Aisheek Ghosh

Troy High School, Class of 2024

Problem

Given a file with the first line as the number of nodes and the other lines as connections between two different nodes,

1. Compute the number of connected components
2. Compute the number of strongly connected components
3. Create histograms of the number of nodes that have a certain degree, whether it be
    a. undirected
    b. directed (in-degrees)
    c. directed (out-degrees)

Terminology

Connected Components - a group of vertices that are connected to each other (think the Internet)

Strongly connected components - a group of vertices that are connected to every other vertex in the group

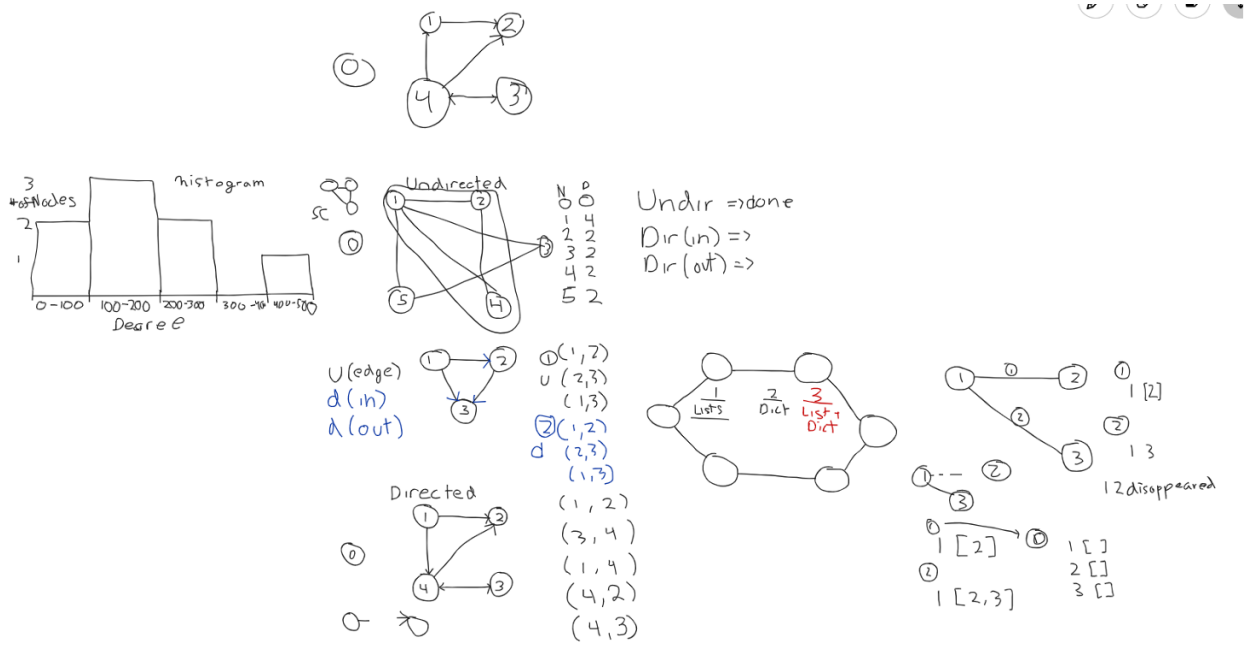
Degrees of nodes - the number of connections with other nodes. It's possible to have a degree of zero

Directed - the edges have a certain order (edges act as arrows)

Undirected - the edges can be in a different order (edges act as lines)

In-degree - has to do with directed graphs, and is where the arrow ends (goes in to)

Out-degree - has to do with directed graphs, and is where the arrow originates (comes out from)



Method

I had a number of different methods I tried to do when trying to solve this problem. First, I tried to take the data from the text file and make it into multiple lists. The problem with that method arose when I tried to sort the list when erasing duplicates. Then, I tried to work with dictionaries instead of lists. This also didn't work, since keys could only have one type of value, and so degrees of above one were impossible. Finally, I tried making the values into lists, and just appending different nodes into the different keys so that I could have one list with all of the connections that one separate node makes.
This method ended up working well, since I was able to keep track of each node's connections with other nodes, but not overwrite any data. All it meant, however, was that the exact connections of the nodes were able to be written, and that the **length** of the list corresponding to the nodes would be the degree.

Algorithm

Number of connected components

Using the dictionary when making the histograms, I made a temporary list with all of the nodes that a certain node is connected to. If there weren't any nodes connected to a node, I put it in a nullList. Then, I made a temporary set, to convert the first temporary list to a set, and updated a final "colony set" with the temporary set. Then, I deleted the node and checked to see if any other nodes had any connections with any members of the "colony set". Every time I found another one, I would update the colony set with the new set and start the iteration over again, until there were no common nodes between the already existing component and the other nodes. Then, I would start again, with the next node that wasn't in a component, and would repeat like this constantly until the entirety of the nodes were in a connected component. The number outputted is a summation of the number of sets created with the nodes and the length of the nullList, which contains the number of degree zero nodes.

Number of strongly connected components

After thinking about how to write this algorithm, I looked online and saw that Python packages, like NetworkX, could do this. But, since I was trying to develop my own algorithm, I was having a hard time organizing my thoughts for this problem. For this problem, if I could talk to you, it would be greatly appreciated.

Histograms

My algorithm took an edge (for the case of the example I will be using *a, b*) and split it into a list (temporary), which was ['a', 'b']. I knew that since this list would only have two values in it, no matter how high the numbers were, there would only ever be two nodes in the temporary list, the first number and the second number. Using this information, I created an edge (b, a), and used it in determining if there were any duplicate edges, by appending (a, b) and (b, a) to a list that I would check for duplicates with. The only problem with this was that I was creating edges that weren't supposed to be there. However, my reasoning behind this was, if a is connected to b, then b is also connected to a. This only worked if the graph was undirected, which would be a problem later down the line. From there, I made different dictionaries for recording the relations between each node (Note: according to this data structure, a was connected to b and b was connected to a). After that, I created a new dictionary with the lengths of the values and the actual keys, as the degrees and a list of nodes, and used it in finding out the undirected degree distribution (histograms below). Then, since I double counted my data, I changed my appending to append only (a, b), which gave me the directed out-degrees. Then, I reasoned that since my undirected degrees were the sum of my directed out degrees and my directed in degrees, I could just 'subtract' my directed out degrees from my undirected degrees, which meant my directed in degree values could be found if I appended (b, a) instead of (a, b) (histograms, again, below).

Tools

Python on Visual Studio Code (Python 2.7.16 64-bit interpreter)

Results

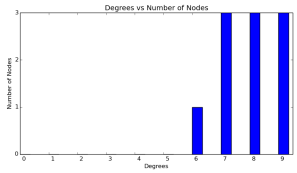
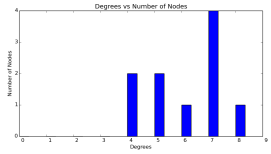
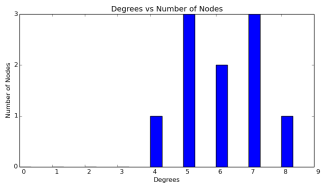
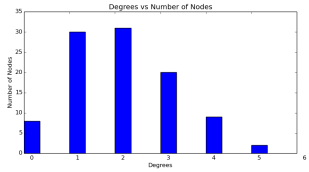
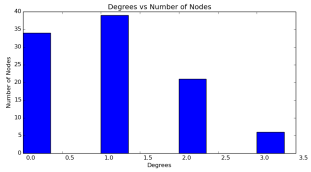
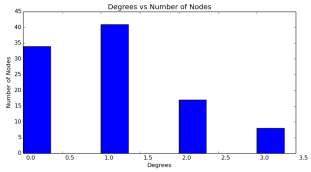
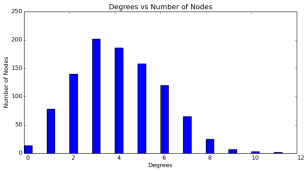
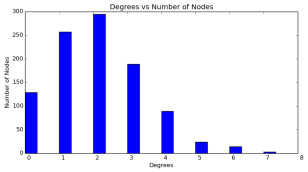
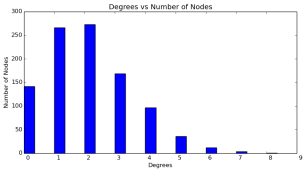
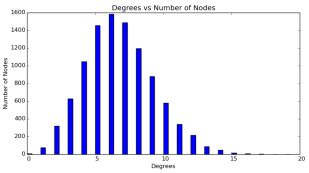
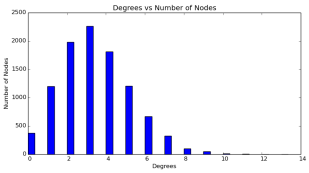
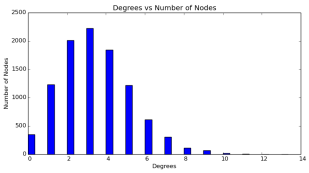
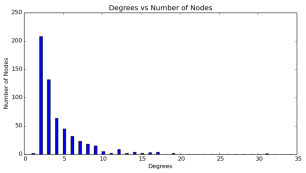
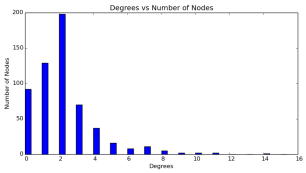
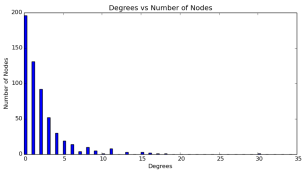
Number of connected components

	# Connected Components	# Connected Components (CORRECTED)
n10	1	1
n100	4	12 = 4 + 8 (degree zero)
n1000	3	17 = 3 + 14 (degree zero)
n10000	2	12 = 2 + 10 (degree zero)
s1	6	6

Number of strongly connected components

	# Strongly Connected Components
n10	See Algorithm
n100	
n1000	
n10000	
s1	

Histograms

	Undirected	Directed In	Directed Out
n10			
n100			
n1000			
n10000			
s1			

References

- Graph Theory 101 https://sitn.hms.harvard.edu/flash/2021/graph-theory-101/
- Graph Theory 101 https://www.lancaster.ac.uk/stor-i-student-sites/katie-howgate/2021/04/27/graph-theory-101/

Appendix

Exercise 1

```
import matplotlib
import numpy as np
import matplotlib.pyplot as plt


numberOfNodes = 0
isLineOne = True
listOfLines = []
frequencyDict = dict()

# ***************
n_nodes = 1
dxDict = {
   1: 3.0/72,
   10: 8.0/72,
   100: 16.0/72,
   1000: 8.0/72,
   10000: 4.0/72

}


f_name = "s{}.txt".format(n_nodes)
graphName = "graphs/s{}_histogram_directed.png".format(n_nodes)
# ****************

f = open(f_name, 'r')
for line in f:
   if isLineOne:
       numberOfNodes = int(line.strip())
       for i in range(numberOfNodes):
           frequencyDict[i] = []
```

```
    else:
        listTemp = line.strip().split(' ')
        v1 = listTemp[0]
        v2 = listTemp[1]
        listOfLines.append(v1 + '-' + v2) # toggle for undirected or
directed in
        listOfLines.append(v2 + '-' + v1) # toggle for undirected or
directed out
    isLineOne = False
f.close()
listOfLines = sorted(listOfLines)
for i in range(len(listOfLines)-1, 0, -1):
    if listOfLines[i] == listOfLines[i-1]:
        listOfLines.pop(i-1)

for item in listOfLines:
    temp = item.split('-')
    key = int(temp[0])
    val = int(temp[1])
    frequencyDict[key].append(val)



degreeToNodes = dict()
for i in range(numberOfNodes):
    degreeToNodes[i] = []

for (k,v) in frequencyDict.items():
    keyNew = len(v)
    valNew = k
    degreeToNodes[keyNew].append(valNew)


actualDegreeToNodes = dict()
for (a, b) in degreeToNodes.items():
```

```
    actualDegreeToNodes[a] = len(b)


degreeNumber = list(actualDegreeToNodes.keys())
numberOfNodesWithDegree = list(actualDegreeToNodes.values())
fig, ax = plt.subplots(figsize = (10, 5))

nullList = []
for i in range(numberOfNodes):
    if len(frequencyDict[i]) == 0:
        nullList.append(i)
        frequencyDict.pop(i)

frequencyDictInitial = frequencyDict

regionsList = []
while len(frequencyDict) > 0:

    k,v = list(frequencyDict.items())[0]
    tempList = [k] + v
    tempSet = set(tempList)

    colonySet = set()
    colonySet.update(tempList)
    frequencyDict.pop(k)

    for i in range(numberOfNodes):
        for k,v in frequencyDict.items():
            tempList = [k] + v
            tempSet = set(tempList)
            z = colonySet.intersection(tempSet)

            if len(z) > 0:
                colonySet.update(tempSet)
                frequencyDict.pop(k)
```

```
    if len(colonySet) > 0:
        regionsList.append(colonySet)


print(len(regionsList) + len(nullList))
```

Exercise 3

```
import matplotlib
import numpy as np
import matplotlib.pyplot as plt


numberOfNodes = 0
isLineOne = True
listOfLines = []
frequencyDict = dict()

# ***************
n_nodes = 10000
dxDict = {
   1: 3.0/72,
   10: 8.0/72,
   100: 16.0/72,
   1000: 8.0/72,
   10000: 4.0/72

}


f_name = "n{}.txt".format(n_nodes)
graphName = "graphs/n{}_histogram_directed_out.png".format(n_nodes)
# ****************

f = open(f_name, 'r')
for line in f:
   if isLineOne:
       numberOfNodes = int(line.strip())
       for i in range(numberOfNodes):
           frequencyDict[i] = []
   else:
       listTemp = line.strip().split(' ')
```

```
        v1 = listTemp[0]
        v2 = listTemp[1]
        listOfLines.append(v1 + '-' + v2) # toggle for undirected or
directed in
        listOfLines.append(v2 + '-' + v1) # toggle for undirected or
directed out
    isLineOne = False
listOfLines = sorted(listOfLines)
for i in range(len(listOfLines)-1, 0, -1):
    if listOfLines[i] == listOfLines[i-1]:
        listOfLines.pop(i-1)
for item in listOfLines:
    temp = item.split('-')
    key = int(temp[0])
    val = int(temp[1])
    frequencyDict[key].append(val)

degreeToNodes = dict()
for i in range(numberOfNodes):
    degreeToNodes[i] = []

print(frequencyDict)
for (k,v) in frequencyDict.items():
    print(k,len(v))
    keyNew = len(v)
    valNew = k
    degreeToNodes[keyNew].append(valNew)

print(degreeToNodes)
for (a, b) in degreeToNodes.items():
    print(a, len(b))

actualDegreeToNodes = dict()
for (a, b) in degreeToNodes.items():
    actualDegreeToNodes[a] = len(b)
```

```
print(actualDegreeToNodes)

degreeNumber = list(actualDegreeToNodes.keys())
numberOfNodesWithDegree = list(actualDegreeToNodes.values())
fig, ax = plt.subplots(figsize = (10, 5))

plt.bar(degreeNumber, numberOfNodesWithDegree, color ='green', width =
0.35)
plt.xlabel('Degrees')
plt.ylabel('Number of Nodes')
plt.title('Degrees vs Number of Nodes')
plt.bar(degreeNumber, numberOfNodesWithDegree, width = 0.35, label='Degree
Numbers')

# plt.xticks(np.arange(min(degreeNumber), max(degreeNumber)+1, 1))
# Uncomment above and below for n_nodes = 10
# plt.yticks(np.arange(min(numberOfNodesWithDegree),
max(numberOfNodesWithDegree)+1, 1))

dx = dxDict[n_nodes]; dy = 0/72.
offset = matplotlib.transforms.ScaledTranslation(dx, dy,
fig.dpi_scale_trans)



for label in ax.xaxis.get_majorticklabels():
   label.set_transform(label.get_transform() + offset)
plt.savefig(graphName)


f.close()

print(sum(actualDegreeToNodes.values()))
```