**HW 8**

* We are implementing a partial and heavily modified B+ tree
  + For an unbalanced BST, O(n) worst case for the height, which affects find/insert/erase
  + For a balanced BST, log(n) worst case for the height, which makes find/insert/erase better
  + B+ tree is even better though as B+ trees are flatter and wider because they store more values as children, which decreases the number of nodes you have to step through so find/insert/erase accesses are faster (loading in each node in certain cases like if the B+ tree is huge and is stored on an external hard drive can be quite expensive in terms of time)
  + We will only have keys and all keys must be unique (so it’s like a set)
  + All keys are stored in leaves (other nodes have duplicates) - note that for normal B+ trees the leaves are connected in a linked list manner to make accessing all the values in the B+ tree quick once you are at a leaf node but for this HW to keep it simple we won’t be doing implementing that
  + All leaves will always be the same height away from the root node
  + Note that for b children, b - 1 values will be stored in each node
  + All the keys that we insert are unique
  + Assume b > 2 in all tests
  + Left child < key0, next child >= key1 & < key2, etc.
  + Print functions only need to work with type/classes that already work with operator<<
  + PrintSideways makes it split at b/2 children
* Hints
  + Iterators will not be implemented so find() should return a pointer to a BPlusTreeNode
    - If the tree is empty, this will be a nullptr
    - Otherwise, this will be the leaf node where the key is/would be
  + In the middle of an insertion, it is okay to let your nodes hold too many keys or children as long as you fix it before the insertion and splits are finished
  + Since we are working with trees, it will be more natural to do something things recursively
  + Make a separate function to handle all the splits that will be a helper function for insert
* You are encouraged to write your own test functions, but they will not be looked at
* You only need to submit a README.txt and BPlusTree.h file
* Make sure your code is memory error/leak free
* Extra credit
  + print\_BFS() currently has output where it is not possible to tell which nodes are children of a particular node
  + Assuming each key is short (i.e. no more than 2 characters wide), implement a functionprint\_BFS\_pretty() that still uses BFS ordering and a vertical layout like print\_BFS() but that has appropriate spacing so that the structure of the tree is apparent
  + There are several possible ways to do this, so you can choose a design you think is reasonable
  + Leave a note in the README if you choose to implement the extra credit