CSCE 221 Assignment 4 Cover Page

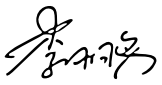
First Name Peng Last Name Li UIN 822003660

User Name abc E-mail address billlipeng@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				
Web pages (provide URL)				
Printed material	CSCE221 Slides			
Other Sources				

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  Date Nov 4, 2014

Program Description

In this lab, we wrote a C++ program that creates a binary search tree, calculates the average search cost for each node in such a tree, and removes a designated node.

Purpose of the Assignment

The purpose of this programming assignment is to learn tree structures, binary search tree algorithm and the basic components of a binary tree.

Data Structures Description

We used binary tree structure in this assignment. During the implementation of the binary tree structure, I learned how create a binary search tree. For each binary node, it has two pointers points to the left and right. Also, each node has a Key and a SearchCost. The BinarySearchTree class is a friend class of the BinaryNode, which has all the functions that are called in the main function.

Algorithm Description

1. insert(int x,BinaryNode *t, int& sc){}
   The insert function compares input x with current node. If it is greater than current node value then current node goes to the left. If it is smaller than current node value then current node goes to the right. When current node is NULL, insert the new node. The maximum comparison of insert function is the height of the tree. So the time complexity is O(h).

2. remove(int x, BinaryNode *t)
   The remove function will is very similar with the insert function. The function searches for node x based on comparisons. When the designated node is found, it substitutes the value of the node with the minimums value of the right sub-tree. If the node has only one sub-tree, then delete the node and move the sub-tree up. Else just delete the tree node. After delete the tree node, the search cost for each node in the tree need to be updated. The time complexity for the delete part is O(h), for the update search cost is O(n). Totally it is O(n+h).

3. resetSearchcost(BinaryNode* t, int i)
   This function will **update the search cost** of an individual node.

The reset search cost function will reset search cost for every nodes in the tree. Time complexity is O(n).

4. preOrderTraversal(BinaryNode *t, int& sc_total)
   This function will return the **summing up of the search cost.**
   The pre/post/in order traversal function is a recursive function. It will go through every node in the tree. The time complexity is O(n).

5. The **individual search cost** of a element in a binary tree is O(n) for worst case(linear tree) and O(logn) for best case(perfect tree).
   For the worst case, like linear tree, the height of the tree is n-1. So the total search cost is n(n+1)/2, divided by n is (n+1)/2, which is O(n).
   For the best case, like the perfect tree example, the height of the tree is log(n). For each level k, there are 2^k nodes. The total search time is (log1+1)+2(log2+1)+ 2^2(log3+1) +2^3log4 +.....2^log(n+1)-1(logn+1)= (n+1)log(n+1)-n. So the average search time is O(log(n)).

Program Organization and Description of Classes

```
class BinaryNode {
private:
    friend class BinarySearchTree;
    int Key;
    int SearchCost;
    BinaryNode *left, *right;
public:
    BinaryNode(int key = 0, int sc=0,BinaryNode *lt = NULL,BinaryNode
*rt = NULL) ://constructor
    Key(key),SearchCost(sc), left(lt), right(rt){}// functions
    BinaryNode *getLeft() {return left; }
    BinaryNode *getRight() {return right; }
};


class BinarySearchTree {
private:
    int node_num; // total num in the tree
    BinaryNode *root;
    BinaryNode *insert(int x, BinaryNode *t, int& sc);//insert x into
tree t
```

```
    BinaryNode *findMin(BinaryNode *t);
    BinaryNode *removeMin (BinaryNode *t);
    BinaryNode *remove(int x, BinaryNode *t);

    void resetSearchcost(BinaryNode* t, int i);
    void inOrderTraversal( BinaryNode *t, int& sc_total);
    void preOrderTraversal( BinaryNode *t, int& sc_total);
    void postOrderTraversal( BinaryNode *t, int& sc_total);
public:
    // constructor
    BinarySearchTree() { root = NULL; node_num=0; }
    bool isEmpty(){return root == NULL;}
    void remove(int x){ root = remove(x, root); } // remove Key = x
    void insert(int x) {
        int sc=0;
        root=insert(x, root, sc);
    }
    int inOrderTraversal(){
        int sc_total = 0;
        inOrderTraversal(root,sc_total);
        cout<<endl;
        return sc_total;
    }
    int preOrderTraversal(){
        int sc_total = 0;
        preOrderTraversal(root,sc_total);
        cout<<endl;
        return sc_total;
    }
    int postOrderTraversal(){
        int sc_total = 0;
        postOrderTraversal(root,sc_total);
        cout<<endl;
        return sc_total;
    }
    int getNodeNum(){return node_num;}

    void OutputTreeLevelByLevel();
    void Outputxt(string filename);
};
```

Instructions to Compile and Run your Program

- **Frist part**

  $ make
  $ ./Main
  The terminal will prompt ”Please input a filename: ”
  Input the filename you want to test. (A file call “example” has been submitted in my homework folder. So you can simply input “example”)
  Then the terminal will prompt ”Please enter a key to remove: ”
  Enter any key in the tree you want.
  Then the program will print out the new tree.

- **Second part**

  Please delete the comment start in the main function in order to run the second part of the code.
  $ make
  $ ./Main
  Please open “output_averageSC” to check the correctness of the second part.

Input and Output Specifications

When delete a node in the tree, please input the Key of the node.

Logical Exceptions

No logical error has been found in testing.

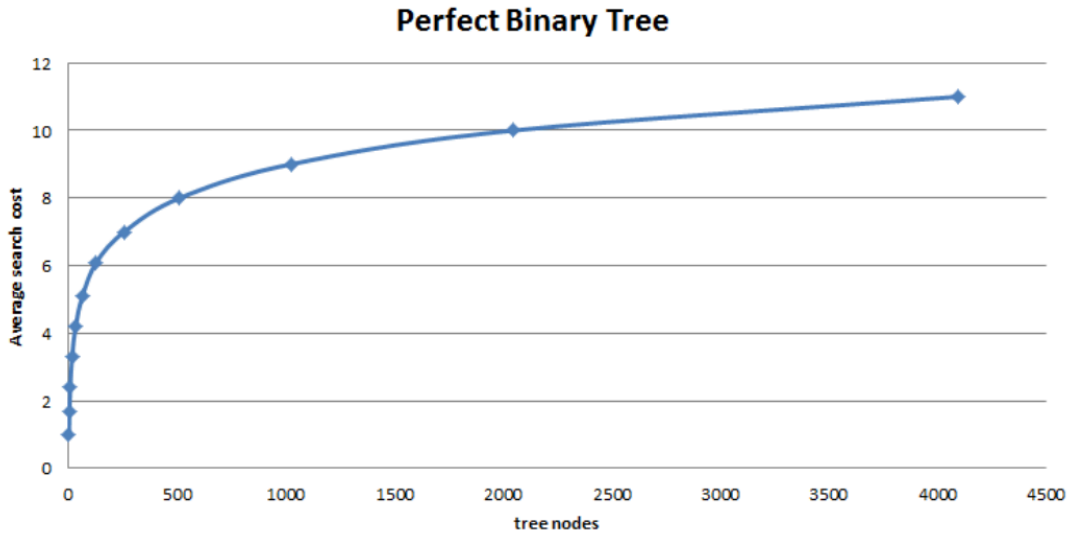
C++ object oriented or generic programming features

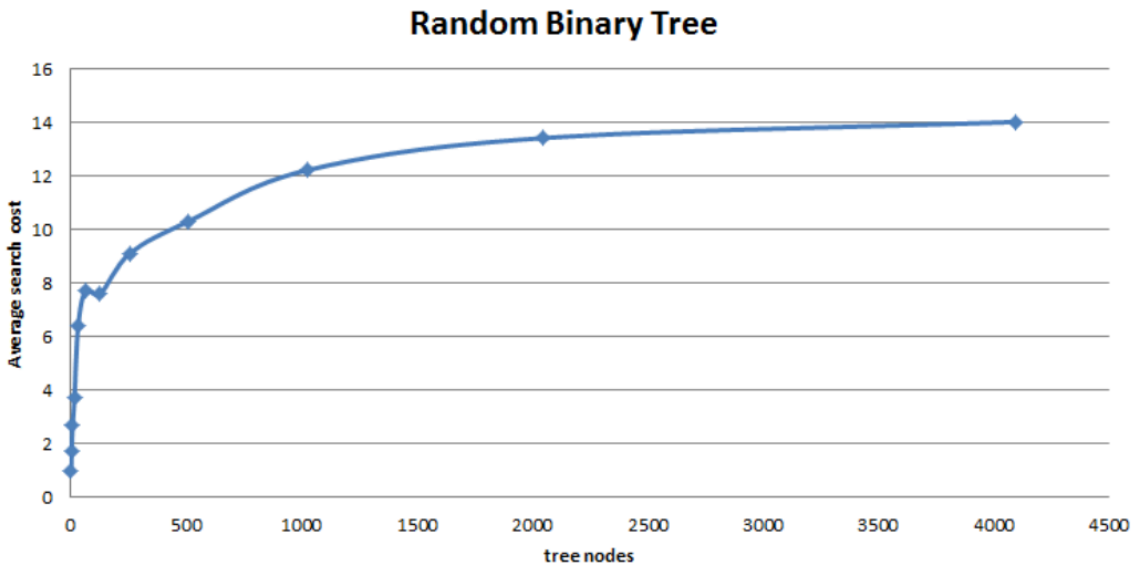
In this lab, object oriented programming features is used. The classes in this lab are shown in Program Organization and Description of Classes section of this lab report.

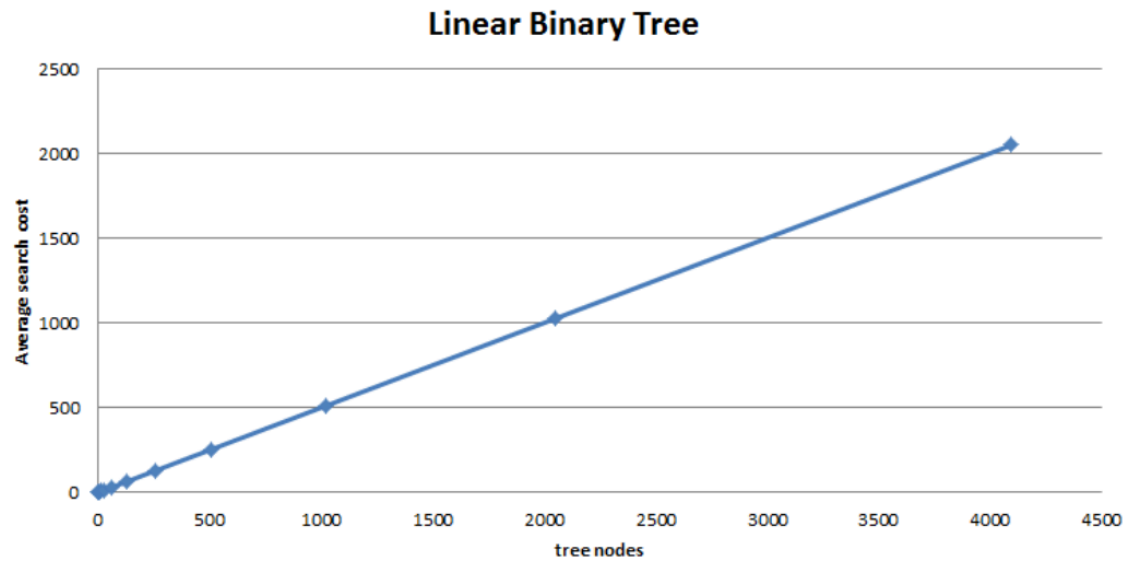
Charts and Graphs

From the charts and the graphswe can see the average search time for the linear search tree is 2n = O(n)(search cost goes linearly as number of nodes increase). The average search time for the perfect tree is O(log(n)) (search cost goes logthmly as number of nodes increase).

Nodes	Perfect	Random	Linear
1	1.00	1.00	1.00
3	1.67	1.67	2.00
7	2.43	2.71	4.00
15	3.27	3.73	8.00
31	4.16	6.39	16.00
63	5.10	7.67	32.00
127	6.06	7.59	64.00
255	7.03	9.07	128.00
511	8.02	10.30	256.00
1023	9.01	12.25	512.00
2047	10.01	13.40	1024.00
4095	11.00	14.02	2048.00




Random Binary Tree
Average search cost
tree nodes
0
2
4
6
8
10
12
14
16
0
500
1000
1500
2000
2500
3000
3500
4000
4500


Linear Binary Tree
Average search cost
tree nodes
0
500
1000
1500
2000
2500
0
500
1000
1500
2000
2500
3000
3500
4000
4500