|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 宜睿琛 | 成  绩 |  |
| 学号 | 120L031815 |

第二章 作业1

【题目】

（1）从时间、空间及操作等层面比较线性表静态存储和动态存储两种存储结构的优缺点。

（2）输入一个已经按升序排序过的数组A和一个数字*x*，在数组A中查找两个数，使得它们的和正好是输入的*x*。

（3）将一个有序数组以一个点为中心进行旋转（前后颠倒）；例如{4，5 ，1，2 ，3}是{1,2,3,4,5}以3为中心的旋转。设计算法，查找一给定的元素x。

（4）已知线性表中的元素以值递增有序排列，并以单链表作存储结构。试写一算法，删除表中所有大于*x*且小于*y*的元素（若表中存在这样的元素）同时释放被删除结点空间。

【答题】（宋体小四号，单倍行距）

（1）①时间层面：静态存储在对元素进行插入，删除操作时较耗时间，对元素进行访问时较为快捷；而动态存储在访问元素时较耗时间，对元素进行插入删除操作时省时间。例如，在数组中删除或删除一个元素时，平均要移动数组中一半的元素，但访问一个元素时只需根据下标直接访问；动态存储例如链表在访问元素时，需要通过一个一个链进行移动。

②空间层面：静态存储，如数组，在初始化的时候需要估算表的长度来进行创建，在实际操作中，为了防止空间不够用，往往需要开一个长度非常大的数组，很容易造成空间的浪费；动态存储则是根据实际的长度来占用空间，需要多少就创建多少，更能节省空间。

③操作层面：在存取操作上，静态存储每一个数据元素的存储位置都和起始位置差一个和数据元素位序成正比的常数，因此只要确定了其起始位置，存储中任一元素在进行存取操作时都可以随机存取；在动态存储中，任意两个元素的存储位置都没有固定的关系，因此在进行存取操作时，对于第i个位置的数据元素，只能采用顺序存取的方式进行存取。

（2）C++函数代码呈现

void SANA(int b[],int lgh,int x) /\*b为数组，lgh该数组的长度，x为给定的数字\*/

{

int i=0; /\*头部初始为数组第一个元素\*/

int j=lgh-1;

while(i<j) /\*进入查找循环，结束条件：头部等于尾部\*/

{

if(b[i]+b[j]<x)

{

i++;

}

else if(b[i]+b[j]>x)

{

j--;

}

else

{

cout<<b[i]<<"+"<<b[j]<<"= "<<x<<endl; /\*输出所得到的结果表达式\*/

return

}

}

cout<<"失败"<<endl /\*若没找到则输出“失败”\*/

return

}

（3）

数据结构定义：（P,R）

其中：P={ ,···} 输入的数组

R={R1 , R2}

R1={<>,1≤i≤x, x+1≪i≪n-1} 数组a从p处被分为两个有序的部分

R2={<,>} 数组a在p,p+1处的关系与其他处相反

算法设计：对于有序数列，查找元素最快捷的方式是二分法，此题依据此设计算法。首先找到旋转点p,在p点两侧为两个有序数列，于是便可分别对这两部分进行二分查找。

关键函数代码实现(在此不妨设该数列为递增数列，若要处理递减情况，只需改一下不等号朝向即可)：

int findp(int a[],int lgh) /\*定义函数查找旋转点p\*/

{

int i;

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

{

if(a[i+1]<a[i])

return i

}

return -1 /\*若没查找到p的位置则返回-1\*/

}

int erfenfind(int a[],int lgh,int x) /\*二分法查找元素x\*/

{

int i=-1, j= lgh;

while(i+1 != j){ /\*结束条件：头尾相接\*/

int m = (l + j) / 2;

if(x > a[m])

{

i = m;

}else{

j = m;

}

}

int p = j;

if(p >= lgh || a[p] != x){

return -1;

}

return j;

}

主函数的代码实现只需要读入数组，使用findp函数查找旋转点p(如果返回的是-1，则数组完整有序，直接对完整数组进行二分查找即可)，分别对p点左右两侧使用erfenfind函数进行二分查找即可。

（4）