****

算法设计与分析

实验报告(递归与分治)

**学生姓名 杨凯楠**

**学 号** 8208201004

**专业班级 信息安全2002班**

**指导教师** 石峰

**学 院** 计算机学院

**完成时间** 2021年11月18日

1. **实验目的**

理解递归算法的思想和递归程序的执行过程，并能熟练编写递归程序。

掌握分治算法的思想，对给定的问题能设计出分治算法予以解决。

1. **实验内容**

1.找第k小值问题

2.最近点对问题

**三、具体设计**

1、**找k小值问题**

分析：找k小值的程序中，我采取的是类似快排的算法，此算法是一种平均情况下时间复杂度为O(n)的快速选择算法，用到的是快速排序中的第一步，将第一个数作为中枢，使大于它的所有数放到它右边，小于它的所有数放到它左边。之后比较该中枢的最后位置i与k的大小，若i比k小，说明第k小的元素在i的右半段，之后对i的右半段进行快速选择；若i比k大，说明第k小的元素在i的左半段，之后对i的左半段进行快速选择；若i正好等于k，则直接返回。

概要设计：

**1.选择中枢：**如下，为了提高快速选择的效率，最好尽可能的选择数值居中的数作为中枢。

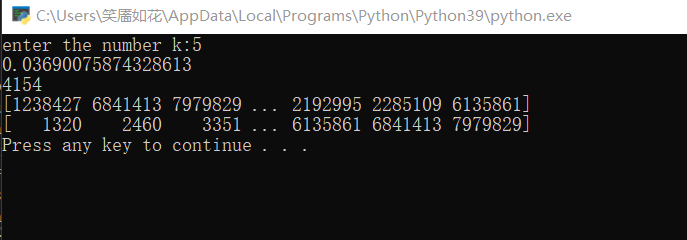
2.对区间[left, right-1]以中枢nums[right-1]**进行一次快速排序**，之后大于nums[right-1]的数全在它的右边，小于nums[right-1]的数全在它的左边。

3.比较i和k的大小：若求序列中的第k大的数，现在由步骤2已经知道一次选择排序后的中枢下标为i，说明前面i个数比pivot大，后面right-i个数比pivot大。如果k < i，说明第k大的数在前面i个数中；如果k > i，说明第k大的数在后面的right-i中；如果i==k，直接返回答案。对于前两种情况只需要对pivot的左半段或者右半段中寻找即可。

详细设计：

1. **import** numpy as np
2. **import** time
3. **import** copy
4. **def** creat(min,max,tot):
5. num=np.random.randint(min,max,tot)
6. **return** num
8. **def** get\_k(num,left,right,k):
9. **if** len(num)<k **or** k<0:
10. **return**
11. temp\_k=get\_index(num,left,right)
12. **if** k == temp\_k+1:
13. **return** num[temp\_k]
14. **if** k<temp\_k+1:
15. **return** get\_k(num,left,temp\_k-1,k)
16. **else**:
17. **return** get\_k(num,temp\_k+1,right,k)
19. **def** swap(num,a,b):
20. temp=num[a]
21. num[a]=num[b]
22. num[b]=temp
24. **def** get\_index(num,left,right):
25. i,j=left,right
26. **while** True:
27. **while** i<right **and** num[i]<num[left]:
28. i+=1
29. **while** left<j **and** num[left]<num[j]:
30. j-=1
31. **if** i>=j:
32. **break**
33. **else**:
34. swap(num,i,j)
35. swap(num,left,j)
36. **return** j
37. **def** main():
38. min,max,tot=1,10000000,10000
39. num=creat(min,max,tot)
40. nums=copy.deepcopy(num)
41. k=input("enter the number k:")
42. starttime=time.time()
43. numk=get\_k(num,0,tot-1,int(k))
44. endtime=time.time()
45. **print**(endtime-starttime)
46. **print**(numk)
47. **print**(nums)
48. **print**(num)
49. main()

**运行结果：**



调试分析：本次测试随机产生了一万个数，经过0.0369s寻找，找到第5小的数为4154。可见算法运行效率还是很高的。

2、**最近点对问题**

分析：

分解

对所有的点按照x坐标（或者y）从小到大排序（排序方法时间复杂度O(nlogn)。

根据下标进行分割，使得点集分为两个集合。

解决

递归的寻找两个集合中的最近点对。

取两个集合最近点对中的最小值min(dis\_left， dis\_right)。

合并

最近距离不一定存在于两个集合中，可能一个点在集合A，一个点在集合B，而这两点间距离小于dis。

难点在于合并。即一个点在集合A，一个在集合B中的情况，可以针对此情况，用之前分解的标准值，即按照x坐标（或者y）从小到大排序后的中间点的x坐标作为mid，划分一个 [mid−dis,mid+dis]区域，如果存在最小距离点对，必定存在这个区域中。

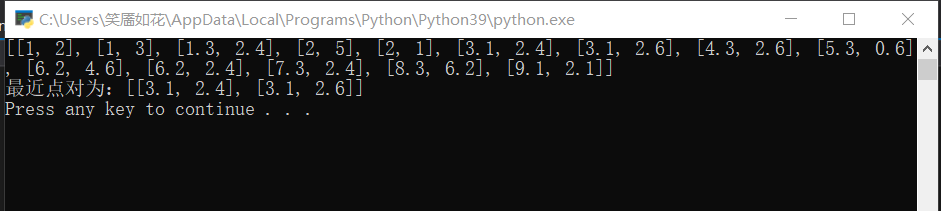
概要设计：一、分解 二、解决 三、合并

详细设计：

1. **from** math **import** sqrt
2. **import** numpy as np
3. **def** nearest\_points(s):
4. len1 = len(s)                                 #len1为点对的数量
5. #print("len=",len1)
6. left = s[0:int(len1/2)]
7. right = s[int(len1/2):]
8. mid\_x = (left[-1][0]+right[0][0])/2               #midx为中间坐标x值
9. **if** len(left) > 2:
10. lmin = nearest\_points(left)    #左侧部分最近点对
11. **else**:   lmin = left
12. **if** len(right) > 2:
13. rmin = nearest\_points(right)   #右侧部分最近点对
14. **else**:   rmin = right
16. **if** len(lmin) >1:
17. disl = get\_distance(lmin)
18. **else**: disl = float("inf")                      #将其设置为正无穷
19. **if** len(rmin) >1:
20. disr = get\_distance(rmin)
21. **else**: disr = float("inf")
23. d = min(disl, disr)   #最近点对距离
25. mid\_min=[]
26. **for** i **in** left:                             #以左部分的点为基准点，找右半部分
27. **if** mid\_x-i[0]<=d :                                          #如果左侧部分与中间线的距离<=d
28. **for** j **in** right:
29. **if** j[0]-i[0]<=d **and** abs(i[1]-j[1])<=d:       #如果右侧部分点在i点的(d,2d)之间
30. **if** get\_distance((i,j))<=d:
31. mid\_min.append([i,j])                      #ij两点的间距若小于d则加入队列
32. **if** mid\_min:
33. dic=[]                                            #定义一个内容为字典的列表
34. **for** i **in** mid\_min:
35. dic.append({get\_distance(i):i})
36. dic.sort(key=**lambda** x: x.keys())                #按字典的键排序
37. **return** list(dic[0].values())[0]
38. **elif** disl>disr:
39. **return** rmin
40. **else**:
41. **return** lmin

44. # 求点对的距离
45. **def** get\_distance(min):
46. **return** sqrt((min[0][0]-min[1][0])\*\*2 + (min[0][1]-min[1][1])\*\*2)
48. **def** main(s):
49. s.sort( key=**lambda** s: s[0])
50. **print**(s)
51. NP = nearest\_points(s)
52. **print**(NP)
53. s=[[1,2],[1,3],[2,5],[1.3,2.4],[6.2,4.6],[9.1,2.1],[2,1],[3.1,2.4],[5.3,0.6],[6.2,2.4],[8.3,6.2],[7.3,2.4],[4.3,2.6],[3.1,2.6]]
54. main(s)

实现：



测试分析：经过计算，最近点对确实为（3.1,2.4）与（3.1,2.6）之间。

1. **实验心得**

完成本次实验时我正好正在学Python，所以萌发了用Python写本实验的念头，然而在写了三个算法实验后，发现Python写的算法程序可读性并不强，并且逻辑也并不是很明确，在此以后将会有所改变。分治法是一种很奇妙的思想，它将原本大的问题分成性质相同的规模更小的问题，在求得更小规模的解后合并，逐步合并成最终解。经过这次实验，我初步体会到，算法不是一门课程，不是一项技术，**而是一种思想**。