EECS 391
Intro to AI

Uniform Cost Search,
Informed Search

L4: Tue Sep 12, 2017

Outline

- characterizing search algorithms
- uniform-cost search: search with path costs
- informed search
- greedy search
- heuristic functions

Generic search function

function Tree-Search(*problem, strategy*) **returns** solution or failure

initialize tree using initial state of *problem*

loop

if no candidates for expansion **return** failure

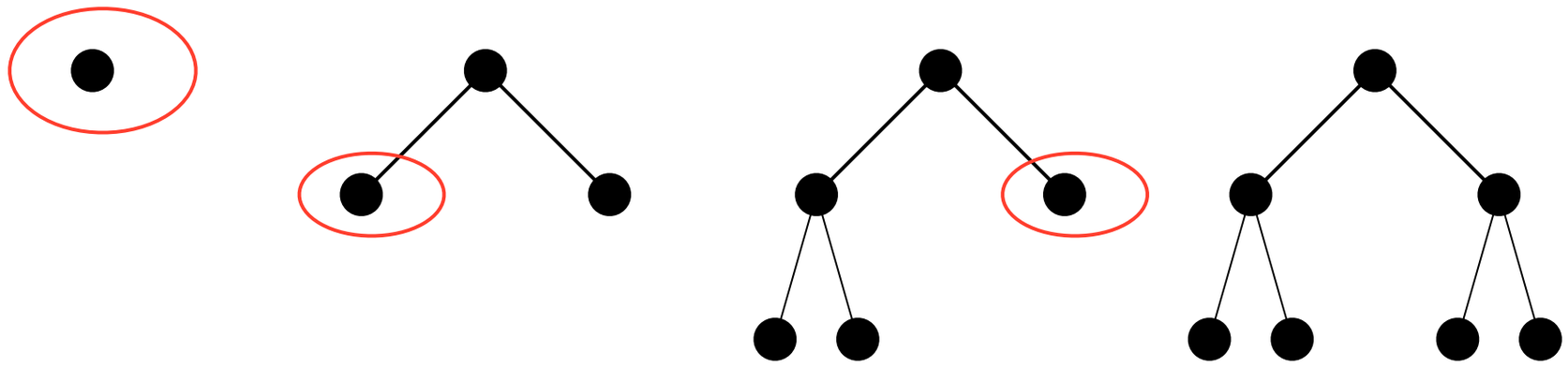
choose leaf node for expansion using *strategy*

if node contains goal state **return** solution

else expand node and add resulting nodes to search tree

Recall: breadth-first search

function BREADTH-FIRST-SEARCH (*problem*) **returns** a solution or failure
  return GENERAL-SEARCH (*problem*, ENQUEUE-AT-END)



- Type type of search is determined by how the *fringe* is expanded
- breadth-first search is implemented using a first-in first-out (FIFO) *queue*
  - all new nodes go at end of queue
  - shallow nodes are expanded before deeper nodes

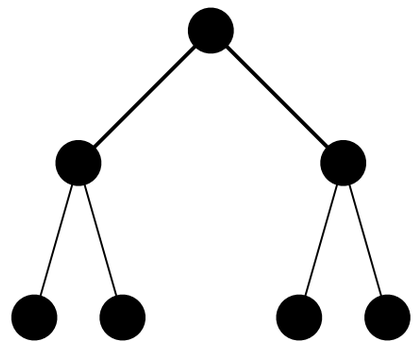
Evaluating different search strategies

- **Completeness**:
  Is the algorithm guaranteed to find the optimal solution? (if one exists)
- **Optimality**:
  Does the algorithm find the solution with lowest path cost?
- **Time complexity**:
  How long does it take to find the solution? (In big-O terms)
- **Space complexity**:
  How much memory is need to perform the search?

Characterizing breadth-first search

- Complete?
- Time?
- Space?
- Optimal?

- Measuring time and space complexity
  - *b* - maximum branching factor of search tree
  - *d* - depth of least-cost solution
  - *m* - maximum depth of state space (can be infinite)



- Useful fact:

$$\sum_{k=0}^{n-1} ar^k = a\frac{1-r^n}{1-r}$$

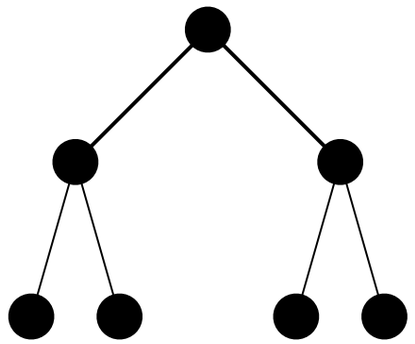
(e.g. for a=1, r=b=2, n-1=d)

$$\sum_{k=0}^{d} 2^k = \frac{1-2^{d+1}}{1-2} = O(2^{d+1})$$

Characterizing breadth-first search

- Complete? Yes
- Time? $O(b^{d+1})$
- Space? $O(b^{d+1})$
- Optimal? yes (if unit cost per step)

- Measuring time and space complexity
  - *b* - maximum branching factor of search tree
  - *d* - depth of least-cost solution
  - *m* - maximum depth of state space (can be infinite)



- Useful fact:

$$\sum_{k=0}^{n-1} ar^k = a\frac{1-r^n}{1-r}$$

(e.g. for a=1, r=b=2, n-1=d)

$$\sum_{k=0}^{d} 2^k = \frac{1-2^{d+1}}{1-2} = O(2^{d+1})$$

The problem with breadth-first search

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

Time and memory requirements for breadth-first search. The figures shown assume branching factor $b = 10$; 1000 nodes/second; 100 bytes/node.

Properties of depth-first search

- Complete? No. What if first first branch is infinite,
  i.e. it doesn't contain a solution?

  What about loops?

- Time? $O(b^m)$

- Space? O(bm) --- linear. Only need to store single path from root to leaf.
  A node can be removed as soon as all its descendants are explored.

- Optimal? No. Could find sub-optimal solution first.

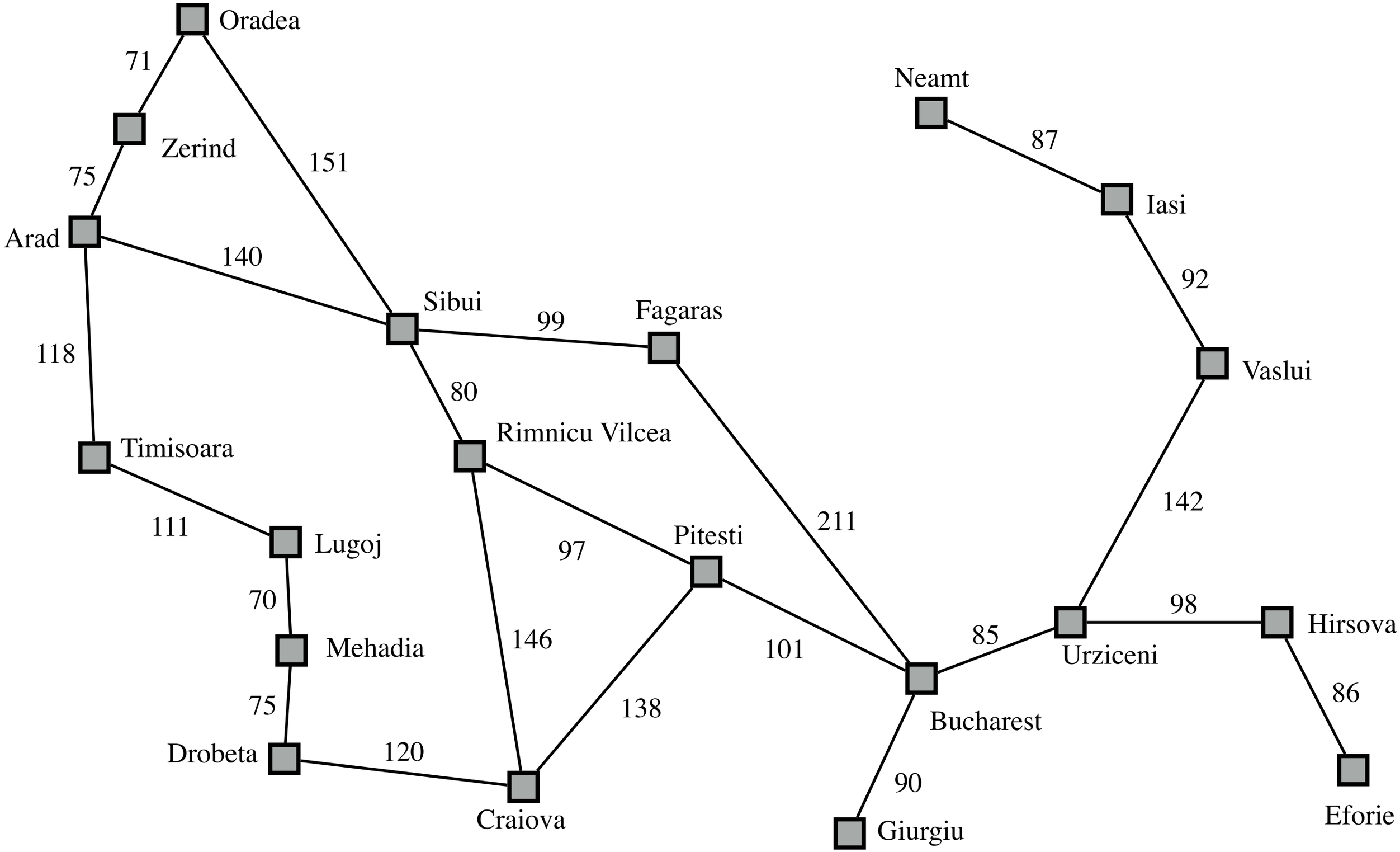
Uninformed search strategies

- Uninformed search strategies can only distinguish goal states from non-goal states, i.e. you can't tell when you're getting closing

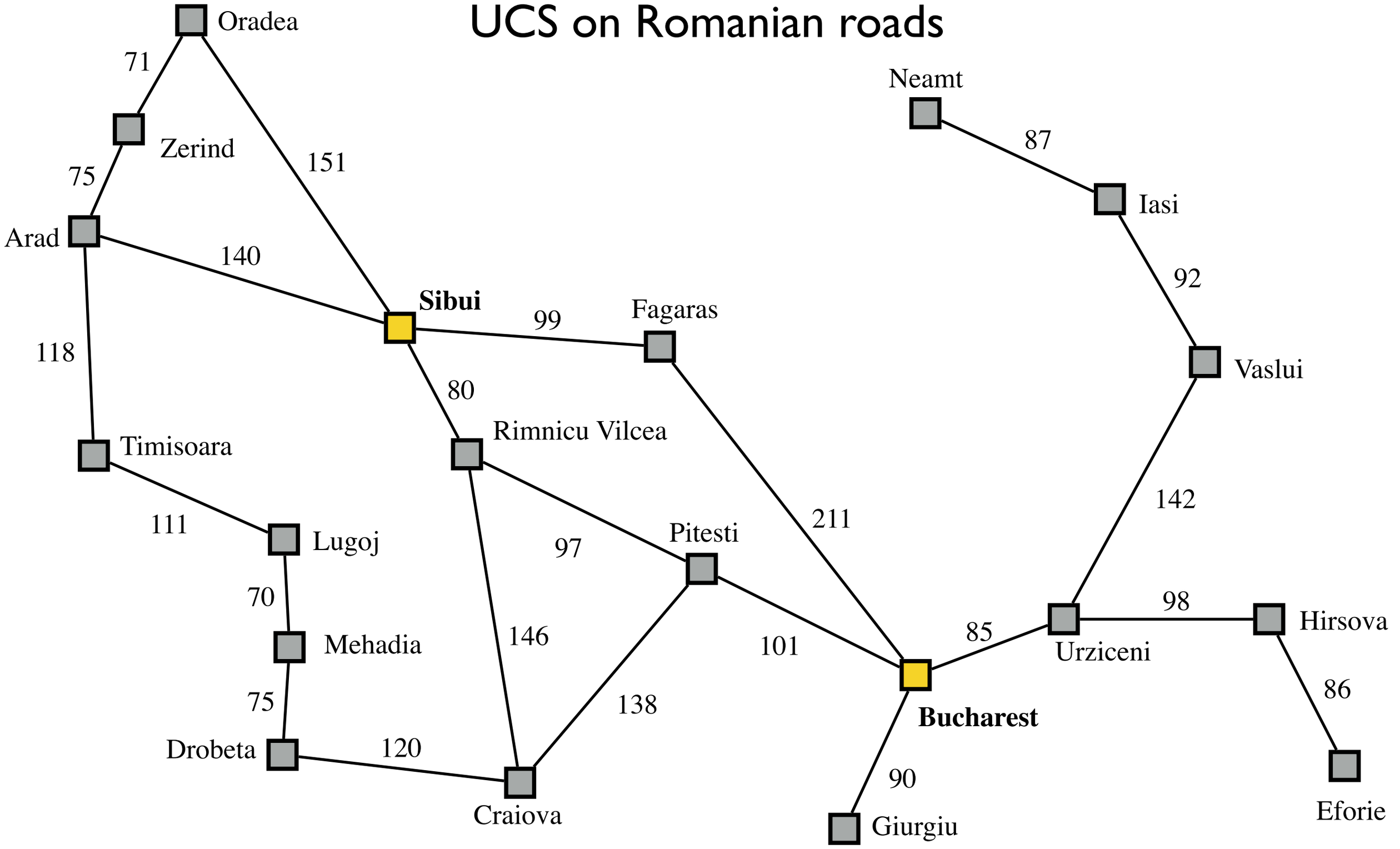
- Types of uninformed search strategies:
  - Breadth-first search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening search
  - Bi-directional search
- New one today: **Uniform-cost search**
  - variant of breadth-first
  - but takes into account *path costs*
  - key idea: *always expand node with lowest path cost*

Uniform cost search

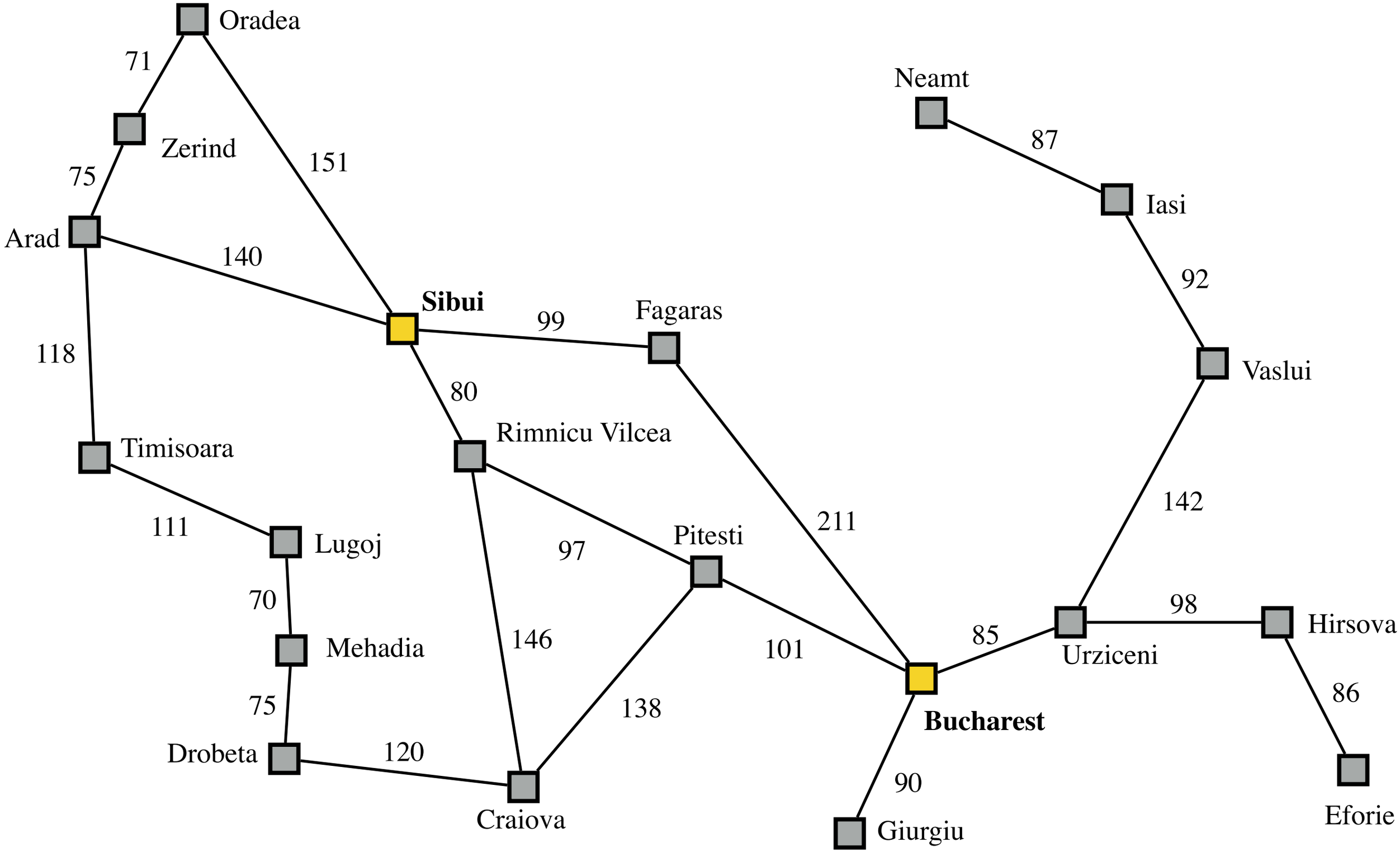
- Expand node with *lowest* ***path cost***.
- How do you implement this?
  - each node ***n*** keeps track of its **path cost** to root ***g(n)***
  - use a ***priority queue*** to (partially) order expansion nodes by path cost
  - select node with lowest path cost for expansion from priority queue
- When do you check for the goal state?
  - need to check when a node is selected for expansion (not when found)
  - Why? In BFS you could end search when first encountered.
  - In UCS, you are extend ***paths***: node ***n*** might not be on shortest path
- What if you find the same node twice?
  - This represents multiple paths
  - Choose smallest path
- Remember: *Uniform cost search expands nodes in order of lowest path cost*


Oradea
71
Neamt
87
Zerind
151
Iasi
75
Arad
140
92
Sibui
99
Fagaras
Vaslui
118
80
Rimnicu Vilcea
Timisoara
142
111
211
Lugoj
97
Pitesti
70
98
Hirsova
146
85
Mehadia
101
Urziceni
75
138
86
Bucharest
Drobeta
120
90
Craiova
Giurgiu
Eforie

UCS on Romanian roads



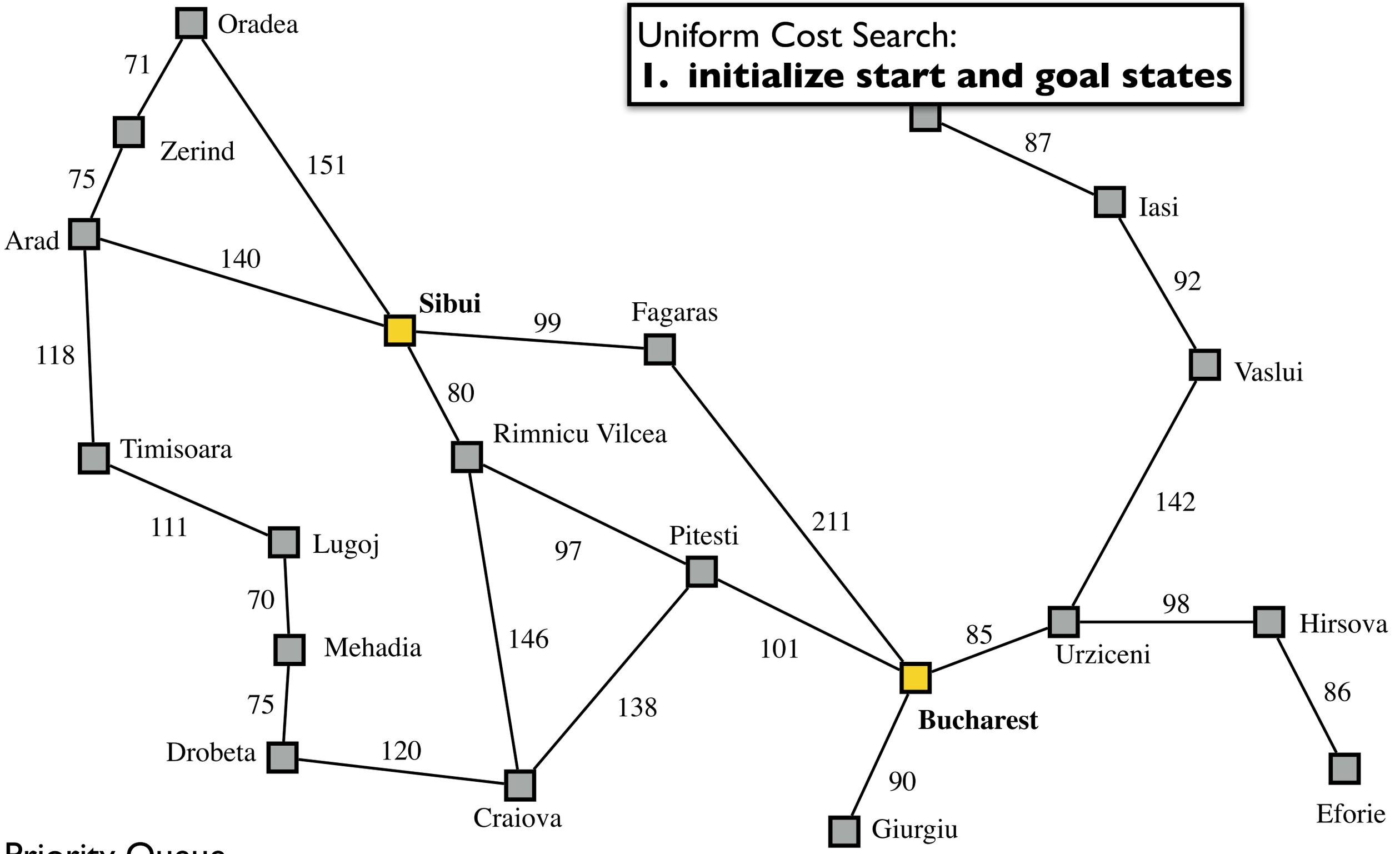
What's the best route from Sibui to Bucharest?



In absence of knowledge, node expansion order is arbitrary.
We will assume neighbors are stored alphabetically.

Uniform Cost Search:

1. initialize start and goal states

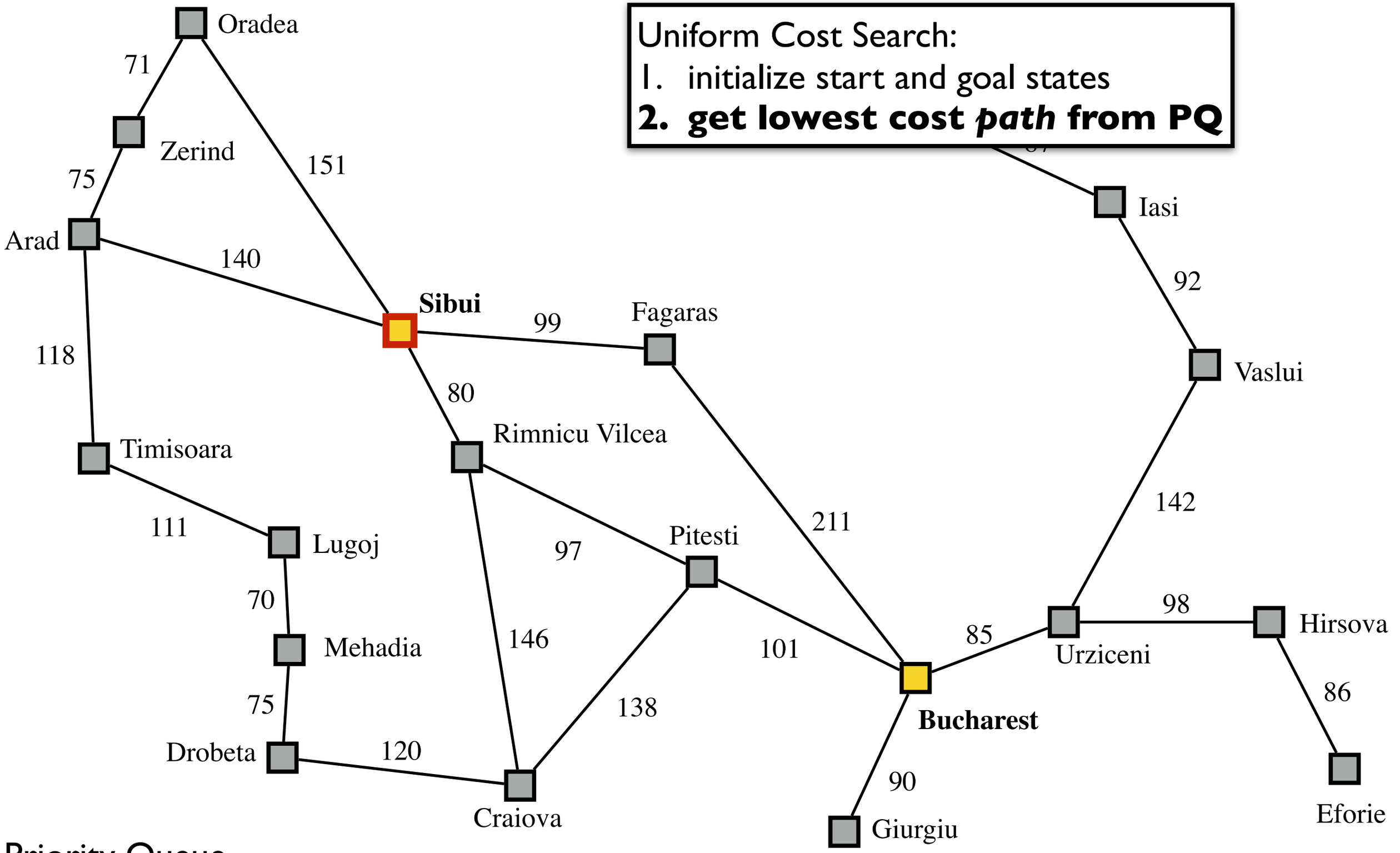


Priority Queue

node	S
parent	-
cost	0

Uniform Cost Search:

1. initialize start and goal states
2. **get lowest cost *path* from PQ**



Priority Queue

node	S
parent	-
cost	0

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. **add neighbors to PQ**

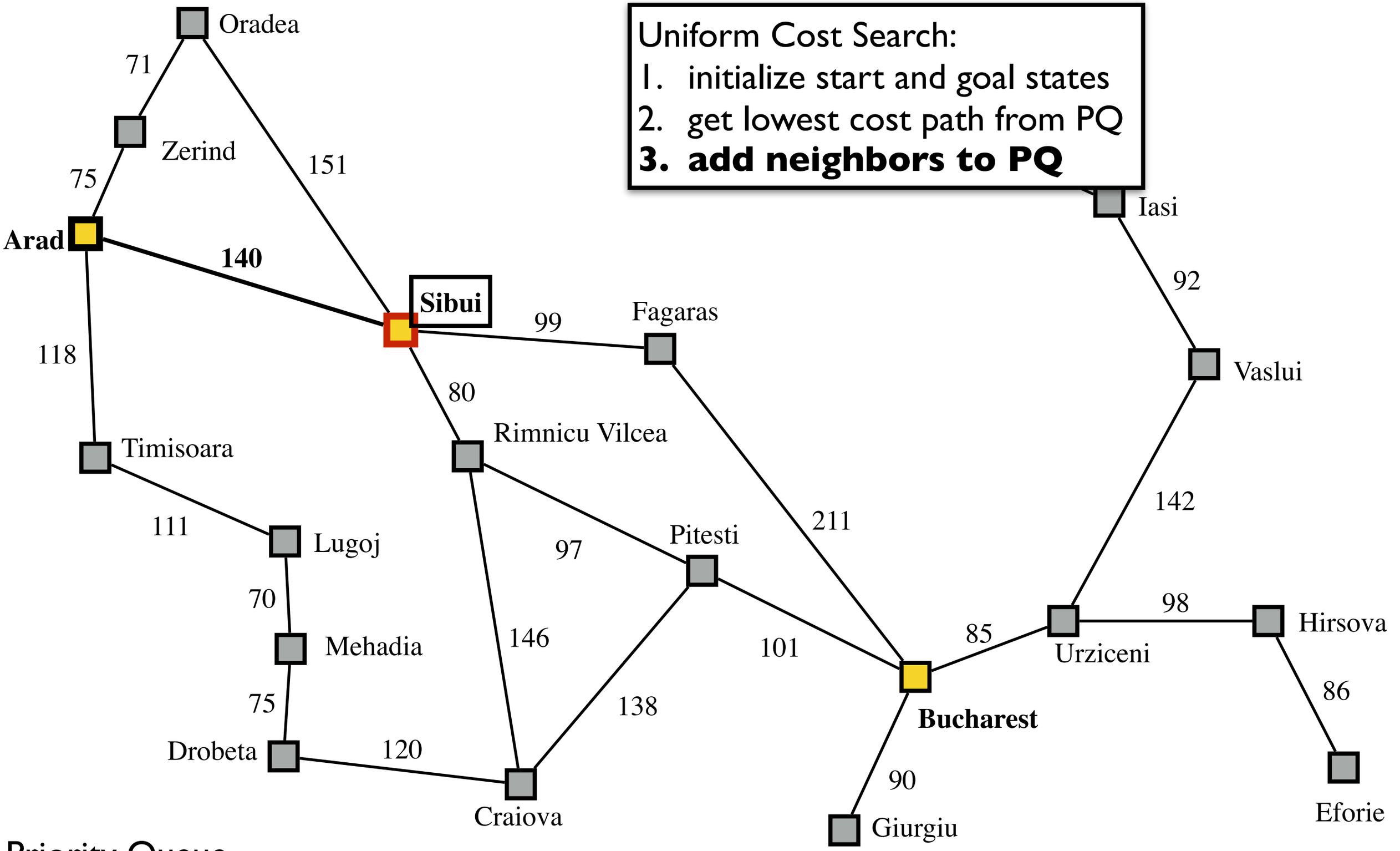
Oradea
71
Zerind
151
75
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
Iasi
92
Vaslui
142
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Eforie
Craiova
Giurgiu

Priority Queue

node
parent
cost

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. **add neighbors to PQ**



Priority Queue

node	A
parent	S
cost	140

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. **add neighbors to PQ**

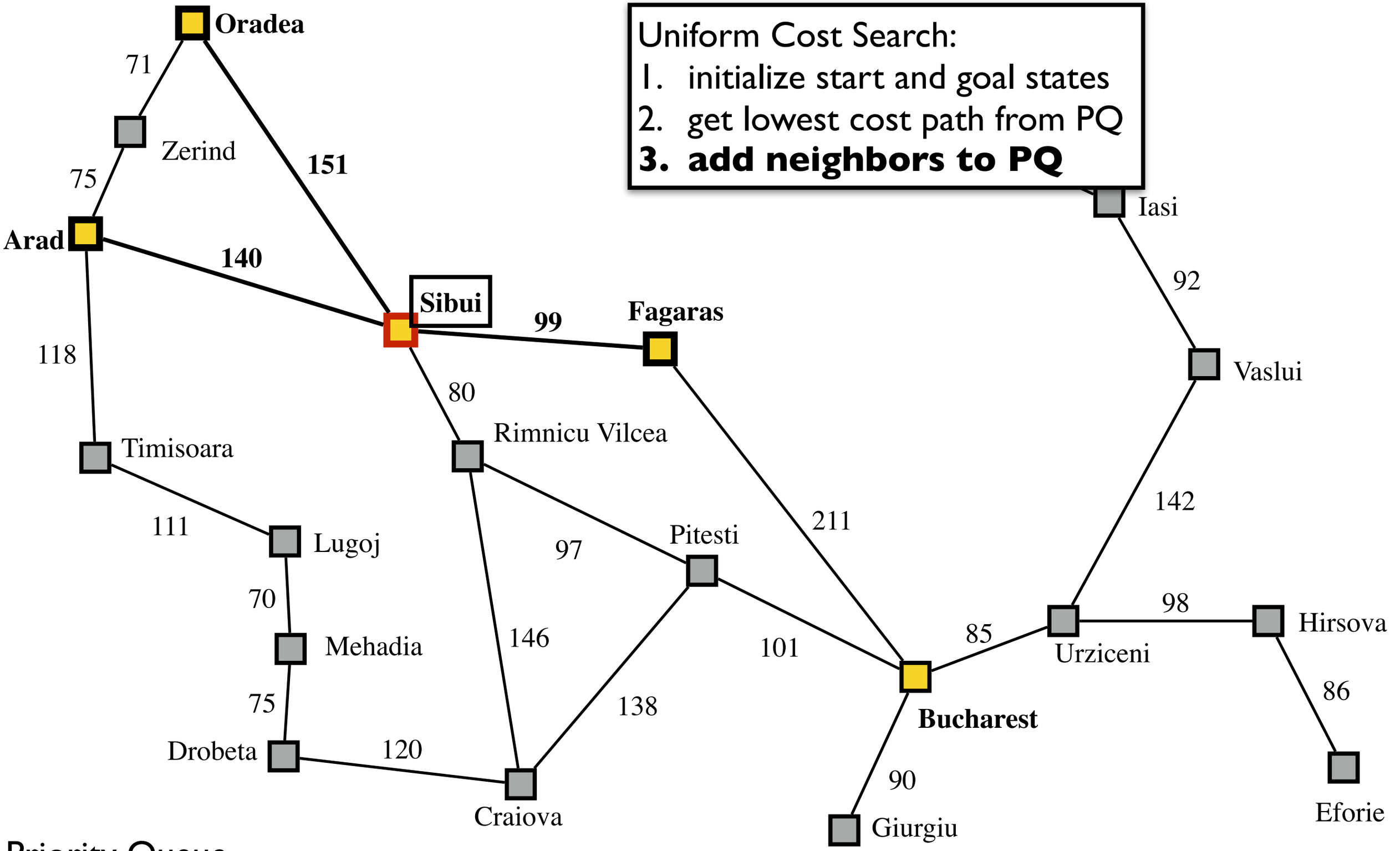
Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
Iasi
92
Vaslui
142
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Eforie
Craiova
Giurgiu

Priority Queue

node	F	A
parent	S	S
cost	99	140

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. **add neighbors to PQ**

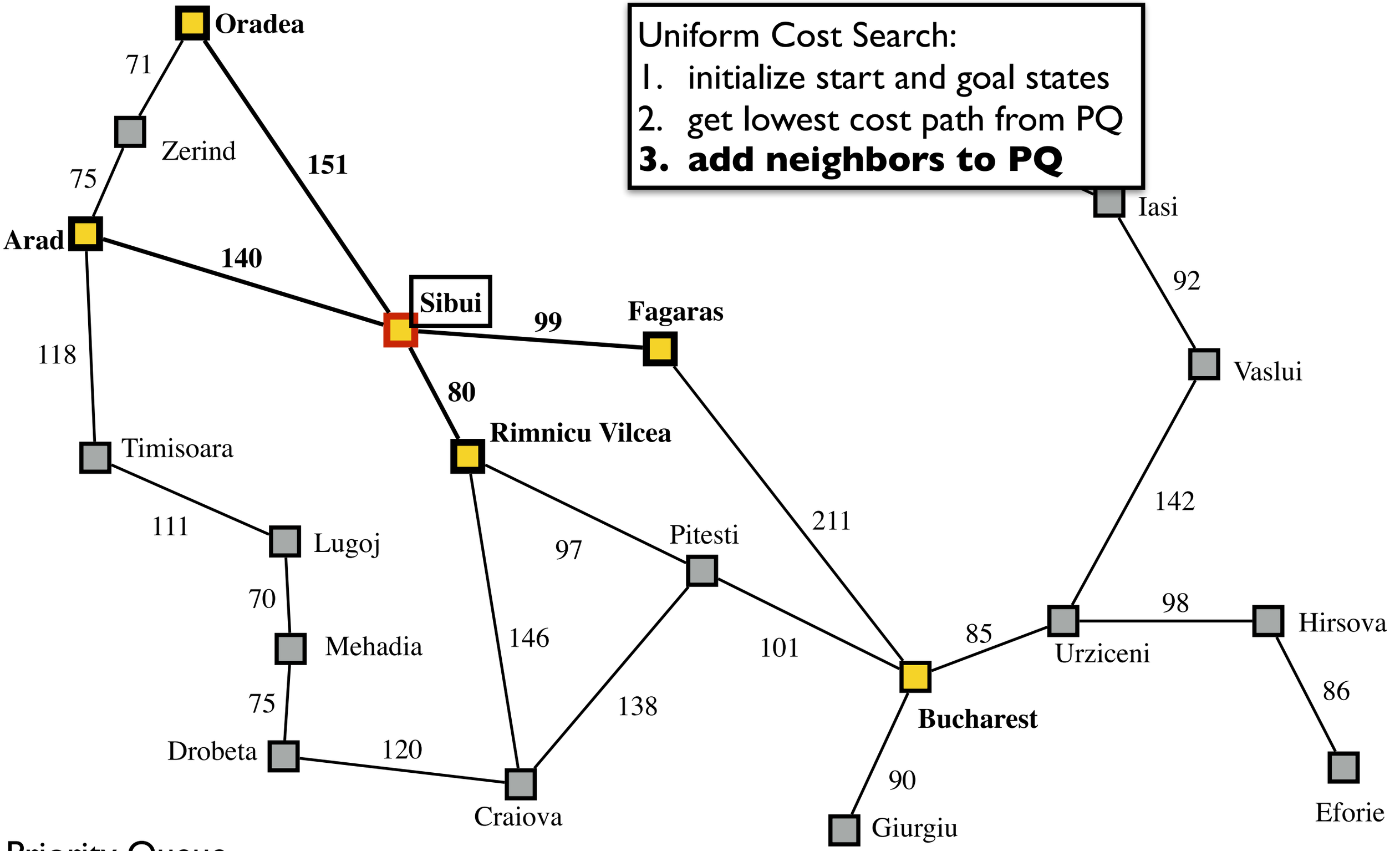


Priority Queue

node	F	A	O
parent	S	S	S
cost	99	140	151

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. **add neighbors to PQ**

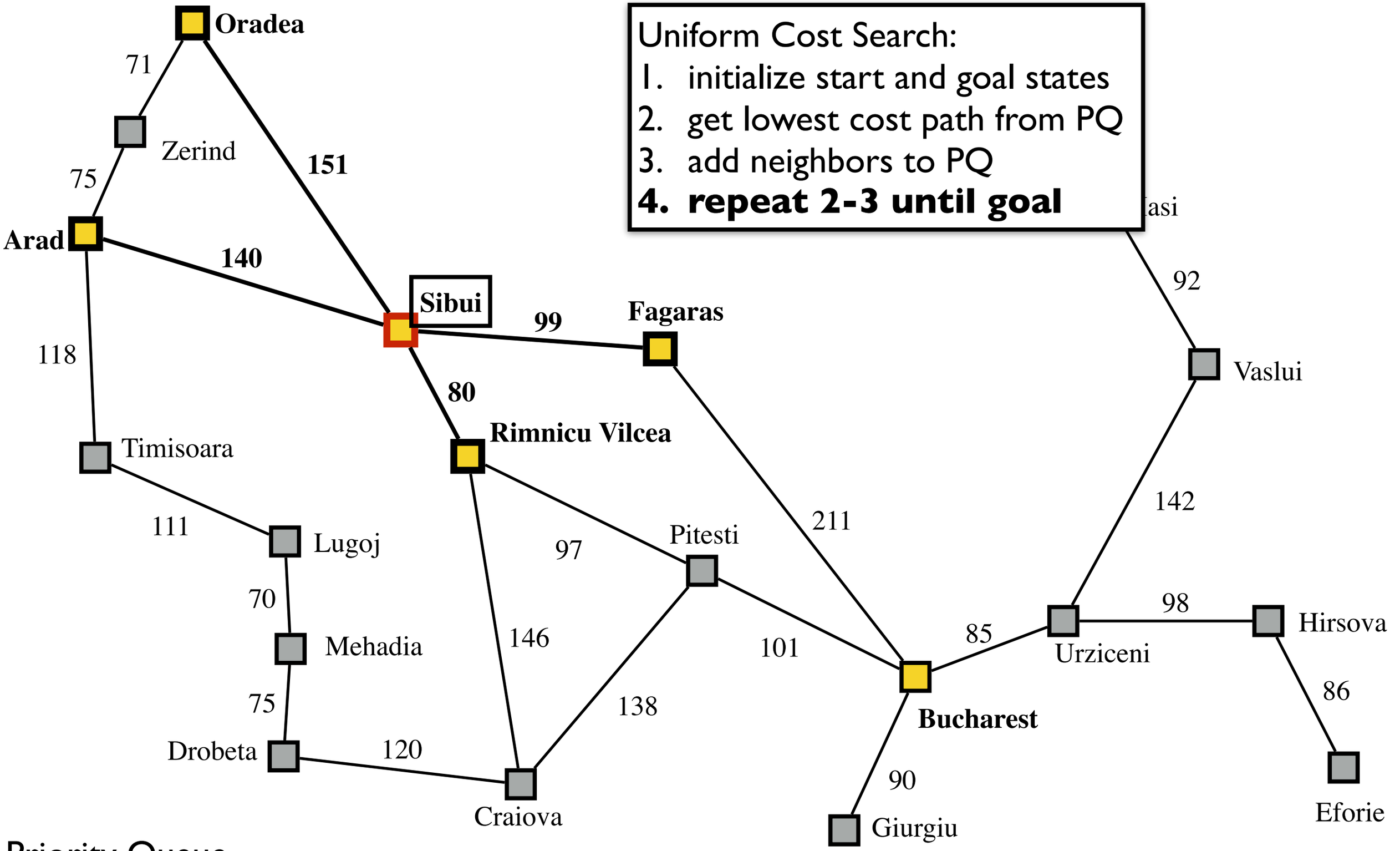


Priority Queue

node	RV	F	A	O
parent	S	S	S	S
cost	80	99	140	151

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. add neighbors to PQ
4. **repeat 2-3 until goal**

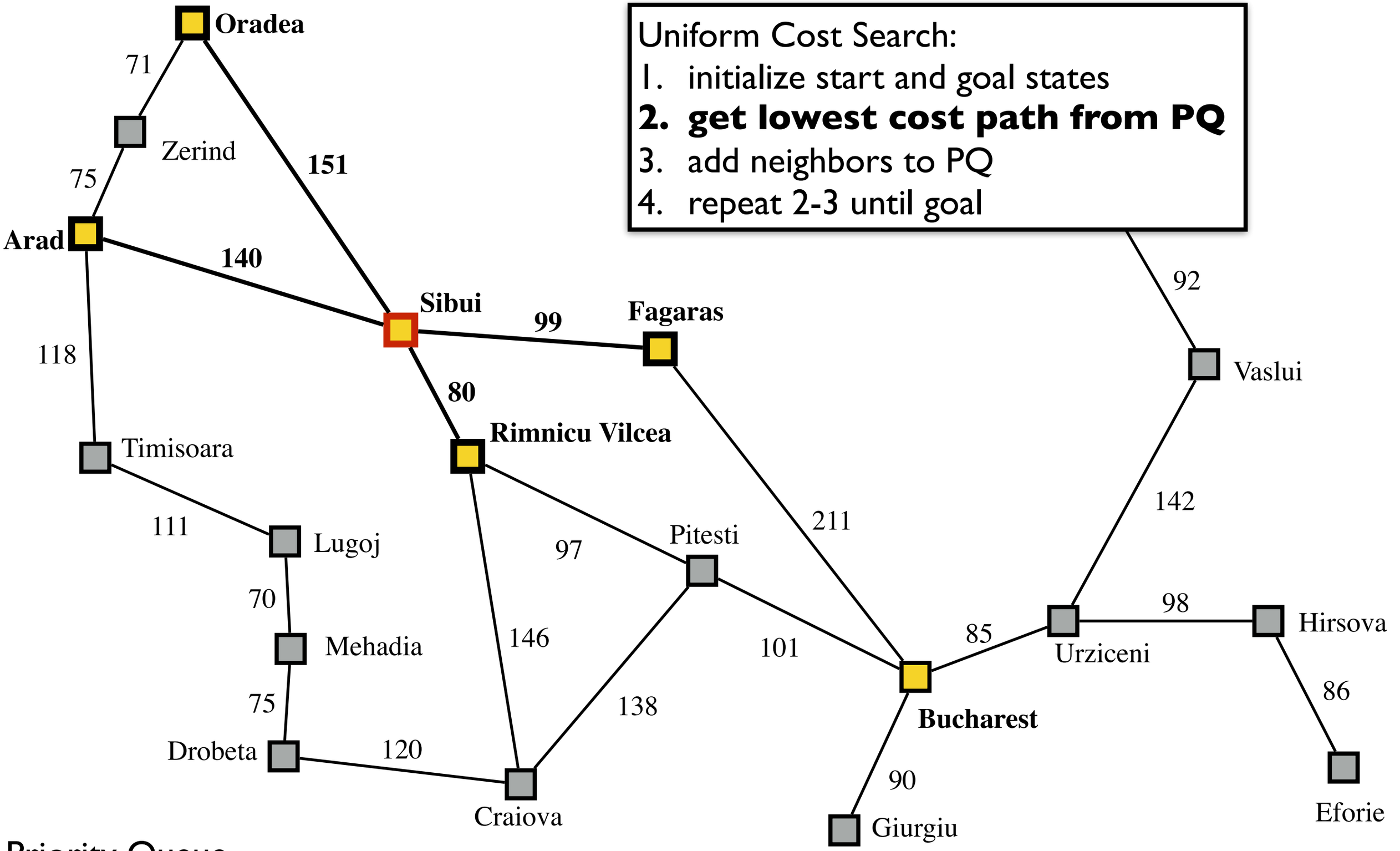


Priority Queue

node	RV	F	A	O
parent	S	S	S	S
cost	80	99	140	151

Uniform Cost Search:

1. initialize start and goal states
2. **get lowest cost path from PQ**
3. add neighbors to PQ
4. repeat 2-3 until goal



Priority Queue

node	RV	F	A	O
parent	S	S	S	S
cost	80	99	140	151

Uniform Cost Search:

1. initialize start and goal states
2. **get lowest cost path from PQ**
3. add neighbors to PQ
4. repeat 2-3 until goal

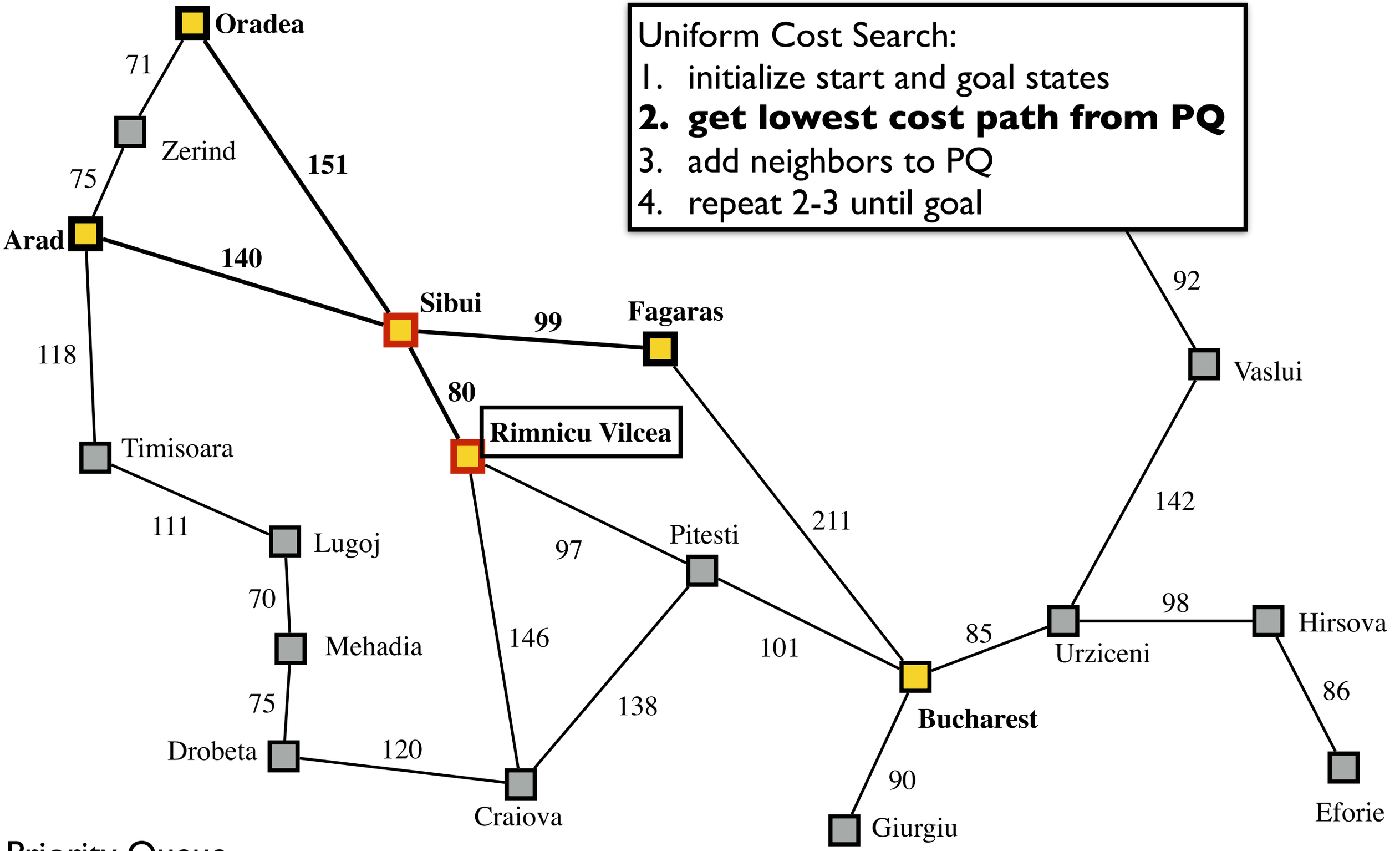
Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
92
Vaslui
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
142
70
146
101
85
98
Hirsova
Mehadia
Urziceni
75
138
86
Bucharest
Drobeta
120
90
Craiova
Giurgiu
Eforie

Priority Queue

node	RV	F	A	O
parent	S	S	S	S
cost	80	99	140	151

Uniform Cost Search:

1. initialize start and goal states
2. **get lowest cost path from PQ**
3. add neighbors to PQ
4. repeat 2-3 until goal

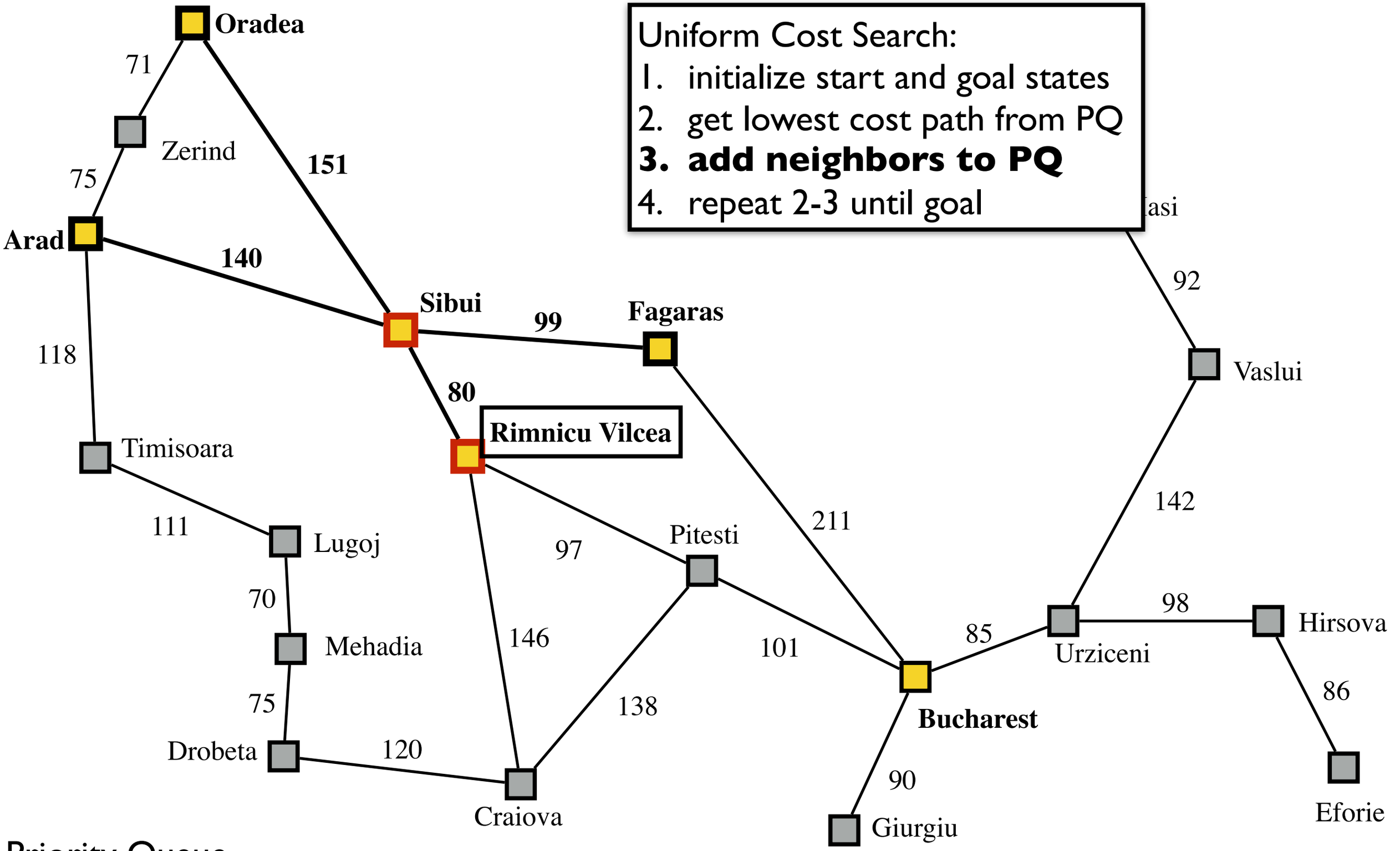


Priority Queue

node	F	A	O
parent	S	S	S
cost	99	140	151

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. **add neighbors to PQ**
4. repeat 2-3 until goal

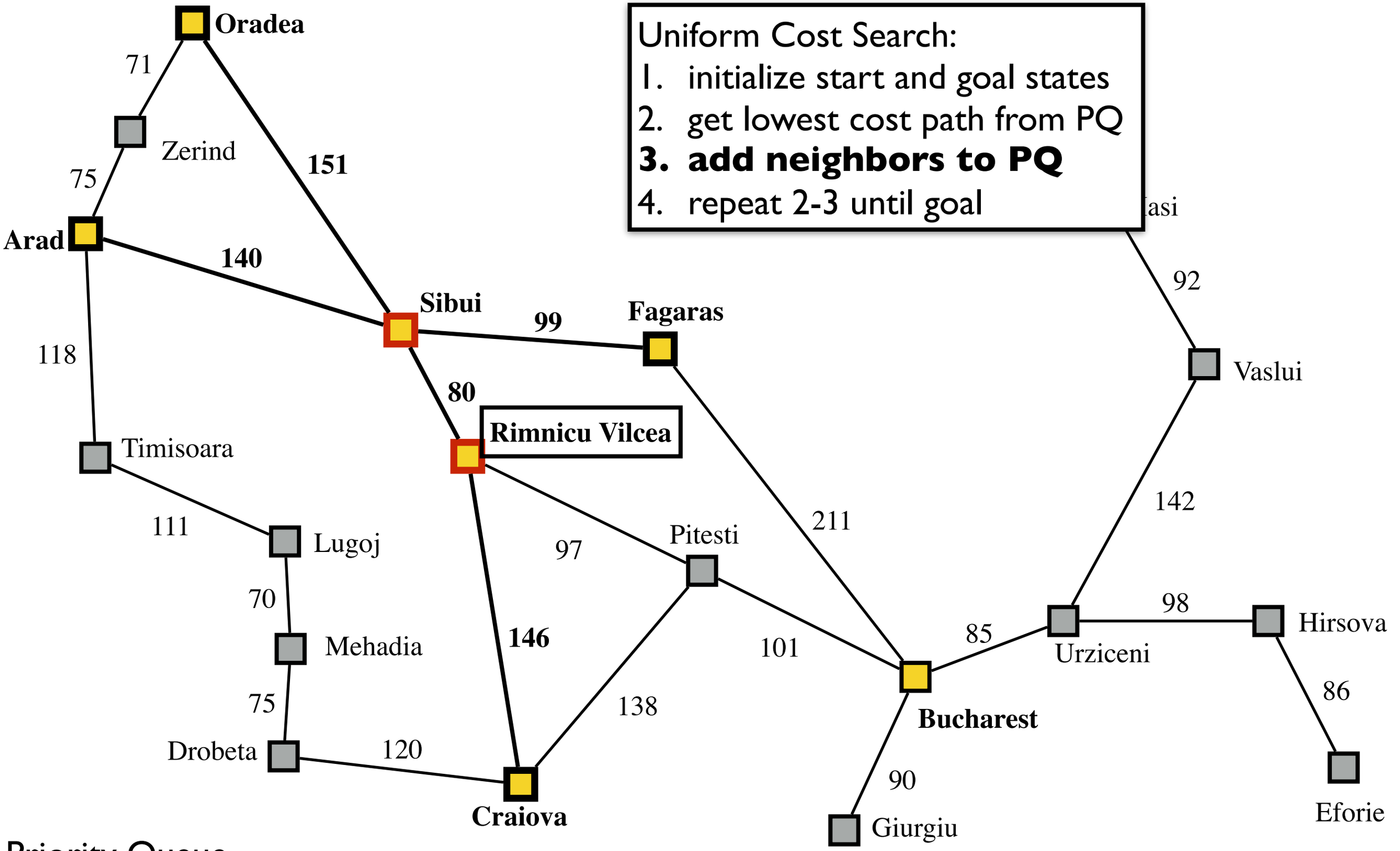


Priority Queue

node	F	A	O
parent	S	S	S
cost	99	140	151

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. **add neighbors to PQ**
4. repeat 2-3 until goal



Priority Queue

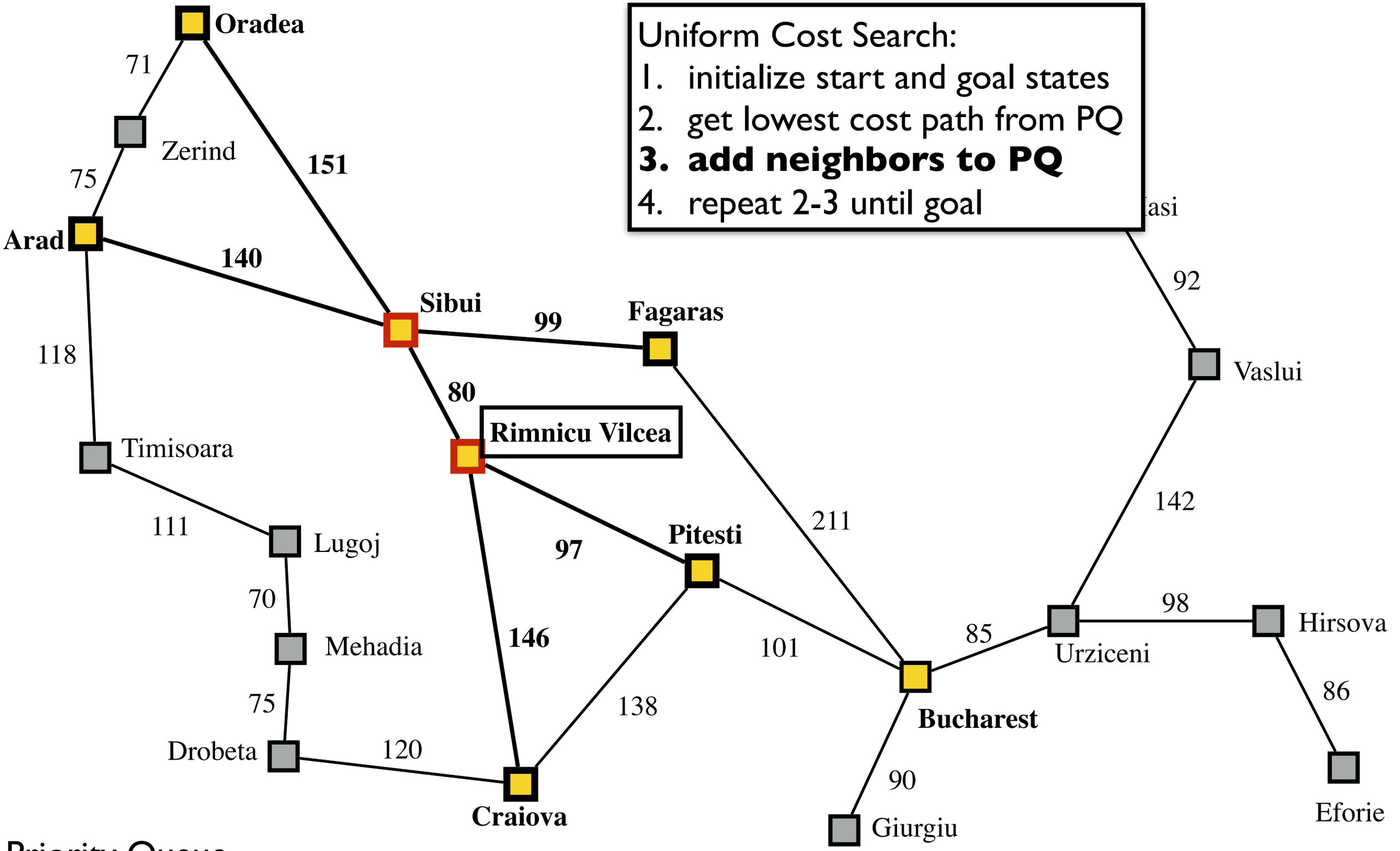
node	F	A	O	C
parent	S	S	S	RV
cost	99	140	151	226

← note different parent

= 80+146, i.e. cost from root to current node

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. **add neighbors to PQ**
4. repeat 2-3 until goal

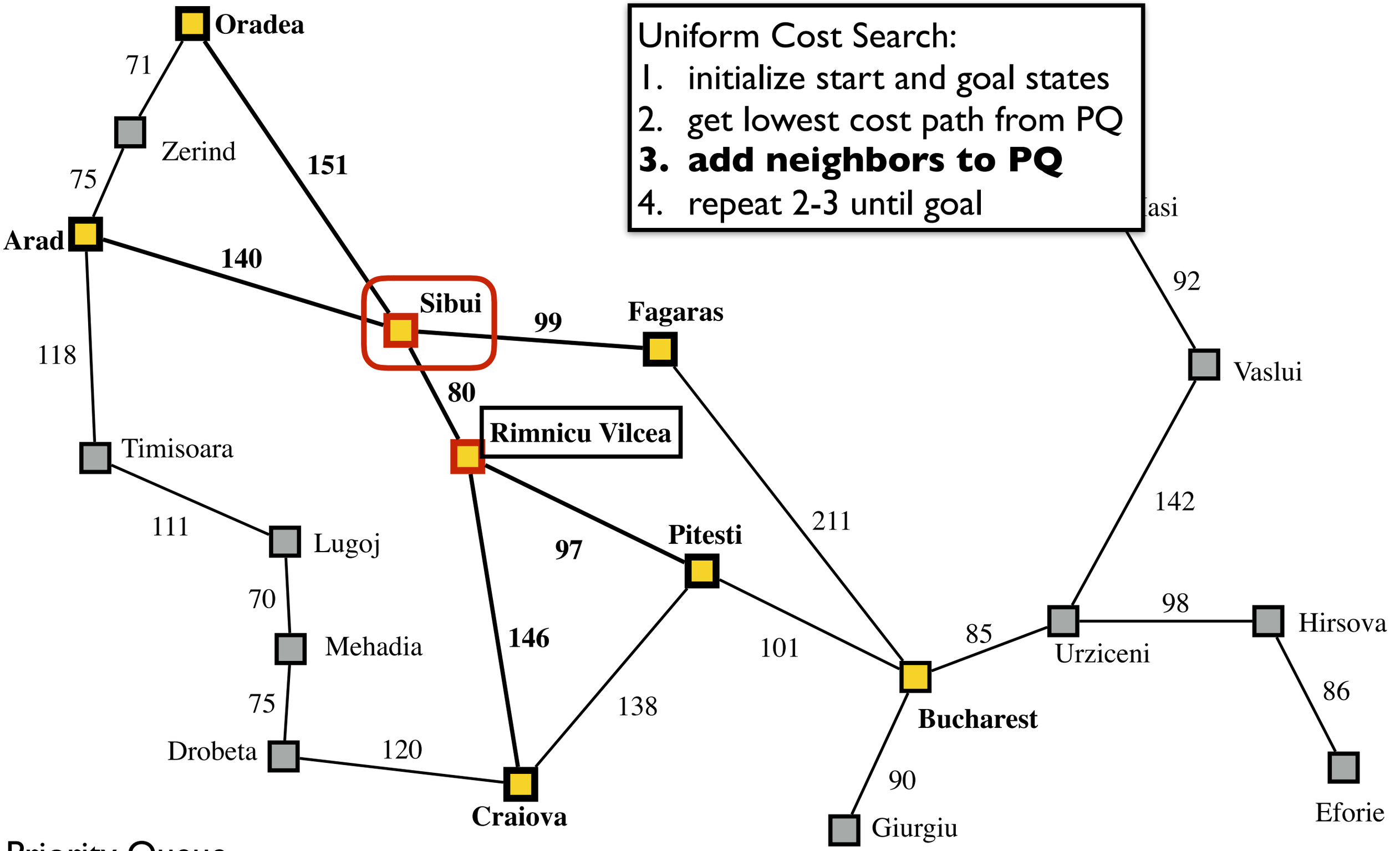


Priority Queue

node	F	A	O	P	C
parent	S	S	S	RV	RV
cost	99	140	151	177	226

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. **add neighbors to PQ**
4. repeat 2-3 until goal



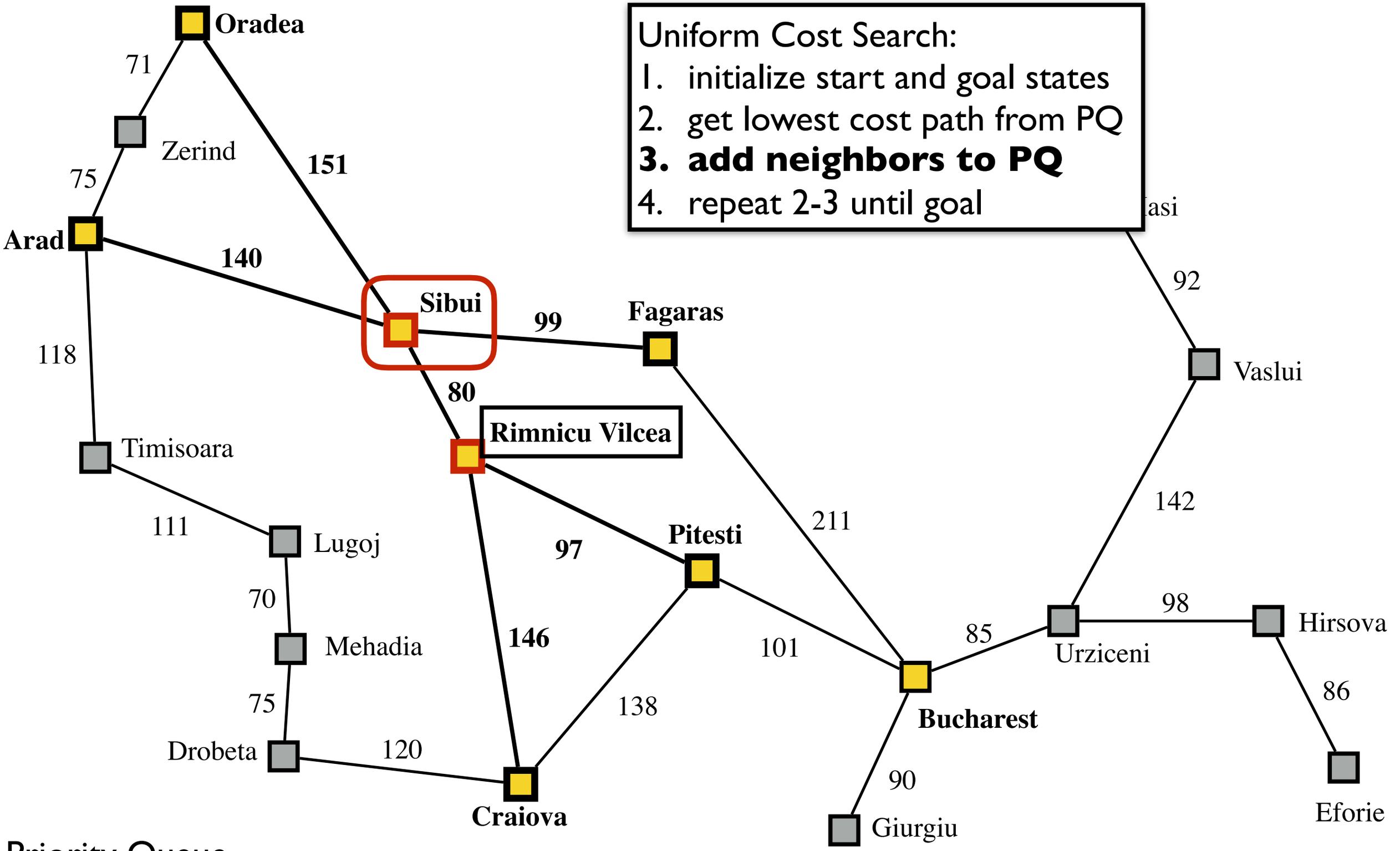
Priority Queue

node	F	A	O	P	C	S
parent	S	S	S	RV	RV	RV
cost	99	140	151	177	226	160

← Do we add this?

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. **add neighbors to PQ**
4. repeat 2-3 until goal



Priority Queue

node	F	A	O	P	C
parent	S	S	S	RV	RV
cost	99	140	151	177	226

S
RV
160

← Do we add this? No.

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. **add neighbors to PQ**
4. repeat 2-3 until goal

Oradea
71
Zerind
151
75
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Eforie
Craiova
Giurgiu
92
Vaslui
142
Iasi

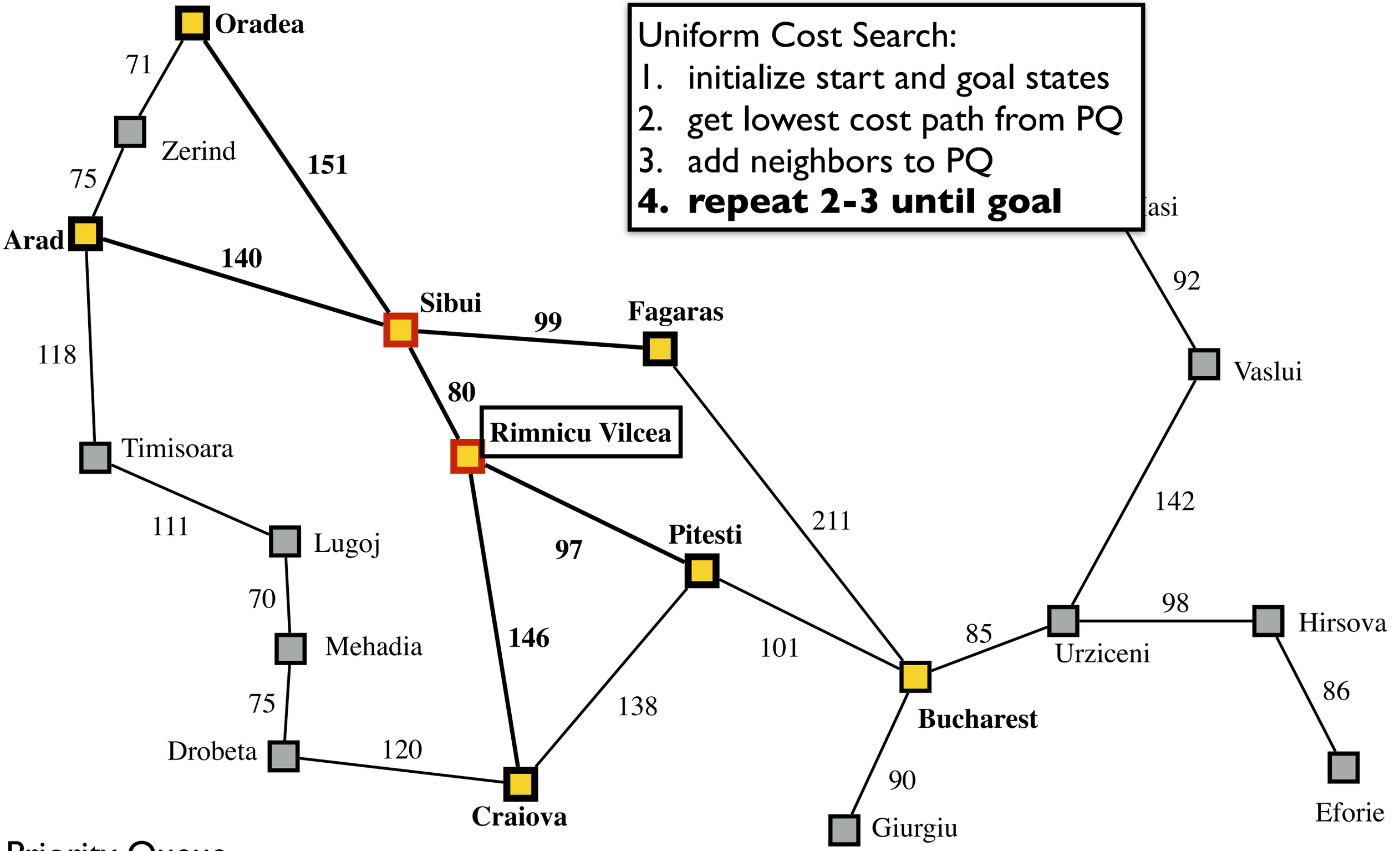
Priority Queue

node	F	A	O	P	C	S
parent	S	S	S	RV	RV	RV
cost	99	140	151	177	226	160

1) Any path through here has to be longer.
2) Nodes read from PQ are marked as "finalized" and not expanded again. R&V use the term "explored".

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. add neighbors to PQ
4. **repeat 2-3 until goal**

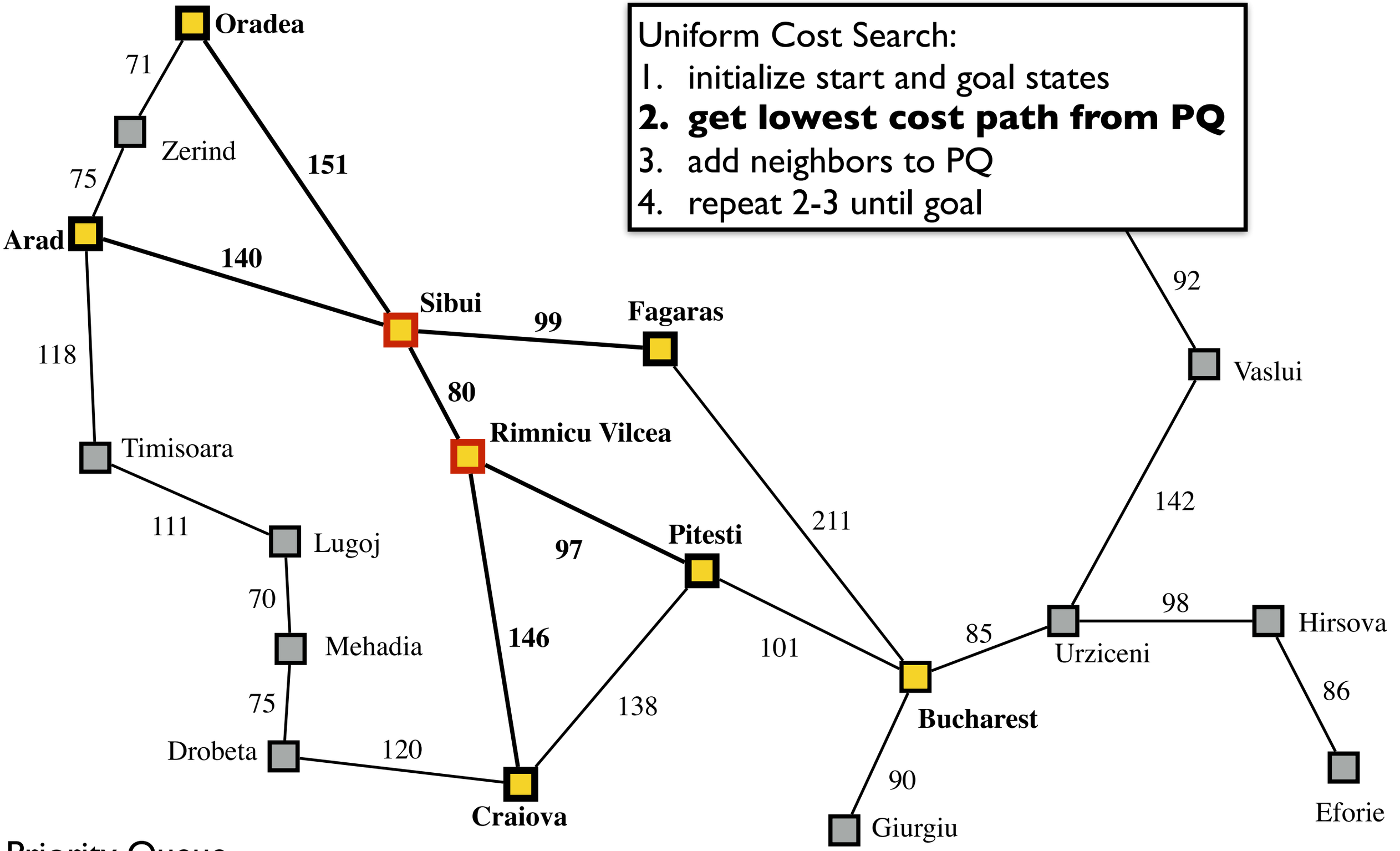


Priority Queue

node	F	A	O	P	C
parent	S	S	S	RV	RV
cost	99	140	151	177	226

Uniform Cost Search:

1. initialize start and goal states
2. **get lowest cost path from PQ**
3. add neighbors to PQ
4. repeat 2-3 until goal

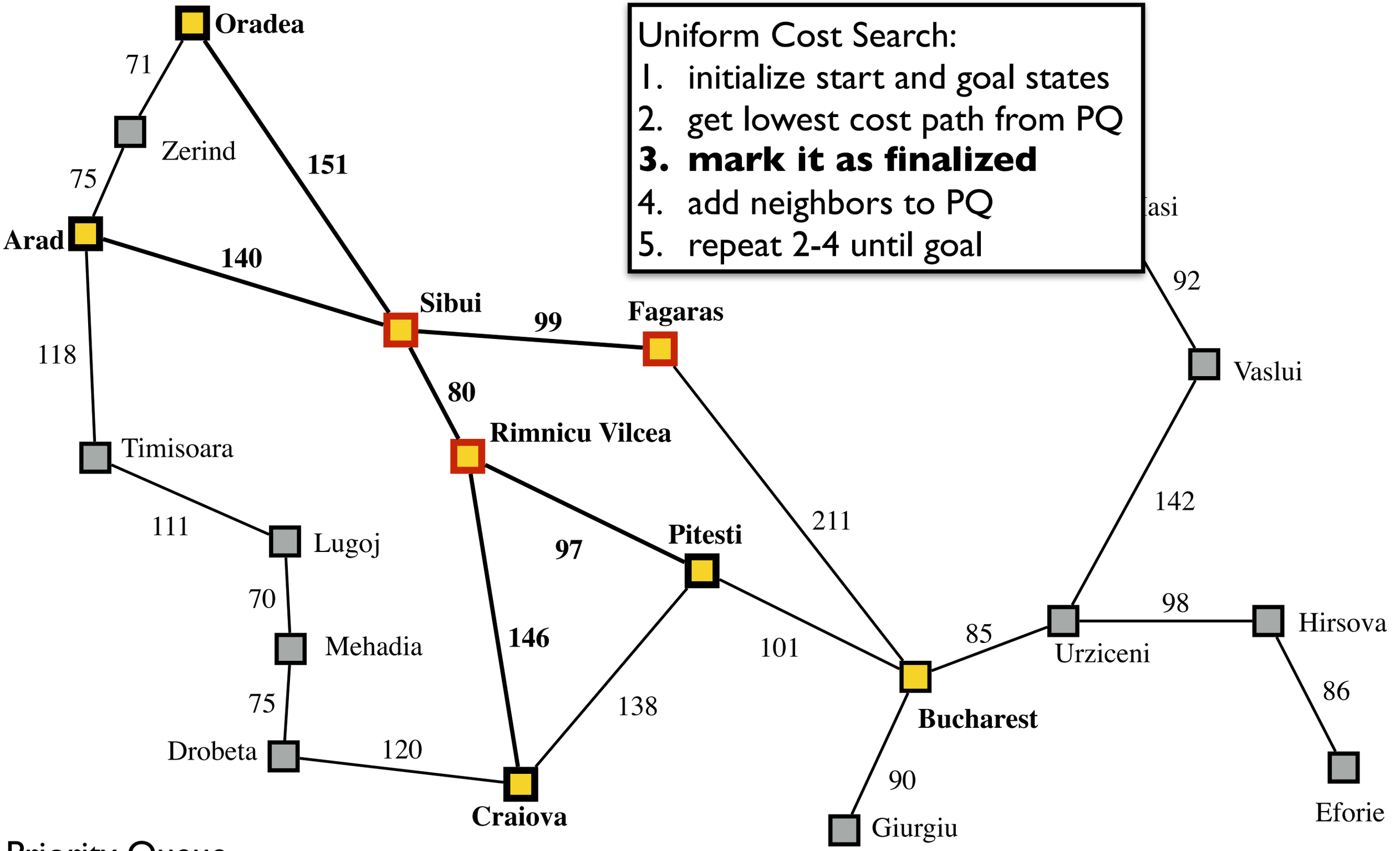


Priority Queue

node	F	A	O	P	C
parent	S	S	S	RV	RV
cost	99	140	151	177	226

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. **mark it as finalized**
4. add neighbors to PQ
5. repeat 2-4 until goal

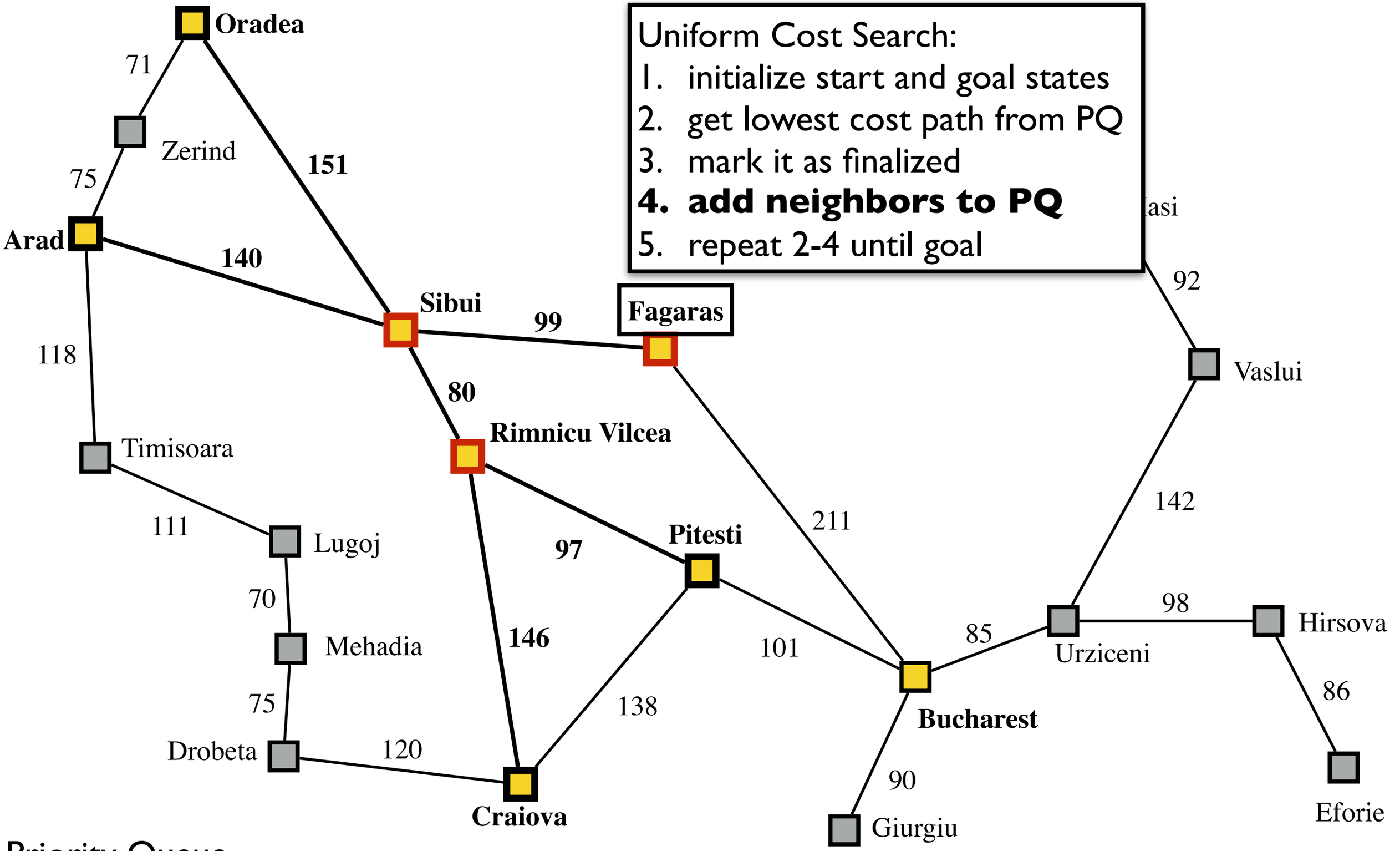


Priority Queue

node	F	A	O	P	C
parent	S	S	S	RV	RV
cost	99	140	151	177	226

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. **add neighbors to PQ**
5. repeat 2-4 until goal



Priority Queue

node	A	O	P	C
parent	S	S	RV	RV
cost	140	151	177	226

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. **add neighbors to PQ**
5. repeat 2-4 until goal

Oradea
71
Zerind
151
75
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Craiova
Giurgiu
Eforie
92
Vaslui
142
asi

Priority Queue

node	A	O	P	C	B
parent	S	S	RV	RV	F
cost	140	151	177	226	310

← Wait — this the goal! Can we stop?

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. **add neighbors to PQ**
5. repeat 2-4 until goal

Oradea
71
Zerind
151
75
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Craiova
Giurgiu
Eforie
asi
92
Vaslui
142

Priority Queue

node	A	O	P	C	B
parent	S	S	RV	RV	F
cost	140	151	177	226	310

← Wait — this the goal! Can we stop? No.
It's not finalized, so we might find a shorter route.

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. **add neighbors to PQ**
5. repeat 2-4 until goal

Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
10
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
146
Mehadia
101
85
Urziceni
98
Hirsova
75
138
86
Bucharest
Drobeta
120
90
Craiova
Giurgiu
Eforie
92
Vaslui
142
asi

Priority Queue

node	A	O	P	C	B
parent	S	S	RV	RV	F
cost	140	151	177	226	310

← Wait — this the goal! Can we stop? No.
It's not finalized, so we might find a shorter route.
A transporter at Arad? Who knows?

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. add neighbors to PQ
5. **repeat 2-4 until goal**

Oradea
71
Zerind
151
75
Arad
140
Sibui
99
Fagaras
92
118
80
Vaslui
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
142
70
146
98
Hirsova
Mehadia
85
Urziceni
101
75
138
86
Bucharest
Drobeta
120
90
Craiova
Giurgiu
Eforie

Priority Queue

node	A	O	P	C	B
parent	S	S	RV	RV	F
cost	140	151	177	226	310

Uniform Cost Search:

1. initialize start and goal states
2. **get lowest cost path from PQ**
3. mark it as finalized
4. add neighbors to PQ
5. repeat 2-4 until goal

Oradea
71
Zerind
151
75
Arad
140
Sibui
99
Fagaras
92
118
80
Vaslui
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
142
70
98
Hirsova
Mehadia
146
85
Urziceni
101
75
138
86
Bucharest
Drobeta
120
Craiova
90
Giurgiu
Eforie

Priority Queue

node	A	O	P	C	B
parent	S	S	RV	RV	F
cost	140	151	177	226	310

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. **mark it as finalized**
4. add neighbors to PQ
5. repeat 2-4 until goal

Oradea
71
Zerind
151
75
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Craiova
Giurgiu
Eforie
92
Vaslui
142
asi

Priority Queue

node	O	P	C	B
parent	S	RV	RV	F
cost	151	177	226	310

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. **add neighbors to PQ**
5. repeat 2-4 until goal

Oradea
71
Zerind
151
75
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Craiova
Giurgiu
Eforie
asi
92
Vaslui
142

Priority Queue

node	O	P	C	B
parent	S	RV	RV	F
cost	151	177	226	310

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. **add neighbors to PQ**
5. repeat 2-4 until goal

Oradea
71
Zerind
151
75
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
asi
92
Vaslui
142
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Eforie
Craiova
Giurgiu

Priority Queue

node	O	P	C	T	B
parent	S	RV	RV	A	F
cost	151	177	226	258	310

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. **add neighbors to PQ**
5. repeat 2-4 until goal

Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
asi
92
Vaslui
142
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Craiova
Giurgiu
Eforie

Priority Queue

node	O	P	Z	C	T	B
parent	S	RV	A	RV	A	F
cost	151	177	215	226	258	310

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. add neighbors to PQ
5. **repeat 2-4 until goal**

Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
98
Hirsova
85
Urziceni
75
138
86
Drobeta
120
Bucharest
90
Craiova
Giurgiu
Eforie
92
Vaslui
142
asi

Priority Queue

node	O	P	Z	C	T	B
parent	S	RV	A	RV	A	F
cost	151	177	215	226	258	310

Uniform Cost Search:

1. initialize start and goal states
2. **get lowest cost path from PQ**
3. mark it as finalized
4. add neighbors to PQ
5. repeat 2-4 until goal

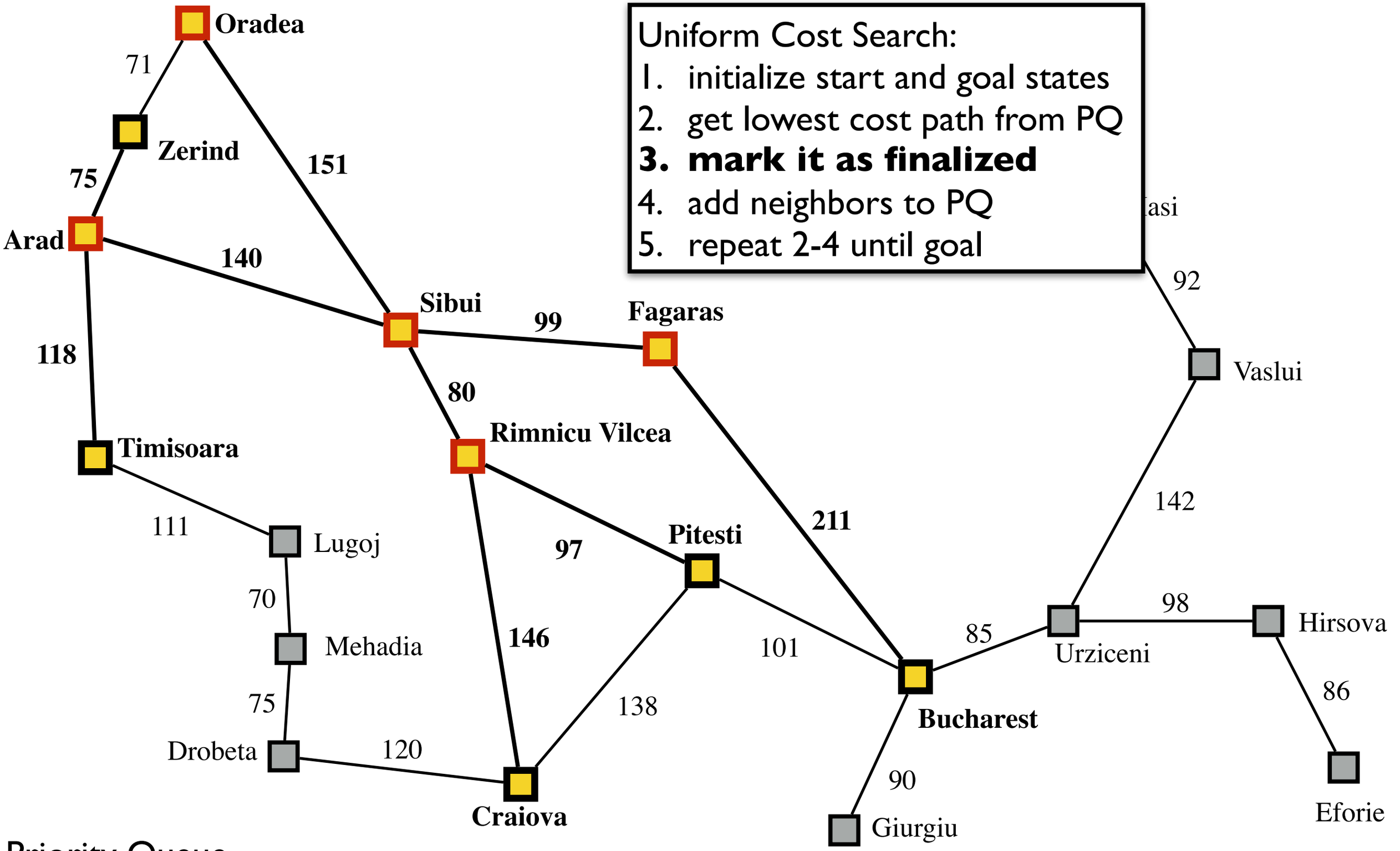
Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Craiova
Giurgiu
Eforie
92
Vaslui
142

Priority Queue

node	O	P	Z	C	T	B
parent	S	RV	A	RV	A	F
cost	151	177	215	226	258	310

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. **mark it as finalized**
4. add neighbors to PQ
5. repeat 2-4 until goal



Priority Queue

node	O	P	Z	C	T	B
parent	S	RV	A	RV	A	F
cost	151	177	215	226	258	310

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. **add neighbors to PQ**
5. repeat 2-4 until goal

Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
asi
92
Vaslui
142
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Craiova
Giurgiu
Eforie

Priority Queue

node	P	Z	C	T	B
parent	RV	A	RV	A	F
cost	177	215	226	258	310

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. **add neighbors to PQ**
5. repeat 2-4 until goal

Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Craiova
Giurgiu
Eforie
92
Vaslui
142
asi

Priority Queue

node	P	Z	Z	C	T	B
parent	RV	A	O	RV	A	F
cost	177	215	222	226	258	310

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. **add neighbors to PQ**
5. repeat 2-4 until goal

Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Craiova
Giurgiu
Eforie
asi
92
Vaslui
142

Priority Queue

node	P	Z	Z	C	T	B
parent	RV	A	O	RV	A	F
cost	177	215	222	226	258	310

Zerind is now in the PQ twice. Is this necessary?

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. **add neighbors to PQ**
5. repeat 2-4 until goal

Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
Craiova
90
Giurgiu
Eforie
asi
92
Vaslui
142

Priority Queue

node	P	Z	Z	C	T	B
parent	RV	A	O	RV	A	F
cost	177	215	222	226	258	310

Zerind is now in the PQ twice. Is this necessary? Yes. The PQ stores *paths*. There are two paths to Zerind. One via Arad, another via Oradea.

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. add neighbors to PQ
5. **repeat 2-4 until goal**

… This is tedious, which is why we write programs.

Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Craiova
Giurgiu
Eforie
92
Vaslui
142
asi

Priority Queue

node	P	Z	Z	C	T	B
parent	RV	A	O	RV	A	F
cost	177	215	222	226	258	310

Uniform Cost Search:

1. initialize start and goal states
2. **get lowest cost path from PQ**
3. mark it as finalized
4. add neighbors to PQ
5. repeat 2-4 until goal

Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Craiova
Giurgiu
Eforie
92
Vaslui
142

Priority Queue

node	P	Z	Z	C	T	B
parent	RV	A	O	RV	A	F
cost	177	215	222	226	258	310

Uniform Cost Search:

1. initialize start and goal states
2. **get lowest cost path from PQ**
3. mark it as finalized
4. add neighbors to PQ
5. repeat 2-4 until goal

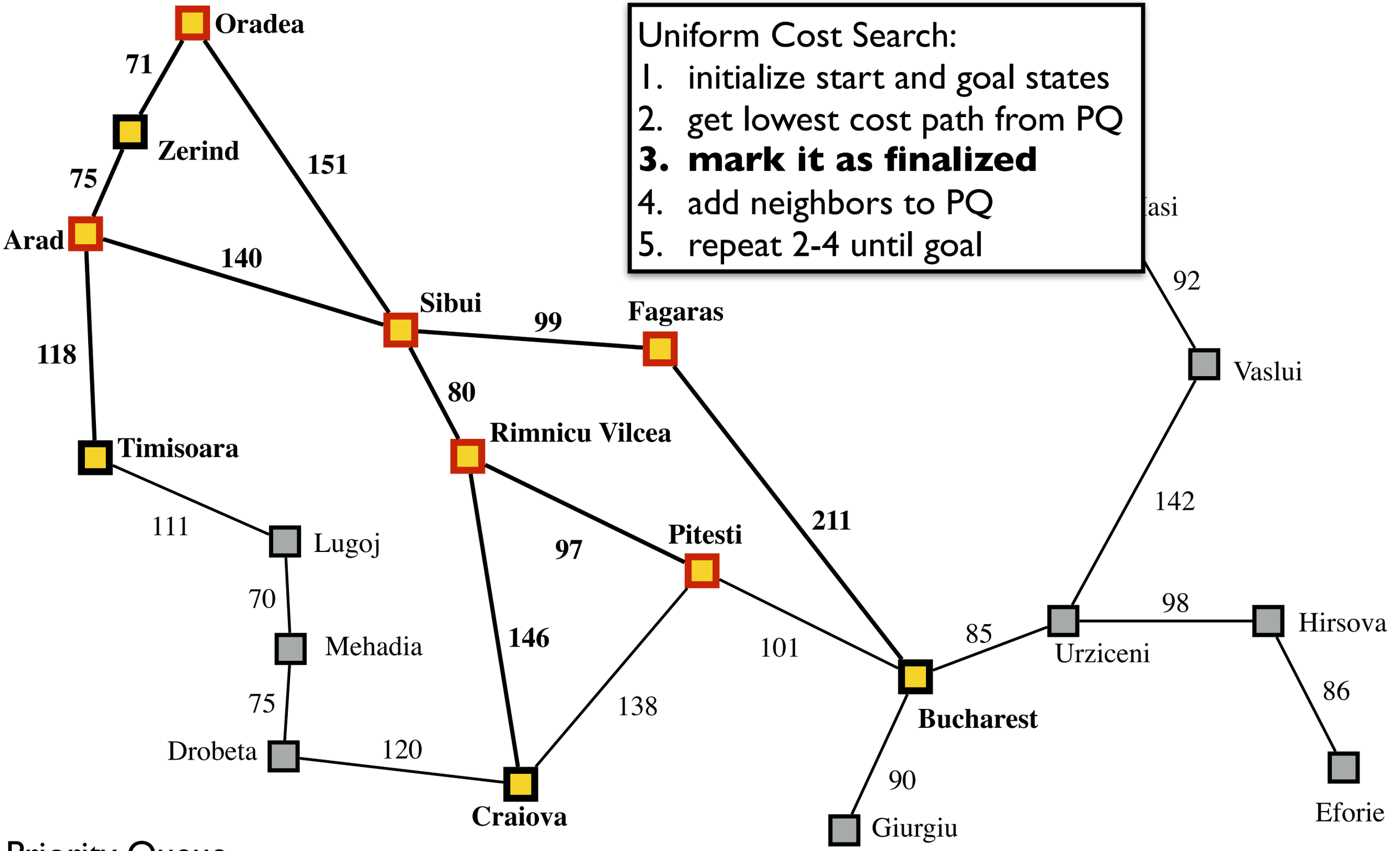
Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Craiova
Giurgiu
Eforie
92
Vaslui
142

Priority Queue

node	P	Z	Z	C	T	B
parent	RV	A	O	RV	A	F
cost	177	215	222	226	258	310

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. **mark it as finalized**
4. add neighbors to PQ
5. repeat 2-4 until goal



Priority Queue

node	P	Z	Z	C	T	B
parent	RV	A	O	RV	A	F
cost	177	215	222	226	258	310

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. **add neighbors to PQ**
5. repeat 2-4 until goal

Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
Craiova
90
Giurgiu
Eforie
92
Vaslui
142
asi

Priority Queue

node	Z	Z	C	T	B
parent	A	O	RV	A	F
cost	215	222	226	258	310

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. **add neighbors to PQ**
5. repeat 2-4 until goal

Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Craiova
Giurgiu
Eforie
asi
92
Vaslui
142

Priority Queue

node	Z	Z	C	T	**B**	B
parent	A	O	RV	A	**P**	F
cost	215	222	226	258	**278**	310

Look — a shorter route to Bucharest!
This is why we didn't stop before.

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. **add neighbors to PQ**
5. repeat 2-4 until goal

Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
asi
92
Vaslui
142
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Eforie
Craiova
Giurgiu

Priority Queue

node	Z	Z	C	T	B	B	C
parent	A	O	RV	A	P	F	P
cost	215	222	226	258	278	310	315

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. add neighbors to PQ
5. **repeat 2-4 until goal**

Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Craiova
Giurgiu
Eforie
92
Vaslui
142
asi

Priority Queue

node	Z	Z	C	T	B	B	C
parent	A	O	RV	A	P	F	P
cost	215	222	226	258	278	310	315

Uniform Cost Search:

1. initialize start and goal states
2. **get lowest cost path from PQ**
3. mark it as finalized
4. add neighbors to PQ
5. repeat 2-4 until goal

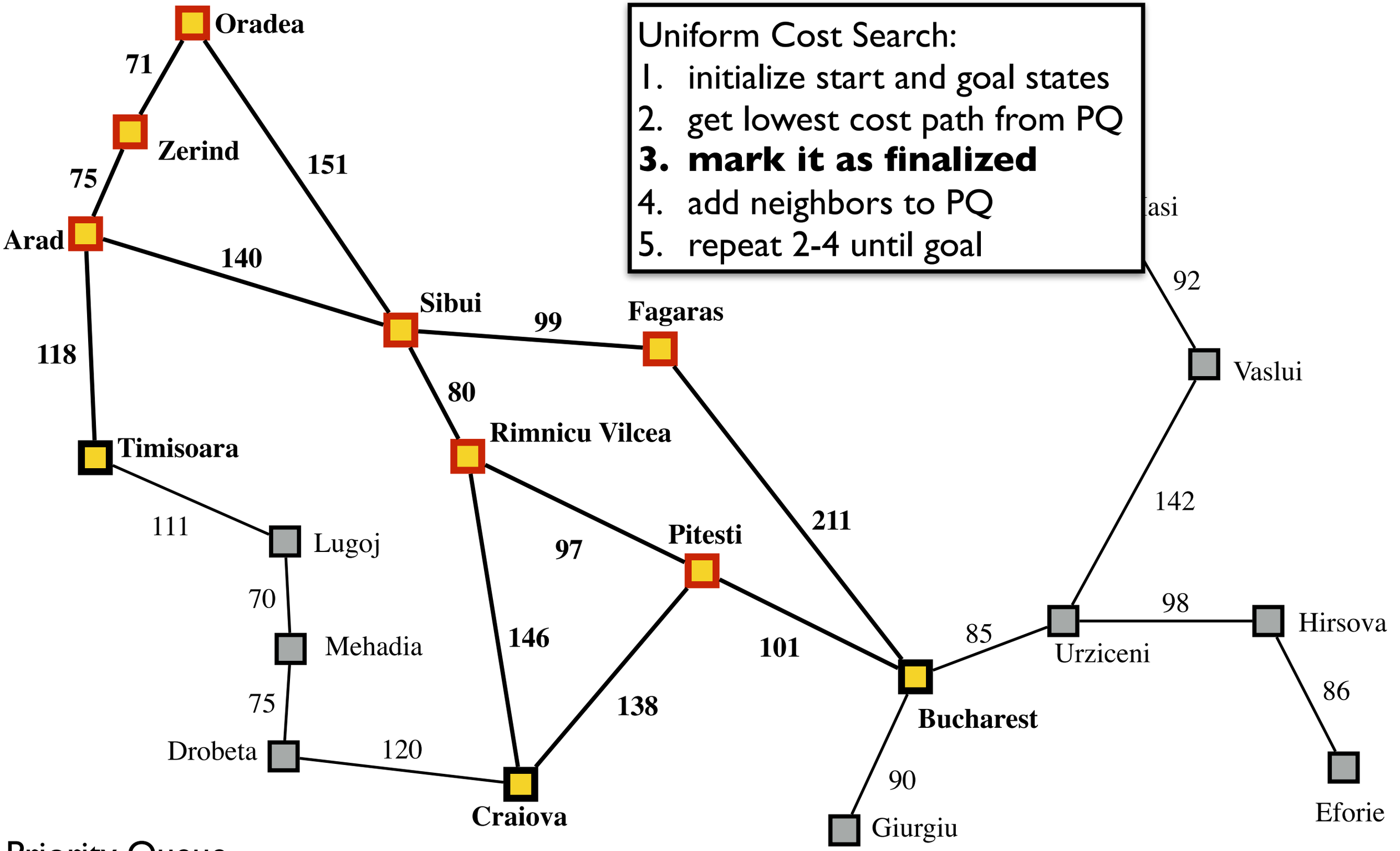
Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Craiova
Giurgiu
Eforie
92
Vaslui
142

Priority Queue

node	Z	Z	C	T	B	B	C
parent	A	O	RV	A	P	F	P
cost	215	222	226	258	278	310	315

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. **mark it as finalized**
4. add neighbors to PQ
5. repeat 2-4 until goal

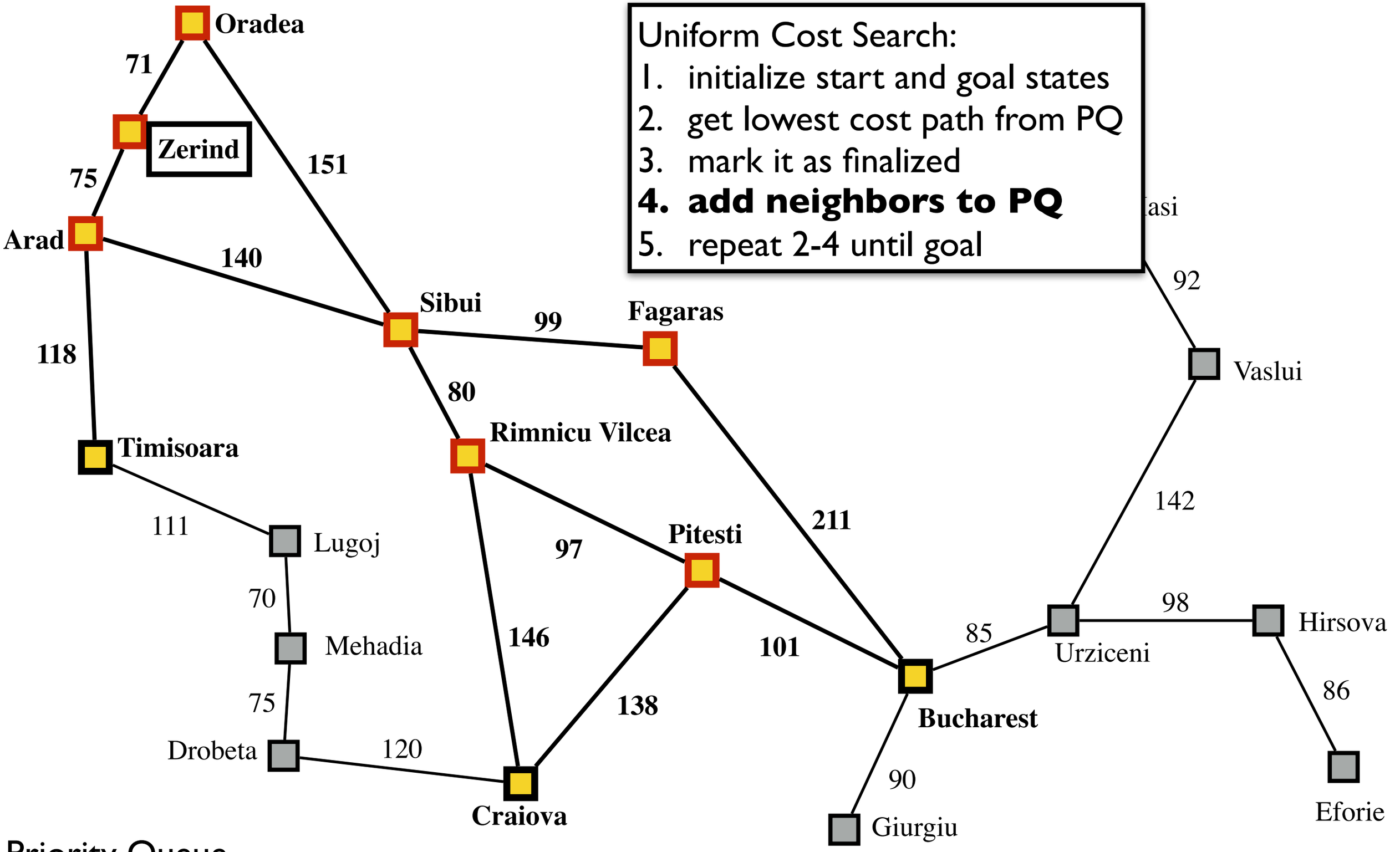


Priority Queue

node	Z	Z	C	T	B	B	C
parent	A	O	RV	A	P	F	P
cost	215	222	226	258	278	310	315

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. **add neighbors to PQ**
5. repeat 2-4 until goal



Priority Queue

node	Z	C	T	B	B	C
parent	O	RV	A	P	F	P
cost	222	226	258	278	310	315

All neighbors are finalized, so there are no paths to add, i.e. all of them would be longer that the ones we already have.

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. add neighbors to PQ
5. **repeat 2-4 until goal**

Oradea
71
Zerind
151
75
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
98
Hirsova
85
Urziceni
75
138
86
Bucharest
Drobeta
120
90
Craiova
Giurgiu
Eforie
asi
92
Vaslui
142

Priority Queue

node	Z	C	T	B	B	C
parent	O	RV	A	P	F	P
cost	222	226	258	278	310	315

We skip the next iteration, which does the same thing.

Uniform Cost Search:

1. initialize start and goal states
2. **get lowest cost path from PQ**
3. mark it as finalized
4. add neighbors to PQ
5. repeat 2-4 until goal

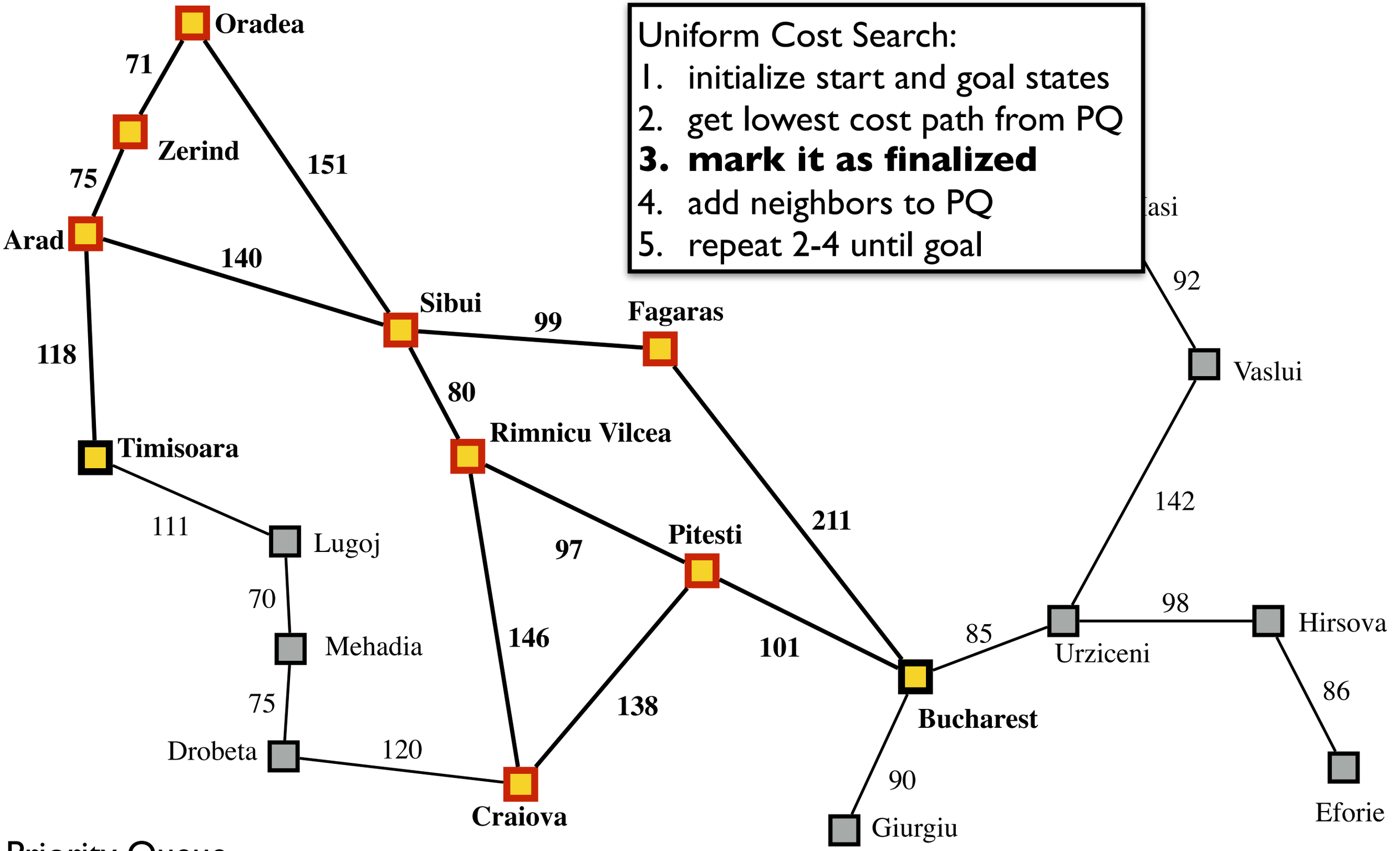
Oradea
71
Zerind
151
75
Arad
140
Sibui
99
Fagaras
92
Vaslui
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
142
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
Craiova
90
Giurgiu
Eforie

Priority Queue

node	C	T	B	B	C
parent	RV	A	P	F	P
cost	226	258	278	310	315

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. **mark it as finalized**
4. add neighbors to PQ
5. repeat 2-4 until goal



Priority Queue

node	C	T	B	B	C
parent	RV	A	P	F	P
cost	226	258	278	310	315

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. **add neighbors to PQ**
5. repeat 2-4 until goal

Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
98
Hirsova
85
Urziceni
75
138
86
Bucharest
Drobeta
120
90
Craiova
Giurgiu
Eforie
asi
92
Vaslui
142

Priority Queue

node	T	B	B	C	D
parent	A	P	F	P	C
cost	258	278	310	315	346

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. add neighbors to PQ
5. **repeat 2-4 until goal**

Oradea
71
Zerind
151
75
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
98
Hirsova
85
Urziceni
75
138
86
Drobeta
120
Bucharest
90
Craiova
Giurgiu
Eforie
142
92
Vaslui
asi

Priority Queue

node	T	B	B	C	D
parent	A	P	F	P	C
cost	258	278	310	315	346

Uniform Cost Search:

1. initialize start and goal states
2. **get lowest cost path from PQ**
3. **mark it as finalized**
4. add neighbors to PQ
5. repeat 2-4 until goal

Oradea
71
Zerind
151
75
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Craiova
Giurgiu
Eforie
92
Vaslui
142

Priority Queue

node	T	B	B	C	D
parent	A	P	F	P	C
cost	258	278	310	315	346

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. **add neighbors to PQ**
5. repeat 2-4 until goal

Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Craiova
Giurgiu
Eforie
92
Vaslui
142
asi

Priority Queue

node	B	B	C	D	L
parent	P	F	P	C	T
cost	278	310	315	346	369

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. mark it as finalized
4. add neighbors to PQ
5. **repeat 2-4 until goal**

Oradea
71
Zerind
151
75
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Craiova
Giurgiu
Eforie
asi
92
Vaslui
142

Priority Queue

node	B	B	C	D	L
parent	P	F	P	C	T
cost	278	310	315	346	369

Uniform Cost Search:

1. initialize start and goal states
2. **get lowest cost path from PQ**
3. mark it as finalized
4. add neighbors to PQ
5. repeat 2-4 until goal

Oradea
71
Zerind
75
151
Arad
140
Sibui
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
85
Urziceni
98
Hirsova
75
138
Bucharest
86
Drobeta
120
90
Eforie
Craiova
Giurgiu
92
Vaslui
142

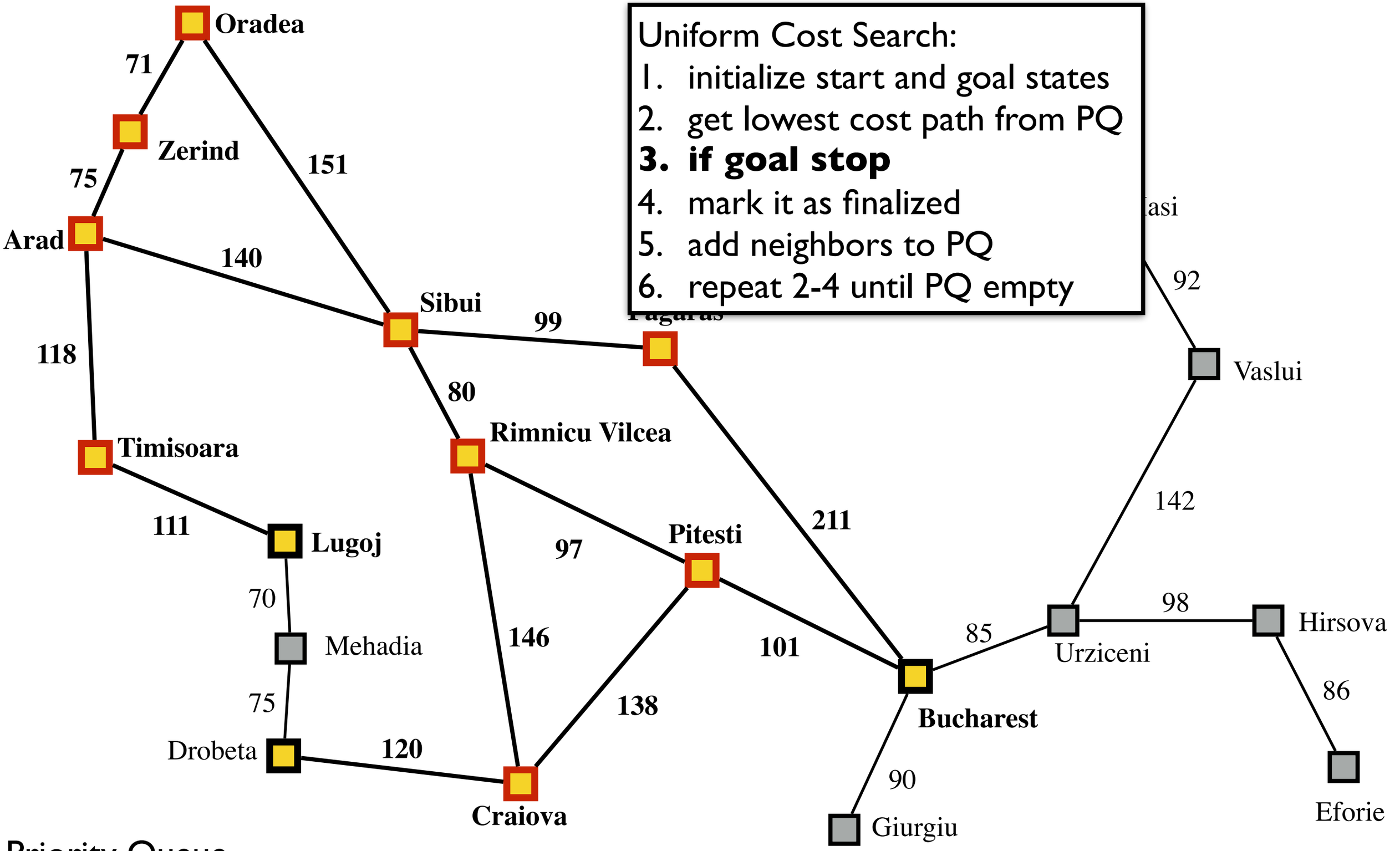
Priority Queue

node	B	B	C	D	L
parent	P	F	P	C	T
cost	278	310	315	346	369

Finally, we read the goal off the queue!
This means this is the shortest path to Bucharest.

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. **if goal stop**
4. mark it as finalized
5. add neighbors to PQ
6. repeat 2-4 until PQ empty



Priority Queue

node	B	B	C	D	L
parent	P	F	P	C	T
cost	278	310	315	346	369

Finally, we read the goal off the queue!
This means this is the shortest path to Bucharest.
And we could test right here.

Uniform Cost Search:

1. initialize start and goal states
2. get lowest cost path from PQ
3. **if goal stop**
4. mark it as finalized
5. add neighbors to PQ
6. repeat 2-4 until PQ empty

Oradea
71
Zerind
151
75
Arad
140
Sibui
99
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
98
Hirsova
85
Urziceni
75
138
86
Drobeta
120
Bucharest
90
Craiova
Giurgiu
Eforie
92
Vaslui
142

Priority Queue

node	B	B	C	D	L
parent	P	F	P	C	T
cost	278	310	315	346	369

We can recover the path by storing the shortest path to each node from the starting node as we go.

Algorithm for Uniform Cost Search

function UNIFORM-COST-SEARCH(*problem*) **returns** a solution, or failure

```
node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
frontier ← a priority queue ordered by PATH-COST, with node as the only element
explored ← an empty set
loop do
    if EMPTY?( frontier) then return failure
    node ← POP( frontier)   /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE( problem, node, action)
        if child.STATE is not in explored or frontier then
            frontier ← INSERT(child, frontier)
        else if child.STATE is in frontier with higher PATH-COST then
            replace that frontier node with child
```

initialization

goal not found

get next shortest path

goal found

mark node as "finalized"

add non-finalized neighbors to PQ

only keep lowest cost paths in PQ ("frontier")

Figure 3.14 Uniform-cost search on a graph. The algorithm is identical to the general graph search algorithm in Figure 3.7, except for the use of a priority queue and the addition of an extra check in case a shorter path to a frontier state is discovered. The data structure for *frontier* needs to support efficient membership testing, so it should combine the capabilities of a priority queue and a hash table.

Uniform-cost search

- Expand node with *lowest path cost.*
- Don't care about number of steps only total cost. (Need positive costs)

- Complete?
- Time?
- Space?
- Optimal?

- Do these apply?
  - *b* - maximum branching factor of search tree
  - *d* - depth of least-cost solution
  - *m* - maximum depth of state space (can be infinite)

Uniform-cost search

- Expand node with *lowest path cost.*
- Don't care about number of steps only total cost. (Need positive costs)

- Complete? Yes.
- Time? Worst case O(b[C/ε]), where C is cost of opt. soln., ε is min cost
- Space? Same.
- Optimal? Yes, notes expanded in increasing order of cost.

- IF C=d and ε=1, same cost as BFS.
- Equivalent to BFS if all costs are equal, except UCS does not stop when finding goal, and instead examines all the nodes on the frontier.
- Can be worse than BFS if costs are unequal and many small steps are ultimately worse than larger less costly steps.

Summary of uninformed search algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

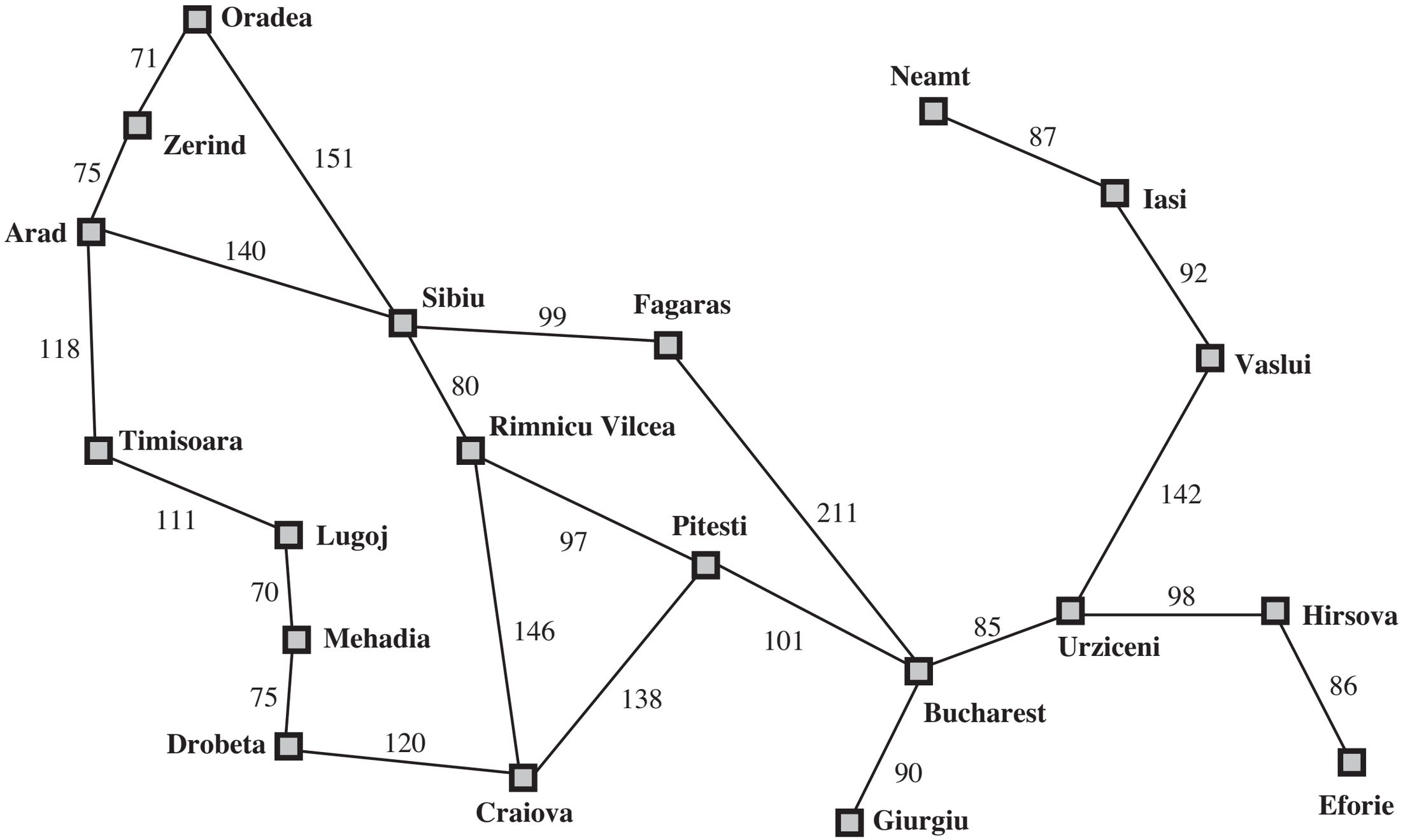
Evaluation of search strategies. b is the branching factor; d is the depth of solution; m is the maximum depth of the search tree; l is the depth limit.

- b = branching factor
- d = depth of shallowest goal state
- m = depth of search space
- l = depth limit

Problems with uninformed search

- time complexity is exponential
- state spaces can be very large

Romanian road map example



In absence of knowledge, node expansion order is arbitrary (e.g. alphabetical).
What other information can we use?

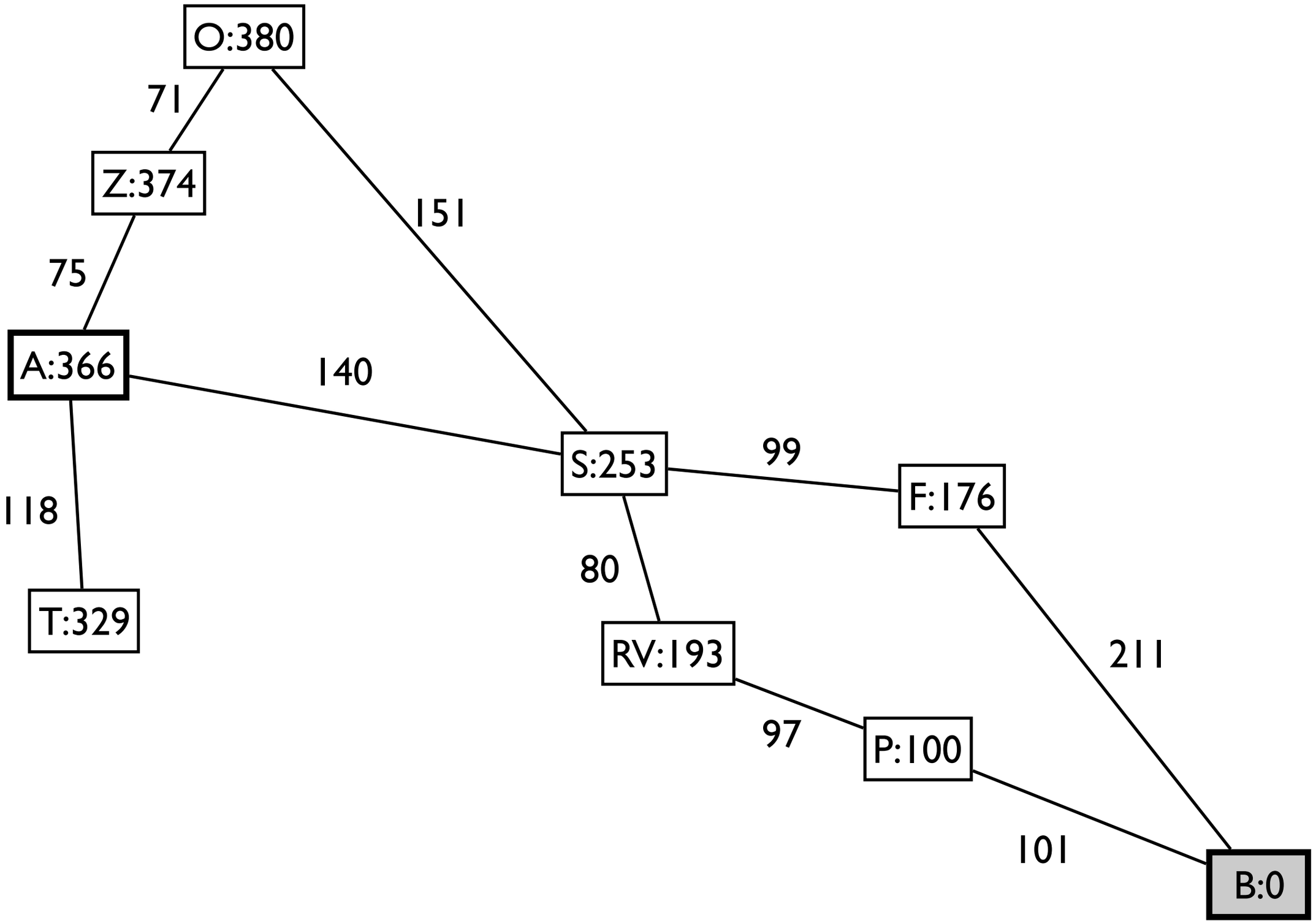
Informed search

- nodes are selected based on an **evaluation function** f(n)
- Idea is that f(n) will help us make better choices about node expansion order
- most informed search methods uses **heuristic function** h(n):
  - h(n) = estimated cost of cheapest path to goal
  - h(n) only depends on *state*, not path, (why?)

Greedy best-first search

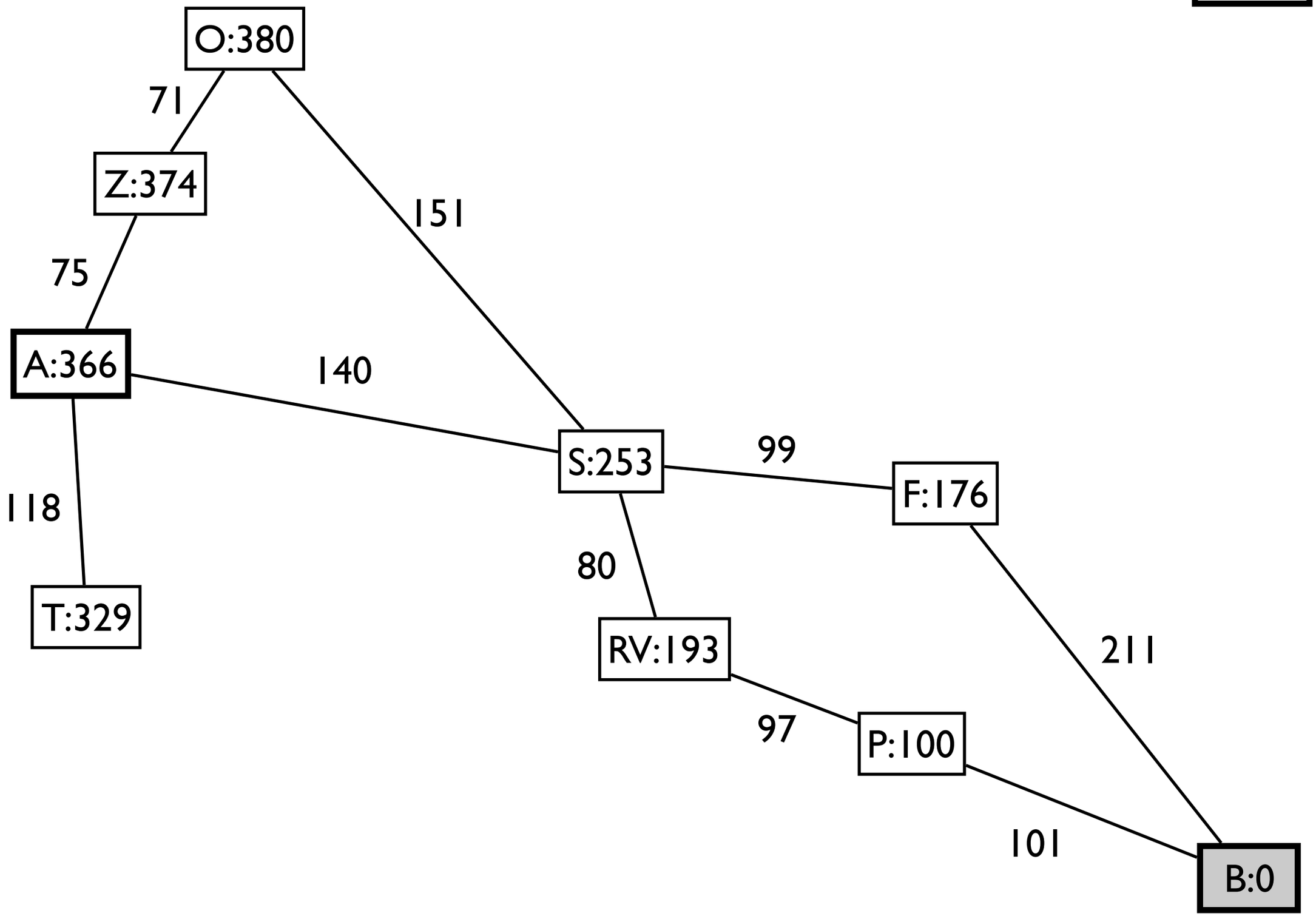
- expand node that is closest (by the heuristic) to the goal
- f(n) = h(n)

Map with straight-line distances to Bucharest for each city

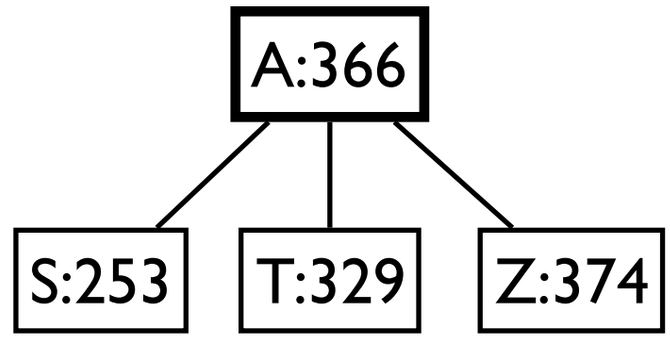


Greedy search: always expand best f(n) = h(n)

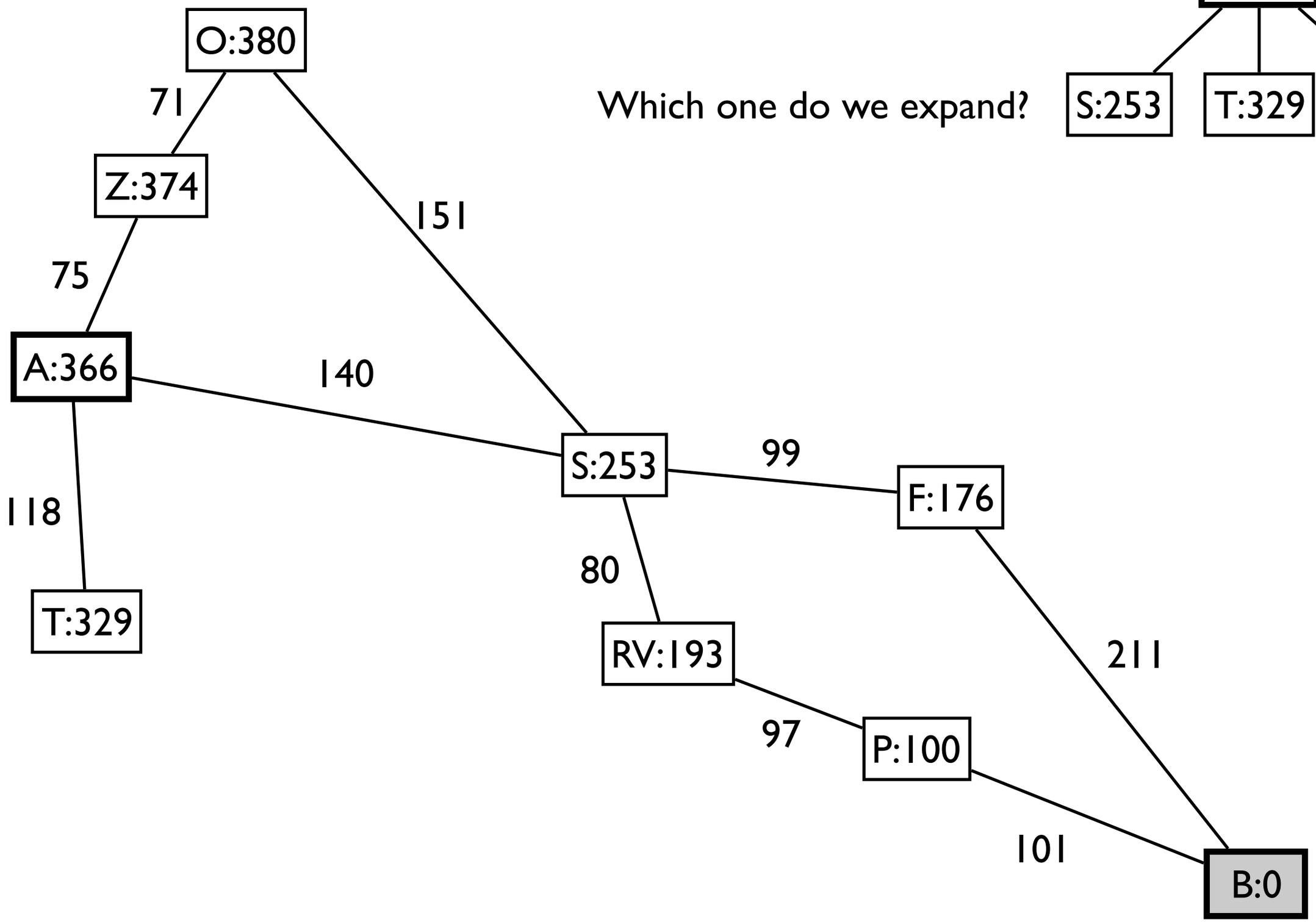
A:366



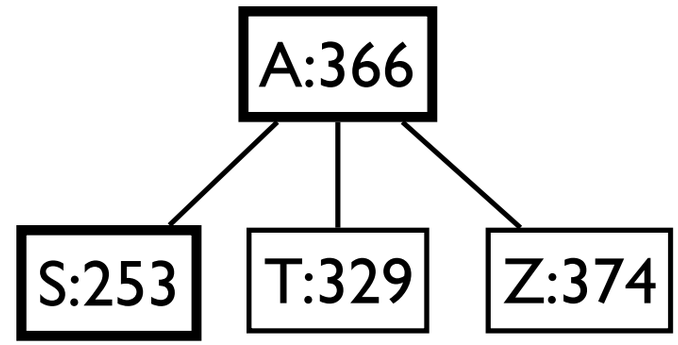
Greedy search: always expand best f(n) = h(n)



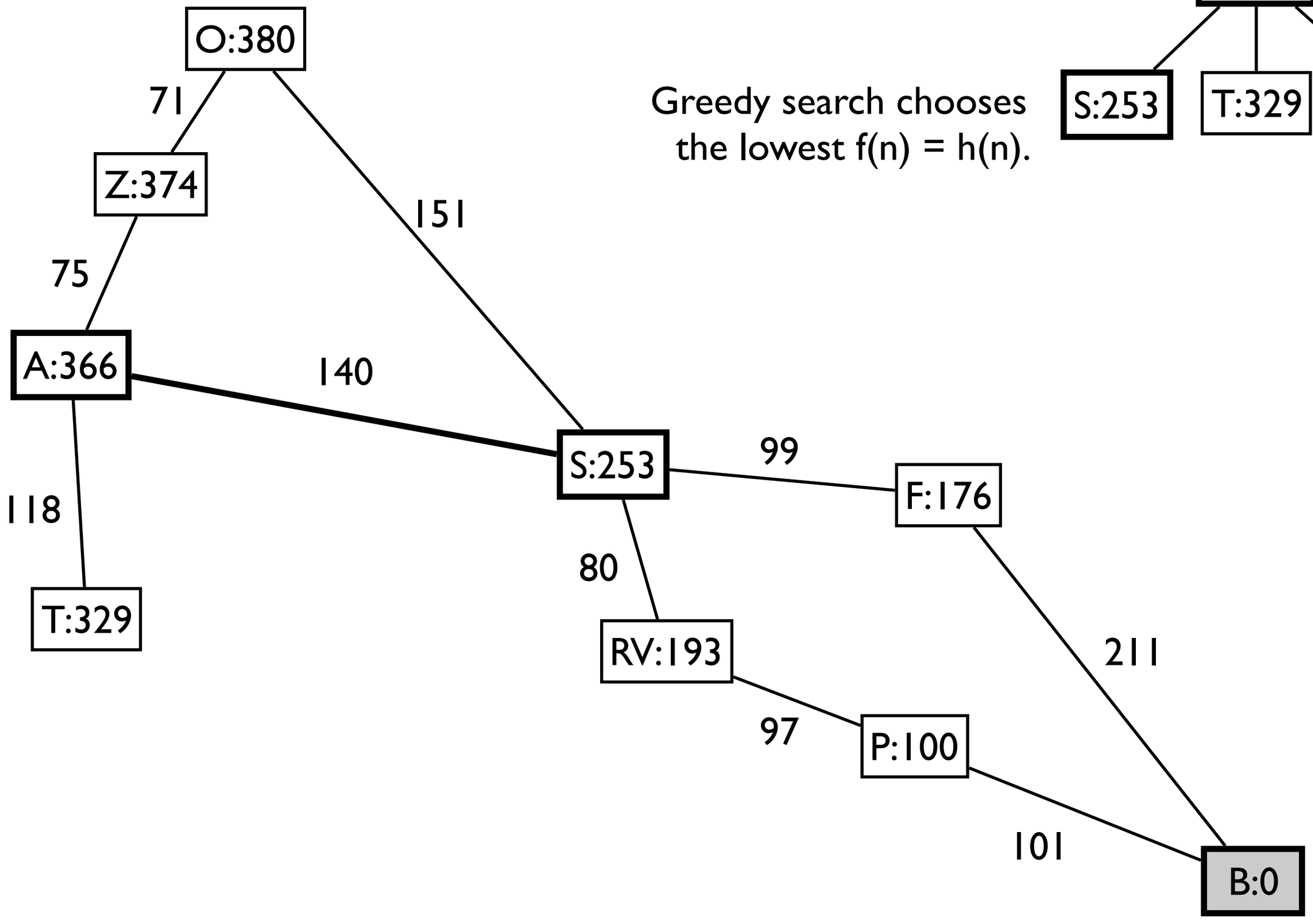
Which one do we expand?



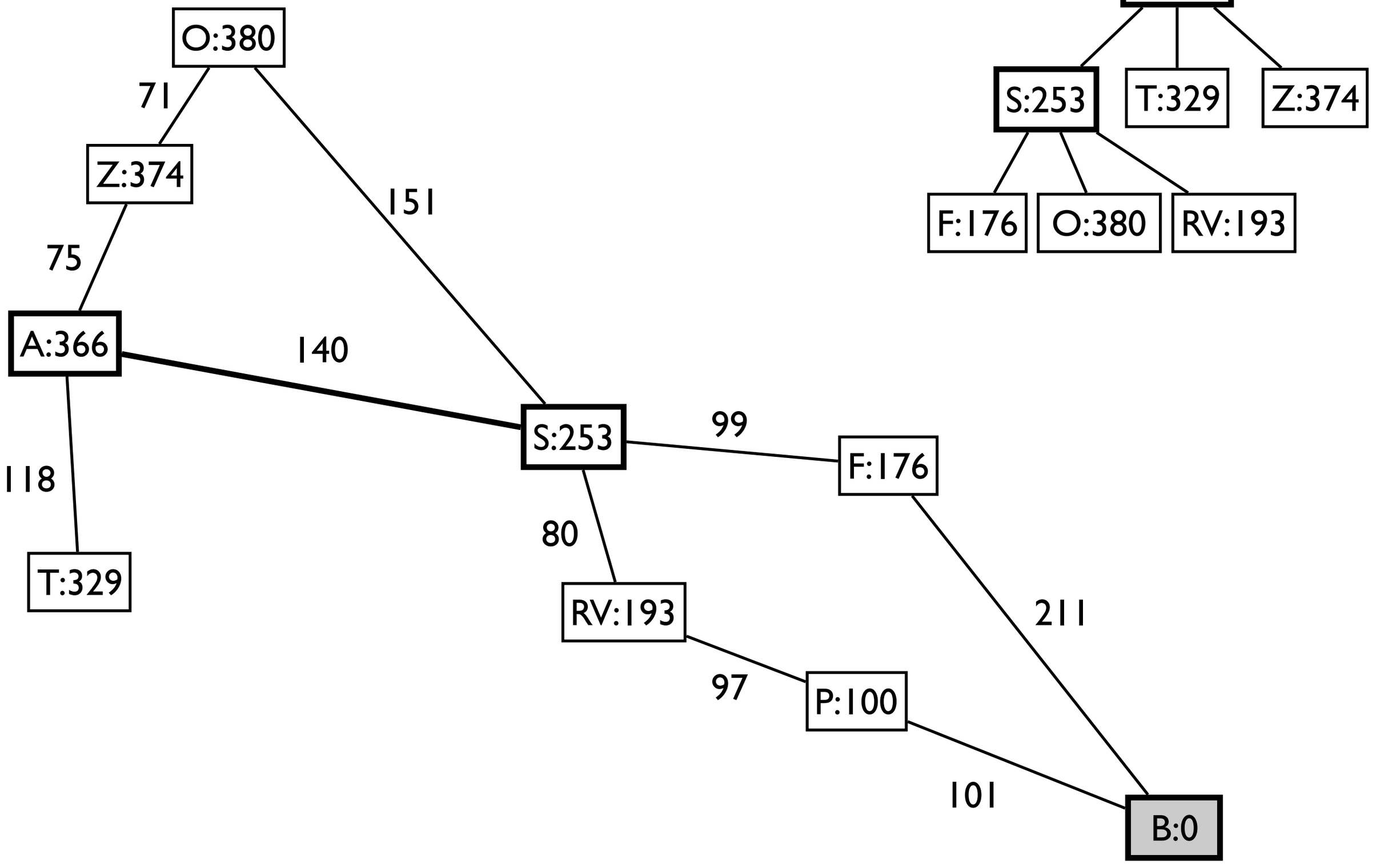
Greedy search: always expand best f(n) = h(n)



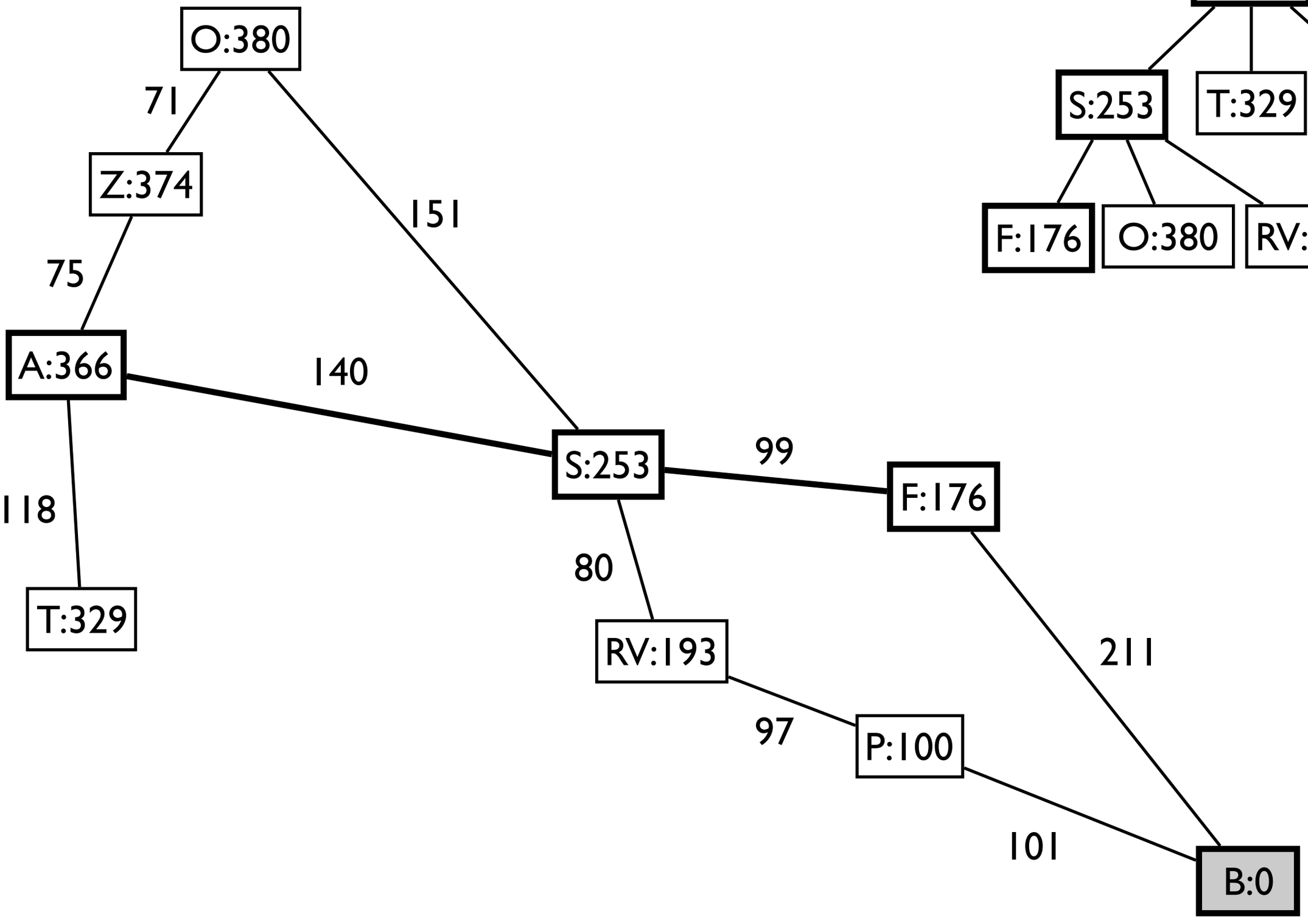
Greedy search chooses the lowest f(n) = h(n).



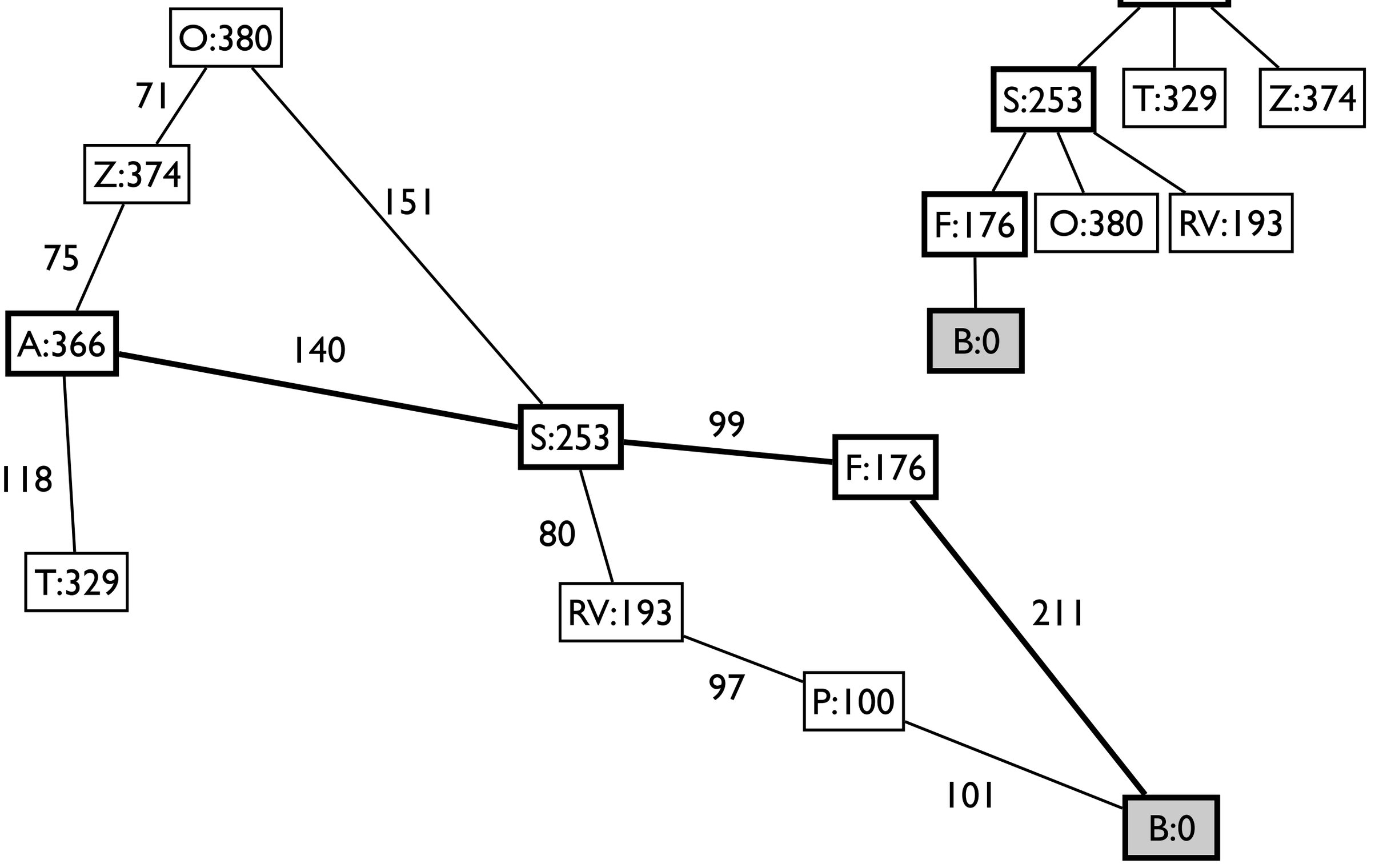
Greedy search: always expand best f(n) = h(n)



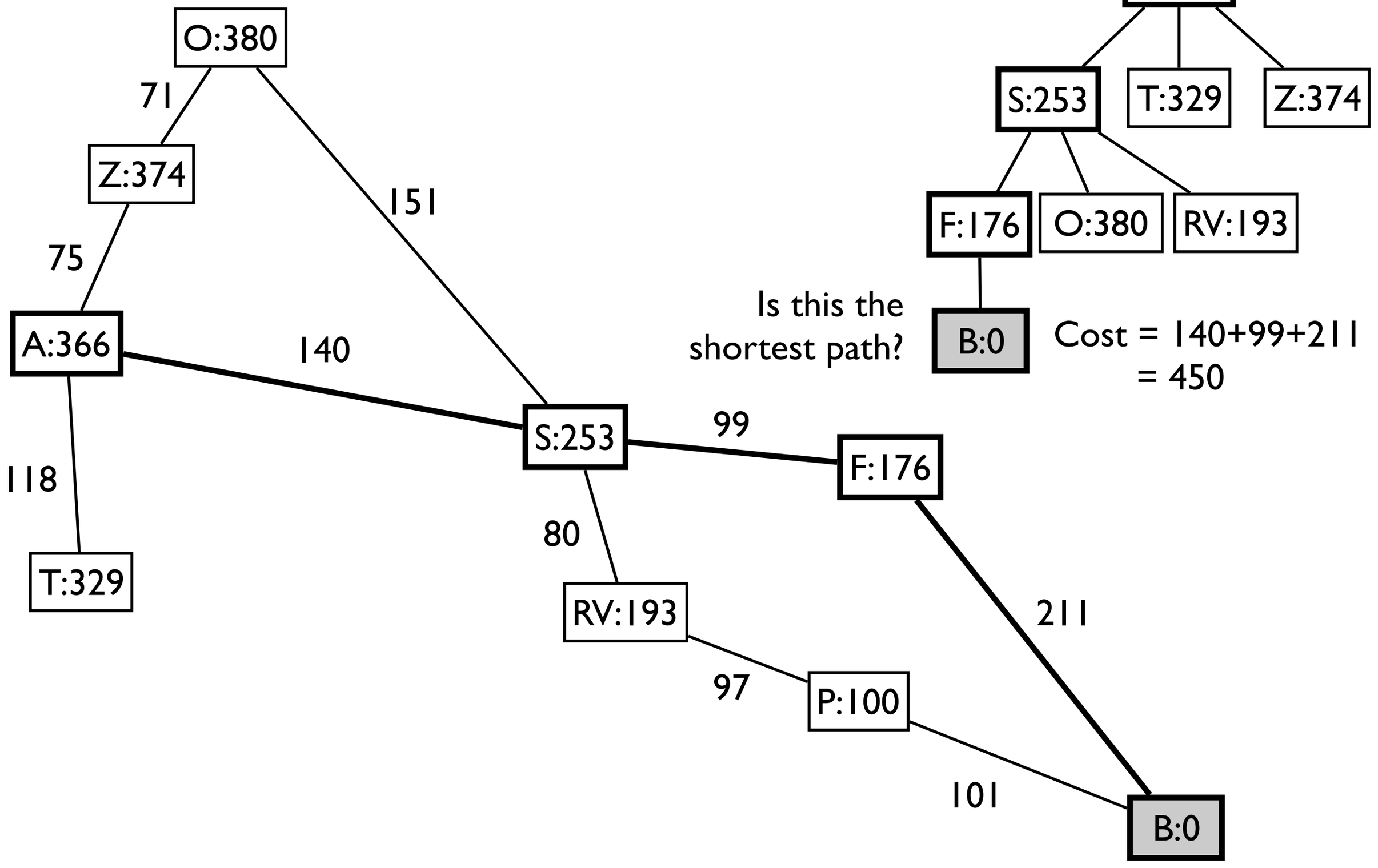
Greedy search: always expand best f(n) = h(n)



Greedy search: always expand best f(n) = h(n)



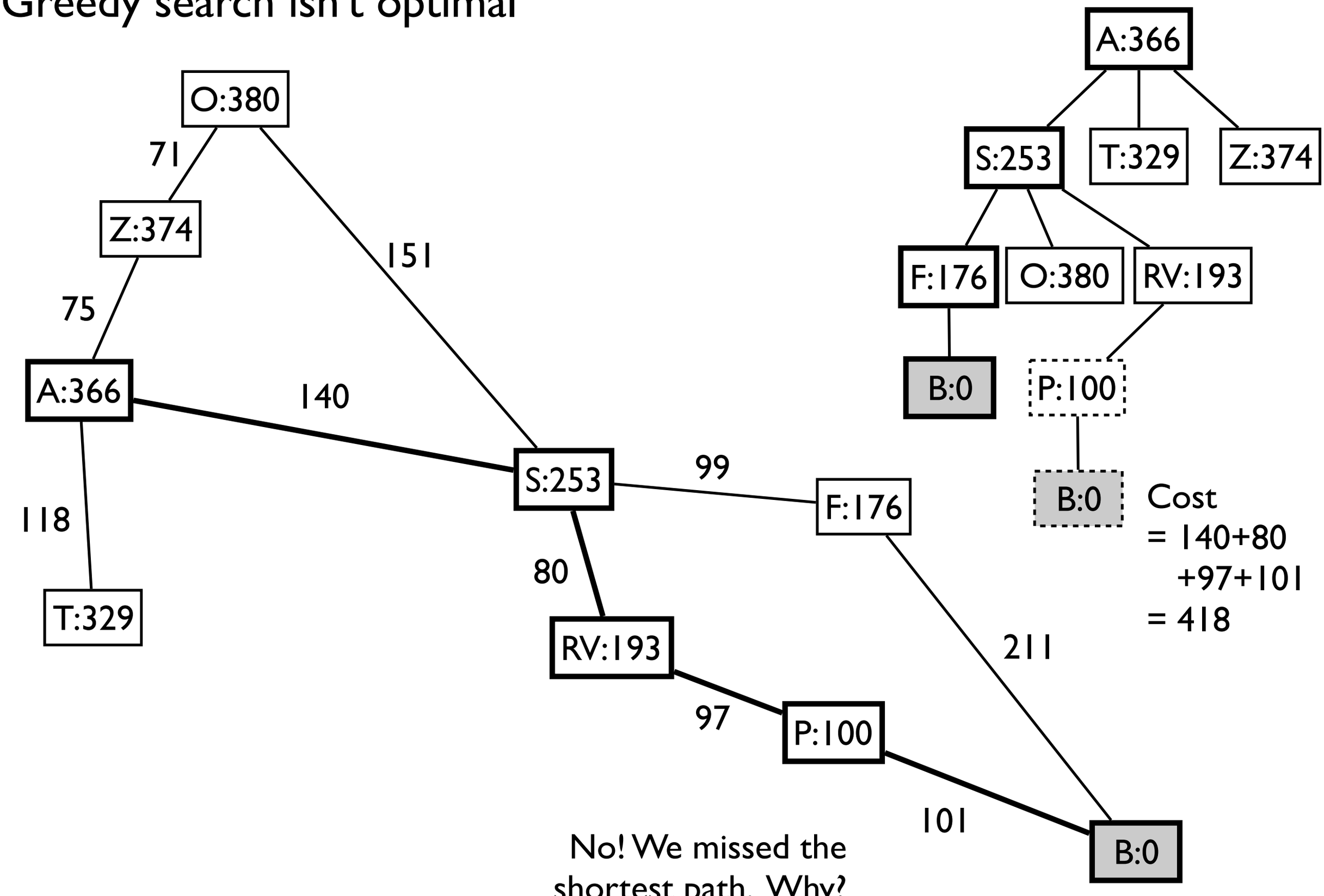
Greedy search: always expand best f(n) = h(n)



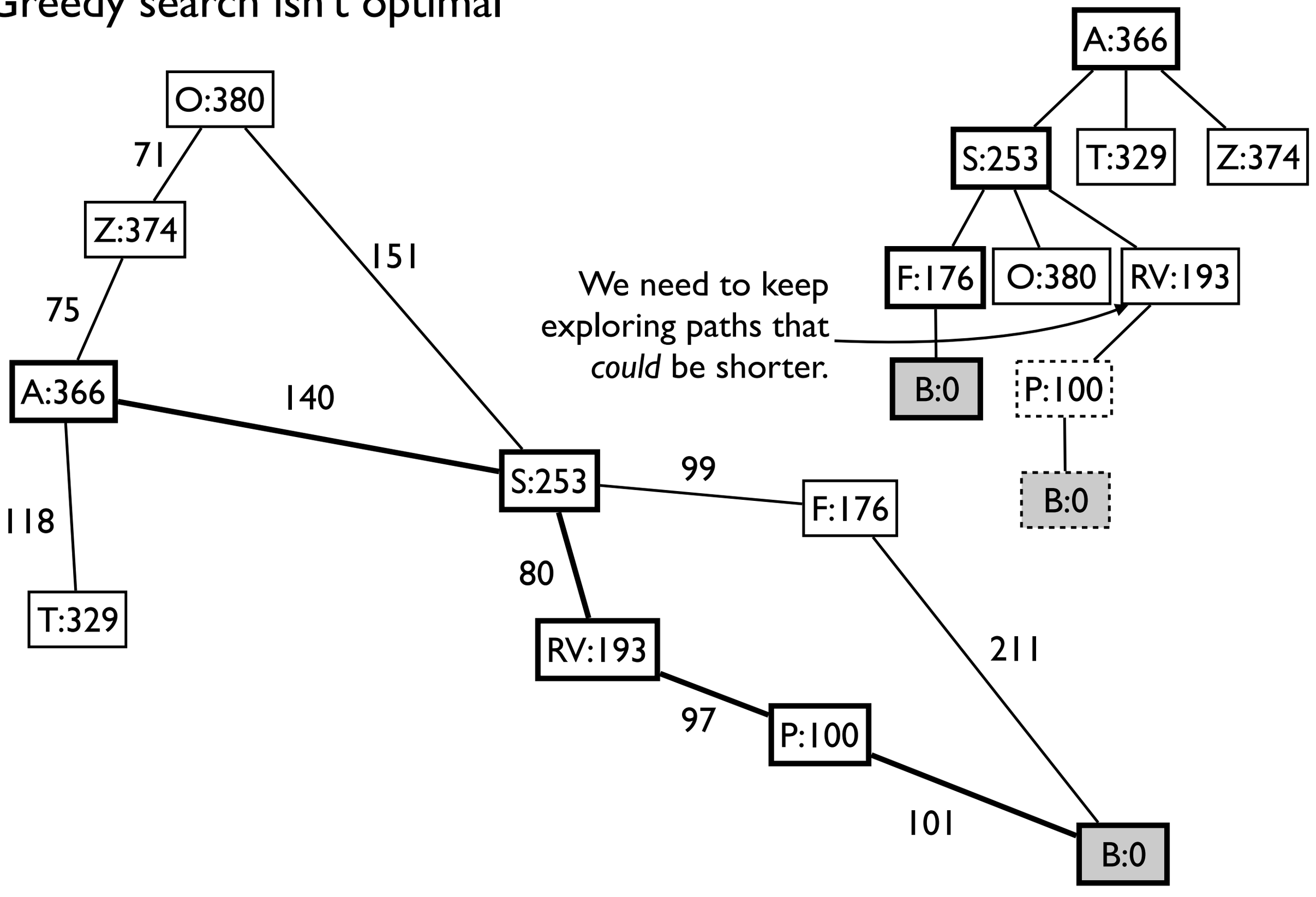
Is this the shortest path?

Cost = 140+99+211 = 450

Greedy search isn't optimal



Greedy search isn't optimal



General case: Greedy best-first search

- nodes are selected based on evaluation function f(n) = h(n),
  i.e. expand node that is closest (by the heuristic) to the goal
- h(n) only depends on state at node n, not path
- h(n = goal) = 0
- implementation is same as uniform cost search, using priority queue
- Complexity:
  - worst case?
  - best case?

General case: Greedy best-first search

- nodes are selected based on evaluation function f(n) = h(n),
  i.e. expand node that is closest (by the heuristic) to the goal
- h(n) only depends on state at node n, not path
- h(n = goal) = 0
- implementation is same as uniform cost search, using priority queue
- Complexity:
  - worst case? $O(b^m)$ b=branching factor, m=max depth
  - best case? O(m) depends on heuristic

General case: Greedy best-first search

- nodes are selected based on evaluation function f(n) = h(n),
  i.e. expand node that is closest (by the heuristic) to the goal
- h(n) only depends on state at node n, not path
- h(n = goal) = 0
- implementation is same as uniform cost search, using priority queue
- Complexity:
  - worst case? $O(b^m)$ b=branching factor, m=max depth
  - best case? O(m) depends on heuristic
- Complete?

General case: Greedy best-first search

- nodes are selected based on evaluation function f(n) = h(n),
  i.e. expand node that is closest (by the heuristic) to the goal
- h(n) only depends on state at node n, not path
- h(n = goal) = 0
- implementation is same as uniform cost search, using priority queue
- Complexity:
  - worst case? $O(b^m)$ b=branching factor, m=max depth
  - best case? O(m) depends on heuristic
- Complete? No. Like DFS, can go down infinite path.

General case: Greedy best-first search

- nodes are selected based on evaluation function f(n) = h(n),
  i.e. expand node that is closest (by the heuristic) to the goal
- h(n) only depends on state at node n, not path
- h(n = goal) = 0
- implementation is same as uniform cost search, using priority queue
- Complexity:
  - worst case? $O(b^m)$ b=branching factor, m=max depth
  - best case? O(m) depends on heuristic
- Complete? No. Like DFS, can go down infinite path.
- Optimal?

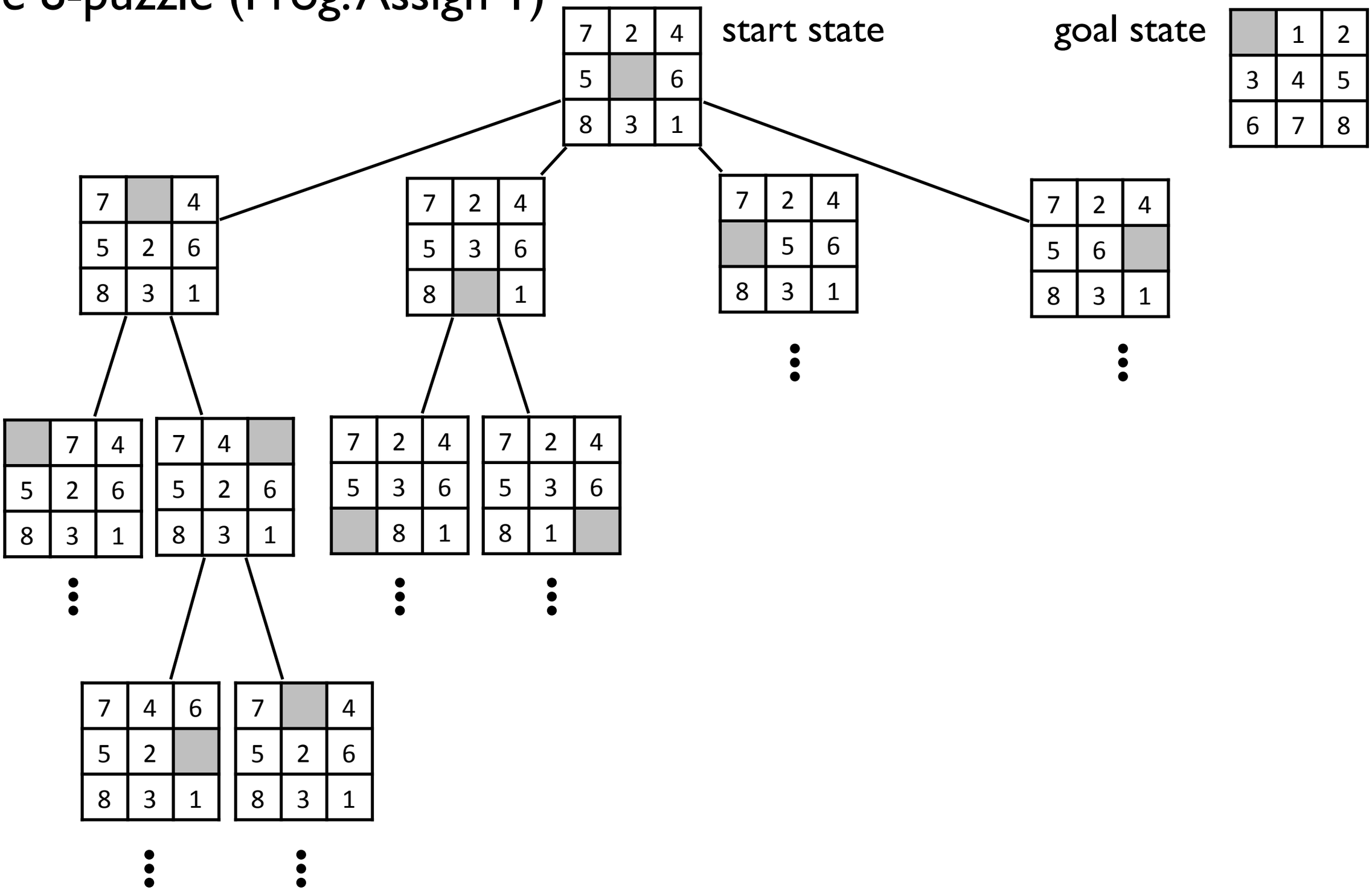
General case: Greedy best-first search

- nodes are selected based on evaluation function f(n) = h(n),
  i.e. expand node that is closest (by the heuristic) to the goal
- h(n) only depends on state at node n, not path
- h(n = goal) = 0
- implementation is same as uniform cost search, using priority queue
- Complexity:
  - worst case? $O(b^m)$ b=branching factor, m=max depth
  - best case? O(m) depends on heuristic
- Complete? No. Like DFS, can go down infinite path.
- Optimal? No.

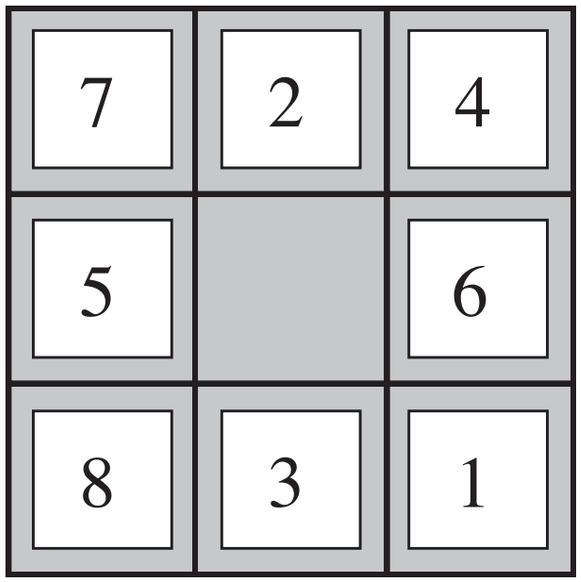
The 8-puzzle (Prog. Assign 1)

start state

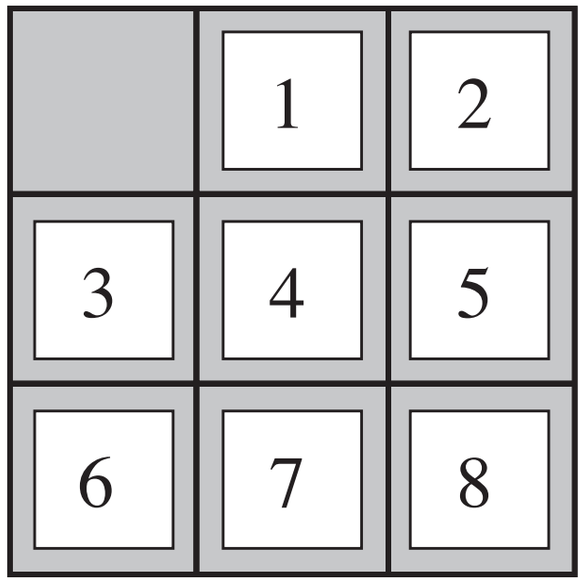
goal state



Heuristic functions



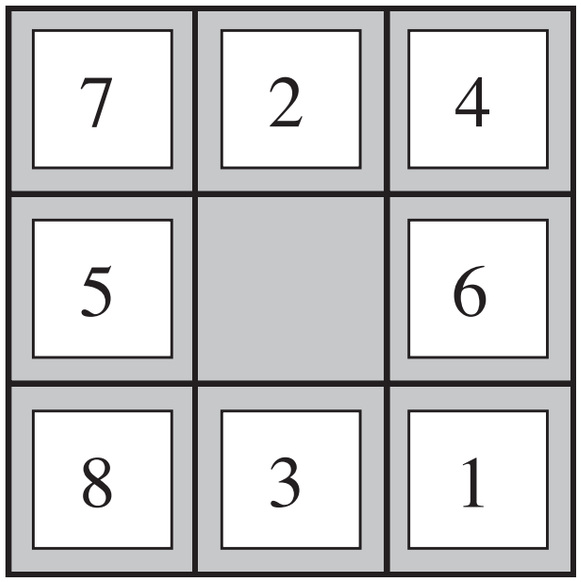
Start State



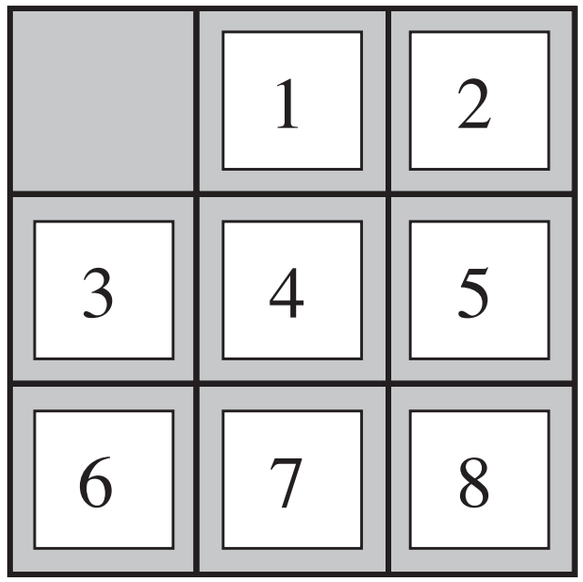
Goal State

- Typical start and goal state for 8-puzzle
- Solution is 26 steps long
- Typical solutions:
  - branching factor is 3
  - average depth for random start is 22
- How many states?

Heuristic functions



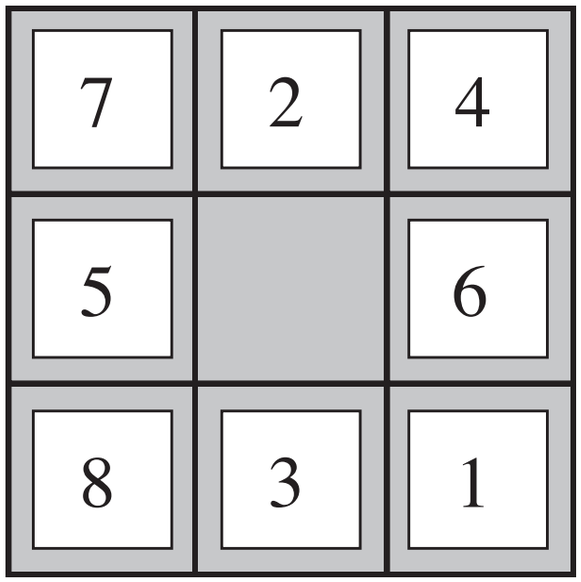
Start State



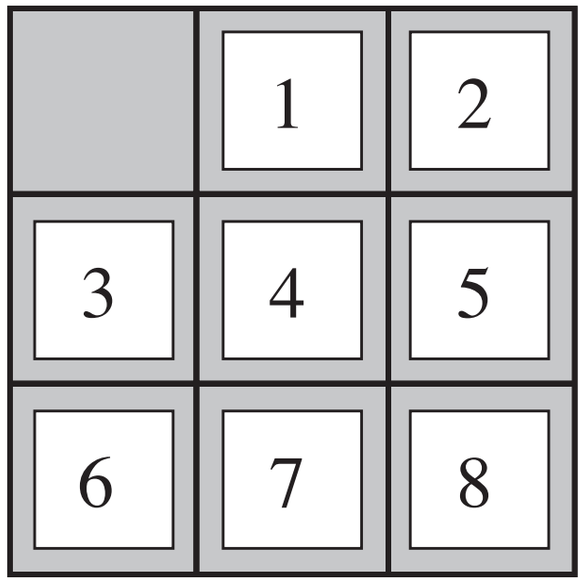
Goal State

- Typical start and goal state for 8-puzzle
- Solution is 26 steps long
- Typical solutions:
  - branching factor is 3
  - average depth for random start is 22
- What heuristic to use?

Heuristic functions



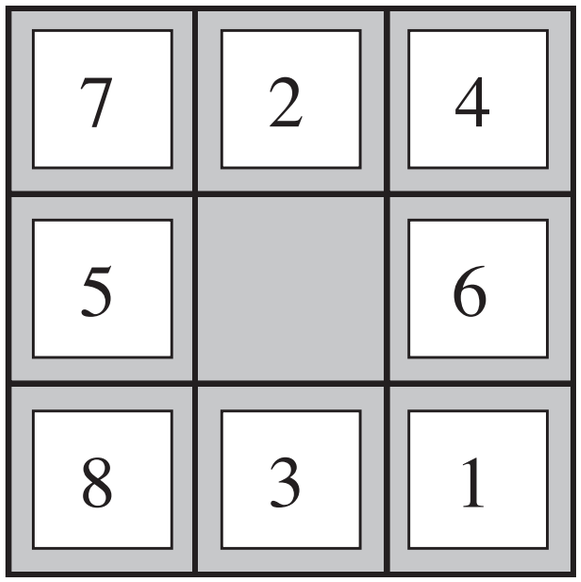
Start State



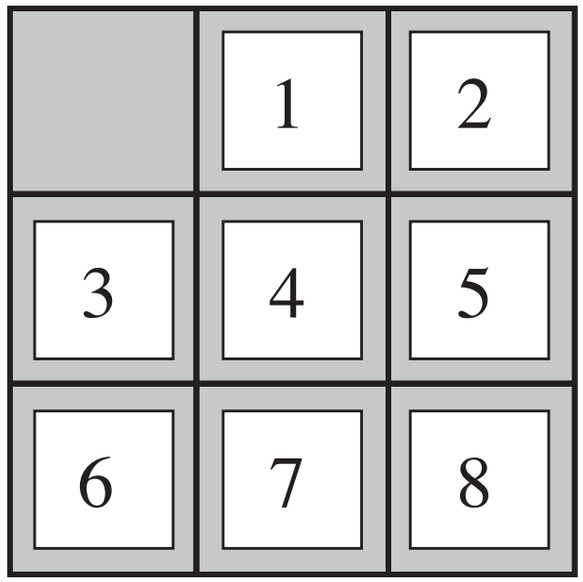
Goal State

- h1 = number of misplaced tiles
  - for start state above h1(n) = 8

Heuristic functions



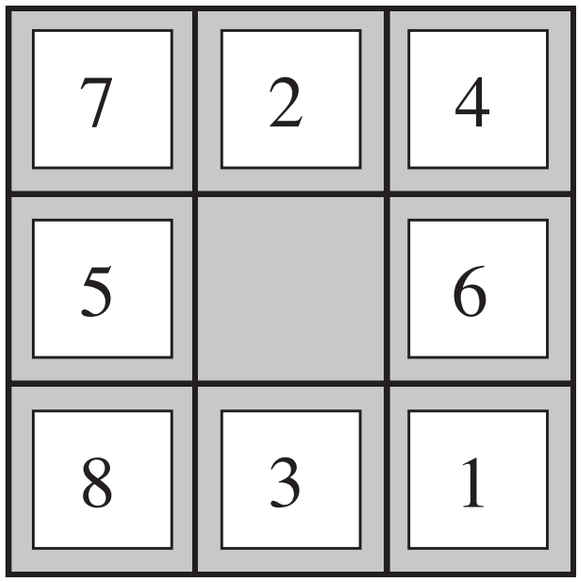
Start State



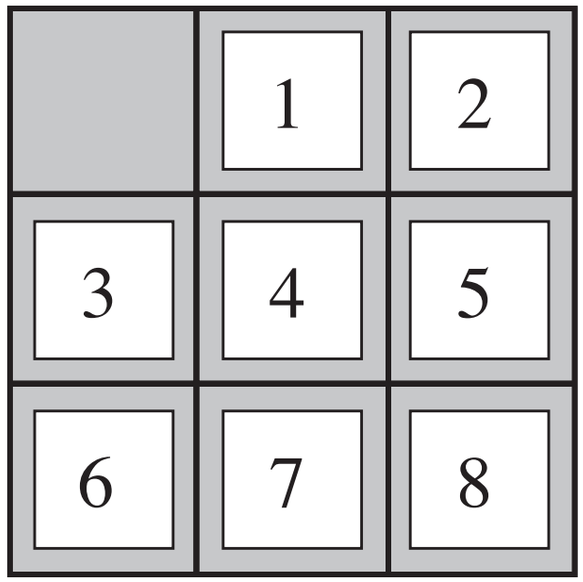
Goal State

- h1 = number of misplaced tiles
  - for start state above h1(n) = 8

Heuristic functions



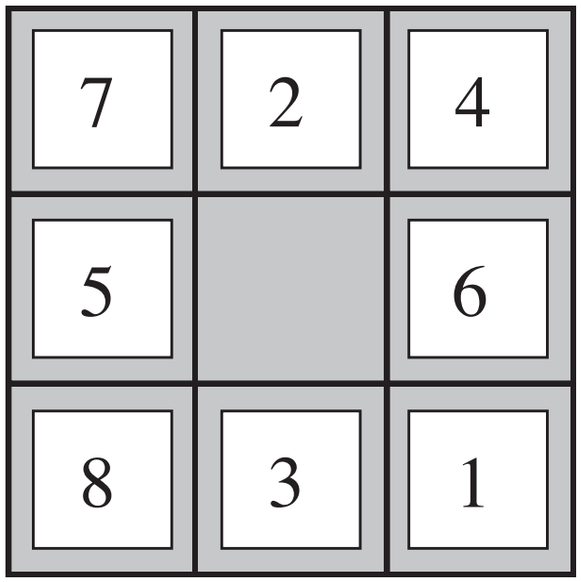
Start State



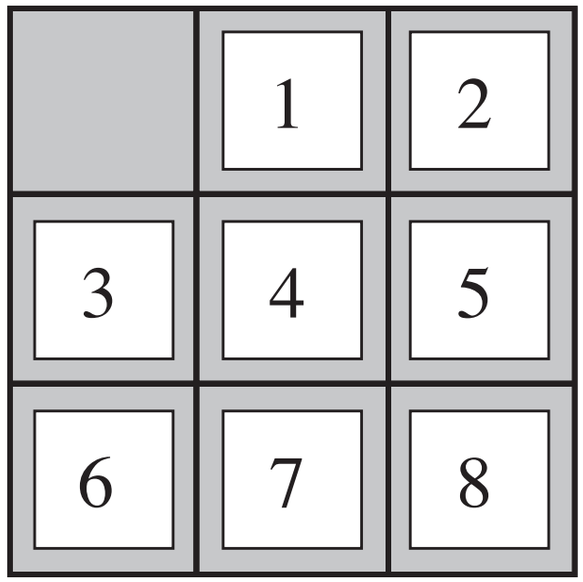
Goal State

- h1 = number of misplaced tiles
  - for start state above h1(n) = 8
- h2 = sum of the distances of the tiles from their goal positions (aka Manhattan)
  - h2(n) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18

Heuristic functions



Start State



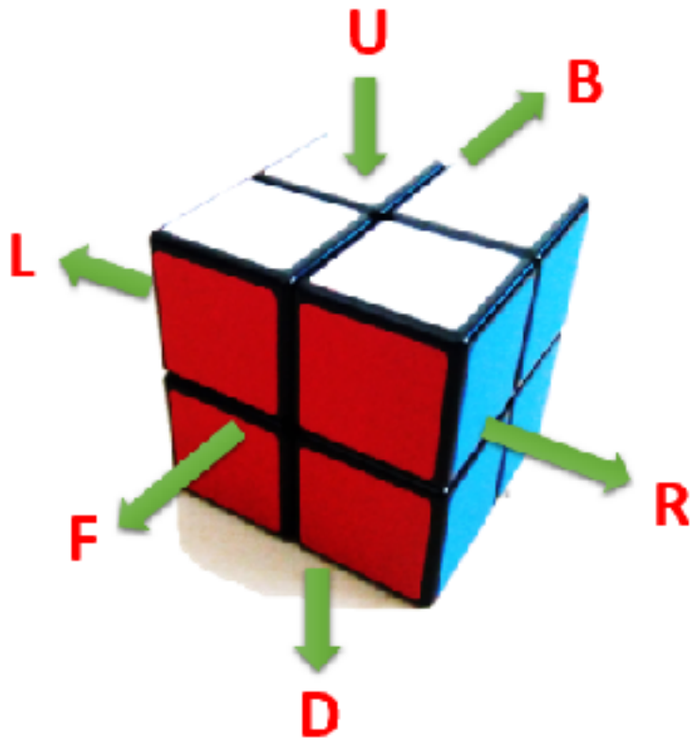
Goal State

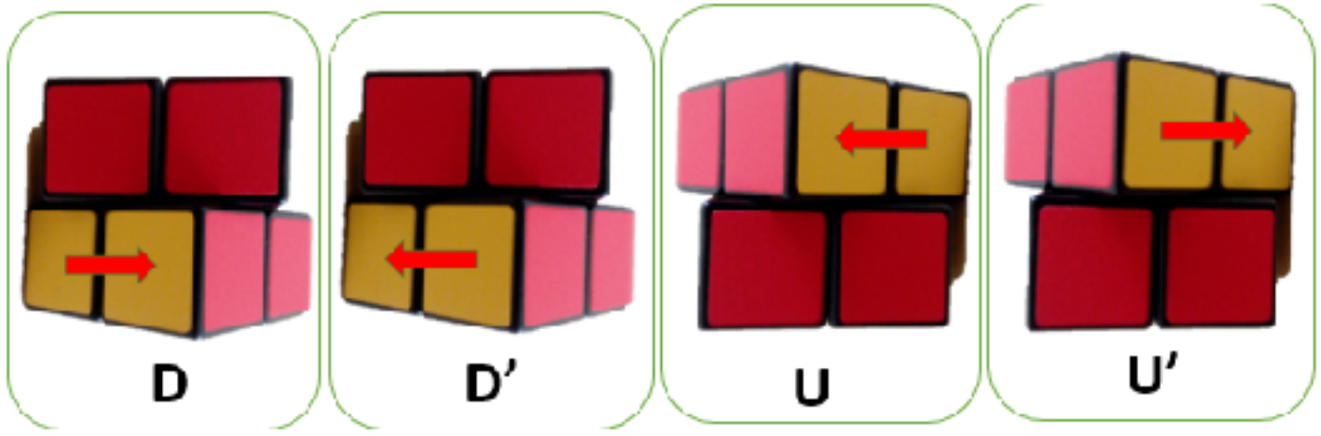
- h1 = number of misplaced tiles
  - for start state above h1(n) = 8
- h2 = sum of the distances of the tiles from their goal positions (aka Manhattan)
  - h2(n) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18

Extra Credit: Solve a 2x2(x2) Rubik's cube

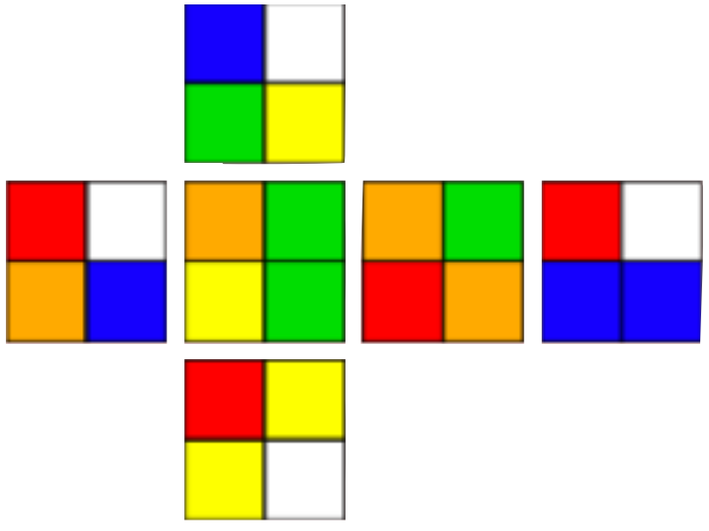
- search space is much smaller than a 3x3 Rubik's cube
- should re-use your existing search algorithms
- use standard notation and colors for the faces (there are numerous guides online)
- For example, moves (or turns) are:
  - R: rotate right face clockwise
  - R' ("R-prime"): rotate right face counter-clockwise
  - D: rotate the down face clockwise
  - *Note*: turns are from the perspective of looking *directly at the front* of the face

cube face names





flat scramble representation



Designing heuristics

- How do you come up with good heuristics?
- General strategy: h is accurate for a *relaxed* version of the problem
  - state graph of relaxed problem is a *super graph* of original
  - restrictions remove edges, i.e. actions
  - relaxing adds edges (and makes problem easier)

- Heuristics are characterized by their *effective branching factor*
  - N = total nodes generated during A* search
  - d = solution depth
  - b* = branching factor needed for a uniform tree to contain N+1 nodes

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