Алгоритмы и структуры данных

Лекция 26

Строки. Суффиксные массивы. Задача LCP. Алгоритм Касаи. Разреженные таблицы.

Сергей Леонидович Бабичев

Суффиксные массивы

Definition

Пусть имеется строка $s = s_0 s_1 \dots s_n$. Обозначим суффикс строки s, начинающийся с позиции pos как S_{pos}. Тогда суффиксным массивом от строки s будет такая перестановка p чисел от 0 до $n-1$ $p_0, p_1, \dots, p_{n-1}$, что суффиксы S_{p_0}, $S_{p_1}, \dots, S_{p_{n-1}}$ лексикографически упорядочены, то есть $S_{p_0} < S_{p_1} < \dots < S_{p_{n-1}}$

Обратим внимание, что равенство невозможно.

Пример суффиксного массива строки $AABABC$

0	AABABC
1	ABABC
2	BABC
3	ABC
4	BC
5	C

0	AABABC
1	ABABC
3	ABC
2	BABC
4	BC
5	C

Пример суффиксного массива строки $CADABRA$

0	CADABRA
1	ADABRA
2	DABRA
3	ABRA
4	BRA
5	RA
6	A

6	A
3	ABRA
1	ADABRA
4	BRA
0	CADABRA
2	DABRA
5	RA

LCP и суффиксные массивы.

Definition

Longest Common Prefix — $LCP(s_1, s_2)$ — длина наибольшего префикса у строк s_1 и s_2.

- Нас интересует массив LCP, построенный на переставленных в лексикографическом порядке суффиксов.
- После построения такого массива можно свести задачу нахождения длины наибольшего префикса на произвольных подстроках строки s к нахождению минимума на подотрезке.

LCP суффиксного массива

- LCP на суффиксном массиве строки s — набор чисел $L_0, L_1, \ldots, L_{n-1}$ таких, что $L_i = LCP(S_{p_i}, S_{p_{i+1}})$.

Пример LCP суффиксного массива строки $ABRACADABRA$.

10	A	1
7	ABRA	4
0	ABRACADABRA	1
3	ACADABRA	1
5	ADABRA	0
8	BRA	3
1	BRACADABRA	0
4	CADABRA	0
6	DABRA	0
9	RA	2
2	RACADABRA	0

Запрос на длину наибольшего суффикса на подстроках

Задача 1. Имеется строка длины s длины N. Требуется многократно отвечать на запросы: найти LCP для подстрок строки s $[s_{l_1}, s_{r_1}]$ и $[s_{l_2}, s_{r_2}]$.
Решение: Строим суффиксный массив по строке s. По суффиксному массиву — массив LCP. В массиве LCP отвечаем на запрос: найти минимальное значение на отрезке.

Задача 2. Для данной строки нужно отвечать на запросы, одинаковые ли подстроки $[s_{l_1}, s_{r_1}]$ и $[s_{l_2}, s_{r_2}]$

Что нам осталось? Уметь строить суффиксный массив за $O(N \log N)$, массив LCP за $O(N)$, структуру данных $sparsetable$ за $O(N \log N)$ и мы сможем отвечать на запросы задач за $O(1)$.

Строим суффиксный массив

- Для того, чтобы убрать из алгоритмов много условных переходов и сравнений, добавим в конец строки s_{orig} символ, меньший любого из символов строки, получив строку s.
- Для строк, состоящих из букв латинского алфавита, это, например, '@'.

Lemma

Суффиксный массив, построенный по всем циклическим суффиксам s совпадает с суффиксным массивом, построенным по s_{orig}.

Пример строки s с циклическими суффиксами

6	A
3	ABRA
1	ADABRA
4	BRA
0	CADABRA
2	DABRA
5	RA

6	A@CADABR
3	ABRA@CAD
1	ADABRA@C
4	BRA@CADA
0	CADABRA@
2	DABRA@CA
5	RA@CADAB

Сортировка циклических суффиксов

- Простая сортировка требует сравнения строк за $O(N)$ и таких сравнений будет $O(N \log N)$, что даёт $O(N^2 \log N)$
- Количество всех возможных подстрок в строке ограничено $O(N^2)$ и подстрока длины 2^k может быть составлена из двух подстрок длины 2^{k-1}.
- Если есть результаты сравнения *малых* подстрок, сравнение *больших* потребует $O(1)$.
- Все подстроки одной длины l (а их ровно N можно разбить на классы эквивалентности — в одном классе будут находиться совпадающие подстроки этой длины, начинающиеся с позиции p.

0	1	2	3	4	5	6	7	8
A	B	A	A	A	B	A	A	@

Классы эквивалентности для циклических подстрок длины 1.

A	B	A	A	A	B	A	A	@
1	2	1	1	1	2	1	1	0

Классы эквивалентности для циклических подстрок длины 2.
Сами подстроки: $@A = 0$, $A@ = 1$, $AA = 2$, $AB = 3$, $BA = 4$

A	B	A	A	A	B	A	A	@
3	4	2	2	3	4	2	1	0

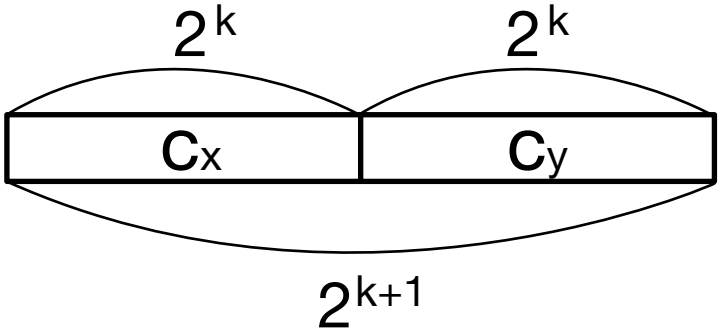
Как теперь отсортировать строки длины 4?
$ABAA = AB + AA = (3) + (2)$
$BAAA = BA + AA = (4) + (2)$
$AAAB = AA + AB = (2) + (3)$.
Сохраняем инвариант, что если s_1 — строка из меньшего класса эквивалентности, а s_2 — из большего, то лексикографически $s_1 < s_2$.

Построение суффиксного массива

- Алгоритм можно разбить на две части — первичную и вторичную.
- Первичная часть. Сортируем подстроки длины 1, получая массив перестановок.
- По отсортированному массиву строится массив классов эквивалентности.
- Теперь каждая строка длины 1 имеет свой класс эквивалентности, чем меньше строка, тем меньше класс.

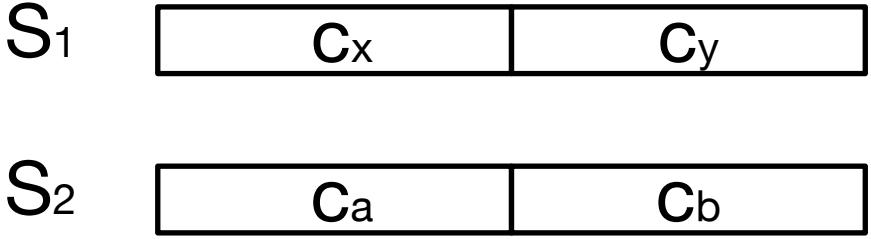
Построение суффиксного массива: вторичная часть

- Мы забываем про значения символов и работаем только с классами эквивалентности.
- Предположим, мы имеем все классы эквивалентности для строк длины 2^k.
- На следующей итерации мы хотим получить классы эквивалентности для строк длины 2^{k+1}.



Построение суффиксного массива: вторичная часть

- Как сравнить две подстроки, для половинок которых известны классы эквивалентности?



- Если $C_x < C_a$, то $S_1 < S_2$.
- Если $C_x > C_a$, то $S_1 > S_2$.
- Если $C_x = C_a$, то S_1 и S_2 имеют такое же отношение, как и C_y и C_b.
- Для строк длины 2^k вторая половинка начинается с позиции $C_x + 2^k$.

Построение суффиксного массива: вторичная часть

- После сортировки каждая позиция строки имеет свой номер в отсортированном уже по длине 2^{k+1} массиве
- Проходим по отсортированному и строим новый массив с классами эквивалентности.
- Алгоритм прост: так как массив отсортирован, присваиваем первому элементу класс 0 и делаем этот класс текущим.
- Далее если следующий элемент совпадает с текущим, то присваиваем текущий класс эквивалентности элементу и идём дальше.
- Если нет — увеличиваем текущий класс эквивалентности, и присваиваем его нужному элементу.
- Во все моменты времени сохраняется инвариант — большему элементу присваивается больший класс эквивалентности.

Построение суффиксного массива: асимптотика

- Вторичная часть заканчивается, когда 2^k превзойдёт $N = |s|$.
- $k = \Theta(\log N)$.
- Простая сортировка массивов происходит за $O(N \log N)$.
- При простой сортировке сложность построения суффиксного массива равна $O(N \log^2 N)$.
- Однако, сортируются числа в диапазоне $[0 \dots N)]$.
- Для таких чисел быстрее работает radix-sort.
- На первичном этапе мы сортируем только символы и требуется один проход.
- На вторичном этапе сортируются пары и нужно сначала отсортировать правые компоненты пар, затем устойчиво левые.
- Сложность алгоритма тогда составит $O(N \log N)$.

Построение LCP.

Построение LCP

Задача 3. Имеется суффиксный массив p в отсортированном порядке. Требуется построить массив LCP.

Решение:

Нам нужно будет по значению p_i быстро определять, для какого i оно получено. Это — задача поиска. По массиву p введём массив $inv(x)$ такой, что $inv(p(x)) = x$. Для строки $ABAABAA$ массив p равен $\{6, 5, 2, 3, 0, 4, 1\}$.

Прямой массив p.

0	1	2	3	4	5	6
6	5	2	3	0	4	1

Обратный массив inv.

0	1	2	3	4	5	6
4	6	2	3	5	1	0

Построение LCP

6	A
5	AA
2	AABAA
3	ABAA
0	ABAABAA
4	BAA
1	BAABAA

- Рассмотрим пока произвольную не последнюю строку, например, $ABAA$. $i = 3$.
- Какая строка идёт за ней? $ABAABAA$. $j = 0$.
- Что будет, если отбросить первый символ в обеих строках?
- Получатся две новых строки, BAA и $BAABAA$.
- Где они находятся? BAA ровно на один символ короче $ABAA$, в оригинальном массиве BAA на одну позицию ниже $ABAA$.

6	A
5	AA
2	AABAA
3	ABAA
0	ABAABAA
4	BAA
1	BAABAA

- Если позиция в суффиксном массиве $ABAA$ равна i, то позиция в оригинальном массиве $ABAA$ равна inv_i.
- Пусть LCP для этой строки равен l.
- Позиция в оригинальном массиве строки без одного первого символа равна $inv_i - 1$.
- Позиция в отсортированном массиве строки без первого символа равна $p_{inv_i - 1}$.
- Длина LCP для этой позиции как минимум равна $l - 1$.
- Это значит, что нам не нужно сравнивать первые l символов для нахождения LCP от более короткой строки.
- Если запустить алгоритм для самой длинной строки и уменьшать длину строки, получим все значения LCP.

Примерный код алгоритма Касаи

```
int l = 0;
for (int x = 0; x < N; x++) {
    int i = inv[x];
    if (i == N-1) {
        l = 0;
    } else {
        int j = p[i+1]; // Next line in sorted sufmas
        if (l > 0) l--; // assert: l >= 0
        while (x+l < N && j+l < N && s[x+l] == s[j+l]) l++;
        lcp[i] = l;
    }
}
```

Почему это работает

- l — длина наибольшего LCP до сих пор.
- x — позиция в неотсортированном суффиксном массиве. Длины таких строк уменьшаются с каждой итерацией.
- i — позиция в отсортированном суффиксном массива. И нас интересует максимальное совпадение с элементов на позиции $i + 1$.
- Для строки в неотсортированном массиве под номером $N - 1$ нет следующей, поэтому мы сбрасываем l.
- Иначе в следующей строке не менее $l - 1$ общих символов, но не менее 0.

На каждой итерации l может увеличиваться до величины не более N. На каждой итерации l может уменьшаться не более, чем на 1, за исключением того, что ровно один раз l может сброситься сразу до 0. Сложность алгоритма:

$$T = O(N).$$

Разреженные таблицы (sparse table)

Задача RMQ

Задача 4. Имеется статический массив $a_1, a_2, \ldots, a_N$. К нему поступают запросы: найти минимум на отрезке $[l, r]$ — Range Minimum Query.
Решение: Несколько решений уже есть: декартовы деревья, деревья отрезков.
Все требуют $O(\log N)$ времени на запрос.
Ценой дополнительной памяти в $O(N \log N)$ можно отвечать на запрос за $O(1)$.

Задача RMQ

- Пусть у нас будет массив $v[L][N]$ такой, что $v[r][i]$ — минимальное число на отрезке длины 2^r, начинающемся в $i-$й позиции.
- L — максимальное из чисел, для которых $2^L \leqslant N$, т. е. $L = \lfloor \log_2 N \rfloor$.
- Нулевой слой содержит элементы самого массива.
- Первый слой содержит минимумы от пар, начинающихся в i.
- Второй слой — минимумы от четвёрок, начинающихся в i.

0	2	7	1	8	2	8	5	4
1	2	1	1	2	2	5	4	
2	1	1	1	2	2			
3	1							

- Как поможет такая таблица?
- Ищем минимум на отрезке $[l, r]$.
- Найдём такое максимальное k, что $2^k \leqslant r - l + 1$.
- Элемент $v[k][l]$ содержит минимум из $a[l]..a[l + 2^k - 1]$.
- Элемент $v[k][r - 2^k + 1]$ содержит минимум из $a[r - 2^k + 1]..a[r]$.
- Эти отрезки пересекаются.
- Операция $\min$ коммутативная, ассоциативная и идемпотентная.
- $\min$ от предложенных элементов есть $\min$ на всём подмассиве $[l, r]$.
- Асимптотика: $N \log N$ на построение, $O(1)$ на каждый запрос, если предподсчитать все $\log_2[1..N]$.