Concurso Auxiliar de 2da categoria

Estructuras de Datos y Algoritmos I

Damian Ariel Marotte

Enunciado

Dadas las siguientes definiciones:

```
struct Empleado {
    char* nombre, *direccion, *dni;
    float sueldo;
    struct Empleado* sig;
};

struct Hash {
    int tam;
    ListaEmpleados *tabla;
};
```

1. Escriba una funcion que retorne la cantidad de elementos que se almacenan en la tabla.
2. Diseñe una funcion que calcule el *indice radial* de una tabla hash. Se define el indice radial de una Tabla Hash como el numero de casillas de la tabla por el numero de elementos de la lista enlazada de mayor tamaño presente en la tabla.

estructuras.h

```
typedef struct Empleado {
    char* nombre, *direccion, *dni;
    float sueldo;
    struct Empleado* sig;
} Nodo;
typedef Nodo* ListaEmpleados;

struct Hash {
    int tam;
    struct Empleado* *tabla;
};
typedef struct Hash* TablaHash;
```

estructuras.h

```
typedef struct Empleado {
    char* nombre, *direccion, *dni;
    float sueldo;
    struct Empleado* sig;
} Nodo;
typedef Nodo* ListaEmpleados;

struct Hash {
    int tam;
    struct Empleado* *tabla;
};
typedef struct Hash* TablaHash;
```

Ejercicio 1

Escriba una funcion que retorne la cantidad de elementos que se almacenan en la tabla.

main.c

```c
#include "estructuras.h"

void main() {
}
```

Ejercicio 1

Escriba una funcion que retorne la cantidad de elementos que se almacenan en la tabla.

Longitud de una lista

main.c

```c
#include "estructuras.h"

// Funcion que dada una lista, retorna la cantidad de elementos que tiene.
int longitud(ListaEmpleados lista) {
}

void main() {
}
```

Ejercicio 1

Escriba una funcion que retorne la cantidad de elementos que se almacenan en la tabla.

main.c

```c
#include <assert.h> // assert
#include <stdlib.h> // NULL
#include "estructuras.h"

// Funcion que dada una lista, retorna la cantidad de elementos que tiene.
int longitud(ListaEmpleados lista) {
}

void main() {
    assert(longitud(NULL) == 0); // Longitud de una lista vacia.

    struct Empleado nodo1;
    nodo1.nombre = "Damian"; nodo1.dni = "31.116.234";
    nodo1.sueldo = 4000; nodo1.sig = NULL;
    assert(longitud(&nodo1) == 1); // Longitud de una lista unitaria.
}
```

Caso base

main.c

```c
// Funcion que dada una lista, retorna la cantidad de elementos que tiene.
int longitud(ListaEmpleados lista) {
    if (lista == NULL) // Lista vacia.
        return 0;
}
```

Ejercicio 1

Escriba una funcion que retorne la cantidad de elementos que se almacenan en la tabla.

Caso recursivo

main.c

```c
// Funcion que dada una lista, retorna la cantidad de elementos que tiene.
int longitud(ListaEmpleados lista) {
    if (lista == NULL) // Lista vacia.
        return 0;
    else
        return 1 + longitud(lista->sig);
}
```

Ejercicio 1

Escriba una funcion que retorne la cantidad de elementos que se almacenan en la tabla.

Cantidad de elementos de una tabla hash

main.c

```
// Funcion que dada una tabla hash, retorna su cantidad de elementos.
int elementos(TablaHash hash) {
}

void main() {
    assert(longitud(NULL) == 0); // Longitud de una lista vacia.

    struct Empleado nodo1;
    nodo1.nombre = "Damian"; nodo1.dni = "31.116.234";
    nodo1.sueldo = 4000; nodo1.sig = NULL;
    assert(longitud(&nodo1) == 1); // Longitud de una lista unitaria.
}
```

Ejercicio 1

Escriba una funcion que retorne la cantidad de elementos que se almacenan en la tabla.

main.c

```c
// Funcion que dada una una tabla hash, retorna su cantidad de elementos.
int elementos(TablaHash hash) {
}

void main() {
    assert(longitud(NULL) == 0); // Longitud de una lista vacia.

    struct Empleado nodo1;
    nodo1.nombre = "Damian"; nodo1.dni = "31.116.234";
    nodo1.sueldo = 4000; nodo1.sig = NULL;
    assert(longitud(&nodo1) == 1); // Longitud de una lista unitaria.

    struct Hash docentes;
    docentes.tam = 5;
    docentes.tabla = malloc(sizeof(struct Empleado*) * docentes.tam);
    docentes.tabla[0] = &nodo1; docentes.tabla[1] = &nodo1;
    assert(elementos(&docentes) == 2); // Cantidad de elementos de una tabla con
    free(docentes.tabla);              // dos listas unitarias.
}
```

Cantidad de elementos de una tabla hash

main.c

```
// Funcion que dada una tabla hash, retorna su cantidad de elementos.
int elementos(TablaHash hash) {
    assert(hash != NULL); // Validamos la entrada

    int cantidad = 0;

    // Recorremos el arreglo
    for(int i = 0; i < hash->tam; i++)
        cantidad += longitud(hash->tabla[i]);

    return cantidad;
}
```

Ejercicio 1

Escriba una funcion que retorne la cantidad de elementos que se almacenan en la tabla.

main.c

```c
// Funcion que dada una lista, retorna la cantidad de elementos que tiene.
int longitud(ListaEmpleados lista) {
    if (lista == NULL) // Lista vacia.
        return 0;
    else
        return 1 + longitud(lista->sig);
}
// Funcion que dada una tabla hash, retorna su cantidad de elementos.
int elementos(TablaHash hash) {
    assert(hash != NULL); // Validamos la entrada.

    int cantidad = 0;
    // Recorremos el arreglo
    for(int i = 0; i < hash->tam; i++)
        cantidad += longitud(hash->tabla[i]);

    return cantidad;
}
```

main.c

```c
#include <assert.h> // assert
#include <stdlib.h> // NULL
#include "estructuras.h"

// ...

void main() {
    // ...
}
```

Ejercicio 2

Diseñe una funcion que calcule el *indice radial* de una tabla hash. Se define el indice radial de una Tabla Hash como el numero de casillas de la tabla por el numero de elementos de la lista enlazada de mayor tamaño presente en la tabla.

Indice radial

main.c

```c
#include <assert.h> // assert
#include <stdlib.h> // NULL
#include "estructuras.h"

// ...

// Funcion que dada una tabla hash, devuelve el indice radial.
int radial(TablaHash hash) {
}

void main() {
    // ...
}
```

Ejercicio 2

Diseñe una funcion que calcule el *indice radial* de una tabla hash. Se define el indice radial de una Tabla Hash como el numero de casillas de la tabla por el numero de elementos de la lista enlazada de mayor tamaño presente en la tabla.

Casos de prueba

main.c

```c
#include <assert.h> // assert
#include <stdlib.h> // NULL
#include "estructuras.h"
// ...

// Funcion que dada una tabla hash, devuelve el indice radial.
int radial(TablaHash hash) {
}

void main() {
    // ...
    assert(radial(&docentes) == 5); // 5 casilleros, 2 listas unitarias
}
```

Ejercicio 2

Diseñe una funcion que calcule el *indice radial* de una tabla hash. Se define el indice radial de una Tabla Hash como el numero de casillas de la tabla por el numero de elementos de la lista enlazada de mayor tamaño presente en la tabla.

Indice radial

main.c

```c
// Funcion que dada un tabla hash, devuelve el indice radial.
int radial(TablaHash hash) {
    assert(hash != NULL); // Validamos la entrada.

    int maximo = 0;
    // Recorremos el arreglo
    for (int i = 0; i < hash->tam; i++) {
        int temp = longitud(hash->tabla[i]);
        if (temp > maximo) maximo = temp;
    }

    return hash->tam * maximo;
}
```

Ejercicio 2

Diseñe una funcion que calcule el *indice radial* de una tabla hash. Se define el indice radial de una Tabla Hash como el numero de casillas de la tabla por el numero de elementos de la lista enlazada de mayor tamaño presente en la tabla.

main.c

```c
int longitud(ListaEmpleados lista) {
    if (lista == NULL) return 0;
    else return 1 + longitud(lista->sig);
}
int elementos(TablaHash hash) {
    assert(hash != NULL);
    int cantidad = 0;
    for(int i = 0; i < hash->tam; i++) cantidad += longitud(hash->tabla[i]);
    return cantidad;
}
int radial(TablaHash hash) {
    assert(hash != NULL);
    int maximo = 0;
    for (int i = 0; i < hash->tam; i++) {
        int temp = longitud(hash->tabla[i]);
        if (temp > maximo) maximo = temp;
    }
    return hash->tam * maximo;
}
```