Sesión 6 - 30 de septiembre. Tabla hash para acceso rápido

Estructuras de Datos.

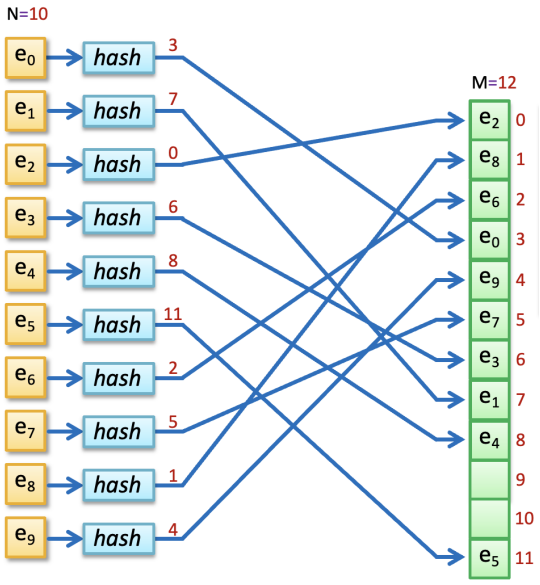
Dpto. Lenguajes y Ciencias de la Computación. Universidad de Málaga

En esta práctica se va a implemenar una estructura que nos permita acceder rápidamente a datos de personas según su identificación única en una empresa. Para ello, vamos a hacer uso de una tabla hash.

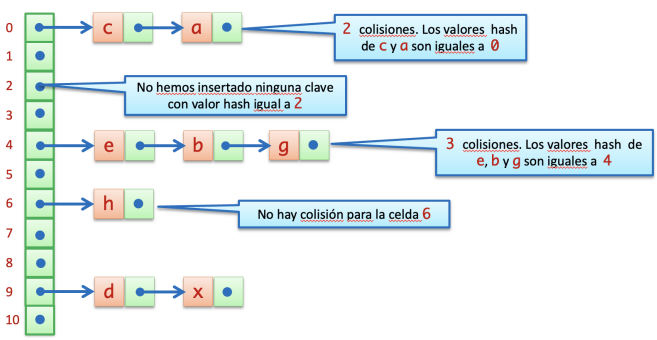
Para crear una tabla hash necesitamos una función hash:

$$hash :: Key \rightarrow \{0..M-1\}$$

Esta función, dado un valor o conjunto de valores, devuelve su clave asociada. En la siguiente imagen se observa como al aplicar la función hash a diferentes objetos (e_0 ... e_9) se obtienen sus claves (3, 7, 0, ..., 4). Estos valores se pueden usar para insertar en posiciones de un array o tabla, haciendo que su búsqueda e insercción sea, en el mejor de los casos O(1).



¿Qué ocurre si dos objetos a insertar tienen el mismo hash? A esto se le denomina colisión. En el siguiente ejemplo se muestra como *{c, a}*,*{e, b, g}* o *{d,x}* colisionan.



Si colisionaran todos (una muy mala función hash), tendriamos a todos los objetos insertados en la misma lista y la complejidad sería O(n).

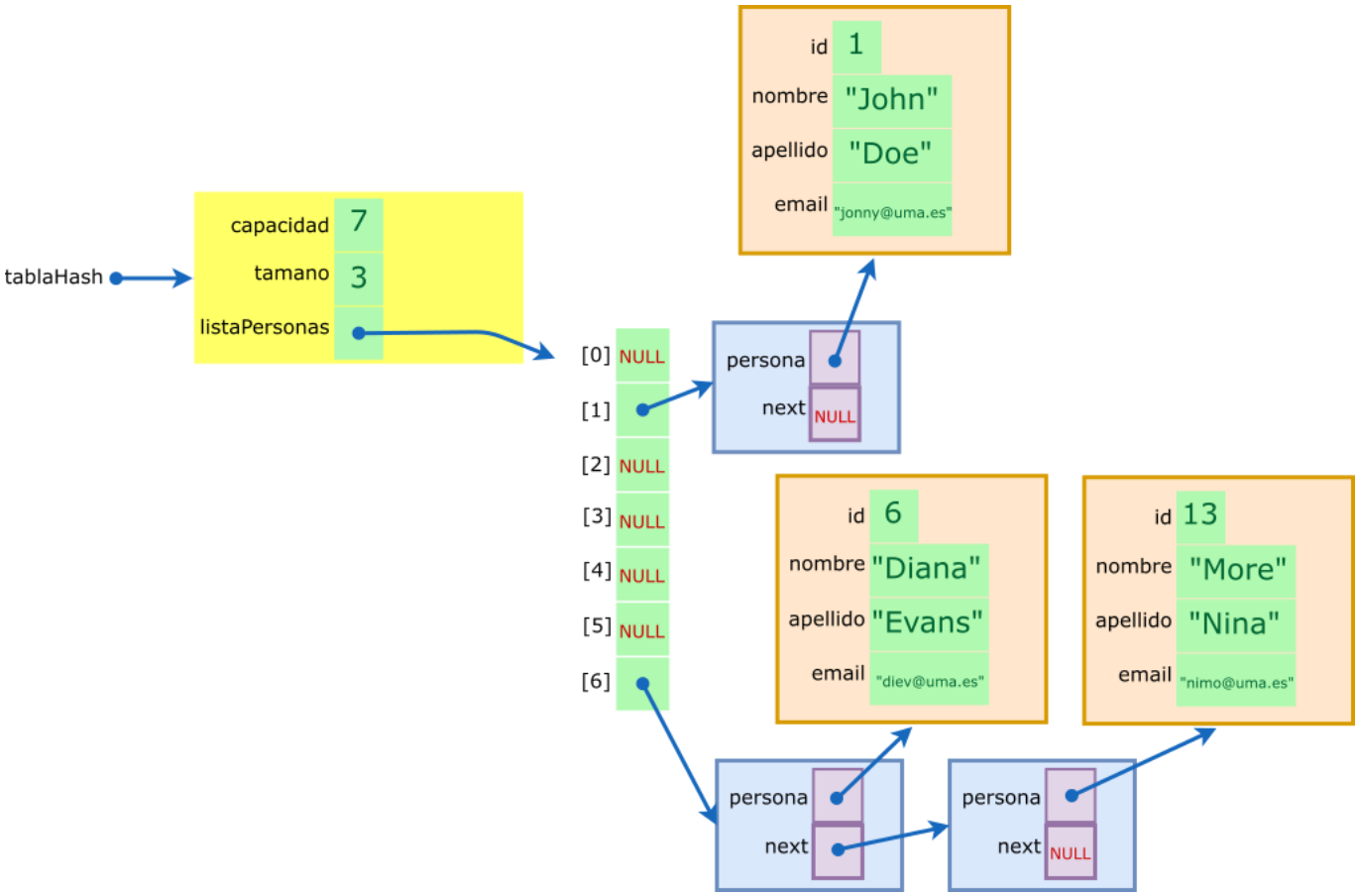
En este ejercicio vamos a resolver las colisiones con una lista enlazada simple. Esta la puedes o no ordenar, mejorando la búsqueda en media a O(n/2) si la ordenas.

Visualizando nuestra tabla Hash

Nuestra implementación de una tabla hash tendrá una estructura primera que contendrá el array de listas de personas, la capacidad (C) de dicho array y el número de personas que hay en total insertadas.

Como función hash vamos a usar el módulo capacidad del id: $id%C$. Por tanto, si la capacidad es 7 y nos entra un id de 13, tendremos que el hash es 6 (13%7=6).

En la siguiente imagen se observa un ejemplo con esta estructura. Esta tiene una capacidad de 7, y por tanto, el array *listaPersonas* tiene 7 elementos. Además, se puede observar como tiene tamaño 3. Teniendo dos personas insertadas en la posición 6 (ids 6 y 13) y una en la posición 1.



Implementa esta estructura y pruébala

Se han separado para aportar más modularidad y aumentar la cohesión en dos librerías la gestión de acceso:

- persona.h: librería encargada de la gestión de personas (creación, liberación y mostrar).
- hash.h: librería para gestionar la tabla hash (creación, liberación, insercción, borrado y búsqueda).

Se facilita un main.c con algunas pruebas, comenta las que no tengas implementadas. Recuerda que deberías probar:

- Insercción en tabla hash completa.
- Gestión de colisiones.
- Búsqueda de elementos que han tenido y que no han tenido colisiones.
- Búsqueda de elementos que no existen.
- Borrado de elementos.
- Liberación de una tabla hash completa.

Cabecera persona.h

```
//...

struct Persona{
    int id;
    char nombre[50];
    char apellido[50];
    char email[50];
};

 /**
    * @brief Creates a new Persona instance.
    *
    * This function allocates memory for a new Persona structure and
initializes it with the provided id, name, surname, and email.
    *
    * @param id The unique identifier for the person.
    * @param nombre The first name of the person.
    * @param apellido The surname of the person.
    * @param email The email address of the person.
    * @return A pointer to the newly created Persona structure.
    */
struct Persona * crearPersona(int id, char *nombre, char *apellido, char
*email);

 /**
    * @brief Displays the information of a Persona.
    *
    * This function prints the details of the given Persona structure to
the standard output.
    *
    * @param persona A pointer to the Persona structure to be displayed.
    */
void mostrarPersona(struct Persona *persona);

/**
 * @brief Frees the memory allocated for a Persona structure.
 *
 * This function takes a pointer to a pointer to a Persona structure and
deallocates the memory associated with it. After calling this function,
the pointer to the Persona structure will be set to NULL.
 *
 * @param persona A double pointer to the Persona structure to be freed.
 */
```

```
void liberarPersona(struct Persona **persona);
```

Cabecera hash.h

```
struct Node{
    struct Persona *persona;
    struct Node *next;
};

struct Hash{
    int capacidad;
    int tamano;
    struct Node **listaPersonas;
};

    /**
     * @brief Creates a new hash table with the specified capacity.
     *
     * This function allocates memory for a hash table and initializes it
with the given capacity. The hash table can be used to store and retrieve
key-value pairs efficiently.
     *
     * @param capacidad The capacity of the hash table, i.e., the number
of
     *                  buckets it will contain.
     * @return A pointer to the newly created hash table.
     */
    struct Hash * crearHashTable(int capacidad);

    /**
     * @brief Computes the hash value for a given ID.
     *
     * This function takes an integer ID and computes its hash value based
on the provided capacity. The hash value is used to determine the index in
a hash table where the ID should be stored.
     *
     * @param id The integer ID to be hashed.
     * @param capacidad The capacity of the hash table.
     * @return The computed hash value for the given ID.
     */
    int hashFunc(int id, int capacidad);

    /**
     * @brief Inserts a Persona into the given Hash table.
     *
     * This function takes a pointer to a Hash table and a pointer to a
Persona and inserts the Persona into the Hash table.
     *
     * @param hash Pointer to the Hash table where the Persona will be
```

```
inserted.
     * @param persona Pointer to the Persona to be inserted into the Hash
table.
     */
    void insertar(struct Hash *hash, struct Persona *persona);

    /**
     * @brief Searches for a person in the hash table by their ID.
     *
     * This function takes a pointer to a hash table and an ID, and
searches for a person with the given ID in the hash table. If the person
is found, a pointer to the corresponding `Persona` structure is returned.
If the person is not found, the function returns `nullptr`.
     *
     * @param hash A pointer to the hash table.
     * @param id The ID of the person to search for.
     * @return A pointer to the `Persona` structure if found, otherwise
`nullptr`.
     */
    struct Persona * buscar(struct Hash *hash, int id);

    /**
     * @brief Elimina una entrada del hash dado su identificador.
     *
     * @param hash Puntero a la estructura Hash de la cual se eliminará la
entrada.
     * @param id Identificador de la entrada que se desea eliminar.
     */
    void eliminar(struct Hash *hash, int id);

    /**
     * @brief Displays the contents of the given hash table.
     *
     * This function takes a pointer to a Hash structure and prints its
contents to the standard output. The exact format of the output depends on
the implementation details of the Hash structure.
     *
     * @param hash A pointer to the Hash structure to be displayed.
     */
    void mostrar(const struct Hash *hash);



    /**
     * @brief Frees the memory allocated for the hash table.
     *
     * This function releases all the resources associated with the given
hash table, including any dynamically allocated memory for the table
itself and its contents.
     *
     * @param hash A pointer to the reference to the Hash structure to be
freed.
     */
    void liberarHash(struct Hash **hash);
```