FIT2004: Tutorial 3 (held in Week 7)
Covering concepts from Weeks 4, 5, & 6

Objectives: The tutorials, in general, give practice in problem solving, in analysis of algorithms and data-structures, and in mathematics and logic useful in the above.

Instructions to the class: Prepare your answers to the questions **before** the tutorial! It will probably not be possible to cover all questions unless the class has prepared them in advance. There is 1 mark worth for this Tute towards **active** participation. 0.5 marks is towards answering the starred questions (*) indicated below, **before** attending your assigned tutorial. You will have to hand in your work on these starred questions to your tutor at the very start of the tutorial. Remaining 0.5 mark is for participating during the rest of the tutorial.

Instructions to Tutors:

1. The purpose of the tutorials is not to solve the practical exercises!
2. The purpose is to check answers, and to discuss particular sticking points, not to simply make answers available.

1. We demonstrated the dynamic programming paradigm using the edit distance problem. Recall that if you are given two strings, $s1$ and $s2$, are, $d(s1, s2) = 0$, if and only if $s1 = s2$. An *optimal* alignment between $s1$ and $s2$ shows how to edit and covert $s1$ into $s2$ with minimal cost.

Consider a *related* yet different problem:

The *longest common subsequence* (LCS) *problem* requires you to find the *longest* possible *subsequence* shared/common between two strings $s1$ and $s2$.[1] An optimal LCS-alignment shows the LCS between $s1$ and $s2$, which can again be solved using the dynamic programming paradigm, in an approach reminiscent of what you learnt for the edit distance problem.

Can you work out the dynamic programming recurrence relationship for this problem. Your focus should be on the optimal substructure

[1]Note that a **subsequence** is *different* from a **substring**, in that the letters in a subsequence need not be appear consecutively in the original string, where as in a substring they do appear consecutively. For example, if the sequence $s = s_1 s_2 \dots s_n$, a **subsequence** is some (or all) of its elements, in the same order, for instance, $s_3 s_5 s_8 s_{10} s_{13}$.

and overlapping subproblem nature of the LCS problem. It would be straightforward to grasp, if you have understood the edit distance problem.

∗ 2. Write a routine '**`between(T,lo,hi)`**' which is given a BST, **`T`**, and must print all entries in **`T`** that are are between **`lo`** and **`hi`** **inclusive**.

∗ 3. Work out **by hand** the resulting the binary search trees (BST) generated by the series of **`insertions`** of numbers/keys below:

$$19, 23, 13, 7, 73, 3, 18, 17, 11$$

Then work out the changes to the tree structure by **`deletions`** of the following numbers/keys:

$$73, 13, 11$$

∗ 4. Repeat the same exercise, this time maintaining an height-balanaced AVL tree.

5. Construct a suffix trie **by hand** of the string '**`woolloomooloo`**'. Path compress this trie into a suffix tree.

-=END=-