# P140

## 7、

1. 对于一个包含100万随机数的数组排序，快速排序比插入排序快多少倍？
2. 是非题：对于n>1的n元素数组，是否存在插入排序比快速排序跟快的情形

解：（1）插入排序的平均速率是Cavg=O(n2)，而快速排序的平均速率是Cavg=nlog2n,得插入排序约/快速排序等于50000，所以快速排序比插入排序快了50000倍

1. 存在当快速排序完成编译时间是属于最坏所需时间O(n2)，所需的额外空间是O(nlog2n)，差入排序的时间复杂度属于O(n2)，所需的额外空间为0(1)此时，当n>1的n元素数组存在插入排序比快速排序更快

# P225

## 6、

为下列问题设计一个动态规划算法。已知小木棍的销售价格pi和长度i相关，i=1,2...,n，如何把长度为n的木棍切割若干根长度为整数的小木棍，使得所能获得的总销售价格最大？该算法的时间效率和空间效率各是多少？

长度为n的最大价值 price(n)=MAX(price(i)+price(n-i))长度为n的价格有两种 第一种：原始长度为n时的价格 第二种：加n分割为个小块 再加起来的价格

设长度1~n长度的木棍价格为p[1…n]

先从最短的长度 1开始找相对应长度可得到的最大价值，因为长度1无法再分，所以maxprice[1] 就为原始长度价格 p[1]

然后长度2的可得到的最大价值maxprice[2]就为maxprice[1] +maxprice[1] 和 p[2]之中最大的那个3的可得到的最大价值maxprice[3]就为 maxprice[1]+maxprice[2] 、maxprice[2]+maxprice[1]和p[3]中最大的那个（为什么没有maxprice[1]+maxprice[1]+maxprice[1]？因为1+1=2，长度为2的最大价格maxprice[2]已知，所以不需要再把2分为1+1了，而1+2=3、2+1=3时的长度3的maxprice[3]的价格才是不知道的。也就是每个长度只需要分为两部分就行了）

因为比当前长度小的所有整数长度的对应的最大价格都是已知的，所以长度为n时只需要找到maxprice[1]+maxprice[n-1]、maxprice[2]+maxprice[n-2]、…、maxprice[i]+maxprice[n-i]、…、maxprice[n-1]+maxprice[1]、p[n]中最大的值，再赋值给maxprice[n]

此算法的时间效率是O(n^2)，空间效率是O (N)

# P229

## 3、

对于背包问题的自底向下动态规划算法，请证明：

它的时间效率属于Ɵ(nW)

它的空间效率属于Ɵ(nW)

从一张填好的动态规划表中求得最优子集的组合所用的时间属于O(n)

对于背包问题的自底向上动态规划算法，请证明:

a.它的时间效率属于O(nW)。

b.它的空间效率属于0(nW)。

c.从一张填好的动态规划表中求得最优子集的组合所用的时间属于O(n)。

解答：

背包问题:有n个物品，第i哥物品的体积为Vi，重量为Wi，对应价值为P1、、P2、P3、...Pn。

在前N件物品中，选取若干件物品放入背包中;

状态是:在前N件物品中，选取若干件物品放入所剩空间为W的背包中的所能获得的最大价值;

决策是:第N件物品放或者不放;

由此可以写出动态转移方程:

f[i,j]表示在前i件物品中选择若干件放在所剩空间为j的背包里所能获得的最大价值

F[i,j]=max{f[i-1,j]-Wi]+Pi (j>=Wi), f[i-1,j]}

这样，我们可以自底向\_上地得出在前M件物品中取出若干件放进背包能获得的最大价值，也就是f[n,w]

算法如下：

Begin

For i=0 to w do

F[0,j]=0;

For i=1 to n do

For j=0 to w do begin

F[i,j]=f[i-1,j]

If (j>=w) and if (j>=w) and (f[i-1,j-w]+Pi>f[i,j]) then

f[i,j]:=f[i-1,j-w]+Pi;

end

　ruturn(f[m,w]);

　　end;

得证：由于是用了一个二重循环，这个算法的时间复杂度是O（n\*w)

B、

背包问题，特点是每种物品仅有一件，可以选择放或不放。

　　f[v]表示前i件物品恰放入一个容量为v的背包可以获得的最大价值。则其[状态转移方程](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=37255466&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)便是：

　　f[v]=max{f[v],f[v-c]+w}

　　这个方程基本上所有跟背包相关的问题的方程都是由它衍生出来的。所以有必要将它详细解释一下：“将前i件物品放入容量为v的背包中”这个子问题，若只考虑第i件物品的策略（放或不放），那么就可以转化为一个只牵扯前i-1件物品的问题。如果不放第i件物品，那么问题就转化为“前i-1件物品放入容量为v的背包中”，价值为f[v]；如果放第i件物品，那么问题就转化为“前i-1件物品放入剩下的容量为v-c的背包中”，此时能获得的最大价值就是f[v-c]再加上通过放入第i件物品获得的价值w。

以上方法的[时间和空间](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=9264164&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)复杂度均为O(N\*w)

# P234

## 11、

矩阵连乘

考虑如何使得在计算n个矩阵的乘积A1 A2...An 时，总的乘法次数最小，这些矩阵的维度分别为d0\*d1,d1\*d2,..，dn-1\*dn,。假设所有两个矩阵的中间乘积都使用蛮力算法(基于定义)计算。

给出一个三个矩阵连乘的例子，当分别用(A1A2)A3 和A1(A2A3)计算时，它们的乘法次数至少相差1 000倍。

有多少种不同的方法来计算n个矩阵的连乘乘积?

c.设计一个求n个矩阵乘法最优次数的动态规划算法。

A:

矩阵乘法不满足分配律，但满足结合律，因此(A\* B)\*C既可以按顺序(A\*B)\*C进行，也可以按A\*(B\*C)来进行。

可得例子：A1：5\*1000、A2：1000\*5、A3：5\*1000

B:

算法数递推公式为，并且m(1)=1

C:

给出n个矩阵组成的序列，设计一种方法把它们依次相乘，使得总运算量最小。假设第i个矩阵Ai;是pi-1\*p;的

设f(i,j)表示把 Ai, Ai+1,...,A 乘起来所需要的乘法次数，枚举“最后一次乘法”是第k个乘号，则f(i,j)=max{f(i,k)+f(k+1,j)+pi-1pkpj},边界是f(i,i)=0,时间复杂度为O(n3)。

# P249

## 7、

有n个人，每个人都拥有不同的谣言。通过发电子信息，他们想互相

共享所有的谣言。假定发送者会在信息中包含他已知的所有谣言，而且一条信息只有一个收信人。设计一个贪心算法，保证在每个人都能获得所有谣言的条件下，使发送的信息数最小。

解答:

按照1发信给2, 2发信给3, 3发信给4，…，

n-1发信给n的方式发送谣言，该贪心算法基于每次发信都使得当前收信人掌握的谣言更多，最后由n将所有谣言发送给其他n-1个人。

发送信息总数为2n-2，这是最小的发信息数。因为每增加一个人，至少需要增加两次发送信息，当n=2是，发送信息数为2，归纳法可证明2n-2为最小发信息数

# P264

## 9、

### a、

写一个程序，为给定的英文文本构造一套哈夫曼编码，生成一套哈夫曼编码。

#include <stdio.h>

#define MAXLEN 100

typedef struct           /\*定义结构体\*/

{

    int weight;         /\*定义一个整型权值变量\*/

    int lchild, rchild,parent;      /\*定义左、右孩子及双亲指针\*/

} HTNode;

typedef HTNode HT[MAXLEN];          /\*向量类型\*/

int n;

void InitHFMT (HT T)

{       /\*哈夫曼树初始化子函数\*/

    int i;

    printf ("\n 请输入共有多少个权值(小于100): ");

    scanf ("%d",&n);

    for (i=0; i<2\*n-1; i++)

    {

        T[i].weight=0;

        T[i].lchild=-1;

        T[i].rchild=-1;

    }

}

void InputWeight (HT T)

{ /\*输入权值子函数\*/

    int w,i;

    for (i=0; i<n;i++)

    {

        printf ("输入第%d 个权值:",i+1);

        scanf ("%d",&w) ;

        getchar() ;

        T[i].weight=w;

    }

}

void SelectMin(HT T,int i,int \*p1,int \*p2)

{   /\*选择两个结点中小的结点\*/

    long min1=88888, min2=888888;   /\* 预设两个值，并使它大于可能出现的最大权值\*/

    int j;

    for (j=0;j<=i;j++)

    {

        if(T[j].parent==-1)

        {

            if(min1>T[j].weight)

                {

                    min1=T[j].weight;           /\*找出最小的权值\*/

                    \*p1=j;                      /\*通过\*p1带回序号\*/

                }

        }

    }

    for(j=0;j<=i;j++)

    {

        if(T[j].parent==-1)

        {

            if(min2>T[j].weight&&j!=(\*p1))

            {

                min2=T[j].weight;/\* 找出次最小的权值\*/

                \*p2=j;

            }

        }

    }

}

void CreatHFMT (HT T)

{                       /\*构造哈夫曼树，T[2\*n-1]为其根结点\*/

    int i,p1,p2;

    InitHFMT(T);

    InputWeight(T);

    for(i=n;i<2\*n-1;i++)

    {

        SelectMin(T,i-1,&p1,&p2);

        T[p1].parent=T[p2].parent=i;

        T[i].lchild=T[p1].weight;

        T[i].rchild=T[p2].weight;

        T[i].weight=T[p1].weight+T[p2].weight;

    }

}

void PrintHFMT (HT T)

{                       /\* 输出向量状态表\*/

    int i;

    printf("\n哈夫曼树的各边显示: \n");

    for (i=0; i<2\*n-1; i++)

    while(T[i].lchild!=-1)

    {

        printf("(%d, %d)，(%d,%d) \n",T[i].weight, T[i].lchild,T[i].weight,T[i].rchild);

        break;

    }

}

void hfnode(HT T,int i,int j)

{                           /\*哈夫曼编码函数\*/

    j=T[i].parent;

    if(T[j].rchild=T[i].weight)

    printf ("1") ;

    else

    printf("0");

    if(T[j].parent!= -1)

    i=j, hfnode(T,i,j);

}

void huffmannode (HT T)

{       /\*求哈夫曼编码\*/

    int i,j,a;

    printf("\n输入的权值的对应哈夫曼编码:");

    for (i=0;i<n;i++)

    {

    j=0;

    a=i;

    printf("\n名1的编码为:",T[i].weight);

    hfnode(T,1,j);

    i=a;

    }

}

void main()     /\*主函数\*/

{

    HT HT;

    CreatHFMT (HT);

    PrintHFMT (HT);

    huffmannode(HT);

    printf("\n ");

}

### b、

写一个程序，对一段用哈夫曼编码的英文文本进行解码

#include<stdio.h>

#define n 5 //叶子数目

#define m (2\*n-1) //结点总数

#define maxval 10000.0

#define maxsize 100 //哈夫曼编码的最大位数

typedef struct

{

char ch;

float weight;

int lchild,rchild,parent;

}hufmtree;

typedef struct

{

char bits[n]; //位串

int start; //编码在位串中的起始位置

char ch; //字符

}codetype;

void huffman(hufmtree tree[]);//建立哈夫曼树

void huffmancode(codetype code[],hufmtree tree[]);//根据哈夫曼树求出哈夫曼编码

void decode(hufmtree tree[]);//依次读入电文，根据哈夫曼树译码

void main()

{

printf(" ——哈夫曼编码——\n");

printf("总共有%d个字符\n",n);

hufmtree tree[m];

codetype code[n];

int i,j; //循环变量

huffman(tree); //建立哈夫曼树

huffmancode(code,tree); //根据哈夫曼树求出哈夫曼编码

printf("【输出每个字符的哈夫曼编码】\n");

for(i=0;i<n;i++)

{

printf("%c: ",code[i].ch);

for(j=code[i].start;j<n;j++)

printf("%c ",code[i].bits[j]);

printf("\n");

}

printf("【读入电文，并进行译码】\n");

decode(tree); //依次读入电文，根据哈夫曼树译码

}

void huffman(hufmtree tree[])//建立哈夫曼树

{

int i,j,p1,p2; //p1,p2分别记住每次合并时权值最小和次小的两个根结点的下标

float small1,small2,f;

char c;

for(i=0;i<m;i++) //初始化

{

tree[i].parent=0;

tree[i].lchild=-1;

tree[i].rchild=-1;

tree[i].weight=0.0;

}

printf("依次读入前%d个结点的字符及权值(中间用空格隔开)\n",n);

for(i=0;i<n;i++) //读入前n个结点的字符及权值

{

printf("输入第%d个字符为和权值",i+1);

scanf("%c %f",&c,&f);

getchar();

tree[i].ch=c;

tree[i].weight=f;

}

for(i=n;i<m;i++) //进行n-1次合并，产生n-1个新结点

{

p1=0;p2=0;

small1=maxval;

small2=maxval; //maxval是float类型的最大值

for(j=0;j<i;j++) //选出两个权值最小的根结点

if(tree[j].parent==0)

if(tree[j].weight<small1)

{

small2=small1; //改变最小权、次小权及对应的位置

small1=tree[j].weight;

p2=p1;

p1=j;

}

else

if(tree[j].weight<small2)

{

small2=tree[j].weight; //改变次小权及位置

p2=j;

}

tree[p1].parent=i;

tree[p2].parent=i;

tree[i].lchild=p1; //最小权根结点是新结点的左孩子

tree[i].rchild=p2; //次小权根结点是新结点的右孩子

tree[i].weight=tree[p1].weight+tree[p2].weight;

}

}//huffman

void huffmancode(codetype code[],hufmtree tree[]) //根据哈夫曼树求出哈夫曼编码

//codetype code[]为求出的哈夫曼编码

//hufmtree tree[]为已知的哈夫曼树

{

int i,c,p;

codetype cd; //缓冲变量

for(i=0;i<n;i++)

{

cd.start=n;

cd.ch=tree[i].ch;

c=i; //从叶结点出发向上回溯

p=tree[i].parent; //tree[p]是tree[i]的双亲

while(p!=0)

{

cd.start--;

if(tree[p].lchild==c)

cd.bits[cd.start]='0'; //tree[i]是左子树，生成代码'0'

else

cd.bits[cd.start]='1'; //tree[i]是右子树，生成代码'1'

c=p;

p=tree[p].parent;

}

code[i]=cd; //第i+1个字符的编码存入code[i]

}

}//huffmancode

void decode(hufmtree tree[]) //依次读入电文，根据哈夫曼树译码

{

int i,j=0;

char b[maxsize];

char endflag='2'; //电文结束标志取2

i=m-1; //从根结点开始往下搜索

printf("输入发送的编码(以'2'为结束标志)：");

gets(b);

printf("译码后的字符为");

while(b[j]!='2')

{

if(b[j]=='0')

i=tree[i].lchild; //走向左孩子

else

i=tree[i].rchild; //走向右孩子

if(tree[i].lchild==-1) //tree[i]是叶结点

{

printf("%c",tree[i].ch);

i=m-1; //回到根结点

}

j++;

}

printf("\n");

if(tree[i].lchild!=-1&&b[j]!='2') //电文读完，但尚未到叶子结点

printf("\nERROR\n"); //输入电文有错

}//decode

### C、

做一个实验，测试对包含1000个单词的一段英文文本进行哈夫曼编码时，典型的压缩率位于什么样的区间。

### D、

对编码程序做一个实验，测试如果用标准的估计频率代替英文文本中字符的实际出现频率，该程序的压缩率会有怎样的变化。

# P331

## 6、

用回溯法生成{1，2，3，4}的所有排列

代码：

import java.util.\*;

public class pailie{

    static int n;

    static int[] a;//一种排列情况

    static boolean[] b;//判断数字i是否已被使用

    static void dfs(int step){//step表示当前到了第几位

        if(step==n+1){//前n位已经放好，输出这种排列情况

            for(int i=1;i<=n;i++)

                System.out.print(a[i]);

            System.out.println();

            return;

        }

        for(int i=1;i<=n;i++){

            if(!b[i]){//若数字i还未被使用

                a[step]=i;//在当前位上放i

                b[i]=true;//防止重用

                dfs(step+1);//放下一位数字

                b[i]=false;//回溯法，进行另一种尝试

            }

        }

    }

    public static void main(String[] args){

        Scanner in=new Scanner(System.in);

        n=in.nextInt();

        a=new int[n+1];

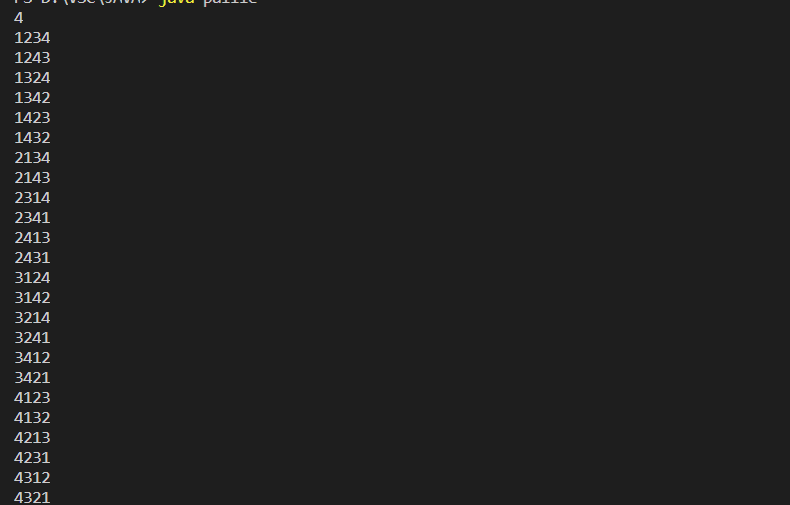
        b=new boolean[n+1];

        dfs(1);//从第一位开始放

    }

}

结果：



# P338

## 7、

写一个程序用分支界限法对背包问题求解

代码：

import java.util.Collections;

import java.util.LinkedList;

/\*\*

 \* 0-1背包问题

 \* @author Lican

 \*

 \*/

/\*\*

 \* 子集空间树中结点类型为BBnode

 \* @author Lican

 \*

 \*/

class BBnode{

    BBnode parent;//父节点

    boolean leftChild;//左儿子结点标志

    public BBnode(BBnode par,boolean left){

        parent=par;

        leftChild=left;

    }

}

/\*\*

 \* 优先级队列的结点

 \* @author Lican

 \*

 \*/

class HeapNode implements Comparable{

    BBnode liveNode;//活结点

    double upperProfit;//结点的价值上界

    double profit;//结点所相应的价值

    double weight;//结点所相应的重量

    int level;//活结点在子集树种所处的层序号

    //构造方法

    HeapNode(BBnode node,double up,double pp,double ww,int lev){

        liveNode=node;

        upperProfit=up;

        profit=pp;

        weight=ww;

        level=lev;

    }

    @Override

    public int compareTo(Object x) {//降序排列

        double xs=((HeapNode) x).upperProfit;

        if(upperProfit>xs) return -1;

        if(upperProfit==xs) return 0;

        return 1;

    }

}

/\*\*

 \* 每个背包的类

 \* @author Lican

 \*

 \*/

class Element implements Comparable{

    int id;//背包编号

    double d;//单位重量价值

    public Element(int id,double d){

        this.id=id;

        this.d=d;

    }

    @Override

    public int compareTo(Object o) {//降序排列

        double xs=((Element) o).d;

        if(d>xs) return -1;

        if(d==xs) return 0;

        return 1;

    }

    public boolean equals(Object o){

        return d==((Element) o).d;

    }

}

public class BBKnapsack {

    public double c;//背包容量

    public int n;//物品总数

    public double[] w;//物品重量数组

    public double[] p;//物品价值数组

    public double cw;//当前重量

    public double cp;//当前价值

    public int[] bestx;//最优解

    public LinkedList<HeapNode> heap;//活结点优先队列

    //上界函数bound计算结点所相应价值的上界

    public double bound(int i){

        double cleft=c-cw;//剩余容量

        double b=cp;

        //以物品单位重量价值递减顺序装填剩余容量

        while(i<=n&&w[i]<=cleft){

            cleft-=w[i];

            b+=p[i];

            i++;

        }

        if(i<=n)

            b+=p[i]\*cleft/w[i];

        return b;

    }

    public void addLiveNode(double up,double pp,double ww,int lev,BBnode par,boolean ch){

        BBnode b=new BBnode(par,ch);

        HeapNode node=new HeapNode(b,up,pp,ww,lev);

        heap.add(node);

        Collections.sort(heap);

    }

    /\*\*

     \* 优先队列式分支限界法，返回最大价值，bestx返回最优解

     \* @return

     \*/

    public double bbKnapsack(){

        //初始化

        BBnode enode=null;

        int i=1;

        double bestp=0.0;//当前最优解

        double up=bound(1);//价值上界

        //搜索子集空间树

        while(i!=n+1){

            //非叶节点

            //检查当前扩展结点的左儿子结点

            double wt=cw+w[i];

            if(wt<=c){//左儿子结点可行

                if(cp+p[i]>bestp){

                    bestp=cp+p[i];

                }

                addLiveNode(up,cp+p[i],cw+w[i],i+1,enode,true);

            }

            up=bound(i+1);

            //检查当前扩展结点的有儿子结点

            if(up>=bestp){//右子树可能含有最优解

                addLiveNode(up,cp,cw,i+1,enode,false);

            }

            HeapNode node=heap.poll();

            enode=node.liveNode;

            cw=node.weight;

            cp=node.profit;

            up=node.upperProfit;

            i=node.level;

        }

        //构造当前最优解

        for(int j=n;j>0;j--){

            bestx[j]=(enode.leftChild)?1:0;

            enode=enode.parent;

        }

        return cp;

    }

    /\*\*

     \* 返回最大价值

     \* @param pp

     \* @param ww

     \* @param cc

     \* @param xx

     \* @return

     \*/

    public double knapsack(double[] pp,double[] ww,double cc,int[] xx){

        c=cc;

        n=pp.length-1;

        double ps=0;//统计所有背包的价值总量

        double ws=0;//统计所有的背包重量之和

        Element[] q=new Element[n];

        for(int i=1;i<=n;i++){

            q[i-1]=new Element(i,pp[i]/ww[i]);

            ps+=pp[i];

            ws+=ww[i];

        }

        if(ws<=c){//所有物品之和<=最大容量C,即可全部物品装包

            for(int i=1;i<=n;i++){

                xx[i]=1;

            }

            return ps;

        }

        //依物品单位重量价值排序

        java.util.Arrays.sort(q);

        //初始化数据成员

        p=new double[n+1];

        w=new double[n+1];

        for(int i=1;i<=n;i++){

            p[i]=pp[q[i-1].id];

            w[i]=ww[q[i-1].id];

        }

        cw=0;

        cp=0;

        bestx=new int[n+1];

        heap=new LinkedList<HeapNode>();

        //调用bbKnapsack求问题的最优解

        double maxp=bbKnapsack();

        for(int i=1;i<=n;i++){

            xx[q[i-1].id]=bestx[i];

        }

        return maxp;

    }

    public static void main(String[] args) {

        double pp[]={0,2,1,4,3};

        double ww[]={0,1,4,2,3};

        double cc=8;

        int n=pp.length-1;

        int[] xx=new int[n+1];

        BBKnapsack b=new BBKnapsack();

        double maxp=b.knapsack(pp, ww, cc, xx);

        System.out.println("装入背包中物品总价值最大为："+maxp);

        System.out.println("装入的物品的序号为：");

        for(int i=1;i<=n;i++){

            System.out.println(i+":"+xx[i]);

        }

    }

}

结果：

