PD Dr. Ralf Klamma
A. Neumann
M. Kißgen
L. Pieschel
I. Ni
B. Zhvania

05.11.2021

Informatik 5 - Informationssysteme und Datenbanken
RWTH AACHEN UNIVERSITY

Sources of Further Information

The following links lead to documentation that may be helpful/necessary for solving the tasks.

Python Documentation (https://docs.python.org/3/)
NumPy Reference (https://numpy.org/doc/stable/reference/index.html)
NetworkX Reference (https://networkx.org/documentation/stable/reference/index.html)
Matplotlib (https://matplotlib.org/stable/contents.html)

Prerequisites

Run the following cell every time you open the notebook to ensure everything works as intended.

In [3]:

```
import numpy as np
import math
import matplotlib.pyplot as plt
import networkx as nx
from scipy import sparse
```

Remarks for Jupyter Notebook exercises

Please read these carefully

1. Do **not** remove, duplicate or add cells.
2. As most of these exercises are automatically graded, please **read the exercises carefully** and use the proposed variable and function names.
3. Autograded exercises are followed by a test cell. This cell is a placeholder for hidden unit-tests. Sometimes some of the tests will be visible such that you can check parts of your solution. Otherwise just ignore these cells.
4. **Do not compress** or otherwise manipulate the filetype (.ipynb). Just fill in your answers and upload it to moodle directly.
5. Have fun and play around but check that there are no errors and everything is correct before handing in your solution.
6. **Comment your code!** This helps you and me find errors and even erroneous code may give points if it contains comprehensible comments.

Not following these remarks may result in lost points. If you have any questions concerning the exercises or the grading, please use the exercise forum in moodle.

Happy coding!

Social Computing Exercise 1 [14 points]

```
## **Matriculation numbers**
Please fill in the names and matriculation numbers of your team:(firstname,surname)
1. Le Zhang 437042
2. Xinyu Liu 433474
3. Xing Fan 417037
```

Exercise 1.1: Recommender Systems and Collaborative Filtering [9 points]

The RWTH wants to implement a novel system to recommend courses to students. This system will incorporate a rating system in which students may rate every attended course with a score from 1 (did not like) to 10 (absolutely loved). Additionally, a score of 0 means that the course wasn't rated. The system will then output a list of courses which the student may also like based on their already rated courses and the preferences of other similar students. You will implement this system step-by-step using the algorithms presented in the lecture.

1.1a: Similarity Measure [3.5 points]

First we need to be able to tell how similar two given students are. For this, we will use the *Pearson Correlation Coefficient*. Implement the body of the function `w = pearson_correlation(user1, user2, ratings)` where:

- `user1` and `user2` are integer indices of both students
- `ratings` is a integer matrix (Numpy Array) where each row represents a student and each column represents a lecture. The matrix contains the ratings from 0 to 10 as explained above (0: not rated, 1: bad lecture, 10: good lecture). So a value `ratings[3,0] = 2` means student 4 rated lecture 1 with score 2.
- The returned `w` is the Pearson Correlation Coefficient of both given users as explained in the lecture.

Hints:

- Only regard lectures which both students rated. This also applies to average user ratings.
- NumPy is your friend! Functions like `np.mean`, `np.sum`, `np.multiply`, `np.square` and `np.sqrt` will make this and the following exercises a lot easier.
- There are some edge cases in which the Pearson Correlation is undefined. If there are no lectures which both students rated and if a student gave all lectures which both rated the same score (in this case the denominator of the formula will be 0). You have to check these cases and return 0.

In [4]:

```
student2 is represented by the row index of matrix ratings.
is the whole data matrix of students' ratings
on_correlation(student1, student2, ratings):
he list of the lectures that are both rated
ated = []
in range(len(ratings[0])):
 ratings[student1][i] != 0 and ratings[student2][i] != 0:
  bothRated.append(i)
k edge case1: If there are no lectures which both students rated
(bothRated) == 0 :
eturn 0
the mean of two arrays respectively
ingStu1 = 0
ingStu2 = 0
in bothRated:
gRatingStu1 += ratings[student1][i]
gRatingStu2 += ratings[student2][i]
ingStu1 /= len(bothRated)
ingStu2 /= len(bothRated)
the formula of pearson_correlation
tor = 0
natorStu1 = 0
natorStu2 = 0
in bothRated:
merator += (ratings[student1][i] - avgRatingStu1)*(ratings[student2][i] - avgRatingS
nominatorStu1 += np.square(ratings[student1][i] - avgRatingStu1)
nominatorStu2 += np.square(ratings[student2][i] - avgRatingStu2)
natorStu1 = np.sqrt(denominatorStu1)
natorStu2 = np.sqrt(denominatorStu2)
k edge case2: if a student gave all lectures which both rated the same score
ominatorStu1 == 0 or denominatorStu2 == 0:
eturn 0
on_correlation = numerator/(denominatorStu1 * denominatorStu2)
(float(pearson_correlation))
```

In [5]:

```
# This is a test cell. You can execute it to see whether your function works as inte
# Example from lecture
test_matrix = np.array([[4,0,5,5],[4,2,1,0],[3,0,3,4],[4,4,0,0],[2,1,3,5]])
res = pearson_correlation(0,4,test_matrix)
assert isinstance(res, float), "The result should be a floating point number."
expect = 0.7559
assert math.isclose(res, expect, rel_tol=0.001), f"There seems to be an error. Expec
print("Your solution seems to be correct.")
# Hidden tests
```

```
Your solution seems to be correct.
```

1.1b: Neighborhood [1 point]

Now we need to determine which students are most similar to a given student. We will call this set neighborhood of a student. For this, we will compute the correlation measure for each student to the given student and then choose the top k students as neighborhood. Implement the body of the function `neighbors = neighborhood(student, k, ratings)` where:

- `student` is the student for whom we want to find the neighborhood.
- `k` is the cardinality of the neighborhood.
- `ratings` is the rating matrix as explained above.
- `neighbors` is either a python list or a NumPy array with size `k` containing the neighbors (as indices) of `student`. The order is irrelevant as long as it contains the `k` most similar students.

Hints:

- A student should not be contained in their own neighborhood even though their correlation to themself should always be 1
- The NumPy functions `np.argpartition` or `np.argsort` may help.

In [6]:

```
def neighborhood(student, k, ratings):
    correlations = []
    for i in range(len(ratings)):
        if i == student:
            #if it's the correlation to themself, set the correlations to infinite s
            #to avoid influence to the result
            correlations.append(float('-inf'))
        else:
            correlations.append(pearson_correlation(student,i,ratings))
    neighbors = np.argsort(correlations)[-k:]
    return neighbors

try:
    test_matrix = sparse.load_npz("ratings.npz").toarray()
except:
    print("Could not load matrix. Is ratings.npz present in the same folder as the r
res = set(neighborhood(0, 10, test_matrix))
```

In [7]:

```
This is a test cell. You can execute it to see whether your function works as intend
y:
  test_matrix = sparse.load_npz("ratings.npz").toarray()
cept:
  print("Could not load matrix. Is ratings.npz present in the same folder as the not
s = set(neighborhood(0, 10, test_matrix))
sert isinstance(res, set), "Could not construct set from result. Use a NumPy array o
pect = set([98, 99, 4, 41, 46, 15, 49, 54, 29, 63])
sert(res == expect), f"There seems to be an error. Expected {expect}, got {res}"
int("Your solution seems to be correct.")
Hidden tests
```

```
Your solution seems to be correct.
```

1.1c: Prediction [3 points]

Using the ratings of students in the neighborhood, we can predict how the target student may rate a given lecture without him ever taking it. For this we will use the *Adjusted Weighted Average* of students in the neighborhood of the target student, weighted by their similarity to said student. Implement the body of the function `score = predict(student, lecture, n_neighbors, ratings)` where:

- `student` is the target student.
- `lecture` is the target lecture.

- `n_neighbors` is the number of neighbors which we want to incorporate.
- `ratings` is the rating matrix as explained above.
- `score` is the adjusted weighted average as shown in the lecture.

Hints:

- A rating of 0 means "not rated" and does not influence the average rating of a student.

In [8]:

```
def average_without_zero(a):
    #caculate the number of 0 in the list of ratings
    num_zeros = (a == 0).sum()
    #denominator of average is reduced by the number of 0
    return sum(a)/(len(a) - num_zeros)

def predict(student, lecture, n_neighbors, ratings):
    neighbors = neighborhood(student,n_neighbors,ratings);
    numerator = 0
    denominator = 0
    avgTarget = average_without_zero(ratings[student])
    for n in neighbors:
        avgNeighbor = average_without_zero(ratings[n])
        numerator += (ratings[n][lecture] - avgNeighbor) * pearson_correlation(stude
        denominator += pearson_correlation(student,n,ratings)
    predict = avgTarget + numerator/denominator
    return float(predict)
```

In [9]:

```
# This is a test cell. You can execute it to see whether your function works as inte
try:
    test_matrix = sparse.load_npz("ratings.npz").toarray()
except:
    print("Could not load matrix. Is ratings.npz present in the same folder as the r
res = predict(15,32,10,test_matrix)
assert isinstance(res, float), "The result should be a floating point number."
expect = 6.6061
assert math.isclose(res, expect, rel_tol=0.001), f"There seems to be an error. Expec
print("Your solution seems to be correct.")
# Hidden tests
```

```
Your solution seems to be correct.
```

1.1d: Recommendation [1.5 points]

Now that we can predict how a target student may rate a lecture, we can recommend them a set of lectures. We can just predict the rating for every untaken lecture and then return the k highest rated lectures. Implement the body of the function `top_k = recommend(student, k, neighbors, ratings)` where:

- `student` is the target student.
- `k` is the number of lectures to recommend.
- `neighbors` is the size of the neighborhood.
- `ratings` is the rating matrix as explained above.
- `top_k` is a Python list or NumPy array of lectures as indices. The order is irrelevant as long as the `k` highest rated lectures are contained.

Hints:

- Do not include already taken lectures but always output k lectures. If there are less than k unrated lectures, undefined behaviour is allowed.
- NumPy functions `np.argpartition` or `np.argsort` may help.

In [10]:

```
def recommend(student, k, neighbors, ratings):
    untaken = []
    for i in range(len(ratings[0])):
        if ratings[student][i] == 0:
            untaken.append(i)
    predicts = []
    for lecture in untaken:
        predicts.append(predict(student,lecture,neighbors,ratings))
    kNearst = np.argsort(predicts)[-k:]
    recommend = []
    for i in kNearst:
        recommend.append(untaken[i])
    return recommend
```

In [11]:

```
# This is a test cell. You can execute it to see whether your function works as inte
try:
    test_matrix = sparse.load_npz("ratings.npz").toarray()
except:
    print("Could not load matrix. Is ratings.npz present in the same folder as the r
res = set(recommend(0,5,10,test_matrix))
assert isinstance(res, set), "Could not construct set from result. Use a NumPy array
expect = set([32, 33, 26, 35, 29])
assert(res == expect), f"There seems to be an error. Expected {expect}, got {res}"
print("Your solution seems to be correct.")
# Hidden tests
```

```
Your solution seems to be correct.
```

Exercise 1.2: TidalTrust [5 points]

Consider TidalTrust from the lecture. It was proposed by J. Golbeck in 2005 for trust-aware recommendation systems. In this trust-aware recommender model, each user can rate items and mention the trust value towards others. In this algorithm, if a node has rated an item then the rating is already known. But if it is unknown, the node asks its neighbors for the recommendation and trust value.

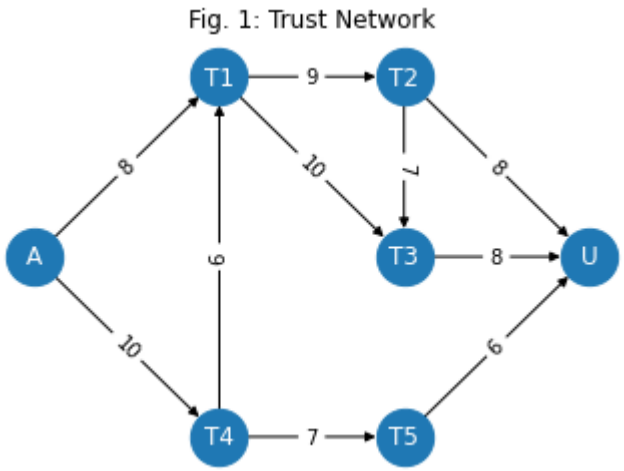
The trust value can be computed based on $t_{a,u} = \frac{\sum_{v \in WOT^+(a)} t_{a,v} t_{v,u}}{\sum_{v \in WOT^+(a)} t_{a,v}}$.

$WOT^+(a)$ represents the set of users for whom a's trust statement exceeds or is equal to the given threshold max.

Consider the following trust network (Fig. 1) (Run the following cell to draw):

In [12]:

```
# Run this cell to draw the graph
g = nx.DiGraph()
g.add_node("A")
g.add_node("T1")
g.add_node("T2")
g.add_node("T3")
g.add_node("T4")
g.add_node("T5")
g.add_node("U")
g.add_edge("A", "T1", weight=8)
g.add_edge("A", "T4", weight=10)
g.add_edge("T1", "T2", weight=9)
g.add_edge("T1", "T3", weight=10)
g.add_edge("T2", "U", weight=8)
g.add_edge("T2", "T3", weight=7)
g.add_edge("T3", "U", weight=8)
g.add_edge("T4", "T5", weight=7)
g.add_edge("T4", "T1", weight=6)
g.add_edge("T5", "U", weight=6)
pos = {"A": (0,0), "T1": (1,1), "T2": (2,1), "T3": (2,0), "T4": (1,-1), "T5": (2,-1)
plt.title("Fig. 1: Trust Network")
nx.draw(g, pos, with_labels=True, node_size=800, font_color="white")
labels = nx.get_edge_attributes(g, "weight")
nx.draw_networkx_edge_labels(g, pos, edge_labels=labels)
plt.show()
```



1.2a: Trust Propagation [2 points]

Compute the trust value $t_{A,U}$ from A to node U.

- $\max = 7$
- $\text{maxdepth} = 3$

Use the given variables and variable names to calculate the solution below.

In [21]:

```
# Given Variables (don't touch these)
t_A_T1 = 8
t_A_T4 = 10
t_T1_T2 = 9
t_T1_T3 = 10
t_T2_T3 = 7
t_T2_U = 8
t_T3_U = 8
t_T4_T1 = 9
t_T4_T5 = 7
t_T5_U = 6

# Use above variables to calculate the variables below. Replace 'None' with your for
t_T1_U = (t_T1_T2 * t_T2_U + t_T1_T3 * t_T3_U)/ (t_T1_T2 + t_T1_T3 )
t_T4_U = (t_T4_T1 * t_T1_U) / t_T4_T1
t_A_U = (t_A_T1 * t_T1_U + t_A_T4 * t_T4_U)/(t_A_T1 + t_A_T4)



# This line just prints your solution
print(f"Your Solution:\nt_T1_U = {t_T1_U}\nt_T4_U = {t_T4_U}\nt_A_U = {t_A_U}")
```

```
Your Solution:
t_T1_U = 8.0
t_T4_U = 8.0
t_A_U = 8.0
```

In [22]:

```
# Hidden tests (ignore this cell)
```

1.2b: Trust Based Prediction [3 points]

Afterwards calculate the predicted rating of node A for item m using trust based prediction:

$$r_{am} = \frac{\sum_{v \in WOT(a)} t_{av} r_{vm}}{\sum_{v \in WOT(a)} t_{av}}$$

Use the following item ratings:

Node	T1	T2	T3	T4	T5	U
Rating	4	5	3	5	4	1

Use the given variables and variable names to calculate the solution below. Also use the variables from **1.2a**.

In [24]:

```
# Given Variables (don't touch these)
r_T1_m = 4
r_T2_m = 5
r_T3_m = 3
r_T4_m = 5
r_T5_m = 4
r_U_m = 1

# Use above variables (also from 1.2a) to calculate the variables below. Replace 'No
t_T4_T2 = (t_T4_T1 * t_T1_T2)/t_T4_T1
t_A_T2  = (t_A_T1 * t_T1_T2 + t_A_T4 * t_T4_T2)/(t_A_T1 + t_A_T4)

#(t_T1_T2 * t_T2_T3)/t_T1_T2
#t_T1_T3 = (t_T1_T3 + t_T1_T2 * t_T2_T3)/t_T1_T2
t_T4_T3 = (t_T4_T1 * t_T1_T3)/t_T4_T1
t_A_T3  = (t_A_T4 * t_T4_T3 + t_A_T1 * t_T1_T3)/(t_A_T4 + t_A_T1)

t_A_T5  = (t_A_T4 * t_T4_T5)/t_A_T4

t_T1_U = (t_T1_T2 * t_T2_U + t_T1_T3 * t_T3_U)/(t_T1_T2 + t_T1_T3)
t_A_U = (t_A_T1 * t_T1_U)/t_A_T1

denominator = t_A_T1 + t_A_T2 + t_A_T3 + t_A_T4 + t_A_T5 + t_A_U
r_A_m = (t_A_T1 * r_T1_m + t_A_T2 * r_T2_m + t_A_T3 * r_T3_m + t_A_T4 * r_T4_m + t_A

# This line just prints your solution
print(f"Your Solution:\nt_A_T2 = {t_A_T2}\nt_A_T3 = {t_A_T3}\nt_A_T5 = {t_A_T5}\nr_A
```

```
Your Solution:
t_A_T2 = 9.0
t_A_T3 = 10.0
t_A_T5 = 7.0
r_A_m = 3.7115384615384617
```

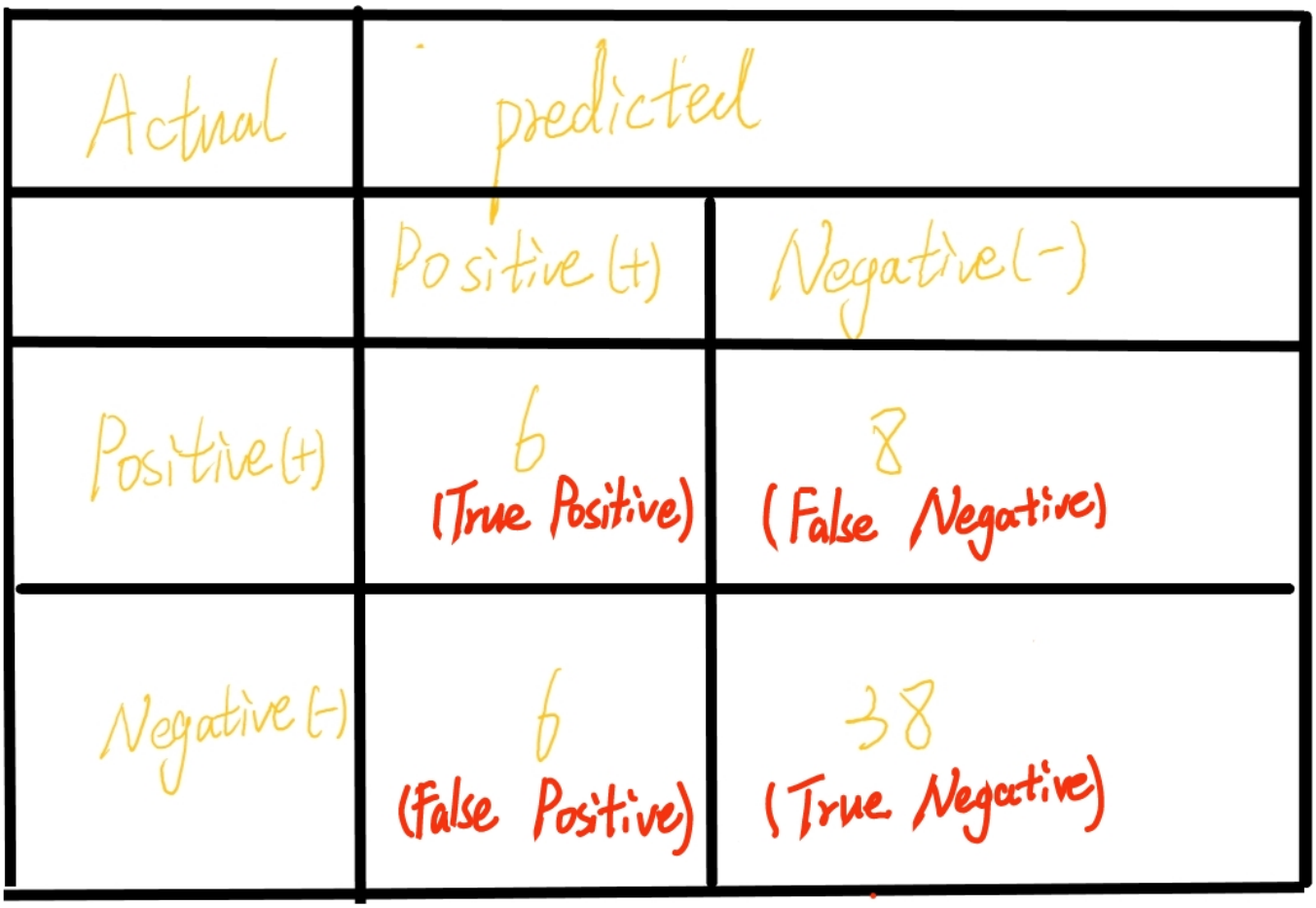
In [25]:

```
# Hidden tests (ignore this cell)
```

Exercise 1.3: Evaluation of Recommendations [4 points]

The car dealer Deals in Wheels has 3,190 customers in his database. He regularly asks his customers to rate their experience they bought from him, and he stores the ratings in his database. He ofers 58 different cars for sale, of which 20 are relevant to Mrs. Williams, who is about to buy a new car. Based on her old car, which she bought from another shop and which newer version is not on stock at Deals in Wheels, the salesman recommends 12 of his cars to her, and she likes 6 of these recommendations.

a) Build the confusion matrix (true/false positives, true/false negatives) for this recommendation.



b) Calculate precision and recall for this recommendation. Briefly interpret the result.

tp = 6 fp = 6 fn = 8 Precision = tp / (tp + fp) = 0.5 Recall = tp /(tp +fn) =0.42857142857142855

Exercise 1.4: Sybil Attacks and SybilRank (7 Points)

Consider the Agure below for an example of applying SybilRank to a network of both Sybil and NonSybil users. Additionally, so called "Trust Seeds" (Users of which is known they can be trusted, in this example node A and B) are given. Please answer the following questions:

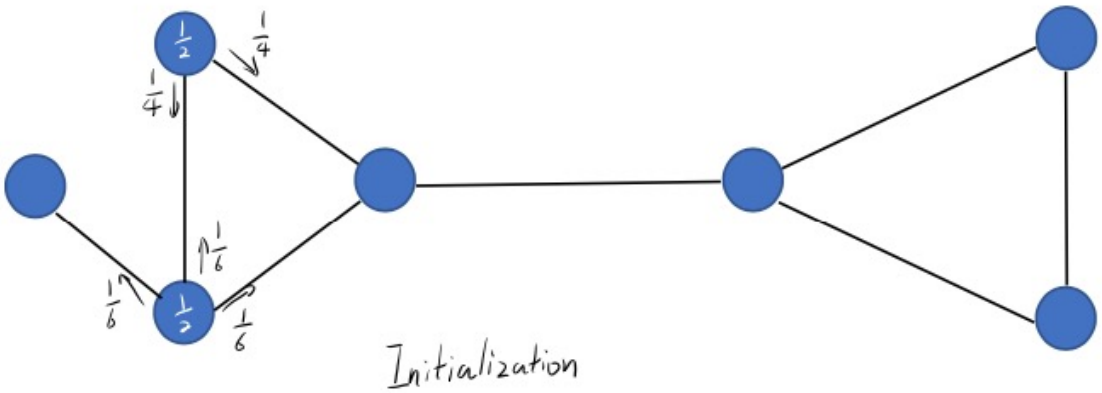
a) Shortly describe the idea of a Sybil Attack and the possible intentions for it.

one peer network has multiple identities to attack some sevices like voting

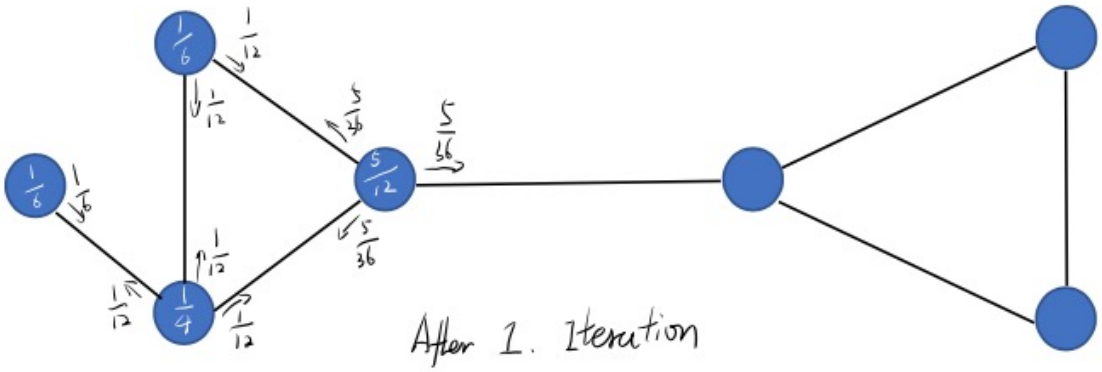
b) Calculate two power iterations for the given example (initial state: A = B =1/2) and provide corresponding ranking.

Initial state: $A = B = \frac{1}{2}$

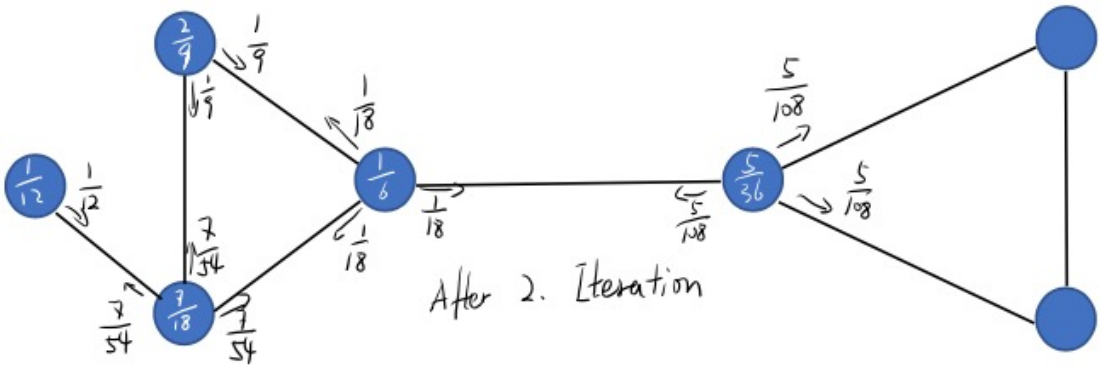
Number of iterations $\lceil \log |V| \rceil = \lceil \log 7 \rceil = 3$



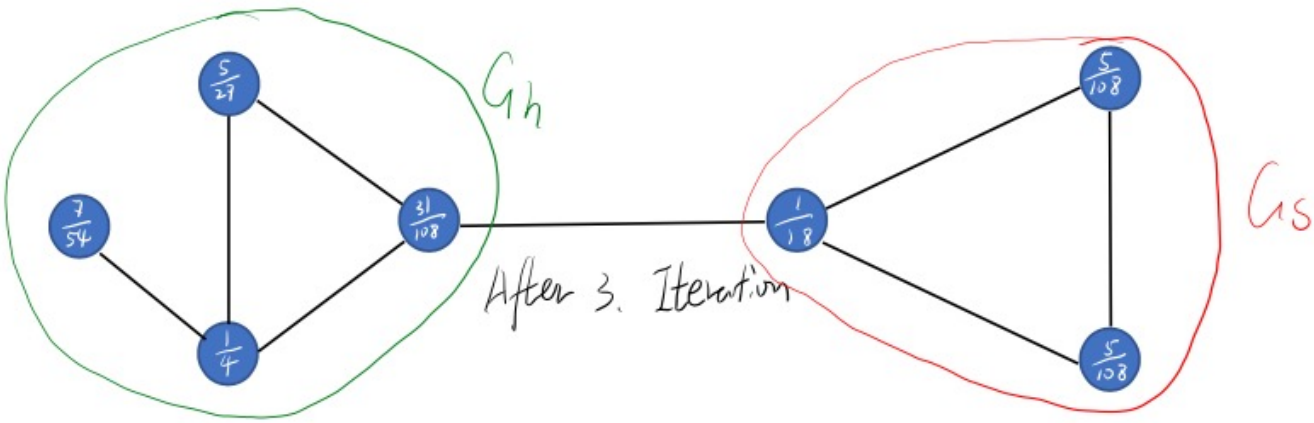
Initialization



After 1. Iteration



After 2. Iteration



After 3. Iteration