**Tests**

● test1()

○ Create a relation with tuples valued 0 to relationSize and perform index tests

on attributes of all three types (int, double, string)

○ Test if the output amount matches the input amount when scanning

● test2()

○ Create a relation with tuples valued 0 to relationSize in reverse order and perform index tests

on attributes of all three types (int, double, string)

○ Test if the output amount matches the input amount when scanning

● test3()

○ Create a relation with tuples valued 0 to relationSize in random order and perform index tests

on attributes of all three types (int, double, string)

○ Test if the output amount matches the input amount when scanning

● test4()

○ This test create a relation with tuples valued 0 to the given number in random order

Tests for randomly given size of 10000

● test5()

○ Test for forward inserting with no split on root

■ Create a relation with tuples valued 0 to the given number

Test for forward inserting with no split on root

● test6()

○ Create a relation with tuples valued in the given range

○ In this case, there will be negative key values

○ Test for backward inserting with given size

● test7()

○ Create a relation with tuples valued 0 to the given size in reverse order

○ Test for backward inserting with given size

● test8()

○ Create a relation with tuples valued 0 to the given size

○ The ​relationSize​ is set to be 683 to test splitting on non-leaf node

○ test the root split

● error test()

○ Test whether ​ScanNotInitializedException​ will be thrown if ​endScan​ is

called before ​startScan

○ Test whether ​ScanNotInitializedException​ will be thrown if scanNext is

called before ​startScan

○ Test whether ​BadOpcodesException​ will be thrown if ​lowOp​ is ​LTE, ​highOp​ is ​GTE, or ​lowValInt >highValInt

We test for the correctness of splitting nodes using two tests: to test if the root would split if no split is needed; to test if the root would split into one non-leaf node and two leaf nodes when inserting into a full root. The former checks the correctness of the condition for splitting, and the latter tests one of the complicated processes involving splitting leaf, inserting into leaf, creating new root, updating header page information, creating mid-pair to move up, and inserting into non-leaf.

We test for the correctness, reliability, and efficiency by inserting different sizes of relations. One extreme case is to test on an empty tree to check the functionality of the constructor. Inserting different sizes of data and using different orders is to test the correctness and reliability in different complicated cases. Among these insertions, we include a large data test, where one million data is inserted. To insert one million data using CreateRelationForward function, whereas inserting one million data into B+ tree is also very fast. This comparison demonstrates the efficiency of B+ tree. Additionally, we test for the reflexivity by inserting both positive and negative keys at the same time. This test is to check if our implementation could accommodate a wider range of cases. In this project, our implementation of the B+ tree is correct and efficient