快速排序及其改进

原创 周雅婷 LOA算法学习笔记 2021-01-28 15:40

01 前言

快速排序是20世纪最重要的10大算法思想之一，也是目前公认的最快的排序算法之一，其主要思想是递归与分治，把一个复杂的问题分成多个的相同或相似的子问题，再把子问题分成更小的子问题……直到最后子问题可以简单的直接求解，原问题的解即子问题的解的合并。本文介绍了经典的快速排序与及其两个改进：双路与三路快速排序。

02 经典快速排序

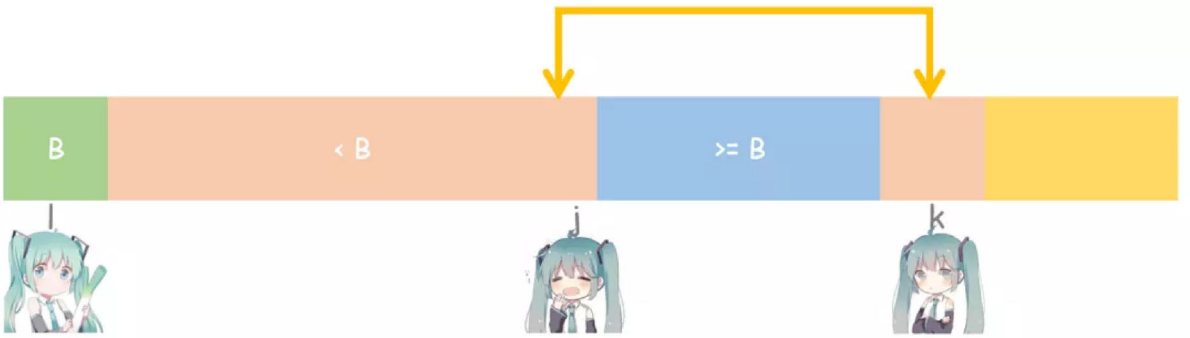
经典的快速排序算法的基本思想是：在序列中选定一个基准值，遍历序列，将序列分为两部分，一部分大于基准值，一部分小于基准值，再递归地对这两部分数据进行快速排序，直到整个序列有序。

图一

如图一所示，随机从序列中选择一个值作为基准值并将其放到序列的开始位置l，声明索引j与k，以闭区间[l+1:j]表示比基准值小的元素，以闭区间[j+1:k-1]表示比基准值大的元素，k表示当前遍历到元素下标，起始时两个区间中应无元素，即初始化j=l,k=l+1，初始状态如图二所示。

图二

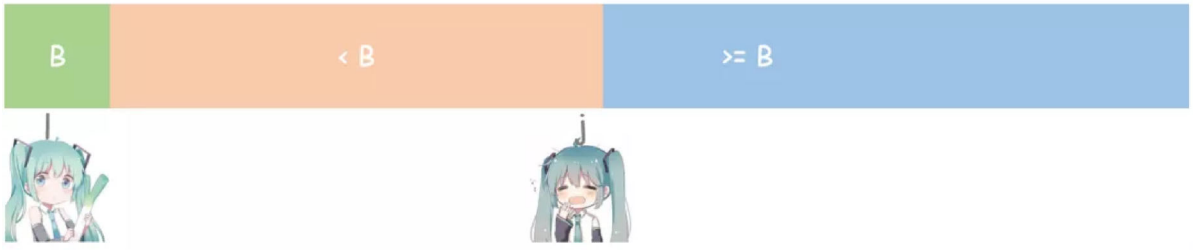
之后以索引k遍历序列，如图三所示，如果array[k]<B，则交换索引j与索引k指向的元素，然后移动k指向下一个元素，否则如图四所示，不需要交换位置，直接移动k至下一个元素。



图三

图四

一次遍历结束后，得到的序列如图五所示，需要再将基准元素放到正确位置，则交换索引i与j指向位置的元素将基准元素放置在正确位置，之后再递归排序大于基准值与小于基准值的两部分至有序即可。



图五

python 代码：

```
    def partition(array, l, r):
        random_index = random.randint( l, r )
        array[l], array[random_index] = array[random_index], array[l]
        basic = array[l]
        j = l     # <basic:[l,j]
        k = l + 1  # >basic:[j+1,k-1]
        for k in range( l + 1, r + 1 ):
            if (array[k] < basic):
                array[j + 1], array[k] = array[k], array[j + 1]
                j += 1
        array[l], array[j] = array[j], array[l]
        return j



    def quickSort(array, l, r):
        if (l > r):
            return;
        j = partition( array, l, r )
        quickSort( array, l, j - 1 )
        quickSort( array, j + 1, r )
```

03 双路快速排序

在经典的快速排序使用过程中，人们也发现了一些问题，比如在面临有很多重复值的情况下，经典的快速排序算法会变得很慢，原因是以其中某个值作为基准值时，由于存在多个重复元素，导致分成的两部分大小不均衡，从而使得算法的复杂度接近 $O(N^2)$ ，为了解决这个问题，就有了双路快速排序，双路快速排序的基本思路是:在序列中选定一个基

准值，建立两个遍历索引j和k，j从头遍历到尾，k从尾遍历到头，将<=基准值的元素保存在j索引的左边，将>=基准值的元素保存在k索引的右边，从而实现了将多个重复元素分散到两部分序列中，避免了两部分大小不均衡的问题。

图六

区间：<=B:[l+1,j] >=B:[k+1,r]
初始化：j=l+1,k=r
操作：如果array[j]<=B,则 $j++$，否则交换array[j]与array[k], $k--$, $j++$
如果array[k]>=B,则 $k--$，否则交换array[j]与array[k], $k--$, $j++$
python 代码：

```
    def partition(array, l, r):
        random_index = random.randint(l,r)
        array[l], array[random_index] = array[random_index], array[l]
        basic = array[l]


        j=l+1   #<=basic:[l+1,j-1]
        k=r     #>=basic:[k+1,r]
        while True:
            while j <= r and array[j] < basic:
                j += 1
            while k >= l + 1 and array[k] > basic:
                k -= 1
            if j > k:
                break


            array[j], array[k] = array[k], array[j]
            j += 1
            k -= 1
        array[k], array[l] = array[l], array[k]
        return k


    def quickSort(array, l, r):
        if l > r :
            return
        j = partition(array, l, r)
        quickSort(array, l, j - 1)
        quickSort(array, j + 1, r)
```

04 三路快速排序

三路快排同样用于处理重复值的问题，只是与双路快排中将序列分成两部分不同，在三路快排中，将序列分为大于基准值，小于基准值与等于基准值三部分，一次递归结束后，所有等于基准值的元素找到了其正确位置，之后只需要再递归处理小于基准值与大于基准值两部分即可。

图七

区间：<B:[l+1,j] >B:[k,r] =B:[j+1,i-1]
初始化：j=l+1,i=l+1,k=r+1
操作：如果array[i]<B,则交换array[j+1]与array[i]，$j++$，$i++$
如果array[i]>B,则交换array[k-1]与array[i]，$k--$
如果array[i]=B,则 $i++$
python 代码：

```
    def partition(array, l, r):
        random_index = random.randint( l, r )
        array[l], array[random_index] = array[random_index], array[l]
        basic = array[l]
        j = l      # <basic:[l+1,j]
        k = r + 1  # >basic:[k,r]
        i = l + 1  # =baic:[j+1,i-1]


        while (i < k):
            if (array[i] < basic):
                array[i], array[j + 1] = array[j + 1], array[i]
                j += 1
                i += 1
            elif (array[i] > basic):
                array[i], array[k - 1] = array[k - 1], array[i]
                k -= 1
            else:
                i += 1


        array[l], array[j] = array[j], array[l]
        return j, k




    def quickSort(array, l, r):
        if l > r:
            return;

```

```
32
33          j, k = partition( array, l, r )
34          quickSort( array, l, j - 1 )
35          quickSort( array, k, r )
```