Алгоритмы и структуры данных поиска.
Домашняя работа №1
Стрелец Владимир

Пусть мы хешируем хорошо, то есть ищем хеши по простому модулю M, а также основанием для хеша служит число r -- первообразный корень для M.
Павел Иржавский говорил, что тогда хеши меньше кореллируют.

Задача 1.

a) **Препроцесс.** найдём массив кумулятивных сумм хешей cumulative_hashes для подстрок, то есть массив, состоящий из элементов h(s[1, 1]), h(s[1, 2]), … , h(s[1, n]). Для полиномиального хеширования это делается за O(n), так как нахождение хеша строки по хешу строки без последнего элемента производится за O(1).

Итак, у нас есть массив кумулятивных хешей, причём
cumulative_hashes[i] = sum for j in [1, j] s[j] * r^j.

Также заведём массив всех степеней r от первой до n-й (также за O(n)).

Сравнение. Как теперь сравнивать s[l1, r1] и s[l2, r2]?

Если их длины различаются (то есть не равны числа (r1 - l1 + 1) и (r2 - l2 + 1)), то эти подстроки различные. Поэтому пусть длины подстрок равны.

Заметим, что
A = cumulative_hashes[r1] - cumulative_hashes[l1 - 1] = h(s[l1, r1]) * r^{l1 - 1}, а
B = cumulative_hashes[r2] - cumulative_hashes[l2 - 1] = h(s[l2, r2]) * r^{l2 - 1}

Пусть для определённости l2 > l1. Теперь, домножив A на r^{l2 - l1}, можем сравнить A' = A * r^{l2 - l1} и B. Если они равны, то строки равны (так как мы доверяем хеш функции). Если не равны, то подстроки, соответственно, не равны.

b) **Препроцесс:** Составим 2 массива кумулятивных сумм хешей для данной строки. Первый -- cumulative_hashes -- такой же, как и в пункте *a),* то есть для всех префиксов строки s. Второй же -- cumulative_hashes_reverse -- для всех суффиксов строки s (в нём на i-той позиции записан хеш строки s[i, n]).

Сравнение. Хеш строки s[l1, r1] мы находить умеем. Как найти хеш реверсной строки? Для этого воспользуемся массивом кумулятивных сумм для суффиксов строки s.

cumulative_hashes_reverse[l2] - cumulative_hashes_reverse[r2 + 1] =
= r^{n - r2 - 1} * h(reverse(s[l2, r2]))

Теперь, домножив на нужную степень числа r, сравним полученные результаты и дадим окончательный ответ.

c) Эта задача -- частный случай задачи *б).* только здесь сравниваются строки s[l, r] и reverse(s[l, r]). Ничего нового.

d) **Препроцесс.** Строим массив кумулятивных сумм хешей для данной строки.

Сравнение. Сравним сразу хеши строк s[i...i] и s[j...j]. Если они не равны, то длина наибольшего префикса равна 0. Если же они равны, то будем сравнивать хеши таких подстрок, чтобы правая граница сравниваемых подстрок "прыгала" бинарным поиском по заданным суффиксам. Таким образом за O(log n) мы найдём искомую величину.

Покажем часть этого поиска: последовательно сравниваем хеши строк
s[i...i + 2^0] и s[j...j + 2^0],
s[i...i + 2^1] и s[j...j + 2^1],
s[i...i + 2^2] и s[j...j + 2^2] и так далее.
Как только найдём неравные хеши, мы определим верхнюю границу для длины максимального общего префикса. Далее продолжаем аналогично "прыгать" правыми границами подстрок бинарным поиском по индексам
[i + 2^{k - 1}, i + 2^k] и [j + 2^{k - 1}, j + 2^k], где s[i...i + 2^k] и s[j...j + 2^k] -- первые неравные подстроки из первого этапа.

e) Сведём эту задачу к прошлой. Аналогичный препроцесс -- O(n).
Для каждого i = 1, 2, … , n нам нужно найти наибольшие общие префиксы для строк s[1...n] и s[i...n]. Это мы уже умеем делать за O(log n). Таким образом, нужно n раз повторить операцию, которая занимает O(log n) времени. Это займёт O(n log n).

f) Препроцесс аналогичен оному из пункта *d).* Далее, за O(log n) найдём длину наибольшего общего суффикса для строк s[i...n] и s[j...n] (способом, описанным в пункте *d)).* Пусть его длина равна k. Далее, честно сравнив элементы s[i + k] и s[j + k], если таковые имеются, то есть суффиксы не выходят за границы строки, и получим ответ, какой из суффиксов лексикографически меньше. Если же такого элемента нет, то лексикографически меньше будет тот суффикс, который короче.

g) Другими словами, в задаче нужно отсортировать (не в памяти, а как бы в уме) все суффиксы данной строки s. Как известно, если время сравнения двух любых элементов из какого-либо сортируемого множества равно compare(a), то время сортировки этого множества равно $O(n \log n * \text{compare}(a))$. В этой задаче compare(a) = $O(\log n)$ (это показано в пункте *f)).* Поэтому имеем время сортировки n суффиксов, равное $O(n \log n * \log n) = O(n \log^2 n)$.

То есть, мы построим нужный нам суффиксный массив, если будем вместе с сортируемыми (в уме) суффиксами будем сортировать массив a = [1, 2, … , n], где i-й элемент является "ярлыком" для i-го суффикса.

Задача 2.

a) Мы можем взять, например, функцию h(A, m) = "сумма хешей всех элементов мультимножества A" mod 2^m. И тогда для любых мультимножеств A = B выполняется h(A, m) = h(B, m).

b) Возьмём, например, функцию h(A, m) = const, такая что 0 <= const < 2^m. И тогда пересчёт функции при добавлении элемента будет за $O(1)$, что удовлетворяет условию.

Если по какой-то причине кому-то не нравится такая хитрая функция, то предложим функцию $h(A + x, m) = h(A, m) + h(x)^{|A|}$. Пересчёт хеш-функции мультимножества производится за $O(\log |A|)$ (если использовать быстрое возведение в степень), что удовлетворяет условию.

c) Функция h(A, m) = "количество элементов в мультимножестве A" mod 2^m. Понятно, что для любого конечного m количество элементов в мультимножестве может быть равно 1, 2, … , $2^m - 1$, 2^m.

d) ----------

Задача 3.

3. Так как массив состоит из различных чисел, то если отнять от каждого элемента массива его индекс, полученный массив останется отсортированным (хотя в нём могут появиться равные числа).
Итак, будем искать такое число a[i] = i бинарным поиском (за $O(\log n)$) со сравниваемыми элементами (a[k] - k). Если нашлось число, "равное" нулю, то это значит, что найден элемент, равный своему индексу. Иначе, его нету в массиве.

4. Пусть $f(n) = n + \log_2 n$, а $g(n) = n$. Тогда $f(n) = O(g(n))$, но, так как $2^{f(n)} = 2^{n + \log n} = n\ 2^n$, то $2^{f(n)} != 2^{g(n)}$. То есть это может не выполняться.
Теперь же приведём пример, когда это выполняется: f(n) = n, g(n) = n^2 > n = f(n). И тогда $2^{f(n)} = O(2^{g(n)})$.

5. Предложу 3 способа сортировки:
Способ 1. (Как в условии). T(n, k) = (n + n) + (2n + n) + (3n + n) + … + ((k - 1) n + n) =
= 2n + 3n + 4n + … + kn = n * (k + 2) * (k - 1) = $O(n\ k^2)$.
Способ 2. На каждой итерации слияния будем искать минимальный элемент из всех первых элементов последовательностей. Так как последовательностей k, а количество все элементов равно количеству итераций и равно k*n , то получим аналогичную сложность
T(n, k) = $O(n\ k^2)$
Способ 3. Храним k минимальных элементов (по одному из каждой последовательности) вместе с номером последовательности, из которой пришёл этот элемент, например, в бинарной куче. И тогда имеем k*n итераций, на каждой из которых удаляем минимальный элемент из кучи, дописываем его в итоговую отсортированную последовательность, и добавляем следующий элемент из той же последовательности в кучу. Таким образом, на каждую итерацию тратится $O(\log k)$ времени. Итоговая сложность -- $O(n\ k \log k)$.