**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

 Факультет прикладной математики — процессов управления

**Проект**

по дисциплине «Теория конечных графов и ее приложения»

 на тему: «Анализ социального графа»

**Выполнила:** студентка 3-го курса направления

«Фундаментальные информатика и информационные технологии»

332 группы

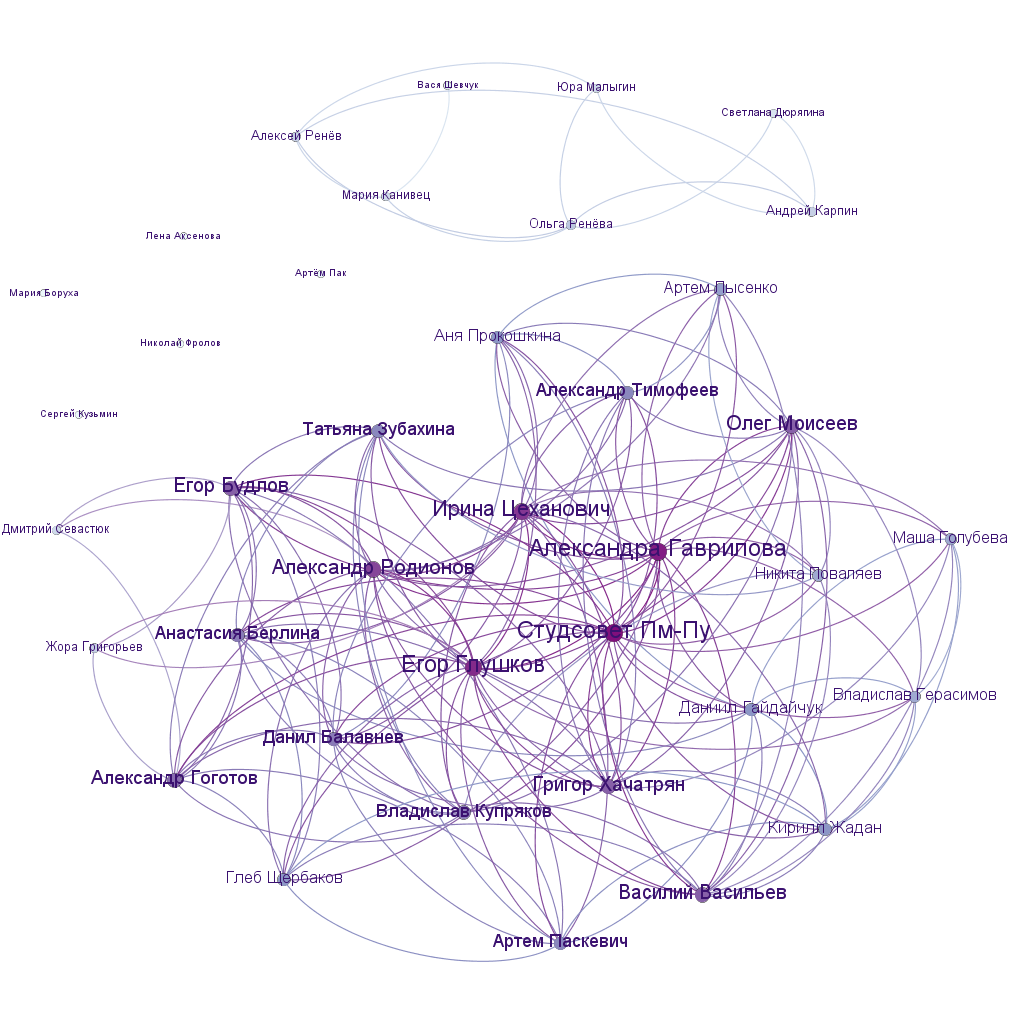
Байнова Екатерина Олеговна

**САНКТ-ПЕТЕРБУРГ**

**2019**

# Введение

В данной работе рассматривается анализ социального графа и его наибольшей компоненты связности без учета ориентации ребер.



Граф был получен при помощи приложения [Export friends graph](https://vk.com/app3861133).

Целью работы является визуализация исходного графа с помощью проведенного анализа, а также изучение использованных алгоритмов и их непосредственное применение.

Для более простого ориентирования в документе рекомендую использовать навигацию (CTRL+F -> Навигация).

# Подготовка входных данных

При помощи программы Gephi были получены список вершин и список рёбер графа. На их основе построены матрица смежности и список смежности.

Построение матрицы смежности:

Создадим нулевую матрицу размера [n х n], где n – количество вершин.

В цикле считываются ребра, в соответствующих им ячейках матрицы поставляются единицы.

Построение списка смежности:

В цикле делаем обход по всем элементам построенной матрицы смежности. Если i-й j-й элемент равен 1, то в список смежности для i-го элемента добавляется j-ый.

Полученные матрицу смежности и список смежность можно посмотреть в архиве с выполненным заданием в папке *Resources* (файлы называются «*Adjacency-matrix.csv»* и «*Adjacency-list.csv»* соответственно).

В папках *Adjacency-matrix* и *Adjacency-list* представлен код для построения матрицы смежности и списка смежности соответственно.

# Задание 1

*Определить является ли граф (слабо/сильно) связным, найти число компонент слабой и сильной связности, число вершин в каждой из них, вычислить, какая доля узлов принадлежит наибольшей компоненте слабой связности.*

Создадим матрицу достижимости из матрицы смежности по следующей формуле:

где n – это количество узлов, – единичная матрица.

Используя матрицу достижимости, мы можем найти матрицу сильной связности C, состоящую из следующих элементов:

где – элемент матрицы

Если все элементы матрицы сильной связности равны 1, тогда граф является сильно связным.

В моем случае граф таковым не является.

Для нахождения компонент сильной связности необходимо взять первую строку матрицы связности и исключить из матрицы те узлы (то есть удалить соответствующие им строки и столбцы), которые смежны с узлом, соответствующим первой строке матрицы. То есть мы определяем смежные между друг другом узлы, записываем их в одну компоненту связности и удаляем из матрицы. Повторяем те же действия с оставшейся матрицей до тех пор, пока в ней не останется элементов.

Таким образом, было получено 7 компонент сильной связности:

|  |  |
| --- | --- |
| Номер компоненты | Число вершин в данной компоненте сильной связности |
| 1 | 7 |
| 2 | 26 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 6 | 1 |
| 7 | 1 |

Чтобы определить является ли граф слабо связным и определить его компоненты связности, нужно проделать те же самые действия с неориентированным графом.

В матрице связности присутствовали 0, значит исходный граф не является слабо связным.

Так как матрица связности для ориентированного графа и неориентированного у меня совпадали, компоненты сильной и слабой связности также совпали.

Таким образом, было получено 7 компонент связности:

|  |  |
| --- | --- |
| Номер компоненты | Число вершин в данной компоненте связности |
| 1 | 7 |
| 2 | 26 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 6 | 1 |
| 7 | 1 |

Наибольшей компоненте слабой связности принадлежат 26 узлов из 38, то есть 0.684211 от общего количества узлов.

В [приложении 1](#_Приложение_1) приведен код на языке C++, который был использован при выполнении задания.

# Задание 2

*Построить гистограмму плотности вероятности распределения степеней вершин. Вычислить среднюю степень вершины.*

*Определить диаметр, радиус графа, найти центральные и периферийные вершины, среднюю длину пути в графе.*

***Далее граф - это наибольшая компонента слабой связности без учета ориентации рёбер.***

## Построение гистограммы

Для построения гистограммы использовалась матрица смежности графа. Посчитаем для каждого узла его степень d. Затем в нулевом массиве D размера [1 x n] к элементу с индексом d прибавим 1. Таким образом по окончании работы цикла в D[i] будет содержаться информация о количестве узлов степени i.

Поделив все элементы данного массива на общее количество узлов в графе получим данные, на основе которых строится гистограмма плотности вероятности распределения степеней вершин.

В программе Microsoft Excel построим искомый график:

Для нахождения средней степени вершины были просуммированы все элементы матрицы смежности и сумма поделена на общее количество узлов.

Средняя степень вершины в графе - 11.8462

## Диаметр, радиус графа, его центральные и периферийные вершины, средняя длина пути в графе

***Радиусом графа****называется минимальный эксцентриситет среди всех вершин графа.*

***Диаметром графа****называется наибольшее расстояние между всеми парами вершин графа.*

***Центральной вершиной******графа*** *является вершина чей эксцентриситет равен радиусу графа.*

***Периферийной вершиной******графа*** *является вершина чей эксцентриситет равен диаметру графа.*

**Для нахождения данных характеристик необходимо определить эксцентриситет каждой вершины графа.**

***Эксцентриситетом****вершины называется расстояние до самой дальней вершины графа.*

**При помощи алгоритма Флойда-Уоршелла найдем матрицу расстояний графа, в которой указаны кратчайшие расстояния между всеми парами вершин.**

На вход алгоритм принимает матрицу, элементы которой равны 1 если существует путь между i-й и j-й вершиной, и n если такого пути нет. n в данном случае выбрано потому что между двумя вершинами путь не может быть длиннее чем количество вершин в графе.

Рассмотрим сам алгоритм:

Пусть вершины графа пронумерованы от 0 до n (количество вершин в графе) и введено обозначение для длины кратчайшего пути от i до j, который кроме самих вершин i и j проходит через вершины 1…k.

Существует два варианта значения , k ∈ (1, … , n)

1. Кратчайший путь между i, j не проходит через вершину k, тогда = .
2. Существует более короткий путь между i, j, проходящий через k, тогда он сначала идёт от i до k, а потом от k до j. В этом случае, очевидно, = .

Для нахождения значения функции необходимо взять минимум из двух обозначенных значений.

Алгоритм Флойда-Уоршелла последовательно вычисляет все значения ∀i, j для k от 1 до n. Полученные значения являются длинами кратчайших путей между вершинами i и j.

Теперь для нахождения эксцентриситетов вершин достаточно в цикле рассмотреть все элементы матрицы и найти максимальный в каждой строке.

Для определения диаметра и радиуса необходимо найти максимальный (диаметр) и минимальный (радиус) элементы в массиве, хранящем значения эксцентриситетов вершин.

Для нахождения центральных и периферийных вершин в цикле от 0 до n (количество вершин графа) определим вершины, чьи эксцентриситеты равны радиусу (центральная вершина) и чьи диаметру (периферийная вершина).

Среднюю длину пути в графе найдем по формуле:

,

n – количество вершин в графе, – кратчайший путь между вершинами i и j.

Результаты работы программы:

|  |  |
| --- | --- |
| Диаметр | 3 |
| Радиус | 2 |
| Центральные вершины | id36175307  id50952050  id58555820  id59477024  id64004147  id67446082  id76411897  id78602687  id82892660  id102945985  id137252115  id169387028  id177123098  id194682140  id216953513  id225812577  id373044930  id374868070 |
| Периферийные вершины | id10015045  id28315548  id41580584  id80659889  id119301334  id134070307  id157062074  id164285180 |
| Средняя длина пути в графе | 1.56213 |
| Среднюю степень вершины | 11.8462 |

*Вектор плотности вероятности распределения степеней вершин:*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 0 | 0 | 0 | 0,04 | 0,04 | 0 | 0 | 0 | 0,12 | 0,04 | 0,15 | 0,12 | 0,12 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 0,04 | 0,15 | 0 | 0,04 | 0,04 | 0,04 | 0,04 | 0,04 | 0 | 0 | 0 | 0 | 0 |

В [приложении 2](#_Приложение_2) приведен код на языке C++, который был использован при выполнении данного задания.

# Задание 3

*Вычислить меры сходства узлов графа (similarity measures) на основе:*

* *Common Neighbors (число общих соседей):*
* *Jaccard’s Coefficient (мера Жаккара):*
* *Adamic/Adar (Frequency-Weighted Common Neighbors):*
* *Preferential Attachment:*

## Число общих соседей

Создадим нулевую матрицу R[n x n]. Для каждой пары узлов i и j матрицы смежности A найдем такие элементы k что A[i][k] = A[j][k] = 1. В случае нахождения такого элемента добавляем 1 к R[i][j]. По окончании работы алгоритма R – матрица содержащая число общих соседей элементов i и j в R[i][j].

Матрица будет обладать свойством симметричности, а также на диагонали будут расположены степени соответствующих вершин графа.

## Мера Жаккара

Как и в предыдущем пункте найдем матрицу общих соседей I и аналогичным образом с заменой условия (A[i][k] = A[j][k] = 1) на (A[i][k] = 1 or A[j][k] = 1) матрицу U. После чего путем поэлементного деления матрицы I на U получим матрицу R, где R[i][j] – мера Жаккара для i-й и j-й вершин

Чтобы найти меру Жаккара, необходимо мощность пересечения разделить на мощность множества объединения соседей двух вершин.

Эта матрица также обладает свойством симметричности, а на диагонали у нее расположены 1, так как мощность пересечения и объединения соседей одной вершины совпадают.

## Frequency-Weighted Common Neighbors

Аналогично предыдущим алгоритмам, создадим нулевую матрицу R[n x n]. В цикле будем находить общих соседей k для вершин i и j и к R[i][j] прибавлять 1/log(N(k)), где N(k) – степень вершины k.

## Preferential Attachment

Для нахождения данной меры сходства узлов необходимо попарно перемножить степени вершин и записать результаты в матрицу R[n x n].

В [приложении 3](#_Приложение_3) приведен код на языке C++, который использовался при выполнении данного задания.

Полный код программы находится в папке «Part 1». В папке «Resources» находятся все найденные метрики, матрицы и списки смежности полного графа, матрица смежности наибольшей компоненты связности.

# Приложение 1

//находим матрицу достижимости

for (int i = 0; i < nodesNumber; i++) {

currMatrix = multiplication(currMatrix, currMatrix, nodesNumber);

resMatrix = logSum(resMatrix, currMatrix, nodesNumber);

}

//находим матрицу сильной связности

resMatrix = logMult(resMatrix, transp(resMatrix, nodesNumber) , nodesNumber);

//находим компоненты связности

int counter = nodesNumber;

int currFirstRow = 0;

while (counter > 0) {

for (int i = 0; i < nodesNumber; i++) {

if (nodesArr[i] == 1)

{

currFirstRow = i;

break;

}

}

for (int i = 0; i < nodesNumber; i++) {

if ((resMatrix[currFirstRow][i] == 1) && (nodesArr[i] == 1)) {

nodesArr[i] = 0;

counter--;

numberOfNodesInComponents[componentsNumber]++;

}

}

componentsNumber++;

}

int maxComponentAmount = 0;

for (int i = 0; i < nodesNumber; i++) {

if (numberOfNodesInComponents[i] > maxComponentAmount)

maxComponentAmount = numberOfNodesInComponents[i];

}

for (int i = 0; i < nodesNumber; i++) {

cout << i << ": " << numberOfNodesInComponents[i] << endl;

}

cout << "Доля узлов, пренадлежащих к наибольшей компоненте слабой связности: " << (float)maxComponentAmount / (float)nodesNumber << endl;

# Приложение 2

int diameter(int \* ecc, int n) {

int res = 0;

for (int j = 0; j < n; j++)

if (ecc[j] > res)

res = ecc[j];

return res;

}

int radius(int \* ecc, int n) {

int res = n;

for (int i = 0; i < n; i++)

if (res > ecc[i])

res = ecc[i];

return res;

}

float avgPathLength(int \*\* arr, int n) {

float res = 0;

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

res += arr[i][j];

return res/(n\*n);

}

int \* centralVertex(int \*ecc, int radius, int n) {

int \* res = new int[n];

for (int i = 0; i < n; i++) {

if (ecc[i] == radius)

res[i] = 1;

else

res[i] = 0;

}

return res;

}

int \* periphVertex(int \*ecc, int diameter, int n) {

int \* res = new int[n];

for (int i = 0; i < n; i++) {

if (ecc[i] == diameter)

res[i] = 1;

else

res[i] = 0;

}

return res;

}

float averageVertexPower(int \*\* arr, int n) {

float res = 0;

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

res += arr[i][j];

return res/n;

}

//с помощью алгоритма Флойда - Уоршелла найдем матрицу расстояний для графа

int \*\* pathLengthArr = new int \*[maxComponentAmount];

for (int i = 0; i < maxComponentAmount; i++) {

pathLengthArr[i] = new int[maxComponentAmount];

}

for (int i = 0; i < maxComponentAmount; i++) {

for (int j = 0; j < maxComponentAmount; j++) {

if (maxComponentAdjMatrix[i][j] == 1)

pathLengthArr[i][j] = 1;

else

pathLengthArr[i][j] = maxComponentAmount; //(вместо беск-ти)

//длина пути между двумя вершинами не может превышаь кол-во вершин в графе

}

}

for (int k = 0; k < maxComponentAmount; k++)

for (int i = 0; i < maxComponentAmount; i++)

for (int j = 0; j < maxComponentAmount; j++)

pathLengthArr[i][j] = min(pathLengthArr[i][j], pathLengthArr[i][k] + pathLengthArr[k][j]);

int \*ecc = new int[maxComponentAmount]; //эксцентриситет

for (int i = 0; i < maxComponentAmount; i++) {

for (int j = 0; j < maxComponentAmount; j++) {

if (pathLengthArr[i][j] > ecc[i])

ecc[i] = pathLengthArr[i][j];

}

}

int \* centralVnumbers = new int[maxComponentAmount];

int \* periphVnumbers = new int[maxComponentAmount];

for (int i = 0; i < maxComponentAmount; i++) {

centralVnumbers[i] = 0;

periphVnumbers[i] = 0;

}

arrRadius = radius(ecc, maxComponentAmount); //радиус графа

arrDiameter = diameter(ecc, maxComponentAmount); //диаметр графа

centralVnumbers = centralVertex(ecc, arrRadius, maxComponentAmount); //если в i-й ячейке стоит 1, то вершина - вентральная

periphVnumbers = periphVertex(ecc, arrDiameter, maxComponentAmount); //аналогично с центральными

avgPathLngth = avgPathLength(pathLengthArr, maxComponentAmount); //средняя длина пут в графе

//ищем количество узлов i-ой степени

float \* countOfDegrees = new float[maxComponentAmount];

for (int i = 0; i < maxComponentAmount; i++) {

countOfDegrees[i] = 0;

}

for (int i = 0; i < maxComponentAmount; i++) {

counter = 0;

for (int j = 0; j < maxComponentAmount; j++) {

counter += maxComponentAdjMatrix[i][j];

}

countOfDegrees[counter]++;

}

for (int i = 0; i < maxComponentAmount; i++) {

countOfDegrees[i] /= maxComponentAmount;

}

# Приложение 3

int \*\* CommonNeighbors(int \*\* adjMatrix, int n) {

int \*\* res = new int\*[n];

for (int i = 0; i < n; i++)

res[i] = new int[n];

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

res[i][j] = 0;

}

}

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

for (int k = 0; k < n; k++) {

if ((adjMatrix[j][k] == 1) && (adjMatrix[i][k] == 1))

res[i][j]++;

}

}

}

return res;

}

float \*\* JaccardCoefficient(int \*\* adjMatrix, int n) {

float \*\* res = new float\*[n];

int \*\* Intersection = new int\*[n];

int \*\* Union = new int\*[n];

for (int i = 0; i < n; i++) {

res[i] = new float[n];

Intersection[i] = new int[n];

Union[i] = new int[n];

}

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

res[i][j] = 0;

Intersection[i][j] = 0;

Union[i][j] = 0;

}

}

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

for (int k = 0; k < n; k++) {

if ((adjMatrix[j][k] == 1) && (adjMatrix[i][k] == 1))

Intersection[i][j]++;

if ((adjMatrix[j][k] == 1) || (adjMatrix[i][k] == 1))

Union[i][j]++;

}

res[i][j] = (float)Intersection[i][j] / (float)Union[i][j];

}

}

return res;

}

float \*\* Adamic(int \*\* adjMatrix, int n) {

float \*\* res = new float\*[n];

int \* degrees = new int[n];

for (int i = 0; i < n; i++) {

res[i] = new float[n];

degrees[i] = 0;

}

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

res[i][j] = 0;

if (adjMatrix[i][j] == 1) {

degrees[i]++;

}

}

}

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

for (int k = 0; k < n; k++) {

if ((adjMatrix[j][k] == 1) && (adjMatrix[i][k] == 1))

res[i][j] += 1 / log(degrees[k]);

}

}

}

return res;

}

int \*\* PreferentialAttachment(int \*\* adjMatrix, int n) {

int \*\* res = new int\*[n];

int \* degrees = new int[n];

for (int i = 0; i < n; i++) {

res[i] = new int[n];

degrees[i] = 0;

}

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

if (adjMatrix[i][j] == 1) {

degrees[i]++;

}

}

}

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

res[i][j] = degrees[i] \* degrees[j];

}

}

return res;

}