阅读：8,636　　作者：解学武

基数排序及其C语言实现

基数排序不同于之前所介绍的各类排序，前边介绍到的排序方法或多或少的是通过使用比较和移动记录来实现排序，而基数排序的实现不需要进行对关键字的比较，只需要对关键字进行“分配”与“收集”两种操作即可完成。

例如对无序表 {50，123，543，187，49，30，0，2，11，100} 进行基数排序，由于每个关键字都是整数数值，且其中的最大值由个位、十位和百位构成，每个数位上的数字从 0 到 9，首先将各个关键字按照其个位数字的不同进行分配分配表如下图所示：

个位	
0	50、30、0、100
1	11
2	2
3	123、543
4	
5	
6	
7	187
8	
9	49

通过按照各关键字的个位数进行分配，按照顺序收集得到的序列变为：{50，30，0，100，11，2，123，543，187，49} 。在该序列表的基础上，再按照各关键字的十位对各关键字进行分配，得到的分配表如下图所示：

十位	
0	0、100、2
1	11
2	123
3	30
4	543、49
5	50
6	
7	
8	187
9	

由上表顺序收集得到的记录表为：`{0、100、2、11、123、30、543、49、50、187}`。在该无序表的基础上，依次将表中的记录按照其关键字的百位进行分配，得到的分配如下图所示：

百位	
0	0、2、11、30、49、50
1	100、123、187
2	
3	
4	
5	543
6	
7	
8	
9	

最终通过三次分配与收集，最终得到的就是一个排好序的有序表：
`{0、2、11、30、49、50、100、123、187、543}`。

例子中是按照个位-十位-百位的顺序进行基数排序，此种方式是从最低位开始排序，所以被称为最低位优先法（简称“LSD法”）。

同样还可以按照百位-十位-各位的顺序进行排序，称为最高位优先法（简称“MSD法”），使用该方式进行排序同最低位优先法不同的是：当无序表中的关键字进行分配后，相当于进入了多个子序列，后序的排序工作分别在

各个子序列中进行（最低位优先法每次分配与收集都是相对于整个序列表而言的）。

例如还是对 {50，123，543，187，49，30，0，2，11，100} 使用最高位优先法进行排序，首先按照百位的不同进行分配，得到的分配表为：

	百位	
序列1：	0	50、49、30、0、2、11
序列2：	1	123、187、100
	2	
	3	
	4	
序列3：	5	543
	6	
	7	
	8	
	9	

由上图所示，整个无序表被分为了 3 个子序列，序列 1 和序列 2 中含有多个关键字，序列 3 中只包含了一个关键字，最高位优先法完成排序的标准为：直到每个子序列中只有一个关键字为止，所以需要分别对两个子序列进行再分配，各自的分配表如下图所示：

十位	
0	0、2
1	11
2	
3	30
4	49
5	50
6	
7	
8	
9	

（序列 1）

十位	
0	100
1	
2	123
3	
4	
5	
6	
7	
8	187
9	

（序列 2）

上表中，序列 1 中还有含有两个关键字的子序列，所以还需要根据个位进行分配，最终按照各子序列的顺序同样会得到一个有序表。

基数排序的具体实现（采用LSD法）

基数排序较宜使用链式存储结构作为存储结构，相比于顺序存储结构更节省排序过程中所需要的存储空间，称之为“链式基数排序”。

其具体的实现代码为：

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_NUM_OF_KEY 8//构成关键字的组成部分的最大个数
#define RADIX 10         //基数，例如关键字是数字，无疑由0~9组成，基数就是10；如果关键字是字符串（字母
#define MAX_SPACE 10000
//静态链表的结点结构
typedef struct{
    int data;//存储的关键字
    int keys[MAX_NUM_OF_KEY];//存储关键字的数组（此时是一位一位的存储在数组中）
    int next;//做指针用，用于是静态链表，所以每个结点中存储着下一个结点所在数组中的位置下标
}SLCell;
//静态链表结构
typedef struct{
    SLCell r[MAX_SPACE];//静态链表的可利用空间，其中r[0]为头结点
    int keynum;//当前所有的关键字中最大的关键字所包含的位数，例如最大关键字是百，说明所有keynum=3
    int recnum;//静态链表的当前长度
} SLList;

typedef int  ArrType[RADIX];//指针数组，用于记录各子序列的首尾位置
//排序的分配算法，i表示按照分配的位次（是个位，十位还是百位），f表示各子序列中第一个记录和最后一个记录的位
void Distribute(SLCell *r,int i,ArrType f,ArrType e){
    //初始化指针数组
    for (int j=0; j<RADIX; j++) {
        f[j]=0;
    }
    //遍历各个关键字
    for (int p=r[0].next; p; p=r[p].next) {
        int j=r[p].keys[i];//取出每个关键字的第 i 位，由于采用的是最低位优先法，所以，例如，第 1 位
        if (!f[j]) {//如果只想该位数字的指针不存在，说明这是第一个关键字，直接记录该关键字的位置即可
            f[j]=p;
        }else{//如果存在，说明之前已经有同该关键字相同位的记录，所以需要将其进行连接，将最后一个相同的关
            r[e[j]].next=p;
        }
        e[j]=p;//移动尾指针的位置
    }
}
//基数排序的收集算法，即重新设置链表中各结点的尾指针
void Collect(SLCell *r,int i,ArrType f,ArrType e){
    int j;
```

```
    //从 0 开始遍历，查找头指针不为空的情况，为空表明该位没有该类型的关键字
    for (j=0;!f[j]; j++);
    r[0].next=f[j];//重新设置头结点
    int t=e[j];//找到尾指针的位置
    while (j<RADIX) {
        for (j++; j<RADIX; j++) {
            if (f[j]) {
                r[t].next=f[j];//重新连接下一个位次的首个关键字
                t=e[j];//t代表下一个位次的尾指针所在的位置
            }
        }
    }
    r[t].next=0;//0表示链表结束
}
void RadixSort(SLList *L){
    ArrType f,e;
    //根据记录中所包含的关键字的最大位数，一位一位的进行分配与收集
    for (int i=0; i<L->keynum; i++) {
        //秉着先分配后收集的顺序
        Distribute(L->r, i, f, e);
        Collect(L->r, i, f, e);
    }
}
//创建静态链表
void creatList(SLList * L){
    int key,i=1,j;
    scanf("%d",&key);
    while (key!=-1) {
        L->r[i].data=key;
        for (j=0; j<=L->keynum; j++) {
            L->r[i].keys[j]=key%10;
            key/=10;
        }
        L->r[i-1].next=i;
        i++;
        scanf("%d",&key);
    }
    L->recnum=i-1;
    L->r[L->recnum].next=0;
}
//输出静态链表
void print(SLList*L){
    for (int p=L->r[0].next; p; p=L->r[p].next) {
        printf("%d ",L->r[p].data);
    }
    printf("\n");
}
```

```
87.  int main(int argc, const char * argv[]) {
88.      SLList *L=(SLList*)malloc(sizeof(SLList));
89.      L->keynum=3;
90.      L->recnum=0;
91.      creatList(L);//创建静态链表
92.      printf("排序前：");
93.      print(L);
94.
95.      RadixSort(L);//对静态链表中的记录进行基数排序
96.
97.      printf("排序后：");
98.      print(L);
99.      return 0;
100.  }
```

运行结果为：

```
50
123
543
187
49
30
0
2
11
100
-1
排序前：50 123 543 187 49 30 0 2 11 100
排序后：0 2 11 30 49 50 100 123 187 543
```