BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

Second Semester 2013-2014

CS /IS F211 Data Structures and Algorithms

Lab Sheet – 5 [Duration: 150 minutes]

General Instructions for Programming

1. All inputs to the program must be either (a) command line arguments (b) or read from a file (other than stdin). DO NOT READ anything from stdin and DO NOT USE PROMPTS like "Please enter a number …".

2. You are required to write the output to a file (other than stdout) and errors if any to a different output channel (stderr or another file).

3. Use standard C coding conventions for multi-file programs. Separate the following: interfaces of functions (use a ".h" file), data type definitions (use another ".h" file), ADT / algorithm implementation (use a ".c" file), and driver/test code (use another ".c" code). In general, each module has to be written in **separate** c files.

4. All files related to a lab **must** be put inside a single directory by the name of the lab (lab1, lab2, etc.).

5. Valid makefile must be present in the directory.

6. Ensure that all the code written by you are compiling correctly. Preferably use gcc with the options **`-W -Wall -O2`**, while compiling your code.

7. Instructions for uploading the files shall be provided separately.

Problem 1

A Binary Search Tree(**`BST`**) is a binary tree in which all nodes in the left sub-tree are less than the current node, and all node in the every node left sub-tree are less than the current node. In this lab, you are expected to create a **`BST`** tree of integers stored in an input file. For this problem, assume that all the values in the input are unique.

Node structure:

left	value	right	succ

Here, value is the element stored in the **`BST, and`** left and right are the pointers to left and right sub-trees. succ pointer points to the in-order successor of the current node.

Balancing mechanism:

A simple mechanism to balance the height of the tree: delete element from a longer subtree, push an element to smaller subtree, and adjust root accordingly.

Type definitions: (**tree.h**)

The structure **`BST`** contains the definitions for the node structure indicated above.

List of Functions to implement:

1. Common list of functions like insert, delete, find.
2. Use a balance mechanism given to balance the tree.
3. Perform comparative evaluation of balanced and unbalanced BST

Problem 2

Given a large text, create an index of words appearing in the text, along with the location(s) in which these words appear.

Data source for this problem can be downloaded from : http://www.gutenberg.org/ebooks/829 (you may use any other ebook as well.)

Algorithm Outline:

1. Sanitize the text to remove all characters that are not alphabets or white spaces.
2. Maintain a data structure to handle index.
3. Traverse the text word by word.
4. If it is a new word,
    a. add to index (along with location),
5. else
    a. search the word in index and add the extra location.

6. Print the Index to a file ***(index.txt)*** indicating word and the location(s) where the word is found.

   Note: use `man fseek` to know how to find position or go to a position

Approach one: AVL tree of words.

Maintain the index as an AVL tree.

You may use the node structure of Problem1 for this.

Final index is an inorder traversal.(follow the nodes like a linked list, by using the succ pointer instead of the next pointer of traditional linked lists.)

Approach two: Hashtable of words. **(Take home)**

Maintain the index as a hash table using separate chaining.

You may use the hash function: hash(X) = (summation of ASCII of all characters) mod m

Where m=511 initially.

Approach 3: use tries. **(Exercise/Take home)**

Use a standard trie(**Exercise**), a compressed trie, compact trie for building the index.

Note that for printing the index in order, you can perform a depth first traversal of the trie.