CS 293 : Data Structures and Algorithms Lab

Lab 3 : Binary Search Trees

(and how to go about storing them)

Part 1 : Building a BST

After the previous week of scheduling train journeys, it's time we put the tickets and pricing into place. We require a system in place that can give our travelers, journey suggestions based on their budget. For this assignment, we shall assign codes to journeys (instead of using station names). And each journey shall be associated with its cost.

Using the concepts learned in CS213, design a Binary Search Tree that stores at each node, a

$$key, value$$

pair of

$$JourneyCode, Cost$$

.

Each node stores the Journey code and the cost associated with it.

You are provided with `struct TreeNode` that embodies most of the attributes you will be requiring in a particular Node.

You might may changes to the header file. Both to `struct TreeNode` as well as `class BST`, but make sure you do not remove any of the past attributes and only add new ones.

Feel free to refer to Prof. Naveen Garg's slides.

As usual, you will need to create another file `BST.cpp`, and with the usual syntax define the following functions :

You may assume that all pairs of *(JourneyCode,Cost)* that we insert are unique.

You need to implement the following functions :

1. `insert(int JourneyCode, int price)` : Insert a new node in your Binary Search Tree, representing this Journey with appropriate Journey Code and Cost, as mentioned.
2. `find(int JourneyCode, int price)` : Check if this *(JourneyCode,Price)* is present in your BST, and return True/False accordingly.
3. `remove(int JourneyCode, int price)` : Check if this Node is to be found in the BST, Delete it, if you can.
4. `traverse(int type)` : Traverse the BST in different ways depending on what `type` is.
5. `getMinimum()` : Return the price of cheapest journey.

Implementing these fully would mean that your tree is now up and running.

It should be clear by now that, you should use the price as the ordering attribute in your BST. You may assume that both the *price* and *JourneyCode* are positive integers.

Part II : Bounds, Store and Load

We would also like you to implement a rather advanced feature in your BST design i.e. the ability to know how many journeys lie within a particular range of prices.

Go ahead and implement :

`countJourneysinPriceBound(int lowerPriceBound, int upperPriceBound)` : Returns the number of tickets which are at least as expensive as `lowerPriceBound` and only as expensive as `upperPriceBound`.

Next Up, You would have learnt in CS 213 about how given the pre/post-order and in-order traversal of any Binary Tree, you can construct the original tree.

Do you think, you can come up with an even better strategy to do the re-construction in Linear Time perhaps?

Can you make some modifications in pre-order traversal itself which might give you sufficient information for re-construction?

We would want you to come up with a method that could store a tree in $O(n)$ time and space (n being the number of nodes in the BST). Furthermore, we'd also want you to come up with a function that could load either the whole BST or just the left-subtree optimally.

You might want to get accustomed to basic I/O functions at this point.

It is quite simple in C++.

If you want to write to a file :

```
#include <fstream>

using namespace std;

int main(){
    ofstream outfile;
    outfile.open("foo.txt");
    outfile << "Hello, World"<<endl;

    // You may treat outfile as cout now!

    int x = 5, y = 6;
    outfile << x + y <<endl;

    // Finally, When you're done, Just close the file.

    outfile.close();
}
```

Reading from a file is similar :

```
#include <fstream>

using namespace std;

int main(){
    ifstream outfile;
    infile.open("foo.txt");

    int N;
    infile >> N;

    // You may treat infile as cin now!

    // Finally, When you're done, Just close the file.

    infile.close();
}
```

Hint for Storing in Linear-Time/Memory : Think about the issues if you store the BST with just a pre-order traversal. Could something like a sentinel character fix these?

Testing for Part I

As usual, run `make`, to build the two files `interact` and `automated`.

For an interactive testing environment. You might enter 1 and test the storing-loading part OR Press 0 and interactively test the first part of this lab.

You will need the following options for that :

1. `ADD <JourneyCode> <Price>` : Inserts a Journey with given Journey Code and Price into your Data Structure.
2. `FIND <JourneyCode> <Price>` : Tells you if a particular entry is present in your Data Structure.
3. `DEL <JourneyCode> <Price>` : Deletes an entry (if it exists)
4. `TRAVERSE <TYPE>` : Prints a particular traversal of the given tree.
    - TYPE : `PRE` : Prints the Pre-order traversal of your tree
    - TYPE : `POST` : Prints the Post-order traversal of your tree
    - TYPE : `IN` : Prints the In-order traversal of your tree
5. `GETMIN` : Returns the price of the cheapest Journey.
6. `PRINT` : Prints the BST on your screen

You might run the interactive version by running the command `./interact` while in the same directory.

You might manually automate the checking on large test cases by copying some of the files from test-cases into your active directory and running : `./automated < input0.txt > out0.txt`

After that you can compare the files `out0.txt` (Your output) with `output0.txt` (Expected Output) by running : `diff out0.txt output0.txt`

We will be releasing code for testing of Part II during the weekend.

Enjoy :)