斐波那契数列： F[n] = F[n-1] + F[n-2] (n>=0), F[0] = 1, F[1] = 1;

斐波那契数列相关概念：黄金比例又称黄金分割，是指事物各部分间一定的数学比例关系，即将整体一分为二，较大部分与较小部分之比等于整体与较大部分之比，

其比值约为1:0.618或1.618:1。0.618被公认为最具有审美意义的比例数字，这个数值的作用不仅仅体现在诸如绘画、雕塑、音乐、建筑等艺术领域，而且在管理、工程

设计等方面也有着不可忽视的作用。因此被称为黄金分割。斐波那契数列：1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89…….（从第三个数开始，后边每一个数都是前两个

数的和）。然后我们会发现，随着斐波那契数列的递增，前后两个数的比值会越来越接近0.618

斐波那契搜索（Fibonacci search）：又称斐波那契查找，是“区间中单峰函数”的搜索技术。

斐波那契搜索是在“二分查找”的基础上根据“斐波那契数列”进行分割的（即也属于对于“有序”数列进行查找！！！）。

斐波那契搜索的时间复杂度： O(n) = log2n,且期望复杂度也为log2n

斐波那契搜索与二分查找的对比：

斐波那契查找的优点是“它只涉及加法和减法运算，而不用除法“，而除法比加减法要占用的时间更多，因此，斐波那契查找的运行时间”理论上比这般查找小“，但是

还是得视具体情况而定。

已经被证明：斐波那契搜索是一种“函数估值次数最少的最优”搜索算法！！！！！

斐波那契搜索思想：在斐波那契数列找到第一个“大于等于”表中元素个数的数F[n]，将原来的查找表“扩展为长度为F[n]（如果要补元素，则补充“重复最后一个元素”，

直到满足F[n]个元素为止）”，完成后进行“斐波那契分割”，即F[n]个元素分割为前半部分F[n-1]个元素，后半部分F[n-2]个元素，找出要查找的元素在哪一部分并递归，直到

找到。

斐波那契搜索实现：

设查找的数组array的大小为m， 查询的值为value

1. 先构建一个”数组元素符合斐波那契数列要求的且第一个大于等于要搜索的数组的大小“的数组F[n]即F[n]-1>=m, F[n]为斐波那契数列

2. 将要查找的数组大小为m的数组array扩展成数组大小为F[n]-1的数组temp。

3. 将array中的值复制给temp，当temp的数组大小大于array时，temp中超出的元素全部复制为array[m-1]

4. 设k=n, low = 0, high = n-1, middle = low + F[n-1]-1

5. 如果temp[middle] == value，则找到需要找的值

6. 如果temp[middle] > value, 则high = middle-1, n = n-1, middle = low + F[n-1]-1

7. 如果 temp[middle] < value, 则low = middle +1，n = n-2, middle = low + F[n-1]-1

8. 重复5、6、7，直到找到或者low>high。

代码:

/\*

Desc: 构造斐波那契数列数组

Param: size 数组长度

\*/

template<typename T>

void CreateFobonacci(T array[], int size)

{

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

{

if(i<= 1)

{

array[i] = 1;

}

else

{

array[i] = array[i-1] + array[i-2];

}

}

}

/\*

Des: 斐波那契搜索

Param: array 查找数组

Param: size 数组大小

Param: value 需要查询的值

\*/

template<typename T>

int FobonacciSearch(T array[], int size, T value)

{

//构造一个斐波那契数列

const int foboSize = 100;

T Fobo[foboSize] = {0};

CreateFobonacci(Fobo,foboSize);

//查找Fobo中第一个大于等于size的值的位置

int n=0;

while(Fobo[n]-1 < size)

{

n++;

}

//将数组array扩展成Fobo[n]-1大小的temp

T\* temp = new T[Fobo[n]-1];

//将array的值赋给temp

memcpy(temp, array, size\*sizeof(T));

//如果temp的长度大于array， 则temp大于array的的长度为下标的值设为array[size-1]

for(int i=size; i<Fobo[n]-1; i++)

{

temp[i] = array[size-1];

}

int low = 0;

int high = size-1;

int middle = low + Fobo[n-1]-1;

while(low <= high)

{

if(temp[middle] < value)

{

n = n-2;

low = middle + 1;

}

else if(temp[middle] > value)

{

n = n-1;

high = middle - 1;

}

else

{

//若相等则说明mid即为查找到的位置

delete [] temp;

temp = NULL;

if(middle >= size)

{

//若mid>=size则说明是扩展的数值,返回size-1

return size - 1;

}

else

{

return middle;

}

}

middle = low + Fobo[n-1]-1;

}

delete [] temp;

temp = NULL;

return -1;

}

//冒泡排序

template<typename T>

void BoubleSort(T array[], int size)

{

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

{

for(int j=0; j<size-i; j++)

{

if(j+1 >= size)

break;

T temp = array[j];

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

{

array[j] = array[j+1];

array[j+1] = temp;

}

}

}

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

int a[100] = {};

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

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

{

a[i] = rand()%200;

}

BoubleSort(a, size);

cout<<"排序前： "<< endl;

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

{

cout<< a[i]<< " ";

}

char needSearch = 'y';

while(needSearch == 'y')

{

int value = 0;

cout<<endl<<"请输入你要查询的数字： ";

cin>>value;

int index = FobonacciSearch(a, size, value);

cout<<endl<<"查询结果： "<<index<<endl;

cout<<"是否需要继续查询（y/n）： ";

cin>>needSearch;

}

return 0;

}

实现结果:

