REPORTE

Análisis de algoritmos

David Alfonso Velasco Sedano

1 de Diciembre del 2017

Descripción del problema

Se desea encontrar una forma de organizar resultados en base a 2 criterios:

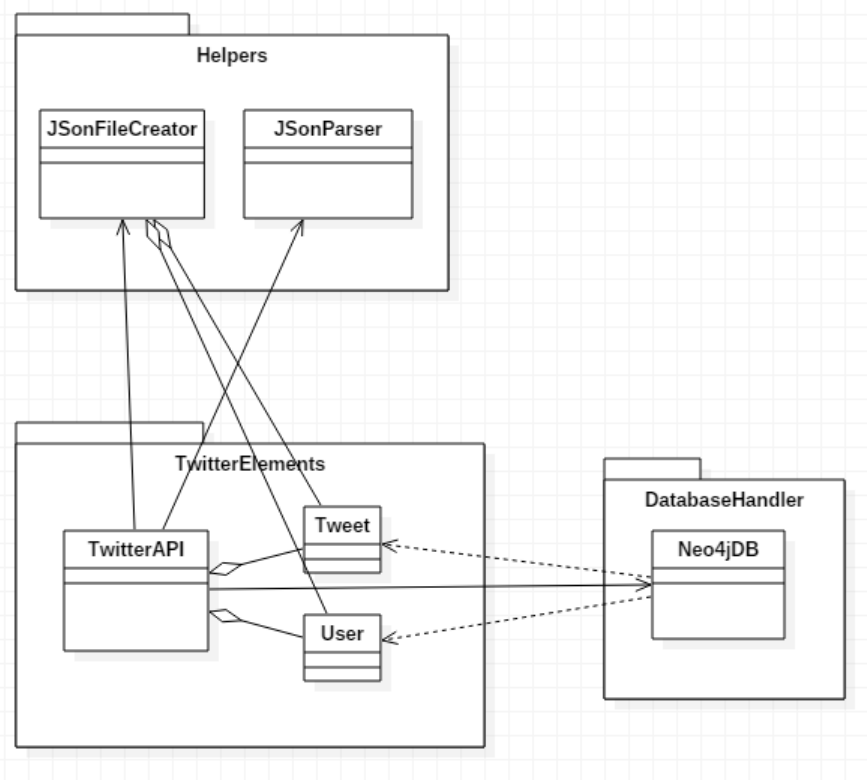
- La influencia de los usuarios sobre un tópico particular
- Dividir y hacer reconocibles las comunidades dentro del criterio de búsqueda

En este escenario, estaremos haciendo una serie de búsquedas en la página Twitter. El criterio de búsqueda puede ser un usuario o un texto deseado dentro de los Tweets. El primer obstáculo que nos estaremos encontrando es la obtención de datos.

El siguiente dato va a ser almacenar los datos e ir identificando el grado que tiene cada usuario (tanto por sus conexiones salientes como entrantes). Finalmente, será el poder identificar en que comunidad se encuentra cada usuario y darle un color único.

Solución propuesta.

La solución está compuesta de un total de 6 clases entre 3 paquetes.



- TwitterElement: Aquí podemos encontrar el punto de partida de todo el sistema. Se va encargar de realizar la conexión con Twitter al igual que gestionar los demás componentes para analizar y limpiar la información que nos esté entrando.
- Helpers: Es un paquete enfocado a cualquier clase que nos ayude con el manejo de los JSons. Se puede traducir a clases que generen un archivo de tipo JSon o clases que interpreten (lean) un JSon.
- DatabaseHandler: Debajo de este paquete es la interacción con la base de datos de Neo4j. Es la forma en que nos estaremos comunicando con nuestra base de datos. (NOTA: Por ejercicios del proyecto será ignorada).

La solución que se estará hablando en el transcurso de este proyecto será la generación de colores (ubicación de que elementos se encuentran en cuál comunidad), el cual se llama 'setCluster' y se encuentra ubicado dentro de la clase JSonFileCreator. La razón de esto es dada a que se tienen que hacer una búsqueda dentro de un grafo bidireccional. Existiendo dos tipos de conexiones, la que contesta la pregunta de qué usuario creo cuál tweet y cuáles usuarios están siendo mencionados por cuál tweet. Por lo cuál se pueden generar circuitos dentro de la comunidad o estas están aisladas.
La idea general del código es agarrar de forma ordenada un Tweet. De este, ir tratando de meternos a profundidad, tratando de ubicar a todos sus usuarios y qué Tweets interactúan con ellos. Se van agregando a una pila los Tweets que estén siendo encontrados de esta forma para su futuro procesamiento.
Idea el código:
Por cada elemento del universo->
Actualizar el ID del nuevo clúster
Mientras haya Tweets que analizar de la raíz elegida
Remuevo el Tweet
Verificar si es candidato para ser procesado, si no es, saltamos al siguiente Tweet
Actualizó los datos del Tweet (pongo clúster y que ha sido procesado)
Si el creador del Tweet no ha sido procesado
Agarrar los Tweets de dónde ha sido mencionado y agregarlos a la lista por procesar
Actualizar sus valores (poner el clúster actual y que ha sido procesado)
Por cada usuario que el Tweet haya mencionado
Si ha sido mencionado, nos saltamos al siguiente
Si no, actualizamos sus datos (poner el clúster actual y que ha sido procesado)
Si tiene Tweets creados por él y que haya sido mencionado en, agregarlos a la lista
Las entradas requeridas para esté elemento son los universos de dónde agarrar los Tweets y Usuarios. La salida son los mismos universos, únicamente con sus campos actualizados para reflejar en que universo pertenecen en.

Código:

```
private bool setCluster() {
            try {
                //Setting the stage
                List<string> pendingToProcess = new List<string>();
                int cluster = 0;
                int tweetsAgregados = 0;
                int tweetsProcesados = 0;
                int usuariosProcesados = 0;
                DateTime start = DateTime.Now;

                foreach(string tweetID in tweetUniverse.Keys) {
                    if (tweetUniverse[tweetID].process)
                        continue;

                    pendingToProcess.Add(tweetID);
                    cluster++;

                    while(pendingToProcess.Count() > 0) {
                        string currentID = pendingToProcess[0];
                        pendingToProcess.RemoveAt(0);
                        tweetsAgregados++;
                        if (tweetUniverse[currentID].process)
```

```
                        continue;
                    tweetsProcesados++;
                    tweetUniverse[currentID].cluster = cluster;
                    tweetUniverse[currentID].process = true;

                    string creatorID = tweetUniverse[currentID].createdBy;

                    List<string> tempList = userUniverse[creatorID].mentionedIn;

                    if (!userUniverse[creatorID].process) {
                        userUniverse[creatorID].process = true;
                        userUniverse[creatorID].cluster = cluster;

                        if (tempList.Count() > 0)
                            pendingToProcess.AddRange(tempList);

                        usuariosProcesados++;
                    }

                    string[] usersMentioned = tweetUniverse[currentID].mentions;
                    if(usersMentioned != null) {
                        foreach (string userMentionedID in usersMentioned) {
                            if (userUniverse[userMentionedID].process)
                                continue;
                            usuariosProcesados++;
                            userUniverse[userMentionedID].cluster = cluster;
                            userUniverse[userMentionedID].process = true;

                            tempList = userUniverse[userMentionedID].created;
                            if (tempList.Count() > 0)
                                pendingToProcess.AddRange(tempList);
                            tempList = userUniverse[userMentionedID].mentionedIn;
                            if (tempList.Count() > 0)
                                pendingToProcess.AddRange(tempList);
                        }
                    }
                }
            }

            DateTime end = DateTime.Now;
            Console.WriteLine("Time passed: " + end.Subtract(start).Milliseconds);
            Console.WriteLine("Tweets totales: " + tweetUniverse.Count() + ",
Usuarios totales: " + userUniverse.Count());
            Console.WriteLine("Tweets agregados: " + tweetsAgregados + " procesados:
" + tweetsProcesados);
            Console.WriteLine("Usuarios que fueron procesados: " +
usuariosProcesados);

            //createHexTable(cluster);
             return true;
        }catch(Exception e) {
            Console.WriteLine(e.Message);
            return false;
        }

    }
```

Pruebas y resultados

Para las pruebas, se están agregando 4 campos para darle seguimiento.

1. La cantidad de tiempo que tarda el código en ejecutarse, en milésimas de segundos.
2. La cantidad de Tweets que son agregados a la lista de ítems por procesar
3. La cantidad de Tweets que realmente fueron procesados
4. La cantidad de usuarios que realmente fueron procesados

De ahí, se hizo un recorrido empezando con 5 y terminando en una cantidad menor a un valor menor a 20500. Esto es por limitación de la aplicación. Al final, conseguimos las siguientes tablas:

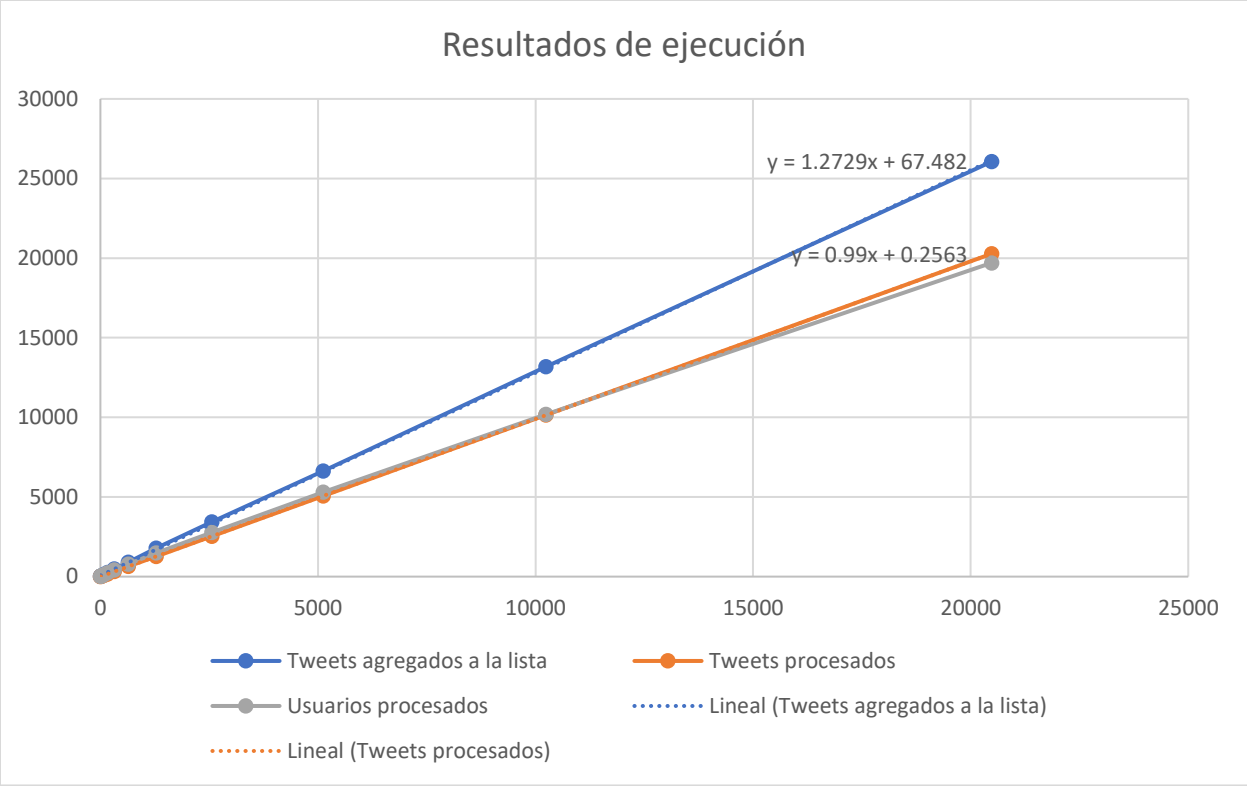
Tabla 1: Resumen de los elementos conseguidos de Twitter

# de Tweets	Total de Tweets	Total de usuarios
5	5	12
10	10	17
20	20	35
40	40	64
80	80	125
160	159	234
320	317	427
640	634	786
1280	1268	1492
2560	2534	2779
5120	5068	5313
10240	10137	10182
20480	20275	19701

Tabla 2: Procesamiento de los datos

# de Tweets	Tiempo en milisegundos	Tweets agregados a la lista	Tweets procesados	Usuarios procesados
5	0	10	5	12
10	0	17	10	17
20	0	36	20	35
40	0	66	40	64
80	0	133	80	125
160	0	256	159	234
320	1	473	317	427
640	0	909	634	786
1280	1	1789	1268	1492
2560	4	3433	2534	2779
5120	4	6637	5068	5313
10240	19	13190	10137	10182
20480	33	26060	20275	19701

Gráfica enseñando la formula de tendencia de cada uno de los campos (excluyendo tiempo de ejecución)



Análisis de la complejidad temporal y espacial (puede ser a priori o a posteriori)

Con resultados agarrados, estaremos asumiendo que nuestra formula cae dentro de una complejidad temporal y espacial de tipo O(n).

Demostración de complejidad temporal

G(N) = 1.27N + 67 – quitamos la constante → 1.27N

F(N) = N

Vamos a comprobar para:

$C_0 = 2$ y $N_0 = 1$

$g(N) \in O(f(N))$
$g(N) = 1.27N$
$f(N) = N$
$1.27N \leq 2N$

N	1.27N	2N
1	1.27	2
2	2.54	4
3	3.81	6
4	5.08	8
5	6.35	10
6	7.62	12
7	8.89	14

8	10.16	16
9	11.43	18
10	12.7	20

Como podemos observar, para todo N mayor e igual a uno, se cumpla que estamos dentro de la complejidad O(N).

Demostración de complejidad espacial

G(N) = .99N + .25 – quitamos la constante → .99N
F(N) = N
Vamos a comprobar para:
$C_0 = 1$ y $N_0 = 1$

g(N) **ϵ** O(f(N))
g(N) = .99N
f(N) = N
.99N ≤ N

N	.99N	N
1	0.99	1
2	1.98	2
3	2.97	3
4	3.96	4
5	4.95	5
6	5.94	6
7	6.93	7
8	7.92	8
9	8.91	9
10	9.9	10

Como podemos observar, para todo N mayor e igual a uno, se cumpla que estamos dentro de la complejidad O(N).