Homework #6

TAs' email: dsata AT csie DOT ntu DOT edu DOT tw

RELEASE DATE: 05/24/2013
DUE DATE: 06/06/2013, noon

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.

Note that this is the last homework of this semester and is therefore your last chance of using the MEDALs, if you still have any!

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-10.2 of the textbook.

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

(4) (20%) Do Exercise R-10.15 of the textbook.

(5) (20%) Do Exercise C-10.20 of the textbook.

(6) (Bonus 20%) Do Exercise C-10.18 of the textbook.

6.2 Balanced Binary Search Trees

(1) (50%) `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 the following 30 strings (that can be compared lexicographically)

> `C`, `Java`, `C++`, `Objective-C`, `C#`, `PHP`, `Visual Basic`, `Python`, `Perl`, `Ruby`, `JavaScript`, `Lisp`, `Pascal`, `Haskell`, `Scala`, `Fortran`, `Prolog`, `Assembly`, `Verilog`, `Erlang`, `MATLAB`, `Bash`, `SmallTalk`, `Caml`, `Scheme`, `Go`, `Ada`, `Cobol`, `Awk`, `Tcl/Tk`, `Delphi`, `Limbo`

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`). Output the resulting trees with a format similar to the output above.

(2) (50%) Write a program `hw6_2_2` that randomly generates 4096 double numbers between $[0, 1]$ before they are inserted into the three different trees above. Then, remove the first 2048 keys you generated from the trees. 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 insertion	minimum height after insertion	average height after 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			

You are strongly encouraged to to talk to the TAs if you encounter problem using/compiling `libavl`.

Submission File (Program) and Written Copy

Please upload your program as a single ZIP compressed file to CEIBA before the deadline. The zip file should be like `b86506054.zip`, where the file name should be changed to your own school ID. The ZIP file should contain the following items:

- `hw6_2_1.c` or `hw6_2_1.cpp`
- `hw6_2_2.c` or `hw6_2_2.cpp`

The TAs will use the `Makefile` provided on the course website to test your code. Please make sure that your code can be compiled with the `Makefile` on CSIE R217 linux machines.
For all the problems that require illustrations, please submit a written (or printed) copy in class or to CSIE R217 before the deadline.

MEDAL USAGE: If you want to use the gold medals for this homework, please write down the number of medals that you want on the first page of your printed copy (something like "use 2 medals"). Use your medals wisely—usage cannot be retracted.