**Киевский национальный университет имени Тараса Шевченка**

**Факультет компьютерных наук и кибернетики**

**Алгоритмы и сложность**

**Лабораторный проект № 8**

**Алгоритмы поиска образца в строке**

**Отчет**

**Подготовил:**

студент группы К-28

Шатохин Максим Сергеевич

**Киев-2018**

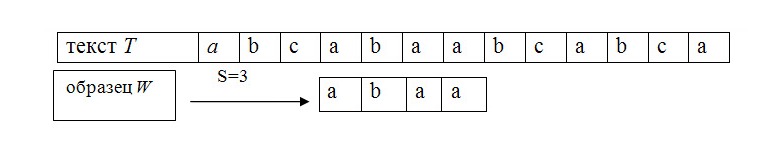
1. **Постановка задачи поиска в строке**

Часто приходится сталкиваться со специфическим поиском, так называемым поиском строки (поиском в строке). Пусть есть некоторый текст Т и слово (или образ) W. Необходимо найти первое вхождение этого слова в указанном тексте. Это действие типично для любых систем обработки текстов. (Элементы массивов Т и W – символы некоторого конечного алфавита – например, {0, 1}, или {a, …, z}, или {а, …, я}.)

Наиболее типичным приложением такой задачи является документальный поиск: задан фонд документов, состоящих из последовательности библиографических ссылок, каждая ссылка сопровождается «дескриптором», указывающим тему соответствующей ссылки. Надо найти некоторые ключевые слова, встречающиеся среди дескрипторов. Мог бы иметь место, например, запрос «Программирование» и «Java». Такой запрос можно трактовать следующим образом: существуют ли статьи, обладающие дескрипторами «Программирование» и «Java».

Поиск строки формально определяется следующим образом. Пусть задан массив Т из N элементов и массив W из M элементов, причем 0<M≤N. Поиск строки обнаруживает первое вхождение W в Т, результатом будем считать индекс i, указывающий на первое с начала строки (с начала массива Т) совпадение с образом (словом).

Пример. Требуется найти все вхождения образца W = abaa в текст T=abcabaabcabca.



Образец входит в текст только один раз, со сдвигом S=3, индекс i=4.

1. **Алгоритм прямого поиска**
2. ***Идея алгоритма:***

1. I=1,

2. сравнить I-й символ массива T с первым символом массива W,

3. совпадение → сравнить вторые символы и так далее,

4. несовпадение → I:=I+1 и переход на пункт 2,

1. ***Условие окончания алгоритма:***

1. подряд М сравнений удачны,

2. I+M>N, то есть слово не найдено.

1. ***Сложность алгоритма:***

Худший случай. Пусть массив T→{AAA….AAAB}, длина │T│=N, образец W→{A….AB}, длина │W│=M. Очевидно, что для обнаружения совпадения в конце строки потребуется произвести порядка N\*M сравнений, то есть O(N\*M).

1. ***Недостатки алгоритма:***

1. высокая сложность — O(N\*M), в худшем случае – Θ((N-M+1)\*M);

2. после несовпадения просмотр всегда начинается с первого символа образца и поэтому может включать символы T, которые ранее уже просматривались (если строка читается из вторичной памяти, то такие возвраты занимают много времени);

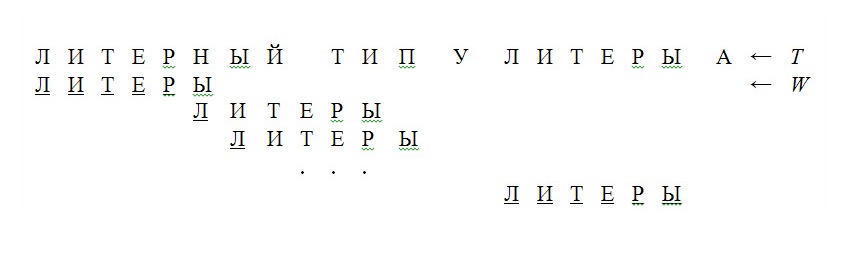
3. информация о тексте T, получаемая при проверке данного сдвига S, никак не используется при проверке последующих сдвигов.

1. **Алгоритм Д. Кнута, Д. Мориса и В. Пратта (КМП-поиск)**

Алгоритм КМП-поиска фактически требует только порядка N сравнений даже в самом плохом случае.

1. ***Пример.***

(Символы, подвергшиеся сравнению, подчеркнуты.)



После частичного совпадения начальной части образа W с соответствующими символами строки Т мы фактически знаем пройденную часть строки и может «вычислить» некоторые сведения (на основе самого образа W), с помощью которых потом быстро продвинемся по тексту.

1. ***Идея КМП-поиска***

при каждом несовпадении двух символов текста и образа образ сдвигается на все пройденное расстояние, так как меньшие сдвиги не могут привести к полному совпадению.

1. ***Особенности КМП-поиска:***

1. требуется порядка (N+M) сравнений символов для получения результата;

2. схема КМП-поиска дает подлинный выигрыш только тогда, когда неудаче предшествовало некоторое число совпадений. Лишь в этом случае образ сдвигается более чем на единицу. К несчастью совпадения встречаются значительно реже чем несовпадения. Поэтому выигрыш от КМП-поиска в большинстве случаев текстов весьма незначителен.

1. **Алгоритм Р. Боуера и Д. Мура (БМ-поиск)**

На практике алгоритм БМ-поиска наиболее эффективен, если образец W длинный, а мощность алфавита достаточно велика.

1. ***Идея БМ-поиска***

сравнение символов начинается с конца образца, а не с начала, то есть сравнение отдельных символов происходит справа налево. Затем с помощью некоторой эвристической процедуры вычисляется величина сдвига вправо s. И снова производится сравнение символов, начиная с конца образца.

Этот метод не только улучшает обработку самого плохого случая, но и даёт выигрыш в промежуточных ситуациях.

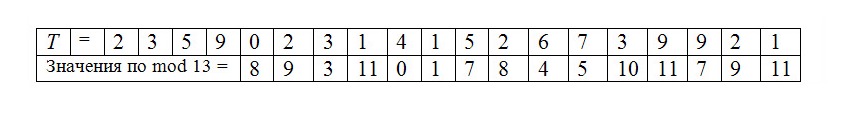
Почти всегда, кроме специально построенных примеров, БМ-поиск требует значительно меньше N сравнений. В самых же благоприятных обстоятельствах, когда последний символ образца всегда попадает на несовпадающий символ текста, число сравнений равно (N / M), в худшем же случае – О((N-M+1)\*M+ p), где p – мощность алфавита.

1. **Алгоритм Рабина-Карпа (РК-поиск)**

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. Пусть образец имеет вид W = 3 1 4 1 5

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



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

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

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

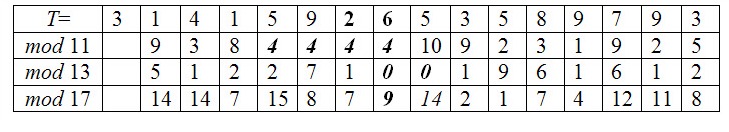
Из равенства ki= kj (mod q) не следует, что ki= kj (например, 31415=67399(mod 13), но это не значит, что 31415=67399). Если ki= kj (mod q), то ещё надо проверить, совпадают ли строки W[1…m] и T[s+1…s+m] на самом деле.

Если простое число q достаточно велико, то дополнительные затраты на анализ холостых срабатываний будут невелики.

В худшем случае время работы алгоритма РК — Θ((N-M+1)\*M), в среднем же он работает достаточно быстро – за время О(N+M).

Пример: Сколько холостых срабатываний k сделает алгоритм РК, если

q= 11, 13, 17. Пусть W={2 6}



26 mod 11=4 → k =3 холостых срабатывания,

26 mod 13=0 → k =1 холостое срабатывание,

26 mod 17=9 → k =0 холостых срабатываний.

Очевидно, что количество холостых срабатываний k является функцией от величины простого числа q (если функция обработки образца mod q) и, в общем случае, от вида функции для обработки образца W и текста Т.

1. **Алгоритм Бойера - Мура – Хорспула**

Алгоритм Бойера - Мура - Хорспула — алгоритм поиска подстроки в строке, упрощённый вариант алгоритма Бойера — Мура.К тому же, требующая многих предварительных вычислений эвристика совпавшего суффикса опускается.

Описание алгоритма

Алгоритм является модификацией алгоритма Бойера — Мура. Идея алгоритма такова.

1. Сканирование слева направо, сравнение в режиме «чёрного ящика». Как и в примитивном алгоритме, совмещается начало текста и шаблона, проводится сравнение обычной процедурой «сравнить участки памяти». Если все символы шаблона совпали с наложенными символами строки, значит, подстрока найдена, и поиск окончен.

Если же какой-то символ шаблона не совпадает с соответствующим символом строки, шаблон сдвигается на несколько символов вправо. Эти «несколько» выбираются в соответствии с такой эвристикой.

2. Изменённая эвристика стоп-символа. Берём символ текста, оказавшийся над последним символом шаблона (независимо от того, где случилось несовпадение!). На рисунке это «b».

↓ стоп-символ

Текст a b a d b \* \* \* \*

Шаблон a b b a d

Следующая проверка a b b a d

Сдвигаем шаблон так, чтобы под стоп-символом оказалась буква «b» шаблона. Это реализуется с помощью таблицы смещений: для каждого символа алфавита храним максимально возможный сдвиг, не пропускающий стоп-символ. То есть (при нумерации строк с 1): shift(c) = |needle|−lastpos(c, needle[1..|needle|−1]), где lastpos — последнее вхождение символа в строку, needle[a..b] — операция взятия подстроки.

Для шаблона «abbad» таблица имеет следующий вид.

Символ a b (все остальные)

Смещение 1 2 5

Для символов, не вошедших в шаблон, величина смещения устанавливается равной длине шаблона — 5. Последний символ шаблона при вычислении таблицы смещений не рассматривается (чревато зацикливанием).

Таблицу удобнее рассчитывать, проходя по всем символам шаблона: