



**INSTITUTO POLITÉNICO NACIONAL**

**ESCUELA SUPERIOR DE CÓMPUTO**

**ESTRUCTURAS DE DATOS**

**PRÁCTICA #3**

**DICCIONARIO CON HASHING ABIERTO**

**GRUPO: 1CM8**

**EQUIPO: LAS MÁS PERRAS**

**INTEGRANTES:**

* JIMÉNEZ DELGADO LUIS DIEGO 2019630461
* SÁNCHEZ CASTRO AARÓN GAMALIEL 2019630079
* SÁNCHEZ TIRADO CITLALI YASMÍN 2019630096

**PROFESOR:** EDGARDO ADRIÁN FRANCO MARTÍNEZ

**FECHA DE ENTREGA:** 22 DE ABRIL 2019

**INTRODUCCIÓN**

A la hora de realizar una búsqueda nos preguntamos si es posible realizarla de una forma más rápida y eficiente, ya que al querer realizar esta clase de búsquedas se nos podría ocurrir simplemente hacerlo por comparaciones entre valores pero aquí es donde entran nuestras funciones hash que nos permiten la localización de una clave en una tabla, el objetivo de estas funciones es evitar valores duplicados.

**MARCO TEÓRICO**

**¿Qué es una tabla Hash?**

Una tabla hash, matriz asociativa, mapa hash, tabla de dispersión o tabla fragmentada es una estructura de datos que asocia llaves o claves con valores.

La operación principal que soporta de manera eficiente es la búsqueda: permite el acceso a los elementos (teléfono y dirección, por ejemplo) almacenados a partir de una clave generada (usando el nombre o número de cuenta, por ejemplo). Funciona transformando la clave con una función hash en un hash, un número que identifica la posición (casilla o cubeta) donde la tabla hash localiza el valor deseado.

Un ejemplo práctico para ilustrar que es una tabla hash es el siguiente: Se necesita organizar los periódicos que llegan diariamente de tal forma que se puedan ubicar de forma rápida, entonces se hace de la siguiente forma – se hace una gran caja para guardar todos los periódicos (una tabla), y se divide en 31 contenedores (ahora es una “hash table” o tabla fragmentada), y la clave para guardar los periódicos es el día de publicación (índice). Cuando se requiere buscar un periódico se busca por el día que fue publicado y así se sabe en que zócalo (bucket) está. Varios periódicos quedarán guardados en el mismo zócalo (es decir colisionan al ser almacenados), lo que implica buscar en la sub-lista que se guarda en cada zócalo. De esta forma se reduce el tamaño de las búsquedas de O(n) a O(log(n)).

Las operaciones básicas implementadas en las tablas hash son:

inserción(llave, valor)

búsqueda(llave) que devuelve valor

La mayoría de las implementaciones también incluyen borrar(llave). También se pueden ofrecer funciones como iteración en la tabla, crecimiento y vaciado. Algunas tablas hash permiten almacenar múltiples valores bajo la misma clave.

Para usar una tabla hash se necesita:

Una estructura de acceso directo (normalmente un array).

Una estructura de datos con una clave

Una función resumen (hash) cuyo dominio sea el espacio de claves y su imagen (o rango) los números naturales.

Existen dos tipos de tablas hash, en función de cómo resuelven las colisiones:

* Encadenamiento separado: Las coliciones se resuelven insertándolas en una lista. De esa forma tendríamos como estructura un vector de listas. Al número medio de claves por lista se le llama factor de carga y habría que intentar que esté próximo a 1.
* Direccionamiento abierto: Utilizamos un vector como representación y cuando se produzca una colisión la resolvemos reasignándole otro valor hash a la clave hasta que encontremos un hueco.

KEY

|  |  |
| --- | --- |
| 0 |  |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| ... |  |

VALOR HASH

**FUNCIÓN**

**HASH**

VENTAJAS:

* Nos permiten guardar información de un mismo tipo de dato
* La guardan de manera contigua (osea junta, una al lado de la otra) en memoria.
* Gracias a lo anterior podemos acceder a cualquier posición que queramos mediante un simple paso, en un tiempo constante sin importar si el el primero o el décimo elemento.

DESVENTAJAS:

* No es dinámico, es decir debemos especificar su tamaño cuando lo declaramos
* Por lo tanto, es muy difícil y tardado añadir o quitar elementos (es más ni siquiera es seguro que haya más espacio libre contiguo en memoria para añadir más elementos)

Colisiones

Una colisión en una tabla hash la podemos definir: En el momento en que dos entradas generan una misma salida.

Por ejemplo si quisiéramos ordenar palabras basadas en su primera letra:

**B**OLICHE

**A**RDILLA

|  |  |
| --- | --- |
| 0 | ARDILLA |
| 1 | **B**OLICHE |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| ... |  |

|  |  |
| --- | --- |
| 0 | **A**RDILLA |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| ... |  |

**FUNCIÓN**

**HASH**

**FUNCIÓN**

**HASH**

|  |  |
| --- | --- |
| 0 | ARDILLA |
| 1 | BOLICHE |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| ... |  |

**PLANTEAMIENTO DEL PROBLEMA**

**FUNCIÓN**

**HASH**

**A**MIGO

Con la implementación del TAD lista realizar la implementación de una tabla hash abierta, capaz de soportar el almacenamiento de palabras y sus definiciones (Diccionario de palabras). La función hash a usar deberá ser analizada por cada equipo y deberá de justificarse (determinar al menos dos y reportarlas).

**DISEÑO Y FUNCIONAMIENTO DE LA SOLUCIÓN**

Se propone la creación de un TAD Tabla Hash (*TADHashTable*) utilizando el TAD Lista Doblemente Ligada, de modo que un usuario de la librería *TADHashTable*no tiene conocimiento explícito que esta librería utiliza la lista doblemente ligada.

AGREGAR DIBUJO

El *TADHashTable*cuenta con las siguientes operaciones.

* InicializarTabla:
* DestruirTabla:
* AgregarATabla:
* EliminarDeTabla:
* elemento BuscarEnTabla:
* ModificarTabla:
* Estadisticas:
* AplicarHash

**IMPLEMENTACIÓN**

/\*

FUNCIÓN: caracterLatino(int c)

RECIBE: int c (VALOR ASCII DE UN CARACTER)

DEVUELVE: int

DESCRIPCIÓN: ESTA FUNCIÓN RECIBE EL VALOR ASCII DE UN CARACTER QUE MUESTRA UN SÍMBOLO EN LA CONSOLA Y LO CONVIERTE A SU CORRESPONDIENTE CARACTER

OBSERVACIONES:

\*/

intcaracterLatino**(**int c**){**

**switch(**c**){**

**case**177**:return**164**;**//ñ

**break;**

**case**145**:return**165**;**//Ñ

**break;**

**case**193**:return**181**;**//Á

**break;**

**case**201**:return**144**;**//É

**break;**

**case**205**:return**214**;**//Í

**break;**

**case**211**:return**224**;**//Ó

**break;**

**case**218**:return**233**;**//Ú

**break;**

**case**225**:return**160**;**//á

**break;**

**case**233**:return**130**;**//é

**break;**

**case**237**:return**161**;**//í

**break;**

**case**243**:return**162**;**//ó

**break;**

**case**250**:return**163**;**//ú

**break;**

**case**241**:return**164**;**//ñ

**break;**

**case**209**:return**165**;**//Ñ

**break;**

**case**252**:return**129**;**//ü

**break;**

**default:return** c**;**//RETORNA EL MISMO CARACTER

**break;**

**}**

**}**

La función caracterLatino hace la corrección de los caracteres (indicados en los comentarios) que se encuentran en una cadena de texto o en un archivo de texto, a través de esta conversión es posible representar los caracteres de manera adecuada en pantalla.

/\*

FUNCIÓN: cargarDiccionario(tablaHash \*diccionario).

RECIBE: int (DIRECCIÓN DE MEMORIA DE LA TABLA HASH A OPERAR).

DEVUELVE: NADA.

DESCRIPCIÓN: SE ENCARGA DE EXTRAER LOS CARACTERES CONTENIDOS EN UN ARCHIVO DE TEXTO, SEPARANDO LA PALABRA DE SU DEFINICIÓN. POSTERIORMENTE

AGREGA ESTE PAR DE CADENAS A UN ELEMENTO Y ES AGREGADO A UNA LISTA DENTRO DE LA TABLA HASH INDICADA EN EL PARÁMETRO.

OBSERVACIONES:

\*/

voidcargarDiccionario**(**tablaHash**\***diccionario**){**

FILE**\***archivo**;**

registro nuevaPalabra**;**

char rutaArchivo**[**100**],**palabra**[**100**],**palabraAlt**[**100**],**definicion**[**251**],**estado**=**'p'**;**

/\* ES NECESARIOO CREAR LA VARIABLE palabraAlt PUESTO QUE ALGUNAS PALABRAS TIENEN "ALTERNATIVAS"

MARCADAS POR EL SIMBOLO /, EL PROGRAMA DEBE RECONOCERLAS COMO PALABRAS DIFERENTES

LA VARIABLE estado INDICA LA TRANSICIÓN DE PALABRA A DEFINICIÓN

\*/

intcharArchivo**,**i**=**0**;**

printf**(**"Ingresa la ruta del archivo:\n>"**);**

scanf**(**"%s"**,&**rutaArchivo**);**

archivo **=**fopen**(**rutaArchivo**,**"r"**);**

palabraAlt**[**0**]=**'#'**;**

**while((**charArchivo**=**getc**(**archivo**))!=** EOF**){**

charArchivo**=**caracterLatino**(**charArchivo**);**

**if(**charArchivo**!=**195**){**

**if(**charArchivo**==**'/'**&&** estado **==**'p'**){**//ESTA PALABRA TIENE UNA VERSIÓN ALTERNATIVA

estado**=**'q'**;**

palabra**[**i**]=**'\0'**;**

i**=**0**;**

**continue;**

**}**

**if(**charArchivo**==**':'**){**//COMIENZA LA DEFINICIÓN

**if(**estado **==**'p'**)**//SE TRATA DE UNA PALABRA SIN ALTERNATIVAS

palabra**[**i**]=**'\0'**;**

**if(**estado **==**'q'**)**//SE TRATA DE LA ALTERNATIVA DE ALGUNA PALABRA

palabraAlt**[**i**]=**'\0'**;**

estado**=**'s'**;**

i**=**0**;**

**continue;**

**}**

**if(**charArchivo**==**'\n'**){**//SI OCURRE UN SALTO DE LÍNEA ES EL FINAL DE LA DEFINICIÓN

definicion**[**i**]=**charArchivo**;**

definicion**[**i**+**1**]=**'\0'**;**//SE CIERRA LA DEFINICIÓN

strcpy**(&**nuevaPalabra**.**palabra**[**0**],&**palabra**[**0**]);**//LA PALABRA SE COPIA AL ATRIBUTO palabra DEL NUEVO registro

strcpy**(&**nuevaPalabra**.**definicion**[**0**],&**definicion**[**0**]);**//LA PALABRA SE COPIA AL ATRIBUTO palabra DEL NUEVO registro

AgregarATabla**(**diccionario**,**nuevaPalabra**);**

printf**(**"\nPalabra %s agregada exitosamente!"**,**nuevaPalabra**.**palabra**);**

**if(**palabraAlt**[**0**]!=**'#'**){**

strcpy**(&**nuevaPalabra**.**palabra**[**0**],&**palabra**[**0**]);**

strcpy**(&**nuevaPalabra**.**definicion**[**0**],&**definicion**[**0**]);**

AgregarATabla**(**diccionario**,**nuevaPalabra**);**

printf**(**"\nPalabra %s agregada exitosamente!"**,**nuevaPalabra**.**palabra**);**

palabraAlt**[**0**]=**'#'**;**

**}**

fflush**(**stdout**);**

estado **=**'p'**;**

palabra**[**0**]=**'#'**;**

i**=**0**;**

**continue;**

**}**

**if(**estado **==**'p'**){**//ESCRIBE LA PALABRA NORMAL

palabra**[**i**]=**charArchivo**;**

i**++;**

**}**

**if(**estado **==**'q'**){**//COMIENzA A ESRIBIR LA PALABRA ALTERNA

palabraAlt**[**i**]=**charArchivo**;**

i**++;**

**}**

**if(**estado **==**'s'**){**//COMIENZA A ESCRIBIR LA DEFINICIÓN

definicion**[**i**]=**charArchivo**;**

i**++;**

**}**

**}**

**}**

printf**(**"\n\nPor favor, espere..."**);**

sleep**(**2**);**

system**(**"cls"**);**

//system("clear");

**return;**

**}**

La función cargar diccionario es capaz de leer un archivo de texto carácter por carácter, cada uno de estos es enviado a la función caracterLatino para una posible corrección y una vez formada una palabra con su respectiva definición son agregadas a un registro (elemento definido como registro), este registro es agregado a su correspondiente lista a través de la función AgregarATabla. Este proceso se repite hasta que el carácter leído del archivo sea igual al carácter *EndOf File* (EOF).

/\*

FUNCIÓN: nuevaPalabra(tablaHash \*diccionario).

RECIBE: int (DIRECCIÓN DE MEMORIA DE LA TABLA HASH A OPERAR).

DEVUELVE: NADA.

DESCRIPCIÓN: SE ENCARGA DE SOLICITAR UNA NUEVA PALABRA CON SU RESPECTIVA DEFINICIÓN, AÑADE ESTAS CADENAS A UN NUEVO ELEMENTO Y ÉSTE ES AGREGADO

AL DICCIONARIO INDICADO EN EL PARÁMETRO.

OBSERVACIONES: NO SE VALIDA SI EL USUARIO INTRODUCE UNA PALABRA CON ALGUNA ALTERNTIVA. EJEMPLO: PIEDRA/ROCA

\*/

voidnuevaPalabra**(**tablaHash**\***diccionario**){**

registro nueva**;**

char palabra**[**100**],**definicion**[**255**];**

intaux**;**

printf**(**"Ingresa la nueva palabra:\n>"**);**

scanf**(**"%s[a-zA-Z ]"**,&**palabra**);**

fflush**(**stdin**);**

//LIMPIANDO CARACTERES

**for(**aux**=**0**;**aux**<**strlen**(**palabra**);**aux**++){**

palabra**[**aux**]=**caracterLatino**(**palabra**[**aux**]);**

**}**

printf**(**"Ingresa su definicion:\n>"**);**

scanf**(**"%[a-zA-Z ,.]s"**,&**definicion**);**

fflush**(**stdin**);**

//LIMPIANDO CARACTERES

**for(**aux**=**0**;**aux**<**strlen**(**definicion**);**aux**++){**

definicion**[**aux**]=**caracterLatino**(**definicion**[**aux**]);**

**}**

strcpy**(&**nueva**.**palabra**[**0**],&**palabra**[**0**]);**

strcpy**(&**nueva**.**definicion**[**0**],&**definicion**[**0**]);**

AgregarATabla**(**diccionario**,**nueva**);**

printf**(**"Palabra %s agregada exitosamente!\n"**,**palabra**);**

printf**(**"Por favor, espere...\n"**);**

sleep**(**2**);**

system**(**"cls"**);**

//system("clear");

**return;**

**}**

La función nuevaPalabra solicita al usuario que introduzca una nueva palabra para ingresar en la tabla, posteriormente solicita su definición y estas dos cadenas son agregadas a un registro. Dicho registro es enviado como parámetro a la función AgregarATabla para su inserción en la tabla hash.

/\*

FUNCIÓN: buscaPalabra(tablaHash \*diccionario).

RECIBE: int (DIRECCIÓN DE MEMORIA DE LA TABLA HASH A OPERAR).

DEVUELVE: NADA.

DESCRIPCIÓN: SE ENCARGA DE SOLICITAR LA PALABRA QUE SE DESEA BUSCAR, ÉSTA ES INGRESADA A UN NUEVO ELEMENTO TEMPORAL Y ENVIADA A LA FUNCIÓN

CORRESPONDIENTE DE LA LIBRERÍA TADHashTable.h

OBSERVACIONES: EL USUARIO NO DEBE INTRODUCIR PALABRAS ALTERNATIVAS EN LA MISMA CADENA. NO ES NECESARIO SOLICITAR LA DEFINICIÓN.

\*/

voidbuscaPalabra**(**tablaHash**\***diccionario**){**

registro buscame**,**encontrada**;**

char palabra**[**100**],**null**[**4**]=**"NULL"**;**

intaux**,**index**;**

printf**(**"Ingresa la palabra a buscar:\n>"**);**

scanf**(**"%s[a-zA-Z ]"**,**palabra**);**

**for(**aux**=**0**;**aux**<**strlen**(**palabra**);**aux**++){**//LIMPIAMOS LA CADENA POR ESO DE LOS CArACTERES RAROS

palabra**[**aux**]=**caracterLatino**(**palabra**[**aux**]);**

**}**

strcpy**(&**buscame**.**palabra**[**0**],&**palabra**[**0**]);**//COPIAMOS LA PALABRA INGRESADA A LA PALABRA DEL NUEVO ELEMENTO

encontrada**=**BuscarEnTabla**(**diccionario**,**buscame**);**

**if(**strcmp**(&**encontrada**.**palabra**[**0**],&**null**[**0**])!=**0**){**

printf**(**"%s: %s\n"**,**encontrada**.**palabra**,**encontrada**.**definicion**);**

**}**

system**(**"pause"**);**

system**(**"cls"**);**

//system("clear");

**return;**

**}**

La función buscaPalabra solicita al usuario que introduzca la palabra de la cual desea consultar su definición, esta palabra es añadida a un registro auxiliar y enviado a la función BuscarEnTabla para que regrese la definición de dicha palabra en caso de encontrarse en el diccionario.

/\*

FUNCIÓN: eliminarPalabra(tablaHash \*diccionario).

RECIBE: int (DIRECCIÓN DE MEMORIA DE LA TABLA HASH A OPERAR).

DEVUELVE: NADA.

DESCRIPCIÓN: SE ENCARGA DE SOLICITAR LA PALABRA QUE SE DESEA ELIMINAR, LA ALMACENA EN UN NUEVO ELEMENTO TEMPORAL Y LA ENVÍA A LA RESPECTIVA

FUNCIÓN DE LA LIBRERÍA TADHashTable.h

OBSERVACIONES: EL USUARIO NO DEBE INTRODUCIR PALABRAS ALTERNATIVAS EN LA MISMA CADENA. NO ES NECESARIO SOLICITAR LA DEFINICIÓN.

\*/

voideliminarPalabra**(**tablaHash**\***diccionario**){**

registro eliminame**;**

charpalabra**[**100**];**

intaux**,**index**;**

printf**(**"Ingresa la palabra a eliminar\n>"**);**

scanf**(**"%s[a-zA-Z ]"**,&**palabra**);**

**for(**aux**=**0**;**aux**<**strlen**(**palabra**);**aux**++){**//LIMPIAMOS LA CADENA POR ESO DE LOS CArACTERES RAROS

palabra**[**aux**]=**caracterLatino**(**palabra**[**aux**]);**

**}**

strcpy**(&**eliminame**.**palabra**[**0**],&**palabra**[**0**]);**//COPIAMOS LA PALABRA INGRESADA A LA PALABRA DEL NUEVO ELEMENTO

printf**(**"Eliminando...\n"**);**

EliminarDeTabla**(**diccionario**,**eliminame**);**

printf**(**"Eliminada exitosamente!\nPor favor espere...\n"**);**

sleep**(**2**);**

system**(**"cls"**);**

//system("clear");

**return;**

**}**

La función eliminarPalabra solicita la usuaria que introduzca la palabra que desea eliminar del diccionario, esta cadena es agregada a un registro auxiliar y enviada a la función

/\*

FUNCIÓN: modificarDefinicion(tablaHash \*diccionario).

RECIBE: int (DIRECCIÓN DE MEMORIA DE LA TABLA HASH A OPERAR).

DEVUELVE: NADA.

DESCRIPCIÓN: SE ENCARGA DE SOLICITAR UNA PALABRA Y SU NUEVA DEFINICIÓN, ESTAS CADENAS SON AGREGADAS A UN NUEVO ELEMENTO Y ENVIADAS A LA RESPECTIVA

FUNCIÓN DE LA LIBRERÍA TADHasTable.h

OBSERVACIONES: NO SE VALIDA SI EL USUARIO INTRODUCE UNA PALABRA CON ALGUNA ALTERNTIVA. EJEMPLO: PIEDRA/ROCA

\*/

voidmodificarDefinicion**(**tablaHash**\***diccionario**){**

registro eModificado**;**

char palabra**[**100**],**nuevaDefinicion**[**255**];**

intaux**,**index**;**

printf**(**"Ingresa la palabra a modificar su definicion\n>"**);**

scanf**(**"%s[a-zA-Z ]"**,&**palabra**);**

**for(**aux**=**0**;**aux**<**strlen**(**palabra**);**aux**++){**//LIMPIAMOS LA CADENA POR ESO DE LOS CArACTERES RAROS

palabra**[**aux**]=**caracterLatino**(**palabra**[**aux**]);**

**}**

strcpy**(&**eModificado**.**palabra**[**0**],&**palabra**[**0**]);**//COPIAMOS LA PALABRA INGRESADA A LA PALABRA DEL ELEMENTO ACTUAL

printf**(**"Ingresa la nueva definicion:\n>"**);**

scanf**(**"%[a-zA-Z ,.]s"**,&**nuevaDefinicion**);**

**for(**aux**=**0**;**aux**<**strlen**(**nuevaDefinicion**);**aux**++){**//LIMPIAMOS LA CADENA POR ESO DE LOS CArACTERES RAROS

nuevaDefinicion**[**aux**]=**caracterLatino**(**nuevaDefinicion**[**aux**]);**

**}**

strcpy**(&**eModificado**.**definicion**[**0**],&**nuevaDefinicion**[**0**]);**

printf**(**"Modificando...\n"**);**

ModificarTabla**(**diccionario**,**eModificado**);**

printf**(**"Modificada exitosamente!\nPor favor espere...\n"**);**

sleep**(**2**);**

system**(**"cls"**);**

//system("clear");

**return;**

**}**

**FUNCIONAMIENTO**

**POSIBLES MEJORAS**

**CONCLUSIONES**

**YASMIN:**

Una tabla hash al ser un método para ingresar y buscar información nos damos cuenta que puede llegar a ser muy eficaz pero esto depende también del algoritmo de la función hash que se implemente ya que nuestra función tiene que resolver el mayor tipo de colisiones. Al tener cada elemento una llave asociada nos facilita el acceso a los elementos o el ingreso/modificación de estos.

Hay distintos tipos de hash y así mismo también hay distintas formas de resolver las colisiones esto depende mucho del tamaño de nuestras tablas, la cantidad de elementos, etc.

Al implementar tablas hash nos dimos cuenta que es un buen método para el manejo de la información de forma eficiente y ordenada.

Además que al tener un carácter asociativo y volviendo a mencionar su eficiencia, son muy útiles cuando el tiempo de acceso a la información es crítico.

**ANEXOS**