Generic Search Algorithm and examples

CS 3600
Intro to Artificial Intelligence

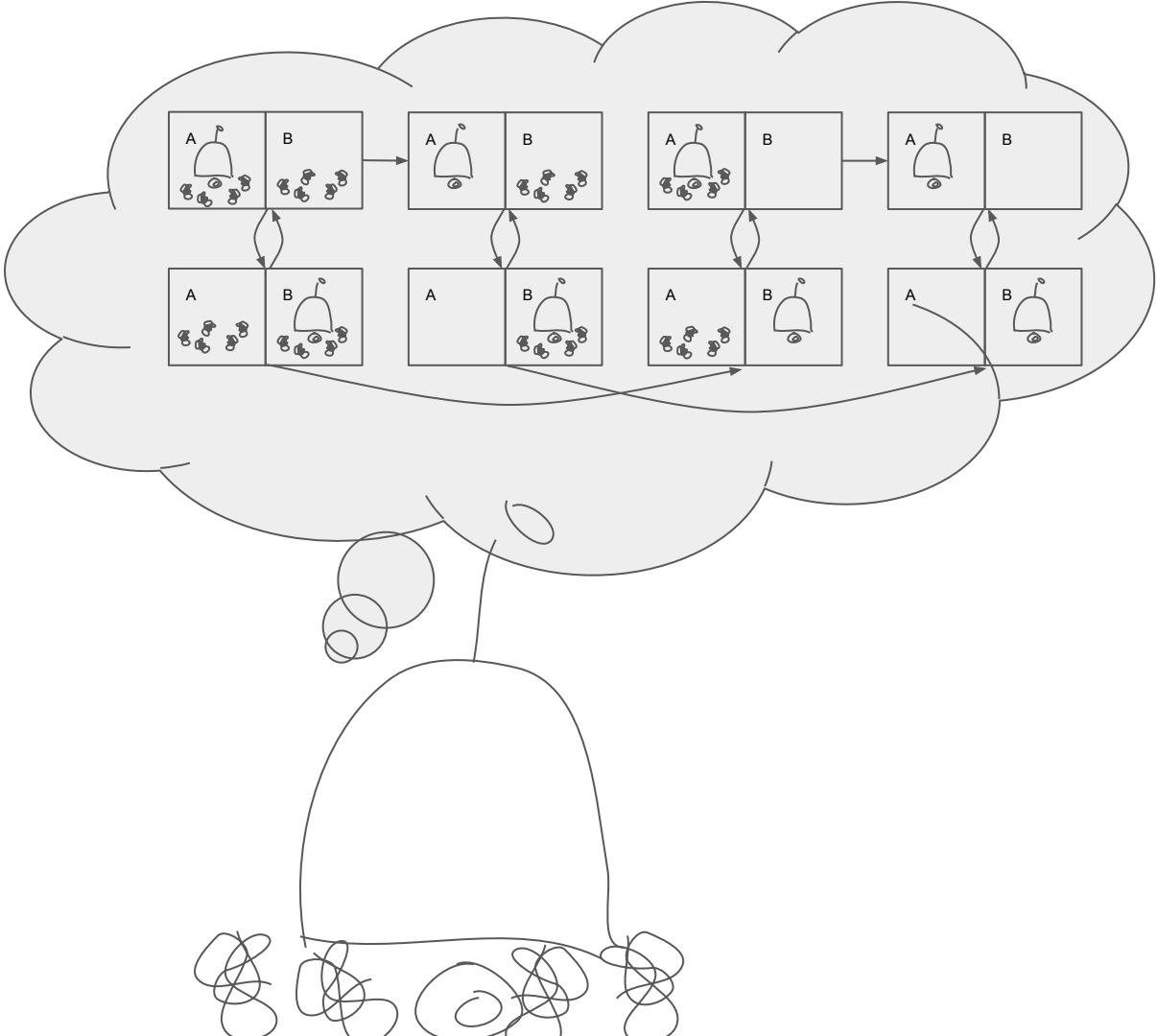
Recall

A Search Agent solves problems by

- Formulating a **state space** and **goal**
- Searching the state space until it has found a **sequence of actions** from the initial state to the goal
- **Executing** each action in turn

We know several ways of searching a state space

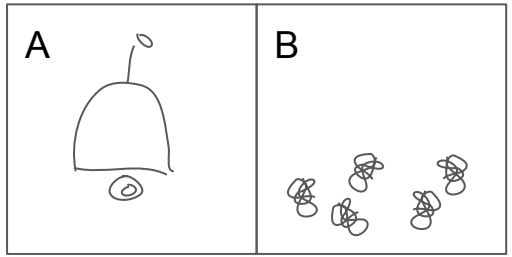
- Depth First Search
- Breadth First Search
- Uniform Cost Search

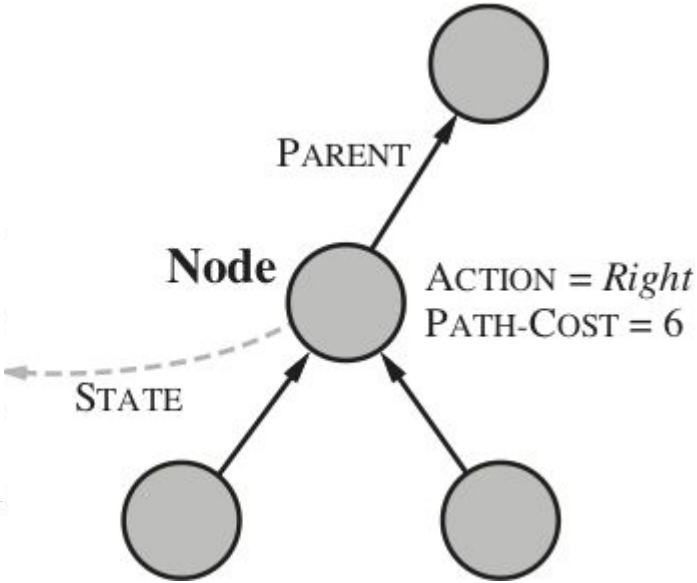


Search Node data structure

We've looked at the state space in the abstract as states connected by actions, but we need some additional bookkeeping to implement our search

- Parent **node** (for backtracking)
- Transition **action**
- Cost of shortest known path from start to this state, **g**
- The actual **state**





'loc' 'A-clean' 'B-clean'

```
node=dict()
node['state'] = ('A', True, False)
node['parent'] = parent_node #<-another node object
node['action'] = 'Clean'
node['g'] = parent_node['g']+clean_cost
```

Generic Search Algorithm

```
Initialize ‘current’ node to start state
Initialize ‘closed’ as an empty list
Initialize ‘open’ as one of (stack, queue, priority queue)
while not( current[‘state’] is goal state):
    Add current[‘state’] to closed
    successors = successors of current[‘state’]
    for s in successors:
        if not(s.state is in closed):
            Add new node for state to open
    current = next node in open that’s not in closed
path = list()
while current has a parent:
    Add current[‘action’] to the front of path
    current = current[‘parent’]
return path
```

Generic Search Algorithm - notes

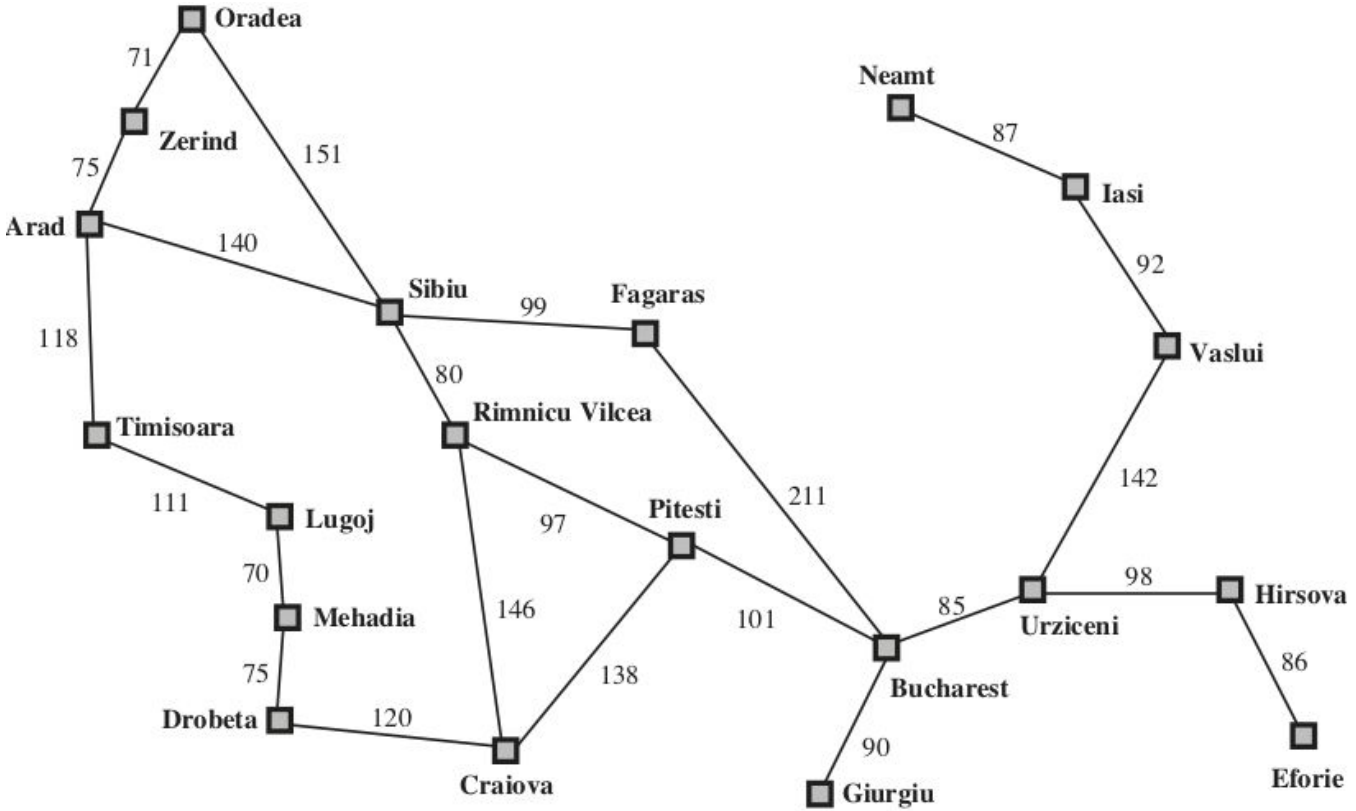
Can implement BFS, DFS, or UCS by picking the right data structure for `open`

- DFS: stack
- BFS: queue
- UCS: priority queue, with priority being `node['g']`

This version is similar to the `Graph-Search` algorithm (Fig 3.7) in the text, with some minor changes

```
Initialize 'current' node to start state
Initialize 'closed' as an empty list
Initialize 'open' as one of (stack, queue, priority queue)
while not( current['state'] is goal state):
       Add current['state'] to closed
       successors = successors of current['state']
       for s in successors:
              if not(s.state is in closed):
                     Add new node for state to open
       current = next node in open
path = list()
while current has a parent:
       Add current['action'] to the front of path
       current = current['parent']
return path
```

Romania


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

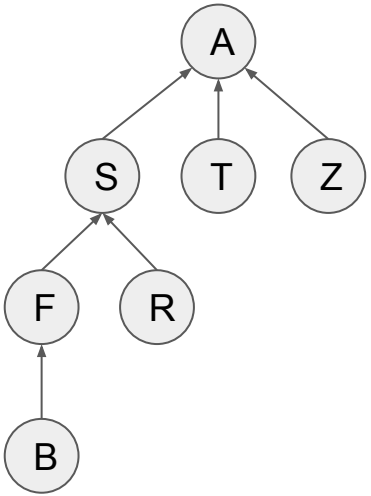
Romania - DFS

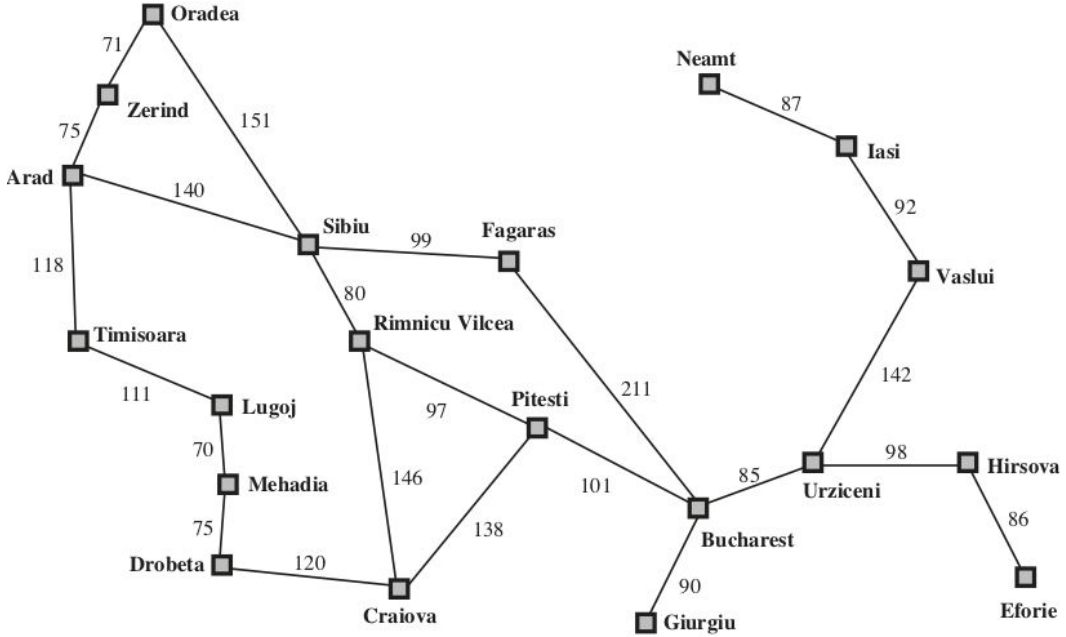
Current: Arad
Open: [**S,T,Z**]
Closed: [A]

Current: Sibiu
Open: [**F**,**O**,**R**,T,Z]
Closed: [A,S]

Current: Fagaras
Open: [**B**,O,R,T,Z]
Closed: [A,S,F]

Current: Bucharest
Open:[**G**,**P**,**U**,O,R,T,Z]
Closed: [B,A,S,F]





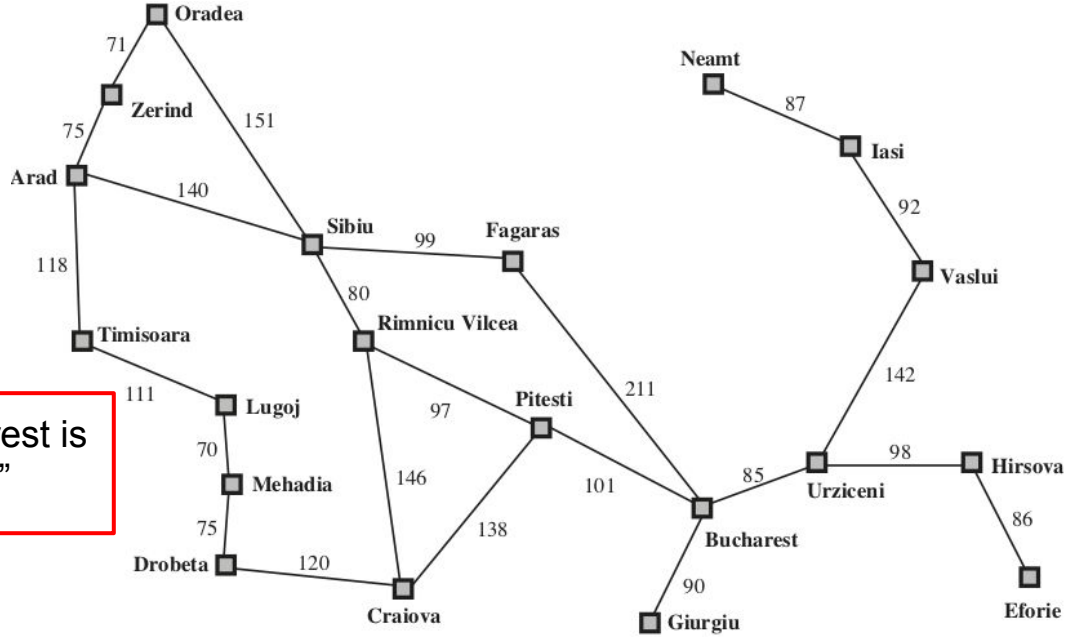
Solution: [A->S, S->F, F->B]
Cost: 450

Romania - DFS (reversed alpha)

For DFS: Order of expansion can have a big impact on number of nodes explored, and the final path returned!

Arad	Rimnicu Vilcea
[**Z,T,S**]	[**P**,**C**,F,T,S]
[A]	[A,Z,O,S,R]
Zerind	Pitesti
[**O**,T,S]	[**C**,**B**,C,F,T,S]
[A,Z]	[A,Z,O,S,R,P]
Oradea	Craiova
[**S**,T,S]	[**D**,B,C,F,T,S]
[A,Z,O]	[A,Z,O,S,R,P,C]
Sibiu	Drobeta
[**R**,**F**,T,S]	[**M**,B,C,F,T,S]
[A,Z,O,S]	[A,Z,O,S,R,P,C,D]

Bucharest is "buried"



Solution (eventually): [A->Z, Z->O, O->S, S->R, R->P, P->B]
Cost: 575

Romania - BFS

Arad
[**S,T,Z**]
[A]

Fagaras
[O,R,L,O,**B**]
[A,S,T,Z,F]

Sibiu
[T,Z,**F,O,R**]
[A,S]

Oradea
[R,L,O,B]
[A,S,T,Z,F,O]

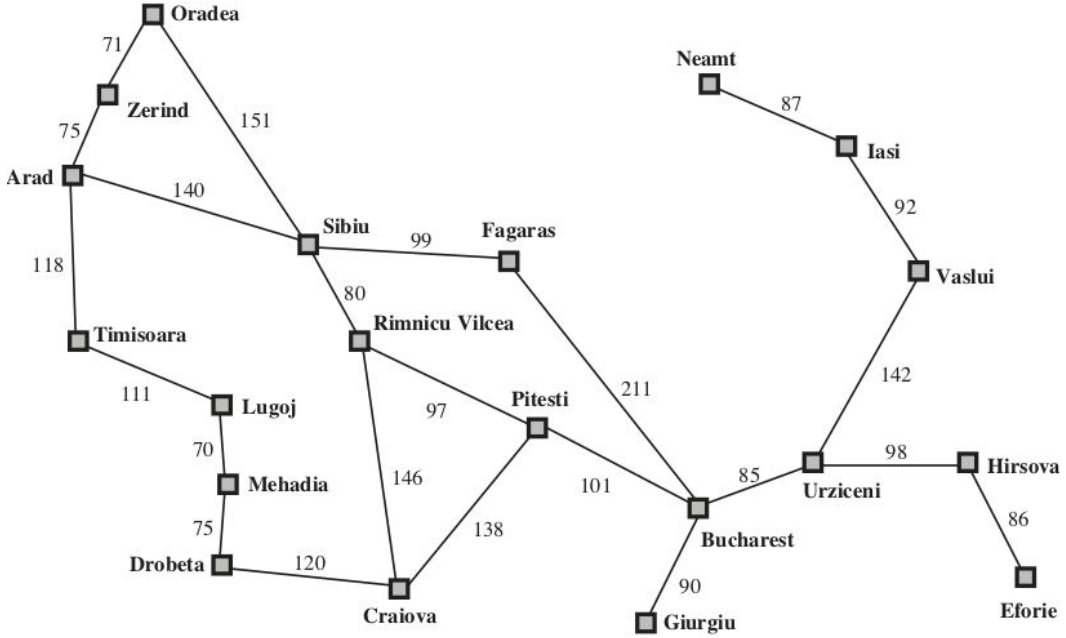
Timisoara
[Z,F,O,R,**L**]
[A,S,T]

Rimnicu Vilcea
[L,O,B,**C**,**P**]
[A,S,T,Z,F,O,R]

Zerind
[F,O,R,L,**O**]
[A,S,T,Z]

Lugoj
[O,B,C,P,**M**]
[A,S,T,Z,F,O,R,L]

Bucharest
[C,P,M,**G**,**U**]
[A,S,T,Z,F,R,L,O,B]



Solution: [A->S, S->F, F->B]
Cost: 450

Romania - UCS

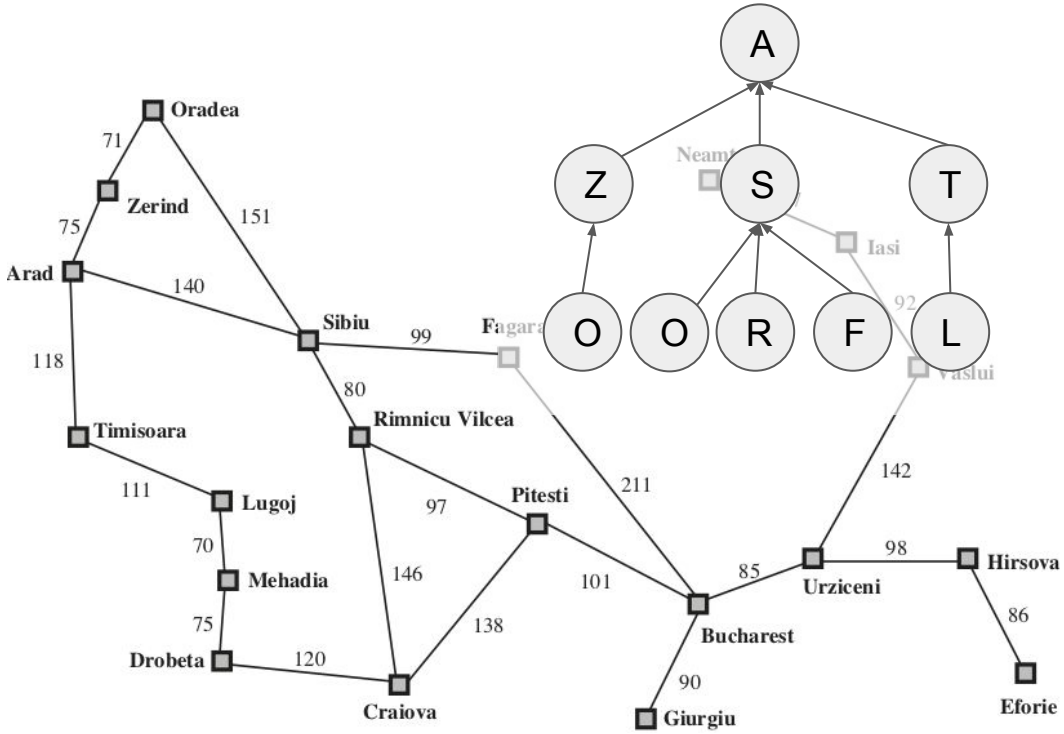
Arad (0)
[**Z**(75), **T**(118), **S**(140)]
[A]

Zerind (75)
[T(118), S(140), **O**(146)]
[A,Z]

Timisoara (118)
[S(140), O(146), **L**(229)]
[A,Z,T]

Sibiu (140)
[O(146),**R**(220),L(229),**F**(239),**O**(291)]
[A,Z,T,S]

Oradea (146)
[R(220),L(229),F(239),O(291)]
[A,Z,T,S,O]



Romania - UCS

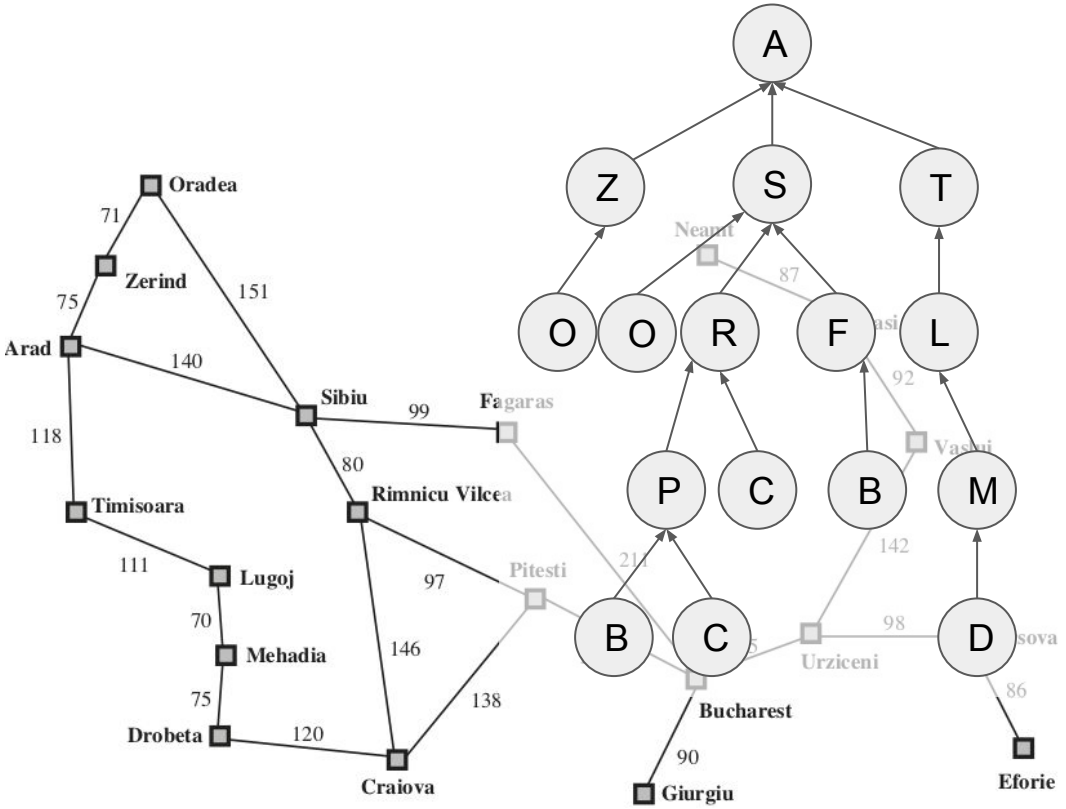
Rimnicu Vilcea (220)
[L(229),F(239),O(291),**P**(317),**C**(366)]
[A,Z,T,S,O,R]

Lugoj(229)
[F(239),O(291),**M**(299),P(317),C(366)]
[A,Z,T,S,O,R,L]

Fagaras(239)
[O(291),M(299),P(317),C(366),**B**(450)]
[A,Z,T,S,O,R,L,F]

Mehadia(299)
[P(317),C(366),**D**(374),B(450)]
[A,Z,T,S,O,R,L,F,M]

Pitesti(317)
[C(366),D(374),**B**(418),B(450),**C**(455)]
[A,Z,T,S,O,R,L,F,M,P]

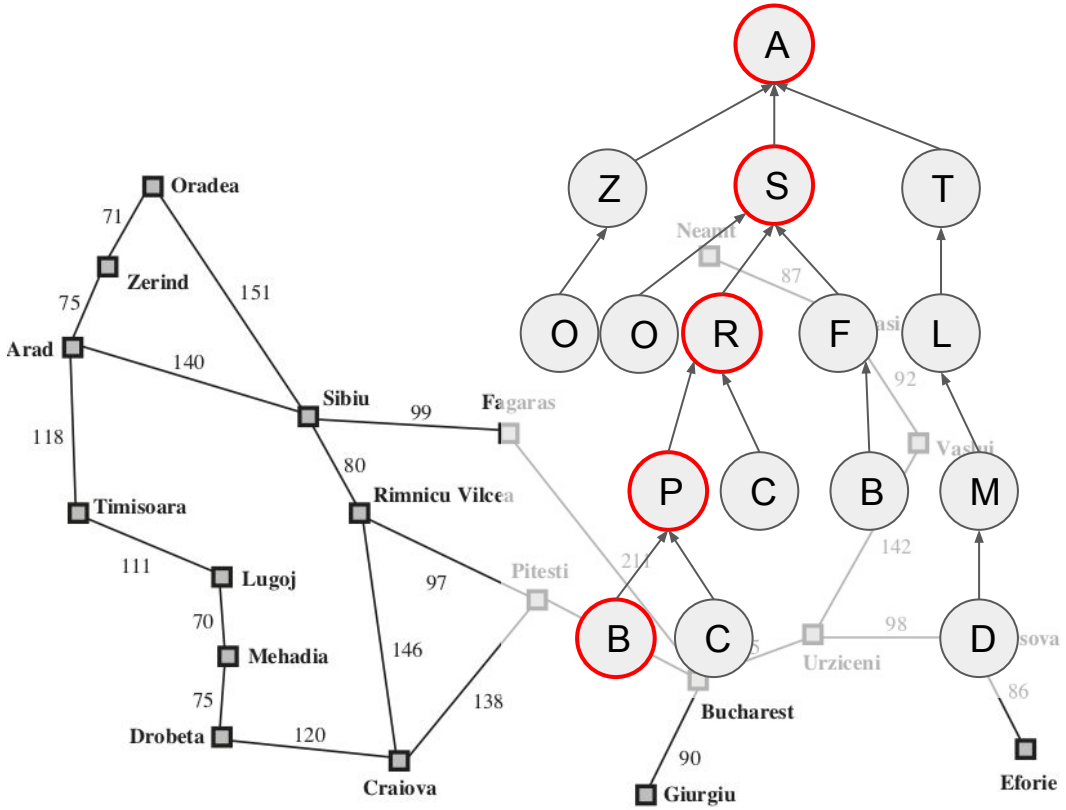


Romania - UCS

Craiova(366)
[D(374),B(418),B(450),C(455),**D**(486)]
[A,Z,T,S,O,R,L,F,M,P,C]

Drobeta(374)
[B(418),B(450),C(455),D(486)]
[A,Z,T,S,O,R,L,F,M,P,C,D]

Bucharest(418)
[B(450),C(455),D(486),**U**(501),**G**(506)]
[A,Z,T,S,O,R,L,F,M,P,C,D]

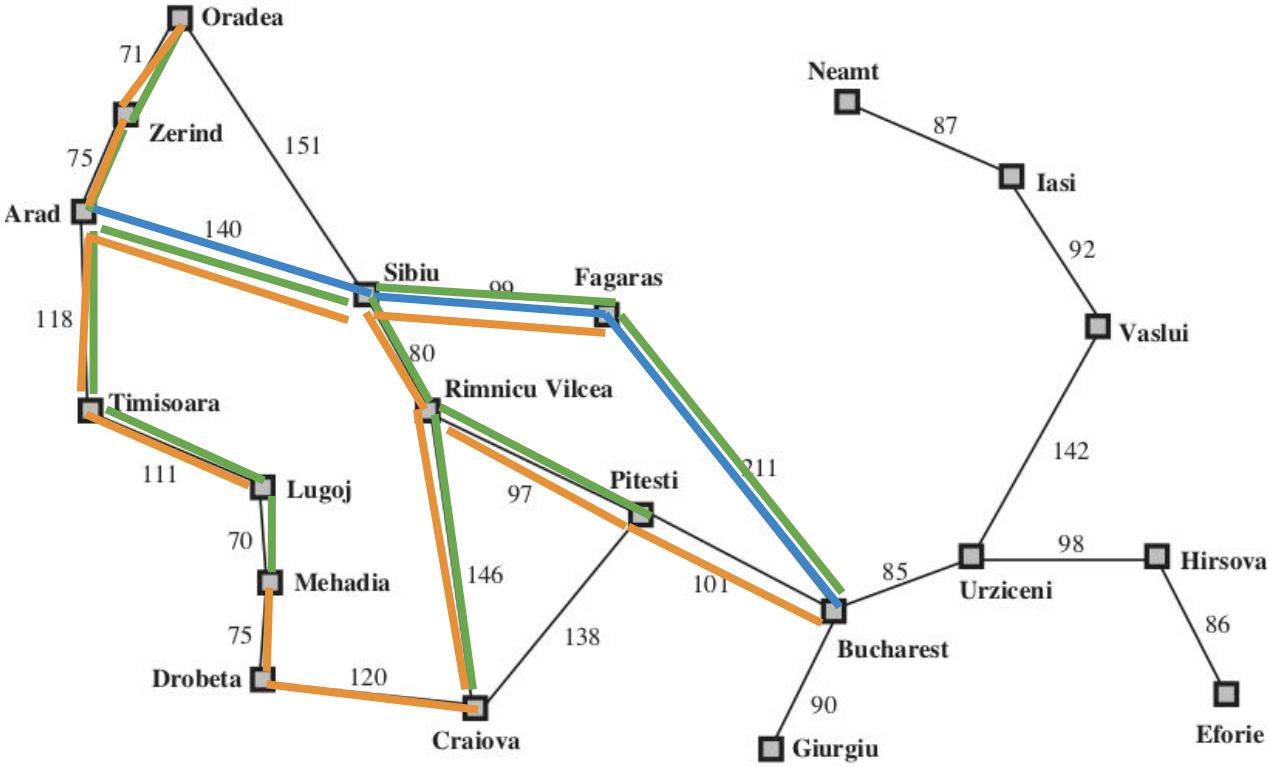


Solution: [A->S, S->R, R->P, P->B]
Cost: 418

Comparing DFS,BFS, and UCS

- DFS was highly dependent on the order that child nodes were explored
- BFS took more iterations than DFS, but less than UCS and DFS-reverse-alpha
- DFS and BFS both found the same (sub-optimal solution)
- UCS found the best solution, but took as long as DFS-reverse-alpha

How can we improve?



Uninformed vs Informed Search

Uninformed Search

- Does not use any domain specific knowledge
- Only looks at **edges** and **edge costs**, the problem is completely abstract
- We can find the optimal path (UCS) but it might take a long time to compute

Informed Search

- Formally represent domain knowledge that can **guide** the search in “good” directions
- Leverage the optimality and completeness guarantees from UCS if possible

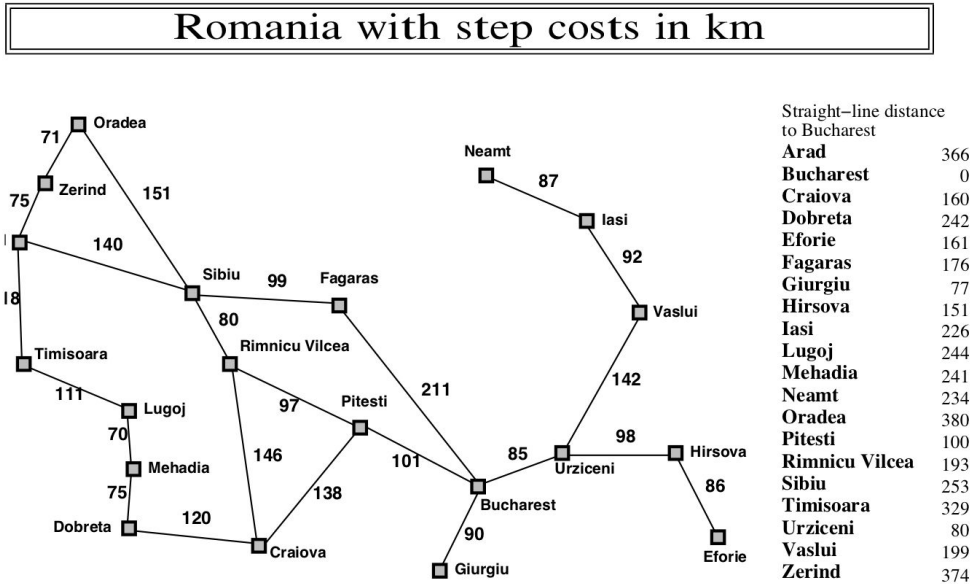
A* Search preview

Something like UCS, but with a little “hint” about the right direction to go

Priority queue with priority
f(s) = g(s)+**h**(s)

h(s): “Heuristic” function, that estimates the cost-to-go from s

Note: h(s) should be easier to compute than solving the original problem!



Summary and preview

Wrapping up

- We can implement DFS, BFS, and UCS with a **single algorithm**, and choose the behavior we want by picking the appropriate data structure
- Examples of applying search

For next time

- A* Search
- Admissibility, Consistency, and Optimality