UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA - UTEC

Tarea 4

Fecha de inicio: *22 de Febrero, 2022*
Fecha de entrega: *28 de Febrero, 2022*

Curso: *Programación 1 (1111)* – Entrega mediante: *Gradescope*

Indicaciones generales

1. Recuerda que la tarea es **individual**. Los casos de copia/plagio serán sancionados con nota cero (0) en la asignatura.

2. (a) La fecha límite de entrega es el **Lunes 28 de Febrero a las 23:59 hrs**.
   (b) Es altamente recomendable no esperar hasta la última hora.
   (c) *Gradescope* desactivará automáticamente los envíos pasada dicha hora límite.
   (d) **No se aceptarán entregas atrasadas ni entregadas por otros medios.**

3. Revisa bien lo que entregas, aunque en esta oportunidad podrás entregar ilimitadas veces la tarea, la última enviada será la evaluada.

4. Recuerda que *Gradescope* corrige automáticamente tu entrega. Dicha plataforma mostrará si has realizado correctamente las pruebas y mostrará algunos mensajes en color verde. Puedes ver un ejemplo de este caso en el anexo 2.

5. Es posible que hayas subido tu entrega pero hayas modificado algo que no se debió en el template. En ese escenario, Gradescope te mostrará algunos mensajes de error. Puedes ver un ejemplo de esto en el anexo 3.

Gradescope

1. Nosotros les proporcionaremos un código base de donde deberán partir para completar dicho ejercicio. Este archivo es llamado `solution.py` y lo encontrarán en la indicación de la tarea en CANVAS.

2. Al finalizar, **solo** subir el archivo `solution.py` (NO cambiar el nombre del archivo y NO comprimirlo).

3. Cada pregunta tiene diversos casos de prueba. Para obtener la nota completa en una pregunta, el algoritmo debe obtener la respuesta correcta en dichos casos de prueba.

4. Si un caso de prueba falla, visualizarán un mensaje de error con sugerencias. **Lee el error**, revisa el código e inténtalo de nuevo.

5. Los input de los casos de prueba son confidenciales.

Indicaciones específicas

1. En el anexo 1 podrán visualizar la plantilla de código que deberán de seguir para resolver los ejercicios.

2. Ustedes deben escribir dentro de la sección y a la misma altura de donde esta escrito *"SU SOLUCION EMPIEZA AQUI"*. Además, no deben modificar nada debajo de *"SU SOLUCION TERMINA AQUI"*. Recuerden tener cuidado con las indentaciones.

3. Los input del ejercicio se encuentran en la plantilla. Recuerden usar estas variables para resolver el ejercicio.

1. Introducción

Los árboles genealógicos son típicamente utilizados para representar las relaciones parentales entre los miembros de una familia, de forma gráfica. Con esta representación es sencillo ordenar a los miembros de una familia por generaciones, siendo cada fila horizontal en el árbol una generación distinta empezando desde la cima del árbol con la primera generación. A continuación se muestra un ejemplo de un árbol genealógico:

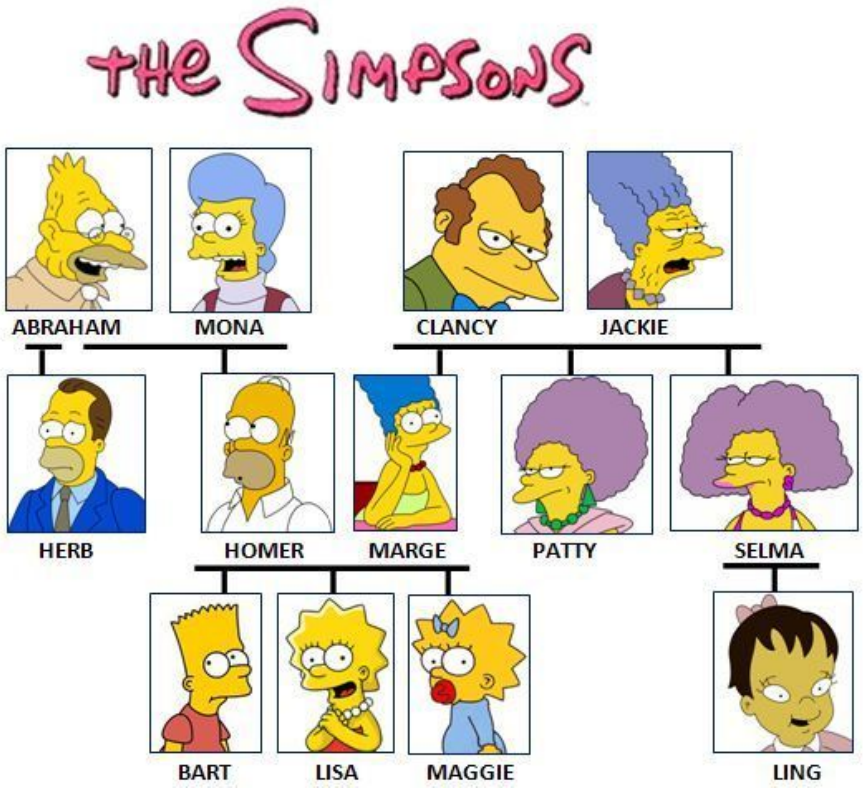

Figure 1: Árbol Genealógico

Computacionalmente, un árbol genealógico no se puede almacenar en su forma gráfica y mantener su fácil acceso a la vez, por lo que es necesario encontrar una forma lineal de representar sus datos. Una solución es almacenar las personas del árbol en una tabla de datos interconectados, donde cada fila hace referencia a otra fila en la tabla. Mientras que visualizar y comprender el árbol de esta manera es más difícil para un humano, una computadora podrá manejar y procesar sus datos de una manera mucho más eficiente. La computadora podrá recorrer velozmente cada fila de la tabla de forma iterativa para acceder a los datos necesarios, y hacer las relaciones correspondientes al momento de ejecución. Otra representación computacional de los árboles genealógicos puede ser los diccionarios en Python, los cuales tienen sus propias ventajas para la computadora, permitiendo acceder a los datos de una persona sin necesidad de recorrer el set de data entero.

En esta tarea se le pide utilizar sus conocimientos de Python para realizar programas que lean y procesen arboles genealógicos, entendiendo su almacenamiento en archivos y diccionarios.

2. Resolver la tarea

2.1. Data. La data de la tarea es un árbol genealógico almacenado en una tabla, que deberá ser leída a partir de un archivo CSV, donde cada fila representa a una persona.

Estructura de un archivo:

id	first_name	last_name	country	mom	dad	bestie
A01	Veronika	Mc Corley	Italy			C01
A02	Sebastian	Passion	Italy			C03
A03	Adam	McCoish	Peru			A02
B01	Chester	McCoish	Peru	A02	A03	A01
B02	Tate	Mc Corley	Peru	A03	A01	C03
B03	Donn	Passion	Spain	A01	A02	C01
C01	Aile	Mc Corley	Spain	B03	B02	C02
C02	Mamie	Passion	Italy	B02	B03	A03
C03	Janek	Passion	Spain	B02	B03	A02

2.2. Lectura del archivo. El archivo 'data.csv' es leído en la plantilla y almacena la data entera en un diccionario, donde la infromación de cada persona es accesible en base a su id. Este diccionario tiene la siguiente estructura:

```
{
    "A01": {
        "first_name": "Veronika",
        "last_name": "Mc Corley",
        "country": "Italy",
        "mom": "",
        "dad": "",
        "bestie": "C01"
    },
    "A02": {
        ...
    },
    ...
```

```
}
```

2.3. Personas por países - 8 puntos. El objetivo de este ejercicio es retornar un diccionario, donde las claves (keys) son los países que aparecen en la data, y los valores (values) la cantidad de personas que viven en el país respectivo (entero).
La solución de esta pregunta se debe ser implementada en la función *ejercicio1()*.

Ejemplos

Para los siguientes ejemplos se usa la tabla de datos de ejemplo en la sección 4.1

Ejemplo1:

```
>>> Solution().ejercicio2()
{"Italy": 3, "Peru":3, "Spain":3}
```

Explicación: Hay 3 personas en cada país.

2.4. Encuentra las conexiones - 12 puntos. En este ejercicio deberás encontrar las conexiones de una persona. Estas pueden ser parientes o amigas. Para esto deberás implementar tu solución en la función `ejercicio2()`:

Donde:

- INPUT: La función `ejercicio2` recibe dos parámetros:
  - `id`: en un string con el identificador de la persona de la cual queremos encontrar sus conexiones.
  - `camino`: es un string con la secuencia de caracteres que nos llevarán a la persona que queremos encontrar. Los únicos posibles valores de los caracteres en camino son:
    - `M`: Representa a la mamá (Mom).
    - `D`: Representa al papá (Dad).
    - `B`: Representa al mejor amigo (Bestie).
- OUTPUT: Esta función retornará dos valores:
  - `conexiones`: es una lista con los identificadores de los antepasados.
  - `persona_buscada`: es un diccionario con los datos de la persona a la que llegamos luego de seguir la secuencia de caracteres definida en el parámetro `camino`.

Algunas especificaciones

- En caso que el `camino` no permita encontrar una persona en el árbol genealógico, entonces deberás retornar las conexiones existentes y un diccionario vacío.
- La lista de conexiones a retornar debe incluir el id de la persona de la cual queremos encontrar sus conexiones.

Ejemplos

Para los siguientes ejemplos se usa la tabla de datos de ejemplo en la sección 4.1

Ejemplo1:

```
>>> Solution().ejercicio3("C02", "MBDD")
["B03", "C01", "B02", "A01"], {"first_name": "Veronika", "last_name": "
    Mc Corley", "country": "Italy", "mom": "", "dad": "", "bestie": "C01"
    }
```

Explicación: Se busca a una persona siguiendo el camino `MBDD` en el siguiente orden:

- `M` La madre (mom) de "C02" → "B03"
- `B` El mejor amigo (bestie) de "B03" → "C01"
- `D` El padre (dad) de "C01" → "B02"
- `D` El padre (dad) de "B02" → "A01"

O en otras palabras, se busca al padre del padre del mejor amigo de la madre de C02

Ejemplo 2:

```
>>> Solution().ejercicio3("B02", "BDMDM")
["C03", "B03", "A01"], {}

```

Explicación: Se busca a una persona siguiendo el camino `BDMDM` en el siguiente orden:

- `B` El mejor amigo (bestie) de "B02" → "C03"
- `D` El padre (dad) de "C03" → "B03"
- `M` La madre (mom) de "B03" → "A01"
- `D` El padre (dad) de "A01" → No tiene padre, por lo que se retornará las conexiones hasta este momento y un diccionario vacío

3. Anexos

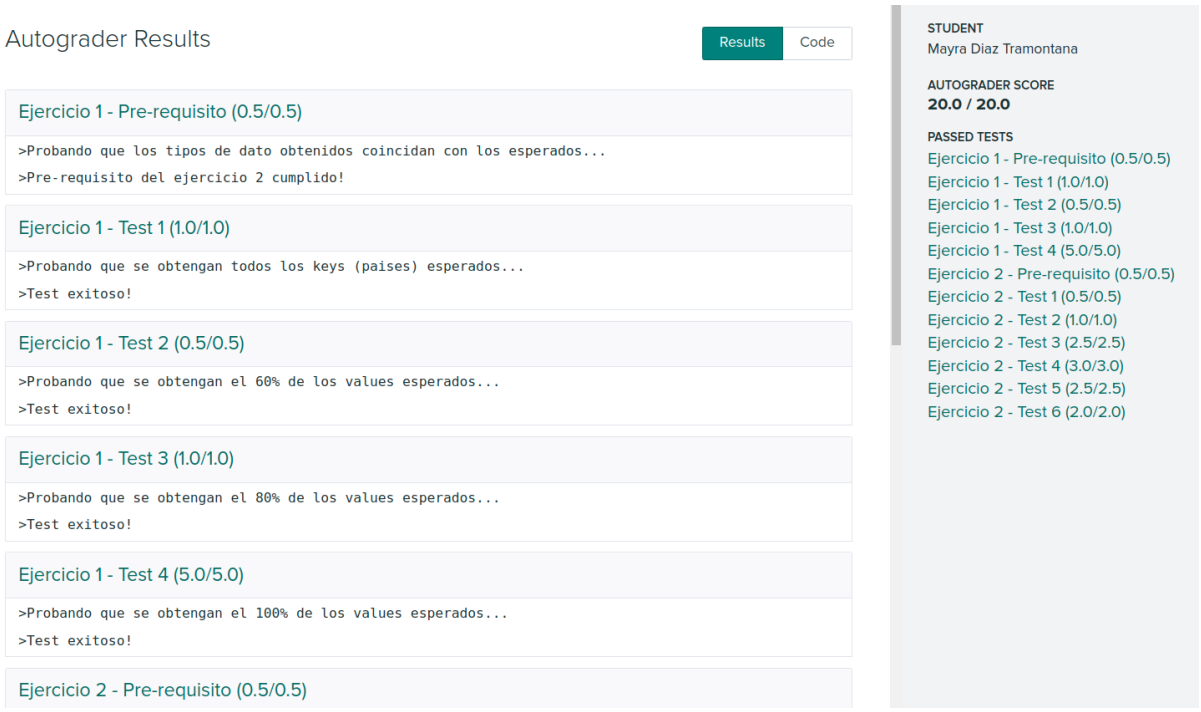

Figure 2: Casos de prueba correctos en Gradescope.

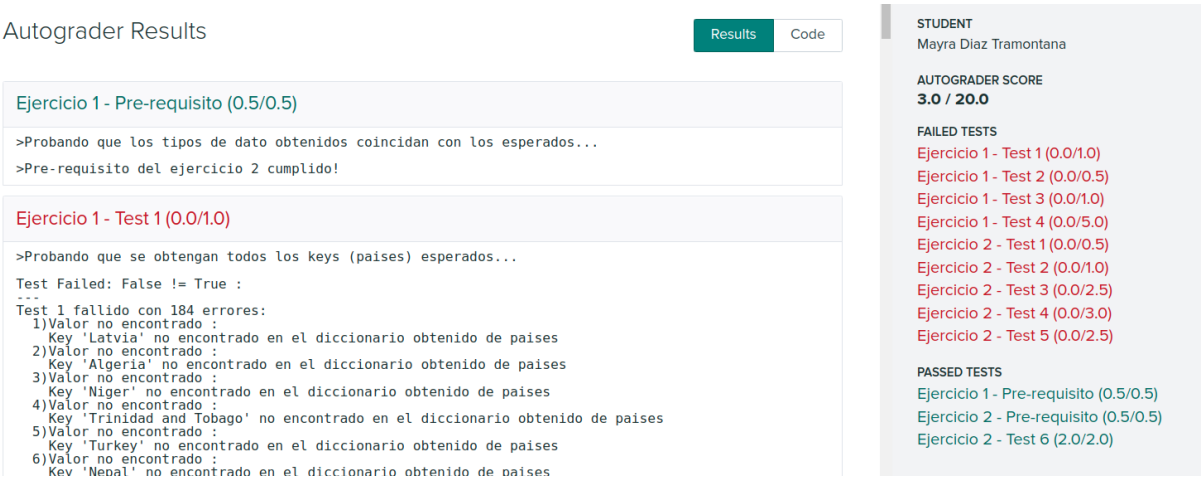

Figure 3: Entrega incorrecta en Gradescope.

```python
class Solution():
  data = {}

  def __init__(self, file_name = 'data.csv'):
    """Read file"""
    with open(file_name) as file:
      mat = [i.split(",") for i in file.read().split("\n")]
    self.data = {i[0] : dict(zip(mat[0][1:], i[1:])) for i in mat[1:]}

  def ejercicio1(self):
    """
    Contar la cantidad de personas por pais
    """
    paises = {}
    # SU SOLUCION EMPIEZA AQUI


        # SU SOLUCION TERMINA AQUI
    return paises

  def ejercicio2(self, id, camino):
    """
    Encontrar las conexiones
    """
    conexiones = []
    persona_buscada = {}

    # SU SOLUCION EMPIEZA AQUI


        # SU SOLUCION TERMINA AQUI
    return conexiones, persona_buscada

"""
s = Solution()
print(s.ejercicio1())

seq, info = s.ejercicio2("C18", "MMBD")
print(seq == ["B06", "A05", "B05", "A07"])
print(info == s.ejercicio1()["A07"])

seq, info = s.ejercicio2("D04", "DDBBMM")
print(seq == ["C04", "B01", "A12", "B13", "A11"])
print(info == {})

```

```
46 """
```

Listing 1: Template `solution.py`.