

# Tablas de hash

Clase 13

IIC 2133 - Sección 3

Prof. Eduardo Bustos

# Sumario

**Introducción**

Tablas de hash

Colisiones

Cierre

# Recordatorio: Diccionarios

## Definición

Un **diccionario** es una estructura de datos con las siguientes operaciones

- **Asociar** un valor a una llave
- **Actualizar** el valor asociado a una llave
- **Obtener** el valor asociado a una llave
- En ciertos casos, **eliminar** de la estructura una asociación llave-valor

# Recordatorio: Diccionarios

## Definición

Un **diccionario** es una estructura de datos con las siguientes operaciones

- **Asociar** un valor a una llave
- **Actualizar** el valor asociado a una llave
- **Obtener** el valor asociado a una llave
- En ciertos casos, **eliminar** de la estructura una asociación llave-valor

Los ABB fueron nuestra primera EDD para implementar diccionarios

# Diccionarios

Los ABB efectivamente soportan las operaciones de diccionario

# Diccionarios

Los ABB efectivamente soportan las operaciones de diccionario

- La complejidad de las operaciones es  $\mathcal{O}(h)$

# Diccionarios

Los ABB efectivamente soportan las operaciones de diccionario

- La complejidad de las operaciones es  $\mathcal{O}(h)$
- Cuando están balanceados,  $h \in \mathcal{O}(\log(n))$  para  $n$  llaves almacenadas

# Diccionarios

Los ABB efectivamente soportan las operaciones de diccionario

- La complejidad de las operaciones es  $\mathcal{O}(h)$
- Cuando están balanceados,  $h \in \mathcal{O}(\log(n))$  para  $n$  llaves almacenadas

## Ejemplo

Podemos mantener pares (llave, valor) de la forma (rut, archivo)



# Diccionarios

Los ABB efectivamente soportan las operaciones de diccionario

- La complejidad de las operaciones es  $\mathcal{O}(h)$
- Cuando están balanceados,  $h \in \mathcal{O}(\log(n))$  para  $n$  llaves almacenadas

## Ejemplo

Podemos mantener pares (llave, valor) de la forma (rut, archivo)

- Podemos saber si un rut está en el dic. haciendo búsqueda por rut

# Diccionarios

Los ABB efectivamente soportan las operaciones de diccionario

- La complejidad de las operaciones es  $\mathcal{O}(h)$
- Cuando están balanceados,  $h \in \mathcal{O}(\log(n))$  para  $n$  llaves almacenadas

## Ejemplo

Podemos mantener pares (llave, valor) de la forma (rut, archivo)

- Podemos saber si un rut está en el dic. haciendo búsqueda por rut
- Inserción usa rut para ubicar el nodo, balanceando si es necesario

# Diccionarios

Los ABB efectivamente soportan las operaciones de diccionario

- La complejidad de las operaciones es  $\mathcal{O}(h)$
- Cuando están balanceados,  $h \in \mathcal{O}(\log(n))$  para  $n$  llaves almacenadas

## Ejemplo

Podemos mantener pares (llave, valor) de la forma (rut, archivo)

- Podemos saber si un rut está en el dic. haciendo búsqueda por rut
- Inserción usa rut para ubicar el nodo, balanceando si es necesario

Hay algo más que los ABB poseen y no es un requisito de los diccionarios

# Diccionarios

Los ABB no solo soportan las operaciones de diccionario

# Diccionarios

Los ABB no solo soportan las operaciones de diccionario

- La propiedad de **árbol de búsqueda** garantiza que los datos están **ordenados**

# Diccionarios

Los ABB no solo soportan las operaciones de diccionario

- La propiedad de **árbol de búsqueda** garantiza que los datos están **ordenados**
- Si el orden es importante, esto es necesario

# Diccionarios

Los ABB no solo soportan las operaciones de diccionario

- La propiedad de **árbol de búsqueda** garantiza que los datos están **ordenados**
- Si el orden es importante, esto es necesario

¿Qué es lo más importante en un diccionario?

# Diccionarios

El principal objetivo de los diccionarios es **búsqueda eficiente** de llaves



# Diccionarios

El principal objetivo de los diccionarios es **búsqueda eficiente** de llaves

- El objetivo secundario es **inserción/modificación eficiente** de pares llave-valor

# Diccionarios

El principal objetivo de los diccionarios es **búsqueda eficiente** de llaves

- El objetivo secundario es **inserción/modificación eficiente** de pares llave-valor
- Por esto, el orden de las llaves deja de ser relevante

# Diccionarios

El principal objetivo de los diccionarios es **búsqueda eficiente** de llaves

- El objetivo secundario es **inserción/modificación eficiente** de pares llave-valor
- Por esto, el orden de las llaves deja de ser relevante

¿Podemos buscar e insertar más rápido  
si nos olvidamos de mantener el orden?

# Diccionarios

Para motivar nuestra siguiente estructura, consideremos un escenario ideal

# Diccionarios

Para motivar nuestra siguiente estructura, consideremos un escenario ideal

- Conjunto de llaves posibles  $K = \{0, \dots, 11\}$  fijo y conocido

# Diccionarios

Para motivar nuestra siguiente estructura, consideremos un escenario ideal

- Conjunto de llaves posibles  $K = \{0, \dots, 11\}$  fijo y conocido
- Dada una llave  $k \in K$ , interesa saber si esta se encuentra asociada a un valor en la EDD

# Diccionarios

Para motivar nuestra siguiente estructura, consideremos un escenario ideal

- Conjunto de llaves posibles  $K = \{0, \dots, 11\}$  fijo y conocido
- Dada una llave  $k \in K$ , interesa saber si esta se encuentra asociada a un valor en la EDD

Este escenario se puede manejar con la siguiente EDD básica

# Diccionarios

Para motivar nuestra siguiente estructura, consideremos un escenario ideal

- Conjunto de llaves posibles  $K = \{0, \dots, 11\}$  fijo y conocido
- Dada una llave  $k \in K$ , interesa saber si esta se encuentra asociada a un valor en la EDD

Este escenario se puede manejar con la siguiente EDD básica

- Arreglo  $A[0 \dots 11]$  iniciado con  $\emptyset$  en cada celda



# Diccionarios

Para motivar nuestra siguiente estructura, consideremos un escenario ideal

- Conjunto de llaves posibles  $K = \{0, \dots, 11\}$  fijo y conocido
- Dada una llave  $k \in K$ , interesa saber si esta se encuentra asociada a un valor en la EDD

Este escenario se puede manejar con la siguiente EDD básica

- Arreglo  $A[0 \dots 11]$  iniciado con  $\emptyset$  en cada celda
- No almacenamos las llaves en las celdas, sino un puntero al valor asociado a la llave  $k$  en  $A[k]$

# Diccionarios

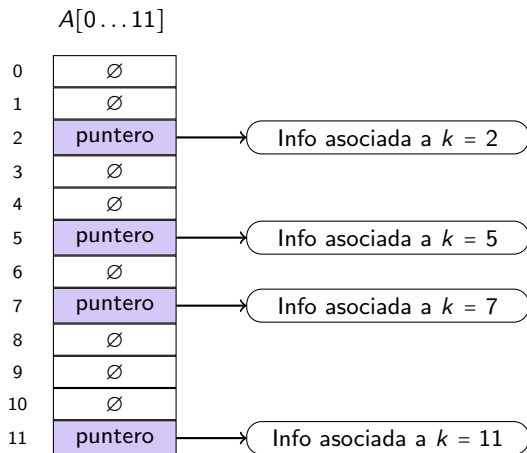
Para motivar nuestra siguiente estructura, consideremos un escenario ideal

- Conjunto de llaves posibles  $K = \{0, \dots, 11\}$  fijo y conocido
- Dada una llave  $k \in K$ , interesa saber si esta se encuentra asociada a un valor en la EDD

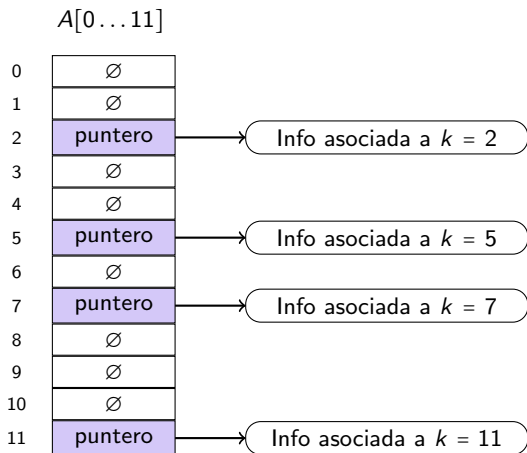
Este escenario se puede manejar con la siguiente EDD básica

- Arreglo  $A[0 \dots 11]$  iniciado con  $\emptyset$  en cada celda
- No almacenamos las llaves en las celdas, sino un puntero al valor asociado a la llave  $k$  en  $A[k]$
- Es decir,  $A[k] = \emptyset \Leftrightarrow$  no hay valor asociado a  $k$  en  $A$

# Diccionarios



# Diccionarios



¿Cuál es la complejidad de la búsqueda y la inserción en  $A$ ?

# Diccionarios

En la estructura  $A$  los accesos a  $A[k]$  son **accesos por índice**

# Diccionarios

En la estructura  $A$  los accesos a  $A[k]$  son **accesos por índice**

- Verificar si  $A[k] = \emptyset$  es  $\mathcal{O}(1)$

# Diccionarios

En la estructura  $A$  los accesos a  $A[k]$  son **accesos por índice**

- Verificar si  $A[k] = \emptyset$  es  $\mathcal{O}(1)$
- Insertar/modificar valor en  $A[k]$  es  $\mathcal{O}(1)$

# Diccionarios

En la estructura  $A$  los accesos a  $A[k]$  son **accesos por índice**

- Verificar si  $A[k] = \emptyset$  es  $\mathcal{O}(1)$
- Insertar/modificar valor en  $A[k]$  es  $\mathcal{O}(1)$

No solo las operaciones deseadas son súper eficientes



# Diccionarios

En la estructura  $A$  los accesos a  $A[k]$  son **accesos por índice**

- Verificar si  $A[k] = \emptyset$  es  $\mathcal{O}(1)$
- Insertar/modificar valor en  $A[k]$  es  $\mathcal{O}(1)$

No solo las operaciones deseadas son súper eficientes

- A diferencia de un ABB, se almacena solo un puntero al valor guardado

# Diccionarios

En la estructura  $A$  los accesos a  $A[k]$  son **accesos por índice**

- Verificar si  $A[k] = \emptyset$  es  $\mathcal{O}(1)$
- Insertar/modificar valor en  $A[k]$  es  $\mathcal{O}(1)$

No solo las operaciones deseadas son súper eficientes

- A diferencia de un ABB, se almacena solo un puntero al valor guardado
- No se usan punteros a padres-hijos para mantener la estructura

# Diccionarios

En la estructura  $A$  los accesos a  $A[k]$  son **accesos por índice**

- Verificar si  $A[k] = \emptyset$  es  $\mathcal{O}(1)$
- Insertar/modificar valor en  $A[k]$  es  $\mathcal{O}(1)$

No solo las operaciones deseadas son súper eficientes

- A diferencia de un ABB, se almacena solo un puntero al valor guardado
- No se usan punteros a padres-hijos para mantener la estructura

¿Qué tan ideal es este escenario de llaves naturales  $K$ ?

# Diccionarios

El escenario de llaves  $K$  puede ocurrir en aplicaciones prácticas

# Diccionarios

El escenario de llaves  $K$  puede ocurrir en aplicaciones prácticas

- Rango de valores razonable

# Diccionarios

El escenario de llaves  $K$  puede ocurrir en aplicaciones prácticas

- Rango de valores razonable
- Llaves siempre naturales (para ser usadas como índices de arreglos)

# Diccionarios

El escenario de llaves  $K$  puede ocurrir en aplicaciones prácticas

- Rango de valores razonable
- Llaves siempre naturales (para ser usadas como índices de arreglos)

## Ejemplo

En la universidad hay aproximadamente 25.000 estudiantes este año.

# Diccionarios

El escenario de llaves  $K$  puede ocurrir en aplicaciones prácticas

- Rango de valores razonable
- Llaves siempre naturales (para ser usadas como índices de arreglos)

## Ejemplo

En la universidad hay aproximadamente 25.000 estudiantes este año.

- Asignamos un  $k \in \{0, \dots, 24.999\}$  a cada estudiante



# Diccionarios

El escenario de llaves  $K$  puede ocurrir en aplicaciones prácticas

- Rango de valores razonable
- Llaves siempre naturales (para ser usadas como índices de arreglos)

## Ejemplo

En la universidad hay aproximadamente 25.000 estudiantes este año.

- Asignamos un  $k \in \{0, \dots, 24.999\}$  a cada estudiante
- Usamos cada natural como índice del arreglo  $A$

# Diccionarios

## Ejemplo

Cada estudiante ya posee un rut único

# Diccionarios

## Ejemplo

Cada estudiante ya posee un rut único

- Rango de rut's abarca hasta el 25.000.000

# Diccionarios

## Ejemplo

Cada estudiante ya posee un rut único

- Rango de rut's abarca hasta el 25.000.000
- Cantidad de estudiantes mucho menor (25.000)

# Diccionarios

## Ejemplo

Cada estudiante ya posee un rut único

- Rango de rut's abarca hasta el 25.000.000
- Cantidad de estudiantes mucho menor (25.000)
- **Problema:** solo 1/1000 celdas del arreglo  $A$  indexado por ruts estarán ocupadas

# Diccionarios

## Ejemplo

Cada estudiante ya posee un rut único

- Rango de rut's abarca hasta el 25.000.000
- Cantidad de estudiantes mucho menor (25.000)
- **Problema:** solo 1/1000 celdas del arreglo  $A$  indexado por ruts estarán ocupadas

No solo los ruts son llaves posibles

# Diccionarios

## Ejemplo

Cada estudiante ya posee un rut único

- Rango de rut's abarca hasta el 25.000.000
- Cantidad de estudiantes mucho menor (25.000)
- **Problema:** solo 1/1000 celdas del arreglo  $A$  indexado por ruts estarán ocupadas

No solo los ruts son llaves posibles

- Números de teléfono

# Diccionarios

## Ejemplo

Cada estudiante ya posee un rut único

- Rango de rut's abarca hasta el 25.000.000
- Cantidad de estudiantes mucho menor (25.000)
- **Problema:** solo 1/1000 celdas del arreglo  $A$  indexado por ruts estarán ocupadas

No solo los ruts son llaves posibles

- Números de teléfono
- Patentes de vehículos



# Diccionarios

## Ejemplo

Cada estudiante ya posee un rut único

- Rango de rut's abarca hasta el 25.000.000
- Cantidad de estudiantes mucho menor (25.000)
- **Problema:** solo 1/1000 celdas del arreglo  $A$  indexado por ruts estarán ocupadas

No solo los ruts son llaves posibles

- Números de teléfono
- Patentes de vehículos
- ...

# Diccionarios

## Ejemplo

Cada estudiante ya posee un rut único

- Rango de rut's abarca hasta el 25.000.000
- Cantidad de estudiantes mucho menor (25.000)
- **Problema:** solo 1/1000 celdas del arreglo  $A$  indexado por ruts estarán ocupadas

No solo los ruts son llaves posibles

- Números de teléfono
- Patentes de vehículos
- ...

¿Cómo acercarnos a un conjunto de llaves  $K$  razonable?

# Objetivos de la clase

# Objetivos de la clase

- ☐ Comprender el concepto de función de hash

# Objetivos de la clase

- ☐ Comprender el concepto de función de hash
- ☐ Identificar limitaciones en el almacenamiento a través de arreglos indexados

# Objetivos de la clase

- ☐ Comprender el concepto de función de hash
- ☐ Identificar limitaciones en el almacenamiento a través de arreglos indexados
- ☐ Comprender concepto de tabla de hash

# Objetivos de la clase

- ☐ Comprender el concepto de función de hash
- ☐ Identificar limitaciones en el almacenamiento a través de arreglos indexados
- ☐ Comprender concepto de tabla de hash
- ☐ Comprender concepto de colisión y sus posibles manejos

# Objetivos de la clase

- ☐ Comprender el concepto de función de hash
- ☐ Identificar limitaciones en el almacenamiento a través de arreglos indexados
- ☐ Comprender concepto de tabla de hash
- ☐ Comprender concepto de colisión y sus posibles manejos
- ☐ Distinguir diferencias entre encadenamiento y direccionamiento abierto



# Sumario

Introducción

**Tablas de hash**

Colisiones

Cierre

# Funciones de hash

# Funciones de hash

## Definición

Dado un espacio de llaves  $K$  y un natural  $m > 0$ , una **función de hash** se define como

$$h : K \rightarrow \{0, \dots, m-1\}$$

Dado  $k \in K$ , llamaremos **valor de hash de  $k$**  a la evaluación  $h(k)$ .

# Funciones de hash

## Definición

Dado un espacio de llaves  $K$  y un natural  $m > 0$ , una **función de hash** se define como

$$h : K \rightarrow \{0, \dots, m-1\}$$

Dado  $k \in K$ , llamaremos **valor de hash de  $k$**  a la evaluación  $h(k)$ .

Notemos que

# Funciones de hash

## Definición

Dado un espacio de llaves  $K$  y un natural  $m > 0$ , una **función de hash** se define como

$$h : K \rightarrow \{0, \dots, m-1\}$$

Dado  $k \in K$ , llamaremos **valor de hash de  $k$**  a la evaluación  $h(k)$ .

Notemos que

- Una función de hash nos permite mapear un espacio de llaves a otro más pequeño (con  $m$  razonable)

# Funciones de hash

## Definición

Dado un espacio de llaves  $K$  y un natural  $m > 0$ , una **función de hash** se define como

$$h : K \rightarrow \{0, \dots, m-1\}$$

Dado  $k \in K$ , llamaremos **valor de hash de  $k$**  a la evaluación  $h(k)$ .

Notemos que

- Una función de hash nos permite mapear un espacio de llaves a otro más pequeño (con  $m$  razonable)
- Una función de hash no necesariamente es **inyectiva**

# Funciones de hash

## Definición

Dado un espacio de llaves  $K$  y un natural  $m > 0$ , una **función de hash** se define como

$$h : K \rightarrow \{0, \dots, m-1\}$$

Dado  $k \in K$ , llamaremos **valor de hash de  $k$**  a la evaluación  $h(k)$ .

Notemos que

- Una función de hash nos permite mapear un espacio de llaves a otro más pequeño (con  $m$  razonable)
- Una función de hash no necesariamente es **inyectiva**
- Si  $m < |K|$ , no puede ser inyectiva

# Funciones de hash

## Definición

Dado un espacio de llaves  $K$  y un natural  $m > 0$ , una **función de hash** se define como

$$h : K \rightarrow \{0, \dots, m-1\}$$

Dado  $k \in K$ , llamaremos **valor de hash de  $k$**  a la evaluación  $h(k)$ .

Notemos que

- Una función de hash nos permite mapear un espacio de llaves a otro más pequeño (con  $m$  razonable)
- Una función de hash no necesariamente es **inyectiva**
- Si  $m < |K|$ , no puede ser inyectiva
- En la práctica,  $m \ll |K|$



# Tablas de hash

# Tablas de hash

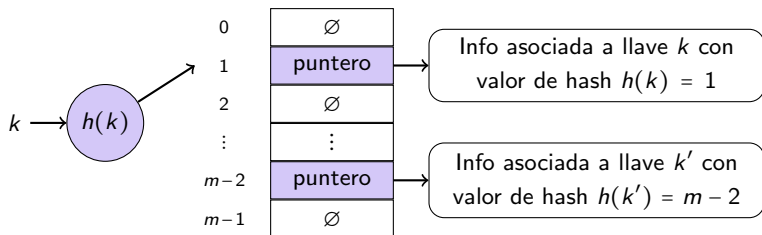
## Definición

Dado  $m > 0$  y un conjunto de llaves  $K$ , una **tabla de hash**  $A$  es una EDD que asocia valores a llaves indexadas usando una función de hash  $h: K \rightarrow \{0, \dots, m-1\}$ . Diremos que tal  $A$  es de tamaño  $m$ .

# Tablas de hash

## Definición

Dado  $m > 0$  y un conjunto de llaves  $K$ , una **tabla de hash**  $A$  es una EDD que asocia valores a llaves indexadas usando una función de hash  $h: K \rightarrow \{0, \dots, m-1\}$ . Diremos que tal  $A$  es de tamaño  $m$ .



# Tabla de hash

El ejemplo *ideal* que estudiamos es una tabla de hash

# Tabla de hash

El ejemplo *ideal* que estudiamos es una tabla de hash

- La función de hash es  $h : K \rightarrow K$  dada por

$$h(k) = k$$

# Tabla de hash

El ejemplo *ideal* que estudiamos es una tabla de hash

- La función de hash es  $h : K \rightarrow K$  dada por

$$h(k) = k$$

- Las operaciones de diccionario son sencillas

# Tabla de hash

El ejemplo *ideal* que estudiamos es una tabla de hash

- La función de hash es  $h : K \rightarrow K$  dada por

$$h(k) = k$$

- Las operaciones de diccionario son sencillas

IdentityHashSearch ( $A, k$ ):

**return**  $A[k]$

IdentityHashInsert ( $A, k, v$ ):

$A[k] = v$

IdentityHashDelete ( $A, k$ ):

$A[k] = \emptyset$

# Tabla de hash

El ejemplo *ideal* que estudiamos es una tabla de hash

- La función de hash es  $h : K \rightarrow K$  dada por

$$h(k) = k$$

- Las operaciones de diccionario son sencillas

`IdentityHashSearch (A, k):`

**return**  $A[k]$

`IdentityHashInsert (A, k, v):`

$A[k] = v$

`IdentityHashDelete (A, k):`

$A[k] = \emptyset$

0	$\emptyset$
1	$\emptyset$
2	puntero
3	$\emptyset$
4	$\emptyset$
5	puntero
6	$\emptyset$
7	puntero
8	$\emptyset$
9	$\emptyset$
10	$\emptyset$
11	puntero



# Tabla de hash

- Usar la misma estrategia para hashing general sería

# Tabla de hash

- Usar la misma estrategia para hashing general sería

HashSearch ( $A, k$ ):

**return**  $A[h(k)]$

HashInsert ( $A, k, v$ ):

$A[h(k)] = v$

HashDelete ( $A, k$ ):

$A[h(k)] = \emptyset$

# Tabla de hash

- Usar la misma estrategia para hashing general sería

HashSearch ( $A, k$ ):

**return**  $A[h(k)]$

HashInsert ( $A, k, v$ ):

$A[h(k)] = v$

HashDelete ( $A, k$ ):

$A[h(k)] = \emptyset$

- Pero sabemos que  $h$  no necesariamente es inyectiva

# Tabla de hash

- Usar la misma estrategia para hashing general sería

HashSearch ( $A, k$ ):

**return**  $A[h(k)]$

HashInsert ( $A, k, v$ ):

$A[h(k)] = v$

HashDelete ( $A, k$ ):

$A[h(k)] = \emptyset$

- Pero sabemos que  $h$  no necesariamente es inyectiva
- Es decir, puede ocurrir una **colisión**  $h(k_1) = h(k_2)$  para  $k_1 \neq k_2$

# Tabla de hash

- Usar la misma estrategia para hashing general sería

HashSearch ( $A, k$ ):

**return**  $A[h(k)]$

HashInsert ( $A, k, v$ ):

$A[h(k)] = v$

HashDelete ( $A, k$ ):

$A[h(k)] = \emptyset$

- Pero sabemos que  $h$  no necesariamente es inyectiva
- Es decir, puede ocurrir una **colisión**  $h(k_1) = h(k_2)$  para  $k_1 \neq k_2$

Veremos formas de manejar las colisiones

STORY TIME

# Sumario

Introducción

Tablas de hash

**Colisiones**

Cierre

# Una función de hash típica



# Una función de hash típica

Para ejemplificar el problema de las colisiones, consideremos la siguiente función de hash

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

# Una función de hash típica

Para ejemplificar el problema de las colisiones, consideremos la siguiente función de hash

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

Se le conoce como **hashing modular** y corresponde al resto al dividir  $k$  entre  $m$

# Una función de hash típica

Para ejemplificar el problema de las colisiones, consideremos la siguiente función de hash

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

Se le conoce como **hashing modular** y corresponde al resto al dividir  $k$  entre  $m$

- Notemos que  $h(k) \in \{0, \dots, m-1\}$  para todo  $k$

# Una función de hash típica

Para ejemplificar el problema de las colisiones, consideremos la siguiente función de hash

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

Se le conoce como **hashing modular** y corresponde al resto al dividir  $k$  entre  $m$

- Notemos que  $h(k) \in \{0, \dots, m-1\}$  para todo  $k$
- Todas las llaves con el mismo resto al dividir entre  $m$  generan una colisión, i.e.

$$h(k_1) = h(k_2) \Leftrightarrow k_1 \equiv_m k_2$$

# Una función de hash típica

Para ejemplificar el problema de las colisiones, consideremos la siguiente función de hash

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

Se le conoce como **hashing modular** y corresponde al resto al dividir  $k$  entre  $m$

- Notemos que  $h(k) \in \{0, \dots, m-1\}$  para todo  $k$
- Todas las llaves con el mismo resto al dividir entre  $m$  generan una colisión, i.e.

$$h(k_1) = h(k_2) \Leftrightarrow k_1 \equiv_m k_2$$

## Ejemplo

Tomando  $m = 100$  y  $K = \{0, \dots, 999\}$ , el hashing modular cumple

# Una función de hash típica

Para ejemplificar el problema de las colisiones, consideremos la siguiente función de hash

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

Se le conoce como **hashing modular** y corresponde al resto al dividir  $k$  entre  $m$

- Notemos que  $h(k) \in \{0, \dots, m-1\}$  para todo  $k$
- Todas las llaves con el mismo resto al dividir entre  $m$  generan una colisión, i.e.

$$h(k_1) = h(k_2) \Leftrightarrow k_1 \equiv_m k_2$$

## Ejemplo

Tomando  $m = 100$  y  $K = \{0, \dots, 999\}$ , el hashing modular cumple

- $h(12) = h(112) = \dots = h(912) = 12$

# Una función de hash típica

Para ejemplificar el problema de las colisiones, consideremos la siguiente función de hash

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

Se le conoce como **hashing modular** y corresponde al resto al dividir  $k$  entre  $m$

- Notemos que  $h(k) \in \{0, \dots, m-1\}$  para todo  $k$
- Todas las llaves con el mismo resto al dividir entre  $m$  generan una colisión, i.e.

$$h(k_1) = h(k_2) \Leftrightarrow k_1 \equiv_m k_2$$

## Ejemplo

Tomando  $m = 100$  y  $K = \{0, \dots, 999\}$ , el hashing modular cumple

- $h(12) = h(112) = \dots = h(912) = 12$
- $h(18) = \dots = h(918) = 18$

# Inserción



# Inserción

Usaremos la función de hashing modular para experimentar con **inserciones**

# Inserción

Usaremos la función de hashing modular para experimentar con inserciones

Consideremos  $m = 7$ . Insertemos la llave 15 en la siguiente tabla de hash

# Inserción

Usaremos la función de hashing modular para experimentar con **inserciones**

Consideremos  $m = 7$ . Insertemos la llave 15 en la siguiente tabla de hash

- Su valor de hash es  $h(15) = 15 \bmod 7 = 1$

# Inserción

Usaremos la función de hashing modular para experimentar con **inserciones**

Consideremos  $m = 7$ . Insertemos la llave 15 en la siguiente tabla de hash

- Su valor de hash es  $h(15) = 15 \bmod 7 = 1$

0	∅
1	∅
2	∅
3	∅
4	∅
5	∅
6	∅

# Inserción

La posición  $h(15) = 1$  está libre y guardamos la llave

0	∅
1	∅
2	∅
3	∅
4	∅
5	∅
6	∅

0	∅
1	∅
2	∅
3	∅
4	∅
5	∅
6	∅

0	∅
1	15
2	∅
3	∅
4	∅
5	∅
6	∅

# Inserción

Ahora insertamos la llave 37

- Su valor de hash es  $h(37) = 37 \bmod 7 = 2$

0	∅
1	15
2	∅
3	∅
4	∅
5	∅
6	∅

0	∅
1	15
2	∅
3	∅
4	∅
5	∅
6	∅

0	∅
1	15
2	37
3	∅
4	∅
5	∅
6	∅

# Inserción

Ahora insertamos la llave 51

- Su valor de hash es  $h(51) = 51 \bmod 7 = 2$

0	∅	0	∅
1	15	1	15
2	37	2	37
3	∅	3	∅
4	∅	4	∅
5	∅	5	∅
6	∅	6	∅

# Inserción

Ahora insertamos la llave 51

- Su valor de hash es  $h(51) = 51 \bmod 7 = 2$

0	∅	0	∅
1	15	1	15
2	37	2	37
3	∅	3	∅
4	∅	4	∅
5	∅	5	∅
6	∅	6	∅

¿Qué hacemos con la colisión?



# Inserción con encadenamiento

# Inserción con encadenamiento

Primera propuesta: **encadenamiento**

# Inserción con encadenamiento

Primera propuesta: **encadenamiento**

- Cada valor guardado es un nodo de una lista ligada

# Inserción con encadenamiento

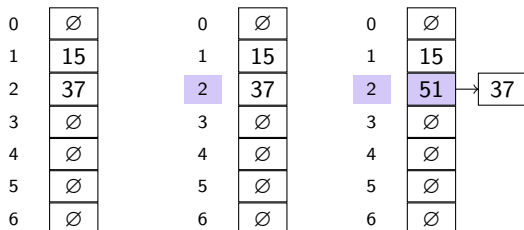
Primera propuesta: **encadenamiento**

- Cada valor guardado es un nodo de una lista ligada
- Cada colisión agrega un nodo al principio/final de la lista

# Inserción con encadenamiento

Primera propuesta: **encadenamiento**

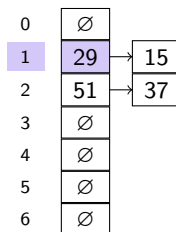
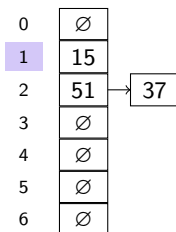
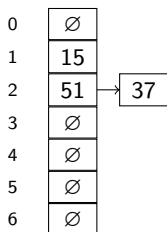
- Cada valor guardado es un nodo de una lista ligada
- Cada colisión agrega un nodo al principio/final de la lista



# Inserción con encadenamiento

Al insertar la llave 29 seguimos la misma idea

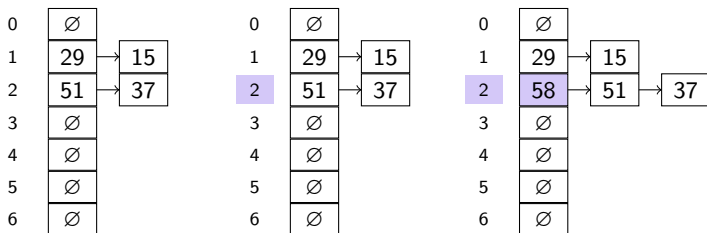
- Su valor de hash es  $h(29) = 29 \bmod 7 = 1$



# Inserción con encadenamiento

Al insertar la llave 58 seguimos la misma idea

- Su valor de hash es  $h(58) = 58 \bmod 7 = 2$



# Encadenamiento

- Las operaciones de diccionario involucran la lista ligada  $A[h(k)]$



# Encadenamiento

- Las operaciones de diccionario involucran la lista ligada  $A[h(k)]$

ChainedHashSearch ( $A, k$ ):

    Buscar llave  $k$  en  $A[h(k)]$

ChainedHashInsert ( $A, k, v$ ):

    Insertar  $(k, v)$  como cabeza de  $A[h(k)]$

ChainedHashDelete ( $A, k$ ):

    Eliminar llave  $k$  de  $A[h(k)]$

# Encadenamiento

- Las operaciones de diccionario involucran la lista ligada  $A[h(k)]$

ChainedHashSearch ( $A, k$ ):

    Buscar llave  $k$  en  $A[h(k)]$

ChainedHashInsert ( $A, k, v$ ):

    Insertar  $(k, v)$  como cabeza de  $A[h(k)]$

ChainedHashDelete ( $A, k$ ):

    Eliminar llave  $k$  de  $A[h(k)]$

- La complejidad de estas operaciones depende de qué tan largas sean las listas

# Encadenamiento

- Las operaciones de diccionario involucran la lista ligada  $A[h(k)]$

ChainedHashSearch ( $A, k$ ):

    Buscar llave  $k$  en  $A[h(k)]$

ChainedHashInsert ( $A, k, v$ ):

    Insertar  $(k, v)$  como cabeza de  $A[h(k)]$

ChainedHashDelete ( $A, k$ ):

    Eliminar llave  $k$  de  $A[h(k)]$

- La complejidad de estas operaciones depende de qué tan largas sean las listas
- Una *buena* función de hash repartiría las llaves de manera más o menos homogénea

# Encadenamiento

- Las operaciones de diccionario involucran la lista ligada  $A[h(k)]$

**ChainedHashSearch** ( $A, k$ ):

    Buscar llave  $k$  en  $A[h(k)]$

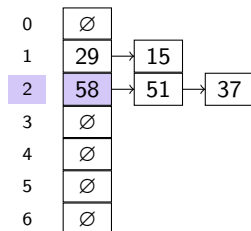
**ChainedHashInsert** ( $A, k, v$ ):

    Insertar  $(k, v)$  como cabeza de  $A[h(k)]$

**ChainedHashDelete** ( $A, k$ ):

    Eliminar llave  $k$  de  $A[h(k)]$

- La complejidad de estas operaciones depende de qué tan largas sean las listas
- Una *buena* función de hash repartiría las llaves de manera más o menos homogénea



# Otra estrategia para colisiones

Volvamos a la inserción con colisión del 51

- Su valor de hash es  $h(51) = 51 \bmod 7 = 2$

0	∅	0	∅
1	15	1	15
2	37	2	37
3	∅	3	∅
4	∅	4	∅
5	∅	5	∅
6	∅	6	∅

# Otra estrategia para colisiones

Volvamos a la inserción con colisión del 51

- Su valor de hash es  $h(51) = 51 \bmod 7 = 2$

0	∅	0	∅
1	15	1	15
2	37	2	37
3	∅	3	∅
4	∅	4	∅
5	∅	5	∅
6	∅	6	∅

¿Alguna alternativa al encadenamiento?

## Inserción con sondeo lineal

# Inserción con sondeo lineal

Segunda propuesta: **direccionamiento abierto**



# Inserción con sondeo lineal

Segunda propuesta: **direccionamiento abierto**

- Buscamos sistemáticamente una celda vacía

# Inserción con sondeo lineal

Segunda propuesta: **direccionamiento abierto**

- Buscamos sistemáticamente una celda vacía
- Puede producir nuevas colisiones no previstas por  $h$

# Inserción con sondeo lineal

Segunda propuesta: **direccionamiento abierto**

- Buscamos sistemáticamente una celda vacía
- Puede producir nuevas colisiones no previstas por  $h$

Una forma de buscar: el **sondeo lineal** inserta en la primera celda vacía a la derecha de la colisión

# Inserción con sondeo lineal

Segunda propuesta: **direccionamiento abierto**

- Buscamos sistemáticamente una celda vacía
- Puede producir nuevas colisiones no previstas por  $h$

Una forma de buscar: el **sondeo lineal** inserta en la primera celda vacía a la derecha de la colisión

0	∅
1	15
2	37
3	∅
4	∅
5	∅
6	∅

0	∅
1	15
2	37
3	∅
4	∅
5	∅
6	∅

0	∅
1	15
2	37
3	51
4	∅
5	∅
6	∅

# Insertión con sondeo lineal

Al insertar la llave 29 seguimos la misma idea del sondeo lineal

- Su valor de hash es  $h(29) = 29 \bmod 7 = 1$

0	∅
1	15
2	37
3	51
4	∅
5	∅
6	∅

0	∅
1	15
2	37
3	51
4	∅
5	∅
6	∅

0	∅
1	15
2	37
3	51
4	∅
5	∅
6	∅

0	∅
1	15
2	37
3	51
4	29
5	∅
6	∅

# Búsqueda con sondeo lineal

Si las inserciones son con sondeo lineal, la búsqueda debe tenerlo en cuenta

# Búsqueda con sondeo lineal

Si las inserciones son con sondeo lineal, la búsqueda debe tenerlo en cuenta

- No necesariamente  $k$  está guardado en  $A[h(k)]$

# Búsqueda con sondeo lineal

Si las inserciones son con sondeo lineal, la búsqueda debe tenerlo en cuenta

- No necesariamente  $k$  está guardado en  $A[h(k)]$
- Debemos revisar esa celda, y si no corresponde, buscar **hacia adelante**



# Búsqueda con sondeo lineal

Si las inserciones son con sondeo lineal, la búsqueda debe tenerlo en cuenta

- No necesariamente  $k$  está guardado en  $A[h(k)]$
- Debemos revisar esa celda, y si no corresponde, buscar **hacia adelante**

Por ejemplo, al buscar la llave 29 comenzamos la búsqueda en la pos. 1

# Búsqueda con sondeo lineal

Si las inserciones son con sondeo lineal, la búsqueda debe tenerlo en cuenta

- No necesariamente  $k$  está guardado en  $A[h(k)]$
- Debemos revisar esa celda, y si no corresponde, buscar **hacia adelante**

Por ejemplo, al buscar la llave 29 comenzamos la búsqueda en la pos. 1

0	∅
1	15
2	37
3	51
4	29
5	∅
6	∅

0	∅
1	15
2	37
3	51
4	29
5	∅
6	∅

0	∅
1	15
2	37
3	51
4	29
5	∅
6	∅

0	∅
1	15
2	37
3	51
4	29
5	∅
6	∅

# Búsqueda con sondeo lineal

Si las inserciones son con sondeo lineal, la búsqueda debe tenerlo en cuenta

- No necesariamente  $k$  está guardado en  $A[h(k)]$
- Debemos revisar esa celda, y si no corresponde, buscar **hacia adelante**

Por ejemplo, al buscar la llave 29 comenzamos la búsqueda en la pos. 1

0	$\emptyset$
1	15
2	37
3	51
4	29
5	$\emptyset$
6	$\emptyset$

0	$\emptyset$
1	15
2	37
3	51
4	29
5	$\emptyset$
6	$\emptyset$

0	$\emptyset$
1	15
2	37
3	51
4	29
5	$\emptyset$
6	$\emptyset$

0	$\emptyset$
1	15
2	37
3	51
4	29
5	$\emptyset$
6	$\emptyset$

La búsqueda sigue la misma secuencia que la inserción

# Búsqueda con sondeo lineal

¿Cómo detectamos si la llave no está?

# Búsqueda con sondeo lineal

¿Cómo detectamos si la llave no está?

- Comenzamos la búsqueda en  $A[h(k)]$

# Búsqueda con sondeo lineal

¿Cómo detectamos si la llave no está?

- Comenzamos la búsqueda en  $A[h(k)]$
- Si al buscar a la derecha llegamos a un  $\emptyset$  significa que no está

# Búsqueda con sondeo lineal

¿Cómo detectamos si la llave no está?

- Comenzamos la búsqueda en  $A[h(k)]$
- Si al buscar a la derecha llegamos a un  $\emptyset$  significa que no está

Por ejemplo, al buscar la llave 10, tal que  $h(10) = 10 \bmod 7 = 3$

# Búsqueda con sondeo lineal

¿Cómo detectamos si la llave no está?

- Comenzamos la búsqueda en  $A[h(k)]$
- Si al buscar a la derecha llegamos a un  $\emptyset$  significa que no está

Por ejemplo, al buscar la llave 10, tal que  $h(10) = 10 \bmod 7 = 3$

0	$\emptyset$
1	15
2	37
3	51
4	29
5	$\emptyset$
6	$\emptyset$

0	$\emptyset$
1	15
2	37
3	51
4	29
5	$\emptyset$
6	$\emptyset$

0	$\emptyset$
1	15
2	37
3	51
4	29
5	$\emptyset$
6	$\emptyset$



# Búsqueda con sondeo lineal

¿Cómo detectamos si la llave no está?

- Comenzamos la búsqueda en  $A[h(k)]$
- Si al buscar a la derecha llegamos a un  $\emptyset$  significa que no está

Por ejemplo, al buscar la llave 10, tal que  $h(10) = 10 \bmod 7 = 3$

0	$\emptyset$
1	15
2	37
3	51
4	29
5	$\emptyset$
6	$\emptyset$

0	$\emptyset$
1	15
2	37
3	51
4	29
5	$\emptyset$
6	$\emptyset$

0	$\emptyset$
1	15
2	37
3	51
4	29
5	$\emptyset$
6	$\emptyset$

Concluimos que 10 no está almacenada

# Eliminación con sondeo lineal

# Eliminación con sondeo lineal

Para eliminar llaves guardadas tenemos un problema

# Eliminación con sondeo lineal

Para eliminar llaves guardadas tenemos un problema

- Si borramos una llave, la reemplazamos por  $\emptyset$

# Eliminación con sondeo lineal

Para eliminar llaves guardadas tenemos un problema

- Si borramos una llave, la reemplazamos por  $\emptyset$

Por ejemplo, si borramos el 51 y buscamos el 29 con  $h(29) = 29 \bmod 7 = 1$

# Eliminación con sondeo lineal

Para eliminar llaves guardadas tenemos un problema

- Si borramos una llave, la reemplazamos por  $\emptyset$

Por ejemplo, si borramos el 51 y buscamos el 29 con  $h(29) = 29 \bmod 7 = 1$

0	$\emptyset$
1	15
2	37
3	51
4	29
5	$\emptyset$
6	$\emptyset$

0	$\emptyset$
1	15
2	37
3	$\emptyset$
4	29
5	$\emptyset$
6	$\emptyset$

0	$\emptyset$
1	15
2	37
3	$\emptyset$
4	29
5	$\emptyset$
6	$\emptyset$

0	$\emptyset$
1	15
2	37
3	$\emptyset$
4	29
5	$\emptyset$
6	$\emptyset$

# Eliminación con sondeo lineal

Para eliminar llaves guardadas tenemos un problema

- Si borramos una llave, la reemplazamos por  $\emptyset$

Por ejemplo, si borramos el 51 y buscamos el 29 con  $h(29) = 29 \bmod 7 = 1$

0	$\emptyset$
1	15
2	37
3	51
4	29
5	$\emptyset$
6	$\emptyset$

0	$\emptyset$
1	15
2	37
3	$\emptyset$
4	29
5	$\emptyset$
6	$\emptyset$

0	$\emptyset$
1	15
2	37
3	$\emptyset$
4	29
5	$\emptyset$
6	$\emptyset$

0	$\emptyset$
1	15
2	37
3	$\emptyset$
4	29
5	$\emptyset$
6	$\emptyset$

Concluimos que 29 no está almacenado...

# Eliminación con sondeo lineal

Para eliminar llaves guardadas tenemos un problema

- Si borramos una llave, la reemplazamos por  $\emptyset$

Por ejemplo, si borramos el 51 y buscamos el 29 con  $h(29) = 29 \bmod 7 = 1$

0	$\emptyset$
1	15
2	37
3	51
4	29
5	$\emptyset$
6	$\emptyset$

0	$\emptyset$
1	15
2	37
3	$\emptyset$
4	29
5	$\emptyset$
6	$\emptyset$

0	$\emptyset$
1	15
2	37
3	$\emptyset$
4	29
5	$\emptyset$
6	$\emptyset$

0	$\emptyset$
1	15
2	37
3	$\emptyset$
4	29
5	$\emptyset$
6	$\emptyset$

Concluimos que 29 no está almacenado...

Si necesitamos eliminación, es mejor usar encadenamiento



## Otros sondeos

# Otros sondeos

Sondeo lineal

# Otros sondeos

## Sondeo lineal

- Si  $h(k) = H$ , para alguna constante  $d$  buscamos en

$$H, H + d, H + 2d, \dots$$

# Otros sondeos

## Sondeo lineal

- Si  $h(k) = H$ , para alguna constante  $d$  buscamos en

$$H, H + d, H + 2d, \dots$$

- Se debe cumplir  $d = 1$  o que  $d$  y  $m$  son primos relativos

# Otros sondeos

## Sondeo lineal

- Si  $h(k) = H$ , para alguna constante  $d$  buscamos en

$$H, H + d, H + 2d, \dots$$

- Se debe cumplir  $d = 1$  o que  $d$  y  $m$  son primos relativos

## Sondeo cuadrático

# Otros sondeos

## Sondeo lineal

- Si  $h(k) = H$ , para alguna constante  $d$  buscamos en

$$H, H + d, H + 2d, \dots$$

- Se debe cumplir  $d = 1$  o que  $d$  y  $m$  son primos relativos

## Sondeo cuadrático

- Si  $h(k) = H$ , buscamos en

$$H, H + 1, H + 4, H + 9, \dots$$

# Otros sondeos

## Sondeo lineal

- Si  $h(k) = H$ , para alguna constante  $d$  buscamos en

$$H, H + d, H + 2d, \dots$$

- Se debe cumplir  $d = 1$  o que  $d$  y  $m$  son primos relativos

## Sondeo cuadrático

- Si  $h(k) = H$ , buscamos en

$$H, H + 1, H + 4, H + 9, \dots$$

## Doble hashing

# Otros sondeos

## Sondeo lineal

- Si  $h(k) = H$ , para alguna constante  $d$  buscamos en

$$H, H + d, H + 2d, \dots$$

- Se debe cumplir  $d = 1$  o que  $d$  y  $m$  son primos relativos

## Sondeo cuadrático

- Si  $h(k) = H$ , buscamos en

$$H, H + 1, H + 4, H + 9, \dots$$

## Doble hashing

- Usamos dos funciones de hash  $h_1$  y  $h_2$  y buscamos en

$$h_1(k), h_1(k) + h_2(k), h_1(k) + 2h_2(k), \dots$$



# Otros sondeos

## Sondeo lineal

- Si  $h(k) = H$ , para alguna constante  $d$  buscamos en

$$H, H + d, H + 2d, \dots$$

- Se debe cumplir  $d = 1$  o que  $d$  y  $m$  son primos relativos

## Sondeo cuadrático

- Si  $h(k) = H$ , buscamos en

$$H, H + 1, H + 4, H + 9, \dots$$

## Doble hashing

- Usamos dos funciones de hash  $h_1$  y  $h_2$  y buscamos en

$$h_1(k), h_1(k) + h_2(k), h_1(k) + 2h_2(k), \dots$$

Todos ellos presentan el problema de la eliminación

## Factor de carga

# Factor de carga

Dado que las colisiones impactan la tabla, nos interesa medir cuántos datos tenemos almacenados

# Factor de carga

Dado que las colisiones impactan la tabla, nos interesa medir cuántos datos tenemos almacenados

## Definición

Dada una tabla de hash  $A$  de tamaño  $m$  con  $n$  valores almacenados, se define su **factor de carga** como

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

# Factor de carga

Dado que las colisiones impactan la tabla, nos interesa medir cuántos datos tenemos almacenados

## Definición

Dada una tabla de hash  $A$  de tamaño  $m$  con  $n$  valores almacenados, se define su **factor de carga** como

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

El factor de carga es una medida de *qué tan llena está la tabla*

# Factor de carga

Dado que las colisiones impactan la tabla, nos interesa medir cuántos datos tenemos almacenados

## Definición

Dada una tabla de hash  $A$  de tamaño  $m$  con  $n$  valores almacenados, se define su **factor de carga** como

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

El factor de carga es una medida de *qué tan llena está la tabla*

Según la estrategia de resolución de colisiones

# Factor de carga

Dado que las colisiones impactan la tabla, nos interesa medir cuántos datos tenemos almacenados

## Definición

Dada una tabla de hash  $A$  de tamaño  $m$  con  $n$  valores almacenados, se define su **factor de carga** como

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

El factor de carga es una medida de *qué tan llena está la tabla*

Según la estrategia de resolución de colisiones

- Encadenamiento: es aceptable  $\lambda \approx 1$

# Factor de carga

Dado que las colisiones impactan la tabla, nos interesa medir cuántos datos tenemos almacenados

## Definición

Dada una tabla de hash  $A$  de tamaño  $m$  con  $n$  valores almacenados, se define su **factor de carga** como

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

El factor de carga es una medida de *qué tan llena está la tabla*

Según la estrategia de resolución de colisiones

- Encadenamiento: es aceptable  $\lambda \approx 1$
- Direccionamiento abierto:  $\lambda > 0.5$  resulta en inserciones y búsquedas muy lentas



# Rehashing

# Rehashing

Si  $\lambda$  es grande y ya no es aceptable, las operaciones se vuelven costosas

# Rehashing

Si  $\lambda$  es grande y ya no es aceptable, las operaciones se vuelven costosas

Una solución es hacer **rehashing**

# Rehashing

Si  $\lambda$  es grande y ya no es aceptable, las operaciones se vuelven costosas

Una solución es hacer **rehashing**

- Se crea una nueva tabla más grande

# Rehashing

Si  $\lambda$  es grande y ya no es aceptable, las operaciones se vuelven costosas

Una solución es hacer **rehashing**

- Se crea una nueva tabla más grande
- Aproximadamente del doble del tamaño original

# Rehashing

Si  $\lambda$  es grande y ya no es aceptable, las operaciones se vuelven costosas

Una solución es hacer **rehashing**

- Se crea una nueva tabla más grande
- Aproximadamente del doble del tamaño original
- Como el espacio de índices ya no es de tamaño  $m$ , se define una nueva función de hash

# Rehashing

Si  $\lambda$  es grande y ya no es aceptable, las operaciones se vuelven costosas

Una solución es hacer **rehashing**

- Se crea una nueva tabla más grande
- Aproximadamente del doble del tamaño original
- Como el espacio de índices ya no es de tamaño  $m$ , se define una nueva función de hash
- Mover los datos a la nueva tabla

# Rehashing

Si  $\lambda$  es grande y ya no es aceptable, las operaciones se vuelven costosas

Una solución es hacer **rehashing**

- Se crea una nueva tabla más grande
- Aproximadamente del doble del tamaño original
- Como el espacio de índices ya no es de tamaño  $m$ , se define una nueva función de hash
- Mover los datos a la nueva tabla

Esta es una operación costosa para tablas de hash



# Rehashing

Si  $\lambda$  es grande y ya no es aceptable, las operaciones se vuelven costosas

Una solución es hacer **rehashing**

- Se crea una nueva tabla más grande
- Aproximadamente del doble del tamaño original
- Como el espacio de índices ya no es de tamaño  $m$ , se define una nueva función de hash
- Mover los datos a la nueva tabla

Esta es una operación costosa para tablas de hash

- Es  $\mathcal{O}(n)$  para  $n$  datos insertados

# Rehashing

Si  $\lambda$  es grande y ya no es aceptable, las operaciones se vuelven costosas

Una solución es hacer **rehashing**

- Se crea una nueva tabla más grande
- Aproximadamente del doble del tamaño original
- Como el espacio de índices ya no es de tamaño  $m$ , se define una nueva función de hash
- Mover los datos a la nueva tabla

Esta es una operación costosa para tablas de hash

- Es  $\mathcal{O}(n)$  para  $n$  datos insertados
- No obstante, es **infrecuente**

# Sumario

Introducción

Tablas de hash

Colisiones

**Cierre**

Considere el problema de entregar los resultados de la prueba PAES del proceso de admisión a las universidades 2025. Este proceso es de muy alta carga en un período muy corto de tiempo. Ud. recibirá del DEMRE el archivo con los resultados solo 2 horas antes de la hora de publicación: 00:00 del día 20 de enero. Ud. considera que el servidor de bases de datos no sería capaz de soportar la carga por lo que considera usar una Tabla de Hash para almacenar los resultados, la cual sería poblada (Insert) entre las 10:00 y 11:59 del día previo (19 de enero).

Considere un rango de los RUN 1-25.000.000, pero solo 200 mil personas rinden la prueba PAES y el 80 % tienen un RUN entre 22.000.000 y 22.400.000

(a) (2 puntos) Suponiendo que el registro puede almacenarse directamente en una tabla de Hash (no es necesario el puntero al registro). Proponga una estructura de datos para dicha Tabla, indicando la función de hash utilizada, el tamaño de la tabla y su factor de carga esperado.

(b) (2 puntos) ¿Cómo cambia su solución si cada bucket de la primera tabla de hash contiene solo un puntero a una segunda tabla de hash que almacena los resultados?

(c) (2 puntos) Para el segundo caso, proponga el pseudocódigo de las funciones HashInsert(A,k,j,r) y HashSearch (A,k,j)  $\rightarrow$  r.

Siendo A la tabla de Hash,  $h(r)=k$  el resultado de la función de hash de la primera tabla y  $g(r)=j$  la de la segunda y r el registro conteniendo los resultados de una persona.

# Objetivos de la clase

# Objetivos de la clase

- ☐ Comprender el concepto de función de hash

# Objetivos de la clase

- ☐ Comprender el concepto de función de hash
- ☐ Identificar limitaciones en el almacenamiento a través de arreglos indexados

# Objetivos de la clase

- ☐ Comprender el concepto de función de hash
- ☐ Identificar limitaciones en el almacenamiento a través de arreglos indexados
- ☐ Comprender concepto de tabla de hash



# Objetivos de la clase

- ☐ Comprender el concepto de función de hash
- ☐ Identificar limitaciones en el almacenamiento a través de arreglos indexados
- ☐ Comprender concepto de tabla de hash
- ☐ Comprender concepto de colisión y sus posibles manejos

# Objetivos de la clase

- ☐ Comprender el concepto de función de hash
- ☐ Identificar limitaciones en el almacenamiento a través de arreglos indexados
- ☐ Comprender concepto de tabla de hash
- ☐ Comprender concepto de colisión y sus posibles manejos
- ☐ Distinguir diferencias entre encadenamiento y direccionamiento abierto