PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2233 - Programación Avanzada (II/2015)

Tarea 2

1. Objetivos

- Aplicar conceptos y nociones de estructuras de datos para la modelación de un problema complejo.
- Aplicar conceptos del paradigma de programación funcional para diseñar algoritmos y optimizar su tiempo de ejecución.

2. Introducción

A pesar de sus intentos de implementar correctamente la gran herramienta "Bummer UC", al momento de poner en marcha el sistema se produjo una división por cero que colapsó la plataforma en su totalidad. Al intentar inscribir el curso con sigla «IIC2233», el sistema no logra agregar el curso a la carga académica de los alumnos, ya que los paquetes de datos entre sus navegadores y la base de datos de "Bummer UC"se están perdiendo en alguna parte. Como los viles lideres actuales se niegan a darle acceso a la red, decide poner en práctica sus vastos conocimientos de Python para hackear el sistema, inspeccionarlo y encontrar dónde se están perdiendo las solicitudes. Rápidamente se da cuenta que se están perdiendo en loops y redireccionamientos infinitos! Es su deber resolver esto, en nombre de todos los aquellos sabios estudiantes que quieren ~~botar~~ tomar «IIC2233».

3. Problema

En esta tarea deberá modelar las distintas conexiones por la que viajan las solicitudes, para así poder estudiar y arreglar la red de forma que se completen exitosamente las inscripciones. Lo anterior debe ser logrado utilizando conceptos y nociones de estructuras de datos y programación funcional.

4. Parte I: Red de conexiones

Al hackear el sistema descubre una librería escrita en Python con los siguientes métodos disponibles:

- `preguntar_puerto_actual()->(int, bool)`: Retorna una tupla donde su primera componente es el identificador correspondiente al puerto en donde esta la solicitud. Su segunda compnente se explica en el siguiente sección.
- `posibles_conexiones()->int` : Retorna la cantidad de conexiones que posee el puerto actual.
- `hacer_conexion(int)->None`: Este comando le permite generar la conexión asociada al camino entregado, por lo que la solicitud se transfiere al nuevo puerto conectado. Los caminos están enumerados desde el 0 hasta cantidad_de_conexiones - 1.

- `puerto_inicio()->int`: Retorna el id del puerto de tu computador.
- `puerto_final()->int`: Retorna el id del puerto de Bummer.

Su primera misión consiste en modelar, armar y entregar la red sobre la cual Bummer UC funciona, a partir de la información que obtiene de la librería anteriormente mencionada. Cada solicitud comienza en un puerto inicial, como su computador, y tiene distintas posibles conexiones y caminos que recorrer. Lo ideal, es lograr que los paquetes de datos que envias lleguen al servidor final, pero como quedó en evidencia en la última toma de ramos, eso parece ser algo complejo.

4.1. Dificultades

Como es de esperarse, el sistema con el que cuenta tiene varios problemas y dificultades que complican su seguimiento. Entre estas se encuentran:

- No sabe cuántos puertos hay.
- Los distintos puertos que se conectan no tienen relación coherente, es decir, cualquier puerto puede conectarse con cualquiera.
- Las conexiones son dirigidas, es decir que la existencia de una conexión desde un puerto A a un puerto B no implica que exista la conexión en el sentido contrario.
- Existen conexiones alternantes, que cambian de objetivo entre dos posibles puertos cada vez que se acceden a ellas.
- Las redes son antiguas y a veces fallan, algunas conexiones carecen de toda lógica y te envían aleatoriamente a uno de entre un grupo de posibles puertos.
- Como los líderes no desean que se metan entre sus redes, han implementado un «robot» que viaja por la red y la conoce a cabalidad. Si se encuentra en el mismo puerto que usted, corta la conexión y se pierde el paquete. En este caso se comienza desde el puerto inicial con un nuevo paquete. Este «robot» hace conexiones en paralelo con usted, es decir, cada vez que usted haga una conexión, él tamibén lo hará. Por lo menos contará con el siguiente metodo para seguirle la pista:
  - `pregunta_puerto_robot()->int/None`: Este comando te entrega el id del puerto donde se encuentra el Robot. El robot tiene un sistema de seguridad que hace que el método no funcione correctamente en dos conexiones consecutivas. Si pregunta cuando no puede, retornará nulo.
  - `pregunta_nodo_actual()->(int, bool)`: la segunda componente de esta tupla es un booleano que identifica si se llego e este puerto acción del robot. (True si te atrapó y te desconectó, False en caso contrario)

4.2. Retorno

Como resultado de esta sección, su programa debe entregar un documento de texto llamado `red.txt` que contenga todos los puertos y conexiones encontradas con la siguiente especificación:

PUERTO ID1
PUERTO ID2
...
PUERTO IDN
CONEXION IDi IDj
CONEXION IDk IDm
...

Como extra, se puede agregar una especificacionn del tipo de conexión que une 2 nodos en caso de ser Alternado o Aleatorio:

...
CONEXION IDi IDj ALT
CONEXION IDk IDm RAND
...

Además, deberá generar el archivo `rutaABummer.txt` que especifique el camino con menor cantidad de conexiones que lo lleve hasta Bummer. Esta debe estar en el mismo formato que `red.txt` pero solo especificando, en orden, las conexiones hechas. Puede asumir que como conexiones normales a las alternadas y aleatorias, es decir, que efectivamente la conexión se realizará si intenta hacer el recorrido.

5. Parte II: Consultas

¡Felicitaciones! Logró descubrir la estructura de la arcaica red de Bummer UC. Ahora se tle ha ocurrido hacer distintas consultas para obtener información de la red e intentar prevenir futuros desastres. La red sigue vulnerable, por lo que no puede asumir que todo sus componentes funcionarán al 100 %.

5.1. Rutas Doble Sentido

Detallar en un archivo `rutasDobleSentido.txt` lo siguiente:

- Encontrar todos los pares de puertos que tengan conexiones bidireccionales, es decir, todos par de puertos A y B tal que existe una conección de A a B y de B a A.
- A partir de lo anterior, determinar todas las rutas doble sentido de mayor tamaño que existen en la red. Es decir, si hay una ruta bidireccional de la forma $A \leftrightarrow B \leftrightarrow C$, usted solo debe detallar esa y obviar las rutas $A \leftrightarrow B$ y $B \leftrightarrow C$, pues estas se desprenden de la primera.

Lo anterior debe seguir el formato:

PAR ID1 ID2
...
RUTA ID1 ID4 ... ID6
...

5.2. Ciclos triangulares y cuadrados.

Como problema inicial, son los ciclos los que ocasionan que las solicitudes de ramos nunca lleguen a Bummer. Por esto es necesario encontrarlos. Para esta consulta nos restringiremos a ciclos del tipo:

- Triangulares: trios de puertos que formen un ciclo.
- Cuadrados: grupos de cuatro puertos que formen un ciclo.

Deben registrarse en un archivo: ciclos.txt en el siguiente formato:

IDj IDk IDm
...
IDa IDb IDc IDd
...

Según el orden del ciclo, es decir: IDj conecta con IDk, que conecta con IDm donde el último conecta con IDj.

5.3. Capacidad de Datos

Los distintos puertos a los que puede acceder tienen cierta capacidad de datos máxima que pueden pasar por ahí. Como en algunos casos uno puede enviar varias solictudes simultaneas, interesa saber lo máximo que puede pesar una solicitud. Aquí debe entregar cuál es la mejor ruta que deben seguir los paquetes para que llegue a el puerto de Bummer la mayor cantidad de información.

Para determinar la capacidad de cada puerto, notas que existe también el comando:

- `get_capacidad()->int`: retorna la capacidad del puerto actual.

Debes entregar un archivo `rutaMaxima.txt` que especifique la ruta encontrada y la capacidad máxima de datos que transporta:

CAP X
ID1 ID2
ID2 ID5
ID5 ID9
...

La consulta anterior puede considerar puertos alternantes y aleatorios. De ser así usted además debe detallar las rutas alternativas a seguir en en el caso de que al conectar aquellos puertos uno termine fuera de la ruta planteada. Separe la ruta principal de las alternativas con una linea con guión:

CAP X
ID1 ID2
...
ID3 ID65
-
CAP Y
ID1 ID3
...

6. Parte III: El Hackeo

Volviendo al tema... a usted le interesa conocer las conexiones para completar su fin, hackear "Bummer UC"de tal manera que los ciclos en las conexiones no ocurran y de esta manera, no congestionar el sistema por solicitudes que se demoran más de lo esperado. Para esta sección se espera que implemente alguna forma de eliminar ciclos de la gran red con el objetivo de que sus solicitudes lleguen a su destino sin entrar en bucles.

6.1. Dificultades

- Entiendase como ciclo cualquier subconjunto de puertos tal que, si parte en el puerto A, existe un camino tal que puede devolverse a A. Por ejemplo, tome los puertos A, B, C; si existe conexión entre A y B, B y C, C y A, entonces hay un ciclo.
- Su algoritmo debe considerar que eliminar conexiones entre puertos es de gran peligro para su finalidad (eliminar conexiones directas entre puertos hace visible sus acciones de hacker). Es por esto que se debiese mantener al mínimo posible las eliminaciones.
- Para no dejar completamente inutilizable la red, en cuyo caso quedaría expuesto su ataque, no pueden existir puertos incomunicados. Esto quiere decir que, dada la red final, todo puerto poseerá al menos una conexión posible.

6.2. Retorno

Para asegurarse de que no empeorará Bummer, será mejor no alterar el sistema directamente y que su programa entregue (según el mismo formato en 4.2) un archivo llamado `noCycle.txt` que especifique los puertos y conexiones finales.

7. Librería

A usted se le hará entrega una librería llamada "sistema.pyc", la puede importar de la misma manera que importa cualquier modulo de python. Esta librería posee todos las funciones especificadas en 4 y 5.

8. Hacker Experto

Como el osado alumno que usted es, quiere llegar aún más allá. No solo debe implementar la estructura para el modelamiento de la red, además deberá implementar toda estructura de datos que vaya a usar, evitando utilizar cualquiera que venga por defecto, es decir: listas, diccionarios, conjuntos, etc... Debe detallar en el readme las especificaciones de cómo implementó cada una de las estructuras utilizadas.

9. Notas

- Su programa debe interactuar con la libreria importándola directamente.
- Su programa debe funcionar en consola de manera libre. Puede pedir input de qué hacer al usuario pero debe poder generar los archivos especificados y cualquier cosa debe ser explicada en el `README.md`.
- Para familiarize con los comandos del sistema, se recomienda que los utilicen manualmente para ver cómo se avanza por las conexiones.
- No intente obtener directamente la red de la librería ya que esta es generada aleatoriamente cada vez que ejecute el programa.
- Se probará su programa con redes de distintos tamaños y naturalezas, por lo que su estructura debiese ser lo más eficiente posible.

10. Restricciones y alcances

- Tu programa debe ser desarrollado en Python 3.4
- Esta tarea es estrictamente individual, y está regida por el Código de Honor de la Escuela: Clickear para Leer.
- Su código debe seguir la guía de estilos `PEP8`
- Si no se encuentra especificado en el enunciado, asuma que el uso de cualquier librería Python está prohibido. Pregunte por foro si se pueden usar librerías específicas.
- El ayudante puede castigar el puntaje[1] de tu tarea, si le parece adecuado. Se recomienda ordenar el código y ser lo más claro y eficiente posible en la creación algoritmos.
- La revisión de la tarea será realizada con distintos archivos `.txt`.

[1] Hasta −5 décimas.

- Debe adjuntar un archivo `README.md` donde comente sus alcances y el funcionamiento de su sistema (*i.e.* manual de usuario) de forma *concisa* y *clara*.
- Cree un módulo para cada conjunto de clases. Divídalas por las relaciones y los tipos que poseen en común.
- Cualquier aspecto no especificado queda a su criterio, siempre que no pase por encima de ningún otro.

11. Entrega

- **Fecha/hora:** 18 de Septiembre - 10:30 AM
- **Lugar:** GIT - Carpeta: Tareas/T02

Tareas que no cumplan con las restricciones señaladas en este enunciado tendrán la calificación mínima (1.0).