

## **Simhashing**

# **Informe de la Quinta Práctica de Laboratorio**

## **Detección de duplicados mediante el uso del simhash**

**Diego Moreno y Daniel Lagos**

**Grupo 13 CAIM**

**Universitat Politècnica de Catalunya**

Octubre 2025

## Índice

<b>Ejercicio inicial.....</b>	<b>3</b>
Objetivos.....	3
Procedimiento y Resultado.....	3
<b>Función simhash.....</b>	<b>4</b>
Objetivos.....	4
Resultados.....	4
<b>Ordenar documentos en tablas de hash.....</b>	<b>4</b>
Objetivos.....	4
Resultados.....	4

# Ejercicio inicial

## Objetivos

Supón que un cierto documento **d** contiene dos entradas que no son cero que corresponden a los términos **bit** y **coin** respectivamente.

	<b>bit</b>	<b>coin</b>
<b>peso td-idf</b>	0'4	1'2
<b>código hash md5</b>	1111	1001

Computa el simhash binario del documento del enunciado con **k** = 1 y **m** = 1

## Procedimiento y Resultado

Dados los códigos hash del enunciado,  $h_{\text{bit}} = \{1, 1, 1, 1\}$  y  $h_{\text{coin}} = \{1, 0, 0, 1\}$ .

- **Paso 1:** Convertir cada hash  $h$  a un vector  $h'$ , tal que todo 0 en  $h$  es un -1 en  $h'$ .

$$h'_{\text{bit}} = \{1, 1, 1, 1\} = h_{\text{bit}}$$

$$h'_{\text{coin}} = \{1, -1, -1, 1\}$$

- **Paso 2:** Multiplicamos cada vector  $h'$  con el peso de su respectivo término.

$$w_{\text{bit}} * h'_{\text{bit}} = 0'4 * \{1, 1, 1, 1\} = \{0'4, 0'4, 0'4, 0'4\}$$

$$w_{\text{coin}} * h'_{\text{coin}} = 1'2 * \{1, -1, -1, 1\} = \{1'2, -1'2, -1'2, 1'2\}$$

- **Paso 3:** Hacemos la suma del simhash

$$(w_{\text{bit}} * h'_{\text{bit}}) + (w_{\text{coin}} * h'_{\text{coin}}) = \{0'4, 0'4, 0'4, 0'4\} + \{1'2, -1'2, -1'2, 1'2\} = \\ = \{1'6, -0'8, -0'8, 1'6\}$$

- **Paso 4:** Convertimos los valores positivos en bits 1 y los negativos en 0.

$$\{1'6, -0'8, -0'8, 1'6\} \Rightarrow \{x_1 > 0, x_2 < 0, x_3 < 0, x_4 > 0\} \Rightarrow \{1, 0, 0, 1\}$$

- **Resultado:** Como hemos previsto, el resultado es 1001.

# Función simhash

## Objetivos

Escribir la función simhash que nos provee el documento del enunciado.

Hay que tener en cuenta que cada bit del simhash codifica qué lado de la representación del documento tf-idf cae en una proyección pseudoaleatoria.

## Resultados

```
def _simhash(id, b):

    sum_vector = np.zeros(b)
    doc_weights = corpus.get(id, {})

    for term, weight in doc_weights.items():
        term_hash_str = _termhash(term, b)
        h_prime = np.array([1 if bit == '1' else -1 for bit in
term_hash_str])
        sum_vector += (h_prime * weight)

    final_hash_bits = ['1' if val > 0 else '0' for val in sum_vector]
    return "".join(final_hash_bits)
```

Esta función calcula el simhash de un solo documento basándose en los pesos tf-idf de sus términos. Primero inicializa un vector de acumulación **sum\_vector** con ceros del tamaño del hash **b**. Luego para cada término y su peso en el documento obtiene un hash binario para ese término, lo convierte a un vector de +1s y -1s **h\_prime** y multiplica este vector por el peso del término. El resultado se suma al vector de acumulación y después de sumar las contribuciones ponderadas de todos los términos la función convierte el **sum\_vector** final en un bitstring en el cual cada entrada positiva se convierte en '1' y cualquier entrada negativa o cero se convierte en '0' devolviendo así el simhash binario del documento

# Ordenar documentos en tablas de hash

## Objetivos

1. Escribir un código basado en simhashes que añade los documentos en sus respectivos buckets de las tablas de Ish hash a lo largo de m repeticiones. Cuando estas tablas se hayan llenado, encuentra todos los potenciales pares de candidatos de casiduplicados. Analiza entre los elementos de cada par su similitud real de coseno, y utiliza este dato para determinar los positivos reales y descartar los falsos positivos.
2. Computa la velocidad del método de detección de falsos positivos para diferentes valores de **m** y **k** (**k** poseerá valores  $\geq 10$ ).

## Resultados

```
for doc_id, hash_str in simhash.items():
    if hash_str is None or len(hash_str) != B: continue

    for m_index in range(M):
        start_index = m_index * K
        end_index = (m_index + 1) * K
        chunk = hash_str[start_index:end_index]

        table = hash_tables[m_index]
        if chunk not in table:
            table[chunk] = []
        table[chunk].append(doc_id)
```

El código itera sobre cada documento y su simhash ya calculado. Para cada simhash divide el hash completo en M trozos más pequeñas cada una de tamaño K bits. Luego trata cada trozo como una clave para una tabla hash específica. Coloca el ID del documento en la lista (**bucket**) correspondiente a ese trozo en la tabla hash apropiada. Al final de este proceso, cualquier par de documentos que haya caído en el mismo bucket en al menos una de las M tablas se considera un candidato potencial a ser un duplicado.

Hemos experimentado con el corpus otorgado por el enunciado para diferentes valores de K y M para obtener tiempos de ejecución en las diferentes fases de análisis del algoritmo y hemos calculado usando la similitud de coseno si los resultados del lsh son parecidos a la realidad.

### resultados del análisis (M=2, K=10)

tiempo de ejecución (fases LSH): 1.2303 s

tiempo de ejecución (verificación): 56.9549 s

tiempo total: 58.1852 s

total pares candidatos: 3589936

verdaderos positivos (sim  $\geq 0.9$ ): 16745

falsos positivos (sim  $< 0.9$ ): 3573191

top 10 pares de duplicados verdaderos

1. Par (27236, 52220) -> Similitud: 1.000000
2. Par (38386, 56286) -> Similitud: 1.000000
3. Par (28061, 57224) -> Similitud: 1.000000
4. Par (3015, 49889) -> Similitud: 1.000000
5. Par (29821, 46270) -> Similitud: 1.000000
6. Par (20513, 56062) -> Similitud: 1.000000
7. Par (15786, 49788) -> Similitud: 1.000000
8. Par (19396, 47485) -> Similitud: 1.000000
9. Par (20513, 58055) -> Similitud: 1.000000
10. Par (28264, 45707) -> Similitud: 1.000000

### **resultados del análisis (M=3, K=10)**

tiempo de ejecución (fases LSH): 2.6291 s

tiempo de ejecución (verificación): 87.3794 s

tiempo total: 90.0085 s

total pares candidatos: 5407375

verdaderos positivos (sim  $\geq 0.9$ ): 17555

falsos positivos (sim  $< 0.9$ ): 5389820

top 10 pares de duplicados verdaderos

1. Par (27236, 52220) -> Similitud: 1.000000
2. Par (38386, 56286) -> Similitud: 1.000000
3. Par (28061, 57224) -> Similitud: 1.000000
4. Par (3015, 49889) -> Similitud: 1.000000
5. Par (29821, 46270) -> Similitud: 1.000000
6. Par (20513, 56062) -> Similitud: 1.000000
7. Par (15786, 49788) -> Similitud: 1.000000
8. Par (28264, 45707) -> Similitud: 1.000000
9. Par (21128, 42086) -> Similitud: 1.000000
10. Par (33915, 47742) -> Similitud: 1.000000

### **resultados del análisis (M=4, K=10)**

tiempo de ejecución (fases LSH): 3.3633 s

tiempo de ejecución (verificación): 116.5948 s

tiempo total: 119.9581 s

total pares candidatos: 7241413

verdaderos positivos (sim  $\geq 0.9$ ): 17976

falsos positivos (sim < 0.9): 7223437

top 10 pares de duplicados verdaderos

1. Par (27236, 52220) -> Similitud: 1.000000
2. Par (38386, 56286) -> Similitud: 1.000000
3. Par (28061, 57224) -> Similitud: 1.000000
4. Par (3015, 49889) -> Similitud: 1.000000
5. Par (29821, 46270) -> Similitud: 1.000000
6. Par (20513, 56062) -> Similitud: 1.000000
7. Par (15786, 49788) -> Similitud: 1.000000
8. Par (28264, 45707) -> Similitud: 1.000000
9. Par (21128, 42086) -> Similitud: 1.000000
10. Par (33915, 47742) -> Similitud: 1.000000

### **resultados del análisis (M=2, K=12)**

tiempo de ejecución (fases LSH): 1.1347 s

tiempo de ejecución (verificación): 14.0868 s

tiempo total: 15.2214 s

total pares candidatos: 919679

verdaderos positivos (sim >= 0.9): 16412

falsos positivos (sim < 0.9): 903267

top 10 pares de duplicados verdaderos

1. Par (19396, 47485) -> Similitud: 1.000000
2. Par (20513, 58055) -> Similitud: 1.000000
3. Par (28061, 57224) -> Similitud: 1.000000
4. Par (27236, 52220) -> Similitud: 1.000000
5. Par (3015, 49889) -> Similitud: 1.000000
6. Par (38386, 56286) -> Similitud: 1.000000
7. Par (29821, 46270) -> Similitud: 1.000000
8. Par (28264, 45707) -> Similitud: 1.000000
9. Par (21128, 42086) -> Similitud: 1.000000
10. Par (20513, 56062) -> Similitud: 1.000000

### **resultados del análisis (M=3, K=12)**

tiempo de ejecución (fases LSH): 0.6911 s

tiempo de ejecución (verificación): 22.3335 s

tiempo total: 23.0246 s

total pares candidatos: 1392508

verdaderos positivos (sim >= 0.9): 17251

falsos positivos (sim < 0.9): 1375257

top 10 pares de duplicados verdaderos

1. Par (19396, 47485) -> Similitud: 1.000000

2. Par (20513, 58055) -> Similitud: 1.000000
3. Par (27236, 52220) -> Similitud: 1.000000
4. Par (38386, 56286) -> Similitud: 1.000000
5. Par (28264, 45707) -> Similitud: 1.000000
6. Par (21128, 42086) -> Similitud: 1.000000
7. Par (33915, 47742) -> Similitud: 1.000000
8. Par (56062, 58055) -> Similitud: 1.000000
9. Par (28061, 57224) -> Similitud: 1.000000
10. Par (3015, 49889) -> Similitud: 1.000000

### **resultados del analisis (M=4, K=12)**

tiempo de ejecución (fases LSH): 0.9911 s

tiempo de ejecución (verificación): 30.8851 s

tiempo total: 31.8762 s

total pares candidatos: 1899388

verdaderos positivos (sim >= 0.9): 17772

falsos positivos (sim < 0.9): 1881616

top 10 pares de duplicados verdaderos

1. Par (19396, 47485) -> Similitud: 1.000000
2. Par (20513, 58055) -> Similitud: 1.000000
3. Par (27236, 52220) -> Similitud: 1.000000
4. Par (38386, 56286) -> Similitud: 1.000000
5. Par (28264, 45707) -> Similitud: 1.000000
6. Par (21128, 42086) -> Similitud: 1.000000
7. Par (33915, 47742) -> Similitud: 1.000000
8. Par (56062, 58055) -> Similitud: 1.000000
9. Par (28061, 57224) -> Similitud: 1.000000
10. Par (3015, 49889) -> Similitud: 1.000000

### **resultados del analisis (M=2, K=18)**

tiempo de ejecución (fases LSH): 0.3815 s

tiempo de ejecución (verificación): 0.5285 s

tiempo total: 0.9100 s

total pares candidatos: 30853

verdaderos positivos (sim >= 0.9): 15201

falsos positivos (sim < 0.9): 15652

top 10 pares de duplicados verdaderos

1. Par (19396, 47485) -> Similitud: 1.000000
2. Par (20513, 56062) -> Similitud: 1.000000
3. Par (38386, 56286) -> Similitud: 1.000000
4. Par (29821, 46270) -> Similitud: 1.000000
5. Par (21128, 42086) -> Similitud: 1.000000

6. Par (20513, 58055) -> Similitud: 1.000000
7. Par (28061, 57224) -> Similitud: 1.000000
8. Par (28264, 45707) -> Similitud: 1.000000
9. Par (41444, 47883) -> Similitud: 1.000000
10. Par (15786, 49788) -> Similitud: 1.000000

### **resultados del analisis (M=3, K=18)**

tiempo de ejecución (fases LSH): 0.1692 s

tiempo de ejecución (verificación): 0.7195 s

tiempo total: 0.8887 s

total pares candidatos: 40276

verdaderos positivos (sim >= 0.9): 16031

falsos positivos (sim < 0.9): 24245

top 10 pares de duplicados verdaderos

1. Par (19396, 47485) -> Similitud: 1.000000
2. Par (20513, 56062) -> Similitud: 1.000000
3. Par (38386, 56286) -> Similitud: 1.000000
4. Par (29821, 46270) -> Similitud: 1.000000
5. Par (21128, 42086) -> Similitud: 1.000000
6. Par (20513, 58055) -> Similitud: 1.000000
7. Par (28061, 57224) -> Similitud: 1.000000
8. Par (28264, 45707) -> Similitud: 1.000000
9. Par (41444, 47883) -> Similitud: 1.000000
10. Par (15786, 49788) -> Similitud: 1.000000

### **resultados del analisis (M=4, K=18)**

tiempo de ejecución (fases LSH): 0.3166 s

tiempo de ejecución (verificación): 0.8575 s

tiempo total: 1.1741 s

total pares candidatos: 48027

verdaderos positivos (sim >= 0.9): 16536

falsos positivos (sim < 0.9): 31491

top 10 pares de duplicados verdaderos

1. Par (19396, 47485) -> Similitud: 1.000000
2. Par (20513, 56062) -> Similitud: 1.000000
3. Par (38386, 56286) -> Similitud: 1.000000
4. Par (29821, 46270) -> Similitud: 1.000000
5. Par (21128, 42086) -> Similitud: 1.000000
6. Par (20513, 58055) -> Similitud: 1.000000
7. Par (28061, 57224) -> Similitud: 1.000000
8. Par (28264, 45707) -> Similitud: 1.000000
9. Par (41444, 47883) -> Similitud: 1.000000

10. Par (15786, 49788) -> Similitud: 1.000000

### **resultados del análisis (M=2, K=27)**

tiempo de ejecución (fases LSH): 0.1510 s

tiempo de ejecución (verificación): 0.2414 s

tiempo total: 0.3924 s

total pares candidatos: 14002

verdaderos positivos (sim >= 0.9): 13903

falsos positivos (sim < 0.9): 99

top 10 pares de duplicados verdaderos

1. Par (21128, 42086) -> Similitud: 1.000000
2. Par (20513, 58055) -> Similitud: 1.000000
3. Par (42464, 56475) -> Similitud: 1.000000
4. Par (26060, 7255) -> Similitud: 1.000000
5. Par (28061, 57224) -> Similitud: 1.000000
6. Par (19396, 47485) -> Similitud: 1.000000
7. Par (28264, 45707) -> Similitud: 1.000000
8. Par (27236, 52220) -> Similitud: 1.000000
9. Par (41444, 47883) -> Similitud: 1.000000
10. Par (15786, 49788) -> Similitud: 1.000000

## **Conclusiones**

Los resultados confirman la regla que nos daba el enunciado sobre el tiempo de ejecución respecto al valor K, y es que el tiempo disminuye cuando K aumenta. Esto se debe a que si la K es mas grande, los grupos de hashes a comparar son más grandes y, por tanto, menos a calcular y comparar.