**P140 第7题**

**a.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？**

答：插入比快速排序时间快10倍。

**b.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更快的情形？**

答：是的，前提是已经排好序的n>1的n元素数组，插入排序在对几乎已经排好序的数据操作时，效率高，即可以达到线性排序的效率，但插入排序一般来说是低效的，因为插入排序每次只能将数据移动一位。

**P225 第6题**

**切割木棍问题：为下列问题设计一个动态规划算法。 已如小木根的销售价格pi和长度i相关，i=1,2, ....，n，如何把长度为n的木根切制为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?**

答：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 第6题**

**对于背包问题写一段自底向上动态规划算法，请证明：**

//背包问题\_自底向上法\_动态规划：

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Text;

using System.Threading.Tasks;

namespace

{

class Program

{

static void Main(string[] args)

{

int m;

int[] w = { 0, 3, 4, 5 };

int[] p = { 0, 4, 5, 6 };

Console.WriteLine(BottomUp(10,3,w,p));

Console.WriteLine(BottomUp(3,3,w,p));

Console.WriteLine(BottomUp(4, 3, w, p));

Console.WriteLine(BottomUp(5, 3, w, p));

Console.WriteLine(BottomUp(7, 3, w, p));

Console.ReadKey();

}

public static int[,] result=new int[11,4];

public static int BottomUp(int m, int i, int[] w, int[] p)

{

if (result[m, i] != 0) return result[m, i];

for (int tempM = 1; tempM < m+1; tempM++)

{

for (int tempI = 1; tempI < i+1; tempI++)

{

if(result[tempM,tempI]!=0)continue;

if (w[tempI] > tempM)

{

result[tempM, tempI] = result[tempM, tempI - 1];

}

else

{

int maxValue1 = result[tempM - w[tempI], tempI - 1] + p[tempI];

int maxValue2 = result[tempM, tempI - 1];

if (maxValue1 > maxValue2)

{

result[tempM, tempI] = maxValue1;

}

else

{

result[tempM, tempI] = maxValue2;

}

}

}

}

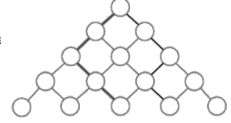
return result[m, i];

}

}

}

**a.它的时间效率属于**Θ**(nW)**

自底向上实现，是规模增加，它的计算过程是一阶段一阶段地向前推进，直到最终状态。

塔底向塔顶计算，增长函数为T（k），第K层加法总数为第k-1层的节点数\*2：

T（k）=2\*（k-1+…+3+2+1）=k\*（k-1），k>0 平方型

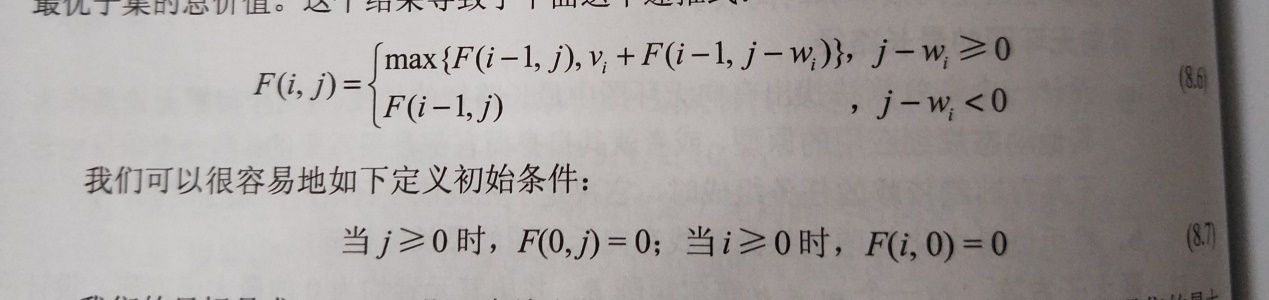
根据该数塔得到k=5，加法总数2\*（4+3+2+1）=20次

**b.它的空间效率属于**Θ**(nW)**

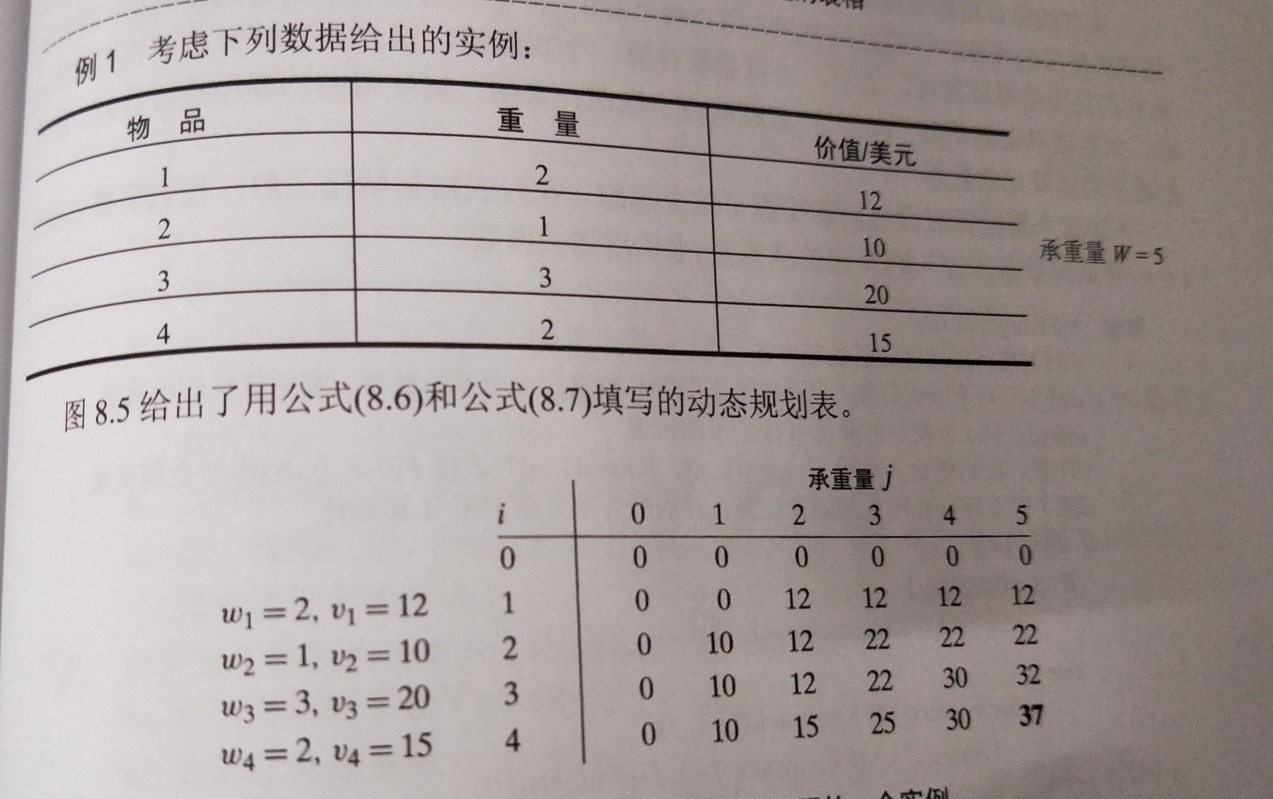
如果在动态规划算法中，计算一个值无法在常数时间内完成，性能的改进可能会更显著。最好记住这一点，相对于自底向上算法的空间优化版本来说，记忆功能算法的空间效率是较低的。

**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)**

根据课本8.6、8.7这两个公式计算出：



例题1的最优子集：



可以得到物品1是最优解{物品1，物品2，物品4}的最后一个部分，该算法要计算19个值，需要访问表格中8个单元格才能计算出最优子集。

**P234 第11题**

**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。**

**a.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。**

三个矩阵连乘，可以有(A1A2)A3和A1(A2A3)两种方法求积 ，乘法次数分别为： p0p1p2+p0p2p3和p0p1p3+p1p2p3

假设p0=10, p1=100, p2=5, p3=50, 两种方法的次数分别是：7500 和 75000

#include<iostream>

using namespace std;

//矩阵连乘问题的解

int MatrixChainOrder(int n,int p[],int a, int b){

int m[n+1][n+1], s[n+1][n+1];//m记录乘法操作次数，s记录分割点k

for(int i = 1;i <= n;i++){

m[i][i] = 0;

s[i][i] = 0;

}

for(int i = n-1;i >= 1;i--){

for(int j = i+1;j <= n;j++){

m[i][j] = 10000000;

for(int k = i;k <= j-1;k++){

int sum = m[i][k] + m[k+1][j] + p[i-1]\*p[k]\*p[j];

if(sum < m[i][j]){

m[i][j] = sum;

s[i][j] = k;

}

}

}

}

printf("%d %d", m[a][b], s[a][b]);

}

int main(){

int n, i, j;

cin >> n;

int p[n+1];

for(int i = 0;i < n+1;i++)cin >> p[i];//第i个矩阵为pi\*p(i+1)

cin >> i >> j;

MatrixChainOrder(n, p, i, j);

}

**b.有多少种不同的方法来计算n个矩阵的连乘乘积?**

矩阵A和B可乘的条件是矩阵A的列数等于矩阵B的行数。若A是一个p×q的矩阵，B是一个q×r的矩阵，则其乘积C=AB是一个p×r的矩阵。计算C=AB总共需要p×q×r次乘法。给定n个矩阵{A1,A2,…,An}。其中Ai与Ai+1是可乘的，i=1,2,…,n-1。

要求计算出这n个矩阵的连乘积A1A2…An最少需要多少次乘法。

#include <iostream>

#include <cstdio>

#include <cstring>

#define MAXSIZE 105

using namespace std;

int A[MAXSIZE];

int mul[MAXSIZE][MAXSIZE];

int loc[MAXSIZE][MAXSIZE];

int main()

{

int n,t,i,r,k,j;

cin >> n;

memset(mul,0,sizeof(mul));

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

{

cin >> A[i];

cin >> t;

}

A[n] = t;

for(r= 1;r<n;r++)

{

for(i =1;i<n;i++)

{

j = i + r;

mul[i][j] = mul[i+1][j]+A[i-1]\*A[i]\*A[j];

loc[i][j] = i;

for(k = i + 1; k < j; k++)

{

t = mul[i][k]+mul[k + 1][j] + A[i - 1]\*A[k]\*A[j];

if(t < mul[i][j])

{

mul[i][j] = t;

loc[i][j] = k;

}

}

}

}

cout << mul[1][n] << endl;

return 0;

}

**c.设计一个求n个矩阵乘法最优次数的动态规划算法。**

#include<stdio.h>

#include<iostream>

#include<algorithm>

#include<stdlib.h>

using namespace std;

const int MAX = 100;

int n;

int p[MAX+1],m[MAX][MAX],s[MAX][MAX];

//p用来记录矩阵，m[i][j]表示第i个矩阵到第j个矩阵的最优解，s[][]记录从哪里断开可以得到最优解

void matrixChain()

{

for(int i=1; i<=n; i++)//初始化数组

m[i][i]=0;

for(int r=2; r<=n; r++)//对角线循环

{

for(int i=1; i<=n-r+1; i++) //行循环

{

int j=i+r-1;//列的控制

m[i][j]=m[i+1][j]+p[i-1]\*p[i]\*p[j];//找m[i][j]的最小值，初始化使k=i;

s[i][j]=i;

for(int k=i+1; k<j; k++)

{

int t=m[i][k]+m[k+1][j]+p[i-1]\*p[k]\*p[j];

if(t<m[i][j])

{

s[i][j]=k;//在k位置断开得到最优解

m[i][j]=t;

}

}

}

}

}

void traceback(int i,int j)

{

if(i==j)

return;

traceback(i,s[i][j]);

traceback(s[i][j]+1,j);

cout<<"Multiply A"<<i<<","<<s[i][j]<<"and A"<<s[i][j]+1<<","<<j<<endl;

}

int main()

{

cin>>n;

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

cin>>p[i];

matrixChain();

traceback(1,n);

cout<<m[1][n]<<endl;

return 0;

}

**P249 第7题**

**7.谣言传播 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。**

答：将这n个人标记为1, 2, …, n，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 第9题**

**9.**

**a. 写一个程序，为给定的英文文本构造一套哈夫曼编码，并对该文本编码。**

**b.写一个程序，对一段用哈夫曼码编码的英文文本进行解码。**

#include<stdio.h>

#include<malloc.h>

#include<string.h>

typedef struct HTNode

{

char charname;

int weight;

int parent, lchild, rchild;

}HTNode, \*HuffmanTree;

typedef char \*\*HuffmanCode; //动态分配数组存储字符编码的动态空间

int n=27,m=53; // m=2\*n-1 m为根节点个数

char \*a;

int \*b;

void HuffmanCoding(HuffmanTree HT,HuffmanCode HC,char \*a,int \*b,int n)

{

int i;//通用整型变量i，用于循环中

int s1 = 0; //变量s1用于哈夫曼树的创建

int s2 = 0;//变量s2用于哈夫曼树的创建

int j;

int d;

int m=2 \* n - 1;

char \*cd=(char\*)malloc(n\*sizeof(char)); //分配临时存放每个字符编码的动态空间，译码使用

//构造哈夫曼树HT

if(n<=1)

{

return ;

}//n值小于等于1，不满足创建哈弗曼树的条件

//HT=(HuffmanTree)malloc(((2\*n-1)+1)\*sizeof(HTNode));

for(i=1;i<=n;++i)

{

HT[i].charname=a[i-1];

HT[i].weight=b[i-1];

HT[i].parent=0;

HT[i].lchild=-1;

HT[i].rchild=-1;

}

for(;i<=m;++i) //所要产生结点初始化所要产生结点

{

HT[i].charname='0';

HT[i].weight=0;

HT[i].parent=0;

HT[i].lchild=-1;

HT[i].rchild=-1;

}

//以下功能为寻找权值字最小的两个值

for(i=n+1;i<=m;++i) //用于筛选出两个权重最小的两个

{

for(s1=1;HT[s1].parent!=0;)

{

s1++;

} //双亲不为0，让变量s1加1，寻找双亲为0的结点

for (j=s1;j<=i-1;j++) //用于寻找第一个双亲为0的结点

{

if(HT[j].parent!=0)

{

continue;

}

s1=HT[j].weight<HT[s1].weight?j:s1; //寻找第一个权重最小的下标

}

HT[s1].parent=i;

HT[i].lchild=s1;

for(s2=1;HT[s2].parent!=0;) //同s1作用一样，找第二个双亲为0的结点

{

s2++;

}

for (j=s2;j<=i-1;j++)

{

if(HT[j].parent != 0)

{

continue;

}

s2 = HT[j].weight<HT[s2].weight?j:s2;

}

HT[s2].parent = i;

HT[i].rchild = s2;

HT[i].weight = HT[s1].weight+HT[s2].weight; //将两个最小的结点权重相加组成父结点

}

//求每个字符的哈夫曼编码，存储在编码HC中

// HC=(HuffmanCode)malloc((n+1)\*sizeof(char\*));

cd[n-1]='\0'; //存储编码值，下标n-1的为'\0'结束

for (i=1; i<=n;++i) //逐个字符求哈夫曼编码

{

int start=n-1,c,f; //start开始时指向最后,既编码结束位置

for (c=i,f=HT[i].parent; f!=0;c=f,f=HT[f].parent) //f指向结点c的双亲结点，f！=0从叶子结点向上回溯，直到根结点

{

if(HT[f].lchild==c) //结点c是f的左孩子，则生成代码0

{

cd[--start]='0';

}

else if(HT[f].rchild==c) //结点c是f的右孩子，则生成代码1

{

cd[--start]='1';

}

}

HC[i]=(char\*)malloc((n-start) \* sizeof(char));//为每个字符分配空间

f=n-start; //继续回溯

for(d=0;d<f;d++,start++) //将cd数组中的字符译码值传送给HC的动态数组中去

{

HC[i][d]=cd[start];

}

}

free(cd); //释放临时空间

}

//信息存储函数

void cunchu\_charnanme\_Num()

{

int i;

char charName[27]={' ','A','B','C','D','E','F','G','H','I','J','K','L','M','N', //字符存储数组

'O','P','Q','R','S','T','U','V','W','X','Y','Z'} ;

int Weight[27]={'186','64','13','22','32','103','21','15','47','57','1','5','32','20','57', //权重存储数组

'63','15','1','48','51','80','23','8','18','1','16','1'};

a=(char \*)malloc(n\*sizeof(char)); //给全局变量a分配空间

b=(int \*)malloc(n\*sizeof(int)); //给全局变量b分配空间

for (i=0;i<n;i++) //将数组charName内容给予数组a

{

a[i]=charName[i];

}

for(i=0;i<n;i++) //将数组Weight内容给予数组b

{

b[i] = Weight[i];

}

}

//显示哈夫曼树

void showHT\_AND\_showHC(HuffmanTree HT,HuffmanCode HC,int n, int m,char\* a)

{

int i=0;

int j=0;

int k=0; //打印循环k

printf("\n");

printf(" >>>>所存储的哈夫曼编码以及译码<<<<\n");

printf(" |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

for (i = 1; i <= n; i++) //已二维数组的形式输出哈夫曼树

{

printf(" |\*\*%4c :",a[i-1]); //输出字符

for(k=0;HC[i][k]!='\0'; k++)

{

printf("%c",HC[i][k]); //输出字符对应的译码值

}

printf("%13c",'\*');

printf("\n");

}

printf(" |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

printf("\n");

}

//查找所需要的字母的编码值

void Find\_letter(HuffmanTree HT,HuffmanCode HC,int n, int m,char\* a)

{

int i;

int j;

int k;

int num=0;

char find1[100]; //存字符码数组

printf(">>>>>>>>>>>>>请输入所要查找的字符的编码值('#'结束输入):");

for(i=0;;i++) //可循环输入字符，'#'结束输入

{

num++;

scanf("%c",&find1[i]);

if(find1[i]=='#')

{

break;

}

}

printf("\n");

printf(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>字符的哈夫曼编码是:");

for (i=0;i<num;i++) //打印所找到对应字符的编码值

{

for(k=1;k<n;k++)

{

if(find1[i]==HT[k].charname) //如果输入字符与存储数组字符相等，打印其编码

{

for(j=0;j<=11;j++)

{

printf("%c",HC[k][j]);

}

}

}

//printf("\n");

}

printf("\n");

printf("\n");

}

void Find\_num(HuffmanTree HT,HuffmanCode HC,int n, int m,char\* a)

{

int i=0;

int j;

int k=0;

int c=0;

char find2[100]; //译码存放数组

printf(">>>>>>>>>>>>请输入所需要查询译码对应字母的值('#'结束输入):");

//scanf("%s",&find2);

gets(find2);

printf("\n");

printf(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>所要查询的字符为:");

while(find2[i]!='#') //如果输入编码值结束不为'#'则进入while循环来找到对应的字符

{

for(j=1;j<28;j++)

{

while(find2[i+k]==HC[j][k]) //如果找到输入译码值与存储译码值相等，进入

{

//printf("0");

k++;

if(HC[j][k]=='\0') //如译码数组结束'\0'

{

k--;

printf("%c",HT[j].charname); //打印对应的字符

break;

}

}

if(find2[i+k]!=HC[j][k]) //不相等跳出循环，判断下一个二维数组值是否相等，直到找到为止

{

k=0;

continue;

}

else

{

i=i+k+1; //相等时让i=i+k+1,判断第二个字符编码值，如果没有则跳出

k=0;

continue;

}

}

}

printf("\n");

}

void main()

{

int t;

char ss;

HuffmanTree HT;

HuffmanCode HC;

HT=(HuffmanTree)malloc((m+1)\*sizeof(HTNode)); //动态分配m+1个单元

HC=(HuffmanCode)malloc(n\*sizeof(HuffmanCode)); //动态分配n个单元

cunchu\_charnanme\_Num(); //存储函数

HuffmanCoding(HT, HC, a, b, n); //存储译码函数

while(t!=4) //输入不同的值进行不同操作

{ printf(" 请输入相应字符开始执行程序\n");

printf("\n");

printf(" >>>>>1. 输出存储的哈夫曼编码和译码值 <<<<<\n");

printf(" >>>>>2.输入字符获取其对应的二进制译码<<<<<\n");

printf(" >>>>>3. 输入二进制获取其对应的字符 <<<<<\n");

printf(" >>>>>4. 退出 <<<<<\n");

printf(" >>>>>>>>>>>>>>>输入<<<<<<<<<<<<<<<\n");

printf(" 指令： ");

scanf("%d",&t);

switch(t)

{

case 1:

showHT\_AND\_showHC(HT, HC, n, m, a); //输入1，打印存储哈夫曼

break;

case 2:

Find\_letter(HT, HC, n, m, a); //输入2，寻找字符对应编码值

break;

case 3:

scanf("%c",&ss);

Find\_num(HT, HC, n, m, a); //输入3，寻找编码对应字符

break;

case 4:

printf(" >>>>>>>>>>>>>>>退出<<<<<<<<<<<<<<<\n"); //输入4，退出系统

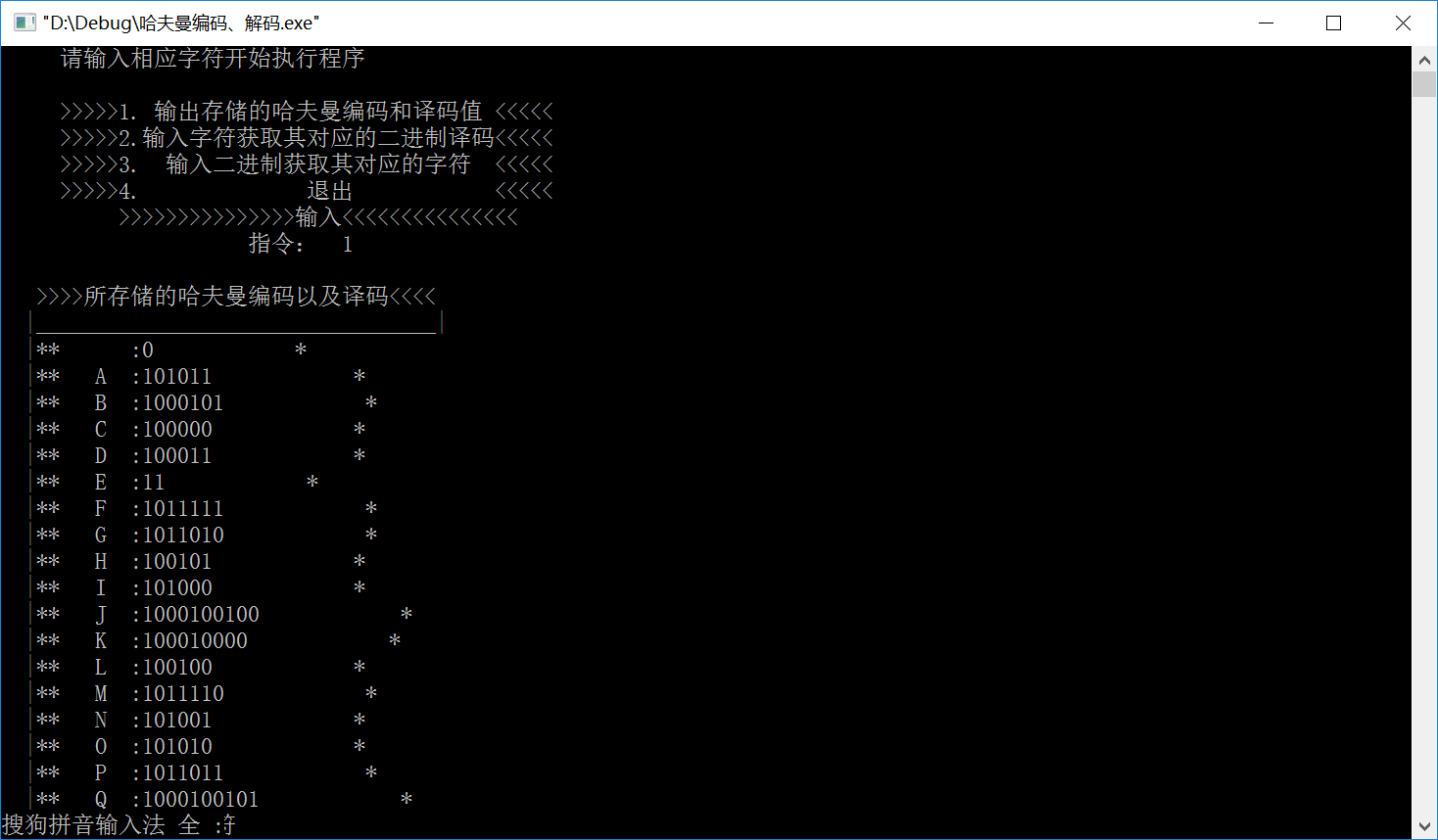
break;

}

}

}

**运行结果：**



**c.做一个实验，测试对包含1000个词的一 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。**

答：大概是198k可以压缩成121k左右的样子，当然压缩效率并不是绝对的，相对于不同内容的文本压缩效率会有所不同。重复性较高的文本压缩效率会更高。

**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什么样的变化。**

答：有可能会出现乱码。

**P331 第7题**

**7.用回溯法生成{1，2，3，4}的所有排列。**

#include <iostream>

using namespace std;

#define n 4

int s[n] = { 1,2,3,4};

int r[4];

int v[n];

void print(){

for(int i = 0; i<4;i++)

cout<<s[r[i]]<<" ";//超过最大值，回溯

cout<<endl;

}

void combination(){

int k = 0 ;r[0] = -1;

while(k>=0){

r[k]++;

while(r[k]<n&&v[r[k]]!=0)

r[k]++;

if(r[k]<n){

v[r[k]] = 1;

if(k==2) //已到最后一位，找到一种排列，则输出

print();

else{

k++;

r[k] = -1;

}

}else{//继续向前搜索

k--;

for(int i = r[k]-1; i< n; i++)

v[i] = 0;

}

}

}

int main(){

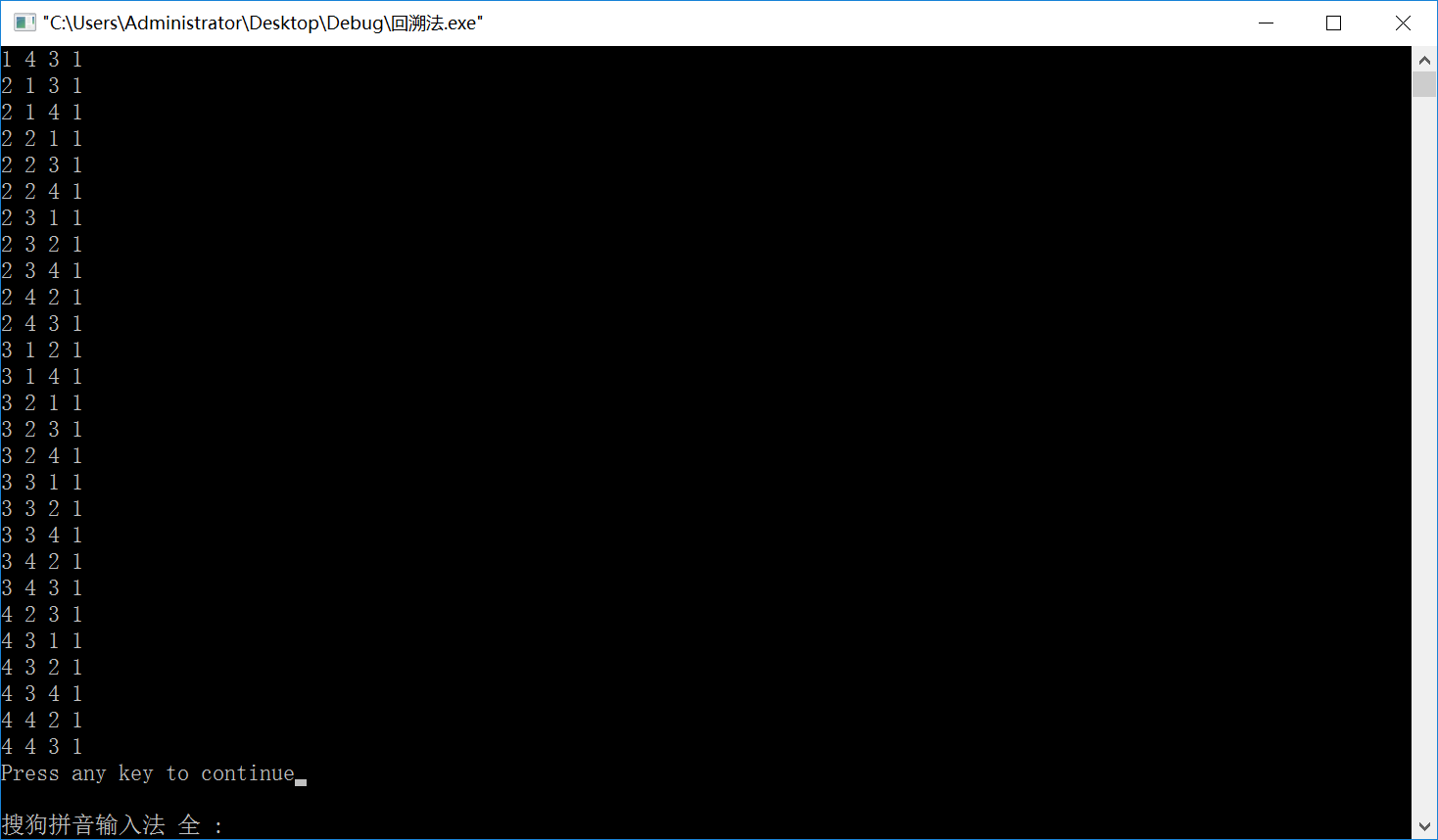
memset(v,0,sizeof(v));

combination();

return 0 ;

}

运行结果，得到序列1,2,3,4的全排列:



**P338 第7题**

**7.写一个程序用分支界限算法对背包问题求解。**

#include<stdio.h>

#include<iostream>

#include<queue>

using namespace std;

//bool select[10];//定义一个bool型数组用于判定是否选中对应物品；

int bagw=50,bagv,bestvalue,cw,cv;//背包可承受重量，背包可承受价值，最佳价值，当前重量，当前价值；

float ub;//上界;

//queue<thing> bestq;//定义一个结构体队列，用于存放最终结果；

struct avgvalue{//结构体存放物体的编号与单位重量的价值；

int num;

float avg;

int value;//对应物品价值；

int weight;//对应物品重量；

}re[10];

struct node{//定义结点；

int cw;

int cv;

float ub;

}point[2];//point[0]表示左结点，[1]表示右结点；

void compare(avgvalue re[10])//排序物品单位重量的价值；

{

int i,j,tmp;

float t;

for(i=0;i<=9;i++)

{

for(j=i+1;j<=9;j++)

{

  if(re[i].avg<re[j].avg)

{

 t=re[i].avg;

 re[i].avg=re[j].avg;

 re[j].avg=t;

tmp=re[i].value;

               re[i].value=re[j].value;

               re[j].value=tmp;

               tmp=re[i].weight;

               re[i].weight=re[j].weight;

               re[j].weight=tmp;

            }

            else

               continue;

        }

    }

}

int der(avgvalue re1[10])//界限函数；

{

    int i,k,t;//j用来存放当前最大单位回报;

    float j;

    cw=0,cv=0;//初始化当前重量与价值；

    j=re[0].avg;

   ub=j\*(bagw-cw)+cv;//上界公式,找到根节点；

    for(i=0;i<=8;i++)

    {

       j=re[i+1].avg;

       point[1].cv=cv;

       point[1].cw=cw;

      point[1].ub=j\*(bagw-point[1].cw)+point[1].cv;//不包含第i个物品时当前最佳回报；

       point[0].cw=re1[i].weight;

       if(cw+point[0].cw<=bagw)//包含第i个物品不超重的情况下；

        {

           point[0].cv=re1[i].value;

           point[0].ub=j\*(bagw-cw-point[0].cw)+point[0].cv+cv;//包含第i个物品时当前最佳回报；

           if(point[0].ub>point[1].ub)

            {

               ub=point[0].ub;

               cw+=point[0].cw;

               cv+=point[0].cv;

            }

            else

               ub=point[1].ub;

        }

        else//若包含超重，直接选择不包含的情况；

           ub=point[1].ub;

    }

    return cv;

}

int main()

{

    int i;

   for(i=0;i<=9;i++)

    {

       cin>>re[i].weight;//输入10个物品的重量；

    }

   for(i=0;i<=9;i++)

    {

       cin>>re[i].value;//输入10个物品的价值；

    }

    for(i=0;i<=9;i++)

    {

        re[i].num=i;

        re[i].avg=(float)re[i].value/re[i].weight;

    }

    compare(re);

   bestvalue=der(re);

   cout<<bestvalue<<endl;

    return 0;

}

规定物品数量为10，背包容量为50，输入为20个数，前十个为物品重量，后十个数为物品价值，得到最大价值为29。

