链表

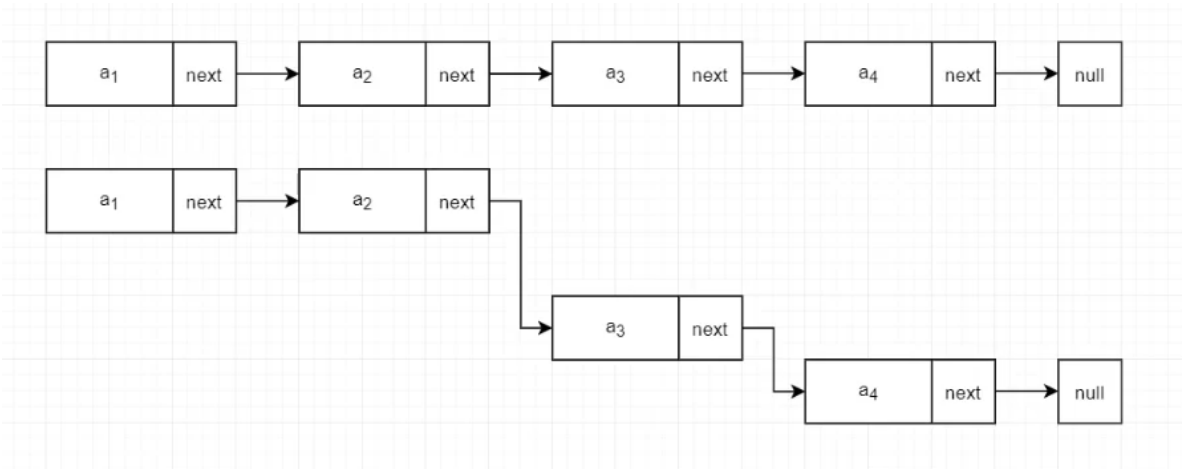
为什么需要链表

顺序表的构建需要**预先知道数据大小来申请连续的存储空间**，而在进行**扩充**时又需要进行数据的搬迁，**所以使用起来并不是很灵活。**

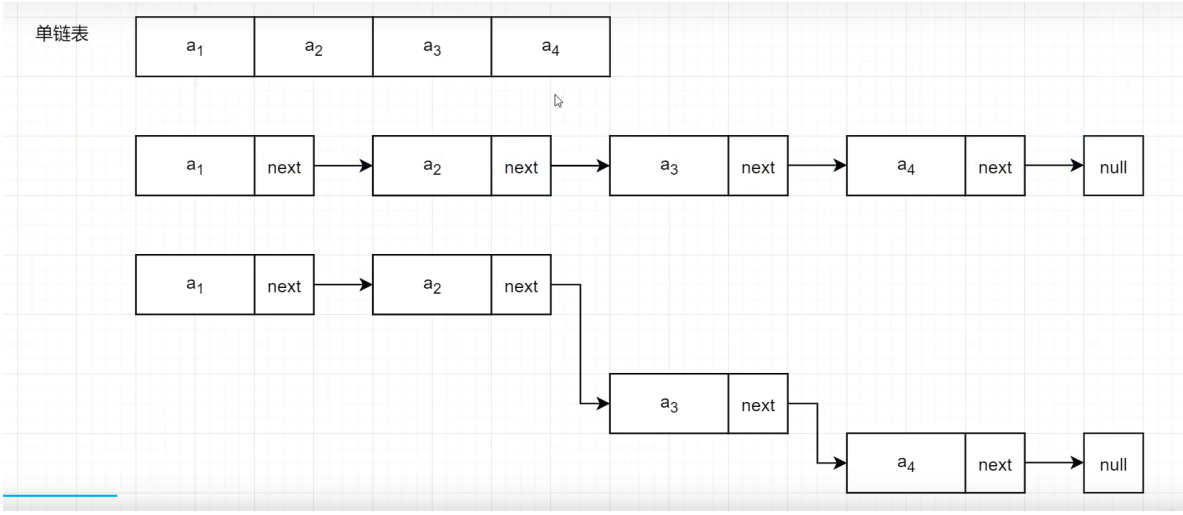
链表结构可以充分利用计算机内存空间，实现灵活的**内存动态**管理。

链表的定义

链表(Linked list)是一种常见的基础数据结构，是一种**线性表**，但是不像顺序表一样连续存储数据，而是在每-个节点 (数据存储单元) 里存放下一个节点的**位置信息** (即地址) 。



列表与链表的区别：



单项链表

单向链表也叫单链表，是链表中最简单的一种形式，它的每个节点包含两个域，一个信息域(元素域)和一个链接域。这个链接指向链表中的下一个节点，而最后一个节点的链接域则指向一个空值。

- 表元素域 elem 用来存放具体的数据。
- 链接域 next 用来存放下一个节点的位置 (python中的标识).
- 变量 p 指向链表的头节点 (首节点)的位置，从 p 出发能找到表中的任意节点.

节点实现

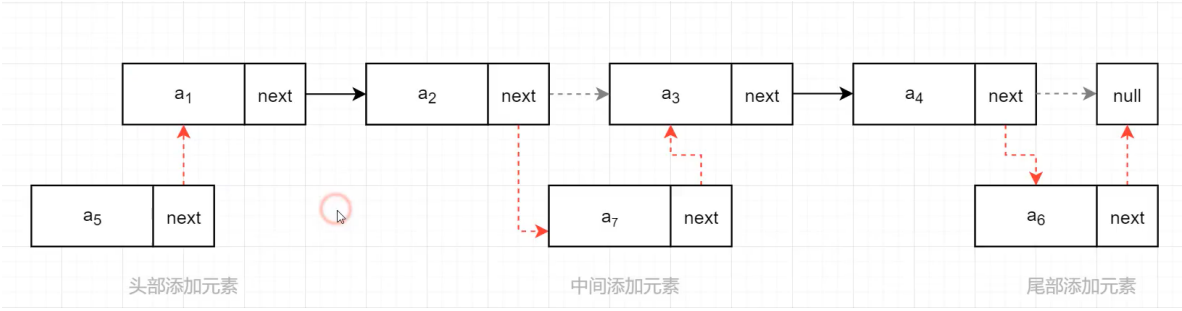
```python
class SingleNode(object):
    """单链表的结点"""
    def __init__(self,item):
        # item存放数据元素
        self.item = item
        # next是下一个节点的标识
        self.next = None
```

单链表的操作

- is_empty()链表是否为空
- length()链表长度
- travel（）遍历整个链表
- add(item)链表头部添加元素
- append(item)链表尾部添加元素
- insert(pos,item)指定位置添加元素
- remove(item)删除节点
- search(item) 查找节点是否存在

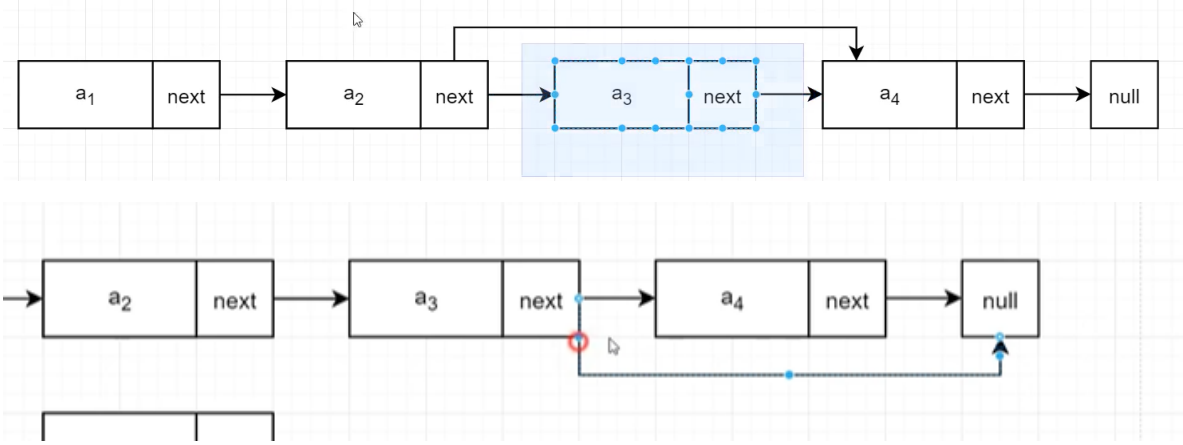
单链表的实现

添加



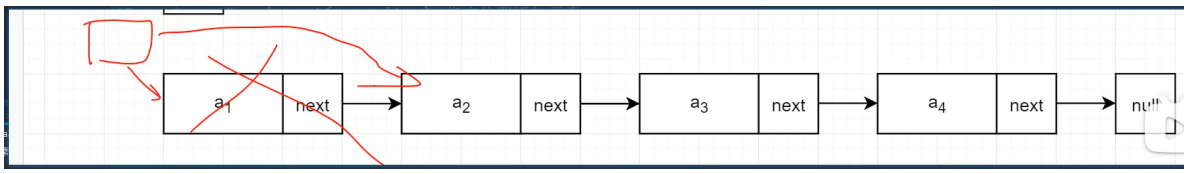
https://github.com/zzqnot996/coding/blob/main/%E5%8D%95%E5%90%91%E9%93%BE%E8%A1%A8_%E6%B7%BB%E5%8A%A0.ipynb

单向链表-查询删除插入等

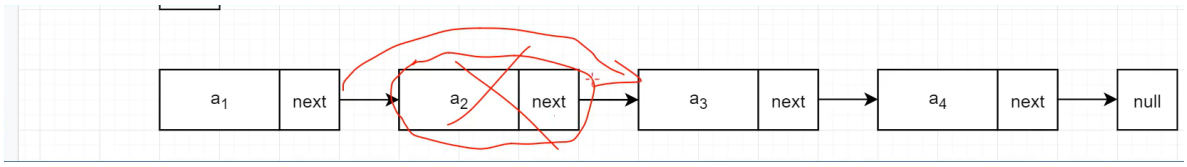


如果找到的是头部：

self.__head直接指向第二个

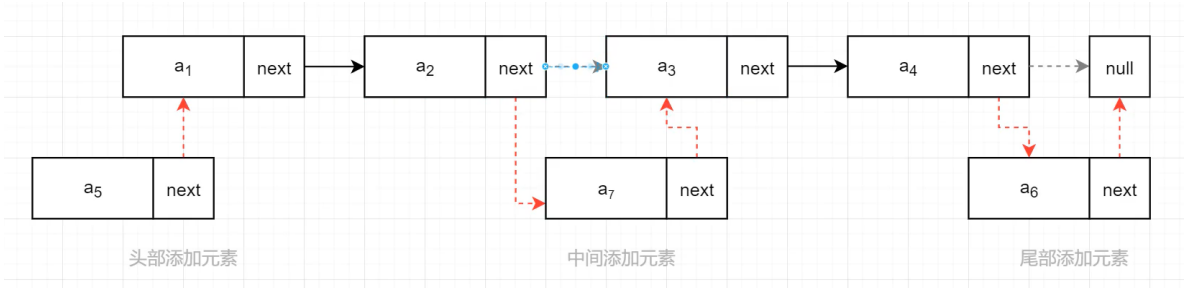


如果不是头部：



插入元素

插入可能插入到前面，插入到后面，可能是0-n



链表与顺序列表的对比

链表失去了顺序表随机读取的优点，同时链表由于增加了结点的指针域，空间开销比较大，但对存储空间的使用要相对灵活。

链表与顺序表的各种操作复杂度如下所示：

操作	链表	顺序表
访问元素	O(n)	O(1)
在头部插入/删除	O(1)	O(n)
在尾部插入/删除	O(n)	O(1)
在中间插入/删除	O(n)	O(n)

注意虽然表面看起来复杂度都是 O(n)，但是链表和顺序表在插入和删除时进行的是完全不同的操作。链表的主要耗时操作是遍历查找，删除和插入操作本身的复杂度是O(1)。顺序表查找很快，主要耗时的操作是拷贝覆盖。因为除了目标元素在尾部的特殊情况，顺序表进行插入和删除时需要对操作点之后的所有元素进行前后移位操作，只能通过拷贝和覆盖的方法进行。

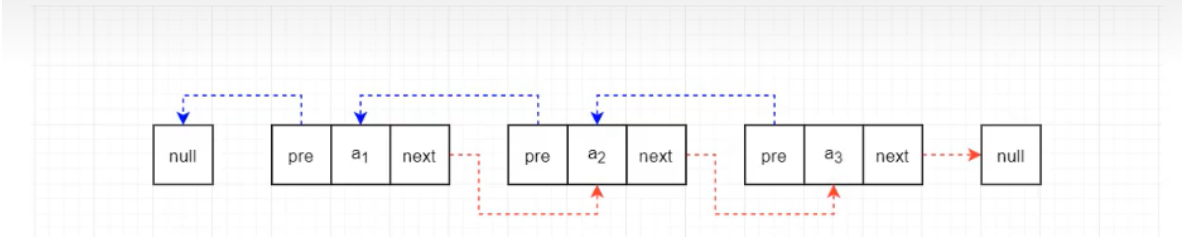
顺序表在头部添加，会把所有的往后挪一遍.

以上代码：

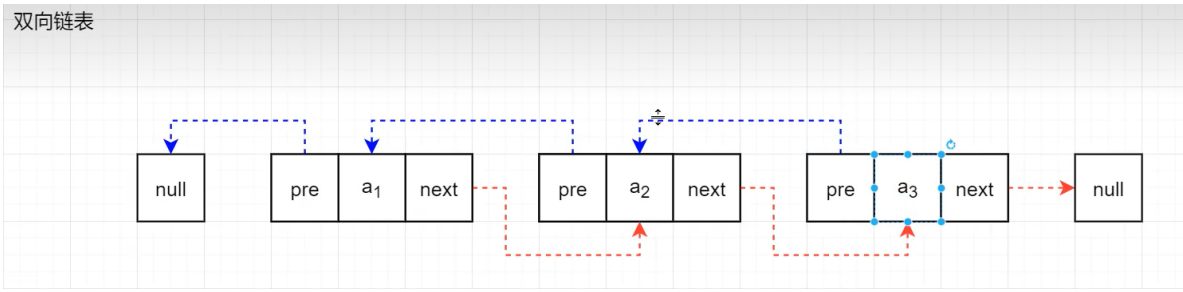
https://github.com/zzqnot996/coding/blob/main/%E5%8D%95%E5%90%91%E9%93%BE%E8%A1%A8.ipynb

双向链表

一种更复杂的链表是"双向链表"或"双面链表"。每个节点有两个链接：一个指向前一个节点，当此节点为第一个节点时，指向空值；而另一个指向下一个节点，当此节点为最后一个节点时，指向空值。



双向链表

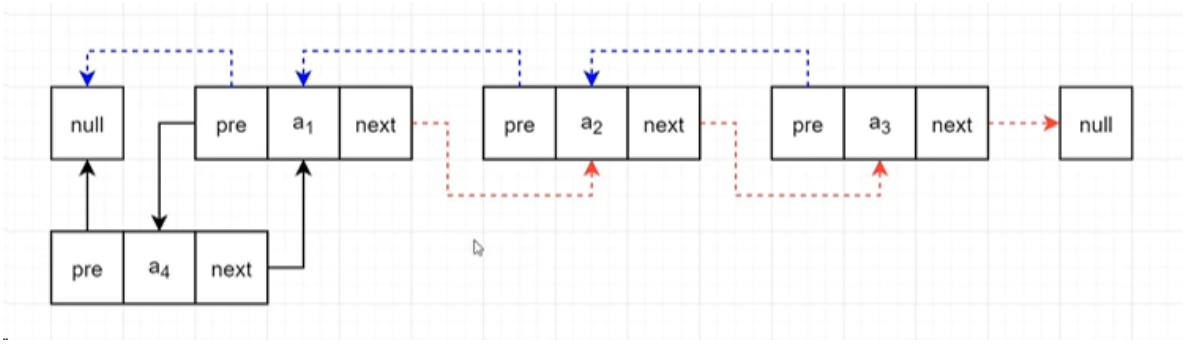


查找方便

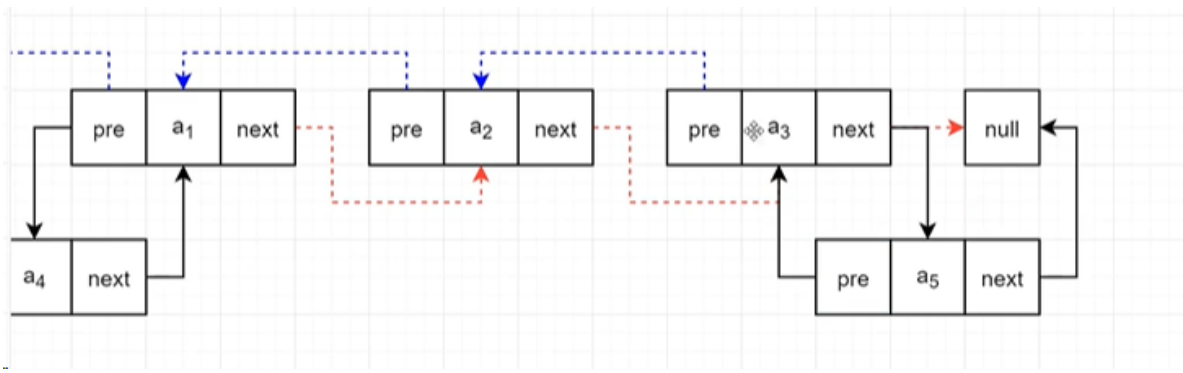
操作

- is_empty0)链表是否为空
- length)链表长度
- travel0) 遍历键表
- add(item)链表头部添加
- append(item)链表尾部添加
- insert(pos,item)指定位置添加
- remove(item)制除节点
- search(item)查找节点是否存在

add



append



以上代码

https://github.com/zzqnot996/coding/blob/main/%E5%8F%8C%E5%90%91%E9%93%BE%E8%A1%A8.ipynb