π-функция. Префикс-функция от строки s определяется так: для каждой позиции i это такая максимальная длина l, что $s_{0,l-1} = s_{i-l+1,i}$. То есть, мы берем наибольший префикс строки s, совпадающий с суффиксом, заканчивающимся в позиции i. При этом мы искусственно запрещаем делать $l = i+1$ (то есть запрещаем брать префикс, совпадающий с суффиксом).

Как эффективно вычислять эту функцию? Можно заметить, что $\pi_{i+1} \leq \pi_i + 1$. При этом нижней границы нет. То есть, если мы будем итеративно считать префикс-функцию, то значения у нас не могут увеличиваться больше чем на 1 на каждом шаге, поэтому суммарно уменьшаться на 1 они будут $O(n)$ раз.

Тогда префикс-функцию можно считать, например, так: сначала положить $\pi_i = \pi_{i-1} + 1$, а потом постепенно уменьшать, и делать проверку на равенство подстрок с помощью хэшей. Но стандартный алгоритм делает так: он по очереди пытается приписать символ к π_{i-1}, $\pi_{\pi_{i-1}-1}$, ... Это верно, потому что если строка максимальна, то ее надо было продлить относительно прошлой подстроки. Если так не получалось сделать (символы не совпадали), то надо было попробовать максимальную длину, которая подходит, после π_{i-1}. Но это ровно и есть значение префикс-функции от $s_{\pi_{i-1}}$, потому что $s_{0,l-1} = s_{i-l+1,i}$, и потому что мы запретили брать префикс, равный суффиксу. Тогда этот алгоритм так же работает за $O(|s|)$.

Чтобы найти подстроку t в строке s, можно запустить этот алгоритм на строке $t\$s$, после чего найти все точки, в которых $\pi_i = |t|$. Тут можно еще сэкономить память, и использовать только $O(|t|)$ памяти.

Автомат префикс-функции. Мы хотим, чтобы Кнут-Морис-Пратт (поиск подстроки в строке) работал неамортизированно. Для этого можно сначала насчитать все переходы, и потом делать переходы. А именно, мы хотим делать $go_{v,c}$ — переход из вершины v по символу c. Вершину v мы будем отождествлять с префиксом длины v. Тогда, если $s_{v+1} = c$, то $go_{v,c} = v + 1$, а иначе $go_{v,c} = go_{\pi_v,c}$.

Z-функция. В этот раз мы хотим найти для каждой позиции наибольший префикс $s_{0,n}$ и $s_{i,n}$. Тут функция монотонна, и ее можно считать наивно: пока можно увеличить значение, и оно корректно, надо увеличивать. Линейный алгоритм будет основан примерно на этом.

Будем строить z-функцию итеративно. Тогда каждый шаг алгоритма давал нам новую подстроку $s_{i,i+z_i}$. Мы их будем обозначать за l_i, r_i. Поддерживать мы из них будем всегда тот подотрезок, у которого правая граница как можно больше.

Тогда посмотрим на какое-то текущее i. Если $i \in [l, r]$, то $z_i \geq \min(r - i, z[i - l])$. Левое число нужно, чтобы мы не вышли за границу отрезка (потому что символы вне отрезка для нас неизвестные), а правое — это то, как мы используем текущий отрезок $[l, r]$. А именно, поскольку мы знаем, что наша строка сейчас такая же, как и $s_{0,r-l}$, то можно «подглядеть», чему было равно значение $z_{i'}$, где i' соответствует парному символу для i.

Дальше мы можем просто наивно увеличивать значение z-функции, и этого хватит для линейного времени. Время окажется линейным, потому что наивное увеличение надо использовать, только если $i + z_i = r$ (то есть, если мы хотим выйти за границы текущего отрезка). А это значит, что у нас сдвинется число r. Поскольку r может сдвинуться не более, чем n раз, то получаем оценку $O(|s|)$ времени и памяти.