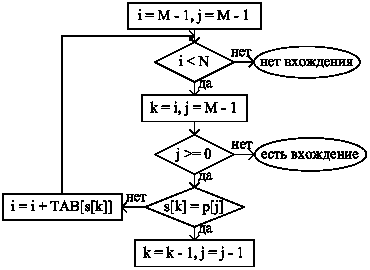
Алгоритмы поиска строки – это инструменты для нахождения подстроки в строке. Они нужны для решения многих задач, связанных с обработкой текстов, например:

1. Поиск ключевых слов в тексте – алгоритмы поиска строки могут быть использованы для поиска ключевых слов в тексте, например при обработке страниц для SEO (оптимизации веб-страниц) или при поиске информации в БД.
2. Анализ логов – данные алгоритмы могут быть использованы для анализа логов веб-серверов или приложений.
3. Фильтрация данных
4. Редактирование текста – например, для автозамены слов в тексте
5. Компиляция программ – данные алгоритмы поиска строки используются в процессе компиляции программного кода для поиска символов, которые нужно заменить на другие символы или для поиска ошибок в коде.

Алгоритмы поиска подстроки в строке:

1. **Алгоритм наивного поиска**

* Этот алгоритм проверяет каждый символ строки по очереди, чтобы убедиться, соответствует ли он первому символу подстроки.
* Если символы соответствуют, то алгоритм проверяет следующий символ подстроки и следующий символ строки и так далее.
* Если символы не соответствуют, то алгоритм переходит к следующему символу в строке и начинает сначала сравнение соответствующих символов.
* Алгоритм завершает работу, когда весь текст будет пройден или когда найдена подстрока.
* Он не требует никаких предварительных вычислений или дополнительной памяти, но может быть очень медленным при поиске в больших строках или при использовании длинных подстрок.
* 

1. **Алгоритм Кнута-Морриса-Пратта (КМП)**

* Этот алгоритм использует дополнительную информацию, которая была вычислена из самой подстроки, чтобы оптимизировать поиск.
* Алгоритм создает массив префиксов подстроки, который показывает, на какой позиции в подстроке начинается самый длинный суффикс, который является также ее префиксом.
* Затем алгоритм проходит по строке с помощью двух указателей - один указатель перемещается по строке, а другой перемещается по подстроке.
* Если символы совпадают, то алгоритм продолжает движение указателей вперед.
* Если символы не совпадают, то алгоритм использует информацию из массива префиксов, чтобы переместить указатель подстроки на позицию, где начинается самый длинный суффикс, который является также префиксом подстроки.
* Алгоритм продолжает движение указателей до тех пор, пока не найдет подстроку или не пройдет весь текст.
* Алгоритм требует предварительного вычисления массива префиксов, но имеет лучшую производительность, чем наивный алгоритм.
* 

1. **Алгоритм Бойера-Мура** - это один из наиболее эффективных алгоритмов поиска подстроки в строке. Он использует два шага для быстрого поиска подстроки:

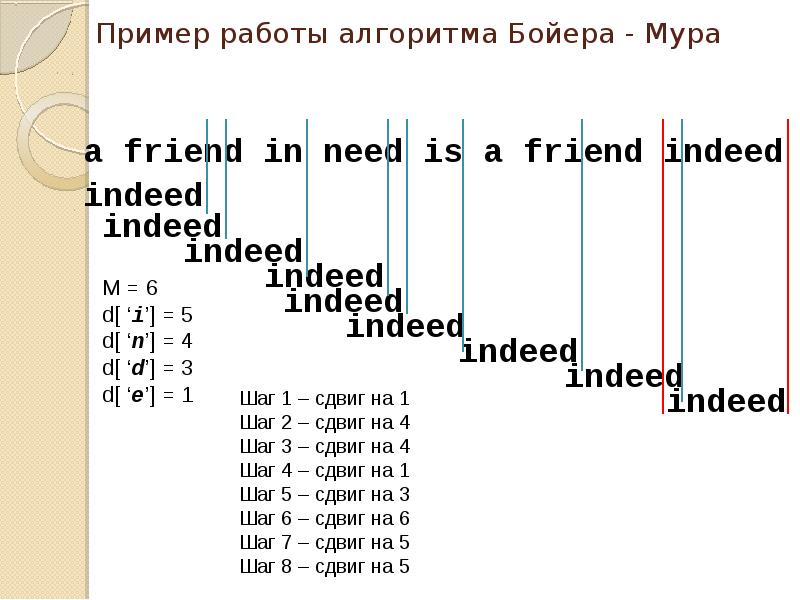
Предварительная обработка подстроки:

* Алгоритм начинает с конца подстроки и работает в обратном направлении.
* Для каждого символа в подстроке алгоритм создает таблицу сдвигов, которая показывает, на сколько символов нужно сдвинуть строку для того, чтобы сравнить символ в подстроке с символом в строке.
* Если символ в подстроке не встречается в оставшейся части подстроки, то алгоритм просто сдвигает всю подстроку на длину подстроки.
* Если символ в подстроке встречается в оставшейся части подстроки, то алгоритм сдвигает подстроку на минимальное количество символов, чтобы сравнить его с символом в строке.

Поиск подстроки:

* Алгоритм начинает поиск подстроки с конца подстроки.
* Он сравнивает последний символ подстроки с последним символом строки.
* Если символы совпадают, то алгоритм продолжает сравнение предыдущих символов в подстроке и строке, пока не достигнет начала подстроки или пока символы не перестанут совпадать.
* Если символы не совпадают, то алгоритм использует таблицу сдвигов, чтобы сдвинуть подстроку на определенное количество символов вправо.
* Алгоритм продолжает сравнение символов до тех пор, пока не найдет подстроку или не пройдет весь текст.
* Если алгоритм находит подстроку, то он возвращает позицию, где она найдена.

Алгоритм Бойера-Мура работает быстро в большинстве случаев и используется во многих приложениях, требующих быстрого поиска подстроки в большой строке.



1. Алгоритм Рабина-Карпа - это алгоритм поиска подстроки в строке, основанный на использовании хэш-функций. Рассмотрим алгоритм более подробно:

* Выбирается подходящая хэш-функция, которая преобразует строку в число. Хорошей хэш-функцией для данного алгоритма может служить, например, полиномиальная хэш-функция.
* Вычисляется хэш-значение подстроки, которую необходимо найти.
* Вычисляется хэш-значение каждой подстроки длины m в тексте, где m - длина искомой подстроки. При этом вычисление новых хэш-значений производится за O(1) при помощи предыдущего значения хэш-функции и добавления нового символа в конец подстроки и удаления символа из ее начала.
* Если хэш-значение текущей подстроки совпадает с хэш-значением искомой подстроки, то проверяются сами строки, чтобы исключить возможность ложного срабатывания хэш-функции. Если строки равны, то искомая подстрока найдена.
* В случае коллизии (когда разные подстроки дают одно и то же хэш-значение), необходимо использовать дополнительную проверку, сравнивая строки символ за символом.
* Алгоритм Рабина-Карпа позволяет выполнять поиск подстроки в строке за время O(n+m), где n - длина текста, а m - длина искомой подстроки.

Однако стоит учитывать, что при использовании хэш-функций возможны коллизии, которые могут замедлить алгоритм. Для уменьшения вероятности коллизий можно использовать несколько хэш-функций или дополнительную проверку при коллизии.

Пример реализации алгоритма Бойера-Мура на C++

#include <iostream>

#include <cstring>

using namespace std;

// Функция для вычисления максимального из двух целых чисел

int max(int a, int b) {

return (a > b) ? a : b;

}

// Функция для выполнения алгоритма Бойера-Мура

void boyer\_moore\_search(char \*text, char \*pattern) {

int n = strlen(text); // Длина текстовой строки

int m = strlen(pattern); // Длина образца

int i, j, k;

// Создание массива смещений

int shift[256];

for (i = 0; i < 256; i++) {

shift[i] = m;

}

for (i = 0; i < m - 1; i++) {

shift[pattern[i]] = m - i - 1;

}

// Поиск образца в текстовой строке

i = m - 1; // Индекс текущего символа в тексте

while (i < n) {

j = m - 1; // Индекс текущего символа в образце

k = i; // Индекс текущего символа в тексте

// Проверка совпадения символов с конца образца до его начала

while (j >= 0 && text[k] == pattern[j]) {

k--;

j--;

}

// Если образец найден, вывод его позиции в тексте

if (j == -1) {

cout << "Pattern found at position " << k + 1 << endl;

}

// Смещение индекса текущего символа в тексте на основе массива смещений

i += shift[text[i]];

}

}

int main() {

char text[] = "Lorem ipsum dolor sit amet, consectetur adipiscing elit.";

char pattern[] = "consectetur";

boyer\_moore\_search(text, pattern);

return 0;

}

В данном примере в текстовой строке « Lorem ipsum dolor sit amet, consectetur adipiscing elit.»

ищется образец «consectetur» с помощью алгоритма Бойера-Мура. Результатом работы будет вывод в консоль позиции найденного образца в тексте : «pattern found at positon 28”

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Preprocessing time** | **Matching time**[[1]](https://translated.turbopages.org/proxy_u/en-ru.ru.039de3b7-645b0f1f-3b17b4bd-74722d776562/https/en.wikipedia.org/wiki/String-searching_algorithm" \l "endnote_Asymptotic_times) | **Space** |
| **Naïve algorithm** | none | Θ(мн) | Нет |
| [**Rabin–Karp**](https://translated.turbopages.org/proxy_u/en-ru.ru.039de3b7-645b0f1f-3b17b4bd-74722d776562/https/en.wikipedia.org/wiki/Rabin%E2%80%93Karp_algorithm) | Θ(м) | Θ(n) in average, O(mn) at worst | О(1) |
| [**Knuth–Morris–Pratt**](https://translated.turbopages.org/proxy_u/en-ru.ru.039de3b7-645b0f1f-3b17b4bd-74722d776562/https/en.wikipedia.org/wiki/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm) | Θ(м) | Θ (п) | Θ(м) |
| [**Boyer–Moore**](https://translated.turbopages.org/proxy_u/en-ru.ru.039de3b7-645b0f1f-3b17b4bd-74722d776562/https/en.wikipedia.org/wiki/Boyer%E2%80%93Moore_string-search_algorithm) | Θ (м + к) | Ω(n/m) at best, O(mn) at worst | Θ(к) |

Sources: <https://habr.com/ru/articles/113266/>

https/en.wikipedia.org/wiki/String-searching\_algorithm

<https://ppt-online.org/768026>

https://myslide.ru/presentation/skachat-poisk-zadacha-poiska-lekciya-7