Comandos Basicos de las tablas Hash:

1. **put(key, value):** Agrega un par clave-valor a la tabla hash. Si la clave ya existe, el valor existente se reemplaza por el nuevo valor.

Ejemplo en Java:

hashTable.put("clave", valor);

1. **get(key):** Recupera el valor asociado a una clave específica en la tabla hash.

Ejemplo en Java:

ValorType valor = hashTable.get("clave");

1. **containsKey(key):** Verifica si la tabla hash contiene una clave específica y devuelve true si la clave existe, de lo contrario, devuelve false.

Ejemplo en Java:

boolean existe = hashTable.containsKey("clave");

1. **remove(key):** Elimina el par clave-valor asociado a una clave específica de la tabla hash.

Ejemplo en Java:

hashTable.remove("clave");

**5) isEmpty():** Verifica si la tabla hash está vacía y devuelve true si no contiene elementos, de lo contrario, devuelve false.

Ejemplo en Java:

boolean estaVacia = hashTable.isEmpty();

**6.) size():** Devuelve el número de elementos (pares clave-valor) en la tabla hash.

Ejemplo en Java:

int tamaño = hashTable.size();

Conceptos de Directorios:

Los directorios en las tablas hash son estructuras utilizadas para gestionar colisiones. Una colisión ocurre cuando dos elementos tienen la misma clave y deben almacenarse en la misma ubicación de la tabla hash. Los tipos de directorios más comunes son:

1. \*\*Directorio de listas enlazadas:\*\* Cada entrada en la tabla hash contiene una lista enlazada de elementos que tienen la misma ubicación de hash. Cuando se produce una colisión, los nuevos elementos se agregan a la lista enlazada correspondiente.

2. \*\*Directorio de árboles:\*\* Similar al directorio de listas enlazadas, pero en lugar de usar listas enlazadas, se utilizan árboles (por ejemplo, árboles binarios de búsqueda) para gestionar los elementos con la misma ubicación de hash.

3. \*\*Directorio de listas dinámicas:\*\* En lugar de listas enlazadas tradicionales, las listas dinámicas se expanden automáticamente cuando se producen colisiones, evitando así problemas de rendimiento con listas largas.

Si dos elementos tienen claves idénticas, es decir, hay una colisión, puedes resolverla utilizando cualquiera de las siguientes estrategias:

1. \*\*Separate Chaining (Listas enlazadas):\*\* Crea una lista enlazada en cada ubicación de la tabla hash y agrega los elementos con la misma ubicación de hash a esta lista. Aquí tienes un ejemplo en Java:

```java

import java.util.LinkedList;

public class HashTable {

private LinkedList<Pair>[] table;

private int capacity;

public HashTable(int capacity) {

this.capacity = capacity;

table = new LinkedList[capacity];

for (int i = 0; i < capacity; i++) {

table[i] = new LinkedList<>();

}

}

public void put(String key, int value) {

int index = hashFunction(key);

LinkedList<Pair> list = table[index];

for (Pair pair : list) {

if (pair.getKey().equals(key)) {

pair.setValue(value);

return;

}

}

list.add(new Pair(key, value));

}

public int get(String key) {

int index = hashFunction(key);

LinkedList<Pair> list = table[index];

for (Pair pair : list) {

if (pair.getKey().equals(key)) {

return pair.getValue();

}

}

return -1; // Clave no encontrada

}

private int hashFunction(String key) {

return key.hashCode() % capacity;

}

private static class Pair {

private String key;

private int value;

public Pair(String key, int value) {

this.key = key;

this.value = value;

}

public String getKey() {

return key;

}

public int getValue() {

return value;

}

public void setValue(int value) {

this.value = value;

}

}

}

```

2. \*\*Linear Probing (Exploración lineal):\*\* Si se produce una colisión, busca la siguiente ubicación libre en la tabla hash para almacenar el elemento. Si esta ubicación está ocupada, continúa buscando hasta encontrar una posición libre. Aquí tienes un ejemplo en Java:

```java

public class HashTable {

private String[] keys;

private int[] values;

private int capacity;

public HashTable(int capacity) {

this.capacity = capacity;

keys = new String[capacity];

values = new int[capacity];

}

public void put(String key, int value) {

int index = hashFunction(key);

while (keys[index] != null) {

if (keys[index].equals(key)) {

values[index] = value; // Actualizar el valor si la clave ya existe

return;

}

index = (index + 1) % capacity; // Exploración lineal

}

keys[index] = key;

values[index] = value;

}

public int get(String key) {

int index = hashFunction(key);

while (keys[index] != null) {

if (keys[index].equals(key)) {

return values[index];

}

index = (index + 1) % capacity; // Exploración lineal

}

return -1; // Clave no encontrada

}

private int hashFunction(String key) {

return key.hashCode() % capacity;

}

}

```

En resumen, los directorios en las tablas hash se utilizan para gestionar colisiones cuando dos elementos tienen la misma clave. Puedes resolver las colisiones utilizando estrategias como "Separate Chaining" (listas enlazadas) o "Linear Probing" (exploración lineal). Cada enfoque tiene sus ventajas y desventajas, y la elección depende de los requisitos específicos de tu aplicación.

Ejemplo de tabla hash implementado: