Andrés Visciglio - 100118, Dan Tchechenistky - 100117

**Informe TP3**

Lo primero que se hizo es crear un objeto Grafo y un objeto Vértice para simplificar el proceso.

En el caso del objeto Grafo, se utilizó un diccionario para guardar objetos Vértice (valores) con sus identificadores (claves). Esto hace que el proceso de guardar, borrar y obtener un vértice sea O(1). Ya que buscar o agregar una clave en un diccionario es O(1). Se utilizaron también dos variables que guardan la cantidad de vértices y de aristas para que obtener estos valores sea O(1).

En el caso del objeto Vértice se utilizó un diccionario para guardar los vértices adyacentes ya que esto hace que el proceso de agregar una arista sea O(1) como se explicó en el objeto Grafo.

A lo largo del TP se utiliza para varias funciones el algoritmo de Random Walks que, recibe un vértice y un largo y va visitando vértices adyacentes de manera equiprobable. Generando de esta manera, como su nombre lo dice, caminos aleatorios entre los vértices del grafo. Esto es útil al momento de querer encontrar relaciones de manera aleatoria.

En nuestro caso el orden sería O(largo\*cantidad) ya que va eligiendo vértices aleatorios en O(1) según el largo, y repite este proceso según la cantidad de caminos que se desean. Devolviendo finalmente un diccionario con la cantidad de apariciones por cada vértice que se visitó en todos los Random Walks.

En el caso de la función “Similares” siendo n la cantidad de elementos que se busca, se hacen n\*100 random walks de largo 30 O((n\*100)\*30). Luego se crea un heap de menores O(n log(n)) para seleccionar los n que tuvieron más apariciones en los random walks. Y finalmente se los imprime por pantalla. El orden final sería O(n\*300) + O(n log(n)).

La función “Recomendar” es casi igual a la función “Similares”. Solo se cambia el tamaño del heap que en este caso es n\*2. Y se agrega una comparación para ver si alguno de los vértices a recomendar ya se encuentra en los adyacentes del vértice recibido por parámetro, ya que esto se quiere evitar en el caso de las recomendaciones. El orden sería O(n\*300) + O(n\*2 log(n\*2)).

Otro comando que utiliza Random Walks es centralidad. Aquí se recibe por parámetro la cantidad de usuarios más centrales que se quiere obtener (n). El siguiente proceso se hace n\*50 veces. Se obtiene un usuario random y se hacen 10 random walks de largo 5 O(50) y luego se llena un diccionario con la cantidad de apariciones en O(1). El orden de este proceso sería entonces O(n\*2500). Luego se crea un heap de largo n, O(n log(n)) y se imprime por pantalla. El orden final sería entonces O(n\*2500) + O(n log(n)).

Para el resto de los comandos utilizamos el algoritmos de BFS. Su función es recorrer, sin repetir, todos los vértices que están conectados, por orden de adyacencia. Durante el recorrido se guarda información en simultáneo sobre la procedencia de cada vértice (padre) y la distancia al vértice de origen (orden). Los comandos utilizados dentro de las iteraciones tienen un orden de O(1). Es por eso que el orden de la función depende de la densidad del grafo, siendo en el peor de los casos O(V + E).

El comando Distancia recibe el identificador de un vértice y utilizando el recorrido BFS, obtiene la cantidad de usuarios existentes para cada nivel(distancia) posible. El orden del algoritmo está determinado por tres bloques principales; el llamado a BFS, la iteración sobre los diccionarios y el uso de la función sort( ).  
Recorridos BFS tiene, en el peor de los casos, un orden de O(V + E). Es por esto que, la iteración sobre los diccionarios, siempre será menor o igual al orden de BFS, ya que este último es el que crea y completa el diccionario.

Finalmente, asumiendo que la función sort() de python utiliza Quicksort, el orden sería de O(E.Log(E)) (se ordena la lista de aristas). Dado que sort( ) tiene el mayor orden de ejecución del comando, se concluye que O(E.Log(E) + O(E + V)) es el orden final.

De manera similar al comando Distancia, Camino realiza un llamado a recorrido BFS y recorre el diccionario conteniendo todos los padres. El objetivo es mostrar una forma de llegar de un punto dado hacia otro, lo que se logra accediendo al diccionario desde el último vértice al primero. Se concluye entonces que el orden del algoritmo es O(E + V), ya que el resto de las operaciones son simples y de tiempos constantes O(1).

Para el comando de estadísticas las operaciones son todas O(1) salvo para calcular el promedio de entrada y de salida de los vértices del grafo. Para esto hay que recorrer todos los vértices e ir sumando la cantidad de adyacentes que tienen. Orden O(V)que sería el orden final de esta función.

Finalmente, el orden del algoritmo Comunidades está determinado por su bloque principal, que consiste de una serie de iteraciones anidadas. La primera y principal, fija en cuatro iteraciones el total del contenido. La siguiente recorre todos los vértices. Las últimas dos no se encuentran anidadas y recorren como máximo la cantidad de vértices adyacentes a cada vértice. Dado que el resto de las operaciones se encuentran acotadas por el orden máximo, siendo este O(4\*V\*E) con “E” la máxima cantidad de adyacentes en un grafo denso.