Assignment 5: Balanced trees and set operations

Date: March 9th 2020
Deadline: March 20th 23:59 2020

Objectives

You must implement a tree and set API and a set manipulation program.

Requirements

Your set manipulation program must be called `sets` and its basic operation is as follows:

- Create a new empty set.
- Read lines from standard input that contain a set operation, and depending on the operation, a number of operands. Perform the set operation.

There are four different set operations:

- `+ number` insert the number into the set.
- `- number` remove the number from the set.
- `? number` check if the number is in the set. If the set contains the number it prints `found: num` and if it does not it will print `not found: num`.
- p print the current set in sorted order.

Your implementation must achieve this by translating the set operations into tree operations and performing these tree operations. Only the find (`? num`) and the print set operation (p) print to stdout.

The `set_init` function takes a turbo parameter. If turbo is set to 0 the set can be implemented as a regular BST. If turbo is set to 1 a balanced tree implementation is used for high performance. You are not forced to implement a slow unbalanced tree implementation, but for grading it can be better to have a slow implementation to fall back on if the fast implementation is not working correctly.

You must submit your work as a tarball. Use the command `make tarball` to create the tarball.

Details on the input and output formats

The following shell session shows an input file that inserts three numbers and prints the set followed by the output of the `sets` program:

```
$ cat tests/01_simple_insert.txt
+ 5
+ 1
+ 10
p
$ ./sets < tests/01_simple_insert.txt
1
5
10
```

The following shell session shows an input file that uses all four set operators:

```
$ cat tests/02_simple_lookup_remove.txt
+ 1
+ 2
```

```
+ 3
p
? 1
? 2
? 3
- 2
p
? 1
? 2
? 3
$ ./sets  < tests/02_simple_lookup_remove.txt
1
2
3
found: 1
found: 2
found: 3
1
3
found: 1
not found: 2
found: 3
```

Getting started

1. Unpack the provided source code archive; then run `make`.
2. Try out the generated `sets` program and familiarize yourself with its interface.
3. Read the files `set.h` and `tree.h` and understand the interface.
4. Implement the binary search tree data structure in `tree.c`.
5. Implement the abstract set data type `sets.c`. These will be mostly calls to the tree data structure.
6. Now that you have a working binary search tree implementation see if you can improve the performance of your implementation.

Testing

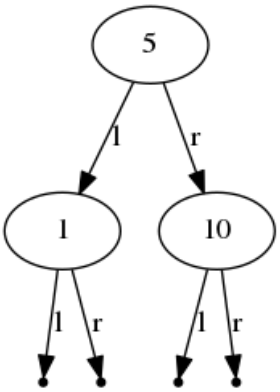
The `tests` directory contains example input and reference output. Use `make check` to test your `sets` program with these input files.

You can time your program with the `time` command:

```
$ time ./sets < tests/10_15000_sorted_inserts.txt  > /dev/null

real    0m1.651s
user    0m1.644s
sys 0m0.000s
```

The `real` time is the actual elapsed time. `user` and `sys` time is the time that the program actually spent executing user code and system code.



The graph `01_simple_insert.txt` *in dot format.*

The `tree_dot()` function provided in `tree.c` writes a tree data structure into graphviz dot format to the file `tree.dot`. To convert the dot format to pdf type `make tree.pdf`. The function `tree_dot()` traverses the tree and expects that the tree node struct contains a `lhs`, `rhs` and a `data` member. If you decide to change the struct you also need to modify `tree_dot()`.

Grading

Your grade starts from 0, and the following tests determine your grade:

- +1pt if your source code builds without errors and you have modified `tree.c` in any way.
- +2pt if your tree API processes insertions correctly.
- +3pt if your tree API processes deletes correctly.
- +2pt for implementing the rest of the tree api functions.
- +2pt if your tree API balances itself when the turbo flag is set to 1.
- -0,5pt if your programs misbehave on zero-sized inputs.
- -0,5pt if your programs misbehave when the last line does not terminate with a newline character.
- -1pt if `valgrind` or ASAN report errors while running your program.
- -1pt if `gcc -Wall` reports warnings when compiling your code.
- -2pt If your implementation has style violations or has a too high complexity.