Drew McLaurin, Diwash Biswa

Cpts 223

PA 4 Report

10/17/18

PA\_4 Report

* The goals of this assignment were implementing a simple board game using a balanced binary search tree with any of the data structures we have discussed. (In our case we used an STL map).
* The machine I ran this program on is a Dell Inspiron with an Intel core i7 processor at 2.70 GHz and 12 GB of RAM. I did my testing on a pc using Linux and PuTTY to compile the code with a g++ compiler.
* For our experiment we used an STL map for the game board. We implemented the player ID into the map key and we created a separate vector for the X and Y coordinates for the map value. This way we could access both the ID and the coordinates from the gameboard map. Since we implemented an STL map, which is a balanced binary tree, the run time complexity for our game board implementation was log(N). Below is a list of each of our main functions for this assignment along with their runtime complexity.

Insert() – O((logN)^2)

Search() – O(log(N))

PositionOccupied() – O(log(N))

MakeVector() – O(1)

Remove() – O(log(N))

Find() – O(log(N))

MoveTo() – O(Nlog(N))

PrintByID() – O(N)

The memory complexity for our program as a whole is O(N) because when implementing the map we do not allocate any memory. The only time we allocate space is when we insert into the map so the memory complexity is whatever the length of the map is since there is no extra space allocated. Even though the game board can have a million squares, we still don’t allocate memory until a player is inserted onto the board. So after the first insert the memory allocated would be 1.

* Our test results for TestBoard.cpp is along with the rest of our assignment files called *TestResults.txt*