

# Tablas Hash

...

Algoritmos y Estructuras de Datos  
2023

# Clase *dict*

- Se basa en el almacenamiento de pares de datos relacionados, denominados **clave** y **valor**.
- Cada clave es un valor **único**, que tiene asociado un valor (los valores pueden repetirse).



# Clase *dict*

- Para definir un objeto de la clase *dict* se utilizan llaves.
- Para acceder a valores dentro de un diccionario se utiliza una sintaxis similar a la de las **listas**.
- Las claves en un diccionario pueden ser **cualquier tipo de datos inmutable**, la única restricción es que cada clave debe ser un valor único.

```
monedas = {}  
monedas["Argentina"] = "Peso"  
monedas["Japon"] = "Yen"  
print(monedas["Argentina"])  
for m in monedas:  
    print(monedas[m])
```

## Operaciones de la clase *list*

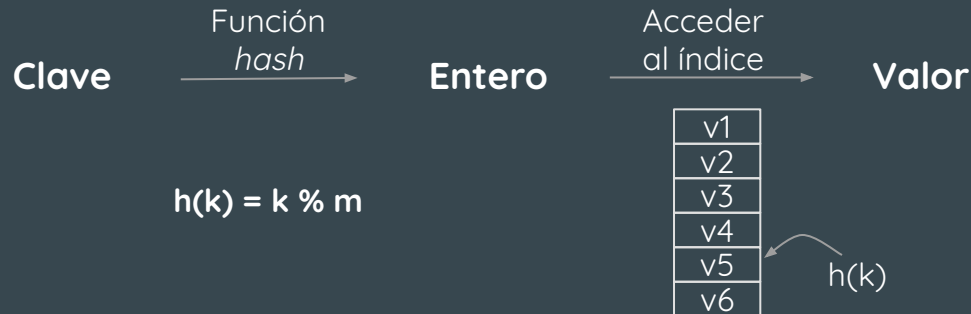
<code>M[k] = v</code>	Asigna a la llave “k” el valor “v”.
<code>M[k]</code>	Devuelve el valor asociado a la llave “k”.
<code>del M[k]</code>	Elimina el par clave-valor.
<code>len(M)</code>	Devuelve el número de pares clave-valor.
<code>k in M</code>	Devuelve True si la llave “k” está presente en “M”.
<code>M.keys()</code>	Devuelve todas las claves dentro del diccionario.
<code>M.values()</code>	Devuelve todos los valores dentro del diccionario.
<code>M.items()</code>	Devuelve tuplas con todos los pares clave-valor.

# Ventajas de utilizar diccionarios

- Los valores dentro de un diccionario pueden ser de cualquier tipo, incluidos otros datos abstractos (como diccionarios).
- Al igual que acceder a un elemento en una lista usando un índice, el acceso a un elemento usando su clave se da, en promedio, en **tiempo constante**.
- Esto es posible gracias al uso de la función *hash*.

# Función *hash*

- El objetivo de esta función es mapear cada clave con un entero.
- Si conceptualizamos a los pares clave-valor de un diccionario ubicados dentro de una lista, el objetivo de la **función *hash*** calculada sobre la **clave** es obtener **índice** para el **valor** correspondiente.
- Sin importar el tipo de dato, si logramos aplicar la función *hash* podremos acceder a un elemento en tiempo constante.



## Función *hash*

- La selección de la función *hash* es un paso clave a la hora de diseñar una clase como *dict*.
- Es necesario que esta función pueda asignarle un entero a **cualquier clave que se quiera ingresar**.
- A priori, la cantidad de claves que se pueden usar es infinita.
- La lista donde se almacenan los valores no puede ser de tamaño **infinito** para almacenar todos los posibles índices de todas las claves posibles.

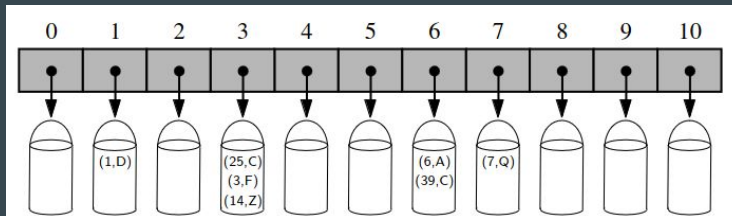
# Colisiones

- Al limitar el tamaño de la lista de valores, se limita el número de índices que puede devolver la función *hash*.
- El mayor problema surge cuando la función devuelve el mismo resultado para 2 claves distintas.
- Cuando estas **colisiones** ocurren, es necesario aplicar una solución que permita seguir utilizando la misma función *hash* y almacenar los elementos conflictivos.



# Colisiones

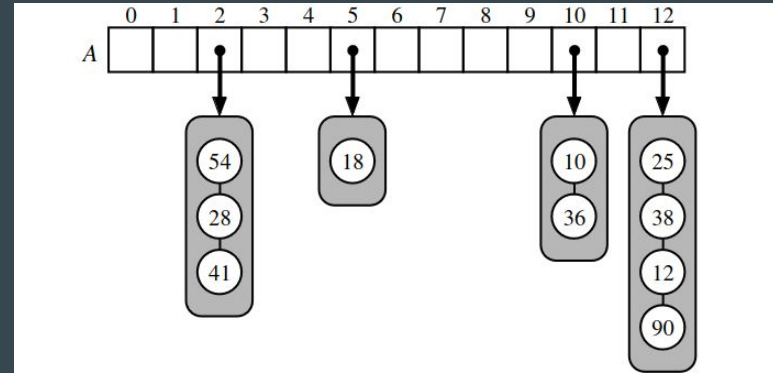
- Supongamos que tenemos los siguientes pares de datos:
  - 1, D
  - 3, F
  - 6, A
  - 7, Q
  - 14, Z
  - 25, C
  - 39, C
- Y usando el modulo (%) como función *hash* para armar una lista de 11 elementos se obtiene lo siguiente:
  - $h(1) = 1$
  - $h(3) = 3$
  - $h(6) = 6$
  - $h(7) = 7$
  - $h(14) = 3$
  - $h(25) = 3$
  - $h(39) = 6$



**Figure 10.4:** A bucket array of capacity 11 with items (1,D), (25,C), (3,F), (14,Z), (6,A), (39,C), and (7,Q), using a simple hash function.

# Soluciones a las colisiones

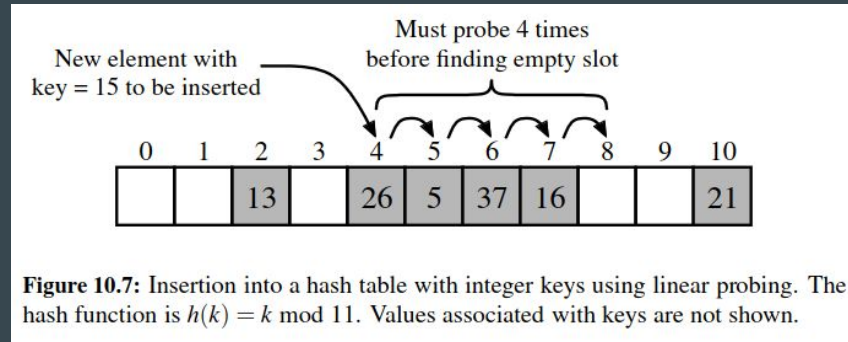
- La solución más sencilla sería almacenar en el índice de la lista donde se produce la colisión una lista, con valores  $(k, v)$
- Si al calcular la función *hash* el valor dentro de la lista de valores es esta segunda lista, se busca la clave deseada dentro de estos valores.
- Esta estrategia se conoce como ***separate chaining*** o **encadenamiento separado**.



**Figure 10.6:** A hash table of size 13, storing 10 items with integer keys, with collisions resolved by separate chaining. The compression function is  $h(k) = k \bmod 13$ . For simplicity, we do not show the values associated with the keys.

# Soluciones a las colisiones

- La mayor desventaja del encadenamiento es la utilización de espacio extra para las listas y de tiempo extra para su recorrido.
- Una serie de estrategias alternativas se denomina **direccionamiento abierto**.
- El ejemplo más simple es el **sondeo lineal**, donde se ocupa el siguiente elemento de la lista, si el correspondiente está ocupado



# Soluciones a las colisiones

- En el sondeo lineal, el espacio ocupado por un valor se puede calcular como  $h(k) + i$ , siendo  $i = 0, 1, 2, 3, \dots$
- Otras alternativas para solucionar colisiones son
  - El **sondeo cuadrático**, similar al sondeo lineal, pero  $i$  toma los valores  $1^2, 2^2, 3^2, 4^2$ .
  - El **doble hasheo**, donde se diseña una segunda función *hash* para asignar un nuevo índice dentro de la lista de claves-valores.
- Todas estas funciones intentan solucionar el problema de las colisiones, pero aumentan la complejidad de la búsqueda dentro de un diccionario.

# Diccionarios en Python

- Python tiene incorporada la función *hash*, que permite calcular un valor entero a partir de un elemento inmutable.
- Esta función devuelve **números flotantes** (incluidos negativos), por lo que la clase *dict* incluye una segunda función, denominada de compresión, para transformar ese flotante en entero positivo.
- Para definir un diccionario, se utilizan las llaves ({ y }), y los pares clave-valor se separan con dos puntos (:)

```
monedas = {"Argentina": "Peso",  
           "Japon":      "Yen",  
           "Italia":     "Euro" }
```

# Ejemplo 1

- Diseñe un diccionario que almacene la cantidad de feriados nacionales en cada mes del 2023. Encuentre cuál es el mes con más feriados.

```
meses = {"Enero": 1,  
         "Febrero": 2,  
         "Marzo": 1,  
         ...  
         "Diciembre": 2,  
         }
```

```
mayor = 0  
mes = ""  
for m in meses:  
    if meses[m] > mayor:  
        mayor = meses[m]  
        mes = m
```

## Ejemplo 2

- El archivo *muestras.txt* contiene la siguiente información:

Identificador	Muestra	Fecha	Edad	Continente	País	Provincia	Ciudad	Sexo
PAIS-E0810	4806	2022-01-24		South America	Argentina	Santa Fe	Santa Fe	F
PAIS-E0811	4832	2022-01-24		South America	Argentina	Santa Fe	Pilar	F
PAIS-E0812	4837	2022-01-24		South America	Argentina	Santa Fe	Pilar	M
PAIS-E0813	5945	2021-12-30	21	South America	Argentina	Misiones	Andresito	F
PAIS-E0815	6817	2021-12-31	54	South America	Argentina	Misiones	Posadas	F

- Cargue todos los datos en un diccionario. Busque
  - Todas las muestras que sean de Misiones
  - Las muestras que vengan de pacientes con más de 40 años.

## Ejemplo 2

```
muestras = {}  
with open ("muestras.txt") as ifh:  
    l = ifh.readline()  
    l = l.rstrip()  
    encabezado = l.split("\t")  
    for l in ifh:  
        l = l.rstrip()  
        linea = l.split("\t")  
        muestras[linea[0]] = {}  
        for i in range(1, len(linea)):  
            muestras[linea[0]][encabezado[i]] = linea[i]  
  
for m in muestras:  
    if muestras[m]["Provincia"] == "Misiones":  
        print(m)  
  
for m in muestras:  
    if muestras[m]["Edad"] != "" and int(muestras[m]["Edad"]) > 40:  
        print(m)
```



# Bibliografía

- Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2013). Data structures and algorithms in Python. Capítulo 10.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to algorithms. MIT press. Capítulo 11.