[75.06 / 95.58] Organización de Datos

Ejercicios Resueltos

Parciales 2014-2018

Federico del Mazo

Diciembre 2018

Índice

1. Compresión

1.1. 2014

Enunciado: Año 2014 Cuatrimestre 1 Oportunidad 2

V/F: Si usamos Block Sorting, MTF y LZW el resultado siempre sera peor a usar directamente LZW sobre el archivo original.

Criterio de Corrección: Año 2014 Cuatrimestre 1 Oportunidad 2

Esto puede ser verdadero o falso dependiendo de la justificación que realicen.

Esto no sucede siempre. La idea principal es que LZW y BS apuntan a distintos objetivos. Block Sorting (en conjunto con Move To Front) localiza un archivo, mientras que LZW aprovecha las secuencias repetidas (toda la familia LZ se trata de aprovechar secuencias).

Es decir, de tener un archivo con algún patrón en donde se repiten muchas secuencias, LZW será un gran compresor, y BS+MTF puede no tener mucho que aportar. Pero, de no tener ninguna secuencia en particular, BS+MTF *puede* (no necesariamente) dejar mejor parado al archivo para que LZW lo comprima.

Como en todos los V/F, cuando hay algún absoluto ('siempre', 'nunca'), hay que dudar bastante... no existe tal cosa como un algoritmo de compresión que en todo caso sea mejor que otro (si no, para que estudiaríamos el otro?), cada uno tiene sus objetivos, ventajas, desventajas y optimizaciones.

Enunciado: Año 2014 Cuatrimestre 1 Oportunidad 2

V/F: Se puede demostrar que la codificación realizada por LZW para literales y repeticiones no es óptima.

Criterio de Corrección: Año 2014 Cuatrimestre 1 Oportunidad 2

Verdadero porque hay códigos de 9 bits que no son posibles, entonces es fácil demostrar que no hacen falta realmente 9 bits para los mismos. Este era por lejos el mas simple de los 4 VoF

Esto es verdadero. Una codificación óptima es la más mínima posible representación del archivo a comprimir. Por lo tanto, por empezar, se debe demostrar que todos los bits a los que se comprime un archivo son necesarios e indispensables para esta compresión (si no lo fuesen, se podrían remover, y por ende, habría una codificación aún más chica posible).

De acá se deduce que un compresor debe aprovechar todos los bits que representan la codificación, no puede haber nada de más o sobrante. Entonces, todo tipo de secuencia de bits debe ser *valido* (debe poder descomprimirse).

En el caso de LZW, la codificación no es optima. LZW agarra secuencias, las indexa en una tabla de secuencias, y luego las codifica a su posición de ella (siendo la tabla inicial de 512 posiciones, cada índice es de 9 bits). Entonces, para ser óptima la compresión, todo número de 9 bits debe representar un caracter en la tabla de secuencias. Pero esto no pasa. Si uno tuviese el archivo de solamente 9 bits `100001110` (270 en binario), no se podría descomprimir a nada y el descompresor tendría un error en ejecución, porque (inicialmente) en la tabla de secuencias no hay nada en la posición 270.

Enunciado: Año 2014 Cuatrimestre 1 Oportunidad 2

V/F: Cualquier bloque binario de la longitud correcta puede descomprimirse usando block-sorting, mtf y Huffman. (en orden inverso claro)

Criterio de Corrección: Año 2014 Cuatrimestre 1 Oportunidad 2

En lineas generales esto es falso porque el indice "L" de block sorting podría apuntar a una rotación que en realidad no existe y por lo tanto el descompresor fallaría.

Que *pueda* descomprimirse es distinto a que se descomprima al contenido original. Que *pueda* no es mas que simplemente se corra el descompresor sin que explote el programa. Sin embargo, esto (que el descompresor tenga un error de ejecución) puede pasar en el caso de BS. Esto es porque el archivo comprimido por BS es la columna de últimas letras (luego de escribir todas las rotaciones posible de la cadena y ordenarlas) más un índice de rotación L que indica donde se ubica la cadena original. De tener un archivo comprimido donde el L sea mayor a la cantidad de posibles rotaciones de la cadena, al intentar acceder al indice L se recibiría un error de indexación ya que la rotación no existe y así no se podría descomprimir el bloque.

Enunciado: Año 2014 Cuatrimestre 1 Oportunidad 2

Comprimir el siguiente archivo usando LZSS con ventana de 4 caracteres usando 2 bits para la posicion y 2 bits para la longitud y con longitud mínima=2. Indique el resultado final en binario del archivo comprimido.
AAAAAAAABCABC

Criterio de Corrección: Año 2014 Cuatrimestre 1 Oportunidad 2

Luego de la primera A que ocupa 9 bits hay una repetición de lontitud 7 que no es representable en 2 bits (2,3,4,5) por lo tanto se genera una repetición de longitud 5, luego una de 2, etc. Si no detectan el match para la segunda A descuento de 7puntos. Si el padding está mal descuento de 3 puntos. Si no usan el bit para distinguir repeticion de literal vale 0 puntos. Si usan L0,L1,L2,L3 en lugar de L2,L3,L4 y L5 descuento de 8 puntos.

Nota general: Siempre que se pide comprimir un archivo hay que mostrar cuanto se comprimió! Nunca olvidarse de indicar cuanto ocupaba la cadena original y cuanto la comprimida, aunque el enunciado no lo diga.

LZSS:

- Cada vez que encuentro un caracter nuevo, lo comprimo a `0C` siendo C el caracter.
- Cada vez que encuentro una repetición (dentro de mi ventana de caracteres) la comprimo a *1(P,S)* siendo P la posición en la ventana desde la cual hay un *match* (y contando de derecha a izquierda es esto), y S la cantidad de caracteres que se repiten.
- Variables
  - **Ventana** Esto es la cantidad de caracteres que estoy dispuesto a retroceder en búsqueda de repeticiones. En este ejercicio, **4 caracteres**.
  - **Representación de posición** Las P de las repeticiones las represento en esta cantidad de bits. El 00 representa que la secuencia empieza en la posición 0 de la ventana (de derecha a izquierda, ya que se *retrocede* sobre la ventana), el 01 representa que la secuencia empieza en la posición 1, etc. En este ejercicio,
  - f2 bits.
  - **Longitud mínima de match** Me tengo que encontrar con al menos esta cantidad de caracteres repetidos como para considerarlo un match. En este ejercicio, **2 caracteres**.
  - **Representación de longitud**Ojo, este número se empieza a contar desde la longitud mínima de match, ya que toda longitud es al menos ese numero. Por lo tanto, de tener una longitud mínima de 2 caracteres y una representación de longitud de 2 bits, el 00 se refiere a una repetición de 2 caracteres, el 01 a una repetición de 3 caracteres, etc. Este número esta limitado por la representación, en este caso, solo se puede escribir hasta el 11 en bits, por lo tanto, solo se pueden contar hasta repeticiones de 5 caracteres. En este ejercicio, **2 bits**

Cadena original: AAAAAAAABCABC
Tamaño cadena original: 13 caracteres, de 8 bits cada uno: 104 bits.
Compresión:

- **A**: Nuevo caracter→0A
- **Segunda A**: Tengo una A en la primera posición (empezando en 0) de la ventana y la repetición es de 7 caracteres. Pero solo solo puedo representar repeticiones de hasta 5 caracteres, porque la representación es de 2 bits comenzando desde el número 2→1(0,5)→1 00 11
- **Sexta A**: Primera posición de ventana, repetición de 2 caracteres→1(0,2)→1 00 00
- **B**→0B
- **C**→0C
- **A**: Retrocedo a la segunda posición de la ventana (mi ventana actual es AABC, empezando desde la derecha el índice 2 se refiere a la A) y la repetición es de 3 caracteres (ABC)→1(2,3)→1 11 01

Cadena comprimida: 0A 1(0,5) 1(0,2) 0B 0C 1(2,3) →0A 10011 10000 0B 0C 11001 →0A10011100000B0C11001

Tamanó cadena comprimida: Las 3 no repeticiones ocupan 9 caracteres (8 del char original, 1 del 0), las 3 repeticiones ocupan 5 bits (uno del 1, dos de la posición, dos de la longitud). Entonces, ocupa 42 bits, pero el resultado siempre tiene que ser en bytes, por ende tiene que ser múltiplo de 8. Se completa con padding de 6 bits, terminando en 48 bits.

Compresión: Se comprimieron 104 bits a 48. Se comprimió un 53.8 %

1.2. 2015

Enunciado: Año 2015 Cuatrimestre 2 Oportunidad 1

Se tiene un archivo con 10 caracteres en total formado por tres caracteres distintos (ej: ABC). De todos los archivos posibles con estas características mostrar el archivo de máxima entropía que se pueda comprimir mejor usando LZ77. No es necesario comprimir el archivo.

Criterio de Corrección: Año 2015 Cuatrimestre 2 Oportunidad 1

Con las caracteristicas pedidas el archivo de maxima entropia tiene que tener frecuencias 4,3,3 para los caracteres, con estas frecuencias el archivo mas comprimible por LZSS es de tipo ABCABCABCA

La máxima entropía en un archivo se logra teniendo una distribución equitativa de los caracteres. El compresor LZ77 comprime aprovechando los patrones en el archivo. Por ende, el archivo con mayor entropía y mejor comprimido por algún algoritmo de la familia LZ es algo como: $ABCABCABCA$ (distribución equitativa y mayor cantidad de patrones).

Enunciado: Año 2015 Cuatrimestre 2 Oportunidad 2

Dado el siguiente archivo, comprimirlo aplicando Block Sorting, MTF y LZHuf con longitud mínima de match = 2 y máxima=4. (aplicarlos algoritmos en el orden especificado) Suponer que solo son posibleslos caracteres A,B,C y D (15 pts) (***).
“AAAAACCCC”

Nota general: Este es de los ejercicios más completos y largos para practicar compresión y utiliza muchos algoritmos claves para saber.

Cadena original: AAAAACCCC
Tamaño cadena original: 9 caracteres, de 8 bits cada uno: 72 bits.
El ejercicio pide:

1. Block Sorting
2. MTF
3. LZHuf

Se empieza por block sorting, un algoritmo de compresion para archivos localizados (caracteres iguales juntos). Este archivo ya es lo más localizado que puede ser (después de todo son todas A seguidas de todas C), pero para hacer completo el ejercicio utilizamos la transformación de Burrows y Wheeler, la cual localiza un archivo para luego utilizar block sorting sobre el (este paso es omitible si se justifica que el archivo ya es localizado, si no, block sorting de por si **implica** utilizar la transformación).

La transformación de Burrows y Wheeler consiste en escribir todas las rotaciones posibles de la cadena y luego ordenar estas rotaciones alfabeticamente. Una vez tenido esto, se encuentra la posición donde se encuentra la cadena original, a la cual llamamos L, y se almacena tanto este L como la última columna de las cadenas ordenadas.

Entonces, sacamos todas las rotaciones posibles de la cadena. Esto se puede hacer escribiendo la cadena e ir llevando la primera letra al final:

$$AAAAACCCC$$
$$AAAACCCCA$$
$$AAACCCCAA$$
$$AACCCCAAA$$
$$ACCCCAAAA$$
$$CCCCAAAAA$$
$$CCCAAAAAC$$
$$CCAAAAACC$$
$$CAAAAACCC$$

Ahora, ordenamos alfabeticamente este vector de cadenas:

$$AAAAACCCC$$
$$AAAACCCCA$$
$$AAACCCCAA$$
$$AACCCCAAA$$
$$ACCCCAAAA$$
$$CAAAAACCC$$
$$CCAAAAACC$$
$$CCCAAAAAC$$
$$CCCCAAAAA$$

Ahora, encontramos que la cadena original se ubica en la primera posición de este vector. Por ende, L es igual a 0. Y la cadena final, que es la que se refiere a la última columna de la matriz, es la cadena CAAAACCCA.

Ahora se aplica Move to Front a la cadena final. Si se salteo el paso de Burrows-Wheeler entonces se debería hacer sobre la cadena original, pero en este caso lo hacemos sobre lo que ahora es nuestra cadena final.

MTF consiste en escribir todos los caracteres posibles (en este ejercicio, A B C y D) e ir fijandose donde se ubican los caracteres de la cadena original sobre estos, una vez escrito el indice, se lleva al comienzo de la cadena el caracter.

char	[ABCD]	i
	ABCD	
C	CABD	2
A	ACBD	1
A	ACBD	0
A	ACBD	0
A	ACBD	0
C	CABD	1
C	CABD	0
C	CABD	0
A	ACBD	1

Entonces, ahora tenemos la cadena de índices 210001001 a la que le aplicaremos LZHuf. Notese que el universo de caracteres posibles para LZHuf será 0 1 2 y 3, ya que partíamos de la base de que solo A B C y D son posibles.

Teniendo la cadena 210001001 ahora aplicamos LZHuf. LZHuf consiste en aplicar LZSS y sobre esto utilizar un árbol de Huffman dinámico para los caracteres literales y para las longitudes de los matches. Empezamos por el LZSS, y nunca hacemos el paso a binario, ya que de esa parte se encarga el árbol de Huffman. El enunciado nos pide que el mínimo match sea de 2 caracteres y que el máximo match (es decir, el tamaño de la ventana) sea de 4 caracteres. Aunque no hagamos el pasaje a binario, se aclara que (elegido por el alumno) se representan las posiciones y longitudes en dos bits. Sin embargo, no hacemos todo lo de los bits en 0 y 1 para LZSS, ya que son bits innecesarios para LZHuf. Solamente anotamos los literales, las distancias (posiciones) y las longitudes.

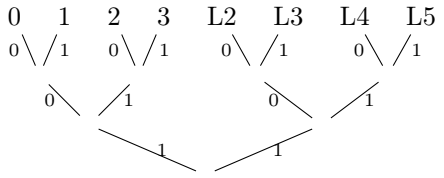
- **2**: Nuevo caracter→2.
- **1**: Nuevo caracter→1.
- **0**: Nuevo caracter→0.
- **0**: Tengo un 0 en la primera posición de la ventana (distancia 0) y la repetición es de 2 caracteres. →D0 y L2
- **1**: A 3 de distancia de la ventana, repetición de 3 caracteres→D3 y L3
- **1**: Si bien tengo un 1 en la ventana, el match mínimo es de 2, así que considero esto como un nuevo caracter. →1

Ahora teniendo la cadena 2 1 0 D0 L2 D3 L3 1 aplicamos la parte de Huf de LZHuf. Esta consiste en usar un árbol para los literales y las longitudes, y no perder rastro de las distancias con otro árbol o de forma fija. Opto por usar otro árbol dinámico para las distancias.

Primer árbol: Literales y Matches. Los literales posibles son 0, 1, 2 y 3, los matches posibles son L2, L3, L4 y L5 (porque como match mínimo teníamos longitud 2, y como decidimos representar en 2 bits las longitudes, el techo es L5).

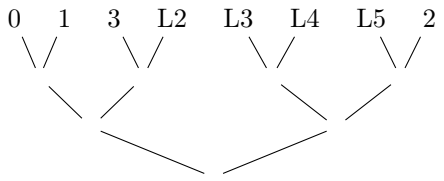
Dibujamos un árbol donde tenemos todos los caracteres posibles y les asignamos una frecuencia (o repetición) de 1. Las ramas del árbol surgen de unir los dos nodos con menor cantidad de repeticiones en cada paso. En este ejercicio no hay tantas repeticiones como para llegar a que se desbalancee el árbol, pero en cualquier otro caso no hay que olvidar de revisar en cada paso como se construyen las ramas.

Partiendo de la raiz, segun el caracter que vamos leyendo, vemos si nos movemos para la izquierda o para la derecha. Si vamos para la izquierda, el caracter se represneta con un 0, si vamos para la derecha, con un 1.

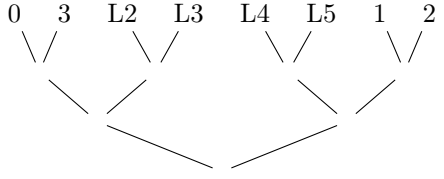


Leemos el primer 2, que se representa con un 010 (porque desde la raiz vamos uno para la izquierda, uno para la derecha, uno para la izquierda).

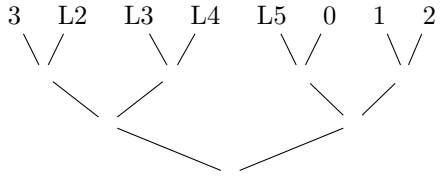
Reconstruimos el árbol, pero en orden de menor a mayor cantidad de repeticiones. En otras palabras, mandamos al 2, que ahora tiene dos repeticiones, a la derecha.



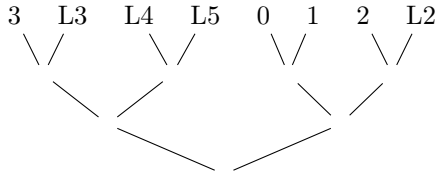
Leemos 1→001.



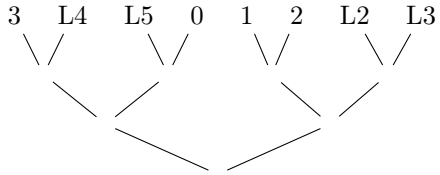
Leemos 0→000.



Leemos L2→001. También vemos que acá hay una distancia a registrar, pero de su código se encarga el otro árbol. Entonces, queda 001+Distancia(L2)

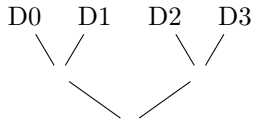


Leemos L3→001+Distancia(L3)

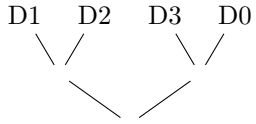


Leemos 1→100.

Terminado el primer árbol saltamos al segundo: el árbol de distancias. De acá solo nos importan las distancias de L2 y L3, que son D0 y D3 respectivamente:



Leemos D0→00



Leemos D3→10

La cadena final de ambos árboles es la de los literales, y cada vez que se cruza con una distancia, buscar el código en el otro árbol. Entonces queda el 010 (primer 2), después el 001, después el 000, después el 00100 (porque se refiere al 001 del caracter y al 00 de como se representa su distancia) y así sucesivamente. La cadena es 010 001 000 00100 00110 100. A esta cadena le faltan 2 bits de padding para ser múltiplo de 8, esto se puede lograr con diversos métodos no relevantes al ejercicio (por ejemplo, un nodo hoja en el árbol de huffman que se refiera al end of file, o guardar la longitud de la cadena a leer). Con solo mencionarlo basta.

Finalmente (!!!), escribimos la solución. La cual consiste en el arbol de literales y longitudes, más el arbol de distancias, más el L de rotación. De más esta decir que el descompresor de esto tendría que saber exactamente a lo que se esta enfrentando, primero descomprimiendo los caracteres del arbol de longitudes, fijandose sus distancias en el de distancias, sabiendo que eso se refiere a una cadena de índices que pasa por MTF, y sabiendo que a lo que descomprime eso es una cadena rotada con la cual L sabrá la cadena original.

Cadena comprimida: 010 001 000 00100 00110 100 (con su padding) junto a el 0 que se refiere al L, pero este 0 es un byte, ya que es un índice.

Tamanó cadena comprimida: La cadena de los árboles ocupa 22 bits, más 2 de padding. Y a esto se le agrega los 8 bits de L. 32 bits

Compresión: Se comprimieron 72 bits a 32. Se comprimió un 55.6 %

2. Hashing

2.1. 2014

Enunciado: Año 2014 Cuatrimestre 2 Oportunidad 1

V/F: Una función de hashing criptográfica produce muy pocas colisiones.

Criterio de Corrección: Año 2014 Cuatrimestre 2 Oportunidad 1

Verdadero porque es un requisito que no sea facil hallar colisiones por lo tanto tiene que minimizarlas.

Verdadero, pero cabe aclarar que no es ninguna propiedad específica de las funciones criptográficas. Es valido para las funciones de hashing en general.

Enunciado: Año 2014 Cuatrimestre 2 Oportunidad 1

V/F: La construcción de Davis-Meyer es necesaria para que la función de hashing produzca muy pocas colisiones.

Criterio de Corrección: Año 2014 Cuatrimestre 2 Oportunidad 1

Falso porque la construcción de DM apunta a que sea difícil hallar colisiones no a que haya pocas. Es decir que no es necesaria la construcción para que la función tenga pocas colisiones. (Por ej Jenkins no usa DM y tiene pocas colisiones)

Falso. Siempre hay que pensar que en un hashing criptográfico no hay *menos* colisiones, si no que es muy difícil encontrarlas.

La construcción de DM es una construcción que hace más *difícil* encontrar colisiones, pero no tiene nada que ver con la cantidad.

Dos construcciones que pueden usar las funciones de hashing criptográficas son:

- **Construcción de Merkle-Damgard**: Dado un string de longitud variable, pasar a un string de longitud fija.
- **Construcción de Davis-Meyer**: Dado un algoritmo de encripción, generar una función de compresión.

Enunciado: Año 2014 Cuatrimestre 2 Oportunidad 1

V/F: El efecto avalancha se produce cuando muchas claves hashean a un mismo valor generando muchas colisiones.

Criterio de Corrección: Año 2014 Cuatrimestre 2 Oportunidad 1

Falso el efecto avalancha se refiere a que un cambio mínimo en los datos produzca un cambio importante en su hash.

Falso. Efecto avalancha: Un cambio mínimo en la clave tiene que producir un cambio significativo en el resultado.

Enunciado: Año 2014 Cuatrimestre 2 Oportunidad 1

V/F: Una función de hashing para strings debe poder generar el resultado muy rapidamente

Criterio de Corrección: Año 2014 Cuatrimestre 2 Oportunidad 1

Verdadero una función de hashing que no es eficiente pone en duda que para acceder a un dato sea conveniente buscarlo aplicando una función de hashing contra, por ejemplo, simplemente buscarlo linealmente.

Verdadero. Una función de hashing tiene que, al menos, ganarle a la búsqueda lineal. Es decir, como mínimo, $O(fhashing) < O(n)$

2.2. 2015

Enunciado: Año 2015 Cuatrimestre 1 Oportunidad 1

V/F: Dado que una funcion de hashing criptografica tiene pocas colisiones es ideal para asegurarnos buscar claves rapidamente y su uso es recomendable en la mayoria de los casos.

Criterio de Corrección: Año 2015 Cuatrimestre 1 Oportunidad 1

Esto es falso ya que las funciones criptográficas suelen ser lentas en comparación con funciones de hashing convencionales que sin ser criptográficas cumplen la misma condición respecto de las colisiones.

La regla general es nunca usar una función de hashing genérica cuando se necesita una criptográfica (o sea, para encriptar algo) y nunca usar una función de hashing criptográfica cuando se necesita una genérica (o sea, conseguir los datos eficientemente).

Hay que tener en cuenta que ambos tipos de funciones apuntan a tener la misma cantidad de colisiones, la diferencia está en la dificultad de encontrarlas, pero no en la cantidad. El "Dado que tiene pocas colisiones"no es relevante, porque ya de por sí es así.

Enunciado: Año 2015 Cuatrimestre 1 Oportunidad 1

V/F: Un compresor estadistico estático comprime siempre mejor que un compresor estadístico dinámico

Criterio de Corrección: Año 2015 Cuatrimestre 1 Oportunidad 1

Hay que tener en cuenta el espacio necesario para la tabla que hace que los métodos estáticos de orden superior solo sean efectivos para archivos gigantes, por lo tanto y en general no podemos afirmar que un método estático siempre sea mejor que uno dinámico. PPMC es un método dinámico y salvo casos excepcionales comprime mejor que cualquier método estático. Esto se logra gracias a poder usar modelos probabilísticos mejores usando un orden alto para las predicciones lo cual, como ya explicamos sería prohibitivo en un método estático por el tamaño necesario para las tablas

Un compresor estático necesita espacio para la tabla de frecuencias. Entonces, si el archivo es lo suficientemente pequeño como para que no se amortice el tamaño de la tabla frente a la compresión del texto, o sea, si la tabla ocupa una porción significativa del archivo comprimido final, probablemente uno dinámico hubiese sido mejor opción.

Básicamente, mientrás más grande el archivo, más se desprecia el tamaño de la tabla de frecuencias, y por ende más se justifica el uso de un compresor estático.

2.3. 2016

Enunciado: Año 2016 Cuatrimestre 2 Oportunidad 1

V/F: Una función de hashing criptográfica como SHA-256 genera menos colisiones que una función genérica como Jenkins pero es mucho mas lenta

Criterio de Corrección: Año 2016 Cuatrimestre 2 Oportunidad 1

Esto es falso, no hay absolutamente ninguna propiedad de las funciones criptográficas que garantice que generen menos colisiones que una función de hashing no-criptográfica.

Falso. Si bien a un hashing genérico se le pide ser eficiente mientras que a uno criptográfico no, a ambos se les pide que produzcan la menor cantidad de colisiones posibles.

Las propiedades son:

Hashing Genérico	Hashing Criptográfico
Muy eficiente Menor cantidad de colisiones posibles	Menor cantidad de colisiones posibles Dado H(x) tiene que ser muy difícil hallar x Tiene que ser muy dificil hallar x e y tal que h(x)=h(y) Efecto avalancha: Un cambio minimo en la clave tiene que producir un cambio significativo en el resultado

Enunciado: Año 2016 Cuatrimestre 2 Oportunidad 1

V/F: Dado un conjunto de claves numéricas de 32 bits la función de hashing x % 1000 es igual de buena que la función de hashing x % 1001.

Criterio de Corrección: Año 2016 Cuatrimestre 2 Oportunidad 1

Esto es verdadero solo si los datos son uniformes, si los datos no son uniformes puede pasar que sean por ej múltiplos de 1000.

Una función de hashing es más eficiente que otra cuando tiene que buscar menos en sus 'baldes'. Es decir, una función de hashing es más eficiente que otra cuando tiene menos colisiones (porque las claves están mejor distribuidas haciendo más chica la búsqueda de una clave en un balde).

También, una función de hashing tiende a ser más eficiente cuando la cantidad de baldes es prima, esto es porque un número que comparte factor común con la cantidad de baldes se hashea a un múltiplo de este factor común, por ende, mientras menos factores tenga la cantidad de baldes, menos factores podrá compartir con los datos a hashear.

Ahora bien, es importante ver que esto solo es relevante cuando los datos no son uniformes. Si los datos a hashear son uniformes, la eficiencia de un hash con un número primo de baldes y uno con otro número será similar, porque no habrá patrón (como recibir múltiplos de n) del cual aprovecharse.

Entonces, un hash con 1001 baldes tiende a ser mejor que un hash con 1000 baldes, ya que 1001 tiene menos factores que 1000 ('1001 se acerca más a ser primo que 1000'), pero esto solo aplicaría en el caso de que la distribución no sea uniforme. En cambio, si es uniforme, serán igual de buenas.

Enunciado: Año 2016 Cuatrimestre 2 Oportunidad 1

Tenemos vectores en 4 dimensiones y usamos "the hashing trick" usando el método de una única función de hashing (es decir sin signo) para reducirlos a 3 dimensiones. Sabemos que la matriz asociada a la función de hashing usada es la siguiente (por filas): [1,0,0; 0,0,1; 0,1,0; 1,0,0]. Se pide construir la función de hashing h(x) equivalente a la matriz presentada.

Criterio de Corrección: Año 2016 Cuatrimestre 2 Oportunidad 1

Hay que plantear una f de hashing en base a la información que nos da la matriz. Si dan el resultado de la función por ejemplo h(3) = 2 pero no la función propiamente dicha entonces descuento de 5 puntos.

En la matriz asociada a un THT la fila se refiere a la posición original (es decir, el valor a hashear) mientras que las columnas se refieren a las posiciones a las que se hashean. Es decir, teniendo la matriz

$$\begin{matrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{matrix}$$

Podemos deducir que la función de hash sera una tal que se cumpla:

pos	h(pos)
0	0
1	2
2	1
3	0

Ahora, hay que buscar una función de hashing que cumpla eso. La manera mas sencilla es agarrar el prototipo de funciones de hashing del hashing universal para números: $h(x) = ((a * x + b)\,\%p)\,\%m$

Entonces buscamos a, b, m y p tal que se cumplan los valores pedidos. Obviamente, el valor de m será 3, ya que esa es la cantidad de baldes a la que queremos hashear. También se recuerda que a esta entre 1 y $p-1$, p es primo y es mayor o igual a m y b esta entre 0 y $p-1$. Se recomienda poner a b en 0.

Una posible función de hashing puede ser:

$$h(x) = ((2 * x + 0)\,\%8191)\,\%3$$

3. LSH

3.1. 2014

Enunciado: Año 2014 Cuatrimestre 1 Oportunidad 1

Se quiere aplicar LSH a un conjunto de documentos para encontrar los pares de documentos mas similares. Queremos que si J(D1,D2)>=0.7 entonces la probabilidad de que D1 y D2 sean candidatos sea >= 0.9 y queremos que si J(D1,D2)<= 0.5 entonces la probabilidad de que sean candidatos sea <= a 0.3. Indique cuántas funciones minhash usaría y que combinación de AND y OR usaría para lograr lo pedido

Criterio de Corrección: Año 2014 Cuatrimestre 1 Oportunidad 1

Hay que buscar r y b de forma tal que $1 - ((1 - 0{,}7^r)^b)$ sea >= 0.9 y $1 - ((1 - 0{,}3^r)^b)$ sea <= 0.2 la cantidad de funciones de hashing es r*b. Una forma de hacer esto mas o menos logica es ir probando con r=b a partir de 2... ya que la amplificacion que queremos es similar en ambas probabilidades. Si usan una cantidad de funciones de hashing insolita (400 o mas) descuento de 5 puntos.

Estamos queriendo armar *LSH(0.7;0.5;0.9;0.3)*, es decir, el LSH que si la distancia entre los docs es menor a 0.7 haya un 0.9 de probabilidad de encontrarlos, y si están alejados a 0.5 o más (más alejados equivale menor semejanza) la probabilidad de que sean candidatos sea menor a 0.3. Acá, con J(D1,D2) se refiere a la semejanza de Jaccard.

Como dice el criterio. Sabemos que la probabilidad de candidatura a semejanza es

$$P_{r,b} = 1 - (1 - P^r)^b$$

Entonces, buscamos el r y b tal que se cumplan las dos ecuaciones

$$1 - (1 - 0{,}7^r)^b \geq 0{,}9$$

$$1 - (1 - 0{,}5^r)^b \leq 0{,}3$$

Esto se debería solucionar con un grid-search de los buenos, intercalando entre r y b hasta que P1 y P2 nos den los resultados buscados. Pero también se pueden ir tirando valores al azar con un poco de intuición de para donde va la cosa.

r	b	$P1(\geq 0{,}9)$	$P2(\leq 0{,}3)$
2	2	0.7	-
3	3	0.86	-
2	4	0.93	0.68
5	10	0.84	0.27
6	20	0.91	0.27

Con 6 AND (r) y 20 OR (b) los resultados son los buscados. La cantidad de minhashes es $r * b = 120$.

Enunciado: Año 2014 Cuatrimestre 2 Oportunidad 1

Si se tiene la siguiente familia de funciones $LSH(0{,}15, 0{,}85, 0{,}85, 0{,}15)$ indique de que forma quedaría amplificada usando $r = 3$ y $b = 4$. Finalmente interprete el resultado de la familia amplificada indicando que cantidad de falsos positivos o falsos negativos se produciría

Criterio de Corrección: Año 2014 Cuatrimestre 2 Oportunidad 1

Hay que calcular $1-(1-(0{,}85^3))^4$ y $1-(1-(0{,}15^3))^4$ y luego explicar que el primer número nos sirve para estimar la cantidad de falsos negativos y el segundo número la cantidad de falsos positivos

La idea de LSH es encontrar documentos similares entre sí. Para hacer esto, se hace una familia de funciones de hashing donde adrede se quieren tener colisiones. Si dos documentos son similares, queremos que vayan a parar al mismo balde del hash. Así, después solo hay que recorrer el balde con documentos y discernir entre cuales verdaderamente son similares y cuales fueron mal clasificados.

Entonces, en cada balde de hash tendremos *candidatos a documentos similares.*

La preocupación de LSH es como clasificar documentos, no según que métrica son similares. Dos documentos pueden considerarse similares dependiendo de que objetivo se tiene en mente. Cantidad de bytes, utilizar las mismas palabras, hablar de las mismas ideas, etc. Lo único que le interesa a LSH es que exista una función de distancia. Mientras más alejados los documentos, menos similares son. Mientrás menor distancia haya entre los documentos, más similares son.

Un LSH se define como la familia de funciones de hashing tales que dos documentos a distancia menor o igual a d1 tengan una probabilidad p1 de ser considerados similares y que dos documentos a distancia mayor o igual a d2 tengan probabilidad p2 de ser considerados similares. Entonces, p1 tiene que ser un número alto y p2 uno bajo.

$$LSH(d1, d2, p1, p2)$$

Hay dos errores posibles:

- **Falsos Negativos**: Un falso negativo es todas las veces que dos documentos eran similares pero LSH no los detecto como tales. Como todas las veces que **sí** funciono LSH es p1, el contrario será todas las veces que no lo hizo. Es decir, la probabilidad de tener un falso negativo 1-p1.
- **Falsos Positivos**: Un falso positivo es todas las veces que dos documentos se consideraron similares pero no lo fueron. Sencillamente, la probabilidad de un falso positivo es p2.

$$P(\text{Falso Negativo}) = 1 - p1$$

$$P(\text{Falso Positivo}) = p2$$

Como utilizar solamente una función de hashing sería muy poco preciso se hacen dos cosas para evitar errores.

Por un lado, se pueden utilizar más tablas de hash en las que buscar. Al usar b tablas, en vez de buscar que dos documentos hayan colisionado en la única tabla, buscamos que hayan colisionado en cualquiera de todas las tablas. Con haber colisionado en una o (OR) la otra, ya se consideran candidatos a similares. Como se traen más documentos que antes, se reduce la posibilidad de haberse salteado algun par de documentos, así reduciendo la cantidad de falsos negativos. Pero, como se traen muchos documentos que antes no se consideraron, se incrementan las chances de considerar similitud donde no la hay, incrementando la cantidad de falsos positivos.

Por el otro lado, se pueden utilizar más funciones de hashing (por tabla). Así, le demandamos a los documentos que se consideren similares según distintos criterios. Se busca que dos documentos sean considerados similares por una función de hashing y (AND) por la otra. Esto hace que se recuperen menos documentos que antes, así reduciendo la cantidad de falsos positivos pero incrementando los falsos negativos.

Teniendo la probabilidad original P de que dos documentos sean candidatos, este número se puede va modificando con ANDs (r) y ORs (b).

$$P' = 1 - (1 - (P^r))^b$$

En el ejercicio se tiene $LSH(0{,}15, 0{,}85, 0{,}85, 0{,}15)$. Estas dos probabilidades (0.85 y 0.15) se van modificando con 3 ANDs y 4 ORs para llegar al siguiente resultado.

$$p1 = 0{,}85 \Rightarrow p1' = 1 - (1 - (p1^3))^4 = 0{,}98$$

$$p2 = 0{,}85 \Rightarrow p2' = 1 - (1 - (p2^3))^4 = 0{,}013$$

Como la probabilidad de que un par de documentos sea un falso negativo es $1 - p1$, si nosotros fuesemos a multiplicar esa probabilidad ($1 - 0{,}98 = 0{,}02$) por la cantidad de pares de documentos disponibles, se estimaría la cantidad de falsos negativos.

Análogamente, como la probabilidad de que un par de documentos sea un falso positivo es $p2$, si nosotros fuesemos a multiplicar esa probabilidad (0,013) por la cantidad de pares de documentos disponibles, se estimaría la cantidad de falsos positivos.

3.2. 2015

Enunciado: Año 2015 Cuatrimestre 2 Oportunidad 1

Tenemos la siguiente tabla representando el valor de 6 minhashes para tres documentos:

	D1	D2	D3
MH1	1	0	1
MH2	3	1	3
MH3	1	2	1
MH4	2	2	3
MH5	0	0	2
MH6	0	0	2

Se usa b=2 y r=3. Se decide usar 7 buckets para cada banda. Encontrar una única función de hashing perteneciente a una flia universal LSH(r1,r2,r3) de forma tal que en la primer banda solo D1 y D3 sean candidatos a ser similares pero en la segunda banda los tres documentos sean candidatos a ser similares.

Criterio de Corrección: Año 2015 Cuatrimestre 2 Oportunidad 1

Hay que encontrar una funcion de hashing de una flia universal tal que h(1,3,1)<>h(0,1,2) y h(2,0,0)=h(3,2,2). Si la funcion no pertenece a una flia de funciones universales descuento de 7 puntos.

Siendo b igual a 2 dividimos en 2 bandas a los minhashes. Por un lado, tenemos MH1,MH2,MH3 y por el otro tenemos MH4,MH5,MH6. Siendo r igual a 3 lo que queremos es que los 3 minhashes de la banda coincidan, no solamente alguno.

Ahora, de la primera banda queremos que D1 y D3 sean similares, entonces que sus 3 minhashes coincidan, pero que solamente pase eso, entonces ninguno de ambos puede coincidir con D2. Esto se resume a:

$$h(MH1(D1), MH2(D1), MH3(D1)) == h(MH1(D3), MH2(D3), MH3(D3))$$
$$h(MH1(D1), MH2(D1), MH3(D1)) \neq h(MH1(D2), MH2(D2), MH3(D2))$$
$$h(MH1(D3), MH2(D3), MH3(D3)) \neq h(MH1(D2), MH2(D2), MH3(D2))$$

Y también, de la segunda banda que los 3 documentos coincidan, y todos en sus 3 minhashes. Entonces:

$$h(MH4(D1), MH5(D1), MH6(D1)) == h(MH4(D2), MH5(D2), MH6(D2))$$
$$h(MH4(D2), MH5(D2), MH6(D2)) == h(MH4(D3), MH5(D3), MH6(D3))$$
$$h(MH4(D1), MH5(D1), MH6(D1)) == h(MH4(D3), MH5(D3), MH6(D3))$$

Reemplazando por los valores, estamos buscando h tal que:

$$h(1,3,1) == h(1,3,1)$$
$$h(1,3,1) \neq h(0,1,2)$$
$$h(1,3,1) \neq h(0,1,2)$$
$$h(2,0,0) == h(2,0,0)$$
$$h(2,0,0) == h(3,2,2)$$
$$h(2,0,0) == h(3,2,2)$$

Se puede ver que aunque sean muchas ecuaciones solo se esta queriendo que $h(1,3,1)$ sea distinto a $h(0,1,2)$ y que $h(2,0,0)$ sea igual a $h(3,2,2)$

Como se pide que la función de hashing sea universal, o bien hay que inventar una función de hashing y demostrar que es universal (lo cual es... increiblemente difícil), o bien hay que usar uno de los templates de funciones universales estudiados.

Lo más sencillo es usar la función de hashing universal para strings de longitud fija, $h(x) = ((a1 * x1 + a2 * x2 + a3 * x3)\,\%p)\,\%m$

- m es igual a 7, porque esa es la cantidad de buckets.
- Como m es primo, se puede saltear el uso de p.
- Los valores de ai son entre 0 y m-1

Finalmente, estamos buscando los valores de $a1$, $a2$ y $a3$, entre 0 y 6, tal que:

$$(a1 * 1 + a2 * 3 + a3 * 1)\,\%7 \neq (a1 * 0 + a2 * 1 + a3 * 2)\,\%7$$
$$(a1 * 2 + a2 * 0 + a3 * 0)\,\%7 == (a1 * 3 + a2 * 2 + a3 * 2)\,\%7$$

3.3. 2016

Enunciado: Año 2016 Cuatrimestre 2 Oportunidad 1

Dados los vectores: x=[1,3,-1,2]; y=[-1,-2,-1,-1], z=[2,4,-1,3] y los hiperplanos aleatorios: r1 = [+1,-1,+1] r2=[+1,+1,-1] r3=[-1,-1,-1],r4=[+1,+1,+1]. Queremos usar 3 (tres) hiperplanos para aproximar el coseno entre los vectores usando LSH. ¿Cuáles son los 3 hiperplanos que hay que elegir entre los 4 propuestos? Justifique adecuadamente.

Criterio de Corrección: Año 2016 Cuatrimestre 2 Oportunidad 1

En primer lugar cualquier proyección de 4 a 3 dimensiones sirve, puede ser usando hashing trick o simplemente tomando 3 de las 4 columnas. Luego simplemente hay que calcular el

signo del producto interno entre vectores e hiperplanos y ver que combinación aproxima mejor el coseno original entre los vectores en 4 dimensiones. No hay descuento por la forma de proyectar.

Empezamos por ver que los vectores son en R4 mientras que los hiperplanos son en R3, por ende, hay que reducir las dimensiones de los vectores. Esto se puede hacer con the hashing trick (THT). Es cierto que el teorema de Johnson y Lindenstrauss justifica esta reducción, pero cabe aclarar que no deja de ser una reducción severa pasar de 4 dimensiones a 3, ya que son pocas dimensiones y todas aportan una gran cantidad de información.

También, aplico el método de Weinberger (para completitud del ejercicio, pero no es ni obligatorio ni importante), que sirve para cuando hay muchas colisiones (lo cual resulta en un vector lleno de ruido). Lo que hace el método es agregar una función de hash que solo devuelve o +1 o -1 para saber el signo resultante de lo que se acumula en THT.

Uso entonces $h(x)h : R^4 \Rightarrow R^3$.

Esta funcion pasa de las *posiciones* $0, 1, 2, 3$ a las posiciones $0, 1, 2$. Como este no es un ejercicio acerca de THT, puedo no usar una función de hashing en particular y solo elegir (arbitrariamente) los resultados de $h(pos)$. Elijo lo siguiente:

pos	h(pos)	Weinberger
0	1	+1
1	0	-1
2	1	-1
3	2	+1

Entonces, convierto todos los vectores de R4 a vectores de R3 siguiendo el método de THT+W. La idea es agarrar el valor en la posición del vector original, fijarse a que posición hashea y poner el mismo valor, con el signo dado por W en esa posición. Si otra posición del vector original hashea ahí, simplemente se suman los valores. Esto se puede hacer o armando la matriz de THT y multiplicando el vector original por ella, o a mano uno por uno. De todas formas, los resultados son:

	R4 (original)	R3 (reducido)
x	1,3,-1,2	-3,2,2
y	-1,-2,-1,-1	2,0,1
z	2,4,-1,3	-4,3,3

Teniendo ahora tanto los vectores en R3 como los hiperplanos en R3, empezamos propiamente el ejercicio.

Cada hiperplano se refiere a un minhash del vector, con los únicos valores posibles siendo +1 y -1. El valor del minhash es simplemente el signo del producto interno entre el vector y el hiperplano. Entonces, haciendo el producto interno entre los vectores X, Y, Z y los hiperplanos R1, R2, R3 y R4, nos quedamos con el signo del producto interno y ese es el valor del minhash.

	r1	r2	r3	r4
x	-1	-1	-1	+1
y	+1	+1	-1	+1
z	-1	-1	-1	+1

Lo que se pide en el ejercicio es buscar la tri-upla de hiperplanos que reduzca lo más posible el coseno entre los vectores. Esto es porque buscamos los hiperplanos que alejen lo más posible a los vectores. Dos vectores pueden estar a lo sumo separados por 180 grados (siendo ese el piso, a menor coseno, mayor ángulo)

Ahora bien, lo relevante del método de los hiperplanos no es ni el coseno ni el ángulo entre los vectores, si no que simplemente lo que importa es la cantidad de minhashes en los que coinciden el par de vectores. Si un par de vectores coinciden en todos los minhashes, entonces (a ojos del minhash) son el mismo, y por ende el ángulo entre ellos será 0 (entonces el coseno será 1), mientras que si un par de vectores no coinciden en ningún minhash, están lo más alejados posible (esto es, a 180 grados) y el coseno será el menor posible.

Entonces, la pregunta se reduce a **¿en que 3 minhashes hay menor cantidad de coincidencias entre vectores?**. Por ejemplo, teniendo los hiperplanos R1 R2 y R3, los vectores X

e Y coinciden en solo 1 minhash. También sucede esto entre los vectores Y y Z. Y finalmente los vectores X y Z coinciden en todos los minhashes, haciendo que estén lo más juntos posibles.

Así, los resultados finales de coincidencias en minhashes son:

	r1,r2,r4	r2,r3,r4	r1,r3,r4	r1,r2,r4
x,y	1/3	2/3	2/3	1/3
y,z	1/3	2/3	2/3	1/3
x,z	3/3	3/3	3/3	3/3

Se puede ver que las tri-uplas de hiperplanos que menores coincidencias dan son tanto r1,r2,r3 como r1,r2 y r4.

Si bien no es importante, se aclara que coincidir en todos los minhashes (3 de 3) equivale a estar separados por 0 grados, coincidir en 2 de 3 minhashes equivale a estar separados por 60 grados, coincidir en solo un minhash equivale a estar separado por 120 grados y coincidir en ningún minhash singifica estar a 180 grados.

3.4. 2017

Enunciado: Año 2017 Cuatrimestre 2 Oportunidad 2

Usando LSH indicar si una construcción de 2 ANDs, 3 ORs, 2 ANDs y 3 ORs es equivalente a una construcción de 3 ORs, 2 ANDs, 3 ORs y 2 ANDs.

Criterio de Corrección: Año 2017 Cuatrimestre 2 Oportunidad 2

Deben plantear las ecuaciones y compararlas. Respuestas que no contengan análisis o justificación valen 0

AND: Se aplican más funciones de hash sobre la misma tabla, y se le exige a los candidatos a colisionar en cada balde a donde van a parar. Si la probabilidad de colisionar es P entonces la probabilidad de colisionar en los r baldes es igual a P^r.

OR: Si usamos más tablas, podemos considerar candidatos a quienes lo sean en alguna de todas las tablas. Si la probabilidad de colisionar es P entonces la probabilidad, entonces la probabilidad de no colisionar es $1 - P$, y por ende la probabilidad de no colisionar en ninguna de todas las tablas es $(1 - P)^b$. De acá, deducimos que la probabilidad de colisionar en alguna de todas las tablas es $1 - (1 - P)^b$.

Primer caso: 2 ANDs, 3 ORs, 2 ANDs y 3 ORs

$$P \Rightarrow P^2 \Rightarrow 1 - (1 - P^2)^3 \Rightarrow (1 - (1 - P^2)^3)^2 \Rightarrow 1 - 1 - ((1 - (1 - P^2)^3)^2)^3$$

Segundo caso: 3 ORs, 2 ANDs, 3 ORs y 2 ANDs

$$P \Rightarrow 1 - (1 - P)^3 \Rightarrow (1 - (1 - P)^3)^2 \Rightarrow 1 - (1 - ((1 - (1 - P)^3)^2))^3 \Rightarrow (1 - (1 - ((1 - (1 - P)^3)^2)^3)^2$$

3.5. 2018

Enunciado: Año 2018 Cuatrimestre 1 Oportunidad 1

Luego de calcular 8 minhashes, se obtiene la siguiente tabla para 3 documentos distintos. Se pide encontrar los documentos candidatos si se usa el siguiente esquema: 2 AND, 2 OR, 2 AND

	D1	D2	D3
H1	1	1	1
H2	-1	1	1
H3	1	-1	1
H4	-1	-1	1
H5	-1	1	-1
H6	1	-1	-1
H7	1	1	1
H8	1	1	-1

Criterio de Corrección: Año 2018 Cuatrimestre 1 Oportunidad 1

Se evalúa la resolución general, si no aplican correctamente el esquema planteado, o si no evalúan los candidatos de a pares se realizan descuentos en base al tipo de error cometido.

Se aplica 2AND, 2OR y 2AND, de izquierda a derecha, partiendo la tabla gigante primero en 2 (con el AND), luego a cada una de esas dos mitades se las parte de nuevo en 2 (con el OR) y finalmente a cada uno de estos cuartos de tabla se los parte en 2 (con el AND)

Primero se aplican 2 ANDs, o sea que se tiene que agarrar todos los H y dividirlos en 2. Los documentos candidatos deben coincidir en ambos subconjuntos (porque tiene que pasar en uno *y* (and) el otro). Los conjuntos de hashes en los que se divide la tabla son elegidos por el alumno. Yo elegí por un lado H1,H2,H3,H4 y por el otro H5,H6,H7,H8.

Después, se aplica un 2 OR, o sea que hay que agarrar estos dos conjuntos por separado, dividirlos a la mitad a cada uno, y los candidatos deben estar en una *o* (or) en la otra mitad para serlo. Yo elegí dividir el primer conjunto en H1,H2 y H3,H4 y el segundo en H5,H6 y H7,H8.

Finalmente, adentro de estos sub sub conjuntos se hace el último 2 AND. Es decir, se dividen en dos, y los candidatos deben estar en ambos H.

Entonces, para que haya semejanza entre dos documentos i y j, deben ser iguales sus hashes $H(D_i) = H(D_j)$ cumpliendo lo que fuimos armando.

Veamos los hashes por sub sub subconjuntos (partidos por el último AND):

- **H1 y H2**: Vemos que en H1 coinciden D2 y D3, mientras que en H2 también lo hacen. Como nosotros pedíamos con el AND que se cumpla en ambos casos, ya tenemos un par de candidatos: (D2,D3).
- **H3 y H4**: No hay par de documentos que coincidan sus H en ambos hashes. Si bien D1 y D3 coinciden en H3, no lo hacen en H4, y por el AND requeríamos que esto se cumpla en ambos casos.
- **H5 y H6**: Al igual que H3 y H4, no hay par de documentos que coincidan en ambos H.
- **H7 y H8**: Acá vemos que el par (D1,D2) coincide en ambos H, haciendolos un par candidato

Hasta ahora, en el último AND, coinciden por un lado D2 y D3 y por el otro D1 y D2. Ahora evaluamos por sub subconjuntos (partidos por el OR):

- **H1,H2 y H3,H4**: Lo que requeríamos era que haya una candidatura en al menos una de las dos partes (uno *o* la otra). Por ende, de acá sobrevive (D2,D3).
- **H5,H6 y H7,H8**: Con la misma lógica, de acá sobrevive (D1,D2).

Finalmente, vemos los subconjuntos originales H1,H2,H3,H4 y H5,H6,H7,H8, partidos por el primer AND. Al ser divididos por un AND, requeríamos que el par de documentos suceda en ambos subconjuntos. Sin embargo, no hay par de documentos candidatos al momento que se repita (por un lado teníamos D2,D3 y por el otro D1,D2). Por ende, no hay documentos candidatos.

Ojo, esto sucede solamente porque esta fue *mí* manera de partir los hashes. Tranquilamente se podrían haber encontrado pares candidatos de haber partido los hashes en otros conjuntos.

4. Reducción de Dimensiones

4.1. 2015

Enunciado: Año 2015 Cuatrimestre 2 Oportunidad 1

V/F: Tenemos una matriz con las calificaciones de películas x usuario. La matriz es de 50000 películas por 150000 usuarios. La dimensionalidad intrínseca de este set de datos seguramente es menor a 150000.

Criterio de Corrección: Año 2015 Cuatrimestre 2 Oportunidad 1

Verdadero, ya que el espacio total comprende la posibilidad de que para una pelicula se de cualquier combinacion de las 150000 calificaciones y esto claramente no es posible, por ejemplo no hay ninguna pelicula a la cual 150000 usuarios le den un 7 sin excepciones o que tengan ratings de tipo 1,2,3,4,5,6,7,8,9,10,1,2,3,4,5,6,7,8,9,10, etc...

Acá tenemos una matriz de 50000 filas (películas) y 150000 columnas (usuarios). La **dimensionalidad intrínseca** de la matriz es la cantidad de dimensiones que verdaderamente aportan información. Este número es sencillamente el rango de la matriz. Esto es porque una DVSR (la descomposición de la matriz quedandome solo con las R columnas, siendo R el rango) devuelve *intacta* a la matriz. Si una matriz tiene rango menor a su cantidad de columnas, entonces tiene columnas que son combinación lineal al resto, y por ende no aportan nada a la información del sistema.

Básicamente, la dimensionalidad intrinseca de una matriz esta asociada a su variedad de datos. Con tener al menos una fila o columna que sea una combinación lineal del resto, ya tenemos una fila o columna que no aporta información, y por ende puede ser removida. Dado el tamaño de la matriz en cuestión, no hay manera de que ninguna columna/fila sea combinación lineal de otra, así que el rango (la dimensionalidad intrinseca) es menor a la cantidad de columnas.

Enunciado: Año 2015 Cuatrimestre 2 Oportunidad 2

V/F: La SVD puede usarse tanto para reducir las dimensiones de un conjunto de puntos como para reducir la cantidad de puntos en un set de datos.

Criterio de Corrección: Año 2015 Cuatrimestre 2 Oportunidad 2

Verdadero hay que explicar que para reducir puntos hay que hacer la SVD de la transpuesta del set de datos.

Como dice el criterio, esto es verdadero ya que si queremos reducir las dimensiones hacemos $SVD(M)$ mientras que para reducir la cantidad de puntos hacemos $SVD(M^T)$

Enunciado: Año 2015 Cuatrimestre 2 Oportunidad 2

V/F: Si queremos representar un set de datos en dos dimensiones para visualizarlo entonces la SVD es siempre la mejor opción

Criterio de Corrección: Año 2015 Cuatrimestre 2 Oportunidad 2

Falso, si los datos no son linealmente separables metodos como ISOMAP o TSNE deberian funcionar mejor.

También bastante sencilla la respuesta. Como dice el criterio, es falso porque SVD es lo mejor para los datos linealmente separables. Si no, hay que recurrir a otra opción.

4.2. 2018

Enunciado: Año 2018 Cuatrimestre 1 Oportunidad 1

V/F: Al reducir dimensiones utilizando la descomposición por valores singulares lo que buscamos en quedarnos con los menores valores singulares que serán los que acumulen la mayor cantidad de energía de la matriz original

Criterio de Corrección: Año 2018 Cuatrimestre 1 Oportunidad 1

Falso, nos quedaremos con los valores singulares más significativos (mayores) los cuales además de permitirnos acumular la mayor cantidad de energía de la matriz, serán los que describen la dimensión intrínseca de los datos.

Esto es falso, los mayores valores singulares son los que contienen la mayor cantidad de energía.

Enunciado: Año 2018 Cuatrimestre 1 Oportunidad 1

Es conveniente realizar una reducción de dimensiones para lograr una mejor performance de ejecución de nuestro algoritmo de Machine Learning.

Criterio de Corrección: Año 2018 Cuatrimestre 1 Oportunidad 1

Falso, es solamente recomendable realizar una reducción de dimensiones si por motivos específicos (propios del algoritmo de Machine Learning, de la cantidad de datos o de los medios en los que podemos ejecutar el algoritmo, por ejemplo hardware) no podemos ejecutarlo con toda la información. Siempre será recomendable como etapa inicial usar toda la información disponible. Esto se basa en el Teorema Fundamental de la Dimensionalidad.

Esto es falso, siempre conviene primero intentar usar toda la información posible, y luego, si el algoritmo no anda o necesita que se le disponga la info de otra manera se le aplica una reducción a la info original.

Enunciado: Año 2018 Cuatrimestre 1 Oportunidad 1

SVD y PCA se enfocan en conservar la varianza, mientras q con T-SNE se concentra en que los puntos cercanos sigan cercanos, y los puntos que originalmente estaban alejados se mantengan alejados

Criterio de Corrección: Año 2018 Cuatrimestre 1 Oportunidad 1

Falso, T-SNE tiene como objetivo que dos puntos que estaban cercanos en el espacio original permanezcan cercanos en el espacio reducido, pero nada podemos inferir de los puntos que originalmente se encontraban alejados.

La sentencia entera es falsa, aunque es cierto que PCA se enfoca en conservar la variabilidad y que TSNE se concentra en que los puntos cercanos lo sigan estando.

Enunciado: Año 2018 Cuatrimestre 1 Oportunidad 1

Cuando trabajamos con ISOMAP, la matriz de entrada de MDS tendrá siempre ceros en la diagonal

Criterio de Corrección: Año 2018 Cuatrimestre 1 Oportunidad 1

Verdadero, ya que con ISOMAP aplicamos MDS a la matriz de distancia de todos los puntos contra todos.

ISOMAP se aplica sobre la matriz de distancias entre los puntos. Como cada punto tiene 0 de distancia contra si mismo, los elemento $M_{i,i}$ (los de la diagonal) serán cero.

5. Information Retrieval

5.1. 2014

Enunciado: Año 2014 Cuatrimestre 1 Oportunidad 1

V/F: Si para una cierta consulta q d1 es mas relevante que d2 entonces luego de aplicar LSI d1 seguirá siendo mas relevante que d2.

Criterio de Corrección: Año 2014 Cuatrimestre 1 Oportunidad 1

Falso, al aplicar LSI los terminos que son sinonimos de los buscados pueden hacer que D2 sobrepase a D1 suponiendo que los sinonimos estan en D2 pero no en D1.

Básicamente, LSI agrupa en conceptos en vez de por palabra, haciendo que *pueda* darse vuelta la relación de relevancia.

Enunciado: Año 2014 Cuatrimestre 1 Oportunidad 2

En un índice invertido en el cuál tenemos 2500 documentos y 785 términos estime cuál es una cota superior para el tamaño total del índice

Criterio de Corrección: Año 2014 Cuatrimestre 1 Oportunidad 2

p nunca puede ser mayor a 0.5 por lo tanto el peor caso es con p=0.5 y en ese caso usando código unario se puede estimar la cantidad máxima de punteros y el tamaño de los mismos.

Por empezar, el máximo (se pide una cota superior)indice sería aquel que esté lo más cargado posible.

Empezamos con el caso más trivial posible: todas las posibles apariciones. Esto es, que todo termino aparezca en todo documento. Esto equivale a $docs * terms$.

Ahora, este índice no tiene sentido. Si la probabilidad de que un término aparezca en un documento es mayor al 50 %, entonces hubiese sido más inteligente que el índice guarde las *no* apariciones.

Entonces, el máximo índice posible, es decir, el peor caso posible para el índice, sería aquel acotado por la idea de un 50 % de $docs * terms$

El tamaño de este indice esta representado en binario. Por lo tanto, haciendo un logaritmo en base 2 obtenemos el tamaño del índice. Entonces

$$log_2(docs * t) * 0{,}5$$

$$log_2(2500 * 785) * 0{,}5 = 10{,}45$$

Enunciado: Año 2014 Cuatrimestre 2 Oportunidad 1

Considerando a cada línea como un documento y cada palabra como un término construya la matriz de términos x documentos usando TF-IDF. Finalmente indique cuantas dimensiones elegiría para usar LSI justificando adecuadamente su respuesta.
D1: Salsa Tomate Pizza Muzzarella Tomate
D2: Fideos Tomate Salsa Fideos Fideos
D3: Pizza Tomate Tomate
D4: Fideos Salsa Salsa Salsa

Criterio de Corrección: Año 2014 Cuatrimestre 1 Oportunidad 2

Puede usarse la frecuencia del término en el documento o log(ftij) indistintamente. El logaritmo para el IF puede estar encualquier base. Si no normalizan descuento de 5 puntos. La parte de LSI vale 5 puntos y apunta a que noten que los documentos hablan o de Pizzas o de Fideos por lo tanto tendría sentido reducir todo a 2 dimensiones (conceptos). A aquellos que hilando fino digan que habría que calcular los autovalores de la matriz y preservar un 80 % de la sumatoria del cuadrado de los mismos lesvamos a decir que tienen razón pero que hay formas mas fáciles de darse cuenta que un documento habla de fideos o pizzas (no hay descuento).

La idea del TF-IDF es hacer una matriz de todos los documentos (como filas) y todos los términos buscados (como columnas) y el valor del elemento sea el TF del término en el documento multiplicado al IDF del término.

El TF de un término en un documento es la cantidad de apariciones de ese T en el D. Este número se puede reducir usando $log(TF + 1)$ pero esto es criterio del alumno (yo no lo uso).

El IDF de un término es $log_2(\frac{N+1}{F(t)})$ siendo N la cantidad de documentos totales, y F(t) la cantidad de documentos donde aparece ese término.

Finalmente, la matriz se puede normalizar dividiendo cada fila (documento) por $1 - b + b * (\frac{\text{palabras totales en documento}}{AVDL})$ siendo el valor de b a elección (usualmente se usa 0.75) y con $AVDL = \frac{\text{palabras totales}}{\text{documentos}}$.

En este ejercicio no hay consulta en particular, así que hay que hacer toda la matriz de TF-IDF, pero es importante saber que si hay una consulta en particular **no** hay que hacer toda la matriz, porque esta mal que la máquina sobre-trabaje.

Armamos la tabla con los respectivos TF e IDF. El IDF es una constante por término, por ende se puede calcular de antes:

- $IDF(\text{salsa}) : log_2(\frac{5}{3}) = 0{,}74$
- $IDF(\text{tomate}) : log_2(\frac{5}{3}) = 0{,}74$
- $IDF(\text{pizza}) : log_2(\frac{5}{2}) = 1{,}32$
- $IDF(\text{muzzarella}) : log_2(\frac{5}{1}) = 2{,}32$
- $IDF(\text{fideos}) : log_2(\frac{5}{2}) = 1{,}32$

Ahora sí, hacemos la tabla/matriz:

	Salsa	Tomate	Pizza	Muzzarella	Fideos
D1	1 * 0.74	2 * 0.74	1 * 1.32	1 * 1.32	0 * 1.32
D2	1 * 0.74	1 * 0.74	0 * 1.32	0 * 1.32	1 * 1.32
D3	0 * 0.74	2 * 0.74	1 * 1.32	0 * 1.32	0 * 1.32
D4	3 * 0.74	0 * 0.74	0 * 1.32	0 * 1.32	1 * 1.32

Ahora, hay que normalizar la tabla por filas (documentos). La normalización la hacemos con b=0.75. El AVDL (el divisor del tercer término) es el mismo para todos los documentos, y en este caso es $\frac{17}{4} = 4{,}25$

Las normalizaciones quedan:

- $norm(D_1) : 1 - 0{,}75 + 0{,}75 * \frac{5}{4{,}25} = 1{,}13$
- $norm(D_2) : 1 - 0{,}75 + 0{,}75 * \frac{5}{4{,}25} = 1{,}13$
- $norm(D_3) : 1 - 0{,}75 + 0{,}75 * \frac{3}{4{,}25} = 0{,}78$
- $norm(D_4) : 1 - 0{,}75 + 0{,}75 * \frac{4}{4{,}25} = 0{,}96$

Así, la matriz final (dividir cada elemento de la fila por $norm(D_i)$) queda:

	Salsa	Tomate	Pizza	Muzzarella	Fideos
D1	0.65	1.31	1.17	1.17	0
D2	0.65	0.65	0	0	1.17
D3	0	1.90	1.69	0	0
D4	2.31	0	0	0	1.38

Finalizada la parte de TF-IDF, pasamos a la parte de LSI.

LSI es simplemente aplicar SVD a la matriz de TF-IDF para luego reconstruir la matriz y así reducir las dimensiones a una cantidad deseada. Para hacer esto LSI junta varias dimensiones (términos) en una cantidad menor. Pero la cantidad de dimensiones a la que se reduce es elegida por quien aplique LSI.

La idea es usar LSI para agrupar los términos según *conceptos*. Entonces, viendo los documentos, hay que elegir la cantidad de conceptos presentes. En este caso, se puede ver que los conceptos son 2: pizzas y fideos.

5.2. 2018

Enunciado: Año 2018 Cuatrimestre 1 Oportunidad 1

Se tienen los siguientes documentos:
D1: VW VW FORD FIAT
D2: ALFAROMEO LANCIA ALFAROMEO
D3: FERRARI FIAT ALFAROMEO LANCIA
D4: FORD FORD FIAT CHEVROLET FIAT
D5: CHEVROLET VW CHEVROLET VW VW
D6: FIAT FERRARI FERRARI
Dada la consulta "FERRARI FIAT"dar el resultado de la consulta rankeada utilizando TF.IDF. - Considerando como relevante los documentos que hablen únicamente sobre marcas italianas (Fiat, Ferrari, Alfa Romeo, Lancia), calcular la Precisión, Recall y F1 Score.

Criterio de Corrección: Año 2018 Cuatrimestre 1 Oportunidad 1

a) Si aplican mal las fórmulas cero. Si no aplican corrección de Laplace -3. Errores de cuentas no descuentan salvo que sea evidente que el resultado obtenido está mal, en ese caso descuenta por error conceptual en base al tipo de error cometido. Si dicen que los documentos recuperados son todos, descuento de 3 puntos. Si solo dicen que se recupera D6, descuento de 3 puntos. Si calculan mal en b) precisión, recall o F1 -2 por cada uno.

Siendo nuestra consulta FERRARI FIAT comenzamos por armar una tabla con los respectivos TF e IDF. El IDF es una constante por término, por ende se puede calcular de antes:
El IDF de 'FERRARI' es $log_2 \frac{5}{2} = 1{,}32$ y el de 'FIAT' $log_2 \frac{5}{3} = 0{,}73$

	Ferrari	Fiat
D1	0 * 1.32	1 * 0.73
D2	0 * 1.32	0 * 0.73
D3	1 * 1.32	1 * 0.73
D4	0 * 1.32	2 * 0.73
D5	0 * 1.32	0 * 0.73
D6	2 * 1.32	1 * 0.73

Ahora, hay que normalizar la tabla por filas (documentos). La normalización la hacemos con b=0.75. El AVDL (el divisor del tercer término) es el mismo para todos los documentos, y en este caso es $\frac{24}{6} = 4$
Las normalizaciones quedan:

- $norm(D_1) : 1 - 0{,}75 + 0{,}75 * \frac{4}{4} = 1$
- $norm(D_2) : 1 - 0{,}75 + 0{,}75 * \frac{3}{4} = 0{,}81$

- $norm(D_3) : 1 - 0{,}75 + 0{,}75 * \frac{4}{4} = 1$
- $norm(D_4) : 1 - 0{,}75 + 0{,}75 * \frac{5}{4} = 1{,}19$
- $norm(D_5) : 1 - 0{,}75 + 0{,}75 * \frac{5}{4} = 1{,}19$
- $norm(D_6) : 1 - 0{,}75 + 0{,}75 * \frac{3}{4} = 0{,}81$

Así, la matriz final (dividir cada elemento de la fila por $norm(D_i)$) queda:

	Ferrari	Fiat
D1	0	0.73
D2	0	0
D3	1.32	0.73
D4	0	1.23
D5	0	0
D6	3.26	0.90

Luego de tener la matriz ordenamos en según el puntaje de cada documento, siendo el puntaje la sumatoria de TFIDFs de los términos:

1. $D_6 = 4{,}16$
2. $D_3 = 2{,}05$
3. $D_4 = 1{,}23$
4. $D_1 = 0{,}73$
5. $D_2 = 0$
6. $D_5 = 0$

Los documentos recuperados son entonces D6, D3, D4 y D1 (todos los no nulos).
Calculamos finalmente la precisión, recall y F1, considerando relevantes a D2 y D6:

$$Precision = \frac{\text{Docs Relevantes Recuperados}}{\text{Docs Recuperados}} = \frac{1}{4} = 0{,}25$$

$$Recall = \frac{\text{Docs Relevantes Recuperados}}{\text{Docs Relevantes}} = \frac{1}{2} = 0{,}5$$

$$F1Score = \frac{2 * P * R}{P + R} = \frac{2 * 0{,}25 * 0{,}5}{0{,}25 + 0{,}5} = 0{,}25$$