数据结构作业(第二次)

PB20111686 黄瑞轩

注

第二章习题使用书上预设的顺序表、链表类型。

不妨设 `ElemType` 是 `int` 类型，比较函数为普通的 `>,<,==`。

2.19

```
void deleteNum(int mink, int maxk, LNode* Head){
    LNode* ptr = Head;
    while(ptr->next){
        if(ptr->next->data > mink){
            if(ptr->next->data < maxk){
                LNode* tmp = ptr->next;
                ptr->next = tmp->next;
                free(tmp);
            }
            else break;
        }
    }
    return;
}
```

时间复杂度：最坏情况下，所有节点都需要删除，时间复杂度为 $O(n)$。

2.21

```
void reverse(SqList* ts){
    int n = ts->length;
    int* elem = ts->elem;
    for(int i = 0;i <= (n - 2) / 2;i++){
        int tmp = elem[i];
        elem[i] = elem[n - 1 - i];
        elem[n - 1 - i] = tmp;
    }
}
```

2.22

这里认为头节点也存储信息。

```
void reverse(LNode* Head){
    if(Head == NULL) return;        // 空表
    if(Head->next == NULL) return; // 仅1个元素
    LNode* p = NULL, r = Head, n = Head->next;
    while(1){
        r->next = p;
        p = r;
        r = n;
        n = n->next;
        if(n->next == NULL){
            r->next = p;
            n->next = r;
            break;
        }
    }
}
```

2.24

这里认为头节点也存储信息，且两个表**都不是空表**。

```
void merge_reverse(LNode* AH, LNode* BH) {
    LNode* p = (AH->data < BH->data) ? AH : BH;
    LNode* p_b = p;
    LNode* min = (p == AH) ? BH : AH;
```

```
    LNode* q = p->next;
    LNode* max = min;
    while (1) {
        if (max->next == NULL) {
            if (max->data < q->data) {
                p->next = min;
                max->next = q;
                break;
            }
        }
        if (max->next->data < q->data) {
            max = max->next;
        }
        else {
            LNode* t = max->next;
            p->next = min;
            max->next = q;
            p = q;
            if (q->next == NULL) {
                q->next = t;
                break;
            }
            q = q->next;
            min = t;
            max = t;
        }
    }
    reverse(p_b);  //2.22中的reverse()函数
    return;
}
```

2.29

```
void duplicate_removal(SqList* A, SqList* B, SqList* C) {
    int la = A->length, lb = B->length, lc = C->length;
    int cnt = 0;
    //记录被删除的个数
    bool* isDeleted = new bool[la + 1];
    if (la & lb & lc) {
        int* ha = A->elem, hb = B->elem, hc = C->elem;
        int min = ha[0], max = ha[la - 1];
        for (int i = 0; i < la; i++)isDeleted[i] = false;
        int na = 0, nb = 0, nc = 0;
        //ni记录i当前到达的位置
        while ((lc - nc) & (lb - nb)) {
            //当b,c有一个没到底时，循环
            if (hb[nb] < hc[nc]) {
                nb++;
            }
            else {
                nc++;
            }
            //让b,c中最小的往前走
            if (hb[nb] == hc[nc]) {
                //如果相等
                if (hb[nb] >= min && hb[nb] <= max) {
                    //如果b,c相等的值可能出现在A中
                    //将A指针移到第一个大于等于b,c的地方
                    while (ha[na] < hb[nb]) {
                        na++;
                        ga++;
                    }
                    ga--;
                    //ga移到最后一个小于b的地方
                    //如果相等，就需要删除所有
                    while (ha[na] == hb[nb] && na < la) {
                        isDeleted[na] = true;
                        cnt++;//删除数量更新
                        na++;
                    }
                    //此时na移到了第一个大于b,c的位置，更新
                    min = ha[na];
                    nb++;
                    nc++;
                }
            }
```

```
        }
        int p = 0;
        isDeleted[la] = true;
        for (int j = 0; j < la; j++) {
            if (isDeleted[j] == false) {
                ha[p] = ha[j];
                p++;
            }
        }
        A->length = la - cnt;
        A->listsize = A->length * sizeof(int);
    }
    return;
}
```

时间复杂度：仅将`A,B,C`和`isDeleted`数组遍历了一遍，时间复杂度是$O(L_A + L_B + L_C)$。

2.38

```
void locate(LNode* L, int x){
    //访问元素值等于x的节点
    LNode* p = L, h = L;
    while(p->data!=x){
        if(p->next==h){
            p=p->next;
        }else{
            return;//没有这样的元素
        }
    }
    //找到了这样的元素
    p->freq++;
    if(p==L)return;
    if(p->prev->freq==p->freq)return;
    LNode* q = p->prev;
    while(q->freq==q->prev->freq){
        q=q->prev;
    }
    p->prev->next = p->next;
    p->next->prev = p->prev;
    q->prev->next = p;
    p->prev = q->prev;
    q->prev = p;
    p->next = q;
    return;
}
```