



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 10 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 rymorphy2. Подробно о них можно прочесть 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.

Для хранения непосредственно слов (их в исходном словаре около 5000000) было опробовано 2 способа: marisa-trie [4] и DAWG [3]. При использовании marisa-trie весь словарь занимал около 19 МБ, а при использовании DAWG — примерно 9 МБ. Поэтому в финальной версии предпочтение было отдано именно DAWG.

В DAWG 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 тем же все совсем тривиально, нормальная форма восстанавливается немногим сложнее. Покажем на примере слова *ПОХОМЯКОВЕЕ* как работает восстановление нормальной формы. После поиска в DAWG 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 тестирования расходуемой памяти использовалась программа **valgrind** [5] (инструмент *massif*). Согласно ее отчету, новая версия программы требует около 10 МБ оперативной памяти, из которых примерно 90% (т.е. 9 МБ) уходит на хранение словаря в DAWG. Таким образом, расход памяти был уменьшен примерно в 100 раз. Кроме того, за счет эффективности реализации префиксного поиска в DAWG, удалось также немного увеличить скорость работы программы.

# 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/>