Explicación de la tabla Hash en C

Nombre: Aldo Ray Vasquez Lopez

Fecha: 23/10/2023

```
struct RandomHashFunction {
    int* fnTable;
};
```

- Esta estructura ***RandomHashFunction*** se utiliza para almacenar una tabla de valores hash aleatorios.

```
void initRandomHashFunction(struct RandomHashFunction* randomFn, int M, int n) {
    randomFn->fnTable = (int*)malloc(M * sizeof(int));
    srand(time(NULL));

    for (int x = 0; x < M; x++) {
        randomFn->fnTable[x] = rand() % n;
    }
}
```

- ***initRandomHashFunction*** es una función que inicializa la función hash aleatoria. Recibe un puntero a una estructura ***RandomHashFunction***, el tamaño *M* de la tabla hash y el valor máximo *n* para generar números aleatorios.
- Primero, reserva memoria para ***fnTable*** y luego utiliza ***srand*** para inicializar la semilla para la generación de números aleatorios.
- Luego, llena ***fnTable*** con *M* valores hash aleatorios calculados como el módulo de valores aleatorios entre 0 y *n*.

```
void destroyRandomHashFunction(struct RandomHashFunction* randomFn) {
    free(randomFn->fnTable);
}
```

- ***destroyRandomHashFunction*** es una función que libera la memoria utilizada por la función hash aleatoria.
- Recibe un puntero a una estructura ***RandomHashFunction*** y libera la memoria asignada para ***fnTable***.

```
struct HashTable {
    int miAtributo;
    int** bucket;
    struct RandomHashFunction hashFunction;
};
```

- Esta estructura ***HashTable*** representa la tabla hash. Contiene un entero ***miAtributo***, un puntero a un puntero de enteros ***bucket***, y una estructura ***RandomHashFunction*** para la función hash aleatoria.

```
struct HashTable* createHashTable(int h, int n) {
    struct HashTable* table = (struct HashTable*)malloc(sizeof(struct HashTable));
    table->miAtributo = h;
    table->bucket = (int**)malloc(n * sizeof(int*));
    initRandomHashFunction(&(table->hashFunction), n, n);

    for (int i = 0; i < n; i++) {
        table->bucket[i] = NULL;
    }

    return table;
}
```

- ***createHashTable*** es una función que crea e inicializa una tabla hash. Recibe un entero h y un entero n como argumentos.
- Reserva memoria para la estructura ***HashTable***, establece ***miAtributo*** en h y reserva memoria para el array ***bucket*** con n elementos.
- Llama a ***initRandomHashFunction*** para inicializar la función hash aleatoria con n valores.
- Inicializa cada entrada del ***bucket*** como ***NULL***.

```
int mi_Mod(int x, int n) {
    return x % n;
}
```

- ***mi_Mod*** es una función que calcula el módulo de un número x con respecto a un valor n.

```c
void insert(struct HashTable* table, int x) {
    int index = table->hashFunction.fnTable[x];
    int* newList = NULL;
    int length = 0;

    if (table->bucket[index] == NULL) {
        newList = (int*)malloc(2 * sizeof(int));
        newList[0] = x;
        newList[1] = -1;
    } else {
        while (table->bucket[index][length] != -1) {
            length++;
        }
        newList = (int*)malloc((length + 2) * sizeof(int));

        for (int i = 0; i < length; i++) {
            newList[i] = table->bucket[index][i];
        }
        newList[length] = x;
        newList[length + 1] = -1;
        free(table->bucket[index]);
    }

    table->bucket[index] = newList;
}
```

- ***insert*** es una función que inserta un elemento x en la tabla hash. Recibe un puntero a la tabla hash y el elemento ***x***.
- Calcula el índice utilizando la función hash aleatoria y luego crea un nuevo array ***newList***.
- Si ***table->bucket[index]*** es ***NULL***, se crea un nuevo array con dos elementos que son x y -1. Si no es ***NULL***, se agrega ***x*** al final del array existente.

```c
void removeItem(struct HashTable* table, int x) {
    int index = table->hashFunction.fnTable[x];
    if (table->bucket[index] != NULL) {
        int length = 0;
        while (table->bucket[index][length] != -1) {
            length++;
        }

        int* newList = (int*)malloc((length - 1) * sizeof(int));
        int i = 0;
        int j = 0;

        while (i < length) {
            if (table->bucket[index][i] != x) {
                newList[j] = table->bucket[index][i];
                j++;
            }
            i++;
        }

        free(table->bucket[index]);
        if (j > 0) {
            newList[j] = -1;
            table->bucket[index] = newList;
        } else {
            table->bucket[index] = NULL;
        }
    }
}
```

- ***removeItem*** es una función que elimina un elemento específico ***x*** de la tabla hash. Comienza por calcular el índice en la tabla usando la función hash aleatoria. Luego, verifica si hay elementos en ese índice. Si los hay, calcula la longitud del arreglo en ese índice, crea un nuevo arreglo para almacenar los elementos sin ***x***, copia los elementos originales excluyendo ***x***, libera la memoria del arreglo original y finalmente actualiza el puntero en la tabla hash con el nuevo arreglo.

```
int find(struct HashTable* table, int x) {
    int index = table->hashFunction.fnTable[x];
    if (table->bucket[index] != NULL) {
        int i = 0;
        while (table->bucket[index][i] != -1) {
            if (table->bucket[index][i] == x) {
                return x;
            }
            i++;
        }
    }
    return -1;
}
```

- La función ***find*** se utiliza para buscar un elemento específico ***x*** en la tabla hash. Comienza por calcular el índice en la tabla usando la función hash aleatoria. Luego, verifica si hay elementos en ese índice. Si los hay, recorre el arreglo en ese índice en busca de **x**. Si se encuentra ***x***, devuelve ***x***. Si no se encuentra, devuelve -1. Esta función se utiliza para determinar si un elemento dado existe en la tabla hash y, en caso afirmativo, devuelve el valor.

```
void destroyHashTable(struct HashTable* table) {
    for (int i = 0; i < table->miAtributo; i++) {
        if (table->bucket[i] != NULL) {
            free(table->bucket[i]);
        }
    }
    free(table->bucket);
    destroyRandomHashFunction(&(table->hashFunction));
    free(table);
}
```

La función ***destroyHashTable*** se encarga de liberar la memoria utilizada por la tabla hash y sus componentes.

1. Recorre la tabla hash y, para cada índice, verifica si hay un arreglo. Si existe un arreglo en un índice, se libera la memoria de ese arreglo.
2. Después de liberar la memoria de los arreglos individuales, se libera la memoria del arreglo principal ***bucket***.
3. Luego, llama a la función ***destroyRandomHashFunction*** para liberar la memoria utilizada por la función hash aleatoria almacenada en la estructura de la tabla.
4. Finalmente, libera la memoria de la propia estructura de la tabla hash **table**.

```c
int main() {
    struct HashTable* hashTable = createHashTable(7, 10);

    insert(hashTable, 7);
    insert(hashTable, 15);
    insert(hashTable, 25);

    printf("Buscando 15: %d\n", find(hashTable, 15));
    printf("Buscando 7: %d\n", find(hashTable, 7));

    removeItem(hashTable, 15);

    printf("Buscando 15 después de eliminarlo: %d\n", find(hashTable, 15));

    destroyHashTable(hashTable);

    return 0;
}
```

1. Se crea una tabla hash utilizando la función ***createHashTable***, con un tamaño de 10 y el atributo ***miAtributo*** establecido en 7.
2. Se insertan tres elementos (7, 15 y 25) en la tabla hash utilizando la función ***insert***.
3. Se busca el elemento 15 en la tabla hash utilizando la función ***find*** y se imprime el resultado. Luego, se busca el elemento 7 y se imprime su resultado.
4. Se elimina el elemento 15 de la tabla hash utilizando la función ***removeItem***.
5. Se busca nuevamente el elemento 15 en la tabla hash después de eliminarlo y se imprime el resultado, que debería ser -1, ya que se eliminó.
6. Finalmente, se destruye la tabla hash utilizando la función ***destroyHashTable*** para liberar toda la memoria asignada.