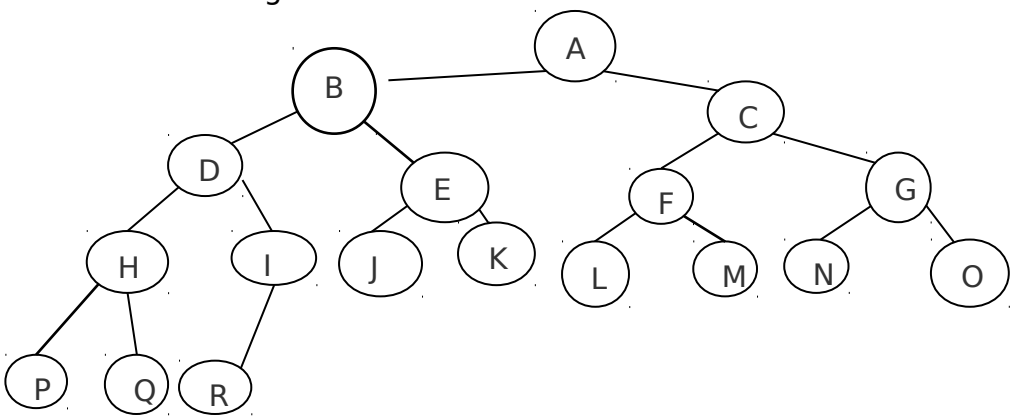
Unidad 5: Árboles, árboles binarios y árboles ordenados.

1. Considérese el árbol siguiente



a. ¿Cuál es su profundidad?
b. ¿Es un árbol binario?¿Es un árbol estrictamente binario? ¿Es casi-completo? Justifique la respuesta.
c. ¿Cuál es el predecesor inmediato, **padre**, del nodo R?
d. Recorra el árbol anterior según los algoritmos de preorden, orden medio, postorden y por niveles, indicando las expresiones parciales obtenidas en cada uno de los pasos

2. Escriba el pseudocódigo de los algoritmos apropiados para determinar:

a. La cantidad de nodos de un árbol binario
b. El número de hojas de un árbol binario.
c. La suma del contenido de los nodos de un árbol binario de elementos de tipo entero.
d. La profundidad de un árbol binario.

```
// Numero de nodos

int  abNodos (ab T)
  if abVacio(T)==TRUE return 0
  return 1+ abNodos (IZQ(T))+ abNodos (DER(T))
```

```
// Numero de hojas
int numHojas (ab T)
 if abVacio(T) ==TRUE return 0
 if esHoja (T) == TRUE return 1
 return numHojas (IZQ(T)) + numHojas (DER(T))

BOOL esHoja (ab T)
 if (abVacio(IZQ(T))==TRUE AND abVacio(DER(T))==TRUE)
          return TRUE
 return FALSE
```

```
// Suma contenido campos info
int  numInfo (ab T)
 if abVacio(T)==TRUE return 0
 return INFO(T) +numInfo (IZQ(T))+ numInfo (DER(T))
```

```
// Profundidad
int  abDepth (ab T)
 if abVacio(T)==TRUE return -1
 d = abDepth (DER(T))
 i = abDepth (IZQ(t))
 return maximo (d,i) + 1

int  maximo (int a, int b)
   if (a>b) return a
   else return b
```

3. Supóngase que para implementar un árbol binario se utilizan las siguiente estructuras y tipo de datos en C. Dar el código C de los algoritmos del problema anterior.

```
typedef enum {FALSE = 0, TRUE=1} BOOL;

typedef struct nodo_ab {
      Elemento *info;
      struct nodo_ab *izq, *der;
} NODO_AB;

struct _ARBOL{
      Nodo *root;
}

#define RIGHT(p) (p)->der
#define LEFT(p) (p)->izq
#define INFO(p) (p)->info
#define ROOT(p) (p)->root
```

```
// Numero de nodos

int abNodes (const ARBOL *pa) {
 if (pa==NULL) return -1;
 return abNodesRec (pa->root);
}

int abNodesRec (Nodo *pa) {
 if (pa == NULL) return 0;
 return (1 + abNodes (LEFT(a))+ abNodes (RIGHT(a)));
}
```

```
// Numero de hojas

int numHojas (ARBOL *pa) {
  if (pa == NULL) return -1;
  return  numHojasREc (pa->root);
}

int numHojasREc (Nodo *pa) {
 if (pa == NULL) return 0;
 if ( esHoja(pa) == TRUE) return 1;
 return  numHojasRec(LEFT(pa))+numHojasRec(RIGHT(pa));
}

 BOOL esHoja (Nodo *pa) {
      if ( LEFT(pa)== NULL  &&  RIGHT(pa) == NULL)
           return TRUE;
      return FALSE;
}
```

4. Escriba el pseudocódigo de un algoritmo para determinar si un árbol binario es un árbol binario de búsqueda.

```
// SOLUCION INCORRECTA   Proporcione un ejemplo de árbol que no sea un abdb pero que este algoritmo devuelve TRUE

BOOL isABdB (const ARBOL a) {
 BOOL l,r;
 if (isEmptyAB(a)==TRUE) return TRUE;

 //compara los campos info
 if ((INFO(a) < INFO(LEFT(a))) || (INFO(a) > INFO(LEFT(a)))
   return FALSE
```

```
// SOLUCIÓN CORRECTA
BOOL isABdB (const ARBOL a) {
 return isBBdBminmax (a, INT_MIN, INT_MAX);
}

BOOL isBBdBminmax (ARBOL a, int min, int max) {
 if (isEmptyAB(a)==TRUE) return TRUE;

 // falso si viola la restriccion min/max
 if ( (INFO(a) < min) || (INFO(a) > max)
   return FALSE

 l= isBBdBminmax (LEFT(a), INFO(a)-1, max);
```

```
 l= isABdB (LEFT(a));
 r= isABdB (RIGHT(a));

 return ( l &&  r);
}
```

// SOLUCION CORRECTA pero MUY INEFICAZ **¿Por que?**
En esta solución se supone que la función MIN devuelve el menor campo info del subárbol que se le pasa como argumento (MAX el máximo)

```
BOOL isABdB (const ARBOL a) {
 BOOL l,r;
 if (isEmptyAB(a)==TRUE) return TRUE;

 //compara los campos info
 if ((INFO(a) < MIN(LEFT(a))) || (INFO(a) > MAX(LEFT(a)))
   return FALSE

 l= isABdB (LEFT(a));
 r= isABdB (RIGHT(a));

 return ( l &&  r);
}
```

```
 r= isBBdBminmax (RIGHT(a), min, INFO(a)+1);

  return ( l &&  r);
}
```

5. Escriba el pseudocódigo de un algoritmo para determinar si un árbol binario es:
   a. Estrictamente binario.
   b. Completo.

```
// Determina si ses estrictamente binario

BOOL isStricBinary (ARBOL a)
  if isEmptyAB(a) == TRUE THEN return TRUE

  else if  isEmpty(LEFT(a))==TRUE AND isEmpty(RIGHT(a))==TRUE
       THEN return TRUE

  else if  isEmpty(LEFT(a))==TRUE OR isEmpty(RIGHT(a))==TRUE
       THEN return FALSE

  return   isStricBinary (LEFT(a)) AND isStricBinary (RIGHT(a))
```

6. Escriba el código C, sin control de errores, de una función para determinar si un árbol binario es estrictamente cuasi-completo.

```
// Recorro el árbol en anchura. Si en algún momento extraigo de la cola un hijo que no es vacio después de haber extraido un hijo vacio, es que el árbol no es estrictamente cuasi-completo.
Boolean arbol_esCuasiCompleto (Arbol *pa) {
    Cola *pc;
    Elemento *ele;
    flag = OK;
```

```
  pc = cola_crear();
  cola_insert (pc, ROOT(pa));
  while (cola_vacia (pc)==FALSE) {
     ele = cola_extraer (pc);
      if (ele !=NULL) {
          // si ant4es he extraido de la cola un NULL no sera cuasi
          if (flag == FIRST_NA) return FALSE
          cola_insert (pc, IZQ(ele));
          cola_insert(pc, DER(ele))
       }
    else  flag = FIRST_NA;
  }
 return TRUE
}
```

7. Escribir una función en C que reciba un árbol binario y un campo info y devuelva el nivel del árbol en que se encuentra ó -1 si no se encuentra en el árbol. Si el árbol tuviese nodos repetidos debería devolver el nodo más profundo. Suponed que se didpone de la función que permite comparar dos elementos de prototipo

   int elemento_comparar (Elemento *ele1, Elemento *ele2)

```
int obtieneNivel (Elemento *info, const ARBOL *a) {
 if (a == NULL) return -1;
 return obtieneNivelRec (info, a->root);
}

int obtieneNivelREc (Elemento *info, Nodo *a) {
 if (a == NULL) return -1;
 if (elemento_comparar(INFO(a), info)==0) return 0;

 i = obtieneNivelRec (IZQ(a));
 d = obtieneNivelRec (DER(a));

 if ( (i!=-1) || (d!=-1) ) return (1 + maximo (i, d) );
 return -1;
}
```

8. Repetir el problema anterior suponiendo que el árbol, es un árbol binario de búsqueda. ¿Por que el algoritmo del ejercicio 6 no es eficiente para un AbdB?

9. Proporcione el código C de una función de prototipo

   void arbol_prtDesdeNivel (Arbol *pa, int nivel, FILE *pf)

   que imprima en un flujo de salida pf el campo info de todos los nodos del árbol pa que estén en un nivel menor o igual a nivel. Si la profundidad del árbol fuese inferior al nivel la función devolverá ERROR y no imprimirá nada.

```
Status arbol_prtDesdeNivel (Arbol *pa, int nivel, FILE *pf) {
   if (pa==NULL || pf ==NULL) return ERROR

   return arbol_prtDesdeNivelREC (ROOT(pa), nivel, 0, pf);
}

Status arbol_prtDesdeNivelREC (Nodo *pn, int nivel, int actual, FILE *pf) {
  //dos condiciones de parada de la recursión
     // 1. he llegado al final de un sub-árbol
  if (pn == NULL) return ERROR;
    //2. he llegado a la profundidad deseada
  if (actual == nivel)  {
          elemento_imprime(pf, INFO(pn));
          return OK;
  }

  // Recursión general
  l = arbol_prtDEsdeNivelREC (IZQ(pn), nivel, actual+1, pf);
  r = arbol_prtDEsdeNivelREC (IZQ(pn), nivel, actual+1, pf);

 // el nodo esta a una profundidad menor que el nivel (parámetro de entrada) pero el arbol tiene una profundidad
superior
 if  (l==OK || r==OK)  {
             elemento_imprime(pf, INFO(pn));
             return OK;
  }
  return ERROR;
}
```

10. Escriba el pseudocódigo y las rutinas en C correspondientes para recorrer un árbol binario en orden previo y en orden posterior. Suponga, para ello, los tipos y estructuras de datos definidas en el ejercicio 3.

11. Dibujar los árboles de expresión que representan las siguientes expresiones
    (A + B) / (C - D);
    A + B + C / D;
    A – (B -(C - D) / (E + F));
    (A + B) * ((C + D) /(E + F));

12. Construir algorítmicamente el árbol de expresión a partir de las siguientes expresiones sufijo:
    A B C * D - E F / G / - * ;
    A B + C D - / E F * G - - ;
    A B C D - - E F G + + * * ;

13. Utilizando los árboles obtenidos en el problema anterior, obtenga las expresiones prefijo e infijp correspondientes a las expresiones sufijo.

14. Supóngase que al recorrer un árbol binario en preorden se obtiene la siguiente lista
    G B Q A C K F P D E R H ;

15. Dibuje una posible estructura de nodos del árbol. ¿Hay más?

16. Construir el árbol binario de búsqueda a partir de las listas siguientes

G B Q A C K F P D E R H ;
A B C D E F G H;
15 22 12 35 31 13 10 27 6 9 25;

17. En un árbol binario de búsqueda se define el predecesor de un dato D como el mayor dato D' del árbol tal que D' < D y su sucesor como el menor dato del árbol tal que D' > D (por ejemplo en el problema 11.c el predecesor de el 15 es el 13 y su sucesor el 22). Sobre los arboles del problema anterior, encontrar los predecesor y el sucesor de cada uno de los elementos.

18. Escribir el código de C de un algoritmo que devuelva un puntero al Elemento con menor campo info de un árbol binario de búsqueda. Dar un procedimiento recursivo y otro no recursivo

NODO_AB *nodoMinimo (ARBOL *pa)

```
// No recursivo

Elemento *nodoMinimo (ARBOL *pa) {
 NODE_AB *pn;

 if (pa==NULL) return NULL;
 if (pa->root==NULL) return NULL;

for (pn=pa->root; LEFT(pn)!= NULL; pn=LEFT(pn));

return INFO(pn);
}
```

```
// Recursivo

Elemento *nodoMinimo( const ARBOL *pa){
 if (pa==NULL) return NULL;
 return nodoMinimoRec (pa->root);
}

Elemento *nodoMinimoRec (NODO  *pa) {
 if (pa==NULL) return NULL;
 if (LEFT(pa)==NULL) return INFO(pa);

 return nodoMinimoREc (LEFT(pa));
}
```

19. Dar el pseudocódigo y el código C de una función que reciba un nodo de un árbol binario de búsqueda y devuelva el compo info de su nodo sucesor. Para ello, considere que en la estructura de datos utilizada para representar los nodos del árbol existe un campo adicional que permite acceder al padre de un nodo dado.

```
ARBOL sucesor (NODO a)
 if abVacio(a)==NULL return NULL
 //si el nodo tiene subarbol derecho devuelve el
 //minimo
 if abVacio (DER(a) == FALSE)
   return nodoMinimo( DER(a))

 //si no asciendo. El sucesor es el padre del
 //primer hijo izquierdo que encuentro al ascender
 P = PADRE(a)
 while abvacio(P)==FALSE AND RIGHT(P)==a
     a = P
     P = PADRE(P)
 return INFO(P)
```

```
Elemento *sucesor (NODO *a) {
 if (a==NULL) return NULL

 if (DER(a) != NULL)
   return nodoMinimo (DER(a))

 P = PADRE(a)
 while (abvacio(P)==FALSE && RIGHT(P)==a )
     a = P
     P = PADRE(P)
return INFO(P)
}
```

20. Dos árboles binarios son topológicamente similares si ambos están vacíos o, si ambos no están vacios, sus subárboles izquierdos son similares y sus

subárboles derechos son similares. De un ejemplo de árboles similares de profundidad mayor o igual a dos.

a. Escriba el pseudocódigo de un algoritmo para determinar si dos árboles son similares.

b. Utilizando las estructuras de datos y macros en C del ejercicio 3 proporcione el código C de una función BOOL sonSimilares (ARBOL T1, ARBOL T2) que implemente el algoritmo anterior.

```
//PsC

BOOL sonSimilares (ARBOL p1, ARBOL p2) {
      if (esVacio(p1)==TRUE && esVacio(p2)==TRUE ) return TRUE;
      if (esVacio(p1)==TRUE || esVacio(p2)==TRUE) return FALSE;

      l = sonSimilares(LEFT(p1), LEFT(p2));
      r = sonSimilares(RIGHT(p1), RIGHT(p2));
      return (l && r);
}
```

```
//Codigo C

BOOL sonSimilares (ARBOL  *p1, ARBOL *p2) {
     if (p1==NULL || p2==NULL) return FALSE;
      return sonSimilaresREc (p1->root, p2->root);
}

BOOL sonSimilaresREc (NODO  *p1, NODO *p2) {
      BOOL l,r;
      if (p1==NULL && p2==NULL) return TRUE;
      if (p1==NULL || p2==NULL) return FALSE;

      l = sonSimilaresREc(LEFT(p1), LEFT(p2));
      r = sonSimilaresREc(RIGHT(p1), RIGHT(p2));
      return (l && r);
}
```

21. Escriba el pseudocódigo de un algoritmo que acepte un árbol binario y cree un nuevo árbol binario copia del anterior. Dar, a continuación, el código C del algoritmo propuesto. Suponga, para ello, los tipos y estructuras de datos definidas en el ejercicio 3.

```
// Pseudocódigo

BOOL copiaAB (ARBOL orig, ARBOL dest)
 if abVacio(orig)==TRUE return TRUE;

 dest = crear_nodo();
 if dest==NULL return FALSE;
 elemento_copiar(INFO(orig),  INFO(dest));

 l = copiaAB (LEFT(orig), LEFT(dest));
 r = copiaAB (RIGHT(orig), RIGHT(dest));
 return l AND r;
```

```
// CODIGO C
#define ROOT(p) (p)->root
#define  LEFT(p) (p)->left
#define RIGHT(p) (p)-> right
#define INFO(p) (p)->info

ARBOL  *copiaAB (const ARBOL *orig){
 ARBOL *dest;
 if (orig==NULL) return NULL;

 dest = arbol_crear();
 if (dest==NULL) return NULL;

 r = copiaABRc (&ROOT(orig),  &ROOT(dest));
 if (r==FALSE) {
    arbol_liberar (dest);
    return NULL;
 }
 return dest;
}

BOOL copiaABRc (const Node **orig, Node  **dest) {
BOOL l, r;
Elemento *ele;
 if (*orig==NULL) return TRUE;

 *dest = node_crear();
 if (*dest==NULL) return FALSE;

ele = elemento_copiar( INFO(*orig),  INFO(*dest));
if (ele==NULL) return FALSE;
```

```
l = copiaABRc (&LEFT(*orig), &LEFT(*dest));
r = copiaABRc (&RIGHT(*orig), &RIGHT(*dest));

return (l && r);}
```

22. Escriba el pseudocódigo de un algoritmo que acepte un árbol binario y lo modifique de forma que sea la imagen refleja del original (es decir, un árbol en el que todos los subárboles derechos del árbol original son ahora subárboles izquierdos y viceversa). Dar, a continuación, el código C del algoritmo propuesto.

```
// Pseudocódigo (versión 1)

void reflejo (ARBOL pa)
     if abVacio(pa)==TRUE return;

     aux =LEFT(pa)
     LEFT(pa) = RIGHT(pa)
     RIGHT(pa) = aux

     reflejo(LEFT(pa))
     reflejo(RIGHT(pa))
     return

/ / Pseudocódigo (versión 2). Mas compacto

void reflejo (ARBOL pa)
    if abVacio(pa) ==FALSE
       aux =LEFT(pa)
       LEFT(pa) = RIGHT(pa)
       RIGHT(pa) = aux

     reflejo(LEFT(pa))
     reflejo(RIGHT(pa))

     return
```

```
// Código C (versión 1)

#define ROOT(p) (p)->root
#define LEFT(p) (p)->izq
#define RIGHT(p) (p)->right

typedef struct _NODO {
  Elemento *info;
  struct _NODO *izq, *der;
}

 struct _ARBOL {
   Nodo *root;
 }

void reflejo (ARBOL *pa) {
 if (*pa==NULL) return;
 return reflejoRec (&ROOT(pa) );
}

void reflejoRec (Nodo **pa) {
     NODE_AB *pn;
     if (*pa==NULL) return;

     pn =LEFT(*pa);
     LEFT(*pa) = RIGHT(*pa);
     RIGHT(*pa) = pn;

     reflejoREc(&LEFT(*pa));
     reflejoREc(&RIGHT(*pa));
     return;
}
```

23. El número de nodos n en un árbol binario completo de altura d es,

$$n = \sum_{j=0}^{d} 2^j = 2^{d+1} - 1 \qquad (1)$$

1. Supongase que tenemos un árbol binario completo con 1024 nodos ($1024=2^{10}$). ¿Cúal sería la altura del árbol?¿Cuantos nodos hay en el último nivel?. Compara estos números con los obtenidos en el ejercicio 4 de la práctica P4.

Según la fórmula (1) la altura del árbol será:

$$1024 = 2^{10} = 2^{d+1} - 1$$

$$2^{10} \sim 2^{d+1}$$

Aplicando logaritmos de base 2 a uno y otro lado de la expresión anterior obtenemos:

$$\log_2 (2^{10}) = \log_2 (2^{d+1})$$

utilizando las propiedades de los logarítmos,

$$10 = d+1$$

Por tanto la profundidad es d = 9. El número de nodos en la última capa es $2^9 = 512$

2. Demostrar que la fórmula (1) es correcta.

Ayuda: Se debe entender como se obtiene la fórmula, pero para demuestrarlo se debe utilizar inducción.

Sea T unárbol binario completo de altura h. Hay que demostrar que el número de nodos de T es $2^{h+1}-1$. Se demostrá por inducción.

Caso Base:

La altura de un árbol con un único nodo es 0 (h=0). Según la fórmula (1) el número de nodos de T es $2^{0+1}-1 = 2 - 1 = 1$. Por tanto la fórmula es correcto cuando el árbol tiene un único nodo.

Caso General:

Sea X un árbol binario de altura p e Y un árbol binario de altura q. Supongamos que el árbol binario T está formado por un nodo raiz, el subárbol izquierdo X y el subárbol derecho Y.

Al ser T un árbol binario completo, la altura de su subárbol izquierdo debe ser igual a la altura del subárbol derecho, p = q. Además, por definicón de altura (número de aristas), la altura de T debe ser uno mas que la de sus subárboles

$$h = p+1 \qquad (2)$$

El número de nodos en el subárbol X es por hipótesis inductiva (fórmula 1) $2^{p+1}- 1$, que en virtud de (2) podemos expresar (3) como $2^{h}-1$

El número de nodos en T es el número de nodos en X mas el número de nodos en Y mas 1, esto es,

$$n = 2^{h}- 1 + 2^{h}- 1 + 1$$

Sacando factor común obtenemos que $n = 2^{h+1} - 1$ tal y como queríamos demostrar.

24. Demostrar que la aplicación del algoritmo de orden medio sobre un árbol binario de búsqueda produce una secuencia ordenada.

Ayuda: Demostrar por inducción (hecho en clase)

25. *Estime* el número de accesos *necesarios* para buscar un dato a un árbol binario de búsqueda de n nodos. ¿Cuál sería el número máximo de accesos? Justifique la respuesta.

Ayuda: Piensa y utliza el resultado del problema 19.