# Lab Exercise 6
# Indexing a Book

You are given a book with various chapters, each chapter in a separate text file. You need to create an index of the book. For creating the index, you need to include only the words which are not so common. Any word which occurs in all the chapters is considered to be a common word and hence excluded from being a part of the index.

Your program will be given a list of file names (.txt files) with the contents of the chapters. Each file needs to be read and each word in the file needs to be stored in an ordered dictionary. The word itself is used as the key to be stored and the chapter name is the element/value corresponding to the key. Ordering of the keys follows the lexicographic ordering of the words. Corresponding to each key there is a histogram which records the frequency (count) of that key (word) in each chapter. A word can occur multiple times in various chapters of the book.

## Task 1: Processing the input files.

As a chapter is processed, the words are inserted into the ordered dictionary. When a word is stored in the ordered dictionary for the first time, the histogram (indicating the chapter wise occurrence of that word) is instantiated. If a word being inserted is already there in the ordered dictionary, its histogram will be updated. The (key, element) pair constitutes the word as the key and the name of the chapter as the element. If the key (word) is already there as a node in the tree, then the name of the chapter is used to update the corresponding bin count of the histogram. If the key is not already there, then we insert a new node for that *key*.

The ordered dictionary needs to be implemented using a red black tree. The histogram needs to be implemented using a data structure (let's call it MRU— most recently used) that is able to retrieve the most recently accessed object in $O(1)$ time. Each object in the MRU is a histogram bin object which keeps information regarding the name of the chapter and the number of occurrences of the word in that chapter. The sum of the bin counts for all the chapters will give the number of occurrences of that word in all the chapters.

## Task 2: Pruning the red black tree

After processing all the chapters, the red black tree will have nodes corresponding to every unique word in the book. For preparing the index, you need to identify the words that do NOT occur in ALL the chapters. All the words that have at least one occurrence in all the chapters need to be removed from the red black tree by calling function deleteKey(…) for each such word. The ***Lexicon*** object now has an red black tree of words that will be used for preparing the index.

## Task 3: Building the index

The index will be a list of ***IndexEntry*** objects arranged in alphabetical order of the words, mentioning the chapter names in which each word occurs and the word count.

Class templates will be available for the following objects:

***Lexicon***
Supports functions for reading the chapters (.txt files), building the index using the RedBlackTree object.

***RedBlackTree***
This class supports functions for

- Inserting a key, element pair into the RedBlack tree — this function returns the pointer to the node that is being inserted/updated,

- Deleting a key from the RedBlack tree — the function returns true if the node has been deleted. It returns false if the key is not there in the tree.
- Searching a key in the red tree black tree — the function returns  a pointer to the node having that key.
- Getting the black height of a node.
- Traversing up a path from a given RedBlack tree node to the root,
- Traversing down a path from a given AVL tree node guided by a bit sequence. The bit sequence, e.g. 1001101010, can indicate to take the left child or the right child, whether a bit is 1 (take left), or 0 (take right).
- Returning the ***pre-order*** sequence of nodes, starting from a given node, and staying within a *specified depth* from the given node. That means, once the *specified depth* is reached, the inorder traversal will backtrack and move to other nodes within the *specified depth*. All the paths/sequences of nodes are to be returned in the form of a linked list. All properties of a red black tree should be preserved after every insertion and deletion. The *specified depth* is not the black depth, but the ***general depth***.

***HybridNode***

Each node of the RedBlackTree is an object of the class HybridNode. It is hybrid because it will allow the node to be used in a double role — a node of the red black tree, as well as the node of a linked list. As a node of a tree, it will have pointers to the parent, the left child, and the right child. As a node of a linked list it will have a pointer to the next node. When a path/sequence of nodes is extracted from the RedBlack tree, the nodes on the path are linked to form a linked list and a pointer to the head (first) node of the linked list is returned.

***IndexEntry***

This class has data fields indicating the word, and a list of chapters in which it occurs along with the chapter-wise word count.

Simplifications:

1. All punctuation marks are to be treated as a white space.
2. Design and implement your own MRU class which can access the most recently accessed histogram bin in $O(1)$ time.
3. The name of the .txt file can be considered as the name of the chapter.