3.列表

（c）有序列表

邓俊辉

deng@tsinghua.edu.cn

唯一化

```
❖template <typename T> int List<T>::uniquify() { //成批剔除重复元素
    if ( _size < 2 ) return 0; //平凡列表自然无重复
    int oldSize = _size; //记录原规模
    ListNodePosi(T) p = first();; //p为各区段起点
    ListNodePosi(T) q; //q为其后继
    while ( trailer != ( q = p->succ ) ) //反复考查紧邻的节点对(p, q)
       if ( p->data != q->data ) p = q; //若互异，则转向下一区段
       else remove(q); //否则（雷同），删除后者
    return oldSize - _size; //规模变化量，即被删除元素总数
 } //只需遍历整个列表一趟，O(n)
```

查找

❖ template <typename T> //在有序列表内节点p的n个（真）前驱中，找到不大于e的最后者

```
Posi(T) List<T>::search(T const & e, int n, Posi(T) p) const {
    while ( 0 <= n-- ) //对于p的最近的n个前驱，从右向左
        if ( ( ( p = p->pred ) -> data ) <= e ) break; //逐个比较
    return p; //直至命中、数值越界或范围越界后，返回查找终止的位置
} //最好O(1)，最坏O(n)；等概率时平均O(n)，正比于区间宽度
```

❖ 语义与向量相似，便于插入排序等后续操作：insertA(search(e, r, p), e)

❖ 为何未能借助有序性提高查找效率？实现不当，还是根本不可能？

❖ 按照循位置访问的方式，物理存储地址与其逻辑次序无关

依据秩的随机访问无法高效实现，而只能依据元素间的引用顺序访问