COP 5536: Advanced Data Structures, Fall 2017
Programming Project Report – B+ Tree

Name – Vishi Jhalani
UFID – 96035253
UF Email – vishijhalani@ufl.edu

Compiling Instructions:

The project has been written in C++ and can be compiled using c++11 with g++ compiler.

1. Unzip the submitted folder. The folder contains the project files (.cpp and .hpp files) and the Makefile. The input text files must be placed in the project directory before running the program.
2. To compile in the Unix environment, run the following command after changing the directory to the project directory:
   ```
   $ make
   ```
3. To run the program such that it reads input from a text-file "filename" that contains input for B+ Tree, use the following command:
   ```
   $ ./treesearch filename
   ```

Program Structure:

The project has a header file, "bplustree.hpp", and three classes, "bplustree.cpp", "node.cpp" and "main.cpp" where main.cpp has the main function along with input and output parsing functioanlity.

main.cpp

This class takes the input file, parses the input data, calls the relevant functions (initialise, insert or search) defined in the respective classes and then writes the output (if any) to the output file, "output_file.txt".

node.cpp

This file describes the structure of nodes used in the B+ Tree.

1. Class "Node" defines a common class inherited by both classes, "InternalNode" and "DataNode". It contains following protected variables and public functions:

   a) Variables:
   ```
   std::vector<float> keys = {FLT_MAX}
   Node *parent = nullptr
   ```

   The last value for keys is assigned as FLT_MAX, as a guard value, to cover the corner case where the element being inserted is the largest one as the insert works by checking elements left to right. It also defines BPlusTree as **friend class**, to access the protected variables.

b) Functions:
`virtual std::string get_type()`
Description: Returns whether the node type is “INTERNAL” or “DATA”, for bookkeeping purposes.

`virtual Node *get_parent()`
Description: Returns the parent of a given node

`virtual void set_parent(Node *node)`
Description: Sets the parent of a given node

`int get_no_of_keys()`
Description: Returns the number of keys present in a given node (InternalNode or DataNode)

2. Class “InternalNode” defines the structure of the index nodes and inherits the class “Node”. It also defines BPlusTree as **friend class**. It has the following variables and public functions:

a) Variables:
`protected std::vector<Node *> child_ptrs`

b) Functions:
`virtual std::string get_type()`
Description: Returns whether the node type is “INTERNAL” or “DATA”, for bookkeeping purposes. Inherited from class “Node”.

`int insert_key(float key)`
Description: Inserts a new pair in the internal node. The function returns the index where the key was inserted which is used to add a child pointer to the internal node.

`void combine(std::pair<InternalNode *, Node *> split_result)`
Description: Combine the parent internal node with given InternalNode which was formed as a result of a split.

`std::pair<InternalNode *, Node *> split(int order)`
Description: Split an InternalNode on the basis of the given order, when no. of keys in the node > order-1. Calculates the number of elements in each node after split, by the formulas taught in class. For InternalNode, it splits into three nodes with ceil(m/2)-1 and m-(ceil(m/2)-1)-1 elements and middle element. This function performs pointer changes to make the new "right" node as the right child of the new "middle" node, and returns the new pair of nodes created as a

result of the split.

`void insert_child(int position, Node *child_ptr)`
Description: Makes child pointer associations for InternalNode.

3. Class "DataNode" defines the structure of the leaf or data nodes and inherits class "Node". "END_MARKER" is defined to make sure where the data in the node ends for genreral bookkeeping. It also defines BPlusTree and InternalNode as **friend class**. It has the following private variables and public functions defined:

    a) Variables:
    ```
    std::vector<std::string> values = {"END_MARKER"};
    DataNode *left = nullptr
    DataNode *right = nullptr
    ```

    b) Functions:
    `virtual std::string get_type()`
    Description: Returns whether the node type is "INTERNAL" or "DATA", for bookkeeping purposes. Inherited from class "Node".

    `void insert(float key, std::string value)`
    Description: Inserts a given Key-Value pair in a given DataNode.

    `std::pair<InternalNode *, Node *> split(int order)`
    Description: Splits a DataNode on the basis of the given order - when no. of pairs > order-1 for the node. For DataNode, it splits into two nodes with ceil(m/2)-1 and m-(ceil(m/2)-1) elements. Returns the new pair of node created during split.

bplustree.hpp
Declares the following private variables and public functions which are defined in bplustree.cpp:

a) Variables:
```
Node *root;
int order;
```

b) Functions:
`void initialise(int order)`
Description: Intialise the B+ tree with the given order, passed as input. Also, creates a new empty DataNode as the root.

`void insert(float insert_key, std::string value)`

Description: Insert a new Key-Value pair in the B+ tree. If no. of keys in any node becomes > order-1, then this function calls split function on that node and then the combine function recursively, till no split occurs.

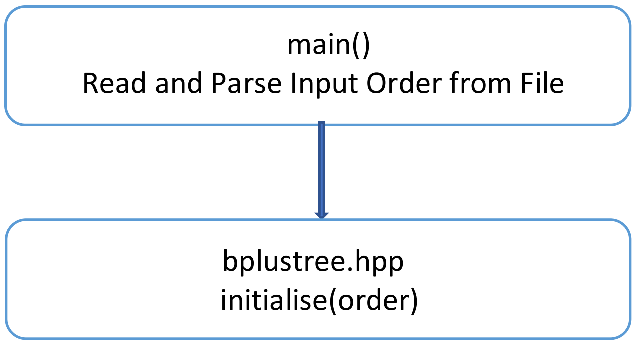
```
std::vector<std::string> search(float key)
```
Description: Performs an exact search given a key input. Since, there can be duplicate keys in the B+ tree, it returns a vector of string values corresponding to the key. If no value found, it returns empty vector, which is parsed as NULL when written to the output file.

```
std::vector<std::pair<float, std::string>> search(float start_key, float
end_key)
```
Description: Performs range search on the B+ Tree. It starts tracing the tree from the root, using the start_key until it reaches the DataNode and then traces the doubly linked list rightwards, in the Data Nodes to fetch all the relevant Key-Value pairs in the range till end_key is reached. Returns a vector of Key-Value pairs in the given range.

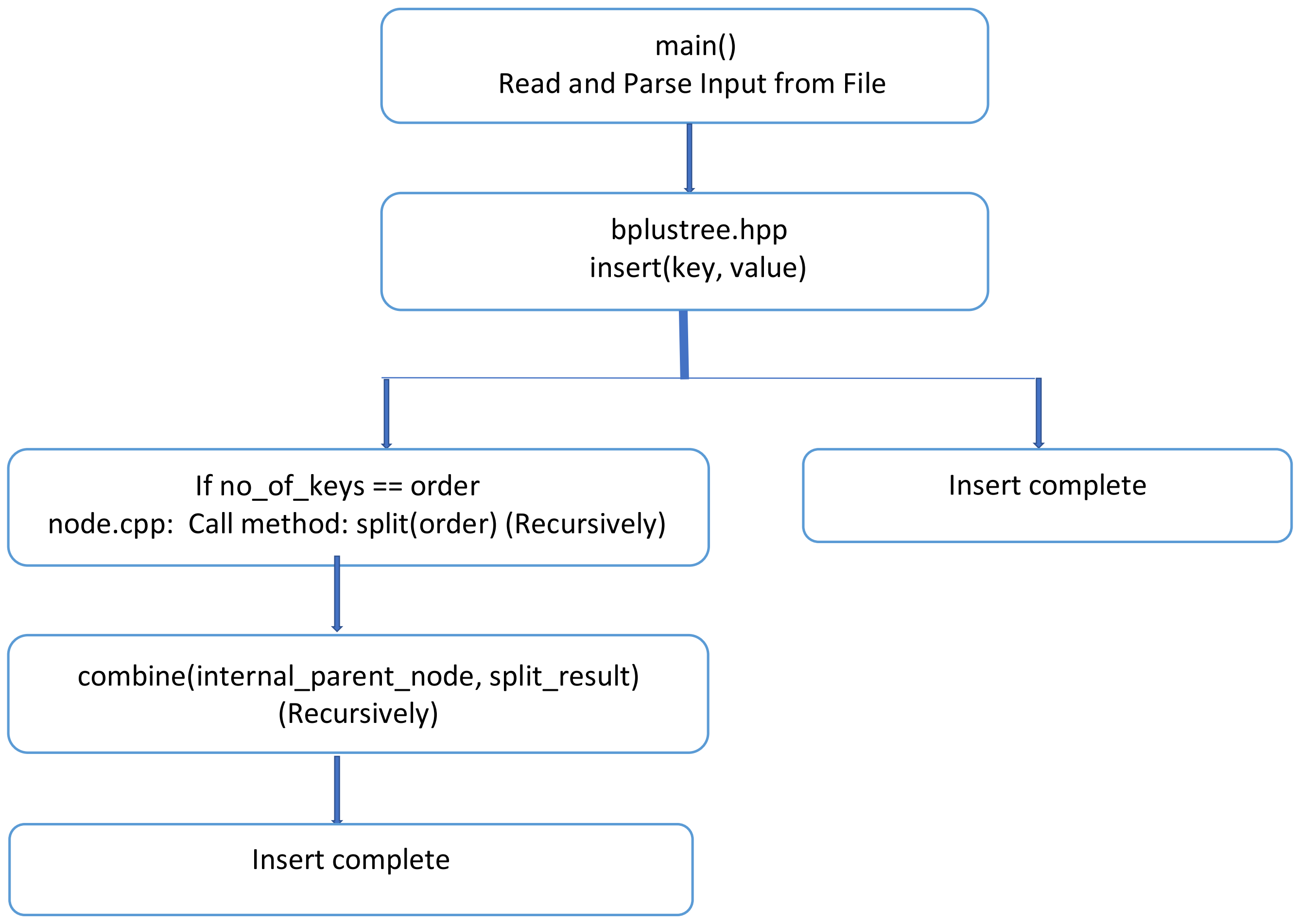
bplustree.cpp
Defines all the functions declared in header file bplus.hpp.

Code Flow for various operations:

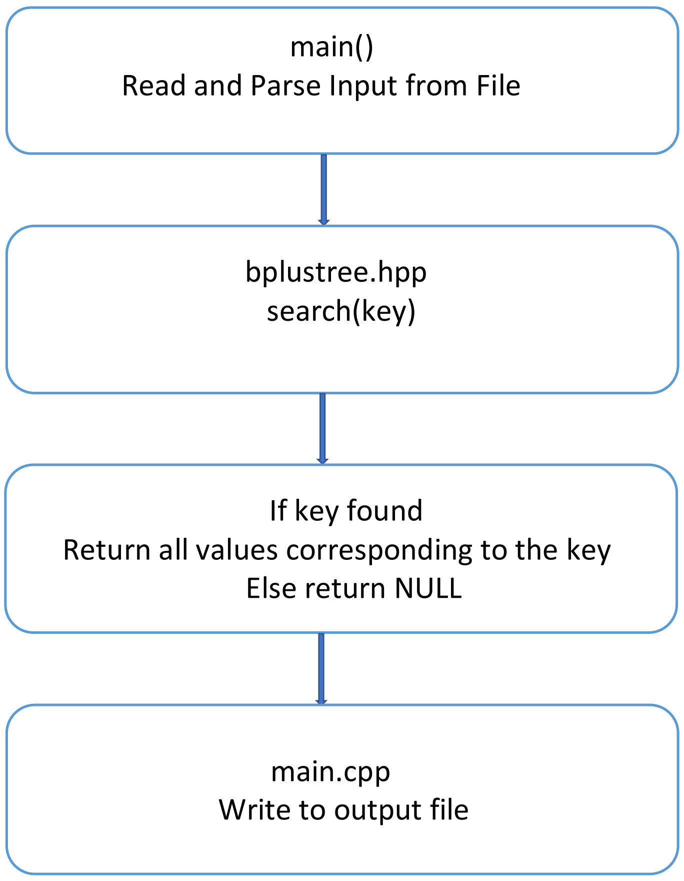
1. **Initialise**



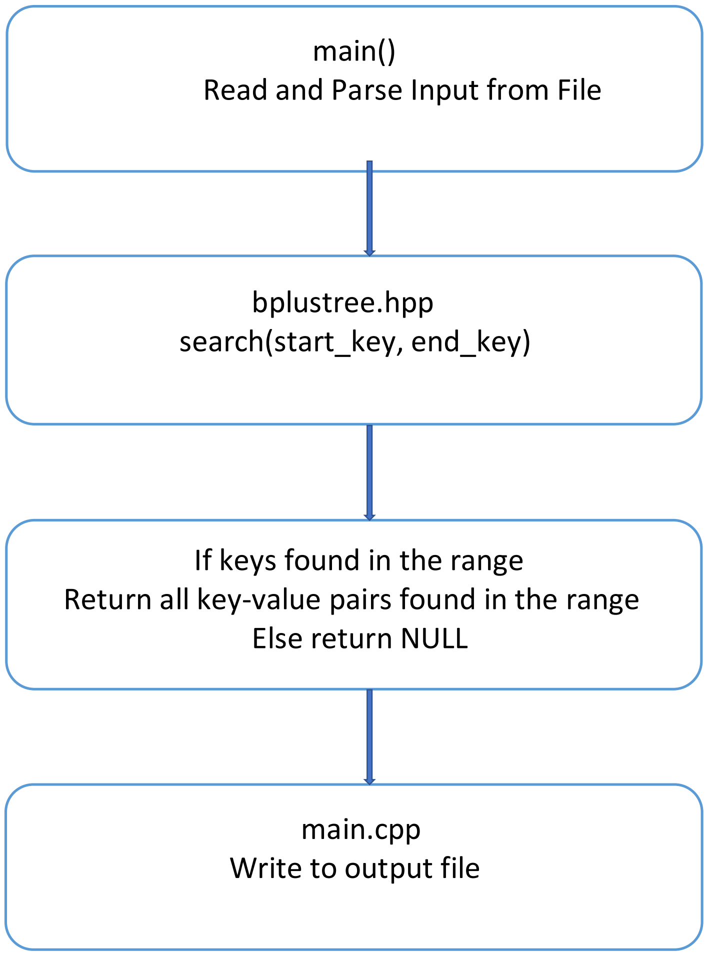
2. Insert


main()
Read and Parse Input from File
bplustree.hpp
insert(key, value)
If no_of_keys == order
node.cpp: Call method: split(order) (Recursively)
Insert complete
combine(internal_parent_node, split_result)
(Recursively)
Insert complete

3. Exact Search



4. Range Search


main()
Read and Parse Input from File
bplustree.hpp
search(start_key, end_key)
If keys found in the range
Return all key-value pairs found in the range
Else return NULL
main.cpp
Write to output file