Ayudantía 5: Hash

Rafael Elberg
Pablo Soto

Contenidos

- Repaso de Hash
- Ejemplos de Hash

¿Qué es Hash?

Tabla de Hash

- **Asociar** un valor con una clave
- **Obtener** el valor asociado a una clave

Tabla de Hash

- **Asociar** un valor con una clave
- **Obtener** el valor asociado a una clave

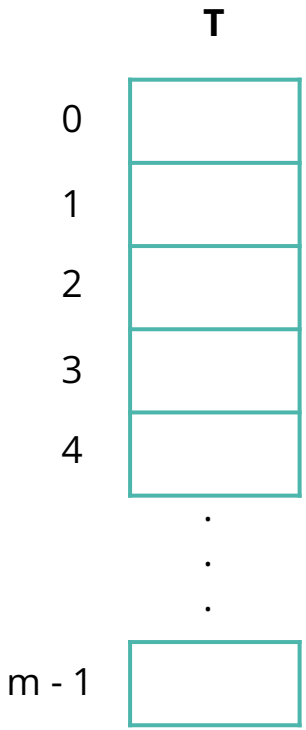


Tabla de Hash

T [k] = ∅

T [k] != ∅

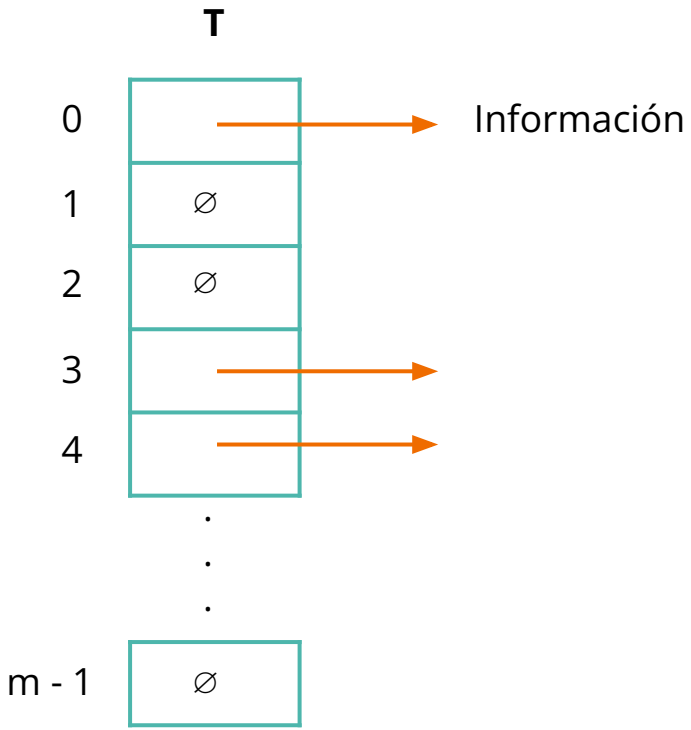


Tabla de Hash

T [k] = ∅ **no está**

T [k] != ∅

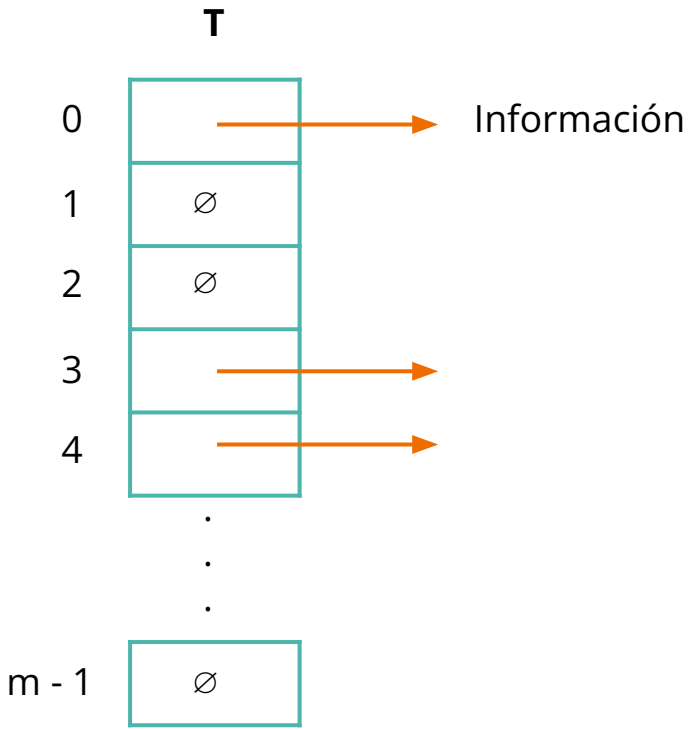
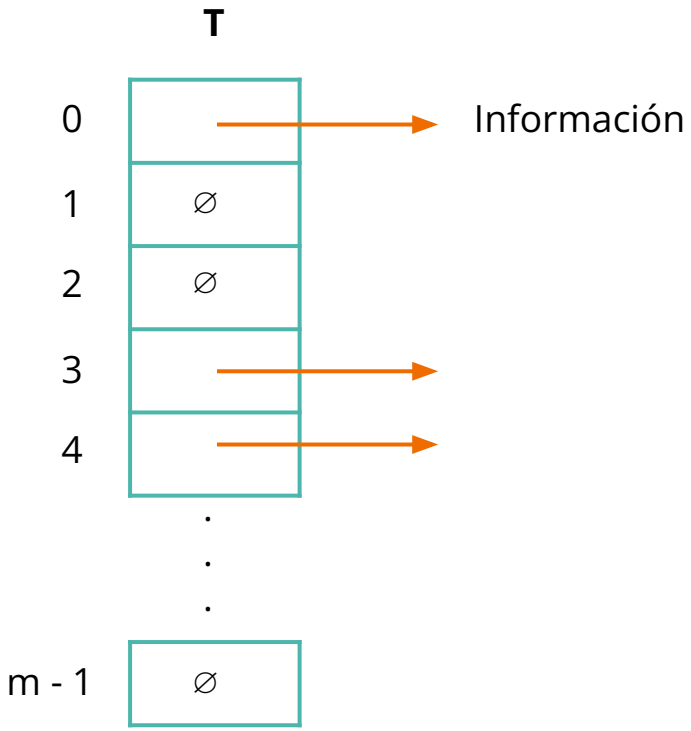


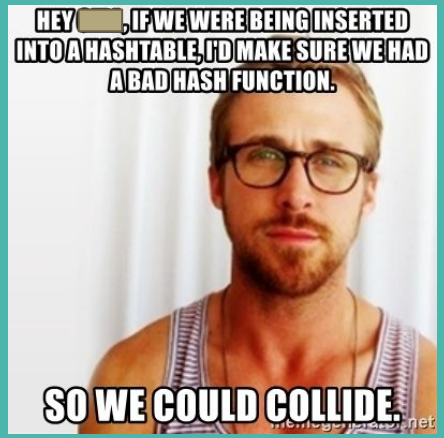
Tabla de Hash

T [k] = ∅ **no está**

T [k] != ∅ **está**

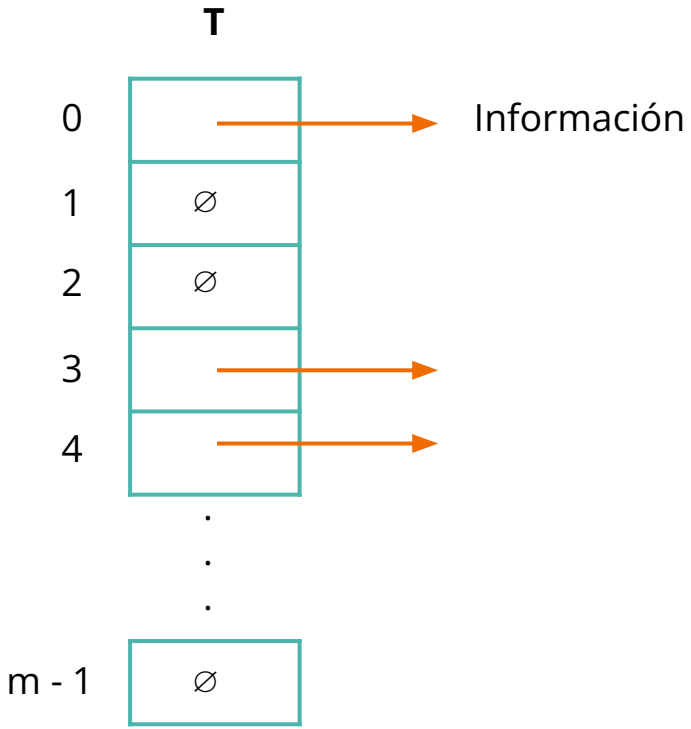


Función de Hash



Función de Hash

- **Calcular** índice a partir de la clave
- **Obtener** el valor asociado a una clave

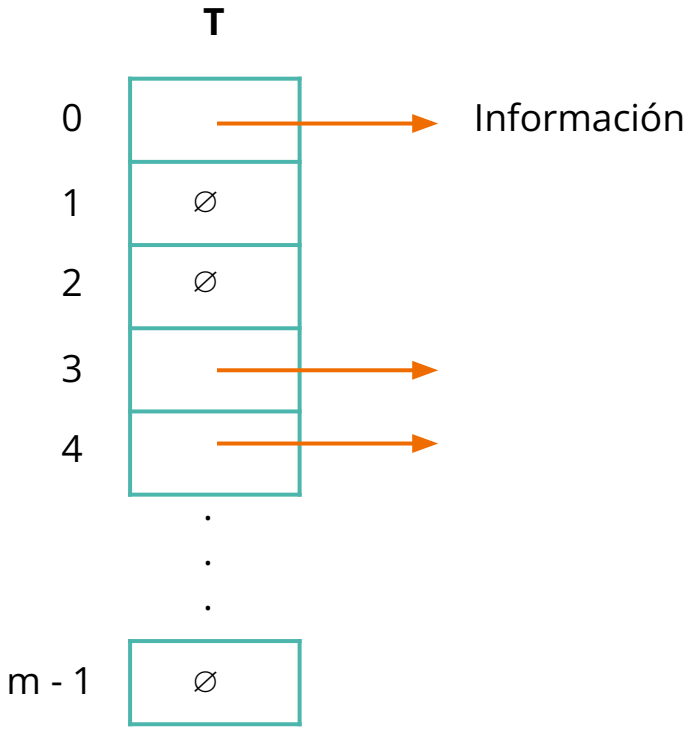


Función de Hash

- **Calcular** índice a partir de la clave
- **Obtener** el valor asociado a una clave

Definimos una función **h**:

h : **D** → {0, 1, ..., m-1}

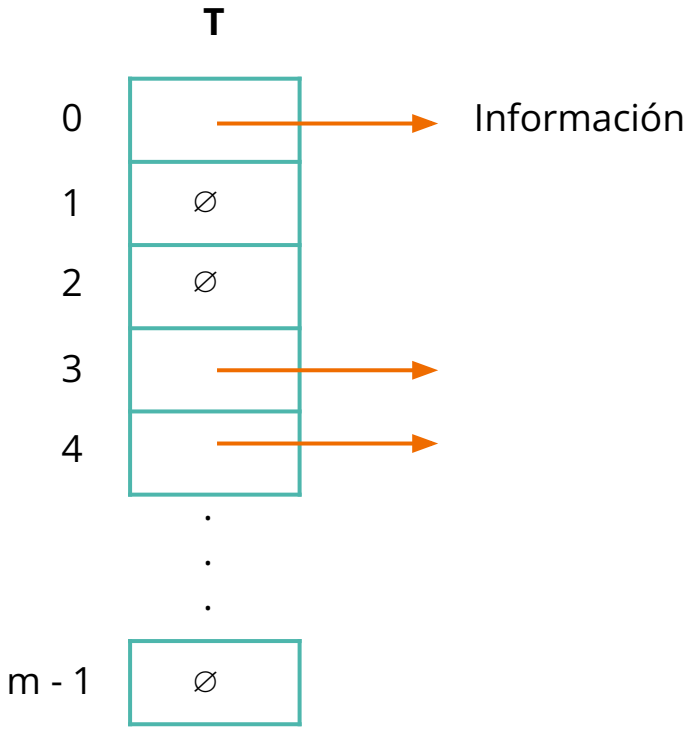


Función de Hash

- **Calcular** índice a partir de la clave
- **Obtener** el valor asociado a una clave

Definimos una función **h**:

h : **D** → {0, 1, ..., m-1}

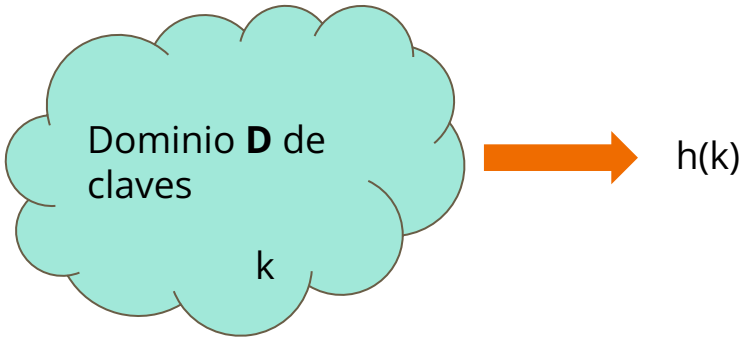


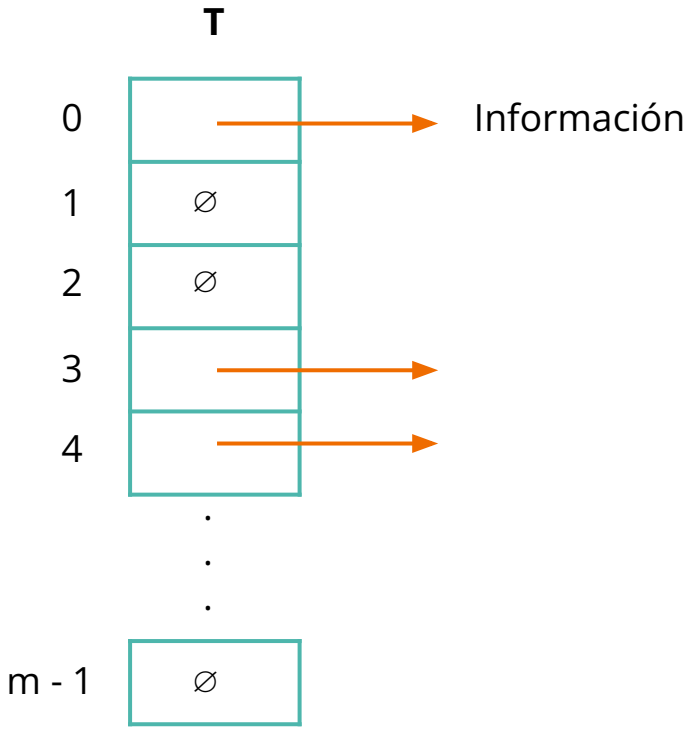
Función de Hash

- **Calcular** índice a partir de la clave
- **Obtener** el valor asociado a una clave

Definimos una función **h**:

h : **D** → {0, 1, ..., m-1}



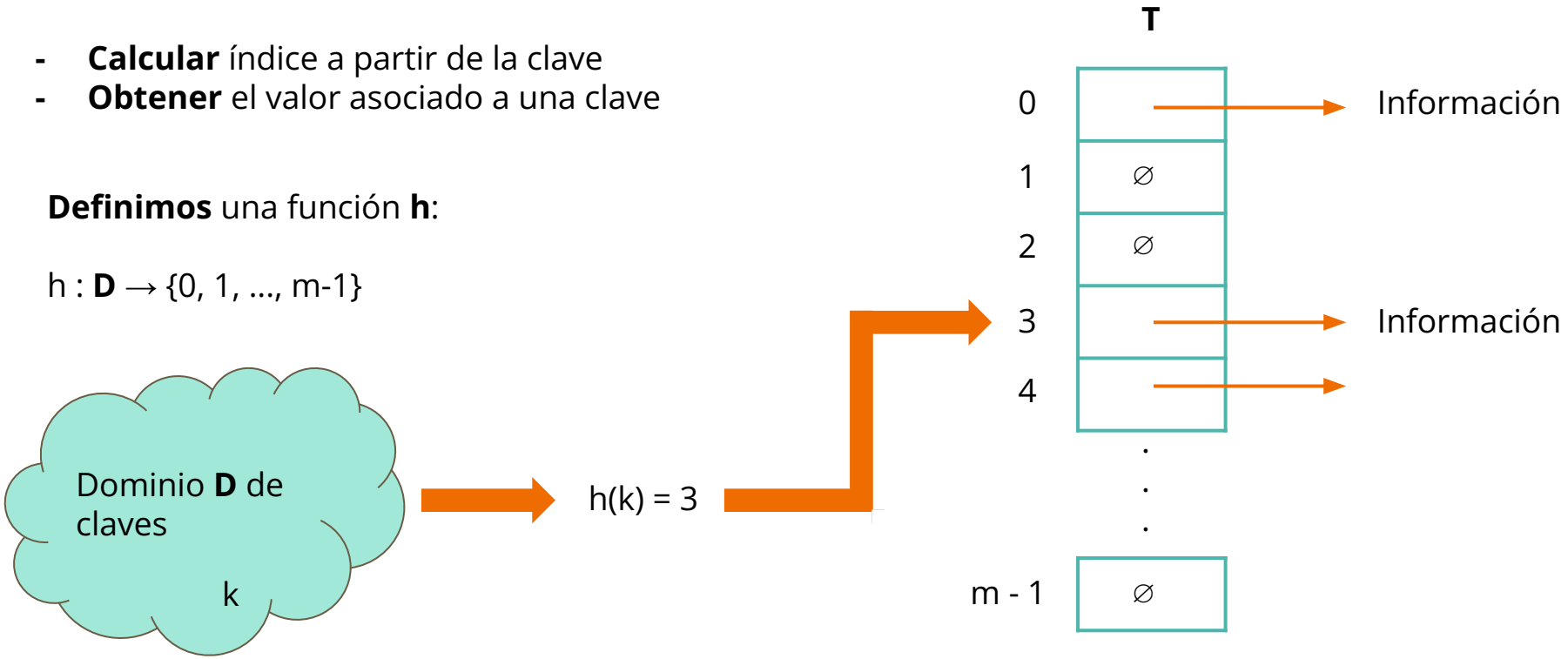


Función de Hash

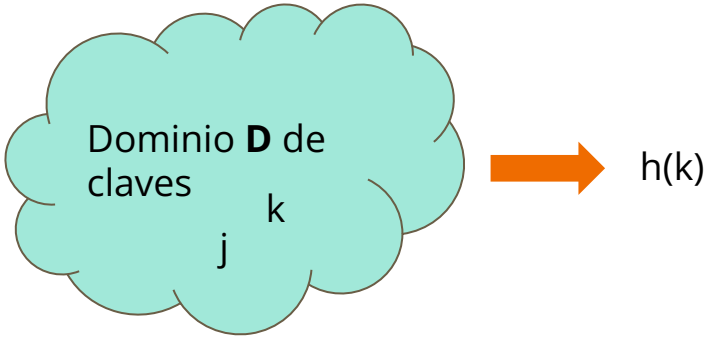
- **Calcular** índice a partir de la clave
- **Obtener** el valor asociado a una clave

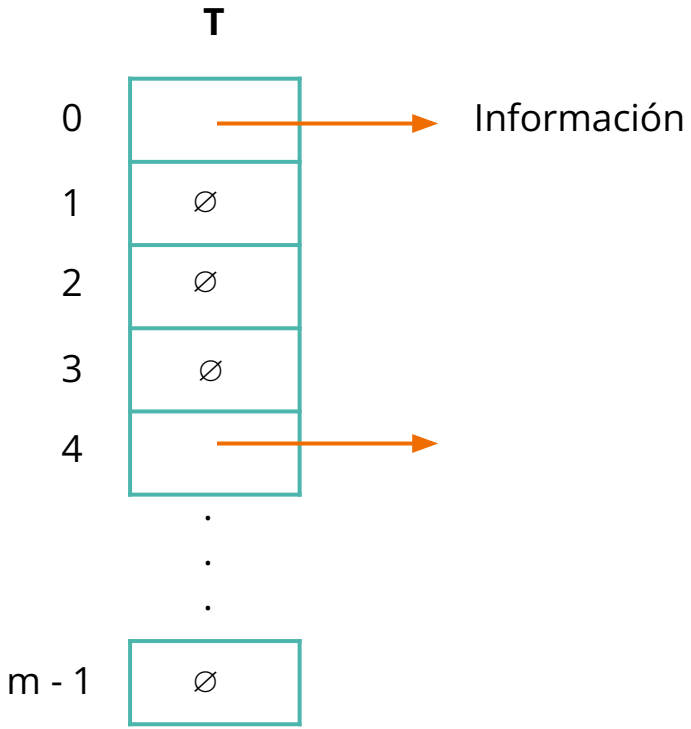
Definimos una función **h**:

h : **D** → {0, 1, ..., m-1}

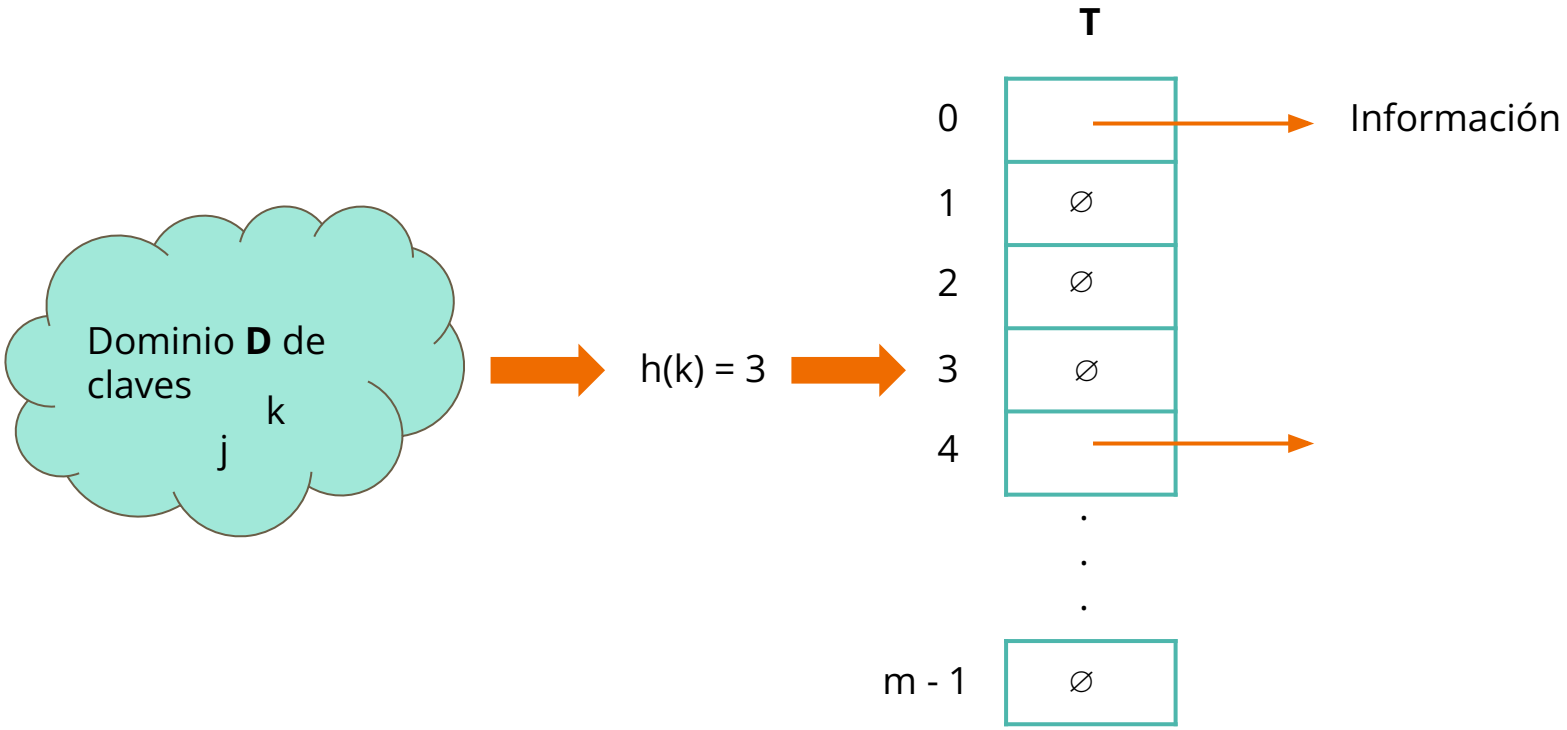


¿Colisiones?

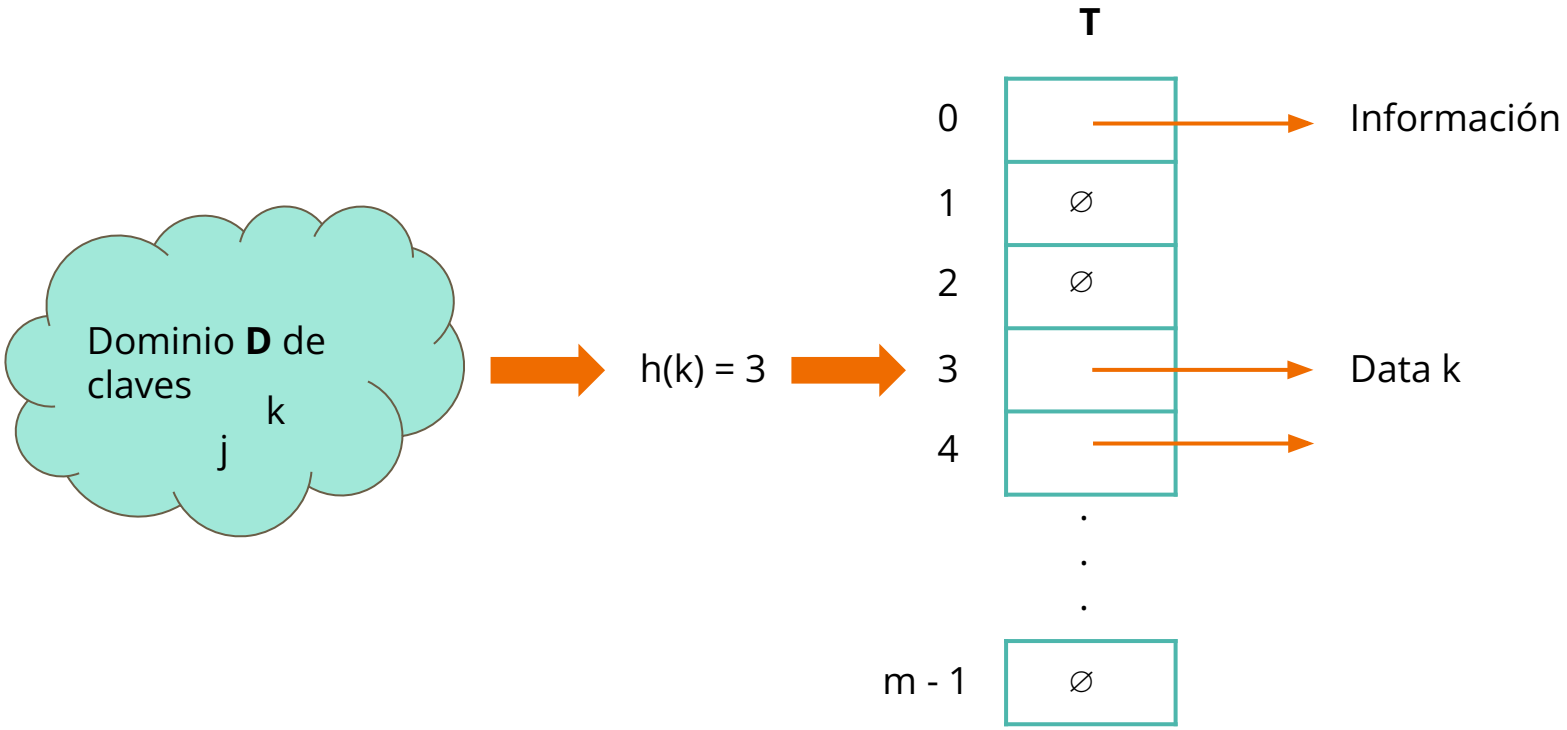




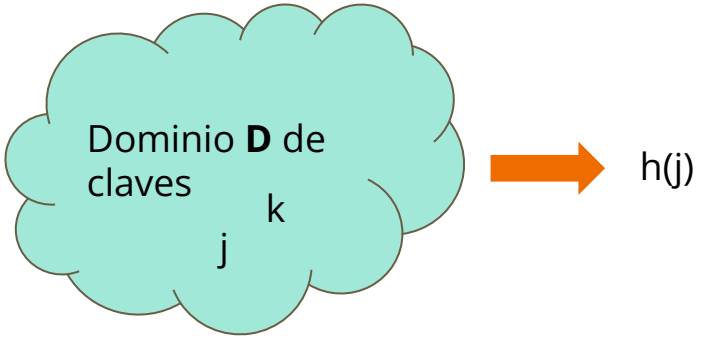
Insertemos la clave k

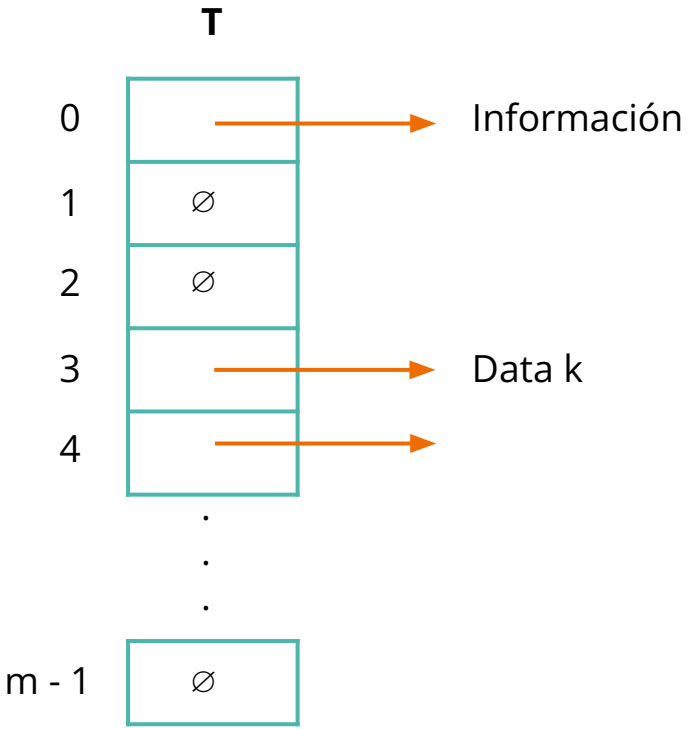


Insertamos información

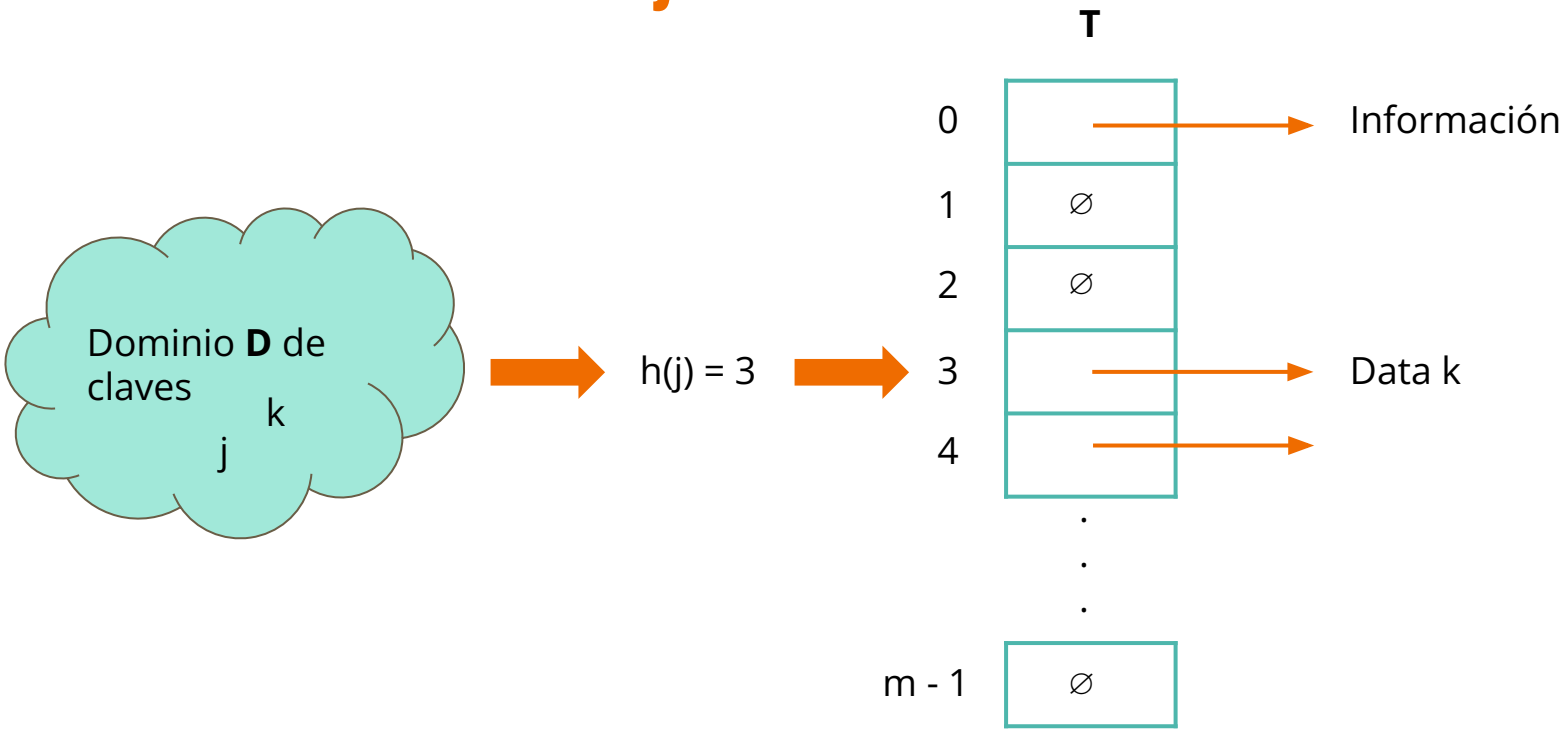


Insertemos la clave j

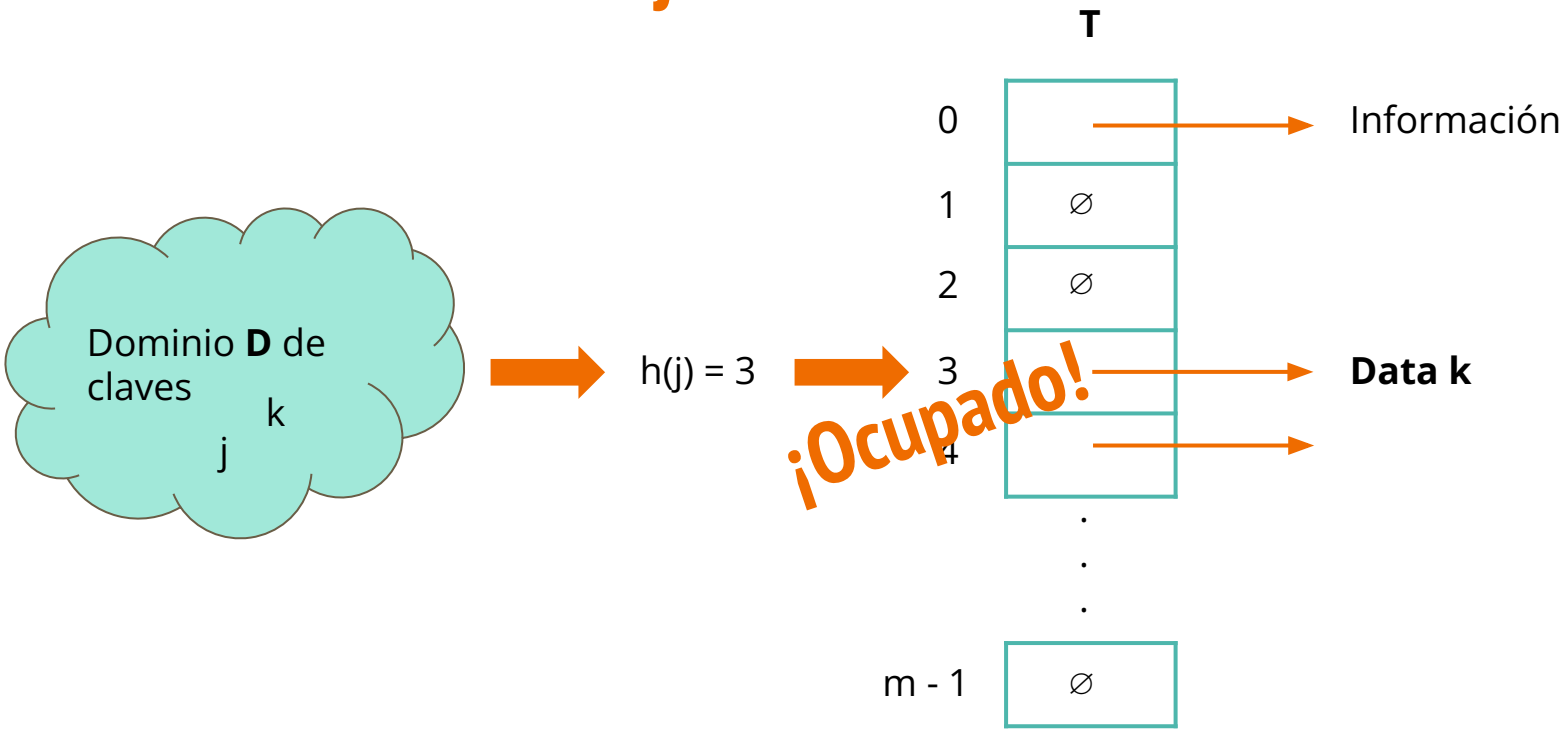




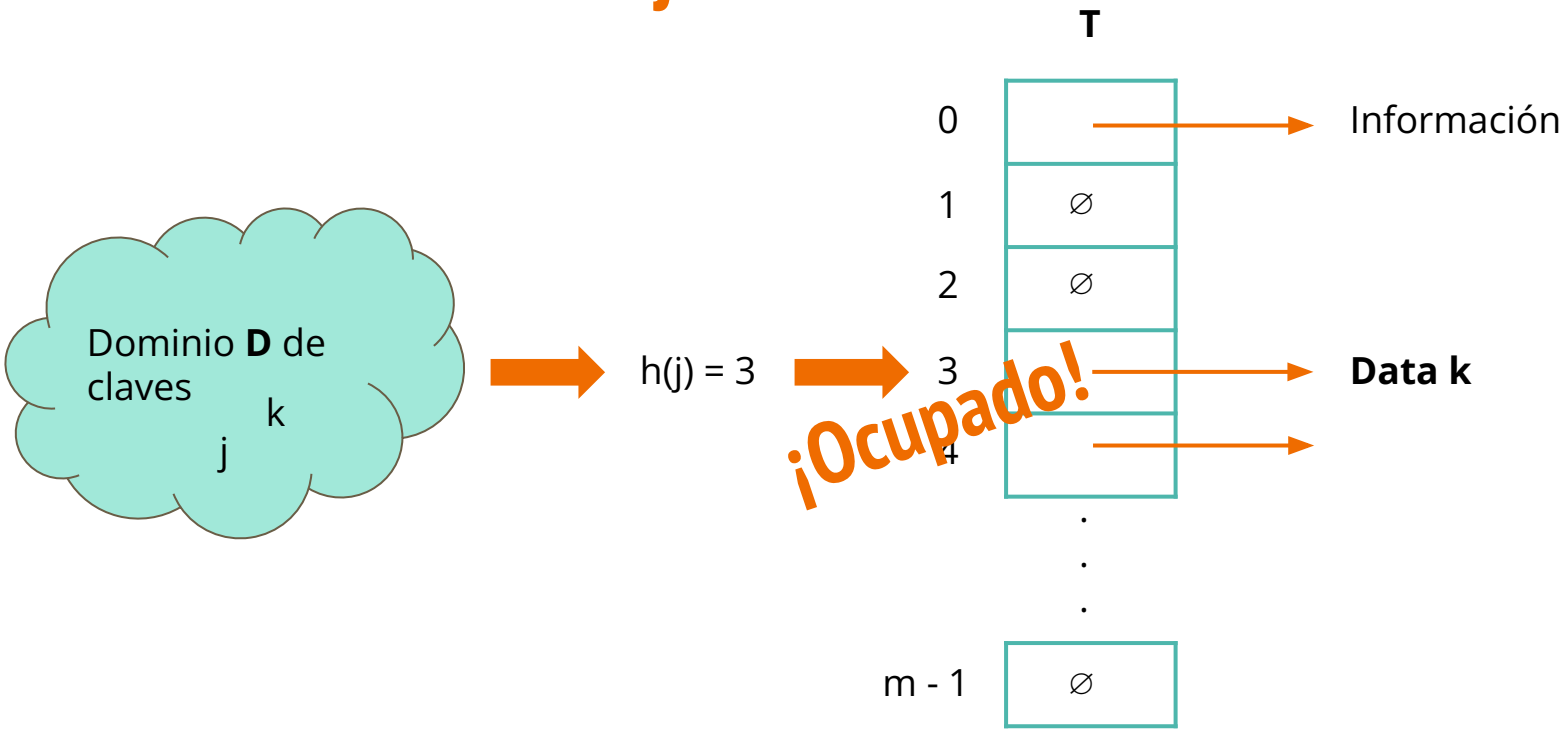
Insertemos la clave j



Insertemos la clave j

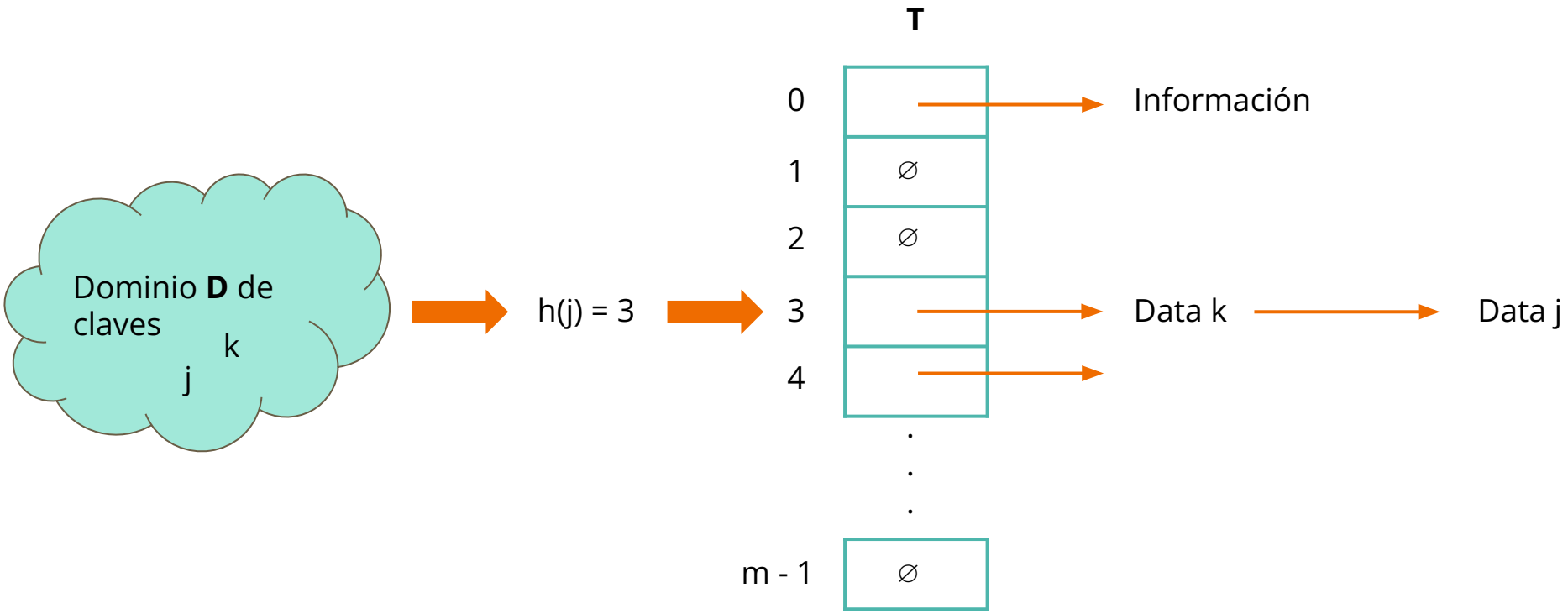


Insertemos la clave j



Solución: Encadenamiento

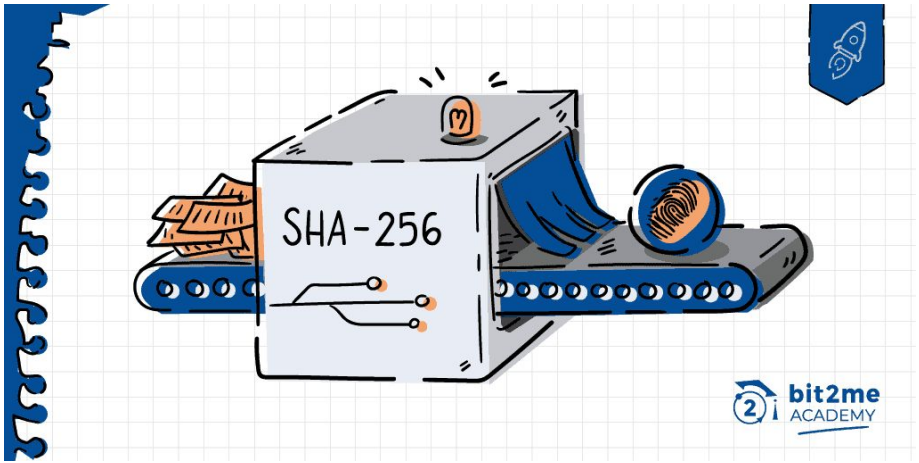
Encadenamiento



¿Por qué Hash?

Propiedades de Hashing

- Almacenar grandes cantidades de información (imágenes, encriptación, integridad de archivos)



Propiedades de Hashing

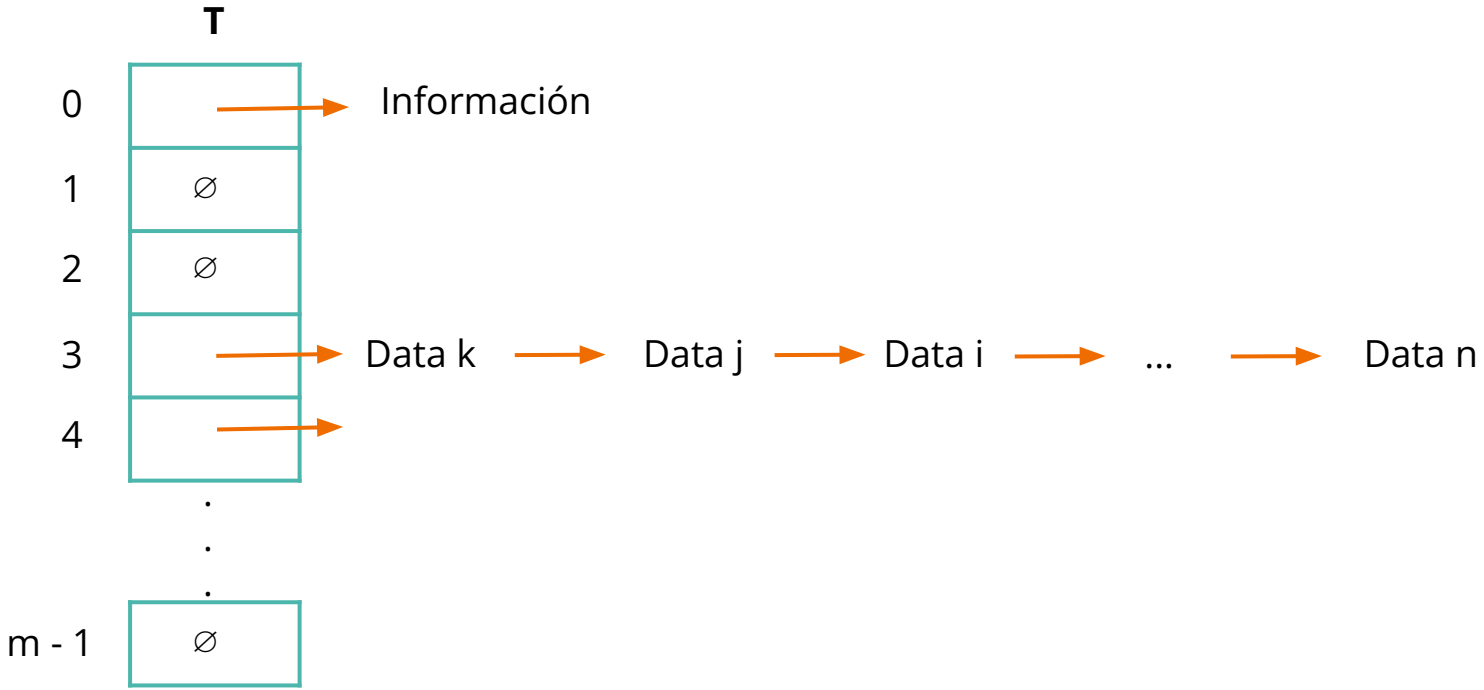
- Almacenar grandes cantidades de información (imágenes, encriptación, integridad de archivos)
- **Buscar** el dato con clave *k* en **O(1)** promedio

Tener en cuenta...

- Uniformidad (encadenamiento)

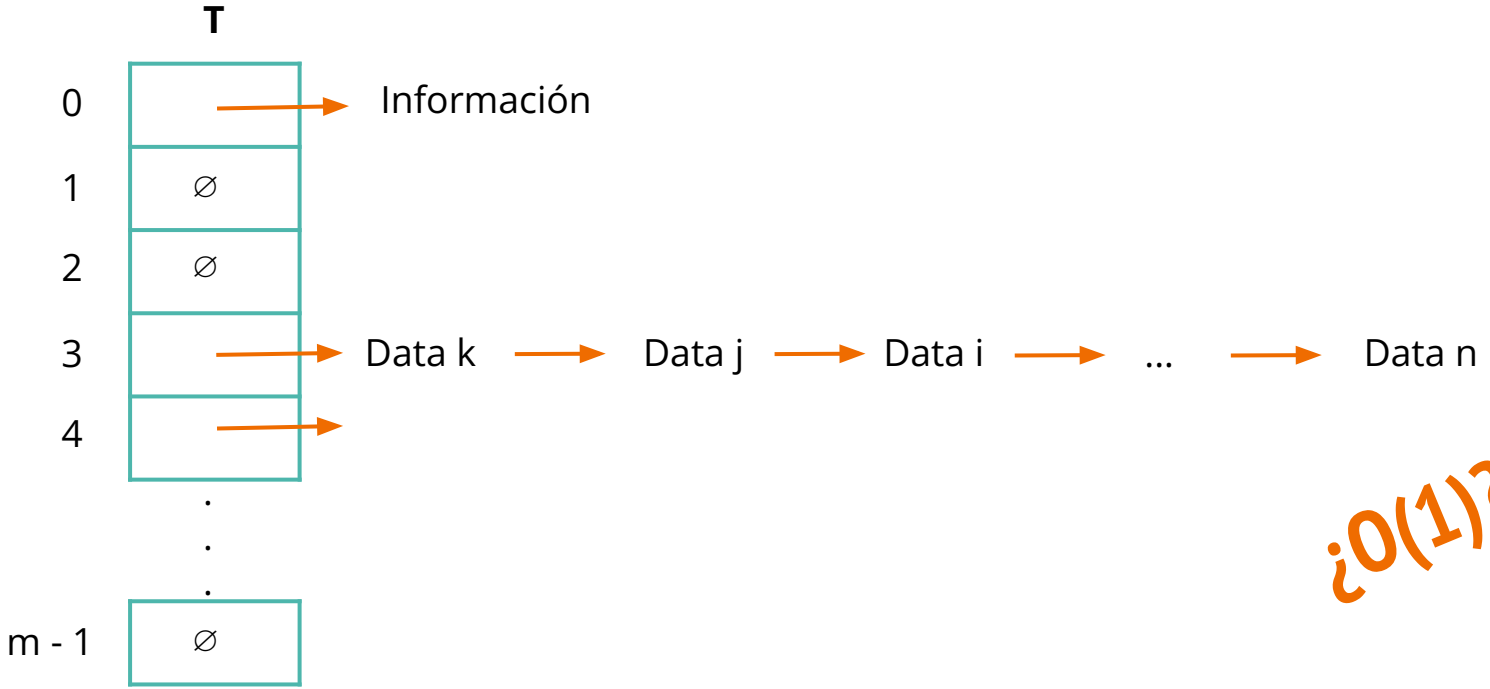
Tener en cuenta...

- Uniformidad (encadenamiento)



Tener en cuenta...

- **Uniformidad** (encadenamiento)



¿O(1)?

Tener en cuenta...

- Uniformidad (encadenamiento)

Tener en cuenta...

- Uniformidad (encadenamiento)
- Tamaño de la tabla

Tener en cuenta...

- Uniformidad (encadenamiento)
- Tamaño

¡*Trade off* tamaño-eficiencia!

Trade off tamaño-eficiencia

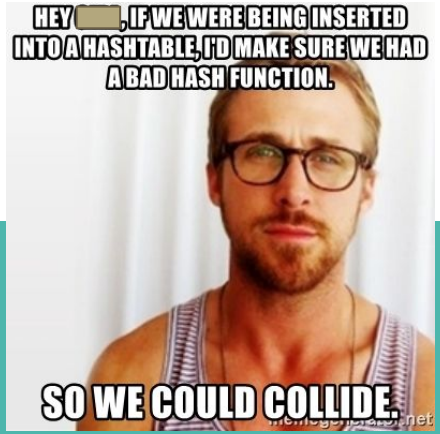
- Uniformidad (encadenamiento)
- Tamaño de la tabla

Trade off tamaño-eficiencia

- Uniformidad (encadenamiento)
- Tamaño de la tabla
- Colisiones

Trade off tamaño-eficiencia

- Uniformidad (encadenamiento)
- Tamaño de la tabla
- Colisiones
- Rapidez de cálculo de la función


HEY [redacted], IF WE WERE BEING INSERTED INTO A HASHTABLE, I'D MAKE SURE WE HAD A BAD HASH FUNCTION.
SO WE COULD COLLIDE.


HAVING NO COLLISIONS IN YOUR HASH TABLE
makeameme.org

Ejemplos

Ej 1 Rolling Hash

Considera una una secuencia de datos, donde queremos hashear cada subsecuencia. Por ejemplo, queremos poder buscar todas las ocurrencias de un substring de tamaño 3 dentro de un string.

holaajwoisnalcmskaaancsjalskdnwaaandwbvdkancaaa

BuscarString(string, substring)

Ej 1 ¿Qué necesitamos?

- ¿Qué característica de nuestra función de hash sería importante?
- ¿Qué hacemos en caso de colisiones?
- ¿Cuál será el tamaño de nuestra tabla?

Ej 1 ¿Qué necesitamos?

- ¿Qué característica de nuestra función de hash sería importante?
    - ¡Que se pueda calcular incrementalmente!
- ¿Qué hacemos en caso de colisiones?
    - ¡Nada!
- ¿Cuál será el tamaño de nuestra tabla?
    - ¡Tablando puras leseras!

Ej 1 Hash incremental sin tabla!

- Para buscar todas las ocurrencias de un único substring en un string arbitrario, no nos sirve guardar en tabla.
- Queremos poder calcular el hash de el substring siguiente utilizando el anterior.

Ej 1 Algoritmo Rabin-Karp

RKFind(t[n] ->string total, p[m] -> substring a buscar, d -> tamaño del alfabeto)

- r = d^(m-1)
- p_hash = hash(p, 0, m, d)
- curr_hash = hash(t, 0, m, d)
- for s = 0 to n - m:
- if curr_hash == p_hash
- if p[1.....m] = t[s + 1..... s + m]
- print "pattern found at position" s
- If s < n-m
- curr_hash = rehash(curr_hash,t[s],t[s+m+1], d, r)

Ej 1 Algoritmo Rabin-Karp

hash(s[] -> string, i ->índice inicial, f -> índice final, d -> tamaño alfabeto)

- h = 0
- for j = i to f
- h += (d*h + ord(s[j])) % q
- return h%q

O

- h = 0
- for j = i to f
- h += (ord(s[j])*d^(f-j)) % q
- return h%q

Paréntesis

$$(a + b) \% c = ((a \% c) + (b \% c)) \% c$$

Ej 1 Algoritmo Rabin-Karp

rehash(current_h, out_c -> caracter que sale, in_c -> caracter que entra, d, r)

- h = ((d * (current_h - out_c * r) + in_c) mod q
- return h

Ejercicio

RKFind(abbcabc, abc, 3)