

Join GitHub today

Sign up

Find file	Copy path
-----------	-----------

2c520b8 26 days ago

Raw Blame History   

```

1 package index;
2
3 import static org.junit.Assert.assertEquals;
4 import static org.junit.Assert.assertTrue;
5
6 import java.io.File;
7 import java.io.IOException;
8 import java.util.ArrayList;
9 import java.util.Arrays;
10 import java.util.List;
11 import java.util.Optional;
12
13 import org.junit.Before;
14 import org.junit.Rule;
15 import org.junit.Test;
16 import org.junit.rules.DisableOnDebug;
17 import org.junit.rules.TemporaryFolder;
18 import org.junit.rules.TestRule;
19 import org.junit.rules.Timeout;
20
21 import common.Pair;
22 import databox.DataBox;
23 import databox.IntDataBox;
24 import databox.Type;
25 import io.PageAllocator;
26 import table.RecordId;
27
28 public class TestInnerNode {
29     public static final String testFile = "TestInnerNode";
30
31     // The page allocator associated with the TestInnerNode file. See
32     // resetMembers for initialization.
33     PageAllocator allocator;
34
35     @Rule
36     public TemporaryFolder tempFolder = new TemporaryFolder();
37
38     // 1 second max per method tested.
39     @Rule
40     public TestRule globalTimeout = new DisableOnDebug(Timeout.seconds(1));
41
42     // inner, leaf0, leaf1, and leaf2 collectively form the following B+ tree:
43     //
44     //                                     inner
45     //                                     +-----+-----+

```

```
//      | 10 | 20 |           |
//      +-----+
//      /       |       \
//     /        |        \
//    /         |         \
//   +-----+ +-----+ +-----+
//  | 1 | 2 | 3 | | 11 | 12 | 13 | | 21 | 22 | 23 | |
//  +-----+ +-----+ +-----+
// leaf0          leaf1          leaf2
//
// innerKeys, innerChildren, keys0, rids0, keys1, rids1, keys2, and rids2
// hold *copies* of the contents of the nodes. To test out a certain method
// of a tree---for example, put---we can issue a put against the tree,
// update one of innerKeys, innerChildren, keys{0,1,2}, or rids{0,1,2}, and
// then check that the contents of the tree match our expectations. For
// example:
//
// IntDataBox key = new IntDataBox(4);
// RecordId rid = new RecordId(4, (short) 4);
// inner.put(key, rid);
//
// // (4, (4, 4)) is added to leaf 0, so we update keys0 and rids0 and
// // check that it matches the contents of leaf0.
// keys0.add(key);
// rids0.add(rid);
// assertEquals(keys0, getLeaf(leaf0).getKeys());
// assertEquals(rids0, getLeaf(leaf0).getRids());
//
// // Leaf 1 should be unchanged which we can check:
// assertEquals(keys1, getLeaf(leaf1).getKeys());
// assertEquals(rids1, getLeaf(leaf1).getRids());
//
// // Writing all these assertEquals is boilerplate, so we can abstract
// // it in checkTreeMatchesExpectations().
// checkTreeMatchesExpectations();
//
// Note that we cannot simply store the LeafNodes as members because when
// we call something like inner.put(k), the inner node constructs a new
// LeafNode from the serialization and forwards the put to that. It would
// not affect our the in-memory values of our members. Also note that all
// of these members are initialized by resetMembers before every test case
// is run.
private List<DataBox> innerKeys;
private List<Integer> innerChildren;
private InnerNode inner;
private List<DataBox> keys0;
private List<RecordId> rids0;
private int leaf0;
private List<DataBox> keys1;
private List<RecordId> rids1;
private int leaf1;
private List<DataBox> keys2;
private List<RecordId> rids2;
private int leaf2;

// See comment above.
@Before
public void resetMembers() throws IOException {
    File file = tempFolder.newFile(testFile);
    this allocator = new PageAllocator(file.getAbsolutePath(), false);

    BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), 2);

    // Leaf 2
    List<DataBox> keys2 = new ArrayList<>();
    keys2.add(new IntDataBox(21));
    keys2.add(new IntDataBox(22));
    keys2.add(new IntDataBox(23));
    List<RecordId> rids2 = new ArrayList<>();
```

```

118     rids2.add(new RecordId(21, (short) 21));
119     rids2.add(new RecordId(22, (short) 22));
120     rids2.add(new RecordId(23, (short) 23));
121     Optional<Integer> sibling2 = Optional.empty();
122     LeafNode leaf2 = new LeafNode(meta, keys2, rids2, sibling2);
123
124     this.keys2 = new ArrayList<>(keys2);
125     this.rids2 = new ArrayList<>(rids2);
126     this.leaf2 = leaf2.getPage().getPageNum();
127
128     // Leaf 1
129     keys1 = new ArrayList<>();
130     keys1.add(new IntDataBox(11));
131     keys1.add(new IntDataBox(12));
132     keys1.add(new IntDataBox(13));
133     rids1 = new ArrayList<>();
134     rids1.add(new RecordId(11, (short) 11));
135     rids1.add(new RecordId(12, (short) 12));
136     rids1.add(new RecordId(13, (short) 13));
137     Optional<Integer> sibling1 = Optional.of(leaf2.getPage().getPageNum());
138     LeafNode leaf1 = new LeafNode(meta, keys1, rids1, sibling1);
139
140     this.keys1 = new ArrayList<>(keys1);
141     this.rids1 = new ArrayList<>(rids1);
142     this.leaf1 = leaf1.getPage().getPageNum();
143
144     // Leaf 0
145     List<DataBox> keys0 = new ArrayList<>();
146     keys0.add(new IntDataBox(1));
147     keys0.add(new IntDataBox(2));
148     keys0.add(new IntDataBox(3));
149     List<RecordId> rids0 = new ArrayList<>();
150     rids0.add(new RecordId(1, (short) 1));
151     rids0.add(new RecordId(2, (short) 2));
152     rids0.add(new RecordId(3, (short) 3));
153     Optional<Integer> sibling0 = Optional.of(leaf1.getPage().getPageNum());
154     LeafNode leaf0 = new LeafNode(meta, keys0, rids0, sibling0);
155     this.keys0 = new ArrayList<>(keys0);
156     this.rids0 = new ArrayList<>(rids0);
157     this.leaf0 = leaf0.getPage().getPageNum();
158
159     // Inner node
160     List<DataBox> innerKeys = new ArrayList<>();
161     innerKeys.add(new IntDataBox(10));
162     innerKeys.add(new IntDataBox(20));
163
164     List<Integer> innerChildren = new ArrayList<>();
165     innerChildren.add(this.leaf0);
166     innerChildren.add(this.leaf1);
167     innerChildren.add(this.leaf2);
168
169     this.innerKeys = new ArrayList<>(innerKeys);
170     this.innerChildren = new ArrayList<>(innerChildren);
171     this.inner = new InnerNode(meta, innerKeys, innerChildren);
172 }
173
174 // See comment above.
175 private LeafNode getLeaf(int pageNum) throws IOException {
176     BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), 2);
177     return LeafNode.fromBytes(meta, pageNum);
178 }
179
180 // See comment above.
181 private void checkTreeMatchesExpectations() throws IOException {
182     LeafNode leaf0 = getLeaf(this.leaf0);
183     LeafNode leaf1 = getLeaf(this.leaf1);
184     LeafNode leaf2 = getLeaf(this.leaf2);
185
186     assertEquals(keys0, leaf0.getKeys());

```

```

183     assertEquals(rids0, leaf0.getRids());
184     assertEquals(keys1, leaf1.getKeys());
185     assertEquals(rids1, leaf1.getRids());
186     assertEquals(keys2, leaf2.getKeys());
187     assertEquals(rids2, leaf2.getRids());
188     assertEquals(innerKeys, inner.getKeys());
189     assertEquals(innerChildren, inner.getChildren());
190 }
191
192
193 private BPlusTreeMetadata getBPlusTreeMetadata(Type keySchema, int order)
194     throws IOException {
195     return new BPlusTreeMetadata(allocator, keySchema, order);
196 }
197
198 // Tests //////////////////////////////////////
199 @Test
200 public void testGet() throws IOException {
201     LeafNode leaf0 = getLeaf(this.leaf0);
202     for (int i = 0; i < 10; ++i) {
203         assertEquals(leaf0, inner.get(new IntDataBox(i)));
204     }
205
206     LeafNode leaf1 = getLeaf(this.leaf1);
207     for (int i = 10; i < 20; ++i) {
208         assertEquals(leaf1, inner.get(new IntDataBox(i)));
209     }
210
211     LeafNode leaf2 = getLeaf(this.leaf2);
212     for (int i = 20; i < 30; ++i) {
213         assertEquals(leaf2, inner.get(new IntDataBox(i)));
214     }
215 }
216
217 @Test
218 public void testGetLeftmostLeaf() throws IOException {
219     assertEquals(getLeaf(leaf0), inner.getLeftmostLeaf());
220 }
221
222 @Test
223 public void testNoOverflowPuts() throws BPlusTreeException, IOException {
224     IntDataBox key = null;
225     RecordId rid = null;
226
227     // Add to leaf 0.
228     key = new IntDataBox(0);
229     rid = new RecordId(0, (short) 0);
230     assertEquals(Optional.empty(), inner.put(key, rid));
231     keys0.add(0, key);
232     rids0.add(0, rid);
233     checkTreeMatchesExpectations();
234
235     // Add to leaf 1.
236     key = new IntDataBox(14);
237     rid = new RecordId(14, (short) 14);
238     assertEquals(Optional.empty(), inner.put(key, rid));
239     keys1.add(3, key);
240     rids1.add(3, rid);
241     checkTreeMatchesExpectations();
242
243     // Add to leaf 2.
244     key = new IntDataBox(20);
245     rid = new RecordId(20, (short) 20);
246     assertEquals(Optional.empty(), inner.put(key, rid));
247     keys2.add(0, key);
248     rids2.add(0, rid);
249     checkTreeMatchesExpectations();
250 }
251
252 // HIDDEN

```

```

253 @Test
254 public void testOverflowPuts() throws BPlusTreeException, IOException {
255     // Overflow the first leaf. The tree look like this:
256     //
257     //   (3, 10, 20)
258     //   (1, 2) (3, 4, 5) (11, 12, 13) (21, 22, 23)
259     //       a
260     inner.put(new IntDataBox(4), new RecordId(4, (short) 4));
261     inner.put(new IntDataBox(5), new RecordId(5, (short) 5));
262
263     int leafa =
264         getLeaf(this.leaf0).getRightSibling().get().getPage().getPageNum();
265     innerKeys.add(0, new IntDataBox(3));
266     innerChildren.add(1, leafa);
267     assertEquals(innerKeys, inner.getKeys());
268     assertEquals(innerChildren, inner.getChildren());
269
270     // Overflow the second leaf.
271     //
272     //   (3, 5, 10, 20)
273     //   (1, 2) (3, 4) (5, 6, 7) (11, 12, 13) (21, 22, 23)
274     //       a       b
275     inner.put(new IntDataBox(6), new RecordId(6, (short) 6));
276     inner.put(new IntDataBox(7), new RecordId(7, (short) 7));
277
278     int leafb = getLeaf(leafa).getRightSibling().get().getPage().getPageNum();
279     innerKeys.add(1, new IntDataBox(5));
280     innerChildren.add(2, leafb);
281     assertEquals(innerKeys, inner.getKeys());
282     assertEquals(innerChildren, inner.getChildren());
283
284     // Again! This one overflows the index too.
285     //   (7)
286     //   (3, 5) (10, 20)
287     //   (1, 2) (3, 4) (5, 6) (7, 8, 9) (11, 12, 13) (21, 22, 23)
288     //       a       b       c
289     inner.put(new IntDataBox(8), new RecordId(8, (short) 8));
290     Optional<Pair<DataBox, Integer>> o =
291         inner.put(new IntDataBox(9), new RecordId(9, (short) 9));
292
293     assertTrue(o.isPresent());
294     Pair<DataBox, Integer> p = o.get();
295     DataBox splitKey = p.getFirst();
296     Integer newInnerPageNum = p.getSecond();
297
298     assertEquals(new IntDataBox(7), splitKey);
299
300     innerKeys = innerKeys.subList(0, 2);
301     innerChildren = innerChildren.subList(0, 3);
302     assertEquals(innerKeys, inner.getKeys());
303     assertEquals(innerChildren, inner.getChildren());
304
305     BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), 2);
306     InnerNode newInner = InnerNode.fromBytes(meta, newInnerPageNum);
307
308     List<DataBox> newInnerKeys = new ArrayList<>();
309     newInnerKeys.add(new IntDataBox(10));
310     newInnerKeys.add(new IntDataBox(20));
311
312     List<Integer> newInnerChildren = new ArrayList<>();
313     int leafc = getLeaf(leafb).getRightSibling().get().getPage().getPageNum();
314     newInnerChildren.add(leafc);
315     newInnerChildren.add(this.leaf1);
316     newInnerChildren.add(this.leaf2);
317
318     assertEquals(newInnerKeys, newInner.getKeys());
319     assertEquals(newInnerChildren, newInner.getChildren());
320
321     // Make sure we can read inner from disk.

```

```

322     int innerPageNum = inner.getPage().getPageNum();
323     InnerNode innerFromDisk = InnerNode.fromBytes(meta, innerPageNum);
324     assertEquals(innerKeys, innerFromDisk.getKeys());
325     assertEquals(innerChildren, innerFromDisk.getChildren());
326 }
327
328 @Test
329 public void testRemove() throws IOException {
330     // Remove from leaf 0.
331     inner.remove(new IntDataBox(1));
332     keys0.remove(0);
333     rids0.remove(0);
334     checkTreeMatchesExpectations();
335
336     inner.remove(new IntDataBox(3));
337     keys0.remove(1);
338     rids0.remove(1);
339     checkTreeMatchesExpectations();
340
341     inner.remove(new IntDataBox(2));
342     keys0.remove(0);
343     rids0.remove(0);
344     checkTreeMatchesExpectations();
345
346     // Remove from leaf 1.
347     inner.remove(new IntDataBox(11));
348     keys1.remove(0);
349     rids1.remove(0);
350     checkTreeMatchesExpectations();
351
352     inner.remove(new IntDataBox(13));
353     keys1.remove(1);
354     rids1.remove(1);
355     checkTreeMatchesExpectations();
356
357     inner.remove(new IntDataBox(12));
358     keys1.remove(0);
359     rids1.remove(0);
360     checkTreeMatchesExpectations();
361
362     // Remove from leaf 2.
363     inner.remove(new IntDataBox(23));
364     keys2.remove(2);
365     rids2.remove(2);
366     checkTreeMatchesExpectations();
367
368     inner.remove(new IntDataBox(22));
369     keys2.remove(1);
370     rids2.remove(1);
371     checkTreeMatchesExpectations();
372
373     inner.remove(new IntDataBox(21));
374     keys2.remove(0);
375     rids2.remove(0);
376     checkTreeMatchesExpectations();
377 }
378
379 @Test
380 public void testMaxOrder() {
381     // Note that this white box test depend critically on the implementation
382     // of toBytes and includes a lot of magic numbers that won't make sense
383     // unless you read toBytes.
384     assertEquals(4, Type.intType().getSizeInBytes());
385     assertEquals(6, RecordId.getIdSizeInBytes());
386     for (int d = 0; d < 10; ++d) {
387         int dd = d + 1;
388         for (int i = 5 + (2*d*4) + ((2*d+1)*4); i < 5 + (2*dd*4) + ((2*dd+1)*4); ++i) {
389             assertEquals(d, InnerNode.maxOrder(i, Type.intType()));
390         }
391     }

```

```

391     }
392 }
393
394 @Test
395 public void testnumLessThanEqual() {
396     List<Integer> empty = Arrays.asList();
397     assertEquals(0, InnerNode.numLessThanEqual(0, empty));
398
399     List<Integer> contiguous = Arrays.asList(1, 2, 3, 4, 5);
400     assertEquals(0, InnerNode.numLessThanEqual(0, contiguous));
401     assertEquals(1, InnerNode.numLessThanEqual(1, contiguous));
402     assertEquals(2, InnerNode.numLessThanEqual(2, contiguous));
403     assertEquals(3, InnerNode.numLessThanEqual(3, contiguous));
404     assertEquals(4, InnerNode.numLessThanEqual(4, contiguous));
405     assertEquals(5, InnerNode.numLessThanEqual(5, contiguous));
406     assertEquals(5, InnerNode.numLessThanEqual(6, contiguous));
407     assertEquals(5, InnerNode.numLessThanEqual(7, contiguous));
408
409     List<Integer> sparseWithDuplicates = Arrays.asList(1, 3, 3, 3, 5);
410     assertEquals(0, InnerNode.numLessThanEqual(0, sparseWithDuplicates));
411     assertEquals(1, InnerNode.numLessThanEqual(1, sparseWithDuplicates));
412     assertEquals(1, InnerNode.numLessThanEqual(2, sparseWithDuplicates));
413     assertEquals(4, InnerNode.numLessThanEqual(3, sparseWithDuplicates));
414     assertEquals(4, InnerNode.numLessThanEqual(4, sparseWithDuplicates));
415     assertEquals(5, InnerNode.numLessThanEqual(5, sparseWithDuplicates));
416     assertEquals(5, InnerNode.numLessThanEqual(6, sparseWithDuplicates));
417     assertEquals(5, InnerNode.numLessThanEqual(7, sparseWithDuplicates));
418 }
419
420 @Test
421 public void testnumLessThan() {
422     List<Integer> empty = Arrays.asList();
423     assertEquals(0, InnerNode.numLessThanEqual(0, empty));
424
425     List<Integer> contiguous = Arrays.asList(1, 2, 3, 4, 5);
426     assertEquals(0, InnerNode.numLessThan(0, contiguous));
427     assertEquals(0, InnerNode.numLessThan(1, contiguous));
428     assertEquals(1, InnerNode.numLessThan(2, contiguous));
429     assertEquals(2, InnerNode.numLessThan(3, contiguous));
430     assertEquals(3, InnerNode.numLessThan(4, contiguous));
431     assertEquals(4, InnerNode.numLessThan(5, contiguous));
432     assertEquals(5, InnerNode.numLessThan(6, contiguous));
433     assertEquals(5, InnerNode.numLessThan(7, contiguous));
434
435     List<Integer> sparseWithDuplicates = Arrays.asList(1, 3, 3, 3, 5);
436     assertEquals(0, InnerNode.numLessThan(0, sparseWithDuplicates));
437     assertEquals(0, InnerNode.numLessThan(1, sparseWithDuplicates));
438     assertEquals(1, InnerNode.numLessThan(2, sparseWithDuplicates));
439     assertEquals(1, InnerNode.numLessThan(3, sparseWithDuplicates));
440     assertEquals(4, InnerNode.numLessThan(4, sparseWithDuplicates));
441     assertEquals(4, InnerNode.numLessThan(5, sparseWithDuplicates));
442     assertEquals(5, InnerNode.numLessThan(6, sparseWithDuplicates));
443     assertEquals(5, InnerNode.numLessThan(7, sparseWithDuplicates));
444 }
445
446 @Test
447 public void testToSexp() {
448     String leaf0 = "((1 (1 1)) (2 (2 2)) (3 (3 3)))";
449     String leaf1 = "((11 (11 11)) (12 (12 12)) (13 (13 13)))";
450     String leaf2 = "((21 (21 21)) (22 (22 22)) (23 (23 23)))";
451     String expected = String.format("(%s 10 %s 20 %s)", leaf0, leaf1, leaf2);
452     assertEquals(expected, inner.toSexp());
453 }
454
455 @Test
456 public void testToAndFromBytes() throws IOException {
457     int d = 5;
458     BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), d);
459 }

```

```
460 List<DataBox> keys = new ArrayList<>();
461 List<Integer> children = new ArrayList<>();
462 children.add(42);
463
464 for (int i = 0; i < 2 * d; ++i) {
465     keys.add(new IntDataBox(i));
466     children.add(i);
467
468     InnerNode inner = new InnerNode(meta, keys, children);
469     int pageNum = inner.getPage().getPageNum();
470     InnerNode parsed = InnerNode.fromBytes(meta, pageNum);
471     assertEquals(inner, parsed);
472 }
473 }
474 }
```