CS5800: Algorithms Instructor: Virgil Pavlu

Homework 8

Problems

1. (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
   - each word (key) can only appear once in the datastructure

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

   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`.
   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.

2. (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.

3. (Implement Skiplists 50 points). Study the skiplist data structure and operations. They are used for sorting values, but in a datastructure more effficient than lists or arrays, and more guaranteed than binary search trees. Review Slides `skiplists.pdf`

and Visualizer `https://people.ok.ubc.ca/ylucet/DS/SkipList.html`. The demo will be a sequence of operations (asked by TA) such as for example
insert 20, insert 40, insert 10, insert 20, insert 5, insert 80, delete 20, insert 100, insert 20, insert 30, delete 5, insert 50, lookup 80, etc

4. (DP problem; Extra credit 50 pts) The **int**[] bag describes a bag of non-negative integers. A bag is the same thing as a set, only it may contain repeated elements. The order of elements in a bag does not matter. Given two bags A and B, we say that A is a *sub-bag* of B if A can be obtained by erasing zero or more elements from B. The *weight* of a bag is the sum of its elements. Examples: The bags $(1, 2, 1, 3, 1)$ and $(3, 1, 1, 1, 2)$ are the same, but different from the bag $(1, 2, 3, 3)$. Bags $(1, 2)$ and $(3, 1, 1)$ are sub-bags of the bag $(1, 2, 1, 3, 1)$, but bag $(1, 2, 2)$ is not. The weight of the bag $(1, 2, 1, 3, 1)$ is $1 + 2 + 1 + 3 + 1 = 8$. The bag will contain less than 50 elements, and each element will be less than 10,000. Write a method that will compute how many sub-bags of bag have a prime weight. Your program should **not** use more than $3M$ memory, and should pass the following testing cases (the running time of each testing case should be within 3 seconds).

   Input 1: {1,1,1,1,1,1,1,1,1,1}

   Return 1: 4

   Input 2: {4,6,8,10,12,14}

   Return 2: 0

   Input 3: {1,2,4,8,16,32,64,128}

   Return 3: 54

   Input 4: {9947, 9948, 9949, 9950, 9951, 9952, 9953, 9954, 9955, 9956, 9957, 9958, 9959, 9960, 9961, 9962, 9963, 9964, 9965, 9966, 9967, 9968, 9969, 9970, 9971, 9972, 9973, 9974, 9975, 9976, 9977, 9978, 9979, 9980, 9981, 9982, 9983, 9984, 9985, 9986, 9987, 9988, 9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996}

   Return 4: 91378169764810

   Input 5: {9947, 9934, 9934}

   Return 5: 54

   Input 6: {1, 1, 2, 2, 4, 4, 8, 8, 16, 16, 32, 32, 64, 64, 128, 128, 256, 256, 512, 512, 1024, 1024, 2048, 2048, 4096, 1, 1, 2, 2, 4, 4, 8, 8, 16, 16, 32, 32, 64, 64, 128, 128, 256, 256, 512, 512, 1024, 1024, 2048, 2048, 4096}

Return 6: 62648678

Input 7: {10000, 9999, 9998, 9997, 9996, 9995, 9994, 9993, 9992, 9991, 9990, 9989, 9988, 9987, 9986, 9985, 9984, 9983, 9982, 9981, 9980, 9979, 9978, 9977, 9976, 9975, 9974, 9973, 9972, 9971, 9970, 9969, 9968, 9967, 9966, 9965, 9964, 9963, 9962, 9961, 9960, 9959, 9958, 9957, 9956, 9955, 9954, 9953, 9952, 9951}

Return 7: 89655114688016

Input 8: {9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999}

Return 8: 4814999

Input 9: {1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 10000}

Return 9: 6555