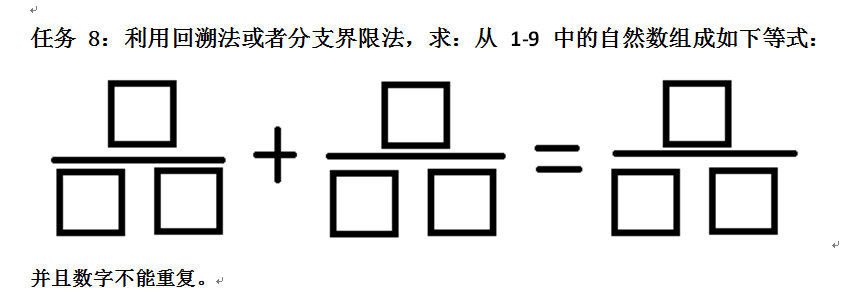
任务1:分别用循环和递归调用的方式实现二分查找，语言不限; (C 语言)。

任务2: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。  
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。  
任务4分别采用循环和递归调用方式实现01背包问题的动态规划算法。  
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。  
任务6:背包问题的贪心算法  
任务7:用回溯法或者分支界限法，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。( 语言不限)。



任务1-1（递归调用实现二分查找）：

代码：

#include<string.h>

#include<stdio.h>

int k;

int binarysearch(int a[],int x,int low,int high)//a表示需要二分的有序数组（升序），x表示需要查找的数字，low，high表示高低位

//binarysearch:二分搜索/折半搜索

{

if(low>high)

{

return -1;//没有找到

}

int mid=(low+high)/2;

if(x==a[mid])//找到x

{

k=mid;

return x;

}

else if(x>a[mid]) //x在后半部分

{

binarysearch(a,x,mid+1,high);//在后半部分继续二分查找

}

else//x在前半部分

{

binarysearch(a,x,low,mid-1);

}

}

int main()

{

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

printf("请输入需要查找的正数字：\n");

int x;

scanf("%d",&x);

int r=binarysearch(a,x,0,9);

if(r==-1)

{

printf("没有查到\n");

}

else

{

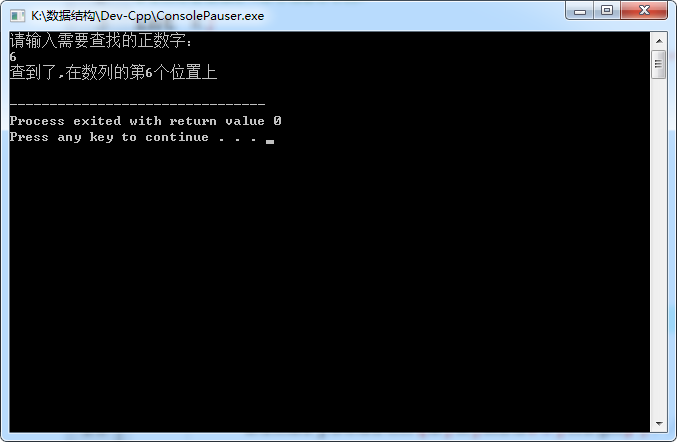
printf("查到了,在数列的第%d个位置上\n",k+1);

}

return 0;

}

运行截图：



任务1-2（循环实现递归调用）：

#include <stdio.h>

#include <stdlib.h>

int BinaryFind(int arr[], int size, int to\_find)

{

int left = 0;

int right = size - 1;

while (left <= right)

{

int mid = (left + right) / 2;

if (to\_find < arr[mid])

{

right = mid - 1;

}

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

{

left = mid + 1;

}

else

{

return mid;

}

}

return -1;//未找到返回 -1

}

int main()

{

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

int to\_find = 4;

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

int i = BinaryFind(arr, size, to\_find);

//判定找到与否

if (i == -1)

{

printf("没找到！\n");

}

else

{

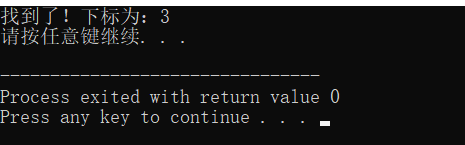
printf("找到了！下标为：%d\n", i);

}

system("pause");

return 0;

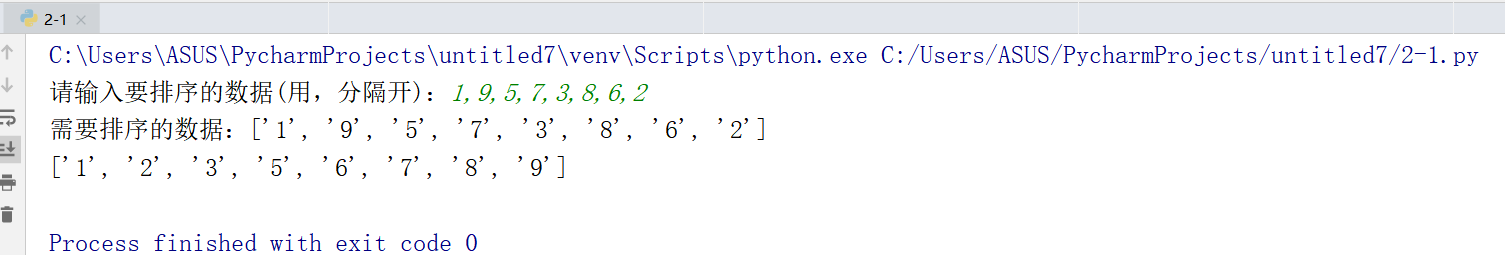
}



任务2-1（快速排序法）：

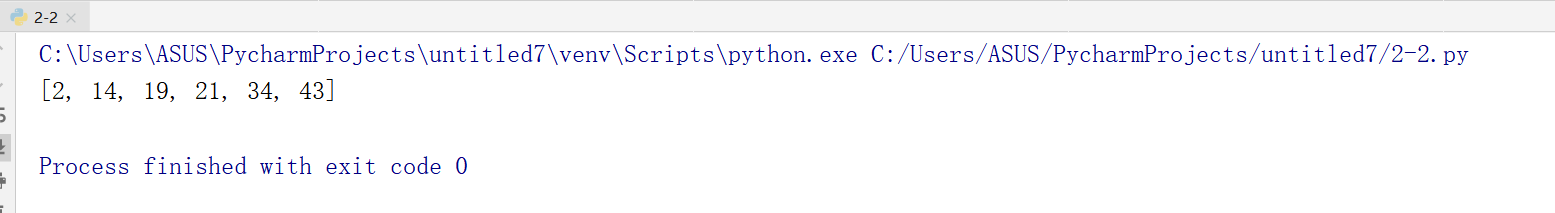
**def** quick\_sort(lst):  
 **if** len(lst) < 2:  
 **return** lst  
 **else**:  
 pivot = lst[0]  
 small\_lst = [I **for** I **in** lst[1:] **if** I < pivot]  
 large\_lst = [J **for** J **in** lst[1:] **if** J > pivot]  
 **return** quick\_sort(small\_lst) + [pivot] + quick\_sort(large\_lst)  
  
lst =input(**'请输入要排序的数据(用，分隔开)：'**)  
lst = lst.split(**','**)  
print(**f'需要排序的数据：{**lst**}'**)  
print(quick\_sort(lst))

截图：



任务2-2（合并排序法）：

**def** merge(a, b):  
 c = []  
 h = j = 0  
 **while** j < len(a) **and** h < len(b):  
 **if** a[j] < b[h]:  
 c.append(a[j])  
 j += 1  
 **else**:  
 c.append(b[h])  
 h += 1  
 **if** j == len(a):  
 **for** i **in** b[h:]:  
 c.append(i)  
 **else**:  
 **for** i **in** a[j:]:  
 c.append(i)  
 **return** c  
**def** merge\_sort(lists):  
 **if** len(lists) <= 1:  
 **return** lists  
 middle = len(lists)//2  
 left = merge\_sort(lists[:middle])  
 right = merge\_sort(lists[middle:])  
 **return** merge(left, right)  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 a = [14, 2, 34, 43, 21, 19]  
 print (merge\_sort(a))



任务3：

#include<stdio.h>

#include<stdlib.h>

int n,k,count=0; //全局变量

int a[21];

int isprime(int num){

int i;

for(i=2;i\*i<=num;i++)

if(num%i==0)

return 0;

return 1;

}

//p标记当前元素的下标，left 标记还有几个数没选出来

void dfs(int sum,int p,int left)

{

if(left==0)

{

if(isprime(sum))

{

count++;

return ;

}

}

//n-left是当前所选下标的最大值，若超过，则会选不够；因为这是按照从前往后的顺序以此进行选择

else for(int i=p;i<=n-left;i++)

dfs(sum+a[i],i+1,left-1); //每递归一次，就会选出一个数来，left就减一

}

int main(){

int i;

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

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

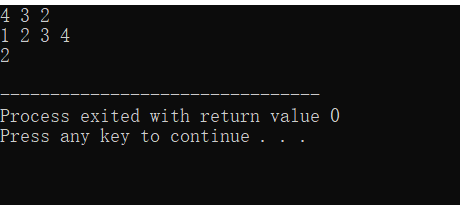
scanf("%d",a+i);

dfs(0,0,k);

printf("%d\n",count);

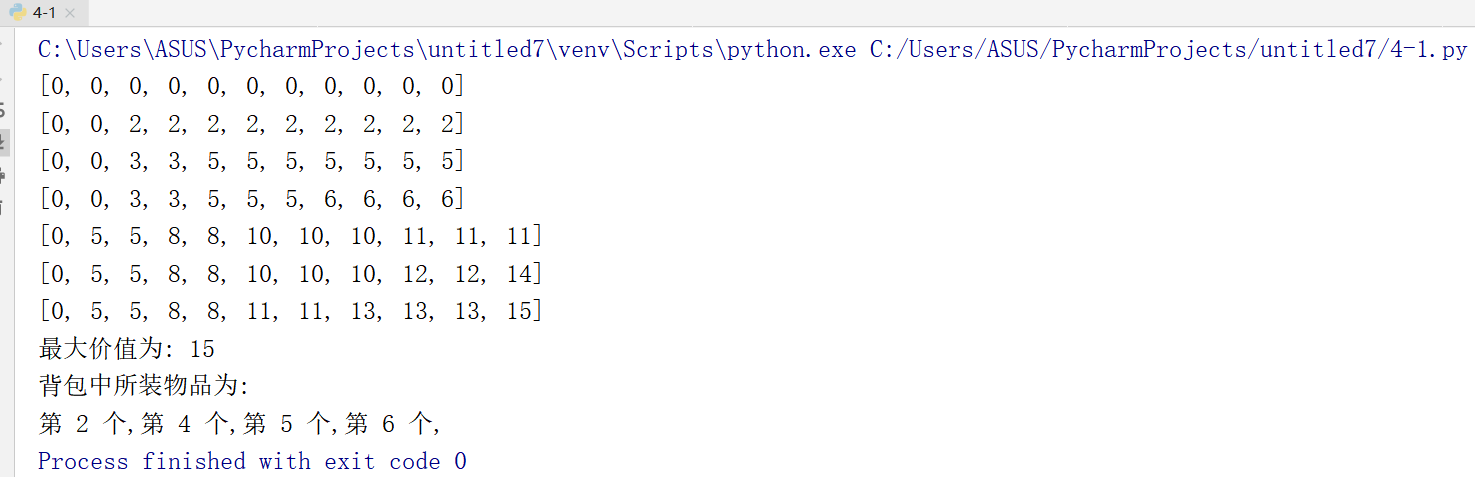
return 0;

}



任务4-1（循环）：

**def** bag(n, c, w, v):  
 value = [[0 **for** j **in** range(c + 1)] **for** i **in** range(n + 1)]  
 **for** i **in** range(1, n + 1):  
 **for** j **in** range(1, c + 1):  
 value[i][j] = value[i - 1][j]  
 *# 背包总容量够放当前物体，遍历前一个状态考虑是否置换* **if** j >= w[i - 1] **and** value[i][j] < value[i - 1][j - w[i - 1]] + v[i - 1]:  
 value[i][j] = value[i - 1][j - w[i - 1]] + v[i - 1]  
 **for** x **in** value:  
 print(x)  
 **return** value  
  
**def** show(n, c, w, value):  
 print(**'最大价值为:'**, value[n][c])  
 x = [**False for** i **in** range(n)]  
 j = c  
 **for** i **in** range(n, 0, -1):  
 **if** value[i][j] > value[i - 1][j]:  
 x[i - 1] = **True** j -= w[i - 1]  
 print(**'背包中所装物品为:'**)  
 **for** i **in** range(n):  
 **if** x[i]:  
 print(**'第'**, i+1, **'个,'**, end=**''**)  
**def** bag1(n, c, w, v):  
 values = [0 **for** i **in** range(c+1)]  
 **for** i **in** range(1, n + 1):  
 **for** j **in** range(c, 0, -1):  
 **if** j >= w[i-1]:  
 values[j] = max(values[j-w[i-1]]+v[i-1], values[j])  
 **return** values  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 n = 6  
 c = 10  
 w = [2, 2, 3, 1, 5, 2]  
 v = [2, 3, 1, 5, 4, 3]  
 value = bag(n, c, w, v)  
 show(n, c, w, value)



任务4-2（递归调用）：

#include<iostream>

#include<cstdio>

#include<string>

using namespace std;

int limitM;//限制的总重量

int option[100];

int n;//物品种数

struct Bag

{

int weight;

int value;

} num[100];

int find(int n, int M)

{

if (n==0 || M==0)//当物品数量为0，或者背包容量为0时，最优解为0

{

return 0;

}

else

{

//从当前所剩物品的最后一个物品开始向前，逐个判断是否要添加到背包中

for (int i=n-1; i>=0; i--)

{

//如果当前要判断的物品重量大于背包当前所剩的容量，那么就不选择这个物品

//在这种情况的最优解为f(n-1,C)

if (num[i].weight>M)

{

option[i]=0;

return find(n-1,M);

}

else

{

int temp1 = find(n-1,M);//不选择物品i的情况下的最优解

int temp2 = num[i].value + find(n-1,M-num[i].weight);//选择物品i的情况下的最优解

//返回选择物品i和不选择物品i中最优解大的一个

if (temp1 > temp2)

{

option[i]=0;//这种情况下表示物品i未被选取

return temp1;

}

else

{

option[i]=1;//物品i被选取

return temp2;

}

}

}

}

}

int main()

{

int k;

char c;

cout<<"物品种数：";

cin>>n;

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

{

cout<<"第"<<k+1<<"种物品(重量，价值):";

cin>>num[k].weight>>c>>num[k].value;

}

cout<<"背包所能承受的总重量：";

cin>>limitM;

cout<<"最佳装填方案是："<<endl;

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

{

if (option[i]=1)//为1表示相应的物品被选取

{

cout << "第"<<i+1<<"种物品"<<endl;

}

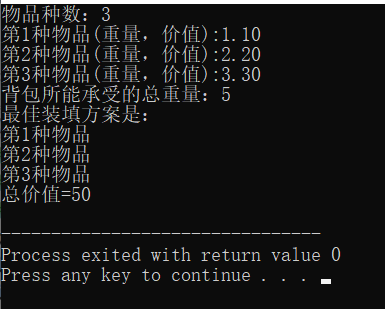
}

cout<<"总价值="<<find(n,limitM)<<endl;

return 0;

}

截图：



任务5-1（循环）：

#include<iostream>

#include<cstring>

#include<algorithm>

#include<cstdio>

#include<limits.h>

using namespace std;

int dp[1000];

int judge(int Start, int End, char\*s,char c,int\*Position)

{

for (int i = Start; i <= End; i++)

{

if (c == s[i])

{

\*Position = i;

return 1;

}

}

return 0;

}

int main()

{

int k,T,a[100];

char s1[100],s2[100];

while(scanf("%d",&T)!=EOF)

{

while (T--)

{

scanf("%s%s", s1, s2);

memset(dp, 0, sizeof(dp));

int c = 0;

int len1= strlen(s1);

int len2 = strlen(s2);

while (c<len2&&s1[0] != s2[c++]);

if (c == len2)

{

a[0] = len2 - 1;

dp[0] = 0;

}else

{

a[0]=c-1;

dp[0]=1;

}

for (int i = 1; i < len1; i++)

{

int Position, Max = -1;

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

{

int num = dp[k] + judge(a[k], len2, s2,s1[i], &Position);

if (num>Max)

{

Max = num;

}

}

dp[i] = Max;

a[i] = Position;

}

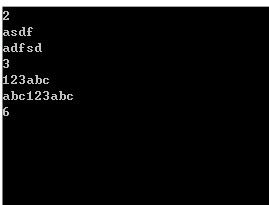
printf("%d\n", dp[len1 - 1]);

}

}

return 0;

}



任务5-2（递归调用）：

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#define MAX 100

#define max(a,b) (a>b)?a:b

int LCS(char stra[],int lena,char strb[],int lenb){

if (lena<=0 || lenb<=0) return 0;

int tmp=0,ret=0,issame=0;//issame串尾字符相同时取1;

if(stra[lena-1] == strb[lenb-1]) issame = 1;

tmp = max(LCS(stra,lena-1,strb,lenb),LCS(stra,lena,strb,lenb-1));

return ret = max(tmp , (LCS(stra,lena-1,strb,lenb-1) + issame));

}

int main(){

char stra[] = "gccct";

char strb[] = "gcgc";

int lena = strlen(stra)/sizeof(char);

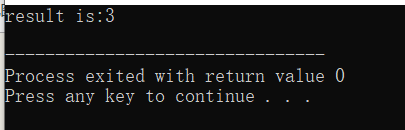
int lenb = strlen(strb)/sizeof(char);

int ret = LCS(stra,lena,strb,lenb);

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

return 0;

}



任务6：

#include <iostream>

#include <algorithm>

using namespace std;

struct item

{

int weight;//物品的重量

int value;//物品的价值

float bi;//物品单位重量的价值

float rate;//使用率：1代表物品完整放入，小于1代表被分割后放入

}items[100];

bool cmp(const item &a,const item &b)

{

return a.bi>b.bi;

}

int main(){

int n;//n件物品

float c;//背包容量为c

cout<<"输入物品件数和背包容量："<<endl;

cin>>n>>c;

cout<<"依次输入每件物品的价值和重量："<<endl;

float v[n],w[n];//v[n]：n件物品的价值，w[n]：n件商品的重量

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

{

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

items[i].bi=items[i].value/items[i].weight;//计算单位重量价值

items[i].rate=0;//初始化每件物品的使用率

}

sort(items,items+n,cmp);//按照单位重量的价值排序

int sum=0,j=0;

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

{

if(items[j].weight<=c)

{//选择单位价值重量最大的并且不超过背包容量的

items[j].rate=1;

sum+=items[j].weight;

c-=items[j].weight;

cout<<"重："<<items[j].weight<<"、价值："<<items[j].value<<"的物品被放入了背包"<<endl<<"放入比例："<<items[j].rate<<endl;

}

else break;

}

if(j<n)

{//物品未装完

items[j].rate=c/items[j].weight;//背包容量还剩c,计算出未装入的物品能装多少的比例

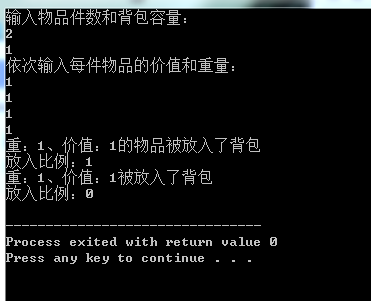
sum+=items[j].rate\*items[j].weight;//加上装入部分比例物品的重量

cout<<"重："<<items[j].weight<<"、价值："<<items[j].value<<"被放入了背包"<<endl<<"放入比例："<<items[j].rate<<endl;

}

return 0;

}



任务7（回溯法）：

#include <stdio.h>

#include <stdlib.h>

int main()

{

int a, b, c, ab, bc, ac;

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

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

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

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

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

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

{

if (a != b&&a != c&&a != ab&&a != bc&&a != ac&&b != c&&b != ab&&b != bc&&b != ac

&&c != ab&&c != bc&&c != ac&&ab != bc&&ab != ac&&bc != ac)

{

if (a + ab + b == a + ac + c&&a + ab + b == b + bc + c)

{

printf("这个三角形有\n");

printf(" %d \n", a);

printf(" %d %d \n", ab, ac);

printf("%d %d %d\n", b, bc, c);

}

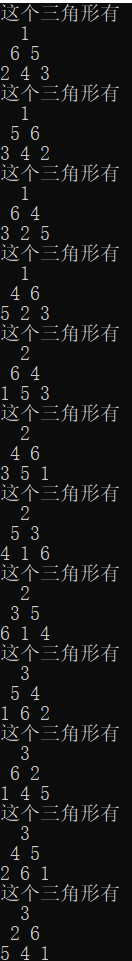
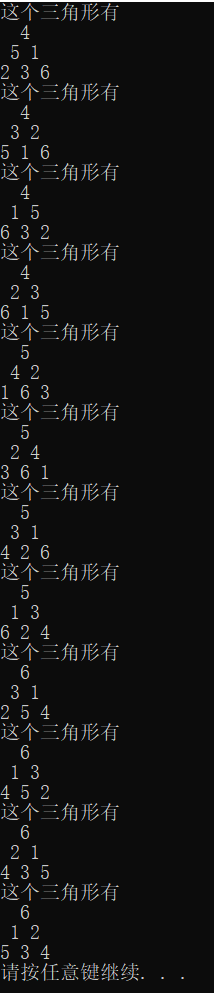
}

}

system("pause");

return 0;

}

任务8（回溯法）：

#include <stdio.h>

int main()

{

int i,k,g,s;

int m1,m2,m3,a[10];

a[1]=1;i=1;g=1;s=0;

while(1)

{

g=1;

for(k=i-1;k>0;k--) //注意此处很容易由于习惯错写成 for(k=i-1;i>0;i--)

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

{

g=0;

break;

} //两数相同，标记g=0

if(i==9 && g==1 && a[1]<a[4])

{ //为了避免解的重复所以a[1]<a[4]

m1=a[2]\*10+a[3]; //m1为第一个分数的分母

m2=a[5]\*10+a[6]; //m2为第一个分数的分母

m3=a[8]\*10+a[9]; //m3为第一个分数的分母

if(a[1]\*m2\*m3+a[4]\*m1\*m3==a[7]\*m1\*m2)//将分数式化为相乘的式子

{

s++;

printf("%d/%d+%d/%d=%d/%d \t",a[1],m1,a[4],m2,a[7],m3);

if(s%2==0) printf("\n");

}

}

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

{

i++;

a[i]=1;

continue;

} //向前继续走,执行continue语句直接跳到while语句，则不在执行下面的语句

while(a[i]==9 && i>1) i--; //向上一步回溯

if(a[i]==9 && i==1) break; //注意此处不能简写成 if(a[1]==9)

else a[i]++;

}

printf("共有%d个解！",s);

}

