Branch: master ▾ DB2 / src / test / java / index / TestBPlusTree.java

Find file | Copy path

dreamlegends init add all files 2c520b8 26 days ago

1 contributor

570 lines (504 sloc) 18.5 KB

Raw | Blame | History

```
package index;
import static org.junit.Assert.assertEquals;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;
import java.util.Optional;
import java.util.Random;

import org.junit.Before;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.DisableOnDebug;
import org.junit.rules.TemporaryFolder;
import org.junit.rules.TestRule;
import org.junit.rules.Timeout;

import databox.DataBox;
import databox.IntDataBox;
import databox.Type;
import table.RecordId;

public class TestBPlusTree {
    public static final String filename = "TestBPlusTree";
    private File file;

    @Rule
    public TemporaryFolder tempFolder = new TemporaryFolder();

    // 10 seconds max per method tested.
    @Rule
    public TestRule globalTimeout = new DisableOnDebug(Timeout.seconds(10));

    // Helpers ////////////////////////////////////////////////////////////////
    @Before
    public void initFile() throws IOException {
      this.file = tempFolder.newFile(filename);
    }

    private BPlusTree getBPlusTree(Type keySchema, int order)
        throws BPlusTreeException, IOException {
      return new BPlusTree(file.getAbsolutePath(), keySchema, order);
```

```
  }

  private static <T> List<T> iteratorToList(Iterator<T> iter) {
    List<T> xs = new ArrayList<>();
    while (iter.hasNext()) {
      xs.add(iter.next());
    }
    return xs;
  }

  // Tests ////////////////////////////////////////////////////////////////////
  // HIDDEN
  @Test
  public void testEmptyTree() throws BPlusTreeException, IOException {
    BPlusTree tree = getBPlusTree(Type.intType(), 2);
    List<RecordId> empty = new ArrayList<>();

    // Make sure that operations on an empty B+ tree doesn't throw any
    // exceptions.
    for (int i = 0; i < 10; ++i) {
      tree.remove(new IntDataBox(i));
      assertEquals(Optional.empty(), tree.get(new IntDataBox(i)));
      Iterator<RecordId> eq = tree.scanEqual(new IntDataBox(i));
      Iterator<RecordId> all = tree.scanAll();
      Iterator<RecordId> ge = tree.scanGreaterEqual(new IntDataBox(i));
      assertEquals(empty, iteratorToList(eq));
      assertEquals(empty, iteratorToList(all));
      assertEquals(empty, iteratorToList(ge));
    }
  }

  // HIDDEN
  @Test
  public void testBPlusTreeFromDisk() throws BPlusTreeException, IOException {
    BPlusTree tree = getBPlusTree(Type.intType(), 2);
    for (int i = 0; i < 100; ++i) {
      tree.put(new IntDataBox(i), new RecordId(i, (short) i));
    }

    BPlusTree fromDisk = new BPlusTree(file.getAbsolutePath());
    for (int i = 0; i < 100; ++i) {
      IntDataBox key = new IntDataBox(i);
      RecordId rid = new RecordId(i, (short) i);
      assertEquals(Optional.of(rid), fromDisk.get(key));
    }
  }

  // HIDDEN
  @Test
  public void testSimpleGets() throws BPlusTreeException, IOException {
    BPlusTree tree = getBPlusTree(Type.intType(), 2);
    for (int i = 0; i < 100; ++i) {
      tree.put(new IntDataBox(i), new RecordId(i, (short) i));
    }

    for (int i = 0; i < 100; ++i) {
      IntDataBox key = new IntDataBox(i);
      RecordId rid = new RecordId(i, (short) i);
      assertEquals(Optional.of(rid), tree.get(key));
    }

    for (int i = 100; i < 150; ++i) {
      assertEquals(Optional.empty(), tree.get(new IntDataBox(i)));
    }
  }

  // HIDDEN
  @Test
  public void testEmptyScans() throws BPlusTreeException, IOException {
```

```
    // Create and then empty the tree.
    BPlusTree tree = getBPlusTree(Type.intType(), 2);
    for (int i = 0; i < 100; ++i) {
      tree.put(new IntDataBox(i), new RecordId(i, (short) i));
    }
    for (int i = 0; i < 100; ++i) {
      tree.remove(new IntDataBox(i));
    }

    // Scan over the tree.
    Iterator<RecordId> actual = tree.scanAll();
    assertEquals(new ArrayList<RecordId>(), iteratorToList(actual));
    actual = tree.scanGreaterEqual(new IntDataBox(42));
    assertEquals(new ArrayList<RecordId>(), iteratorToList(actual));
    actual = tree.scanGreaterEqual(new IntDataBox(100));
    assertEquals(new ArrayList<RecordId>(), iteratorToList(actual));
  }

  // HIDDEN
  @Test
  public void testPartiallyEmptyScans()
      throws BPlusTreeException, IOException {
    // Create and then empty part of the tree.
    BPlusTree tree = getBPlusTree(Type.intType(), 2);
    for (int i = 0; i < 100; ++i) {
      tree.put(new IntDataBox(i), new RecordId(i, (short) i));
    }
    for (int i = 25; i < 75; ++i) {
      tree.remove(new IntDataBox(i));
    }

    // Scan over the tree.
    Iterator<RecordId> actual = tree.scanAll();
    List<RecordId> expected = new ArrayList<>();
    for (int i = 0; i < 25; ++i) {
      expected.add(new RecordId(i, (short) i));
    }
    for (int i = 75; i < 100; ++i) {
      expected.add(new RecordId(i, (short) i));
    }
    assertEquals(expected, iteratorToList(actual));

    actual = tree.scanGreaterEqual(new IntDataBox(42));
    expected = new ArrayList<>();
    for (int i = 75; i < 100; ++i) {
      expected.add(new RecordId(i, (short) i));
    }
    assertEquals(expected, iteratorToList(actual));

    actual = tree.scanGreaterEqual(new IntDataBox(99));
    expected = new ArrayList<>();
    expected.add(new RecordId(99, (short) 99));
    assertEquals(expected, iteratorToList(actual));
  }

  @Test(expected = BPlusTreeException.class)
  public void testDuplicatePut() throws BPlusTreeException, IOException {
    BPlusTree tree = getBPlusTree(Type.intType(), 2);
    tree.put(new IntDataBox(0), new RecordId(0, (short) 0));
    tree.put(new IntDataBox(0), new RecordId(0, (short) 0));
  }

  // HIDDEN
  @Test
  public void testRandomRids() throws BPlusTreeException, IOException {
    int d = 3;
    BPlusTree tree = getBPlusTree(Type.intType(), d);

    List<DataBox> keys = new ArrayList<DataBox>();
```

```
    List<RecordId> rids = new ArrayList<RecordId>();
    for (int i = 0; i < 50 * d; ++i) {
      keys.add(new IntDataBox(i));
      rids.add(new RecordId(i, (short) i));
    }
    Collections.shuffle(rids, new Random(42));

    for (int i = 0; i < keys.size(); ++i) {
      tree.put(keys.get(i), rids.get(i));
      assertEquals(Optional.of(rids.get(i)), tree.get(keys.get(i)));
    }

    for (int i = 0; i < keys.size(); ++i) {
      assertEquals(Optional.of(rids.get(i)), tree.get(keys.get(i)));
    }
  }

  @Test
  public void testWhiteBoxTest() throws BPlusTreeException, IOException {
    BPlusTree tree = getBPlusTree(Type.intType(), 1);
    assertEquals("()", tree.toSexp());

    // (4)
    tree.put(new IntDataBox(4), new RecordId(4, (short) 4));
    assertEquals("((4 (4 4)))", tree.toSexp());

    // (4 9)
    tree.put(new IntDataBox(9), new RecordId(9, (short) 9));
    assertEquals("((4 (4 4)) (9 (9 9)))", tree.toSexp());

    //   (6)
    //  /   \
    // (4) (6 9)
    tree.put(new IntDataBox(6), new RecordId(6, (short) 6));
    String l = "((4 (4 4)))";
    String r = "((6 (6 6)) (9 (9 9)))";
    assertEquals(String.format("(%s 6 %s)", l, r), tree.toSexp());

    //     (6)
    //    /   \
    // (2 4) (6 9)
    tree.put(new IntDataBox(2), new RecordId(2, (short) 2));
    l = "((2 (2 2)) (4 (4 4)))";
    r = "((6 (6 6)) (9 (9 9)))";
    assertEquals(String.format("(%s 6 %s)", l, r), tree.toSexp());

    //      (6 7)
    //     /  |  \
    // (2 4) (6) (7 9)
    tree.put(new IntDataBox(7), new RecordId(7, (short) 7));
    l = "((2 (2 2)) (4 (4 4)))";
    String m = "((6 (6 6)))";
    r = "((7 (7 7)) (9 (9 9)))";
    assertEquals(String.format("(%s 6 %s 7 %s)", l, m, r), tree.toSexp());

    //         (7)
    //        /   \
    //     (6)     (8)
    //    /   \   /   \
    // (2 4) (6) (7) (8 9)
    tree.put(new IntDataBox(8), new RecordId(8, (short) 8));
    String ll = "((2 (2 2)) (4 (4 4)))";
    String lr = "((6 (6 6)))";
    String rl = "((7 (7 7)))";
    String rr = "((8 (8 8)) (9 (9 9)))";
    l = String.format("(%s 6 %s)", ll, lr);
    r = String.format("(%s 8 %s)", rl, rr);
    assertEquals(String.format("(%s 7 %s)", l, r), tree.toSexp());
```

```
//           (7)
//          /   \
//     (3 6)       (8)
//   /   |   \    /   \
// (2) (3 4) (6) (7) (8 9)
tree.put(new IntDataBox(3), new RecordId(3, (short) 3));
ll = "((2 (2 2)))";
String lm = "((3 (3 3)) (4 (4 4)))";
lr = "((6 (6 6)))";
rl = "((7 (7 7)))";
rr = "((8 (8 8)) (9 (9 9)))";
l = String.format("(%s 3 %s 6 %s)", ll, lm, lr);
r = String.format("(%s 8 %s)", rl, rr);
assertEquals(String.format("(%s 7 %s)", l, r), tree.toSexp());

//           (4 7)
//          /  |  \
//   (3)      (6)       (8)
//  /   \    /   \    /   \
// (2) (3) (4 5) (6) (7) (8 9)
tree.put(new IntDataBox(5), new RecordId(5, (short) 5));
ll = "((2 (2 2)))";
lr = "((3 (3 3)))";
String ml = "((4 (4 4)) (5 (5 5)))";
String mr = "((6 (6 6)))";
rl = "((7 (7 7)))";
rr = "((8 (8 8)) (9 (9 9)))";
l = String.format("(%s 3 %s)", ll, lr);
m = String.format("(%s 6 %s)", ml, mr);
r = String.format("(%s 8 %s)", rl, rr);
assertEquals(String.format("(%s 4 %s 7 %s)", l, m, r), tree.toSexp());

//           (4 7)
//          /  |  \
//    (3)      (6)       (8)
//   /   \    /   \    /   \
// (1 2) (3) (4 5) (6) (7) (8 9)
tree.put(new IntDataBox(1), new RecordId(1, (short) 1));
ll = "((1 (1 1)) (2 (2 2)))";
lr = "((3 (3 3)))";
ml = "((4 (4 4)) (5 (5 5)))";
mr = "((6 (6 6)))";
rl = "((7 (7 7)))";
rr = "((8 (8 8)) (9 (9 9)))";
l = String.format("(%s 3 %s)", ll, lr);
m = String.format("(%s 6 %s)", ml, mr);
r = String.format("(%s 8 %s)", rl, rr);
assertEquals(String.format("(%s 4 %s 7 %s)", l, m, r), tree.toSexp());

//           (4 7)
//          /  |  \
//    (3)      (6)       (8)
//   /   \    /   \    /   \
// (  2) (3) (4 5) (6) (7) (8 9)
tree.remove(new IntDataBox(1));
ll = "((2 (2 2)))";
lr = "((3 (3 3)))";
ml = "((4 (4 4)) (5 (5 5)))";
mr = "((6 (6 6)))";
rl = "((7 (7 7)))";
rr = "((8 (8 8)) (9 (9 9)))";
l = String.format("(%s 3 %s)", ll, lr);
m = String.format("(%s 6 %s)", ml, mr);
r = String.format("(%s 8 %s)", rl, rr);
assertEquals(String.format("(%s 4 %s 7 %s)", l, m, r), tree.toSexp());

//           (4 7)
//          /  |  \
//    (3)      (6)       (8)
```

```
        //   /   \    /   \    /   \
        // (  2) (3) (4 5) (6) (7) (8  )
        tree.remove(new IntDataBox(9));
        ll = "((2 (2 2)))";
        lr = "((3 (3 3)))";
        ml = "((4 (4 4)) (5 (5 5)))";
        mr = "((6 (6 6)))";
        rl = "((7 (7 7)))";
        rr = "((8 (8 8)))";
        l = String.format("(%s 3 %s)", ll, lr);
        m = String.format("(%s 6 %s)", ml, mr);
        r = String.format("(%s 8 %s)", rl, rr);
        assertEquals(String.format("(%s 4 %s 7 %s)", l, m, r), tree.toSexp());

        //            (4 7)
        //           /  |  \
        //    (3)      (6)       (8)
        //   /   \    /   \    /   \
        // (  2) (3) (4 5) (  ) (7) (8  )
        tree.remove(new IntDataBox(6));
        ll = "((2 (2 2)))";
        lr = "((3 (3 3)))";
        ml = "((4 (4 4)) (5 (5 5)))";
        mr = "()";
        rl = "((7 (7 7)))";
        rr = "((8 (8 8)))";
        l = String.format("(%s 3 %s)", ll, lr);
        m = String.format("(%s 6 %s)", ml, mr);
        r = String.format("(%s 8 %s)", rl, rr);
        assertEquals(String.format("(%s 4 %s 7 %s)", l, m, r), tree.toSexp());

        //            (4 7)
        //           /  |  \
        //    (3)      (6)       (8)
        //   /   \    /   \    /   \
        // (  2) (3) (  5) (  ) (7) (8  )
        tree.remove(new IntDataBox(4));
        ll = "((2 (2 2)))";
        lr = "((3 (3 3)))";
        ml = "((5 (5 5)))";
        mr = "()";
        rl = "((7 (7 7)))";
        rr = "((8 (8 8)))";
        l = String.format("(%s 3 %s)", ll, lr);
        m = String.format("(%s 6 %s)", ml, mr);
        r = String.format("(%s 8 %s)", rl, rr);
        assertEquals(String.format("(%s 4 %s 7 %s)", l, m, r), tree.toSexp());

        //            (4 7)
        //           /  |  \
        //    (3)      (6)       (8)
        //   /   \    /   \    /   \
        // (    ) (3) (  5) (  ) (7) (8  )
        tree.remove(new IntDataBox(2));
        ll = "()";
        lr = "((3 (3 3)))";
        ml = "((5 (5 5)))";
        mr = "()";
        rl = "((7 (7 7)))";
        rr = "((8 (8 8)))";
        l = String.format("(%s 3 %s)", ll, lr);
        m = String.format("(%s 6 %s)", ml, mr);
        r = String.format("(%s 8 %s)", rl, rr);
        assertEquals(String.format("(%s 4 %s 7 %s)", l, m, r), tree.toSexp());

        //            (4 7)
        //           /  |  \
        //    (3)      (6)       (8)
        //   /   \    /   \    /   \
```

```
        // (   ) (3) (   ) ( ) (7) (8  )
        tree.remove(new IntDataBox(5));
        ll = "()";
        lr = "((3 (3 3)))";
        ml = "()";
        mr = "()";
        rl = "((7 (7 7)))";
        rr = "((8 (8 8)))";
        l = String.format("(%s 3 %s)", ll, lr);
        m = String.format("(%s 6 %s)", ml, mr);
        r = String.format("(%s 8 %s)", rl, rr);
        assertEquals(String.format("(%s 4 %s 7 %s)", l, m, r), tree.toSexp());

        //           (4 7)
        //          /  |  \
        //    (3)      (6)      (8)
        //   /   \    /   \    /   \
        // (   ) (3) (   ) ( ) ( ) (8  )
        tree.remove(new IntDataBox(7));
        ll = "()";
        lr = "((3 (3 3)))";
        ml = "()";
        mr = "()";
        rl = "()";
        rr = "((8 (8 8)))";
        l = String.format("(%s 3 %s)", ll, lr);
        m = String.format("(%s 6 %s)", ml, mr);
        r = String.format("(%s 8 %s)", rl, rr);
        assertEquals(String.format("(%s 4 %s 7 %s)", l, m, r), tree.toSexp());

        //           (4 7)
        //          /  |  \
        //    (3)      (6)      (8)
        //   /   \    /   \    /   \
        // (   ) ( ) (   ) ( ) ( ) (8  )
        tree.remove(new IntDataBox(3));
        ll = "()";
        lr = "()";
        ml = "()";
        mr = "()";
        rl = "()";
        rr = "((8 (8 8)))";
        l = String.format("(%s 3 %s)", ll, lr);
        m = String.format("(%s 6 %s)", ml, mr);
        r = String.format("(%s 8 %s)", rl, rr);
        assertEquals(String.format("(%s 4 %s 7 %s)", l, m, r), tree.toSexp());

        //           (4 7)
        //          /  |  \
        //    (3)      (6)      (8)
        //   /   \    /   \    /   \
        // (   ) ( ) (   ) ( ) ( ) (   )
        tree.remove(new IntDataBox(8));
        ll = "()";
        lr = "()";
        ml = "()";
        mr = "()";
        rl = "()";
        rr = "()";
        l = String.format("(%s 3 %s)", ll, lr);
        m = String.format("(%s 6 %s)", ml, mr);
        r = String.format("(%s 8 %s)", rl, rr);
        assertEquals(String.format("(%s 4 %s 7 %s)", l, m, r), tree.toSexp());
    }

    @Test
    public void testRandomPuts() throws BPlusTreeException, IOException {
        List<DataBox> keys = new ArrayList<>();
        List<RecordId> rids = new ArrayList<>();
```

```
    List<RecordId> sortedRids = new ArrayList<>();
    for (int i = 0; i < 1000; ++i) {
      keys.add(new IntDataBox(i));
      rids.add(new RecordId(i, (short) i));
      sortedRids.add(new RecordId(i, (short) i));
    }

    // Try trees with different orders.
    for (int d = 2; d < 5; ++d) {
      // Try trees with different insertion orders.
      for (int n = 0; n < 2; ++n) {
        Collections.shuffle(keys, new Random(42));
        Collections.shuffle(rids, new Random(42));

        // Insert all the keys.
        BPlusTree tree = getBPlusTree(Type.intType(), d);
        for (int i = 0; i < keys.size(); ++i) {
          tree.put(keys.get(i), rids.get(i));
        }

        // Test get.
        for (int i = 0; i < keys.size(); ++i) {
          assertEquals(Optional.of(rids.get(i)), tree.get(keys.get(i)));
        }

        // Test scanAll.
        assertEquals(sortedRids, iteratorToList(tree.scanAll()));

        // Test scanGreaterEqual.
        for (int i = 0; i < keys.size(); i += 100) {
          Iterator<RecordId> actual = tree.scanGreaterEqual(new IntDataBox(i));
          List<RecordId> expected = sortedRids.subList(i, sortedRids.size());
          assertEquals(expected, iteratorToList(actual));
        }

        // Load the tree from disk.
        BPlusTree fromDisk = new BPlusTree(file.getAbsolutePath());
        assertEquals(sortedRids, iteratorToList(fromDisk.scanAll()));

        // Test remove.
        Collections.shuffle(keys, new Random(42));
        Collections.shuffle(rids, new Random(42));
        for (DataBox key : keys) {
          fromDisk.remove(key);
          assertEquals(Optional.empty(), fromDisk.get(key));
        }
      }
    }
  }

  // HIDDEN
  @Test
  public void testRepeatedInsertsAndRemoves()
      throws BPlusTreeException, IOException {
    BPlusTree tree = getBPlusTree(Type.intType(), 4);

    // Insert [0, 200).
    for (int i = 0; i < 200; ++i) {
      tree.put(new IntDataBox(i), new RecordId(i, (short) i));
    }

    // Delete [100, 200).
    for (int i = 100; i < 200; ++i) {
      tree.remove(new IntDataBox(i));
    }

    // Insert [150, 300).
    for (int i = 150; i < 300; ++i) {
      tree.put(new IntDataBox(i), new RecordId(i, (short) i));
```

```
    }

    // Delete [250, 300).
    for (int i = 250; i < 300; ++i) {
      tree.remove(new IntDataBox(i));
    }

    // Add [100, 150]
    for (int i = 100; i < 150; ++i) {
      tree.put(new IntDataBox(i), new RecordId(i, (short) i));
    }

    // Add [250, 300]
    for (int i = 250; i < 300; ++i) {
      tree.put(new IntDataBox(i), new RecordId(i, (short) i));
    }

    // Range [0, 300) should be full.
    List<RecordId> rids = new ArrayList<>();
    for (int i = 0; i < 300; ++i) {
      rids.add(new RecordId(i, (short) i));
    }
    assertEquals(rids, iteratorToList(tree.scanAll()));
  }

  @Test
  public void testMaxOrder() {
    // Note that this white box test depend critically on the implementation
    // of toBytes and includes a lot of magic numbers that won't make sense
    // unless you read toBytes.
    assertEquals(4, Type.intType().getSizeInBytes());
    assertEquals(6, RecordId.getSizeInBytes());
    int pageSizeInBytes = 100;
    Type keySchema = Type.intType();
    assertEquals(4, LeafNode.maxOrder(pageSizeInBytes, keySchema));
    assertEquals(5, InnerNode.maxOrder(pageSizeInBytes, keySchema));
    assertEquals(4, BPlusTree.maxOrder(pageSizeInBytes, keySchema));
  }

}
```