Tarea Programada 1 Entrega 1

Análisis de Algoritmos y Estructuras de Datos

Tarea Programada 1

Entrega 1

Lucy Camacho Hoppe C01544

Modelo Diccionario

Un Diccionario es una colección de pares ordenados, donde cada par está compuesto por una clave única y un valor asociado a esa clave. Matemáticamente, se puede representar como un conjunto de pares (k, v), donde k es la clave y v es el valor asociado a k. La principal característica de un Diccionario es que no permite claves duplicadas, es decir, cada clave k es única en el Diccionario. Las operaciones básicas que se pueden realizar en un Diccionario incluyen la inserción, eliminación y búsqueda de pares por su clave.

En términos formales, un Diccionario D se puede definir como un conjunto de pares (k, v) tal que:

1. k pertenece a un conjunto K de claves posibles.
2. v pertenece a un conjunto V de valores posibles.
3. Para cada k en K, existe exactamente un v en V tal que (k, v) está en D.

Esta estructura permite una rápida recuperación de valores basada en claves, lo que la hace esencial en muchas aplicaciones de programación, como bases de datos, caches y más.

Operadores

Init(D)	**Destroy**(D)
Insert(e, D)	Delete(e, D)
Member(e, D):bool	

Estructuras de Datos

Lista Ordenada

Una Lista Ordenada es una colección lineal de elementos en la que cada elemento tiene una posición específica, y estos están dispuestos según un criterio de orden predeterminado. A diferencia de una lista común, donde los elementos pueden estar en cualquier orden, en una Lista Ordenada los elementos se insertan en la posición que les corresponde según el criterio de ordenamiento definido, ya sea ascendente o descendente.

Las características y ventajas principales de una Lista Ordenada son:

1. **Ordenamiento Automático:** Al insertar un nuevo elemento, este se coloca automáticamente en la posición correcta según el criterio de ordenamiento.
2. **Búsqueda Eficiente:** Dado que los elementos están ordenados, se pueden emplear algoritmos de búsqueda eficientes, como la búsqueda binaria, para encontrar un elemento rápidamente.

3. **Claridad y Consistencia:** Al mantener los elementos en un orden específico, se facilita la comprensión y el procesamiento de los datos.
4. **Eliminación Simplificada:** Al eliminar un elemento, no es necesario reordenar la lista, ya que los elementos restantes aún conservan el orden correcto.

Sin embargo, es importante mencionar que la inserción en una Lista Ordenada puede ser menos eficiente que en una lista común, debido al tiempo que se toma encontrar la posición correcta para el nuevo elemento. A pesar de esto, las ventajas en términos de búsqueda y organización suelen superar este inconveniente en muchas aplicaciones prácticas.

Lista Simplemente Enlazada

La implementación de una Lista Ordenada mediante el uso de una Lista Enlazada Simplemente Enlazada se basa en la estructura fundamental de nodos enlazados, pero con la adición de un criterio de ordenamiento al insertar elementos. A continuación, se describe la implementación:

Estructura Básica:

1. **Nodo:** Cada nodo de la lista enlazada tiene dos componentes:
   - **Dato:** El valor o elemento que se almacena en el nodo.
   - **Siguiente:** Un puntero que apunta al siguiente nodo en la lista.
2. **Cabeza (head):** Un puntero que apunta al primer nodo de la lista. Si la lista está vacía, este puntero es nulo.

Inserción en una Lista Ordenada:

Para insertar un elemento en una Lista Ordenada implementada con una Lista Enlazada Simplemente Enlazada, se sigue el siguiente procedimiento:

1. **Crear un nuevo nodo:** Se crea un nuevo nodo y se asigna el valor a insertar al campo de dato del nodo.
2. **Buscar la posición correcta:** Se recorre la lista desde la cabeza, comparando el valor a insertar con los valores de los nodos existentes, hasta encontrar la posición correcta o hasta que el puntero siguiente sea nulo.
3. **Insertar el nodo:** Una vez encontrada la posición, se ajustan los punteros para insertar el nuevo nodo en la posición correcta.

Ventajas de esta Implementación:

1. **Flexibilidad:** La memoria se asigna dinámicamente, lo que significa que no es necesario definir un tamaño máximo para la lista.
2. **Inserción y Eliminación Eficientes:** A diferencia de las listas basadas en arreglos, no es necesario mover elementos para insertar o eliminar nodos.

Desventajas:

1. **Uso de Memoria Adicional:** Cada nodo requiere memoria adicional para el puntero "Siguiente".
2. **Acceso Secuencial:** A diferencia de los arreglos, no se puede acceder directamente a un elemento por su índice. Es necesario recorrer la lista desde la cabeza hasta el elemento deseado.

Arreglo

Una Lista Ordenada implementada con un arreglo es esencialmente un arreglo en el que los elementos están dispuestos en un orden específico, ya sea ascendente o descendente. A medida que se insertan o eliminan elementos, se mantiene este orden.

Pasos para la Implementación:

1. **Inicialización:** Se crea un arreglo de un tamaño determinado. Se puede utilizar un contador para llevar un registro del número de elementos en la lista.
2. **Inserción:**
   - Se busca la posición correcta donde debe insertarse el nuevo elemento para mantener el orden.
   - Todos los elementos a la derecha de esa posición se desplazan una posición hacia la derecha para hacer espacio para el nuevo elemento.
   - Se inserta el nuevo elemento en la posición correcta.
3. **Búsqueda:**
   - Dado que el arreglo está ordenado, se puede utilizar la búsqueda binaria para encontrar un elemento. Esto divide repetidamente el arreglo por la mitad hasta que se encuentra el elemento o se determina que no está presente.
4. **Eliminación:**
   - Se busca el elemento a eliminar.
   - Una vez encontrado, todos los elementos a la derecha de ese elemento se desplazan una posición hacia la izquierda para llenar el espacio vacío.
5. **Acceso:** Se puede acceder directamente a cualquier elemento del arreglo mediante su índice, lo que proporciona un acceso en tiempo constante.

Ventajas:

- **Acceso Rápido:** El acceso directo mediante índices permite recuperar cualquier elemento en tiempo constante.
- **Búsqueda Eficiente:** La naturaleza ordenada del arreglo permite la búsqueda binaria, que es más rápida que la búsqueda lineal.

Desventajas:

- **Inserción y Eliminación Costosas:** Estas operaciones requieren desplazar elementos, lo que puede ser costoso en términos de tiempo, especialmente si el arreglo es grande.
- **Tamaño Fijo:** A menos que se implemente un arreglo dinámico, el tamaño del arreglo es fijo, lo que puede llevar a problemas de espacio si la lista crece más allá de la capacidad del arreglo.

Tabla Hash

Una Tabla Hash, también conocida como "hash map" o "diccionario", es una estructura de datos que permite almacenar pares de clave-valor de manera eficiente. Utiliza una función de hash para convertir la clave en un índice en un arreglo, lo que permite acceder rápidamente al valor asociado.

Características Principales:

1. **Función de Hash:** Transforma las claves en índices. Su objetivo es distribuir las claves de manera uniforme en el arreglo para evitar colisiones (situaciones donde dos claves diferentes tienen el mismo índice).
2. **Arreglo o Lista de Cubetas:** Cada posición del arreglo, conocida como cubeta, puede contener uno o más pares de clave-valor.
3. **Manejo de Colisiones:** Cuando dos claves se mapean al mismo índice, se necesita una estrategia para manejar esta situación. Las técnicas comunes incluyen encadenamiento (donde cada cubeta contiene una lista de elementos) y direccionamiento abierto (donde se busca la siguiente cubeta disponible).

Utilidad para Implementar el Modelo Diccionario:

1. **Acceso Rápido:** Gracias a la función de hash, las Tablas Hash ofrecen tiempos de acceso, inserción y eliminación cercanos a O(1) en promedio.
2. **Flexibilidad:** Pueden almacenar cualquier tipo de clave-valor, y la clave no necesita ser numérica.
3. **Manejo Eficiente de Grandes Cantidades de Datos:** Son especialmente útiles cuando se necesita acceder o insertar datos en grandes conjuntos de manera rápida.

Tabla Hash Abierta

Una Tabla Hash Abierta es una variante de la Tabla Hash que maneja las colisiones mediante el uso del direccionamiento abierto. En lugar de almacenar una lista de elementos en cada cubeta (como en el encadenamiento), la Tabla Hash Abierta busca cubetas alternativas en el arreglo hasta encontrar una posición libre donde se pueda insertar el par clave-valor.

Características Principales:

1. **Direccionamiento Abierto:** Cuando ocurre una colisión (dos claves diferentes que se mapean al mismo índice), la tabla busca otra cubeta en el arreglo para almacenar el par clave-valor. Esto se logra mediante técnicas como el sondeo lineal, el sondeo cuadrático o la dispersión doble.
2. **Tamaño Fijo:** A diferencia del encadenamiento, donde cada cubeta puede contener una lista de elementos, en el direccionamiento abierto, el arreglo tiene un tamaño fijo, lo que puede requerir redimensionar la tabla si se llena.
3. **Menor Uso de Memoria:** Al no utilizar listas adicionales para manejar colisiones, la Tabla Hash Abierta generalmente utiliza menos memoria que las tablas con encadenamiento.

Ventajas:

1. **Acceso Más Rápido:** Al no tener que recorrer listas en las cubetas, el acceso a los elementos suele ser más rápido, especialmente si la tasa de colisión es baja.
2. **Eficiencia en Memoria:** Al no almacenar punteros adicionales para listas enlazadas, se ahorra memoria.

Desventajas:

1. **Manejo de Colisiones:** Si la tabla se llena o tiene una alta tasa de colisión, el rendimiento puede degradarse, ya que se requiere más tiempo para encontrar una cubeta libre.
2. **Redimensionamiento:** Si la tabla se llena, puede ser necesario redimensionarla y rehashing todos los elementos, lo cual puede ser costoso en términos de tiempo.

Funcion Hash: Método de la División

```
int hash_func(key, table_size)
{
        return key % table_size
}
```

Esta función hash es simple y utiliza el método de la división. Toma una clave numérica y el tamaño de la tabla hash, y devuelve el resto de la división de la clave por el tamaño de la tabla. Por ejemplo, si la clave es 27 y el tamaño de la tabla es 10, la función devolverá 7.

Aleatoriedad de la Función Hash:

La aleatoriedad en una función hash se refiere a cómo distribuye uniformemente las claves en las cubetas disponibles, minimizando las colisiones. Una buena función hash debe dispersar las claves de manera que cada cubeta tenga aproximadamente el mismo número de claves.

En el caso del método de la división:

- **Ventajas:** Es rápido y simple. Funciona bien si el tamaño de la tabla es un número primo que no está cerca de una potencia de 2 o 10, ya que esto ayuda a garantizar una distribución más uniforme.
- **Desventajas:** No es inherentemente aleatorio, ya que depende del tamaño de la tabla y de las propias claves. Si las claves tienen un patrón específico y el tamaño de la tabla no se elige adecuadamente, podría no distribuir las claves uniformemente.

La verdadera aleatoriedad es difícil de lograr en las funciones hash, pero el objetivo es acercarse lo más posible a ella para obtener una distribución uniforme. En la práctica, se utilizan funciones hash más complejas y bien estudiadas, como MurmurHash o CityHash, que están diseñadas para ofrecer una buena aleatoriedad y distribución uniforme de las claves en una variedad de escenarios.

Redistribución

1. **Determinar el Nuevo Tamaño:** Primero, se decide el tamaño de la nueva tabla hash. Por lo general, se elige un tamaño que sea al menos el doble de la tabla original para acomodar más elementos en el futuro y reducir la necesidad de redistribuciones frecuentes.
2. **Crear una Nueva Tabla:** Se inicializa una nueva tabla hash vacía con el tamaño determinado.
3. **Transferir Elementos:** Se recorre cada cubeta de la tabla hash original. Para cada par clave-valor en la tabla original, se calcula su nuevo índice en la nueva tabla usando la función hash y se inserta en la nueva tabla. Es esencial recalcula el índice porque el tamaño de la tabla ha cambiado.
4. **Reemplazar la Tabla Original:** Una vez que todos los elementos han sido transferidos, la tabla hash original se descarta o se libera, y la nueva tabla toma su lugar.

Tiempo de Duración:

El tiempo de duración del proceso de redistribución es proporcional al número de elementos en la tabla original, ya que cada elemento debe ser procesado y reinsertado en la nueva tabla. En términos de complejidad algorítmica, el proceso de redistribución es O(n), donde n es el número de elementos en la tabla hash.