Albert Wu

CptS 223

PA3 Report

**A: Problem Statement**

The goal of this exercise was to practice using binary search trees (specifically, C++ maps) in order to efficiently store and search for objects by the value of their integer data fields.

**B: Experimental Setup**

Tested on EECS Linux servers with:

g++ Board.cpp Player.cpp test\_board.cpp -o result -std=c++0x

**C: Algorithm Design**

I created a Player class consisting of an ID, an x-coordinate, and a y-coordinate. I used a C++ map (binary search tree implementation), indexed by ID, to store the Player objects.

In order to efficiently find a given position, I used a map indexed by x-coordinate, where the values were another map, indexed by y-coordinate, whose values were the player ID.

The insert method consisted of finding the right place in the ID-indexed map to insert the new player, which is O(log N) time complexity, where N is the number of players. I additionally had to ensure that a position was not already occupied, so I had to search through the x-indexed map to get the y-indexed map of all players at the position’s x-coordinate, in which I then searched for the position’s y-coordinate. Assuming the number of players at the same x-coordinate is a small constant, the time complexity is also O(log N). Thus, the complexity of the insert method overall is O(log N).

The remove method consisted of searching for the given player’s ID in the ID-indexed map then removing it, an O(log N) operation, and searching for the given player’s position in the x-indexed map of y-indexed maps of player IDs and removing the element from the y-indexed map, which is also O(log N) operation assuming the number of players at the same x-coordinate is a small constant.

The moveTo method consisted of only lookups, inserts, and removals, so the overall complexity was also O(log N).

The total memory complexity was O(N), as I stored a constant amount of data for each player.

**D: Test Results**

See the “test\_results.txt” file.