



---

**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 comprensión 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 evaluación. 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 oportunidad 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 insituciones 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 tanto 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 tenía 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

<b>Prefacio</b>	<b>1</b>
<b>1. Introducción</b>	<b>5</b>
<b>2. Algoritmos para Streaming</b>	<b>7</b>
2.1. Introducción . . . . .	7
2.2. Modelo en Streaming . . . . .	9
2.3. Estructura básica . . . . .	11
2.4. Medidas de Análisis y Conceptos Matemáticos . . . . .	13
2.5. Algoritmo de Morris . . . . .	17
2.6. Algoritmo de Flajolet-Martin . . . . .	18
2.7. Aproximación a los Momentos de Frecuencia . . . . .	19
2.8. Conclusiones . . . . .	21
<b>3. Estructuras de Datos de Resumen</b>	<b>23</b>
3.1. Introducción . . . . .	23
3.2. Tipos de Estructuras de Datos de Resumen . . . . .	24
3.3. Bloom Filter . . . . .	30
3.4. Count-Min Sketch . . . . .	30
3.5. Count Sketch . . . . .	30
3.6. AMS Sketch . . . . .	30
3.7. HyperLogLog . . . . .	30
<b>4. Algoritmos para Grafos</b>	<b>31</b>
4.1. Introducción . . . . .	31
4.2. Modelo en Semi-Streaming . . . . .	31
<b>5. Reducción de la Dimensionalidad</b>	<b>33</b>
5.1. Introducción . . . . .	33
5.2. Teorema de Johnson-Lindenstrauss . . . . .	33
5.3. Búsqueda de Vecinos más Cercanos . . . . .	33
<b>6. Técnicas de Minería de Datos</b>	<b>35</b>
6.1. Introducción . . . . .	35
6.2. Aprendizaje Supervisado y No Supervisado . . . . .	35
6.3. Árboles de Decisión . . . . .	35
6.4. Regresión Lineal . . . . .	35
6.5. Redes Neuronales . . . . .	35
6.6. Manifold Learning . . . . .	35

<b>7. Paralelización a Gran Escala</b>	<b>37</b>
7.1. Introducción . . . . .	37
7.2. Sistemas de Ficheros Distribuidos . . . . .	37
7.3. Modelo de acceso a Memoria . . . . .	37
7.4. Complejidad de la Comunicación . . . . .	37
7.5. MapReduce . . . . .	37
<b>A. Metodología de Trabajo</b>	<b>39</b>
<b>B. ¿Cómo ha sido generado este documento?</b>	<b>41</b>
<b>Bibliografía</b>	<b>42</b>



# 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 del que hablaremos es **Computación en Tiempo Real**, que tal y cómo describen Shin y Ramanathan [SR94] se caracteriza por tres términos que se exponen a continuación:

- **Tiempo**(*time*): En la disciplina de *Computación en Tiempo Real* el tiempo de ejecución de una determinada tarea es especialmente crucial para garantizar el correcto desarrollo del cómputo, debido a que se asume un plazo de ejecución permitido, a partir del cual la solución del problema deja de tener validez. 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 *Computación en Tiempo Real* es la determinación de una unidad de medida o indicador acerca de las garantías 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. Por 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 *aperiodic 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, y que 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 impone la restricción de que el  $A$  debe ser invariante respecto del tiempo, lo que conlleva que para la adaptación del resultado a cambios en la entrada, este tenga que realizar una nueva ejecución desde su estado inicial. Nótese por tanto, que 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 de la entrada actual, y posiblemente de las anteriores. A partir de dicha estrategia, se añade una componente dinámica, la cual permite que el tamaño del conjunto de datos de entrada  $A$  no tenga impuesta una restricción acerca de su longitud *a-priori*. Por contra, en este modelo no se permite conocer el suceso  $A[i + 1]$  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 indicar, estas dos estrategias de diseño de algoritmos encajan en disciplinas distintas, 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 determinista.

### 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 contrarrestar este suceso basta con 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 tanto, lo único que se puede obtener al respecto es un indicador de la estimación del resultado correcto hacia la que converge tras varias ejecuciones. Además, se asegura una determinada cota del error  $\epsilon$ , que se cumple con probabilidad  $\delta$ .

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 de 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 valores en la 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 genera en la actualidad, que sigue creciendo a un ritmo desorbitado. Este fenómeno ha convertido en algo necesario el diseño de estrategias basadas en técnicas probabilísticas que reduzcan en gran medida el coste computacional que como consecuencia eliminan 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 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 cálculo de la solución por lo que es necesario implementar técnicas algorítmicas con un reducido nivel de complejidad computacional 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 diferentes: *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.. Además, se debe prestar especial atención en la información que se almacena, tratando de reducirla al máximo prescindiendo de datos redundantes o irrelevantes.

### 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  $a_1, a_2, \dots, a_t, \dots$  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.

$$\mathbf{A}_t : [1 \dots N] \rightarrow \mathbb{R}^2 \quad (2.1)$$

El siguiente paso para describir el formalismo es añadir la función  $\mathbf{A}_t$ , cuyo dominio e imagen se muestran en la ecuación (2.1). Esta función tiene distintas interpretaciones dependientes del *Modelo en Streaming* bajo el cual se esté trabajando en cada caso, pero la idea subyacente 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  $\mathbf{A}$  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  $a_t$  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  $\mathbf{A}$ , tal y como se ilustra en la ecuación (2.2). Textualmente, esto puede traducirse diciendo que la función  $\mathbf{A}$  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.

$$\mathbf{A}(t) = a_t \quad (2.2)$$

### 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  $\mathbf{A}_t$ , se construye tal y como se indica en la ecuación (2.3).

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

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 carácter 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.

Al igual que 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  $\mathbf{A}_t$  se describe en la ecuación (2.4).

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

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  $\forall i, \forall t \mathbf{A}_t(i) \geq 0$ , 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 intrínseca de los *Algoritmos para Streaming* hace que procesen los elementos de entrada según van llegando, esto presenta peculiaridades con respecto de otras categorías algorítmicas más asentadas y utilizadas

en la actualidad. 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 algoritmos clásicamente estudiados para resolver la mayoría de problemas se basan en 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 los datos, que genera como resultado una salida. Nótese que esta idea no impone 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 tiene como consecuencia 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 sobre el resultado calculado, es decir, sería una estrategia puramente procedural. Para corregir dicha problemática surge el concepto de consulta o *query*. 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 procesamiento de la entrada de 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 consulta o *query* necesaria para obtener los resultados requeridos. a continuación se dividen las fases para el funcionamiento de un algoritmo de dichas características.

- **Inicialización:** En esta fase se llevan a cabo el conjunto de tareas necesarias para inicializar la estructura de datos que actuará como almacén de información durante el procesamiento del flujo de datos de entrada. Generalmente esto consiste en el proceso de reservar memoria, inicializar a un valor por defecto la estructura de datos, etc. Sin embargo, existen técnicas más 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 diferencia 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 transforma mediante alguna operación, para después obtener un valor como resultado de dicha petición.



## 2.4. Medidas de Análisis y Conceptos Matemáticos

Los *Algoritmos para Streaming* se caracterizan por utilizar propiedades estadísticas en alguna parte (generalmente en el procesamiento) 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 sublineal ( $o(N)$ ).

El objetivo es encontrar soluciones con un intervalo 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 complejidad *poli-logarítmico* ( $O(\text{polylog}(N))$ ).

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

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

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 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 extraídas 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 las posibles. Al conjunto de todas las observaciones posibles lo denotaremos como  $\Omega$  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  $x_1 = \text{cara}$  y  $x_2 = \text{cruz}$  como los sucesos posibles de lanzar una moneda. Por tanto el espacio  $\Omega$  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  $\Omega$ . 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  $(\mathbf{E}, \mathbf{X}, \mathbf{Y}, \text{etc.})$ . Para la variable aleatoria  $\mathbf{X}$ , sean  $x_1, x_2, \dots, x_i, \dots$  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  $Pr[\mathbf{X} = x_i]$ . Suponiendo la equiprobabilidad en el ejemplo de la moneda, podemos definir sus valores de probabilidad como  $Pr[\mathbf{X} = \text{cara}] = \frac{1}{2}$  y  $Pr[\mathbf{X} = \text{cruz}] = \frac{1}{2}$

Una vez descritos estos conceptos simples, a continuación hablaremos sobre distintos conceptos estadísticos utilizados en el análisis de algoritmos probabilísticos 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)

$$\mathbb{E}[\mathbf{X}] = \sum_{i=1}^{\infty} x_i \cdot Pr[\mathbf{X} = x_i] \quad (2.6)$$

$$\mathbb{E}[c\mathbf{X}] = c\mathbb{E}[\mathbf{X}] \quad (2.7)$$

$$\mathbb{E}[\mathbf{X} + \mathbf{Y}] = \mathbb{E}[\mathbf{X}] + \mathbb{E}[\mathbf{Y}] \quad (2.8)$$

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).

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

$$Var[\mathbf{X}] = \mathbb{E}[\mathbf{X}^2] - \mathbb{E}^2[\mathbf{X}] \quad (2.10)$$

$$Var[c\mathbf{X}] = c^2 Var[\mathbf{X}] \quad (2.11)$$

A continuación se describe el concepto de **Independencia** entre dos variables aleatorias  $\mathbf{X}, \mathbf{Y}$ . 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).

$$Pr[\mathbf{X} = x \cap \mathbf{Y} = y] = Pr[\mathbf{X} = x] \cdot Pr[\mathbf{Y} = y] \quad (2.12)$$

Cuando nos referimos al concepto de independencia referido a un conjunto  $n$  variables aleatorias  $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n$  lo denominaremos **Independencia Mutua**, que impone la restricción descrita en la ecuación (2.13).

$$Pr\left[\bigcap_{i=1}^n \mathbf{X}_i = x_i\right] = \prod_{i=1}^n Pr[\mathbf{X}_i = x_i] \quad (2.13)$$

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  $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n$ . 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 3.2.4. El caso más simple es para  $k = 2$ , el cual se denomina **independencia pareada**, cuya modelización matemática se muestra en la ecuación (2.14).

$$\forall i, \forall j \ Pr[\mathbf{X}_i = x_i \cap \mathbf{X}_j = x_j] = Pr[\mathbf{X}_i = x_i] \cdot Pr[\mathbf{X}_j = x_j] \quad (2.14)$$

Desde el punto de vista de conjuntos de  $n$  variables aleatorias  $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n$ , existen distintas propiedades de linealidad 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**.

$$\mathbb{E}\left[\sum_{i=1}^n \mathbf{X}_i\right] = \sum_{i=1}^n \mathbb{E}[\mathbf{X}_i] \quad (2.15)$$

$$Var\left[\sum_{i=1}^n \mathbf{X}_i\right] = \sum_{i=1}^n Var[\mathbf{X}_i] \quad (2.16)$$

La **Probabilidad Condicionada** entre dos variables aleatorias  $\mathbf{E}_1$  y  $\mathbf{E}_2$  se puede definir como la medida de verosimilitud de la ocurrencia del suceso  $\mathbf{E}_1$  sabiendo que ya ha ocurrido  $\mathbf{E}_2$ . Esto se puede modelizar matemáticamente tal y como se muestra en la ecuación (2.17).

$$Pr[\mathbf{E}_1|\mathbf{E}_2] = \frac{Pr[\mathbf{E}_1 \cap \mathbf{E}_2]}{Pr[\mathbf{E}_2]} \quad (2.17)$$

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.

$$Pr[\mathbf{X}_1 = x_1 | \mathbf{X}_2 = x_2] = \frac{Pr[\mathbf{X}_1 = x_1 \cap \mathbf{X}_2 = x_2]}{Pr[\mathbf{X}_2 = x_2]} = \frac{Pr[\mathbf{X}_1 = x_1] \cdot Pr[\mathbf{X}_2 = x_2]}{Pr[\mathbf{X}_2 = x_2]} = Pr[\mathbf{X}_1 = x_1] \quad (2.18)$$

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).

$$Pr\left[\bigcup_{i=1}^n \mathbf{E}_i\right] \leq \sum_{i=1}^n Pr[\mathbf{E}_i] \quad (2.19)$$

### 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  $f : \mathbf{X} \rightarrow \mathbb{R}^+$  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*.

$$\forall \lambda \geq 0, Pr[\mathbf{X} \geq \lambda] \leq \frac{\mathbb{E}[\mathbf{X}]}{\lambda} \quad (2.20)$$

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

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

### 2.4.3. Desigualdad de Chebyshev

La *Desigualdad de Chebyshev* utiliza la técnica descrita en la subsección anterior apoyandose en la 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  $f(\mathbf{X}) = \mathbf{X}^2$ , que es estrictamente creciente en el dominio de aplicación de  $\mathbf{X}$ . Además, se selecciona como variable aleatoria  $|\mathbf{X} - \mathbb{E}[\mathbf{X}]|$ , es decir, el error absoluto de una  $\mathbf{X}$  respecto de su valor esperado. La demostración de esta idea se muestra en la ecuación (2.24).

$$\forall \lambda \geq 0, \Pr[|\mathbf{X} - \mathbb{E}[\mathbf{X}]| \geq \lambda] \leq \frac{\text{Var}[\mathbf{X}]}{\lambda^2} \quad (2.23)$$

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

### 2.4.4. Desigualdad de Chernoff

En este apartado se realiza una descripción acerca de la *Desigualdad de Chernoff*. Dicha descripción ha sido extraída 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  $\mathbf{S} = \sum_{i=1}^n \mathbf{X}_i$  donde cada  $\mathbf{X}_i$  es una variable aleatoria uniformemente distribuida e independiente del resto. También describiremos la esperanza de cada una de las variables  $\mathbf{X}_i$  como  $\mathbb{E}[\mathbf{X}_i] = p_i$ .

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

$$\mu = \mathbb{E}\left[\sum_{i=1}^n \mathbf{X}_i\right] = \sum_{i=1}^n \mathbb{E}[\mathbf{X}_i] = \sum_{i=1}^n p_i \quad (2.25)$$

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  $\lambda$  como constante, se prefiere  $\delta \in [0, 1]$ , que se relaciona con la anterior de la siguiente manera:  $\lambda = (1 + \delta)\mu$ . Entonces la ecuación (2.26) muestra el paso inicial para llegar a la *Desigualdad de Chernoff*.

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

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

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

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

### 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 sus niveles de complejidad así como de precisión, en las siguientes secciones se realiza una descripción sobre algunos de los algoritmos más relevantes en este área. 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 restricciones 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 de partir dicha restricción se consigue un orden de complejidad espacial sublineal ( $o(N)$ ) con respecto del 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 relativo que se obtiene cuando el número de ocurrencias es pequeño es muy elevado, lo cual lo convierte en una solución impracticable.

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 del número de ocurrencias, con lo cual se consigue que la decisión de incrementar el contador sea muy probable en los primeros casos, lo cual elimina el problema del error relativo. *Morris* propone aumentar el contador  $X$  con probabilidad  $\frac{1}{2^X}$ . Tras  $n$  ocurrencias, el resultado que devuelve dicho algoritmo es  $\tilde{n} = 2^X - 1$ . El pseudocódigo se muestra en el algoritmo 1.

---

**Algorithm 1:** Morris-Algorithm

---

**Result:**  $\tilde{n} = 2^X - 1$   
 $X \leftarrow 0$ ;  
**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 extraída 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  $n$ .

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

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

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

$$Pr[|\tilde{n} - n| > \epsilon n] < \frac{1}{\epsilon^2 n^2} \cdot \mathbb{E}[\tilde{n} - n]^2 = \frac{1}{\epsilon^2 n^2} \cdot \mathbb{E}[2^X - 1 - n]^2 \quad (2.31)$$

$$= \frac{1}{\epsilon^2 n^2} \cdot \frac{n^2}{2} \quad (2.32)$$

$$= \frac{1}{2\epsilon^2} \quad (2.33)$$

La ventaja de esta estrategia algorítmica con respecto de la trivial es la cota del error relativo producido en cada iteración del algoritmo, 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 en mayor medida dicha cota de error. 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 esta estrategia se consiguen las tasas de error que se indican en la ecuación (2.34).

$$Pr[|\tilde{n} - n| > \epsilon n] < \frac{1}{2s\epsilon^2} \quad s > \frac{3}{2\epsilon^2} \quad (2.34)$$

## 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 conoce como el cálculo del 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 intuición en 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 prever 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 probabilista 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  $[0, \dots, 2^L - 1]$ .
- $bit(y, k)$  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  $\rho$  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).

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

*Flajolet y Martin* se apoyan en una estructura de datos indexada a la cual denominan **BITMAP**, de tamaño  $[0 \dots L - 1]$  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 **BITMAP** $[\rho(\text{hash}(x))]$ . Seguidamente, queda definir el resultado de la consulta sobre cuántos elementos distintos han aparecido en el flujo de datos de entrada. Para ello se calcula  $2^{\rho(\text{BITMAP})}$ . El pseudocódigo se muestra en el algoritmo 2.

---

**Algorithm 2:** FM-Algorithm

---

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

---

El análisis de esta solución ha sido extraído de los apuntes del libro *Mining of massive datasets* [LRU14] de la *Universidad de Cambridge*. En este caso lo representa teniendo en 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, adaptaremos 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 en la actualidad. 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] para tratar de mejorar el grado de precisión de su estrategia de conteo. 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  $F_\infty$  que se muestra en la ecuación

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

$$F_k = \sum_{i=1}^n m_i^k \quad (2.36)$$

$$F_\infty = \max_{1 \leq i \leq n} m_i \quad (2.37)$$

El resto de la sección trata sobre la exposición de los algoritmos para el cálculo de los momentos de frecuencia descritos en el artículo *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 *Szegedy* 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  $1 \leq j \leq S_1$ . 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 índice  $j$ .

Para el análisis del algoritmo 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  $a_{p \in (1, 2, \dots, m)}$  del *Stream*, siendo  $a_p = l \in (1, 2, \dots, n)$ . 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).

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

En cuanto al coste espacial del algoritmo, se puede demostrar tal y como indican *Alon*, *Matias* y *Szegedy* 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  $S_1 \times S_2$  variables aleatorias para mantener  $X$ .



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

## 2.8. Conclusiones

[TODO comentario final]



## Capítulo 3

# Estructuras de Datos de Resumen

### 3.1. Introducción

El gran crecimiento tecnológico que se está llevando a cabo en la actualidad a todos los niveles está propiciando además un aumento exponencial en cuanto a la cantidad de información que se genera. La reducción de costes en cuanto a la instalación de sensores que permiten recoger información de muchos procesos productivos, así como la obtención de metadatos a partir del uso de internet y las redes sociales por parte de los usuarios hace que el ritmo de crecimiento en cuanto a información generada por unidad de tiempo haya crecido a un gran ritmo.

Una de las razones que han facilitado dicha tendencia es la disminución de costes de almacenamiento de información a la vez que las capacidades de cómputo necesarias para procesar dicha información han aumentado. Sin embargo, debido al crecimiento exponencial en cuanto al tamaño del conjunto de datos, es necesario investigar nuevas técnicas y estrategias que permitan obtener respuestas satisfactorias basadas en la gran cantidad de información de la que se dispone en un tiempo razonable.

Tradicionalmente, la investigación en el campo de las *bases de datos* se ha centrado en obtener respuestas exactas a distintas consultas, tratando de hacerlo de la manera más eficiente posible, así como de tratar de reducir el espacio necesario para almacenar la información. *Acharya y otros* proponen en el artículo *Join synopses for approximate query answering* [AGPR99] el concepto de *Approximate Query Processing*. Dicha idea se expone en la subsección 3.1.1.

#### 3.1.1. Approximate Query Processing

El *procesamiento aproximado de consultas*, (*Approximate Query Processing* o **AQP**) se presenta como una estrategia de consulta basada en conceptos y propiedades estadísticas que permiten una gran reducción de la complejidad computacional y espacial necesaria para la resolución de consultas a una base de datos. Por contra, dicha reducción a nivel de complejidad tiene como consecuencia la inserción de un determinado nivel de imprecisión en el resultado a la cual denominaremos tasa de error. Se pretende que dicha tasa de error pueda ser acotada en una desviación máxima determinada por  $\epsilon$  y se cumpla con un índice de probabilidad  $\delta$ . Al igual que en capítulos anteriores, en este caso también se presta especial importancia en la minimización del error relativo lo cual consigue que las soluciones mediante el *procesamiento aproximado de consultas* sean válidas tanto para consultas de tamaño reducido como de gran tamaño.

Durante el resto del capítulo se describen y analizan distintas estrategias que permiten llevar a cabo implementaciones basadas en *procesamiento aproximado de consultas* centrando especial atención en los *Sketches* por su similitud con el *Modelo en Streaming* descrito en el capítulo 2. En la sección 3.2 se realiza una descripción a partir de la cual se pretende aclarar las diferencias entre las distintas *estructuras de datos de resumen*. Posteriormente, en la sección ?? se

explican en detalle las cualidades de las estrategias basadas en *Sketching*. En las secciones 3.4, 3.5, 3.6 y 3.7 se habla de *Count-Min Sketch*, *Count Sketch*, *AMS Sketch* e *HyperLogLog* respectivamente.

## 3.2. Tipos de Estructuras de Datos de Resumen

Para el diseño de soluciones basadas en *procesamiento aproximado de consultas* en bases de datos existen distintas estrategias, las cuales presentan distintas ventajas e inconvenientes tal y como se pretende mostrar en esta sección. Dichas descripciones han sido extraídas del libro *Synopses for massive data* [CGHJ12] redactado por *Cormode y otros*. En las secciones 3.2.1, 3.2.2, 3.2.3 y 3.2.4 se habla de *Sampling*, *Histogram*, *Wavelet* y *Sketches* respectivamente.

### 3.2.1. Sampling

El *Sampling* o *muestreo* es la estrategia más consolidada entre las que se presentan. Las razones se deben a su simplicidad conceptual así como su extendido uso en el mundo de la estadística. Uno de los primeros artículos en que se trata el muestreo aplicado a bases de datos es *Accurate estimation of the number of tuples satisfying a condition* [PSC84] redactado por *Piatetsky-Shapiro y Connell*. La intuición en que se basa dicha estrategia es la selección de un subconjunto de elementos denominado *muestra* de entre el conjunto global al cual se denomina *población*. Una vez obtenida la *muestra* del conjunto de datos global cuyo tamaño es significativamente menor (lo cual reduce drásticamente el coste computacional), se realizan los cálculos que se pretendía realizar sobre toda la *población*, a partir de los cuales se obtiene un estimador del valor real que habría sido obtenido al realizarlos sobre el conjunto de datos global.

Para que las estrategias de sumariación de información obtengan resultados válidos o significativos respecto del conjunto de datos, es necesario que se escojan adecuadamente las instancias de la *muestra*, de manera que represente de manera fiel la información global. Para llevar a cabo dicha labor existen distintas estrategias, desde las más simples basadas en la selección aleatoria sin reemplazamiento como otras mucho más sofisticadas basadas en el mantenimiento de *muestras* estratificadas. Sea  $R$  la población y  $|R|$  el tamaño de la misma. Denominaremos  $t_j$  al valor  $j$ -ésimo de la población y  $X_j$  al número de ocurrencias del mismo en la *muestra*. A continuación se describen distintas técnicas de muestreo:

- **Selección Aleatoria Sin Reemplazamiento:** Consiste en la estrategia más simple de generación de *muestras*. Se basa en la selección aleatoria de un valor entero  $r$  en el rango  $[1, |R|]$  para después añadir el elemento localizado en la posición  $r$  de la *población* al subconjunto de *muestra*. Después repetir dicha secuencia durante  $n$  veces para generar una *muestra* de tamaño  $n$ . El estimador para la operación *SUMA* se muestra en la ecuación (3.1) además de la desviación de dicho estimador en la ecuación (3.2).

$$Y = \frac{|R|}{n} \sum_j X_j t_j \quad (3.1)$$

$$\sigma^2(Y) = \frac{|R|^2 \sigma^2(R)}{n} \quad (3.2)$$

- **Selección Aleatoria Con Reemplazamiento:** En este caso se supone que la selección de una instancia de la población tan solo se puede llevar a cabo una única vez como mucho, por lo tanto se cumple que  $\forall X_j \in 0, 1$ . La selección se lleva a cabo de la siguiente manera: se genera de manera aleatoria un valor entero  $r$  en el rango  $[1, |R|]$  para después añadir el elemento localizado en la posición  $r$  de la *población* al subconjunto de *muestra* si este no ha sido añadido ya, sino volver a generar otro valor  $r$ . Después repetir dicha secuencia durante  $n$  veces para generar una *muestra* de tamaño  $n$ . Al igual que en la estrategia anterior, en este caso también se muestra el estimador para la operación *SUMA* en la ecuación (3.3). Nótese que el cálculo es el mismo que en el caso de

la estrategia sin reemplazamiento. Sin embargo, la varianza obtenida a partir de dicha estrategia es menor tal y como se muestra en la ecuación (3.4).

$$Y = \frac{|R|}{n} \sum_j X_j t_j \quad (3.3)$$

$$\sigma^2(Y) = \frac{|R|(|R| - n)\sigma^2(R)}{n} \quad (3.4)$$

- **Bernoulli y Poisson:** Mediante esta alternativa de muestreo se sigue una estrategia completamente distinta a las anteriores. En lugar de seleccionar la siguiente instancia aleatoriamente de entre todas las posibles, se decide generar  $|R|$  valores aleatorios  $r_j$  independientes en el intervalo  $[0, 1]$  de tal manera que si  $r_j$  es menor que un valor  $p_j$  fijado a priori, la instancia se añade al conjunto de *muestra*. Cuando se cumple que  $\forall i, j \ p_i = p_j$  se dice que es un muestreo de *Bernoulli*, mientras que cuando no se cumple dicha condición se habla de muestreo de *Poisson*. El cálculo de la *SUMA* en este caso es muy diferente de los anteriores tal y como se muestra en la ecuación (3.5). La desviación de este estimador se muestra en la ecuación (3.6), que en general presenta peores resultados (mayor desviación) que las anteriores alternativas, sin embargo, esta alternativa posee la cualidad de aplicar distintos pesos a cada instancia de la población, lo que puede traducirse en que una selección adecuada de dichos valores  $p_j$  puede mejorar significativamente dichos resultados.

$$Y = \sum_{i \in \text{muestra}} \frac{t_i}{p_i} \quad (3.5)$$

$$\sigma^2(Y) = \sum_i \left( \frac{1}{p_i} - 1 \right) t_i^2 \quad (3.6)$$

- **Muestreo Estratificado:** El muestreo estratificado trata de minimizar al máximo las diferencias entre la distribución del conjunto de datos de la *población* de la *muestra* que se pretende generar. Para ello existen distintas alternativas entre las que se encuentra una selección que actualiza los pesos  $p_j$  tras cada iteración, lo que reduce la desviación de la *muestra*, sin embargo produce un elevado coste computacional en su generación. Por lo tanto se proponen otras estrategia más intuitiva basada en la partición del conjunto de datos de la *población* en subconjuntos disjuntos con varianza mínima entre las instancias que contienen a los cuales se denomina *estratos*. Posteriormente se selecciona mediante cualquiera de los métodos anteriores una *muestra* de cada *estrato*, lo cual reduce en gran medida la desviación típica global del estimador.

La estrategia de sumariación de información mediante *muestreo* tiene como ventajas la independencia de la complejidad con respecto a la dimensionalidad de los datos (algo que como se verá a continuación no sucede con otras alternativas) además de su simplicidad conceptual. También existen cotas de error para las consultas, para las cuales no ofrece restricciones en cuanto al tipo (debido a que se realizan sobre un subconjunto con la misma estructura que el global). El muestre es apropiado para conocer información general acerca del conjunto de datos que cada instancia del mismo posee. Además, presenta la cualidad de permitir la modificación en tiempo real, es decir, se pueden añadir o eliminar nuevas instancias a la muestra conforme se añaden o eliminan del conjunto de datos global.

Sin embargo, en entornos donde el ratio de adiciones/eliminaciones es muy elevado el coste del mantenimiento de la muestra puede hacerse impracticable. El *muestreo* es una buena alternativa para conjuntos de datos homogéneos en los cuales la presencia de valores atípicos es irrelevante. Tampoco obtiene buenos resultados en consultas relacionadas con el conteo de elementos distintos. En las siguientes secciones se describen alternativas que resuelven estas dificultades y limitaciones.

### 3.2.2. Histogram

Los *histogramas* son estructuras de datos utilizadas para sumarizar grandes conjuntos de datos, pero por contra, tienen un enfoque completamente diferente al que siguen las estrategias de *muestreo* de la sección anterior. En este caso, el concepto es similar a la visión estadística de los histogramas. Consiste en dividir el dominio de valores que pueden tomar las instancias del conjunto de datos de tal manera que se mantiene un conteo del número de instancias pertenecientes a cada partición.

Durante el resto de la sección se describen de manera resumida distintas estrategias de estimación del valor de las particiones así como las distintas estrategias de particionamiento del conjunto de datos. Para llevar a cabo dicha labor es necesario describir la notación que se seguirá: Sea  $D$  el conjunto de datos e  $i \in [1, M]$  cada una de las categorías de los mismos. Denotaremos por  $g(i)$  el número de ocurrencias del valor  $i$ . Para referirnos a cada uno de las particiones utilizaremos la notación  $S_j$  para  $j \in [1, B]$ . Nótese por tanto que  $M$  representa el número de categorías distintas mientras que  $B$  es el número de particiones utilizadas para “comprimir” los datos. La mejora de eficiencia en cuanto a espacio se consigue debido a la presuposición de que  $B \ll M$

Cuando se habla de *esquemas de estimación* se trata de describir la manera en que se almacena o trata el contenido de cada una de las particiones  $S_j$  del histograma. La razón por la cual este es un factor importante a la hora de caracterizar un histograma es debida a que está altamente ligada a la precisión del mismo.

- **Esquema Uniforme:** Los esquemas que presuponen distribución uniforme se subdividen en dos categorías:  
a) *continous-value asumption* que presupone que todas las categorías  $i$  contenidas en la partición  $S_j$  presentan el mismo valor para la función  $g(i)$  y b) *uniform-spread asumption* que presupone que el número de ocurrencias de la partición  $S_j$  se localiza distribuido uniformemente al igual que en el caso anterior, pero en este caso entre los elementos de un subconjunto  $P_j$  generado iterando con un determinado desplazamiento  $k$  sobre las categorías  $i$  contenidas en  $S_j$ . El segundo enfoque presenta mejores resultados en el caso de consultas de cuantiles que se distribuyen sobre más de una partición  $S_j$
- **Esquema Basado en Splines:** En la estrategia basada en splines se presupone que los valores se distribuyen conforme una determinada función lineal de la forma  $y_j = a_j x_j + b_j$  en cada partición  $S_j$  de tal manera que el conjunto total de datos  $D$  puede verse como una función continua a trozos. Nótese que en este caso se habla de una función lineal, sin embargo puede generalizarse a funciones no lineales.
- **Esquema Basado en Árboles:** Consiste en el almacenamiento de las frecuencias de cada partición  $S_j$  en forma de árbol binario, lo cual permite seleccionar de manera apropiada el nivel del árbol que reduzca el número de operaciones necesarias para obtener la estimación del número de ocurrencias según el tamaño rango de la consulta. La razón por la cual se elige un árbol binario es debida a que se puede reducir en un orden de 2 el espacio necesario para almacenar dichos valores manteniendo únicamente los de una de las ramas del mismo. La razón de ello es debida a que se puede calcular el valor de la otra mediante una resta sobre el valor almacenado en el nodo padre y la rama que si contiene el valor.
- **Esquema Heterogéneo:** El esquema heterogéneo se basa la intuición de que la distribución de frecuencias de cada una de las particiones  $S_j$  no es uniforme, por lo tanto sigue un enfoque diferente en cada una de ellas tratanto de minimizar al máximo la tasa de error producida. Para ello existen distintas heurísticas basadas en distancias o teoría de la información entre otros.

Una vez descritas distintas estrategias de estimación del valor de frecuencias de una determinada partición  $S_j$ , el siguiente paso para describir un *histograma* es realizar una descripción acerca de las distintas formas de generación de dichas particiones. Para tratar de ajustarse de manera más adecuada a la distribución de los datos, se realiza un *muestreo* con el cual se generan las particiones. A continuación se describen las técnicas más comunes para dicha labor:

- **Particionamiento Heurístico:** Dichas estrategias de particionamiento se basan en distintas heurísticas que en la práctica han demostrado comportamientos aceptables en cuanto a resultados a nivel de precisión, sin embargo, no proporcionan ninguna garantía a nivel de optimalidad. Su uso está ampliamente extendido debido al reducido coste computacional. Dentro de esta categoría las heurísticas más populares son las siguientes:
  - **Equi-Width:** Consiste en la división del dominio de categorías  $[1, M]$  en particiones equi-espaciadas unas de otras. Para dicha estrategia tan solo es necesario conocer *a-priori* el rango del posible conjunto de valores. Es la solución con menor coste computacional, a pesar de ello sus resultados a nivel práctico son similares a otras estrategias más sofisticadas cuando la distribución de frecuencias es uniforme.
  - **Equi-Depth:** Esta estrategia de particionamiento requiere conocer la distribución de frecuencias *a-priori* (o aproximarla a partir de algún método de muestreo). Se basa en la división del dominio de valores de tal manera que las particiones tengan la misma frecuencia. Para ello se crean particiones de tamaños diferentes.
  - **Singleton-Bucket:** Para tratar de mejorar la precisión esta estrategia de particionamiento se basa en la utilización de dos particiones especiales, las cuales contienen las categorías de mayor y menor frecuencia respectivamente para después cubrir el resto de categorías restante mediante otra estrategia (generalmente *equi-depth*).
  - **Maxdiff:** En este caso, el método de particionamiento se apoya en la idea de utilizar los puntos de mayor variación de frecuencias mediante la medida  $|g(i + 1) - g(i)|$ , para dividir el conjunto de categorías en sus respectivas particiones, de tal manera que las frecuencias contenidas en cada partición sean lo más homogéneas posibles.
- **Particionamiento con Garantías de Optimalidad:** En esta categoría se enmarcan las estrategias de generación de particiones que ofrecen garantías de optimalidad a nivel de la precisión de resultados en las consultas. Para ello se apoyan en técnicas de *Programación Dinámica* (DP), de tal manera que la selección de las particiones se presenta como un problema de *Optimización*. Sin embargo, dichas estrategias presentan un elevado coste computacional que muchas veces no es admisible debido al gran tamaño del conjunto de datos que se pretende sumarizar. Como solución ante dicha problemática se han propuesto distintas estrategias que se basan en la resolución del problema de optimización, pero sobre una *muestra* del conjunto de datos global, lo cual anula las garantías de optimalidad pero si se escoge de manera adecuada ofrece una buena aproximación hacia ellas.
- **Particionamiento Jerárquico:** Las estrategias de particionamiento jerárquico se basan en la utilización de particiones siguiendo la idea de un árbol binario. Por lo tanto, dichas particiones no son disjuntas entre ellas, sino que se contienen unas a otras. Esto sigue la misma idea que se describió en el apartado de *Esquemas de estimación Basados en Árboles*. Apoyándose en este estilo de particionamiento se consigue que las consultas de rangos de frecuencias tengan un coste computacional menor en promedio (aún en el casos en que el rango sea muy amplio). En esta categoría destacan los histogramas *nLT* (n-level Tree) y *Lattice Histograms*. Estos últimos tratan de aprovechar las ventajas a nivel de flexibilidad y precisión que presentan los histogramas, además de las estrategias jerárquicas de sumarización en que se apoyan las *Wavelets* tal y como se describe en la siguiente sección.

Las ideas descritas en esta sección sobre los *histogramas* son extrapolables conforme se incrementa la dimensionalidad de los datos, en el caso de los esquemas de estimación, esto sucede de manera directa. Sin embargo, en el caso de los esquemas de particionamiento surgen distintos problemas debido al crecimiento exponencial tanto del espacio como del tiempo conforme aumenta el número de dimensiones de los datos.

Los *Histogramas* representan una estrategia sencilla, tanto a nivel de construcción como de consulta, la cual ofrece buenos resultados en un gran número de casos. Dichas estructuras han sido ampliamente probadas para aproximación de consultas relacionadas con suma de rangos o frecuencias puntuales. Tal y como se ha dicho previamente, su comportamiento en el caso unidimensional ha sido ampliamente estudiado, sin embargo, debido al crecimiento exponencial a nivel de complejidad conforme las dimensiones del conjunto de datos aumentan, estas estrategias son descartadas en

muchas ocasiones. Los *Histogramas* requieren de un conjunto de parámetros fijados *a-priori*, los cuales afectan en gran medida al grado de precisión de los mismos (pero cuando se seleccionan de manera adecuada esta solución goza de una gran cercanía al punto de optimalidad), por tanto, en casos en que la estimación de dichos valores necesarios *a-priori* se convierte en una labor complicada existen otras técnicas que ofrecen mejores resultados.

### 3.2.3. Wavelet

Las estructuras de sumariación denominadas *Wavelets*, a diferencia de las descritas anteriormente, han comenzado a utilizarse en el campo del *procesamiento aproximado de consultas* desde hace relativamente poco tiempo, por lo que su uso no está completamente asentado en soluciones comerciales sino que todavía están en fase de descubrimiento e investigación. Las *Wavelets* (u *ondículas*) se apoyan en la idea de representar la tabla de frecuencias del conjunto de datos como una función de ondas discreta. Para ello se almacenan distintos valores (dependiendo del tipo de *Wavelet*) que permiten reconstruir la tabla de frecuencias. Tal y como se describirá a continuación cuando se describa la *transformada de Haar*, la mejora de eficiencia en cuanto a espacio a partir de esta estructura de sumariación se apoya en el mantenimiento aproximado de los valores que representan el conjunto de datos.

A continuación se describe la *transformada de Haar*, a partir de la cual se presentan las distintas ideas en que se apoyan este tipo de estructuras de sumariación. En los últimos años se ha trabajado en estrategias más complejas como la *Daubechies Wavelet* [AH01] de Akansu y otros o la *transformada de Wavelet basada en árboles duales completos* [SBK05] de Selesnick y otros.

#### Haar Wavelet Transform

La *Haar Wavelet Transform* (**HWT**) consiste en una construcción de estructura jerárquica que colapsa las frecuencias de las distintas categorías de manera pareada recursivamente hasta llegar a un único elemento. Por tanto, la idea es la similar a la creación de un árbol binario desde las hojas hasta la raíz. Esta estrategia es similar a la que siguen los *Histogramas jerárquicos* de la sección anterior. Además, se aprovecha una idea similar al caso anterior para optimizar el espacio, consistente en almacenar la variación de uno de los nodos hoja con respecto del padre, lo cual permite reconstruir el árbol completo mediante una simple operación.

Para simplificar el entendimiento de la construcción de la *transformada de Haar* se describe un ejemplo extraído del libro *Synopses for massive data* [CGHJ12] de Cormode y otros. Supongamos los valores de frecuencias recogidos en  $A = [2, 2, 0, 2, 3, 5, 4, 4]$ . Para construir la transformada realizaremos la media de elementos dos a dos de manera recursiva, de tal manera que obtenemos para los distintos niveles los resultados de la tabla 3.1. Además, se muestran los coeficientes de detalle, los cuales se obtienen de calcular la diferencia entre el primer y segundo elemento de cada media.

Nivel	Medias	Coeficientes de Detalle
3	[2, 2, 0, 2, 3, 5, 4, 4]	—
2	[2, 1, 4, 4]	[0, -1, -1, 0]
1	[3/2, 4]	[1/2, 0]
0	[11/4]	[-5/4]

Tabla 3.1: Ejemplo de construcción de *Haar Wavelet Transform*

Nótese que a partir de la media de nivel 0 a la cual denominaremos  $c_0 = 11/4$  así como el conjunto de coeficientes de detalle, que denotaremos por  $c_1 = -5/4, c_2 = 1/2, \dots, c_7 = 0$  es posible reconstruir la tabla de frecuencias  $A$ .



Una vez entendida la estrategia de construcción en que se apoya la *transformada de Haar*, se puede apreciar que esta no ofrece ventajas a nivel de coste de almacenamiento respecto del conjunto de frecuencias con el cual ha sido construida. Sin embargo, posee la siguiente cualidad, en la cual se apoya esta estrategia de sumarización: *Para las categorías contiguas en que la variación de frecuencias es muy reducida, los coeficientes de detalle tienden a aproximarse a 0.*

Por la razón descrita en el párrafo anterior, se intuye que dichos coeficientes de detalle pueden ser obviados, de tal manera que el espacio utilizado para el almacenamiento de la *Wavelet* se convierte en sublineal ( $o(N)$ ), en lugar de lineal ( $O(N)$ ) respecto del espacio del conjunto de datos. Para elegir qué coeficientes de detalle se utilizan estrategias que tratan de minimizar el error. Comúnmente, las *Wavelets* han sido construidas a partir del *error cuadrático medio* o *norma- $L_2$* , la cual se describe en la ecuación (3.7). Sin embargo, distintos estudios como el descrito en el artículo *Probabilistic wavelet synopses* [GG04] de *Garofalakis y otros* muestran como esta obtienen medida del error obtiene malos resultados en el caso de sumarización de datos mediante *Wavelets*.

Por tanto, se proponen otras medidas de error como la minimización del máximo error absoluto o relativo, que se describen en las ecuaciones (3.8) y (3.9). También se propone como alternativa la minimización de la *norma- $L_p$*  que se describe en la ecuación (3.10). Dicha medida de error es una generalización del *error cuadrático medio* (caso  $p = 2$ ) a cualquier valor de  $p \geq 0$ . Por último se muestra en la ecuación (3.11) el caso del cálculo del error mediante la *norma- $L_p$*  con pesos, lo cual permite añadir mayor o menor importancia una determinada categoría sobre otras en la representación mediante *Wavelets*.

$$\|A - \tilde{A}\|_2 = \sqrt{\sum_i (A[i] - \tilde{A}[i])^2} \quad (3.7)$$

$$\max_i \{absErr_i\} = \max_i \{|A[i] - \tilde{A}[i]|\} \quad (3.8)$$

$$\max_i \{relErr_i\} = \max_i \left\{ \frac{|A[i] - \tilde{A}[i]|}{|A[i]|} \right\} \quad (3.9)$$

$$\|A - \tilde{A}\|_p = \left( \sum_i (A[i] - \tilde{A}[i])^p \right)^{\frac{1}{p}} \quad (3.10)$$

$$\|A - \tilde{A}\|_{p,w} = \left( \sum_i w_i \cdot (A[i] - \tilde{A}[i])^p \right)^{\frac{1}{p}} \quad (3.11)$$

Al igual que en el caso de los *Histogramas*, las *Wavelets* presentan problemas de eficiencia cuando se usa en entornos en los cuales el conjunto de datos está compuesto por una gran número de atributos. Por lo tanto se dice que sufren la *Maldición de la dimensionalidad* (*Curse of Dimensionality*) que provoca un crecimiento en el coste de orden exponencial tanto en espacio como tiempo respecto del número de dimensiones.

Tal y como se puede apreciar, esta estrategia es muy similar a la basada en *Histogramas*, dado que ambas se basan en el almacenamiento de valores que tratan de describir o resumir la tabla de frecuencias de los datos de manera similar. Sin embargo, mientras que en el caso de los *Histogramas* estos destacan cuando se pretende conocer la estructura general de los datos, las *Wavelets* ofrecen muy buenos resultados cuando se pretenden conocer valores atípico o extremos (a los cuales se denomina *Heavy Hitters*).

Por su estrategia de construcción, las *Wavelets* permiten sumarizar una mayor cantidad de información utilizando menos espacio. Además, en el caso de la *transformada de Haar*, que posee la característica de linealidad, se puede adaptar de manera sencilla al *modelo en Streaming*. Tal y como se ha dicho en el párrafo anterior, las desventajas de esta alternativa vienen derivadas en gran medida de los problemas relacionados con el incremento de la dimensionalidad de los datos.

### 3.2.4. Sketch

Los *Sketches* son una estrategia de sumariación de información enmarcada dentro del *Modelo en streaming* descrito en la sección 2.2. Dicha estrategia se caracteriza por procesar cada dato de entrada de manera secuencial, frente a otras alternativas como *Sampling*. Los *Sketches lineales* se caracterizan por procesar los datos de entrada realizando una *transformación lineal* sobre cada elemento y una estructura encargada de sumarizar la información de los sucesos previos.

[TODO modelo streaming]

[TODO tipos de sketches]

- **Estimación de Frecuencias:** [TODO ]
- **Elementos Distintos:** [TODO ]

[TODO dimensionalidad]

[TODO ventajas e inconvenientes]

[TODO área de investigación para soluciones futuras]

## 3.3. Bloom Filter

[TODO] *Space/time trade-offs in hash coding with allowable errors* [Blo70]

## 3.4. Count-Min Sketch

[TODO ] *An improved data stream summary: the count-min sketch and its applications* [CM05]

## 3.5. Count Sketch

[TODO ] *Finding frequent items in data streams* [CCFC02]

## 3.6. AMS Sketch

[TODO ] *The space complexity of approximating the frequency moments* [AMS96]

## 3.7. HyperLogLog

[TODO ] *Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm* [FFGM07]

## 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. Búsqueda 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?

En este apéndice se describen tanto la estructura como las tecnologías utilizadas para redactar este documento. El estilo visual que se ha aplicado al documento se ha tratado de almoladar lo máximo posible a las especificaciones suministradas en la *guía docente* de la asignatura *Trabajo de Fin de Grado* [uva17].

Este documento ha sido redactado utilizando la herramienta de generación de documentos  $\text{\LaTeX}$  [toog], en concreto se ha utilizado la distribución para sistemas *OS X* denominada *MacTeX* [tooh] desarrollada por la organización *TeX User Group*. Mediante esta estrategia todas las labores de compilación y generación de documentos *PDF* (tal y como se especifica en la guía docente) se realizan de manera local. Se ha preferido esta alternativa frente a otras como la utilización de plataformas online de redacción de documentos  $\text{\LaTeX}$  como *ShareLateX* [tooj] u *Overleaf* [tooi] por razones de flexibilidad permitiendo trabajar en lugares en que la conexión a internet no esté disponible. Sin embargo, dichos servicios ofrecen son una buena alternativa para redactar documentos sin tener que preocuparse por todos aquellos aspectos referidos con la instalación de la distribución u otros aspectos como un editor de texto. Además garantizan un alto grado de confiabilidad respecto de pérdidas inesperadas.

Junto con la distribución  $\text{\LaTeX}$  se han utilizado una gran cantidad de paquetes que extienden y simplifican el proceso de redactar documentos. Sin embargo, debido al tamaño de la lista de paquetes, esta será obviada en este apartado, pero puede ser consultada visualizando el correspondiente fichero `thestyle.sty` del documento.

Puesto que la alternativa escogida ha sido la de generar el documento mediante herramientas locales es necesario utilizar un editor de texto así como un visualizador de resultados. En este caso se ha utilizado *Atom* [tooa], un editor de texto de propósito general que destaca sobre el resto por ser desarrollado mediante licencia de software libre (*MIT License*) y estar mantenido por una amplia comunidad de desarrolladores además de una extensa cantidad de paquetes con los cuales se puede extender su funcionalidad. En este caso, para adaptar el comportamiento de *Atom* a las necesidades de escritura de texto con latex se han utilizados los siguientes paquetes: *latex* [tooc], *language-latex* [toob], *pdf-view* [tood] encargados de añadir la capacidad de compilar ficheros latex, añadir la sintaxis y permitir visualizar los resultados respectivamente.

Puesto que el *Trabajo de Fin de Grado* se refiere a algo que requiere de un periodo de tiempo de elaboración largo, que además sufrirá una gran cantidad de cambios, se ha creído conveniente la utilización de una herramienta de control de versiones que permita realizar un seguimiento de los cambios de manera organizada. Para ello se ha utilizado la tecnología *Git* [tooe] desarrollada originalmente por *Linus Torvalds*. En este caso en lugar de confiar en el entorno local u otro servidor propio se ha preferido utilizar la plataforma *GitHub* [toof], la cual ofrece un alto grado de confiabilidad respecto de posibles perdidas además de alojar un gran número de proyectos de software libre. A pesar de ofrecer licencias para estudiantes que permiten mantener el repositorio oculto al público, no se ha creído necesario en este caso, por lo cual se puede acceder al través de la siguiente url: <https://github.com/garciparedes/tfg-big-data-algorithms> [GP17]

```

.
+-- document
|   +-- bib
|       |   +-- article.bib
|       |   +-- ...
|   +-- img
|       |   +-- logo_uva.eps
|       |   +-- ...
|   +-- tex
|       |   +-- appendices
|       |       |   +-- how_it_was_build.tex
|       |       |   +-- ...
|       |   +-- chapters
|       |       |   +-- introduction.tex
|       |       |   +-- ...
|       |   +-- bibliography.tex
|       |   +-- ...
|   +-- document.tex
|   +-- ...
+-- notes
|   +-- readme.md
|   +-- ...
+-- summary
|   +-- summary.tex
|   +-- ...
+-- README.md
+-- ...

```

Figura B.1: Árbol de directorios del repositorio

Una vez descritas las distintas tecnologías y herramientas utilizadas para la elaboración de este trabajo, lo siguiente es hablar sobre la organización de ficheros. Todos los ficheros utilizados para este documento (obviando las referencias bibliográficas) han sido incluidos en el repositorio indicado anteriormente [GP17].

Para el documento, principal alojado en el directorio `/document/` se ha seguido una estructura modular, dividiendo los capítulos, apéndices y partes destacadas como portada, bibliografía o prefacio entre otros en distintos ficheros, lo cual permite un acceso sencillo a los mismos. Los apéndices y capítulos se han añadido en los subdirectorios separados. Para la labor de combinar el conjunto de ficheros en un único documento se ha utilizado el paquete *subfiles*. El fichero raíz a partir del cual se compila el documento es `document.tex`. La importación de los distintos paquetes así como la adaptación del estulo del documento a los requisitos impuestos se ha realizado en `thestyle.sty` mientras que el conjunto de variables necesarias como el nombre de los autores, del trabajo, etc. se han incluido en `thevars.sty`.

En cuanto al documento de resumen, en el cual se presenta una vista panorámica acerca de las distintas disciplinas de estudio relacionadas con el *Big Data* se ha preferido mantener un único fichero debido a la corta longitud del mismo. Este se encuentra en el directorio `/summary/`.

Por último se ha decidido añadir otro directorio denominado `/notes/` en el cual se han añadido distintas ideas de manera informal, así como enlaces a distintos cursos, artículos y sitios web en que se ha basado la base bibliográfica del trabajo. En la figura B.1 se muestra la estructura del repositorio en forma de árbol.

# Bibliografía

- [AGPR99] Swarup Acharya, Phillip B Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. Join synopses for approximate query answering. In *ACM SIGMOD Record*, volume 28, pages 275–286. ACM, 1999.
- [AH01] Ali N Akansu and Richard A Haddad. *Multiresolution signal decomposition: transforms, subbands, and wavelets*. Academic Press, 2001.
- [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.
- [Blo70] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
- [CCFC02] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.
- [CGHJ12] Graham Cormode, Minos Garofalakis, Peter J Haas, and Chris Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1–3):1–294, 2012.
- [Cha04] Shuchi Chawla. Notes on randomized algorithms: Chernoff bounds. October 2004.
- [CM05] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.
- [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.
- [GG04] Minos Garofalakis and Phillip B Gibbons. Probabilistic wavelet synopses. *ACM Transactions on Database Systems (TODS)*, 29(1):43–90, 2004.
- [GP17] Sergio García Prado. Trabajo de Fin de Grado: Algoritmos para Big Data, 2017. <https://github.com/garciparedes/tfg-big-data-algorithms>.
- [Ins17] Institute for Telecommunication Sciences. Definitions: Data stream, March 2017. [https://www.its.bldrdoc.gov/fs-1037/dir-010/\\_1451.htm](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.
- [PSC84] Gregory Piatetsky-Shapiro and Charles Connell. Accurate estimation of the number of tuples satisfying a condition. *ACM Sigmod Record*, 14(2):256–276, 1984.
- [SBK05] Ivan W Selesnick, Richard G Baraniuk, and Nick C Kingsbury. The dual-tree complex wavelet transform. *IEEE signal processing magazine*, 22(6):123–151, 2005.
- [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.
- [tooa] Atom. <https://atom.io>.
- [toob] Atom Package: Language LaTeX. <https://atom.io/packages/language-latex>.
- [tooc] Atom Package: LaTeX. <https://atom.io/packages/latex>.
- [tood] Atom Package: PDF View. <https://atom.io/packages/pdf-view>.
- [tooe] Git. <https://www.git-scm.com>.
- [toof] GitHub. <https://github.com>.
- [toog] LaTeX. <https://www.latex-project.org>.
- [tooh] MacTex. <http://www.tug.org/mactex>.
- [tooi] Overleaf. <https://www.overleaf.com>.
- [tooj] ShareLaTeX. <https://www.sharelatex.com>.
- [uva17] Trabajo de Fin de Grado (Mención en Computación), 2016/17. <https://www.inf.uva.es/wp-content/uploads/2016/06/G46978.pdf>.
- [Wik17] Wikipedia. Big Data, February 2017. [https://es.wikipedia.org/wiki/Big\\_data](https://es.wikipedia.org/wiki/Big_data).
- [Woo09] David Woodruff. Frequency moments. *Encyclopedia of Database Systems*, pages 1169–1170, 2009.