**Алгоритм Бойера — Мура**

**Алгоритм поиска строки Бойера — Мура,** считается наиболее быстрым среди алгоритмов общего назначения, предназначенных для поиска подстроки в строке. Был разработан [Робертом Бойером](https://en.wikipedia.org/wiki/Robert_S._Boyer) и [Джеем Муром](https://ru.wikipedia.org/w/index.php?title=%D0%9C%D1%83%D1%80,_%D0%94%D0%B6%D0%B5%D0%B9&action=edit&redlink=1) в 1977 году. Преимущество этого алгоритма в том, что ценой некоторого количества предварительных вычислений над шаблоном (но не над строкой, в которой ведётся поиск) шаблон сравнивается с исходным текстом не во всех позициях — часть проверок пропускаются как заведомо не дающие результата.

Рассмотрим сначала.

**Алгоритм Бойера — Мура — Хорспула** — алгоритм [поиска подстроки в строке](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B8%D1%81%D0%BA_%D0%BF%D0%BE%D0%B4%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B8), упрощённый вариант [алгоритма Бойера — Мура](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%91%D0%BE%D0%B9%D0%B5%D1%80%D0%B0_%E2%80%94_%D0%9C%D1%83%D1%80%D0%B0). Здесь требующая многих предварительных вычислений эвристика совпавшего суффикса опускается.

Идея алгоритма такова.

**1. Сканирование слева направо.** Как и в примитивном алгоритме, совмещается начало текста и шаблона. Если все символы шаблона совпали с наложенными символами строки, значит, подстрока найдена, и поиск окончен.

Если же какой-то символ шаблона не совпадает с соответствующим символом строки, шаблон сдвигается на несколько символов вправо. Эти «несколько» выбираются в соответствии с такой эвристикой.

**2. Изменённая эвристика стоп-символа.** Берём символ текста, оказавшийся над **последним** символом шаблона (независимо от того, где случилось несовпадение!). На рисунке это «b».

↓ стоп-символ

Текст a b a d b \* \* \* \*

Шаблон a b b a d

Следующая проверка a b b a d

Сдвигаем шаблон так, чтобы под стоп-символом оказалась буква «b» шаблона. Это реализуется с помощью таблицы смещений: для каждого символа алфавита храним максимально возможный сдвиг, не пропускающий стоп-символ. То есть (при нумерации строк с 1): *shift*(*c*) = |*needle*|−lastpos(*c*, *needle*[1..|*needle*|−1]), где lastpos — последнее вхождение символа в строку, *needle*[*a*..*b*] — операция взятия подстроки.

Для шаблона «abbad» таблица имеет следующий вид.

|  |  |  |  |
| --- | --- | --- | --- |
| **Символ** | a | b | (все остальные) |
| **Смещение** | 1 | 2 | 5 |

Для символов, не вошедших в шаблон, величина смещения устанавливается равной длине шаблона — 5. Последний символ шаблона при вычислении таблицы смещений не рассматривается (чревато [зацикливанием](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D1%86%D0%B8%D0%BA%D0%BB%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)).

Таблицу удобнее рассчитывать, проходя по всем символам шаблона:

for c=MIN\_CHAR..MAX\_CHAR

shift[c] = |needle|

for i=1..|needle|-1

shift[needle[i]] = |needle|-i

## Пример работы алгоритма.

Искомый шаблон — «abbad» (таблица для этого шаблона построена выше).

abeccacbadbabbad

abbad

Накладываем образец на строку. Последний символ исходной строки «с» не содержится в образце. Сдвигаем образец вправо на 5 позиций в соответствии со значением смещения для «с»:

abeccacbadbabbad

abbad

Три символа образца совпали, а четвёртый — нет. Сдвигаем образец вправо на 5 позиций в соответствии со значением смещения для «d»:

abeccacbadbabbad

abbad

Последний символ строки «a» не совпадает с символом шаблона. Сдвигаем образец вправо на 1 в соответствии со значением смещения для «a» и получаем искомое вхождение образца:

abeccacbadbabbad

abbad

*Отступление.*

Таблица *N* наиболее правых вхождений символов в строку *Т* заполняется элементарно:

N=new Array()

for(j=0;j<T.length-1;j++)

N[T.charAt(j)]=j+1

for(j in N)

WScript.Echo(‘N[‘,j, ‘] = ’,N[j])

Ранее мы выдвигали требования ≪известно, что *char* принадлежит строке Т≫ или ≪известно, что *char* не принадлежит строке Т≫. Как эффективно проверить их выполнение? Действительно, если каждый раз, отвечая на этот вопрос, просматривать

весь шаблон Т, то процесс поиска будет долгим и неэффективным. Используя таблицу *N*, можно дать ответ.

if (!(N [char])) {

WScript.Echo(char,’не входит в строку Т.’)

WScript .Echo(’Сдвиг на ’, T.length)}

else{

WScript .Echo (char,’входит в строку T.’)

WScript.Echo(’Сдвиг на ’, T.length -N[char])}

В алгоритме грубой силы делали сдвиг шаблона на 1 символ вправо независимо от того, все символы шаблона совпали с соответствующими символами строки *S* или нет.

і=0

Пока (i<=n-m)

Начало.цикла

j=0

…

і++

Конец\_цикла

В алгоритме Хорспула вместо строки «i++» вставляем

if (!(N [S[i+ m-1]]))

i=i+ m

else

i=i+ m-N[S[i+ m-1]]

**Описание алгоритма Бойера — Мура**

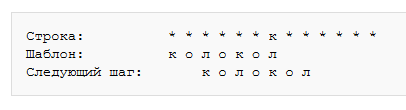
Алгоритм основан на трёх идеях.

**1. Сканирование слева направо, сравнение справа налево.** Совмещается начало текста (строки) и шаблона, проверка начинается с последнего символа шаблона. Если символы совпадают, производится сравнение предпоследнего символа шаблона и т. д. Если все символы шаблона совпали с наложенными символами строки, значит, подстрока найдена, и поиск окончен.

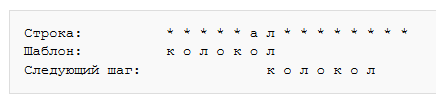
Если же какой-то символ шаблона не совпадает с соответствующим символом строки, шаблон сдвигается на **несколько** символов вправо, и проверка снова начинается с последнего символа.

Эти «несколько», упомянутые в предыдущем абзаце, вычисляются по двум эвристикам.

**2. Эвристика стоп-символа.** Предположим, что мы производим поиск слова «колокол». Первая же буква не совпала — «к» (назовём эту букву *стоп-символом*). Тогда можно сдвинуть шаблон вправо до *последней* его буквы «к».

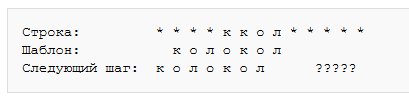


Если стоп-символа в шаблоне вообще нет, шаблон смещается за этот стоп-символ.



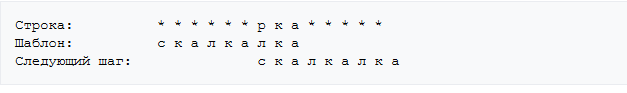
В данном случае стоп-символ — «а», и шаблон сдвигается так, чтобы он оказался прямо за этой буквой. В алгоритме Бойера-Мура эвристика стоп-символа вообще не смотрит на совпавший суффикс (см. ниже), так что первая буква шаблона («к») окажется под «л», и будет проведена одна заведомо холостая проверка.

Если стоп-символ «к» оказался за другой буквой «к», эвристика стоп-символа не работает.

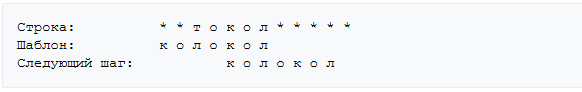


В таких ситуациях выручает третья идея алгоритма Бойера — Мура — эвристика совпавшего суффикса.

**3. Эвристика совпавшего суффикса.** Неформально, если при чтении шаблона справа налево совпал суффикс C, а символ b, стоящий перед C в шаблоне (т. е. шаблон имеет вид PbC), не совпал, то эвристика совпавшего суффикса сдвигает шаблон на наименьшее число позиций вправо так, чтобы строка C совпала с шаблоном, а символ, предшествующий в шаблоне данному совпадению C, отличался бы от b (если такой символ вообще есть). Формально, для данного шаблона T [ 0.. m − 1 ] считается целочисленный массив suffshift[0..m], в котором suffshift[i] равно минимальному числу j>0, такому что T [ i − j − 1] ≠ T [ i − 1 ] (если i > 0 и i − j > 0) и T [ i − j + k ] = T[ i + k ] для любого k ≥ 0, для которого выполняется 0 ≤ i − j + k < m и 0 ≤ i + k < m (для пояснения смотрите примеры ниже). Затем, если при чтении шаблона T справа налево совпало k символов T [ m − 1 ] , T [ m − 2 ] , … , T [ m − k ], а символ T [ m – k − 1 ] не совпал, то шаблон сдвигается на suffshift[m-k] символов вправо. Например:



В данном случае совпал суффикс «ка», и шаблон сдвигается вправо до ближайшего «ка», перед которым нет буквы «л».



В данном случае совпал суффикс «окол», и шаблон сдвигается вправо до ближайшего «окол», перед которым нет буквы «л». Если подстроки «окол» в шаблоне больше нет, но он начинается на «кол», сдвигается до «кол», и т. д.

**Внимание**: несовпадение буквы перед ближайшим вхождением совпавшего суффикса является необходимым условием. Если ограничиться только сдвигом до ближайшего вхождения совпавшего суффикса, то алгоритм может работать неприемлемо медленно. Например, при поиске шаблона cabababa… ba длины 2n в строке aaa… aababab…ba, содержащей 2n символов «a», за которыми следует строка baba…ba длины 2n, алгоритм, использующий сдвиги без учёта несовпадения символа, выполняет O ( n2 ) операций даже при использовании эвристике стоп-символов. С другой стороны доказано, что время работы алгоритма БМ, учитывающего несовпадение символов (то есть использующего массив suffshift, определённый выше), равно O ( n + m ) даже без использования эвристики стоп-символов.

Обе эвристики требуют предварительных вычислений — в зависимости от шаблона поиска заполняются две таблицы. Таблица стоп-символов по размеру соответствует алфавиту — O ( | Σ | ) (например, если алфавит состоит из 256 символов, то её длина 256); таблица суффиксов — искомому шаблону, т. е. O ( m ).

Опишем подробнее обе таблицы.

**Таблица стоп-символов**

В таблице стоп-символов указывается последняя позиция в шаблоне T (**исключая последнюю букву**) каждого из символов алфавита. Для всех символов, не вошедших в T, пишем 0, если нумерация символов начинается с 1, и пишем -1, если нумерация начинается с 0. Например, если T=abcdadcd, таблица стоп-символов будет выглядеть так (таблица приведена для случая строки, нумеруемой с нуля; при нумерации символов с единицы нужно прибавить ко всем числам единицу):

Символ a b c d [все остальные]

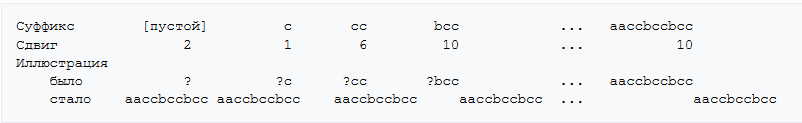
Последняя позиция 4 1 6 5 -1

Обратите внимание, для стоп-символа «d» последняя позиция будет 5, а не 7 — последняя буква не учитывается. Это известная ошибка, приводящая к неоптимальности. Для алгоритма БМ она не фатальна («вытягивает» эвристика суффикса), но фатальна для упрощённой версии алгоритма БМ — [алгоритма Хорспула](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%91%D0%BE%D0%B9%D0%B5%D1%80%D0%B0-%D0%9C%D1%83%D1%80%D0%B0-%D0%A5%D0%BE%D1%80%D1%81%D0%BF%D1%83%D0%BB%D0%B0).

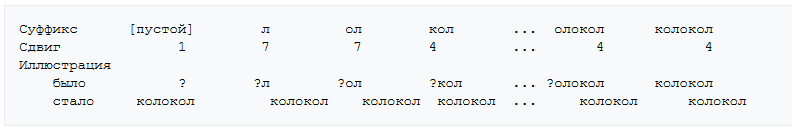
Если при сравнении справа налево шаблона T[0..m-1] со строкой S[i.. i+m-1] несовпадение произошло в позиции j, а стоп-символ c — T[j], то шаблон T необходимо сдвинуть на j - StopTable[c] символов.

**Таблица суффиксов**

Для каждого возможного суффикса C данного шаблона T указываем наименьшую величину, на которую нужно сдвинуть вправо шаблон, чтобы он снова совпал с C и при этом символ, предшествующий этому вхождению C, не совпадал бы с символом, предшествующим суффиксу C. Если такой сдвиг невозможен, ставится | T | = m (в обеих системах нумерации). Например, для T = aaccbccbcc T = aaccbccbcc будет:



Для T = колокол :



Существует алгоритм вычисления таблицы суффиксов suffshift[0..m] с временем работы O( m ).Этот алгоритм основан на тех же идеях, что и алгоритмы вычисления [префикс-функции](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D1%84%D0%B8%D0%BA%D1%81-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F) и Z-функции строки.

m=T.length

suffshift=new Array()

z=new Array()

maxZidx = 0

maxZ = 0;

for (var j = 0; j <=m; j++) {

z[j]=0

suffshift[j]=m

}

//Первый цикл

for (var j = 1; j < m; j++) {

if (j <= maxZ) z[j] = Math.min(maxZ - j + 1, z[j - maxZidx]);

while (j + z[j] < m && T.charAt(m - 1 - z[j]) == T.charAt(m - 1 - (j + z[j]))) z[j]++;

if (j + z[j] - 1 > maxZ) {

maxZidx = j;

maxZ = j + z[j] - 1;

}

}

for (var j = m - 1; j > 0; j--) suffshift[m - z[j]] = j; // цикл №1

r = 0;

for (var j = 1; j <= m - 1; j++) // цикл №2

if ((j + z[j]) == m)

for( ; r <= j; r++)

if (suffshift[r] == m) suffshift[r] = j;

WScript.echo(suffshift)

В первом цикле в коде воспроизведён код вычисления так называемой Z-функции z[1.. m−1 ] , но для перевёрнутой строки T [ 0.. m − 1 ]. А именно, для любого j, такого что 0 ≤ j < m − 1 , элемент z [ m − 1 − j ] содержит длину длиннейшего суффикса строки T [ 0.. j ], который также является суффиксом всей строки T. С помощью массива z далее вычисляется искомый массив suffshift[0..m].

Для доказательства корректности представленного кода удобно представлять себе, что анализируется строка Y [ 0.. m − 1 ] , которая равна перевёрнутой строке T. По определению suffshift, имеем suffshift[m − k]= j тогда и только тогда, когда j — это наименьшее положительное число, такое что либо 1) строка Y [ j . . m − 1 ] является префиксом строки Y [ 0.. k − 1 ], либо 2) строка Y [ j . . j + k − 1 ] равна Y[ 0.. k − 1 ], а символы Y [ j + k ] и Y [ k ] отличаются. В случае 2), по определению z, обязательно выполняется z [ j ] = k. Таким образом, пробегая по j от m − 1 до 1, цикл №1 находит все значения suffshift, полученные по правилу 2). Для вычисления значений suffshift, полученных по правилу 1), заметим, что, во-первых, если Y [ j . . m − 1 ] — префикс Y [ 0.. k − 1 ], то обязательно выполняется j + z [ j ] = m, а во-вторых, если suffshift[ r ] = j для какого-то r, то обязательно r ≤ j. Опираясь на эти два наблюдения, цикл №2 вычисляет оставшиеся неинициализированными значения suffshift (т. е. такие что suffshift[k] = m).

**Пример работы алгоритма**

Искомый шаблон — «abbad». Таблица стоп-символов будет выглядеть как

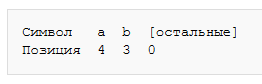
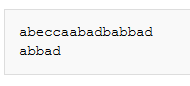
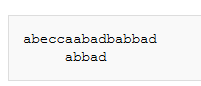


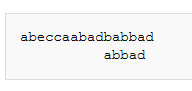
Таблица суффиксов для всех возможных суффиксов (кроме пустого) даёт максимальный сдвиг — 5.



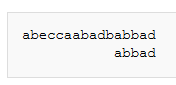
Накладываем образец на строку. Совпадения суффикса нет — таблица суффиксов даёт сдвиг на одну позицию. Для несовпавшего символа исходной строки «с» (5-я позиция) в таблице стоп-символов записан 0. Сдвигаем образец вправо на 5-0=5 позиций.



Символы 3—5 совпали, а второй — нет. Эвристика стоп-символа для «а» не работает (2-4=-2). Но поскольку часть символов совпала, в дело включается эвристика совпавшего суффикса, сдвигающая шаблон сразу на пять позиций!



И снова совпадения суффикса нет. В соответствии с таблицей стоп-символов сдвигаем образец на 1 позицию и получаем искомое вхождение образца:



### Реализация алгоритма.

Пусть массив сдвигов suffshift[0..m] для данного шаблона T [ 0.. m − 1 ] посчитан. Тогда реализация алгоритма Бойера — Мура для поиска T в тексте S[ 0.. n−1 ] без применения эвристики стоп-символов выглядит следующим образом:

і=0

Пока (i<=n-m)

Начало.цикла

j=m-1

Пока (j>=0 и S[i+j]==T[j])

Начало.цикла

j--

Конец\_цикла

Если (j<0)

Начало.если

Вывести (і)

Конец\_если

i+=suffshift[j+1]

Конец\_цикла

Такой алгоритм непригоден для поиска всех вхождений шаблона в текст. Алгоритм найдёт все вхождения, но в худшем случае, возможно, выполнит O ( m n ) операций, в чём легко убедиться, рассмотрев строку, состоящую из одних букв «a». Для поиска всех вхождений используют следующую модификацию за счёт так называемого правила Галиля.

і=0

bound = 0; //всегда либо bound = 0, либо bound = m - suffshift[0]

Пока (i<=n-m)

Начало.цикла

j=m-1

Пока (j>= bound и S[i+j]==T[j])

Начало.цикла

j--

Конец\_цикла

Если (j< bound)

Начало.если

Вывести (і)

bound = m - suffshift[0]

j = -1; //установить j так, как будто мы прочитали весь шаблон, а не только до границы bound

иначе

bound = 0

Конец\_если

i+=suffshift[j+1]

Конец\_цикла

Правило Галиля основано на следующем несложном наблюдении. Если вхождение T найдено в позиции i, то следующий поиск будет пытаться найти вхождение шаблона в позиции i′= i + suffshift[0] и, по определению suffshift, уже известно, что символы S [ i′ ] ,  S [ i ′ + 1 ] , … , S [ i + m − 1 ] совпадают с символами T [ 0 ] , T [ 1 ] , … , T [ m − suffshift[0] − 1 ]. Значит, при выполнении поиска справа налево для определения того, есть ли вхождение шаблона в позиции i′, нет смысла проверять символы T [ 0 ] , T [ 1 ] , … , T [ m − suffshift[0]− 1 ]. Именно для этого и служит переменная bound. Доказано, что такая эвристика помогает получить O ( n + m ) времени для поиска всех вхождений шаблона в строку.

Для включения эвристики стоп-символов надо заменить блок программы, начиная с «Если (j< bound)» в конце основного цикла:

if (j < bound)

{…

i += suffshift[0];

}

else

{…

i = max(i + suffshift[j+1], i + j + 1 - StopTable[S[i + j]]);

}

(Напоминание. В показанном ранее StopTable[char]=N[char], если char входит в строку T, иначе StopTable[char]=0.)

**Задание.**

Напоминание.

В текстовом файле input.txt записана на первой строке - строка *S*, на второй строке - строка *Т*. Требуется найти все вхождения строки *Т* в строку S, т. е. указать все позиции строки S, начиная с которых читается строка *Т:*

1. использование алгоритма Бойера — Мура — Хорспула ;
2. использование алгоритма Бойера — Мура с обеими эвристиками. Выдать совпавший суффикс строки T, если длина его больше 0.

Пробуем на примере:

1.

S: kgkgkgkgggfjnaaccbccbccjikmraaccbccbccjjjaaccbcbbccmkjijkomkuaaccbccbccniijniuhunaacccccbccaaccbccbcc

T: aaccbccbcc

2.

S:

абракадабра – магическое слово, употреблявшееся в древности и в средние века как заклинание против разных болезней. Слово абракадабра записывалось на амулетах или на бумаге в форме магического треугольника, в котором в каждой последующей строке убирается одна последняя буква, и так до тех пор, пока буквы не заканчиваются. Английский исследователь Ветхого Завета Джон Аллегро считает, что слово абракадабра происходит от месопотамского выражения АБ-БА-ТАБ-БА-РИ. абракадабракадабрара 5127401469476

T: абракадабра