



1 数组

1.1.1 读取

1. 数组包含的四种基本操作：读取、查找、插入、删除
2. 操作速度是按照步数进行计算的
3. 读取的一些特点

计算机之所以在读取数组中某个索引所指的值得时，能直接跳到那个位置上，是因为它具备以下条件。

(1) 计算机可以一步就跳到任意一个内存地址上。（就好比，要是你知道大街 123 号在哪儿，那么就可以直奔过去。）

(2) 数组本身会记有第一个格子的内存地址，因此，计算机知道这个数组的开头在哪里。

(3) 数组的索引从 0 算起。

这些特点是读取操作的前提 一个读取的例子

1000	1001	1002	1003	1004	1005	1006	1007	1008	1009
1010	1011	1012	1013	1014	1015	1016	1017	1018	1019
1020	1021	1022	1023	1024	1025	1026	1027	1028	1029
1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
1040	1041	1042	1043	1044	1045	1046	1047	1048	1049
1050	1051	1052	1053	1054	1055	1056	1057	1058	1059
1060	1061	1062	1063	1064	1065	1066	1067	1068	1069
1070	1071	1072	1073	1074	1075	1076	1077	1078	1079
1080	1081	1082	1083	1084	1085	1086	1087	1088	1089
1090	1091	1092	1093	1094	1095	1096	1097	1098	1099

购物清单数组的索引和内存地址，如下图所示。

	"apples"	"bananas"	"cucumbers"	"dates"	"elderberries"
内存地址:	1010	1011	1012	1013	1014
索引:	0	1	2	3	4

~~~~~

回到刚才的例子，当我们叫计算机读取索引 3 的值时，它会做以下演算。

- (1) 该数组的索引从 0 算起，其开头的内存地址为 1010。
- (2) 索引 3 在索引 0 后的第 3 个格子上。
- (3) 于是索引 3 的内存地址为 1013，因为  $1010 + 3 = 1013$ 。

当计算机一步跳到 1013 时，我们就能获取到"dates"这个值了。

读取只需要用一步就可以了

## 1.1.2 查找

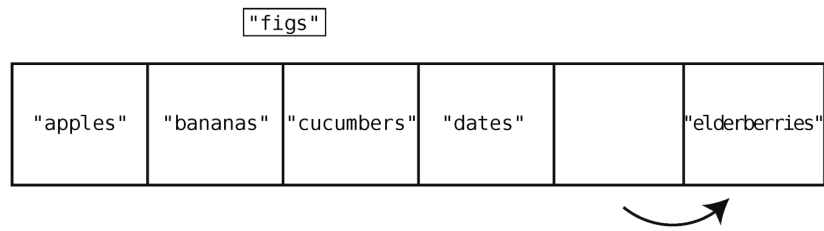
1. 这种逐个格子去检查的做法，就是最基本的查找方法——**线性查找**。
2. 线性查找需每个都检查一遍

如果我们要找的值刚好在数组的最后一个格子里（如本例的 `elderberries`），那么计算机从头到尾检查每个格子，会在最后才找到。同样，如果我们要找的值并不存在于数组中，那么计算机也还是得查遍每个格子，才能确定这个值不在数组中。

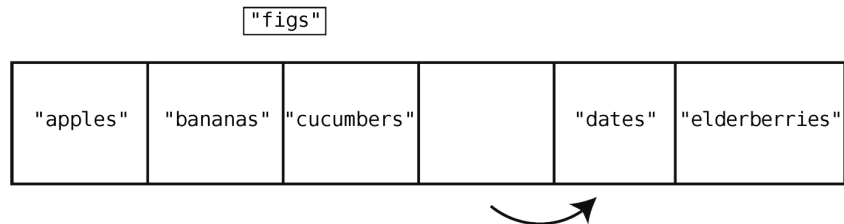
3. 一个  $N$  格的数组，其线性查找的最多步数是  $N$ （ $N$  可以是任何自然数）
4. 无论是多长的数组，查找都比读取要慢，因为读取永远都只需要一步，而查找却可能需要多步

## 1.1.3 插入

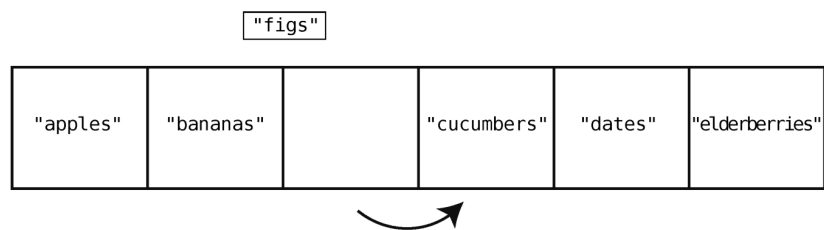
1. 在数组中插入新元素的速度取决于插入的**位置**。 比如在末尾插入，只需要一步。因为计算机知道数组开头的内存地址，一共包含多少元素。
2. 一个数组插入的例子：把figs插入到索引2



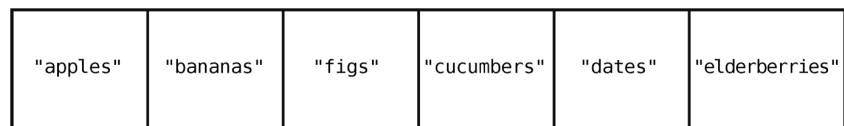
第 2 步: "date"右移。



第 3 步: "cucumbers"右移。



第 4 步: 至此, 可以在索引 2 处插入"figs"了。



如上所示, 整个过程有 4 步, 开始 3 步都是在移动数据, 剩下 1 步才是真正的插入数据。

3. 插入在数组开头花费的步数最多 (最低效), 需要把所有的元素向右移动。
4. 数组, 含有N个元素, 插入数据最坏的情况需要N+1步。(插入在数组开头)

## 1.1.4 删除

1. 删除: 在删掉数据后, 要把数组向左移, 填补空格
2. 一个删除的例子

我们找回最开始的那个数组，删除索引 2 上的值，即"cucumbers"。


第 1 步：删除"cucumbers"。

|          |           |  |         |                |
|----------|-----------|--|---------|----------------|
| "apples" | "bananas" |  | "dates" | "elderberries" |
|----------|-----------|--|---------|----------------|

虽然删除"cucumbers"好像一步就搞定了，但这带来了新的问题：数组中间空出了一个格子。因为数组中间是不应该有空格的，所以，我们得把"dates"和"elderberries"往左移。


第 2 步：将"dates"左移。

|          |           |         |  |                |
|----------|-----------|---------|--|----------------|
| "apples" | "bananas" | "dates" |  | "elderberries" |
|----------|-----------|---------|--|----------------|



第 3 步：将"elderberries"左移。

|          |           |         |                |  |
|----------|-----------|---------|----------------|--|
| "apples" | "bananas" | "dates" | "elderberries" |  |
|----------|-----------|---------|----------------|--|



结果，整个删除操作花了 3 步。其中第 1 步是真正的删除，剩下的 2 步是移数据去填空格。所以，删除本身只需要 1 步，但接下来需要额外的步骤将数据左移以填补删除所带来的空隙。

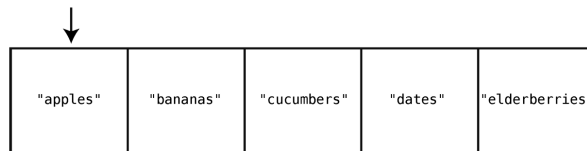
3. 数组含有N个元素，删除最多需要N步（删除第一个）。

## 1.2 集合

1. 集合中不允许元素重复
2. 读取、查找、删除操作都和数组一致。插入需要先查询是不是有重复值
3. 一个集合中插入操作的例子

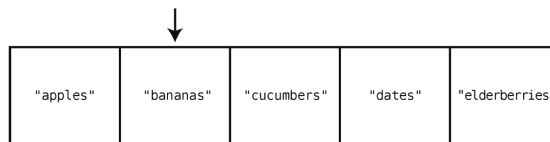
假设我们的购物清单是一个集合——用集合还是不错的，毕竟你不会想买重复的东西。如果当前集合是["apples", "bananas", "cucumbers", "dates", "elderberries"]，然后想插入"figs"，那么就需要做一次如下的查找。

第 1 步：检查索引 0 有没有"figs"。

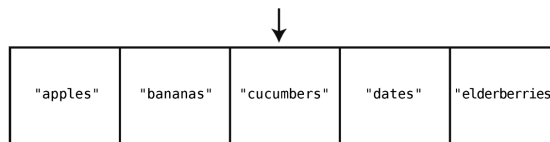


没有，不过说不定其他索引会有。为了在真正插入前确保它不存在于任何索引上，我们继续。

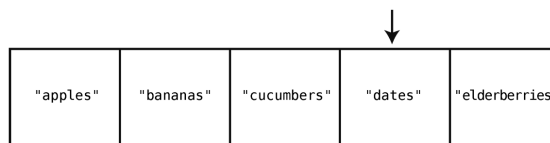
第 2 步：检查索引 1。



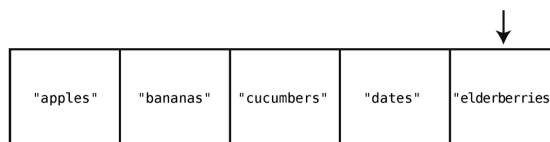
第 3 步：检查索引 2。



第 4 步：检查索引 3。



第 5 步：检查索引 4。



直到检查完整个集合，才能确定插入"figs"是安全的。于是，到最后一步。

第 6 步：在集合末尾插入"figs"。

4. 在N个元素的集合中进行插入，最好的情况需要 $N+1$ 步（N步确认插入的值不在集合中，然后在最后插入）。最坏的情况是在开头插入，用N检查是否重复，N把后面的值右移，放在开头1步。最多需要 $2N+1$ 步。