University of Waterloo
CS240 Spring 2021
Assignment 3

Written Questions Due: Wednesday, June 30 at 5:00pm
Programming Question Due: Wednesday, July 7 at 5:00pm

The integrity of the grade you receive in this course is very important to you and the University of Waterloo. As part of every assessment in this course you must read and sign an Academic Integrity Declaration before you start working on the assessment and submit it **before the deadline of June 30th** along with your answers to the assignment; i.e. **read, sign and submit A03-AcInDe.txt now or as soon as possible**. The agreement will indicate what you must do to ensure the integrity of your grade. If you are having difficulties with the assignment, course staff are there to help (provided it isn't last minute).

The Academic Integrity Declaration must be signed and submitted on time or the assessment will not be marked.

Please read `http://www.student.cs.uwaterloo.ca/~cs240/s21/guidelines.pdf` for guidelines on submission. **Each question must be submitted individually to MarkUs as a PDF** with the corresponding file names: a3q1.pdf, a3q2.pdf, ... , a3q5.pdf, avlskip.cpp .

It is a good idea to submit questions as you go so you aren't trying to create several PDF files at the last minute.
Remember, late assignments will not be marked.
Late assignments, however, can be reviewed for **feedback only** upon request to the ISAs at cs240@uwaterloo.ca.
Note: you may assume all logarithms are base 2 logarithms: $\log = \log_2$.

You may use the following notation for *AVL* ADT in any of your solutions: The key stored in a node v is denoted by $key(v)$; the root of T is denoted by $root(T)$; parent node, left and right child of a node v is denoted by $parent(v)$, $leftchild(v)$ and $rightchild(v)$ respectively; the height of the subtree rooted at v is denoted by $height(v)$.
You may assume that all of them are of $O(1)$ running time.

Problem 1 [3+3+3=9 marks]

Let T_h be an AVL tree of height h with **minimum** number of nodes. Let $N(h)$ be the number of nodes and $N_L(h)$ be the number of leaves of T_h, respectively.
Note that T_h can be viewed recursively as a tree containing a root r with two, possibly empty, subtrees T_{h-1} and T_{h-2}.

a) Prove that for $h \geq 1$, $N(h) = N(h-1) + N_L(h)$.

b) Prove that for $h \geq 0$, $N_L(h) = F(h+1)$, where $F(i)$ is the *ith Fibonacci* number. $(F(0) = 0, F(1) = 1, F(2) = 1, F(3) = 2 \ldots)$

c) Prove that for $h \geq 0$, $N(h) = F(h+3) - 1$. You must use parts a and b to prove c.

Note that the analysis in this question can help to put a tighter bound on the height of AVL tree.

Problem 2 [10 marks]

Given two *AVL trees T1* and *T2*, where the largest key in *T1* is less than the smallest key in *T2*, *Join(T1,T2)* returns an *AVL tree* containing the union of the elements in *T1* and *T2*. Give an algorithm (in pseudocode) for *Join* that runs in time $O(\log n)$, where n is the size of the resulting *AVL tree*. Justify the correctness and efficiency of your algorithm.
Assume the height of *T1* is greater than or equal to the height of *T2*; the other case is symmetric.

Problem 3 [12 marks]

Design an ADT containing integers that supports the following methods. Show in pseudocode how to implement each of these methods in the requested time. You must solve this problem with $O(n)$ auxiliary space.

method	purpose	time
init()	initialize the ADT	$O(1)$
insert(x)	insert x into ADT, if it is not in the ADT yet	$O(\log n)$
delete(x)	delete x from ADT, if it exists	$O(\log n)$
delete_ith(i)	delete the element from ADT which has timestamp equals to i as determined by the timestamp order of insertion	$O(\log n)$
get_index(x)	return the place/index (which is determined by the order of insertion) of x. if x doesn't exist, return -1	$O(\log n)$

Note1: timestamps are assumed to take $O(1)$ space.
Note2: timestamp starts with 0.
For example, after initialization, and calling *insert(3)*, *insert(5)*, *insert(11)*, *insert(4)*,*insert(7)*,*delete(5)*; *get_index(7)* returns 4, and *delete_ith(2)* will delete 11 from the ADT.

Problem 4 [6+6=12 marks]

a) Draw a diagram of a skip list starting with an empty one, insert the seven keys $67, 28, 64, 66, 60, 81, 49$. Use the following coin tosses to determine the heights of towers (note, not every toss is necessarily used):

$$T, T, H, H, T, H, T, H, H, T, H, H, T, T, H, T, H, H, T, T, H, H, H, T, \ldots$$

b) The worst case time for searching in a singly linked list is $\Theta(n)$. Now consider a variation of a skip list which has fixed height $h = 3$ even though n can become arbitrarily large. Level S_0 contains the keys $-\infty, k_1, k_2, \ldots, k_n, \infty$. Level S_3 contains only $-\infty$ and ∞. Describe subsets of keys that should be included in levels S_1 and S_2 so that searching in the skip list has worst case runtime of $\Theta(n^{1/3})$. Provide justification for the runtime of your skip list. You may assume that n is a power of 3.

Problem 5 [3+3+3=9 marks]

Consider the list of keys:

[1 2 3 4 5 6 7 8 9 10]

and assume we perform the following searches:

9, 5, 2, 6, 4*, 4, 1, 5, 3, 9*, 2, 8, 2, 9*

a) Using the move-to-front heuristic, give the list ordering after the starred ($\star$) searches are performed. Additionally, record the number of comparisons between keys after each search, as well as the total number of comparisons.

9	5	2	6	4	4	1	5	3	9	2	8	2	9	Total

b) Repeat part (a), using the transpose heuristic instead of the move-to-front heuristic.

c) Another heuristic is *move-to-front2 (MTF2)* that is similar to *move-to-front (MTF)* except that when an element is found at position i it is moved to be in position $\lceil \frac{i}{2} \rceil$ while the old element in that position becomes in position $\lceil \frac{i}{2} \rceil + 1$ and so on (for example, after the first search, the list becomes 1,2,3,4,9,5,6,7,8,10). The first element in the list is considered to be in position 0. Repeat part (a), using this heuristic.

Problem 6 [12 marks]

In this programming question you are asked to implement both an AVL tree and a skiplist to compare the number of key comparisons made as items are inserted, searched for and deleted. Your implementation must follow the algorithms given in lecture notes so the number of key comparisons will match. Your runtimes should also match those discussed in class. For simplicity, the key and value of an item will be int values. Also, you do not need to consider duplicate key values within the data structures - we will not attempt to insert a key that already exists in the data structure.

You may not use any pre-existing code that would trivialize the implementation (e.g. built in data structures from STL, smart pointers, etc). You may, however, use the C++ vector data structure. If in doubt, make a private Piazza post and ask.

Implement your program in C++ and provide a main function that accepts the following commands from stdin (you may assume that all inputs are valid):

- `i key value coin` - inserts an item $(key, value)$ into both your AVL tree and skiplist and prints the number of key comparisons required by AVL.insert and skipList.insert, respectively, followed by a newline. Use the value *coin* as the number of heads flipped before a tail is reached when inserting into the skiplist.

- `d key` - deletes the item with the given key from both the AVL tree and skiplist and prints the number of key comparisons required by AVL.delete and skipList.delete, respectively, followed by a newline. For a BST, delete should use the inorder successor (if needed).

- `savl key` - searches the AVL tree for the item with the given key and prints the corresponding value or string `ERROR` if not found, the number of key comparisons used in the search, followed by a newline.

- `ssl key` - searches the skiplist for the item with the given key and prints the corresponding value or string `ERROR` if not found, the number of key comparisons used in the search, followed by a newline.

- `stats` - prints 3 int values followed by a newline corresponding to: number of items in the AVL tree/skiplist, height of the AVL tree, and total number of nodes in the skiplist (do not count $+\infty$ or $-\infty$).
  The runtime for this operation is $\Theta(1)$ time.

- `r` - initializes an empty AVL tree and skiplist. If a nonempty AVL tree or skiplist already exists, they are destroyed and new empty data structures are created.

Note: check the sample output. There is a single space between each two values printed on the same line.
Place your entire program in the file `avlskip.cpp`