Búsqueda

Jorge Mario Londoño Peláez & Varias AI

April 2, 2025

1 Definición del problema de la búsqueda

El problema de la búsqueda es fundamental en la informática y se presenta en una variedad de aplicaciones. En su esencia, el problema consiste en encontrar un elemento específico dentro de un conjunto de datos. Este conjunto de datos puede ser una lista, un arreglo, una base de datos, o incluso la web. Formalmente, dado un conjunto de elementos y una llave de búsqueda, el objetivo es determinar si existe un elemento en el conjunto que coincida con la llave, y si existe, recuperar información asociada a ese elemento.

Algunos ejemplos de aplicaciones del problema de búsqueda son:

- Buscar un contacto en la aplicación de contactos del celular.
- Buscar un contenido en la web.
- Encontrar un estudiante por su `ID`.
- Buscar registros en una base de datos.

1.1 La tabla de símbolos

Una **tabla de símbolos** (también conocida como diccionario o mapa) es un Tipo de Dato Abstracto (TDA) que modela el problema de la búsqueda de manera eficiente. Una tabla de símbolos almacena un conjunto de pares llave-valor, donde cada llave es única y está asociada a un valor específico. Las tablas de símbolos permiten realizar operaciones de búsqueda, inserción y eliminación de elementos basándose en su llave.

Existen dos tipos principales de tablas de símbolos:

- **Tablas de símbolos no ordenadas:** En este tipo de tabla, las llaves no tienen un orden específico. La inserción de nuevos elementos se realiza de manera sencilla, pero la búsqueda puede requerir una exploración exhaustiva de todos los elementos.
- **Tablas de símbolos ordenadas:** En este tipo de tabla, las llaves se mantienen ordenadas según lo defina la comparación de llaves. Esto permite realizar búsquedas más eficientes, como la búsqueda binaria, pero la inserción y eliminación de elementos pueden ser más costosas debido a la necesidad de mantener el orden.

1.1.1 API de la tabla de símbolos no ordenada

La Application Programming Interface (API) básica de una tabla de símbolos no ordenada incluye las siguientes operaciones:

- `put(llave, valor)`: Inserta un nuevo par llave-valor en la tabla.
- `get(llave)`: Busca la llave en la tabla y devuelve el valor asociado. Si la llave no existe, devuelve un valor nulo o un valor por defecto.
- `delete(llave)`: Elimina el par llave-valor asociado a la llave dada.
- `contains(llave)`: Verifica si la llave existe en la tabla.
- `isEmpty()`: Verifica si la tabla está vacía.
- `size()`: Devuelve el número de pares llave-valor en la tabla.
- `keys()`: Devuelve un iterador sobre las llaves de la tabla.

1.1.2 API de la tabla de símbolos ordenada

Además de las operaciones de la tabla de símbolos no ordenada, la API de una tabla de símbolos ordenada incluye las siguientes operaciones:

- `min()`: Devuelve la llave mínima en la tabla.
- `max()`: Devuelve la llave máxima en la tabla.
- `floor(llave)`: Devuelve la llave más grande en la tabla que es menor o igual a la llave dada.
- `ceiling(llave)`: Devuelve la llave más pequeña en la tabla que es mayor o igual a la llave dada.
- `rank(llave)`: Devuelve el número de llaves en la tabla que son menores que la llave dada.
- `select(k)`: Devuelve la llave de rango k en la tabla (es decir, la llave que tiene k llaves menores que ella).
- `size(lo,hi)`: Devuelve el número de pares llave-valor cuya llave se encuentra entre lo y hi inclusive.
- `keys(lo,hi)`: Devuelve un iterador sobre las llaves de la tabla que se encuentran entre lo y hi inclusive.

1.1.3 Convenciones sobre llaves y valores

- No se permiten llaves nulas.
- No pueden existir llaves duplicadas.
- Si un valor es nulo, indica que la pareja llave-valor ha sido eliminada.

2 La búsqueda secuencial

2.1 Tabla de símbolos por lista simple no ordenada

Una forma sencilla de implementar una tabla de símbolos es utilizando una lista enlazada simple no ordenada. En esta representación, cada elemento de la lista contiene un par llave-valor.

2.1.1 Estructura del nodo

Cada nodo de la lista enlazada contiene los siguientes campos:

- `llave`: La llave del par llave-valor.
- `valor`: El valor asociado a la llave.
- `siguiente`: Un puntero al siguiente nodo en la lista.

2.1.2 Implementación de las operaciones

A continuación, se describe la implementación de las operaciones principales de la tabla de símbolos utilizando pseudocódigo:

put(llave, valor)

```
1:  procedure PUT(llave, valor)
2:      actual ← cabeza
3:      while actual ≠ null do
4:          if actual.llave = llave then
5:              actual.valor ← valor                ▷ Actualizar el valor si la llave ya existe
6:              return
7:          end if
8:          actual ← actual.siguiente
9:      end while
10:     nuevo ← Nodo(llave, valor)                  ▷ Crear un nuevo nodo
11:     nuevo.siguiente ← cabeza                    ▷ Apuntar el nuevo nodo al inicio de la lista
12:     cabeza ← nuevo                              ▷ Actualizar la cabeza de la lista
13: end procedure
```

get(llave)

```
1:  procedure GET(llave)
2:      actual ← cabeza
3:      while actual ≠ null do
4:          if actual.llave = llave then
5:              return actual.valor                 ▷ Devolver el valor asociado a la llave
6:          end if
7:          actual ← actual.siguiente
8:      end while
9:      return null                                 ▷ Devolver null si la llave no se encuentra
10: end procedure
```

delete(llave)

```
1:  procedure DELETE(llave)
2:      actual ← cabeza
3:      anterior ← null
4:      while actual ≠ null do
5:          if actual.llave = llave then
6:              if anterior = null then
7:                  cabeza ← actual.siguiente                ▷ Eliminar el nodo de la cabeza
8:              else
9:                  anterior.siguiente ← actual.siguiente    ▷ Eliminar el nodo del medio o final
10:             end if
11:             return
12:         end if
13:         anterior ← actual
14:         actual ← actual.siguiente
15:     end while
16: end procedure
```

2.1.3 Eficiencia de las operaciones

A continuación, se presenta el orden de crecimiento del tiempo de ejecución de las operaciones principales de la tabla de símbolos implementada con una lista enlazada simple no ordenada:

- `put(llave, valor)`: $O(n)$ - Tiempo lineal, ya que en el peor de los casos se debe recorrer toda la lista para encontrar la llave o insertar al principio.
- `get(llave)`: $O(n)$ - Tiempo lineal, ya que en el peor de los casos se debe recorrer toda la lista para encontrar la llave.
- `delete(llave)`: $O(n)$ - Tiempo lineal, ya que en el peor de los casos se debe recorrer toda la lista para encontrar la llave a eliminar.

Conclusión: La implementación de la tabla de símbolos con una lista simple no ordenada es sencilla, pero ineficiente para grandes conjuntos de datos debido a que las operaciones de búsqueda y eliminación tienen un tiempo de ejecución lineal.

3 La búsqueda binaria

3.1 Representación de la tabla de símbolos por medio de arreglo ordenado

Una tabla de símbolos se puede implementar utilizando dos arreglos paralelos: uno para las llaves y otro para los valores. Es requisito fundamental que el arreglo de llaves esté ordenado de forma ascendente. Esta organización permite aprovechar el algoritmo de búsqueda binaria para encontrar rápidamente la llave deseada.

- **Arreglo de llaves:** Almacena las llaves de la tabla de símbolos en orden ascendente.
- **Arreglo de valores:** Almacena los valores correspondientes a cada llave en la misma posición que su llave en el arreglo de llaves.

Es importante destacar que las llaves deben ser comparables, es decir, debe existir una forma de determinar si una llave es menor, igual o mayor que otra. Esto es esencial para mantener el orden en el arreglo de llaves y para realizar la búsqueda binaria de manera eficiente.

3.2 Implementación de las operaciones de la tabla de símbolos

3.2.1 Operación rank(llave)

La operación `rank(llave)` es fundamental en la búsqueda binaria. Devuelve el número de llaves en la tabla que son menores que la llave dada. En otras palabras, indica la posición que la llave dada debería ocupar en el arreglo de llaves si estuviera presente.

```
1: procedure RANK(llave)
2:     lo ← 0
3:     hi ← N − 1                                  ▷ N es el número de llaves en la tabla
4:     while lo ≤ hi do
5:         mid ← lo + (hi − lo)/2
6:         if llave < llaves[mid] then
7:             hi ← mid − 1
8:         else if llave > llaves[mid] then
9:             lo ← mid + 1
10:        else
11:            return mid                           ▷ Se encontró la llave
12:        end if
13:    end while
14:    return lo          ▷ No se encontró la llave, devolver la posición donde debería estar
15: end procedure
```

3.2.2 Operación get(llave)

La operación `get(llave)` busca la llave en el arreglo de llaves utilizando la búsqueda binaria. Si la llave se encuentra, devuelve el valor asociado en el arreglo de valores. Si la llave no se encuentra, devuelve nulo.

```
1: procedure GET(llave)
2:     i ← rank(llave)
3:     if i < N and llaves[i] = llave then
4:         return valores[i]
5:     else
6:         return nulo                              ▷ La llave no se encontró
7:     end if
8: end procedure
```

3.2.3 Operación put(llave, valor)

La operación `put(llave, valor)` inserta un nuevo par llave-valor en la tabla de símbolos. Si la llave ya existe, actualiza el valor asociado. Si la llave no existe, inserta la llave y el valor en la posición correcta para mantener el orden del arreglo de llaves.

```
1: procedure PUT(llave, valor)
2:     i ← rank(llave)
3:     if i < N and llaves[i] = llave then
```

```
        valores[i] ← valor                    ▷ Actualizar el valor si la llave ya existe
        return
    end if
    Mover todas las llaves mayores que la llave dada una posición a la derecha
    Mover todos los valores mayores que la llave dada una posición a la derecha
    llaves[i] ← llave                         ▷ Insertar la nueva llave
    valores[i] ← valor                        ▷ Insertar el nuevo valor
    N ← N + 1                                 ▷ Incrementar el número de llaves en la tabla
end procedure
```

3.2.4 Operación delete(llave)

La operación `delete(llave)` elimina la llave y su valor asociado de la tabla de símbolos.

```
procedure DELETE(llave)
    i ← rank(llave)
    if i < N and llaves[i] = llave then
        Mover todas las llaves mayores que la llave dada una posición a la izquierda
        Mover todos los valores mayores que la llave dada una posición a la izquierda
        N ← N − 1                             ▷ Decrementar el número de llaves en la tabla
    end if
end procedure
```

3.3 Eficiencia de la búsqueda binaria

A continuación, se presenta el orden de crecimiento del tiempo de ejecución de las operaciones principales de la tabla de símbolos implementada con arreglos ordenados y búsqueda binaria:

- `rank(llave)`: $O(\lg n)$ - Tiempo logarítmico, debido a la búsqueda binaria.
- `get(llave)`: $O(\lg n)$ - Tiempo logarítmico, debido a la búsqueda binaria.
- `put(llave, valor)`: $O(n)$ - Tiempo lineal, ya que en el peor de los casos se deben mover todos los elementos del arreglo para insertar la nueva llave.
- `delete(llave)`: $O(n)$ - Tiempo lineal, ya que en el peor de los casos se deben mover todos los elementos del arreglo para eliminar la llave.

Conclusión: La implementación de la tabla de símbolos con arreglos ordenados y búsqueda binaria ofrece una mejora significativa en el tiempo de búsqueda en comparación con la búsqueda secuencial. Sin embargo, las operaciones de inserción y eliminación siguen siendo costosas debido a la necesidad de mantener el orden del arreglo.