SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

# Informed Search

PROF. LIM KWAN HUI

50.021 Artificial Intelligence

*The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources.*

# Outline & Objectives

- Learn about two different informed search algorithms
  - Greedy Best-First Search
  - A* Search
- Understand the role that heuristic plays in these algorithms
- Being able to use Greedy Best-First Search and A* Search to solve a search problem

# Recap: Types of Search

- Uninformed Search
  - No additional information about states beyond that in the problem definition (AKA blind search)
- Informed Search
  - Uses problem-specific knowledge beyond the definition of the problem itself
- Adversarial Search
  - Used in multi-agent environment where the agent needs to consider the actions of other agents and how they affect its own performance.

# Recap: Search Strategies

- A **search strategy** is defined by picking the *order of node expansion*
- How do we implement this?
  - Using different types of queue structures to represent the frontier
  - *Order of node expansion = Adding/removal sequence in queue*

***Expand step: Determines our search strategy***

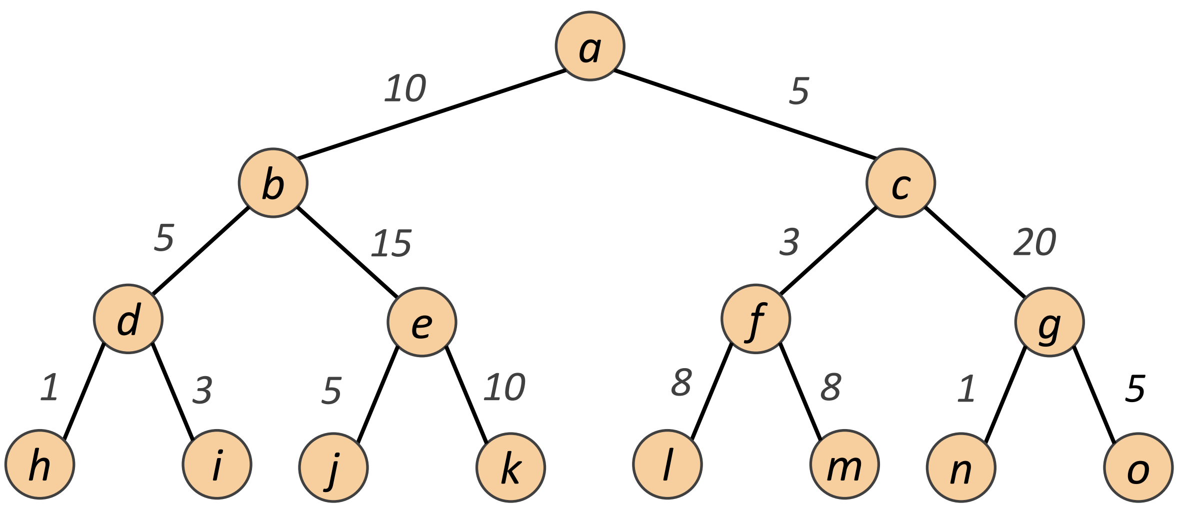
```
function GRAPH-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  initialize the explored set to be empty
  loop do
     if the frontier is empty then return failure
     choose a leaf node and remove it from the frontier
     if the node contains a goal state then return the corresponding solution
     add the node to the explored set
     expand the chosen node, adding the resulting nodes to the frontier
        only if not in the frontier or explored set
```

# Recap: Uninformed Search

- Breadth-First Search
- Uniform-cost search
- Depth-First Search
- Depth-limited search
- Iterative deepening search

# Uniform Cost Search

- General idea: Expand unexpanded node *n* with the *lowest path cost g(n)*
- Implementation: Use a *priority* queue ordered by *path cost g(n)*

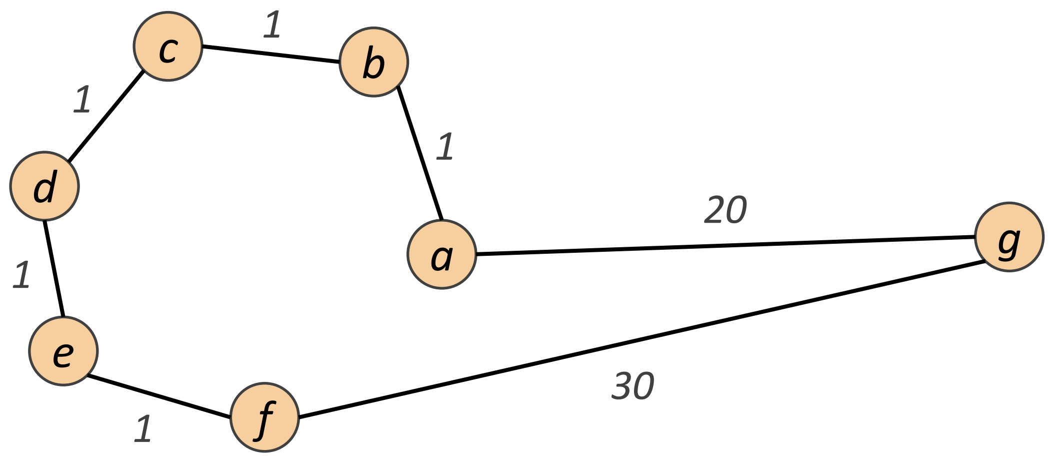


# Uniform Cost Search

- General idea: Expand unexpanded node *n* with the *lowest path cost g(n)*
- Implementation: Use a *priority* queue ordered by *path cost g(n)*
- But is path cost a good performance measure?

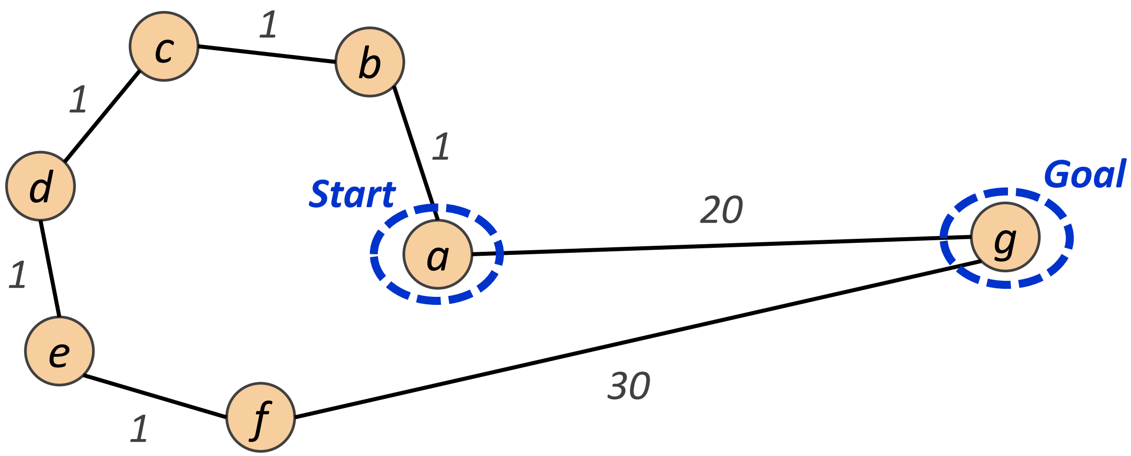
# Uniform Cost Search

- General idea: Expand unexpanded node *n* with the *lowest path cost g(n)*
- Implementation: Use a *priority* queue ordered by *path cost g(n)*
- But is path cost a good performance measure? What if…



# Uniform Cost Search

- General idea: Expand unexpanded node *n* with the *lowest path cost g(n)*
- Implementation: Use a *priority* queue ordered by *path cost g(n)*
- But is path cost a good performance measure? What if…
  - Possible redundant searches

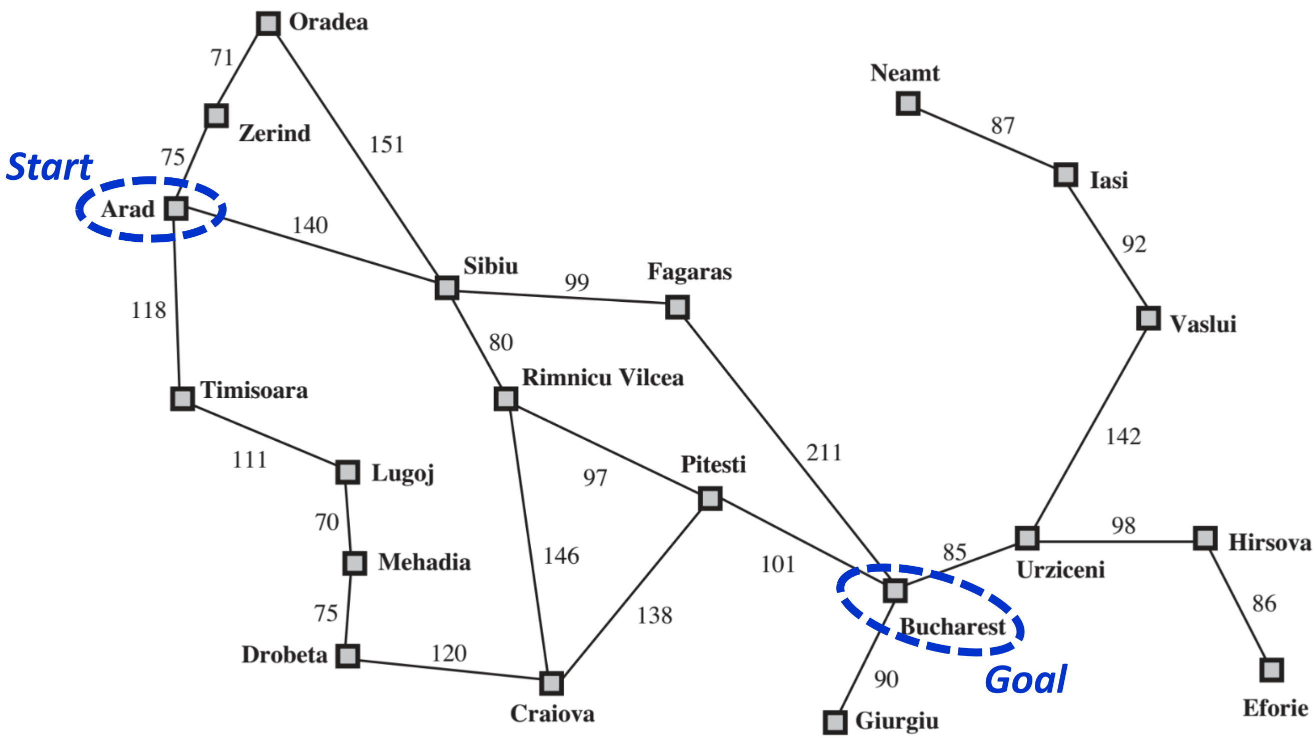


# Improvements to UCS

- UCS idea: Expand unexpanded node *n* with the *lowest path cost g(n)*
  - Formally, this is via an *evaluation function f(n) = g(n)*
- UCS Implementation: Use a *priority* queue ordered by *path cost g(n)*
- ***Evaluation function f(n)***
  - For UCS, path cost measures how far a node is from initial state
  - Is this effective? Does a shorter path from the initial state mean we are nearer to the goal state?

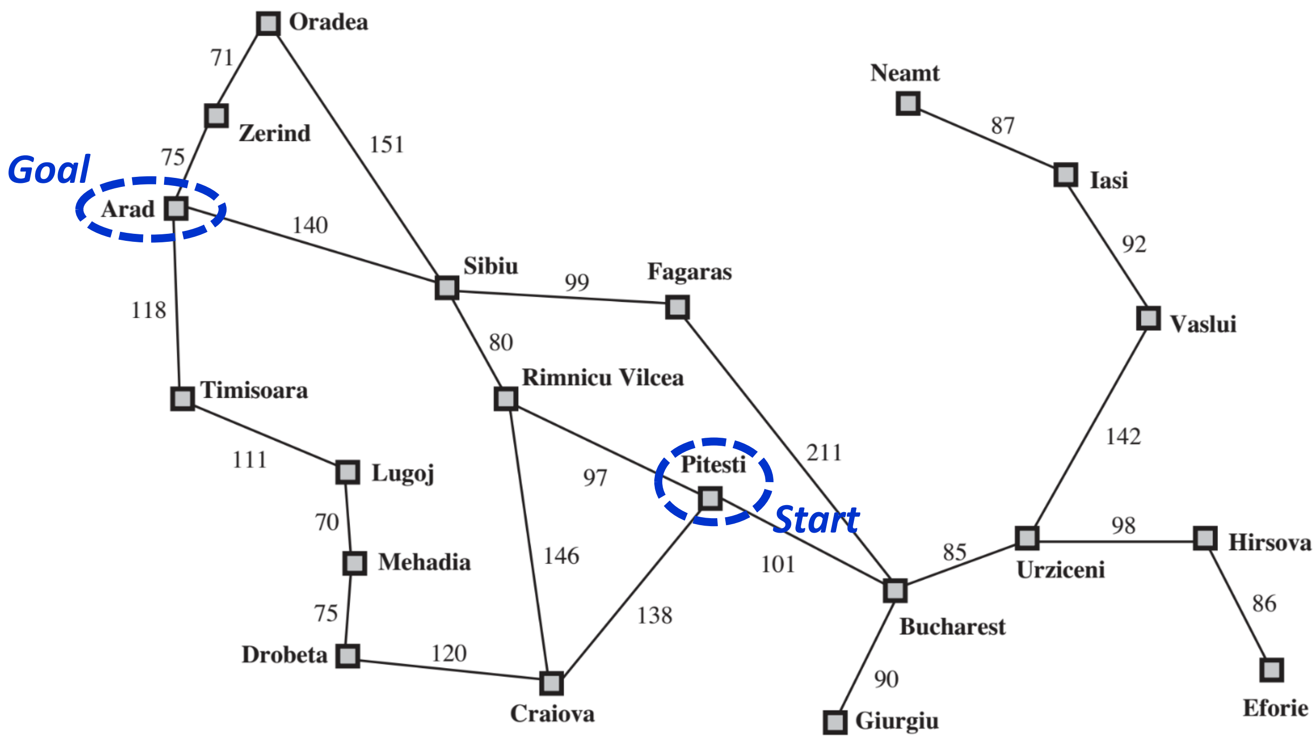
# Revisiting Romania Holiday

- Task: Get to Bucharest from Arad
  - Seems ok using UCS



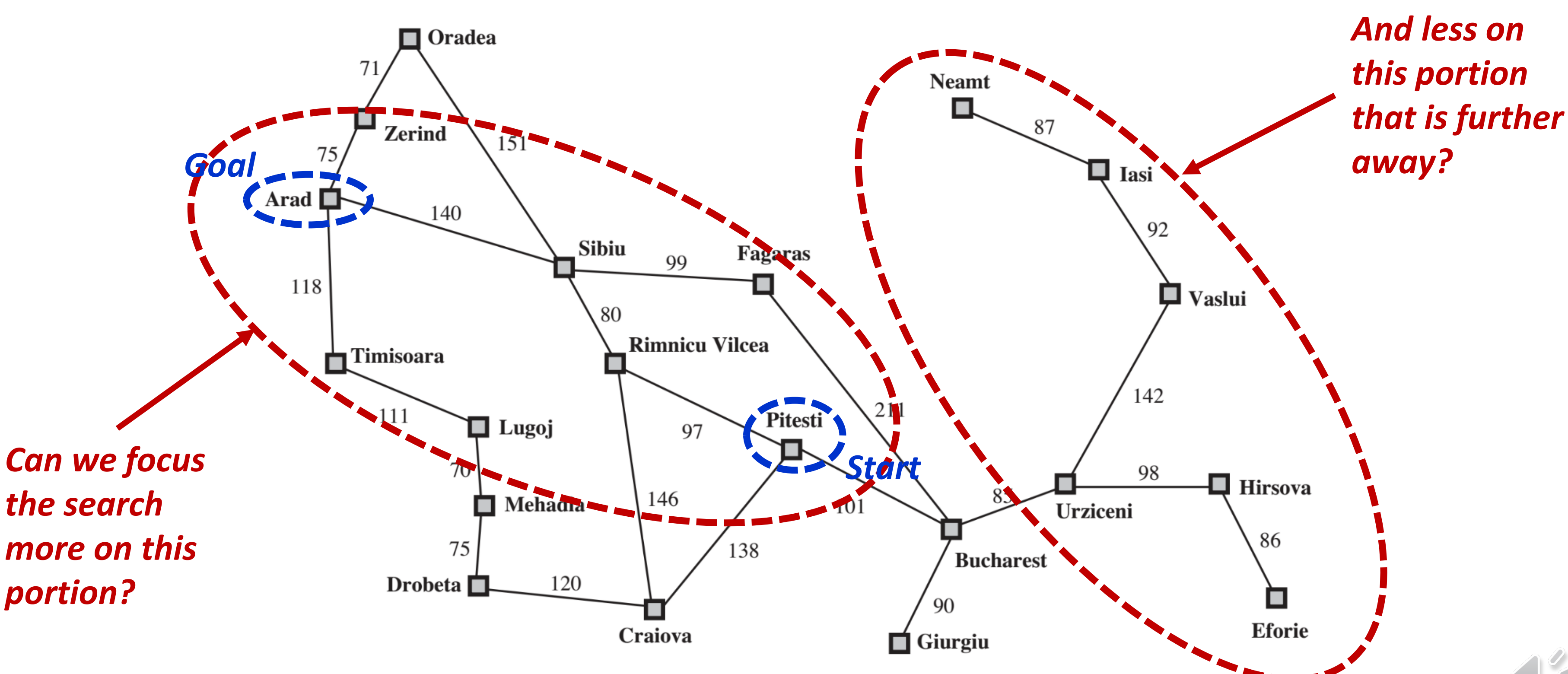
# Revisiting Romania Holiday

- Task: Get to Arad from Pitesti
  - What about using UCS now?



# Revisiting Romania Holiday

- Task: Get to Arad from Pitesti
  - Path cost ignores direction. Is there something else we can use?



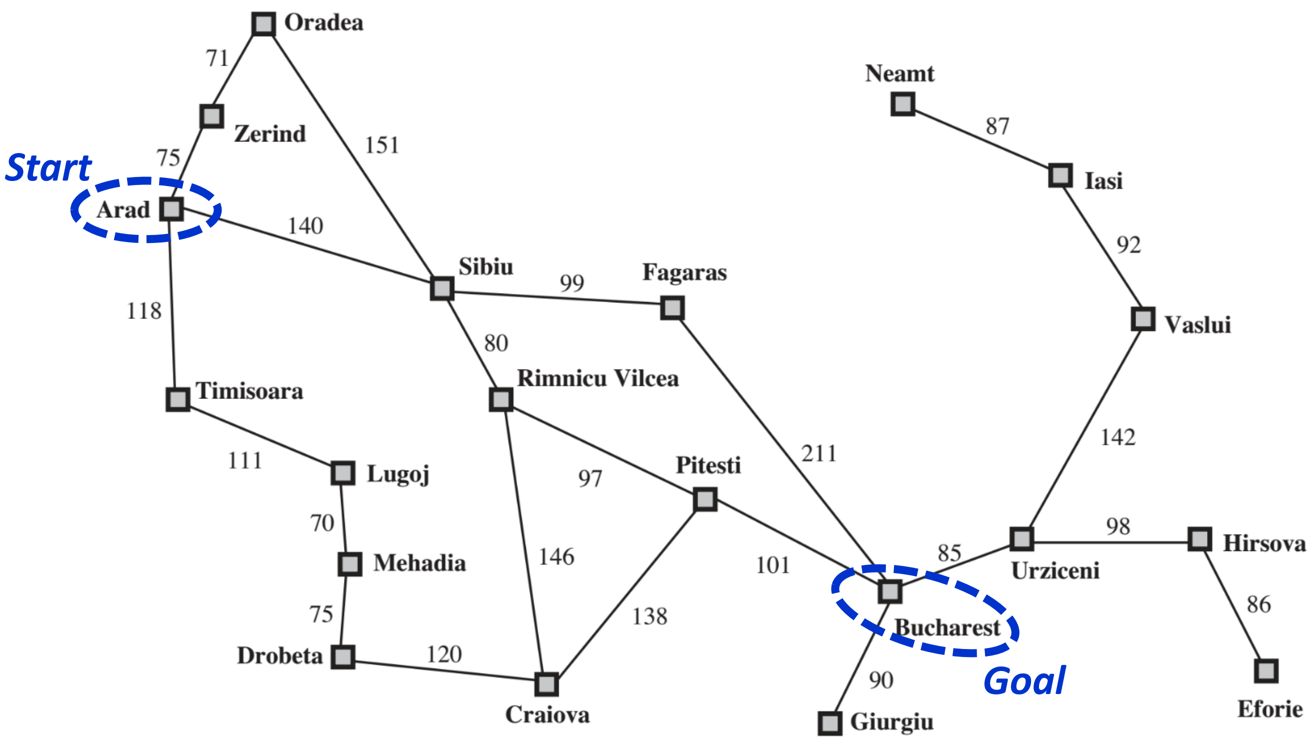
# Heuristics

- What is a heuristic?
  - "A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood." [dictionary definition]

- ***Heuristic function h(n)***
  - The heuristic function h(n) is an *estimate of how close* a state *n* is to the goal state
  - Informed search algorithms uses heuristics to solve the search problem
  - There are good and bad heuristics!

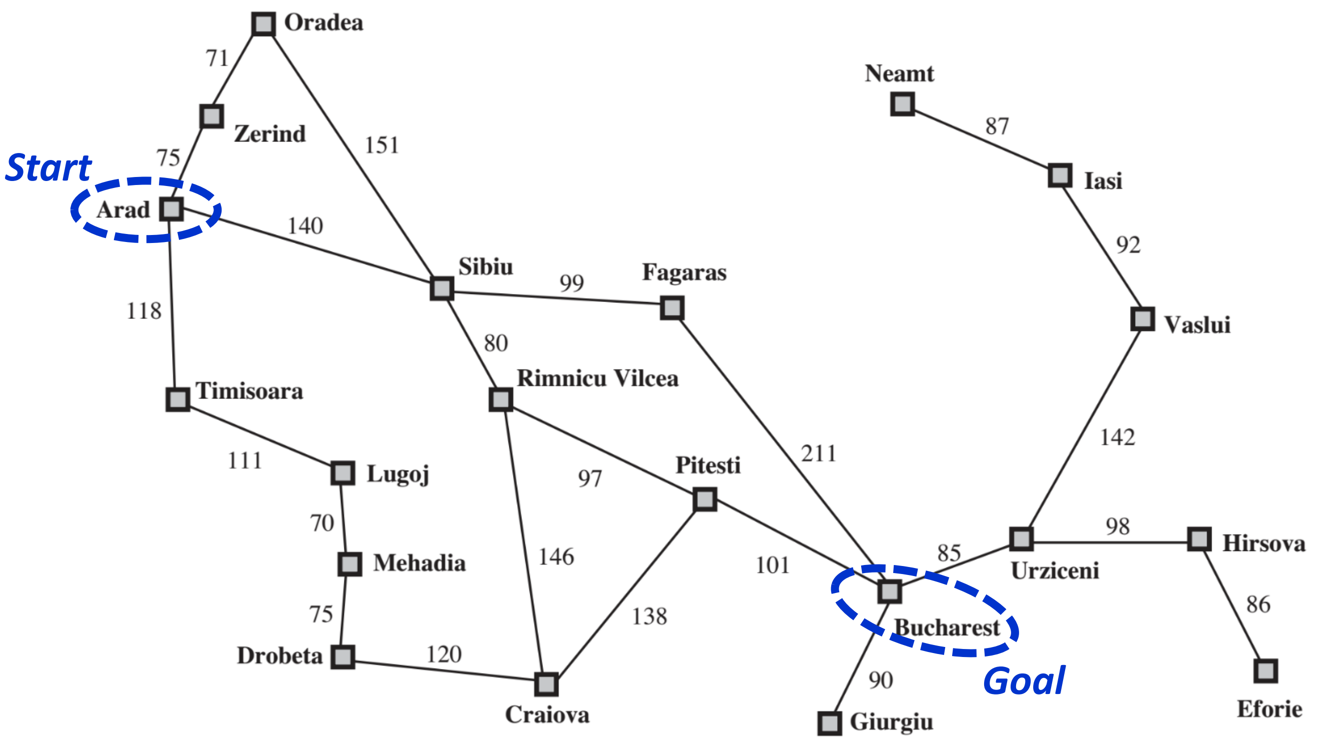
# Revisiting Romania Holiday

- Task: Get to Bucharest from Arad
  - Path cost ignores direction. Is there something else we can use?



# Revisiting Romania Holiday

- Task: Get to Bucharest from Arad
  - Path cost ignores direction. Is there something else we can use?

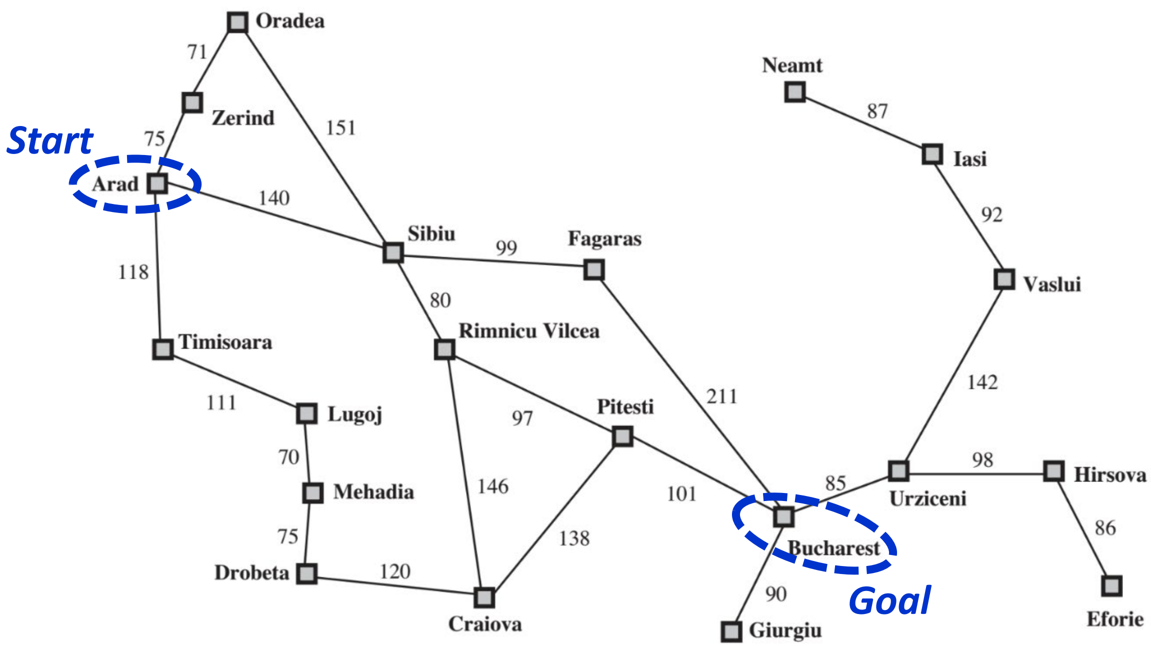



**<u>Straight Line Distances to Bucharest</u>**

| | |
|---|---|
| Arad | 366 |
| Bucharest | 0 |
| Craiova | 160 |
| Drobeta | 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 |

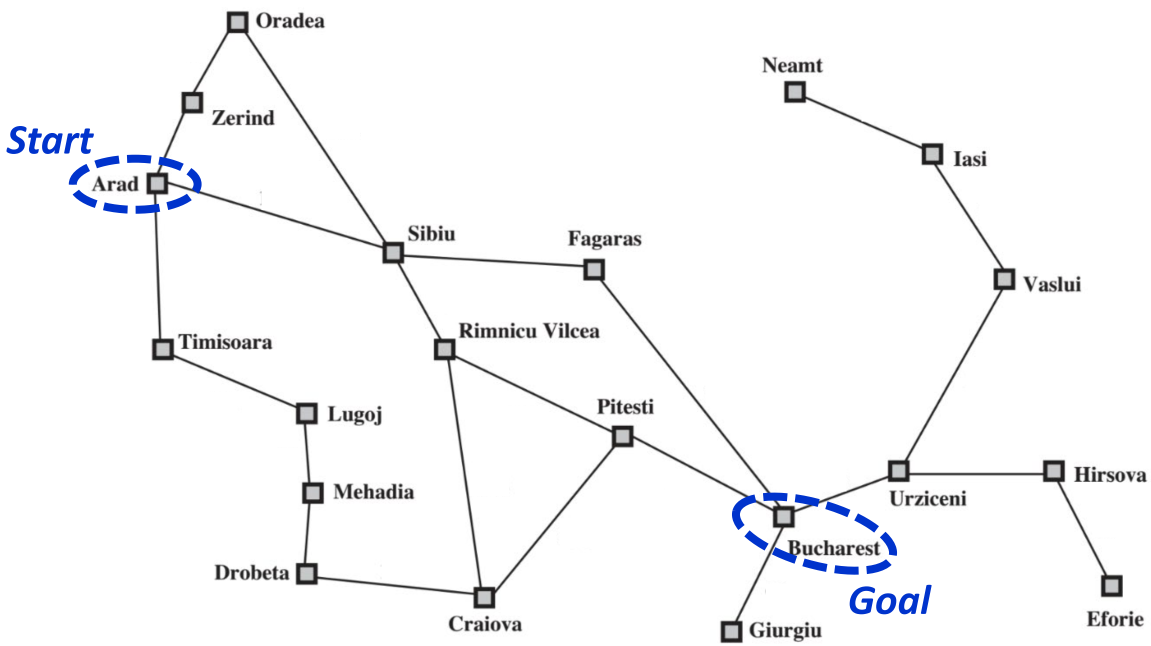
# Recap: Uniform Cost Search

- General idea: Expand unexpanded node *n* with *lowest path cost g(n)*
- Implementation: Use a *priority* queue ordered by *path cost g(n)*
  - Evaluation function *f(n) = g(n)*



# Greedy Best-First Search

- General idea: Expand the node n that is *closest to the goal* (estimate)
- Implementation: Using a *priority* queue ordered by this *measure of closeness*

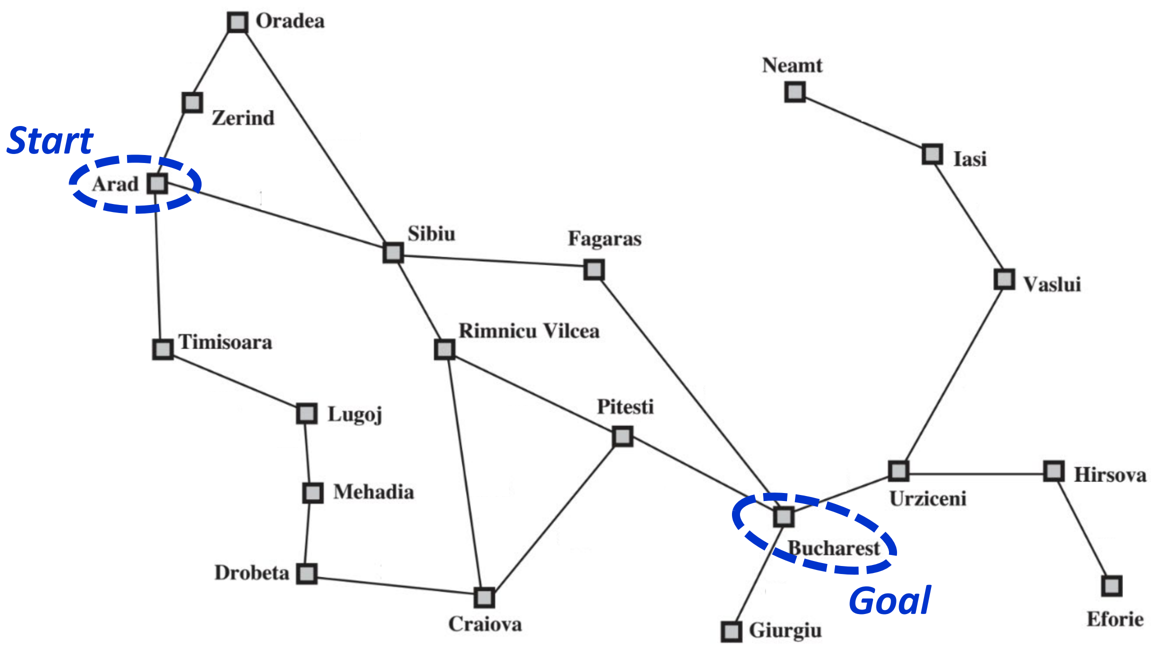



**Straight Line Distances to Bucharest**

| | |
|---|---|
| Arad | 366 |
| Bucharest | 0 |
| Craiova | 160 |
| Drobeta | 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 |

# Greedy Best-First Search

- General idea: Expand the node *n* with the *lowest heuristic h(n)*
- Implementation: Using a *priority* queue ordered by *heuristic h(n)*
  - Evaluation function *f(n) = h(n)*




***h(n) = Straight Line Distances to Bucharest***

| | |
|---|---|
| **Arad** | 366 |
| **Bucharest** | 0 |
| **Craiova** | 160 |
| **Drobeta** | 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 |

# Greedy Best-First Search

- General idea: Expand the node *n* with the *lowest heuristic h(n)*
- Implementation: Using a *priority* queue ordered by *heuristic h(n)*
  - Evaluation function *f(n) = h(n)*

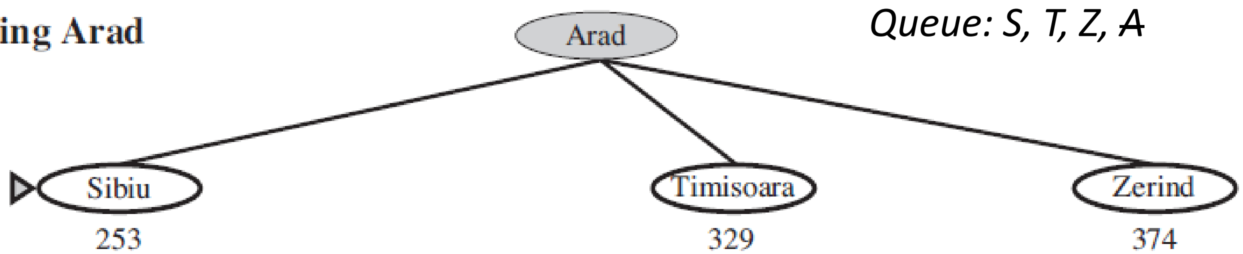
**(a) The initial state**


*Queue: A*

# Greedy Best-First Search

- General idea: Expand the node *n* with the *lowest heuristic h(n)*
- Implementation: Using a *priority* queue ordered by *heuristic h(n)*
  - Evaluation function *f(n) = h(n)*

**(b) After expanding Arad**




*Queue: S, T, Z, ~~A~~*

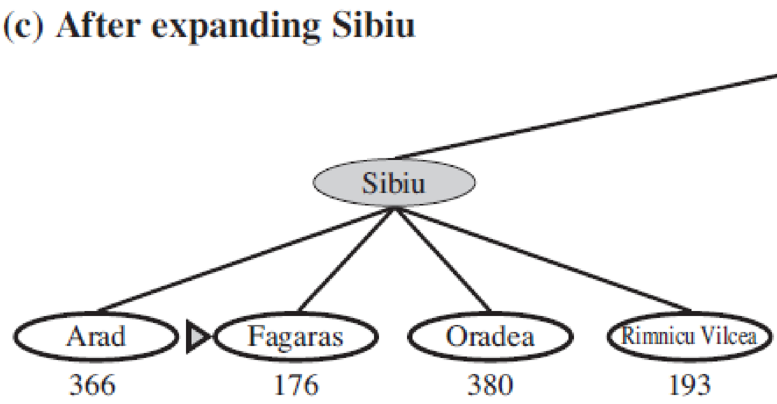
*Queue@D1: AS (253), AT (329), AZ (374)*

# Greedy Best-First Search

- General idea: Expand the node *n* with the *lowest heuristic h(n)*
- Implementation: Using a *priority* queue ordered by *heuristic h(n)*
  - Evaluation function *f(n) = h(n)*

**(c) After expanding Sibiu**

*Queue: F, R, ~~S,~~ T, A, Z, O, ~~A~~*

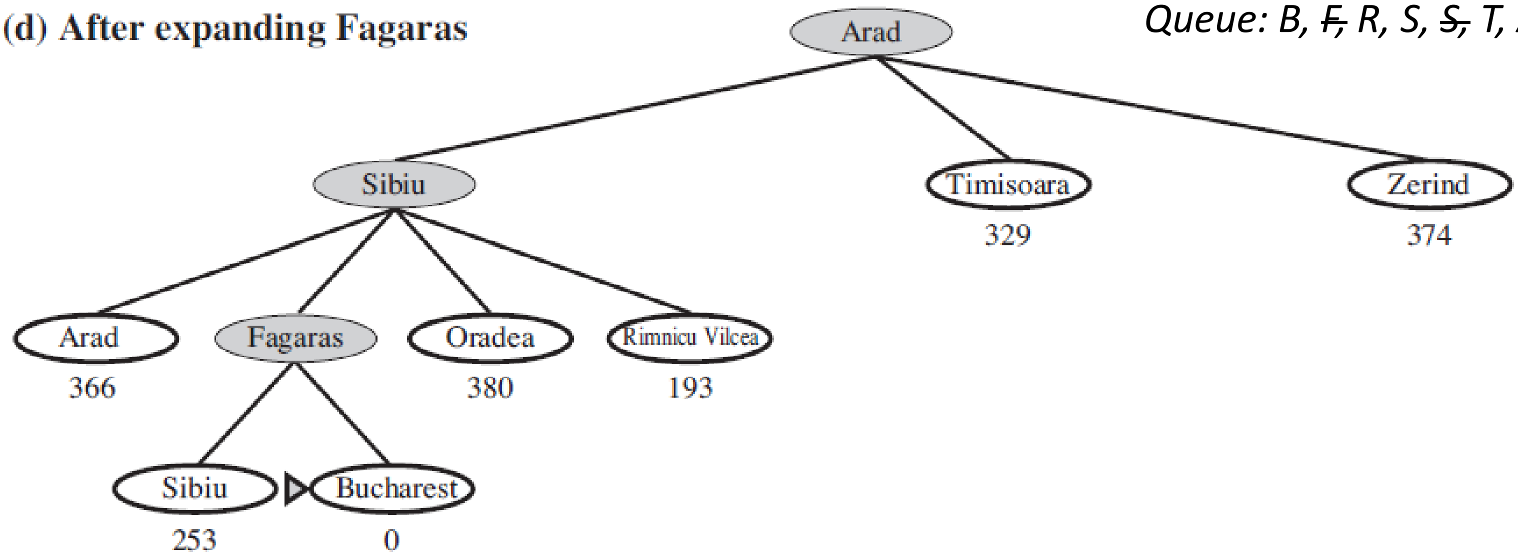


# Greedy Best-First Search

- General idea: Expand the node *n* with the *lowest heuristic h(n)*
- Implementation: Using a *priority* queue ordered by *heuristic h(n)*
  - Evaluation function *f(n) = h(n)*

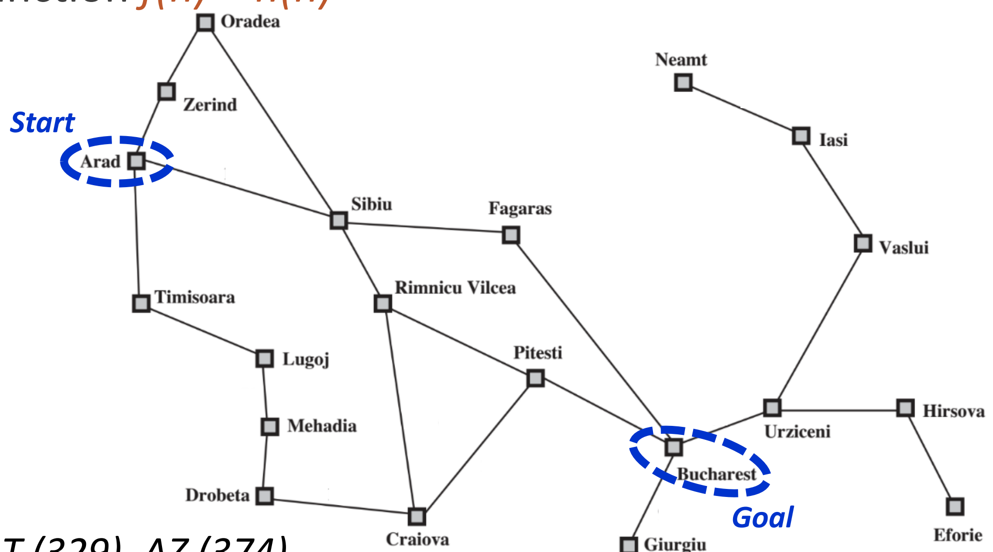
**(d) After expanding Fagaras**

*Queue: B, ~~F~~, R, S, ~~S~~, T, A, Z, O, ~~A~~*



# Greedy Best-First Search

- General idea: Expand the node *n* with the *lowest heuristic h(n)*
- Implementation: Using a *priority* queue ordered by *heuristic h(n)*
  - Evaluation function *f(n) = h(n)*




***h(n) = Straight Line Distances to Bucharest***

| | |
|---|---|
| Arad | 366 |
| Bucharest | 0 |
| Craiova | 160 |
| Drobeta | 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 |

**Full Solution**

*Queue@D0: A (366)*
*Queue@D1: AS (253), AT (329), AZ (374)*
*Queue@D2: ASF (176), ASR (193), AT (329), ASA (366), AZ (374), ASO (380)*
*Queue@D3: ASFB (0), ASR (193), ASFS (253), AT (329), ASA (366), AZ (374), ASO (380)*

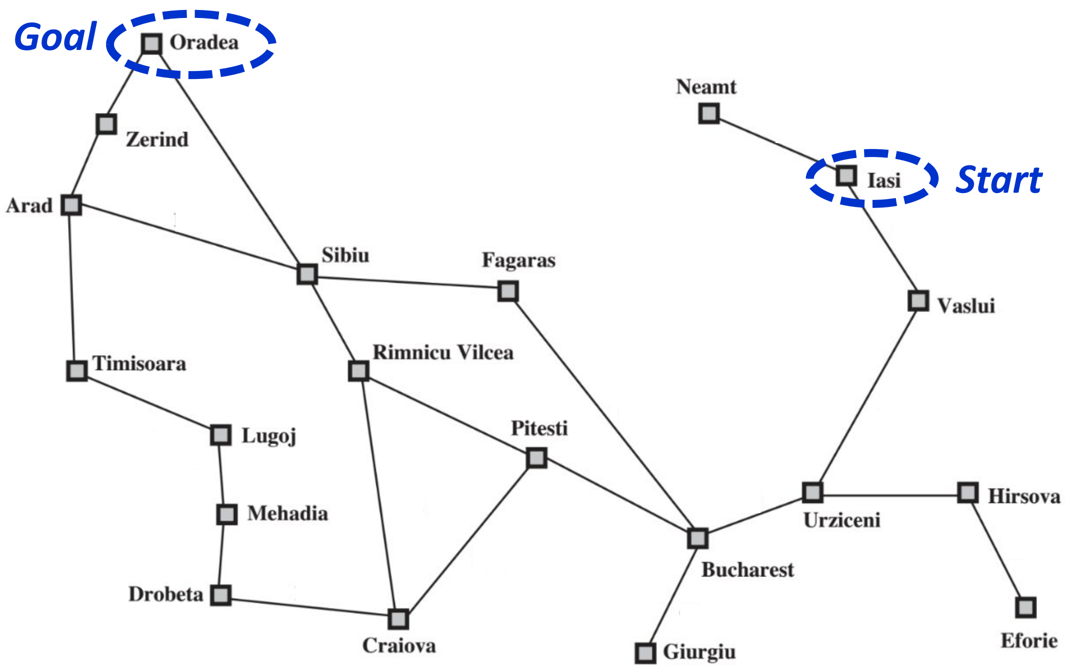
# Properties of Greedy Best-First Search

- Completeness: ?

# Properties of Greedy Best-First Search

- Completeness: No

  (Can get stuck in loops, unless?)



# Properties of Greedy Best-First Search

- Completeness: No

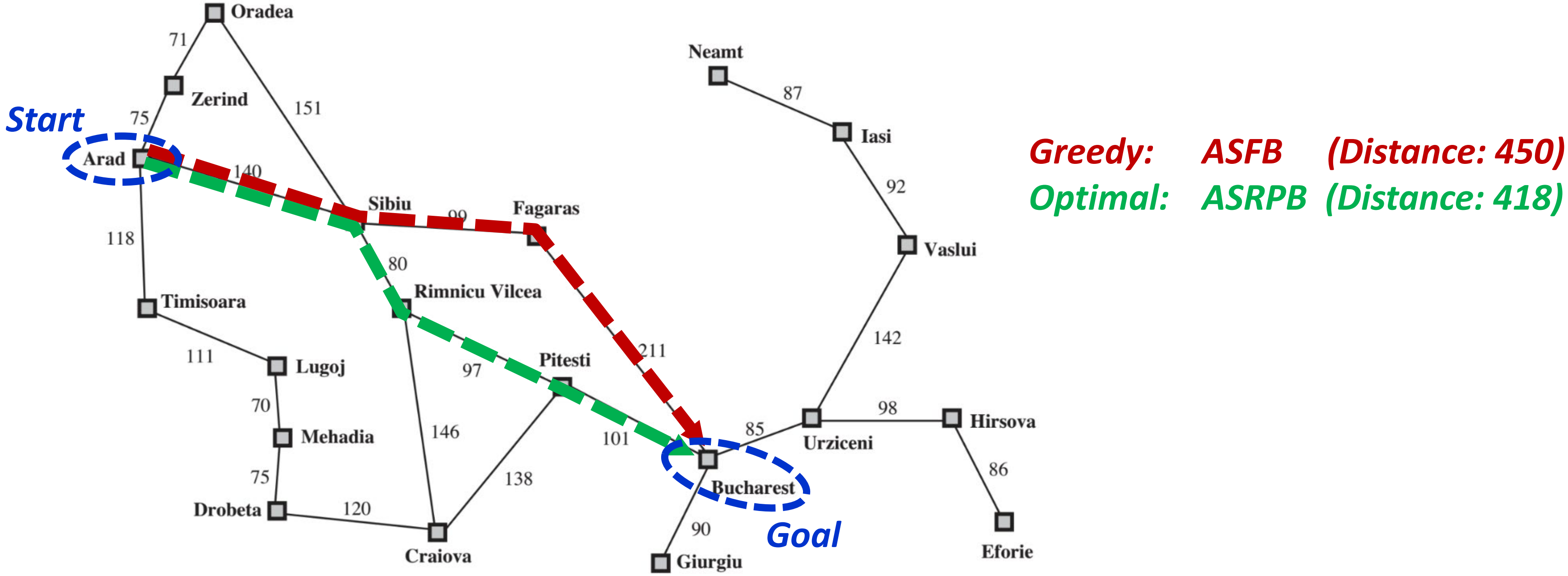
  (Can get stuck in loops, unless?)
- Optimality: ?

# Properties of Greedy Best-First Search

- Completeness: No

  (Can get stuck in loops, unless?)

- Optimality: No



# Properties of Greedy Best-First Search

- Completeness: No

  (Can get stuck in loops, unless?)
- Optimality: No
- Time complexity: $O(b^m)$

  (Like DFS but a good heuristic can improve performance greatly)
- Space complexity: $O(b^m)$

  (Keeps all nodes in memory)

# Exercise: UCS vs Greedy Best-First Search

- What are the issues with UCS, in contrast to greedy search?

- What are the issues with greedy best-first search, in contrast to UCS?

- Is there a way to combine the two and address each's shortcomings?

# Exercise: UCS vs Greedy Best-First Search

- What are the issues with UCS, in contrast to greedy search?
  - Complete and optimal but may “waste” search in the wrong direction
- What are the issues with greedy best-first search, in contrast to UCS?
- Is there a way to combine the two and address each’s shortcomings?

# Exercise: UCS vs Greedy Best-First Search

- What are the issues with UCS, in contrast to greedy search?
  - Complete and optimal but may “waste” search in the wrong direction
- What are the issues with greedy best-first search, in contrast to UCS?
  - Search generally in the “right” direction but not complete or optimal
- Is there a way to combine the two and address each’s shortcomings?

# Exercise: UCS vs Greedy Best-First Search

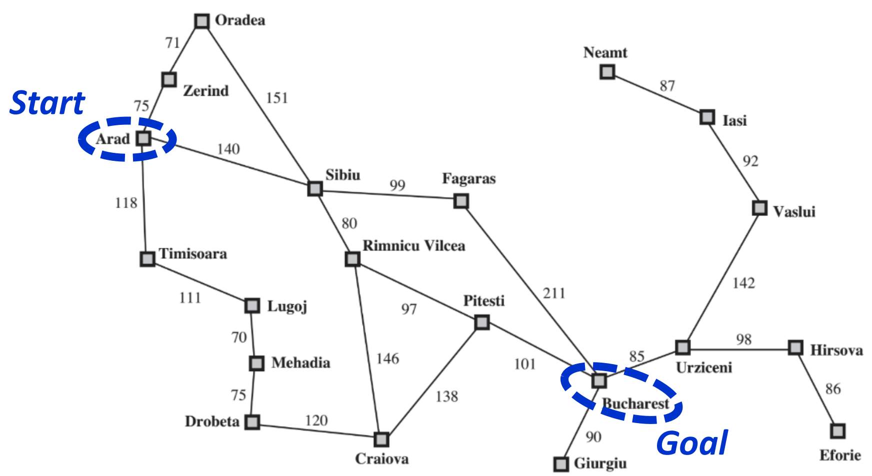
- What are the issues with UCS, in contrast to greedy search?
  - Complete and optimal but may “waste” search in the wrong direction

- What are the issues with greedy best-first search, in contrast to UCS?
  - Search generally in the “right” direction but not complete or optimal

- Is there a way to combine the two and address each’s shortcomings?
  - UCS: $f(n) = g(n)$
  - Greedy: $f(n) = h(n)$
  - Combined: $f(n) = g(n) + h(n) \rightarrow$ ???

# A* Search

- General idea: Expand the node *n* that has *incurred the least cost* and is *nearest to the goal state*
- Implementation: Using a *priority* queue ordered by *eval. func. f(n)*
  - Evaluation function *f(n) = g(n) + h(n)*
  - Path cost g(n) = total path cost from start node to node *n*
  - Heuristic h(n) = estimated distance from node *n* to goal state




**Straight Line Distances to Bucharest**

| | |
|---|---|
| Arad | 366 |
| Bucharest | 0 |
| Craiova | 160 |
| Drobeta | 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 |

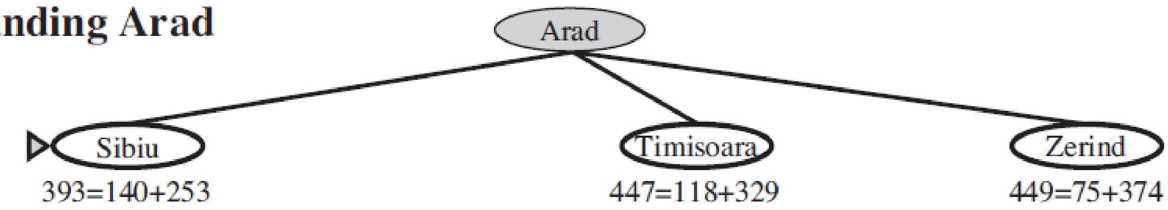
# A* Search

- General idea: Expand the node *n* that has *incurred the least cost* and is *nearest to the goal state*
- Implementation: Using a *priority* queue ordered by *eval. func. f(n)*
  - Evaluation function $f(n) = g(n) + h(n)$

# A* Search

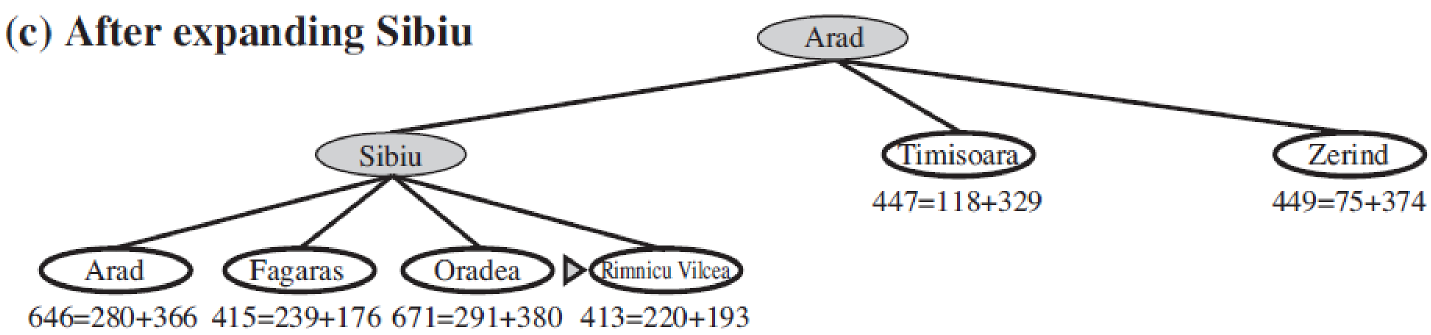
- General idea: Expand the node *n* that has *incurred the least cost* and is *nearest to the goal state*
- Implementation: Using a *priority* queue ordered by *eval. func. f(n)*
  - Evaluation function *f(n) = g(n) + h(n)*

**(b) After expanding Arad**



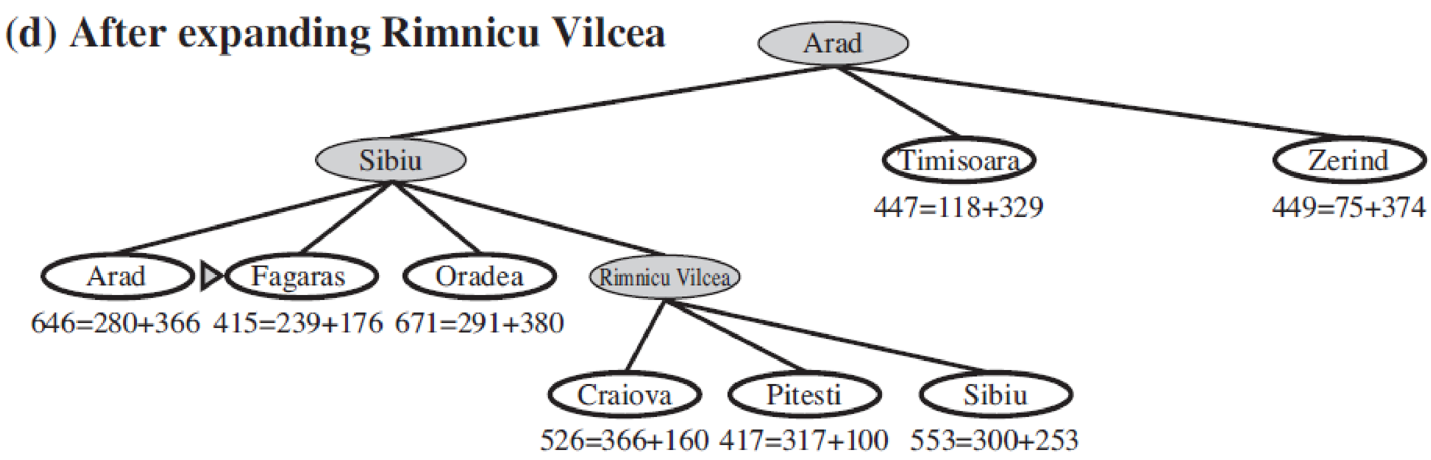
# A* Search

- General idea: Expand the node *n* that has *incurred the least cost* and is *nearest to the goal state*
- Implementation: Using a *priority* queue ordered by *eval. func. f(n)*
  - Evaluation function $f(n) = g(n) + h(n)$



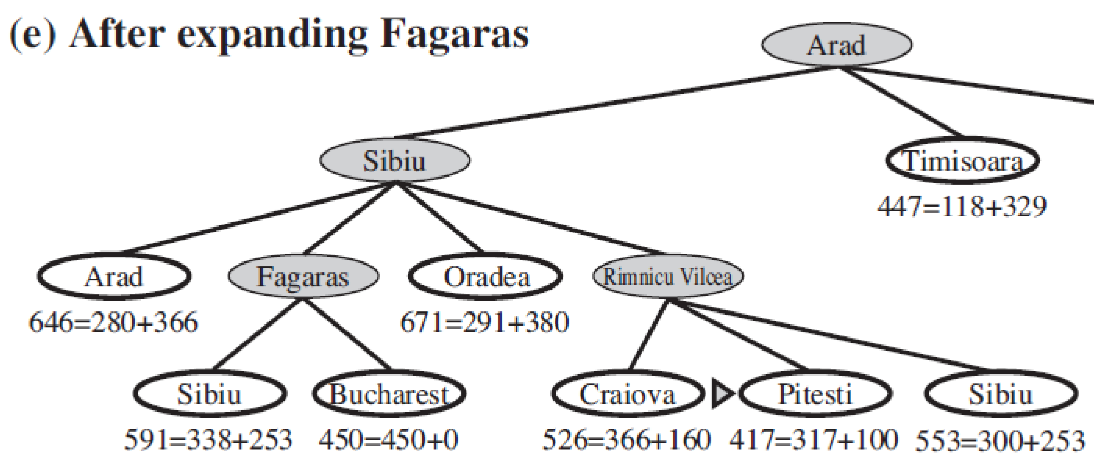
# A* Search

- General idea: Expand the node *n* that has *incurred the least cost* and is *nearest to the goal state*
- Implementation: Using a *priority* queue ordered by *eval. func. f(n)*
  - Evaluation function $f(n) = g(n) + h(n)$



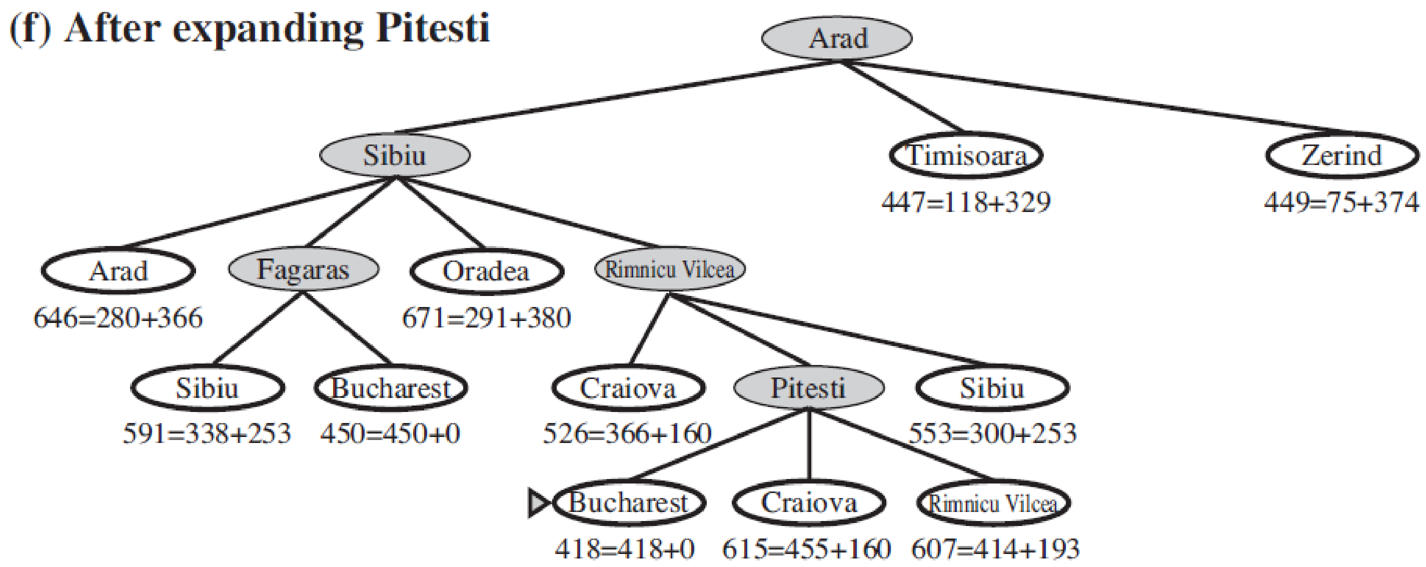
# A* Search

- General idea: Expand the node *n* that has *incurred the least cost* and is *nearest to the goal state*
- Implementation: Using a *priority* queue ordered by *eval. func. f(n)*
  - Evaluation function *f(n) = g(n) + h(n)*



# A* Search

- General idea: Expand the node *n* that has *incurred the least cost* and is *nearest to the goal state*
- Implementation: Using a *priority* queue ordered by *eval. func. f(n)*
  - Evaluation function *f(n) = g(n) + h(n)*



# Properties of A* Search

- Completeness: Yes
- Optimality: Yes

  (If heuristics are admissible/consistent, more on this later)
- Time complexity: Same as UCS
- Space complexity: Same as UCS

# Summary & Objectives

- Learn about two different informed search algorithms
  - Greedy Best-First Search
  - A* Search
- Understand the role that heuristic plays in these algorithms
- Being able to use Greedy Best-First Search and A* Search to solve a search problem