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

IIC2233 – Programación Avanzada
Interrogación 2

13 de Noviembre 2015

Duración: 2,5 horas. Sin consultas.

1. (20 pts.) Se desea evaluar un proyecto de infraestructura vial, donde se propone la construcción de un sistema de carreteras que supuestamente va a tener un impacto importante en el flujo del tráfico de la ciudad de Santiago. El estado actual de la ciudad está descrito a través de un grafo no dirigido (todos los caminos son bidireccionales), donde los nodos corresponden a ciertos sectores críticos predefinidos de la ciudad. Dos nodos están conectados en el grafo si existe una secuencia de calles bidireccionales que conectan los dos sectores de la ciudad representados por los nodos. El tiempo estimado de viaje entre dos nodos depende de: la distancia, el número de semáforos y la afluencia de tráfico. Existe una función f_v que dada una hora y día de la semana retorna el número estimado de vehículos en la ciudad.

   Para simular la existencia de semáforos y calles, existe una función f_s que dados dos nodos i y j, retorna dos listas: una con los tiempos entre cambios de colores (rojo-verde) para cada uno de los semáforos entre los nodos i y j, y otra con los tiempos medios (μ's minutos) de viaje entre semáforos (cada tiempo distribuye expovariate($\frac{1}{\mu}$)). Si los nodos i y j no se pueden conectar, la función genera una excepción (debe evitar que se genere).

   Cuando la luz de un semáforo es roja, los vehículos forman una cola para cruzar, esto produce que un vehículo además de esperar que cambie la luz debe esperar que todos los que lo anteceden hayan cruzado. Este tiempo extra de espera distribuye uniforme $[0, \frac{c}{2}]$ minutos, donde c es el número de vehículos que lo anteceden en la cola.

   El grafo está representado por una matriz donde en la posición i, j hay un 1 si los nodos i, j están conectados, 0 en caso contrario (imagine una línea recta con varios semáforos entre dos nodos).

   Implemente, usando **Threads**, el código en Python que genere una simulación de un día de circulación de vehículos en la región. La cantidad de vehículos inicial es 0 (la función f_v debe ser llamada en $t = 0$). Cada vez que la función f_v retorne un número de vehículos mayor al número actual, usted debería generar nuevos vehículos que comiencen a circular, el punto de partida debe ser un nodo seleccionado aleatoriamente. Si la función f_v retorna un número de vehículos menor al que está circulando actualmente, usted deberá retirar vehículos. Para retirar un vehículo usted debe interrumpir su trayectoria, obligándolo a que viaje de vuelta al nodo de partida, una vez que llega al nodo, puede eliminarlo de la circulación. Cada trayectoria debe ser generada aleatoriamente.

   Usted deberá consultar a la función f_v cada una hora, además **debe generar un registro de la cantidad de viajes terminados**, es decir, cuántos autos llegan a su destino original sin tener que devolverse al

origen. Deje explicitados todos los supuestos que utilice, solo serán válidos mientras no violen el enunciado

2. a) (8 pts.) Explique cómo se puede interceptar la carga de datos json (*json.load*) de tal forma de modificar el formato y/o contenido de los datos que se cargaron, escriba un código como ejemplo.

   b) (8 pts.) Explique cómo se puede personalizar la serialización de datos en formato json (*json.dumps*), muestre un código de ejemplo.

   c) (4 pts.) ¿Es posible pasar como argumento a la función *format* una instancia de una clase? Justifique.

   d) (10 pts.) Una señal de audio queda representada por un gráfico de amplitud vs tiempo. Supongamos que se tiene una señal serializada de tal forma que el byte de la posición t representa la amplitud de la señal en el tiempo t, partiendo de izquierda a derecha. El efecto de "eco" de una señal se puede simular sumando a la amplitud de cada instante t una constante c multiplicada por la amplitud en el tiempo $t - w$, donde c y w son parámetros que deben ser definidos. A mayor valor de c se logra un mayor nivel de eco, a mayor valor de w se logra que el eco dure más tiempo. Implemente en Python un algoritmo que dado un bytearray que representa una señal de audio genere una señal nueva que corresponde a la original con un eco agregado.

3. (20 pts.) Se requiere implementar un sistema de chat multiusuario para un grupo de personas en una empresa. Por restricciones en la infraestructura de la red, el servidor solo podrá mantener la comunicación entre un máximo de N usuarios. La principal función del servidor es la de recibir las conexiones entrantes y mantener la comunicación entre los clientes conectados. Para ello debe redireccionar los mensajes emitidos por un cliente a todo el resto de los clientes conectados indicando: el cliente que emitió el mensaje, el mensaje emitido y la hora en que se emitió en mensaje. Por ejemplo:

   ```
   Cliente X dijo: <mensaje>(hora).
   ```

   Asumiendo que ya existe la aplicación cliente, implemente el servidor que sea capaz manejar la comunicación entre los clientes según las especificaciones solicitadas. Además, el servidor debe manejar la situación cuando alguno de los clientes se desconecta o termina su conexión. Haga los supuestos necesarios.

4. (20 pts.) Asuma que usted tiene el frontend para una GUI la que tiene: dos widgets QLineEdit, y un widget QPushButton. El widget QLineEdit1 contiene el path y nombre del archivo de entrada y el widget QLineEdit2 el nombre del archivo de salida. El botón controla la transformación de formatos entre el archivo de entrada y el archivo de salida. Implemente la clase Parser, la cual debe tener los siguientes dos métodos:

   - (15 pts.) parsear(*args): ejecuta el algoritmo de transformación entre los formatos y crea el archivo de salida si y solo si pasa la verificación del parseo. De lo contrario debe imprimir que hubo un error.
   - (5 pts.) verificar_parseo(*args): que contiene el test para verificar la apertura y cierre de tags (<> y </>)

   Su respuesta debe permiter trabajar genéricamente con otros archivos y no solamente con la instancia mostrada a continuación como ejemplo. Los formatos de entrada y salida son los siguientes:

   a) Entrada: formato JSON

```
{
  "catalog": {
    "field": 256,
    "id" : 434, {
      "filename": "00000401.dat",
      "band": "k",
      "measure": [
        {"mjd": 0.0000, "m": 14.152, "e": 0.033},
          ...
        {"mjd": 0.0349, "m": 14.146, "e": 0.048}
        ]
      }
   }
}
```

   b) Salida:

```
<catalog field="256", id="434">
    <filename>"00000401.dat"</filename>
    <band>"k"</band>
    <measure>
        <_measure>
            <mjd>0.0000</mjd> <m>14.152</m> <e>0.033</e>
        </_measure>
            ...
        <_measure>
            <mjd>0.0349</mjd> <m>14.146</m> <e>0.048</e>
        </_measure>
    </measure>
</catalog>
```

HINT GENERAL: El ejemplo a continuación muestra una forma para detener un thread mediante un evento externo a él.

```
# Forma de crear un threard que puede ser detenido
# voluntariamente

class MyThread(threading.Thread):

    def __init__(self, *args):
        super().__init__()
        self.__stop_request = threading.Event()

        <TO-DO>

    def stop(self):
        # Esta property controla el metodo run()
        self.__stop_request.set()

    def run(self):
        while not self.__stop_request.isSet():
            <TO-DO>


#####################################
# MAIN
#####################################

if __name__ == '__main__':

    # Esto es generico
    t = MyThread(*args)
    t.start() # Start capture

    <TO-DO>

    t.stop()
    t.join()
```

Nombre: **Sección:**

Nombre: **Sección:**

Nombre: **Sección:**

Nombre: **Sección:**

Nombre: **Sección:**

Nombre: **Sección:**