10.6 基数排序

1、 基数排序的概念

基数r：对于二进制数r为2，对于十进制数r为10。

示例：2 3 9 ← r=10

百位　十位　个位

一般地

记录$R[j]$的关键字$R[j].key$ ⇨ $k_j^{d-1}\ k_j^{d-2}\ \cdots\ k_j^{1}\ k_j^{0}$

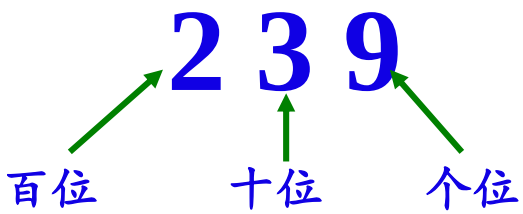
最高位

d位数字或字符组成
每一位的值都在0～r−1范围内，
其中r称为基数

最低位

2、基数排序的分类

基数排序有两种：最低位优先（LSD）和最高位优先（MSD）。



- 最低位优先：从个位 ⇨ 百位
- 最高位优先：从百位 ⇨ 个位

选择哪种基数排序，需要根据数据的特点来定。例如，对整数序列递增排序，选择最低位优先，越重要的位越在后面排序。

最低位优先排序过程如下：

对$i = 0, 1, \cdots, d-1$，依次做一次“分配”和“收集”（使用r个队列$Q_0, Q_1, \cdots, Q_{r-1}$。）。

- 分配：开始时，把$Q_0, Q_1, \cdots, Q_{r-1}$各个队列置成空队列，然后依次考察线性表中的每一个节点a_j（$j=0,1,\cdots,n-1$），如果a_j的关键字$k_j^i=k$，就把a_j放进Q_k队列中。

- 收集：按$Q_0, Q_1, \cdots, Q_{r-1}$顺序把各个队列中的节点首尾相接，得到新的节点序列，从而组成新的线性表。

由于数据需要放入队列，又要从队列取出来，需要大量元素移动。所以排序数据和队列均采用链表存储更好。

例如（369,367,167,239,237,138,230,139），采用基数排序

p →369 →367 →167 →239 →237 →138 →230 →139

建立10个队列，*f*为队头，*r*为队尾

❶ 进行第1次分配：按个位

f[0] →230 ← *r*[0]

f[7] →367 →167 →237 ← *r*[7]

f[8] →138 ← *r*[8]

f[9] →369 →239 →139 ← *r*[9]

进行第1次收集

- 分配时是按一个一个元素进行的
- 收集时是按一个一个队列进行的

p →230 →367 →167 →237 →138 →369 →239 →139

第1趟排序完毕

p →230 →367 →167 →237 →138 →369 →239 →139

❷ 进行第2次分配：按拾位

f[3] →230 →237 →138 →239 →139 ← *r*[3]

f[6] →367 →167 →369 ← *r*[6]

进行第2次收集

p →230 →237 →138 →239 →139 →367 →167 →369

第2趟排序完毕

p →230 →237 →138 →239 →139 →367 →167 →369

❸ 进行第3次分配：按百位

f[1] →138 →139 →167 ← *r*[1]

f[2] →230 →237 →239 ← *r*[2]

f[3] →367 →369 ← *r*[3]

进行第3次收集

p →138 →139 →167 →230 →237 →239 →367 →369

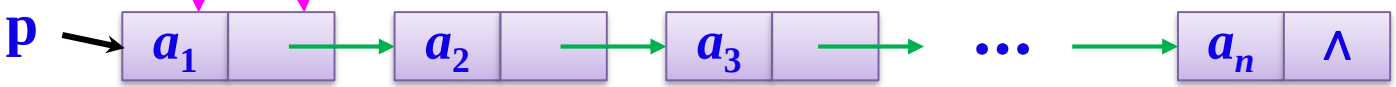
第3趟排序完毕

结论

基数排序是通过“分配”和“收集”过程来实现排序，不需要关键字的比较。

3、 基数排序算法

```
#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
typedef struct node
{   char data[MAXD];     //记录的关键字定义的字符串
    struct node *next;
} RecType1;              //单链表中每个节点的类型
```



基数排序数据的存储结构

```
void RadixSort(RecType1 *&p,int r,int d)
//p为待排序序列链表指针,r为基数,d为关键字位数
{    RecType1 *head[MAXR], *tail[MAXR], *t;  //定义各链队的首尾指针
     int i, j, k;
     for (i=0;i<d;i--)                       //从低位到高位做d趟排序
     {       for (j=0;j<r;j++)               //初始化各链队首、尾指针
                  head[j]=tail[j]=NULL;
```

分配

```
        while (p!=NULL)                      //对于原链表中每个节点循环
        {      k=p->data[i]-'0';             //找第k个链队
               if (head[k]==NULL)            //进行分配,即采用尾插法建立单链表
               {     head[k]=p;  tail[k]=p;  }
               else
               {    tail[k]->next=p;  tail[k]=p;   }
               p=p->next;                    //取下一个待排序的节点
        }
```

```
    p=NULL;
    for (j=0;j<r;j++)          //对于每一个链队循环进行收集
        if (head[j]!=NULL)
        {       if (p==NULL)
                {       p=head[j];
                        t=tail[j];
                }
                else
                {     t->next=head[j];
                      t=tail[j];
                }
         }
      t->next=NULL;          //最后一个节点的next域置NULL
    }
}
```

收集

排序完成后，*p*指向的是一个有序单链表。

注意：C/C++将数值转换为字符串：

2 1 0

1 2 3　　应该从3 ⇨ 1，即低位到高位

“3 2 1”　　为了从3 ⇨ 1，应该高位到低位

2 1 0

4、基数排序算法分析

基数排序的时间复杂度为$O(d(n+r))$

其中：分配为$O(n)$

收集为$O(r)$（r为“基数”）

d为“分配-收集”的趟数

基数排序的空间复杂度为$O(r)$

【例10-10】以下排序方法中，_____不需要进行关键字的比较。

A.快速排序　　B.归并排序

C.基数排序　　D.堆排序

思考题

基数排序中为什么不需要进行关键字的比较？

——本讲完——