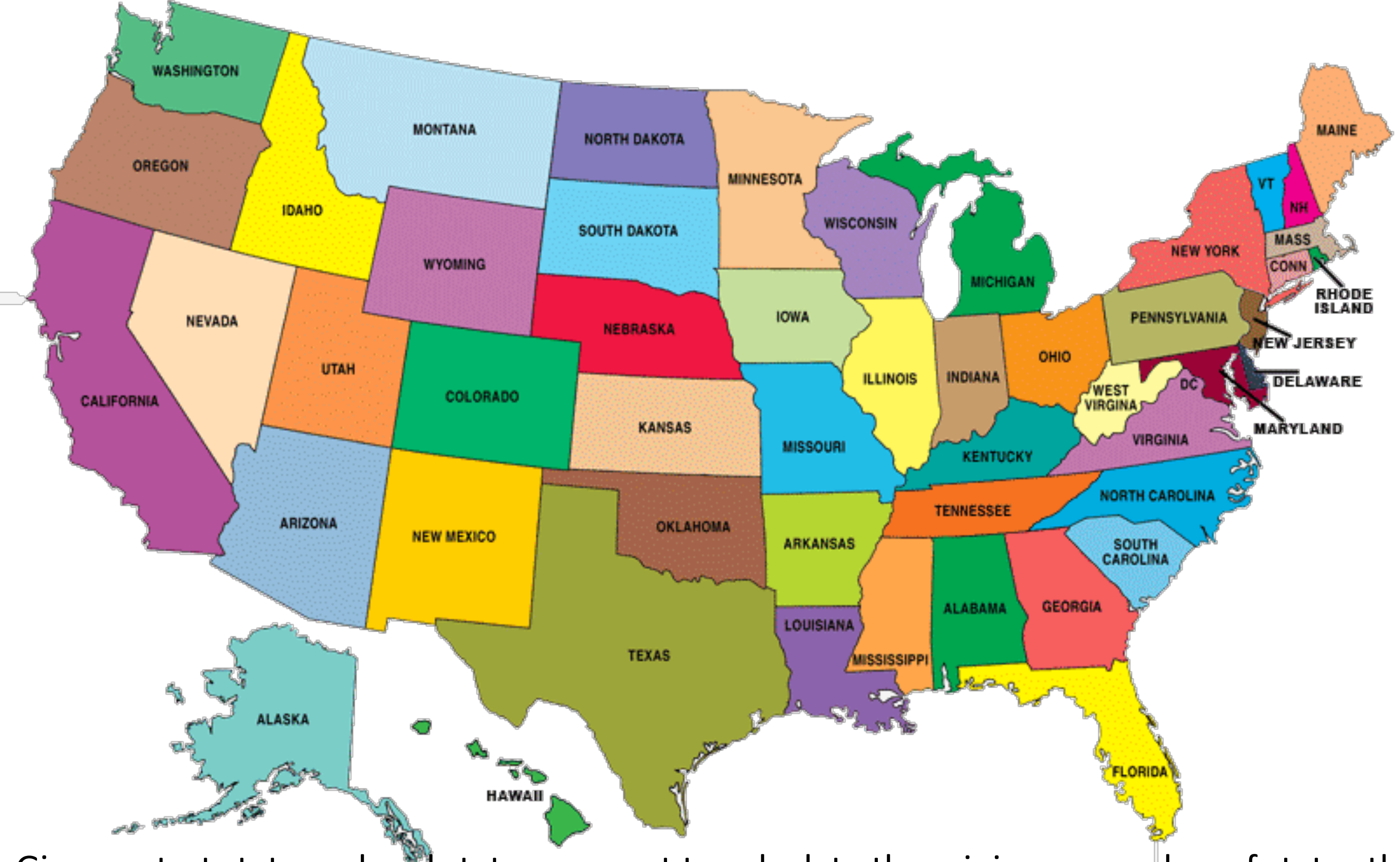
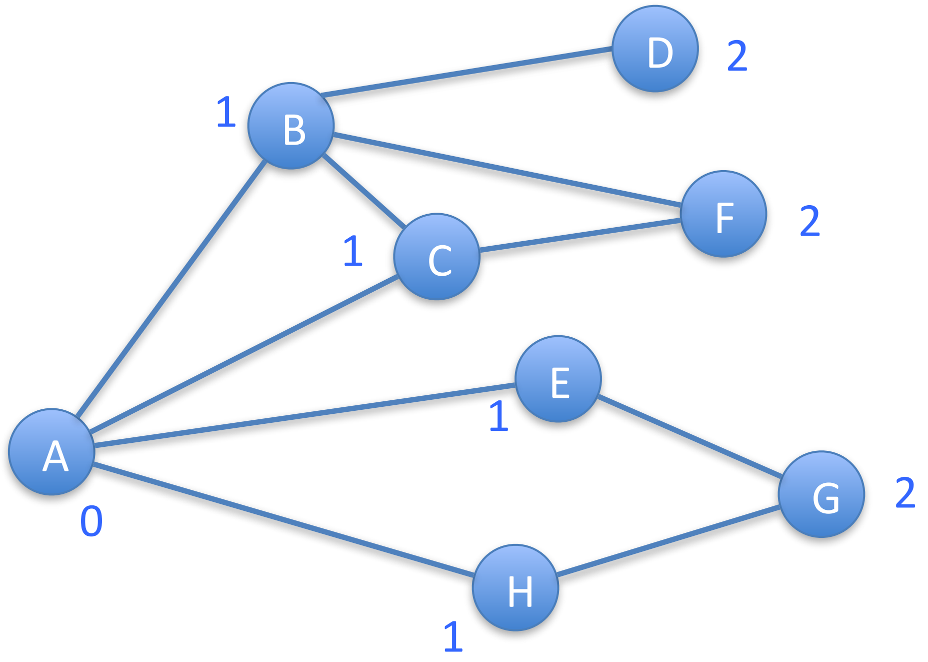


Given a start state and end state, we want to calculate the minimum number of states that must be entered (including the end state but not the start) to get from one to the other.

So, if start == Colorado and end == Montana, the answer is 2 (Colorado->Wyoming->Montana)
We will use a small modification of breadth-first-search to compute these distances.
Can you quickly see the min. for California -> Maine? Might be easier to let bfs do the work …

How we will modify bfs to compute distances from start node



Add a distance property to Node, value None by default

Update node distance when first seen

Mark all nodes 'unseen'
Mark A 'seen', **set A's distance to 0**, and put A on queue Q
Until queue empty do:

- Remove the front node of the queue and call it the current node
- Consider each neighbor of the current node. If its status is 'unseen', mark as 'seen', **set its distance to one more than current node's distance**, and put it on the queue.
- Mark the current node 'processed'

Steps to complete today – we'll tell you **exactly** what to do this time – you just need to follow follow directions carefully

1. Download four files (one is this file) from: http://www.cs.uiowa.edu/~cremer/courses/cs1210/#ds and put them all in the same folder

2. Open ds9x.py in Python, and do

```
>>> g = genStatesGraph()
>>> g
```

You should see something like:

```
[Graph with:
 Nodes: <AK> <AL> <AR> <AZ> <CA> <CO> <CT> <DC> <DE> <FL> <GA> <HI> <IA> <ID> <IL> <IN> <KS> <KY> <LA>
<MA> <MD> <ME> <MI> <MN> <MO> <MS> <MT> <NC> <ND> <NE> <NH> <NJ> <NM> <NV> <NY> <OH> <OK> <OR>
<PA> <RI> <SC> <SD> <TN> <TX> <UT> <VA> <VT> <WA> <WI> <WV> <WY>
 Edges: AL-MS, AL-TN, AL-GA, AL-FL, AR-MO, AR-TN, AR-MS, AR-LA, AR-TX, AR-OK, AZ-CA, AZ-NV, AZ-UT, AZ-CO, AZ-
NM, CA-OR, CA-NV, CO-WY, CO-NE, CO-KS, CO-OK, CO-NM, CO-UT, CT-NY, CT-MA, CT-RI, DC-MD, DC-VA, DE-MD, DE-
PA, DE-NJ, FL-GA, GA-TN, GA-NC, GA-SC, IA-MN, IA-WI, IA-IL, IA-MO, IA-NE, IA-SD, ID-MT, ID-WY, ID-UT, ID-NV, ID-OR,
ID-WA, IL-IN, IL-KY, IL-MO, IL-WI, IN-MI, IN-OH, IN-KY, KS-NE, KS-MO, KS-OK, KY-OH, KY-WV, KY-VA, KY-TN, KY-MO, LA-
TX, LA-MS, MA-RI, MA-NY, MA-NH, MA-VT, MD-VA, MD-WV, MD-PA, ME-NH, MI-WI, MI-OH, MN-WI, MN-SD, MN-ND,
MO-TN, MO-OK, MO-NE, MS-TN, MT-ND, MT-SD, MT-WY, NC-VA, NC-TN, NC-SC, ND-SD, NE-SD, NE-WY, NH-VT, NJ-PA,
NJ-NY, NM-UT, NM-OK, NM-TX, NV-UT, NV-OR, NY-PA, NY-VT, OH-PA, OH-WV, OK-TX, OR-WA, PA-WV, SD-WY, TN-VA,
UT-WY, VA-WV]
```

Look at it: it represents neighbor relationships between states (note: we consider CO, NM, AZ, UT to each be neighbors with the other three despite touching exactly at a corner)

3. Open and edit basicgraph.py

4. Add distance property to Node class in

```
in __init__:
self.distance = None
```

5. Add getDistance, setDistance methods to Node class:

```
def getDistance(self):
    return self.distance

def setDistance(self, value):
    self.distance = value
```

NOTE: make sure to save basicgraph.py before going on.

6. Re-open (or go back to editing) ds9x.py

7. Modify bfs to use distance property:

a) in first loop, add

```
n.setDistance(None)          ← add this line
```

b) just after that loop, before putting startNode on queue, set distance to 0

```
startNode.setDistance(0)          ← add this line
```

c) before putting a newly seen neighbor node on queue, set its distance to one greater than currentNode's distance

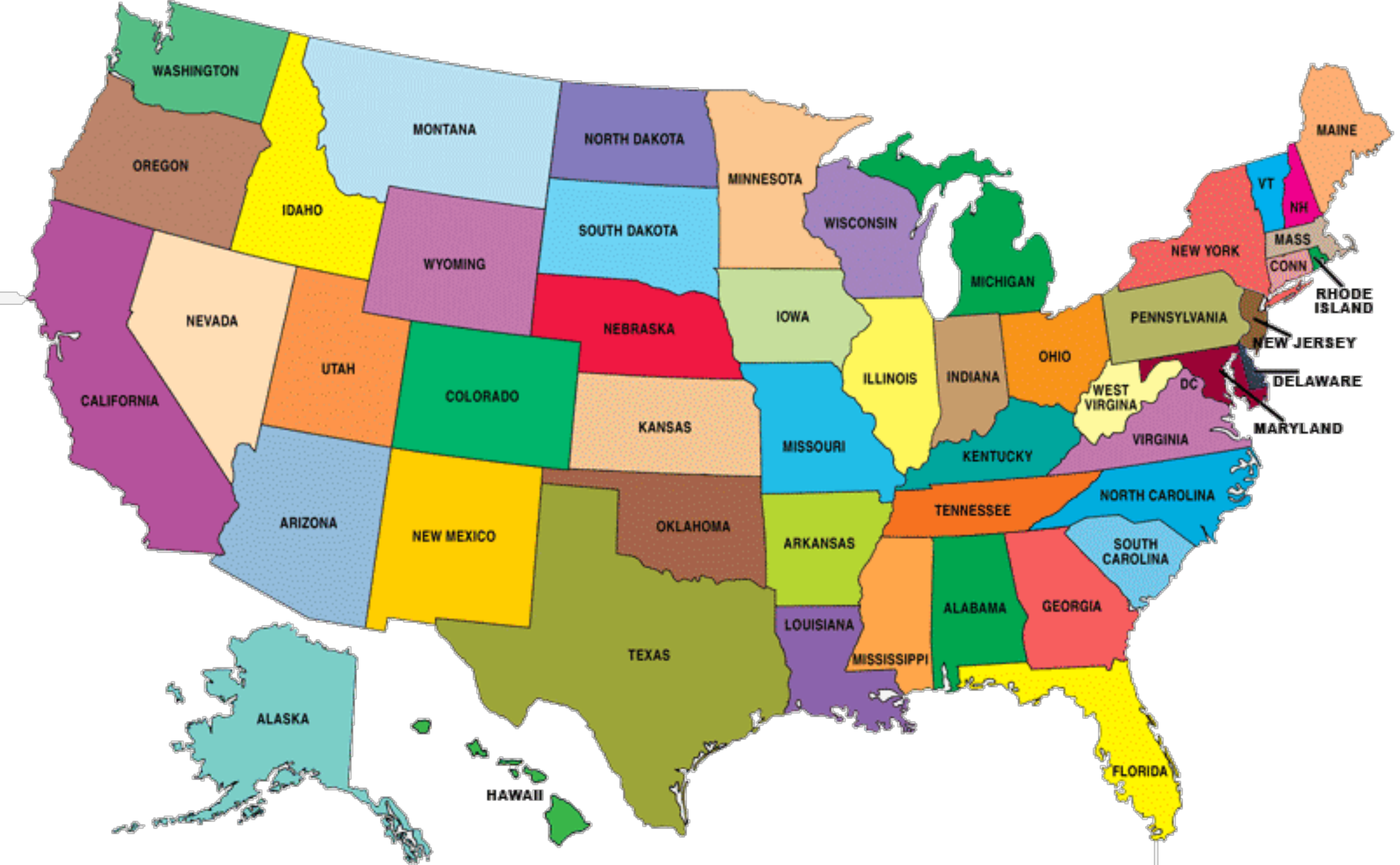
```
node.setDistance(currentNode.getDistance() + 1)     ← add this
queue.enqueue(node)
```

8. Test your work using testDS9x function in ds9x.py. For example,

```
>>> testDS9x('IA', 'TX')
To get from IA to TX you must travel through at least 3 states.
```

Try 'AK' -> 'IA'. How many states? Does that make sense?

Try 'CA' -> 'ME'. How many states? Can you find that route on the original map? Not immediately obvious.



What did testDS9x('CA', 'ME') say? Can you find route with that distance?

It is easy to modify the code further to keep track of "parent" nodes. When a node is first "seen", save as its "parent" the node that discovered/first saw it.
In you have time, you can try to make this modification – you'll need to do essentially the same thing as part of HW8.