# Taller ProgComp: Track Básico

## Estructuras de Datos No Lineales

**Gabriel Carmona Tabja**

Universidad Técnica Federico Santa María,
Università di Pisa

October 11, 2024

# Part I

## EDD No Lineales

# EDD LINEALES

## Características

- Posibles multiplos 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

```cpp
int main() {
  map< int, int > mp;

  mp.insert({1, 8});
  mp.insert({2, 1});
  mp.insert({4, 3});
  mp.insert({3, 2});
  mp.insert({3, 2}); // no inserta porque ya existe la llave

  // casi equivalente

  mp[1] = 8;
  mp[2] = 1;
  mp[4] = 3;
  mp[3] = 2;
  mp[3] = 1; // reemplazo la llave 3

  // cuantas llaves hay
  cout << mp.size() << "\n";
  return 0;
}
```

# Utilización - Acceso

```cpp
int main() {
  map< int, int > mp;

  mp.insert({1, 8});
  mp.insert({2, 1});
  mp.insert({4, 3});
  mp.insert({3, 2});

  // funcion find
  map< int, int >::iterator it = mp.find(2);
  if(it == mp.end()) {
    cout << "La llave 2 no esta\n";
  } else {
    cout << "La llave " << it->first << "esta y su valor es " << it->second << "\n";
  }

  // operator []
  // si la llave 2 no esta se inserta automatica con valor default segun el tipo de dato
  // en caso de los enteros es 0
  // tener cuidado al hacer esto, porque aumenta el tamano del diccionario
  cout << mp[2] << "\n";
  return 0;
}
```

# Utilización - Borrar

```cpp
int main() {
  map< int, int > mp;

  mp.insert({1, 8});
  mp.insert({2, 1});
  mp.insert({4, 3});
  mp.insert({3, 2});

  cout << mp.size() << "\n";

  mp.erase(3); // si la llave no esta no hace nada

  cout << mp.size() << "\n";
  return 0;
}
```

# UTILIZACIÓN - RECORRER

```cpp
int main() {
  map< int, int > mp;

  mp.insert({1, 8});
  mp.insert({2, 1});
  mp.insert({4, 3});
  mp.insert({3, 2});

  // usando iteradores
  for(map< int, int >::iterator it = mp.begin(); it != mp.end(); it++) {
    cout << "{" << it->first << ", " << it->second << "} ";
  }
  cout << "\n";

  // usando el foreach
  for(pair< int, int > p : mp) {
    cout << "{" << p.first << ", " << p.second << "} ";
  }
  cout << "\n";

  return 0;
}
```

# Utilización - lower/upper_bound

```cpp
int main() {
  map< int, int > mp;

  mp.insert({1, 8});
  mp.insert({2, 1});
  mp.insert({10, 3});
  mp.insert({5, 2});

  std::map< int, int >::iterator low;
  low = mp.lower_bound(5);
  // 5 2
  cout << low->first << " " << low->second << "\n";

  std::map< int, int >::iterator up;
  up = mp.upper_bound(5);
  // 10 3
  cout << up->first << " " << up->second << "\n";

  return 0;
}
```

# REFERENCES I