# COP 5536 Programming Project Report

Name: Rong Fan    UFID: 61381351    UF Email Account: rfan@ufl.edu

## I. Introduction

In this project, I have written a program to build a B+ Tree with the following functions: initialization, insertion, search and range search. In the next section, I will introduce the structure of my program.

## II. Structure of Program

I have created 3 classes in total to build the B+ tree with above functions, and a tree search class to implement the B+ tree with given text file.

### *1) MyPair.java*

This class has build a structure called MyPair, which is used to store pairs in the form (key, value).

| MyPair.java | | |
|---|---|---|
| | *Variables* | |
| Double | key | Store information of key |
| String | value | Store information of value |
| | *Methods* | |
| name | return type | function |
| key( ) | Double | Obtain the key information from the pair. |
| value( ) | String | Obtain the value information from the pair. |

## 2) *Node.java*

This class has build the node structure of the B+ tree.

| **Node.java** | | |
|---|---|---|
| | *Variables* | |
| boolean | isLeaf | true, only if the node is a leaf |
| boolean | isRoot | true, only if the node is a root |
| int | n | indicates the number of keys within a leaf node |
| boolean | duplicate | true only when the leaf has pairs with duplicate key |
| node | parent | pointer to its parent (empty when the node is a root) |
| List<Node> | children | pointer to its children (empty when the node is a leaf) |
| Node | prev | pointer to its right sibling (empty when the node is not a leaf) |
| Node | next | pointer to its left sibling (empty when the node is not a leaf) |
| List<MyPair> | pairs | store pairs in the node |
| | *Methods* | |
| name | return type | function |
| Node(boolean isLeaf) | void | Initialize a node with isLeaf information |
| Node(boolean isRoot) | void | Initialize a node with isLeaf and isRoot information |

The initialization methods could initialize a node with isLeaf/isRoot information, create an empty ArrayList for pairs, set n = 0, duplicate = false, parent = null, and create an empty ArrayList for children if isLeaf is not true.

## 3) *BPlusTree.java*

| **BPlusTree.java** | | |
|---|---|---|
| | *Variables* | |
| Node | root | root of the B+ tree |
| int | degree | degree of the B+ tree |
| Node | head | the 1st node of tree's leaves |
| Node | tail | the last node of tree's leaves |
| | *Methods* | |
| name | return type | function |
| Initialize(int m) | void | Initialize a B+ tree node with degree m, create a new node(isLeaf: true, isRoot: true) called root, set head as root |
| Search(Double key) | ArrayList<String> | Returns all values associated with a specific key. |
| Search(Double key1, Double key2) | ArrayList<MyPair > | Returns all key values pairs such that key1 <= key <= key2. |
| traverseLeaf(Node leaf, ArrayList<MyPair> result, Double key1, Double key2) | void | Traverse all the leaf nodes which contain keys in range of (key1, key2), start from the leaf with key1, stops until the first key of next leaf is larger than key2, or it is the last leaf of the tree. |
| searchLeafNode(Node root, Double key) | Node | Returns the leaf node which might contain value associated with key |
| Insert (Double key, String value) | void | Insert a pair of (key, value) into the tree |
| copy2Nodes (Node left, Node right, Node leaf) | void | Copy the pairs of a leaf node into two new leaf nodes |
| updateInsert(Node parent) | void | Update the parent node of a node after insertion |
| insertIntoLeafNode (Node leaf, Double key, String value) | void | insert a new pair(key, value) into a given leaf node, keep the pairs in the node sorted by their keys. |

In the following paragraphs I am going explain the above methods into detail.

*searchLeafNode(Node root, Double key)* : start with the root node if the root is a leaf, return the root node; Else, using binary search to compare the given key with keys of pairs in the node, find the correct children of this node to continue the searchLeafNode process, until a leaf node is reached.

*Search(Double key)*: If key or root of the tree is null, return null; Else, create an empty ArrayList to store results, use searchLeafNode(root, key) to find the leaf node which might contain the pair with this key, loop the pairs inside the leaf and compare the keys with the given key, if there is a match, add the pair's value into the result. Finally, return the result.

*traverseLeaf(Node leaf, ArrayList<MyPair> result, Double key1, Double key2)*: Start with a given leaf node, if the key of the first pair in the node is larger than key2, return the result; Else, loop the pairs inside the leaf node, if key1 <= key <= key2, add the pair with this key to result. If the leaf node is not the tail, repeat the process of traverseLeaf with its right sibling.

*Search(Double key1, Double key2)*: If key1 or key2 or root of the tree is null, return Null. Otherwise, first, using searchLeafNode(root, key1) to find the leaf node which might contain pairs with key >= key1, name the node as leaf. Then create an empty ArrayList names result to store output pairs. Next, using traverseLeaf(leaf, result, key1, key2) to traverse the leaves of the tree and add pairs with key within range [key1, key2] into the result. Finally, return the result.

*insertIntoLeafNode(Node leaf, Double key, String value)*: Use binary search to compare the given key with keys of pairs inside the leaf node. Find the right place to insert the key, create a new pair with (key, value), add the pair into the pairs of the leaf node. During binary search, if a duplicate key is found, change the variable — "duplicate" of this leaf node to true.

*copy2Nodes(Node left, Node right, Node leaf)*: Find the split point of the old leaf node, which is the pair with (ceil(m / 2))th smallest key in the leaf. Find the index of the pair with (ceil(m / 2))th smallest key in the leaf's pairs ArrayList while looping the leaf node, add pairs with smaller index to the left node, add pairs with equal or bigger index to the right node. Set the pairs of leaf node to be null.

*updateInsert(Node parent)*: If the size of children of the parent node is not bigger than degree of the tree, done. Else, create two index nodes left node and right node with Node(isLeaf: false). Find the split point of the pairs which is the pair with (ceil(m / 2))th smallest key, add nodes in the children ArrayList with index smaller than or equal to ceil(m / 2) to the left node's children ArrayList, others to the right node. Set children of parent node to be null. Then split the pairs, add pairs in the pairs ArrayList with index smaller than to ceil(m / 2) - 1 to the left node's pairs ArrayList, add pairs in the pairs ArrayList with index larger than to ceil(m / 2) - 1 to the right node's pairs ArrayList. If the node parent is the root, create a new node to be a root, add left node and right node to its children, add pair in the pairs ArrayList of parent node with index ceil(m / 2) - 1 to its pairs, done; Else, get the index of parent node within its parent, add left node and right node to be its parent's children at that index and the index +1, add pair in the pairs ArrayList of parent node with index ceil(m / 2) - 1 to its parent's pairs at that index. Repeat updateInsert with the grandparent node.

<u>*Insert(Double key, String value)*</u>: First, use searchLeafNode(key) to find a leaf to add the new pair (key, value). Second, use insertIntoLeafNode(leaf, key, value) to add this pair into the leaf. Third, update the number of different keys in the leaf, if duplicate is not true, n = n + 1. If n >= degree, split the node, else return. When splitting, create two leaf node, link these node to be sibling (update the doubly linked list), use copy2Node to copy the information into these nodes. If the leaf is the root node, create a new root node, add left and right nodes to its children, add the first pair of right node to its pairs, link the three nodes, done. If the leaf is not the root node, add left and right nodes to its parent's children with leaf's index, add the first pair of right node to its parent's pairs with leaf's index, link the three nodes. Remove leaf from its parent's children. Using updateInsert to update its parent node, done.

***4) treesearch.java***

| **BPlusTree.java** | | |
|---|---|---|
| | *Variables* | |
| | *Methods* | |
| name | return type | function |
| main(String[] args) | void | Reads and implements instructions from text file. |
| getCommands(String fileName) | List<String> | Reads the text file, converts all the instructions as String and store them in a list |
| processCommandLine(String[] commandArgs, BPlusTree tree, PrinterWriter writer) | void | Process the instruction. |

<u>*getCommands(String fileName)*</u>: First, create an ArrayList commandLine to store commands, create a scanner, use scanner to read the lines from the file, add each line to the ArrayList.

<u>*processCommandLine(String[] commandArgs, BPlusTree tree, PrinterWriter writer)*</u>: Compare the first element of commandArgs with String "Search", "Insert", then implement the proper method with the BPlusTree tree.

<u>*main(String[] args)*</u>: First, read the file name of the text file. Then use getCommands to get a list of command lines. Create a new BPlusTree, and a PrinterWriter. Next, use processCommandLine to process the commands line by line. Finally, close the writer.