Лекция 7

Поиск

Задача поиска

Объекты в общем случае будем рассматривать как записи произвольной природы, однако имеющие в своей структуре один и тот же *ключ* — поле, содержащее значение, которое сравнивается в процессе поиска с искомым ключом. В более общем случае ключ можно рассматривать как числовую функцию, которая строит значение ключа на основании сколь угодно сложного анализа всех данных, представленных в записи.

Далее при рассмотрении методов поиска и сортировки мы для простоты будем отождествлять записи с их ключами.

Следующие описания структур данных будут использоваться в дальнейших алгоритмах:

Последовательный поиск

Начинаем просмотр с первого элемента массива, продвигаясь дальше до тех пор, пока не будет найден нужный элемент или пока не будут просмотрены все элементы массива.

```
int  seek_linear(key x,  key a[],   int N)
{
   int  i=0;
   while ((i<N) && (a[i] != x))
      i++;
   if (i<N)
      return i;     /* элемент найден */
   else
      return -1;    /* элемент не найден */
}
```

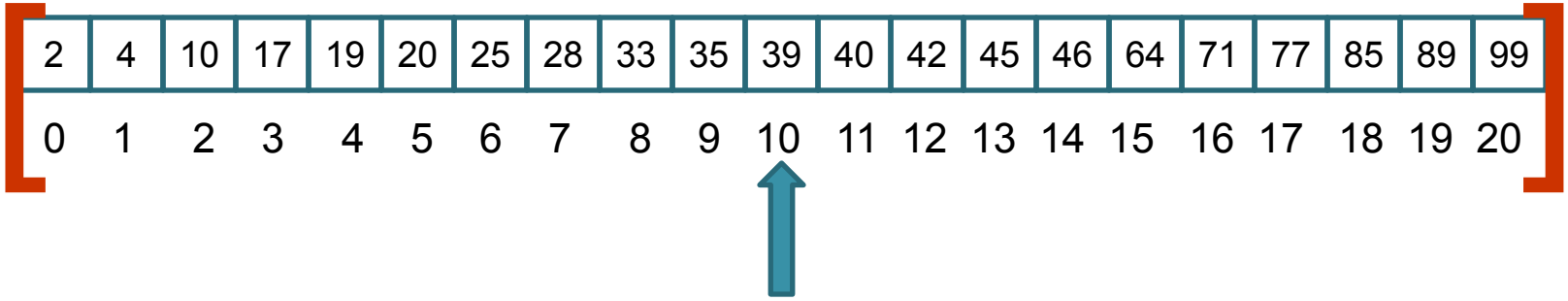
Бинарный поиск в массиве

Условие применения:

массив должен быть отсортированным.

Идея:

массив на каждом шаге делится пополам и выбирается та его часть, в которой должен находиться искомый элемент.



X = 33

Бинарный поиск - программа

```
int seek_binary(key x, key a[], int N)
{
    int left = 0;
    int right= N-1;
    int middle;
    do
     {
        middle=(left+right)/2;
        if (x == a[middle])
            return middle;
        else
            if(a[middle]< x)
                  left = middle + 1;
            else right = middle - 1;
    }
     while (left <= right);
     return -1;
}
```

Максимальное число сравнений равно $log_2 N$.

Прямой поиск подстроки

Пусть заданы строка s из N элементов и строка q из M элементов,

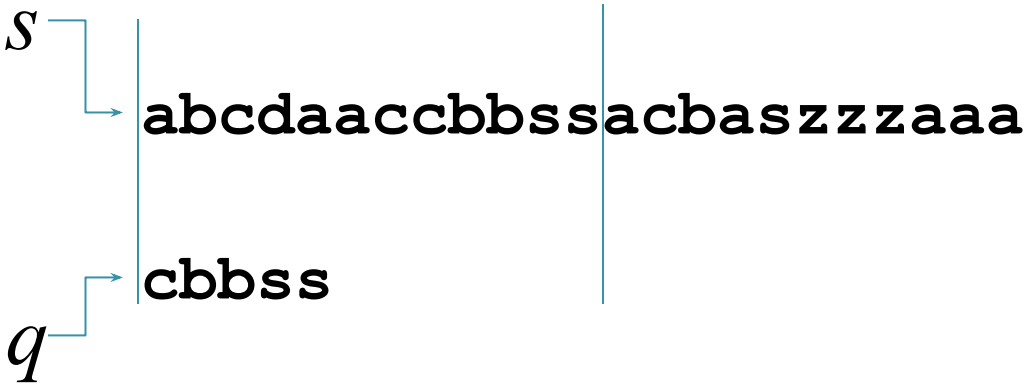
где $0 < M \leq N$.

Требуется определить, содержит ли строка s подстроку q, и в случае положительного результата выдать позицию k, с которой начинается вхождение q в s.

$$q[0] = s[k],\ q[1] = s[k+1],\ \ldots,\ q[M-1] = s[k+M-1].$$

Будем называть строку q *шаблоном* поиска.

Задача прямого поиска заключается в поиске индекса k, указывающего на первое с начала строки s совпадение с шаблоном q.



Прямой поиск подстроки - алгоритм

Вход: Строка s длины N и строка q длины M, где $0 < M \le N$.

Шаг 1. Шаблон q «накладывается» на строку s начиная с первого символа строки.

$k = 0;$ // номер символа строки, соответствующий

// первому символу шаблона

Шаг 2. $i = 0;$

Выполняется последовательное сравнение соответствующих символов, начиная от первого символа шаблона.

Если до i-й позиции шаблона соответствующие символы строки совпали,

а $q[i] \neq s[k + i]$, и $i < M$, то надо «сдвинуть» шаблон, т. е. «наложить» его на строку, начиная со следующего символа строки:

$k = k + 1;$

Шаг 3. Если $k < N - M + 1$, и $i < M$ то перейти на Шаг 2.

Выход: Если $q[1 .. M] = s[k .. k+M - 1]$, то выдать k,

иначе выдать сообщение, что подстрока не найдена.

Данный алгоритм реализуется с помощью двух вложенных циклов и в худшем случае требуется произвести $(N - M) \cdot M$ сравнений.

Прямой поиск подстроки - программа

```
int  seek_substring_A  (char s[],   char q[])
{
   int  i, j, k, N, M;
   N =  strlen(s);
   M =  strlen(q);
   k  =   -1;
   do  {
      k++;   /* k - смещение шаблона по строке */
      j = 0; /* j - смещение по шаблону; */
      while ((j<M) && s[k+j]==q[j]))
         j++;
      if (j == M)
         return k; /* шаблон найден */
   }
   while (k < N - M );
   return -1;           /* шаблон не найден */
}
```

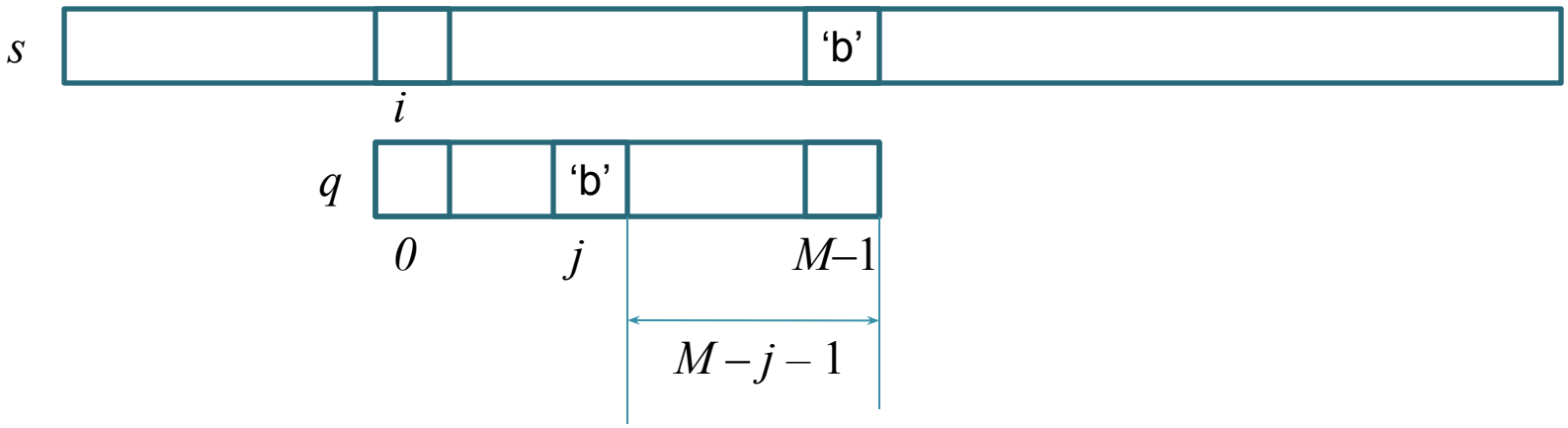
Алгоритм Бойера—Мура поиска подстроки в строке

Данный алгоритм ведет сравнение символов из строки и шаблона, начиная с $q[M-1]$ и с $s[i+M-1]$ элементов в обратном порядке.

В нем используется дополнительная таблица сдвигов d.

Для каждого символа x из алфавита (кроме последнего в шаблоне)

$d[x]$ есть расстояние от самого правого вхождения x в шаблоне до последнего символа шаблона. Для последнего символа в шаблоне $d[x]$ равно расстоянию от предпоследнего вхождения x до последнего или M, если предпоследнего вхождения нет.



Пример построения таблицы сдвигов

Для шаблона “**abcabeabce**” ($M = 10$)

$d[\text{'a'}] = 3$,

$d[\text{'b'}] = 2$,

$d[\text{'c'}] = 1$,

$d[\text{'e'}] = 4$

и для всех символов x алфавита, не входящих в шаблон,

$d[x] = 10$.

Алгоритм Бойера-Мура - описание

Будем последовательно сравнивать шаблон q с подстроками $s[i - M + 1..i]$ (в начале $i = M$).

Введем два рабочих индекса: $j = M, M - 1, \ldots, 1$ — пробегающий символы шаблона, $k = i, \ldots, i - M + 1$ — пробегающий подстроку.

Оба индекса синхронно уменьшаются на каждом шаге.

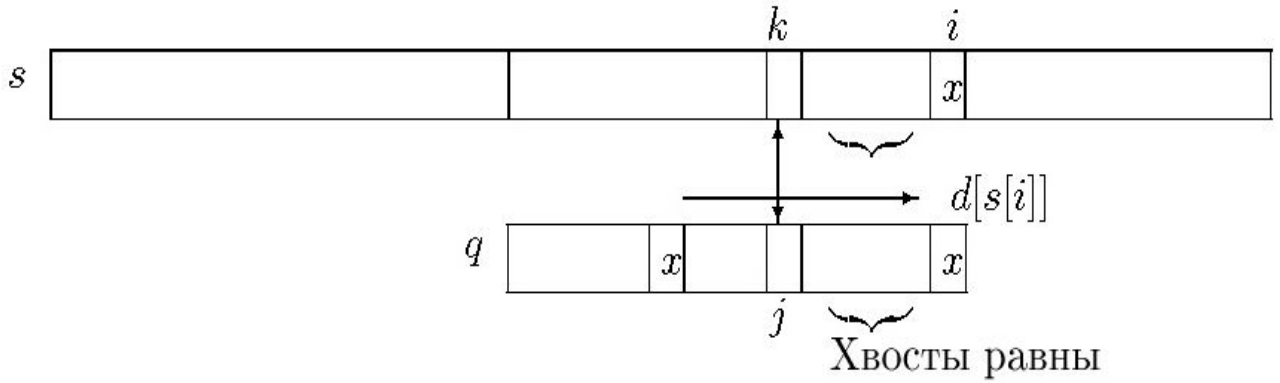
Если все символы q совпадают с подстрокой (т. е. j доходит до 0), то шаблон q считается найденным в s с позиции k ($k = i - M + 1$).

Если $q[j] \neq s[k]$ и $k = i$, т. е. расхождение случилось сразу же, в последних позициях, то q можно сдвинуть вправо так, чтобы последнее вхождение символа $s[i]$ в q совместилось с $s[i]$.

Если $q[j] \neq s[k]$ и $k < i$. т. е. последние символы совпали, то q сдвинется так, чтобы предпоследнее вхождение $s[i]$ в q совместилось с $s[i]$.

В обоих случаях величина сдвига равна $d[s[i]]$, по построению.

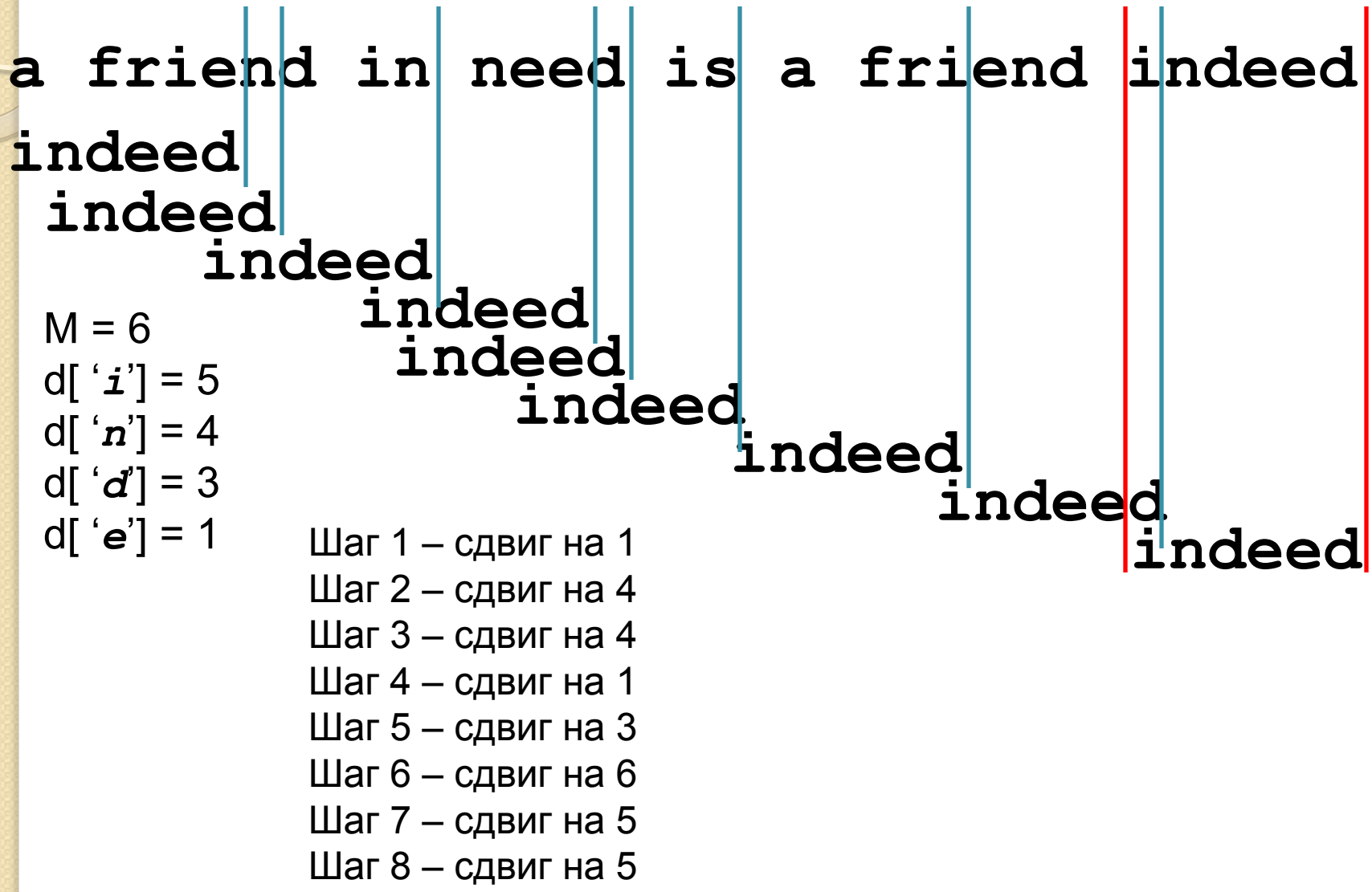
В частности, если $s[i]$ вообще не встречается в q, то смещение происходит сразу на полную длину шаблона M.



Реализация алгоритма Бойера-Мура на си

```
int  seek_substring_BM(unsigned char s[], unsigned char q[])
{     int d[256];
   int i, j, k, N, M;
   N =  strlen(s);
   M =  strlen(q);
  /* построение d */
   for (i=0; i<256; i++)
       d[i]=M;              /* изначально M во всех позициях */
   for (i=0; i<M-1; i++) /* M - i - 1 - расстояние до конца d */
     d[(unsigned char)q[i]]= M-i-1;/* кроме последнего символа*/
/* поиск */
   i= M-1;
   do {
       j = M-1; /* сравнение строки и шаблона */
       k = i;   /* j - по шаблону, k - по строке */
       while ((j >= 0) && (q[j] == s[k])) {
          k--; j--;
       }
       if (j < 0) return k+1; /* шаблон просмотрен полностью */
       i+=d[(unsigned)s[i]];/*сдвиг на расстояние d[s[i]]вправо*/
   } while (i < N);
   return -1;
}
```

Пример работы алгоритма Бойера - Мура



M = 6
d['*i*'] = 5
d['*n*'] = 4
d['*d*'] = 3
d['*e*'] = 1

Шаг 1 – сдвиг на 1
Шаг 2 – сдвиг на 4
Шаг 3 – сдвиг на 4
Шаг 4 – сдвиг на 1
Шаг 5 – сдвиг на 3
Шаг 6 – сдвиг на 6
Шаг 7 – сдвиг на 5
Шаг 8 – сдвиг на 5

Исследование сложности алгоритма Бойера-Мура

Определение длин исходных строк выполняется в Си поиском заключительного нулевого символа и требует, таким образом, времени *N* + *M.*

Для построения таблицы *d* необходимо занести значение *M* во все позиции таблицы и выполнить один проход по всем элементам шаблона *q,* т. е. таблица строится за время (256 + *M).*

Считаем, что *M* намного меньше *N.* Как правило, данный алгоритм требует значительно меньше *N* сравнений. В благоприятных обстоятельствах, а именно если последний символ шаблона всегда попадает на несовпадающий символ текста, максимальное число сравнении символов есть N/M.

$\frac{N}{M}$