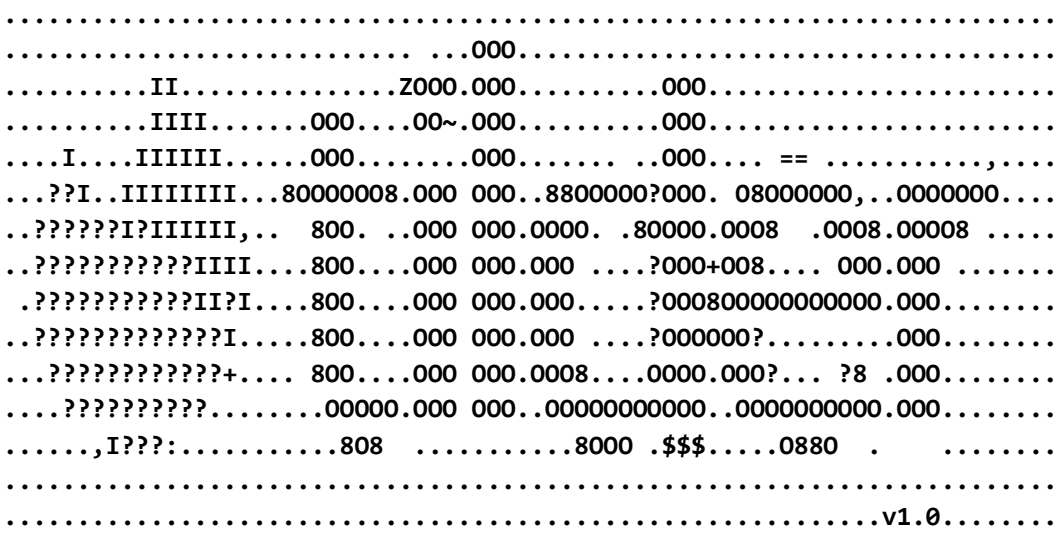
Universidad de Buenos Aires
Facultad de Ingeniería

75.40 Algoritmos y Programación I

Cátedra Pablo Guarna

Trabajo Práctico Grupal



Enunciado v1.2

El trabajo práctico grupal consiste en desarrollar un programa en Python que brinde la funcionalidad requerida y detallada a continuación.

Se busca crear un prototipo de una red social para las personas que buscan relacionarse con otras.

Personas

Definimos a las personas con los siguientes datos (se deben realizar todas las validaciones indicadas):

- Nombre
- Apellido
- Pseudónimo (único en el sistema, solo minúsculas, números y guión bajo)
- Contraseña (requiere al menos una mayúscula, una minúscula, un dígito decimal, y un largo mínimo de 5 caracteres)
- Sexo
    - Indefinido
    - Mujer
    - Hombre
- Edad (de 18 a 99)
- Ubicación actual: Latitud y longitud en grados decimales (ejemplo: 41.40338, 2.17403)
- Intereses
    - Los intereses son una lista de etiquetas (palabras sin espacios ni acentos, unidas con guiones medios) a ser ingresadas por los usuarios. Por ejemplo: basquet, green-day, star-wars, nueva-york, fotografia, francia, asado, bicicleta, taekwondo, buenos-aires.

Estructuras de datos

Para el trabajo práctico tendrán que almacenar las personas en un diccionario, cuya clave sea el pseudónimo de la persona, y el valor asociado a esta clave, todos sus datos. En otro diccionario, también usando el pseudónimo como clave, tendrán las búsquedas en una lista.

Menú principal y carga de usuarios

Cuando se ingresa al programa, habrá un menú principal que deberá tener las siguientes opciones:

- Cargar un grupo de personas predeterminado (conjunto de prueba) que estará fijo en el código y definido por el grupo. Este conjunto de datos deberán generarlo con los datos de 10 usuarios como mínimo.

- Cargar una nueva persona, ingresando todos los datos pertinentes y haciendo las validaciones adecuadas.
- *Opcional*: Permitir editar la información de una persona
- Ingresar al sistema

Ingreso al sistema y búsquedas

La entrada para ingresar al sistema solicitará pseudónimo y contraseña.

Una vez que el usuario ingresó al sistema con credenciales válidas se le pedirán por pantalla los parámetros para realizar las búsquedas:

-Radio máximo de búsqueda (en kilómetros, puede ser un flotante).

-Sexos de interés (puede ser hombre, mujer o ambos).

-Rango de edades (edad máxima y edad mínima a buscar).

Una vez solicitados los parámetros, se mostrará uno a continuación del otro aquellos usuarios que cumplan con las siguientes condiciones:

- Vivan dentro del área buscada por el usuario actual.
    - Deberá usarse la fórmula de Haversine para calcular la distancia entre 2 puntos y transformación de sistemas de coordenadas si fuera necesario. Se acepta utilizar alguna implementación de código abierto de esta fórmula obtenida de internet (citar URL en los comentarios). Esta funcionalidad deberá estar en un archivo separado del código del Trabajo Práctico y utilizarse mediante directiva **import**.
- Su sexo coincida con los sexos buscados por el usuario actual

Se mostrará también el porcentaje de coincidencia de intereses. Este cálculo será teniendo en cuenta el número total de etiquetas comunes entre ambos usuarios dividido el número de etiquetas de ambos sumado. Por ejemplo:

Usuario actual: *borges, **bicicleta**, rio-de-janeiro, los-simpsons, **sushi***
Usuario X: *arte, brasil, **sushi**, calamaro, mafalda, **bicicleta***

Porcentaje de coincidencia = 100 * 2 / (5 + 6) = 18% (redondear sin dígitos decimales).

Contacto entre usuarios

Antes de pasar al siguiente, el usuario actual puede optar por interesarse o no en contactarse con esta persona. En caso de interesarse, podrá marcarlo como interesado. En caso de no interesarse, marcará como ignorado.

Si al momento de marcarlo como interesado el otro usuario ya había marcado como interesado al usuario actual, entonces se ofrece como funcionalidad dejar un mensaje (texto libre). Al momento de ingresar el otro usuario al sistema se le mostrará el pseudónimo del usuario que respondió y el texto del mensaje recibido antes de mostrarle sus coincidencias.

El usuario puede optar por seguir viendo más coincidencias, o salir. El programa sale automáticamente (vuelve al menú principal) cuando no quedan más coincidencias en el sistema para mostrar.

Modo de entrega

El trabajo práctico deberá ser realizado en grupos de cinco personas.

El desarrollo del programa deberá ser modular, aplicando los conceptos de programación estructurada vistos en clase. En caso de que el corrector lo considere necesario, habrá una instancia de reentrega del tp con las correcciones indicadas.

La entrega será realizada en formato digital en la sección del campus destinada para ello. Deberán subir un archivo comprimido .zip (no subir extensiones .rar). En el zip deben incluir todos los archivos de extensión .py que hayan escrito (inclusive el set de prueba) y este enunciado.

Además, deberán preguntarle a su corrector si es necesaria una versión escrita de su código. En ese caso, deberán realizar la impresión del código utilizando una fuente monoespaciada con los números de línea del archivo.

Recuerden que en el campus cuentan con un foro para realizar consultas generales. También pueden enviarles mails a sus ayudantes o preguntar en clase.

Fórmula de Haversine:
https://web.archive.org/web/20090813162802/http:/gorny.edu.pl/haversine.py

```
#coding:UTF-8
"""
  Python implementation of Haversine formula
  Copyright (C) <2009>  Bartek Górny <bartek@gorny.edu.pl>

  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
"""

import math

def recalculate_coordinate(val,  _as=None):
  """
    Accepts a coordinate as a tuple (degree, minutes, seconds)
    You can give only one of them (e.g. only minutes as a floating point
number) and it will be duly
    recalculated into degrees, minutes and seconds.
    Return value can be specified as 'deg', 'min' or 'sec'; default return
value is a proper coordinate tuple.
  """
  deg,  min,  sec = val
  # pass outstanding values from right to left
  min = (min or 0) + int(sec) / 60
  sec = sec % 60
  deg = (deg or 0) + int(min) / 60
  min = min % 60
  # pass decimal part from left to right
  dfrac,  dint = math.modf(deg)
  min = min + dfrac * 60
  deg = dint
  mfrac,  mint = math.modf(min)
  sec = sec + mfrac * 60
  min = mint
  if _as:
    sec = sec + min * 60 + deg * 3600
    if _as == 'sec': return sec
    if _as == 'min': return sec / 60
    if _as == 'deg': return sec / 3600
  return deg,  min,  sec
```

```
def points2distance(start,  end):
  """
    Calculate distance (in kilometers) between two points given as (long,
latt) pairs
    based on Haversine formula
(http://en.wikipedia.org/wiki/Haversine_formula).
    Implementation inspired by JavaScript implementation from
http://www.movable-type.co.uk/scripts/latlong.html
    Accepts coordinates as tuples (deg, min, sec), but coordinates can be
given in any form - e.g.
    can specify only minutes:
    (0, 3133.9333, 0)
    is interpreted as
    (52.0, 13.0, 55.998000000008687)
    which, not accidentally, is the lattitude of Warsaw, Poland.
  """
  start_long = math.radians(recalculate_coordinate(start[0],  'deg'))
  start_latt = math.radians(recalculate_coordinate(start[1],  'deg'))
  end_long = math.radians(recalculate_coordinate(end[0],  'deg'))
  end_latt = math.radians(recalculate_coordinate(end[1],  'deg'))
  d_latt = end_latt - start_latt
  d_long = end_long - start_long
  a = math.sin(d_latt/2)**2 + math.cos(start_latt) * math.cos(end_latt) *
math.sin(d_long/2)**2
  c = 2 * math.atan2(math.sqrt(a),  math.sqrt(1-a))
  return 6371 * c


if __name__ == '__main__':
 warsaw = ((21,  0,  30),  (52, 13, 56))
 cracow = ((19, 56, 18),  (50, 3, 41))
 print points2distance(warsaw,  cracow)
```