COMP20003 Algorithms and Data Structures

Worksheet 4

[week starting 15 of August]

Second (Spring) Semester 2016

Overview

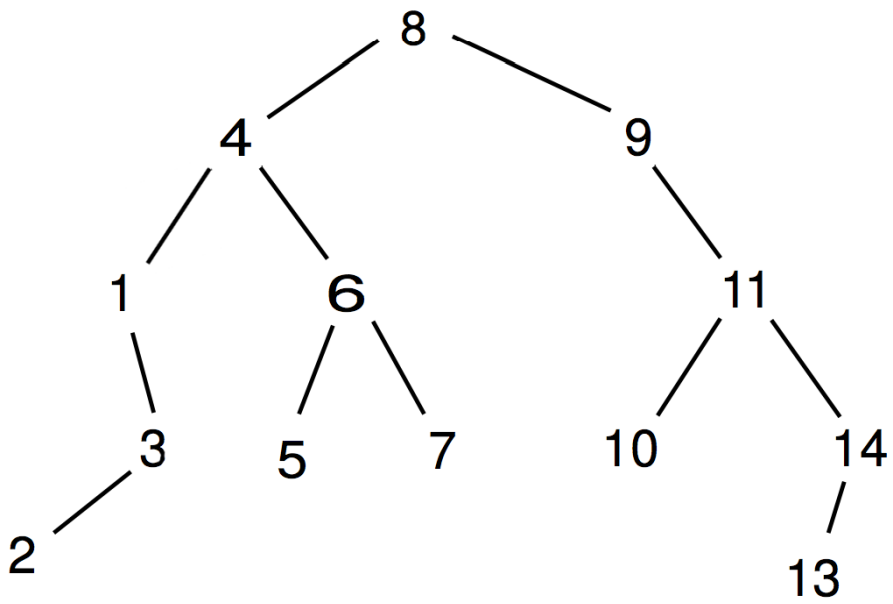
The workshop for Week 4 will start with a tutorial on binary search tree insertion and how to keep a node counter updated once an item is inserted.

Tutorial Questions

Question 4.1 Given the following input:

- 8 4 9 11 6 7 1 5 3 14 10 13 2

Write the insertion function used to get the binary search tree shown below.



Draw a new tree after inserting each number and show the pointer reassignments and comparisons necessary to implement each step of these insertions.

Question 4.2 In some circumstances, it might be useful to have a doubly-linked tree, i.e. each node has a parent pointer (root pointer is initialized to `null`), as well as two child pointers. Assume each node has a `counter` and a `depth` variable. After each insertion some counters get updated using the following assignment

```
1 node.counter = node.left.depth − node.right.depth;
```

Show how can you use the doubly-linked tree to keep the counters updated after a node is `inserted`.

Programming exercises

Programming 4.1 For this exercise, you will program a BINARY TREE and the INSERTION function.

Given the input from the tutorial question, you should be able to build the tree depicted in the picture above. Once the tree is loaded, you should be able to perform any VALUE LOOKUP and PRINT HOW MANY COMPARISONS you needed to find the item, or prove that it does not exist in the tree.

You will practice again how to compile using makefile and debug your program if a bug exists.

Use **GDB** and **Valgrind**. You will find the **GDB reference card** file useful, along with the **Valgrind tutorial** on LMS-Resrources section. Valgrind is a great tool to find memory leaks!

This is the most basic valgrind command:

```
valgrind --tool=memcheck program_name
```

If you want a detailed description of memory leaks, add the following option `--leak-check=yes`

This is a quick summary of the basic commands for GDB.

How to run your program:

```
gdb --args .\program_name
```

once you're inside gdb:

- b filename.cpp:line (breakpoint)
- r (run)
- CTRL+c (stop execution)
- c (continue)
- n (next)
- s (step)
- d #number (disable breakpoint #number)
- p name_variable (print the value of the variable)
- bt (backtrace: displays program stack)
- q (quit: exits gdb)

Once you have created a `breakpoint`, and `run` your program, then you can visualize your code.

Visualization :

- CTRL-x + 1 (visualize the code you're debugging)
- CTRL-x + o (change focus of screen from code to gdb terminal)
- CTRL-x + a (kill visualization of code)

If you finish in time, try to MANTAIN A BALANCED TREE WHILE INSERTING using rotations.