University of Zurich^UZH	**Test Plan for AutoTask Library**	Author: Ali, Chen, Liu Date: 08-11-2018 Page of Pages: 1/15

Revision History

Date	Version	Description	Author(s)
10/11/2018	Final Version	Test Plan Completed	Ali, Chen, Liu
23/11/2018	Revised	Add plans for routine unit tests	Ali, Chen, Liu
09/12/2018	Revised	Revised for Final API Implementation	Ali, Chen, Liu

1. Public API Overview

1.1. Public classes to be tested

AUTO_TASK

DIRECTED_GRAPH

CONVERTER

DISPLAYER

ALGORITHM

1.2. Public routines to be tested

AUTO_TASK

- add_element (element_name: STRING)
- add_constraint (element_name_1, element_name_2: STRING)
- add_makefile (makefile: PLAIN_TEXT_FILE)
- delete_element (element_name: STRING)
- delete_constraint (element_name_1, element_name_2: STRING)
- topo_sort: LINKED_LIST [STRING]
- display: (output_file_name: STRING)

DIRECTED_GRAPH

- set_largest_node (largest_node: INTEGER)
- add_edge (predecessor, successor: INTEGER)
- delete_node (node: INTEGER)
- delete_edge (predecessor, successor: INTEGER)

CONVERTER

- name2id (name: STRING): INTEGER
- id2name (id: INTEGER): STRING
- insert_name (name: STRING)
- delete_name (name: STRING)

DISPLAYER

- to_graphviz_format (graph: DIRECTED_GRAPH, converter: CONVERTER, output_file_name: STRING)

ALGORITHM

- cycle_detection (graph: DIRECTED_GRAPH): BOOLEAN
- topo_sort (graph: DIRECTED_GRAPH): LINKED_LIST [INTEGER]

2. Test Suite Description

2.1. Test Cases

Test ID	2.1.1.1
Requirements under test	3.2.01, 3.2.02, 3.2.04, 3.2.05, 3.2.09
Routines under test	Class: AUTO_TASK add_element (element_name: STRING) add_constraint (element_name_1, element_name_2: STRING) delete_element (element_name: STRING) delete_constraint (element_name_1, element_name_2: STRING) topo_sort
Description	Integration Test: The library shall be able to add or remove elements and constraints with strings as an identifiers of elements.
Set-up	An instance of AUTO_TASK should be created
Tear-Down	The instance of AUTO_TASK should be deleted automatically by the Eiffel garbage collector after termination of testing.

Test data	add_element(“event1”) add_element(“event2”) add_element(“event3”)
	add_constraint(“event1”, “event2”)
	add_constraint(“event1”, “event3”)
	add_constraint(“event2”, “event3”)
	delete_constraint(“event1”, “event3”) delete_element(“event3”)

Oracle	Call AUTO_TASK.topo_sort, and then a LINKED_LIST would be returned containing “event1 -> event2” only.

Test ID	2.1.1.2
Requirements under test	3.2.11, 3.2.12, 3.2.13, 3.2.14
Routines under test	Class: AUTO_TASK add_makefile (makefile: PLAIN_TEXT_FILE) display()
Description	The library shall be take an makefile as input, and generate a Graphviz text file accordingly.
Set-up	An instance of AUTO_TASK should be created. A makefile (`input_makefile_test_auto_task.txt`) will be generated as input.
Tear-Down	The input makefile (`input_makefile_test_auto_task.txt`) and output Graphviz text file (`output_graph_test_auto_task.txt`) will be deleted from the file system.
Test data	`input_makefile_test_auto_task.txt` event1: event2 event3 event4:

Oracle	The generated Graphviz text file will be checked line by line.
	Expected Graphviz text file (`output_graph_test_auto_task.txt`):
	diagraph output_graph {
	    event1 -> { event2 event3 }
	    event2 -> { }
	    event3 -> { }
	    event4 -> { }
	}

Test ID	2.1.2.1
Requirements under test	3.2.01, 3.2.02, 3.2.04, 3.2.05
Routines under test	Class: CONVERTER insert_name (name: STRING)
Description	Unit Test: insert an element name into converter, and then check whether the name is inserted correctly
Set-up	An instance of CONVERTER should be created
Test data	insert_name(“event1”)
Oracle	Converter.id2name_array [1] = “” Converter.name2id_table.item(“event1”) = 1

Test ID	2.1.2.2
Requirements under test	3.2.01, 3.2.02, 3.2.04, 3.2.05
Routines under test	Class: CONVERTER delete_name (name: STRING)
Description	Unit Test: delete an element name from converter, and then check whether the name is deleted correctly
Set-up	An instance of CONVERTER should be created
Test data	insert_name(“event1”) delete_name(“event1”)
Oracle	name2id(“event1”) = 0 (which means not exists or deleted) id2name(3) = “” (which means not exists or deleted)

Test ID	2.1.2.3
Requirements under test	3.2.01, 3.2.02, 3.2.04, 3.2.05
Routines under test	Class: CONVERTER id2name (id: INTEGER) name2id (name: STRING)
Description	Every element is uniquely identified by its name (STRING) or by its id (INTEGER). The converter should be able to convert an element’s name to its corresponding id, and vice versa.
Set-up	An instance of CONVERTER should be created

Test data	insert_name(“event1”)
	insert_name(“event2”)
	id := name2id(“event1”)
	name := id2name(2)

Oracle	id = 1
	name = “event2”

Test ID	2.1.3.1
Requirements under test	3.2.01, 3.2.02, 3.2.04, 3.2.05, 3.2.09
Routines under test	Class: DIRECTED_GRAPH set_largest_node (largest_node: INTEGER) delete_node (node: INTEGER)
Description	Every node is uniquely identified by an id (INTEGER) in the directed graph. The DIRECTED_GRAPH class should be manipulated with node addition/deletion.
Set-up	An instance of DIRECTED_GRAPH should be created.

Test data	set_largest_node(5)
	delete_node(3)

Oracle	graph.deleted[1] = false
	graph.deleted[2] = false
	graph.deleted[3] = true
	graph.deleted[4] = false
	graph.deleted[5] = false

Test ID	2.1.3.2
Requirements under test	3.2.01, 3.2.02, 3.2.04, 3.2.05, 3.2.09
Routines under test	Class: DIRECTED_GRAPH add_edge (predecessor, successor: INTEGER) delete_edge (predecessor, successor: INTEGER)
Description	Every node is uniquely identified by an id (INTEGER) in the directed graph. The DIRECTED_GRAPH class should be manipulated with edge addition/deletion.
Set-up	An instance of DIRECTED_GRAPH should be created.
Tear-Down	The instance of DIRECTED_GRAPH should be deleted automatically by the Eiffel garbage collector after termination of testing.

Test data	set_largest_node(3)
	add_edge(1, 2)
	add_edge(1, 3)
	delete_edge(1, 3)

Oracle	graph.check_edge_existence (1, 2) = true graph.check_edge_existence (1, 3) = false The adjacency list (successors ARRAY) should be set properly.

Test ID	2.1.4.1
Requirements under test	3.2.06, 3.2.07, 3.2.08, 3.2.09, 3.2.10
Routines under test	Class: ALGORITHM cycle_detection(graph:DIRECTED_GRAPH): LINKED_LIST[INTEGER]
Description	Every node is uniquely identified by an id (INTEGER) in the directed graph. The ALGORITHM must be able to detect the cycle between entered elements/constraints.
Set-up	An instance of ALGORITHM should be created
Tear-Down	The instance of ALGORITHM should be deleted automatically by the Eiffel garbage collector after termination of testing.
Test data	An instance `graph_with_cycle:` DIRECTED_GRAPH should be created, with four nodes, and four edges as following: 1->2, 2->3, 3->2, 3->4. Then, perform cycle detection on the `graph_with_cycle`.
Oracle	The cycle_detection routine will return a linked list of integers, containing 1 and 2, meaning that there exists a cycle 1->2->1.

Test ID	2.1.4.2
Requirements under test	3.2.06, 3.2.07, 3.2.08
Routines under test	Class: ALGORITHM cycle_detection (graph: DIRECTED_GRAPH) : BOOLEAN
Description	Every node is uniquely identified by an id (INTEGER) in the directed graph. The cycle_detection must return an empty linked list if no cycle is found in the graph.
Set-up	An instance of ALGORITHM should be created.
Test data	An instance `graph_with_cycle:` DIRECTED_GRAPH should be created, with four nodes, and four edges as following: 1->2, 1->3, 2->4, 3->4. Then, perform cycle detection on the `graph_with_cycle.`
Oracle	By calling cycle_detection (graph: DIRECTED_GRAPH), an empty linked list will be returned.

Test ID	2.1.4.3
Requirements under test	3.2.06, 3.2.07, 3.2.08, 3.2.09, 3.2.10
Routines under test	Class: ALGORITHM topo_sort (graph: DIRECTED_GRAPH) : LINKED_LIST[INT]
Description	Every node is uniquely identified by an id (INTEGER) in the directed graph. The ALGORITHM must be able to perform topological sort on the non-cyclic part if no cycle is found in the graph.
Set-up	An instance of ALGORITHM should be created
Test data	An instance `graph_with_cycle:` DIRECTED_GRAPH should be created, with four nodes, and four edges as following: 1->2, 2->3, 3->2, 3->4. Then, perform topo sort on the `graph_with_cycle`.
Oracle	The cycle_detection routine will return a linked list of one single integer, i.e. 1, meaning that 1 is the only element in the non-cyclic part of the graph.

Test ID	2.1.4.4
Requirements under test	3.2.06, 3.2.07, 3.2.08
Routines under test	Class: ALGORITHM topo_sort (graph: DIRECTED_GRAPH) : LINKED_LIST[INT]
Description	Every node is uniquely identified by an id (INTEGER) in the directed graph. The algorithm must be able to perform topological sort, if no cycle is found in the graph.
Set-up	An instance of ALGORITHM should be created.
Test data	An instance `graph_with_cycle:` DIRECTED_GRAPH should be created, with four nodes, and four edges as following: 1->2, 1->3, 2->4, 3->4. Then, perform topo_sort on the `graph_with_cycle.`
Oracle	By calling topo_sort (graph: DIRECTED_GRAPH), a linked list of 2 integers will be returned, i.e. 1->2->3->4 or 1->3->2->4.

Test ID	2.1.5
Routines under test	DISPLAYER. to_graphviz_format (graph: DIRECTED_GRAPH, converter: CONVERTER output_file_name: STRING)
Description	The library should be able to generate a text file with compatible graphviz format for display purposes.
Set-up	An instance of DISPLAYER should be created.
Tear-down	The output graphviz text file(`output_graph_test_displayer.txt`) should be deleted from the file system.

Test data	`converter.insert_name ("event1")`
	`converter.insert_name ("event2")`
	`converter.insert_name ("event3")`
	`converter.insert_name ("event4")`
	`converter.delete_name ("event3")`
	`graph.set_largest_node (4)`
	`graph.add_edge (1, 2)`
	`graph.add_edge (1, 3)`
	`graph.add_edge (1, 4)`
	`graph.add_edge (2, 3)`
	`graph.add_edge (2, 4)`
	`graph.add_edge (3, 4)`
	`graph.delete_edge (1, 2)`
	`graph.delete_node (3)`

Oracle	By calling to_graphviz_format, a graphviz text file (`output_graph_test_displayer.txt`) should be generated, containing. “event1->{event2}”, “event2->{event4}”, and “event4->{}”.

3. Expected Coverage

We expect that the tests described in section 2 covers all public routines of the 5 classes. Also the expected coverage rate over all branches would be 80%.

4. Benchmarks

We will use a randomly generated acyclic directed graph with 2,000 elements and 100,000 constraints as the benchmark. The library should be able to generate a text file for Graphviz in 500ms, and perform topological sort in 1,000ms. The results would be compared among different operating systems, including MacOSX, Windows, Ubuntu.