**Foundations of Software Testing, 2E**

**Test Order Generation from Object Relation Diagrams**

Java Programs for the Algorithms in Chapter 11: Integration Testing

Updated: July 29, 2013

Table of Contents

1. How to generate a test order? 2

2. Encoding an ORD 3

3. Folders and files 4

4. Debugging switches 6

5. Object Relation Diagrams (ORDs) 7

6. Generating random ORDs 7

7. Bugs 8

# How to generate a test order?

(a) The following programs need to be edited, compiled and executed:

TaiDanielsAlgorithm.java To apply the TD algorithm

LaTraonAlgorithm.java To apply the TJJM algorithm

BriandAlgorithm.java To apply the BLW algorithm

Locations of the above three programs are given in the following section. Each of the above programs contains a main() method. Several sample ORDs are stored in the TestGraphs folder. Each ORD file name ends with .txt.

(b) To select the ORD on which one of the three algorithms is to be app to generate a test seqeunce, uncomment the line that creates the algorithm object (b for BLW algorithm, t for TD algorithm, and l for TJJM algorithm). For example, uncomment the following line

// LeTraonAlgorithm l=new LeTraonAlgorithm("../TestGraphs/BriandExample2003.txt");

to apply the TJJM algorithm on ORD BriandExample2003.txt.

(c) Once an ORD has been selected as described above, compile and run the program in your favorite IDE (e.g., Eclipse, DrJava). As an example, the following output is generated when the BriandAlgorithm.java file is compiled and executed against ORD TaiDanielsGraph.txt.

> run BriandAlgorithm

Input graph from file: ../TestGraphs/TaiDanielsGraph.txt

--Graph successfully input.

-------------Begin Test Order Determination using BLW algorithm. V1.0. July 29, 2013.

Find strongly connected components.

------ Start: Recursive call to find additional Sccs.

------ End: Recursive call.

Test order:

Stubs needed: 4 Specific stubs: 4

End of BLW algorithm.

# Encoding an ORD

Each ORD needs to be encoded and saved in a file with extension .txt and is a plain text file. The test order generation algorithms read the ORD from a file and create a graph to be processed. The format of a .txt file containing an ORD is as follows.

Data for each node or edge occupies one line. Lines are separated by CRLF. Each node contains the following information: node label There is exactly one line for each node label.

Node-label

The line for each node is followed by as many lines as there are edges going out of that node.

edge-label destination-node-label

An ampersand (&) indicates the end of edges for a node. It is placed by itself on a line. It indicates the end of all edges for a node. A node label by itself on a line and followed on the next line by an ampersand (&) indicates that this node has no edges.

The label of a node must be prefixed by an "n". For example, label nB indicates that B is a node. Similarly an edge label must be prefixed by an "e". For example, label eF indicates an edge labeled F. A node can be labeled as Abstract by using the tag tA following the node name.

**Example input: ORD coding for Figure 11.11 (a) on page 640 of the textbook.**

nE

eA nF

eA nA

&

nF

eC nE

eA nD

&

nA

eA nC

&

nD

eI nA

eI nH

&

nC

eA nA

eA nH

eA nE

&

nB

eA nC

eA nD

eC nH

&

nG

eA nF

eI nB

&

nH

eA nB

eC nC

&

The above data is for a graph with eight nodes labeled A, B, C, D, E, F, G, and H. Node E has two outgoing edges and node C has three. An inheritance edge is indicated as eI, association edge as eA, and aggregation edge as eC.

# Folders and files

|  |  |  |
| --- | --- | --- |
| **Folder** | **Contents** | **Comments** |
| TestOrderAlgorithms | Folders containing programs related to integration testing |  |
| BriandAlgorithm | BriandAlgorithm.java | Execution of the BLW method begins here. |
| CreateGraph | Reads ORD data from a file and generates a graph object. Format of the ORD file is listed in this file. An example illustrates the format. |
| Edge.java | Represents an Edge object in class Node. |
| EdgeWeight.java | Holds the weight of an edge in the ORD. |
| Graph.java | Denotes a graph (ORD) object. |
| Node.java | Denotes a node in a Graph object. |
| ProgramErrors.java | Holds errors discovered while running the class test sequence generation algorithm. |
| Retest.java | Holds classes to be rested. |
| Scc.java | Holds strongly connected components in a graph found using Tarjan’s algorithm. |
| SccTarjan.java | Encoding of Tarjan’s algorithm to find strongly connected components. |
| Stub.java | Holds a stub object. |
| Utilities.java | Contains several utility methods for printing, debugging, as well as constants used throughout the sequence determination process. |
| LeTraonAlgorithm | LaTraonAlgorithm.java | Execution of the TJJM algorithm begins here.  All other files listed under the BriandAlgorithm folder are also found in this folder. |
|  |  |  |
| Tai-DanielsAlgorithm | TaiDanielsAlgorithm.java | Execution of the TD method begins here. All other files listed under the BriandAlgorithm folder are also found in this folder. |
| GenerateORD | Folders containing programs to generate random ORDs. | Random ORDs are useful n testing and analyzing the performance of class test sequence generation algorithms |
| TestGraphs | Contains a number of ObjectRelationDiagrams. Each ORD file uses .txt as its extension. |  |
| SecondEditionExperiments | This folder is located inside TestGraphs. It contains the ORDs used in experiments described in Section 11.6.4 of the textbook. |  |
| FullyConnectedORDs | This folder is located inside TestGraphs. It contains few fully connected ORDs. |  |

# Debugging switches

Following is a list of switches, their location, and purpose. These could be turned ON/OFF to enable/disable printing of various intermediate data generated during the execution of any of the three class test sequence generation algorithms.

|  |  |  |
| --- | --- | --- |
| **Switch** | **Location** | **Set this to true to …..** |
| cycleBreakingDebug | TDAlgorithm/ Utilities.java | Debug cycle breaking. |
| cycleCheckingDebug | TDAlgorithm/ Utilities.java | Debug cycle checking. |
| dependenceCreationDebug | TDAlgorithm/ Utilities.java | Debug class dependence determination. |
| edgeDeletionDebug | TDAlgorithm/ Utilities.java | Debug the edge deletion algorithm. |
| graphInputDebug | TDAlgorithm/ Utilities.java | Debug graph input method. |
| levelAssignmentDebug | TDAlgorithm/ Utilities.java | Debug major/minor levels in the TD algorithm. |
| majorLevelProcessingDebug | TDAlgorithm/ Utilities.java | Debug major-level processing in TD algorithm. |
| minorLevelProcessingDebug | TDAlgorithm/ Utilities.java | Debug major-level processing in TD algorithm. |
| newStubCreationDebug | TDAlgorithm/ Utilities.java | Debug new stub creation in TD algorithm. |
| retestCreationDebug | TDAlgorithm/ Utilities.java | Debug retest class portion of the TD algorithm. |
| sccDebug | BriandAlgorithm/  Utilities.java | Debug SCC creation in BLW algorithm. |
| stubCreationDebug | TDAlgorithm/ Utilities.java | Debug stub creation in the TD algorithm |
| tarjanDebug | TDAlgorithm/ Utilities.java | Debug the code for Tarjan’s algorithm. |
| testSequenceProcessingDebug | TDAlgorithm/ Utilities.java | Debug test sequence generation after the ORD has been processed. |
| topologicalSortDebug | BriandAlgorithm/  Utilities.java  LaTraonAlgorithm/  Utilities.java | Debug the topological sort in the findTestOrder() method also found in Utilities.java. |
| weightDebug | BriandAlgorithm/  Utilities.java | Debug edge weight check in BLW algorithm. |
| frondDebug | LaTraonAlgorithm/  Utilities.java | Debug frond edge check in TJJM method. |
| testOrderDebug | LaTraonAlgorithm/  Utilities.java | Debug test order generation in TJJM method. |
|  |  |  |

# Object Relation Diagrams (ORDs)

The TestGraphs folder contains number of files with ORDs. Each file ends with .txt as its extension. Inside TestGraphs a folder named FulyConnectedORDs contains several fully connected ORDs with 2, 3, 4, 5, and 6 nodes. Another folder inside TestGraphs is named SecondEditionExperiments and contains a number of randomly generated ORDs. Each such ORD is labeled as NxxEyy, where xx is the number of nodes and yy the number of edges. For example, N100E3044 is an ORD with 100 nodes and 3044 edges.

# Generating random ORDs

Folder GenerateORD contains Java programs to generate ORDs. The program asks for the number of nodes in the graph, graph density (0 to 1), proportion of edges that are inheritance edges (0 to 1), and the proportion of edges that are aggregation edges (0 to 0.8). It then generates an ORD satisfying the requested ORD configuration as best as possible. The output of the program is a graph that is saved in folder SecondEditionExperiments. The program outputs additional information about the graph as, for example, shown below.  
  
Random graph generation completed.

ORD checks successful.

This graph has cycle(s)

Nodes: 15

Density requested: 0.10 Actual density: 0.13

Max possible edges: 210

Edges requested: 21

Edges in ORD: 28

Edges discarded to avoid cycle: 4

Edges discarded due to parent-child relationship: 3

Inheritance edges: 3 Proportion: 0.1071

Aggregation edges: 2 Proportion: 0.0714

Association edges: 23 Proportion: 0.8214

Total time: 224.2400 seconds.

Saving ORD in N15E28.txt

ORD saved in ../TestGraphs/SecondEditionExperiments/N15E28.txt

Generate another ORD (Enter Y for Yes and N for No):

# Bugs

The implementation of TJJM algorithm has a bug that leads to the following error when applied to the ORD LabichExample2000.txt.

Program error (processSingleton/LeTraonAlgorithm). All SCCs singleton but no root.

Report bugs to [apm@cs.purdue.edu](mailto:apm@cs.purdue.edu). However, it would be great if you can fix the bug and send me the modified set of files.

<End of document>