Java程序“快速排序”设计报告

马玉坤-1150310618

2016年7月20日

目录

1 题目描述

快速排序采用的思想是分治思想。

快速排序是找出一个元素（理论上可以随便找一个）作为基准(pivot),然后对数组进行分区操作,使基准左边元素的值都不大于基准值,基准右边的元素值都不小于基准值，如此作为基准的元素调整到排序后的正确位置。递归快速排序，将其他n-1个元素也调整到排序后的正确位置。最后每个元素都是在排序后的正确位置，排序完成。所以快速排序算法的核心算法是分区操作，即如何调整基准的位置以及调整返回基准的最终位置以便分治递归。

使用Java语言编写一个简单的快速排序程序，能够将一个序列按照递增序排序。

2 总体设计思想

使用quickSort函数对数组的一个连续子序列$a[l..r]$进行排序。在quickSort中，找到基准元素$x = a[p]$(p为$[l,r]$中的一个随机数)，然后按照基准元素，将$a[l..r]$分为两个部分，左半部分小于x，右半部分大于等于x。设x的位置为mid，递归调用quickSort，对区间$[l, mid-1]$和$[mid+1, r]$进行排序。

3 详细设计

3.1 quickSort函数

3.1.1 函数原型

```
1    static void quickSort(int l, int r, int[] a);
```

3.1.2 作用

将a数组的$l..r$区间进行排序。

3.1.3 函数体

```
1  static void quickSort(int l, int r, int[] a) {
2    if (l < r) {
3      int mid = partition(l, r, a);
4
5      // 递归排序左半部分
6      quickSort(l, mid - 1, a);
7
8      // 递归排序右半部分
9      quickSort(mid + 1, r, a);
```

```
    }
  }
```

3.2 partition函数

3.2.1 函数原型

```
  static int partition(int l, int r, int[] a);
```

3.2.2 作用

随机获取$a[l..r]$中一个基准元素，然后按照基准元素的大小将$a[l..r]$分为$a[l..mid-1]$和$a[mid+1..r]$两部分，并返回mid。

3.2.3 函数体

```
static int partition(int l, int r, int[] a) {

  // 在区间[l,r]中获得一个随机的位置p
  int p = l + rnd.nextInt(r-l);

  // 交换a[l]和a[p]
  int tmp = a[l];
  a[l] = a[p];
  a[p] = tmp;

  // 将a[l]设为基准（pivot）
  int x = a[l];

  while (l < r) {

    // 从右边找到第一个小于基准的元素，并将其插入到自由位置（l）中
    while (l < r && a[r] >= x)
      r--;
    if (l < r) {
      a[l] = a[r];
      l++;
    }

    // 从左边找到第一个大于等于基准的元素，并将其插入到自由位置（r）中
    while (l < r && a[l] < x)
      l++;
```

```
      if (l < r) {
        a[r] = a[l];
        r--;
      }
    }

    // 将基准元素重新插入到序列中
    a[l] = x;
    return l;
  }
```

4 具体实现

```
import java.util.*;

public class quicksort {

  static void shuffleArray(int[] a)
  {
    /**
     * 该函数的作用是将a数组随机打乱
     */
    Random rnd = new Random();
    for (int i = a.length - 1; i > 0; i--)
    {
      int index = rnd.nextInt(i + 1);
      int tmp = a[index];
      a[index] = a[i];
      a[i] = tmp;
    }
  }

  static Random rnd = new Random();
  static int partition(int l, int r, int[] a) {

    // 在区间[l,r]中获得一个随机的位置p
    int p = l + rnd.nextInt(r-l);

    // 交换a[l]和a[p]
    int tmp = a[l];
    a[l] = a[p];
    a[p] = tmp;
```

```
30
31      // 将a[l]设为基准（pivot）
32      int x = a[l];
33
34      while (l < r) {
35
36        // 从右边找到第一个小于基准的元素，并将其插入到自由位置（l）中
37        while (l < r && a[r] >= x)
38          r--;
39        if (l < r) {
40          a[l] = a[r];
41          l++;
42        }
43
44        // 从左边找到第一个大于等于基准的元素，并将其插入到自由位置（r）中
45        while (l < r && a[l] < x)
46          l++;
47        if (l < r) {
48          a[r] = a[l];
49          r--;
50        }
51      }
52
53      // 将基准元素重新插入到序列中
54      a[l] = x;
55      return l;
56    }
57
58    static void quickSort(int l, int r, int[] a) {
59      if (l < r) {
60        int mid = partition(l, r, a);
61
62        // 递归排序左半部分
63        quickSort(l, mid - 1, a);
64
65        // 递归排序右半部分
66        quickSort(mid + 1, r, a);
67      }
68    }
69
70    public static void main(String Args[]) {
71
72      // 生成一个0..99999的随机排列
73      int[] a= new int[100000];
```

```
74      for (int i = 0; i < a.length; i++)
75        a[i] = i;
76      shuffleArray(a);
77
78      // 进行快速排序
79      quickSort(0, a.length-1, a);
80
81      // 检查排序结果
82      boolean flag = true;
83      for (int i = 0; i < a.length - 1; i++) {
84        if (a[i] != i) {
85          System.out.println("ERROR");
86          flag = false;
87        }
88      }
89      if (flag)
90        System.out.println("FINISHED");
91    }
92  }
```

5 运行结果

```
javac "e:/HITCS/java/exp1/quicksort.java" && java quicksort
FINISHED
```