Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра алгоритмических языков

Шрамов Георгий Николаевич

# Уменьшение занимаемой оперативной памяти парсером зависимостей

Москва, 2015

# Введение

Первая версия парсера зависимостей во время работы занимала немногим меньше 1 ГБ оперативной памяти, из-за чего его использование было весьма затруднено на слабых компьютерах. Поэтому первоочередной задачей являлось уменьшение этого показателя. Этому и посвящена данная работа, в результате которой удалось уменьшить расход памяти примерно до 20 МБ. Далее пойдет речь о том, как именно это было сделано.

# Детали реализации

## Исходный словарь

Из 1 ГБ занимаемой памяти большая часть приходилась на хранение словаря. В качестве исходного словаря были и остались словари из проекта openCorpora [1]. В старой версии программы все слова исходного словаря просто хранились в хеш-таблице (класс *QMultiHash*), из-за чего и занимали очень много оперативной памяти. За построение и компактное хранение во внешней памяти этой таблицы отвечала вспомогательная программа *genhashtable*.

В новой версии программа *genhashtable* была полностью удалена и с нуля написана новая — *gendict* (причем без использования библиотеки Qt). Она отвечает за преобразование исходного словаря openCorpora в новое внутреннее представление, о котором речь пойдет ниже. Остальные файлы исходного проекта подверглись совсем незначительным изменениям, связанными с новыми интерфейсами, отличными от аналогичных в классе *QMultiHash*.

## Внутреннее представление словаря

Многие идеи преобразования исходного словаря во внутреннее представление были взяты из проекта pymorphy2. Подробно о них можно прочитать в [2], здесь же я приведу только основные позиции.

Все теги (можно встретить и другое название, грамматическая информация), префиксы и суффиксы хранятся в обычных массивах. Поскольку их немного, то они занимают очень мало места. Для парадигм был создан отдельный класс `Paradigm`, который, по своей сути, представляет обычный массив чисел, но с некоторыми особенностями интерфейса. Все парадигмы (коих около 3000) хранятся в массиве.

Слова (их в исходном словаре около 5000000) хранятся в marisa-trie [4]. Это ключевое отличие от реализации в pymorphy2, поскольку там для этих целей используется DAWG [3]. К сожалению, это вынужденная мера, связанная с тем, что в DAWG не

было обнаружено нужного интерфейса поиска ключей, у которых запрос является префиксом (другими словами, аналога функции `predictive_search()` в marisa-trie).

В marisa-trie хранятся не сами слова, а строки вида

```
<слово><разделитель><номер парадигмы><разделитель><номер слова в парадигме>
```

В качестве разделителя выступает обычный пробел. Благодаря такому представлению словаря для получения всех возможных вариантов разбора слова достаточно найти все ключи, начинающиеся с

```
<слово><разделитель>
```

После получения номера парадигмы и номера слова в парадигме восстановить грамматическую информацию (тег) и нормальную форму слова уже несложно. С тегом все совсем тривиально, нормальная форма восстанавливается немногим сложнее. Покажем на примере слова *ПОХОМЯКОВЕЕ* как работает восстановление нормальной формы. После поиска в marisa-trie всех ключей, начинающихся с

```
<ПОХОМЯКОВЕЕ> <пробел>
```

мы узнаем, что это слово изменяется по парадигме

```
ПРЕФИКС          ХВОСТ   ТЕГ
                 ый      ADJF,Qual masc,sing,nomn
                 ого     ADJF,Qual masc,sing,gent
                 ы       ADJS,Qual plur
                 ее      COMP,Qual
                 ей      COMP,Qual V-ej
    по           ее      COMP,Qual Cmp2
    по           ей      COMP,Qual Cmp2,V-ej
```

а номер слова в парадигме — 5 (нумерация начинается с 0). В парадигме первая строка и есть нормальная форма. Поэтому в начале отбрасываем от исходного слова *ПОХОМЯКОВЕЕ* префикс и хвост, соответствующие 5 слову в парадигме. Получаем слово *ХОМЯКОВ*. Затем прибавляем префикс и хвост, соответствующие 0 слову в парадигме. Префикс пустой, а вот окончание *ый* прибавить необходимо. В результате получим слово *ХОМЯКОВЫЙ*, которое и будет нормальной формой.

# Выводы

Для тестированя расходуемой памяти использвалась программа **valgrind** [5] (инструмент *massif*). Согласно ее отчету, новая версия программы требует немногим больше 20 МБ оперативной памяти. Поэтому можно говорить примерно о 50-кратном улучшении расхода памяти.

Из этих 20 МБ около 90% уходит на хранение marisa-trie. Это примерно 18 МБ. Для сравнения, DAWG, содержащий ту же информацию, занимает около 4 МБ. Поэтому если получится сделать в DAWG поиск по префиксу, то можно еще сократить расход памяти. Для еще более существенного эффекта можно кодировать букву русского алфавита 5 битами (отбросив букву Ё), вместо 2 байтов, которые требуются сейчас из-за использования юникода.

# Литература

[1] http://opencorpora.org

[2] https://pymorphy2.readthedocs.org/en/latest/internals/dict.html

[3] https://code.google.com/p/dawgdic/

[4] http://marisa-trie.googlecode.com/svn/trunk/docs/readme.en.html

[5] http://valgrind.org/