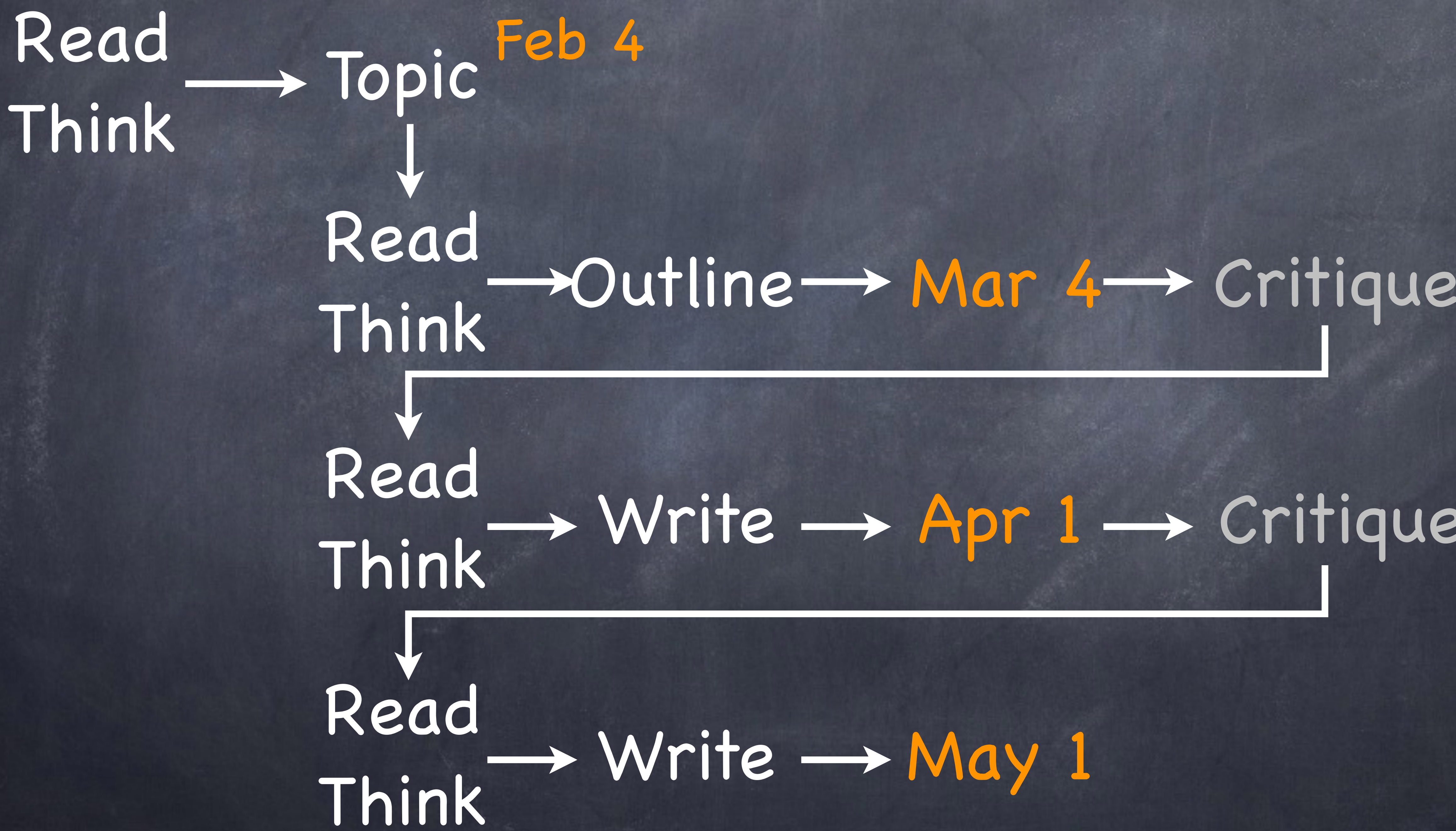
CSC242: Artificial Intelligence

Lecture 2: Problem Solving

Upper Level Writing

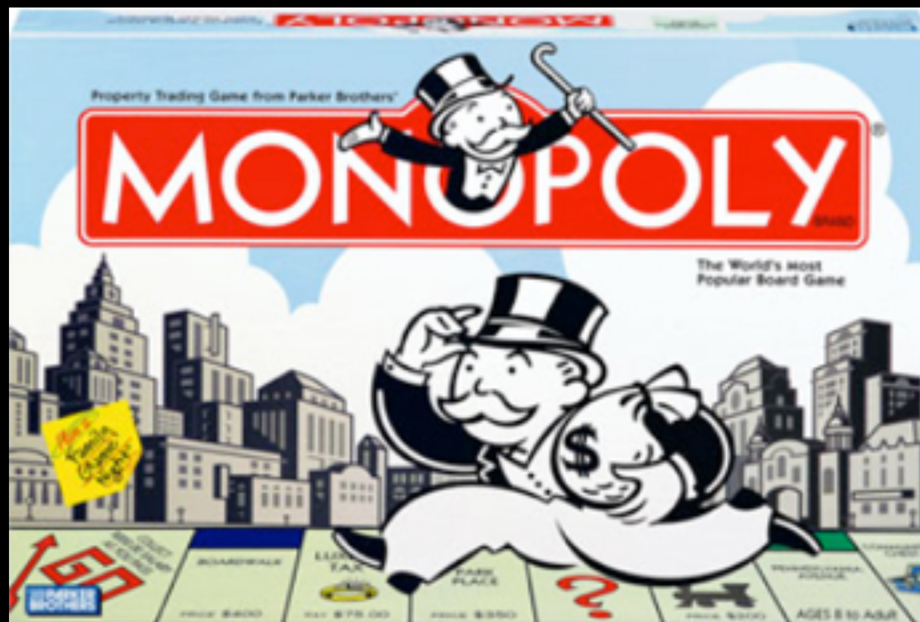


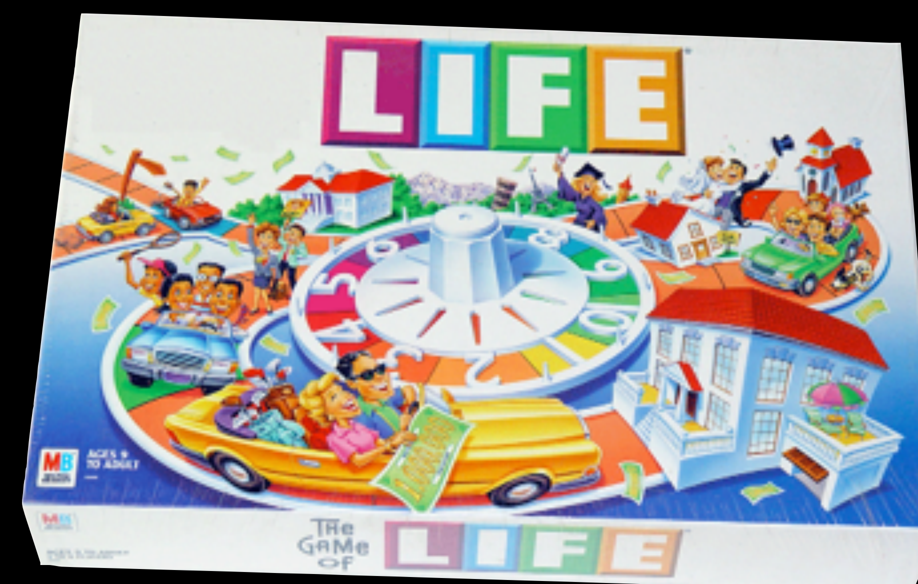
One More Policy

One More Policy

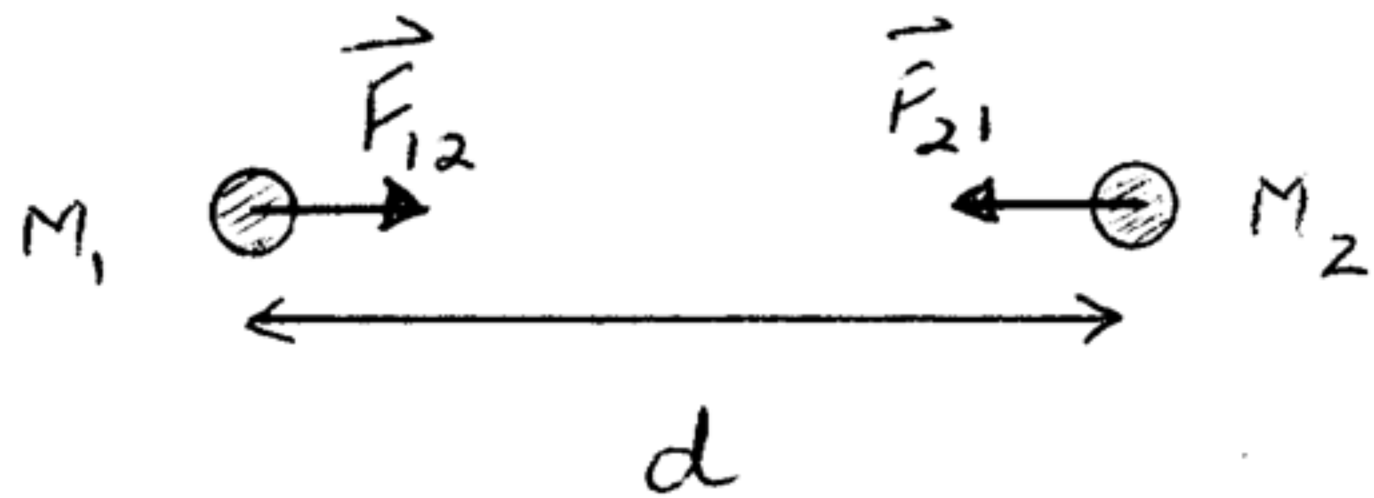
General Problem Solving


WD-40


Property Trading Game from Parker Brothers
MONOPOLY
The World's Most Popular Board Game


LIFE
THE GAME OF LIFE


OPERATION
SKILL GAME
WHERE YOU'RE THE DOCTOR
SPECIALIST



$$|\vec{F}_{12}| = |\vec{F}_{21}| = \frac{G M_1 M_2}{d^2}$$

Cooperative Problem Solving

Cooperative Problem Solving

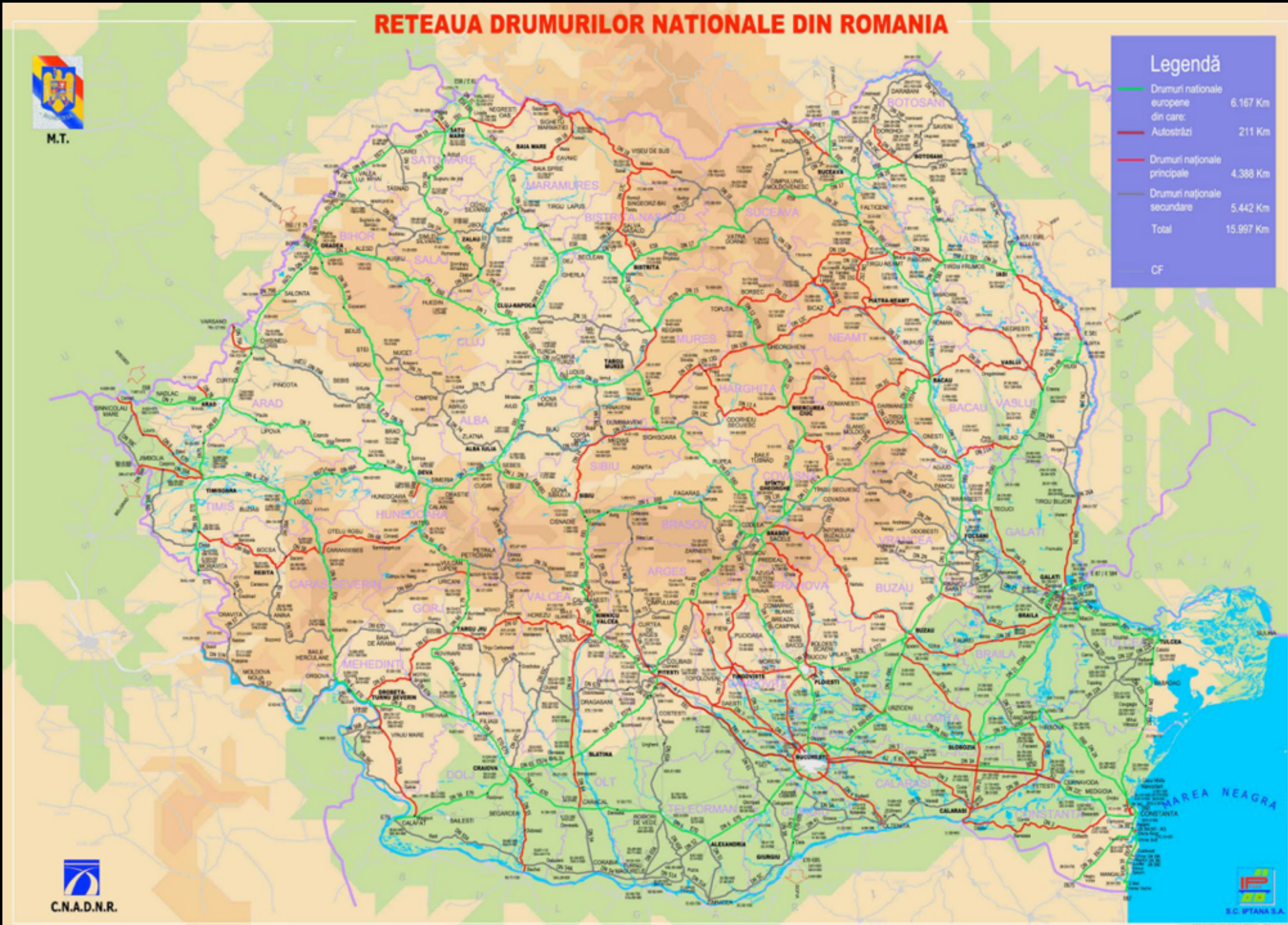
Cooperative Problem Solving

CSC242

We don't make the computers.*
We make the computers solve problems.

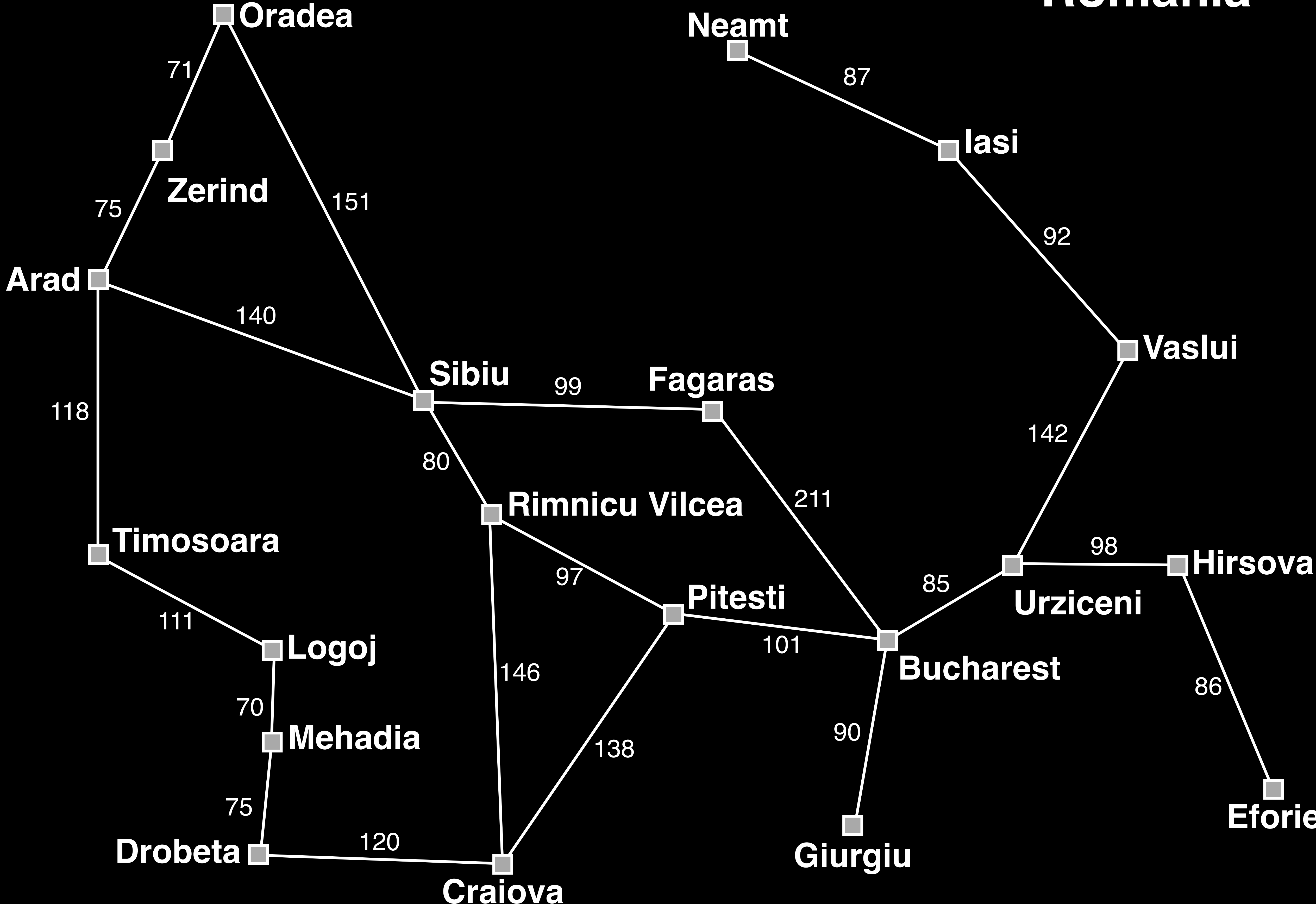
*Or the programming languages, compilers, debuggers, databases, graphics pipelines, network protocols, web servers, ...

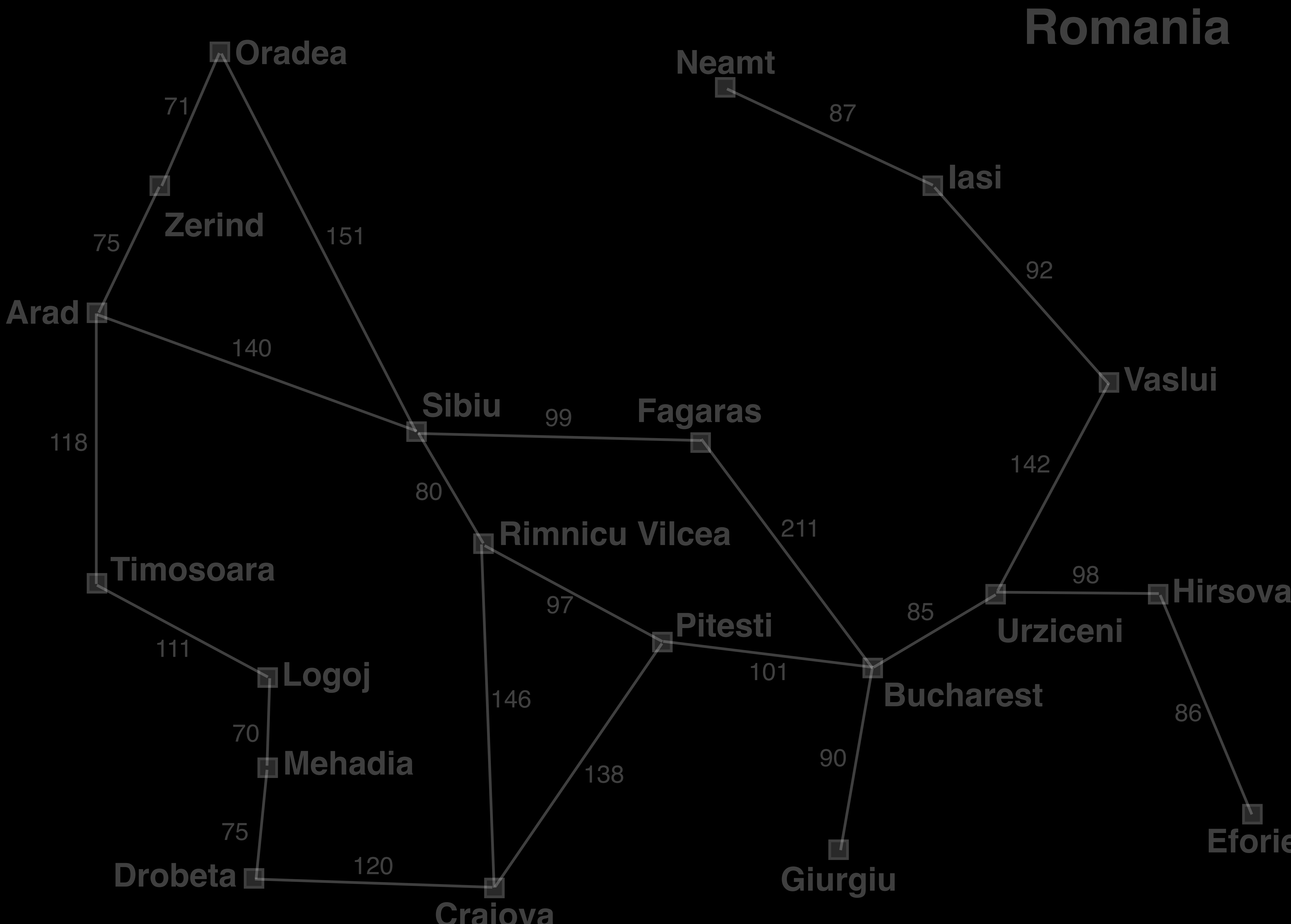
Our first problem



RETEAUA DRUMURILOR NATIONALE DIN ROMANIA
M.T.
Legendă
Drumuri nationale europene 6.167 Km
din care:
Autostrăzi 211 Km
Drumuri naționale principale 4.388 Km
Drumuri naționale secundare 5.442 Km
Total 15.997 Km
CF
MAREA NEAGRA
C.N.A.D.N.R.
S.C. IPTANA S.A.

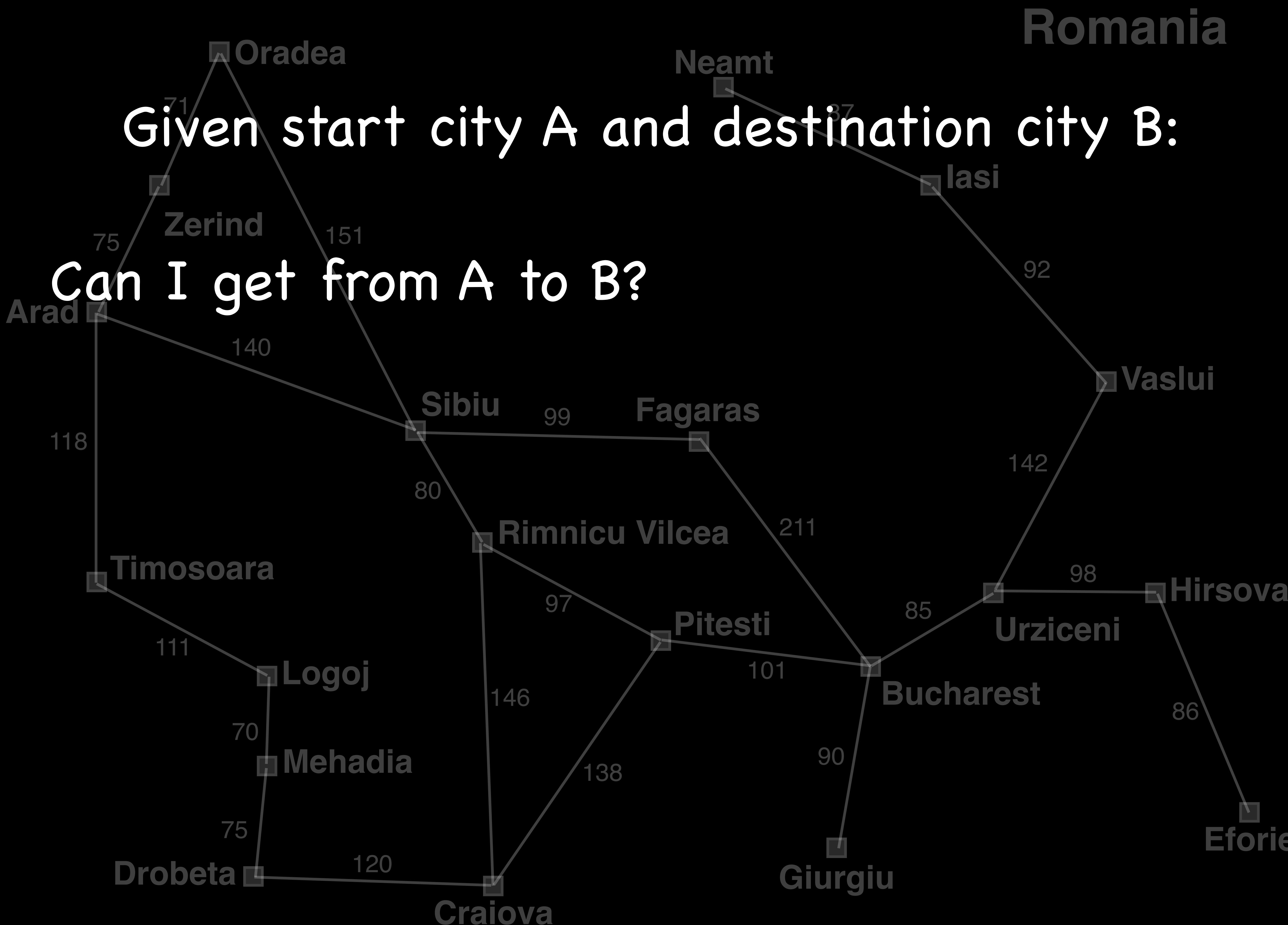
Romania


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


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

Given start city A and destination city B:

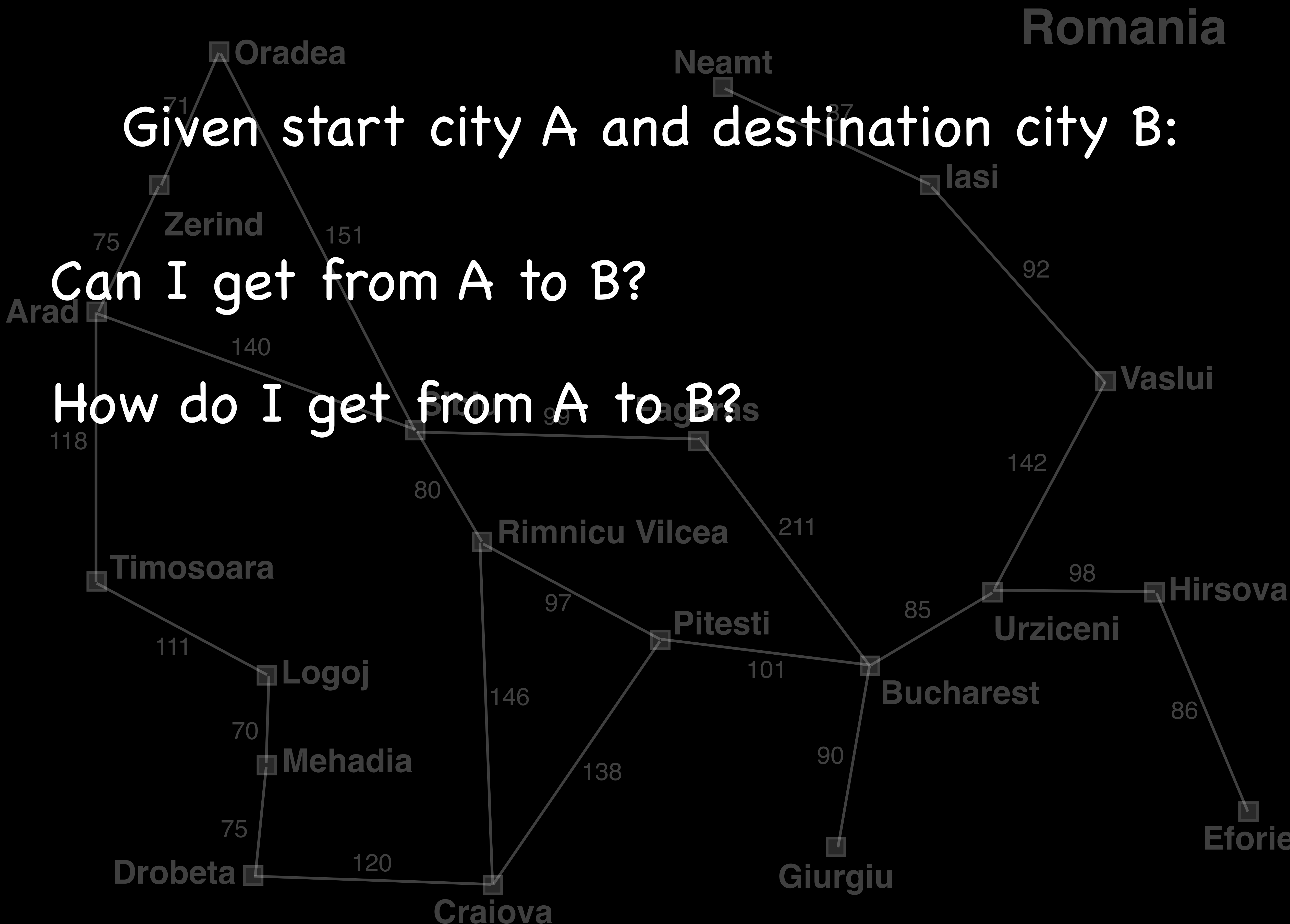
Can I get from A to B?



Given start city A and destination city B:

Can I get from A to B?

How do I get from A to B?



Given start city A and destination city B:

Can I get from A to B?

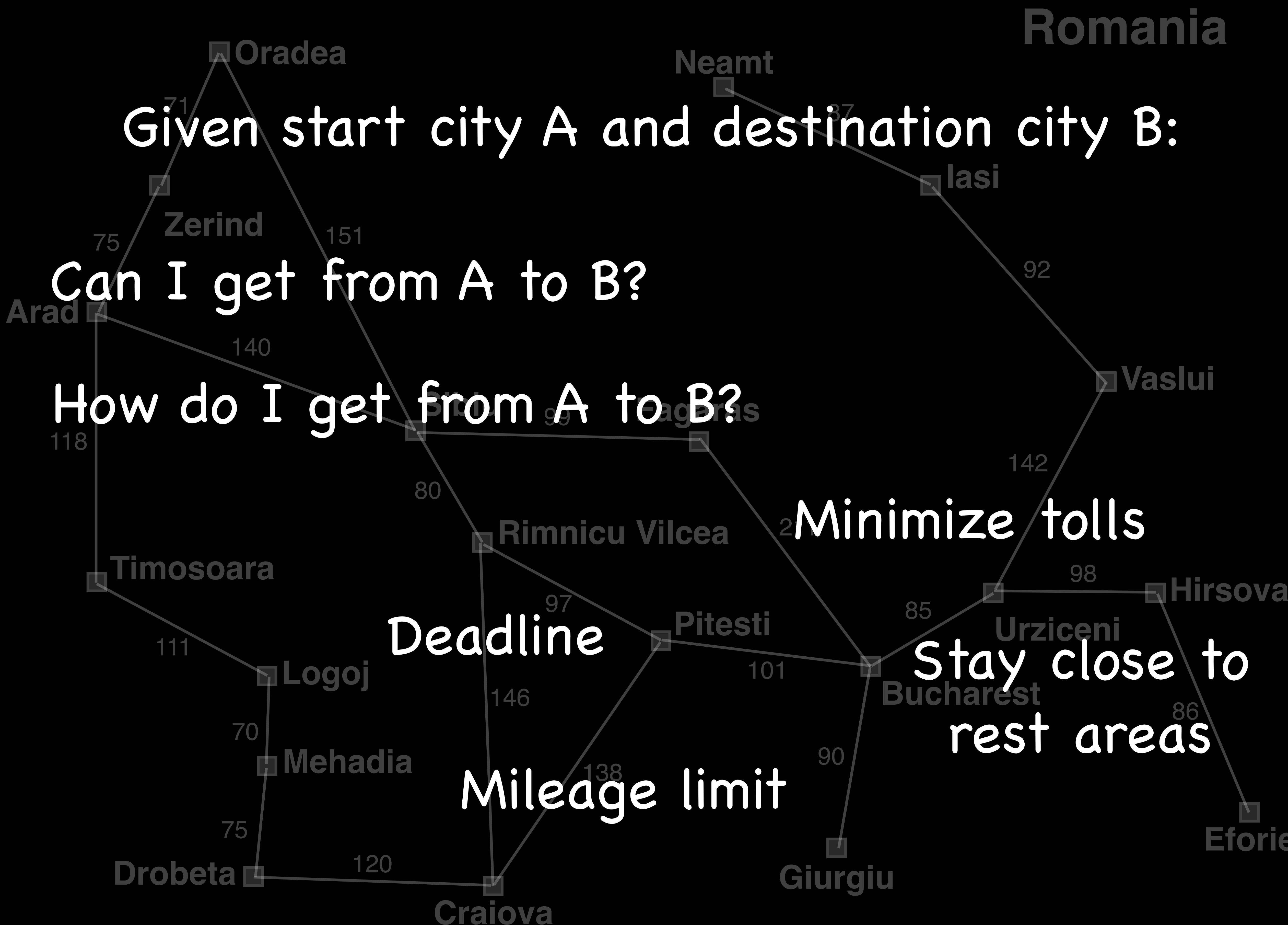
How do I get from A to B?

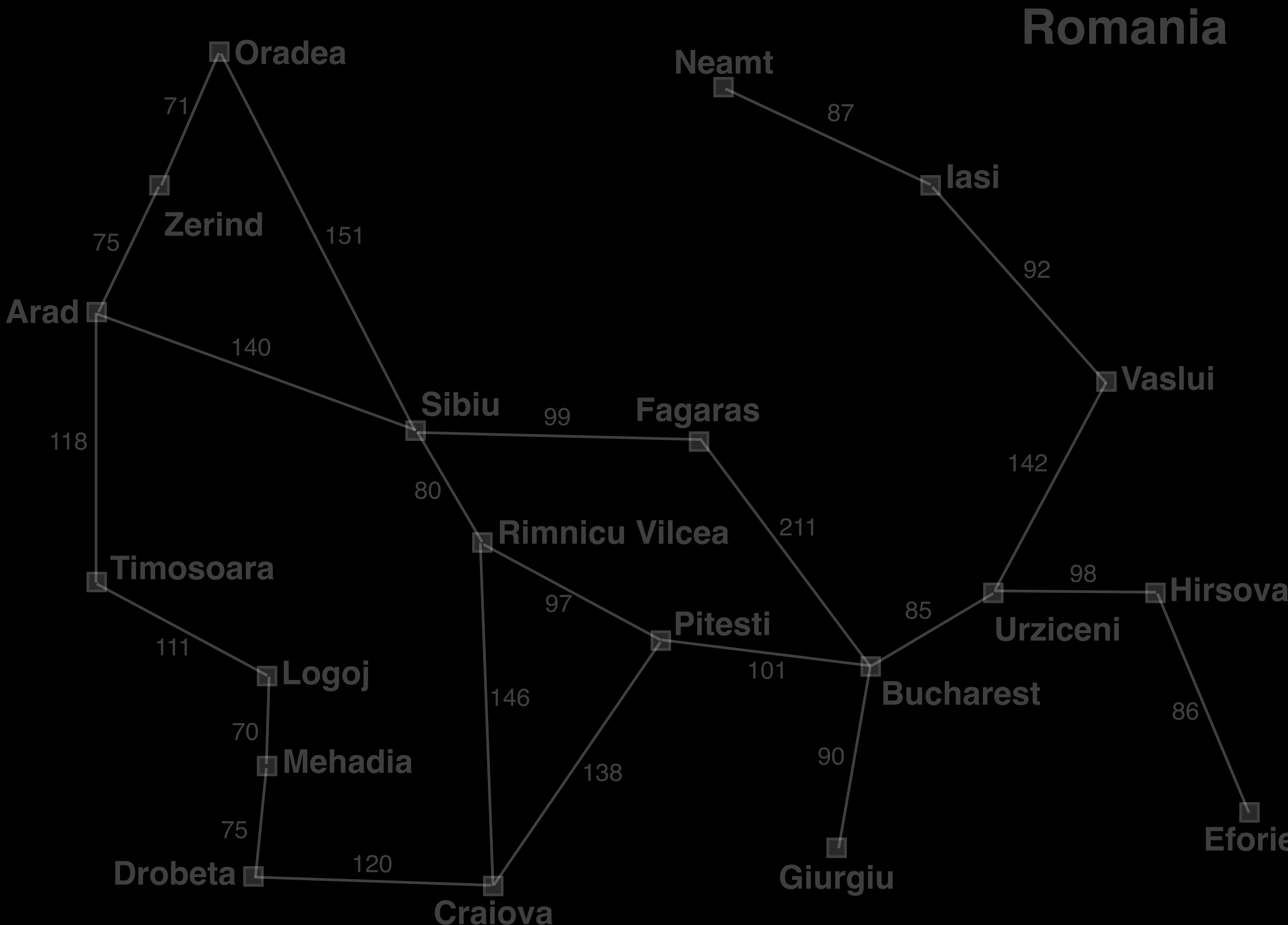
Minimize tolls

Deadline

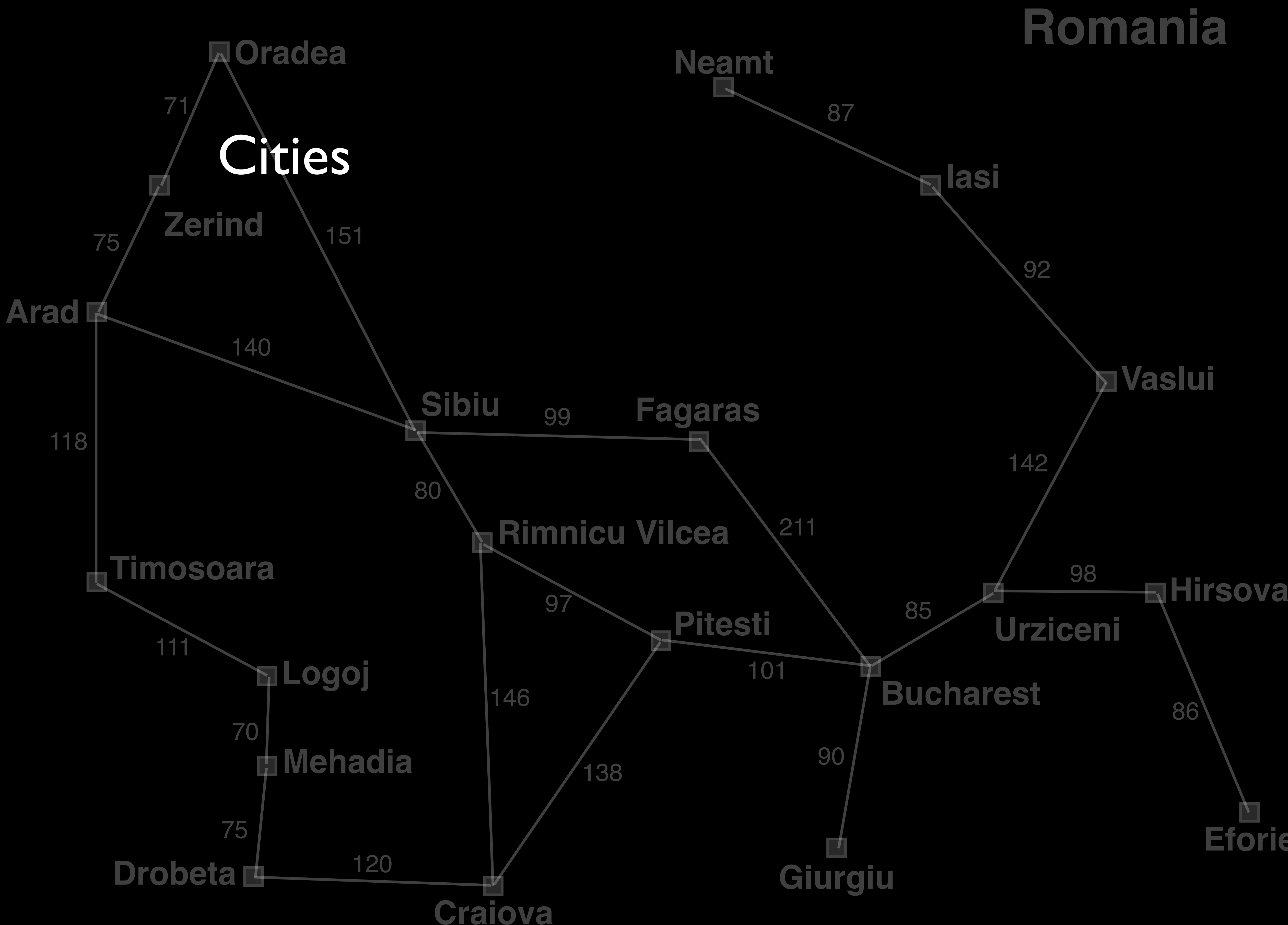
Stay close to rest areas

Mileage limit




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

Cities



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

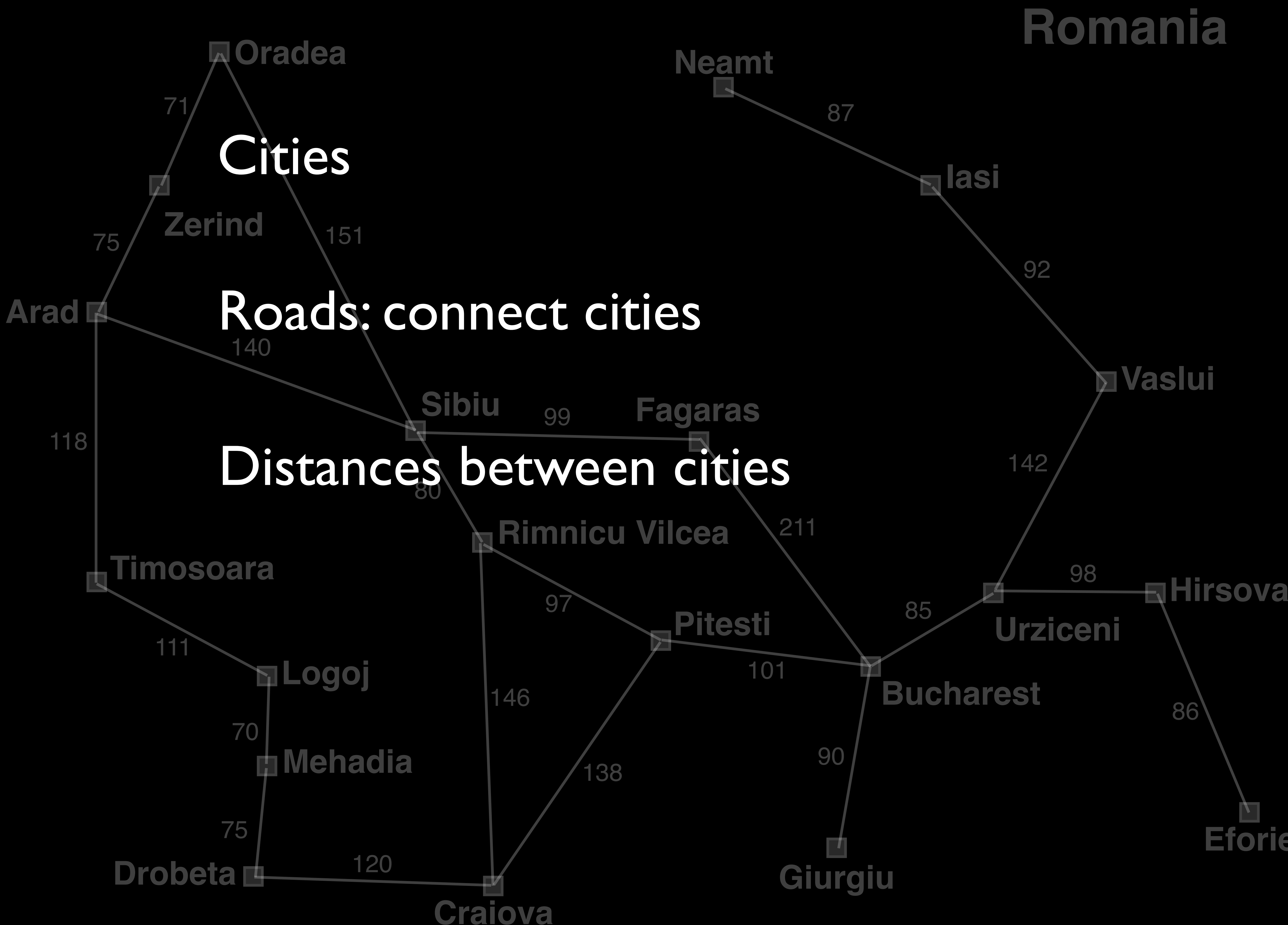
Cities

Roads: connect cities

Cities

Roads: connect cities

Distances between cities



Cities

Roads: connect cities

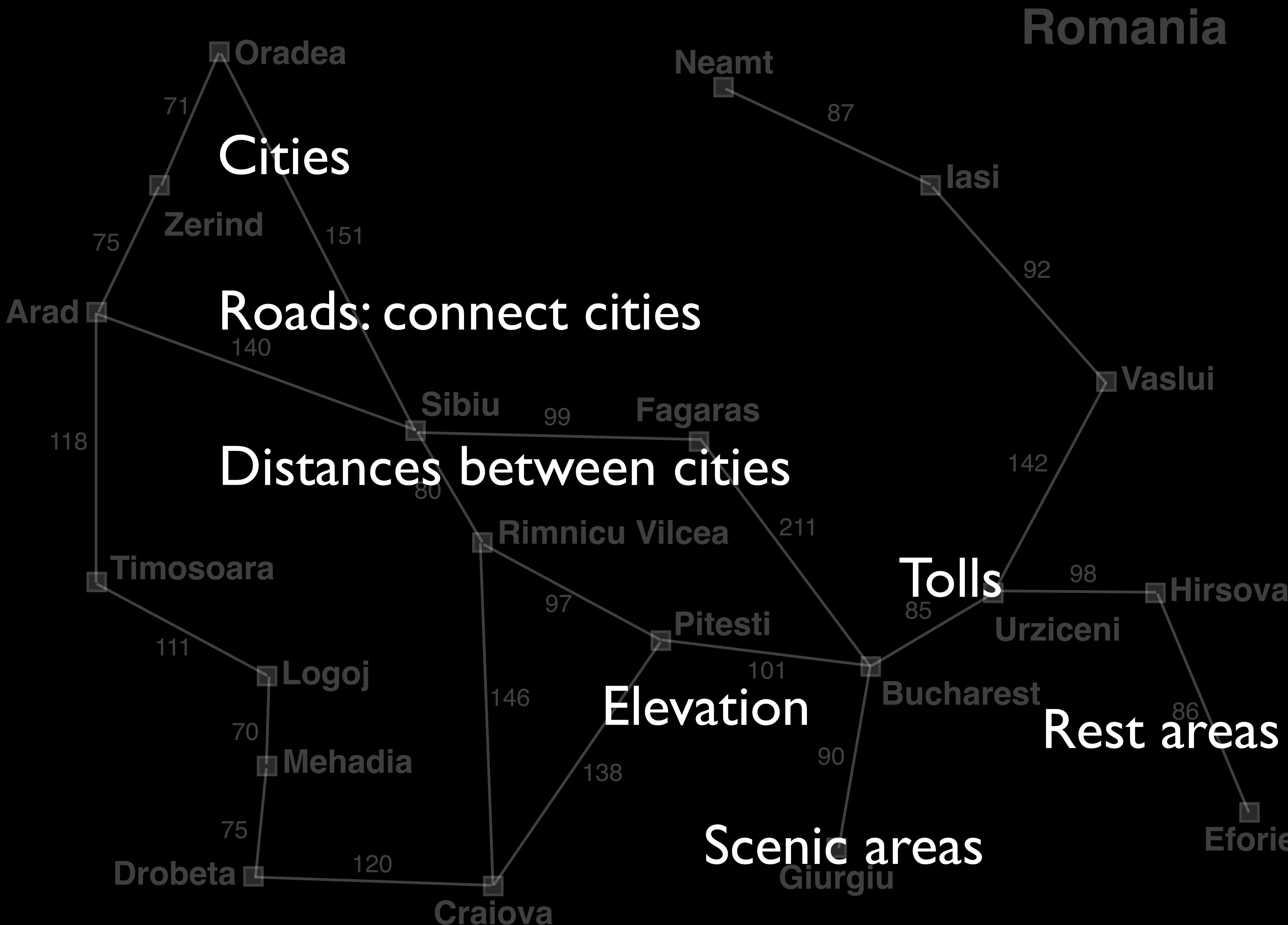
Distances between cities

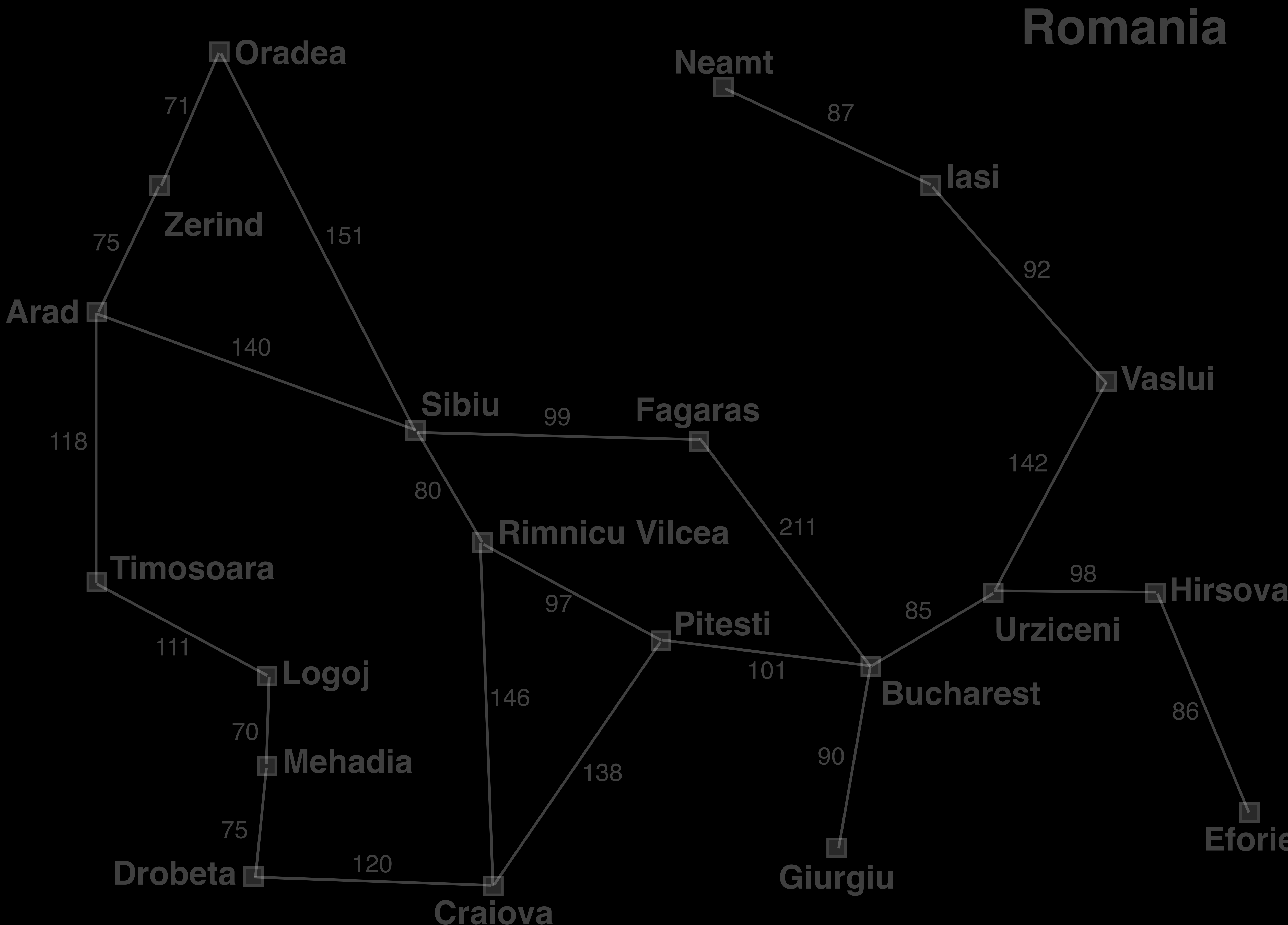
Tolls

Elevation

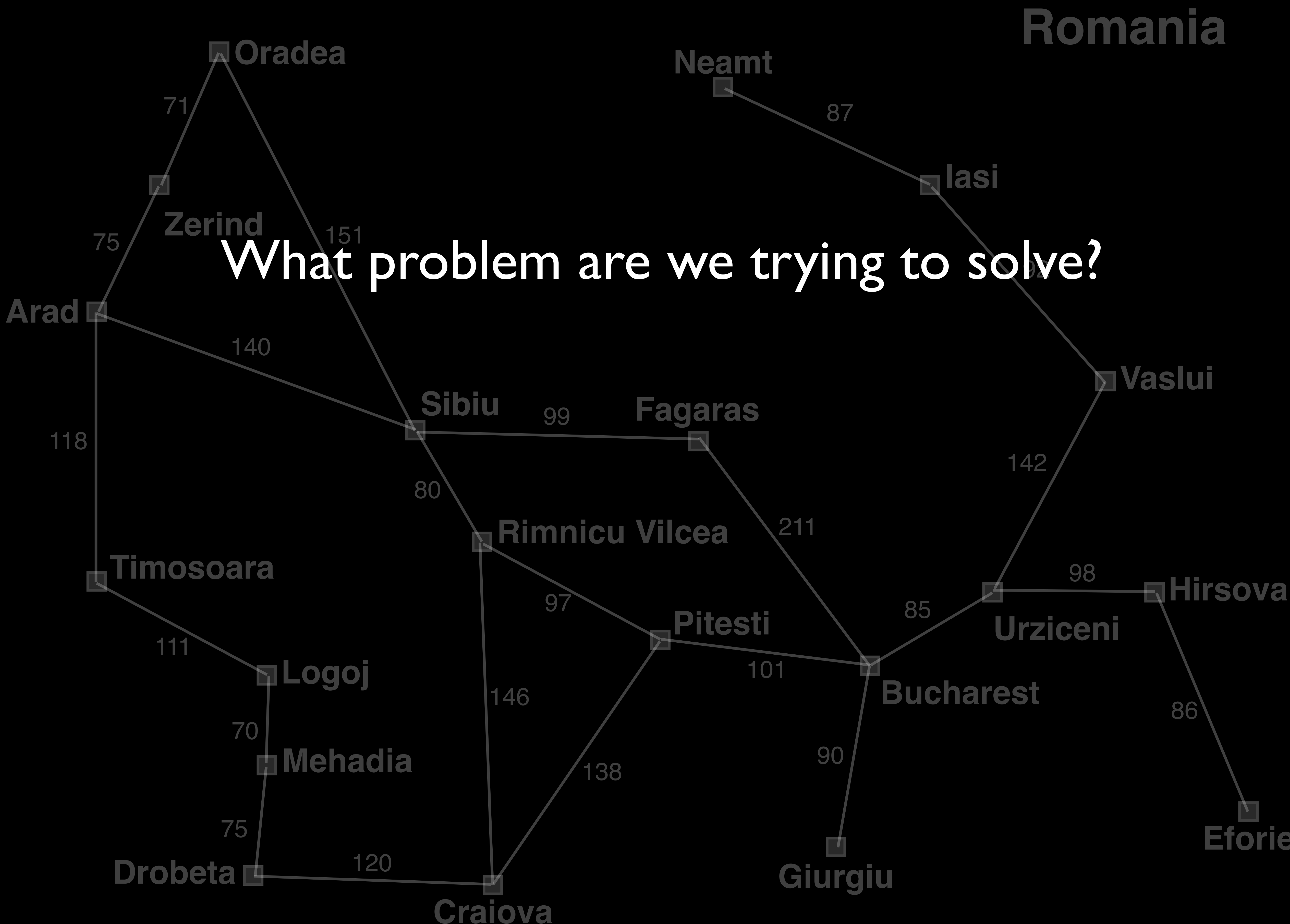
Rest areas

Scenic areas



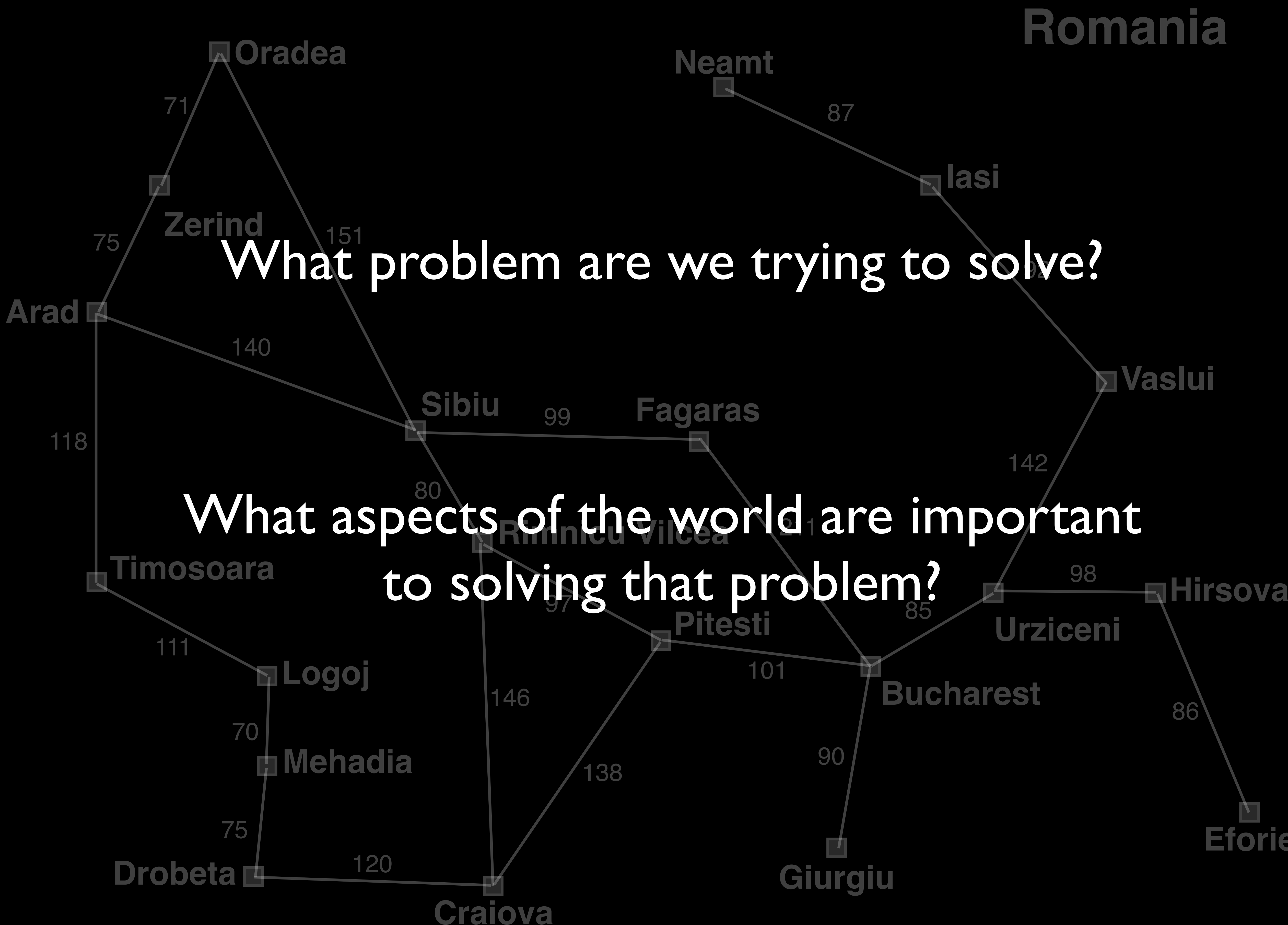

Romania
Oradea
Neamt
71
87
Zerind
Iasi
75
151
92
Arad
140
Vaslui
Sibiu
99
Fagaras
118
142
80
Rimnicu Vilcea
211
Timosoara
98
Hirsova
97
85
Pitesti
Urziceni
111
Logoj
101
Bucharest
146
70
86
Mehadia
90
138
75
Eforie
120
Giurgiu
Drobeta
Craiova

What problem are we trying to solve?

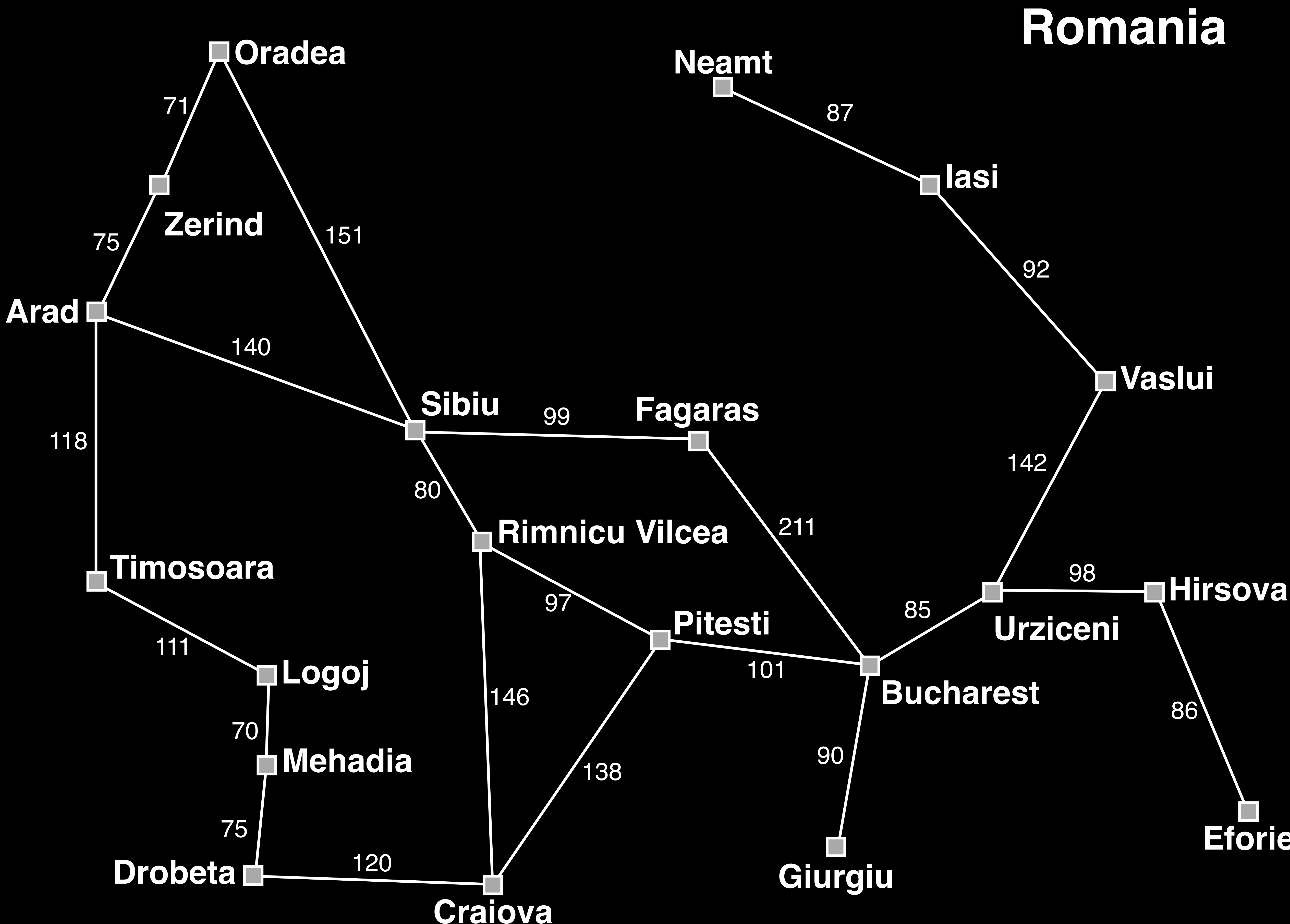


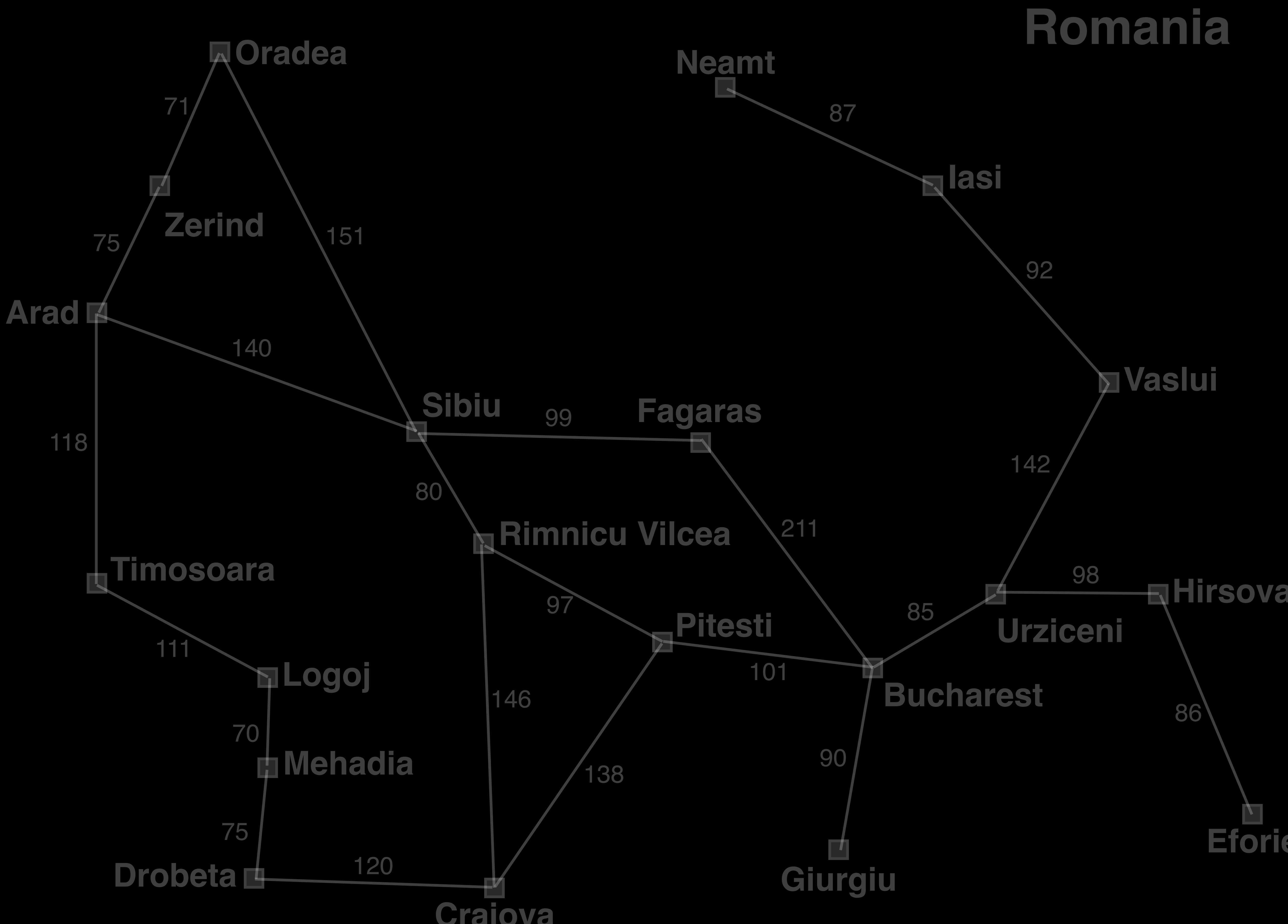
What problem are we trying to solve?

What aspects of the world are important to solving that problem?

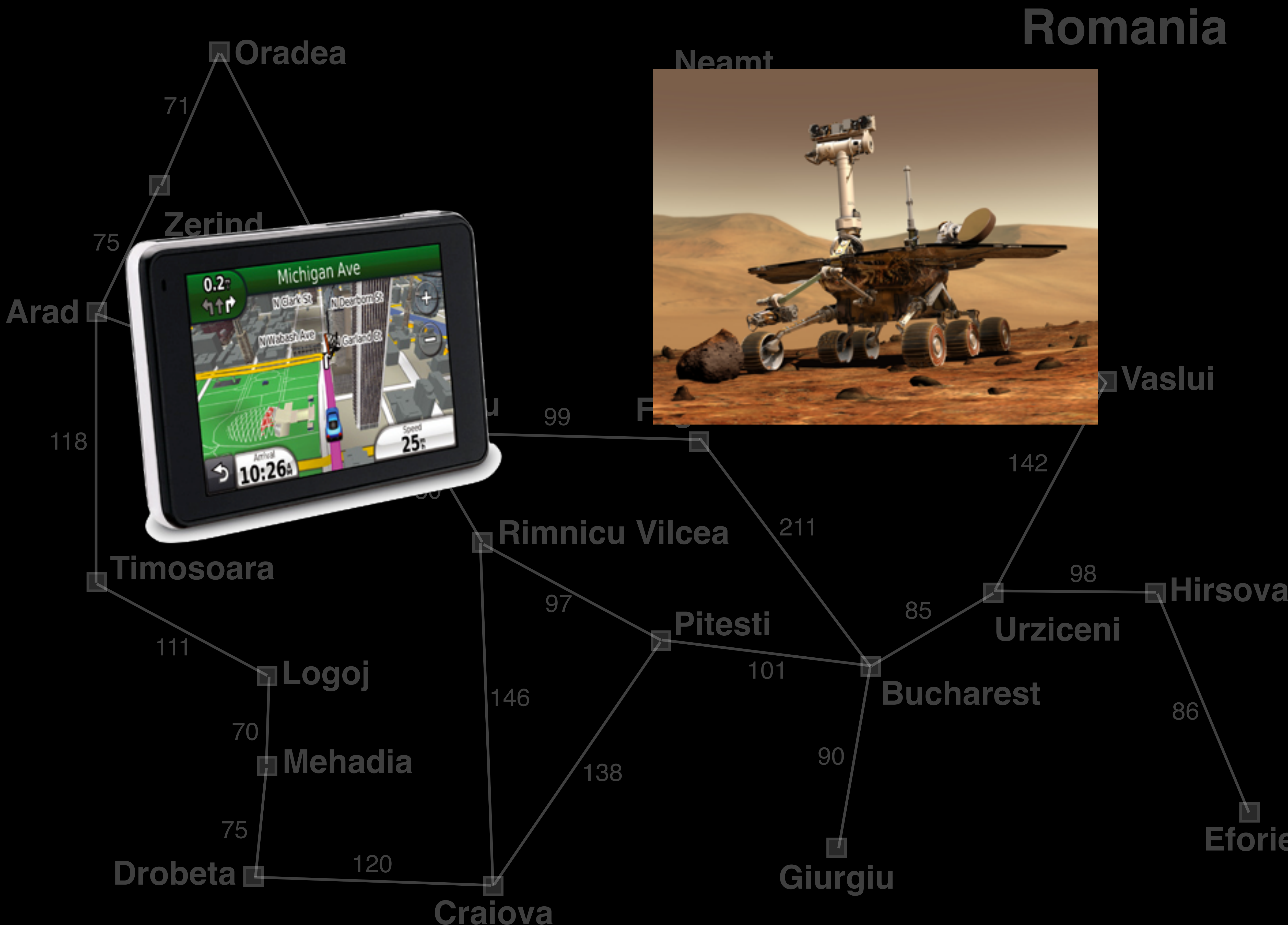


Problem Solving by Computers


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

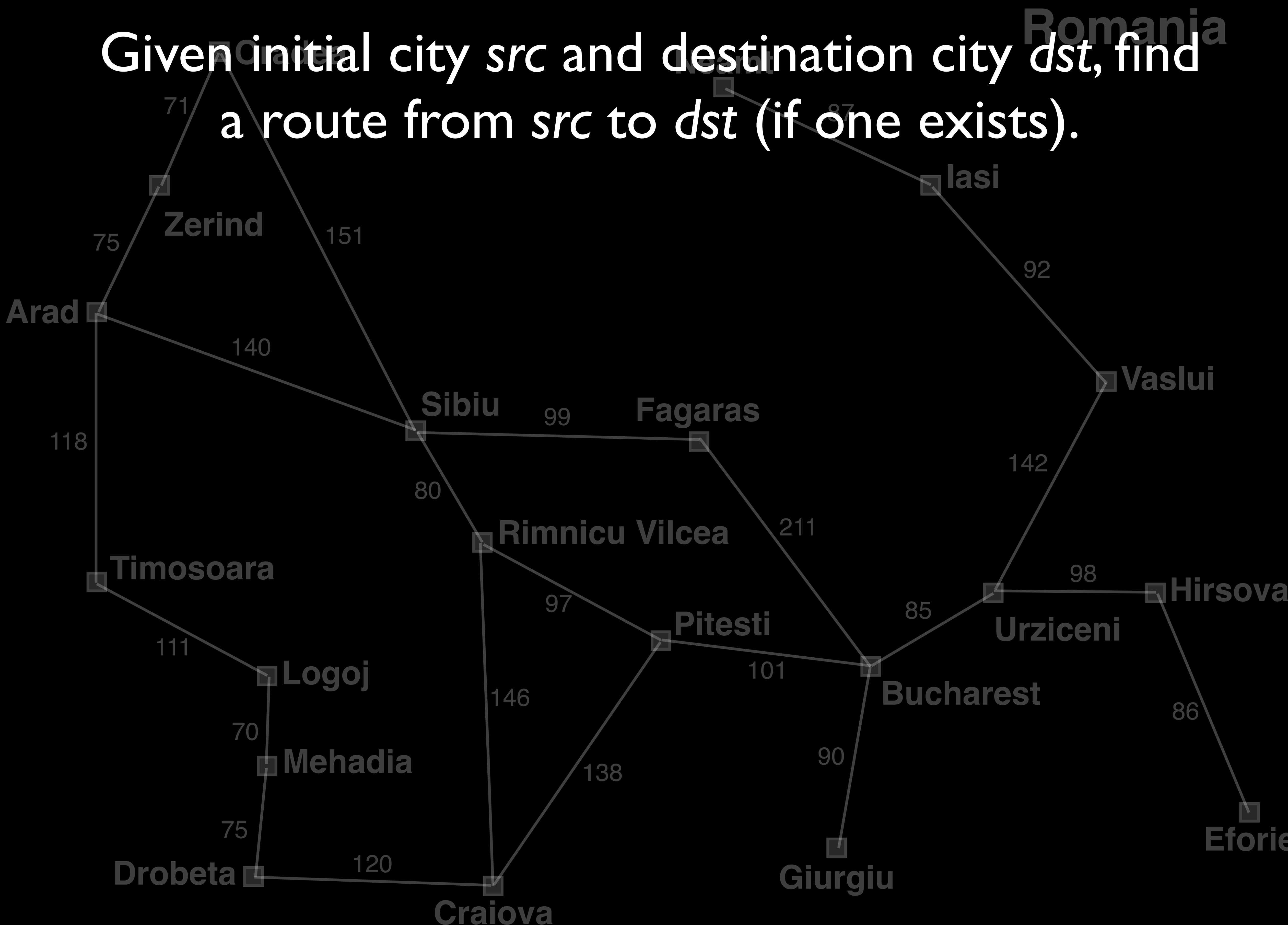

Romania
Oradea
Neamt
71
87
Zerind
Iasi
75
151
92
Arad
140
Vaslui
Sibiu
99
Fagaras
118
142
80
Rimnicu Vilcea
211
Timosoara
98
97
Hirsova
85
Pitesti
Urziceni
111
Logoj
101
Bucharest
146
86
70
Mehadia
90
138
75
Eforie
120
Drobeta
Giurgiu
Craiova

Romania
Oradea
Neamt
71
87
Zerind
Iasi
75
92
Arad
Michigan Ave
0.2
Vaslui
99
Fagaras
118
142
Arrival
10:26
Speed
25
211
Rimnicu Vilcea
Timosoara
98
Hirsova
97
85
Pitesti
Urziceni
111
Logoj
101
Bucharest
146
86
70
Mehadia
90
138
75
Drobeta
120
Eforie
Giurgiu
Craiova


Romania
Oradea
Neamt
71
Zerind
75
Arad
118
99
Vaslui
142
211
Rimnicu Vilcea
Timosoara
97
98
Hirsova
85
Urziceni
Pitesti
111
Logoj
101
Bucharest
146
70
Mehadia
86
90
138
75
Eforie
120
Drobeta
Giurgiu
Craiova
Michigan Ave
0.2
N Clark St
N Dearborn St
N Wabash Ave
Garland Ct
Speed
25
Arrival
10:26


Romania
Oradea
71
Zerind
75
Arad
118
Timosoara
111
Logoj
70
Mehadia
75
Drobeta
120
Craiova
146
99
142
Vaslui
Hirsova
86
Eforie
Michigan Ave
0.2
N Clark St
N Dearborn St
N Wabash Ave
Garland Ct
Arrival
10:26
Speed
25
03
MDV
android

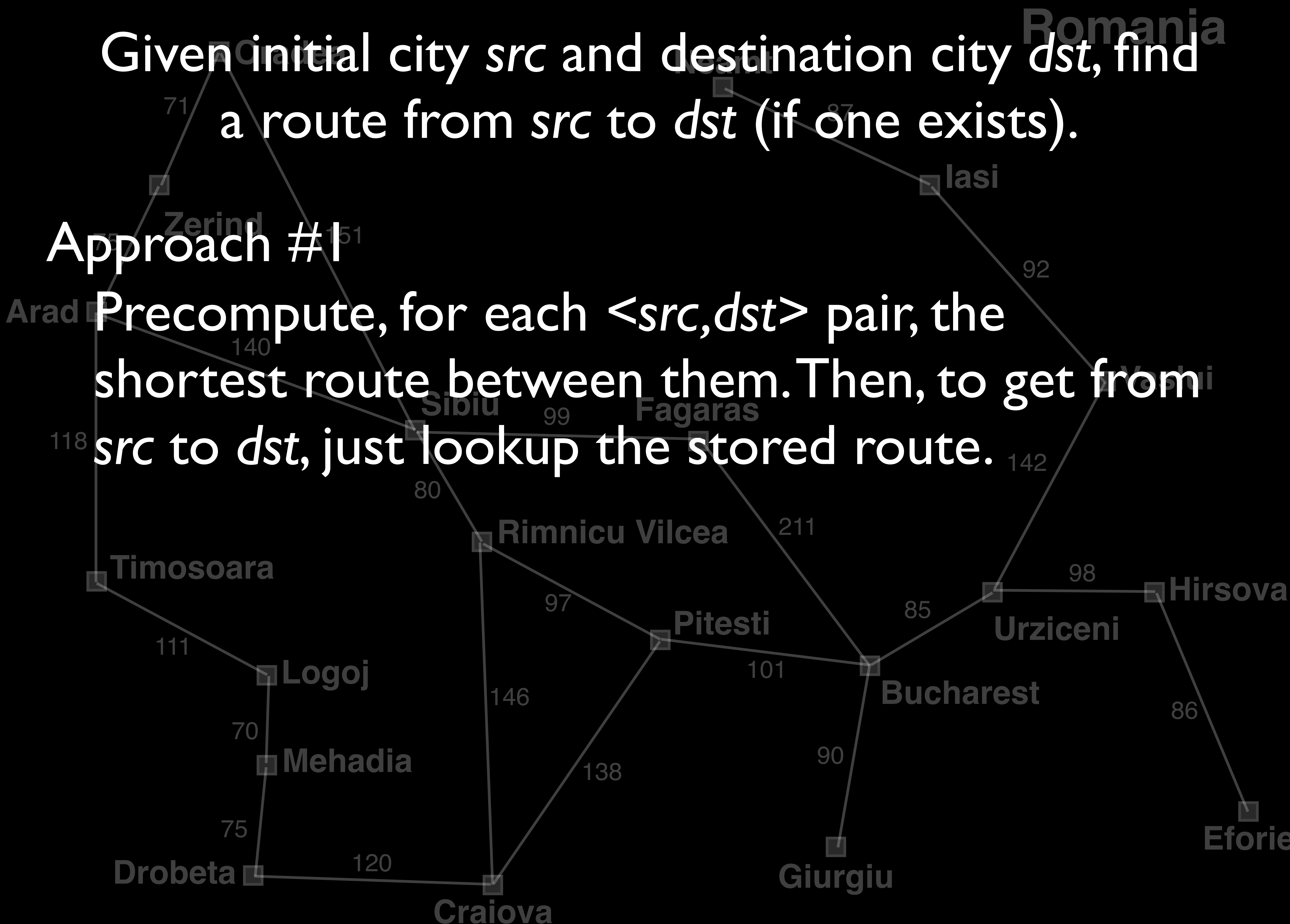
Given initial city *src* and destination city *dst*, find a route from *src* to *dst* (if one exists).



Given initial city *src* and destination city *dst*, find a route from *src* to *dst* (if one exists).

Approach #1

Precompute, for each <*src,dst*> pair, the shortest route between them. Then, to get from *src* to *dst*, just lookup the stored route.

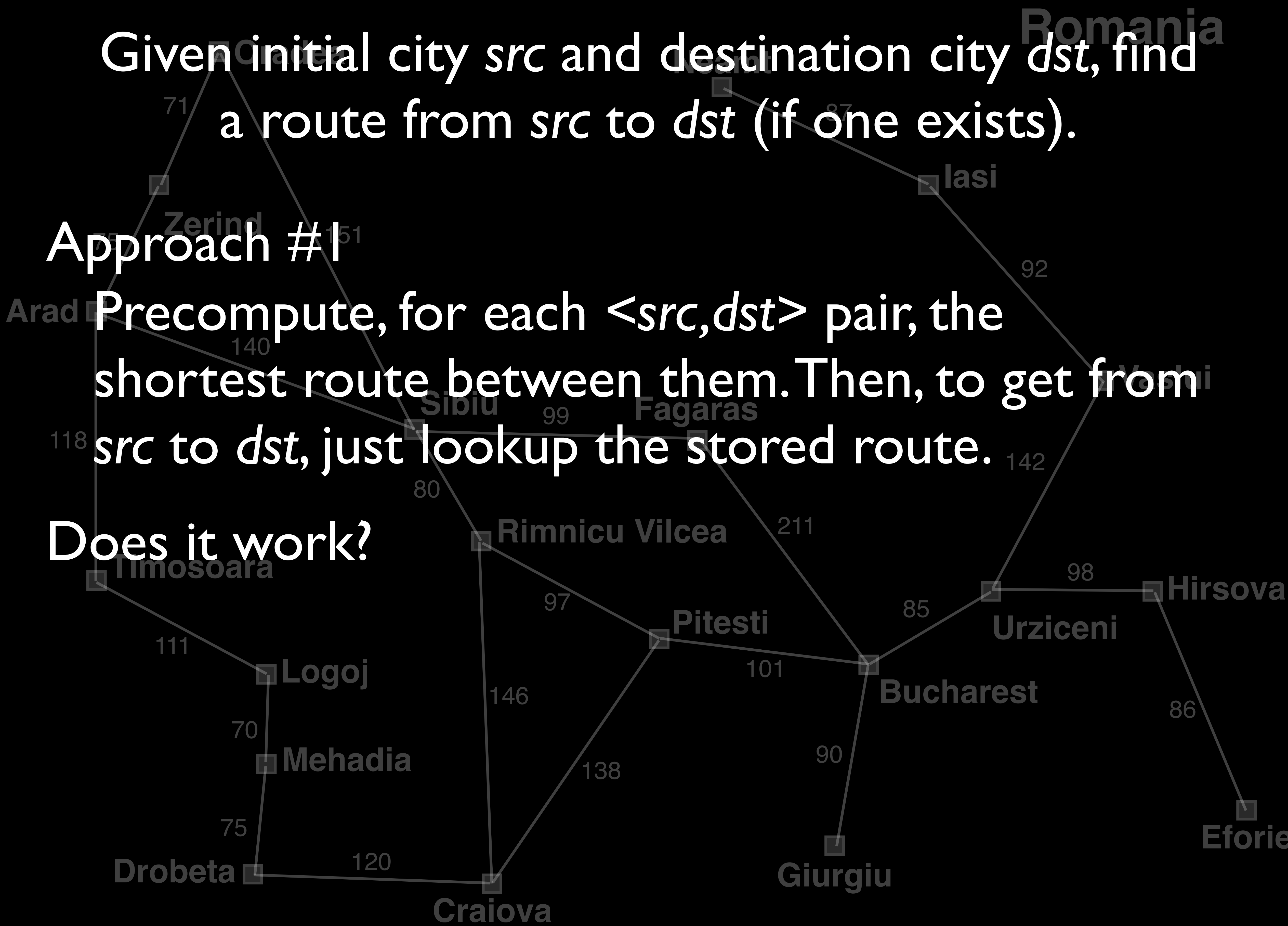


Given initial city *src* and destination city *dst*, find a route from *src* to *dst* (if one exists).

Approach #1

Precompute, for each <*src*,*dst*> pair, the shortest route between them. Then, to get from *src* to *dst*, just lookup the stored route.

Does it work?

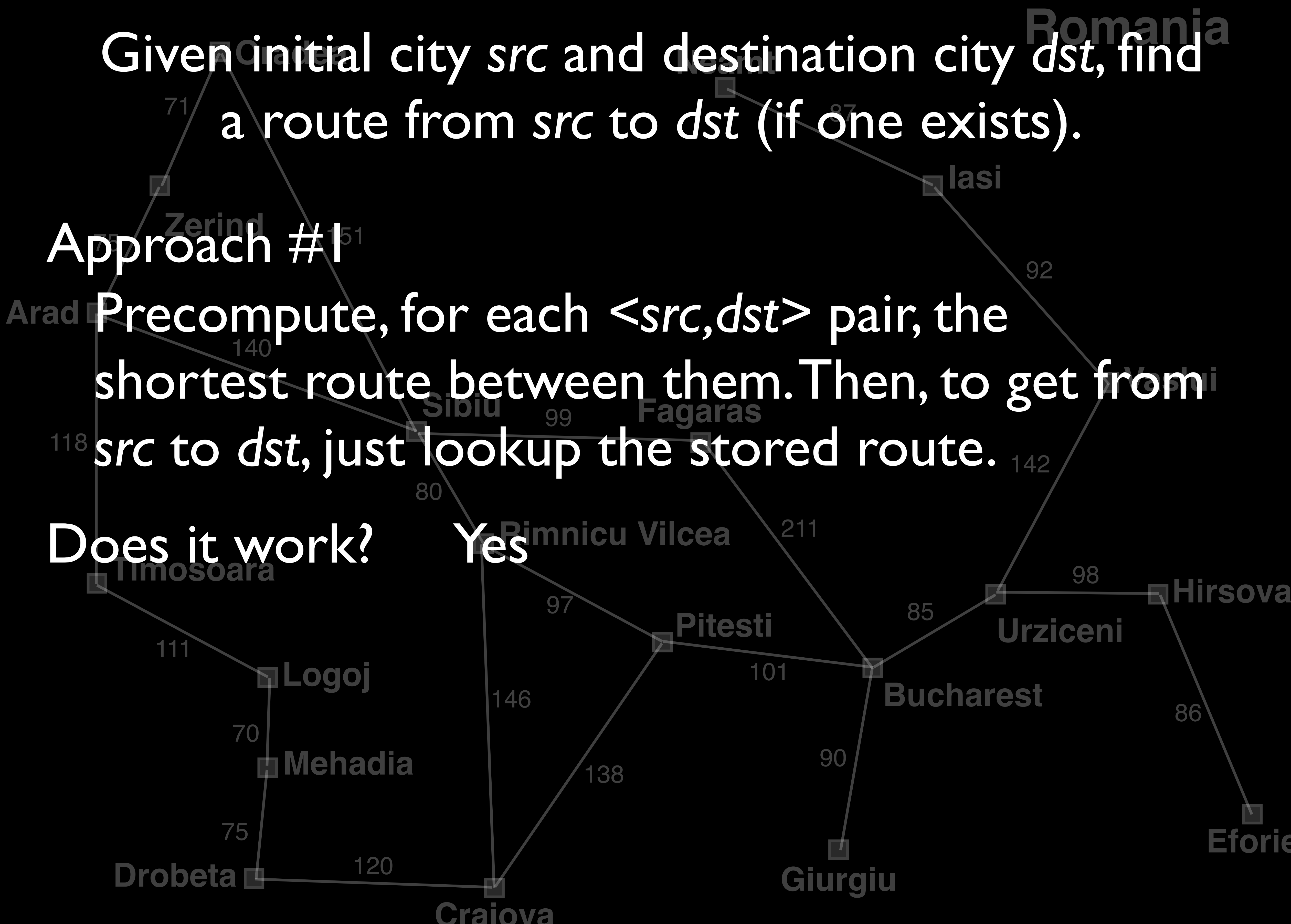


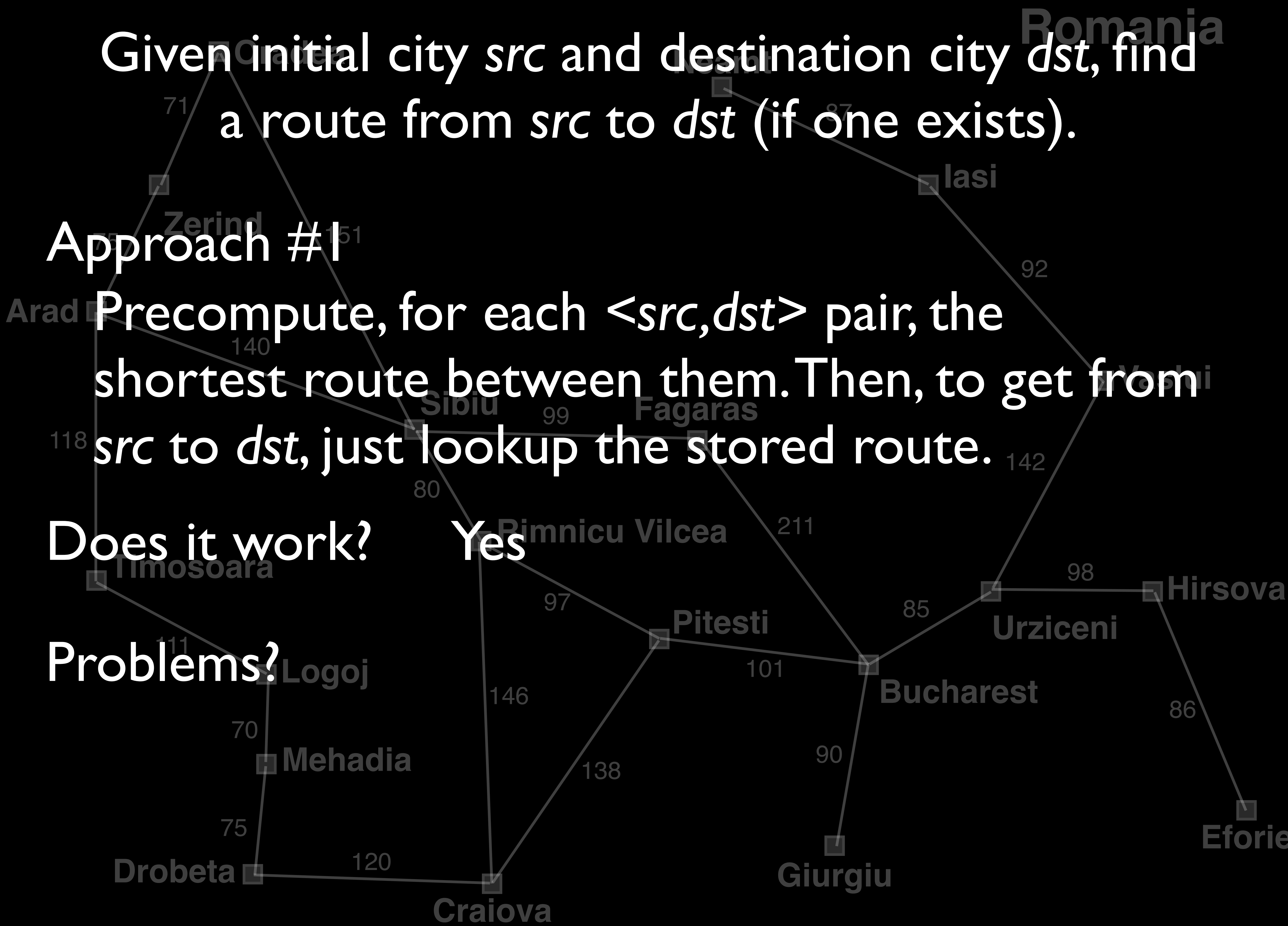
Given initial city *src* and destination city *dst*, find a route from *src* to *dst* (if one exists).

Approach #1

Precompute, for each <*src*,*dst*> pair, the shortest route between them. Then, to get from *src* to *dst*, just lookup the stored route.

Does it work? Yes





Given initial city *src* and destination city *dst*, find a route from *src* to *dst* (if one exists).

Approach #1

Precompute, for each <*src*,*dst*> pair, the shortest route between them. Then, to get from *src* to *dst*, just lookup the stored route.

Does it work? Yes

Problems?

Given initial city *src* and destination city *dst*, find a route from *src* to *dst* (if one exists).

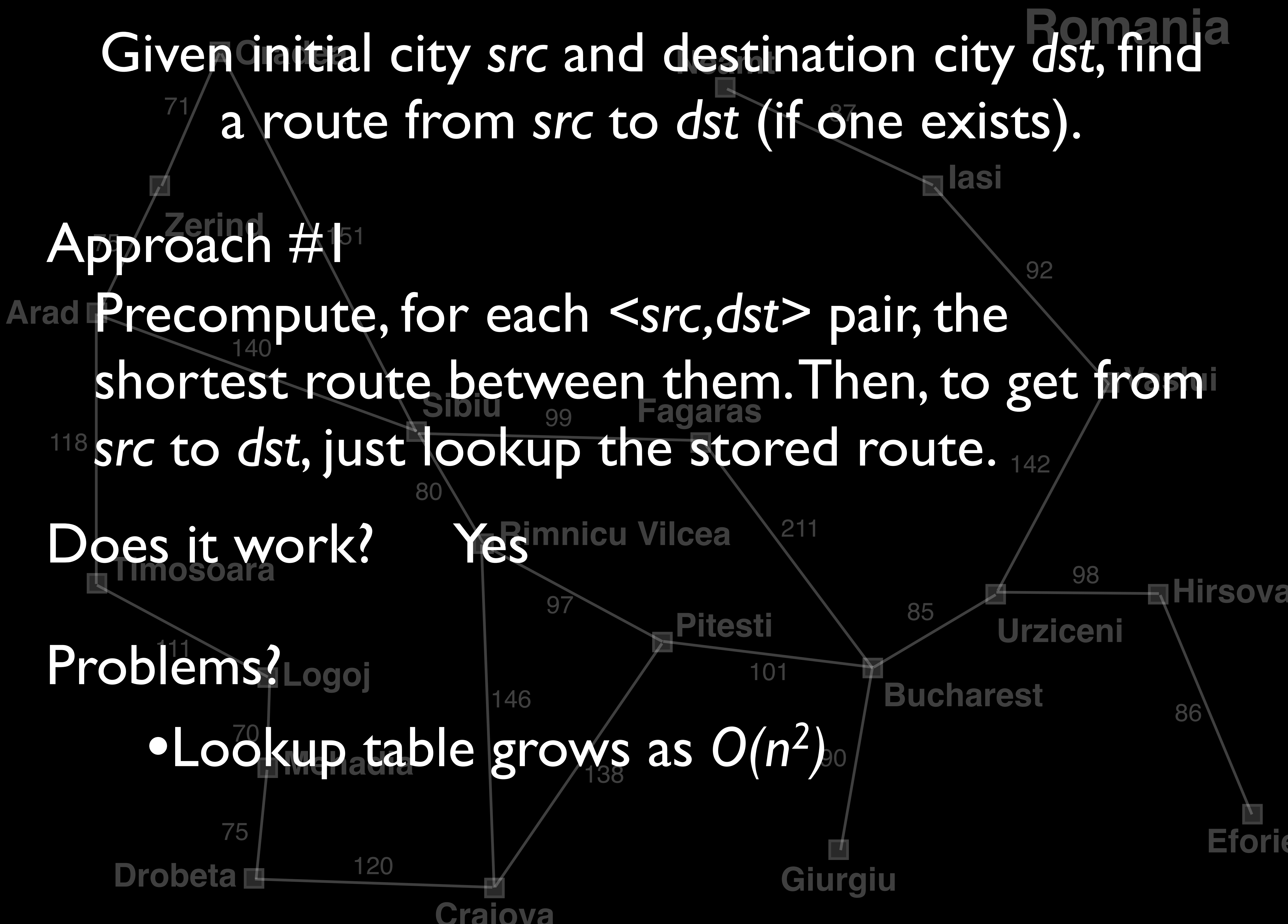
Approach #1

Precompute, for each <*src*,*dst*> pair, the shortest route between them. Then, to get from *src* to *dst*, just lookup the stored route.

Does it work? Yes

Problems?

- Lookup table grows as $O(n^2)$



Given initial city *src* and destination city *dst*, find a route from *src* to *dst* (if one exists).

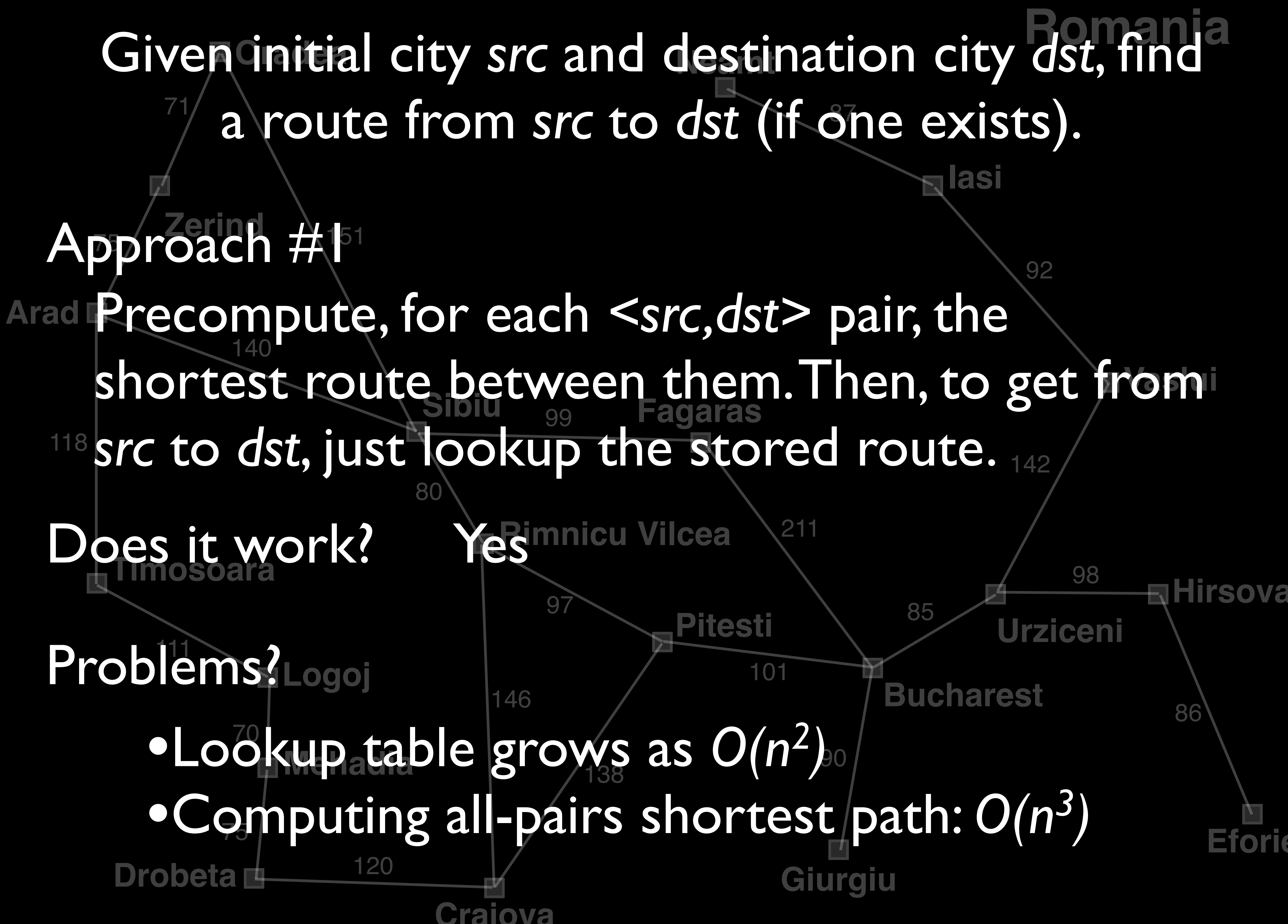
Approach #1

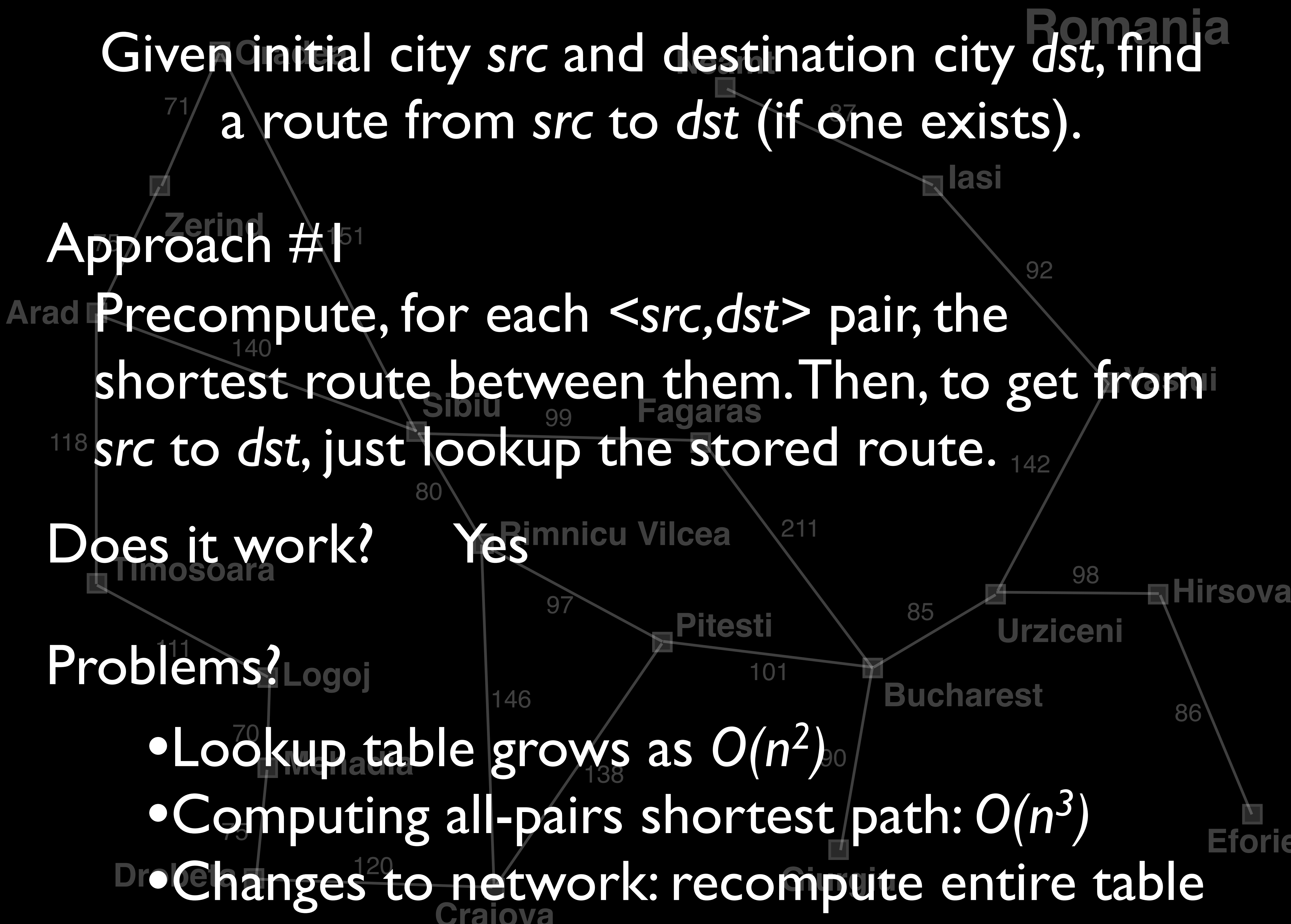
Precompute, for each <*src*,*dst*> pair, the shortest route between them. Then, to get from *src* to *dst*, just lookup the stored route.

Does it work?   Yes

Problems?

- Lookup table grows as $O(n^2)$
- Computing all-pairs shortest path: $O(n^3)$





Given initial city *src* and destination city *dst*, find a route from *src* to *dst* (if one exists).

Approach #1

Precompute, for each <*src*,*dst*> pair, the shortest route between them. Then, to get from *src* to *dst*, just lookup the stored route.

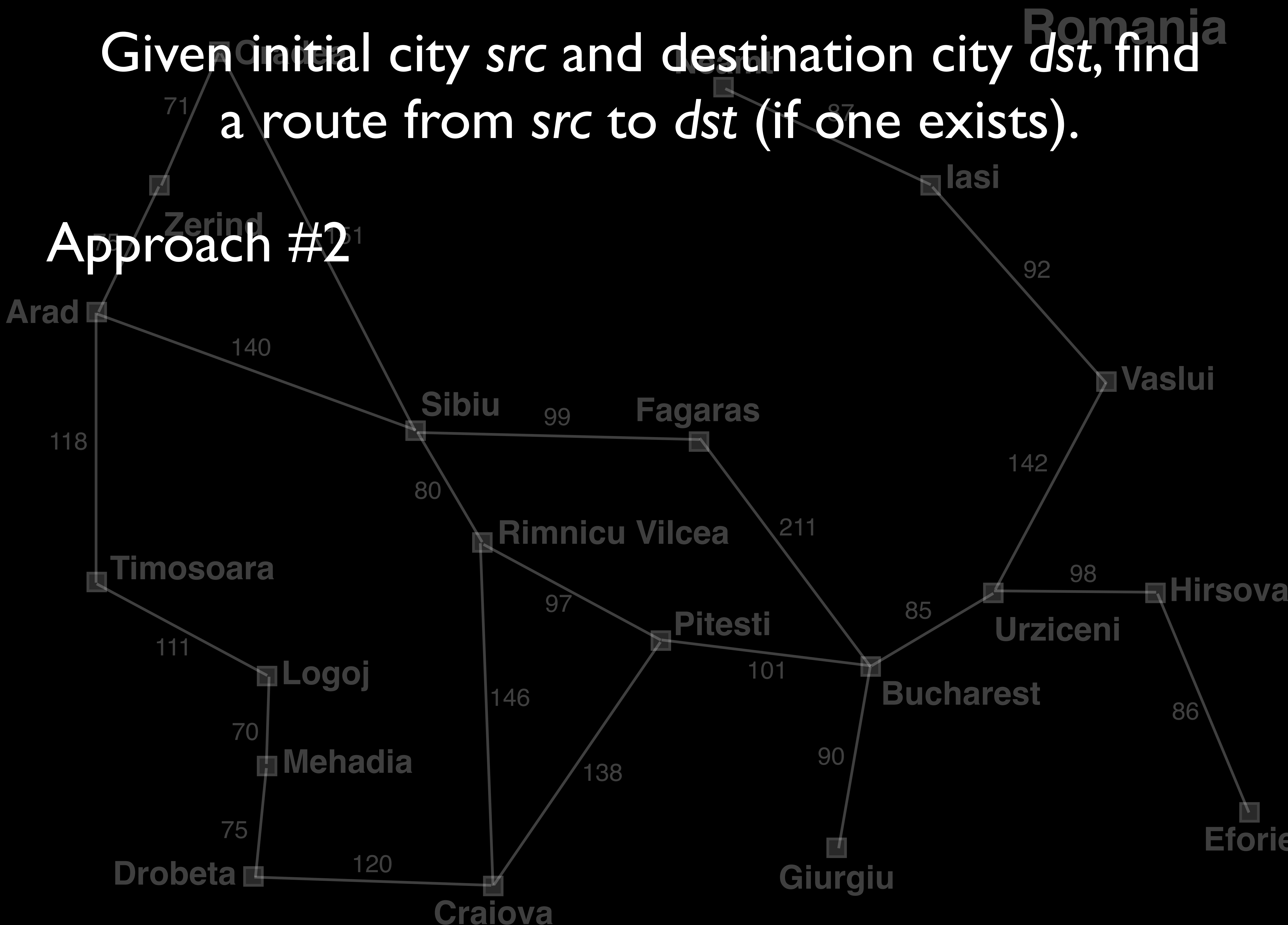
Does it work? Yes

Problems?

- Lookup table grows as $O(n^2)$
- Computing all-pairs shortest path: $O(n^3)$
- Changes to network: recompute entire table

Given initial city *src* and destination city *dst*, find a route from *src* to *dst* (if one exists).

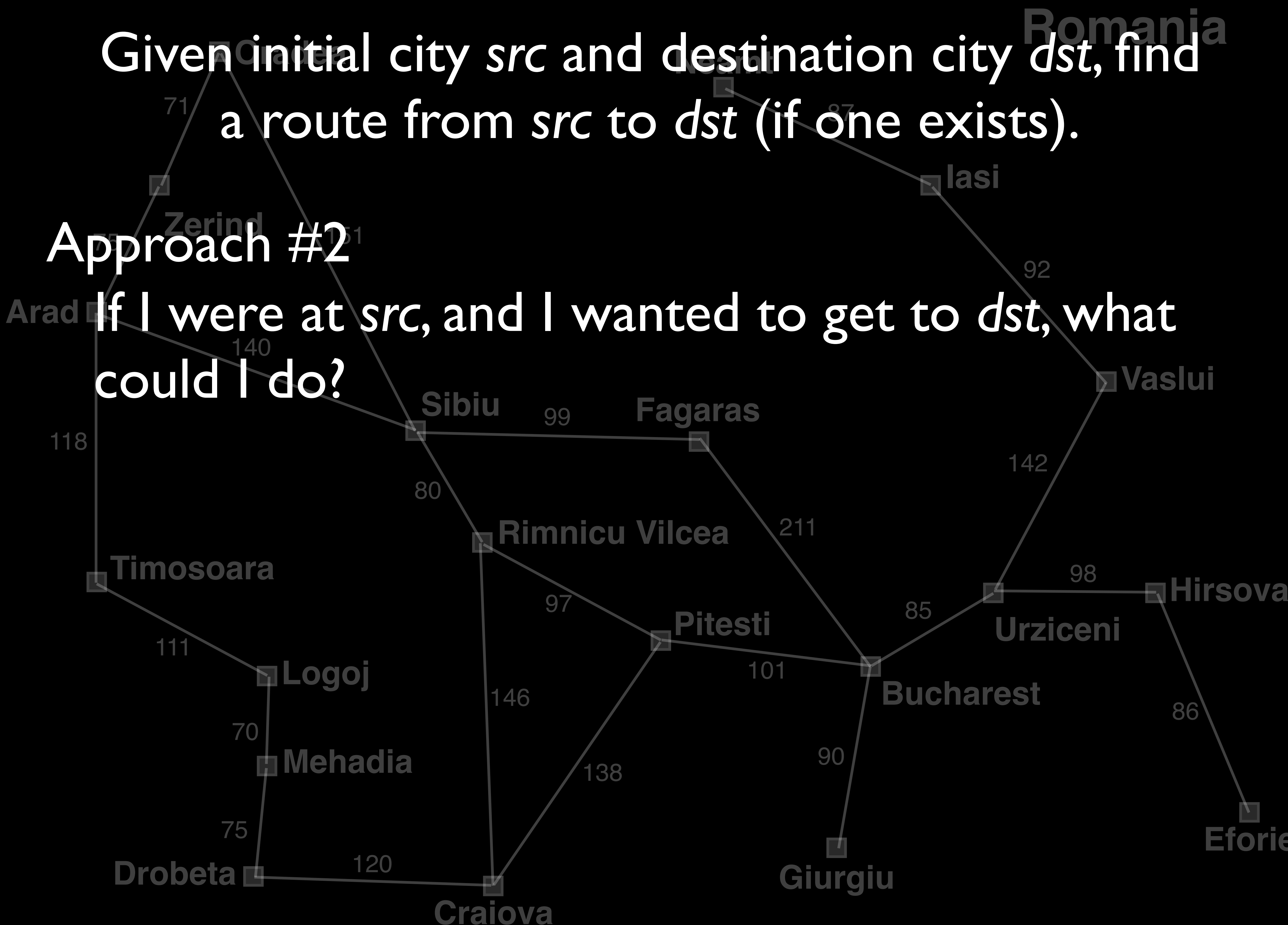
Approach #2



Given initial city *src* and destination city *dst*, find a route from *src* to *dst* (if one exists).

Approach #2

If I were at *src*, and I wanted to get to *dst*, what could I do?

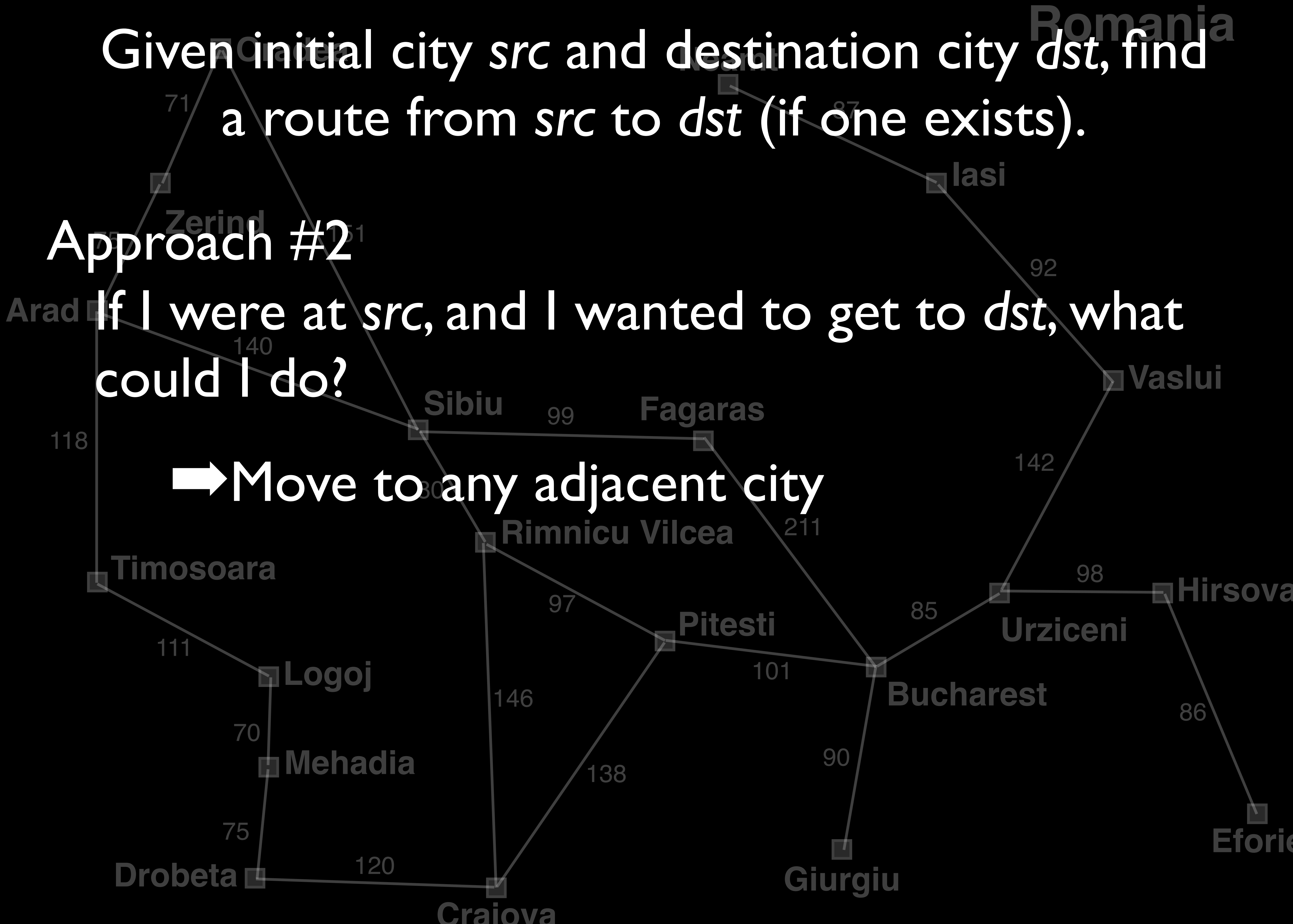


Given initial city *src* and destination city *dst*, find a route from *src* to *dst* (if one exists).

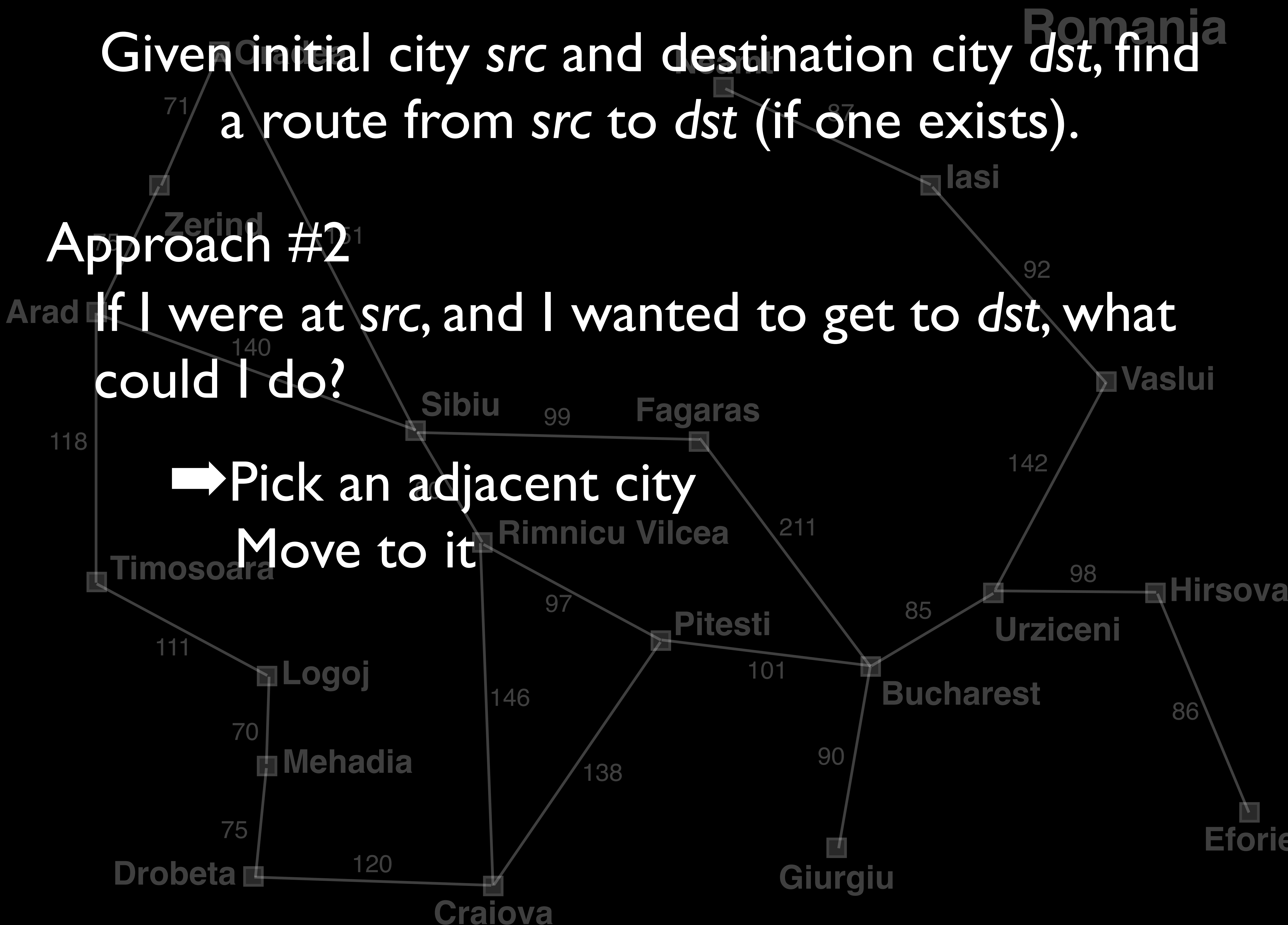
Approach #2

If I were at *src*, and I wanted to get to *dst*, what could I do?

➡Move to any adjacent city



Given initial city *src* and destination city *dst*, find a route from *src* to *dst* (if one exists).



Approach #2

If I were at *src*, and I wanted to get to *dst*, what could I do?

➡ Pick an adjacent city
Move to it

Given initial city *src* and destination city *dst*, find a route from *src* to *dst* (if one exists).

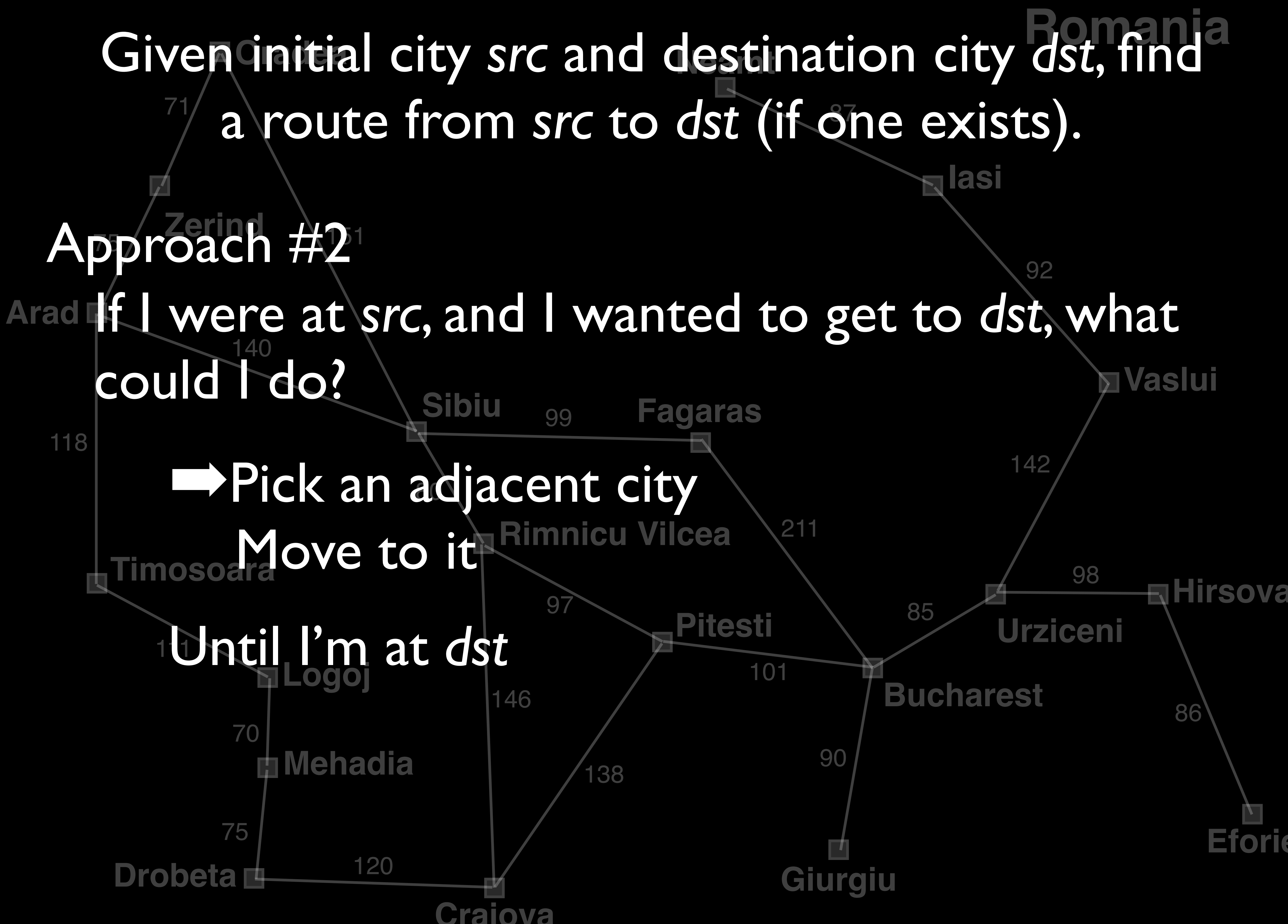
Approach #2

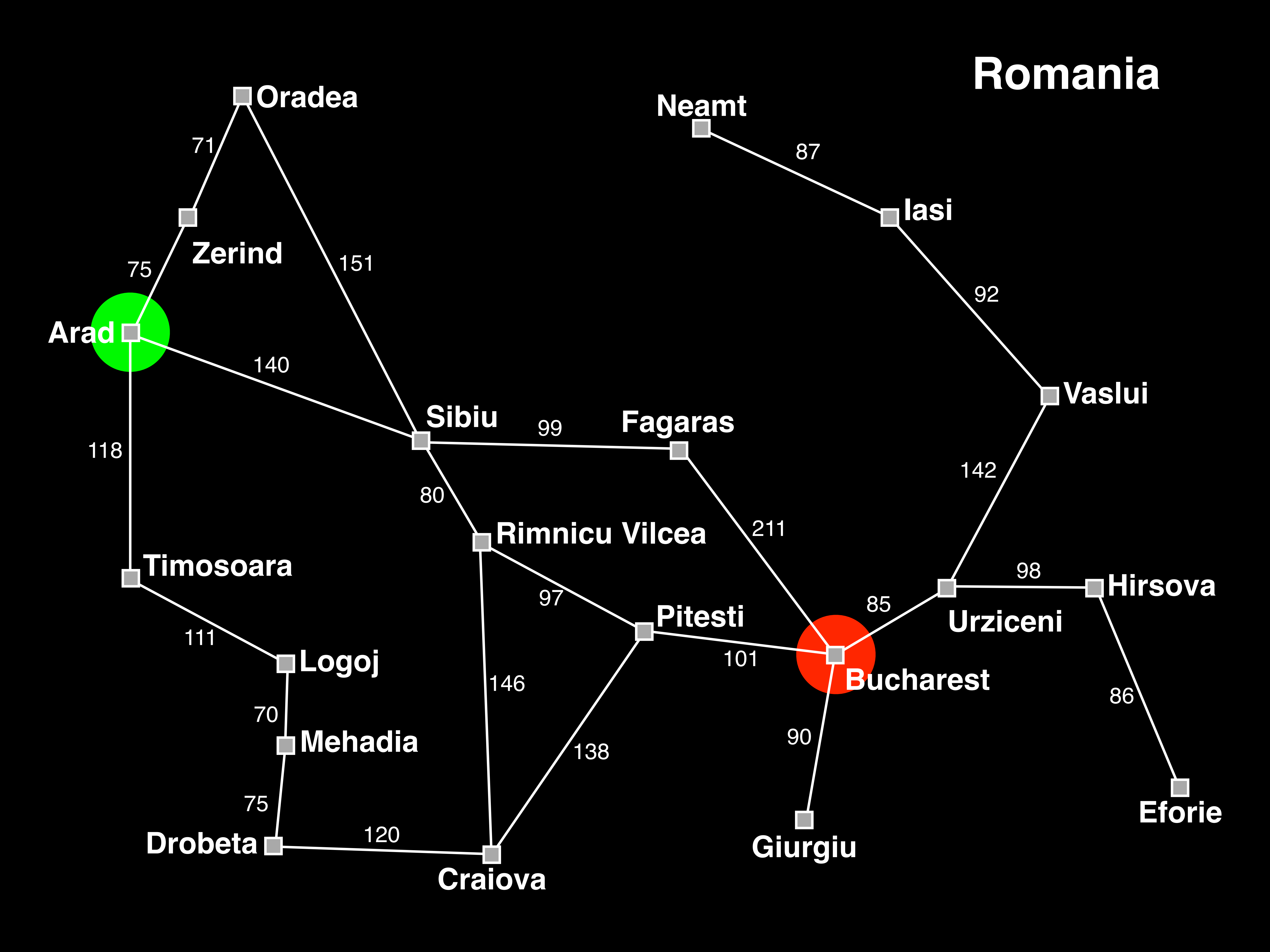
If I were at *src*, and I wanted to get to *dst*, what could I do?

➡ Pick an adjacent city

Move to it

Until I'm at *dst*




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

```
find_solution(City src, City dst) {
  City[] solution = [];
  City c = src;
  while (c != dst) {
    City[] neighbors = adjacent_cities(c);
    City next_c = select_one(neighbors);
    solution.add(next_c);
    c = next_c;
  }
  return solution;
}
```

```
find_solution(City src, City dst) {
  City[] solution = [];
  City c = src;
  while (c != dst) {
    City[] neighbors = adjacent_cities(c);
    City next_c = select_one(neighbors);
    solution.add(next_c);
    c = next_c;
  }
  return solution;
}
```

Problems?

Intelligence and Generality

- Intelligence includes the ability to solve many kinds of problems
- Including problems we haven't seen before
- Every new problem-solving method needs to be designed, implemented, tested, and debugged


ThinkFun
Fifteen Puzzle
INCLUDES:
INSTRUCTION BOOK WITH OVER 30 CHALLENGES
HANDY TRAVEL CASE
SCRAMBLE
SOLVE
STAINLESS STEEL CONSTRUCTION
8 to adult
Fifteen Puzzle

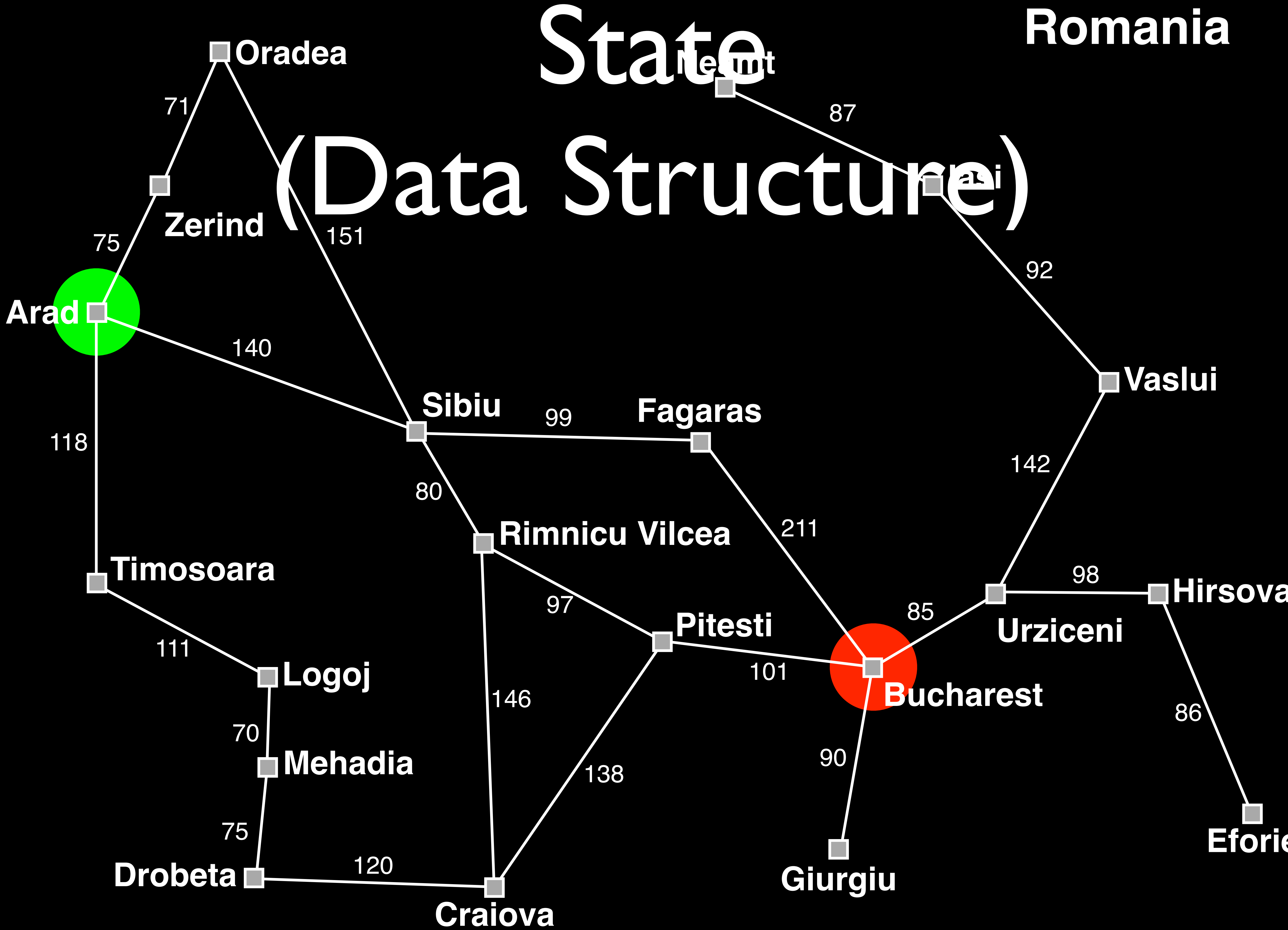
1	2	3
4	5	6
7	8	

Given initial puzzle configuration *start* and desired configuration *goal*, find a sequence of moves that goes from from *start* to *goal* (if one exists).

1	2	3
4	5	6
7	8	

State
(Data Structure)

State
(Data Structure)



```
find_solution(City src, City dst) {
  City[] solution = [];
  City c = src;
  while (c != dst) {
    City[] neighbors = adjacent_cities(c);
    City next_c = select_one(neighbors);
    solution.add(next_c);
    c = next_c;
  }
  return solution;
}
```

State
(Data Structure)

7		1
6	2	8
3	4	5

State
(Data Structure)

7		1
6	2	8
3	4	5

$$M_{i,j} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix}$$

State
(Data Structure)

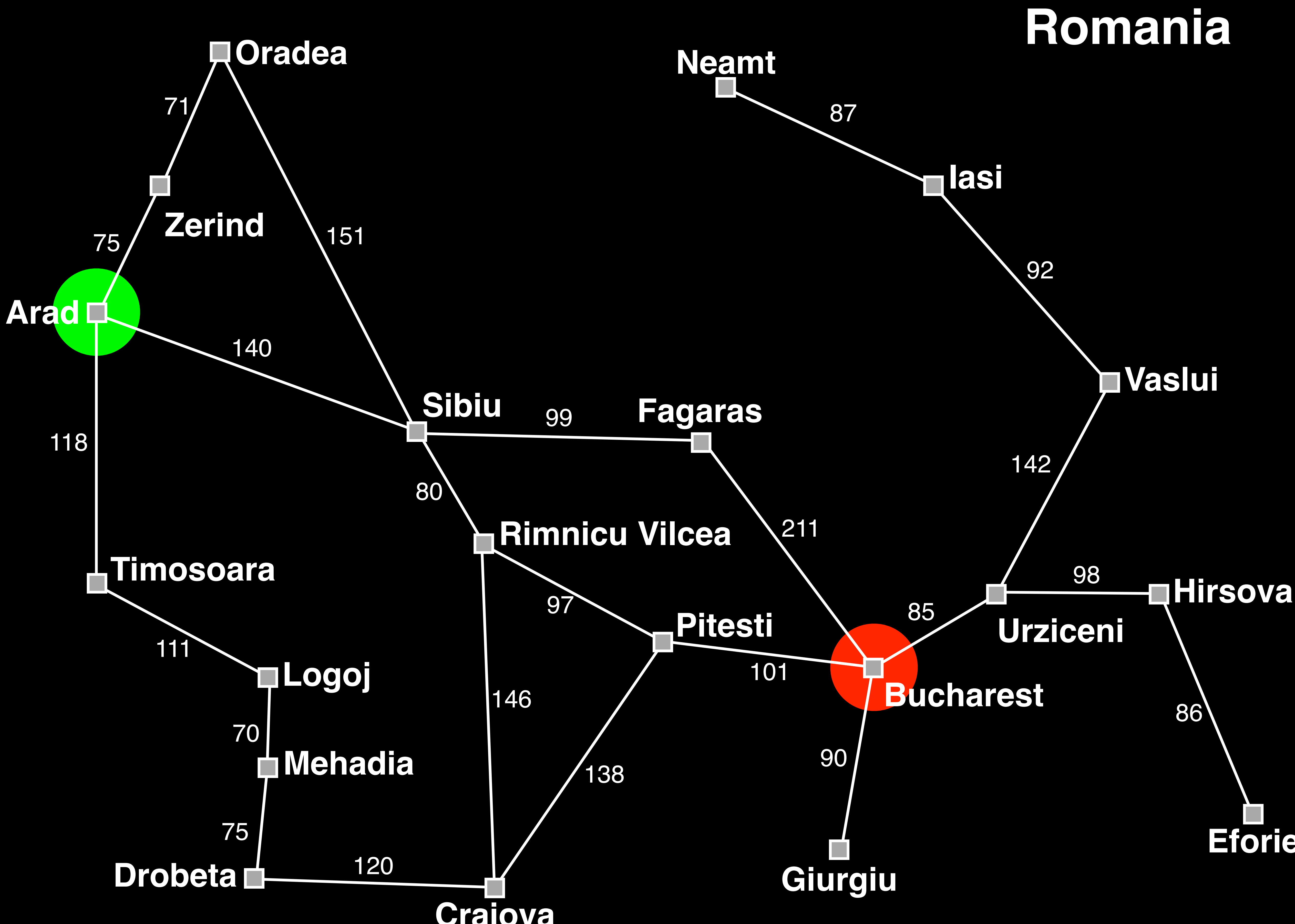
7		1
6	2	8
3	4	5

$$M_{i,j} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix}$$

```
int[][] M = new int[3][3];
M[0][0] = 7;
M[0][1] = 0;
...
M[2][2] = 5;
```

Actions

- Can be performed in a state
- Change the state to a resulting state


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

```
find_solution(City src, City dst) {
  City[] solution = [];
  City c = src;
  while (c != dst) {
    City[] neighbors = adjacent_cities(c);
    City next_c = select_one(neighbors);
    solution.add(next_c);
    c = next_c;
  }
  return solution;
}
```

7		1
6	2	8
3	4	5

Path:

7	1	
6	2	8
3	4	5

Path: 1

7	1	8
6	2	
3	4	5

Path: 1- 8

7	1	8
6		2
3	4	5

Path: 1- 8 - 2

7	1	8
	6	2
3	4	5

Path: 1- 8 - 2 - 6

7		1
6	2	8
3	4	5

Path:

7	1	
6	2	8
3	4	5

Path: East

7	1	8
6	2	
3	4	5

Path: East - South

7	1	8
6		2
3	4	5

Path: East - South - West -

7	1	8
	6	2
3	4	5

Path: East - South - West - West

Actions

- For any state and action:
  - Can I perform this action in this state?
    - “Applicability”
  - How do I update the state if this action is performed?
    - “Result” or “transition” function

$$M^{\mathrm{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \qquad M^{\mathrm{G}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

$$M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \qquad M^{\text{G}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

$$M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \qquad M^{\text{G}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

Approach #1

Precompute, for each <M^{in},M^{G}> pair, a sequence of moves that transforms M^{in} into M^{G}. Then, to solve M^{in}, just lookup the stored sequence.

$$M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \qquad M^{\text{G}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

Approach #1

Precompute, for each <M^{in},M^G> pair, a sequence of moves that transforms M^{in} into M^G. Then, to solve M^{in}, just lookup the stored sequence.

Does it work?

$$M^{\mathrm{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \qquad M^{\mathrm{G}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

Approach #1

Precompute, for each <M^{in},M^{G}> pair, a sequence of moves that transforms M^{in} into M^{G}. Then, to solve M^{in}, just lookup the stored sequence.

Does it work? No

$$M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \qquad M^{\text{G}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

Approach #1

Precompute, for each <M^{in},M^{G}> pair, a sequence of moves that transforms M^{in} into M^{G}. Then, to solve M^{in}, just lookup the stored sequence.

Does it work? No

Problems?

$$M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \qquad M^{\text{G}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

Approach #1

Precompute, for each <M^{in},M^{G}> pair, a sequence of moves that transforms M^{in} into M^{G}. Then, to solve M^{in}, just lookup the stored sequence.

Does it work? No

Problems?

- There are 9! = 362880 (*O(n!)* in general) cases

$$M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \qquad M^{\text{G}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

Approach #1

Precompute, for each <M^{in},M^{G}> pair, a sequence of moves that transforms M^{in} into M^{G}. Then, to solve M^{in}, just lookup the stored sequence.

Does it work? No

Problems?

- There are 9! = 362880 (*O(n!)* in general) cases
- No obvious way to solve the <M^{in},M^{G}> cases

$$M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \qquad M^{\text{G}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

Approach #1

Precompute, for each <M^{in},M^G> pair, a sequence of moves that transforms M^{in} into M^G. Then, to solve M^{in}, just lookup the stored sequence.

Does it work? No

Problems?

- There are 9! = 362880 ($O(n!)$ in general) cases
- No obvious way to solve the <M^{in},M^G> cases
- This problem is known to be NP-complete

$$M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \qquad M^{\text{G}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

Approach #2

$$M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \qquad M^{\text{G}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

Approach #2

- If the puzzle is currently M, and I want it to be M^G, what could I do?

$$M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \qquad M^{\text{G}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

Approach #2

If the puzzle is currently M, and I want it to be M^G, what could I do?

➡Move the blank to an adjacent space

$$M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \qquad M^{\text{G}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

Approach #2

If the puzzle is currently M, and I want it to be M^G, what could I do?

➡Pick an adjacent space
Move the blank to it

$$M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \qquad M^{\text{G}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

Approach #2

If the puzzle is currently M, and I want it to be M^G, what could I do?

➡Pick an adjacent space
Move the blank to it

Until $M == M^G$

Given initial city *src* and destination city *dst*, find a route from *src* to *dst* (if one exists).

Approach #2

If I were at *src*, and I wanted to get to *dst*, what could I do?

➡Pick an adjacent city
Move to it

Until I'm at *dst*

Initial state: S^{IN} Goal state: S^{G}

If the problem state is currently S, and I want it to be S^{G}, what could I do?

➡Pick an applicable action A
Update S with the result of applying A

Until $S == S^{G}$

```
find_solution(State initial) {
  State s = initial;
  Action[] solution = [];
  while (!is_goal(s)) {
    Action a = pick(actions(s));
    solution.add(a);
    s = result(s, a);
  }
  return solution;
}
```

state

```
find_solution(State initial) {
  State s = initial;
  Action[] solution = [];
  while (!is_goal(s)) {
    Action a = pick(actions(s));
    solution.add(a);
    s = result(s, a);
  }
  return solution;
}
```

initial state

```
find_solution(State initial) {
  State s = initial;
  Action[] solution = [];
  while (!is_goal(s)) {
    Action a = pick(actions(s));
    solution.add(a);
    s = result(s, a);
  }
  return solution;
}
```

```
find_solution(State initial) {
  State s = initial;
  Action[] solution = [];
  while (!is_goal(s)) {
    Action a = pick(actions(s));
    solution.add(a);
    s = result(s, a);
  }
  return solution;
}
```

goal test

```
find_solution(State initial) {
  State s = initial;
  Action[] solution = [];
  while (!is_goal(s)) {
    Action a = pick(actions(s));
    solution.add(a);
    s = result(s, a);
  }
  return solution;
}
```

action

```
find_solution(State initial) {
  State s = initial;
  Action[] solution = [];
  while (!is_goal(s)) {
    Action a = pick(actions(s));
    solution.add(a);
    s = result(s, a);
  }
  return solution;
}
```

applicable actions

```
find_solution(State initial) {
  State s = initial;
  Action[] solution = [];
  while (!is_goal(s)) {
    Action a = pick(actions(s));
    solution.add(a);
    s = result(s, a);
  }
  return solution;
}
```

transition function

```
find_solution(State initial) {
  State s = initial;
  Action[] solution = [];
  while (!is_goal(s)) {
    Action a = pick(actions(s));
    solution.add(a);
    s = result(s, a);
  }
  return solution;
}
```

solution

```
find_solution(State initial) {
  State s = initial;
  Action[] solution = [];
  while (!is_goal(s)) {
    Action a = pick(actions(s));
    solution.add(a);
    s = result(s, a);
  }
  return solution;
}
```

Problem (Domain): $\langle \mathcal{S}, \mathcal{A}, \textsc{Actions}, \textsc{Result} \rangle$

$$\textsc{Actions} : s \in \mathcal{S} \rightarrow$$

$$\{a \in \mathcal{A} : a \text{ can be executed (is applicable) in } s\}$$

$$\textsc{Result} : s \in \mathcal{S}, a \in \mathcal{A} \rightarrow$$

$$s' : s' \in \mathcal{S} \text{ s.t. } s' \text{ is the result of performing } a \text{ in } s$$

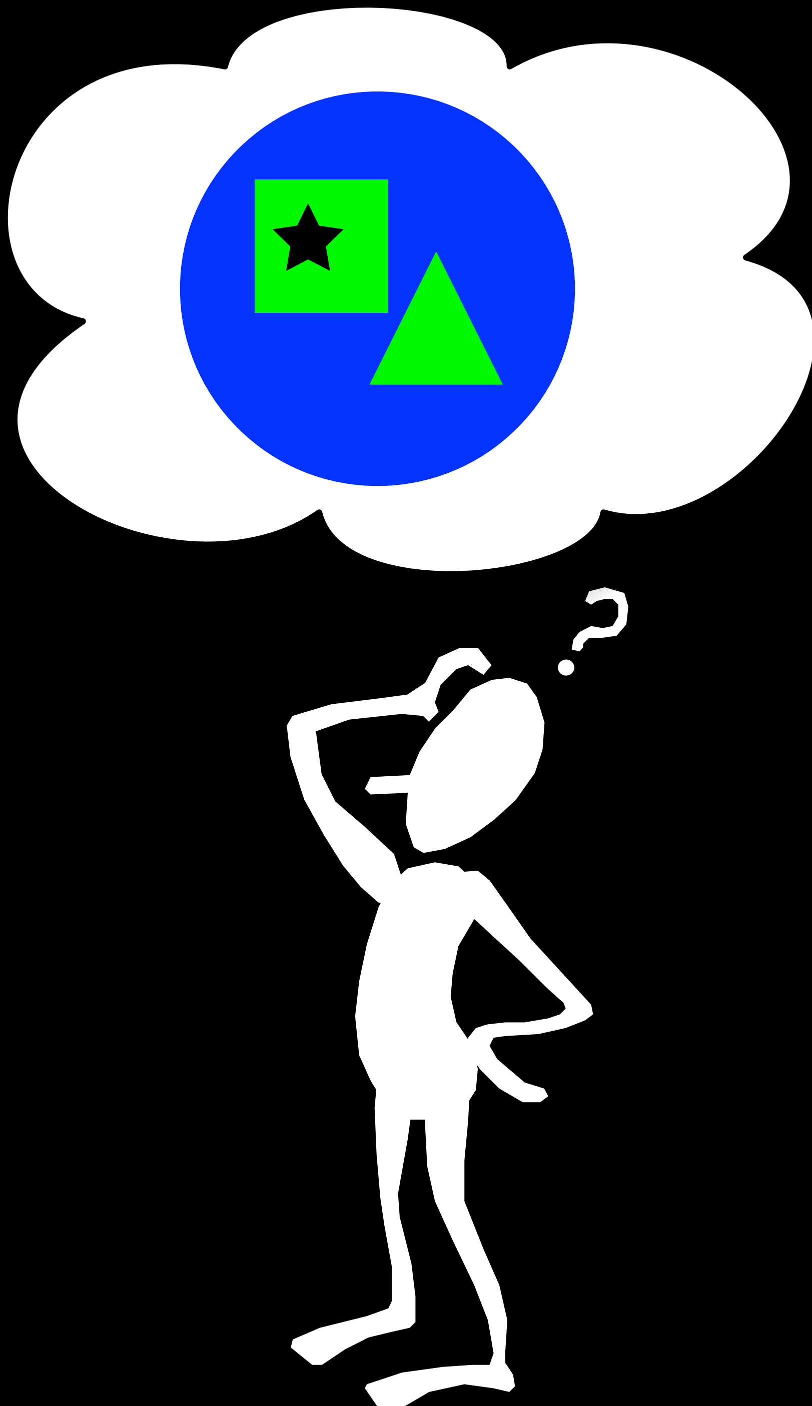
Problem (Instance): $\langle \mathcal{I} \in \mathcal{S}, \mathcal{G} \subseteq \mathcal{S} \rangle$

Solution: $\langle a_0, a_1, \ldots, a_n \rangle \in \mathcal{A}$ s.t.

$$\textsc{Result}(\cdots \textsc{Result}(\textsc{Result}(\mathcal{I}, a_0), a_1) \cdots, a_n) \in \mathcal{G}$$

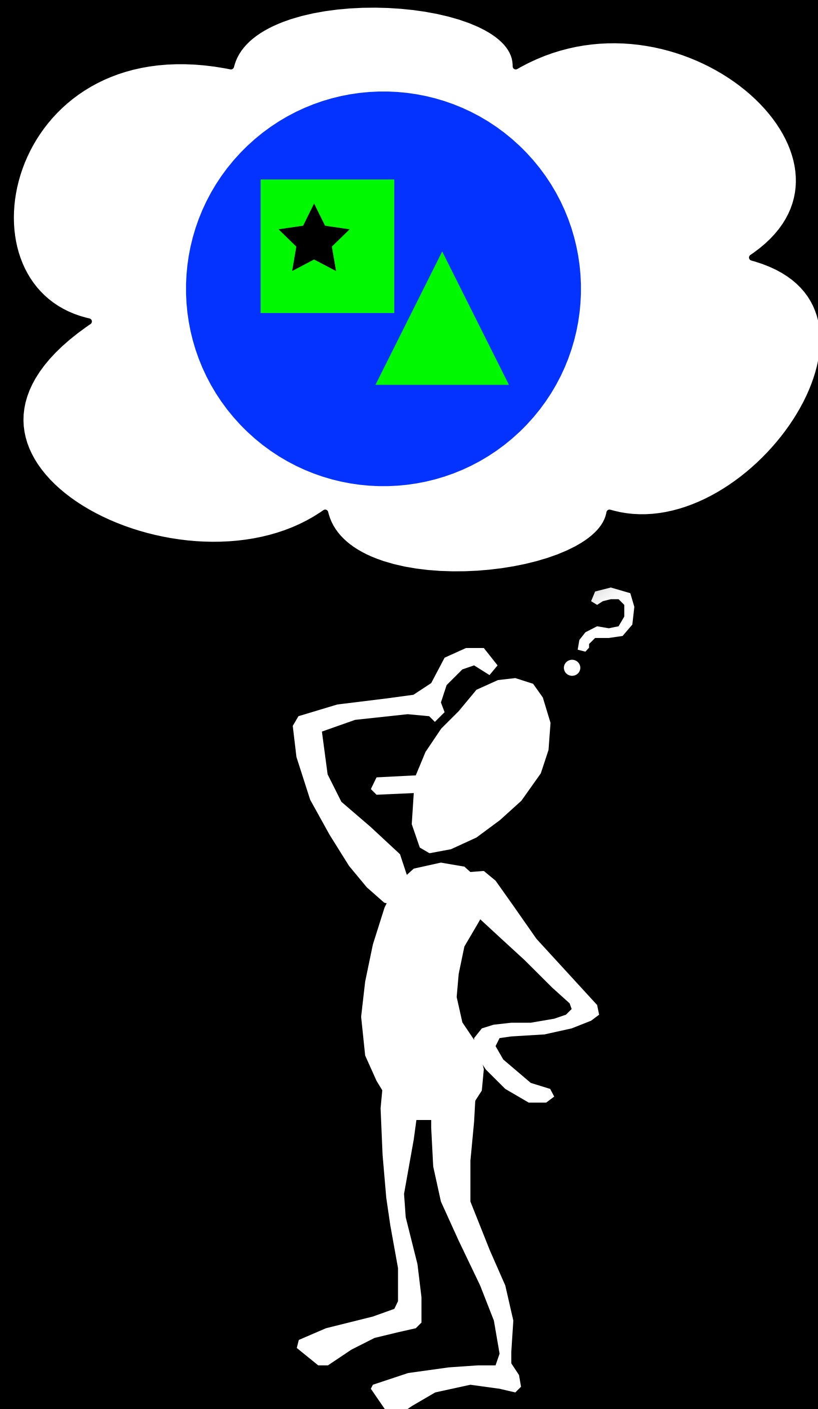
World state

World state



Problem-solving state

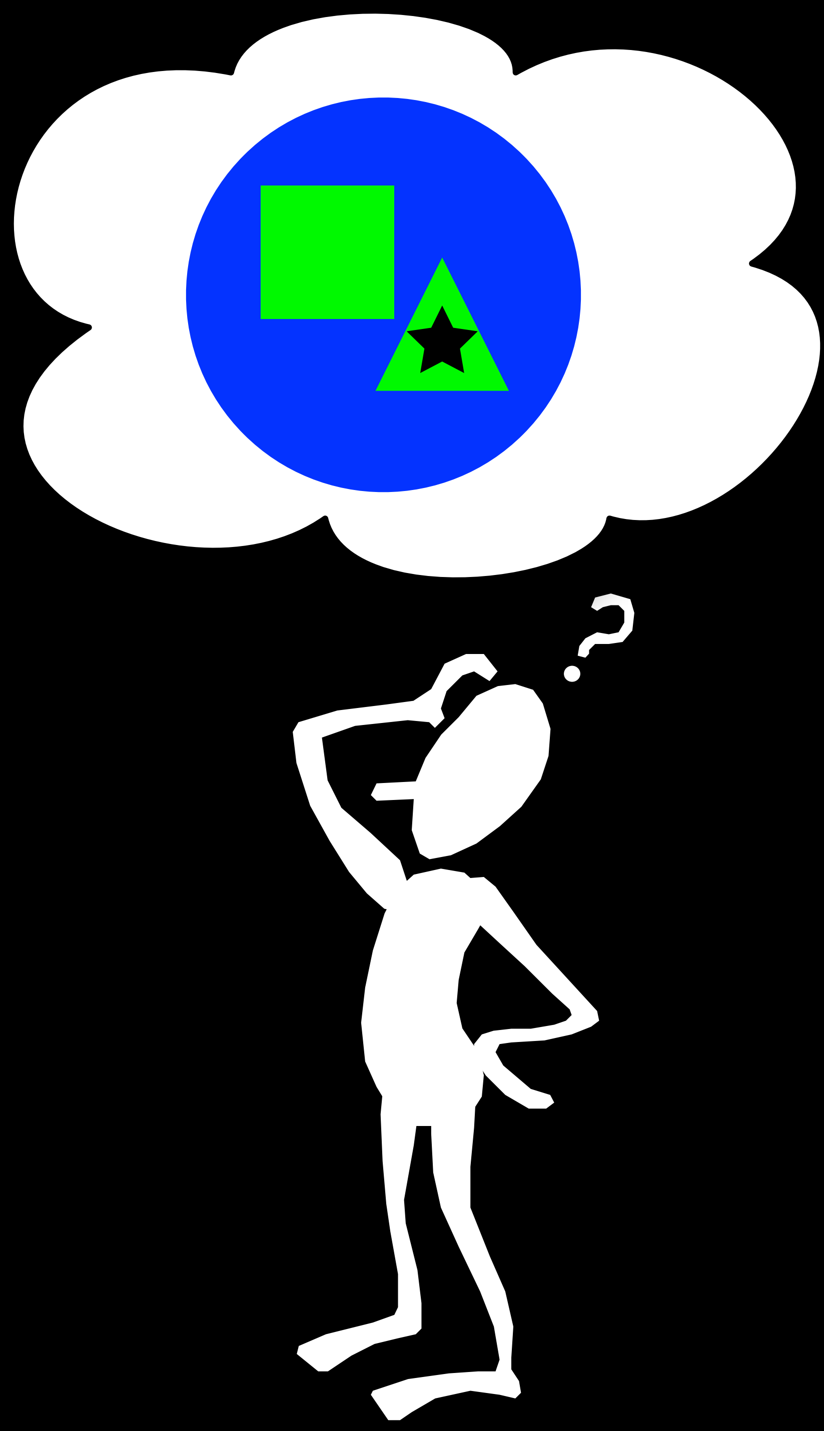
World state



Problem-solving state

World state

Action



Problem-solving state

Transition Model

World state

Action

Universal Problem-Solving Procedure

```
find_solution(initial) {
  state s = initial;
  action[] solution = [];
  while (!is_goal(s)) {
    action a = pick(actions(s));
    solution.add(a);
    s = result(s, a);
  }
  return solution;
}
```

Are We Done Yet?

```
find_solution(initial) {
  state s = initial;
  action[] solution = [];
  while (!is_goal(s)) {
    action a = pick(actions(s));
    solution.add(a);
    s = result(s, a);
  }
  return solution;
}
```


FAILARMY


FAILARMY

Are We Done Yet?

```
find_solution(initial) {
  state s = initial;
  action[] solution = [];
  while (!is_goal(s)) {
    action a = pick(actions(s));
    solution.add(a);
    s = result(s, a);
  }
  return solution;
}
```

Are We Done Yet?

```
find_solution(initial) {
  state s = initial;
  action[] solution = [];
  while (!is_goal(s)) {
    action a = pick(actions(s));
    solution.add(a);
    s = result(s, a);
  }
  return solution;
}
```

FAIL

State-Space Search

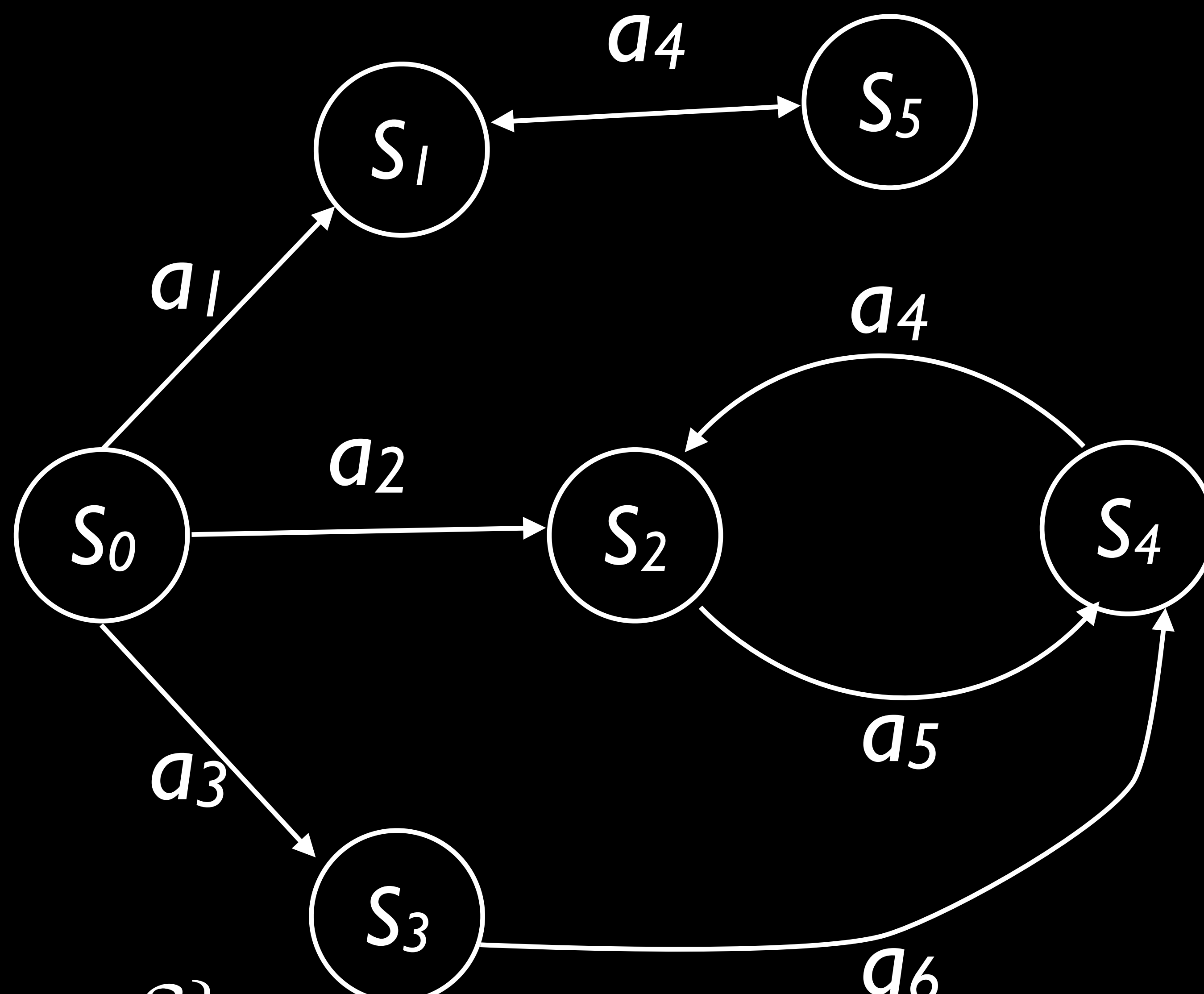
States + Actions + Transition Model

=

State Space

The set of all states reachable from the initial state by some sequence of actions

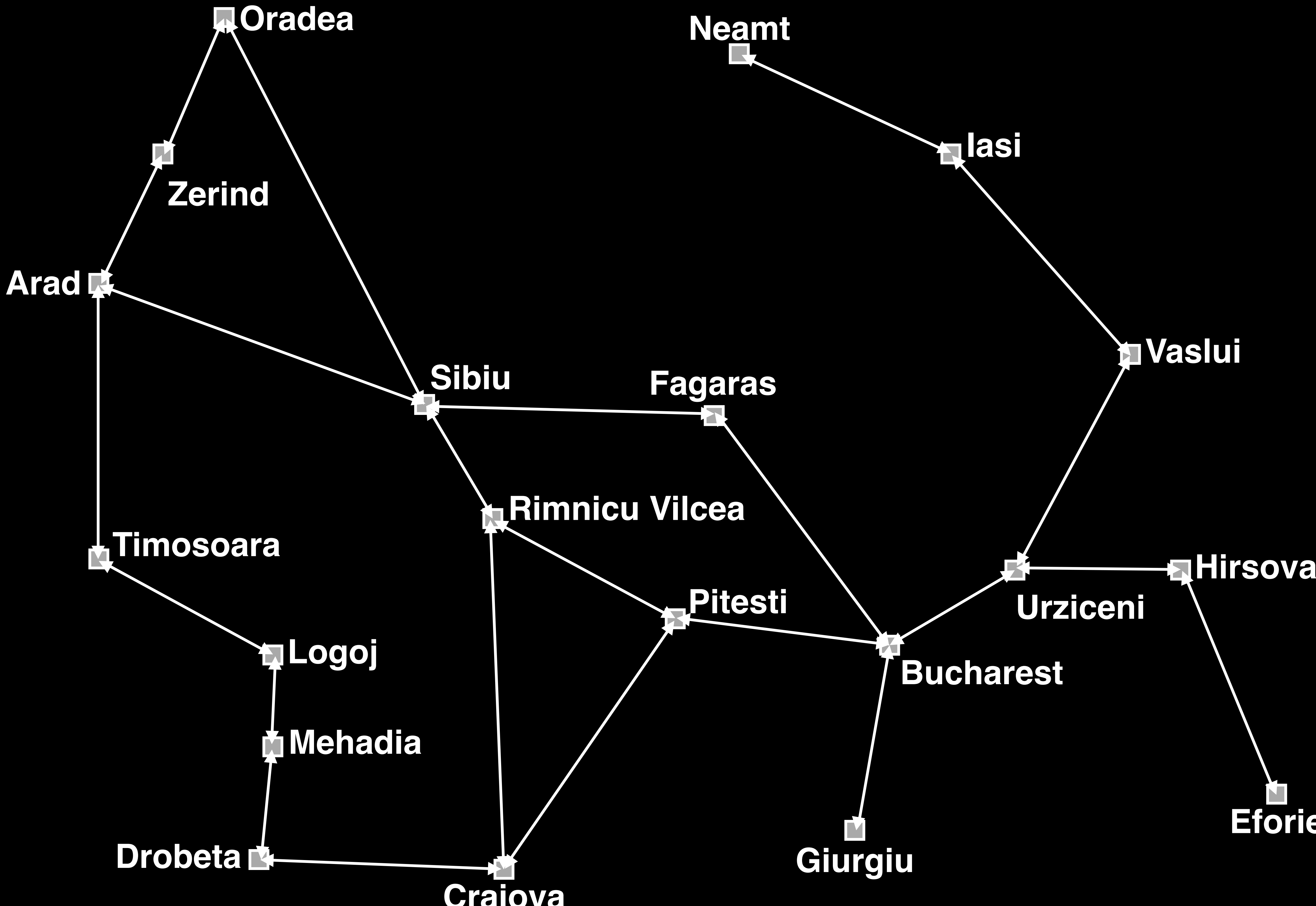
State Space



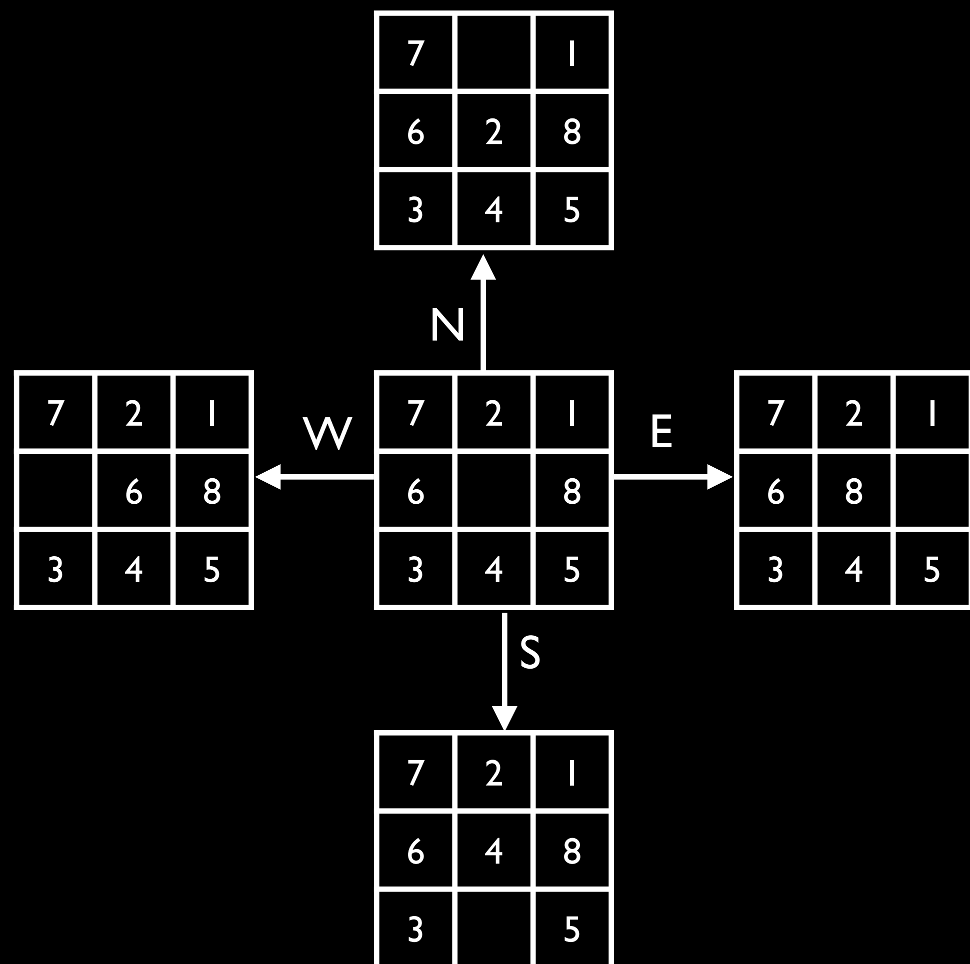
$\langle V, E \rangle :$

$$V = \{v_i \mid s_i \in \mathcal{S}\}$$

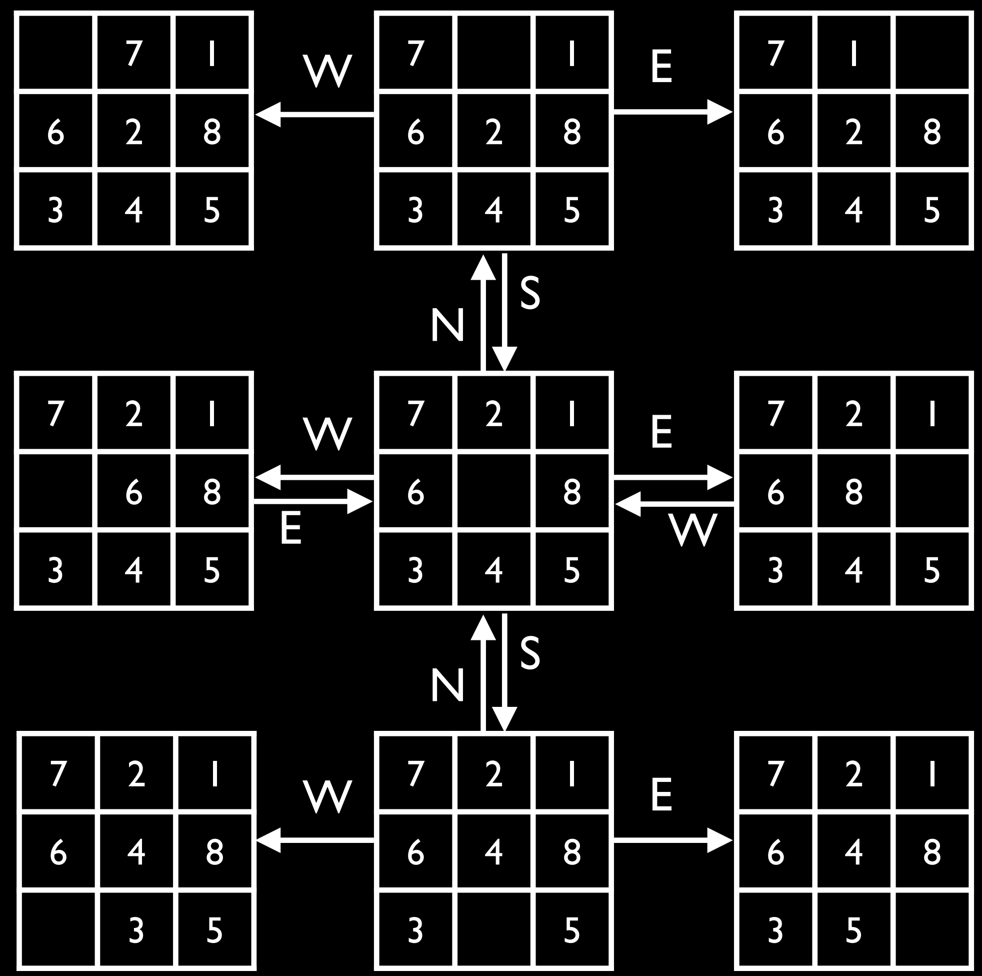
$$E = \{\langle v_i, v_j, a \rangle \mid s_j = \textsc{Result}(s_i, a)\}$$


Oradea
Neamt
Zerind
Iasi
Arad
Vaslui
Sibiu
Fagaras
Rimnicu Vilcea
Timosoara
Hirsova
Pitesti
Urziceni
Logoj
Bucharest
Mehadia
Eforie
Drobeta
Giurgiu
Craiova

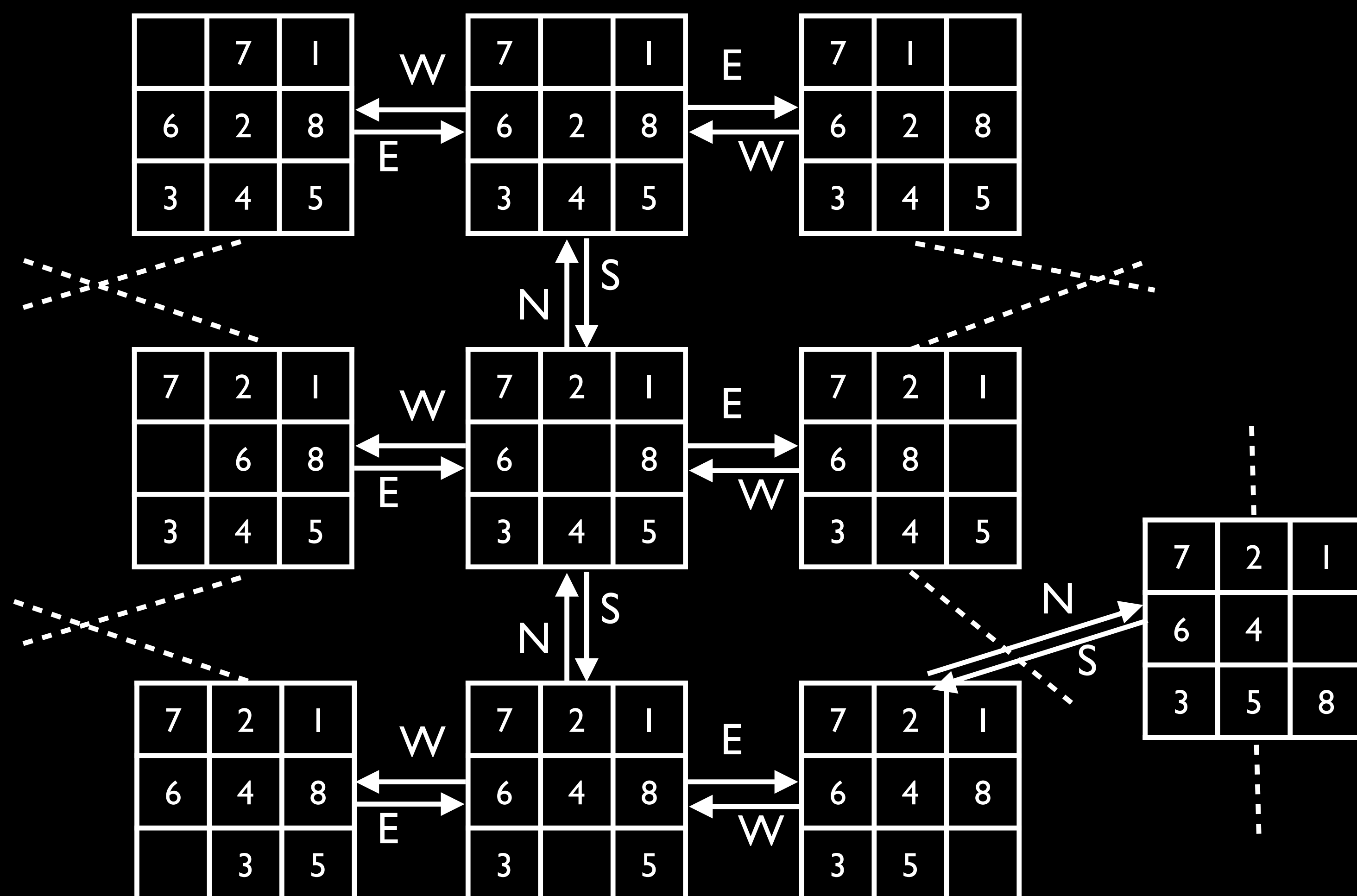
7	2	1
6		8
3	4	5



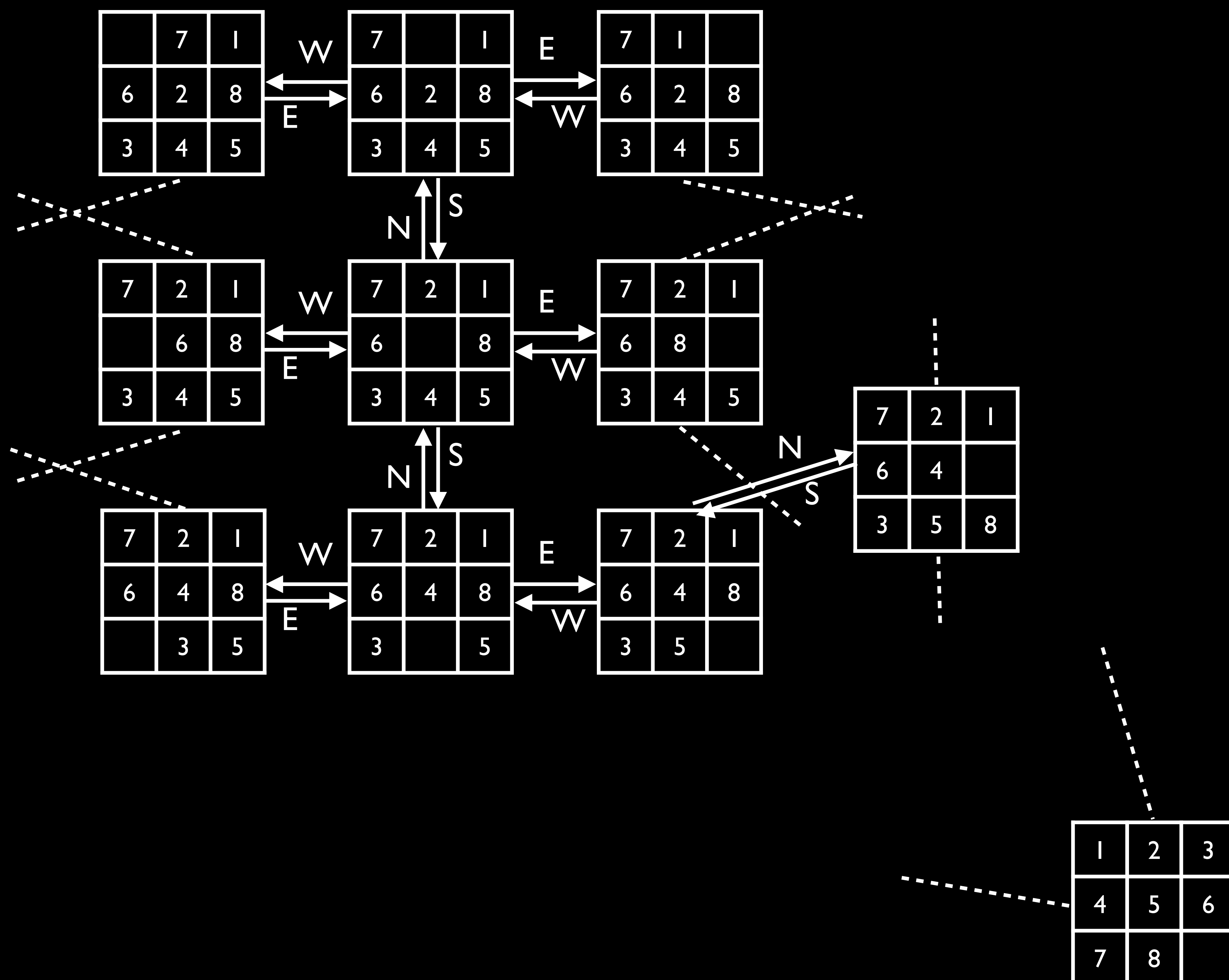
7
1
6
2
8
3
4
5
N
7
2
1
6
8
3
4
5
W
7
2
1
6
8
3
4
5
E
7
2
1
6
8
3
4
5
S
7
2
1
6
4
8
3
5

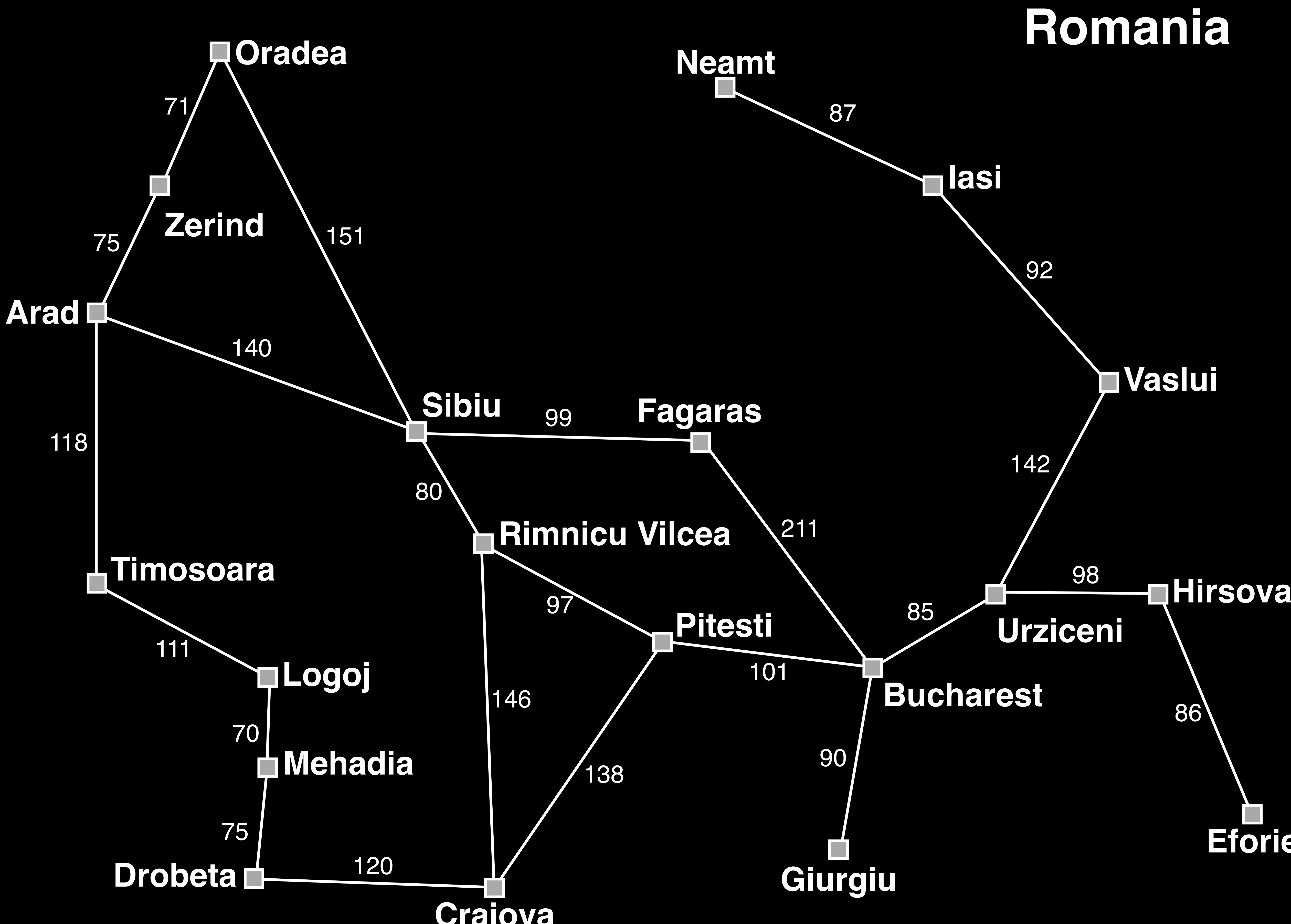


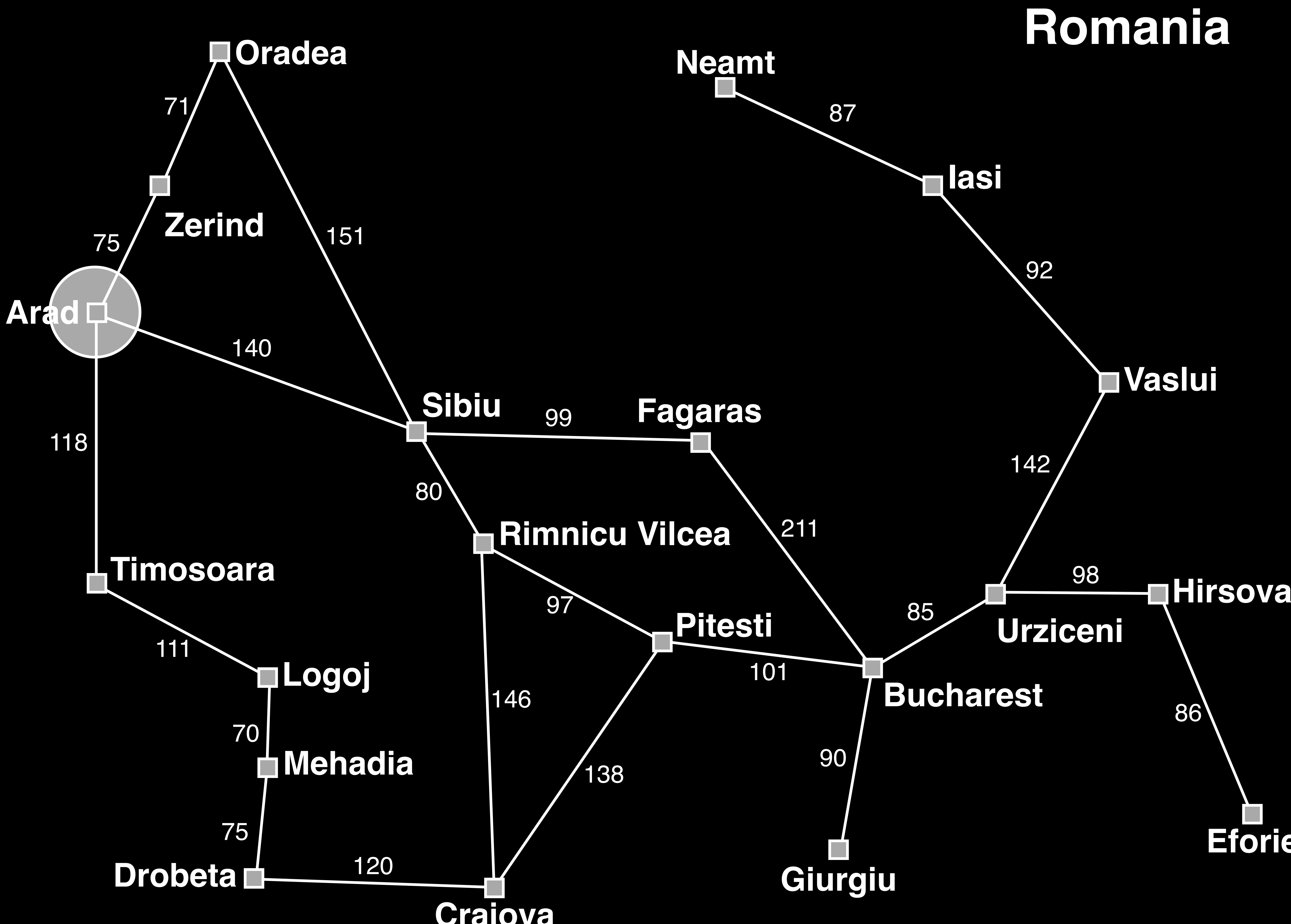
7
1
W
7
1
E
7
1
6
2
8
3
4
5
N
S
7
2
1
6
8
W
E
E
W
N
S
7
2
1
6
4
8
3
5



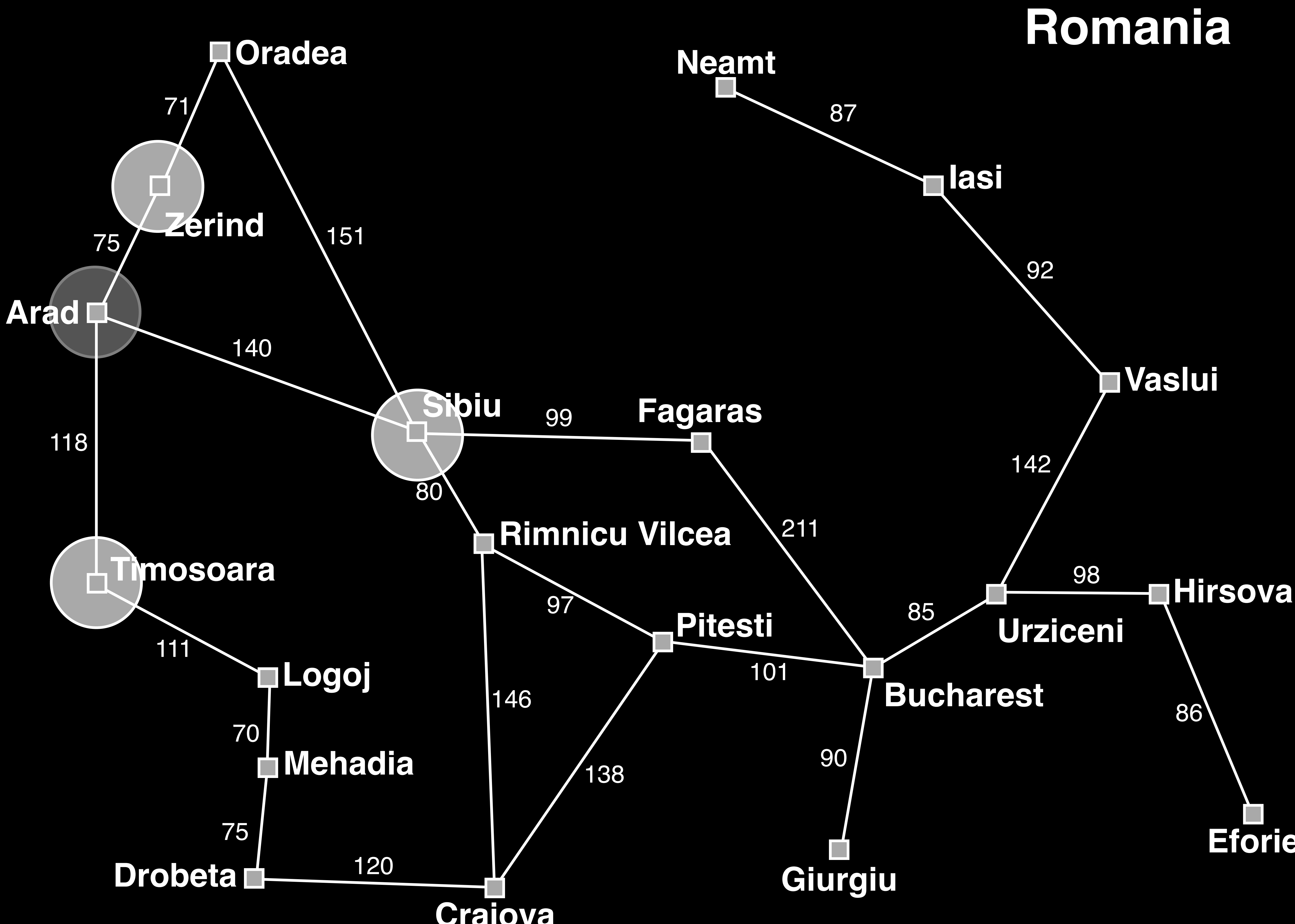
7 1
6 2 8
3 4 5
7 1
6 2 8
3 4 5
7 1
6 2 8
3 4 5
7 2 1
6 8
3 4 5
7 2 1
6 8
3 4 5
7 2 1
6 8
3 4 5
7 2 1
6 4 8
3 5
7 2 1
6 4 8
3 5
7 2 1
6 4 8
3 5
7 2 1
6 4
3 5 8
W
E
N
S


W
E
N
S

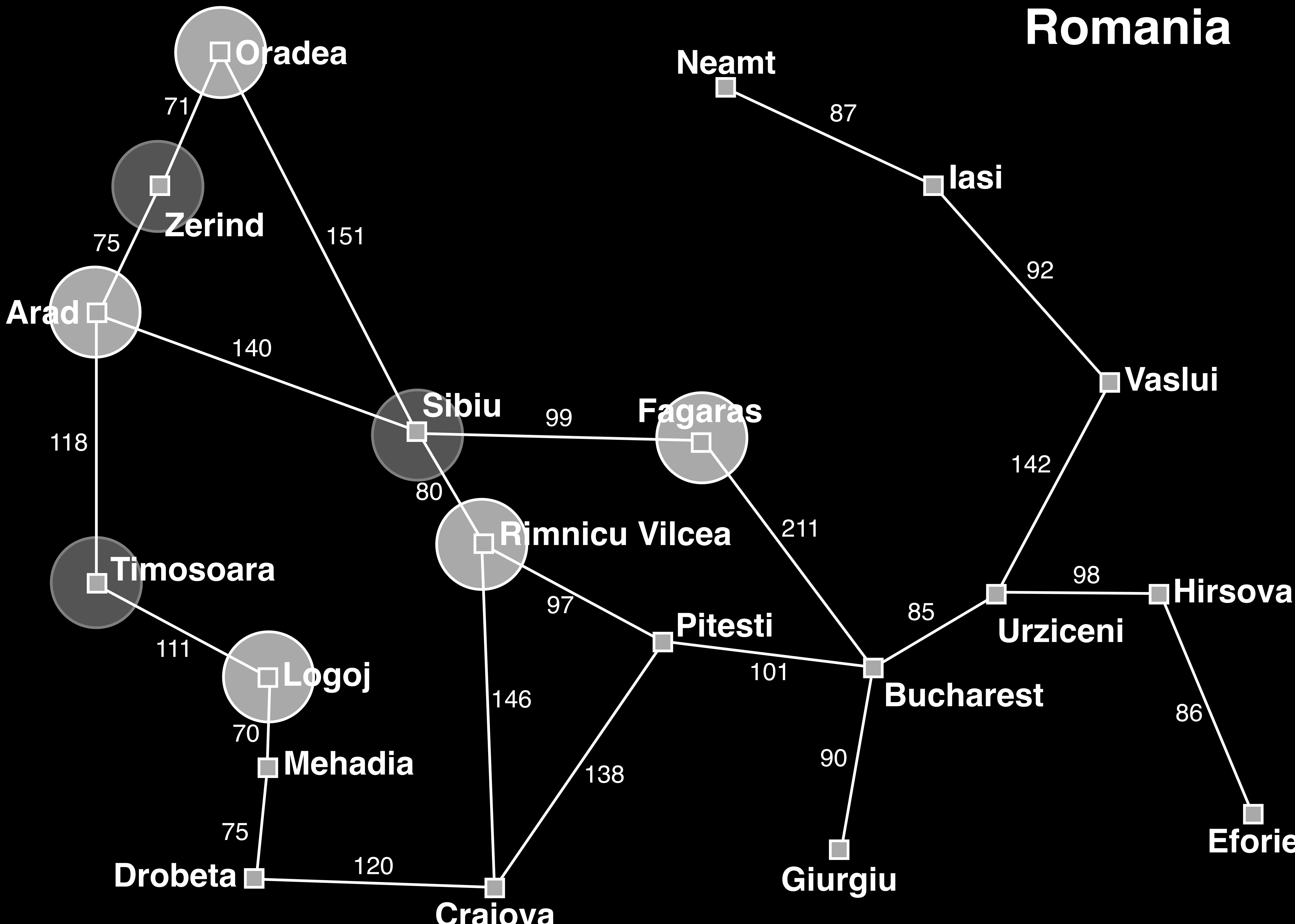

Romania
Oradea
Neamt
71
87
Zerind
Iasi
151
75
92
Arad
140
Vaslui
Sibiu
99
Fagaras
118
142
80
211
Rimnicu Vilcea
Timosoara
98
Hirsova
97
Pitesti
85
Urziceni
111
Logoj
101
Bucharest
146
86
70
Mehadia
90
138
75
Eforie
120
Drobeta
Giurgiu
Craiova



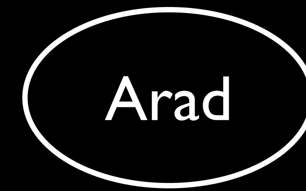
Romania
Oradea
Neamt
71
87
Zerind
Iasi
75
151
92
Arad
140
Vaslui
Sibiu
99
Fagaras
118
142
80
Rimnicu Vilcea
211
Timosoara
98
97
Pitesti
85
Hirsova
Urziceni
111
Logoj
101
Bucharest
146
86
70
Mehadia
90
138
75
Eforie
Drobeta
120
Giurgiu
Craiova

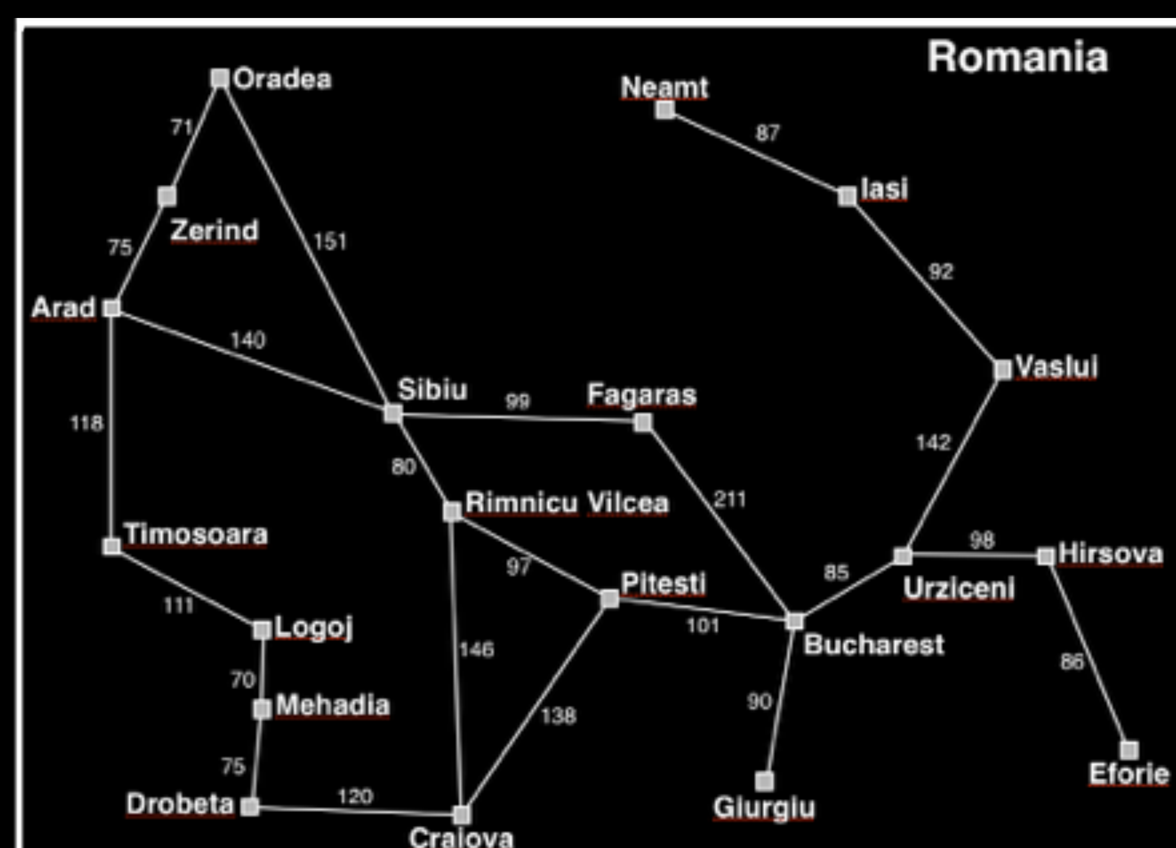

Romania
Oradea
Neamt
71
87
Zerind
Iasi
151
75
92
Arad
140
Vaslui
Sibiu
99
Fagaras
118
142
80
Rimnicu Vilcea
211
Timosoara
98
Hirsova
97
Pitesti
85
Urziceni
111
Logoj
101
Bucharest
146
86
70
Mehadia
90
138
Eforie
75
Drobeta
120
Giurgiu
Craiova

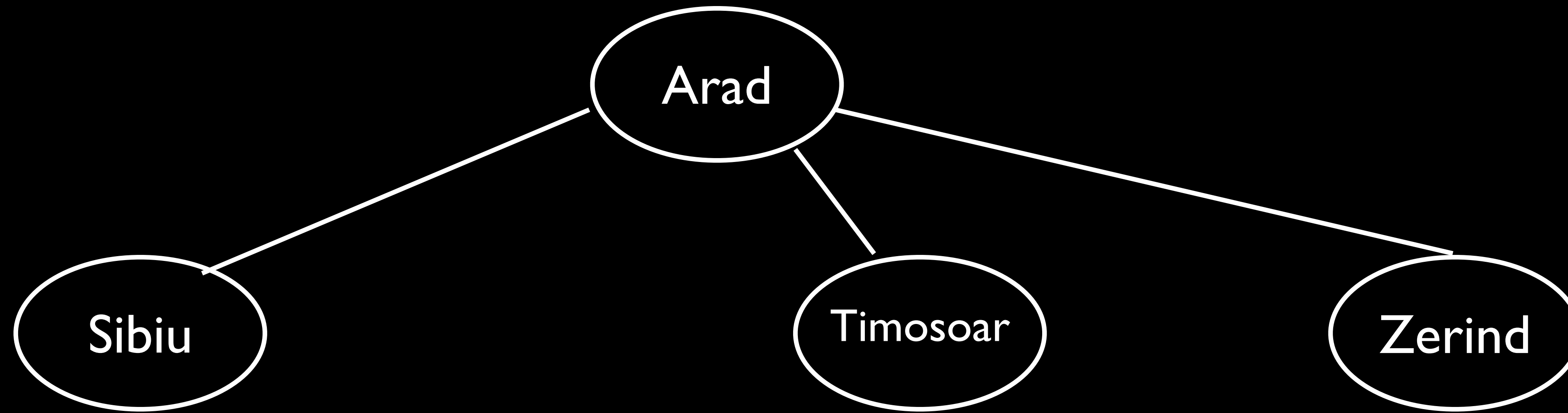
Romania

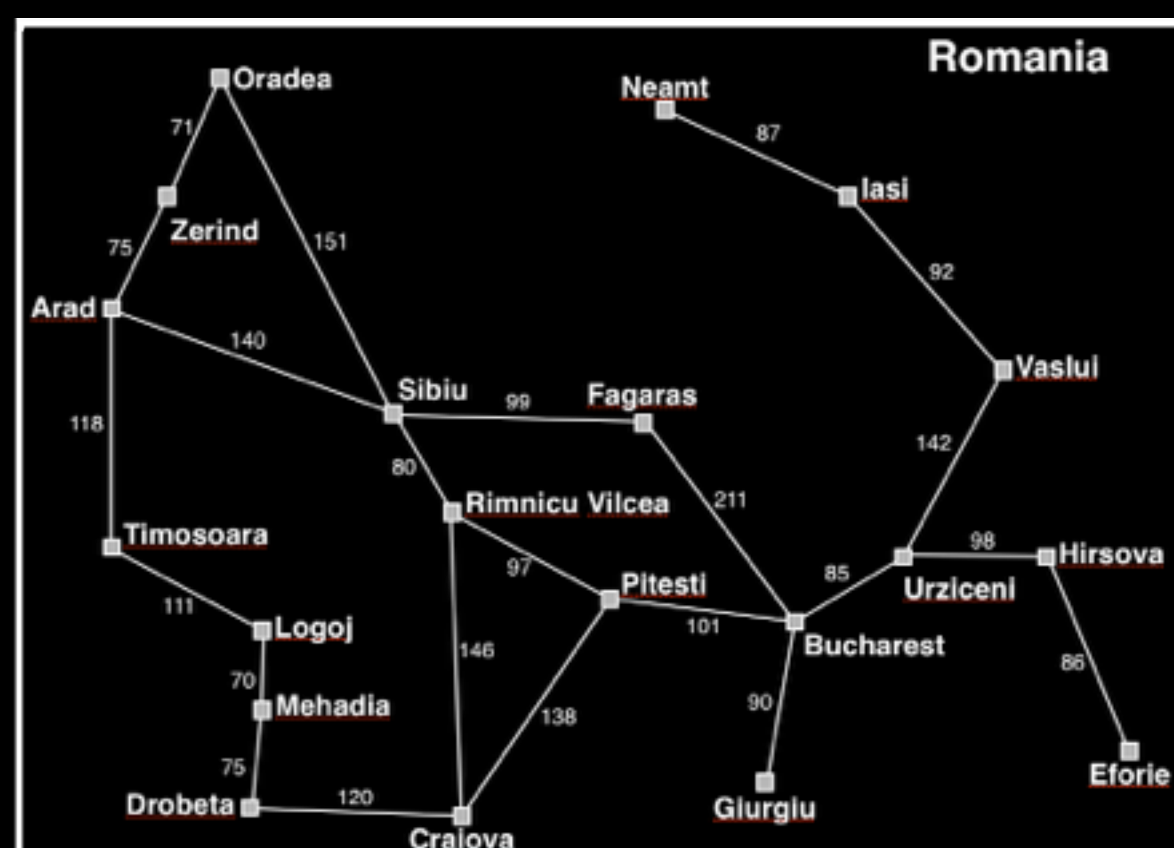


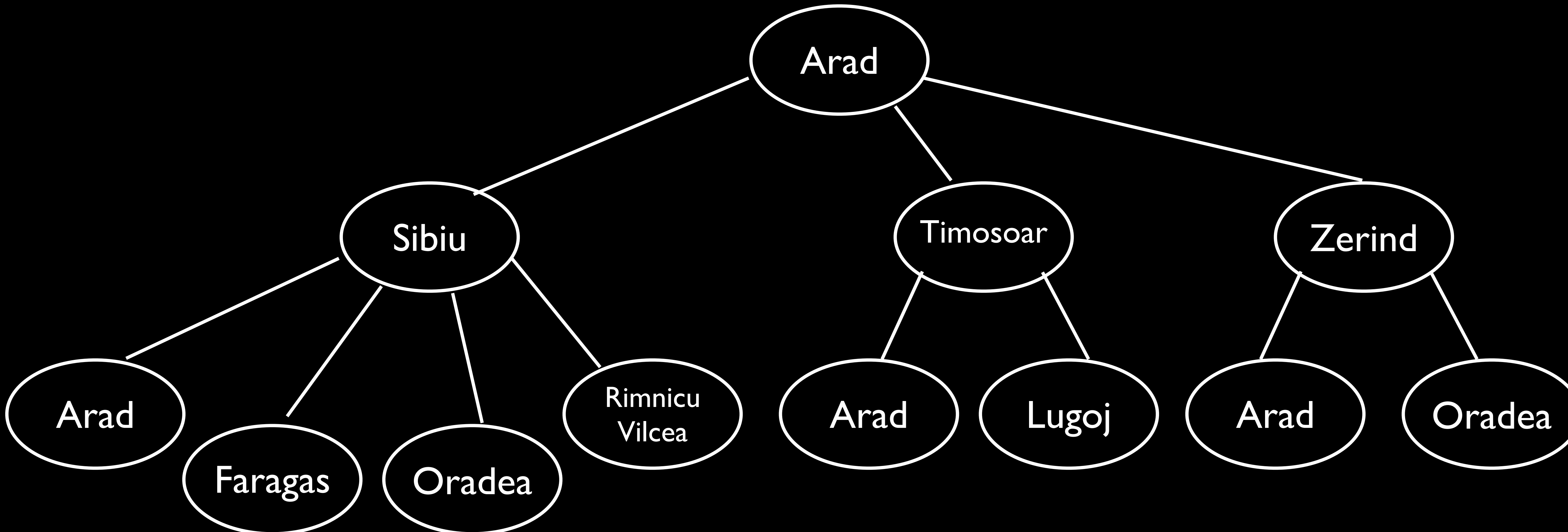
Oradea
Neamt
71
87
Zerind
Iasi
75
151
92
Arad
140
Vaslui
Sibiu
99
Fagaras
118
142
80
Rimnicu Vilcea
211
Timosoara
98
Hirsova
97
Pitesti
85
Urziceni
111
Logoj
101
Bucharest
146
86
70
Mehadia
90
138
75
Eforie
Drobeta
120
Giurgiu
Craiova

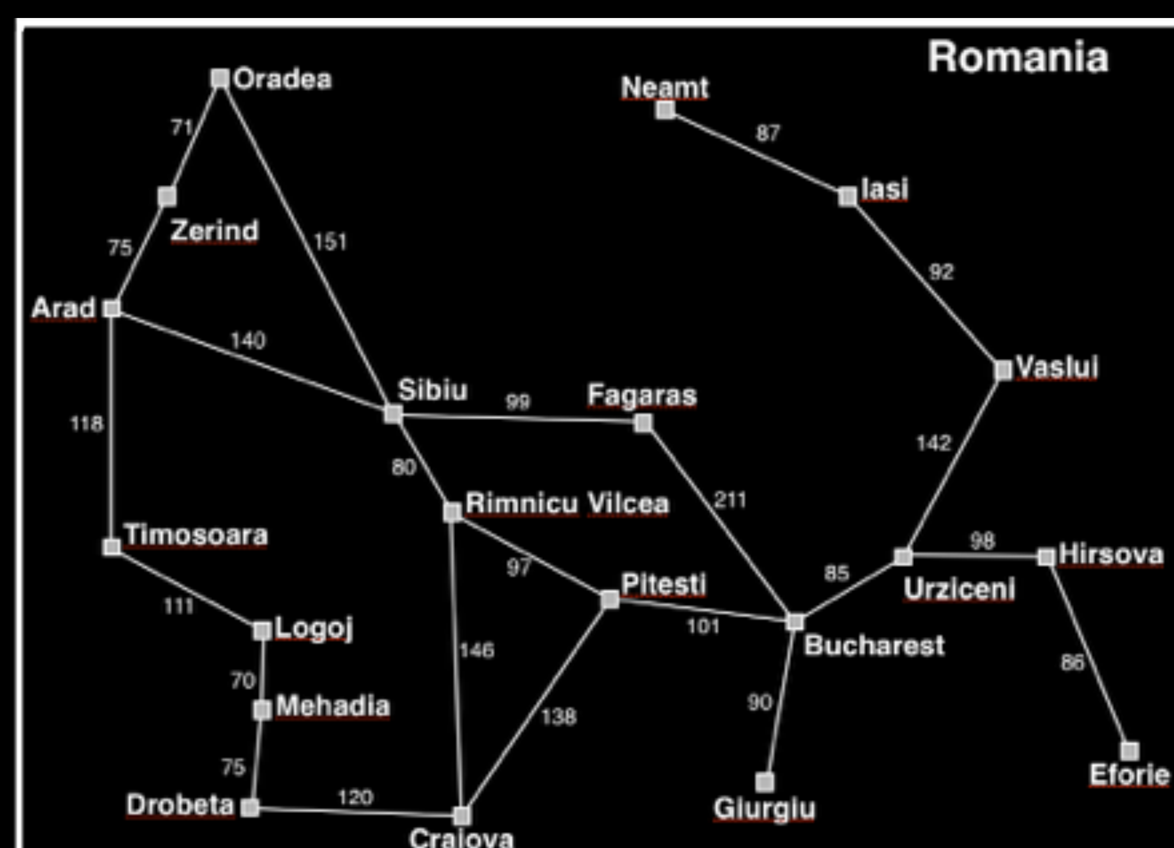

Arad

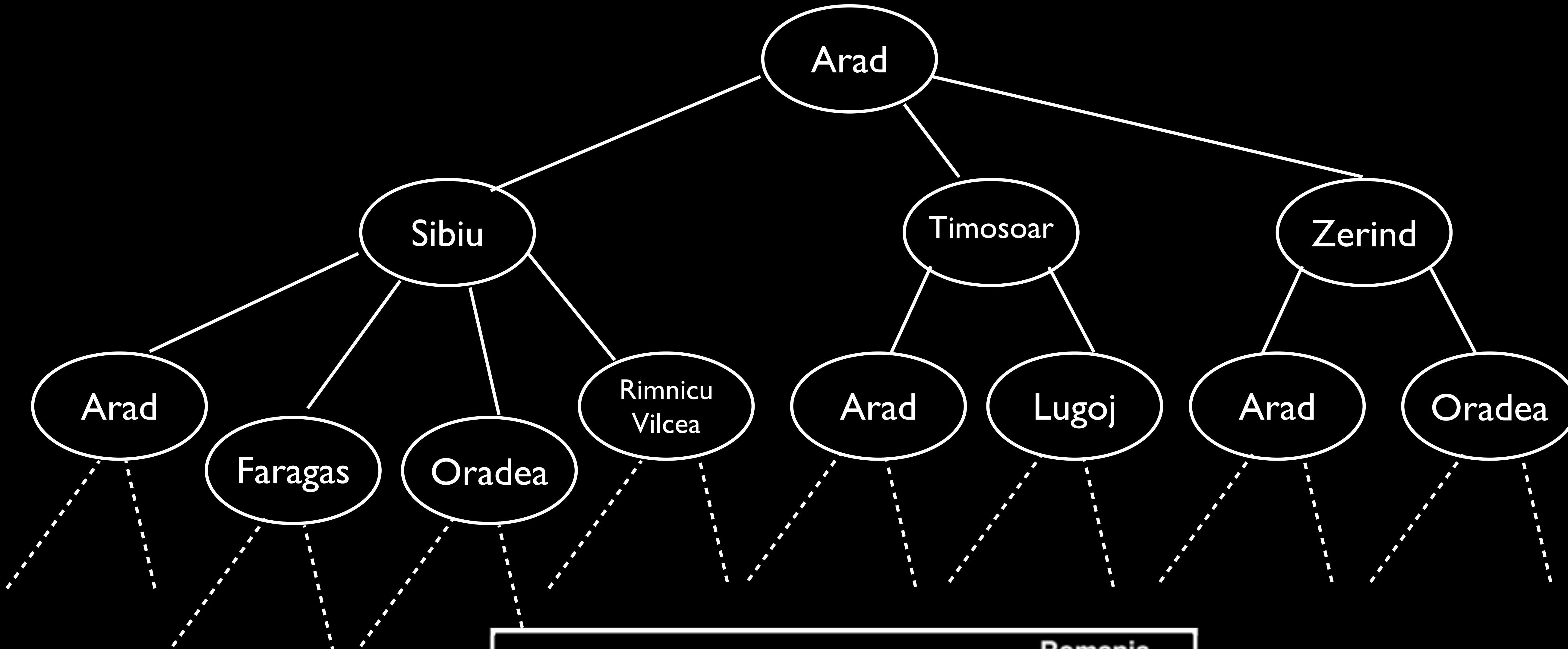

Romania
Oradea
Neamt
71
87
Iasi
Zerind
151
75
92
Arad
140
Vaslui
Sibiu
99
Fagaras
118
80
142
Rimnicu Vilcea
211
Timosoara
98
Hirsova
97
85
Pitesti
Urziceni
111
Logoj
101
Bucharest
146
86
70
Mehadia
90
138
75
Eforie
Drobeta
120
Giurgiu
Cralova

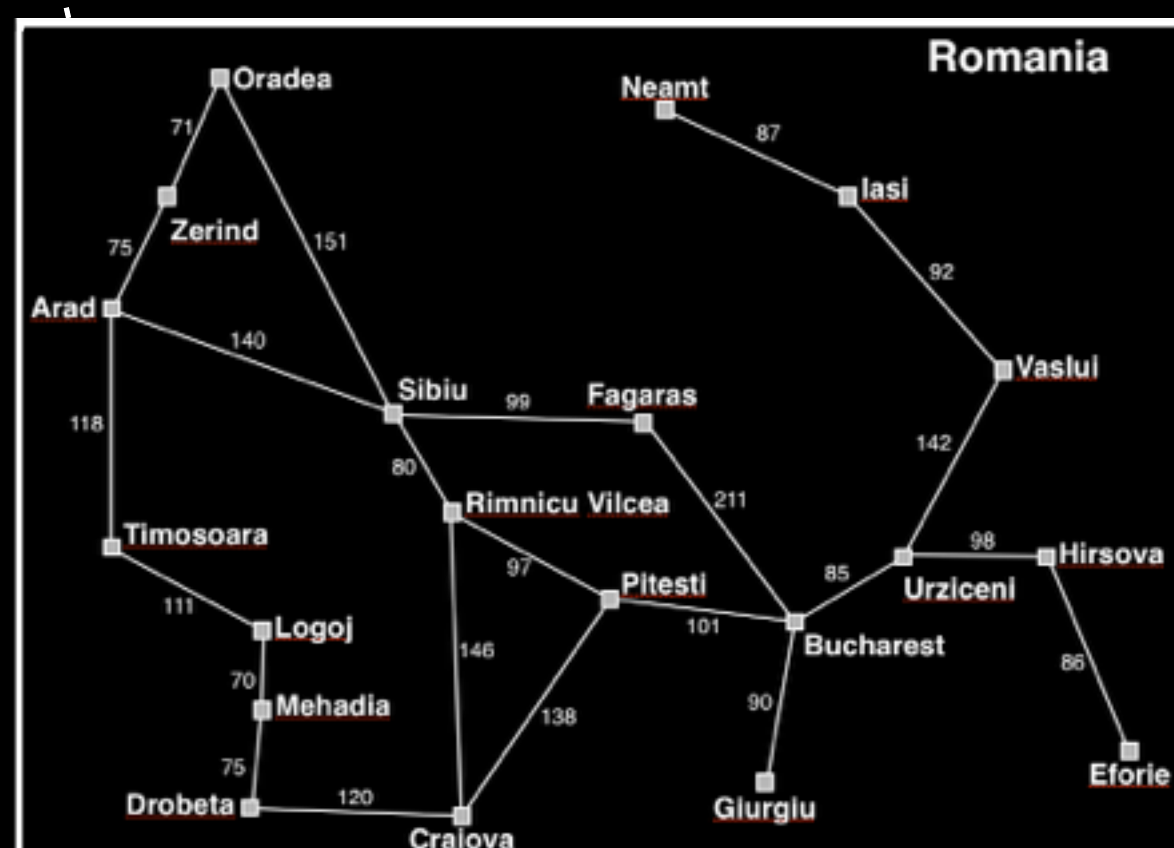

Arad
Sibiu
Timosoar
Zerind


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

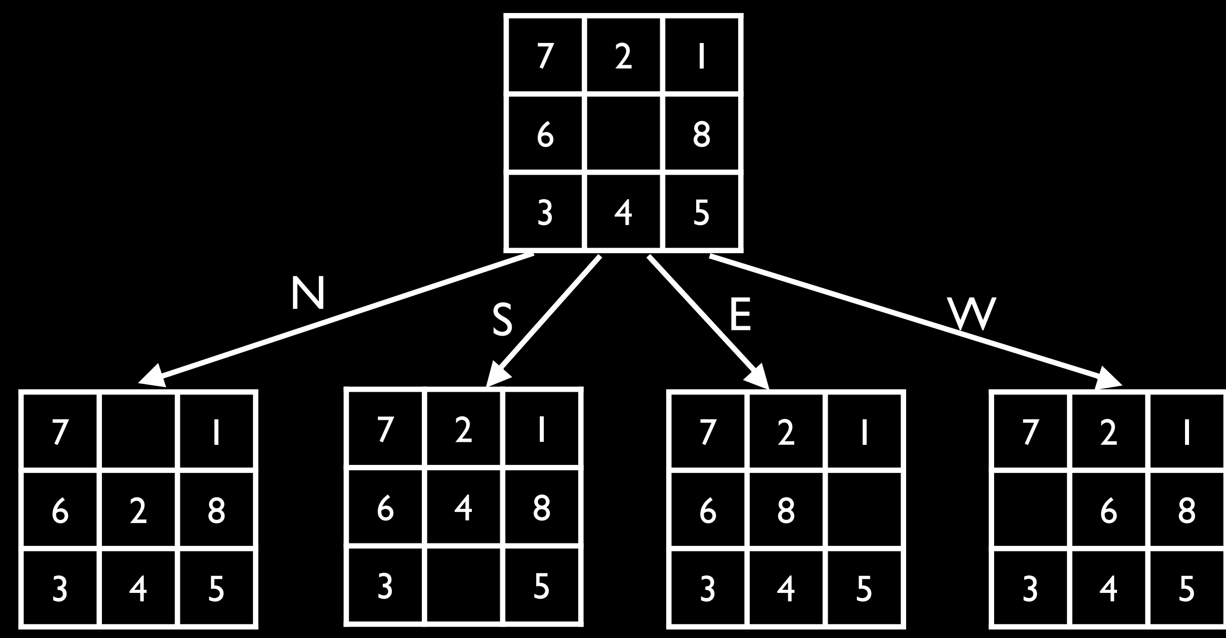

Arad
Sibiu
Timosoar
Zerind
Arad
Faragas
Oradea
Rimnicu Vilcea
Arad
Lugoj
Arad
Oradea


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

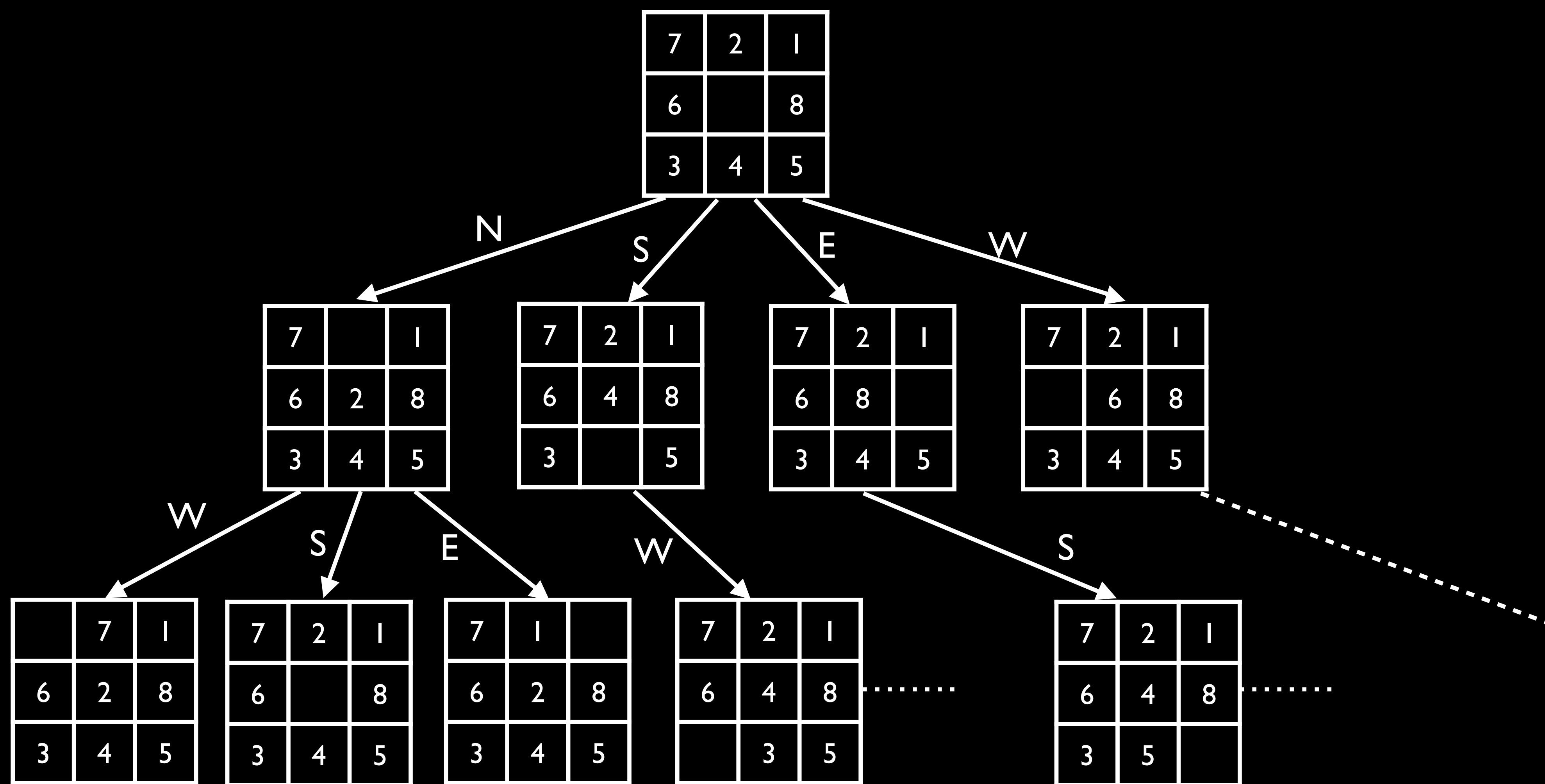

Arad
Sibiu
Timosoar
Zerind
Arad
Faragas
Oradea
Rimnicu Vilcea
Arad
Lugoj
Arad
Oradea

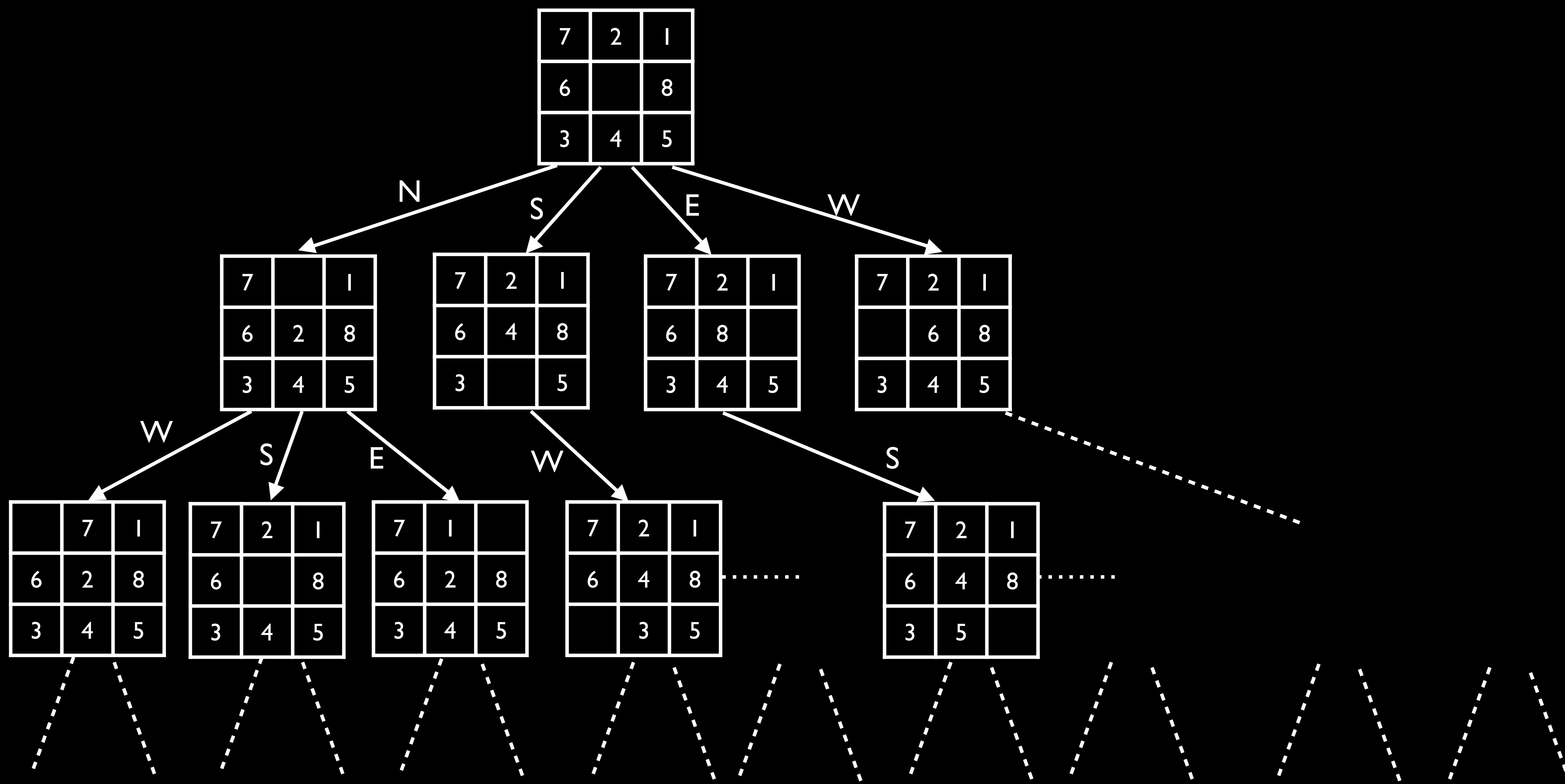

Romania
Oradea
Neamt
71
87
Iasi
Zerind
151
75
92
Arad
140
Vaslui
Sibiu
99
Fagaras
118
80
Rimnicu Vilcea
211
142
Timosoara
98
Hirsova
97
Pitesti
85
Urziceni
111
Logoj
101
Bucharest
146
86
70
Mehadia
90
138
75
Eforie
Drobeta
120
Giurgiu
Craiova

7	2	1
6		8
3	4	5



7 2 1
6 8
3 4 5
N
S
E
W
7 1
6 2 8
3 4 5
7 2 1
6 4 8
3 5
7 2 1
6 8
3 4 5
7 2 1
6 8
3 4 5


N
S
E
W
W
S
E
W
S


7 2 1
6 8
3 4 5
N
S
E
W
7 1
6 2 8
3 4 5
7 2 1
6 4 8
3 5
7 2 1
6 8
3 4 5
7 2 1
6 8
3 4 5
W
S
E
W
S
7 1
6 2 8
3 4 5
7 2 1
6 8
3 4 5
7 1
6 2 8
3 4 5
7 2 1
6 4 8
3 5
7 2 1
6 4 8
3 5

State-Space Search

- Start with initial state
- Generate successor states by applying applicable actions
- Until you find a goal state

```
Solution treeSearch(Problem p) {
  Set<Node> frontier = new Set<Node>(p.getInitialState());

  while (true) {
    if (frontier.isEmpty()) {
      return null;
    }
    Node node = frontier.selectOne();
    if (p.isGoalState(node.getState())) {
      return n.getSolution();
    }

    for (Node n : node.expand()) {

        frontier.add(n);

    }
  }
}
```

```
Solution graphSearch(Problem p) {
  Set<Node> frontier = new Set<Node>(p.getInitialState());
  Set<Node> explored = new Set<Node>();
  while (true) {
    if (frontier.isEmpty()) {
      return null;
    }
    Node node = frontier.selectOne();
    if (p.isGoalState(node.getState())) {
      return n.getSolution();
    }
    explored.add(node);
    for (Node n : node.expand()) {
      if (!explored.contains(n)) {
        frontier.add(n);
      }
    }
  }
}
```

Summary

- General-purpose algorithm for solving any problem that can be represented using states and actions that transition between them
- State-space search framework will allow us to explore and compare alternatives

```
Solution graphSearch(Problem p) {
  Set<Node> frontier = new Set<Node>(p.getInitialState());
  Set<Node> explored = new Set<Node>();
  while (true) {
    if (frontier.isEmpty()) {
      return null;
    }
    Node node = frontier.selectOne();
    if (p.isGoalState(node.getState())) {
      return n.getSolution();
    }
    explored.add(node);
    for (Node n : node.expand()) {
      if (!explored.contains(n)) {
        frontier.add(n);
      }
    }
  }
}
```

Next Class

Your Homework this Weekend:
Read Chapters 2 and 3