Z-функция

Для данной строки S (длины n) определим целочисленный массив z. z[0] не опредлено, остальные элементы определены следующим образом: z[i] - длина максимального общего префикса строки S и строки S[i..n-1].

Вычислим z[1] за линейное время. Теперь пусть мы хотим вычислить z[k], при этом для всех позиций левее k ответ уже вычислен. Пусть R - это максимум по всем i + z[i] для i < k, а L - соответствующее ему i. Тогда в силу определения z-функции *s[0..R-L-1] = s[L..R-1]* и *s[R-L]* не равно *s[R]*. Отсюда следует, что *z[k] >= min(z[k - L], R - k)*.

Док-во: Действительно, по определению z-функции равны строки *s[0..z[k - L] - 1]* и *s[(k - L)..(k - L) + z[k - L] - 1]* (а следовательно, равны и все их префиксы). Отсюда получаем *s[0..min(z[k - L], R - k) - 1] = s[(k - L)..(k - L) + min(z[k - L], R - k) - 1] = s[k..k + min(z[k - L], R - k) - 1]*.

В последнем равенстве мы воспользовались тем, что *s[0..R - L - 1] = s[L..R - 1]* и добавили к обоим границам второй строки *L*, получив таким образом границы третьей, искомой. При этом нам пришлось заменить *z[k - L]* на *min(z[k - L], R - k)*, чтобы не вылезти за *R - L - 1* в левой части равенства и за *R - 1* в правой соответственно. □

Таким образом, мы можем проинициализировать *z[k] = min(z[k - L], R - k)*, а затем “досчитать” z-функцию за линейное время, увеличивая её на единицу пока равны символы *s[k + z[k]]* и *s[z[k]]*. При этом можно заметить, что если *k + z[k - L] < R*, то *z[k] = z[k - L]*, то есть, мы не сделаем ни одной итерации по увеличению *z[k]*, а иначе с каждым увеличением *z[k]*, *R* будет увеличиваться вместе с ним.

Чтобы определить асимптотику алгоритма, достаточно заметить, что на каждом шаге либо мы вычисляем новую z-функцию за O(1), либо увеличиваем R. При этом мы никогда не уменьшаем R. На последнем шаге R = n, а значит мы не могли увеличивать R более n раз. Отсюда время работы алгорима линейно.

На подумать: пусть вы имеете строки S и P, требуется найти все вхождения P в S за линейное время с помощью z-функции. Подсказка: рассмотрите строку P#S.

Примечание: в алгоритме нахождение z-функции используется идея о том, что есть 2 равных блока в строке и некоторые значения для правого блока мы можем получать, модифицируя значения для левого блока. Существует линейный алгоритм Манакера, находящий все подпалиндромы заданной строки. Он основан на схожей идее. Только там блоки равны с точностью до переворота. Его можно найти по ссылке: http://codeforces.com/blog/entry/12143

Стоит отметить, что алгоритм можно существенно упростить, если рассматривать палиндромы только нечетной длины. Для этого можно пойти на следующий трюк: вставить метасимвол ‘#’ между каждой парой символов. Таким образом код (и восприятие алгоритма) существенно упростится, т.к. у каждого палиндрома начальной строки теперь будет точный центр - у палиндромов нечетной длины он находится в символе алфавита, а у палиндрома четной длины - в метасимволе ‘#’.

Префикс-функция

Для данной строки S назовём её гранью строку P, если P является одновременно префиксом и суффиксом строки S. Также мы будем иногда называть гранью длину такой строки P.

Для данной строки S определим целочисленный массив pi. pi[0] полагается равной 0 (здесь и далее будет использоваться 0-индексация), остальные элементы определены следующим образом pi[i] - максимальная грань строки S[0..i]. Массив pi будем называть префикс-функцией.

Пусть мы хотим вычислить pi[k], а все предыдущие уже вычислены. Заметим следующие свойства префикс-функции:

1) если строка s[0..k] имеет грань t, то строка s[0..k-1] имеет грань t - 1

2) если строка s[0..k-1] имеет грань t-1 и s[k] = s[t], то s[0..k] имеет грать t. Т.е. по сути мы за O(1) можем проверить, возможно ли продлить грань строки до грани строки на 1 большей.

Тогда перебрав все грани строки s[0..k-1] по убыванию длины мы можем выбрать наибольшую из них, которую можно продлить до грани s[0..k]. Так мы получим максимальную грань для s[0..k]. Для строки s[0..k-1] максимальная грань равна pi[k - 1], следующая по величине равна pi[pi[k - 1] - 1] и т.д.

Теперь строже сформулируем алгоритм поиска префикс-функции. Последовательно будем вычислять префикс-функцию для всех префиксов данной строки. Для вычисления очередной префикс-функции переберём все грани строки по убыванию длины и выберем первую, которую получится продлить. После этого остановим цикл, продлим её, получим префикс-функцию следующего префикса.

Рассмотрим длину текущей грани префикса. Мы с гранью делаем два вида действий:

1) переходим к следующей по убыванию длины (тем самым уменьшая длину)
2) продляем текущую грань (тем самым увеличиваю длину на единицу)

Второй вид действия мы выполняем не больше n раз. А значит мы не можем более n раз делать первое действие (т.к. длина всегда не отрицательная)

А значит, что алгоритм выполняется за линейное время.

На подумать: пусть вы имеете строки S и P, требуется найти все вхождения P в S за линейное время с помощью префикс-функции. Подсказка: рассмотрите строку P#S.

Примечание: На каждом шаге алгоритм может тратить линейное время. Существуют алгоритмы, которые тратят на каждом шаге O(1) времени (если считать алфавит размера константой) или O(log n). Для этого нужно построить автомат префикс-функции (гуглить по запросу pattern matching automaton). Об этом, а также обобщении префикс-функции на случай множества строк (т.н. алгоритм Ахо-Корасик) можно прочесть в следующей статье:

http://codeforces.com/blog/entry/14854