Universidad de Valladolid

E.T.S Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática,
mención en Computación

Algoritmos para Big Data

Autor:
Sergio García Prado

Universidad de Valladolid

E.T.S Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática,
mención en Computación

Algoritmos para Big Data

Autor:
Sergio García Prado

Tutor:
Manuel Barrio Solórzano

Prefacio

Para entender el contenido de este documento así como la metodología seguida para su elaboración, se han de tener en cuenta diversos factores, entre los que se encuentran el contexto académico en que ha sido redactado, así como el tecnológico y social. Es por ello que a continuación se expone una breve descripción acerca de los mismo, para tratar de facilitar la compresión sobre el alcance de este texto.

Lo primero que se debe tener en cuenta es el contexto académico en que se ha llevado a cabo. Este documento se ha redactado para la asignatura de **Trabajo de Fin de Grado (mención en Computación)** para el *Grado de Ingeniería Informática*, impartido en la *E.T.S de Ingeniería Informática* de la *Universidad de Valladolid*. Dicha asignatura se caracteriza por ser necesaria la superación del resto de las asignaturas que componen los estudios del grado para su evaluacion. Su carga de trabajo es de **12 créditos ECTS**, cuyo equivalente temporal es de *300 horas* de trabajo del alumno, que se han llevado a cabo en un periodo de 4 meses.

La temática escogida para realizar dicho trabajo es **Algoritmos para Big Data**. El Big Data es la disciplina que se encarga de "todas las actividades relacionadas con los sistemas que manipulan grandes conjuntos de datos. Las dificultades más habituales vinculadas a la gestión de estas cantidades de datos se centran en la recolección y el almacenamiento, búsqueda, compartición, análisis, y visualización. La tendencia a manipular enormes cantidades de datos se debe a la necesidad en muchos casos de incluir dicha información para la creación de informes estadísticos y modelos predictivos utilizados en diversas materias."[Wik17]

Uno de los puntos más importantes para entender la motivación por la cual se ha escogido dicha temática es el contexto tecnológico en que nos encontramos. Debido a la importante evolución que están sufriendo otras disciplinas dentro del mundo de la informática y las nuevas tecnologías, cada vez es más sencillo y económico recoger gran cantidad de información de cualquier proceso que se dé en la vida real. Esto se debe a una gran cantidad de factores, entre los que se destacan los siguientes:

- **Reducción de costes derivados de la recolección de información**: Debido a la constante evolución tecnológica cada vez es más barato disponer de mecanismos (tanto a nivel de hardware como de software), a partir de los cuales se puede recabar datos sobre un determinado suceso.

- **Mayor capacidad de cómputo y almacenamiento**: La recolección y manipulación de grandes cantidades de datos que se recogen a partir de sensores u otros métodos requieren por tanto del apoyo de altas capacidades de cómputo y almacenamiento. Las tendencias actuales se están apoyando en técnicas de virtualización que permiten gestionar sistemas de gran tamaño ubicados en distintas zonas geográficas como una unidad, lo cual proporciona grandes ventajas en cuanto a reducción de complejidad algorítmica a nivel de aplicación.

- **Mejora de las telecomunicaciones**: Uno de los factores que ha permitido una gran disminución de la problemática relacionada con la virtualización y su capacidad de respuesta ha sido el gran avance a nivel global que han sufrido las telecomunicaciones en los últimos años, permitiendo disminuir las barreras geográficas entre sistemas tecnológicos dispersos.

Gracias a este conjunto de mejoras se ha llegado al punto en que existe la oporturnidad de poder utilizar una gran cantidad de conocimiento, que individualmente o sin un apropiado procesamiento, carece de valor a nivel de información.

El tercer factor que es necesario tener en cuenta es la tendencia social actual, que cada vez más, está concienciada con el valor que tiene la información. Esto se ve reflejado en un amplio abanico de aspectos relacionados con el comportamiento de la población:

- **Monitorización de procesos laborales**: Muchas empresas están teniendo en cuenta la mejora de la productividad de sus empleados y máquinas. Por tanto, buscan nuevas técnicas que les permitan llevar a cabo dicha tarea. En los últimos años se ha dedicado mucho esfuerzo en implementar sistemas de monitorización que permitan obtener información para después procesarla y obtener resultados valiosos para dichas organizaciones.
- **Crecimiento exponencial de las plataformas de redes sociales**: La inherente naturaleza social del ser humano hace necesaria la expresión pública de sus sentimientos y acciones, lo cual, en el mundo de la tecnología se ha visto reflejado en un gran crecimiento de las plataformas de compartición de información así como de las de comunicación.
- **Iniciativas de datos abiertos por parte de las administraciones públicas**: Muchas insitituciones públicas están dedicando grandes esfuerzos en hacer visible la información que poseen, lo que conlleva una mejora social aumentando el grado de transparencia de las mismas, así como el nivel de conocimiento colectivo, que puede ser beneficioso tampo para ciudadanos como para empresas.

Como consecuencia de este cambio social, posiblemente propiciado por el avance tecnológico anteriormente citado, la población tiene un mayor grado de curiosidad por aspectos que antes no tenia la capacidad de entender, debido al nivel de complejidad derivado del tamaño de los conjuntos de muestra necesarios para obtener resultados fiables.

En este documento no se pretenden abordar temas relacionados con las técnicas utilizadas para recabar nuevos datos a partir de los ya existentes. A pesar de ello se realizará una breve introducción sobre dicho conjunto de estrategias, entre las que se encuentran: *Heurísticas*, *Regresión Lineal*, *Árboles de decisión*, *Máquinas de Vector Soporte (SVM)* o *Redes Neuronales Artificiales*.

Por contra, se pretende realizar un análisis acerca de los diferentes algoritmos necesarios para manejar dichas cantidades ingentes de información, en especial de su manipulación a nivel de operaciones básicas, como operaciones aritméticas, búsqueda o tratamiento de campos ausentes. Para ello, se tratará de acometer dicha problemática teniendo en cuenta estrategias de paralelización, que permitan aprovechar en mayor medida las capacidades de cómputo existentes en la actualidad.

Otro de los aspectos importantes en que se quiere orientar este trabajo es el factor dinámico necesario para entender la información, lo cual conlleva la búsqueda de nuevas estrategias algorítmicas de procesamiento en tiempo real. Por lo tanto, se pretende ilustrar un análisis acerca de las soluciones existentes en cada caso con respecto a la solución estática indicando las ventajas e inconvenientes de la versión dinámica según corresponda.

Índice general

Capítulo 1

Introducción

[TODO]

Capítulo 2

Algoritmos para Streaming

2.1. Introducción

[TODO Introducción a los algoritmos para streaming]

2.1.1. Computación en Tiempo Real

El primer concepto que se describe es **Computación en Tiempo Real**, que tal y cómo describen Shin y Ramanathan [SR94] se carácteriza por tres términos que se describen a continuación:

- **Tiempo***(time)*: En la disciplina de *Computación en Tiempo Real* el tiempo de ejecucción de una determinada tarea es especialmente crucial para garantizar el correcto desarrollo del cómputo, debido a que se asume un plazo de ejecucción permitido, a partir del cual la solución del problema deja de tener un valor óptimo. Shin y Ramanathan[SR94] diferencian entre tres categorías dentro de dicha restricción, a las cuales denominan *hard*, *firm* y *soft*, dependiendo del grado de relajación de la misma.

- **Confiabilidad***(correctness)*: Otro de los puntos cruciales en un sistema de *Cómputación en Tiempo Real* es la determinación de una unidad de medida o indicador acerca de las garantias de una determinada solución algorítmica para cumplir lo que promete de manera correcta en el tiempo esperado.

- **Entorno***(environment)*: El último factor que indican Shin y Ramanathan[SR94] para describir un sistema de *Computación en Tiempo Real* es el entorno del mismo, debido a que este condiciona el conjunto de tareas y la periodicidad en que se deben llevar a cabo. Debido a esta razón, realizan una diferenciación entre: *a*) tareas periódicas *periodic tasks* las cuales se realizan secuencialmente a partir de la finalización de una ventana de tiempo, y *b*) tareas no periódicas *periodic tasks* que se llevan a cabo debido al suceso de un determinado evento externo.

2.1.2. Problemas Dinámicos

Una vez completada la descripción acerca de lo que se puede definir como *Computación en Tiempo Real*, conviene realizar una descripción desde el punto de vista de la *teoría de complejidad computacional*. Para definir este tipo de problemas, se utiliza el término *problemas dinámicos*, los cuales consisten en aquellos en los cuales es necesario recalcular su solución conforme el tiempo avanza debido a variaciones en los parámetros de entrada del problema (Nótese que dicho término no debe confundirse con la estrategia de *programación dinámica* para el diseño de algoritmos).

Existen distintas vertientes dependiendo del punto de vista desde el que se estudien, tanto de la naturaleza del problema (soluciones dependientes temporalmente unas de otras o soluciones aisladas) como de los parámetros de entrada (entrada completa en cada nueva ejecución o variación respecto de la anterior). Los *Algoritmos para Streaming* están diseñados para resolver *problemas dinámicos*, por lo que en la sección 2.2, se describe en profundidad el modelo en que se enmarcan.

A continuación se indican los principales indicadores utilizados para describir la complejidad de una determinada solución algorítmica destinada a resolver un problema de dicha naturaleza:

- Espacio: Cantidad de espacio utilizado en memoria durante la ejecución del algoritmo.
- Inicialización: Tiempo necesario para la inicialización del algoritmo.
- Procesado: Tiempo necesario para procesar una determinada entrada.
- Consulta: Tiempo necesario para procesar la solución a partir de los datos de entrada procesados hasta el momento.

2.1.3. Algoritmos Online vs Algoritmos Offline

Una vez descrita la problemática de *Computación en Tiempo Real* en la sección 2.1.1 y la categoría de *Problemas Dinámicos* en la sección 2.1.2, en esta sección se pretende ilustrar la diferencia entre los *Algoritmos Online* y los *Algoritmos Offline*. Para ello, se ha seguido la diferenciación propuesta por Karp [Kar92], en la cual se plantea el problema de la siguiente manera (Se utilizará la misma notación que sigue Muthukrishnan[Mut05] para tratar mantener la consistencia durante todo el documento): Sea A el conjunto de datos o eventos de entrada, siendo cada $A[i]$ el elemento *i-ésimo* del conjunto. En el caso de los *Algoritmos Online* supondremos que es el elemento recibido en el instante i. A continuación se muestran las características de cada subgrupo:

- **Algoritmos Offline**: Esta categoría contiene todos los algoritmos que realizan el cómputo suponiendo el acceso a cualquier elemento del conjunto de datos A durante cualquier momento de su ejecución. Además, en esta categoría se impore la restricción de que el A debe ser invariante respecto del tiempo, lo que impone que para la adaptación del resultado a cambios, este tenga que realizar una nueva ejecución desde su estado inicial. Nótese que por tanto, dentro de este grupo se engloba la mayoría de algoritmos utilizados comunmente.
- **Algoritmos Online**: Son aquellos que calculan el resultado a partir de una secuencia de sucesos $A[i]$, los cuales generan un resultado dependiente del valor, y posiblemente de los sucedidos anteriormente. A partir de dicha estrategia, se añade una componente dinámica, la cual permite que tamaño del conjunto de datos de entrada A no tenga impuesta una restricción acerca de su longitud. Por contra, en este modelo no se permite conocer el cuceso $A[i+1]$ en el momento i. Esto encaja perfectamente en el modelo que se describirá en la sección 2.2.

Según la diferenciación que se acaba de describir, estas dos estrategias de diseño de algoritmos encajan en disciplinas distintas de diseño de algoritmos, teniendo una gran ventaja a nivel de eficiencia en el caso estático los *Algoritmos Offline*, pero quedando prácticamente inutilizables cuando la computación es en tiempo real, donde es mucho más apropiado el uso de estrategias de diseño de *Algoritmos Online*.

Como medida de eficiencia para los *Algoritmos Online*, Karp [Kar92] propone el **Ratio Competitivo**, el cual se define como la cota inferior del coste de cualquier nueva entrada con respecto de la que tiene menor coste. Sin embargo, dicha medida de eficiencia no es comúnmente utilizada en el caso de los *Algoritmos para Streaming* por la componente estocástica de los mismos, para los cuales son más apropiadas medidas probabilistas. A continuación se describen las ventajas de estos respecto de su vertiente estática.

2.1.4. Algoritmos Probabilistas

Los *Algoritmos Probabilistas* son una estrategia de diseño que emplea en un cierto grado de aleatoriedad en alguna parte de su lógica. Estos utilizan distribuciones uniformes de probabilidad para tratar de conseguir un incremento del rendimiento en su caso promedio. A continuación se describen los dos tipos de algoritmos probabilísticos según la clasificación realizada por Babai [Bab79]:

- **Algoritmos Las Vegas**: Devuelven un resultado incorrecto con una determinada probabilidad, pero avisan del resultado incorrecto cuando esto sucede. Para contrarrestrar este suceso basta llevar a cabo una nueva ejecución del algoritmo, lo cual tras un número indeterminado de ejecuciones produce un resultado válido.
- **Algoritmos Monte Carlo**: Fallan con un cierto grado de probabilidad, pero en este caso no avisan del resultado incorrecto. Por lo tando, lo único que se puede obtener al respecto es una indicador de la estimación del resultado correcto hacia la que converge tras varias ejecuciones. Además, se asegura una determinada cota del error ϵ, que se cumple con probabilidad δ.

La razón anecdótica por la cual Babai [Bab79] decidió denominar dichas categorías de algoritmos de esta manera se debe a lo siguiente (teniendo en cuenta el contexto de lengua inglesa): cuando se va a un casino en *Las Vegas* y se realiza una apuesta el *croupier* puede decir si se ha ganado o perdido porque habla el mismo idioma. Sin embargo, si sucede la misma situación en *Monte Carlo*, tan solo se puede conocer una medida de probabilidad debido a que en este caso el *croupier* no puede comunicarlo por la diferencia dialéctica.

2.1.5. Algoritmos Online Probabilistas vs Deterministas

La idea subyacente acerca del diseño de los *Algoritmos Online* es la mejora de eficiencia con respecto a sus homónimos estáticos cuando el conjunto de valores de entrada es dependiente de los resultados anteriores. Sin embargo, existen casos en que la frecuencia de ejecución del algoritmo, debido a una alta tasa de llegada de sucesos de entrada, las soluciones deterministas se convierten en alternativas poco escalables.

Dicha problemática se ha incrementado de manera exponencial debido al avance tecnológico y la gran cantidad de información que se está generando en la actualidad. Este fenómeno ha convertido en algo necesario el diseño de estrategias basadas en sucesos probabilísticos que reduzcen en gran medida el coste computacional eliminando el determinismo de la solución.

2.2. Modelo en Streaming

En esta sección se describen los aspectos formales del *Modelo en Streaming*. Para ello se ha seguido la representación definida por Muthukrishnan [Mut05]. Lo primero por tanto, es definir lo que es un flujo de datos o *Data Stream* como una "secuencia de señales digitalmente codificadas utilizadas para representar una transmisión de información" [Ins17]. Muthukrishnan [Mut05] hace una aclaración sobre dicha definición y añade la objeción de que los datos de entrada deben tener un ritmo elevado de llegada. Debido a esta razón existe complejidad a tres niveles:

- **Transmisión**: Ya que debido a la alta tasa de llegada es necesario diseñar un sistema de interconexiones que permita que no se produzcan congestiones debido a la obtención de los datos de entrada.
- **Computación**: Puesto que la tarea de procesar la gran cantidad de información que llega por unidad de tiempo produce cuellos de botella en el planteamiento. Por lo que es necesario implementar técnicas algorítmicas con un reducido nivel de complejidad computacional que para contrarrestar dicha problemática.

- **Almacenamiento**: Debido a la gran cantidad de datos que se presentan en la entrada, deben existir técnicas que permitan almacenar dicha información de manera eficiente. Esto puede ser visto desde dos puntos de vista diferences: *a*) tanto desde el punto de vista del espacio, tratando de minimizar el tamaño de los datos almacenados, maximizando la cantidad de información que se puede recuperar de ellos, *b*) como desde el punto de vista del tiempo necesario para realizar operaciones de búsqueda, addición, eliminación o edición.

[TODO ejemplo de los tres niveles de complejidad a partir del análisis meteorológico]

2.2.1. Formalismo para Streaming

Una vez descritos los niveles de complejidad a los que es necesario hacer frente para abordar problemas en el *Modelo en Streaming*, se realiza una descripción de los distintos modelos que propone Muthukrishnan [Mut05] en los apartados 2.2.2, 2.2.3 y 2.2.4. La especificación descrita en dichos apartados será seguida durante el resto del capítulo. Para ello nos basaremos en el siguiente formalismo:

Sea $a_1, a_2, ..., a_t, ...$ un flujo de datos de entrada (*Input Stream*), de tal manera que cada elemento debe presentar un orden de llegada secuencial respecto de $t \in \mathbb{M}$. Esto también se puede ver de la siguiente manera: el elemento siguiente a la llegada de a_{t-1} debe ser a_t y, por inducción, el próximo será a_{t+1}. Es necesario aclarar que t no se refiere a unidades propiamente temporales, sino a la posición en la entrada.

$$\boldsymbol{A}_t : [1...N] \rightarrow \mathbb{R}^2 \tag{2.1}$$

El siguiente paso para describir el formalismo es añadir la función $\boldsymbol{A}_t$, cuyo dominio e imagen se muestran en la ecuación (2.1). Esta función tiene distintas interpretaciones dependientes del *Modelo en Streaming* con el cual se esté tratando en cada caso, pero la idea subyacente se puede resumirse asumiendo que la primera componente almacena el valor, mientras que la segunda almacena el número de ocurrencias de dicho valor. Algo común a todos ellos es la variable t, que se corresponde con resultado de la función en el instante de tiempo t. Por motivos de claridad, en los casos en que nos estemos refiriendo un único momento, dicha variable será obviada en la notación.

2.2.2. Modelo de Serie Temporal

El *Modelo de Serie Temporal* o *Time Series Model* se refiere, tal y como indica su nombre, a una serie temporal, es decir, modeliza los valores que toma la variable i respecto de t, codificados en el modelo como $a_t = (i, 1)$. Nótese que se utiliza el valor 1 en la segunda componente de a_t, la razón de ello se debe a la definición de la imagen de $\boldsymbol{A}$ en la ecuación (2.1). A pesar de ello, dicho campo es irrelevante en este modelo, por lo que se podría haber escogido cualquier otro arbitrariamente. La razón por la cual se ha utilizado el valor 1 ha sido el refuerzo de la idea de que en este caso, el valor que toma la a_t en un determinado momento, no volverá a variar su valor, pero quedará obsoleto con la llegada de a_{t+1}.

El modelo se describe de manera matemática mediante la función $\boldsymbol{A}$, tal y como se ilustra en la ecuación (2.2). Textualmente, esto puede traducirse diciendo que la función $\boldsymbol{A}$ representa una estructura de datos que almacena el valor de todos los elementos recibidos en la entrada hasta el instante de tiempo t, es decir, actúa como un historial. Un ejemplo de este modelo son los valores en bolsa que toma una determinada empresa a lo largo del tiempo.

$$\boldsymbol{A}(t) = a_t \tag{2.2}$$

2.2.3. Modelo de Caja Registradora

El *Modelo de Caja Registradora* o *Cash Register Model* consiste en la recepción de incrementos de un determinado valor i. El nombre del modelo hace referencia al funcionamiento de una caja registradora (suponiendo que el pago se realiza de manera exacta), que recibe billetes o monedas de tipos diferentes de manera secuencial.

Para describir dicho modelo, previamente hay que realizar una aclaración acerca del contenido del elemento $a_t = (i, I_t)$, de manera que i representa el valor recibido, mientras que $I_t \geq 0$ indica el incremento en el instante t. Una vez aclarada esta definición, la función $\boldsymbol{A}_t$, se construye tal y como se indica en la ecuación (2.3).

$$\boldsymbol{A}_t(i) = A_{t-1}(i) + I_t \tag{2.3}$$

El *Modelo de Caja Registradora* es ampliamente utilizado en la formalización de problemas reales debido a que muchos fenómenos siguen esta estructura. Un ejemplo de ello es el conteo de accesos a un determinado sitio web, los cuales se corresponden con incrementos I_t, en este caso de caracter unitario realizados por un determinado usuario i en el momento t.

2.2.4. Modelo de Molinete

El *Modelo de Molinete* o *Turnstile Model* se corresponde con el caso más general, en el cual no solo se permiten incrementos, sino que también se pueden realizar decrementos en la cuenta. El nombre que se le dió a este modelo se debe al funcionamiento de los molinetes que hay en las estaciones de metro para permitir el paso a los usuarios, que en la entrada incrementan la cuenta del número de personas, mientras que en la salida los decrementan. La relajación originada por la capacidad de decremento ofrece una mayor versatilidad, que permite la contextualización de un gran número de problemas en este modelo. Por contra, añade un numerosas complicaciones a nivel computacional, tal y como se verá a lo largo del capítulo.

Tal y como ocurre en el caso anterior, para describir este modelo, lo primero es pensar en la estructura de los elementos en la entrada, que están formados por $a_t = (i, U_t)$, algo muy semejante a lo descrito en el *Modelo de Caja Registradora*. Sin embargo, en este caso U_t no tiene restricciones en su imagen, sino que puede tomar cualquier valor tanto positivo como negativo, lo cual añade el concepto de decremento. La construcción de la función $\boldsymbol{A}_t$ se describe en la ecuación (2.4).

$$\boldsymbol{A}_t(i) = A_{t-1}(i) + U_t \tag{2.4}$$

Muthukrishnan [Mut05] hace una diferenciación dentro de este modelo dependiendo del nivel de exigencia que se le pide al modelo, se dice que es un *Modelo de Molinete estricto* cuando se añade la restricción $\forall i, \forall t\ \boldsymbol{A}_t(i) \geq 0$, mientras que se dice que es un *Modelo de Molinete relajado* cuando dicha restricción no se tiene en cuenta.

Un ejemplo de este modelo es el conteo del número de usuarios que están visitando un determinado sitio web, tomando U_t el valor 1 en el caso de una nueva conexión y -1 en el caso de de una desconexión. En este ejemplo el valor i representa una determinada página dentro del sitio web.

2.3. Estructura básica

Puesto que la naturaleza intríseca de los *Algoritmos para Streaming* hace que procesen los elementos de entrada según van llegando, esto presenta peculiaridades con respecto a otras categorías algorítmicas utilizadas comúnmente.

Por tanto, primero se describirá la estructura básica que siguen los algoritmos más comunmente utilizados para después mostrar la estrategia seguida en el caso de Streaming.

Los algorítmos clásicamente estudiados para resolver la mayoría de problemas se basan la idea de funciones matemáticas. Es decir, se les presenta un conjunto de valores en la entrada, y a partir de ellos, realizan una determinada transformación sobre ellos, que genera como resultado una determinada salida. Nótese que esta idea no genera ninguna restricción acerca de lo que puede suceder en dicho proceso, es decir, no se restringe el uso de estructuras de datos auxiliares o técnicas similares.

Esta visión no se enmarca correctamente en el contexto de los *Algoritmos para Streaming*. La razón se debe a que la entrada no es propiamente un conjunto de datos, sino que se refiere a un flujo en sí mismo. Esta característica conlleva que en un gran número de ocasiones ya no sea necesario obtener los resultados tras cada llamada al algoritmo, ya que estos podrían carecer de interés o requerir un sobrecoste innecesario. Por lo tanto, el concepto de función matemática pierde el sentido en este caso, ya que estas exigen la existencia de un valor como resultado.

Un concepto más acertado para modelizar un *Algoritmo para Streaming* podría ser lo que en los lenguajes de programación derivados de *Fortran* se denomina subrutina, es decir, una secuencia de instrucciones que realizan una tarea encapsulada como una unidad. Sin embargo, para poder describir correctamente la estructura de un *Algoritmo para Streaming* hace falta algo más. La razón de ello es que a partir de dicho modelo de diseño no sería posible realizar peticiones acerca de lo calculado, es decir, sería una estrategia puramente procedural. Para corregir dicha problemática surge el concepto de *query* o pregunta. A través de dicha idea se pretende representar la manera de obtener un resultado a partir del cómputo realizado hasta el momento actual.

En resumen, con dicha estrategia de diseño se consigue separar la parte de procesado de la entrada con la parte de consulta del resultado, lo cual proporciona amplias ventajas para el modelo seguido por los *Algoritmos para Streaming*. Sin embargo, dicha estrategia produce un sobrecoste espacial con respecto del modelo de algoritmo clásico. Este se debe a la necesidad de mantener una estructura de datos en la cual se almacenen los resultados parciales referentes al flujo de entrada.

Los algoritmos *Algoritmos para Streaming* se componen por tanto de algoritmo de procesamiento del flujo de datos, una estructura de datos que almacena dichos resultados, y por último, un algoritmo de procesamiento de la *query* o pregunta necesaria para obtener los resultados requeridos. a continuación se dividen las fases para el funcionamiento de un algoritmo de dichas caraterísticas.

- **Inicialización**: En esta fase se llevan a cabo el conjunto de tareas necesarias para inicializar la estructura de datos que actuará como almacen de información durante el procesamiento del flujo de datos de entrada. Generalmente esto consite en el proceso de reservar memoria, inicializar a un valor por defecto la estructura de datos, etc. Sin embargo, existen técnicas sofisticadas que requieren de una mayor carga computacional en esta fase.

- **Procesado**: Se corresponde con el procesamiento del flujo de datos de manera secuencial. La idea subyacente en esta fase es la de realizar una determinada operación sobre la estructura de datos y el elemento de entrada actual, de manera que se lleve a cabo una actualización sobre la misma. Nótese en que la manera en que se manipula dicha estructura de datos condiciona en gran medida el conjunto de peticiones que se podrán realizar sobre ella.

- **Consulta**: La fase de consulta se caracteriza con respecto de la anterior por ser de carácter consultivo. Con esto nos estamos refiriendo a que dicha tarea no modifica el estado actual de la estructura de datos, sino que recoge información de la misma, que posiblemente transforme mediante alguna operación, para despues obtener un valor como resultado de dicha petición.

2.4. Medidas de Análisis y Conceptos Matemáticos

Los *Algoritmos para Streaming* se caracterizan por utilizar soluciones estadísticas en alguna parte (generalmente en el procesado) de su cómputo para obtener la solución con un menor coste computacional. En este caso, el coste que se pretende minimizar es el referido al espacio necesario para almacenar la estructura de datos auxiliar. Tal y como se ha dicho anteriormente, la razón de ello es debida a que se presupone un conjunto masivo de datos en la entrada, por lo que se pretende que el orden de complejidad espacial respecto de la misma sea de carácter sublinear ($o(N)$).

El objetivo es encontrar soluciones con un índice de error acotado que permitan llegar a la solución en un orden espacial de complejidad logarítmica ($O(log(N))$). Sin embargo, existen ocasiones en que no es posible llegar a una solución en dicho orden de complejidad, como es el caso de *Algoritmos para Streaming* aplicados a problemas de *Grafos*, en los cuales se relaja dicha restricción a un orden de complejiddad *poli-logarítmico* ($O(polylog(N))$).

El órden de complejidad *poli-logarítmico* engloba el conjunto de funciones cuyo orden de complejidad presenta un crecimiento acorde a una función polinomial formada logaritmos. Matemáticamente esto se modeliza a través de la ecuación (2.5)

$$a_k \log^k(N) + \cdots + a_1 \log(N) + a_0 = O(polylog(N)) \in o(N). \tag{2.5}$$

En esta sección se muestran distintas estrategias para poder llevar a cabo la demostración de pertenencia a un determinado orden de complejidad de un *Algoritmo para Streaming*. Debido a la elevada base estadística que requieren dichas demostraciones, a continuación se definen algunos conceptos básicos en relacionadas con estimadores estadísticos, para después realizar una breve demostración acerca de distintas cotas de concentración de valores en una distribución en las subsecciones 2.4.2, 2.4.3 y 2.4.4. Las definiciones que se exponen a continuación han sido extraidas de los apuntes del curso sobre *Randomized Algorithms* [Asp16] impartido por Aspnes en la *Universidad de Yale* así como las de la asignatura de *Estadística*[SJNSB16] impartida en el Grado de Ingeniería Informática de la *Universidad de Valladolid*.

2.4.1. Conceptos básicos de Estadística

Denotaremos como x_1 a una observación cualquiera contenida en el espacio de todas los posibles. Al conjunto de todas las observaciones posibles lo denotaremos como Ω y lo denominaremos espacio muestral, por lo tanto, $x \in \Omega$. Este concepto se puede entender de manera más sencilla mediante el siguiente ejemplo. Supongamos el lanzamiento de una moneda, que como resultado puede tomar los valores cara o cruz. Definiremos entonces $x_1 =$ cara y $x_2 =$ cruz como los sucesos posibles de lanzar una moneda. Por tanto el espacio Ω se define como $\Omega = \{x_1, x_2\} = \{\text{cara}, \text{cruz}\}$.

El siguiente paso es definir el concepto de **Variable Aleatoria**, que representa una función que mapea la realización de un determinado suceso sobre el espacio Ω. Dicha función se denota con letras mayúsculas y puesto que sus parámetros de entrada son desconocidos, estos se ignoran en la notación. Por tanto denotaremos las variables aleatorias como ($\boldsymbol{E}, \boldsymbol{X}, \boldsymbol{Y}$, etc.). Para la variable aleatoria $\boldsymbol{X}$, sean $x_1, x_2, ..., x_i, ...$ cada una de las observaciones posibles. Siguiendo el ejemplo anterior, se puede modelizar el lanzamiento de una moneda como X. Nótese por tanto, que una variable aleatoria puede definirse de manera textual como la modelización del resultado de un suceso *a-priori* desconocido.

Definiremos probabilidad como la medida de certidumbre asociada a un suceso o evento futuro, expresada como un valor contenido en el intervalo $[0, 1]$, tomando el valor 0 un suceso imposible y 1 un suceso seguro. La notación seguida para representar esto será $Pr[\boldsymbol{X} = x_i]$. Suponiendo la equiprobabilidad en el ejemplo de la moneda, podemos definir sus valores de probabilidad como $Pr[\boldsymbol{X} = \text{cara}] = \frac{1}{2}$ y $Pr[\boldsymbol{X} = \text{cruz}] = \frac{1}{2}$

Una vez descritos estos conceptos simples, a continuación hablaremos sobre distintos conceptos estadísticos utilizados en el análisis de algorítmos probabilisticos tales como *Esperanza*, *Varianza*, *Variables Independientes* y *Probabilidad Condicionada*.

Denominaremos **Esperanza Matemática** al valor medio o más probable que se espera que tome una determinada variable aleatoria. La modelización matemática de dicho concepto se muestra en la ecuación (2.6). Además, la esperanza matemática es de carácter lineal, por lo que se cumplen las ecuaciones (2.7) y (2.8)

$$\mathbb{E}[\boldsymbol{X}] = \sum_{i=1}^{\infty} x_i \cdot Pr[\boldsymbol{X} = x_i] \tag{2.6}$$

$$\mathbb{E}[c\boldsymbol{X}] = c\mathbb{E}[\boldsymbol{X}] \tag{2.7}$$

$$\mathbb{E}[\boldsymbol{X} + \boldsymbol{Y}] = \mathbb{E}[\boldsymbol{X}] + \mathbb{E}[\boldsymbol{Y}] \tag{2.8}$$

La **Varianza** se define como una medida de dispersión de una variable aleatoria. Dicho estimador representa el error cuadrático respecto de la esperanza. Su modelización matemática se muestra en la ecuación (2.9). Aplicando propiedades algebraicas se puede demostrar la veracidad de las propiedades descritas en las ecuaciones (2.10) y (2.11).

$$Var[\boldsymbol{X}] = \mathbb{E}[(\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}])^2] \tag{2.9}$$

$$Var[\boldsymbol{X}] = \mathbb{E}[\boldsymbol{X}^2] - \mathbb{E}^2[\boldsymbol{X}] \tag{2.10}$$

$$Var[c\boldsymbol{X}] = c^2 Var[\boldsymbol{X}] \tag{2.11}$$

A continuación se describe el concepto de **Independencia** entre dos variables aleatorias $\boldsymbol{X}, \boldsymbol{Y}$. Se dice que dos variables son independientes cuando los sucesos de cada una de ellas no están condicionados por los de otras. Esto puede verse a como el cumplimiento de la igualdad de la ecuación (2.12).

$$Pr[\boldsymbol{X} = x \cap \boldsymbol{Y} = y] = Pr[\boldsymbol{X} = x] \cdot Pr[\boldsymbol{Y} = y] \tag{2.12}$$

Cuando nos referimos al concepto de independencia referido a un conjunto n variables aleatorias $\boldsymbol{X_1}, \boldsymbol{X_2}, ..., \boldsymbol{X_n}$ lo denominaremos **Independencia Mutua**, que impone la restricción descrita en la ecuación (2.13).

$$Pr\left[\bigcap_{i=1}^{n} \boldsymbol{X_i} = x_i\right] = \prod_{i=1}^{n} Pr[\boldsymbol{X_i} = x_i] \tag{2.13}$$

También es de especial interés en el campo de los algoritmos probabilísticos el caso de la **k-independencia** sobre un conjunto de n variables aleatorias $\boldsymbol{X_1}, \boldsymbol{X_2}, ..., \boldsymbol{X_n}$. Dicha idea se puede resumir como la independencia de todas las variables en grupos de k variables. Este concepto tiene mucha importancia en el ámbito de los *Sketches*, tal y como se verá en la sección **??**. El caso más simple es para $k = 2$, el cual se denomina **independencia pareada**, cuya modelización matemática se muestra en la ecuación (2.14).

$$\forall i, \forall j \; Pr[\boldsymbol{X_i} = x_i \cap \boldsymbol{X_j} = x_j] = Pr[\boldsymbol{X_i} = x_i] \cdot Pr[\boldsymbol{X_j} = x_j] \tag{2.14}$$

Desde el punto de vista de conjuntos de n variables aleatorias $\boldsymbol{X_1}, \boldsymbol{X_2}, ..., \boldsymbol{X_n}$, existen distintas propiedades de linearidad que se cumplen entre ellas a nivel del cálculo de la *Esperanza* y la *Varianza*. En el caso de la *Esperanza*, la linealidad respecto de la suma (ecuación (2.15)) se cumple para variables dependientes e independientes. Sin embargo,

en el caso de la *Varianza*, la linealidad respecto de la suma (ecuación (2.16)) se cumple tan solo para variables **independientes pareadas**.

$$\mathbb{E}\left[\sum_{i=1}^{n} \boldsymbol{X_i}\right] = \sum_{i=1}^{n} \mathbb{E}[\boldsymbol{X_i}] \tag{2.15}$$

$$Var\left[\sum_{i=1}^{n} \boldsymbol{X_i}\right] = \sum_{i=1}^{n} Var[\boldsymbol{X_i}] \tag{2.16}$$

La **Probabilidad Condicionada** entre dos variables aleatorias $\boldsymbol{E_1}$ y $\boldsymbol{E_2}$ se puede definir como la medida de verosimilitud de la ocurrencia del suceso $\boldsymbol{E_1}$ sabiendo que ya ha ocurrido $\boldsymbol{E_2}$. Esto se puede modelizar matemáticamente tal y como se muestra en la ecuación (2.17).

$$Pr[\boldsymbol{E_1}|\boldsymbol{E_2}] = \frac{Pr[\boldsymbol{E_1} \cap \boldsymbol{E_2}]}{Pr[\boldsymbol{E_1}]} \tag{2.17}$$

En el caso de la **Probabilidad Condicionada** sobre variables independientes, surge la propiedad descrita en la ecuación (2.18). Es fácil entender la razón, que se apoya en la idea de que si dos variables aleatorias no guardan relación, entonces la ocurrencia de una de ellas, no condicionará el resultado de la otra.

$$Pr[\boldsymbol{X_1} = x_1|\boldsymbol{X_2} = x_2] = \frac{Pr[\boldsymbol{X_1} = x_1 \cap \boldsymbol{X_2} = x_2]}{Pr[\boldsymbol{X_1} = x_1]} = \frac{Pr[\boldsymbol{X_1} = x_1 \cdot \boldsymbol{X_2} = x_2]}{Pr[\boldsymbol{X_1} = x_1]} = Pr[\boldsymbol{X_2} = x_2] \tag{2.18}$$

Una vez descritos los conceptos estadísticos básicos para el análisis de algoritmos probabilísticos, lo siguiente es realizar una exposición acerca de las distintas cotas de concentración de valores, lo cual permite obtener resultados aproximados acerca de los resultados esperados por dichos algoritmos, así como sus niveles de complejidad. Primero se describirá *Desigualdad de Boole*, para después tratar las desigualdades de *Markov*(2.4.2), *Chebyshev*(2.4.3) y *Chernoff* (2.4.4)

La **Desigualdad de Boole** consiste en una propiedad básica que indica que la probabilidad de que se cumpla la ocurrencia de un suceso es menor o igual que ocurrencia de la suma de todas ellas. Esto se modeliza matemáticamente en la ecuación (2.19).

$$Pr\left[\bigcup_{i=1}^{n} \boldsymbol{E_i}\right] \leq \sum_{i=1}^{n} Pr[\boldsymbol{E_i}] \tag{2.19}$$

2.4.2. Desigualdad de Markov

La *Desigualdad de Markov* es la técnica base que utilizan otras desigualdades más sofisticadas para tratar de acotar la *Esperanza* de una determinada *Variable Aleatoria*. Proporciona una cota superior de probabilidad respecto de la *Esperanza* tal y como se muestra en la ecuación (2.20). Tal y como se puede intuir, dicha cota es muy poco ajustada, sin embargo, presenta una propiedad muy interesante como estructura base. Sea $f : \boldsymbol{X} \to \mathbb{R}^+$ una función positiva, entonces también se cumple la desigualdad de la ecuación (2.21). El punto interesante surge cuando se escoge la función f de tal manera que sea estrictamente creciente, entonces se cumple la propiedad descrita ecuación (2.22), a través de la cual podemos obtener cotas mucho más ajustadas. Dicha idea se apoya en la *Desigualdad de Jensen*.

$$\forall \lambda \geq 0,\ Pr[\boldsymbol{X} \geq \lambda] \leq \frac{\mathbb{E}[\boldsymbol{X}]}{\lambda} \tag{2.20}$$

$$\forall \lambda \geq 0,\ Pr[f(\boldsymbol{X}) \geq f(\lambda)] \leq \frac{\mathbb{E}[f(\boldsymbol{X})]}{f(\lambda)} \tag{2.21}$$

$$\forall \lambda \geq 0,\ Pr[\boldsymbol{X} \geq \lambda] = Pr[f(\boldsymbol{X}) \geq f(\lambda)] \leq \frac{\mathbb{E}[f(\boldsymbol{X})]}{f(\lambda)} \tag{2.22}$$

2.4.3. Desigualdad de Chebyshev

La *Desigualdad de Chebyshev* utiliza la técnica descrita en la subsección anterior apoyandose en al idea de la función f para obtener una cota de concentración mucho más ajustada basandose en la *Varianza*. Dicha propiedad se muestra en la ecuación (2.23). En este caso se utiliza $f(\boldsymbol{X}) = \boldsymbol{X}^2$, que es estrictamente creciente en el dominio de aplicación de $\boldsymbol{X}$. Además, se selecciona como variable aleatoria $|\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}]$, es decir, el error absoluto de una $\boldsymbol{X}$ respecto de su valor esperado. La demostración de esta idea se muestra en la ecuación (2.24).

$$\forall \lambda \geq 0,\ Pr[|\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}]| \geq \lambda] \leq \frac{Var[\boldsymbol{X}]}{\lambda^2} \tag{2.23}$$

$$\forall \lambda \geq 0,\ Pr[|\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}]| \geq \lambda] = Pr[(\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}])^2 \geq \lambda^2] \leq \frac{\mathbb{E}[(\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}])^2]}{\lambda^2} = \frac{Var[\boldsymbol{X}]}{\lambda^2} \tag{2.24}$$

2.4.4. Desigualdad de Chernoff

En este apartado se realiza una descripción acerca de la *Desigualdad de Chernoff*. Dicha descripción ha sido extraida de los apuntes de la asignatura de Algoritmos Probabilísticos (*Randomized Algorithms*) [Cha04] impartida por Shuchi Chawla en la *Carnegie Mellon University* de Pennsylvania.

La *Desigualdad de Chernoff* proporciona cotas mucho más ajustadas que por contra, exigen unas presunciones más restrictivas para poder ser utilizada. La variable aleatoria en este caso debe ser de la forma $\boldsymbol{S} = \sum_{i=1}^{n} \boldsymbol{X_i}$ donde cada $\boldsymbol{X_i}$ es una variable aleatoria uniformemente distribuida e independiente del resto. También describiremos la esperanza de cada una de las variables $\boldsymbol{X_i}$ como $\mathbb{E}[\boldsymbol{X}] = p_i$.

Denotaremos como μ a la esperanza de $\boldsymbol{S}$, tal y como se describe en la ecuación (2.25). También se define la función f como $f(\boldsymbol{S}) = e^{t\boldsymbol{S}}$.

$$\mu = \mathbb{E}\left[\sum_{i=1}^{n} \boldsymbol{X_i}\right] = \sum_{i=1}^{n} \mathbb{E}[\boldsymbol{X_i}] = \sum_{i=1}^{n} p_i \tag{2.25}$$

El siguiente paso es utilizar la ecuación (2.22) de la *Desigualdad de Markov* con la función f, que en este caso es posible puesto que es estrictamente creciente. En este caso en lugar de utilizar λ como constante, se prefiere $\delta \in [0, 1]$, que se relacionada con la anterior de la siguiente manera: $\lambda = (1 + \delta)$. Entonces la ecuación (2.26) muestra el paso inicial para llegar a la *Desigualdad de Chernoff*.

$$Pr[\boldsymbol{X} > (1+\delta)\mu] = Pr[e^{t\mathbb{X}} > e^{(1+\delta)t\mu}] \leq \frac{\mathbb{E}[e^{t\boldsymbol{X}}]}{e^{(1+\delta)t\mu}} \tag{2.26}$$

Aplicando operaciones aritméticas y otras propiedades estadísticas, se puede demostrar la veracidad de las ecuaciones (2.27) y (2.28), que proporcionan cotas mucho muy ajustadas de concentración de la distribución de una *variable aleatoria* formada por la suma de n variables aleatorias independientes uniformemente distribuidas.

$$\forall \delta \geq 0,\ Pr[\boldsymbol{X} \geq (1+\delta)\mu] \leq e^{\frac{-\lambda^2\mu}{2+\lambda}} \tag{2.27}$$

$$\forall \delta \geq 0,\ Pr[\boldsymbol{X} \leq (1-\delta)\mu] \leq e^{\frac{-\lambda^2\mu}{2+\lambda}} \tag{2.28}$$

2.4.5. Funciones Hash

[TODO]

Una vez descritos los conceptos básicos acerca de lo que son los *Algoritmos para Streaming* y las bases estadísticas necesarias para poder entender el funcionamiento de los mismo y entender sus niveles de complejidad así como de precisión, en las siguientes secciones se realizará una descripción sobre algunos de los algoritmos más relevantes en esta area. En especial se explica el algoritmo de *Morris* en la sección 2.6, el de *Flajolet-Martin* en la 2.6 y por último se hablará de la *Estimación de Momentos de Frecuencia* en el modelo en streaming en la sección 2.7.

2.5. Algoritmo de Morris

El *Algoritmo de Morris* fue presentado por primera vez en el artículo *Counting Large Numbers of Events in Small Registers* [Mor78] redactado por *Robert Morris*. En dicho documento se trata de encontrar una solución al problema de conteo de ocurrencias de un determinado suceso teniendo en cuenta las restriciones de espacio debido a la elevada tasa de ocurencias que se da en muchos fenómenos. El problema del conteo (**Count Problem**) de ocurrencias también se denomina el momento de frecuencia F_1 tal y como se verá en la sección 2.7.

Por tanto, *Morris* propone realizar una estimación de dicha tasa para reducir el espacio necesario para almacenar el valor. Intuitivamente, a partir dicha restricción se consigue un orden de complejidad espacial sublineal ($o(N)$) con respecto al número de ocurrencias. Se puede decir que el artículo publicado por *Morris* marcó el punto de comienzo de este área de investigación. El conteo probabilista es algo trivial si se restringe a la condición de incrementar el conteo de ocurrencias siguiendo una distribución de *Bernoulli* con un parámetro p prefijado previamente. Con esto se consigue un error absoluto relativamente pequeño con respecto al valor p escogido. Sin embargo, el error cuando el número de ocurrencias es pequeño es muy elevado.

Para solucionar dicha problemática y conseguir una cota del error relativo reducida, la solución propuesta por *Morris* se basa en la selección del parámetro p variable con respecto al número de ocurrencias, con lo cual se consigue que la decisión de incrementar el contador sea muy probable en los primeros casos, lo cual eliminar dicho problema. *Morris* propone aumentar el contador X con probabilidad $\frac{1}{2^X}$. Tras n ocurrencias, el resultado que devuelve dicho algoritmo es $\widetilde{n} = 2^X - 1$. El pseudocódigo se muestra en el algoritmo 1.

Algorithm 1: Morris-Algorithm

Result: $\widetilde{n} = 2^X - 1$
$X \leftarrow 0$;
for *cada evento* **do**
  $X \leftarrow X + 1$ con probabilidad $\frac{1}{2^X}$;
end

A continuación se realiza un análisis de la solución. Esta ha sido extraida de los apuntes de la asignatura *Algorithms for Big Data* [Nel15] impartida por *Jelani Nelson* en la *Universidad de Harvard*. Denotaremos por X_n el valor del contador X tras n ocurrencias. Entonces se cumplen las igualdades descritas en las ecuaciones (2.29) y (2.30). Esto se puede demostrar mediante técnicas inductivas sobre n.

$$\mathbb{E}[2^{X_n}] = n + 1 \tag{2.29}$$

$$\mathbb{E}[2^{2X_n}] = \frac{3}{2}n^2 + \frac{3}{2}n + 1 \tag{2.30}$$

Por la *Desigualdad de Chebyshev* podemos acotar el error cometido tras n repeticiones, dicha formulación se muestra en la ecuación (2.31).

$$Pr[|\widetilde{n} - n| > \epsilon n] < \frac{1}{\epsilon^2 n^2} \cdot \mathbb{E}[\widetilde{n} - n]^2 = \frac{1}{\epsilon^2 n^2} \cdot \mathbb{E}[2^X - 1 - n]^2 \tag{2.31}$$

$$= \frac{1}{\epsilon^2 n^2} \cdot \frac{n^2}{2} \tag{2.32}$$

$$= \frac{1}{2\epsilon^2} \tag{2.33}$$

La ventaja de esta estrategia algorítmica con respecto de la trivial es la cota del error relativo producido en cada iteracción del algoritmos, lo cual aporta una mayor genericidad debido a que esta se mantiene constante con respecto del número de ocurrencias. Sin embargo, se han propuesto otras soluciones para tratar de reducir más dicha cota. El algoritmo *Morris+* se basa en el mantenimiento de s copias independientes de *Morris* para después devolver la media del resultado de cada una de ellas. A partir de dicha solución se consigue lo descrito en la ecuación (2.34).

$$Pr[|\widetilde{n} - n| > \epsilon n] < \frac{1}{2s\epsilon^2} \qquad s > \frac{3}{2\epsilon^2} \tag{2.34}$$

2.6. Algoritmo de Flajolet-Martin

En esta sección se describe el *Algoritmo de Flajolet-Martin*, cuya descripción aparece en el artículo *Probabilistic Counting Algorithms for Data Base Applications* [FM85] redactado por *Philippe Flajolet* y *G. Nigel Martin*. En este caso, la problemática que se pretende resolver no es el número de ocurrencias de un determinado suceso, sino el número de sucesos distintos (**Count Distinct Problem**) en la entrada. Este problema también se denomina el momento de frecuencia F_0 tal y como se verá en la sección 2.7. Al igual que en el caso del algoritmo de *Morris*, se apoya en estrategias probabilistas para ajustarse a un orden de complejidad espacial de carácter sublineal ($o(N)$)) manteniendo una cota de error ajustada.

La intuicción a partir de la cual se basa el *Algoritmo de Flajolet-Martin*, es la transformación de los elementos de entrada sobre una *función Hash* universal binaria con distribución uniforme e independiente de probabilidad. La propiedad de distribución uniforme permite entonces preveer que la mitad de los elementos tendrán un 1 en el bit menos significativo, que una cuarta parte de los elementos tendrán un 1 en el segundo bit menos significativo y así sucesivamente. Por tanto, a partir de esta idea se puede realizar una aproximación estadística del número de elementos distintos que han sido presentados en la entrada. Requiere de L bits de espacio para el almacenamiento del número de elementos distintos. Por la notación descrita en anteriores secciones $L = log(n)$, donde n es el número máximo de elementos distintos en la entrada. A continuación se explica esta estrategia, para ello nos apoyaremos en las siguientes funciones:

- $hash(x)$ Es la función hash con distribución uniforme e independiente de probabilidad que mapea una entrada cualquiera a un valor entero en el rango $[0, ..., 2^L - 1]$.
- $bit(y, k)$ Esta función devuelve el bit *k-ésimo* de la representación binaria de y, de tal manera que se cumple que $y = \sum_{k \geq 0} bit(y, k)2^k$
- $\rho(y)$ La función ρ devuelve la posición en la cual se encuentra el bit con valor 1 empezando a contar a partir del menos significativo. Por convenio, devuelve el valor L si y no contiene ningún 1 en su representación binaria, es decir, si $y = 0$. Esto se modeliza matemáticamente en la ecuación (2.35).

$$\rho(y) = \begin{cases} min_{k\geq 0} bit(y,k) \neq 0 & y \geq 0 \\ L & y = 0 \end{cases} \tag{2.35}$$

Flajolet y *Martin* se apoyan en una estructura de datos indexada a la cual denominan **BITMAP**, de tamaño $[0...L-1]$ la cual almacena valores binarios $\{0,1\}$ y se inicializa con todos los valores a 0. Nótese por tanto, que esta estructura de datos puede ser codificada como un string binario de longitud L. La idea del algoritmo es marcar con un 1 la posición $\mathbf{BITMAP}[\rho(hash(x))]$. Seguidamente, queda definir el resultado de la consulta sobre cuántos elementos distintos han aparecido en el Stream de datos de entrada. Para ello se calcula $2^{\rho(\mathbf{BITMAP})}$. El pseudocódigo se muestra en el algoritmo 2.

Algorithm 2: FM-Algorithm

Result: $2^{\rho(\mathbf{BITMAP})}$
for $i \in [0, ..., L-1]$ **do**
  $\mathbf{BITMAP}[i] \leftarrow 0$;
end
for *cada evento* **do**
  if $\boldsymbol{BITMAP[\rho(hash(x))] = 0}$ **then**
    $\mathbf{BITMAP}[\rho(hash(x))] \leftarrow 1$;
  end
end

El análisis de esta solución ha sido estraido de los apuntes del libro *Mining of massive datasets* [LRU14] de la *Universidad de Cambridge*. En este caso lo representa teniendo el cuenta el número de 0's seguidos en la parte menos significativa de la representación binaria de $h(x)$. Nótese que esto es equivalente al valor de la función $\rho(h(y))$, por tanto, adapteremos dicho análisis a la solución inicial propuesta por *Flajolet* y *Martin*. La probabilidad de que se cumpla $\rho(h(x)) = r$ es 2^{-r}. Supongamos que el número de elementos distintos en el stream es m. Entonces la probabilidad de que ninguno de ellos cumpla $\rho(h(x)) = r$ es al menos $(1-2^{-r})^m$ lo cual puede ser reescrito como $((1-2^{-r})^{2^r})^{m2^{-r}}$. Para valores suficientemente grandes r se puede asumir que dicho valor es de la forma $(1-\epsilon)^{1/\epsilon} \approx 1/\epsilon$. Entonces la probabilidad de que no se cumpla que $\rho(h(x)) = r$ cuando han aparecido m elementos distintos en el stream es de $e^{-m2^{-r}}$

La problemática de este algoritmo deriva de la suposición de la capacidad de generación de claves Hash totalmente aleatorias, lo cual no se ha conseguido aún. Por lo tanto posteriormente, *Flajolet* ha seguido trabajando el problema de conteo de elementos distintos en *Loglog counting of large cardinalities* [DF03] y *Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm* [FFGM07]. En el artículo *An optimal algorithm for the distinct elements problem* [KNW10] *Daniel Kane y otros* muestran un algoritmo óptimo para el problema. Los resultados de dichos trabajos se discuten en la sección 3.7 por su cercana relación con las *estructuras de datos de resumen*.

2.7. Aproximación a los Momentos de Frecuencia

La siguiente idea de la que es interesante hablar para terminar la introducción a los *Algoritmos para Streaming* son los *Momentos de Frecuencia*. Una generalización de los conceptos del número de elementos distintos (F_0) y el conteo de elementos (F_1) que se puede extender a cualquier F_k para $k \geq 0$. La definición matemática del momento de frecuencia k-ésimo se muestra en la ecuación (2.36). Nótese el caso especial de F_∞ que se muestra en la ecuación

(2.37) y se corresponde con el elemento más veces común en el *Stream*. Estas ideas han sido extraidas del documento *Frequency Moments* [Woo09] redactado por *David Woodruff*.

$$F_k = \sum_{i=1}^{n} m_i^k \tag{2.36}$$

$$F_\infty = max_{1 \leq i \leq n} m_i \tag{2.37}$$

El resto de la sección trata sobre la exposición de los algoritmos para el cálculo de los momentos de frecuencia descritos en el articulo *The space complexity of approximating the frequency moments* [AMS96] redactado por *Noga Alon*, *Yossi Matias* y *Mario Szegedy*, por el cual fueron galardonados con el premio *Gödel* en el año 2005. En dicho trabajo además de presentar *Algoritmos para Streaming* para el cálculo de F_k (cabe destacar su solución para F_2), también presentan cotas inferiores para el problema de los *Momentos de Frecuencia*. Posteriormente *Piotr Indyk* y *David Woodruff* encontraron un algoritmo óptimo para el problema de los*Momentos de Frecuencia* tal y como exponen en *Optimal Approximations of the Frequency Moments of Data Streams* [IW05]. A continuación se discuten los resultados de dichos trabajos.

Para el cálculo de F_k para $k \geq 0$ *Alon*, *Matias* y *Szedgedy* proponen un enfoque similar a los propuestos en algoritmos anteriores (la definición de una variable aleatoria X tal que $\mathbb{E}[X] = F_k$). Sin embargo, la novedad en este caso es que su algoritmo no está restringido a un k concreto, sino que en su caso es generalizable para cualquier entero positivo, sin embargo, en este caso la exposición es a nivel teórico. La definición del algoritmo a nivel práctico se describe en la sección 3.6.

Definiremos las constantes $S_1 = O(n^{1-1/k}/\lambda^2)$ y $S_2 = O(\log(1/\varepsilon))$. El algoritmo utiliza S_2 variables aleatorias denominadas Y_1, Y_2, Y_{S_2} y devuelve la mediana de estas denominandola Y. Cada una de estas variables Y_i está formada por la media de X_{ij} variables aleatorias tales que $1leqjleqS_1$. La forma en que se actualiza el estado del algoritmo tras cada nueva llegada se apoya en el uso de S_2 funciones hash uniformemente distribuidas e independientes entre si que mapean cada símbolo a un determinado indice j.

Para el análisis del algoritmos supondremos que el tamaño n del *Stream* es conocido *a-priori*. La demostración se apoya en una variable aleatoria X construida de la siguiente manera:

- Seleccionaremos de manera aleatoria el elemento $a_{p \in (1,2,...,m)}$ del Stream, siendo $a_p = l \in (1, 2, ...n)$. Es decir, el elemento procesado en el momento p representa la llegada del símbolo l.
- Definiremos $r = |\{q : q \geq p, a_p = l\}|$ como el número de ocurrencias del símbolo l hasta el momento p.
- La variable aleatoria X se define como $X = m(r^k - (r-1)^k)$

Desarrollando la Esperanza Matemática de la variable aleatoria X se puede demostrar que esta tiende al momento de frecuencia k tal y como se muestra en la ecuación (2.38).

$$\begin{aligned}
\mathbb{E}(X) &= \textstyle\sum_{i=1}^{n} \sum_{i=1}^{m_i} (j^k - (j-1)^k) \\
&= \tfrac{m}{m}[(1^k + (2^k - 1^k) + \ldots + (m_1^k - (m_1 - 1)^k)) \\
&\quad + (1^k + (2^k - 1^k) + \ldots + (m_2^k - (m_2 - 1)^k)) + \ldots \\
&\quad + (1^k + (2^k - 1^k) + \ldots + (m_n^k - (m_n - 1)^k))] \\
&= \textstyle\sum_{i=1}^{n} m_i^k = F_k
\end{aligned} \tag{2.38}$$

En cuanto al coste espacial del algoritmo, se puede demostrar tal y como indican *Alon*, *Matias* y *Szedgedy* en su artículo original [AMS96] que este sigue el orden descrito en la ecuación (2.39) puesto que es necesario almacenar a_p y r lo cual requiere de $log(n) + log(m)$ bits de memoria, además de $S_1 x S_2$ variables aleatorias para mantener X.

$$O\left(\frac{k\log\frac{1}{\varepsilon}}{\lambda^2}n^{1-\frac{1}{k}}\left(\log n+\log m\right)\right) \tag{2.39}$$

[TODO comentario final]

Capítulo 3

Estructuras de Datos de Resumen

3.1. Introducción

[TODO]

3.2. Tipos de Estructuras de Datos de Resumen

[TODO]

3.3. Sketching

[TODO]

3.4. Count-Min Sketch

[TODO]

3.5. Count Sketch

[TODO]

3.6. AMS Sketch

[TODO]

3.7. HyperLogLog

[TODO]

Capítulo 4

Algoritmos para Grafos

4.1. Introducción

[TODO]

4.2. Modelo en Semi-Streaming

[TODO]

Capítulo 5

Reducción de la Dimensionalidad

5.1. Introducción

[TODO]

5.2. Teorema de Johnson-Lindenstrauss

[TODO]

5.3. Busqueda de Vecinos más Cercanos

[TODO]

Capítulo 6

Técnicas de Minería de Datos

6.1. Introducción

[TODO]

6.2. Aprendizaje Supervisado y No Supervisado

[TODO]

6.3. Árboles de Decisión

[TODO]

6.4. Regresión Lineal

[TODO]

6.5. Redes Neuronales

[TODO]

6.6. Manifold Learning

[TODO]

Capítulo 7

Paralelización a Gran Escala

7.1. Introducción

[TODO]

7.2. Sistemas de Ficheros Distribuidos

[TODO]

7.3. Modelo de acceso a Memoria

[TODO]

7.4. Complejidad de la Comunicación

[TODO]

7.5. MapReduce

[TODO]

Apéndice A

Metodología de Trabajo

Apéndice B

¿Cómo ha sido generado este documento?

Bibliografía

[AMS96] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29. ACM, 1996.

[Asp16] James Aspnes. Notes on randomized algorithms cpsc 469/569. Fall 2016.

[Bab79] László Babai. Monte-carlo algorithms in graph isomorphism testing, 1979.

[Cha04] Shuchi Chawla. Notes on randomized algorithms: Chernoff bounds. October 2004.

[DF03] Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities. In *European Symposium on Algorithms*, pages 605–617. Springer, 2003.

[FFGM07] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Analysis of Algorithms 2007 (AofA07)*, pages 127–146, 2007.

[FM85] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[Ins17] Institute for Telecommunication Sciences. Definitions: Data stream, March 2017. `https://www.its.bldrdoc.gov/fs-1037/dir-010/_1451.htm`.

[IW05] Piotr Indyk and David Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 202–208. ACM, 2005.

[Kar92] Richard M. Karp. On-line algorithms versus off-line algorithms: How much is it worth to know the future? pages 416–429, 1992.

[KNW10] Daniel M Kane, Jelani Nelson, and David P Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 41–52. ACM, 2010.

[LRU14] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2014.

[Mor78] Robert Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978.

[Mut05] S. Muthukrishnan. Data streams: Algorithms and applications. *Found. Trends Theor. Comput. Sci.*, 1(2):117–236, August 2005.

[Nel15] Jelani Nelson. Algorithms for big data. Fall 2015.

[SJNSB16] Luis Augusto San José Nieto and Araceli Suárez Barrio. Grado en ingeniería informática: Estadística. 2016.

[SR94] K. G. Shin and P. Ramanathan. Real-time computing: a new discipline of computer science and engineering. *Proceedings of the IEEE*, 82(1):6–24, Jan 1994.

[Wik17] Wikipedia. Big Data, February 2017.

[Woo09] David Woodruff. Frequency moments. *Encyclopedia of Database Systems*, pages 1169–1170, 2009.