Advanced Data Structures

Programming project – Create a m-way B+tree implementation d for internal-memory dictionaries

Name: Kamal Sai Raj Kuncha

UFID: 48548114

Email: k.kuncha@ufl.edu

Function Prototype

(Functions arranged according to their usage respective to the structure under which they are mentioned)

I) B+tree:

a) public void insert(int key, double value) //given dictionary instance insert into tree
b) public void delete(int key) //given key to a dictionary instance delete from tree
c) public Double search(int key) // given key to a dictionary instance search in tree and return value
d) public ArrayList<Double> search(int lowerBound, int upperBound) // given range perform search and return values of keys present in range

II) Index node: // node that holds only keys

a) private boolean checkoverfull() // check if node is overfull
b) private boolean checklend() // check if node can lend to other nodes
c) private boolean checkdef() // check if node is deficient
d) private boolean checkmerge() // check if node can be used for merging
e) private void insertKey(int key) //insert a given key into index node
f) private void removeKey(int index) //remove a key at a position
g) private int findindpointer(Node pointer) //given a pointer find the index of that pointer in the index node and returns that index
h) private void addchildpointer(Node pointer) //insert a child pointer by incrementing the number of pointers
i) private void insertposchildpointer(Node pointer, int index) //insert a child pointer at a given position in child pointers of index node
j) private void removePointer(Node pointer) // remove pointer from child pointers given pointer of index node
k) private void removePointer(int index) //remove a pointer at a position
l) private void removenullindex() //cases where a null pointers happens to appear in the middle of childpointers (happens in removing a pointer) so all remaining pointers after are needed to be moved one spot to the left
m) private int searchnullk(Integer[] keys) // search for the first null in given keys used for creating degree in a indexnode, used in delete borrow, returns the index value

n) private int searchnullp(Node[] pointers) // search for the first null in given pointers used for creating degree in a indexnode, used in split, returns the index value
o) private void shiftpointersamt(Node[] pointers, int amount) // shift pointers by a given amount to left or right ,used in delete borrow
p) private int findindpointer(Node[] pointers, Leafnd node) // given pointers (child pointers) used for index node whose child is leaf node and returns the index value of pointer in child pointers
q) private void sortKeyswithnull(Integer[] dictionary) // sort with null values in a given keys array
r) private Integer[] splitkeys(Integer[] keys, int split) // given position of split, split the key array into two halves with one half staying in original array, deleting the keys from original array and creating a new array for the other half excluding the middle element and returning them
s) private Node[] splitchildp(Indexnd in, int split) //given position of split, split the child pointers array into two halves with one half staying in original array, deleting the pointers from original array and creating a new array for the other half excluding the middle element and returning them
t) private void splitIndexnd(Indexnd in) //split the index node into two nodes by taking care of pointers
u) private void managedef(Indexnd in) // handle deficiency of index node by borrow or merge

III) Leaf node: // node that holds keys and values in format of dictionary instances

a) public boolean checkfull() // check if node is full
b) public boolean checklend() // check if node can lend to other nodes
c) public boolean checkdef() // check if node is deficient
d) public boolean checkmerge() // check if node can be used for merging
e) public boolean insert(Dictionaryinstance dp) // insert a given dictionary instance into node
f) public void delete(int index) //delete a dictionary instance at a given postiton
g) private Leafnd findLeafnd(Indexnd node, int key) // find leaf node where key exists and return that leaf node - uses the root
h) private Leafnd findLeafnd(Indexnd node, int key) // find leaf node where key exists and return that leaf node – uses the given index node
i) private Dictionaryinstance[] splitdict(Leafnd ln, int split) // given position of split, split the dictionary array into two halves with one half staying in original array, deleting the instances from original array and creating a new array for the other half including the middle element and returning them

IV) Dictionary instance: // Data structure that holds the keys and their respective values

a) private int binarySearch(Dictionaryinstance[] dps, int np, int key1) // perform binary search for searching for a key in a given dictionary instance given the number of instances in the node return index of given key
b) private void sortdict(Dictionaryinstance[] dictionary) // sort with null values in a given dictionary instance
c) private int searchnulld(Dictionaryinstance[] dps) // search for the first null in given dictionary instances used for creating number of instances in a leafnode, used in split returns the index value

Program structure

- The path to input file name is given as an argument to the program (bplustree.java)

- The input file (any given name to input text file) is processed in the main function of the program. Each line gets scanned and gets split using parenthesis and commas, into tokens.
- The first token is used to differentiate the functions.

- An output file (output_file.txt) is also created in the same directory for the results needed to be written out.

- If the line consists of initialize then a new B+tree will be created of given order using bplustree(order) in the given line. If the line consists of insert then insert(key,value) of the B+tree will be called with its parameters given in the line. So similarly delete(key) will also be invoked. These functions invoke their separate dependent functions inorder to get the desired results.

- And when search is invoked, the B+ tree performs search operation and returns values, either the normal search(key) or the range search(lowerbound,upperbound), each search output is written in a different line.

- This process goes on until there is no line to read from the input file. And then the program terminates.