BPlus-tree-详解

阅读更多

1 定义

1.1 节点

节点的属性

1. n：关键字个数
2. key：关键字数组
3. c：孩子数组
4. leaf：是否为叶节点
5. next：右兄弟节点，当节点为叶子节点时该字段才有用

节点的性质

1. 节点关键字的个数
    - 根节点：[1,2t]个关键字
    - 非根节点[t,2t]个关键字
2. 所有叶子节点连接成一个单向链表

1.2 树

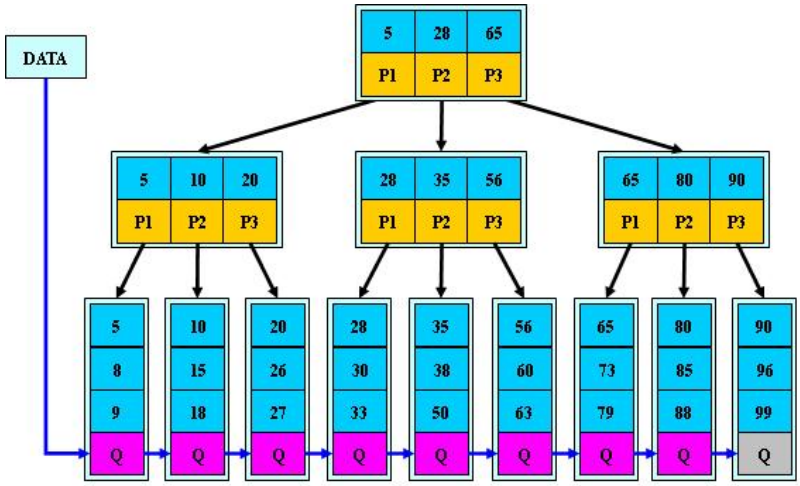
属性

1. t：B+树的度
2. root：根节点
3. data：第一个叶节点

与B树的区别

1. 所有的关键字均存储在叶节点中
2. 非叶节点中的关键字仅仅起到索引作用
3. 索引关键字不一定存在于叶节点中(由于删除，会导致该索引关键字从叶节点删除，但是索引关键字的索引作用仍然成立)
4. 每个非叶节点的索引关键字是该索引关键字对应的子树的最值的上届(由于删除，可能取不到)，可以任选一种来实现(本章选用的是最大值)
5. 非叶子节点的关键字数量和孩子数量一一对应，而B-tree中，孩子数量要比关键字数量多1
6. 所有叶子节点通过next指针串接起来，这样一来范围内的遍历要非常快捷

B±tree示意图如下



2 伪代码

B±tree的操作与B-tree的操作基本类似，注意维护好非叶子节点关键字的索引性质即可

2.1 Split

分裂一个节点

1. x.c[i]是满节点
2. x是非满节点

```
1  B+-TREE-SPLIT-CHILD(x,i)//x.ci是满节点，x是非满节点
2  y=x.c[i]
3  z= ALLOCATE-NODE()
4  y.n=z.n=t
5  for j=1 to t
6      z.key[j]=y.key[j+t]
7  if not y.leaf
8      for j=1 to t
9          z.c[j]=y.c[j+t]
10 else
11     z.next=y.next
12     y.next=z
13 z.leaf=y.leaf
14 for j=x.n+1 downto i+2
15     x.key[j]=x.key[j-1]
16     x.c[j]=x.c[j-1]
17 x.key[i+1]=x.key[i]//新增节点的索引值为原索引值
18 x.key[i]=y.key[y.n]//原节点的索引值为现有y中最大的值
19 x.c[i+1]=z
20 x.n++
```

2.2 Merge

合并指定节点，合并x.c[i]和x.c[i+1]

1. x.c[i]和x.c[i+1]关键字数量必定为t
2. x的关键字数量必定大于t

```
B+-TREE-MERGE(x,i)
y=x.c[i]
z=x.c[i+1]
y.n=2t
for j=1 to t
    y.key[j+t]=z.key[j]
if not y.leaf
    for j=1 to t
        y.c[j+t]=z.c[j]
else
    y.next=z.next
for j=i+1 to x.n-1
    x.key[j]=x.key[j+1]
    x.c[j]=x.c[j+1]
x.key[i]=y.key[y.n]//更新索引值
x.n--
```

2.3 Shift

shift方法用于删除操作时，为了保证递归的节点关键字数量大于t，要从左边或者右边挪一个节点到当前节点，这两个方法就是执行这个操作，当且仅当左右节点的关键字数量均为t时(即没有多余的关键字可以挪给其他节点了)，才执行merge操作

shift操作会导致左侧节点的索引失效，但仅仅需要改动该父节点即可（即不会触发递归向上的索引值修改），因为父节点需要修改的索引必定不是父节点的最值

```
B+-TREE-SHIFT-TO-LEFT-CHILD(x,i)
y=x.c[i]
z=x.c[i+1]
y.key[y.n+1]=z.key[1]
for j=1 to z.n-1
    z.key[j]=z.key[j+1]
if not y.leaf
    y.c[y.n+1]=z.c[1]
    for j=1 to z.n-1
        z.c[j]=z.c[j+1]
y.n++
z.n--
x.key[i]=y.key[y.n]
```

```
B+-TREE-SHIFT-TO-RIGHT-CHILD(x,i)
p=x.c[i]
y=x.c[i+1]
for j=y.n+1 downto 2
    y.key[j]=y.key[j-1]
y.key[1]=p.key[p.n]
if not y.leaf
    for j=y.n+1 downto 2
        y.c[j]=y.c[j-1]
    y.c[1]=p.c[p.n]
y.n++
p.n--
x.key[i]=p.key[p.n]
```

2.4 插入

为了避免额外的父节点指针以及实现复杂度，采用了自顶向下的**预分裂**式的插入操作

```
B+-TREE-INSERT(k)
if root.n==2t
    newRoot= ALLOCATE-NODE()
    newRoot.n=1
    newRoot.key[1]=root.key[2t]
    newRoot.c[1]=root
    newRoot.leaf=false
    root=newRoot
    B+-TREE-SPLIT-CHILD (root,1)
B+TREE-INSERT-NOT-FULL(root,k)
```

```
B+TREE-INSERT-NOT-FULL(x,k)
i=x.n
if x.leaf
    while i>=1 and x.key[i]>k
        x.key[i+1]=x.key[i]
        i--
    i++
    x.key[i]=k
    x.n++
else
    while i>=1 and x.key[i]>=k   //这个等于至关重要，虽然B+树节点不会重复，但是由
        i--
    i++
    if x.n+1==i   //这里至关重要，插入节点时需要维护索引的正确性，就在这唯一一处进
        x.key[x.n]=k
        i--
y=x.c[i]
```

```
if y.n==2t
    B+-TREE-SPLIT-CHILD(x,i)
    if k>y.key[y.n]
        i++
B+TREE-INSERT-NOT-FULL(x.c[i],k)
```

2.5 删除

为了避免额外的父节点指针以及实现复杂度，采用了自顶向下的
预分裂式的插入操作

```
B+-TREE-DELETE(k)
if not root.leaf and root.n==1
    root=root.c[1]
if root.n==2
    if not root.leaf and root.c[1].n==t and root.c[2].n==t
        B+-TREE-MERGE(root,1)
B+-TREE-DELETE-NOT-NONE(root,k)
```

```
 B+-TREE-DELETE-NOT-NONE(x,k)
 i=1
 if x.leaf
     while i<=x.n and x.key[i]<k
         i++
     while i<=x.n-1
         x.key[i]=x.key[i+1]
         i++
     x.n--
else
    while i<=x.n and x.key[i]<k  //这里必须严格小于，找到第一个满足k<=x.key[i]的
        i++
    y=x.c[i]
    if i>1
        p=x.c[i-1]
    if i<x.n
        z=x.c[i+1]
if y.n==t
    if p!=null and p.n>t
        B+-TREE-SHIFT-TO-RIGHT-CHILD(x,i-1)
    else if z!=null and z.n>t
        B+-TREE-SHIFT-TO-LEFT-CHILD(x,i)
    else if p!=null
        B+-TREE-MERGE(x,i-1)
        y=p
    else
        B+-TREE-MERGE(x,i)
B+-TREE-DELETE-NOT-NONE(y,k)
```

3 Java代码

3.1 节点

```
public class BPlusTreeNode {
    /**
     * 关键字个数
     */
    int n;

    /**
     * 关键字
     */
    int[] keys;

    /**
     * 孩子
     */
    BPlusTreeNode[] children;

    /**
     * 叶子节点
     */
    boolean isLeaf;

    /**
     * 兄弟节点
     */
    BPlusTreeNode next;

    public BPlusTreeNode(int t){
        n=0;
        keys=new int[2*t];
        children=new BPlusTreeNode[2*t];
        isLeaf=false;
        next=null;
    }

    public String toString(){
        StringBuilder sb=new StringBuilder();
        sb.append("{ size: "+n+", keys: [");
        for(int i=0;i<n;i++){
```

```
            sb.append(keys[i]+", ");
        }
        sb.append("] }");
        return sb.toString();

    }
}
```

3.2 BPlus-tree

```
package org.liuyehcf.algorithm.datastructure.tree.bplustree;

import java.util.*;

/**
 * Created by liuye on 2017/5/3 0003.
 * 该版本的B+树同样不支持重复关键字
 */
public class BPlusTree {
    private int t;

    private BPlusTreeNode root;
    private BPlusTreeNode data;

    public BPlusTree(int t) {
        this.t = t;
        root = new BPlusTreeNode(t);
        root.n = 0;
        root.isLeaf = true;

        data = root;
    }

    public void insert(int k) {
        if (root.n == 2 * t) {
            BPlusTreeNode newRoot = new BPlusTreeNode(t);
            newRoot.n = 1;
            newRoot.keys[0] = root.keys[2 * t - 1];
            newRoot.children[0] = root;
            newRoot.isLeaf = false;
            root = newRoot;
            split(root, 0);
        }
        insertNotFull(root, k);

        if (!check()) {
```

```
37              throw new RuntimeException();
38          }
39      }
40
41      private void split(BPlusTreeNode x, int i) {
42          BPlusTreeNode y = x.children[i];
43          BPlusTreeNode z = new BPlusTreeNode(t);
44
45          y.n = z.n = t;
46          for (int j = 0; j < t; j++) {
47              z.keys[j] = y.keys[j + t];
48          }
49          if (!y.isLeaf) {
50              for (int j = 0; j < t; j++) {
51                  z.children[j] = y.children[j + t];
52              }
53          } else {
54              z.next = y.next;
55              y.next = z;
56          }
57          z.isLeaf = y.isLeaf;
58
59          for (int j = x.n; j > i + 1; j--) {
60              x.keys[j] = x.keys[j - 1];
61              x.children[j] = x.children[j - 1];
62          }
63
64          x.keys[i + 1] = x.keys[i];
65          x.keys[i] = y.keys[y.n - 1];
66
67          x.children[i + 1] = z;
68          x.n++;
69      }
70
71      private void insertNotFull(BPlusTreeNode x, int k) {
72          int i = x.n - 1;
73          if (x.isLeaf) {
74              while (i >= 0 && x.keys[i] > k) {
75                  x.keys[i + 1] = x.keys[i];
76                  i--;
77              }
78              if (i >= 0 && x.keys[i] == k) {
79                  throw new RuntimeException();
80              }
81              i++;
82              x.keys[i] = k;
83              x.n++;
```

```
84          } else {
85              //todo 这个等号非常关键，执行过删除操作后，遗留下来的元素可能并不存在
86              while (i >= 0 && x.keys[i] >= k) {
87                  i--;
88              }
89              i++;
90              //todo 关键，自上而下寻找插入点时，即维护了索引的正确性
91              if (i == x.n) {
92                  //此时说明新插入的值k比当前节点中所有关键字都要大，因此当前节点的
93                  x.keys[x.n - 1] = k;
94                  i--;
95              }
96
97              BPlusTreeNode y = x.children[i];
98              if (y.n == 2 * t) {
99                  split(x, i);
100                 if (k > y.keys[y.n - 1])
101                     i++;
102             }
103             insertNotFull(x.children[i], k);
104         }
105     }
106
107     private boolean check() {
108         return checkIndex(root)
109                 && checkN(root)
110                 && checkOrder();
111     }
112
113     private boolean checkIndex(BPlusTreeNode x) {
114         if (x == null) return true;
115         for (int i = 1; i < x.n; i++) {
116             if (x.keys[i] <= x.keys[i - 1]) {
117                 return false;
118             }
119         }
120         if (!x.isLeaf) {
121             for (int i = 0; i < x.n; i++) {
122                 BPlusTreeNode child = x.children[i];
123                 if (x.keys[i] < child.keys[child.n - 1]) return false;
124                 if (i > 0 && child.keys[0] <= x.keys[i - 1]) return false;
125                 if (!checkIndex(child)) return false;
126             }
127         }
128         return true;
129     }
130
```

```
131     private boolean checkN(BPlusTreeNode x) {
132         if (x.isLeaf) {
133             return (x == root) || (x.n >= t && x.n <= 2 * t);
134         } else {
135             boolean flag = (x == root) || (x.n >= t && x.n <= 2 * t);
136             for (int i = 0; i < x.n; i++) {
137                 flag = flag && checkN(x.children[i]);
138             }
139             return flag;
140         }
141     }
142 
143     private boolean checkOrder() {
144         BPlusTreeNode x = data;
145         Integer pre = null;
146         int i = 0;
147         while (x != null && x.n > 0) {
148             if (pre == null) {
149                 pre = x.keys[i++];
150             } else {
151                 if (pre >= x.keys[i]) return false;
152                 pre = x.keys[i++];
153             }
154             if (i == x.n) {
155                 x = x.next;
156                 i = 0;
157             }
158         }
159         return true;
160     }
161 
162     private BPlusTreeNode search(BPlusTreeNode x, int k) {
163         while (!x.isLeaf) {
164             int i = 0;
165             while (i < x.n && k > x.keys[i]) {
166                 i++;
167             }
168             x = x.children[i];
169         }
170         for (int i = 0; i < x.n; i++) {
171             if (x.keys[i] == k) return x;
172         }
173         return null;
174     }
175 
176     public void delete(int k) {
177         if (!root.isLeaf && root.n == 1) {
```

```
178             root = root.children[0];
179         }
180         if (root.n == 2) {
181             if (!root.isLeaf && root.children[0].n == t && root.children[1]
182                 merge(root, 0);
183             }
184         }
185         deleteNotNone(root, k);
186         if (!check()) {
187             throw new RuntimeException();
188         }
189     }
190 
191     private void deleteNotNone(BPlusTreeNode x, int k) {
192         int i = 0;
193         if (x.isLeaf) {
194             while (i < x.n && x.keys[i] < k) {
195                 i++;
196             }
197             if (k != x.keys[i]) {
198                 throw new RuntimeException();
199             }
200             while (i < x.n - 1) {
201                 x.keys[i] = x.keys[i + 1];
202                 i++;
203             }
204             x.n--;
205         } else {
206             while (i < x.n && x.keys[i] < k) {
207                 i++;
208             }
209             BPlusTreeNode y = x.children[i];
210             BPlusTreeNode p = null, z = null;
211             if (i > 0) {
212                 p = x.children[i - 1];
213             }
214             if (i < x.n - 1) {
215                 z = x.children[i + 1];
216             }
217             if (y.n == t) {
218                 if (p != null && p.n > t) {
219                     shiftToRight(x, i - 1);
220                 } else if (z != null && z.n > t) {
221                     shiftToLeft(x, i);
222                 } else if (p != null) {
223                     merge(x, i - 1);
224                     y = p;
```

```
225                 } else {
226                     merge(x, i);
227                 }
228             }
229             deleteNotNone(y, k);
230         }
231     }
232 
233     private void merge(BPlusTreeNode x, int i) {
234         BPlusTreeNode y = x.children[i];
235         BPlusTreeNode z = x.children[i + 1];
236 
237         y.n = 2 * t;
238         for (int j = 0; j < t; j++) {
239             y.keys[j + t] = z.keys[j];
240         }
241         if (!y.isLeaf) {
242             for (int j = 0; j < t; j++) {
243                 y.children[j + t] = z.children[j];
244             }
245         } else {
246             y.next = z.next;
247         }
248 
249         for (int j = i + 1; j < x.n - 1; j++) {
250             x.keys[j] = x.keys[j + 1];
251             x.children[j] = x.children[j + 1];
252         }
253         x.keys[i] = y.keys[y.n - 1];
254         x.n--;
255     }
256 
257     private void shiftToLeft(BPlusTreeNode x, int i) {
258         BPlusTreeNode y = x.children[i];
259         BPlusTreeNode z = x.children[i + 1];
260 
261         y.keys[y.n] = z.keys[0];
262         for (int j = 0; j < z.n - 1; j++) {
263             z.keys[j] = z.keys[j + 1];
264         }
265         if (!y.isLeaf) {
266             y.children[y.n] = z.children[0];
267             for (int j = 0; j < z.n - 1; j++) {
268                 z.children[j] = z.children[j + 1];
269             }
270         }
271         y.n++;
```

```
272         z.n--;
273 
274         x.keys[i] = y.keys[y.n - 1];
275     }
276 
277     private void shiftToRight(BPlusTreeNode x, int i) {
278         BPlusTreeNode p = x.children[i];
279         BPlusTreeNode y = x.children[i + 1];
280 
281         for (int j = y.n; j > 0; j--) {
282             y.keys[j] = y.keys[j - 1];
283         }
284         y.keys[0] = p.keys[p.n - 1];
285 
286         if (!y.isLeaf) {
287             for (int j = y.n; j > 0; j--) {
288                 y.children[j] = y.children[j - 1];
289             }
290             y.children[0] = p.children[p.n - 1];
291         }
292         y.n++;
293         p.n--;
294 
295         x.keys[i] = p.keys[p.n - 1];
296     }
297 
298     public void levelOrderTraverse() {
299         Queue<BPlusTreeNode> queue = new LinkedList<BPlusTreeNode>();
300         queue.offer(root);
301         while (!queue.isEmpty()) {
302             int len = queue.size();
303             while (len-- > 0) {
304                 BPlusTreeNode peek = queue.poll();
305                 System.out.print(peek + ", ");
306                 if (!peek.isLeaf) {
307                     for (int i = 0; i < peek.n; i++) {
308                         queue.offer(peek.children[i]);
309                     }
310                 }
311             }
312             System.out.println();
313         }
314     }
315 
316     public List<Integer> getOrderedList() {
317         List<Integer> list = new ArrayList<Integer>();
318         BPlusTreeNode x = data;
```

```
319         while (x != null) {
320             for (int i = 0; i < x.n; i++) {
321                 list.add(x.keys[i]);
322             }
323             x = x.next;
324         }
325         return list;
326     }
327 
328     public static void main(String[] args) {
329         long start = System.currentTimeMillis();
330 
331         Random random = new Random();
332 
333         int TIMES = 500;
334 
335         while (--TIMES > 0) {
336             System.out.println("剩余测试次数: " + TIMES);
337             BPlusTree bPlusTree = new BPlusTree(random.nextInt(20) + 2);
338 
339             int N = 10000;
340 
341             Set<Integer> set = new HashSet<Integer>();
342             for (int i = 0; i < N; i++) {
343                 set.add(random.nextInt());
344             }
345 
346             List<Integer> list = new ArrayList<Integer>(set);
347             Collections.shuffle(list, random);
348             //插入N个数据
349             for (int i : list) {
350                 bPlusTree.insert(i);
351             }
352 
353             int M = list.size() / 2;
354 
355             //删除M个数据
356             Collections.shuffle(list, random);
357 
358             for (int i = 0; i < M; i++) {
359                 set.remove(list.get(i));
360                 bPlusTree.delete(list.get(i));
361             }
362 
363             //再插入M个数据
364             for (int i = 0; i < M; i++) {
365                 int k = random.nextInt();
```

```
366                 if (set.add(k)) {
367                     bPlusTree.insert(k);
368                 }
369             }
370             list.clear();
371             list.addAll(set);
372             Collections.shuffle(list, random);
373 
374             //再删除所有元素
375             for (int i : list) {
376                 bPlusTree.delete(i);
377             }
378         }
379         long end = System.currentTimeMillis();
380         System.out.println("Run time: " + (end - start) / 1000 + "s");
381     }
382 }
```