算法实训作业

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）

#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, 3, 4, 6, 7, 8, 10, 13, 14};

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");

}

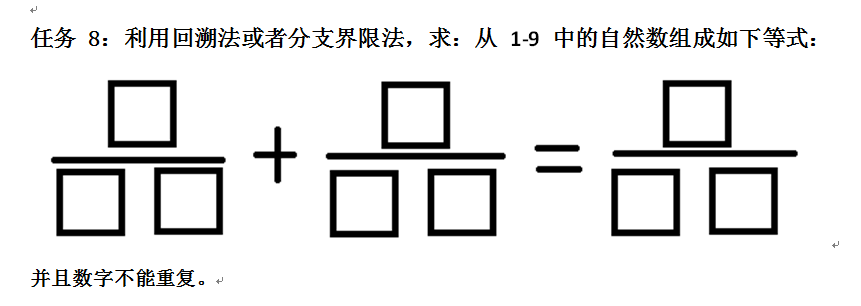
else

{

printf("找到了！下标为：%d\n", i);

}

}

2

代码：/\*

将1-9九个数不重复地赋给不同的9个元素 ，实现形如a/bc+d/ef=f/hi 的形式：

例：1/26+5/78=4/39 1/32+5/96=7/84 （注意：1/26+5/78=4/39 和5/78+1/26=4/39 只能算一种解）

求满足条件的解共有多少个？

\*/

#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);

}

3、用回溯法或分支界限法，实现：用1-9其中6个自然数组成三角形，让这个三角形的边值之和相等（语言不限）:

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++)

                {

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

                                {

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

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

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

                                }

                            }

                        }

                    }

                }

            }

        }

    }

    system("pause");

    return 0;

}

4、教授的目的是最小化横穿途中补充水的次数。设计一个高效的方法。以帮助教授确定应该在哪些补充水，证明你的策略会生成最优解，分析其运行时间：

代码如下：

/\*取水问题贪心算法-迭代写法\*/

GetWater(m,S[num],L)

let B[] be a array ,and B[0]=0;

j=0;

for i= 0~num-1;

if(S[i]-B[j]>m)

j++;

B[j] = S[i-1];

if (L-S[]<=m)

B[++j] = S[i];

return B[];

5、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:

代码：

#include<iostream>

#include<stdio.h>

#include<stdlib.h>

using namespace std;

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

void Merge(int c[], int d[], int n, int m, int r);

void copy(int a[], int b[], int l, int r);

//合并c[n:m]和c[m+1:r]到d[n:r]

void Merge(int c[], int d[], int n, int m, int r)

{

int i = n, j = m + 1, k = n;

while ((i <= m) && (j <= r))

{

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

d[k++] = c[i++];

else

d[k++] = c[j++];

}

if (i > m)

{

for (int q = j; q <= r; q++)

d[k++] = c[q];

}

else

{

for (int q = i; q <= m; q++)

d[k++] = c[q];

}

}

void copy(int a[], int b[], int l, int r)

{

for (int i = l; i <= r; i++)

{

a[i] = b[i];

}

}

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

{

int b[200];

if (left < right)

{

int i = (left + right) / 2;

MergeSort(a, left, i);

MergeSort(a, i + 1, right);

Merge(a, b, left, i, right);//合并到数组b

copy(a, b, left, right);//复制回数组a

}

}

void quick\_sort(int a[], int start, int end) {

if (start < end) {

int s = a[start];

int i = start;

int j = end;

//注意刚开始排序一定得从右向左查找才可以

while (i < j) {

//从右向左查找第一个小于s的值与a[0]交换

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

j--;

if (i<j)

a[i++] = a[j];//把a[j]的值给a[i]，并让i向后走一位

//从左向右查找第一个大于等于s的值与a[0]交换

while (i < j && a[i] < s)

i++;

if (i < j) {

a[j--] = a[i];//把a[i]的值给此时s所在的地址，并让j向前退一位

}

}

a[i] = s;//因为在之前的交换值过程中a[i]的值已经不是最初的值，所以要变回来

quick\_sort(a, start, i - 1);//排序枢轴前数列

quick\_sort(a, i + 1, end);//排序枢轴后数列

}

}

void put(int a[],int length) {

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

cout << a[i]<<' ';

}

}

int main()

{

int array[14] = { 1,4,2,3,9,11,10,7,95,64,100,151,121,113 };

cout << "排序前：";

put(array, 14);

quick\_sort(array, 0, 13);

cout << "\n排序后：";

put(array, 14);

cout << "\n初始时，array里的元素顺序是：" << endl;

for (int i = 0; i < 14; i++)

{

cout << array[i] << "，";

}

cout << endl;

MergeSort(array, 0, 13);

cout << "排序后，array里的元素顺序是：" << endl;

for (int i = 0; i < 14; i++)

{

cout << array[i] << "，";

}

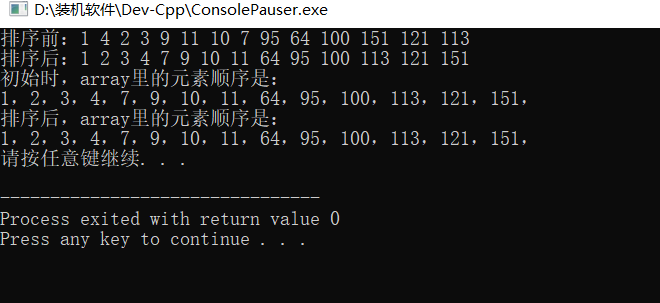
cout << endl;

system("pause");

return 0;

}

结果截图：



6、设计算法从给定的n个数找出最小的k个数，。采用递归调用实现算法：

代码如下：

#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;

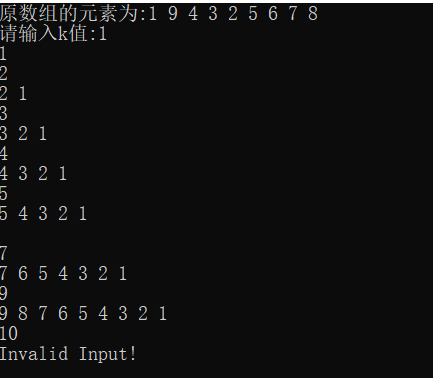
delete []result;

}

return 0;

}

结果截图：



7、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：

代码：#include<iostream>

#include<stdio.h>

#include<stdlib.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]);

}

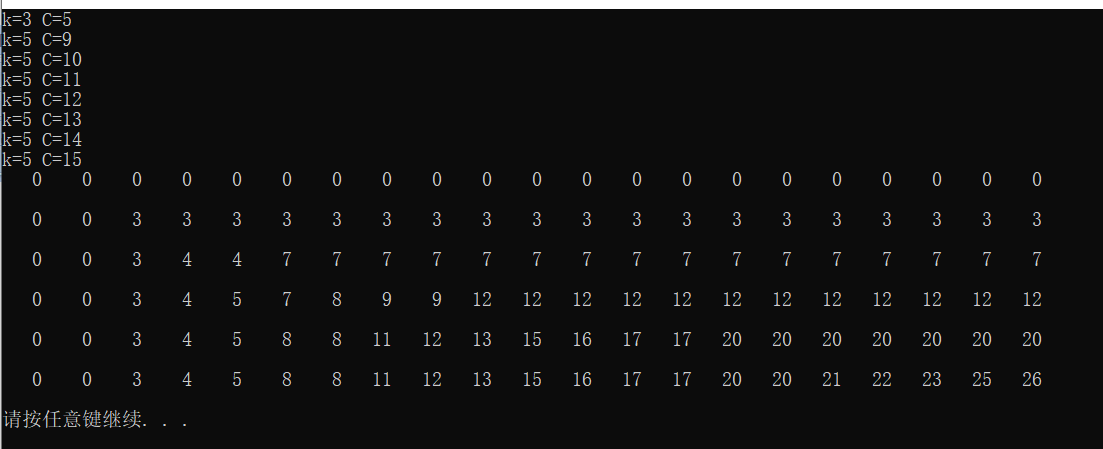
printf("\n\n");

}

system("pause");

}

结果截图：



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

int \*\*Lcs\_length(string X,string Y,int \*\*B)

{

int x\_len = X.length();

int y\_len = Y.length();

int \*\*C = new int \*[x\_len+1];

for (int i = 0; i <= x\_len; i++)

{

C[i] = new int[y\_len + 1]; //定义一个存放最优解的值的表；

}

for (int i = 1; i <= x\_len; i++)

{

C[i][0] = 0;

B[i][0] = -2; //-2表示没有方向

}

for (int j = 0; j <= y\_len; j++)

{

C[0][j] = 0;

B[0][j] = -2;

}

for (int i = 1; i <= x\_len; i++)

{

for (int j = 1; j <= y\_len; j++)

{

if (X[i-1]==Y[j-1])

{

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

B[i][j] = 0; //0表示斜向左上

}

else

{

if (C[i-1][j]>=C[i][j-1])

{

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

B[i][j] = -1; //-1表示竖直向上；

}

else

{

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

B[i][j] = 1; //1表示横向左

}

}

}

}

return C;

9、背包问题的贪心算法：

代码如下：

#include <iostream>

using namespace std;

//按照单位重量的价值量大小降序排列

void Sort(int n,float \*w,float \*v)

{

int i,j;

float temp1,temp2;

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

for(j=1;j<=n-i;j++)//冒泡排序

{

temp1=v[j]/w[j];

temp2=v[j+1]/w[j+1];

if(temp1<temp2)

{

swap(w[j],w[j+1]);

swap(v[j],v[j+1]);

}

}

}

int main()

{

float w[101];//用来表示每个物品的重量

float v[101];//用来表示每个物品的价值量

float x[101];//表示最后放入背包的比例

int n;//物品数

float M;//背包最大容纳重量

cin>>n>>M;

//依次输入每件物品的重量和价值量

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

cin>>w[i]>>v[i];

//按照单位重量的价值量大小降序排列

Sort(n,w,v);

int i;

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

x[i]=0;//初始值，未装入背包，x[i]=0

float c=M;//更新背包容纳量

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

{

if(c<w[i]) break;//不能完全装下

x[i]=1;

c=c-w[i];

}

if(i<=n)

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

//输出

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

cout<<"重量为"<<w[i]<<"价值量为"<<v[i]<<"的物品"<<"放入的比例为"<<x[i]<<endl;

return 0;

}