华东师范大学数据科学与工程学院上机实践报告

课程名称：算法设计与分析　　**年级：**22 级　　**上机实践成绩：**

指导教师：金澈清　　**姓名：**郭夏辉

上机实践名称：数据结构　　**学号：**10211900416　　**上机实践日期：**2023 年 4 月 6 日

上机实践编号：No.6　　**组号：**1-416

一、目的

1．熟悉算法设计的基本思想。
2．掌握栈、链表等基本数据结构。
3．掌握使用非显式指针来表示链表等数据结构的方法。

二、实验内容

1. 用多重数组实现单链表。
2. 实现在该单链表上的“增删查改排”操作，插入并不一定在头部插入，排序可以用任意一种排序方法。注意释放对象。
3. 和用指针实现的链表相比，本次实验中实现的链表有什么使用和性能上的异同呢？（选做）请通过实验来验证你的猜想。

三、使用环境

可以使用你自己喜欢的编程语言和环境来实现。

四、实验过程

1. *写出多重数组单链表的源代码。*
2. *验证实现的正确性，给出各种情况的测试数据和运行结果。*
3. *分析并提出假设并验证你的猜想。*

- 指针实现链表

如图 10-3 所示，**双向链表**(doubly linked list)L 的每个元素都是一个对象，每个对象有一个关键字 key 和两个指针：$next$ 和 $prev$。对象中还可以包含其他的辅助数据(或称卫星数据)。设 x 为链表的一个元素，$x.next$ 指向它在链表中的后继元素，$x.prev$ 则指向它的前驱元素。如果 $x.prev$=NIL，则元素 x 没有前驱，因此是链表的第一个元素，即链表的头(head)。如果 $x.next$=NIL，则元素 x 没有后继，因此是链表的最后一个元素，即链表的尾(tail)。属性 $L.head$ 指向链表的第一个元素。如果 $L.head$=NIL，则链表为空。

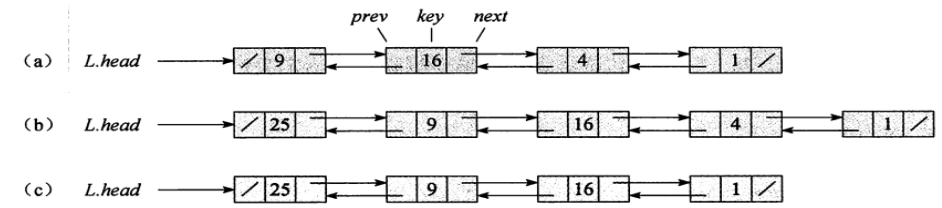


图 10-3　(a)表示动态集合{1，4，9，16}的双向链表 L。链表中的每个元素都是一个对象，拥有关键字和指向前后对象的指针(用箭头表示)。表尾的 $next$ 属性和表头的 $prev$ 属性都是 NIL，用一个斜杠表示。属性 $L.head$ 指向表头元素。(b)在执行 LIST-INSERT(L，X)之后(这里 $x.key$=25)，链表以关键字为 25 的新对象作为新的表头。该新对象指向原来关键字为 9 的表头元素。(c)随后调用 LIST-DELETE(L，X)的结果，其中 x 指向关键字为 4 的对象

链表的搜索

过程 LIST-SEARCH(L，k)采用简单的线性搜索方法，用于查找链表 L 中第一个关键字为 k 的元素，并返回指向该元素的指针。如果链表中没有关键字为 k 的对象，则该过程返回 NIL。对于图 10-3(a)中的链表，调用 LIST-SEARCH(L，4)返回指向第三个元素的指针，而调用 LIST-SEARCH(L，7)则返回 NIL。

```
LIST-SEARCH(L, k)
1  x = L.head
2  while x ≠ NIL and x.key ≠ k
3      x = x.next
4  return x
```

要搜索一个有 n 个对象的链表，过程 LIST-SEARCH 在最坏情况下的运行时间为 $\Theta(n)$，因为可能需要搜索整个链表。

链表的插入

给定一个已设置好关键字 key 的元素 x，过程 LIST-INSERT 将 x“连接入”到链表的前端，如图 10-3(b)所示。

```
LIST-INSERT(L, x)
1  x.next = L.head
2  if L.head ≠ NIL
3      L.head.prev = x
4  L.head = x
5  x.prev = NIL
```

(我们知道属性符号是可以嵌套的，因此 $L.head.prev$ 表示的是 $L.head$ 所指向的对象的 $prev$ 属性。)在一个含 n 个元素的链表上执行 LIST-INSERT 的运行时间是 $O(1)$。

链表的删除

过程 LIST-DELETE 将一个元素 x 从链表 L 中移除。该过程要求给定一个指向 x 的指针，然后通过修改一些指针，将 x“删除出”该链表。如果要删除具有给定关键字值的元素，则必须先调用 LIST-SEARCH 找到该元素。

```
LIST-DELETE(L, x)
1  if x.prev ≠ NIL
2      x.prev.next = x.next
3  else L.head = x.next
4  if x.next ≠ NIL
5      x.next.prev = x.prev
```

图 10-3(c)展示了从链表中删除一个元素的操作。LIST-DELETE 的运行时间为 $O(1)$。但如果要删除具有给定关键字的元素，则最坏情况下需要的时间为 $\Theta(n)$，因为需要先调用 LIST-SEARCH 找到该元素。

通过指针实现链表是繁琐的，我结合实际情况写了一下相关的代码(见附件)。

- 多数组实现链表

对象和指针。

对象的多数组表示

对每个属性使用一个数组表示，可以来表示一组有相同属性的对象。图 10-5 举例说明了如何用三个数组实现图 10-3(a)所示的链表。数组 *key* 存放该动态集合中现有的关键字，指针则分别存储在数组 *next* 和 *prev* 中。对于一个给定的数组下标 x，三个数组项 $key[x]$、$next[x]$和 $prev[x]$一起表示链表中一个对象。根据这种解释，指针 x 即为数组 *key*、*next* 和 *prev* 的一个共同下标。

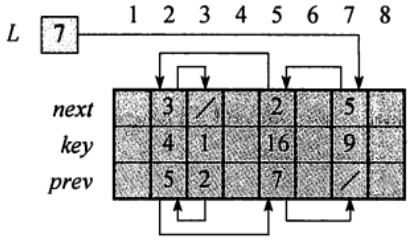


图 10-5 用数组 *key*、*next* 和 *prev* 表示图 10-3(a)中的链表。每一列数组项表示一个单一的对象。数组内存放的指针对应于上方所示的数组下标；箭头给出其形象表示。浅阴影的位置存放的是表内元素。变量 L 存放表头元素的下标

在图 10-3(a)所示的链表中，关键字为 4 的对象紧邻关键字为 16 的对象之后。在图 10-5 中，关键字 4 出现在 $key[2]$，关键字 16 出现在 $key[5]$，因此 $next[5]=2$，$prev[2]=5$。尽管常数 NIL 出现在表尾的 *next* 属性和表头的 *prev* 属性中，但我们通常用一个不能代表数组中任何实际位置的整数(如 0 或－1)来表示。此外变量 L 存放表头元素的下标。

用多数组来实现单链表相对来说要更加容易一些，为了便利自己的操作，我将表示链表的三个数组放到了一个类中，通过封装相关的函数简化了实现的过程。虽然这个题目只是要求实现单链表，但为了简化相关的流程，我采用的其实是双链表模型。

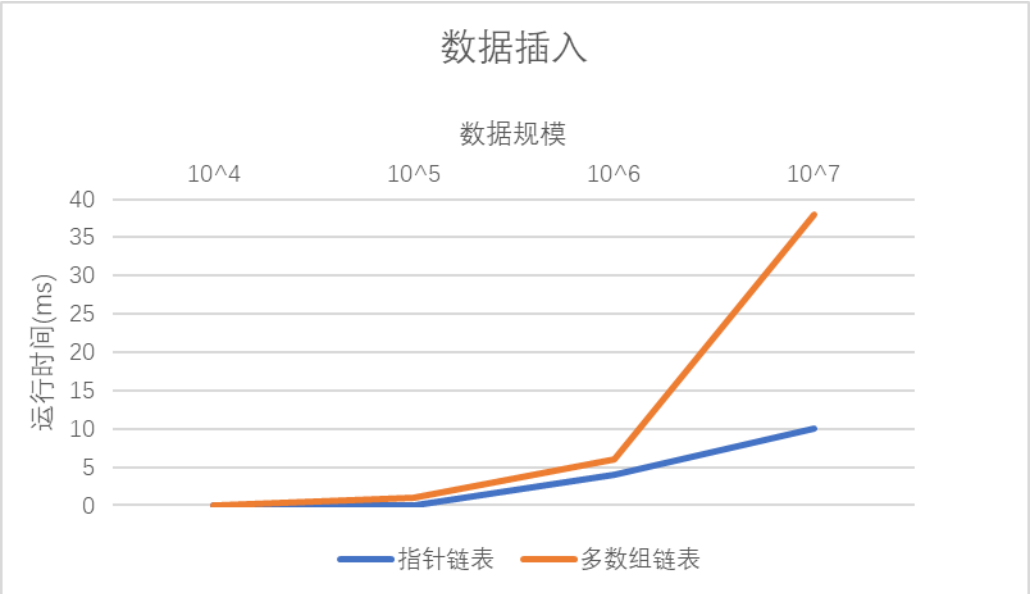
对于“增”操作，就正常的判断、循环即可，重点是对边界条件的处理；对于“删”操作，也是很简单的操作，重点是先后顺序不要搞混了（数据结构小知识点/敲黑板）；对于“查”操作，要按顺序地从头找到对应的元素，此时要注意边界条件；对于“改”操作，应该先特判一下改的索引值不能越界，然后再执行一个“查”之后实现对相应元素的修改，这里不可避免的查询操作是时间复杂度无法降低的根本原因。而最后对于排序操作，我在实际操作的过程中碰到了许多问题。

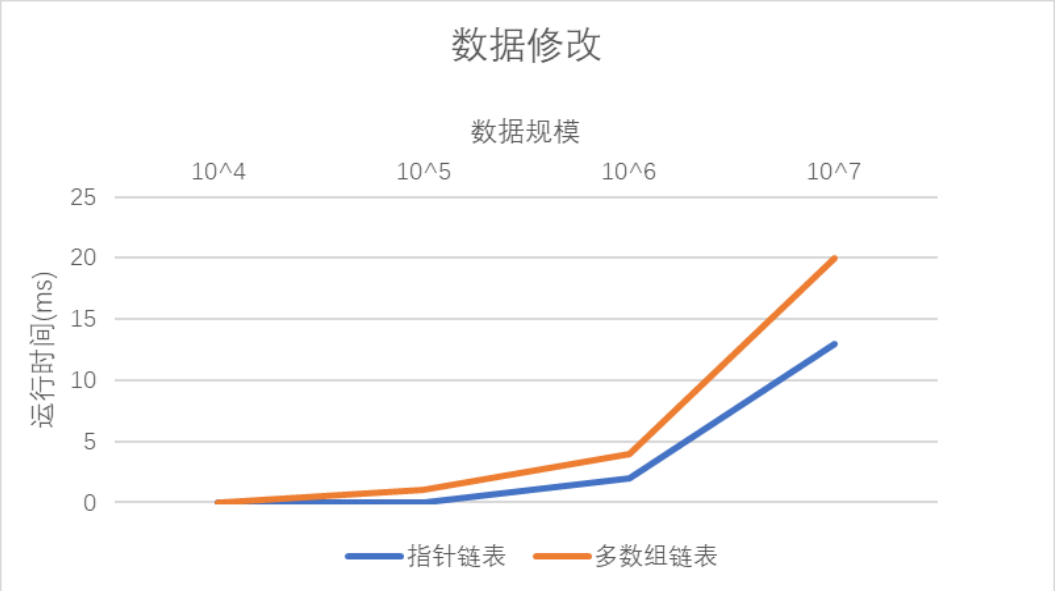
首先从最优化的排序算法入手，由于链表在内存空间上是不一定连续的，快速排序这样的基于连续空间的排序算法是无法使用的——每次都要找一个合适的 pivot 消耗的平均时间太大了！但是利用分治的归并排序算法却行，主要思路是这样的:首先找到链表的中心结点，然后以此二分两边进行 merge 操作即可——这样的时间复杂度和正常的归并排序一样，是Θ(nlgn).

可是实现链表上的归并排序是复杂的，对于我们眼前的问题，是不是可以采用虽然时间复杂度较差但更容易实现的算法呢？很直接的想法是冒泡排序和插入排序，因为这两种排序算法都是根据某个元素相邻的元素不断比较最后达到使一个数列有序化的目的，而链表中我们可控的元素不正是某个结点的相邻结点吗？我在实际的代码实践过程中是以插入排序完成的，具体的方法就不再赘述了，可以参考传统的插入排序的流程。但是这里有一个小细节，就是插入排序找的那个插入的地方距离当前元素的位置毕竟还是有些距离的，不像冒泡排序那样即便交换也就仅仅是相邻元素的交换。基于链表的性质，如果这里使用冒泡排序，在需要交换相邻元素时其实不需要使得相邻的那两个元素的三个特征全部交换，我们只需要交换其中的 val 值就行了！这样就可以节省很多赋值等操作消耗的时间。即便这样确实取得了很大的优化，但从时间复杂度角度来讲冒泡排序和插入排序一样都是Θ(n^2).

五、总结

对上机实践结果进行分析，问题回答，上机的心得体会及改进意见。





虽然本次实验没有明确地要求，但我还是简单地比较了一下两种链表的修改和插入操作。在实际情况中，由于我的指针式链表通过的是尾插法插入结点但是多数组链表在插入之前要先通过一个循环到对应的位置，所以很明显多数组链表的插入操作时间复杂度是线性的。对于数据修改的情况与之类似，时间复杂度也是线性的。即便我没有绘制查找操作的时间复杂度对比图，但是其运行情况应该是类似于上述两张图的，还是线性的时间复杂度。

如果使用头插法构建链表，多数组实现链表的运行时间应该是小于指针实现的，这主要是因为指针结点要花费大量的时间在无意义的操作上，但是多数组的运行过程却没有额外的操作。

本次实验从整体上来说还是比较简单的，通过实现多数组链表的简单操作，我对链表的原理有了一个更为深刻的认识，相信未来能更加好的利用链式数据结构在特定场合下的价值。