Richard Huber, 1207319785

CSE 310

Project 2 Report

1. In order to collect the data on the hash table and binary search trees that were implemented, I had to calculate a few things. First, I developed a simple iterative algorithm to print the length of each chain at all indices of the table. Then, I used my numOfBooks and tableSize variables to calculate the load factor of the table. I did this by hand; this is not represented in my program. In analyzing the binary search trees, I developed a recursive algorithm that counts the nodes in a tree given any root. This function was used in calculating the total number of nodes in the tree, and the number of nodes in the left/right sub-trees. Finally, I developed another recursive algorithm to count the number of nodes at any given height of the tree.
2. The size of my table ended up being 239, so while it would be impractical to graph the length of each chain for *all* 239 indices, I calculated the load factor to be about 0.4895, which is the average number of books per index of the table. This indicates that there are lots of unused indexes and that we are not creating extremely large chains. When I examined the lengths of these chains, I found that the highest number of books in a chain was 3. This is good, because we want the number of books in each chain to be small, so that searching is fast.



This graph represents the number of nodes in each binary search tree.



Similarly, this graph depicts the number of nodes in the left and right sub-trees, to see if the tree is well-balanced.





The software that I used to make these graphs only allowed for 6 samples per graph, and we had 9 genres, so I had to split this one into two. These two graphs represent the number of nodes at a given height of a tree.

Answers to questions:

1. I would say that this hash table is absolutely large enough to store all of the books. If anything, it may be a little too large; although I did not graph the length of each index’s chain because the graph would be massive, when I did inspect them, I saw that many indexes did not have any books saved in them at all, and as I said, the maximum number that I saw in one chain was 3, which is manageable.
2. I think that the hash function is evenly distributing the books throughout the table. The load factor was less than 0.5, and the chains were not long, so I would definitely say that this is an effective hash function.
3. The binary search trees actually don’t seem to be well balanced in terms of height. I graphed the height of the left and right sub-trees of each BST, and there tends to be a big difference in the sizes, indicating that they are not well-balanced.
4. I would not modify the hash table or hash function, but if I really wanted to make the searching in the binary search trees faster, I would change the structure to one of the trees we discussed in class (like a 2-3 tree or a red-black tree) in order to keep the trees balanced and make searching more efficient. Other than that, I think these structures work fine for the information we are storing.

Note: Because the graph would have been rather large, I did not graph the number of items in the linked lists for each index of the table, but I did leave *all* of my code used to carry out these investigations commented out at the bottom of my program, so the comments can be removed if you would like to see more about the lengths of each chain to validate my claims.