第二周作业参考

2020 秋

1、简述下列术语：线性表，顺序表，链表。

- 线性表：线性表是由 N 个具有相同特征的元素构成的有限序列，各个元素有唯一的前驱和后继（队首可以无前驱，队尾可以无后继）。
- 顺序表：是在计算机中以数组的形式保存的线性表，其存储方式是将线性表中的元素依次存放在连续的存储空间中，其特点是逻辑上相邻的元素，在物理位置上也相邻。
- 链表：是一种物理存储单元上非连续、非顺序的存储结构，其特点是数据元素散列在内存中，地址不一定连续，其存储方式是存储单元通过相邻元素的地址来关联。

2、何时选用顺序表，何时选用链表作为线性表的存储结构合适？各自的主要优缺点是什么?

当表的容量固定，有大量随机访问数据时，选用顺序表；当表的容量时长变化、增删操作较多时，选用链表。

顺序表：优点：结构简单、存储效率高、是随机存储结构；缺点：插入或删除时需要移动多个数组元素，效率低；需要事先分配较大的存储空间或经常对表进行扩充。

链表：优点：表的长度灵活、易于扩充；插入删除的效率高。缺点：只能顺序访问元素；存储效率低。

3、在顺序表中插入和删除一个结点平均需要移动多少个结点？具体的移动次数取决于哪两个因素?

插入：$\frac{0+1+2+\cdots+n}{n+1}=\frac{n}{2}$。删除：$\frac{0+1+2+\cdots+n-1}{n}=\frac{n-1}{2}$。

具体的移动次数取决于顺序表中元素的个数和插入和删除的结点的位置。

4、链表所表示的元素是否有序？如有序，则有序性体现在何处？链表所表示的元素是否一定要在物理上是相邻的？有序表的有序性又如何理解？

有序。体现在通过存储的指针来关联，每个元素有且仅有唯一的直接前驱和后继（表头表尾元素除外）。物理上不一定要相邻。顺序表的有序性既体现在逻辑结构的有序，也体现在物理结构的有序。

5、设顺序表 L 是递增有序表，试写一算法，将 x 插入到 L 中并使 L 仍是递增有序表。

如算法 1所示。

Algorithm 1 有序表插入.

Input: 有序表 L；插入元素 x；

Output: 有序表 L；

1: **for** $i = L \to Length - 1$; $i >= 0$ and $L[i] > x$; $i - -$ **do**
2: $\quad L[i+1] = L[i]$;
3: **end for**
4: $L[i] = x$;
5: $L \to Length + +$;
6: **return** L;

6、写一求单链表的节点数目 ListLength(L) 的算法。

如算法 2所示。

Algorithm 2 求单链表长度.

Input: 单链表 L；

Output: 单链表长度 n；

1: $n = 0$;
2: LinkList* $p = L$;
3: **while** p **do**
4: $\quad n + +$;
5: $\quad p = p \to next$;
6: **end while**
7: **return** n;

7、写一算法将单链表中值重复的结点删除，使所得的结果链表中所有结点的值均不相同。

如算法 3所示。

Algorithm 3 删除单链表重复元素.

Input: 单链表 L;

Output: 单链表 L;

```
1: LinkList* p, q, r;
2: p = L;
3: while p do
4:     pre = p;
5:     while pre → next do
6:         if pre → next → value == p → val then
7:             cur = pre → next;
8:             pre → next = cur → next;
9:             delete r;
10:        else
11:            pre = pre → next;
12:        end if
13:        p = p → next;
14:    end while
15: end while
16: return L;
```

8、写一算法从一给定的向量 A 删除值在 x 到 y 之间的所有元素。

解法 1：O(n^2)，如算法 4所示。

解法 2：O(n)，如算法 5所示，思想为快慢指针。

9、设 A 和 B 是两个按元素值递增有序的单链表，写一算法将 A 和 B 归并为按元素值递减有序的单链表 C，试分析算法的时间复杂度。

解法如 6所示。

Algorithm 4 删除指定元素.

Input: 向量 A; 元素 x, y;

Output: 向量 A;

```
1: i = 0;
2: while i < A → Length do
3:     if A[i] ≥ x and A[i] ≤ y then
4:         for j = i; j < A → Length − 1; j++ do
5:             A[j] = A[j + 1];
6:         end for
7:         A → Length− = 1;
8:     else
9:         i++;
10:    end if
11: end while
12: return A;
```

Algorithm 5 删除指定元素.

Input: 向量 A; 元素 x, y;

Output: 向量 A;

```
1: low = −1;
2: for i = 0; i < A → Length; i++ do
3:     if A[i] < x or A[i] > y then
4:         low++;
5:         if low! = i then
6:             swap(A[low],A[i]);
7:         end if
8:     end if
9: end for
10: A → Length = low + 1
11: return A;
```

Algorithm 6 倒序合并链表.

Input: 链表 A 和 B;

Output: 链表 C;

```
1: Node *C=NULL;
2: while !A and !B do
3:     if A → value < B → value then
4:         Node *cur=new Node();
5:         cur → value = A → value;
6:         cur → next = C;
7:         C = cur;
8:         A = A → next;
9:     else
10:        Node *cur=new Node();
11:        cur → value = B → value;
12:        cur → next = C;
13:        C = cur;
14:        B = B → next;
15:    end if
16: end while
17: while !A do
18:    Node *cur=new Node();
19:    cur → value = A → value;
20:    cur → next = C;
21:    C = cur;
22:    A = A → next;
23: end while
24: while !B do
25:    Node *cur=new Node();
26:    cur → value = B → value;
27:    cur → next = C;
28:    C = cur;
29:    B = B → next;
30: end while
31: return C;
```