Advanced Algorithms and Data Structures project

Exercise 1

1. Provide an implementation of the B-Tree data structure. Note that you are required to choose how to implement the Node structure. Recall that different choices may have different performances both in terms of computational complexity and in terms of I/O complexity. Motivate your choice.

   Provide, moreover, a script that verifies all implemented functionalities.

Our implementation of the Node structure is based on two SortedTableMaps, that support the sorted map ADT. The first SortedTableMap contains the elements of the current node of the BTree and the second contains all the children.
The most notable feature of this structure is the inclusion of a find_index utility function. This method uses the search algorithm, but by convention returns the index of the left-most item in the search interval having key greater than or equal to k. Therefore, if the key is present, it will return the index of the item having that key (keys are unique in a map). The body of each of the get_item, set_item, and del_item methods begins with a call to find_index to determine a candidate index at which a matching key might be found.
This choice is useful because the search needs to be fast and the analysis for the various forms of search all depend on the fact that a binary search on a table with n entries runs in O(logn) time. In conclusion, we use SortedTableMaps because we expect many searches but relatively few updates.

Operation	Running Time
len(M)	$O(1)$
k in M	$O(\log n)$
M[k] = v	$O(n)$ worst case; $O(\log n)$ if existing k
del M[k]	$O(n)$ worst case
M.find_min( ), M.find_max( )	$O(1)$
M.find_lt(k), M.find_gt(k) M.find_le(k), M.find_ge(k)	$O(\log n)$
M.find_range(start, stop)	$O(s+\log n)$ where s items are reported
iter(M), reversed(M)	$O(n)$