Data Structure

Chapter 3： Sorting

Main

3.1 Introduction

3.2 Simple Insertion Sort (直接插入排序)

3.3 Shellsort (希尔排序)

3.4 Cardinality Sort (基数排序)

3.4.1 Multiple keyword sorting (多关键字排序)

3.4.2 Radix Sort : using Sq List (基数排序)

Bucket sort : using Linked List (桶式排序)

3.1 Introduction

- Record （记录）： Multiple data items
- Key (关键字)： *The items* used to sort records
- RecordType：

```
typedef  struct {
    KeyType  key;           // 关键字项
    ……                     // 其它数据项
} RecordType, RcdType;     // 记录类型
```

3.1.1 Concepts

- Example：

  （175, 85, 260, 63, 412, 504, 840, 518, 630, 950）

  Sorting：

  （63, 85, 175, 260, 412, 504, 518, 630, 840, 950）

- The important critical properties of sorting algorithm:

  （排序算法的两大关键步骤）

  - **Number of comparisons to be made (比较)**
  - **Number of data movements (移动)**

3.1.1 Concepts

◆The objectives of sorting algorithm:

(1) Minimize the number of movements of data

(2) Movements of data from secondary storage to main memory in large blocks

(3) Retaining all the data items in the main memory

◆The factors for choosing a sorting methods:

(1) Programming time

(2) Execution time of the program

(3) Memory or auxiliary storage space needed for programming environment.

RcdSqList

Elem **saved in RcdSqList:**

```
typedef  struct {
    KeyType  key;              // 关键字项
    ……                         // 其它数据项
} RecordType, RcdType;         // 记录类型
```

***The RcdSqList*:**

```
typedef struct {
    RcdType *rcd;      // 存储空间基址
    int length;        // 当前长度
    int size;          // 存储容量
} RcdSqList;  // 记录的顺序表
```

注意：SqList 的0号位闲置或用作*sentry post*(哨位)

3.1.1 Concepts

◆SORTing: the arrangement of a set of DATA in some order *Ascending* 升序 or *Descending*降序

◆Two categories sorting:

(1) Internal Sorting：

Sorting of data items in the Main memory

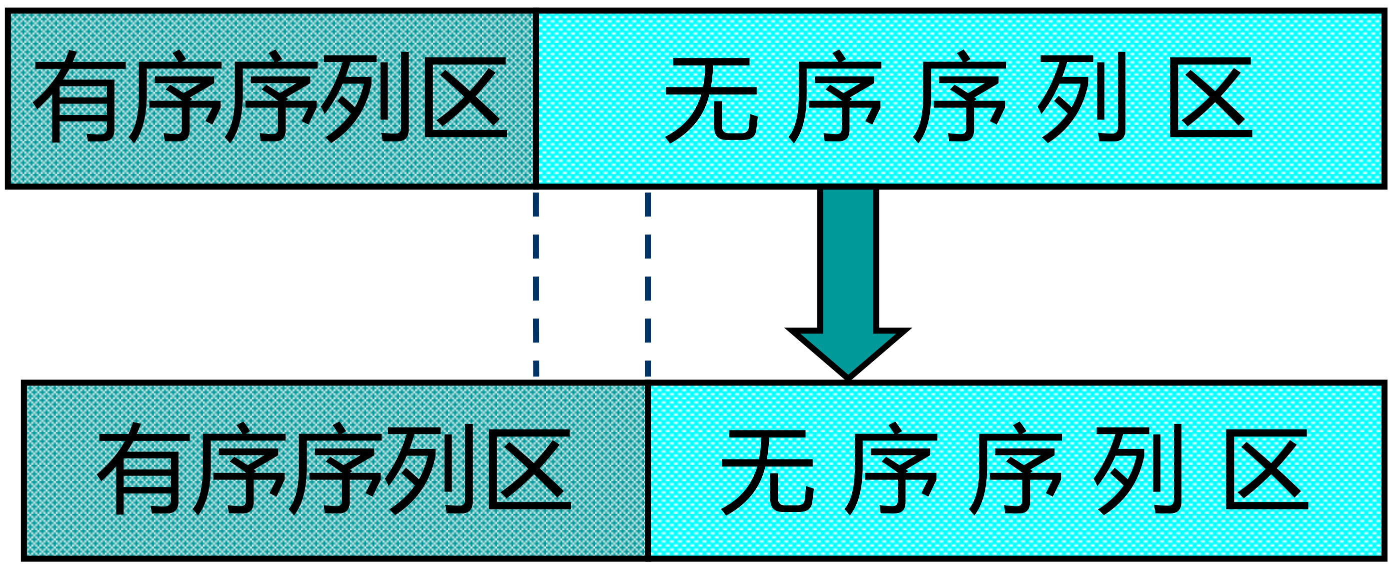
(2) External Sorting：

Sorting of data items partly in the main memory and partly in Auxiliary memory

Classification of internal sorting

- Five categories： Exchange sort, selection sort, insertion sort, merge sort and cardinality sort.

  (交换排序、选择排序、插入排序、归并排序和基数排序)



Classification of internal sorting

- Array storage (internal store)
  Files storage (external store)

(1) Insertion
(2) Selection
(3) Bubble（冒泡）
(4) Shell
(5) Quick
(6) Binary
(7) Heap
(8) Radix (基数)
(9) Merge

Sort (Internal Sorting)

Records r1,r2,…,rn
With KEY values k1,k2,…,kn,
respectively

SORTed

Records ri1,ri2,…,rin
with KEY values ki1≤ki2 ≤… ≤ kin,
respectively

Internal sorting

◆The objectives of sorting algorithm:

(1) Minimize the number of movements of data

(2) Movements of data from secondary storage to main memory in large blocks

(3) Retaining all the data items in the main memory

◆The factors for choosing a sorting methods:

(1) Programming time

(2) Execution time of the program

(3) Memory or auxiliary storage space needed for programming environment.

Complexity of Internal Sorting Algorithm

Algorithm	Worst (Time)	Average (Time)	Best (Time)	Auxiliary Space	Stab-ility
Insertion	$O(n^2)$	$O(n^2)$	$O(n-1)$	$O(1)$	Yes
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	No
Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	Yes
Shell	$\sim O(n)$	$O(n(\log_2 n)^2)$	$\sim O(n)$	$O(1)$	No
Quick	$O(n^2)$	$O(\log_2 n)$	$O(\log_2 n)$	$O(\log n)$	No
BinaryTree	$O(n^2)$	$O(n\log n)$	$O(n\log n)$		Yes
Heap	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(1)$	No
Radix	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n+r)$	Yes
Merge	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$	Yes

Stability 算法稳定性：

- Stability:

  Before Sorting：

  If Rec(Ki) > Rec(Kj) for KEY Ki=Kj

  After Sorting:

  Sort is STABLE: if Rec(Ki) > Rec(Kj)

  Sort is UNStable: if Rec(Ki) < Rec(Kj)

3.2 Simple Insertion Sort (直接插入排序)

- The key sequence: (56, 68, 25, 45, 90, 38, 10, 72)

初始序列：[56] **68** 25 45 90 38 10 72

第一趟排序结果：[56 68] 25 45 90 38 10 72

第二趟排序结果：**[**25 56 68**]** **45** 90 38 10 72

第三趟排序结果：[25 45 56 68] **90** 38 10 72

第四趟排序结果：**[25 45 56 68 90] 38 10 72**

第五趟排序结果：[25 38 45 56 68 90] 10 72

第六趟排序结果：[10 25 38 45 56 68 90] 72

第七趟排序结果：[10 25 38 45 56 68 72 90]

算法实现分析

- Find Insert Position

  j = 0; **do {** j++; **}** **while** (L.rcd[j].key<L.rcd[i+1].key); // 从前到后查找插入位置

- Move record to empty insertion position

  L.rcd[0] = L.rcd[i+1]; // 先将记录L.rcd[i+1]保存在空闲的0号单元

  k = i+1; **do {** k--; L.rcd[k+1] = L.rcd[k]; **}** **while**(k>j); // 从后到前移动记录

- Use Sentry and sentry post:

  L.rcd[0] = L.rcd[i+1];   // 置入哨位，作为哨兵

  j = i+1; **do{** j--;L.rcd[j+1] = L.rcd[j];**}**

  while(j>1 && L.rcd[0].key<L.rcd[j-1].key);

  // 从后到前查找并移动记录

- Set up sentry
- Find Insert Position. Move records to empty insertion position
- Insert a Record

Simple Insertion Sort

void InsertSort (RcdSqList &L)

0	1	j			i	i+1		
38	25	45	56	68	90	38	10	72

```
void InsertSort(RcdSqList &L) {  // 对顺序表L作直接插入排序。
    int i, j;
    for(i = 1; i<L.length; ++i)
        if(L.rcd[i+1].key<L.rcd[i].key) {  // 需将L.rcd[i+1]插入有序序列
            L.rcd[0] = L.rcd[i+1];  // 先将记录L.rcd[i+1]保存在空闲的0号单元
            j = i+1;
            do {  j--;  L.rcd[j+1] = L.rcd[j];  // 记录后移
            } while(L.rcd[0].key<L.rcd[j-1].key);  // 判断是否需要继续移动
            L.rcd[j] = L.rcd[0];   // 插入
        }
}
```

Complexity of Algorithm

- The time consumption of Insertion sort is mainly related to the frequency of records compared with key and moved.

- **Best case (data are in order)**

“比较”的次数：

$$\sum_{i=1}^{n-1} 1 = n-1$$

“移动”的次数：

0

- **Worst case (data are in reverse order)**

“比较”的次数：

$$\sum_{i=1}^{n-1} (i+1) = \frac{(n+2)(n-1)}{2}$$

“移动”的次数：

$$\sum_{i=1}^{n-1} (i+2) = \frac{(n+4)(n-1)}{2}$$

- **Best case ：** $O(n)$，**Worst case :** $O(n^2)$。

Average case (data are in random order)

- Only one record auxiliary space is required. Space complexity $O(1)$

3.3 Shell sort (希尔排序)

- **Shell sort : diminishing increment sort**
  The items are Divided into smaller Segments (e.g., k segment), then these segments are Sorted separately using Insertion sorting.

1	2	3	4	5	6	7	8	9	10
13	27	49	55	04	49	38	65	97	76

3.3 Steps of Shell sort

- Set up Increment ***d*** **, d**ividing the list into smaller segments
- Sorting separately these smaller Segments using Insertion sorting.
- Continuously decrease increment d, Repeat the above steps until *d* is reduced to 1.

The value sequence of increment *d* is called *increment sequence*。 sorting operation can be marked *d-sort* .

ShellSort is also named **diminishing increment sort (缩小增量排序)**

Example of Shell sort

- Original sequence: (49, 38, 65, 97, 76, 13, 27, <u>49</u>, 55, 04)
- *Increment sequence* : *d* = (5 , 3 , 1)

The first: $d_1 = 5$:

5-sort ⇨

1	2	3	4	5	6	7	8	9	10
(49,	38,	65,	97,	76,	13,	27,	<u>49</u>,	55,	04)

The result : （13, 27, 49, 55, 04, 49, 38, 65, 97, 76 ）

◆ The second : $d_2 = 3$

1 2 3 4 5 6 7 8 9 10

3-sort ⇨ （13, 27, 49, 55, 04, 49, 38, 65, 97, 76）

The result : （13, 04, 49, 38, 27, 49, 55, 65, 97, 76）

◆ The third : $d_3 = 1$ ***1-sort*** **:**

（ 04, 13, 27, 38, 49, 49, 55, 65, 76, 97）

一趟希尔排序

- 暂存待插入记录
- 按增量dk查找插入位置, 移动记录空出插入位置
- 插入记录

void ShellInsert(RcdSqList **&**L, **int** dk)

0	i-3			i			i+3			
38	**13**	**04**	**<u>49</u>**	**55**	**27**	**49**	**38**	**65**	**97**	**76**
				j						

```
void ShellInsert(RcdSqList &L, int dk) { // 对顺序表L作一趟希尔排序，增量为dk
  int i, j;
  for(i = 1; i<=L.length-dk; ++i)
    if(L.rcd[i+dk].key < L.rcd[i].key) { // 需将L.rcd[i+dk]插入有序序列
      L.rcd[0] = L.rcd[i+dk];  // 暂存在L.rcd[0]
      j = i+dk;
     do{ j-=dk; L.rcd[j+dk] = L.rcd[j]; // 记录后移
     }while(j-dk>0 && L.rcd[0].key<L.rcd[j-dk].key); // 判断是否需要继续移动
     L.rcd[j] = L.rcd[0]; // 插入
   }
}
```

希尔排序

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

Time Complexity ： $O(n^{1.5})$

Stability: Shell sort is unstable.