**Оценка сложности префикс-функции, Z-функции и алгоритма КМП**

**Префикс-функция**

**Пре́фикс-фу́нкция** от строки **{\displaystyle S}S** и позиции **i** {\displaystyle i}iiв ней — длина **k** наибольшего *собственного* (не равного всей подстроке) префикса подстроки S[1…i] , который одновременно является [суффиксом](https://ru.wikipedia.org/wiki/%D0%A1%D1%83%D1%84%D1%84%D0%B8%D0%BA%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) этой подстроки.

То есть, в начале подстроки S[1…i] длины **i** нужно найти такой префикс максимальной длины k < 1, который был бы суффиксом данной подстроки (S[1…k] = S[(i – k + 1)…i].{\displaystyle S[1..i]}

vector<int> Prefix(string x) { // Префикс-Функция

vector<int> X(x.size()); // Вспомогательный вектор размером количества символов в строке

for (int i = 1; i < x.size(); i++) { // Пробегаемся по строке

int j = X[i - 1];

while (j > 0 && x[i] != x[j]) // Пока начало больше 0 и элементы не одинаковы

j = X[j - 1]; // Присваиваем предыдущий

if (x[i] == x[j]) // Если значения равны

j++; // То перемещаем на 1 дальше

X[i] = j; // Передаем полученное значение в элемент массива

}

return X;

}

Несмотря на то, что пункт 3 представляет собой внутренний цикл, время вычисления префикс-функции оценивается как O(|S|){\displaystyle O(|S|)}OO. Докажем это.

Все **i** {\displaystyle i}iделятся на:

1. Увеличивающие {\displaystyle k}kkk на единицу. Цикл проходит одну итерацию.
2. Не изменяющие нулевое {\displaystyle k}k. Цикл также проходит одну итерацию. Случаев 1 и 2 в сумме не более {\displaystyle \left|S\right|-1}|S| - 1 штук.
3. Не изменяющие или уменьшающие положительное {\displaystyle k}k. Поскольку внутри цикла значение {\displaystyle k}k может только уменьшаться, а увеличение {\displaystyle k}k возможно лишь на единицу, то суммарно значение {\displaystyle k}k не может уменьшиться более, чем {\displaystyle \left|S\right|-2}|S\ - 2 раза, что и ограничивает количество срабатываний внутреннего цикла.

Итого алгоритм требует не более 2|S| итераций, что доказывает порядок скорости O(|S|).

«Худшим» для алгоритма является случай обработки строки вида aa…ab.

**Z-функция**

Пусть дана строка s длины n. Тогда **Z-функция** ("зет-функция") от этой строки — это массив длины n, i-ый элемент которого равен наибольшему числу символов, начиная с позиции i, совпадающих с первыми символами строки s.

Иными словами, z[i] — это наибольший общий префикс строки s и её i-го суффикса.

Первый элемент Z-функции, z[0], обычно считают неопределённым. Будем считать, что он равен нулю (хотя ни в алгоритме, ни в приведённой реализации это ничего не меняет).

Время вычисления Z-функции  O (n).

vector<int> Z(string x) { // Z-Функция

vector<int> X(x.size()); // Вспомогательный вектор размером количества символов в строке

for (int i = 1, l = 0, r = 0; i < x.size(); i++) { // Пробегаемся по строке

if (i <= r) // Если входит в отрезок

X[i] = min(r - i + 1, X[i - l]); // Присваиваем наименьший

while (i + X[i] < x.size() && x[X[i]] == x[i + X[i]]) // Пока смещение + наименьший менее размера строки и элемент по значению равен элементу по смещению + значению

X[i]++; // Прибавляем единичку

if (i + X[i] - 1 > r) // Если лежит за пределами

l = i,

r = i + X[i] - 1;

}

return X;

}

Всё решение оформлено в виде функции, которая по строке возвращает массив длины n — вычисленную Z-функцию.

Внутри цикла for (int i = 1, l = 0, r = 0; i < x.size(); i++) мы сначала определяем начальное значение z[i] — оно либо останется нулём, либо вычислится на основе формулы.

После этого выполняется тривиальный алгоритм, который пытается увеличить значение z[i] настолько, насколько это возможно.

В конце выполняется обновление текущего самого правого отрезка совпадения [l;r], если, конечно, это обновление требуется — т.е. если i+z[i]-1 > r.

## Кнут — Моррис — Пратт

## Алгоритм Кнута — Морриса — Пратта (КМП-алгоритм) — эффективный алгоритм, осуществляющий поиск подстроки в строке. Время работы алгоритма линейно зависит от объёма входных данных, то есть разработать асимптотически более эффективный алгоритм невозможно.

int KMP(string X, string Y) { // Алгоритм Кнута — Морриса — Пратта

if (Y.size() == 0) // Если строка-образец пустая

return 0;

vector <int> A = Prefix(Y); // Получаем Префикс-Фукнцию по строке-образце

int k = 0;

for (int i = 0; i < X.size(); i++) // Пробегаемся по строке

while (true) {

if (Y.substr(k, 1) == X.substr(i, 1)) { // Если буква строки-образца равна букве строки

k++;

if (k == Y.size()) // Если вспомогательная переменная равна размеру строки-образца

return i + 1 - Y.size(); // Возвращаем номер элемента + 1 - размер строки-образца

break; // Останавливаем цикл

}

if (k == 0) // Если не нашлось

break; // Останавливаем цикл

k = A[k - 1];

}

return -1;

}

## Реализация построения автомата работает за O(n k), где *k* – мощность алфавита.