Programación y Estructura de Datos

Temario de Teoría y resumen del Seminario de C++

Universitat d'Alacant
Universidad de Alicante

Resumen

Explicación del documento

Eduardo Espuch

Curso 2019-2020

Índice

prueba

1. Introducción a los TADs, los tipos lineas

prueba

1.1. Introducción a los TADs

prueba

1.2. Vectores

prueba

1.3. Listas

prueba

1.4. Pilas

prueba

1.5. Colas

prueba

2. La eficiencia de los algoritmos

prueba

2.1. Noción de complejidad

prueba

2.1.1. Complejidad temporal, tamaño del problema y paso

prueba

2.2. Cotas de complejidad

prueba

2.2.1. Cota superior, inferior y promedio

prueba

2.3. Notación asintótica

prueba

2.4. Obtención de cotas de complejidad

prueba

3. El tipo árbol

Vamos a comenzar recordando brevemente que es un árbol, dado previamente en la asignatura de Matemáticas Discreta. Podéis verlo con más detalle en los apuntes de esa asignatura.

Los arboles son grafos conexos y aciclicos y diremos que es un árbol generador si es árbol y subgrafo generador a la vez.

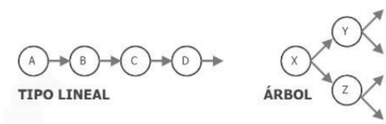
1. Dos vértices cualesquiera están conectados por un único camino.
2. Un grafo sera conexo si y solo si tiene un árbol generador.
3. El numero de relaciones en un árbol difiere en uno al numero de vértices de este
4. Todo árbol no trivial tiene al menos 2 vértices de grado 1.

Dado un vértice r_0, el cual actúa como extremo inicial, únicamente y todo vértice está conectado con éste, podemos definir un árbol enraizado a r_0 y distinguiremos los distintos niveles o alturas n a los vértices cuyos caminos sean de longitud n.

Cabe decir que, aunque es recomendable conocer que es un árbol en un contexto matemático, no es necesario.

3.1. Definiciones generales

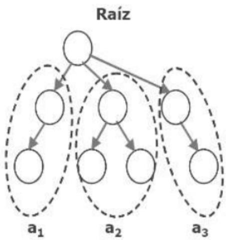
Diremos que la estructura de datos del tipo árbol aparece a la hora de establecer una jerarquía entre los elementos que lo constituyen, obtenido a partir de eliminar el requisito de que cada elemento en una estructura de datos del tipo lineal solo podría tener como máximo un sucesor. Llamaremos a los elementos de estructura como **nodos**.



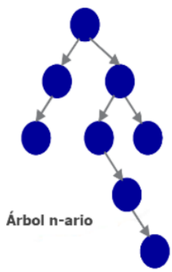
Procedamos a enlistar una serie de propiedades y definiciones respecto al concepto de la estructura del tipo árbol:

- Un único nodo puede constituir un árbol, ademas de ser la raíz de éste.
- Diremos que un **árbol es vació** o **nulo** si este tiene 0 nodos, es decir, no tiene una raíz.
- La información almacenada dentro del nodo se denomina **etiqueta**.

- Dados n arboles $a_1, \ldots, a_n$ se puede construir uno nuevo al enraizar todos arboles a un nuevo nodo, pasando estos a ser **subárboles** del nuevo árbol, siendo el nuevo nodo la raíz de éste.

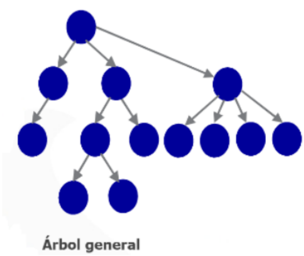


- **Árbol n-ario** es un arraigado árbol en el que cada nodo no tiene mas que n sucesores. Se podría decir que todo árbol solo podrá tener como máximo n árboles. Lo llamaremos **árbol binario** cuando $n = 2$.

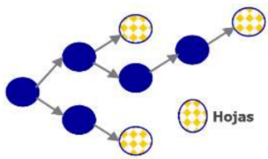


  Es mas, llamaremos grado de un árbol al número máximo de sucesores que podrán tener los subárboles, siendo en el árbol n-ario un grado de n.

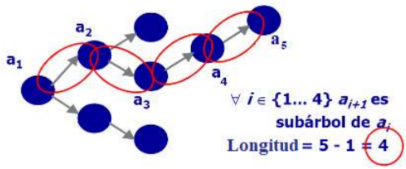
- **Árbol general** es un árbol sin una restricción en el número de sucesores por nodos, es decir, que podrá tener una cantidad de subárboles indeterminada.



- Llamaremos **hojas** a los arboles compuestos por un único nodo, o dicho de otra manera, cuando un nodo no tiene sucesores.

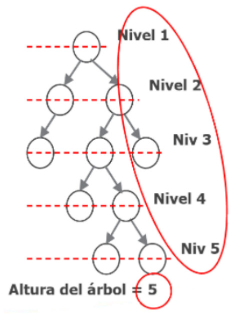


- **Camino:**
  - una secuencia $a_1, \ldots, a_s$ de árboles tal que $\forall i \in \{1, \ldots, s-1\}$, a_{i+1} es subárbol de a_i. Pueden darse casos de caminos de un árbol a si mismo, siendo la sucesión el propio árbol.
  - La **longitud** del camino corresponde a la cantidad árboles menos uno ya. Se considera que la longitud de un árbol a si mismo es de 0. Esto se debe a que los caminos consisten en enlistar los nodos que son sucesores pero la longitud es el numero de sucesiones necesarias para alcanzar desde la raíz el ultimo elemento del camino.



- Terminologías para sucesores:
  - Diremos que x es **ascendiente** de y si existe un camino con x como la raíz del árbol e y uno de los sucesores directos o indirectos de x. También podemos decir que y es **descendiente** de x.

    Debido a la definición de camino, todo árbol es ascendiente(descendiente) de sí mismo.
  - Diremos que son **ascendiente(descendientes) propios** todos los ascendiente(descendientes) excluidos del propio árbol.
  - **Padre** es el primer ascendiente propio de un árbol. No todos los arboles tienen padre.
  - **Hijos** son los primeros descendientes propios, si existen, de un árbol.
  - **Hermanos** son los subárboles con el mismo padre.
  - **Profundidad** de un subárbol es la longitud del único camino desde la raíz a dicho subárbol. Recordad que un árbol/subárbol consiste en el nodo y sus sucesores.



•**Nivel** de un nodo:

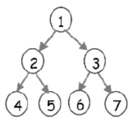
-el nivel de un árbol vacío es 0
-el nivel de la raíz es 1
-si un nodo está en el nivel i,
sus hijos están en el nivel $i+1$

•**Altura(profundidad)** de un árbol:

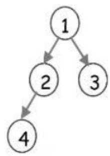
-es el máximo nivel de los nodos de un árbol

Hay casos en los que la gente inicializa la raíz al nivel 0 o el nivel mas bajo es donde esta la hoja con el camino de mayor longitud a la raíz (de igual manera, pudiendo ser el nivel inicial 1 o 0). Y la altura, según la definición dada, se aplicaría sin ningún problema a la forma que se decida usar. Para esta asignatura se considerara lo mostrado previamente, pero considerar esto como posible ya que la gente puede hacer lo que quiera siempre que se explique al principio.

•**Árbol lleno** es árbol en el que todos sus subárboles tienen n hijos(con n siendo el grado del árbol) y todas sus hojas tienen la misma profundidad.



•**Árbol completo** es un árbol cuyos nodos corresponden a los nodos numerados (la numeración se realiza desde la raíz hacia las hojas y, en cada nivel, de izquierda a derecha) de 1 a n en el árbol lleno del mismo grado, siendo todo lleno uno completo.



En algunos casos, se dice que un árbol completo n-ario cuando todos los nodos tienen n sucesores o son hoja, parecido a lo que seria un árbol lleno, pero aquí se dice que es un árbol que se va llenando de izquierda a derecha por nivel, nunca se podrá tener un una hoja a profundidad x y otra a un nivel inferior a $x-1$.

3.2. Árboles Binarios

El árbol binario es un caso particular de árbol n-ario donde $n = 2$. Podemos definir lo como un conjunto de elementos del mismo tipo tal que, distinguiéremos un elemento que llamaremos raíz y el resto de elementos del conjunto se distribuyen en dos subconjuntos disjuntos, siendo estos a su vez dos arboles binarios. Si el árbol es un conjunto vació, es decir, no hay elementos, sera un árbol vació o nulo.

Para entendernos, la idea es que tu tienes el árbol que esta compuesto por un nodo o ninguno, y el nodo siempre se esperara que tenga 2 arboles asociados a él, pudiendo estar estos con o sin nodo.

Vamos a definir una serie de propiedades:

- El máximo numero de nodos en un nivel i de un árbol binario será $N(i) = 2^{i-1}, i \geq 1$. Podemos probarlo para $N(1) = 2^0 = 1$ como la base, y, asumiendo que es cierto para $N(i-1) = 2^{(i-1)-1} = 2^{i-2}$, podemos inducir, como para el siguiente nivel serán el doble de nodos, $N(i) = N(i-1) * 2 = 2^{i-2+1} = 2^{i-1}$. Haremos uso de esta función en el documento usando $n(i)$

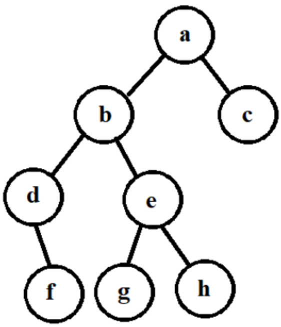
- El máximo numero de nodos en un árbol binario de altura k es $N(k) = 2^k - 1, k \geq 1$. Básicamente se obtiene de hacer el sumatorio del numero máximo de nodos por nivel, es decir, $\sum_{i=1}^{k} n(i) = 2^0 + 2^1 + \ldots 2^{k-1} = 2^k - 1$. Cuando hagamos referencia a esta formula en el documento usando $h(i)$.

Llamamos **recorrer** un árbol a la acción de visitar cada nodo del árbol una sola vez, siendo el **recorrido** la lista de etiquetas del árbol ordenado según se visitan los nodos. Distinguiremos dos categorías, **recorridos en profundidad** y **recorridos en anchura o por niveles**.

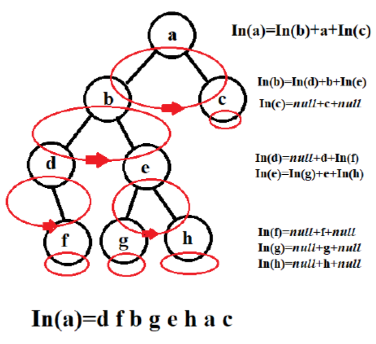
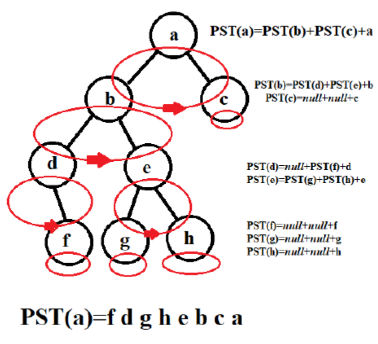
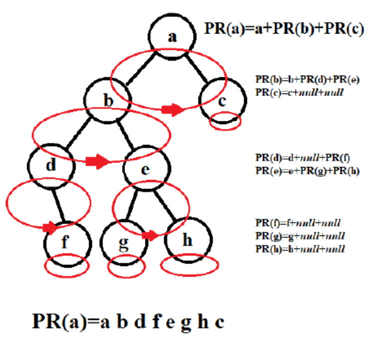
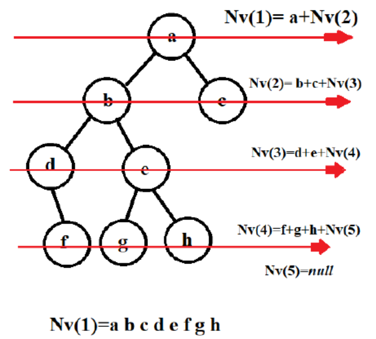
- Recorrido en profundidad: representando desplazarse por el árbol por la izquierda con **I**, desplazarse por la derecha como **D** y acceder a la etiqueta del nodo como **R**, hay 6 formas de recorrerlo diferenciados por el sentido, de derecha a izquierda (**DIR**, **RDI**, **DRI**) y de izquierda a derecha (**IDR**,**IRD**,**RID**), veremos estos tres últimos con mas detalle:
  - **IDR** Postorden o orden posterior
  - **IRD** Inorden o orden simétrico
  - **RID** Preorden o orden previo

- Recorrido en anchura: consiste en visitar los nodos desde la raíz hacia las hojas, de izquierda a derecha dentro de cada nivel.

Veamos un ejemplo de los distintos recorridos para un mismo árbol, siendo el siguiente:



Notese que este árbol no es uno completo pero si recibe la numeración apropiada.

<table>
<tr><td>Inorden</td><td>Postorden</td></tr>
<tr><td>
</td>
<td>
</td></tr>
<tr><td>Preorden</td><td>Niveles</td></tr>
<tr><td>
</td>
<td>
</td></tr>
</table>

Programación y estructuración de un árbol binario

Definamos la sintaxis y la semántica para este tipo de dato, no sin antes añadir un listado de conceptos a tener en cuenta:

- **Crear()**:*ab*, devuelve un árbol vació.
- **Enraizar(ab,item,ab)**:*ab*, devuelve un árbol con item como raíz que contiene los dos subárboles indicados.
- **raiz(ab)**:*item*, devuelve el item que se encuentra en la raiz del arbol. Puede darse el caso que el árbol este vació y la raíz sea nula.
- **esVacio(ab)**:*bool*, comprueba si un árbol es vació comprobando únicamente si su raíz es nula.
- **hijoiz, hijode(ab)**:*ab*, devuelve el subárbol enraizado a la izquierda o la derecha del árbol. Si el árbol es nulo, entonces se devolverían árboles vacíos.

- **altura(ab)**:*int*, devolverá la longitud del camino que vaya desde la raíz a una de sus hojas que sea mayor.
- **nodosHoja(ab)**:*int*, devolverá el numero de nodos que no están enraizados a arboles no vacíos. Cabe decir que un árbol vació no tiene nodos.
- **simétricos(ab,ab)**:*bool*, comprobara si dos arboles son simétricos (reflexión de espejo).
- **iguales(ab,ab)**:*bool*, similar a **simétricos**, comprobara si el contenido de dos arboles es idéntico y tienen la misma estructura.
- **quita_hojas(ab)**:*ab*, dado un árbol, devolverá otro el cual no contenga los nodos hoja del introducido.
- Tipos de recorrido de profundidad:
  - **Inorden(ab)**:*lista*, usando el recorrido en inorden, se van añadiendo a la lista las etiquetas que se van leyendo durante el recorrido.
  - **Preorden(ab)**:*lista*, usando el recorrido en preorden, se van añadiendo a la lista las etiquetas que se van leyendo durante el recorrido.
  - **Postorden(ab)**:*lista*, usando el recorrido en postorden, se van añadiendo a la lista las etiquetas que se van leyendo durante el recorrido.
- Recorrido en anchura:
  - **Niveles(ab)**:*lista*, usando el recorrido en niveles, se van añadiendo a la lista las etiquetas que se van leyendo durante el recorrido.
- Tipos de representación:
  - Representación secuencial: consiste en llenar los nodos hipotéticamente numerándolos de izquierda a derecha por nivel, desde el nivel 1. De esta forma, cuando añadamos un item al árbol en una posición, esta posición ya estaría considerada previamente a la asignación.
  - Representación enlazada: cada árbol tiene una dirección única y, recordando que todo árbol contiene un nodo el cual apunta a otros dos arboles, se puede asignar un subárbol al nodo apropiado e ir modificándose en el caso de se deba de hacer.
- Asignacion(copia) entre arboles e iteradores:
  - **ab::operator=(ab)**:*void*, copia el arbol completamente en el árbol al que se asigna.
  - **ab::operator=(ab_iter)**:*void*, copia la rama a la que apunta el iterador en el árbol al que se asigna.
  - **ab_iter::operator=(ab)**:*void*, copia el árbol en la rama a la que apunta el iterador al que se asigna.
  - **ab_iter::operator=(ab_iter)**:*void*, sirve para inicializar el iterador al que se asigna para que apunte al mismo iterador.

  # Esto se traduciría usando puntero e ir accediendo a los subárboles de un árbol.
- Moviemiento de ramas entre arboles e iteradores:
  - **mover(ab,ab)**:*void*, el contenido del segundo árbol pasaría a estar en el primero, el segundo queda vació.

- **mover(ab,ab_iter))**:*void*, el contenido de la rama a la que apunta el iterador pasara a ser el contenido del árbol.
- **mover(ab_iter,ab)**:*void*, mueve el árbol a la posición del iterador, dejando el árbol vació.
- **mover(ab_iter,ab_iter)**:*void*, el contenido de la rama del segundo sustituirá la rama a la que apunta el primero iterador.

Esto se traduciría usando puntero e ir accediendo a los subárboles de un árbol.

En los apuntes del profesorado hay mas funciones como **todos**, **transforma**, **dos_hijos** (aunque modificado se podría usar para comprobar si un árbol es lleno o completo),. Los que se muestran son los considerados de mayor utilidad.

Definamos ahora la sintaxis y la semantica para un tipo arbol:

Sintaxis

Modulo Arbol_Binario
Usa BOOL, NATURAL, LISTA
Parámetro tipo Item
Operaciones del Item
error_ item()→item
FParamétro
Tipo ab
Operaciones

Crear()→ab	Enraizar(ab,item,ab)→ab
raiz(ab)→item	esVacío(ab)→bool
hijoiz,hijode(ab)→ab	altura(ab)→int
nodosHoja(ab)→int	simetricos(ab,ab)→bool
iguales(ab,ab)→bool	quita_hojas(ab)→ab
inorden(ab)→lista	preorden(ab)→lista
postorden(ab)→lista	niveles(ab)→lista

Operaciones que no definirán en la semántica:

asignacion(ab,ab)→void	mover(ab,ab)→void

Semántica

VAR i,d: ab; x:item;
Ecuaciones

Crear()→ab	Enraizar(ab,item,ab)→ab
f. canónica se denotara con CR	f. canónica, se denotara con ER
raiz(ab)→item	esVacío(ab)→bool
raiz(CR())=error_item	esVacío(ab)=T
raiz(ER(i,x,d))=x	esVacío(ER(i,x,d))=F
hijoiz,hijode(ab)→ab	altura(ab)→int
hijoiz(CR())=CR()	altura(CR())=0
hijoiz(ER(i,x,d))=i	altura(ER(i,x,d))=
hijode(CR())=CR()	1+max(altura(i),altura(d))
hijode(ER(i,x,d))=d	

iguales(ab,ab)→bool
iguales(CR(),CR())=V
iguales(CR(),d)=F
iguales(d,CR())=F
iguales(i,d)=
SI raiz(i)=raiz(d)
ENTONCES
iguales(hijoiz(i),hijoiz(d))&&
iguales(hijode(i),hijode(d))
SINO F

nodosHoja(ab)→int se denotara con NH
NH(CR())=0
NH(ER(CR(),x,CR()))=1
NH(ER(i,x,d))=NH(i)+NH(d)

inorden(ab)→lista se denotara con IN
IN(CR())=CR_lista()
IN(ER(i,x,d))=conc(insDe(IN(i),x),IN(d))

postorden(ab)→lista se denotara con PS
PS(CR())=CR_lista()
PS(ER(i,x,d))=insDe(conc(PS(i),PS(d)),x)

simetricos(ab,ab)→bool se denotara con SMT
SMT(CR(),CR())=V
SMT(CR(),d)=F
SMT(d,CR())=F
SMT(i,d)=
SI raiz(i)=raiz(d)
ENTONCES
SMT(hijoiz(i),hijode(d))&&
SMT(hijode(i),hijoiz(d))
SINO F

quita_hojas(ab)→ab se denotara con QH
QH(CR())=CR()
QH(ER(CR(),x,CR()))=CR()
QH(ER(i,x,d))=ER(QH(i),x,QH(d))

preorden(ab)→lista se denotara con PR
PR(CR())=CR_lista()
PR(ER(i,x,d))=conc(insIZ(x,PR(i)),PR(d))

niveles(ab)→lista se denotara con NV
NV(CR())=CR_lista()
NV(ER(i,x,d))= **PENDIENTE**

Funciones usadas de tipo **lista**:

CR_lista()→lista
crea una lista vacia

insIz(item,lista)→lista
inserta el item a la izquierda de la lista

conc(lista,lista)→lista
concatena dos listas

insDe(lista,item)→lista
inserta el item a la derecha de la lista

FModulo

3.3. Árboles de búsqueda

Los árboles de búsqueda, que también pueden ser árboles n-arios de búsqueda o árboles multicamino de búsqueda, son un tipo particular de arboles que pueden definirse cuando el tipo de los elementos del árbol posee una relación de orden total.

Para aclarar, un arbol multicamino de busqueda, que denotaremos como T es un arbol n-ario vacio o que cumple las siguientes propiedades:

- La raíz de T contiene $A_0, \ldots, A_{n-1}$ subárboles y $K_1, \ldots, K_{n-1}$ etiquetas.

  NOTA personal recordad que tratamos con árboles n-arios, un árbol que contiene un nodo el cual puede tener un máximo de n sucesores tratados como arboles, el nodo seria T, la raíz del árbol pero vería con más sentido que tuviésemos $K_0, \ldots, K_{n-1}$ y $A_1, \ldots, A_{n-1}$ siendo K_0 la etiqueta de la raíz. La forma en la que se define la veo rara pero mas adelante veremos que esto se debe a que habrán tantas etiquetas como sucesores que tenga T y no hace falta considerar la etiqueta para K_n, que estaría asociado con el árbol A_{n-1} ya que por el algoritmo. Dicho de otra manera, la etiqueta K_i correspondería al nodo raíz del subárbol A_{i-1}, con $1 \leq i \leq n$.

- $K_i < K_{i+1}$, $1 \leq i < n-1$

- Todas las etiquetas del subárbol A_i son:

$$\text{menores que } K_{i+1},\ 0 \leq i < n-1$$
$$\text{mayores que } K_i,\ 0 < i \leq n-1$$

- Los subárboles A_i, $0 \leq i \leq n-1$ son también árboles multicamino de búsqueda.

El algoritmo de búsqueda a seguir se basa en los siguientes pasos:

- Para buscar un valor x en el árbol, primero se mira el nodo raíz y se realiza la comparación respecto a la relación total(usaremos por comodidad la relación de orden total $\leq$.

1 $x = K_i$ Hemos encontrado el elemento.

2 Si $x < K_i$, por la definición de un árbol multicaminos x debe de estar en el subárbol A_{i-1} en el caso de que x exista en éste. Si no lo hiciera, probaríamos con $i = i + 1$.

3 $x > K_i$,de igual manera, asumimos que x debe de estar en el subárbol A_i. Si no lo hiciera, probaríamos con $i = i + 1$.

Los arboles multicamino son útiles cuando la memoria principal es insuficiente para utilizarla como almacenamiento permanente y, en el caso de usar una representacion enlazada, los punteros pueden representar direcciones de disco en lugar de direcciones de memoria principal.

3.3.1. Árboles binarios de búsqueda

Cuando el arbol de busqueda es binario, veremos las siguientes propiedades:

- Todos los elementos en el subárbol de la izquierda son previos a la raíz con respecto a la relación de orden total. Ésto se debe a que, con las definiciones y propiedades previas (y definiendo la relación como leq), $x \leq K_1$ teniendo todos los elementos previos organizados en el subárbol A_0.
- Todos los elementos en el subárbol de la izquierda son posteriores a la raíz con respecto a la relación de orden total. De igual manera, si se da $K_1 \leq x$ tendremos todos los elementos posteriores organizados en A_1.
- Ambos subárboles son a su vez binarios de búsqueda y, en algunos casos no se permite la repetición de etiquetas.

Sintaxis

Modulo Arbol_Binario_Busqueda
  Usa BOOL, Arbol_Binario
  Parámetro tipo Item
    Operaciones del Item
    $<, \ldots, >$:item,item→bool
    error_ item()→item
  FParamétro
  Operaciones

Insertar(ab,item)→ab	Buscar(ab,item)→bool
Borrar(ab,item)→ab	Min(ab)→item

Para indicar que hay operaciones previamente definidas en el árbol binario se pondrán las terminaciones **OP**_ab detrás de cada operación (**OP**).