分配排序

基本思想：把排序码分解成若干部分，然后通过对各个部分排序码的分别排序，最终达到整个排序码的排序

例：对52张扑克牌按以下次序排序：

♣2<♣3<……<♣A<♦2<♦3<……<♦A<♥2<♥3<……<♥A<♠2<♠3<……<♠A

两个关键字：花色（ ♣<♦<♥<♠ ）
面值（2<3<……<A）

并且“花色”地位高于“面值”

假设文件F有n个记录，$F=(R_0, R_1, \ldots R_{n-1})$
且每个记录R_i的排序码中含有d个部分$(k_i^0, k_i^1, \ldots, k_i^{d-1})$，
文件F对排序码$(k^0,k^1,\ldots,k^{d-1})$有序是指：
文件中任意两个记录R_i和$R_j$$(0\le i\le j\le n-1)$满足
字典次序有序关系$(k_i^0, k_i^1, \ldots, k_i^{d-1}) < (k_j^0, k_j^1, \ldots, k_j^{d-1})$，
其中k^0称为最高位排序码，k^{d-1}称为最低位排序码

第一种是先对最高位排序码k^0排序，称为高位优先法
第二种是先对最低位排序码k^{d-1}排序，称为低位优先法

基数排序步骤

- 链式基数排序(低位优先)
  - 借助“分配”和“收集”对单逻辑关键字进行排序的方法

◆基数r = 10 的自然数

◆设置10个队列，f[i]和e[i]分别为第i个队列的头指针和尾指针

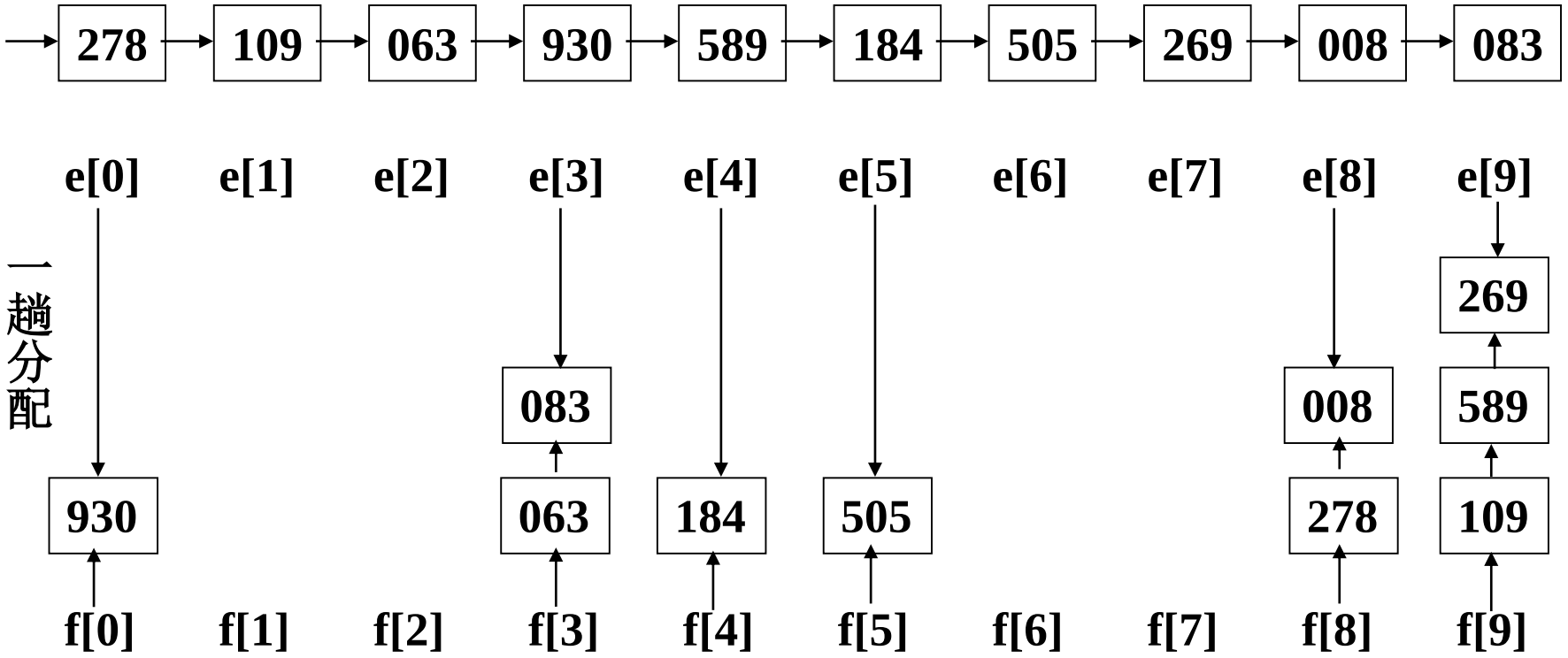
第一趟分配对最低位关键字（个位）进行，改变记录的指针值，将链表中记录分配至10个链队列中，每个队列记录的关键字的个位相同

◆第一趟收集是改变所有非空队列的队尾记录的指针域，令其指向下一个非空队列的队头记录，重新将10个队列链成一个链表

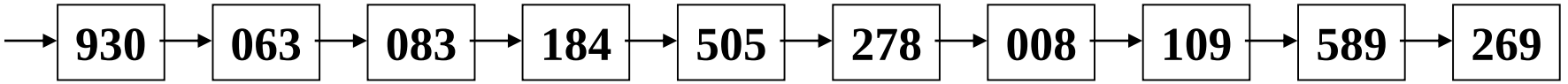
◆重复上述两步，进行第二趟、第三趟分配和收集，分别对十位、百位进行，最后得到一个有序序列

基数排序的演示

初始状态

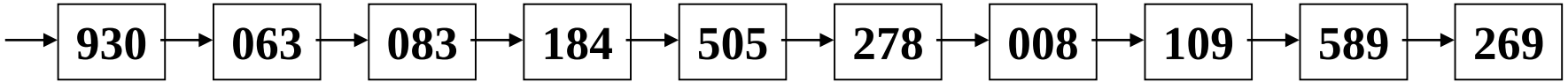


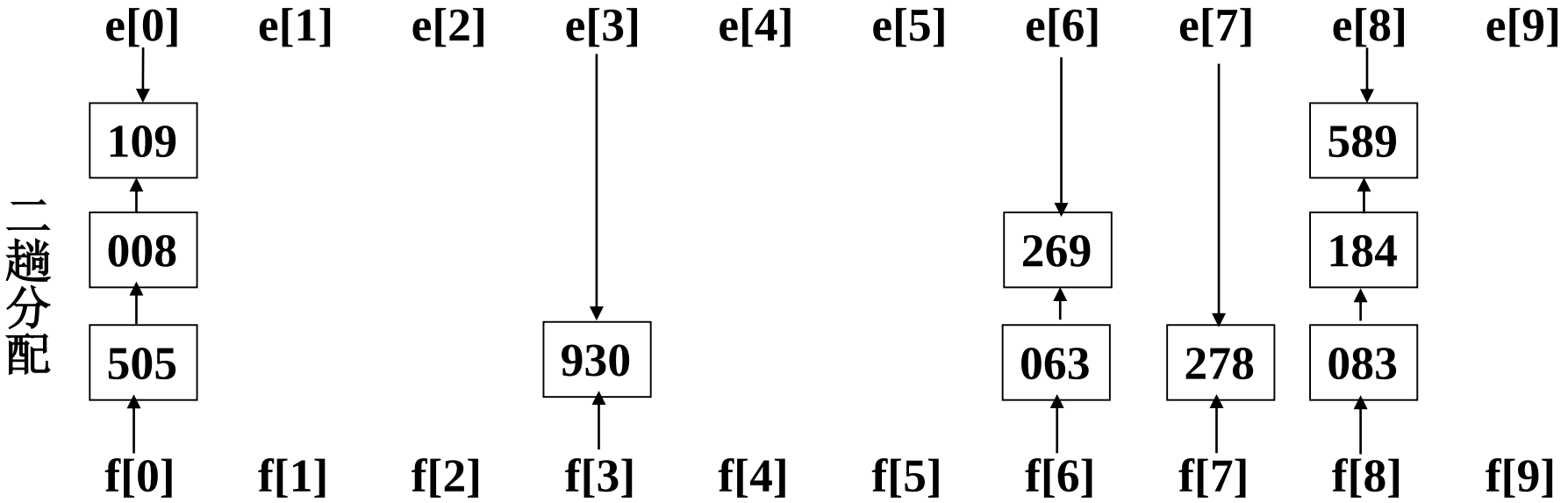
一趟收集：



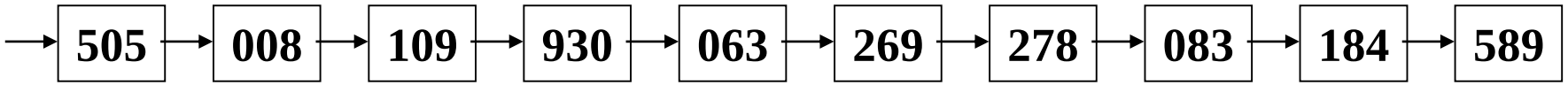
基数排序的演示

一趟收集：



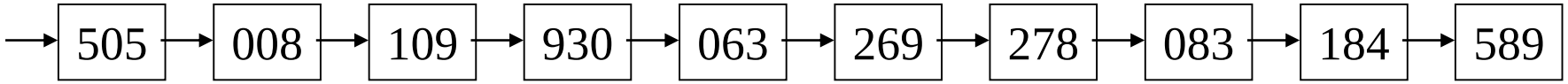


二趟收集：

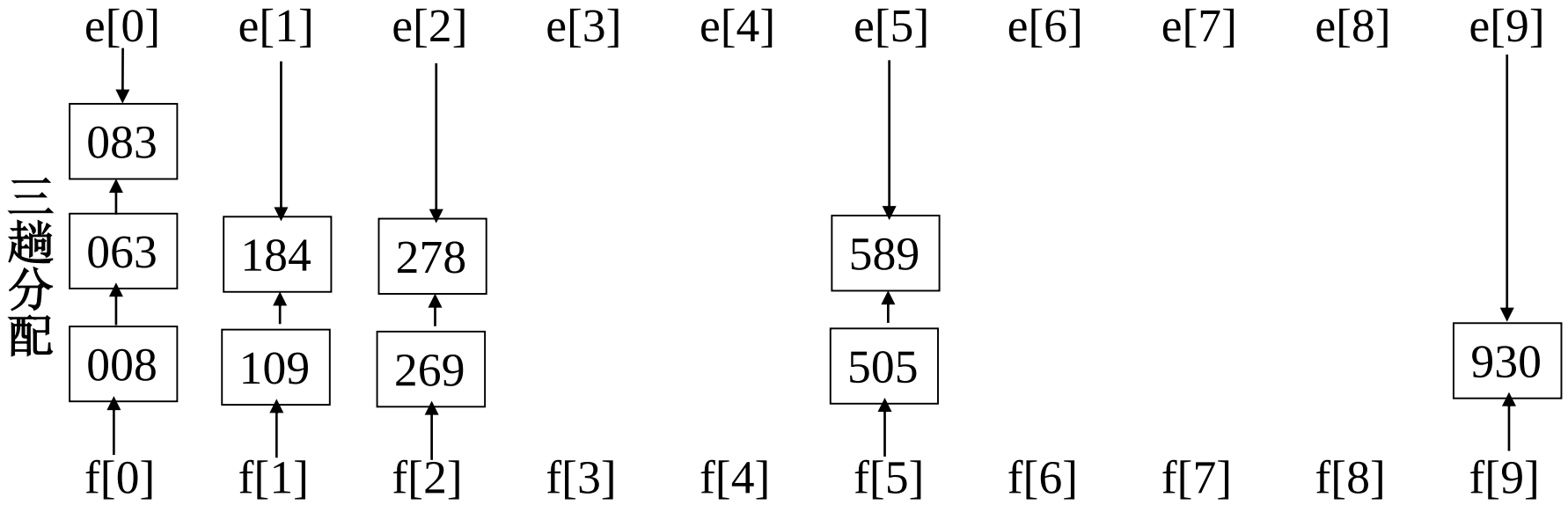


基数排序的演示

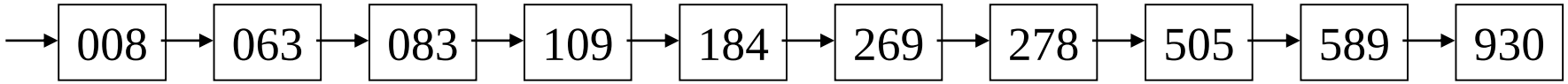
二趟收集



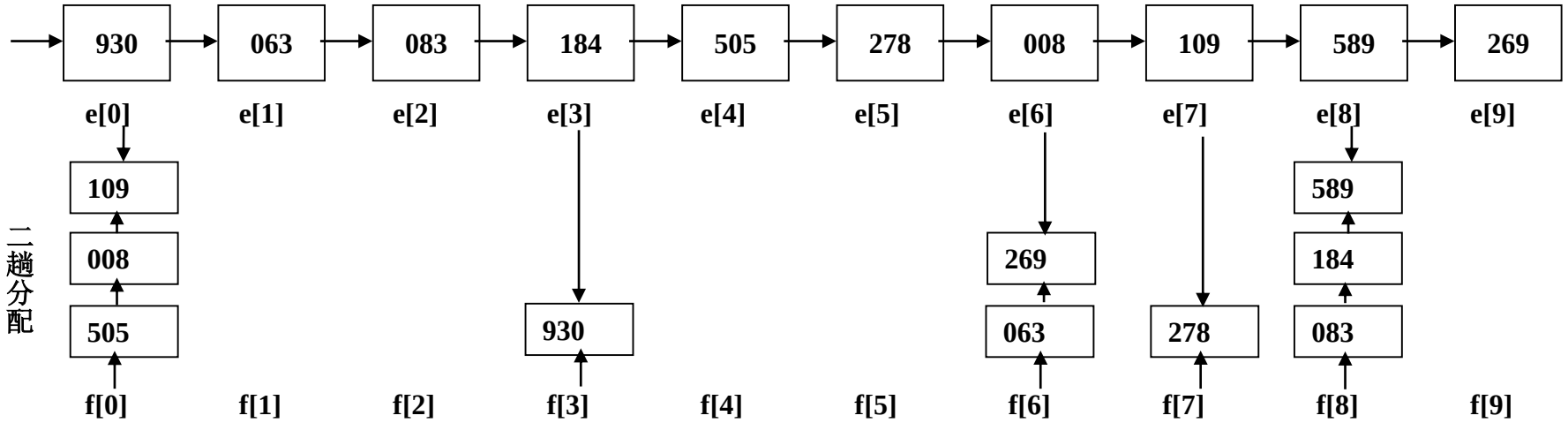
三趟分配



三趟收集：



```
1  struct Node;    /* 单链表结点类型 */
2  typedef struct Node RadixNode;
3  struct Node
4  {     KeyType key[D];
5        DataType info;
6        RadixNode *next;
7  };
8
9  typedef RadixNode * RadixList;
10 typedef struct QueueNode
11 {    RadixNode *f;          /* 队列的头指针 */
12       RadixNode *e;          /* 队列的尾指针 */
13 }Queue;
```




```
15  void radixSort(RadixList * plist, int d, int r)
16  {
17       int i,j,k; RadixNode *p, *head; head=(*plist)->next;
18       for(j=d-1; j>=0; j--)  /* 进行d次分配和收集*/
19       {
20            p=head;
21            for(i=0; i<r; i++)
22            { queue[i].f=NULL;  queue[i].e=NULL;}   /* 清队列 */
23            for(p=head; p!=NULL; p=p->next)
24            {   k=p->key[j];                          /* 按排序码的第j个分量进行分配*/
25                 if(queue[k].f==NULL) queue[k].f=p; /* 若第k个堆为空，则当前记录为队头*/
26                 else  (queue[k].e)->next=p;   /* 否则当前记录链接到第k队的队尾*/
27                 queue[k].e=p;
28            }
29            for(i=0; queue[i].f==NULL);  i++);   /* 从r个队列中找出第一个非空的队列*/
30            p=queue[i].e;  head=queue[i].f;     /* head为收集链表的头指针*/
31            for(i++; i<r; i++)
32               if(queue[i].f!=NULL)
33                 {p->next=queue[i].f;  p=queue[i].e;} /* 收集非空队列 */
34            p->next=NULL;
35       }
36        (*plist)->next=head;
37  }
```

算法分析

时间复杂度：

时间耗费主要在修改指针上，一趟排序的时间为O(r+n)

总共要进行d趟排序，基数排序的时间复杂度是T(n)=O(d*(r+n))

当n较大、d较小，特别是记录的信息量较大时，基数排序非常有效

空间复杂度：

基数排序中，每个记录中增加了一个next字段，还增加了一个queue数组，故辅助空间为S(n)=O(n+r)

稳定性：基数排序是稳定的