

HASHING

LAS FUNCIONES DE HASHING NOS PERMITEN CONVERTIR CUALQUIER TIPO DE DATO EN UN NÚMERO. UTILIZANDO ESTAS FUNCIONES DE HASHING, PODEMOS ACCEDER DE FORMA MUY RÁPIDA A UNA TABLA/VECTOR EN MEMORIA DENOMINADA 'TABLA DE HASH'.

ESTO NOS BENEFICIA PARA PODER REPRESENTAR DATOS EN FORMA COMPACTA.

PODEMOS PENSAR LAS FUNCIONES DE HASHING COMO:

$$H: U \rightarrow \{0, 1, \dots, m - 1\}$$

DONDE U ES EL ESPACIO DE CLAVES Y $\{ \dots \}$ ES EL ESPACIO DE DIRECCIONES.

FUNCIONES DE HASH: NO CRIPTOGRAFICAS

SON AQUELLAS FUNCIONES QUE CUMPLEN:

- DEBE SER MUY EFICIENTE CALCULAR $H(x)$
 - DEBE PRODUCIR LA MENOR CANTIDAD DE COLISION
- UNOS EJEMPLOS SON:

- **FNV**: SE BASA EN PROPIEDADES DE CIERTOS NÚMEROS PRIMOS.
- **JENKINS**: SE BASA EN REALIZAR OPERACIONES LÓGICAS COMO XOR Y SHIFTS.
- **PEARSON**: CONTIENE VARIAS PROPIEDADES INTERESANTES, ESTA SE ACERCA A UN HASHING PERFECTO.

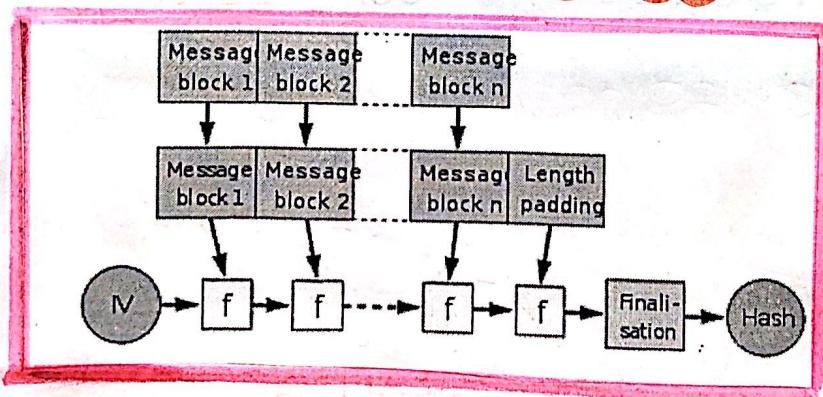
CRÍPTOGRAFÍCAS

ESTAS DEBEN CUMPLIR REQUISITOS ADICIONALES QUE HACEN A LA SEGURIDAD DE LAS MISMAS. ESTAS DEBEN CUMPLIR CON:

- + TIENE QUE PRODUCIR LA MENOR CANTIDAD DE COLISIONES POSIBLES.
- + DADO $H(x)$ TIENE QUE SER MUY DIFÍCIL HALLAR x
- + TIENE QUE SER MUY DIFÍCIL HALLAR x E Y DE FORMA TAL QUE $H(x) = H(y)$
- + UN CAMBIO MINIMO EN LA CLAVE TIENE QUE PRODUCIR UN CAMBIO SIGNIFICATIVO EN EL RESULTADO → **EFEITO AVALANCHA**.

LA FUNCIÓN CRÍPTOGRAFICA MÁS USADA ES **SHA - 256**. SHA - 256 SE BASA EN LA COMBINACIÓN DE DOS PRIMITIVAS CRÍPTOGRAFICAS MUY IMPORTANTES **MERKLE - DAMGARD** Y **DAVIS MAYER**.

MERKLE - DAMGARD



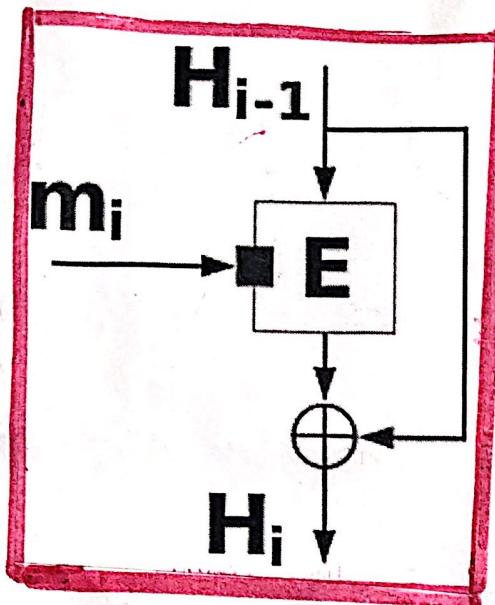
EN ESTA CONSTRUCCIÓN SUPONEMOS QUE EXISTE UNA FUNCIÓN 'F' QUE RECIBE UN BLOQUE DE M BITS Y DEVUELVE UN BLOQUE DE N BITS ($N < M$)

→ FUNCIÓN DE COMPRESIÓN
ENTONCES EL OBJETIVO ES DADO UN STADING CON RANDOM B

EMITIR UN ÚNICO RESULTADO DE TAMAÑO IGUAL AL EMITIDO POR FC.
SUPONGAMOS QUE N ES LA CANTIDAD DE BITS QUE EMITE FC,
DIVIDIMOS NUESTRO MENSAJE EN BLOQUES DE N , AGREGANDO UN PADDING AL FINAL SI ES NECESARIO.

COMO FC RECIBE BLOQUES DE M BITS SE AGREGA UN BLOQUE FICTICIO LLAMADO IV CON EL QUE COMENZAMOS EL ALGORITMO.
ENTONCES, ESTA CONSTRUCCIÓN SE BASA EN CONCATENAR EL RESULTADO ANTERIOR DE F AL BLOQUE PRÓXIMO Y VOLVER A APLICAR

DAVIS - MEYER



LA CONSTRUCCIÓN DE DAVIS - MEYER PERMITE GENERAR UNA FUNCIÓN DE COMPRESIÓN SEGURA, A PARTIR DE UN ALGORITMO DE ENCIPIÓN SEGURO.

RECIBIMOS DOS BLOQUES Y USAMOS UNO COMO CLAVE PARA ENCIPTAR EL OTRO, FINALIZANDO CON EL XOR ENTRE EL BLOQUE 1 Y EL RESULTADO DE LA ENCIPIÓN

HASHING UNIVERSAL

SEA H UNA FAMILIA DE FUNCIONES DE HASHING DE LA FORMA $h \in H : U \rightarrow \{0 \dots M-1\}$. DECIMOS QUE H ES UNIVERSAL SI PARA CUALQUIER F DE HASHING DE NUESTRA FAMILIA LA PROBABILIDAD DE QUE DOS CLAVES DIFERENTES COUSIONEN DEBE SER MENOR O IGUAL $\frac{1}{M}$, SIENDO M EL ESPACIO DE DIRECCIONES.

$$\forall x, y \in U, h \in H, x \neq y$$

$$P[H(x) = H(y)] \leq \frac{1}{M}$$

EL HASHING UNIVERSAL SE PUEDE APLICAR PARA DISTINTOS TIPOS DE DATOS :

- VALORES ATÓMICOS (NUMÉRICO)
- CLAVES DE LONGITUD FIJA
- CLAVES DE LONGITUD VARIABLE

VALORES NUMÉRICOS

$$H(x) = [Ax + B \pmod{P}] \pmod{M}$$

CARTER - WEGMAN

DONDE P ES UN NÚMERO PRIMO ($P \geq M$), A ES UN NÚMERO ENTRE 1 Y $P-1$ Y B ES UN NÚMERO ENTRE 0 Y $P-1$. PARA RECORDAR, M ES EL ESPACIO DE DIRECCIONES.

CLAVES DE LONG. FIJA

$$H(x) = \sum_{i=0}^R [A_i * x_i \pmod{P}] \pmod{M}$$

SEA M NUESTRO ESPACIO DE DIRECCIONES, P UN NÚMERO PRIMO ($P \geq M$) Y SEAN CLAVES X COMPUESTAS POR $R+1$ $R+1$ NÚMEROS: $x_0 \dots x_R$. ELEGIMOS $R+1$ NÚMEROS A_i ENTRE 1 Y $M-1$.

SI M ES UN NÚMERO PRIMO: $H(x) = \sum_{i=0}^R A_i * x_i \pmod{M}$

CLAVES DE LONG VARIABLE

$$H(x) = H_{\text{INT}} \left(\left(\sum_{i=1}^L x_i * A^i \right) \pmod{P} \right)$$

P ES NÚMERO PRIMO GRANDE, A ES UN NÚMERO ALEATORIO ENTRE 0 Y $P-1$, H_{INT} ES UNA FUNCIÓN DE HASHING DE LONG FIJA QUE RECIBE UN NÚMERO ENTRE 0 Y $P-1$ Y DEVUELVE OTRO ENTRE 0 Y $M-1$ SIENDO M EL ESPACIO DE DIRECCIONES Y L LA LONGITUD DE LA CLAVE.

HASHING PERFECTO

UNA FUNCIÓN DE HASHING PERFECTA ES AQUELLA QUE NO TIENE COLISIONES, ESTO NOS GARANTIZA $O(1)$ PARA CUALQUIER BUSQUEDA DE CLAVES.

VAMOS A VER UN MECANISMO PARA CREAR UNA FUNCIÓN DE HASHING PERFECTA.

FKS

- GARANTIZA $O(1)$ EN BUSQUEDA
- ES $O(N)$ EN ESPACIO
- RESUELVE COLISIONES BUSCANDO FUNCIONES DE HASH QUE DIFERENCEN LAS CLAVES.

- ELEMENTOS A ALMACENAR $\{1, 5, 6, 9, 12, 13\}$

COMO SON 6 ELEMENTOS, EL MÍNIMO TAMAÑO DE TABLA ES $M = 6$. PERO VAMOS A TRABAJAR CON NÚMEROS PRIMOS ASÍ QUE EL MÍNIMO ES $K = 7$.

$$\bullet H(x) = x \bmod k$$

- PASAMOS TODOS LOS ELEMENTOS POR $H(x)$ Y LOS COLOCAMOS EN LA TABLA.

K	CLAVES	HASH 2	SEGUNDA TABLA
0			
1	1	$x \bmod 1$	{1}
2	9	$x \bmod 1$	{9}
3			
4			
5	5, 12	$x \bmod 5$	{5, -, 12, -, -}
6	6, 13	$2x \bmod 5$	{-, 13, 6, -, -}

COLISION DE
TAMAÑO = 2
 $K = 5$

- SI HAY COLISIONES
- VAMOS A DECIR QUE EL TAMAÑO ES M_i Y ELEGIMOS UN k_i CERCANO A M_i^2 Y ELEGIMOS H_i TAL QUE NO COLISIONE

$$- M_5 = 2, K = 5$$

$$- M_6 = 2, K = 5$$

- GENERO UNAS NUEVAS TABLAS
- GENERO UNAS NUEVAS FUNCIONES DE H QUE MANDEN A LUGARES DISTINTOS.

PUEDE SUCEDER QUE EL k MINIMO NO CUMPLA PARA NINGUN CASO, ENTONCES PASAMOS AL SIGUIENTE PRIMO

FINALMENTE LA TABLA NOS QUEDA:

K	CLAVES	HASH 2	SEGUNDA TABLAS
0			
1	1	X MOD 1	{1}
2	9	X MOD 1	{9}
3			
4			
5	5, 12	X MOD 5	{5, -, 12, --}
6	6, 13	2X MOD 5	{-, 13, 6, --}

ESTA ESTRUCTURA NOS ASEGURA O(1). YA QUE, HASHEO EL ELEMENTO, ME FIJO LA SEGUNDA FUNCIÓN DE HASH, LA APLICO Y OBTENGO LA SEGUNDA TABLA.

PARA PODER APLICARLO CONOCÉMOS TODAS LAS CLAVES Y TOLERAR LA REDIMENSIÓN. PERO ESTE HASH PERFECTO NO ES MÍNIMO, YA QUE LA PRIMERA TABLA TIENE POSICIONES VACIAS.

HASHING PERFECTO Y MINIMO

EL HASHING PERFECTO Y MINIMO NOS ASEGURA:

- $O(1)$ PARA CONSULTAS
 - $O(M)$ DE ESPACIO.

ESTO ES POSIBLE SÓLO SI CONOCEMOS CUÁLES SON TODAS LAS CLAVES QUE QUEREMOS ALMACENAR EN LA TABLA Y LOS DATOS SON ESTÁTICOS. CUANDO TODO ESTO SUCEDE ES POSIBLE DESARROLLAR UNA SOLUCIÓN QUE MAPEE CADA CLAVE A UNA ÚNICA POSICIÓN.

ELEMENTOS A ALMACENAR { 1, 3, 11, 19, 13 }

NECESITAMOS UNA FAMILIA DE FUNCIONES DE HASHING UNIVERSALES QUE PODAMOS PARAMETRIZAR CON UN NÚMERO ENTERO $H(i, x)$ ESTA FAMILIA. NOS GENERAN UN NÚMERO ENTRE 0 Y $N-1 \rightarrow$ CANT DE CLAVES.

EN NUESTRO CASO, PARA REALIZAR EL EJEMPLO VAMOS A USAR $H(i, x) = (x \bmod p_i) \bmod M$

SIENDO $p_i = \{7, 11, 13, \dots\}$ (NUMEROS PRIMOS)

• 1 HASHEAMOS TODOS LOS ELEMENTOS

CON $H(0, x) = (x \bmod 7) \bmod 5$



PRIMER NUMERO PRIMO EN p_i

CANT DE CLAVES

ELEMENTOS	$H(0, x)$	$H(1, x)$
1	1	1
3	3	
11	4	
19	0	
13	1	2

• 3 PODEMOS OBSERVAR UNA COLISION EN LA POS 1, POR LO TANTO A ESTOS DOS ELEMENTOS LE APPLICAMOS:

$H(1, x) = (x \bmod 11) \bmod 5$

SEGUNDO N.P EN p_i

• 2 BITMAP → COLISION

1	2	2	3	4
---	---	---	---	---

EL BITMAP SIRVE PARA LLEVAR LA CUENTA DE LAS POSICIONES OCUPADAS

EN ESTE CASO HUBO UNA SOLA COLISION, PERO DE HABER MAS COMENZAMOS POR EL BUCKET DE MAS COLISIONES.

• 4 DESPUES DE APLICAR $H(1, x)$ PODEMOS VER QUE NO HAY COLISIONES

BITMAP

0	1	2	3	4
---	---	---	---	---

NO HAY COLISIONES

• 5 EN UNA TABLA G ANOTAMOS LOS VALORES i QUE SE USO PARA DESOLVER x .

TABLA G	
POSICION	
0	- 0
1	1
2	
3	- 3
4	- 4

ENTONCES, PARA BUSCAR UNA CLAVE USANDO ESTA FUNCION, PRIMERO HACEMOS $H(0, x)$ ESTO NOS DA UNA POSICION EN LA TABLA G. ACCEDIENDO A LA TABLA SI EL NUM ES NEGATIVO, EL VALOR ABSOLUTO NOS DA EL RESULTADO EN LA TABLA DE HAS. SI EL NUM ES POSITIVO (z), EL VALOR DE LA TABLA DE HASH ES $H(z, x)$.

CUCKOO HASHING

ES UN ALGORITMO BASADO EN MULTIPLES FUNCIONES DE HASHING.

CADA FUNCIÓN DE HASHING VA A DEFINIR UNA POSICIÓN ALTERNATIVA PARA NUESTRO DATO EN UNA TABLA DE HASH ALMACENAMOS EL DATO EN LA POSICIÓN INDICADA POR $H_1(x)$

SI EL LUGAR ESTA OCUPADO, LO REUBICAMOS EN SU POSICIÓN ALTERNATIVA POR $H_2(x)$ Y ALMACENAMOS LO DSEADO. ASÍ SUCESSIVAMENTE HASTA TERMINAR CON TODOS. (ES POSIBLE ENTRAR EN UN **LOOP**) → **DEHASHEAR**

LOOKUP(x)

RETORN $T_1[H_1(x)] \vee T_2[H_2(x)]$

END

CUCKOO RECARGADO

ES LA SOLUCIÓN AL PROBLEMA DE DEMASHEADO (FACTOR DE CARGA BAJO). EN PRINCIPIO PODEMOS JUNTAR LAS DOS TABLAS EN UNA, TENIENDO EN CUENTA QUE FUNCIÓN DE HASH ($H_1(x)$, $H_2(x)$) SE UTILIZO.

— ADEMÁS, PODEMOS USAR UN **STASH** PARA GUARDAR LOS DATOS PROBLEMATICOS Y NO TENER QUE DEHASHEAD

	A	B	C	D
H1	5	3	3	5
H2	4	4	2	4

→ CLAVES

• 1 INSEBTAMOS LAS CLAVES EN ORDEN.

1 → INSEBTO A ✓

2 → INSERTO B ✓

3 → INSERTO C ✗ OCUPADO!

MUEVO B AL VALOR DE H_2 Y EN 3 PONGO C.

4 → INSEBTO D ✗ OCUPADO!

MUEVO A AL T2 Y COLOCO D, PERO

SUCEDA LO MISMO PARA A! ASI QUE

MUEVO B A 3, PERO TMB ESTA OCUPADO

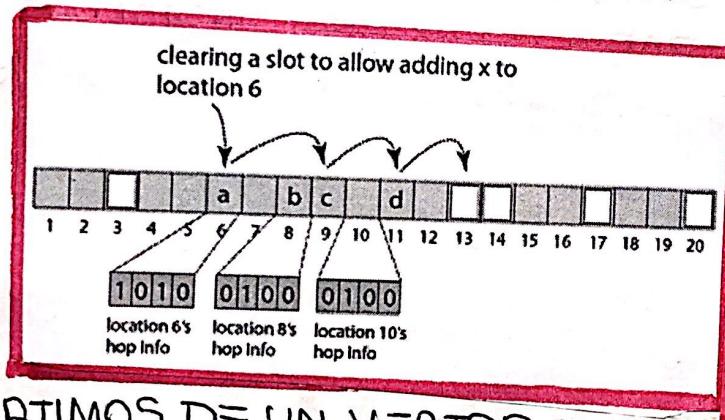
ASI QUE MUEVO C.

TABLA 1	
POS	ELEMENTO
0	
1	
2	
3	B/C/B
4	
5	A/D

TABLA 2	
POS	ELEMENTO
0	
1	
2	C
3	
4	B/A
5	

HOPSCOTCH HASHING

ESTE MÉTODO TIENE COMO OBJETIVO SOLUCIONAR ALGUNO DE LOS PROBLEMAS DE BUSQUEDA LINEAL Y EL ENCADENAMIENTO DE SINONIMOS. QUEREMOS GARANTIZAR $O(1)$ PARA LAS BUSQUEDAS, Y QUE LA ZONA EN LA CUAL TENEMOS QUE BUSCAR SEA IGUAL O MUY CERCANA A LA INDICADA POR $H(x)$, PARA FAVORDECER EL USO DE MEMORIA.



EJEMPLO

SI NO SE PUEDE REUBICAR, HAY QUE **REHASHEAD**

PARTIMOS DE UN VECTOR DE M ELEMENTOS, QUE ES NUESTRA TABLA DE HASH, UN K QUE INDICA QUE TAN LEJOS PODEMOS GUARDAR UN DATO RESPECTO DE $H(x)$. PARA RESOLVER COLISIONES:

SITODAS LAS POSICIONES ($H(x), H(x)+1 \dots H(x)+K-1$) ESTAN OCUPADAS, BUSCAMOS LINEALMENTE HASTA ENCONTRAR UNA POSICIÓN LIBRE Y APARTIR DE ESTA RECORDEMOS HACIA ATRÁS VIENDO SI PODEMOS IR MOVIENDO DATOS HACIA ADELANTE DE FORMA DE LIBERAR UNA POSICIÓN VALIDA PARA NUESTRO DATO.

→ EN NUESTRO EJEMPLO $M=21, K=4$. LAS POSICIONES GRISAS ESTAN OCUPADAS.

OBSEVAMOS EL CASO DE $H(x)=6$, LAS POSICIONES 6, 7, 8 Y 9 ESTAN OCUPADAS, POR LO TANTO BUSCAMOS LA PRÓXIMA LIBRE → 13. BUSCANDO HACIA ATRÁS VEMOS QUE $H(D)=11$, D LO PODEMOS MOVER A 13, Y $H(C)=9$, TMB PODEMOS MOVER A C A 11, LIBERANDO LA POS 9, PARA PODER INGRESAR $H(x)=6$ EN 9.

ITI Isla de S D Av Rivadavia 8338	4861-9597	■ Ida G B de Junin 1266	4541-9133	■ Josefina Terero 2042	4582-1828	■ Vicente M Sastré 5572	4567-2440	■ Mario Gomez
Juan B Pasco 1079	4672-1971	■ Juan M B Encalada 3190	4941-7888	■ Liliana C Esnola 615	4783-2986	■ Lidia L 1039	4583-8782	■ RIDI EXPORT
Iglia F G de Av M Campos 1640	4783-2986	■ Lidia L 1039	4583-8782	■ Lidia L 1039	4583-8782	■ Lidia L 1039	4805-0660	■ Au Ranta Fe 931
Ula M Terrada 250	46*							
Maria I Av Directorio 3028	46							
Maria R E Pasco 1083	49							
Marta A Cachimayo 167	44							
Teresa J Melincus 6172	45							
OVER Roberto Palpa 2481	45							
Roberto Zabala 2250	4							
OVERI Juan F Virrey D Pino 1725	4							
Juan F Virrey D Pino 1739	4							
Super! 1459	4							

FEATURE HASHING

THE FEATURDE HASHING, TAMBIÉN CONOCIDO COMO THE HASHING TRICK, TIENE COMO OBJETIVO REDUCIR EL TAMAÑO DE VECTORES (QUE CONTENGAN DATOS) SIN PERDER EL PODER DE REPRESENTACIÓN. (REDUCIR TAMAÑO → BAJAR LA DIMENSIÓN)

TIENE COMO PROPIEDAD

- MANTIENE LA DISTANCIAS ENTRE LOS DATOS.
- SE PUEDE USAR PARA CUALQUIER VARIANTE DE DATOS REPRESENTADOS EN FORMA DE VECTOR.

TEOREMA DE JOHNSON - LINDEN STRAUSS

ESTE TEOREMA JUSTIFICA DE FORMA TEÓRICA QUE ES POSIBLE TRABAJAR CON UN SET DE DATOS EN MENOS DIMENSIONES QUE LAS QUE TIENE ORIGINALMENTE.

PARA TODO CONJUNTO DE N DATOS EN D DIMENSIONES Y UNA CIERTA TASA DE ERROR ϵ ($0 < \epsilon < 1$) EXISTE UNA PROYECCIÓN $F: \mathbb{R}^D \rightarrow \mathbb{R}^k$ QUE CUMPLE:

$$(1-\epsilon) \|u-v\| \leq \|F(u)-F(v)\| \leq (1+\epsilon) \|u-v\|$$

LO QUE ESTO DICE QUE ES POSIBLE REPRESENTARLOS EN EL ORDEN DE $O(\log(N))$, MANTENIENDO LA DISTANCIA ENTRE DATOS.

¿QUE PROYECCIÓN VAMOS A UTILIZAR PARA ESTO?

- UNA MATRIZ DONDE SOLO UN ELEMENTO PODRÍA SER 1.

LA NUEVA DIMENSIÓN i ESTÁ COMPUESTA POR LA SUMA DE LAS DIMENSIONES ORIGINALES j QUE HASHEAN A i .

- MIS DATOS TIENEN 7 DIMENSIONES

$$\text{Y TENEMOS } h(x) = (3x + 1 \bmod 5) \bmod 4$$

DIM FINAL QUE QUIERO

CON ESTA $h(x)$ VAMOS A GENERAR LA MATRIZ PARA MAPEAR LA DIM ORIGINAL A LA NUEVA.

- SI MULTIPLICO MIS DATOS POR LA MATRIZ OBTENDO LAS NUEVAS DIMENSIONES

DATA A REDUCIR $V = [1, 2, 3, 4, 5, 6, 7] \rightarrow i$

$$\text{DIM}(V) = 7$$

HASHEO ESTOS VALORES

$$h(x) = (3x + 1 \bmod 5) \bmod 4$$

PARA FORMAR EL VECTOR REDUCIDO LO QUE VOY A HACER ES:

- PARA LA POS 0, SUMO SUS DIMENSIONES

ORIGINALES QUE HASHEADON A CERO

$$\text{VECTOR REDUCIDO} = [2+4+7, 1+6, 3, 5]$$

EN ESOS NUMEROS COLOCO UN 1 EN LA MATRIZ.

EL VECTOR REDUCIDO SE PODEIA OBTENER MULTIPLICANDO

$$V * M_A = V_R$$

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = M_A$$

"LA NUEVA DIMENSIÓN i , ESTA COMPUESTA POR LA SUMA DE LAS DIMENSIONES ORIGINALES j QUE HASHEAN A i "

ESTO LO PODEMOS ESCRIBIR COMO

$$\phi^{(H)}(x) \times \text{PASADO POR FEATURE HASHING}$$

$$\phi_i^{(H)}(x) = \sum_{j: H(j)=i} x_j \quad \text{LOS NUEVOS COMPONENTES DEL VECTOR REDUCIDO.}$$

ESTA PROYECCIÓN QUE CREAMOS NO CUMPLE CON EL TEOREMA DE JOHNSON - LUNDENSTRAUSS, POR LO QUE LA PODEMOS ACOMODAR DE LA SIGUIENTE FORMA, QUE ES AGREGANDO OTRA FUNCIÓN HASH QUE PERMITA A LA MATRIZ TENER VALORES DE -1

$$\phi_i^{(H, \varepsilon)}(x) = \sum_{j: H(j)=i} \varepsilon(j) x_j$$

$\varepsilon : x \rightarrow \{-1, 1\}$
↓ NOS DA UN SIGNO AL 1 PARA LA MATRIZ
OSEA EN EL RED SE RESTA.

- EL PRODUCTO INTERNO ENTRE DOS DATOS NOS DA UN INDICIO DE SIMILARIDAD. EN GENERAL, ES CÁRDIL CALCULARLO CUANDO TENEMOS MUCHOS PUNTOS DE DIMENSIONES MUY GRANDES. EN ESTE CASO PODEMOS UTILIZAR FEATURE HASHING PARA CALCULAR UN P.I. APROXIMADO.