COSE213 Data Structures (Spring 2021)
Instructor: Prof. Won-Ki Jeong
Due date: June 20, 2021, 11:59 pm.

Assignment 4: Binary Search Trees (total 150 pts)

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

1. Binary search trees implementation (60 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 the 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 (90 pts)

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

In order to see the performance difference, you need to conduct the following experiment and report the result.

(a) `insert` only
(b) `find` only
(c) `erase` and `find` in a random order

For each test, you must generate random test sets with different sizes (100, 1000, 10000, 100000, and 1000000). Assume the key type is `unsigned int`, and the value type is `float`. Run (a), (b) and (c) experiments using the random 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) and (c), you must start with a search tree that is already filled with a test set from experiment (a).

For experiment (c), you should test various `erase/find` ratios (for example, 20:80, 40:60, 50:50, 60:40, and 80:20). You must apply `erase` and `find` in a random order with a random dataset.

For accurate timing measurement, you must create random datasets before running (a)~(c) experiments. That means, you must measure the running time of `insert/erase/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!