COP 5536 Fall 2017
Programming Project
Due Date: Nov 21

Name: Sourav Dutta
UFID: 9863-3443
UF mail: duttasourav@ufl.edu

B+ tree implementation

The project implements a memory resident B+ tree. It supports multiple pairs with same key and implements below operations

1. Initialize – Given m, it initializes a B+ tree with node size m – 1 which can have m children.
2. Insert(key, value) – Insert a key, value pair in the tree
3. Search(key) – Search for a given key and display all its values
4. Search(key1, key2) – returns all keys within the range inclusive of key1 and key2

The structure of the program is defined below:

1. BTree.cpp
   This is the file which contains the main function. Here input file is read and parsed to get m to initialize the tree. Then operations are read, and appropriate tree update and search calls are made.

2. BTreeTypeDefinitions.h
   The structure of the tree is present here. The main structures and functions are described below.

   Node contains the type which defined whether it is an internal node or a leaf node. It contains a sorted vector of keys and a vector of child Node pointers. The type is 0 for internal node and 1 for leaf node.

```
struct Node
{
    int type;
    vector < double > keys;
    vector < Node* > children;
};
```

   LeafNode derives from Node and additionally has a vector of values. Each element within it is again a vector to support duplicate keys. It also contains a pointer to the left and right leaf node to support searching.

```
struct LeafNode : Node
{
    vector < vector < string > > values;
    LeafNode* leftNode;
    LeafNode* rightNode;
};
```

   BTree has the size of the node and the root node pointer.

```
struct BTree
{
    int m;
    Node * root;
};
```

SearchResult is the data returned on searching. It contains the pointer to the leaf node where the key exists or the leaf node where the search ends. It also contains the position of the searched value in that node. If the key is not present, the position value points to the value which is the smallest value greater than the key that is being searched or end of the vector if no such value exists.

```
struct SearchResult
{
    LeafNode* n;
    int pos;
};
```

initializeBTree takes m as a parameter. It is called once to initialize the B+ tree with the node size.

```
BTree* initializeBTree(int m);
```

insertBTree takes three parameters, the B+ tree pointer, key and value. It calls into insertIntoBTreeNode with the root of the tree as a parameter. If the root is empty it creates a LeafNode and then makes the function call. In case insertIntoBTreeNode returns a new root pointer because of a split, it updates the root pointer of the tree. It returns 0 if the insert is successful otherwise 1.

```
int insertBTree(BTree* bt, double key, string data);
```

insertIntoBTreeNode takes four parameters, the node, size of the node, key and value. It searches for the key in the node. If the node is a non-leaf node then it makes a recursive call to the appropriate child node. In case of a leaf node it inserts the value. If the key already exists it appends the value to it and returns NULL otherwise it adds another entry to the keys vector and adds the value.
It checks for overflow and splits the node if the number of elements in the node reaches m. It returns a pointer to the node which is the parent of the two new nodes crated due to split.

```
Node* insertIntoBTreeNode(Node* n, int m, double key, string value);
```

searchKeyBTree takes the tree and key as a parameter. It calls into searchKeyBTreeNode with isRange and isLess set to false.

```
SearchResult* searchKeyBTree(BTree* bt, double key);
```

searchKeyRangeBTree takes three parameters, the tree, starting key and end key of the range. It makes two calls to searchKeyBTreeNode with isRange set to true but once with isLess set to false and true in the second call.

```
vector < SearchResult* > searchKeyRangeBTree(BTree* bt, double key1, double key2);
```

searchKeyBTreeNode takes four parameters. Tree and key are provided when the operation in initiated. isRange parameter is true when the query is for a range search operation and false when it is for a key search operation. When isRange is true and the key is not found, it returns the position of the smallest value which is greater than the key searched. But when it is set to false it returns NULL when the key is not found.
isLess is a parameter which tells whether the searched value is a lower bound search (false) or an upper bound search(true). It is true when the largest value less than the key is required.

```
SearchResult* searchKeyBTreeNode(Node* bt, double key, bool isRange, bool isLess);
```

Apart from this there is a set of helper functions which are present in the file Helper.cpp. These functions are used throughout the program.

Takes a string as input and returns an int.

```
int stringToInt(string s);
```

Takes a string as input and returns a double.

```
double stringToDouble(string s);
```

It is used to print values with atleast one decimal digit. By default 13.0 is printed as 13 and not 13.0.

```
void toPrecisionOneForInt(ofstream& file, double d);
```

Does a binary search on the given array for the value d and returns the positon.

```
int searchArray(vector < double > array, double d);
```

Compares double value and return 0 (equal) if they are within EPSILON, -1 if d1 < d2 and +1 if d1 > d2.

```
#define EPSILON 0.00000001
int compareValues(double d1, double d2);
```

Running Instructions

make
Builds the project and creates an executable named treesearch.

make clean
Cleans the object files and executable.

./treesearch input.txt
Creates an output file named output_file.txt