Hashing

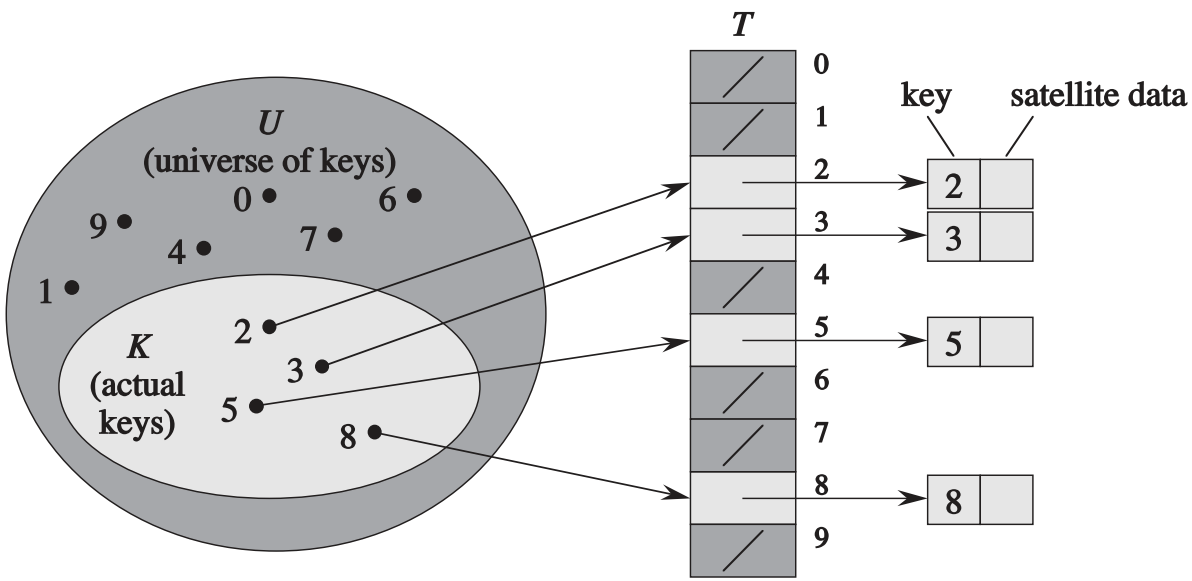
Algoritmos y Estructuras de Datos II

1^{er} cuatrimestre de 2018

Motivación

Queremos implementar un diccionario con sus operaciones de búsqueda, inserción y remoción muy eficientes.

• **Primera idea:** para un diccionario `dicc(U,V)` tenemos un arreglo A de largo $|U|$ (tamaño del universo U) donde A_i guarda `NULL` o el significado del elemento $i \in U$ [1].



[1] Si no son números, podemos usar una *codificación* $U \leftrightarrow \{0, \ldots, |U| - 1\}$.

Motivación

Ventajas de nuestra idea:

- Consultar es O(1).
- Insertar es O(1).
- Borrar es O(1).
- Sencilla de implementar.

Problemas de nuestra idea:

- Inicializar es O($|U|$).
- Necesitamos O($|U|$) memoria, incluso para 'guardar' un solo elemento.

El universo U podría ser enorme dependiendo del contexto. Por ejemplo en un `dicc(int,int)`, o `dicc(string,int)`.

Motivación

Ventajas de nuestra idea:

- Consultar es O(1).
- Insertar es O(1).
- Borrar es O(1).
- Sencilla de implementar.

Problemas de nuestra idea:

- Inicializar es O($|U|$).
- Necesitamos O($|U|$) memoria, incluso para 'guardar' un solo elemento.

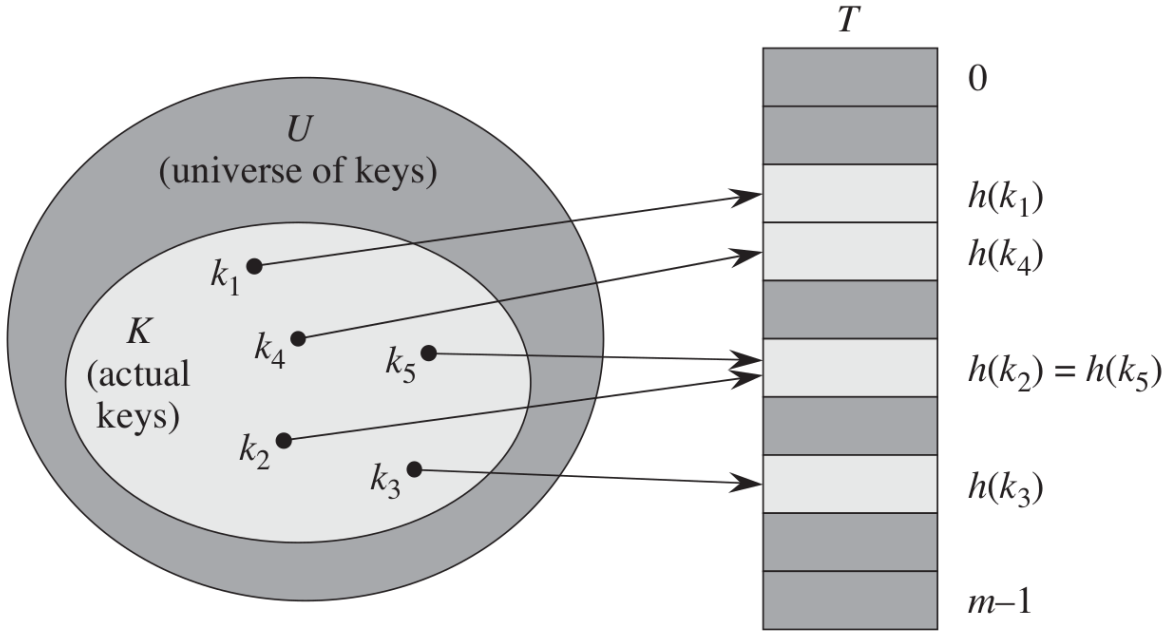
El universo U podría ser enorme dependiendo del contexto. Por ejemplo en un `dicc(int,int)`, o `dicc(string,int)`.

- **Segunda idea:** tabla de hash.

Tablas de hash

Una tabla de hash está basada en los siguientes elementos:

- Un arreglo T de tamaño m para guardar los datos. Usualmente m es proporcional al número de claves almacenadas.
- Una función de hash $h : U \to \{0, \ldots, m-1\}$
- Un procedimiento para tratar los casos de colisiones.



Colisiones

En una función de hash $h : A \to B$, una colisión se da cuando dos elementos $x, y \in A$ tienen igual imagen en la función de hash: $h(x) = h(y)$.

En nuestro diccionario sobre tabla de hash, ¿pueden ocurrir colisiones en $h : U \to \{0, \ldots, m-1\}$?

- Si nuestra tabla tiene el tamaño del universo de las claves y nuestra función de hash es 'buena', podríamos no tener colisiones (*perfect hashing*).
- Pero esa era nuestra primera idea. En general la tabla no es tan grande como $|U|$.

Colisiones

Con nuestra función $h : U \to \{0, \ldots, |T| - 1\}$ podríamos guardar $|T|$ elementos sin colisiones. Si tenemos que guardar más, agrandamos la tabla y cambiamos nuestra función h.

Entonces, ¿es realmente necesario lidiar con colisiones?

Colisiones

Con nuestra función $h : U \to \{0, \ldots, |T| - 1\}$ podríamos guardar $|T|$ elementos sin colisiones. Si tenemos que guardar más, agrandamos la tabla y cambiamos nuestra función h.

Entonces, ¿es realmente necesario lidiar con colisiones?

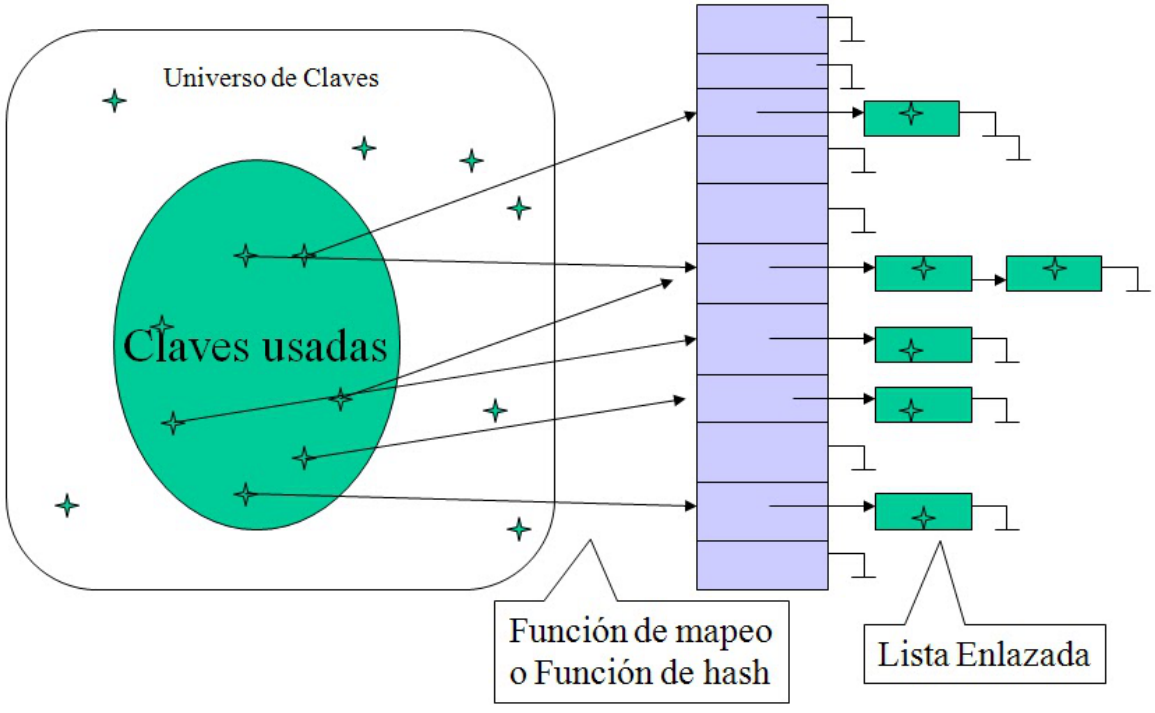
En general sí, es necesario:

- No sabemos por adelantado qué m claves se van a insertar. Peor aún, normalmente ni conocemos la distribución.

- Dos soluciones usuales para lidiar con colisiones:
  - Hashing abierto.
  - Hashing cerrado.

Hashing abierto (*chaining*)

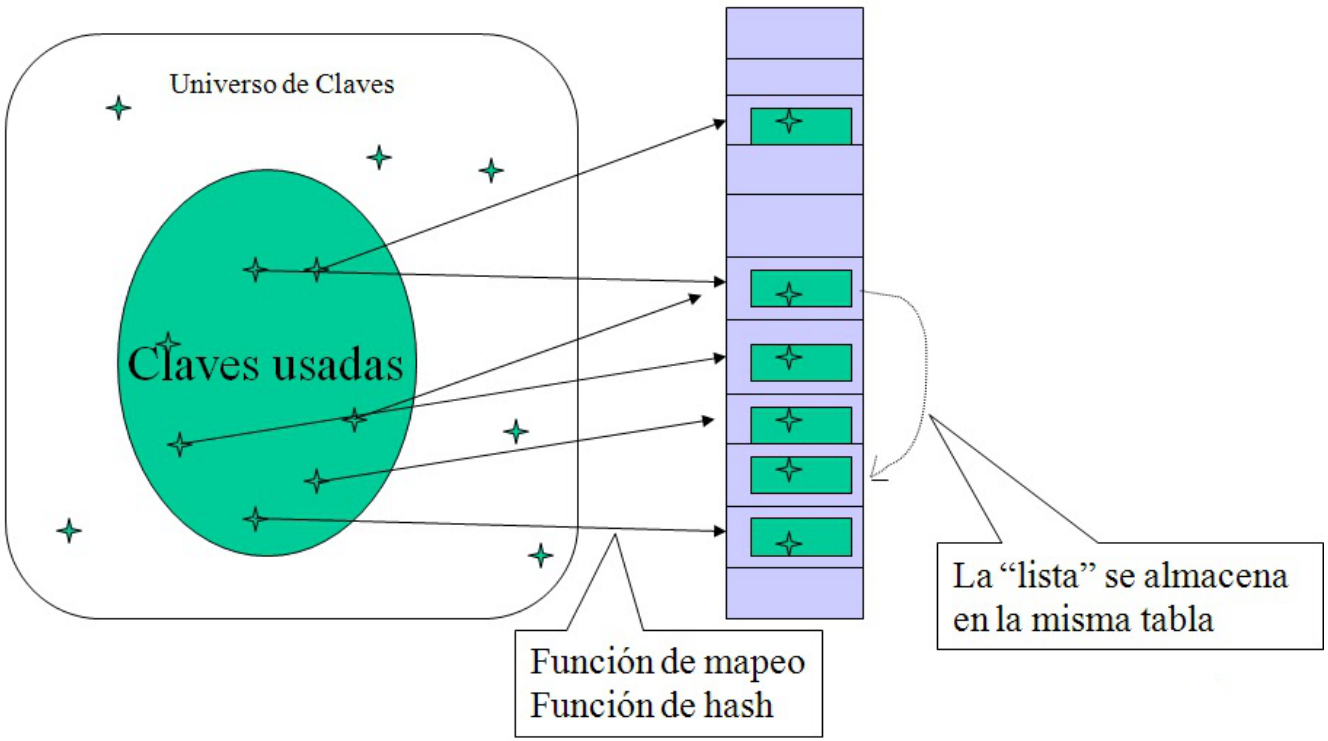
En cada posición de nuestra tabla tenemos una lista enlazada. Los elementos que tienen igual imagen en h se guardan en la misma lista.



- ¿Complejidades de las operaciones?

Hashing cerrado (*probing*)

Dada una clave $k \in U$, no trabajamos con un único hash $h(k)$, sino una secuencia $[h_1(k), ..., h_m(k)]$[2], que vamos probando hasta encontrar un casillero libre.



- ¿Complejidades de las operaciones?

[2] *Probe sequence.*

Hashing cerrado (*probing*)

Para obtener una secuencia de probing hay 3 técnicas usuales:

- *Linear probing*:

$$h(k,i) = (h(k) + c.i) \mod m$$

- *Quadratic probing*:

$$h(k,i) = (h(k) + c_1.i + c_2.i^2) \mod m$$

- *Double hashing*:

$$h(k,i) = (h_1(k) + i.h_2(k)) \mod m$$

La secuencia para una clave k será $[h(k,0), ..., h(k,m-1)]$.

Factor de carga

Sobre el tamaño de nuestra tabla...

- ¿Qué pasa si es muy chico? (Caso límite: $|T| = 1$).
- ¿Qué pasa si es muy grande? (Caso límite: $|T| = |U|$).

Definimos el factor de carga como: $\alpha = \dfrac{n}{|T|}$

Donde n es el número de elementos guardados y $|T|$ el tamaño de la tabla.

- Cuando el factor de carga es grande, agrandamos la tabla.
- Cuando el factor de carga es chico, achicamos la tabla.

¿Cuáles serían buenas cotas para α?

Otros usos de funciones de hash

Las funciones de hash sirven para muchas cosas...

- Diccionarios sobre hash.

Otros usos de funciones de hash

Las funciones de hash sirven para muchas cosas...

- Diccionarios sobre hash.
- Criptografía.

Otros usos de funciones de hash

Las funciones de hash sirven para muchas cosas...

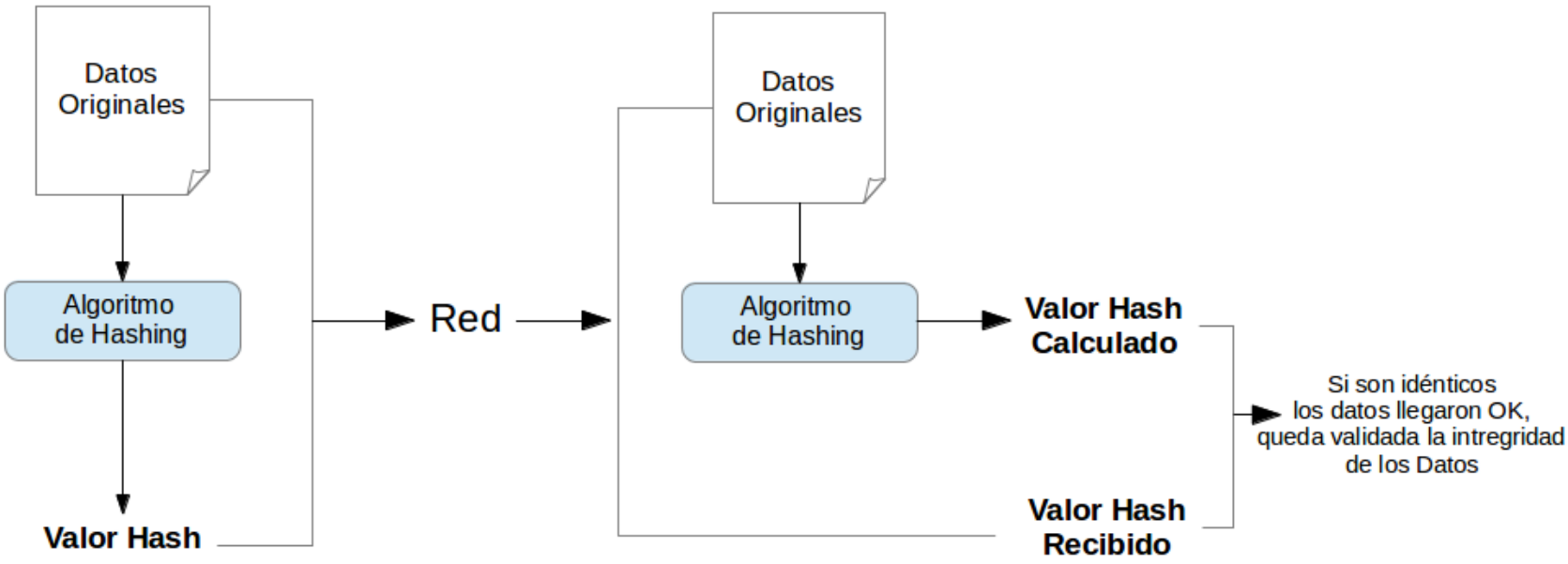
- Diccionarios sobre hash.
- Criptografía.
- Detección de errores, integridad de datos.

Otros usos de funciones de hash

Las funciones de hash sirven para muchas cosas...

- Diccionarios sobre hash.
- Criptografía.
- Detección de errores, integridad de datos.
- Comparación de datos.

Otros usos de funciones de hash



Taller de hoy

- Implementar un diccionario de claves string sobre una estructura que utilice *hashing abierto (chaining)*.
- Cuando el *factor de carga* supere cierto valor, debemos redimensionar el vector.
- Debemos implementar: definido?, definir, significado, borrar... *redimensionar* y *hash*.

FAQ

Pregunta 1

- ¿Qué es redimensionar? ¿cuándo y cómo lo invocamos?

FAQ

Pregunta 1

- ¿Qué es redimensionar? ¿cuándo y cómo lo invocamos?

Respuesta

- Redimensionar duplica la longitud del vector y vuelve a insertar todos los valores.
- Se invoca cuando *definimos* una nueva clave se corrobora que no se supere el umbral predefinido.

FAQ II

Pregunta 2

- ¿Cómo implementamos la función de hash?

FAQ II

Pregunta 2

- ¿Cómo implementamos la función de hash?

Respuesta

- *Hint:* `unsigned int fn_hash(string clave)`
- Ojo con las colisiones...
  1. ¿Castear cada `char` a `int` y sumarlos? $\Rightarrow h(\texttt{"ab"}) = h(\texttt{"ba"})$
  2. Cormen et al. *Introduction to Algorithms*. Sección 11.3.
  3. Knuth. *The Art of Computer Programming*. Sección 6.4.
  4. ¿Internet?
  5. Magia oscura.

FAQ III

Pregunta 3

- ¿Cómo utilizamos la función de hash?

FAQ III

Pregunta 3

- ¿Cómo utilizamos la función de hash?

Respuesta

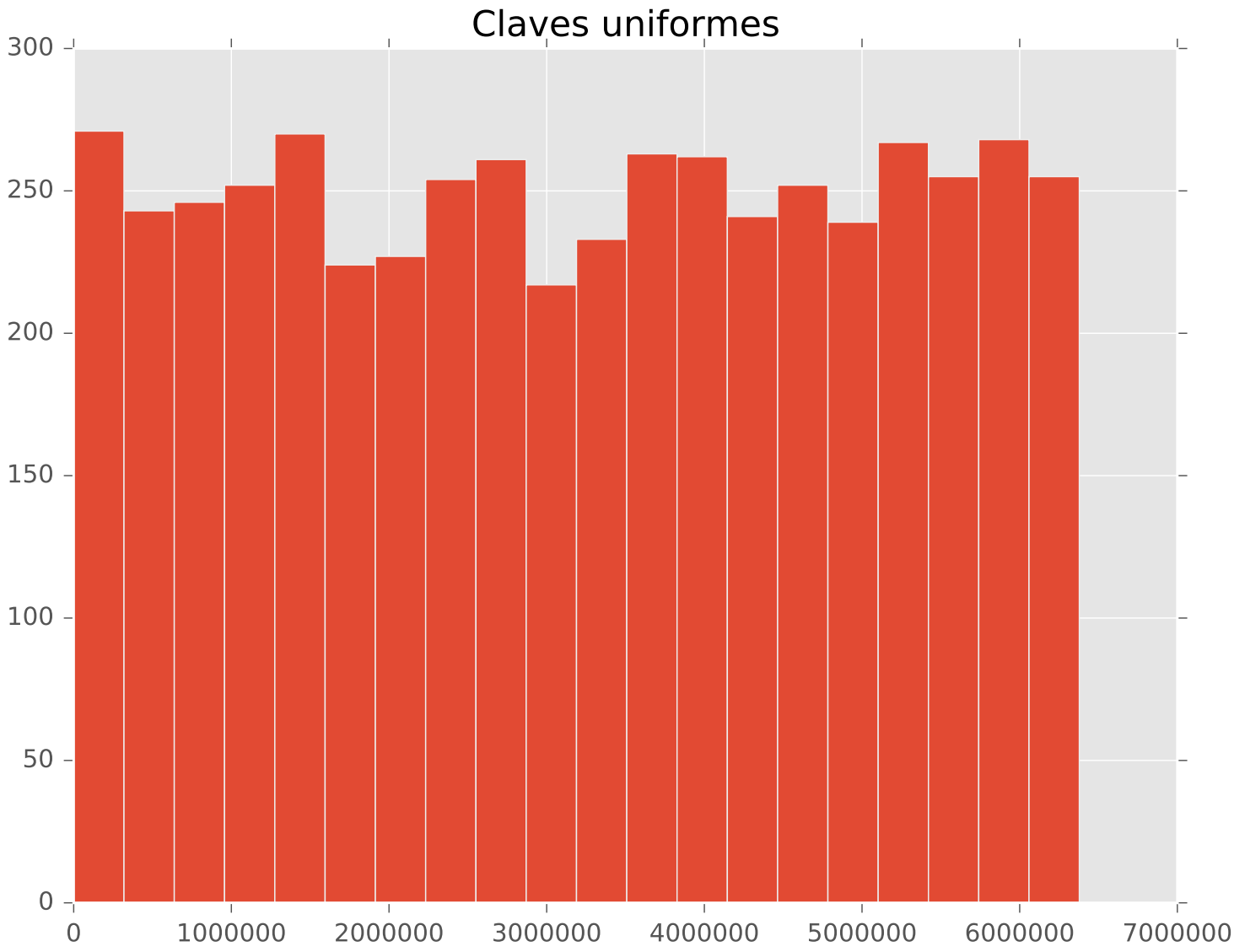
- Cada vez que queremos determinar la posición de una clave en el vector:

$$\texttt{posición} = \texttt{fn\_hash(clave)} \ \textbf{mod} \ |T|$$

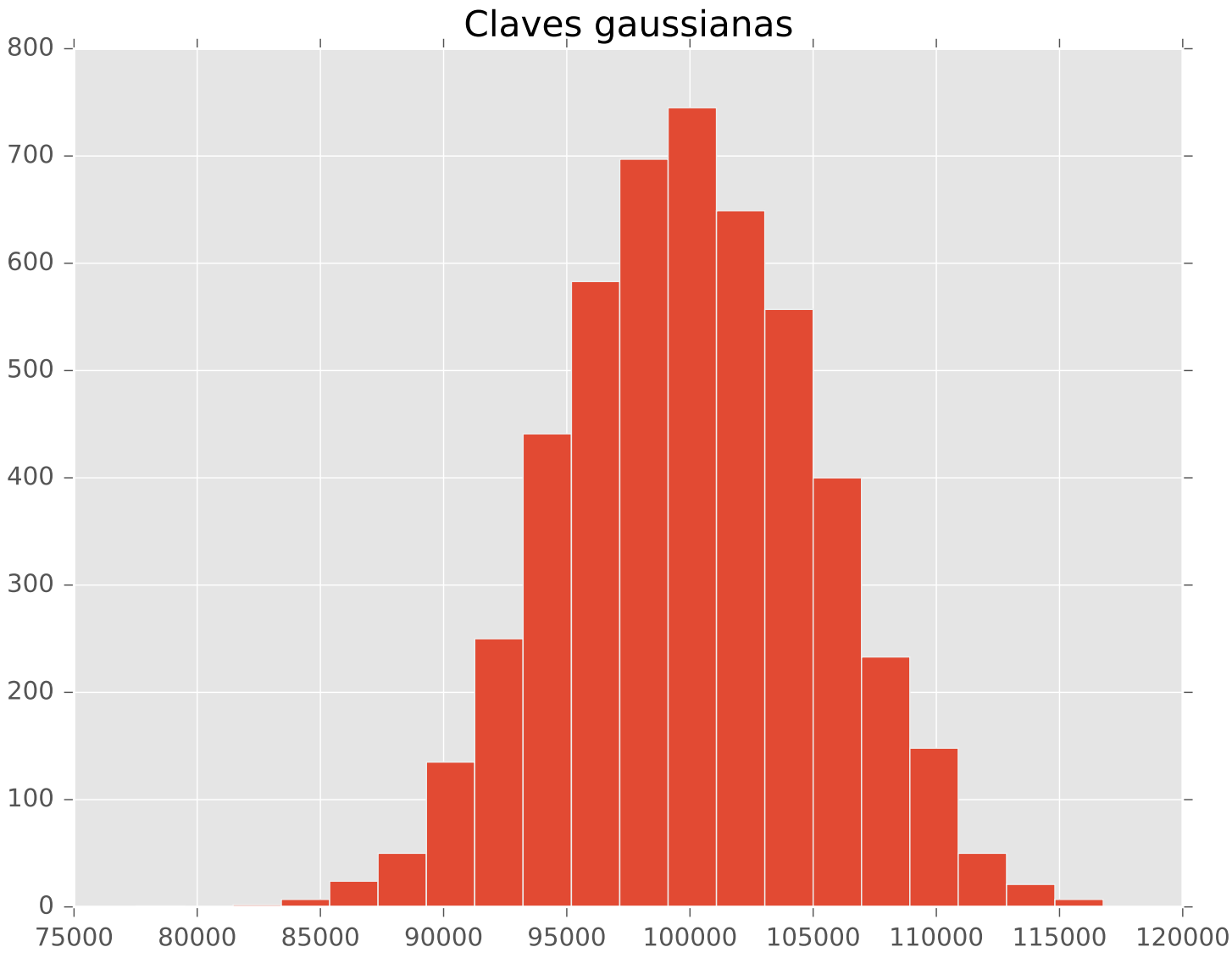
Más

- El testing también evaluará la performance de la estructura.
- Inserciones masivas con distintas distribuciones.

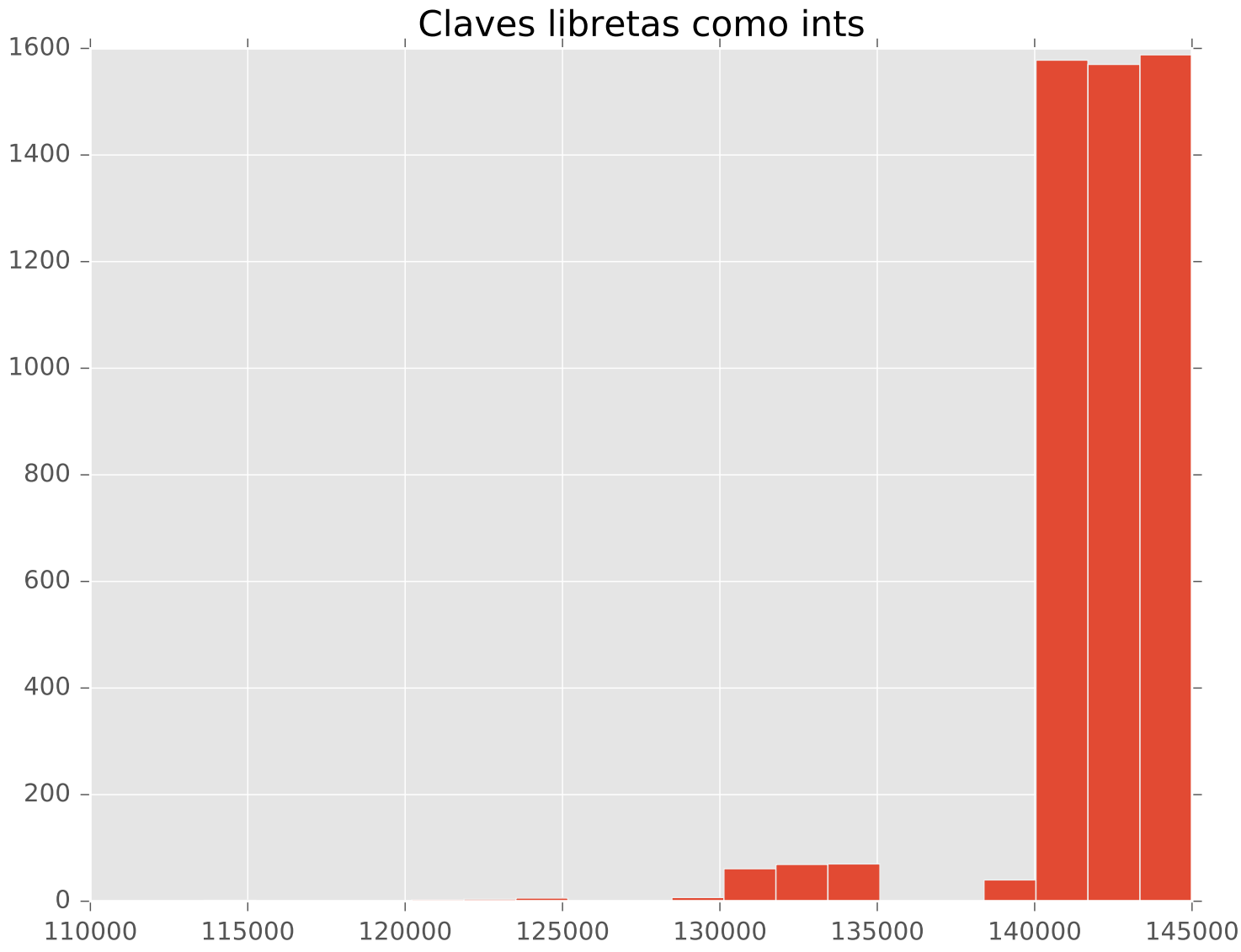
Testing


Claves uniformes
300
250
200
150
100
50
0
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000

Testing


Claves gaussianas
800
700
600
500
400
300
200
100
0
75000
80000
85000
90000
95000
100000
105000
110000
115000
120000

Testing


Claves libretas como ints
1600
1400
1200
1000
800
600
400
200
0
110000
115000
120000
125000
130000
135000
140000
145000

¿Preguntas?