

ESTRUCTURAS DE DATOS

NOTAS SOBRE C++

STL: Contenedores asociativos

Manuel Montenegro Montes

Departamento de Sistemas Informáticos y Computación
Facultad de Informática – Universidad Complutense de Madrid



Contenedores asociativos

- Soportan mecanismos eficientes (no lineales) de búsqueda:
 - TAD Conjunto.
 - TAD Diccionario.
 - TAD Multiconjunto.




Tipos de datos asociativos en la STL

Clase	Fich. cabecera	Estructura
<code>std::set</code>	<code><set></code>	TAD Conjunto (árboles equilibrados)
<code>std::map</code>	<code><map></code>	TAD Diccionario (árboles equilibrados)
<code>std::unordered_set</code>	<code><unordered_set></code>	TAD Conjunto (tablas <i>hash</i>)
<code>std::unordered_map</code>	<code><unordered_map></code>	TAD Diccionario (tablas <i>hash</i>)

Tienen sentido los nombres de los ficheros de cabecera porque en el caso de los árboles binarios de búsqueda, estos están ordenados según la clave. Las tablas hash no tienen un orden predefinido como tal ya que cada elemento irá en la posición calculada al hacer $\text{hash}(k) \bmod \text{CAPACITY}$.

Operaciones

- Mismo nombre que las implementaciones realizadas por nosotros:
 - `insert()`
 - `erase()`
 - `operator[]`
 - `at()`
 - `empty()` / `size()`
 - `contains()` 
 - `begin()`
 - `end()`
 - `etc.`

Son las mismas operaciones que utilizábamos en árboles binarios de búsqueda.

Sobre el método `contains()`

- El método `contains()` se introduce en el estándar **C++20**.
- Soportado por Visual Studio 2019 (16.1) si se utiliza la opción `/std:c++latest`.
- Si tu compilador no soporta este estándar, se puede utilizar el método `count()`:

`int count(const K &key)` En vez de un booleano devuelve un entero. Si el entero es 1 entonces la clave está en el diccionario.

- Devuelve 1 si el elemento/clave está en el conjunto/diccionario.
- Devuelve 0 en caso contrario.

Ejemplo

```
#include <iostream>
#include <set>
```

```
using namespace std;
```

```
int main() {
```

```
    set<string> personas;
```

Para cómo usar el método count.

Utilizando el TAD conjunto de manera automática los incluye ordenados.

```
    personas.insert("Gerardo");
    personas.insert("Sergio");
    personas.insert("Guillermo");
    personas.insert("Laura");
```

```
    personas.insert("Gerardo");
```

```
    bool esta_sergio = personas.count("Sergio");
    cout << personas.size() << endl;
    cout << boolalpha << esta_sergio << endl;
```

```
    for (const string &s: personas) {
        cout << s << endl;
    }
```

```
    return 0;
```

```
}
```

4
true

Gerardo
Guillermo
Laura
Sergio

Ejemplo

```
#include <iostream>
#include <unordered_set>
```

```
using namespace std;
```

```
int main() {
```

```
    unordered_set<string> personas;
```

No los introduce de manera ordenada. Utiliza

```
    personas.insert("Gerardo");
    personas.insert("Sergio");
    personas.insert("Guillermo");
    personas.insert("Laura");
```

```
    personas.insert("Gerardo");
```

```
    bool esta_sergio = personas.count("Sergio");
    cout << personas.size() << endl;
    cout << boolalpha << esta_sergio << endl;
```

```
    for (const string &s: personas) {
        cout << s << endl;
    }
```

```
    return 0;
```

```
}
```

4
true

Laura
Guillermo
Gerardo
Sergio

Ejemplo

```
#include <iostream>
#include <map>
```

```
using namespace std;
```

```
int main() {
    map<string, int> personas;    de manera ordenada
    personas["Francisco"] = 41;
    personas["Gloria"] = 21;
    personas["Alejandra"] = 38;
    personas.insert({"Enrique", 36});

    for (auto entry: personas) {
        cout << entry.first << " → " << entry.second << endl;
    }

    return 0;
}
```

Alejandra → 38
Enrique → 36
Francisco → 41
Gloria → 21

Multiconjuntos y multidiccionarios

- Permiten claves duplicadas:

Clase	Estructura
<code>std::multiset</code>	TAD Multiconjunto (árboles equilibrados)
<code>std::multimap</code>	TAD Multidiccionario (árboles equilibrados)
<code>std::unordered_multiset</code>	TAD Multiconjunto (tablas <i>hash</i>)
<code>std::unordered_multimap</code>	TAD Multidiccionario (tablas <i>hash</i>)

Al permitir claves repetidas NO se cuando sería útil utilizar estos tipos de TAD.

Ejemplo

```
#include <iostream>
#include <set>

using namespace std;

int main() {
    multiset<string> personas;

    personas.insert("Gerardo");
    personas.insert("Sergio");
    personas.insert("Guillermo");
    personas.insert("Belén");
    personas.insert("Gerardo");

    cout << personas.size() << endl;
    cout << personas.count("Patricia") << endl;
    cout << personas.count("Gerardo") << endl;

    for (const string &s: personas) {
        cout << s << endl;
    }

    return 0;
}
```

5
0
2

Belén
Gerardo
Gerardo
Guillermo
Sergio