


查找算法之顺序、二分、二叉搜索树、红黑树 详细比较总结

6 回复 449 查看



(<https://www.shiyanlou.com/user/8490>) 实验楼管理员  (<https://www.shiyanlou.com/vip>)

2016-03-24 16:50

技术分享 (<https://www.shiyanlou.com/questions/?tag=技术分享>)


一、前言

一般用符号表来储存键值对，就好像字典那样，通过索引来查找值，若键重复则覆盖值。我们能希望找到一种高效的查找算法使在平均情况和最差情况下，时间复杂度都能达到 $O(\log n)$ 。下面会逐步介绍四种算法，最终达到我们的目的。



全部回答



实验楼管理员 (<https://www.shiyanlou.com/user/8490>)  (<https://www.shiyanlou.com/vip>)

(<https://www.shiyanlou.com/user/8490>)

二、顺序查找

用链表实现，无法索引数据，必须遍历找数据，速度比较慢，查找插入时间复杂度都为 $O(n)$ ，而且无法保证有序。但是实现简单，适用于小型数据。

```
public class SequentialSearchST<Key, Value> {
    private Node head;
    private int size=0;
    public void put(Key key, Value v) {


        Node p=head;
        while (p!=null) {
            if (p.key.equals(key)) {
                p.v=v;
                return;
            }
            p=p.next;
        }

        head=new Node(key, v, head);
        size++;

    }

    public Value get(Key key) {
        Node p=head;
        while (p!=null) {
            if (p.key.equals(key)) {
                return p.v;
            }
            p=p.next;
        }
        return null;
    }
}
```

2016-03-24 16:50

实验楼管理员 (<https://www.shiyanlou.com/user/8490>)  (<https://www.shiyanlou.com/vip>)



三、二分查找

(<https://www.shiyanlou.com/user/8490>) 用数组保存数据,保证有序。二分查找速度很快,但是仅限于查找。因为插入的时候要保证有序,所以要往后移动数据以便插入。查找复杂度 $O(\log n)$,插入复杂度 $O(n)$ 。

```
public class BinarySearch<Key extends Comparable,Value> {

    public void put(Key key,Value value){
        int index=rank(key);
        //键相等则覆盖值
        if(keys[index]!=null&&key.compareTo(keys[index])==0){
            values[index]=value;
            return;
        }
        //把数据往后移,以便插入
        for(int i=size+1;i>index;i--){
            keys[i]=keys[i-1];
            values[i]=values[i-1];
        }
        keys[index]=key;
        values[index]=value;
        size++;
    }


    public Value get(Key key){
        int index=rank(key); //二分查找
        if(keys[index]!=null && key.compareTo(keys[index])==0){
            return values[index];
        }
        return null;
    }

    public int rank(Key key){return rank(key,0,size);}

    public int rank(Key key,int l,int h){
        if(l>h) return l;
        int mid = (l+h)/2;
        int cmp=0;
        if(keys[mid]!=null)
            cmp=key.compareTo(keys[mid]);
        if(cmp<0)
            return rank(key,l,mid-1);
        else if(cmp>0)
            return rank(key,mid+1,h);
        return mid;
    }
}
```

2016-03-24 16:51



实验楼管理员 (<https://www.shiyanlou.com/user/8490>)  (<https://www.shiyanlou.com/vip>)

(<https://www.shiyanlou.com/user/8490>)

四、二叉搜索树

通过前面两个算法,我们可以知道链表能快速删除插入,而二分能快速查找。所以我们想找到一种结构既是链式结构,同时又能进行二分查找,同时保证查找和插入的高效性。

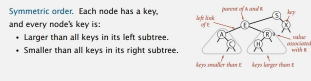
答案就是二叉搜索树。

4.1 定义

- 是二叉树
- 每个节点含有一个键和关联的值
- 且每个节点的键大于左儿子且小于右儿子

4.2 实现

其实给出定义,实现就已经很清楚了。说白了就是从无到有构造一个二叉树,每次插入都和树中的节点进行比较,小的放左边,大的放右边。就如同快速排序,用一个主元把左右两边分开。



还是直接看代码清楚点

```
public class BST<Key extends Comparable,Value>{

    Node root;

    public void put(Key key,Value value){
        root = put(root,key,value);
    }

    public Node put(Node x, Key key, Value value) {
        if(x==null){
            return new Node(key,value,0);
        }
        int cmp = key.compareTo(x.key);
        if(cmp<0) x.left=put(x.left,key,value);
        else if(cmp>0) x.right=put(x.right,key,value);
        else {
            x.value=value;
            x.N = size(x.right)+size(x.left)+1;
        }

        return x;
    }

    public Value get(Key key){
        return get(root,key);
    }

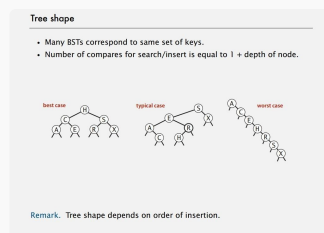
    private Value get(Node x, Key key) {
        if(x==null)
            return null;
        int cmp =key.compareTo(x.key);
        if(cmp<0) return get(x.left,key);
        else if(cmp>0) return get(x.right,key);

        return x.value;
    }
}
```

4.3 效率问题

二叉搜索树的查找和搜索在平均情况下时间复杂度都能达到 $O(\log n)$ ，而且能保证数据有序。二叉搜索树的中序遍历就是数据的顺序。我们貌似已经找到了一个最理想的算法。

但是这个效率只是在平均情况下。如果数据是逆序，或者顺序，那么这棵树就会发生一边倒的情况使复杂度直接达到 $O(n)$ ，就如同快排中选择到糟糕的主元(最大或者最小)。比快排糟糕的是，快排我们能通过随机打乱数据来避免这种情况发生。但二叉搜索树则不行，数据都是客户提供，直接插入到树中的，这种情况其实经常发生。



幸运的是我们有平衡二叉树可以解决这个问题。

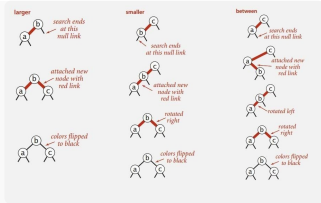
2016-03-24 16:51



实验楼管理员 (<https://www.shiyanlou.com/user/8490>) [vip](https://www.shiyanlou.com/vip)

(<https://www.shiyanlou.com/user/8490>)

五、平衡二叉树



2016-03-24 16:51



实验楼管理员 (<https://www.shiyanlou.com/user/8490>) <https://www.shiyanlou.com/vip>

(<https://www.shiyanlou.com/user/8490>)

```
public class BPlusRedBST<Key extends Comparable,Value> {
    private final boolean RED = true;
    private final boolean BLACK = false;
    private Node root;
    public void put(Key key,Value value){
        root = put(root,key,value);
        root.color = BLACK;
    }

    private Node put(Node x, Key key, Value value) {
        if(x==null) return new Node(key,value,0,RED);
        int cmp = key.compareTo(x.key);
        if(cmp<0) x.left = put(x.left,key,value);
        else if(cmp>0) x.right = put(x.right,key,value);
        else if(cmp==0) x.value =value;

        if( isRED(x.right) && !isRED(x.left)) x=rotateLeft(x);
        if( isRED(x.left) && isRED(x.left.left)) x=rotateRight(x);
        if( isRED(x.left) && isRED(x.right)) flipColor(x);
        x.N = size(x.right) + size(x.left) +1;
        return x;
    }

    private void flipColor(Node x) {
        x.right.color = BLACK;
        x.left.color = BLACK;
        x.color = RED;
    }

    private Node rotateLeft(Node x) {
        Node r =x.right;
        x.right = r.left;
        r.left = x;
        r.color = x.color;
        x.color = RED;
        x.N = size(x.left)+size(x.right) +1;
        return r;
    }

    private Node rotateRight(Node x) {
        Node r =x.left;
        x.left = r.right;
        r.right = x;
        r.left.color = RED;
        r.right.color = RED;
        r.color =BLACK;
        x.N = size(x.left)+size(x.right) +1;
        return r;
    }
}
```

性能分析

无论数据如何，插入删除时间复杂度都为 $O(\log n)$ ，可以说达到了理想状态，且代码简单。

2016-03-24 16:51



实验楼管理员 (<https://www.shiyanlou.com/user/8490>) <https://www.shiyanlou.com/vip>

(<https://www.shiyanlou.com/user/8490>)

六、性能测试

分别对四个文件进行插入搜索操作。

- **tale.txt (779kb)**

顺序查找(7.143秒);二分查找(0.46秒);二叉搜索树(0.191秒);红黑树(0.237秒)

- **leipzig100k.txt (12670kb)**

顺序查找(无);二分查找(13.911秒);二叉搜索树(1.389秒);红黑树(1.442秒)

- **leipzig300k.txt (37966kb)**

顺序查找(无);二分查找(60.222秒);二叉搜索树(2.742秒);红黑树(3.104秒)

- **leipzig1m.txt (126607kb)**

顺序查找(无);二分查找(无);二叉搜索树(3.016秒);红黑树(2.797秒)

由上面的数据分析，顺序查找实际是非常慢的。而二分查找对小型数据还是比较快，但是数据一大就不行了。

而这里的二叉搜索树和红黑树，无论什么数据效率都是极高。而且由 `leipzig300k.txt` 到 `leipzig1m.txt` 数据几乎翻了4倍，而这两种算法的效率几乎没收什么影响。

这里因为我的数据比较平均的关系，比较不出红黑树和二叉搜索树的差异。我自己构造了一组数据进行测试。完全逆序的100000个数进行插入删除。

- 红黑树(0.173秒)
- 二叉搜索树(StackOverflow)

七、Summary

Implementation	worst-case cost (after N inserts)			Average case (after N random inserts)			ordered insertion	key membership
	search	insert	delete	search for	insert	delete		
sequential search (unsorted list)	N	N	N	N/2	N	N/2	no	equal(1)
binary search (sorted array)	$\lg N$	N	N	$\lg N$	N/2	N/2	yes	compare(1)
BST	N	N	N	$1.39 \lg N$	$1.39 \lg N$	7	yes	compare(1)
2-3 tree	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	yes	compare(1)
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	compare(1)

八、Reference

- 维基百科
- 算法 4th (http://www.amazon.cn/图灵程序设计丛书-算法-塞奇威克/dp/B009OCFQ0O/ref=sr_1_1?s=books&ie=UTF8&qid=1457882078&sr=1-1&keywords=算法)

文章地址：<http://threezj.com/2016/03/20/查找算法之顺序、二分、二叉搜索树、红黑树/> (<http://threezj.com/2016/03/20/查找算法之顺序、二分、二叉搜索树、红黑树/>)

作者：threezj

2016-03-24 16:52

登录 (<https://www.shiyanlou.com/login?next=/questions/3587>)后回答问题

我要提问

标签

Linux (<https://www.shiyanlou.com/questions/?tag=Linux>) 课程相关 (<https://www.shiyanlou.com/questions/?tag=课程相关>)

Python (<https://www.shiyanlou.com/questions/?tag=Python>) 实验环境 (<https://www.shiyanlou.com/questions/?tag=实验环境>)

C/C++ (<https://www.shiyanlou.com/questions/?tag=C/C++>) 技术分享 (<https://www.shiyanlou.com/questions/?tag=技术分享>)

课程需求 (<https://www.shiyanlou.com/questions/?tag=课程需求>) 功能建议 (<https://www.shiyanlou.com/questions/?tag=功能建议>)

Java (<https://www.shiyanlou.com/questions/?tag=Java>) 其他 (<https://www.shiyanlou.com/questions/?tag=其他>)

Web (<https://www.shiyanlou.com/questions/?tag=Web>) Hadoop (<https://www.shiyanlou.com/questions/?tag=Hadoop>)

NodeJS (<https://www.shiyanlou.com/questions/?tag=NodeJS>) SQL (<https://www.shiyanlou.com/questions/?tag=SQL>)

PHP (<https://www.shiyanlou.com/questions/?tag=PHP>) Shell (<https://www.shiyanlou.com/questions/?tag=Shell>)

常见问题 (<https://www.shiyanlou.com/questions/?tag=常见问题>) Git (<https://www.shiyanlou.com/questions/?tag=Git>)

HTML (<https://www.shiyanlou.com/questions/?tag=HTML>) 网络 (<https://www.shiyanlou.com/questions/?tag=网络>)

HTML5 (<https://www.shiyanlou.com/questions/?tag=HTML5>) 信息安全 (<https://www.shiyanlou.com/questions/?tag=信息安全>)

Android (<https://www.shiyanlou.com/questions/?tag=Android>) GO (<https://www.shiyanlou.com/questions/?tag=GO>)

NoSQL (<https://www.shiyanlou.com/questions/?tag=NoSQL>) Ruby (<https://www.shiyanlou.com/questions/?tag=Ruby>)

训练营 (<https://www.shiyanlou.com/questions/?tag=训练营>) Perl (<https://www.shiyanlou.com/questions/?tag=Perl>)



实验楼客户端

即开即用



会员专属

(<https://www.shiyanlou.com/vip>)

相关问题

C/C++的mem函数和strcpy函数的区别和应用 (<https://www.shiyanlou.com/questions/5274>)

python之线程、进程和协程 (<https://www.shiyanlou.com/questions/5107>)

从底层理解Python的执行 (<https://www.shiyanlou.com/questions/5247>)

20个为前端开发者准备的文档和指南（2） (<https://www.shiyanlou.com/questions/5215>)

震惊小伙伴的单行代码—Python篇 (<https://www.shiyanlou.com/questions/4157>)

九个Console命令，让js调试更简单 (<https://www.shiyanlou.com/questions/5178>)

10款最佳PHP自动化测试框架 (<https://www.shiyanlou.com/questions/2211>)

Git 远程操作的正确姿势 (<https://www.shiyanlou.com/questions/5134>)

JavaScript异步编程解决方案笔记 (<https://www.shiyanlou.com/questions/5094>)

Vim 起步的五个技巧 (<https://www.shiyanlou.com/questions/5072>)



动手做实验，轻松学IT。

(<http://weibo.com/shiyanlou2013>)

公司

关于我们 (<https://www.shiyanlou.com/aboutus>)

联系我们 (<https://www.shiyanlou.com/contact>)

加入我们 (<http://www.simplecloud.cn/jobs.html>)

技术博客 (<https://blog.shiyanlou.com/>)

合作

我要投稿 (<https://www.shiyanlou.com/contribute>)

教师合作 (<https://www.shiyanlou.com/labs>)

高校合作 (<https://www.shiyanlou.com/edu/>)

友情链接 (<https://www.shiyanlou.com/friends>)

服务

实战训练营 (<https://www.shiyanlou.com/bootcamp/>)

会员服务 (<https://www.shiyanlou.com/vip>)

实验报告 (<https://www.shiyanlou.com/courses/reports>)

常见问题 (<https://www.shiyanlou.com/questions/?tag=常见问题>)

隐私条款 (<https://www.shiyanlou.com/privacy>)

学习路径

Python学习路径 (<https://www.shiyanlou.com/paths/python>)

Linux学习路径 (<https://www.shiyanlou.com/paths/linuxdev>)

大数据学习路径 (<https://www.shiyanlou.com/paths/bigdata>)

Java学习路径 (<https://www.shiyanlou.com/paths/java>)

PHP学习路径 (<https://www.shiyanlou.com/paths/php>)

全部 (<https://www.shiyanlou.com/paths/>)

Copyright @2013-2016 实验楼在线教育

蜀ICP备13019762号 (<http://www.miibeian.gov.cn/>) 站长统计 (http://www.cnzz.com/stat/website.php?web_id=5902315)