Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC3745 — Testing
2023 - 2

Tarea 2

Fecha de entrega: Jueves 28 de Septiembre del 2023 a las 23:59

Información General

La siguiente tarea contempla como objetivo principal entender la lógica de una técnica de análisis dinámico usada en la actualidad: code instrumentation y sampling. Al implementar un instrumentador sencillo el estudiante podrá ver las ventajas, el alcance y las desventajas de las técnicas en general. No solo cuando uno implementa un instrumentador sino cuando uno usa alguno existente.

Objetivos

- Entender en detalle como funciona la técnica code instrumentation y sampling
- Entender las ventajas, desventajas, y limitaciones de las técnicas.
- Ser capas de adaptar un instrumentador para reportar cierta información de un programa analizado.

Contexto

En clase se vio dos técnicas de analisis dinámico: instrumentación(instrumentation) y muestreo(sampling). Esta tarea trata de utilizar estas dos muestras para recolectar información de ejecución de programas pequeños en Python.

Intrumentación

Modifique la herramienta de instrumeción de código fuente vista en clase para calcular las siguientes métricas:

- *Frecuencia (5 pts, básico)* – Esta métrica reporta el número de veces que una función fue llamada al ejecutar un programa.
- *Tiempo de Ejecución (15 pts, básico)* – Se debe mostrar el tiempo máximo, mínimo y promedio de ejecución de cada función llamada durante la ejecución de un programa. Note que cada función se puede ejecutar muchas veces. Cada vez que se ejecute una función se debe consultar el reloj del sistema operativo al iniciar y al finalizar la ejecución. La diferencia entre estas dos medidas sera el tiempo que tomo en ejecutar una función. Como una función se puede ejecutar muchas veces, se debe calcular el tiempo promedio, el tiempo máximo y el mínimo que toma ejecutar cada función del programa.
- *Callers (15 pts, intermedio)* – Un función puede ser invocada de diferentes partes dentro un programa, en particular, desde otras funciones. El reporte en consola debe reportar desde que otras funciones la función bajo análisis fue invocada. Pista, para implementar esta funcionalidad, se debe guardar la información del ultimo método que fue llamado antes de ejecutar cada función.
- *Cacheable (25 pts, avanzado)* – Normalmente, si una función devuelve siempre los mismos cuando recibe los mismos argumentos esta podría ser una cándida a implementar un cache. La herramienta debe reportar las funciones que pueden ser candidatas a aplicar cache. A contar del primer llamado a la función esta contara como candidata a cache y en caso de ser llamada múltiples veces bastara un llamado con argumentos o retorno distinto al resto de los llamados para descartarla como candidata a cache.

Valores de Retorno. Note que que implementar la ultima métrica no es posible con la forma actual con la que instrumentamos el código. Actualmente, agregamos una expresión al inicio y al final de la función. Por ejemplo, considere la siguiente función:

```
1 def foo():
2     for _ in range(10):
3         bar()
4         time.sleep(3)
```

Después de aplicar la instrumentación queda de la siguiente manera.

```
1 def foo():
2     Profiler.record_start('foo', [])
3     for _ in range(10):
4         bar()
5         time.sleep(3)
6     Profiler.record_end('foo', None)
```

Cuando existe una función que retorna un valor como en la función factorial, la instrumentación se ve de la siguiente forma:

```
def factorial(n):
    Profiler.record_start('factorial', [n])
    answer = 1
    time.sleep(3)
    if n > 0:
        answer = n * factorial(n - 1)
    Profiler.record_end('factorial', None)
    return answer
```

Por lo anterior, no se puede rastrear que exactamente devolvió la función. En este sentido no podemos monitorear si una función devuelve el mismo valor siempre. Para implementar la ultima métrica, primero es necesario modificar la forma en que instrumentamos. Usted debe modificar el instrumentador para que después de instrumentar quede de la siguiente forma:

```
def factorial(n):
    Profiler.record_start('factorial', [n])
    answer = 1
    time.sleep(3)
    if n > 0:
        answer = n * factorial(n - 1)
    return Profiler.record_end('factorial', answer)
```

Si se implementa bien, esta modificación no afectaria el computo de las anteriores métricas. Si usted no logra realizar esta modificación, deje la instrumentación antigua y obtendra los puntajes de las primeras 3 métricas.

Como lo pruebo. Para probar su implementación debe ejecutar el siguiente comando en consola:

```
instrumentor>python3 profile.py code1
```

code1, es el nombre del archivo a ejecutar. La carpeta input_code tiene varios archivos de prueba. El comando anterior debe arrojar el siguiente reporte en consola cuando termine de ejecutar el código a analizar:

```
fun     freq        avg         max         min         cache       callers
main    1           13.013      13.013      13.013       1           []
foo     1           13.013      13.013      13.013       1           ['main']
bar     2           5.011       5.015       5.007        0           ['foo']
```

El código base cuenta con 4 códigos para probar, ustedes puede agregar mas. El equipo docente tendrá sus propios códigos privados de prueba. Los códigos privados serán parecidos al los proporcionados en el código base.

Muestreo

Actualmente, nuestra herramienta de muestreo solo muestra las funciones que se encuentran dentro de la pila de ejecución cada segundo. Debe modificar la herramienta implementada en clase para sacar un reporte que muestre el tiempo de ejecución de cada método considerando el contexto en el que fue invocado. Por ejemplo, considere el siguiente código:

```
def main():
    foo()
    bar()
    zoo()
def foo():
    for _ in range(10):
        bar()
        time.sleep(1)
def zoo():
    foo()
def bar():
    time.sleep(1)
```

En el código anterior la función bar() se la llama desde en tres contextos diferentes:

1. *main, foo, bar* – Donde main llama a foo y foo llama a bar.
2. *main, bar* – Donde main llama a bar.
3. *main, zoo, bar* – Donde main llama a zoo y zoo a bar.

Call Context Tree (40 pts, intermedio/avanzado) – En esta tarea usted debe reportar el tiempo de ejecución que tiene cada método en diferente contexto. El reporte debe tener en forma de árbol. A esta estructura se la conoce como Call Context Tree. Por ejemplo, analizando el código anterior debe arrojar algo similar al siguiente reporte:

```
sampler> python3 profile.py code3
total (41 seconds)
  _bootstrap(41 seconds)
    _bootstrap_inner(41 seconds)
      run(41 seconds)
        execute_script(41 seconds)
          <module>(40 seconds)
            main(40 seconds)
              foo(19 seconds)
                bar(10 seconds)
              bar(1 seconds)
              zoo(20 seconds)
                foo(20 seconds)
                  bar(10 seconds)
```

Pista. Recuerde que la técnica de muestreo actual puede imprimir los métodos que se encuentran en la pila de ejecución cada segundo. Los elementos dentro de la pila ya están ordenados por contexto. Usted debe armar el árbol de llamadas utilizando la información que obtiene de la pila cada segundo. Recuerde que como se saca muestras cada segundo, el numero de veces que aparece un método en la pila es "equivalente" a su tiempo de ejecución.

Como lo pruebo. Para probar su implementación debe ejecutar el siguiente comando en consola:

```
sampler>python3 profile.py code1
```

code1, es el nombre del archivo a ejecutar. La carpeta input_code tiene varios archivos de prueba. El comando anterior debe arrojar algo similar al siguiente reporte en consola:

```
total (7 seconds)
  _bootstrap(7 seconds)
    _bootstrap_inner(7 seconds)
      run(7 seconds)
        execute_script(7 seconds)
          <module>(6 seconds)
            main(6 seconds)
              foo(6 seconds)
                bar(1 seconds)
                zoo(3 seconds)
                  bar(1 seconds)
```

El código base cuenta con 4 códigos para probar, ustedes puede agregar mas. El equipo docente tendrá sus propios códigos privados de prueba. Los códigos privados serán parecidos al los proporciandos en el código base.

Tareas

Desarrolle los siguientes ejercicios:

- ***Ejercicio 1*** – Implemente el calculo de las siguientes métricas en el instrumentor:
  - `Frecuencia`
  - `Tiempo de ejecución`
  - `Callers`
  - `Cacheable`

  Para implementar las métricas deberá modificar el código contenido en la carpeta `instrumentor`. Para obtener puntaje el valor calculado de las métricas se deberá poder observar en el reporte generado con el método print_fun_report. Si no es posible observar el calculo correcto de las métricas en el reporte no se asignara puntaje en el ítem de la métrica correspondiente.
- ***Ejercicio 2*** – Implemente el Call Context Tree utilizando muestreo. Para lograr este ejercicio deberá modificar el código contenido en la carpeta `sampler` y implementar el método `printReport` en la clase `Sampler` tal que este imprima el reporte del call context tree.

Entregables

- Debe subir todo el código de su tarea en el buzón en canvas mediante un archivo .zip.
- Junto al código deberán entregar un archivo README que contiene los nombres de quienes realizaron la tarea y su aporte (Declaración de tarea). Puede ser un archivo .md o un archivo de texto.

Archivos iniciales

El código base entregado esta organizado de la siguiente manera:

- *instrumentor(folder)* – contiene el código relacionado con el instrumentor.
- *instrumentor\input_code (folder)* – contiene códigos de ejemplo para probar el funcionamiento del instrumentor.

- *sampler (folder)* – contiene el código relacionado con el sampler.
- *sampler\input_code (folder)* – contiene códigos de ejemplo para probar el funcionamiento del sampler.

Para la tarea usted debe modificar o agregar código a las ubicaciones anteriormente mencionados según corresponda.

Reportar problemas en el equipo

En el caso de que algún integrante no aportara como fue esperado en la tarea, podrán reportarlo enviando un correo con asunto **Problema Equipo {NumeroGrupo} Testing** a *juanandresarriagada@uc.cl* explicando en detalle lo ocurrido. Posterior a eso se revisara el caso en detalle con los involucrados y se analizara si corresponde aplicar algún descuento. Instamos a todas las parejas que mantengan una buena comunicación y sean responsables con el resto de su equipo para evitar problemas de este estilo.
Advertencia: Si algún integrante del grupo no aporta en las tareas puede implicar recibir nota mínima en esa entrega.

Restricciones y alcances

- Su programa debe ser desarrollado en **Python 3.10**.
- No debe modificar los nombres de los archivos y clases entregados ya que de lo contrario los tests de la corrección podrían fallar. Pueden crear nuevas clases o archivos adicionales a los entregados para usar dentro de las clases pedidas.
- Los archivos de código entregados deben terminar con la extensión **.py**.
- En caso de dudas con respecto al enunciado deben realizarlas en un foro relacionado a la tarea que se encontrara disponible en canvas.
- Si no se encuentra especificado en el enunciado, supón que el uso de cualquier librería de Python adicional a las utilizadas en el código base se encuentran prohibidas. Esta permitido ocupar librerías nativas que hemos usado en clases por ejemplo operator, collections, functools, entre otras. En caso de que estimes necesario podrás preguntar en el foro de la tarea por el uso de alguna librería adicional.

Entrega

- **Código:** Deberán entregar todo el código por medio de un buzón de canvas habilitado para esta tarea.
- **Declaración de tarea:** Deberán entregar un archivo README que contiene los nombres de quienes realizaron la tarea y su aporte. La entrega de este archivo se realizara junto con el código por medio de un buzón de canvas habilitado para esta tarea.

Atraso: Cada Grupo cuenta con un cupón de atraso para las Tareas el cual entrega dos días adicionales para entregar la tarea. Para ocuparlo solo deben realizar una entrega pasado la fecha señalada en el enunciado. Realizar la entrega atrasada sin contar con el cupón implicara obtener nota mínima en la entrega.

Integridad académica

Este curso se adscribe al Código de Honor establecido por la Escuela de Ingeniería. Todo trabajo evaluado en este curso debe ser hecho **individualmente** o en **los grupos asignados** según sea definido en la evaluación y **sin apoyo de terceros**. Se espera que los alumnos mantengan altos estándares de honestidad académica, acorde al Código de Honor de la Universidad. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un Procedimiento Sumario. Es responsabilidad de cada alumno conocer y respetar el documento sobre Integridad Académica publicado por la Dirección de Pregrado de la Escuela de Ingeniería.

¡Éxito! :)