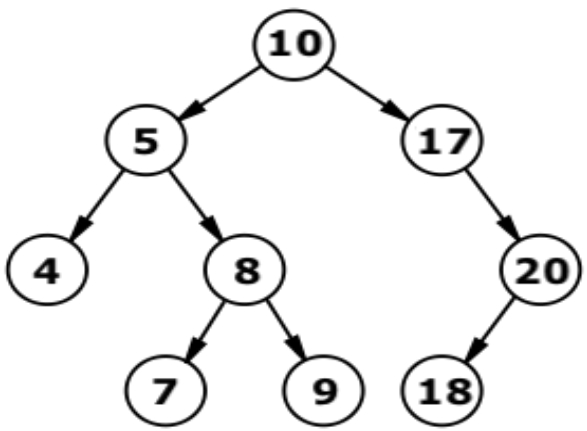
UNIVERSIDAD DE MURCIA

Tipos de Datos Abstractos Sin Orden Lineal

Tecnología de la Programación

L. Daniel Hernández < *ldaniel@um.es* >

Dpto. Ingeniería de la Información y las Comunicaciones
Universidad de Murcia
30 de octubre de 2023

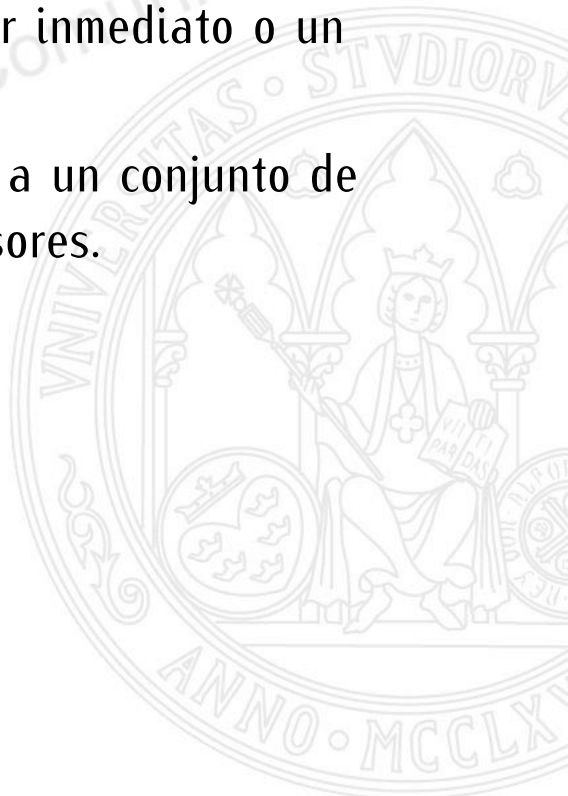


Índice de Contenidos

Desarrollo

Qué son los TDAs sin relación de orden

- Los TDA con **ordenación lineal** son aquellos que contienen a un conjunto de objetos pero donde cada objeto tiene exactamente un predecesor inmediato o un sucesor inmediato.
- Los TDA con **sin ordenación lineal** son aquellos que contienen a un conjunto de objetos pero donde cada objeto puede tener de 0 a varios sucesores.
- Podemos distinguir dos tipos de ordenaciones:
  - Ordenación jerárquica.
  - Ordenación parcial.



TDAs con Relación Jerárquica - I

- Los TDAs con **ordenación jerárquica** quedan definidos mediante una relación binaria $a \preceq b$, que será una ordenación no lineal sii hay un solo objeto **raíz** r (un solo primer elemento) y se cumple:
  - Para todo a, $a \preceq a$ (Reflexiva),
  - Para todo a, existe r verificando $r \preceq a$ (existe un mínimo en el orden).
  - Si $a \preceq c$ y $b \preceq c$, esto implica que $a \preceq b$ o $b \preceq a$.
  - La relación es antisimétrica: $a \preceq b$ y $b \preceq a$, esto implica que $a = b$
  - Es transitiva: $a \preceq b$ y $b \preceq c$, esto implica que $a \preceq c$
- En este orden pueden existir **elementos no comparables**.
- Los elementos tienen **nombre** en un orden jerárquico:
  - Si $a \preceq b$, a es **padre** de b y b es **hijo** de a.
  - Cualquier objeto x que cumpla $a \preceq x$ es un **descendiente** o sucesor de a.
  - Cualquier objeto x que cumpla $x \preceq b$ es un **ascendiente** o ancestro de b.
  - El primer elemento de la ordenación recibe el nombre de **raíz**, y por tanto cumple que solo tiene descendientes y no tiene padre.
  - Los elementos que no tienen hijos se llaman **hojas**.
  - Los que no son ni hojas ni raíz se llaman **intermedios**.

TDAs con Relación Jerárquica - II

- **Las operaciones** que se pueden realizar en un orden jerárquico son:
  - *Acceder* directamente al objeto raíz.
  - Dado un objeto *acceder* a su padre y a sus hijos.
  - *determinar* si es la raíz o es una hoja.
  - Dados dos objetos, *encontrar el primer ancestro común*.
  - Dados dos objetos, *encontrar el recorrido* para llegar de uno a otro.
  - Hacer un *recorrido por todos* los descendientes de un objeto.
  - *Eliminar* un objeto y todos sus descendientes.

TDAs con Relación Parcial

- En esta ordenación lo que no puede ocurrir es que dado un objeto, nos lo volvamos a encontrar siguiendo la relación de orden. Es decir, $a \preceq b \preceq \ldots, \preceq a$ no puede darse: es un orden en el que **no se permiten ciclos**.
- Los TDAs con **ordenación parcial** quedan definidos mediante una relación binaria $a \preceq b$, que será una ordenación no lineal sii se cumple:
  - Relación reflexiva: Para todo a, $a \preceq a$.
  - Relación es antisimétrica: $a \preceq b$ y $b \preceq a$, esto implica que $a = b$
  - Es transitiva: $a \preceq b$ y $b \preceq c$, esto implica que $a \preceq c$
- En este orden pueden existir **elementos no comparables**.
- Los elementos tienen **nombre** en un orden parcial: Reciben los mismos nombres que si tuvieran una ordenación jerárquica (raíz, hijo, intermedio, etc).
- Algunas **operaciones** que se pueden realizar en un orden jerárquico son:
  - *Acceder* a todos los objeto raíces.
  - Dado un objeto *acceder* a sus padres y a sus hijos, *determinar* si es una raíz o es una hoja.
  - Dados dos objetos, *encontrar* el primer ancestro común o el recorrido para llegar de uno a otro.
  - Hacer un *recorrido* por todos todos los objetos

Desarrollo

Profundizamos en ...

Definición Recursiva del TDA Árbol

- Un **árbol** responde a la siguiente definición recursiva que consta de los siguientes casos:
  - **Caso base:** El **conjunto vacío** es un árbol. El árbol vacío.
  - **Caso recursivo:** Un árbol es una colección de datos que está formada por **un dato**, r, **y una lista** de 0 o más árboles, $A_1, A_2, \ldots A_n$.
- Se deberá cumplir una ordenación jerárquica donde:
  - Cada dato r será el elemento raíz del árbol que se esté definiendo, y será padre de las raíces de los árboles A_i.
  - Los árboles A_i se llaman subárboles del árbol cuya raíz es r.
- Está definición nos da un método de **construcción estructural**, empezando por construir el árbol vacío. ¡¡ Hágase !!
- A los objetos de un árbol se les llama **vértices** o **nodos**, pero no debe confundir una **estructura de datos de tipo nodo** (estructura enlazada de datos).

Operaciones del TDA Árbol - I

- Operaciones básicas:
  - `Tree(): Tree`. Crea un nuevo árbol
  - `append(value, node=pos): None`. Añade un nuevo nodo al árbol con el valor `value`. Si se especifica el segundo parámetro, el nuevo nodo será hijo de `pos`.
  - `value(pos): type_value`. Retorna el contenido del nodo (posición) `pos`.
  - `replace(pos, value): value`. Cambia el valor del nodo de la posición `pos` por un nuevo valor (el de entrada) y retorna el antiguo valor.
  - `remove(pos): None`. Elimina el nodo de la posición `pos`.
  - `parent(pos): pos`. Retorna la posición del padre para la posición `pos`. Será `None` si la posición de entrada se corresponde con la raíz.
  - `children(pos): container`. Retorna un contenedor iterable con todos los hijos de `pos`.
  - `positions(): container`. Retorna un contenedor iterable con todos los nodos del árbol.
  - `elements(): container`. Retorna un contenedor iterable con todos los valores del árbol.
  - `num_children(pos): int`. Indica el número de hijos que tiene el nodo `pos`.
  - `len(): int`. Retorna el número de elementos que tiene el grafo
  - `depth(pos): int`. Retorna la profundidad del nodo `pos`.

Operaciones del TDA Árbol - II

- `height(pos): int`. Retorna la altura del nodo `pos`.
- `root(): pos`. Retorna la posición del nodo raíz del árbol.
- `isRoot(pos): Bool`. Retorna *True* si la posición `pos` es el nodo raíz del árbol. Retorna *False* en otro caso.
- `isInternal(pos): Bool`. Retorna *True* si la posición `pos` es el de un nodo interno. Retorna *False* en otro caso.
- `isLeaf(pos): Bool`. Retorna *True* si `pos` es un nodo hoja del árbol. Retorna *False* en otro caso.
- `isEmpty(): Bool`. Indica si el árbol está vacío o no.

- Se pueden ampliar con una gran cantidad de métodos como, por ejemplo:
  - `borrar(): None`. Borrar los nodos empezando por el nivel más profundo.
  - `copiar(): Tree`. Copiar un árbol empezando por la raíz.
    Copia superficila o profunda (a determinar).
  - `contar(criterio): int`. Contar el número de elementos que cumplan cierto criterio.
  - `buscar(valor): bool`. Buscar un elemento en el árbol.
  - `comparar(arbol): bool`. Indica si el árbol dado coincide con el actual.
  - `altura(): int`. Calcula la altura del árbol.
  - `hojas(): int`. Calcula el número de hojas del árbol.

Profundizamos en ...

Implementación del TDA Árbol

- Estructura:

```
struct Arbol
    Nodo raiz
```

```
struct Nodo
    TDato dato
    Secuencia nodosHijos
```

- Un árbol vacío es aquel que cumpla arbol.raiz = None.
- Notar que al ser un Árbol una composición de nodos se debe aplicar **delegación**. Por ejemplo: dos árboles son iguales si lo son cada uno de sus nodos (el dato y la secuencia de hijos).
- Ejercicio:
  - ¿Cuál es la función de abstracción? $Abst : \mathbf{rep} \longrightarrow \mathcal{A}$
  - ¿Cuál es el invariante de la representación? $I : \mathbf{rep} \longrightarrow \mathbb{B}$

Desarrollo

Profundizamos en ...

Definición Recursiva del TDA Árbol

- De entre los árboles 2-ários distinguimos los árboles binarios.
- **Definición:**
  Un **árbol binario** es un árbol que **siempre** tiene dos subárboles que reciben el nombre de hijo (o subárbol) izquierdo e hijo (o subárbol) derecho. En un árbol binario los subárboles pueden ser vacíos.
- Algunos árboles binarios importantes:
  - **Árbol de búsqueda binario.** Se utiliza en muchas aplicaciones de búsqueda donde los datos entran/salen constantemente, como los objetos map y set en las bibliotecas de muchos idiomas.
  - **Partición de espacio binario.** Determina que objetos de un escenario deben renderizarse antes. Se utiliza en casi todos los videojuegos 3D.
  - **Montones.** Se usan para implementar colas de prioridad. Estas colas se usan en sistemas operativos (prioridad de procesos) y en algoritmo de búsqueda de caminos (A^* se usa en aplicaciones de AI , incluyendo robótica y videojuegos).
  - **Huffman Coding Tree.** Se utiliza como algoritmo de compresión, como los utilizados por los formatos de archivo .jpeg y .mp3.

Profundizamos en ...

Recorridos en un Árbol Binario - I

-
```
def recorrido_anchura (nodo):
    cola = Cola()
    cola.encola(nodo)
    while not cola.isEmpty():
        node = cola.desencola()
        accion(node) # P.e. print(node)
        if  nodo.izquierdo is not None:
            cola.encola(nodo.izquierdo)
        if nodo.derecho is not None:
            cola.encola(nodo.derecho)
```

-
```
def recorrido_profundidad (nodo):
    pila = Pila()
    pila.push(nodo)
    while not pila.isEmpty():
        node = pila.pop()
        accion(node) # P.e. print(node)
        if  nodo.izquierdo is not None:
            pila.push(nodo.izquierdo)
        if nodo.derecho is not None:
            pila.push(nodo.derecho)
```

Recorridos en un Árbol Binario - II

- El **recorrido en profundidad** sigue alguna de las siguientes 3 estrategias recursivas:
  1. Recorrido prefijo.

```
def recorrido_prefijo (nodo):
    accion(nodo)
    recorrido(nodo.izquierdo)
    recorrido(nodol.derecho)
```

  2. Recorrido infijo.

```
def recorrido_infijo (nodo):
    recorrido(nodo.izquierdo)
    accion(nodo)
    recorrido(nodol.derecho)
```

  3. Recorrido postfijo.

```
def recorrido_postfijo (nodo):
    recorrido(nodo.izquierdo)
    recorrido(nodol.derecho)
    accion(nodo)
```

Operaciones Basadas en Recorridos

Consultar la documentación para una descripción más detallada

- **mostrar(arbol)**: Mostrar los elementos de un árbol como si fuera una expresión aritmética:
  - Caso base: Mostrar los elementos de un árbol vacío es hacer nada.
  - Caso general: Realizar un recorrido infijo (mostrando los nodos)
- **mostrar(arbol)**: Mostrar los elementos de un árbol como si fuera una estructura de directorio.
- **borrar(arbol)**: Borrar los nodos empezando por el nivel más profundo.
- **copiar(arbol): arbol**: Copiar un árbol empezando por la raíz.
- **contar(arbol): int**: Contar el número de elementos que cumplan cierto criterio.
- **buscar(arbol, valor): bool**: Buscar un elemento en el árbol.
- **comparar(arbol1, arbol2): bool**: Comparar dos árboles.
- **altura(arbol): int**: Calcular la altura de un árbol.
- **hojas(arbol): int**: Calcular el número de hojas.
- Y hay más ....

Profundizamos en ...

Árbol Binario de Búsqueda.

- **Arboles binarios** de búsqueda cumplen la siguientes2 **propiedades**:
  - El valor del hijo izquierdo es menor que el valor de la raíz.
  - El valor del hijo derecho es mayor que el valor de la raíz.
- Por **recursividad**, se cumplirá que
  - todos los descendiente de la rama izquierda son menores que el valor de la raíz.
  - todos los descendiente de la rama derecha son mayores que el valor de la raíz.
- Ejemplo el de la portada.
- **Operaciones** del TDA Árbol Binario de Búsqueda:
  - `buscar(arbol, valor): bool`. Algoritmo de búsqueda.
  - `añadir(arbol, valor)`. Añadir un elemento en el árbol. Los nuevos elemento se añaden siempre como nodos hojas.
  - `minimo(arbol): valor`. Buscar el valor mínimo de un árbol.
  - `maximo(arbol): valor`. Buscar el valor máximo de un árbol.
  - `borrar(arbol, valor): arbol`: Eliminar el nodo que tiene un valor. El árbol resultante tiene que seguir siendo un árbol binario de búsqueda.

Montículos (Heap)

- Un heap es un árbol binario donde cada nodo consta de una pareja (*clave*, *valor*). Existe una ordenación entre los nodos:

$$(clave, valor) < (clave', valor') \Leftrightarrow clave < clave'$$

- En muchos casos la clave se obtiene a partir del valor. Por ejemplo, si los valores son numéricos entonces se puede usar como clave dicho valor numérico.
- Todo heap satisface estas dos propiedades:
- CONTINUARÁ ....