COMP219: Artificial Intelligence

Lecture 10: Heuristic Search

Class Tests

- Class Test 1 (Prolog):
  - Friday 17th November (Week 8), 15:00-17:00.
- Class Test 2 (Everything but Prolog)
  - Friday 15th December (Week 12), 15:00-17:00
- Location
  - depending on what letter your last name starts with you should report to one of these:
    - JHERD-LT: A-E
    - DUN-LT2: F-Q
    - CHAD-ROTB: R-Z

Overview

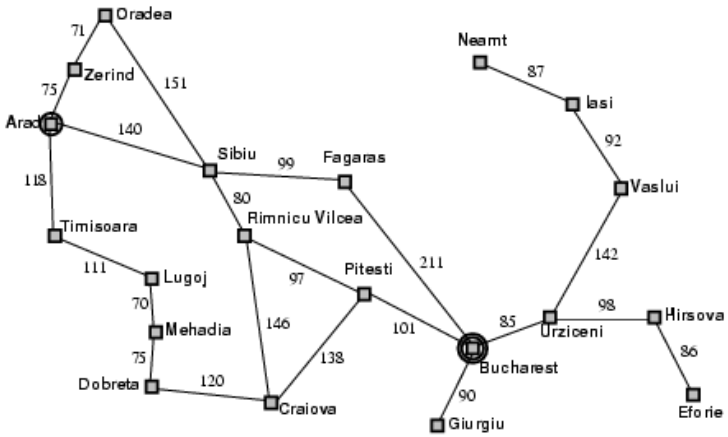
- Last time
  - Depth-limited, iterative deepening and bi-directional search
  - Avoiding repeated states
- Today
  - Show how applying knowledge of the problem can help
  - Introduce uniform cost search: dependent on the cost of each node
  - Introduce heuristics: rules of thumb
  - Introduce heuristic search
    - Greedy search
    - A* search
- Learning outcome covered today:
  Identify, contrast and apply to simple examples the major search techniques that have been developed for problem-solving in AI

Real Life Problems

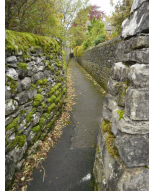
- Whatever search technique we use, we have exponential time complexity
- Tweaks to the algorithm will not reduce this to polynomial
- We need problem specific knowledge to ***guide*** the search
- Simplest form of problem specific knowledge is heuristics
- Usual implementation in search is via an ***evaluation function*** which indicates desirability of expanding a node

Path Cost Function

Recall: we have a **path cost function g(n)**, which gives the cost of each path. This is comprised of the step costs of each action on the path.

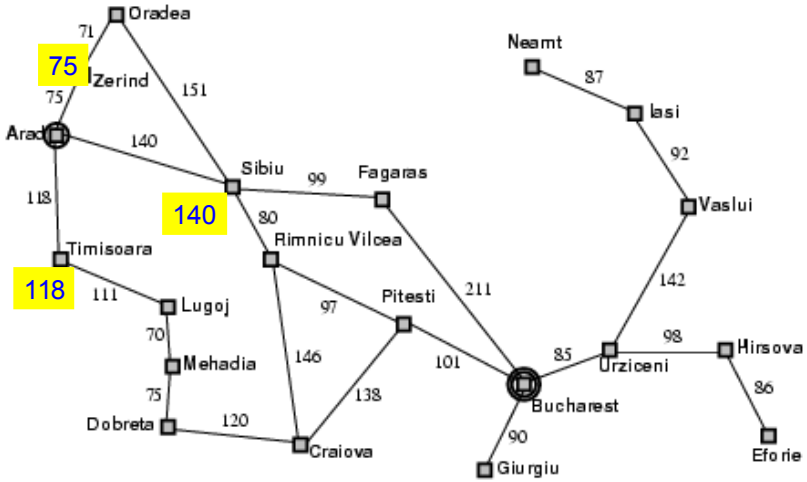


Finding The Best Paths



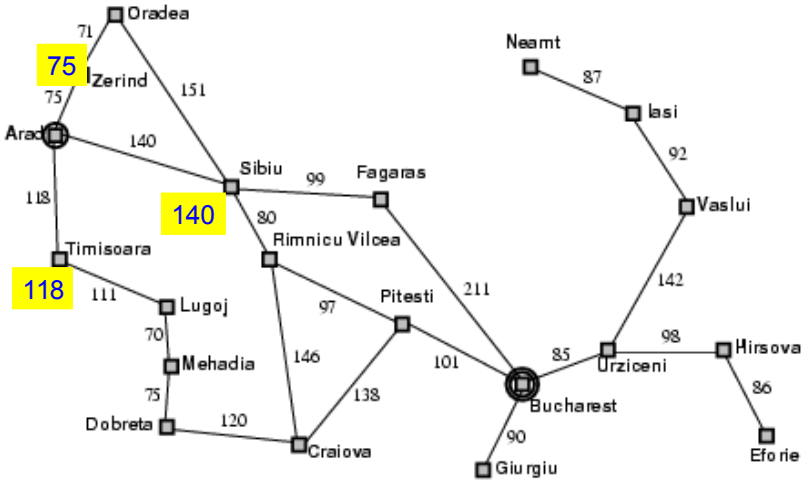
- Why not expand the *cheapest path first*?
- Intuition: cheapest is likely to be best!
- Performance is like breadth-first search but we use the minimum cost path rather than the shallowest to expand
- **Uniform cost search** orders the agenda as a priority queue using the lowest path cost of a node

Cheapest First



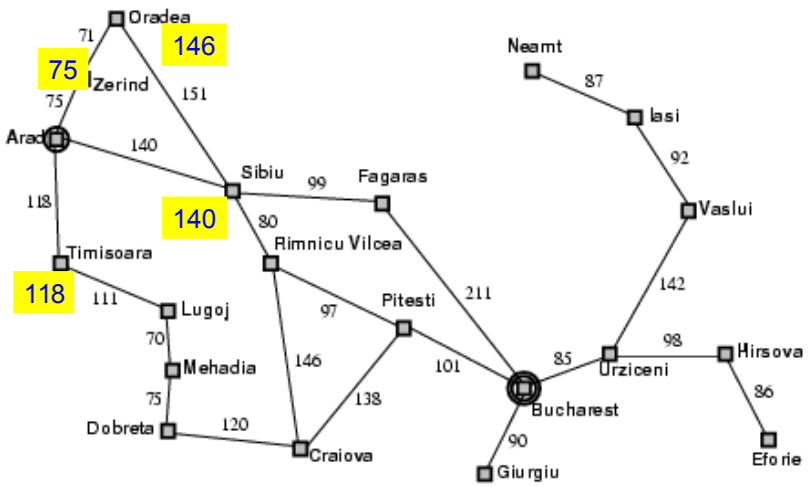
Cheapest First

Choose Zerind



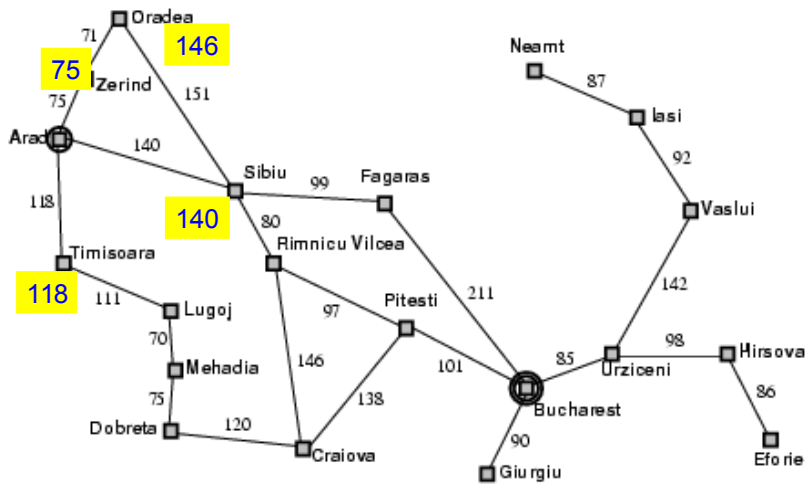
Cheapest First

Choose Zerind



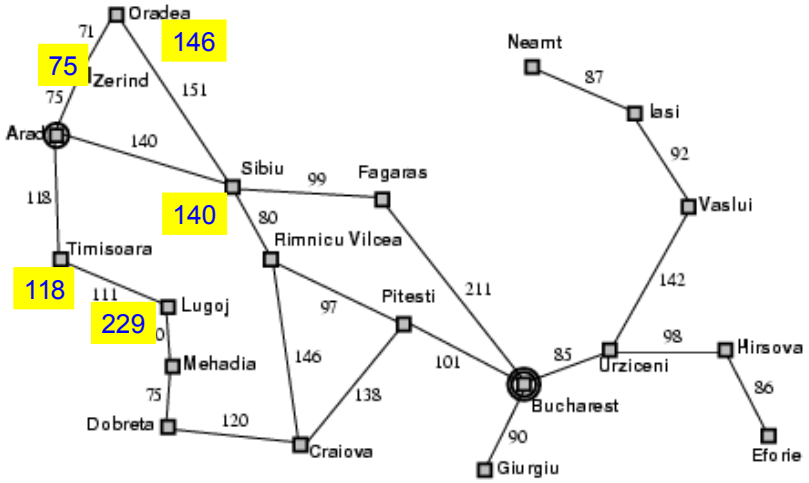
Cheapest First

Choose Zerind Choose Timisoara



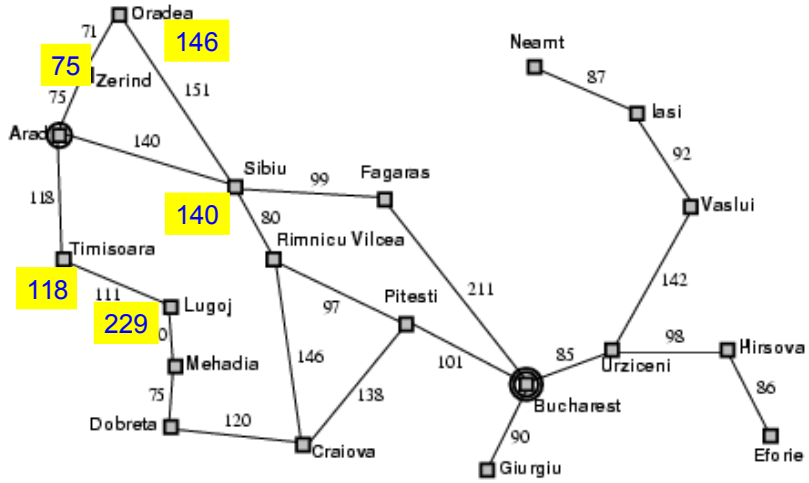
Cheapest First

Choose Zerind Choose Timisoara



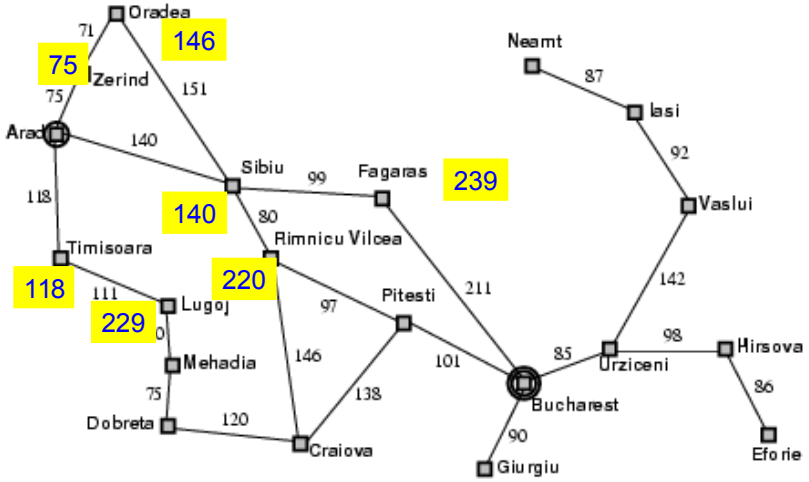
Cheapest First

Choose Zerind Choose Timisoara Choose Sibiu



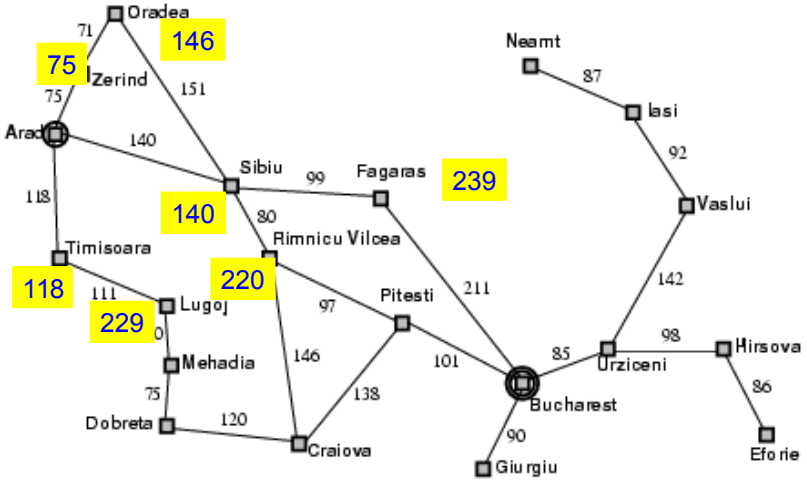
Cheapest First

Choose Zerind | Choose Timisoara | Choose Sibiu



Cheapest First

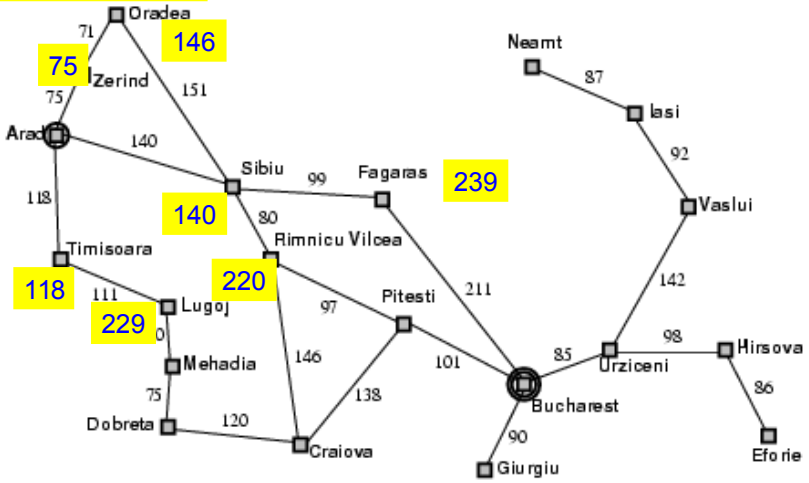
Choose Zerind | Choose Timisoara | Choose Sibiu | Choose Oradea



Cheapest First

Choose Zerind | Choose Timisoara | Choose Sibiu | Choose Oradea

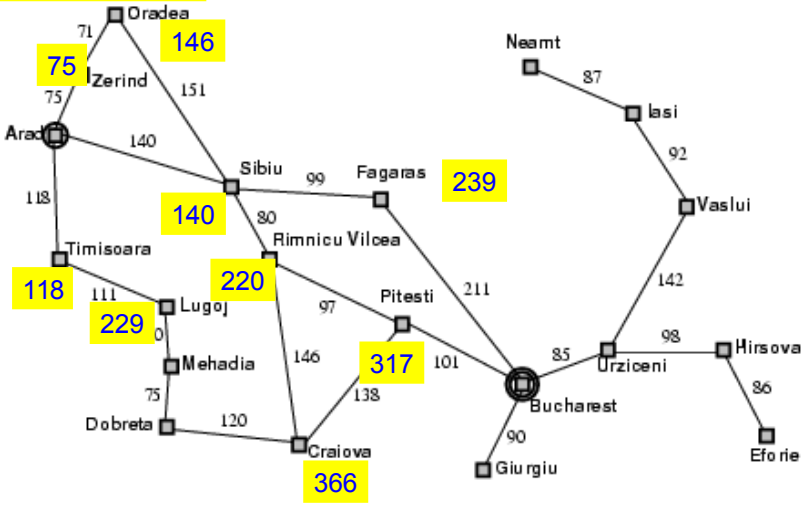
Choose Rimnicu Vilcea



Cheapest First

Choose Zerind | Choose Timisoara | Choose Sibiu | Choose Oradea

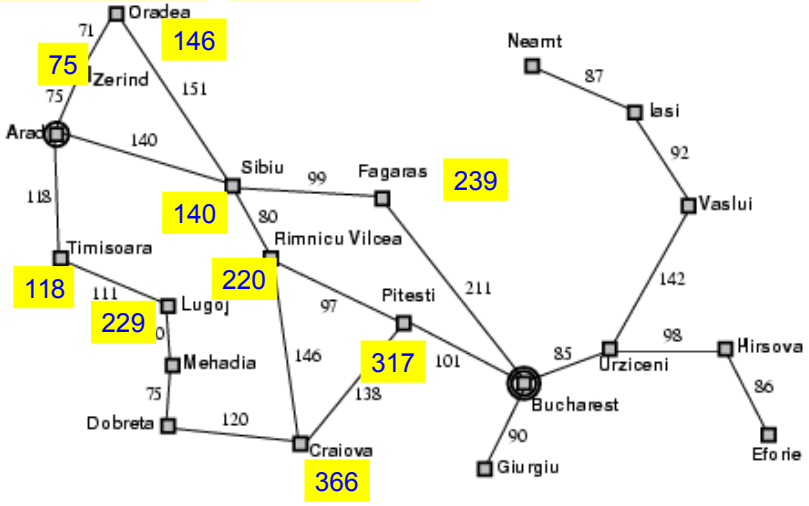
Choose Rimnicu Vilcea



Cheapest First

Choose Zerind | Choose Timisoara | Choose Sibiu | Choose Oradea

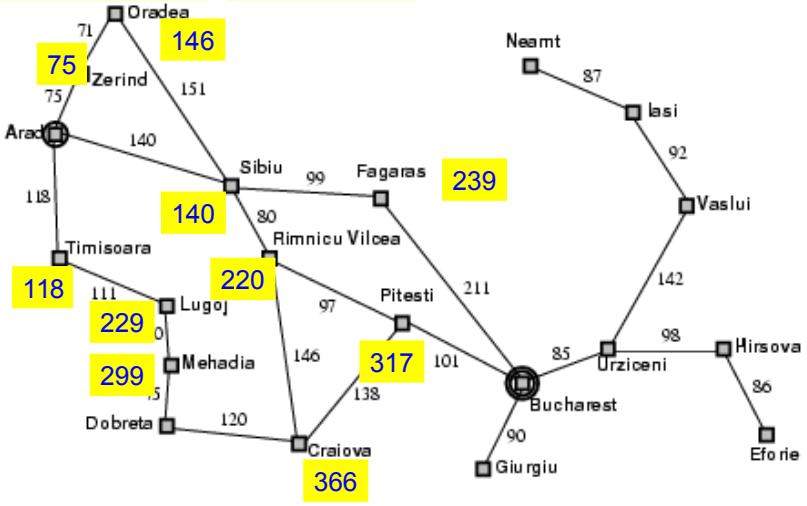
Choose Rimnicu Vilcea | Choose Lugoj



Cheapest First

Choose Zerind | Choose Timisoara | Choose Sibiu | Choose Oradea

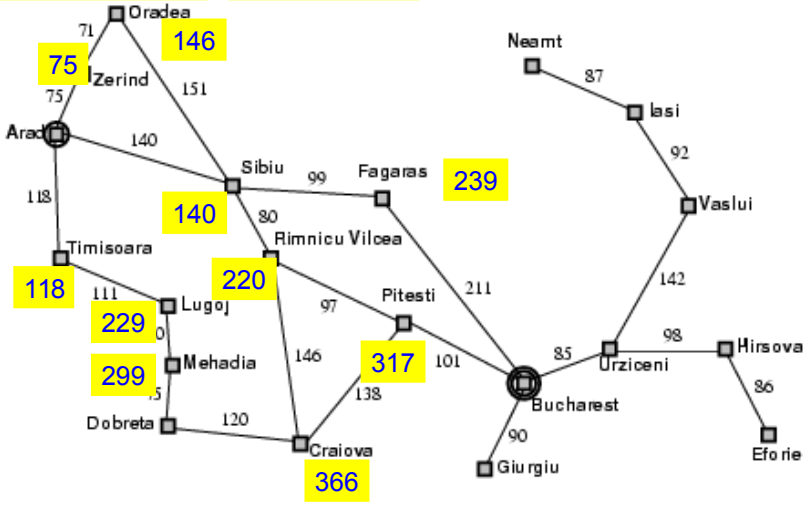
Choose Rimnicu Vilcea | Choose Lugoj



Cheapest First

Choose Zerind | Choose Timisoara | Choose Sibiu | Choose Oradea

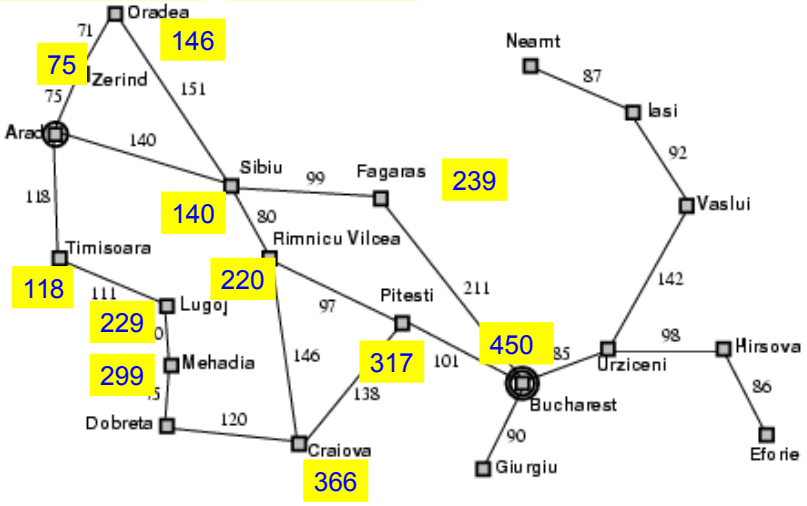
Choose Rimnicu Vilcea | Choose Lugoj | Choose Fagaras



Cheapest First

Choose Zerind | Choose Timisoara | Choose Sibiu | Choose Oradea

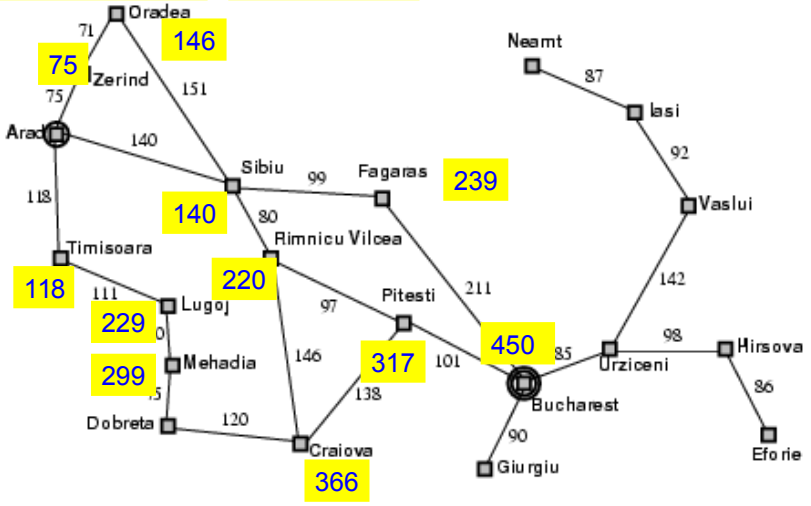
Choose Rimnicu Vilcea | Choose Lugoj | Choose Fagaras



Cheapest First

Choose Zerind | Choose Timisoara | Choose Sibiu | Choose Oradea

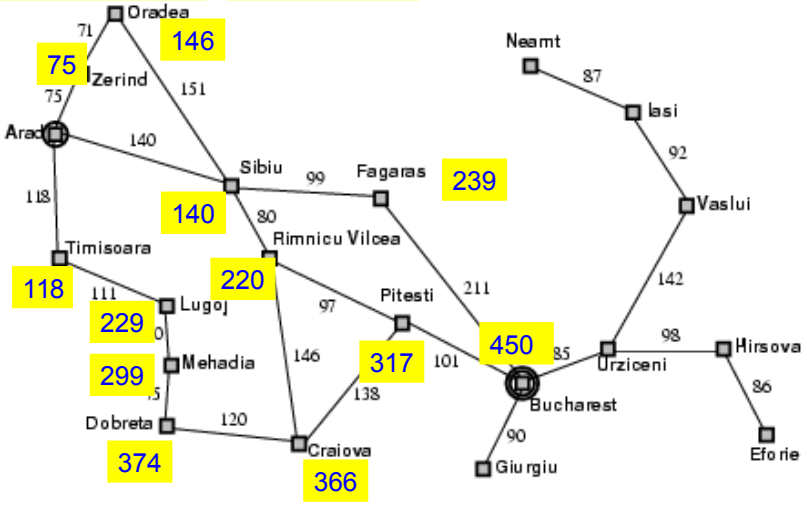
Choose Rimnicu Vilcea | Choose Lugoj | Choose Fagaras | Choose Mehadia



Cheapest First

Choose Zerind | Choose Timisoara | Choose Sibiu | Choose Oradea

Choose Rimnicu Vilcea | Choose Lugoj | Choose Fagaras | Choose Mehadia

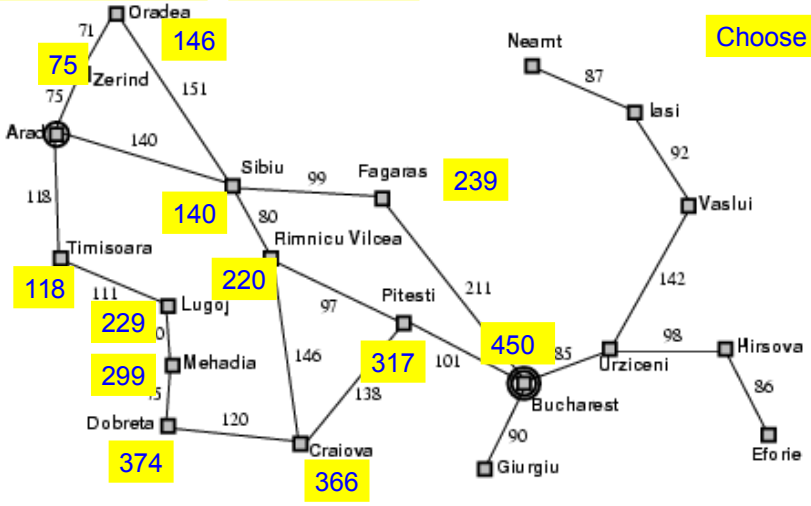


Cheapest First

Choose Zerind | Choose Timisoara | Choose Sibiu | Choose Oradea

Choose Rimnicu Vilcea | Choose Lugoj | Choose Fagaras | Choose Mehadia

Choose Pitesti

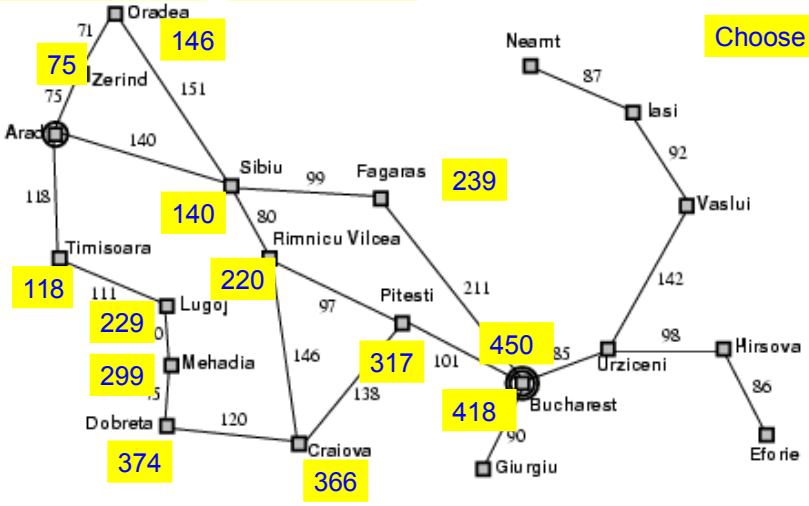


Cheapest First

Choose Zerind | Choose Timisoara | Choose Sibiu | Choose Oradea

Choose Rimnicu Vilcea | Choose Lugoj | Choose Fagaras | Choose Mehadia

Choose Pitesti



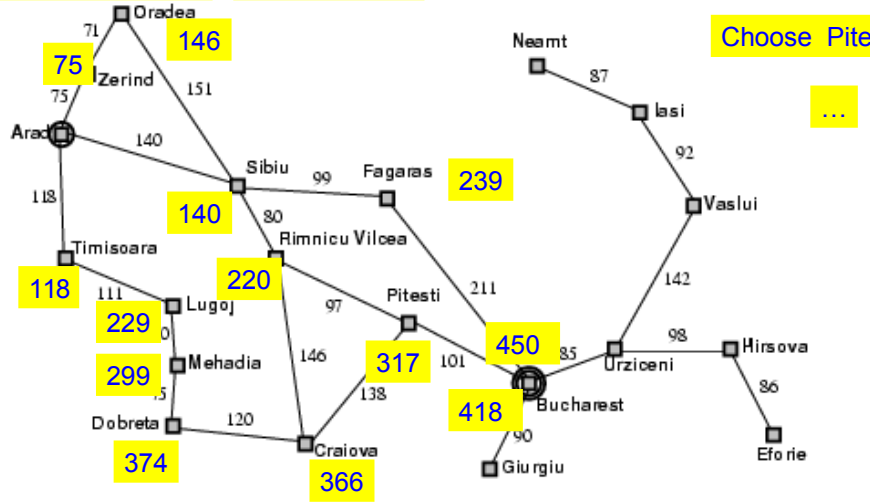
Cheapest First

Choose Zerind | Choose Timisoara | Choose Sibiu | Choose Oradea

Choose Rimnicu Vilcea | Choose Lugoj | Choose Fagaras | Choose Mehadia

Choose Pitesti

...



General Algorithm for Uniform Cost Search

```
agenda = [initial state];
while agenda not empty do
  take node from agenda such that
    g(node) = min { g(n) | n in agenda}
  if node is goal state then
    return solution;
  new nodes = apply operations to node;
  add new nodes to the agenda;
```


General Algorithm for Uniform Cost Search

```
agenda = [initial state];
while agenda not empty do
  take node from agenda such that
    g(node) = min { g(n) | n in agenda}
  if node is goal state then
    return solution;
  new nodes = apply operations to node;
  add new nodes to the agenda;
```

How?
→ priority queue

Properties of Uniform Cost Search

- Uniform cost search guaranteed to find cheapest solution assuming path costs grow monotonically, i.e. the cost of a path never decreases as we move along it
- In other words, adding another step to the solution makes it more costly, i.e.

  `g(successor(n)) > g(n).`
- If path costs don't grow monotonically, then exhaustive search is required
- Still requires many nodes to be examined

Informed Strategies

- Use problem-specific knowledge
- More efficient than blind search
- The most ***promising*** path first!
- Rather than trying all possible search paths, you try to focus on paths that seem to be getting you nearer your target/goal state

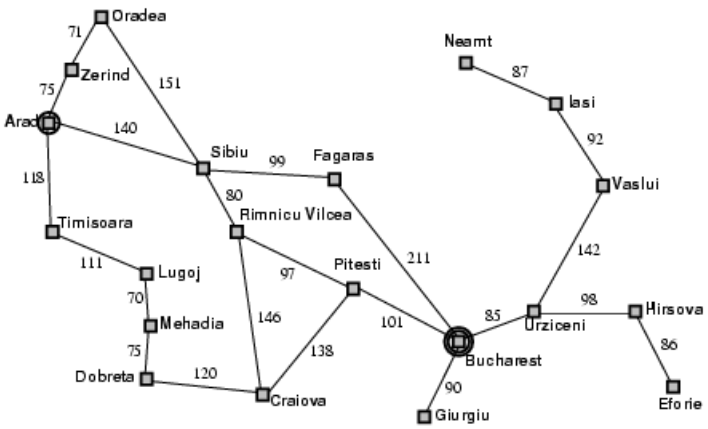
Greedy Search

- Heuristics **estimate cost of cheapest path from node to solution**
- We have a *heuristic function*,

  `h : Nodes → R`

  which estimates the distance from the node to the goal
- `h` can be any function
  - but should have `h(n) = 0` if `n` is a goal
- Example: In route finding, heuristic might be straight line distance from node to destination
- Greedy search: expand node that *appears to be* closest to goal

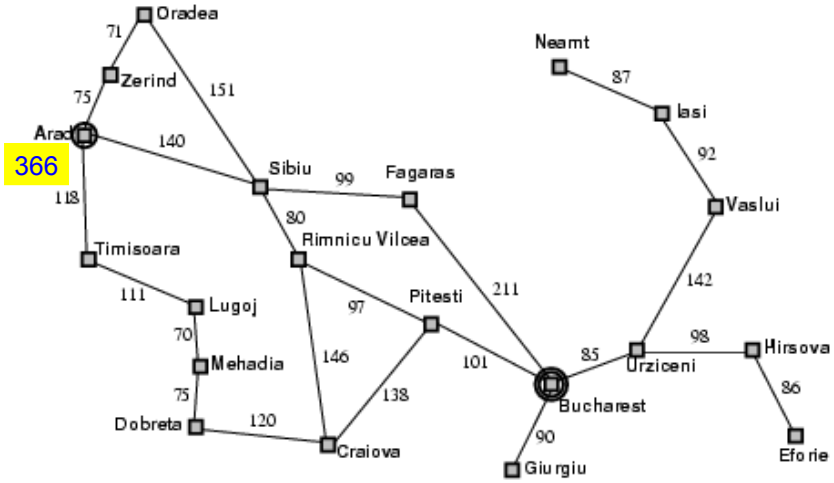
Romania Example



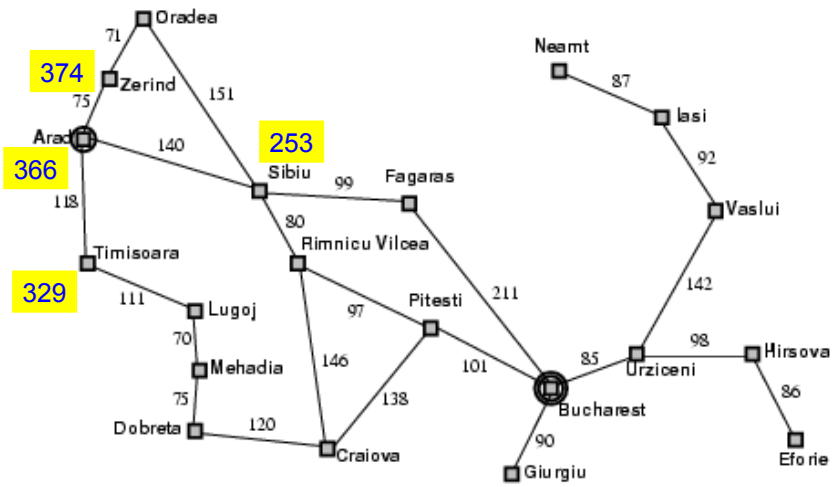
Straight-line distance to Bucharest

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	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Search Example

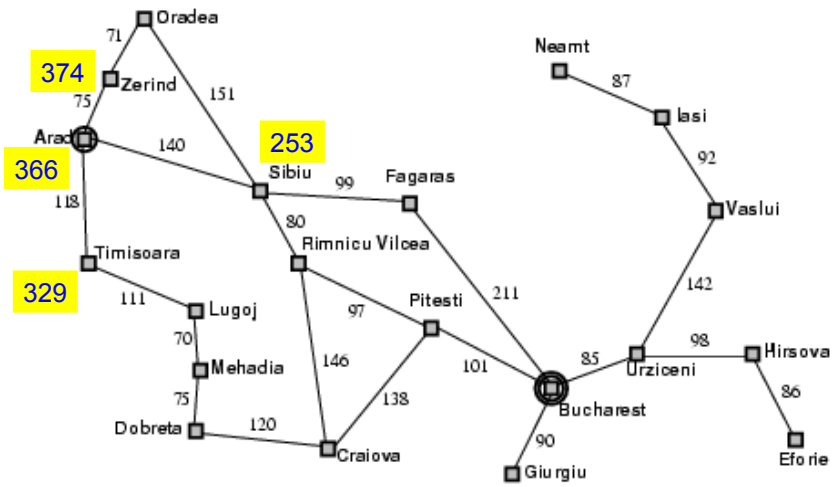


Greedy Search Example



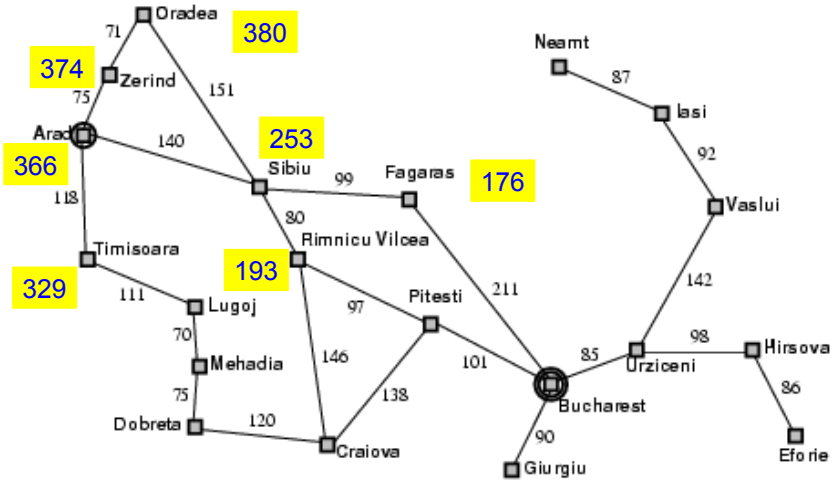
Greedy Search Example

Choose Sibiu



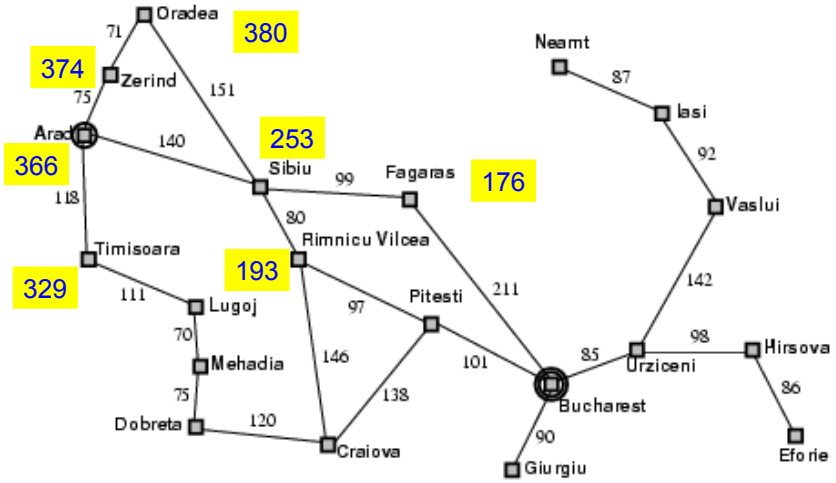
Greedy Search Example

Choose Sibiu



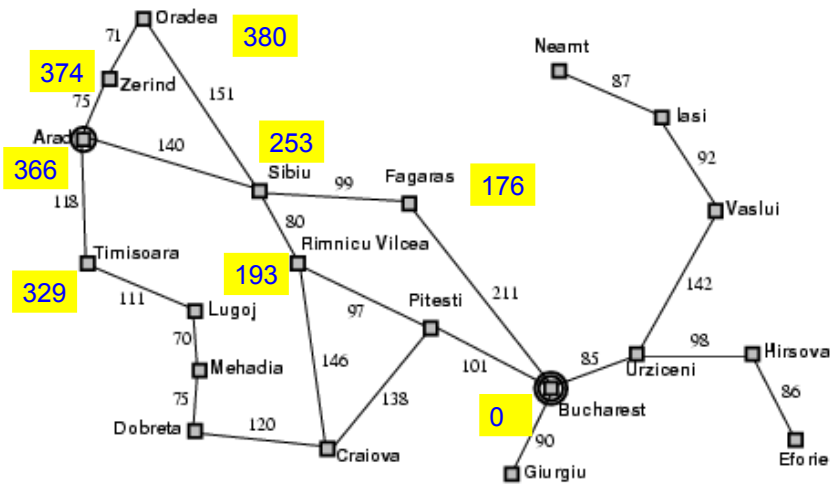
Greedy Search Example

Choose Sibiu Choose Fagaras



Greedy Search Example

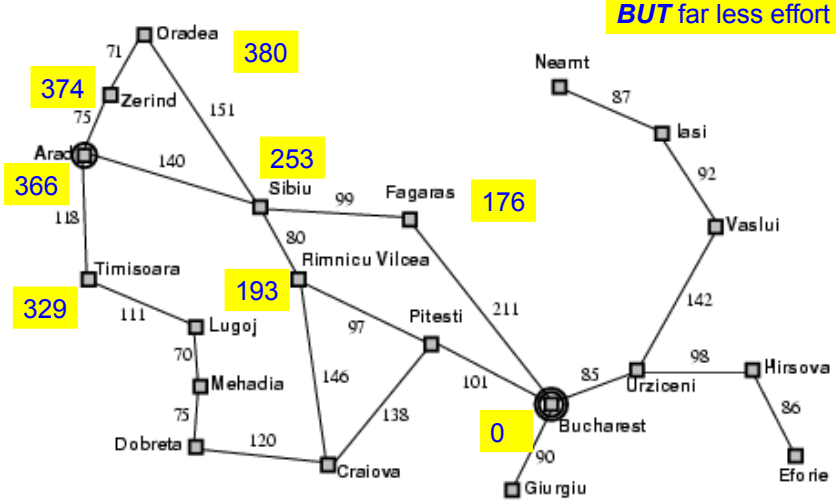
Choose Sibiu Choose Fagaras



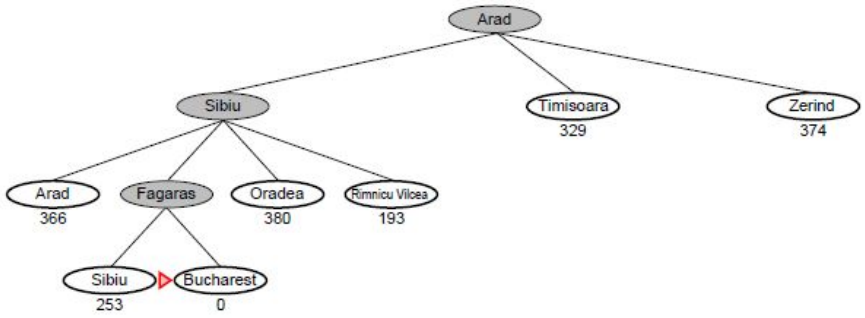
Greedy Search Example

Choose Sibiu Choose Fagaras

NOT the cheapest path
BUT far less effort



Search Tree



Exercise

- Suppose that in the Romania example the initial state is Iasi and the goal state is Fagaras. How do you think a greedy search, using the straight line distance as a heuristic, might proceed?

General Algorithm for Greedy Search

```
agenda = [initial state];
while agenda not empty do
  take node from agenda such that
    h(node) = min { h(n) | n in agenda}
  if node is goal state then
     return solution;
  new nodes = apply operations to node;
  add new nodes to the agenda;
```


Properties of Greedy Search

- Greedy search finds solutions quickly
- Doesn't always find the best
  - *'greedy' = short sighted*
- May not find a solution if there is one (incomplete)
  - Only looking at *current node. Ignores past!*

A* Search

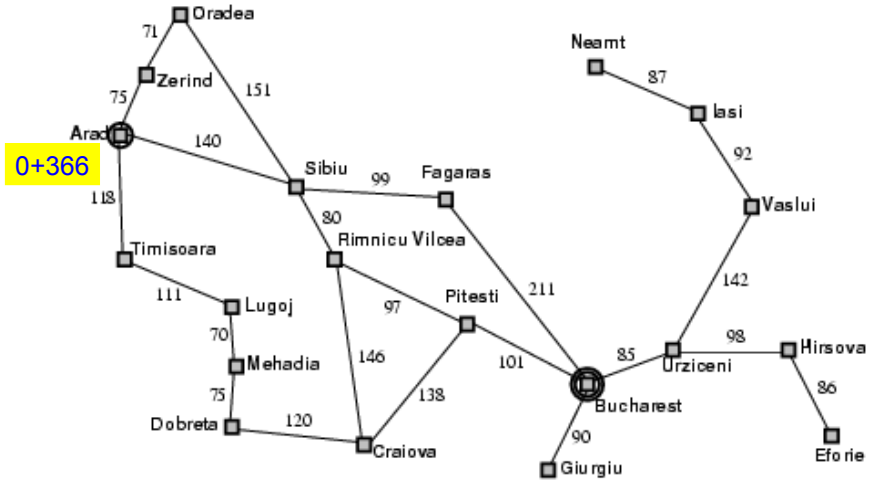
- A* is a very efficient search strategy
- Basic idea is to combine uniform cost search and greedy search
- We look at the cost so far ***and*** the estimated cost to goal
- Gives heuristic `f`:

  `f(n) = g(n) + h(n)`
- where

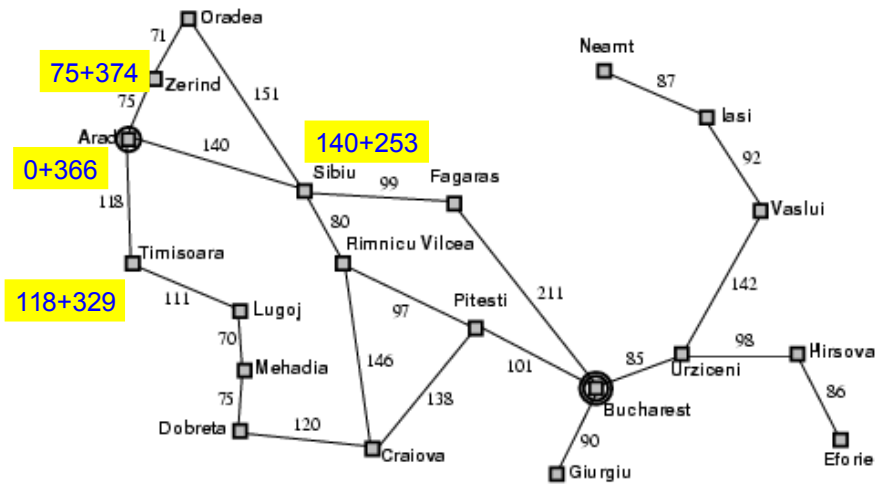
  `g(n)` is path cost of `n`

  `h(n)` is expected cost of cheapest solution from `n`
- Aims to minimise overall cost

A* Search Example

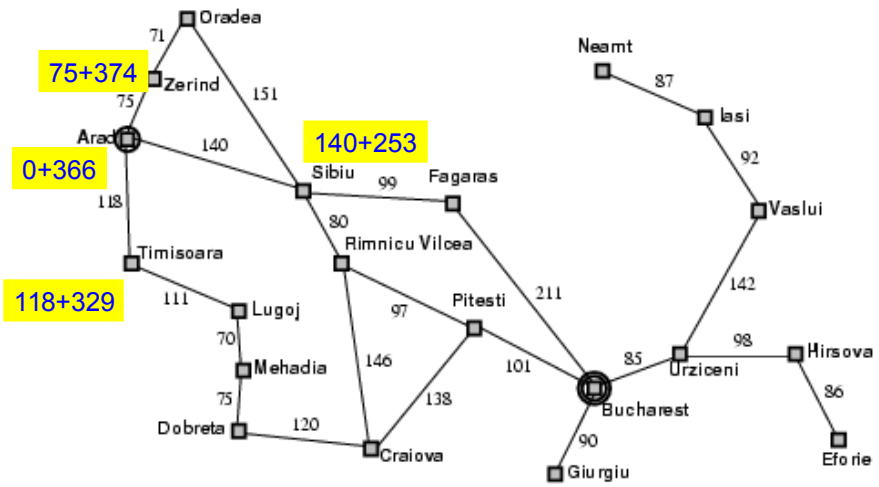


A* Search Example



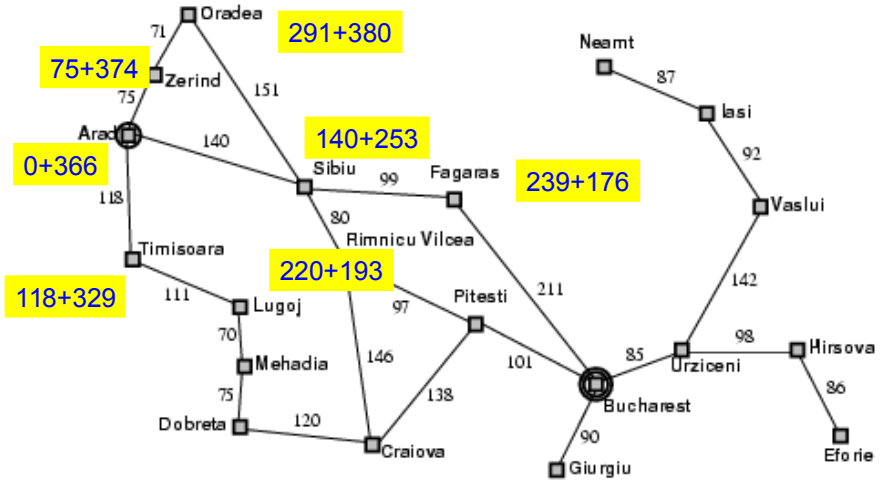
A* Search Example

Choose Sibiu



A* Search Example

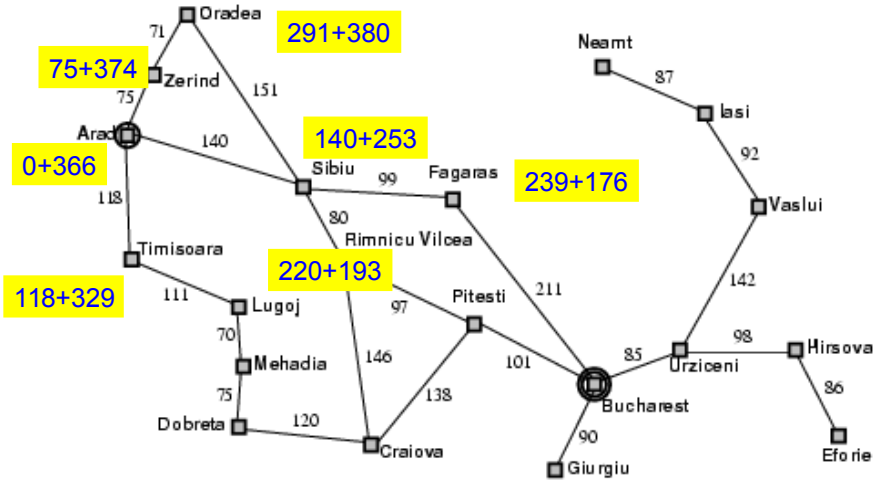
Choose Sibiu



A* Search Example

Choose Sibiu

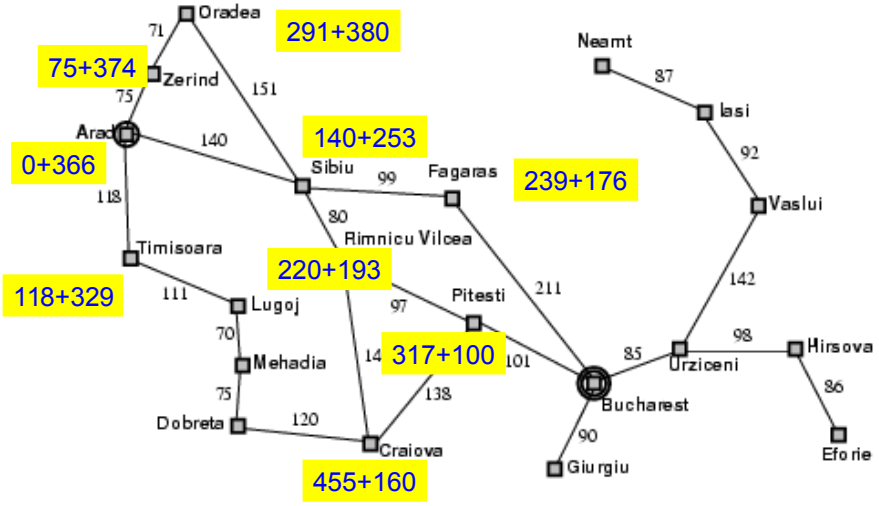
Choose Rimnicu Vilcea
413 < 415



A* Search Example

Choose Sibiu

Choose Rimnicu Vilcea
413 < 415

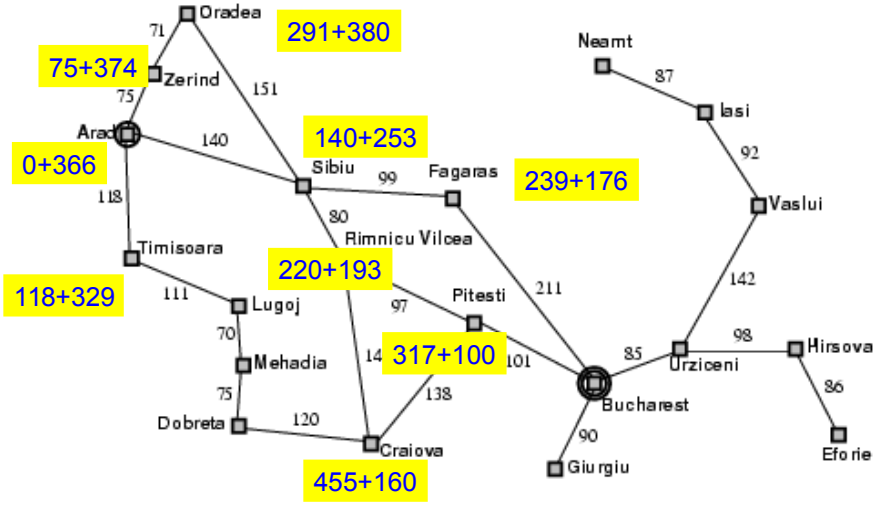


A* Search Example

Choose Sibiu

Choose Rimnicu Vilcea
413 < 415

Choose Fagaras
415 < 417

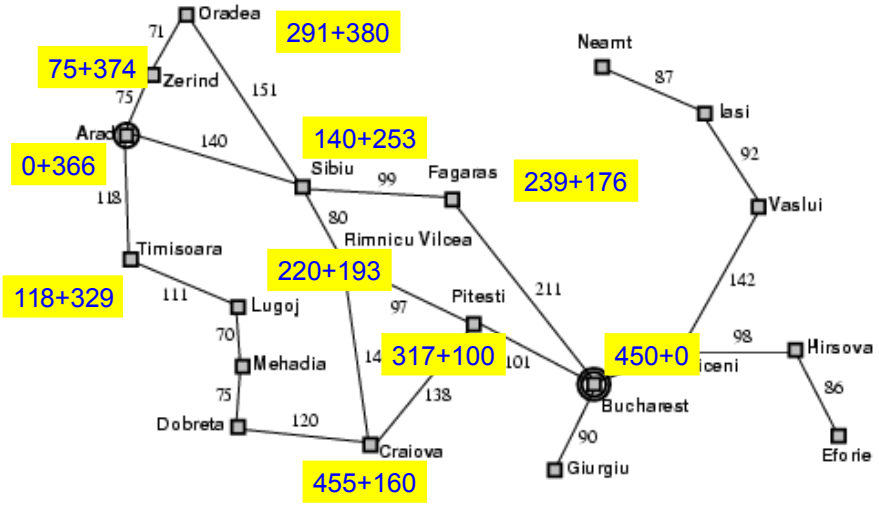


A* Search Example

Choose Sibiu

Choose Rimnicu Vilcea
413 < 415

Choose Fagaras
415 < 417



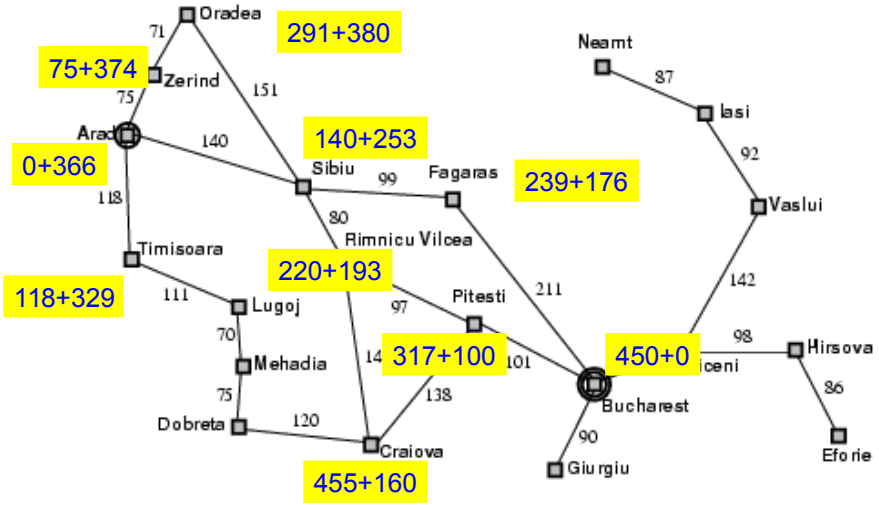
A* Search Example

Choose Sibiu

Choose Rimnicu Vilcea
413 < 415

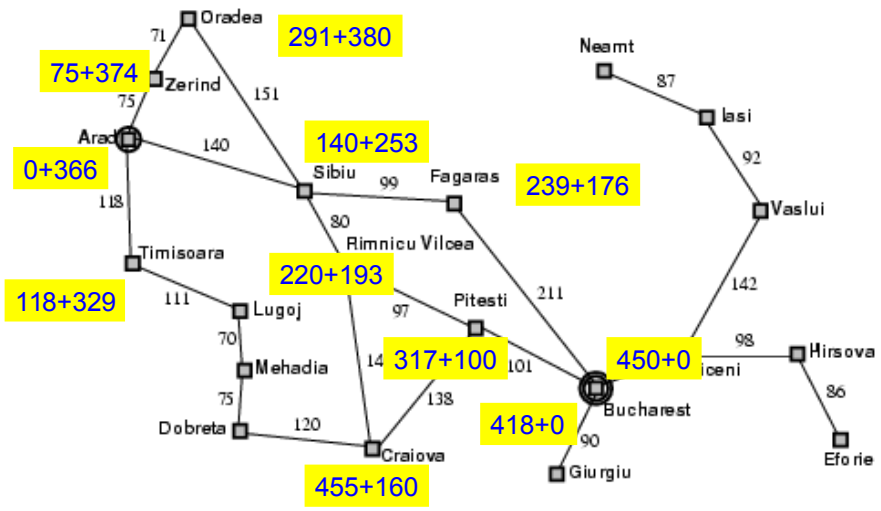
Choose Fagaras
415 < 417

Choose Pitesti
417 < 450



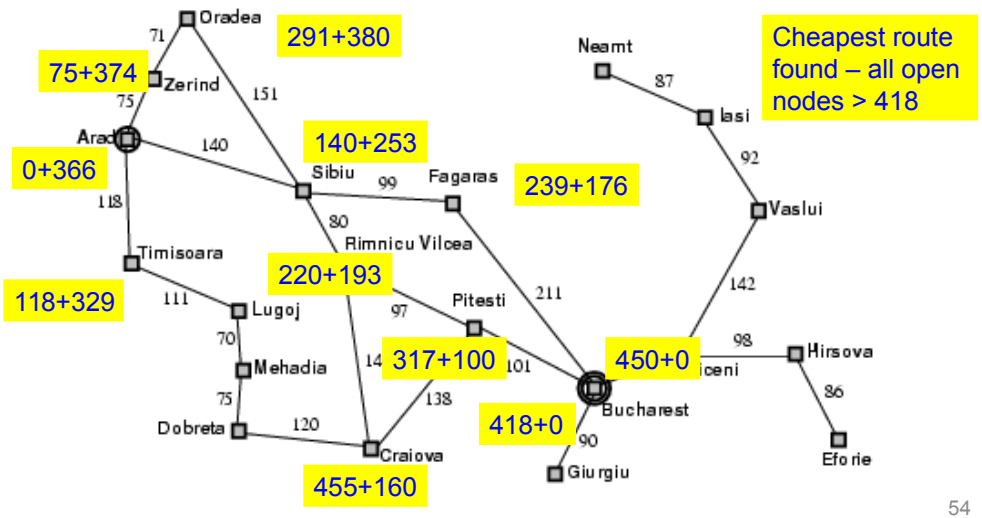
A* Search Example

Choose Sibiu | Choose Rimnicu Vilcea 413 < 415 | Choose Fagaras 415 < 417 | Choose Pitesti 417 < 450

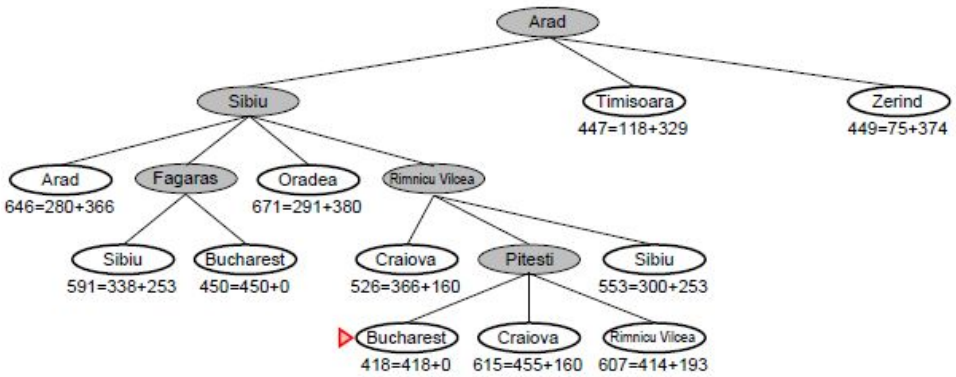


A* Search Example

Choose Sibiu | Choose Rimnicu Vilcea 413 < 415 | Choose Fagaras 415 < 417 | Choose Pitesti 417 < 450



Search Tree



General Algorithm for A* Search

```
agenda = [initial state];
while agenda not empty do
   take node from agenda such that
     f(node) = min { f(n) | n in agenda}
     where f(n) = g(n) + h(n)
   if node is goal state then
    return solution;
   new nodes = apply operations to node;
   add new nodes to the agenda;
```

Properties of A* Search

- Complete, provided
  - only finitely many nodes `n` with `f(n) < C*`
    - `C*` is the true minimal cost to reach the goal
  - an admissible heuristic is used
    - Never overestimates the true cost ('distance')
    - i.e., `h(n) < true(n)`
      where `true(n)` is the true cost from `n`
    - Also require `h(n) ≥ 0`, so `h(G) = 0` for any goal `G`
- Exponential time
- Keeps all nodes in memory
- Optimal

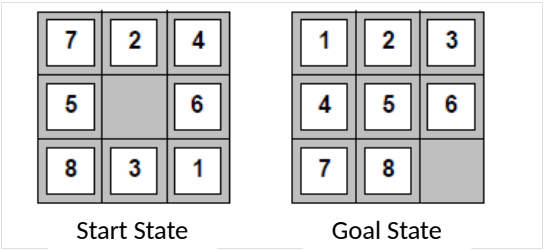
Admissible Heuristics

- Example - for the 8-puzzle:

  $h_1(n)$ = number of misplaced tiles

  $h_2(n)$ = sum of *Manhattan distances*

  - (i.e., no. of squares from desired location of each tile)



Exercise

- Calculate h_1 and h_2 for the 8 puzzle on the previous slide.
- Are these heuristics admissible?

Importance of the Heuristic Choice

- Typical search costs:
  - *d = 14*
    - IDS = 3,473,941 nodes
    - A*(h_1) = 539 nodes
    - A*(h_2) = 113 nodes
  - *d = 24*
    - IDS ≈ 54,000,000,000 nodes
    - A*(h_1) = 39,135 nodes
    - A*(h_2) = 1,641 nodes

Summary

- Heuristic functions estimate costs of shortest paths
- Good heuristics can dramatically reduce search cost
- Greedy best-first search expands lowest `h`
  - incomplete and not always optimal
- A* search expands lowest `g + h`
  - complete and optimal
  - also optimally efficient

- Next time
  - Search in complex environments (partial observation) and in game playing