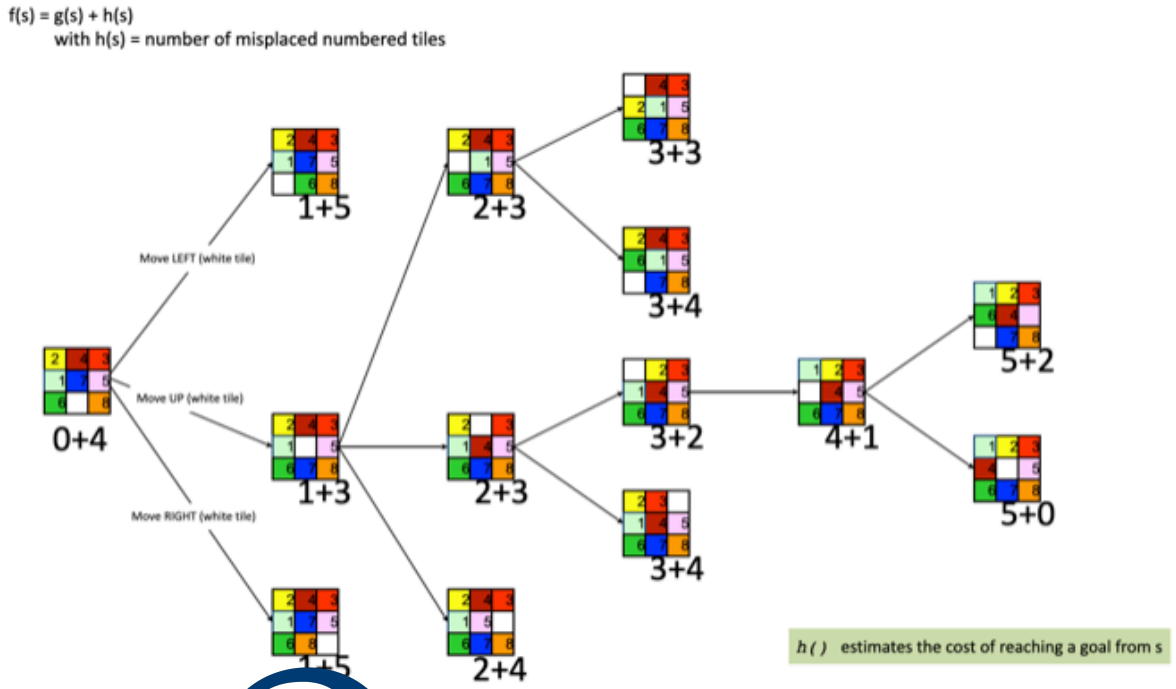
CS143: Artificial Intelligence

Uninformed Search vs. Informed Search
A* Search



Drake UNIVERSITY

Categorizing Search Strategies

- Search algorithms we have seen so far
  - Breadth-First Search (BFS)
  - Depth-First Search (DFS)
  - Iterative Deepening Search (IDS)
  - Uniform Cost Search (UCS)

Two Types of Search Strategies

- Uninformed ("blind") search:
  - algorithms have no additional information about states.
  - all they can do is generate successors and determine if they are goal states
    - Breadth-First Search (BFS)
    - Depth-First Search (DFS)
    - Iterative Deepening Search (IDS)

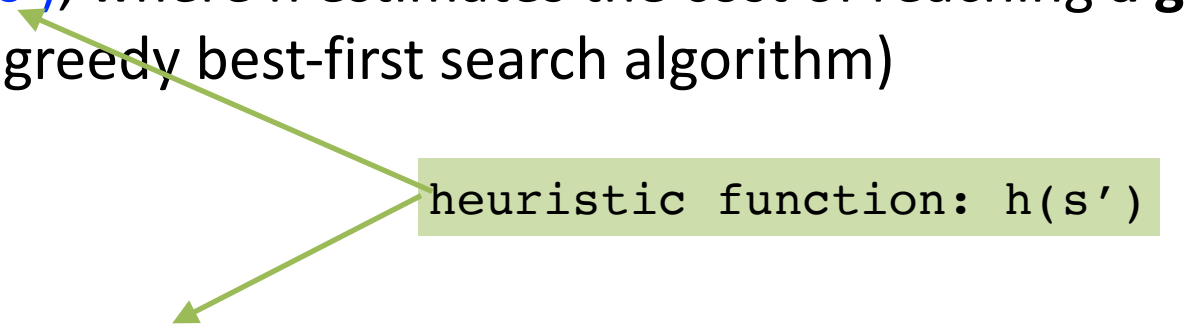
- Informed ("heuristic") search:
  - more information given about whether a non-goal state is "more promising" with respect to a goal state
    - Greedy Best-First Search
    - A* Search

Informed (Heuristic) Search

- Explores most 'promising' state from FRONTIER first
  - typically implemented with a priority queue
  - requires *evaluation function* $f(s') \geq 0$ that estimates the **"cost" from initial state, through s, to a goal state**

Informed (Heuristic) Search

- Typical choices for the evaluation function f(s'):
  - f(s') = h(s'), where h estimates the cost of reaching **a goal from** s' (greedy best-first search algorithm)



  - f(s') = g(s') + h(s'), where g() is **cost of path from start state to s** and heuristic function h() estimates the **cost of reaching a goal from** s' (A* search algorithm)

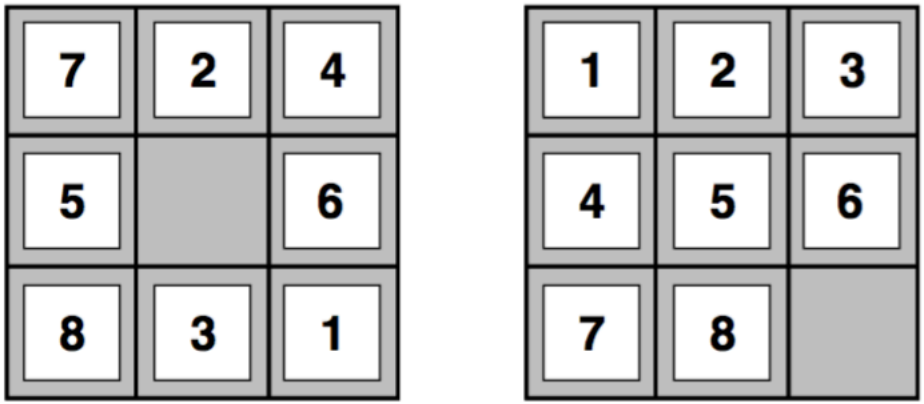
Informed (Heuristic) Search

- Greedy Best-First Search (idea 1):
  - always expand the node that appears closest to goal

- A* Search (idea 2):
  - expand the node along what appears to be the best overall path (path cost + heuristic)

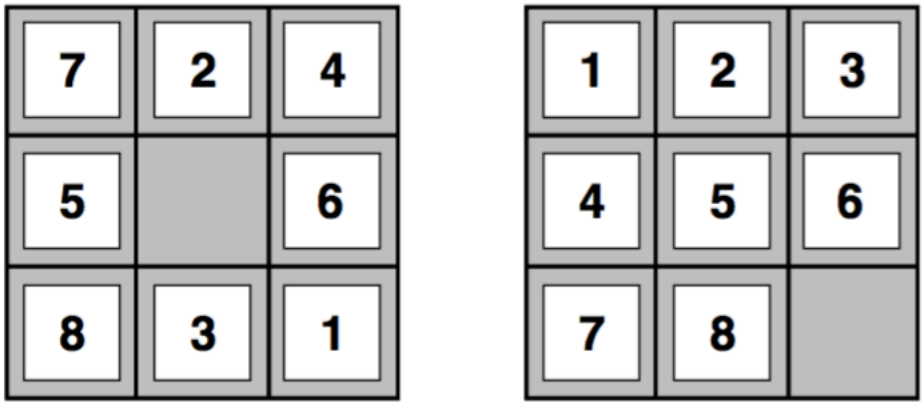
Heuristic Function Example

- Let's consider 8-puzzle problem
- What heuristic to use that will estimate how close we are to a solution?



Heuristic Function Example

- $h_1(n)$ = number of misplaced tiles (not including empty tile)
- $h_2(n)$ = total Manhattan distance (number of squares from desired location of each tile)



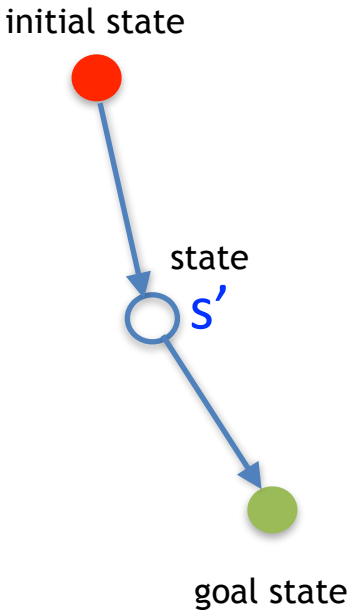
Start State Goal State

$h_1(n)$ = ?

$h_2(n)$ = ?

A* Search

- Main idea: avoid expanding paths that are already expensive



- Use an evaluation function f(s'):
  - f(s') = g(s') + h(s')
  - g(s') = cost of path from initial state to s'
  - h(s') = cost of reaching a goal state from s'

`heuristic function: h(s')`

- f(s') measures an estimated total cost of path through s' to goal

A* Search

- Use Graph Search algorithm by making three changes:
  - use priority queue as data structure instead of stack/queue
  - Insert a node s' into the FRONTIER after computing its evaluation function f(s') = g(s') + h(s')
  - remove a node s from the FRONTIER based on the lowest estimate of f(s) among all the nodes inside the FRONTIER

A* Search

Use priority queue for FRONTIER

1. if GOAL == (initial-state) then return initial-state
2. INSERT(initial-node, FRONTIER)
3. repeat:
4. if empty(FRONTIER) then return failure
5. s ← REMOVE(FRONTIER)
6. if GOAL ==s then return s and/or path
7. for every state s' in Successor(s):
8. INSERT(s', FRONTIER)

REMOVE node s with minimum value of f()

Compute f(s') then INSERT s' into FRONTIER
f(s') = g(s')+ h(s')

A* Search for 8-puzzle

- For A* search, we need to compute h(s') for a state s' before inserting it into the FRONTIER
- Let's consider: h(s') = number of misplaced tiles (not including empty tile)

state **s'**

Is tile#1 misplaced? =	1
Is tile#2 misplaced? =	1
Is tile#3 misplaced? =	0
Is tile#4 misplaced? =	1
Is tile#5 misplaced? =	0
Is tile#6 misplaced? =	0
Is tile#7 misplaced? =	1
Is tile#8 misplaced? =	0

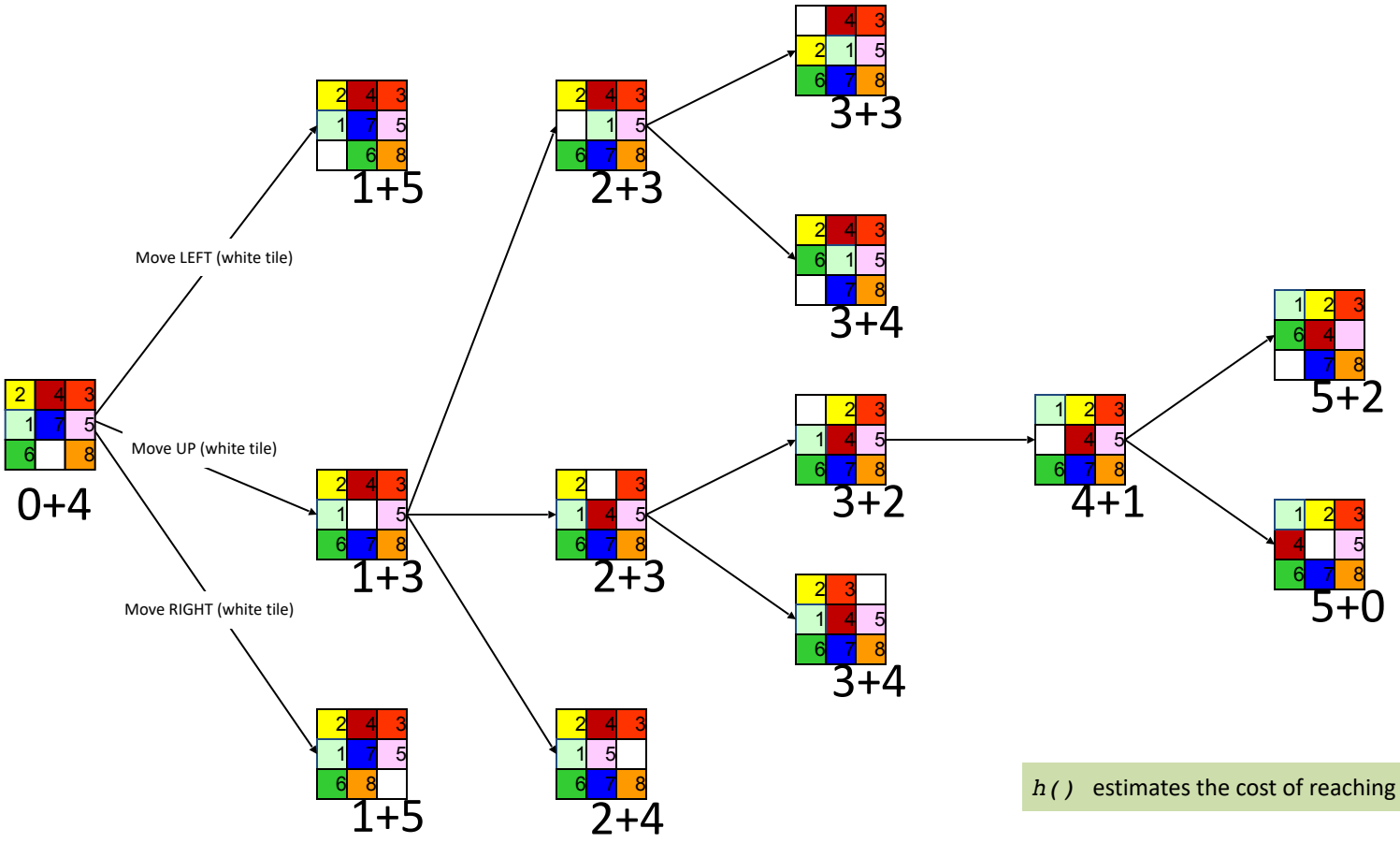
h(s') = total # of misplaced tiles = 4

Goal state

h( ) estimates the cost of reaching a goal from s'

A* Search

f(s') = g(s') + h(s')
with h(s') = number of misplaced numbered tiles



h( ) estimates the cost of reaching a goal from s'

A* Search

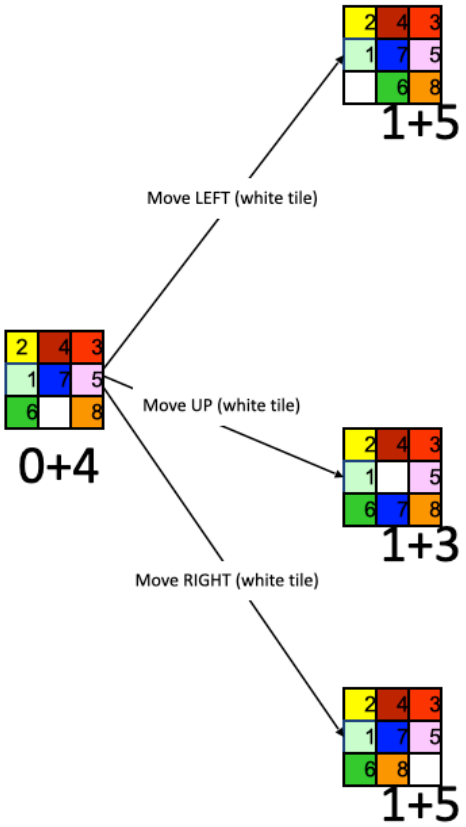
f(s) = g(s) + h(s)
with h(s) = number of misplaced numbered tiles

0+4

$h()$ estimates the cost of reaching a goal from s'

A* Search

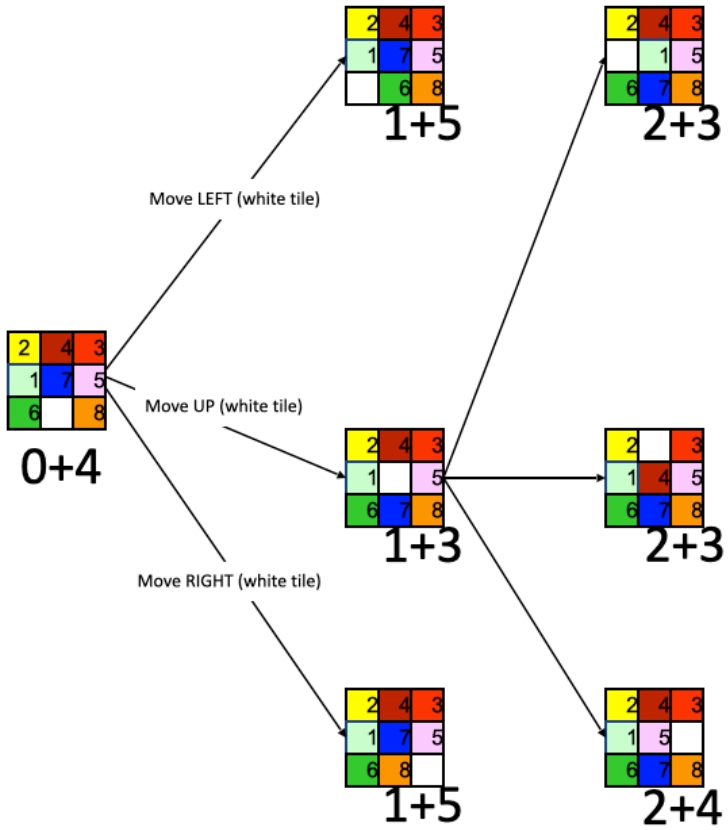
f(s) = g(s) + h(s)
with h(s) = number of misplaced numbered tiles



`h()` estimates the cost of reaching a goal from s'

A* Search

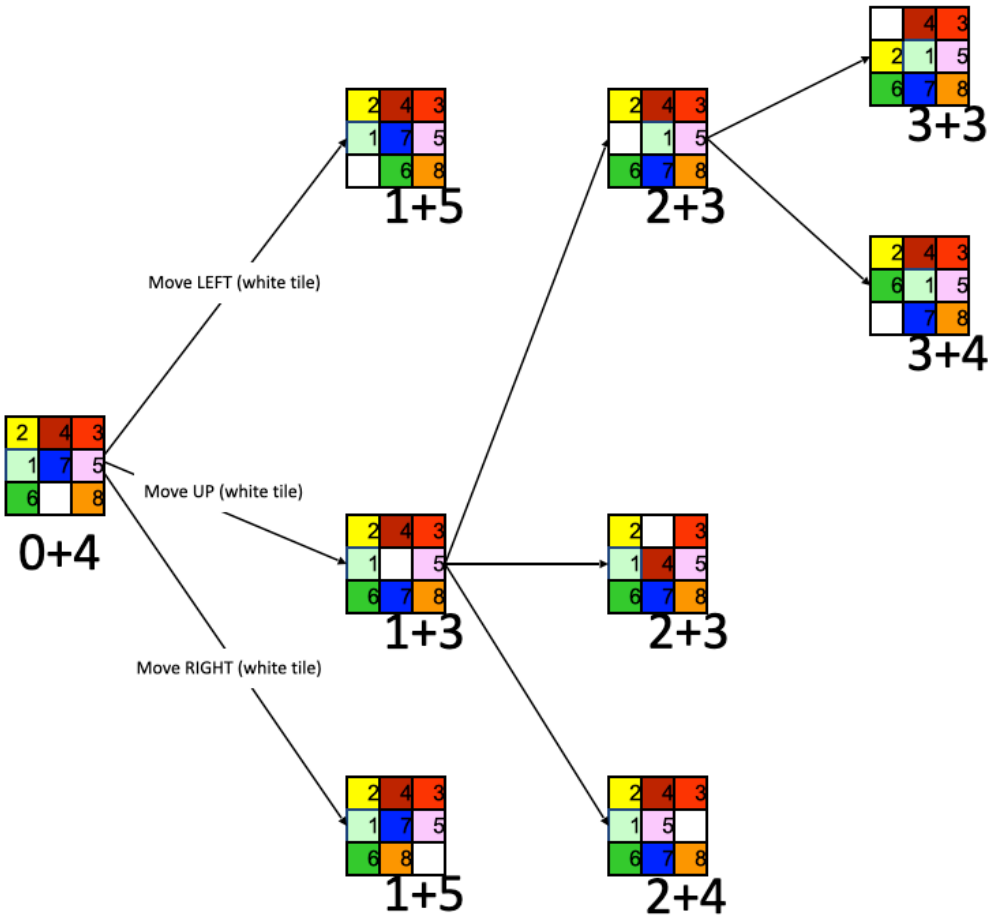
f(s) = g(s) + h(s)
with h(s) = number of misplaced numbered tiles



$h()$ estimates the cost of reaching a goal from s'

A* Search

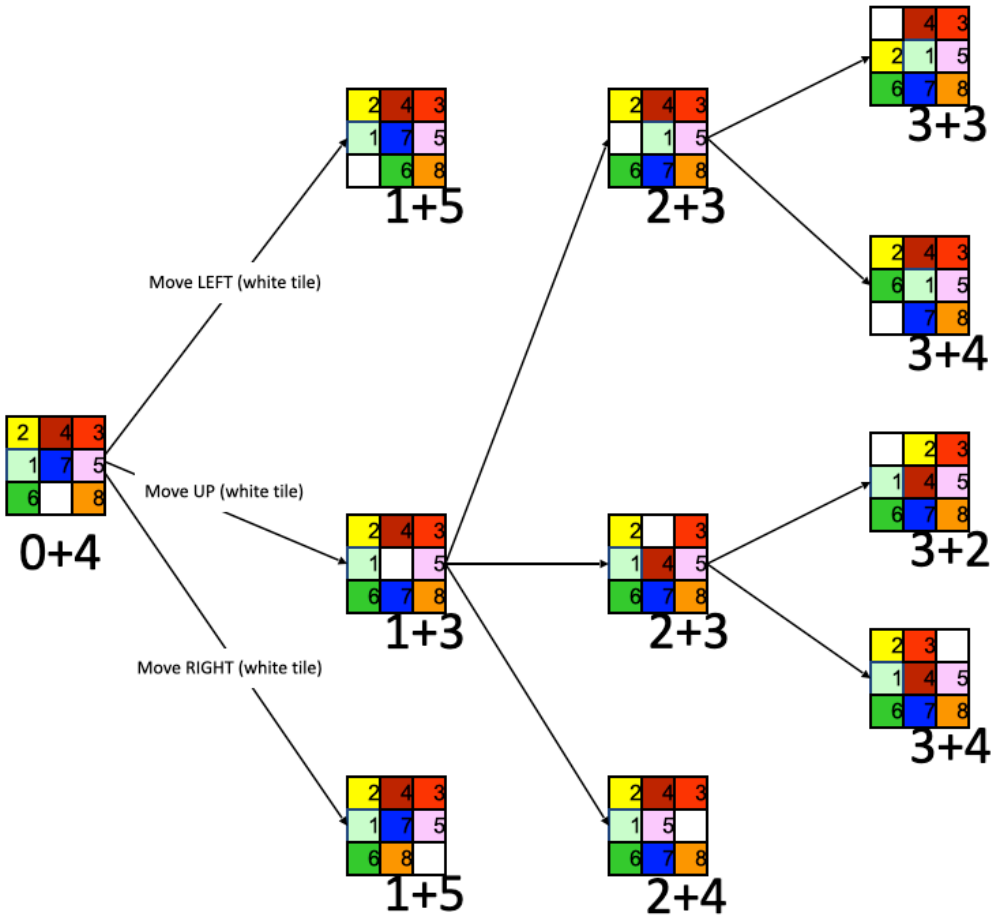
f(s) = g(s) + h(s)
with h(s) = number of misplaced numbered tiles



h( ) estimates the cost of reaching a goal from s'

A* Search

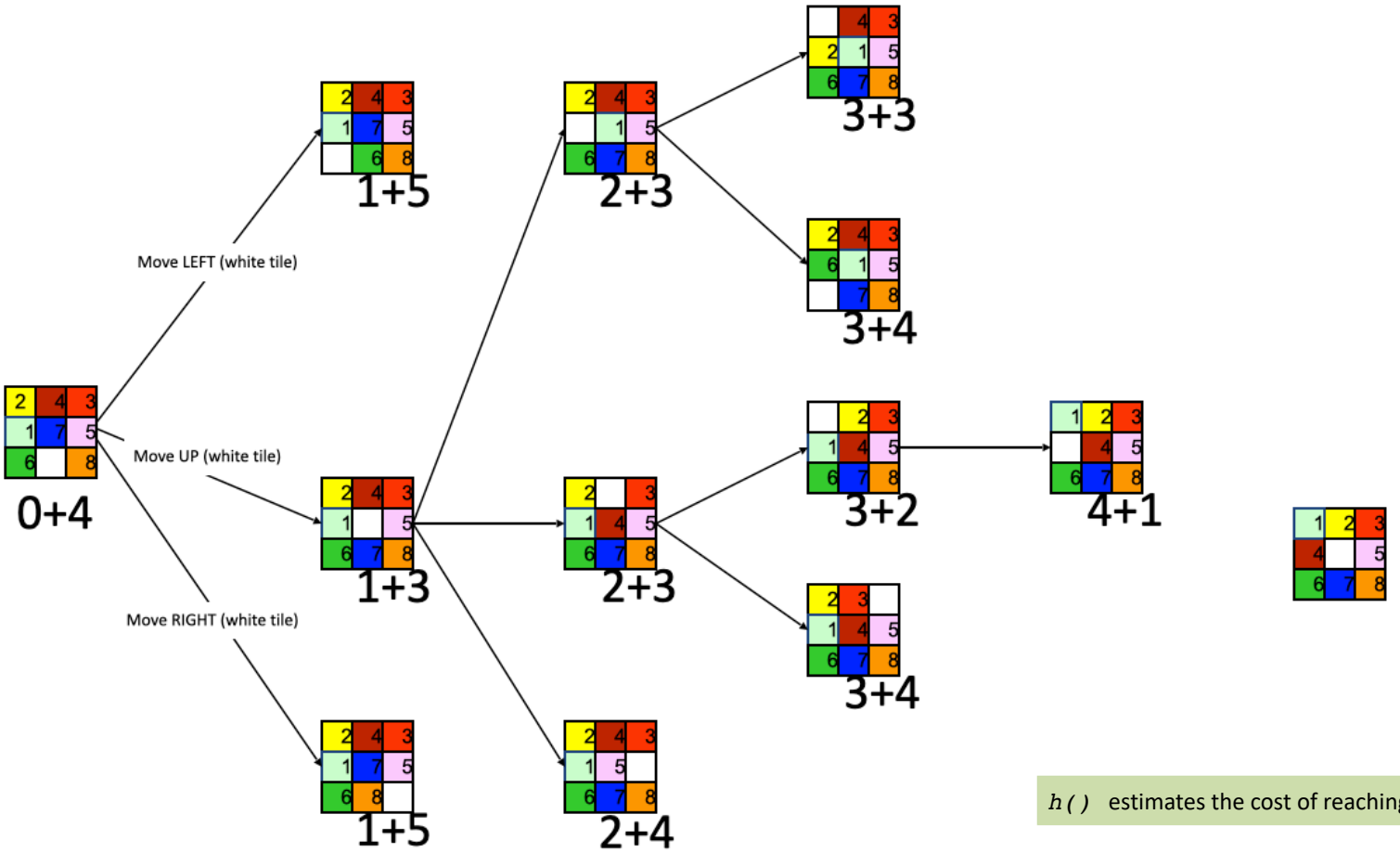
f(s) = g(s) + h(s)
with h(s) = number of misplaced numbered tiles



h( ) estimates the cost of reaching a goal from s'

A* Search

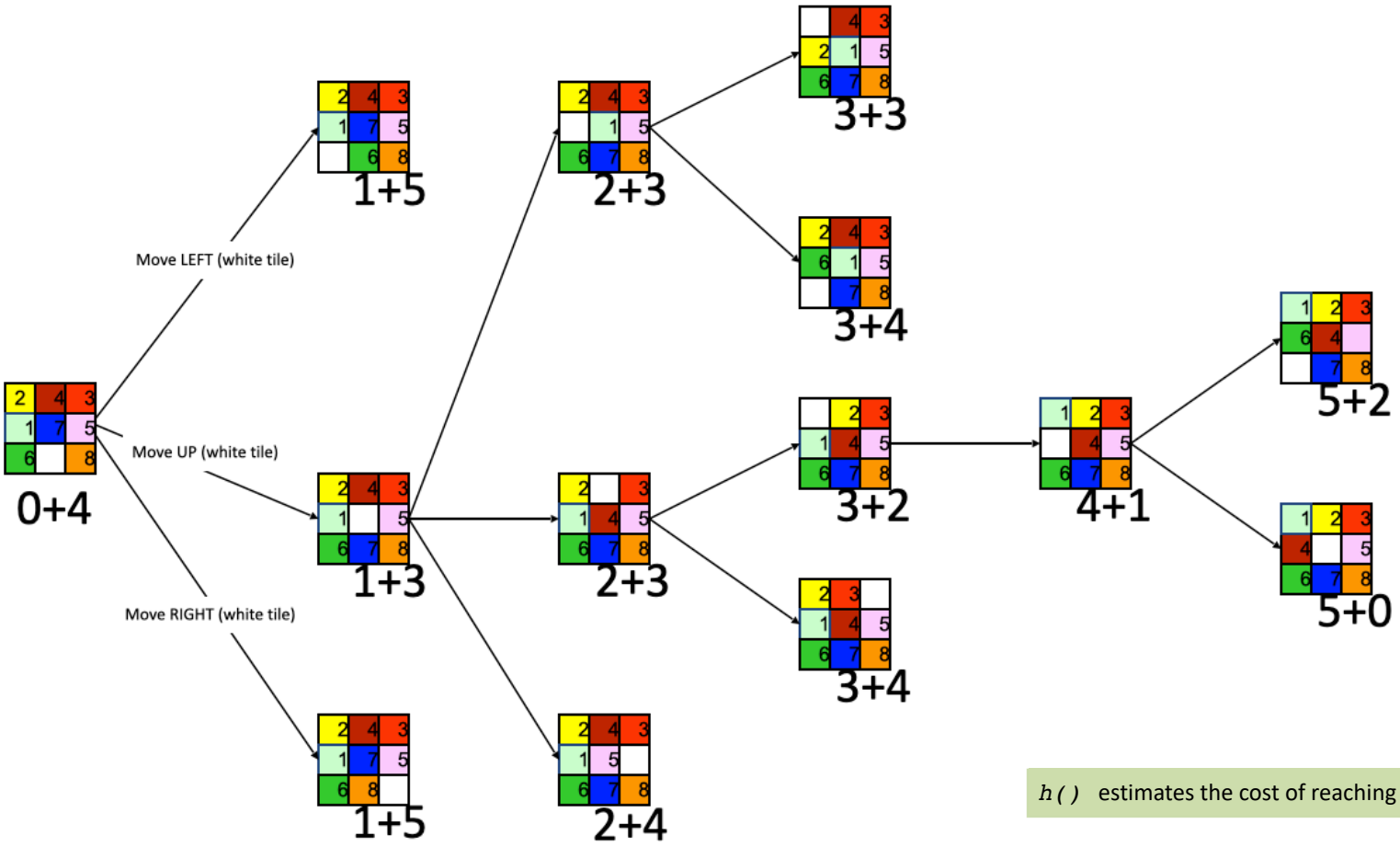
f(s) = g(s) + h(s)
with h(s) = number of misplaced numbered tiles



h( ) estimates the cost of reaching a goal from s'

A* Search

f(s) = g(s) + h(s)
with h(s) = number of misplaced numbered tiles



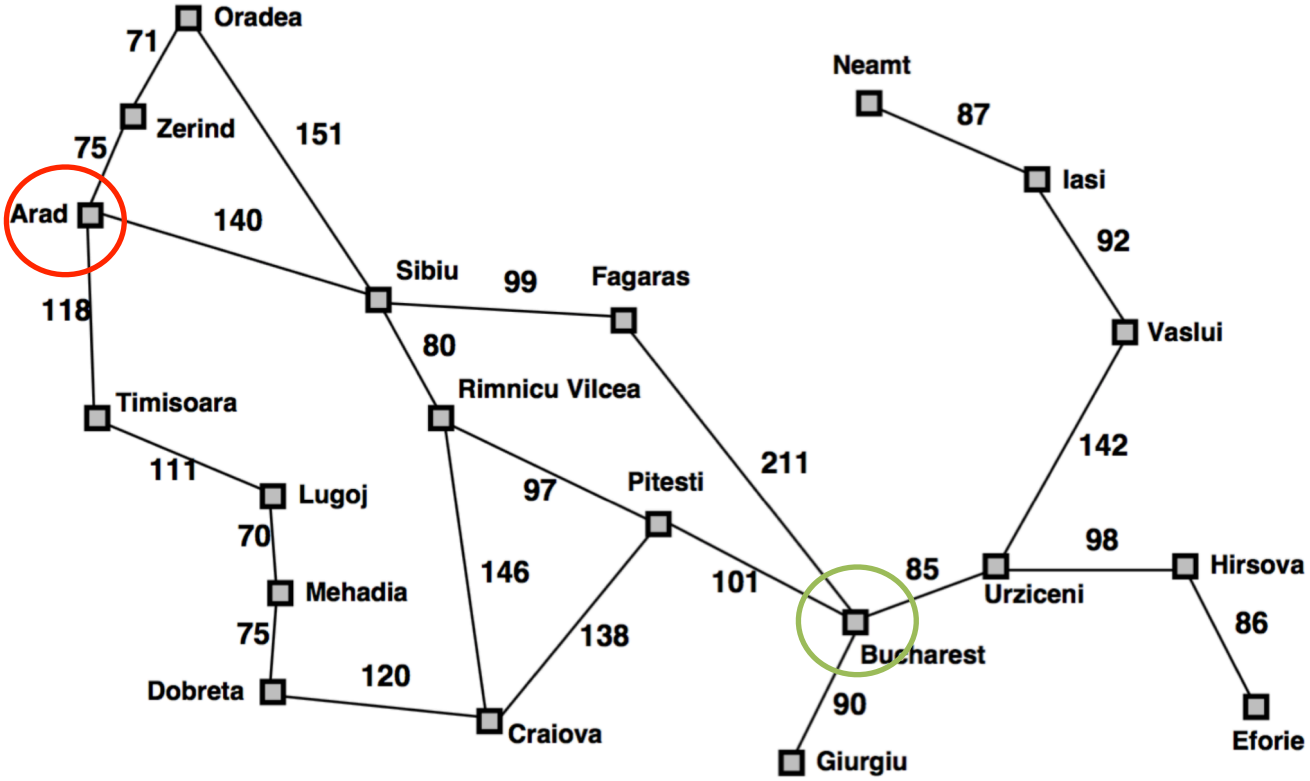
$h(\,)$ estimates the cost of reaching a goal from s'

Class activity: A* Search

A* Search

- Starting from city 'Arad', create the search tree using A* search
  - Initial state = 'Arad'
  - Goal state = 'Bucharest'

`evaluation function: g(s') + h(s')`



Heuristic function

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

Heuristic function

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

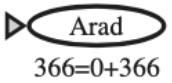
A* Search

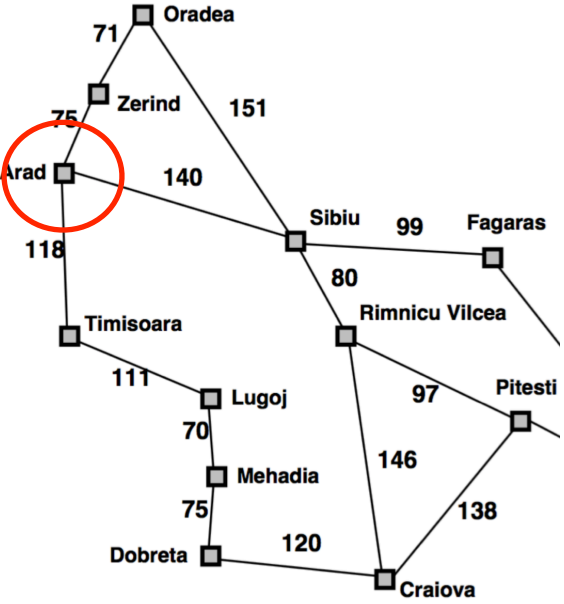
Starting from city 'Arad', create the search tree using A* search

- Initial state = 'Arad'
- Goal state = 'Bucharest'

```
evaluation function: g(s') + h(s')
```

Initial state





Expand the node along what appears to be the best overall path
path cost from source to the node + heuristic value at the node

A* Search

Heuristic function

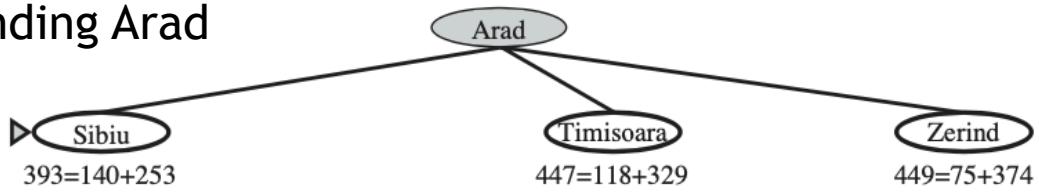
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

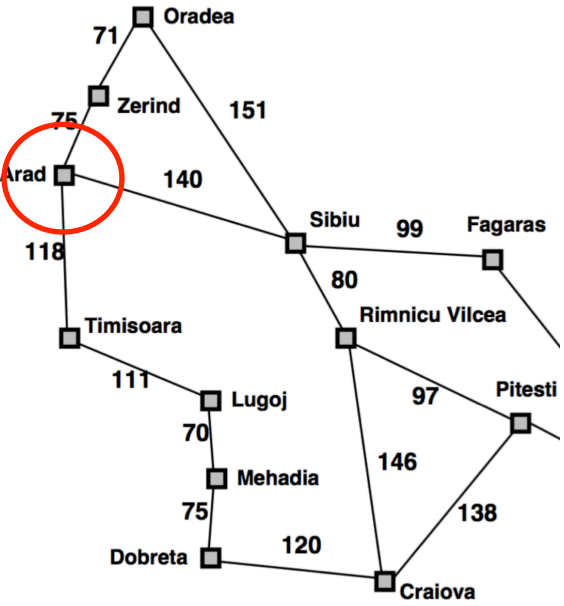
Starting from city 'Arad', create the search tree using A* search

- Initial state = 'Arad'
- Goal state = 'Bucharest'

`evaluation function: g(s') + h(s')`

After expanding Arad





Expand the node along what appears to be the best overall path
path cost from source to the node + heuristic value at the node

Heuristic function

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

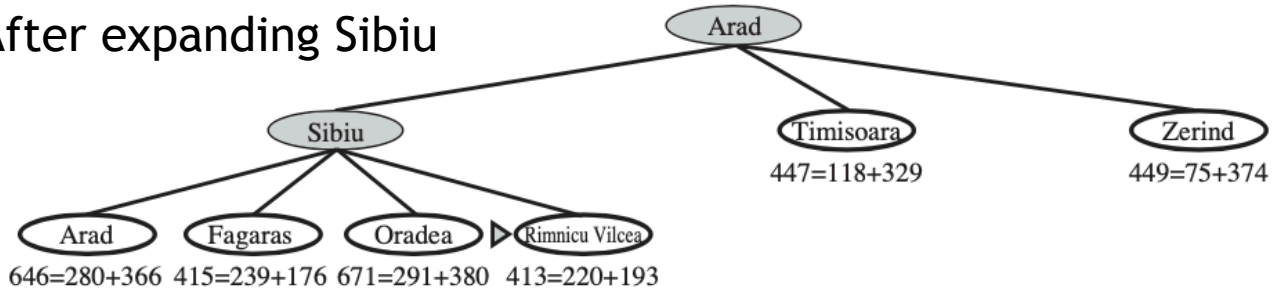
A* Search

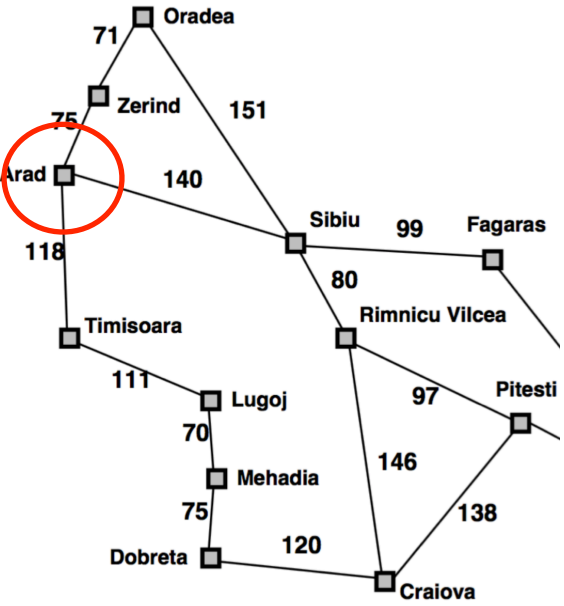
Starting from city 'Arad', create the search tree using A* search

- Initial state = 'Arad'
- Goal state = 'Bucharest'

`evaluation function: g(s') + h(s')`

After expanding Sibiu





Expand the node along what appears to be the best overall path
path cost from source to the node + heuristic value at the node

A* Search

Heuristic function

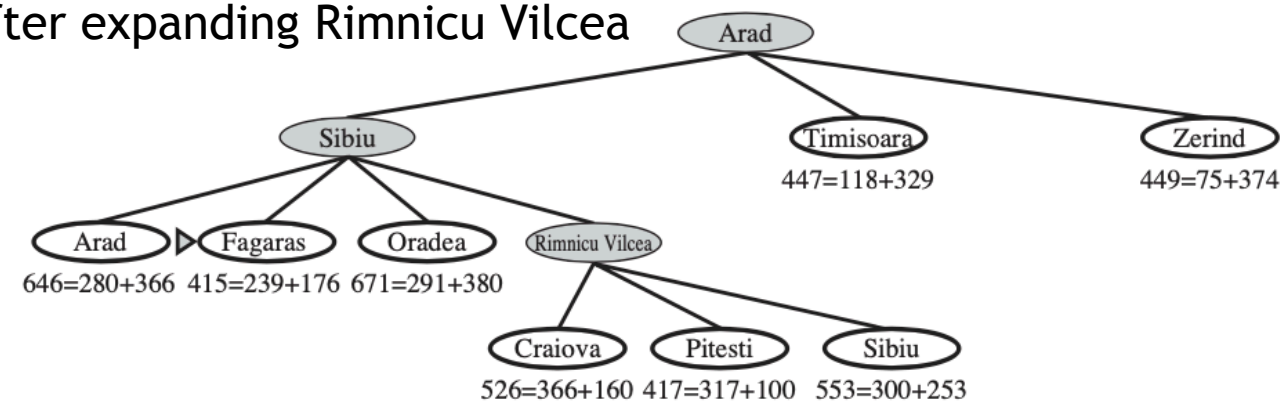
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

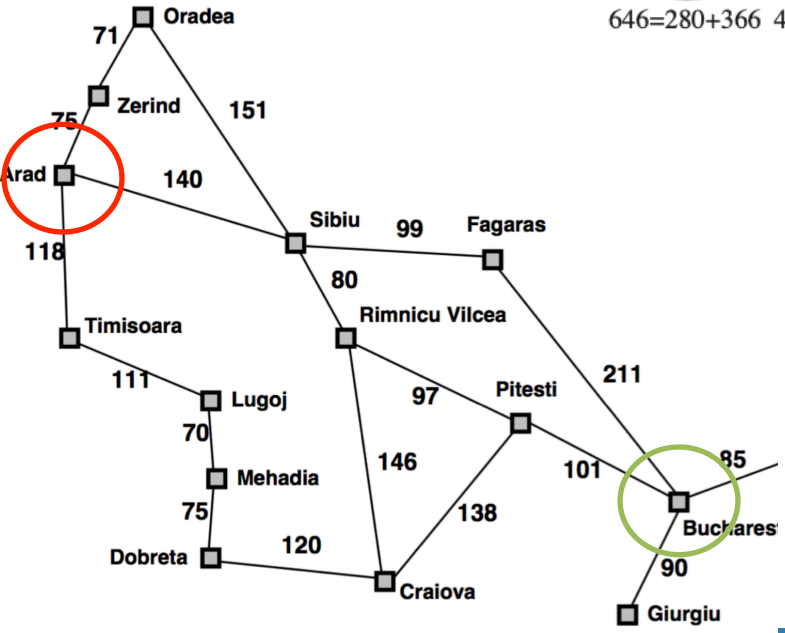
Starting from city 'Arad', create the search tree using A* search

- Initial state = 'Arad'
- Goal state = 'Bucharest'

`evaluation function: g(s') + h(s')`

After expanding Rimnicu Vilcea





Expand the node along what appears to be the best overall path
path cost from source to the node + heuristic value at the node

A* Search

Heuristic function

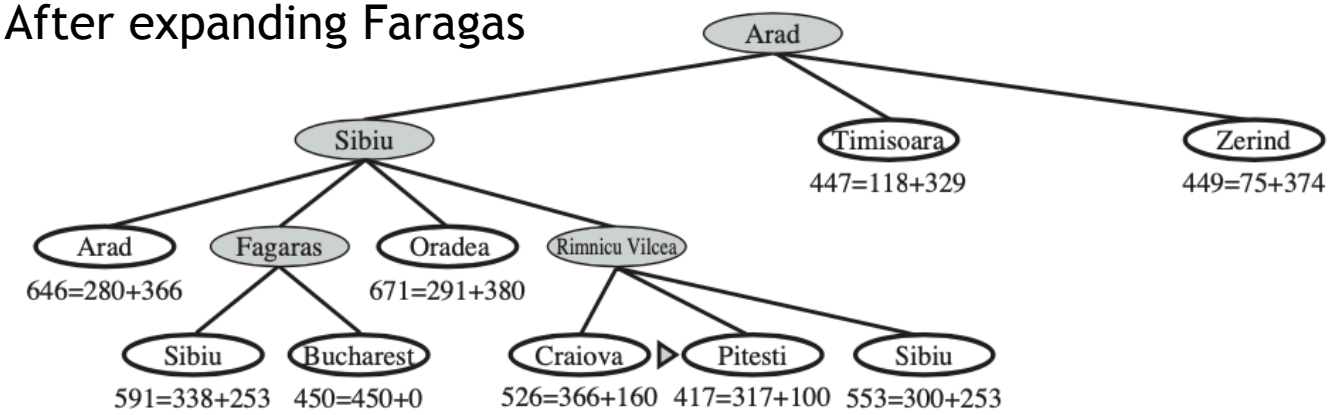
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

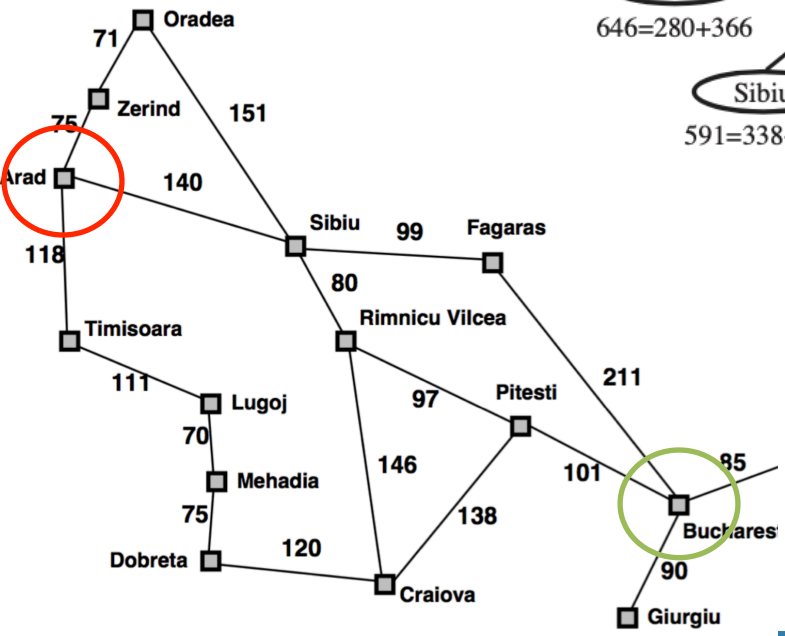
Starting from city 'Arad', create the search tree using A* search

- Initial state = 'Arad'
- Goal state = 'Bucharest'

`evaluation function: g(s') + h(s')`

After expanding Faragas





Expand the node along what appears to be the best overall path
path cost from source to the node + heuristic value at the node

A* Search

Heuristic function

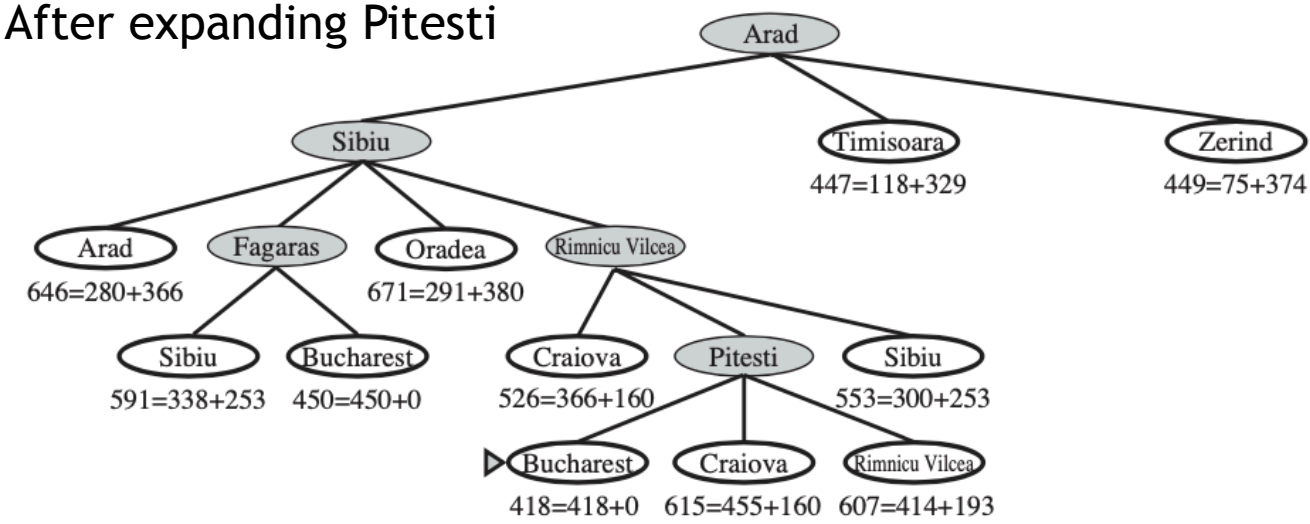
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

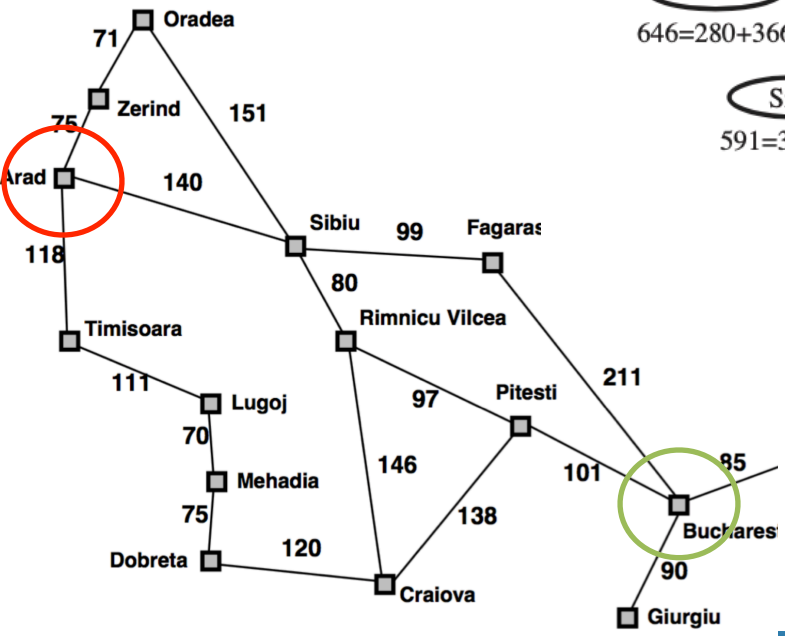
Starting from city 'Arad', create the search tree using A* search

- Initial state = 'Arad'
- Goal state = 'Bucharest'

`evaluation function: g(s') + h(s')`

After expanding Pitesti





History of A* Search

- In 1964 Nils Nilsson invented a heuristic-based approach to increase the speed of Dijkstra's algorithm.
  - This algorithm was called A1.

- In 1967 Bertram Raphael made dramatic improvements upon this algorithm, but failed to show optimality.
  - He called this algorithm A2.

History of A* Search

- Then in 1968 Peter E. Hart introduced an argument that proved A2 was **optimal** when using a **consistent** heuristic with only minor changes.

- His proof of the algorithm also included a section that showed that the new A2 algorithm was the **best algorithm possible given the conditions**.

- He thus named the new algorithm in Kleene star syntax
  - algorithm that starts with A and includes all possible version numbers or A*.

Properties of A*

- **Is A* Complete?**

  Complete unless there are infinitely many nodes n with $f(n) \leq f(G)$.

- **Is A* Optimal?**

  Optimal

- **What is the running time complexity of A*?**

  Time complexity is exponential in (relative error in h $\times$ length of the solution

- **What are the memory requirement of A*?**

  Memory complexity is exponential: keeps all nodes in memory

Useful Application of A*

- **Path planning or route planning**
- **Robot motion planning**
- **Resource planning**
- **Language analysis**
- **Video games**
- **Machine translation**