**Префикс-функция**

1.vector<int> PrefFunc(string s)//префикс функция

2. {

3. int n = (int)s.size();//переменная длина строки

4. vector<int> pr(n);//новый вектор от n

5. for (int i = 1; i < n; ++i)//идем по pr от i

6. {

7. int j = pr[i - 1];//переменная для подсчета нашего значения

8. while (j > 0 && s[i] != s[j])//сравниваем наши значения и смотрим на нашу j 9. пока не найдем отличающиеся

10. j = pr[j - 1]; //уменьшаем нашу j на 1 и снова идем смотрим

11. if (s[i] == s[j]) //если совпадают то

12. j++;//переходим к следующему значению

13. pr[i] = j;//если дошли до конца и нет совпадений то

14. }

15. return pr;//возвращаемся

16. }

Алгоритм:

1. При S[i+1]=S[k+1] - положить pi(S,i+1)=k+1

2. Иначе при k=0 - положить pi(S,i+1)=0

3. Иначе - установить k=pi(S,k) и перейти к пункту 1.

Время вычисления префикс-функции оценивается как O(n)

Все i делятся на:

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

Итого алгоритм требует не более 2 n итераций, что доказывает порядок скорости O(n). «Худшим» для алгоритма является случай обработки строки вида "aa…ab".

**Z-функция**

1.vector<int> ZFunc(string s) // з-функция

2.{

3. int n = (int)s.size();//переменная длина строки

4. vector<int> z(n);//новый вектор от n

5. for (int i = 1, l = 0, r = 0; i < n; ++i) //цикл для прозодки\создания

6. {

7. if (i <= r)//если наш элемент находится внутри от {L & R}

8. z[i] = min(r - i + 1, z[i - l]);//минимальное значение для {z от i}

9. while (i + z[i] < n && s[z[i]] == s[i + z[i]])//увеличиваем {z от i} на

10. максимальное значение

11. z[i]++;//прибавляем

12. if (i + z[i] - 1 > r)//если мы выходим за предел

13. l = i, r = i + z[i] - 1;//то сдвигаем наши границы

14. }

15. return z;//возвращаемся

16. }

Покажем, что приведённый выше алгоритм работает за линейное относительно длины строки время, т.е. за O (n) и покажем, что каждая итерация этого цикла while приведёт к увеличению правой границы r на единицу.

Для этого рассмотрим обе ветки алгоритма:

1)i <= r

В этом случае мы по приведённой формуле инициализируем значение z[i] некоторым числом z0.

2) i > r

В этом случае либо цикл while не сделает ни одной итерации (если s[0]!=s[i]), либо же сделает несколько итераций, продвигаясь каждый раз на один символ вправо, начиная с позиции i, а после этого — правая граница r обязательно обновится.

Поскольку i > r, то мы получаем, что действительно каждая итерация этого цикла увеличивает новое значение r на единицу.

Сравним это значение r-i+1 с величиной z0 и получаем три варианта:

1. z0 = r-i+1

В этом случае цикл while может совершить несколько итераций, однако каждая из них будет приводить к увеличению нового значения r на единицу: потому что первым же сравниваемым символом будет s[r+1], который вылазит за пределы отрезка [l;r].

1. z0 > r-i+1

Этот вариант принципиально невозможен, в силу определения z0.

Таким образом, мы доказали, что каждая итерация вложенного цикла приводит к продвижению указателя r вправо. Т.к. r не могло оказаться больше n-1, это означает, что всего этот цикл сделает не более n-1 итерации.

Поскольку вся остальная часть алгоритма, очевидно, работает за O (n), то мы доказали, что и весь алгоритм вычисления Z-функции выполняется за линейное время.

1. z0 < r-i+1

Докажем, что в этом случае ни одной итерации цикл while не сделает.

Это легко доказать, например, от противного: если бы цикл while сделал хотя бы одну итерацию, это бы означало, что определённое нами значение z0 было неточным, меньше настоящей длины совпадения. Но т.к. строки s[l....r] и s[0......r-l] совпадают, то это означает, что в позиции z[i-l] стоит неправильное значение: меньше, чем должно быть.

Таким образом, в этом варианте из корректности значения z[i-l] и из того, что оно меньше r-i+1, следует, что это значение совпадает с искомым значением z[i].

**КМП**

1. int Find\_Substirng(string pattern, string text)//функция

2. {

3. int end = -1;//переменная

4. vector<int> pf = prefix\_function(pattern);//вычисляем префиксы

5. int marker = 0;//счетчик = 0

6.

7. for (int i = 0; i < text.size(); i++)//идем по строке

8. {

9. while (marker > 0 && pattern[marker] != text[i]) { marker = pf[marker - 1]; 10. }//пока не дошли до конца

11. if (pattern[marker] == text[i]) marker++;//если символы подстроки и

12. паттерна совпадают то

13. if (marker == pattern.size())//идем и проверяем если дошли до конца

14. подстроки то

15. {

16. return end = i - marker + 1;//возвращаем счетчик

17. }

18. }

19. return end ;//возвращаем рес

20. }

Обозначим для удобства через n длину строки s, а через m — длину текста t.

Образуем строку s + # + t, где символ # — это разделитель, который не должен нигде более встречаться. Посчитаем для этой строки префикс-функцию. Теперь рассмотрим её значения, кроме первых n+1 (которые, как видно, относятся к строке s и разделителю). В нашем случае это pi[i] — фактически длина наибольшего блока совпадения со строкой s и оканчивающегося в позиции i. Больше, чем n, эта длина быть не может — за счёт разделителя. А вот равенство pi[i] = n означает, что в позиции i оканчивается искомое вхождение строки s (только не надо забывать, что все позиции отсчитываются в склеенной строке s+#+t).

Таким образом, если в какой-то позиции i оказалось pi[i] = n, то в позиции i - (n + 1) - n + 1 = i - 2 n строки t начинается очередное вхождение строки s в строку t.

Как уже упоминалось при описании алгоритма вычисления префикс-функции, если известно, что значения префикс-функции не будут превышать некоторой величины, то достаточно хранить не всю строку и префикс-функцию, а только её начало. В нашем случае это означает, что нужно хранить в памяти лишь строку s + # и значение префикс-функции на ней, а потом уже считывать по одному символу строку t и пересчитывать текущее значение префикс-функции.

Итак, алгоритм Кнута-Морриса-Пратта решает эту задачу за O(n+m) времени и O(n) памяти.