Студент: Тимур Хабибуллин
Группа: 1
Дата: 5 мая 2022 г.

1. В этом коде проблема в том, что цикл по i начинается не с 1, а с нуля. Тогда происходит следующее: при $i = 0$ выбирается $k = 0$, а затем сравниваются элементы `s[i+k]==s[i]`, что при данных i, k есть один и тот же элемент. Тогда строка сравнится сама с собой и $r = k = n - 1$, $l = 0$. После этого шага последний `if` больше никогда не сработает, потому что $r = n - 1$ и $l = 0$ будет всегда. А из-за того, что $l = 0$, в начале каждой следующей итерации цикла будет $k = 0$ и строки s и $s[i :]$ будут сравниваться с самого начала.

   Асимптотика $\Omega(n^2)$ достигается в случае, если строка состоит из одинаковых элементов. Тогда на каждой итерации цикла k будет пробегать от i до конца, т.е. $n-1$. Тогда алгоритм будет работать за
   $$T(n) = n + (n - 1) + \cdots = \Omega(n^2)$$

4. Пронумеруем суффиксы естественным образом (i-суффикс — `s[i,n)`) и сдетаем поиск $k-$ порядковой статистики в списке всех суффиксов с помощью Parition из QuickSort. Умеем находить первое различие и сравнивать 2 строки с помощью хэша за $\log(n)$ времени, поэтому делаем как раньше — выбираем случайный опорный суффикс ($i-$й какой-то), проходим за $O(n)$ шагов по всему списку суффиксов и находим порядковую статистику для $i-$го суффикса, пусть k_i. Если $k < k_i$, то требуемая $k-$порядковая статистика где-то слева и вызываемся в нее, если $k > k_i$ — справа, иначе мы нашли то что нужно. Работает это все за $O(n \log(n))$, потому что оригинальный алгоритм для чисел работает за $O(n)$, а сейчас сравнение 2 элементов не за $O(1)$, а за $O(\log(n))$.

5. Пусть мы стоим в t некоторой позиции $i \in [0, |t| - |s|)$), помощью хэша найдем первое различие между строкой s и подстрокой $t[i, i + |s|)$ за $O(\log |s|)$, потом перейдем через него и будем искать первое различие у оставшихся кусочков строк и тд. Если нашли k ошибок и не дошли до конца строк s и t, то тогда ошибок слишком много.

   Алгоритм работает за $O(|t|k \log |s|)$, потому что мы перебираем $O(|t|)$ индексов i и на каждом таком шаге делаем $O(k \log |s|)$ операций

6. Пусть дана строка s. Сделаем бин поиск по ответу (длине подстроки l): для данного l заведем хэш-таблицу и будем идти по s слева на право и складывать в хэш-таблицу тройки

   `lst = [i, 1, h(s[i, i+l))],`

   где второй аргумент - сколько раз встретился этот элемент без пересечений. Проверка на коллизии по третьему аргументу пары. Тогда если на $k-$ шаге мы получили строку

   `h(s[k, k+l)) == h(s[i, i+l)),`

   то проверяем, что что они не пересекаются, т.е. $k \notin (i - l, i + l)$, и если все окей, то проверяем `lst[2] == 1` и обновляем `lst[2] = 2`, и тогда это возможный претендент на ответ для длинны l. Если последняя проверка не проходит, то значит эта подстрока встретилась нам много раз и раз и она нам не подходит (можно еще сделать проверку `lst[2]==2` и если верно, то удалим эту подстроку из множества претендентов на ответ данной длины l).

   Алгоритм работает за $O(n \log(n)$, потому что делаем бин поиск по l и на каждом шаге делаем не более $O(n)$ операций.

7. (a) За $O(n)$ пересчитаем хэш функцию на префиксах для s и $s[:: -1]$. Если строка t — палиндром, то $t[:: -1] == t$. Тогда на каждый запрос $[l, r)$ будем проверять, что $h(s[l : r]) == h(s[l : r : -1])$ и если достигается равенство, то это палиндром.

   (b) Идея: найти палиндром максимальной длины с серединой в данном символе. Тогда мы найдем количество всех палиндромов с центром в этом же символе и сможем добавить это число к ответу.

   Здесь можно воспользоваться трюком с превращением четных палиндромов в нечетные и все делать единообразно. Пусть нам дана строка s, создадим новую строку, в которой между всеми символами s, а так же в начале и в конце вставлена, например, звездочка $*$, теперь это строка t.

   Пусть мы стоим в i позиции, если найдем палиндром наибольшой длины в центре с $i-$м символом в t. Выбираем верхнюю границу для полуширины $l_{max} = \min(i, n-1-i)$, а дальше для $l \in [0, l_{max}]$ бин поиском сравниваем с поломощью хэша подстроки $s[i-l, i)$ и $s[i+1, i+l+1)[:: -1]$ (цетральный символ не включаем). Если нашли наибольшее $l = N$, то добавляем в счетчик всех палиндромов `acc += N//2` (здесь важно округлениее вниз). Алгоритм работает за $O(n \log(n))$, потому что для каждого $i \in O(n)$ делаем $O(\log(n))$ шагов бинпоиска