**周报**

本周学习了《零基础学算法》的第四、第五章。第四章学习的是常用的算法排序，第五章学习查找。具体的知识点总结如下：

1. 常用算法——排序
2. 分类：内部排序、外部排序
3. 内部排序：整个过程不需要访问外存便能完成。

内部排序：

* 1. 交换排序：冒泡排序、快速排序
  2. 选择排序：直接选择排序、堆排序
  3. 插入排序：直接插入排序、希尔排序
  4. 合并排序

1. 外部排序
2. 冒泡排序

一般冒泡排序

1. 假设从大到小排序，则从左到右，先比较第一个与第二个数，如果第一个数比第二个数小则交换位置，如果第二个数比第一个数小则位置不变。
2. 接着第二个数与第三个数比较以此类推。则比较到最后一个就可以得到最小的数。
3. 重复以上的步骤，就可以完成排序。时间复杂度是lg(n2)。

改进冒泡排序：设置一个标志位当全部排序之后，就不要进行后续的排序操作。

1. 快速排序：（是一个递归过程）
2. 首先选择一个基准数，两个指针一个指向最后一个数，一个指向最前面的数，第一次以左边的第一个数为基准数
3. 一个指针先从最后一个开始遍历，找到比基准数小的数，然后指针指向这个数；
4. 再从左边第一个数，也就是基准数开始从左到右开始遍历，找到与这个基准数大的数，然后指针指向这个数。
5. 把这两个数进行对换，如果指针一直遍历到两个指针都指向同一个元素的时候，就把这个数与基准数进行对换。
6. 然后以这个基准数为基准，分为两边，左边的数小于这个基准数，右边的数大于这个基准数，这样就找到了这个基准数的位置。第一轮遍历完毕。
7. 把以上的两边单独重复一轮的遍历完成排序。
8. 简单选择排序法：对N个记录进行扫描，选择最小的记录将其输出，接着在剩下的N-1个记录中扫描，选择最小记录将其输出，不断重复这个过程，就可完成排序。
9. 分类：（选择排序的分类）简单选择排序、堆排序
10. 简单选择排序思想：（不是递归实现）

从数据中选择最小（或最大）的数据，然后将其与第一个数交换，接下来从剩下数据中选择最小（或最大）的数据与第二个数据交换，这样一直下去就可完成排序。

1. 选择排序——堆排序

堆是一个完全二叉树，树中每个节点对应于原始数据的一个记录，并且每个节点应满足一下条件：非叶节点大于等于其左右孩子节点的数据。

1. 堆排序过程包括一下两个阶段：
   1. 将无序的数据构成堆。
   2. 利用堆排序

（2） 构成堆：把无序的数据按堆定义进行交换，使父节点的数据大于子节点的数据。具体步骤如下：（构成堆以后根节点的数据最大）

① 将无序数据放入二叉树的各个节点；

② 由二叉树的下层逐层进行父节点的数据比较，使用一种称为“筛”的运算进行节点数据的调整，知道节点最后满足条件为止。

代码：构造堆函数

while(2\*s+1<n) { //第s个结点有右子树

j=2\*s+1 ;

if((j+1)<n) {

if(a[j]<a[j+1])//右左子树小于右子树，则需要比较右子树

j++; //序号增加1，指向右子树

}

if(a[s]<a[j]) { //比较s与j为序号的数据

t=a[s]; //交换数据

a[s]=a[j];

a[j]=t;

s=j ;//堆被破坏，需要重新调整

}

else //比较左右孩子均大则堆未破坏，不再需要调整

break;

}

1. 利用堆排序：（实现的过程）
   1. 去根节点（最大值），将其放在数组的最后；
   2. 重新执行面前的构成堆的方法；
   3. 重复执行上面过程，逐步从根节点去除最大值。
2. 直接插入排序法

描述：

1. 对于第一个元素，因为没有比较将其作为一经有序的序列；
2. 从数组的下一个元素，在已经排序的元素序列中从后向前扫描，并判断元素与已排序的大小；
3. 若排序序列的元素大于新元素，则将该元素移到下一个位置；
4. 重复步骤3，知道找到已排序的元素小于或者等于新元素的位置；
5. 将新元素插入到该位置；
6. 重复步骤2-5，直到将数组中的数据处理完。
7. 希尔（shell）排序法（插入算法）
8. 将需要排序的序列分为若干个较小的序列，对这些排序进行直接插入排序，通过这用的操作可使需要排序的数列基本有序，最后再使用一次直接插入排序对整个数列进行排序。这样，首先对数量较小的序列进行直接排序，也可提高效率。从而使整个排序过程的效率得到提升。
9. 合并排序：

将两个或多个有序表合成一个有序表。

1. 合并排序过程
   1. 经过一遍合并得到长度为2的有序表；
   2. 再将长度为2的有序表进行两两合并，得到长度为4的有序表；
   3. 
2. 合并相邻有序表：
   * + - 1. 取第一个系列的地i个元素A[i]，与第二个系列的第一个元素A[j],比较。
         2. 若A[i]小于A[j]，则将A[i]复制到R[k]中使i和K分别增加1；
         3. 与b相反，则将A[j]复制到R[k]中，使j和k分别增加1；
         4. 重复第一步至第三步，知道一个序列复制完毕位置；
         5. 将另一个序列中未比较的数据复制到R的剩余位置。
3. 完成一遍完整合并。
4. 合并排序
5. 常用算法——查找
6. 静态查找和动态查找
7. 静态查找：数据结构不改变查找表
8. 动态查找表：数据结构岁查找过程进行改变查找表；
9. 简单查找
10. 顺序查找：从线性表的一段开始，依次将每个记录的关键字与给定值进行比较。

For（i=0;i<n;&&s[i]==key;i++）;

1. 顺序表查找改进方法：在创建静态查找表时候，在该表的末端真假一个空的单元，用来保存查找的关键之。
2. 折半查找
3. 条件：查找表的数据时线性结构保存，并且还要求查找表中的数据时按关键字由小到大有序排列。
4. 具体过程：假设有N个元素的查找表，首先计算位于查找表中间位置。序号m（m=n/2）,取S[m]的关键字与给定的值key比较，有三种情况：
   * + 1. 若S[m]=key,查找成功；
       2. 若S[m]<key，则在后半部分继续进行查找；
       3. S[m] > key，则表示关键字key可能在查找的前半部分，则在前半部分进行查找。
5. 代码：

mid=(low+high)/2; //计算中间位置序号

if(s[mid]==key) //中间元素与关键字相等

return mid; //返回序号

else if(s[mid]>key) //中间元素大于关键字

high=mid-1; //重定义查找范围

else //中间元素小于关键字

low=mid+1; //重定义查找范围

1. 二叉排序树
2. 树形查找（动态查找表）：是对树形存储结构中的元素进行查找操作
3. 二叉排序树的定义:
   1. 左子树上的节点数据均小于根节点
   2. 右节点上所有节点的数据均大于根节点的数据；
   3. 左右子树本身又是一颗二叉排序树。
4. 二叉排序树插入节点（假设t为树根指针）
   1. 申请保存节点p的内存，将关键字保存在节点数据域；
   2. 在二叉排序中查找插入节点p的父节点；
   3. 比较P节点数据与父节点的数据，决定将p插入父节点的左子树还是右子树
5. 插入节点的操作：插入一个节点、插入多个节点、中序遍历排序二叉树，得到的结果是从小到大。
6. 查找节点：
   1. 从根节点开始查找；
   2. 如节点为空查找失败；
   3. 若key值与节点的数据相等，表示查找节点成功，返回节点指针；
   4. 若key值小于节点的数据，继续在当前节点的左子树查找；
   5. 若key值大于节点的数据，继续在当前节点的右子树查找。
7. 索引查找
8. 创建所以的一般步骤：将一个线性表（主表）安一低昂的函数关或条件划分为若干个子表，然后为把每个子表创建一个引索项，把这些引索组合在一起构成主表的一个索引表。接着，采用线性表或者链接表方式分别爆粗索引和个子表。
9. 在索引表中进行查找：
   1. 根据给定的关键字key，按定义的函数计算出索引值index1，在索引表上查找出索引值等一index1的索引项，以确定对应子表在主表的开始位置和长度；
   2. 根据从索引中获取的开始徐璈start，在主表制定位置（即子表的开始处）顺序查找关键字key;
10. 在索引表中的主要代码：

start=indextable[i].start; //获取数组开始序号

length=indextable[i].length; //获取元素长度

1. 向主表中插入数据（具体过程）：
   1. 根据待插入数据的值查找索引表，确定对应的子表；
   2. 根据待插入元素的关键字，在该子表中做插入元素的操作；
   3. 插入完成后，修改引索表中的相应子表子表的长度。

关键代码: stu[start+length]=key;//保存关键字到主表

indextable[i].length++;//修改索引表中的子表长度

1. 散列表：

将关键字与保存位置之间形成一种函数对应关系的函数称为散函数，按这种方式建立的表称为散列表。

1. 基本思想：一线性表中每个元素的关键字key为自变量，通过一定的函数关系h(key)，计算出函数的值，把这个值作为数组的下标，将元素存入定影数组元素中，函数h（key）称为散列函数，函数的值称为散列地址。
2. 构造散列函数的方法：
   1. 直接定址法：h(key) = key +c
   2. 除法取余法：除法取余法就是用关键字key初一散列表长度N，得到的余数作为散列地址；
   3. 数字分析法：取关键字中某些比较分散的数字作为散列地址的方法。这种方法适用于所有关键字已知。
   4. 平方取中发：将关键字key求平方后，取中间的几位作为散列地址的方法。
   5. 折叠法：将关键字key按散列地址要求的位数分成长度相等的几段，最后一段的长度可能回短些，接着将几段进行求和，然后去掉最高位的进制，将得到的值作为散列地址。
3. 冲突处理：
   * 1. 开放地址法：线性探测法、爽散函数探测法
     2. 链接法：每个存储单元中增设一个指针域，将散列地址相同的元素链接起来。
4. 创建和查找散列表：
   1. 用线性探测法解决冲突：

void InsertHash(int hash[],int m,int data) //将关键字data插入哈希表hash中

{

int i;

i=data % 13;//计算哈希地址

while(hash[i]) //元素位置已被占用

i=(++i) % m; //线性探测法解决冲突

hash[i]=data;

}

**2016-10-10**

**系统控制与信息处理重点实验室**