ВикипедиЯ

Поиск подстроки

Материал из Википедии — свободной энциклопедии

Поиск подстроки в строке — одна из простейших задач поиска информации. Применяется в виде встроенной функции в текстовых редакторах, СУБД, поисковых машинах, языках программирования и т. п.

В задачах поиска традиционно принято обозначать шаблон поиска как *needle* (с англ. — «иголка»), а строку, в которой ведётся поиск — как *haystack* (с англ. — «стог сена»). Также обозначим через Σ алфавит, на котором проводится поиск.

Содержание

Несостоятельность примитивного алгоритма

Если считать, что строки нумеруются с 1, простейший алгоритм (англ. *brute force algorithm, naïve algorithm*) выглядит так.

```
for i=0...|haystack|-|needle|
  for j=0...|needle|
    if haystack[i+j + 1]<>needle[j]
      then goto 1
  output("Найдено: ", i+1)
  1:
```

Простейший алгоритм поиска даже в *лучшем* случае проводит $|haystack|-|needle|+1$ сравнение; если же есть много частичных совпадений, скорость снижается до $O(|haystack|\cdot|needle|)$.

Показано, что примитивный алгоритм отрабатывает в среднем $2h$ сравнений[1].

Для чего нужно так много алгоритмов?

На сегодняшний день существует огромное разнообразие алгоритмов поиска подстроки. Программисту приходится выбирать подходящий в зависимости от таких факторов.

1. Нужна ли вообще оптимизация, или хватает примитивного алгоритма? Как правило, именно его реализуют стандартные библиотеки языков программирования.
2. «Враждебность» пользователя. Другими словами: будет ли пользователь намеренно задавать данные, на которых алгоритм будет медленно работать? Существуют очень простые алгоритмы, оценка которых $O(|haystack|\cdot|needle|)$ в худшем случае, но на «обычных» данных количество сравнений намного меньше $|haystack|$. Только в 1990-е годы были созданы алгоритмы, дающие сложность $O(|haystack|)$ в худшем случае и меньше $|haystack|$ в среднем.
3. Грамматика языка может быть недружественной к тем или иным эвристикам, которые ускоряют поиск «в среднем».
4. Архитектура процессора. Некоторые процессоры имеют автоинкрементные или SIMD-операции, которые позволяют быстро сравнить два участка ОЗУ (например, `rep cmpsd` на x86). На таких процессорах заманчиво применить алгоритм, который просто бы сравнивал *needle* с *haystack* — разумеется, не во всех позициях.
5. Размер алфавита. Многие алгоритмы (особенно основанные на сравнении с конца) имеют эвристики, связанные с несовпавшим символом. На больших алфавитах таблица символов будет занимать много памяти, на малых — соответствующая эвристика будет неэффективной.
6. Возможность проиндексировать *haystack*. Если таковая есть, поиск серьёзно ускорится.
7. Требуется ли одновременный поиск нескольких строк? Приблизительный поиск? Побочные свойства некоторых алгоритмов (Ахо-Корасик, двоичного алгоритма) позволяют такое.

Как правило, в текстовом редакторе достаточно взять самый простой эвристический алгоритм наподобие Бойера — Мура — Хорспула — даже очень медленный ПК справится с поиском за доли секунды. Если же объём текста измеряется гигабайтами, либо поиск запущен на сервере, который обрабатывает множество запросов — приходится выбирать наиболее удачный алгоритм из доступных. Например, программы определения плагиата осуществляют онлайн-проверку, используя алгоритмы поиска подстроки среди большого количества документов, хранящихся в собственной базе.

Алгоритмы

Для сокращения обозначим:

- $|\Sigma|=\sigma$ — размер алфавита.
- $|haystack|=H$ — длина строки, в которой ведётся поиск.
- $|needle|=n$ — длина шаблона поиска.

Вычислительная сложность определяется *до первого совпадения*. **Жирным шрифтом** выделены важнейшие с практической точки зрения алгоритмы.

Основанные на сравнении как «чёрном ящике»

Во всех этих алгоритмах сравнение строк является «чёрным ящиком». Это позволяет использовать стандартные функции сравнения участков памяти, зачастую оптимизированные на ассемблерном уровне под тот или иной процессор и не выдающие точки, в которой наступило несовпадение.

К этой категории относится и примитивный алгоритм поиска.

Название	Предв. обработка	Сложность		Примечания
		типичная	макс.	
Примитивный алгоритм	Нет	$2H$	$O(Hn)$	
Алгоритм Бойера — Мура — Хорспула	$O(n+\sigma)$	$\sim 2H / \sigma$[2]	$O(Hn)$	Упрощённый до предела алгоритм Бойера — Мура; использует только видоизменённую эвристику стоп-символа — за стоп-символ всегда берётся символ *haystack*, расположенный напротив последнего символа *needle*.
Алгоритм быстрого поиска Алгоритм Санди	$O(n+\sigma)$	$<H$	$O(Hn)$	Также использует исключительно эвристику стоп-символа — но за стоп-символ берётся символ *haystack*, идущий за последним символом *needle*.

Основанные на сравнении с начала

Это семейство алгоритмов страдает невысокой скоростью на «хороших» данных, что компенсируется отсутствием регрессии на «плохих».

Название	Предв. обработка	Сложность		Примечания
		типичная	макс.	
Алгоритм Рабина-Карпа	$O(n)$	$<H+n$	$O(Hn)$	Хеширование позволяет серьёзно снизить сложность в среднем
Автоматный алгоритм Алгоритм Ахо-Корасик	$O(n\sigma)$	$= H$		Строит конечный автомат, который распознаёт язык, состоящий из одной-единственной строки. После небольшой модификации позволяет за один проход по *haystack* найти одну строку из нескольких.
Алгоритм Кнута-Морриса-Пратта	$O(n)$	$\leq 2H$		Один из первых алгоритмов с линейной оценкой в худшем случае. Модификация алгоритма Ахо-Корасик, строящая автомат неявно на основе префикс-функции.
Алгоритм Апостолико-Крошмора	$O(n)$	$< H$	$\leq 1{,}5H$	
Алгоритм Shift-Or Bitap-алгоритм Двоичный алгоритм	$O(n+\sigma)$	$=H\cdot\mathrm{ceil}(n/w)$		Эффективен, если размер *needle* (в символах) не больше размера машинного слова (в битах, обозначен как w). Легко переделывается на приблизительный поиск, поиск нескольких строк.

Основанные на сравнении с конца

В этом семействе алгоритмов *needle* движется по *haystack* слева направо, но сравнение этих строк друг с другом проводится справа налево. Сравнение справа налево позволяет в случае несовпадения сдвинуть *needle* не на одну позицию, а на несколько.

Название	Предв. обработка	Сложность		Примечания
		типичная	макс.	
Алгоритм Бойера — Мура	$O(n+\sigma)$	$<H$	$O(Hn)$	Стандартный алгоритм поиска подстроки в строке. Считается наиболее эффективным алгоритмом общего назначения.[3]
Алгоритм Чжу-Такаоки	$O(n+\sigma^2)$	$<H$	$O(Hn)$	Алгоритм Бойера — Мура, оптимизированный под короткие алфавиты
Алгоритм Апостолико-Джанкарло	$O(n+\sigma)$	$<H$	$\leq 1{,}5H$	Одна из первых попыток получить $<H$ в типичном случае и $O(H)$ в худшем. Очень сложен в реализации.
Турбо-алгоритм Бойера — Мура	$O(n+\sigma)$	$<H$	$\leq 2H$	Один из наиболее эффективных алгоритмов, не дающих регрессии на «плохих» данных

Проводящие сравнение в необычном порядке

Название	Предв. обработка	Сложность		Примечания
		типичная	макс.	
Непримитивный алгоритм	const	$<H$	$O(Hn)$	Простой алгоритм, сравнивающий второй символ, затем начиная с третьего в режиме «чёрного ящика», и, наконец, первый. При $n[1] \neq n[2]$[4] и несовпадении на второй-третьей стадии — сдвиг на 2 вправо.
Алгоритм Райты Алгоритм Бойера — Мура — Хорспула — Райты	$O(n+\sigma)$	$<H$	$O(Hn)$	Эмпирический алгоритм, оптимизированный под английские тексты. Сравнивает последний символ, потом первый, потом средний, потом все остальные; при несовпадении — сдвиг по Хорспулу.

См. также

- Подстрока
- Алгоритмы на строках
- Сопоставление с образцом
- Алгоритмы: построение и анализ

Примечания

1. Brute force algorithm (http://www-igm.univ-mlv.fr/~lecroq/string/node3.html) (англ.)
2. Horspool algorithm (http://www-igm.univ-mlv.fr/~lecroq/string/node18.html#SECTION00180)
3. Boyer-Moore algorithm (http://www-igm.univ-mlv.fr/~lecroq/string/node14.html#SECTION00140)
4. Напомним, символы нумеруются с 1, как в Паскале.

Литература

- *Гасфилд Д.* Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология = Algorithms on String, Trees, and Sequences. Computer Science and Computational Biology / Пер. с англ. И. В. Романовского. — 2-е изд. — СПб.: Невский Диалект, 2003. — 654 с. — ISBN 5-7940-0103-8, 5-94157-321-9, 0-521-58519-8.
- *Смит Б.* Методы и алгоритмы вычислений на строках = Computing Patterns in Strings. — М.: Вильямс, 2006. — 496 с. — ISBN 5-8459-1081-1, 0-201-39839-7.
- *Окулов С. М.* Алгоритмы обработки строк. — М.: Бином, 2013. — 255 с. — ISBN 978-5-9963016-2-1.

Ссылки

- Большая подборка алгоритмов поиска подстроки (http://www-igm.univ-mlv.fr/~lecroq/string/index.html) (англ.)
