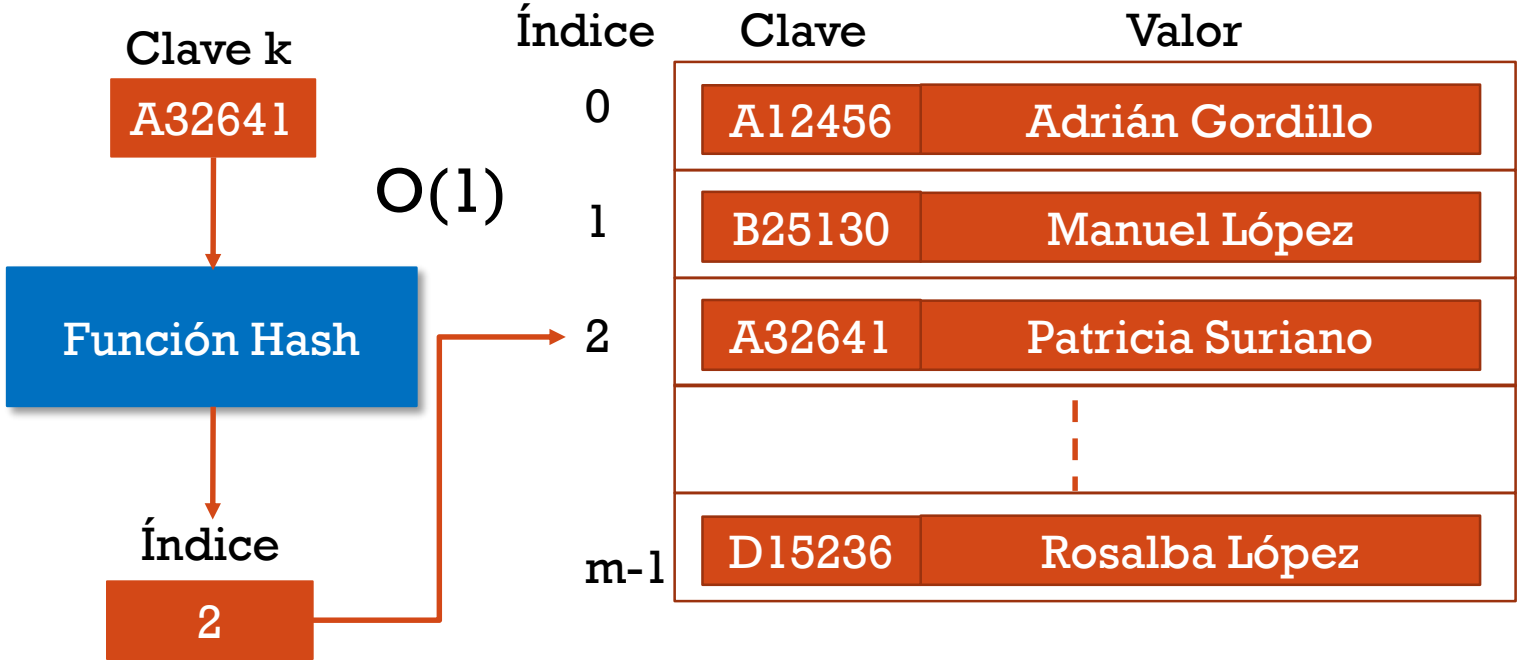
UNIDAD II. ESTRUCTURA DE DATOS AVANZADAS (TABLAS HASH)

Francisco J. Hernández López

fcoj23@cimat.mx

TABLAS HASH

- Contenedores que asocian claves con valores mediante un procedimiento conocido como "hash" o "hashing"



Introduction to Algorithms, Cormen, T.H., Leiserson, Ch. E., Rivest R. L., Stein, C., MIT Press, 2001.

HASHING

- Asociar pares de claves y valores usando arreglos mediante operaciones aritméticas, las cuales transforman las claves en los índices de dicho arreglo

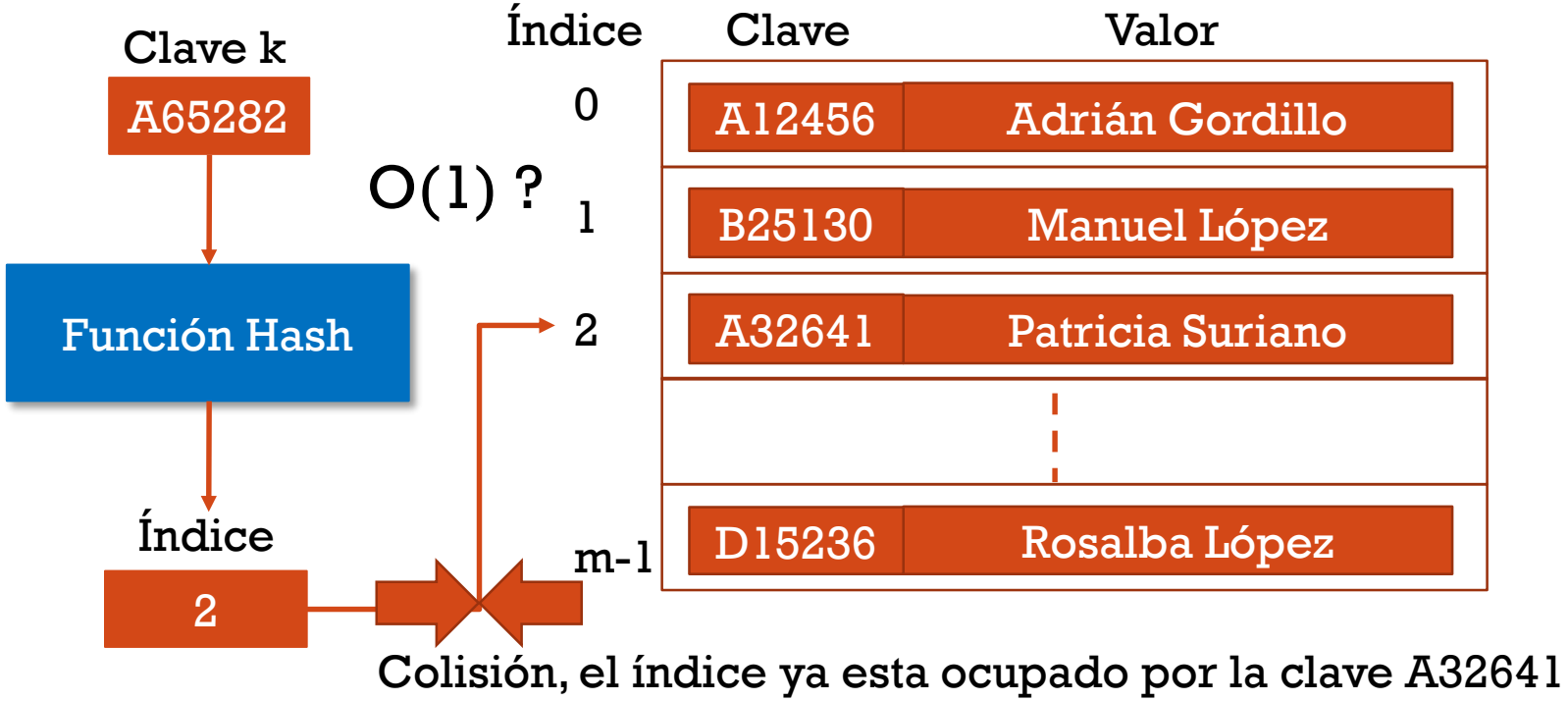
- Si no hay limites de memoria
  - Podríamos utilizar una tabla de grandes dimensiones, de esta forma, utilizaríamos la clave como el índice de la tabla

- Si no hay restricciones de tiempo
  - Podemos usar un arreglo desordenado y realizar las búsquedas de forma secuencial

- Lo que hace el hashing es tener un balance entre la cantidad de memoria a utilizar y las restricciones del tiempo al realizar operaciones en los registros (consultas, inserciones, eliminaciones, etc…)

COLISIONES

- Cuando una función hash mapea dos o más claves en un mismo índice o registro



PARTES DE UN HASHING

- Los algoritmos de búsqueda que usan hashing, consisten en dos partes importantes:

  Función hash → que mapee o transforme la clave en un índice del arreglo o tabla. "Idealmente se quiere que diferentes claves se mapeen en diferentes índices"

  Proceso para resolver colisiones → Como en la practica lo que sucede es que diferentes claves pueden mapearse al mismo índice, entonces se debe usar algún método que resuelva dicho problema, por ejemplo:

  - Encadenamiento separado
  - Sondeo Lineal

FUNCIÓN HASH

- Asumiendo que tenemos un arreglo o tabla T que puede almacenar M pares de claves-valores
- Necesitamos una función hash que pueda transformar una clave dada en un índice (un entero en el rango $[0, M-1]$)

Se busca una función hash que tenga las siguientes características:

- Que sea consistente, claves iguales deben producir el mismo índice
- Que sea fácil de calcular
- Que distribuya las claves de forma uniforme (indep. para cada clave)

La función hash va a depender también del tipo de clave que estemos manejando, entonces necesitamos una función hash diferente para cada tipo de clave

TIPOS DE CLAVE

Enteros positivos

- El método más común es el hashing modular:
  a) Se elige un arreglo de tamaño M (primo)
  b) Para cualquier entero positivo k se calcula el residuo de la división de k entre M:

$$h(k) = (k \,\%\, M)$$

Número en punto flotante

- Si las claves son números reales entre 0 y 1:
  a) Se multiplica por M
  b) Se redondea al entero más cercano para obtener un índice entre 0 y M

TIPOS DE CLAVE

Cadenas

- Cuando las claves son cadenas grandes, estas se pueden trabajar como si fueran enteros grandes
- Ejemplo:

```
int funcion_hash_cadena(char *clave_k){
    int hash=0;
    int R=31;//Numero primo
    for(int i=0;i<strlen(clave_k);i++){
        hash=(R*hash + clave_k[i]) % M;
    }
    return(hash);
}
```

Nota: Modificar R y verificar que si usamos un número primo hay menos colisiones

TIPOS DE CLAVE

Claves compuestas

- Si el tipo de clave tiene múltiples campos enteros, entonces podemos mezclarlos de igual forma que con los caracteres de las cadenas
- Ejemplo:

```
int funcion_hash_compuesta(int clave_dia_k,int clave_mes_k,int clave_anhio_k){
    int hash;
    int R=31;//Numero primo
    hash=(((clave_dia_k*R + clave_mes_k) % M)*R + clave_anhio_k) % M;
    return(hash);
}
```

HASHING CON ENCADENAMIENTO SEPARADO

- Es un método para resolver colisiones en una tabla hash
- La idea es construir para cada índice de la tabla, una lista ligada de todos los pares (clave-valor), cuyos hash den el mismo índice
- Se le llama encadenamiento separado, ya que los hash que colisionan son encadenados por separado en una lista ligada
- Aquí hay que elegir una M lo suficientemente grande para que el tamaño de las listas sea lo más pequeño posible

El orden esperado es: $O(1 + \alpha)$

Con $\alpha = \frac{N}{M}$ → el factor de carga

N→ Número de pares clave-valor

M→ Tamaño de la tabla hash

HASHING CON SONDEO LINEAL

- Otra solución a las colisiones es almacenar N pares (clave-valor) en una tabla hash de tamaño $M > N$
- Se espera que las entradas vacías ayuden a la resolución de las colisiones
- Los métodos que usan esta estrategia son llamados: Métodos hashing de direccionamiento abierto

El más simple de estos es el Sondeo Lineal:
Cuando ∃ una colisión, solo se checa la siguiente entrada (o índice) hasta encontrar una desocupada
Aquí el orden esperado es: $O\left(\frac{1}{1-\alpha}\right)$

CASOS EN UN SONDEO LINEAL

1. La clave que está en la tabla es igual a la clave_k que se intenta buscar

2. Se encuentra una posición vacía: Significa que la clave_k no se encuentra en la tabla

3. La clave no es igual a la clave_k que se intenta buscar, entonces hay que intentar en la siguiente entrada

Eliminar un par (clave-valor) en una tabla hash con sondeo lineal

- Si solamente ponemos NULL en la posición que se obtuvo con la función hash, puede haber un problema
- Recuerde que cuando insertamos elementos a la tabla, y si existen colisiones:
  - Solo buscamos una posición desocupada avanzando de uno en uno

EJEMPLO EN UNA TABLA HASH CON SONDEO LINEAL

Clave	hash()
A	0
B	8
C	3
D	3
E	1
F	3
G	9

Suponemos que en la siguiente tabla hemos almacenado las claves en las posiciones que nos devuelve cierta función hash():

0	1	2	3	4	5	6	7	8	9
A	E		C	D	F			B	G

Note que tanto C, D y F tienen el mismo índice=3

El problema con la eliminación es:

1. Eliminamos a D y solo ponemos un NULL en su respectiva posición
2. Buscar F: Resulta que su hash=3, pero ahí esta la C, entonces al seguir buscando una posición adelante, encontramos un NULL, y daríamos por concluida la búsqueda diciendo que F no está (ERROR)

EJEMPLO EN UNA TABLA HASH CON SONDEO LINEAL

Clave	hash()
A	0
B	8
C	3
D	3
E	1
F	3
G	9

Suponemos que en la siguiente tabla hemos almacenado las claves en las posiciones que nos devuelve cierta función hash():

0	1	2	3	4	5	6	7	8	9
A	E		C	F				B	G

Note que tanto C, D y F tienen el mismo índice=3

Una solución a dicho problema es:

1. Eliminamos a D y ahora recorremos las claves (cuyo hash coincidan con el hash de la clave a eliminar) una posición antes
2. Buscar F: Resulta que su hash=3, pero ahí esta la C, entonces al seguir buscando una posición adelante, ahora si encontramos F