Алгоритмы и структуры данных. Семинар какой-то(второй листок по строкам).
Суффиксные структуры.
Григорьев Дмитрий БПМИ-163

Задача 1.

a) Для начала построим суффиксный массив и lcp(где $lcp[i]$ равен длине наибольшего общего префикса суффиксов $p[i]$ и $p[i+1]$) за $O(n \log n)$.
Далее будем проходить по отсортированным суффиксам в суффиксном масиве и для каждого суффикса будем смотреть на длину общего наибольшего префикса рассматриваемого суффикса и предыдущего суффикса, длина уникальной подстроки от данного суффикса равна $n - p[i] - lcp[i-1]$(кроме $i = 0$, для нее длина уникальных подстрок равна $n - p[i]$). Почему это так – очевидно: для суффиксов начинающихся с одинаковых подстрок(а такие суффиксы идут по порядку, так как они отсортированны) мы один раз учитываем эту подстроку, а остальной раз, если есть суффиксы, которые начинаются с одинаковых подстрок мы вычитаем длину этой подстроки(длину наибольшего общего префикса), так как каждый префикс дает нам уникальную строку(именно поэтому мы считаем количество различных префиксов у суффиксов).
В итоге, чтобы посчитать количество различных подстрок нужно посчитать

$$\sum_{i=0}^{n}(n - p[i]) - \sum_{i=0}^{n-1} lcp[i]$$

И мы потратим $O(n \log n)$ времени, так как мы предпосчитали суффиксный массив и lcp за $O(n \log n)$ и прошли по ним один раз.

b) Для начала построим суффиксное дерево за $O(n)$. Далее посмотрим на структуру суффисорго дерева и поймем, что все префиксы суффиксов хранятся только один раз(очевидно из-за структуры суффиксного дерева). Далее заметим, что каждая подстрока соответствует какой-то позиции на ребре в суффиксном дереве. Поэтому количество различных подстрок – это сумма длин всех ребер в дереве(длина ребра – это длина подстроки написанной на ребре). Так как вершин в суфф. дереве не более $2n + 1$, то проход по всем ребрам – $O(n)$ времени.

Задача 2.

Построим суффиксное дерево за $O(n)$. Далее для каждой вершины посчитаем количество листьев в ее поддереве – $leaves[v]$(это и есть количество вхождений строки, соответствующей пути от корня до v). Тогда нам нужно найти такую вершину, для котрой $leaves[c] \cdot len[v]$ будет максимально($len[v]$ – длина строки от корня до вершины v). Так как максимальное количество вершин в дереве $2n + 1$, то мы можем обойти все суфф. дерево обходом в глубину и найти рефрен строки за $O(n)$.

Задача 3.

a) Просто бинпоиском найдем строку p, или поймем что ее нет. По суфф. массиву бинпоиск работает $O(\log n)$ времени и для сравнения суффикса с p используем $O(m)$ времени. Итого получили $O(m \log n)$.

b)Нам нужно проверить, является ли строка t подстрокой s. Теперь будем так же делать бинпоиск, но для каждого шага будем поддерживать $l_{pref} = lcp(p[l], t)$ и $r_{pref} = lcp(p[r], t)$.
$mid = (l + r)/2$.
Если $lcp(p[l], p[m]) < l_{pref}$, то сдвигаем правую границу и обновляем $r_pref = lcp(p[l], p[m])$
Если же $lcp(p[l], p[m]) \geq l_{pref}$, то тогда сдвинем левую границу и найдем $lcp(p[m], t)$, но делать это будем, стартовав с l_{pref}, так как $lcp(p[m], t) \geq l_{pref}$. Тогда на таком шаге мы сделаем не больше, чем $lcp(p[m], t) - l_{pref} + 1$ сравнений.
Тогда в итоге будет потрачено $O(m + \log n)$ времени.

Задача 5.

Построим суффиксное дерево для строки sA_1tA_2, где A_1 и A_2 – разные разделители, которые не встречаются в s и t. Потом уберем из дерева ребра от вершин, для которых последний символ строки, которая получается на пути от корня до этой вершины, равен разделителю.
Теперь в листья для которых последний символ на ребре, ведущим в эту вершину равен A_1 поставим 1, и запомним все эти листья в порядке обхода(пусть это будет $v_1, v_2, \ldots v_k$). Теперь в вершины равные $lca(v_1, v_2), lca(v_2, v_3) \ldots lca(v_{k-1}, v_k)$ положим -1.
Теперь в листья для которых последний символ на ребре, ведущим в эту вершину равен A_2 поставим 1, и запомним все эти листья в порядке обхода(пусть это будет $u_1, u_2, \ldots u_m$). Теперь в вершины равные $lca(u_1, u_2), lca(u_2, u_3) \ldots lca(u_{m-1}, u_m)$ положим -1.
Теперь посчитаем сумму в каждом поддереве.
Тогда вершина в суфф. дереве соответствует подстроке входящей в s и t тогда и только тогда когда, сумма в ее поддереве равна 2.
Тогда $O((|s| + |t|) \log(|s| + |t|))$ получаем из-за того, что ищем lca, а остальные действия выполняются за $O(|s| + |t|)$. В итоге получаем $((|s| + |t|) \log(|s| + |t|))$.