COP5536: Advanced Data Structures, Fall 2017

Project Report

Vineeth Chennapalli | 31242465
vchennapalli@ufl.edu

COMPILING INSTRUCTIONS

The project has been written in java. The programs were compiled using JDK 9.0.1+11 with javac compiler.The unzipped folder contains the project files (.java files), the makefile and this project report.

The compilation can be done by running the following command in the present directory:
$ make

To run the program with the input from a text file "filename", use the following command:
$ java searchtree filename

The output is written into a text file with the name "output_file.txt".

Note: The output is always written into the same file. When different files are executed, the output is concatenated to the previous output.

OPERATIONS SUPPORTED

The set of programs support the following instructions:

1. Initialization of B+ Tree: Initializes a B+ tree where each node in the tree will have a degree that's mentioned in input.
2. Insertion of a <key, value> Pair: Inserts the pair into the tree.
3. Exact Key Search: Returns all the values associated with the key. Returns null if none exist.
4. Range Search: Returns (all <key, value> pairs in the in the non-decreasing key order such that lowerKey <= key <= higherKey, where lowerKey is the lower limit of the range and higherKey is the upper limit of the range. Returns null if none exist in this range.

DATA STRUCTURES USED (AFTER MANUAL CREATION WITH BASIC STRUCTURES)

This project makes use of ArrayLists to create a couple of data structures to aid in its development.

1. The first one is a structure called **Pair**, which basically holds a key (of *float* type) and an ArrayList of values (of *String* type) for the leaf nodes. The ArrayList of values is helpful when one stores multiple values for the same key. They are inserted into values list corresponding to the same key (instead a duplicate key value pair insertion).
2. The second is a **stack** data structure which pushes the nodes into an array list while traversing down and pops them on the way up as and when needed. This also makes use of ArrayList data structure.

PROGRAM STRUCTURE

The project has seven classes: "BPlusTree.java", "Node.java", "IndexNode.java", "LeafNode.java", "Pair.java", "Stack.java" and "searchtree.java". "searchtree.java" contains the main function.

The top-level description of each class and the function prototypes of each class are as follows:

Pair.java

The instances of this class form the fundamental data type that's used in the leaf nodes.

The class variables of this class are:

- private float key - a placeholder to store a key.
- private ArrayList<String> values - an ArrayList of Strings that stores the value(s) of the respective key.

The function prototypes of this class are as follows:

- public Pair(float key, String value) - constructor of the class. Is created whenever a new key and its value are inserted into the tree.
- public float getKey() - returns key of the pair.
- public void setKey(float key) - sets the key for the pair.
- public ArrayList<String> getValues() - returns values of the pair.
- public void appendValue(String value) - appends the new value for the respective key when a pair with already existing key is inserted.

Stack.java

An instance of this class keeps track of the nodes through which the traversal is made from the root to a leaf node while inserting a key value pair.

It contains the following class variable:

- private ArrayList<Node> stack - An ArrayList to keep track of the nodes.

The function prototypes of this class are as follows:

- public Stack() - constructor of the class. It initializes the ArrayList.
- public boolean isEmpty() - checks if the stack is empty.
- public Node pop() - pops the top element from the stack.
- public void reset() - resets the stack whenever there isn't a need to follow all the way to the top. The traversal is stopped when the function comes across a node that doesn't have to be split.
- public int getSize() - returns the size of the stack.
- public void push(Node newElement) - pushes a new Node onto the stack.

searchtree.java

searchtree.java is the interface class that reads the commands from the input file, executes them on a b+ tree object and writes the values it returns into *output_file.txt.*

The function prototypes of this class are as follows:

- public static void main(String[] args) - main function. Takes the input file as the argument. An instance of tree search class is created and calls the readCommands function to read the commands from input file.
- private void readCommands(File inputFile) - reads the commands from the input file and calls execute function.
- private void initializeBPlusTree(int degree) - creates a b+ tree and initializes it.
- private void executeCommand(String command) - extracts the command from the string input and executes it. If anything is returned, calls function to write into output file.
- private String convertToString(ArrayList<Pair> pairs) - concatenates all multiple keys and corresponding values into one string returns null if no keys exists in the interval.
- private String concatenateHelper(float key, String value) - helper function for concatenation of all values of one key.
- private String convertToString(Pair pair) - concatenates all values corresponding to given key returns null if given key doesn't exist.
- private void writeOutput(String output) - writes the output into output_file.txt file.

BPlusTree.java

This is the class that creates the nodes (two types: leaf node and index node) as and when needed, executes the commands by inserting/searching and returns appropriate output (if any).

The function prototypes of this class are as follows:

- public BPlusTree(int degree) - constructor of b+ tree. Initializes root to leaf node.
- public void insertPair(float key, String value) - top level function that takes care of insertion and adjustment of tree (if needed).

- private Node topDownInsert(float key, String value) - inserts the <key, value> from the top of the tree. Keeps track of the path by copying the traversed path into a stack. Returns the leaf node into which insertion was made.
- Private void bottomUpAdjust(Node node) - adjusts the tree from the bottom to top to maintain capacity conditions of nodes (if adjustment is needed). Makes use of the data stacked in topDownInsert to traverse to the top.
- public Pair searchPair - top level function that takes care of searching the values(s) corresponding to the given key. Returns the pair if found or null otherwise.
- public ArrayList<Pair> searchPair(float key1, float key2) - top level function that takes care of searching the keys and corresponding values in the requested range. Returns an ArrayList of pairs if found or null otherwise.

Node.java

This is the super class that has two children classes: IndexNode and LeafNode.
It contains a list of all the abstract classes that are part of both the children classes.

The common class variables are described below:

- protected int degree - placeholder to store the node's degree value.
- protected boolean hasChild - boolean that stores whether the node has children or not.

The function prototypes of this class are as follows:

- abstract Node insertDataPair(float key, String value) - inserts the given data pair into the tree.
- abstract Pair search(float key) - searches the tree for a pair with the requested key and returns it. Returns null if such key doesn't exist in the tree.
- abstract ArrayList<Pair> search(float key1, float key2) - searches the tree for the requested range and returns all key value pairs in ArrayList form if any are found. Returns an empty ArrayList if none are found.
- abstract boolean isFull() - checks if the node's capacity overflowed or not. This is used while traversing to the root from the leaf after insertion to adjust the tree if needed.
- abstract Node splitNode() - Splits the node into two when a node has is holding number of keys that's equal to the degree of the tree. Also, creates a new root node if one is needed.
- abstract boolean insertChild(float key, Node rightChild) - Inserts a key and a childPointer into the parent node whenever one of the child overflows.
- abstract float getMiddleKey() - returns the middle key of the node that has to be inserted in the parent. In case of leaf node, it is copied and in case of index node, it is moved up.

IndexNode.java

This class inherits the properties of the Node class and has the properties that are specific to the interior/index nodes of a B+ Tree.

It contains the following class variables:

- private ArrayList<Float> indices - an ArrayList to keep track of the indices in the node.

- **private ArrayList<Node> childPointers** - an ArrayList to keep track of the pointers to the child nodes of the present node.

The function prototypes of this class are given below:

- **public IndexNode(int degree, float key, Node leftChild, Node rightChild)** - constructor for the creation of a root index node.
- **public IndexNode(int degree)** - constructor for the creation of a non-root index node.
- **public void insertPair(float key, String value)** - top level function that takes care of insertion and adjustment of tree (if needed).
- **public boolean insertChild(float key, Node rightChild)** - inserts a new index and its right child pointer at appropriate position by traversing through the indices.
- **public Node insertDataPair(float key, String value)** - passes the data pair recursively into the appropriate child node.
- **public Pair search(float key)** - passes the search key recursively into the appropriate child node at each level.
- **public ArrayList<Pair> search(float key1, float key2)** - passes the key1 (lower limit of range search) recursively into the child
- **public Node findChildPointer(float key)** - finds and returns the appropriate child pointer for the given key. This is used to traverse down the tree while inserting and searching.
- **public boolean isFull()** - checks if the node's capacity overflowed or not. This is used while traversing from the leaf node to the root. Whenever the capacity has reached the degree of the node, it returns true so that the node split occurs.

LeafNode.java

This class inherits the properties of the node class and has additional properties that are specific to the leaf nodes of a B+ tree.

It contains the following class variables:

- **private ArrayList<Pair> pairs** - an ArrayList that stores the pairs of keys and the corresponding values.
- **private LeafNode next** - A pointer to the next leaf node. This is useful while doing a range query as one just needed to traverse through using the pointer if needed.
- **private LeafNode previous** - A pointer to the previous leaf node.

The function prototypes of this class are given below:

- **public LeafNode(int degree)** - constructor to initialize the leaf node.
- **private void setDegree(int degree)** - sets the degree of the node.
- **public boolean isFull()** - checks if the node's capacity overflowed. When the node reaches capacity equal to the degree of the node, this returns true so that split process is carried out.
- **public Node insertDataPair(float key, String value)** - inserts the node pair into the leaf node at appropriate position based on traversing through the previously sorted ArrayList (based on key

values). If the key already exists, the value is appended to the ArrayList corresponding to that specific key.

- **public Node splitNode()** - splits the leaf node and returns the newly formed node.
- **public float getMiddleKey() -** returns the middle key of the node that has to be copied into parent index node.
- **public Pair search(float key)** - searches the node for the requested key. Returns the pair if found and null if not found.
- **public ArrayList<Pair> search(float key1, float key2)** - searches for the pairs in the node and the subsequent nodes on the right if needed. Returns an ArrayList (concatenated if traversal was made to any does on the right) of all the found pairs. Returns null if none exist.
- **public ArrayList<Pair> searchNextNode(LeafNode nextNode, float key2)** - an iterative function that returns all the pairs whose keys are less than key2 in the nodes that are on right side of the first node that called this function. This returns an ArrayList of the pairs to the search(key1, key2) function.