本节主题：

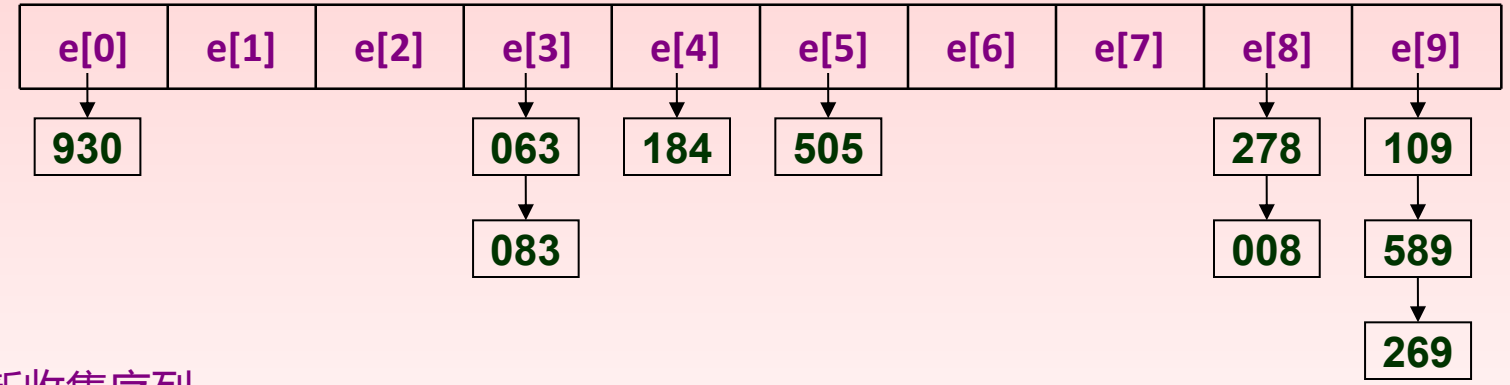
基数排序

基数排序过程(分配+收集)

- 问题：排序下面的序列

278	109	063	930	589	184	505	269	008	083

- 按个位数分配



- 重新收集序列

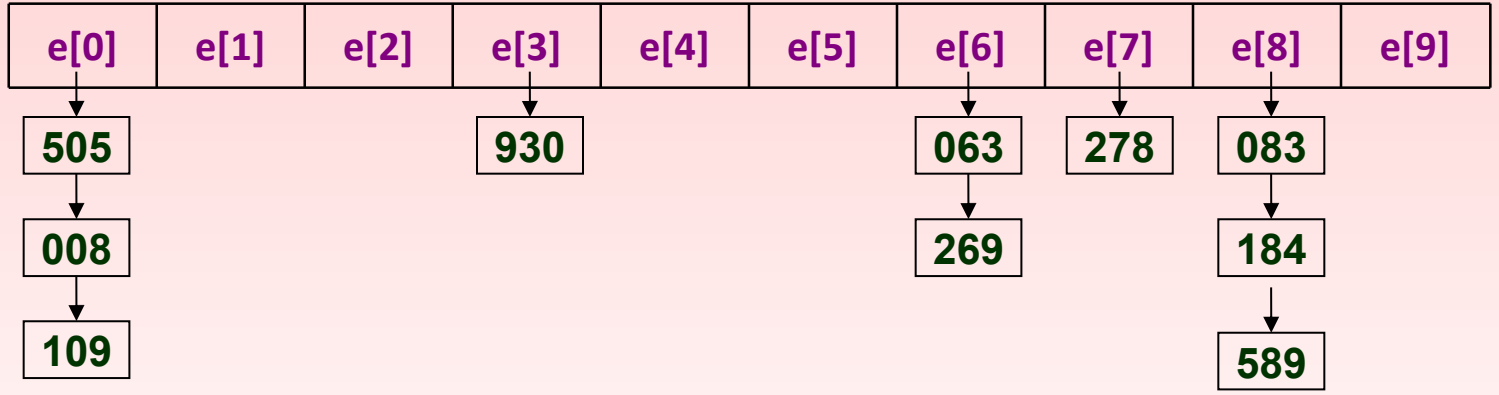
930	063	083	184	505	278	008	109	589	269

基数排序过程(续一)

- 按个位数分配、收集后

930	063	083	184	505	278	008	109	589	269

- 按十位数分配



- 重新收集序列

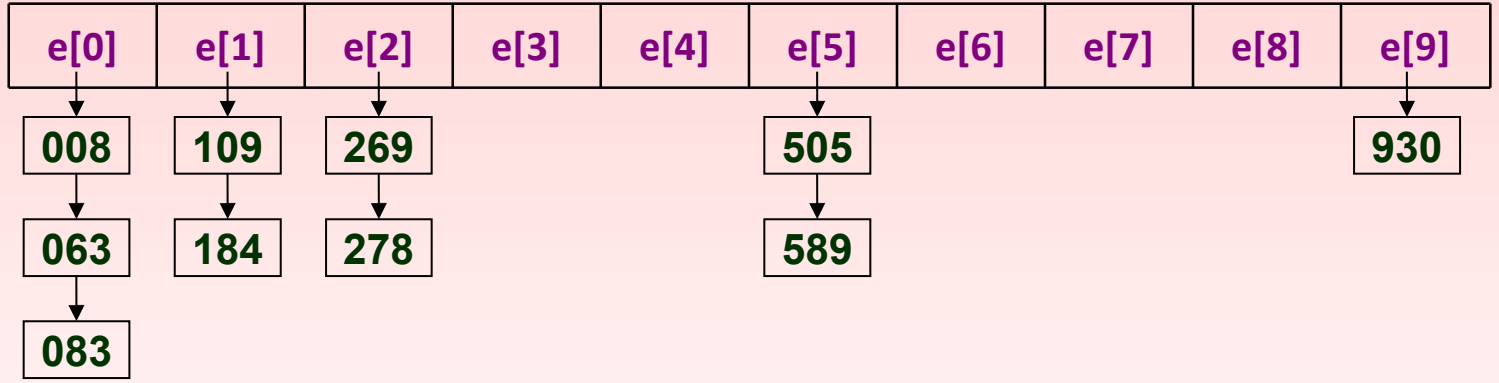
505	008	109	930	063	269	278	083	184	589

基数排序过程(续二)

- 按个、十位数分配、收集后

505	008	109	930	063	269	278	083	184	589

- 按百位数分配



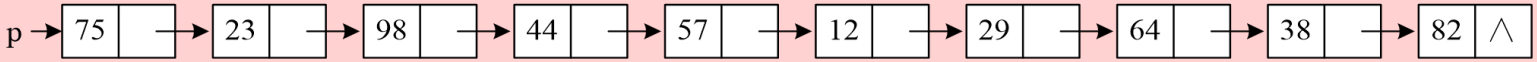
- 收集得到最终有序序列!!!

008	063	083	109	184	269	278	505	589	930

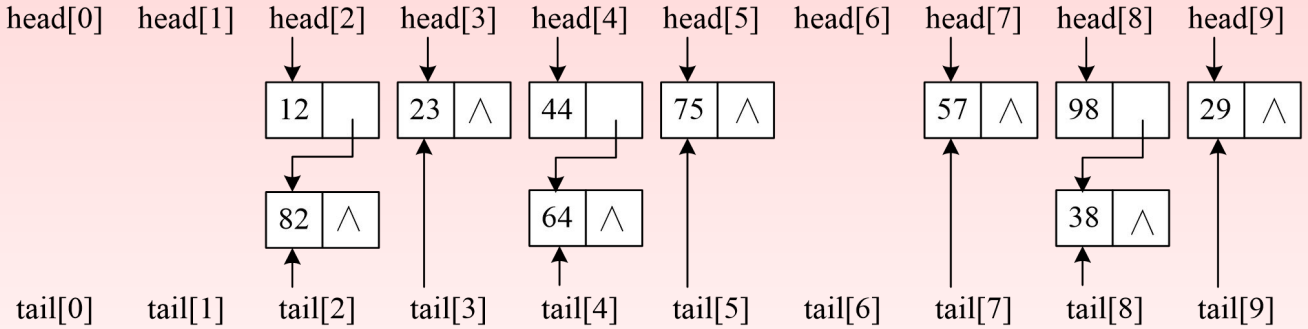
不比较的排序——基数排序原理

- 问题
  - 记录R[i]的关键字R[i].key是由d位数字组成，即$k^{d-1}k^{d-2}...k^0$，每一个数字表示关键字的一位，每一位的值都在$0 \le k^i < r$范围内（r=10为基数，表示用十进制）
- 两种基数排序
  - 最低位优先（LSD）
  - 最高位优先（MSD）
- 通过“分配”和“收集”过程来实现排序(以最低位优先为例）
  - 先按最低位的值对记录进行分配、收集
  - 在前一趟的基础上，再对高位的值分配和收集，直至最高位，则完成了基数排序的整个过程。
- 基数排序是一种借助于多关键字排序的思想对单关键字排序的方法。

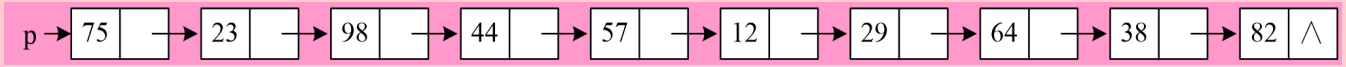
基数排序数据的存储结构

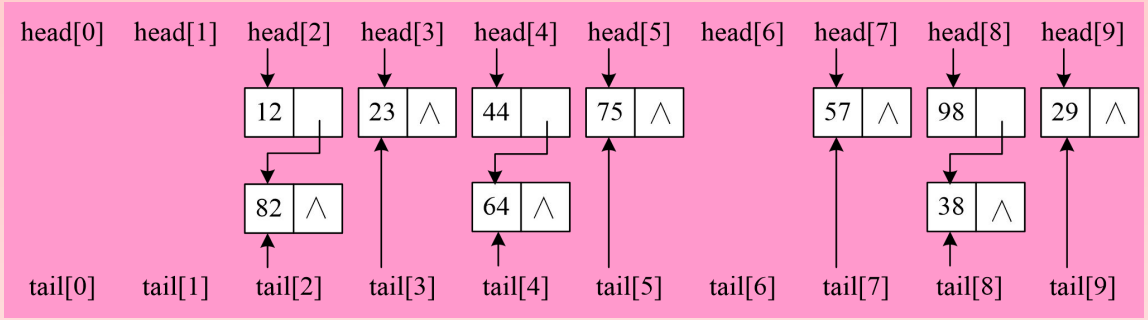



```
#define MAXE 20          //线性表中最多元素个数
#define MAXR 10          //基数的最大取值
#define MAXD 8           //关键字位数的最大取值
//排序数据节点类型
typedef struct node
{
  char data[MAXD];
  struct node *next;
} RecType1;
RecType1 *p;
//用于分配和收集的队列
RecType1 *head[MAXR],*tail[MAXR]
```



基数排序实现



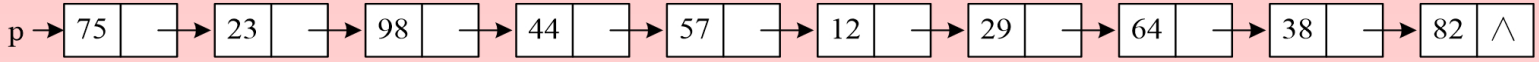



```
for (j=0; j<r; j++)
   head[j]=tail[j]=NULL
```

```
while (p!=NULL)
{
   k=p->data[i]-'0';
   if (head[k]==NULL)
   {
      head[k]=p;
      tail[k]=p;
   }
   else
   {
      tail[k]->next=p;
      tail[k]=p;
   }
   p=p->next;
}
```

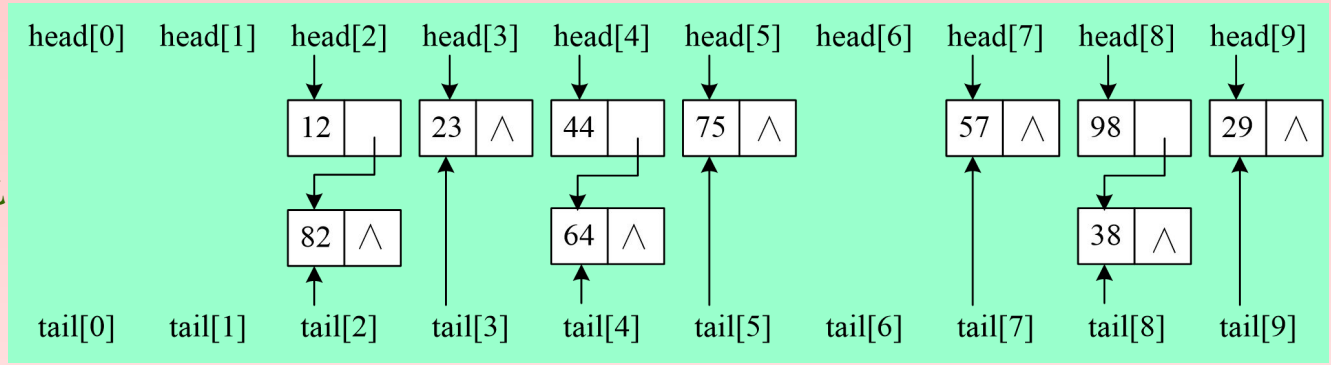
```
void RadixSort(RecType *&p,int r,int d)
{
   RecType *head[MAXR],*tail[MAXR],*t;
   int i,j,k;
   for (i=0; i<=d-1; i++)  //从低位到高位
   {
      //初始化各链队首、尾指针

      //分配每一个节点

      //再用p来收集所有节点

   }
}
```

```
p=NULL;
for (j=0; j<r; j++)
   if (head[j]!=NULL)
   {
      if (p==NULL)
      {
         p=head[j];
         t=tail[j];
      }
      else
      {
         t->next=head[j];
         t=tail[j];
      }
   }
t->next=NULL;
```



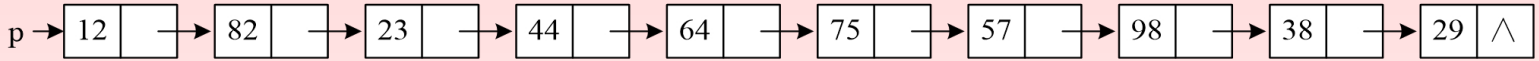
d=0

分配



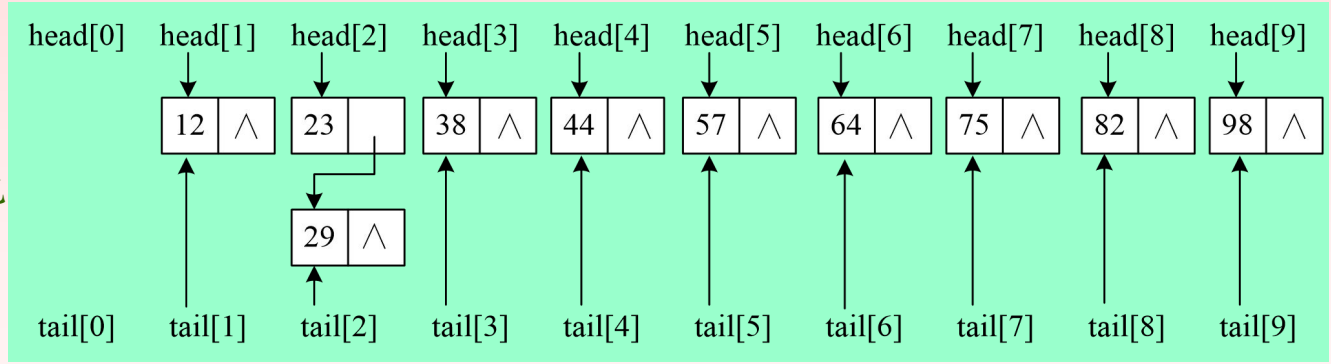
基数排序
示例

收集



d=1

分配



时间复杂度: O(*d*(*n*+*r*))
分配为O(n)
收集为O(r)
“分配-收集”*d* 趟

收集