1. 分别用循环和递归调用方式实现二分查找，语言不限

1.1循环调用   
def A(lst ,y):  
    left=0  
    right=len(lst)-1  
  
    while (left<=right):  
        middle = (left + right) // 2  
        if (lst[middle]==y):  
            return y  
        elif (lst[middle]>y):  
           #return A(lst[:middle],y)  
           right=middle-1  
        elif (lst[middle]<y):  
            #return A(lst[middle+1:],y)  
            left=middle+1  
lst = [1,2,3,4,5,6,7]  
print(A(lst,6))

1.2递归调用  
  
def binarySort(lst,value):  
    min=0 #下标从0开始  
    max=len(lst)-1  
    if min>max:  
        return -1  
    mid=(min+max)//2 #中间数并取整  
    if lst[mid]==value:  
        return value  
    elif lst[mid]>value:  
        return binarySort(lst[:mid],value)  
    elif lst[mid<value]:  
        return binarySort(lst[mid+1:],value)  
lst = [1,2,3,4,5,6,9]  
#lst = sorted(lst)  
print(binarySort(lst,9))

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时间。

2.1递归调用序列实现顺序排序

#include<stdio.h>

void Split(int left,int a[],int right);

int Quicksort(int left,int a[],int right);

int main()

{

int N=8;

scanf("%d",&N);

int a[8];

int i;

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

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

Split(0,a,N-1);

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

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

return 0;

}

void Split(int left,int a[],int right)

{

int middle;

if(left<right)

{

middle=Quicksort(left,a,right);

Split(left,a,middle-1);

Split(middle+1,a,right);

}

else

return;

}

int Quicksort(int left,int a[],int right)

{

int tag=a[left];

int temp;

for(;;)

{

if(left<right)

{

while(a[right]>tag)

right--;

//保证循环结束时 left = right

if(left>=right)

break;

temp=a[right];

a[right]=a[left];

a[left++]=temp;

}

else

break;

if(left<right)

{

while(a[left]<tag)

left++;

//保证循环结束时 left = right

if(left>=right)

break;

temp=a[left];

a[left]=a[right];

a[right--]=temp;

}

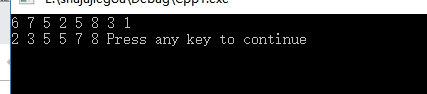
else

break;

}

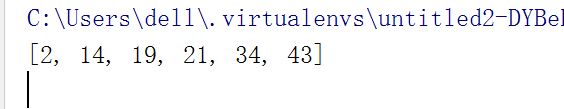
return right;

}



1. 2合并排序

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))



C语言：

#include<stdio.h>

#define MAXN 100

//A[p,q] A[q+1,r]是两个有序数组，想办法把他们结合成一个有序数组

void merge(int A[],int p,int q,int r){

int n=0;

int i=p;

int j=q+1;

int tmp[MAXN];

while(i<=q&&j<=r){

if(A[i]<=A[j])

tmp[n++]=A[i++];

else

tmp[n++]=A[j++];

}

while(i<=q)

tmp[n++]=A[i++];

while(j<=r)

tmp[n++]=A[j++];

int k;

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

A[k]=tmp[k-p];

}

//归并排序主体

void merge\_sort(int A[],int p,int r){

if(p<r){

int q=(p+r)/2;

merge\_sort(A,p,q);

merge\_sort(A,q+1,r);

merge(A,p,q,r);

}

}

int main(){

int A[]={14, 2, 34, 43, 21, 19};

merge\_sort(A,0,5);

int k;

for(k=0;k<=5;k++)

printf("%d ",A[k]);

printf("\n");

return 0;

}

1. 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。

#include<iostream>

#include<set>//用到deque容器

#include<iterator>//用到输出迭代器

#include<algorithm>//用到copy函数

#include<functional>//用到仿函数

using namespace std;

int\* GetMinK(int \*arr,int n,int k,int \*Result)//用红黑树实现

{

if(!arr || n<1 || k<1 || k>n || !Result)

{

cout<<"Invalid Input!";

return NULL;

}

multiset<int,greater<int> >mset(arr,arr+k);//用降序排序

int i=k;

multiset<int,greater<int> >::iterator iter;

for(;i<n;i++)

{

iter=mset.begin();

if(arr[i]<\*iter)//如果当前元素小于map中的最大值

{

mset.erase(iter);

mset.insert(arr[i]);

}

}

i=0;

while(!mset.empty())//将集合中的元素复制到输出数组中后删除

{

iter=mset.begin();

Result[i++]=\*iter;

mset.erase(iter);

}

return Result;

}

int main()

{

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

int n=sizeof(arr)/sizeof(int);

cout<<"原数组的元素为:";

copy(arr,arr+n,ostream\_iterator<int>(cout," "));

cout<<endl;

cout<<"请输入k值:";

int k;

while(cin>>k)

{

int \*result=new int[k];

result=GetMinK(arr,n,k,result);

if(result)

{

copy(result,result+k,ostream\_iterator<int>(cout," "));

}

cout<<endl;

delete []result;

}

return 0;

}

1. 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。

4.1#include<iostream>

#include<stdio.h>

#define N 6 //物品的个数

#define W 21 //背包容量

int B[N][W] = { 0 };

int w[6] = { 0,2,3,4,5,9 }; //物品重量

int v[6] = { 0,3,4,5,8,10 };//物品价值

void knapsack()

{

int k; //第K个物品

int C; //背包剩余重量

//填表

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

{

for (C = 1; C < W; C++)

{

if (w[k] > C) //第k件物品放不进去 此时背包的价值 = 判断完上一件物品之后背包的价值

{

B[k][C] = B[k - 1][C];

}

else

{

int value1 = B[k - 1][C - w[k]] + v[k]; //放入第k件物品后 背包总价值 = 先给这件物品留出空间，剩余的背包大小能装进的最大价值 + 这件物品的价值

int value2 = B[k - 1][C]; //不放入第k件物品 背包总价值 = 不用给这件物品留出空间，当前背包大小能装进的最大价值(就是判断完上一件物品之后背包的价值)

if (value1 > value2)

{

B[k][C] = value1;

}

else

{

B[k][C] = value2;

if (value1 < value2)printf("k=%d C=%d\n", k, C);

}

}

}

}

}

int main()

{

knapsack();

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

{

for (int j = 0; j < W; j++)

{

printf("%4d ", B[i][j]);

}

printf("\n\n");

}

system("pause");

}

1. 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。

#include<stdio.h>

#include<string.h>

char a[500],b[500];

char num[501][501]; ///记录中间结果的数组

char flag[501][501]; ///标记数组，用于标识下标的走向，构造出公共子序列

void LCS(); ///动态规划求解

void getLCS(); ///采用倒推方式求最长公共子序列

int main()

{

int i;

strcpy(a,"ABCBDAB");

strcpy(b,"BDCABA");

memset(num,0,sizeof(num));

memset(flag,0,sizeof(flag));

LCS();

printf("%d\n",num[strlen(a)][strlen(b)]);

getLCS();

return 0;

}

void LCS()

{

int i,j;

for(i=1;i<=strlen(a);i++)

{

for(j=1;j<=strlen(b);j++)

{

if(a[i-1]==b[j-1]) ///注意这里的下标是i-1与j-1

{

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

flag[i][j]=1; ///斜向下标记

}

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

{

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

flag[i][j]=2; ///向右标记

}

else

{

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

flag[i][j]=3; ///向下标记

}

}

}

}

void getLCS()

{

char res[500];

int i=strlen(a);

int j=strlen(b);

int k=0; ///用于保存结果的数组标志位

while(i>0 && j>0)

{

if(flag[i][j]==1) ///如果是斜向下标记

{

res[k]=a[i-1];

k++;

i--;

j--;

}

else if(flag[i][j]==2) ///如果是斜向右标记

j--;

else if(flag[i][j]==3) ///如果是斜向下标记

i--;

}

for(i=k-1;i>=0;i--)

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

}

1. 背包问题的贪心算法

#include<stdio.h>

float Knapsack(float \*w,float \*v,float \*x,float c,int n)

{

int i,j,l;

float temp,sum=0;

float \*a=new float[n];

for(i=0;i<n;i++)a[i]=v[i]/w[i];

printf("\n");

printf("以下分别是物品的重量、价值、价值/重量: \n");

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

{

printf("%f\t%f\t%f\t",w[i],v[i],a[i]);

printf("\n");

}

for(i=0;i<n-1;i++)

for(j=0;j<n-1-i;j++)

{

if(a[j]>a[j+1])

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

temp=w[j];

w[j]=w[j+1];

w[j+1]=temp;

temp=v[j];

v[j]=v[j+1];

v[j+1]=temp;

}

}

printf("\n");

printf("以下根据价值/重量从低到高排序的w[] ,v[] ,a[] : \n");

for(i=0;i<n;i++)printf("%f %f %f\n",w[i],v[i],a[i]);

printf("\n");

for(l=n-1;l>=0;l--)

{

if(w[l]<=c)

{

x[l]=1;

sum=sum+v[l];

c=c-w[l];

}

else break;

}

x[l]=c/w[l];

sum=sum+x[l]\*v[l];

delete []a;

printf("排序后的物品分别装入入背包多少:\n");

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

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

printf("\n\n");

printf("背包装东西后最大总价值为：%f\n",sum);

printf("\n");

return sum;

}

void main()

{

int n,i;

float c;

printf("背包的重量为: ");

scanf("%f",&c);

printf("\n");

printf("一共有n个物品，n的值为: ");

scanf("%d",&n);

printf("\n");

float \*w=new float[n];

float \*v=new float[n];

float \*x=new float[n];

printf("请分别输入物品的重量、价值(列如物品一:20 60(换行输入物品二)):\n");

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

{

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

}

Knapsack(w,v,x,c,n);

delete []w;

delete []v;

delete []x;

}

1. 用回溯法或者分支界限法，实现：用1-9其中的六个自然数组成一个三角形，让这个三角形的边值之和相等。

#include<stdio.h>

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++)

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

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

{

printf("%d+%d+%d ",a,b,c);

printf("%d+%d+%d ",c,d,e);

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

}

}

1. 利用回溯法或者分支界限法，求:从1-9中的自然数组成如下等式:并且数字不能重复#include "stdio.h"

void 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];

m2=a[5]\*10+a[6];

m3=a[8]\*10+a[9];

if(a[1]\*m2\*m3+a[4]\*m1\*m3==a[7]\*m1\*m2){

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);

}