



República Bolivariana De Venezuela
Universidad Nacional Experimental De Guayana
Vicerrectorado Académico
Coordinación General De Pregrado
Proyecto de carrera: Ingeniería en informática
Unidad curricular: Sistemas distribuidos

Evaluación I. Parte I

Profesora:
Virginia Padilla

Bachiller:
Geruby Bermudez C.I: 26.264.179

Ciudad Guayana, julio 2022

El recolector de basura es un algoritmo que busca eliminar ciclo, los cuales son variables cuya referencia es distinta de cero, pero no se pueden acceder a ellas de forma convencional.

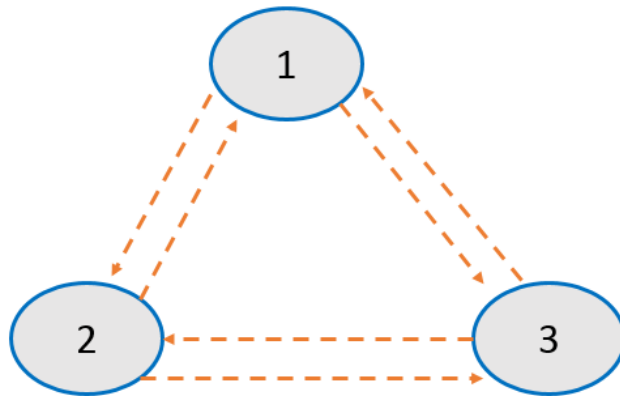


Figura 1. Referencias cíclicas con grafos (Elaboración propia)

Lo que hace el recolector de basura para tratar con los ciclos, es que tomar todos los objetos que se tiene en el programa y restarles 1 a sus referencias.

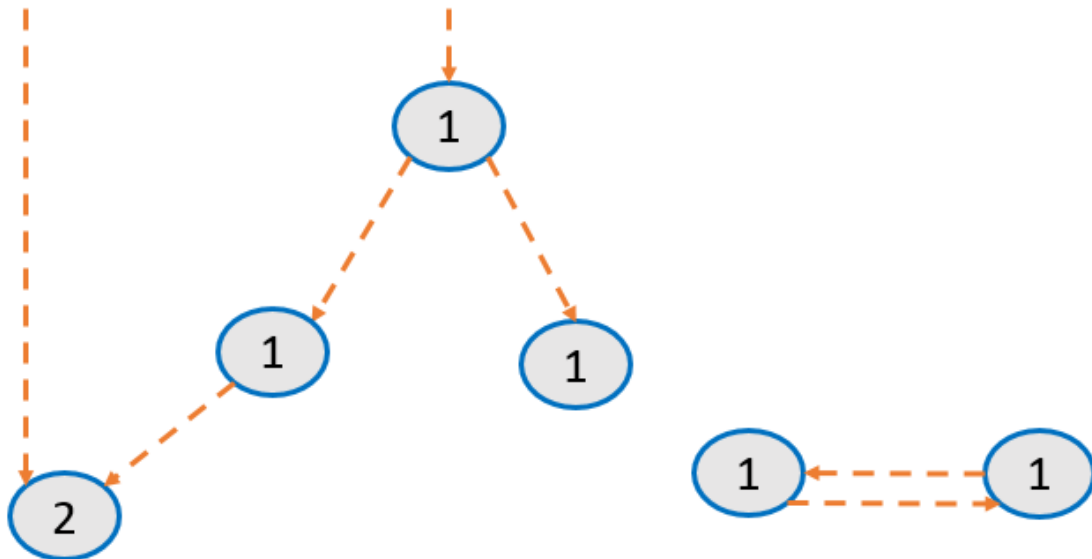


Figura 2. Recorridos por referencias (Elaboración propia)

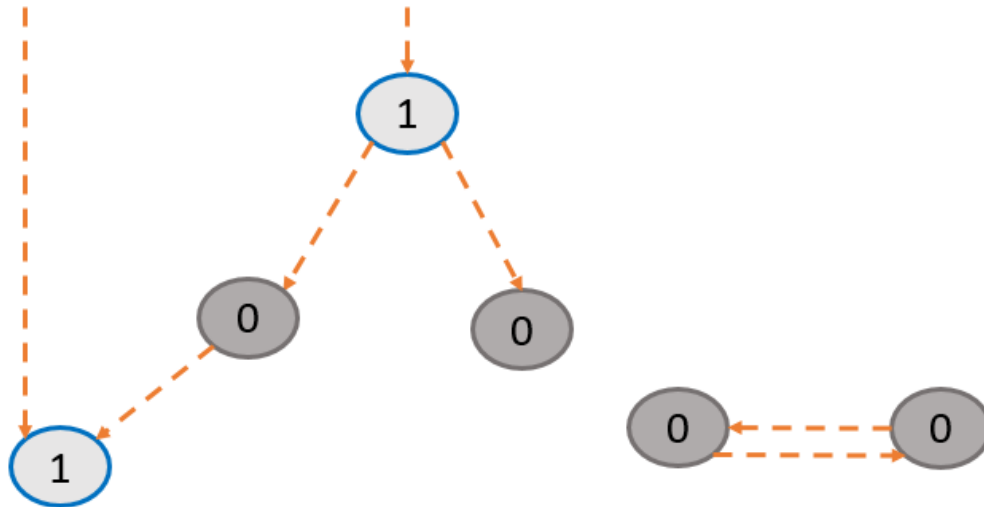


Figura 3. Eliminación de referencias (Elaboración propia)

El recolector de basura de Python tiene la particularidad de que permite dividir estas variables que están referenciadas por otras, en generaciones, donde los primeros elementos creados son generación joven y a medida que van sobreviviendo al recolectar basura, van convirtiéndose en generaciones más viejas.

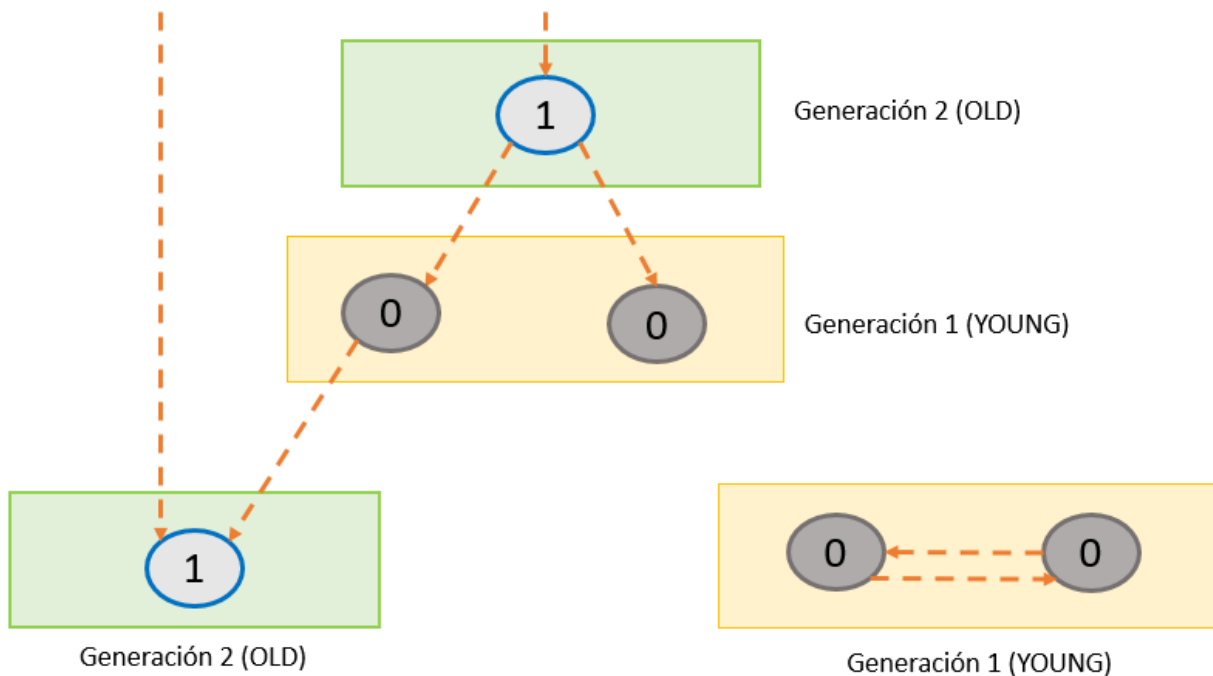


Figura 4. Generaciones (Elaboración propia)

Si bien, la recolección de basura es algo que realiza Python de forma automática, se puede forzar la recolección de basura el método ***gc.collect()***. De esta forma se puede recolectar basura de generaciones específicas.

RMI (Remote Method Invocation)

En RMI un objeto que realiza una llamada, puede invocar a un método alojado en un objeto remoto y todos los detalles de la comunicación son ocultos al usuario.

Este programa, elaborado con Python para la primera parte del examen de distribuidos, consiste en simular el comportamiento del algoritmo recolector de basura distribuida. Para esto se hizo uso de la estructura cliente servidor con RMI.

El programa consta de tres partes, la consulta (en la que el cliente hace las peticiones), el servidor y grafos.

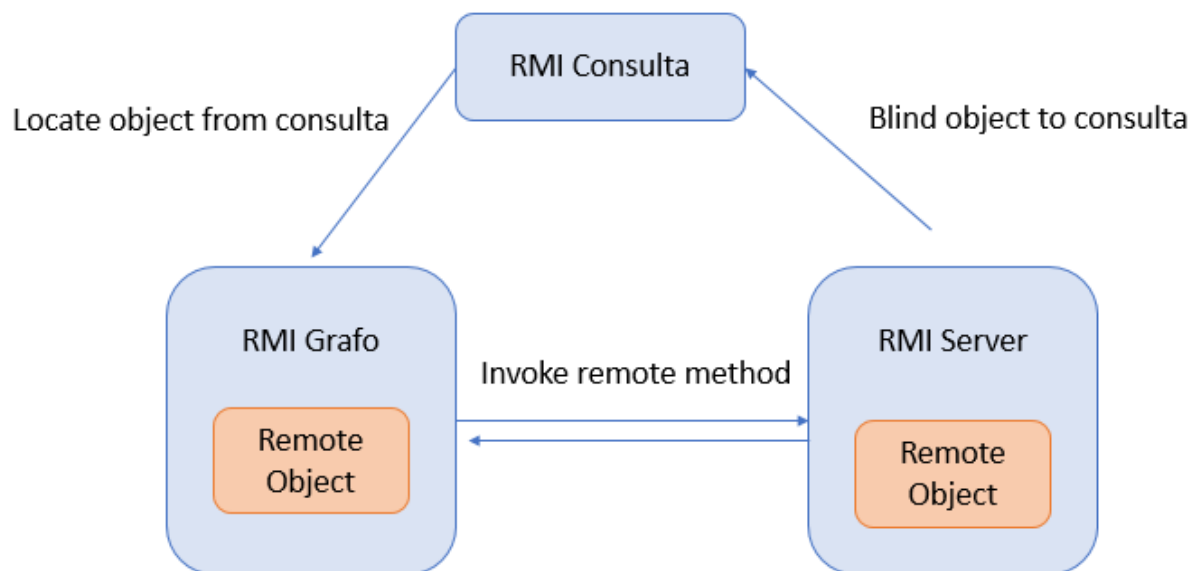


Figura 5. Esquema RMI del programa en Python

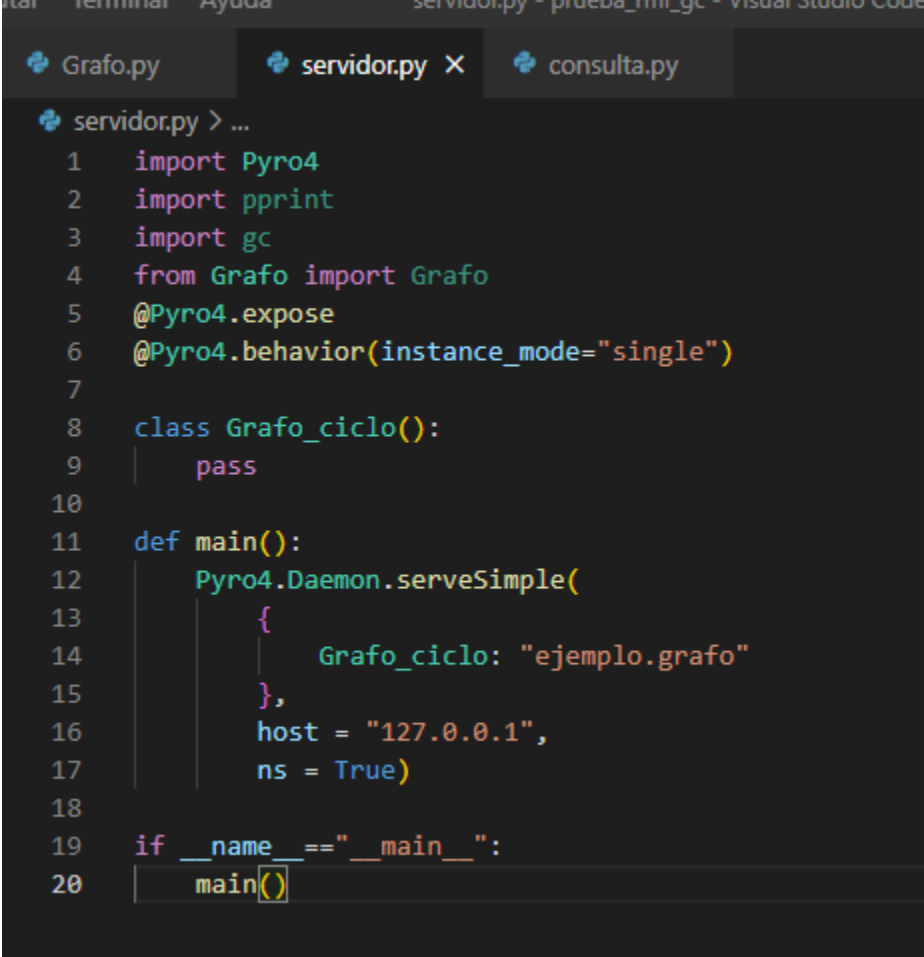
En la parte Grafo.py se encuentra la clase grafo, y tiene las funciones para crear un ciclo de tres grafos y un método para buscar dichas referencias con ayuda de la librería *gc* (colector de basura de Python).

```
Grafo.py  X  servidor.py  consulta.py
Grafo.py > ...
1  import pprint
2  import gc
3
4  class Grafo(object):
5      def __init__(self, nombre):
6          self.nombre = nombre
7          self.siguiente = None
8
9      def set_next(self, next):
10         print('Enlanzando nodos {}'.next = {}'.format(self, next))
11         self.next = next
12
13     def __repr__(self):
14         return '{}({})'.format(
15             self.__class__.__name__, self.nombre)
16
17     #creando ciclo con grafos
18     uno = Grafo('uno')
19     dos = Grafo('dos')
20     tres = Grafo('tres')
21     uno.set_next(dos)
22     dos.set_next(tres)
23     tres.set_next(uno)
24
25     # Eliminar las referencias a los nodos grafo
26     uno = dos = tres = None
27
28     # Mostrar el efecto de la recolección de basura
29     for i in range(2):
30         print('\nColectando {} ...'.format(i))
31         n = gc.collect()
32         print('Objetos inalcanzables:', n)
33         print('Basura restante:', end=' ')
34         pprint.pprint(gc.garbage)
```

Figura 6. Clase Grafo.py

En servidor.py esta nuestro servidor, en el cual se utiliza **Pyro4** (es lo que permite hacer uso de RMI). Con el **@Pyro4.expose** se indica que la clase se anunciará en la red y con **@Pyro4.behavior** se anuncia el comportamiento que tendrá, es decir, los cambios que tendrá el objeto creado de esa clase, podrán ser observados. También se define un **main**, el cual

contendrá el **Daemon** de Pyro4 (es un socket). En el Daemon tenemos la variable *ns* que representa al *name server* y de esta forma no hay necesidad de proporcionar el **uri** a cada aplicación que desea acceder al servidor.



```
servidor.py > ...
1  import Pyro4
2  import pprint
3  import gc
4  from Grafo import Grafo
5  @Pyro4.expose
6  @Pyro4.behavior(instance_mode="single")
7
8  class Grafo_ciclo():
9      pass
10
11 def main():
12     Pyro4.Daemon.serveSimple(
13         {
14             Grafo_ciclo: "ejemplo.grafo"
15         },
16         host = "127.0.0.1",
17         ns = True)
18
19 if __name__ == "__main__":
20     main()
```

Figura 7. Servidor.py

En consulta.py se realiza un *import* a la clase grafos y no a la clase servidor, ya que esta podrá acceder al servidor por medio de la red. También se encuentra el proxy que contiene la cadena “PYRONAME: *ejemplo. grafo*” que le indica que debe buscar en el servidor el “*ejemplo.grafo*”

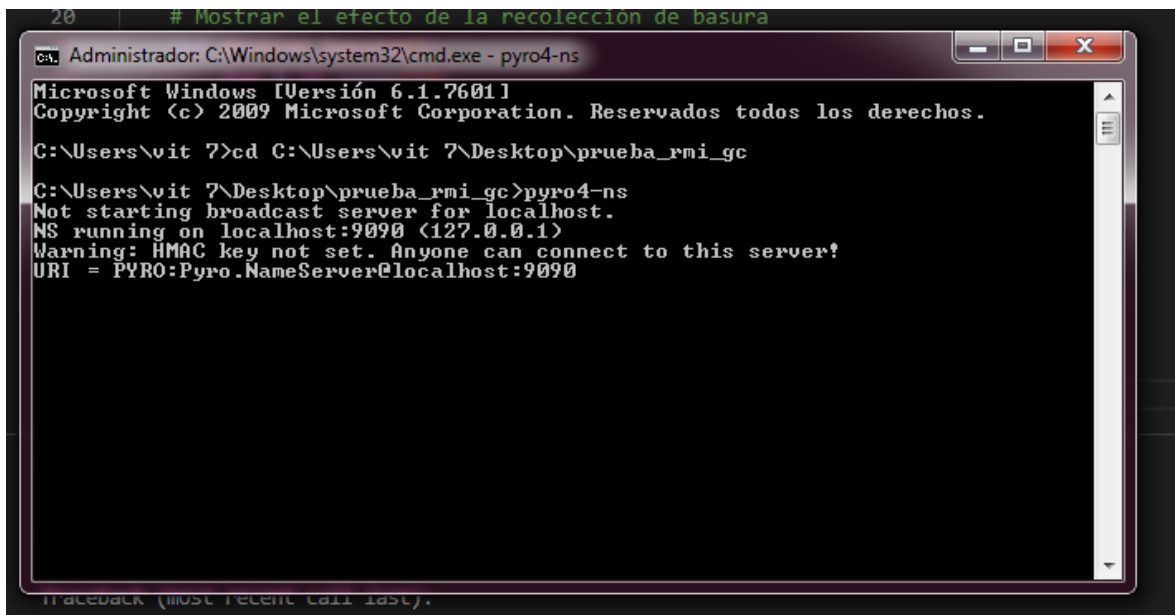
```

consulta.py > ...
1  import Pyro4
2  import sys
3  from Grafo import Grafo
4
5  sys.excepthook = Pyro4.util.excepthook
6
7  ms = Pyro4.Proxy("PYRONAME:ejemplo.grafo")
8  g1 = Grafo("uno")
9  g2 = Grafo("dos")
10 g3 = Grafo("tres")
11
12

```

Figura 8. Consulta.py

Para ejecutar el servidor en *cmd* se debe escribir *pyro4-ns* para que las conexiones sean con el localhost. Como se muestra en la figura 9.



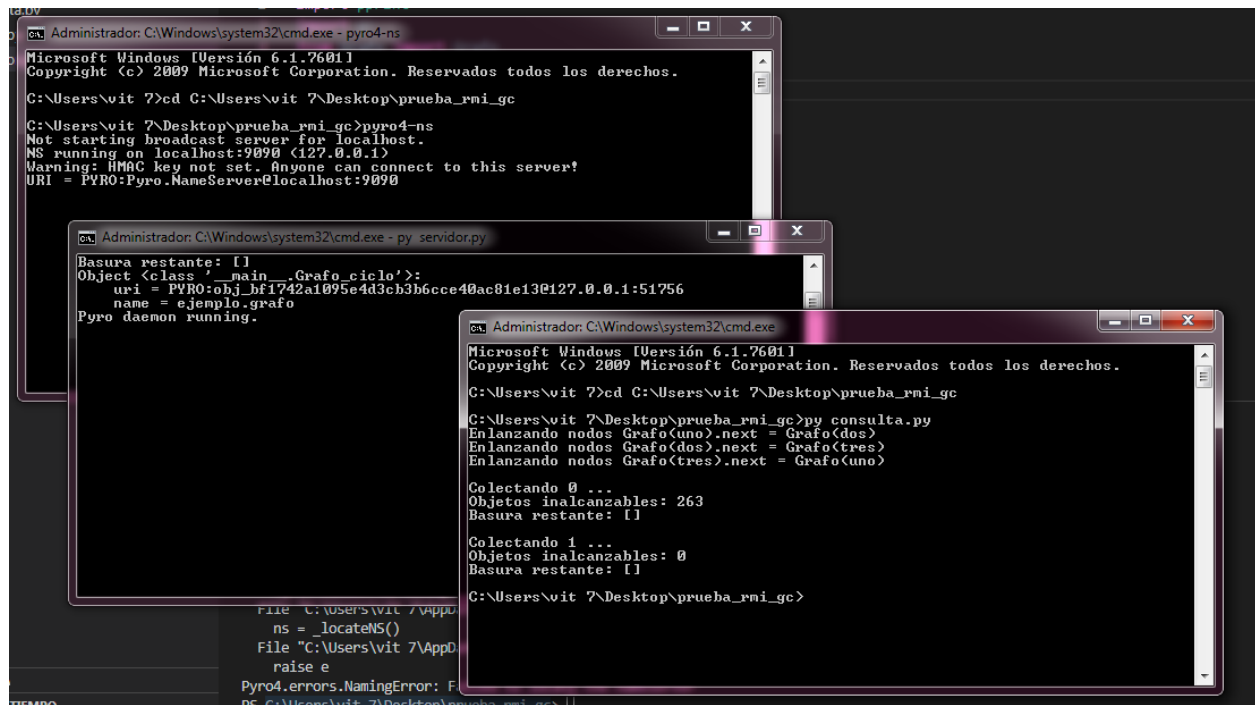
```

28 # Mostrar el efecto de la recolección de basura
C:\Users\vit 7>cd C:\Users\vit 7\Desktop\prueba_rmi_gc
C:\Users\vit 7\Desktop\prueba_rmi_gc>pyro4-ns
Not starting broadcast server for localhost.
NS running on localhost:9090 (127.0.0.1)
Warning: HMAC key not set. Anyone can connect to this server!
URI = PYRO:Pyro.NameServer@localhost:9090

```

Figura 9. Pyro4-ns

Luego de que se establece la conexión, se ingresa al *servidor.py*, al comprobar que el Daemon de *pyro4* funciona, entonces se pueden hacer las consultas.



```
Administrador: C:\Windows\system32\cmd.exe - pyro4-ns
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\vit 7>cd C:\Users\vit 7\Desktop\prueba_rmi_gc

C:\Users\vit 7\Desktop\prueba_rmi_gc>pyro4-ns
Not starting broadcast server for localhost.
NS running on localhost:9090 (127.0.0.1)
Warning: HMAC key not set. Anyone can connect to this server!
URI = PYRO:Pyro.NameServer@localhost:9090

Administrador: C:\Windows\system32\cmd.exe - py servidor.py
Basura restante: []
Object <class 'main.Grafo_ciclo'>:
  uri = PYRO:obj_bf1742a1095e4d3cb3b6c40ac81e130127.0.0.1:51756
  name = ejemplo.grafo
Pyro daemon running.

Administrador: C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\vit 7>cd C:\Users\vit 7\Desktop\prueba_rmi_gc

C:\Users\vit 7\Desktop\prueba_rmi_gc>py consulta.py
Enlazando nodos Grafo<uno>.next = Grafo<dos>
Enlazando nodos Grafo<dos>.next = Grafo<tres>
Enlazando nodos Grafo<tres>.next = Grafo<uno>

Colectando 0 ...
Objetos inalcanzables: 263
Basura restante: []

Colectando 1 ...
Objetos inalcanzables: 0
Basura restante: []

C:\Users\vit 7\Desktop\prueba_rmi_gc>
```

Figura 10. RMI

Funcionamiento de la ampliación con RMI

1. El cliente ingresa el nombre de 3 grafos con los que se crearan el ciclo de grafos referenciados entre sí.
2. Estos datos son enviados a la clase Grafo a través del servidor.
3. En la clase Grafo se convierten en un ciclo de grafos donde cada uno apunta al siguiente.
4. Se eliminan las referencias a los nodos grafos
5. se hace uso de la recolección de basura al buscar las referencias a dichos nodos cíclicos que fueron borrados.
6. Como se estableció un rango (2) de búsqueda, obtenemos en la primera búsqueda una cantidad de referencias de 6 (porque son 3 nodos que se apuntan entre sí, logrando así un total de 6 referencias).
7. En la segunda búsqueda que arroja un total de 0 referencias. Esto porque ya fueron eliminadas las 6 referencias por el colector de basura en la primera búsqueda realizada.

Tal y como se aprecia en la figura 11.

```
27
28 # Mostrar el efecto de la recolección de basura
29 for i in range(2):
30     print('\nColectando {} ...'.format(i))
31     n = gc.collect()
32     print('Objetos inalcanzables:', n)
33     print('Basura restante:', end=' ')
34     pprint.pprint(gc.garbage)
35
36
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN **TERMINAL** JUPYTER

```
PS C:\Users\vlt 7\Desktop\prueba_rmi_gc> & "C:/Users/vlt 7/AppData/Local/Programs/Python/Python38/python.exe" "c:/Users/vlt 7/Desktop/prueba_rmi_gc/Grafo.py"
Enlizando nodos Grafo(unos).next = Grafo(dos)
Enlizando nodos Grafo(dos).next = Grafo(tres)
Enlizando nodos Grafo(tres).next = Grafo(unos)

Colectando 0 ...
Objetos inalcanzables: 6
Basura restante: []

Colectando 1 ...
Objetos inalcanzables: 0
Basura restante: []
PS C:\Users\vlt 7\Desktop\prueba_rmi_gc>
```

Figura 11 Búsqueda nivel 1 y 2

Estas búsquedas se pueden establecer para que se realicen una tercera vez si modificamos el valor del rango establecido en el método que se muestra en la figura 13.

```
25 # Eliminar las referencias a los nodos grafo
26 uno = dos = tres = None
27
28 # Mostrar el efecto de la recolección de basura
29 for i in range(3):
30     print('\nColectando {} ...'.format(i))
31     n = gc.collect()
32     print('Objetos inalcanzables:', n)
33     print('Basura restante:', end=' ')
34     pprint.pprint(gc.garbage)
35
36
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN **TERMINAL** JUPYTER

```
Colectando 1 ...
Objetos inalcanzables: 0
Basura restante: []
PS C:\Users\vlt 7\Desktop\prueba_rmi_gc> & "C:/Users/vlt 7/AppData/Local/Programs/Python/Python38/python.exe" "c:/Users/vlt 7/Desktop/prueba_rmi_gc/Grafo.py"
Enlizando nodos Grafo(unos).next = Grafo(dos)
Enlizando nodos Grafo(dos).next = Grafo(tres)
Enlizando nodos Grafo(tres).next = Grafo(unos)

Colectando 0 ...
Objetos inalcanzables: 6
Basura restante: []

Colectando 1 ...
Objetos inalcanzables: 0
Basura restante: []

Colectando 2 ...
Objetos inalcanzables: 0
Basura restante: []
```

Figura 12. Búsqueda nivel 3