算法设计与分析实验

> 计算机85 张子诚

实验一

一.问题描述

加权中位数 设有n个互不相同的元素$x_1, x_2, \cdots, x_n$,每一个元素x_i带有一个权值w_i,且满足$\sum_{i=1}^n w_i = 1$。若元素x_k 满足$\sum_{x_i<x_k} w_i \le \frac{1}{2}$ 且 $\sum_{x_i>x_k} w_i \le \frac{1}{2}$，则称元素$x_k$ 为$x_1, x_2, \cdots, x_n$ 的带权中位数。请编写一个算法，能够在最坏情况下用$O(n)$ 时间找出n 个元素的带权中位数。

二.算法设计与分析

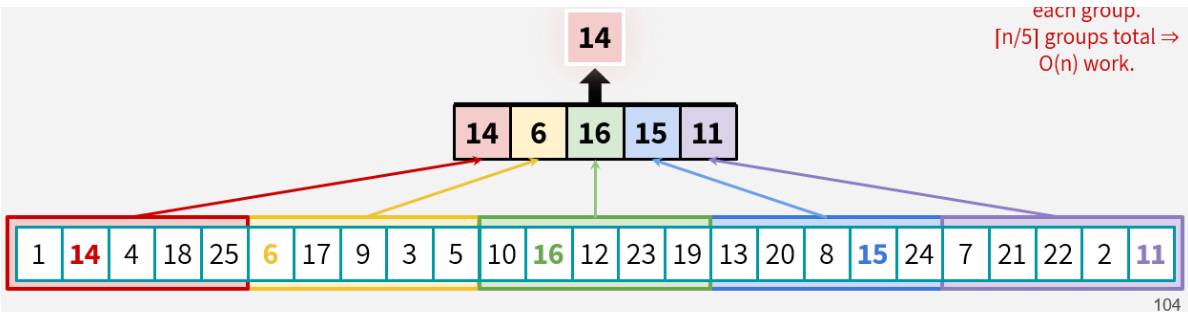
加权中位数的退化版本就是不带权的中位数寻找，如果我们能实现在$O(n)$ 时间复杂度下找到一个数组的中位数，那么求解加权中位数只需要稍作修改即可。

(1)不带权中位数$O(n)$ 算法

对于不带权的中位数一个比较自然的想法是先对原数组进行排序，然后再按中位数的定义进行求解，这种算法的时间复杂度为$O(n\log n)$。
为了寻找$O(n)$复杂度的算法，我们可以借鉴快速排序(quick sort)中的划分选择,以及分治的思想，编写下面寻找一个数组中第k小元素的算法

```
SELECT(A,k):
    if len(A) == 1:
        return A[0]
    p = GET_PIVOT(A)
    L, R = PARTITION(A,p)
    if len(L) == k-1:
        return p
    else if len(L) > k-1:
        return SELECT(L, k)
    else if len(L) < k-1:
        return SELECT(R, k-len(L)-1)
```

该算法若要达到$O(n)$ 的复杂度，关键在于划分的平衡性，下面的寻找中位数的中位数算法保证了划分的平衡性不会超过7n/10。算法原理如下



```python
    def medianOfMedian(self,A):
        median_arr=[]
        for i in range(0,len(A),5):
            k=min(i+5,len(A))
            arr=A[i:k].copy()
            arr=quicksort(arr)
            median_arr.append(arr[len(arr)//2])
        return self.select(median_arr,math.ceil(len(median_arr)//2))
```

这样算法复杂度表达式为

$$T(n)\begin{cases} O(1) & \text{, when } 1 \le n \le 10 \\ T(n/5)+T(7n/10)+O(n) \end{cases}$$

现在来

证明$T(n)=O(n)$

proof

- 归纳假设：$T(n) \le 10n$
- 归纳基础：当$1 \le n \le 10, T(n)=1 \le 10n$
- 归纳
  - 当$k>10$ 假设归纳假设对$1 \le n < k$ 恒成立
  - $$\begin{aligned} T(k) &= k+T(k/5)+T(7k/10) \\ &\le k+10*(k/5)+10*(7k/10) \\ &= k(1+10/5+7*10/10) \\ &\le 10k \end{aligned}$$
  - 因此假设对$n=k$ 也成立
- 从而证明$T(n)=O(n)$

至此我们找到了在O(n)时间复杂度下的寻找中位数算法，将其封装成模块如下

```python
import random
import math
import numpy as np

def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr)//2]
    left = [x for x in arr if x < pivot]
    right = [x for x in arr if x > pivot]
    middle=[x for x in arr if x==pivot]
    return quicksort(left)+middle+quicksort(right)

# O(n) 中位数
class LinearSelection(object):
    def select(self,A,k):
        if len(A)==1:
            return A[0] #只有一个元素时第k小只能是A[0]
        p=self.medianOfMedian(A) #T(n/5)
        L,M,R=self.partition(A,p)# O(n)
```

```
        # T(7n/10)
        if len(L)<k and k<=len(M)+len(L):
            return p
        elif k<=len(L):
            return self.select(L,k)
        elif k>len(M)+len(L):
            return self.select(R,k-len(M)-len(L))

    def partition(self,A,pivot):
        L=[x for x in A if x<pivot]
        R=[x for x in A if x>pivot]
        M=[x for x in A if x==pivot]
        return L,M,R

    def medianOfMedian(self,A):
        median_arr=[]
        for i in range(0,len(A),5):
            k=min(i+5,len(A))
            arr=A[i:k].copy()
            arr=quicksort(arr)
            median_arr.append(arr[len(arr)//2])
        return self.select(median_arr,math.ceil(len(median_arr)//2))
    def findMedian(self,A):
        if len(A)%2==0:
            return (self.select(A,len(A)//2)+self.select(A,len(A)//2+1))/2
        else:
            return self.select(A,math.ceil(len(A)/2)) # ceil(n/2)

if __name__=='__main__':
    arr1=[random.randint(0,1000) for i in range(10)]
    arr2=[random.randint(0,1000) for i in range(9)]
    fun=LinearSelection()
    print("numpy.median:{}".format(np.median(arr1)))
    print("my algorithm:{}".format(fun.findMedian(arr1)))
    print("numpy.median:{}".format(np.median(arr2)))
    print("my algorithm:{}".format(fun.findMedian(arr2)))
```

(2) 加权中位数算法

对线性查找算法稍加修改我们就可以得到加权中位数算法

```
WeightedMedian(x,w):
    if len(x)==1:
        return x1
    elif len(x)==2:
        if w1>w2:
            return x1
        else:
            return x2
    P=median(x)                      # O(n)
    L,R=partition(x,P)               # O(n) Li<P 且 Ri>P
    wl=L中所有元素的权值和           # O(n)
    wr=R中所有元素的权值和           # O(n)
```

```
    if wl<0.5 and wr<0.5:
        return P
    elif wl>wr:
        w_p+=wr
        x'={z| z in x 且 z<=p}
        return WeightedMedian(x',w')            #T(n/2+1) 由按中位数划分保证
    else:
        w_p+=wl
        x'={z|z in x 且 z>=p}
        return WeightedMedian(x',w')            #T(n/2+1) 由按中位数划分保证
```

算法复杂度分析

由上述描述算法我们可以得到

$$T(n) = T(n/2+1) + O(n)$$

根据主定理(*Master Theorem*)

因为 $1 < 2^1$ 所以 $T(n) = O(n)$

三.算法实现以及结果展示

(1) 数据生成

使用numpy.random 中的dirichlet 函数生成随机的概率分布列

```
w=np.random.dirichlet(np.ones(3),size=1)
print(w)
print(np.sum(w))
```

```
[[0.05502781 0.77568647 0.16928572]]
1.0
```

(2) 算法实现

```
from function import *

def sortedMethod(x,table):
    # 排序
    temp=quicksort(x)
    cumprob=0.0
    for item in temp:
        cumprob+=table[item]
        if cumprob>0.5:
            return (item,table[item])

class solution(object):
    def __init__(self):
        self.fun=LinearSelection()

    def partition(self,x,p):
```

```python
        L=[item for item in x if item<p]
        R=[item for item in x if item>p]
        return L,R
    def weightedMedian(self,x,table):
        # 递归的base
        if len(x)==1:
            return x[0]
        if len(x)==2:
            if table[x[0]]==table[x[1]]:
                return (x[0]+x[1])/2
            elif table[x[0]]>table[x[1]]:
                return x[0]
            else:
                return x[1]
        p=self.fun.select(x,math.ceil(len(x)/2))
        # print("中位数{}".format(p))
        # print("长度{}".format(len(x)))
        L,R=self.partition(x,p)

        # print(L)
        # print(R)
        wl,wr=0.0,0.0

        for i in range(len(L)):
            wl+=table[L[i]]
        for i in range(len(R)):
            wr+=table[R[i]]

        if wl<0.5 and wr<0.5:
            return p
        else:
            if wl>wr:
                table[p]+=wr
                x_hat=[item for item in x if item<=p]
                return self.weightedMedian(x_hat,table.copy())
            else:
                table[p]+=wl
                x_hat=[item for item in x if item>=p]
                return self.weightedMedian(x_hat,table.copy())



if __name__=='__main__':
    # 制造随机数据
    x1=[random.randint(0,1000) for i in range(10)]
    w1=np.squeeze(np.random.dirichlet(np.ones(len(x1)),size=1)).tolist()
    table1={}
    lst1=list(zip(x1,w1))
    sol=solution()
    for i in range(len(x1)):
        table1[x1[i]]=w1[i]
    print("随机生成数组")
    print(sorted(lst1,key=lambda x: x[0]))
    print("brute-force:{}".format(sortedMethod(x1,table1)))
    value=sol.weightedMedian(x1,table1)
    print("O(n) select:{}".format((value,table1[value])))
```

(3) 结果展示

```
PS D:\study\algorithm\ex\ex1> & "D:/Program Files/Py/python.exe" d:/study/algorithm/ex/ex1/main_hat.py
随机生成数组
[(145, 0.04269602963034714), (225, 0.02636283798079554), (246, 0.06304745418730079), (267, 0.20669331632371915), (417, 0.03471355136892367), (518, 0.0978017996324321), (553, 0.21506934163593772), (602,
 0.09631880570389582), (665, 0.11986229951131416), (746, 0.097434564025334)]
brute-force:(553, 0.21506934163593772)
O(n) select:(553, 0.21506934163593772)
PS D:\study\algorithm\ex\ex1>
```

- 第一行是用排序算法来实现的加权中位数，用来检验我们算法的正确性
- (a,b) 分别代表x值和w值