

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСИС»

ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК (ИКН)

Курсовая работа на тему «Алгоритм Бойера-Мура для поиска подстроки в строке»

Выполнил: студент группы БИВТ-23-4
Борисенко Павел Дмитриевич

Москва 2025

Содержание

1.Введение

2.Описание алгоритма

- Идея алгоритма
- Псевдокод
- Пример выполнения

3.Анализ алгоритма

- Корректность
- Временная сложность
- Используемая память
- Плюсы и минусы данного алгоритма

4.Реализация

- Код на Python
- Описание кода

5.Заключение

1. Введение

Алгоритм Бойера-Мура, разработанный двумя учеными — Бойером (Robert S. Boyer) и Муром (J. Strother Moore), считается наиболее быстрым среди алгоритмов общего назначения, предназначенных для поиска подстроки в строке. Важной особенностью алгоритма является то, что он выполняет сравнения в шаблоне справа налево в отличие от многих других алгоритмов.

Он использует два эвристических правила — правило плохого символа и правило хорошего суффикса — для пропуска заведомо неподходящих участков текста, что ускоряет поиск.

Алгоритм Бойера-Мура считается наиболее эффективным алгоритмом поиска шаблонов в стандартных приложениях и командах, таких как Ctrl+F в браузерах и текстовых редакторах.

2. Описание алгоритма

Идея алгоритма:

- Сравнивает подстроку с текстом справа налево.
- При несовпадении символов использует правила для сдвига подстроки.

Алгоритм сравнивает символы шаблона X справа налево, начиная с самого правого, один за другим с символами исходной строки Y . Если символы совпадают, производится сравнение предпоследнего символа шаблона и так до конца. Если все символы шаблона совпали с наложенными символами строки, значит, подстрока найдена, и поиск окончен. В случае несовпадения какого-либо символа (или полного совпадения всего шаблона) он использует две предварительно вычисляемых эвристических функций, чтобы сдвинуть позицию для начала сравнения вправо.

Таким образом для сдвига позиции начала сравнения алгоритм Бойера-Мура выбирает между двумя функциями, называемыми эвристиками хорошего суффикса и плохого символа (иногда они называются эвристиками совпавшего суффикса и стоп-символа). Так как функции эвристические, то выбор между ними простой — ищется такое итоговое значение, чтобы мы не проверяли максимальное число позиций и при этом нашли все подстроки равные шаблону.

Алфавит обозначим буквой Σ .

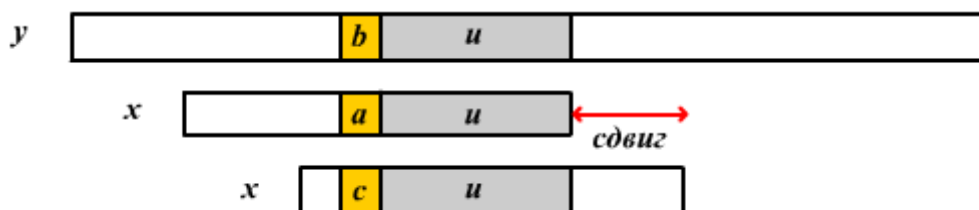
Пусть $|y|=n$, $|x|=m$ и $|\Sigma|=\sigma$.

Предположим, что в процессе сравнения возникает несовпадение между символом $x[i]=ax[i]=a$ шаблона и символом $y[i+j]=by[i+j]=b$ исходного текста при проверке в позиции jj . Тогда $x[i+1 \dots m-1]=y[i+j+1 \dots j+m-1]=u$ и $x[i] \neq y[i+j]$, и $m-i-1$ символов шаблона уже совпало.

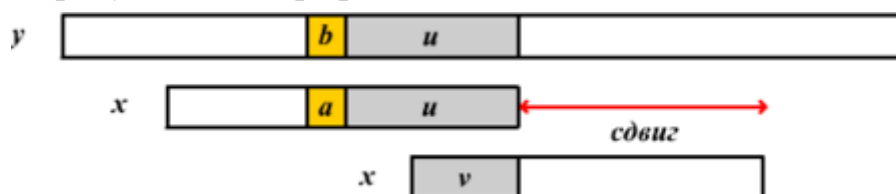
Правило сдвига хорошего суффикса

Если при сравнении текста и шаблона совпало один или больше символов, шаблон сдвигается в зависимости от того, какой суффикс совпал.

Если существуют такие подстроки равные U , что они полностью входят в X и идут справа от символов, отличных от $x[i]$, то сдвиг происходит к самой правой из них, отличной от U . Понятно, что таким образом мы не пропустим никакую строку, так как сдвиг происходит на следующую слева подстроку U от суффикса. После выравнивания шаблона по этой подстроке сравнение шаблона опять начнется с его последнего символа. На новом шаге алгоритма можно строку U , по которой был произведён сдвиг, не сравнивать с текстом — возможность для модификации и дальнейшего ускорения алгоритма.



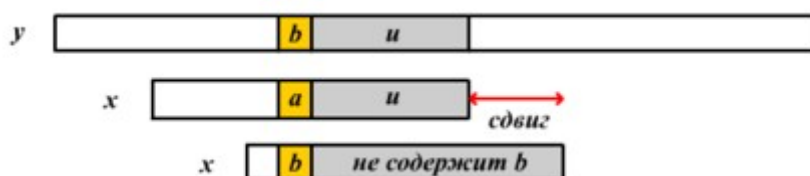
Если не существует таких подстрок, то смещение состоит в выравнивании самого длинного суффикса Y подстроки $y[i+j+1 \dots j+m-1]$ с соответствующим префиксом X . Из-за того, что мы не смогли найти такую подстроку, то, очевидно, что ни один суффикс шаблона X уже не будет лежать в подстроке $y[i+j+1 \dots j+m-1]$, поэтому единственный вариант, что в эту подстроку попадет префикс.



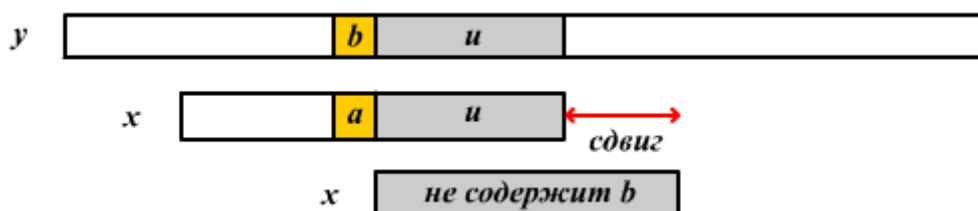
Правило сдвига плохого символа

В таблице плохих символов указывается последняя позиция в шаблоне (исключая последнюю букву) каждого из символов алфавита. Для всех символов, не вошедших в шаблон, пишем М. Предположим, что у нас не совпал символ S из текста на очередном шаге с символом из шаблона. Очевидно, что в таком случае мы можем сдвинуть шаблон до первого вхождения этого символа S в шаблоне, потому что совпадений других символов точно не может быть. Если в шаблоне такого символа нет, то можно сдвинуть весь шаблон полностью.

Если символ исходного текста $y[i+j]y[i+j]$ встречается в шаблоне X , то происходит его выравнивание с его самым правым появлением в подстроке $x[0...m-2]x[0...m-2]$.



Если $y[i+j]y[i+j]$ не встречается в шаблоне X , то ни одно вхождение X в Y не может включать в себя $y[i+j]y[i+j]$, и левый конец окна сравнения совмещен с символом непосредственно идущим после $y[i+j]y[i+j]$, то есть символ $y[i+j+1]y[i+j+1]$.



Обратите внимание, что сдвиг плохого символа может быть отрицательным, поэтому исходя из ранее приведенных свойств этих функций берется значение равное максимуму между сдвигом хорошего суффикса и сдвигом плохого символа.

Псевдокод

```

1  function boyer_moor(text, pattern):
2      n = длина(text)
3      m = длина(pattern)
4      if m == 0:
5          return 0
6      # Предварительная обработка правил
7      bad_char = таблица_плохих_символов(pattern)
8      good_suffix = таблица_хороших_суффиксов(pattern)
9      s = 0
10     while s <= n - m:
11         j = m - 1
12         while j >= 0 and pattern[j] == text[s + j]:
13             j -= 1
14         if j < 0:
15             return s # Найдено совпадение
16         else:
17             s += max(bad_char[text[s + j]] - (m - 1 - j), good_suffix[j])
18     return -1 # Совпадений нет

```

Пример:

Пусть нам дана строка $y = \text{GCATCGCAGAGAGTATACAGTACG}$ и образец $x = \text{GCAGAGAG}$.

Построим массивы `bmVcbmVc` и `bmGsbmGs` :

a	A	C	G	T
$bmBc[a]$	1	6	2	8

i	0	1	2	3	4	5	6	7
$x[i]$	G	C	A	G	A	G	A	G
$suff[i]$	1	0	0	2	0	4	0	8
$bmGs[i]$	7	7	7	2	7	4	7	1

Изображение	$(j, bmGs[y[j]])$	Описание
	(7, 1)	Сравниваем последние символы, они не равны, поэтому сдвигаемся на $bmGs[y[j]]$, где $y[j]$ — это не совпавший символ. В данном случае $y[j] = 7$, а $bmGs[7] = 1$.
	(8, 4)	Последние символы совпали. Предпоследние совпали. Третьи символы с конца различны, сдвигаемся на $bmGs[5] = 4$.
	(12, 7)	Последние символы совпали, сравниваем далее. Строчка найдена. Продолжаем работу и сдвигаемся на $bmGs[0] = 7$.
	(19, 4)	Последние символы совпали. Предпоследние совпали. Третьи символы с конца различны, сдвигаемся на $bmGs[5] = 4$.
	(23, 7)	Последние символы совпали, предпоследние различны. Алгоритм закончил работу.

В итоге, чтобы найти одно вхождение образца длиной $m=8$ в образце длиной $n=24$, нам понадобилось 17 сравнений символов.

3. Анализ алгоритма

Корректность:

Алгоритм гарантированно находит все вхождения подстроки благодаря правилам сдвига, которые не пропускают возможные совпадения.

Временная сложность:

- В лучшем случае: $O(n/m)$
- В худшем случае: $O(n*m)$
- На практике близок к $O(n)$ для естественных текстов.

Память: $O(m)$ для хранения таблиц.

Достоинства

Алгоритм Бойера-Мура на хороших данных очень быстр, а вероятность появления плохих данных крайне мала. Поэтому он оптимален в большинстве случаев, когда нет возможности провести предварительную обработку текста, в котором проводится поиск.

На больших алфавитах (относительно длины шаблона) алгоритм чрезвычайно быстрый и требует намного меньше памяти, чем алгоритм Ахо-Корасик)).

Позволяет добавить множество модификаций, таких как поиск подстроки, включающей *любой символ (?)* (но для реализации *множества символов (*)* не подходит, так как длина шаблона должна быть известна заранее).

Недостатки

Алгоритмы семейства Бойера-Мура не расширяются до приблизительного поиска, поиска любой строки из нескольких.

На больших алфавитах (например, Юникод) может занимать много памяти. В таких случаях либо обходятся хэш-таблицами, либо дробят алфавит, рассматривая, например, 4-байтовый символ как пару двухбайтовых.

На искусственно подобранных неудачных текстах скорость алгоритма Бойера-Мура серьезно снижается.

4. Реализация

```
def boyer_moore(text, pattern):  
    """  
    Реализация алгоритма Бойера-Мура для поиска подстроки.  
    Возвращает список позиций всех вхождений pattern в text.  
    """  
  
    def preprocess_bad_char(pattern):  
        """  
        Создает таблицу плохих символов (bad character rule).  
        Ключ - символ, значение - последняя позиция символа в шаблоне.  
        """  
        bad_char = {}  
        length = len(pattern)  
        for i in range(length):  
            bad_char[pattern[i]] = i  
        return bad_char  
  
    def preprocess_good_suffix(pattern):  
        """  
        Создает таблицу хороших суффиксов (good suffix rule).  
        """  
        length = len(pattern)  
        bmGs = [0] * length  
        suffix = [0] * (length + 1)  
  
        # Первый этап: вычисление suffix  
        suffix[length] = length  
        for i in range(length-1, -1, -1):  
            j = i  
            while j >= 0 and pattern[j] == pattern[length - 1 - i + j]:  
                j -= 1  
            suffix[i] = i - j  
  
        # Второй этап: заполнение bmGs  
        for i in range(length):  
            bmGs[i] = length  
        j = 0  
        for i in range(length-1, -1, -1):  
            if suffix[i] == i + 1:  
                while j < length - 1 - i:  
                    if bmGs[j] == length:  
                        bmGs[j] = length - 1 - i  
                    j += 1  
        for i in range(length-1):  
            bmGs[length - 1 - suffix[i]] = length - 1 - i  
  
        return bmGs
```



```

# Основная функция
n = len(text)
m = len(pattern)
if m == 0:
    return []

# Предварительная обработка
bad_char = preprocess_bad_char(pattern)
bmGs = preprocess_good_suffix(pattern)

result = []
s = 0 # Текущая позиция сравнения в тексте

while s <= n - m:
    j = m - 1
    # Сравниваем справа налево
    while j >= 0 and pattern[j] == text[s + j]:
        j -= 1

    if j < 0:
        # Найдено совпадение
        result.append(s)
        # Сдвигаемся на длину шаблона (аналог bmGs[0])
        s += bmGs[0]
    else:
        # Вычисляем сдвиг по правилу плохого символа
        bc_shift = j - bad_char.get(text[s + j], -1)
        # Вычисляем сдвиг по правилу хорошего суффикса
        gs_shift = bmGs[j]
        # Выбираем максимальный сдвиг
        s += max(bc_shift, gs_shift, 1)

return result

# Пример использования
text = "GCATCGAGAGAGAGTATACAGTACGCAGAGAG"
pattern = "GCAGAGAG"

positions = boyer_moore(text, pattern)
print("Найденные позиции:", positions) # Должно вывести [8, 19]

```

Данный код реализует алгоритм Бойера-Мура для эффективного поиска подстроки в строке. Алгоритм использует два основных правила для оптимизации поиска:

1. Правило плохого символа:

- Создается таблица `bad_char`, которая для каждого символа шаблона запоминает его последнюю позицию.
- При несовпадении символов это правило позволяет сдвинуть шаблон на максимально возможное расстояние.

2. Правило хорошего суффикса:

- Создается таблица `bmGs`, которая учитывает совпадения суффиксов шаблона.
- Это правило помогает определить оптимальный сдвиг, если часть шаблона уже совпала с текстом.

Основные шаги алгоритма:

1. Предварительная обработка:

- Формируются таблицы `bad_char` и `bmGs` для шаблона.

2. Поиск в тексте:

- Шаблон сравнивается с текстом справа налево.
- При несовпадении символов вычисляется сдвиг на основе таблиц `bad_char` и `bmGs`.
- При полном совпадении позиция добавляется в результат, и шаблон сдвигается для продолжения поиска.

Пример работы:

- Текст: «GCATCGAGAGAGAGTATACAGTACGCAGAGAG»
- Шаблон: «GCAGAGAG»
- Результат: [8,19] — позиции, где найдены вхождения шаблона.

5. Заключение

Алгоритм Бойера-Мура является одним из самых эффективных алгоритмов для поиска подстроки в тексте, особенно для больших объемов данных. Его основное преимущество заключается в использовании двух эвристик — правила плохого символа и правила хорошего суффикса, — которые позволяют значительно сократить количество сравнений.

Ключевые особенности:

- **Скорость:** В лучшем случае алгоритм работает за линейное время $O(n/m)$, где n — длина текста, а m — длина шаблона.
- **Эффективность:** Алгоритм особенно полезен для поиска в больших текстах или при многократных поисках одного шаблона.

•**Гибкость:** Может быть адаптирован для различных задач, включая поиск с учетом регистра или использование wildcard-символов.

Реализация, представленная в коде, демонстрирует классический подход к алгоритму Бойера-Мура и может быть использована как основа для более сложных модификаций. Для дальнейшего улучшения можно оптимизировать предварительную обработку таблиц или добавить поддержку дополнительных правил.

GITHUB: <https://github.com/sadPablik777/BoyMure>

Оригинальная статья авторов: (я был в шоке что смог найти ее)

<https://www.cs.utexas.edu/~moore/publications/fstrpos.pdf>

<https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/>

https://rosettacode.org/wiki/Boyer-Moore_string_search