Аннотированные суффиксные деревья: особенности реализации

Михаил Дубов[1], Екатерина Черняк[2]

[1]Отделение программной инженерии НИУ ВШЭ, Москва, Россия. msdubov@gmail.com
[2]Отделение прикладной математики и информатики НИУ ВШЭ, Москва, Россия. ek.chernyak@gmail.com

Аннотация. В статье описываются особенности эффективной программной реализации разработанной с участием одного из авторов модификации суффиксных деревьев, предполагающей аннотацию узлов дерева частотами встречаемости соответствующих им подстрок в исходной коллекции текстов. Данная структура данных имеет ряд практически важных приложений, таких как оценка степени вхождения последовательности символов в текст или анализ связей между ключевыми словосочетаниями. Предложенные в данной работе модификации известных алгоритмов быстрого построения обычных суффиксных деревьев, а также описываемые в ней приемы хранения аннотированных суффиксных деревьев в памяти делают возможным их практическое применение для анализа больших коллекций текстов, что подтверждается приводимыми в тексте статьи результатами сравнительного исследования производительности различных реализаций метода АСД на реальных данных.

Ключевые слова: анализ текстов, аннотированные суффиксные деревья, обобщенные суффиксные деревья, алгоритм Укконена, комбинированные алгоритмы, суффиксные массивы.

Выходные данные сборника.
© Национальный Открытый Университет «ИНТУИТ», 2013

Введение

Среди многообразия методик анализа текстов метод аннотированного суффиксного дерева (далее – АСД) выделяется тем, что принадлежит к немногочисленному классу моделей, представляющих текст как последовательность символов, но не как последовательность слов. Это позволяет достичь высокой степени независимости от языка анализируемых текстов, а также исключает необходимость выполнять их предобработку, которая зачастую оказывается довольно трудоемкой.

В основе метода АСД лежит построение суффиксного дерева специального вида, позволяющего оценивать степень вхождения ключевых словосочетаний в исходный корпус текстов. Получаемая оценка может использоваться в таких приложениях, как фильтрация спама [2], анализ связей между ключевыми словосочетаниями или анализ структуры корпуса текстов путем их иерархической группировки [3].

До настоящего времени в публикациях практически не уделялось внимания эффективной программной реализации рассматриваемой структуры данных. Между тем, производительная реализация метода АСД чрезвычайно важна для анализа крупных коллекций текстов. Основным способам достижения высокой производительности метода АСД и посвящена настоящая работа.

В данной научной работе использованы результаты, полученные в ходе выполнения проекта «Методы визуализации текстовой информации с помощью построения суффиксных деревьев, мультифасетных классификаций и иерархических онтологий: алгоритмическое и программное обеспечение», выполненного в рамках Программы «Научный фонд НИУ ВШЭ» в 2013 году, грант № 13-05-0047.

Наивная реализация метода АСД

Приведем краткое изложение метода АСД. Более подробное его описание читатель может найти в [3].

Основой для анализа набора текстов по методу АСД являются собственно построение обобщенного АСД для поступающей на вход коллекции строк, а также наложение на полученное АСД ключевых словосочетаний.

В существующих публикациях [2][3] предполагается организация АСД как корневого дерева, в котором каждый узел (кроме корня) помечен одним из символов алфавита, на котором определена входная коллекция строк. АСД при этом кодирует все суффиксы всех строк в коллекции: им соответствуют пути от корня до листьев дерева. Каждый узел v в дереве помечен целым числом $f(v)$, обозначающим количество

вхождений соответствующего фрагмента строки (от корня до данной вершины) в исходный набор текстов. Пример такого дерева для коллекции, состоящей из одной строки “XABXAC”, приведен на рис.1.

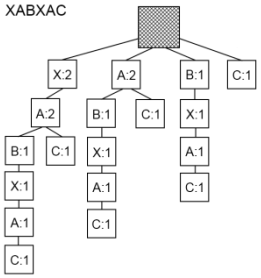


Рис. 1. Наивная реализация АСД

Для построения АСД предлагается следующий алгоритм.

Алгоритм **NaiveConstruction**(C)
Вход. Коллекция строк $C = \{S_1 \dots S_m\}$.
Выход. Обобщенное АСД для C.

```
1.  for i ← 1 to m
2.    for j ← 1 to n_i = |S_i|
3.      do k ← длина совпадения суффикса S_i[j:] с АСД
4.          for узел u из S_i[j:(j + k − 1)]
5.              do присвоить f(u) ← f(u) + 1
6.          for l ← j + k to n_i
7.              do вставить узел v
8.                  присвоить f(v) ← 1
```

Анализ трудоемкости данного алгоритма достаточно прост. Для каждой строки мы посимвольно просматриваем все ее суффиксы, затрачивая, таким образом, на i-ю строку длины n_i количество операций, пропорциональное $(1 + 2 + \cdots + n_i) = \Theta({n_i}^2)$. Общее время работы алгоритма для коллекции из m строк, таким образом, может быть оценено как $\Theta({n_1}^2 + \cdots + {n_m}^2)$, или, если использовать несколько более грубую оценку, $\boldsymbol{O(mn_{max}^2)}$. Отметим, что обустроенное описанным выше способом АСД невозможно построить с использованием меньшего числа операций, так как само оно занимает в памяти место, квадратично зависящее от длины закодированных в нем строк.

На втором этапе метода АСД оценивается степень вхождения ключевых словосочетаний в исходную коллекцию слов с помощью их

«наложения» на построенное дерево. Результатом этой оценки является число в интервале $[0; 1]$, характеризующее степень вхождения строки в коллекцию текстов, на основе которой было построено АСД (например, при наложении на АСД на рис. 1 строки “ABC” оценка составит 0.25).

Для оценки вхождения строки S в коллекцию текстов (или, что то же самое, в дерево T) необходимо выполнить следующие шаги [3]. Назовем условной **вероятностью узла** $\boldsymbol{v}$ число $\hat{\boldsymbol{p}}(\boldsymbol{v}) = f(v)/f(parent(v))$, где $f(v)$ – частота узла v, $f(parent(v))$ – частота узла-родителя v (частотой корня будем считать сумму частот узлов на первом уровне дерева). Пусть для некоторой строки s максимальное ее совпадение с символами в АСД по любому пути от корня к листьям имеет длину k. В таком случае оценка совпадения $v_1 \dots v_k$ вычисляется как сумма условных вероятностей входящих в него узлов, нормированная по длине совпадения:

$$score(s) = \frac{\sum_{i=1}^{k} \hat{p}(v_i)}{k}$$

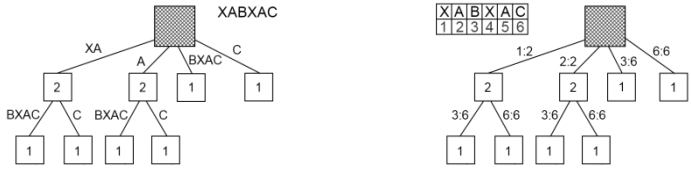
Наконец, степень вхождения строки S в АСД вычисляется как сумма оценок совпадений ее суффиксов, нормированная по длине строки:

$$SCORE(S) = \frac{\sum_{i=1}^{|S|} score(S[i:])}{|S|}$$

Завершая данный раздел, упомянем важное **свойство АСД**: частота любого его внутреннего узла равна сумме частот соответствующих дочерних узлов. Это свойство будет использовано в алгоритме быстрого построения обобщенного АСД, который мы опишем далее.

Построение АСД за линейное время

Отметим, что описанная выше структура данных, называемая в [2] и [3] аннотированным суффиксным деревом, строго говоря, суффиксным деревом не является. Действительно, в ней нарушено одно из основных свойств суффиксных деревьев [1]: в большом количестве присутствуют узлы с единственным потомком, в то время как в суффиксном дереве у каждой внутренней вершины, отличной от корня, должно быть не менее двух детей. Выполнить это условие можно, только схлопнув каждую цепь из узлов с единственным потомком в одну вершину и пометив входящее в нее ребро конкатенацией символов, которыми были помечены узлы в этой цепи. Частота самой вершины остается неизменной, так как у всех вершин в цепи она была одинаковой. Преобразовав таким образом исходное дерево, получим структуру данных, изображенную на рис. 2а (дерево вновь построено для строки “XABXAC”):



а) Схлопывание вершин б) Сжатие меток

Рис. 2. Оптимизация представления АСД

Реализованное таким образом дерево все еще требует $O(mn_{max}{}^2)$ памяти из-за необходимости хранить все метки ребер в явном виде. Существует, однако, простой прием сжатия дуговых меток, позволяющий снизить количество используемой деревом памяти до линейного относительно длин строк в коллекции. Он заключается в том, чтобы хранить в каждом ребре только индексы начала и конца соответствующей подстроки, а не всю подстроку в явном виде [1]. Окончательный вид АСД после всех описанных оптимизаций представлен на рис. 2б.

Отметим, что, несмотря на уменьшение количества вершин в дереве, вычисление степени вхождения в него строк все еще остается возможным. Все, что требуется – это немного видоизменить формулу оценки совпадения строки с АСД:

$$score(s) = \frac{\left(\sum_{i=1}^{k} \hat{p}(v_i)\right)+l-k}{l},$$

где k, как и ранее – число узлов в совпадении, а l – фактическая длина совпадения в символах.

Возможность столь значительного снижения количества используемой для хранения АСД памяти позволяет также рассчитывать на существование асимптотически менее трудоемких алгоритмов построения АСД, чем тот, который был описан выше. Действительно, существует целый ряд линейных по времени алгоритмов построения обычных (неаннотированных) суффиксных деревьев. В литературе в качестве основных принято выделять алгоритмы П. Вайнера (1973), Э. МакКрейга (1976) и Э. Укконена (1995) [1]. Определенную проблему, однако, представляет построение с помощью этих алгоритмов аннотированных суффиксных деревьев, где узлы помечены частотами соответствующих им подстрок. Дело в том, что временная эффективность всех этих алгоритмов достигается путем игнорирования на каждом шаге определенных путей в дереве, что делает невозможным своевременное обновление меток узлов, как это происходит в наивном алгоритме. Оказывается, однако, что аннотировать дерево можно и после его построения, если предварительно выполнить несложную предобработку коллекции строк:

Алгоритм **LinearConstruction**(C)
Вход. Коллекция строк $C = \{S_1 \dots S_m\}$.
Выход. Обобщенное АСД для C.

1. Построить $C' = \{S_1\$_1 \dots S_m\$_m\}$, где $\$_i$ - уникальные символы.
2. Построить обобщенное суффиксное дерево T для коллекции C', используя алгоритм с линейной сложностью.
3. **for** l **in** $leaves(T)$
4. **do** присвоить $f(l) \leftarrow 1$
5. Выполнить **обход** дерева T **снизу вверх**; в каждом внутреннем узле v присвоить $f(v) \leftarrow \sum_{u \in T:\, parent(u)=v} f(u)$.

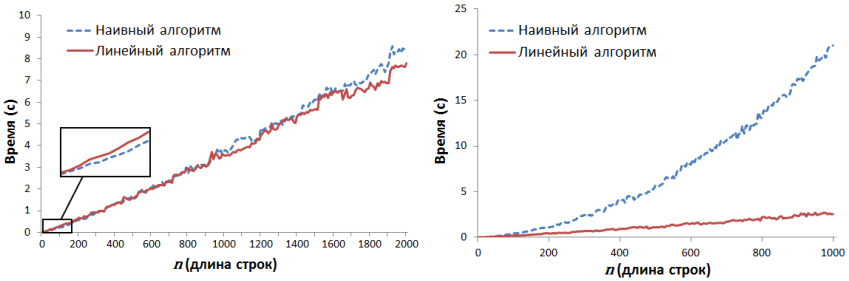
Основанием для аннотирования в шаге 3 всех листьев числом 1 является тот факт, что на первом шаге мы приписываем к каждой строке в коллекции уникальный символ – а значит, каждая подстрока, соответствующая одному из путей от корня до листа, встречается в исходном наборе строк только один раз. Метод присвоения меток внутренним вершинам в шаге 5 напрямую следует из описанного выше свойства АСД. Обход дерева при этом требует времени, пропорционального числу вершин. Таким образом, все шаги алгоритма выполняются за линейное время, и общая оценка его трудоемкости составляет $\Theta(n_1 + \dots + n_m)$ или $\boldsymbol{O(mn_{max})}$.

Сравнительный анализ алгоритмов построения АСД

Обратимся к экспериментальному исследованию производительности описанных выше алгоритмов. В качестве основы для экспериментов выступала реализация метода АСД на языке Python 2.7 (без использования дополнительных библиотек вроде NumPy). Основой для линейного алгоритма стал алгоритм Укконена, как наиболее простой в реализации.

Важно отметить, что с использованием описанного выше оптимизированного способа хранения АСД в памяти оценка трудоемкости наивного алгоритма становится квадратичной только в худшем случае. Действительно, теперь для каждой части суффикса, которой нет в дереве, алгоритму требуется добавить в дерево не целый ряд вершин, а только один лист, дуга к которому помечена отсутствующими в дереве символами. В ряде случаев это делает наивный алгоритм сопоставимым с линейным, а иногда и превосходящим его по трудоемкости (как в случае построения АСД для коллекции случайно генерируемых строк, в структуре которых отсутствуют какие-либо закономерности, рис. 3а). Это объясняется тем, что алгоритм Укконена организован сложнее наивного и требует для своей работы ряда накладных расходов (например, на организацию в дереве так называемых суффиксных связей).

Худшим же случаем для наивного алгоритма является такая коллекция строк, при обработке которой он вынужден на каждой итерации опускаться на всю глубину уже построенной части дерева. Примером такой коллекции может быть набор строк, различающихся только одним-двумя последними символами. На таких входных данных трудоемкость наивного алгоритма действительно деградирует до квадратичной, а выигрыш от использования линейного алгоритма построения АСД становится значительно более очевидным (рис. 3б).



а) случайные строки б) худший случай

Рис. 3. Экспериментальная трудоемкость построения обобщенного АСД для коллекции из 100 строк длины n

Заметим, что и в случае случайных строк при росте числа закодированных в дереве символов наивный алгоритм все больше начинает уступать линейному в скорости: это вызвано тем, что плотность дерева на верхних уровнях растет и наивному алгоритму в среднем приходится глубже опускаться по дереву при добавлении в него новых строк.

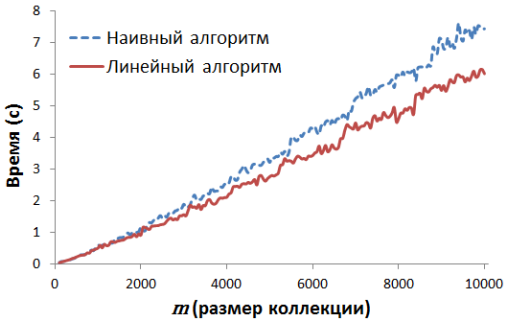


Рис. 4. Экспериментальная трудоемкость построения обобщенного АСД для набора из m строк, полученного из коллекции текстов Reuters [4]

В текстах на естественных языках слова, начинающиеся с одинаковых приставок или имеющие одинаковые основы, встречаются достаточно часто, поэтому мы можем ожидать в среднем большей глубины дерева, чем для случайных строк, и, следовательно, лучшей производительности линейного алгоритма, чем наивного. Действительно, при построении АСД для англоязычной коллекции текстов Reuters-21578 [4] линейный алгоритм в среднем оказался на 25-30% производительнее, чем наивный (рис. 4). Отметим, что при построении АСД тексты из коллекции разбивались на строки по 3 слова (такая предобработка была предложена в [3] для получения более адекватных результатов наложения на дерево ключевых словосочетаний, длина которых, как правило, также не превышает трех-четырех слов).

Комбинированные алгоритмы

Тот факт, что наивный алгоритм оказывается немного производительнее линейного при малой глубине дерева (которая является следствием малого числа закодированных в нем строк), дает основания для попыток построения комбинированных алгоритмов конструирования АСД, использующих преимущества как наивного, так и линейного алгоритмов. Наиболее очевидными представляются следующие стратегии:

- Для каждой следующей строки найти длину ее совпадения при наложении на дерево. Если это число ниже некоторого порога t, то использовать наивный алгоритм, иначе - алгоритм Укконена (точное значение t варьируется в пределах от 1 до 10 и определяется экспериментально);
- Первые k строк из поступающей на вход коллекции добавляются в дерево наивным алгоритмом, остальные – алгоритмом Укконена (точное значение k определяется экспериментально).

Первый алгоритм в ходе экспериментов не дал никакого выигрыша: накладные расходы на проверку вхождения строки в АСД превысили выгоду от комбинирования двух алгоритмов. Второй же алгоритм позволяет достичь небольшого преимущества в скорости: оно составило до 5% на использованной для тестирования коллекции Reuters и до 10-15% на случайных строках. Столь небольшой выигрыш отчасти объясняется тем, что использование наивного алгоритма в комбинации с линейным требует усложнения первого: теперь в наивном алгоритме также необходимо добавлять в дерево суффиксные связи, чтобы обеспечить нормальное выполнение алгоритма Укконена на втором этапе.

Использование суффиксных массивов

Чрезвычайно привлекательным при работе с суффиксными деревьями является переход к использованию суффиксных массивов [1]. Эта структура данных, также как и суффиксное дерево, кодирует все суффиксы строки, но устроена при этом таким образом, что позволяет сократить использование памяти в 2-3 раза.

Одним из последствий перехода к суффиксным массивам, однако, является отказ от понятия «узел», и, как следствие, от родительско-дочернего отношения между узлами. Так как лежащее в основе метода АСД вычисление степени вхождения строк в дерево активно использует понятие «условной вероятности узла», которая является отношением частоты узла к частоте узла-родителя, использование суффиксных массивов при реализации метода АСД представляется нам невозможным.

Выводы

В ходе нашего исследования была разработана программная реализация метода АСД, значительно превосходящая по эффективности реализации, предложенные в более ранних работах. Результаты экспериментального исследования ее производительности при этом наглядно продемонстрировали, что эффективность используемых нами алгоритмов сильно зависит от вида входных данных и, вероятно, до некоторой степени варьируется между коллекциями текстов на разных языках. Изучение этих зависимостей и разработка алгоритмов построения АСД, чувствительных к особенностям входных данных, составляет предмет дальнейших исследований.

Список источников

1. Gusfield D. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, 1997.

2. Pampapathi, R., Mirkin, B., Levene M. A suffix tree approach to anti-spam email filtering. Machine Learning, v.65 n.1, October 2006. p. 309-338.

3. Миркин Б. Г., Черняк Е. Л., Чугунова О. Н. Метод аннотированного суффиксного дерева для оценки степени вхождения строк в текстовые документы. Бизнес-информатика, №3(21), 2012. с. 31-41.

4. Reuters-21578 Text Categorization Test Collection[Электронный ресурс] URL: http://www.daviddlewis.com/resources/testcollections/reuters21578/ (дата обращения: 10.02.2013)