CS 202, Fall 2021

Homework #3 – Heaps and AVL Trees

Due Date: December 3, 2021

Question 1 (50 points)

Use the given file names and function signatures during implementation. Feel free to write helper functions to accomplish certain tasks but remember to give meaningful function names.

(a) [15 points] Implement an array-based max-heap data structure named as MaxHeap for maintaining a list of integer values that supports the following operations:

Listing 2: MaxHeap

```
void insert (int value ); // inserts integer into heap
int peek (); // returns the value of the max element
int extractMax(); // retrieves and removes the max element
int size (); // returns the number of elements in the heap
```

Put your code into MaxHeap.h and MaxHeap.cpp files.

(b) [15 points] Implement an array-based min-heap data structure named as MinHeap for maintaining a list of integer values that supports the following operations:

Listing 3: MinHeap

```
void insert (int value ); // inserts integer into heap
int peek (); // returns the value of the min element
int extractMin(); // retrieves and removes the min element
int size (); // returns the number of elements in the heap
```

Put your code into MinHeap.h and MinHeap.cpp files.

(c) [20 points] Design a MedianHeap data structure reusing MaxHeap and MinHeap, which supports getting median of the elements in O(1) time and inserting an element in O(log n) time. If the length of the list is even, choose the larger value of the two middle elements as the median.

Listing 4: MedianHeap

```
void insert (int value );  // inserts an element into median heap
int findMedian();  // returns the value  of the median
```

Put your code into MedianHeap.h and MedianHeap.cpp files.

(d) [Mandatory to get points from part a,b,c] Test your implementations in a file named main.cpp.

Question 2 (50 points)

Implement AVL tree data structure named as AVLTree for maintaining a list of integer keys with the following methods:

void insert(int value); //inserts an element into the tree
int getNumberOfRotations(); //returns the number of rotations performed so far while constructing the tree

After that you will analyze the average number of rotations and determine if different patterns of insertion affect it. Write a global function, void rotationAnalysis() which does the following:

(a) Creates 1000 random numbers in the range [1,100000] and inserts them into an empty AVL tree. While inserting elements into the AVL tree, it counts the number of rotations and it outputs the total number of rotations in the end. Repeat the experiment for the following sizes: {2000, 3000, 4000,..., 10000}

(b) Instead of creating arrays of random integers, create arrays with elements in the same range in ascending order and repeat the steps in **part (a)**.
(c) Instead of creating arrays of random integers, create arrays with elements in the same range in descending order and repeat the steps in **part (a)**.

Put function's code into analysis.h and analysis.cpp files. Create a main.cpp file which calls rotationAnalysis() function. When the rotationAnalysis() function is called, it needs to produce an output similar to the following one:

Array Size	Random	Ascending	Descending
1000	x	x	x
2000	x	x	x
3000	x	x	x
...	x	x	x

After running your program, you are expected to prepare a report about the experimental results that you obtained in Question 2. In your report, you need to discuss the following points:

- Do your findings related to average number of rotations in the AVL tree agree with theoretical results?

- Do different patterns of insertion affect the number of rotations in the AVL tree? If so, explain how. If not, explain why not.

Important notes

- Before 23:55, December 3rd, upload your solutions in a single **ZIP** archive using the Moodle submission form. Name the file as **studentId_hw3.zip**.

- Your **ZIP** archive should contain the following files:

    - **hw3.pdf**, the file containing the answers to Question 2

    - A folder named **q1** which includes **MaxHeap.h**, **MaxHeap.cpp, MinHeap.h, MinHeap.cpp, MedianHeap.h, MedianHeap.cpp** and **main.cpp** files containing the C++ source codes and **Makefile** which compiles all your code and creates an executable file named **q1**

    - A folder named **q2** which includes **AVLTree.h, AVLTree.cpp, analysis.h, analysis.cpp** files and **main.cpp** files containing the C++ source codes and **Makefile** which compiles all your code and creates an executable file named **q2**

    - **README.txt**, the file containing anything important on the compilation and the execution of your program in Question 1 and 2.

- Do not forget to put your name, student ID, and section number in all of these files. Well comment your implementation. Add a header as the following to the beginning of each file:

```
/*
*   Title : Heaps and AVL Trees
*   Author : Name Surname
*   ID : 21000000
*   Section : x
*   Assignment : 3
*   Description : description of your code
*/
```

- Do not put any unnecessary files such as the auxiliary files generated from your favorite IDE. Be careful to avoid using any OS dependent utilities.

- You should prepare the answers of Question 2 using a word processor (in other words, do not submit images of handwritten answers).

- Although you may use any platform or any operating system to implement your algorithms and obtain your experimental results, your code should work on the **dijkstra** server. We will compile and test your programs on that server. Thus, you may lose significant amount of points if your C++ code does not compile or execute on the dijkstra server.

Attention: For this assignment, you are allowed to use the codes given in our textbook and/or our lecture slides. However, you ARE NOT ALLOWED to use any codes from other sources (including the codes given in other textbooks, found on the Internet, belonging to your classmates, etc.). Furthermore, you ARE NOT ALLOWED to use any data structure or algorithm related function from the C++ standard template library (STL).

Do not forget that plagiarism and cheating will be heavily punished. Please do the homework yourself.