Izmir Institute of Technology

CENG 461 – Artificial Intelligence

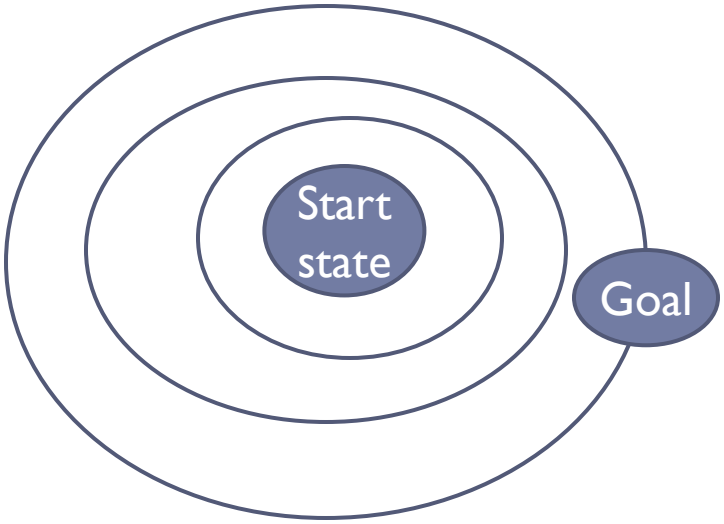
Informed Search and Heuristics

Search Algorithms and Blind Search

- In the previous lecture, we needed a strategy for picking nodes from the **frontier** at each iteration.
- We examined a group of strategies for blind search, i.e. when we do not have extra information about the problem.
- The only criteria we considered was the path cost up to the nodes in the frontier.

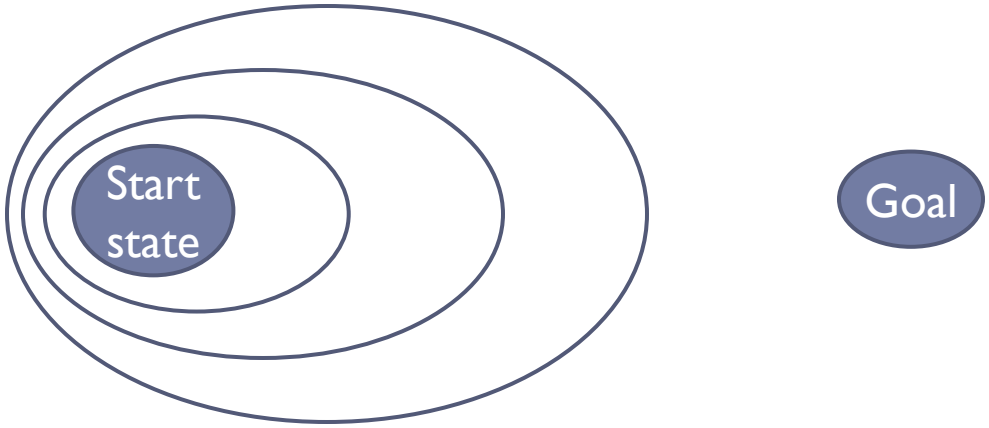
Search Algorithms and Blind Search

- Take the uniform-cost search, which is the only strategy that is optimal with varying step costs.
- In uniform-cost search, we expand in the space as the path cost increases (resembling topological contours).
- The search is not directed, which is not efficient when the space is large.



Informed Search

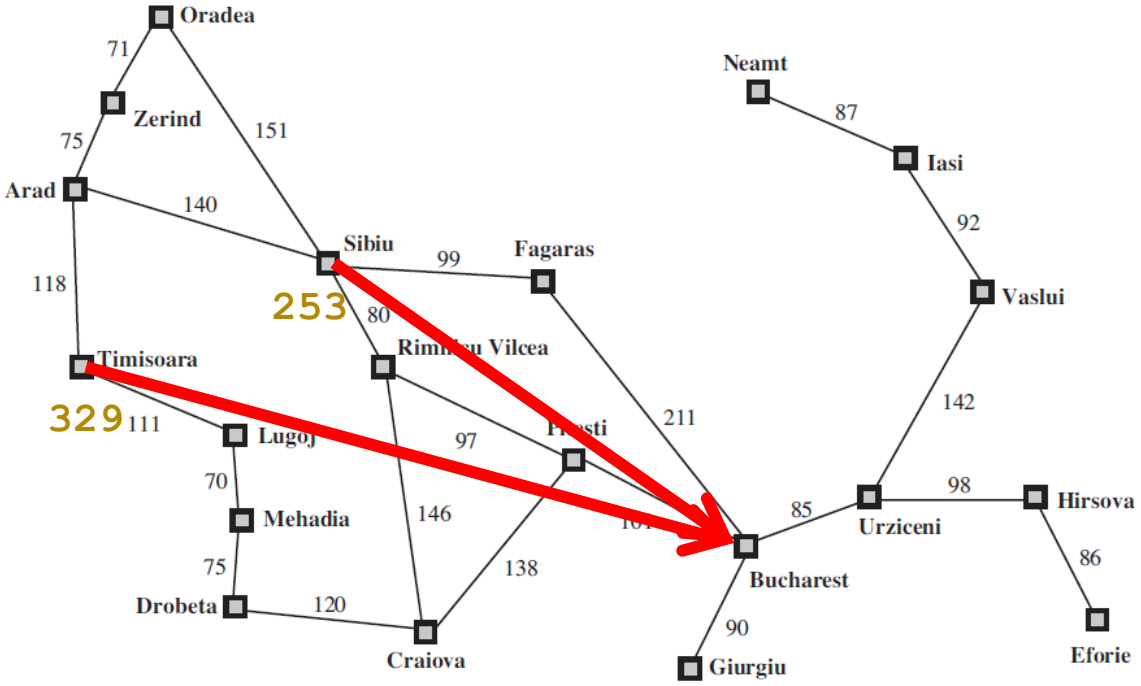
- We can do better if we have some intuition about the quality of the steps we take, called an informed search.



- The general strategy is called the best-first search.
- We can pick the graph node that minimizes an evaluation function f(n), which might be different than the path cost.
- Usually, the information is stored in a distinct part of f(n), called a heuristic function h(n).

Heuristics

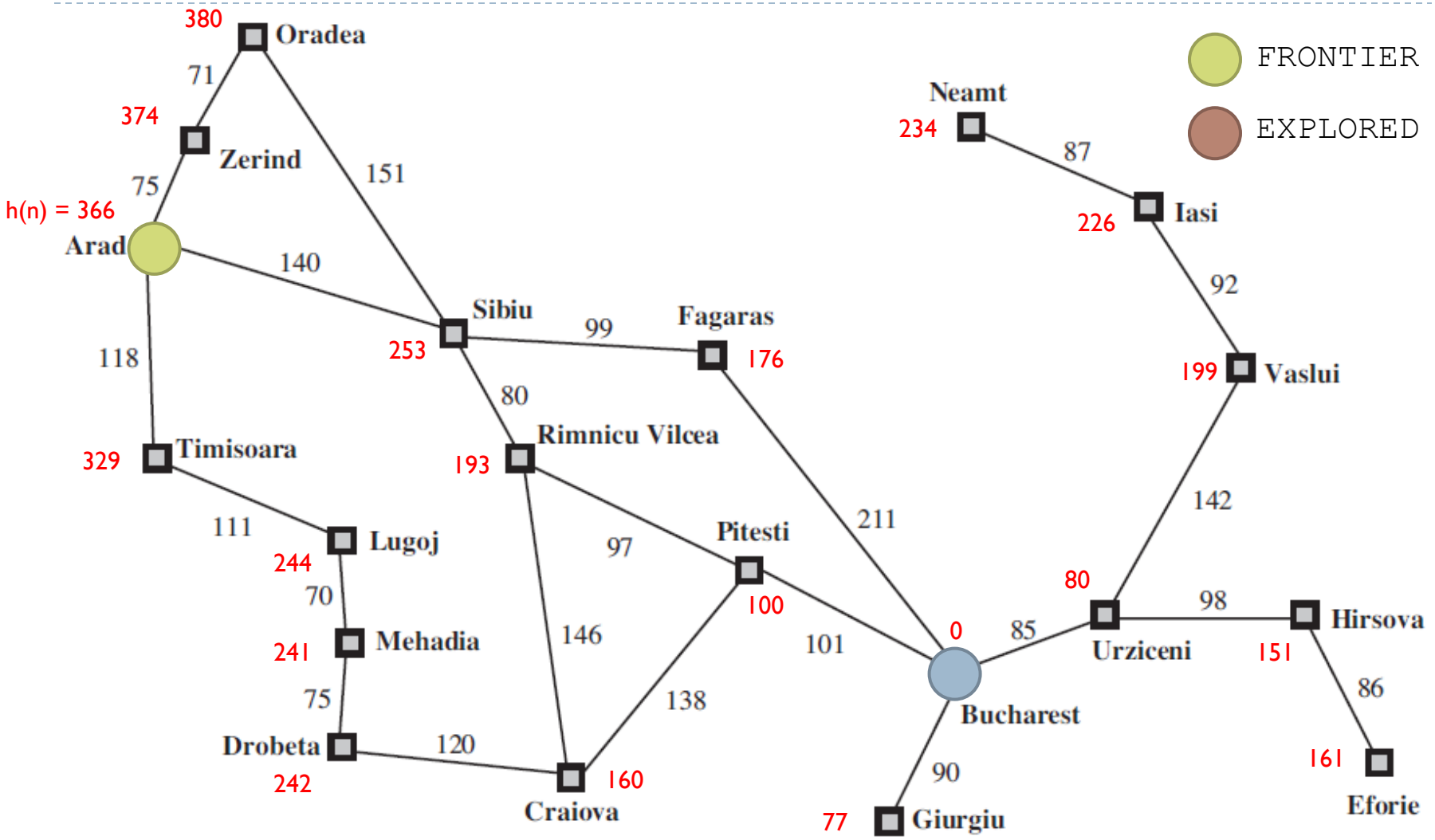
- For now, h(n) will be arbitrary and non-negative.
- h(n) will be zero at the goal states.
- For the route finding problem, a possible heuristic can be the Euclidean distance to the nearest goal.



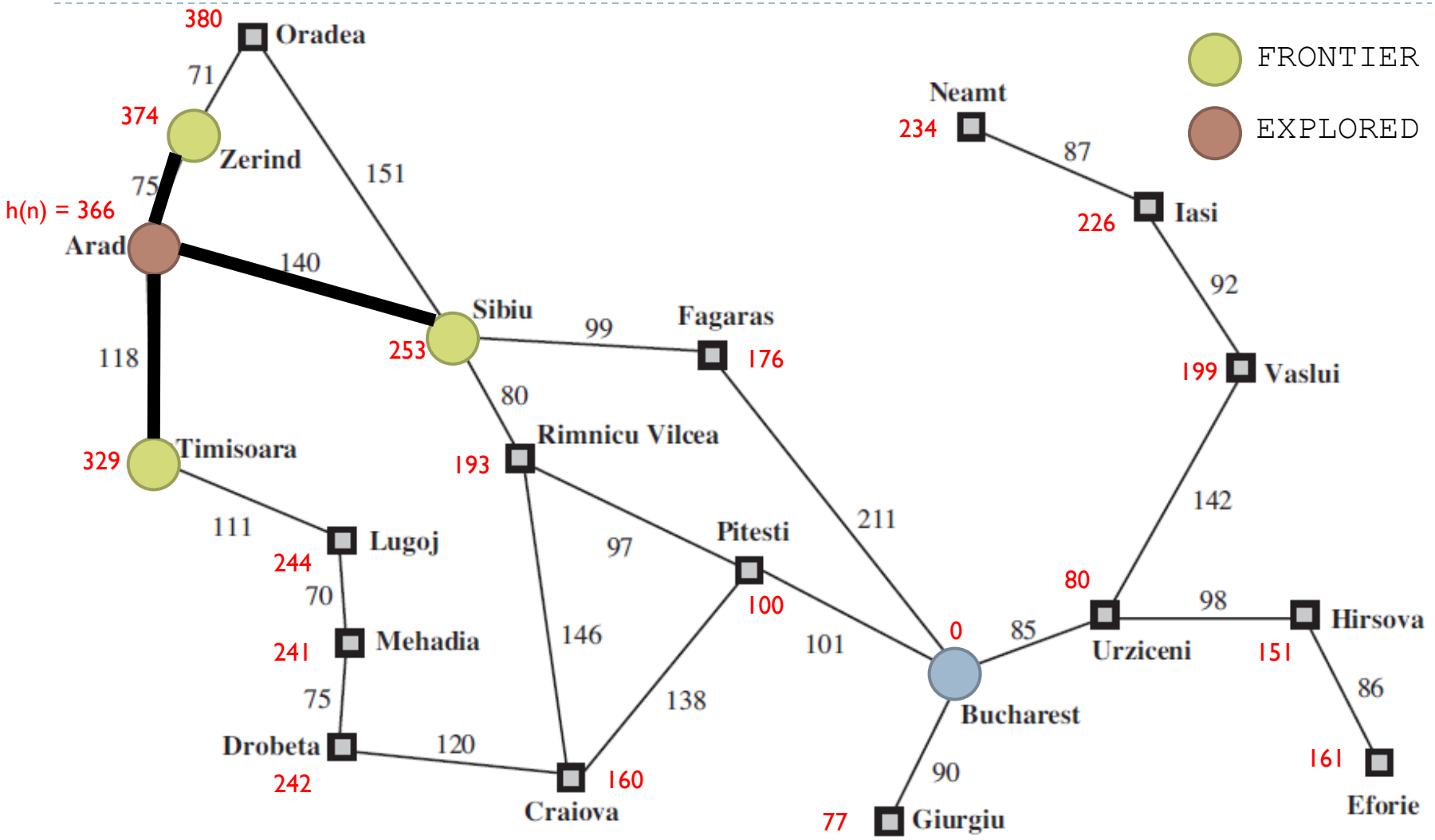
Greedy Best-First Search

- When we consider only the heuristic function for node selection without taking into account the path cost, we perform a greedy best-first search.
- For the route finding problem and the straight line (Euclidean) distance, this means we explore nodes that are closest to the goal first.

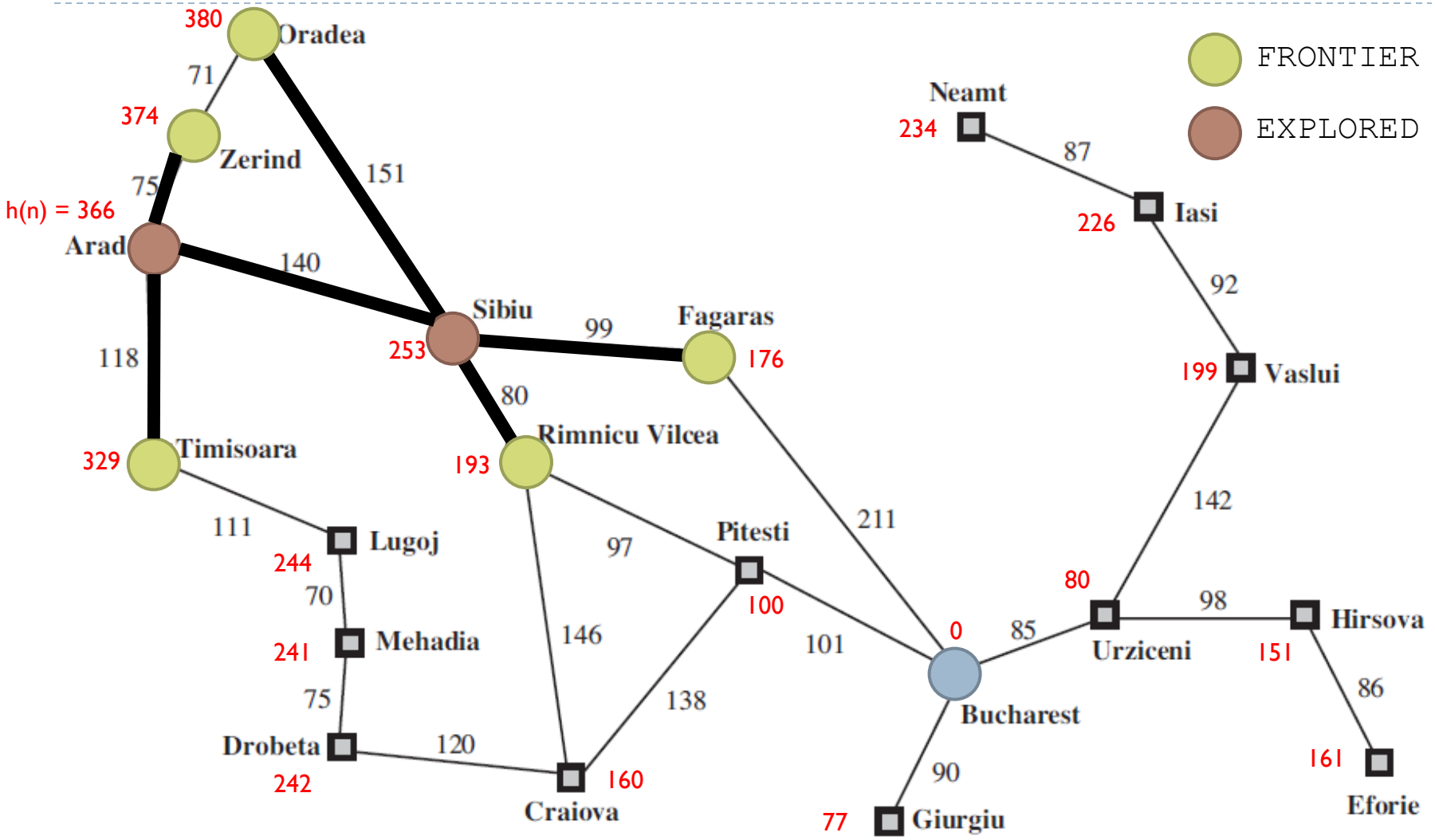
Greedy Best-First Search for Route Finding



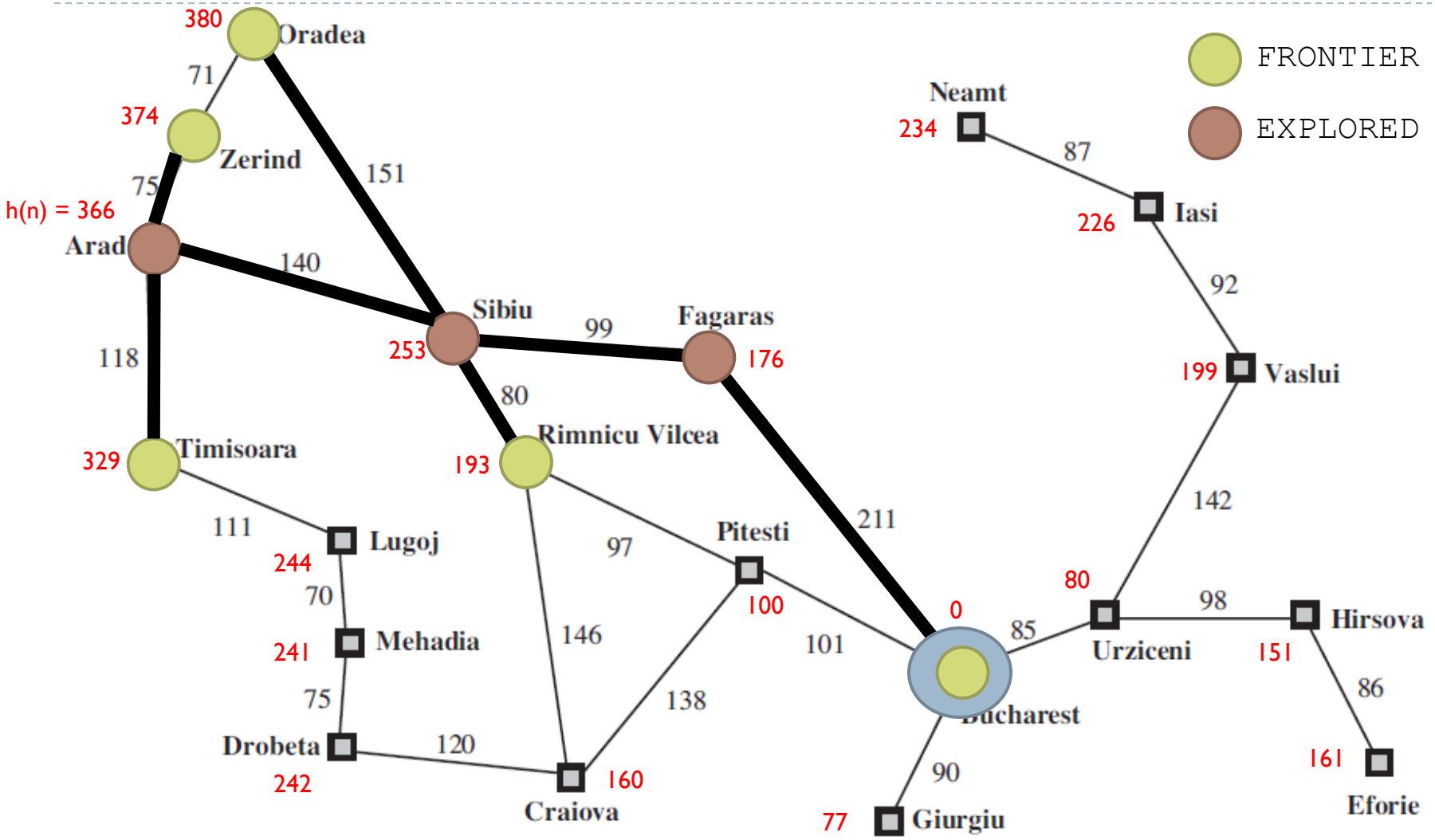
Greedy Best-First Search for Route Finding



Greedy Best-First Search for Route Finding



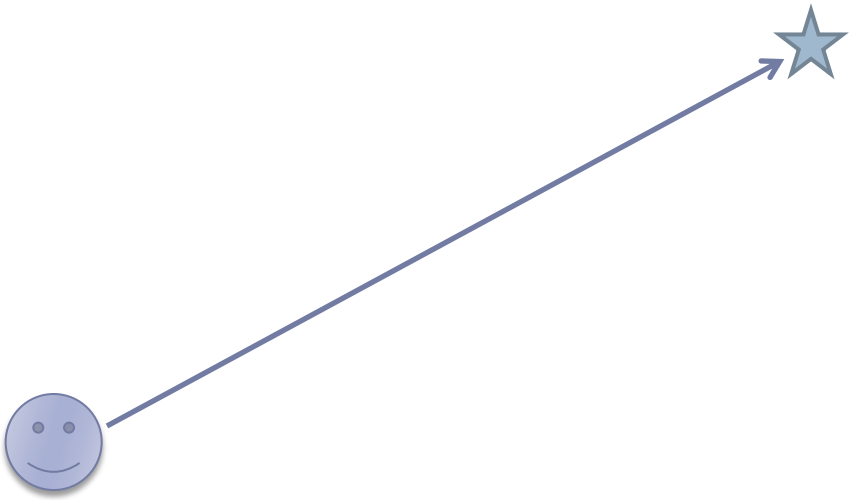
Greedy Best-First Search for Route Finding



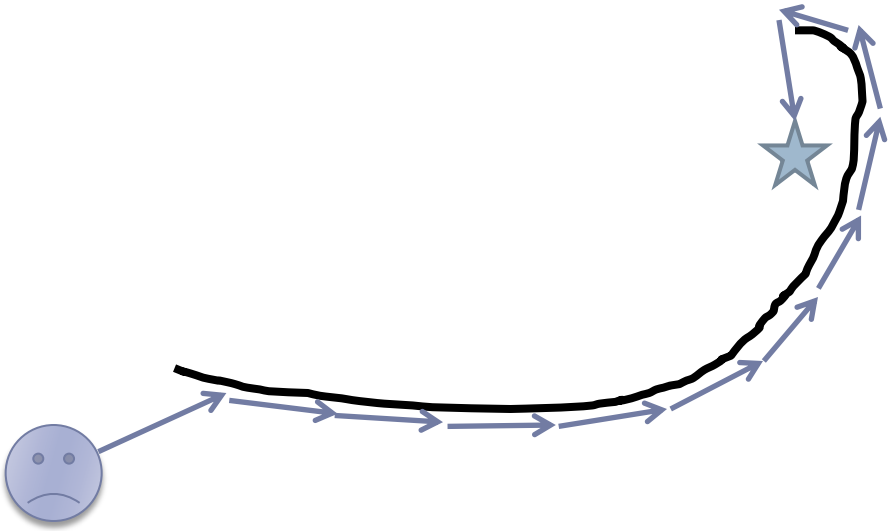
Greedy Best-First Search

- We have found a solution without expanding nodes on the other parts of the graph.
- This resembles the depth-first search in many ways.
  - It follows a single path to the goal, it is not optimal.
  - It is incomplete for infinite state spaces.
  - Worst-case time complexity is $O(b^m)$, where m is the maximum depth of the search space.
- In practice, if the heuristic function is good, we can reach the solution much faster.

Greedy Behavior



Greedy Behavior



We would like an algorithm that combines the best parts of greedy best-first search and uniform-cost search.

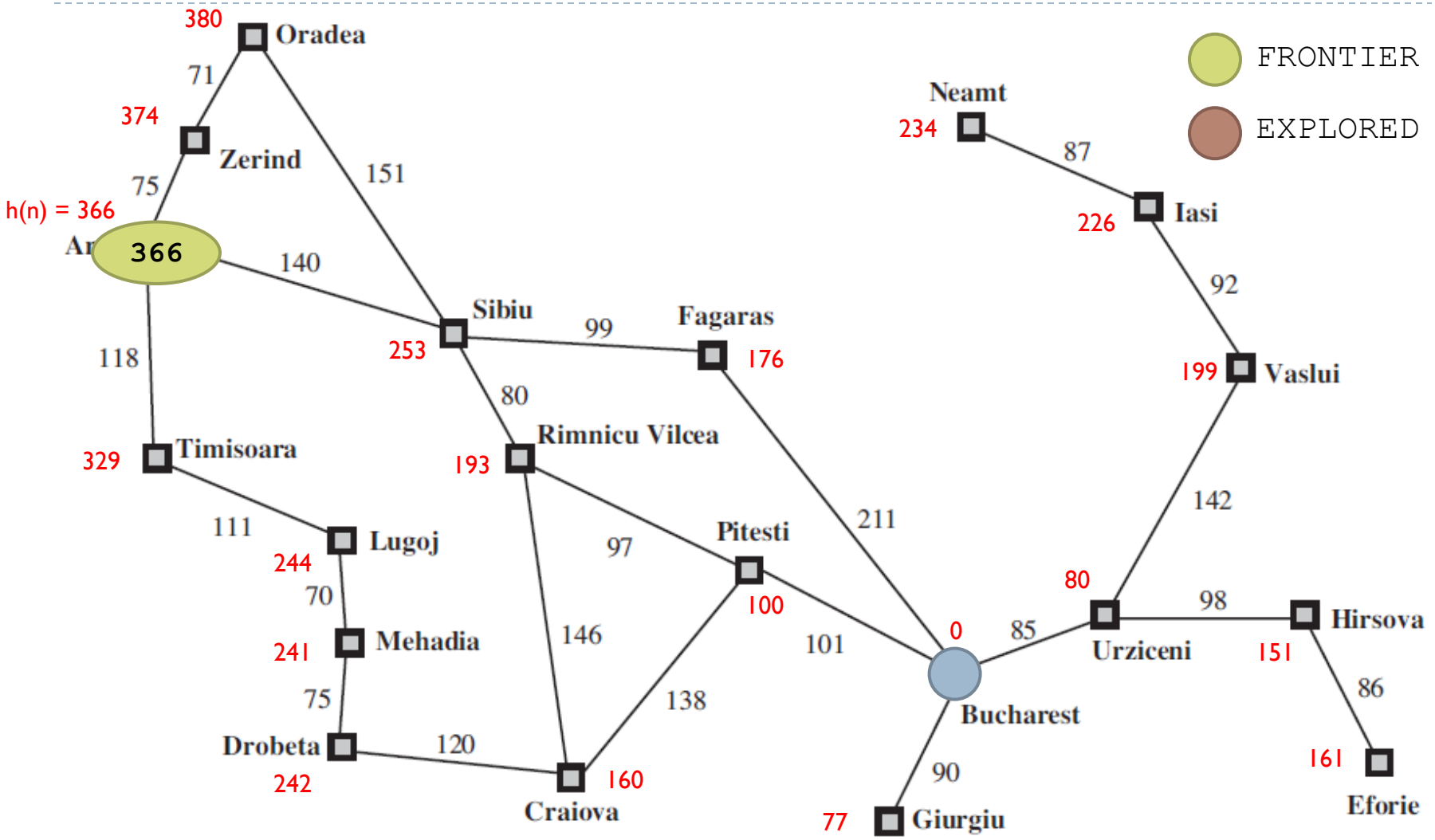
A* Search

- A* search minimizes the total evaluation cost.
- At each state we know the path cost for each frontier node. The remaining cost is estimated by the heuristic function h(n).

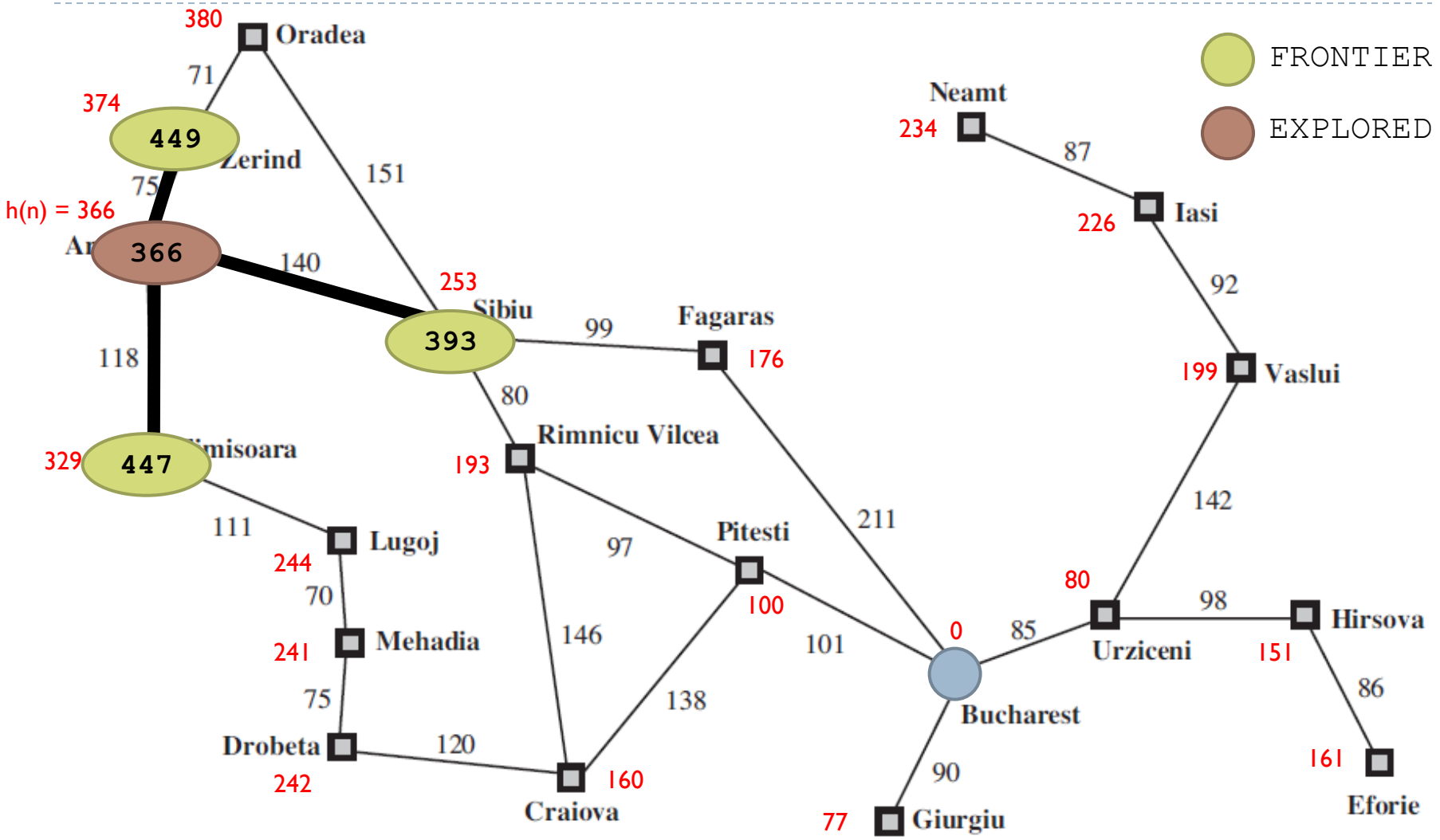
$$f(n) = path\_cost(n) + h(n)$$

- The algorithm is identical to uniform-cost search except for h(n).
- Minimizing path_cost(n) helps us keep the path short, minimizing h(n) helps us to reach the goal faster.

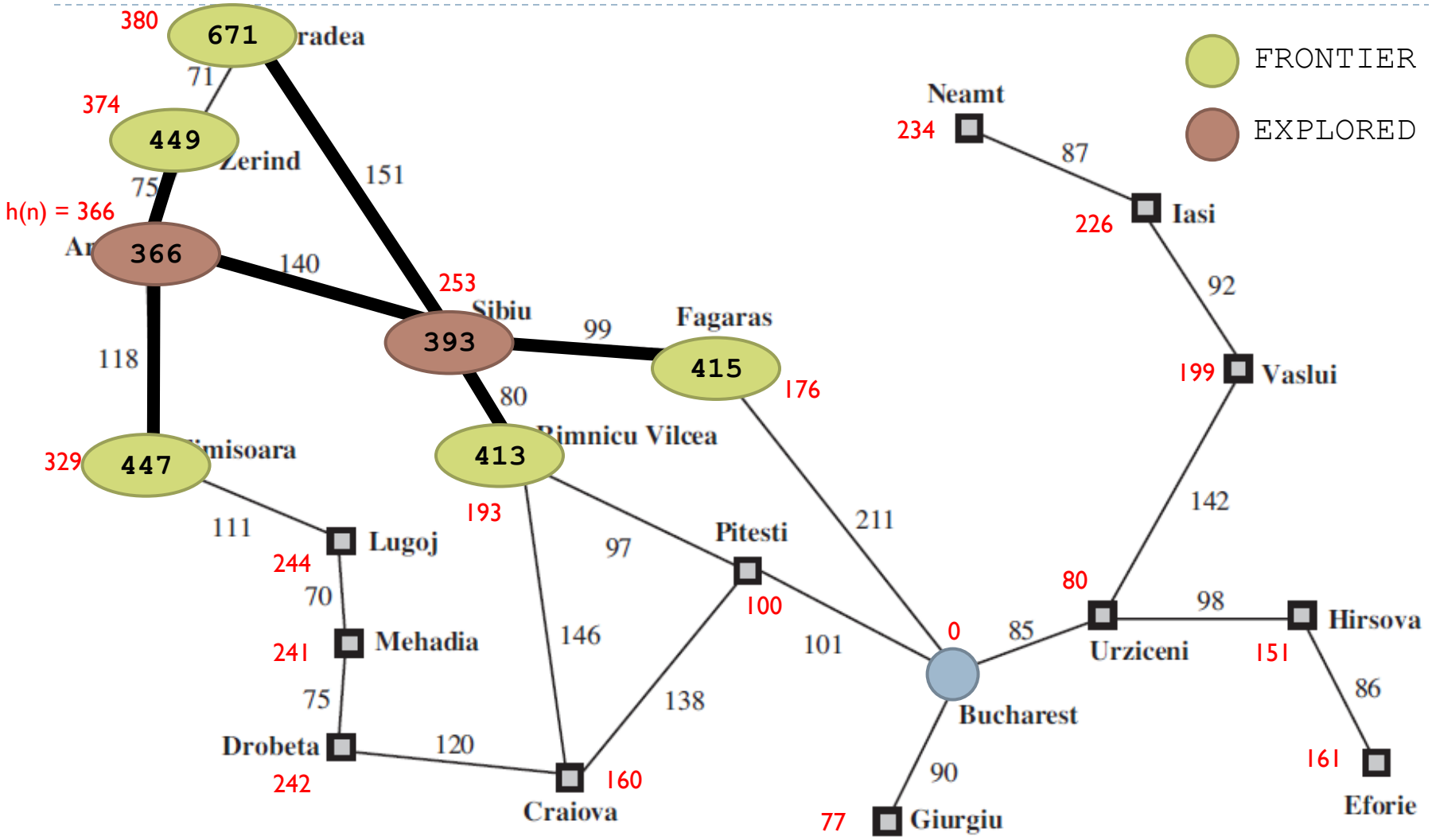
A* Graph Search for Route Finding



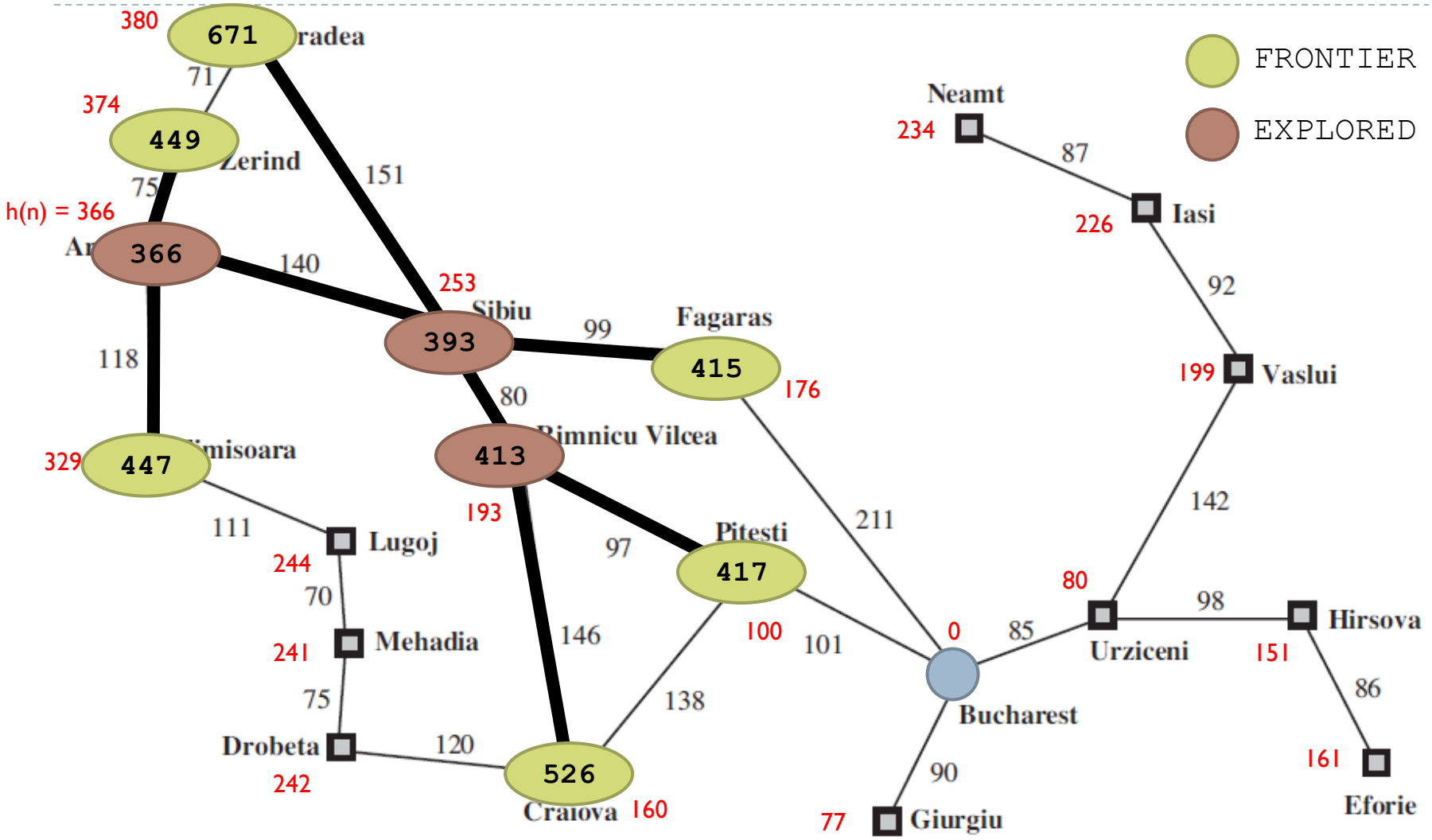
A* Graph Search for Route Finding



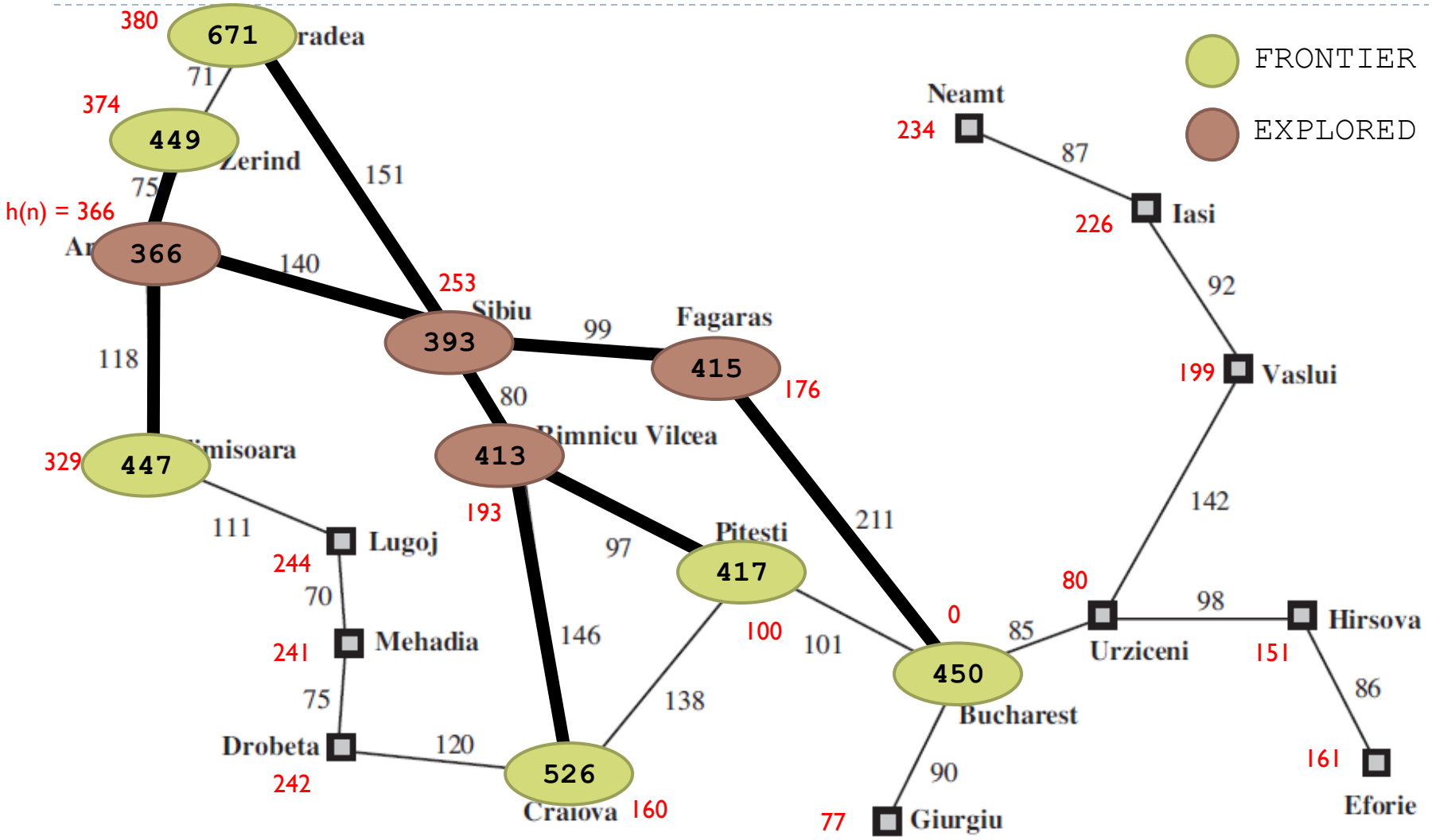
A* Graph Search for Route Finding



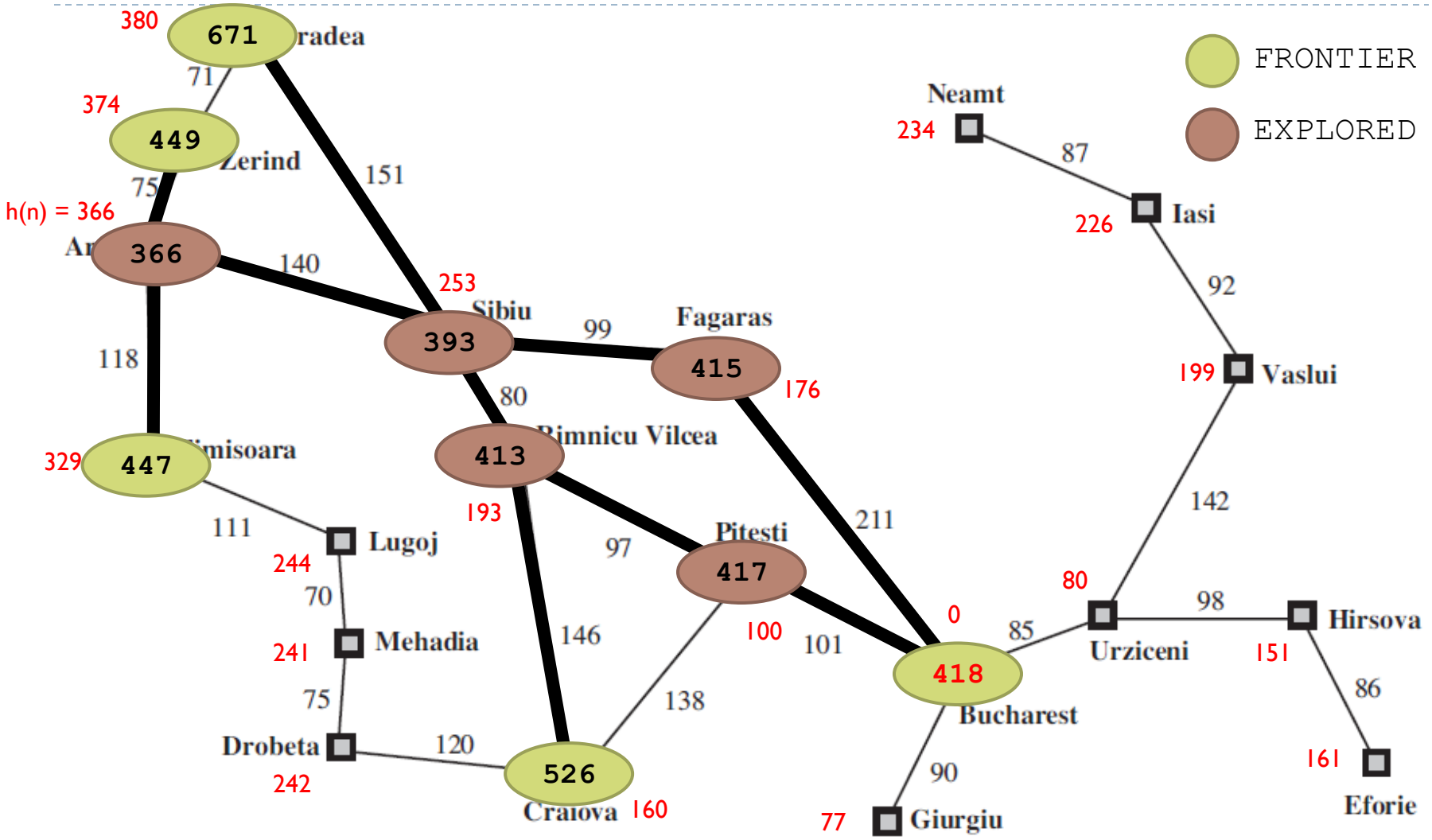
A* Graph Search for Route Finding



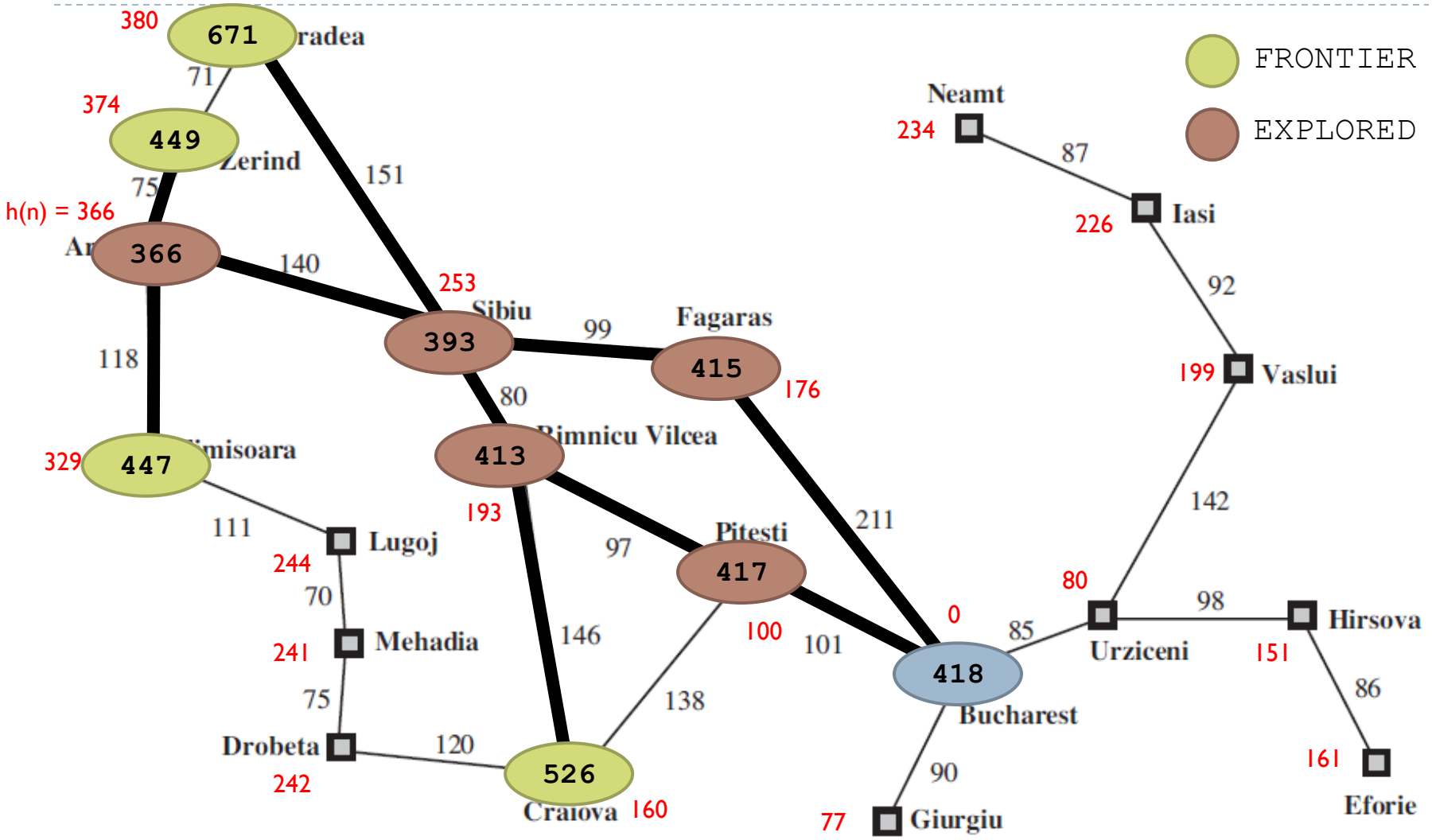
A* Graph Search for Route Finding



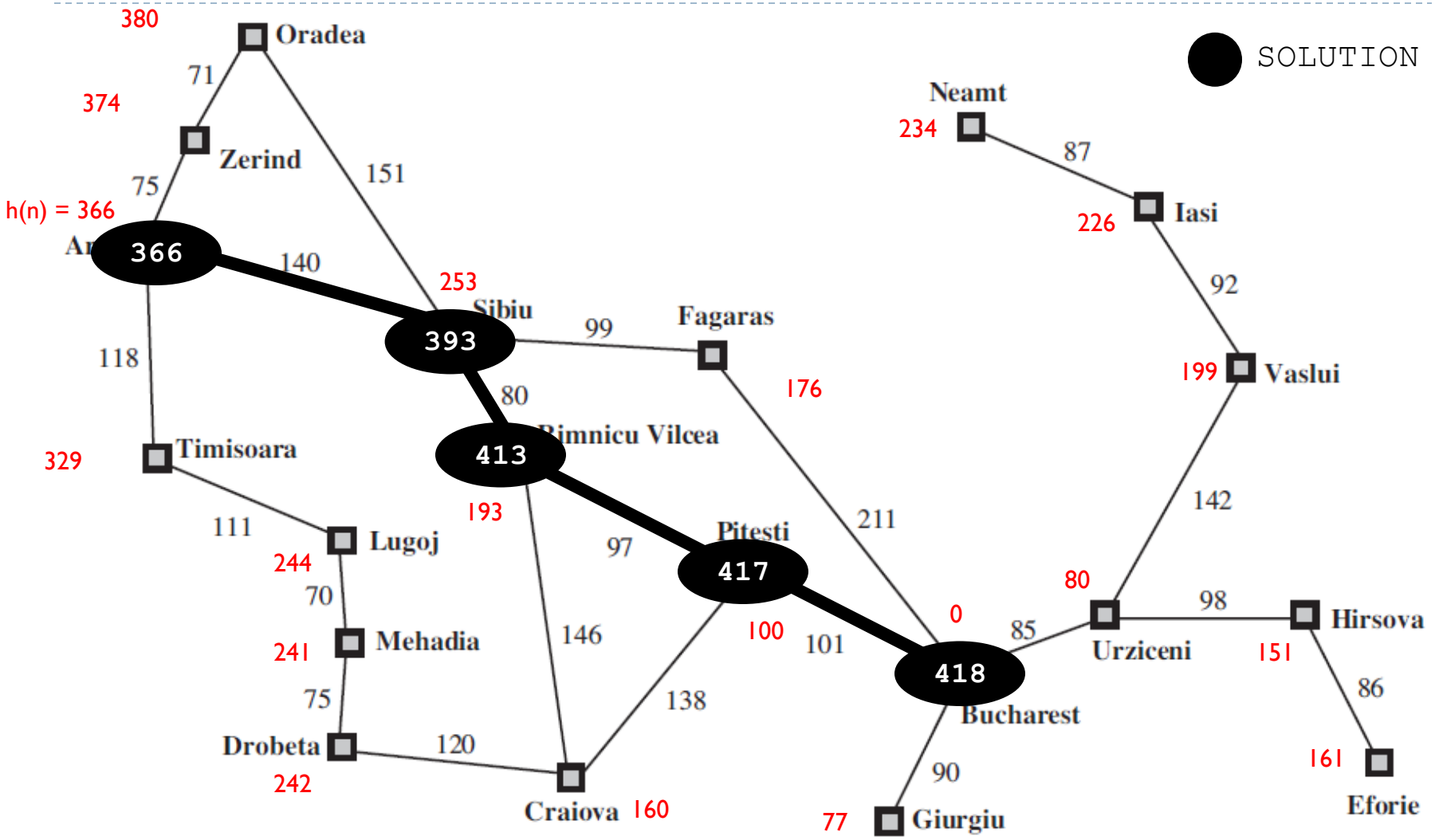
A* Graph Search for Route Finding



A* Graph Search for Route Finding



A* Graph Search for Route Finding



A* Search – Admissibility and Consistency

- Is A* always complete and optimal?
  - It depends on the heuristic part.
- An admissible heuristic is one that never overestimates the cost to reach a solution.

  h(n) <= Cost of reaching the closest goal from n
  - Since straight line distance is always shorter than or equal to the real path cost, this heuristic is admissible.
- A consistent heuristic: the heuristic cost from any node n is less than or equal to the sum of the step cost of going from n to n' and the heuristic cost at n'.

  h(n) <= Cost(n, a, n') + h(n')
- Consistency implies admissibility.

A* Search – Optimality

- A* Tree Search is optimal if h(n) is admissible.
- A* Graph Search is optimal if h(n) is consistent.
- A* search is complete if the branching factor is finite and all path costs are larger than a small finite positive number ε.
- A* search is also optimally efficient among algorithms that search similarly using the same heuristic.

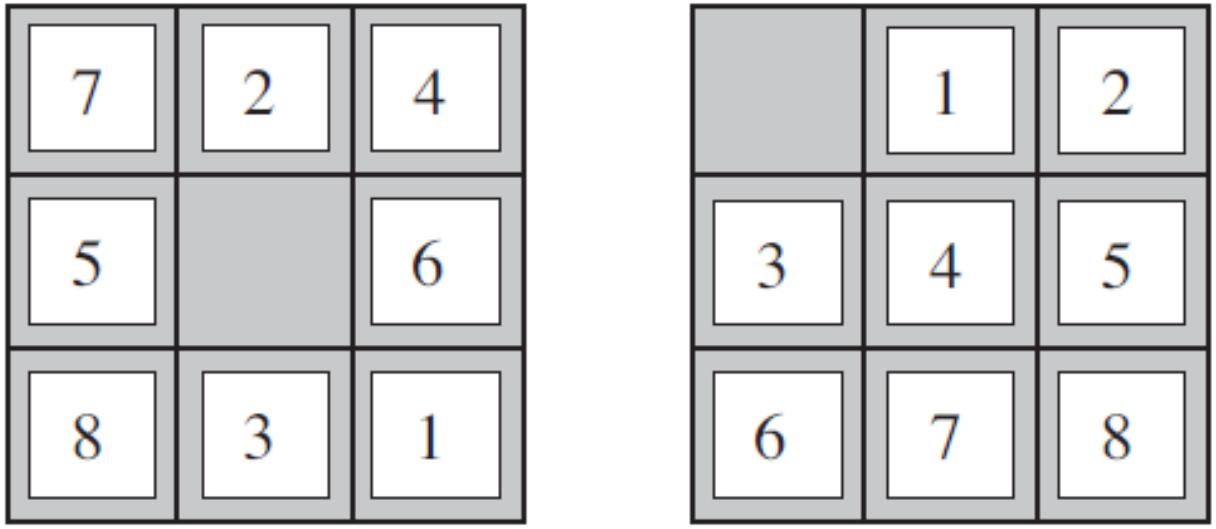
- Still, A* search complexity is exponential in the length of the solution.

Advanced Informed Search Methods

- There are variations of the Best-First and A* Search algorithms that sacrifice optimality to reduce memory complexity.
  - Iterative-Deepening A*
  - Recursive Best-First Search
  - Memory-bounded A*
- We will not cover these algorithms in this course.

More on Heuristic Functions

- The 8-puzzle



A typical configuration, solution is 26 steps away.

- If average solution depth is 22, and branching factor is 3 then an exhaustive tree search expands about 3^{22} states.
- A graph search reduces this down considerably.

More on Heuristic Functions

- h_1 = number of misplaced tiles
- h_2 = sum of the distances of tiles from their goal positions (Manhattan/City block distance)
- For the initial state in the previous slide, h_1=8, h_2=18.
- Good! They do not overestimate the cost, i.e. they are smaller than actual solution.
- How can we compare the two heuristics?
- Effective Branching Factor:
  - If A* Search produces a solution using N nodes and the solution depth is d, then the effective branching factor b* is the branching factor of a uniform tree containing N + 1 nodes at depth d:

    $$N+1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$

  - E.g. If N = 52, d = 5, then b* = 1.92

More on Heuristic Functions

- Comparing the two heuristics on a randomly generated set of problems:

d	$b^*(h_1)$	$b^*(h_2)$
2	1.79	1.79
4	1.48	1.45
6	1.34	1.30
8	1.33	1.24
10	1.38	1.22
12	1.42	1.24
14	1.44	1.23
16	1.45	1.25
18	1.46	1.26

- The smaller the b*, the better the heuristic

More on Heuristic Functions

- For the particular selection of h_1 and h_2, we can make a direct comparison:

$$h_2(n) >= h_1(n)$$

- In general, heuristics with larger values are better as long as they are consistent and are not computationally too expensive.
- All consistent heuristics are upper bounded by the true path cost to the goal. The closer we get to the true value, the better it is.

Where do heuristics come from?

- Generating heuristics by relaxation, i.e. we remove constraints from the description of the problem

  Rule: A tile can move A $\rightarrow$ B if (~~A is adjacent to B~~) and (~~B is blank~~)

  Removing the constraint leads us to h_1 since one can move tiles anywhere directly.

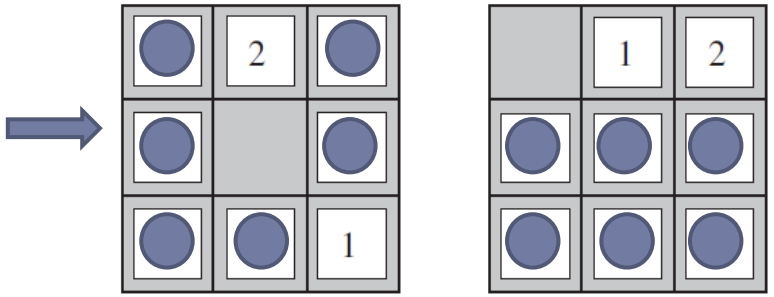
  Removing the constraint leads us to h_2 since tiles can move over occuped tiles.

- The cost of a relaxed problem is less than the actual cost so the generated heuristic is admissible. It is also consistent.
- What if we generate many heuristics but not sure which one to use?

$$h = \max(h_1, h_2, h_3, \ldots, h_N)$$

Where do heuristics come from?

- Generating heuristics by problem subdivision:
- We can create a database of solutions to subproblems:
  - E.g. Cost of getting tiles 1, 2 to their correct positions (regardless of other tiles)
  - E.g. Cost of getting 2, 4, 6, 8 to their correct positions …



- Each subproblem generates a lower bound on the true cost and this cost can be used as a heuristic.
- We can combine the heuristics from as many subproblems we want.

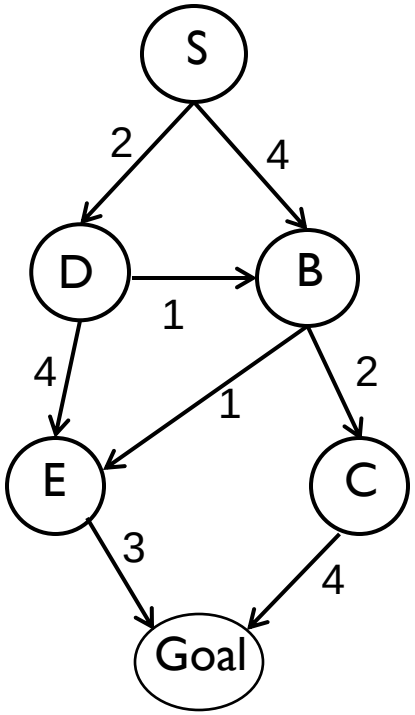
Where do heuristics come from?

- Generate heuristics by experience:
  - We can generate solutions for a random set of problems.
  - Then use machine learning methods to obtain information about similar situations.

Note: Problem solving techniques we just saw work when the domain is fully observable, discrete and deterministic.

Home Exercise

- Apply A* search to the following graph (when necessary break the ties by selecting the alphabetically-first node).
- Is the result an optimum path? If it is not, why?



State	h
S	7
D	7
B	4
C	2
E	3
G	0