Real Lecture 23: More Graphs: DFS, BFS and shortest paths

MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

CSAIL

Search in a Graph

Basic search algorithm

```
def reachableNodes(origin, graph, empty):
    S = empty  #collection of pending nodes
    V = set({})  #set of visited nodes
    S.add(origin)
    V.add(origin)
    while not S.empty():
        n = S.pop()
        print str(n)
        for edge in graph.edgesOf(n):
            dest = edge.getDestination()
            if dest not in V:
                V.add(dest)
                S.add(dest)
    return V
```

Breath First Search

Keep the collection of nodes to visit as a FIFO *queue*

```
class Fifo(Collection):
    def __init__(self):
        self.lst = []
    def add(self, x):
        self.lst.append(x)
    def empty(self):
        return len(self.lst)==0
    def pop(self):
        return self.lst.pop(0)
```

```
data = cl.getDataPoints('responses2.csv')

(g,nodes) = cg.getGraphFromData(data)

reachableNodes(nodes[0], g, Fifo())
```

Depth First Search

Keep the collection of nodes to visit as a LIFO *stack*

```
class Lifo(Collection):
    def __init__(self):
        self.lst = []
    def add(self, x):
        self.lst.append(x)
    def empty(self):
        return len(self.lst)==0
    def pop(self):
        return self.lst.pop()
```

```
data = cl.getDataPoints('responses2.csv')

(g,nodes) = cg.getGraphFromData(data)

reachableNodes(nodes[0], g, Fifo())
```

Shortest path

For an unweighted graph

- BFS will find the shortest path
- Argument:
  - Every element in the wait queue was reached through the shortest path
  - At the entry point of the outer loop, the length of the path to any unvisited element is greater than or equal to the length of the path to any element in the wait queue
  - We can prove this by induction!
    - The property is preserved because the FIFO queue ensures that we always pick the 'nearest' node to visit next

Shortest path

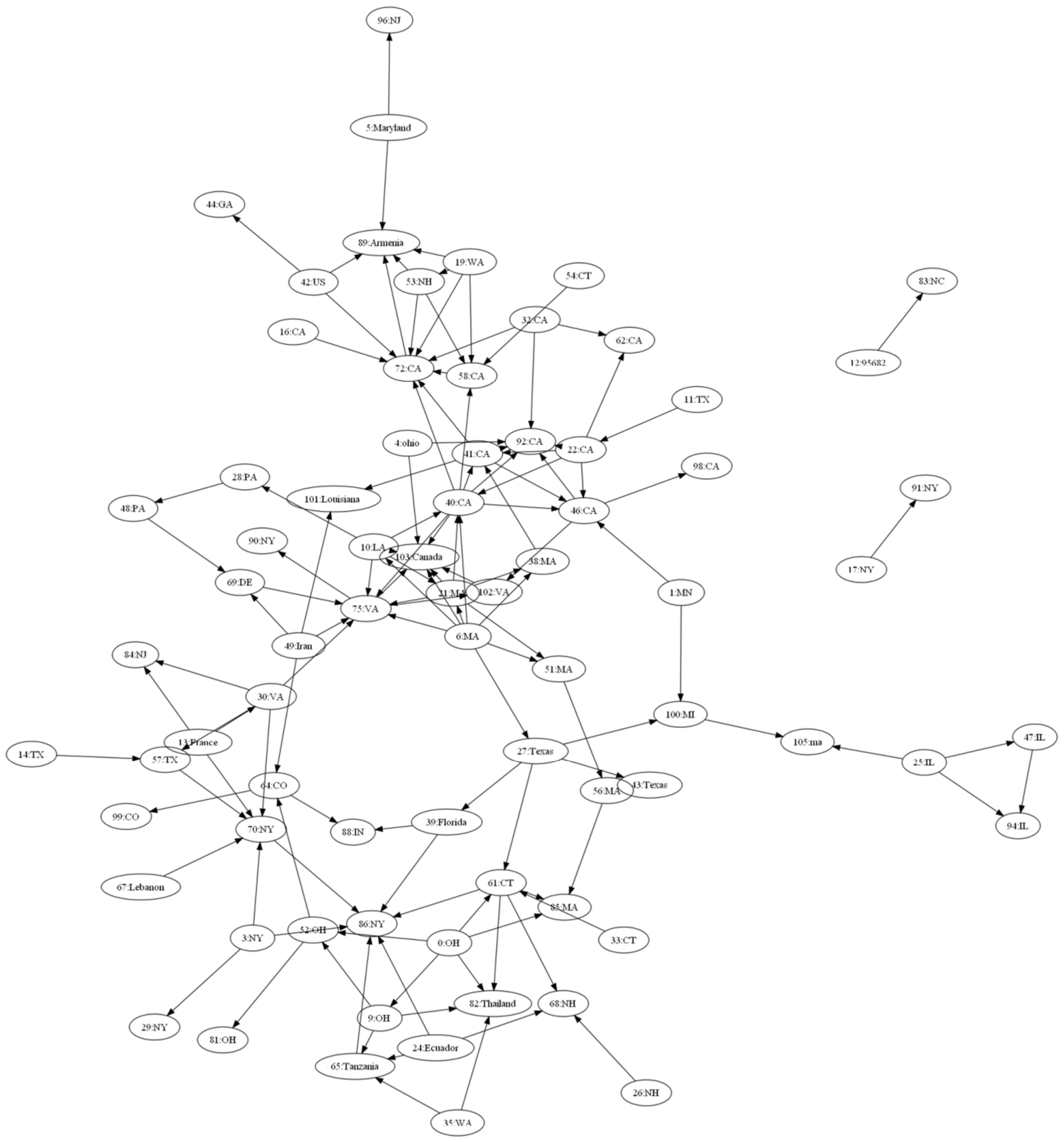
What if the graph has weights?

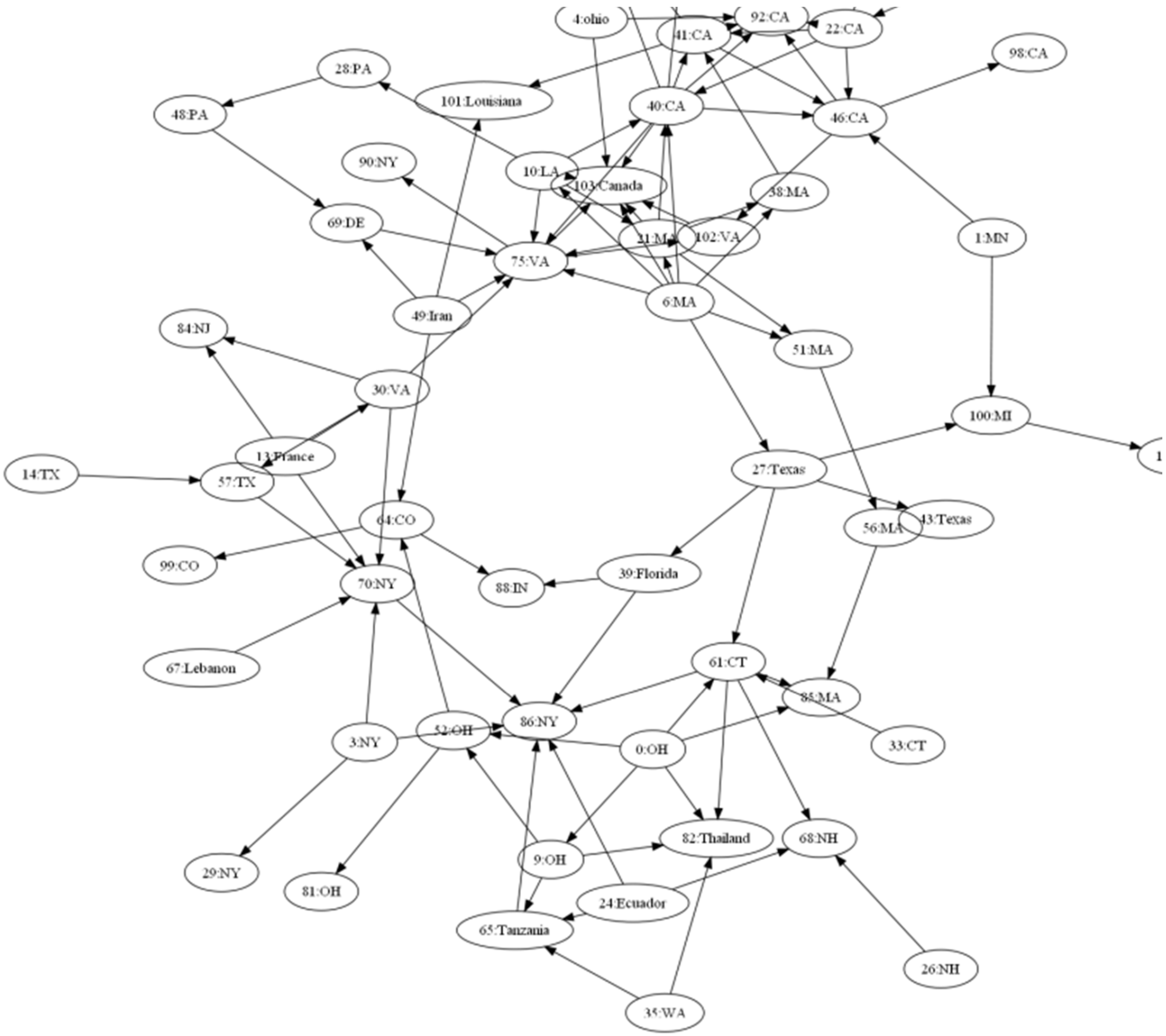
- The same idea as BFS, but we can no longer assume that the head of the FIFO queue is the nearest node
- We need to keep a distance estimate for each node in the collection of nodes to visit and visit the one with the smallest distance
- This is called Dijkstra's algorithm

The Code

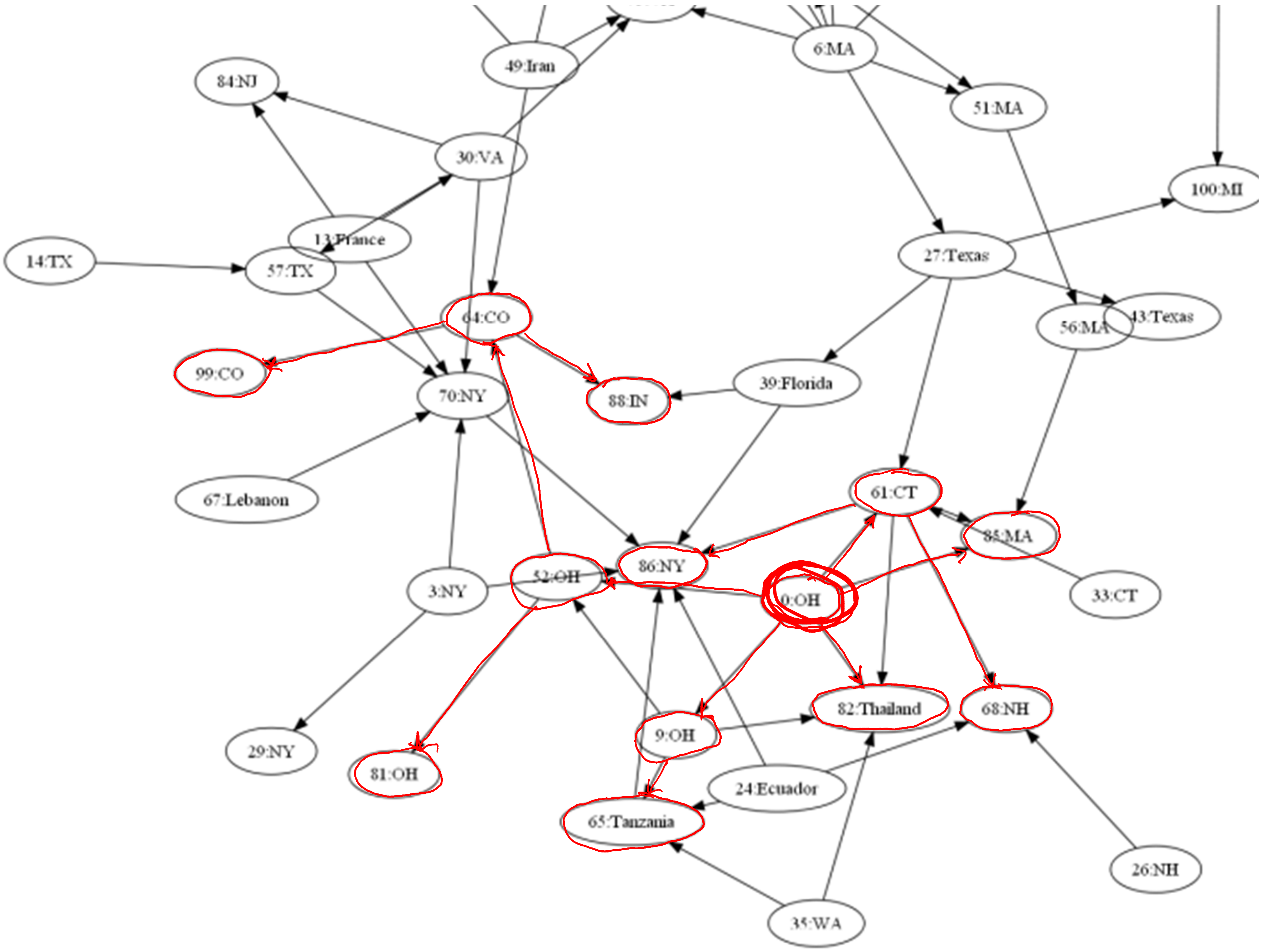
The search code for the graph can be found here: http://bit.ly/13GQvXz

- The code to build the graph from the class data is here: http://bit.ly/YxpBR7
- The graph definition itself is here: http://bit.ly/11ocFRp
- You will also need the definition of the data itself from here: http://bit.ly/ZCwSQg
- Remember all the files must be in the same folder


96:NJ
5:Maryland
44:GA
89:Armenia
19:WA
42:US
53:NH
54:CT
83:NC
32:CA
16:CA
62:CA
12:95682
72:CA
58:CA
11:TX
4:ohio
92:CA
22:CA
41:CA
98:CA
28:PA
101:Louisiana
91:NY
48:PA
40:CA
46:CA
90:NY
10:LA
103:Canada
38:MA
17:NY
69:DE
21:MA
102:VA
1:MN
75:VA
49:Iran
6:MA
84:NJ
51:MA
30:VA
100:MI
13:France
105:ma
47:IL
14:TX
57:TX
27:Texas
25:IL
43:Texas
56:MA
64:CO
99:CO
39:Florida
88:IN
70:NY
94:IL
67:Lebanon
61:CT
85:MA
52:OH
86:NY
3:NY
0:OH
33:CT
82:Thailand
68:NH
9:OH
29:NY
81:OH
24:Ecuador
65:Tanzania
26:NH
35:WA



4:ohio
92:CA
41:CA
22:CA
98:CA
28:PA
101:Louisiana
40:CA
46:CA
48:PA
90:NY
10:LA
103:Canada
38:MA
69:DE
21:MA
102:VA
1:MN
75:VA
6:MA
49:Iran
84:NJ
51:MA
30:VA
100:MI
13:France
14:TX
57:TX
27:Texas
64:CO
56:MA
43:Texas
99:CO
70:NY
39:Florida
88:IN
67:Lebanon
61:CT
85:MA
86:NY
52:OH
3:NY
0:OH
33:CT
82:Thailand
68:NH
9:OH
29:NY
81:OH
24:Ecuador
65:Tanzania
26:NH
35:WA


49:Iran
6:MA
84:NJ
51:MA
30:VA
100:MI
13:France
14:TX
57:TX
27:Texas
64:CO
56:MA
43:Texas
99:CO
70:NY
88:IN
39:Florida
67:Lebanon
61:CT
85:MA
86:NY
52:OH
3:NY
0:OH
33:CT
82:Thailand
68:NH
9:OH
29:NY
81:OH
24:Ecuador
65:Tanzania
26:NH
35:WA