HEURISTIC SEARCH STRATEGIES

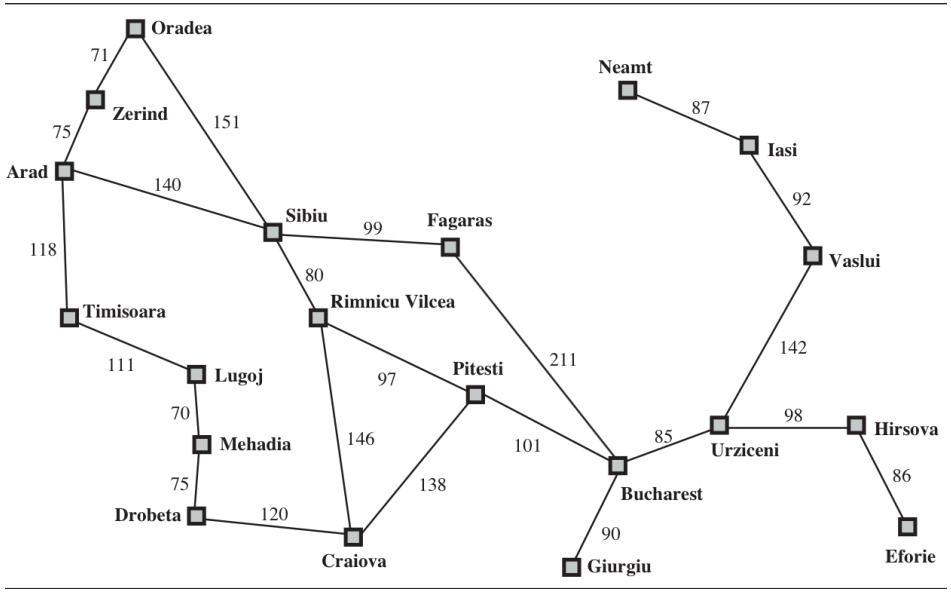
- **Informed (Heuristic) search** strategy—one that uses problem-specific knowledge beyond the definition of the problem itself—can find solutions more efficiently than can an uninformed strategy.

- Most algorithms include a **heuristic function**, denoted by h(n), where h(n) = estimated cost of the cheapest path from the state at node *n* to a goal state.

- For example, in Romania, one might estimate the cost of the cheapest path from Arad to Bucharest via the **straight-line distance function**.

Greedy best-first search

- **Greedy best-first search** tries to expand the node that is closest to the goal by using the **straight- line distance** heuristic h_{SLD}. If the goal is Bucharest, we need to know the straight-line distances to Bucharest, which are shown in Figure on RHS.



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

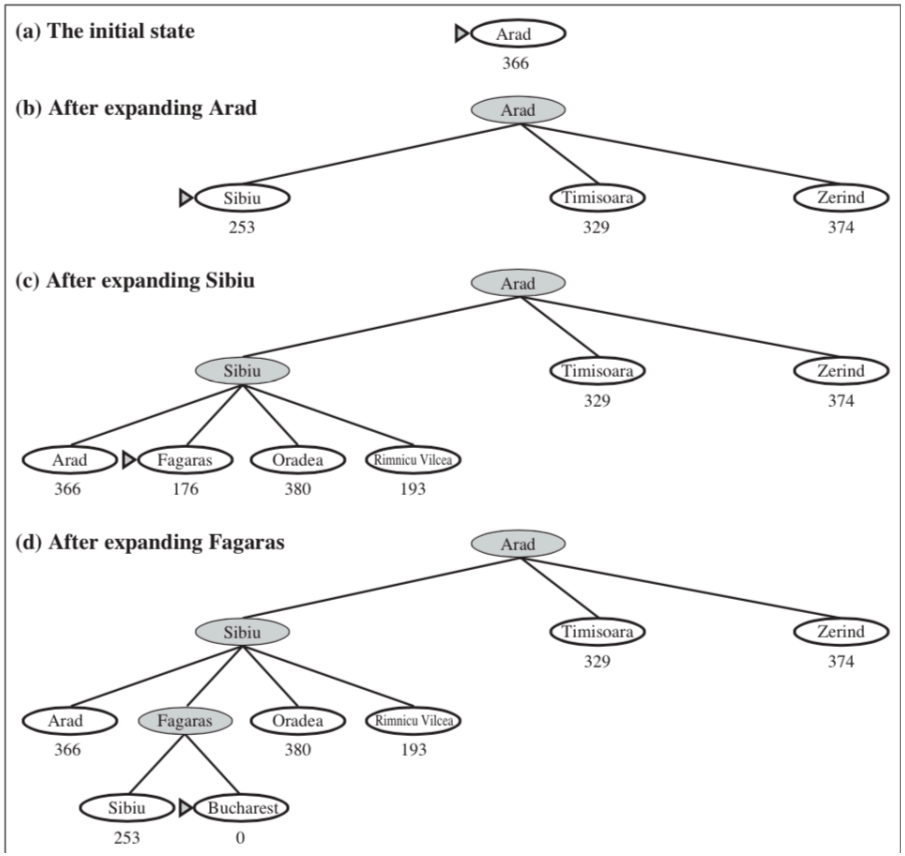
Values of h_{SLD}—straight-line distances to Bucharest.

Greedy best-first search

- Following Figure shows the progress of a greedy best-first search using h_{SLD}

- It is not optimal as the path via Sibiu and Fagaras to Bucharest is 32 kilometers longer than the path through Rimnicu Vilcea and Pitesti.

- This shows why the algorithm is called “greedy”—at each step it tries to get as close to the goal as it can.



Greedy best-first search

```
// This pseudocode is adapted from below // source: //https://courses.cs.washington.edu/
Best-First-Search(Grah g, Node start)
    1) Create an empty PriorityQueue
        PriorityQueue pq;
    2) Insert "start" in pq.
        pq.insert(start)
    3) Until PriorityQueue is empty
        u = PriorityQueue.DeleteMin
        If u is the goal
            Exit
        Else
            Foreach neighbor v of u
                If v "Unvisited"
                    Mark v "Visited"
                    pq.insert(v)
                Mark u "Examined"
End procedure
```

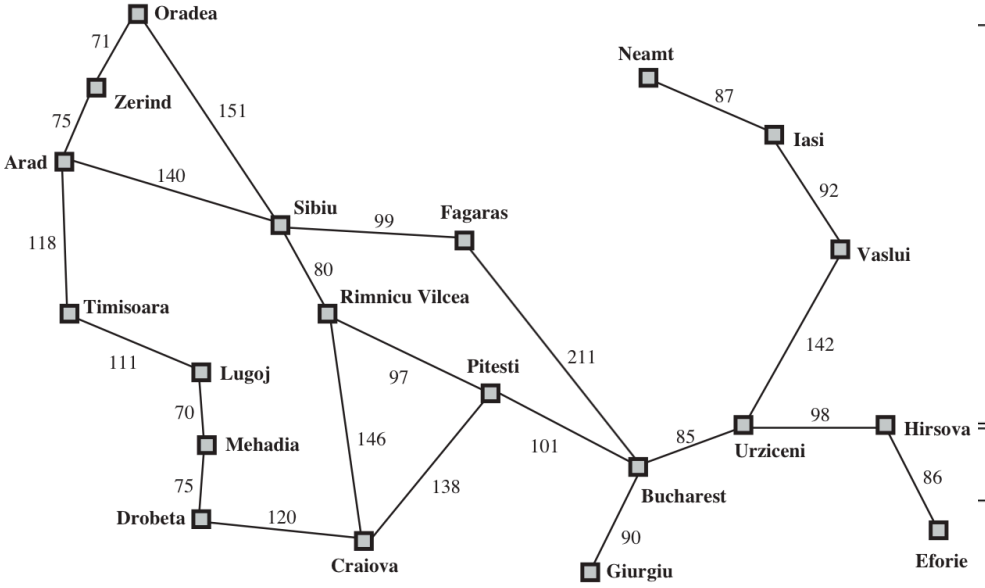
Properties of greedy search

- Complete?? No–it can get stuck in loops, e.g., Iasi → Neamt → Iasi → Neamt → … if we want to reach from Lasi to Fagaras.
  - Complete in finite space with repeated-state checking (Graph Search)
- Time?? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space?? $O(b^m)$—keeps all nodes in memory
- Optimal?? No

- With a good heuristic function, however, the complexity can be reduced substantially.

A* search

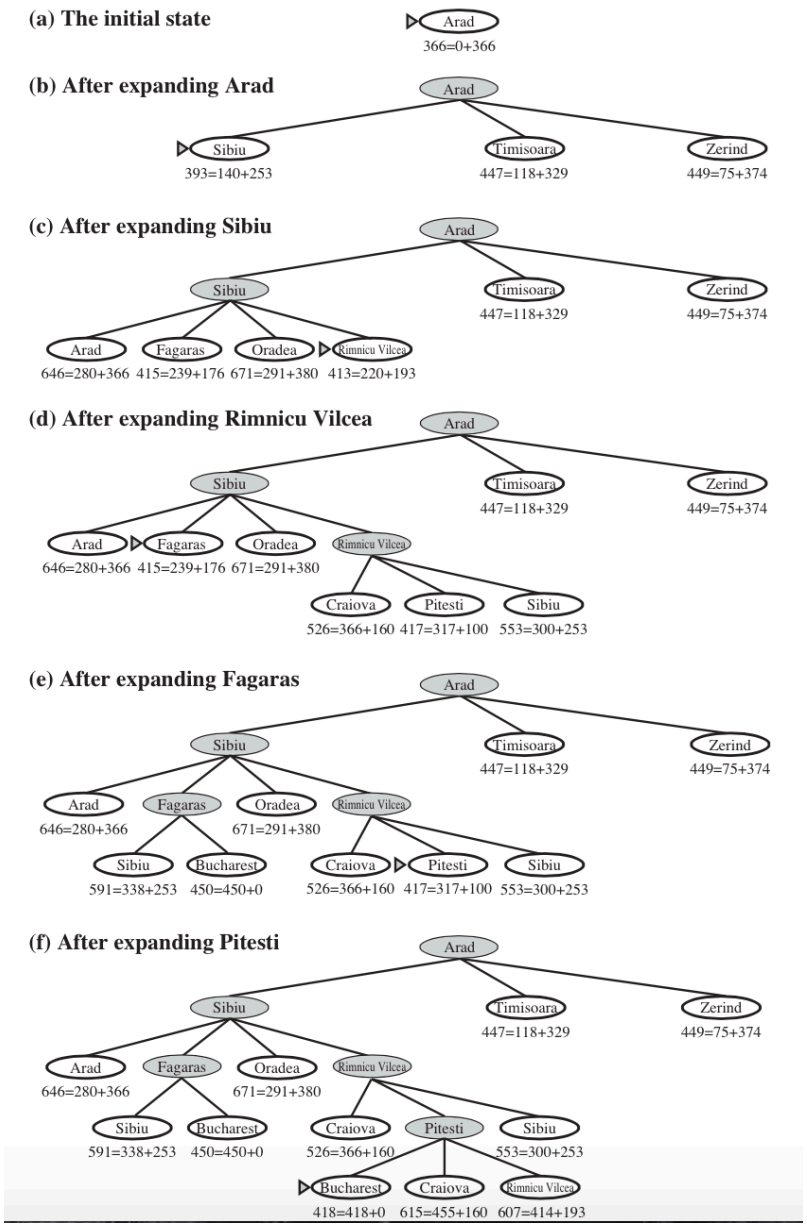
- Idea: avoid expanding paths that are already expensive.
  - Evaluation function f(n) = g(n) + h(n)
  - g(n) = cost so far to reach n
    h(n) = estimated cost to goal from n
    f(n) = estimated total cost of path through n to goal
- A* search uses an admissible heuristic:
  - $h_{SLD}(n)$ never overestimates the actual road distance.



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

Values of h_{SLD}—straight-line distances to Bucharest.

Following figure explain it's execution using the information given in previous two figures.



Pseudo code for A* search

```
1  Put node_start in the OPEN list with f(node_start) = h(node_start) (initialization)
2  while the OPEN list is not empty {
3    Take from the open list the node node_current with the lowest
4         f(node_current) = g(node_current) + h(node_current)
5    if node_current is node_goal we have found the solution; break
6    Generate each state node_successor that come after node_current
7    for each node_successor of node_current {
8      Set successor_current_cost = g(node_current) + w(node_current, node_successor)
9      if node_successor is in the OPEN list {
10       if g(node_successor) ≤ successor_current_cost continue (to line 20)
11     } else if node_successor is in the CLOSED list {
12       if g(node_successor) ≤ successor_current_cost continue (to line 20)
13       Move node_successor from the CLOSED list to the OPEN list
14     } else {
15       Add node_successor to the OPEN list
16       Set h(node_successor) to be the heuristic distance to node_goal
17     }
18     Set g(node_successor) = successor_current_cost
19     Set the parent of node_successor to node_current
20   }
21   Add node_current to the CLOSED list
22 }
23 if(node_current != node_goal) exit with error (the OPEN list is empty)
```

Properties of A*

- **Complete**?? Yes, unless there are infinitely many nodes.

- **Time**?? $O(b^{\varepsilon m})$—where $\varepsilon = (h^* - h)/h^*$ is the relative error in h. where h^* is the optimal heuristic, the exact cost to get from *x* to the goal.
  - If h = 0, then ε = 1 and we get uniform-cost search
  - If h = h^*, then it is perfect and we find the solution immediately

- **Space**?? $O(b^m)$—it keeps all nodes in memory

- **Optimal**?? Yes
  - A^* expands all nodes with $f(n) < C^*$
  - A^* expands some nodes with $f(n) = C^*$
  - A^* expands no nodes with $f(n) > C^*$

Issues in A* search

- Exponential complexity of A* with respect to depth of goal often makes it impractical in finding an optimal solution.

- Computation time is not the main drawback of A*. As A* keeps all generated nodes in memory (as do all GRAPH-SEARCH algorithms), and usually runs out of space long before it runs out of time.

- For this reason, A* is not practical for many large-scale problems.

Read it Yourself

- 3.5.3 Memory-bounded heuristic search
- 3.5.4 Learning to search better
- 3.6 Heuristic functions

Lab Work No.3

- Design a program for the greedy best first search and A* search
  - Implement the greedy best first search using the following link https://www.geeksforgeeks.org/best-first-search-informed-search/
  - Implement the A* search using the following link https://www.geeksforgeeks.org/a-search-algorithm/