

# Приложения суффиксных деревьев

- Суффиксное дерево – один из наиболее мощных современных инструментов вычислений на строках
- Классическое применение: поиск вхождений образца в текст
- Пусть задан текст  $T$  длины  $n$ . Тогда за время  $O(n)$  (например, алгоритмом Укконена) можно построить суффиксное дерево
- Далее поиск в  $T$  любого образца  $P$  длины  $m$  выполняется за время  $O(m + k)$ , где  $k$  – искомое число вхождений образца в текст
- Работает без препроцессинга образца и независимо от длины текста
- Подход эффективен при постоянном тексте и меняющихся образцах
- В полной мере преимущества суффиксных деревьев проявляются при решении ряда других задач на строках, нередко более сложных
- Ниже рассматриваются некоторые из них

# Множественный поиск

- Аналогично решается известная задача *множественного поиска*
- Суть: обнаружение в тексте всех вхождений образцов, принадлежащих заданному множеству («словарю»)
- Последовательно выполняется поиск каждого образца, как описано выше
- Время работы оценивается как  $O(n + m + k)$ , где  $m$  – общая длина всех образцов,  $k$  – число всех их вхождений
- Данная оценка аналогична оценке алгоритма Ахо-Корасик, решающему ту же задачу на основе специального *дерева ключей*

# Наибольшая общая подстрока

- Даны два текста  $T_1$  и  $T_2$  (длиной соответственно  $n_1$  и  $n_2$ ). Требуется найти их наибольшую общую подстроку
- *Замечание.* Может существовать несколько наибольших общих подстрок равной длины, поэтому термин «наибольшая» – не идеален
- Наивный алгоритм: перебор всех подстрок меньшего текста (квадратичная сложность) и поиск их первого вхождения в большем тексте
- Сложность решения – не менее кубической
- Более эффективно – на основе суффиксного дерева

# Наибольшая общая подстрока – инициализация

- Построим суффиксное дерево текста  $T = T_1 \# T_2$  (длиной  $n = n_1 + n_2 + 1$ )
- Маркер  $\#$  не встречается в текстах  $T_1, T_2$
- Любой лист дерева соответствует суффиксу общего текста до символа  $\#$  или только второго текста
- Можно за константное время уточнить данный факт, зная индекс листа, то есть стартовую позицию в  $T$  соответствующего ему суффикса
- Имеем разбиение множества всех листьев на два непересекающихся подмножества – листья *типа 1* и листья *типа 2*

# Наибольшая общая подстрока – схема вычислений

- Отметим каждую внутреннюю вершину как 1, если в ее поддереве есть лист типа 1, и 2, если среди ее потомков есть лист типа 2
- Реализация: обход дерева в глубину, при возврате запоминаются типы листьев. Требуется линейное время по числу вершин, то есть  $O(n)$
- Некоторые вершины получают обе отметки. Путевая метка такой вершины собирает подстроку, общую для  $T_1$  и  $T_2$ . Самая длинная из таковых – наибольшая общая подстрока для двух текстов
- Остается найти вершину с наибольшей строковой глубиной (число символов в пути до нее) среди вершин с обеими отметками (1 и 2)
- Вычисления строковой глубины могут быть выполнены также с помощью стандартных методов обхода дерева, линейных по времени
- Итог: задача о наибольшей общей подстроке двух текстов решается за время  $O(n)$ , где  $n$  – их суммарная длина (Д. Кнут отрицал такую возможность)

# Обобщенное суффиксное дерево

- Дает эффективное решение ряда задач о наборах строк
- Задано множество строк. Сцепим их в текст  $T$  длиной  $n$ , ограничив каждую концевым маркером (маркеры различны и в строках не встречаются)
- Построим для  $T$  суффиксное дерево за время  $O(n)$
- Дерево содержит один лист для каждого суффикса объединенного текста
- Индекс листа отображается в пару чисел – индекс строки, где он начинается ( $i$ ), и стартовую позицию внутри нее ( $j$ ) (для всех листьев за время  $O(n)$ )
- Недостаток: некоторые суффиксы охватывают более одной строки. Такие «синтетические» суффиксы бесполезны
- Маркеры гарантируют, что метка  $\forall$  пути от корня до внутренней вершины не содержит маркеров (она встречается в  $T > 1$  раза, а маркер =1)
- Поэтому можно «обрезать» все синтетические суффиксы до реальных суффиксов строк набора, уменьшая верхний индекс метки в листовых дугах

# Общие подстроки более чем двух строк – приложения

- Важный вопрос, связанный с набором строк: нахождение наибольших общих подстрок, которые встречаются в нем достаточно часто.
- Актуален для биологических последовательностей (ДНК, РНК и т.д.)
- В частности, возникает при поиске функционально значимых участков ДНК
- Мутации, появляющиеся в ДНК при расхождении видов, с большей вероятностью изменяют функционально незначимые участки
- Части ДНК, критичные для функционирования молекулы, сохраняются лучше, и появляющиеся в них мутации как правило неустойчивы
- Нахождение подстрок ДНК, встречающихся у многих видов, позволяет выявить критичные участки для функции и структуры биологической строки
- Задача нахождения общих фрагментов встречается также в эвристических методах *выравнивания* набора строк

# Общие подстроки более чем двух строк – постановка

- Рассматривается задача о точном совпадении
- Пусть имеются  $K$  строк, сумма длин которых равна  $n$
- При  $2 \leq k \leq K$  обозначим  $l(k)$  длину наибольшей подстроки для не менее чем  $k$  строк исходного набора
- $l(k)$  – невозрастающая последовательность
- Требуется вычислить таблицу из  $K - 1$  элементов, где  $k$ -й элемент содержит  $l(k)$  и указывает на одну из общих подстрок такой длины



# Пример

- Рассмотрим набор строк {sandollar, sandlot, handler, grand, pantry}
- Таблица для этого набора (указатели подстрок заменены самими подстроками):

$k$	$l(k)$	Одна из подстрок
2	4	sand
3	3	and
4	3	and
5	2	an

- Существует метод, вычисляющий такую таблицу за  $O(n)$
- Мы опишем лишь алгоритм, который решает сформулированную задачу за время  $O(Kn)$  – обобщение предыдущего алгоритма

# Общие подстроки – инициализация

- Построим обобщенное суффиксное дерево для исходных  $K$  строк
- Каждый лист соответствует суффиксу одной из этих строк
- Лист помечен парой целых чисел  $(i, j)$ :
  - Идентификатор  $i$  (от 1 до  $K$ ) указывает, из которой строки этот суффикс
  - $j$  – стартовая позиция суффикса в этой строке
- Строкам сопоставлены разные терминальные символы, поэтому суффиксы из разных строк всегда завершаются в разных листьях
- Таким образом, каждому листу дерева сопоставлена единственная и уникальная пара чисел  $(i, j)$

# Общие подстроки – эквивалентная постановка

- Для каждой внутренней вершины  $v$  обозначим  $C(v)$  число различных идентификаторов  $i$  у листьев, содержащихся в поддереве  $v$
- Путь в дереве от корня до вершины  $v$  собирает подстроку, общую для  $C(v)$  строк исходного набора
- В терминах дерева и введенных обозначений исходную задачу можно сформулировать эквивалентным образом
- Элемент  $k$  таблицы должен содержать информацию о внутренней вершине, которая имеет максимальную строковую глубину среди всех вершин  $v$ , удовлетворяющих условию  $C(v) \geq k$

# Общие подстроки – 1 стадия решения

- Пусть известны строковые глубины  $d(v)$  всех вершин (требуется линейное время) и числа  $C(v)$
- Значения  $l(k)$  можно получить за линейный по времени обход дерева
- Строится вектор  $V$ , где  $V(k)$  ( $k = 2, \dots, K$ ) содержит  $d(v)$  (и указатель) самой глубокой вершины  $v$ , для которой  $C(v) = k$
- Для  $\forall$  вершины  $v$  сравниваем  $d(v)$  с текущим  $V(k)$  ( $k = C(v)$ ) и, если  $d(v) > V(k)$ , полагаем  $V(k) = d(v)$  (запоминая в  $V$  и указатель вершины)
- По сути  $V(k)$  будет содержать длину наибольшей подстроки, которая встречается в наборе ровно  $k$  раз
- По завершении обхода имеет место  $V(k) \leq l(k)$ , т.к. максимум вычислен по меньшему множеству вершин ( $C(v) = k$ ), чем требуется ( $C(v) \geq k$ )

# Пример

- Рассмотрим предыдущий пример {sandollar, sandlot, handler, grand, pantry}
- Таблица для него:

$k$	$l(k)$	Одна из подстрок
2	4	sand
3	3	and
4	3	and
5	2	an

- На промежуточной стадии вектор  $V$  будет иметь следующие компоненты:  
 $V = \{4, 0, 3, 2\}$
- 0 при  $k = 3$ , т.к. нет общей подстроки, встречающейся ровно 3 раза
- Есть подстрока *and*, появляющаяся 4 раза, но она не учитывается этой стадией алгоритма

# Общие подстроки – 2 стадия решения

- Для получения величин  $I(k)$  достаточно преобразовать вектор  $V$  (за время  $O(K)$ )
- Просматривая его с конца к началу, будем записывать в текущую позицию  $k$  максимальное из пройденных ранее значений
- Если  $V(k)$  пусто или  $V(k) < V(k + 1)$ , положим  $V(k)$  равным  $V(k + 1)$
- Результирующий вектор содержит необходимые значения  $I(k)$
- В рассмотренном примере на промежуточной стадии вектор  $V$  имел компоненты:  $V = \{4, 0, 3, 2\}$
- Итоговый вектор примет вид  $V = \{4, 3, 3, 2\}$

# Вычисление величин $C(v)$

- $C(v)$  – число различных идентификаторов  $i$  у листьев в поддереве  $v$
- Число всех листьев в поддереве  $v$  может быть  $> C(v)$ , т.к. листья могут иметь одинаковые  $i$ , что затрудняет быстрое вычисление  $C(v)$
- Рассмотрим алгоритм, который непосредственно вычисляет эти идентификаторы
- У вершине  $v$  сопоставим битовый вектор длины  $K$ , где  $i$ -й бит установлен в 1, если в поддереве  $v$  присутствует идентификатор  $i$
- Вектор для  $v$  формируется при обходе дерева логическим сложением векторов потомков  $v$
- $C(v)$  = числу единиц в векторе; вычисляется во внутреннем цикле за  $O(K)$
- Число вершин в дереве  $O(n)$ , на построение всей таблицы потребуется  $O(Kn)$
- Существует алгоритм, работающий за  $O(n)$

# Задача о подстроке для базы образцов – постановка

- Есть набор строк (БД). Для еще одной строки  $S$  надо найти в БД все строки, содержащие  $S$  как подстроку – «инверсия» задачи множественного поиска
- Физический смысл – в контексте БД генома ДНК
- Есть коллекция предварительно расшифрованных (*секвенированных*) строк
- Новая расшифровываемая строка ДНК может входить в секвенированную ранее строку
- Актуален эффективный метод обнаружения такой ситуации



# Задача о подстроке для базы образцов – решение

- Основано на обобщенном суффиксном дереве для исходного набора строк
- Вхождение  $\forall$  строки  $S$  длины  $m$  (или ее отсутствие) может быть идентифицировано в базе данных за время  $O(m)$
- $S$  целиком присутствует в базе данных, если ее сравнение завершается в листе, атрибут  $j$  которого равен 0
- Если же  $S$  – подстрока в базе данных, то дерево позволяет найти все строки в базе данных, содержащие  $S$  в качестве подстроки
- Сложность этой операции составляет  $O(m + k)$ , где  $k$  – число всех вхождений
- Если  $S$  не совпадает с путем, то она не содержится в базе данных, но совпавшая часть = самый длинный префикс  $S$ , содержащийся в базе данных
- Такие результаты по эффективности не достигаются ни одним из других представленных ранее алгоритмов

# Недостатки суффиксных деревьев

- Несмотря на мощь и эффективность, имеются недостатки
- Высокое потребление памяти, свойственное любым деревьям: хранятся ссылки на потомков и родителей
- Добавляются суффиксные ссылки, а в приложениях – еще информация
- Зависимость от длины алфавита. В узле располагаются ссылки на дочерние вершины в количестве от 1 до размера алфавита
- Можно в узле размещать массив ссылок размером в алфавит. Работает быстро – определение наличия у узла потомка по исходящему символу  $\sim O(1)$
- Однако большинство элементов массива = пустые ссылки – неэффективно
- Критично для больших алфавитов (тексты, Unicode)
- Компактные способы хранения ссылок (динамический сортированный массив, хэш-таблицы) экономят память, но увеличивают время работы

# Альтернатива – суффиксный массив

- Компромисс между памятью и производительностью при решении аналогичных задач дает суффиксный массив
- По сравнению с суффиксным деревом увеличивает время обработки строк
- Однако в отличие от усложненного размещения дочерних ссылок, упрощается как хранение данных, так и их использование
- Впервые описан в 1989г. Дж. Майерсом и У. Манбером, как более экономная по памяти альтернатива суффиксному дереву для поиска подстрок
- *Определение.* Суффиксный массив – это массив лексикографически отсортированных суффиксов строки (или строк)
- Для хранения суффиксов необязательно размещать их в массиве. Достаточно позиции начала суффикса в исходной строке

# Суффиксный массив – пример

• Известная строка: a b r a c a d a b r a

0 1 2 3 4 5 6 7 8 9 10

Индекс в массиве	Суффикс	Позиция суффикса
0	<i>a</i>	10
1	<i>abra</i>	7
2	<i>abracadabra</i>	0
3	<i>acadabra</i>	3
4	<i>adabra</i>	5
5	<i>bra</i>	8
6	<i>bracadabra</i>	1
7	<i>cadabra</i>	4
8	<i>dabra</i>	6
9	<i>ra</i>	9
10	<i>racadabra</i>	2

{ 10, 7, 0, 3, 5, 8, 1, 4, 6, 9, 2 }

# Наивный поиск в суффиксном массиве

- Если требуется найти подстроку *ab*, достаточно найти все суффиксы, которые начинаются с *ab*.
- За счёт сортировки они расположены в массиве подряд
- Бинарным поиском найдем суффиксы *abra* и *abracadabra*, которым соответствуют 1-й и 2-й элемент суффиксного массива (7 и 0)  $\Rightarrow$
- Подстрока *ab* встречается в нулевой и седьмой позиции исходного текста
- Однократное сравнение образца с суффиксом в худшем случае занимает время  $O(m)$   $\Rightarrow$  общее время такого поиска составит  $O(m \lg n)$
- Другой способ – посимвольный. По символу образца бинарным поиском находим диапазон с суффиксами, начинающимися с него
- Все элементы диапазона отсортированы, начальные символы одинаковы  $\Rightarrow$  оставшиеся после символа суффиксы также отсортированы
- Процедура повторяется до получения пустого диапазона либо успешного нахождения всего образца; оценка аналогична –  $O(m \lg n)$

# Ранг суффикса

- Ранг суффикса = позиция, которую суффикс займет в суффиксном массиве (в результате сортировки)
- Отображение позиций суффиксов в тексте и суффиксном массиве – взаимно однозначное
- *Массив рангов* – целочисленный массив, где ранги расположены в порядке возрастания позиций суффиксов в тексте
- Суффиксный массив и массив рангов легко преобразуются друг в друга за линейное время

# Массив рангов - пример

