A Star

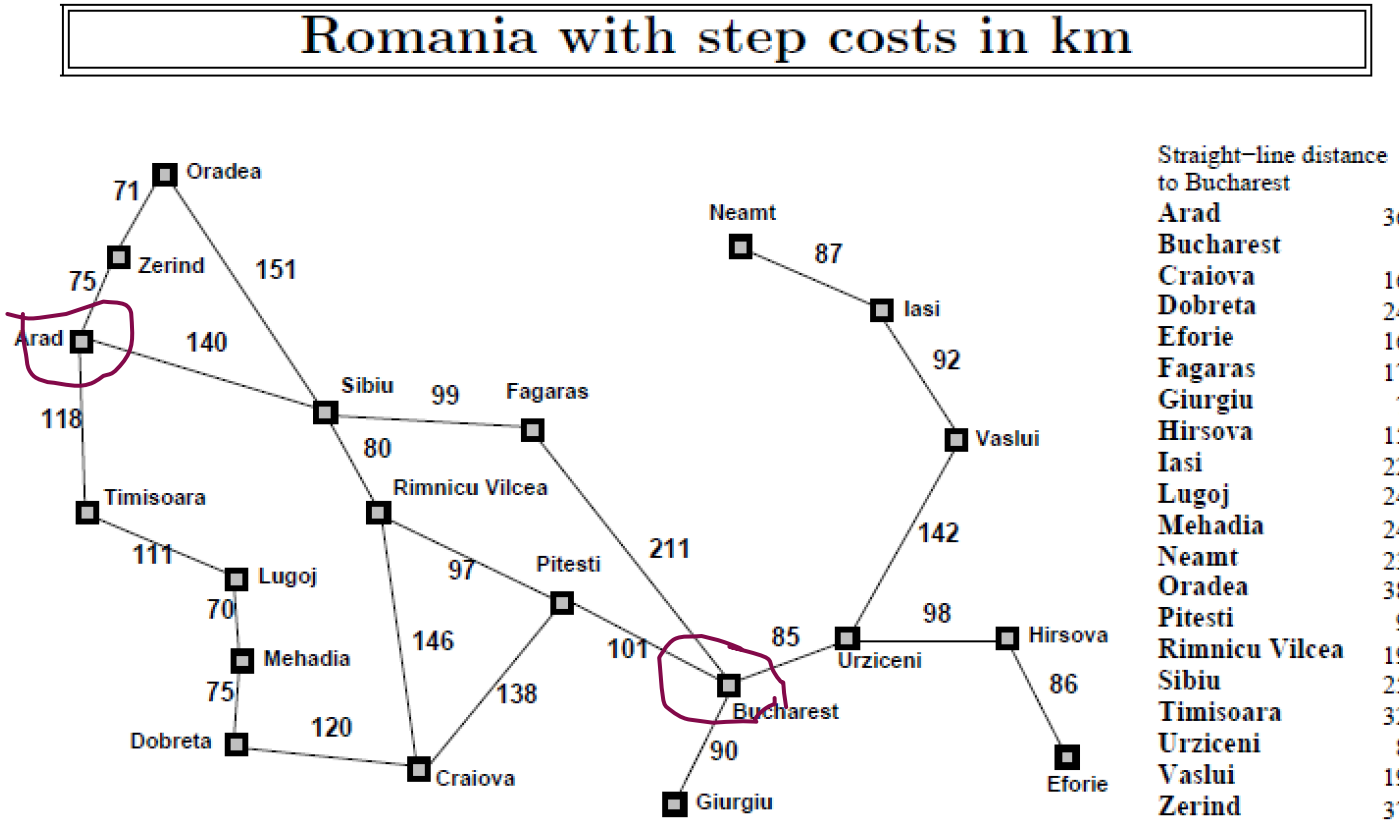
YAO ZHAO

Collections

- Set
- Priority Queue

A*

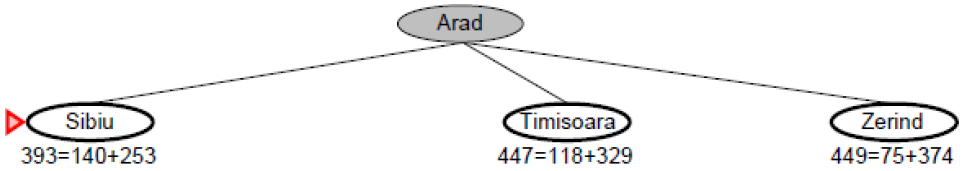
- Set:{}
- Priority Queue:{Arad:366}

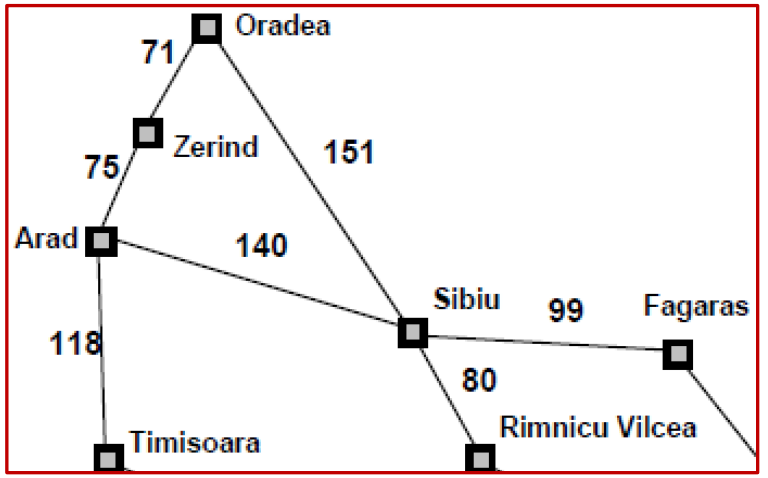


A*

- Set:{Arad}
- Priority Queue: {Sibiu:393,Timisoara:447,Zerind :449}

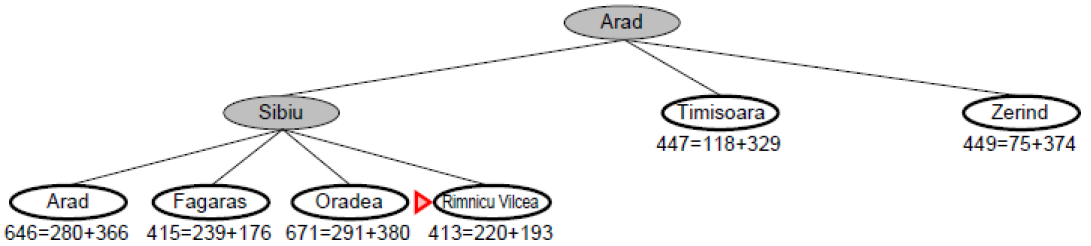
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



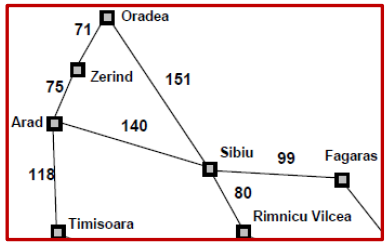


A*

- Set:{Arad,Sibiu}
- Priority Queue: {RimnicuVilcea:413,Timisoara:447,Zerind:449,Fagaras:415,Oradea:671}

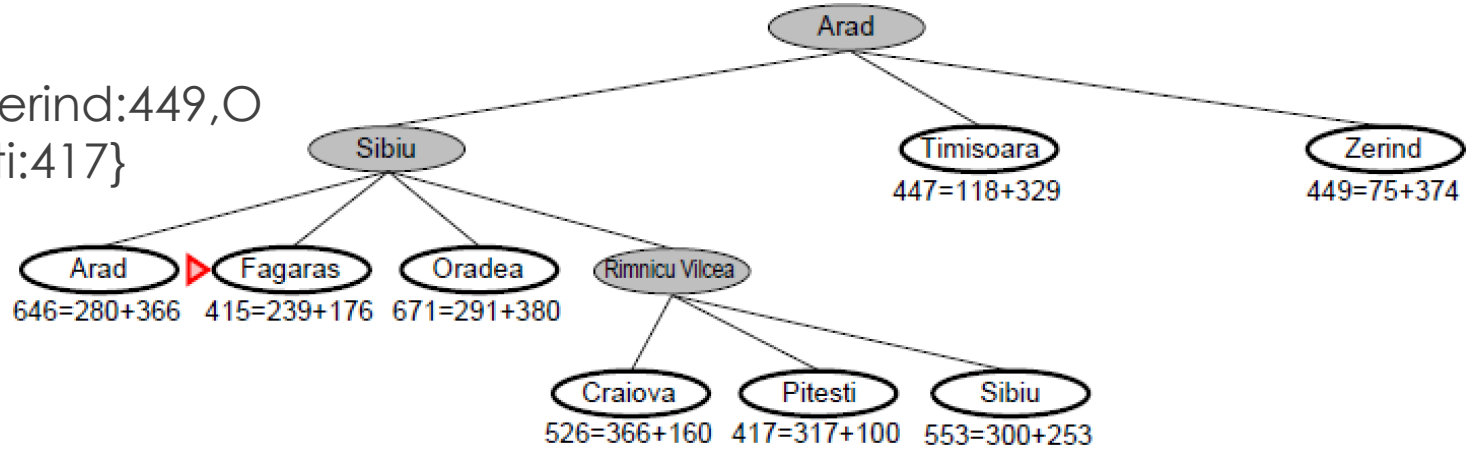


Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193



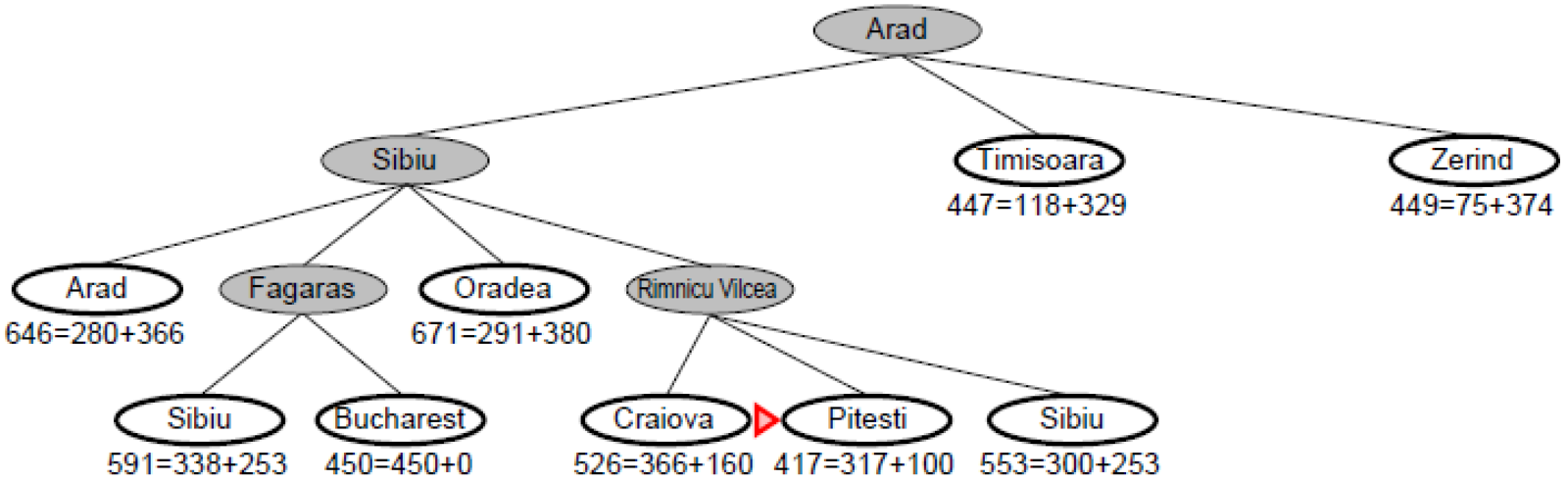
A*

- Set:{Arad,Sibiu,Rimnicu}
- Priority Queue: {Fagaras:415,Timisoara:447,Zerind:449,Oradea:671,Craiova:526,Pitesti:417}



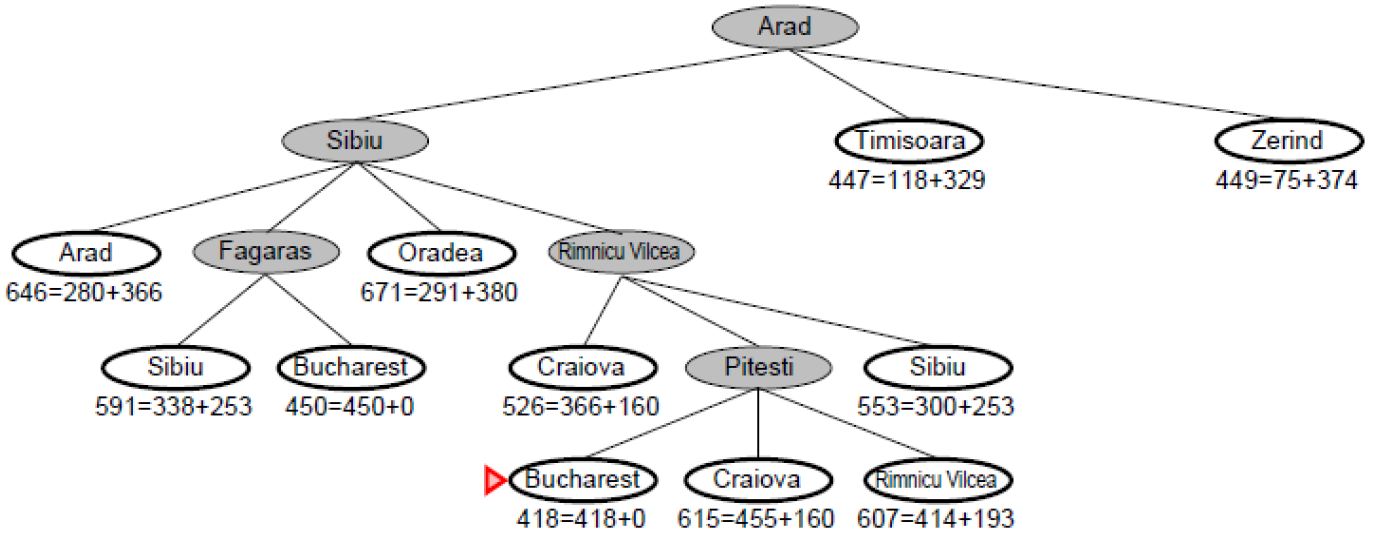
A*

- Set:{Arad,Sibiu,Rimnicu, Fagaras}
- Priority Queue: {Pitesti:417,Timisoara:447,Zerind:449, Oradea:671,Craiova:526,Pitesti:417,Bucharest:450,Craiova:526}



A*

- Set:{Arad,Sibiu,Rimnicu, Fagaras, Pitesti}
- Priority Queue: {Bucharest:418,Timisoara:447,Zerind:449, Oradea:671,Craiova:526,Pitesti:417,~~Bucharest:450~~,Craiova:526}



Note: Cities and distances in PPT and test case are not the same.

UCS vs Greedy Best First vs A*

- UCS：f(n) = g(n)
- Greedy Best First: f(n) = h(n)
- A*: f(n)=g(n)+h(n)

```
275 def uniform_cost_search(problem):
276     """[Figure 3.14]"""
277     return best_first_graph_search(problem, lambda node: node.path_cost)
```

```
391 greedy_best_first_graph_search = best_first_graph_search
392 # Greedy best-first search is accomplished by specifying f(n) = h(n).
```

```
395 def astar_search(problem, h=None):
396     """A* search is best-first graph search with f(n) = g(n)+h(n).
397     You need to specify the h function when you call astar_search, or
398     else in your Problem subclass."""
399     h = memoize(h or problem.h, 'h')
400     return best_first_graph_search(problem, lambda n: n.path_cost + h(n))
```

```
244 def best_first_graph_search(problem, f):
245     """Search the nodes with the lowest f scores first.
246     You specify the function f(node) that you want to minimize; for example,
247     if f is a heuristic estimate to the goal, then we have greedy best
248     first search; if f is node.depth then we have breadth-first search.
249     There is a subtlety: the line "f = memoize(f, 'f')" means that the f
250     values will be cached on the nodes as they are computed. So after doing
251     a best first search you can examine the f values of the path returned."""
252     f = memoize(f, 'f')
253     node = Node(problem.initial)
254     if problem.goal_test(node.state):
255         return node
256     frontier = PriorityQueue(min, f)
257     frontier.append(node)
258     explored = set()
259     while frontier:
260         node = frontier.pop()
261         if problem.goal_test(node.state):
262             return node
263         explored.add(node.state)
264         for child in node.expand(problem):
265             if child.state not in explored and child not in frontier:
266                 frontier.append(child)
267             elif child in frontier:
268                 incumbent = frontier[child]
269                 if f(child) < f(incumbent):
270                     del frontier[incumbent]
271                     frontier.append(child)
272     return None
```