Nombre: Anthony Fabian Ramirez Orellana

Carné: 9490-22-958

Sección: “A”

Caso de estudio 1.

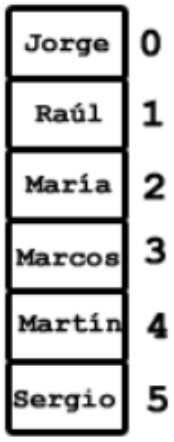
25/05/2024

Tablas hash

Las tablas hash, también conocidas como tablas de dispersión, son una estructura de datos que asocia claves (keys) con valores (values). En Python, la implementación más común de una tabla hash es el diccionario (dictionary).

Una tabla hash se puede ver como un conjunto de entradas. Cada una de estas entradas tiene asociada una clave única, y por lo tanto, diferentes entradas de una misma tabla tendrán diferentes claves. Esto implica, que una clave identifica unívocamente a una entrada en una tabla hash.

Por otro lado, las entradas de las tablas hash están compuestas por dos componentes, la propia clave y la información que se almacena en dicha entrada.



La estructura de las tablas hash es lo que les confiere su gran potencial, ya que hace de ellas unas estructuras extremadamente eficientes a la hora de recuperar información almacenada. El tiempo medio de recuperación de información es constante, es decir, no depende del tamaño de la tabla ni del número de elementos almacenados en la misma. Una tabla hash está formada por un array de entradas, que será la estructura que almacene la información, y por una función de dispersión. La función de dispersión permite asociar el elemento almacenado en una entrada con la clave de dicha entrada. Por lo tanto, es un algoritmo crítico para el buen funcionamiento de la estructura.

Cuando se trabaja con tablas hash es frecuente que se produzcan colisiones. Las colisiones se producen cuando para dos elementos de información distintos, la función de dispersión les asigna la misma clave. Como se puede suponer, esta solución se debe arreglar de alguna forma. Para ello las tablas hash cuentan con una función de resolución de colisiones.

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

- Encadenamiento separado: Las colisiones 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.

Características de las tablas hash

Eficiencia: Las tablas hash proporcionan una forma eficiente de buscar, insertar y eliminar pares clave-valor, generalmente en tiempo constante, $O(1)$ O(1), en promedio.

La ocupación no es muy elevada. A partir del 75% de ocupación, se producen demasiadas colisiones y la tabla se vuelve ineficiente. La función resumen distribuye uniformemente las claves. Si la función está mal diseñada, se producirán muchas colisiones.

Hashing: Utilizan una función hash para transformar la clave en un índice dentro de un arreglo (array). Esta función hash convierte la clave en un número que se usa como índice para acceder al valor correspondiente.

El siguiente código muestra cómo funciona la función hash():

```
my_string = "hello world"

# Calculate the hash value of the string
hash_value = hash(my_string)

# Print the string and its hash value
print("String: ", my_string)
print("Hash value: ", hash_value)
```

Si guardamos ese código en un archivo llamado hash.py, podemos ejecutarlo (y ver la salida) así:

```
% python3 hash.py
String:  hello world
Hash value:  2213812294562653681
```

Ejecutandolo de nuevo:

```
% python3 hash.py
String:  hello world
Hash value:  -6318977648087346O9
```

El valor hash es diferente cuando se invoca por segunda vez porque las versiones recientes de Python (a partir de la 3.3) aplican, por defecto, una semilla hash aleatoria para esta función. La

semilla cambia en cada invocación de Python. Dentro de una misma instancia, los resultados serán idénticos.

La función hashing integrada de Python, hash() , devuelve un valor entero que representa el objeto de entrada. El código utiliza el valor hash resultante para determinar la ubicación del objeto en la tabla hash. Esta tabla hash es una estructura de datos que implementa diccionarios y conjuntos.

Colisiones: Cuando dos claves diferentes se transforman en el mismo índice, ocurre una colisión. Las tablas hash en Python resuelven las colisiones mediante un mecanismo llamado "open addressing" o "chaining".

Ocurre cuando dos claves diferentes son transformadas por la función hash en el mismo índice de la tabla. Esto significa que ambos elementos competirían por el mismo espacio en el array subyacente de la tabla hash. Las colisiones son inevitables debido a la naturaleza finita del array y la posibilidad infinita de claves.

Ejemplo de Colisión

Supongamos que tienes una tabla hash con un array de tamaño 10 y una función hash que transforma las claves en índices de 0 a 9. Si las claves "clave1" y "clave2" son transformadas por la función hash en el índice 5, se produce una colisión porque ambas claves desean ocupar el mismo espacio en el array.

Mecanismos:

Encadenamiento (Chaining): Cada entrada del array en la tabla hash apunta a una lista enlazada de todos los elementos que tienen la misma posición hash. Cuando ocurre una colisión, se agrega el nuevo par clave-valor al final de la lista enlazada correspondiente a ese índice.

Ej:

```
tabla_hash = {
    5: [("clave1", "valor1"), ("clave2", "valor2")]
}
```

Dirección Abierta (Open Addressing): Todos los elementos se almacenan en el array mismo. Cuando ocurre una colisión, el algoritmo de hash busca otro lugar disponible en el array mediante técnicas como la sondeo lineal (linear probing), sondeo cuadrático (quadratic probing) o hashing doble (double hashing). Sondeo Lineal (Linear Probing): Busca el siguiente índice libre secuencialmente: Siguiendo con el ejemplo anterior, si el índice 5 está ocupado, se intenta con el índice 6, luego 7, y así sucesivamente.

Factor de carga: El "factor de carga" (load factor) es un concepto importante en el contexto de las tablas hash, ya que influye directamente en el rendimiento y eficiencia de estas estructuras de datos, se define como la relación entre el número de elementos almacenados en la tabla hash (n) y el tamaño del array subyacente (m). Matemáticamente, se expresa como: $\alpha = n/m$

Donde:

- n es el número de elementos almacenados en la tabla hash.
- m es el tamaño del array o la capacidad de la tabla hash.

El factor de carga es crucial porque determina el equilibrio entre el uso de la memoria y la velocidad de las operaciones en la tabla hash.

Algunos de los defectos que tiene son:

Rendimiento: Bajo factor de carga (α bajo): Si el factor de carga es bajo, hay muchas celdas vacías en el array. Esto reduce la probabilidad de colisiones y las operaciones de búsqueda, inserción y eliminación tienden a ser muy rápidas, pero puede resultar en un uso ineficiente de la memoria.

Por otro lado, alto factor de carga (α alto): Si el factor de carga es alto, significa que muchas celdas del array están ocupadas. Esto aumenta la probabilidad de colisiones y puede degradar el rendimiento de las operaciones debido a la necesidad de resolver las colisiones.

Rehasing: Cuando el factor de carga supera un cierto umbral (a menudo alrededor de 0.75), es común que las implementaciones de tablas hash automáticamente redimensionen el array subyacente y rehasheen todos los elementos. Este proceso, conocido como rehashing, redistribuye los elementos para mantener el rendimiento eficiente de las operaciones.

Las aplicaciones más comunes son:

Base de datos:

- Índices: Las tablas hash se utilizan para implementar índices que permiten búsquedas rápidas de registros en bases de datos.
- Tablas de unión (hash join): En las operaciones de unión de bases de datos, las tablas hash pueden ser utilizadas para acelerar la combinación de registros de dos tablas.

Sistemas de archivos:

- Sistemas de archivos distribuidos: Las tablas hash son fundamentales en sistemas de archivos distribuidos como Hadoop Distributed File System (HDFS), donde se utilizan para distribuir y localizar datos de manera eficiente.
- Sistemas de almacenamiento en caché: Utilizadas en sistemas de almacenamiento en caché para la recuperación rápida de datos.

Compiladores:

- Tablas de símbolos: Los compiladores utilizan tablas hash para almacenar información sobre variables, funciones y otras entidades en el código fuente, permitiendo una recuperación rápida durante el proceso de compilación.

Redes:

- Enrutamiento y almacenamiento en caché: Las tablas hash se utilizan en algoritmos de enrutamiento y en cachés de red para acelerar la búsqueda y almacenamiento de datos.
- Filtros Bloom: Utilizados para comprobar la pertenencia de un elemento a un conjunto con un pequeño porcentaje de error, y utilizan múltiples funciones hash.

Seguridad y Criptografía:

- Tablas de contraseñas: Se utilizan para almacenar contraseñas encriptadas y verificar rápidamente la autenticidad de una contraseña introducida.
- Tablas de resumen (hash tables): Usadas en algoritmos de hashing criptográfico para comprobar la integridad de los datos.

Algoritmos y estructuras de datos:

- Conjuntos y Mapas: Las implementaciones de conjuntos (set) y mapas (dict) en muchos lenguajes de programación utilizan tablas hash para realizar operaciones rápidas de inserción, eliminación y búsqueda.
- Tablas de dispersión abiertas (Open Addressing Hash Tables): Implementaciones específicas de tablas hash que manejan colisiones de una manera que evita el uso de listas enlazadas.

Juegos y aplicaciones de IA:

- Tablas de transposición: Utilizadas en motores de juegos como ajedrez y otros juegos de estrategia para almacenar y recuperar posiciones ya evaluadas, acelerando el proceso de toma de decisiones.
- Almacenamiento de estados: En aplicaciones de inteligencia artificial, se utilizan para almacenar y recuperar rápidamente estados visitados o soluciones parciales.

Motores de búsqueda:

- Índices invertidos: En motores de búsqueda, se utilizan para mapear términos de búsqueda a los documentos en los que aparecen, permitiendo una recuperación rápida de resultados de búsqueda relevantes.

Estructuras de datos relacionadas:

Diccionarios (Maps): Los diccionarios son colecciones de pares clave-valor donde cada clave es única, los diccionarios suelen estar implementados utilizando tablas hash para permitir la búsqueda, inserción y eliminación rápida de elementos.

Ej:

```
mi_diccionario = {"clave1": "valor1", "clave2": "valor2"}
print(mi_diccionario["clave1"])  # Salida: valor1
```

Conjuntos (Sets): Los conjuntos son colecciones no ordenadas de elementos únicos, los conjuntos suelen estar implementados con tablas hash para proporcionar operaciones rápidas de comprobación de pertenencia, inserción y eliminación.

Ej:

```
mi_conjunto = {"a", "b", "c"}
mi_conjunto.add("d")
print("a" in mi_conjunto)  # Salida: True
```

Tablas de Símbolos: Tablas que almacenan información sobre identificadores (como variables y funciones) en compiladores e intérpretes, utilizan tablas hash para permitir la búsqueda rápida de información durante la compilación o la ejecución del código.

Índices de base de datos: Estructuras que mejoran la velocidad de recuperación de datos en bases de datos, los índices hash son una implementación común que permite la búsqueda rápida de registros.

Filtros bloom: Estructuras de datos probabilísticas que se utilizan para comprobar si un elemento pertenece a un conjunto, utilizan múltiples funciones hash para determinar la pertenencia con una cierta probabilidad de error.

Tablas de transposicion: Utilizadas en motores de juegos como ajedrez para almacenar y recuperar posiciones evaluadas previamente, utilizan tablas hash para almacenar las posiciones del juego y sus evaluaciones, acelerando la toma de decisiones.

Hash maps cocurrentes: Variantes de tablas hash diseñadas para un entorno multi-hilo, implementan técnicas específicas para manejar la concurrencia y asegurar la integridad de los datos.

Tablas de hashing perfecto: Tablas hash diseñadas para evitar colisiones mediante el uso de una función hash perfecta, garantizan la inserción, eliminación y búsqueda en tiempo constante sin colisiones.

Estructuras de datos persistentes: Estructuras que mantienen versiones anteriores de los datos, algunas implementaciones utilizan tablas hash para manejar la versión y recuperación de datos.

Enlace Git-Hub parte práctica:

https://github.com/tony0001234/Tablas_hash_ProgaIII.git

Enlaces de interés:

https://www.guru99.com/es/hash-table-data-structure.html

https://www.ionos.mx/digitalguide/servidores/seguridad/tablas-hash/

https://kinsta.com/es/blog/python-hashing/#:~:text=La%20funci%C3%B3n%20hashing%20integrada%20de,que%20implementa%20diccionarios%20y%20conjuntos.

https://pythondiario.com/2018/06/tabla-hash-en-python.html

https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/programacion-con-estructuras-de-datos/2020/i/guia-8.pdf

https://www.hci.uniovi.es/Products/DSTool/hash/hash-queSon.html