

**计算机算法设计与分析**

**实训实验报告**

**班别： 大数据18班**

**学号： 201806404133**

**姓名： 黄燕丽**

**任务一：分别用循环和递归调用方式实现二分查找，语言不限。**

1. **循环**

**代码：**

#include<stdio.h>

int Find(int a[],int size,int to\_find)

{

int left=0;

int right=size-1;

while(left<=right)

{

int mid=(left+right)/2;

if(to\_find<a[mid])

right=mid-1;

else if(to\_find>a[mid])

left=mid+1;

else

return mid;

}

}

int main()

{

int a[]={1,2,3,4,5,6,7,8,9,10};

int to\_find;

printf("请输入要查找的数：");

scanf("%d",&to\_find);

int size=sizeof(a)/sizeof(a[0]);

int i=Find(a,size,to\_find);

if(i==-1)

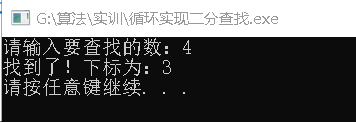
printf("没找到！\n");

else

printf("找到了！下标为：%d\n",i);

}

运行结果：



1. **递归**

代码：

#include<stdio.h>

int a[10]={1,2,3,4,5,6,7,8,9,10}; //定义查找数组；

int Find(int left,int right,int number) //left表示起点 right表示终点；

{

while(left<right)

{

int mid=(left+right)/2; //设置中间值，用中间值与number比较；

if(a[mid]==number) //等于的话，说明找到了这个元素；

return mid;

else if(a[mid]>number) //要找的值在mid的左边；

return Find(left,mid-1,number); //重新定义区间长度，调用Find函数；

else //要找的值在mid的右边；

return Find(mid+1,right,number); //重新定义区间长度，调用Find函数；

}

int main()

{

int number;

printf("请输入要查找的数: ");

scanf("%d",&number);

int pos=Find(0,9,number);

printf("原数据表: ");

for(int i=0;i<10;i++)

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

printf("\n");

if(pos>=0)

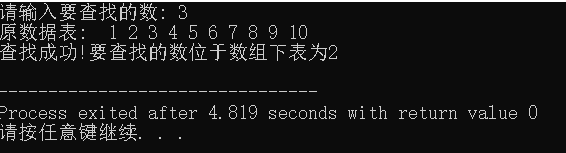
printf("查找成功!要查找的数位于数组下表为%d\n",pos);

else

printf("查找失败!!!\n");

}

运行结果：



**任务二：用递归调用实现快速排序和合并排序算法，语言不限，并随机生成测试算例，统计比较两种算法占用的时间。**

1. **递归实现快速排序**

**代码：**

#include <stdio.h>

#include <stdlib.h>

//快速排序

void quick\_sort(int a[],int left,int right) //直接排序数组，返回值为void

{

//定义左右指针指向数组的下标

int i = left;

int j = right;

//递归出口

if(i>j)

return;

//记录标杆值

int key=a[i];

//一次循环可以得到：标杆的左侧都比标杆小，标杆的右侧都比标杆大

while(i<j)

{

//从right开始找比标杆小的为止

while(i<j && a[j]>=key)

j--;

//将较小的数赋值到left上

a[i]=a[j];

//从left开始找比标杆大的为止

while(i<j && a[i]<=key)

i++;

//将较大的数赋值到right上

a[j]=a[i];

}

//将标杆值补到数组中

a[i]=key;

//一次循环完成

//进行左侧和右侧的递归

quick\_sort(a,left,i-1);

quick\_sort(a,i+1,right);

}

//调用快速排序

void quick\_sort(int a[],int len)

{

quick\_sort(a,0,len-1);

}

void show(int a[],int len)

{

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

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

printf("\n");

}

int main()

{

int arr[10]={1,3,5,7,9,-4,-2,-6,0,-9};

printf("原序列为：") ;

show(arr,10);

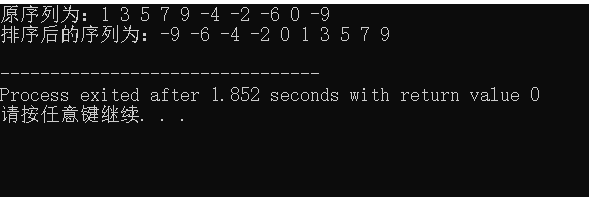
printf("排序后的序列为：") ;

quick\_sort(arr,10);

show(arr,10);

}

**运行结果：**



1. **递归实现合并排序**

**代码：**

#include <stdio.h>

void merge(int \*a, int b, int m, int e)

{

int l = m-b+1, r = e-m, i;

int L[l+1], R[r+1];

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

{

L[i] = a[b+i];

}

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

{

R[i] =a[m+i+1];

}

L[l] = 32767;

R[r] = 32767;

l = 0;

r = 0;

for(i=0; i< e-b+1; i++)

{

if(L[l] < R[r])

{

a[b+i] = L[l];

l ++;

}

else {

a[b+i] = R[r];

r ++;

}

}

}

void merge\_sort(int \*a, int b, int e)

{

if(b < e)

{

int m = (b + e) / 2;

merge\_sort(a, b, m);

merge\_sort(a, m+1, e);

merge(a, b, m, e);

}

}

int main()

{

int a[500];

int len, i;

printf("请输入数组长度:");

scanf("%d", &len);

printf("请输入数组元素:");

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

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

merge\_sort(a, 0, len-1);

printf("归并排序后的结果为:\n");

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

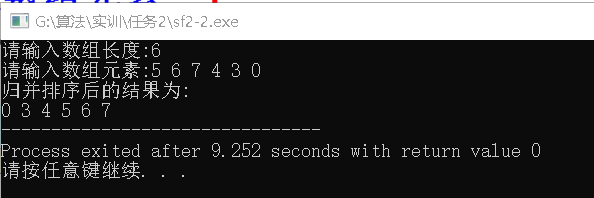
{

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

}

}

运行结果：



**任务3：设计算法从给定的n个数中找出最小的k个数，采用递归调用实现算法。**

**代码：**

#include <stdio.h>

#include <stdlib.h>

/\* 产生start到end的随机数\*/

int randomInRange(int start, int end)

{

int ret = start + rand()%(end - start + 1);

return ret;

}

/\* 交换两个数字的位置\*/

void swap(int \*num1, int \*num2)

{

int tmp = \*num1;

\*num1 = \*num2;

\*num2 = tmp;

}

/\* 返回选中数字位置，并使得选中的数字的左边的数都小于data[index],选中数字右边的数都大于data[index]\*/

int partition(int \*data, int length, int start, int end)

{

int ret = 0;

int index = randomInRange(start, end);

int small = start - 1;

swap(&data[index], &data[end]);

if (data == NULL || length < 0 ||start < 0 || end >= length)

{

ret = -1;

printf("error:%d\n", ret);

return ret;

}

for (index = start; index < end; index++)

{

if(data[index] < data[end])

{

++small;

if (small != index)

{

swap(&data[index], &data[small]);

}

}

}

++small;

swap(&data[small],&data[end]);

return small;

}

void GetLeastNum(int \*input, int n, int \*output, int k)

{

int start = 0;

int end = n - 1;

int i = 0;

int index = partition(input, n, start, end);

if (input == NULL || output == NULL ||k > n||k <= 0)

{

return ;

}

while (index != k-1)

{

if (index > k-1)

{

end = index - 1;

index = partition(input, n, start, end);

}

else

{

start = index + 1;

index = partition(input, n, start, end);

}

}

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

{

output[i] = input[i];

}

}

int main()

{

int data[] = {4, 5, 1, 6, 2, 7, 3, 8};

int outdata[10] = {0};

GetLeastNum(data, 8, outdata, 4);

int i = 0;

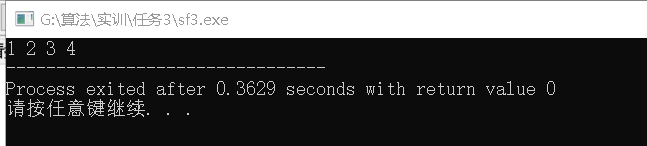
for (i = 0; i < 4; i++)

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

return 0;

}

**运行结果：**



**任务:4：分别采用循环和递归调用方式实现0-1背包问题的动态规划算法。**

1. **循环算法**

**代码：**

#include<stdio.h>

int package(int n, int w[], int v[], int x[], int c);

int max(int a, int b);

int value[100][100];//value[i][j]前i个物品装入最大容量为j的背包中最大价值

int package(int n, int w[], int v[], int x[], int c)

{

int i,j;//填表，第一行和第一列全为0

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

{

value[i][0]=0;

}

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

{

value[0][j]=0;

}

//找最优解

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

{

printf("%d %d %d ",i,w[i-1],v[i-1]);

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

{

if(j < w[i-1])

{

value[i][j]=value[i-1][j];

printf("[%d][%d]=%2d",i,j,value[i][j]);

}

else

{

value[i][j]=max(value[i-1][j], value[i-1][j-w[i-1]]+v[i-1]);

printf("[%d][%d]=%2d",i,j,value[i][j]);

}

}

printf("\n");

}

//判断物品标记状态

j=c;

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

{

if(value[i][j] > value[i-1][j])

{

x[i]=1;

j = j - w[i-1];//放入一个物品后的背包最大容量

}

else

{

x[i]=0;

}

}

printf("选中的物品是：");

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

{

printf("%2d",x[i]);

}

printf("\n");

return(value[n][c]);

}

int max(int a, int b)

{

if(a >= b)

{

return a;

}

else

{

return b;

}

}

int main()

{

int i;

int n; //物品数量

int c; //背包最大容量

int s;//获得最大价值

int w[20];//物品质量

int v[20];//物品价值

int x[20];//物品标记状态

printf("输入背包最大容量：");

scanf("%d",&c);

printf("输入物品数量（不超过20件）：");

scanf("%d",&n);

printf("输入物品各质量：\n");

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

{

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

}

printf("输入物品各价值：\n");

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

{

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

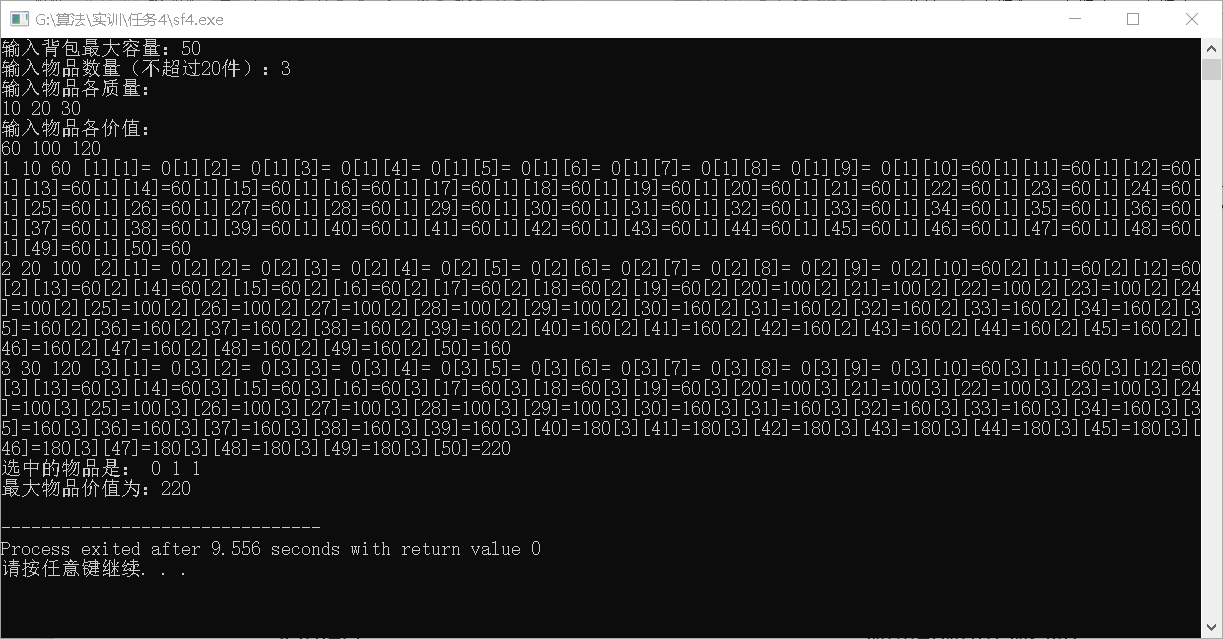
}

s=package(n,w,v,x,c);

printf("最大物品价值为：%d\n",s);

}

运行结果：



1. **递归算法：**

**代码：**

#include "stdio.h"

#define C 10 //背包容量数

#define N 5 //物品总数

void Knapsack(int \*w, int \*v, int n, int c, int (\*m)[C+1])

{

//先处理记录表格中的最后一行，也即完成对m[n][j]的“填空”

int j,i;

int lowc = w[n]-1 > c ? c : w[n]-1; //判断 只有一个物品时，背包容量和 这个物品的重量的相对大小

//可以避免在数组赋值中防止越界

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

m[n][j] = 0;

for (j = lowc+1; j <= c; j++) /\*lowc+1等价于w[n]\*/

m[n][j] = v[n];

//循环处理记录表格中的其他行，也即完成对m[i][j]的“填空”

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

{

lowc = w[i]-1 > c ? c : w[i]-1;

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

m[i][j] = m[i+1][j];

for ( j = w[i]; j <= c; j++)

{

int t1 = m[i+1][j];

int t2 = m[i+1][j - w[i]] + v[i];

m[i][j] = t1 > t2 ? t1 : t2;

}

}

//处理记录表格中的第一行，也即完成对m[1][c]的“填空”

m[1][c] = m[2][c]; //背包容量c不足,物品1舍去

if(c >= w[1] && m[2][c-w[1]] + v[1] > m[2][c]) //背包容量c足够,并且选取物品1的情况价值较大

m[1][c] = m[2][c-w[1]] +v[1];

}

/\*

该函数用来确定最终装入背包的商品是哪些

\*/

void Traceback(int (\*m)[C+1],int \*w, int \*x,int n, int c)

{

int i;

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

{

if(m[i][c] == m[i+1][c])

x[i] = 0;

else

{

x[i] = 1;

c -= w[i];

}

}

x[n] = (m[n][c])?1:0;

}

int main(void)

{

int i ;

int x[N+1];

int w[N+1] ={-1,2,2,6,5,4}; //各物品的质量

int v[N+1] ={-1,6,3,5,4,6}; //各物品的价值

int m[N+1][C+1];

Knapsack(w,v,N,C,m);

printf("%d\n",m[1][C]);

Traceback(m, w, x, N, C);

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

{

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

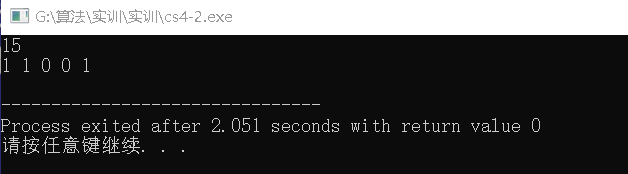
}

printf("\n");

return 0;

}

**运行结果：**



**任务5：分别采用循环和递归调用方式实现公共最长子序列的动态规划算法。**

**代码：**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

int LCSLength(char\* str1, char\* str2, int \*\*b)

{

int i,j,length1,length2,len;

length1 = strlen(str1);

length2 = strlen(str2);

//双指针的方法申请动态二维数组

int \*\*c = new int\*[length1+1]; //共有length1+1行

for(i = 0; i < length1+1; i++)

c[i] = new int[length2+1];//共有length2+1列

for(i = 0; i < length1+1; i++)

c[i][0]=0; //第0列都初始化为0

for(j = 0; j < length2+1; j++)

c[0][j]=0; //第0行都初始化为0

for(i = 1; i < length1+1; i++)

{

for(j = 1; j < length2+1; j++)

{

if(str1[i-1]==str2[j-1])//由于c[][]的0行0列没有使用，c[][]的第i行元素对应str1的第i-1个元素

{

c[i][j]=c[i-1][j-1]+1;

b[i][j]=0; //输出公共子串时的搜索方向

}

else if(c[i-1][j]>c[i][j-1])

{

c[i][j]=c[i-1][j];

b[i][j]=1;

}

else

{

c[i][j]=c[i][j-1];

b[i][j]=-1;

}

}

}

/\*

for(i= 0; i < length1+1; i++)

{

for(j = 0; j < length2+1; j++)

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

printf("\n");

}

\*/

len=c[length1][length2];

for(i = 0; i < length1+1; i++) //释放动态申请的二维数组

delete[] c[i];

delete[] c;

return len;

}

void PrintLCS(int \*\*b, char \*str1, int i, int j)

{

if(i==0 || j==0)

return ;

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

{

PrintLCS(b, str1, i-1, j-1);//从后面开始递归，所以要先递归到子串的前面，然后从前往后开始输出子串

printf("%c",str1[i-1]);//c[][]的第i行元素对应str1的第i-1个元素

}

else if(b[i][j]==1)

PrintLCS(b, str1, i-1, j);

else

PrintLCS(b, str1, i, j-1);

}

int main(void)

{

char str1[100],str2[100];

int i,length1,length2,len;

printf("请输入第一个字符串：");

gets(str1);

printf("请输入第二个字符串：");

gets(str2);

length1 = strlen(str1);

length2 = strlen(str2);

//双指针的方法申请动态二维数组

int \*\*b = new int\*[length1+1];

for(i= 0; i < length1+1; i++)

b[i] = new int[length2+1];

len=LCSLength(str1,str2,b);

printf("最长公共子序列的长度为：%d\n",len);

printf("最长公共子序列为：");

PrintLCS(b,str1,length1,length2);

printf("\n");

for(i = 0; i < length1+1; i++)//释放动态申请的二维数组

delete[] b[i];

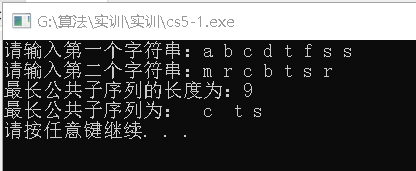
delete[] b;

system("pause");

return 0;

}

**运行结果：**



**任务6：背包问题的贪心算法**

代码：

#include<stdio.h>

#define maxnumber 20

typedef struct node{

int w;

int v;

int i;

}Object;

float find(Object wp[],int n,float M)

{

int x[maxnumber];

int i;

int maxprice=0;

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

{

x[i]=0.0;

}

i=0;

while(wp[i].w<M)

{

x[wp[i].i]=1;

M=M-wp[i].w;

i++;

}

x[wp[i].i]=M/wp[i].w;

printf("解向量是：\n");

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

{

printf("x[%d]=%f",i,x[i]);

}

printf("\n");

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

{

maxprice=maxprice+wp[i].v\*x[wp[i].i];

}

return maxprice;

}

int main()

{

Object wp[maxnumber];

int i,j,n;

int C;

int flag;

int maxprice,temp;

printf("请输入物品的总数:");

scanf("%d",&n);

printf("请输入背包的重量：");

scanf("%d",&C);

printf("\n请输入物品的序号、重量和价值:");

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

{

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

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

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

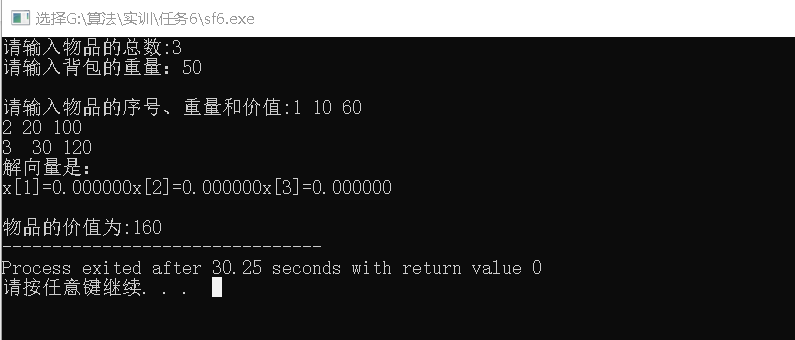
}

maxprice=find(wp,n,C);

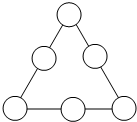
printf("\n物品的价值为:%d",maxprice);

}

**运行结果：**



**任务7：用回溯法或者分支限界法实现：用1-9其中的6个自然数组成一个三角形，让这个三角形的边的值之和相等。**



**代码：**

#include<stdio.h>

#include<stdlib.h>

int main()

{

int a,b,c,d,e,f;

for(a=1;a<=6;a++)

for(b=1;b<=6;b++)

for(c=1;c<=6;c++)

for(d=1;d<=6;d++) //此处笔误，应该是d

for(e=1;e<=6;e++)

for(f=1;f<=6;f++)

if(a!=b&&a!=c&&a!=d&&a!=e&&a!=f&&b!=c&&b!=d&&b!=e&&b!=f&&c!=d&&c!=e&&c!=f&&d!=e&&d!=f&&e!=f)

if(a+b+c==c+d+e&&c+d+e==e+f+a)

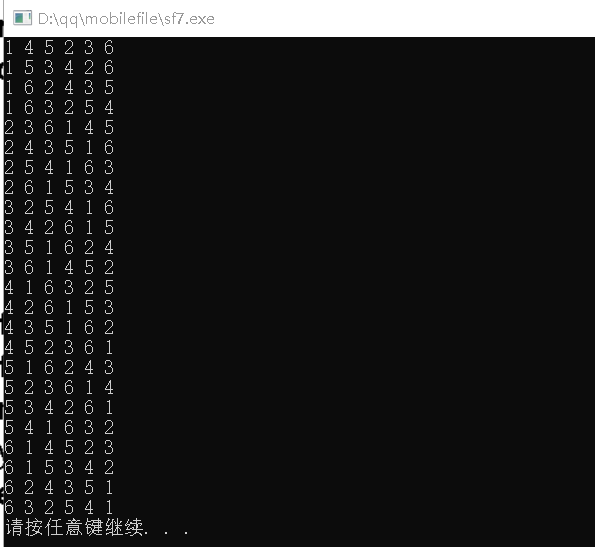
printf("%d %d %d %d %d %d\n",a,b,c,d,e,f);

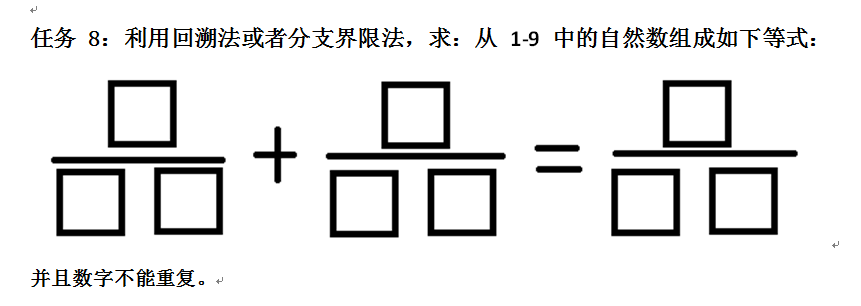
system("pause");

return 0;

}

**运行结果：**



**任务8：**

代码如下：

#include <stdio.h>

int main()

{

int g,i,k,s,a[10];//定义四个变量及一个数组，数组用来表示式中的每一个口

long m1,m2,m3;//定义三个长整型变量m1,m2,m3作为为分母

printf("桥本分数式有:\n");

i=1;//初始化i=1

a[1]=1;//初始化a[1]=1

s=0;//初始化s=0

while (1)//判断条件为真，只要不是“0”，这个循环就一直循环下去

{

g=1;//令g=1

for(k=i-1;k>=1;k--)//for循环的条件是当k=i-1,也就是当k=9，k>=1,k--

{

if(a[i]==a[k]) //当 a[i]==a[k]，即两个数相同,标记g=0

{

g=0;

break;

}

}

if(i==9 && g==1 && a[1]<a[4])//令a[1]<a[4]主要是为了避免重复 ， 比如a/bc+f/de=g/hi和 f/de+a/bc=g/hi是相同的

{

m1=a[2]\*10+a[3];//第一个分母= a[2]\*10+a[3]

m2=a[5]\*10+a[6];//第二个分母 =a[5]\*10+a[6]

m3=a[8]\*10+a[9];//第三个分母=a[8]\*10+a[9]

if(a[1]\*m2\*m3+a[4]\*m1\*m3==a[7]\*m1\*m2) // 判断等式

{

s++;//等式成立，s加 1

printf("(%2d) ",s);//输出s

printf("%d/%ld+%d/",a[1],m1,a[4]);

printf("%ld=%d/%ld ",m2,a[7],m3);

if(s%2==0)

printf("\n");

}

}

if(i< 9 && g==1)

{

i++;

a[i]=1;

continue;

} // 不到9个数,往后继续

while(a[i]==9 && i>1)

i--; // 往前回溯

if(a[i]==9 && i==1)

break;

else

a[i]++; // 至第1个数为9结束

}

printf(" 共以上%d个解。\n",s);

}

运行结果如下：

