Поиск подстрок

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 искомое подслово в исходном слове.
  Для каждой из позиций можно проверить, действительно ли с нее начинается подслово, 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.

Конечные автоматы

- 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). Читая очередную букву, мы переходим в
  следующее состояние по правилу

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 программы-1

- 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 программы-2

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

Фрагмент программы-3

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

Фрагмент программы-4

- } 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

Надо 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 π(*S*,*i*) = *k*. Попробуем вычислить префикс-функцию для *i* + 1.
- Если *S*[*i* + 1] = *S*[*k* + 1], то, естественно, π(*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

Алгоритм с использованием префикс-функции

- 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.

Реализация алгоритма

- Реализовать самостоятельно