10.2.3 希尔排序

希尔排序(Shell's Sort)又称"缩小增量排序"(Diminishing Increment Sort),它也是一种属插入排序类的方法,但在时间效率上较前述几种排序方法有较大的改进。

从对直接插入排序的分析得知,其算法时间复杂度为 $O(n^2)$,但是,若待排记录序列为"正序"时,其时间复杂度可提高至 $O(n)$。由此可设想,若待排记录序列按关键字"基本有序",即序列中具有下列特性

$$L.r[i].key < \max_{1 \le j < i} |L.r[j].key| \tag{10-7}$$

的记录较少时,直接插入排序的效率就可大大提高,从另一方面来看,由于直接插入排序算法简单,则在 n 值很小时效率也比较高。希尔排序正是从这两点分析出发对直接插入排序进行改进得到的一种插入排序方法。

它的基本思想是:先将整个待排记录序列分割成为若干子序列分别进行直接插入排序,待整个序列中的记录"基本有序"时,再对全体记录进行一次直接插入排序。

仍以式(10-4)中的关键字为例,先看一下希尔排序的过程。初始关键字序列如图10.5的第1行所示。首先将该序列分成五个子序列$\{R_1, R_6\}$,$\{R_2, R_7\}$,…,$\{R_5, R_{10}\}$,如图10.5的第2行至第6行所示,分别对每个子序列进行直接插入排序,排序结果如图10.5的第7行所示,从第1行的初始序列得到第7行的序列的过程称为一趟希尔排序。然后进行第二趟希尔排序,即分别对下列三个子序列:$\{R_1, R_4, R_7, R_{10}\}$,$\{R_2, R_5, R_8\}$和$\{R_3, R_6, R_9\}$进行直接插入排序,其结果如图10.5的第11行所示,最后对整个序列进行一趟直接插入排序。至此,希尔排序结束,整个序列的记录已按关键字非递减有序排列。

从上述排序过程可见,希尔排序的一个特点是:子序列的构成不是简单地"逐段分割",而是将相隔某个"增量"的记录组成一个子序列。如上例中,第一趟排序时的增量为5,第二趟排序时的增量为3,由于在前两趟的插入排序中记录的关键字是和同一子序列中的前一个记录的关键字进行比较,因此关键字较小的记录就不是一步一步地往前挪动,而是跳跃式地往前移,从而使得在进行最后一趟增量为1的插入排序时,序列已基本有序,只要作记录的少量比较和移动即可完成排序,因此希尔排序的时间复杂度较直接插入排序低。下面用算法语言描述上述希尔排序的过程,为此先将算法10.1改写成如算法10.4所示的一般形式。希尔排序算法如算法10.5所示。

```
void ShellInsert ( SqList &L, int dk ) {
  // 对顺序表 L 作一趟希尔插入排序。本算法是和一趟直接插入排序相比,作了以下修改:
```

```
    //      1. 前后记录位置的增量是 dk,而不是 1;
    //      2. r[0]只是暂存单元,不是哨兵。当 j<=0 时,插入位置已找到。
    for ( i = dk+1; i<= L.length; ++i )
        if LT(L.r[i].key, L.r[i-dk].key) {   // 需将 L.r[i]插入有序增量子表
          L.r[0] = L.r[i];                   // 暂存在 L.r[0]
          for (j = i-dk;  j>0 && LT(L.r[0].key, L.r[j].key);  j-= dk)
             L.r[j+dk] = L.r[j];             // 记录后移,查找插入位置
          L.r[j+dk] = L.r[0];                // 插入
        }
}// ShellInsert
```

算法 10.4

```
void ShellSort (SqList &L, int dlta[], int t) {
   // 按增量序列 dlta[0..t-1]对顺序表 L 作希尔排序。
   for (k=0; k<t; ++t)
      ShellInsert(L, dlta[k]);            // 一趟增量为 dlta[k]的插入排序
}// ShellSort
```

算法 10.5

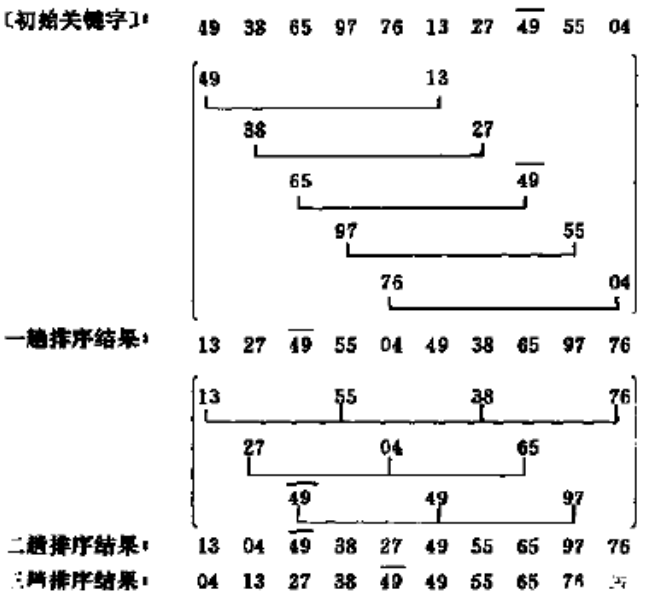

图 10.5 希尔排序示例

希尔排序的分析是一个复杂的问题,因为它的时间是所取"增量"序列的函数,这涉及一些数学上尚未解决的难题。因此,到目前为止尚未有人求得一种最好的增量序列,但大量的研究已得出一些局部的结论。如有人指出,当增量序列为 $dlta[k]=2^{t-k+1}-1$ 时,希尔排序的时间复杂度为 $O(n^{3/2})$,其中 t 为排序趟数,$1\leqslant k\leqslant t\leqslant \lfloor \log_2(n+1)\rfloor$。还有人在大量的实验基础上推出:当 N 在某个特定范围内,希尔排序所需的比较和移动次

```
#include<stdio.h>
#include<math.h>

#define MAXNUM 11
```

```
int main()
{
    void shellSort(int array[],int n,int t);//t为排序趟数
    int array[MAXNUM],i;
    for(i=0;i<MAXNUM;i++)
        scanf("%d",&array[i]);
    shellSort(array,MAXNUM,(int)(log(MAXNUM+1)/log(2)));//排序趟数应为log2(n+1)
的整数部分
    for(i=0;i<MAXNUM;i++)
        printf("%d ",array[i]);
    printf("\n");
    return 0;
}

//根据当前增量进行插入排序
void shellInsert(int array[],int n,int dk)
{
    int i,j,temp;
    for(i=dk;i<n;i++)//分别向每组的有序区域插入
    {
        temp=array[i];
        for(j=i-dk;(j>=i%dk)&&array[j]>temp;j-=dk)//比较与记录后移同时进行
            array[j+dk]=array[j];
        if(j!=i-dk)
            array[j+dk]=temp;//插入
    }
    for(i=0;i<MAXNUM;i++)
        printf("%d ",array[i]);
    printf("\n");
}

//计算Hibbard增量
int dkHibbard(int t,int k)
{
    return (int)(pow(2,t-k+1)-1);
```

```
}

//希尔排序
void shellSort(int array[],int n,int t)
{
    void shellInsert(int array[],int n,int dk);
    int i;
    for(i=1;i<=t;i++)
        shellInsert(array,n,dkHibbard(t,i));
}

//此写法便于理解，实际应用时应将上述三个函数写成一个函数。
```