Поиск подстрок

Определения

- **Алфавит** – конечное множество символов.
- **Строка (слово)** – это последовательность символов из некоторого алфавита. *Длина строки* – количество символов в строке.
- X=x[1]x[2]...x[n] – строка длинной n, где x[i] – i-ый символ строки X, принадлежащий алфавиту.
- Строка, не содержащая ни одного

Определения

- Строка X называется **подстрокой** строки Y, если найдутся такие строки Z_1 и Z_2, что $Y=Z_1XZ_2$.
- Подстрока X называется **префиксом** строки Y, если есть такая подстрока Z, что Y=XZ.
- Подстрока X называется **суффиксом** строки Y, если есть такая подстрока Z, что Y=ZX.

Постановка задачи

- Есть образец и строка, надо определить индекс, начиная с которого образец содержится в строке. Если не содержится — вернуть индекс, который не может быть интерпретирован как позиция в строке (например, отрицательное число).
- При необходимости отслеживать каждое вхождение образца в текст имеет смысл завести дополнительную функцию, вызываемую при каждом обнаружении образца.

Пример

- Дана последовательность символов x[1]..x[n]. Определить, встречаются ли в ней идущие друг за другом символы "abcd".
  (Другими словами, требуется выяснить, есть ли в слове x[1]..x[n] подслово "abcd".)

Простой алгоритм

- Решение. Имеется примерно n (если быть точным, n-3) позиций, на которых может находиться искомая подстрока в исходной строке.
  Для каждой из позиций можно проверить, действительно ли с нее начинается подстрока , сравнив четыре символа.
  Но обычно слово x[1]..x[n] просматривается слева направо, до появления буквы 'a'. Как только она появилась, ищем за ней букву 'b', затем 'c', и, наконец, 'd'. Если ожидания оправдываются, то слово "abcd" обнаружено. Если же какая-то из нужных букв не появляется, начинаем поиск с новой позиции.

Применение простого алгоритма

$i \rightarrow i \rightarrow i \rightarrow i \rightarrow i \rightarrow i \rightarrow i \rightarrow i$

Строка	A	B	C	A	B	C	A	A	B	C	A	B	D
Подстрока	**A**	**B**	**C**	**A**	**B**	**D**							
		A	B	C	A	B	D						
			A	B	C	A	B	D					
				A	**B**	**C**	**A**	**B**	D				
					A	B	C	A	B	D			
						A	B	C	A	B	D		
							A	**B**	C	A	B	D	
								A	**B**	**C**	**A**	**B**	**D**

Алгоритм Рабина-Карпа

Пусть алфавит D={0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, то есть каждый символ в алфавите есть d–ичная цифра, где d=│D│.

Пример

Образец имеет вид W = 3 1 4 1 5

Вычисляем значения чисел из окна длины |W|=5 по mod q, q — простое число.

Использование хеш-функции

T	=	2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1
Значения по mod 13 =						8	9	3	11	0	1	7	8	4	5	10	11	7	9	11

23590(mod 13)=8, 35902(mod 13)=9, 59023(mod 13)=9, …

k1=31415 (mod 13) ≡7– вхождение образца,

k2=67399(mod 13) ≡ 7– холостое срабатывание.

$$\sum_{i=1}^{m} p[i]x^{m-i} \bmod q$$

Хеш функция

Hash(p[1..m] = σ [illegible] 1 [illegible] − [illegible] [illegible],

- Ключ к производительности алгоритма Рабина — Карпа - низкая вероятность коллизий и эффективное вычисление значения хеш последовательных подстрок текста.
- Рабин и Карп предложили использовать *пол хеш*.

$$\sum_{i=1}^{m} p[i]x^{m-i} \bmod q$$

Алгоритм Рабина-Карпа

- Для ускорения модульной арифметики q выбирают равным степени двойки минус один (так называемые простые числа Мерсенна): для 32-х битовых машин лучше всего подходит q=2^{31}-1, для 64-х битовых — q=2^{61}-1

Алгоритм Рабина-Карпа

Для быстрого вычисления p используют схему Горнера:

P[1..m]- образец, p – число, которое является десятичной записью образца.

p=P[m]+10(P[m-1]+10(P[m-2] + … + 10(P[2] + 10p[1])…))

Алгоритм Рабина-Карпа

T[1..n]- текст, t_s – число, которое является десятичной записью T[s+1..s+m].

Если вычислено t_s , t_{s+1} можно вычислить за O(1)

$t_{s+1} = 10(t_s - 10^{m-1} T[s+1]) + T[s+m+1]$

Алгоритм Рабина-Карпа

- n=length[T]
- m=length[P]
- h= d^{m-1} mod q
- p=0
- t_0 = 0
- For i=1 to m
- {p= (d p +P[i]) mod q
- t_0 = (d t_0 +T[i]) mod q
- }

Алгоритм Рабина-Карпа (продолжение)

- For s=0 to n-m
- { if p== t_s
- if P[1..m]== T[s+1..s+m] print образец входит со сдвигом s;
- If s<n-m t_{s+1} = d(t_s -h T[s+1])+T[s+m+1] mod q
- }

Временная сложность алгоритма Рабина-Карпа

$$O(n)+O(mv),$$

v – количество вхождений образца в текст

Поиск подстрок с помощью конечных автоматов(abcd)

- при чтении слова x слева направо мы в каждый момент находимся в одном из следующих состояний:
- "начальное" (0),
  "сразу после a" (1),
- "сразу после ab" (2),
- "сразу после abc" (3)
  и "сразу после abcd" (4).

Конечные автоматы

- Читая очередную букву, мы переходим в следующее состояние по правилу:
- <Текущее состояние> <Очередная буква> <Новое состояние >

Алгоритм

- состояние буква состояние

0	a	1
0	кроме a	0
1	b	2
1	a	1
1	кроме a,b	0
2	c	3
2	a	1
2	кроме a,c	0
3	d	4
3	a	1
3	кроме a,d	0

- Состояние 4 - конечное

Фрагмент алгоритма1

- i=1; state=0;
  {i - первая непрочитанная буква, state - состояние}
  while (i <> n+1) and (state <> 4) {
  if state ==0 {
  if x[i] == ‘a’ {
  state= 1;
  }

Фрагмент алгоритма2

- else {
  state= 0;
  }
  } else if state = 1 {
  if x[i] == ‘b’ {
  state= 2;
  } else if x[i] == ‘a’ {
  state= 1;
  }else

Фрагмент алгоритма3

- {
  state= 0;
  }
  }else if state == 2 {
  if x[i] == ‘c’ {
  state= 3;
  } else if x[i] == ‘a’ {
  state= 1;
  } else {
  state= 0;}

Фрагмент алгоритма4

- }else if state == 3 {
  if x[i] == ‘d’ {
  state= 4;
  } else if x[i] == ‘a’ {
  state= 1;
  } else {
  state= 0;
  }
  }
  }
  answer = (state = 4);

Усовершенствованный алгоритм

Написать программу, которая ищет произвольный образец в произвольном слове. Это можно делать в два этапа:

1. сначала по образцу строится таблица переходов конечного автомата
2. читается входное слово и состояние преобразуется в соответствии с этой таблицей

Алгоритм Кнута - Морриса – Пратта (КМП)

- Работает за время O(m+n), где m – длина образца, n – длина текста
- Для произвольного слова X рассмотрим все его начала (префиксы), одновременно являющиеся его концами (суффиксами), и выберем из них самое длинное
- Примеры: l(aba)=a, l(abab)=ab, l(ababa)=aba, l(abc) = пустое слово.

КМП

- Длина наиболее длинного префикса, являющегося одновременно суффиксом есть префикс-функция от строки.
- Префикс –функция заданного образца несет информацию о том, где в образце повторно встречаются различные префиксы образца. Использование этой информации позволяет избежать проверки заведомо недопустимых сдвигов.

π-функция

- Алгоритм вычисления
- *Символы строк нумеруются с 1.*
- Пусть $\pi(S,i) = k$. Попробуем вычислить префикс-функцию для $i + 1$.
- Если $S[i + 1] = S[k + 1]$, то, естественно, $\pi(S,i + 1) = k + 1$. Если нет — пробуем меньшие суффиксы. Очевидно, что также будет суффиксом строки, а для любого строка суффиксом не будет. Таким образом, получается алгоритм:

π-функция

- При $S[i + 1] = S[k + 1]$ — положить $\pi(S,i + 1) = k + 1$.
- Иначе при $k = 0$ — положить $\pi(S,i + 1) = 0$.
- Иначе — установить $k := \pi(S,k)$, GOTO 1.

Пример

- Для строки 'abcdabscabcdabia' вычисление будет таким:
- 'a'!='b' => π=0;(длина строки 2; строка ab)
- 'a'!='c' => π=0; (длина строки 3; строка abc)
- 'a'!='d' => π=0;(длина строки 4; строка abcd)
- 'a'=='a' => π=π+1=1; (длина строки 5; строка abcda)
- 'b'=='b' => π=π+1=2;(длина строки 6; строка abcdab)
- 'c'!='s' => π=0; (длина строки 6; строка abcdabs)
- 'a'!='c' => π=0; (длина строки 7; строка abcdabsc)

Пример

'a'=='a' => π=π+1=1; (длина строки 8; строка abcdabsca)

'b'=='b' => π=π+1=2; (длина строки 9; строка abcdabscab)

'c'=='c' => π=π+1=3; (длина строки 9; строка abcdabscabc)

'd'=='d' => π=π+1=4;(длина строки 10; строка abcdabscabcd)

'a'=='a' => π=π+1=5; (длина строки 10; строка abcdabscabcda)

'b'=='b' => π=π+1=6; (длина строки 11; строка abcdabscabcdab)

‘s'!='i' => π=0; (длина строки 12; строка abcdabscabcdabi)

'a'=='a' => π=π+1=1; (длина строки 13; строка abcdabscabcdabia)

Пример реализации

Пример.
(Символы, подвергшиеся сравнению, подчеркнуты.)

```
Л И Т Е Р Н Ы Й   Т И П   У   Л И Т Е Р Ы   А ← T
Л И Т Е Р Ы                                   ← W
          Л И Т Е Р Ы
            Л И Т Е Р Ы
                . . .
                          Л И Т Е Р Ы
```

Алгоритм КМП-поиска

- После частичного совпадения начальной части образца W с соответствующими символами строки T фактически известна пройденная часть строки и можно «вычислить» некоторые сведения (на основе самого образа W), с помощью которых далее быстро продвинемся по тексту.

Алгоритм КМП

- Идея КМП-поиска – при каждом несовпадении двух символов текста и образца образец сдвигается на все пройденное расстояние, так как меньшие сдвиги не могут привести к полному совпадению

Z- функция

- Пусть ищется строка S1 в строке S2. Построим строку S= S1$S2, где $ — символ, не встречающийся ни в S1, ни в S2. Далее вычислим значения префикс-функции от строки S и всех её префиксов. Теперь, если префикс-функция от префикса строки S длины i равна n, где n—длина S1, и i>n, то в строке S2есть вхождение S1, начиная с позиции i-2n.

Алгоритм поиска строки Бойера —Мура

Был разработан Робертом Бойером и и Джеем Муром в 1977г.

Считается наиболее быстрым среди алгоритмов общего назначения, предназначенных для поиска подстроки в строке.

Общая оценка вычислительной сложности алгоритма О (длтекста+длобразца+мощню алф)

Описание алгоритма.

Алгоритм основан на трёх идеях

1. **Сканирование слева направо, сравнение справа налево.**
2. **Эвристика стоп-символа.**
3. **Эвристика совпавшего суффикса.**

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 символами строки, значит, подстрока найдена, и поиск окончен.

Сканирование слева направо, сравнение справа налево

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

Количество символов, на которые выполняется сдвиг, вычисляется по двум эвристикам

Эвристика стоп-символа.

Пример: поиск слова «колокол».

Пусть первая же буква не совпала — «к» (назовём эту букву *стоп-символом*).

Можно сдвинуть шаблон вправо до *последней* буквы «к».

- Строка: * * * * * * * к * * * * *
- Шаблон: к о л о к о л
- Следующий шаг: к о л о к о л

Эвристика стоп-символа.

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

- Строка: * * * * * * а л * * * * * * * *
- Шаблон: к о л о к о л
- Следующий шаг: к о л о к о л

В данном случае стоп-символ — «а»,

Эвристика стоп-символа.

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

- Строка: * * * * к к о л * * * * *
- Шаблон: к о л о к о л
- Следующий шаг: к о л о к о л ?????

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

-

Эвристика совпавшего суффикса

Если при сравнении строки и шаблона совпало один или больше символов, шаблон сдвигается в зависимости от совпавшего суффикса

- Строка: * * * т о к о л * * * * *
- Шаблон: к о л о к о л
- Следующий шаг: к о л о к о л

Алгоритм Бойера —Мура

Обе эвристики требуют предварительных вычислений.

По шаблону поиска заполняются две таблицы.

Таблица стоп-символов по размеру соответствует алфавиту (для алфавита из 256 символов, её длина 256);

Таблица суффиксов соответствует шаблону.

Таблица стоп-символов

В таблице стоп-символов указывается последняя позиция в образце (**исключая последнюю букву**) каждого из символов алфавита. Для символов, не вошедших в образец, пишем 0 (для нумерации с 0 — соответственно, −1).

Таблица стоп-символов

образец =«abcdadcd»

- Символ a b c d [остальные]
- Последняя позиция 5 2 7 6 0

Обратите внимание, для стоп-символа «d» последняя позиция будет 6, а не 8 — последняя буква не учитывается

При несовпадении в позиции i по стоп-символу c, сдвиг будет i-StopTable[c].

Таблица суффиксов

Для каждого возможного суффикса *S* шаблона указывается наименьшая величина, на которую нужно сдвинуть вправо шаблон, чтобы он снова совпал с *S*. Если такой сдвиг невозможен, ставится длина шаблона(в обеих системах нумерации).

Таблица суффиксов

Например, для «abcdadcd»

Суффикс

[пустой] d cd dcd ... abcdadcd

Сдвиг 1 2 4 8 ... 8

Иллюстрация

было ? ?d ?cd ?dcd ... abcdadcd

стало abcdadcd abcdadcd abcdadcd abcdadcd abcdadcd

Таблица суффиксов

Если шаблон начинается и заканчивается одной и той же комбинацией букв, |*шаблон*| вообще не появится в таблице. Например, для шаблон = «колокол» для всех суффиксов (кроме пустого) сдвиг будет равен 4.

Быстрый алгоритм вычисления таблицы суффиксов

Использует префикс-функцию строки

- m = length(suff)
- pi[] = префикс-функция(suff)
- pi1[] = префикс-функция(обращение(suff))
- for j=0..m
- suffshift[j] = m - pi[m]
- for i=1..m
- j = m - pi1[i]
- suffshift[j] = min(suffshift[j], i - pi1[i])

Быстрый алгоритм вычисления таблицы суффиксов

suffshift[0] соответствует всей совпавшей строке;

suffshift[m] — пустому суффиксу.

Так как префикс-функция вычисляется за $O(|шаблон|)$ операций, вычислительная сложность этого шага также равняется $O(|шаблон|)$

Пример работы алгоритма БМ

Искомый шаблон — «abbad».

Таблица стоп-символов:

- Символ a b [остальные]
- Позиция 4 3 0

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

Пример работы алгоритма БМ

Накладываем образец на строку.

- abeccaabadbabbad
- abbad

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

Пример работы алгоритма БМ

abeccaabadbabbad

abbad

Символы 3—5 совпали, а второй — нет. Эвристика стоп-символа для «a» не работает (2-4=-2).

Часть символов совпала, используем эвристику совпавшего суффикса. Шаблон сдвигается на пять позиций!

Пример работы алгоритма БМ

abeccaabadbabbad

abbad

Совпадения суффикса нет. По таблице стоп-символов сдвигаем образец на 1 позицию и получаем искомое вхождение образца:

abeccaabadbabbad

abbad

Алгоритм Бойера-Мура

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

На коротких текстах выигрыш не оправдывает предварительных вычислений

Алгоритм Бойера-Мура

Недостатки

не расширяются до приблизительного поиска, поиска любой строки из нескольких.

Не рекомендуют использовать алгоритм, если текст изменяется редко.

На больших алфавитах таблица стоп-символов может занимать много памяти. В таких используют спец. методы хранения таблиц

Алгоритм Бойера-Мура

На искусственно подобранных «неудачных» текстах (например, шаблон=«колоколоколоколоколокол») скорость алгоритма Бойера-Мура серьёзно снижается