1. 线性表的定义

线性表是具有**相同特性**数据元素的一个**有限序列**。该序列中所含元素的个数叫做线性表 的长度，用 n 表示 ($n \geq 0$); 注意 n 可以等于零，表示线性表是一个空表，**空表也可以作为一个线性表**。

2. 线性表的逻辑特性

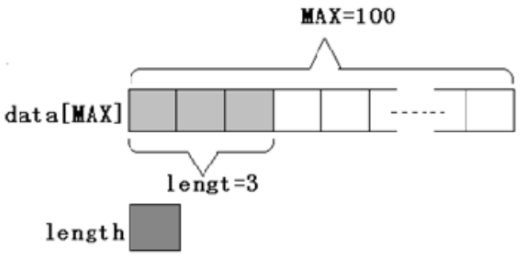
> 线性表只有一个表头元素，只有一个表尾元素，**表头元素没有前驱**，**表尾元素没有后继**，除表头和表尾元素之外其他元素只有一个直接前驱，也只有一个直 接后继。

3. 线性表的存储结构

线性表的存储结构有**顺序存储**结构和**链式存储**结构两种。前者称为顺序表，后者称为 链表。

3.1 顺序表

顺序表就是把线性表中的所有元素按照其逻辑顺序，依次存储到存储器中从指定存储位置开始的**一块连续的存储空间**中。



- a. **随机访问特性**
- b. 顺序表要求占用**连续的存储空间**，存储分配只能预先进行，即**静态分配**，一旦分配好了，在对其操作过程中不变。

- c. 插入操作的时候要移动多个元素。

3.2 链表

在链表存储中，每个结点不仅包含所存元素的信息，还包含**元素之间逻辑关系的信息**，即前驱结点包含后继结点的*地址信息*，这样就可以通过前驱结点中的地址信息方便的找到后继结点的位置。

- a. **不支持随机访问**
- b. 支持存储空间的**动态分配**，即在需要新的结点时再进行空间划分，而不需要一次性的划分所有所需空间给链表。

3.2.1 单链表

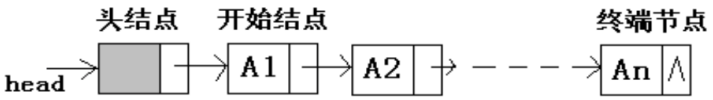


图 2.2 带头结点的单链表

在每个结点中除了包含**数据域**外，还包含一个**指针域**，用以指向其后继结点。如图 2.2 所示，即为带头结点的单链表。这里要区分一下带头结点的单链表和不带头结点的单链表。带头结点的单链表有一个结点不存储信息，只是作为标记， 而不带头结点的单链表所有结点都存储信息。

1. 带头结点的单链表中头**指针 head 指向头结点**，头结点的值域不含任何信息，从头 结点的后继结点开始存储信息。**头指针 head 始终不等于 NULL，head->next==NULL 的时候链表为空。**
2. 不带头结点的单链表其中的头指针 head 直接指向开始结点，即图 2.2 中的结点 A_1， 当 **head==NULL 的时侯链表为空。**

3.2.2 双链表

双链表就是在单链表结点上增添了一个指针域，指向当前结点的前驱。这样就**可以方便的由其后继来找到其前驱*， 而实现输出终端结点到开始结点的数据序列。

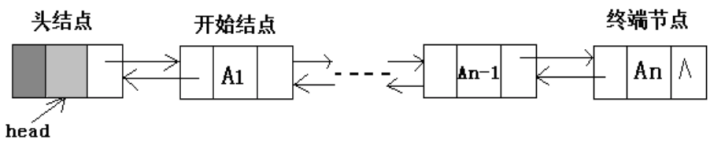


图 2.3 双链表

- 带头结点的双链表 **head->next==NULL** 的时候链表为空。
- 不带头结点的双链表 **head==NULL** 的时候链表为空。

3.2.3 循环单链表

只要将单链表的**最后一个指针域(空指针)指向链表中第一个结点即可**(这里之所以说第一个结点而不说是头结点是因为，如果循环单链表是带头结点的则最后一个结点的指针域要指向头结点; 如果循环单链表不带头结点，则最后一个指针域要指向开始结点)。
循环单链表**可以实现从任一个结点出发访问链表中任何结点**，而单链表从任一结点出 发后只能访问这个结点 身及其后边的所有结点。

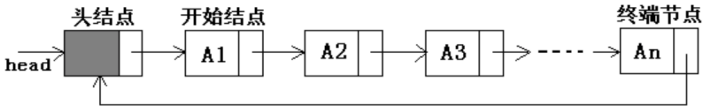


图 2.4 循环链表

- 带头结点的循环单链表当 **head==head->next** ;时链表为空。
- 不带头结点的循环单链表当 **head==NULL** 时链表为空。

3.2.4 循环双链表

循环双链表的构造源自双链表，即将**终端结点的 next 指针指向链表中第一个结点**，将**链表中第一个结点的 prior 指针指向终端结点**，如图 2.5 所示。循环 双链表同样有带头结点和不带头结点之分。

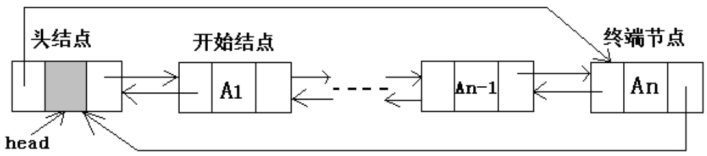


图 2.5 双循环链表

- 带头结点的循环双链表当 **head->next==head** 和 **head->prior==NULL** 时链表为空。
- 不带头结点的循环双链表当 head==NULL 的时候为空。

3.2.5 静态链表

这种链表借助一维数组来表示

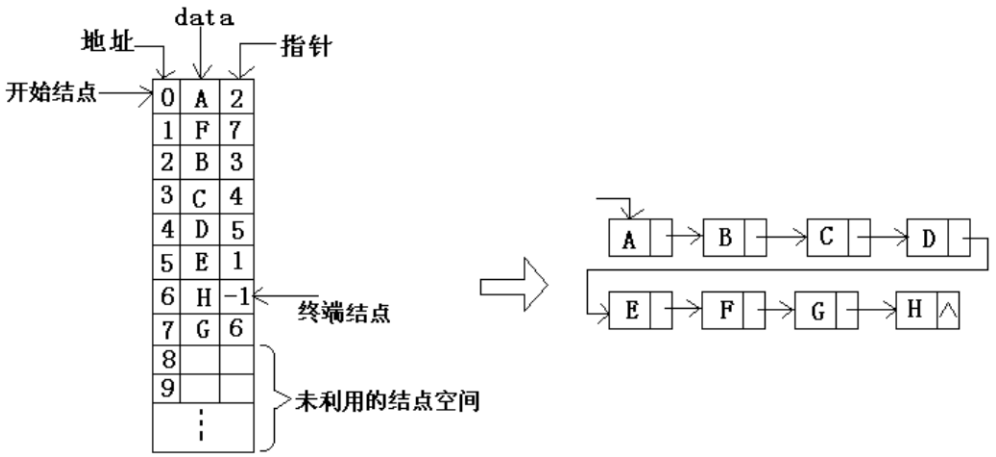


图 2.6 静态链表的表示

左图是静态链表，右图是其对应的一般链表。一般链表结点空间是来自于整个内存，静态链表则来自于一个**结构体数组**，数组中每一个结点含有两个分量，一个是**数据元素分量 data**，另一个是**指针分量**，指示了**当前结点的直接后继结点在数组中的位置**(这 和一般链表中 next 指针的地位是同等的)。

4. 顺序表和链表的比较

4.1 基于空间的比较

- (1) **存储分配**的方式:
  - a. 顺序表的存储空间是静态分配的。
  - b. 链表的存储空间是动态分配的。

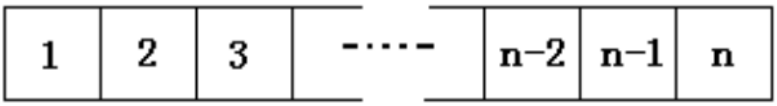
- (2) **存储密度**(存储密度= 结点数据域所占的存储量/结点结构所占的存储总量):
  - a. 顺序表的存储密度 =1。
  - b. 链表的存储密度<1(因为结点中有指针域)。

4.2 基于时间的比较

- (1) **存取方式**:
  - a. 顺序表可以随机存取，也可以顺序存取(对于顺序表一般只答随机存取即可)
  - b. 链表是顺序存取的
- (2) **插入/删除时移动元素个数**:
  - a. 顺序表平均需要**移动近一半元素**。
  - b. 链表不需要移动元素，只需要修改指针。

4.3 顺序表插入和删除算法时间复杂度分析

具有 n 个元素的顺序表(如下图)，插入(删除)一个元素所进行的**平均移动个数**为多少(假设新元素插入在表中每个元素之后)。



1. 求概率:
   因为插入位置的选择是随机的，所以所有位置被插入的可能性都是相同的，有 n 个可 插入位置，所以任何一个位置被插入元素的概率都为 $p = \frac{1}{n}$。
2. 求对应于每个插位置需要移动的元素个数:
   假设要**把新元素插入在表中第 i 个元素之后，则需要将 i 元素之后的所有元素往后移动一个位置**，因此**移动元素个数为 $n - i$** 。 由 1 和 2 知，移动元素个数的期望 E 为:

$$E = p \times \sum_{i=1}^{n} (n - i) = \frac{n - 1}{2}$$