Задачи поиска в структурах данных

- **Поиск - нахождение какой-либо конкретной информации в большом объеме ранее собранных данных.**
- **Данные делятся на записи, и каждая запись имеет хотя бы один ключ. Ключ используется для того, чтобы отличить одну запись от другой.**
- **Целью поиска является нахождение всех записей подходящих к заданному ключу поиска.**

Задачи поиска в структурах данных

- **Кроме поиска совпадению аргумента поиска с ключом записи, существует поиск по близости аргумента и ключа и поиск по интервалу, означающий попадание ключа в заданный двумя аргументами (границами) интервал.**
- **Логически сложные условия поиска могут быть конъюнктивными (обязательно выполнение в искомых записях всех заданных элементарны условий), дизъюнктивными (достаточно выполнения одного из них) и смешанной природы.**

Задачи поиска в структурах данных

•

- a: [0..N -1] of Item;
- **Item** **описывает запись с некоторым полем, играющим роль ключа.**
- **Задача заключается в поиске элемента, ключ которого равен заданному аргументу поиска x**

- **Полученный в результате индекс i, удовлетворяющий условию a[i].key = x, обеспечивает доступ к другим полям обнаруженного элемента.**

- **Так как мы рассматриваем, прежде всего, сам процесс поиска, то мы будем считать, что тип Item включает только ключ key**

Линейный поиск

- **Если нет никакой дополнительной информации о разыскиваемых данных, то очевидный подход - простой последовательный просмотр массива с увеличением шаг за шагом той его части, где желаемого элемента не обнаружено.**

- **Такой метод называется линейным поиском**

- Алгоритм 1. **a**: [0..N -1] **of Item**;

```
i← 0;
while (i<N) and (a[i]<>x) do
  i ← i+1;
```

Временная сложность O(n)

Линейный поиск

- Алгоритм 2.(алгоритм линейного поиска с барьером)
- a: [0..N] **of** **<примитивный тип>**

```
a[N] ← x; i← 0;
while a[i]<>x do
  i:=i+1;
```

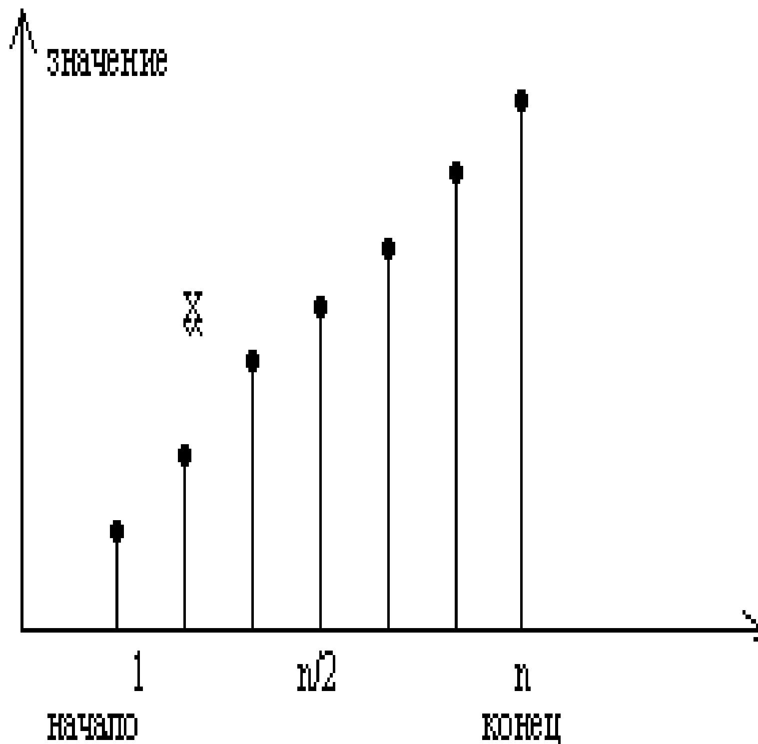
Поиск делением пополам (двоичный поиск)

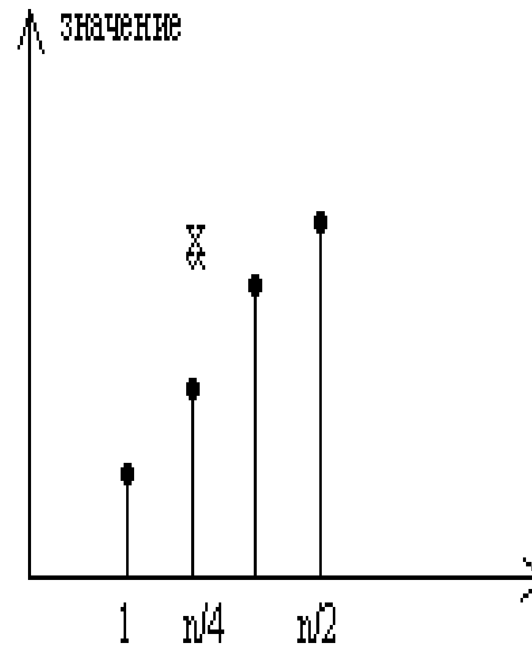
- массив А упорядочен, т. е. удовлетворяет условию

$$a_{k-1} \le a_k, \qquad 1 \le k < N$$

Поиск делением пополам (двоичный поиск)

•





Поиск делением пополам (двоичный поиск)

```
• L ← 0; R← N-1; Found← false;
while (L<=R) and not Found do
        m← (L+R) div 2;
        if a[m]=x then  Found← true
                   else
                     if a[m]<x then L← m+1
                                 else R← m-1
```

Поиск делением пополам (двоичный поиск)

- Максимальное число сравнений для этого алгоритма равно $\log_2 n$
- Временная сложность $O(\log_2 n)$

Прямой поиск строки

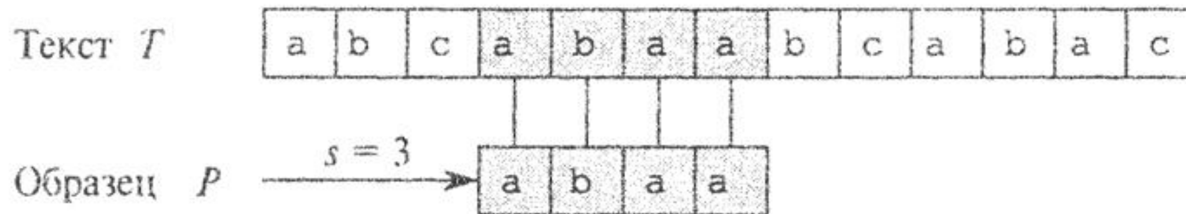
- Пусть задан массив t из N элементов и массив p из M элементов, причем 0 < M =< N.

t: **array**[0..N-1] of **char**;

p: **array[**0..M-1] of **char**;

Поиск строки обнаруживает первое вхождение **p** в **t**.

Прямой поиск строки

- 

Алгоритм BruteForceMatch(T,P):

Input: строка T (текст) из n символов и P (шаблон) из m символов.

Output: индекс начала подстроки, соответствующей P, в строке T или признак того, что P не является подстрокой T.

```
for i ← 0 to n — m { для каждого подходящего индекса в T } do
    j ← 0
    while (j < m and T[i + j] = P[j]) do
        j ← j + 1
    if j = m then
        return i
return "There is no substring of T matching P"
```

Прямой поиск строки

- Временная сложность
- $T(n) = O((n-m+1)m)$
- $T(n) = O(nm)$
- demo

Алгоритм Кнута, Мориса и Пратта (1977г)

Основным отличием КМП-алгоритма от алгоритма прямого поиска является осуществления сдвига слова не на один символ на каждом шаге алгоритма, а на некоторое переменное количество символов.

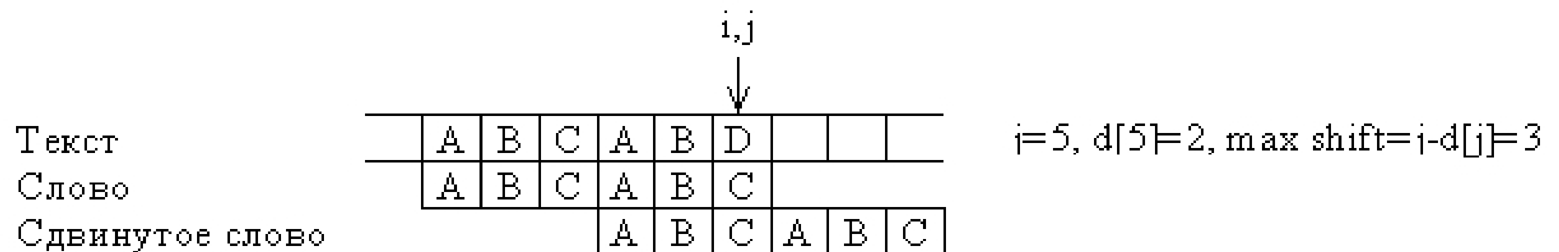
Таким образом, перед тем как осуществлять очередной сдвиг, необходимо определить величину сдвига.

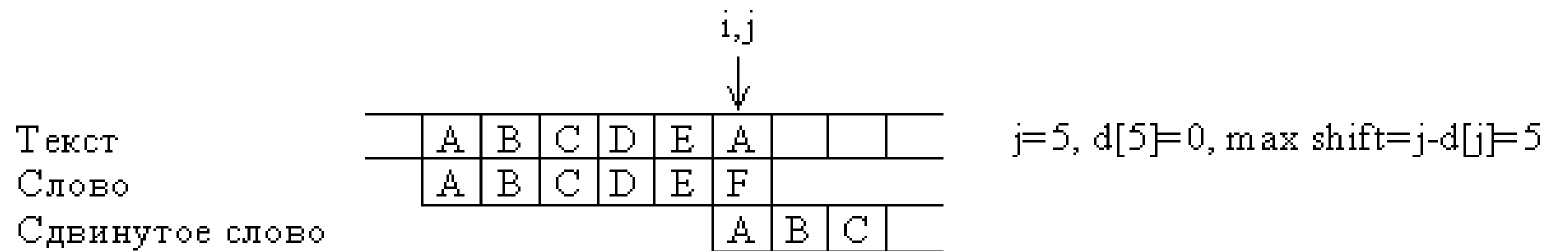
Для повышения эффективности алгоритма необходимо, чтобы сдвиг на каждом шаге был бы как можно большим.

Алгоритм Кнута, Мориса и Пратта

Если j определяет позицию в слове-образце, содержащую первый несовпадающий символ то величина сдвига определяется как $j\text{-}d_j$.
Значения D – таблица сдвигов определяется как размер самой длинной последовательности символов слова, непосредственно предшествующих позиции j (суффикс), которая полностью совпадает с началом слова (префикс).
D зависит только от слова и не зависит от текста. Для каждого j будет своя величина D, которую обозначим d_j.
Перед поиском осуществляется формирование D

Алгоритм Кнута, Мориса и Пратта





Алгоритм Кнута, Мориса и Пратта

ABCABCABAABCABD

ABCABD

ABCABD

ABCABD

ABCABD

ABCABD

Алгоритм Кнута, Мориса и Пратта

- Общая схема КМП-алгоритма

```
i← 0; j ← 0;
While (j<M) and (i<N) do
  {
    While (j>=0) and (t[i] <>p[j]) do
                        j ← d[j];
    i ← i+1; j ← j+1
  }
```

- $T(n) = O(n+m)$ **Shift demo**

Алгоритм Боуера и Мура (1977г.)

1. **Сканирование слева направо, сравнение справа налево.**
2. **Совмещается начало текста (строки) и шаблона, проверка начинается с последнего символа шаблона.**
3. **Если символы совпадают, производится сравнение предпоследнего символа шаблона и т. д.**
4. **Если все символы шаблона совпали с наложенными символами строки, значит, подстрока найдена, и поиск окончен, в противном случае производится сдвиг слова по тексту**

Алгоритм Боуера и Мура

ABCABCABFABCABD

ABCABD {Не совпало с 'C', d['C']=3}

  ABCABD {Не совпало с F, 'F' нет в слове}

    ABCABD {Полное совпадение, слово найдено}

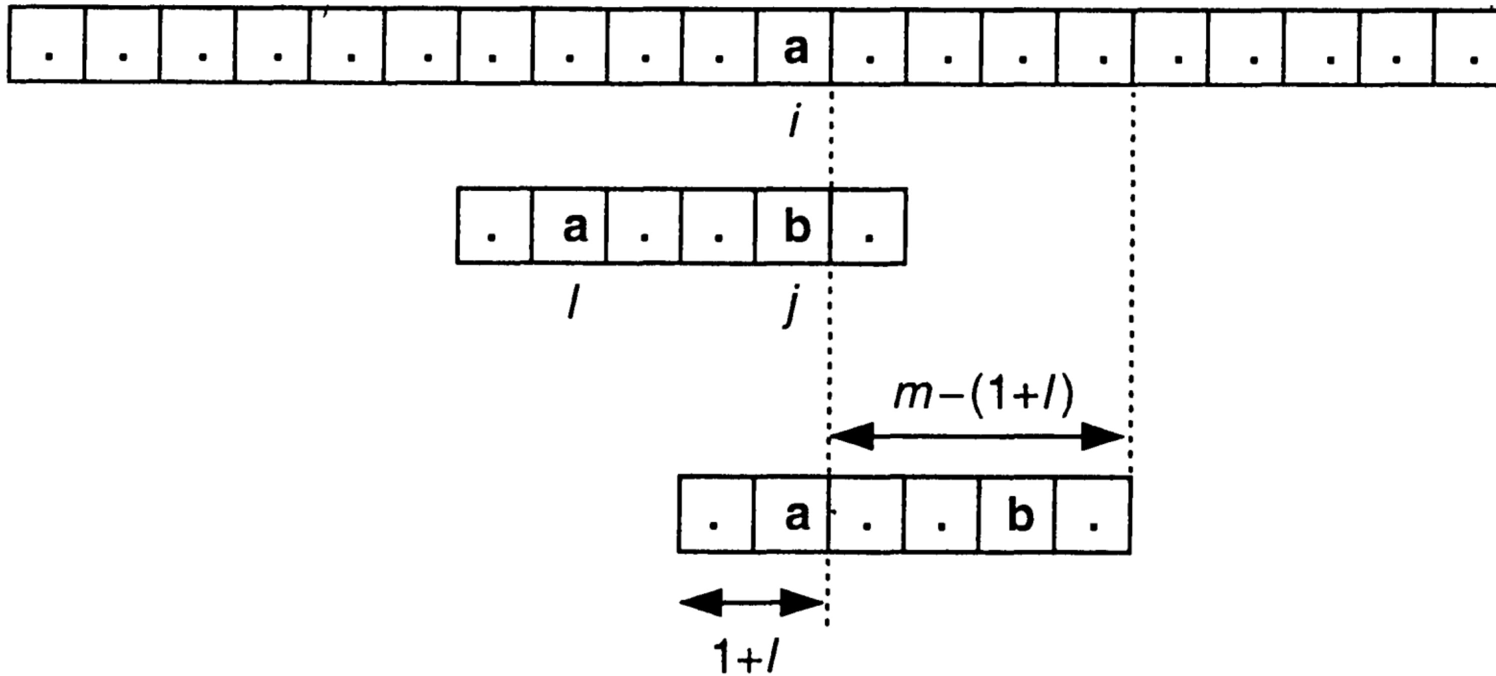
Алгоритм Боуера и Мура

Таблица сдвигов:

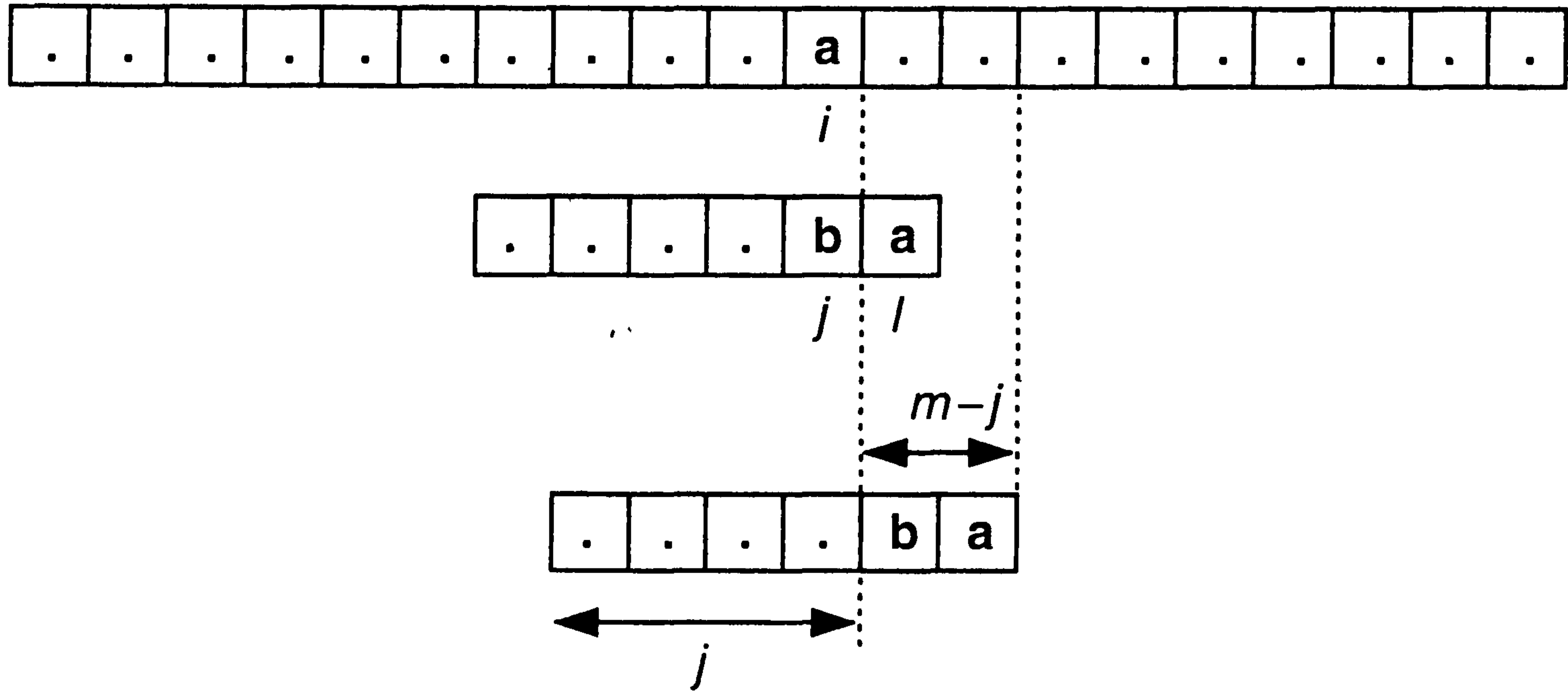
Для символов, отсутствующих в образце, сдвиг равен длине образца

Для символов из образца сдвиг равен расстоянию от последнего вхождения символа в образец до конца образца

Алгоритм Боуера и Мура



Алгоритм Боуера и Мура



Алгоритм Боуера и Мура

Алгоритм BMMatch(T,P):

Input: строка T (текст) из n символов и шаблон P из m символов.

Output: индекс начала первой подстроки T, совпадающей с P, или сообщение о том, что T не содержит подстроку P.

```
вычислить функцию last
i ← m − 1
j ← m − 1
    repeat
        if P[i] = T[i] then
            if j = 0 then
                return i          { возможно совпадение! }
            else
                i ← i − 1
                j ← j − 1
```

Алгоритм Боуера и Мура

else

$i \leftarrow i + m - \min(j, 1 + \text{last}(T[i]))$ { переход }

$j \leftarrow m - 1$

until $i > n - 1$

return "There is no substring of T matching P"

если c находится в P, то last(c) является индексом последнего (самого правого) появления c в P. В противном случае last(c) = −1.

T(n) =O(n+m+|Σ |) demo

http://www.fastcult.ru/606403.html

Алгоритм Рабина-Карпа

- **Рабин (Rabin) и Карп (Karp) (1987г.) предложили алгоритм поиска подстрок, показывающий на практике хорошую производительность в поиске совпадений множественных шаблонов.**
- **Алгоритм имеет уникальную особенность находить любую из s строк менее чем за время O(n) в среднем, независимо от размера s (поиск плагиата)**

В алгоритме Рабина-Карпа время O (m) затрачивается на предварительную обработку, а время его работы в наихудшем случае равно

$$O((n - m + 1)m)$$

Алгоритм Рабина-Карпа

Основная идея а

сигнатур

набор характеристик, однозначно идентифицирующий объект

Сигнатура — это с

представление да

удовлетворять следующим требованиям.

1. **Должна быть малая вероятность того, что сигнатуры разных строк имеют одинаковую сигнатуру.**
2. **Сигнатуру можно эффективно вычислить.**
3. **Сигнатуру можно вычислить на основе другой сигнатуры.**

Алгоритм Рабина-Карпа

- **В общем случае можно предположить, что каждый символ — это цифра в системе счисления с основанием d, где d = |Σ|**
- **Строку из k последовательных символов можно рассматривать как число длиной k.**
- **Таким образом, символьная строка ‘31415’ соответствует числу 31415.**

Алгоритм Рабина-Карпа

Для заданного образца $P [1..m]$ обозначим через p соответствующее ему десятичное значение (сигнатуру).

Аналогично, для заданного текста $T[1..n]$ обозначим через t_s десятичное значение подстроки $T [s + 1…s + m]$ длиной m

при $s = 0,1,..., n\text{-}m$

Алгоритм Рабина-Карпа

- **Очевидно, что $t_s = p$ тогда и только тогда, когда $T[s + 1..s + m] = P[1..m]$**
- **таким образом, s — допустимый сдвиг тогда и только тогда, когда $t_s = p$.**

Алгоритм Рабина-Карпа

Если бы значение p можно было вычислить за время $O(m)$, а все значения t_s за суммарное время $O(n-m+1)$,

то значения всех допустимых сдвигов можно было бы определить за время

$$O(m) + O(n-m+1) = O(n)$$

путем сравнения значения p с каждым из значений t_s

$p = a_o x^0 + a_1 x + a_2 x^2 + a_3 x^3 + .. + a_n x^n$

$$p = P[m] + 10(P[m-1] + 10(P[m-2] + \cdots + 10(P[2] + 10P[1])\cdots))$$

Значение t_o можно вычислить из ма

T[1..m] за время O

способом.

Чтобы вычислить о

$t_1, t_2, \ldots t_{n-m}$ за врем

Можно использовать рекуррентную формулу

Удаление цифры в старшем разряде

Добавление цифры в младший разряд

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

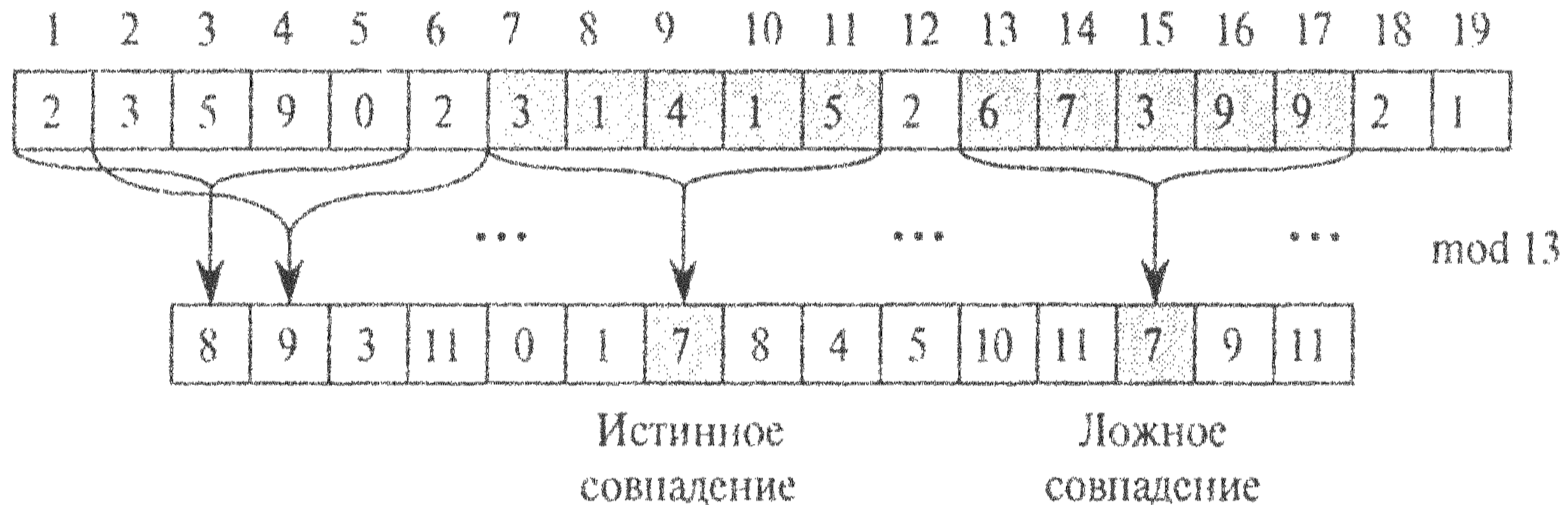
Алгоритм Рабина-Карпа

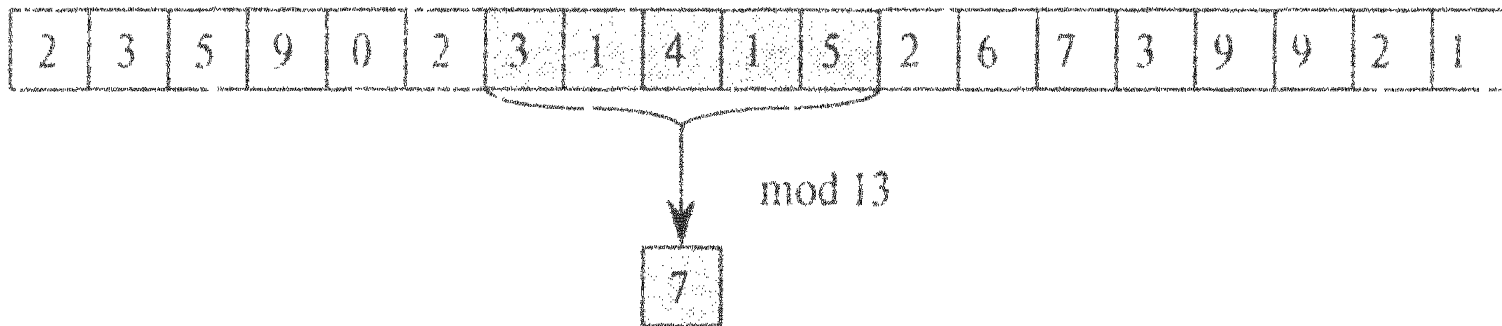
Для вычисления чисел p и t_s можно использовать операцию деление по модулю –q и вычисление t_s трансформируется в

$$t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1]) \bmod q,$$

где $h = d^{m-1} \pmod q$

Алгоритм Рабина-Карпа





Алгоритм Рабина-Карпа

```
RABIN_KARP_MATCHER(T, P, d, q)
 1  n ← length[T]
 2  m ← length[P]
 3  h ← d^(m-1) mod q
 4  p ← 0
 5  t_0 ← 0
 6  for i ← 1 to m                          ▷ Предварительная обработка
 7       do p ← (dp + P[i]) mod q
 8          t_0 ← (dt_0 + T[i]) mod q
 9  for s ← 0 to n − m                      ▷ Проверка
10       do if p = t_s
11             then if P[1..m] = T[s + 1..s + m]
12                     then print "Образец обнаружен при сдвиге" s
13          if s < n − m
14             then t_(s+1) ← (d(t_s − T[s + 1]h) + T[s + m + 1]) mod q
```

Алгоритм Рабина-Карпа

Во многих приложениях ожидается небольшое количество допустимых сдвигов (возможно, выражающееся некоторой константой c);

в таких приложениях математическое ожидание времени работы алгоритма равно сумме величины

$$O((n-m+1) + cm) = O(n + m)$$

и времени, необходимого для обработки ложных совпадений.

Алгоритм Рабина-Карпа

можно показать, что число ложных совпадений равно $O(n/q)$, потому что вероятность того, что произвольное число t_s будет эквивалентно p по модулю q, можно оценить как $1/q$.

- **Поскольку имеется всего $O(n)$ позиций, в которых проверка в строке 10 дает отрицательный результат, а на обработку каждого совпадения затрачивается время $O(m)$, математическое ожидание времени сравнения в алгоритме Рабина-Карпа равно**

$O(n)+ O(m(v+n/q))$,

- **Где v-кол-во допустимых сдвигов**

Алгоритм Рабина-Карпа

если $v=O(1)$ а $q \geq m$,

то $T(n) = O(n)$

Demo

Алгоритм Бойера — Мура — Хорспула

Алгоритм Демелки–Бейза-Ятса–Гоннета