Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем

Зиновьева Анна Геннадьевна

Реализация возможности сжатия строки в КС-грамматику в YaccConstructor

Курсовая работа

Научный руководитель:
ст. преп, к. ф-м. н. Григорьев С. В.

Санкт-Петербург
2017

Оглавление

Введение

В информации очень часто можно наблюдать иерархические зависимости. Например, текст состоит из слов, фраз и предложений, которые могут повторяться, и намного выгоднее указать, что в нескольких местах у нас один и тот же фрагмент текста и описать его один раз, а не для каждого случая отдельно. Компьютерные программы состоят из модулей, функций, высказываний. Такие зависимости также можно увидеть в музыке. Также, такой подход удобен в биоинформатике, например, при анализе и хранении ДНК и РНК, ведь они являются последовательностями символов из алфавита мощностью всего 4 элемента, а также имеют много общего у живых организмов, и чем ближе эти организмы биологически (из одного типа, семейства, рода и тем более, если родственники), тем больше одинаковых подпоследовательностей и, следовательно, зависимостей будет в их ДНК и РНК.

YaccConstructor [2] - исследовательский проект лаборатории языковых инструментов JetBrains, применяемого для задач лексического и синтаксического анализа. В нем уже реализованы интерфейсы для разбора грамматик в форматах .yrd, .g и .fsy . К существующей функциональности было решено добавить возможность построения грамматики из строки посредством нахождения в ней иерархических зависимостей.

1. Постановка задачи

Целью данной работы является реализация возможности сжатия текстовой строки в грамматику проекте YaccConstructor. Были поставлены следующие задачи:

- реализовать алгоритм сжатия строки;
- реализовать возможность построения общей грамматики для нескольких строк;
- реализовать конечное представление грамматики в формате YARD.IL [3];
- оформить фронтенд для данного алгоритма в рамках YaccConstructor;
- протестировать решение;
- проверить эффективность данного решения.

2. Обзор

2.1. Алгоритм Sequitur

Для решения данной задачи был предложен алгоритм Sequitur [1] для сжатия последовательностей из дискретных символов. Данный алгоритм строит контекстно-свободную грамматику из последовательности символов, обрабатывая в ней все возможные повторы фрагментов. Контекстно-свободная грамматика — грамматика с правилами вида: NonTerminal -> sequence of (Terminal | NonTerminal).

Опишем суть алгоритма. Посимвольно сканируется последовательность терминалов, и строится список всех диграм (пар символов, стоящих рядом). Например, для строки “abcd” список диграм будет выглядеть так:

[“ab”, “bc”, “cd”]

У этого алгоритма есть два принципа:

1. Уникальность — никакая пара символов не встречается дважды. Каждый раз, когда какая-то пара символов появляется во второй раз, обе пары в последовательности заменяются на нетерминальный символ, список правил соответственно меняется, и алгоритм продолжает работу.

2. Полезность — все правила используются не менее двух раз. Если же среди созданных нетерминалов встречается такой, что используется только один раз, то есть по сути является бесполезным, то он заменяется его определением.

Рассмотрим работу алгоритма на строке “abcabc” Будем сканировать строку и добавлять в список новые диграмы

S → a digrams: [ ]

S → ab digrams: [“ab”]

S → abc digrams: [“ab”, “bc”]

S → abca digrams: [“ab”, “bc”, “ca”]

S → abcab

Вот тут мы получили новый диграм, который уже встречался рань-

ше. Создаем новое правило A → ab, заменяем пары в последовательности и изменяем соответственно список диграм. То есть:

S → AcA; A → ab digrams: [“Ac”, “cA”]

S → AcAc

Снова получили диграм, который уже встречался, создали правило B → Ac и заменили пары на нетерминал B.

S → BB; B → Ac; A → ab digrams: [“BB”, ”Ac”, ”ab”]

Но теперь правило A → ab используется только один раз, поэтому избавляемся от него.

Таким образом, получаем:

S → BB; B → abc digrams: [“BB”, ”ab”, ”bc”]

Последний список правил и будет искомой грамматикой и результатом работы алгоритма.

2.2. YARD

YARD — язык спецификаций грамматик, являющийся частью YaccConstructor. Он позволяет использовать EBNF (Расширенную Форму Бэкуса-Наура), метаправила и условную генерацию. Также поддерживает грамматики с L и S атрибутами.

Для описания конечного результата программы требуется два вида выражений:

- выбор — для реализации сжатия нескольких строк;
- конкатенация — для описания самих правил грамматики.

Поддерживая EBNF, YARD позволяет описывать контекстно-свободные грамматики, и, значит, позволяет представить результат алгоритма в нужной форме.

3. Основная часть

3.1. Реализация

Первый вопрос, который надо было решить, какой структурой можно представить грамматику. Нам необходимо выполнять следующие операции:

1. добавление символа в правило;

2. замена двух терминальных символов на нетерминальный.

Так как эти действия будут повторяться постоянно, нужна структура с эффективной поддержкой таких действий. Например, двусвязный список. В такой структуре оба действия будут выполняться за константное время.

Во-вторых, когда мы находим диграм, который встречался уже до этого, мы подчиняем его правилу, после чего мы обязаны рассмотреть его соседей, если возможно подчинить их какому-либо правилу. То есть из состояния, когда мы рассматривали первый диграм, мы переходим в состояние рассмотрения соседних диграмов, которые тоже могут повлечь за собой просмотр уже их соседей. То есть образуется стек состояний, которые надо обработать. Этот принцип и был взят за основную функцию в реализации алгоритма, которая работает со стеком состояний.

И третий вопрос, как реализовать возможность построения общей грамматики для нескольких последовательностей, склеенных через специальный символ (например, символ &). Был найден такой выход. Немного модифицируем алгоритм. При разборе стека состояний, каждый раз, когда нам встречается диграм, который содержит &, мы не будем его обрабатывать: то есть не будем добавлять в список диграм, и, следовательно, смотреть, встречался ли такой диграм раньше (ведь его не может быть в списке). Так как все диграмы, состоящие из обычных символов, стоящих перед и после специального символа &, будут при-

сутствовать в нашем списке, то, значит, они будут участвовать в образовании грамматики, и алгоритм будет работать корректно.

Также, был оформлен интерфейс для алгоритма — фронтенд к YaccConstructor. В нем реализована возможность сжать обычную строку, несколько строк, разделенных специальным символом, и массив строк. Представление YARD.IL, в виде которого возвращается грамматика, строится так:

- если символ терминальный, применяется конструктор PTok;
- если нетерминальный, то PRef;
- последовательность строится с помощью PSeq;
- а возможность сжатия нескольких строк реализуется через PAlt.

3.2. Эксперимент

Для проверки эффективности данного решения была взята последовательность 16s РНК бактерий. Каждая последовательность состоит примерно из 1500 нуклеотидов, то есть последовательность символов из алфавита {A; C; G; T}

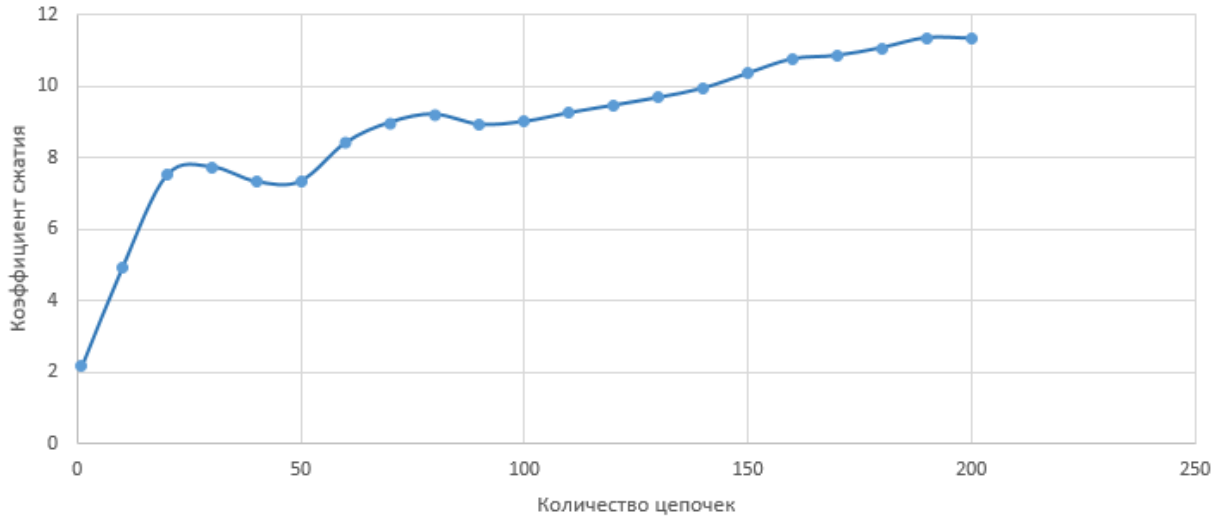

Рис. 1: График зависимости сжатия входных данных

На графике показана зависимость коэффициента сжатия от количества сжимаемых последовательностей. Коэффициент сжатия вычисляется как отношение количества символов во всех последовательностях изначальных цепочек к количеству символов во всех правилах полученной грамматики. Как можно заметить, при сжатии небольшого количества строк, коэффициент получается маленьким, но потом быстро вырастает, и дальше, при росте количества цепочек РНК, коэффициент тоже увеличивается, хотя и немного колеблется.

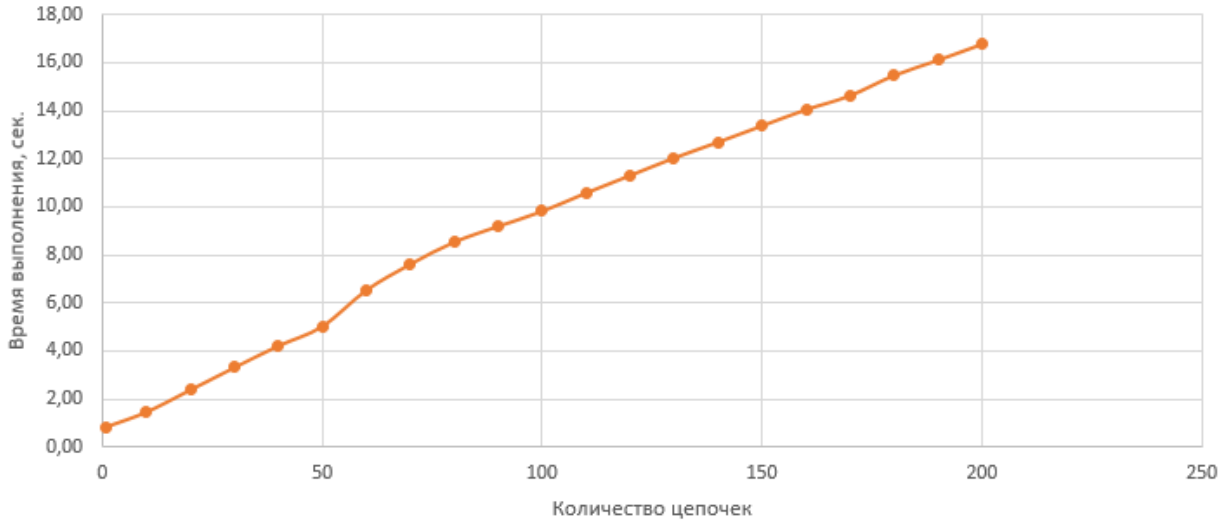

Рис. 2: График зависимости времени выполнения программы

Это график зависимости времени выполнения программы от входных данных. На нем видно, что сложность реализованного алгоритма O(n).

Таким образом, можно назвать алгоритм достаточно эффективным.

4. Заключение

В ходе данной работы было сделано следующее:

- реализован алгоритм Sequitur;
- реализована возможность построения общей грамматики для нескольких срок;
- оформлен фронтенд к YaccConstructor для построения грамматики из строки в формате YARD.IL;
- проведено тестирование с помощью системы NUnit;
- проверена эффективность и производительность данного алгоритма.

В дальнейшем планируется использовать данный проект для метагеномных сборок, а также компактного представления больших объемов данных.

Список литературы

[1] Nevill-Manning C.G., Witten I.H. Identifying Hierarchical Structure in Sequences: A linear-time algorithm. — 1997.

[2] YaccConstructor. YaccConstructor // YaccConstructor official page. — URL: `http://yaccconstructor.github.io`.

[3] YaccConstructor. YARD // YaccConstructor official page. — 2015. — URL: `http://yaccconstructor.github.io/YaccConstructor/yard.html`.