数据结构与算法（八）

张铭 主讲

采用教材：张铭，王腾蛟，赵海燕 编写
高等教育出版社，2008. 6（“十一五”国家级规划教材）

http://www.jpk.pku.edu.cn/pkujpk/course/sjjg

大 纲

- 8.1 排序问题的基本概念
- 8.2 插入排序 (Shell 排序)
- 8.3 选择排序 (堆排序)
- 8.4 交换排序
  - 8.4.1 冒泡排序
  - 8.4.2 快速排序
- 8.5 归并排序
- **8.6 分配排序**和索引排序
- 8.7 排序算法的时间代价
- 内排序知识点总结

8.6 分配排序和基数排序

- 不需要进行纪录之间两两比较
- 需要事先知道记录序列的一些具体情况

8.6.1 桶式排序

- 事先知道序列中的记录都位于某个小区间段 [0, m) 内
- 将具有相同值的记录都分配到同一个桶中，然后依次按照编号从桶中取出记录，组成一个有序序列

8.6.1 桶式排序

待排数组： 7 3 8 9 6 1 8′ 1′ 2

每个桶 count：

0	1	2	3	4	5	6	7	8	9
0	2	1	1	0	0	1	1	2	1

＋ ＋ ＋ ＋

前若干桶的累计count：

0	1	2	3	4	5	6	7	8	9
0	2	3	4	4	4	5	6	8	9

8.6.1 桶式排序

桶排序示意

待排数组： 7 3 8 9 6 1 8′ 1′ 2

每个桶count：

0	1	2	3	4	5	6	7	8	9
0	2	1	1	0	0	1	1	2	1

前若干桶的累计count：

0	0	2	3	4	4	4	5	6	8

收集：

0	1	2	3	4	5	6	7	8
1	1’	2	3	6	7	8	8’	9

桶式排序算法

```
template <class Record> void BucketSort(Record Array[], int n, int max) {
    Record *TempArray = new Record[n];   // 临时数组
    int *count = new int[max];           // 桶容量计数器
    int i;
    for (i = 0; i < n; i++)              // 把序列复制到临时数组
        TempArray[i] = Array[i];
    for (i = 0; i < max; i++)            // 所有计数器初始都为0
        count[i] = 0;
    for (i = 0; i < n; i++)              // 统计每个取值出现的次数
        count[Array[i]]++;
    for (i = 1; i < max; i++)            // 统计小于等于i的元素个数
        count[i] = count[i-1]+count [i];  // c [i]记录i+1的起址
    for (i = n-1; i >= 0; i--)           // 尾部开始，保证稳定性
        Array[--count[TempArray[i]]] = TempArray[i];
}
```

算法分析

- 数组长度为 n, 所有记录区间 $[0, m)$ 上
- 时间代价：
  - 统计计数：$\Theta(n+m)$ ， 输出有序序列时循环 n 次
  - 总的时间代价为 $\Theta(m+n)$
  - 适用于 m 相对于 n 很小的情况
- 空间代价：
  - m 个计数器，长度为 n 的临时数组，$\Theta(m+n)$
- 稳定

8.6.2 基数排序

- 桶式排序只适合 m 很小的情况
- 基数排序： 当 m 很大时，可以将一个记录的值即排序码拆分为多个部分来进行比较

8.6.2 基数排序

- 假设长度为 n 的序列

  $$R = \{ r_0 , r_1 , \cdots , r_{n-1} \}$$

  记录的排序码 K 包含 d 个子排序码

  $$K = (k_{d-1} , k_{d-2} , \cdots , k_1 , k_0)$$

- R 对排序码有序，即对于任意

  两个记录 R_i，R_j $(0 \le i < j \le n-1)$，都满足

  $$(k_{i,d-1} , k_{i,d-2} , \cdots , k_{i,1} , k_{i,0}) \le$$

  $$(k_{j,d-1} , k_{j,d-2} , \cdots , k_{j,1} , k_{j,0})$$

例子

例如：对 0 到 9999 之间的整数进行排序

- 将四位数看作是由四个排序码决定，即千、百、十、个位，其中千位为最高排序码，个位为最低排序码。基数 r=10。
- 可以按千、百、十、个位数字依次进行4次桶式排序
- 4趟分配排序后，整个序列就排好序了

8.6.2 基数排序

黑桃♠(S) > 红心♥(H) > 方片♦(D) > 梅花♣ (C)

♠3 ♥J ♣8 ♥9 ♠9 ♦3 ♣1 ♦7

- 高位先排，递归分治
  - 先按花色：♣8 ♣1 ♦3 ♦7 ♥J ♥9 ♠3 ♠9
  - 再按面值：♣1 ♣8 ♦3 ♦7 ♥9 ♥J ♠3 ♠9
- 低位先排，要求稳定排序！
  - 先面值：♣1 ♠3 ♦3 ♦'7 ♣8 ♥9 ♠'9 ♥'J
  - 再花色：♣1 ♣'8 ♦3 ♦'7 ♥9 ♥'J ♠3 ♠'9

高位优先法

- MSD，Most Significant Digit first
- 先处理高位 $k_{d\text{-}1}$ 将序列分到若干桶中
- 然后再对每个桶处理次高位 $k_{d\text{-}2}$ ，分成更小的桶
- 依次重复，直到对 k_0 排序后，分成最小的桶，每个桶内含有相同排序码（$k_{d\text{-}1}$，…，k_1，k_0）
- 最后将所有的桶中的数据依次连接在一起，成为一个有序序列
- 这是一个 **分、分、…、分、收** 的过程

低位优先法

- LSD，Least Significant Digit first
- 从最低位 k_0 开始排序
- 对于排好的序列再用次低位 k_1 排序；
- 依次重复，直至对最高位 $k_{d\text{-}1}$ 排好序后，整个序列成为有序的
- **分、收**；**分、收**；…；**分、收**的过程
  - 比较简单，计算机常用

基数排序的实现

- 主要讨论 LSD
  - 基于顺序存储
  - 基于链式存储
- 原始输入数组 R 的长度为 n，基数为 r，排序码个数为 d

基于顺序存储的基数排序

初始数组内容： 97 53 88 59 26 41 88’ 31 22

	0	1	2	3	4	5	6	7	8	9
第一趟：count	**0**	2	1	1	**0**	**0**	1	1	2	1

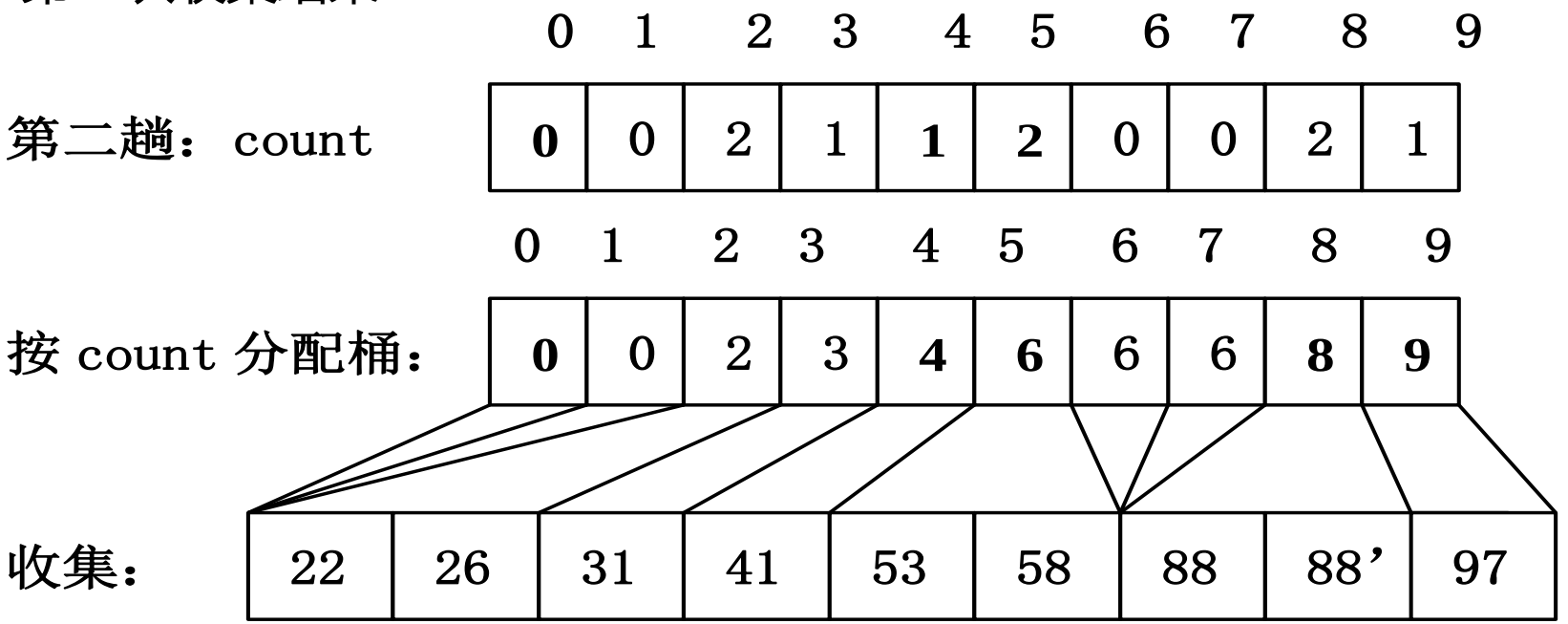
	0	1	2	3	4	5	6	7	8	9
按 count 分配桶：	**0**	2	3	4	**4**	**4**	5	6	**8**	**9**

收集：	41	31	22	53	26	97	88	88’	59

(a) 第一趟分配个位

基于顺序存储的基数排序

第一次收集结果：41 31 22 53 26 97 88 88’ 59



最终排序结果：22 26 31 41 53 59 88 88’ 97

(b) 第二趟分配十位

基于数组的基数排序

```
template <class Record>
void RadixSort(Record Array[], int n, int d, int r) {
   Record  *TempArray = new Record[n];
   int *count = new int[r];        int i, j, k;
   int Radix = 1;     // 模进位，用于取Array[j]的第i位
   for (i = 1; i <= d; i++)  {    // 对第 i 个排序码分配
      for (j = 0; j < r; j++)
         count[j] = 0;                // 初始计数器均为0
      for (j = 0; j < n; j++) {     // 统计每桶记录数
         k = (Array[j] / Radix) % r;  // 取第i位
         count[k]++;                  // 相应计数器加1
      }
```

8.6.2 基数排序

```
        for (j = 1; j < r; j++)      // 给桶划分下标界
            count[j] = count[j-1] + count[j];
        for (j = n-1; j >= 0; j--) {  // 从数组尾部收集
            k = (Array[j] / Radix ) % r;   // 取第 i 位
            count[k]--;                    // 桶剩余量计数器减1
            TempArray[count[k]] = Array[j]; // 入桶
        }
        for (j = 0; j < n; j++)      // 内容复制回 Array 中
            Array[j] = TempArray[j];
        Radix *= r;                  // 修改模Radix
    }
}
```

顺序基数排序代价分析

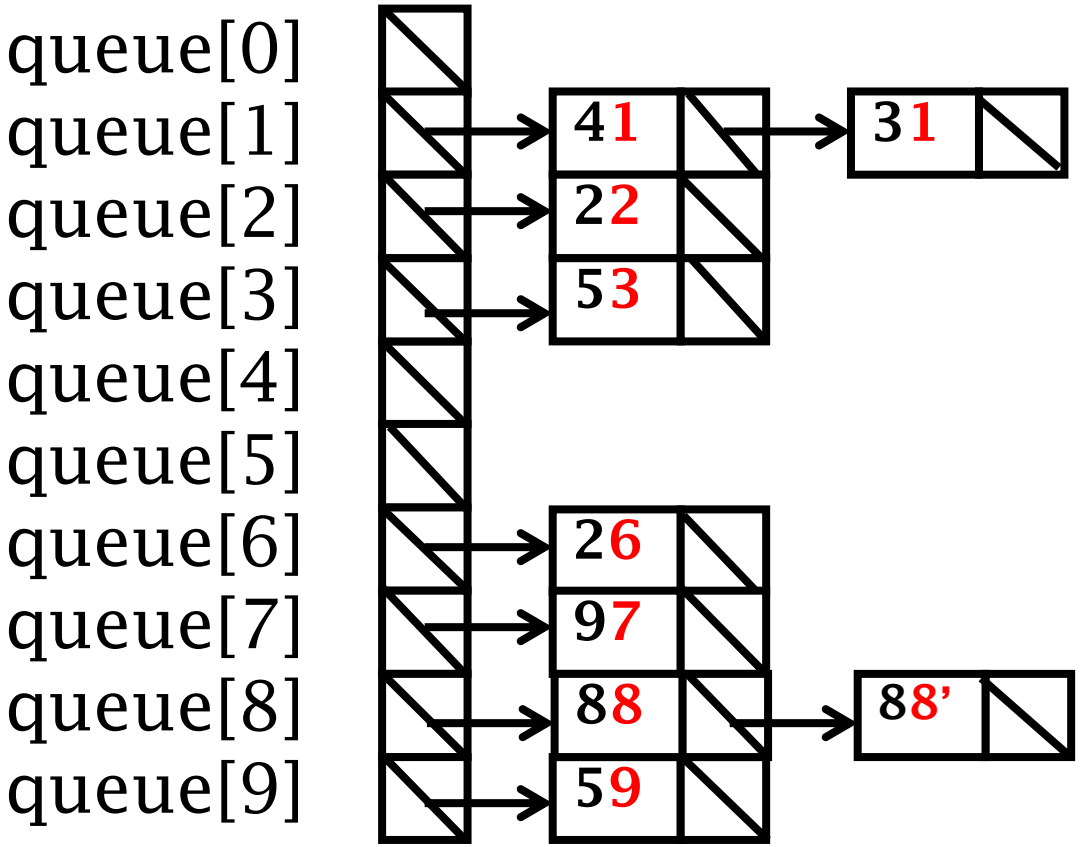
- 空间代价：
  - 临时数组，n
  - r 个计数器
  - 总空间代价 $\Theta(n+r)$
- 时间代价
  - 桶式排序：$\Theta(n+r)$
  - d 次桶式排序
  - $\Theta(d\cdot(n+r))$

基于静态链的基数排序

- 将分配出来的子序列存放在 r 个 (静态链组织的) 队列中
- 链式存储避免了空间浪费情况

97	53	88	59	26	41	88’	31	22

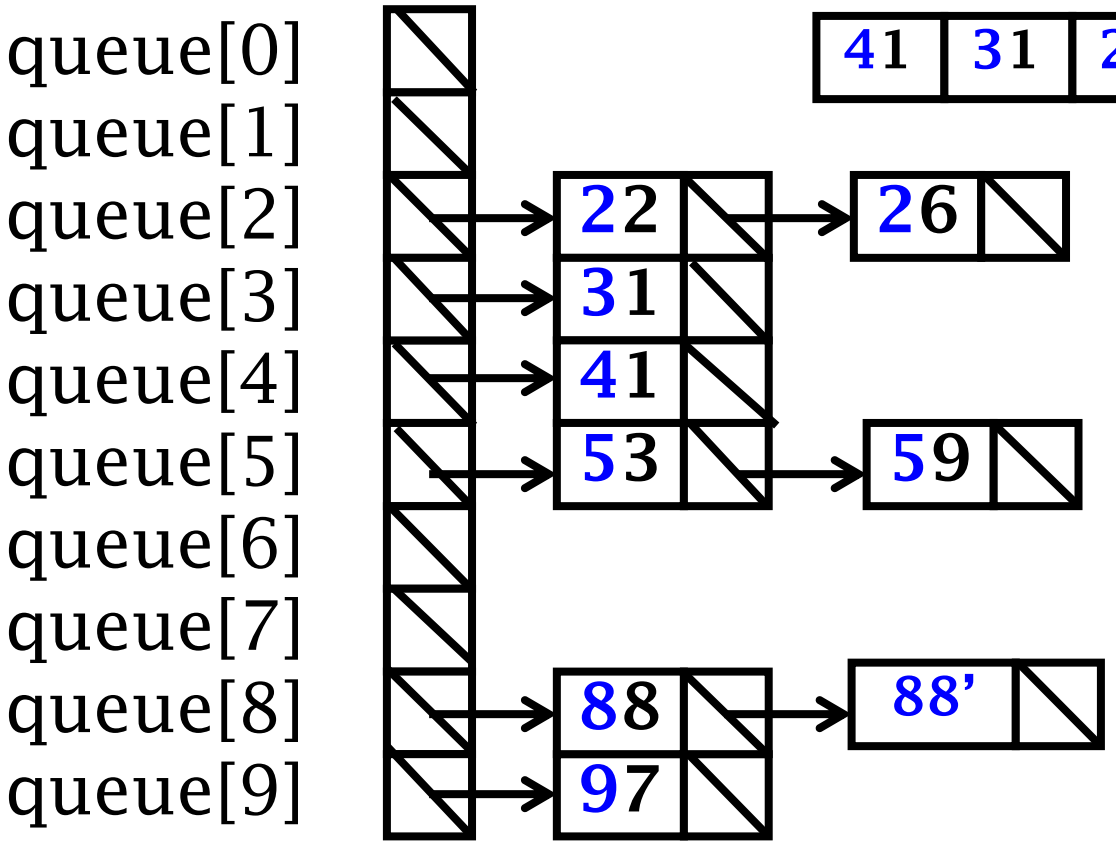
(a)初始链表内容

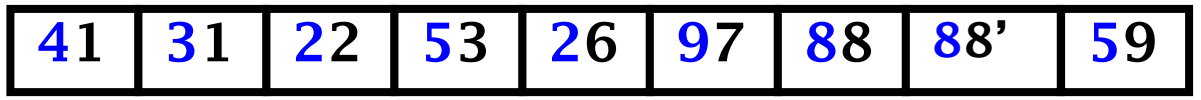


(b)第一趟分配

(c) 第一趟收集

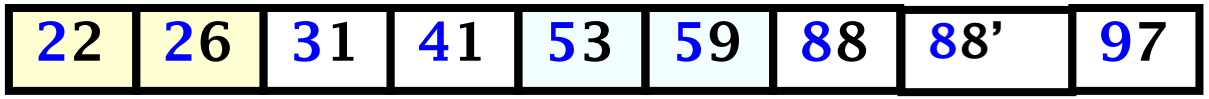
41	31	22	53	26	97	88	88’	59





(d)第二趟分配

(e) 第二趟收集结果（最终结果）



静态队列定义

```
class Node {          // 结点类
public:
    int key;          // 结点的关键码值
    int next;         // 下一个结点在数组中的下标
};

class StaticQueue { // 静态队列类
public:
    int head;
    int tail;
};
```

基于静态链的基数排序

```
template <class Record>
void RadixSort(Record *Array, int n, int d, int r) {
    int i, first = 0;                  // first指向第一个记录
    StaticQueue *queue = new StaticQueue[r];
    for (i = 0; i < n-1; i++)
        Array[i].next = i + 1;    // 初始化静态指针域
    Array[n-1].next = -1;          // 链尾next为空
    // 对第i个排序码进行分配和收集，一共d趟
    for (i = 0; i < d; i++)  {
        Distribute(Array, first, i, r, queue);
        Collect(Array, first, r, queue);
    }
    delete[] queue;
    AddrSort(Array, n, first); // 整理后，按下标有序
}
```

8.6.2 基数排序

```
template <class Record>
void Distribute(Record *Array, int first, int i, int r,
StaticQueue *queue) {
    int j, k, a, curr = first;
    for (j =  0; j < r; j++)   queue[j].head = -1;
    while (curr != -1)  {          // 对整个静态链进行分配
        k = Array[curr].key;
        for (a = 0; a < i; a++)   // 取第i位排序码数字k
            k = k / r;
        k = k % r;
        if (queue[k].head == -1)  // 把数据分配到第k个桶中
            queue[k].head = curr;
        else  Array[queue[k].tail].next = curr;
        queue[k].tail = curr;
        curr = Array[curr].next;   // curr移动，继续分配
    }
}
```

8.6.2 基数排序

```
template <class Record>
void Collect(Record *Array, int& first, int r, StaticQueue
*queue) {
    int last, k=0;                  // 已收集到的最后一个记录
    while (queue[k].head == -1)     k++; // 找到第一个非空队
    first = queue[k].head; last = queue[k].tail;
    while (k < r-1) {               // 继续收集下一个非空队列
        k++;
        while (k < r-1 && queue[k].head == -1)
            k++;
        if (queue[k].head != -1) {// 试探下一个队列
            Array[last].next = queue[k].head;
            last = queue[k].tail;    // 最后一个为序列的尾部
        }
    }
    Array[last].next = -1;          // 收集完毕
}
```

链式基数排序算法代价分析

- 空间代价
  - n 个记录空间
  - r 个子序列的头尾指针
  - $\Theta(n + r)$
- 时间代价
  - 不需要移动记录本身，
    只需要修改记录的 next 指针
  - $\Theta(d \cdot (n+r))$

基数排序效率

- 时间代价为 $\Theta(d \cdot n)$， 实际上还是 $\Theta(n\log\ n)$
  - 没有重复关键码的情况，需要 n 个不同的编码来表示它们
  - 也就是说，$d >= \log_r n$，即在 $\Omega(\log\ n)$ 中

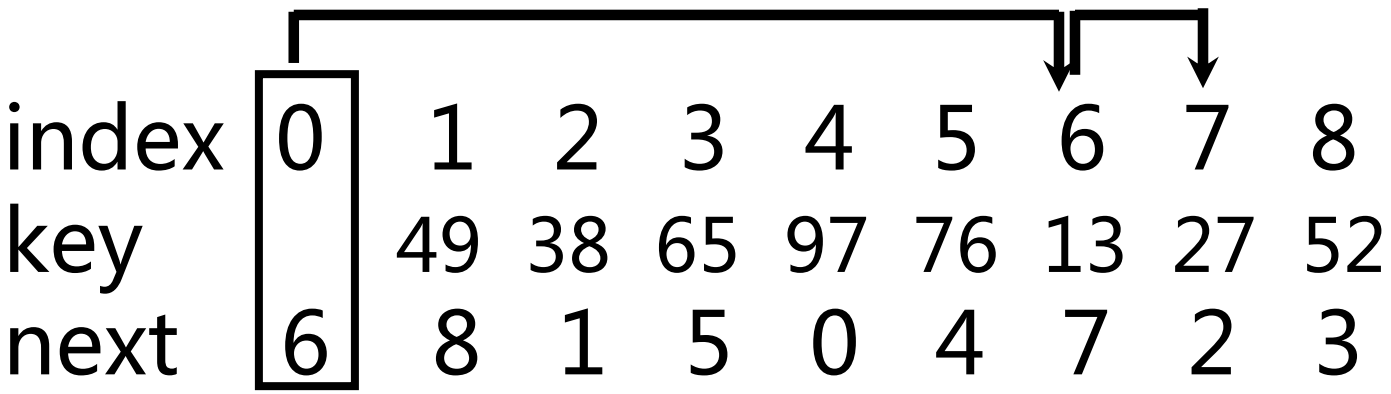
8.6.1 桶式排序

思考：排序该排什么？

```
Template <class Elem >
void BucketSort(int n,int min,int max)  {
    int i, num;
    int count[max-min];
    for (i=min; i<max; i++)   // 计数器初始为0
        count[i]=0;
    for (i=0; i<n; i++)  {        // 开始统计计数
        cin >> num;
        count[num]++;
    }
    for (i=min; i<max; i++)   //按顺序输出
        while (count[i]>0) {
            cout <<  i;
            count[i]--;
        }
}
```

思 考

1. 顺序和链式基数排序的优劣？
2. 链式基数排序的结果整理？



有头结点的单链表的插入算法

链式基数排序的结果

数据结构与算法

谢谢聆听

国家精品课“数据结构与算法”
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

张铭，王腾蛟，赵海燕
高等教育出版社，2008. 6。“十一五”国家级规划教材