**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？

快速排序基本思想:通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以[递归](https://baike.so.com/doc/5725016-5937752.html" \t "https://baike.so.com/doc/_blank)进行，以此达到整个数据变成有序序列。

插入排序基本思想：有一个已经有序的数据序列，要求在这个已经排好的数据序列中插入一个数，但要求插入后此数据序列仍然有序，这个时候就要用到一种新的排序方法--[插入排序法](https://baike.so.com/doc/801157-847507.html" \t "https://baike.so.com/doc/_blank),插入排序的基本操作就是将一个数据插入到已经排好序的有序数据中，从而得到一个新的、个数加一的有序数据，算法适用于少量数据的排序，[时间复杂度](https://baike.so.com/doc/107803-113805.html" \t "https://baike.so.com/doc/_blank)为O(n^2)。

在忽略常数、误差的平均情况下，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**

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？

对于长度为n的木棍，他的递推关系：price(n)=max(price(i)+price(n-i))

长度为n的价格有两种：

第1种情况：原始长度为n时的价格

第2种情况：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]

首先可以从最短的长度 1开始找相对应长度可得到的最大价值，因为当长度1无法再分，因此maxprice[1] 就为原始长度价格 p[1]；

长度2的可得到的最大价值maxprice[2]就是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]就是 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，因此长度为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]。此算法的时间复杂度为O(n^2)，空间复杂度为O (N)。

**第229页 第3题**

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**

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。

例如3个矩阵A1，A2，A3，阶分别为10×100，100×5和5×50。

计算连乘积时如果按 ((A1A2)A3)来计算，需要10×100×5+10×5×50=7500次的数乘。

如果按 (A1(A2A3))来计算，则需要100×5×50+10×100×50=75000次的数乘。在计算连乘积时，加括号的方式对计算量有很大影响。因此，它们的乘法次数至少相差75000/7500=1000倍。

b.有多少种不同的方法来计算n个矩阵的连乘乘积？

有3种不同的方法来计算n个矩阵的连乘乘积分别为：

穷举法、重叠递归法、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>

#define N 30

#define M 30

#define O 30

#define MAX 5000//为保证兼容性，相乘次数不应超过整数的表示范围

void main()

{

int a[N],mm[N][M],result[N][M];

int i,j,k,t,l,temp,rr;

int n;

printf("请输入数组数:\n");

scanf("%d",&rr);

printf("请输入正确的数据:\n");//输入的时候重复的数据只输入一次

for(i=0;i<rr+1;i++)

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

n=rr;

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

mm[i][i]=0;//数组本身和自己相乘，最少次数为0次

for(l=2;l<=n;l++)

{

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

{

j=i+l-1;

mm[i][j]=MAX;//开始时赋值为最大值，无穷大的意思

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

{

temp=mm[i][k]+mm[k+1][j]+a[i-1]\*a[k]\*a[j];//

if(temp<mm[i][j])

{

mm[i][j]=temp;//取得最小的次数

result[i][j]=k;//保存每次截断的位置

}

}

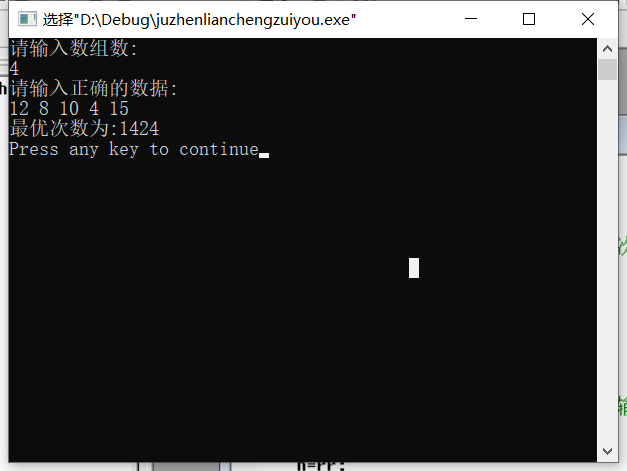
}

}

printf("最优次数为:%d\n",mm[1][n]);

}

**运行截图：**



**第249页 第7题**

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。

可以将这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，这个贪心算法是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>

#include<conio.h>

#define N 100 //定义N的值为100

typedef struct

{

int weight;

int parent, lchild, rchild;//定义父母、左右子树

}hafuman;

typedef struct

{

char data[N]; //字符数据

char copy[N][10\*N];//编码

}bianma;

void display();

int input(int w[], bianma \*bm);

void creat\_hafuman(hafuman ht[], int w[], int n);

void select(hafuman ht[], int m, int \*s1, int \*s2);

void encoding(hafuman ht[], bianma \*bm, int n);

void coding(bianma \*bm, int n);//解码

void coding\_2(hafuman ht[], bianma \*bm, int n);//解码

void output(bianma \*bm, int n);

int i, j, k;

void display()

{

printf("\n\n\n");

printf("\t\t\t1.输出编码\n\n");

printf("\t\t\t2.进行解码\n\n");

printf("\t\t\t3.退出\n\n");

printf("\t\t请选择(1~3): ");

}

int input(int w[], bianma \*bm)

{

int n=0;

printf("请输入你要进行编码的字符！\n");

while(1)

{

bm->data[n]=getchar();

if(bm->data[n]=='\n')

break;

n++;

}

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

{

w[i]=1;

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

{

if( bm->data[i] == bm->data[j] )

{

w[i]++;

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

{

bm->data[k]=bm->data[k+1];

}

n--; //覆盖完之后n--;

}

else

j++;

}

}

printf("\n\n");

printf("不同的字符有：\n");

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

{

printf("%c", bm->data[i]);

}

return n;

}

void creat\_hafuman(hafuman ht[], int w[], int n)

{

int s1, s2;

int t;

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

{

ht[t].weight=w[t-1];

ht[t].parent=0;

ht[t].lchild=0;

ht[t].rchild=0;

}

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

{

ht[t].weight=0;

ht[t].parent=0;

ht[t].lchild=0;

ht[t].rchild=0;

}

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

{

select(ht, t-1, &s1, &s2);//前i-1中选双亲为0, 权值最小

ht[t].weight=ht[s1].weight + ht[s2].weight;

ht[t].lchild=s1, ht[t].rchild=s2;

ht[s1].parent=t, ht[s2].parent=t;

}

}

void select(hafuman ht[], int m, int \*s1, int \*s2)

{

int min1, min2, a, b;

i=1;

while( ht[i].parent != 0)

{

i++;

}

min1=ht[i].weight;

a=i;

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

{

if(min1 > ht[j].weight && ht[j].parent==0)

{

min1=ht[j].weight;

a=j;

}

}

i=1;

while( ht[i].parent != 0 || a==i )

{

i++;

}

min2=ht[i].weight;

b=i;

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

{

if(j==a)

continue;

if(min2 > ht[j].weight && ht[j].parent==0)

{

min2=ht[j].weight;

b=j;

}

}

\*s1=a; \*s2=b;

}

void encoding(hafuman ht[], bianma \*bm, int n)//编码, \*copy[]为复制编码

{

int start, c, p;

char \*ch;

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

ch[n-1]='\0';

for(i=1; i<=n; i++)//n个叶子节点

{

start=n-1;

c=i, p=ht[i].parent; //p为parent, c为child

while(p!=0)

{

start--;

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

ch[start]='0';

else

ch[start]='1';

c=p; p=ht[p].parent; //printf("\n123\n");

}

strcpy( bm->copy[i-1], &ch[start] );

//printf("\n%s\n", bm->copy[i-1]);

}

free(ch);

}

void coding\_2(hafuman ht[], bianma \*bm, int n)//解码

{

char s[10\*N];

int p;

printf("\n请输入要解码的字符：\n");

fflush(stdin);

gets(s);

printf("\n解码为：\n\n");

p=2\*n-1;

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

{

if(s[i]=='0')

p=ht[p].lchild;

else

if(s[i]=='1')

p=ht[p].rchild;

if(ht[p].lchild == 0 && ht[p].rchild == 0)

{

printf("%c", bm->data[p-1]);//p: 1~~2\*n-1, bbm->data[0~~n-1]

p=2\*n-1;

continue;

}

}

puts("\n\n");

}

void output(bianma \*bm, int n)

{

printf("\n");

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

{

printf("%c\t", bm->data[i] );

printf("%s\n", bm->copy[i]);

}

}

int main()

{

hafuman ht[N];

bianma \*bm;

int w[N];

int n, m;

bm=(bianma \*)malloc( sizeof(bianma) );

n=input(w, bm);

printf("\n\n不同字符总数为: %d\n\n", n);

creat\_hafuman(ht, w, n);

encoding(ht, bm, n);

getch();

system("cls");

loop: display();

scanf("%d", &m);

switch(m)

{

case 1:

output(bm, n);

printf("\n\n请按任意键继续");

getch();

system("cls");

goto loop;

break;

case 2:

//coding(bm, n);//解码

coding\_2(ht, bm, n);//解码

printf("\n\n请按任意键继续");

getch();

system("cls");

goto loop;

break;

case 3:

break;

default:

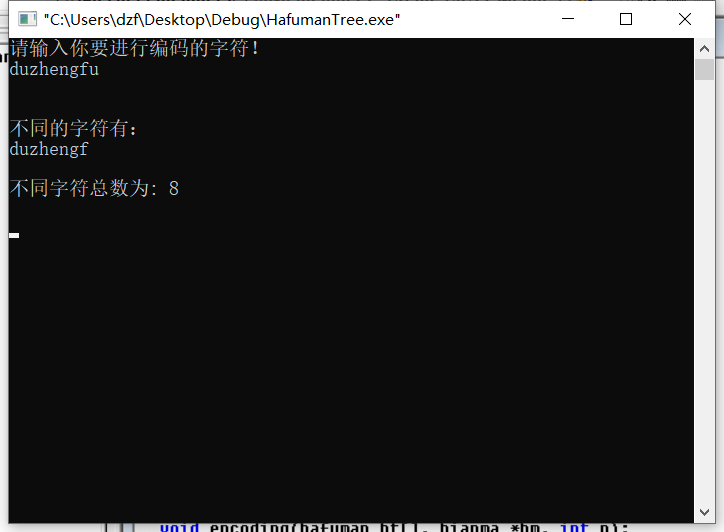
system("cls");

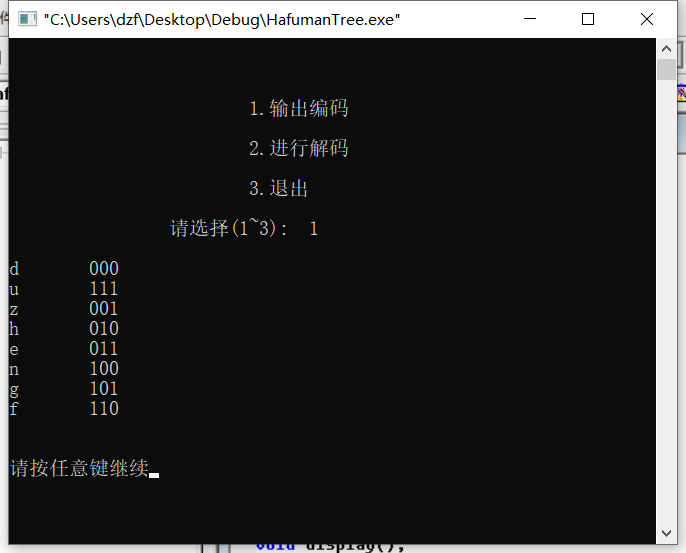
goto loop;

}

}

**运行截图：**





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。

**解码代码：**

void coding\_2(hafuman ht[], bianma \*bm, int n)//解码

{

char s[10\*N];

int p;

printf("\n请输入要解码的字符：\n");

fflush(stdin);

gets(s);

printf("\n解码为：\n\n");

p=2\*n-1;

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

{

if(s[i]=='0')

p=ht[p].lchild;

else

if(s[i]=='1')

p=ht[p].rchild;

if(ht[p].lchild == 0 && ht[p].rchild == 0)

{

printf("%c", bm->data[p-1]);//p: 1~~2\*n-1, bbm->data[0~~n-1]

p=2\*n-1;

continue;

}

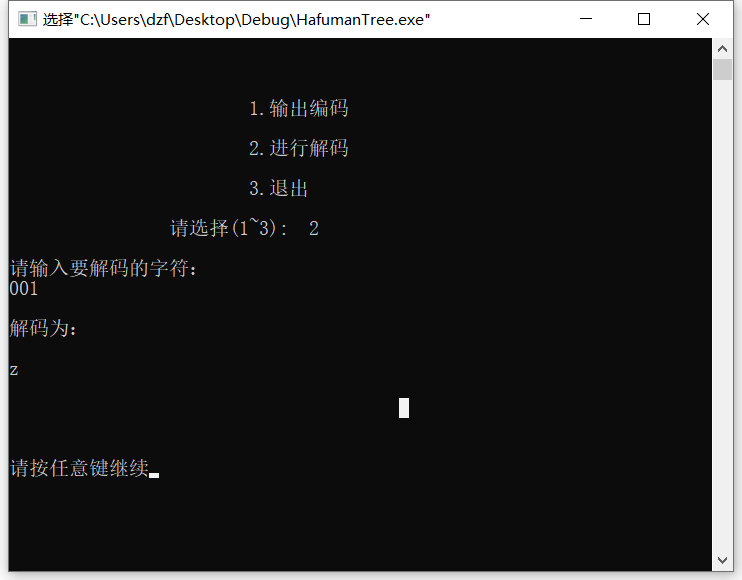
}

puts("\n\n");

}

**运行截图：**





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。

在英文中，e的出现[机率](https://baike.baidu.com/item/%E6%9C%BA%E7%8E%87/7497788" \t "https://baike.baidu.com/item/_blank)最高，而z的出现概率则最低。当利用哈夫曼编码对一篇英文进行压缩时，e极有可能用一个[比特](https://baike.baidu.com/item/%E6%AF%94%E7%89%B9/3431582" \t "https://baike.baidu.com/item/_blank)来表示，而z则可能花去25个[比特](https://baike.baidu.com/item/%E6%AF%94%E7%89%B9/3431582" \t "https://baike.baidu.com/item/_blank)（不是26）。用普通的表示方法时，每个英文字母均占用一个[字节](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82/1096318" \t "https://baike.baidu.com/item/_blank)，即8个[比特](https://baike.baidu.com/item/%E6%AF%94%E7%89%B9/3431582" \t "https://baike.baidu.com/item/_blank)。二者相比，e使用了一般编码的1/8的长度，z则使用了3倍多。

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**

用回溯法生成{1,2,3，4}的所有排列。

代码：

#include <stdio.h>

#define N 4

void print\_elems(int\* a)

{

int i;

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

{

printf("%d ", a[i]);

}

}

void dfs(int step, int\* a, int\* book)

{

if (step==N+1)

{

print\_elems(a);

printf("\n");

return;

}

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

{

if (\*(book + i) == 0)

{

\*(book + i) = 1;

a[step] = i;

dfs(step + 1, a, book);

\*(book + i) = 0;

}

}

}

int main(void)

{

int a[N+1] = { 0 };

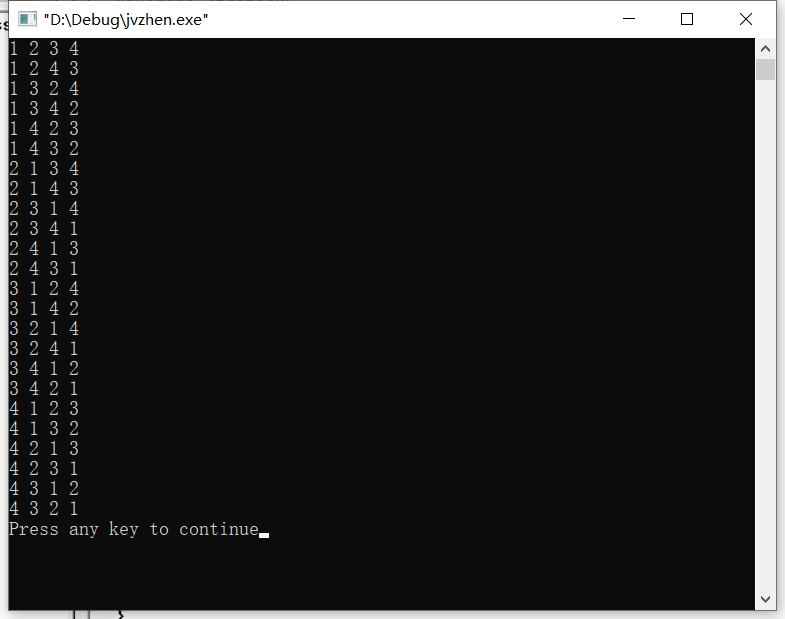
int book[N+1] = { 0 };

dfs(1, a, book);

return 0;

}

**运行截图：**



**第338页第7题**

写一个程序用分支界限算法对背包问题求解。

**代码：**

#include <stdio.h>

#include <stdlib.h>

typedef struct QNode{

int value; //当前结点的总价值

int weight; //当前的总重量

struct QNode \*next;

}QNode, \*QueuePtr;

typedef struct{

QueuePtr front;

QueuePtr rear;

}Queue;

int initQueue(Queue &Q)

{

Q.front=Q.rear=(QueuePtr)malloc(sizeof(QNode));

if(!Q.front)

return -1;

Q.front->next=NULL;

return 1;

}

int emptyQueue(Queue Q)

{

if (Q.front==Q.rear)

return 1;

else

return 0;

}

int destroyQueue(Queue &Q)

{

while(Q.front){

Q.rear=Q.front->next;

free(Q.front);

Q.front=Q.rear;

}

return 1;

}

int enQueue(Queue &Q, int value, int weight)

{

QueuePtr p=(QueuePtr)malloc(sizeof(QNode));

if(!p)

return -1;

p->value=value;

p->weight=weight;

p->next=NULL;

Q.rear->next=p;

Q.rear=p;

return 1;

}

int deQueue(Queue &Q, int &value, int &weight)

{

QueuePtr p;

if(Q.front==Q.rear)

return -1;

p=Q.front->next;

value=p->value;

weight=p->weight;

Q.front->next=p->next;

if(Q.rear==p) Q.rear=Q.front;

free(p);

return 1;

}

Queue loadingQueue;

int bestvalue, n;

void inQueue(int value, int weight, int i)

{

if(i==n-1)

{

if(value>bestvalue)

bestvalue=value;

}

else

enQueue(loadingQueue,value,weight);

}

int main()

{

int i,j,k;

int \*w, \*v, ew, ev;

int c;

printf("请输入物品的数量和背包的承重量:");

scanf("%d%d",&n,&c);

w=new int[n];

v=new int[n];

printf("请输入重量:");

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

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

printf("请输入价值:");

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

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

initQueue(loadingQueue);

enQueue(loadingQueue,-1,0);

//int deQueue(Queue &Q, int &value, int &weight)

//void inQueue(int value, int weight, int i)

//int enQueue(Queue &Q, int value, int weight)

i=0; //层数

ew=0; //扩展结点对应的承重量

ev=0;

while(true)

{

if(ew+w[i]<=c)

inQueue(ev+v[i],ew+w[i],i);

inQueue(ev,ew,i);

deQueue(loadingQueue, ev,ew);

if(ev==-1) //同层结点尾部

{

if(emptyQueue(loadingQueue))

{

printf("结果为 %d.\n",bestvalue);

}

enQueue(loadingQueue,-1,0);

deQueue(loadingQueue, ev,ew);

i++;

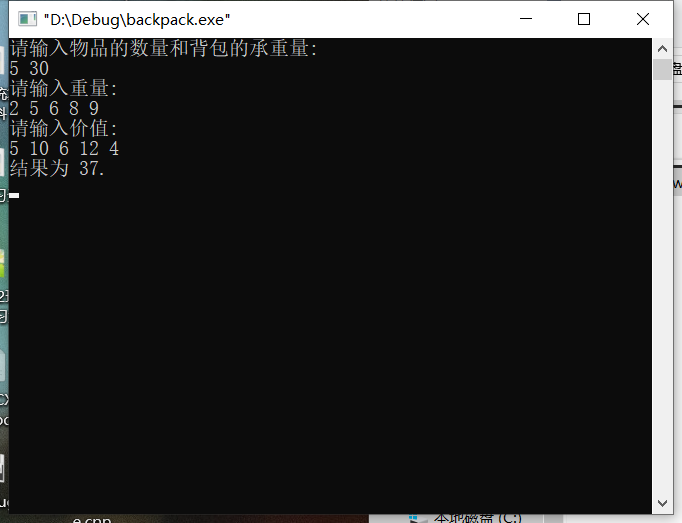
}

}

return 0;

}

**运行截图：**

****