Fundamentos de Algoritmos – Trabajo Práctico 2

Master in Management + Analytics, Escuela de Negocios, UTDT

Primer semestre 2020

Observaciones generales:

- El trabajo se debe realizar en los mismos grupos del TP1.
- El código fuente debe estar bien comentado. Cualquier aclaración adicional que se considere necesaria debe ser incluida como comentarios en el código fuente.
- Nombrar los archivos con el código fuente como se indica abajo, y subirlos a la *Entrega del TP2* en el campus virtual de la materia.
- Los programas entregados deben correr correctamente en `Python3`.
- La fecha límite de entrega es **el domingo 17 de mayo a las 23:55**. Los TPs entregados fuera de término serán aceptados, pero la demora incidirá negativamente en la nota.

Mapa Judicial de PBA

De la página `https://catalogo.datos.gba.gob.ar/dataset/mapa-judicial` puede descargarse un dataset sobre las dependencias judiciales de la Provincia de Buenos Aires. (También puede descargarse una copia de la página de la materia.) El archivo se llama `mapa-judicial.csv` y describe distintas características de cada dependencia judicial, incluyendo el fuero al que pertenece, el nombre de la dependencia, su dirección, el departamento judicial[1] en el que se encuentra, sus coordenadas geográficas (latitud y longitud) y otros. Cabe aclarar que para muchas dependencias judiciales, solo se tiene registro de algunos de sus atributos. El archivo contiene un total de 1823 dependencias judiciales.

El objetivo de este Trabajo Práctico es escribir programas para procesar estos datos y efectuar algunas consultas sobre los mismos. En concreto, se pide:

1. Implementar un tipo `DependenciaJudicial` que tenga, al menos, las siguientes operaciones:
   - `DependenciaJudicial(...)`: construye un nuevo elemento de tipo `DependenciaJudicial`, con los argumentos que se consideren necesarios.
   - `dj.fuero()`: devuelve el fuero (`str`) de la dependencia judicial `dj`.
   - `dj.nombre()`: devuelve el nombre (`str`) de la dependencia judicial `dj`.
   - `dj.tipo_de_ente()`: devuelve el tipo de ente (`str`) de la dependencia judicial `dj`.
   - `dj.direccion()`: devuelve la dirección (`str`) de la dependencia judicial `dj`.
   - `dj.localidad()`: devuelve la localidad (`str`) de la dependencia judicial `dj`.
   - `dj.departamento_judicial()`: devuelve el departamento judicial (`str`) de la dependencia judicial `dj`.
   - `dj.telefono()`: devuelve el teléfono (`str`) de la dependencia judicial `dj`.
   - `dj.distancia(lat,lng)`: devuelve la distancia euclídea (`float`) entre la dependencia judicial `dj` y el punto $\langle lat, lng \rangle$.[2]

[1] Los Departamentos Judiciales son las unidades territoriales en las que se divide la provincia a fin de su administración de justicia.
[2] La raíz cuadrada de un número `x` se puede calcular en Python con `x**0.5`.

- Operaciones para comparar dos dependencias judiciales por menor estricto (`dj1<dj2`) y por igual (`dj1==dj2`) (en Python esto se consigue definiendo `__lt__` y `__eq__`). Las comparaciones deben realizarse según el departamento judicial, fuero y el nombre de la dependencia judicial, usando en cada caso las comparaciones del tipo `str`. En el caso de `dj1<dj2`, comparar primero sus departamentos judiciales; si hay empate, sus fueros; y, si hay empate entre las dos anteriores, los nombres de ambas dependencias judiciales.
- Una operación para representar como string a una dependencia judicial (en Python esto se consigue definiendo `__repr__`), respetando el siguiente formato:
  `{fuero;nombre;dirección;localidad}`
  Por ejemplo, si `dj` es la dependencia judicial de la línea 2 del archivo CSV, entonces `str(dj)` debería devolver:
  `{PENAL;ASESORIA DE INCAPACES 1;AVENIDA PRESIDENTE PERON 525;AZUL}`

Guardar el código de la definición del tipo `DependenciaJudicial` en un archivo con nombre `dependencia_judicial.py`.

2. Escribir un programa (con nombre `localizar.py`) que lea las dependencias judiciales de un archivo CSV, y luego imprima la dependencia judicial que se encuentra más cercana a un punto dado.

   Este programa también deberá poder ejecutarse desde la línea de comandos y recibirá como argumentos el nombre del archivo CSV con las dependencias judiciales, y la latitud y longitud de un punto en el planeta. Como salida, imprimirá la información de la dependencia judicial más cercana encontrada. Por ejemplo, la invocación `python localizar.py mapa-judicial.csv -36.77571 -59.0886` debería imprimir por pantalla `{PAZ;JUZGADO DE PAZ;BOLIVAR 56;RAUCH}`.

   Para este programa deben ignorarse las dependencias judiciales que no tienen cargadas sus coordenadas geográficas.

3. Escribir un programa (con nombre `departamentos_judiciales.py`), que cargue los registros de dependencias judiciales de un archivo CSV en una lista de elementos de tipo `DependenciaJudicial`, y que luego escriba un archivo de texto con información de las dependencias judiciales de cada uno de los departamentos judiciales de la provincia.

   El archivo resultante deberá listar los departamentos judiciales, uno por línea, y ordenados en forma creciente según el nombre del departamento. Para cada departamento, debe imprimirse el nombre, seguido de dos puntos (“`:`”) y la lista de dependencias judiciales de ese departamento, ordenadas de menor a mayor, usando las operaciones de comparación y representación como string definidas en el tipo `DependenciaJudicial`, y separadas por punto y coma (“`;`”).

   El programa presentado deberá poder ejecutarse desde la línea de comandos y recibirá como argumentos los nombres de los archivos CSV de entrada y de salida.

   Por ejemplo, el comando
   `python departamentos_judiciales.py mapa-judicial.csv departamentos.txt` leerá el archivo `mapa-judicial.csv` y escribirá un nuevo archivo `departamentos.txt`, el cual debería comenzar así:
   `AZUL:{CIVIL Y COMERCIAL;CAMARA DE APELACION EN LO CIVIL Y COMERCIAL SALA 1; AVENIDA PRESIDENTE PERON 525;AZUL}{CIVIL Y COMERCIAL;CAMARA DE APELACION EN LO CIVIL Y COMERCIAL SALA 2;AVENIDA PRESIDENTE PERON 525;AZUL}{CIVIL Y COMERCIAL;CURADURIA OFICIAL - AZUL;BELGRANO 853;AZUL}` (...)

   Para este programa deben ignorarse las dependencias judiciales que no tengan cargado su departamento judicial.

4. Escribir un breve informe analizando la complejidad temporal de los programas de los puntos 2 y 3. Expresar el orden de complejidad, en el peor caso, en función de la cantidad de dependencias

judiciales (N) y/o de la cantidad de departamentos judiciales (M), según corresponda. Máximo una carilla, en formato PDF, con nombre `complejidad.pdf`.

Observaciones:

- Definir todas las funciones auxiliares que se consideren necesarias.
- Solo está permitido importar la biblioteca `sys`, para el manejo de los argumentos del programa principal con `sys.argv`.
- Para ordenar listas, está permitido usar las funciones provistas en Python `sorted(ls)` o `ls.sort()`. La implementación en Python de estas dos funciones tiene complejidad temporal $O(n \log n)$, donde n es la longitud de la lista `ls`.
- El archivo `mapa-judicial.csv` sigue la codificación `latin-1`. Puede suponerse que esto será así con cualquier archivo CSV con que se invoque a los programas. Al abrirlo para su lectura, deberá entonces indicarse el *encoding*: `archivo = open(archivo_csv, encoding="latin-1")`.
- En `mapa-judicial.csv` las latitudes y longitudes siguen el formato `e,d`, mientras que los floats de Python usan `e.d`. Por lo tanto, para hacer la conversión de strings a floats, se deberá previamente adaptar el formato. Puede suponerse que el formato de floats usado en los archivos CSV con que se invoque a los programas será el mismo que el de `mapa-judicial.csv`.
- Los valores de cada línea de `mapa-judicial.csv` están separados por punto y coma (`;`). Puede suponerse que este será el formato de cualquier archivo CSV con que se invoque a los programas.
- Puede suponerse que el usuario siempre invocará a los programas de manera correcta. Es decir, si hay errores en la cantidad, tipo o valor de los parámetros de entrada, no se espera comportamiento alguno del programa (puede colgarse o morir, por ejemplo).