UNINFORMED SEARCH

Lara J. Martin (she/they)
TA: Aydin Ayanzadeh (he)

9/12/2023

CMSC 671

By the end of class today, you will be able to:

1. Choose when to use breadth-first, depth-first, or uniform-cost search for a given problem
2. Assess the limitation of uninformed search

Modified from slides by Dr. Cassandra Kent

HW 1 RELEASED

Pac-Man search

Due 9/26 at 11:59pm

https://laramartin.net/Principles-of-AI/homeworks/search/search.html

RECAP

What does it mean for an agent to have a “model”?

RECAP

Components of a Search Problem

- Actions
- States
- Initial state
- Transition model
- Goal test
- Step cost

RECAP

What's the difference between DFS and BFS?

EVALUATING SEARCH ALGORITHMS

WHY HAVE DIFFERENT WAYS TO TRAVERSE A TREE?

We can look at four criteria:

- **Completeness**
- **Optimality**
- **Time complexity**
- **Space complexity**

COMPLETENESS

Completeness: Will the algorithm always find a solution?

Example problem: Brute-force simple password guessing agent

- States: all combinations of letters a-z
- Initial state: empty string
- Actions: add a letter a-z
- Transition model: append letter to end of password
- Goal test: type in password and see if it works

BFS VS DFS: COMPLETENESS

Completeness: Will the algorithm always find a solution?

Example problem: Brute-force simple password guessing agent

Breadth-First Search

- States expanded: a, b, c, …, aa, ab, ac, …, ba, bb, bc, …, ca, cb, …
- Is complete (given infinite time)

Depth-First Search

- States expanded: a, aa, aaa, aaaa, aaaaa, aaaaaa, aaaaaaa, …
- Is **not** complete

DFS can not handle state spaces with infinite depth!

BFS VS DFS: OPTIMALITY

Optimality: Will the algorithm always find the best (shortest) path to the goal?

Breadth-First Search

- Fully expands all nodes at minimum depth before continuing
- Is optimal

Depth-First Search

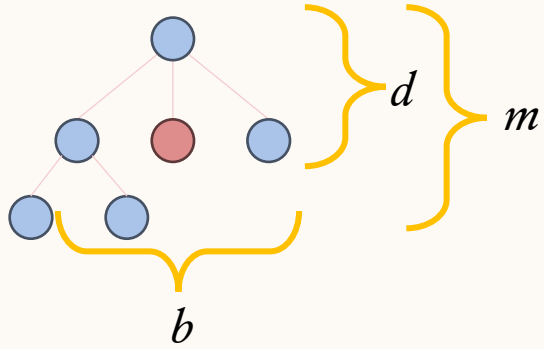
- Counter example: http://graphonline.ru/en/?graph=rOFlnolAZkfZXorA
- Is **not** optimal

BFS VS DFS: TIME COMPLEXITY

Time complexity: How long does it take to find a solution?

Depends on:

- Branching factor b: the number of actions the agent can take at any state
- Depth of solution d: the optimal solution length
- Maximum depth of tree m



Breadth-First Search

- $O(b^d)$
- Must search all branches until solution depth is reached

Depth-First Search

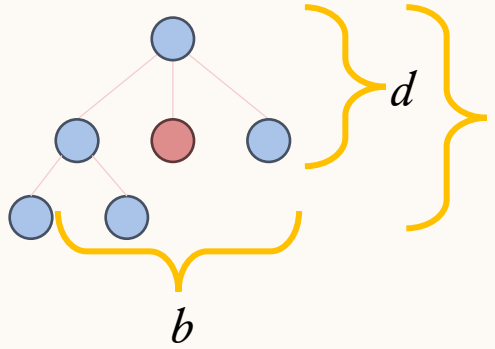
- $O(b^m)$
- Must search to the end of the tree

BFS VS DFS: SPACE COMPLEXITY

Space complexity: How much memory does it take to execute?

Depends on:

- Branching factor b: the number of actions the agent can take at any state
- Depth of solution d: the optimal solution length
- Maximum depth of tree m



Breadth-First Search

- $O(b^d)$
- Must hold full row of search tree in memory

Depth-First Search

- $O(bm)$
- Holds 1 set of successors at each row of search tree in memory

BFS VS DFS

Why would we ever use DFS?

Breadth-First Search

- Complete
- Optimal
- $O(b^d)$ time complexity
- $O(b^d)$ space complexity

Depth-First Search

- Not complete
- Not optimal
- $O(b^m)$ time complexity
- $O(bm)$ space complexity

BFS can be **physically impossible to run** for large search depths and branching factors!

DIFFERENT OPTIMALITY CRITERIA

Will using search algorithms for our agent functions produce rational behavior?

BACK TO VAMPIRE WORLD

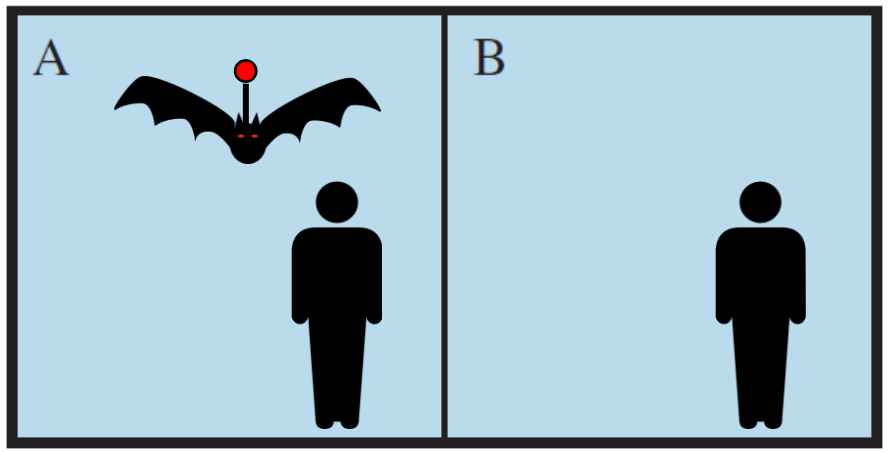
Agent: Vampire

Performance measure: Suck as much blood over time as possible

Environment: Location, humans

Actuators: Flying, sucking

Sensors: Short-range sonar (human detection)




```
if [A, Empty]: return Right
if [B, Empty]: return Left
if [A or B, Human]: return Suck
```

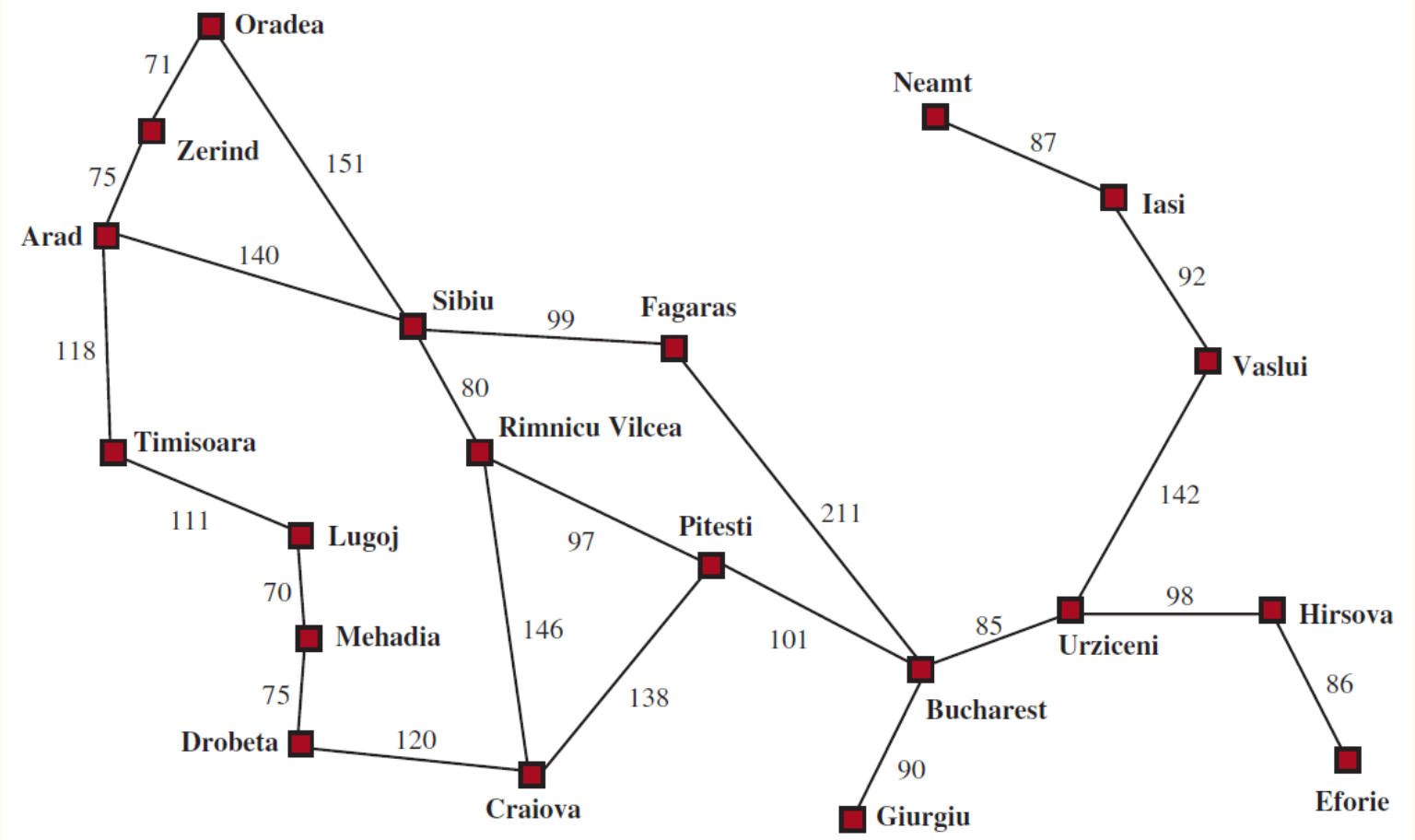
Is this a rational (vampire) agent?

DIFFERENT OPTIMALITY CRITERIA

Will using search algorithms for our agent functions produce rational behavior?

- BFS + DFS performance measure: number of actions taken
- What about other measures?

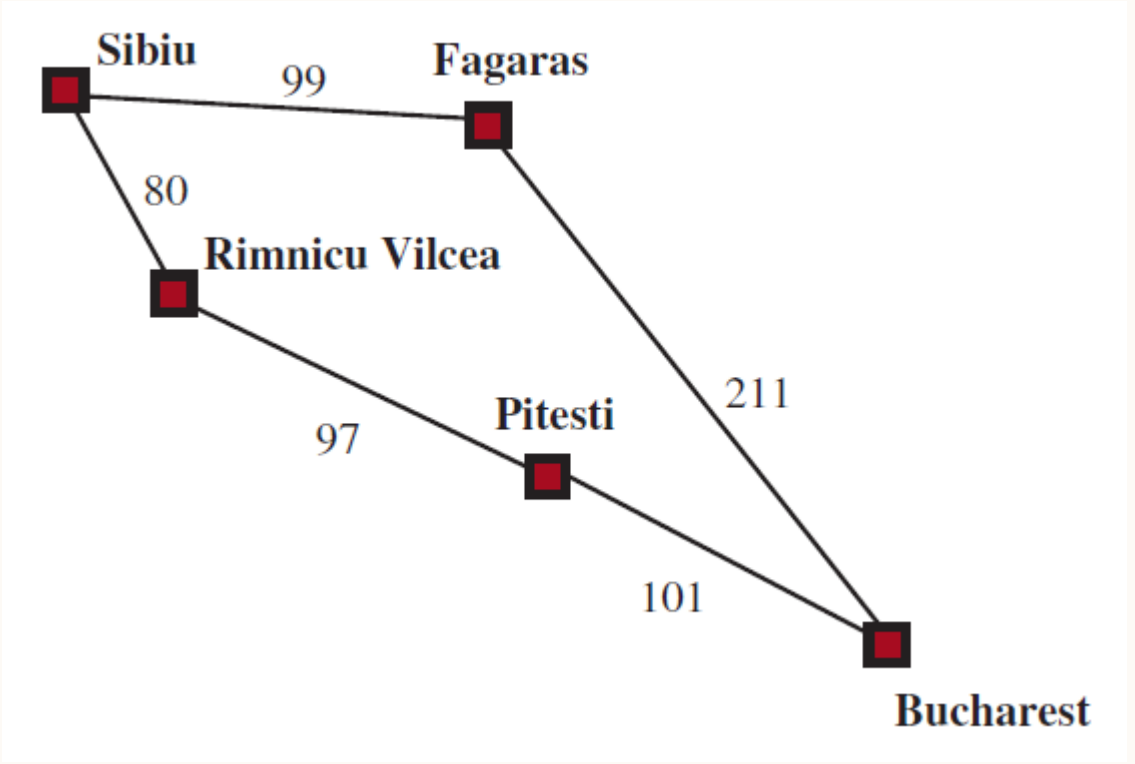
DIFFERENT OPTIMALITY CRITERIA



DIFFERENT OPTIMALITY CRITERIA

Problem: find the shortest (in terms of distance) path from Sibiu to Bucharest

- BFS is no longer optimal
- Need to account for step cost
- Use **Uniform-Cost Search**



UNIFORM-COST SEARCH (UCS)

```
function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  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
    expand the chosen node, adding the resulting nodes to the frontier
```

Store frontier as a priority queue, prioritized by path cost $g(n)$

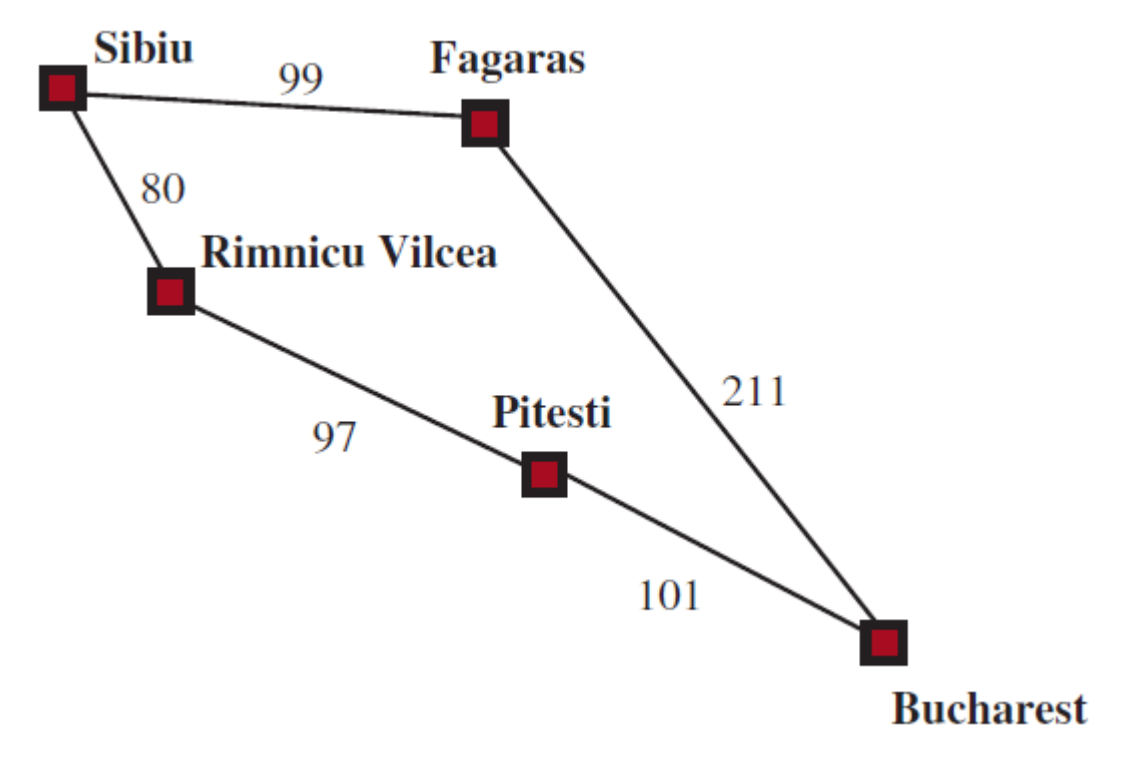
- Path cost calculated as accumulation of step costs from initial state to each node

UNIFORM-COST SEARCH (UCS)

Problem: find the shortest (in terms of distance) path from Sibiu to Bucharest

Frontier:
[Sibiu (0)]

Current node:
none



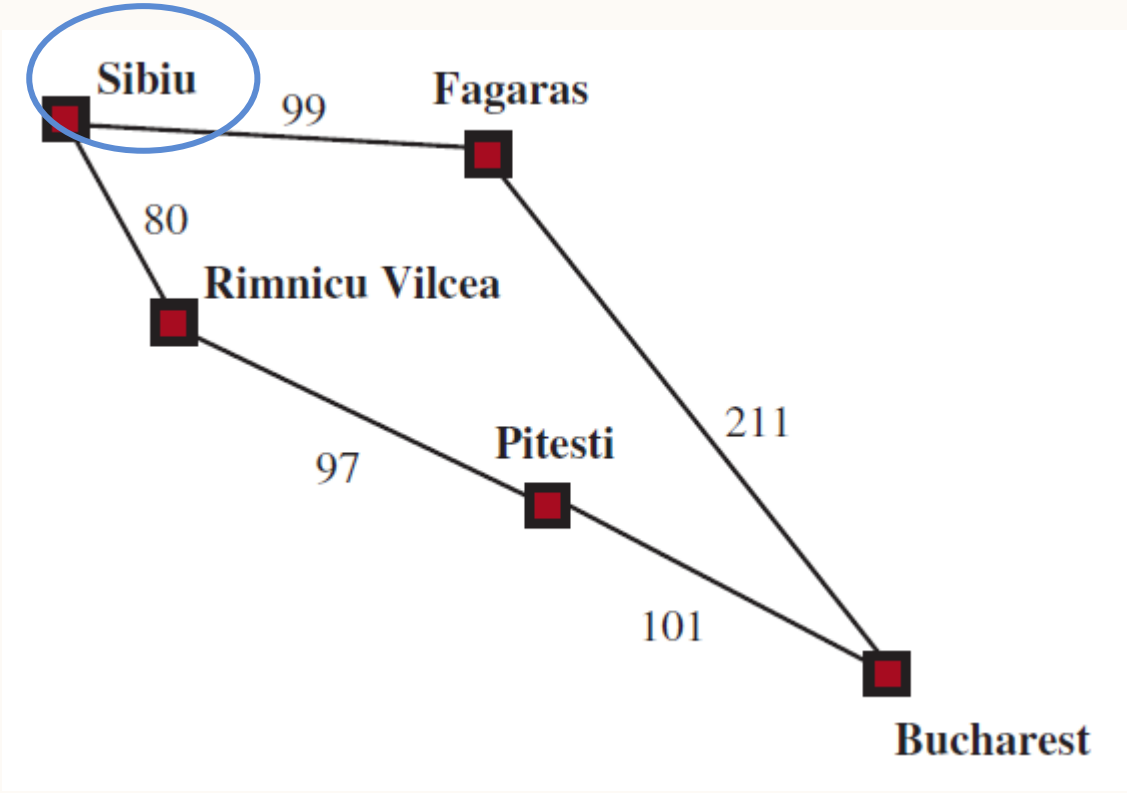
UNIFORM-COST SEARCH (UCS)

Problem: find the shortest (in terms of distance) path from Sibiu to Bucharest

Frontier:

[R.V. (80), F. (99)]

Current node:
Sibiu (0)



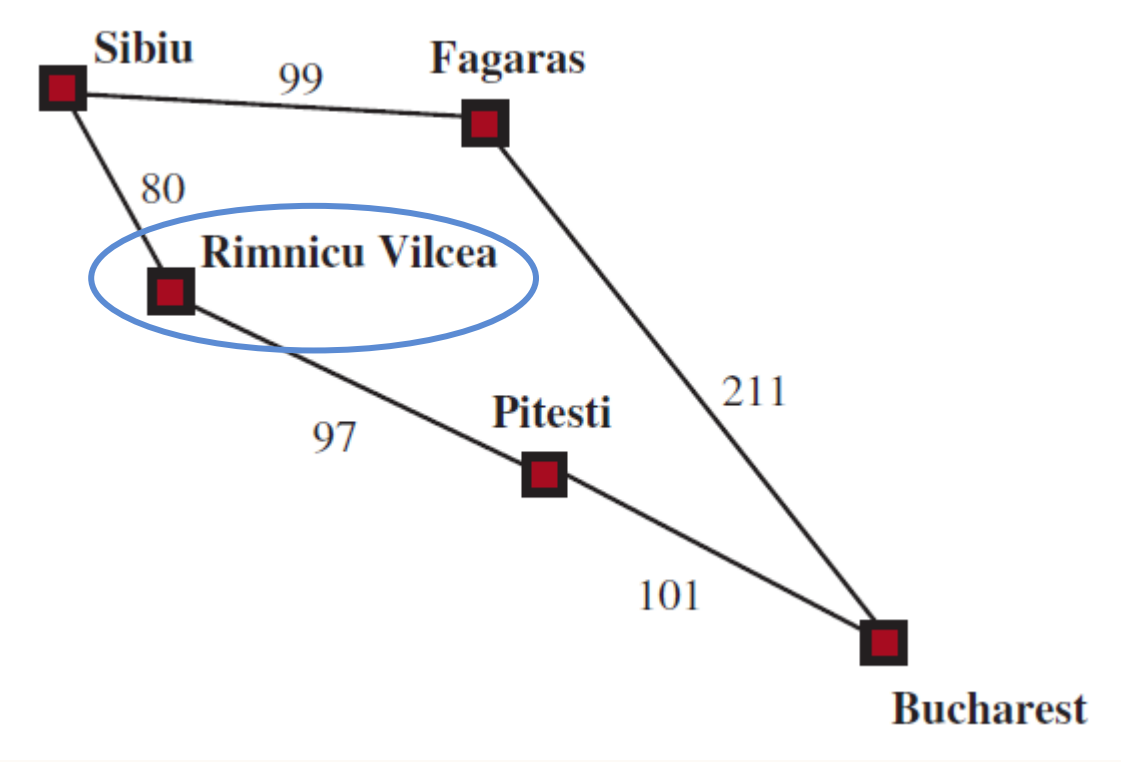
UNIFORM-COST SEARCH (UCS)

Problem: find the shortest (in terms of distance) path from Sibiu to Bucharest

Frontier:

[F. (99)]

Current node:
R.V. (80)



UNIFORM-COST SEARCH (UCS)

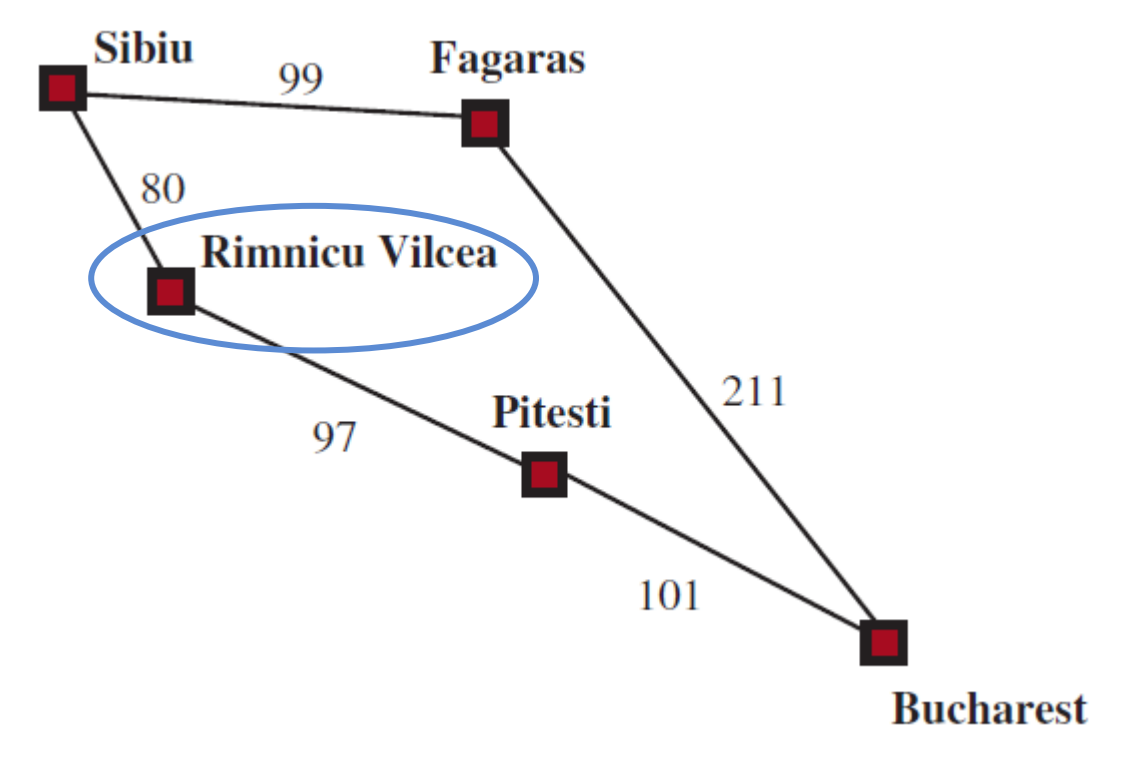
Problem: find the shortest (in terms of distance) path from Sibiu to Bucharest

Frontier:

[F. (99), P. (80+97=177)]

Current node:

R.V. (80)



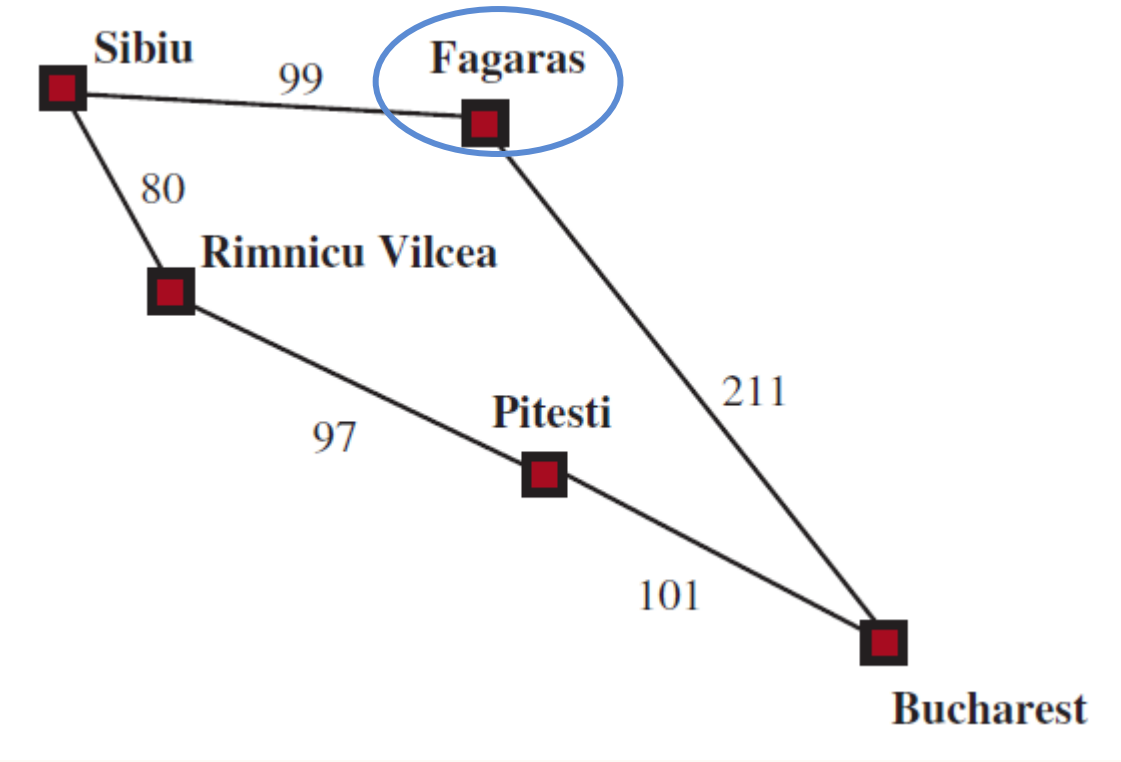
UNIFORM-COST SEARCH (UCS)

Problem: find the shortest (in terms of distance) path from Sibiu to Bucharest

Frontier:

[P. (177)]

Current node:
F. (99)



UNIFORM-COST SEARCH (UCS)

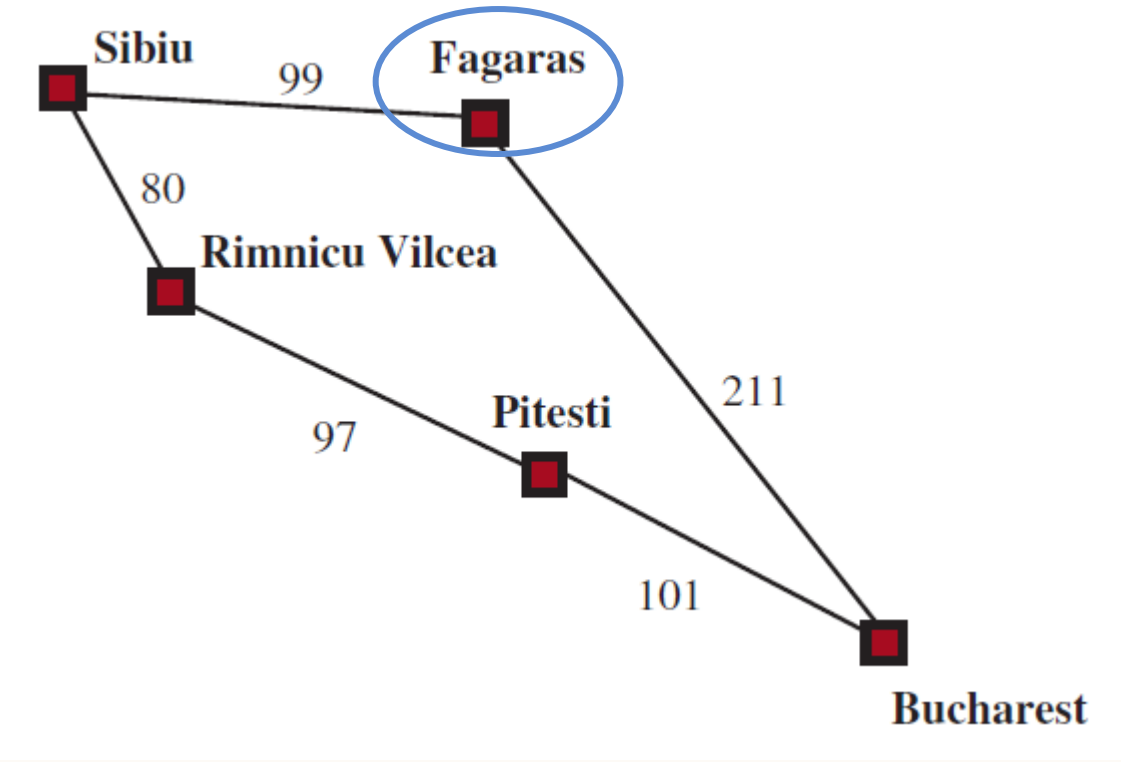
Problem: find the shortest (in terms of distance) path from Sibiu to Bucharest

Frontier:

[P. (177), B. (99+211=310)]

Current node:

F. (99)



UNIFORM-COST SEARCH (UCS)

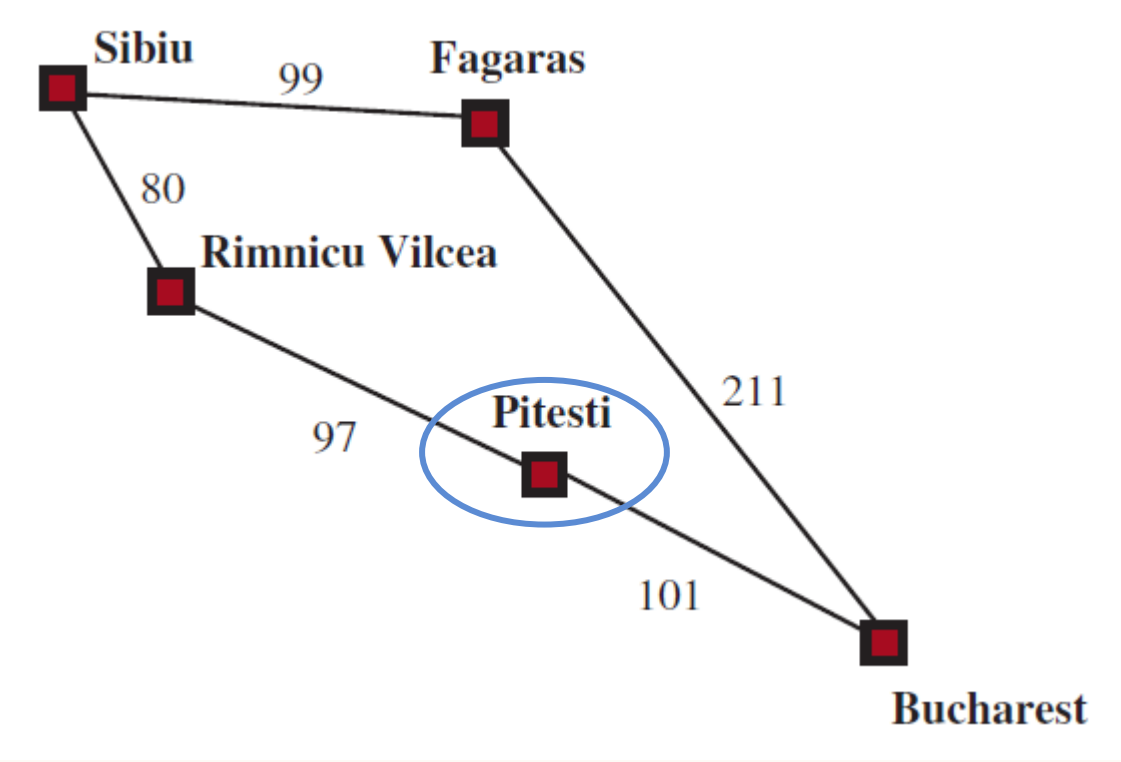
Problem: find the shortest (in terms of distance) path from Sibiu to Bucharest

Frontier:

[B. (310)]

Current node:

P. (177)



UNIFORM-COST SEARCH (UCS)

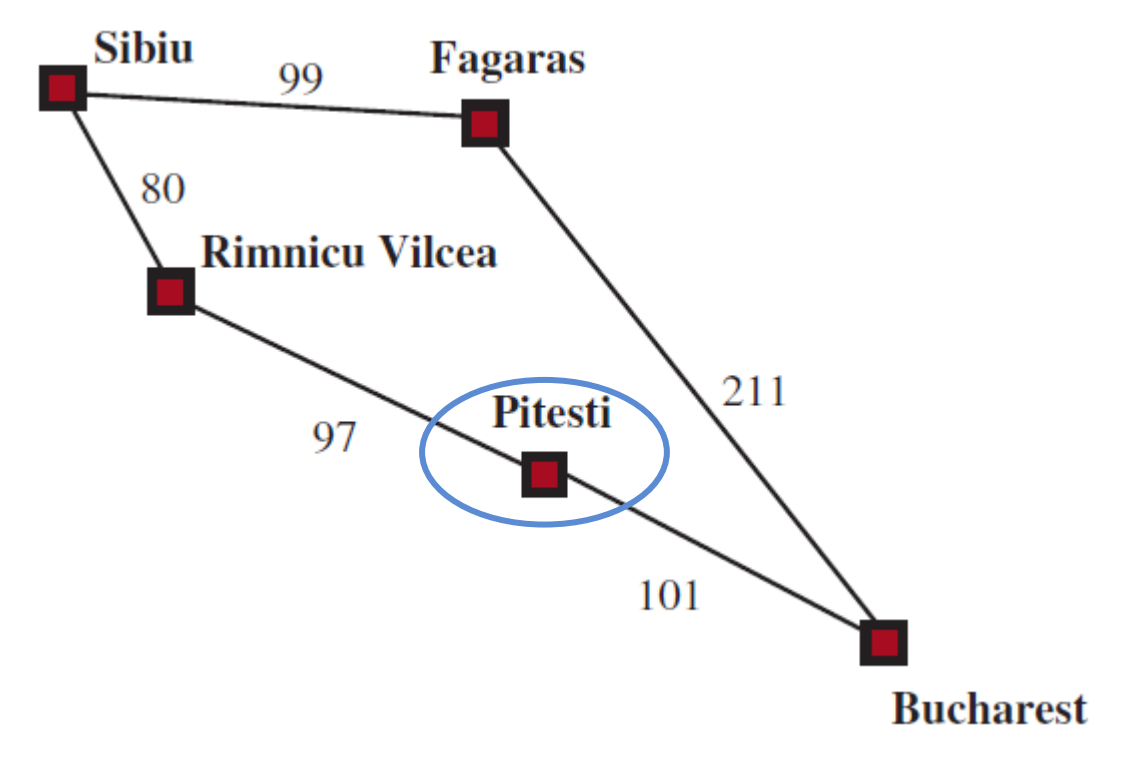
Problem: find the shortest (in terms of distance) path from Sibiu to Bucharest

Frontier:

[B. (177+101=278), B. (310)]

Current node:

P. (177)



UNIFORM-COST SEARCH (UCS)

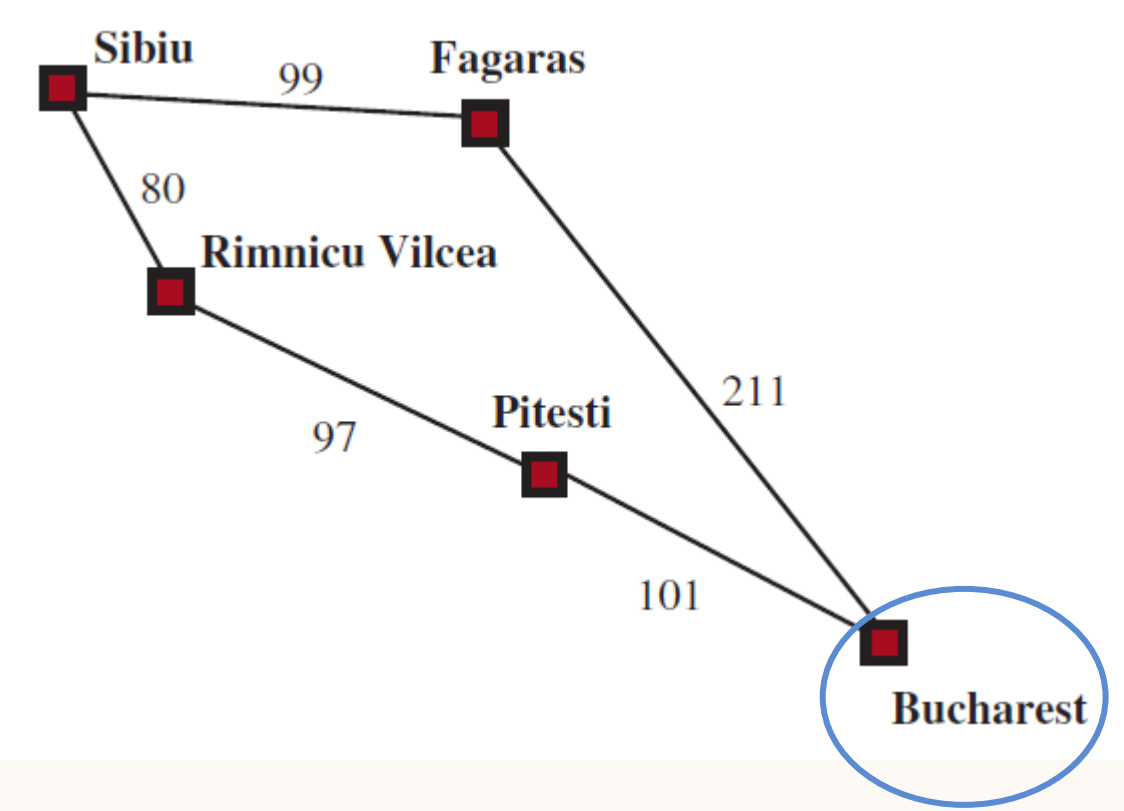
Problem: find the shortest (in terms of distance) path from Sibiu to Bucharest

Frontier:

[]

Current node:
B. (278)

Return:
[Sibiu, Rimnicu Vlicea, Pitesti, Bucharest]



PROPERTIES OF UCS (VS BFS)

Uniform-Cost Search

- Complete
- Optimal
- $O(b^{1+C^*/\epsilon})$ time complexity
- $O(b^{1+C^*/\epsilon})$ space complexity

Breadth-First Search

- Complete
- Optimal
- $O(b^d)$ time complexity
- $O(b^d)$ space complexity

where C^* is the optimal path cost to the goal, and ϵ is the minimum step cost (must be positive)

IN GROUPS: WHEN WOULD YOU USE THEM?

Depth-First Search	Breadth-First Search	Unified-Cost Search
• Mazes • Simple robot path (limited memory) • Finding wreck in ocean (branching factor too high) • Social network (finding how a friend is connected)	• Find closest destination • Finding people within an organization (e.g., LinkedIn) • Web crawler	• Shortest path (distance)/map directions • Internet packet sending (fastest network) • Finding cheapest flight w/ layovers

BACK TO BFS VS DFS

Breadth-First Search

- Complete
- Optimal
- $O(b^d)$ time complexity
- $O(b^d)$ space complexity

Depth-First Search

- Not complete
- Not optimal
- $O(b^m)$ time complexity
- $O(bm)$ space complexity

We have a complete, optimal algorithm that may be impossible to run, and a bad algorithm that's very memory efficient…

Can we get the benefits of both?

https://celebrity.nine.com.au/latest/old-el-paso-ad-girl-why-not-both-porque-no-los-dos-then-and-now/aba850f7-f848-4c43-a1b1-9233d499da3d

DFS AND THE PROBLEM OF INFINITE DEPTH

Problem: Brute-force simple password guessing agent

- States: all combinations of letters a-z
- Initial state: empty string
- Actions: add a letter a-z
- Transition model: append letter to end of password
- Goal test: type in password and see if it works

Depth-First Search

- States expanded: a, aa, aaa, aaaa, aaaaa, aaaaaa, aaaaaaa, …
- At a certain depth, it doesn't make sense to expand states
  - (probably no one will have a password with 1,000 characters? 10,000? ∞?)

DEPTH-LIMITED SEARCH

- Set a reasonable depth limit for the search problem
- Run depth-first search as normal, but do **not** add successor nodes to the frontier if the current node is at the depth limit

DEPTH-LIMITED SEARCH

Problem: Brute-force simple password guessing agent

Depth-First Search

- States expanded: a, aa, aaa, aaaa, aaaaa, aaaaaa, aaaaaaa, …

Depth-Limited Search

- For a depth limit of 4:
- States expanded: a, aa, aaa, aaaa, aaab, aab, aaba, aabb, … b, ba, baa, baaa, …
  - Eventually will generate all strings of length <= 4!

PROPERTIES OF DEPTH-LIMITED SEARCH

- Is depth-limited search complete?
- Is depth-limited search optimal?

Take a couple minutes and try to figure out under what conditions depth-limited search is complete or optimal.

Some terms that may help:

b: branching factor d: depth of the goal

l: depth limit m: maximum depth of tree

FOR NEXT CLASS

- Start reading Chapter 3.5-3.7
- If you have Module 1/Search for your paper presentation, start looking for a paper and send it to me & Aydin
  - Presentation on Thursday Sept 21