Tema 8: **Hashing i Cerca**

Resum del tema 8

- Concepte de col·lisió
- Propietats de les funcions hash
- SimHash
  - Índex de Jaccard $J(A/B) = A \cap B / A \cup B$

Cerca i llistes

Considerem el problema de buscar un determinat valor en una llista.

- Si la llista no està ordenada, la complexitat és `O(n)`.
  - Si el valor no hi és, farem `n` comparacions.
  - Si el valor hi és, farem una mitja de `n/2` comparacions.
- Si la llista està ordenada farem cerca binaria:
  - La cerca binaria té una complexitat `O(log n)`
  - La complexitat és la mateixa tant si hi és com si no.

Com podem millorar-ho?

Cerca i llistes

Considerem el problema de buscar un determinat valor en una llista.

- Si la llista no està ordenada, la complexitat és `O(n)`.
  - Si el valor no hi és, farem `n` comparacions.
  - Si el valor hi és, farem una mitja de `n/2` comparacions.
- Si la llista està ordenada farem cerca binaria:
  - La cerca binaria té una complexitat `O(log n)`
  - La complexitat és la mateixa tant si hi és com si no.

Com podem millorar-ho?

Suposem que tenim una *funció màgica* que, donat un valor a cercar, ens digués a quina posició de la llista mirar, i:

- Si hi és, és que estava a la llista.
- Si no hi és, és que no hi era.

Aquesta funció *màgica* tindria una complexitat `O(1)` per fer cerca en una llista!

Hashing i Cerca

Imaginem que volem guardar en una llista cada un dels elements d'un **determinat subconjunt (inicialment desconegut)** d'adreces IP d'entre les possibles `2^48` adreces IP.

Imaginem que sabem que el nostre subconjunt no serà mai més gran de 250 noms, i per tant creem una llista `a` de 250 posicions.

Imaginem que tenim una funció `h` tal que donada una IP és capaç de generar un index `i` de la llista. Llavors, i suposant que subconjunt ens arriba de manera seqüencial, anem guardant el nom de les IP a la posició `a[i]` a mesura que van arribant.

En el cas que `h` assigni el mateix índex a dues adreces (aquest cas s'anomena **col·lisió**) enlloc de guardar directament l'adreça IP a la llista, hi guardem una llista amb l'adreces IP que comparteixen el mateix índex.

Aquest esquema tindria sentit per fer cerques molt ràpides sempre i quant les llistes de col·lisió fossin petites. Per què?

The designers of TCP/IP defined an IP address as a 32-bit number and this system, known as Internet Protocol Version 4 (IPv4), is still in use today. However, due to the enormous growth of the Internet and the predicted depletion of available addresses, a new addressing system (IPv6), using 128 bits for the address, was developed in 1995, standardized by RFC 2460 in 1998, and is in world-wide production deployment.

An IPv6 address (dotted-decimal notation)

$$\begin{array}{cccccc} 172\,. & 16\,. & 254. & 1\,. & 8 & .124 \\ \Downarrow & \Downarrow & \Downarrow & \Downarrow & \Downarrow & \Downarrow \end{array}$$

$\underbrace{10101100}_{}.\underbrace{00010000}_{}.11111110.00000001.00001000.01111100$

One byte = Eight bits

Forty-eight bits (6 * 8); or 6 bytes

Hashing i Cerca

Problema: Donades unes dades d'entrada, com podem crear una funció `h` tal que els índexos que crea minimitzin la probabilitat de col·lisió de les dades en la llista que creem?

Aquest és el rol de les funcions **hash**: a l'exemple, una funció `h` que generi indexos per adreces IP.

L'índex assignat a una adreça `x` és `h(x)`, i llavors `x` s'emmagatzema a la posició `h(x)` de la llista.

Hashing i Cerca

Les funcions hash han de ser:

- *Aleatòries*, en el sentit que han de *distribuir uniformement* les dades entre tots els noms curts possibles.
- *Consistents*, en el sentit d'assignar a cada ítem sempre el mateix lloc.

Imaginem que usem una funció que assigna l'adreça al nombre corresponent als últims 8 bits: `h(128.32.168.24.7.80)=80`

És una bona funció hash?

Hashing i Cerca

Les funcions hash han de ser:

- *Aleatòries*, en el sentit que han de *distribuir uniformement* les dades entre tots els noms curts possibles.
- *Consistents*, en el sentit d'assignar a cada ítem sempre el mateix lloc.

Imaginem que usem una funció que assigna l'adreça al nombre corresponent als últims 8 bits: `h(128.32.168.24.7.80)=80`

És una bona funció hash?

No!

Hashing i Cerca

Les funcions hash han de ser:

- *Aleatòries*, en el sentit que han de *distribuir uniformement* les dades entre tots els noms curts possibles.
- *Consistents*, en el sentit d'assignar a cada ítem sempre el mateix lloc.

Imaginem que usem una funció que assigna l'adreça al nombre corresponent als últims 8 bits: `h(128.32.168.24.7.80)=80`

És una bona funció hash?

No!

Les posicions corresponents als nombres baixos estaran molt plenes, atès que a la pràctica aquests nombres s'assignen successivament.

I si assigna l'adreça al nombre corresponent als primers 8 bits?

Hashing i Cerca

Les funcions hash han de ser:

- *Aleatòries*, en el sentit que han de *distribuir uniformement* les dades entre tots els noms curts possibles.
- *Consistents*, en el sentit d'assignar a cada ítem sempre el mateix lloc.

Imaginem que usem una funció que assigna l'adreça al nombre corresponent als últims 8 bits: `h(128.32.168.24.7.80)=80`

És una bona funció hash?

No!

Les posicions corresponents als nombres baixos estaran molt plenes, atès que a la pràctica aquests nombres s'assignen successivament.

I si assigna l'adreça al nombre corresponent als primers 8 bits?

Tampoc! Imaginem que la majoria d'adreces són d'Àsia.

Hashing i Cerca

Les funcions hash han de ser:

- *Aleatòries*, en el sentit que han de *distribuir uniformement* les dades entre tots els noms curts possibles.
- *Consistents*, en el sentit d'assignar a cada ítem sempre el mateix lloc.

Imaginem que usem una funció que assigna l'adreça al nombre corresponent als últims 8 bits: `h(128.32.168.24.7.80)=80`

És una bona funció hash?

No!

Les posicions corresponents als nombres baixos estaran molt plenes, atès que a la pràctica aquests nombres s'assignen successivament.

I si assigna l'adreça al nombre corresponent als primers 8 bits?

Tampoc! Imaginem que la majoria d'adreces són d'Àsia.

Si les adreces vinguessin de forma **uniformement distribuïda en el conjunt total** no hi hauria problema. Però això no passa mai.

Hashing i Cerca

Atès que fem una correspondència entre `2^48` adreces i una llista de `250` noms hi ha d'haver una col·lecció de `2^48/250` = 1 bilió d'adreces que potencialment col·lisionaran.

Però podem seguir el següent esquema per **minimitzar les col·lisions**: escollir de forma aleatòria una funció per alguna classe de funcions hash

> Per definir una bona funció hash haurem de poder demostrar que, sigui quin sigui el conjunt de dades d'entrada, la majoria de funcions escollides generaran poques col·lisions!

Hashing i Cerca

Anem a definir una classe de funcions que puguem escollir aleatòriament:

- Suposem que creem una llista de `n=257` noms (enlloc de `256`). `257` és un nombre primer. Ha de ser un nombre primer si volem que sigui una bona funció hash!
- Representem una adreça com una `6`-tupla d'enters mòdul `n: x = (x1,x2,x3,x4,x5,x6)`

Definim la classe de funcions hash `h: IP -> n` de la següent manera:

- Escull 6 nombres qualsevol mòdul `n=257`. Per exemple: `87, 23, 125, 4, 17, 231`.
- Transforma l'adreça `(x1,x2,x3,x4,x5,x6)` a `h(x1,x2,x3,x4,x5,x6)= (87*x1 + 23*x2 + 125*x3 + 4*x4 + 17*x5 + 231*x6) mod 257`.

Hashing i Cerca

Dit d’una altra manera, per qualsevol conjunt de coeficients

$$a_1, \ldots, a_6$$

d'enters amb un valor entre `0` i `n-1`, podem definir la següent funció hash:

$$h_a(x_1, \ldots, x_6) = \sum_{i=1}^{6} a_i \cdot x_i \bmod n$$

Hashing i Cerca

Es pot demostrar que donada qualsevol parella d'adreces IP `x = (x1,x2,x3,x4,x5,x6)` i `y = (y1,y2,y3,y4,y5,y6)`, si els coeficients (`a1,a2,a3,a4,a5,a6`) s'escullen aleatòriament de `{0,1,...,n-1}`, llavors la probabilitat de que `h(x1,x2,x3,x4,x5,x6)=h(y1,y2,y3,y4,y5,y6)` és `1/n`.

És a dir, la mateixa probabilitat que si les adreces haguessin estat escollides de forma aleatòria.

Per tant, siguin quines siguin les adreces que arribin, la majoria de funcions escollides tindran poques col·lisions.

Hashing i Cerca

Quin és el temps de cerca d’una adreça?

= El càlcul de la funció hash + la cerca en la llista assignada al mateix nom.

Però només hi ha 250 adreces i la probabilitat de col·lisió és 1/257. Per tant, el nombre esperat d'ítems emmagatzemats al mateix nom és 250/257 (o sigui, pel nostre problema, menys de dos).

Hashing i Cerca

Recapitulem el que hem fet:

- Com que no teníem control sobre el conjunt de dades que ens arribava, hem escollit una funció `h` de forma uniformement aleatòria d'entre una família `H`. En el nostre cas:
  - Per escollir-la, hem escollit aleatòriament 6 nombres mòdul `n`.
- Hem dit que es pot demostrar que per dos ítems qualsevol `x` i `y`, aquests aniran al mateix lloc amb una probabilitat `1/n`, on `n` és el nombre de llocs.

Hashing i Cerca

Una família de funcions hash amb la capacitat de poder-se aplicar a qualsevol conjunt de dades es diu **universal**.

La seva aplicació a d'altres problemes és simple:

- Triem una taula de mida `n` tal que n sigui un nombre primer una mica més gran que el nombre d'elements de la taula (o millor encara, el doble).
- Assumim que el domini dels ítems és `N=n^k` (encara que ho sobreestimem).
- Llavors cada ítem es representa amb una `k`-tupla d'enters mòdul `n`.

Hashing i Cerca

Una família de funcions hash amb la capacitat de poder-se aplicar a qualsevol conjunt de dades es diu **universal**.

La seva aplicació a d’altres problemes és simple:

- Triem una taula de mida `n` tal que n sigui un nombre primer una mica més gran que el nombre d’elements de la taula (o millor encara, el doble).
- Assumim que el domini dels ítems és `N=n^k` (encara que ho sobreestimem).
- Llavors cada ítem es representa amb una `k`-tupla d’enters mòdul `n`.

Hi ha moltes families de funcions hash. Per triar la millor funció hash pel nostre problema hem d'examinar les dades i decidir quina és la millor funció hash a partir de les propietats de les dades.

Hashing i Cerca

Per exemple, quina funció fariem servir si les nostres dades **ja estan uniformement distribuides**?

Hashing i Cerca

Per exemple, quina funció faríem servir si les nostres dades **ja estan uniformement distribuides**?

Suposem que les nostres dades són enters `z` de rang `0..N-1`, uniformement distribuits, i que el nostre objectiu és assignar-lis un enter `h` en el rang `0..n-1`, tal que `n << N`.

Llavors la funció hash podria ser `h = z mod n` o `h = (z × n) ÷ N`.

Hashing i Cerca: enters no uniformement distribuits.

Quina funció fariem servir si les nostres dades són enters que **no estan uniformement distribuides**?

> Per exemple, suposem que són nombres de telèfon. Els nombres de telèfon tenen la propietat que els primers dígits no estan uniformement distribuits, pero els darrers si!

Suposem que 'n= 10000'. En aquest cas, la fòrmula `h = z mod n` encara funciona, però `h = (z × n) ÷ N` no (perquè depen molt dels dígits del principi)!

Hashing i Cerca: seqüències de longitud variable.

Per les dades de longitud variable necessitem una funció que depengui de tots els elements de la seqüència. Una bona estratègia en aquest cas és trencar la seqüència en troços petits i combinar-los un darrera l'altre:

```
def DEKHash(key):
    hash = len(key)
    for i in range(len(key)):
      hash = ((hash << 5) ^ (hash >> 27)) ^ ord(key[i])
    return hash
```

Exemple:

Exemple:

```
DEKHash('algorismica')
>>> 2910147705165117 49
```

a = 0011 1100 b = 0000 1101

^: *Binary XOR Operator*: copies the bit if it is set in one operand but not both. (a ^ b) will give 49 which is 0011 0001.

<<: *Binary Left Shift Operator*. The left operands value is moved left by the number of bits specified by the right operand. a << 2 will give 240 which is 1111 0000.

>>: *Binary Right Shift Operator*. The left operands value is moved right by the number of bits specified by the right operand. a >> 2 will give 15 which is 0000 1111.

Exemple:

```
def StringHash(a, m=257, C=1024):
    hash=0
    for i in range(len(a)):
        hash = (hash * C + ord(a[i])) % m
    return hash

StringHash('hola')
>>> 182
StringHash('adeu')
>>> 245
StringHash('adeu hol adeu')
>>> 208
StringHash('a')
>>> 97
```

`m` representa el nombre esperat d'ítems a guardar.

`C` és un nombre més gran que qualsevol `ord(c)`.

Aquí ho apliquem a caràcters, però ho podríem fer amb bigrames, etc.

Exemple:

```
def StringHash(a, m=257, C=1024):
    hash=0
    for i in range(len(a)):
        hash = (hash * C + ord(a[i])) % m
    return hash

diccionari
>>> ['the', 'and', 'to', 'of', 'a', 'I', 'in', 'was', 'he', 'that', 'it', 'his',
'her', 'you', 'as', 'had', 'with', 'for', 'she', 'not', 'at', 'but',
'be', 'my', 'on', 'have', 'him', 'is', 'said', 'me', 'which', 'by',
'so', 'this', 'all', 'from', 'they', 'no', 'were', 'if', 'would', 'or',
'when', 'what', 'there', 'been', 'one', 'could', 'very', 'an', 'who',
'them', 'Mr', 'we', 'now', 'more', 'out', 'do', 'are', 'up', 'their',
'your', 'will', 'little', 'than', 'then', 'some', 'into', 'any', 'well',
'much', 'about', 'time', 'know', 'should', 'man', 'did', 'like', 'upon',
'such', 'never', 'only', 'good', 'how', 'before', 'other', 'see', 'must',
'am', 'own', 'come', 'down', 'say', 'after', 'think', 'made', 'might',
'being', 'Mrs', 'again']
```

Exemple:

```
taula=[]
for i in diccionari:
    taula.append(StringHash(i))

taula
>>> [256, 184, 161, 172, 97, 73, 204, 89, 199, 136, 210, 74, 89, 67, 241,
91, 129, 17, 240, 147, 242, 188, 223, 199, 180, 179, 68, 209, 40, 179, 10,
243, 165, 103, 200, 101, 125, 185, 70, 196, 228, 184, 179, 201, 143, 190,
152, 248, 154, 236, 57, 113, 63, 139, 150, 99, 139, 225, 169, 158, 192,
103, 165, 82, 130, 114, 249, 84, 205, 101, 1, 195, 89, 241, 189, 181, 252,
95, 138, 131, 164, 256, 237, 54, 67, 7, 252, 206, 235, 125, 245, 150, 31,
81, 229, 188, 173, 178, 120, 207]
```

Exemple:

```
def StringHash(a, m=5749, C=1024):
    hash=0
    for i in range(len(a)):
        hash = (hash * C + ord(a[i])) % m
    return hash

taula
>>> [589, 4073, 3915, 4535, 97, 73, 4148, 209, 3115, 1260, 4154, 3276, 4928,
1816, 1710, 818, 3248, 4903, 4080, 5722, 1711, 2017, 2720, 2506, 4543, 5313,
3270, 4153, 4034, 2486, 827, 2740, 2891, 3702, 2033, 5606, 5361, 3520, 3663,
4140, 4550, 4547, 2883, 4542, 3800, 1880, 1192, 3283, 2589, 1705, 1624, 5349,
4225, 1228, 5725, 3805, 2626, 4778, 2421, 4940, 346, 2771, 809, 824, 1254,
5350, 5249, 4978, 4094, 3275, 1996, 3590, 4293, 2054, 2931, 620, 5727, 4991,
254, 2811, 1654, 3360, 5666, 3675, 1738, 2900, 1008, 1145, 1704, 4668, 4992,
4837, 2681, 2870, 2993, 3849, 4778, 2591, 3267, 5397]
```

SimHash i cerca per semblança.

Imaginem que tenim un conjunt d'adreces. Com podem decidir que són repetides?

Cas 1:

- `Burra Hotel, 5 Market Sq, Burra, SA, 5417`
- `Camping Country Superstore, 401 Pacific Hwy, Belmont North, NSW, 2280`

Cas 2:

- `One Stop Bakery, 1304 High St Rd, Wantirna, VIC, 3152`
- `One Stop Bakery, 1304 High Street Rd, Wantirna South, VIC, 3152`

SimHash i cerca per semblança.

Cas 3:

- `Park Beach Interiors, Showroom Park Beach Plaza Pacific Hwy, Coffs Harbour, NSW, 2450`
- `Park Beach Interiors, Showroom Park Beach Plaza Pacific Highway, Coffs Harbour, NSW, 2450`
- `Park Beach Interiors, Park Beach Plaza Pacific Hwy, Coffs Harbour, NSW, 2450`
- `Park Beach Interiors, 26 Park Beach Plaza, Pacific Hwy, Coffs Harbour, NSW, 2450`

SimHash i cerca per semblança.

Cas 4:

- `Weaver Interiors, 955 Pacific Hwy, Pymble, NSW, 207`
- `Weaver Interiors, 997 Pacific Hwy, Pymble, NSW, 2073`

Cas 5:

- `Gibbon Hamor Commercial Interiors, 233 Johnston St, Annandale, NSW, 2038`
- `Gibbon Hamor Development Planners, 233 Johnston St, Annandale, NSW, 2038`

Necessitem una manera de “comparar” aquestes adreces.

SimHash i cerca per semblança.

Les tècniques de **shingling** són una forma de generar un conjunt que pot ser usat com a representant del text.

Per exemple, podem usar bigrames de paraules:

- Els bigrames que representen `the cat sat on the` són `(the,cat)`, `(cat,sat)`, `(sat,on)`, `(on,the)`.

Llavors podem aplicar l'**índex de Jaccard** (el quocient entre la cardinalitat de la intersecció de dos conjunt i la cardinalitat de la unió) com a mesura de semblança entre adreces!

Veiem un exemple de com calcular l'índex de Jaccard. Tenim `A = {1,2,3}`, `B = {2,3,4}`, `C = {4,5,6}`. Si anomenem `J(A,B)` a l'índex de Jaccard entre `A` i `B`, llavors, `J(A,B) = 2/4 = 0.5`, perquè A intersecció B = 2, i A unió B = 4; i respectivament `J(A,C) = 0/6 = 0`, i `J(B,C) = 1/5 = 0.2`

A partir d'aquests índexs podem afirmar que els conjunts més semblants són `A` i `B` i els menys semblants són `A` i `C`.

SimHash i cerca per semblança.

Burra Hotel, 5 Market Sq, Burra, SA, 5417 es pot representar (amb bigrames de lletres) com: A={" 5", " B", " H", " M", " S", ", ", "17", "41", "5 ", "54", "A,", "Bu", "Ho", "Ma", "SA", "Sq", "a ", "a,", "ar", "el", "et", "ke", "l,", "ot", "q,", "ra", "rk", "rr", "t ", "te", "ur"}

Camping Country Superstore, 401 Pacific Hwy, Belmont North, NSW, 2280 es pot representar (amb bigrames de lletres) com: B={" 2", " 4", " B", " C", " H", " N", " P", " S", ", ", "01", "1 ", "22", "28", "40", "80", "Be", "Ca", "Co", "Hw", "NS", "No", "Pa", "SW", "Su", "W,", "ac", "am", "c ", "ci", "e,", "el", "er", "fi", "g ", "h,", "ic", "if", "in", "lm", "mo", "mp", "ng", "nt", "on", "or", "ou", "pe", "pi", "re", "rs", "rt", "ry", "st", "t ", "th", "to", "tr", "un", "up", "wy", "y ", "y,"}.

J(A,B)= 6/87 = 0.068

SimHash i cerca per semblança.

Cas 2

- A = One Stop Bakery, 1304 High St Rd, Wantirna, VIC, 3152
- B = One Stop Bakery, 1304 High Street Rd, Wantirna South, VIC, 3152
- J(A,B) = 46/57 = 0.807

Cas 3

- A = Park Beach Interiors, Showroom Park Beach Plaza Pacific Hwy, Coffs Harbour, NSW, 2450 + B = Park Beach Interiors, Showroom Park Beach Plaza Pacific Highway, Coffs Harbour, NSW, 2450
- C= Park Beach Interiors, Park Beach Plaza Pacific Hwy, Coffs Harbour, NSW, 2450
- D= Park Beach Interiors, 26 Park Beach Plaza, Pacific Hwy, Coffs Harbour, NSW, 2450
- J(A,B)=0.888, J(A,C)=0.861, J(A,D)=0.808, J(B,C)=0.760, J(B,D)=0.716, J(C,D)=0.932

SimHash i cerca per semblança.

Cas 4:

- A = Weaver Interiors, 955 Pacific Hwy, Pymble, NSW, 2073
- B = Weaver Interiors, 997 Pacific Hwy, Pymble, NSW, 2073
- J(A,B)= 43/49 = 0.877

Cas 5

- A = Gibbon Hamor Commercial Interiors, 233 Johnston St, Annandale, NSW, 2038
- B = Gibbon Hamor Development Planners, 233 Johnston St, Annandale, NSW, 2038
- J(A,B)= 49/76 = 0.644

SimHash i cerca per semblança.

Si volem buscar el parell d'adreces més semblants tenim un problema: hem de comparar cada element amb tots els elements i això és `O(n^2)`.

- 50 adreces són 1,225 comparacions.
- 500 adreces són 124,750 comparacions
- 2000 adreces són 1,999,000 comparacions
- 1,000,000 adreces són 499,999,500,000 comparacions.

Podem calcular un valor `Sim(A,B)` que "aproximi" `J(A,B)` amb menys complexitat que `O(n^2)`?

SimHash i cerca per semblança.

Per fer-ho necessitem una funció de hash que assigni codis semblants (en el sentit de la distància de Hamming) a ítems semblants.

Hash clàssic vs SimHash:

```
p1 = 'the cat sat on the mat'
p2 = 'the cat sat on a mat'
p3 = 'we all scream for ice cream'
hash(p1)
>>> 415542861
hash(p2)
>>> 668720516
hash(p3)
>>> 767429688
simhash(p1)
>>> 851459198
>>> 00110010110000000011110001111110
simhash(p2)
>>> 847263864
>>> 00110010100000000011100001111000
simhash(p3)
>>> 984968088
>>> 00111010101101010110101110011000
```

SimHash i cerca per semblança.

```
def hamming_distance(s1, s2):
    return sum(ch1 != ch2 for ch1, ch2 in zip(s1, s2))

zip('hola','hole')
>>> [('h', 'h'), ('o', 'o'), ('l', 'l'), ('a', 'e')]
hamming_distance('hola','hole')
>>> 1
```

SimHash i cerca per semblança.

L’algorisme SimHash genera un codi de la manera següent:

- Escollim una mida de la taula, per exemple 32 bits.
- Sigui `V=[0]*32`.
- Representem el text amb els bigrames:

  ```
  'the cat sat on the mat' =>
  {"th", "he", "e ", " c", "ca", "at", "t ", " s", "sa", " o", "on", "n ", " t", " m", "ma"}
  ```

- Fem el hash de cada característica amb una funció de 32 bits.

  `Hash(‘th’) = -502157718` `Hash(‘he’) = -369049682` …

- Per cada hash, si el bit `i` del hash és 1 llavors sumem 1 a `V[i]`; si el bit `i` del hash és 0 llavors restem 1 de `V[i]`.

- El bit `i` de simhash és 1 si `V[i] > 0` i 0 en els altres casos.

SimHash i cerca per semblança.

Simhash és útil perquè si la distància de Hamming entre els simhash de dues frases és petit, llavors el coeficient de Jaccard és alt. Però com ho usem pel nostre objectiu (sense fer `O(n^2)` comparacions)?

En el cas que 2 nombres tenen una distància de Hamming petita i la seva diferència està en els bits menys significatius, llavors això vol dir que són dos ítems que estan propers en una llista ordenada (vegeu taula a la següent diapositiva).

SimHash i cerca per semblança (taula)

A la taula es mostra un identificador correlatiu per cada cadena de bits (1..8), el simhash corresponent a cada cadena, la cadena pròpiament dita i la distància de hamming de la cadena actual amb l'anterior. Per ex. la cadena 3 té el simhash 2648 i té una distància de hamming de 11 respecte la cadena 2. A la dreta les mateixes cadenes s'ordenen pel simhash.

Sense ordenar

1	37586	1001001011010010	
2	50086	1100001110100110	7
3	2648	0000101001011000	11
4	934	0000001110100110	9
5	40957	1001111111111101	9
6	2650	0000101001011010	9
7	64475	1111101111011011	7
8	40955	1001111111111011	4

Calculem les distàncies de Hamming al nombre anterior.

Ordenats

4	934	0000001110100110	
3	2648	0000101001011000	9
6	2650	0000101001011010	1
1	37586	1001001011010010	5
8	40955	1001111111111011	6
5	40957	1001111111111101	2
2	50086	1100001110100110	9
7	64475	1111101111011011	9

Calculem les distàncies de Hamming al nombre anterior.

SimHash i cerca per semblança.

Per tant, només cal comprovar parelles adjacents de la llista per trobar la parella més semblant!

Això redueix la complexitat de `n*(n-1)/2` comparacions amb `O(n^2)` a `n` càlculs de la funció hash amb `O(n)` + una ordenació `O(n log n)` + `n` distàncies amb `O(n)` = la complexitat total és `O(n log n)`.

SimHash i cerca per semblança (exemple de taula)

A la taula es mostren 2 parelles de cadenes, la (3,6) i la (8,5) amb una distància de Hamming de 1 i de 2 respectivament. Queden juntes ordenades perquè el simhash també és semblant.

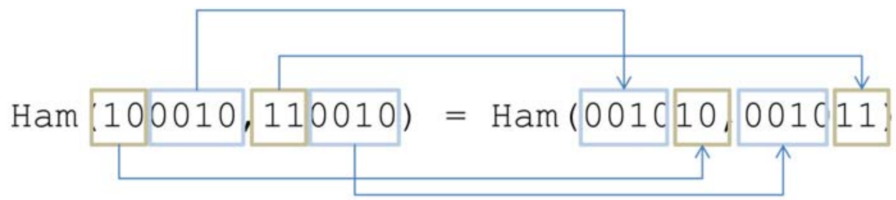
4	934	0000001110100110	
3	2648	0000101001011000	9
6	2650	0000101001011010	1
1	37586	1001001011010010	5
8	40955	1001111111111011	6
5	40957	1001111111111101	2
2	50086	1100001110100110	9
7	64475	1111101111011011	9

SimHash i cerca per semblança.

Però hi ha una parella semblant que ha deixat apartada: la formada per les cadenes 4 (0000001110100110) i la 2(1100001110100110), que tenen distància 2…

L'ordenació només ha ajuntat els que tenien bits poc significatius semblants…

Però podem aprofitar una propietat de la distància de Hamming: una permutació dels bits dels nombres preserva la distància!



Si permutem mitjançant la rotació dels bits tenim la mateixa distància però els bits més significatius passen a ser els menys significatius.

SimHash i cerca per semblança.

Però hi ha una parella semblant que ha deixat apartada, la (4,2), que té distància 2…

Rotem els bits a l’esquerra dues vegades i després ordenem:

4	3736	0000111010011000	
3	10592	0010100101100000	9
6	10600	0010100101101000	1
1	19274	0100101101001010	5
8	32750	0111111111101110	6
5	32758	0111111111110110	2
2	3739	0000111010011011	9
7	61295	1110111101101111	9

4	3736	0000111010011000	
2	3739	0000111010011011	2
3	10592	0010100101100000	11
6	10600	0010100101101000	1
1	19274	0100101101001010	5
8	32750	0111111111101110	6
5	32758	0111111111110110	2
7	61295	1110111101101111	6

SimHash i cerca per semblança.

Per tant, l'algorisme ha de desplaçar els bits `B` vegades (on `B` és la longitud en bits) i fer:

- Rotar els bits
- Ordenar
- Calcular la distància de Hamming entre files adjacents.

Com que segurament `B << n` el cost de l'algorisme és més semblant a `O(n log n)` que a `O(n^2)`.

Possibles preguntes d'exàmen relacionades amb el tema 8

1. Si tenim un conjunt amb 99 elements i fem un hash. Quants valors diferents haurà de tenir aquest hash. Raona la teva resposta

2. Quines propietats ha de complir una funció hash?

3. Com valoraràs si dues adreces són similars?