第140页，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？

在忽略常数、误差的平均情况中，快速排序执行约10^12次，插入排序执行约10^7次，快速排序比插入排序快多大约十万倍。

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？

第225页 第6题

6.切割木棍问题：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；长度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。

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)。

第229页 第3题

3.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Θ（nW）。

b.它的空间效率属于Θ（nW）。

c.从一张填好的动态规划表中求得最优子集得组合所用的时间属于Ο（n）。

第234页 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，这些矩阵的纬度分别为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1000倍。

设这三个矩阵的维数分别为10×100，100×5和5×50。

若按 ((A1A2)A3)来计算，需要10×100×5+10×5×50=7500次的数乘。

若按 (A1(A2A3))来计算，需要100×5×50+10×100×50=75000次的数乘。

所以，它们的乘法次数至少相差1000倍。

b.有多少种不同的方法来计算n个矩阵的连乘乘积？

3种：1、穷举法；2、重叠递归法；3、备忘录递归法

c.设计一个求n个矩阵乘法最优次数的动态规划算法。

代码：

#include<stdio.h>

int r[50],com[50][50];

long m[50][50];

long int course(int i,int j){

long int u,t;

int k;

if(m[i][j]>=0)

return m[i][j];

if(i==j)

return 0;

if(i==j-1){

com[i][i+1]=i;

m[i][j]=r[i]\*r[i+1]\*r[i+2];

return m[i][j];

}

u=course(i,i)+course(i+1,j)+(long)r(i)\*r(i+1)\*r(j+1);

com[i][j]=i;

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

t=course(i,k)+course(k+1,j)+(long)r(i)\*r(k+1)\*r(j+1);

if(t<u){

u=t;

com[i][j]=k;

}

}

m[i][j]=u;

return u;

}

int main(){

int n,i,j;

printf("请输入矩阵个数:\n");

scanf("%d",&n);

printf("请输入矩阵型号:\n");

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

scanf("%d",&r[i]);

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

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

m[i][j]=-1;

printf("最少的乘法次数为:\n",course(1,n));

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

printf("\n");

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

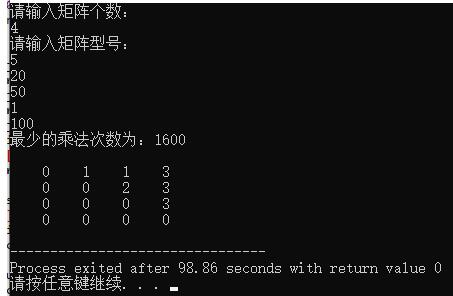
printf("%5d",com[i][j]);

}

printf("\n");

}

结果：



第249页 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。

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。

第264 第9题

1. 写一个程序，为给定的英文文本构造套哈夫曼编码， 并对该文本编码。

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define N 100 //定义叶子节点最多100个

#define M 2\*N-1 //所有结点的个数

typedef char \* HuffmanCode[2\*M]; //huffman编码

typedef struct

{

int weight; //权重

int parent; //父节点

int LChild; //左节点

int RChild; //右节点

}HTNode,Huffman[M+1]; //huffman树

typedef struct Node

{

int weight; //叶子结点的权重

char c; //叶子节点

int num; //叶子结点的二进制码的长度

}WNode,WeightNode[N];

/\*产生叶子结点的字符和权重\*/

void CreateWeight(char ch[],int \*s,WeightNode CW,int \*p)

{

int i,j,k;

int tag;

\*p=0; //叶子节点个数

//统计字符出现个数,并放入CW

for(i=0;ch[i]!='\0';i++){

tag=1;

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

if(ch[j]==ch[i]) //通过判断让相同元素存在同一个数组单元中

{

tag=0;

break;

}

if(tag)

{

CW[++\*p].c=ch[i];

CW[\*p].weight=1;

for(k=i+1;ch[k]!='\0';k++)

if(ch[i]==ch[k])

CW[\*p].weight++; //权值累加

}

}

\*s=i; //字符串长度

}

void CreateHuffmanTree(Huffman ht,WeightNode w,int n)

{

int i,j;

int s1,s2;

//初始化哈夫曼树

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

{

ht[i].weight=w[i].weight;

ht[i].parent=0;

ht[i].LChild=0;

ht[i].RChild=0;

}

for(i=n+1;i<=2\*n-1;i++)

{

ht[i].weight=0;

ht[i].parent=0;

ht[i].LChild=0;

ht[i].RChild=0;

}

for(i=n+1;i<=2\*n-1;i++)

{

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

if(!ht[j].parent)

break;

s1=j; //找到第一个双亲不为0的节点

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

if(!ht[j].parent)

s1=ht[s1].weight>ht[j].weight?j:s1;

ht[s1].parent=i;

ht[i].LChild=s1;

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

if(!ht[j].parent)

break;

s2=j; //找到第二个双亲不为0的节点

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

if(!ht[j].parent)

s2=ht[s2].weight>ht[j].weight?j:s2;

ht[s2].parent=i;

ht[i].RChild=s2;

ht[i].weight=ht[s1].weight+ht[s2].weight; //权值累加

}

}

void CrtHuffmanNodeCode(Huffman ht,char ch[],HuffmanCode h,WeightNode weight,int m,int n)

{

int i,c,p,start;

char \*cd;

cd=(char \*)malloc(n\*sizeof(char));

cd[n-1]='\0'; //末尾置0

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

{

start=n-1;

c=i;

p=ht[i].parent; //p在n+1至2n-1；

while(p)

{

start--; //依次向前置值

if(ht[p].LChild==c)

cd[start]='0';

else

cd[start]='1';

c=p;

p=ht[p].parent;

}

weight[i].num=n-start; //二进制码的长度（包含末尾0）

h[i]=(char \*)malloc((n-start)\*sizeof(char));

strcpy(h[i],&cd[start]);//将二进制字符串拷贝到指针数组h中

}

free(cd); //释放cd内存

system("pause");

}

void CrtHuffmanCode(char ch[],HuffmanCode h,HuffmanCode hc,WeightNode weight,int m,int n)

{

int i,k;

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

{

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

if(ch[i]==weight[k].c)

break;

hc[i]=(char \*)malloc((weight[k].num)\*sizeof(char));

strcpy(hc[i],h[k]);

}

}

void TrsHuffmanTree(Huffman ht,WeightNode W,HuffmanCode hc,int m,int n)

{

int i=0,j,p;

printf("\*\*\*解码出的字符串\*\*\*\n");

while(i<m)

{

p=2\*n-1;

for(j=0;hc[i][j]!='\0';j++)

{

if(hc[i][j]=='0')

p=ht[p].LChild;

else

p=ht[p].RChild;

}

printf("%c",W[p].c);

i++;

}

}

void FreeHuffmanCode(HuffmanCode h,HuffmanCode hc,int n,int m)

{

int i;

for(i=1;i<=n;i++) //释放叶子结点的编码

free(h[i]);

for(i=0;i<m;i++) //释放所有结点的编码

free(hc[i]);

}

void main(){

int i,n=0;

int m=0;

char ch[N];

Huffman ht;

HuffmanCode h,hc;

WeightNode weight;

printf("请输入要编码的字符串:");

gets(ch);

CreateWeight(ch,&m,weight,&n);

printf("\*\*\*叶子节点与权重信息\*\*\*\n Node");

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

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

printf("\n weight");

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

printf("%d",weight[i].weight);

system("pause");

CreateHuffmanTree(ht,weight,n);

printf("\n\*\*\*显示哈夫曼树\*\*\*\n");

printf("\ti\tweight\tparent\tLChild\tRChild\n");

for(i=1;i<=2\*n-1;i++)

printf("\t%d\t%d\t%d\t%d\t%d\n",i,ht[i].weight,ht[i].parent,ht[i].LChild,ht[i].RChild);

CrtHuffmanNodeCode(ht,ch,h,weight,m,n);

printf("\*\*\*显示叶子节点编码\*\*\*\n");

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

{

printf("\t%c:",weight[i].c);

printf("%s\n",h[i]);

}

system("pause");

CrtHuffmanCode(ch,h,hc,weight,n,m);

printf("\*\*\*显示字符串编码\*\*\*\n");

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

printf("%c",hc[i]);

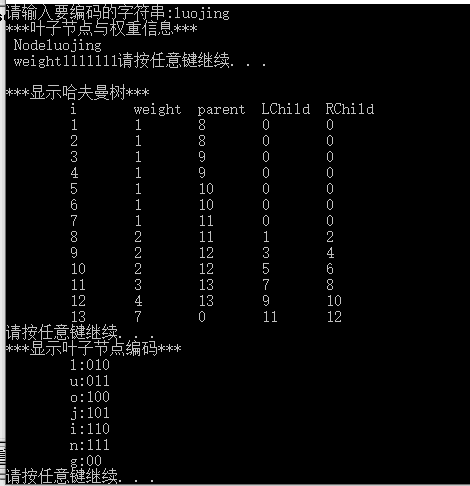
system("pause");

TrsHuffmanTree(ht,weight,hc,n,m);

FreeHuffmanCode(h,hc,n,m);

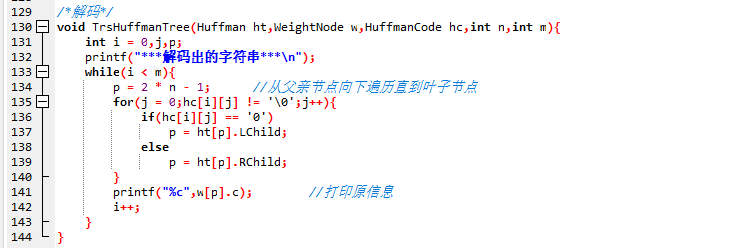
system("pause");

}



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。

代码：



结果：

C:\Users\Lenovo\Documents\Tencent Files\2710187542\Image\SharePic\20200628165137.png

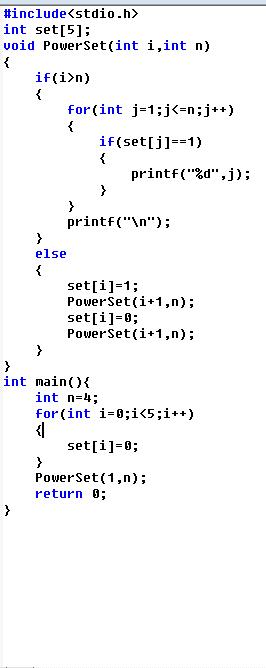
c.做一个实验，测试对包含1000个词的一段英文文本进行哈夫曼编码时，典型的压缩率位于什么样的区间。

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。

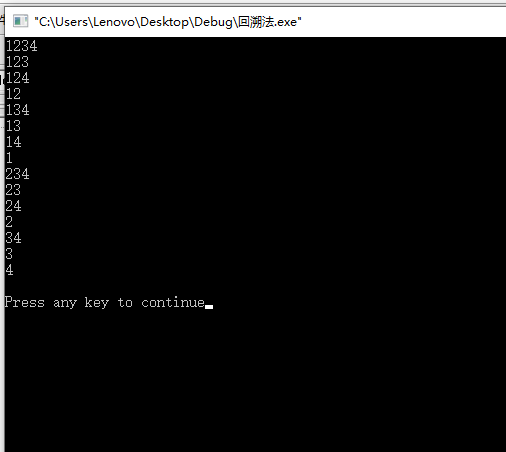
第331页 第7题

7.用回溯法生成{1,2,3，4}的所有排列。

代码：



结果：



第338页第7题

7.写一个程序用分支界限算法对背包问题求解。

代码：

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

//物品

typedef struct goods {

int item; //物品名称

int weight; //重量

int value; //价值

int ave; //平均价值=价值/重量

};

//运算符重载，优先队列降序排列

typedef struct temp {

bool operator()(goods a, goods b) {

return a.ave < b.ave;

}

};

priority\_queue < goods, vector<goods>, temp> q;

void Knapsack(int W) {

vector<bool> choose(q.size() + 1, false); //物品是否选择放入背包

int ub = q.top().ave\*W; //初始化上界

int w = 0, v = 0; //当前背包里的重量w，价值v

while (!q.empty()){

if (w + q.top().weight > W) {

q.pop();

if (!q.empty())

ub = v + q.top().ave\*(W - w);

continue;

}

goods a = q.top(); //取出队首位置的物品

q.pop();

if (!q.empty()) {

int yes = v + a.value + q.top().ave\*(W - w - a.weight); //放入物品

int no = v + q.top().ave\*(W - w); //不放物品

if (yes > no) {

ub = yes;

w += a.weight;

v += a.value;

choose[a.item] = true;

}

else

ub = no;

}

else {

v += a.value;

choose[a.item] = true;

}

}

cout << "背包最大价值为：" << v << endl;

cout << "装入的物品编号为：";

for (int i = 1; i < choose.size(); i++)

if (choose[i])

cout << i << " ";

cout << endl;

}

int main() {

int n, W;

cout << "输入物品个数：";

cin >> n;

cout << "输入背包容量：";

cin >> W;

vector<goods> package(n);

cout << "输入物品信息：（重量w 价值w）" << endl;

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

package[i].item = i + 1;

cin >> package[i].weight >> package[i].value;

package[i].ave = package[i].value / package[i].weight;

q.push(package[i]); //存入信息入队

}

Knapsack(W); //01背包

system("pause");

return 0;

}

结果：

