**ADVANCED DATA STRUCTURES**
**PROGRAMMING PROJECT REPORT**

# IMPLEMENTATION OF MEMORY RESIDENT B+ TREE

**COP 5536**
**Fall 2017**


**HARI HARA SUBRAMANI SUNDARARAMAN**
**UFID: 3360-2189**
**Email: shhs289@ufl.edu**

## PROJECT DESCRIPTION:

This project includes the complete implementation of a B+ Tree along with its necessary data structures, wherein all the dictionary pairs reside only in the main memory (RAM) instead of a storage disk. It reads the given input file which consists of the order of the tree and the operations to be performed on the tree, namely, inserting a (key, value) pair, searching for a particular key and searching for records that lie within a range of keys. This implementation also has the capability of handling duplicate keys, basically, we can insert same keys but with different values.

## PROJECT ENVIRONMENT:

This project has been implemented using Java and can be compiled using javac command, which has included in the Makefile.
Instructions to run the file:

1. make clean
2. make
3. java treesearch <input_file_name>
4. Output of the performed operations can be found in output_file.txt

## STRUCTURE OF THE PROGRAM:

**treesearch.java:**

1. This is the class which contains the main function.
2. It reads the input file that is given as a command line argument while running the program.
3. Creates an instance of the B+ Tree with the order that is present in the first line of the input file.
4. Each line in the input file contains the operations to be performed on the tree along with the necessary arguments. The code in the main function parses it and invokes the appropriate function on the tree.

5. Finally, it gets the return values for these operations and writes them into the output file.

**BPlusTree.java:**

1. This class contains the complete definition and instantiation of the relevant data structures for maintaining the tree.
2. It includes some custom classes like Node, NodeEntry, InternalNode, DataNode which assist in keeping track of the state of the tree.
3. BPlusTree contains the root of the tree and carries out the insertion, search and rangeSearch on it, and returns the result accordingly.

## CLASS DEFINITION AND FUNCTION PROTOTYPES:

**1. FILE NAME:** BPlusTree.java

- CLASS NAME: Node - represents any nodes in the tree.
  DATA MEMBERS:
  isLeaf - boolean to tell if the node is internal node or leaf node
  keys - arraylist to hold the keys of this node

- CLASS NAME: NodeEntry - used for maintaining a (key, Node) pair, encapsulates the key and node while performing split operations.
  DATA MEMBERS:
  key - double which stores the given key
  node - Node object corresponding to the key

  MEMBER FUNCTIONS:
  Constructor: NodeEntry(Double, Node) - initializes the key and node with the given arguments, namely, k and v.

- CLASS NAME: Util - class which has some static util methods
  MEMBER FUNCTIONS:
  Function Name: binarySearch()

Parameters: ArrayList<Double> list, Double key
Return Values: int
Function to perform binary search for a key in the given list of sorted elements. If the keys exists, it returns the index in the array, else it returns the ideal location (negated value) to insert this key.

- CLASSNAME: InternalNode - extends the Node class to represent the internal node of a B+ Tree, hence it has inherited isLeaf and keys.
  DATA MEMBERS:
  ArrayList<Node> children - List of child nodes for this internal node
  isLeaf, keys - inherited from Node

  MEMBER FUNCTIONS:
  Constructor: InternalNode(Double k, Node n1, Node n2) - Creates an instance of internal node with k as key and n1&n2 being its children.
  Constructor: InternalNode(ArrayList<Double>, ArrayList<Node>) - Creates a new instance of internal node by copying the given list of double values as its keys and the list of nodes being its children.

  Function Name: insert()
  Parameters: NodeEntry e
  Return Values: void
  Function to insert the new node entry into the appropriate child of the internal node - by finding the exact range for this key among its keys

- CLASSNAME: DataNode - extends the Node class to represent the data node (where the dictionary pairs reside) of a B+ Tree. Consists of pointers to maintain the doubly linked list of all the leaves.
  DATA MEMBERS:
  ArrayList<ArrayList<String>> values - To handle the possibility of inserting duplicate keys, it keeps track of a list of strings' list.
  DataNode next - Pointer object to hold the node that is to its right
  DataNode previous - Pointer object to hold the node to its left

isLeaf, keys - inherited from Node class

MEMBER FUNCTIONS:
Constructor: DataNode(Double k, String v) - Creates a new instance of the data node with k as key and v being one of its values.
Constructor: DataNode(ArrayList<Double> keys, ArrayList<ArrayList<String>> values) - Creates a new instance of the data node by copying the given list of double values as its keys and the list of strings' list into its values field.

Function Name: insert()
Parameters: Double k, String v
Return Values: void
Function to insert the new (key, value) pair of the dictionary at the appropriate location of its list of keys and list of values.

- CLASSNAME: BPlusTree - class which represents the B+ Tree and performs the relevant operations (insert or search) on it.
  DATA MEMBERS:
  int M - integers which denotes the order of B+ Tree.
  Node root - refers to the root of the tree, it can either be an internal node or data node.

  MEMBER FUNCTIONS:
  Constructor: BPlusTree(int m) - Creates a new instance of the tree with the given argument as its order.

  Function Name: insert()
  Parameters: Double k, String v
  Return Values: void
  Function which inserts the given (key, value) pair of the dictionary into the tree. This is a public function that gets exposed outside.

Function Name: search()
Parameters: Double k
Return Values: String
Function which searches for the presence of given key in the tree and returns a string containing comma separated list of values (in case of a duplicate key). This is a public function that gets exposed outside.

Function Name: search()
Parameters: Double k1, Double k2
Return Values: String
Function which searches for the bunch of keys in the tree that lie with k1 to k2 and returns a string containing comma separated list of value (key, value) pairs. This is a public function that gets exposed outside.

Function Name: findDataNode()
Parameters: Node node, Double k
Return Values: Node
Private function which searches for the data node that contains the given key (k) under the children of *node*. If found, it returns the DataNode object else returns *null*.

Function Name: getSplitEntry()
Parameters: Node node, NodeEntry entry, NodeEntry nSplitEntry
Return Values: NodeEntry
Private function which adds the new node entry to the appropriate children of *node* and propagate (returns) the split node entry to the caller if needed (to update the root).

Function Name: splitNode()
Parameters: Node node
Return Values: NodeEntry

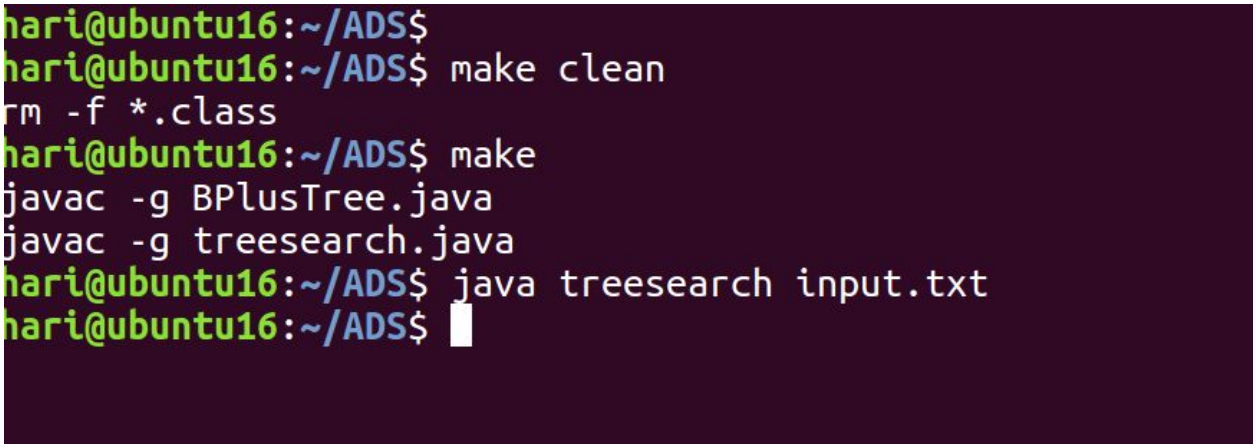
Private function which splits the given overflown node and returns the split node entry to the caller.

**2. FILE NAME:** treesearch.java

- CLASS NAME: treesearch - class which includes the main function.
  MEMBER FUNCTIONS:
  Function Name: main()
  Parameters: String[] args
  Return Values: void
  Function which reads the input file (given in args), and instantiates the BPlusTree to perform relevant operation and write back the result to output_file.txt.

## SCREENSHOTS OF PROGRAM

### Includes the steps to run the program:



## Presents the sample input to the program as given by the TA:

```
6
Insert(-416.24,Value0)
Insert(-385.24,Value1)
Insert(492.67,Value2)
Insert(-458.08,Value3)
Insert(-362.1,Value4)
Insert(-411.43,Value5)
Insert(-449.53,Value6)
Insert(97.4,Value7)
Insert(64.73,Value8)
Insert(27.37,Value9)
Insert(437.49,Value10)
Insert(230.55,Value11)
Insert(251.11,Value12)
Insert(433.92,Value13)
Insert(311.0,Value14)
Insert(435.2,Value15)
Insert(105.88,Value16)
Insert(-403.61,Value17)
Insert(-47.84,Value18)
Insert(426.84,Value19)
Insert(143.98,Value20)
Insert(100.15,Value21)
Insert(-390.13,Value22)
Insert(42.02,Value23)
Insert(-253.57,Value24)
Insert(-234.56,Value25)
```

## Presents the output to above shown input:

```
Value41
(-0.31,Value84), (0.89,Value42), (1.04,Value50), (15.52,Value73), (22.75,Value48), (26.72,Value49), (27.37,Value9)
Value113, Value149, Value184, Value212
(-28.83,Value99), (-28.74,Value100), (-20.28,Value125), (-13.78,Value86), (-12.82,Value199), (-8.95,Value222), (-4.66,Value47), (-3.84,Value207), (-0.31,Value84), (0.89,Value42), (1.04,Value50), (15.52,Value73), (17.99,Value170), (22.75,Value48), (25.29,Value139), (26.72,Value49), (27.37,Value9), (34.58,Value186), (36.57,Value226), (37.58,Value168), (37.78,Value71), (39.46,Value164), (42.02,Value23), (44.15,Value133), (46.7,Value103), (48.68,Value135), (54.56,Value60), (54.74,Value213), (56.49,Value132)
Value7219
(10.46,Value792), (10.51,Value1018), (10.63,Value3458), (10.64,Value5376), (10.94,Value5868), (11.14,Value877), (11.14,Value3533), (11.2,Value3202), (11.31,Value6699), (11.32,Value7023), (11.77,Value3391), (11.81,Value6215), (11.84,Value2988), (11.92,Value6099), (12.02,Value2612), (12.28,Value3971), (12.29,Value4365), (12.42,Value6303), (12.43,Value5350), (12.51,Value4808), (12.66,Value5501), (12.85,Value5127), (12.99,Value1133), (13.0,Value5073), (13.25,Value684), (13.41,Value579), (13.44,Value1019), (13.7,Value6921), (13.74,Value5563), (13.81,Value6078), (14.12,Value5875), (14.14,Value5404), (14.15,Value1027), (14.25,Value5216), (14.49,Value3884), (14.55,Value4719), (14.75,Value6437), (14.75,Value7043), (14.97,Value1572), (14.98,Value7159), (15.06,Value715)
Null
Value9957, Value9983
Value9952
Null
```