COSE213 Data Structures (Fall 2021)
Instructor: Prof. Won-Ki Jeong
Due date: Dec 5, 2021, 11:59 pm.

Assignment 4: Binary Search Trees (total 100 pts)

In this assignment, you will implement two binary search trees and test the performance of the search trees.

1. Binary search trees implementation (50 pts)

Three binary tree classes are provided in the skeleton code:

- `LinkedBinaryTree` : base class for binary trees
- `SearchTree` : binary search tree
- `AVLTree` : hight-balanced binary search tree

`.h` files are the interface definitions (header), and `.txx` files are their implementation. You are required to fill in ToDo parts in `txx` files only, and you are not allowed to modify the header files. Therefore, you only need to submit the following four files: `SearchTree.txx`, `AVLTree.txx`, `main.cpp`, and a report (pdf).

In `txx` files, you are required to implement the following functions:

a. `SearchTree.txx`

- `Iterator& operator++()`
- `TPos eraser(TPos& v)`
- `TPos finder(const K& k, const TPos& v)`
- `TPos inserter(const K& k, const V& x)` // replace if k exists

b. `AVLTree.txx`

- `void erase(const K& k)`
- `Iterator insert(const K& k, const V& x)` // replace if k exists
- `void rebalance(const TPos& v)`
- `TPos restructure(const TPos& v)`

Note:

1. `insert` operation in `SearchTree` and `AVLTree` must replace the existing key rather than duplicating same keys.
2. You will be able to find most of the implementation of the above functions from our textbook, so you may use them. However, simple cut-and-paste would not work because the code in the textbook may have some errors/bugs. Make sure your code works correctly.
3. You can freely modify `main.cpp` for your experiment (see below)

2. Performance comparison of binary search trees (50 pts)

As you may realize, `SearchTree` is not height-balanced while `AVLTree` is height-balanced. Therefore, the performance of the trees might be different.

To check the performance difference, you need to conduct the following experiments and report the result.

(a) `insert` operations only
(b) `find` operations only

For this test, you must generate two types of test sets (keys are in skewed or random orders) with different sizes (1000, 10000, 100000, and 1000000). For example, if you generate test sets of size 10, then you may generate the following two sets of keys:

Skewed order : {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Random order : {3, 10, 1, 4, 6, 8, 7, 2, 9, 5}

You can assume the key type is `unsigned int`, and the value type is `float`, and assign any random float number as a value for a given key. Run (a) and (b) experiments using the generated test sets, and measure the running time using `clock()`. The provided `main.cpp` contains some examples how to generate random data and use clock to measure time.

For experiments (b), you must start with a search tree that is already filled with a test set from experiment (a).

For accurate timing measurement, you must create random datasets before

running the experiments. That means, you must measure the running time of `insert/find` operations only and do not include the time for initializing the datasets.

You must submit the modified `main.cpp` file you used for this experiment, and write a ***report*** (in pdf format) describing your experiment and the result in detail (draw graphs/tables to compare the performance of the search trees).

3. Compile and submit

You must submit the code online via blackboard. You can compile the code as follows:

> make

The output executable name is assign_4. You can run your code by simply type in this name in the terminal.

> assign_4

You need to submit the below four files only. When you submit your files, please change the filename to contain your student ID as follows:

```
2021XXXXXX_SearchTree.txx
2021XXXXXX_AVLTree.txx.
2021XXXXXX_main.cpp.
2021XXXXXX_report.pdf
```

Good luck and have fun!