



# 内部排序

---

- 基本概念
- 内部排序算法
  - 插入排序
    - 直接插入、折半插入、希尔排序
  - 交换排序
    - 冒泡排序、快速排序
  - 选择排序
    - 简单选择排序、堆排序
  - 归并排序
    - 二路归并排序
  - 分配排序
    - 基数排序
- 内部排序算法的比较



# 概述

---

- 排序：
  - 将一组杂乱无章的数据按一定的规律顺次排列起来
- 数据表(*datalist*):
  - 待排序数据元素的有限集合
- 排序码(*key*):
  - 数据元素的组成：多个属性域/多个数据成员
  - 其中一个属性域用来区分元素, 作为排序依据, 即排序码
  - 每个数据表选用哪个属性域作为排序码, 视具体的应用需要而定



---

# 分配排序



# 基数排序 (Radix Sort)

---

- 基数排序是采用“分配”与“收集”的办法
- 用对多排序码进行排序的思想，实现对单排序码进行排序的方法

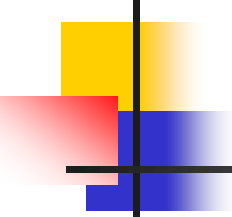


## 多排序码排序

---

扑克牌排序：

- 每张扑克牌有两个“排序码”：花色和面值
- 有序关系为：
  - ◆ 花色：♣ < ♦ < ♥ < ♠
  - ◆ 面值： $2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < 10 < J < Q < K < A$



---

多排序码排序:

把所有扑克牌排成以下次序:

♣ 2, ..., ♣ A, ♦ 2, ..., ♦ A, ♥ 2, ..., ♥ A, ♠ 2,  
..., ♠ A

- 排序后形成的有序序列叫做词典有序序列
- 两种排序方法:
  - 先按花色排序, 再按面值排序
  - 先按面值排序, 再按花色排序



## ■ 一般情况下

- 假定有一个  $n$  个元素的序列  $\{V_0, V_1, \dots, V_{n-1}\}$ ，且每个元素  $V_i$  中含有  $d$  个排序码：

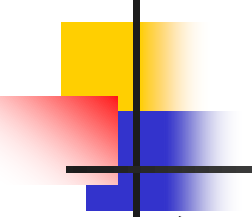
$$(K_i^1, K_i^2, \Lambda, K_i^d)$$

- 如果对于序列中任意两个元素  $V_i$  和  $V_j$  ( $0 \leq i < j \leq n-1$ ) 都满足：
$$(K_i^1, K_i^2, \Lambda, K_i^d) < (K_j^1, K_j^2, \Lambda, K_j^d)$$

- 则称序列对排序码  $(K^1, K^2, \dots, K^d)$  有序。

- 其中， $K^1$  称为最高位排序码， $K^d$  称为最低位排序码

- 排序码是由多个数据项组成的数据项组
- 依据它进行排序时，需要利用多排序码排序

- 
- 实现多排序码排序有两种常用的方法
    - ◆ 最高位优先MSD (Most Significant Digit first)
    - ◆ 最低位优先LSD (Least Significant Digit first)
  - LSD和MSD方法
    - 可应用于对一个排序码进行的排序
    - 将单排序码  $K_i$  看作是一个子排序码组

$$(K_i^1, K_i^2, \Lambda, K_i^d)$$





# 最高位优先法MSD

---

- 一个递归的过程：
  - ◆ 根据最高位排序码  $K^1$  排序, 得到若干元素组, 元素组中各元素都有相同排序码  $K^1$
  - ◆ 分别对每组中元素根据排序码  $K^2$  进行排序
    - 按  $K^2$  值的不同, 再分成若干个更小的子组, 每个子组中的元素具有相同的  $K^1$  和  $K^2$  值
  - ◆ 依此重复, 直到对排序码  $K^d$  完成排序为止
  - ◆ 把所有子组中的元素依次连接起来, 就得到一个有序的元素序列



# 最低位优先法LSD

---

- 依据最低位排序码 $K^d$ 对所有元素进行一趟排序
- 依据次低位排序码 $K^{d-1}$ 对上一趟排序的结果再排序
- 依次重复，直到依据排序码 $K^1$ 最后一趟排序完成，就可以得到一个有序的序列
- 使用LSD对每一个排序码进行排序时
  - 不需要再分组
  - 是整个元素组都参加排序



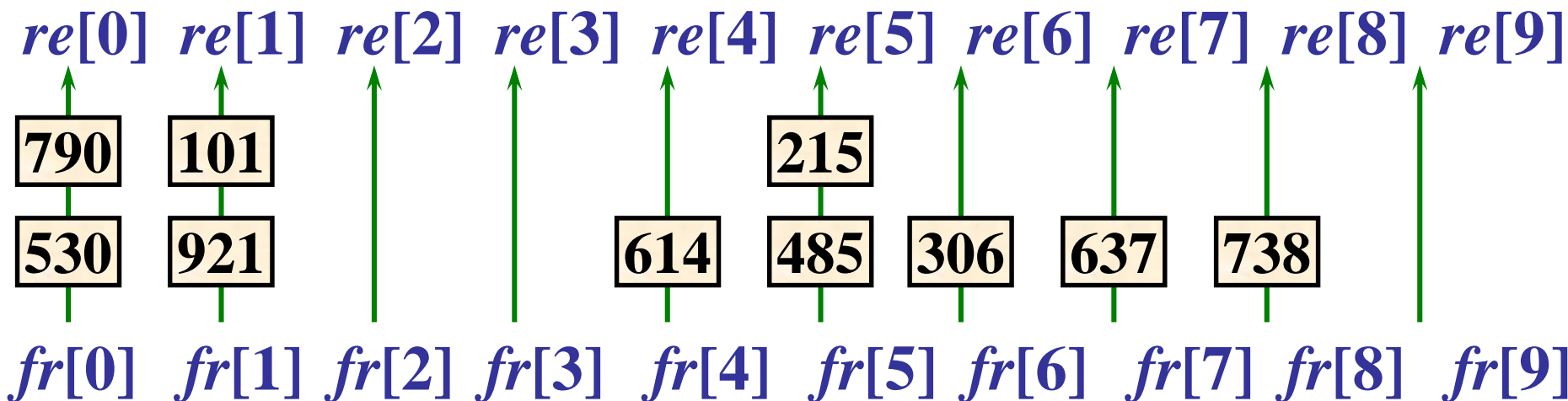
# 链式基数排序

- 基数排序是典型的LSD排序方法, 利用“分配”和“收集”对单排序码进行排序。
  - 把单排序码  $K_i$  看成是一个  $d$  元组:
$$(K_i^1, K_i^2, \dots, K_i^d)$$
  - 每一个分量  $K_i^j$  ( $1 \leq j \leq d$ ) 也可看成是一个排序码。
  - 分量  $K_i^j$  有  $radix$  种取值, 称  $radix$  为基数。
  - 例如排序码 921 可以看成是一个 3 元组 (9, 2, 1),
    - 每一位有 0, 1, ..., 9 等 10 种取值
    - 基数  $radix = 10$

## 基数排序的“分配”与“收集”过程第一趟

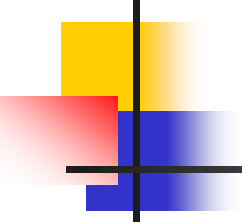
614 → 738 → 921 → 485 → 637 → 101 → 215 → 530 → 790 → 306

第一趟分配（按最低位  $i = 3$ ）



第一趟收集

530 → 790 → 921 → 101 → 614 → 485 → 215 → 306 → 637 → 738

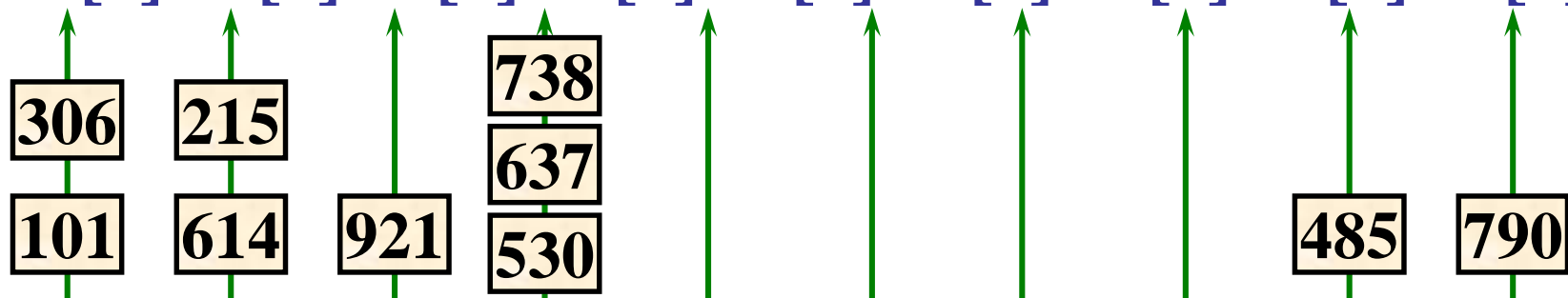
- 
- $d$ 元组中的每一位分量
  - 把元素序列中的所有元素, 按  $K_i^j$  的取值, 先 “分配” 到  $rd$  个队列中去
  - 再按各队列的顺序, 依次把元素从队列中 “收集” 起来, 这样所有元素按取值  $K_i^j$  排序完成

## 基数排序的“分配”与“收集”过程第二趟

530 → 790 → 921 → 101 → 614 → 485 → 215 → 306 → 637 → 738

第二趟分配（按次低位  $i = 2$ ）

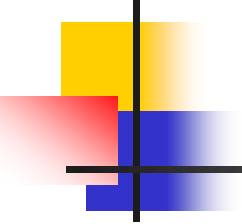
$re[0]$   $re[1]$   $re[2]$   $re[3]$   $re[4]$   $re[5]$   $re[6]$   $re[7]$   $re[8]$   $re[9]$



$fr[0]$   $fr[1]$   $fr[2]$   $fr[3]$   $fr[4]$   $fr[5]$   $fr[6]$   $fr[7]$   $fr[8]$   $fr[9]$

第二趟收集

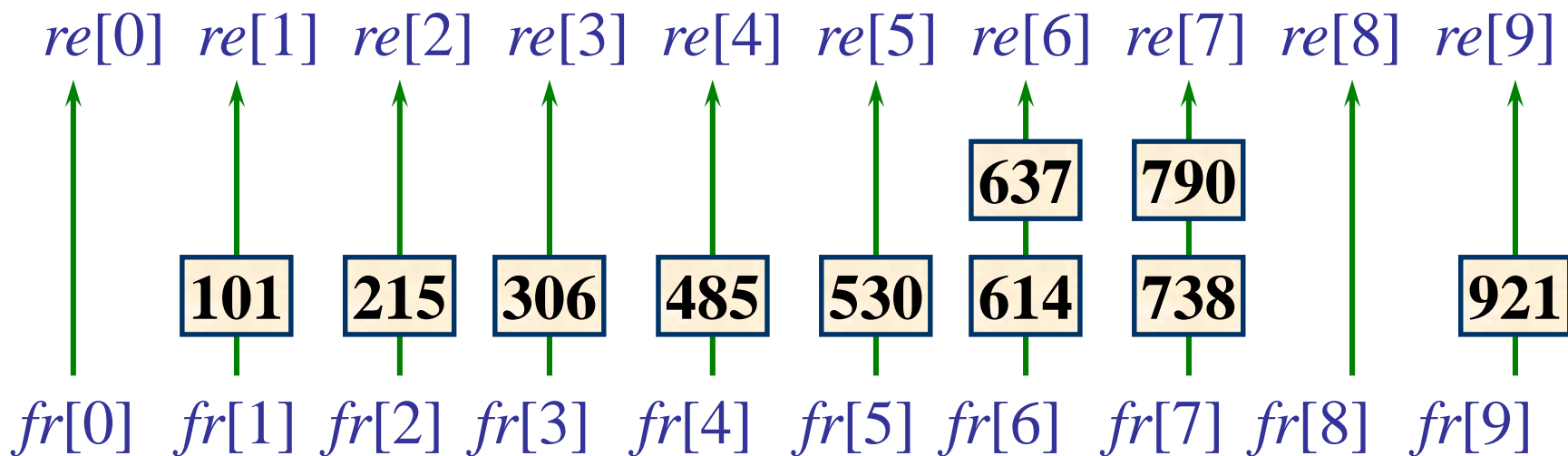
101 → 306 → 614 → 215 → 921 → 530 → 637 → 738 → 485 → 790

- 
- 
- 对于所有元素的排序码 $K_0, K_1, \dots, K_{n-1}$
  - 依次对各位的分量, 让  $j = d, d-1, \dots, 1$
  - 分别用“分配”、“收集”的运算逐趟进行排序

## 基数排序的“分配”与“收集”过程第三趟

101 → 306 → 614 → 215 → 921 → 530 → 637 → 738 → 485 → 790

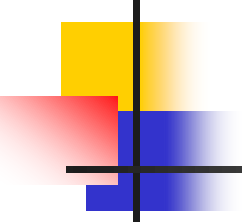
第三趟分配（按最高位  $i = 1$ ）

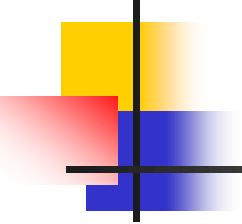


第三趟收集

101 → 215 → 306 → 485 → 530 → 614 → 637 → 738 → 790 → 921



- 
- 
- 最后一趟“分配”、“收集”完成后
  - 所有元素就按其排序码的值从小到大排好序了

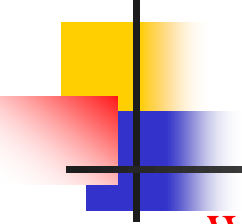
- 
- 各队列：采用链式队列结构
  - 分配到同一队列的排序码，用链接指针链接起来
  - 每一队列设置两个队列指针：
    - `int front [radix]`指示队头
    - `int rear [radix]`指向队尾
  - 为了有效地存储和重排  $n$  个待排序元素，以静态链表作为它们的存储结构



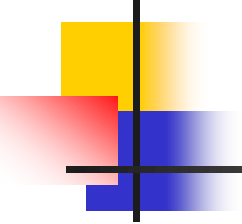
# 链表基数排序

---

```
#include "staticList.h"           //静态链表头文件
const int radix = 10;             //基数
template <class T>
void Sort (staticlinkList<T>& L, int d)
{
    int rear[radix], front[radix]; //队尾与队头指针
    int i, j, k, last, current, n = L.Length();
    for (i = 0; i < n; i++) L[i].link = i+1;
    L[n].link = 0;                 //初始化, 循环链表
    current = 1;                   //取表元素计数
    for (i = d; i >= 1; i--)       //按排序码key[i]分配
    {
        for (j = 0; j < radix; j++) front[j] = 0;
```



```
while (current != 0) //分配
{ k = getDigit (L[current], i); //取第i个排序码
  if (front[k] == 0) front[k] = current;
    //第k个队列空, 该元素为队头
  else L[rear[k]].link = current; //不空, 尾链接
  rear[k] = current; //该元素成为新的队尾
  current = L[current].link; //下一个元素
}
j = 0; //依次从各队列收集并拉链
while (front[j] == 0) j++; //跳过空队列
L[0].link = current = front[j]; //新链表的链
last = rear[j]; //非空队列链尾
```



---

```
for (k = j+1; k < radix; k++)//连接其余队列
```

```
    if (front[k] != 0)        //队列非空
```

```
    { L[last].link = front[k];
```

```
    //前一非空队列队尾链接到第k队列队头
```

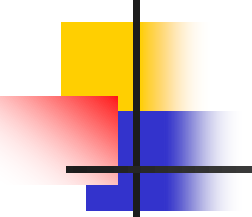
```
        last = rear[k];        //记第k队列队尾
```

```
    }
```

```
L[last].link = 0;                //新链表表尾
```

```
}                                //下一趟分配与收集
```

```
}
```

- 
- 每个排序码有 $d$ 位，需要重复执行 $d$ 趟“分配”与“收集”
  - 每趟对 $n$ 个元素进行“分配”，对 $radix$ 个队列进行“收集”
  - 总时间复杂度为： $O(d(n+radix))$
  - 基数 $radix$ 相同，对于元素个数较多而排序码位数较少的情况，使用链式基数排序较好
  - 基数排序：增加 $n+2radix$ 个附加链接指针
  - 基数排序是稳定的排序方法



# 内部排序

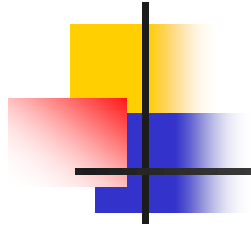
---

- 基本概念
- 内部排序算法
  - 插入排序
    - 直接插入、折半插入、希尔排序
  - 交换排序
    - 冒泡排序、快速排序
  - 选择排序
    - 简单选择排序、堆排序
  - 归并排序
    - 二路归并排序
  - 分配排序
    - 基数排序
- 内部排序算法的比较

# 各种排序方法的比较

排 序 方 法	比较次数		移动次数		稳 定 性	附加存储	
	最好	最差	最好	最差		最好	最差
直接插入排序	$n$	$n^2$	0	$n^2$	$\sqrt$	1	
折半插入排序	$n \log_2 n$		0	$n^2$	$\sqrt$	1	
冒泡排序	$n$	$n^2$	0	$n^2$	$\sqrt$	1	
快速排序	$n \log_2 n$	$n^2$	$\log_2 n$	$n$	$\times$	$\log_2 n$	$n$
简单选择排序	$n^2$		0	$n$	$\times$	1	
锦标赛排序	$n \log_2 n$		$n \log_2 n$		$\sqrt$	$n$	
堆排序	$n \log_2 n$		$n \log_2 n$		$\times$	1	
归并排序	$n \log_2 n$		$n \log_2 n$		$\sqrt$	$n$	





The End