CS 5854: Networks, Crowds, and Markets
Homework 4

Instructor: Rafael Pass TAs: Cody Freitag, Drishti Wali

Assigned: November 26, 2019 Due: December 12, 2019, 11:59 pm

Late Deadline: At most one slip day can be used!
Submissions cannot be accepted after December 13, 11:59 pm

Kushal Singh

Collaborators: I worked with Chris Shei and Jackson (Scooter) Blume.
Outside resources: I used the following outside resources.
Late days: I have used 1 many late days on this assignment.

Part 1: Voting

1. In the American presidential elections, while the popular vote is used up to the state level, the electoral college decides the winner at the national level. Assuming there are only two candidates, is this system strategy-proof? Does it elect a Condorcet winner? Justify your answers. (Note: You can assume for simplicity that each state gets a single "vote" in a national election, and the state then runs an election by popular vote with however many people are in that state to determine which vote to cast at the national level.)

   Solution:

   1. If there are only two candidates, then the system is indeed strategy-proof. This is because when voters vote for their preferred candidate, it maximizes their utility. Consequently, if voters do not vote for their preferred candidate, then they minimize their utility. Therefore, there is no scenario where voting for a less preferred candidate will yield higher utility for an individual. We can corroborate this observation via a proof by cases:

      a) If your preferred candidate is going to win, then your vote does not matter, since voting for or against will not make a difference.

      b) If your preferred candidate is going to lose, regardless of your vote, then changing your vote does not matter.

      c) If your vote is the deciding vote, then you will want to vote for your preferred candidate.

      If there are more than two candidates, however, this system is no longer strategy-proof. This is because if your preferred candidate is all but guaranteed to lose, and your swing vote could help your second preferred candidate to win, then you can vote strategically to maximize your utility.

      This system is not guaranteed to elect the Condorcet winner. Suppose we have three states: s_1, s_2, s_3, where s_1 and s_2 have 100 people apiece and they all vote for candidate c_1. Suppose also that s_3 has 1000 people and they all vote for candidate c_2.

      c_1: 2 state votes and 200 actual votes
      c_2: 1 state vote and 1000 actual votes.

      In this case, c_1 beats c_2 with more state votes, but fewer actual votes. Clearly, the winner is not the Condorcet winner.

      □

2. Suppose we want to run a popular vote election between two candidates A and B. There are 1,000,000 eligible voters and suppose 52% of them prefer A to B. Instead of running a full election where every person casts a vote, we poll m randomly selected people (with replacement for simplicity) and ask them to report their preferred candidate. The result of the poll is the majority of the responses received.

   (a) Compute a value of m so that the result of the poll is incorrect with probability at most 1%? (Use the Chernoff bound in the book, show your work.)

   (b) Let n be the number of people in the population, ϵ be defined such that $(1/2 + \epsilon) \cdot n$ prefer A to B, and let δ be the desired accuracy (so the probability the result is incorrect is at most δ). Write m as a function of n, ϵ, and δ.

   If the number of people in the population increased by a factor of 10, how would that affect m? If ϵ decrease by a factor of 2, how would that affect m? If we want to increase our confidence by a factor of 10, how would that change m? If $\epsilon = 1/n$ (so 1 person would be the deciding vote), what would this imply about m given your bound from above?

   (c) In practice, what might be wrong with the above assumptions (i.e. why might we not we use polls to run our elections)?

Solution:

(a) From Theorem 16.2 in the book, we know that the Chernoff bound looks like the following:

> **Theorem 16.2.** *Let $W \in \{0,1\}$, and let $X_1, \ldots, X_n \in \{0,1\}$ be independent random variables such that $\Pr[X_i = W] \geq 1/2 + \epsilon$. Then:*
>
> $$\Pr[Majority(x_1, \ldots, x_n) = W] \geq 1 - 2e^{-2\epsilon^2 n}$$

$Pr[X_i = A] \geq 0.5 + \epsilon = 0.52 \implies \epsilon = 0.02$
Furthermore, since $Pr[incorrect] \leq 1\%$...

$$Pr[Majority(x1, .., xn) = A] \geq 1 - 2e^{-2\epsilon^2 n} = 99\%;$$
$$1 - 2e^{-2*0.02^2*n} = 99\%;$$
$$n \approx 6622$$

(b) From Theorem 16.2, we have:

$$Pr[Majority(x1, .., xn) = A] \geq 1 - 2e^{-2\epsilon^2 n} = 99\%$$

As $P(incorrect) \leq \delta$, we have:

$$1 - 2e^{-2*\epsilon^2*m} = 1 - \delta$$
$$2e^{-2*\epsilon^2*m} = \delta$$
$$e^{-2*\epsilon^2*m} = \frac{\delta}{2}$$
$$-2\epsilon^2 m = \ln(\frac{\delta}{2})$$

$$m = -\frac{\ln(\frac{\delta}{2})}{2\epsilon^2}$$

From this, we can see that varying n will not have an effect on m. If we were to decrease δ by a factor of 2, then m would increase by a factor of 4. If we were to increase the confidence by a factor of 10, then m increases by $\frac{ln(10)}{2\epsilon^2}$.

(c) In practice, we might not want to use polls to run our elections because people may be wary of divulging their true opinions if they do not know or trust where their data is going, in contrast to real elections. Furthermore, making the poll truly random is fairly hard to accomplish.

□

Part 2: Stable Matchings

3. For the following setting, find **(a)** the male-optimal and **(b)** the female-optimal stable matching. For each part, simulate the Gale-Shapley algorithm (i.e. in words, clearly indicate what happens at each step) to show that it arrives at the matching you find.

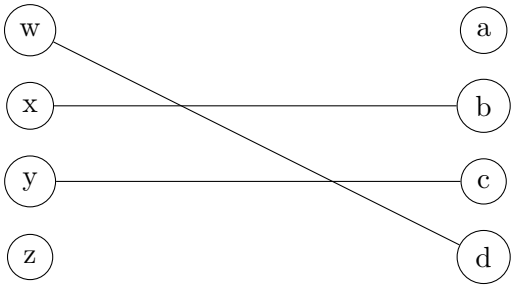
Females	Preferences	Males	Preferences
A	$X > W > Y > Z$	W	$D > B > C > A$
B	$X > W > Y > Z$	X	$D > B > A > C$
C	$X > W > Z > Y$	Y	$C > B > D > A$
D	$Y > W > Z > X$	Z	$D > B > C > A$

4. Prove that there exists a non-bipartite matching setting (where every individual has preferences over all other individuals) for which no stable matching exists.

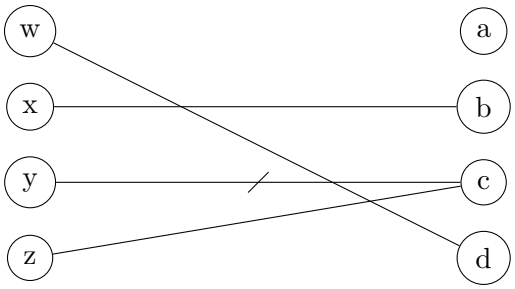
Solution:

3. a) **Male optimal case**
   Iter 0

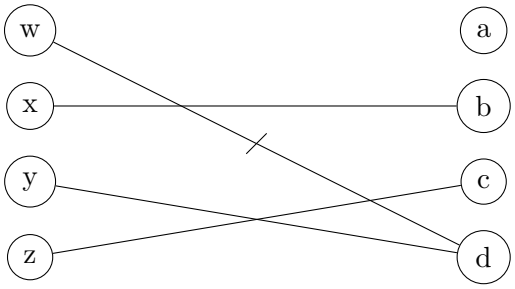
   Iter 1: w chooses top choice; y chooses top choice; x asks top choice (d) and gets **rejected**; x asks second best choice (b) and gets **accepted**.



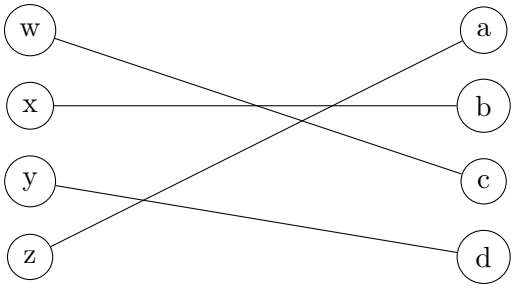
   Iter 2: z asks ideal (d) and gets **rejected**; z asks second best choice (b) and gets **rejected**; z asks third best choice (c) and gets **accepted**.



Iter 3: y is now unengaged from c; y cannot pair with c or b, so it pairs with d; this leaves w unengaged.

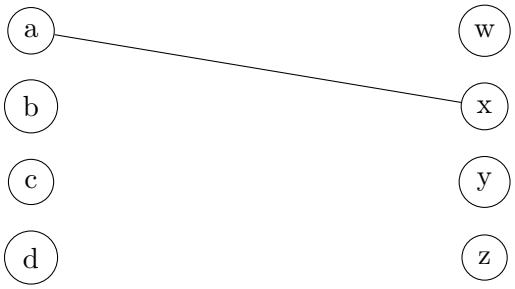


Iter 4: w connects with best available option (c); z pairs with best available option (a).

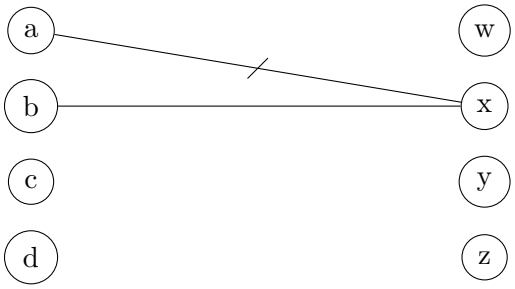


b) **Female optimal case**
Iter 0

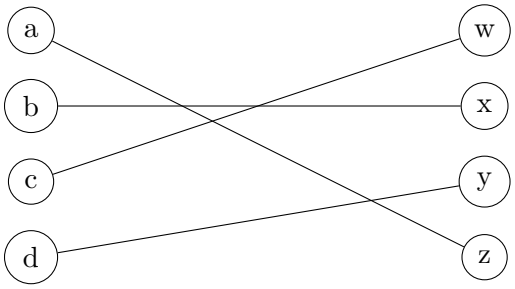
Iter 1: a chooses top choice (x).



Iter 2: b chooses top choice (x); a is unmatched as x leaves a for b.



Iter 3: c chooses best option x and gets **rejected**; c proposes to w and gets **accepted**; d chooses best option (y); a chooses best option (z).



4. Consider the following scenario.

Players	Preferences
A	$B > C$
B	$C > A$
C	$A > B$

If we have this table of players and preferences, then:

1. A connects with its top preference, which is B.
2. B prefers C, so B connects with C, leaving A disconnected.
3. C prefers A, so C connects with A, leaving B disconnected.
4. A prefers B, so A connects with B, leaving C disconnected.

From this circular scenario of preferences, we can see that running this algorithm will result in an infinite loop and, thus, no stable matching.

□

Part 3: Beliefs

5. There is a test for a certain disease that has a 15% false positive rate and a 25% false negative rate. (So, if someone has the disease, there is a 75% chance the test will return positive; if they do not, there is an 85% chance it will return negative.) If 1% of the population has this disease, what is the probability that someone who tests positive for the disease actually has it? (Use Bayes' Rule; show your work.)

6. In the "foolishness of crowds" example from section 16.2 of the notes, let's assume that people decide that their own evidence should instead be weighed c times as heavily as others' prior guesses, where c is an integer greater than 1. Now what is the probability that a cascade occurs and *everyone* is incorrect (in terms of ϵ, where the probability that a player receives correct evidence is $1/2 + \epsilon$)?

Solution:

5. Let's suppose that: D = Disease, $\neg D$ = No Disease, T = Test Positive, $\neg T$ = Test Negative. Then, $Pr[D] = \frac{1}{100}$, $Pr[T|\neg D] = 0.15$, $Pr[\neg T|D] = 0.25$.

   Using Bayes' Rule, we have:

$$Pr[D|T] = \frac{Pr[T|D] * Pr[D]}{(Pr[T|D] * Pr[D]) + (Pr[T|\neg D] * Pr[\neg D])};$$
$$Pr[D|T] = \frac{(1 - 0.25) * 0.01}{[(1 - 0.25) * 0.01] * [0.15 * (1 - 0.01)]};$$
$$Pr[D|T] = \frac{0.0075}{0.0075 + 0.1485}$$
$$Pr[D|T] \approx 0.048$$

6. A cascade occurs when the guesses from the crowd are more influential than the person's own evidence. This happens when there are $\geq c + 1$ people making the incorrect guess before the next person's guess. From there, everyone else ($c + 2$ and onward) guess with the crowd. Therefore, the probability that a cascade occurs and *everyone* is incorrect is equivalent to the probability of the first $c + 1$ people guessing wrong $= (\frac{1}{2} - \epsilon)^{c+1}$.

□

7. Recall the "muddy children" example covered in class and in the notes. Section 17.2 of the notes gives a formal argument that Claim 17.1 (that, if there are m muddy children, they will answer "yes" on and not before round m) holds for the case where there are two total children.

   (a) Now draw the knowledge network for the case where there are three total children.

   (b) Using a similar argument to the case of two children, use your network and the formal "possible worlds" model of knowledge to formally show that Claim 17.1 holds for the case when there are three total children (and any non-zero number of muddy children).

Solution:

7. (a) *Note: you may include a handwritten drawing in the aux files for this homework.*
   (b)

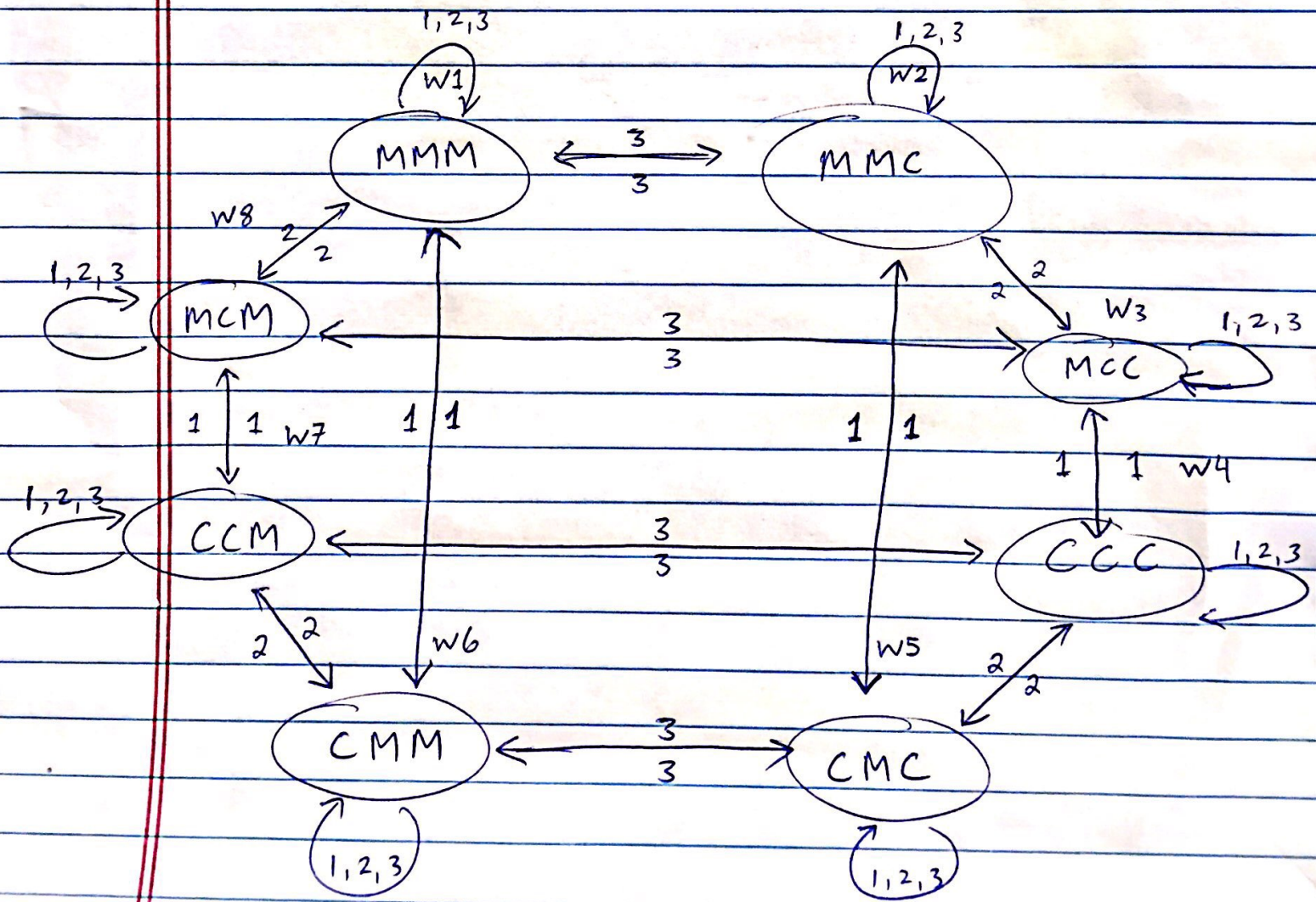
□

a) The set of all possible worlds is

$\Omega = (w_1 \ldots w_8)$

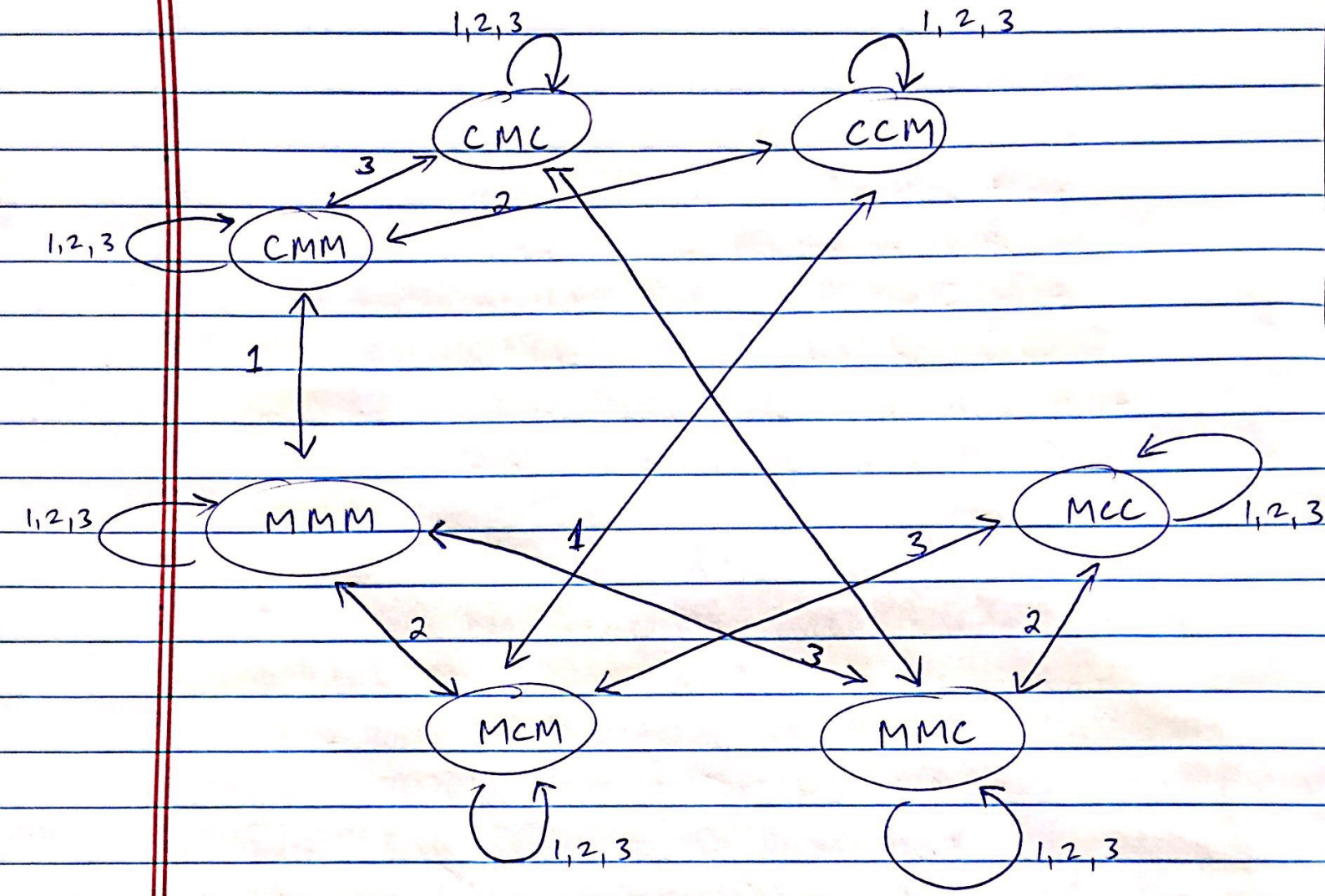
The outcome space is

$\{M, C\}^3$

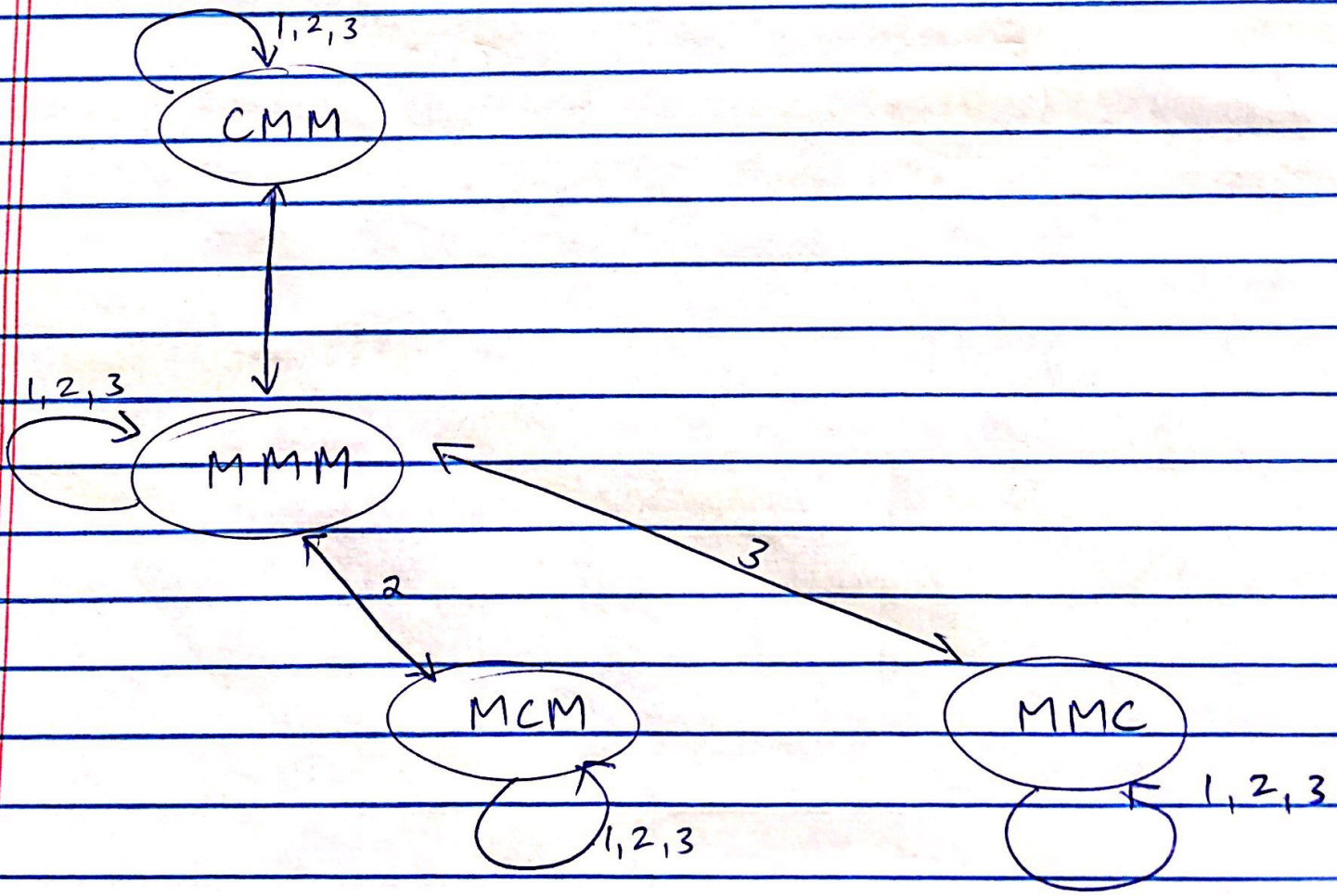


b) First, we show P(1). Since the father mentions that there are some muddy children, if there is only one muddy child, then they will see nobody else in the room with mud and know in the first round itself that they are muddy. If there are two or more muddy children, then they only know that some children (possibly including them) are muddy.

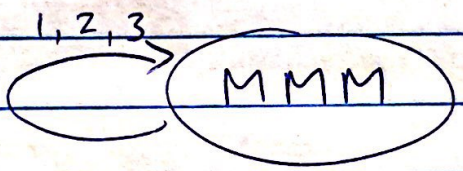
As soon as the father announces "there are muddy children", this destroys CCC, since someone must have mud on their face. For 1 muddy child, they would observe that all other kids have clear faces, which means that they would know they have muddy faces and would want to clean up.



For P(2), since there are ~~more~~ is than 1 muddy child, noone will say "yes" before the second round. In the second round, each muddy child sees 1 other muddy child, and therefore knows that there are either 1 or 2 muddy children total. They do know that if there were only 1 muddy child, someone would have said "yes" in the previous round. Since no one has spoken yet, each muddy child is able to say that there are 2 muddy children, including themselves. Therefore, we have the only world with at least 2 muddy children left:



For P(3), we have: Since there are > 2 muddy children, no one will say "yes" before round 3. In the third round, each muddy child sees 2 other muddy children, and knows that there are either 2 or 3 muddy children in total. However, they know that if there truly were only 2 muddy children, someone would have said "yes" in the second round. Since no one has spoken yet, each muddy child knows that there are in fact 3 muddy children, including themselves. So, the only world with 3 muddy children left is:



Let's assume that P(k) is true for $0 \leq k \leq n$. We will show the case for P(k+1). Suppose there are exactly k+1 muddy children. Since there are more than k muddy children, no one will say "yes" before round k+1 by the induction hypothesis. In that round, each muddy child sees k other muddy children, and knows that there are either k or k+1 muddy children total. By the induction hypothesis, they are able to infer that if there were only n muddy children, someone would have said "yes" in the previous round. Since no one has spoken yet, each muddy child is able to infer that there are indeed k+1 muddy children,

including themselves. If there are strictly more than k+1 children, however, then all children can tell that there at least k+1 muddy children Simply by looking at the others. Therefore, by the induction hypothesis, they can infer from the start that nobody will say "yes" in round k. So, they will have no more information than they did initially in round k+1, and will be unable to tell whether they are muddy as a result.

Part 4: PageRank and Social Networks

The objective of this question is to use the PageRank algorithm as a way to determine how "influential" a node is in a social network based on its in-links from influential nodes. For this question, we provided a template in python called 'hw4.py'. You must use the template to submit your code, as we will grade your code in a (partially) automated way. *You should submit the hw4.py file, not a .pynb or other python file.*

8. Design an algorithm that runs the iterative ϵ-scaled PageRank algorithm for a specified number n of rounds on a given directed graph, with $\epsilon = 1/7$. Run it (with $n = 10$) on the examples in figures 15.1 (both left and right) and 15.2 (the two disjoint triangle graph), as well as at least two other simple test cases with at least 10 nodes.

9. Now we'll run PageRank on the Facebook data.

   [http://snap.stanford.edu/data/egonets-Facebook.html].

   The file is called "facebook_combined.txt.gz"; remember that it has 4,039 nodes.

   (a) Once again, remember that this is an undirected graph! Before running your algorithms from the previous problem, implement a transformation into a directed graph, i.e. each undirected edge corresponds to two different directed edges.

   (b) Now, run the PageRank algorithms from the last problem on this new graph. You shouldn't need n to be much higher than 10-20 for the algorithm to converge to a fixed point.

   (c) Where did most of the score tend to end up in your experiments? Look at the nodes that have the highest or lowest scores; is there a consistent pattern among your trials?

   (d) Intuitively explain your results in terms of a measure of influence in a social network. Do you think that this is an accurate measurement? How could we try to improve it (for instance, by incorporating link strengths or other measures of popularity)?

Solution:

8. In the graph images below that demonstrate the iterative ϵ-scaled PageRank algorithm, the blue nodes have the lowest page rank, while the red nodes have the highest page rank.

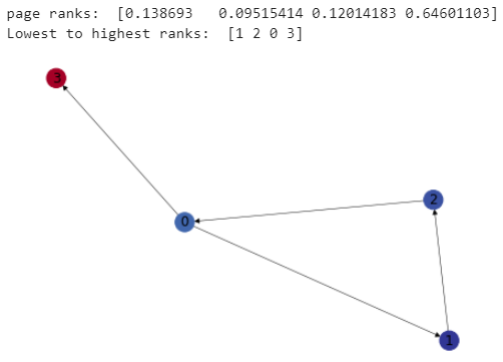


Figure 15.1 left

```
page ranks:  [0.0947099  0.0556314  0.07716155 0.38624857 0.38624857]
Lowest to highest ranks:  [1 2 0 3 4]
```

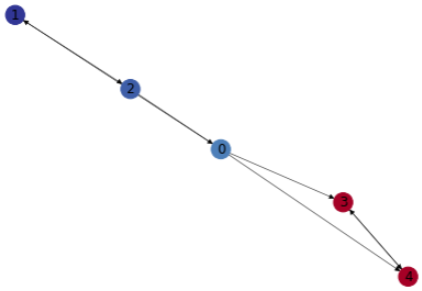


Figure 15.1 right

```
page ranks:  [0.16666667 0.16666667 0.16666667 0.16666667 0.16666667 0.16666667]
Lowest to highest ranks:  [0 1 2 3 4 5]
```

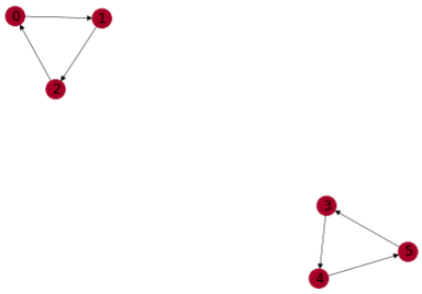
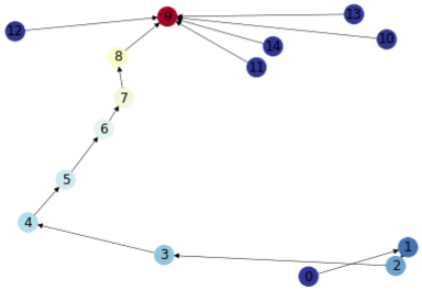


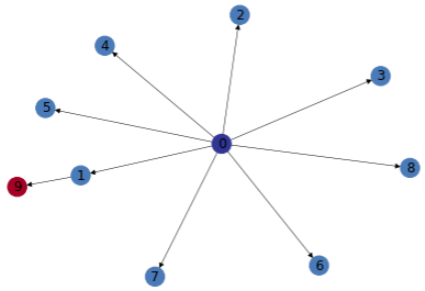
Figure 15.2

```
page ranks:  [0.00952381 0.01768707 0.02468416 0.03068166 0.03582238 0.0402287
 0.04400555 0.04724286 0.05001769 0.09321244 0.00952381 0.00952381
 0.00952381 0.00952381 0.00952381]
Lowest to highest ranks:  [ 0 10 11 12 13 14  1  2  3  4  5  6  7  8  9]
```



Example 1

```
page ranks:  [0.01428571 0.01581633 0.01581633 0.01581633 0.01581633 0.01581633
 0.01581633 0.01581633 0.01581633 0.02784257]
Lowest to highest ranks:  [0 1 2 3 4 5 6 7 8 9]
```

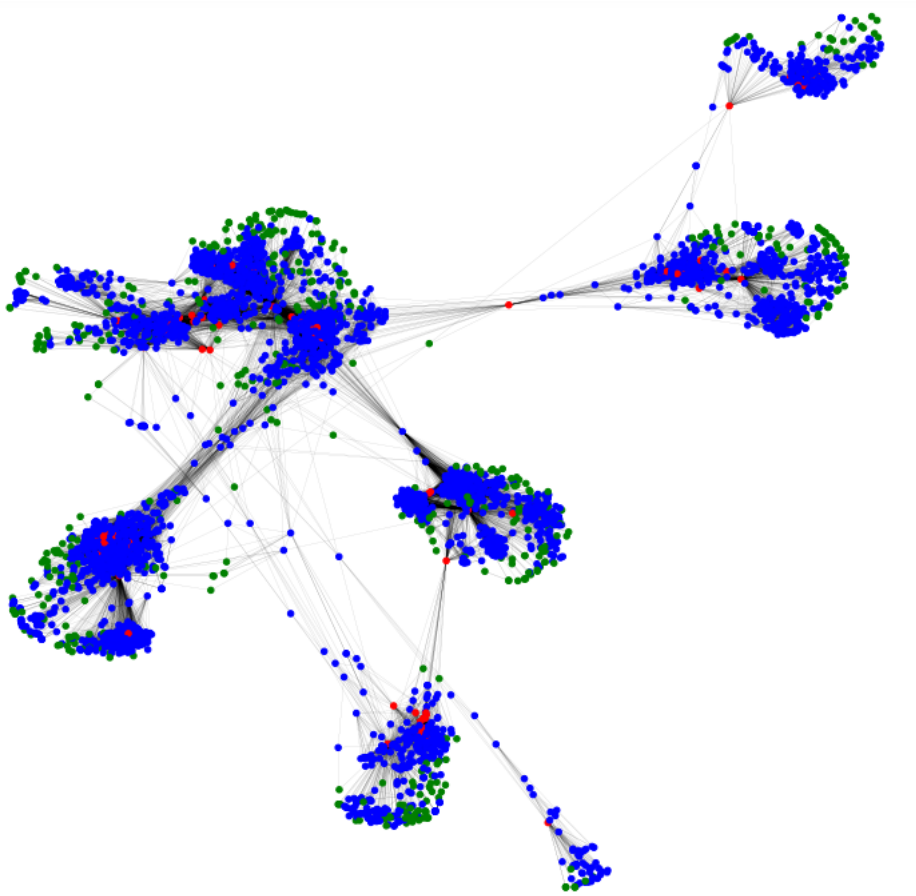


Example 2

From the graphs above, we can see that the Pagerank algorithm does a fairly decent job in identifying nodes with low and high page ranks. In Example 1, node 9 has several incoming links and, thus, the highest rank. In Example 2, we can see that node 0 has several outgoing links, but zero incoming links and, thus, a low rank. Node 9, which does have an incoming link, is ranked highest, since it's parent also has an incoming link.

9. (a) *****SEE ATTACHED CODE FILES*****

(b) The graphic below shows the Pagerank algorithm implemented on the Facebook network. The red nodes represent the top 100 Page-ranked nodes, while the green nodes represent the bottom 1000 Page-ranked nodes.



Example 2

(c) Most of the score tended to end up at nodes that are in the center of clumps. The nodes with highest scores tend to have the most incoming and outgoing links. The nodes with low scores tend to lie on the edge of the graph and are minimally connected. An interesting observation is that a portion of the score went toward nodes that bridged gaps between these large clumps.

(d) Intuitively, this score distribution makes sense. The nodes that have the most connections tend to have the highest score, and the ones with few connections tend to have a smaller score. Each clump has at least one high scoring node and most of the connections between clumps have a node with a high score.

□

Part 5: Essay Question

(This problem should be completed individually and not in a group. However, it will be graded based on completion.)

Write 1/2 to 1 page discussing or analyzing one of the following prompts using any of the concepts taught in this class:

- Read the following New York Times article adapting a recent work from Nobel Prize winners Esther Duflo and Abhijit Banerjee:
  https://www.nytimes.com/2019/10/26/opinion/sunday/duflo-banerjee-economic-incentives.html
  The article makes the claim that people aren't driven by financial incentives as much as you would assume. In particular, they cite a study that claims that people believe that "Everyone else responds to incentives, but I don't." Why might this undermine certain assumptions made in this class? How can we model this situation using ideas from this class?

- Watch the following speech from Sacha Baron Cohen:
  https://www.youtube.com/watch?v=ymaWq5yZIYM
  The speech discusses the relevance of the spread of (mis)information in social network. For example, he makes the claim that "fake news outperforms real news because lies spread faster than truth." How can use tools from this class to explain this phenomena? How could we use tools from this class to identify the spread of misinformation?

- Pick one or more recent news article (within the last few months) related to networks, markets, or beliefs to analyze and discuss. (In particular, you may look at one of the above examples and discuss a different aspect if you wish.)

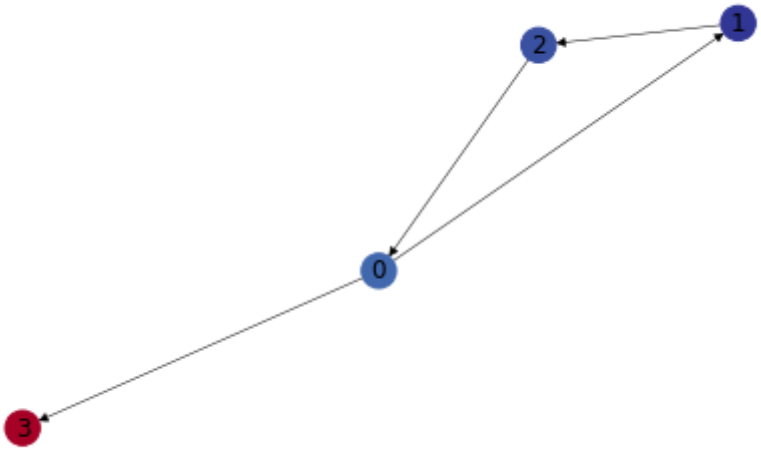
Sacha Baron Cohen states that fake news outperforms real news because lies spread faster than truth. This is possible because of the sheer strength of social networks such as Facebook and Twitter, where graphs are very strongly connected. In fact, there is a very prevalent known fact that, on average, any individual in the entire world is connected to some random stranger through six mutual connections (aka six degrees of freedom). Consequently, if individuals peruse and divulge news articles without checking to see how truthful and accurate they are, then this can lead to the "Foolishness of Crowds" and, more specifically, the "herding" behavior as a wrong previous answer can adversely influence an individual's current answer. The herding model can help understand the spread of misinformation, as individuals are more likely to believe a news source from within a familiar close network (e.g. friends and family), over a stranger or unfamiliar source.

```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from matplotlib import cm


class DirectedGraph:

    def __init__(self, number_of_nodes):
        self.__number_of_nodes = number_of_nodes
        #self.number_of_nodes = number_of_nodes
        self.adjacency_matrix = np.zeros((number_of_nodes, number_of_nodes))

    def add_edge(self, origin_node, destination_node):
        self.adjacency_matrix[origin_node, destination_node] = 1

    def edges_from(self, origin_node):
        ''' This method should return a list of all the nodes u such that the edge (or
        part of the graph.'''
        get_row = self.adjacency_matrix[origin_node]
        get_edge_nodes = np.where(get_row == 1)[0]
        return list(get_edge_nodes)

    def check_edge(self, origin_node, destination_node):
        ''' This method should return true is there is an edge between origin_node and
        and destination_node, and false otherwise'''
        return(self.adjacency_matrix[origin_node, destination_node] == 1)
        #if (self.adjacency_matrix[origin_node, destination_node] == 1):
          #return True
        #return False

    def number_of_nodes(self):
        ''' This method should return the number of nodes in the graph'''
        return(self.__number_of_nodes)
        #return(self.number_of_nodes)

    # Additional Methods
    def out_degree(self, origin_node):
        return(len(self.edges_from(origin_node = origin_node)))

    def get_nodes(self):
        return(list(range(self.number_of_nodes())))

    def edges_to(self, destination_node):
        return (list(np.where(self.adjacency_matrix[:, destination_node] == 1)[0]))


testGraph = DirectedGraph(5)
assert testGraph.number_of_nodes() == 5
assert testGraph.check_edge(0, 1) == False
testGraph.add_edge(0, 1)
```

```
    testGraph.add_edge(0, 1)
    assert testGraph.check_edge(0, 1) == True
    assert testGraph.edges_from(0) == [1]


    def scaled_page_rank(graph, num_iter, eps = 1/7.0):
        ''' This method, given a directed graph, should run the epsilon-scaled page-rank
        algorithm for num-iter iterations and return a mapping (dictionary) between a node
        In the case of 0 iterations, all nodes should have weight 1/number_of_nodes'''
        previous = np.full(graph.number_of_nodes(),(1/graph.number_of_nodes()))
        ranks = np.zeros(graph.number_of_nodes())

        for i in range(num_iter):
          for j in range(graph.number_of_nodes()):
            rank = 0
            for node in graph.edges_to(j):

              rank += previous[node] / graph.out_degree(node)
            ranks[j] = (eps/graph.number_of_nodes()) + (1-eps) * rank

          previous = np.copy(ranks)

        return previous


    def show_graph(graph, show):
      viridis = cm.get_cmap('RdYlBu')
      ranks = scaled_page_rank(graph,10)
      sorted_rank = np.argsort(ranks)
      network = nx.DiGraph()
      network.add_nodes_from(np.arange(graph.number_of_nodes()))

      for node in range(graph.number_of_nodes()):
        for to in graph.edges_from(node):
          network.add_edge(node,to)

      colors = np.full((graph.number_of_nodes()),(.5))
      for rank in sorted_rank:
        colors[rank] = 1-ranks[rank]

      print('page ranks: ',ranks)
      print("Lowest to highest ranks: ",sorted_rank)
      nx.draw(network, node_size=300, width=.5,with_labels=True, node_color=colors, cmap=v
      plt.show()


    def graph_15_1_left():
        ''' This method, should construct and return a DirectedGraph encoding the left exa
        Use the following indexes: A:0, B:1, C:2, Z:3 '''
        g = DirectedGraph(4)
        g.add_edge(0,1)
        g.add_edge(1,2)
        g.add_edge(2,0)
```

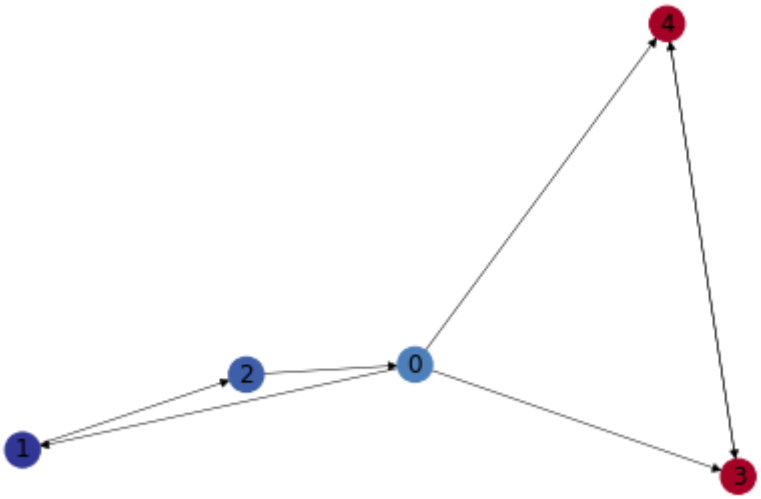
```
    g.add_edge(0,3)
    g.add_edge(3,3)
    return g

show_graph(graph_15_1_left(),1)
```

```
page ranks:  [0.138693   0.09515414 0.12014183 0.64601103]
Lowest to highest ranks:  [1 2 0 3]
```



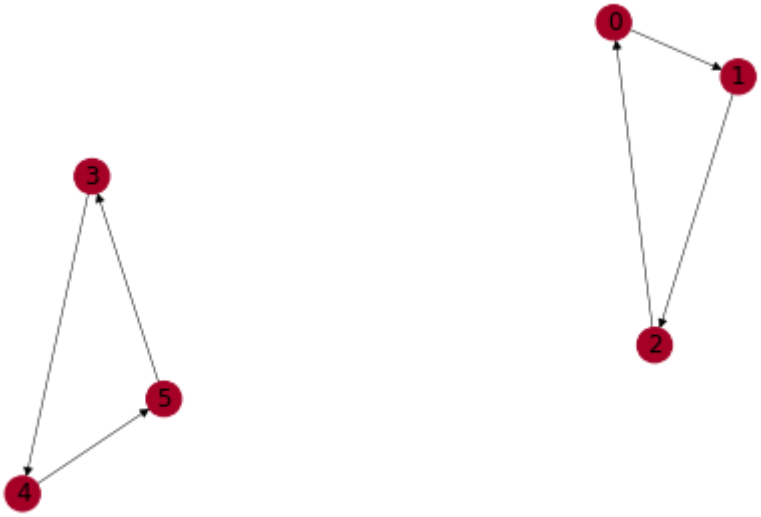

```
def graph_15_1_right():
    ''' This method, should construct and return a DirectedGraph encoding the right ex
    Use the following indexes: A:0, B:1, C:2, Z1:3, Z2:4'''
    g = DirectedGraph(5)
    g.add_edge(0, 1)
    g.add_edge(1, 2)
    g.add_edge(2, 0)
    g.add_edge(0, 3)
    g.add_edge(0, 4)
    g.add_edge(3, 4)
    g.add_edge(4, 3)
    return g

show_graph(graph_15_1_right(),2)
```

```
page ranks:  [0.0947099  0.0556314  0.07716155 0.38624857 0.38624857]
Lowest to highest ranks:  [1 2 0 3 4]
```



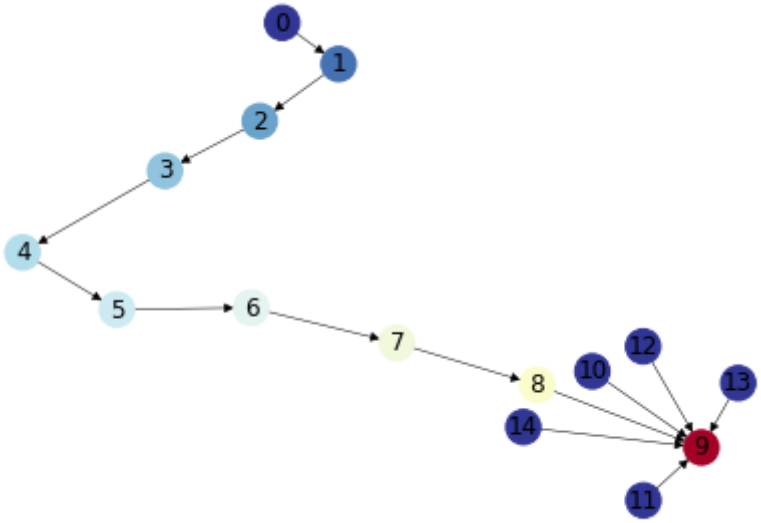

```
def graph_15_2():
    ''' This method, should construct and return a DirectedGraph encoding example 15.2
        Use the following indexes: A:0, B:1, C:2, A':3, B':4, C':5'''
    g = DirectedGraph(6)
    g.add_edge(0, 1)

    g.add_edge(1, 2)
    g.add_edge(2, 0)
    g.add_edge(3, 4)
    g.add_edge(4, 5)
    g.add_edge(5, 3)
    return g
show_graph(graph_15_2(),1)
```

```
page ranks:  [0.16666667 0.16666667 0.16666667 0.16666667 0.16666667 0.16666667]
Lowest to highest ranks:  [0 1 2 3 4 5]
```




```
#test 1
#testing
g = DirectedGraph(15)
g.add_edge(0,1)
g.add_edge(1,2)
g.add_edge(2,3)
g.add_edge(3,4)
g.add_edge(4,5)
g.add_edge(5,6)
g.add_edge(6,7)
g.add_edge(7,8)
g.add_edge(8,9)
g.add_edge(10,9)
g.add_edge(11,9)
g.add_edge(12,9)
g.add_edge(13,9)
g.add_edge(14,9)


show_graph(g,3)
```

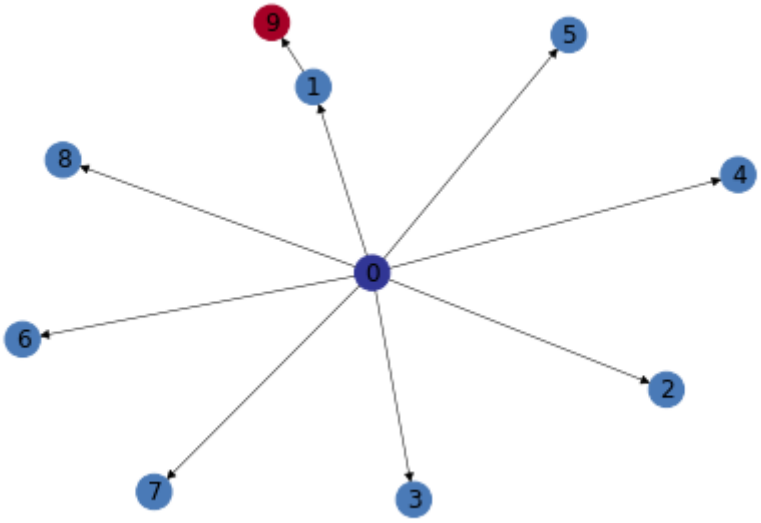
```
page ranks:  [0.00952381 0.01768707 0.02468416 0.03068166 0.03582238 0.0402287
 0.04400555 0.04724286 0.05001769 0.09321244 0.00952381 0.00952381
 0.00952381 0.00952381 0.00952381]
Lowest to highest ranks:  [ 0 10 11 12 13 14  1  2  3  4  5  6  7  8  9]
```




```
#test 1
#testing
g = DirectedGraph(10)
g.add_edge(0,1)
g.add_edge(0,2)
g.add_edge(0,3)
g.add_edge(0,4)
g.add_edge(0,5)
g.add_edge(0,6)
g.add_edge(0,7)
g.add_edge(0,8)
g.add_edge(1,9)
```

```
show_graph(g,1)
```

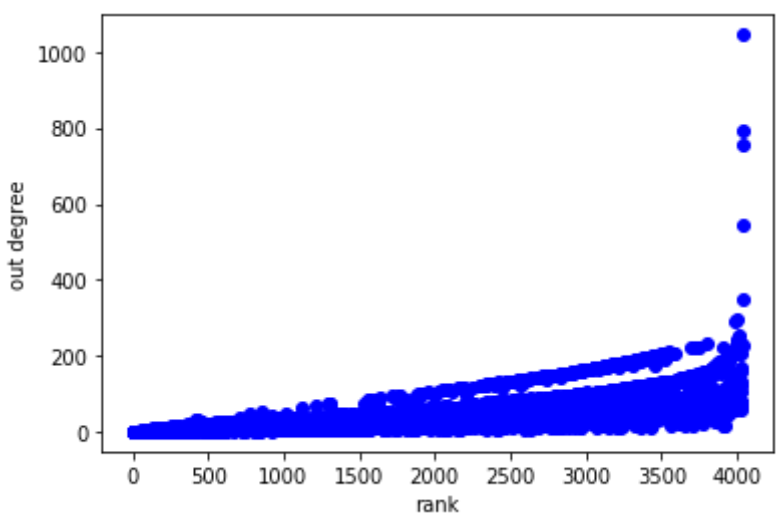
```
page ranks:  [0.01428571 0.01581633 0.01581633 0.01581633 0.01581633 0.01581633
 0.01581633 0.01581633 0.01581633 0.02784257]
Lowest to highest ranks:  [0 1 2 3 4 5 6 7 8 9]
```




```
def facebook_graph(filename = "facebook_combined.txt"):
    ''' This method should return a DIRECTED version of the facebook graph as an insta
    In particular, if u and v are friends, there should be an edge between u and v and
    with open(filename, mode="r") as f:
        content = f.readlines()
    content = [x.strip().split(' ') for x in content]

    g = DirectedGraph(4039)
    for edge in content:
        g.add_edge(origin_node=int(edge[0]), destination_node=int(edge[1]))
        g.add_edge(origin_node=int(edge[1]), destination_node=int(edge[0]))

    return g


def question8b():
    # Load Facebook graph
    filepath = "facebook_combined.txt"
    FB_g = facebook_graph(filename = filepath)

    # Run PR for 20 iterations
    pr = scaled_page_rank(graph = FB_g, num_iter=20)

    return pr


#TEST
facebook_ranks = question8b()


fb_ranks_sorted = np.argsort(facebook_ranks)
```

```
facebook_directed = facebook_graph()
```

```
for rank in fb_ranks_sorted:
  plt.plot(rank,facebook_directed.out_degree(fb_ranks_sorted[rank]),'bo')
plt.ylabel('out degree')
plt.xlabel('rank')
plt.show()
```




```
facebook_network = nx.Graph()
facebook_network.add_nodes_from(np.arange(facebook_directed.number_of_nodes()))
for node in range(facebook_directed.number_of_nodes()):
  for to in facebook_directed.edges_from(node):
    facebook_network.add_edge(node,to)

colors = np.full(facebook_directed.number_of_nodes(),'b')
for node in fb_ranks_sorted[-100:]:
    colors[node]='r'
for node in fb_ranks_sorted[:500]:
    colors[node]='g'
```

```
plt.figure(1,figsize=(10,10))
nx.draw(facebook_network, node_size=20, width=.05, with_labels=False, node_color=colo
plt.show()
```