

Estructura de Datos II

David Concha Gómez

Asignatura obligatoria

Segundo cuatrimestre

Créditos: 6

[Moodle de la asignatura](#)

[Guía docente](#)

Índice

- Conjuntos y mapas
- Tablas Hash
 - Introducción
 - Funciones Hash
 - Códigos Hash
 - Función de compresión
 - Colisiones
- Diccionarios
 - unordered_multiset y unordered_multimap

Índice

- **Conjuntos y mapas**
- **Tablas Hash**
 - Introducción
 - Funciones Hash
 - Códigos Hash
 - Función de compresión
 - Colisiones
- **Diccionarios**
 - `unordered_multiset` y `unordered_multimap`

Conjuntos y Mapas

- Un conjunto almacena elementos que pueden ser localizados eficientemente $O(1)$.
- Los elementos no se repiten y no tienen orden, no hay un primero, no hay un último.
- Ya los conocéis como arrays de booleanos.

Conjuntos y Mapas

- Un mapa es similar a un conjunto, pero guarda parejas clave:valor.
- Solo hay una entrada por clave, aunque varias claves pueden tener el mismo valor.
- La búsqueda por clave es eficiente.

Conjuntos y Mapas

- Son muy usados cuando se necesita indexar por índices no enteros.
- También se usan con índices enteros cuando estos no son consecutivos .
- Se implementan mediante tablas hash.

Conjuntos y Mapas

- Operaciones generales:
 - **isEmpty** y **size**?
 - **T& operator[]**(k): Devuelve una referencia al valor de clave “k”. Si no existe devuelve un elemento vacío.
 - **size_t erase**(k): Elimina el elemento con clave “k” si existe. Devuelve cuantos elementos se han borrado (0 o 1).
 - **bool contains**(k): Devuelve si existe una entrada con clave “k”.
 - **begin, end** : Iteradores a los valores de la estructura.

```
std::unordered_map<int, double> m = {  
    {3, 1.0}, {42, 4.2 }, {37, 3.5}};
```

```
m[9] = 0.0;
```

```
m.erase(42);
```

```
for (const auto& p: m) {  
    std::cout << "Key: " << p.first;  
    std::cout << " Value: " << p.second;  
    std::cout << "\n";  
}
```

Índice

- Conjuntos y mapas
- **Tablas Hash**
 - Introducción
 - Funciones Hash
 - Códigos Hash
 - Función de compresión
 - Colisiones
- **Diccionarios**
 - unordered_multiset y unordered_multimap

Tablas Hash

- Las tablas hash son una forma eficiente de implementar conjuntos y mapas.
 - Se dispone de una función hash que permite convertir la clave en un entero utilizado para indexar.
 - Las operaciones de inserción, borrado y consulta son $O(1)$ amortizado.
- Una tabla hash se componen de:
 - Un almacén para elementos, típicamente un array llamado bucket.
 - Una función hash $i = h(k)$
- Internamente, se almacenan los pares (k, v) en la posición $i = h(k)$.

Tablas Hash

- Si las claves son enteros en el rango de la tabla, la implementación es directa.

0	E			2	A			4	B	5	D
---	---	--	--	---	---	--	--	---	---	---	---

Clave
 Valor

- Problemas:
 - Si la tabla es menor que la clave máxima (DNI, número de expediente, ...).
 - Si las claves no son enteros (nombres, código alfanumérico, ...)

Tablas Hash

- La función hash “mapea” las claves de un determinado tipo a enteros en un determinado rango.
 - Para variables enteras, por ejemplo: $h(k) = k \% N$
- El valor se conoce como clave dispersa (hash value).

☐ Clave
☐ Valor

23490	*			09562	*	36893	*			12385	*
-------	---	--	--	-------	---	-------	---	--	--	-------	---

`m[09562] = Point{};`

`m[36893] = Point{};`

`m[12385] = Point{};`

`m[23490] = Point{};`

Tablas Hash

□ Clave
□ Valor

23490	*			09562	*	36893	*			12385	*
-------	---	--	--	-------	---	-------	---	--	--	-------	---

- Si dos claves presentan el mismo hash value se produce una **colisión**.
- Es importante seleccionar una buena función de hash:
 - Que minimice las colisiones
 - Que sea fácil de calcular

m[09562] = Point{};

m[36893] = Point{};

m[12385] = Point{};

m[23490] = Point{};

m[10233] = Point{};

Índice

- Conjuntos y mapas
- **Tablas Hash**
 - Introducción
 - **Funciones Hash**
 - Códigos Hash
 - Función de compresión
 - Colisiones
- **Diccionarios**
 - unordered_multiset y unordered_multimap

Funciones Hash

- Las funciones hash se pueden ver como una composición de dos funciones:
 - Código hash:
$$h_H: \text{Key} \rightarrow \text{int}$$
 - Función de compresión:
$$h_C: \text{int} \rightarrow [0, N-1]$$
- El código hash transforma la clave en un entero.
 - Es dependiente de la clave y el problema.
- La función de compresión limita el entero a un rango aceptable para la tabla.
 - Es dependiente de la tabla hash.

Código hash

- Deben ser consistentes:
 - Dos claves que se consideren iguales deben presentar el mismo código hash.
- Se mapea un objeto a un valor entero.
 - En general definir un buen código hash es complicado.
- Si se quiere que un objeto pueda actuar como clave será necesario definir una función hash adecuada.

Código hash

- Ejemplos:

- Dirección de memoria del objeto.
- Interpretación de los bits de la clave como un entero.
- Dividir los bits en bloques, sumarlos e interpretarlos como un entero
- Concatenar los atributos de un objeto y proceder con algún otro método.
- Acumulación polinómica:
 - Dividir los bits de la clave en componentes de la misma longitud.

$$k = \{k_0, k_1, k_2, \dots, k_{n-1}\}$$

- Dada una constante a calcular el polinomio:

$$k_0 + k_1a + k_2a^2 + \dots + k_{n-1}a^{n-1}$$

$$\begin{aligned} p_0(a) &= k_{n-1} \\ p_i(a) &= k_{n-i-1} + ap_{i-1}(a) \end{aligned}$$

- Para $a = 33$ se generan unas 6 colisiones para 50.000 palabras.

Funciones de compresión

- Ejemplos:

- División:

$$h_c(y) = y \bmod N$$

- El tamaño de la tabla N suele ser un número primo.
- Ejemplo: Para {200, 205, 210, ... 600}
 - Si N = 100 cada clave colisiona con otras 4
 - Si N = 101 no hay colisión

- MAD - Multiplicar, sumar (Add) y dividir

$$h_c(y) = ((ay + b) \bmod p) \bmod N$$

- p es un primo mayor que N.
- a y b son enteros no negativos aleatorios en el rango [0, p-1]

Índice

- Conjuntos y mapas
- **Tablas Hash**
 - Introducción
 - Funciones Hash
 - Códigos Hash
 - Función de compresión
 - **Colisiones**
 - Encadenamiento separado
 - Direccionamiento abierto
- **Diccionarios**
 - `unordered_multiset` y `unordered_multimap`

Colisiones

- Dos claves pueden coincidir en la misma dirección.
 - Principio del palomar
- A este fenómeno se le denomina colisión.
- Existen técnicas efectivas de resolver los conflictos:
 - Encadenamiento separado.
 - Direccionamiento abierto.

Encadenamiento Separado

- Cada celda de la tabla tiene guarda todas las entradas con la misma clave hash.
 - La clave del par clave:valor es distinta.
- La forma de guardar las distintas entradas es utilizar una lista.
- La lista añade complejidad a las operaciones del conjunto/mapa, pero, idealmente, solo hay un elemento por lista.

```
void operator[](K k, T e) {  
    auto& l = m[h(k)];  
    if (contains(k)) {  
        l.remove(k);  
    }  
    l.insert({k, e});  
};  
  
bool contains(K k) {  
    return m[h(k)].size();  
}
```

Direccionamiento abierto

- Cuando una entrada colisiona, se guarda en otra posición.
 - Prueba lineal: se inserta en la siguiente posición libre.

$$h(k, p) = h_c(k) + p$$

p es la prueba (probe), cuantas veces se ha llamado a la función hash para resolver una colisión. Inicialmente 0.

Direcccionamiento abierto

- Cuando una entrada colisiona, se guarda en otra posición.
 - Prueba lineal: se inserta en la siguiente posición libre.

$$h(k, p) = h_c(k) + p$$

Clúster primario.

- Prueba cuadrática: Se desplaza dando saltos definidos por dos constantes (c_1 y c_2)

$$h(k, p) = h_c(k) + c_1p + c_2p^2$$

p es la prueba (probe), cuantas veces se ha llamado a la función hash para resolver una colisión. Inicialmente 0.

Direcccionamiento abierto

- Cuando una entrada colisiona, se guarda en otra posición.
 - Prueba lineal: se inserta en la siguiente posición libre.

$$h(k, p) = h_c(k) + p$$

Clúster primario.

p es la prueba (probe), cuantas veces se ha llamado a la función hash para resolver una colisión. Inicialmente 0.

- Prueba cuadrática: Se desplaza dando saltos definidos por dos constantes (c_1 y c_2)

$$h(k, p) = h_c(k) + c_1p + c_2p^2$$

Clúster secundario.

- Hashing doble:

$$h(k, p) = (h_c(k) + d(k)p) \bmod N$$

$$d(k) = q - (h_c(k) \bmod q)$$

Tablas Hash

- Factor de carga (λ).

$$\lambda = \text{ceil}(n / N)$$

- Indica la ocupación del array interno.
 - Es importante que sea menor que 1.
 - Encadenamiento separado < 0.75
 - Direcccionamiento abierto < 0.5
-
- Si no se cumple habrá que redimensionar la tabla:
 - Añadir más buckets
 - Definir una nueva función de hash para adaptarse al nuevo tamaño
 - Rehasing de las entradas ya existentes.

Tablas Hash

- La complejidad en el peor caso es $O(n)$ pero se considera $O(1)$ amortizada.
 - Solo ocurrirá si todas las claves colisionan.
- El factor de carga afecta al rendimiento.
- Aplicaciones:
 - Pequeñas BBDD
 - Procesamiento de texto
 - Caché para juegos y navegadores

Índice

- Conjuntos y mapas
- Tablas Hash
 - Introducción
 - Funciones Hash
 - Códigos Hash
 - Función de compresión
 - Colisiones
 - Encadenamiento separado
 - Direccionamiento abierto
- **Diccionarios**
 - `unordered_multiset` y `unordered_multimap`

Diccionarios

- Existe una variante de los conjuntos y mapas que permite claves repetidas.
- En c++ se llaman `unordered_multiset` y `unordered_multimap`.
- Cuando una estructura guarda claves:valor y permite repeticiones (`unordered_multimap`) se le suele denominar diccionario desordenados.

Resumen

- Los conjuntos, mapas y diccionarios desordenados son eficientes en inserción, borrado y consulta.
- Para c++ la única diferencia entre un conjunto y un mapa es:
 - Si la clave y el valor son lo mismo, es un conjunto.
 - Si la clave es distinta al valor, es un mapa.
- No confundir la versión desordenada (`unordered_map` por ejemplo) con la versión ordenada (`map` por ejemplo). Los `set`, `multiset`, `map` y `multimap` los veremos en el próximo tema.

Estructura de Datos II

David Concha Gómez

Asignatura obligatoria

Segundo cuatrimestre

Créditos: 6

[Moodle de la asignatura](#)

[Guía docente](#)
