****

**算法分析与设计实验报告**

****

**学 院： 电子信息与人工智能学院**

**专业名称： 计算机科学与技术**

**班 级： 计算机203**

**学 号： 202007020625**

**姓 名： 於俊涛**

**任课老师： 齐 勇**

**实验2 查找算法性能分析实验**

班级 计算机203 学号 202007020625 姓名 於俊涛 角色 reader

**一、实验目的**

1、掌握7大查找算法的原理。

2、分析不同算法的性能及适用环境。

**二、实验环境**

线上平台sustoj.com

**三、实验内容**

**（一）独立算法性能分析**

**1、顺序查找**

说明：顺序查找适用于存储结构为顺序存储或链接存储的线性表

基本思想：顺序查找也称为线形查找，属于无序查找算法。从数据结构线形表的一端开始，顺序扫描，依次将扫描到的结点关键字与给定值k相比较，若相等则表示查找成功；若扫描结束仍没有找到关键字等于k的结点，表示查找失败。

复杂度分析：

　　查找成功时的平均查找长度为：（假设每个数据元素的概率相等）

ASL = 1/n(1+2+3+…+n) = (n+1)/2 ;  
　　当查找不成功时，需要n+1次比较，时间复杂度为O(n);

　　所以，**顺序查找的时间复杂度为O(n)。**

核心代码：

int SequenceSearch(int a[], int value, int n)

{

int i;

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

if(a[i]==value)

return i;

return -1;

}

**2、二分查找**

说明：元素必须是有序的，如果是无序的则要先进行排序操作。

基本思想：也称为是折半查找，属于有序查找算法。用给定值k先与中间结点的关键字比较，中间结点把线形表分成两个子表，若相等则查找成功；若不相等，再根据k与该中间结点关键字的比较结果确定下一步查找哪个子表，这样递归进行，直到查找到或查找结束发现表中没有这样的结点。

复杂度分析：

最坏情况下，关键词比较次数为log2(n+1)，且**期望时间复杂度为O(log2n)**

核心代码：

//二分查找（折半查找）

int BinarySearch1(int a[], int value, int n)

{

int low, high, mid;

low = 0;

high = n-1;

while(low<=high)

{

mid = (low+high)/2;

if(a[mid]==value)

return mid;

if(a[mid]>value)

high = mid-1;

if(a[mid]<value)

low = mid+1;

}

return -1;

}

**3、插值查找**

基本思想：基于二分查找算法，将查找点的选择改进为自适应选择，可以提高查找效率。当然，差值查找也属于有序查找。

**复杂度分析：查找成功或者失败的时间复杂度均为O(log2(log2n))。**

核心代码：

//插值查找

int InsertionSearch(int a[], int value, int low, int high)

{

int mid = low+(value-a[low])/(a[high]-a[low])\*(high-low);

if(a[mid]==value)

return mid;

if(a[mid]>value)

return InsertionSearch(a, value, low, mid-1);

if(a[mid]<value)

return InsertionSearch(a, value, mid+1, high);

}

**4、斐波那契查找**

也是二分查找的一种提升算法，通过运用黄金比例的概念在数列中选择查找点进行查找，提高查找效率。同样地，斐波那契查找也属于一种有序查找算法。

　斐波那契查找与折半查找很相似，他是根据斐波那契序列的特点对有序表进行分割的。他要求开始表中记录的个数为某个斐波那契数小1，及n=F(k)-1;

 开始将k值与第F(k-1)位置的记录进行比较(及mid=low+F(k-1)-1),比较结果也分为三种

　　1）相等，mid位置的元素即为所求

　　2）>，low=mid+1,k-=2;

说明：low=mid+1说明待查找的元素在[mid+1,high]范围内，k-=2 说明范围[mid+1,high]内的元素个数为n-(F(k-1))= Fk-1-F(k-1)=Fk-F(k-1)-1=F(k-2)-1个，所以可以递归的应用斐波那契查找。

　3）<，high=mid-1,k-=1。

　说明：low=mid+1说明待查找的元素在[low,mid-1]范围内，k-=1 说明范围[low,mid-1]内的元素个数为F(k-1)-1个，所以可以递归 的应用斐波那契查找。

**复杂度分析：最坏情况下，时间复杂度为O(log2n)，且其期望复杂度也为O(log2n)。**

**5、树表查找**

**5.1 最简单的树表查找算法——二叉树查找算法。**

**基本思想：**二叉查找树是先对待查找的数据进行生成树，确保树的左分支的值小于右分支的值，然后在就行和每个节点的父节点比较大小，查找最适合的范围。这个算法的查找效率很高，但是如果使用这种查找方法要首先创建树。

**二叉查找树**（BinarySearch Tree，也叫二叉搜索树，或称二叉排序树Binary Sort Tree）或者是一棵空树，或者是具有下列性质的二叉树：

　1）若任意节点的左子树不空，则左子树上所有结点的值均小于它的根结点的值；

　2）若任意节点的右子树不空，则右子树上所有结点的值均大于它的根结点的值；

　3）任意节点的左、右子树也分别为二叉查找树。

**二叉查找树性质**：**对二叉查找树进行中序遍历，即可得到有序的数列。**

**复杂度分析：它和二分查找一样，插入和查找的时间复杂度均为O(logn)，但是在最坏的情况下仍然会有O(n)的时间复杂度。原因在于插入和删除元素的时候，树没有保持平衡。我们追求的是在最坏的情况下仍然有较好的时间复杂度，这就是平衡查找树设计的初衷。**

**5.2 平衡查找树之红黑树（Red-Black Tree）**

**基本思想：**红黑树的思想就是对2-3查找树进行编码，尤其是对2-3查找树中的3-nodes节点添加额外的信息。红黑树中将节点之间的链接分为两种不同类型，红色链接，他用来链接两个2-nodes节点来表示一个3-nodes节点。黑色链接用来链接普通的2-3节点。特别的，使用红色链接的两个2-nodes来表示一个3-nodes节点，并且向左倾斜，即一个2-node是另一个2-node的左子节点。这种做法的好处是查找的时候不用做任何修改，和普通的二叉查找树相同。

红黑树的定义：

红黑树是一种具有红色和黑色链接的平衡查找树，同时满足：

①红色节点向左倾斜

②一个节点不可能有两个红色链接

③整个树完全黑色平衡，即从根节点到所以叶子结点的路径上，黑色链接的个数都相同。

5.3 B树和B+树（B Tree/B+ Tree）

B树（B-tree）是一种树状数据结构，它能够存储数据、对其进行排序并允许以O(log n)的时间复杂度运行进行查找、顺序读取、插入和删除的数据结构。B树，概括来说是一个节点可以拥有多于2个子节点的二叉查找树。与自平衡二叉查找树不同，B树为系统最优化大块数据的读和写操作。B-tree算法减少定位记录时所经历的中间过程，从而加快存取速度。普遍运用在**数据库**和**文件系统**。

B和B+树的区别在于，B+树的非叶子结点只包含导航信息，不包含实际的值，所有的叶子结点和相连的节点使用链表相连，便于区间查找和遍历。

　B+ 树的优点在于：

* 由于B+树在内部节点上不好含数据信息，因此在内存页中能够存放更多的key。 数据存放的更加紧密，具有更好的空间局部性。因此访问叶子几点上关联的数据也具有更好的缓存命中率。
* B+树的叶子结点都是相链的，因此对整棵树的便利只需要一次线性遍历叶子结点即可。而且由于数据顺序排列并且相连，所以便于区间查找和搜索。而B树则需要进行每一层的递归遍历。相邻的元素可能在内存中不相邻，所以缓存命中性没有B+树好。

　但是B树也有优点，其优点在于，由于B树的每一个节点都包含key和value，因此经常访问的元素可能离根节点更近，因此访问也更迅速。

**6. 分块查找**

　分块查找又称索引顺序查找，它是顺序查找的一种改进方法。  
　　算法思想：将n个数据元素"按块有序"划分为m块（m ≤ n）。每一块中的结点不必有序，但块与块之间必须"按块有序"；即第1块中任一元素的关键字都必须小于第2块中任一元素的关键字；而第2块中任一元素又都必须小于第3块中的任一元素，……  
　　算法流程：  
　　step1 先选取各块中的最大关键字构成一个索引表；  
　　step2 查找分两个部分：先对索引表进行二分查找或顺序查找，以确定待查记录在哪一块中；然后，在已确定的块中用顺序法进行查找。

**7、哈希查找**

算法思想：哈希的思路很简单，如果所有的键都是整数，那么就可以使用一个简单的无序数组来实现：将键作为索引，值即为其对应的值，这样就可以快速访问任意键的值。这是对于简单的键的情况，我们将其扩展到可以处理更加复杂的类型的键。

算法流程：

　1）用给定的哈希函数构造哈希表；

　2）根据选择的冲突处理方法解决地址冲突；

　常见的解决冲突的方法：拉链法和线性探测法。

　3）在哈希表的基础上执行哈希查找。

　复杂度分析：

　单纯论查找复杂度：对于无冲突的Hash表而言，查找复杂度为O(1)（注意，在查找之前我们需要构建相应的Hash表）

**四、实验结论**

七大查找算法各有优缺点，在数据有序时，使用二分查找效率更高，但在现实生活中，很小几率是有序的静态查找表。因此知道不同算法在哪种场景下能有最大效率是我们需要学习的目标。

在本次实验中，我学习到了不同算法的原理，也知道了这些算法的差异与优缺点，同时通过算法对比，了解到了不同算法的运用场景。