Homework #6

RELEASE DATE: 05/20/2020

DUE DATE: 06/09/2020, **noon** on Gradescope/CEIBA

As directed below, you need to submit your code to the designated place on the course website.

Any form of cheating, lying or plagiarism will not be tolerated. Students can get zero scores and/or get negative scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.

Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from.

Since everyone needs to write the final solutions alone, there is absolutely no need to lend your homework solutions and/or source codes to your classmates at any time. In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.

Both English and Traditional Chinese are allowed for writing any part of your homework (if the compiler recognizes Traditional Chinese, of course). We do not accept any other languages. As for coding, either C or C++ or a mixture of them is allowed.

This is the last homework and hence your last chance to use your medals, if you still have any! Everyone has 8 medals in total this semester.

This homework set comes with 200 points and 20 bonus points. In general, every homework set of ours would come with a full credit of 200 points.

6.1 Skip Lists and Binary Search Trees

(1) (20%) Do Exercise C-9.15 of the textbook.

(2) (20%) Do Exercise R-10.5 of the textbook.

(3) (20%) Do Exercise R-10.6 of the textbook.

(4) (20%) Do Exercise C-10.20 of the textbook.

(5) (20%) Do Exercise R-10.18 of the textbook.

(6) (Bonus 20%) After sequentially inserting 1, 2, 3, ..., $2^k - 1$ into an AVL tree, where k is an integer, prove or disprove that the resulting AVL tree is a complete binary tree.

6.2 Balanced Binary Search Trees

(1) (30%) `libavl` (`http://adtinfo.org/`) is a useful library for binary search trees. For instance, the following short code constructs an AVL tree of 16 integers and print it out.

```
#include <stdio.h>
#include <stdlib.h>
#include "avl.h"
void postorder_integer_avl(const struct avl_node *node){
  if (node == NULL)
    return;
  printf ("%d ", *((int *)node->avl_data));
  if (node->avl_link[0] != NULL || node->avl_link[1] != NULL){
      putchar('(');
      postorder_integer_avl(node->avl_link[0]);
      putchar(',');
```

```
      putchar(' ');
      postorder_integer_avl(node->avl_link[1]);
      putchar(')');
    }
}

int int_compare(const void *pa, const void *pb, void *param)
{
  int a = *(const int *)pa;
  int b = *(const int *)pb;

  if (a < b) return -1;
  else if (a > b) return +1;
  else return 0;
}

int main(){
  struct avl_table *tree;
  tree = avl_create(int_compare, NULL, NULL);

  int i;
  for(i=0;i<16;i++){
    int* element = (int*)malloc(sizeof(int));
    *element = i;
    void **p = avl_probe(tree, element);
  }

  postorder_integer_avl(tree->avl_root);
  puts("");
  return 0;
}
```

Note that the manual of `libavl` may be difficult to read; the code above comes from modifying the `avl-test.c` in `libavl`. When the code is compiled with `avl.c`, it correctly outputs an AVL tree.

```
7 (3 (1 (0 , 2 ), 5 (4 , 6 )), 11 (9 (8 , 10 ), 13 (12 , 14 (, 15 ))))
```

Write a program `hw6_2_1` that inserts strings using the order they are provided to the height-bounded binary search tree (`bst.c`), AVL tree (`avl.c`) and Red-Black tree (`rb.c`). Please feel free to modify `libavl` to fit your use. Output the resulting trees with a format like the output above. The TAs will announce more detailed specs later.

(2) (30%) Write a program `hw6_2_2` that inserts 1, 2, 3, ..., 2048 as integers into the three different trees above. Then, remove the first 1024 keys you generated from each tree. Finally, add 2049, 2050, ..., 4096 into each tree. Record the height of the trees that you get (a) after the insertion (b) after the removal. Then, fill in the following the table and briefly state your findings.

tree type	height after operations
height-bounded binary search tree	
AVL tree	
Red-Black tree	

(3) (40%) Write a program `hw6_2_3` that randomly generates 2048 signed integers between before they are inserted into the three different trees above. Then, remove the first 1024 keys you generated from each tree. Finally, add 2048 random signed integers into each tree. For 10000 different random rounds, record the height of the trees that you get (a) after the insertion (b) after the removal. Then, fill in the following the table and briefly state your findings.

tree type	maximum height after 1st insertion	minimum height after 1st insertion	average height after 1st insertion
height-bounded binary search tree			
AVL tree			
Red-Black tree			

tree type	maximum height after removal	minimum height after removal	average height after removal
height-bounded binary search tree			
AVL tree			
Red-Black tree			

tree type	maximum height after 2nd insertion	minimum height after 2nd insertion	average height after 2nd insertion
height-bounded binary search tree			
AVL tree			
Red-Black tree			

Please feel free to talk to the TAs if you encounter problem using/compiling `libavl`.

Submission File

Please submit your written part of the homework to Gradescope before the deadline. Also, you need to upload your coding part as a single ZIP compressed file to CEIBA before the deadline. The zip file should contain no directories and only the following items:

- all the source code and your own `Makefile` such that `make hw6_2_1` would generate a program named `hw6_2_1`, `make hw6_2_2` would generate a program named `hw6_2_2`, and `make hw6_2_3` would generate a program named `hw6_2_3`.
- your copy of `libavl`; that is, the TAs will not copy any other code to your directory this time.
- an optional `README`, anything you want the TAs to read before grading your code

Please make sure that your code can be compiled and run with the Makefile on CSIE R217 linux machines. Otherwise your program "fails" its most basic test and can result in ZERO!