CITS5501 Software Testing and Quality Assurance
CITS5501
Semester 1, 2018
Workshop 3 – Whitebox testing – solution

1. Control flow

Consider the following Java method for collapsing sequences of blanks, taken from the `StringUtils` class of Apache Velocity (http://velocity.apache.org/), version 1.3.1.

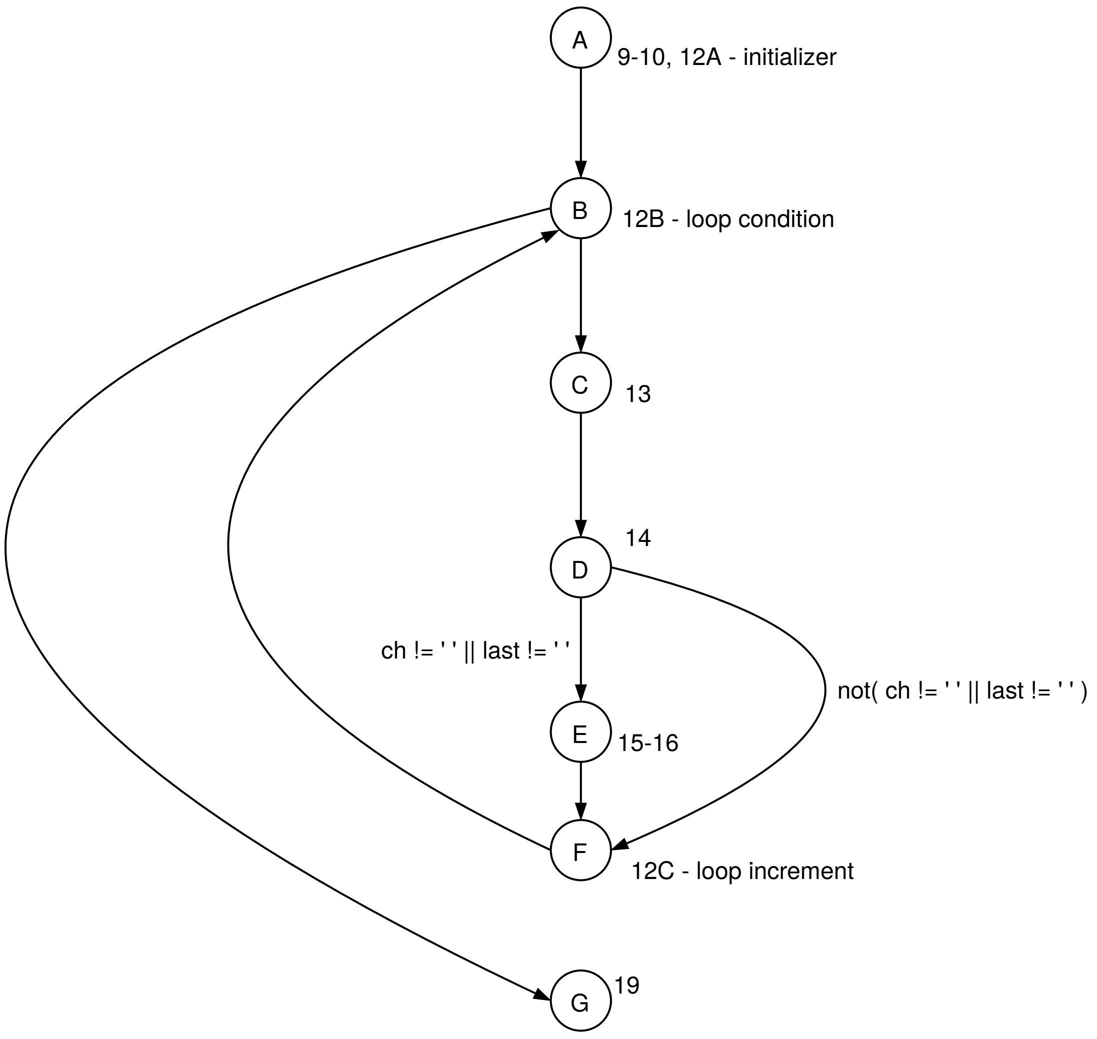
```java

/**
* Remove/collapse multiple spaces.
*
* @param String string to remove multiple spaces from.
* @return String
*/
public static String collapseSpaces(String argStr) {
  char last = argStr.charAt(0);
  StringBuffer argBuf = new StringBuffer();

  for (int cIdx = 0 ; cIdx < argStr.length(); cIdx++) {
    char ch = argStr.charAt(cIdx);
    if (ch != ' ' || last != ' ') {
      argBuf.append(ch);
      last = ch;
    }
  }
  return argBuf.toString();
}
```

1. Construct a control flow graph of the method. How many nodes are required? How many edges?

Sample solution:



- The graph contains 7 nodes, and 8 edges

2. Can you construct test cases that execute the loop …
   … zero times?
   … once?
   … more than once?

Sample solution:

- zero times: pass the empty string – `collapseSpaces("")`
- one time: pass any length-1 string – e.g. `collapseSpaces("a")`
- more than once: pass any length-2 string – e.g. `collapseSpaces("ab")`

3. What sort of coverage do your test cases have? Do they have …
   … node coverage?
   … statement coverage?
   … edge coverage?

Solution:

- Even when combined, the above test cases don't have any of those sorts of coverage. None of them execute the statements within the "if" block, so they are missing coverage of the nodes, statements and edges representing the contents of the "if" block.

4. What are the prime paths in the graph? (Reminder: a simple path never re-visits the same node, except that the first and last nodes may be the same. A prime path is a simple path that is not a proper subpath of any other simple path – i.e., it's a sort of "maximal simple path".)

   Do your tests have prime path coverage?

Solution:

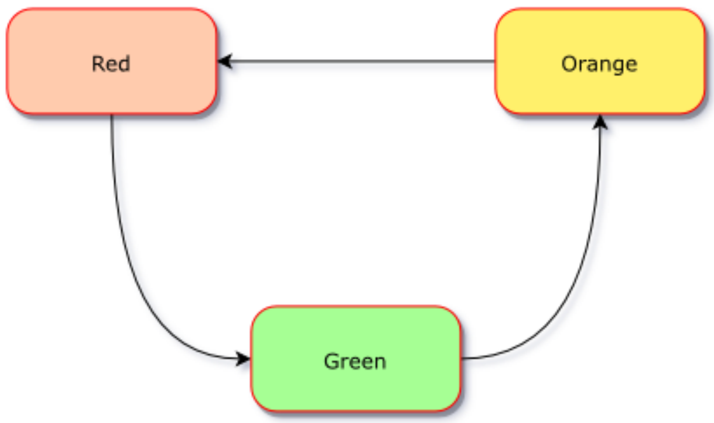
- The prime paths are:
  ABG, ABCDEF, ABCDF,
  BCDEFB, CDEFBC, DEFBCD, EFBCDEF, FBCDEF *(these all execute the left-hand 'if' branch)*
  BCDFB, CDFBC, DFBCD *(these all execute the right-hand 'if' branch)*
  EFBCDF, *(this takes one left and one right branch)*
  CDEFBG, *(takes left branch)*
  CDFBG *(take rights branch)*
- The tests specified above, when combined, certainly do not have prime path coverage. The empty string gives us the path ABG, a single non-space letter gives us ABCDF, and two non-space letters gives us (for instance) DFBCD.
  We would at a minimum need to specify some tests which pass the "if" conditional, if we wanted to get prime path coverage. For instance, the two-letter string `" a"` gives us (among others) coverage of the path EFBCD.

2. State diagrams

State diagrams are used to give an abstract description of the behaviour of a stateful system. *States* are represented by nodes in a directed graph, and permissible *transitions* between states are represented by edges.

For example, the following state diagram could be used to represent a (very simple) traffic light system[1]:

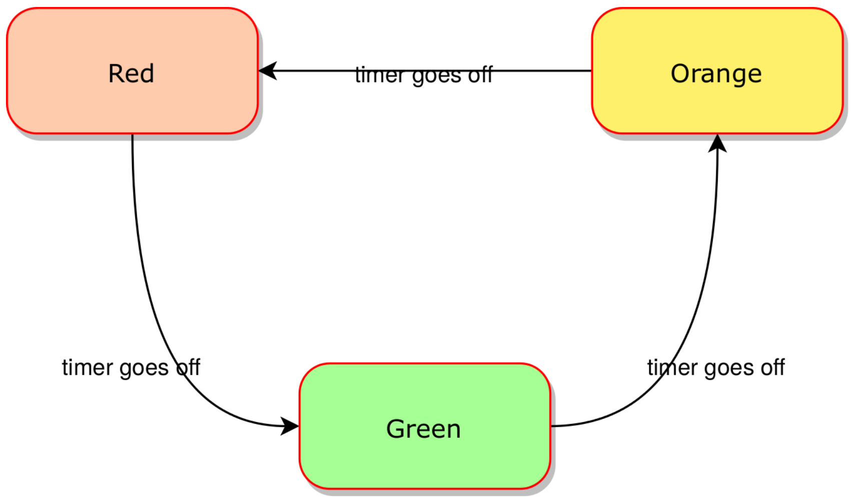
[1]Bonus question: how hard do you think it is to hack into a traffic light system? (See https://www.csoonline.com/article/2466551/microsoft-subnet/hacking-traffic-lights-with-a-laptop-is-easy.html.) Is this a scenario that could have been identified by any test case design techniques we have looked at?



This diagram doesn't represent any information about when the traffic light changes state, or under what circumstances – just what the valid *state transitions* are.

In most systems, we want to represent what *events* will *trigger* transitions. (These events might arise from within the system – e.g., a timer going off – or from outside – e.g. a button being pressed by a user.) We can indicate events using labels on the node edges. (This is a simplified version of the syntax used for UML state charts.)

Suppose for our traffic light system that an internal timer goes off every 10 seconds, making the system change state. (This is a *very* fast-changing traffic light. Perhaps the designers hate all pedestrians and drivers.) We could represent that as follows:



1. Draw a state diagram for a lift (or elevator, in American). The lift services a building with two floors. On each floor is a single button which call calls the lift to that floor. On the inside of the lift are two buttons, one which directs the lift to go to the ground floor (the "first floor", in American), and another which directs the lift to go the first floor (the "second floor", in American).

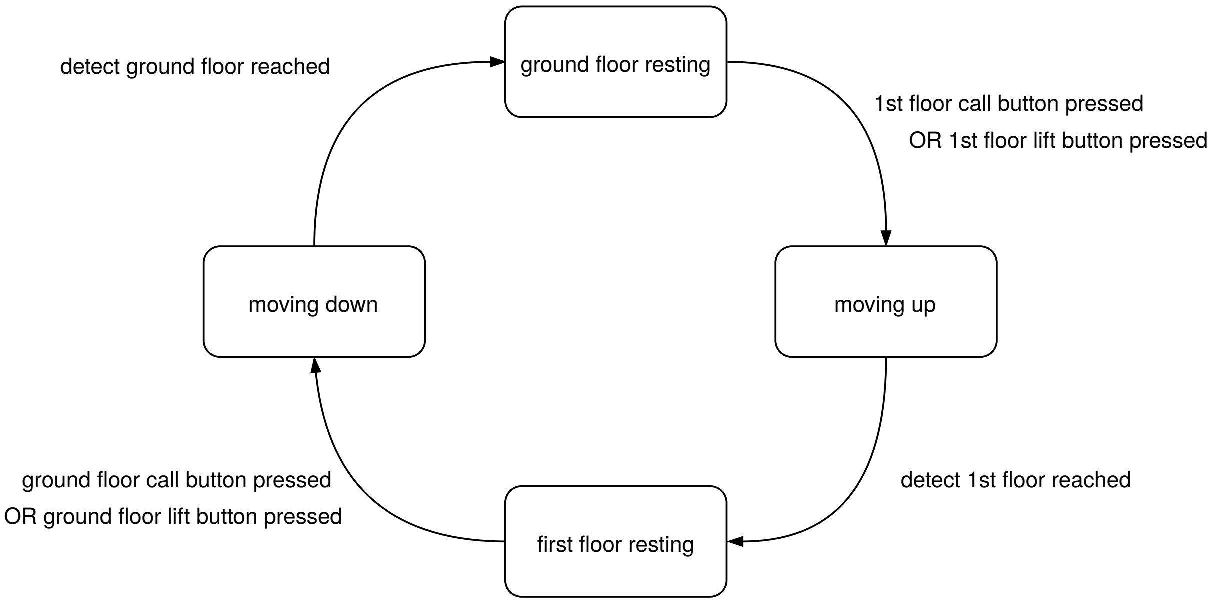
   What event types will be required? What states will be required?

2. We can also add *guards* on each edge, which limit the circumstances in which a transition will occur. These are added in square brackets after the event name.

   Are any guards needed for our lift system?

Solution:

- A state diagram for the lift is given below. Note that the question specifies an *extremely* simple model of a lift. It doesn't ask us to model "door open"/"door close" buttons for the lift; nor does the lift have any "memory" additional to the states. (For instance, in an actual lift, if the lift is travelling down, and gets a request to travel back *up*, the request might be added to a list of requests. In our simple model, if the lift is travelling down, and you are on the 1st floor and press the call button, this will have no effect – you will have to keep on pressing, until the elevator reaches the "ground floor resting" state.)


detect ground floor reached
ground floor resting
1st floor call button pressed
OR 1st floor lift button pressed
moving down
moving up
ground floor call button pressed
OR ground floor lift button pressed
detect 1st floor reached
first floor resting

- This model has 6 event types that can occur - ground floor call button pressed, ground floor lift button pressed, 1st floor call button pressed, 1st floor lift button pressed, detect ground floor reached, and detect 1st floor reached. It does not use any guards.

3. If we define test cases for this system, what will be the inputs? And what will be the outputs?

   Can you define a set of test cases which give *edge coverage* of the lift system?

Sample solution:

- A test case for this system will take the form of a starting state, and a sequence of events. If we consider the inputs to be "everything required to run the test case", then that will comprise both the starting state and the sequence of events.

  For each test case, we would have an *expected output*, which is the sequence of states we expect the system to go through, and an *actual output*, which would be the sequence of states we actually observe occurring.

- To define test cases concisely, let us abbreviate the events as follows:
  - GR – ground floor reached
  - 1R – 1st floor reached
  - GC – ground floor call button pressed
  - 1C – 1st floor call button pressed
  - GL – ground floor lift button pressed
  - 1L – 1st floor lift button pressed.

  And let us abbreviate the states:
  - G – ground floor resting
  - 1 – 1st floor resting
  - MD – moving down
  - MU – moving up
- Consider the following test case:
  - start state: G
  - events: 1C, 1R, GL, GR

  This test case all on its own gives us edge coverage.