**ADS C++ Elapsed CA due 22/03/17**

**Lee Healy X00120179**

**Analysis Document.**

**Runtimes:**

Running the project with **1 City object**:



Running with **3 City objects**:



You can see the wall time jump from **0.0016** to **0.0076**.

Running with **8 City objects**:



You can see that the wall time is rising due to how many cities are in the tree.

**EXTREME CASE:**

This involved creating 26 cities named letters of the alphabet A-Z. The three was 1 sided (Linked list) as the root was A. This case finally invoked a response time from the CPU time where the rest where all **0** this was **0.0625**.

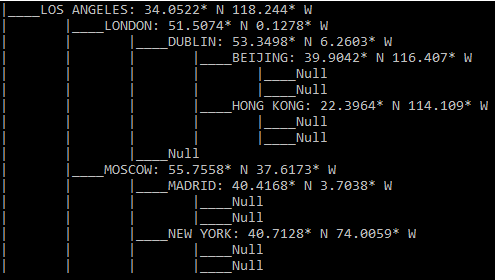


**Observations:**

During the development of my project I came across several problems that I needed to address.

The first issue being the insertion into the binary tree. I had to make sure that when data was inserted into the tree it was checked whether it was greater or less than the root based on name. This gives the tree balance, completeness and improves runtimes and search times.

The second problem I came across was displaying the tree. I found out early on that without formatting, the tree displayed data vertically without any spacing or visual help to distinguish separate parts of the tree. In order to change this I passed in a string to the preOrderTraversal() function which then tabbed and added a tab (indent += “\t”) every time it was called. I also added lines and dashes to show branches more clearly. This gave the tree a pretty print and a visual view in which the user could see each branch and leaf of the binary tree.



Another issue I came across was when creating the searchGPS function. This function took in two co-ordinates and returned the city that matched these co-ordinates. To do this we needed to traverse the tree in some order. However, since the tree is balanced by city name and not city GPS, in this function we could not distinguish whether the co-ordinates where either greater than or less than the root’s co-ordinates to enable us to go left or right down the tree to search for the matching city. To solve this I used brute force to traverse down both sides of tree. This meant forcing the function to go left and right every time it was called so that the function would always go down both sides of the tree regardless. This isn’t ideal but it had to be done in order to get down the tree.

Other solutions to the co-ordinates problem could have been to create a K-D Tree or a Quad Tree instead. The K-D Tree gives you any number of dimensions that you want, hence K. This tree gives 2 dimensional objects depth and creates a multi-dimensional space for them to grow.

The Quad Tree graphs 2 co-ordinates onto an X, Y, -X, -Y axis. It then gives each point a square based on their region (NE, NW, SE, SW). Every time a point is added to the graph a recursive call is made to make a square for the 2 points exact spot. Over time a region gets divided up into smaller regions, this enables each point to have its own square and to be found by traversing through the many smaller regions inside bigger regions.

I also noticed that in my maxHeight() function it was returning the height + 1. So to fix this, when the public maxHeight() calls the private maxHeight() it takes away one so:

**Int maxHeight() {**

**Return (maxHeight(root) ) – 1;**

**}**