Assignment 1 solution

ECE 653

Yanxin Wang

20439686

y574wang@uwaterloo.ca

Question 1

For -5%2, C# and Java give -1 while Perl and Python give 1.

Two strategies to cope with this problem:

1. Do not use % for modulo calculation, write your own method instead. This can be implemented by overriding operator % or just writing a new method to conduct this type of calculation.
2. Modify compliers to give out warnings when see any % operation on negative values.

Question 2

(a) A test case that does not execute fault: X = []

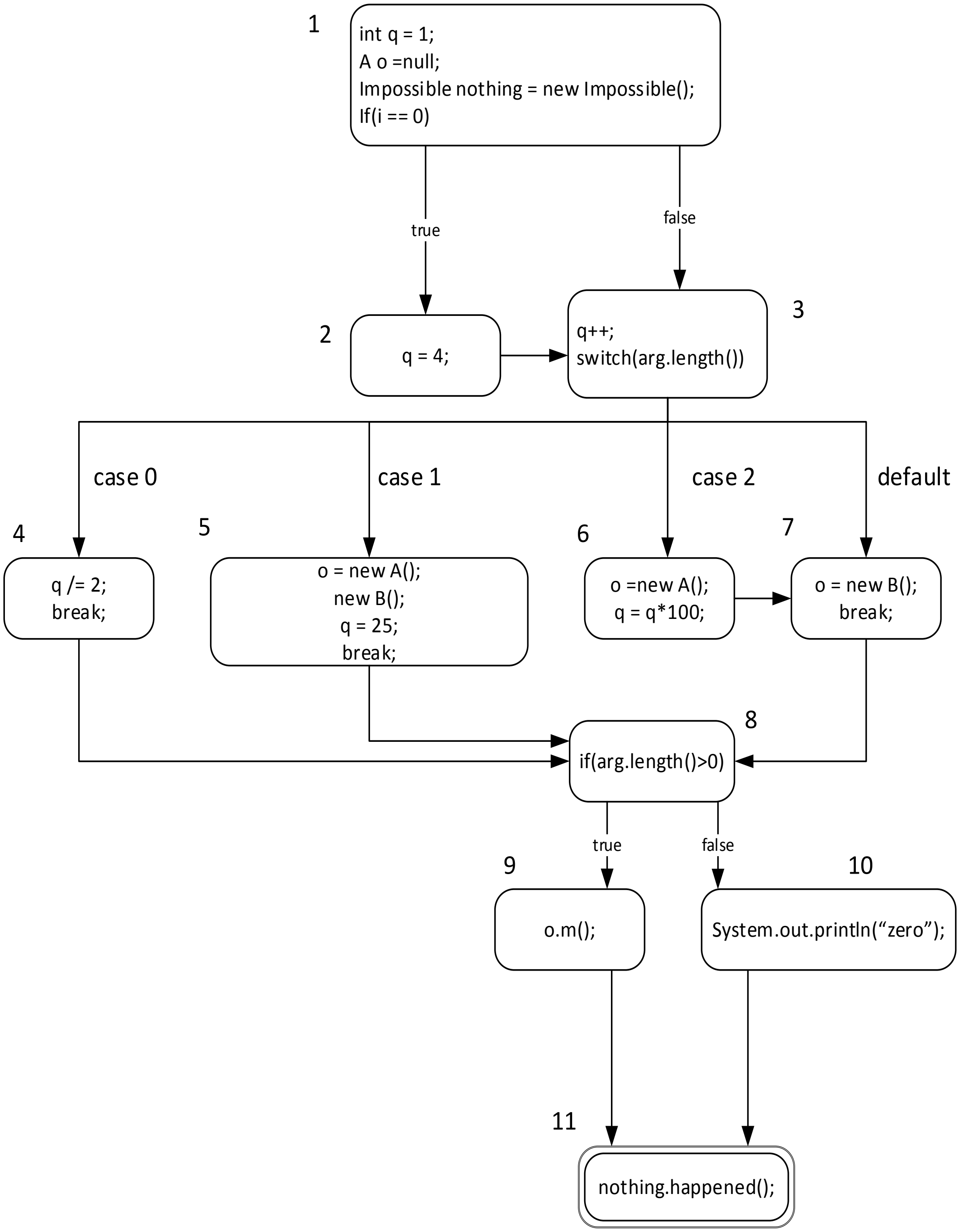
(b) A test case that executes the fault, but not result in an error state: X = [2]
As long as there is no negative odd number in X, X satisfied test requirement.

(c) No such test case. Any error will definitely cause a failure.

(d) First error state:
X = [-10,-9,0,99,100]
i = 1
count = 0
PC = i++

Question 3

(a) Control Flow Graph



(b) $TR_{NC} = \{1,2,3,4,5,6,7,8,9,10,11\}$

TR_{EC}
$= \{[1,2], [1,3], [2,3], [3,4], [3,5], [3,6], [3,7], [4,8], [5,8], [6,7], [7,8], [8,9], [8,10], [9,11], [10,11]\}$

TR_{EPC} = {[1,2,3], [1,3,4], [1,3,5], [1,3,6], [1,3,7],
[2,3,4], [2,3,5], [2,3,6], [2,3,7], [3,4,8], [3,5,8], [3,6,7], [3,7,8],
[4,8,9], [4,8,10], [5,8,9], [5,8,10], [6,7,8], [7,8,9], [7,8,10], [8,9,11], [8,10,11]}

Some edge-pair are infeasible because the value of arg.length() affects both "switch" and "if" sentences. Therefore, [4,8,9],[5,8,10],[7,8,10]are infeasible.

$TR_{\text{feasibleEPC}}$ = {[1,2,3], [1,3,4], [1,3,5], [1,3,6], [1,3,7],
[2,3,4], [2,3,5], [2,3,6], [2,3,7], [3,4,8], [3,5,8], [3,6,7], [3,7,8],
[4,8,10], [5,8,9], [6,7,8], [7,8,9], [8,9,11], [8,10,11]}

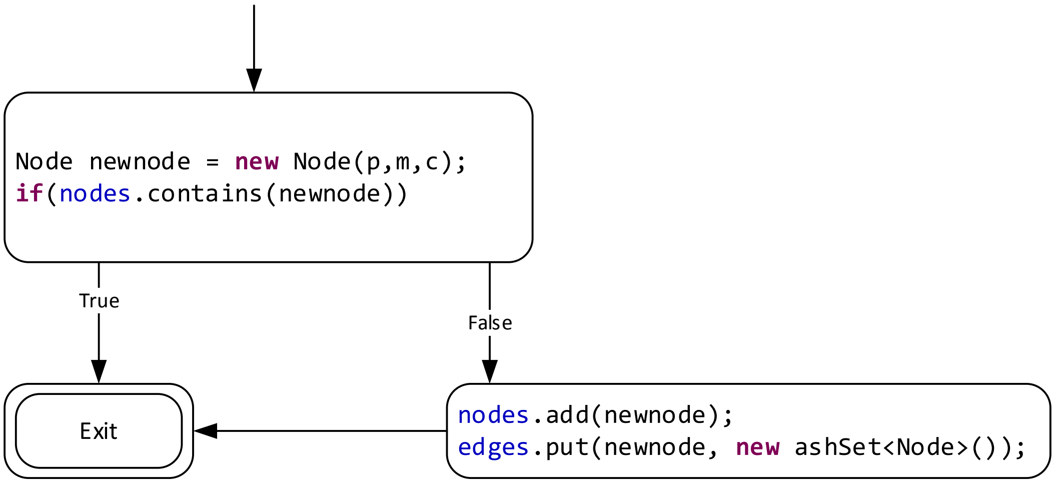
$TR_{feasiblePPC}$ = {[1,2,3,4,8,10,11], [1,2,3,5,8,9,11], [1,2,3,6,7,8,9,11], [1,2,3,7,8,9,11],
[1,3,4,8,10,11], [1,3,5,8,9,11], [1,3,6,7,8,9,11], [1,3,7,8,9,11]}

$TR_{infeasiblePPC}$ = {[1,2,3,4,8,9,11], [1,2,3,5,8,10,11], [1,2,3,6,7,8,10,11], [1,2,3,7,8,10,11],
[1,3,4,8,9,11], [1,3,5,8,10,11], [1,3,6,7,8,10,11], [1,3,7,8,10,11]}

Question 4

Testing:

1. addNode



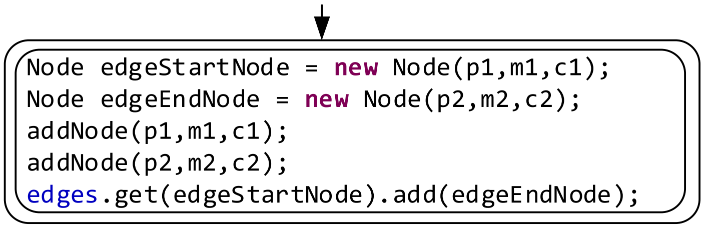
```
Node newnode = new Node(p,m,c);
if(nodes.contains(newnode))
```

```
nodes.add(newnode);
edges.put(newnode, new ashSet<Node>());
```

a) Node Coverage is satisfied.
TestCase={addNode,addNode_duplicate}

b) Edge Coverage is satisfied.
TestCase={addNode,addNode_dulplicate}

2. addEdge

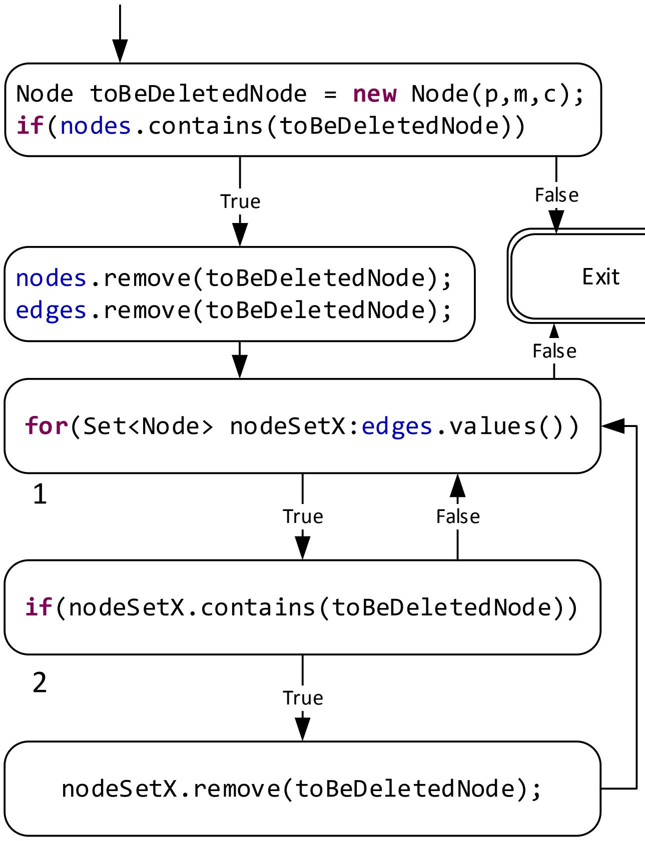


```
Node edgeStartNode = new Node(p1,m1,c1);
Node edgeEndNode = new Node(p2,m2,c2);
addNode(p1,m1,c1);
addNode(p2,m2,c2);
edges.get(edgeStartNode).add(edgeEndNode);
```

a) Node Coverage is satisfied.
TestCase={addEdge, addEdge_oneNewNode}

b) Edge Coverage is satisfied.
TestCase={addEdge, addEdge_oneNewNode}

3. deleteNode



```
Node toBeDeletedNode = new Node(p,m,c);
if(nodes.contains(toBeDeletedNode))
```

```
nodes.remove(toBeDeletedNode);
edges.remove(toBeDeletedNode);
```

```
for(Set<Node> nodeSetX:edges.values())
```

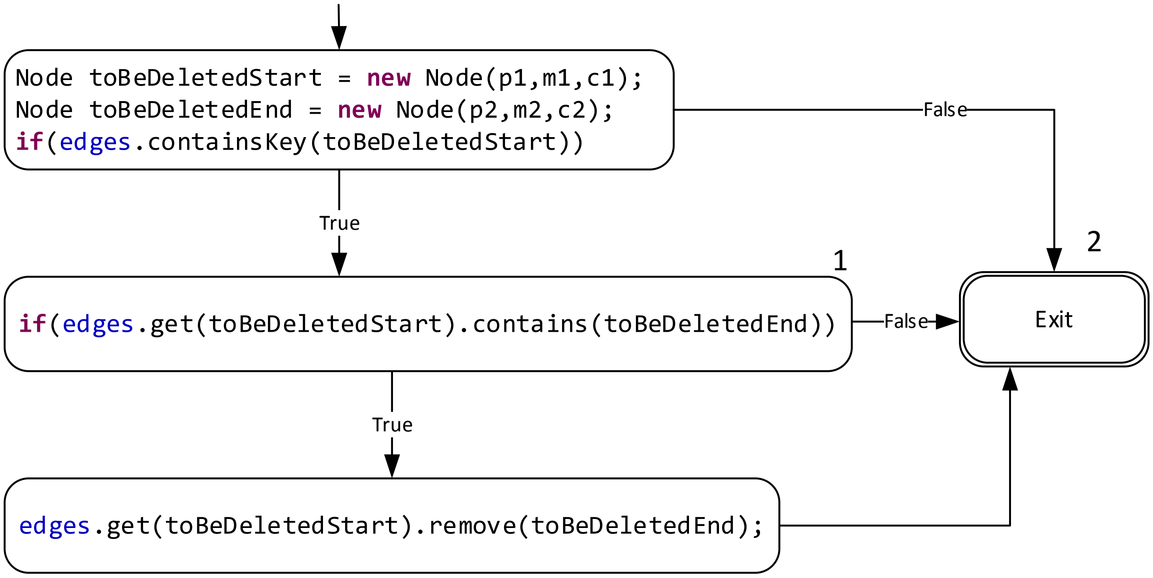
```
if(nodeSetX.contains(toBeDeletedNode))
```

```
nodeSetX.remove(toBeDeletedNode);
```

a) Node Coverage is satisfied.
TestCase={deleteNode}

b) Edge Coverage is unsatisfied.
Edge[2,1] is not covered.
TestCase={deleteNode, deleteNode_missing, deleteNode_isolatedNode}

4. deleteEdge



```
Node toBeDeletedStart = new Node(p1,m1,c1);
Node toBeDeletedEnd = new Node(p2,m2,c2);
if(edges.containsKey(toBeDeletedStart))
```

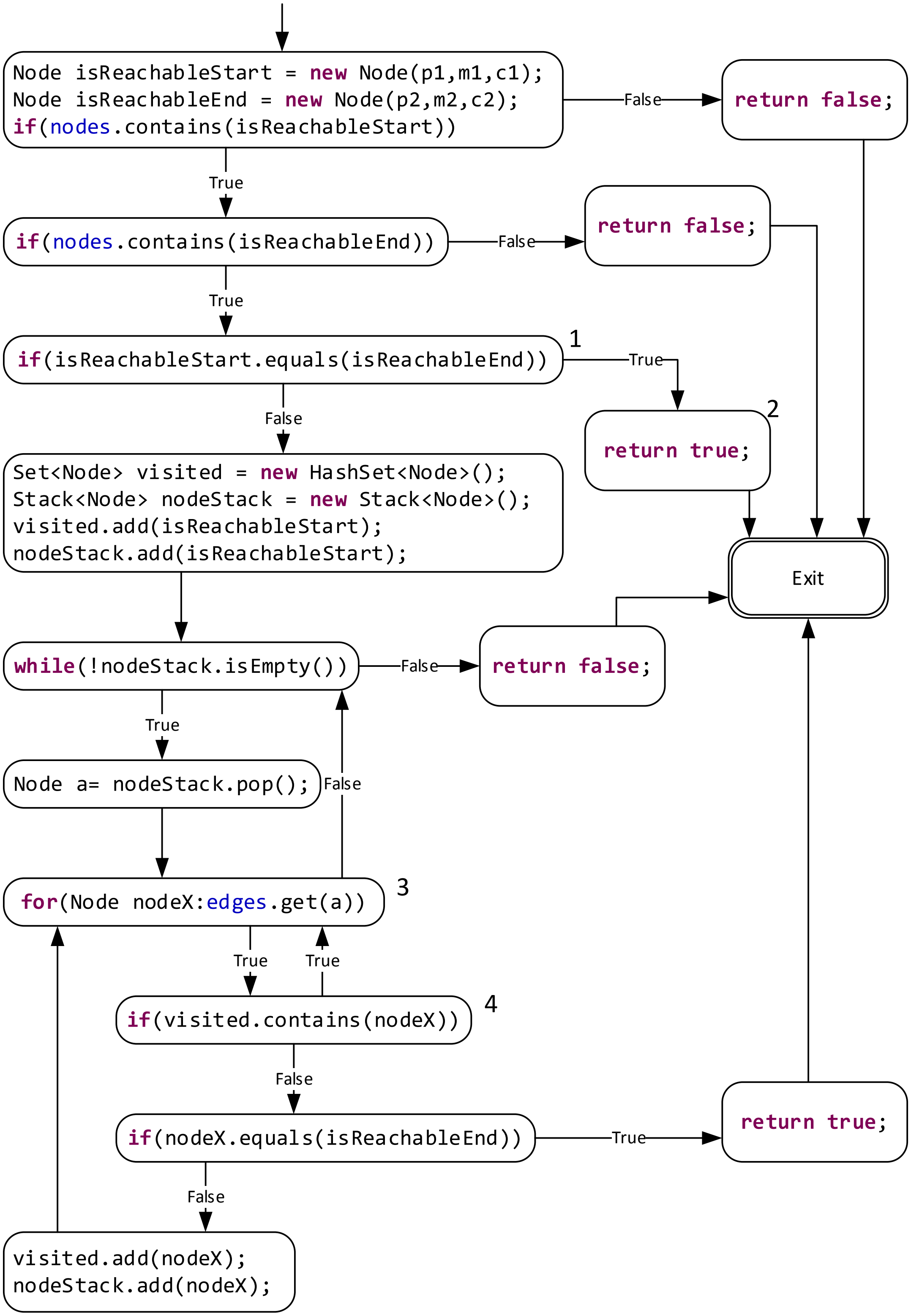
```
if(edges.get(toBeDeletedStart).contains(toBeDeletedEnd))
```

```
edges.get(toBeDeletedStart).remove(toBeDeletedEnd);
```

a) Node Coverage is satisfied.
   TestCase={deleteEdge}

b) Edge Coverage is unsatisfied.
   Edge[1,2] is not covered.
   Add another test case, deleteEdge_missingTargetNode, to satisfy test requirement.
   TestCase={deleteEdge, deleteEdge_missingSrcNode, deleteEdge_missingTargetNode}

5. isReachable



```
Node isReachableStart = new Node(p1,m1,c1);
Node isReachableEnd = new Node(p2,m2,c2);
if(nodes.contains(isReachableStart))
```

```
return false;
```

```
if(nodes.contains(isReachableEnd))
```

```
return false;
```

```
if(isReachableStart.equals(isReachableEnd))
```

```
return true;
```

```
Set<Node> visited = new HashSet<Node>();
Stack<Node> nodeStack = new Stack<Node>();
visited.add(isReachableStart);
nodeStack.add(isReachableStart);
```

```
while(!nodeStack.isEmpty())
```

```
return false;
```

```
Node a= nodeStack.pop();
```

```
for(Node nodeX:edges.get(a))
```

```
if(visited.contains(nodeX))
```

```
if(nodeX.equals(isReachableEnd))
```

```
return true;
```

```
visited.add(nodeX);
nodeStack.add(nodeX);
```

a) Node Coverage is unsatisfied.
Node 2 is not covered.
Add another test case, reachable_srcEqualsTarget, to satisfy the test requirement.
TestCase={ reachable_true, reachable_unreachable, reachable_missingSrc, reachable_missingTarget, reachable_srcEqualsTarget}

b) Edge Coverage is unsatisfied.
Edge[1,2] is not covered.
Add another test case, deleteEdge_srcEqualsTarget, to satisfy test requirement.
TestCase={ reachable_true, reachable_unreachable, reachable_missingSrc, reachable_missingTarget, reachable_srcEqualsTarget}
*Note:
Edge[4,3] is covered in test case {reachable_unreachable} where we check if there is a path from node(M.mnull:59) to node(M.mnull:0). Because node(M.mnull:69) is visited twice(from node(M.mnull:63) and node(M.mnull:63)).
The content of cfg can be observed in following method:

```
for(CFG.Node x:cfg.nodes){
        System.out.println(x.toString()+cfg.edges.get(x).toString());
}
```