# CS 521/621
# Homework 1
# Dynamic Analysis

Due: **Thursday, Feb 16, 2017, 9:00 AM EST** via Moodle. You may work with others on this assignment but each student must submit his or her own write up, clearly specifying the collaborators. The write ups should be individual, not created jointly, and written in the student's own words. Late assignments will not be accepted without **prior** permission.

## Overview

The goal of this assignment is to learn about dynamic program invariant detection. Specifically, we use a tool called Daikon to analyze the invariants in a 2-3-4 tree implementation.

The assignment consists of:

1. Installing and using Daikon to dynamically detect program invariants of a 2-3-4 tree implementation.
2. Answering problem questions about Daikon and the detected invariants.

## Resources

- The Daikon Invariant Detector User Manual: http://plse.cs.washington.edu/daikon/download/doc/daikon.html
  (Sections 3.1 and 5.2-5.5 can be helpful for understanding the output.)
- Download the following:
  - The Daikon binary: daikon.jar
    http://plse.cs.washington.edu/daikon/download/daikon.jar
  - The source code for 2-3-4 tree, including tests:
    * http://cs.umass.edu/~brun/class/2017Spring/CS521.621/hw1/TwoThreeFourIntSet.java
    * http://cs.umass.edu/~brun/class/2017Spring/CS521.621/hw1/IntNode.java
    * http://cs.umass.edu/~brun/class/2017Spring/CS521.621/hw1/TwoThreeFourTester.java
  - Input test data for the 2-3-4 set test driver:
    * http://cs.umass.edu/~brun/class/2017Spring/CS521.621/hw1/input1.txt
    * http://cs.umass.edu/~brun/class/2017Spring/CS521.621/hw1/input2.txt

## Background on 2-3-4 trees

The test program implements a multiset data structure for `int` values, backed by a 2-3-4 tree. If you are familiar with B-Trees, a 2-3-4 is just a special case of a B-Tree with order 4. More information is available at http://en.wikipedia.org/wiki/2-3-4_tree

A 2-3-4 tree is a self-balancing, n-way search tree. Like a binary tree, it consists of nodes with children. Unlike a binary tree, each node may have more than just a left child and a right child; each node of a 2-3-4 tree may have 2, 3, or 4 children. Instead of a single element at each non-root node, a node may have anywhere from 1 to 3 elements contained within it. Elements are stored in a node in sorted order.

A 2-node is just like a binary tree node with a single element $x$, with a left child $L$ and right child $R$. Each child node is the root of a subtree. All elements in $L$ and its descendants are less than or equal to $x$, and all elements in $R$ and its descendants are greater than or equal to $x$. A 3-node contains 2 elements $x_1$ and $x_2$, with children left $L$, middle $M$, and right $R$. Elements in $L$'s subtree are all less than or equal to $x_1$, elements in $M$'s subtree are greater than or equal to $x_1$ and less than or equal to $x_2$, and elements in $R$'s subtree are all greater than or equal to $x_2$. A 4-node contains 3 elements, with a left, left-middle, right-middle, and right child, with similar bounds on the ranges of elements in the child subtrees.

A 2-3-4 tree avoids imbalanced subtrees by moving elements and children between nodes in the tree whenever an element is added or removed. The algorithms for adding and removing elements can be quite involved because of the large number of special cases. For this assignment, you should not need to know how these rearranging algorithms work (although knowing how they work may be helpful); you should be able to answer the questions by thinking of the tree simply as a balanced n-way tree.

## Setup

daikon.jar contains all of Daikon's executables. It is known to work under Oracle (Sun) JRE 1.7 and above. **If you are using the outdated Java 1.6 (a.k.a. Java 6) or earlier, please upgrade).** Daikon supports multiple programming languages, but we will only be using the Java front end to Daikon, called Chicory. Run Daikon using these three steps:

- Compile the .java files with the following command line (with -g to include debugging symbols):

  ```
  javac -g -source 6 -target 6 IntNode.java TwoThreeFourIntSet.java TwoThreeFourTester.java
  ```

- Have Chicory run the program (TwoThreeFourTester), and Chicory will record the values of variables it observes into `.dtrace.gz` file. Chicory's arguments are the Java class name and the command line arguments to be passed to that Java program. (Notes: The `--` is important in the line below to tell Chicory to pass the `--file` flag to TwoThreeFourTester instead of trying to parse it as a Daikon flag. `input#.txt` will be a file specified by one of the problem statements below.)

  ```
  java -cp .:daikon.jar daikon.Chicory TwoThreeFourTester -- --file=input#.txt
  ```

- Have Daikon read all the observed values and analyze the data, looking for statistically significant invariants. Daikon stores all these likely invariants into a binary format in a compressed `.inv.gz` file. This command also stores them in a human-readable txt file that you will analyze and submit. For TwoThreeFourTester, this step may take over a minute. (Note: `output#.txt` will be a file specified by one of the problem statements below.)

  ```
  java -cp .:daikon.jar daikon.Daikon TwoThreeFourTester.dtrace.gz > output#.txt
  ```

## Problem questions

The test driver takes a script file as input containing instructions for adding (ADD), removing (REMOVE), and querying the tree. When querying the tree, a script may check if a single element is present (CONTAINS), find out how many elements are present (SIZE), or check how deep the tree is based on distances from the root to childless leaves (HEIGHT). The contents of the tree may also be printed (PRINT), with parentheses depicting the levels of the tree.

1. Run the test driver with the `input1.txt` as input. Save the output of PrintInvariants as `output1.txt`, and examine the likely global invariants found for `IntNode:::OBJECT`, which hold at every entrance and exit for every public method of the `IntNode` class. For each of the *first 5 invariants*, answer the following questions:

   (a) In plain English, what does the likely invariant mean?

   (b) Is it really an invariant of the program, regardless of input? If not, explain why Daikon reports the false invariant. (Note: You do not need to consider inputs that try to cause crashes, e.g., ignore "out of memory" errors and integer overflow.)

2. Run the test driver with the file `input2.txt` as input, and save the output of PrintInvariants as `output2.txt`. The invariants found for the different inputs will be slightly different. Each of the likely invariants below appears in output1.txt but was either changed or removed in output2.txt. For each (there are a total of four), explain why Daikon found a different invariant or did not find the invariant in output2.txt, and whether the invariant is true.

   (a) `TwoThreeFourIntSet.containsRecursive(IntNode, int):::ENTER`
   - `node != null`

   (b) `TwoThreeFourIntSet.height():::EXIT`
   - `return one of {1,2,3}`
   - `this.root.elements[] elements < this.size`

   (c) `TwoThreeFourIntSet.remove(int):::EXIT`
   - `return == true`

3. For `input2.txt`, Daikon found a likely invariant for `TwoThreeFourIntSet.height()` that relates the height of the tree to the size in number of elements in the tree. The likely invariant is actually incorrect but even if it were, since a 2-3-4 tree is self balancing, a correct tighter bound exists relating size and height. Daikon does not support looking for such an invariant. Describe a generic invariant that can be added to Daikon that would allow it to report this height invariant. (Hint: Think about bounds that can be found for balanced binary trees such as a red-black tree.)

## Deliverables

Write the answers to the problem questions in either `writeup.txt` or `writeup.pdf`. Include your name in this file.

Compress the following three files into a single `.zip` archive (call it `daikon_lastname.zip`) and upload it via Moodle.

- `writeup.txt` or `writeup.pdf`
- `output1.txt`
- `output2.txt`

You may use any resource you wish in this assignment but you must list your collaborators and cite all your sources. Failure to do so will result in a grade of 0.

## Helpful hints

- The questions can be solved by observing the input, but if you want to dig deeper into the individual invariants, you can follow Daikon as it tries to invalidate individual invariants. Using the invariant classes from Section 5.5 of the manual, use a variant of the following command line:

```
java -cp .:daikon.jar daikon.Daikon TwoThreeFourTester.dtrace.gz
--track "NonZero<return>@IntNode.size(int):::ENTER"
```

- The recorded observations are human readable once decompressed. If you are stuck, try looking at the trace file with the following command line:

```
gunzip -c TwoThreeFourTester.dtrace.gz | less
```