16337341_朱志儒_数据结构作业（二）

1、 线性表：由 n（n≥0）个数据元素（结点）a[0]，a[1]，a[2],…, a[n-1]组成的有限序列
顺序表：在计算机内存中以数组的形式保存的线性表，是指用一组地址连续的存储单元依次存储数据元素的线性结构
链表：散列在计算机中的存储单元，地址不一定连续，存储单元通过保存存储的地址来关联。

2、 选用顺序表：常进行查找操作而很少进行插入、删除操作的长度变化不大的线性表
选用链表：常进行插入、删除操作而很少进行查找操作的长度变化很大的线性表
顺序表的优点：结构简单，存储效率高，是一个随机存储结构，直接存取结构
缺点：进行插入、删除操作时，需要移动数据元素；对长度变化较大的线性表，要预先分配较大空间或经常扩充线性表，操作不便
链表的优点：灵活，插入、删除效率高
缺点：按值或位置查找数据效率低，存储密度低

3、 在顺序表中插入和删除一个结点平均需要移动结点个数：N
具体的移动次数取决的两个因素：插入位置和表的长度

4、 链表所表示的元素不一定有序；有序性体现在储存数据的有序性；链表所表示的元素不一定要在物理上相邻。有序表的有序性：逻辑上相邻的元素在物理储存上也相邻。

5、 算法：先从第一个元素进行查找，直到第 i 个元素小于 x 而第 i+1 个元素大于 x，再将第 i+1 个元素到最后一个元素均向后移动一个单位，最后将 x 插入到原第 i+1 个元素的位置。

6、 算法：从表头开始，遍历单链表直到最后一个结点指针指向 NULL 并计数。

7、 算法：

```
void delete_repeat(int *head) {
    if (head->next == nullptr || head == nullptr) return;
    int * p1 = head;
    int * p2 = head;
    while (p1->next != nullptr) {
        while (p2->next != nullptr) {
            if (p1->value == p2->next->value) {
                int *tmp = p2;
                if (p2->next->next != nullptr) {
                    p2 = p2->next->next;
                    delete tmp->next;
                    tmp->next = p2;
                }
                else {
                    delete tmp->next;
                    tmp->next = nullptr;
                }
            }
            else p2 = p2->next;
        }
        p1 = p1->next;
```

```
            p2 = p1;
        }
    }
```

8、算法：遍历向量 A 中的所有元素，如果有一元素的值在 x 到 y 之间，则将其删除。

9、算法：

```
    int *descending_sort(int *ahead, int* bhead) {
        if (ahead->value > bhead->value) {
            int *tmp = ahead;
            ahead = bhead;
            bhead = tmp;
        }
        int *p = ahead;
        while (p->next != nullptr && bhead != nullptr) {
            if (bhead->value >= p->value) {
                if (bhead->value >= p->next->value) {
                    p = p->next;
                }
                else {
                    int *tmp = bhead;
                    bhead = bhead->next;
                    tmp->next = p->next;
                    p->next = tmp;
                    p = p->next;
                }
            }
        }
        if (p->next == nullptr) p->next = bhead;
        int *p1 = ahead;
        int *p2 = p1->next;
        ahead = nullptr;
        while (p1->next != nullptr) {
            p1->next = ahead;
            ahead = p1;
            p1 = p2;
            p2 = p2->next;
        }
        p1->next = ahead;
        return p1;
    }
```

算法复杂度：O(n)