Федеральное государственное бюджетное образовательное учреждение высшего образования «Санкт-Петербургский государственный университет»

Факультет прикладной математики-процессов управления

# Задание по курсу “Теория графов и ее приложения”: Исследование социальных графов

Выполнила
Цеханович Ирина Сергеевна
3 курс, группа 16.Б11-пу

Санкт-Петербург
2019 г.

**Содержание**

## Введение

Работа выполнена на графе друзей, полученном из социальной сети ВКонтакте. Граф включает 240 узлов и 4434 рёбер.

Для работы с графом использовались следующие программы:

- Gephi 0.9.2 для экспорта графа в виде матрицы смежности и списка смежности,
- PyCharm для программирования на языке Python,
- Microsoft Excel для построения гистограммы (задание 2).

Для удобства работы с графом использовалась такая структура, как словарь множеств. Словари в Python - неупорядоченные коллекции произвольных объектов с доступом по ключу. Словарь был получен из списка смежности графа.

## Задание 1. Нахождение компонент слабой и сильной связности

Для нахождения компонент слабой связности используется неориентированный граф, список смежности преобразован в словарь множеств.

```
listDict = dict()
with open('friends_list_unor.csv', newline='') as csvfile:
    listfile = csv.reader(csvfile, delimiter=';')
    for row in listfile:
        tempSet = set()
        for i, element in enumerate(row):
            if i != 0:
                tempSet.add(element)
        listDict[row[0]] = tempSet
```

Для решения была использована следующая реализация поиска в глубину:

```
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    for next in graph[start] - visited:
        dfs(graph, next, visited)
    return visited
```

Алгоритм решения заключается в следующем: создается множество непосещенных вершин, равное множеству всех вершин графа; из этого множества выбирается произвольная вершина, из которой запускается поиск в глубину. Каждая вершина удаляется из множества непосещенных вершин, как только её обошли при поиске в глубину. Алгоритм прекращает свою работу, когда множество непосещенных вершин оказывается пустым.

Для нахождения компонент сильной связности используется алгоритм Косарайю:

1 шаг: Запускается поиск в глубину с фиксированием времени выхода (вершины хранятся в стеке).

2 шаг: Транспонирование графа.

3 шаг: Запускается поиск в глубину на транспонированном графе в порядке, определенном в стеке на шаге 1.

Граф для удобства задается отдельным классом, в котором прописывается метод инициализации, метод транспонирования графа и сам метод поиска компонент сильной связности.

Инициализация:

```
class Graph:
    def __init__(self,vertices):
        self.V= vertices
        self.graph = defaultdict(set)
```

Транспонирование:

```
    def getTranspose(self):
        g = Graph(self.V)
        for i in self.graph:
            for j in self.graph[i]:
                g.addEdge(j,i)
        return g
```

Поиск компонент сильной связности:

```
    def printSCCs(self):
        componentVertices = set()
        numOfComponents = 0

        stack = []
        # Mark all the vertices as not visited (For first DFS)
        visited = dict.fromkeys(self.graph, False)
        # Fill vertices in stack according to their finishing
        # times
        for i in self.graph:
            if visited[i]==False:
                self.fillOrder(i, visited, stack)

        # Create a reversed graph
        gr = self.getTranspose()

        # Mark all the vertices as not visited (For second DFS)
        # visited =[False]*(self.V)
        visited = dict.fromkeys(self.graph, False)

        # Now process all vertices in order defined by Stack
        while stack:
            i = stack.pop()
            if visited[i]==False:
                gr.DFSUtil(i, visited, componentVertices)
                print(componentVertices, " --- component size: ", len(componentVertices))
                print("")
                numOfComponents += 1
                componentVertices.clear()
        print("Number of strongly connected components: ", numOfComponents)
```

Полученные результаты:

Число компонент слабой связности: 10, наибольшая компонента слабой связности состоит из 228 узлов, что составляет 0,95 от всех узлов графа.

```
Component size:  228
Component size:  1
Component size:  3
Component size:  1
Component size:  2
Component size:  1
Component size:  1
Component size:  1
Component size:  1
Component size:  1
Largest component:  228 ;  0.95
```

Число компонент сильной связности: 13

```
Component size:  1
Component size:  1
Component size:  1
Component size:  1
Component size:  1
Component size:  2
Component size:  1
Component size:  1
Component size:  1
Component size:  1
Component size:  227
Component size:  1
Component size:  1
Number of strongly connected components:  13
```

## Задание 2. Построение гистограммы плотности вероятности распределения степеней вершин, вычисление средней степени вершины.

Далее под графом понимаем наибольшую компоненту связности графа без учета ориентации ребер. Граф был экспортирован в отдельный файл exported.csv

```
import csv
with open('exported.csv', 'w', newline='') as csvfile:
    csvwriter = csv.writer(csvfile, delimiter=';', quotechar='|', quoting=csv.QUOTE_MINIMAL)
    for el in EXPORT:
        row = [el]
        row.extend(listDict[el])
        csvwriter.writerow(row)
        row = []
```

Граф рассматривается как словарь множеств (как в задании 1). Для построения гистограммы плотности вероятности распределения вершин создается массив степеней вершин. Также сразу считается сумма степеней вершин для нахождения средней степени вершины.

```
sumOfDegrees = 0
degrees = []
for el in listDict:
    degree = len(listDict[el])
    degrees.append(degree)
    sumOfDegrees += degree

print('Average vertex degree: ', sumOfDegrees/228)
```

Далее для каждой имеющейся степени подсчитывается количество соответствующих вершин.

```
degreeCount = dict.fromkeys(set(degrees), 0)
for d in degrees:
    degreeCount[d] += 1
```

Полученные результаты:

Средняя степень вершины: 19.535087719298247

Результаты вычислений хранятся в файле histogram.csv: матрица степеней вершин (1 ряд), количества вершин соответствующей степени (2 ряд) и вероятностей степеней вершин (3 ряд)

Гистограмма плотности вероятности распределения степеней вершин

## Задание 2. Диаметр, радиус графа, центральные и периферийные вершины, средняя длина пути в графе.

Для нахождения средней длины путей в графе, а также радиуса и диаметра графа прежде всего для каждой вершины создаётся список кратчайших расстояний до каждой вершины. Для этого используется поиск в ширину, который запускается из каждой вершины графа.

```
def bfs(graph, start):
    visited, q1,q2, d, paths = set(), [start],[],0, []
    while q1 or q2:
        if (d%2 == 0):
            while q1:
                vertex = q1.pop(0)
                if vertex not in visited:
                    visited.add(vertex)
                    q2.extend(graph[vertex] - visited)
                    paths.append(d)
            d += 1


        else:
            while q2:
                vertex = q2.pop(0)
                if vertex not in visited:
                    visited.add(vertex)
                    q1.extend(graph[vertex] - visited)
                    paths.append(d)
            d += 1
    return paths
```

Для каждой вершины находится эксцентриситет, после чего определяются радиус и диаметр графа. Параллельно с этим вычисляем среднюю длину пути в графе - для каждой вершины находится среднее значение, после чего сумма этих значений делится на количество вершин.

```
radius, diameter, mylist ,avg = 228,0,[], 0
for i, el in enumerate(listDict):
    res = bfs(listDict,el)
    avg += sum(res)/(len(res)-1)
    mylist.append([el, max(res)])
    if max(res) > diameter:
        diameter = max(res)
    if max(res) < radius:
        radius = max(res)
avg /= 228
```

Для определения центральных и периферийных вершин в массиве эксцентриситетов выполняется поиск значений, равных радиусу, и значений, равных диаметру графа.

```
central, peripheral = [],[]
for row in mylist:
    if row[1] == radius:
        central.append(row[0])
    elif row[1] == diameter:
        peripheral.append(row[0])
```

Полученные результаты:

Радиус графа: 5

Диаметр графа: 8

Средняя длина пути в графе: 3.1683669526238494

Центральные вершины: ['40693989', '66606777', '53389453', '65561782', '138062764', '92302263', '69011954', '90215761', '175708881', '163817106', '145311702', '118591534', '30189819', '54226354', '101779154', '194561462', '367798724', '53433574', '64668986', '105897517', '143467744', '101667591', '109174409', '64387636', '180548850', '28202231', '134095207', '64522368', '42366752', '135174451', '208957087', '79958332', '162297277', '161193027', '209066411', '39244153', '13023176', '98256139', '38128854', '70536455', '381977137', '26837438', '69828080', '36704197', '39555462', '193329243', '93718578', '94822918', '84725288', '239292372', '133250453', '13417151', '96392913', '235897006', '77868213', '105363091', '137071050', '113697460', '62722639', '22454482', '36880180', '232831985', '80429607', '59477024', '66202027']

Периферийные вершины: ['134181320', '11321711', '145549362', '55512780']

## Задание 3. Меры сходства узлов графа.

Для вычисления меры сходства узлов графа граф рассматривался как словарь множеств (как в задании 1). Для записи каждого результата создавалась матрица.

Были вычислены меры сходства на основе следующих метрик:

1. Common Neighbors (число общих соседей): $s(x, y) = |N(x) \cap N(y)|$

```
n = len(listDict)+1
CommonNeighbors = [[0] * n for i in range(n)]
for i,el in enumerate(listDict):
    CommonNeighbors[0][i+1] = el
    CommonNeighbors[i+1][0] = el
for i,x in enumerate(listDict):
    for j,y in enumerate(listDict):
        CommonNeighbors[i+1][j+1] = len(listDict[x]&listDict[y])
```

2. Jaccard's Coefficient (мера Жаккара) $s(x, y) = \frac{|N(x) \cap N(y)|}{|N(x) \cup N(y)|}$

```
n = len(listDict)+1
JaccardsCoef = [[0] * n for i in range(n)]
for i,el in enumerate(listDict):
    JaccardsCoef[0][i+1] = el
    JaccardsCoef[i+1][0] = el
for i,x in enumerate(listDict):
    for j,y in enumerate(listDict):
        JaccardsCoef[i+1][j+1] = len(listDict[x]&listDict[y])/len(listDict[x]|listDict[y])
```

3. Adamic/Adar (Frequency-Weighted Common Neighbors)

$$A(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{\log |N(u)|}$$

```
n = len(listDict)+1
AdamicAdar = [[0] * n for i in range(n)]
for i,el in enumerate(listDict):
    AdamicAdar[0][i+1] = el
    AdamicAdar[i+1][0] = el
for i,x in enumerate(listDict):
    for j,y in enumerate(listDict):
        sum = 0
        for el in listDict[x]&listDict[y]:
            if(len(listDict[el])):
                sum += 1/np.log(len(listDict[el]))
        AdamicAdar[i + 1][j + 1] = sum
```

4. Preferential Attachment $s(x, y) = |N(x)| \times |N(y)|$

```
n = len(listDict)+1
PreferentialAttachment = [[0] * n for i in range(n)]
for i,el in enumerate(listDict):
    PreferentialAttachment[0][i+1] = el
    PreferentialAttachment[i+1][0] = el
for i,x in enumerate(listDict):
    for j,y in enumerate(listDict):
        PreferentialAttachment[i + 1][j + 1] = len(listDict[x])*len(listDict[y])
```

Полученные результаты: Результаты вычислений находятся в соответствующих файлах: CommonNeighbors.csv, JaccardsCoef.csv, AdamicAdar.csv, PreferentialAttachment.csv.

## Задание 4. Метрики центральности вершин графа

Для вершин графа рассчитаны нормализованные метрики центральности

- по степени (degree centrality):

  $$C_d = \frac{deg(v)}{|V|}$$

```
degrees = []
for el in listDict:
    degree =[el, len(listDict[el])]
    degrees.append(degree)
degreecentrality = []
for el in degrees:
   degreecentrality.append([el[0],el[1] / 227])
```

- по близости (closeness centrality):

  $C(x) = \frac{N-1}{\sum_y d(y,x)}$, где $d(y,x)$ - кратчайшее расстояние между вершинами x и y

```
closenesscentrality =[]
for i, el in enumerate(listDict):
    res = bfs(listDict,el)
    closenesscentrality.append([el,(227)/sum(res)])
```

- метрику центральности по собственному вектору (eigenvector centrality): результатом является собственный вектор, соответствующий наибольшему собственному числу матрицы смежности

```
matrix = np.genfromtxt('exported_matrix.csv', delimiter=';', dtype='int')
x = np.linalg.eig(matrix)[1][:,np.linalg.eig(matrix)[0].argmax(axis=0)].real
```

Полученные результаты: Результаты вычислений находятся в одном файле: Centrality.csv. Каждая метрика центральности записана в отдельном столбце. Визуализация результатов представлена в файлах degree_centrality.pdf, closeness_centrality.pdf, eigen_centrality.pdf.

## Приложения

К данному отчету приложены следующие файлы:

1. Ориентированный граф всех друзей в виде матрицы смежности: friends_matrix_or.csv
2. Ориентированный граф всех друзей в виде списка смежности: friends_list_or.csv
3. Неориентированный граф всех друзей в виде матрицы смежности: friends_matrix_unor.csv
4. Неориентированный граф всех друзей в виде списка смежности: friends_list_unor.csv
5. Визуализация графа всех друзей: allfriends.pdf
6. Наибольшая компонента слабой связности в виде списка смежности: exported.csv
7. Матрица степеней вершин (1 ряд), количества вершин соответствующей степени (2 ряд) и вероятностей степеней вершин (3 ряд) в файле histogram.csv
8. Результаты задания 3 в соответствующих файлах: CommonNeighbors.csv, JaccardsCoef.csv, AdamicAdar.csv, PreferentialAttachment.csv.
9. Результаты задания 4 в файле centrality.csv.
10. Визуализация результатов 4 здания в файлах degree_centrality.pdf, closeness_centrality.pdf, eigen_centrality.pdf