Суффиксное дерево

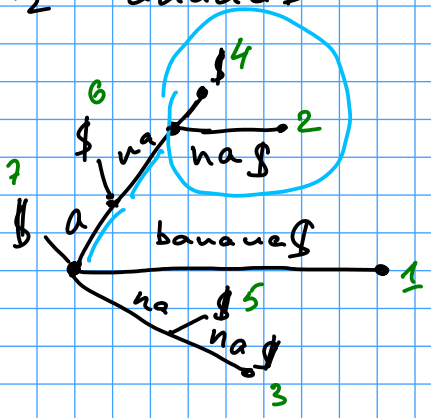
Питер Вайнер, 1973

mississippi

T = banana$

T_1 = banana$ T_3 = nana$ T_5 = na$ T_7 = $

T_2 = anana$ T_4 = ana$ T_6 = a$



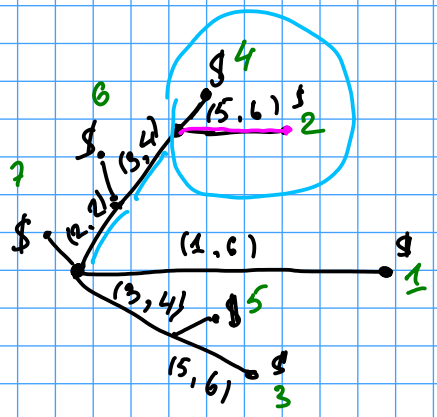
P = ana

Проверка вхождения P в T?

$O(|P|)$

Поиск – ?

≡ Сжатое суффиксное дерево



Память O(# внутр. вершин)

внутр. вершин ≤ # листьев

$O(|T|)$

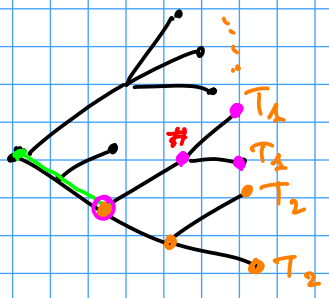
Поиск всех вхождений P в T

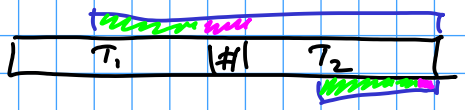
$O(|P| + $ # вхождений$)$

Поиск наибольшей общей подстроки.

Вход: T_1 ~ T_2

Построим ST для $T_1 \# T_2$ $





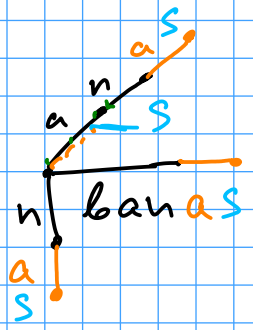
$O(|T_1| + |T_2|)$

Кнут '70. Гипотеза, что $\Omega(n \log n)$

Алгоритм Укконена

Неявное СД - СД для текста без #

Фаза i: Построение НСД для $T[1..i]$ из дерево для $T[1..(i-1)]$



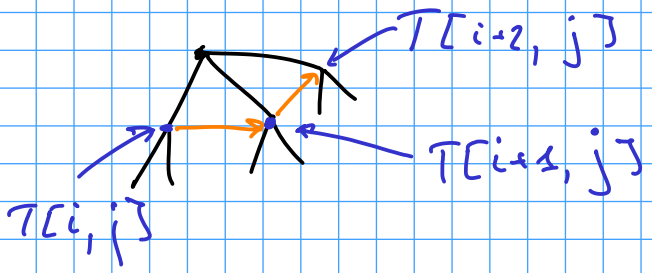
Внутри фазы:

продлеваем все суффиксы $T_1, T_2, \dots, T_{i-1}$ на одну букву и дописываем T_i

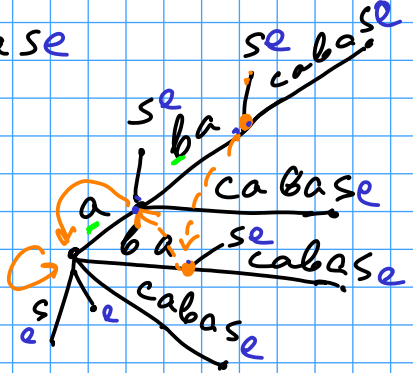
Время: $\sum_{i=1}^{n} \sum_{j=1}^{i} (i-j) = O(n^3)$

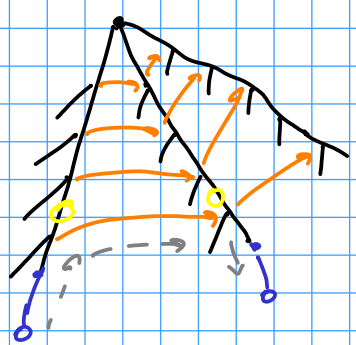
Типы продления:

1. продление листа
2. ветвление
3. "пустое продление"



abacabase





Переход к след суффиксу:

1. вверх до первой вершины // глубина -1
2. переход по суффиксу // глубина -1
3. вниз до 'листа' // глубина ↑↑

$O(n)$ на фазу.

Сложность $O(n^2)$

NB: Важно, что на шаге 3 при спуске вниз по ребрам мы можем найти конец следующего суффикса за количество операций пропорциональное кол-ву рёбер, а не длине строки. Для этого будем смотреть только на первую букву ребра.

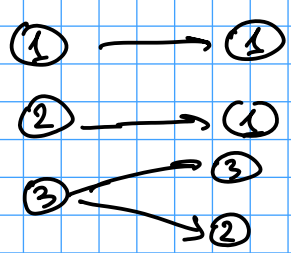
Как поддерживать суффиксные ссылки?

Если произошло ветвление, то запомним этот факт и нарисуем ссылку после того, как найдём её конец (при переходе к следующему суффиксу.

Типы продления:

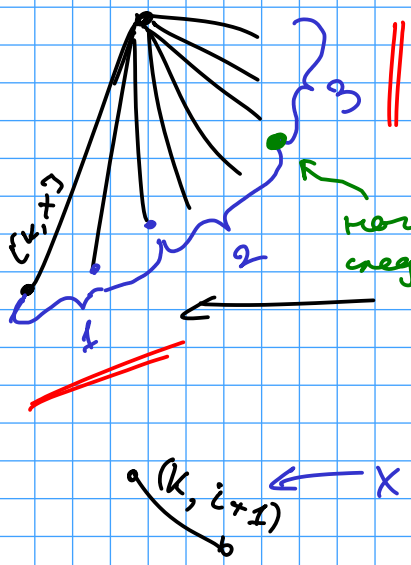
1. продление листа
2. ветвление
3. „пустое продление"

Правила:



Для конкретного суффикса

Внутри фазы



Если T_j пропущено пустым продолжением => можно перейти к след. фазе.

Все пропущенные листья добавим переместимую.


```
x = 0
k = 1
for i = 1 to n:
    x = x + 1
    for j = k to i:
        if ③:
            k = j
            break
    split
```

⇒ Сложность $O(n)$

Недостатки суффиксных деревьев:

1. Много памяти с большой фрагментацией
2. Зависит от $|\Sigma|$

Суффиксный массив

1	banana$
2	anana$
3	nana$
4	ana$
5	na$
6	a$
7	$

	anana$
	ana$
	a$
	banana$
	nana$
	na$
	$

2		SA →	4
4			1
6	←		5
1		ISA ←	2
3			6
5			3
7			7

P = nan

Поиск шаблона бинарным поиском

$O(\log|T| \cdot |P|)$

Можно ускорить до $O(\log|T| + |P|)$

Построение за $n \log n$

banana$
anana$
nana$
ana$
na$
a$
$

→

anana$
ana$
a$
banana$
nana$
na$
$

Σ ↓

Σ' ∩ $\Sigma \times \Sigma$

1 anana$
1 ana$ 2
2 a$ 4
3 banana$
4 nana$ 4 5
4 na$ 5
5 $ 6

→

T_1^{-1} ← T_3

1 anana$ 3
2 ana$
3 a$
4 banana$
5 nana$
6 na$
1 $

$\Sigma'' \subset \Sigma' \times \Sigma'$

$O(n \log n)$

Naive binary search

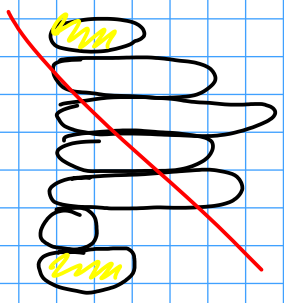
$O(|P| \cdot \log |T|)$

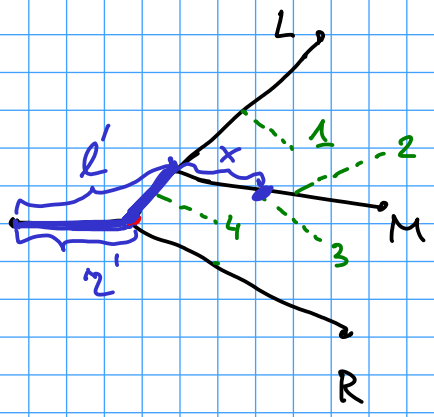
Зад. 1

ababc S_1
abbac S_2
abcab S_3
abcda S_4
abced S_5
abdad S_6

longest common prefix = lcp

$|lcp(S_1, S_6)| = \min_i |lcp(S_i, S_{i+1})|$





$l = |lcp(L, P)|$
$2 = |lcp(R, P)|$
$l' = |lcp(L, M)|$
$2' = |lcp(R, M)|$

] в рекурсии

] ?

$l \geq 2$: ①, ②, ③, ④

$l' > l \Rightarrow$ ④, go right

$l' < l \Rightarrow$ ①, go left

→ $l = l' \Rightarrow$ ② vs ③

$l < 2$:

симметрично.

$O(|P| + \log(T))$

x символов после текущего на этом шаге

$2 \leftarrow 2' + x$

рекурсия ②

$l \leftarrow l' + x$

рекурсия ③

Как считать $lcp(L, M)$?

1. Предпосчёт $lcp(T_i, T_j)$ $O(|T|^2)$
2. Предпосчёт только для тех (i, j), которые встречаются в дереве $O(|T|^2)$ $O(|T|)$ памяти.

3. Вобайте посчитаем массив $\overline{LCP}$:

$$\overline{LCP}[i] = |lcp(T_{SA[i]}, T_{SA[i+1]})|$$

Утв 1 + RMQ за $O(1)$ $\Rightarrow$

по $\overline{LCP}$ посчитать

$$|lcp(T_{SA[i]}, T_{SA[j]})| = \min_{i \le k < j} \overline{LCP}[k]$$

abcabc
abcdef
abcdeg
bcab
bcad
beaf

$O(n^2)$

Алгоритм:
[Arimura,
Arikawa,
Lee,
Kosaki,
Park]

SA ... $\underset{i}{[k][\ell]}$ ...

$lcp(T_{SA[i]}, T_{SA[i+1]}) = L > 0$

$\Downarrow$

$lcp(T_{k+1}, T_{\ell+1}) = L - 1$

$\overline{LCP}[i] = L$

$\geq L-1$

SA ... $[k+1]$ ... $[\ell+1]$

$\overline{LCP}[ISA[k+1]] \geq L - 1$

$L = |lcp(T_1, T_{SA[ISA(1)+1]})|$

for $i = 2$ to n:

$L = \max(0, L-1)$

while $T_i[L+1] = T_{SA[ISA(i)+1]}[L+1]$

$L = L + 1$

$O(n)$