

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

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

记录的排序码 K 包含 d 个子排序码

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

- R 对排序码有序，即对于任意

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

$$(k_{i,d-1}, k_{i,d-2}, \dots, k_{i,1}, k_{i,0}) \leq$$

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

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



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-1} 将序列分到若干桶中
- 然后再对每个桶处理次高位 k_{d-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 $(k_{d-1}, \dots, k_1, k_0)$
- 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-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

8.6.2 基数排序

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

8.6.2 基数排序

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

0 1 2 3 4 5 6 7 8 9

第二趟：count

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

按 count 分配桶：

0	0	2	3	4	6	6	6	8	9
---	---	---	---	---	---	---	---	---	---

收集：

22	26	31	41	53	58	88	88'	97
----	----	----	----	----	----	----	-----	----

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
 - $\Theta(d(n+r))$

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]--;
        }
}
```