顺序查找

最基本的查找技术，它的查找过程是：从一个表中的第一个或者最后一个记录开始，逐个进行记录的关键字和给定的值是否相等，若相等，则查找成功。

时间复杂度（o(n)）,缺点：当数组数据量很大的时候，查找的效率很低下

public static int findIndex(int[] arr,int target) {

```
        if (arr==null || arr.length==0) return -1;
        for(int i=0;i<arr.length;i++){
                if (arr[i]==target) {
                        return i;
                }
        }
        return -1;

}

```

有序表查找

斐波那契查找

二分查找（折半查找）

是一种在有序数组中查找某一特定元素的搜索算法。索过程从数组的中间元素开始，如果中间元素正好是要查找的元素，则搜索过程结束；如果某一特定元素大于或者小于中间元素，则在数组大于或小于中间元素的那一半中查找，而且跟开始一样从中间元素开始比较。如果在某一步骤数组为空，则代表找不到。这种搜索算法每一次比较都使搜索范围缩小一半 [1] 。

时间复杂度：o(logn)

public static int binarySearch2(int [] arr,int target,int low,int high) {

if (arr==null || arr.length==0) return -1;

```
int mid=low+(high-low)/2;
if (low>high) {
        return -1;
}
if (arr[mid]==target) {
        return mid;
}
if (arr[mid]<target) {
        return binarySearch2(arr, target, mid+1, high);
}else {
         return binarySearch2(arr, target, low, mid-1);
}
```

```
13
```

}

)

非递归的写法：

//非递归的写法：

public static int binarySearch(int [] arr,int target) {

if (arr==null || arr.length==0) return -1;

int low=0;

int high=arr.length-1;

int mid;

while (low<=high) {

mid=low+(high-low)/2;

if (arr[mid]==target) {

return mid;

}else if (arr[mid]>target) {

high=mid-1;

```
1           }else {
2                   low=mid+1;
3           }
4   }
5
6   return -1;
7
```

}

结论：

对于递归操作而言，如果每次递归使问题的规模减半，而其他操作都是常数时间

T(N)=T(N/2)+O(1)， 则T(N)=O(logN)

若每次递归使用问题的规模减1，而其他操作是常数时间

T(N)=T(N-1)+O(1)，则T(N)=O(N)

若每次递归使问题的规模减半，而其他操作是线性时间，T(N) = T(N/2)+O(N)

则T(N)=O(NlogN)

例题：实现 int sqrt(int x) 函数。计算并返回 x 的平方根，其中 x 是非负整数。由于返回类型是整数，结果只保留整数的部分，小数部分将被舍去。(https://leetcode-cn.com/problems/sqrtx/)

public static float sqrt(int x, double d) {

```
1           if(x==1||x==0) return (float)x;
```

```
2
3           float low=1;    //¿??n?Ï??????
4           float high=x;
5           float mid = 0;
6           while(high-low>d){
7                    mid=(low+high)/2;
8
9                   if(x/mid<mid) {
10                          high=mid;
11
12                  }else if(x/mid>mid) {
13                          low=mid;
14                  }else {
15                          return mid;
16                  }
17          }
18
19
20
21          return mid;
22
23
24  }
25
```

二叉排序树查找

二叉排序树或者是一棵空树，或者是具有如下特性的二叉树：

若左子树不空，则左子树上所有结点的值均小于或等于根结点的值；

若右子树不空，则右子树上所有结点的值均大于或等于根结点的值；

左、右子树也分别为二叉排序树

由于二叉排序树可以看成是一个有序表，所以在二叉排序树上进行查找类似于折半查找，即逐步缩小查找范围的过程。具体步骤为：

若查找的关键字等于根结点的关键字，查找成功；

若查找的关键字小于根结点的关键字，递归查找左子树；

若查找的关键字大于根结点的关键字，递归查找右子树。

若子树为空，则查找不成功。

构建二叉排序树的目的不是为了排序，而是为了提高插入删除和查找的速率。

查找的示例代码：（时间复杂度为o(logn)）

1、递归的实现：

//递归实现二叉树查找

public static boolean BinarySortedSearch1(TreeNode root,int target) {

```
        if (root==null) return false;
        if (root.val==target) {
                return true;
        }
        if (root.val>target) {
                return BinarySortedSearch1(root.left, target);

        }else {
                return BinarySortedSearch1(root.right, target);
        }


}

```

2、非递归的实现

public static boolean BinarySortedSearch2(TreeNode root,int target) {

```
                TreeNode temp=root;
                while (temp!=null) {
                        if (temp.val==target) {
                                return true;
                        }else if (temp.val>target) {
                                temp=temp.left;
                        }else {
                                temp=temp.right;
                        }
                }

                return false;

        }

```

散列表查找（时间复杂度o(1)）

在进行查找时，在记录的存储位置与它的关键字之间建立一个确定的对应关系h,以线性表中每个元素的关键字K为自变量，通过函数h(K)计算出该元素的存储位置，我们将h函数称为散列函数或哈希函数。这种查找方法称为散列查找。

散列技术的记录之间不存在什么逻辑关系，它只与关键字有关，因此散列只是面向查找的存储结构。散列技术最适合的求解问题就是查找与给定值相等的记录。

原文链接：https://blog.csdn.net/weixin_44625138/article/details/101122677