Fundamentos de Algoritmos

Profs. Agustín Gravano y Hernán Czemerinski

Primer Semestre de 2020

Clase 6: Verificación de programas

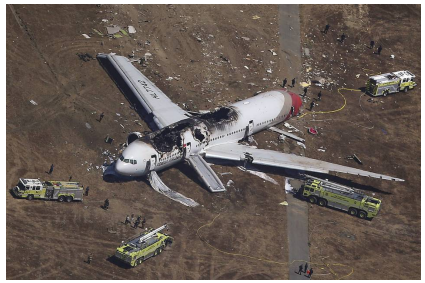
Verificación de programas

¿Cómo podemos saber si nuestro código es correcto (es decir, hace lo que esperamos que haga)?

Verificación de programas

Los errores de software pueden causar enormes pérdidas de dinero y vidas humanas.

Therac-25, 3 muertos (1985)

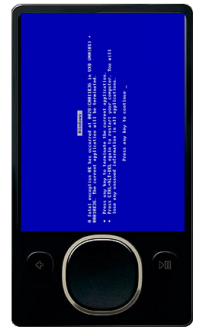


Airbus A300, 264 muertos (1994)

Knight Capital, 440M USD (2012)

Ariane 5 (1996)

Y2K (1999/2000)



MS Zune (2008/2009)

`http://www.cse.psu.edu/~gxt29/bug/softwarebug.html`

Verificación formal de programas

Demostrar formalmente que el programa cumple con una especificación para todos los posibles datos de entrada.

- Rara vez se cuenta con una especificación formal (matemática) de un programa.
- Hacerlo a mano es muy difícil y propenso a errores. Es equivalente a demostrar un teorema matemático.
- Se pueden usar herramientas (*model checkers*, demostradores de teoremas, etc.), pero sus algoritmos de verificación tienen altísimos órdenes de complejidad. Solo aplicable a pequeños fragmentos de un programa.

Verificación de programas: Revisión por pares

1. Cada programador escribe su código, lo prueba, lo corrige, etc.
2. Al terminar un módulo, lo somete a evaluación de 2+ pares.
3. Los pares revisan el código: su funcionamiento y también las reglas de formato, comentarios, documentación, etc.
4. Cada par revisor puede aprobar o rechazar el código, dando el feedback necesario.
5. Cuando todos los revisores aprobaron el código, ya se puede subir a producción.

Este enfoque tiene dos ventajas:

- Varios ojos ven más que dos.
- Saber que van a inspeccionar nuestro código nos predispone a ser más cuidadosos, ordenados, claros, etc.

Verificación de programas: Testing

El software se pone a prueba en algunos contextos de uso y con una selección de los posibles datos de entrada, buscando que sean representativos del conjutno total de entradas posibles.

> El testing puede demostrar la presencia de errores; nunca su ausencia. (Edsger W. Dijkstra)

Idealmente, los tests debe diseñarlos alguien que no haya programado el software.

La forma más difundida y sencilla es el testing de unidad:

- Testear cada módulo por separado (ej: una función, un tipo).
- Elegir pares entrada-salida que deben cumplirse.
- Ejecutar el módulo para cada entrada, y revisar que se cumpla la salida esperada.

Verificación de programas: Testing

Ejemplo: testing de unidad para la función raíz cuadrada entera

Entrada	Salida esperada
1	1
2	1
4	2
9	3
15000	122
15129	123
...	...

Ejercicio. Pensar y escribir (en papel) pares de entrada/salida para realizar tests de unidad a las siguientes funciones:

1. `suma(lst)`. Devuelve la suma de los elementos de la lista lst.
2. `es_primo(n)`. Devuelve True si n es primo y False en caso contrario.

Tests de unidad en Python

Python tiene una biblioteca para realizar tests de unidad que nos permite comparar el resultado de evaluar una expresión (por ej., el resultado de llamar a una función) contra un valor.

```
# Importamos la biblioteca de tests
import unittest
from suma_lista import suma   # Importamos el codigo a testear.

# Creamos el tipo en el que incluiremos los tests.
class TestSumaLista(unittest.TestCase):

  # Cada test de unidad debe comenzar con la palabra test.
  def test_todos_pos(self):
    self.assertEqual(suma([1,1,1,1]), 4)
    self.assertEqual(suma([1,2,3,4]), 10)

  def test_todos_neg(self):
    self.assertEqual(suma([-1,-1,-1,-1]), -4)
    self.assertEqual(suma([-1,-2,-3,-4]), -10)

  def test_pos_neg(self):
    self.assertEqual(suma([1,2,3,-3]), 3)
    self.assertEqual(suma([1,-1,1,-1]), 0)

  [...]

unittest.main()
```

Tests de unidad en Python

Otro ejemplo: tests para la función `es_primo(n)`.

```
import unittest
from numeros_primos import es_primo

class TestPrimos(unittest.TestCase):

  def test_primos(self):
    self.assertTrue(es_primo(5))
    self.assertTrue(es_primo(37))
    self.assertTrue(es_primo(101))

  def test_no_primos(self):
    self.assertFalse(es_primo(8))
    self.assertFalse(es_primo(44))
    self.assertFalse(es_primo(100))

  [...]

unittest.main()
```

https://docs.python.org/3/library/unittest.html

Tests de unidad en Python

Ejercicio. Descargar los archivos `suma_lista.py` y `numeros_primos.py`, y escribir tests de unidad en Python para las funciones `suma_lista(lst)` y `es_primo(n)` usando los valores de entrada/salida elegidos en el ejercicio anterior.
¡No mirar cómo están implementadas las funciones!

```
import unittest
from suma_lista import suma

class TestSumaLista(unittest.TestCase):
  [...]
```

```
import unittest
from numeros_primos import es_primo

class TestPrimos(unittest.TestCase):
  [...]
```

Depuración de programas

La depuración (o *debugging*) es el proceso de encontrar y corregir errores de programación.

```
import unittest
from numeros_primos import es_primo

class TestPrimos(unittest.TestCase):

  def test_uno(self):
    self.assertFalse(es_primo(1))

  [...]
```

```
======================================================================
FAIL: test_uno (__main__.TestPrimos)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "clase6/src/numeros_primos_test.py", line 7, in test_uno
    self.assertFalse(es_primo(1))
AssertionError: True is not false

----------------------------------------------------------------------
Ran 1 test in 0.000s

FAILED (failures=1)
```

Depuración de programas

Primer paso: localizar el error

```
def es_primo(n):
  for i in range(2,n):   # range(2,1) = { }
    if n % i == 0:
      return False
  return True
```

Segundo paso: corregirlo

```
def es_primo(n):
  if n == 1:
    return False
  for i in range(2,n):
    if n % i == 0:
      return False
  return True
```

Depuración de programas

Ejemplo: queremos testear el tipo Fecha.

- `Fecha(anio, mes, dia)`: Crea un nuevo elemento de tipo Fecha.
- `f.anio()`: Devuelve el año de la fecha f.
- `f.mes()`: Devuelve el mes de la fecha f.
- `l.dia()`: Devuelve el día de la fecha f.
- Operaciones para comparar fechas por igual (`==`) y por menor (`<`).

```
  def test_menor(self):
    fecha_1 = Fecha(2020,1,1)
    fecha_2 = Fecha(2020,4,29)
    self.assertTrue(fecha_1 < fecha_2)
```

```
======================================================================
FAIL: test_menor (__main__.TestFecha)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "clase6/src/tipo_fecha_test.py", line 9, in test_menor
    self.assertTrue(fecha_1 < fecha_2)
AssertionError: False is not true
----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (failures=1)
```

El test revela un error.

Depuración de programas

Ejemplo: queremos testear el tipo Fecha.

- `Fecha(anio, mes, dia)`: Crea un nuevo elemento de tipo Fecha.
- `f.anio()`: Devuelve el año de la fecha f.
- `f.mes()`: Devuelve el mes de la fecha f.
- `l.dia()`: Devuelve el día de la fecha f.
- Operaciones para comparar fechas por igual (==) y por menor (<).

```
def test_menor(self):
  fecha_1 = Fecha(2020,1,1)
  fecha_2 = Fecha(2020,4,29)
  self.assertTrue(fecha_1 < fecha_2)
```

```
def __lt__(self, other):
    if self._anio < other._anio:
        return True
    elif self._anio == other._anio and self._mes < other._mes:
        return True
    elif (self._anio == other._anio and self._mes == other._mes
          and self._dia < other._dia):
        return True
    else:
        return False
```

La función `__lt__` se encuentra correctamente implementada.

Depuración de programas

Ejemplo: queremos testear el tipo Fecha.

- `Fecha(anio, mes, dia)`: Crea un nuevo elemento de tipo Fecha.
- `f.anio()`: Devuelve el año de la fecha f.
- `f.mes()`: Devuelve el mes de la fecha f.
- `l.dia()`: Devuelve el día de la fecha f.
- Operaciones para comparar fechas por igual (`==`) y por menor (`<`).

```
def test_menor(self):
  fecha_1 = Fecha(2020,1,1)
  fecha_2 = Fecha(2020,4,29)
  self.assertTrue(fecha_1 < fecha_2)
```

```
def __init__(self, anio, mes, dia):
    self._anio = anio
    self._mes = anio
    self._dia = anio
```

¡Error de Copy + Paste!

Depuración de programas

Ejemplo: queremos testear el tipo Fecha.

- `Fecha(anio, mes, dia)`: Crea un nuevo elemento de tipo Fecha.
- `f.anio()`: Devuelve el año de la fecha f.
- `f.mes()`: Devuelve el mes de la fecha f.
- `l.dia()`: Devuelve el día de la fecha f.
- Operaciones para comparar fechas por igual (==) y por menor (<).

```
def test_menor(self):
  fecha_1 = Fecha(2020,1,1)
  fecha_2 = Fecha(2020,4,29)
  self.assertTrue(fecha_1 < fecha_2)
```

1. El programador introduce un error al escribir el código (ej: haciendo copy + paste).
2. Cuando el defecto se ejecuta, se genera una infección que corrompe el estado del programa (ej: cuando se crean las fechas).
3. La infección se propaga y se manifiesta como una falla (ej: cuando se comparan las fechas).

Depuración de programas

Al trabajar con programas grandes es difícil localizar errores. La falla puede producirse mucho después de la infección.

Una técnica de búsqueda consiste en inspeccionar el estado del programa (i.e., el valor de las variables) para recabar información que pueda guiarnos hacia el error. Esto podemos hacerlo agregando `prints` en nuestros programas.

```
def __lt__(self, other):
  print("self anio, mes, dia: ", self._anio, self._mes, self._dia)
  print("other anio, mes, dia: ", other._anio, other._mes, other._dia)
  if self._anio < other._anio:
    [...]

fecha_1 = Fecha(2020,1,1)
fecha_2 = Fecha(2020,4,29)
print("fecha_1 < fecha_2", fecha_1 < fecha_2)
```

```
$$ python tipo_fecha
self a/m/d: 2020/2020/2020
other a/m/d: 2020/2020/2020
fecha_1 < fecha_2: False
```

En el valor de las variables muestra `self._anio`, `self._mes` y `self._dia` se inicializaron con el valor correspondiente al año.

Depuración de programas

Ejercicio. Descargar el archivo `tipo_persona.py`, escribir tests de unidad en Python para testear el Tipo Persona y, en caso de producirse fallas, depurarlo.

Interfaz del tipo Persona:

- `Persona(nombre, apellido, dni, email)`: Crea un nuevo elemento de tipo Perona.
- `p.nombre()`: Devuelve el nombre de la persona p.
- `p.apellido()`: Devuelve el apellido de la persona p.
- `p.dni()`: Devuelve el dni de la persona p.
- `p.email()`: Devuelve el email de la persona p.
- Operaciones para comparar personas por igual y por menor, comparando en ambos casos el número de dni.

```
import unittest
from tipo_persona import Persona

class TestPersona(unittest.TestCase):
  [...]
```

Repaso de la clase de hoy

- Verificación de programas. Tres enfoques:
  - Verificación formal
  - Revisión del código por pares
  - Testing
- No podemos estar 100 % seguros de que un programa es correcto.
- Técnica simple de debugging: seguimiento de variables durante ejecución.

Con lo visto, ya pueden resolver la sección 7 de la guía de ejercicios.