Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет»
(национальный исследовательский университет)
Факультет математики, механики и компьютерных наук
Кафедра прикладной математики

РАБОТА ПРОВЕРЕНА
Рецензент
Конова
«27» 06 2016 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н.,
доцент
Л.А.Прокудина
«18» 06 2016 г.

Разработка приложения для анализа истории распространения информации в сети Интернет.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–231000.2016.090.ПЗ ВКР

Руководитель работы, доцент кафедры прикладной математики
/А. К. Демидов
«18» 06 2016 г.
Автор работы
Студент группы ММиКН-472
/ З. И. Хуснутдинова
«__» _____ 2016 г.
Нормоконтролер, доцент кафедры прикладной математики, к.ф.-м.н.
/С. У. Турлакова
«18» 06 2016 г.

Челябинск 2016

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«Южно-Уральский государственный университет»
(национальный исследовательский университет)
Факультет математики, механики и компьютерных наук
Кафедра прикладной математики

Направление подготовки Программная инженерия

УТВЕРЖДАЮ
Заведующий кафедрой
_____ (Прокуратов)
_____ 2015 г.

З А Д А Н И Е

на выпускную квалификационную работу студента
Хуснутдиновой Зили Ильгизовны
Группа ММиКН-474

1. **Тема работы** Разработка приложения для анализа истории распространения информации в сети Интернет.
   утверждена приказом по университету от « 15 » апреля 2016 г. № 661
2. **Срок сдачи студентом законченной работы** « 17» **июня 2016 г.**
3. **Исходные данные к работе**
   3.1. Среда разработки - IntelliJ IDEA.
4. **Перечень вопросов, подлежащих разработке**
   4.1. Обзор существующих методов выявления информационных вбросов и слухов в сеть Интернет.
   4.2. Изучение форматов оформления сообщений в социальных сетях и форматов выдачи поисковых запросов.
   4.3. Разработка интерфейса пользователя и форматов выдачи результатов.
   4.4. Написание и отладка программы, проводящей анализ истории распространения новостей в сети
   4.5. Тестирование и отладка программы.
   4.6. Разработка программной документации (описание программы, текст программы, руководство пользователя)
5. **Иллюстративный материал** (плакаты, альбомы, раздаточный материал, макеты, электронные носители и др.)
   5.1. Схема алгоритма. Демонстрационный плакат – 1 л.
   5.2. Мультимедийная презентация – 15 слайдов.

Общее количество иллюстраций – 16.

6. **Дата выдачи задания** «1» октября 2015 г.

Руководитель _____ /А. К. Демидов/

(подпись)

Задание принял к исполнению _____ /З. И. Хуснутдинова/

(подпись)

7. **Календарный план**

Наименование этапов выпускной квалификационной работы	Срок выполнения этапов работы	Отметка о выполнении руководителя
1. Изучение методов распространения информации в сети интернет.	01.10.15 – 07.11.15	
2. Изучение форматов оформления сообщения в социальных сетях и форматов выдачи поисковых запросов.	09.11.15– 19.12.15	
3. Разработка алгоритма, интерфейса пользователя и форматов выдачи результатов.	21.12.15-06.02.16	
4. Программная реализация алгоритма	08.02.16 – 05.03.16	
5. Отладка и тестирование программы	07.03.16 – 26.03.16	
6. Вычислительные эксперименты	28.03.16 – 16.04.16	
7. Подготовка пояснительной записки дипломной работы	18.04.16 – 15.05.16	
8. Разработка программной документации	02.05.16 – 15.05.16	
9. Проверка работы руководителем, исправление замечаний	16.05.16 – 19.05.16	
10. Нормоконтроль	20.05.16 – 26.05.16	
11. Подготовка иллюстративного материала и доклада	23.05.16 – 14.06.16	
12. Рецензирование, представление зав. кафедрой	14.05.16 – 17.06.16	

Заведующий кафедрой _____ /Л. А. Прокудина /

(подпись)

Руководитель работы _____ / А. К. Демидов /

(подпись)

Студент _____ / З. И. Хуснутдинова /

(подпись

АННОТАЦИЯ

Хуснутдинова З. И. Разработка приложения для анализа истории распространения информации в сети Интернет.– Челябинск: ЮУрГУ, ММиКН-474, 40 с., 26 ил.,1 табл., библиогр. список – 20 наим., 1 прил., 1 л. плакатов ф. А1.

В работе исследованы методы и существующие системы анализа истории распространения слухов, информационных вбросов и естественных новостей в сети Интернет. Построена сеть распространения информации в Интернете с указанием предполагаемой даты публикации новости. Реализована программа визуализации полученной сети распространения. Построенную модель можно улучшить и использовать в качестве исходных данных для некоторого алгоритма классификации.

ОГЛАВЛЕНИЕ

Введение 6
1 Информационные вбросы и слухи 8
1.1 Слухи 8
1.2 Анализ слухов при помощи параметров их распространения 9
1.3 Модель SpikeM 11
1.4 Observatory on Social Media 12
1.5 Emergent 14
1.6 Информационные вбросы 15
1.7 Вычисление источников распространения 17
Выводы по разделу 18
2 Проектирование системы 19
2.1 Выбор инструментов для разработки 19
2.2 Представление результатов выдачи поисковых запросов 19
2.3 Построение сети распространения 21
2.4 Структура представления данных в социальной сети Facebook 21
2.5 Структура представления данных в социальной сети ВКонтакте 22
2.6 Определение даты публикации новости 23
2.7 Интерфейс пользователя 24
2.8 Описание архитектуры системы. Диаграмма классов 25
Выводы по разделу 30
2 Алгоритмы 31
3.1 Алгоритм определения даты создания веб-страницы. 31
3.2 Алгоритм построения сети распространения 31
3.3 Преобразование дерева сети распространения для вывода на экран 34
Выводы по разделу 35
4 Результаты 36
Заключение 38
Библиографический список. 39
Приложения 41
ПРИЛОЖЕНИЕ 1. Текст программы. 41

ВВЕДЕНИЕ

Интернет является огромной коммуникационной площадкой, которая часто используется для манипулирования общественным мнением. Сеть позволяет не только сформулировать определенные взгляды общества на некоторое событие или явление, но и изменить их - что показало себя как интересное направление для исследований.

Сегодня внимание многих исследователей привлечено к изучению особенностей распространения информации в сети Интернет. Широкие возможности современных компьютеров, повсеместное распространение и возрастающая доступность Интернета создают благодатную почву для распространения информации. Любое информационное сообщение, любая новость передается быстрее и шире, чем в реальном мире. А с увеличением количества людей, получивших доступ к Всемирной паутине, увеличивается и объем информации в Сети.

Среди всего объема информации, который циркулирует по сети Интернет, особый интерес для изучения вызывают различные информационные вбросы и слухи. Учитывая то, что Интернет обладает большим доверием пользователей, очень важно выявлять, какая, например, новость, является естественной, а какая выброшенной намеренно. Связано это с тем, что различные блоги, социальные сети, новостные каналы и другие Интернет-ресурсы все больше увеличивают свое влияние на другие виды средств массовой информации, даже в реальной жизни.

Приведем пример того, как может работать информационных вброс и как он может повлиять на СМИ в реальности. Некоторый человек опубликовал некоторую новость в своем блоге. Другие люди заинтересовались этой новостью и начали о ней писать. Так информация получает распространение. Далее, какая-нибудь газета (телевизионная передача и т. д.) публикует статью об этом событии, говоря о нем «так пишут в Интернете». Эти СМИ порождают новую волну распространения информации в Интернете, ведь теперь об этом говорят не только в блогах, но и в газетах. Подобная тенденция усиливается с каждым годом.

Из вышесказанного следует, что важно понимать, что из информации в Интернете является слухом, что естественной новостью, а что вбросом, так как последние являются средством сильного воздействия на сознание людей. Так же распространение слухов и вообще любой ложной информации может привести к значительным экономическим и политическим последствиям. Нередко они используются как орудие Информационной войны.

Большую часть информации люди получают из различных социальных сетей и блогов. Подобные сайты являются огромным неконтролируемым источником распространения сообщений, которые могут искажать реальные факты.

Пользователи, с настороженностью относящиеся к информации в социальных сетях, а также люди, которые услышали о чем-то «на улице», стремятся проверить информацию, которую они получили, и узнать, что об этом говорят в мире. Для этого обычно применяются поисковые системы, например, такие как Google, Яндекс и тому подобные. В зависимости от того, какие результаты будут получены, волна распространения информации может как усилиться, так и

ослабнуть. Поэтому важно рассматривать в качестве средства распространения информации не только блоги и социальные сети, но и поисковые системы. Более того, можно искусственно повысить позицию сайта при выдаче результатов поискового запроса и простимулировать дальнейшее распространение какого-либо сообщения.

В связи в этим была предпринята попытка разработать систему, которая смогла бы проводить автоматический сбор информации о распространении некоторой новости (сообщения, факта, то есть порции информации) по результатам поискового запроса в некоторой поисковой системе и предоставить полученные данные в виде, который сможет облегчить анализ истории распространение информации в сети Интернет.

Цель работы: разработать систему, которая могла бы по результатам некоторого поискового запроса (в запросе задана информация, которая нас интересует) предоставить данные для анализа истории распространения некоторой информации в сети Интернет. Представить полученные результаты в виде, удобном для анализа. Построить граф сети распространения новости по различным сайтам, а также собрать информацию о дате публикации полученных источников распространения.

Задачи: для того, чтобы построить систему, необходимо:

1) Изучить существующие методы выявления информационных вбросов и слухов в сети Интернет.
2) Изучить форматы оформления сообщений в социальных сетях и форматы выдачи результатов поисковых запросов.
3) Разработать алгоритм, обрабатывающий полученный набор ссылок и устанавливающий связи между ними
4) Разработать алгоритм, устанавливающий дату создания страницы по заданной ссылке
5) Представить полученные данные в виде графа, в котором также будет видна интенсивность распространения (количество упоминаний) по дням

Объектом исследования является явление распространения различных информационных вбросов и слухов в сети Интернет.

Предметом разработки является программа, которая собирает результаты выдачи поискового запроса и строит граф распространения информации.

Полученные результаты в дальнейшем можно применить для построения анализатора, который по представленным данным может определить, является ли данная тема информационным вбросом(некоторые слухи также могут являться вбросом) или слухом, который появился самостоятельно, без воздействия извне, или естественным событием, которое без какого-либо внешнего воздействия распространялась по Всемирной паутине.

1 ИНФОРМАЦИОННЫЕ ВБРОСЫ И СЛУХИ

Каждый день в сети Интернет циркулируют громадные объемы информации. Среди них встречаются и ложные, и правдивые сведения. Каждая такая тема, порция информации привлекает к себе некоторое внимание. Но среди них есть и такие, которые находят особенно сильный отклик у пользователей. Подобные темы становятся предметом активного обсуждения, люди делятся с ними друг с другом не только в сети, но и в реальности.

Особенность Интернета заключается в том, что любая появившаяся в нем информация может захлестнуть огромное количество людей. Новость может появиться из-за того, что произошло некоторое событие. Такие новости распространяются сами по себе, интерес к ним никак не подогревается. Убывание и возрастание популярности этой темы происходит самостоятельно. Но есть и такие новости, выброшенные по чьей-то злой воле, по чьему-то желанию. Их называют информационными вбросами.

Значительную часть информации в Интернете составляют слухи. Они в большинстве случаев быстро распространяются и привлекают к себе огромный интерес. Одним из методов продвижения какой-либо новости или слуха является выброс информации по данной теме во Всемирную паутину, то есть слухи могут быть пущены при помощи информационных вбросов.

В данном разделе будут даны более подробные определения и описания слухов и информационных вбросов, а также приведены современные методы их анализа.

1.1 Слухи

Слухи являются важным элементом коммуникации между людьми. Они являются недостаточно проверенными сведениями, которые передаются в ходе межличностного общения. В наше время, когда электронные СМИ способны разнести новость во все концы Земли, когда любой человек может обменяться мнением другим человеком и поделиться информацией, особенно важно понимать механизм распространения слухов.

Особенно опасными слухи становятся во время войн, так как они могут подорвать веру людей в победу и посеять панику. В связи с этим американцы очень тщательно подошли к изучению этого вопроса. Во время Второй Мировой войны они поняли, что слухи активней всего распространяются во время отсутствия информации, поэтому ведомство военной информации старалось быстро и с максимальной полнотой предоставлять людям точную информацию о положении дел на фронтах. В СССР распространение слухов активно преследовалось, таких людей называли паникерами и приравнивали к пособникам врага.

Наибольший вклад в вопросы изучения слухов внесли американцы [1, 2]. Так, Р. Напп, автор статьи «Психология слуха» предложил свою классификацию слухов (1944 г.). Основой для классификации было положено мотивационное желание распространителей:

- Слух-мечта (Pipe dream rumor) – отражает и удовлетворяет надежды и стремления людей
- Слух-пугало (Bogie rumor) – приводит к тревоге, неуверенности, вызывает страхи.
- Слух-раделитель (Wedge-driving aggression Rumor) – подобные слухи призваны разделить людей на группы и вызывать агрессию между ними

Основоположниками же теории исследования слухов являются американские социологи Г. Олпорт и М. Постман. Они ввели определение слуха как истории или сообщения, которое не имеет под собой никаких оснований или фактов, доказывающих его истинность на момент его распространения. При том слухи распространяются, если даже распространитель указывает, что все что он говорит, является слухом. Сама суть слухов в их неопределенности, поскольку именно она способствует из широкому распространению. Также Олпорт и Постман построили некоторую модель распространения слуха, положив в ее основу два факта: важность темы для окружающих и неясность контекста, к которому они привязываются. Но последние исследования показали, что их предположение насчет важности слухов неверно, так как люди все зависимости от этого параметра распространяют слухи.

На данный момент существует множество теорий распространения слухов. Одна из них гласит, что люди распространяют слухи для того, чтобы почувствовать себя значимыми, почувствовать себя частью общества и привлечь к себе внимание [3, 4]. Приверженцы другой теории утверждают, что слухи характеризуются некоторой сильной эмоциональной окраской, которая привлекает людей. Также слухи особенно активно распространяются в моменты, когда люди не уверены в будущем, либо испытывают сильное чувство неопределённости и страха.

В частности, слухи активно распространяются в военные и послевоенные периоды. Ярким доказательством такого подхода может быть слух о начале Третьей мировой войны, который распространился в Калининградской области в 1947 году [5] и вызвал довольно большую панику среди населения, вплоть до того, что люди начала выезжать из области.

С ростом популярности социальных сетей исследование методов идентификации слухов получило особенно большое распространение. Рассмотрим некоторые работы из этой области.

1.2 Анализ слухов при помощи параметров их распространения

Корейский институт передовых технологий совместно с Microsoft research провели исследование динамики распространения слухов в социальных сетях [3, 4, 6]. Они использовали данные, полученные из Twitter. В наборе содержалась информация о профилях 54 миллионов пользователей, 1.9 миллиардов подписок пользователей друг на друга и 1.7 миллиардов твитов, сделанных в период с марта 2009 года до августа 2009 года. Анализ полного набора пользователей, подписок и твитов предоставил уникальную возможность изучить поведение пользователей

непосредственно во время процесса распространения информации. Отобрав истинные и ложные слухи, они проанализировали 90 дней с начала периода распространения каждой новости.

Исследование особенностей распространения слухов проводилось на основе следующих параметрах распределения: временных (temporal), структурных (structural) и лингвистических(linguistic).

Временные характеристики. Социальные психологи установили, что распространителям слухов внимание уделяется в течение короткого промежутка времени, так как слух распространяется только в течение некоторого времени, пока нет информации, подтверждающей или опровергающей эту новость.

Исследователи рассмотрели несколько популярных тематических меток – хэштегов и построили временной ряд для количества упоминаний слухов и не слухов в течение промежутка времени, приблизительно равного 2-3 месяцам. Слухами являлись следующие темы (см. рис. 1.2.1): Bigfoot, AdCall, Catfish, HIV Ketchup, FanDeath, Dork. Реальным событиям соответствовали следующие метки: Havard, Summize, Rockefeller, District9, BreastIceCream, ObamaFly.

Было выявлено, что отличительной чертой слухов является наличие множественных периодических скачков значений функции интенсивности распределения, в то время как у неслухов наблюдается не более одного скачка. Исследователи также ввели понятие функции внешних периодических воздействий, которая не применялась до сих пор. Эта функция описывала воздействия, которые использовались, чтобы поддержать распространение слуха.

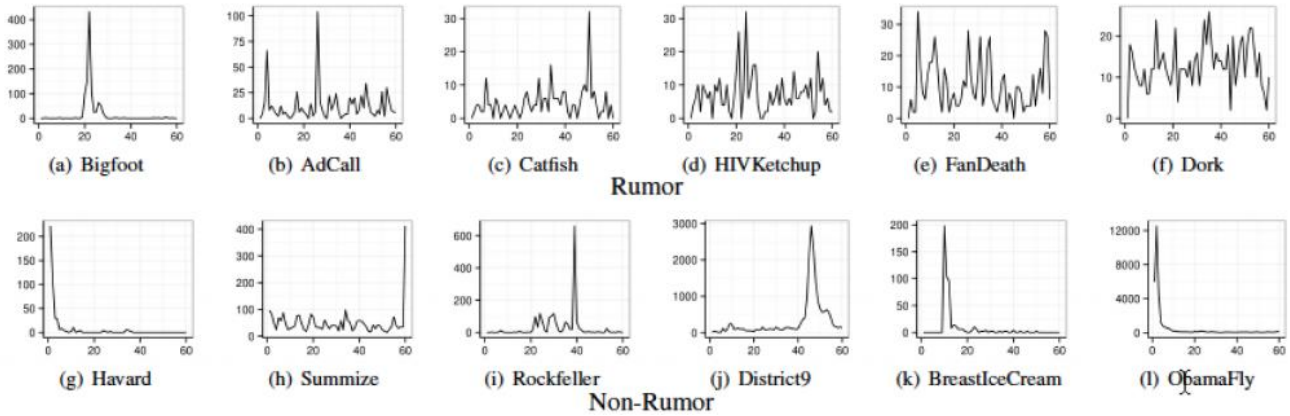

Рисунок 1.2.1 - Графики интенсивности распространения слухов (rumor, a-f) и не слухов (non rumor, g-l)

Структурные характеристики. Было введено понятие дружеской сети, как подграф первоначального графа подписчик-подписка, вершинами в котором являются пользователи, сделавшие хотя бы один перепост записи другого пользователя или имеющий подписки на других пользователей. Удалив ребра (связи) из дружеской сети, по которым не происходит распространение новости, была получена сеть распространения новости. Было выявлено, что для реальности и слухов сети распространения имеют различные структуры. По графу,

приведенному на рисунке 1.2.2, видно, что граф для темы Bigfoot (слух) содержит больше вершин одиночек, чем Summize(реальность). Это значит, что слухи распространяются от множества отдельных пользователей, а реальные новости от одних пользователей к другим.

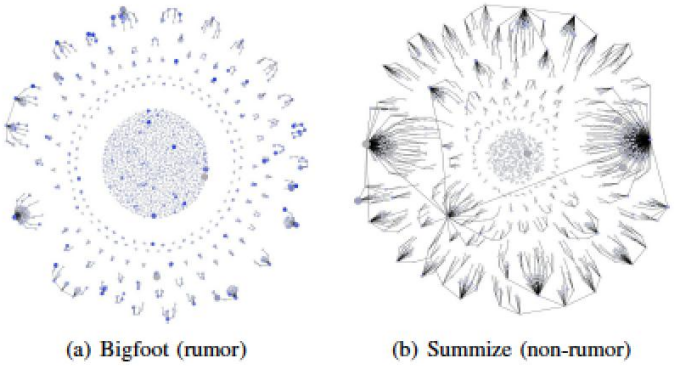

Рисунок 1.2.2 - Сеть распространения: а) слуха(Bigfoot); б)реальности (Summize)

Лингвистические характеристики. С помощью программы LIWC (Linguistic Inquiry and Word Count – Лингвистическое исследование и подсчет слов) была проанализирована тональность каждого твита. Было выделено, что:

- В слухах преобладают слова, вызывающие скептицизм, отрицание и сомнения.
- Много слов, указывающих на социальные отношения (друг, знакомы и т. д.)

Предложенная модель классифицировала слухи с точностью 87-92%, что выше, чем у других классификаторов.

1.3 Модель SpikeM

Matsubara и др. предложили модель SpikeM [7], описывающую возрастание и убывание интенсивности распространения различных новостей в Сети. SpikeM применяет модель заражения SI (Susceptible Infected – Восприимчивая к заражению) из эпидемиологии для описания поведения резких скачков функции интенсивности распространения и приближения полученного временного ряда к экспоненциальному закону. Тем не менее, модель SpikeM не разрабатывалась для анализа слухов, потому что в ней нет параметров, которые могли бы объяснить множественные скачки значений функции для слухов, и единичные скачки для реальных новостей.

Каждый объект в данной модели может находиться только в одном из двух состояний – уязвимом (S) и инфицированном (I). Каждый зараженный узел стремится заразить каждый из соседних узлов независимо, с некоторой вероятностью β, выражающей заражающую способность узла. Заразившись однажды, каждый узел навсегда остается в таком состоянии.

Модель была протестирована на данных MemeTracker, Twitter и GoogleTrends. Выполнив приближение этих данных, модель показала довольно точные результаты приближения. Как указывают авторы модели, ее можно применять для

предсказания дальнейшего поведения новости. Что важно, так как если новость являлась вбросом или слухом, предсказав поведение можно вовремя принять меры, чтобы приостановить ее распространение.

1.4 Observatory on Social Media

Интересным является проект Индианского университета OSOME (Observatory on Social Media) о том, как информация распространяется в социальной сети Twitter [8, 9]. OSOME представляет собой набор инструментов для теоретического и эмпирического анализа того, как структура социальных сетей может помочь предсказать, какие темы могут стать вирусными, как внимание на новость и эмоции, которые эта новость вызывает у людей, могут повлиять на процесс распространения и взаимосвязь между активностью пользователей и текущей структурой сети. Проект включает в себя 4 крупных сервиса.

Для использования этих приложений необходима авторизация в Twitter.

BotOrNot (проект Truthy). Программа выполняет проверку активности пользователя и предоставляет оценку того, с какой вероятностью пользователь является ботом. На рисунке 1.4.1 приведен пример работы программы для пользователя @JusBieberPhotos.

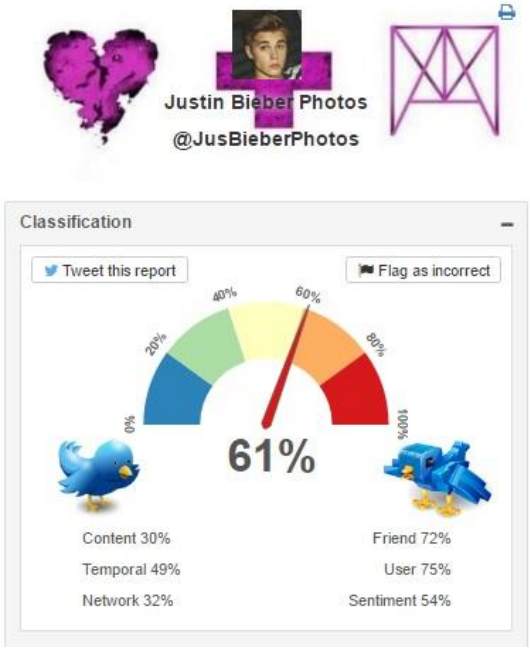

Рисунок 1.4.1 - Пример анализа учетной записи Twitter с помощью BotOrNot

Подобные боты часто применяются для распространения новостей. Есть упоминания о случаях, когда их использовали во время политических кампаний, чтобы увеличить популярность того или иного кандидата.

Trends. Сервис анализирует количество упоминаний некоторого хэштега (тематической метки) и показывает, как оно изменялось за заданный промежуток времени (см. рис. 1.4.2). По оси абсцисс отложены даты, по оси ординат –

количество упоминания данной тематической метки в Twitter, представленное в логарифмической шкале.

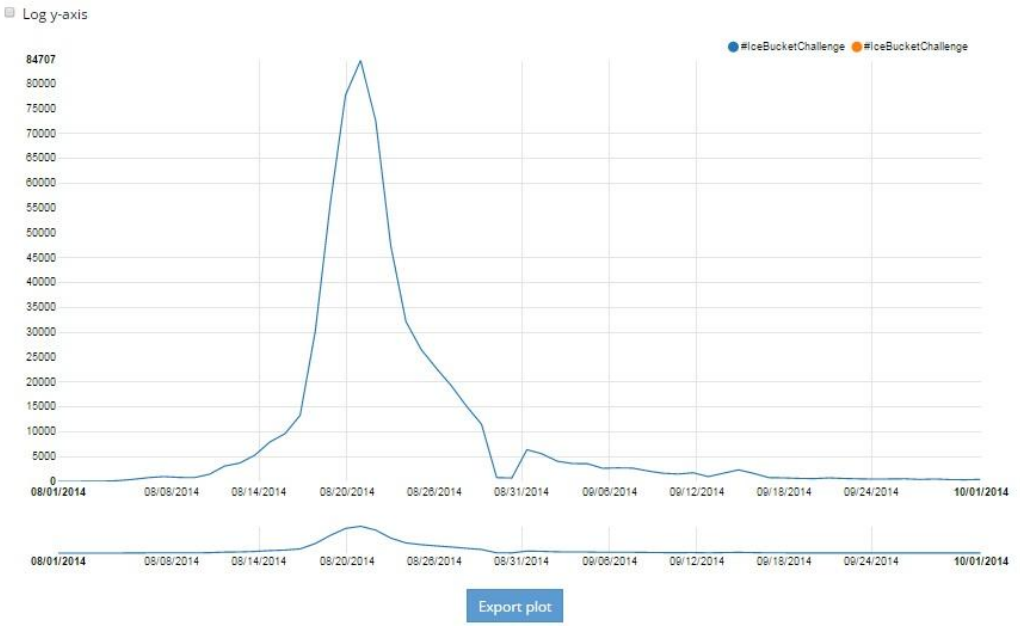

Рисунок 1.4.2 - Пример работы Trends для #IceBucketChallenge.

Networks. Сервис, который строит граф распространения новости. Вершины – пользователи, опубликовавшие новость с указанным хештегом или сделавшие перепост, ребра – перепосты между пользователями, и упоминания одних пользователей другими. На рисунке 1.4.3 приведён пример для тематической метки #IceBucketChallenge. Помимо того, что сервис строит граф сети распространения информации, он также предоставляет возможность получить профили людей, участвовавших в процессе распространения.

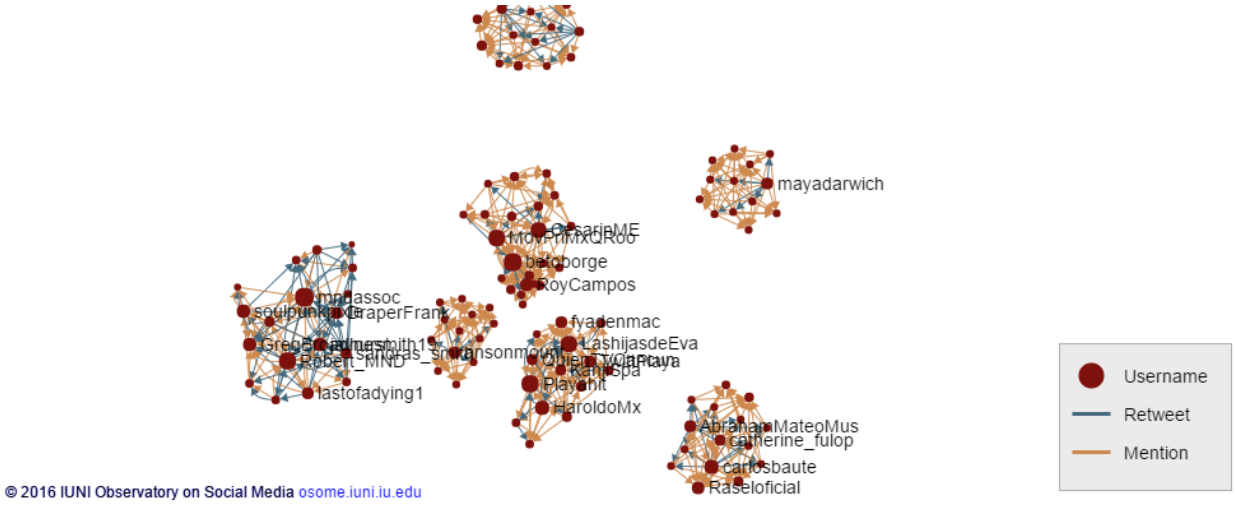

Рисунок 1.4.3 - Пример работы Networks для #IceBucketChallenge.

Maps. Создает точную карту, где указано, в каких уголках Земли тема обсуждается. Также можно посмотреть карту для указанного дня. На рисунке 1.4.4 приведен пример такой карты. Сервис интегрирован с платформой Google Maps, что позволяет подробно рассмотреть географические положения участников распространения.

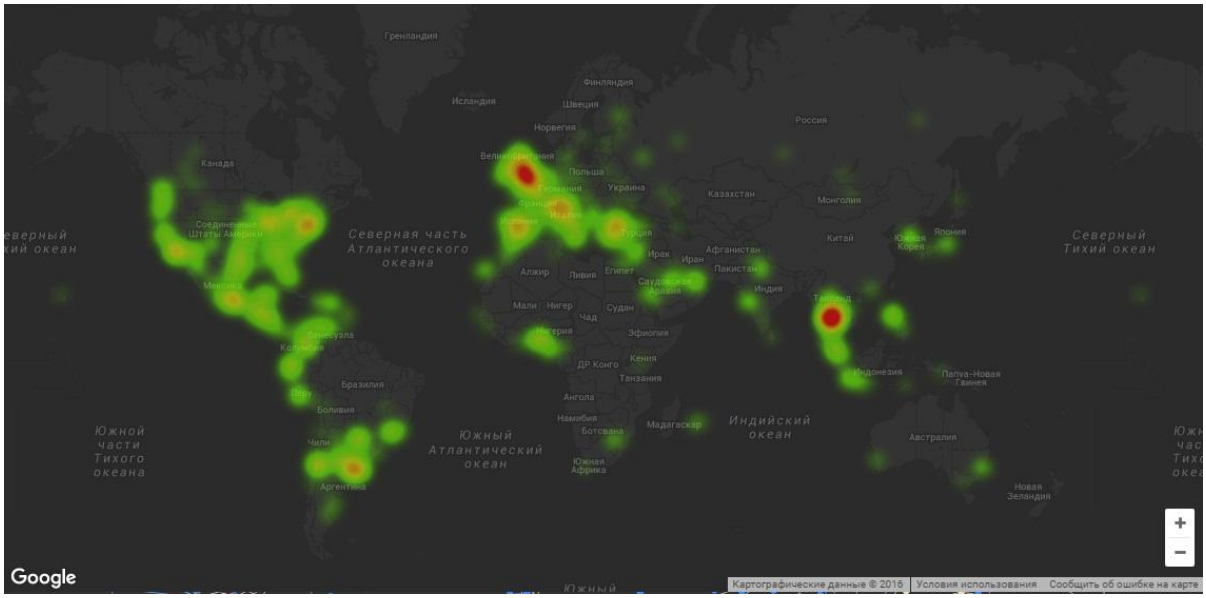

Рисунок 1.4.4 - Пример работы Maps для #IceBucketChallenge

1.5 Emergent

Журналист Крейг Сильверман и американский центр цифровой журналистики при Колумбийском университете запустили сервис Emergent, который позволяет отслеживать, как непроверенная информация и слухи распространяются по сети [10, 11]. Они собирают слухи, которые появляются в интернет-новостях и заносят их в базу данных. Каждой истории присваивается «рейтинг правдивости», дата, когда история появилась в сети и число ресурсов, которые на него ссылаются (либо перепечатали). Далее специальный алгоритм каждый час отслеживает URL-адрес этой страницы и проверяет, были ли внесены изменения в страницу. Кликнув на тему, которая занесена в базу данных сайта, можно увидеть, сколько ресурсов подтвердили этот слух, сколько – опровергли, и сколько ресурсов на него ссылается. Сервис был разработан для того, чтобы помочь журналистам и людям разобраться в новостях, которые они получают из сети Интернет, особенно из социальных сетей. Каждая новость отмечается Unverified (Если новость пока не получила ни подтверждения, ни опровержения), True(подтверждена) и False(опровергнута) (см. рис 1.5.1).

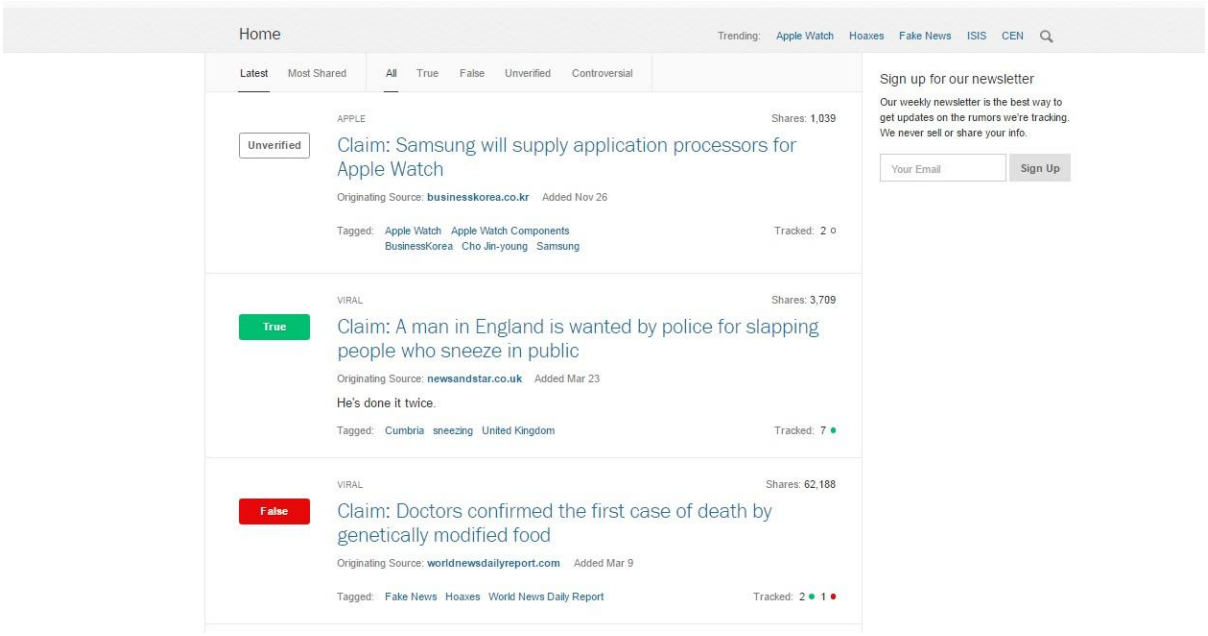

Рисунок 1.5.1 - Главная страница сайта Emergent.

Кликнув по каждому заголовку можно просмотреть информацию о том, как распространялся слух, какие источники его подтверждали, а какие опровергали. Также отслеживается, какие изменения были внесены в каждый источник. Пример подробной записи о некоторой новости приведен на рисунке 1.5.2.

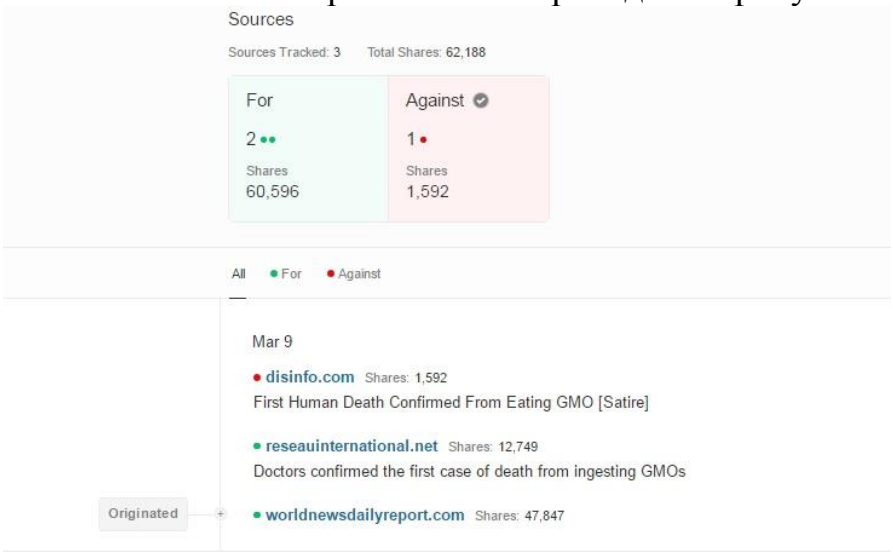

Рисунок 1.5.2 - Анализ новости Claim: Doctors confirmed the first case of death by genetically modified food (ложная)

1.6 Информационные вбросы

Одним из методов распространения слухов являются информационные вбросы – намеренное опубликование некоторой информации в сети, с расчетом неожиданности их обнародования, чтобы вызвать у общества некоторую ожидаемую реакцию; искусственное раздувание интенсивности упоминания некоторой темы. Чаще всего в таких новостях указывают на некоторого общего врага, против которого нужно бороться, то есть мы наблюдаем явное

противопоставление двух сторон. Некоторые слухи так и не достигают своей цели после вброса, и быстро угасают. Но есть и те, которые вызывают бурную реакцию у людей.

Исследованием темы вбросов занимается управляющий партнёр компании «Ашманов и партнеры» Игорь Ашманов [12, 13]. Компания работает в сфере интернет-маркетинга. Они разработали такую систему как «Крибрум» — систему мониторинга и анализа социальных медиа для управления репутацией в Интернете. Созданная в компании система анализа происходящего в блогах и социальных сетях применялась для получения информации клиентов об их бизнесе. Но увеличение количества обсуждений вокруг Партиарха и РПЦ заставили задуматься и проанализировать то, как те или иные темы освещались в блогах.

Проанализировав распространения различных новостей, как естественных, так и вбросов, Ашманов пришел к заключению, что существует несколько факторов, которые указывают на то, что новость является вбросом:

- Характер новости. Чаще всего для вброса выбирается реальный факт о событиях, которые произошли довольно давно. История не только приукрашивается, но и имеет некоторый срок давности.
- Характер распространения. Суточный график распространения естественного события выглядит как хаотичная череда всплесков (выяснились новые подробности, произошли новые события, связанные с данной новостью), много разных авторов пишут из разных мест (см. рис. 1.6.1). А суточный график вброса имеет резкий пик – то есть большинство авторов перепубликовали одно и тоже, в одно и то же время (см. рис. 1.6.2). Также интересно то, что график распространения для вбросов по неделям выглядит как «полочка», как только интерес к новости начинает спадать, выполняется новый вброс. А для естественной новости мы наблюдаем нормальный фронт распространения, со множественными скачками. Связано это с тем, что с течением времени люди узнают новые подробности о событии, что снова вызывает интерес у пользователей.
- Отношение количества перепечаток и оригиналов. Для вбросов характерно малое количество оригиналов. Естественная новость имеет высокую значимость, каждый автор старается создать свой материал и написать его своими словами. У вбросов наблюдается всего несколько оригиналов и огромное количество перепечаток, как явных, так и неявных. Также естественные новости чаще всего начинают распространение через крупные новостные источники, которые чаще всего сами создают свой контент.
- Характер аккаунта из социальной сети, из которого приходит оригинальное сообщение. Чаще всего вбросы делаются через никому неизвестные аккаунты, которые были созданы совсем недавно. Имена пользователей таких аккаунтов обычно сгенерированы случайно и выглядят неправдоподобно. Подобные боты создаются заранее и в течение некоторого времени имитируют поведение обычного пользователя.

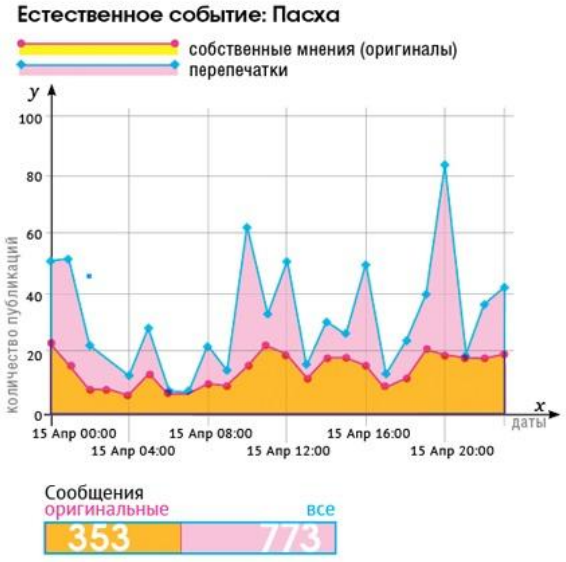

Рисунок 1.6.1 - График суточного распространения новости "Пасха".

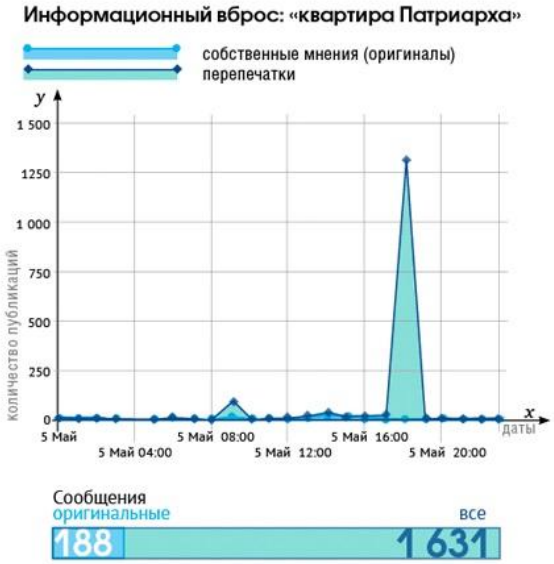

Рисунок 1.6.2 - График суточного распространения вброса "квартира Патриарха"

Иногда вбросы имеют настолько сильное воздействие, что новость полностью завладевает вниманием людей. В этом случае ее именуют ментальным или информационным вирусом. Назовём ментальным вирусом упорядоченную информационную структуру, созданную искусственно, с целью захвата власти над умами, способную при передаче в виде информационного сообщения того или иного формата (новость, книга, статья, письмо, ролик, фильм, песня, пр.) захватывать внимание неподготовленного субъекта, превращаться для него в навязчивую идею, подчинять себе мышление, структурировать его и делать субъекта восприимчивым к внешнему управлению. Сетями распространения ментальных вирусов обычно служат различные СМИ, а «местом посева» - Интернет.

1.7 Вычисление источников распространения

Группа исследователей из Швейцарского технологического института в Лозанне (EPFL) разработали алгоритм, позволяющий добираться до источников эпидемии любого рода, включая информационные [14, 15]. При наличии даже ограниченного объема информации о распространении (то есть, необязательно значит состояние всех узлов сети распространения), алгоритм выявляет наиболее вероятный первоисточник заражения.

Исследователи ввели сетевую модель, где каждый зараженный объект представлялся в виде узла, соединенного с некоторыми другими при помощи линий (взаимодействие). Сначала все узлы являются чистыми, затем один источник s начинает распространять инфекцию, причем каждое заражение происходит с произвольной задержкой. В итоге каждый «узел» оказывается

заражённым — и запоминает время, и персональный источник заражения. Также вводится *K* наблюдателей, которые определяют, в каком направлении информация двигалась к активным наблюдателям, и время, когда поток достиг этих наблюдателей. Предварительно алгоритм был протестирован на четырёх различных графовых структурах (Аполлониана, Барабаши-Альберта, Эрдёша-Реньи (при Np=0.5 и Np=2.5)).

Испытание алгоритма проводилось на реальных данных об эпидемии холеры. В 2000 году в Южной Африке произошла вспышка холеры. Вектором распространения заражения оказались реки, в качестве «узлов» исследователи рассматривали пострадавшие селения, связанные реками. Даже при использовании метода случайного отбора «наблюдателей», при наличии данных о 20% пострадавших селений исследователям удалось предсказать реальный источник эпидемии с относительно небольшой погрешностью.

Выводы по разделу.

С распространением интернета выяснение того, является ли новость информационном слухом, вбросом, или естественной новостью, особенно важна. Также существует множество работ, которые анализируют новости, но они привязаны к конкретным платформам и социальным сетям, так как сбор данных для исследования является трудоемкой задачей. Данная работа предоставит удобный и простой инструмент для визуализации структуры сети распространения новости.

2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ

В данной главе будут рассмотрены особенности проектирования системы, методы извлечения информации и сторонние библиотеки, используемые в работе. Также будет описана архитектура разрабатываемой программы.

2.1 Выбор инструментов для разработки

В качестве языка разработки был выбран язык программирования Java, так как он предоставляет большой набор инструментов для обработки web-страниц и манипулирования различными данными. Среда разработки: IntelliJ IDEA.

Для извлечения данных из HTML-документов была использована библиотека Jsoup. Она предоставляет удобный API для извлечения данных из web-страниц и манипулирования ими. Библиотека представляет обрабатываемый файл в виде DOM – древовидной структуры элементов страницы.

Jsoup может получать HTML-документы из файлов, строк или по URL. Также библиотека позволяет манипулировать элементами страницы, их атрибутами и содержимым. Также библиотека позволяет проводить синтаксический анализ не только HTML-документов, но и XML файлов, что будет использовано для получения результатов поисковых запросов от Yandex.

Для работы с датами публикаций новостей была выбрана библиотека Joda-time. Она предоставляет некоторые функции обработки дат, которых нет в стандартных библиотеках Java.

Для визуализации объектов будем использовать Swing 2d API. Данный интерфейс дает возможность рисовать простейшие геометрические фигуры, которые будут использованы для визуализации графа сети распространения информации, который будет отображать не только связи между различными источниками, но и интенсивность распространения сообщений по дням.

2.2 Представление результатов выдачи поисковых запросов *Google.*

Рассмотрим HTML-разметку результатов выдачи поискового запросы. Для анализа документа воспользуемся библиотекой Jsoup. Она предоставляет возможность получить доступ к документу без использования специальных API.

Для того, чтобы получить набор ссылок, проанализируем структуру документа.

Результаты выдачи поисковых запросов Google находятся в div-е, имеющем класс srg. Как видно из рисунка 2.6.1, в этот элемент вложены контейнеры div с классом g (далее div.g), которые непосредственно содержат описание полученных страниц.

Также дополнительно рассмотрим элемент, который появляется для актуальных новостей. Результаты из этого контейнера также включим в набор обрабатываемых ссылок.

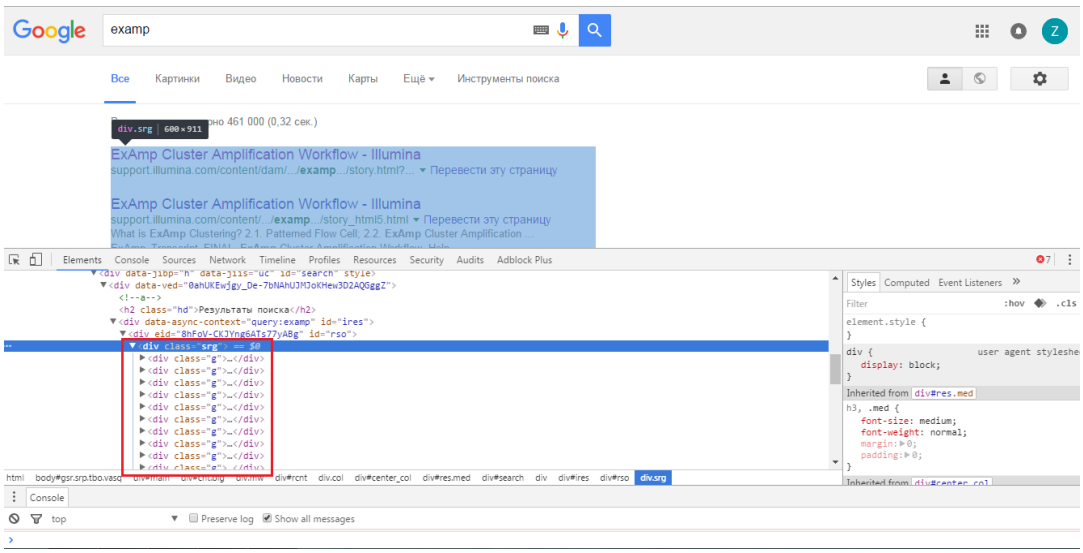

Рисунок 2.6.1 - Представление результатов поискового запроса Google.

Рассмотрим содержимое контейнера div.g подробнее (рис. 2.6.2).

```
▼<div class="g"> == $0
  <!--m-->
▼<div class="rc" data-hveid="49">
  ▼<h3 class="r">
      <a href="https://www.linkedin.com/company/examp" onmousedown="return
      rwt(this,'','','','4','AFQjCNG6RCnm4HPezwUpgUmWZphrzV_fJQ','P6y4pB11PVirqOWNrFIKPA','0ahUKEwjgy_De-
      7bNAhUJMJoKHew3D2AQFggyMAM','','',event)" target="_blank">EXAMP | LinkedIn</a>
    </h3>
  ▶<div class="s">…</div>
  </div>
  <!--n-->
</div>
```

Рисунок 2.6.2 - Содержимое контейнера div.g.

Значение, которое мы ищем, находится внутри тега <a>, находящегося внутри тега <h3>. Значит, для того, чтобы получить набор результатов ссылок из поискового запроса Google, найдем контейнеры div.srg, и оттуда извлечем все значения параметра href тега <a> из тега h3.r.

Yandex.

Yandex предоставляет API Yandex.XML, который позволяет получить результаты поискового запроса в виде XML-документа. Сервис позволяет настроить строку запроса, по которой будет получен результат.

Проанализировав структуру полученного документа, заметим, что ссылки на страницы результатов выдачи находятся внутри тегов <url></url>, следовательно, адреса требуемых страниц будет извлекать из содержимого указанных тегов (см. рис. 2.6.3).

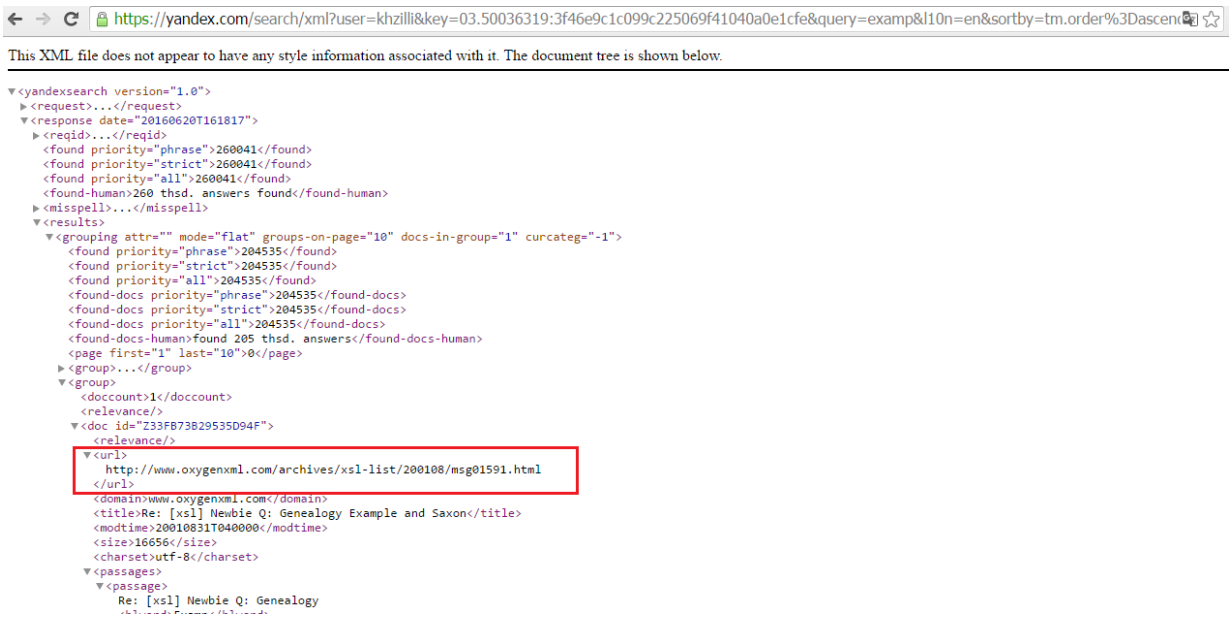

Рисунок 2.6.3 - Результаты выдачи поискового запроса Yandex в формате XML.

2.3 Построение сети распространения

Имеем набор некоторых ссылок, соответствующий заданному набору слов. Необходимо построить сеть (граф), по которой распространялась новость. Для этого загрузим страницы, соответствующие каждой ссылке и извлечем для каждой страницы все ссылки, которые в ней содержатся. По этим данным построим сеть распространения: для каждой ссылки укажем список источников, где упоминалась рассматриваемая страница. Тогда получим ориентированный граф $G=(V, E)$, где V – множество рассматриваемых сайтов (вершин), а E – множество ссылок одного источника на другой.

Теперь необходимо определить дату публикации новости на каждой странице. Так как данная задача нетривиальна, и нет способа, позволяющего точно извлекать информацию о каждой странице, рассмотрим HTML-структуру некоторых социальных сетей, чтобы точно извлекать из них дату создания страницы. Для остальных сайтов будем применять поиск при помощи регулярных выражений.

2.4 Структура представления данных в социальной сети Facebook

Поисковая система может выдать результат запроса для Facebook в двух видах: либо в виде ссылки на пользователя и его ленту, где находится данная новость, либо на непосредственно запись.

Рассмотрим структуру каждого из этих документов (см. рис. 2.8.1).

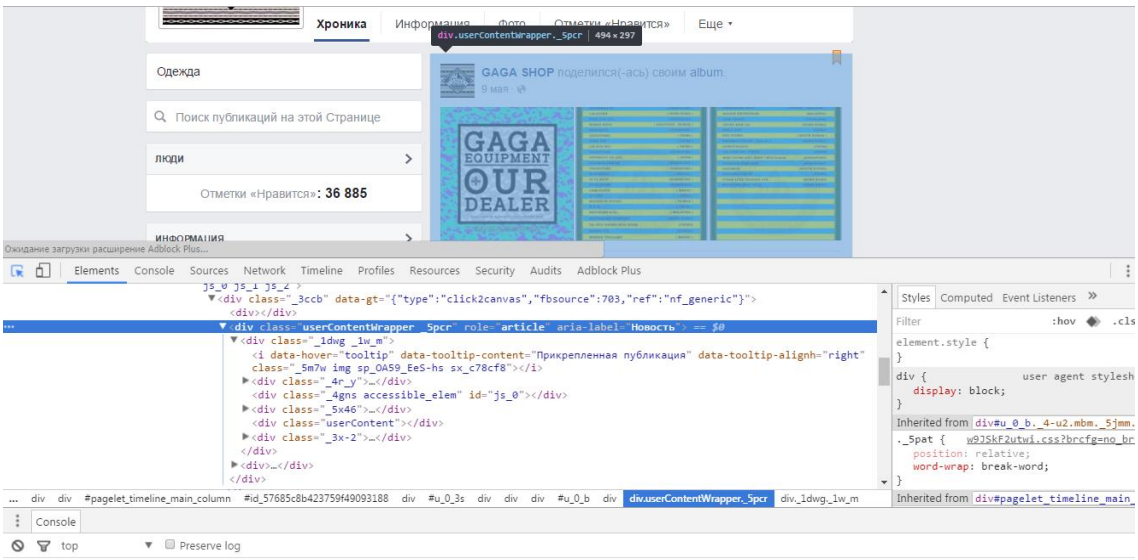

Рисунок 2.8.1 - Разметка страницы Facebok.

В новостной ленте каждая новость обернута в контейнер “div.userContentWrapper _5pcr”. Сначала определим, удовлетворяет ли запись в этом элементе поисковой строке, то есть потребуем точного совпадения некоторой подстроки в контейнере с искомой строкой

Проанализировав содержимое этого контейнера, обнаружим, что информация о дате публикации записи хранится в контейнере “span.fsm fwn fcg” (рис. 2.8.2) в свойстве title тега <abbr>. Оттуда извлечем информацию о дате публикации страницы.

Рисунок 2.8.2 - Местонахождение информации о дате создания записи

2.5 Структура представления данных в социальной сети ВКонтакте

Проанализировав HTML-код страницы ВКонтакте, мы обнаружили, что каждая запись находится в контейнере div.post_table (см. рис. 2.8.3), где находится непосредственно содержание записи и информация о публикации. Дата публикации записи находится в контейнере span.rel_date. Откуда, далее, будет извлечена требуемая дата.

Особенность представления данных на этом сайте заключается в употреблении таких слов как «вчера», «сегодня», «несколько минут назад» для обозначения недавно опубликованный информации. В связи с этим при разборе строки нужно помимо классического представления дат уметь анализировать дату создания страниц для подобных форматов.

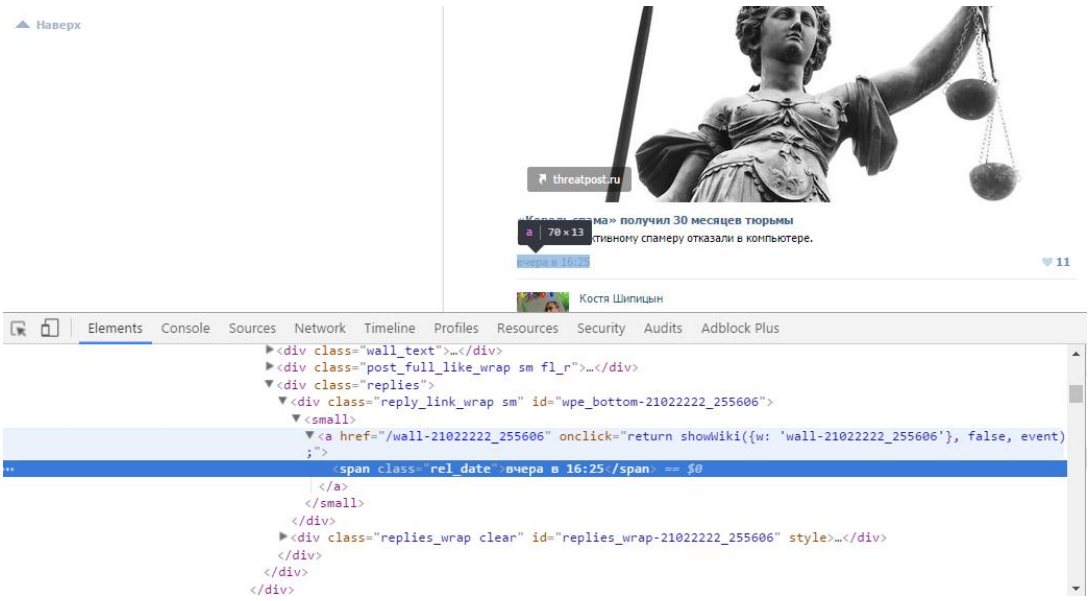

Рисунок 2.8.3 - Структура документа сайта ВКонтакте.

2.6 Определение даты публикации новости

На данный момент не существует точного метода определения даты публикации (создания) веб-страницы. Для статических страниц можно запросить дату создания/модификации у сервера, на котором данная страница хранится. Но, так как большинство страниц сегодня являются динамическими и создаются в момент их запроса, нет простого способа определения даты модификации/создания документа.

Также трудности порождает тот факт, что дата публикации находится в различных местах на различных страницах, не существует единого формата и стиля описания.

Даты в интернет могут быть представлены различными способами. Их поиск и распознавание облегчает такой инструмент, как регулярные выражения.

Приведем форматы представления дат и соответствующие регулярные выражения для заданного формата. Так как распространение данных рассматривается по дням, а часы не учитываются, будем учитывать только ту часть строки, где представлена дата.

В данной работе были рассмотрены те форматы представления дат, которые чаще всего используются для указания даты публикации новости (см. Таблица 1).

Такой метод не дает 100% результата, но в большинстве случаев мы получаем корректный результат.

Корректность результата можно проверить, проанализировав соседей вершины в сети распространения.

Таблица 1. Форматы представления дат и регулярные выражения

Формат представления даты	Регулярное выражение
dd.mm.yyyy	(0?[1-9]|[12][0-9]|3[01])[\.\,\\/](0?[1-9]|1[012])[\.\,\\/]((19|20)\\d\\d)
dd-mm-yyyy	
dd/mm/yyyy	
mm.dd.yyyy	(0?[1-9]|1[012])[\.\,\\/](0?[1-9]|[12][0-9]|3[01])[\.\,\\/]((19|20)\\d\\d)
mm-dd-yyyy	
mm/dd/yyyy	
yyyy.mm.dd	((19|20)\\d\\d)[\.\,\\/](0?[1-9]|1[012])[\.\,\\/](0?[1-9]|[12][0-9]|3[01])
yyyy-mm-dd	
yyyy/mm/dd	

Как видно из таблицы, для форматов 1-6 нет возможности определить, где находится день, а где месяц. Поэтому примем, что сначала указывается день, а потом месяц.

Для некоторых новостей невозможно определить дату публикации. Поэтому, для установки даты публикации мы проанализируем все источники, который ссылаются на данный. Среди них выберем источник, который появился в Сети раньше всех и эту дату зададим как дату публикации рассматриваемой новости. Если не удалось найти такую дату, то попробуем найти источники, которые сами ссылаются на данный и необходимые данные извлечем из них.

2.7 Интерфейс пользователя

Интерфейс пользователя программы имеет следующий вид (см. рис. 2.7.1).

В текстовое поле вводится новость, которую мы хотим проанализировать. После нажатия на кнопку «Запуск» на панели под текстовым полем появится изображение графа распространения информации. Полученный граф сети распространения и информация о вершинах будет сохранена в текстовый файл lastSession.txt.

Так как сбор данных и построение графа являются довольно трудоемким процессом, была добавлена кнопка «Загрузить предыдущий», которая позволяет загрузить из файла lastSession.txt последний рассмотренный граф. Если файл с записанным сообщением не найден, то программа выведет соответствующее сообщение (см. рис. 2.7.2).

Обычно распространители вбросов распространяют один и тот же текст. То есть, если в их публикациях встретилась некоторая последовательность слов, то она будет встречаться и в остальных. Следовательно, мы будем анализировать те запросы, в которых наш поисковый запрос (новость, которую мы хотим проанализировать) содержится в таком же виде, в каком мы его задаем.

В поле под заголовком «Интенсивность распространения информации» будет изображена сеть распространения информации, на которой указано, с какой интенсивностью в обнаруженные дни распространилась новость.

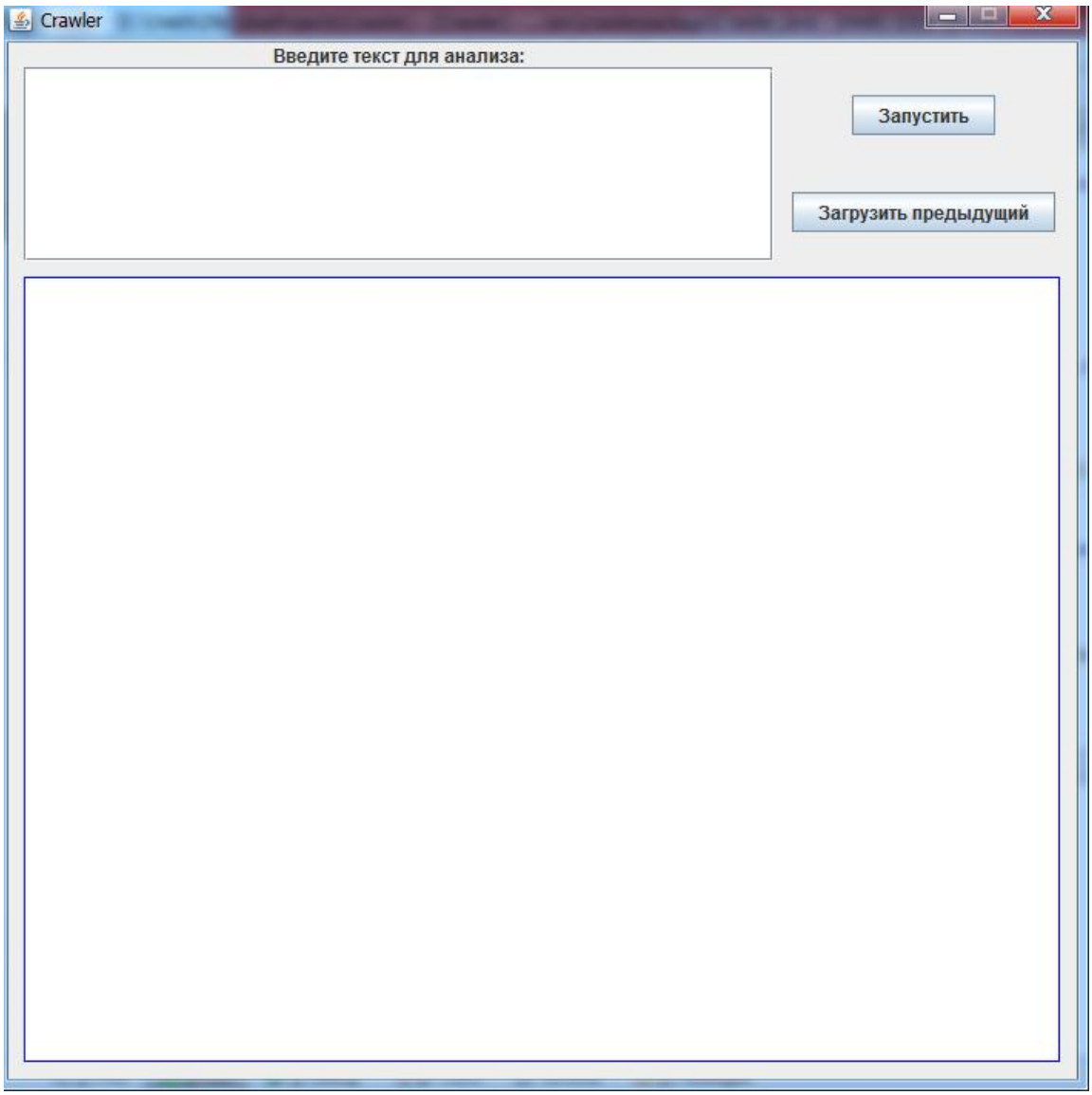

Рисунок 2.7.1 - Интерфейс пользователя.

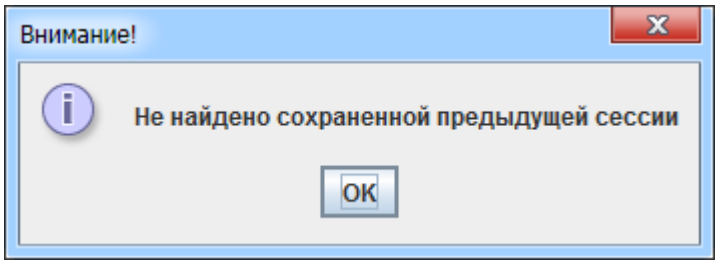

Рисунок 2.7.2 - Сообщение об отсутствии файла для загрузки графа

2.8 Описание архитектуры системы. Диаграмма классов

Одной из важнейших частей разработки приложения является проектирование архитектуры системы, описания связи компонентов системы друг с другом. Предварительный анализ системы и разработка диаграммы классов позволяет облегчить процесс разработки, так как программисту не приходится думать о том, как добавить ту или иную функциональность, не нарушая работоспособность кода, написанного ранее.

Диаграмма классов позволяет описать систему в терминах Объектно-Ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы.

Диаграмма классов представляет собой некоторый граф, вершинами которого являются элементы типа "классификатор", которые связаны различными типами структурных отношений. Следует заметить, что диаграмма классов может также содержать интерфейсы, пакеты, отношения и даже отдельные экземпляры, такие как объекты и связи. Когда говорят о данной диаграмме, имеют в виду статическую структурную модель проектируемой системы. Поэтому диаграмму классов принято считать графическим представлением таких структурных взаимосвязей логической модели системы, которые не зависят или инвариантны от времени.

Диаграмма классов состоит из множества элементов, которые в совокупности отражают декларативные знания о предметной области. Эти знания интерпретируются в базовых понятиях языка UML, таких как классы, интерфейсы и отношения между ними и их составляющими компонентами. При этом отдельные компоненты этой диаграммы могут образовывать пакеты для представления более общей модели системы. Если диаграмма классов является частью некоторого пакета, то ее компоненты должны соответствовать элементам этого пакета, включая возможные ссылки на элементы из других пакетов.

Для разрабатываемой системы предлагается следующая архитектура (см. рис. 2.9.1). Рассмотрим каждый класс системы подробнее.

Класс Crawler - инициирует главное окно приложения и обрабатывает события нажатия кнопки мыши.

Поля класса:

String lastQuery – в переменной храним строку последнего рассмотренного поискового запроса.

JFrame frame – главное окно приложения.

JTextArea textArea – текстовое поле для ввода поискового запроса.

JButton run – кнопка запуска анализа поискового запроса.

JButton loadPrevResult – кнопка для загрузки сети распространения последнего рассмотренного запроса.

Visualisation panel – панель визуализации результатов.

DiffusionNetwork diffusionNetwork – сеть распространения информации.

Методы:

Crawler() – конструктор класса. В данном методе создаются элементы окна приложения.

void analyse(String query) – анализ поискового запроса query.

void visualise(DiffusionNetwork diffusionNetwork) – нарисовать сеть распространения diffusionNetwork.

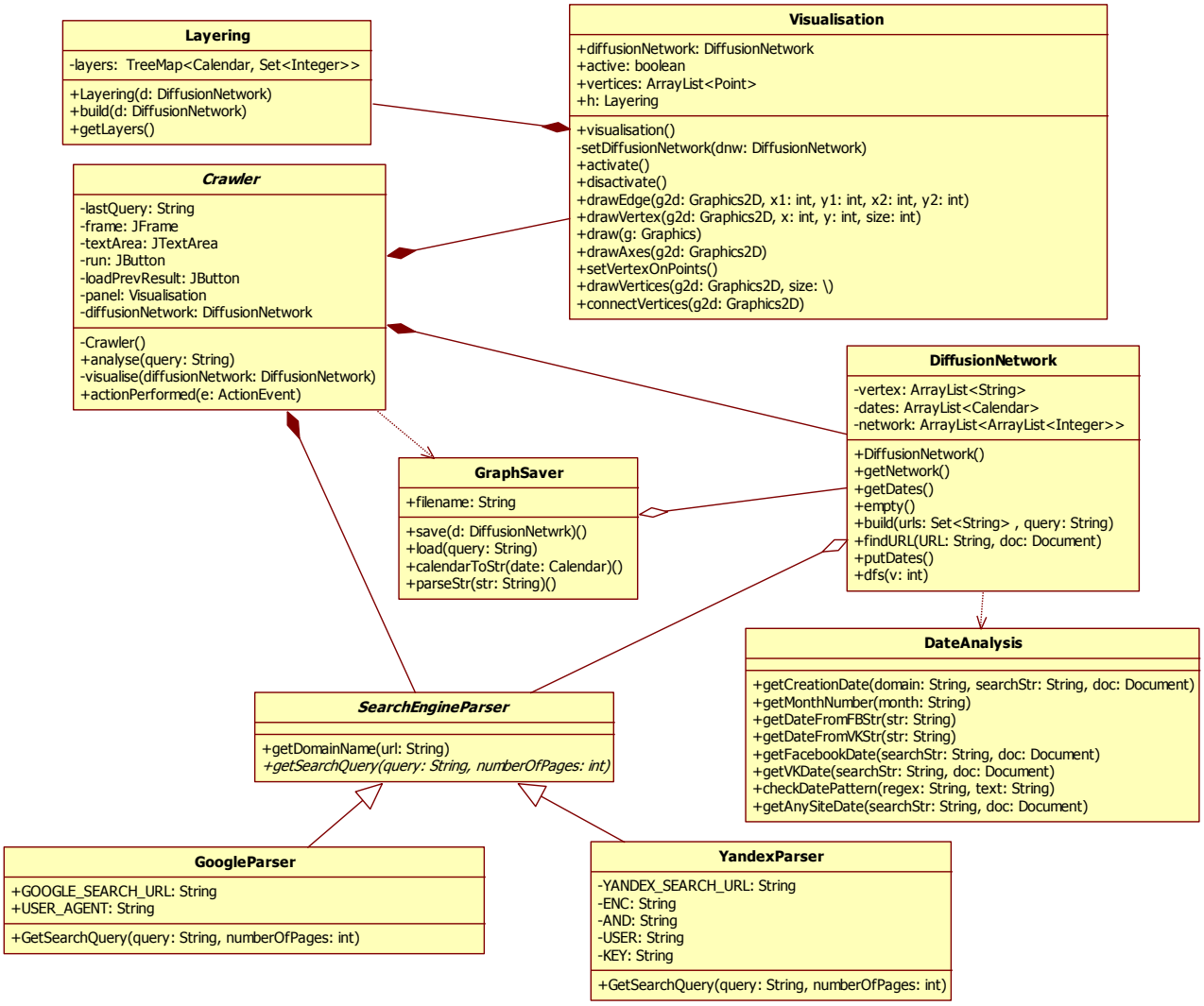

Рисунок 2.9.1 - Диаграмма классов системы.

public void actionPerformed(ActionEvent e) – обработчик событий.

Класс DateAnalysis – класс для работы с датами. Предоставляет методы для оценки даты создания страниц. Все методы этого класса, если не могут установить дату создания страницы, возвращают значение null.

public static Calendar getCreationDate(String domain, String searchStr, Document doc) – получить дату создания страницы doc с доменным именем domain для новости searchStr.

public static int getMonthNumber(String month) – возвращает порядковый номер месяца month.

static Calendar getDateFromFBStr(String str) – извлекает дату из строки str в формате, используемом на страничке Facebook.

static Calendar getDateFromVKStr(String str) – извлекает дату из строки str в формате, используемом на страничке ВКонтакте.

public static Calendar getFacebookDate(String searchStr, Document doc) – извлекает дату публикации новости на Facebook.

public static Calendar getVKDate(String searchStr, Document doc) - извлекает дату публикации новости на ВКонтакте.

private static Matcher checkDatePattern(String regex, String text) – проверяет, соответствует ли строка text регулярному выражению regex.

public static Calendar getAnySiteDate(String searchStr, Document doc) – предположить дату создания сайта по данным в документе doc.

Класс DiffusionNetwork – класс сети распространения. Объекты данного класса представляют собой сеть распространения информации для некоторого поискового запроса.

Поля класса:

ArrayList<String> vertex – список ссылок, соответствующих вершинам графа. Индекс соответствующего элемента соответствует номеру вершины.

ArrayList<Calendar> dates – даты создания страниц.

ArrayList<ArrayList<Integer>> network – сеть распространения, представленная в виде списков смежности.

public static final String USER_AGENT – названия браузеров, чье поведение должна эмитировать Jsoup для получения веб-страницы.

ArrayList<ArrayList<Integer>> getNetwork() – метод позволяет получить сеть распространения.

ArrayList<Calendar> getDates() – получить список дат публикация новостей.

public boolean empty() – если граф пуст, возвращает true, иначе – false.

public DiffusionNetwork() – конструктор класса.

public void Build(Set<String> urls, String query) построить сеть распространения запроса query по набору ссылок urls.

public boolean findURL(String URL, Document doc) – ищем, есть ли в документе doc ссылка на сайт URL.

void putDates() – метод, который пробует определить дату создания для тех страниц, у которых это значение не определено.

Calendar dfs(Integer v) – алгоритм обхода в ширину. Используется при расстановке дат по значениям в соседних вершинах.

Класс SearchEngineParser – абстрактный класс, определяющий методы для реализации обработки результатов поисковых запросов.

public static String GetDomainName(String url) – получить доменное имя сайта url.

public abstract Set<String> GetSearchQuery(String query, int numberOfPages) – абстрактный метод, реализуемый в потомках класса SearchEngineParser. Возвращает список ссылок, выданных поисковой системой для запроса query. Сбор берется по первым numberOfPages страницам результатов выдачи.

Класс GoogleParser – реализация класса SearchEngineParser для поисковой системы Google.

public static final String *GOOGLE_SEARCH_URL* – начальная часть адресной строки поискового запроса Google.

public Set<String> GetSearchQuery(String query, int numberOfPages) - Возвращает список ссылок, выданных поисковой системой для запроса query. Сбор берется по первым numberOfPages страницам результатов выдачи.

Класс YandexParser – реализация класса SearchEngineParser для поисковой системы Yandex.

private static final String *YANDEX_SEARCH_URL* начальная часть адресной строки поискового запроса Yandex.

private static final String ENC – кодировка, в которой нужно задать поисковых запрос.

private static final String AND – логическая операция И.

private static final String USER – имя пользователя ЯндексXML.

private static final String KEY – ключ доступа к ЯндексXML

Класс GraphSaver – используется для сохранения и чтения из файла некоторого графа.

Поля класса:

static final String filename - имя файла, куда сохраняется граф.

Методы:

public static boolean save(DiffusionNetwork d, String query) – функция, записывающая граф сети распространения d для поискового запроса query в файл filename. Если удалось записать в файл, то метод возвращает значение true, иначе - false.

public static String CalendarToStr(Calendar date) – преобразует объект класса Calendar в строку вида dd/mm/yyyy.

public static Calendar parseString(String str) – восстанавливает из строки вида dd/mm/yyyy объект класса Calendar.

public static DiffusionNetwork load() – считывает и возвращает граф сети распространения из файла filename. В случае неудачи возвращает значение null.

Класс Layering. Данный класс для графа сети распространения выделяет слои, то есть для каждой даты определяет список источников, которые были упомянуты в этот день. То есть, один слой – это один день.

Поля класса:

TreeMap<Calendar, Set<Integer>> layers – массив, в котором представлено разбиение на слои некоторой сети распространения.

int undefinedValues – количество вершин, для которых дата публикации не определена.

Методы:

public Layering(DiffusionNetwork d) – создать объект класса Layering для сети распространения d.

public void build(DiffusionNetwork d) – построить слои для графа d.

public Map<Calendar, Set<Integer>> getLayers() – возвращает разбиение Layering.

Класс Visualisation – данный класс занимается отрисовкой сети распространения на экране.

private static final float BORDER = 20 значение смещения от краев области рисования в пикселях.

private static final int VERTEXSIZE = 10 – ширина одной вершины на экране.

private float cellW, cellH – максимальная ширина и высота, выделяемые элементы на экране.

DiffusionNetwork diffusionNetwork – рисуемая сеть распространения.

ArrayList<Point> vertices – Координаты точек каждой вершины сети рапспространения.

Layering h – разбиение сети распространения на слои.

private boolean active – если значение переменной равно true, значит панель для рисования активна, и мы может на ней рисовать, иначе – нельзя.

public Visualisation() – конструктор класса.

public void setDiffusionNetwork(DiffusionNetwork dnw) – функция позволяет задать сеть распространения.

public void activate() – «включить» панель для рисования.

public void deactivate() – «выключить» панель для рисования.

void drawEdge(Graphics2D g, int x1, int y1, int x2, int y2) – нарисовать ребро в графическом контексте g от точки с координатами (x1, y1) до точки с координатами (x2, y2).

void drawVertex(Graphics2D g2d, int x, int y, int size) – нарисовать вершину размером size в графическом контексте g2d в точке с координатами (x, y).

void drawAxes(Graphics2D g2d) – нарисовать оси координат.

void setVertexOnPoints() – подсчитать координаты точек вершин с учетом разбиения по слоям.

void drawVertices(Graphics2D g2d, float size) – метод отрисовки вершин с заданным размером size.

void connectVertices(Graphics2D g2d) – метод отрисовки всех ребер графа.

public void draw(Graphics g) – рисуем сеть распространения в графическом контексте g.

Выводы по разделу

В данном разделе приведены результаты анализа структуры представления результатов выдачи поисковых запросов поисковых систем Google и Yandex, также рассмотрены структуры представления новостей в социальных сетях Facebook и ВКонтакте. Полученная информация будет использована для извлечения результатов-ссылок, на основе которых в дальнейшем будет построен граф сети распространения информации. Информация о структуре представления новостей в социальных сетях позволяет точно определить дату публикации новости.

Также в данном разделе приведен интерфейс пользователя программы и также описана архитектура системы.

2 АЛГОРИТМЫ

В данной главе будут рассмотрены некоторые алгоритмы, используемые в программе.

3.1 Алгоритм определения даты создания веб-страницы.

Для определения даты создания веб-страницы выгрузим ее содержимое по соответствующей ссылке и проанализируем полученный текст при помощ регулярных выражений. Для некоторых социальных сетей можно получить точное местоположение информации о дате, следовательно, для них разработаем отдельные методы определения даты. На рисунке 3.1.1 приведена схема полученного алгоритма.

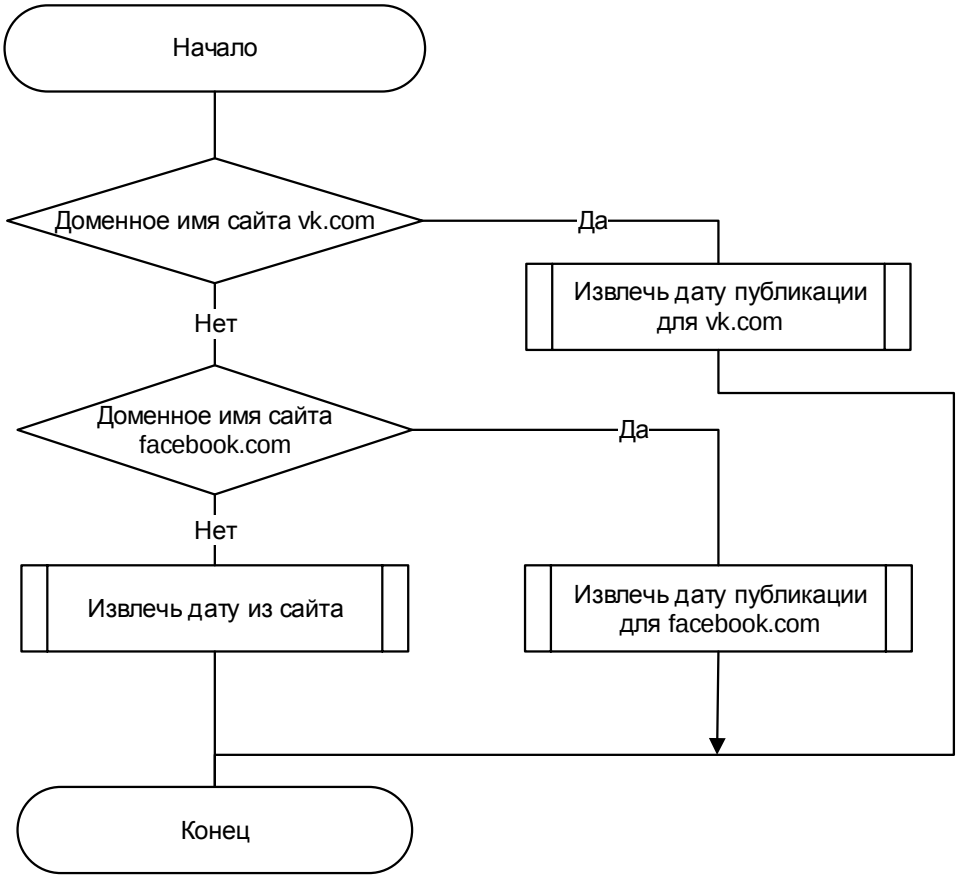

Рисунок 3.1.1 - Схема алгоритма подпрограммы определения даты публикации новости на странице.

3.2 Алгоритм построения сети распространения

Обработав результаты выдачи поисковых систем мы получили некоторый набор ссылок. Построим по этому набору ссылок ориентированный граф сети распространения информации. Для этого выгрузим содержимое каждой из страниц и проанализируем соответствующий набор ссылок. На рисунке 3.2.1 – 3.2.2 приведен граф алгоритма соответствующей подпрограммы:

Рисунок 3.2.1 – Схема алгоритма подпрограммы построения сети распространения новости.

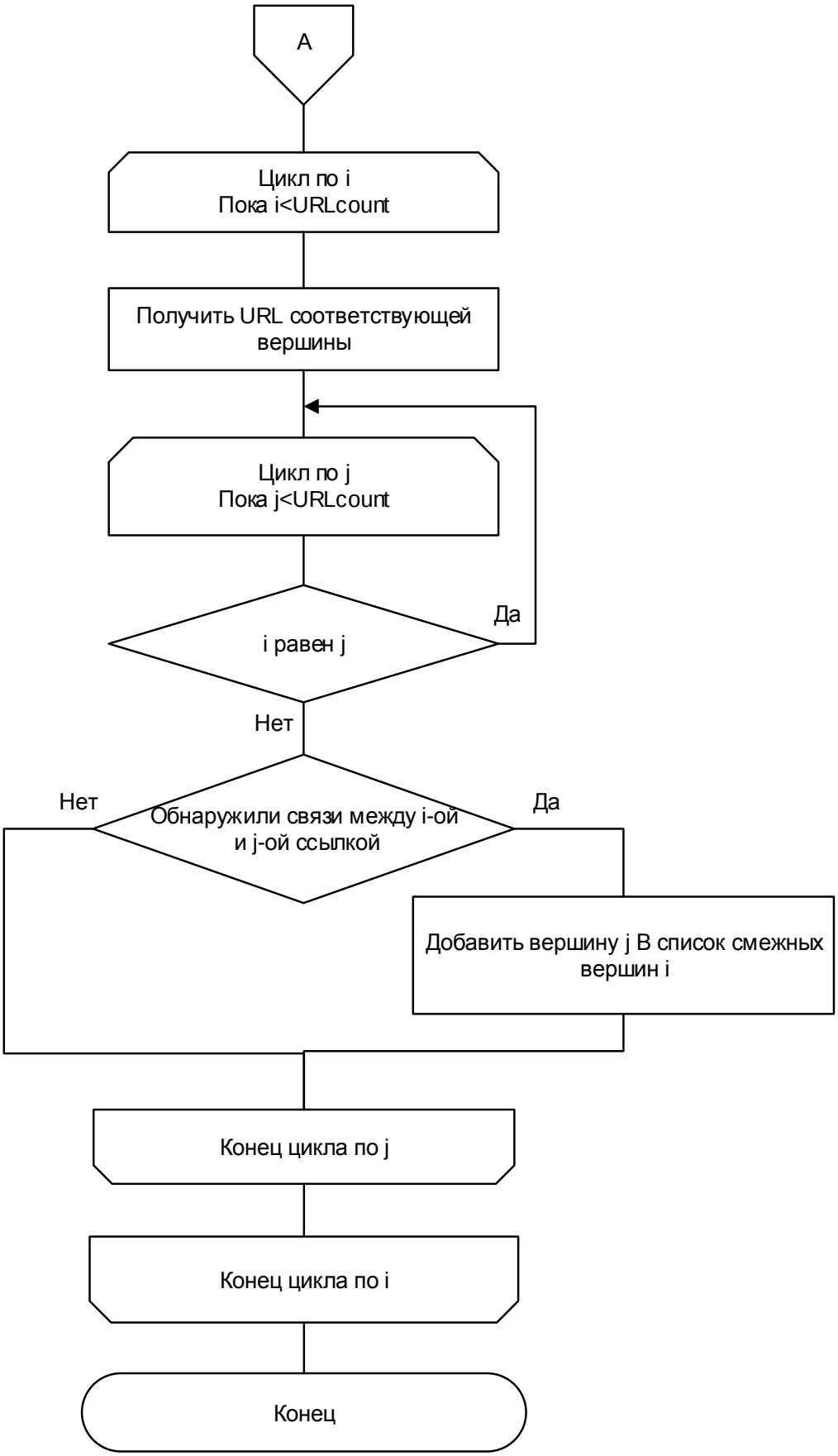

Рисунок 3.2.1 – Схема алгоритма подпрограммы построения сети распространения новости (продолжение).

3.3 Преобразование дерева сети распространения для вывода на экран

Приведем блок-схему алгоритма визуализации полученной сети распространения (рис 3.3.1). Сначала разобьем сеть распространения на слои по датам, когда каждая новость была опубликована. Далее расставим вершины графа по области рисования. Таким образом, зная координаты вершин, нарисуем ребра графа и его вершины.

Рисунок 3.3.1 – Схема алгоритма подпрограммы визуализации сети распространения

Выводы по разделу

Проанализировав особенности представления данных на различных Интернет-ресурсах, были разработаны алгоритмы, которые в большинстве случаев правильно определяют дату публикации новости на странице.

Также был разработан алгоритм построения сети распространения, основанный на методе полного перебора, что позволяет проверить все возможные связи внутри имеющегося набора ссылок.

4 РЕЗУЛЬТАТЫ

Для тестирования программы выбраны 3 темы, которые недавно обсуждались в Сети. Для этого была выбрана некоторая новость о Путине, так как недавно в СМИ упоминалось, что против него был выполнен информационный вброс, новость о Чемпионате Европы по футболу и слух с сайта http://perebezhchik.ru/, где публикуются различные слухи, циркулирующие в русскоязычном интернете.

Рассмотрим каждый пример подробнее.

1) *Информационный вброс.* Текст новости: «Люди из окружения Путина вывели через офшоры более 2 млрд долларов». Результат работы программы приведен на рисунке 4.1.

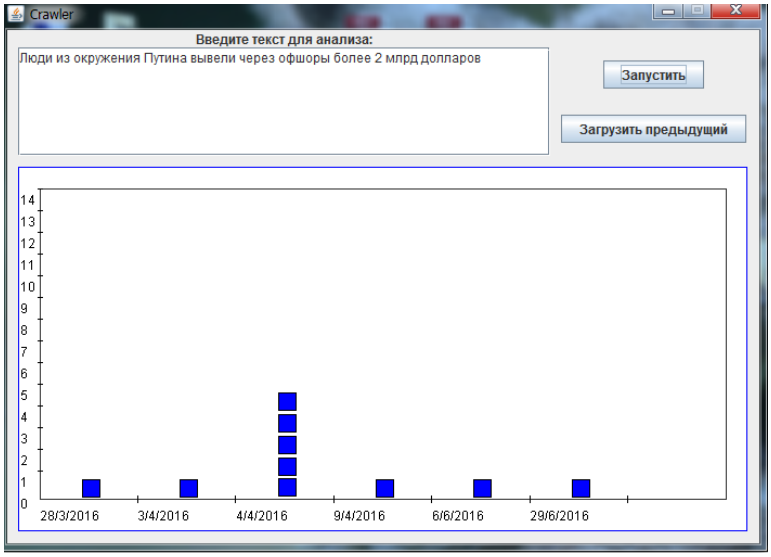

Рисунок 4.1 - информационный вброс о Путине

2) Естественная новость: «Сборная России вылетела с Евро-2016, заняв последнее место в группе В»

Как видим на рисунке 4.2:

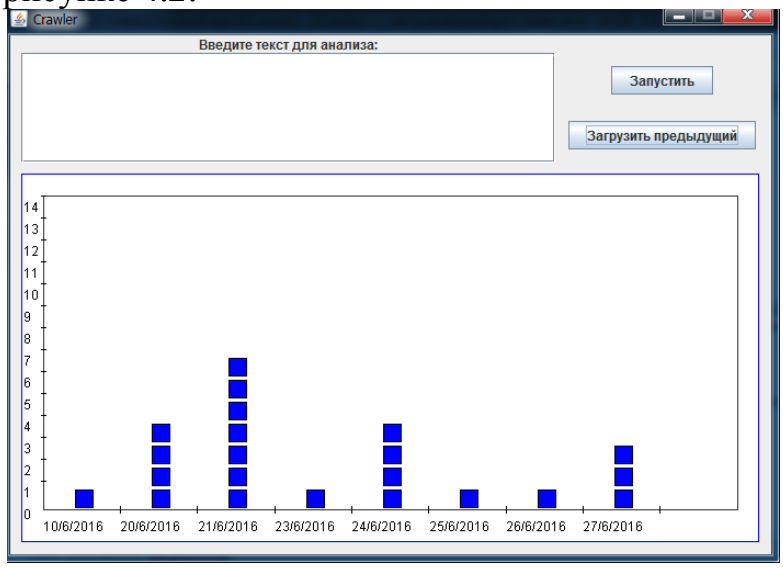

Рисунок 4.2 - Сеть распространения для естественной новости.

3) Слух. Также рассмотрим пример распространения слуха в Сети. Текст новости имеет следующий вид: «ВКонтакте опровергла информацию СМИ о взломе 100 млн. учетных записей пользователей».

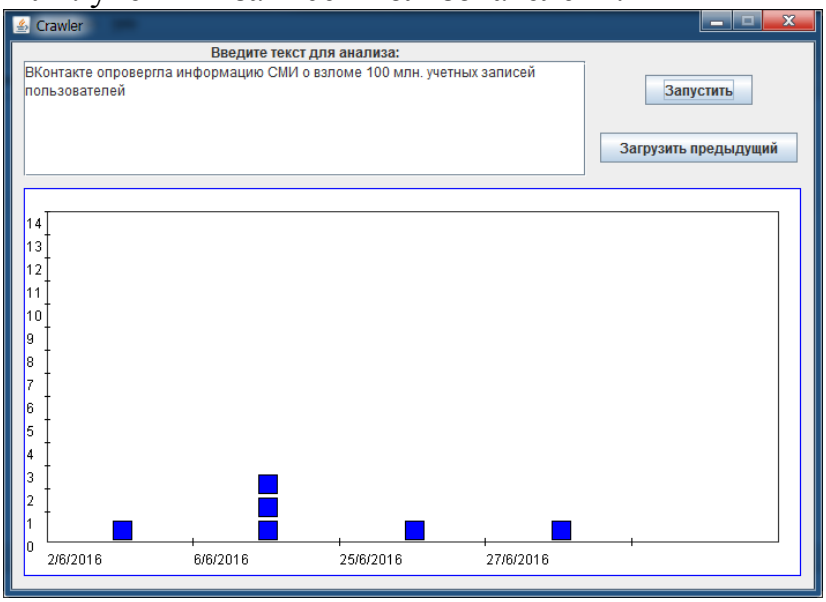

Рисунок 5 Результат работы программы для слуха

ЗАКЛЮЧЕНИЕ

С распространением Интернета определение того, является ли новость информационном слухом, вбросом, или естественной новостью, особенно важна. Существует множество работ, в которых анализируют динамику распространения информации, но они привязаны к конкретным платформам и социальным сетям, и ни одна из низ не проводит анализ выдачи поисковых систем. Данная работа предоставит простой инструмент для визуализации структуры графа распространения новости, что может помочь определить класс новости.

В разделе 2 приведены результаты анализа структуры представления результатов выдачи поисковых запросов поисковых систем Google и Yandex, также рассмотрены структуры представления новостей в социальных сетях Facebook и ВКонтакте. Полученная информация будет использована для извлечения результатов-ссылок, на основе которых в дальнейшем будет построен граф сети распространения информации. Информация о структуре представления новостей в социальных сетях позволяет точно определить дату публикации новости, что позволит определить дату публикации новости для смежных вершин(сайтов) с данной, дата создания которых неизвестна.

В ходе работы был разработан интерфейс пользователя программы. Также была разработана архитектура системы с применением объектно-ориентированного подхода.

В результате работы программы была получена схема распространения информации. Наличие большого числа одиночных вершин и отсутствие длинных цепочек распространения объясняется малым размером исследуемой выборки и тем, что авторы крупных порталов, которые обычно находятся в первых строчках результатов выдачи поисковых запросов, редко ссылаются на другие источники и сами генерируют контент.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК.

1. Латынов, В. В. Слухи: социальные функции и условия появления / В. В. Латынов. – URL: http://ecsocman.hse.ru/data/115/920/1219/002.LATINOV.pdf (дата обращения 1.06.2016)
2. Петросян А. Э. В паутине Фомы/А. Э. Петросян. - URL: http://www.ins-car.ru/Articles/Fama.pdf(дата обращения 1.06.2016)
3. Sejeong Kwon Aspects of Rumor Spreading on a Microblog Network / Sejeong Kwon, Meeyoung Cha, Kyomin Jung at al. – URL: http://milab.snu.ac.kr/pub/Socinfo2013.pdf (дата обращения 1.06.2016)
4. Sejeong Kwon Prominent Features of Rumor Propagation in Online Social Media /Sejeong Kwon, Meeyoung Cha, Kyomin Jung at al. - URL: http://milab.snu.ac.kr/pub/ICDM2013.pdf (дата обращения 1.06.2016)
5. Адерихин, А. Третья мировая война в областном масштабе: Как боролись со слухами в послевоенной Калининградской области и как слухи боролись с Калининградской областью. Рассказывают когда-то строго секретные документы из Гос. архива Калининградской области /А. Адерихин- URL:http://rugrad.eu/communication/blogs/Konigsberg_archive/8663/ (дата обращения: 20.06.2016)
6. Sejeong Kwon: Prominent Features of Rumor Propagation in Online Social Media / Sejeong Kwon, Meeyoung Cha, Kyomin Jung, Wei Chen at al/ - URL: http://mia.kaist.ac.kr/publications/rumor/ (дата обращения: 1.06.2016)
7. Yasuko Matsubara Rise and Fall Patterns of Information Diffusion: Model and Implications / Yasuko Matsubara, Yasushi Sakurai, B. Aditya Prakash et al. – URL: http://www.cs.cmu.edu/~badityap/papers/risefall-spikem-kdd12.pdf (дата обращения: 1.06.2016)
8. Observatory on Social Media. - URL: http://truthy.indiana.edu/ (дата обращения: 1.06.2016)
9. IU data scientists launch free tools to analyze online trends, memes. - URL: http://news.indiana.edu/releases/iu/2016/05/observatory-on-social-media.shtml (дата обращения: 1.06.2016)
10. Emergent. - URl: http://www.emergent.info (дата обращения: 1.06.2016)
11. Karyne Levy, This Site Wants To Debunk Internet Rumors Before They Can Even Start To Spread, URL: http://www.businessinsider.com/emergent-tracks-and-debunks-rumors-2014-10 (дата обращения: 1.06.2016)
12. Ашманов, И. Про мозговые вирусы / И. Ашманов. - URL: https://roem.ru/07-08-2014/109742/pro-mozgovye-virusy/ (дата обращения: 1.06.2016)
13. Под властью ментального вируса, или как отличить вброс от новости. - URL: http://www.pravmir.ru/pod-vlastyu-mentalnogo-virusa-ili-kak-otlichit-vbros-ot-novosti/ (дата обращения: 1.06.2016)
14. Разработан алгоритм, позволяющий вычислять источники эпидемий. - URL: http://www.computerra.ru/36627/razrabotan-algoritm-pozvolyayushhiy-vyich/ (дата обращения: 2.06.2016)
15. Pinto, Pedro C. Locating the Source of Diffusion in Large-Scale Networks / Pedro C. Pinto, Patrick Thiran, Martin Vetterli. - URL:

http://www.pedropinto.org.s3.amazonaws.com/publications/locating_source_diffusion_networks.pdf (дата обращения: 2.06.2016)
16. jsoup: Java HTML Parser. - URL: https://jsoup.org/ (дата обращения: 1.06.2016)
17. Леоненков, А. Самоучитель UML/А. Леоненков. – Санкт-Петербург. БХВ-Петербург, 2007. – 372 с.
18. Шилдт, Г. Java 8. Руководство для начинающих. Шестое издание/ Г. Шилдт – Москва, СПб, Киев. Издательский дом «Вильямс», 2015. – 712.
19. Хорстманн, К. Java 2. Том 1. Основы/ К. Хорстманн, Г. Корнелл - Москва, СПб, Киев. Издательский дом «Вильямс», 2012. – 712.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1. Текст программы.

Модуль Crawler.

```
import crawlerpackage.*;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.net.URISyntaxException;
import java.util.HashSet;
import java.util.Set;

/**
 * Created by Zily on 01.03.2016.
 */
class Crawler implements ActionListener {

    String lastQuery;
    JFrame frame;
    JTextArea textArea;
    JButton run, loadPrevResult;
    Visualisation panel;

    DiffusionNetwork diffusionNetwork;

    Crawler(){
        lastQuery = new String();
        frame = new JFrame("Crawler");
        frame.setSize(700, 700);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(new GridBagLayout());
        frame.setResizable(false);


        GridBagConstraints constraints = new GridBagConstraints();
        constraints.weightx = 0;
        constraints.weighty = 0;
        constraints.gridx = 0;
        constraints.gridy = 0;
        constraints.gridwidth = 2;
        constraints.gridheight = 1;

        JLabel label = new JLabel("Введите текст для анализа:");
        frame.add(label, constraints);

        run = new JButton("Запустить");
        run.addActionListener(this);
        constraints.insets.right = 10;
        constraints.weightx = 50;
        constraints.weighty = 5;
        constraints.gridx = 2;
        constraints.gridy = 1;
        constraints.gridwidth = 1;
        constraints.gridheight = 1;
        frame.add(run, constraints);
```

```
        run = new JButton("Загрузить предыдущий");
        run.addActionListener(this);
        constraints.insets.right = 10;
        constraints.weightx = 50;
        constraints.weighty = 5;
        constraints.gridx = 2;
        constraints.gridy = 2;
        constraints.gridwidth = 1;
        constraints.gridheight = 1;
        frame.add(run, constraints);

        textArea = new JTextArea();
        textArea.setLineWrap(true);
        textArea.setWrapStyleWord(true);

        JScrollPane scrollPane = new JScrollPane(textArea);
        //scrollPane.setPreferredSize(new Dimension(200, 100));
        constraints.insets.left = 10;
        constraints.weightx = 150;
        constraints.weighty = 15;
        constraints.gridx = 1;
        constraints.gridy = 1;
        constraints.gridwidth = 1;
        constraints.gridheight = 2;
        constraints.fill = GridBagConstraints.BOTH;
        frame.add(scrollPane, constraints);

        panel = new Visualisation();
        panel.setBorder(BorderFactory.createLineBorder(Color.BLUE));
        panel.setBackground(Color.WHITE);
        constraints.insets.top = constraints.insets.bottom =
constraints.insets.left = constraints.insets.right = 10;
        constraints.weightx = 100;
        constraints.weighty = 100;
        constraints.gridx = 1;
        constraints.gridy = 3;
        constraints.gridwidth = 2;
        constraints.gridheight = 1;
        constraints.fill = GridBagConstraints.BOTH;
        frame.add(panel, constraints);

        frame.setVisible(true);
//        frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
    }

    public static void main(String[] args) throws IOException{
        SwingUtilities.invokeLater(
                new Runnable() {
                    @Override
                    public void run() {
                        new Crawler();
                    }
                }
        );
    }

    void analyse(String query){
        lastQuery = query;
        Set<String> URLs = new HashSet<String>();
        SearchEngineParser google = new GoogleParser();
        SearchEngineParser yandex = new YandexParser();
        try {
```

```
            URLs.addAll(google.GetSearchQuery(query, 20));
        } catch (IOException e1) {
            e1.printStackTrace();
        }
        try {
            URLs.addAll(yandex.GetSearchQuery(query, 5));
        } catch (IOException e1) {
            e1.printStackTrace();
        }
        diffusionNetwork = new DiffusionNetwork();
        try {
            diffusionNetwork.Build(URLs, query);
        } catch (IOException e1) {
            e1.printStackTrace();
        } catch (URISyntaxException e1) {
            e1.printStackTrace();
        }
        GraphSaver.save(diffusionNetwork, query);
        this.visualise(diffusionNetwork);
    }

    void visualise(DiffusionNetwork diffusionNetwork){
        try {
            panel.setDiffusionNetwork(diffusionNetwork);
            panel.activate();
            panel.repaint();
        } catch (NullPointerException ex){
            ex.printStackTrace();
        }
    }
    @Override
    public void actionPerformed(ActionEvent e){
        if (e.getActionCommand() == "Запустить") {
            String query = textArea.getText();
            if (query.isEmpty() || lastQuery.equals(query)){
                return;
            }
            this.analyse(query);
            GraphSaver.save(diffusionNetwork, query);
        } else if (e.getActionCommand()=="Загрузить предыдущий") {
            String str = new String();
            DiffusionNetwork temp = GraphSaver.load(str);
            if (temp == null) {
                JOptionPane.showMessageDialog(frame,
                        "Не найдено сохраненной предыдущей сессии", "Внимание!",
                        JOptionPane.INFORMATION_MESSAGE);
            } else {
                diffusionNetwork = temp;
                visualise(temp);
            }

        }
    }
}
```

Модуль DateAnalysis.

```
package crawlerpackage;

import org.joda.time.Days;
import org.joda.time.LocalDate;
import org.jsoup.nodes.Document;
```

```
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class DateAnalysis {
    public static final String FACEBOOK_CONTENT_ELEM_CLASS = "userContentWrapper
_5pcr";
    public static final String SPAN_CLASS_FB = "fsm fwn fcg";
    public static final String[] months = {"января", "февраля", "марта", "апреля",
"мая", "июня", "июля",
    "августа", "сентября", "октября", "ноября", "декабря"};

    static final String DELIMITER = "[-:\\/.,]";
    static final String DAY = "(0?[1-9]|[12][0-9]|3[01])";
    static final String MONTH = "(0?[1-9]|1[0-2])";
    static final String YEAR = "([12]\\d\\d\\d)";

    public static Calendar getCreationDate(String domain, String searchStr,
Document doc){
        if (domain.equals("vk.com")){
            return getVKDate(searchStr, doc);
        }
        if (domain.indexOf(".facebook.com")!=-1){
            return getFacebookDate(searchStr, doc);
        }
        return getAnySiteDate(searchStr, doc);
    }

    public static int getMonthNumber(String month){
        for (int i=0; i<12; i++){
            if (month.equals(months[i])){
                return i;
            }
        }
        return -1;
    }

    static Calendar getDateFromFBStr(String str){
        int day, month = 0, year;
        int i=0;
        while(str.charAt(i)!=' '){
            i++;
        }

        day = Integer.parseInt(str.substring(0, i));
        str = str.substring(i+1);
        i = 0;
        while (str.charAt(i)!=' '){
            i++;
        }
        String monthName = str.substring(0, i);
        for (int j=0; j<12; j++){
            if (monthName.equals(months[j])){
                month = j;
                break;
            }
        }
```

```
        str = str.substring(i+1);
        year = Integer.parseInt(str.substring(0, 3));
        return new GregorianCalendar( year,month,day );
    }

    static Calendar getDateFromVKStr(String str){
        Calendar d = new GregorianCalendar();
        if (str.startsWith("вчера")){
            d.add(Calendar.DAY_OF_MONTH, -1);
            return d;
        }
        if (str.startsWith("сегодня")){
            return d;
        }
        int i=0;
        while(str.charAt(i)!=' '){
            i++;
        }
        int day = Integer.parseInt(str.substring(0, i));
        str = str.substring(i+1);
        i = 0;
        while (str.charAt(i)!=' '){
            i++;
        }
        String monthName = str.substring(0, i);
        int month = 0;
        for (int j=0; j<12; j++){
            if (months[j].startsWith(monthName)){
                month = j;
                break;
            }
        }
        str = str.substring(i+1);
        int year;
        if (str.charAt(0) == 'в'){
            year = 2016;
        } else {
            year = Integer.parseInt(str.substring(0, 3));
        }
        d.set(year, month, day);
        return d;
    }

    public static Calendar getFacebookDate(String searchStr, Document doc){
        Elements elements = doc.select("div[class=userContentWrapper _5pcr]");
        Calendar d = new GregorianCalendar();
        for (Element elem:elements){
            String content = elem.html();
            if (content.indexOf(searchStr)!=-1){
                Elements span = elem.select("span[class=fsm fwn fcg]"+ " abbr");
                String temp = span.first().attr("title");
                d =  getDateFromFBStr(temp);
                return d;
            }
        }
        return null;
    }

    public static Calendar getVKDate(String searchStr, Document doc){
        Elements elements = doc.select("div[class=post_table]");
        Calendar d = new GregorianCalendar();
        for (Element elem:elements){
```

```
            String content = elem.html();
            if (content.indexOf(searchStr)!=-1){
                Elements span = elem.select("span[class=rel_date]"+ " abbr");
                String temp = span.first().attr("title");
                d =  getDateFromVKStr(temp);
                return d;
            }
        }

        return null;
    }

    private static Matcher checkDatePattern(String regex, String text) {
        Pattern p = Pattern.compile(regex, Pattern.CASE_INSENSITIVE |
Pattern.DOTALL);
        return p.matcher(text);
    }

    public static Calendar getAnySiteDate(String searchStr, Document doc){
        if (doc == null){
            return null;
        }
        Calendar d = new GregorianCalendar();
        String regexp = DAY+ DELIMITER+MONTH+DELIMITER+YEAR;

        Elements div = doc.select("body");
        String innetHTML = div.text();

        Matcher m = checkDatePattern(regexp, innetHTML);
        if (m.find()){
             int day = Integer.parseInt(m.group(1));
             int month = Integer.parseInt(m.group(2));
             int year = Integer.parseInt(m.group(3));
             return new GregorianCalendar(year, month, day);
        } else {
            regexp = YEAR+ DELIMITER+MONTH+DELIMITER+DAY;
            m = checkDatePattern(regexp, innetHTML);
            if (m.find()) {
                int year = Integer.parseInt(m.group(1));
                int month = Integer.parseInt(m.group(2));
                int day = Integer.parseInt(m.group(3));
                return new GregorianCalendar(day, month, year);
            }
        }

        return null;
    }
}
```

Модуль DiffusionNetwork.

```
package crawlerpackage;

import org.jsoup.HttpStatusException;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

import java.io.IOException;
import java.net.URISyntaxException;
import java.util.*;
```

```
/**
 * Created by Zily on 07.06.2016.
 */
public class DiffusionNetwork {
    ArrayList<String> vertex;
    ArrayList<Calendar> dates;
    ArrayList<ArrayList<Integer>> network;

    public static final String USER_AGENT = "Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36";

    public  ArrayList<ArrayList<Integer>> getNetwork(){
        return network;
    }

    public  ArrayList<Calendar> getDates(){
        return dates;
    }

    public boolean empty(){
        return this.dates.size() ==0 && this.network.size() == 0;
    }

    public DiffusionNetwork(){
        vertex = new ArrayList<String>();
        dates = new ArrayList<Calendar>();
        network = new ArrayList<ArrayList<Integer>>();
    }

    public void Build(Set<String> urls, String query) throws IOException,
URISyntaxException {
        Iterator<String> it = urls.iterator();
        int URLcount = urls.size();
        Document docs[] = new Document[URLcount];
        int i=0;
        while (it.hasNext()){
            String currentURL = it.next();
            Document doc;
            try {
                doc =
Jsoup.connect(currentURL).userAgent(USER_AGENT).timeout(10000).get();
            } catch (HttpStatusException e){
                e.printStackTrace();
                continue;
            }
            docs[i] = doc;
            vertex.add(currentURL);
            Calendar date =
DateAnalysis.getCreationDate(SearchEngineParser.GetDomainName(currentURL),query,
docs[i++]);
            if (date!=null)
                System.out.println(date.get(Calendar.DAY_OF_MONTH) +
"."+date.get(Calendar.MONTH)+ "."+date.get(Calendar.YEAR));
            dates.add(date);
            network.add(new ArrayList<Integer>());
        }

        for (i=0; i<URLcount; i++){
            String currentURL = vertex.get(i);

            for (int j=0; j<URLcount; j++){
                if (i == j){
```

```
                    continue;
                }
                if (findURL(currentURL, docs[j])){
                    network.get(i).add(j);
                }
            }
        }
    }

    public boolean findURL(String URL, Document doc){
        Elements links = doc.select("a[href]");
        if (links == null){
            return  false;
        }
        for (Element link : links) {
            String temp = link.attr("href");
            if(temp.equals(URL)){
                return true;
            }
        }
        return false;
    }

    void putDates(){
        for (int i=0; i<vertex.size(); i++){
            if (dates.get(i) == null){
                Calendar date = dfs(i);
                dates.set(i, date);
            }
        }
    }

    Calendar dfs(Integer v){
        boolean compared = false;
        Calendar date = new GregorianCalendar();
        for (int i = 0; i< network.get(v).size(); i++){
            Calendar current =dates.get(network.get(v).get(i));;
            if (!compared){
                date = current;
            } else {
                if (current!= null && current.before(date)){
                    date = current;
                }
            }
        }
        if (!compared){
            return null;
        }
        return date;
    }
}
```

Модуль GoogleParser.

```
package crawlerpackage;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;
```

```
/**
 * Created by Zily on 28.04.2016.
 */
public class GoogleParser extends SearchEngineParser{
    public static final String GOOGLE_SEARCH_URL =
"https://www.google.com/search?q=";
    public static final String USER_AGENT = "Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36";
    @Override
    public Set<String> GetSearchQuery(String query, int numberOfPages)throws
IOException {

        Set<String> result = new HashSet<String>();
        String searchURL = GOOGLE_SEARCH_URL + query+ "&num="+numberOfPages;

        try {
            Document doc = Jsoup.connect(searchURL).userAgent(USER_AGENT)
                    .timeout(1000).get();
            Element resultRefDiv = doc.getElementById("ires");
            Elements results = resultRefDiv.select("h3.r a");

            for (Element element : results) {
                String linkHref = element.attr("href");
                result.add(getURLfromStr(linkHref));

            }

            results = resultRefDiv.select("a._Dk");

            for (Element element : results) {
                String linkHref = element.attr("href");
                result.add(getURLfromStr(linkHref));

            }
        }catch (IOException e) {
            e.printStackTrace();
        }
        return result;
    }

    public String getURLfromStr(String str){
        String url = new String();
        int garbageIndex = str.indexOf("&sa=");
        if (garbageIndex!=-1){
            url = str.substring(0, garbageIndex);
        } else {
            url = str;
        }
        int inCashe = url.indexOf("googleusercontent");
        if (inCashe!=-1){
            int newHead = url.indexOf(":http");
            return url.substring(newHead + 1);
        }
        return url;
    }
}
```

Модуль GraphSaver.

```
package crawlerpackage;

import java.io.*;
import java.util.*;
```

```
/**
 * Created by Zily on 24.06.2016.
 */
public class GraphSaver {
    static final String filename = "C:\\lastession.txt";

    public static boolean save(DiffusionNetwork d, String query){
        RandomAccessFile file;

        try(FileReader reader = new FileReader(filename))
        {
            try{
                file = new RandomAccessFile(filename, "rw");//попытка открыть файл
            }catch(Throwable e){

                return false;
            }
            try{//файл было открыть попробуем записать
                file.writeUTF(new String(query.getBytes("UTF-8"), "UTF-
8"));//записываем адрес
                file.writeInt(d.vertex.size());//записываем число
                for(int i = 0; i < d.vertex.size(); i++){
                    file.writeUTF(new String(d.vertex.get(i).getBytes("UTF-8"),
"UTF-8"));//записываем адрес
                    file.writeUTF(new
String(CalendarToStr(d.dates.get(i)).getBytes("UTF-8"), "UTF-8"));//записываем
дату

                }

                for (int i = 0; i<d.network.size(); i++){
                    file.writeInt(i);//записываем число
                    file.writeInt(d.network.get(i).size());
                    for (int j = 0; j<d.network.get(i).size(); j++){
                        file.writeInt(d.network.get(i).get(j));//записываем число
                    }
                }

            }catch(IOException e){
                e.printStackTrace();
            }

            file.close();//всё записали и закрыть файл
        }
        catch(IOException ex){
            ex.printStackTrace();
        }
        return true;
    }

    public static String CalendarToStr(Calendar date){
        if (date == null){
            return "undefined";
        }
        Integer day  = date.get(Calendar.DAY_OF_MONTH);
        Integer month = date.get(Calendar.MONTH);
        Integer year = date.get(Calendar.YEAR);
        return day.toString() + "/"+month.toString()+"/"+year.toString();
    }

    public static Calendar parseString(String str){
```

```
        if (str.startsWith("undefined")){
            return null;
        }
        String s[] = str.split("/");
        int day = Integer.parseInt(s[0]);
        int month = Integer.parseInt(s[1]);
        int year = Integer.parseInt(s[2]);
        return new GregorianCalendar(year, month, day);
    }

    public static DiffusionNetwork load(String query){
        DiffusionNetwork diffusionNetwork = new DiffusionNetwork();
        RandomAccessFile file;
        try{
            file = new RandomAccessFile(filename, "r");//открыть для чтения
        }catch(Throwable e){
            return null;//сообщить об ошибке открытия файла и прекратить работу
        }
        try{//файл был открыт попробуем прочесть
            query = file.readUTF();
            int n = file.readInt();
            for (int i=0; i<n; i++){
                String url;
                Calendar date;
                url = file.readUTF();
                date = parseString(file.readUTF());
                diffusionNetwork.vertex.add(url);
                diffusionNetwork.dates.add(date);

            }
            for (int i=0; i<n; i++){
                int v = file.readInt();
                int size = file.readInt();
                diffusionNetwork.network.add(new ArrayList<Integer>());
                for (int j=0; j<size; j++){
                    diffusionNetwork.network.get(i).add(file.readInt());
                }
            }
            file.close();//закрыть файл
        }catch(UTFDataFormatException e){
            return null;
        }catch(EOFException e){
            return null;
        }catch(IOException e){
            return null;
        }
        return diffusionNetwork;
    }

}
```

Модуль Layering.

```
package crawlerpackage;

import java.util.*;

/**
 * Created by Zily on 22.06.2016.
 */
public class Layering {
    TreeMap<Calendar, Set<Integer>> layers;
```

```
    int undefinedValues;

    public Layering(DiffusionNetwork d){
        layers = new TreeMap<Calendar, Set<Integer>>();
        undefinedValues = 0;
        build(d);
    }

    public void build( DiffusionNetwork d){
       int n = d.vertex.size();

        for (int i=0; i<n; i++){
            Calendar date = d.dates.get(i);
            if (date == null){
                undefinedValues++;
                continue;
            }
            System.out.println(date.get(Calendar.DAY_OF_MONTH));
            System.out.println(date.get(Calendar.MONTH));
            System.out.println(date.get(Calendar.YEAR));
            if (!layers.containsKey(date)){
                layers.put(date, new HashSet<Integer>());
            }
            layers.get(date).add(i);
        }
    }

    public Map<Calendar, Set<Integer>> getLayers(){
        return layers;
    }
}
```

Модуль Point.

```
package crawlerpackage;

/**
 * Created by Zily on 26.06.2016.
 */
public class Point{
    public float x, y;
    Point (float xx, float yy){
        x = xx;
        y = yy;
    }

    Point (int xx, int yy){
        x = xx;
        y = yy;
    }

    Point(){
        x = 0;
        y = 0;
    }
}
```

Модуль SearchEngineParser.

```
package crawlerpackage;

import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;
```

```
import java.util.Set;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Created by Zily on 20.05.2016.
 */
public abstract class SearchEngineParser {
    public static String GetDomainName(String url) throws URISyntaxException {
        String domainName = new String();
        URI uri = new URI(url);
        domainName = uri.getHost();
        return domainName.startsWith("www.")?domainName.substring(4):domainName;
    }
    public abstract Set<String> GetSearchQuery(String query, int numberOfPages)
throws IOException;
}
```

Модуль Visualisation.

```
package crawlerpackage;

import org.joda.time.LocalDate;

import javax.swing.*;
import java.awt.*;
import java.awt.geom.AffineTransform;
import java.util.*;

/**
 * Created by Zily on 21.06.2016.
 */
public class Visualisation extends JPanel{

    private static final float BORDER = 20;
    private static final int VERTEXSIZE = 10;
    private float cellW, cellH;

    DiffusionNetwork  diffusionNetwork;
    ArrayList<Point> vertices;
    Layering h;

    private boolean active;

    public Visualisation(){
        active = false;
        diffusionNetwork = new DiffusionNetwork();
        vertices = new ArrayList<Point>();
    }

    public void setDiffusionNetwork(DiffusionNetwork dnw){
        diffusionNetwork = dnw;
    }

    public void activate(){
        active = true;
    }

    public void disactivate(){
        active = false;
    }
```

```
    int max(int a, int b){
        return a>b?a:b;
    }

    void drawEdge(Graphics2D g, int x1, int y1, int x2, int y2) {

        double dx = x2 - x1, dy = y2 - y1;
        double angle = Math.atan2(dy, dx);
        int len = (int) Math.sqrt(dx*dx + dy*dy);
        AffineTransform at = AffineTransform.getTranslateInstance(x1, y1);
        at.concatenate(AffineTransform.getRotateInstance(angle));
        g.transform(at);

        // Draw horizontal arrow starting in (0, 0)
        g.drawLine(0, 0, len, 0);
        g.fillPolygon(new int[] {len, len- PIKE_SIZE, len- PIKE_SIZE, len},
                new int[] {0, -PIKE_SIZE, PIKE_SIZE, 0}, 4);
    }

    void drawVertex(Graphics2D g2d, int x, int y, int size){
        g2d.setColor(Color.BLUE);
        g2d.fillRect(x+2, y+2, size, size);
        g2d.setColor(Color.BLACK);
        g2d.drawRect(x + 2, y+2, size, size);
    }

    void drawAxes(Graphics2D g2d){
        float winW = getWidth();
        float winH = getHeight();

        // g2d.drawString("Количество сутпвоом и н анноивйо с вгд е н"ь (int)(winW/4),
12);
        g2d.drawRect(((int) BORDER), ((int) BORDER), ((int)(winW-2*BORDER)),
((int)(winH-2.5*BORDER)));
        g2d.setColor(Color.BLACK);
        float i=0;
        while (i<winH - 2*BORDER){
            g2d.drawLine(((int) BORDER - 2),((int) i), ((int) BORDER + 2), (int)i
);
            String s = "" + ((int)(i/cellH));
            g2d.drawString(s, 2, ((int)(winH - BORDER - i - 1)));
            i += cellH;
        }

        float x = BORDER, y = winH - 1.5f*BORDER;

        for (Map.Entry<Calendar, Set<Integer>> element:h.layers.entrySet()) {
            LocalDate date = new LocalDate(element.getKey());
            g2d.drawString(date.toString(), x, y + BORDER);

            float x1 = x, y1 = y;
            for (Integer vertex: element.getValue()){
                y1 -= cellH;
                float size = (int)cellH - 4;
               drawVertex(g2d, (int)(x1 + cellW/2 - size/2),(int)y1, (int)size);
            }

            x += cellW;
            g2d.drawLine((int)x, (int)y-3, (int)x, (int)y+3);
        }
    }
```

```
    void setVertexOnPoints(){
        if (diffusionNetwork == null){
            return;
        }
        int count = diffusionNetwork.vertex.size();
        if (count == 0){
            return;
        }
        for (int i=0; i<count; i++){
            vertices.add(new Point());
        }

        float winW = getWidth();
        float winH = getHeight();

        cellW = (winW - 2*BORDER)/(h.layers.size() + 1);
        cellH = 20;
        float H = winH - 3*BORDER;
        float W = winW - 2*BORDER;
        float x0 = BORDER, y0 = winH - 1.5f*BORDER;

        float x = x0, y = y0;

        for (Map.Entry<Calendar, Set<Integer>> element:h.layers.entrySet()) {

            float x1 = x, y1 = y;
            for (Integer vertex: element.getValue()){
                y1 -= cellH;
                float size = (int)cellH - 4;
                int v = vertex;

                vertices.get(v).x = (x1 + cellW/2 - size/2);
                vertices.get(v).y = y1;
            }
            x += cellW;
        }
    }

    void drawVertices(Graphics2D g2d, float size){

        for (int i=0; i<vertices.size(); i++){
            if (diffusionNetwork.dates.get(i) == null){
                return;
            }
            Point p = vertices.get(i);
            drawVertex(g2d, (int)(p.x),(int)p.y, (int)size);
        }
    }

    void connectVertices(Graphics2D g2d){
        float size = (int)cellH - 4;
        for (int i = 0; i< diffusionNetwork.network.size(); i++){
            if (diffusionNetwork.dates.get(i) == null){
                continue;
            }
            Point p1 = vertices.get(i);
            p1.x += size;
            p1.y += size/2.0;
            for (int j = 0; j<diffusionNetwork.network.get(i).size(); j++){
                if
(diffusionNetwork.dates.get(diffusionNetwork.network.get(i).get(j)) == null){
                    continue;
```

```
                }
                Point p2 = vertices.get(diffusionNetwork.network.get(i).get(j));
                p2.y += size/2.0;
                drawEdge(g2d, (int)p1.x, (int)p1.y, (int)p2.x, (int)p2.y);
            }
        }
    }

    public void draw(Graphics g){
        Graphics2D g2d = (Graphics2D) g;
       // g2d.drawString("К о л и ч е сутпвоом и н анноивйо с ег д е н"ь (int)(winW/4),
12);

        h = new Layering(diffusionNetwork);

        setVertexOnPoints();

        drawAxes(g2d);

        drawVertices(g2d, (int)cellH - 4);

        connectVertices(g2d);
    }
    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        if (active)
            draw(g);
    }
}
```

Модуль YandexParser.

```
package crawlerpackage;
import java.io.IOException;
import java.net.URLEncoder;
import java.util.HashSet;
import java.util.Set;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;


/**
 * Created by Zily on 28.04.2016.
 */
public class YandexParser extends SearchEngineParser{
    private static final String YANDEX_SEARCH_URL  =
"https://yandex.com/search/xml?";
    private static final String ENC = "UTF-8";
    private static final String AND = "&";
    private static final String USER = "user=khzilli";
    private static final String KEY =
"key=03.50036319:3f46e9c1c099c225069f41040a0e1cfe";

    @Override
    public Set<String> GetSearchQuery(String query, int numberOfPages)throws
IOException{
        Set<String> result = new HashSet<String>();
        String searchURL = YANDEX_SEARCH_URL +  USER + AND +  KEY+ AND +
               "query=" + URLEncoder.encode(query, ENC) + AND + "page=" +
```

```
numberOfPages;
        try {
            Document doc = Jsoup.connect(searchURL).userAgent("Mozilla/5.0
(compatible; Googlebot/2.1; +http://www.google.com/bot.html)")
                    .timeout(1000).get();
            Elements results = doc.select("url");

            for (Element element : results) {
                String linkHref = element.text();
                result.add(linkHref);
            }
        }catch (IOException e) {
            e.printStackTrace();
        }
       return result;
    }
}
```