**CSCE 221 Assignment 3 Cover Page**

First Name: **Kaivan** Last Name: **Shah** UIN: **522006920**

User Name: **kks6920**  E-mail address: **kaivan29@tamu.edu**

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more on Aggie Honor.

System Office website: http://aggiehonor.tamu.edu/

|  |  |  |  |
| --- | --- | --- | --- |
| Type of Sources |  |  |  |
| People | Kyle Almeida | Brandon |  |
| Web Pages (provide URL) | http://www.cplusplus.com/ |  |  |
| Printed Material | Data Structures and Algorithms in C++ |  |  |
| Other Sources | PowerPoint |  |  |

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.

On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.

Your Name Kaivan Shah

Assignment Report 4

November 22, 2014

1. The main objective of the assignment is to write a C++ program that creates a binary search tree, and empirically calculates the average search cost for each node in such a tree, and removes a designated node. The following program uses 2 types of classes, one class is created to create a Binary Node, which has all the elements a node would consist for a tree, and Binary Search Tree would be another class type which is firstly made a friend class of Binary Node to access the private members of the class. The Binary search tree class would have all types of functions which would be able to insert a node, remove a node, traverse the tree and many more functions as required.

2. The data structure created in this project is a tree data structure, more specifically a binary tree data structure. A tree is a widely used abstract data type (ADT) or data structure implementing this ADT that simulates a hierarchical tree structure, with a root value and subtrees of children, represented as a set of linked nodes. The tree has different kinds within known as proper trees, complete trees and expanding trees which all have individual properties. The data is arranged in the ordered manner where the left node has a lower value then the root node, which would apparently have lower value then the right node. leftnode<rootnode<rightnode. The data is traversed in 3 different orders in the classes:

1. Inorder: <left><root><right>

2. Preordrer: <root><left><right>

3. Postorder: <left><right><root>

Thus this is basically how the data is stored, printed and read from the tree with the help of these three traversal funcitons.

3.

a) The individual search cost is calculated by setting a counter factor in the insert node and initializing it to zero, and then increment first when the new node is created, and then increment the count variable before the recursive functions are called. Then the search count is then displayed along the key in the inorder traversal function.

b) For the average search cost two functions are introduced in the Binary tree header file total and average, and these function recursively calculate the average search cost. It is then called in the main funciton where another count is set to calculate the total number of nodes inserted. So the averagecost is calculated by dividing the function called in main which is divided by the count set up in the main function, and thats how the search cost is calculated.

c) The updated cost is calculated when a node is removed and is implemented by creating a new function to update the search cost, and then the function is called in the traversal asked for by the user. The implementation of search cost, average search cost and the updated search cost is asscociated with the insert, find and remove function.

a) individual search cost: O(log(n))

b) updating the search cost: O(log(n))

c) Summing up: O(n^2)

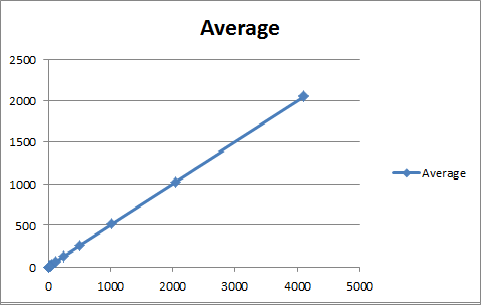
4. Perfect Binary Tree: O(log(n))

Linear Binary Tree: O(n)

|  |  |
| --- | --- |
| Number of nodes | Average |
| 1 | 1 |
| 3 | 2 |
| 7 | 4 |
| 15 | 8 |
| 31 | 16 |
| 63 | 32 |
| 127 | 64 |
| 255 | 128 |
| 511 | 256 |
| 1023 | 512 |
| 2047 | 1024 |
| 4095 | 2048 |

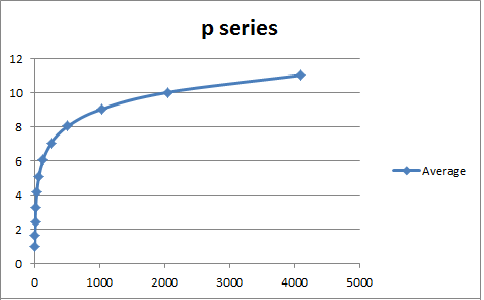
5.

a) Linear Tree Average Cost



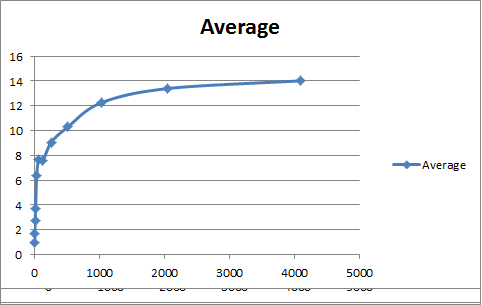
b) Perfect Tree Average cost

|  |  |
| --- | --- |
| Number of nodes | Average |
| 1 | 1 |
| 3 | 1.667 |
| 7 | 2.428 |
| 15 | 3.266 |
| 31 | 4.161 |
| 63 | 5.095 |
| 127 | 6.055 |
| 255 | 7.031 |
| 511 | 8.0176 |
| 1023 | 9.0098 |
| 2047 | 10.0054 |
| 4095 | 11.0029 |



c) Random Tree Average Cost

|  |  |
| --- | --- |
| Number of nodes | Average |
| 1 | 1 |
| 3 | 1.667 |
| 7 | 2.714 |
| 15 | 3.733 |
| 31 | 6.387 |
| 63 | 7.667 |
| 127 | 7.59 |
| 255 | 9.067 |
| 511 | 10.303 |
| 1023 | 12.246 |
| 2047 | 13.397 |
| 4095 | 14.023 |



**Instructions to Compile and Run your Program**

Compile the files with **g++ -std=c++11 -o \*.cpp**

To run them use: ./a.out