Branch: master ▾ DB2 / src / main / java / index / BPlusNode.java

Find file | Copy path

dreamlegends init add all files 2c520b8 7 days ago

1 contributor

244 lines (231 sloc) 10 KB

Raw | Blame | History

```
package index;

import common.Pair;
import databox.DataBox;
import io.Page;
import table.RecordId;

import java.nio.ByteBuffer;
import java.util.Optional;


/**
 * An inner node or a leaf node. See InnerNode and LeafNode for more
 * information.
 */
abstract class BPlusNode {
  // Core API ////////////////////////////////////////////////////////////////
  /**
   * n.get(k) returns the leaf node on which k may reside when queried from n.
   * For example, consider the following B+ tree (for brevity, only keys are
   * shown; record ids are ommitted).
   *
   *                              inner
   *                              +----+----+----+----+
   *                              | 10 | 20 |    |    |
   *                              +----+----+----+----+
   *                             /     |     \
   *                        ____/      |      \____
   *                       /           |           \
   *   +----+----+----+----+  +----+----+----+----+  +----+----+----+----+
   *   |  1 |  2 |  3 |    |->| 11 | 12 | 13 |    |->| 21 | 22 | 23 |    |
   *   +----+----+----+----+  +----+----+----+----+  +----+----+----+----+
   *   leaf0                  leaf1                  leaf2
   *
   * inner.get(x) should return
   *
   *   - leaf0 when x < 10,
   *   - leaf1 when 10 <= x < 20, and
   *   - leaf2 when x >= 20.
   *
   * Note that inner.get(4) would return leaf0 even though leaf0 doesn't
   * actually contain 4.
   */
  public abstract LeafNode get(DataBox key);
```

```
/**
 * n.getLeftmostLeaf() returns the leftmost leaf in the subtree rooted by n.
 * In the example above, inner.getLeftmostLeaf() would return leaf0, and
 * leaf1.getLeftmostLeaf() would return leaf1.
 */
public abstract LeafNode getLeftmostLeaf();

/**
 * n.put(k, r) inserts the pair (k, r) into the subtree rooted by n. There
 * are two cases to consider:
 *
 *   Case 1: If inserting the pair (k, r) does NOT cause n to overflow, then
 *           Optional.empty() is returned.
 *   Case 2: If inserting the pair (k, r) does cause the node n to overflow,
 *           then n is split into a left and right node (described more
 *           below) and a pair (split_key, right_node_page_num) is returned
 *           where right_node_page_num is the page number of the newly
 *           created right node, and the value of split_key depends on
 *           whether n is an inner node or a leaf node (described more below).
 *
 * Now we explain how to split nodes and which split keys to return. Let's
 * take a look at an example. Consider inserting the key 4 into the example
 * tree above. No nodes overflow (i.e. we always hit case 1). The tree then
 * looks like this:
 *
 *                              inner
 *                              +----+----+----+----+
 *                              | 10 | 20 |    |    |
 *                              +----+----+----+----+
 *                             /     |     \
 *                        ____/      |      \____
 *                       /           |           \
 *   +----+----+----+----+  +----+----+----+----+  +----+----+----+----+
 *   |  1 |  2 |  3 |  4 |->| 11 | 12 | 13 |    |->| 21 | 22 | 23 |    |
 *   +----+----+----+----+  +----+----+----+----+  +----+----+----+----+
 *   leaf0                  leaf1                  leaf2
 *
 * Now let's insert key 5 into the tree. Now, leaf0 overflows and creates a
 * new right sibling leaf3. d entries remain in the left node; d + 1 entries
 * are moved to the right node. DO NOT REDISTRIBUTE ENTRIES ANY OTHER WAY. In
 * our example, leaf0 and leaf3 would look like this:
 *
 *   +----+----+----+----+  +----+----+----+----+
 *   |  1 |  2 |    |    |->|  3 |  4 |  5 |    |
 *   +----+----+----+----+  +----+----+----+----+
 *   leaf0                  leaf3
 *
 * When a leaf splits, it returns the first entry in the right node as the
 * split key. In this example, 3 is the split key. After leaf0 splits, inner
 * inserts the new key and child pointer into itself and hits case 0 (i.e. it
 * does not overflow). The tree looks like this:
 *
 *                          inner
 *                          +--+--+--+--+
 *                          | 3|10|20|  |
 *                          +--+--+--+--+
 *                         /   |  |   \
 *                _______/    |  |    \_________
 *               /            |  \              \
 *   +--+--+--+--+  +--+--+--+--+  +--+--+--+--+  +--+--+--+--+
 *   | 1| 2|  |  |->| 3| 4| 5| 6|->|11|12|13|  |->|21|22|23|  |
 *   +--+--+--+--+  +--+--+--+--+  +--+--+--+--+  +--+--+--+--+
 *   leaf0          leaf3          leaf1          leaf2
 *
 * When an inner node splits, the first d entries are kept in the left node
 * and the last d entries are moved to the right node. The middle entry is
 * moved (not copied) up as the split key. For example, we would split the
 * following order 2 inner node
```

```
 *
 *   +---+---+---+---+
 *   | 1 | 2 | 3 | 4 | 5
 *   +---+---+---+---+
 *
 * into the following two inner nodes
 *
 *   +---+---+---+---+  +---+---+---+---+
 *   | 1 | 2 |   |   |  | 4 | 5 |   |   |
 *   +---+---+---+---+  +---+---+---+---+
 *
 * with a split key of 3.
 *
 * DO NOT redistribute entries in any other way besides what we have
 * described. For example, do not move entries between nodes to avoid
 * splitting.
 *
 * Our B+ trees do not support duplicate entries with the same key. If a
 * duplicate key is inserted, the tree is left unchanged and an exception is
 * raised.
 */
public abstract Optional<Pair<DataBox, Integer>> put(DataBox key, RecordId rid)
  throws BPlusTreeException;

/**
 * n.remove(k) removes the key k and its corresponding record id from the
 * subtree rooted by n, or does nothing if the key k is not in the subtree.
 * REMOVE SHOULD NOT REBALANCE THE TREE. Simply delete the key and
 * corresponding record id. For example, running inner.remove(2) on the
 * example tree above would produce the following tree.
 *
 *                              inner
 *                              +----+----+----+----+
 *                              | 10 | 20 |    |    |
 *                              +----+----+----+----+
 *                             /     |     \
 *                        ____/      |      \____
 *                       /           |           \
 *   +----+----+----+----+  +----+----+----+----+  +----+----+----+----+
 *   |  1 |  3 |    |    |->| 11 | 12 | 13 |    |->| 21 | 22 | 23 |    |
 *   +----+----+----+----+  +----+----+----+----+  +----+----+----+----+
 *   leaf0                  leaf1                  leaf2
 *
 * Running inner.remove(1) on this tree would produce the following tree:
 *
 *                              inner
 *                              +----+----+----+----+
 *                              | 10 | 20 |    |    |
 *                              +----+----+----+----+
 *                             /     |     \
 *                        ____/      |      \____
 *                       /           |           \
 *   +----+----+----+----+  +----+----+----+----+  +----+----+----+----+
 *   |  3 |    |    |    |->| 11 | 12 | 13 |    |->| 21 | 22 | 23 |    |
 *   +----+----+----+----+  +----+----+----+----+  +----+----+----+----+
 *   leaf0                  leaf1                  leaf2
 *
 * Running inner.remove(3) would then produce the following tree:
 *
 *                              inner
 *                              +----+----+----+----+
 *                              | 10 | 20 |    |    |
 *                              +----+----+----+----+
 *                             /     |     \
 *                        ____/      |      \____
 *                       /           |           \
 *   +----+----+----+----+  +----+----+----+----+  +----+----+----+----+
 *   |    |    |    |    |->| 11 | 12 | 13 |    |->| 21 | 22 | 23 |    |
 *   +----+----+----+----+  +----+----+----+----+  +----+----+----+----+
```

```
     *   leaf0                 leaf1                 leaf2
     *
     * Again, do NOT rebalance the tree.
     */
    public abstract void remove(DataBox key);

    // Helpers /////////////////////////////////////////////////////////////////
    /** Get the page on which this node is persisted. */
    abstract Page getPage();

    // Pretty Printing /////////////////////////////////////////////////////////
    /**
     * S-expressions (or sexps) are a compact way of encoding nested tree-like
     * structures (sort of like how JSON is a way of encoding nested dictionaries
     * and lists). n.toSexp() returns an sexp encoding of the subtree rooted by
     * n. For example, the following tree:
     *
     *                      +---+
     *                      | 3 |
     *                      +---+
     *                     /     \
     *   +---------+---------+  +---------+---------+
     *   | 1:(1 1) | 2:(2 2) |  | 3:(3 3) | 4:(4 4) |
     *   +---------+---------+  +---------+---------+
     *
     * has the following sexp
     *
     *   (((1 (1 1)) (2 (2 2))) 3 ((3 (3 3)) (4 (4 4))))
     *
     * Here, (1 (1 1)) represents the mapping from key 1 to record id (1, 1).
     */
    public abstract String toSexp();

    /**
     * n.toDot() returns a fragment of a DOT file that draws the subtree rooted
     * at n.
     */
    public abstract String toDot();

    // Serialization ///////////////////////////////////////////////////////////
    /** n.toBytes() serializes n. */
    public abstract byte[] toBytes();

    /**
     * BPlusNode.fromBytes(m, p) loads a BPlusNode from page p of
     * meta.getAllocator().
     */
    public static BPlusNode fromBytes(BPlusTreeMetadata metadata, int pageNum) {
      Page p = metadata.getAllocator().fetchPage(pageNum);
      ByteBuffer buf = p.getByteBuffer();
      byte b = buf.get();
      if (b == 1) {
        return LeafNode.fromBytes(metadata, pageNum);
      } else if (b == 0) {
        return InnerNode.fromBytes(metadata, pageNum);
      } else {
        String msg = String.format("Unexpected byte %b.", b);
        throw new IllegalArgumentException(msg);
      }
    }
  }
```