线性表

最初，程序员编写程序使用的是机器语言（数据和指令都是由于0和1构成的），敲代码就是0和1这两个主要输入按键，很难想象当时的程序员有多么的艰辛。因此，现代程序员们要珍惜现在的学习和编程环境。对于数据而言，不管是数值、字符，存储到计算机中进行处理，都需要先找到存储地址，然后把数据对应的0和1存储进入。存储非常低效，且容易出错。数据没有类型，在处理数据时可能对字符类型数据施加不存在的运算，如数值数据的加减乘除。每次储存数据，还有亲自去找数据的存储空间。

为了解决上述问题，高级程序设计语言，引入了数据类型的概念。使得程序员在进行数据存储时，不需要再直接和存储器的地址打交道。并且在编译时，编译器可以检测该数据类型运算的合法性。

基本数据类型

数据类型反映了数据的取值范围以及对这类数据可以施加的运算。

抽象数据类型

抽象数据类型（abstract data type,ADT）只是一个数学模型以及定义在模型上的一组操作。通常是对数据的抽象，定义了数据的取值范围以及对数据操作的集合。

就像“超级玛丽”这个经典的任天堂游戏，里面的游戏主角是马里奥，我们给他定义了基本操作，前进、后退、跳、打子弹等。这就是一个抽象数据类型，定义了一个数据对象、对象中各元素之间的关系及对数据元素的操作。至于，到底是哪些操作，这只能由设计者根据实际需要来定。像马里奥可能开始只能走和跳，后来发现应该增加一种打子弹的操作，再后来又有了按住打子弹键后前进就有跑的操作。这都是根据实际情况来定的。

处理手法：对数据归一化处理。

线性表的抽象数据类型

线性表抽象数据类型主要包括两个方面：既数据集合和该数据集合上的操作集合。

线性结构

如果一个数据元素序列满足：

1. 除第一个和最后一个数据元素外，每个数据元素只有一个前驱数据元素和一个后继数据元素；
2. 第一个数据元素没有前驱数据元素；
3. 最后一个数据元素没有后继数据元素；

我们称这样的结构就叫做 线性结构

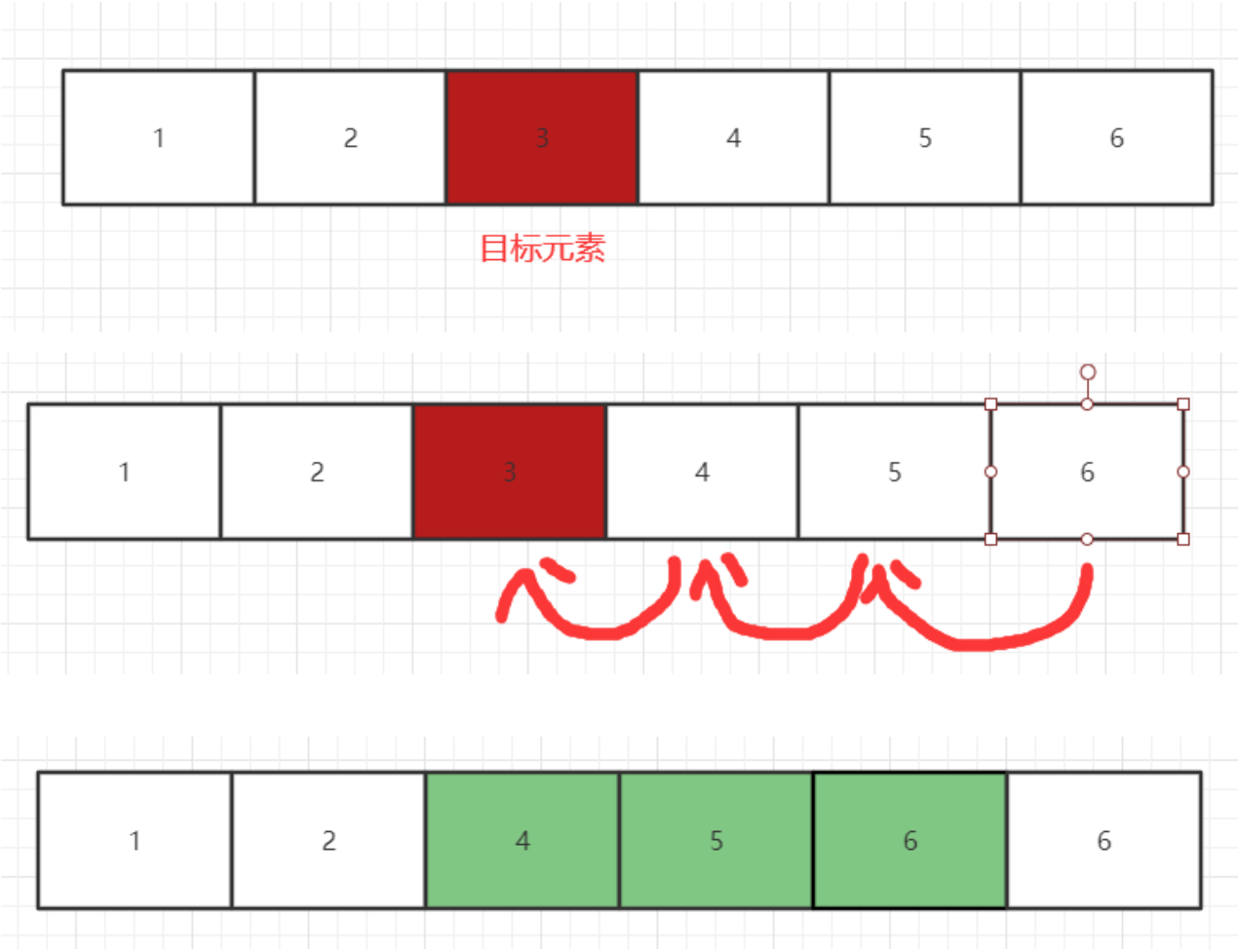
顺序表

在计算机内存中，顺序表是以数组的形式保存的线性表。也就是一组地址连续的存储单元依次存储数据元素的线性结构。

在数组中，我们会先申请一段连续的内存空间，然后把数组以此存入内存当中，中间没有一点空隙。这就是一种顺序表存储数据的方式。对于顺序表的基本操作有：增（add），删（remove），改（set），查（find），插（insert）。

顺序表删除元素

从顺序表中删除指定的元素，其实实现起来是非常简单的，只需要找到目标元素，并将其后续的所有元素整体前移1个位置即可。比如



上面我们发现，顺序表删除元素，其实是在间接删除目标元素。就是将后续元素整体向前移动一个位置。

时间复杂度分析：

从顺序表中删除元素，最好的情况是删除的元素刚好是最后一个元素，这时候不需要移动元素，只需要把表的size-1即可。最坏的时间复杂度刚好是第一个元素，这个时候需要后面全部的元素向前移动一位，同时size - 1，时间复杂度是O(N)。我们分析时间复杂度的原则是分析最快情况，这样才有意义。因此删除操作的时间复杂度为O(N)。

顺序表插入元素

向已有的顺序表中插入元素，根据插入的位置不同，可以分为3种情况。

1.插入到顺序表的表头

2.在表中间位置插入

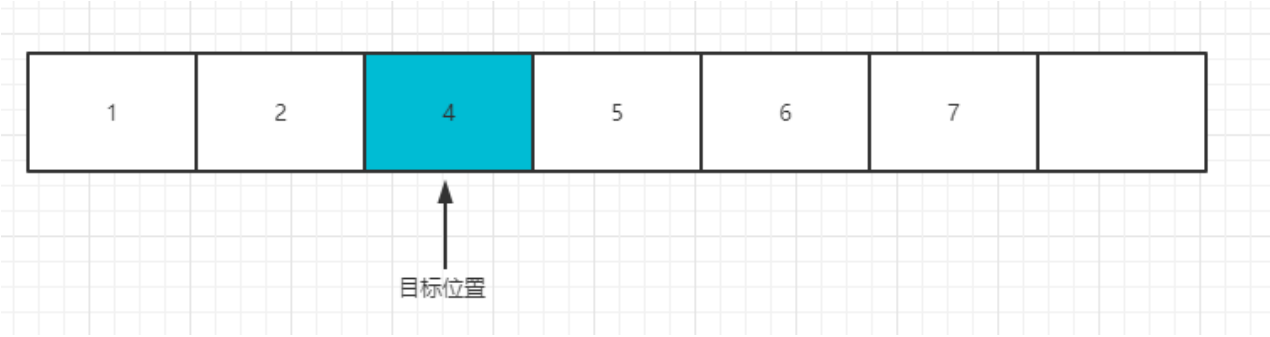
3.尾随顺序表中已有的元素，作为最后一个元素插入

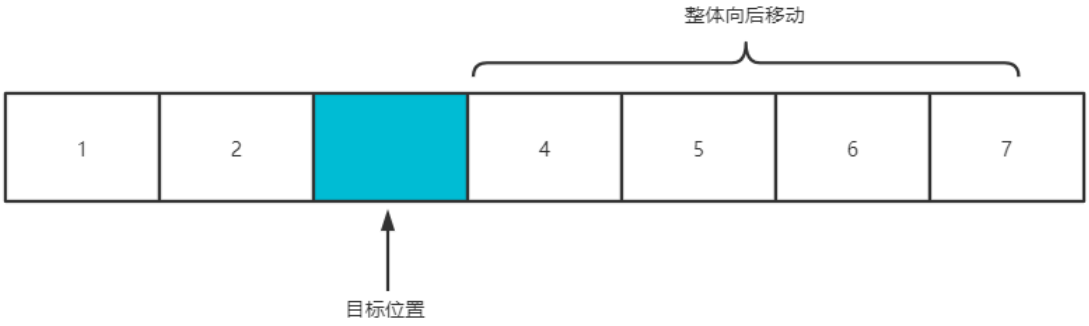
虽然数据元素插入顺序表中的位置有所不同，但是都是使用的是同一个方法去解决的。就是通过遍历，找到数据元素要插入的位置。然后要做到如下两步工作。

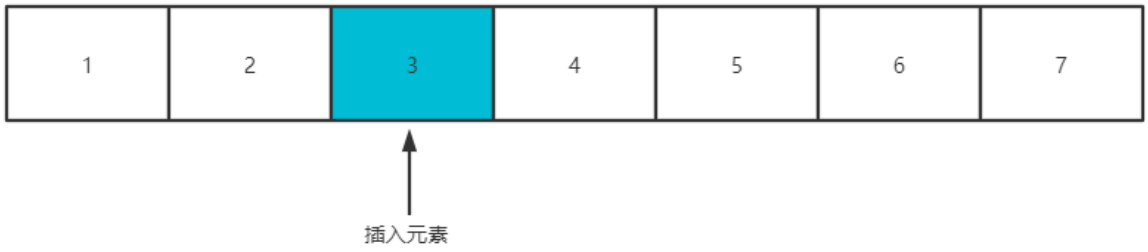
将要插入的位置元素以及后续的元素整体向后移动一个位置

将元素放到腾出来的位置上。

例如，在{1,2,4,5,6,7}的第三个位置插入3.实现过程如下







从时间复杂度来看，同删除元素是一样的，均为O(N)。

优势

因为数据在数组中按顺序存储，可以通过数组下标直接访问，因此顺序表的查找定位元素很快。

劣势

插入和删除元素都需要大量的操作。

因为数组在声明的时候需要确定长度，因此顺序表的长度是确定的。若需要扩大顺序表长度，有需要大量的操作，不够灵活。（要将该数组中的元素全部copy到另外一个数组）

由于数据大小的不可测性，有时会浪费掉大量的空间。

应用场景

总之，顺序表适用于那些不需要对于数据进行大量改动的结构。

顺序表的效率分析

综上所述，可以得出。顺序表对于插入、删除一个元素的时间复杂度是O(n)。

因为顺序表支持随机访问，顺序表读取一个元素的时间复杂度为O(1）。因为我们是通过下标访问的，所以时间复杂度是固定的，和问题的规模无关。

最大的优点是空间利用率高。最大的缺点是大小固定。

链表

刚才我们了解到，数组作为数据存储结构有一定的缺点。在无序数组中，搜索时十分低效。在有序数组中，插入的效率很低。不管在哪种数组中，删除的效率都很低。而且大小无法改变。为了应对顺序表的缺陷，链表就此诞生。链表也是继数组之后第二种使用的最广泛的通用数据结构

链表结构：在物理上不连续，在逻辑上连续。大小不固定。

链式存储结构是基于指针实现的。我们把一个数据元素和一个指针称之为节点。

数据域：存数据元素的区域

指针域：存储直接后继位置的区域。

链式存储，其实就是用指针将相互关联的结点链接起来。

链式存储根据链表的构造不同，可以分成：单向链表，单向循环链表，双向链表，双向循环链表。

单向链表

概念

链表的每个节点只包含一个指针域。叫做单链表（即构成链表的每个节点只有一个指向后继节点的指针）

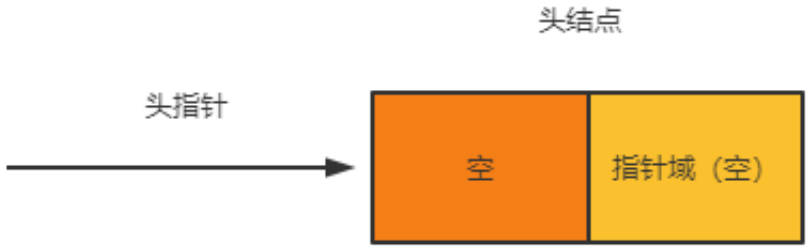
1、头指针和头节点

单链表有带头节点和不带头节点两种结构。

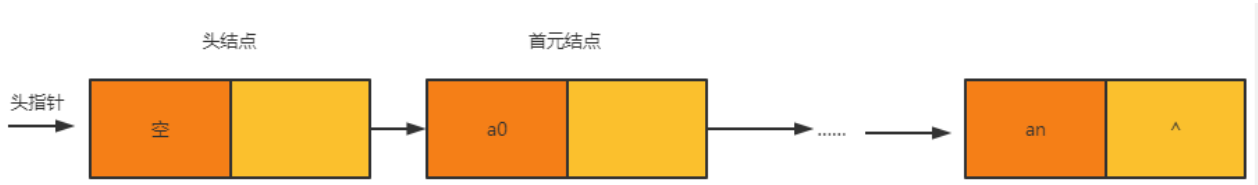
链表中，第一个结点存储的位置叫头指针，如果链表有头结点，那么头指针就是指向头结点的指针。

头指针所指的不存在数据元素的第一个结点就叫做头结点（而头结点又指向首元结点）。头结点一般不放数据（有的时候也是放的，比如链表的长度，用做监视）。

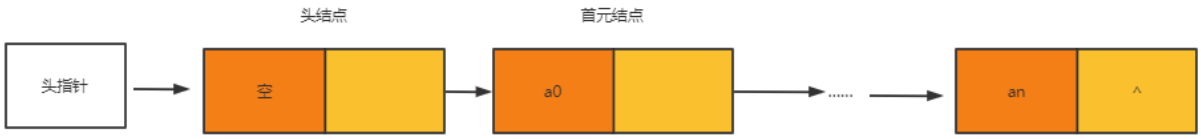
存放第一个数据元素的结点叫做第一个数据元素结点，也叫做首元结



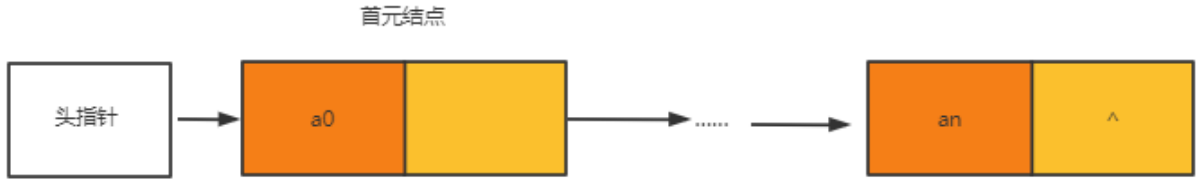
空链



非空链表

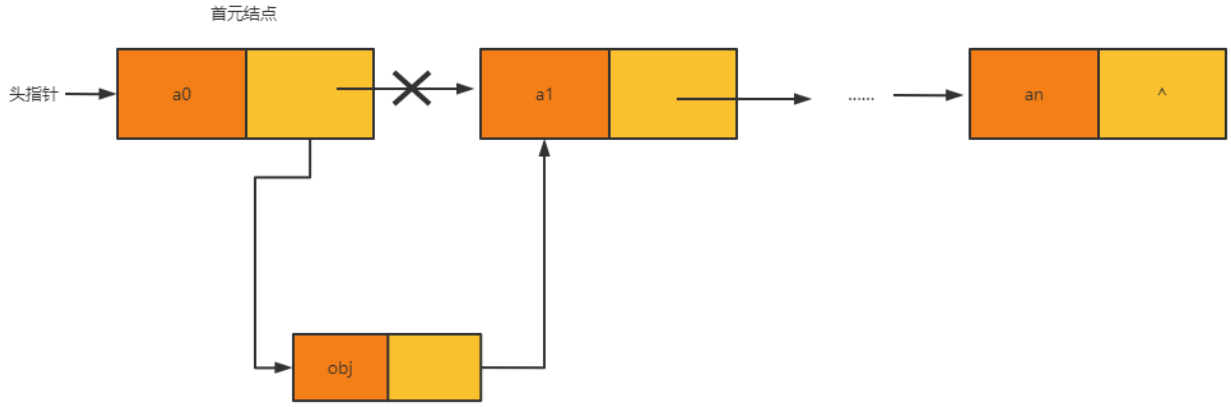


非空链表 带头结点的单链表



不要带头结点的单链表

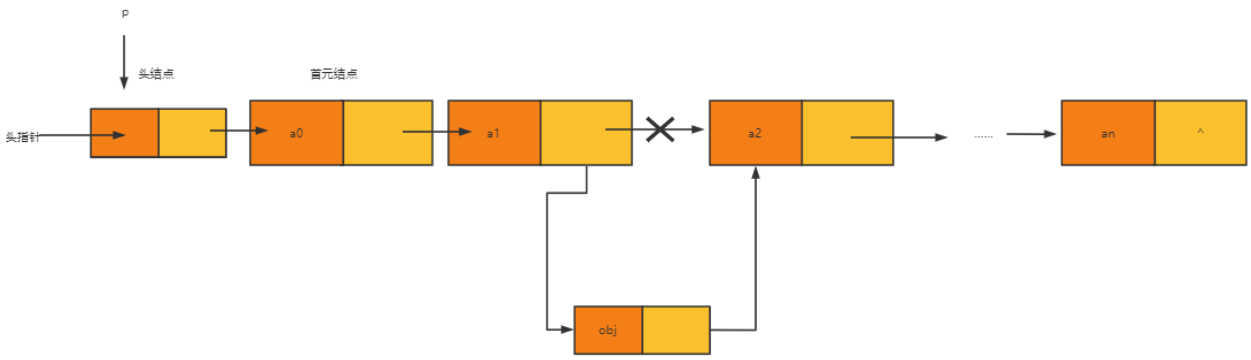
2、不带头结点的单链表的插入操作



上图中，是不带头结点的单链表的插入操作。如果我们在非第一个结点之前做插入操作，我们只需要把新结点的指针域指向a（i），然后将a（i - 1）的指针域指向新的结点。如果我们在第一个结点之前进行插入操作，那么头指针就要等于新插入的节点，这和在非第一个数据元素结点前插入时的情况不同。而且，还有一些不同的情况需要我们做考虑。

所以，当我们设计此类链表的时候，就要分别设计实现方法。

3、带头结点的单链表的插入操作



上图中，如果采用带头结点的单链表结构，算法实现的时候，p指向头结点，改变p的指针的next的值，就可以了，而头指针head的值不变。

因此，算法的实现方法比较简单，其操作与对其他结点的操作统一。

带头结点的好处就是，方便对于链表的操作。对于空表、非空表的情况以及对于首元结点都可以进行统一的处理。