El **analizador de Tags** proporcionará información de los tweets desde diferentes perspectivas.

- Visión global:
  · Top 5 Clientes más utilizados para acceder a twitter
  · Top 5 Clientes más utilizados para escribir tweets
  · Top 5 Clientes más utilizados al escribir tweets con tags
  · Top 5 Clientes más utilizados al escribir tweets con menciones
  · Top 5 Clientes más utilizados para hacer retweet

- Visión desde los tags:
  · Top 10 Hashtags más utilizados durante la vida del sistema
  · Top 10 Hashtags más utilizados en retweets
  · Top 10 Hashtags más utilizados en tweets con menciones
  · Representación gráfica con evolución temporal

- Visión desde los usuarios (por cada tag del top 10):
  · Top 10 usuarios con más actividad con el tag
  · Top 10 usuarios con más tweets escritos con el tag
  · Top 10 usuarios con más retweets con el tag
  · Top 10 usuarios con más menciones en tweets con el tag
  · Representación gráfica con evolución temporal

Implementación:
Para llevarlo acabo, habrá un proceso principal que se comunicará con el gestor de colas para recibir la información de los tags.
Lo irá almacenando en un buffer de tamaño BUFFSIZE.
También hay un proceso tagsAnalyzer y un proceso globalAnalyzer que leerán del buffer la información extraída en concurrencia y la procesarán.

tagsAnalyzer guardará los resultados en una lista enlazada en la que cada nodo contendrá el Hashtag que es, el número de apariciones totales, en retwteets y en tweets con menciones, y para el total de apariciones, los retweets y los tweets con mención, un string con el siguiente esquema: “fechaMasAntigua;AutorPrimerTweet/fecha4;ATweet/…”
“fechaMasAntigua;AutorPrimerRetweet/…”
“fechaMasAntigua;AutorPrimerMenciont/…”

globalAnalyzer guardará también los resultados en una lista enlazada en la que cada nodo tendrá el cliente que es, número de usos totales del cliente con el que se accede a twitter, con el que se escriben twteets, con el que se escriben tweets con tags, con el que se escriben tweets con menciones y con el que se hace retweet.
Además habrá un string asociado a cada número, que guardará el registro del momento en que se utilizó el cliente para las distintas acciones, por ejemplo:
“fechaMasAntigua/fecha4/…”

Una vez hayan terminado de leer ambos procesos el contenido del buffer, el proceso principal reescribirá el buffer con nuevos contenidos.
Se continúa con el mismo funcionamiento hasta que no hay más datos que procesar, determinado por una marca de fin que mandará el gestor de colas.
Una vez se finalice, se lanzarán 10 procesos userAnalyzer que producirán los resultados de cada tag del top 10 obtenidos anteriormente para los 10 usuarios más activos con el Hashtag.

Cada userAnalyzer contendrá una lista enlazada con una estructura similar a las 2 listas anteriores: un identificador, y pares formados por un entero que representa un número de acciones, y un string que registra cada accion leida en el tiempo.

Finalmente, el proceso Cliente, leerá de cada lista generada por cada proceso descrito la información necesaria para mostrar por pantalla las conclusiones que mostramos al principio

Un acercamiento en pseudocódigo de las listas anteriores sería lo siguiente:

```
TAD lista enlazada resultadosTags {
  struct nodo {
     string Hashtag
     interger apHashtag;
     string quienAH;
     interger apHashtagRT;
     string quienAHRT;
    interger apHashtagM;
    string quienAPM;
  }
  nodo* init;
  interger totalHashtags;
};

TAD lista enlazada resultadosGlobales {
  struct nodo {
     string Client;
     interger usoClient;
     string cuandoUC;
     interger usoTweetC;
     string cuandoUTC;
    interger usoTTagC;
    string cuandoUTTC;
    interger usoTMencionC;
    string cuandoUTMC;
    interger usoRTC;
    string cuandoURTC;
  }
  nodo* init;
  interger totalClientes;
};
```

***Nota: TODOS los procesos involucrados en el analizador, tendrán como memoria compartida los siguientes datos:**

- **el buffer que se gestiona su uso con el monitor contado a continuación y su entero size que indica hasta dónde hay datos en el buffer.**
- **los punteros a las diversas listas enlazadas contadas anteriormente**

Para la gestión de la sincronización del buffer con tagsAnalyzer, globalAnalyzer y el principal, hemos pensado en usar un monitor gestorBuffer que tendrá como recursos compartidos:

- Un contador “listo” inicializado a 0 para identificar cuando han completado de leer los 2 procesos
- Una variable booleana estaEscribiendo inicializada a falso para avisar que el principal está escribiendo en el buffer.

*Nota: en el constructor del monitor, al inicializarlo, se rellena el buffer con lo leído por el principal.

Las variables condición serán esperandoLectores y esperandoEscritor.
Este monitor tiene las operaciones:

- quieroLeerBuffer() {
  if (estaEscribiendo = TRUE)
  waitC(esperandoLectores);
  end
  }
- dejoLeerBuffer() {
  listo:= listo + 1;
  if (listo = 2)
  signalC(esperandoEscritor);
  end
  }
- quieroEscribirBuffer() {
  if (listo != 2)
  waitC(esperandoEscritor);
  end
  estaEscribiendo:= TRUE;
  }
- dejoEscribirBuffer() {
  listo:= 0;
  estaEscribiendo:= FALSE;
  signalCAll(esperandoLectores);
  }

Los userAnalyzer no necesitan ninguna gestión de sincronización ya que no son dependientes entre sí y únicamente leen los datos compartidos. Es un trabajo en paralelo para agilizar la tarea.

Un esquema de alto nivel de los diferentes procesos que intervienen:

Nota: el main lanza todos los procesos involucrados y los lanza en el siguiente orden:
start
lanzo procesos globalAnalyzer y tagsAnalyzer
(concurrencia)
fin procesos anteriores
lanzo procesos userAnalyzer (i:1..10)
(trabajo en paralelo)
fin procesos
lanzo proceso cliente
end

```
channel of string c
Process main
  string mensaje;
  c => mensaje
  while (mensaje != “FIN”)
          quieroEscribirBuffer();
          //escribir(c,mensaje,buffer,size);
          dejoEscribirBuffer();
  end
end

Process globalAnalyzer
  // TAD lista enlazada resultadosGlobales;
  boolean end:= FALSE
  while (end = FALSE)
          quieroLeerBuffer();
          //leer(end,resultadosGlobales,buffer,size);
          dejoLeerBuffer();
  end
end

Process tagsAnalyzer
  // TAD lista enlazada resultadosTags;
  boolean end:= FALSE
  while (end = FALSE)
          quieroLeerBuffer();
          //leer(end,resultadosTags,buffer,size);
          dejoLeerBuffer();
  end
end

Process userAnalyzer (i:1..10)
  // TAD lista enlazada resultadoUser i
  // producirResultados(resultadoUser i, resultadosTags)
end
```

Process Cliente

```
  // mostrarResultadosGlobales(resultadosGlobales);
  // mostrarResultadosTags(resultadosTags);
  // mostrarGraficas(resultadosTags)
  for (i:1..10)
          // mostrarResultadosUser(resultadoUser,i)
           // mostrarGraficas(resultadosUser,i)
  end
end
```

El **analizador de rendimiento** pretende mostrar los siguientes resultados:

-tiempo medio que le cuesta resolver a cada worker 200 bloques de tareas
-gráfico temporal (para cada worker) tiempo de ejecución cada 200 bloques
-worker más lento y el más rápido (rango en que oscila el tiempo de ejecución medio)
-bloque más lento y más rápido

Implementación:
Se creará un proceso que se comunicará con el gestor de colas e irá guardando los resultados de cada worker en terminar un bloque en una variable que tras llegar a haber acumulado 200, se calculará la media y se entregará la información de cada worker en un nodo de la lista enlazada que utilizaremos (el cual contiene el identificador del worker, un string con la estructura: “primerBloqueDoscientos/bloque56Doscientos/…”) y un entero donde se guarda la media. Además se irá actualizando conforme se completen bloques de 200 almacenado en las variables.
Además al calcular la media en las variables, se tendrá una variable que guardará el bloque más rápido y el más lento que se tendrá que comparar a cada 200 bloques completos.

Al finalizar el sistema habrá un proceso cliente que creará un informe que se mostrará por pantalla.
*Nota: el cliente y el analizador de Rendimiento tendrán en común la lista con los worker llamada timeWorkers y las 2 variables bestTime y worstTime.
Un esquema de alto nivel de los procesos:

```
channel of string c
Process AnalizadorRendimiento
  string mensaje;
  struct auxWorker {
        interger acumworker;
        interger numBloque;
  }
  struct array auxWorker workers[5]
  c => mensaje
  while (mensaje != “FIN”)
        tratarMensaje(mensaje,workers,timeWorkers,bestTime,worstTime);
        // para averiguar donde guardar mensaje y
        // si se llega alguno del vector a numBloque = 200 se divide su acumWorker/200
        // y se manda a la lista enlazada
  end
end

Process Cliente
  // mostrarRendimientoMedio(timeWorkers)
  // mostrarGraficas(timeWorkers)
  // mostrarRangoTimeWorkers(timeWorkers)
  // mostrarMasLentoYRapido(bestTime,worstTime)
```