Report – Jerry Liu 20186440

**Description of data**

A large dataset was desired for this project and the actual application of the B+ tree, as each internal node typically has many children. For this, an online dataset was found…

υ Source, schema, number of tuples υ Order of the B+ tree, what attributes are used for the key of the B+ tree

**Description of B+ tree**

The B+ tree follows the standard implementation of a B+ tree, with a few customizations due to the unique structure of the keys and values. In addition to the regular B+ tree characteristics, this B+ tree has keys with two attributes, a primary attribute for comparison and a secondary attribute for comparison. This means that when ordering keys and performing searches, the primary key needs to be compared followed by the secondary key if the primary key values are equal for exact precision.

Furthermore, the other main customization was that most B+ trees seem to have leaf nodes with single key value pairs. To follow the diagrams given in the original assignment, the leaf nodes have no repeated keys, and if there are multiple entries with identical keys (both primary and secondary), then they are put together in an ordered list matching the index of the specified key.

**Running the program**

The program was written using Notepad++ as a .py file. To run the program, any command line editor with Python should be able to be used. For testing during development, an Anaconda 3.6 Python terminal was used, and once reaching the directory of the files, the command “python Bplustree.py” needed to be executed and the program would start automatically

**Testing the 6 Operations**

Any time an operation is to be select, the program will ask “What do you want to do: “ and expect a single integer from 1-7 as the input. Following this will be the specific input for each of the 6 operations and exit.

To test load, a provided data file in a .csv format is given. This .csv file can be opened in Microsoft Excel or a similar program to see what the real contents are. In the actual program, once it has run the first input should be a “1” followed by the file name of the data and start and end tids. This means 4 inputs in total after the program has begun running. The data used for testing is included as a file called “SampleData.csv”. The tids start at 1, so there is no entry that belongs to an tid of 0, which could lead to errors if referenced.

There is no error check for tids that are out of bounds or invalid, so it is assumed that the start and end are both valid logically and programmatically. The tree will automatically be build. The load function not only reads the entire csv and stores it in a table for retrieval and reference, but it inserts the tids from the start to the end as an initialization of the tree. Due to this, there is no way to start with an empty tree for the testing of the other functions, but an empty tree could be created by deleting all the entries. This also means load must be the first operation that is performed with the program every execution. The print operation can be used to output the current state of the tree and matches the expected output from the loading.

To test print, once data has been loaded and other commands have been inputted, the user should have an expected result for the tree and what it looks like based on their inputs. Testing print simply requires typing “2” and the current state of the tree will be printed in a breadth first manner. Comparison of the output and the expected tree can easily be done by looking at the print result layer by layer.

To test insert, input “3” as the choice followed by the tid that is desired to be inserted. A print message will be given when the tid is successfully added, or a print message will be given if the tid already exists in the tree. A print operation following the insertion can be used to test that it worked successfully and didn’t cause any problems in the tree.

To test delete, input “4” as the choice followed by the tid that is desired to be deleted. If the deletion is successful, a print message will be given that it was successfully deleted, otherwise it will print that it was not successful (most likely due to the tid not existing in the tree). A print operation following the deletion can be used to test that the deletion worked properly and as expected if the tree needed to be rearranged.

To test search, enter “5” as the operation input and it will ask for the value of the primary and secondary key attributes the user would like to use in the search. The program will combine both these attributes into the single key that it is looking for in the tree, and perform a search. It will output the tids that match exactly the primary and secondary key attributes provided by the user. To test this function, a tree with keys that match and don’t match the user input can be created, and knowing what tids should be expected from a specific key search, the search operation can be run and seen if it returns the same results.

To test range search, enter “6” as the operation input and then the program will ask for the primary and secondary keys twice. This is to generate the minimum and maximum boundary keys for the range search. There is no error checking for logic or programmatically correct inputs. Once the inputs are given, the program will search through the tree and display all valid keys with their values. To test that it is correct, a print of the tree can be made and a manual check of which keys meet the range can be compared to the one generated by the program.