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

Шрамов Георгий Николаевич

Оптимизация использования оперативной памяти для хранения морфологической информации слова

Москва, 2015

Введение

Первая версия парсера зависимостей во время работы занимала 0.993 ГБ оперативной памяти, из-за чего его использование было затруднено на компьютерах с недостаточным количеством оперативной памяти. В результате нашей работы удалось уменьшить расход оперативной памяти примерно до 9.358 МБ.

Детали реализации

Исходный словарь

Из 0.993 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 (5081042 слов) хранится в хеш-таблице (класс *QMultiHash*), из-за чего и занимает очень много оперативной памяти. За построение и компактное хранение во внешней памяти этой таблицы отвечает вспомогательная программа *genhashtable*.

Представление словаря в нашей программе

Наш метод преобразования исходного словаря во внутреннее представление основан на идеях проекта pymorphy2 [1]. Вся грамматическая информация о слове, префикс и суффикс и парадигмы слова хранятся в 4 массивах, занимающих 183.8кб, 9.2Кб, 36.3кб и 390.7кб, соответственно. Парадигма слова содержит индексы, указывающие на грамматическую информацию для каждой словоформы данного слова, а также префиксы и суффиксы. Для хранения парадигм мы создали отдельный класс `Paradigm`, который отображает индекс парадигмы в информацию о парадигме слова. В нашей программе сгенерировались 2914 парадигмы.

В новой версии программы мы удалили *genhashtable* и вместо неё добавили новую программу — *gendict*, которая также сжимает исходный словарь openCorpora. В файлах model.cpp и model.h исходного проекта мы добавили функцию, считывающую парадигму входного слова из словаря DAWG. Для удобства отладки мы также использовали std::string и char*.

В DAWG хранятся не сами слова, а строки вида

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

. В качестве разделителя мы использовали пробел, например "красивый 123 0"или "красивая 123 3". Благодаря такому представлению словаря для получения всех возможных вариантов разбора слова достаточно найти все ключи, начинающиеся с

[1] http://opencorpora.org

```
<слово><разделитель>
```

Использование словаря

На вход словарю подаётся слово в кодировке utf8. После этого в словаре DAWG мы находим номера парадигмы N_p и номера слова N_w в парадигме. После этого мы находим префикс слова в массиве префиксов, и суффикс в массиве суффиксов. Например, для восстановления нормальной формы слова *ПОКРАСИВЕЕ* в DAWG ищутся все ключи, начинающиеся с

```
<ПОКРАСИВЕЕ> <пробел>
```

В результате программа узнаёт, что это слово изменяется по парадигме

```
ПРЕФИКС          СУФФИКС    ГРАММАТИЧЕСКАЯ ИНФОРМАЦИЯ
                   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
```

В парадигме первая строка считается нормальной формой, а нумерация начинается с 0. Поэтому номер исходного слова в парадигме — 5. Для генерации нормальной формы слова *ПОКРАСИВЕЕ* отбрасываются префикс и суффикс, и получается основа *КРАСИВ*. Затем прибавляются префикс и суффикс, соответствующие нормальной форме в парадигме. В результате получается слово *КРАСИВЫЙ*.

Эксперимент

Для тестирования расходуемой памяти использвалась программа **valgrind**[2] (инструмент *massif*). Для хранения слов мы испытали 2 способа: marisa-trie[3] и DAWG[4]. При использовании marisa-trie весь словарь занимал около 19 МБ. Мы получили, что версия программы на основе DAWG требует 9.358 МБ оперативной памяти, из которых 93% (т.е. 9 МБ) уходит на хранение словаря в DAWG. Таким образом, нам удалось оптимизировать расход памяти в 106.1 раз. Кроме того, благодаря эффективной реализации префиксного поиска в DAWG, удалось также немного увеличить скорость работы программы по сравнению с исходной версией на хэше. В финальной версии программы предпочтение было отдано DAWG.

[2] http://valgrind.org/

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

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

Литература

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