Universidad Mariano Gálvez de Guatemala

Facultad de Ingeniería en Sistemas de Información

Curso: Programación III

Ingeniero: José Villatoro

Investigacion de tablas hash

Alder Isaac Solis De Leon Carné: 9490-22-227

Abner Salvador Cancinos Cabrera Carné: 9490-22-2101

Cristhian Sebastián Rodas Arriola, Carné: 9490-22-523

Ángel Emilio Méndez Muralles, Carné: 9490-22-5851

Joshua Iván Andre Méndez Vásquez, Carné: 9490-22-4032

Fecha: 25/05/24

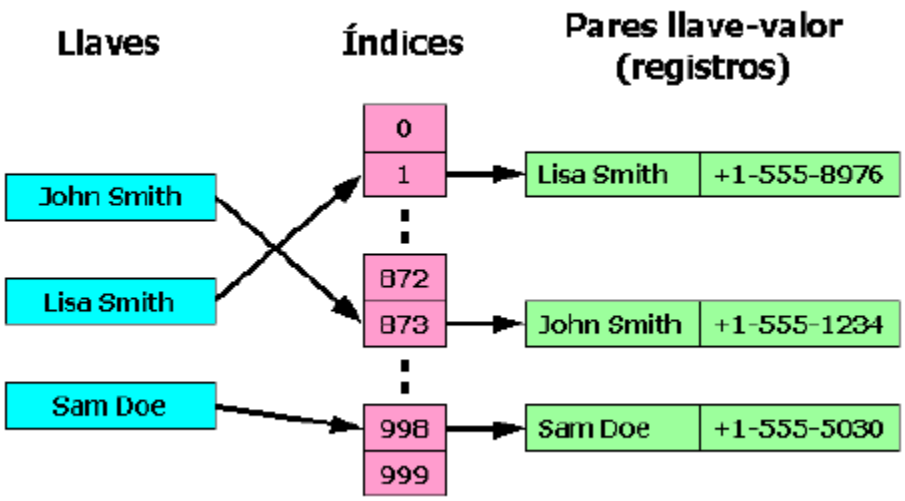
INDICE

Introducción

En el mundo de la informática, las estructuras de datos juegan un papel fundamental en el diseño y la implementación de algoritmos eficientes. Entre estas estructuras, las tablas hash se destacan por su capacidad de proporcionar accesos rápidos a datos mediante el uso de funciones hash. Este documento explora diversos aspectos relacionados con las tablas hash, incluyendo su definición, aplicaciones, el concepto de factor de carga, técnicas para manejar colisiones, y su relación con otras estructuras de datos como diccionarios, conjuntos, listas enlazadas, árboles de búsqueda balanceados, filtros de Bloom y tries. A través de estos temas, se busca ofrecer una comprensión integral de cómo las tablas hash y sus estructuras relacionadas pueden optimizar el almacenamiento y la recuperación de datos en múltiples contextos informáticos.

Tablas Hash:

Una tabla hash es un contenedor asociativo que permite un almacenamiento y posterior recuperación eficiente de elementos (denominados claves) a partir de otros objetos, es decir, una estructura de datos que asocia claves con valores.



Funcionamiento:

El funcionamiento de dichas tablas se puede dividir en 4 partes que son las siguientes:

1. inserción: Para insertar un par clave-valor se aplica la función hash a la clave lo que produce un índice. El valor se almacena en la posición correspondiente a ese índice.

2. Búsqueda: Para buscar un valor asociado a una clave, se aplica la función hash para obtener el índice y se accede a la posición correspondiente.

3. Colisiones: Ocurren cuando dos claves diferentes producen el mismo índice. Las colisiones se pueden manejar de dos maneras.

- Encadenamiento: Cada posición del array apunta a una lista enlazada que contiene todos los pares clave-valor que producen el mismo índice.
- Dirección Abierta: Cuando ocurre una colisión, se busca otra posición en el array para almacenar el par clave-valor.

4. Factor de carga: El factor de carga de una tabla hash es una medida que indica cuan el rendimiento de la tabla hash, se define como la razón entre el numero de elementos almacenados en la tabla y el tamaño total de la tabla, es decir, el numero de buckets disponibles.

Diferentes aplicaciones:

Las tablas hash tiene una amplia variedad de aplicaciones en la informática debido a su eficiencia y rapidez en operaciones de búsqueda, inserción y eliminación.

1. Bases de Datos:

Indexación: Las tablas hash se utilizan para indexar registros en bases de datos, permitiendo búsquedas rápidas de datos.

Caches: Se utilizan en sistemas de caché para almacenar y recuperar datos de manera eficiente.

2. Compiladores e Intérpretes:

Tablas de Símbolos: Los compiladores utilizan tablas hash para almacenar información sobre identificadores (variables, funciones, etc.) durante el análisis léxico y sintáctico.

3. Sistemas Operativos:

Tablas de Páginas: En la gestión de memoria, se utilizan tablas hash para implementar tablas de páginas que mapean direcciones virtuales a direcciones físicas.

Gestión de Archivos: Se utilizan para implementar sistemas de archivos, donde los nombres de archivo se hash para acelerar las búsquedas.

4. Redes Informáticas:

Ruteo: En algoritmos de enrutamiento, las tablas hash se utilizan para almacenar y buscar rutas rápidamente.

Tablas ARP (Address Resolution Protocol): Utilizan tablas hash para mapear direcciones IP a direcciones MAC.

5. Criptografía:

Funciones Hash Criptográficas: Se utilizan en la creación de firmas digitales, contraseñas y para la integridad de datos.

6. Desarrollo de Software:

Mapas y Diccionarios: En lenguajes de programación, las tablas hash son la base para la implementación de estructuras de datos como diccionarios (Python), mapas (C++), y objetos (JavaScript).

Detección de Duplicados: Se utilizan para identificar duplicados en conjuntos de datos grandes de manera eficiente.

7. Juegos y Simulaciones:

Tableros de Juego: En juegos de ajedrez y otros juegos de mesa, se utilizan tablas hash para almacenar y buscar posiciones de tablero rápidamente.

Simulaciones: Para manejar grandes conjuntos de datos de entidades en simulaciones y videojuegos.

Técnicas para evitar colisiones en las tablas hash:

Como lo mencionamos anteriormente en el apartado de colisiones nos explican que las colisiones ocurren cuando dos claves diferentes generan el mismo índice a través de la función hash, estas técnicas llevan el nombre de encadenamiento y dirección. A continuación, veremos a mas detalle las ventajas y desventajas de cada técnicas.

1. Encadenamiento:

- **Ventajas:**

  Fácil de implementar

  No requiere de un limite en el numero de elementos que se puede almacenar

- **Desventajas:**

  Puede llevar a un uso no uniforme del espacio de memoria.

  El rendimiento puede degradarse si las listas enlazadas se vuelven muy largas

2. Dirección Abierta: Antes de mencionar las ventajas y desventajas es importante estar relacionado con las definiciones de sondeo lineal que se trata de buscar la siguiente posición libre en secuencia lineal. Sondeo cuadrático que busca la siguiente posición libre usando una función cuadrática. Doble Hashing usa una segunda función hash para determinar el intervalo de sondeo.

- **Ventajas:**

  No necesitas estructuras adicionales como listas enlazadas

  Puede utilizar mejor el espacio de memoria

- **Desventajas:**

  Puede llevar a una agrupación de datos, especialmente en el sondeo lineal.

Mas complicado de implementar y gestionar que el encadenamiento.

Importancia del Factor de Carga:

La importancia del factor de carga lo podemos dividir en dos segmentos que son los siguientes:

1. **Desempeño:** Un factor de carga bajo se encuentre entre 0.7 o 0.75, generalmente implica que hay menos colisiones y, por lo tanto, operaciones de búsqueda, inserción y eliminación son las rápidas.

2.**Rendimensionamiento:** Con el factor de carga supera el facto de carga bajo, la tabla de hash generalmente se redimensiona y se rehacen las hash de todos los elementos para mantener un buen desempeño.

Estructuras de datos relacionadas con las tablas hash:

Las tablas hash están relacionadas con varias otras estructuras de datos que comparten principios similares o que se utilizan en combinación con ellas para diversos propósitos, entre los mas conocidos son los siguientes:

1. **Diccionarios:** Los diccionarios son una implementación común de tablas hash en muchos lenguajes de programación. Son estructuras de datos que asocian claves únicas con valores.

2. Sets: Los sets son estructuras de datos que almacenan elementos únicos y utilizan tablas hash internamente para proporcionar operaciones de inserción, eliminación y búsqueda rápidas.

3. **Listas Enlazadas:** Las listas enlazadas se utilizan en la técnica de encadenamiento para resolver colisiones en tablas hash. Cada cubeta de la tabla

hash apunta a una lista enlazada que contiene todos los elementos que hash a esa cubeta.

4. **Arboles de Búsqueda Balanceados:** Árboles como AVL y Red-Black Trees pueden ser utilizados en lugar de listas enlazadas para manejar colisiones en tablas hash, proporcionando tiempos de acceso logarítmicos en el peor caso.

Desarrollo y funcionamiento del ejercicio práctico:

Esta es una implementación de una tabla hash simple utilizando listas enlazadas para manejar colisiones. A continuación, veremos cada parte del código importante para poder explicar cada apartado de manera directa y concisa.

Código utilizado y explicación más importante

Clase HashTable

Inicialización

```
class HashTable:
 def __init__(self, size=100):
  self.size = size
  self.table = [[] for _ in range(size)]
```

- **El constructor __init__ inicializa la tabla hash con un tamaño especificado (por defecto 100). La tabla es una lista de listas vacías, donde cada sublista representa un "bucket" que almacenará pares clave-valor.**

Función Hash

```
 def hash_function(self, key):
  return hash(key) % self.size
```

- **La función hash_function toma una clave y devuelve un índice dentro del rango del tamaño de la tabla hash. Utiliza la función incorporada hash de Python y aplica el operador módulo para asegurarse de que el índice esté dentro de los límites de la tabla.**

Inserción

```
def insert(self, key, value):
  index = self.hash_function(key)
  for kvp in self.table[index]:
    if kvp[0] == key:
      kvp[1] = value
      return
  self.table[index].append([key, value])
```

- **La función insert agrega un par clave-valor a la tabla hash.**
- **Calcula el índice usando hash_function.**
- **Revisa si la clave ya existe en el bucket correspondiente; si es así, actualiza su valor.**
- **Si la clave no existe, agrega el nuevo par clave-valor al bucket.**

Búsqueda por clave

```
def search_by_key(self, key):
  index = self.hash_function(key)
  for kvp in self.table[index]:
    if kvp[0] == key:
      return kvp[1]
  return None
```

- **La función search_by_key busca un valor dado su clave.**
- **Calcula el índice y busca en el bucket correspondiente.**
- **Devuelve el valor si la clave es encontrada; de lo contrario, devuelve None.**

Búsqueda por valor

```
def search_by_value(self, value):
  for bucket in self.table:
    for kvp in bucket:
      if kvp[1] == value:
        return kvp[0]
  return None
```

- **La función search_by_value busca una clave dado su valor.**
- **Itera por todos los buckets y por cada par clave-valor en los buckets.**
- **Devuelve la clave si el valor es encontrado; de lo contrario, devuelve None.**

Mostrar tabla Hash

```
def display(self):
  for i, bucket in enumerate(self.table):
    if bucket:
      print(f"Index {i}: {bucket}")
```

- **La función display imprime el contenido de la tabla hash, mostrando los índices y sus respectivos buckets no vacíos.**

Cargar datos desde CSV

```
def load_from_csv(self, file_path):
  try:
    with open(file_path, mode='r', encoding='utf-8') as file:
      csv_reader = csv.reader(file)
      for row in csv_reader:
        if len(row) == 2:
          self.insert(row[0], row[1])
  except FileNotFoundError:
    print("File not found.")
```

- **La función load_from_csv carga datos desde un archivo CSV.**
- **Abre el archivo, lee cada fila y, si la fila tiene dos elementos, los inserta como un par clave-valor en la tabla hash.**
- **Maneja el error FileNotFoundError si el archivo no es encontrado.**

Función Principal ‘main’

```
def main():
  hash_table = HashTable()

  while True:
    print("\n1. Inserte valores manualmente:")
    print("2. Buscar por clave:")
```

```
print("3. Buscar por valor: ")
print("4. Cargar datos desde archivo CSV: ")
print("5. Mostrar tabla hash: ")
print("6. Salir: ")
choice = input("Seleccione una opción: ")

if choice == '1':
 key = input("Ingrese clave: ")
 value = input("Ingrese valor: ")
 hash_table.insert(key, value)
 print(f"Clave: {key}, Valor: {value}, Hash: {hash_table.hash_function(key)}")

elif choice == '2':
 key = input("Ingrese la clave a buscar: ")
 result = hash_table.search_by_key(key)
 if result is not None:
  print(f"Valor encontrado: {result}")
 else:
  print("Clave no encontrada.")

elif choice == '3':
 value = input("Ingrese el valor a buscar: ")
 result = hash_table.search_by_value(value)
 if result is not None:
  print(f"Clave encontrada: {result}")
 else:
```

```
            print("Valor no encontrado.")

        elif choice == '4':
            file_path = input("Ingrese la ruta del archivo CSV: ")
            hash_table.load_from_csv(file_path)
            print("Datos cargados desde el archivo CSV.")

        elif choice == '5':
            hash_table.display()

        elif choice == '6':
            break

        else:
            print("Opción no válida. Intente nuevamente.")

if __name__ == "__main__":
    main()
```

- **La función main proporciona una interfaz de usuario simple basada en texto para interactuar con la tabla hash.**
- **Ofrece opciones para insertar datos manualmente, buscar por clave o valor, cargar datos desde un archivo CSV, mostrar la tabla hash y salir del programa.**
- **Utiliza un bucle while para seguir mostrando el menú hasta que el usuario elija salir.**

Conclusión

Las tablas hash son una herramienta poderosa en el arsenal de un desarrollador, permitiendo una manipulación de datos rápida y eficiente. Su eficiencia depende en gran medida de un buen manejo del factor de carga y de la implementación adecuada de técnicas para resolver colisiones, como el encadenamiento y la dirección abierta. Además, la relación de las tablas hash con otras estructuras de datos, como listas enlazadas y árboles de búsqueda balanceados, ofrece flexibilidad y eficiencia en diversas aplicaciones. En el ámbito práctico, estructuras como diccionarios y conjuntos en Python demuestran cómo las tablas hash se integran en lenguajes de programación modernos para facilitar el desarrollo de software. Por otro lado, herramientas avanzadas como los filtros de Bloom y los tries muestran cómo los principios subyacentes de las tablas hash se pueden extender para resolver problemas específicos, como la búsqueda eficiente y la comprobación de pertenencia en grandes conjuntos de datos. En conjunto, la comprensión y el uso adecuado de las tablas hash y sus estructuras relacionadas son esenciales para diseñar sistemas de información robustos y escalables.

EGRAFIA

- colaboradores de Wikipedia. (2024, 4 mayo). *Tabla hash*. Wikipedia, la Enciclopedia Libre. https://es.wikipedia.org/wiki/Tabla_hash
- guia-8.pdf (udb.edu.sv)
- Walker, A. (2024, 9 marzo). *Hash Table in Data Structure: Python Example*. Guru99. https://www.guru99.com/es/hash-table-data-structure.html
- Admin. (2022, 23 junio). *¿Qué es una colisión en tablas hash?* TutoManiac. https://tutomaniac.com/que-es-una-colision-en-tablas-hash/#:~:text=Las%20colisiones%20en%20tablas%20hash%20son%20situaciones%20en,hash%2C%20afectando%20las%20operaciones%20de%20inserci%C3%B3n%20y%20b%C3%BAsqueda.
- *Colisión de hash _ AcademiaLab*. (s. f.). https://academia-lab.com/enciclopedia/colision-de-hash/#google_vignette
- hashing.pdf (utfsm.cl)