七大查找算法

没考上研究生的张同学 2018-02-26 13:57:08 22474 收藏 100

分类专栏： java 文章标签： java

找是在大量的信息中寻找一个特定的信息元素，在计算机应用中，查找是常用的基本运算，如编译程序中符号表的查找。本文简单概括性的介绍了常见的七种查找算法，说是七种，其二分查找、插值查找以及斐波那契查找都可以归为一类——插值查找。插值查找和斐波那契找是在二分查找的基础上的优化查找算法。树表查找和哈希查找会在后续的博文中进行详细绍。

查找定义：根据给定的某个值，在查找表中确定一个其关键字等于给定值的数据元素（或记）。

找算法分类：

1）静态查找和动态查找；

注：静态或者动态都是针对查找表而言的。**动态表指查找表中有删除和插入操作的**。

2）无序查找和有序查找。

无序查找：被查找数列有序无序均可；

有序查找：被查找数列必须为有序数列。

均查找长度（Average Search Length，ASL）：需和指定key进行比较的关键字的个数的望值，称为查找算法在查找成功时的平均查找长度。

对于含有n个数据元素的查找表，查找成功的平均查找长度为：ASL = Pi*Ci的和。
Pi：查找表中第i个数据元素的概率。
Ci：找到第i个数据元素时已经比较过的次数。

顺序查找

说明：顺序查找适合于存储结构为顺序存储或链接存储的线性表。

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；若扫描结束仍有找到关键字等于k的结点，表示查找失败。

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+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)。

C++实现源码：

```
int SequenceSearch(int a[], int value, int n)
{
    int i;
    for(i=0; i<n; i++)
        if(a[i]==value)
            return i;
    return -1;
}
```

ava实现源码：

```
public class sequence{
    public static  boolean SequenceSearch(int a[],int k,int value){
        for( int i = 0 ; i<k;i++){
            if( value == a[i])
                return true;
            else
                return false;
        }
        return false;
    }
    public static void main(String[] args) {
        int[] a = {8,2,4,5,3,10,11,6,9};
        System.out.println(SequenceSearch(a,a.length,20));
    }
}
//printf: false
```

```
    public static  boolean SequenceSearch(int a[],int k,int value){
            for( int i = 0 ; i<k;i++){
                    if( value == a[i])
                            return true;
                    else
                            return false;
            }
            return false;
    }
    public static void main(String[] args) {
            int[] a = {8,2,4,5,3,10,11,6,9};
            System.out.println(SequenceSearch(a,a.length,20));
    }
```

```
/printf: false
```

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交，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)；

注：折半查找的前提条件是需要**有序表顺序存储**，对于静态查找表，一次排序后不再变化，折半查找能得到不错的效率。但对于需要**频繁执行插入或删除操作**的数据集来说，维护有序的排序会带来不小的工作量，那就不建议使用。

```
//二分查找（折半查找），版本1
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;
}

//二分查找，递归版本
int BinarySearch2(int a[], int value, int low, int high)
{
    int mid = low+(high-low)/2;
    if(a[mid]==value)
        return mid;
    if(a[mid]>value)
        return BinarySearch2(a, value, low, mid-1);
    if(a[mid]<value)
        return BinarySearch2(a, value, mid+1, high);
}
```

va 实现源码：

```
/*1.*/
public class BinarySearch1{

    public static int binarysearch(int[] a,int n,int value){
        int low = 0;
        int high = n - 1;
        int mid;
        while(low < high){
            mid = (low + high)/2;
            if(value < a[mid])
                high = mid - 1;
            if(value > a[mid])
                low = mid + 1;
            if(value == a[mid])
                return mid;
        }
        return -1;
    }
    public static void main(String[] args) {
        //int[] a = {1,4,2,9,8,6,7,0,3,5}
        int[] a = {0,1,2,3,4,5,6,7,8,9};
        System.out.println(binarysearch(a,a.length,7));
    }
}
```

```
    public static int binarysearch(int[] a,int n,int value){
            int low = 0;
            int high = n - 1;
            int mid;
            while(low < high){
                    mid = (low + high)/2;
                    if(value < a[mid])
                            high = mid - 1;
```

```
                    if(value == a[mid])
                            return mid;
            }
            return -1;
    }
    public static void main(String[] args) {
            //int[] a = {1,4,2,9,8,6,7,0,3,5}
            int[] a = {0,1,2,3,4,5,6,7,8,9};
            System.out.println(binarysearch(a,a.length,7));
    }
```

```
/*2.recursive algorithm     */
public class BinarySearch2{

    public static int binarysearch(int[] a,int value,int low,int high){
            int mid = (low + high)/2;
            if(value == a[mid])
                return mid;
            mid = (low + high)/2;
            if(value < a[mid])
                return binarysearch(a,value,low,mid - 1);
            if(value > a[mid])
                return binarysearch(a,value,mid + 1,high);
            return -1;
    }
    public static void main(String[] args) {
        //int[] a = {1,4,2,9,8,6,7,0,3,5}
        int[] a = {0,1,2,3,4,5,6,7,8,9};
        System.out.println(binarysearch(a,4,0,a.length-1));
    }
}
```

插值查找

在介绍插值查找之前，首先考虑一个新问题，为什么上述算法一定要是折半，而不是折四之一或者折更多呢?

打个比方，在英文字典里面查“apple”，你下意识翻开字典是翻前面的书页还是后面的书页？如果再让你查“zoo”，你又怎么查？很显然，这里你绝对不会是从中间开始查起，而是有一目的的往前或往后翻。

同样的，比如要在取值范围1 ~ 10000 之间 100 个元素从小到大均匀分布的数组中查找我们自然会考虑从数组下标较小的开始查找。

经过以上分析，折半查找这种查找方式，不是自适应的（也就是说是傻瓜式的）。二分查中查找点计算如下：

mid=(low+high)/2, 即mid=low+1/2*(high-low);

通过类比，我们可以将查找的点改进为如下：

mid=low+(key-a[low])/(a[high]-a[low])*(high-low)，

也就是将上述的比例参数1/2改进为自适应的，根据关键字在整个有序表中所处的位置，让id值的变化更靠近关键字key，这样也就间接地减少了比较次数。

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。然，差值查找也属于有序查找。

注：对于表长较大，而关键字分布又比较均匀的查找表来说，插值查找算法的平均性能比半查找要好的多。反之，数组中如果分布非常不均匀，那么插值查找未必是很合适的选择。

复杂度分析：查找成功或者失败的时间复杂度均为O(log2(log2n))。

va代码实现：

```
    public static 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);
        return -1;
    }
    public static void main(String[] args) {
        int[] a = {0,1,2,3,4,5,6,7,8,9};
        System.out.println(InsertionSearch(a,2,0,a.length-1));
    }
}
```

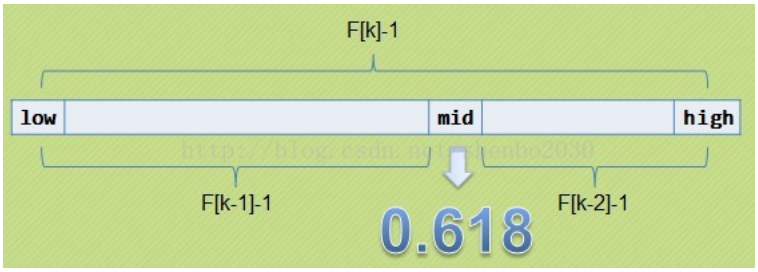
斐波那契查找

介绍斐波那契查找算法之前，我们先介绍一下很它紧密相连并且大家都熟知的一个概念——金分割。

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。

大家记不记得斐波那契数列：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，利用这个特性，我们就可以将黄金比例运用到查找技术中。



基本思想：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。

相对于折半查找，一般将待比较的key值与第mid=（low+high）/2位置的元素比较，比较结果分三种情况：

1）相等，mid位置的元素即为所求

2）>，low=mid+1;

3）<，high=mid-1。

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;

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)。

. 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：

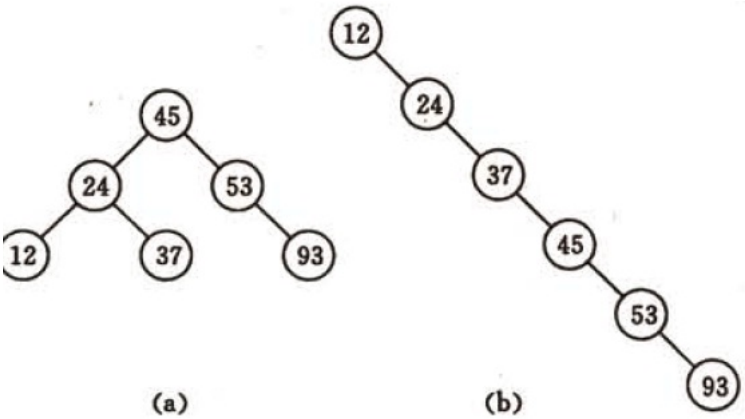
1）若任意节点的左子树不空，则左子树上所有结点的值均小于它的根结点的值；

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。

不同形态的二叉查找树如下图所示：



不同形态的二叉查找树

(a) 关键字序列为(45,24,53,12,37,93)的二叉排序树；

(b) 关键字序列为(12,24,37,45,53,93)的单支树

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（比如，我们查找上图（b）中的“93”，我们需要进行n次查找操作）。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。

下图为二叉树查找和顺序查找以及二分查找性能的对比图：

implementation	search	insert	delete	search hit	insert	delete	iteration?	on keys
sequential search (linked list)	N	N	N	N/2	N	N/2	no	equals()
binary search (ordered array)	lg N	N	N	lg N	N/2	N/2	yes	compareTo()
BST	N	N	N	1.39 lg N	1.39 lg N	√N	yes	compareTo()

other operations also become √N if deletions allowed

基于二叉查找树进行优化，进而可以得到其他的树表查找算法，如平衡树、红黑树等高效算法

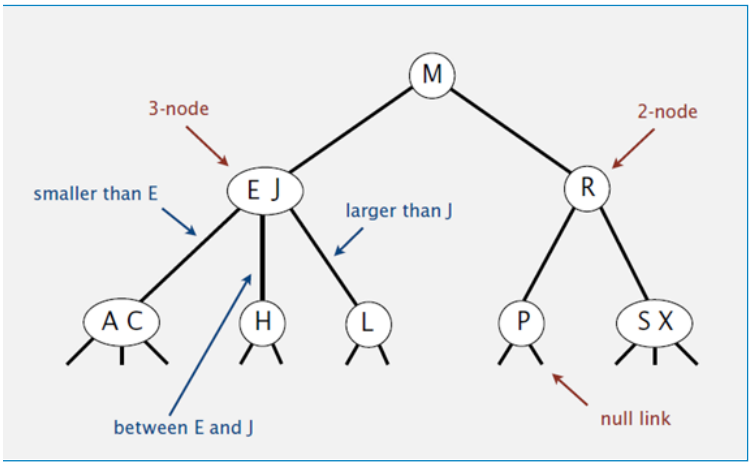
5.2 平衡查找树之2-3查找树（2-3 Tree）

2-3查找树定义：和二叉树不一样，2-3树运行每个节点保存1个或者两个的值。对于普通的2节点(2-node)，他保存1个key和左右两个自己点。对应3节点(3-node)，保存两个Key，2-3查找树的定义如下：

1）要么为空，要么：

2）对于2节点，该节点保存一个key及对应value，以及两个指向左右节点的节点，左节点也是一个2-3节点，所有的值都比key要小，右节点也是一个2-3节点，所有的值比key要大。

3）对于3节点，该节点保存两个key及对应value，以及三个指向左中右的节点。左节点也是一个2-3节点，所有的值均比两个key中的最小的key还要小；中间节点也是一个2-3节点，中间节点的key值在两个跟节点key值之间；右节点也是一个2-3节点，节点的所有key值比两个key中的最大的key还要大。

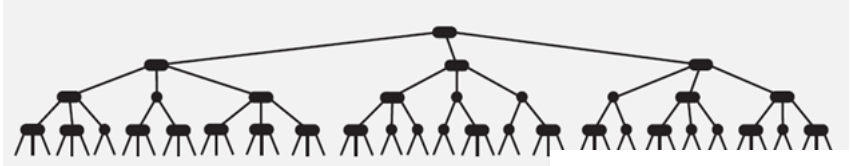


2-3查找树的性质：

1）如果中序遍历2-3查找树，就可以得到排好序的序列；

2）在一个完全平衡的2-3查找树中，根节点到每一个为空节点的距离都相同。（这也是平衡树中“平衡”一词的概念，根节点到叶节点的最长距离对应于查找算法的最坏情况，而平衡树中根节点到叶节点的距离都一样，最坏情况也具有对数复杂度。）

性质2）如下图所示：



夏杂度分析：

2-3树的查找效率与树的高度是息息相关的。

- 在最坏的情况下，也就是所有的节点都是2-node节点，查找效率为lgN
- 在最好的情况下，所有的节点都是3-node节点，查找效率为log3N约等于0.631lgN

距离来说，对于1百万个节点的2-3树，树的高度为12-20之间，对于10亿个节点的2-3对，树的高度为18-30之间。

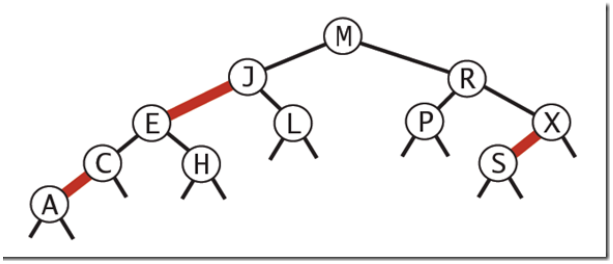
对于插入来说，只需要常数次操作即可完成，因为他只需要修改与该节点关联的节点即可，不需要检查其他节点，所以效率和查找类似。下面是2-3查找树的效率：

implementation	worst-case cost (after N inserts)			average case (after N random inserts)			ordered iteration?	key interface
	search	insert	delete	search hit	insert	delete		
sequential search (unordered list)	N	N	N	N/2	N	N/2	no	`equals()`
binary search (ordered array)	lg N	N	N	lg N	N/2	N/2	yes	`compareTo()`
BST	N	N	N	1.39 lg N	1.39 lg N	?	yes	`compareTo()`
2-3 tree	c lg N	c lg N	c lg N	c lg N	c lg N	c lg N	yes	`compareTo()`

3.3 平衡查找树之红黑树（Red-Black Tree）

2-3查找树能保证在插入元素之后能保持树的平衡状态，最坏情况下即所有的子节点都是2-node，树的高度为lgn，从而保证了最坏情况下的时间复杂度。但是2-3树实现起来比较复杂，于是就有了一种简单实现2-3树的数据结构，即红黑树（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。

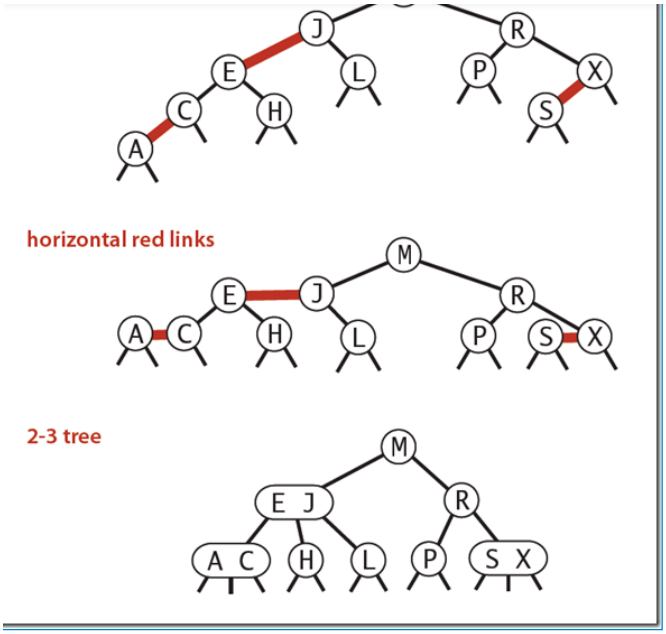


红黑树的定义：

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：

- 红色节点向左倾斜
- 一个节点不可能有两个红色链接
- 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。

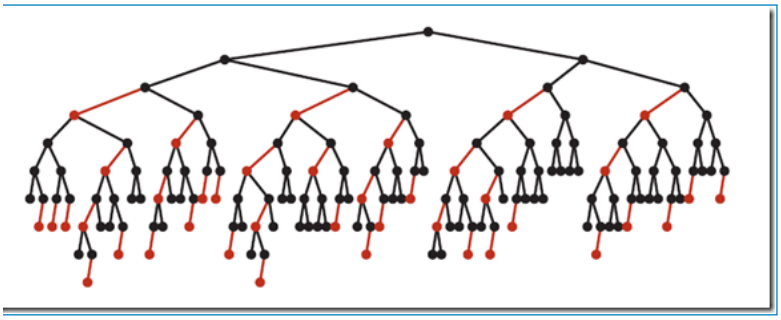
下图可以看到红黑树其实是2-3树的另外一种表现形式，如果我们将红色的连线水平绘制，那么他链接的两个2-node节点就是2-3树中的一个



工黑树的性质：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（2-3树的第2）性质，从根节点到叶子节点的距离都相等）。咋·

复杂度分析：最坏的情况就是，红黑树中除了最左侧路径全部是由3-node节点组成，即工黑相间的路径长度是全黑路径长度的2倍。

下图是一个典型的红黑树，从中可以看到最长的路径(红黑相间的路径)是最短路径的2倍：



红黑树的平均高度大约为logn。

下图是红黑树在各种情况下的时间复杂度，可以看出红黑树是2-3查找树的一种实现，它能保证最坏情况下仍然具有对数的时间复杂度。

implementation	worst-case cost (after N inserts)			average case (after N random inserts)			ordered iteration?	key interface
	search	insert	delete	search hit	insert	delete		
sequential search (unordered list)	N	N	N	N/2	N	N/2	no	`equals()`
binary search (ordered array)	lg N	N	N	lg N	N/2	N/2	yes	`compareTo()`
BST	N	N	N	1.39 lg N	1.39 lg N	?	yes	`compareTo()`
2-3 tree	c lg N	c lg N	c lg N	c lg N	c lg N	c lg N	yes	`compareTo()`
red-black BST	2 lg N	2 lg N	2 lg N	1.00 lg N *	1.00 lg N *	1.00 lg N *	yes	`compareTo()`

红黑树这种数据结构应用十分广泛，在多种编程语言中被用作符号表的实现，如：

- C++ STL中的：map,multimap,multiset,
- .NET中的：SortedDictionary,SortedSet等。

续....2018/02/26

考文献：http://www.cnblogs.com/maybe2030/p/4715035.html#_labelTop

据结构-七大查找算法 zidingxiangyu1993的博客 697
篇博客关于七大查找算法总结得非常好，参考地址如下： http://www.cnblogs.com/maybe2030/p/4715035.html#_lab…

大排序算法原理及实现 a3192048的博客 6万+
述 排序有内部排序和外部排序，内部排序是数据记录在内存中进行排序，而外部排序是因排序的数据很大，一次不能…

优质评论可以帮助作者获得更高权重 评论

ya鸡给给： 顺序查找，算法写错了吧。两个return直接退出方法了，i的值不会有增加 1年前 回复 3
谢晓永 回复： 我编译运行了，结果没错 4月前 回复
谢晓永 回复： 没写错把，作者没打括号，但是用了缩进，我编译运行了，结果没错 4月前 回复
紫柳 回复： 是啊，我这报了dead code，仔细一看不对啊 5月前 回复

麋鹿麋鹿迷了路： 二分查找错了吧。应该是while(low<=high)，注意等于号 1年前 回复 1
我～ 回复： 是的 2月前 回复

爱码士 dream_uping： 我在： 2020年 5月 24日 16:15:20 看过本篇博客！ 4月前 回复

登录 查看 11 条热评

转】七大查找算法总结 zhuhongde的博客 2877
考博客 1.顺序查找 思路：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，…

/a实现常见查找算法 朝花夕拾 1万+
查找 查找(Searching)就是根据给定的某个值，在查找表中确定一个其关键字等于给定值的数据元素(或记录)。 在互…

大查找算法_yimixgg的博客-CSDN博客 9-14
找是在大量的信息中寻找一个特定的信息元素,在计算机应用中,查找是常用的基本运算,例如编译程序中符号表的查找…

据结构–七大查找算法总结_fafawf的博客-CSDN博客 9-21
数据结构–七大查找算法总结 2017年08月15日 21:06:17 阅读数:10610 ... 基本思想:基于二分查找算法,将查找点的选…

ata Structure & Algorithm] 七大查找算法 weixin_34128839的博客 1547
查找是在大量的信息中寻找一个特定的信息元素，在计算机应用中，查找是常用的基本运算，例如编译程序中符号…

大排序算法 September 268
述 排序有内部排序和外部排序，内部排序是数据记录在内存中进行排序，而外部排序是因排序的数据很大，一次不能…

据结构--七大查找算法总结_sayhello_world的博客-CSDN博客 9-13
找是在大量的信息中寻找一个特定的信息元素,在计算机应用中,查找是常用的基本运算,例如编译程序中符号表的查找…

大查找算法_Z_hehe的博客-CSDN博客 9-8
找是在大量的信息中寻找一个特定的信息元素,在计算机应用中,查找

点赞12 评论11 分享 收藏100 手机看 关注 一键三连

忌搜索算法详解 StevenSun的博客空间 4万+
言 对于优化问题相关算法有如下分类： 禁忌搜索是由局部搜索算法发展而来，爬山法是从通用局部搜索算法改进而…

法之七大查找算法简介_weixin_35873355的博客-CSDN博客 9-25
在前面:传说查找算法有七大算法,本文将介绍简单的介绍七大查找算法,之后的文章,将对每个算法进行深入的解析,并…

大查找算法_Allenlzcoder的博客-CSDN博客 9-24
大查找算法常用的七大查找算法总结的比较好的链接: https://www.cnblogs.com/maybe2030/p/4715035.html

查找算法解析 The Coding World 4万+
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…

度优先搜索算法 流月的博客 1万+
、简介 广度优先搜索算法（Breadth-First Search，BFS）是一种盲目搜寻法，目的是系统地展开并检查图中的所有…

大查找常见算法(下)_奋斗才是年轻该有的状态-CSDN博客 9-9
、线性索引查找 1.1 简介 前面讲的几种比较高效的查找方法是基于有序的基础之上的(详见七大查找常见算法(上)…

言的七大查找算法,非常值得学习_weixin_42019584的博客-CSDN博客 9-13
天带着大家学习下,C语言七大查找算法!!!这里我们首先看下算法的概念: 算法(Algorithm)是指解题方案的准确而完整的…

没考上研究生的张同学
码龄3年 暂无认证

184	8773	2万+	20万+	
原创	周排名	总排名	访问	等级
3293	45	61	31	258
积分	粉丝	获赞	评论	收藏

TA的主页 私信 关注

搜博主文章

分类专栏


python进阶 15篇


python基础 5篇


算法 4篇


数据库 1篇


Ubuntu 1篇


python爬虫 1篇

最新评论

Java 集合中关于Iterator 和ListIterator的…
睿智福禄娃 回复 优雅的bug: 这个index是指的游标(cursor)位置,而非元素位置

七大查找算法

没错

七大查找算法

谢晓永 回复 ya鸡给给: 没写错把，作者没打括号，但是用了缩进，我编译运行了，纟 ...

七大查找算法

dream_uping: 我在： 2020年 5月 24日 16:15:20 看过本篇博客！

最新文章

2021考研计算机苏州大学872数据结构和操作系统习题

django-models类索引外键时候的related_name属性作用

django框架中表单

2020年 1篇	2019年 83篇
2018年 100篇	2017年 1篇

目录

红黑树的平均高度大约为logn。