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

IIC1103 – Introducción a la Programación
1 - 2017

Interrogación 2

Instrucciones generales:

- La interrogación consta de tres preguntas con igual ponderación.
- Solo se reciben consultas sobre el enunciado en los primeros 30 minutos.
- La última página es un recordatorio; no está permitido utilizar material de apoyo adicional.
- Las tres preguntas están pensadas para ser resueltas en forma sencilla con lo visto en clases. No es necesario utilizar otros mecanismos más avanzados.
- Si el enunciado de la pregunta no lo indica explícitamente, no es necesario validar los ingresos del usuario. Puedes suponer que no hay errores por ingreso de datos.
- Responde cada pregunta en una hoja separada, marcando tu RUT y el número de pregunta en todas las hojas.
- Puedes utilizar cualquier cosa que te hayan pedido programar en un ítem anterior dentro de las preguntas (incluso si no respondiste ese ítem).

Pregunta 1 (1/3)

Tienes los datos históricos de tiempo entre paradas de una ruta de bus, los cuales están almacenados en una lista con la siguiente estructura:

Irarrázaval	4	6	Alameda	0	0	Ñuble	10	12	PUC	10	-1	Quilín	10	9
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Cada tres posiciones están los datos de una parada, los cuales se componen de:

- Un texto con el nombre de la parada.
- El tiempo para llegar hasta esa parada desde la parada anterior en la ruta.
- Un valor entero que indica la posición en la lista de los datos de la siguiente parada en la ruta.

Por ejemplo la parada **Alameda**, en la posición siguiente al nombre tiene el tiempo para llegar hasta esa parada (en este caso es 0, porque aquí comienza la ruta) y en la posición subsiguiente al nombre tiene el índice de la lista donde está la siguiente parada de la ruta (en este caso el índice es 0, lo que indica que la siguiente parada es **Irarrázaval**). Cuando una parada tiene el valor -1 como siguiente, significa que la parada es el final de la ruta. En el ejemplo, la ruta termina en la parada **PUC**.

a) **(20 puntos)** Implementa la función `obtener_itinerario`, la que recibe la ruta y el nombre de una parada de origen y retorna una lista con las paradas que faltan para llegar hasta el final de la ruta (debe incluir el origen). Por ejemplo, si `ruta=['Irarrázaval',4,6,'Alameda',0,0,'Ñuble',10,12,'PUC',10,-1,'Quilin',10,9]`, `obtener_itinerario(ruta,'Irarrázaval')` retorna `['Irarrázaval', 'Ñuble', 'Quilin', 'PUC']`

b) **(30 puntos)** Implementa la función `calcular_tiempo_de_viaje`, la cual recibe una ruta y debe retornar el tiempo total de viaje, partiendo de la parada inicial y terminando en la parada final. Con la misma ruta anterior, `calcular_tiempo_de_viaje`(ruta) retorna 34.

c) **(10 puntos)** Implementa la función `encontrar_ruta_mas_corta`, la que recibe como parámetro una lista de rutas y debe buscar y retornar la ruta más corta en tiempo que hay en la lista de rutas. Por ejemplo, si

```
rutas=[ ['Irarrázaval',4,6,'Alameda',0,0,'Ñuble',10,12,'PUC',10,-1,'Quilín',10,9],
['Matta',8,3,'Ñuble',10,6,'PUC',15,-1,'Alameda',0,0] ]
```

`encontrar_ruta_mas_corta(rutas)` retorna `['Matta',8,3,'Ñuble',10,6,'PUC',15,-1,'Alameda',0,0]`

Pregunta 2 (1/3)

Una compañía de telefonía móvil te pide que hagas un programa para crear un archivo (**"saldos.txt"**) con los saldos de segundos de llamadas y kilobytes de descarga que le quedan a cada uno de sus clientes, de acuerdo al plan que tienen contratado. Para ello, debes leer la información de clientes (**"clientes.txt"**), planes (**"planes.txt"**), y consumos ya efectuados (**"consumos.txt"**).

A continuación, se muestra un ejemplo simple:

clientes.txt

```
Hugo;Normal
Paco;Super
Luis;Platino
```

planes.txt

```
Normal;120;10
Super;240;40
Platino;360;100
```

consumos.txt

```
Hugo;llamada;24
Paco;llamada;12
Hugo;descarga;800000
Paco;descarga;800000
Luis;descarga;800000
Luis;llamada;14
Luis;llamada;21
Paco;llamada;24
Hugo;llamada;12
Luis;llamada;18
Hugo;descarga;500000
Paco;llamada;18
Luis;descarga;300000
Hugo;llamada;14
Paco;descarga;300000
Paco;descarga;500000
Luis;descarga;500000
Hugo;descarga;300000
```

saldos.txt

```
Hugo;7150;8400000
Paco;14346;38400000
Luis;21547;98400000
```

Notar que la información de los planes está en minutos y gigabytes (GB), en tanto que la información de los consumos y saldos está en segundos y kilobytes (KB). Por simplicidad, puedes asumir que 1 GB = 1000000 KB. Puedes también asumir que el separador entre los datos es un ";".

Por ejemplo, **Hugo** ha hecho 3 llamadas por un total de 50 segundos (**24** + **12** + **14**), y por estar en un plan **Normal** tiene 7200 segundos (**120** * 60). Por lo tanto, su saldo es **7150** segundos. A su vez, ha hecho 3 descargas, totalizando 1600000 KB (**800000** + **500000** + **300000**). Dado que su plan **Normal** considera **10** GB, su saldo es de **8400000** KB (**10** * 1000000 – 1600000). Se destaca en "negritas" aquellos datos que se leen desde los archivos o que se escriben al archivo **"saldos.txt"**.

Pregunta 3 (1/3)

El Servicio de Aduanas está preocupado porque algunas empresas importan una cierta cantidad de productos, pero solo declaran una parte de ellos (para pagar menos impuestos). Quieren detectar este fraude, comparando el número de productos que las empresas importaron contra el número que vendieron (pues si venden más de lo que importan, es un fuerte indicador de que están cometiendo fraude).

Cada empresa puede vender sus productos en varios mercados (por ejemplo, una importadora de plátanos, los vende en La Vega y en Lo Valledor), por lo que se creará un sistema que monitoree las ventas de las empresas en varios mercados.

Para resolver este problema, debes implementar las siguientes clases y métodos. Tu código debe ser compatible con el código principal proporcionado al final de este problema.

a) Crea la clase **Mercado** con los siguientes métodos:

(I) **(1 punto) __init__(self, m_id, nombre)**: crea un Mercado. Los strings *m_id* y *nombre* representan el identificador y el nombre del mercado.

(II) **(3 puntos) declarar_venta(self, e_id, p_id, c)**: registra una venta realizada en el mercado. Los strings *e_id* y *p_id* representan los identificadores de la empresa y el producto vendido, y el entero *c* la cantidad vendida. No se retorna nada.

(III) **(3 puntos) numero_ventas(self)**: retorna un entero indicando el número de ventas declaradas en el mercado.

(IV) **(3 puntos) __str__(self)**: retorna un string que representa al Mercado, en el siguiente formato: *m_id:Nombre(NumVentas)* Por ejemplo, `'m1:La Vega(4)'`.

(V) **(8 puntos) obtener_empresas(self)**: retorna una lista de strings con los identificadores de las empresas que han vendido algo en el mercado. La lista no debe contener repetidos.

(VI) **(8 puntos) obtener_total_vendido(self, e_id, p_id)**: dados los strings *e_id* y *p_id*, identificadores de una empresa y un producto, retorna un entero con la cantidad total vendida del producto *p_id* por la empresa *e_id* en el mercado.

b) Crea la clase **SistemaMonitor** con los siguientes métodos:

(I) **(1 punto) __init__(self)**: crea un SistemaMonitor, para poder monitorear a los mercados registrados en el sistema.

(II) **(3 puntos) registrar_mercado(self, m)**: dado un objeto *m* de tipo Mercado, el sistema lo registra (agrega) como mercado que está siendo monitoreado. Cada Mercado será registrado una sola vez. No se retorna nada.

(III) **(3 puntos) numero_mercados(self)**: retorna un entero indicando cuántos mercados están siendo monitoreados por el sistema.

(IV) **(3 puntos) existe_mercado(self, m_id)**: retorna True si el sistema está monitoreando el mercado con el identificador *m_id*, False si no.

(V) **(8 puntos) declarar_venta(self, m_id, e_id, p_id, c)** Dados *m_id, e_id, p_id* (strings identificadores de: un mercado, una empresa, y un producto), y el entero *c*, el sistema intenta declarar la venta en el mercado *m_id* por parte de la empresa *e_id* del producto *p_id* en una cantidad *c*. En el caso de que el sistema esté monitoreando el mercado *m_id*, el sistema debe registrar la venta en ese mercado y retornar True, para indicar que la venta se registró correctamente. En el caso de que el sistema no esté monitoreando el mercado *m_id*, el método solamente debe retornar False.

(VI) **(16 puntos) comprobar_fraude(self, sospechosos)** Se considera fraude si la cantidad de productos vendidos por una empresa es mayor a la cantidad de productos importados por la misma empresa. El método recibe una lista de *sospechosos*, donde cada sospechoso es una lista de formato `[e_id, p_id, c]` indicando la cantidad *c* del producto *p_id* que declara haber importado la empresa *e_id*. Por ejemplo, `sospechosos = [ ['e33','p08014', 3000], ['e66', 'p08014', 3000], ['e66', 'p08080', 2000] ]`.

Se debe retornar una lista de listas, de la forma `[ [e_id, p_id, dif], ....]` donde *e_id* y *p_id* son identificadores de la empresa y producto donde **efectivamente se ha cometido fraude**, y *dif* es un entero que representa la diferencia entre las importaciones declaradas y las ventas reales. Por ejemplo, el resultado para la lista anterior podría ser: `[ ['e33', 'p08014', 2000], ['e66', 'p08080', 1000] ]`

El método no debe incluir pares empresa-producto donde NO se ha cometido fraude, y en el caso de no existir ningún fraude, la lista retornada debe ser vacía.

```
#CODIGO PRINCIPAL DE EJEMPLO
#Crear los mercados
mer1 = Mercado("m1", "La Vega")
mer2 = Mercado("m2", "Lo Valledor")
```

```
mer3 = Mercado("m3", "Central")

#Crear sistema de monitoreo
s = SistemaMonitor()
print(s.numero_mercados()) #Imprime 0

#Registrar los mercados
s.registrar_mercado(mer1)
s.registrar_mercado(mer2)
s.registrar_mercado(mer3)
print(s.numero_mercados()) #Imprime 3

print(s.existe_mercado("m1")) #Imprime True
print(s.existe_mercado("m9")) #Imprime False

#Declarar algunas ventas
s.declarar_venta("m1", "e33", "p08014", 1000)
s.declarar_venta("m1", "e33", "p08014", 1000)
s.declarar_venta("m2", "e33", "p08014", 2000)
s.declarar_venta("m3", "e33", "p08014", 1000)
s.declarar_venta("m3", "e33", "p08080", 2000)
s.declarar_venta("m1", "e66", "p08014", 3000)
s.declarar_venta("m3", "e66", "p08080", 3000)
s.declarar_venta("m1", "e99", "p08014", 3000)
r1 = s.declarar_venta("m3", "e11", "p08221", 92)
print(r1) #Imprime True
r2 = s.declarar_venta("m9", "e11", "p08221", 33)
print(r2) #Imprime False porque m9 no está registrado

#Comprobar fraude
sos = [["e33", "p08014", 3000],
       ["e66", "p08014", 3000],
       ["e66", "p08080", 2000]]
fraudes = s.comprobar_fraude(sos)
print(fraudes)
#Imprime [['e33','p08014',2000],['e66','p08080',1000]

#Codigo test mercado
print(mer1.numero_ventas()) #Imprime 4
print(mer1) #Imprime m1:La Vega(4)
print(mer1.obtener_empresas()) #Imprime ['e33','e66','e99']
print(mer1.obtener_total_vendido("e33","p08014")) #Imprime 2000
```