9. ניתוח מידע רשתי באמצעות Python

9.1. מטרת היחידה

אחת הדרכים לייצוג מופשט של אובייקטים הקשורים זה לזה היא באמצעות רשת, שבה אובייקטים (צמתים) הקשורים זה לזה (באמצעות קשתות), וביכולתה לבטא בצורה יעילה ספקטרום רחב של בעיות וטופולוגיות. לדוגמה, רשת יכולה לתאר רשת חברתית, שבה צומת מייצג ישות וירטואלית וקשת מייצגת קשר חברי בין זוג ישויות, או לחלופין רשת תחבורה ציבורית שבה צומת הוא תחנה וקשת המחברת צמתים מייצגת את האפשרות לעבור מתחנה לתחנה באמצעות אוטובוס. מטרת יחידה זו היא להקנות את היסודות המעשיים של ניתוח מידע רשתי.

בפרק זה נסקור את הנושאים האלה:

- מבוא לרשתות ב-NetworkX
- מבוא לניתוח רשתות וויזואליזציה ב-NetworkX
- מדדי קישוריות רשתיים
- עמידות (robustness) של רשת
- השפעה ומרכזיות ברשת
- חיזוי שינויים עתידיים במבנה הרשת
- חילוץ מאפיינים מרשת

9.2. בחינה מקדימה טכנית ליחידה

אנא עקבו בקפידה אחר ההנחיות שבסעיף זה. אם יש פער בין המוצג אצלכם לבין המוצג במדריך, חזרו למדריך ההתקנות ופעלו בהתאם להנחיות.

- ודאו גרסת Python (לכל הפחות גרסה 3.6.3, התקנת ברירת מחדל של Anaconda גרסת 5.0.1). תוכלו לבדוק זאת על ידי הרצת:

```
import sys; print(sys.version)
—> 3.6.3 |Anaconda, Inc.|
```

- ודאו גרסת pandas (לכל הפחות גרסה 0.2, התקנת ברירת מחדל של Anaconda גרסת 5.0.1). תוכלו לבדוק זאת על ידי הרצת:

```
import pandas; print (pandas.__version__)
—> 0.20.3
```

- ודאו גרסת matplotlib (לכל הפחות גרסה 2.1.2, התקנת ברירת מחדל של Anaconda גרסת 5.0.1). תוכלו לבדוק זאת על ידי הרצת:

```
import matplotlib as plt; print (plt.__version__)
—> 2.1.1
```

- ודאו גרסת sklearn (לכל הפחות גרסה 0.19.1, התקנת ברירת מחדל של Anaconda גרסת 5.0.1). תוכלו לבדוק זאת על ידי הרצת:

```
import sklearn as skl; print (skl.__version__)
—> 0.19.1
```

- ודאו גרסת NetworkX (לכל הפחות גרסה 2.1, התקנת ברירת מחדל של Anaconda גרסת 5.0.1). תוכלו לבדוק זאת על ידי הרצת:

```
import networkn; print (networkn.__version__)
–> 2.1
```

9.3. ידע קודם נדרש ליחידה

ודאו כי קראתם והבנתם את:

- יחידות הלימוד הקודמות בקורס
- הפרקים בלמידה מפוקחת ולא מפוקחת מן הקורס "מבוא לבינה מלאכותית"

9.4. מבוא לרשתות ב-NetworkX

9.4.1. הגדרת רשתות ב-NetworkX

ראשית, נייבא את ספריית networkX ונאפשר הדפסה של גרפים על גבי ה-Jupyter notebook:

```
import networkx as nx
%matplotlib notebook
```

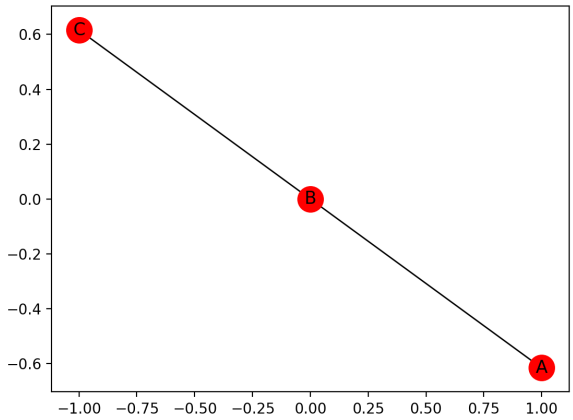
נדגים את הגדרתו של **גרף לא מכוון** בעל 3 צמתים A, B, C ושתי קשתות, בין A ל-B ובין B ל-C, באמצעות אובייקט Graph:

```
G = nx.Graph()
G.add_edge('A', 'B')
G.add_edge('B', 'C')
```

שימו לב, אם הצמתים לא הוגדרו קודם לכן, הם יוגדרו אוטומטית תוך הגדרת הקשתות.

אם הרשת פשוטה מספיק, נוכל להדפיסה כך:

```
nx.draw_networkx(G)
```



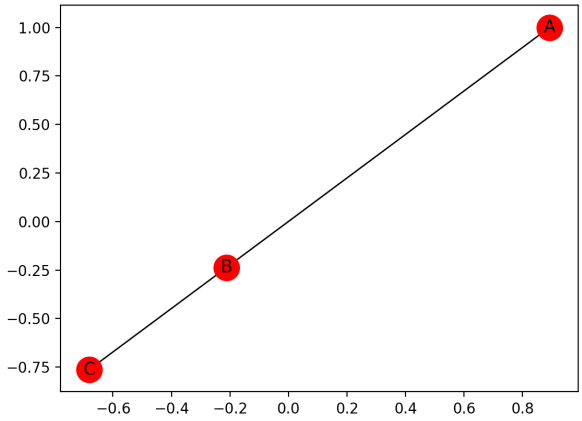
נוכל להגדיר באופן דומה **רשת מכוונת**. מובן שבעת הגדרתה, תהיה כעת חשיבות לסדר הצמתים שנזין לפונקציית יצירת הקשת. נגדיר את הרשת באמצעות אובייקט DiGraph:

```
G1 = nx.DiGraph()
G1.add_edge('B', 'A')
G1.add_edge('B', 'C')
```

אם נדפיס את הרשת כעת, יהיו חיצים שיביעו את כיווניות הקשתות.

באופן דומה, נוכל להגדיר **רשת ממושקלת**. במקרה כזה, לכל קשת יהיה משקל. נעשה זאת באמצעות הוספת attribute בשם weight. נדפיס את הרשת ונראה שהמרחק בין הצמתים מבטא את משקל הקשת המחברת ביניהם.

```
G = nx.Graph()
G.add_edge('A', 'B', weight = 6)
G.add_edge('B', 'C', weight = 60)
nx.draw_networkx(G)
```



פעמים רבות יהיה זה שימושי להגדיר **רשת מסומנת**. ברשת מסוג זה הקשתות יהיו חיוביות (דוגמת like בביקורת סרטים), או שליליות (dislike). נעשה זאת באמצעות הגדרת attribute מסוג sign.

```
G = nx.Graph()
G.add_edge('A', 'B', sign = '+')
G.add_edge('B', 'C', sign = '-')
```

פעמים רבות יהיה זה שימושי במיוחד להגדיר את סוג היחס שמתארת הקשת. למשל, קשת יכולה לבטא קשר משפחתי, קשר עבודה או קשר ידידות. נוכל להגדיר את סוג היחס שמוגדר על ידי הקשת כך:

```
G = nx.Graph()
G.add_edge('A', 'B', relation = 'friend')
G.add_edge('B', 'C', relation = 'coworker')
G.add_edge('D', 'E', relation = 'family')
```

לעיתים נרצה להגדיר קשרים מרובים בין אותם שני צמתים. למשל, שני אובייקטים שהם גם חברים וגם בני משפחה. רשת המאפשרת ריבוי יחסים בין צמתים נקראית **מולטי רשת**. ההגדרה מבוצעת באמצעות אובייקט MultiGraph, כך:

```
G = nx.MultiGraph()
G.add_edge('A', 'B', relation = 'friend')
G.add_edge('A', 'B', relation = 'coworker')
G.add_edge('C', 'D', relation = 'family')
```

נוכל להגדיר **מולטי-רשת ממושקלת** באמצעות הוספת attribute, כפי שעשינו למעלה.

```
G = nx.MultiGraph()
G.add_edge('A', 'B', relation = 'friend',weight = 6)
G.add_edge('A', 'B', relation = 'coworker', weight = 4)
```

נוכל להגדיר באופן דומה מאוד **מולטי-רשת ממושקלת מכוונת**, באמצעות יצירת אובייקט מסוג MultiDiGraph.

```
G = nx.MultiDiGraph()
G.add_edge('A', 'B', relation = 'friend',weight = 6)
G.add_edge('A', 'B', relation = 'coworker', weight = 4)
```

ניתן להגדיר attributes גם לצמתים עצמם. למשל, ברשת יחסי עבודה נוכל לסווג חלק מן הצמתים כמנהלים וחלק כעובדים. ניתן לעשות זאת במסגרת פונקציית add_node, שבאמצעותה נוסיף attributes לצומת גם אם הוא כבר הוגדר.

```
G = nx.Graph()
G.add_edge('A', 'B', relation = 'family', weight = 6)
G.add_edge('B', 'C', relation = 'friend', weight = 13)
G.add_node('A', role='manager')
G.add_node('B', role='worker')
```

ברשת דו-צדדית (bipartite graph), אפשר לחלק את הצמתים לשתי קבוצות כך שכל הקשתות הן בין צמתים מקבוצות שונות ואין שום קשת המחברת שני צמתים מאותה הקבוצה. לרשת מסוג זה יכולת ביטוי שיכולה לאפשר ניתוח נתונים רשתיים המבטאים יחסי התאמה. למשל, קבוצה של סטודנטים וקבוצה של קורסים. סטודנטים בגרף זה לא יהיו מקושרים זה לזה, וגם קורסים לא יהיו מקושרים זה לזה, אבל קשר בין סטודנט לקורס ילמד על כך שהסטודנט למד אותו במסגרת לימודיו. אנו ניצור גרף דו-צדדי כמו שעשינו קודם לכן, אך ראשית, נייבא את היכולת המתאימה מ-networkx ונשתמש בה בעת יצירת הרשת.

נדגים זאת: הפעם נייצר את הרשת מתוך רשימה בדרך המאפשרת להגדיר רשתות גדולות בקלות רבה יותר.

```
from networkx.algorithms import bipartite
B = nx.Graph() # no seperate class for bipartite
B.add_nodes_from(['A', 'B', 'C', 'D'], bipartite = 0)
B.add_nodes_from([1, 2, 3, 4], bipartite = 1)
B.add_edges_from([('A', 1),('A', 2), ('B', 1), ('C', 1), ('C', 2),
                  ('D', 4), ('D', 3), ('D', 2)])
```

כעת נוכל לגשת ל-bipartite ולבחון אם הגרף המבוטא ברשת הוא דו-צדדי:

```
bipartite.is_bipartite(B)
->
True
```

כעת נוסיף קשת בקבוצת הסטודנטים ונוודא שהגרף אינו דו-צדדי. לאחר שנעשה זאת, נסיר את הקשת באמצעות פונקציית remove_edge.

```
B.add_edge('A', 'B')
print(bipartite.is_bipartite(B))
B.remove_edge('A', 'B')
print(bipartite.is_bipartite(B))
->
False
True
```

ייתכן שתת-רשת (node_set) היא דו-צדדית, בעוד שהרשת עצמה אינה כזאת. נראה כיצד לבדוק זאת.

```
X = set([1, 2, 3, 4])
X2 = set(['A', 'B', 'C', 'D'])
X3 = set(['A', 'B', 'C', 'D', 1])
print(bipartite.is_bipartite_node_set(B, X))
print(bipartite.is_bipartite_node_set(B, X2))
print(bipartite.is_bipartite_node_set(B, X3))
->
True
True
False
```

פעמים רבות יהיה זה שימושי לבחון אם קיימת חלוקה יחידה של הרשת כך שכל הקשתות מחברות רק בין צמתים מקבוצות שונות, אם הדבר אינו ידוע.

```
bipartite.sets(B)
->
({'A', 'B', 'C', 'D'}, {1, 2, 3, 4})
```

9.4.2. ייבוא רשתות ב-NetworkX

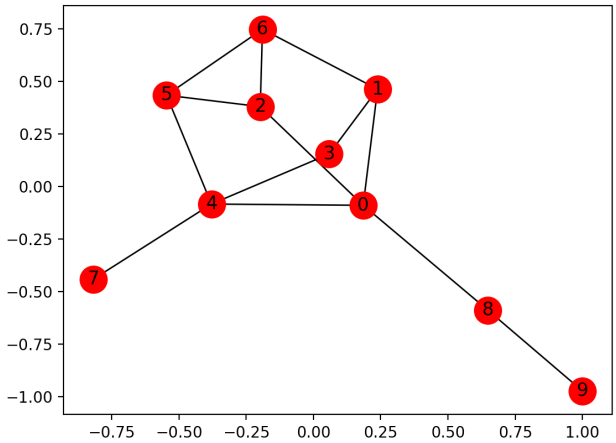
לרוב, כאשר נרצה להגדיר רשת, לא נרצה להגדיר אותה ב-Python, צומת אחר צומת, קשת אחר קשת. תחת זאת, נעדיף לייבא את הרשת מקובץ חיצוני. נבחן כמה אופנים שבהם ניתן לייצג רשת ואת האופן שבו ניתן לייבא אותה ל-networkX.

דרך אחת לייצג גרף היא באמצעות Adjacency List. זוהי רשימה שבה כל שורה מייצגת צומת ואת הצמתים המחוברים אליה. למשל, השורה 5 4 2 1 מבטאת צומת מספר 1 המחובר באמצעות קשתות לצמתים 2, 4 ו-5. ניתן להגדיר את הרשימה בקובץ טקסט, ואז לבנות באמצעותו רשת ב-networkX. נכתוב לדוגמה את קובץ הטקסט adjacency_list.txt עם הרשימה:

```
0 1 2 4 8
1 3 6
2 5 6
3 4
4 5 7
5 6
6
7
8 9
9
```

נשתמש בקובץ כדי להגדיר את הרשת ב-networkX ונדפיס אותה.

```
G = nx.read_adjlist('adjacency_list.txt', nodetype=int)
nx.draw_networkx(G)
```



דרך נוספת היא באמצעות מטריצת סמיכויות, שבה האיבר (i,j) הוא 1 אם קיימת קשת בין i ל-j, ו-0 אם אין קשת כזאת:

```
G2 = nx.to_numpy_matrix(G)
print(G2)
->
[[0. 1. 1. 1. 1. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 1. 1. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 1. 1. 0. 0.]
 [1. 0. 0. 0. 0. 1. 0. 1. 1. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 1. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 1. 1. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 1. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]]
```

אם הרשת נתונה באמצעות מטריצה, נוכל לייבא אותה בקלות ל-networkX כך:

```
G = nx.Graph(G2)
G.edges()
->
EdgeView([(0, 1), (0, 2), (0, 3), (0, 4), (1, 5), (1, 6), (2, 6), (2, 7), (3, 5), (3,
7), (3, 8), (4, 9), (6, 7)])
```

נעיר שניתן לתאר כך גם רשת ממושקלת, אם במקום 1 (שציין כעת קשת בין צומת i לצומת j, כאשר i, j הם האינדקסים שלו במטריצה) נציין את משקל הקשת - מספר - למשל, 4.

דרך אחרת לייצוג רשת היא באמצעות רשימת קשתות. ברשימה זו, כל שורה מייצגת קשת. זאת כאשר ניתן לציין בעמודה נוספת מהו משקל הקשת. נכתוב את הרשימה בקובץ טקסט: edge_list.txt:

```
0 1 4
0 2 3
0 4 5
0 8 1
1 3 1
1 6 2
2 5 2
2 6 2
3 4 2
4 5 6
4 7 7
5 6 1
8 9 1
```

ואז נייבא אותו:

```
G = nx.read_edgelist('edge_list.txt', data = [('weight', int)])
G.edges(data=True)
->
EdgeDataView([('0', '1', {'weight': 4}), ('0', '2', {'weight': 3}), ('0', '4',
{'weight': 5}), ('0', '8', {'weight': 1}), ('1', '3', {'weight': 1}), ('1', '6',
{'weight': 2}), ('2', '5', {'weight': 2}), ('2', '6', {'weight': 2}), ('4', '3',
{'weight': 2}), ('4', '5', {'weight': 6}), ('4', '7', {'weight': 7}), ('8', '9',
{'weight': 1}), ('6', '5', {'weight': 1})])
```

ניתן לייבא מידע רשתי ל-networkX מתוך data frame שהוגדר באמצעות pandas. לצורך הדוגמה, נקרא את קובץ הטקסט הנ"ל אל data frame באמצעות פונקציית read_csv. נעשה זאת כך:

```
G_df = pd.read_csv('edge_list.txt', delim_whitespace=True,
                   header=None, names=['node1', 'node2', 'weight'])
G_df.head()
```

נייבא את ה-data frame אל networkX באופן הזה:

```
G = nx.from_pandas_edgelist(G_df, 'node1', 'node2', edge_attr='weight')
```

9.4.3. בחינת רשתות ב-NetworkX

נבחן כעת כיצד ניתן לגשת לנתונים בתוך רשת. אל רשימת הקשתות ניתן לגשת באמצעות פונקציית edges שתחזיר אובייקט מסוג EdgeView שיכיל אותן. אם נפנה לפונקציה עם data = True, הפונקציה תחזיר EdgeDataView שיכיל את הרשימה, יחד עם ה-attributes של כל קשת. כמו כן, ניתן לבקש את רשימת הרשתות עם מידע על attribute מסוים, באמצעות אתחול data בהתאם.

```
G = nx.MultiGraph()
G.add_edge('A', 'B', relation = 'friend')
G.add_edge('A', 'B', relation = 'coworker')
G.add_edge('C', 'D', relation = ‘family')
print(G.edges())
print(G.edges(data=True))
print(G.edges(data=‘relation'))
->
[('A', 'B'), ('A', 'B'), ('C', ‘D')]
[('A', 'B', {'relation': 'friend'}),
 ('A', 'B', {'relation': 'coworker'}),
 ('C', 'D', {'relation': ‘family'})]
[('A', 'B', 'friend'),
 ('A', 'B', 'coworker'),
 ('C', 'D', 'family')]
```

באופן דומה, נוכל לגשת אל רשימת הצמתים.

```
G = nx.Graph()
G.add_edge('A', 'B', relation = 'family', weight = 6)
G.add_edge('B', 'C', relation = 'friend', weight = 13)
G.add_node('A', role='manager')
G.add_node('B', role=’worker')
G.nodes(data=True)
print(G.nodes(data=True))
->
[('A', {'role': 'manager'}),
 ('B', {'role': 'worker'}),
 ('C', {})]
```

ניתן לגשת אל כל צומת בנפרד ולבחון את הצמתים הקשורים אליו ואת ה-atrributes הרלוונטיים. זאת תוך התייחסות לאובייקט הרשת שיצרנו כמילון.

```
print(G['A'])
print(G['A']['B'])
print(G’A']['B']['weight'])
->
{'B': {'relation': 'family', 'weight': 6}}
{'relation': 'family', 'weight': 6}
6
```

נשים לב לכך, שבמקרה שקשת היא חסרת כיוניות, נוכל לגשת אליה משני הכיוונים. היינו, הן באמצעות G[‘A’][‘B’] והן באמצעות G[‘B’][‘A’]. לעומת זאת, בקשת שהכיוון בה חשוב, כתיבה בסדר ההפוך לכיוון הקשת תייצר הודעת שגיאה. בדקו זאת.

במקרה של מולטי-רשת, יכולות להיות כמה קשתות בין צומת לצומת. במקרה זה, נתייחס לאינדקס של הקשת. למשל:

```
G = nx.MultiGraph()
G.add_edge('A', 'B', relation = 'friend')
G.add_edge('A', 'B', relation = ’coworker')
print(G['A']['B'][0]['relation'])
print(G[‘A']['B'][1]['relation'])
->
friend
coworker
```

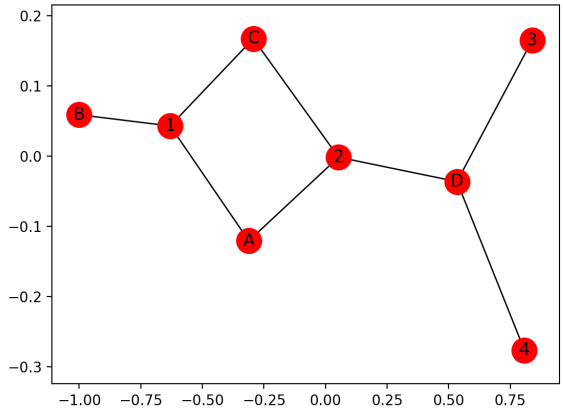
9.5. מבוא לניתוח רשתות ב-NetworkX

9.5.1. הטלה של גרף דו-צדדי

בסעיף הקודם הסברנו כיצד ניתן לייצר גרף דו-צדדי ב-networkX. אחד מן השימושים התדירים בגרף דו-צדדי הוא הטלתו על אחת מבין האוכלוסיות. נחזור לדוגמת הסטודנטים המקושרים לקורסים שאותם למדו בשנים האחרונות. כזכור, הסטודנטים אינם קשורים זה לזה. עם זאת, נאמר שהאוניברסיטה רוצה לבחון את המשותף בין סטודנטים שלקחו את אותם קורסים - נאמר לצורכי שיווק. נוכל להגדיר קשר בין שני סטודנטים אם למדו את אותו הקורס. כלומר, נטיל (projection) את הגרף הדו-צדדי המקורי על גרף רגיל.

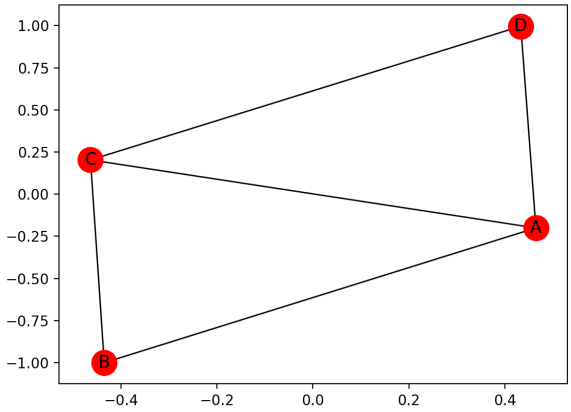
ראשית נטען את גרף הסטודנטים-קורסים ונבחן אותו.

```
B = nx.Graph()
B.add_nodes_from(['A', 'B', 'C', 'D'], bipartite = 0)
B.add_nodes_from([1, 2, 3, 4], bipartite = 1)
B.add_edges_from([('A', 1),('A', 2), ('B', 1), ('C', 1), ('C', 2), ('D', 4), ('D', 3),
('D', 2)])
nx.draw_networkx(B)
```



נטיל את הגרף על קבוצת הסטודנטים:

```
X = set(['A', 'B', 'C', 'D'])
L = bipartite.projected_graph(B, X)
nx.draw_networkx(L)
```



ניתן למשקל את ההטלה וכך להפוך אותה למעט יותר אינפורמטיבית. נאמר שהסטודנטים שלמדו מספר רב יותר של קורסים משותפים יהיו מחוברים בקשת בעלת משקל גבוה יותר מאלו שלמדו פחות קורסים משותפים. נעשה זאת כך:

```
X = set(['A', 'B', 'C', 'D'])
L = bipartite.weighted_projected_graph(B, X)
L.edges(data=True)
->EdgeDataView([('A', 'B', {'weight': 1}), ('A', 'D', {'weight': 1}), ('A', 'C',
{'weight': 2}), ('B', 'C', {'weight': 1}), ('D', 'C', {'weight': 1})])
```

מכיוון שסטודנט A וסטודנט C למדו שני קורסים משותפים, הקשת המחברת אותם היא בעלת משקל כפול מהאחרות.

ניתן כמובן להטיל את הגרף בכיוון השני: בניית גרף המבטא את הקשר בין הקורסים על ידי בחינת מספר הסטודנטים שלקחו את שניהם. נסו זאת.

9.5.2. מקדמי clustering רשתיים

ביישומים רבים, מתברר כי ישויות הקשורות לאותה ישות הן גם קשורות ביניהן. למשל, ברשתות החברתיות, סביר שהחברים שלנו גם חברים ביניהם. תכונה זו מכונה סגור בדרגה 3 (triadic closure).

מדד חשוב אחד הקשור לנושא הוא ה-local clustering coefficient. בהינתן צומת v, המדד מחשב את אחוז הצמתים מבין הצמתים הקשורים ל-v, שקשורים לפחות לעוד צומת אחד מבין הצמתים הקשורים ל-v. נבנה רשת ונחשב את המקדמים באמצעות פונקציית clustering של networkx. לצורך העניין, נשתמש ברשת שהגדרנו למעלה בקובץ adjacenct_list.txt. תמונת הרשת הובאה למעלה.

```
print(nx.clustering(G, 0))
print(nx.clustering(G, 2))
print(nx.clustering(G, 7))
->
0
0.3333333333333333
0
```

נשים לב שצומת 2 הוא המקושר ביותר, ובעל מדד clustering של 1/3. לצומת 0 יש אומנם 3 חברים, אך אלו אינם מחוברים זה לזה ואילו לצומת 7 יש חבר אחד בלבד, ומכיוון שכך אין לו מדד clustering רלוונטי ולכן הוא נקבע כברירת מחדל ל-0.

יכול להיות מעניין לבחון את מידת ה-clustering של הרשת כולה - ולא מנקודת המבט של צומת ספציפי. דרך אחת היא באמצעות ממוצע ערך ה-clustering של כל הצמתים. נעשה זאת כך:

```
print(nx.average_clustering(G))
->
0.1
```

מדד נוסף נקרא transitivity, והוא מבטא את אחוז המשולשים הפתוחים ברשת (שלישייה של צמתים המחוברים על ידי 2 קשתות) ביחס לאלו הסגורים (שלישייה המחוברת על ידי 3 קשתות). נוכל לחשב את ה-transitivity של הרשת כך:

```
print(nx.transitivity(G))
->
0.11538461538461539
```

ההבדל המהותי בין שני המדדים הוא שבעוד ש-transitivity מעניק משקל רב יותר לצמתים בעלי מספר קשתות רב יותר, בממוצע, ערך ה-clustering, מספר הקשתות - או דרגת הצומת - אינו בא לידי ביטוי. חשיבותם של צמתים מקושרים בגרף (מישהו שיש לו חברים רבים ברשת חברתית, למשל) הופכת את ה-transitivity למדד רלוונטי לניתוחים רבים.

9.5.3. מדדי מרחק רשתיים

מסלול (path) הוא סדרה של קשתות שבאמצעותן ניתן "לעבור" מצומת אחד לאחר. כך שאם צומת A מחובר לצומת C שמחובר לצומת B, הרי שיש לנו מסלול מ-A ל-B דרך C. נוכל, למשל, לבחון את כל המסלולים בגרף שבאמצעותם נוכל להגיע מצומת אחד לאחר.

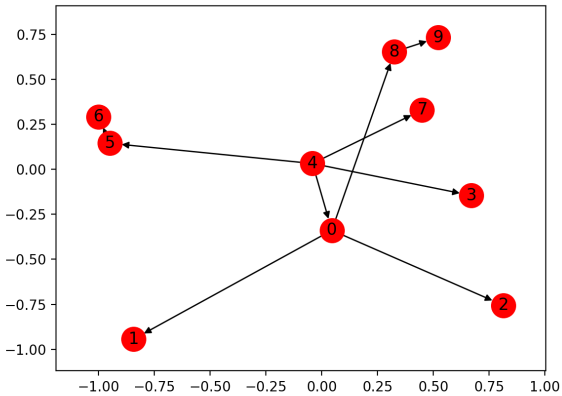
```
print(sorted(nx.all_simple_paths(G, 0, 4)))
->
[[0, 1, 3, 4], [0, 1, 6, 2, 5, 4], [0, 1, 6, 5, 4], [0, 2, 5, 4], [0, 2, 5, 6, 1, 3, 4],
[0, 2, 6, 1, 3, 4], [0, 2, 6, 5, 4], [0, 4]]
```

אורך המסלול בגרף לא-משוקלל הוא מספר הקשתות במסלול. מרחק בין צמתים בגרף לא-משוקלל מוגדר כאורך המסלול הקצר ביותר בין שני הצמתים. נוכל לחשב זאת ב-networkx כך:

```
print(nx.shortest_path(G, 7, 9))
print(nx.shortest_path_length(G, 7, 9))
->
[7, 4, 0, 8, 9]
4
```

ניתן לחשב את המרחק מצומת אחד לכל אחד מן הצמתים האחרים. הדבר נעשה באמצעות Breadth-First (BFS) Search, שבוודאי מוכר לכם מקורסים קודמים. ניתן להריץ BFS על הרשת מצומת מסוים באמצעות networkx עם פונקציית bfs_tree. הפונקציה תחזיר עץ BFS שלמעשה פורס את הרשת.

```
T = nx.bfs_tree(G, 4)
nx.draw_networkx(T)
```



נשים לב שהוחזר עץ מכוון שמתוכו נוכל להסיק את המרחקים בין צומת לצומת (נזכיר שהרשת אינה ממושקלת).

במקרה הכללי, נוכל לקבל את המרחקים בין צומת לכל צומת אחר כך:

```
print(nx.shortest_path(G, 7))
->
{7: [7], 4: [7, 4], 0: [7, 4, 0], 3: [7, 4, 3], 5: [7, 4, 5], 1: [7, 4, 0, 1],
 2: [7, 4, 0, 2], 8: [7, 4, 0, 8], 6: [7, 4, 5, 6], 9: [7, 4, 0, 8, 9]}
```

בגרף משוקלל, אורך מסלול בין שני צמתים מוגדר כסכום ערכי הקשתות לאורך המסלול. חישוב המרחק בין שני צמתים בגרף ייעשה כך:

```
Gw = nx.Graph()
Gw.add_edge('A', 'B',weight = 6)
print(nx.shortest_path_length(Gw, 'A', weight='weight'))
->
{'A': 0, 'B': 6}
```

לצורך הפשטות, בהמשך הפרק, לא נתייחס למצב ממושקל. היינו, בגרפים ממושקלים, כל קשת היא בעלת משקל 1.

ניתן להגדיר מדד מרחק גלובלי של הגרף בכמה דרכים. המקובלת ביותר היא באמצעות ממוצע המרחק שבין כל צומת לכל צומת. נוכל לחשב אותו כך:

```
print(nx.average_shortest_path_length(G))
->
2.1333333333333333
```

קוטר של רשת הוא המרחק המקסימלי בין שני צמתים ברשת. נוכל לחשבו כך:

```
print(nx.diameter(G))
->
4
```

מדד חשוב נוסף הוא ה-eccentricity של צומת. המדד מבטא את המרחק המקסימלי של צומת מסוים מכל צומת אחר. נוכל לחשבו כך:

```
print(nx.eccentricity(G))
->
{0: 2, 1: 3, 2: 3, 4: 3, 8: 3, 3: 4, 6: 4, 5: 4, 7: 4, 9: 4}
```

הרדיוס של רשת הוא ה-eccentricity המינימלי שלה:

```
print(nx.radius(G))
->
2
```

כעת נרצה להבין מיהם הצמתים הרחוקים או הקרובים לכל האחרים. במקרה הראשון נגדיר את הפריפריה של הרשת כסט הצמתים שה-eccentricity שלהם שווה לקוטר הרשת. במקרה השני נגדיר את המרכז כסט הצמתים שה-eccentricity שלהם שווה לרדיוס הרשת. נבחן זאת:

```
print(nx.periphery(G))

print(nx.center(G))

->

[3, 6, 5, 7, 9]
[0]
```

ואומנם, הצמתים 3, 6, 5, 7, 9 הם פריפריאליים ביחס לצומת המרכזי - 0.

נשים לב שמדדי הפריפריה והמרכז רגישים לצמתים הנמצאים במקומות קיצון. למשל, יכול להיות צומת אחד המקושר ביחס לאחרים, אך הוא לא יהיה מרכזי מכיוון שישנו צומת אחר הרחוק ממנו במרחק הגדול מן הרדיוס.

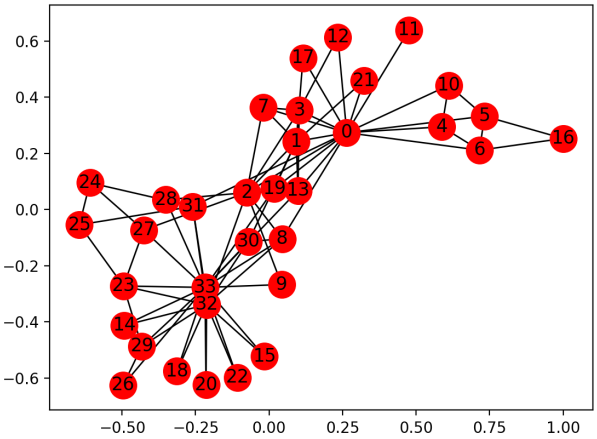
תרגיל 9.1

דוגמה מפורסמת אחת לרשת חברתית היא רשת של מועדון קרטה. תוכלו לטעון את הרשת ישירות מ-networkx כך:

G = nx.karate_club_graph()

בתרגיל זה ננתח את הגרף.

- טענו את הגרף והציגו אותו. על הגרף להיראות כך:



- מהו ממוצע המרחקים, הרדיוס והקוטר בגרף?
- מיהם הצמתים המצויים במרכז הגרף ומי מהם בפריפריה?
- ממבט ברשת, האם תוכלו למצוא צומת שהוא אומנם מרכזי מאוד, אך אינו נמצא ברשימת הצמתים המרכזיים? מדוע?

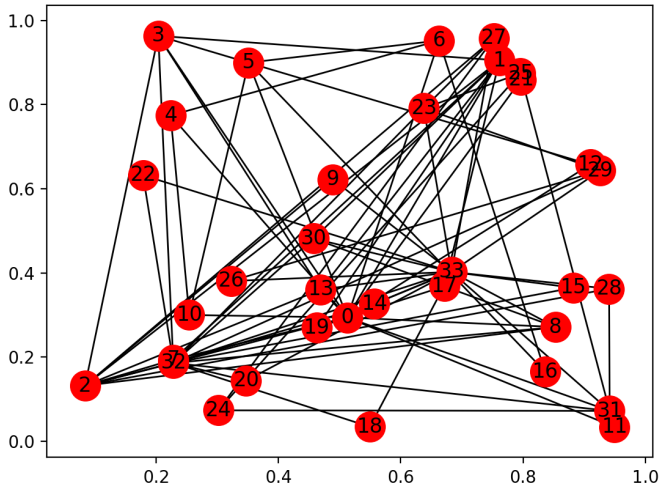
9.6. ויזואליזציה של רשתות עם NetworkX

לצורך הדוגמה, נעבוד עם דוגמת מועדון הקרטה שהוצגה קודם לכן. נטען את מסד הנתונים ונציג את הרשת.

```
G = nx.karate_club_graph()
```

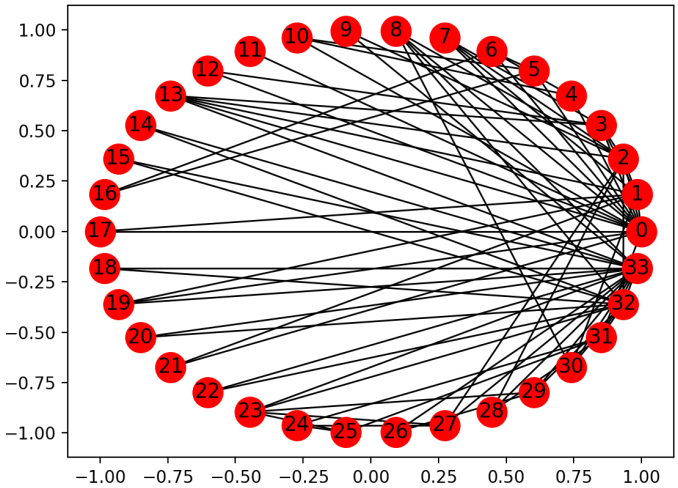
האופן שבו מחליטה networkx היכן למקם את הצמתים הוא באמצעות מודל שנקרא spring layout. במודל זה, הצמתים מסודרים מתוך מטרה למזער את מספר נקודות החיתוך של הקשתות. נוכל לבחור layout אחר בהתאם לאופי המידע. נוכל למשל להשתמש ב-layout אקראי באופן הזה:

```
nx.draw_networkx(G, nx.random_layout(G))
```



אופן ציור פופולרי וחשוב אחר הוא ה-circular layout. נשתמש בו כך:

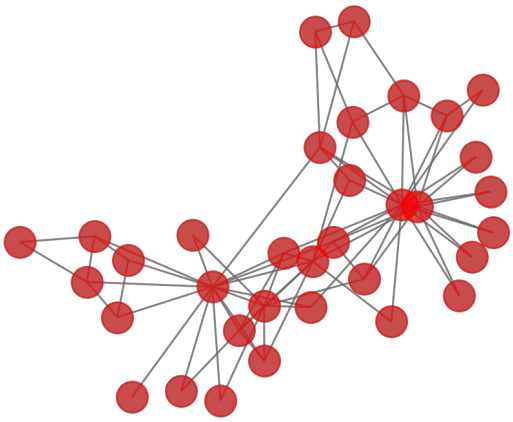
```
nx.draw_networkx(G, nx.circular_layout(G))
```



נוכל לשנות מאפיינים שונים ברשת. למשל, נמחק את הצירים, נצייר את הצמתים שקופים מעט (alpha), נהפוך את הקשתות לאפורות ולא נדפיס תוויות.

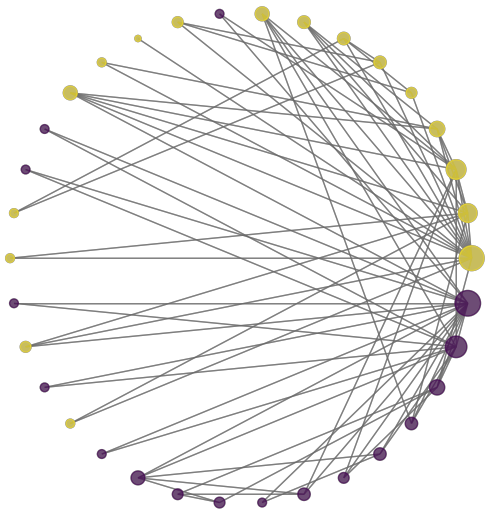
```
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(5, 4))
nx.draw_networkx(G, alpha=0.7, with_labels=False, edge_color='.4')
plt.axis('off')
plt.tight_layout()
```



כעת נחלץ מאפיינים חשובים לצורך ויזואליזציה. ראשית נוכל, באמצעות פונקציית degree, לחלץ את דרגת הצומת. דרגה של צומת היא מספר הצמתים המחוברים אליו. ככל שדרגתו של הצומת גבוהה יותר, כך ניתן לטעון שהוא "חשוב" יותר. נוכל לקבוע את גודל הצומת באיור על פי הדרגה שלו. מעבר לכך, נראה שבגרף קבוצת הקרטה לכל node ישנו attribute בשם club. ערכו של ה-attribute הוא Mr. Hi או Officer. נרצה לצבוע את הצומת בהתאם ל-attribute שלו. לצורך כך נייצר שני מערכים - מערך צבע ומערך גודל, ונשתמש בהם לוויזואליזציה.

```
node_size = [G.degree(v) * 20 for v in G]
node_color = [1 if nx.get_node_attributes(G, 'club')[v] == 'Mr. Hi' else 0.5 for v in G]
plt.figure(figsize=(6, 6))
nx.draw_networkx(G, nx.circular_layout(G), alpha=0.7, with_labels=False, edge_color='.
4', node_size = node_size, node_color = node_color)
plt.axis('off')
plt.tight_layout()
```



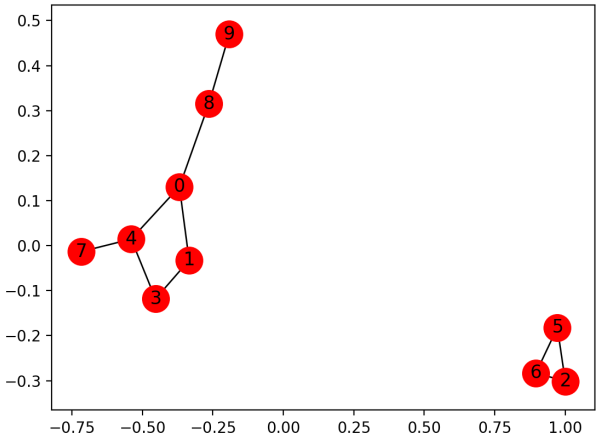
9.7. מדדי קישוריות רשתיים

ראשית, ניתן לבדוק אם רשת היא קשירה או לא (אם בעבור כל זוג צמתים ישנו מסלול המאפשר להגיע מהאחד לאחר) באמצעות פונקציית is_connected:

```
G = nx.read_adjlist('adjacency_list.txt', nodetype=int)
nx.is_connected(G)
->
True
```

ניתן להגיע למצב שבו יש לנו גרף עם שתי אוכלוסיות זרות, אם ננתק קשרים אסטרטגיים. למשל:

```
G.remove_edges_from([(1, 6), (0, 2), (5, 4)])
nx.is_connected(G)
nx.draw_networkx(G)
->
False
```



נגדיר רכיב קשירות (connected component) כתת-גרף המקיים שתי תכונות: תת-הגרף מקושר ואין קשתות היוצאות ממנו אל צמתים שאינם נמצאים בתת-הגרף. שני תתי-הגרפים באיור לעיל ממחישים שני רכיבי קשירות של הגרף. נוכל לחשב את מספר הישויות המחוברות ברשת כמו גם את זהותן כך:

```
print(nx.number_connected_components(G))
print(sorted(nx.connected_components(G)))
->
2
[{0, 1, 3, 4, 7, 8, 9}, {2, 5, 6}]
```

כדי לבדוק לאיזה רכיב קשירות משתייך צומת נתון, נכתוב:

```
print(nx.node_connected_component(G, 0))
->
```

```
{0, 1, 3, 4, 7, 8, 9}
```

נאמר כי רשת כיוונית היא קשירה חזק (strongly connected) אם קיים מסלול מכל צומת לכל צומת. נאמר כי הרשת היא קשירה חלש (weakly connected) אם הרשת הלא-כיוונית המושרית על ידיה היא מקושרת. ניתן לבדוק אם הרשת קשירה חזק או חלש באמצעות הפונקציות is_weakly_connected ו-is_strongly_connected. בהתאם, ניתן להגדיר strongly connected components ו-weakly connected components. בדקו זאת.

9.8. עמידות (robustness) של רשת

"עמידות" של רשת מתייחסת לתכונה מוגדרת. רשת היא עמידה לתכונה מסוימת אם התכונה נשמרת גם אם קשתות או צמתים "נופלים". אחת התכונות שנרצה לבדוק עמידות בעבורה היא קשירות הרשת. למשל, ברשת תחבורה נרצה להבטיח שגם אם יש תקלה בכביש מסוים, עדיין ניתן יהיה להגיע ממקום למקום.

נביט שוב ברשת שתוארה למעלה. נבחן את העמידות שלה לקשירות בעת הסרת צמתים. הפונקציה node_connectivity תחזיר את מספר הצמתים הנמוך ביותר שאם יוסרו תיפגע הקשירות של הרשת, והפונקציה minimum_node_cut תחזיר את הצמתים עצמם. למשל:

```
G = nx.read_adjlist('adjacency_list.txt', nodetype=int)
print(nx.is_connected(G))
print(nx.node_connectivity(G))
print(nx.minimum_node_cut(G))
```

ואומנם, אילו הייתה זו רשת תחבורה, היא לא הייתה עמידה במיוחד לתקלות.

נוכל לבצע את אותו התהליך בעבור הקשתות:

```
print(nx.edge_connectivity(G))
print(nx.minimum_edge_cut(G))
->
1
{(8, 9)}
```

ככל שמספר הצמתים שהסרתם תפגע בקשירות גבוה יותר, כך הרשת עמידה יותר להסרת צמתים לשימור תכונת הקשירות. כך גם לגבי הסרת קשתות.

באופן דומה, ניתן לבדוק את העמידות של קשירות בין שני צמתים מסוימים להסרה של צמתים או של קשתות:

```
print(nx.node_connectivity(G, 0, 5))
print(nx.minimum_node_cut(G, 0, 5))
print(nx.edge_connectivity(G, 0, 5))
print(nx.minimum_edge_cut(G, 0, 5))
->
3
{2, 4, 6}
3
{(4, 5), (2, 5), (6, 5)}
```

```
—
print(nx.node_connectivity(G, 0, 9))
print(nx.edge_connectivity(G, 0, 9))
->
1
1
```

ואומנם, ישנם צמדי צמתים העמידים יותר מאחרים.

9.9. השפעה ומרכזיות ברשת

ישנן מספר דרכים לאפיין צמתים מרכזיים ברשת. הראשונה שבהן מבוססת על דרגת הצומת. ככל שדרגת הצומת ביחס למספר הצמתים ברשת גבוהה יותר, נוכל לטעון שהצומת מרכזי יותר. נבחן זאת על דוגמת מועדון הקרטה:

```
G = nx.karate_club_graph()
G = nx.convert_node_labels_to_integers(G, first_label=1)
degCent = nx.degree_centrality(G)
->
{0: 0.48484848484848486, 1: 0.2727272727272727, 2: 0.30303030303030304, 3:
0.18181818181818182, 4: 0.09090909090909091, 5: 0.12121212121212122,…}
```

ברשת מכוונת, מדד זה יפוצל לשני מדדים: in-degree centrality, המתאר את מרכזיות הצומת על סמך מספר הכניסות אליו, ו-out-degree centrality, המתאר את מרכזיות הצומת על סמך מספר הקשתות היוצאות ממנו. נוכל לחשב אותם באמצעות הפונקציות in_degree_centrality ו-out_degree_centrality. נסו זאת.

המדד השני למרכזיותה של צומת מכונה מדד הקרבה (closeness centrality) והוא מניח שבצמתים חשובים, ממוצע המרחקים שלהם לצמתים אחרים יהיה נמוך ביחס לאחרים. נחשב את ה-closeness centrality ברשת:

```
closeCent = nx.closeness_centrality(G)
->
{1: 0.5689655172413793, 2: 0.4852941176470588, 3: 0.559322033898305, 4:
0.4647887323943662, 5: 0.3793103448275862, …}
```

כאשר הרשת אינה קשירה, הסיפור הופך להיות מעט מורכב יותר. ואומנם, כיצד נמדוד closeness בעבור צומת שלא ניתן להגיע ממנו אל הצמתים האחרים ברשת? דרך אחת היא לחשב את המדד בעבור הצמתים הנגישים לצומת בלבד. זו אפשרות בעייתית - צומת שניתן להגיע ממנו לצומת אחר בקשת ישירה יהיה בעל closness centrality של 1 - הערך הגבוה ביותר האפשרי. האפשרות השנייה היא לנרמל את המספר ביחס למספר הצמתים הכללי. ניתן לחשב זאת באמצעות הפונקציה closeness_centrality עם attribute המציין נורמליזציה: normalized = False / True. נסו זאת.

דרך נוספת לתיאור מרכזיות של צומת היא באמצעות ההנחה שצמתים חשובים מחברים, או מקשרים, צמתים אחרים. ערך זה נקרא betweenness centrality. בקצרה, אלה אותם צמתים שמספר המסלולים הקצר ביותר בין כל צומת לכל צומת עובר דרכם. חישוב מדד זה עלול לארוך זמן רב בעבור רשתות גדולות. ניתן לחשב את המדד כך:

```
betweenCent = nx.betweenness_centrality(G, normalized=True)
->
```

```
{1: 0.43763528138528146, 2: 0.053936688311688304, 3: 0.14365680615680618, 4:
0.011909271284271283, 5: 0.0006313131313131313,…}
```

כדי להגביל את זמן החישוב, ניתן להזין לפונקציה פרמטר k שגודלו יתאר כמה צמתים ישמשו לחישוב המדד בעבור כל צומת. קראו עליו בתיעוד הרשמי של הפונקציה ונסו לשנות אותו.

ניתן לחשב betweenness centrality בין שתי תת-רשתות. זאת ניתן לעשות באמצעות הפונקציה betweenness_centrality_subset. קראו על הפונקציה בתיעוד של networkx ונסו אותה. כמו כן ניתן להשתמש במדד זה על קשתות במקום על צמתים. תוכלו לחשב זאת עם פונקציית edge_betweenness_centrality באופן דומה מאוד למה שנעשה למעלה.

דרך נוספת וחשובה במיוחד למדידת חשיבותו של צומת מכונה PageRank, והיא פותחה בבסיסה בחברת Google. מנגנון החיפוש של Google עושה שימוש, בין היתר, בשיטה זו כדי לדרג תוצאות חיפוש. בבסיס השיטה מונחת ההנחה שלצמתים חשובים הרבה כניסות מצמתים חשובים אחרים. השיטה רלוונטית במיוחד לרשתות מכוונות. נשים לב שמדובר בהגדרה מעגלית - ה-PageRank שלי תלוי ב-PageRank של אלו המחוברים אלי. לפיכך, נקבע את ערך ה-PageRank ההתחלתי שלי ל-1 חלקי מספר הצמתים ברשת. לאחר מכן, באופן איטרטיבי נעדכן את המדד בעבור כל צומת כך שערכו יהיה שווה לסכום ה-PageRanks של הצמתים המחוברים אליו. זאת כאשר התרומה של כל צומת לזה המחובר אליו היא יחסית למספר הצמתים הכללי שעליו הוא מצביע. כלומר, ככל שדרגת היציאה של צומת נמוכה יותר, התרומה שלו לצמתים המחוברים אליו משמעותית יותר. תוכלו לקרוא עוד על השיטה בלינק: https://en.wikipedia.org/wiki/PageRank/. נוכל לחשב את המדד ברשת שלנו כך:

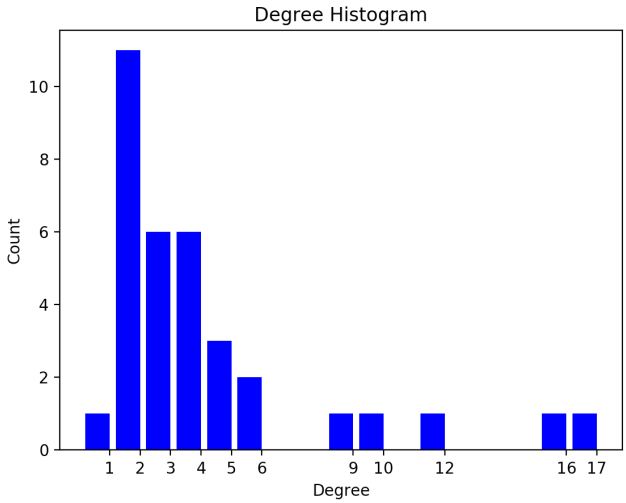
```
pageRankCent = nx.pagerank(G)

->

{1: 0.09700181758983709, 2: 0.05287839103742701, 3: 0.057078423047636745, 4:
0.03586064322306479, 5: 0.021979406974834498,…}
```

מטבע הדברים, ישנם עוד פרטים רבים בשיטה, ובהתאם לכך מספר הפרמטרים האפשרי שניתן להשתמש בהם הוא רב. תוכלו לקרוא על כך עוד בתיעוד הרשמי של networkX.

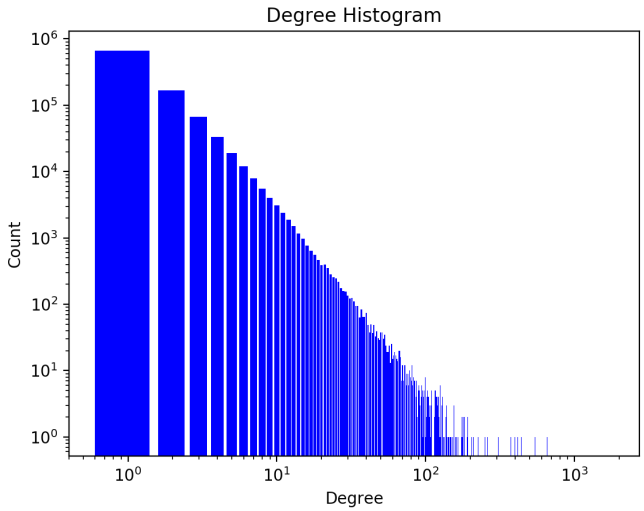
תרגיל 9.2

לעיתים, מעבר לדרגתו של כל צומת, נתעניין בפיזור (distribution) של הדרגות ברשת. בתרגיל זה נחשב את פיזור הדרגות ברשת של מועדון הקרטה.

- טענו את רשת מועדון הקרטה כפי שנעשה למעלה.
- חלצו את רשימת הדרגות של הצמתים מן הרשת ומיינו אותם במערך.
- ספרו את מספר הפעמים שכל דרגה נמצאת במערך. דרך יעילה אחת לעשות זאת היא באמצעות ספריית collections של Python. ייבאו אותה והשתמשו בפונקציית Counter כדי לבצע את הספירה.
- ציירו היסטוגרמה של התוצאות באמצעות plt.bar. על התוצאה להיראות כך:



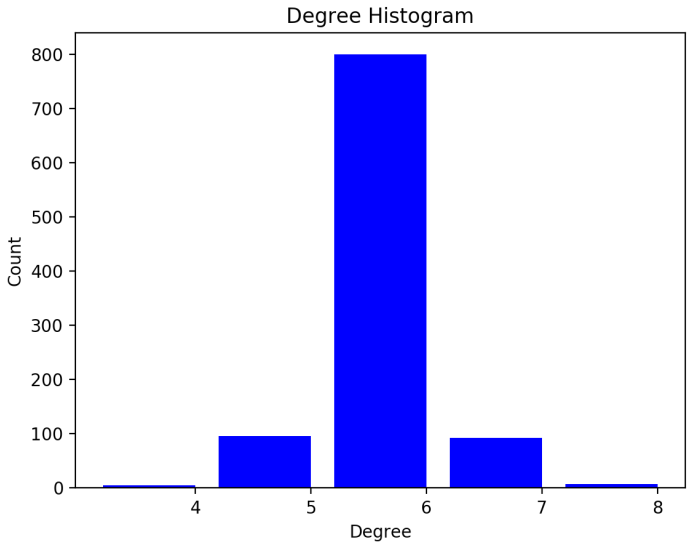
- ברשתות רחבות-היקף רבות, נראה שישנם מספר רב של צמתים שהדרגה שלהם נמוכה ומספר נמוך של צמתים שהדרגה שלהם גבוהה במיוחד. זאת באופן המתיישב עם עקומה לוגריתמית, ועל כן מכונה power-law phenomena. בהתאם לכך, נוכל להגדיר מודל רשתי באופן כזה שפיזור הדרגות על פני הרשת יעקוב אחר ה-power law. אחד מן המודלים הללו הוא ה-preferential attachment model. המודל מבטא את המוטו "העשיר הופך לעשיר יותר" (rich get richer). ברשת חברתית, אלו בעלי המספר הרב יותר של חברים ככל הנראה ירכשו מספר רב יותר של חברים בעתיד, ביחס לאלו שמספר החברים ההתחלתי שלהם נמוך יותר. ניתן לייצר רשת כזאת ב-networkX באמצעות פונקציית: nx.barabasi_albert_graph. קראו על הפונקציה בתיעוד הרשמי של networkX והדפיסו את ההיסטוגרמה של הרשת שתיצרו (הגדירו מיליון צמתים). שימו לב לקבוע את הצירים בגרף זה כלוגריתמיים. על הגרף להיראות כך:



תרגיל 9.3

תופעה מעניינת היא תופעת "העולם הקטן" (Small World Model). ברשת פייסבוק, למשל, נמצא שב-2008 ניתן היה להגיע מכל ישות ברשת לאחרת בממוצע של 5.28 קשתות. ב-2011 ירד הערך הזה ל-4.74. זאת חרף המספר הרב כל כך של צמתים ברשת. כמו כן, הרשתות הללו נוטות להיות בעלות מדדי clustering גבוהים (שעליהם דיברנו קודם לכן). זאת, אגב, בניגוד לרשתות שנייצר באופן רנדומלי. במילים אחרות, תכונות "העולם הקטן" נובעות מן הרשת החברתית.

- אחת מן הפונקציות שבאמצעותן ניתן לייצר "עולם קטן" היא פונקציית watts_strogatz_graph. קראו על הפונקציה בתיעוד של networkX. צרו רשת בעלת 1,000 צמתים והדפיסו את היסטוגרמת ה-degrees שלה. התוצאה תיראה כך:



- מה תוכלו ללמוד מן ההיסטוגרמה?

9.10. ניבוי שינויים עתידיים ברשת

רשת אינה מבטאת בדרך כלל מערך יחסים סטטי. בדרך כלל, מערך הקשרים ברשת משתנה, או מתפתח. יישום חשוב במיוחד לרשתות יכול להיות ניבוי אופי התפתחותה של הרשת. למשל, מנהלי רשת חברתית ירצו להיות מסוגלים להמליץ לחברים ברשת על חברויות חדשות, ומנהלי רשת לקוחות-סרטים ירצו להיות מסוגלים להמליץ ללקוחותיהם על סרטים רלוונטיים. תהליך זה נקרא link prediction.

מדד אחד יכול להתבסס על מספר חברים משותפים. כלומר, ההסתברות שצומת אחד יחובר בעתיד לאחר תלויה במספר החברים המשותפים שלהם. נבחן זאת על דוגמת רשת מועדון הקרטה. שימו לב לשימוש בפונקציית common_neighbors לצורך בחינת מספר ה"חברים המשותפים", כמו גם לפונקציית non_edges שבה נשתמש כדי לעבור על צמדי הצמתים שעוד אינם חברים.

```
G = nx.karate_club_graph()

c_m = [(e[0], e[1], len(list(nx.common_neighbors(G, e[0], e[1]))))

       for e in nx.non_edges(G)]

c_m

->

[(0, 32, 3), (0, 33, 4), (0, 9, 1), (0, 14, 0), (0, 15, 0), (0, 16, 2), … ]
```

נמיין את הרשימה על פי מספר החברים המשותפים, וכך נוכל להצביע על החברים שבסבירות גבוהה יחסית לאחרים יהפכו לחברים בעתיד.

```
import operator

sorted(c_m, key=operator.itemgetter(2), reverse=True)

->
[(2, 33, 6), (0, 33, 4), (7, 13, 4), (0, 32, 3), (1, 8, 3), (1, 33, 3), … ]
```

מדד אחר מכונה מקדם Jaccard, ובמסגרתו מספר החברים המשותפים מנורמל לפי איחוד כלל החברים של שני הצמתים. ניתן לחשב את המדד באמצעות פונקציית jaccard_coefficient באופן דומה מאוד לנעשה לעיל. נסו זאת.

מדד נוסף מכונה Resource allocation. המדד מבוסס על כך שאם צומת מחובר לצומת אחר דרך צומת שלישי שיש לו דרגה גבוהה במיוחד, הרי שהקשר בין הראשון לשני נמוך. היינו, ככל שהחבר המחבר בין השניים בעל דרגה נמוכה יותר, הקשר בין השניים משמעותי יותר (הדבר אינטואיטיבי - למשל, בין שני חברים המחוברים דרך אדם שלישי מפורסם, שיש לו מאות אלפי חברים, לא סביר שייווצר קשר). ככל שצמתים מחוברים יחדיו דרך חברים שדרגתם נמוכה, ההסתברות שלהם לחברות גבוהה יותר. ניתן לחשב את המדד כך:

```
L = list(nx.resource_allocation_index(G))

->

[(0, 32, 0.4666666666666667), (0, 33, 0.9), (0, 9, 0.1), (0, 14, 0), (0, 15, 0),… ]
```

מדד אחר הוא ה-preferential attachment, המניח שצמתים יחוברו יחד אם לשניהם דרגה גבוהה. לדוגמה, ישנה סבירות גבוהה ששני שחקנים מפורסמים יהיו חברים ברשת חברתית. המדד עושה שימוש במכפלת הדרגות של צמתים, וניתן להשתמש בו באמצעות פונקציית preferential_attachment באופן דומה מאוד למה שנעשה למעלה. נסו זאת.

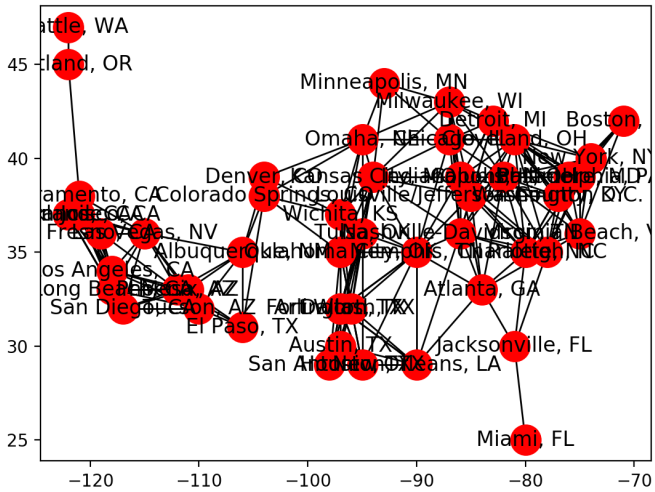
9.11. חילוץ מאפיינים מרשת

בפרקים הקודמים נתקלנו במספר רב של מדדים. שלל המדדים האלה מבטאים הלכה למעשה features שניתן לעשות בהם שימוש במסגרת למידת מכונה, כפי שכבר עשינו פעמים רבות במהלך הקורס. נראה לדוגמה את האופן שבו ניתן להתייחס למדדים הללו כ-features. לצורך הדוגמה, נעשה שימוש ברשת המבטאת ערים גדולות בארצות הברית, את גודל האוכלוסייה בהן, ואת הקישוריות ביניהן. נטען את הרשת כך (תוכלו להוריד את הקובץ מאתר הקורס):

```
G = nx.read_gpickle('US_cities.gp')
```

נציג את הגרף, תוך הסתמכות על מאפיין ה-location של הערים המבוטאות בו.

```
G = nx.read_gpickle('US_cities.gp')
nx.draw_networkx(G, nx.get_node_attributes(G, 'location'))
```



נוכל לחלץ מן הרשת את המאפיינים שלה אל תוך data frame שבספריית pandas. למשל:

```
import pandas as pd
df = pd.DataFrame(index=G.nodes())
df['location']   = pd.Series(nx.get_node_attributes(G, 'location'))
df['population'] = pd.Series(nx.get_node_attributes(G, 'population'))
df.head()
```

	location	population
El Paso, TX	(-106, 31)	674433
Long Beach, CA	(-118, 33)	469428
Dallas, TX	(-96, 32)	1257676
Oakland, CA	(-122, 37)	406253
Albuquerque, NM	(-106, 35)	556495

כעת ניתן להרחיב את ה-data frame עם מדדים שונים על אודות כל צומת. למשל, מקדם ה-clustering ודרגת הצומת:

```
df['clustering'] = pd.Series(nx.clustering(G))
df['degree']     = pd.Series(dict(G.degree()))
df.head()
```

	location	population	clustering	degree
El Paso, TX	(-106, 31)	674433	0.700000	5
Long Beach, CA	(-118, 33)	469428	0.745455	11
Dallas, TX	(-96, 32)	1257676	0.763636	11
Oakland, CA	(-122, 37)	406253	1.000000	8
Albuquerque, NM	(-106, 35)	556495	0.523810	7

באופן דומה, נוכל לבנות טבלת features בעבור הקשתות. נוכל להתייחס למשל למשקל הקשתות כ-features באופן הזה:

```
df = pd.DataFrame(index=G.edges())
df['weight'] = pd.Series(nx.get_edge_attributes(G, 'weight'))
df.head()
```

	weight
(El Paso, TX, Albuquerque, NM)	367.885844
(El Paso, TX, Mesa, AZ)	536.256660
(El Paso, TX, Tucson, AZ)	425.413867
(El Paso, TX, Phoenix, AZ)	558.783570
(El Paso, TX, Colorado Springs, CO)	797.751712

כמו בעבור ה-nodes, נוכל להוסיף features המבטאים מדדים שונים, דוגמת ה-preferential attachment שתואר בפרק הקודם:

```
df['preferential attachment']= [i[2] for i in nx.preferential_attachment(G, df.index)]
df.head()
```

	weight	preferential attachment
(El Paso, TX, Albuquerque, NM)	367.885844	35
(El Paso, TX, Mesa, AZ)	536.256660	40
(El Paso, TX, Tucson, AZ)	425.413867	40
(El Paso, TX, Phoenix, AZ)	558.783570	45
(El Paso, TX, Colorado Springs, CO)	797.751712	30

פתרונות לתרגילים

תרגיל 9.1

```
G = nx.karate_club_graph()
nx.draw_networkx(G)
print(nx.diameter(G))
print(nx.radius(G))
print(nx.center(G))
print(nx.periphery(G))
```

תרגיל 9.2

```
import collections
degree_sequence = sorted([d for n, d in G.degree()], reverse=True)
degreeCount = collections.Counter(degree_sequence)
deg, cnt = zip(*degreeCount.items())
plt.bar(deg, cnt, width=0.80, color='b')
plt.title("Degree Histogram")
plt.ylabel("Count")
plt.xlabel("Degree")
ax.set_xticks([d + 0.4 for d in deg])
ax.set_xticklabels(deg)
plt.show()

G = nx.barabasi_albert_graph(1000000, 1)
degree_sequence = sorted([d for n, d in G.degree()], reverse=True)
degreeCount = collections.Counter(degree_sequence)
deg, cnt = zip(*degreeCount.items())
plt.bar(deg, cnt, width=0.80, color='b')
plt.title("Degree Histogram")
plt.ylabel("Count")
plt.xlabel("Degree")
plt.xscale('log')
plt.yscale‘log')
plt.show()
```

תרגיל 9.3

```
G = nx.watts_strogatz_graph(1000, 6, 0.04)
degree_sequence = sorted([d for n, d in G.degree()], reverse=True)
degreeCount = collections.Counter(degree_sequence)
deg, cnt = zip(*degreeCount.items())

fig, ax = plt.subplots()
plt.bar(deg, cnt, width=0.80, color='b')
plt.title("Degree Histogram")
plt.ylabel("Count")
plt.xlabel("Degree")
ax.set_xticks([d + 0.4 for d in deg])
ax.set_xticklabels(deg)
```