Session 4: Implementing Pagerank

Q1 2022-23

Integrantes:
Pol Pérez Castillo
Daniel Ruiz Jiménez

Fecha de entrega: 18-11-2022

Índice

Introducción

En esta sesión vamos a implementar un programa en Python que simule el algoritmo de PageRank. Contaremos con dos ficheros sacados de OpenFlights, airports.txt y routes.txt, que cuentan con 5740 aeropuertos distintos de todo el mundo, con sus respectivos nombres, códigos IATA de 3 dígitos (más unos 2000 que no tienen código, por lo tanto no son útiles, que son filtrados previamente) y las diferentes rutas entre aeropuertos.

El algoritmo de PageRank calcula la importancia de un nodo basándose en las aristas entrantes de los demás nodos del grafo.

La fórmula es la siguiente:

$$P_{k+1}[i] = \lambda \cdot \sum_{j} \frac{P_k[j] \cdot w(j,i)}{out(j)} + \frac{(1-\lambda)}{n},\ \forall (j,i) \epsilon Routes$$

- w(i,j) es el peso de la arista del nodo i a j, en nuestro caso es el número de repeticiones de la misma arista.
- out(i) representa el número de aristas que salen del nodo i.
- λ es el factor damping.
- $\frac{1-\lambda}{n}$ es el factor de teleportación, es decir probabilidad de un salto aleatorio.

Implementación

Para implementar el algoritmo, hemos seguido al pie de la letra el pseudocódigo que se nos proporcionaba en el documento de la práctica:
- declaramos la variable N, que representa el número de aeropuertos (nodos del grafo).
-declaramos P, un vector inicializado a 1/N de tamaño N (la suma de todos sus elementos es 1).
-declaramos L, el damping factor (nosotros lo hemos inicializado con un valor de 0.85).
-mientras dos elementos del conjunto no converjan (su diferencia sea menor que 0.000001), declaramos un vector Q inicializado a 0, y sobre el número de nodos, igualamos cada valor de Q a L+el sumatorio del pageRank de todos los aeropuertos que tienen aristas entrantes al nodo en el que estamos, multiplicado por el peso de su arista (la ruta), y dividido por su grado de salida, y le sumamos un valor constante (1-L)/n.
-finalmente igualamos Q a P.

Con estos resultados se nos muestran una serie de pares (peso, nombre del aeropuerto), ordenados descendentemente por su PageRank.
No obstante, teníamos errores, y es que el valor en cada iteración de la suma de los elementos de P no era 1. Esto era porque no teníamos en cuenta aquellos **nodos desconectados**, aquellos que no tenían rutas hacia ellos (no era el destino de nadie) ni rutas hacia los demás aeropuertos (no era origen de nadie).
Esto lo solucionamos de la siguiente manera:
A cada Q[i] hemos añadido la suma proporcional de cada nodo desconectado del grafo.
De esta forma pasamos de un valor de P de 0.63 (con damping factor de 0.85) a converger siempre a 1 (sin importar el valor del damping factor).[1]

Estructuras de datos utilizadas

airportList: Lista de los aeropuertos con código IATA válido en orden de aparición. Usada para el acceso a todos los aeropuertos 1 a 1 sin importar el orden (en un bucle for - coste O(nº aeropuertos)) y para el acceso a un aeropuerto sabiendo su índice (coste O(1)).
airportHash: Diccionario de los aeropuertos siendo su llave el código IATA del respectivo aeropuerto. Usada para el acceso a un objeto aeropuerto sabiendo la llave (coste O(log(nº aeropuertos))) y para la comprobación de la existencia de un aeropuerto en en el conjunto de aeropuertos totales (coste O(log(nº aeropuertos))).
routes : Atributo de un objeto aeropuerto que es una lista de las aristas que tienen como destino ese aeropuerto. Funcionalidad idéntica a la de aiportList en cuanto a eficiencia.
routeHash: Atributo de un objeto aeropuerto que es un diccionario de las aristas que tienen como destino ese aeropuerto. Funcionalidad idéntica a airportHash en cuanto a eficiencia.

[1] Véase el archivo PageRank.py para la implementación exacta del método.

Coste algorítmico del programa

Observando el código desde fuera, se podría llegar a pensar que el coste del programa es O(nº aeropuertos * nº aristas) ya que existe un bucle for de nº aeropuerto iteraciones que contiene una llamada a la función sumatorio que tiene coste O(nº aristas). Pero en este caso cada arista no puede ser visitada más de una vez, esto quiere decir que si en la primera iteración ya se visitan todas las aristas, el resto simplemente devolvería 0 como valor del sumatorio. Esto nos indica que el nº aeropuertos no influye en el coste real del programa. Por lo tanto el coste es O(nº aristas), tal y como se nos pedía.

Experimentación

error/damping factor	**10*10^-10**	**10*10^-14**	**10*10^-18**
0.5	1 iteración 0.01376 s	35 iteraciones 0.4081 s	42 iteraciones 0.5458 s
0.7	1 iteración 0.01363 s	67 iteraciones 0.8813 s	78 iteraciones 1.0357 s
0.9	1 iteración 0.01372 s	225 iteraciones 2.6625 s	259 iteraciones 2.9316 s

Hemos realizado un conjunto de pruebas para analizar cómo afectaba el error y damping factor en el número de iteraciones y tiempo de ejecución del programa.
Hemos observado lo siguiente:

- Hasta un error mínimo de 10*10^-14, el cómputo del PageRank se ejecuta siempre en 1 iteración. Esto lo hemos asociado a que por la definición de una iteración en la fórmula que se nos da, el sumatorio de todos los valores de P converge siempre a 1, y la primera diferencia notable es a partir de 14 decimales.
- Cuanto mayor es el damping factor, más iteraciones y tiempo tarda en ejecutarse. Esto se debe a que la probabilidad de hacer un salto aleatorio es más alta.
- Cuanto menor es el error, más iteraciones y tiempo tarda en ejecutarse, ya que la condición para salir del bucle se reduce.
- Aún con errores considerablemente pequeños como es el 10*10^-18 podemos ver que el tiempo de ejecución sigue siendo de pocos segundos tal y como se nos pide en el enunciado.