

# Más diccionarios



Ya vimos como implementar diccionarios usando ABB.

¿Que pasa si **no nos interesa** el orden de los datos?

Esto nos debería permitir complejidades menores que  $O(\log n)$

Idealmente serían  $O(1)$ . ¿Qué estructura nos permite esto?

# Dónde guardar una clave



¿Podremos guardar los datos en un arreglo?

¿En qué posición?

Debería depender solo de la clave. ¿Cómo hacemos esto?

# Función de Hash

Una función de hash se define como sigue:

$$h : D \rightarrow \mathbb{N}_0$$

Donde  $D$  es el dominio de las claves.

Podemos usar esto para definir en que celda guardar una clave  $key \in D$

# Tabla de Hash

Como diccionario, guarda pares de la forma  $(key, value)$

donde  $key \in D$

Se implementa como un arreglo  $T$  de tamaño  $m$

El par  $(k, v)$  se guarda en  $T[h(k)]$ . ¿Qué pasa si  $h(k) > m$ ?

# Método de la división

Simplemente, usar el módulo:

$$h'(k) = h(k) \bmod m$$

Por lo tanto,  $(k, v)$  se guarda en  $T[h'(k)]$

Se recomienda que  $m$  sea un número primo

# Método de la multiplicación

Sea  $A$  un número entre 0 y 1:

$$h'(X) = \lfloor m \cdot (A \cdot h(X) \bmod 1) \rfloor$$

Por lo tanto,  $(k, v)$  se guarda en  $T[h'(k)]$

Se recomienda  $A = \frac{1}{\varphi}$

# Colisiones

Si  $A \neq B$ ,

Decimos que una función de hash produce una **colisión** si

$$h(A) = h(B)$$

Además, diremos que existe una **colisión en la tabla** si

$$h'(A) = h'(B)$$

Una colisión en la tabla **no implica** una colisión en la función

# Manejo de colisiones



En ambos casos,  $A$  y  $B$  caen en la misma celda de la tabla

¿Cómo podemos guardar ambos datos en la tabla?

Nos interesa poder buscarlos en el futuro



# Direccionamiento abierto



Podemos buscar otra celda que esté disponible

¿Dónde la buscamos?

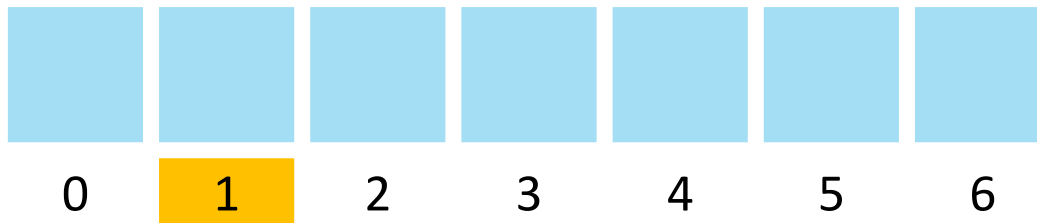
Debemos seguir alguna regla

# Una posible regla: Sondeo lineal

$$m = 7$$

Insertemos la  $A$

$$h(A) = 15; 15 \bmod 7 = 1$$

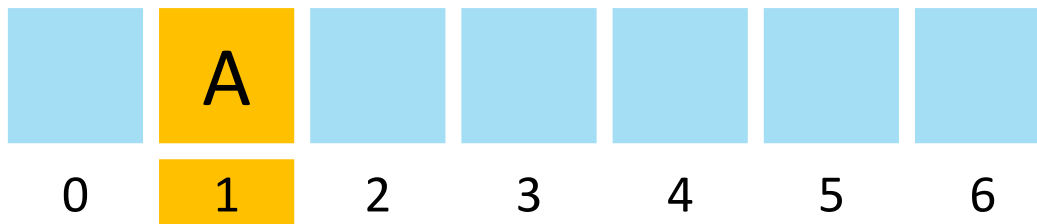


...

$$m = 7$$

Insertemos la  $A$

$$h(A) = 15; 15 \bmod 7 = 1$$

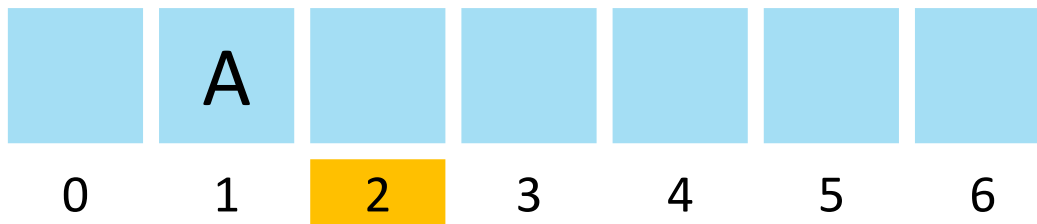


...

$$m = 7$$

Insertemos la  $Q$

$$h(Q) = 37; 37 \bmod 7 = 2$$



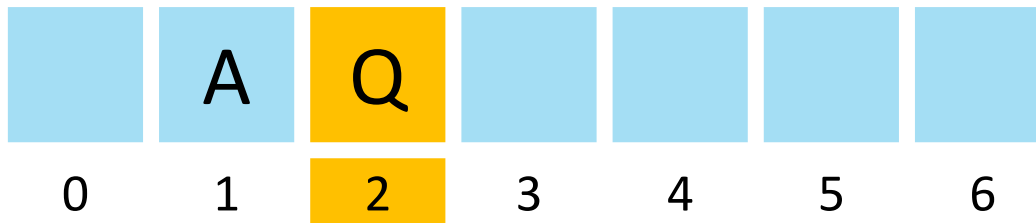
...

$$m = 7$$

Insertemos la  $Q$

$$h(Q) = 37$$

$$37 \bmod 7 = 2$$

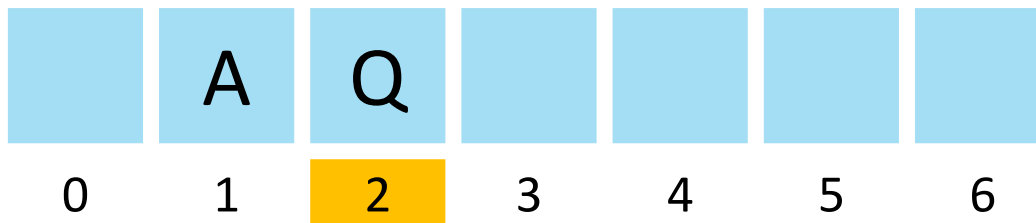


...

$$m = 7$$

Insertemos la  $L$

$$h(L) = 51; 51 \bmod 7 = 2$$

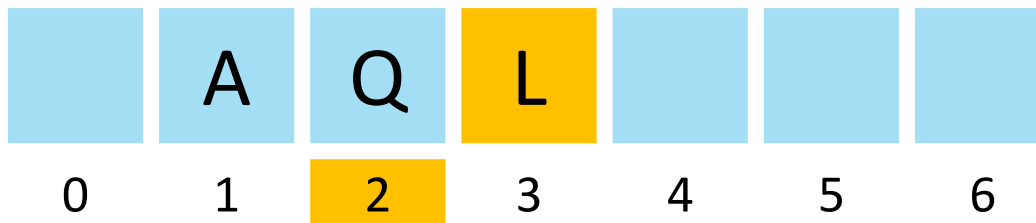


...

$$m = 7$$

Insertemos la  $L$

$$h(L) = 51; 51 \bmod 7 = 2$$

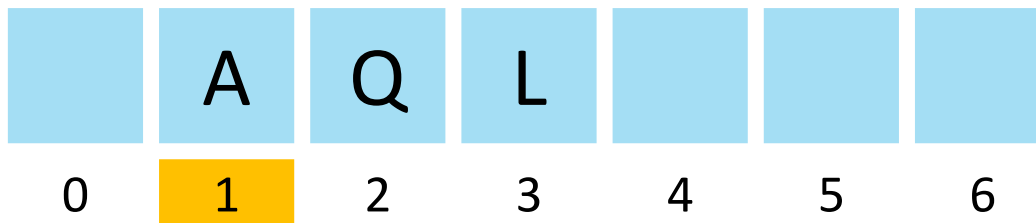


...

$$m = 7$$

Insertemos la  $X$

$$h(X) = 29; 29 \bmod 7 = 1$$



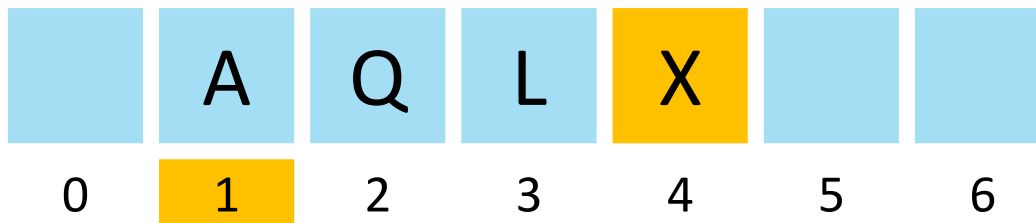


# Fin del ejemplo

$$m = 7$$

Insertemos la X

$$h(X) = 29; 29 \bmod 7 = 1$$



# Otras posibles reglas

Métodos populares de direccionamiento abierto son:

- Sondeo lineal:
  - buscar en  $H, H + 1, H + 2, H + 3 \dots$
- Sondeo cuadrático:
  - buscar en  $H, H + 1c_1 + 1^2c_2, H + 2c_1 + 2^2c_2 \dots$
- *Double hashing*:
  - buscar en  $h_1(k), h_1(k) + h_2(k), h_1(k) + 2h_2(k), \dots$

# Búsqueda exitosa bajo sondeo lineal

$$m = 7$$

	A	Q	L	X		
0	1	2	3	4	5	6

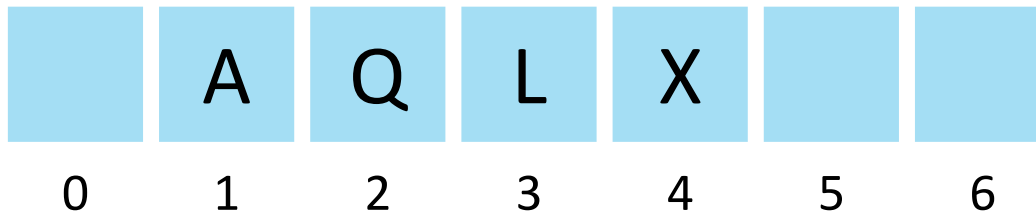
¿Cómo buscamos la  $X$  con sondeo lineal?

$$h(X) = 29; 29 \bmod 7 = 1$$

	A	Q	L	X		
0	1	2	3	4	5	6

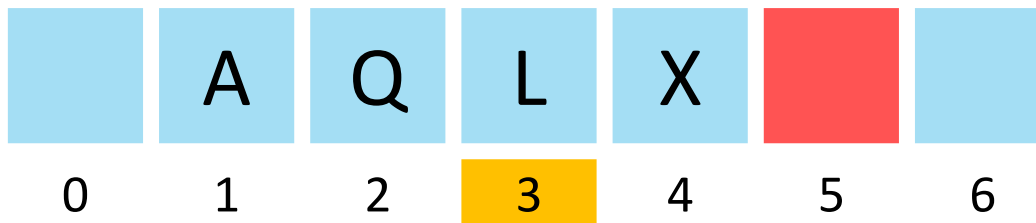
# Búsqueda de un dato que no está

$$m = 7$$



¿Cómo buscamos la  $R$  con sondeo lineal?

$$h(R) = 10; 10 \bmod 7 = 3$$



# Problemas del direccionamiento abierto

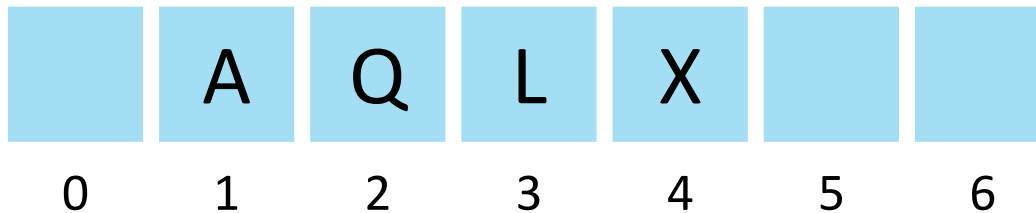


Este esquema no permite eliminar datos de la tabla

¿Por qué?

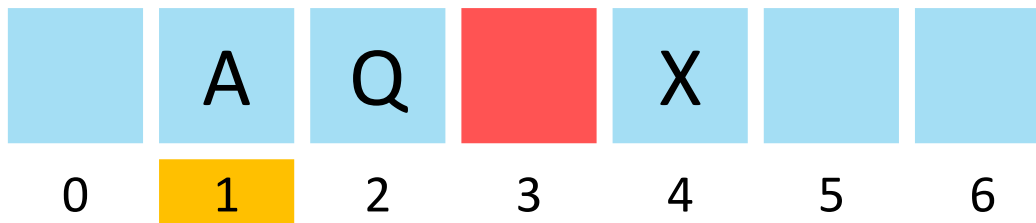
# La eliminación es problemática

$$m = 7$$



Eliminemos la *L*. ¿Cómo buscamos la *X* con sondeo lineal?

$$h(X) = 29; 29 \bmod 7 = 1$$



# Otra posibilidad



Si  $A$  ya estaba en la tabla:

Podemos guardar  $A$  y  $B$  en la misma celda...

... pero ¡dentro de otra estructura de datos!

# Encadenamiento



Por ejemplo, si tenemos una **lista** en cada celda de la tabla

Hemos guardado  $n$  datos, y la tabla es de tamaño  $m$

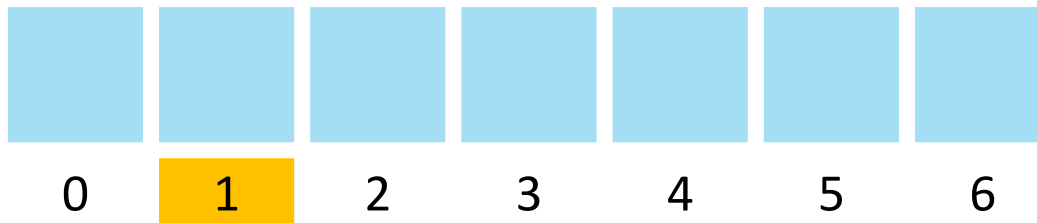


# Encadenamiento: ejemplo

$$m = 7$$

Insertemos la  $A$

$$h(A) = 15; 15 \bmod 7 = 1$$

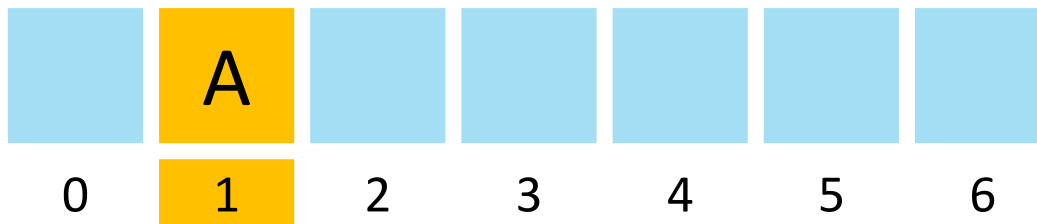


# Sigue el ejemplo

$$m = 7$$

Insertemos la  $A$

$$h(A) = 15; 15 \bmod 7 = 1$$

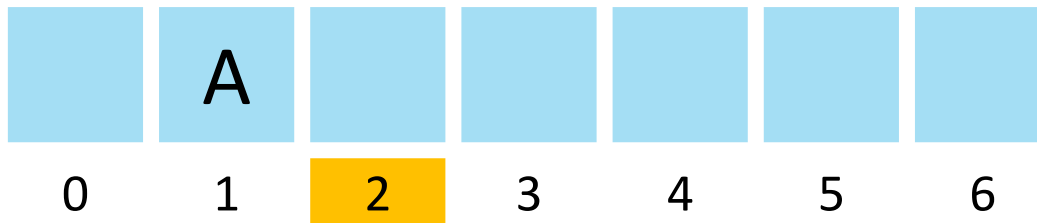


...

$$m = 7$$

Insertemos la Q

$$h(Q) = 37; 37 \bmod 7 = 2$$

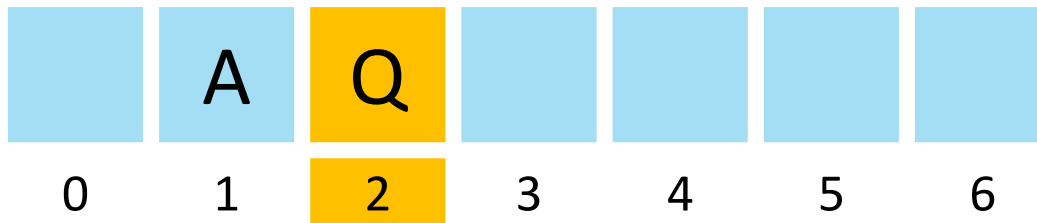


...

$$m = 7$$

Insertemos la  $Q$

$$h(Q) = 37; 37 \bmod 7 = 2$$

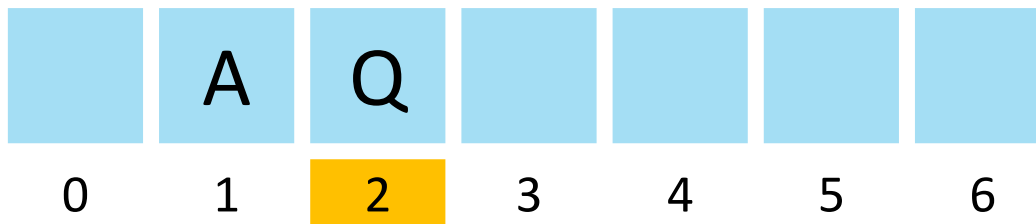


...

$$m = 7$$

Insertemos la  $L$

$$h(L) = 51; 51 \bmod 7 = 2$$

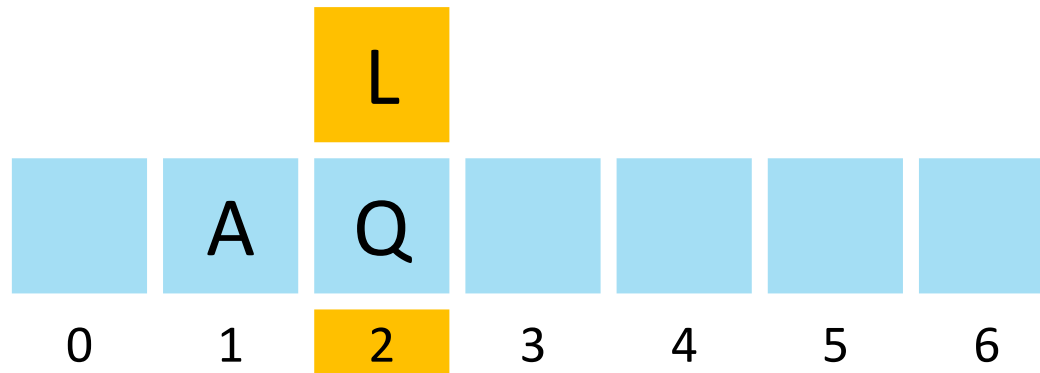


...

$$m = 7$$

Insertemos la  $L$

$$h(L) = 51; 51 \bmod 7 = 2$$

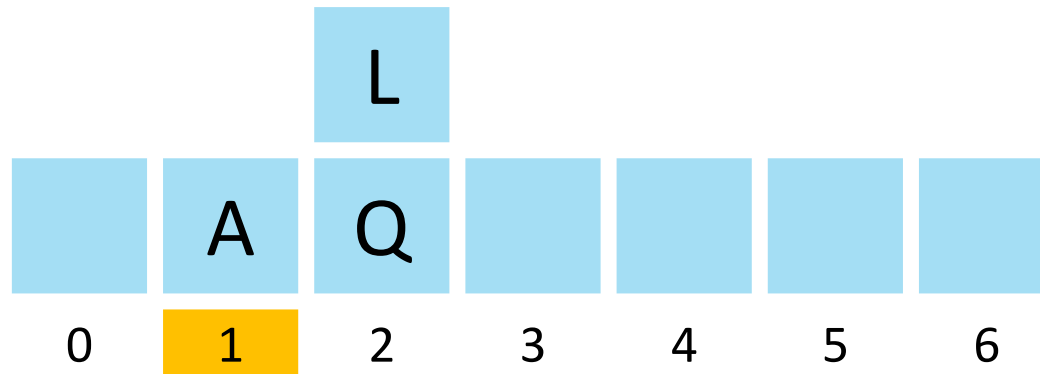


...

$$m = 7$$

Insertemos la  $X$

$$h(X) = 29; 29 \bmod 7 = 1$$

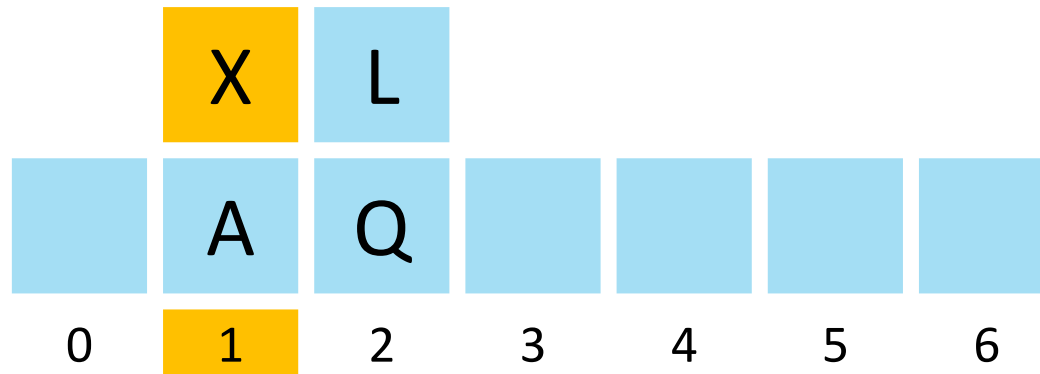


...

$$m = 7$$

Insertemos la  $X$

$$h(X) = 29; 29 \bmod 7 = 1$$



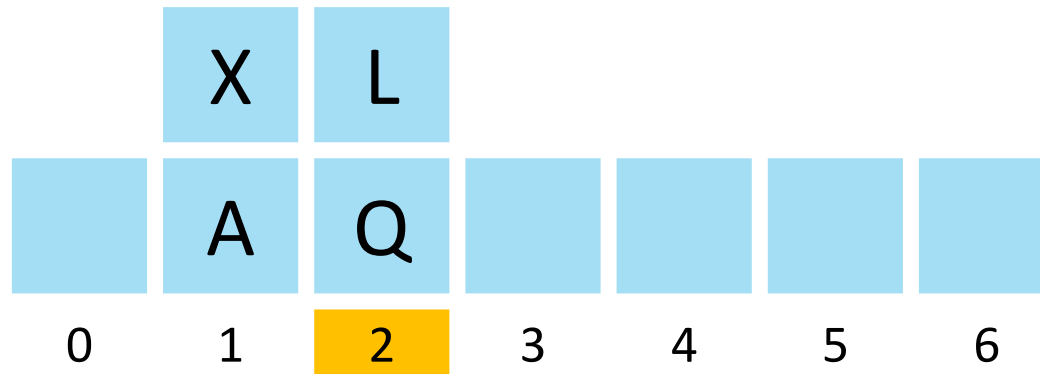


...

$$m = 7$$

Insertemos la  $F$

$$h(F) = 58; 58 \bmod 7 = 2$$

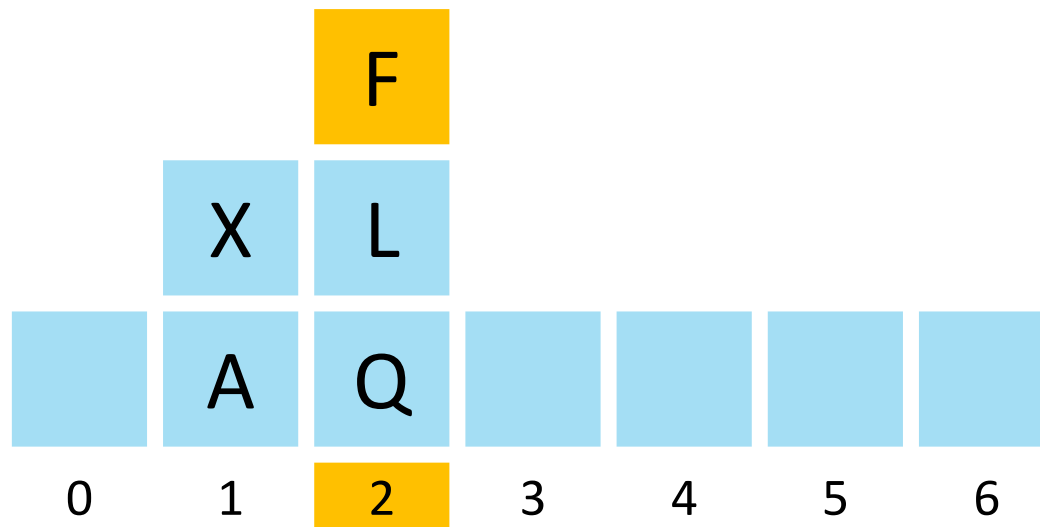


# Fin del ejemplo

$$m = 7$$

Insertemos la  $F$

$$h(F) = 58; 58 \bmod 7 = 2$$



# Factor de carga



Sea  $n$  la cantidad de claves almacenadas en la tabla

Se define el factor de carga  $\lambda$  como:

$$\lambda = \frac{n}{m}$$

Podemos fijar un valor  $\lambda_{max}$ , y garantizar que

$$\lambda < \lambda_{max}$$

# Rehashing



Si  $\lambda < \lambda_{max}$ , en algún momento hay que hacer crecer la tabla

A este proceso se le dice **rehashing**

¿Cuál es su complejidad?

# En resumen

Para implementar una tabla debemos entonces:

- Identificar el dominio de las claves
- Diseñar una función de hash para ese dominio
- Definir que hacer con los valores que se salen de la tabla
- Definir un esquema de resolución de colisiones
- Elegir un  $\lambda_{max}$  para garantizar eficiencia