Universität Heidelberg
Institut für Technische Informatik

Prof. Dr. Robert Strzodka
Steffen Lammel

Advanced Parallel Computing
Term 2015 (Summer)

Exercise 7

- **Return electronically (MOODLE) until Tuesday, 09.06.2015 14:00**
- **Include name on the top sheet. Stitch several sheets together.**
- **A maximum of two students is allowed to work jointly on the exercises.**

7.1 Reading

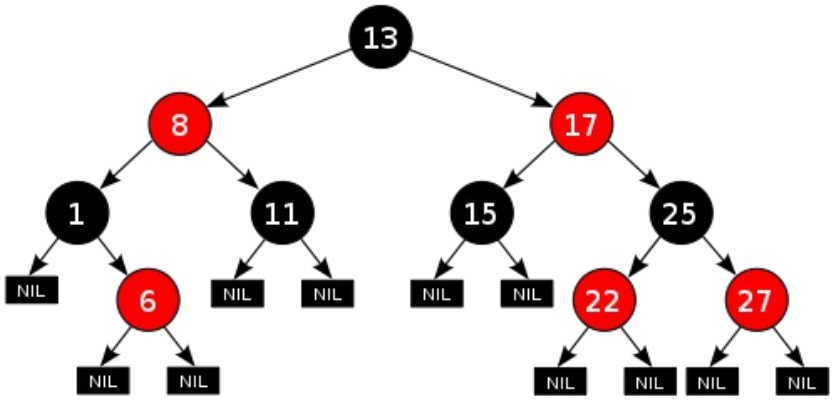
Read the following paper and provide a review as explained in the first lecture (see slides):

- Paul E. McKenney, Maged M. Michael, and Jonathan Walpole. 2007. Why the grass may not be greener on the other side: a comparison of locking vs. transactional memory. In *Proceedings of the 4th workshop on Programming languages and operating systems (PLOS '07)*.
- Aleksandar Dragojevic; Rachid Guerraoui, and Michal Kapalka. 2009. Stretching transactional memory. In *Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation (PLDI '09)*.

(25 points)

7.2 Red-Black Tree – Implementation of sequential version

For this exercise, we will use a complex data structure to analyze the impact of different locking schemes and methods. The data structure to be implemented is a **Red-Black Tree (RBT)**, which is a form of self-balancing binary search tree.



The following reference provides a very nice explanation: http://en.wikipedia.org/wiki/Red-black_tree (I). In particular notice the five properties of a RBT:

1. A node is either red or black.
2. The root is black.
3. All leaves (NIL) are black.
4. Both children of every red node are black.
5. Every simple path from a given node to any of its descendant leaves contains the same number of black nodes.

A nice explanation of the required functionality can be found here:
http://www.slideshare.net/piotrszymanski/red-black-trees (II)

- Implement red-black tree based on the description provided in the link (II) above. It should support sequential versions of the following two operations: `add, search`. **Use the top-down versions, not the bottom-up ones!**
- Ensure that the RBTs your code generates fulfill the 5 requirements stated in the link (I) above. Opportunities to check the consistency would be after initialization and after the experiment (maybe after each operation for verification purposes). Obviously, do not include these checks in your time measurements.
- For insert, ensure that all elements are unique.

(50 points)

7.3 Red-Black Tree – Sequential Performance

- Perform a performance evaluation of the sequential RBT function.
- Initialize the RBT with 10M elements before starting the measurement (i.e., perform 10M insert operations).
- Generate a stream of update actions, with each action including the command (add, search) and an associated data (index to be inserted or searched). Generate this update stream using random values. Ensure that the ratio between insert and search operations is 10/90 (make this ratio a parameter of your program!).
- Measure the performance in terms of operations per second (OPS) for an update stream with varying lengths like shown below. Report also graphically and interpret your results! Use `moore` for this performance assessment.

	Operations per second
Operations	**Sequential**
100k	
500k	
1M	
5M	
10M	
25M	
50M	

(25 points)

Prof. Dr. Robert Strzodka
Steffen Lammel

Note: RBT Visualization

The working material includes a python script (rbtv.py) to visualize your RBTs and is supposed to help you with debugging your code.

In order to use the visualization you have to dump your tree in a specific (but simple) format. The source code of the script contains a syntax description, example C code to produce output in the desired format and usage information.

Obviously, subsequent exercises will look at parallelizing this code, so ensure a nice coding and plenty of meaningful comments! Make use of header files, and structure your code in functions!

Total: 100 points