FIT2004: Lab questions for week 6

Objectives: This prac provides a platform for you to learn the formal concepts introduced during lectures in weeks 3, 5& 6. Primarily, these concepts include retrieval trees, partitioning problems, recursive sorting, and divide and conquer strategies. NOTE: This prac is **NOT** assessed. These programs are to be written in python programming language.

1. Write a program to generate a `Trie` from a given set of strings (to be read from a file at command line). Your program should support basic operations such as: `insertWord`, `deleteWord`, `searchWord`, `printWords`[1]. As one of the possible set of strings to start testing your program, consider the following:

   ```
   1 rubicundus
   2 romulus
   3 romane
   4 rubicon
   5 ruber
   6 romanus
   7 rubens
   ```

   Note: Assessment lab for week 08 requires you to extend the above basic implementation of a Trie. You will need to implement the above version of Trie for the assessment anyway. Therefore, it is better to utilize your time in this lab (before/after your interview with the tutor) to implement the basic version of Trie and iron out bugs. If you face any issues, you can post on Moodle forum, email me, search online, or ask your tutor in the next lab.

2. Last week in tutorial, we discussed the **three-way** Dutch-National-Flag partitioning problem (see the PDF file on Moodle under week 05 tab). Write a program implementing this algorithm.

   Your program must be able to generate a random instance of this partitioning problem and solve it correctly. To generate the random instance, fix the size of the set (to be partitioned) to 1 million and populate it with 1 million *uniform* random integers [0,1,2].[2]

   Once this instance is partitioned, your program should automatically check if the partitioning your programming is generating is correct – that is all 0s, 1s and 2s have been correctly grouped in that order – or assert otherwise. When correct, your program should output the summary of partition sizes, i.e., the number of 0s, 1s and 2s along with their partition boundaries.

[1] in a lexicographic order

[2] Random number generation in Python, see `http://docs.python.org/2/library/random.html#random.randrange`

3. Now extend the above program to achieve a 4-way partitioning of a set of million random integers in the range [0,1,2,3].

4. If you were to generalize this Dutch National Flag approach to an arbitrary k-way partitioning problem, what would the time-complexity and space-complexity of this algorithm? (**This is NOT a computer-based question**, although eager students **should** attempt to write a generalized version.)

5. Write a Quick Sort program, using the 3-way Dutch-National Flag algorithm. Test your program on how many integers you can feasibly sort (in some practical amount of time) using your implementation.

   To do this, using the background information in the week 4 prac, generate a random permutation over N integers, and compute the time your program takes to run for increasing limits of N.

   Another exercise is to generate the numbers randomly (which may contain varying number of repeat elements) to create instances that are sorted using your program.

   Also test whether your algorithm runs faster or slower if you gave it a fully sorted list of numbers, and fully reversed list of numbers.

```
-=o0o=-
  END
-=o0o=-
```