САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

Куликов Егор Константинович

Хранение и анализ хеш-значений файлов в компьютерно-технической экспертизе

Курсовая работа

Научный руководитель:
ст. преп. Губанов Ю. А.

Санкт-Петербург
2015

Оглавление

Введение

Всё чаще для установления причастности определённого лица к совершённому преступлению или доказательства его вины прибегают к помощи компьютерно-технической экспертизы. В соответствии с судебным постановлением у подозреваемого конфискуются имеющиеся устройства (ноутбуки, планшеты, айфоны и др.), которые в последствии анализируются с помощью специальных программ.

В цифровой криминалистике важное значение имеют алгоритмы, основанные на хешировании[8].Так, например, с их помощью можно установить, является ли образ, находящийся в руках исследователя, точной копией некого оригинального дискового устройства, что в дальнейшем позволяет использовать результаты, полученные при анализе образа при помощи специального криминалистического программного обеспечения, в качестве улик в суде [5]. Далее речь пойдёт о другом, не менее важном применении алгоритмов хеширования в компьютерно-технической экспертизе.

Главным объектом исследования является совокупность хранимых на устройстве файлов. Содержащаяся в ней информация позволяет установить причастность подозреваемого к совершению деяний, осведомлённость в определённых вопросах, его сферу деятельности и круг интересов, а также, возможно, выявить наличие файлов, запрещённых законом к хранению, то есть непосредственные улики преступления. Такими файлами, например, в ряде стран (США, Япония и др.) являются изображения, содержащие детскую порнографию.

Однако огромное количество хранящихся файлов на самом деле не представляют для следователя ни малейшего интереса. Это файлы операционной системы, файловой системы, системные библиотеки, прикладное программное обеспечение. В связи с этим возникает естественная необходимость отсечь “лишние”, не представляющие интереса как возможные улики файлы, и не исследовать их.

Для решения этой задачи была разработана библиотека NSRL (National Software Reference Library), которая на данный момент содержит 9660 ISO-образов размером примерно в 700 мегабайт каждый. Каждый из образов, в свою очередь, содержит текстовый файл, в котором перечислен набор данных о файлах, гарантированно не содержащих важной для эксперта-криминалиста информации. Этот набор, в частности, включает в себя *имя файла*, например *ProjSettingsNode.java* и значения его *хеш-функции*, посчитанные с помощью двух различных алгоритмов: *MD5* и *SHA-1*, например, *8D025B6AE1994A40FCBB5AEC2EF273F9*. NSRL, однако, является далеко не единственной подобной библиотекой. Так, например, на сайте *http://www.hashsets.com/* находятся несколько архивов хеш-значений заведомо нелегитимных файлов, главным образом вирусных. Такие файлы мы будем в дальнейшем называть плохими. Также там опубликованы базы хеш-значений известных файлов, в основном относящихся к работе операционной системы. Их, в свою очередь, договоримся называть хорошими.

Для успешного проведения анализа хранимых на устройстве файлов, необходимо:
1) Создать эффективное хранилище для информации о “хороших” и “плохих” файлах, основанное на NSRL и пригодное для работы в условиях использования одного компьютера следователя или эксперта, без дополнительной памяти и вычислительных мощностей.
2) Собрать полную информацию о файлах, содержащихся на электронном устройстве, и вычислить для каждого из них хеш-функцию с помощью применяемых в NSRL алгоритмов хеширования.
3) Для каждого файла проверить, не находится ли он в хранилище, и принять решение о возможности его исключения из списка необходимых для изучения в рамках следственных действий или же включения в ряд возможных улик.
4) Интегрировать предложенный алгоритм поиска в продукт цифровой криминалистики Belkasoft Evidence Center.

Задачей данной курсовой работы является реализация наиболее эффективного хранилища и способа хранения данных в контексте проведения компьютерно-технической экспертизы совокупности файлов и разработка эффективных алгоритмов проверки наличия или отсутствия файла в хранилищах, исключение ”лишних файлов”. Для вычисления хеш-функций будет использоваться криптографический пакет с открытым исходным кодом *OpenSSL*.

Необходимо отметить, что, во-первых, объём информации, хранимой в NSRL базах, составляет несколько терабайт, и в этом объёме (возможно, некоторым образом преобразованном), необходимо организовать эффективный поиск. Во-вторых, количество файлов, хранимых на исследуемом устройстве, может составлять несколько миллионов, и для каждого из них необходимо реализовывать поиск по хранилищу.

1. Обзор существующих решений

Прежде чем проводить обзор существующих решений, выявляя соответственно их достоинства и недостатки, установим эталонный набор данных, на примере обработки которого будем производить сравнение производительности. Возьмём в качестве такового предлагаемый разработчиками библиотеки NSRL файл с хеш-значениями, содержащий 19998552 записи и имеющий размер около 1.3 гигабайта, и назовём этот набор *эталонным*.

Все приводимые в тексте работы временные характеристики получены при помощи вычислительной машины Acer Aspire 5745G с процессором Intel Core i5 и установленной оперативной памятью 4.00 ГБ. Отметим, что большинство используемых отечественными экспертами-криминалистами устройств имеет вычислительную мощность, схожую с мощностью того устройства, на котором проводилось исследование.

1.1. Autopsy

Autopsy — инструмент компьютерной криминалистики и графический интерфейс к бибилиотеке *The Sleuth Kit*, позволяющей производить различные исследования образов дисков и файловой системы [1]. В частности, имеется возможность создания базы хеш-значений на основе текстовых данных библиотеки NSRL, вычисление хеш-функции для заданного набора файлов и проверки наличия файла в сгенерированной базе.

Сначала приложение производит *индексирование* исходного файла с данными. Суть алгоритма индексирования такова: в дополнение к существующему файлу с данными о хешах создаётся ещё один, содержащий ту же самую информацию, но уже специальным образом упорядоченную: у каждой записи исходного файла рассматривают первые три символа хеш-значения и вставляют информацию о рассматриваемом документе в новый файл в то место, где уже содержатся записи о документах, имеющих точно такие же первые три символа. Кроме того, формируется ряд чисел, который мы в дальнейшем будем называть *списком отступов*. Их смысл следующий: первый отступ показывает, с какого места начинаются данные о файлах, хеш которых начинается с символов *000* и равен нулю, второй — место начала данных о имеющих в начале своего хеша *001*, и так далее. Всего 256 отступов. Если документов, хеши которых начинаются с символов *ijk* в исходном документе нет, то отступ устанавливается равным наименьшему из отступов, определённых для хешей, имеющих большее значение.

Теперь рассмотрим алгоритм поиска хеша в базе: осуществляется бинарный поиск по таблице отступов; для каждого рассматриваемого отступа сравнивается значение его первого хеша и разыскиваемого и в зависимости от результата сравнения поиск

сужается на левую или правую половину таблицы отступов.

Недостатки такого подхода начинают выявляться уже при работе с эталонным набором хеш-значений и усугубляться при росте размера входного документа. Причина в том, что при таком хранении размер индексированного файла оказывается приближённо равным суммарному размеру входных текстовых данных. Подобный файл затруднительно открыть, просмотреть содержимое, редактировать, и.т.п. Кроме того, у файловой системы существуют ограничения на максимальный размер файла, и возможен риск этот предел превысить. Наконец, несмотря на хорошую асимптотику $O(logn)$, многократное применение бинарного поиска к файлу очень большого размера пагубно влияет на производительность. При индексировании сканирование определённых областей файла, возможно, содержащих сотни тысяч записей, для нахождения того места, куда необходимо вставить текущую, также не может не оказывать существенного влияния на производительность.

Отметим, однако, что при работе с небольшими базами хеш-значений этот простой и понятный алгоритм, предложенный разработчиками Autopsy, весьма эффективен. Так, индексирование эталонного набора занимает несколько минут, поиск десятка тысяч документов в сгенерированном файле с отступами — примерно столько же.

1.2. Internet Evidence Finder

Ещё одной утилитой, позволющей производить анализ файлов на основе их хеш-значений, является Internet Evidence Finder, разрабатываемый компанией Magnet Forensics.

Утилита позволяет вычислить хеш-функции для каждого файла из заданного набора, построить базу хешей на основе текстового файла и проверить, присутствуют ли интересующие пользователя хеш-значения в базе. Кроме того, имеются и дополнительные возможности, связанные с применением технологии PhotoDNA, о которой речь пойдёт далее. Утилита посылает запрос на специальный сервер и получает ответ о характеристике файла.

Хранилище, предлагаемое разработчиками Internet Evidence Finder, кардинально отличается от применяемого в Autopsy. Так, вся информация сохраняется в одну SQLite базу данных и в дальнейшем по этой базе осуществляется поиск. Все прочитанные хеши загружаются в единственную таблицу *Entries* базы данных *FileHashList.db*. Производительность при таком подходе оказывается низкой: после полутора часов работы в таблице можно обнаружить менее двух миллионов записей, то есть около 10% записей эталонного файла.

Неудивительно, что разработчики приложения рекомендуют поставить специальный флаг отказа от индексирования файлов, размер которых превышает 500 мегабайт. Впрочем, как мы только что показали, файл даже такого размера будет индек-

сироваться около шести часов.

1.3. Photo DNA

PhotoDNA — технология, разработанная корпорацией Microsoft в целях борьбы с распространением детской порнографии. Эта технология также основана на хешировании, однако применяется исключительно для анализа изображений.

Разница с остальными рассматриваемыми алгоритмами заключается в том, что хеш-функция вычисляется не для полного изображения, а для его маленьких кусочков, причём предварительно изображение переводится в чёрно-белый вид. Такая методика позволяет обнаруживать схожие важные фрагменты на изображениях совершенно разного размера и цвета. Технология, оказывается, безусловно, гораздо более эффективной, нежели стандартный анализ хеша целого файла, однако применима она лишь к изображениям[5].

В нашем случае использовать технологию не представляется возможным в силу уже отмеченных недостатков и того, что необходимость отправлять сотни тысяч запросов на сервер нас в рамках поставленной задачи не устраивает. Тем не менее, эта технология, безусловно, имеет практическое применение: National Center for Missing & Exploited Children (NCMEC) использует PhotoDNA [6].

1.4. Forensics Explorer

Forensics Explorer — один из продуктов криминалистического анализа, разрабатываемый компанией GetData. Этот продукт также осуществляет вычисление хеш-значений и их поиск в базах, однако отличается от описанных выше инструментов тем, что не позволяет самостоятельно создавать базы хеш-значений, а лишь пользуется готовыми, которые требуется скачать в специальную папку с официального сайта продукта или с сайта *http://www.hashsets.com/*.

Рассмотрим чуть подробнее базы с только что упомянутого сайта и выявим достоинства и недостатки их использования. Это наборы как хороших, так и плохих хеш-значений (например, вредоносного ПО, порнографии). Наборы хороших хешей содержат хеши документов, распространяемых органами власти США, а также некоторые наиболее часто встречающиеся записи NSRL-баз.

Разработчики хранилища на своём официальном сайте отмечают, что изначальной причиной его разработки была необходимость устранения одного недостатка NSRL: того, что NSRL не содержит информацию о большом количестве системных файлов, связанных с установкой операционных систем и драйверов.

В результате проведённых разработчиками хранилища исследования получился набор баз хеш-значений, который является дополнением к NSRL, а также поднабором часто встречающихся её элементов. Однако размер этих баз ничтожно мал, составляет

лишь около десятка миллионов хеш-значений, что делает поиск по ним полезным и быстрым, но явно недостаточным.

2. Подходы к реализации

2.1. Подходы, использующие базы данных

В рамках проводимого исследования были изучены как возможности использования ставших традиционными реляционных баз данных, так и завоевавшие огромную популярность в последнее десятилетие технологии NoSQL.

Здесь и далее в тексте работы возникает необходимость в использовании определённой терминологии, поэтому для определённости периодически будут приводиться некоторые определения.

Базой данных (БД) называется организованная в соответствии с определенными правилами и поддерживаемая в памяти компьютера совокупность сведений об объектах, процессах, событиях или явлениях, относящихся к некоторой предметной области, теме или задаче [9].

Реляционная база данных представляет собой множество взаимосвязанных таблиц, каждая из которых содержит информацию об объектах определенного вида. Каждая строка таблицы содержит данные об одном объекте (например, автомобиле, компьютере, клиенте), а столбцы таблицы содержат различные характеристики этих объектов — атрибуты (например, номер двигателя, марка процессора, телефоны фирм или клиентов).

Для работы с данными используются системы управления базами данных (СУБД), основными функциями которых являются описание структуры баз данных, обработка данных и управление ими.

2.1.1. Применение реляционных БД

В связи с тем, что необходимо решать задачу с помощью вычислительных ресурсов одного компьютера, применяемая нами база должна быть в достаточной степени "легковесной". Кроме того, необходимо обойтись без использования серверов, необходимых, например, MS SQL Server, так как организация работы с сервером и другие подобные действия - достаточно затруднительная задача для типичного пользователя-криминалиста. Тем не менее, подходящие традиционные базы всё же существуют.

Среди разработчиков весьма популярной реляционной базой данных является SQLite. Эта встраиваемая база , то есть "движок" SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а предоставляет библиотеку, с которой программа компонуется. Таким образом, движок становится составной частью программы, что уменьшает время отклика и упрощает программу. SQLite также весьма проста в использовании.

Как уже было отмечено в обзоре существующих решений, разработчики Internet

Evidence Finder попытались применить SQLite к решению поставленной задачи и добились производительности около 20000 записей в минуту.

В рамках поиска эффективного хранилища были рассмотрены две оптимизации применяемого алгоритма:

1) использование нескольких таблиц в рамках одной базы

2) использование нескольких баз данных

Для ускорения записи применялись транзакции: с их помощью записи в базу загружаются не по одной, а группами, что ускоряет процесс сохранения, так как требует меньшее количество раз открывать базу на дозапись.

Использования нескольких таблиц могло бы существенно ускорить поиск нужной записи, потому что при таком подходе размер таблицы, в которой производится поиск уменьшается в число раз, равное количеству таблиц. Это количество, следовательно, необходимо максимизировать.

В рамках исследования разбиение на таблицы проводилось по первым нескольким символам хеша файла. Если в одну таблицу попадали те хеши, у которых совпадали первые пять символов, то число таблиц оказывалось равным 2^{20}, то есть около миллиона. Но только сам процесс создания таблиц уже оказался чрезвычайно длительным: за 5 часов было создано около 45000 таблиц, что делает идею создания большого числа таблиц практически неприменимой. Если же разбиение на таблицы производить по первым четырём символам, то процесс создания таблиц займёт, конечно, меньшее время, но, во-первых, существенно сократить объём информации, в которой необходимо производить поиск, не получится, во-вторых, производительность записи оказывается приблизительно равной 20000 штук в минуту, то есть почти такой же, как и при записи в одну таблицу.

Если же вместо множества таблиц использовать большое количество баз данных и распределять файлы в базы по тому же принципу (начальные символы определяют базу), то мы увидим, что создание 2^{20} степени баз занимает около трёх часов, что вполне нормально, однако скорость записи всё равно окажется почти такой же как и в предыдущих случаях.

Отметим, что скорость сохранения данных о файлах, равная 20000 записей в минуту, является неудовлетворительной по следующим соображениям. Если эталонный файл размером около 1.3 ГБ содержит примерно 20 миллионов записей, то время, требуемое на запись 7 ТБ, как показывают несложные вычисления, оказывается приближённо равным $7 * 1024/1.3 * 2 * 10^7/20000$, то есть около десяти лет .

Таким образом, можно заключить, что применение SQLite в контексте поставленной задачи является бесперспективным.

2.1.2. Применение NoSQL

Технология NoSQL возникла сравнительно недавно, в начале 2000-х годов в дополнение к традиционным СУБД. В своей книге "NoSQL. Новая методология разработки нереляционных баз данных"[10] Мартин Фаулер приводит причины популярности NoSQL, одна из которых заключается в том, что нереляционные базы оказываются более эффективными в работе с большими объёмами данных, в особенности в ситуациях, когда не требуется делать сложных выборок с многократным применением операции join. В связи с этим в рамках проводимого исследования была предпринята попытка применить NoSQL к решаемой задаче. Как будет показано в дальнейшем, данная технология действительно позволяет добиться выигрыша во времени сохранения данных о хешах в 3.5 раза, однако получившееся время, требуемое на обработку всей информации NSRL-хранилища всё равно окажется слишком большим.

С точки зрения структуры, содержания и ограничений целостности выделяют четыре типа NoSQL баз.

1)*хранилища «ключ-значение»* являются простейшими хранилищами данных, использующими ключ для доступа к значению. Они используются для хранения изображений, создания специализированных файловых систем, в качестве кэшей для объектов.

2)В *хранилище семейств колонок* данные хранятся в виде разреженной матрицы, строки и столбцы которой используются как ключи. Типичным применением этого вида СУБД является веб-индексирование, а также задачи, связанные с пониженными требованиями к согласованности данных.

3)*документо-ориентированные СУБД* служат для хранения иерархических структур данных. Находят своё применение в системах управления содержимым, издательском деле, документальном поиске.

4)*графовые базы данных* применяются для задач, в которых данные имеют большое количество связей, например, мониторинг активности пользователей в социальных сетях.

В рамках проводимого исследования было принято решение остановится на хранилищах типа 1 и 3, так как решаемые с их помощью задачи наиболее близки к поставленной. В качестве базы типа 1 была рассмотрена BerkeleyDB, типа 3 - MongoDB.

2.1.3. Berkeley DB

BerkeleyDB — база с открытым исходным кодом, написанная на языке C. Первая версия была разработана ещё в 1986 году.

Была произведена попытка записать в базу данных информацию о файлах, причём для ускорения процесса использовались транзакции, с помощью которых в ба-

зу добавлялась группа из 10000 записей. Однако вновь возникла с проблема крайне длительного сохранения информации. Причём если в ситуации с SQLite наблюдалась хотя бы линейная зависимость времени, затрачиваемого на добавление в базу от числа добавляемой информации, то при применении Berkeley DB возникают ещё и серьёзные проблемы масштабирования. Так, уже после добавления 100000 записей производительность падает в разы: после часа работы по сохранению в базе оказались лишь 592000 записей, то есть скорость составила менее 10000 записей в минуту, что в два раза меньше, чем при использовании SQLite.

2.1.4. Mongo DB

MongoDB - база с открытым исходным кодом, написанная на языке C++ в 2009 году[7].

Эта база удобна тем, что она достаточно хорошо документирована: существует сайт www.mongodb.org, на котором можно найти полезную для пользователя информации.

Был проведён такой же эксперимент по сохранению большого объёма однотипных данных в это хранилище. В MongoDB вместо понятия базы данных используется семантически схожий термин *“коллекция”*.

Будем считать, что в одной коллекции находятся записи о файлах, хеши которых начинаются с нескольких одинаковых символов. К сожалению, число коллекций в MongoDB ограничено числом 24000[3], поэтому проводить разбиение по первым четырём или пяти символам не представляется возможным - столь большое число коллекций уже не поддерживается. При разделении по трём первым символам добавление в базу происходит корректно, с линейной зависимостью времени от числа добавляемой информации, причём скорость составляет около 70 тысяч записей в минуту. Несмотря на то, что это в 3.5 раза быстрее, чем при использовании SQLite и, соответственно, одно из упомянутых в обзоре существующих решений, такая скорость добавления информации всё равно оказывается медленной.

Отметим, что в MongoDB отсутсвует понятие транзакции. Таким образом, их применение (многократно используемое нами ранее) в данном случае повысить скорость записи невозможно.

2.2. Применение плоских файлов

Плоский файл — это набор данных, представляющий из себя текстовый файл, в котором информация храниться построчно без какой-либо специальной дополнительной структуры. В некотором смысле плоский файл есть очень простая база данных.

Отметим, что наиболее удачная попытка решения поставленной задачи, предпринятая разработчиками Autopsy, основывается как раз на применении плоских файлов.

Каковы же преимущества использования плоских файлов?

Во-первых, не вызывает затруднений создать достаточно большое количество плоских файлов. В частности, если разделить все записи NSRL-архива по совпадению или несовпадению первых четырёх символов, то есть создать 2^{16} файлов, то в одной таблице окажется примерно $7 * 1024/1.3 * 2 * 10^6/2^{16}$ записей, или около 160000.

Во-вторых, с помощью обычного бинарного поиска по файлу, содержащему около 160000 записей, почти мгновенно можно ответить на вопрос, находится ли рассматриваемое хеш-значение в этом плоском файле.

В-третьих, несмотря на то, что при работе с плоскими файлами разработчик несёт непосредственную ответственность за целостность и корректность данных, технически работать с плоскими файлами значительно проще, чем с имеющими сложную внутреннюю структуру и особый, зачастую уникальный синтаксис NoSQL.

Выделим ряд недостатков, существующих у алгоритма, используемого разработчиками Autopsy, детально рассмотренного в обзоре существующих решений.

1)Отсутствие каких-либо заранее сгенерированных баз известных хеш-значений.

2)Использование одного плоского файла для хранения записей. При таком подходе, во-первых, возникают сложности просмотра и, в случае необходимости, редактирования файла ввиду его огромного размера. Во-вторых, не исключено, что размер файла в определённый момент превысит максимальное допустимое значение. Наконец, при каждом запросе на поиск придётся обращаться к гигантскому файлу, что, в свою очередь, оказывает негативное влияние на производительность.

3)Разделение всего объёма поступающих хеш-значений по первым трём символам. При таком подходе по одному адресу отступа может оказаться более чем 2.5 миллиона хеш-значений при использовании NSRL-хранилища. В этом случае объём информации, в которой необходимо многократно производить поиск, достаточно велик, а значит, занимает больше времени.

4) Запись информации о каждом отдельном хеш-значении по его адресу. Вместо этого лучше накапливать некоторое количество хеш-значений одного вида, и уже потом записывать их в файл все сразу. Таким образом, сокращается число обращений к плоскому файлу, а также образующийся там по итогам записи набор данных оказывается сразу же частично отсортированным.

Наконец, предполагается, что пользователю будет удобнее использовать отдельное хранилище для тех файлов, которые он объявил известными и надёжными, отделив его от NSRL-хранилища.

3. Предлагаемый алгоритм

3.1. Организация файловой системы

Сначала рассмотрим вопрос о количестве используемых текстовых файлов. Как уже было отмечено ранее, если сделать это количество слишком маленьким, то файлы, соответственно, окажутся достаточно большими, а значит поиск по ним будет работать недостаточно быстро. Кроме того, существует опасность, что размер файла превысит максимально допустимый файловой системой компьютера.

В то же время, если сделать количество файлов слишком большим, то как при поиске, так и при записи придётся с некоторой регулярностью все эти файлы открывать и закрывать, производить дозапись. Если операционной системе необходимо регулярно совершать подобные действия, это негативно сказывается на производительности.

Экспериментальным путём было установлено, что оптимальные результаты по времени записи и поиска достигаются при использовании 2^{16} файлов.

После определения необходимого количества файлов требуется организовать хранение файлов с хешами в памяти компьютера. Известно, что если создать каталог, содержащий много файлов или много подкаталогов, то работа ОС Windows с такими каталогами будет крайне медленной.

В связи с этим, было решено распределить 65536 файлов по 64 папкам поровну, то есть по 1024 файла в папке а папки организовать в виде двоичного дерева: корневая папка с номером 1, в неё вложены папки 2 и 3 и так далее, на нижнем уровне расположены папки-листья с номерами 64 - 128, в которых и содержатся текстовые файлы. Такой подход позволяет избежать падения производительности, связанного с особенностями операционной системы.

3.2. Алгоритм сохранения

Предлагаемый алгоритм сохранения хеш-значений в текстовый файл устроен следующим образом:

1) из входного потока читается очередная запись о файле

2) по первым четырём символам хеш-значения вычисляется номер нужной папки и нужного файла.

3) данные о файле (хеш и имя) добавляются в буфер в место, соответствующее требуемому файлу и папке

4) когда общий размер буфера становится достаточно большим, происходит полная выгрузка данных из него в соответствующие текстовые файлы

Процесс продолжается, пока не будут обработаны все входные данные.

Такой алгоритм позволяет добиться производительности, равной приблизительно миллиону записей в минуту. Этот показатель значительно превосходит все те, ко-

торые на исполняемой машине были получены при использовании реляционных и нереляционных баз.

3.3. Алгоритм поиска

Для поиска по файлу будем применять классический бинарный поиск, отличающийся вполне удовлетворительной эффективностью при встречающихся в нашем решении размерах текстовых документов. Однако при его реализации возникают некоторые трудности, связанные с тем, что классический алгоритм подразумевает загрузку всего файла в оперативную память. А такая загрузка либо не представляется возможной в связи с большим размером файла, либо сильно снижает производительность, так как требует линейное время (невозможно считать строчку с произвольным номером из текстового файла, не считывая предыдущие).

Однако же, существуют возможности для побайтного чтения входного потока, начиная с некоторого места. В связи с этим было принято решение выравнять строки по длине, отделив хеш файла и его имя необходимым числом пробелов. Учитывая, что длина хеша составляет 32 символа, а имена файлов — до 200 символов, было решено отвести строчку длиной 250 символов под информацию об одном файле.

Приведённые выше размышления позволяют сформулировать следующий алгоритм.

Алгоритм поиска хеша в текстовом файле:

1) читается очередная запись о файле из входного потока

2) по первым четырём символам хеша вычисляются номера папки и файла, в котором нужная информация потенциально может содержаться.

3) осуществляется бинарный поиск по определённому во втором пункте файлу и генерируется ответ на запрос о принадлежности данного файла хранилищу.

3.4. k-way merge sort

Как известно, алгоритм бинарного поиска работает только по отсортированным файлам. Поэтому после того, как все данные о файлах сохранены в соответствующие текстовые документы и перед тем, как осуществлять поиск, необходимо отсортировать все файлы всех папок хранилища. Как уже было отмечено выше, загружать целиком файл в оперативную память — неподходящее решение, поэтому необходимо поступить несколько иначе: применить одну из модификаций алгоритма k-way merge sort [4].

Идея метода состоит в том, чтобы разбить файл на k кусочков, последовательно подгрузить их в оперативную память, отсортировать и загрузить уже упорядоченные

версии во временные текстовые файлы. Потом произвести слияние временных файлов в один окончательный файл, а все временные удалить.

Значение константы k было экспериментально выбрано равным 1000.

3.5. Интеграция в Belkasoft Evidence Center

3.5.1. Общие сведения

Важной особенностью реализации алгоритма поиска и хранения хеш-значений файлов является то, что он должен быть встроен в продукт цифровой криминалистики Belkasoft Evidence Center.

Это продукт отечественной компании Belkasoft[2], появившийся на рынке в 2010 году. Инструмент предоставляет возможность поиска сообщений, писем, активности в социальных сетях, P2P-программах, разговоров в онлайн-играх и т.п. в целях проведения цифровой криминалистической экспертизы. Поиск может осуществляться на разных источниках — на жёстком диске или на других носителях (компакт-диски, USB-флеш-накопители и т.п.), на образах оперативной памяти, файлах подкачки и гибернации и т.д. Помимо всего прочего, в продукте существует возможность добавлять закладки на найденные артефакты, осуществлять по ним поиск и строить отчёты.

3.5.2. Интеграция

В продукте Belkasoft Evidence Center существует огромный класс механизмов исследования, называемых анализаторами. Классы-анализаторы позволяют исследовать носитель в зависимости от поставленных задач побайтово или посимвольно на предмет определённой информации. Так, например, анализаторы браузеров позволяют определять список посещённых сайтов, дату и время их посещения, закладки, файлы “cookie”, другую полезную информацию.

В рамках решения задачи был создан отдельный анализатор, позволяющий собрать полный набор хеш-значений файлов исследуемого устройства, проверить их на принадлежность хранилищу и вернуть пользователю список тех, которые были обнаружены.

Сгенерированные в процессе решения задачи базы хеш-значений, основанные на NSRL, планируется в ближайшее время опубликовать на официальном сайте компании Belkasoft.

Заключение

Было проведено исследование по поиску оптимального хранилища хеш-значений файлов. в качестве такового было выбрано множество плоских файлов, организованных в древовидную структуру, было организовано простое в использовании хранилище для хеш-значений файлов. Была достигнута скорость записи в базу, равная миллиону единиц в минуту. Из всех существующих решений подобной мощностью в вопросе сохранения обладает лишь Autopsy.

Были предложены оптимизации применяемых в Autopsy алгоритмов сортировки и поиска. Сортировка была ускорена посредством применения алгорима k-way merge sort, позволяющего не записывать большие объёмы данных в оперативную память для их упорядочения. Поиск в уже отсортированных файлах производился бинарный, имеющий хорошую асимптотику по времени работы. Для его применения данные в файлах базы записаны со специальным выравниванием, предложенным автором работы.

Работа над данной задачей может быть продолжена. Так, может быть интересным производить сравнение хеш-значений фрагментов файлов, подобно тому, как Photo DNA делает это для поиска детской порнографии. Помимо аналогичной задачи разработанная технология может также быть применена для обнаружения плагиата.

Список литературы

[1] Autopsy. Ресурс сайта производителя // http://www.sleuthkit.org/autopsy.

[2] Belkasoft. Ресурс сайта производителя // http://ru.belkasoft.com/.

[3] MongoDB Limits and Thresholds. // http://docs.mongodb.org/manual/reference/limits/.

[4] Nakhli C. Sorting big files using k-way merge sort // http://www.sinbadsoft.com. — 2013.

[5] Saliba J. Using Common Hashing Algorithms to Identify and Categorize Pictures // http://www.magnetforensics.com. — 2014.

[6] Voluntary Industry Initiatives. // http://www.missingkids.com/Exploitation/Industry.

[7] Wikipedia. MongoDB // Википедия, свободная энциклопедия. — 2015.

[8] Wikipedia. Хеширование // Википедия, свободная энциклопедия. — 2015.

[9] Евсеева О.Н., Шамшев А.Б. Работа с базами данных на языке C#. Технология ADO.NET: Учебное пособие. — УлГТУ, 2009.

[10] Фаулер М., Прамодкумар Дж.С. NoSQL: новая методология разработки нереляционных баз данных. — Вильямс, 2013.