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

IIC1103 – Introducción a la Programación
2018 - 2

Interrogación 2

Instrucciones generales:

- Para resolver la interrogación solo necesitas **hasta la sección 12** del Recordatorio.
- La interrogación consta de **3 preguntas con igual ponderación**.
- Solo se reciben **consultas** sobre el enunciado en los primeros 30 minutos.
- No está permitido utilizar **material de apoyo adicional** al recordatorio adjunto.
- Todas las 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**, anotando tu **número de alumno** y el **número de pregunta** en todas las hojas. Si usas más de una hoja en una pregunta, **enumera las hojas** de esa pregunta.
- Puedes utilizar cualquier cosa que te hayan pedido programar en un **ítem anterior** dentro de las preguntas (incluso si no respondiste ese ítem).
- Tienes **dos horas y media** para resolver la prueba desde que se entregan los enunciados.

Pregunta 1

Estás realizando tu práctica en `GeoTree`, un emprendimiento UC que usa Python para armar árboles genealógicos. Sin embargo, están teniendo problemas en la implementación, por lo que te piden a ti, flamante alumno de Introducción a la Programación, que los ayudes a terminar su programa.

Hasta ahora, lo único que tienen es una representación con *listas* del árbol de una familia, y la declaración de las funciones que vas a necesitar. El árbol está representado por una lista donde, en cada elemento se guarda un *string* con el nombre del miembro de la familia (no hay *strings* repetidos), seguido de una *lista* de las posiciones de sus hijos dentro de la lista. Puedes ver un ejemplo de la lista en el código más abajo. Las funciones que necesitan que implementes son las siguientes:

(a) **(30 puntos)** `hijos(a,n)`: recibe una *lista* `a` con el árbol genealógico y un *string* `n` con el nombre de un integrante de la familia. Retorna una *lista* de *strings* con los nombres de los hijos de ese integrante. Si el integrante no tiene hijos, retorna una *lista* vacía.

(b) **(30 puntos)** `padres(a,n)`: recibe una *lista* `a` con el árbol genealógico y un *string* `n` con el nombre de un integrante de la familia. Retorna una *lista* de *strings* con los nombres de los padres de ese integrante. Si el integrante no tiene padre, retorna una *lista* vacía.

A continuación, se encuentra el código con el ejemplo de la familia Simpson y el *output* esperado escrito en los comentarios. Tu deber es completar las funciones `hijos(a,n)` y `padres(a,n)`, para que los `print` del final del código impriman lo solicitado. Ten en cuenta que no es necesario que la lista de los nombres salgan exactamente en el orden del *output* esperado.

```
def hijos(a, n):
    #Completar

def padres(a, n):
    #Completar

simp = [
    ["Abraham",[1,2]],    #0
    ["Herb",[]],          #1
    ["Homer",[3,4,5]],    #2
    ["Bart",[]],          #3
    ["Maggie",[]],        #4
    ["Lisa",[]],          #5
    ["Marge",[3,4,5]],    #6
    ["Mona",[2]],         #7
    ["Clancy",[6,10,11]], #8
    ["Jackie",[6,10,11]], #9
    ["Selma",[12]],       #10
    ["Patty",[]],         #11
    ["Ling",[]]           #12
    ]

print(hijos(simp,'Marge'))       #['Bart', 'Maggie', 'Lisa']
print(hijos(simp,'Clancy'))      #['Marge', 'Patty', 'Selma']
print(hijos(simp, 'Maggie'))     #[]
print(padres(simp, 'Maggie'))    #['Homer', 'Marge']
print(padres(simp, 'Homer'))     #['Abraham', 'Mona']
print(padres(simp, 'Ling'))      #['Selma']
print(padres(simp, 'Abraham'))   #[]
```

Para que entiendas mejor la estructura de la lista, la Figura 1 muestra el árbol genealógico de los Simpson:

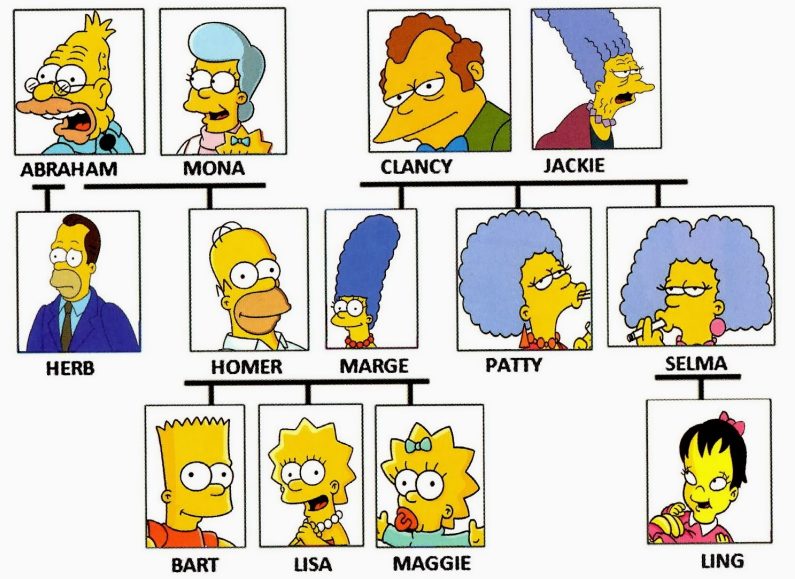

Figura 1: Árbol genealógico de Los Simpson

Pregunta 2

El DCC ha ofrecido un gran premio al profesor más popular del curso IIC1103. Para determinar al ganador, los profesores del curso han decidido realizar una votación en cada sección, para que cada alumno vote por su profesor favorito (pueden votar por el profesor de cualquier sección). Luego de que todos votaron, notaron que varios alumnos se equivocaron: **votaron dos veces en la misma sección** y/o **votaron en más de una sección**.

Las votaciones de los alumnos se encuentran en `N` archivos de nombre `votaciones_sX.txt` (`votaciones_s1.txt`, `votaciones_s2.txt`, ..., `votaciones_sN.txt`). Cada uno de estos archivos tiene el siguiente formato en cada línea: `num_alumno,profesor`, donde `num_alumno` es un entero con el número de alumno, y `profesor` es un *string* de una lista de `N` elementos llamada `lista_profesores` que puedes utilizar en tu programa. Puedes suponer que cada archivo `votaciones_sX.txt` cuenta con al menos la votación de todos los alumnos de la sección `X`.

Además, cuentas con el archivo `alumnos.txt` que tiene el siguiente formato en cada línea: `num_alumno,seccion`, donde `num_alumno` es un entero con el número de alumno, y `seccion` es un entero desde 1 a `N` (inclusive). No puedes asumir que los archivos se encuentran ordenados (por número de alumno o sección). A continuación se muestra un ejemplo con `N=3` secciones donde `lista_profesores = ["Marquinez", "Sepulveda", "Lopez"]`.

```
1,Marquinez
2,Marquinez
3,Sepulveda
4,Marquinez
5,Marquinez
```

(a) votaciones_s1.txt

```
5,Marquinez
6,Sepulveda
7,Lopez
8,Sepulveda
8,Sepulveda
```

(b) votaciones_s2.txt

```
8,Lopez
9,Sepulveda
10,Lopez
11,Lopez
12,Lopez
```

(c) votaciones_s3.txt

```
1,1
2,1
3,1
4,1
5,2
6,2
7,2
8,2
9,3
10,3
11,3
12,3
```

(d) alumnos.txt

Para este ejemplo, vemos que hubo 5 votos en cada sección. Sin embargo, existen los siguientes errores:

- El alumno `5` no pertenece a la sección 1.
- El alumno `8` votó dos veces en la sección 2.
- El alumno `8` no pertenece a la sección 3.

De esta forma, los votos válidos serían los siguientes:

s1	s2	s3
1,Marquinez	5,Marquinez	9,Sepulveda
2,Marquinez	6,Sepulveda	10,Lopez
3,Sepulveda	7,Lopez	11,Lopez
4,Marquinez	8,Sepulveda	12,Lopez

(e) votos válidos

Tu objetivo es consolidar los votos de todos los alumnos, corrigiendo los errores que pueden existir (considera que en caso que un alumno vota dos veces, puedes tomar cualquiera de los dos votos). Además, dado que el DCC quiere reutilizar el programa que escribas para otros semestres, debes considerar que este funcione para cualquier número de secciones (donde `N` corresponde al número de secciones). Deberás escribir un nuevo archivo llamado `resultados.txt` en que cada línea debe tener el formato: `profesor,nvotos` donde `profesor` es un *string* de `lista_profesores` y `nvotos` es la cantidad de votos que obtuvo cada profesor. El resultado del ejemplo anterior se muestra a continuación:

```
Marquinez,4
Sepulveda,4
Lopez,4
```

(f) resultados.txt

Pregunta 3

¡Bienvenido a la Feria de Investigación del DCC! El siguiente código usa programación orientada a objetos, incluyendo las clases de objetos `Stand` y `Feria` para gestionar la asignación de stands durante la feria. En concreto, primero se crea la feria con `ns` stands vacíos. A continuación, se escoge entre tres opciones:

1. **Inscribir un stand**. Si quedan stands vacíos, se solicita el título del stand y el número de participantes, de lo contrario, se imprime que no quedan stands vacíos.
2. **Información**. Dado un número de stand, en caso de estar vacío, se indica que está vacío. En caso contrario, se indica el título del stand y el número de participantes.
3. **Cerrar**. Cierra la feria.

```
ns = int(input("Número de stands en la Feria: "))
f = Feria(ns)

op = int(input("1-Inscribir 2-Info 3-Cerrar: "))
while op != 3:

    if op == 1:
        if f.quedan_stands_vacios():
            t = input("Título del stand: ")
            p = int(input("Número de participantes del stand: "))
            f.inscribir(t,p)
        else:
            print("No quedan stands vacíos")

    elif op == 2:
        n = int(input("Número de stand: "))
        s = f.obtener_stand(n) #s de tipo Stand
        if s.vacio():
            print("Stand vacío")
        else:
            print("Título =>", s.titulo())
            print("Participantes =>", s.participantes())

    op = int(input("1-Inscribir 2-Info 3-Cerrar: "))

print("Feria Cerrada")
print("Quedaron", f.numero_stands_vacios(), "stands por llenar")
```

Se te pide que implementes las clases `Stand` y `Feria` con los siguientes métodos con el fin de **que sea compatible con el código principal de ejemplo**:

1) `Stand`:

- **(6 puntos)** `__init__`: No recibe parámetros. Inicializa un stand vacío con sus atributos que corresponda.
- **(3 puntos)** `titulo`: No recibe parámetros. Retorna un *string* con el título del stand.
- **(3 puntos)** `participantes`: No recibe parámetros. Retorna un *int* con la cantidad de participantes del stand.
- **(3 puntos)** `vacio`: No recibe parámetros. Retorna `True` si el stand está vacío; y `False`, en caso contrario.

2) `Feria`:

- **(10 puntos)** `__init__`: Recibe como parámetro un *int* con el número de stands vacíos (de la clase `Stand`) que tiene la feria al crearse. Se inicializa la feria con los stands vacíos creados. Los stands están ordenados, y se pueden identificar con un número que va de `0` al número de stands -1.

- **(15 puntos)** `inscribir`: Recibe un *string* con el título del stand y un *int* con la cantidad de participantes de ese stand (debe ser mayor a 0 y puedes asumir que el input para este atributo siempre cumplirá dicha restricción). Se inscribe el primer stand vacío con dichos parámetros. Puedes suponer que al inscribirse, ya se verificó antes si hay stands vacíos. No retorna nada.
- **(5 puntos)** `obtener_stand`: Recibe un *int* con el índice del stand (va de 0 al número de stands -1) Retorna un objeto de tipo `Stand` con el stand solicitado.
- **(10 puntos)** `numero_stands_vacios`: No recibe parámetros. Retorna un *int* con el número de stands vacíos de la feria.
- **(5 puntos)** `quedan_stands_vacios`: No recibe parámetros. Retorna `True` si quedan stands vacíos o `False` en caso contrario. **Este método debe usar el método** `numero_stands_vacios` **anterior.**