ESTRUCTURAS DE DATOS

TIPOS ABSTRACTOS DE DATOS ARBORESCENTES

Implementando recorridos en profundidad (*DFS*)

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

Tipos de recorridos

- Recorrido en profundidad

  Depth First Search (DFS)

  - Preorden
  - Inorden
  - Postorden

SE PUEDEN IMPLEMENTAR DE MANERA RECURSIVA.

Son los más fáciles de implementar.

- Recorrido en anchura

  Breadth First Search (BFS)

Se implementan todos de manera muy parecida. Lo único que cambia es el momento en el que se visita cualquiera de los subárboles y la raíz.

Preorden: 1ºRaíz 2º Hijo izquierdo 3º Hijo derecho
Inorden: 1º Hijo izquierdo 2º Raíz 3º Hijo derecho
Postorden: 1º Hijo izquierdo 2º Hijo derecho 3º Raíz.

Recordatorio: interfaz de BinTree<T>

```
template<class T>
class BinTree {
public:
  BinTree();
  BinTree(const T &elem);
  BinTree(const BinTree &left, const T &elem, const BinTree &right);

  const T & root() const;
  BinTree left() const;
  BinTree right() const;
  bool empty() const;

private:
  ...
};
```

Métodos que implementa la interfaz BinTree

Elemento de la raíz.

árbol izquierdo y derecho respectivamente.

árbol vacío.

Atributo Node *root_node un puntero al nodo raíz.

Recordatorio: interfaz de `BinTree<T>`

```
template<class T>
class BinTree {
public:
  // ...
  void preorder() const;
  void inorder() const;
  void postorder() const;

private:
  ...
};
```

- Añadimos tres nuevos métodos a `BinTree<T>`.

Operaciones para poder recorrer los árboles binarios.

Recordatorio: interfaz de BinTree<T>

```
template<class T>
class BinTree {
public:
  // ...
  void preorder() const {
    preorder(root_node);
  }

  void inorder() const {
    inorder(root_node);
  }

  void postorder() const {
    postorder(root_node);
  }

private:
  static void preorder(const NodePointer &node);
  static void postorder(const NodePointer &node);
  static void inorder(const NodePointer &node);
 ...
};
```

apunta a la raíz del árbol

- Estos métodos harán uso de otros tres métodos privados auxiliares.

Imprimirán el nodo por pantalla.

Puntero a la raíz del árbol. De esta manera será más fácil tratar con árboles vacíos.

Método auxiliar preorder

```
template<typename T>
void BinTree<T>::preorder(const NodePointer &node) {
  if (node ≠ nullptr) {
    std::cout << node→elem << " ";
    preorder(node→left);
    preorder(node→right);
  }
}
```

Solo si no es nullptr. Nullptr no es solo un árbol vacío sino que también puede ser una rama del árbol después de una hoja.

Primero visita el nodo raíz luego visita el subárbol izquierdo y luego el subárbol derecho

Método auxiliar inorder

```
template<typename T>
void BinTree<T>::inorder(const NodePointer &node) {
  if (node ≠ nullptr) {
    inorder(node→left);
    std::cout << node→elem << " ";
    inorder(node→right);
  }
}
```

Primero visitamos el subárbol izquierdo, luego la raíz y después el subárbol derecho

Método auxiliar postorder

Entre todos solo camban el orden en el cual se realizan las llamadas recursivas. Fácil implementación.

```
template<typename T>
void BinTree<T>::postorder(const NodePointer &node) {
  if (node ≠ nullptr) {
    postorder(node→left);
    postorder(node→right);
    std::cout << node→elem << " ";
  }
}
```

Primero visitamos el subárbol izquierdo luego el subárbol derecho y por último la raíz del árbol

Ejemplo

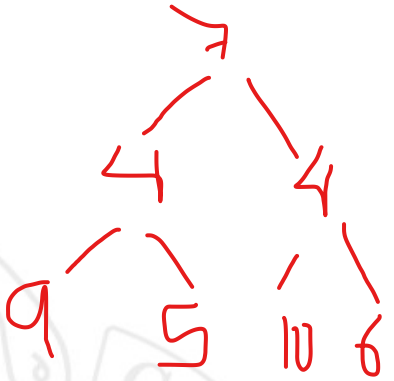
```
int main() {
  BinTree<int> tree = {{{ 9 }, 4, { 5 }}, 7, {{ 10 }, 4, { 6 }}};

  std::cout << "Recorrido en preorden: " << std::endl;
  tree.preorder();
  std::cout << std::endl;

  std::cout << "Recorrido en inorden: " << std::endl;
  tree.inorder();
  std::cout << std::endl;

  std::cout << "Recorrido en postorden: " << std::endl;
  tree.postorder();
  std::cout << std::endl;

  return 0;
}
```




```
Recorrido en preorden:
7 4 9 5 4 10 6
Recorrido en inorden:
9 4 5 7 10 4 6
Recorrido en postorden:
9 5 4 10 6 4 7
```