

Цифровая сортировка (DigitalSort)

Пример. Дана последовательность S из 8 чисел:

$S : \overline{31} \quad \overline{03'} \quad \overline{20} \quad \overline{02} \quad \overline{03''} \quad \overline{33} \quad \overline{30} \quad \overline{21}$

$Q_0 : 20 \quad 30$

$Q_1 : 31 \quad 21$

$Q_2 : 02$

$Q_3 : 03' \quad 03'' \quad 33$

$S : \overline{20} \quad \overline{30} \quad \overline{31} \quad \overline{21} \quad \overline{02} \quad \overline{03'} \quad \overline{03''} \quad \overline{33}$

$Q_0 : 02 \quad 03' \quad 03''$

$Q_1 :$

$Q_2 : 20 \quad 21$

$Q_3 : 30 \quad 31 \quad 33$

$S : \overline{20} \quad \overline{30} \quad \overline{31} \quad \overline{21} \quad \overline{02} \quad \overline{03'} \quad \overline{03''} \quad \overline{33}$

Цифровая сортировка (DigitalSort)

Вначале числа из списка S распределяются по очередям, причём номер очереди определяется **последней цифрой каждого числа**.

Затем полученные очереди соединяются в список, для которого все действия повторяются, но распределение по очередям производится в соответствии **со следующей цифрой и т. д.**

В примере использованы **4 очереди**, т.к. каждая цифра принимает значение от 0 до 3, т.е. числа представлены в четверичной системе счисления.

Цифровая сортировка (DigitalSort)

В общем случае:

Дана последовательность из S чисел, представленных в m -ичной системе счисления.

Каждое число состоит из L цифр $d_1 d_2 \dots d_L$ (первая цифра – старшая, L -тая – младшая).

Тогда для каждой цифры d выполняется неравенство:

$$0 \leq d \leq m - 1$$

Поэтому для проведения сортировки потребуется

m очередей Q_0, Q_1, \dots, Q_{m-1} .

Цифровая сортировка (DigitalSort)

Пример.

Необходимо сортировать последовательность целых чисел типа `longint` (32 бита).

Сколько потребуются очередей?

Можно рассматривать **каждый байт** числа как **цифру**, принимающую значения от 0 до 255.

Таким образом, целое число: $d_1 d_2 d_3 d_4$,
 $L = 4$ цифры, $m = 256$ очередей.

Цифровая сортировка (DigitalSort)

Укрупненная схема алгоритма

DO ($j := L, L-1, \dots, 1$)


< Инициализация очередей Q >

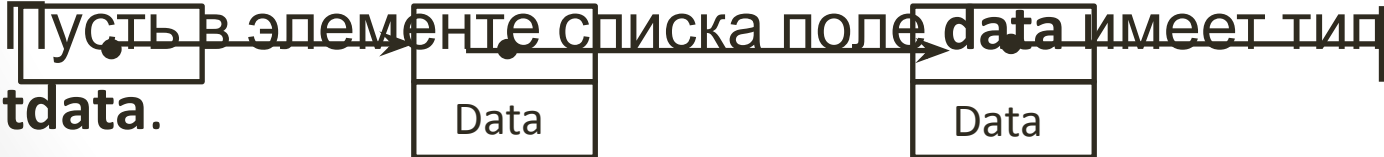
< Расстановка элементов из списка S в очереди Q

по j -той цифре >

< Соединение очередей Q в список S >

OD

Элемент списка: 

Пусть в элементе списка поле **data** имеет тип **tdata**. 

Цифровая сортировка (DigitalSort)

Рассмотрим основные операции:

1) Определение j -той цифры ключа сортировки

Задача: выделение произвольного байта в поле Data

Решение: необходимо в структуре элемента списка определить массив байтов, который накладывается в памяти компьютера на компоненту Data.

Удобно использовать описатель `union` (объединение).

Тогда структура элемента списка:

```
struct tLE { tLE * Next;  
            union { tData Data;  
                   byte Digit [ sizeof (tData) ];  
            }  
        }
```

Доступ к каждому k -тому байту поля Data: `Digit[k]`.

Цифровая сортировка (DigitalSort)

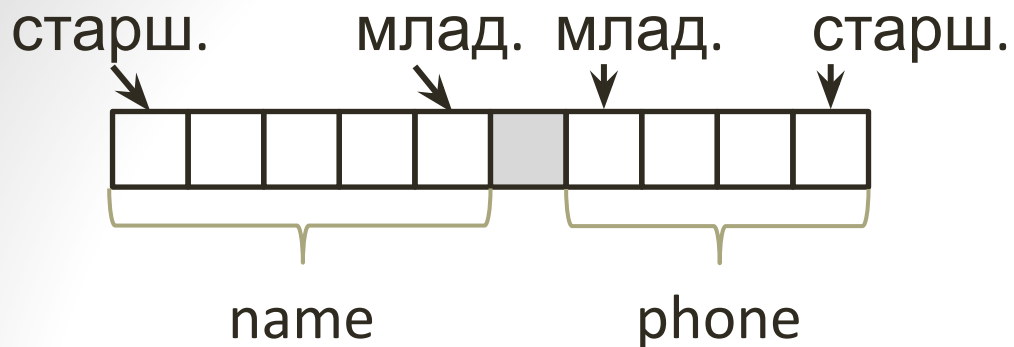
Рассмотрим **особенности реализации цифровой сортировки** для сложных структур:

Пример:

```
struct tData { char Name [5];  
               long Phone;  
};
```

sizeof (tData) = ?

sizeof (tData) = 10 байтов



Используем индексацию для удобства выбора байта.

Введем индексный массив KDI (Key Digit Index):

byte KDI [sizeof (tData)];

Пример: 1) ключ = Name, KDI = (1,2,3,4,5), L=5

2) ключ = Phone, KDI = (10,9,8,7), L=4

3) ключ = Phone +Name, KDI = (10,9,8,7,1,2,3,4,5)

4) ключ = 3 младших байта Phone +
+3 первых буквы Name,

KDI = (9 8 7 1 2 3) L=6

Цифровая сортировка (DigitalSort)

Тогда $KDI [j]$ - номер байта,
соответствующего j -той цифре ключа
сортировки, $j := L, L-1, \dots, 1$.

Операция выбора j -той цифры:

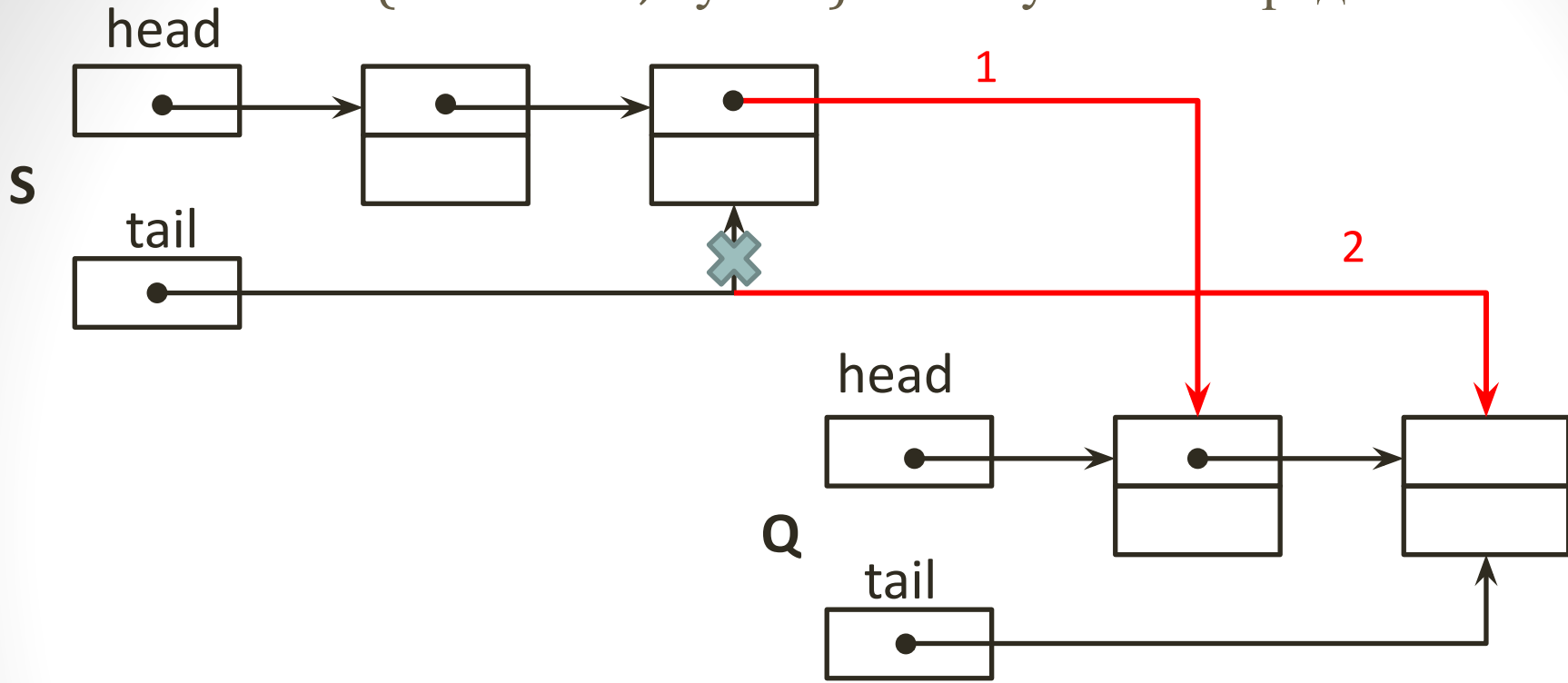
$d := \text{Digit} [KDI [j]]$

В алгоритме цифровой сортировки
операция выполняется в два этапа:

$k := KDI [j]$

$d := \text{Digit} [k]$

2) Соединение очередей. Имеется очередь Q (возможно, пустая) и непустая очередь S.



1) $S.tail \rightarrow next := Q.head$

2) $S.tail := Q.tail$

Трудоёмкость соединения очередей не зависит от количества элементов в очередях.

Если очередь Q пуста, выполнять присоединение нельзя

Цифровая сортировка (DigitalSort)

Алгоритм на псевдокоде

```
DO ( j := L, L-1, ... 1 )  
    DO ( i := 0, 1, ... 255 )  
        Qi. Tail := & Qi. Head  
    OD  
  
    p := S  
    k := KDI [j]  
    DO ( p ≠ NIL )  
        d := p → Digit [k]  
        Qd.Tail → Next := p  
        Qd.Tail := p  
        p := p → Next  
    OD  
OD
```

Цифровая сортировка (DigitalSort)

Алгоритм на псевдокоде (продолжение)

```
p := & S
  DO ( i := 0, 1, ... 255 )
    IF ( Qi. Tail ≠ & Qi. Head )
      p → Next := Qi. Head
      p := Qi. Tail
    FI
  OD
p → Next := NIL
OD
```

Цифровая сортировка (DigitalSort)

Трудоёмкость метода

$$T = O(L(n + m))$$

Замечания:

- 1) Цифровая сортировка устойчива.
- 2) Чтобы изменить направление сортировки на обратное, нужно очереди присоединять в обратном порядке.
- 3) При фиксированных m и L : $T = O(n) < O(n \log n)$

Границы применимости метода:

- 1) Метод применим, если задача сортировки сводится к задаче упорядочивания чисел, что не всегда возможно.
- 2) Если длина чисел (L) велика, то метод может проигрывать обычным методам сортировки (например, методу Хоара).

Метод	Трудоемкость	Устойчивость	Зависимость от упорядоченности
ShellSort	$O(n^{1,2})$	Не устойчив	Зависит
HeapSort	$O(n \log_2 n)$	Не устойчив	Практически не зависит
QuickSort	$O(n \log_2 n)$	Не устойчив	Зависит
MergeSort	$O(n \log_2 n)$	Устойчив	?
DigitalSort	$O(n)$	Устойчив	Не зависит