第 2 章 线性表

注：所有未特别说明的链表，均为带头结点的单向不循环链表

1、 设线性表有 n 个元素，以下操作中，__A__在顺序表上实现比在链表上实现效率更高

A 输出第 i 个元素值（i 在 1-n 之间）

B 交换第 1 个元素与第 2 个元素的值

C 顺序输出这 n 个元素的值

D 输出与给定值 x 相等的元素在线性表中的序号

2、 设线性表中有 2n 个元素，以下操作中，__A__在单链表上实现要比在顺序表上实现效率更高

A 删除指定的元素

B 在最后一个元素的后面插入一个新元素

C 顺序输出前 k 个元素

D 交换第 i 个元素和第 2n-i-1 个元素的值(i 在 0 - n-1 间)

3、 如果最常用的操作是取第 i 个结点及其前驱，则采用_D_存储方式最节省时间

A 单链表

B 双链表

C 单循环链表

D 顺序表

4、 将两个各有 n 个元素的有序顺序表(某个表中的元素，两个表之间的元素，值均有可能相同)归并成一个有序顺序表，其最少比较次数是__A__

A n

B 2n-1

C 2n

D n-1

5、 一个长度为 n(n>1)的带头结点单链表 h 上,另设有尾指针 r(指向尾结点),执行_B_的操作与链表的长度有关

A 删除单链表中的第一个元素

B 删除单链表的最后一个元素

C 在单链表的第一个元素前插入一个新元素

D 在单链表的最后一个元素后插入一个新元素

6、 双向循环链表中,在 p 结点之前插入 q 结点的操作是_D_

A
```
p->prior=q;
q->next=p;
p->prior->next=q;
q->prior=p->prior;
```
B
```
p->prior=q;
p->prior->next=q;
q->next=p;
q->prior=p->prior;
```
C
```
q->next=p;
q->prior=p->prior;
p->prior=q;
p->prior->next=q;
```

D q->next=p;
  q->prior=p->prior;
  p->prior->next=q;
  p->prior=q;

7、 在一个单链表中删除 p 结点(假设 p 不是尾结点)时，应执行如下操作:
(1) q=p->next;
(2) p->data=p->next->data;
(3) p->next=_p->next->next_;
(4) free(q);

8、 在一个单链表中的 p 结点之前插入一个 s 结点，可执行如下操作：
(1) s->next=__p->next__
(2) p->next=s;
(3) t=p->data;
(4) p->data=_s->data_
(5) s->data=_t_

9、 在一个双向循环链表中删除 p 结点时，应执行如下操作:
(1) _p->next->prior_ = p->prior;
(2) p->prior->next = _p->next_;
(3) free(p);

10、 在单链表、双向链表和单循环链表中，若仅知道指针 p 指向某结点，不知道头指针，能否将 p 从相应的链表中删除(不允许进行结点之间数据域的复制)？若可以，时间复杂度各为多少？

单链表：不行
双向链表：可以，时间复杂度 O(1)
但循环链表：可以，时间复杂度 O(n)

11、 设计一个高效算法，将顺序表的所有元素逆置，要求算法的空间复杂度为 O(1)

```
//以中间为界限，交换两边对称位置的元素，时间复杂度 O(n)
void Traverse(SeqList list, int length)
{
    for(int i = 0; i < length / 2; i++)
    {
        int temp = list[i];
        list[i] = list[length-i+1];
        list[length-i+1] = temp;
    }
}
```

12、 设计一个高效算法，从顺序表中删除所有元素值为 x 的元素，要求空间复杂度为 O(1)

```
//遍历顺序表中元素，将顺序表中不为 x 的元素，重新从头加入顺序表，同时修改顺序表长度，时间复杂度 O(n)
void DeleteElement(SeqList &list, ElemType x, int &length)
{
    int cur = 0
```

```
    for(int i = 0; i < length; i++)
    {
        if (list[i] != x) //节点的值不为 x
        {
            list[cur++] = list[i]; //将元素重新添加到表中的新位置
        }
        else //节点的值为 x
        {
            length--; //缩短表的长度
        }
    }
}
```

13、 用顺序表表示集合，设计一个求集合交集的算法

```
//遍历 a，如果 a 中的元素 x 在 b 中，则将 x 加入 c，时间复杂度 O(mn)
void Intersection(SeqList a, int aLength, SeqList b, int bLength, SeqList c, int &cLength)
{
    for(int i = 0; i < aLength; i++)
    {
        for(int j = 0; j < bLength; j++)
        {
            if(a[i] == b[j])
            {
                c[cLength++] = a[i];
                break;
            }
        }
    }
}
```

14、 从带头结点的循环单链表中删除值为 x 的第一个结点

```
void DeleteValue(LinkList L, ElemType x)
{
    LinkList p = L->next, q = L;
    while (p != L)
    {
        if(p->data == x) //找到了值为 x 的节点
        {
            q->next = p->next;
            free(p);
            break;
        }
        q = p;
        p = p->next;
    }
}
```

15、 假定有一个带头结点的链接表，头指针为 HL，每个结点含三个域:data，next 和 range，其中 data 为值域，next 和 range 均为指针域，现在所有结点已经由 next 域链接起来，试编一算法，利用 range 域(此域的初始值均为 NULL)把所有结点按照其值从小到大的顺序链接起来（next 域不变）

```
void SortLinkList(LinkList L)
{
    LinkList p = L->next;
    while (p != NULL)
    {
        LinkList q = L->range, t = L;
        while(q != NULL)
        {
            if (q->data > p->data) //找到第一个大于 p 的节点，此时 t 为最后一个小于等于 p 的节点
            {
                break;
            }
            t = q;
            q = q->range;
        }
        t->range = p; //建立链接
        p->range = q;
        p = p->next;
    }
}
```

16、 已知带头结点的单链表 L 是一个递增有序表，设计一个高效算法，删除表中 data 值在[min .. max]之间的所有结点，并分析算法的时间复杂度

```
//只需一次遍历就能完成全部操作，时间复杂度 O(n)
void DeleteRange(LinkList L, ElemType min, ElemType max)
{
    LinkList p = L->next, pMin = L->next, pMax = L->next;
    //更新 pMax，同时删除中间的节点，直到第一个大于等于 min 的节点
    while (p != NULL && p->data < min)
    {
        pMin = p;
        p = p->next;
    }
    if(p != NULL) //如果 p 为尾节点，则不需要删除任何节点
    {
        pMax = p = p->next;
        //更新 pMax，同时删除中间的节点，直到第一个大于 max 的节点
        while (p != NULL && p->data > max)
        {
            pMax = p;
            p = p->next;
            free(pMax);
        }
        pMin->next = pMax;
```

```
    }
}
```

17、　有一个值按非递减有序排列的单链表，设计一个算法删除值域重复的结点，并分析算法的时间复杂度

```
//默认链表有头结点，同样只需一次遍历就能完成全部操作，时间复杂度 O(n)
void DeleteRepeatValue(LinkList L)
{
    if (L->next == NULL || L->next->next == NULL)
        return;
    LinkList p = L->next->next, q = L->next, t = L->next;
    ElemType value = L->next->data
    while (p->next != NULL)
    {
        if (p->data == value) //出现重复的值
        {
            t = q; //记录这个重复值组成的子链表的第一个节点 t
            while (p->data == value) //删除有重复值的节点，直到找到第一个有不同值的节点
            {
                q = p;
                p = p->next;
                free(q);
            }
            t->next = p; //将重复值组成的子链表的第一个节点 t 与当前有不同值的节点 p 连接起来
        }
        q = p;
        p = p->next; //p 向前推进
        value = p->data; //更新 value 的值
    }
}
```

18、　用单链表表示集合，设计一个算法表示集合的交

```
//先将 La 的所有节点加入 Lc，在将 Lb 中没有出现在 La 中的所有节点加入 Lc
void Intersection(LinkList La, LinkList Lb, LinkList Lc)
{
    LinkList pa, pb, pc;
    pa = La->next;
    while(p1 != NULL) //将 La 中的节点加入 Lc
    {
        pc = (LinkList)malloc(sizeof(LNode));
        pc->data = pa->data;
        pc->next = Lc->next;
        Lc->next = pc;
        pa = pa->next;
    }
    pb = Lb->next;
    while(pb != NULL) //遍历 Lb
    {
```

```
        pa = La->next;
        while((pa != NULL) && (pa->data != pb->data)) //找到第一个 La 中与 pb 相同的节点
            pa = pa->next;
        if (pa == NULL) //没有找到这个节点
        {
            pc=(LinkList)malloc(sizeof(LNode)); //将节点 pb 加入 Lc
            pc->data = pb->data;
            pc->next = pc->next;
            Lc->next = pc;
        }
        pb = pb->next;
    }
}
```

19、 写出将带头结点的双向循环链表倒置的算法

```
void ReverseList(LinkList L)
{
    LinkList p = L->next;
    while (p != L)
    {
        LinkList t = p;
        p = p->next;
        LinkList temp = t->prior; //交换前驱和后继
        t->prior = t->next;
        t->next = temp;
    }
}
```

20、 设有一个双向链表 h,设计一个算法查找第一个元素值为 x 的结点，将其与后继结点进行交换

```
//假设双向链表有头结点
void SwapElem(LinkList h, ElemType x)
{
    LinkList p = h->next;
    while (p != NULL) //找到第一个元素值为 x 的节点
    {
        if (p->data == x)
            break;
        p = p->next;
    }
    if (p != NULL && p->next != NULL) //找到了这个节点并且这个节点有后继
    {
        ElemType e = p->data; //记录当前节点 p 的数据
        p->prior->next = p->next; //删除当前节点 p
        p->next->prior = p->prior;
        LinkList q = p->next; //记录 p 的后继节点 q
        free(p);
        //在 q 的位置之后插入一个新的节点，元素值以 p 相同
```

```
        LinkList s = (LinkList)malloc(sizeof(LNode));
        s->next = q->next;
        s->prior = q;
        s->data = e;
        if (q->next != NULL) //保证 q 的下一个节点存在
            q->next->prior = s;
        q->next = s;
    }
}
```

【作业要求：】

1、5 月 8 日前网上提交本次作业（直接在本文件中作答，转换为 PDF 后提交即可）

2、每题所占平时成绩的具体分值见网页

3、超过截止时间提交作业会自动扣除相应的分数，具体见网页上的说明

4、**答案用蓝色标注(选择题将正确选项直接设置为蓝色文字即可)**