3. Estructura de Datos: HashMaps (`unordered_map`)

Una **tabla hash** o **HashMap** es una estructura de datos que almacena pares clave-valor, permitiendo acceso eficiente a los datos mediante una clave unica. En C++, la implementacion estandar es `unordered_map`, parte de la STL.

Internamente, utiliza una tabla hash que convierte cada clave en una posicion de memoria donde se almacena el valor correspondiente. Esto permite tiempos de acceso, insercion y eliminacion promedio en $\mathcal{O}(1)$.

Caracteristicas de `unordered_map`

- Almacena datos como pares clave-valor: `clave` $\rightarrow$ `valor`.
- Las claves son unicas; si se inserta una clave duplicada, se sobreescribe su valor.
- El orden de los elementos no esta definido.
- Acceso e insercion promedio en tiempo constante gracias al hashing.

Funciones Utiles de `unordered_map`

- `insert({clave, valor})` — Inserta un par clave-valor.
- `mapa[clave]` — Accede al valor asociado a la clave (crea la clave si no existe).
- `at(clave)` — Accede al valor asociado, lanza excepcion si no existe.
- `erase(clave)` — Elimina el par asociado a la clave.
- `count(clave)` — Devuelve 1 si existe, 0 si no.
- `size()` — Numero de elementos.
- `clear()` — Elimina todos los pares almacenados.
- `find(clave)` — Devuelve iterador al elemento, o `end()` si no existe.

Ejemplo Basico de Uso

Listing 1: Uso de unordered_map

```
#include <iostream>
#include <unordered_map>

using namespace std;

int main() {
    unordered_map<string, int> edades;
```

```cpp

    // Insertar elementos
    edades["Ana"] = 25;
    edades["Luis"] = 30;
    edades["Maria"] = 28;

    // Acceder a valores
    cout << "Edad de Ana: " << edades["Ana"] << endl;

    // Recorrer el hashmap
    cout << "Todas las edades:" << endl;
    for (auto par : edades) {
        cout << par.first << ": " << par.second << endl;
    }

    // Verificar existencia
    if (edades.count("Luis")) {
        cout << "Luis esta en el mapa." << endl;
    }

    // Eliminar
    edades.erase("Ana");

    // Tamano final
    cout << "Tamano del mapa: " << edades.size() << endl;

    return 0;
}
```

Complejidad de Operaciones

- **Acceso, insercion y eliminacion:** $\mathcal{O}(1)$ en promedio, $\mathcal{O}(n)$ en el peor caso.
- **Busqueda:** $\mathcal{O}(1)$ promedio.

Ventajas y Desventajas

Ventajas:

- Acceso rapido por clave.
- Sintaxis sencilla y clara.
- Inserciones y eliminaciones eficientes.

Desventajas:

- No mantiene ningun orden.

- El peor caso puede ser lineal si hay muchas colisiones (Por el mismo motivo que los sets).
- Requiere funciones hash adecuadas para tipos personalizados, aunque normalmente vamos a usar datos primitivos, que no las requieren.

Aplicaciones Comunes

- Contadores de frecuencia.
- Indexacion de datos por identificador o clave.
- Optimizacion de algoritmos con memoizacion (de forma recursiva).
- Representacion de grafos con listas de adyacencia.