Algorithm Third Homework

Zhang Yuan

Student ID: 2015E8013261189

1

1.1 Describe the basic idea in natural language AND pseudo-code

At first, we should check whether the number is even, if not, wrong. Then we should sort the list according to the natural number, choose the largest number d1, check whether the number is larger than the list's size, if so, wrong, then the last d1 numbers all minus 1, if one of these numbers is smaller than 0, wrong.

```
#coding=utf-8

def DetectGraph(d):
    sum=0
    for i in range(len(d)):
        sum+=d[i]

    if sum%2!=0:
        print "Wrong, because sum%2!=0"
        return

    t=d
    n=len(d)

    if 0 in d:
        print "Wrong, because item has 0"
        return
    a=t
    while(n):
        n-=1
        a=sorted(a)
        key=a[-1]
        if key>len(a)-1:
            print "Wrong, because too large item"
            return
        a.pop()
        print "before:",a
        for i in range(key):
            if a[len(a)-1-i]<=0:
                print "Wrong, because too small item"
                return
            a[len(a)-1-i]-=1
        print "after:",a

d=[2,3,2,3]
DetectGraph(d)
```

1.2 Prove the correctness of your algorithm

The three conditions above remark three different situation, whether it can be a graph, whether the graph exists loop or multiple edges between same nodes, whether the graph exists single node without any edge. We use the greedy selection to assure that the node which have largest edges can be connected to other nodes.

1.3 Analyze the complexity of your algorithm

Time complexity: $O(n^2 log(n))$

2

2.1 Describe the basic idea in natural language AND pseudo-code

Sorting the job in decreasing order of their PC times (f_i), and schedule them in this order.

2.2 Prove the correctness of your algorithm

$Assume f_1 is larger than f_2$:

$$p_1 + f_1 < p_1 + p_2 + f_1 \quad IF f_1 > p_2 + f_2$$

$$p_1 + p_2 + f_2 < p_1 + p_2 + f_1 \quad IF f_1 < p_2 + f_2$$

After enumerating all relations between p_1, p_2, f_1, f_2, we can find the greedy selection, if the schedule exits a inversion of order, we can change the order of the inversion items, then we can get a better result,select the job contain longest PC times each time, other times can not be smaller than this schedule.

2.3 Analyze the complexity of your algorithm

Time complexity: O(nlog(n))

3

3.1 Describe the basic idea in natural language AND pseudo-code

We can just sort two lists, and calculate the sum of abs.

```
def findbestminus(a,b):
    c=sorted(a)
    d=sorted(b)

    sum=0
    for i in range(len(a)):
        sum+=abs(c[i]-d[i])

    print sum

a=[1,2,3,7,3]
b=[9,8,6,3,1]
findbestminus(a,b)
```

3.2 Prove the correctness of your algorithm

Consider two boys $b_1 > b_2$, two girls $g_1 > g_2$, after enumerating all possible relations between b_1, b_2, g_1, g_2, we can get the result:

$$|b_1 - a_1| + |b_2 - a_2| \geq |b_2 - a_1| + |b_1 - a_2|$$

We can induce larger cases through above inequality. Now we get the greedy selection property, so we can just sort two lists, and calculate the sum of abs. If there are disorder items in lists, we can exchange the disorder items to get a better result.

3.3 Analyze the complexity of your algorithm

Time complexity: O(nlog(n))

4

4.1 Describe the basic idea in natural language AND pseudo-code

We can just simply sort two list, then the two lists can be sum.

```
def findbestsum(a,b):
    c=sorted(a)
    d=sorted(b)

    sum=0
    for i in range(len(a)):
        sum+=c[i]**d[i]

    print sum

a=[1,3,6,2]
b=[2,4,1,2]
findbestsum(a,b)
```

4.2 Prove the correctness of your algorithm

We assume $a_1 > a_2, b_1 > b_2$, we compared $a_1^{b_1} + a_2^{b_2}$ and $a_1^{b_2} + a_2^{b_1}$.

$$\frac{a_1^{b_1} - a_2^{b_1}}{a_1^{b_2} - a_2^{b_2}} = \frac{a_1^{b_1-b_2}(a_1^{b_2} - a_1^{b_1-b_2}a_2^{b_1})}{a_1^{b_2} - a_2^{b_2}} > 1$$

$$a_1^{b_1} + a_2^{b_2} > a_1^{b_2} + a_2^{b_1}$$

We can induce larger cases through above inequality. So we find the greedy selection property. We can just simply sort two lists, and calculate the sum.If there are disorder items in lists, we can exchange the disorder items to get a better result.

4.3 Analyse the complexity of your algorithm

Time complexity: O(nlog(n))

5

After compression, AesopFables.txt's size is 111KB compared to 193KB before, compression ratio is 0.57, and graph.txt's size is 934KB compared to 2.1MB before, compression ratio is 0.44.

```
#encoding=utf-8
import struct

class Node():
    def __init__(self,id=None,name=None,weight=None,parent=None,left=None,
        self.id=id
        self.name=name
        self.weight=weight
        self.parent=parent
        self.left=left
        self.right=right
        self.code=code

def HuffmanTree(list):
    HuffmanNode=[]
    idmark=0
    for i in list.keys():
        HuffmanNode.append(Node(idmark,i,list.get(i),-1,-1,-1))
        idmark+=1
    n=len(HuffmanNode)

    for i in range(n-1):
        weight1=float("inf")
        weight2=float("inf")
        id1=0
        id2=0
        for j in range(n+i):
            if(HuffmanNode[j].weight<weight1 and HuffmanNode[j].parent==-1
                id2=id1
                weight2=weight1
                id1=j
                weight1=HuffmanNode[j].weight
            elif(HuffmanNode[j].weight<weight2 and HuffmanNode[j].parent==
                id2=j
                weight2=HuffmanNode[j].weight

        HuffmanNode[id1].parent=n+i
        HuffmanNode[id2].parent=n+i
        HuffmanNode.append(Node(n+i,'Tree',weight1+weight2,-1,-1,-1))
        HuffmanNode[n+i].left=HuffmanNode[id1].id
        HuffmanNode[n+i].right=HuffmanNode[id2].id

    return HuffmanNode

def PrintHuffmanTree(HuffmanList):
    n=len(HuffmanList)
    name=[]
    code=[]
    dict={}

    for i in range(n):
        if HuffmanList[n-1-i].parent==-1:
            HuffmanList[n-1-i].code=""
        else:
            index=HuffmanList[n-1-i].parent
            idMark=HuffmanList[n-1-i].id
            if idMark==HuffmanList[index].left:
                HuffmanList[n-1-i].code=HuffmanList[index].code+"0"
            elif idMark==HuffmanList[index].right:
                HuffmanList[n-1-i].code=HuffmanList[index].code+"1"
            if HuffmanList[n-1-i].name!="Tree":
                dict.setdefault(HuffmanList[n-1-i].name,HuffmanList[n-1-i]

    print dict

    return dict

def compress(filename):
    file=open(filename,'rw+')
    line=file.read()
    dict={}
    for i in range(len(line)):
        if line[i] not in dict:
            dict.setdefault(line[i],1)
        else:
            dict[line[i]]+=1


    mid=HuffmanTree(dict)
    dict_1=PrintHuffmanTree(mid)

    output=open("/Users/zhangyuan/Desktop/learn/demo.txt",’rb+’)
    code_data=""
    for i in range(len(line)):
        code_data+=dict_1[line[i]]

#   print code_data

    for j in range(0,len(code_data),8):
        if j+8<len(code_data):
            bytes=struct.pack(’B’,int(code_data[j:j+8],2))
            output.write(bytes)
        else:
            bytes=struct.pack(’B’,int(code_data[j:],2))
            output.write(bytes)
    output.close()

    dict_2={}
    for i in dict_1.keys():
        dict_2.setdefault(dict_1[i],i)
    print dict_2
    return dict_2

def decompress(infilename,outfilename,dict):
    file=open(infilename,’rb+’)
    line=file.read(1)
    k=""
    while not line==’’:
        bdata=bin(struct.unpack(’B’,line)[0])[2:]
        if len(bdata)!=8:
            for i in range(8-len(bdata)):
                k+=’0’
        k+=bdata
        line=file.read(1)

    code_data=k[:-8]
    temp=""
    flag=True
    for i in range(8):
        if k[-8+i]==’0’ and flag:
            pass
        else:
            temp+=k[-8+i]
            flag=False
    code_data+=temp
#print code_data

    File=open(outfilename,’w+’)
    mark=""
    for i in range(len(code_data)):
        mark+=code_data[i]
        if mark in dict:
            File.write(dict[mark])
            mark=""
        else:
            pass

    File.close()

dict=compress("/Users/zhangyuan/Desktop/learn/graph.txt")
decompress("/Users/zhangyuan/Desktop/learn/demo.txt","/Users/zhangyuan/Desk
```