OMSCS 6340 - Fall 2016
Assignment 3 (100 points)
Due on: 8:00 EDT September 26, 2016

Objective: The goal of this assignment is to become familiar with systematic automated test case generation techniques, in particular, Korat that allows exhaustively generates all relevant test cases up to a particular size bound.

Resources:
1. Korat tutorial: http://korat.sourceforge.net/tutorial.html
2. Korat manual: http://korat.sourceforge.net/manual.html
3. Korat paper: http://mir.cs.illinois.edu/~marinov/publications/BoyapatiETAL02Korat.pdf
4. Korat javadocs: http://korat.sourceforge.net/docs/korat_api/index.html

Setup: To start this assignment, you need to have the following software installed:
1. JDK 6 or later versions, which are available at http://www.oracle.com/technetwork/java/javase/downloads/index.html
2. Korat - provided as korat.zip with the assignment.

To set up Korat,
1. Unzip the provided korat.zip file.
2. Include all the JAR files in the unzipped folder in your CLASSPATH environment variable to run Korat.
3. Try the following example to make sure things are working: `java -Xverify:none korat.Korat --class korat.examples.binarytree.BinaryTree --args 3,3,3`
4. Korat should produce an output similar to the following:

```
Start of Korat Execution for
korat.examples.binarytree.BinaryTree (repOK, [3, 3, 3])
Total explored:63 New found:5
End of Korat Execution Overall time: 0.187 s.
```

The files for this assignments should be compiled against the latest Korat distribution, provided with this assignment.

Problem 1. [50 points]
In this problem you will generate test cases for a K-ary heap using Korat. A K-ary heap is defined as follows: A K-ary heap is a heap data structure created using a K-ary tree where each node in

the tree has at most K children. A K-ary heap can be seen as a K-ary tree with two additional constraints:

- The shape property: All levels of the tree, except possibly the last one (deepest) are fully filled, and, if the last level of the tree is not filled, the nodes of that level are filled from left to right.
- The heap property: The data item in each node is greater than or equal to the data items in each of its children according to some comparison predicate which is fixed for the entire data structure.

An array implementation of the K-ary heap is provided in the file KHeap.java. Write the repOK method for the K-ary heap based on the above description. Use the finitization method to set the range of values in the array and the value of K. An array implementation for a binary heap is provided on the Korat website (http://korat.sourceforge.net/tutorial.html) which should help you in testing the K-ary heap. You should write your finitization method to reject heaps with null elements in the tree.
Generate test cases for a K-ary heap using Korat. To run Korat, first compile KHeap.java using the command javac KHeap.java. Next, invoke Korat as follows: `java -Xverify:none korat.Korat --class KHeap --args k,n,n,n` where k is the value of K and n is the number of nodes. Generate test cases for K-ary heaps where K is 3 and 4. Choose values from 0-5 for the number of nodes and generate all the canonical heaps for each value using Korat.

Problem 2. [50 points]
In this problem you will generate test cases for a directed acyclic graph using Korat. A directed acyclic graph is a graph with directed edges and no cycles. Note that between any pair of nodes there is at most one edge in each direction. An array implementation of a directed graph is provided in the file AcyclicGraph.java.
Write the repOK and finitization methods for a directed acyclic graph based on the above description.
Generate test cases for directed acyclic graphs using Korat. To run Korat, first compile AcyclicGraph.java using the command `javac AcyclicGraph.java`. Next, invoke Korat as follows: `java -Xverify:none korat.Korat --class AcyclicGraph --args n` where n is the number of nodes. Choose values from 0-5 for the number of nodes and generate all the canonical graphs for each value using Korat.

Items to Submit:
Completed KHeap.java and AcyclicGraph.java files. You may test the correctness of your submission by comparing the number of generated tests to the provided CountTestsSoln.txt file.

Plagiarism: Note that we will be running a fairly sophisticated tool to check if the submitted code has been copied. This tool finds deep equivalences between different versions of the code, and if you have copied, your code is likely to get flagged. Any copying will result in the strictest possible consequences.