算法

叶茂林 2024年3月24日星期日

**快速排序**

快速排序递归算法使用双指针，有两种走法，假设枢纽元素取第一个，从小到大排序。

一种是一个指针先走，利用枢纽元素空出来的位置填充找到的比它小的，接着另一个指针走动，利用刚刚填充空出来的位置填充找到的比枢纽元素大的，最后两个指针相同时填充上枢纽元素。

1. **void** QS(**int**\*a,**int** low,**int** high){ // 一指针先走，一指针后走
2. **if**(low>=high)
3. **return**;
4. **int** pivot=a[low],i=low,j=high;
5. **while**(i!=j){
6. **while**(i<j&&a[j]>=pivot) // 右指针找小的
7. j--;
8. **if**(i<j) // 把小的放到枢纽位置
9. a[i++]=a[j];
10. **while**(i<j&&a[i]<=pivot) // 左指针找大的
11. i++;
12. **if**(i<j) // 把大的放到空出来的位置
13. a[j--]=a[i];
14. }
15. a[i]=pivot; // 放置枢纽元素
16. QS(a, low, i - 1);// 继续排左边
17. QS(a, j + 1, high);// 继续排右边
18. }

还有一种是两个指针同时走，左指针找到大的，右指针找到小的，然后交换这两个元素的位置，问题在于枢纽元素和谁交换，让右指针先走，两个指针走到相同位置的时候必然是右指针走到左指针的地方了，而左指针指向的元素是刚刚交换完比枢纽元素小的，而枢纽元素选的是第一个，因此它们进行位置交换就是正确的。

1. **void** QS(**int** \*a, **int** low, **int** high) { // 两个指针同时走
2. **if** (low >= high)
3. **return**;
4. **int** pivot = a[low], i = low, j = high;
5. **while** (i != j) {
6. **while** (i < j && a[j] >= pivot) // 右指针找小的
7. j--;
8. **while** (i < j && a[i] <= pivot) // 左指针找大的
9. i++;
10. **if**(i<j)
11. std::swap(a[i],a[j]);
12. }
13. a[low]=a[i]; // 腾位置给枢纽元素
14. a[i] = pivot; // 放置枢纽元素
15. QS(a, low, i - 1);// 继续排左边
16. QS(a, j + 1, high);// 继续排右边
17. }

**快速选择**

在快速排序中每一轮交换元素过后都可以确定枢纽元素的排序位置，而我们需要寻找第K大的元素，通过记录枢纽元素两边元素的长度可以知道第K大的元素在那一边，因此只需要继续递归一边进行快排，直到枢纽元素的位置是第K个

**优化快速排序**

如果每次都选到最大或者最小的元素作为枢纽元素那每次分区都只能减少一个元素，导致递归树的高度为n，此时时间复杂度退化到O(n²)，空间复杂度退化到O(n)，存储递归所需的栈空间

因此可以优化枢纽元素的选择：三数取中（头尾中各取一个元素选中间值）、随机选、小规模数组使用插入排序