ESTRUCTURAS DE DATOS

TIPOS ABSTRACTOS DE DATOS ARBORESCENTES

Árboles binarios de búsqueda

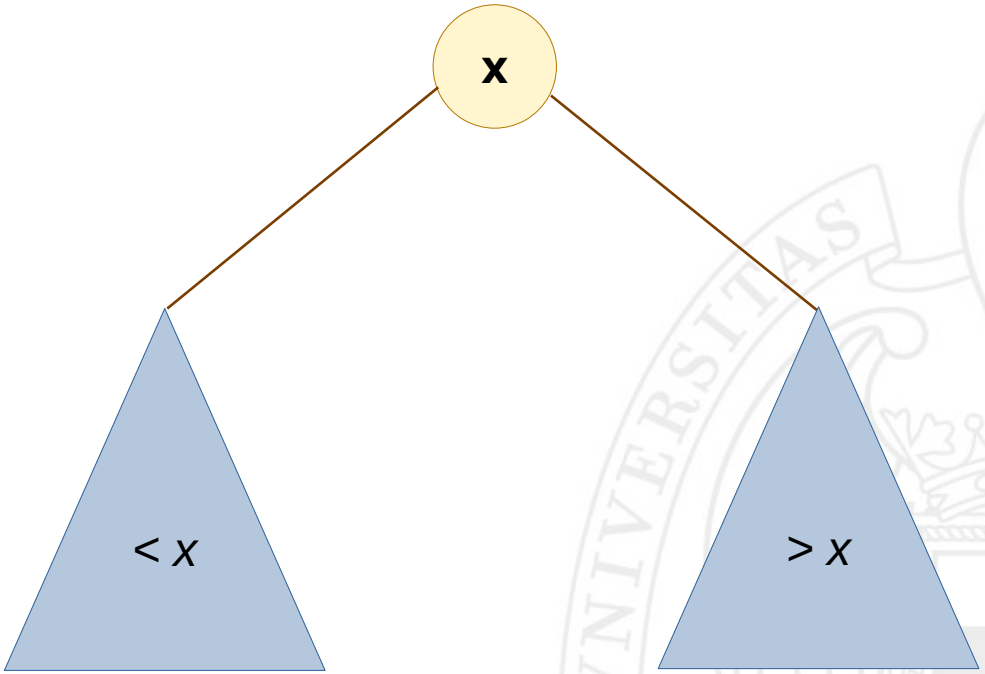
Manuel Montenegro Montes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática – Universidad Complutense de Madrid

Árboles binarios de búsqueda (ABBs)

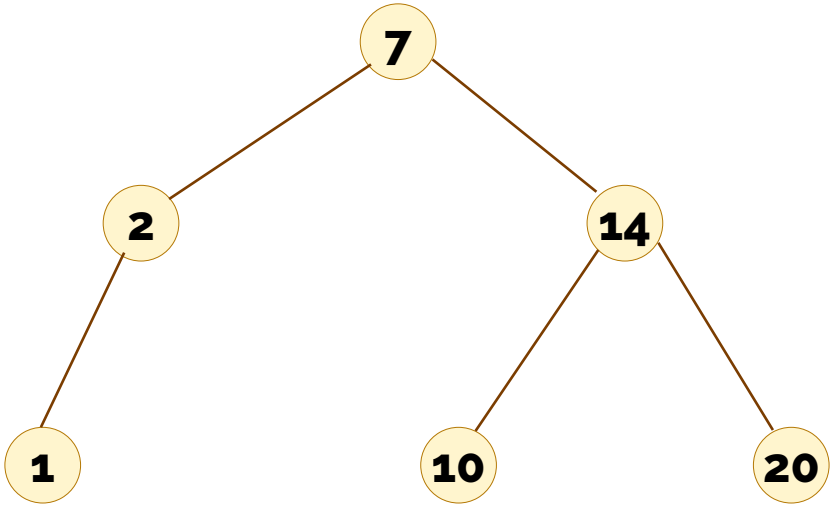
Un árbol binario es de búsqueda si:

- Es un árbol vacío, o bien,
- Es una hoja, o bien,
- Su raíz es un nodo interno, y además:
  - Todos los elementos de su **hijo izquierdo** son estrictamente **menores** que la raíz.
  - Todos los elementos de su **hijo derecho** son estrictamente **mayores** que la raíz.
  - Los hijos izquierdo y derecho son árboles de búsqueda.

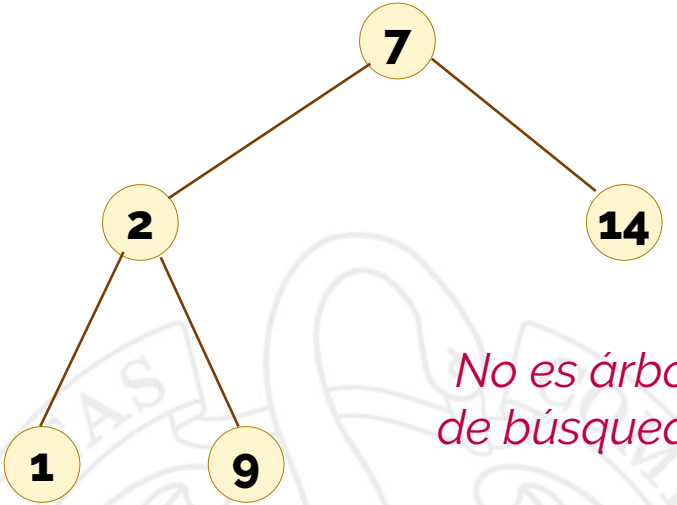
Árboles binarios de búsqueda



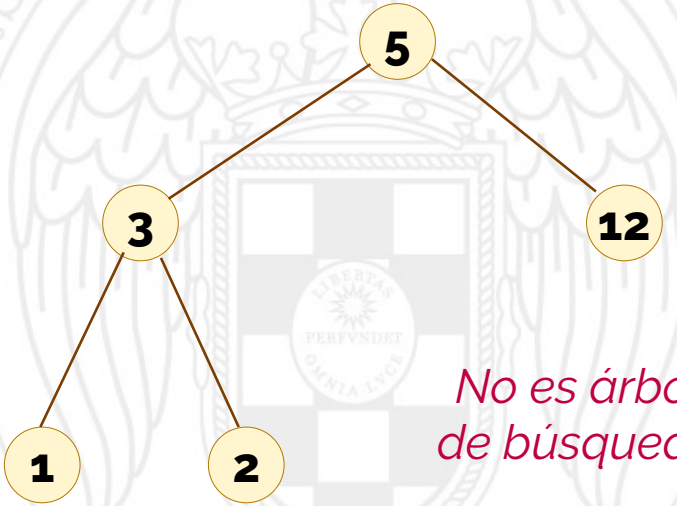
Ejemplos



Árbol de búsqueda



No es árbol de búsqueda

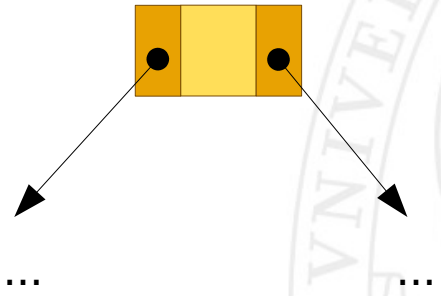


No es árbol de búsqueda

Representación mediante nodos

```
template <typename T>
struct Node {
  T elem;
  Node *left, *right;

  Node(Node *left, const T &elem, Node *right): left(left), elem(elem), right(right) { }
};
```

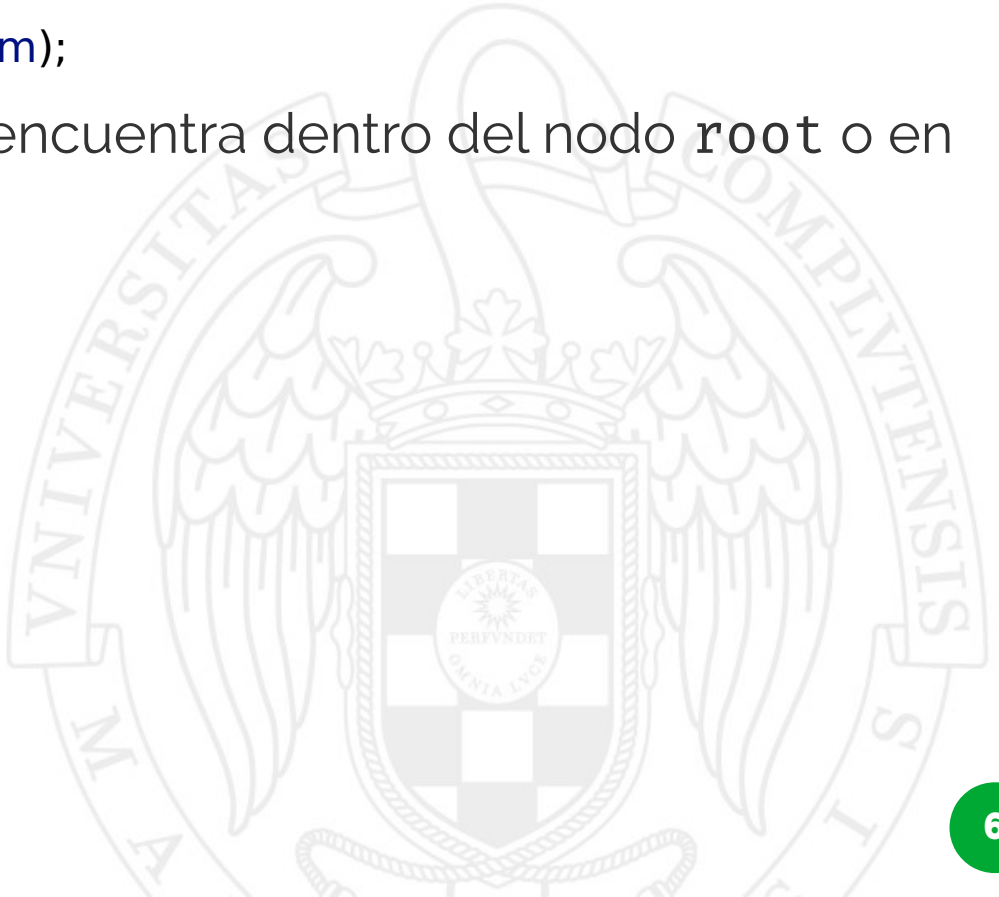


Búsqueda en un ABB

- Queremos implementar una función que determine si un elemento se encuentra en un árbol de búsqueda

```
bool search(const Node *root, const T &elem);
```

- La función determina si el `elem` se encuentra dentro del nodo `root` o en alguno de sus descendientes.
- Distinguimos cuatro casos.



Caso 1: Árbol vacío

```
bool search(const Node *root, const T &elem);
```

- Si `root == nullptr`, el árbol es vacío.
- En ese caso, `elem` no pertenece al árbol.
- Devolvemos `false`.

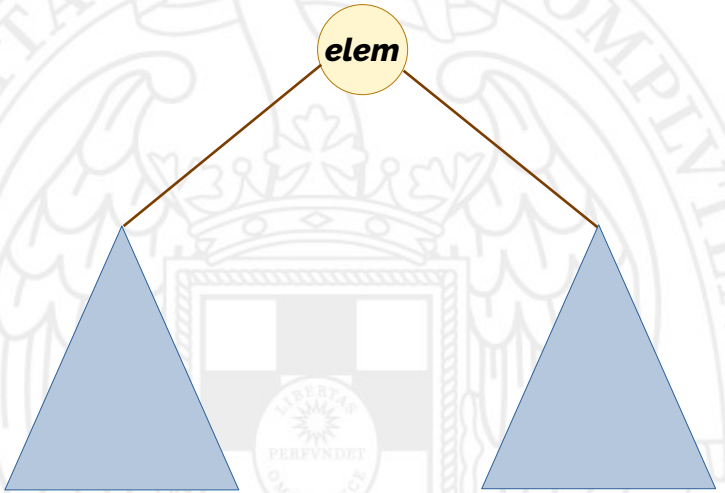
```
bool search(const Node *root, const T &elem) {
  if (root == nullptr) {
    return false;
  } else { ... }
}
```

Caso 2: `elem` == raíz del árbol

```
bool search(const Node *root, const T &elem);
```

- En este caso, hemos encontrado `elem` en el árbol. Devolvemos `true`.

```
bool search(const Node *root, const T &elem) {
  if (root == nullptr) {
    return false;
  } else if (elem == root->elem) {
    return true;
  } else { ... }
}
```

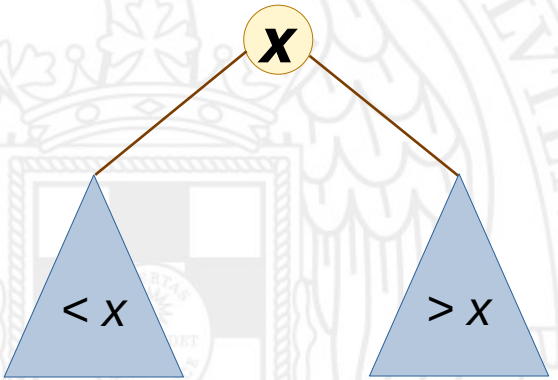


Caso 3: `elem` < raíz del árbol

```
bool search(const Node *root, const T &elem);
```

- Si el elemento a buscar es estrictamente menor que la raíz del árbol, lo buscamos recursivamente en el hijo izquierdo.

```
bool search(const Node *root, const T &elem) {
  if (root == nullptr) {
    return false;
  } else if (elem == root->elem) {
    return true;
  } else if (elem < root->elem) {
    return search(root->left, elem);
  } else { ... }
}
```

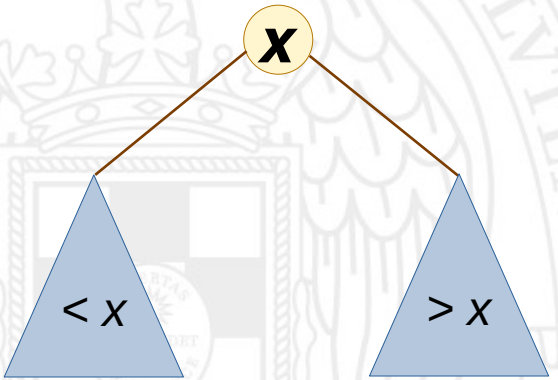


Caso 4: `elem` > raíz del árbol

```
bool search(const Node *root, const T &elem);
```

- Si el elemento a buscar es estrictamente mayor que la raíz del árbol, lo buscamos recursivamente en el hijo derecho.

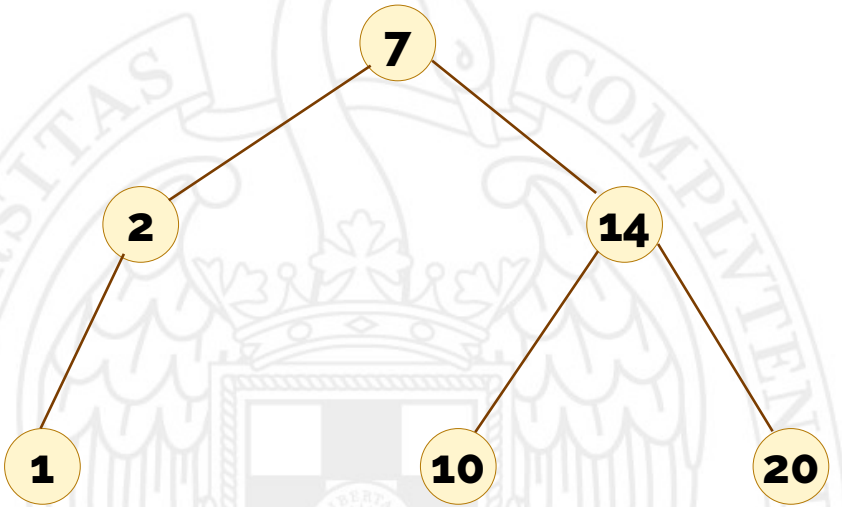
```
bool search(const Node *root, const T &elem) {
  if (root == nullptr) {
    return false;
  } else if (elem == root->elem) {
    return true;
  } else if (elem < root->elem) {
    return search(root->left, elem);
  } else {
    return search(root->right, elem);
  }
}
```



Ejemplo

- Buscamos el 10

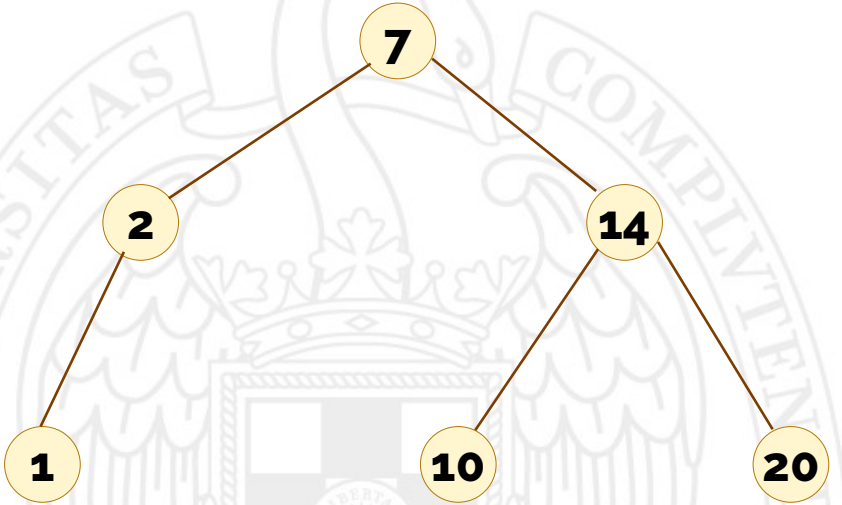
```
bool search(const Node *root, const T &elem) {
  if (root == nullptr) {
    return false;
  } else if (elem == root->elem) {
    return true;
  } else if (elem < root->elem) {
    return search(root->left, elem);
  } else {
    return search(root->right, elem);
  }
}
```



Ejemplo

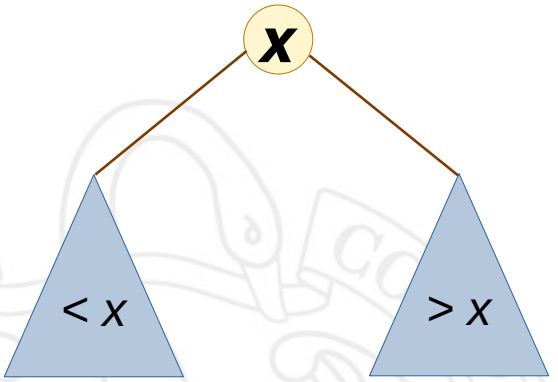
- Buscamos el 3

```
bool search(const Node *root, const T &elem) {
  if (root == nullptr) {
    return false;
  } else if (elem == root->elem) {
    return true;
  } else if (elem < root->elem) {
    return search(root->left, elem);
  } else {
    return search(root->right, elem);
  }
}
```



Coste de la función search

```
bool search(const Node *root, const T &elem) {
  if (root == nullptr) {
    return false;
  } else if (elem == root->elem) {
    return true;
  } else if (elem < root->elem) {
    return search(root->left, elem);
  } else {
    return search(root->right, elem);
  }
}
```



Coste de la función `search`

- En el caso peor, la función `search` desciende desde la raíz hasta las hojas.
- El coste en tiempo de la función `search` es lineal con respecto a la altura del árbol.

¿Y con respecto al número de nodos?

Recordatorio

Sea h la altura de un árbol y n su número de nodos.

- Si el árbol es **degenerado**, $h \in O(n)$
- Si el árbol es **equilibrado**, $h \in O(\log n)$
- Si no sabemos nada acerca de si el árbol está equilibrado o no, el caso peor es el árbol degenerado.

Coste de la función `search`

- Si el árbol es **degenerado**, el coste de `search` es $O(n)$, donde n es el número de nodos del árbol.
- Si el árbol está **equilibrado**, el coste de `search` es $O(\log n)$, donde n es el número de nodos del árbol.