INF-253 Lenguajes de Programación
Tarea 1: Python

10 de marzo de 2022

1. Nueva Ciudad, Nueva Vida

Al comenzar el año, Patode Hule, un informático de la USM, se mudó a la ciudad de Pun Toycoma. La ciudad debido a la pandemia está modernizando sus sistemas de consultas respecto a las calles y personas que viven en esta, para ello desarrollan un lenguaje de programación que permita realizar estas consultas. Un informático de la ciudad, al escuchar que un nuevo informático llegó a la ciudad, le va directamente a pedir ayuda, ya que la USM tiene mucho prestigio. Le pide a Patode si es que puede ayudarlo a programar un checker, Patode pensando que era juego de damas asintió. Al descubrir que era para validar un lenguaje y ejecutarlo, Patode se asustó y les pide ayuda a ustedes para realizar el trabajo que aceptó.

Se debe utilizar Python 3 y la librería RegEx para las expresiones regulares de la tarea, en caso de que alguna de estas dos condiciones no se cumpla no se revisará la tarea.

2. Grafos++

2.1. Acerca de Grafos++

Grafos++ es un lenguaje generado para manejar sintaxis compleja de grafos, donde los nodos de este tengan información relevante de la ciudad: las calles, personas que vivan ahí y caminos posibles.

Los nodos de este grafo serán conocidos como una calle, esta tendrá la siguiente información:

- El nombre de la calle, este tiene que empezar sí o sí con una letra mayúscula y después puede contener números, letras, espacios y guiones.
  - Correcto: Vicuna Mackenna 3939
  - Incorrecto: vicuna
- El ID de la calle, está debe empezar con un #, luego debe seguir dos letras mayúsculas y finalizar con dos números entre 0 y 9, inclusives.
  - Correcto: #AC09
  - Incorrecto: AC
- Una o varias personas, donde las personas tienen la siguiente información:
  - Uno o más nombres, donde el primer carácter debe ir en mayúscula, luego le siguen caracteres en minúscula, guiones o espacios.
    - Correcto: Super king-boo

- Incorrecto: mario

- Uno o más apellidos, donde el primer carácter debe ir en mayúscula, luego le siguen caracteres en minúscula, guiones o espacios.
  - Correcto: Perez
  - Incorrecto: perez
- Teléfono, donde al principio tiene un carácter +, luego le sigue un número entre 1 y 9, para finalizar con cinco números entre 0 y 9.
  - Correcto: +599999
  - Incorrecto: +092
- Rut, el primer número del rut siempre tiene que ser entre 1 y 9, luego los siguientes números del rut pueden ser entre 0 y 9. Los últimos dos caracteres son: un guión, un número de un dígito o k. Además este rut debe ser válido, siguiendo el algoritmo descrito en la página: https://es.wikipedia.org/wiki/Rol_Único_Tributario.
  - Correcto: 19961926-8
  - Incorrecto: 011111-1

Toda esta información está descrita con el siguiente EBNF:

Code 1: EBNF general

```
calle ::= nombre_calle '.' ID_calle '.' persona/{persona}

ID_calle ::= '#' (A-Z) (A-Z) digito_o_zero digito_o_zero

nombre_calle ::= (A-Z) {(a-z | A-Z | digito_o_zero | '-' | ' ' )}

persona ::= nombre_persona {',' nombre_persona} '_'
            apellido_persona {',' apellido_persona} '_'
            telefono '_' rut

apellido_persona ::= (A-Z) {(a-z | '-' | ' ' )}

nombre_persona ::= (A-Z) {(a-z | '-' | ' ' )}

rut ::= digito {digito_o_zero} '-' (digito_o_zero | 'k')

telefono ::= '+' digito digito_o_zero digito_o_zero digito_o_zero
             digito_o_zero digito_o_zero

digito_o_zero ::= '0' | digito

digito ::= '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```

También se puede describir la conexión entre calles, de la siguiente forma ($calleA : calleB$), indicando que de la calle A se puede llegar a la calle B, esto se conoce como un *camino*. También se puede encontrar la siguiente sintaxis ($caminoA : caminoB$), indicando que de la última calle del camino A se puede llegar a la primera calle del camino B. Por último, se asumirá que los ID de la calles son únicos, por lo tanto si se puede tener la siguiente sintáxis ($calleA : ID\_calleC$), eso significaría que de la calle A se puede llegar a la calle que tenga la ID correspondiente.

Todo lo anterior se puede representar con el siguiente EBNF:

Code 2: EBNF camino

```
camino ::= '(' (camino | calle | ID_calle) ':' (camino | calle | ID_calle) ')'
```

Ejemplo de una sintaxis correcta sería:

Code 3: Ejemplo correcto

```
((Algarrobo.#GG11.S_T_+598765_111111-6:Salamanca.#EF12.M_L_+598765_111111-6):#DE12)
```

Esto representaría que de Algarrobo se puede ir a Salamanca y de Salamanca a la calle con id #DE12.

2.2. Comandos del lenguaje

El lenguaje Grafos++ corresponderá a un string que contiene diferentes comandos separados por ';'. Los posibles comandos que se pueden tener serán los siguientes:

- *calle*: creación de una calle en el grafo.
- *camino*: creación de un camino y las calles definidas dentro del camino en el grafo, siguiendo el formato anteriormente mencionado.
- *print ID_calle*: se debe imprimir por pantalla la información de la calle que tenga esa ID, con el formato siguiente:

```
CALLE:
nombre_calle
ID_calle
nombre_persona
apellido_persona
telefono
rut
```

  Si hay más de una persona, simplemente debe seguir imprimiendo con el mismo formato.

- *print_all*: se debe imprimir por pantalla todos los nodos con su información respectiva del grafo actual.
- *print_caminos ID_calle*: se debe imprimir por pantalla todos los caminos del grafo actual que tengan como nodo inicial ID_calle, con el siguiente formato:

```
CAMINOS DESDE ID_calle:
ID_calle->...->ID_calle_final
```

  En caso que el ID no exista imprima por pantalla "No existe esa calle".

- *print_by_nombre nombre*: se debe imprimir por pantalla el nombre de las calles que tengan personas con ese nombre, con el siguiente formato:

```
CALLES CON PERSONAS DE NOMBRE nombre:
nombre_calle
```

  En caso que no exista calle con ese nombre imprima por pantalla "No existen calles con personas con ese nombre".

- *print_by_apellido apellido*: se debe imprimir por pantalla el nombre de las calles que tengan personas con ese apellido, con el siguiente formato:

```
CALLES CON PERSONAS DE APELLIDO apellido:
nombre_calle
```

En caso que no exista calle con ese apellido imprima por pantalla "No existen calles con personas con ese apellido".

- *print_by_rut rut*: se debe imprimir por pantalla el nombre de las calles que tengan personas con ese rut, con el siguiente formato:

```
CALLES CON PERSONAS DE RUT rut:
nombre_calle
```

En caso que no exista calle con ese rut imprima por pantalla "No existen calles con personas con ese rut".

- *print_by_telefono telefono*: se debe imprimir por pantalla el nombre de las calles que tengan personas con ese teléfono, con el siguiente formato:

```
CALLES CON PERSONAS DE TELEFONO telefono:
nombre_calle
```

En caso que no exista calle con ese teléfono imprima por pantalla "No existen calles con personas con ese telefono".

- *update ID_calle calle*: se debe actualizar la calle con el respectivo id, con la información nueva.

- *valid_camino ID_calle_1 ID_calle_2*: se debe imprimir por pantalla "Si se puede" o "No se puede", si es que se puede llegar desde la calle con la ID_calle_1 a la calle con ID_calle_2. En caso que no exista una calle con alguna o ambas de las ID, debe imprimir por pantalla "No existe con ID_calle_1 o ID_calle_2".

2.3. Checkeo y ejecución

El objetivo es desarrollar un programa que permita verificar que los comandos sean escritos correctamente. En caso de que estén bien escritos, respetando la sintaxis descrita previamente, se debe ejecutar según lo descrito anteriormente; en caso contrario debe ignorarlo y escribir en un archivo llamado errores.txt el comando mal escrito. Si no hay líneas mal escritas, entonces el archivo errores.txt debe estar vació.

El input será ingresado a través de un archivo llamado input.txt y todo output de los comandos debe ser mostrado por consola.

En los siguientes enlaces se encuentran dos archivos de un input completamente correcto con su output respectivo:

- Input correcto
- Output consola respectivo
- Errores.txt respectivo

En los siguientes enlaces se encuentran dos archivos de un input con errores con su output respectivo:

- Input con algunos errores
- Output consola respectivo
- Errores.txt respectivo

3. Datos de Vital Importancia

- Al imprimir por consola los caminos, no importa el orden.
- Al escribir en el archivo errores.txt los comandos incorrectos, si importa el orden esto debido a que los comandos se van agregando a medida que se lee el string del código.
- Al no hacer uso de expresiones regulares, la tarea no será revisada.

4. Sobre Entrega

- Se deberá entregar un programa llamado grafospp.py.
- Si no existe orden en el código habrá descuento.
- Las funciones implementadas deben ser comentadas de la siguiente forma. **SE HARÁN DESCUENTOS POR FUNCIÓN NO COMENTADA**

  ”’
  Nombre de la función

  Parametro 1 : Tipo
  Parametro 2 : Tipo
  Parametro 3 : Tipo
  .....

  Breve descripción de lo que realiza la función y lo que retorna ”’
- Se debe trabajar de forma individual obligatoriamente.
- **La entrega debe realizarse en tar.gz y debe llevar el nombre: Tarea1LP_RolAlumno.tar.gz**
- **El archivo README.txt debe contener nombre y rol del alumno e instrucciones detalladas para la correcta utilización de su programa.**
- La entrega será vía aula y el plazo máximo de entrega es hasta el **28 de marzo a las 23:55 hora aula**.
- Por cada día de atraso se descontarán 20 puntos (10 puntos dentro la primera hora).
- Las copias serán evaluadas con nota 0 y se informarán a las respectivas autoridades.

5. Calificación

5.1. Entrega

- Uso correcto de expresiones regulares (total 20 pts):
  1. No usa expresiones regulare. Utiliza otros métodos que no son los pedidos para la tarea, por ejemplo split e IF (0 pts)
  2. Usa una gran y única expresión regular. Provocando así una falta de modularización de las expresiones, como las vistas en clases (MAX 6 pts)
  3. Crea diferentes expresiones regulares, logrando así las modularización, pero no las usa en conjunto para resolver el problema (MAX 12 pts)

4. Crea diferentes expresiones y las utiliza de manera correcta, aprovechandosé de la modularización generada. (MAX 20 pts)

- Detección de errores (total 25 pts):

1. No detecta correctamente ningún tipo de error (0 pts)
2. Permite detectar errores simples de sintaxis, sea este por ejemplo el uso de letras que no existen en el lenguaje (MAX 7 pts)
3. Detecta de errores de mayor dificultad, siendo estos secuencias largas de comandos anidados, pero detecta incorrectamente en una partes de estos (MAX 15 pts)
4. Detecta todo tipo de error probado, siendo no solo errores simples de sintaxis, si no que también errores de anidación de comandos o secuencias incorrectas (MAX 25 pts)

- Ejecución de comandos (total 55 pts, 5 pts c/u):

1. No puede realizar ningún tipo de ejecución de comandos, sea porque los detecta todos como errores o no lo realiza bien (0 pts)
2. Funciona en alguna de las características del comando (MAX 2,5 pts)
3. Funciona todas las características del comando correctamente (MAX 5 pts)

NOTA: Puede existir puntaje parcial, por ejemplo en expresiones regulares puede obtener 3 pts

5.2. Descuentos

- Falta de comentarios (-10 pts c/u MAX 30 pts)
- Falta de README (-20 pts)
- Falta de alguna información obligatoria en el README (-5 pts c/u)
- Falta de orden (-20 pts)
- Día de atraso (-20 pts por día, -10 dentro de la primera hora)
- Mal nombre en algún archivo entregado (-5 pts c/u)