

TALLER PROGCOMP: TRACK BÁSICO

ESTRUCTURAS DE DATOS NO LINEALES

Gabriel Carmona Tabja

Universidad Técnica Federico Santa María,
Università di Pisa

October 27, 2024

Part I

EDD No LINEALES

EDD No LINEALES

Características

- ▶ Posibles multiples sucesores y predecesores (forma no lineal)

Part II

DICCIONARIOS

DICCIONARIOS

Definición

- ▶ Almacenan una relación **llave** con **valor**

DICCIONARIOS

Definición

- ▶ Almacenan una relación **llave** con **valor**
- ▶ La llave es única

DICCIONARIOS

Definición

- ▶ Almacenan una relación **llave** con **valor**
- ▶ La llave es única

Tipos de Diccionarios

- ▶ Diccionarios Ordenados
- ▶ Diccionarios No Ordenados

DICCIONARIOS ORDENADOS

Definición

Diccionario en cual ordena las llaves por algún método de comparación a medida que se insertan/eliminan llaves

DICCIONARIOS ORDENADOS

Definición

Diccionario en cual ordena las llaves por algún método de comparación a medida que se insertan/eliminan llaves

En C++

- Declaración: `map< T, T2 >`

DICCIONARIOS ORDENADOS

Definición

Diccionario en cual ordena las llaves por algún método de comparación a medida que se insertan/eliminan llaves

En C++

- ▶ Declaración: `map< T, T2 >`
- ▶ La implementación es un árbol llamado *Red-Black Tree* (especie de avl)

DICCIONARIOS ORDENADOS

Definición

Diccionario en cual ordena las llaves por algún método de comparación a medida que se insertan/eliminan llaves

En C++

- ▶ Declaración: `map< T, T2 >`
- ▶ La implementación es un árbol llamado *Red-Black Tree* (especie de avl)
- ▶ Operaciones como: encontrar, borrar e insertar son $O(\log n)$

UTILIZACIÓN - INSERCIÓN

```
1  int main() {
2      map< int, int > mp;
3
4      mp.insert({1, 8});
5      mp.insert({2, 1});
6      mp.insert({4, 3});
7      mp.insert({3, 2});
8      mp.insert({3, 2}); // no inserta porque ya existe la llave
9
10     // casi equivalente
11
12     mp[1] = 8;
13     mp[2] = 1;
14     mp[4] = 3;
15     mp[3] = 2;
16     mp[3] = 1; // reemplazo la llave 3
17
18     // cuantas llaves hay
19     cout << mp.size() << "\n";
20     return 0;
21 }
```

UTILIZACIÓN - ACCESO

```
1  int main() {
2      map< int, int > mp;
3
4      mp.insert({1, 8});
5      mp.insert({2, 1});
6      mp.insert({4, 3});
7      mp.insert({3, 2});
8
9      // funcion find
10     map< int, int >::iterator it = mp.find(2);
11     if(it == mp.end()) {
12         cout << "La llave 2 no esta\n";
13     } else {
14         cout << "La llave " << it->first << "esta y su valor es " << it->second << "\n";
15     }
16
17     // operator []
18     // si la llave 2 no esta se inserta automatica con valor default segun el tipo de dato
19     // en caso de los enteros es 0
20     // tener cuidado al hacer esto, porque aumenta el tamaño del diccionario
21     cout << mp[2] << "\n";
22     return 0;
23 }
```

UTILIZACIÓN - BORRAR

```
1  int main() {
2      map< int, int > mp;
3
4      mp.insert({1, 8});
5      mp.insert({2, 1});
6      mp.insert({4, 3});
7      mp.insert({3, 2});
8
9      cout << mp.size() << "\n";
10
11     mp.erase(3); // si la llave no esta no hace nada
12
13     cout << mp.size() << "\n";
14     return 0;
15 }
```

UTILIZACIÓN - RECORRER

```
1  int main() {
2      map< int, int > mp;
3
4      mp.insert({1, 8});
5      mp.insert({2, 1});
6      mp.insert({4, 3});
7      mp.insert({3, 2});
8
9      // usando iteradores
10     for(map< int, int >::iterator it = mp.begin(); it != mp.end(); it++) {
11         cout << "{" << it->first << ", " << it->second << "} ";
12     }
13     cout << "\n";
14
15     // usando el foreach
16     for(pair< int, int > p : mp) {
17         cout << "{" << p.first << ", " << p.second << "} ";
18     }
19     cout << "\n";
20
21     return 0;
22 }
```

UTILIZACIÓN - LOWER/UPPER_BOUND

```
1  int main() {
2      map< int, int > mp;
3
4      mp.insert({1, 8});
5      mp.insert({2, 1});
6      mp.insert({10, 3});
7      mp.insert({5, 2});
8
9      map< int, int >::iterator low;
10     low = mp.lower_bound(5);
11     // 5 2
12     cout << low->first << " " << low->second << "\n";
13
14     map< int, int >::iterator up;
15     up = mp.upper_bound(5);
16     // 10 3
17     cout << up->first << " " << up->second << "\n";
18
19     return 0;
20 }
```


REFERENCES I