数据结构第7章习题

7-3

- 设文件(*R*1, *R*2, …, *Rn*)以单链表方式表示，指针变量*FIRST*指向表头结点，且表中的结点结构为：

KEY	*LINK*

- 其中*KEY*为该结点的关键词域，*LINK*为链接域。请给出这种线性表的直接插入排序算法，并要求算法的时间复杂度为$O(n^2)$，且算法是稳定的。

算法InsertSort(*first*. *first*) // *first*指向表头哨位结点

IS1[边界]

IF *link*(*first*)=NULL **THEN RETRUN**.

IF *link*(*link*(*first*))=NULL **THEN RETRUN**.

IS2[插入排序]

q1 ← *first*. *q* ←*link*(*q1*).

WHILE *q* ≠NULL **DO** (

if key(q1) <= key(q) **then** (

q1 ←q. q ←link(q1).)

p1 ← *first*. *p* ← link(first).

WHILE key(p) < key(q) **DO** (

p1 ←p. p ←link(p).

)

link(q1)←link(q). link(p1) ← q.

link(q) ←p. q ←link(q1).

)

7-8

- 讨论冒泡排序算法的稳定性。

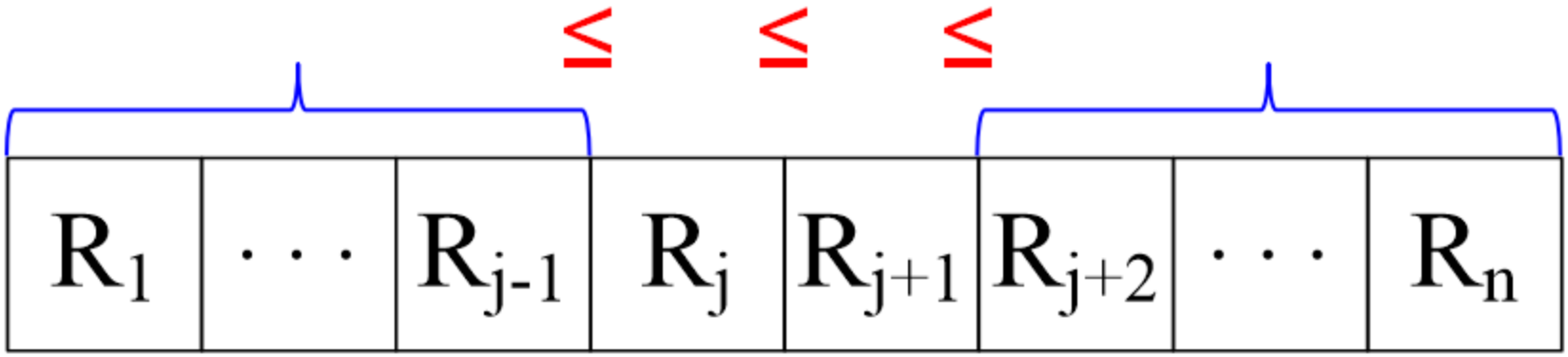
参考答案

- 冒泡排序中，每一趟冒泡，相邻的关键词只有满足Rj > Rj+1时才会发生交换，关键词相同的记录不会发生交换，即关键词相同的元素的相对位置不变，因此冒泡排序算法是稳定的。

7-10

- 类似于冒泡过程（从下到上），与之对应的是下沉过程（从上到下）。如果排序是冒泡和下沉的交替过程，证明如果经过一趟冒泡和一次下沉后发现Rj和$Rj+1$（$1 \le j \le n-1$）没有交换，则它们已经进入最终排序位置。

分析



参考答案

证明：

- 在一趟冒泡中，要比较Ri和Ri+1时， Ri已处于R0到Ri中的最大值；
- 在一趟下沉中，要比较Ri和Ri+1时， Ri+1已处于Ri+1到Rn中的最小值；
- 因为Ri和Ri+1没有交换，显然Ri小于等于Ri+1
- 综上， Rj和Rj+1进入最终位置。

7-18

- 奇偶交换排序算法的基本思想描述如下：排序过程通过对文件x[i](1≤i≤n)的若干次扫描来完成，第奇数次扫描，对所有下标为奇数的记录x[i]与其后面的记录x[i+1](1≤ i≤ n–1)相比较，若x[i].KEY＞x[i+1].KEY，则交换x[i]和x[i+1]的内容；第偶数次扫描，对所有下标为偶数的记录x[i]与其后面的记录x[i+1](2≤i≤n–1)相比较，若x[i].KEY＞x[i+1]. KEY，则交换x[i]和x[i+1]之内容，重复上述过程直到排序完成为止。

- (1)排序的终止条件是什么？

  (2)完成该算法的具体设计.

  (3)分析该算法的平均运行时间.

参考答案

- (1) 一趟奇偶比较无交换发生
- (2) 算法如下

```
template <typename T> void ParitySort(T X[], int n)
{
    for ( bool change = true; change; ) {
        change = false;
        for ( int i = 0; i <= n-1; i+=2 ) { //奇交換
            if (X[i].key>X[i+1].key) {
                Swap(X[i], X[i+1]);  change = true;}
        )
        for ( int i = 1; i <= n-1; i+=2 ) { //偶交換
            if (X[i].key>X[i+1].key ) {
                Swap(X[i], X[i+1]);  change = true;}
        }
    }
}
```

- (3) 一次交换只能减少一个反序对，根据书上P201反序对的平均数，平均时间为$O(n^2)$

7-24

- 填充如下排序算法中的方框，并证明该算法能正确性地分划.
- 算法PartA(R,s,e)

/*分划文件(R_s,R_{s+1},…,R_e), 且K_{s-1}=-∞,K_{e+1}=+∞ */

PA1[初始化]

i ←s. j ← ① .　　e+1

K ←K_s. R ←R_s.

- PA2[分划过程]

  WHILE i<j DO

  (j←j-1.

  WHILE ____②____ DO j←j-1.

  IF i ≥ j THEN j←i.

  ELSE (R_i ←R_j.

  i=i+1.

  WHILE K_i<K DO i ←i+1.

  IF ____③____ THEN R_j ←R_i.)).

- PA3 ____④____ ▌

$K_j \geq K$

i<j

$R_j \leftarrow R$

参考答案

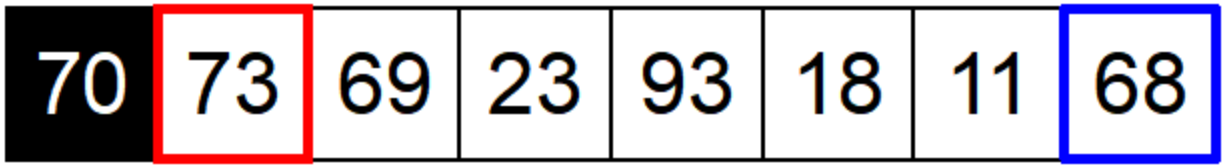
① e+1

② $K_j \geq K$

③ $i < j$

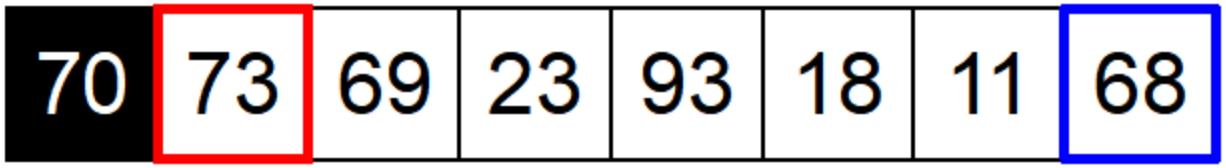
④ $R_j \leftarrow R$

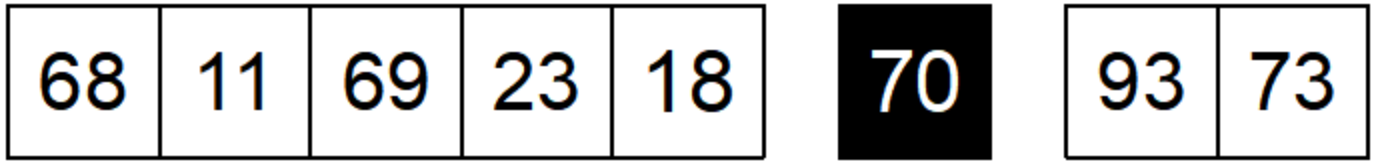
70	73	69	23	93	18	11	68


70
73
69
23
93
18
11
68


18
68
69
23
11
70
93
73

70	73	69	23	93	18	11	68


70
73
69
23
93
18
11
68


68
11
69
23
18
70
93
73

7-26

- 分析算法HSort中的堆栈*S*可能包含的最大元素个数（表示成*M*和*n*的函数）。

分析

- 假设f(n)表示n个元素时辅助堆栈的最大规模，显然f(n)是一个递增的函数
- n个元素划分之后剩下的前半部分和后半部分的总长度$n_1+n_2 = n-1$，当划分之后剩下的两长度不相等时，HSort算法会选择长度大的部分压入堆栈之中，不妨假设$n_1 <= n_2$，

$$f(n)=\begin{cases}0 & n<M\\ \max\{1+f(n_1),f(n_2)\} & n\geq M\end{cases}$$

- 若$1+f(n_1) <= f(n_2)$，相当于参数变小了，但函数值没增大，因此n_1接近n-1一半时，最大

参考答案

- 当 (n-1)/2 >= M，至少会向辅助堆栈中压入1个元素
- 当[(n-1)/2-1]/2 = (n-2^0-2^1)/2^2= (n-2^2+1)/2^2 >= M，至少会向辅助堆栈中压入2个元素
- ……，依此类推，
- 当(n-2^k+1)/2^k >=M，即2^k <= (n+1)/(M+1)，至少会向辅助堆栈中压入k个元素
- 因此向辅助堆栈中压入最大元素数为

$$\lfloor \log_2(n+1) - \log_2(M+1) \rfloor$$

7-30

- 证明：用淘汰赛找n个元素的最大元素正好需要n–1次元素比较。

参考答案

- 证明：在淘汰赛中，每进行一场比赛，即进行依次比较，都恰淘汰1个元素，找到最大元素需要淘汰n-1个元素，因此需要n-1比较。

7-31

- 证明：用淘汰赛找 n个元素的最大元素所形成的树高为$\lceil \log_2 n \rceil$.

参考答案

- 证明：

当$n=2^k$时，第1轮后剩下$n/2=2^{k-1}$，第二轮后剩下$n/4=2^{k-2}$,依次类推, 第k轮淘汰赛后就剩下1个选手，就是最大元素。每一轮淘汰赛都对应比赛树中的一层，因此当$n=2^k$时，比赛树的最大层数是k，比赛树的高度是$\log_2 n$

当n为任意数时，总可以找到一个k，使得$2^{k-1}<n<=2^k$，只需要k轮淘汰赛就可以最大元素，对应的比赛树高为k。由$2^{k-1}<n<=2^k$，得$\log_2 n<=k<\log_2 n+1$，即形成的树高为$\lceil \log_2 n \rceil$.

7-35

- 设文件($R1, R2, \ldots, Rn$)是一个堆，$Rn+1$是任意一个结点。请设计一个算法，该算法把$Rn+1$添加到堆($R1, R2, \ldots, Rn$)中，并使($R1, R2, \ldots, Rn, Rn+1$)成为一个堆（要求算法的时间复杂度为$O(\log_2 n)$）.

- 分析：堆有大根堆和小根堆。教材上用的是大根堆。

参考答案

算法Insert(R, n, x. R, n)

/*在堆中插入元素x，从下往上调整堆*/

I1[增加x元素]

n ← n+1. R[n] ←x.

I2[从下往上调整，称上浮]

i ← n.

WHILE i>1 AND R[i/2]<R[i] **DO** (

R[i/2] ↔ R[i].

i ← i/2.

)▌

7-36

- 文件($R1, R2, \ldots, Rn$)是一个堆，$1 \le i \le n$，请给出一个算法，该算法从($R1, R2, \ldots, Rn$)中删除 Ri，并使删除后的文件仍然是堆，要求算法的时间复杂度为$O(\log_2 n)$.

参考答案

算法Delete(R, n, i. R, n) // 在堆中删除元素R[i]，从上往下调整堆

D1 [删除第i个元素]

R[i] ←R[n]. n ← n-1.

D2 [从上往下调整，称下沉]

WHILE 2i＋1 < n **DO** (

j ← 2i.

IF K[j] < K[j+1] **THEN** j ← j+1.

IF K[i] < K[j] **THEN** (R[i]↔R[j]. i ← j.)

ELSE *i* ← n.

)

IF 2i < n **AND** K[i] < K[n] **THEN** R[i]↔R[n]. ▌

7-49

- 填充如下排序算法中的方框，并讨论该算法的稳定性.

算法C(R, n)

/*比较计数，本算法按关键词K_1、K_2、…、K_n排序记录R_1、R_2、…、R_n . 一维数组COUNT[1 : n]用来记录各个记录的排序位置*/

[C1]

FOR i=1 TO n **DO** ____①____. count[i] ←1

[C2]

FOR i=n TO 2 ____②____ **DO** STEP –1

FOR j=i–1 TO 1 STEP –1 **DO**

IF ____③____ **THEN** $K_j > K_i$

COUNT[j] ←COUNT[j]+1

ELSE ____④____ count[i]← count[i]+1

参考答案

- count[i] ←1
- STEP –1
- $K_j > K_i$
- count[i]← count[i]+1
- 算法具有稳定性

7-50

- 有一种简单的排序算法，叫做计数排序(*Count Sorting*). 这种排序算法对一个待排序的表(用数组表示)进行排序，并将排序结果存放到另一个新的表中。必须注意的是，表中所有待排序的关键词互不相同，计数排序算法针对表中的每个记录，扫描待排序的表一趟，统计表中有多少个记录的关键词比该记录的关键词小，假设针对某一个记录，统计出的计数值为*c*，那么，这个记录在新的有序表中的合适的存放位置即为*c* .
  - (1) 给出适用于计数排序的数据表定义。
  - (2) 对于有*n*个记录的表，关键词比较次数是多少？
  - (3) 与简单选择排序相比较，这种方法是否更好？为什么？

参考答案

- (1) 数据表放在数组R中，元素个数为n，下标从0开始（因为小于关键词的个数为c,就放在c的位置）。
- (2) 对于每个记录，扫描待排序的表一趟，即比较n次。一共n个记录，共需比较n^2次。
- (3) 时间效率：与简单选择排序相比，比较次数增加，移动次数减少。
  空间效率：辅助空间要高于简单选择排序。

THE END