Lab06

miércoles, 11 de junio de 2025 17:35

Reasignación lineal

→ Búsqueda:

```
public Cliente buscar(String clave) {
    int hash = generarHash(clave, capacidad); → O(4K+2)
    int original = hash; → O(1)
    while (tabla[hash] != null && !tabla[hash].getClaveHash().equals(clave)) {
        hash = (hash + 1) % capacidad; → O(3)
        if (hash == original) return null; → O(1)
    }
    return tabla[hash];
}
```

O(4) x n veces → O(4n)

```
public static int generarHash(String clave, int tamanoTabla) {
    int hash = 0; → O(1)
    for (char c : clave.toCharArray()) {
        hash = (hash * 31 + c) % tamanoTabla; → O(4)
    }
    return hash;
}
```

K veces

el for se repetirá por K char que tenga la clave

- función tiempo: 4K + 1
- notación asintótica: O(K)

En el peor de los casos, el bucle while se recorre n veces, el tamaño de la tabla

- función tiempo: 4n + 4K + 3
- notación asintótica: O(n)

→ Inserción

```
public void insertar(Cliente cliente) {
    int hash = generarHash(cliente.getClaveHash(), capacidad); → O(4K+2)
    int original = hash; → O(1)
    while (tabla[hash] != null) { → O(1)
        hash = (hash + 1) % capacidad; → O(3)
        if (hash == original) return; → O(1)
    }
    tabla[hash] = cliente; → O(1)
}
```

O(5) → n veces → O(5n)

En el peor de los casos, se recorre toda la tabla buscando un espacio libre.

- función tiempo: 5n + 4K + 5
- notación: O(n)

Encadenamiento con ABB

→ Insertar

```
public void insertar(Cliente cliente) {                    → O(4K+1)
    int hash = HashLineal.generarHash(cliente.getClaveHash(), capacidad);
    tabla[hash].insertar(cliente); → O(m)
}
```

- f. tiempo: 4K + 1 + m
- notación: O(K + m)

```
private NodoArbol insertarRecursivo(NodoArbol nodo, Cliente cliente) {
    if (nodo == null) return new NodoArbol(cliente); → O(1)
    if (cliente.getClaveHash().compareTo(nodo.cliente.getClaveHash()) < 0)   → O(1)
        nodo.izquierda = insertarRecursivo(nodo.izquierda, cliente); → O(1)
    else
        nodo.derecha = insertarRecursivo(nodo.derecha, cliente); → O(1)
    return nodo;
}
```

max = O(1+1) = O(2)

La recursividad en el mejor caso será cuando está balanceado O(log m), en el peor de los casos O(m) → m: # elementos en la lista/arbol de colisión

- f. tiempo: 3m
- notación: O(m)

→ Búsqueda:

```
public Cliente buscar(String clave) {
    int hash = HashLineal.generarHash(clave, capacidad);
    return tabla[hash].buscar(clave);
}
```

Mismo análisis que con la inserción

- f. tiempo: 4K + 1 + m
- notación: O(K + m)

```
private Cliente buscarRecursivo(NodoArbol nodo, String clave) {
    if (nodo == null) return null;
    if (nodo.cliente.getClaveHash().equals(clave)) return nodo.cliente;
    if (clave.compareTo(nodo.cliente.getClaveHash()) < 0)
        return buscarRecursivo(nodo.izquierda, clave);
    else
        return buscarRecursivo(nodo.derecha, clave);
}
```