顺序表

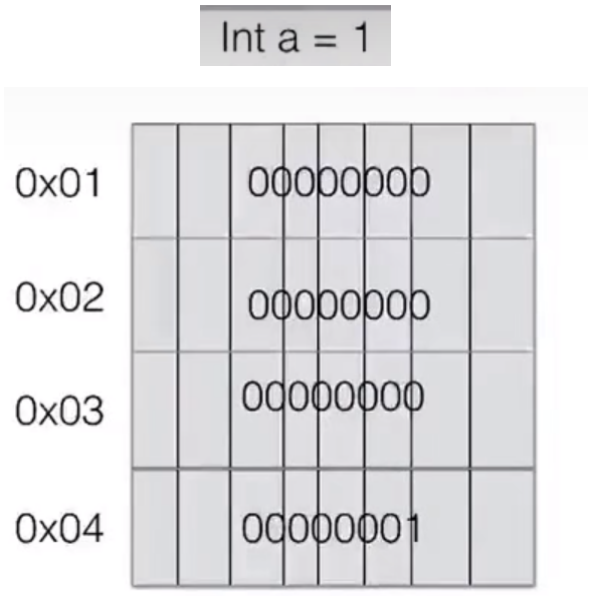
https://www.bilibili.com/video/BV1e4411s7Kw?p=9&spm_id_from=pageDriver&vd_source=e1de9f6d02128b9c85f5fdd03c7e72fc

顺序表的形式

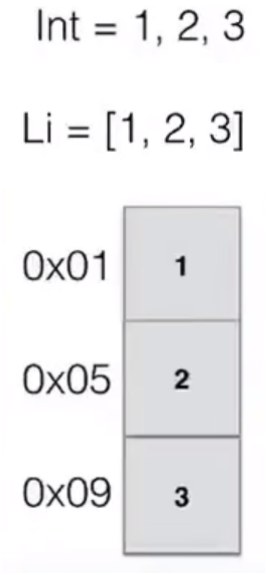
基本顺序表与元素外围顺序表

内存存储

int 占据 4个字节

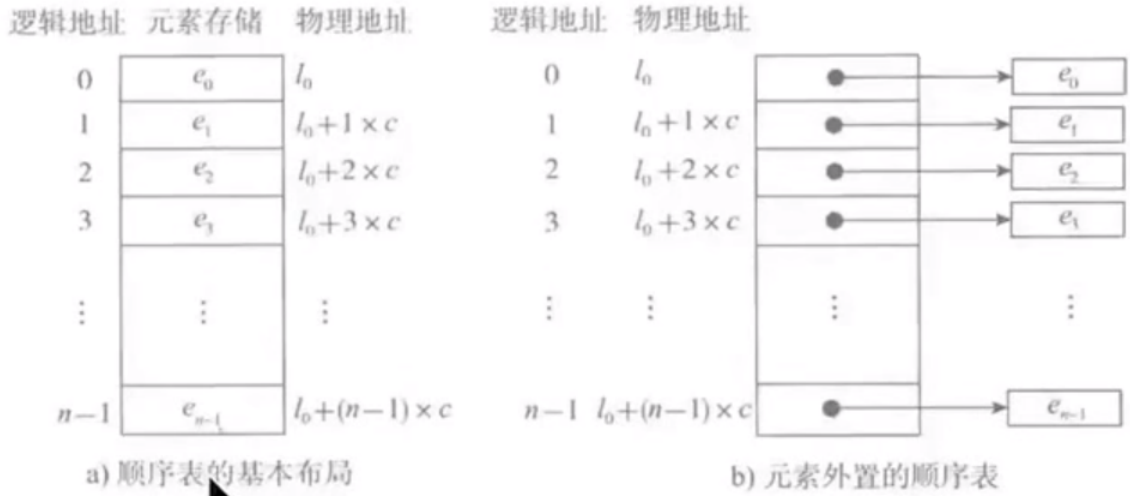


这个int类型整体取出，如果是四个char类型的就涉及到怎么计算机取出。



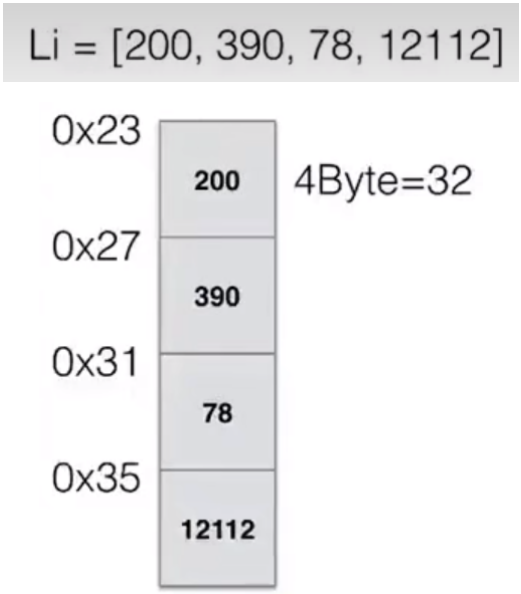
以上连续存储，可以通过第一个来查找所有的，因为数据类型确定，存储方式确定，则已知每个占据多少空间，就可以索引。

一组数据，相同类型怎样存储。最直观 存储在一起。

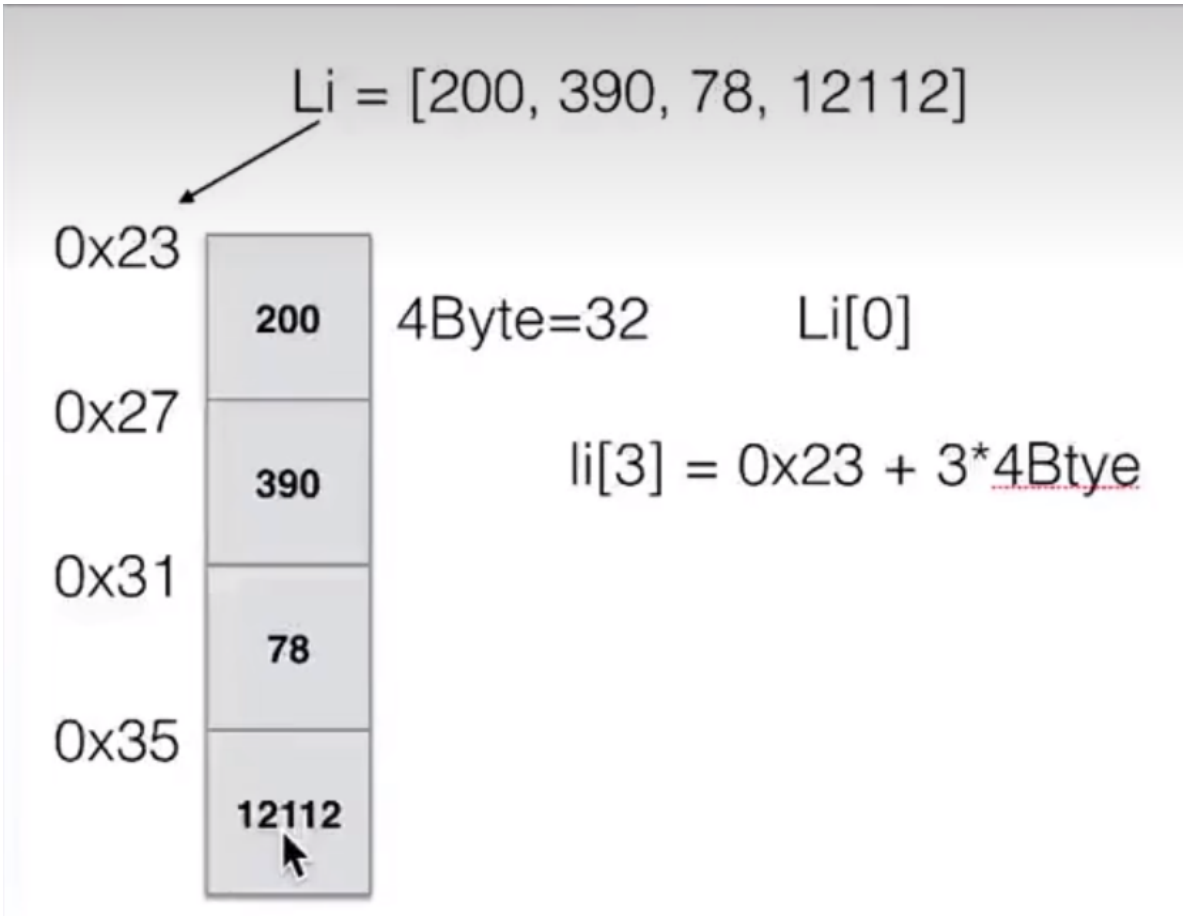


内存、类型本质、连续存储

计算机表示最小按照字节



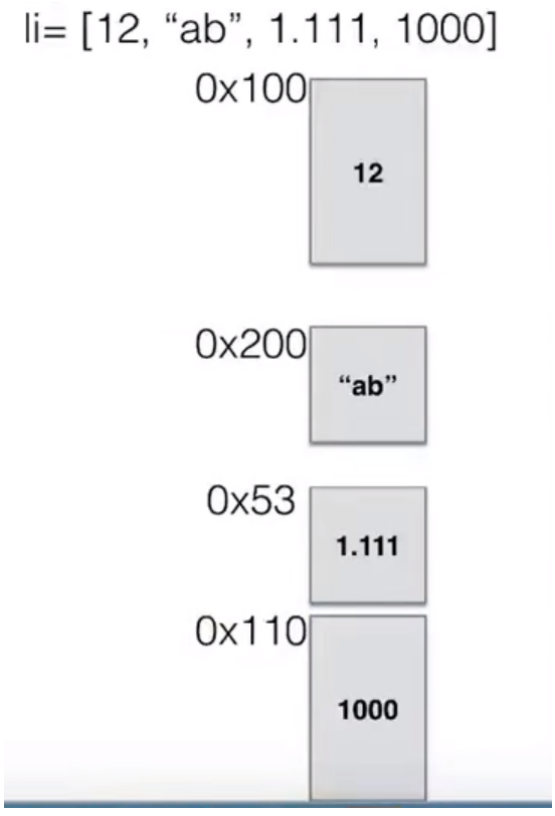
这时，Li指向第一个元素的地址0x23，



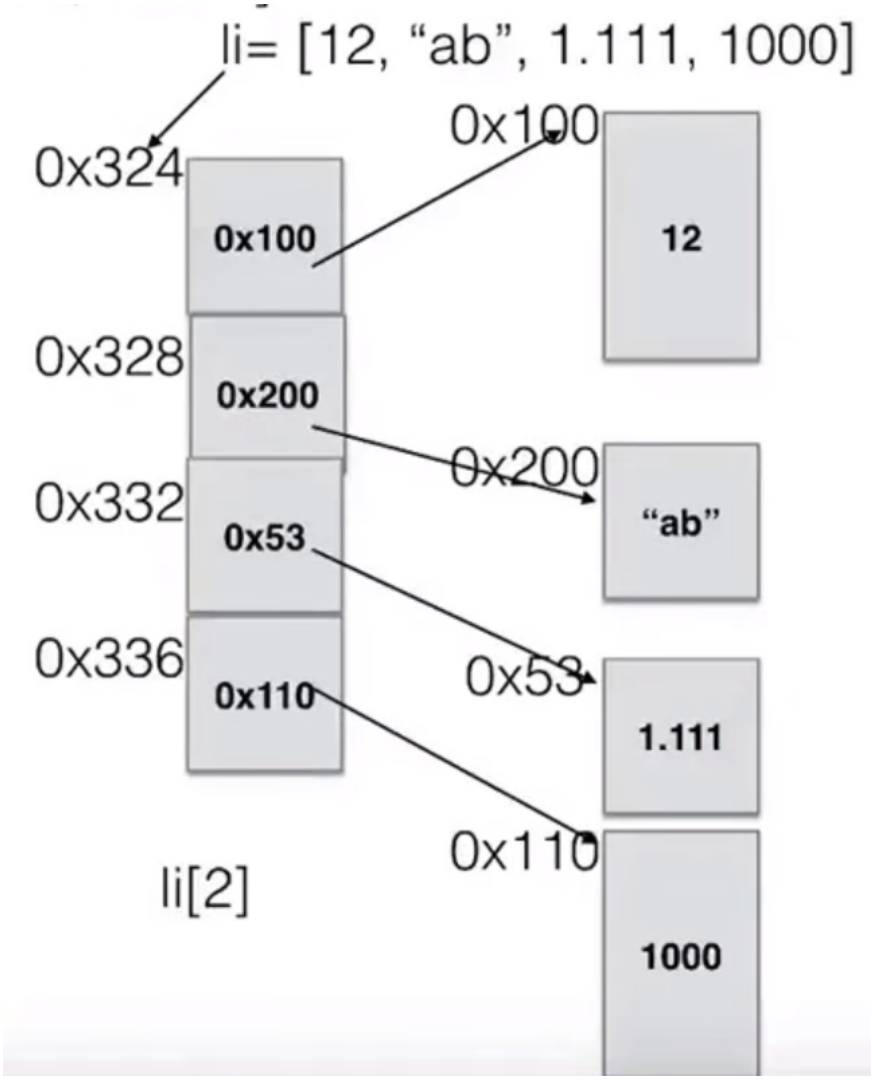
类型不同的时候

```
li= [12, “ab”]
```

在存储不同的数据的时候所占据的存储空间不一样



数据随机存储，数据所在的地址顺序地址，通过地址进行指向



通过元素外围顺序表存储不同类型数据

地址存储的地址是顺序的，根据访问地址顺序取数据

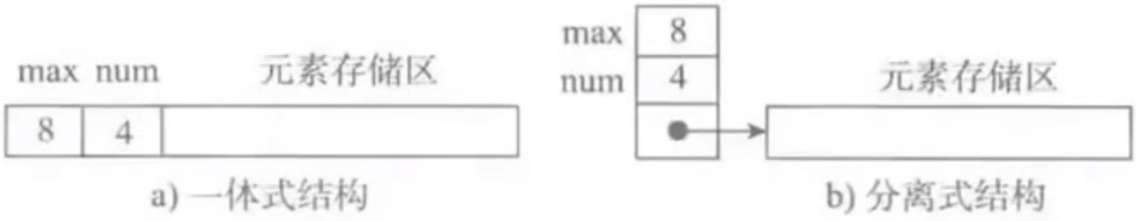
顺序表的结构与实现

顺序表的结构

一个顺序表的完整信息包括两部分，一部分是表中的元素集合，另一部分是为实现正确操作而需记录的信息，即有关表的整体情况的信息，这部分信息主要包括**元素存储区的容量**和**当前表中已有的元素个数**两项

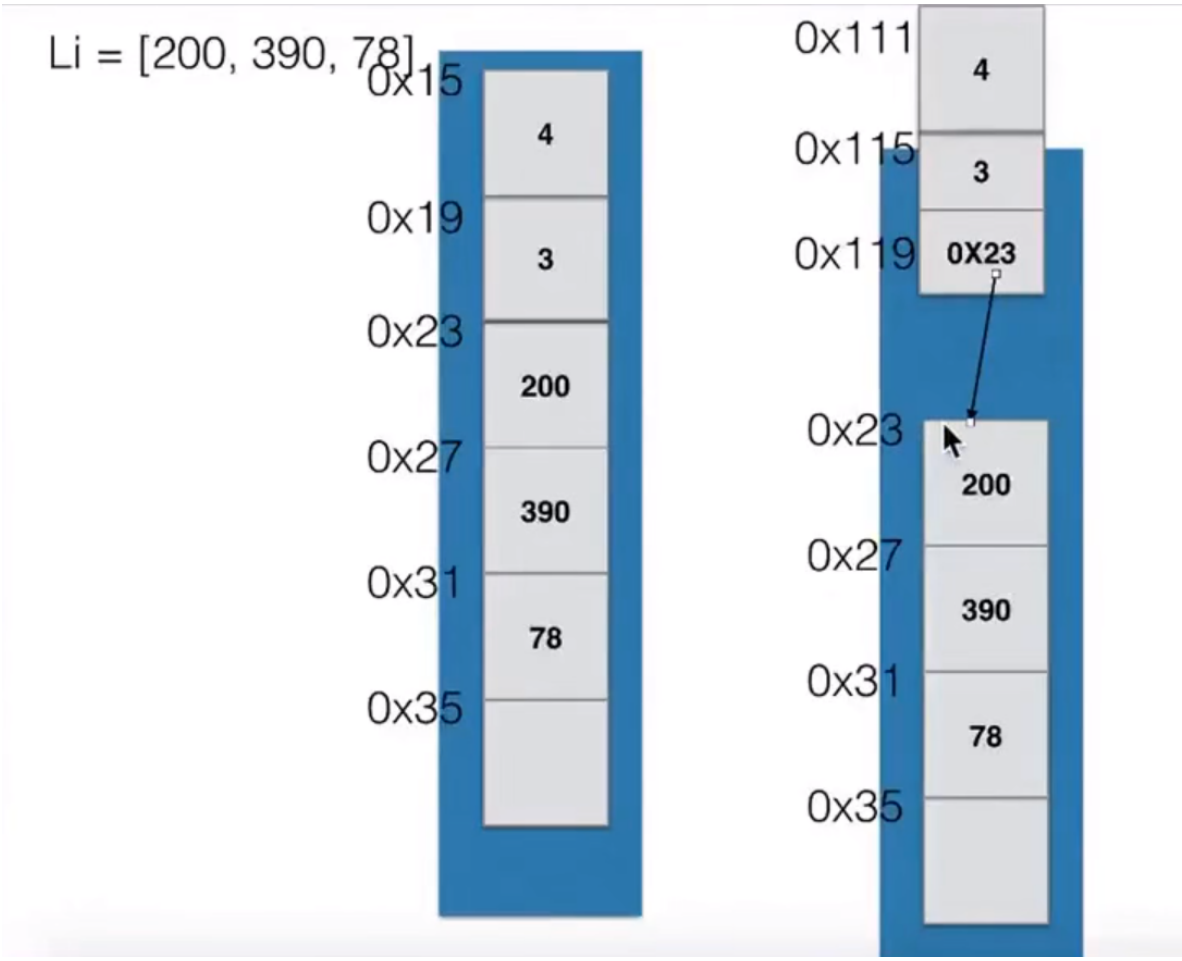
顺序表的两种基本实现形式

顺序表的一体式结构与分离式结构



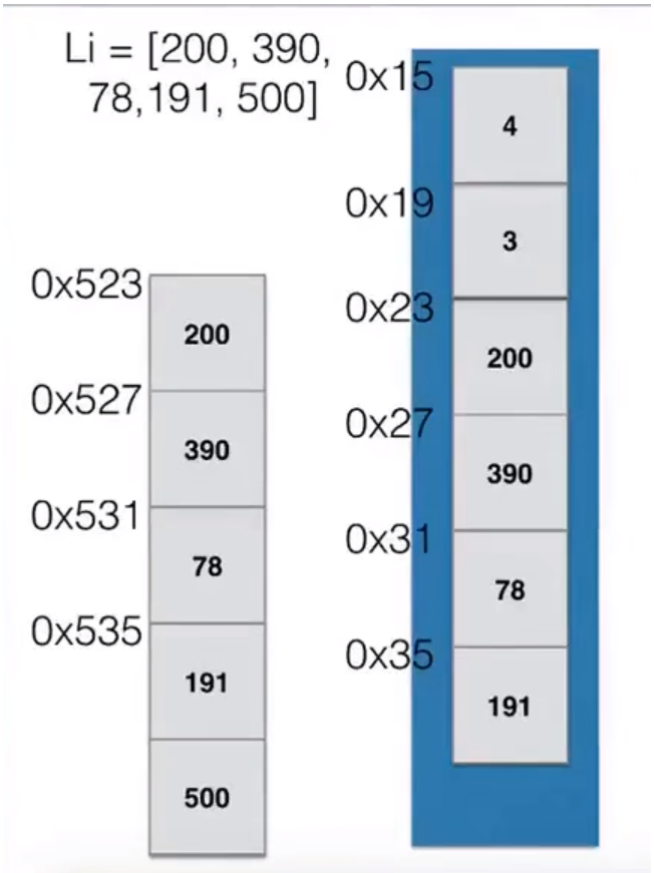
图a为一体式结构，存储表信息的单元与元素存储区以连续的方式安排在一块存储区里，两部分数据的整体形成一个完整的顺序表对象。
一体式结构整体性强，易于管理。但是由于数据元素存储区域是表对象的一部分，顺序表创建后，元素存储区就固定了。

图b为分离式结构，表对象里只保存与整个表有关的信息(即容量和元素个数)，实际数据元素存放在另一个独立的元素存储区里，通过链接与基本表对象关联.



连续存储可通过地址偏移量访问，分离式涉及间接访问

连续存储如果再想申请，需要重新申请数据内存，进行数据搬迁。
表头需要改变新的地址。



扩充分离式不需要表头的替换。

一般使用分离式

元素存储区替换

一体式结构由于顺序表信息区与数据区连续存储在一起，所以若想更换数据区，则只能整体搬迁，即整个顺序表对象(指存储顺序表的结构信息的区域) 改变了。

分离式结构若想更换数据区，只需将表信息区中的数据区链接地址更新即可，而该顺序表对象不变。

元素存储区扩充

采用分离式结构的顺序表，若将数据区更换为存储空间更大的区域，则可以在不改变表对象的前提下对其数据存储区进行了扩充，所有使用这个表的地方都不必修改。只要程序的运行环境(计算机系统)还有空闲存储，这种表结构就不会因为满了而导致操作无法进行。人们把采用这种技术实现的顺序表称为动态顺序表因为其容量可以在使用中动态变化。

扩充的两种策略

- 每次扩充增加固定数目的存储位置，如每次扩充增加10个元素位置，这种策略可称为线性增长。

  特点:节省空间，但是扩充操作频繁，操作次数多

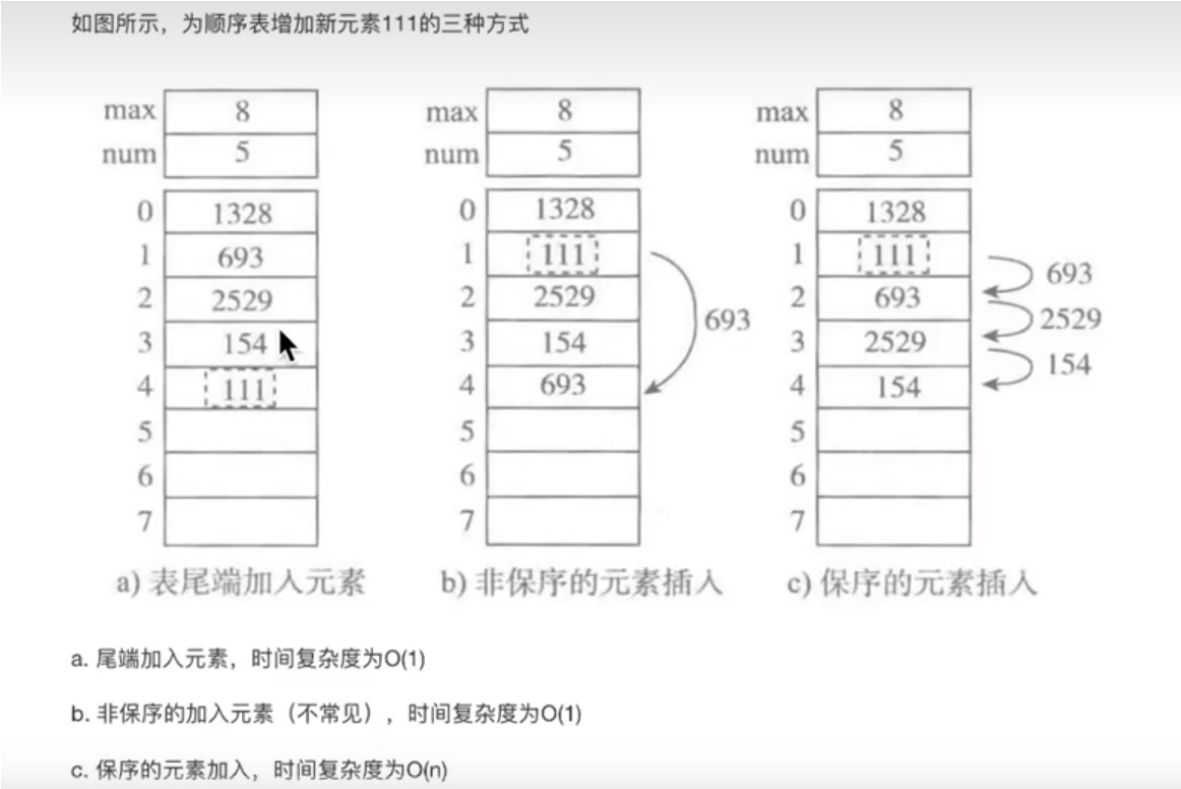
- 每次扩充容量加倍，如每次扩充增加一倍存储空间

  特点:减少了扩充操作的执行次数，但可能会浪费空间资源。**以空间换时间，推荐的方式**

顺序表的操作

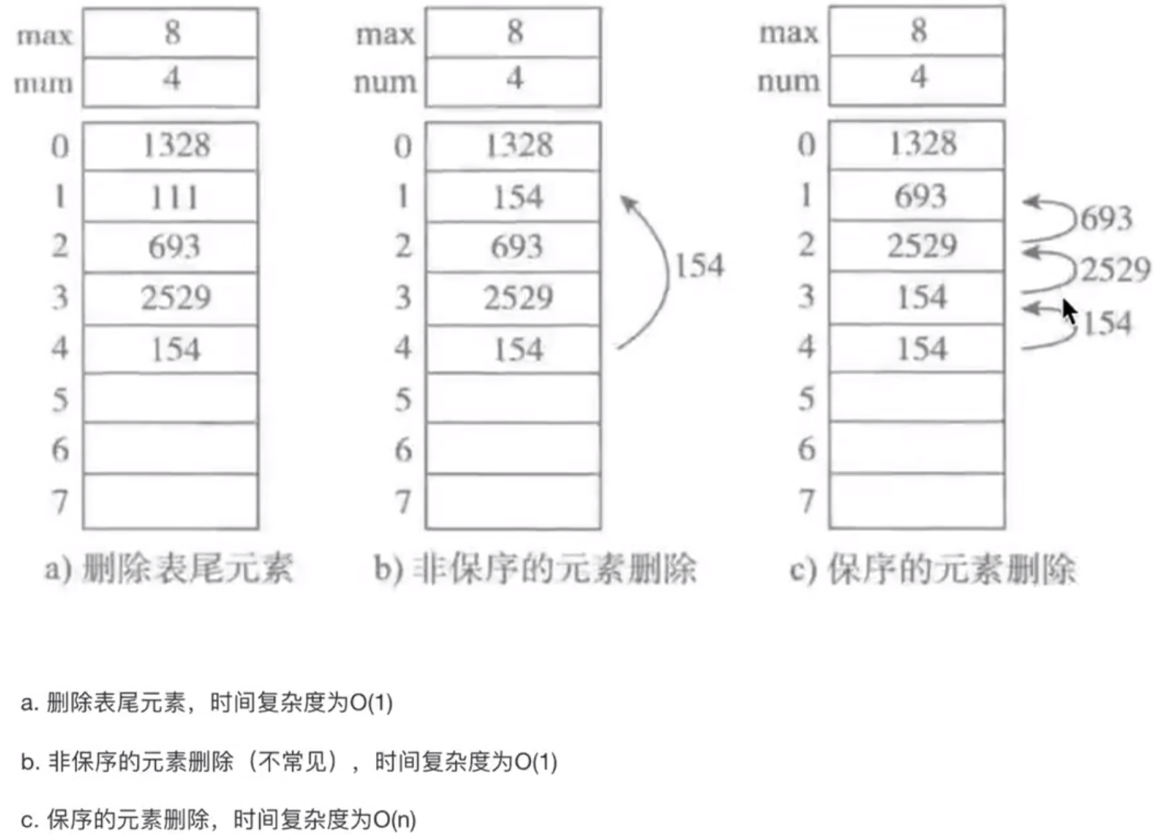
https://www.bilibili.com/video/BV1G4411h7BA/?p=13&vd_source=e1de9f6d02128b9c85f5fdd03c7e72fc

增加元素



保序： 为了插入一个元素，其他元素位置顺序不变

删除元素



Python中的顺序表

Python中的**list和tuple两种类型**采用了顺序表的实现技术，具有前面讨论的顺序表的所有性质.

tuple是不可变类型，即不变的顺序表，因此不支持改变其内部状态的任何操作，而其他方面，则与list的性质类似。

list的基本实现技术

Python标准类型list就是一种元素个数可变的线性表，可以加入和删除元素，并在各种操作中维持已有元素的顺序(即保序)，而且还具有以下行为特征:

- 基于下标(位置)的高效元素访问和更新，时间复杂度应该是O(1);

  为满足该特征，应该采用顺序表技术，表中元素保存在一块连续的存储区中。

- 允许任意加入元素，而且在不断加入元素的过程中，表对象的标识 (函数id得到的值)不变。

  为满足该特征，就必须能更换元素存储区，并且为保证更换存储区时list对象的标识id不变，只能采用分离式实现技术。

在Python的官方实现中，**list就是一种采用分离式技术实现的动态顺序表。**这就是为什么用list.append(x)(或 list.insert(len(list),x)，即尾部插入) 比在指定位置插入元素效率高的原因.

在Python的官方实现中，list实现采用了如下的策略:

在建立空表 (或者很小的表)时，系统分配一块能容纳8个元素的存储区;在执行插入操作 (insert或append)时，如果元素存储区满就换一块4倍大的存储区。但如果此时的表已经很大(目前的阀值为50000)，则改变策略，采用加一倍的方法。引入这种改变策略的方式，是为了避免出现过多空闲的存储位置。

list内置操作的时间复杂度

Operation	Big-O Efficiency
indexx[]	$O(1)$
index assignment	$O(1)$
append	$O(1)$
pop()	$O(1)$
pop(i)	$O(n)$
insert(i,item)	$O(n)$
del operator	$O(n)$
iteration	$O(n)$
contains (in)	$O(n)$
get slice [x:y]	$O(k)$
del slice	$O(n)$
set slice	$O(n+k)$
reverse	$O(n)$
concatenate	$O(k)$
sort	$O(n \log n)$
multiply	$O(nk)$

Table 2.2: Big-O Efficiency of Python List Operators