Суффиксные массивы

Дискретный анализ 2012/13

Андрей Калинин, Татьяна Романова

10 декабря 2012 г.

Суффиксные массивы
- Определение
- Построение с использованием деревьев

Использование суффиксных массивов
- Поиск образцов
- Ускорение с использованием Lcp

Построение без использования деревьев

Литература

- Дэн Гасфилд, «Строки деревья и последовательности в алгоритмах: Информатика и вычислительная биология», 2003. Глава 7, «Первые приложения суффиксных деревьев», стр. 158–213.
- Построение суффиксных массивов за $O(n \log n)$ http://e-maxx.ru/algo/suffix_array

Раздел

Суффиксные массивы
- Определение
- Построение с использованием деревьев

Использование суффиксных массивов
- Поиск образцов
- Ускорение с использованием Lcp

Построение без использования деревьев

Мотивация

- Полное суффиксное дерево занимает слишком много памяти.
- Суффиксное дерево зависит от размера алфавита.
- При построении: либо память $\Theta(m|\Sigma|)$, либо время $O(\min\{m \log m, m \log |\Sigma|\})$.
- Поиск образца выполняется за время $O(n)$ при размере памяти $\Theta(m|\Sigma|)$. Иначе, за $O(n \min\{\log m, \log |\Sigma|\})$.
- Можно ли придумать иную структуру данных, которая бы позволяла бы искать некоторые задачи, решаемые суффиксным деревом, с аналогичными временными характеристиками?

Суффиксный массив

Определение

Суффиксным массивом для m-символьной строки Pos называется массив целых чисел от 1 до m, определяющий лексикографический порядок m суффиксов строки T.

- Т.е., суффикс, начинающийся в $Pos(1)$ строки T лексикографически самый маленький, а суффикс $Pos(i)$ меньше $Pos(i+1)$.
- Терминальный символ счиатется лексикографически меньше любого другого символа исходного алфавита.
- При добавлении ещё $2m$ значений суффиксный массив можно использовать для поиска всех вхождений в T образца P за $O(n + \log m)$ операций вне зависимости от размера алфавита.

Пример суффиксного массива

Для строки $mississipi$ масив $Pos = (11, 8, 5, 2, 1, 10, 9, 7, 4, 6, 3)$:

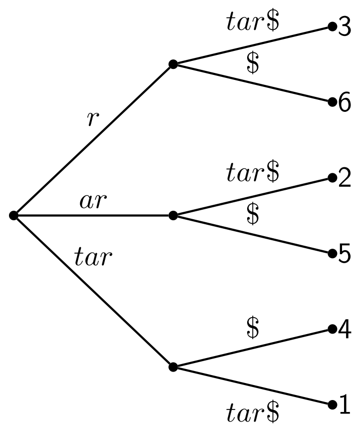
11	i
8	ippi
5	issippi
2	ississippi
1	mississippi
10	pi
9	ppi
7	sippi
4	sisippi
6	ssippi
3	ssissippi

Построение суффиксного массива

- Для текста T построить суффиксное дерево $\mathbb{T}$.
- Обойти дерево $\mathbb{T}$ в глубину таким образом, что первыми проходятся дуги, чьи метки меньше остальных в лексикографическом смысле.
- Если дуги хранятся в порядке возрастаний первых символов меток, то такой обход будет натуральным
- Суффиксный массив — просто список посещений листьев при таком обходе.
- Тем самым, суффиксный массив строится за время $O(m)$.

Пример построения суффиксного массива

Строка $tartar$:



Обход в порядке 5, 2, 6, 3, 4, 1.

Раздел

Суффиксные массивы
- Определение
- Построение с использованием деревьев

Использование суффиксных массивов
- Поиск образцов
- Ускорение с использованием Lcp

Построение без использования деревьев

Поиск образца

- Если образец T входит в T, то все суффиксы, начинающиеся с P в массиве Pos будут располагаться рядом.
- Нужно выполнить двоичный поиск P в массиве Pos: найти наименьший индекс i, такой что $Pos(i-1)$ не начинается с P и наибольший i', что $Pos(i'+1)$ не начинается с P. Тогда есть вхождение образца в позициях от $Pos(i)$ до $Pos(i')$.
- Пессимистичная оценка времени работы $O(n \log m)$, достигается при наличии большого количества длинных префиксов P.
- Можно улучшить до $O(n + \log m)$.

Простое ускорение

- L и R — текущие границы интервала поиска. В начале $L = 1$ и $R = m$.
- Запоминается длина префиксов $Pos(L)$ и $Pos(R)$, совпадающих с префиксом P: l, r.
- $mlr = \min(l, r)$.
- При очередном сравнеии в позиции $M = \lfloor (R + L)/2 \rfloor$ можно начинать обрабатывать символы не с первой позиции, а с $mlr(l, r) + 1$.
- На практике достигается $O(n + \log m)$, однако в худшем случае остается $O(n \log m)$.

Значения $Lcp(i, j)$

- Проверка символа в P избыточная, если этот символ был проверен ранее.
- Цель: уменьшить количество избыточных проверок до не более одной на каждую итерацию бинарного поиска.
- mlr не подходит, т.к при при $l \neq r$ все символы до $\max(l, r) > 1$ уже проверялись.
- $Lcp(i, j)$ — длина наибольшего общего префикса суффиксов $Pos(i)$ и $Pos(j)$.
- Для $T = mississippi$ $Pos(3)$ — $issippi$, а $Pos(4)$ — $ississippi$, т.е. $Lcp(3, 4) = 4$.

Использование Lcp

- $M \leftarrow \lfloor (R+L)/2 \rfloor$.
- $l = r$, тогда сравнение P с $Pos(M)$ начинается с позиции $mlr(l,r)+1 = l+1 = r+1$.
- $l \neq r$, предположим что $l > r$:
  1. $Lcp(L,M) > l$: P совпадает с $Pos(M)$ на l символов, и $l+1$-й символ у $Pos(L)$ и $Pos(M)$ совпадает, т.е. $L \leftarrow M$ без проверок символов.
  2. $Lcp(L,M) < l$: P совпадает с $Pos(M)$ на $Lcp(L,M)$ символов, а $Lcp(L,M)+1$-й символ у $Pos(L)$ и P совпадает, т.е. $R \leftarrow M$, $r \leftarrow Lcp(L,M)$.
  3. $Lcp(L,M) = l$: P совпадает с $Pos(M)$ на l символов, нужно явно сравнивать символы с $l+1$-го и решить, что делать по правилам бинарного поиска (запомнив l и r).

$Lcp(L, M) > l$

P	`abcdemn`	
L	`abcdefg`	$l = 5$
M	`abcdefg`	$Lcp(L, M) = 6$
R	`abcdxyz`	$r = 4$

Эффективность использования Lcp

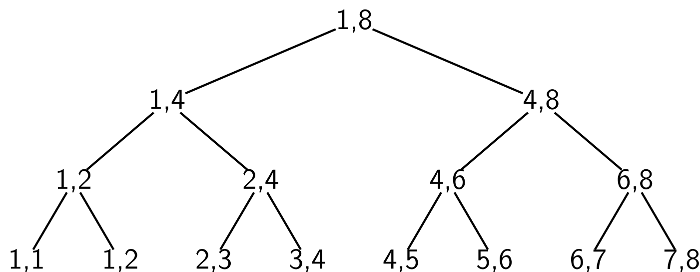
Теорема

При использовании значений Lcp поисковый алгоритм делает не более $O(n + \log m)$ сравнений и работает за такое же время.

Доказательство.

- l и r не уменьшаются.
- В сучаях $l = r$ или $Lcp(L, M) = l > r$ алгоритм начинает проверку с $\max(l, r)$ символа и либо увеличивает l или r, или заканчивает работу.
- Тем самым, не более одного избыточного сравнения на итерацию, т.е. количество сравнений $n + \log m$.

Вычисление Lcp



Здесь каждая внутренняя вершина (i, j) имеет двух детей, $(i, \lfloor (i+j)/2 \rfloor)$ и $(\lfloor (i+j)/2 \rfloor, j)$. Т.е., при работе двоичного поиска понадобятся только $2m - 1$ значений $Lcp(i, j)$.

Вычисление Lcp

- При обходе дерева $\mathbb{T}$ во время построения суффиксного массива Pos ближайшая к корню вершина, посещённая при переходе от $Pos(i)$ к $Pos(i+1)$ определяет значение $Lcp(i, i+1)$ равным её строковой глубине.
- $Lcp(i, j) = \min_{i \leq k < j} \{Lcp(k, k+1)\}$
- Отсюда, можно вычислить при построении суффиксного массива значения $Lcp(i, i+1)$, а потом достроить за линейное время оставшиеся значения, которые могут понадобиться при бинарном поиске.
- Тем самым: с использованием суффиксных массивов предварительная обработка занимает время $O(m)$, а время работы составляет $O(n + \log m)$.

Раздел

Суффиксные массивы
Определение
Построение с использованием деревьев

Использование суффиксных массивов
Поиск образцов
Ускорение с использованием Lcp

Построение без использования деревьев

Общее описание

- Вместо суффиксов будем сортировать циклические сдвиги строки.
- Для получения этим алгоритмом суффиксного массива достаточно добавить в конец строки символ с наименьшим кодом (например, нулевой символ в Си).
- Алгоритм будет состоять из $k = 0 \dots \lceil \log n \rceil$ фаз, на k-й фазе сортируются циклические подстроки длины 2^k.
- На последней фазе получим остортированный массив циклических сдвигов.

Вспомогательные массивы

- Массив p на k-й итерации будет содержать индексы отстортированных циклических строк размера 2^k.
- В массиве c на i-й позиции будет храниться номер класса эквивалентности, которому принадлежит строка, начинающаяся с i размером 2^k.
- Меньшая строка получает меньший класс эквивалентности. У одинаковых строк классы совпадают. Классы нумеруются с нуля.

Пример:

Строка aaba

k	p	c	сортируемые подстроки
0	(0, 1, 3, 2)	(0, 0, 1, 0)	(a, a, b, a)
1	(0, 3, 1, 2)	(0, 1, 2, 0)	(aa, ab, ba, aa)
2	(3, 0, 1, 2)	(1, 2, 3, 0)	(aaba, abaa, baaa, aaab)

Нулевой шаг

- При $k = 0$ сортируются отдельные символы.
- Это можно сделать сортировкой подсчетом, на нулевом месте массива p окажется индекс наименьшего символа.
- С помощью прохода по p и сравнения заполним массив c (если $s[p[i]]! = s[p[i-1]]$ увеличиваем номер класса эквивалентности).
- Время работы на нулевом шаге — $O(n)$.

Если мы научимся за $O(n)$ переходить от k-го шага к $k+1$-му, то так как фаз всего $\log n$, общее время работы алгоритма будет $O(n \log n)$.

От k-го к $k+1$-му

- Циклическая строка длины 2^k состоит из двух циклических подстрок длины 2^{k-1}, которые мы сравнивали на предыдущем шаге.
- Для строки, начинающейся в позиции i, возьмем данные из массива c, построенного на предыдущем шаге: $(c[i], c[i+2^{k-1}])$.
- Сортировка по этим парам даст нам новый массив p.
- Новый c построим, пройдя по p и сравнив две пары значений из предыдущего массива c.
- Получившееся время работы: $O(n \log^2 n)$ — $\log n$ шагов с сортировкой на каждом шаге.

Ускорение

- Для сортировки пар используем поразрядную сортировку: сначала по вторым элементам пары, затем по первым.
- Сортировка по вторым элементам уже содержится в предыдущем массиве p. Поэтому просто вычисляем индекс соответствующего первого элемента $(p[i] - 2^{k-1})$ и сохраняем его на i-й позиции. (Придется завести дополнительный массив).
- Для упорядочения по первым элементам используем сортировку подсчетом, как на нулевом шаге.
- Массив c получаем аналогично предыдущим вариантам алгоритма.
- Время работы: $O(n \log n)$, код алгоритма: http://e-maxx.ru/algo/suffix_array.

Пример

Строка: $s = $ aabacaac

- $n = 8 \Rightarrow k = 0 \dots 3$
- $k = 0$
- Сортировка подсчетом:
  - $cnt = (5, 1, 2)$ — считаем, сколько раз встречается каждая буква
  - $cnt = (5, 6, 8)$ — $cnt[i] += cnt[i-1]$
  - $p = (0, 1, 3, 5, 6, 2, 4, 7)$ — заполняем массив p индексами строки s, проходя по ней с конца.
- $c = (0, 0, 1, 0, 2, 0, 0, 2)$ — заполянем $c[p[i]]$, сравнивая $s[p[i]]$ и $s[p[i-1]]$.

Пример

- $k = 1$ (рассматриваем строки длиной 2)
- $pn = (7, 0, 2, 4, 5, 1, 3, 6)$ — заполняем вспомогательный массив: $pn[i] = p[i] - 1$.
- Получили массив, отсортированный по вторым буквам строк длиной 2.
- Сортировка подсчетом:
  - $cnt = (5, 1, 2)$, $cnt = (5, 6, 8)$ — подсчитываем, используя предыдущий c.
  - $p = (....6...) \to (...36...) \to (..136...) \to (.5136...) \to (.5136..4) \to (.51362.4) \to (051362.4) \to (05136274)$.
- $c = (0.......) \to (0....0..) \to (01...0..) \to (01.2.0..) \to (01.2.02.) \to (0132.02.) \to (0132.024) \to (01324024)$

Пример

- $k = 2$ (рассматриваем строки длиной 4)
- $pn = (6, 3, 7, 1, 4, 0, 5, 2)$ — заполняем вспомогательный массив: $pn[i] = p[i] - 2$.
- Получили массив, отсортированный по вторым двум буквам строк длиной 4.
- Сортировка подсчетом:
  - $cnt = (2, 1, 2, 1, 2)$, $cnt = (2, 3, 5, 6, 7)$ — подсчитываем, используя предыдущий c.
  - $p = (.....2..) \rightarrow (.5...2..) \rightarrow (05...2..) \longrightarrow (05163274)$. Заполняется проходом по pn из конца в начало: $p[- - cnt[c[pn[i]]]] = pn[i]$.
- $c = (02436135)$

Пример

- $k = 3$ (рассматриваем строки длиной 8)
- $pn = (4, 1, 5, 2, 7, 6, 3, 0)$ — заполняем вспомогательный массив: $pn[i] = p[i] - 4$.
- Сортировка подсчетом:
  - $cnt = (1, 1, 1, 2, 1, 1, 1)$, $cnt = (1, 2, 3, 5, 6, 7, 8)$ — подсчитываем, используя предыдущий c.
  - $p = (0.......) \to (0...3...) \to (0..63...) \longrightarrow (05163274)$.

Результирующий суффиксный массив: $p = (0, 5, 1, 6, 3, 2, 7, 4)$.