Download the code that implements B-trees and extend it in the following ways:

1. Modify the insertion operation to prevent duplicate elements from being inserted in the tree. If the user tries to insert an element that is already in the tree, your program should simply ignore the request.

2. Add a variable to the BTree object to store the number of keys in the tree and make sure the variable is kept up to date after every insertion operation (observe how the height of the tree is kept in a variable for efficiency, but it could also be easily computed).

3. Add a variable to the BTree object to store the number of nodes in the tree and make sure the variable is kept up to date after every insertion operation.

4. Write methods that perform the following operations:
   
   (a) Print the keys in the tree in ascending order
   (b) Print the keys in the tree in descending order
   (c) Determine if a given element $k$ is in the tree.
   (d) Return the minimum element in the tree
   (e) Return the maximum element in the tree
   (f) Return the number of nodes in the tree
   (g) Return the number of keys in the tree
   (h) Return the number of leaves in the tree
   (i) Given an integer $d$, print all the keys in the tree that have depth $d$
   (j) Extract the keys in the tree into a sorted array.
   (k) Given an integer $d$ and a b-tree $T$, build and return a B-tree that consists of the first $d$ levels of $T$. Thus, if $d == 0$ you method should return a copy of the root of $T$ and if $d == \text{height}(T)$ your method should return a copy of the whole tree $T$.

As usual, write a report describing your work.