Grado en Ingeniería Información

Estructura de Datos y Algoritmos

Sesión 12

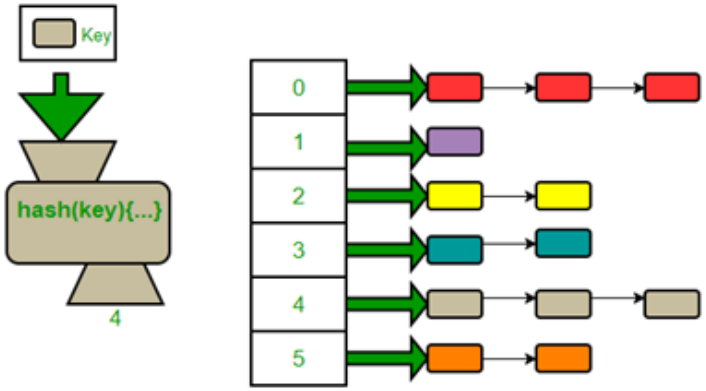
Curso 2022-2023

Marta N. Gómez

UNIVERSIDAD NEBRIJA

T3. Tipos Abstractos de Datos (TAD)

- **Tablas.**
  - Introducción
  - Tablas Lineales
  - Tablas Arborescentes
  - Tablas Hash

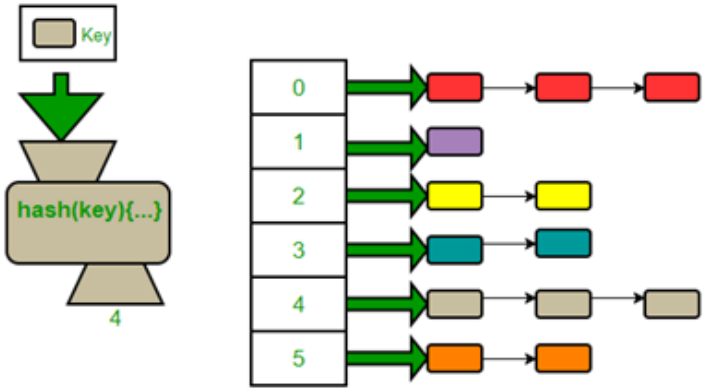


UNIVERSIDAD NEBRIJA

ÍNDICE

T3. Tipos Abstractos de Datos (TAD)

- **Tablas.**
  - **Introducción**
  - Tablas Lineales
  - Tablas Arborescentes
  - Tablas Hash



UNIVERSIDAD NEBRIJA

ÍNDICE

Una tabla es una **colección de elementos**, cada uno de los cuales tiene una **clave** y una **información** asociada.

Ejemplos:

Guía telefónica,

Diccionario,

Tablas de equivalencias (euros-pesetas, talla-peso, etc.).

Clave 1	**Información 1**
Clave 2	**Información 2**
. . .	. . .

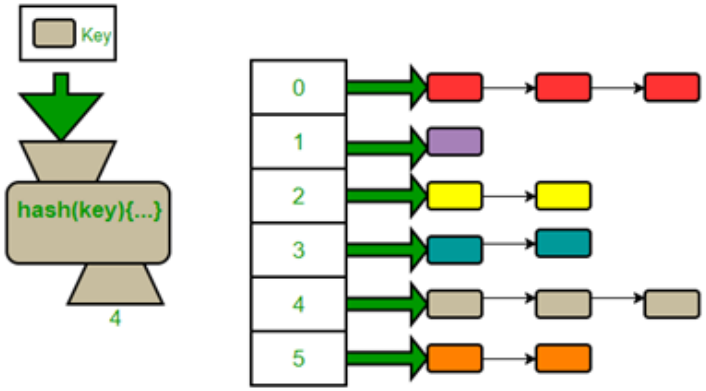
Elementos

Las **operaciones** más usuales son:

- ***esTablaVacia***: decide si una tabla tiene o no elementos.
- ***estaLaClave***: decide si una clave está en una tabla o no.
- ***recupInforTabla***: recuperar de la tabla la información asociada a una clave.
- ***insertarElemTabla***: incluye un nuevo elemento (clave-información) en la tabla.
- ***borrarElemTabla***: elimina un elemento (clave-información) con una determinada clave de la tabla.

T3. Tipos Abstractos de Datos (TAD)

- **Tablas.**
  - Introducción
  - **Tablas Lineales**
  - Tablas Arborescentes
  - Tablas Hash



UNIVERSIDAD NEBRIJA

ÍNDICE

Tablas Lineales

Una ***tabla lineal*** puede considerarse como una secuencia de elementos, ordenados o desordenados, y su realización puede hacerse de alguna de las siguientes formas:

1. Tablas vectoriales: donde una tabla es un **vector** en el que cada posición contiene un elemento de la tabla (clave-información).

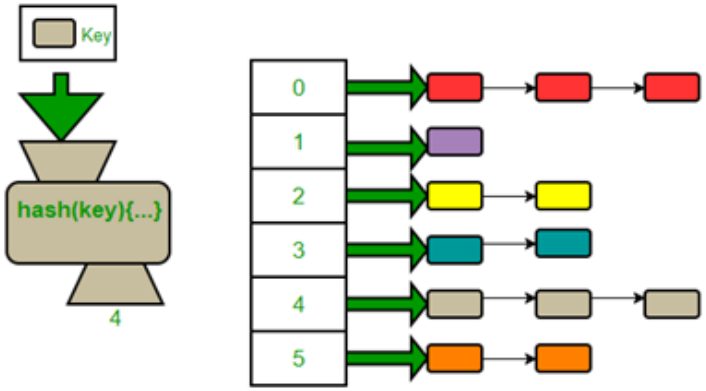
Problema: Limitación de espacio en las ***tablas vectoriales.***

2. Tablas encadenadas: donde la tabla es una **lista enlazada**.

Problema: Complejidad de proceso de búsqueda en las ***tablas encadenadas.***

T3. Tipos Abstractos de Datos (TAD)

- **Tablas.**
  - Introducción
  - Tablas Lineales
  - **Tablas Arborescentes**
  - Tablas Hash



UNIVERSIDAD NEBRIJA

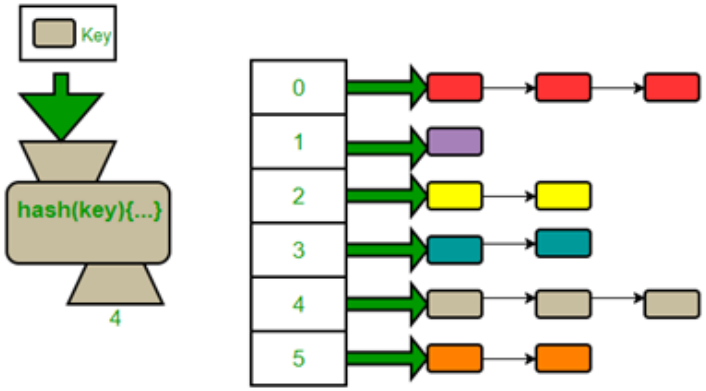
ÍNDICE

SOLUCIÓN: *Árbol Binario de Búsqueda*

- Todas las claves del ***subárbol izquierdo*** son ***menores*** que la **clave del nodo** o **elemento raíz**.
- Todas las claves del ***subárbol derecho*** son ***mayores*** que la **clave del nodo** o **elemento raíz**.
- Los ***subárboles izquierdo*** y ***derecho*** también son **Árboles Binarios de Búsqueda**.

T3. Tipos Abstractos de Datos (TAD)

- **Tablas.**
  - Introducción
  - Tablas Lineales
  - Tablas Arborescentes
  - **Tablas Hash**



UNIVERSIDAD NEBRIJA

ÍNDICE

DEFINICIÓN:

Tablas que permiten ***localizar unívocamente*** la **posición** de un elemento en la tabla en **función de su clave**.

FUNCIÓN HASH:

Utilizando una función que transforma una **clave** de un elemento en una **posición** determinada y concreta dentro del vector. **Difícil de seleccionar**.

función *hash(clave)* → *índice de la tabla*

El **objetivo** es **reducir el coste** de las operaciones y para ello, los potenciales elementos de la colección se tienen que distribuir en un número finitos de ***clases*** (**c**)

Por tanto, habrá que buscar una **función** (función de ***dispersión*** o **hash**) para establecer la correspondencia de cada elemento con alguna de las clases establecidas (valor dentro del intervalo [***0...c−1***].

La función de ***dispersión*** o función **hash** (***f***) tiene que:

- Ser **fácil** de calcular
- Distribuir de forma uniforme los ***n*** **elementos** de la colección entre las ***c*** **clases**. Así, las **clases** tendrán aproximadamente el mismo número de elementos (***n/c***).

EL PROBLEMA DE LAS *COLISIONES*

- ***Nº claves posibles > Nº índices de la tabla***

  Por tanto, habrá **varias claves** que tendrán un **mismo índice** de la tabla.
- La **función hash** devuelve ***un mismo valor para distintas claves***.

OBJETIVO:

Localizar una función *hash* que **minimice** el número de colisiones y **diseñar métodos de resolución de colisiones** cuando éstas se produzcan.

Direccionamiento encadenado: Mantiene en una **lista enlazada** todos los ***sinónimos***, es decir, aquellos elementos a los que les corresponde la misma posición en la tabla después de la transformación de sus claves (aplicar la función hash).

Direccionamiento abierto: Calcula una **secuencia de posiciones** en la tabla hash hasta que encuentra una vacía para ubicar el elemento que colisiona.

EJEMPLO: **se supondrá el valor ClaveNula = 0**

y la función hash: hash(k) = k % M = k % 12

donde M es el nº de posiciones de la tabla hash

j	k_j	h1(k_j)	j	k_j	h1(k_j)
1	154	10	7	78	6
2	92	8	8	158	2
3	81	9	9	56	8
4	110	2	10	119	11
5	164	8	11	161	5
6	280	4	12	24	0

Primera forma:

- La tabla se crea de forma que **cada posición** contenga **un elemento y una lista enlazada** con todos los elementos que colisionen (**sinónimos**) en esa posición/dirección.
- La detección de **posiciones libres** en la tabla se hace a través de una **clave nula** (valor que hay que definir).
- El **tiempo necesario para localizar una clave** dependerá del **tamaño de las listas** y de las posiciones relativas de las claves en ellas, es decir, su **ordenación**.
- Las **listas individuales** no han de tener un **tamaño excesivo**, y se suele optar por la alternativa más simple, la **LIFO**.

Segunda forma:

Connsiderar la tabla como un **vector de listas enlazadas**.

Así, cada posición del vector solo tiene un campo que, tendrá el valor de lista vacía (***nulptr***), o bien enlazará con la **lista de claves sinónimas**.

Ventajas importantes de esta forma:

1. Se evita la necesidad de la una ***clave nula***.
2. Las operaciones de manejo de la tabla hash se realizan totalmente en base a las **operaciones básicas del TAD Lista**.

```
#define MaxElem  . . .     //  Número de entradas en la tabla.
#define ClaveNula  . . .   //  Valor de la clave nula.
. . .  Tinfor;       //  Tipo de dato para la información de cada elemento.
. . .  Tclave;     //  Tipo de dato para la clave de cada elemento.
class CDato {
      Tinfor infor;
      Tclave clave;
      //  Métodos públicos CDato
};
class Nodo {
      CDato elem;
      shared_ptr<Nodo> sig = nullptr;
      //  Métodos públicos Nodo
};
```

```
class Lista {
        shared_ptr<Nodo> first;
       //  Métodos públicos Lista
};
class THash {
        array<Nodo, MaxElem> tabla;
    public:
        Thash()
        bool esTablaVacia () const;
        bool estaLaClave (Tclave const &k) const;
        Tinfor recuperarInforTabla (Tclave const &k) const;
        void insertarElemTabla (CDato const &elem);
        void borrarElemTabla (Tclave const &k);
};
```

Consiste en utilizar **un vector** donde **cada clave** debe ocupar **una posición** en el mismo.

Cuando se produce **una colisión** se busca dentro de la tabla hasta que **se encuentra dicha clave** o bien, hasta que **se ubica dicha clave en una posición libre** de la tabla.

SOLUCIÓN del PROBLEMA

Rehashing o Reasignación: Una vez producida una **colisión** al insertar un elemento se utiliza una **función adicional** para determinar cual será la casilla que le corresponde dentro de la tabla, a esta función la llamaremos **función de reasignación, $reh_i(k)$.**

La **forma más fácil** de realizar una función de reasignación es tratar **las posiciones de la tabla de forma secuencial**.

Si dicha posición vuelve a estar **ocupada**, partiendo de ella, se **recalcula la siguiente** y así sucesivamente. De esta forma la función quedaría de la siguiente forma:

$$reh_i(k) = (h(k) \% M) + i \qquad i = 1, 2, 3\ldots$$

k es la clave

M es el número de posiciones de la tabla

h() es la función hash

Tablas Hash – Direccionamiento Abierto

```
#define MaxElem  . . .          //  Número de entradas en la tabla.
#define ClaveNula  . . .        //  Valor de la clave nula.
#define ClaveBorrada  . . .     //  Valor de la clave borrada de la tabla
. . .  Tinfor;                  //  Tipo de dato para la información de cada elemento.
. . .  Tclave;                  //  Tipo de dato para la clave de cada elemento.
class CDato {
        Tinfor infor;
        Tclave clave;
        //  Métodos públicos Cdatos     };
class THash {
        array<CDato, MaxElem> tabla;
    public:
        Thash();
        bool esTablaVacia () const;
        bool estaLaClave (Tclave const &k) const;
        Tinfor recupInformTabla (Tclave const &k) const;
        void insertarElemTabla (CDato const &elem);
        void borrarElemTabla (Tclave const &k);
};
```

Tablas Hash – Direccionamiento Abierto

Orden	Elemento		h1(kj)	rehj(kj)
	kj	Información		
1	154	I1	**10**	
2	92	I2	**8**	
3	81	I3	**9**	
4	110	I4	**2**	
5	164	I5	**8**	**9 - 10 - 11**
6	280	I6	**4**	
7	78	I7	**6**	
8	158	I8	**2**	**3**
9	56	I9	**8**	**9 - 10 - 11 - 0**
10	119	I10	**11**	**0-1**
11	161	I11	**5**	
12	24	I12	**0**	**1 - 2 - 3 - 4 - 5 - 6 – 7**

Cuando intentamos **borrar un elemento** con clave *k* de una tabla gestionada por **direccionamiento abierto** puede existir un problema:

Si dicho elemento de clave *k* se **incorporó antes** que otros elementos de **clave sinónima**, **no se puede eliminar directamente**, ya que si se hiciera, los **elementos de claves sinónimas incluidos posteriormente a este quedarían aislados** y posteriormente al intentar borrar otro elemento de clave sinónima a *k* podríamos obtener que dicho elemento no existe en la tabla, pudiendo ser falso.

Para evitarlo se marca con un valor que lo señale como **clave borrada.**