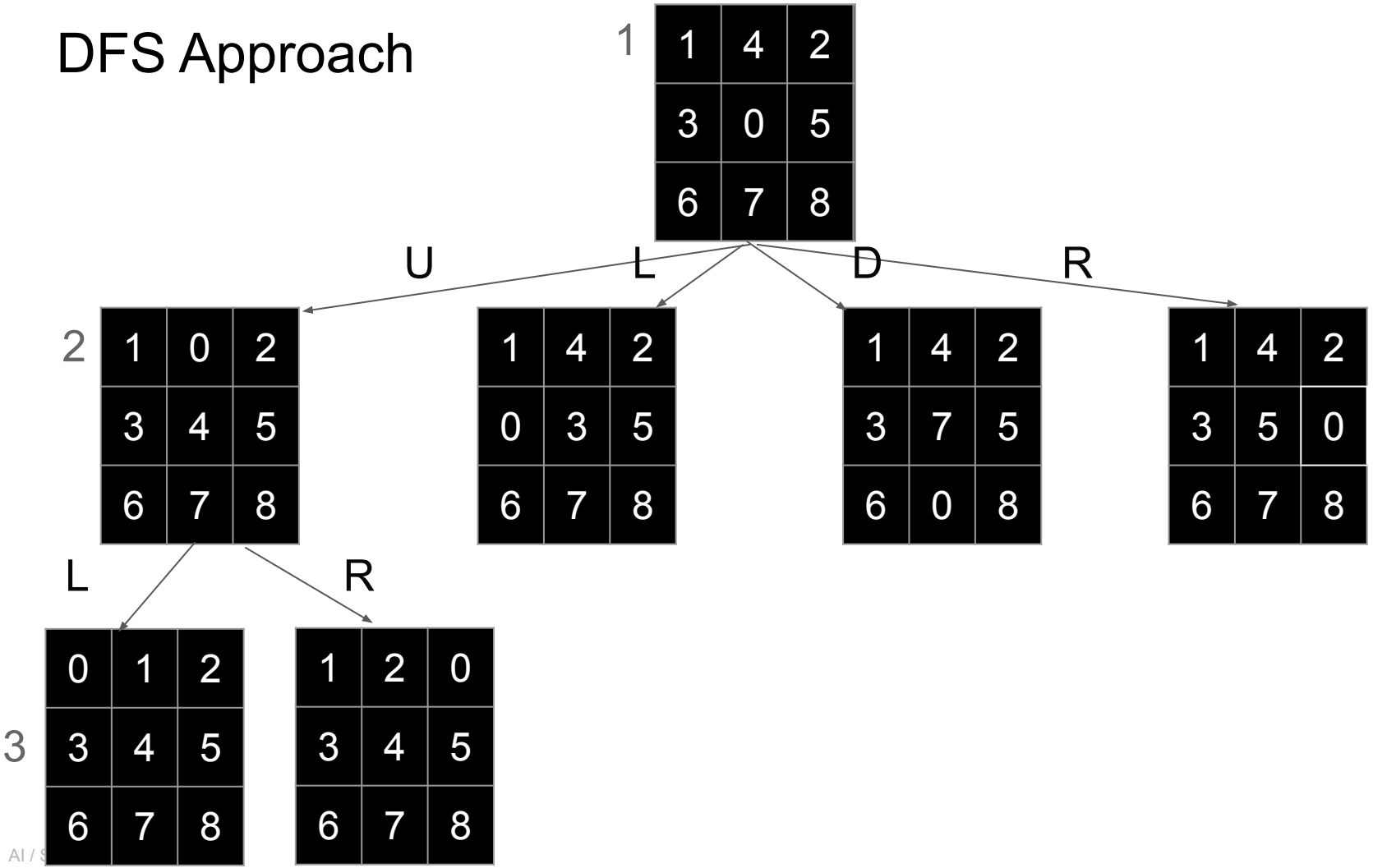
BFS Approach



Depth-Limited Search (DLS) Algorithm

```
function DEPTH-LIMITED-SEARCH(problem, ℓ) returns a node or failure or cutoff
  frontier ← a LIFO queue (stack) with NODE(problem.INITIAL) as an element
  result ← failure
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    if DEPTH(node) > ℓ then
      result ← cutoff
    else if not IS-CYCLE(node) do
      for each child in EXPAND(problem, node) do
        add child to frontier
  return result
```

DFS Approach

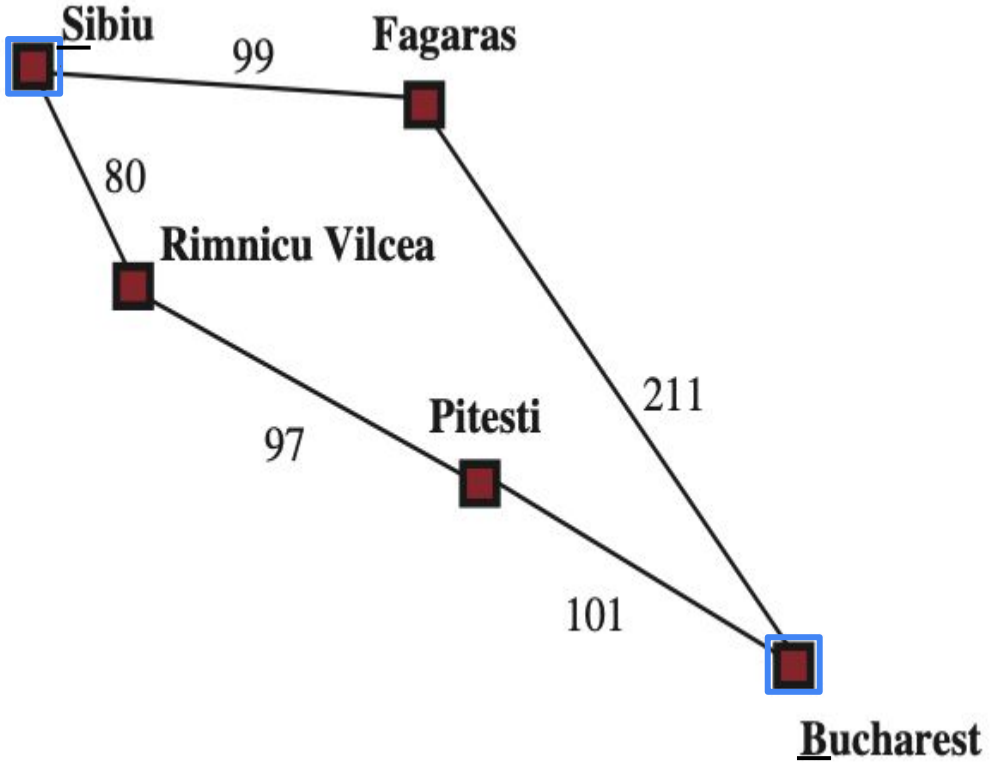


Uninformed Search Strategies

- Breadth-first search
- Depth-first search (DFS)
- Depth-limited search (DLS)
- Iterative deepening search (IDS)
- **Uniform-cost search (UCS)**
- Bidirectional search

Cost-Sensitive Search Problems

- Start: **S**ibiu
- Destination: **B**ucharest
- Goal
  - Find the **least-cost** path

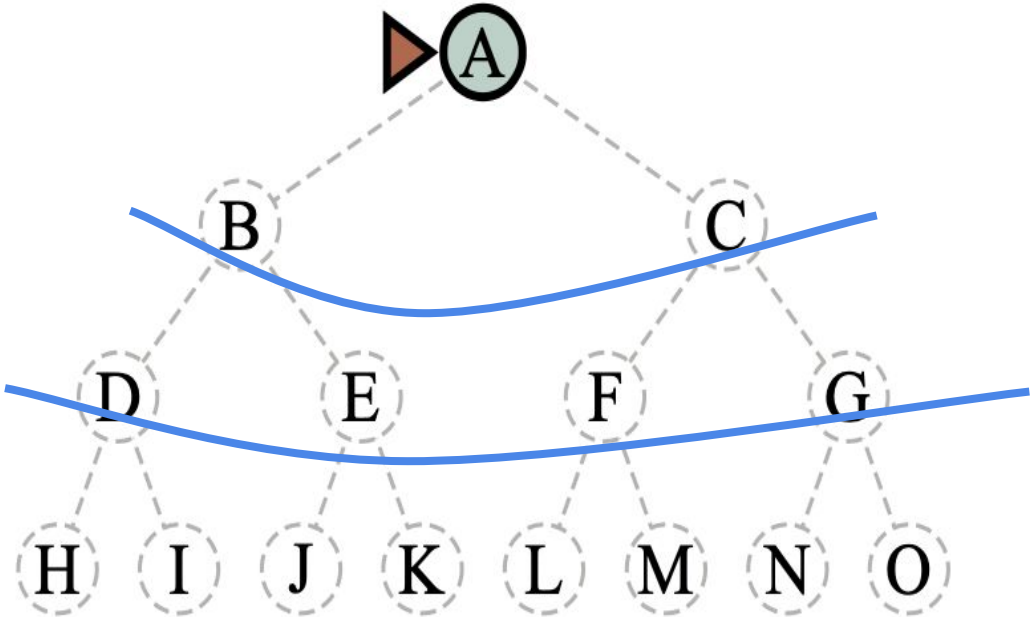


Uniform Cost Search (UCS)

- Uniform cost search spreads out in waves of **uniform path-cost** while breadth-first search spreads out in waves of uniform depth — first depth 1, then depth 2, and so on
  (i.e., expand least-cost ~~unexpended~~ unexpanded node)

Example: Breadth-First Search on a Simple Binary Tree

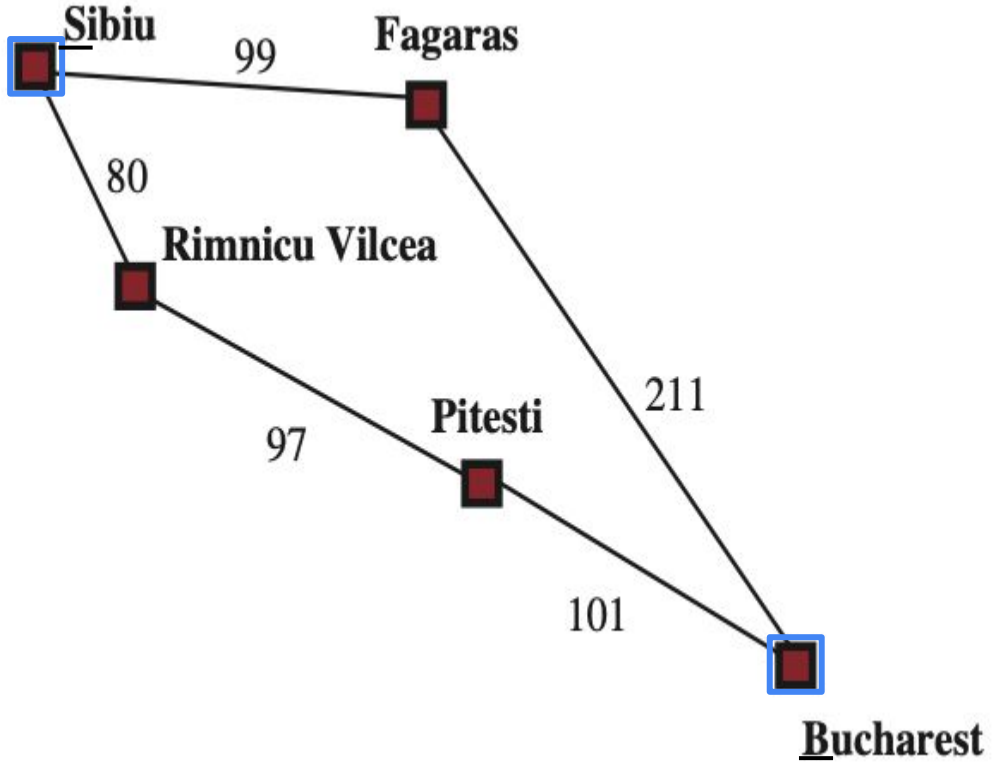
- Start: A
- Goal: M



Example: UCS for the Route Planning

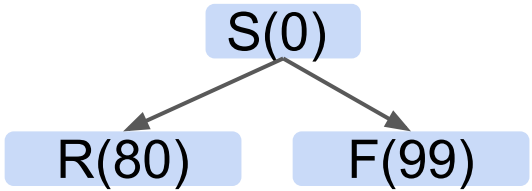
- Goal: find the least-cost path from Sibiu to Bucharest

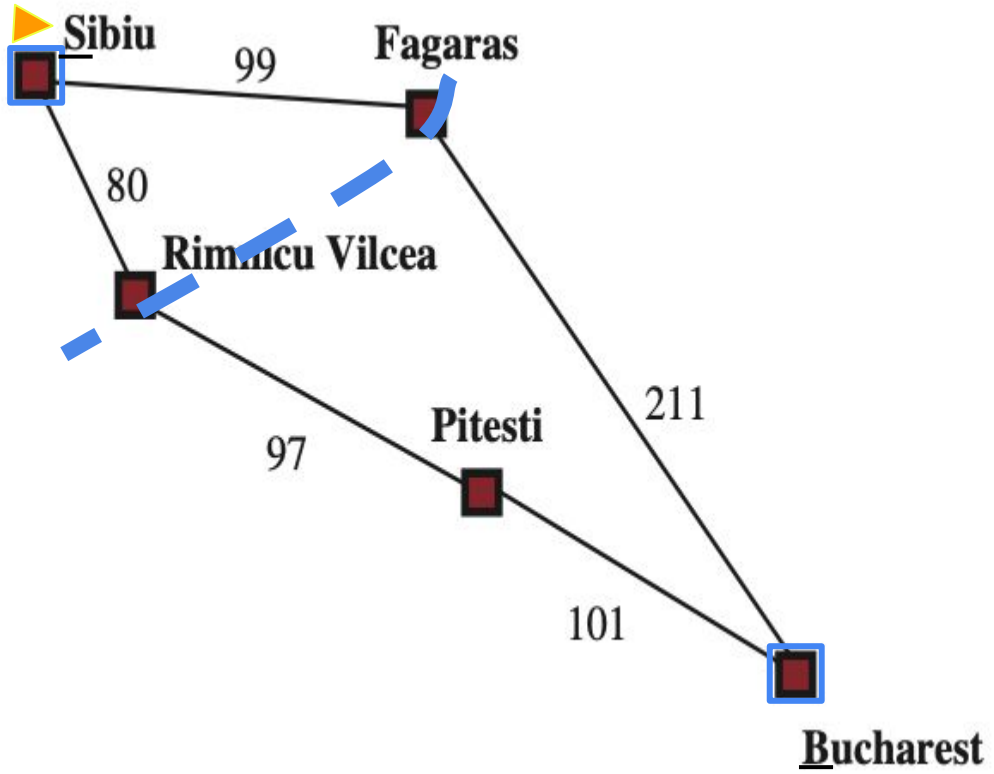
Start Goal



Example: UCS for the Route Planning

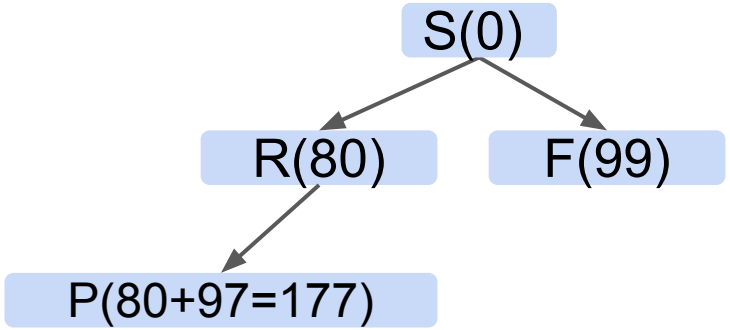
- Goal: find the least-cost path from Sibiu to Bucharest
- Steps:

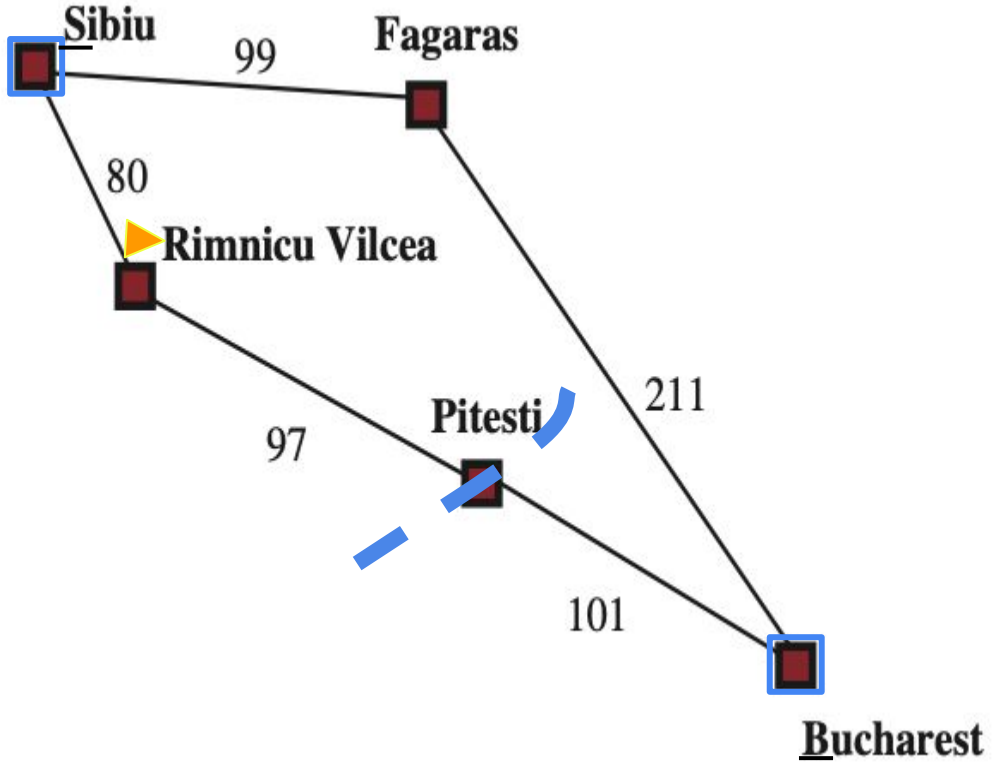




Example: UCS for the Route Planning

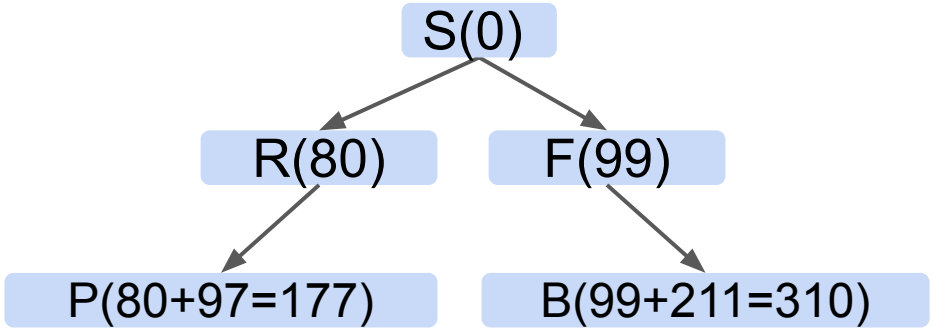
- Goal: find the least-cost path from Sibiu to Bucharest
- Steps:

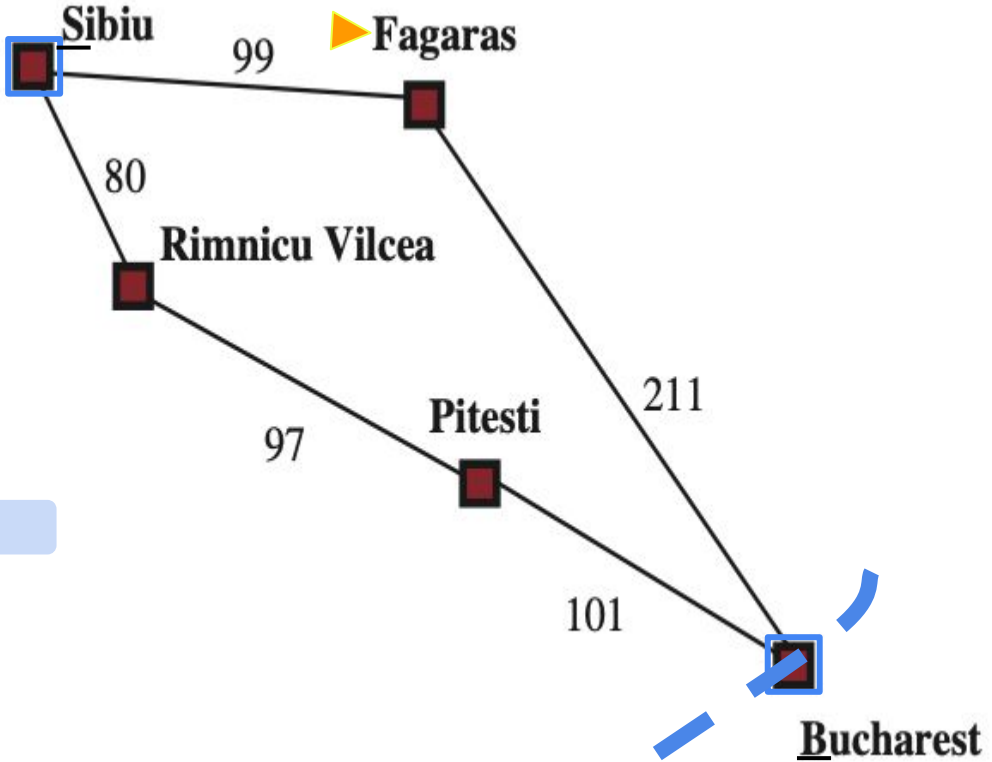




Example: UCS for the Route Planning

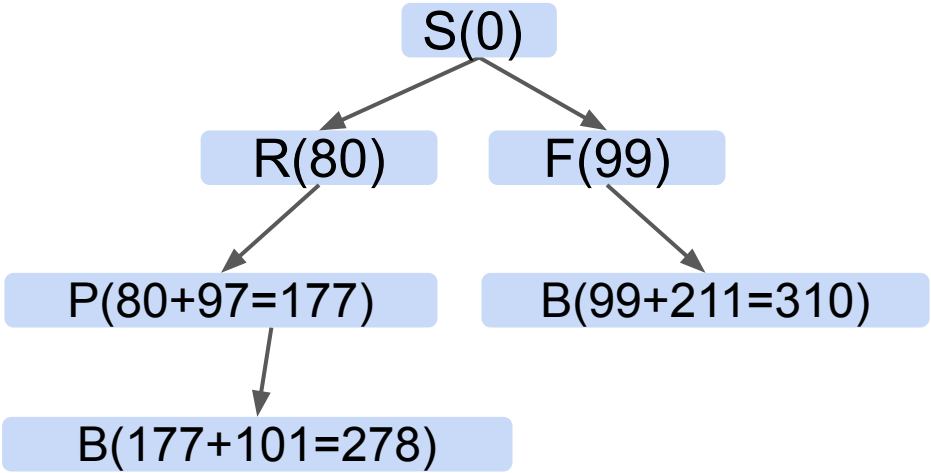
- Goal: find the least-cost path from Sibiu to Bucharest
- Steps:

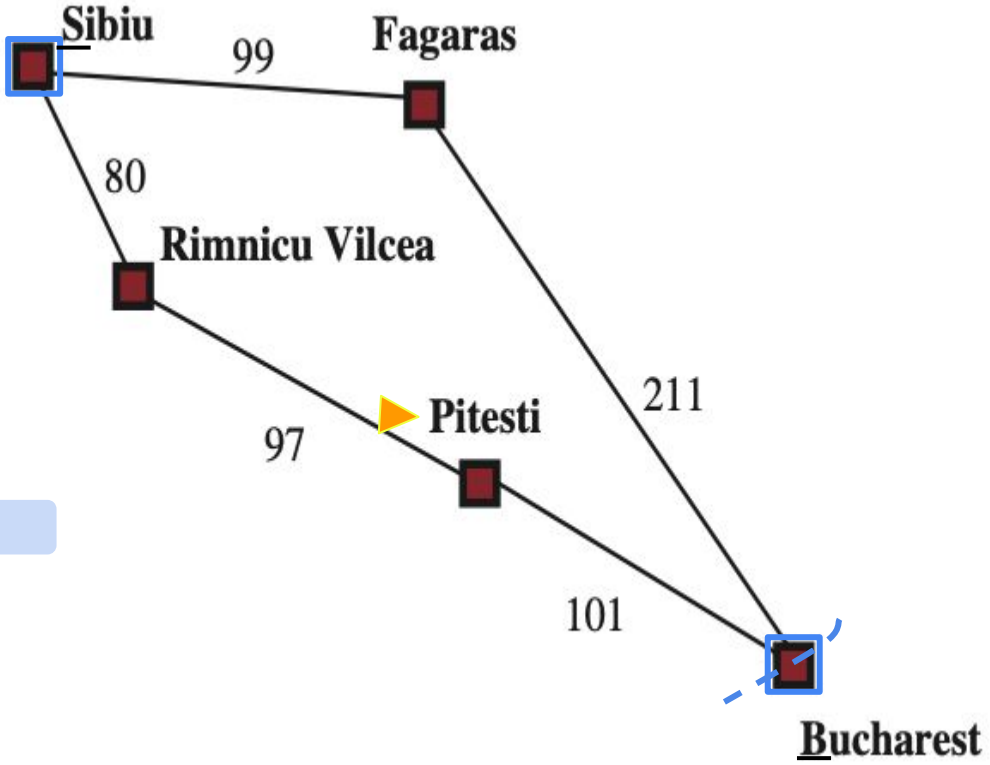




Example: UCS for the Route Planning

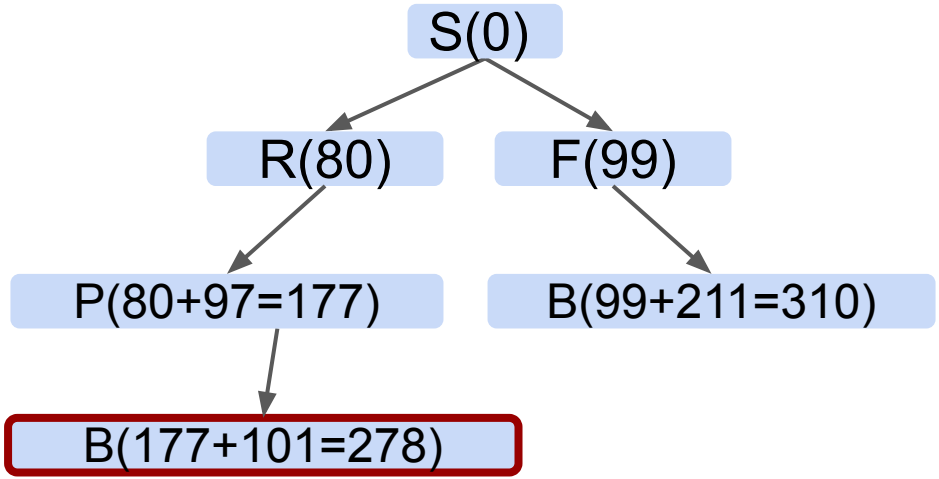
- Goal: find the least-cost path from Sibiu to Bucharest
- Steps:

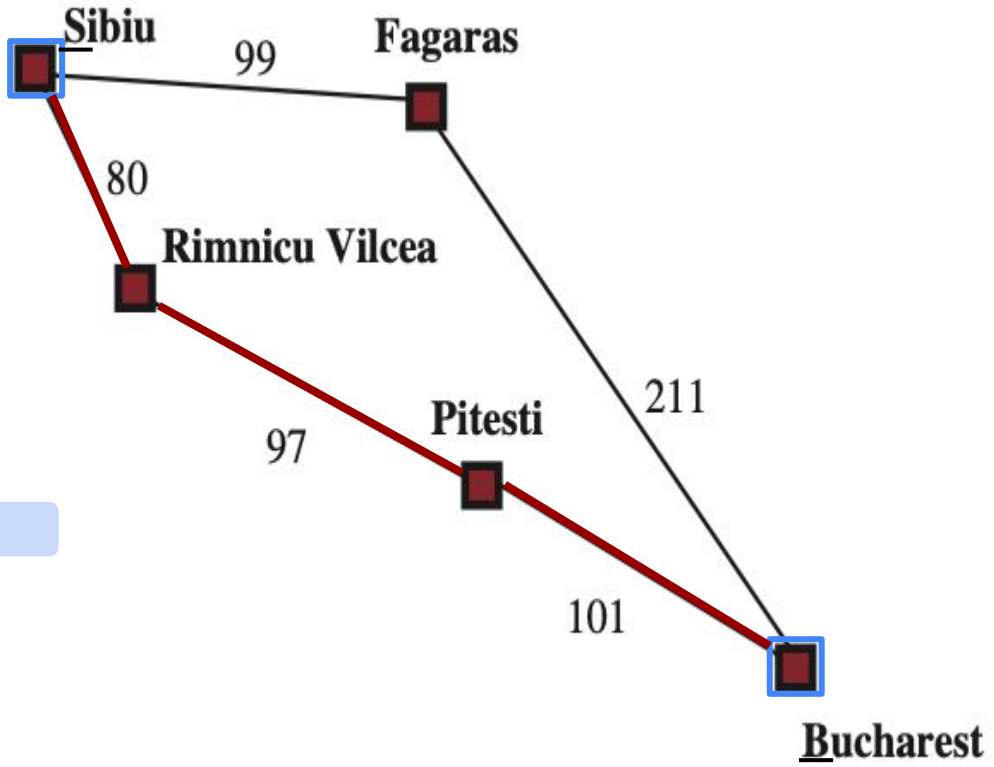




Example: UCS for the Route Planning

- Goal: find the least-cost path from Sibiu to Bucharest
- Steps:





Best-First Search

- On each iteration, we choose a node, *n*, with **minimum** value of some evaluation function *f*(*n*)

refer to Breadth First Search

Best-First Search Algorithm

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node : solution found
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```

accumulated path cost

Uniform Cost Search (UCS) Algorithm

function UNIFORM-COST-SEARCH(*problem*) **returns** a solution node, or *failure*
  return BEST-FIRST-SEARCH(*problem*, PATH-COST)

Properties of UCS (Refer to Breadth First Search)

Note. b is the branching factor, C^* is the cost of the optimal solution, and a lower bound ε on the cost of each action, with $\varepsilon > 0$

- Complete?
  - Yes (like BFS, and and all action costs are $\geq \varepsilon$)
- Optimal cost?
  - Yes
- Time complexity?
  - $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$ → Depth
- Space complexity?
  - $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$

Best-First Search (cont.)

- By employing different $f(n)$ functions, we get different specific algorithms
  - Breadth-first search is a special case of best-first search
    - Let $f(n)$ = depth(n) where depth(n) is the depth of node n
  - Depth-first search is a special case of best-first search (HW1)

If we set all transition costs $d(x,x') = 1$

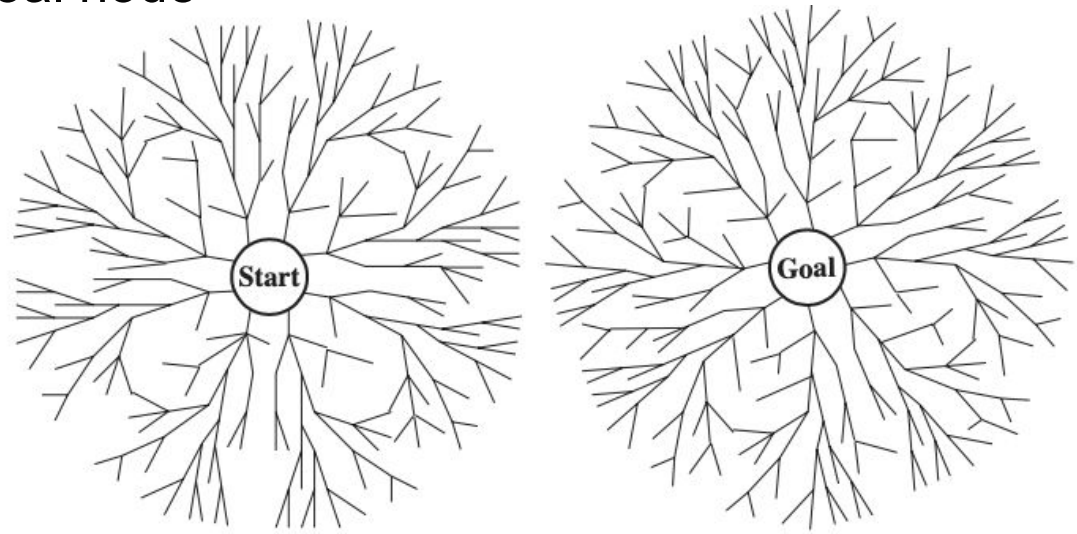
We could define $f(n)$ as a function that assigns lower score to nodes deeper in the tree

Uninformed Search Strategies

- Breadth-first search
- Depth-first search (DFS)
- Depth-limited search (DLS)
- Iterative deepening search (IDS)
- Uniform-cost search (UCS)
- **Bidirectional search**

Bidirectional Search

- Simultaneously searches forward from the initial state and backwards from the goal state(s)
  - Succeed when a branch from the start node meets a branch from the goal node



Properties of Bidirectional Search

Note. b is the branching factor and d is the depth of the shallowest solution

- Complete?
  - Yes (if b is finite and the state space either has a solution or is finite; if both directions are breadth-first or uniform-cost)
- Optimal cost?
  - Yes (if action costs are all identical and both directions are breadth-first or uniform-cost)
- Time complexity?
  - $O(b^{d/2})$
- Space complexity?
  - $O(b^{d/2})$

Search Topics

- Search problems (Ch. 3)
- Uninformed search (Ch. 3)
- **Informed search (Ch. 3)**
- Local search (Ch. 4)
- Adversarial search (Ch. 6)
- Constraint Satisfaction Problems (CSPs) (Ch. 5)

Informed Search

L.-Y. Wei

Spring 2024

Environment

Fully Observable
Deterministic
Static

Informed Search (Heuristic Search)

- An informed search algorithm uses domain-specific hints about the location of goals
  - The hints come in the form of a heuristic function
- **Heuristic function**
  - *h(n)* = **estimated cost** of the cheapest path from the state at node *n* to **goal state**

heuristic information

Informed Search Algorithms

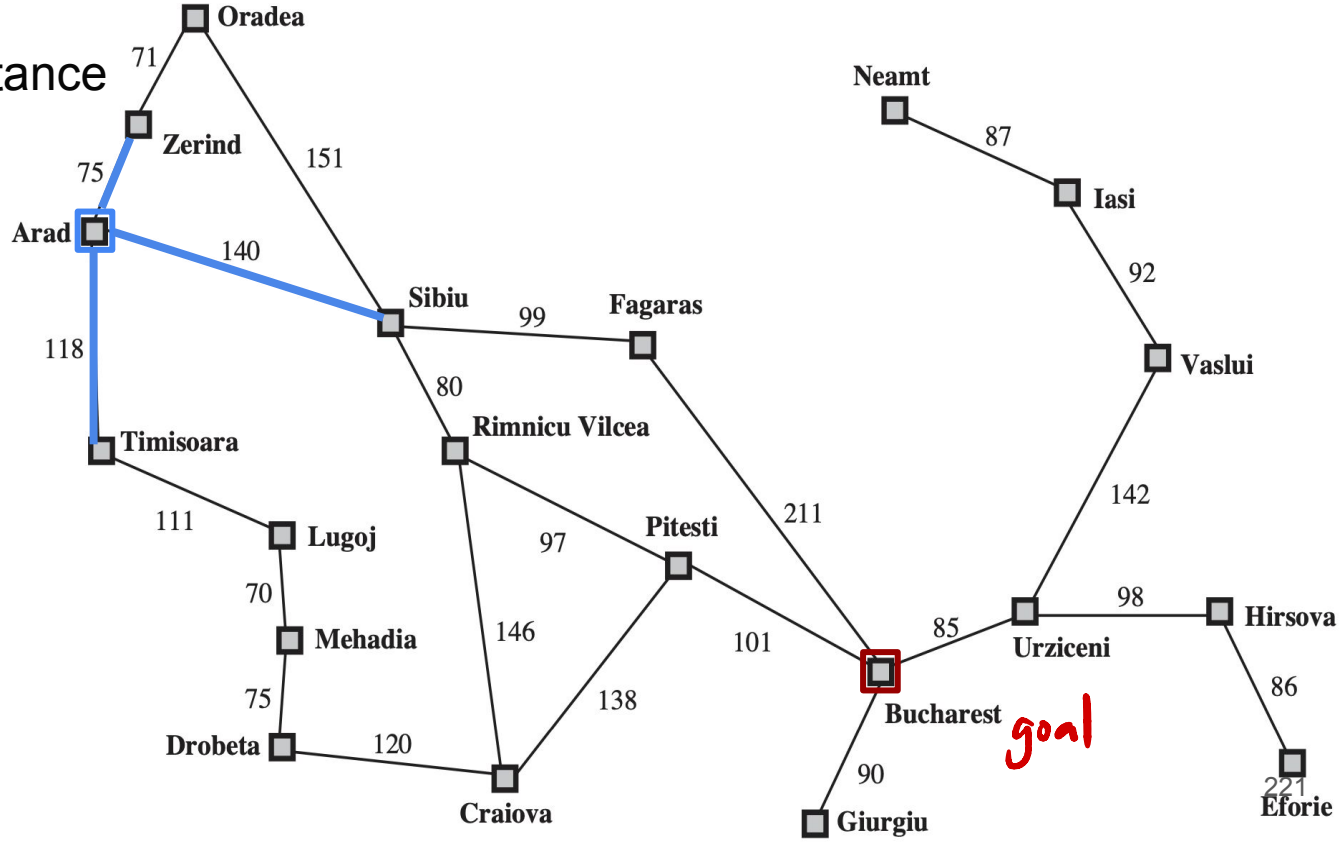
- Greedy best-first search
- A* search

Greedy Best-First Search

- Greedy best-first search expands the node that appears to be **closest to the goal** (i.e., the node with the lowest $h(n)$)

Example: Greedy search for Bucharest

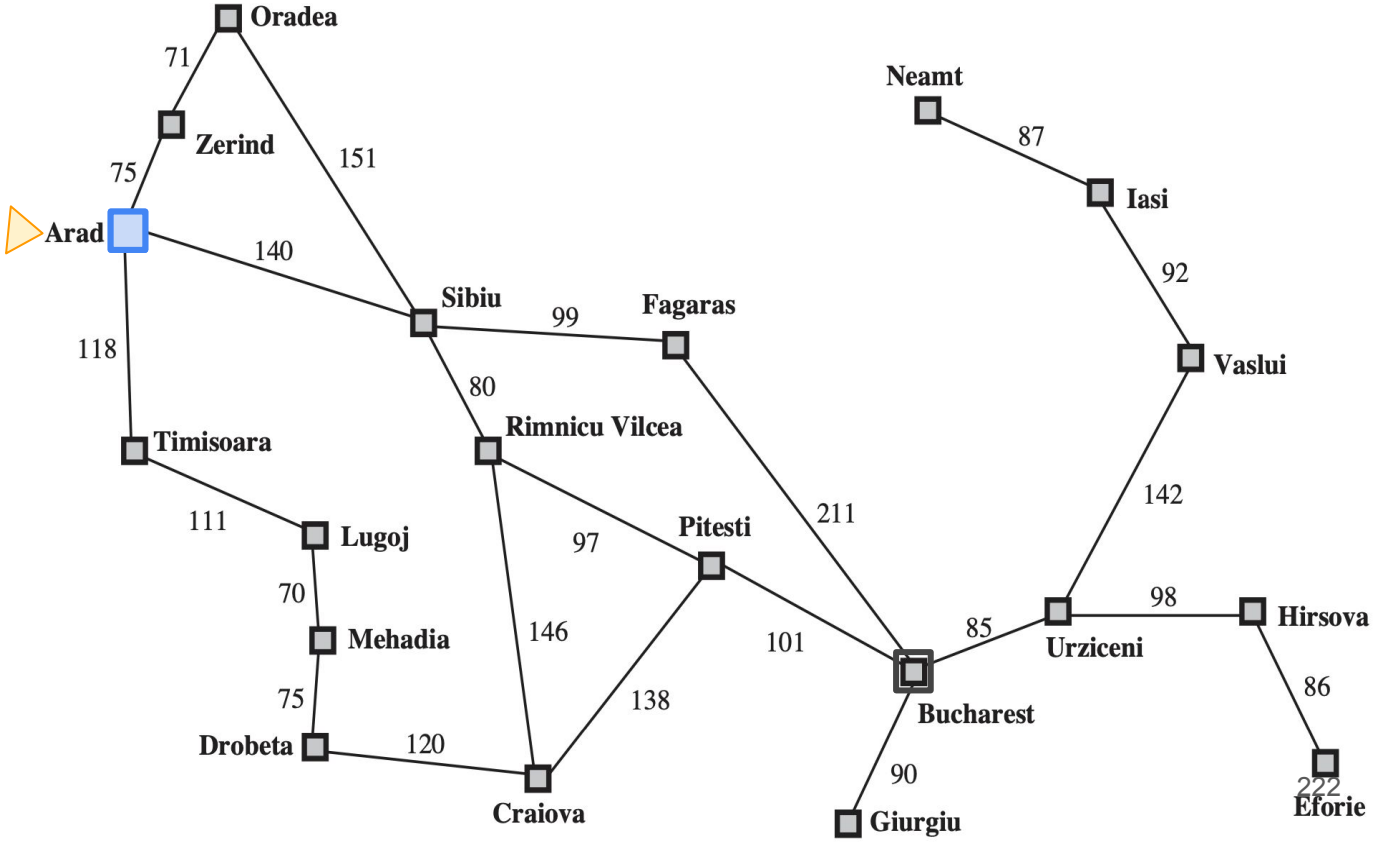
- Heuristic function?
  - Straight-line distance (h_{SLD})



Example: Greedy search for Bucharest

- Values of h_{SLD}

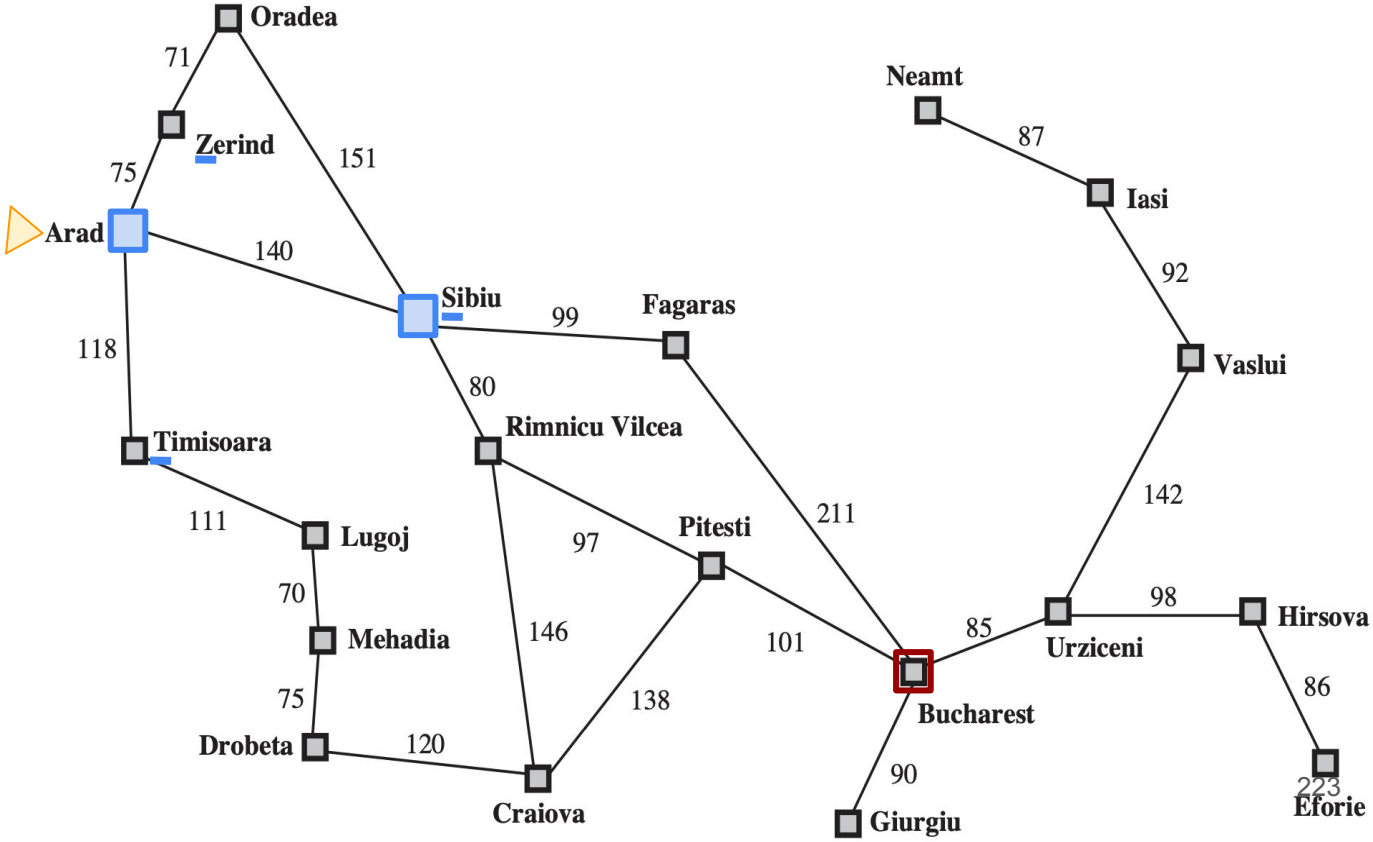
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



Example: Greedy search for Bucharest

- Values of h_{SLD}

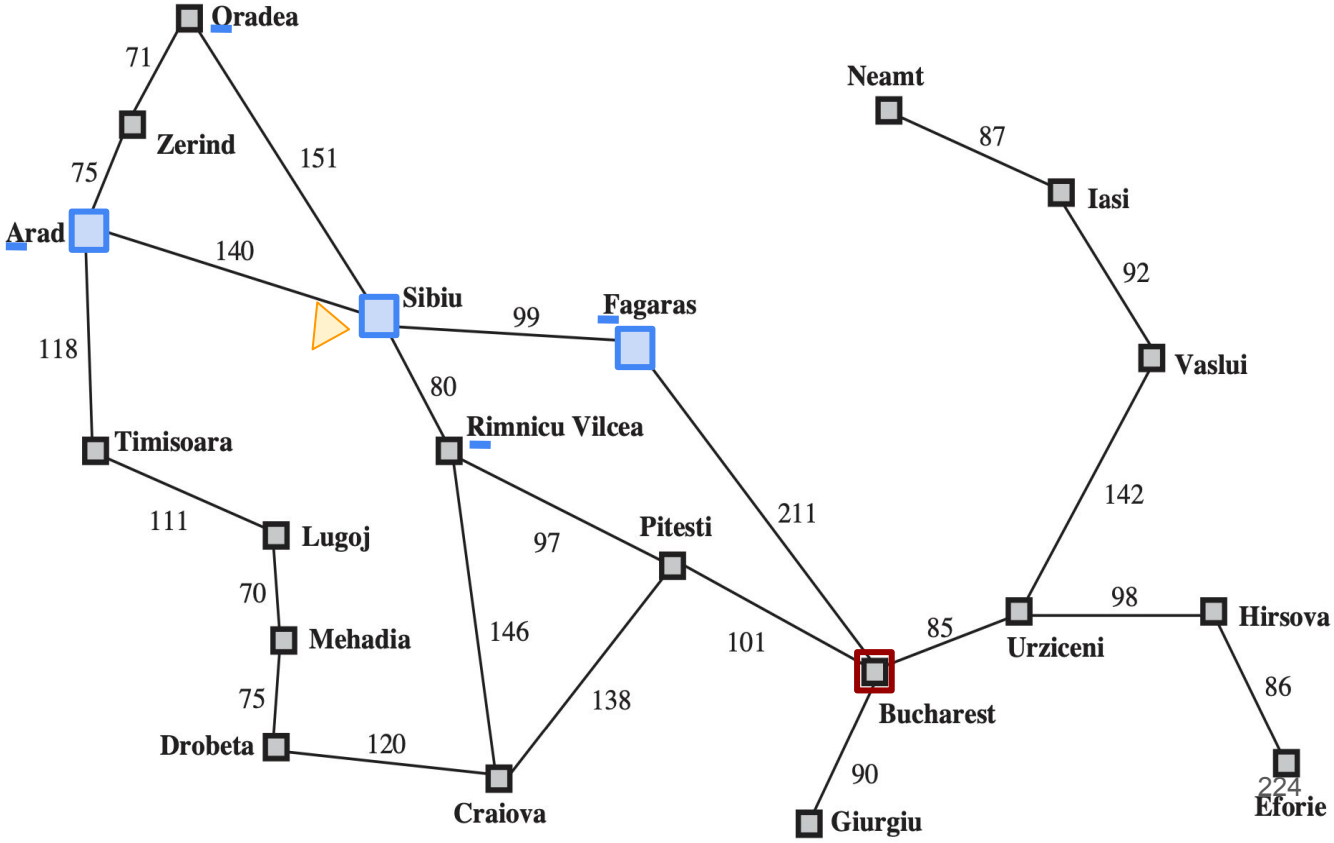
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



Example: Greedy search for Bucharest

- Values of h_{SLD}

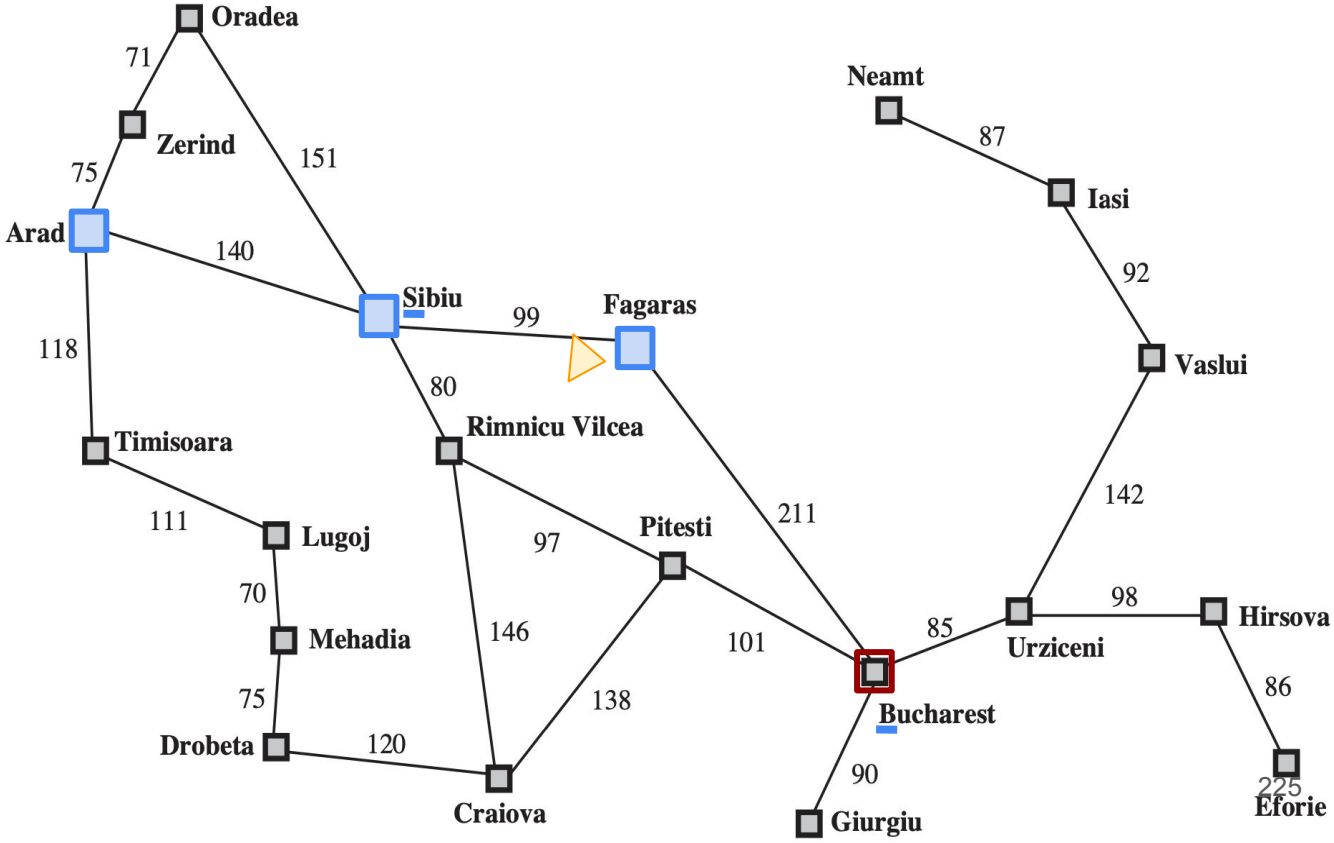
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



Example: Greedy search for Bucharest

- Values of h_{SLD}

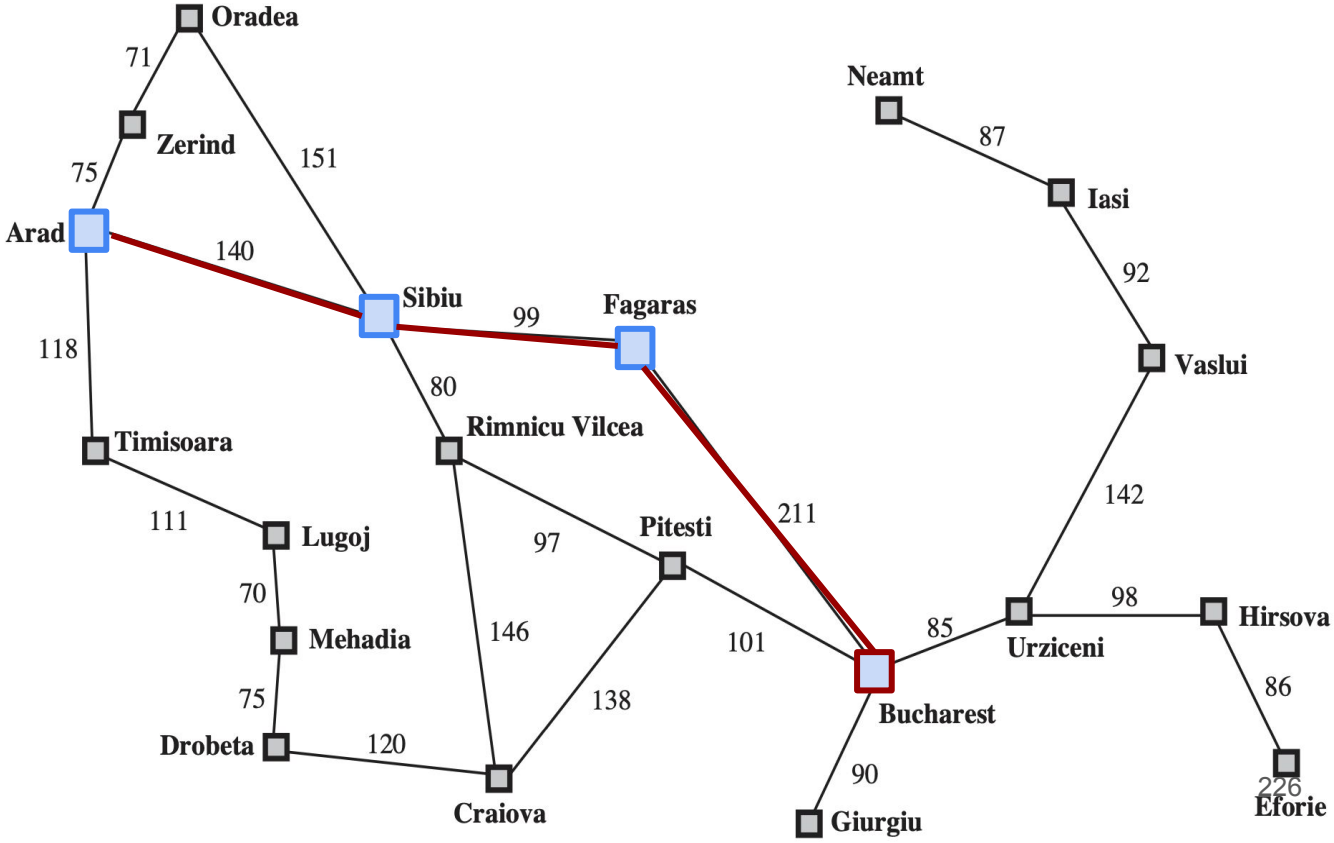
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

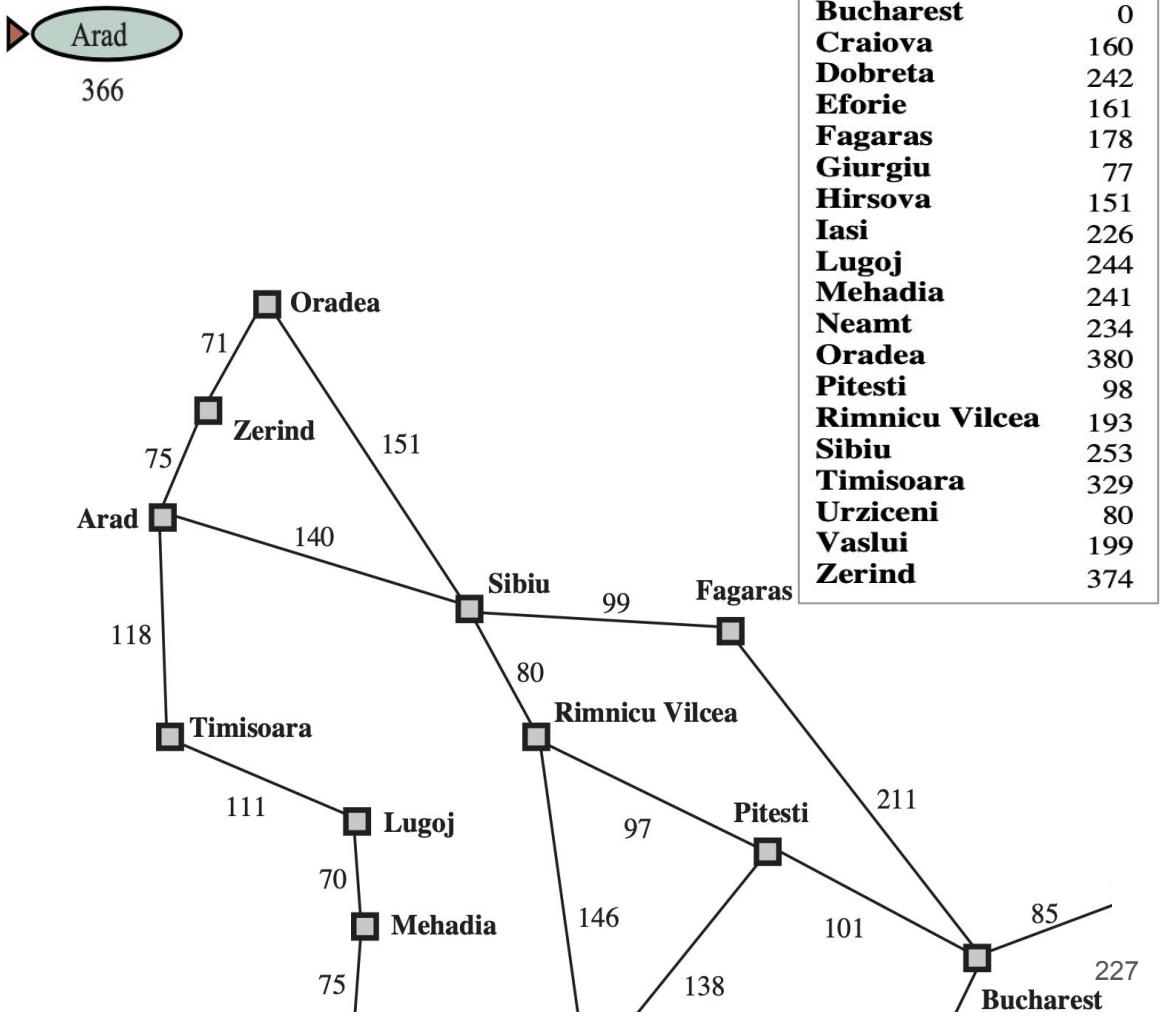


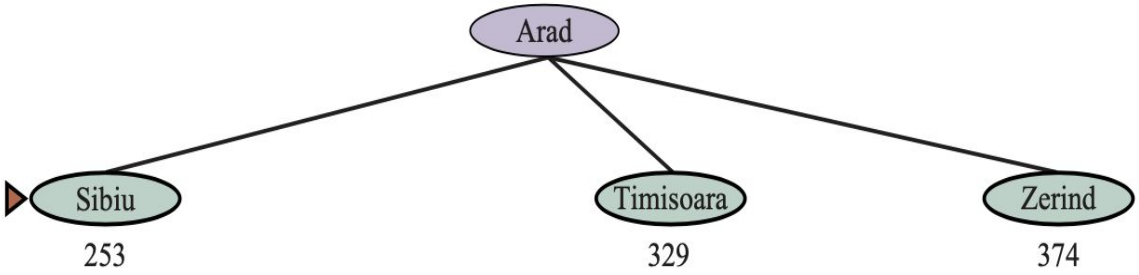
Example: Greedy search for Bucharest

- Values of h_{SLD}

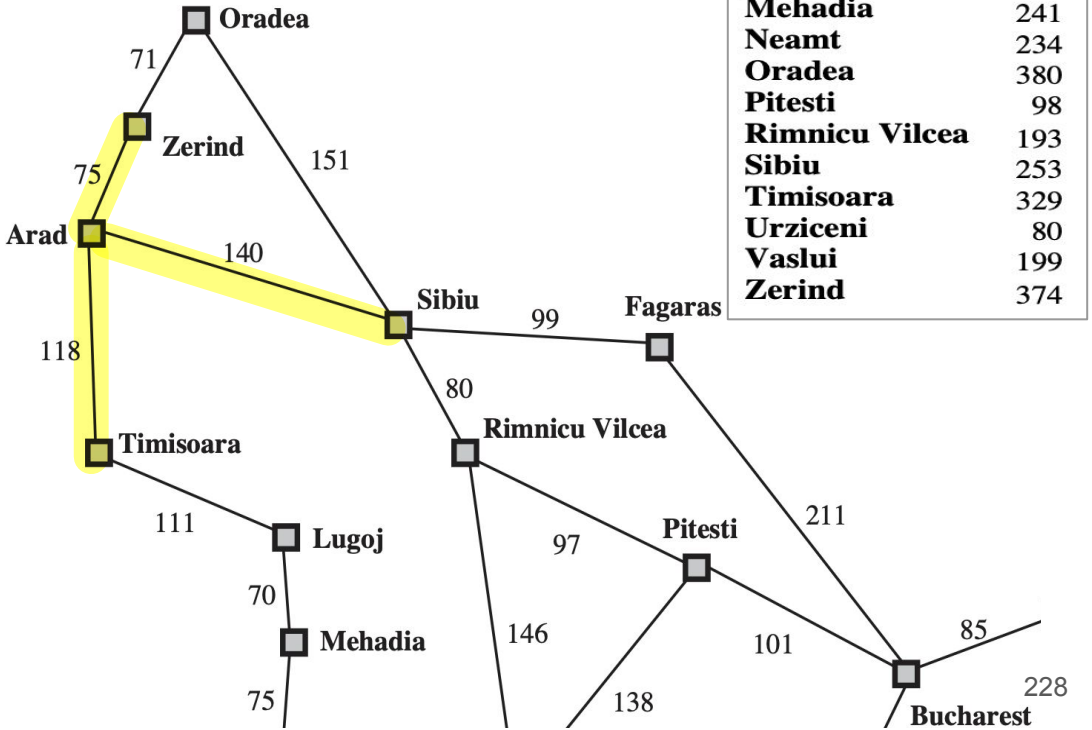
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

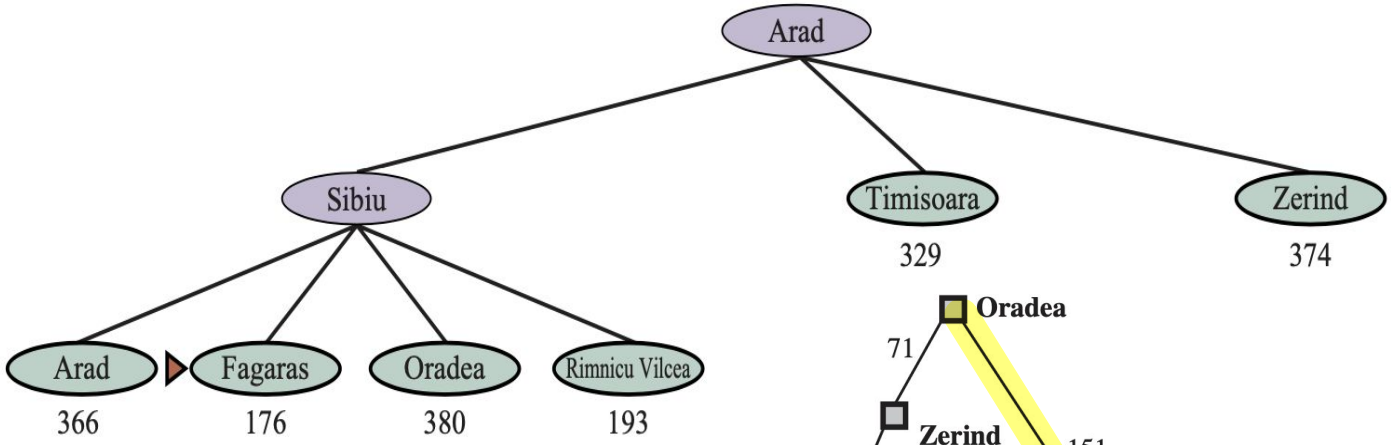



Arad
366
Oradea
71
Zerind
75
151
Arad
140
Sibiu
99
Fagaras
118
80
Timisoara
Rimnicu Vilcea
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
75
138
227
Bucharest
Arad 366
Bucharest 0
Craiova 160
Dobreta 242
Eforie 161
Fagaras 178
Giurgiu 77
Hirsova 151
Iasi 226
Lugoj 244
Mehadia 241
Neamt 234
Oradea 380
Pitesti 98
Rimnicu Vilcea 193
Sibiu 253
Timisoara 329
Urziceni 80
Vaslui 199
Zerind 374

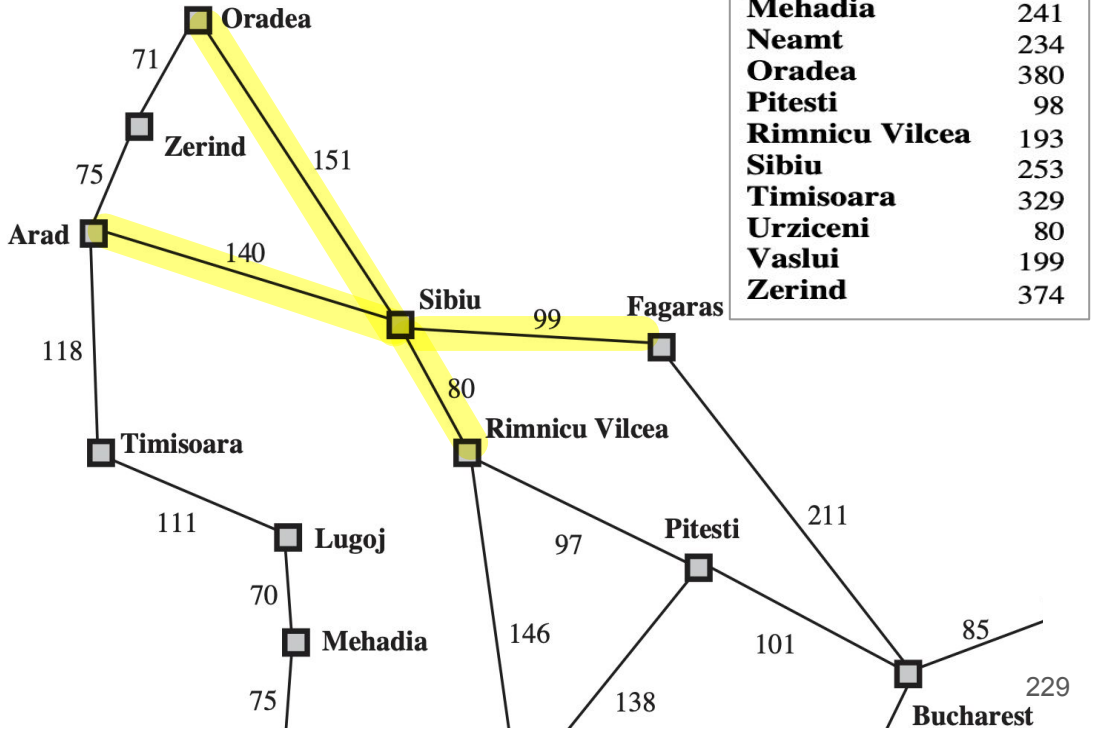


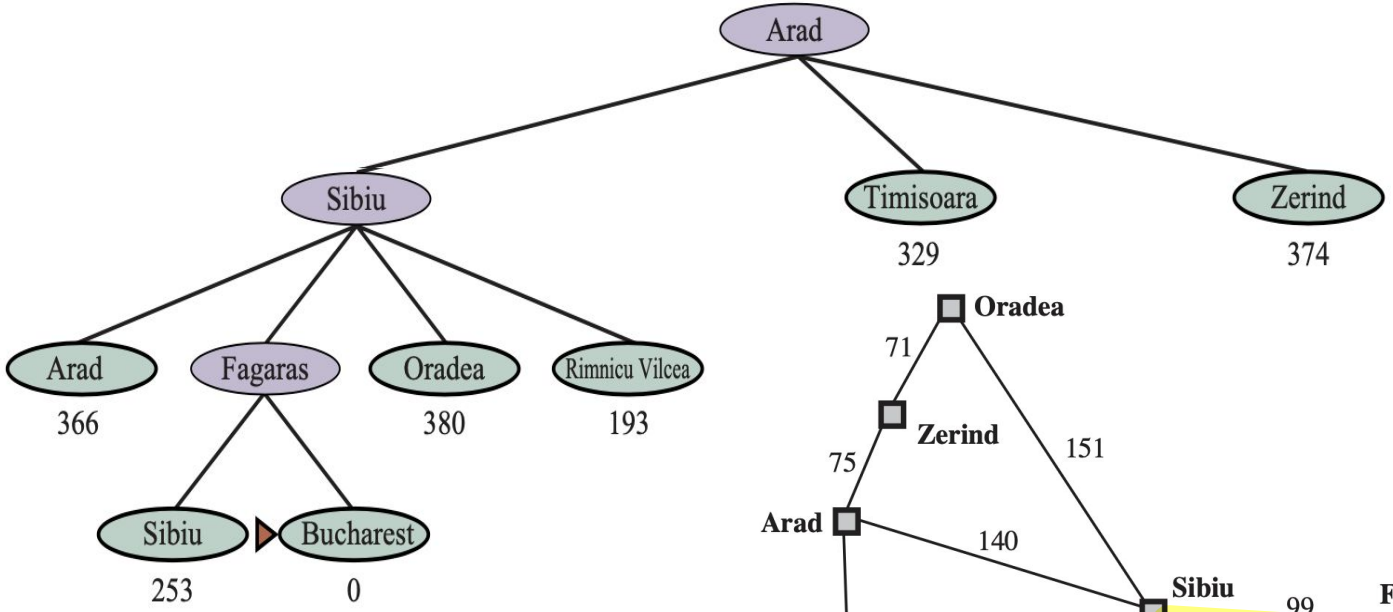
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

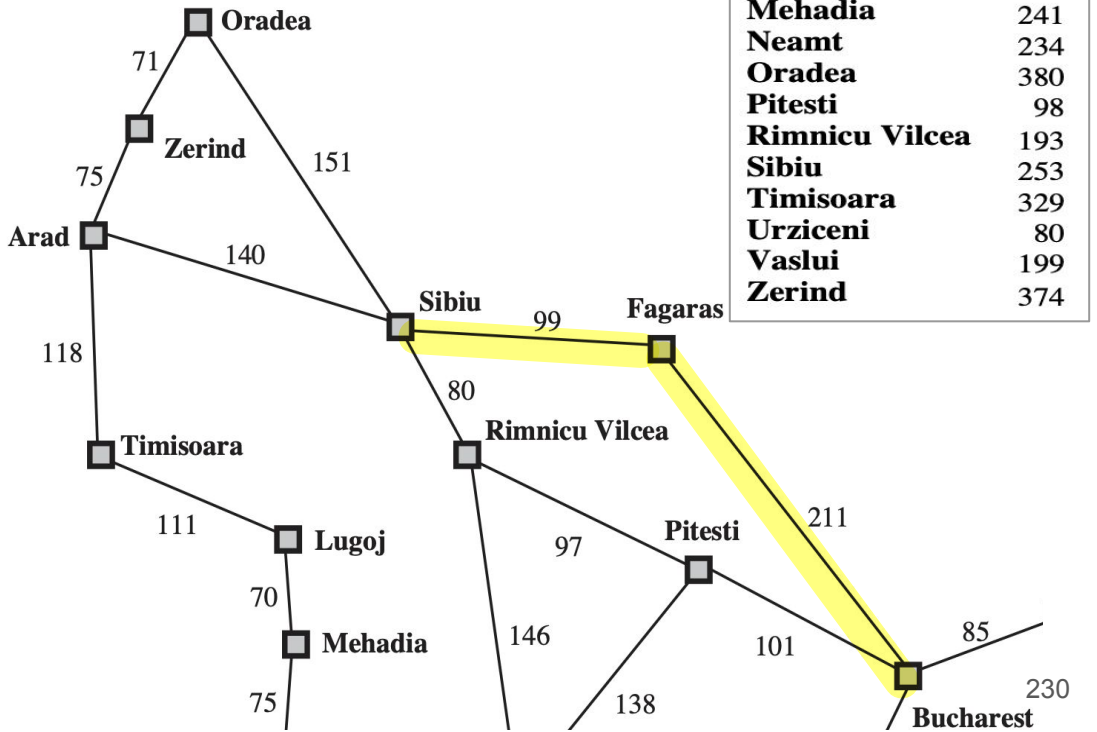




Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374







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

Properties of Greedy Best-First Search

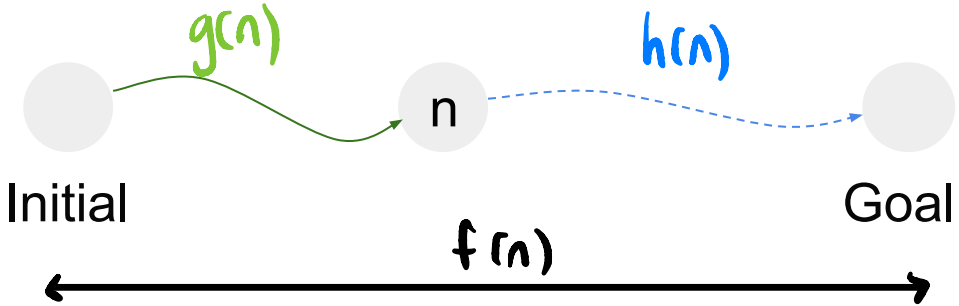
- Complete?
  - Yes (in finite space with repeated-state checking) / No
- Optimal cost?
  - No
- Time complexity?
  - O(|V|) # depends on a heuristic function
- Space complexity?
  - O(|V|) # keeps all nodes in memory

A* Search

- A best-first search that uses the evaluation function

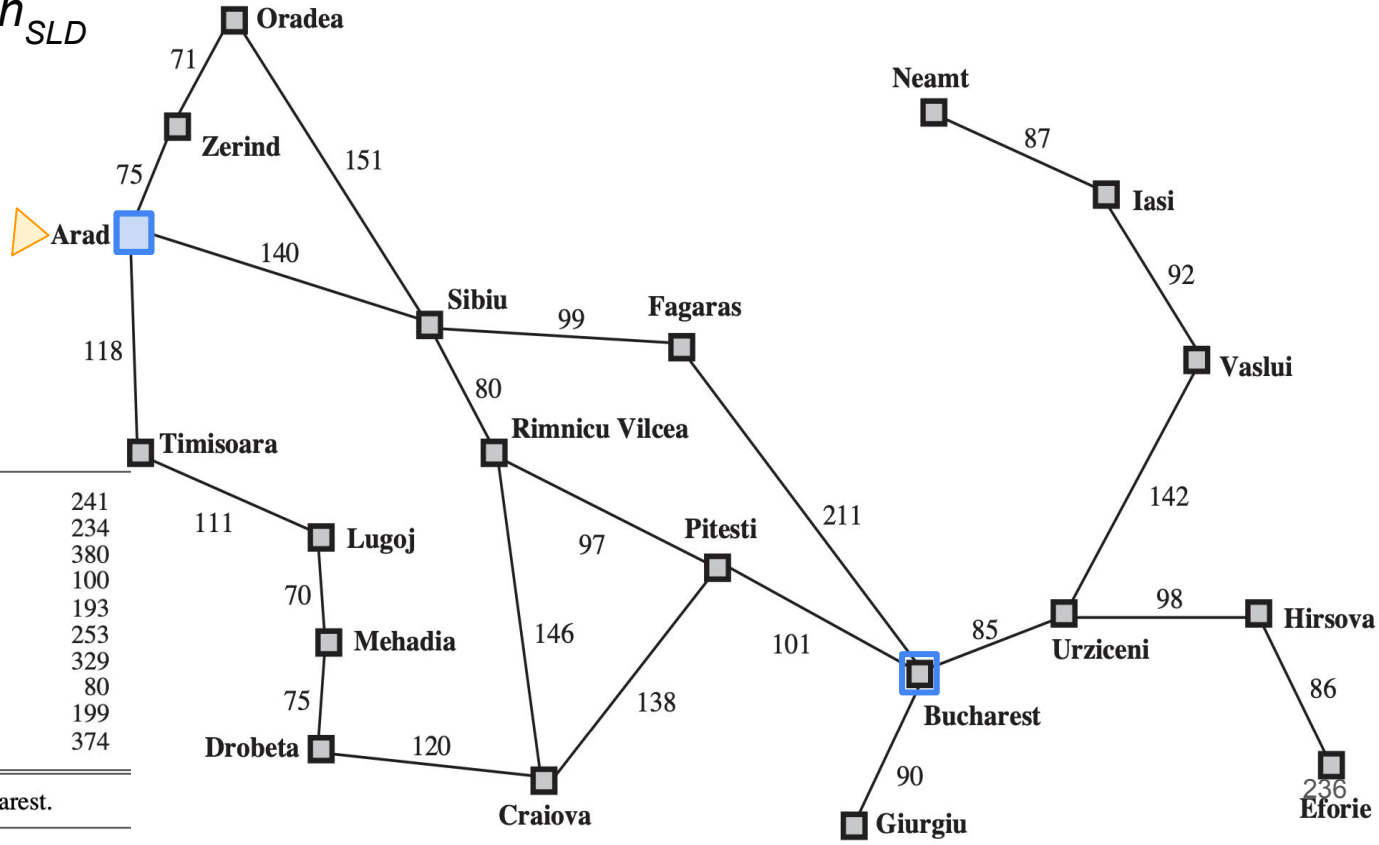
$$f(n) = g(n) + h(n)$$

  - $g(n)$ = path cost from the initial state to node n
  - $h(n)$ = **estimated cost** of the **shortest path** from n to a **goal state**
  - $f(n)$ = estimated cost of the best path that continues from n to a goal



Example: A* search for Bucharest

- Heuristic function, h_{SLD}

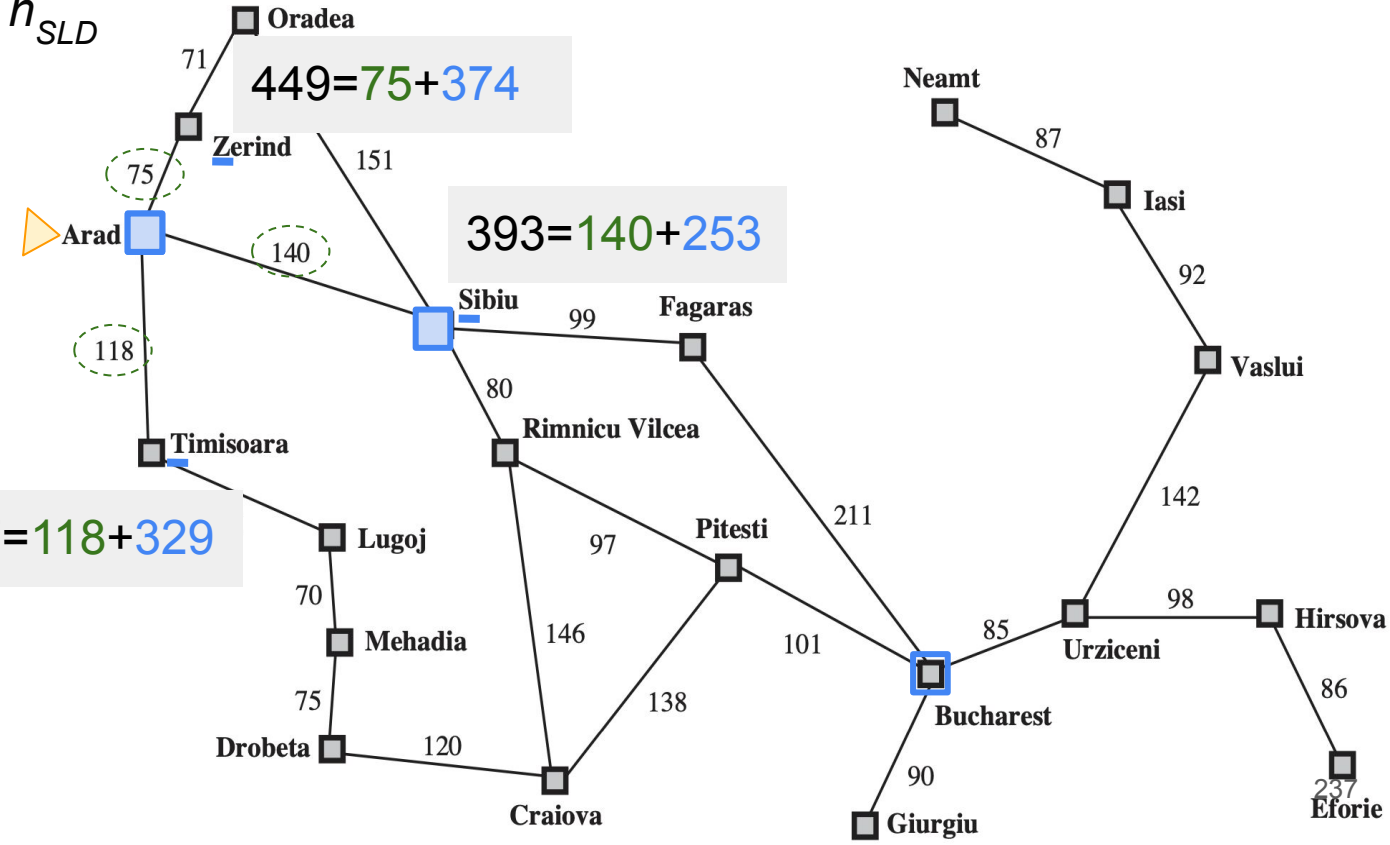


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

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

Example: A* search for Bucharest

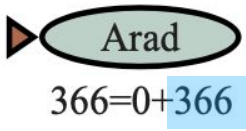
- Heuristic function, h_{SLD}



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

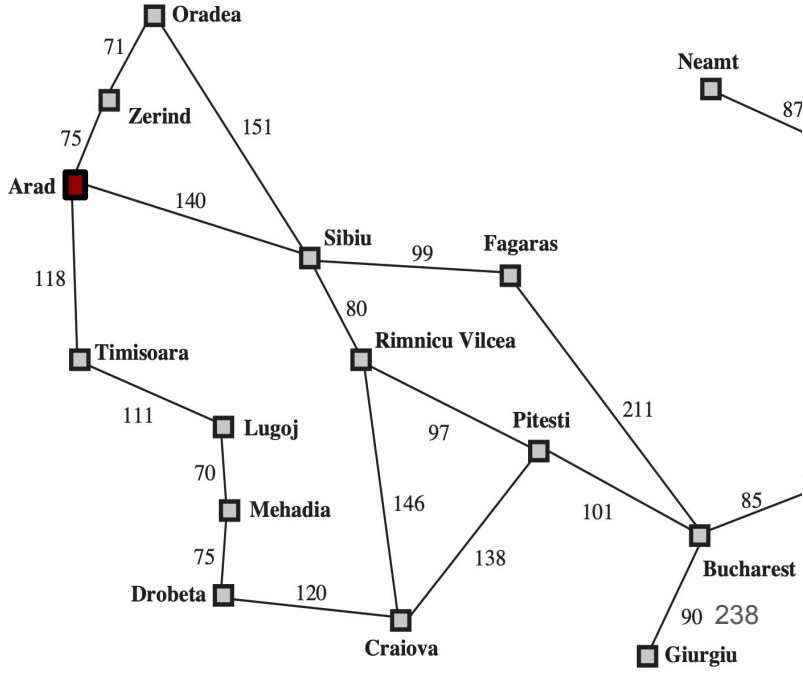
Values of h_{SLD}—straight-line distances to Bucharest.

(a) The initial state

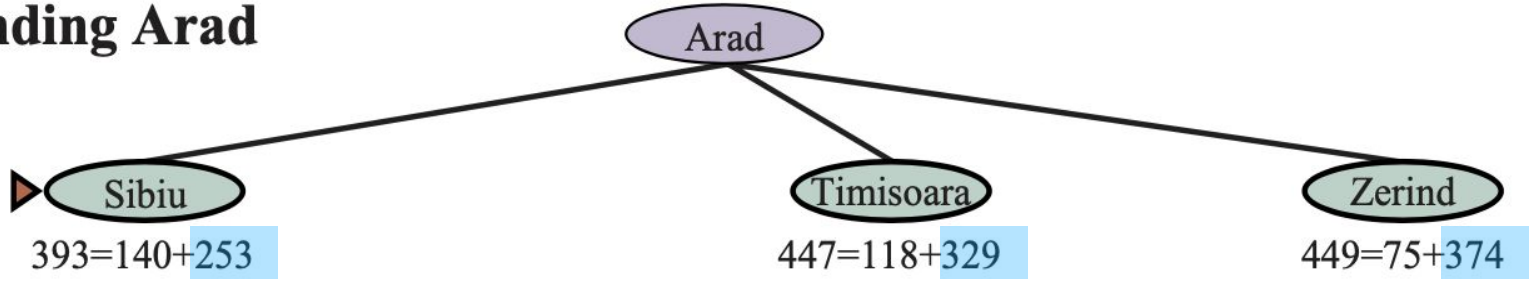


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

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

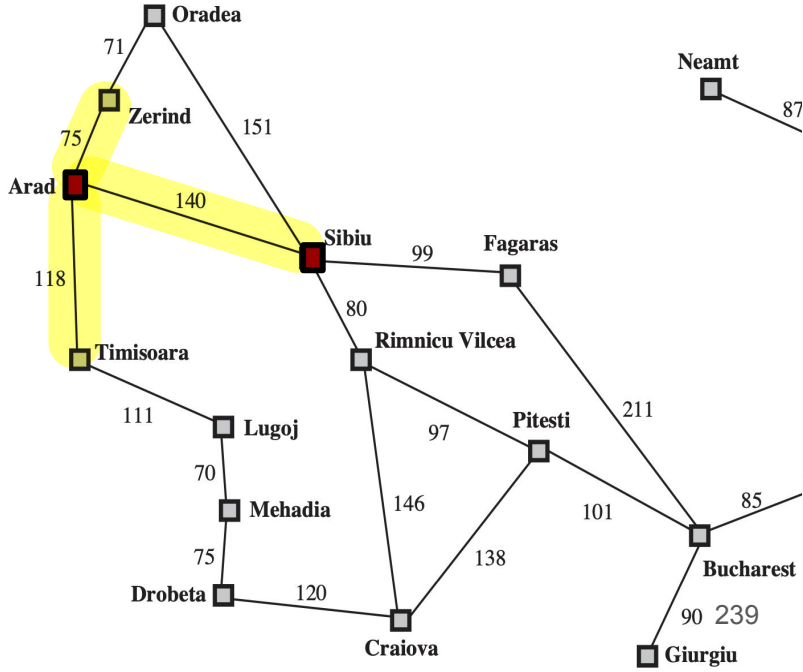


(b) After expanding Arad

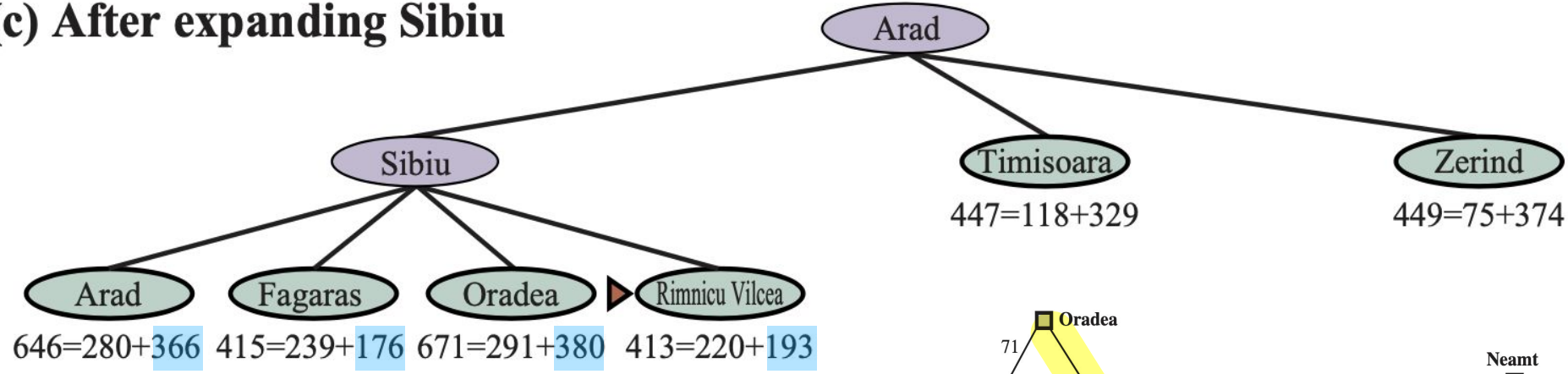


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

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

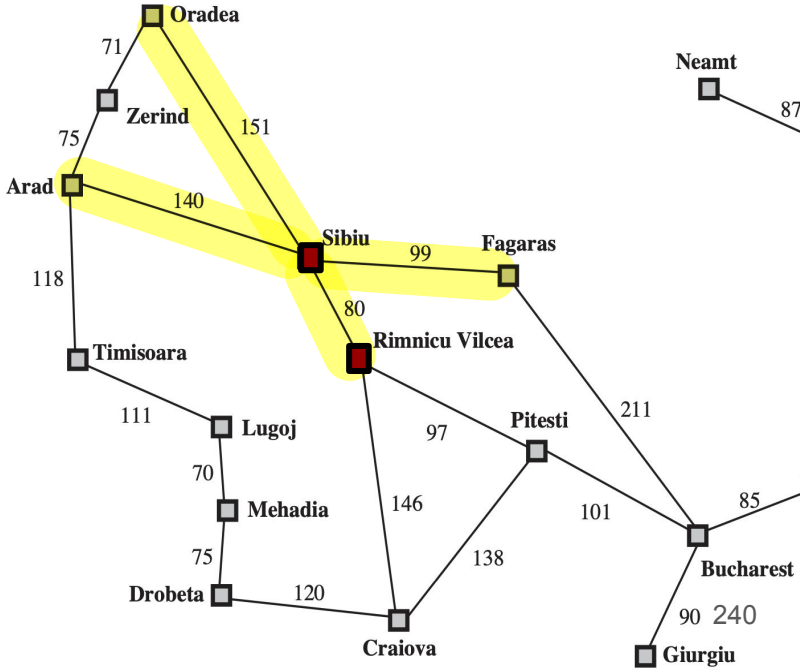


(c) After expanding Sibiu

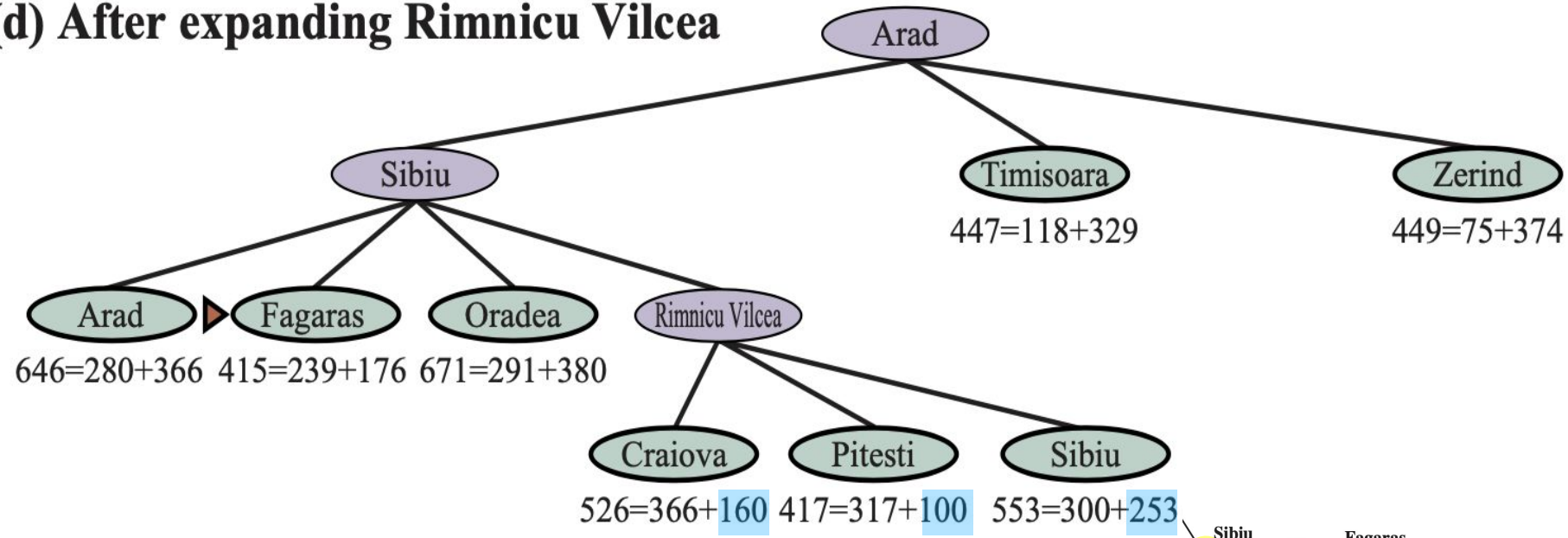


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

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

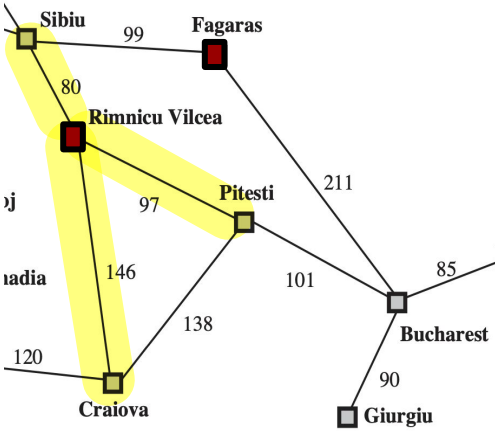


(d) After expanding Rimnicu Vilcea

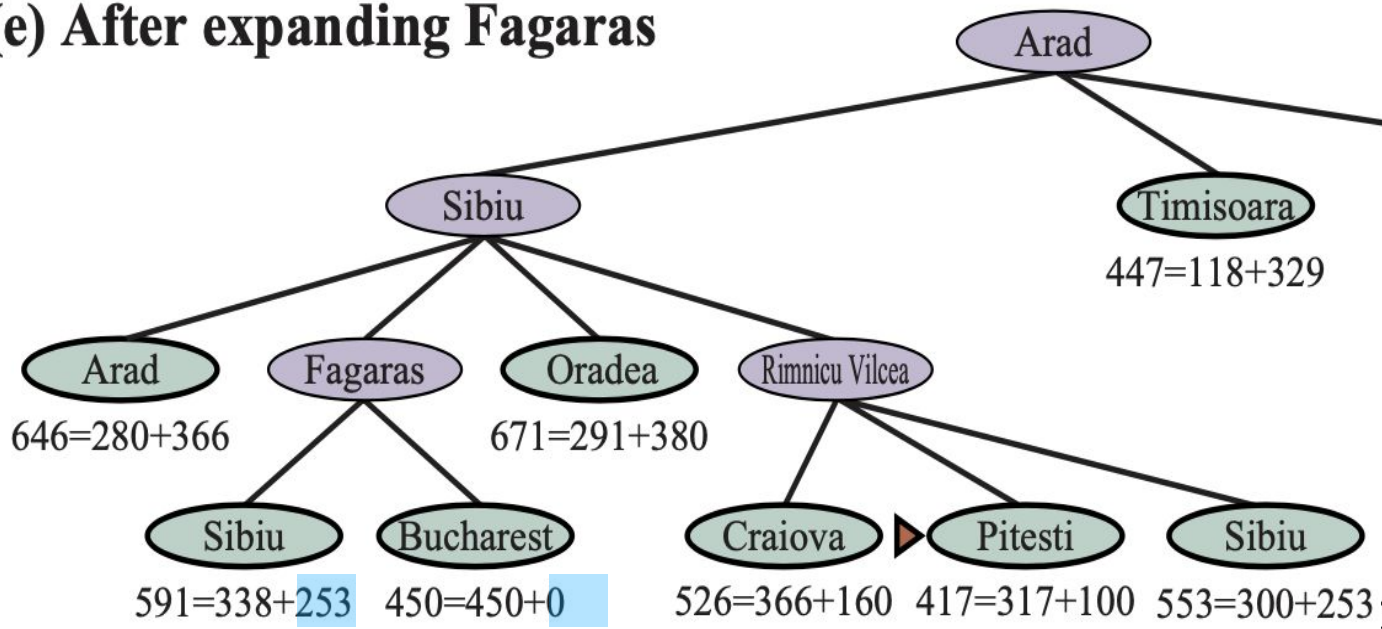


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

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

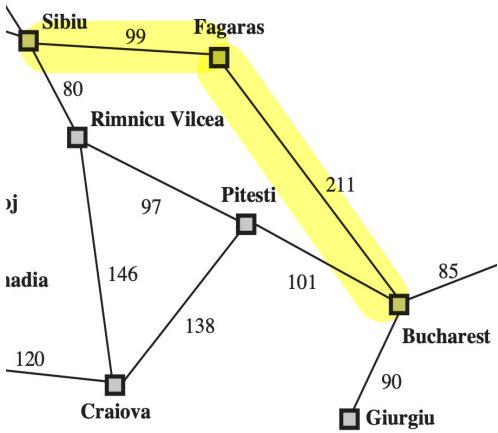


(e) After expanding Fagaras

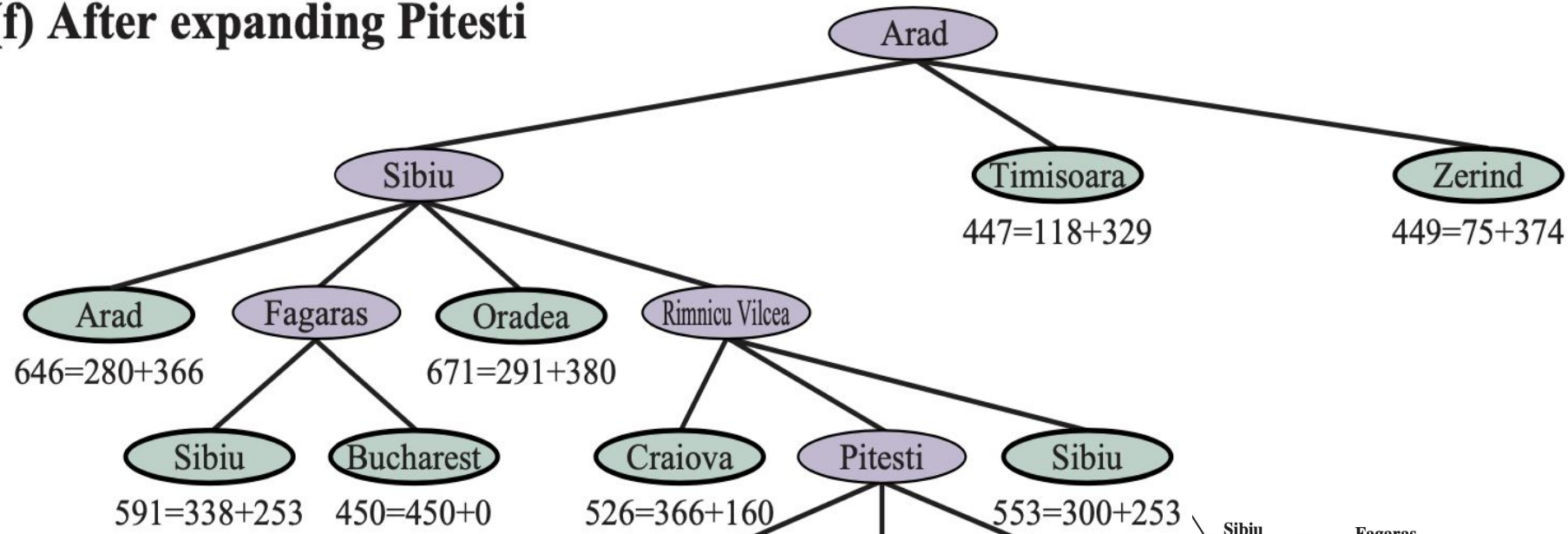


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

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

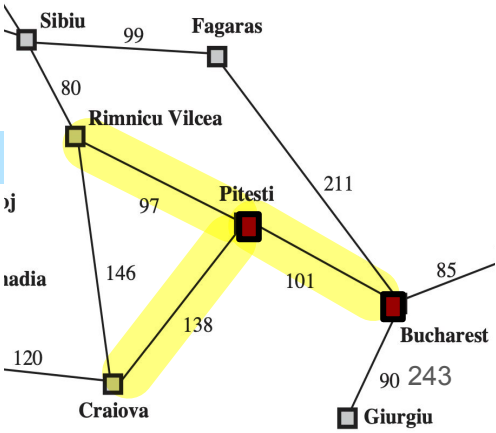


(f) After expanding Pitesti



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

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



A* Search is Optimal?

Admissible Heuristics

Any heuristic that ensures this can "admit" a solution consistently, making it an "admissible heuristic"

- An **admissible heuristic** is one that **never overestimates** the cost to reach a goal, that is,

  an admissible heuristic function $h(n) \leq h^*(n)$
  where $h^*(n)$ is the true cost from n

- E.g., $h_{SLD}(n)$? (SLD : straight-line distance)

  Yes
  (because $h_{SLD}(n)$ never overestimates the actual road distance)

A* is cost-optimal with an admissible heuristic.

Before proving it …

If all the nodes on the optimal path had been expanded, then we would have returned that optimal solution.

There must be some node *n* which is on the optimal path and is unexpanded.

admissible heuristic ⇒ cost optimal

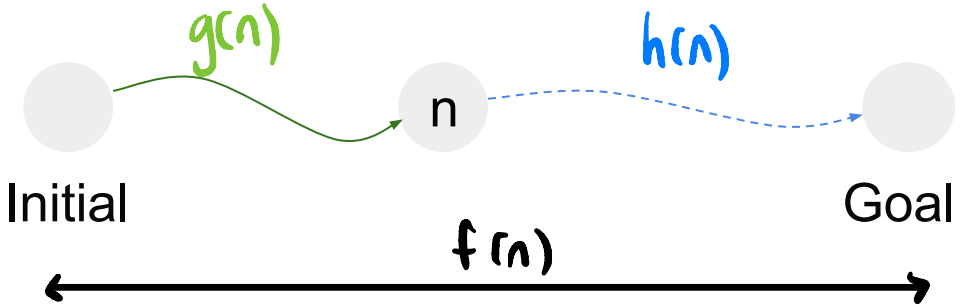
(the converse is not true)

A* Search (Recall)

- A best-first search that uses the evaluation function

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

  - $g(n)$ = path cost from the initial state to node n
  - $h(n)$ = **estimated cost** of the **shortest path** from n to a **goal state**
  - $f(n)$ = estimated cost of the best path that continues from n to a goal

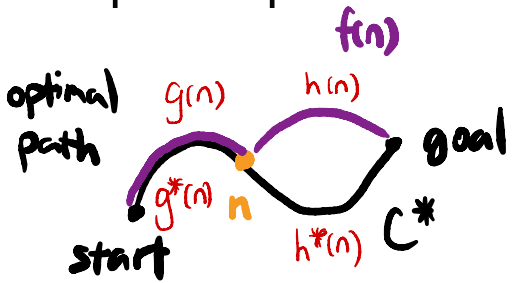


A* is cost-optimal with an admissible heuristic.

(Hint: proof by contradiction)

Pf. Supp. the algorithm returns a path with cost $C > C^*$ where the optimal path has cost C^*.

Then, there must be some node n which is on the optimal path and is unexpanded.



$$
\begin{aligned}
f(n) &> C^* \\
f(n) &= g(n) + h(n) \quad \text{(by definition)} \\
f(n) &= g^*(n) + h(n) \quad \text{(Let } g^*(n) \text{ be the cost of the optimal path from the start to } n\text{)} \\
f(n) &\leq g^*(n) + h^*(n) \quad \text{(because of admissibility, } h(n) \leq h^*(n)\text{)} \\
f(n) &\leq C^* \quad \text{(by definition, } C^* = g^*(n) + h^*(n)\text{)}
\end{aligned}
$$

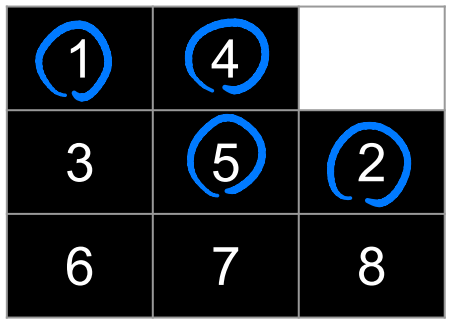
⊥ contradiction (∵ C^* should be the optimal path)

Properties of A* Search

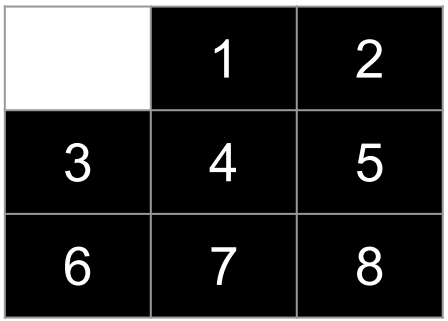
- Complete?
  - Yes (all action costs are >0 and the stace space either has a solution or is finite)
- Optimal cost?
  - Yes (with an admissible heuristic function)
- Time complexity?
  - O(|V|) # depends on a heuristic function
- Space complexity?
  - O(|V|) # keeps all nodes in memory

Example: Heuristic functions for the 8-puzzle problem

- Heuristic function
  - $h_1(n)$ = the number of misplaced tiles (blank not included)



Start



Goal

- E.g., $h_1(S)$ = ? 4

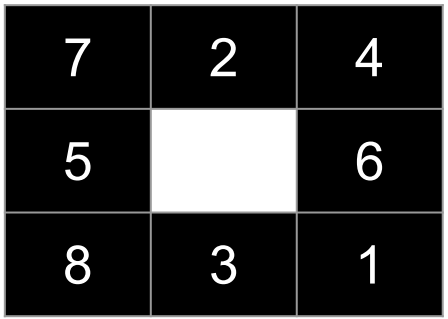
  Is h_1 an admissible heuristic? Yes

Reasoning:

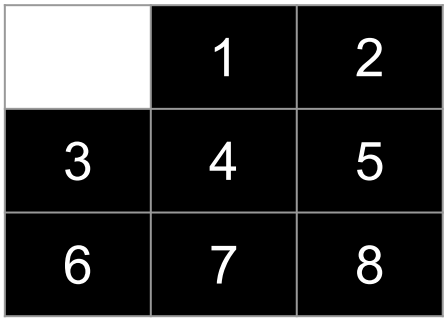
1. Each move in the puzzle can only move a tile to the adjacent empty space.
2. If a tile is misplaced, it must be moved at least once to its correct position.
3. Therefore, the number of misplaced tiles represents a lower bound on the number of moves required to reach the goal state because it does not account for the additional moves required to position the tiles relative to each other.

Example: Heuristic functions for the 8-puzzle problem

- Heuristic function
  - $h_2(n)$ = the sum of the distances of the tiles from their goal positions (i.e., Manhattan distance: the sum of the horizontal and vertical distances.)



Start



Goal

- E.g., $h_2(S)$ = ? 3+1+2+2+2+3+3+2=18

  Is h_2 an admissible heuristic? Yes

(Refer to the reasoning in the previous slide)

(Monotonic)

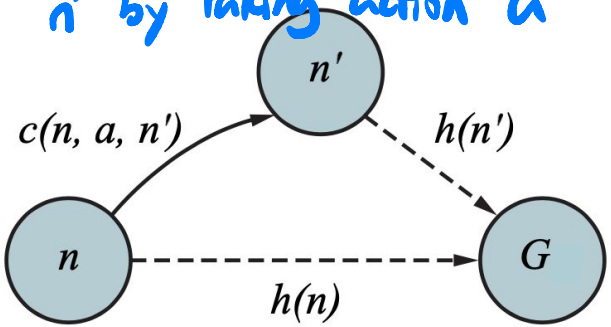
Consistent Heuristics

- A heuristic $h(n)$ is **consistent** if, for every node n and every successor n' of n generated by an action a, we have

  ↗ step cost from n to n' by taking action a

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

  and $h(G) = 0$



- E.g., $h_{SLD}(n)$? Yes

  (SLD: straight-line distance
  triangle inequality)

the first time we visit any state, we are guaranteed to reach it by the shortest path / we won't later find a shorter path through that same node

A* is cost-optimal with a consistent heuristic.

(Hint: Every consistent heuristic is admissible), the opposite is not necessarily true

However, for a heuristic, admissibility doesn't necessarily make the search efficient

e.g. Let $h(s) = 0$

Priority of $S = g(S) + h(S) = g(S) + 0 = g(S)$

$\equiv$ Uniform Cost Search (UCS)

$\Rightarrow$ Consistency ensures the algorithm's performance

If a heuristic is consistent, it must be admissible.

(Hint: proof by induction on the number k of nodes on the shortest path to any goal from n)

proof.

For $k = 1$, let n' be the goal node, then

$$h(n) \le c(n,a,n') + h(n') \le c(n,a,n') = h^*(n).$$

Let n' be on the shortest path k steps from the goal, and suppose $h(n')$ is admissible.

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

So $h(n)$ at $k + 1$ steps from the goal is also admissible.