РАЗРАБОТКА ПАРСЕРА ДЛЯ РАСПОЗНАВАНИЯ PDF-ФАЙЛОВ

Л.М. Киселев
Научный руководитель: В.В.Соколова, к.т.н., доцент каф. ПИ
Томский политехнический университет
lmk2@tpu.ru

Введение

В настоящее время формат Adobe PDF является весьма популярным, так как удобно сочетает в себе растровые и текстовые элементы. Однако процесс считывания PDF файла не является тривиальной задачей в виду сложного устройства документа.

В ходе разработки мобильного приложения «Wordlist» для развития вербальной аддтивности перед нами встала задача считывать данные из PDF файла, структурированного особым образом.

Так как основной задачей приложения является работа со списком слов, предоставленных студенту преподавателем в формате PDF файла, был реализован алгоритм считывания необходимых данных из таких файлов.

Структура PDF файла

Текст в PDF файлах, созданных с помощью специальных программ-редакторов состоит из объектов, находящихся внутри тегов “object” и “endobj”, некоторые из которых содержат потоки бинарных данных, зашифрованных в формате GZIP и заключенных между тегами “stream” и “endstream”. В самом потоке и содержится текстовая информация, подлежащая извлечению. Перед тегом “stream” есть строка, содержащая информацию о данном потоке, необходимую для корректной расшивровки потока. Например, свойство Length - длина потока в байтах, FlateDecode - способ шифрования.

```
20 0 obj
<</Filter/FlateDecode/Length 322>>
stream
x[illegible]
endstream
endobj
21 0 obj
```

Рис.1. PDF файл в текстовом редакторе

Расшифрованные потоки, необходимые нам, содержат наборы блоков информации, расположенными между тегами “BT” и “ET”. Внутри данных тегов содержится метаинформация о характеристиках текста и о его расположении, а также сам текст, заключенный в квадратные скобки, внутри которых он хранится в следующем формате:

```
W* n
BT
/F2 12 Tf
1 0 0 1 71.424 755.16 Tm
[(a)-3( sh)-5(a)-3(re)9(h)-3(o)-3(ld)8(e)-3(r)] TJ
ET
Q
```

Рис.2. Блок текста из расшифрованного потока

Для того, чтобы получить текст, который можно зашифровать 128 первыми символами кодировки ASCII, нам достаточно склеить все символы, находящиеся в круглых скобках внутри квадратных скобок.

Текст из иных символов хранится в виде последовательностей шестнадцатеричных цифр, заключенных в треугольные скобки. Для перевода этих цифр в символы, необходимо использовать карту соответствия кодов символам, которая имеет следующий вид:

```
>> def
/CMapName /Adobe-Identity-UCS def
/CMapType 2 def
1 begincodespacerange
<0000> <FFFF>
endcodespacerange
3 beginbfchar
<0003> <0020>
<0005> <0022>
<000F> <002C>
endbfchar
2 beginbfrange
<0011> <0012> <002E>
<0045> <0046> <0062>
endbfrange
1 beginbfchar
<0048> <0065>
endbfchar
3 beginbfrange
<004B> <004C> <0068>
<0050> <0052> <006D>
<0055> <0058> <0072>
endbfrange
1 beginbfchar
<00B6> <2019>
endbfchar
1 beginbfrange
<025A> <0279> <0430>
endbfrange
4 beginbfchar
<027A> <0451>
<0525> <0259>
<075C> <028A>
<0796> <02C8>
endbfchar
endcmap
CMapName currentdict /CMap defineresource
```

Рис.3. Расшифрованный поток словаря

Два шестнадцатеричных числа, находящихся между тегами “begincodespacerange” и “endcodespacerange” задают диапазон шифрования, так же указывая количество цифр, обозначающих один символ.

Между тегами “beginbfchar” и “endbfchar” находятся пары шестнадцатеричных чисел, первое из которых задает номер символа в локальной шифровке, а второе – соответствующий символу код Юникода.

Между тегами “beginbfrange” и “endbfrange” находятся три шестнадцатеричных числа, задающие символьный интервал. Первые два из них задают, соответственно, начало и конец локального интервала, а третье – Юникод-кодировка первого символа интервала. Далее этому в местной кодировке ставится в соответствие первое число из

местного интервала, затем следующему символу в Юникод кодировке ставится в соответствие второй символ из местного интервала, следующему, третий, и так до конца интервала.

Алгоритм парсинга PDF

Ввиду ограниченного размера оперативной памяти мобильных устройств, было принято решение реализовать парсинг потоково, не сохраняя промежуточных данных и проходя файл 1 раз.

Парсинг PDF файла происходит одновременно в двух параллельных потоках, связанных конвеером.

Сначала начитает работу парсер, ищущий ключевое слово Length, чтобы затем считать весь поток как один массив байт. Затем он с помощью декодера дешифрует поток, получая новый массив байт и пишет этот массив в конвеер, которым он связан со следующим потоком.

Другой поток получает порциями данные, вытаскивая из которых текст, он составляет сырой текст. Когда Экстрактор наткнется на словарь, он создает специальные структуры данных, хранящие и дополняющие словарь.

Когда поток данных в парсере закроется и экстрактор пройдет через последнюю порцию данных, он заменит все локальные коды символов на сами символы с помощью словаря и разобьет полученную строку на строки списка слов, записывая их в Базу Данных.

Заключение

Как результат, был разработан и реализован на Java алгоритм, чье основное преимущество заключается в малом потреблении оперативной памяти независимо от размера PDF файла.

Список использованных источников

1. Document management. [Электронный ресурс]. – Режим доступа: https://www.adobe.com/content/dam/acom/en/devnet/pdf/PDF32000_2008.pdf, свободный (дата обращения 18.11.2017).
2. Текст любой ценой: PDF – Хабрахабр [Электронный ресурс]: [офиц.сайт] / ТМ, 2006-2018. Режим доступа: https://habrahabr.ru/post/321050/, свободный. (дата обращения: 14.10.2017).

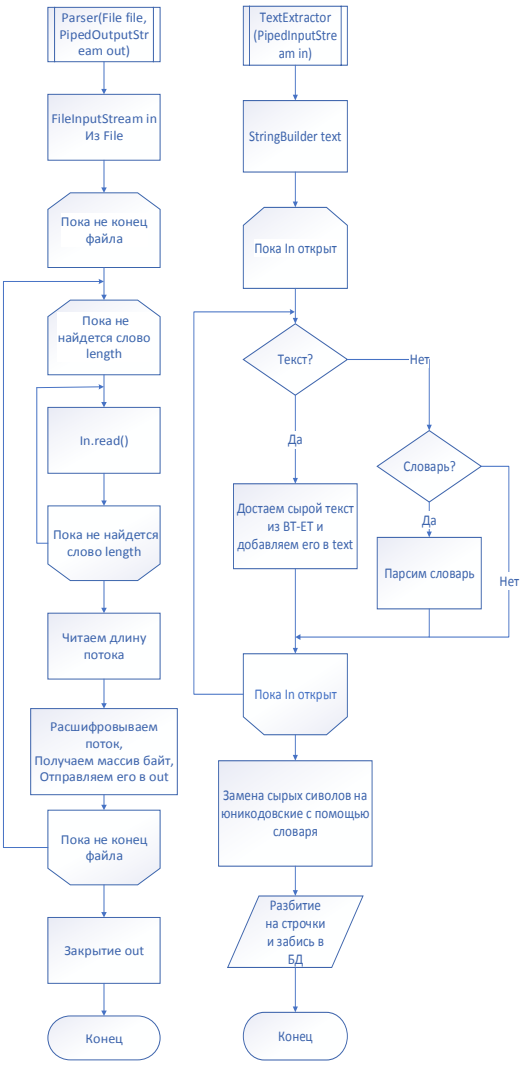

Рис.4. Блок-схема работы алгоритма