УДК 681.3.07

С.Ф. Свиньин, И.А. Андрианов

ПРИМЕНЕНИЕ РАВНОМЕРНО РАЗРЕЖЕННЫХ СУФФИКСНЫХ ДЕРЕВЬЕВ ДЛЯ ЗАДАЧ ОБРАБОТКИ СТРОК

Свиньин С.Ф., Андрианов И.А. **Применение равномерно разреженных суффиксных деревьев для задач обработки строк.**

Аннотация. Потребность в эффективных алгоритмах обработки строк возникает во многих практических задачах. Одним из наиболее универсальных подходов является применение суффиксных деревьев. Однако, данная структура предъявляет высокие требования к памяти ЭВМ, что ограничивает область её применения. В данной статье на примере задачи о максимальной симметричной подстроке рассматривается способ, позволяющий частично устранить данный недостаток. Описанный способ может быть использован и для других задач.

Ключевые слова: информационный поиск, разреженные суффиксные деревья, ближайший общий предок, максимальный палиндром, алгоритм Укконена.

Svinyin S.F., Andrianov I.A. **Application of evenly sparse suffix tree for string processing tasks.**

Abstract. The need for efficient algorithms for processing strings arises in many practical problems. One of the most universal approaches is the use of suffix trees. However, this data structure has high memory requirements, which limits area of its application. In this article we consider a way to partially eliminate this disadvantage and give an example of solving the problem of the longest symmetric substring. The described method can be also be used for other problems too.

Keywords: information retrieval, sparse suffix trees, least common ancestor, maximal palindrome, Ukkonen's algorithm.

1. Введение. Эффективные структуры данных и алгоритмы для обработки строк и последовательностей являются актуальным объектом исследования в настоящее время, что объясняется потребностями практики в различных областях – информационно-поисковые системы, вычислительная биология и др.

Чтобы добиться высокой скорости работы, обычно используется подход, который заключается в проведении предобработки входных данных, то есть построении индексов над ними. Применение индексов позволяет получить алгоритмы поиска с меньшей вычислительной сложностью, чем в случае полного сканирования входных данных.

Важным вопросом при разработке индекса является выбор минимального индексируемого элемента. При выполнении поиска в документах, не являющихся текстами на естественном языке, удобным механизмом является декомпозиция на основе подстрок – n-грамм, префиксов или суффиксов.

Заметим, что для текстов на естественном языке данный подход также применим, хотя и имеет некоторые особенности. Например, в

статье [1] рассматривается модель на основе n-грамм, которые строятся не над отдельными символами, а над словами текста с учётом морфологии. Вообще, можно считать, что множество слов (словоформ) естественного языка образует алфавит, при этом каждое слово является его символом. С этой точки зрения последовательность слов, по сути, представляет собой символьную строку, к которой применимы все известные алгоритмы обработки строк.

Конечно, реализация такого типа индексов требует больших накладных расходов. Однако, с развитием средств вычислительной техники и совершенствованием алгоритмов становится возможным успешно применять подобные методы для данных всё больших размеров. При этом становится возможным выполнять, в частности, такие виды поиска, как поиск по сходству и регулярным выражениям.

Известно несколько структур данных для построения индексов на основе подстрок – суффиксные деревья, суффиксные массивы, строковые B-деревья, различные способы представления n-грамм (в том числе переменной длины) и др. Одной из наиболее универсальных структур данных является суффиксное дерево (далее СД). Известны по крайней мере несколько десятков задач, для которых может использоваться данная структура [2]. Например, СД может использоваться для поиска подстрок, при этом сложность поиска подстроки p в строке s составляет $O(|p|+occ)$, где $|p|$ – длина p, occ – число вхождений. СД могут быть использованы для эффективного поиска заданного набора строк (как альтернатива алгоритму Ахо–Корасик) и при решении ряда других задач.

2. Основные понятия и обозначения. Суффиксным деревом T для строки S длины m называется дерево с корнем, имеющее ровно m листьев, занумерованных от 1 до m. Каждая внутренняя вершина, отличная от корня, имеет не меньше двух детей, каждая дуга помечена дуговой меткой – парой чисел (i,d), определяющих подстроку $S[i..i+d-1]$ строки S (говорят, что дуга помечена этой подстрокой). Никакие две дуги, выходящие из одной и той же вершины, не могут быть помечены подстроками, начинающимися с одного и того же символа. Для каждого листа i конкатенация меток дуг на пути от корня к листу в точности составляет суффикс строки S, который начинается в позиции i. Пример СД представлен на рисунке 1.

Будем использовать далее следующие обозначения. Узлы СД обозначаются маленькими латинскими буквами. Каждому узлу в дереве соответствует единственная подстрока – конкатенация подстрок на дугах на пути из корня дерева в соответствующий узел. Подстроку,

соответствующую узлу p, будем обозначать как $\overline{p}$. Если s – строка, то $|s|$ – её длина. Аналогично, если S – множество, то $|S|$ – его мощность. Дугу, ведущую из узла u в узел v, будем обозначать как $u \to v$. Соответственно, $|u \to v|$ – длина подстроки, которой нагружена дуга $u \to v$.

Для обращения к параметрам, хранящимся в узлах дерева, будет использоваться точечная нотация, т.е. запись вида $u.parameter$.

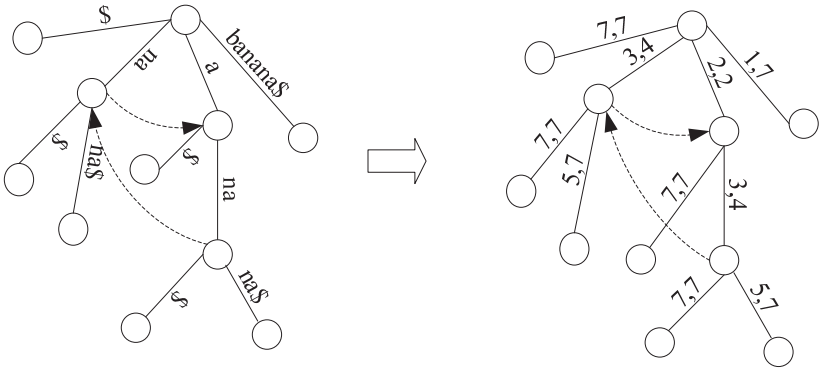

Рис. 1. Пример суффиксного дерева для строки 'banana'. Пунктирными стрелками показаны суффиксные связи

Позицией в СД называется тройка $< u, i, j >$, где u – узел дерева, i и j – индексы подстроки $s[i..j]$, которая определяет дальнейший путь в дереве от узла u до требуемого места в узле или внутри какой-то дуги. В случае, если требуемое место в дереве – сам узел u, то $i > j$.

Одно и то же положение в дереве можно задать по-разному. Позиция $< u, i, j >$, в которой узел u имеет наибольшую глубину, называется канонической.

LCP – наибольший общий префикс (longest common prefix) двух (или более) строк. LCA – ближайший общий предок (least common ancestor) двух данных узлов дерева.

Существует несколько алгоритмов построения суффиксного дерева за линейное время. Для многих задач предпочтительным оказывается алгоритм Укконена. Его описание достаточно объёмно и в данной статье не приводится. Заметим, однако, что для полного понимания нижеприведённых результатов желательно иметь представление о том, как работает алгоритм Укконнена. Подробное его описание можно найти, например, в книге [2].

Построив дерево за линейное время, можно использовать его для решения целого ряда задач. Классическим примером служит задача о подстроке. Задан текст T длины m. За время предобработки $O(m)$ необходимо приготовиться к тому, чтобы, получив неизвестную строку S длины n, за время $O(n)$ (то есть независимо от длины T) определить, входит ли S в T как подстрока или нет. Решение выглядит следующим образом. Построив один раз суффиксное дерево для T, для каждой очередной строки S будем искать совпадения для её символов вдоль единственного пути от корня дерева до тех пор, пока либо S не исчерпается, либо очередное совпадение не станет невозможным. При этом в первом случае каждый лист в поддереве, начинающемся в месте последнего совпадения, определяет своим номером начальную позицию S в T. И наоборот, каждая позиция S в T нумерует такой лист.

3. Основные недостатки суффиксных деревьев и способы их устранения. Основными недостатками суффиксных деревьев являются высокие требования к памяти и плохая пространственная локальность. Несмотря на то, что размер дерева прямо пропорционален размеру исходного текста, коэффициент пропорциональности составляет около 10 – 15 даже в лучших программных реализациях. Это влечёт необходимость в существенных затратах памяти.

Кроме этого, построение больших деревьев затруднено, поскольку известные линейные алгоритмы построения рассчитаны на работу в оперативной памяти и предполагают наличие быстрого доступа к произвольным адресам, вследствие чего они плохо адаптируются под использование внешних носителей.

В результате на обычной вычислительной машине при достаточно скромных размерах входных данных СД перестаёт помещаться в физическую память. Использование же виртуальной памяти резко снижает скорость работы программы.

При работе с уже построенным суффиксным деревом во внешней памяти также часто будут происходить обращения к узлам дерева в разных страницах. В результате этого стратегии кеширования, используемые операционной системой, оказываются малоприменимыми.

Можно выделить два пути решения данной проблемы. Первый заключается в поиске специальных способов представления суффиксного дерева во внешней памяти, чтобы более эффективно использовать особенности блочного обмена с диском и используемый операционной системой способ кеширования (либо реализовать собственный механизм управления кешированием).

Второй способ заключается в использовании разреженных (неплотных) суффиксных деревьев [3], которые строятся не над всеми суффиксами индексируемых текстов, а лишь над некоторым их подмножеством (при этом критерий выбора суффиксов может определяться конкретной задачей). Однако, такая структура данных подходит для решения далеко не всех задач, которые можно решить с помощью обычных СД.

На рисунке 2 приведён пример обобщённого разреженного СД для строк 'abcabe$' и 'bcabce#'. В дерево включены только суффиксы, начинающиеся с символов ‘a’ и ‘b’. Дуги нагружены подстроками исходной строки (дерево слева), реально на дугах хранятся позиции подстрок в исходных строках (дерево справа). В тройках чисел на дугах дерева первое число соответствует номеру строки (1 или 2).

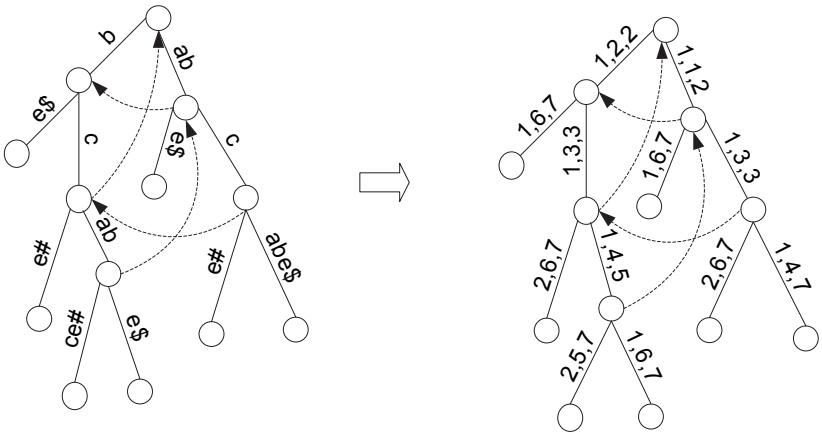

Рис. 2. Пример обобщенного разреженного суффиксного дерева. Пунктирными стрелками показаны суффиксные связи

Эффективное построение разреженного СД для общего случая (нет никаких ограничений на позиции суффиксов) представляет собой достаточно сложную задачу. В полной мере в настоящее время она пока не решена, однако, определённые наработки в данной области имеются. В частности, в статье [4] описывается вероятностный алгоритм построения разреженного СД. Дерево, построенное с его помощью, будет полностью корректным с вероятностью, близкой к единице. Временная сложность алгоритма составляет $O(n \cdot \log^2(b))$, требования к памяти – $O(b)$, где b – количество суффиксов.

Если ввести некоторые ограничения на позиции суффиксов, включаемых в разреженное СД, то становится возможным разработать

более быстрые специализированные алгоритмы. Так, в работе [5] предложен способ построения дерева за линейное время для случая, когда все суффиксы начинаются на границах слов.

Эта идея была обобщена в статье [6], где в качестве входных данных рассматривается последовательность кодовых слов какого-либо префиксного кода (например, кода Хаффмана), а индексные позиции располагаются на границах слов. В [6] предложен алгоритм построения дерева с временем работы $O(n+m)$, где m - размер детерминированного конечного автомата, распознающего данный префиксный код.

4. Равномерно разреженные СД. В работе [3] для уменьшения памяти, занимаемой деревом, предлагается хранить в нём не каждый суффикс исходной строки, а лишь каждый k–й суффикс. При этом объём памяти для хранения такой структуры уменьшается примерно в k раз. Полученную структуру Каркайнен и Укконен назвали равномерно разреженным суффиксным деревом ("evenly spaced sparse suffix tree") степени k.

Авторы статьи [3] показали, что можно модифицировать алгоритм Укконена так, чтобы он строил разреженое суффиксное дерево и при этом временная сложность алгоритма оставалась прежней. Пример равномерно разреженного СД степени $k = 2$ для строки "bananas" показан на рисунке 3.

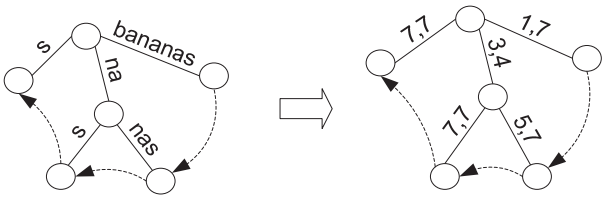

Рис. 3. Пример равномерно разреженного суффиксного дерева степени 2. Пунктирными стрелками показаны разреженные суффиксные связи

В работе [3] предложено несколько изменить определение суффиксной связи по сравнению с оригинальным алгоритмом Укконена. Пусть T – разреженное суффиксное дерево, содержащее каждый k–й суффикс исходной строки, u – некоторый внутренний узел дерева. Разреженная суффиксная связь для узла u определяется следующим образом.

Если $\overline{u} = ap$, где $|a| = k$, то разреженная суффиксная связь для узла u – указатель на узел v, для которого $\overline{v} = p$. Если $|\overline{u}| \le k$, то разре-

женная суффиксная связь для узла u – указатель на корень дерева.

Изменения в алгоритме Укконена, которые необходимы при использовании разреженных суффиксных связей, в статье [3] в явном виде не приводятся. Поэтому опишем свой вариант (вероятно, совпадающий с оригинальным).

Пусть на очередном шаге алгоритма Укконена выполнилось явное продолжение суффикса j, и требуется сделать то же самое с суффиксом $j+k$. Для этого с помощью суффиксной связи перейдём от позиции $< v, first, i >$ к позиции $< u := v.suffixlink, first, i >$.

Если узел u, в который мы перешли, отличен от корня, то мы действительно оказались в позиции $j+k$ –го суффикса в дереве, и остаётся лишь привести позицию к каноническому виду.

Однако, если u – корень дерева, то, согласно определению разреженной суффиксной связи, результирующая позиция может соответствовать любому из суффиксов, начинающихся в $\{j+1,..j+k\}$, и необходимо выполнить её корректировку. Для этого достаточно заметить, что раз u – корень дерева, то суффикс, задаваемый позицией $< u, first, i >$, равен $s[first..i]$. Следовательно, нужно передвинуть $first$ вправо на начало следующего суффикса. Обозначим через t начальное смещение, то есть позицию начала первого суффикса $(1 \le t \le k)$: Тогда изменение $first$ можно найти, например, так:

```
delta := k - (first - t) mod k;
if (delta ≠ k) then first := first + delta.
```

В работе [3] равномерно разреженные СД были предложены для решения задачи о поиске вхождений подстроки в строку для случая, когда полное дерево не помещается в память. Для решения этой задачи в [3] предложен алгоритм, учитывающий особенности структуры разреженного дерева. Впоследствии на его основе в диссертации [7] был разработан алгоритм с улучшенными характеристиками.

В нашей статье на конкретном примере демонстрируется, что разреженные суффиксные деревья могут быть использованы для эффективного решения более широкого круга задач обработки строк. При этом также зачастую удаётся получить существенно лучшие характеристики производительности по сравнению с обычными суффиксными деревьями. Это позволяет эффективно как по времени, так и по памяти решать задачи большой размерности.

4. Использование равномерно разреженных СД на примере решения задачи о максимальном палиндроме. В качестве конкретного примера рассмотрим решение задачи о поиск максимальной симметричной подстроки (палиндрома). Задача выглядит следующим образом. Требуется во входной строке найти подстроку наибольшей длины, которая одинаково читается в обоих направлениях.

Может показаться, что данная задача далека от потребностей практики. На самом деле, это не совсем так. Ведь палиндром – это случай симметрии, которая встречается в природе достаточно широко. Например, похожие реальные задачи возникают в вычислительной биологии при анализе ДНК.

В [2] описан следующий алгоритм решения. Необходимо взять исходную строку и её перевёрнутую копию, дописать к ним по специальному служебному символу, склеить полученные строки и над результатом построить суффиксное дерево.

Далее нужно решить следующую подзадачу. Пусть даны две позиции i и j в строке. Над строкой построено суффиксное дерево. Требуется найти длину наибольшего общего префикса строк, начинающихся с позиций i и j. Для этого возьмём два листа, соответствующие нашим строкам, и найдём их ближайшего общего предка.

Возникает следующая подзадача: за константное время найти ближайшего общего предка (LCA) двух заданных вершин дерева. При этом допускается выполнить предобработку дерева, но не более чем с линейной вычислительной сложностью. Имеется несколько эффективных алгоритмов для поиска ближайшего общего предка – алгоритм двоичного подъёма, Бендера–Фараха–Колтона, Шибера–Вишкина и др.

Теперь осталось только перебрать возможные варианты позиции середины палиндрома, взять эту позицию в исходной и перевёрнутой строке и найти наибольшее общее продолжение. Из всех найденных результатов запоминается лучший.

Стоит отметить, что есть небольшие особенности при работе с палиндромами чётной и нечётной длины, которые необходимо учитывать в программной реализации.

Описанный алгоритм выполняет поиск за линейное время, но имеет значительные требования к памяти как под суффиксное дерево, так и под дополнительные структуры данных, используемые в алгоритме поиска ближайшего общего предка. Можно предложить следующий способ снижения затрат памяти. Выберем такую минимальную степень разреженности k, чтобы полученное дерево вместе с до-

полнительными структурами данных занимало почти весь объём доступной физической памяти.

Пусть t - позиция первого суффикса в строке. Переберём все значения t от 1 до k, для каждого из них построим строку так, чтобы в построенном разреженном СД для каждого суффикса исходной строки имелся дополняющий его суффикс перевернутой строки. Для этого при конкатенации строк между ними придётся вставить не более чем $2k$ служебных символов. На рисунке 4 показан пример для $t = k = 3$.

bananas$$$sananab#
^ ^ ^ ^ ^

Рис. 4. Позиции суффиксов в строке для равномерно разреженного суффиксного дерева при $t = 3$, $k = 3$

Теперь осталось для каждого дерева применить алгоритм поиска ближайшего предка, найти максимальный палиндром и запомнить лучший из вариантов ответа.

Вычислительная сложность данного алгоритма составляет $O(kn)$, требуемый объём памяти – $O(n/k)$. Заметим, однако, что реальное время работы будет существенно зависеть от того, помещается ли построенная структура данных целиком в оперативную память – то есть при малых k время работы будет весьма высоким, а затем, начиная с некоторого k_0, время резко уменьшится, после чего продолжит расти с линейной сложностью. Подтвердим данные рассуждения экспериментально.

5. Вычислительные эксперименты. Экспериментальная проверка результатов выполнялась на вычислительной машине с частотой процессора 2,67 ГГц и объёмом оперативной памяти 1,5 Гб. При этом около 0,5 Гб было занято операционной системой и другим ПО, то есть нашей программе был доступен 1 гигабайт памяти.

В реализации разреженного СД требовалось 32 байта памяти на каждый узел дерева. Заметим, что это не самая эффективная программная реализация СД. Однако, для проверки вышеприведённых теоретических результатов этого вполне достаточно.

Размер входной строки был взят равным 10 миллионов символов. Заметим, однако, что в вышеописанном решении дерево строится над конкатенацией входной строки и её перевёрнутой копии. Поэтому реальный размер входных данных для построения дерева будет в два раза выше – то есть приблизительно 20 мегабайт.

В первом эксперименте мы проверили, как изменяется время работы и потребление памяти в зависимости от степени разреженности

k при построении множества всех разреженных деревьев (то есть в совокупности содержащих все суффиксы исходной строки).

Полученные экспериментальные результаты показаны на рисунке 5, они полностью согласуются с теоретическими. При $k=1$ дерево не помещается целиком в оперативную память. Вследствие использования виртуальной памяти и подкачки страниц время работы оказывается достаточно большим. Однако, уже при $k=2$ дерево помещается в память целиком, и время работы сократилось более чем на порядок. При дальнейшем увеличении k время растёт линейно.

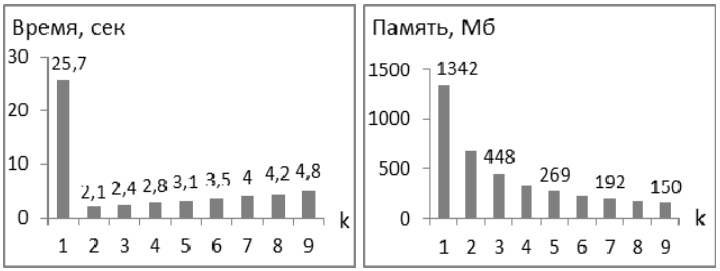

Рис. 5. Зависимость времени и памяти от степени разреженности при построении полного набора разреженных СД

Во втором эксперименте мы добавили в предыдущую программу построение необходимых структур данных для алгоритма Шибера-Вишкина, что почти вдвое увеличило объём памяти. Кроме того, был добавлен код, собственно решающий задачу о поиске максимального палиндрома. Результаты показаны на рисунке 6.

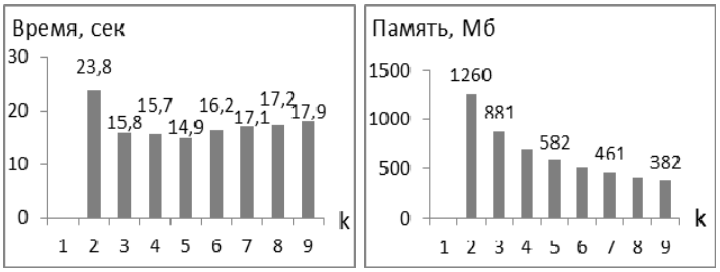

Рис. 6. Зависимость времени и памяти от степени разреженности при поиске максимального палиндрома

Как видно из рисунка, при $k=1$ замеры не получены. Это связано с тем, что требуемый объём памяти превысил допустимые для 32-битных приложений два гигабайта. Интересно отметить, что оптимальное значение k оказалось равно уже не 2, а 5, что связано с особенностями работы алгоритма поиска ближайшего общего предка.

6. Заключение. Описанный подход можно обобщить для эффективного по объему используемой памяти ЭВМ решения ряда других задач обработки строк. Основная идея заключается в использовании известного принципа разбиения исходной задачи на более мелкие подзадачи таким образом, чтобы каждую из них можно было решать по отдельности, после чего на основе их решений восстановить решение исходной задачи.

В нашем случае критерием независимости подзадач является возможность разбиения множества суффиксов исходной строки на непересекающиеся подмножества (или, по крайней мере, с заведомо ограниченной мощностью пересечений) так, чтобы её решение могло бы быть использовано далее.

Основываясь на данных соображениях, удалось также получить эффективное по памяти решение другой задачи обработки строк − поиске наибольшей общей подстроки. Оно подробно описано в статье [8].

Таким образом, можно сделать вывод, что разреженные суффиксные деревья применимы для более широкого круга задач, чем задача, для которой они были изначально предложены. В качестве конкретных областей применения можно привести задачу поиска дубликатов в коллекциях исходного и бинарного программного кода, поиск в тексте по подмножествам регулярных выражений (в частности, по LIKE-шаблонам языка SQL) и др. При этом с помощью разреженных суффиксных деревьев становится возможным проводить обработку данных таких объёмов, которые было бы затруднительно обработать за приемлемое время другими методами.

Литература

1. *Кипяткова И.С., Карпов А.А.* Автоматическая обработка и статистический анализ новостного текстового корпуса для модели языка системы распознавания русской речи // Информационно-управляющие системы. СПб. 2010. № 4. С. 2–8.
2. *Gusfield D.* Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology // Cambridge University Press. 1997. 654 p.
3. *Kärkkäinen J., Ukkonen E.* Sparse Suffix Trees // Proceedings of the Second Annual International Conference on Computing and Combinatorics (COCOON '96). 1996. pp. 219–230.
4. *Bille P., Fischer J., Gortz I. L., Kopelowitz T., Sach B., Vildhoj H. W.* Sparse suffix tree construction in small space // Proceedings of the 40th international conference on Automata, Languages, and Programming (ICALP'13). 2013. pp. 148–159.
5. *Andersson A., Larsson N.J., Swansson K.* Suffix trees on words // Proceedings of 7th Symposium on Combinatorial Pattern Matching (CPM). 1996. pp. 102–115.
6. *Uemura T., Arimura H.* Sparse and truncated suffix trees on variable-length codes // Proceedings of the 22nd annual conference on Combinatorial pattern matching (CPM'11). 2011. pp. 246-260.
7. *Стариковская Т.А.* Эффективные алгоритмы для некоторых задач обработки слов: дис. канд. физ.-мат. наук // МГУ, Москва. 2012. 94 с.
8. *Лисс А.Р., Андрианов, И.А.* Использование разреженных суффиксных деревьев для задач обработки текстов // Известия СПбГЭТУ "ЛЭТИ". 2013. № 3. С. 51–57.

References

1. Kipjatkova I.S., Karpov A.A. [Automatic processing and statistical analysis of the news text corpus for model of language of the system for Russian speech recognition]. *Informacionno-upravljajushhie sistemy – Information and Control Systems*. 2010. no. 4. pp. 2–8. (In Russ.).
2. Gusfield D. Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge University Press, 1997. 654 p.
3. Kärkkäinen J., Ukkonen E. Sparse Suffix Trees. Proceedings of the Second Annual International Conference on Computing and Combinatorics (COCOON '96). 1996. pp. 219–230.
4. Bille P., Fischer J., Gortz I. L., Kopelowitz T., Sach B., Vildhoj H. W. Sparse suffix tree construction in small space. Proceedings of the 40th international conference on Automata, Languages, and Programming (ICALP'13). 2013. pp. 148–159.
5. Andersson A., Larsson N.J., Swansson K. Suffix trees on words // Proceedings of 7th Symposium on Combinatorial Pattern Matching (CPM). 1996. pp. 102–115.
6. Uemura T., Arimura H. Sparse and truncated suffix trees on variable-length codes. Proceedings of the 22nd annual conference on Combinatorial pattern matching (CPM'11). 2011. pp. 246-260.
7. Starikovskaja T.A. *Effektivnye algoritmy dlja nekotoryh zadach obrabotki slov*: *dis. kand. fiz.-mat. nauk* [Efficient algorithms for some word processing problems: Ph.D. dissertation]. MGU, Moskva. 2012. 94 p. (In Russ.).
8. Liss A.R., Andrianov I.A. [The use of sparse suffix trees for text processing tasks]. *Izvestija SPbGJeTU "LJeTI" – Proceedings of ETU* "LETI". 2013. no. 3. pp. 51–57. (In Russ.).

Свиньин Сергей Федорович — д-р техн. наук, ведущий научный сотрудник лаборатории автоматизации научных исследований СПИИРАН. Область научных интересов: цифровая обработка биомедицинских сигналов. Число научных публикаций — 150. sergeus@iias.spb.su; СПИИРАН, 14 линия, Санкт-Петербург, 199178, РФ; р.т. +7(812)323-5139, факс+7(812)328-4450.

Svinyin Sergey Fedorovich — Ph.D., Dr. Sci., leading researcher, laboratory for research automation, SPIIRAS. Research interests: digital processing of biomedical signals. Number of publications — 150. sergeus@iias.spb.su; SPIIRAS, 14-th line V.O., 39, St. Petersburg, 199178, Russia; office phone +7 (812) 323-5139, fax: +7 (812) 328-4450.

Андрианов Игорь Александрович — к-т техн. наук, доцент, кафедры автоматики и вычислительной техники Вологодского государственного университета. Область научных интересов: информационный поиск, методы доступа к данным. Число научных публикаций — 53. igand@mail.ru; ВоГУ, Ленина, 15, Вологда, 160000, РФ; р.т. +7(8172)72-8410, факс+7(8172)72-8410.

Andrianov Igor Alexandrovich — Ph.D., assistant professor, Department of Automatics and Computer Engineering, VoSU. Research interests: information retrieval, data access methods. Number of publications — 53. igand@mail.ru; VoSTU, Lenina street, 15, Vologda, 160000, Russia; office phone +7 (8172) 72-8410, fax: +7 (8172) 72-8410.

Поддержка исследований. Работа выполнена при поддержке Минобрнауки РФ в рамках ФЦП «Научные и научно-педагогические кадры инновационной России» (Госконтракт №02.740.11.0625).

Acknowledgements. This research was supported by Russian Ministry of Education under the Federal Program "Scientific and scientific-pedagogical personnel of innovative Russia" (contract № 02.740.11.0625).

РЕФЕРАТ

Свиньин С.Ф., Андрианов И.А. **Применение равномерно разреженных суффиксных деревьев для задач обработки строк.**

В статье выполнен обзор способов предобработки (индексирования) строк и используемых для этого структур данных с целью ускорения поиска в строках. Показано, что одной из наиболее универсальных структур данных является суффиксное дерево. Приведены основные понятия и определения, касающиеся суффиксных деревьев.

Выявлены два основных недостатка суффиксных деревьев – высокие требования к памяти и плохая пространственная локальность, вследствие чего используемые операционной системой стратегии кеширования оказываются малоприменимыми. В результате суффиксное дерево почти не пригодно для обработки больших объёмов входных данных.

Рассмотрена модификация данной структуры - разреженное суффиксное дерево, которое строится не над всеми суффиксами индексируемых текстов, а лишь над некоторым их подмножеством. Описаны необходимые изменения, которые делаются в алгоритме Укконена построения суффиксного дерева.

Сделано предположение, что разреженные суффиксные деревья могут быть эффективно использованы не только для ускорения поиска длинных подстрок (для чего эта структура была разработана), но и для решения широкого круга других задач обработки строк. В качестве конкретного примера рассмотрена задача о поиске максимальной симметричной подстроки (палиндрома).

Предложено решение данной задачи с использованием разреженных суффиксных деревьев. Показано, что данное решение может обрабатывать данные значительно большей размерности, чем с использованием обычных суффиксных деревьев.

Проведены вычислительные эксперименты, в которых замерялась зависимость времени работы и расхода памяти от степени разреженности дерева. Эксперименты успешно подтвердили теоретические результаты.

Также в статье приводится ссылка на описание полученного нами эффективного по объему используемой памяти ЭВМ для решения задачи о наибольшей общей подстроке, где используются аналогичные идеи.

Сделан вывод, что описанный способ может быть применён и для ряда других задач обработки строк, в том числе поиск по сходству и подмножествам регулярных выражений.

SUMMARY

Svinyin S.F., Andrianov I.A. **Application of evenly sparse suffix tree for string processing tasks.**

The article gives an overview of the ways of preprocessing (indexing) of strings to accelerate the search, and also of the data structures used for this purpose. It is shown that one of the most universal data structures is a suffix tree. The basic concepts and definitions related to suffix trees are given.

Two major shortcomings of suffix trees are revealed, it is high memory requirements and poor spatial locality. Consequently, the caching strategies used by operating systems are not useful. As a result, the suffix tree is almost not suitable for processing large amounts of input data.

An extended version of this structure is a sparse suffix tree. It is built over not all suffixes of indexed text, but only over a subset of them. The necessary changes in the Ukkonen's algorithm for sparse suffix trees are described.

It is suggested that the sparse suffix trees can be used not only to accelerate the search of the long substrings, but also for a wide range of other problems of processing strings. As a specific example, the problem of search of maximal symmetrical substring (palindrome) is taken.

A new solution to this problem with using sparse suffix trees is proposed. It is shown that this solution can process data of much larger volume than with conventional suffix trees.

The computational experiments were performed in which the dependence between the operation time and memory usage of the sparseness level was measured. The experiments successfully confirmed the theoretical results.

The link to our memory efficient solution for the search of the longest common substring is given. It uses similar ideas.

It is concluded that the described method can be applied to a number of problems of processing strings, including similarity search and search by a subset of regular expressions.