

## Problems

1. (30 pts, **Mandatory**) Write up a max one-page summary of all concepts and techniques in CLRS Chapter 10 (Simple Data Structures).
2. (30 pts, **Mandatory**) Write up a max one-page summary of all concepts and techniques in CLRS Chapter 12 (Binary Search Trees).
3. (10 pts) Exercise 10.1-1.
4. (10 pts) Exercise 10.1-4.
5. (10 pts) Exercise 10.1-6.
6. (Extra Credit) Exercise 10.1-7.
7. (10 pts) Exercise 10.2-2.
8. (10 pts) Exercise 10.2-6.
9. (10 pts) Exercise 10.4-2.
10. (Extra Credit) Problem 10-1.
11. (10 pts) Exercise 12.2-5.
12. (10 pts) Exercise 12.2-7.
13. (10 pts) Exercise 12.3-3.
14. (Extra Credit) Problem 12-3.
15. (Extra Credit) Problem 10-2.
16. (Extra Credit) Problem 15-6.
17. (50 pts) Implement a hash for text. Given a string as input, construct a hash with words as keys, and word counts as values. Your implementation should include:
  - a hash function that has good properties for text
  - storage and collision management using linked lists
  - operations: insert(key,value), delete(key), increase(key), find(key), list-all-keys

Output the list of words together with their counts on an output file. For this problem, you cannot use built-in-language datastructures that can index by strings (like hash tables). Use a language that easily implements linked lists, like C/C++.

You can test your code on “Alice in Wonderland” by Lewis Carroll, at [http://www.ccs.neu.edu/home/vip/teach/Algorithms/7\\_hash\\_RBtree\\_simpleDS/hw\\_hash\\_RBtree/alice\\_in\\_wonderland.txt](http://www.ccs.neu.edu/home/vip/teach/Algorithms/7_hash_RBtree_simpleDS/hw_hash_RBtree/alice_in_wonderland.txt).

The test file used by TA will probably be shorter.

(Extra Credit) Find a way to record not only word counts, but also the positions in text. For each word, besides the count value, built a linked list with positions in the given text. Output this list together with the count.

18. (50points) Implement a red-black tree, including binary-search-tree operations *sort*, *search*, *min*, *max*, *successor*, *predecessor* and specific red-black procedures *rotation*, *insert*, *delete*. The *delete* implementation is Extra Credit (but highly recommended).

Your code should take the input array of numbers from a file and build a red-black tree with this input by sequence of “inserts”. Then interactively ask the user for an operational command like “insert x” or “sort” or “search x” etc, on each of which your code rearranges the tree and if needed produces an output. After each operation also print out the height of the tree.

You can use any mechanism to implement the tree, for example with pointers and struct objects in C++, or with arrays of indices that represent links between parent and children. You cannot use any tree built-in structure in any language.