Tecnología de la programación
Sesión 23

Deberes para la evaluación continua

1. Los deberes de esta sesión se mandarán la semana que viene, para que tengáis tiempo de mirarla antes.

Objetivos de la sesión

1. Introducción a los TADs NO lineales. Conceptos sobre árboles (transparencias 1 hasta 17).
2. Ejercicios de contar el número de nodos de un árbol binario y construir un árbol simétrico

Guion

Introducción

Este tema es difícil. Se mezcla:

1. TADs, que encima serán NO son lineales (hasta ahora todos los que hemos visto son lineales)
2. Punteros
3. Recursividad

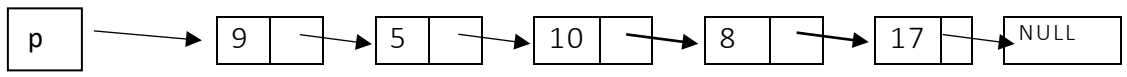
Por tanto, hay que mirarse despacio y con mucha atención todo para entenderlo bien.

Los árboles son muy importantes, porque se usan muchísimo en programación debido a sus propiedades. Son una estructura de datos que puede permitir realizar operaciones básicas, como inserción, búsqueda y eliminación en complejidad O(logn), o incluso O(1) dependiendo del árbol. De hecho, hay árboles de muchos tipos. Solo por nombrar algunos que se me ocurren ahora mismo:

- Árbol binario: https://es.wikipedia.org/wiki/%C3%81rbol_binario
- Árbol binario de búsqueda: https://es.wikipedia.org/wiki/%C3%81rbol_binario_de_b%C3%BAsqueda
- Árbol equilibrado: https://es.wikipedia.org/wiki/%C3%81rbol_binario_de_b%C3%BAsqueda_auto-balanceable
- Árbol rojo-negro: https://es.wikipedia.org/wiki/%C3%81rbol_rojo-negro
- Árbol AVL: https://es.wikipedia.org/wiki/%C3%81rbol_AVL
- Árbol 2-3: https://es.wikipedia.org/wiki/%C3%81rbol_2-3
- Árbol B: https://es.wikipedia.org/wiki/%C3%81rbol-B
- Árbol Splay: https://es.wikipedia.org/wiki/%C3%81rbol_biselado

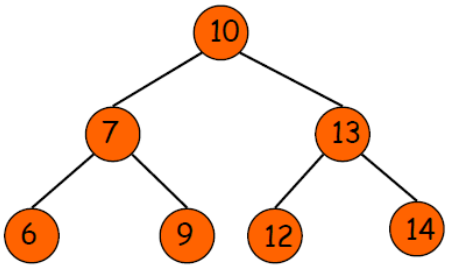
TADs no lineales

Hasta ahora hemos trabajado con TADs lineales. Esto significa que cada elemento tiene como mucho un siguiente (como en pilas, colas y listas).

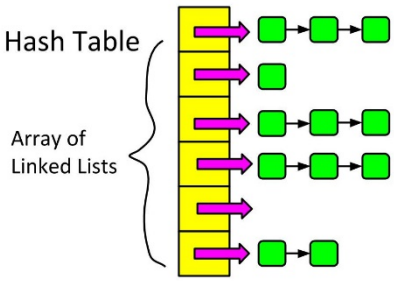


Los TADs no lineales serán aquellos en los que la estructura de datos permite que cada elemento pueda tener varios siguientes. Por ejemplo.

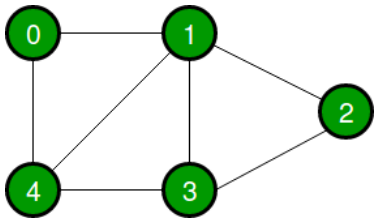
Árbol



Tabla



Grafo

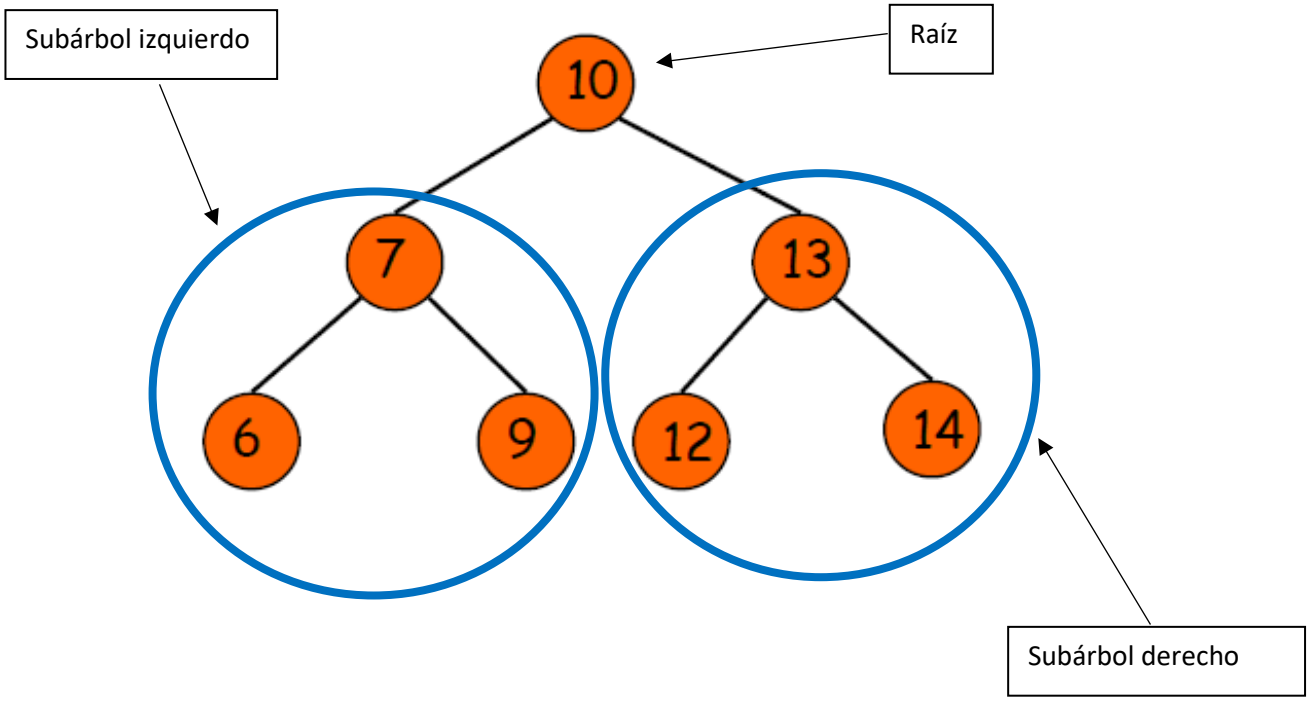


Definición de árbol

En este tema nos vamos a centrar en árboles. La definición de árbol a veces os cuesta entenderla, por ser recursiva. Pero tenéis que pensar dos casos:

- O el árbol es vacío
- O el árbol tiene una raíz, y como hijos otros árboles (llamados subárboles).

El caso de árbol binario, es un árbol donde cada nodo tiene como mucho dos hijos (que forman el subárbol izquierdo y el subárbol derecho). Es un caso muy particular de árbol.



Como la definición es recursiva, cada subárbol a su vez, tiene que cumplir las propiedades de árbol. Es decir, o es vacío, o tiene raíz con dos subárboles.

Términos asociados al concepto de árbol.

Tenéis que aprenderos bien los siguientes conceptos (transparencias 9, 10 y 11). Son definiciones fáciles.

- Nodo padre
- Camino
- Nivel
- Nodo antecesor
- Nodo descendiente
- Nodo hoja
- Nodo interior
- Profundidad o altura
- Grado de un nodo
- Grado de un árbol
- Árbol n-ario
- Árbol binario

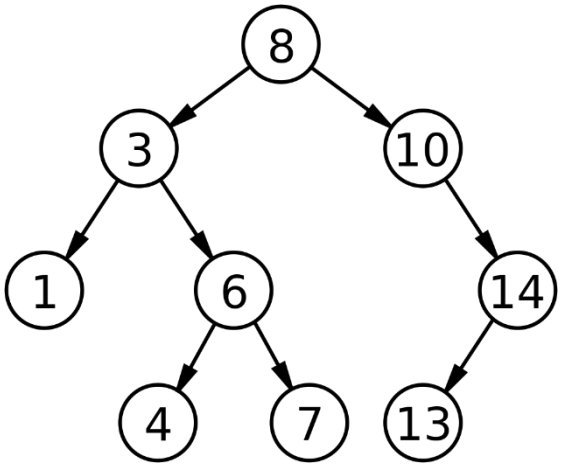
Árboles binarios

La definición formal de árbol binario es la siguiente:

Definición. Un árbol binario es un árbol tal que:

- Es el conjunto vacío, en cuyo caso se denomina árbol vacío; o,
- Existe un elemento distinguido llamado raíz y el resto de elementos se distribuyen en dos conjuntos disjuntos, cada uno de los cuales es un árbol binario, llamados subárboles izquierdo y derecho del árbol

Es importante remarcar que todos los nodos (salvo cuando estemos en un árbol vacío) tienen dos subárboles. Pero esos subárboles pueden ser vacíos. Es decir, lo siguiente también es un árbol binario:



De hecho, el nodo 13 también tiene dos hijos: dos subárboles que son vacíos.

Es un TAD muy conocido, y vamos a especificarlo (suponiendo que tElemento es un tipo básico).

```
tad árbolBinario(tElemento)
usa
      tElemento
género
      arbolBin
operaciones
      acción iniciarArbol(sal arbolBin A)
      {Pre: }
      {Post: inicia A como un árbol vacío}
Operaciones (cont)
acción enraizar(sal arbolBin A, e/s arbolBin Aiz, e/s arbolBin Ader,
                ent tElemento d)
{Pre: Aiz y Ader son árboles iniciados o construidos previamente}
{Post: Construye el árbol A cuya raíz es el dato d, y del cual penden
los árboles Aiz y Ade. El acceso a los subárboles de A mediante Aiz y Ade
queda anulado}

función izquierdo(arbolBin A) dev arbolBin
{Pre: A es un árbol no vacío}
{Post: devuelve el árbol izquierdo que pende del nodo raíz de A}

función derecho(arbolBin A) dev arbolBin
{Pre: A es un árbol no vacío}
{Post: devuelve el árbol derecho que pende del nodo raíz de A}

función raíz(arbolBin A) dev tElemento
{Pre: A es un árbol no vacío}
{Post: devuelve el dato situado en el nodo raíz de A}

función árbolVacío(arbolBin A) dev booleano
{Pre: A es un árbol iniciado o creado previamente}
{Post: devuelve Verdad si A está vacío y Falso en caso contrario}
```

fin_especificación

Cosas "raras" de este TAD:

No tiene una operación insertar. Es decir, no tenemos la operación que, dado un elemento y un árbol, te inserta dicho elemento en el árbol. Realmente habría muchas formas de insertar. Así que lo que tenemos es una operación enraizar: dados dos árboles y un elemento, te construye el árbol que tiene como raíz el elemento dado y como subárboles los árboles dados.

Tiene dos funciones llamadas izquierdo y derecho que devuelven árboles. Como sabéis, no deberíamos devolver nunca un TAD por evitar comportamientos inesperados. Pero los árboles van a ser un caso especial, y además, por simplicidad, trabajaremos siempre con punteros. De hecho, nos interesa hacerlo así, para evitar tener que trabajar con copias de los subárboles cada vez que accedemos a izquierdo y derecho (sería muy ineficiente). Un árbol será un puntero al nodo raíz. Un nodo será un registro formado por tres elementos: el dato, y dos punteros a los dos siguientes nodos (subárbol izquierdo y derecho).

No hay operación de crear a partir de los datos por pantalla (habría varias formas), ni de mostrar (hay varias formas de mostrar por pantalla los datos), ni de copiar.

USO del TAD Árbol Binario

Como siempre, si conocemos la especificación, podemos usarla e implementar algoritmos como usuarios. Nos da igual cómo esté implementado el TAD.

Os recomiendo intentar los ejercicios propuestos, y luego mirar la solución.

```
función numNodos (arbolBin A) dev entero
{Pre: A es un árbol binario iniciado}
{Post: devuelve el número de nodos del árbol}
principio
    si ArbolVacio(A)
        dev 0
    si_no
        dev(1 + numNodos(izquierdo(A)) + numNodos(derecho(A)))
    fsi
fin
```

```
acción simétrico (e/ arbolBin A, e/s arbolBin b)
{Pre: A es un árbol binario iniciado}
{Post: B es el árbol simétrico de A}
variables
    arbolBin izdo, dcho
principio
    si ArbolVacio(A)
        iniciarArbol(B)
    si_no
        simétrico(derecho(A), dcho)
        simétrico(izquierdo(A), izquierdo)
        enraizar(B, dcho, izdo, raíz(A))
    fsi
fin
```

Este os suele costar mucho!!!

Necesitamos variables auxiliares para construir los subárboles izquierdo y derecho

Primero construimos, luego enraizamos.