SquareRootCalculator: a key-value data structure

CompSys.202 / MechEng.270

Due to security reasons, the WebIDE version of ACP does not allow code to run for a long time. So, use a local machine to run this lab exercise.

Introduction

We have not looked into *key-value* data structures (i.e. **maps**) in this course, but we will now get an idea of how one might work. The difference between a map, from say a list, is that we add **pairs** to the collection. What does this mean? Assume we want to create a dictionary of names, and each name containing a phone number. What would be an efficient way to store these? In a list, we typically use indexes (an integer starting from zero) to help us associate the ordering of the elements. But if we have name:phoneNumber pairings, then this isn't going to be much help. What would be useful is a way to quickly "lookup" a name, and then immediately get the associated number. We will simplify this, by using an int:double association in our next data structure. But rather than using a list (since the numbers might get large, random and sparse), a more efficient data structure will be the **binary search tree**.

Your task is to build a map (using an int:double pairing) using a binary search tree as implementation. We will make it a little more interesting, by associating it with a specific problem: calculating the square root of a given integer. Here is the main class:

```
SquareRootCalculator sqRoot;
double result = sqRoot.calculate(196);  // takes a long time to compute
cout << result << endl;

result = sqRoot.calculate(225);  // takes a long time to compute
cout << result << endl;

result = sqRoot.calculate(196);  // this should be instantaneous!
cout << result << endl;
```

What happens is we invoke the `calculate()` function of the `SquareRootCalculator` class (which represents our map implemented as a binary search tree). The first time we ask it to calculate the square root of 196, it takes a long time. If we try a different and fresh number (the larger the number, the more time it needs), then it also takes a long time. However, we would like this class to be intelligent when we ask it to give us the square of a number we previously calculated (e.g. 196 again). In such a case, it should do a quick lookup to see if it has the result for 196 saved. If it does, then it returns it without re-calculating it.

An efficient storage

The next question then is, how do we store the previously calculated results? If we used an array, a vector, or a linked list, this is not a very efficient solution, especially when we have a large number of numbers to square root, and those values are in a random order and likely to be dispersed. Instead, we will use a binary search tree to implement the map. See the lecture slides for more details on the BST.

The code provided

You are given 5 files:

SquareRootCalculator.h Header file declaring our square root calculator class (implemented as a simple BST)

SquareRootCalculator.cpp Definition of the square root calculator class

Node.h A BST node (i.e. has **left**/**right** children) that stores a **key** and **value**

Node.cpp Definition of the Node class

main.cpp: The main testing file. Creates a calculator and gets the square root of some numbers. The time taken for each number is also printed.

What to do

You need to only modify `SquareRootCalculator.cpp`, by completing the empty methods with "todo" comments. Make sure you read the comments clearly, as they help you with hints. The expected output of your program should be similar to the following. Of course, the actual timings might be different (depending on your computer), but the ones with `[0s]` should be the same:

```
SquareRoot(81) = 9 ... [3s]
SquareRoot(25) = 5 ... [2s]
SquareRoot(50) = 7.07107 ... [2s]
SquareRoot(36) = 6 ... [3s]
SquareRoot(81) = 9 ... [0s]
SquareRoot(200) = 14.1421 ... [5s]
SquareRoot(60) = 7.74597 ... [2s]
SquareRoot(81) = 9 ... [0s]
SquareRoot(200) = 14.1421 ... [0s]
SquareRoot(169) = 13 ... [5s]
SquareRoot(200) = 14.1421 ... [0s]

Total time for these 11 numbers: 22 seconds.

Values stored:
--------------
[81: 9]
[25: 5]
[50: 7.07107]
[36: 6]
[60: 7.74597]
[200: 14.1421]
[169: 13]
```

Notice how calculating SquareRoot(200) the second (or third...) time is instantaneous. Also notice that the values stored do not include duplicates.

Hints:

- To speed up your testing, comment out some of the numbers in your **main.cpp** file so that you don't have to wait for it to calculate them all.
- You do not have a tree to start off with, so first implement the **calculate()** function so that it calculates the square root from scratch all the time to make sure it's all working.
- The comments in the code that you have to write form pseudocode . So try and make the most of it.
- Almost all the functions you have to write are recursive! Have a look at **removeChildrenNodes()**, as it is also recursive. But the other functions will still be very different.