# Ingeniería del Software II

Práctica #2 – Testing Automatizado

## Parte 1 - Mutation Analysis

### Ejercicio 1

Sea el siguiente programa `foo` escrito en el lenguaje Java:

```
int foo(int[] A, int[] B) {
 int r = 0;
 for (int i = 0; i < A.length; i++) {
   r += A[i] * B[i];
 }
 return r;
}
```

Escribir una expresión booleana Java que sea la pre-condición más débil que impida que se produzcan excepciones (NullPointerException y IndexOutOfBoundsException) durante la ejecución del programa.

### Ejercicio 2

Sea el siguiente programa:

```
int foo(int x, int y) {
L1:  int z = 0;
L2:  if (x<=y) {
L3:    z = x;
     } else {
L4:    z = y;
     }
L5:  return z;
}
```

Y el siguiente test suite:

```
void test1() {
  int rv = foo(1,0);
  assertEquals(0, rv);
}
```

Proveer un nuevo test case que logre 100 % de cobertura de sentencias y de branches

### Ejercicio 3

Sea el siguiente programa original Java:

```
int foo(int x, int y) {
  int z = 0;
  if (x <= y) {
    z = x;
  } else {
    z = y;
  }
  return z;
}
```

Y el siguiente mutante al que llamaremos mutante #2:

```
int foo(int x, int y) {
  int z = 0;
  if (x != y) { // x<=y
    z = x;
  } else {
     z = y;
   }
   return z;
}
```

Y el siguiente test suite:

```
void test1() {
  assertEquals(0, foo(0,1));
}
void test2() {
  assertEquals(0, foo(0,0));
}
```

Extender el test suite con un nuevo test capaz de detectar al mutante #2.

**Ejercicio 4**

Sea el siguiente programa que clasifica un triángulo por sus lados:

```
int triangle(int a, int b, int c) {
 if (a <= 0 || b <= 0 || c <= 0) {
   return 4; // invalid
 }
 if (! (a + b > c && a + c > b && b + c > a)) {
   return 4; // invalid
 }
 if (a == b && b == c) {
   return 1; // equilateral
 }
 if (a == b || b == c || a++ == c) {
   return 2; // isosceles
 }
 return 3; // scalene
}
```

Escribir un mutante no-equivalente del mismo y presentar un test case capaz de matar al mutante.

## Generación Automática de Tests

**Ejercicio 5**

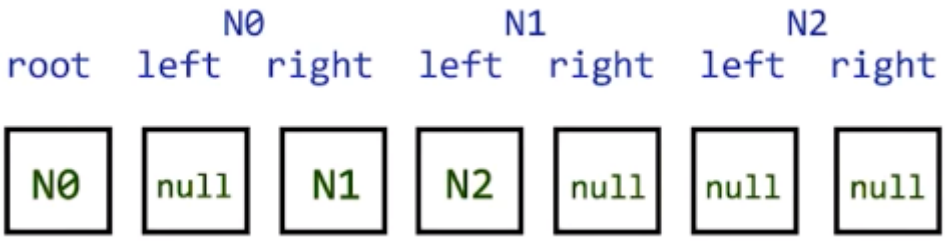
Sea la siguiente estructura de datos:

```
class BinaryTree {
 Node root;
 class Node {
   Node left;
   Node right;
  }
}
```

Y el siguiente código de repOK (precondición) que valida si el invariante de representación de la estructura es válido o no.

```
public boolean repOK(BinaryTree bt) {
  if (bt.root == null) return true;
  Set visited = new HashSet();
  List workList = new LinkedList();
  visited.add(bt.root);
  workList.add(bt.root);
  while (!workList.isEmpty()) {
    Node current = workList.removeFirst();
    if (current.left != null) {
      if (!visited.add(current.left)) return false;
      workList.add(current.left);
    }
    if (current.right !=null) {
      if (!visited.add(current.right)) return false;
      workList.add(current.right);
    }
  }
  return true;
}
```

Si el estado del vector candidato de Korat es el siguiente:

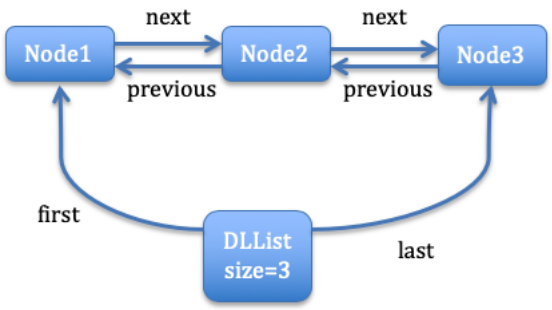



¿Cuál serán las siguientes dos instancias válidas (repOK==true) y no isomorfas que generará Korat? Indicar el orden de lectura de campos en todos los casos (incluyendo el vector candidato que se muestra arriba)

**Ejercicio 6**

Escribir el repOK() para que la siguiente estructura de datos sea una lista doblemente encadenada:

```
class Node {
 Node next;
 Node previous;
}
class DLList {
 Node first;
 Node last;
 int size;
}
```




**Ejercicio 7**

Sea el siguiente código de Binary Tree:

```
class BinaryTree {
  Node root;
  public BinaryTree(Node r) {
    root = r;
    assert(repOk(this));
  }
  public Node removeRoot() {
    assert(root != null);
    ...
  }
}
```

```
class Node {
 Node left;
 Node right;
 public Node(Node l, Node r) {
   left = l;
   right = r; }
}
```

1. ¿Cuál es la secuencia más pequeña que puede generar Randoop para crear una instancia válida de BinaryTree?
2. ¿Cuál es la secuencia más pequeña que puede generar Randoop para violar la aserción de removeRoot()?
3. ¿Cuál es la secuencia más pequeña que puede generar Randoop que viole la aserción en el constructor de BinaryTree?

Una vez generadas, indicar cómo clasificará Randoop a estas secuencias:

1. La descartará como una secuencia ilegal (i.e. incumple precondiciones)
2. La clasificará como un bug
3. La agregará como una secuencia de invocaciones a métodos para futuras extensiones

## Parte 3 - Ejecución simbólica dinámica

### Ejercicio 8

DSE testea el programa siguiente comenzando con el input x=1. ¿Cuál es el valor de "x" y el valor de la condición de ruta enviada al demostrador de teoremas para cada iteración de DSE?

Use DFS para explorar el árbol de cómputo y un unroll de u=3.

```
int test_me(int x) {
  int[] A = { 5, 7, 9 };
  int i = 0;
  while (i<3) {
   if (A[i]==x) break;
   i++;
  }
  return i;
}
```

A modo de ayuda, en la primer iteración del ejecución del algoritmo DSE se cumple que el valor concreto de x es "1" y el valor de la condición de ruta enviada para ser resuelta por el demostrador de teoremas es:

$$5 \neq x_0 \wedge 7 \neq x_0 \wedge 9 = x_0$$

## Parte 4 - Search-based Testing

### Ejercicio 9

Sea el siguiente programa:

```
static void foo(int x){
L1: if (x>0) {
L2:    while (x!=0) {
L3:       x=x-1;
L4:       if (x==0) {
L5:          System.out.println("Gol!");
          }
       }
    }
L6: return;
}
```

1. Dibujar el control-flow graph del programa "foo"
2. Dibujar el control-dependy graph del programa "foo"
3. Asumiendo un valor de K=5 ¿cuáles son las distancias de branch para el input $x = -15$ en los branch L1-true y L1-false?
4. Asumiendo un valor de K=5 ¿cuáles son las distancias de branch para el input $x = 10$ en los branch L4-true y L4-false?