



# 数据结构与算法 (Python版)

## 顺序查找算法及分析

陈斌 北京大学 [gischen@pku.edu.cn](mailto:gischen@pku.edu.cn)

# 顺序查找Sequential Search

- ❖ 如果数据项保存在如列表这样的集合中，我们会称这些数据项具有线性或者顺序关系。
- ❖ 在Python List中，这些数据项的存储位置称为下标 (index)，这些下标都是有序的整数。
- ❖ 通过下标，我们就可以按照顺序来访问和查找数据项，这种技术称为“顺序查找”

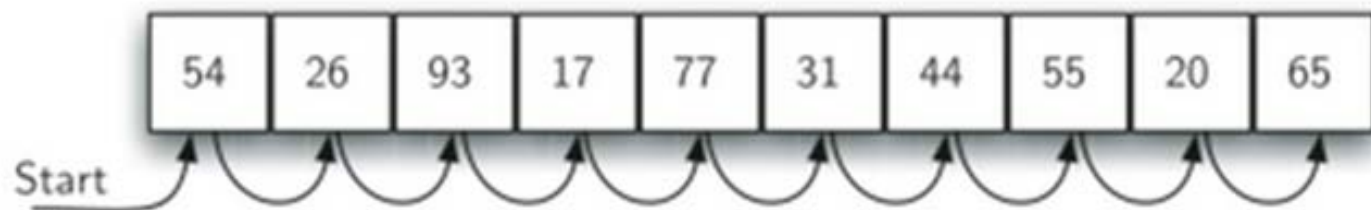
# 顺序查找Sequential Search

## ❖ 要确定列表中是否存在需要查找的数据项

首先从列表的第1个数据项开始，

按照下标增长的顺序，逐个比对数据项，

如果到最后一个都未发现要查找的项，那么查找失败。



# 顺序查找：无序表查找代码

```
def sequentialSearch(alist, item):  
    pos = 0  
    found = False  
  
    while pos < len(alist) and not found:  
        if alist[pos] == item:  
            found = True  
        else:  
            pos = pos+1  
  
    return found
```

下标顺序增长

```
testlist = [1, 2, 32, 8, 17, 19, 42, 13, 0]  
print(sequentialSearch(testlist, 3))  
print(sequentialSearch(testlist, 13))
```

# 顺序查找：算法分析

- ❖ 要对查找算法进行分析，首先要确定其中的**基本计算步骤**。
- ❖ 回顾第二章算法分析的要点，这种基本计算步骤必须要**足够简单**，并且在算法中**反复执行**
- ❖ 在查找算法中，这种基本计算步骤就是进行数据项的**比对**  
当前数据项等于还是不等于要查找的数据项，**比对的次数决定了算法复杂度**

# 顺序查找：算法分析

❖ 在顺序查找算法中，为了保证是讨论的一般情形，需要假定列表中的数据项并没有按值排列顺序，而是**随机放置**在列表中的各个位置

换句话说，数据项在列表中各处出现的概率是相同的



# 顺序查找：算法分析

- ❖ 数据项是否在列表中，比对次数是不一样的
- ❖ 如果数据项不在列表中，需要比对所有数据项才能得知，比对次数是 $n$
- ❖ 如果数据项在列表中，要比对的次数，其情况就较为复杂
  - 最好的情况，第1次比对就找到
  - 最坏的情况，要 $n$ 次比对

# 顺序查找：算法分析

## ❖ 数据项在列表中，比对的一般情形如何？

因为数据项在列表中各个位置出现的概率是相同的；所以平均状况下，比对的次数是 $n/2$ ；

## ❖ 所以，顺序查找的算法复杂度是 $O(n)$

Case	Best Case	Worst Case	Average Case
item is present	1	$n$	$n/2$
item is not present	$n$	$n$	$n$

## ❖ 这里我们假定列表中的数据项是无序的，那么如果数据项排了序，顺序查找算法的效率又如何呢？



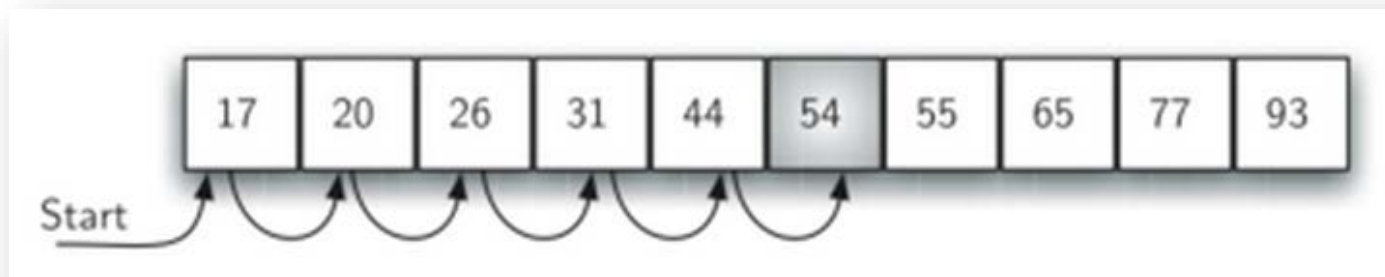
# 顺序查找：算法分析

## ❖ 实际上，我们在第三章的有序表Search方法实现中介绍过**顺序查找**

当数据项存在时，比对过程与无序表完全相同

不同之处在于，如果数据项不存在，比对可以提前结束


- 如下图中查找数据项50，当看到54时，可知道后面不可能存在50，可以提前退出查找



# 顺序查找：有序表查找代码

```
def orderedSequentialSearch(alist, item):
    pos = 0
    found = False
    stop = False
    while pos < len(alist) and not found and not stop:
        if alist[pos] == item:
            found = True
        else:
            if alist[pos] > item:
                stop = True
            else:
                pos = pos+1

    return found
```



```
testlist = [0, 1, 2, 8, 13, 17, 19, 32, 42,]
print(orderedSequentialSearch(testlist, 3))
print(orderedSequentialSearch(testlist, 13))
```

# 顺序查找：算法分析

## ❖ 顺序查找有序表的各种情况分析

Case	Best Case	Worst Case	Average Case
item is present	1	n	$n/2$
item is not present	1	n	$n/2$

- ❖ 实际上，就算法复杂度而言，仍然是 $O(n)$
- ❖ 只是在数据项不存在的时候，有序表的查找能节省一些比对次数，但并不改变其数量级。

