Assignment 3: HashTable Tree

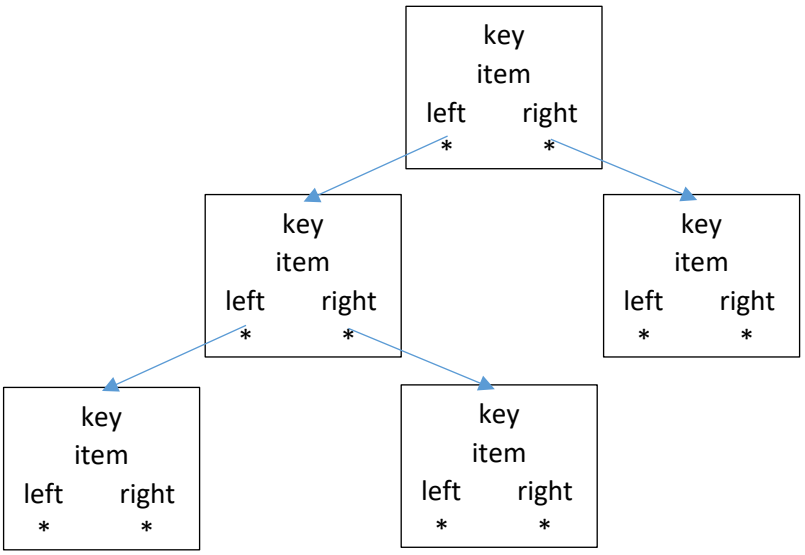
By Varick Erickson

Introduction

For this project you will implement a Hash Table using a Tree. This data structure is similar to the linked list hash dictionary described in the book. However, instead of storing the values in each "bucket" as a linked list, you will store the values in a binary tree.

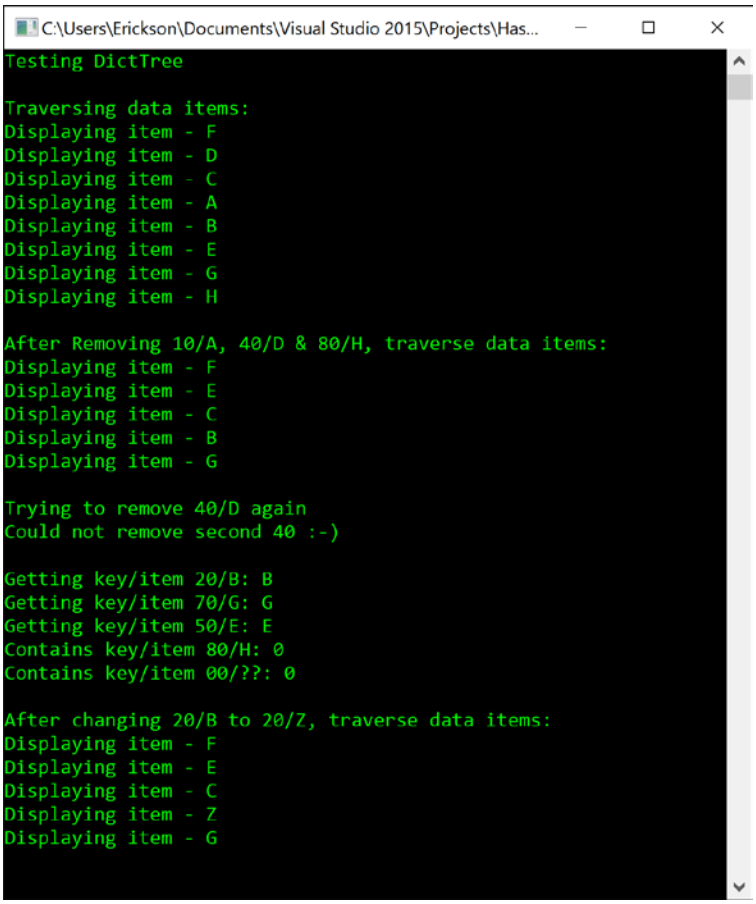
Part 1: DictTree

The dictionary tree class stores key/value pairs using a binary tree where the order is based on the value of the key. The implementation is essentially the same as a standard binary tree except the tree order is based on the key. The following is a representation of the data structure followed by the header file.



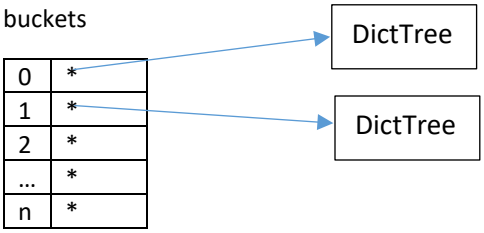
```
template <class KeyType, class ItemType>
class DictTree
{
private:
        TreeNode<KeyType, ItemType>* root;
        // any other functions you find useful

        int numNodes;
public:
        DictTree();
        ~DictTree();
        void add(KeyType newKey, ItemType newItem);
        bool remove(KeyType newKeys);
        bool contains(KeyType key);
        ItemType lookupItem(KeyType key);
        int size();
        void traverse(void visit(ItemType&)) const;
};
```

Your job is to implement each of the methods in the header class. You also need to define the Node as a class or struct. The implementation should be very similar to a standard binary tree. The following is the output if your dictionary tree is implemented correctly:



Part 2: HashTable

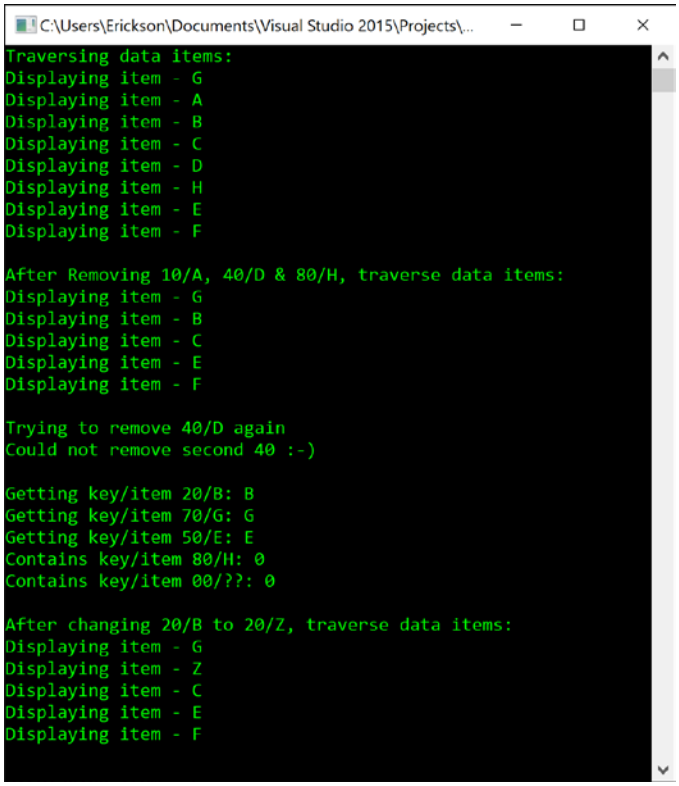
Once the tree dictionary is implemented, completing the HashTable implementation requires little additional code. The HashTable is implemented using an array of buckets where each of these buckets is a DictTree pointer:



When adding an item to the hash table using a key for the lookup, use of a hash function is required. This hash function takes the key value as input and returns the bucket the item should be associated with. The hash function is called as follows:

```
int b = getHashIndex(key);
```

b now stores the bucket index that the item associated with key is located. Refer to the notes given in the started code. The following is the output if the hash table is implemented correctly:



Test Driver

You should have a test driver similar to the list driver from the previous program. In particular, it should support the following commands:

- Add
- Remove
- GetItem
- Contains
- Size
- Print

IMPORTANT: Be sure to craft your input tests to ensure that your node removal works correctly.

Part 3: Questions

1. What is the advantages/disadvantages of a tree approach for each bucket rather than a linked list approach?
2. What is the HashTable big O of:
    - add
    - remove
    - getItem
    - contains
    - size
3. Suppose the hash function always returned the bucket value of 0. What is the HashTable big O of:
    - add
    - remove
    - getItem
    - contains
    - size

Deliverables

- Your template class implementations for the DictTree and HashTable with comments for each of the methods
- Your test driver
- Input to the test driver demonstrating functionality
- The output file generated from the test driver
- Answers to the questions for part 3