

**Universidad San Carlos de Guatemala  
Centro Universitario de Oriente -CUNORI-  
Ingeniería en Ciencias y Sistemas**

Manejo e Implementación de Archivos  
Ing, Indira Valdes

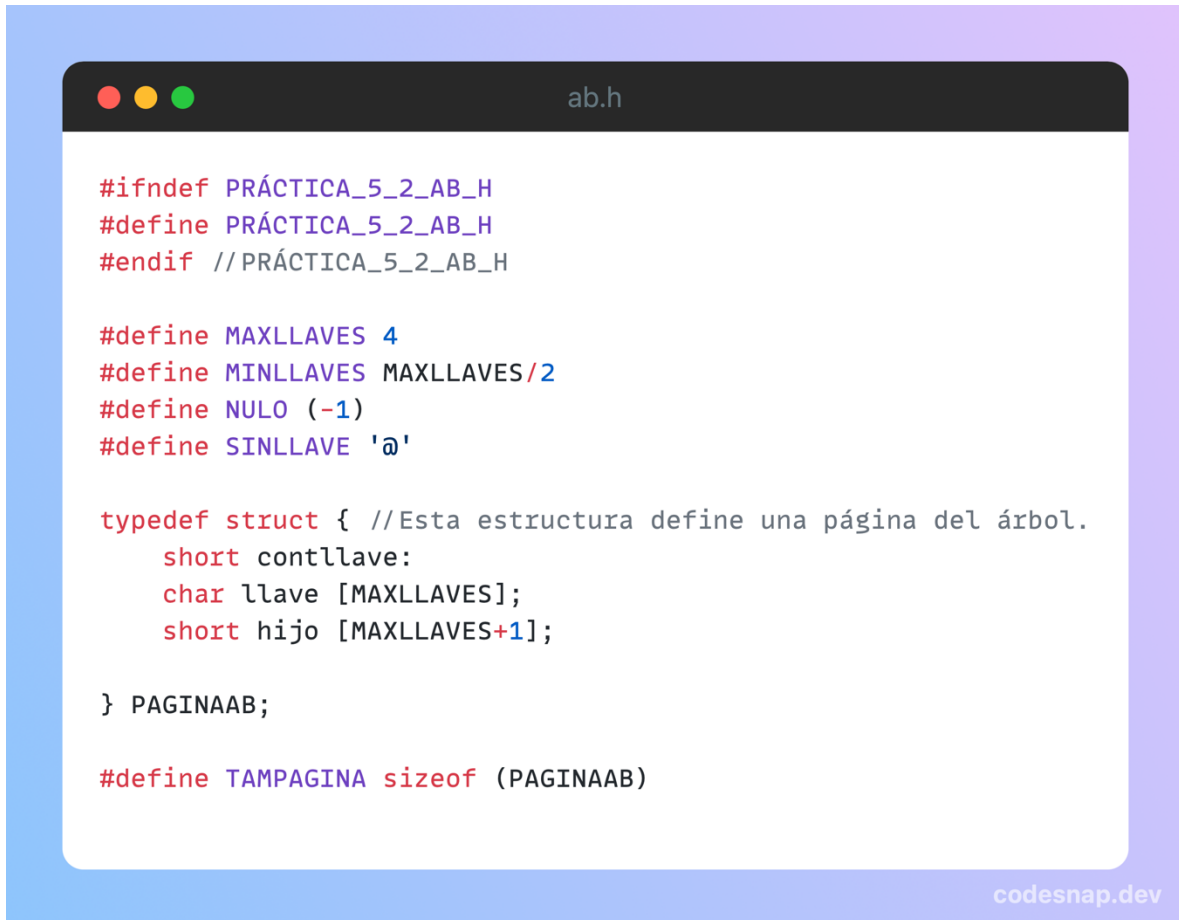
**Practica 5.2 y 5.1 B-TREE**

**Mynor Bezaleel Ramos González – 201944540**

**29 de septiembre de 2023, Chiquimula**

## 5.2

ab.h



```
#ifndef PRÁCTICA_5_2_AB_H
#define PRÁCTICA_5_2_AB_H
#endif // PRÁCTICA_5_2_AB_H

#define MAXLLAVES 4
#define MINLLAVES MAXLLAVES/2
#define NULO (-1)
#define SINLLAVE '@'

typedef struct { //Esta estructura define una página del árbol.
    short contllave;
    char llave [MAXLLAVES];
    short hijo [MAXLLAVES+1];

} PAGINAAB;

#define TAMPAGINA sizeof (PAGINAAB)
```

codesnap.dev

Proporciona las definiciones necesarias para representar una pagina en un arbol binario o arbol b y que tienen un numero limite de llaves definido por MAXLLAVES. Y PAGINAAB tiene informacion sobre la cantidad de llaves, las llaves mismas y los numeros de registros de los hijos de la pagina.

## manejador.c

```
manejador.c

#include <stdio.h>
#include <ab.h>

main () { // main function
    int promovido;

    short raiz,
           nrr_plomo;

    char llave_plomo
          llave;

    if (abreab())
        raiz = tomarraiz ();
    else
        raiz = crea-arbol ();

    while ((llave = getchar ()) != 'q') {
        promovido = inserta (raiz, llave, &nrr_plomo, &llave_plomo);
    }
    cierraab ();
}
```

codesnap.dev

El código se utiliza para interactuar con un árbol binario almacenado en un archivo, el cual permite la inserción de llaves en el árbol a través de la entrada estándar.

## inserta.c

```
inserta.c

#include <ab.h>

inserta (nrr, llave, hijo_d_plomo, llave_plomo) //Esta función inserta una llave en el árbol.
short nrr
    *hijo_d_plomo
char llave
    *llave_plomo;

{
    PAGINAAB pagina,
    pagnueva; //pagnueva es la página que se crea al dividir una página llena.

    int encontro //Encontro es una bandera que indica si se encontró la llave en la página.
    short poss,
    nrr_p_a;
    char llave_p_a;

    if (nrr == NULO) { //Si el árbol está vacío, se crea la raíz.
        *llave_plomo = llave;
        *hijo_d_plomo = NULO;
        return (SI);
    }

    leeab (nrr, &pagina); //Se lee la página.
    encontro = busca_nodo (llave, &pagina, &poss);
    if (encontro){ //Si se encontró la llave en la página, se imprime un mensaje de error.
        printf ("Error: intento de insercion de llave duplicada: %c\n\007", llave);
        return (0);
    }

    promocion = inserta (pagina.hijo[pos], llave, &nrr_p_a, &llave_p_a); //Se inserta la llave en la página.
    if (!promocion) //Si no hay promoción, se regresa un cero.
        return (NO);
    if (pagina.contllave < MAXLLAVES) { //Si la página no está llena, se inserta la llave.
        ins_en_pag (llave_p_a, nrr_p_a, &pagina);
        escribeab(nrr, &pagina);
        return (NO);
    } else { //Si la página está llena, se divide la página.
        divide(llave_p_a, nrr_p_a, &pagina, nrr_plomo, llave_plomo, &pagnueva);
        escribeab(nrr, &pagina);
        escribeab(*hijo_d_plomo, &pagnueva);
        return (SI);
    }
}
```

codesnap.dev

El código implementa la función “inserta” se utiliza para mantener un árbol binario (que el árbol siga siendo un árbol binario de búsqueda después de realizar insercciones) al insertar llaves en él.

## abes.c

El código está diseñado para administrar un árbol binario almacenado en un archivo persistente, permitiendo la apertura, cierre, lectura y escritura de páginas y la manipulación de la raíz del árbol.

```
#include "ab.h"
#include <stdio.h>
#include <arches.h>

int daab; //daab es el descriptor del archivo que contiene el árbol.

abreab(){ //Esta función abre el archivo que contiene el árbol.
    daab = open ("arbolb.dat", READWRITE);
    return (daab > 0);
}

cierraab(){ //Esta función cierra el archivo que contiene el árbol.
    close (daab);
}

short tomarraiz (){ //Esta función toma la raíz del árbol.
    short raiz;
    long lseek ();
    lseek (daab, 0L, 0);
    if (read (daab,&raiz, 2)==0){ //Si el archivo está vacío, se imprime un mensaje de error.
        printf ("Error: archivo vacío\n\007");
        exit (1);
    }
    return (raiz);
}

colocarraiz (raiz) //Esta función coloca la raíz del árbol.
short raiz;
{
    long lseek ();
    lseek (daab, 0L, 0);
    write (daab, &raiz, 2);
}

short crea_arbol(){ //Esta función crea el árbol.
    char llave;
    daab = creat ("arbolb.dat", PMODE);
    close(daab);
    abreab();
    llave = getchar();
    return (crea_raiz(llave, NULO, NULO));
}

short tomapag() //Esta función toma una página del árbol.
{
    long lseek(), dir;
    dir = lseek(daab, 0L, 2) -2L;
    return ((short) dir/ TAMPAGINA);
}

leeab(nrr, apunt_pagina) //Esta función lee una página del árbol.
short nrr;
PAGINAAB *apunt_pagina;
{
    long lseek(), dir;
    dir = (long)nrr * (long)TAMPAGINA + 2L;
    lseek(daab, dir, 0);
    return (read(daab, apunt_pagina, TAMPAGINA));
}

escribreab (nrr, apunt_pagina) //Esta función escribe una página del árbol.
short nrr;
PAGINAAB *apunt_pagina;
{
    long lseek(), dir;
    dir = (long)nrr * (long)TAMPAGINA + 2L;
    lseek(daab, dir, 0);
    return (write(daab, apunt_pagina, TAMPAGINA));
}
```

# utilab.c

```
#include "ab.h"

crea-raiz(llave, izq, der)
char llave;
short izq, der;
{ //Esta función crea la raíz del árbol.
    PAGINAAB pagina;
    short nrr;
    iniciapag(&pagina);
    pagina.llave[0] = llave;
    pagina.hijo[0] = izq;
    pagina.hijo[1] = der;
    pagina.contllave = 1;
    escribepag(nrr, &pagina);
    colocaliza(nrr);
    return (nrr);
}

iniciapag(a_pagina) //Esta función inicializa una página del árbol.
PAGINAAB *a_pagina;
{ //Esta función inicializa una página del árbol.
    int j;
    for (j=0; j<MAXLLAVES; j++) {
        a_pagina->llave[j] = SINLLAVE;
        a_pagina->hijo[j] = NULO;
    }
    a_pagina->hijo[MAXLLAVES] = NULO;
}

busca_nodo (llave, a_pagina, pos) //Esta función busca una llave en una página del árbol.
char llave;
PAGINAAB *a_pagina;
short *pos; //pos es un apuntador a la posición de la llave en la página.
{
    int i;
    for (i=0; i<a_pagina->contllave && llave > a_pagina->llave[i]; i++);
    *pos = i;
    if (*pos < a_pagina->contllave && llave == a_pagina->llave[*pos])
        return (SI);
    else
        return (NO);
}

ins_en_pag(llave, hijo_d, a_pagina) //Esta función inserta una llave en una página del árbol.
char llave;
short hijo_d;
PAGINAAB *a_pagina;
{ //Esta función inserta una llave en una página del árbol.
    int i;
    for(i=a_pagina->contllave; llave < a_pagina->llave[i-1] && i>0; i--) {
        a_pagina->llave[i] = a_pagina->llave[i-1];
        a_pagina->hijo[i+1] = a_pagina->hijo[i];
    }
    a_pagina->contllave++;
    a_pagina->llave[i] = llave;
    a_pagina->hijo[i+1] = hijo_d;
}

divide (llave, hijo_d, a_pagant, llave_promo, hijo_d_promo, a_panue) //Esta función divide una página del árbol.
char llave, *llave_promo;
short hijo_d, *hijo_d_promo;

PAGINAAB *a_pagant, *a_pagnue;
{//Esta función divide una página del árbol.
    int i;
    short mitad;
    char llavesaux[MAXLLAVES+1];
    short caraux[MAXLLAVES+2];

    for (i=0; i<MAXLLAVES; i++) { //Se copian las llaves y los hijos de la página en un arreglo auxiliar.
        llavesaux[i] = a_pagant->llave[i];
        caraux[i] = a_pagant->hijo[i];
    }
    llavesaux[i] = a_pagant->llave[i];
    for (i=MAXLLAVES; llave < llavesaux[i-1] && i>0; i--) { //Se inserta la llave en el arreglo auxiliar.
        llavesaux[i] = llavesaux[i-1];
        caraux[i+1] = caraux[i];
    }
    llavesaux[i] = llave;
    caraux[i+1] = hijo_d;

    *hijo_d_promo = tomapag();
    iniciapag(a_panue); //Se inicializa la página nueva.

    for(i=0; i<MINLLAVES; i++) { //Se copian las llaves y los hijos del arreglo auxiliar en la página antigua y en la página nueva.
        a_pagant->llave[i] = llavesaux[i];
        a_pagant->hijo[i] = caraux[i];
        a_pagnue->llave[i] = llavesaux[i+MINLLAVES];
        a_pagnue->hijo[i] = caraux[i+1+MINLLAVES];
        a_pagant->llave[i+MINLLAVES]=SINLLAVE;
        a_pagant->hijo[i+1+MINLLAVES]=NULO;
    }

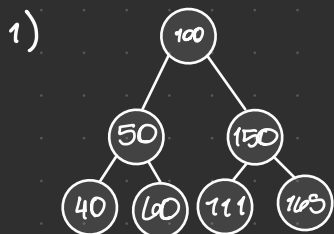
    a_pagant->hijo[MINLLAVES] = caraux[MINLLAVES];
    a_pagant->hijo[MINLLAVES] = caraux[i+1+MINLLAVES];
    a_pagnue->contllave = MAXLLAVES-MINLLAVES;
    a_pagant->contllave = MINLLAVES;

    *llave_promo = llavesaux[MINLLAVES];
}
```

codessnap.dev

El código funciona para gestionar un archivo que almacena información de una estructura de un árbol. Las funciones realizan tareas como crear la raíz del árbol, buscar llaves en una página, insertar llaves en una página, y dividir una página cuando se alcanza la capacidad máxima de llaves.

Analizar los recorridos en orden, preorden y postorden, utilice para creación de los recorridos la representación ligada del árbol nario.



NRR	Nodo	Izg.	Der.
0	100	1	4
1	50	2	3
2	40	-1	-1
3	60	-1	-1
4	150	5	6
5	111	-1	-1
6	165	-1	-1

In Orden:

40, 50, 60, 100, 111, 150, 165

Pre Orden:

100, 50, 40, 60, 150, 111, 165

Post Orden:

40, 60, 50, 111, 165, 150, 100

In Orden:

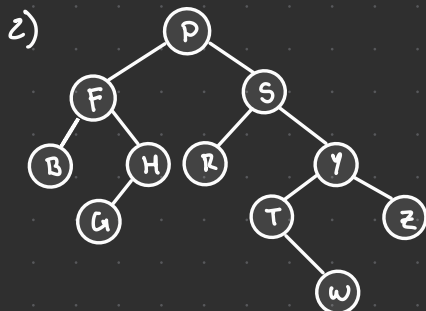
2, 1, 3, 0, 5, 4, 6

Pre Orden:

0, 1, 2, 3, 4, 5, 6

Post Orden:

2, 3, 1, 5, 0, 4, 6



In Orden:

B, F, G, H, P, R, S, T, W, Y, Z

Pre Orden:

P, F, B, H, G, S, R, Y, T, W, Z

Post Orden:

B, G, H, F, R, W, T, Z, Y, S, P

nrr	Nodo	Izg	Der
0	P	1	5
1	F	2	3
2	B	-1	-1
3	H	4	-1
4	G	-1	-1
5	S	6	7
6	R	-1	-1
7	Y	8	10
8	T	-1	9
9	W	-1	-1
10	Z	-1	-1

In Orden:

2, 1, 4, 3, 0, 6, 5, 8, 9, 7, 10

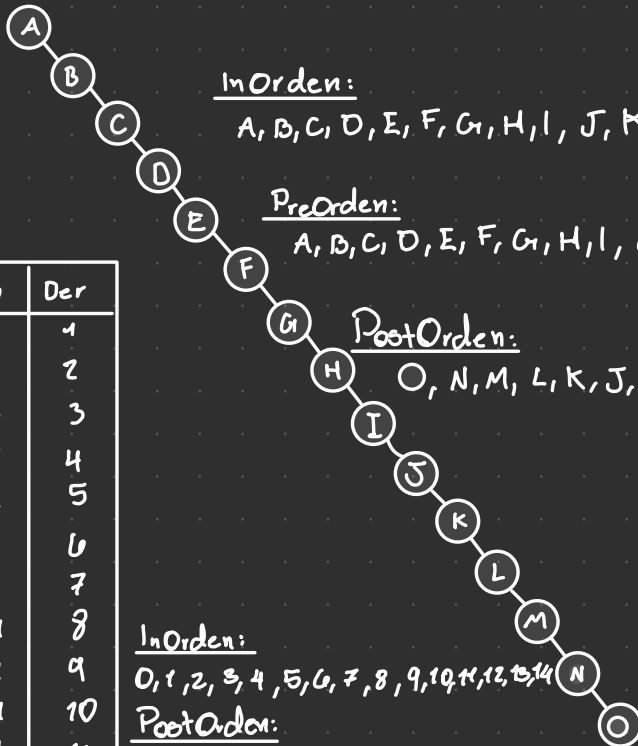
Post Orden:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Pre Orden:

2, 4, 3, 1, 6, 9, 8, 10, 7, 5, 0

3)



In Orden:

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O

Pre Orden:

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O

Post Orden:

O, N, M, L, K, J, I, H, G, F, E, D, C, B, A

In Orden:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14

Post Orden:

14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0

Pre Orden:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14

Pos Orden: hijo izquierdo  
hijo derecho  
Padre

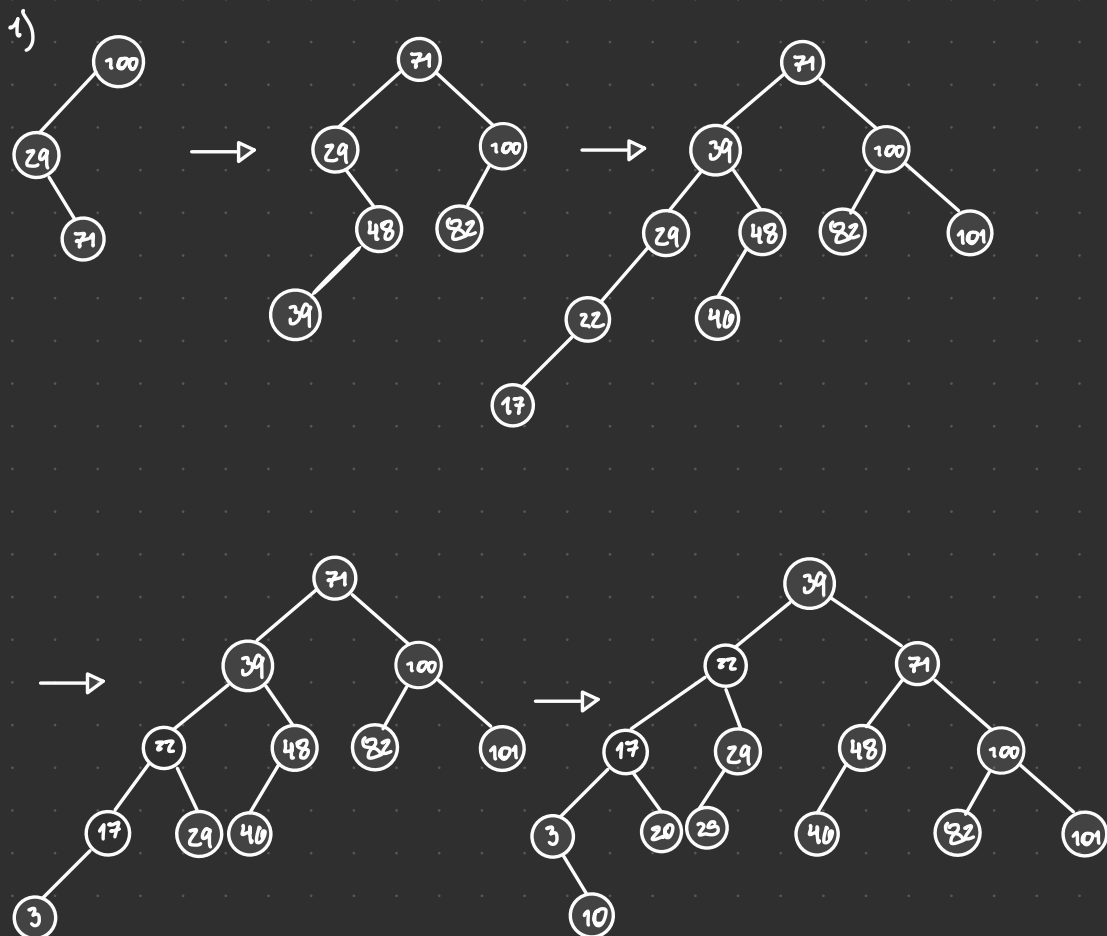
Pre Orden: Padre  
hijo izquierdo  
hijo derecho

In Orden: hijo izquierdo  
Padre  
hijo derecho

NIT	Nodo	Izq	Der
0	A	-1	1
1	B	-1	2
2	C	-1	3
3	D	-1	4
4	E	-1	5
5	F	-1	6
6	G	-1	7
7	H	-1	8
8	I	-1	9
9	J	-1	10
10	K	-1	11
11	L	-1	12
12	M	-1	13
13	N	-1	14
14	O	-1	-1



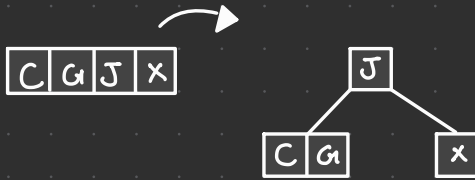
## Ejercicio 2: cree el árbol AVL en representación gráfica:



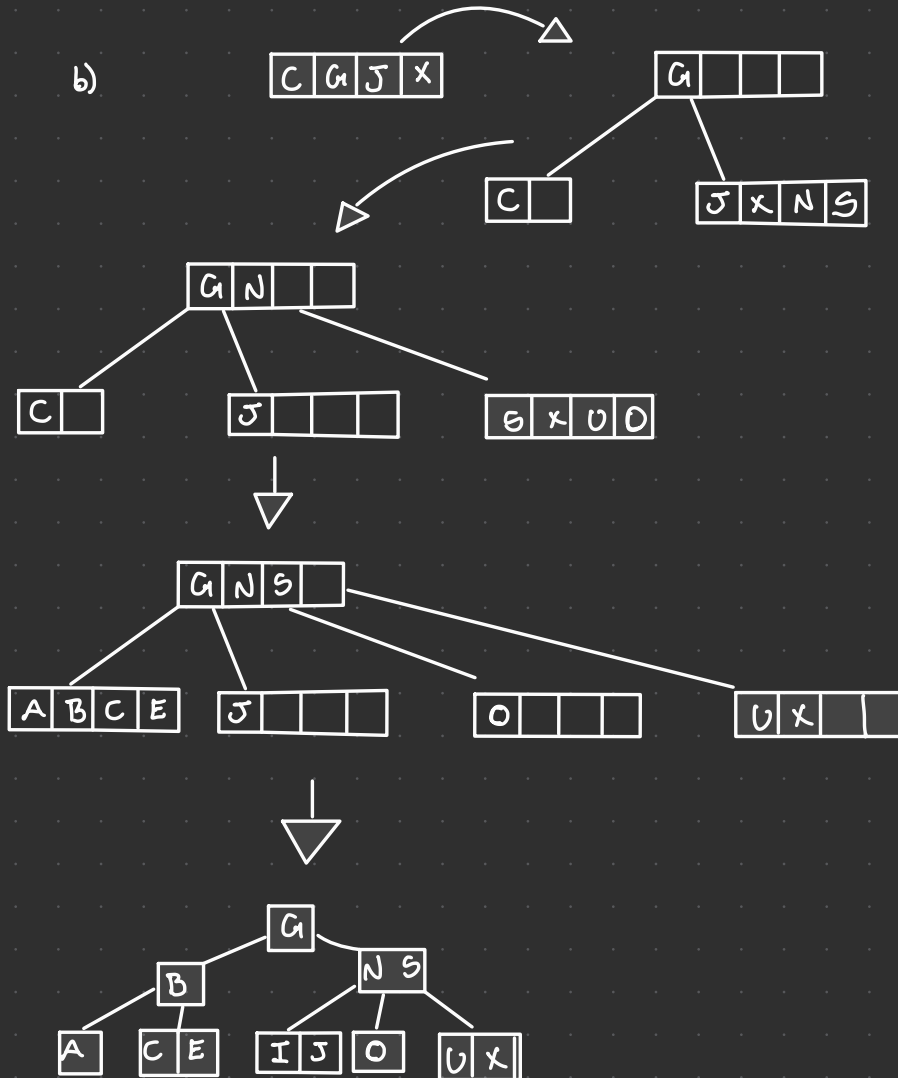
- 1) El árbol AVL se mantiene balanceado en todo momento. En cada nodo, la diferencia de altura entre sus subárboles izquierdo y derecho es como máximo 1.

Ejercicio 3: cree los árboles B de orden que resultan de cargar los siguientes conjuntos de llaves en orden:

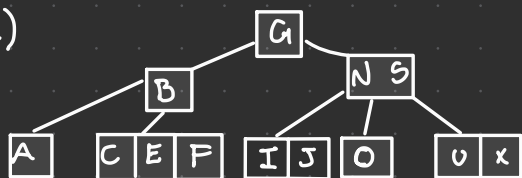
a)



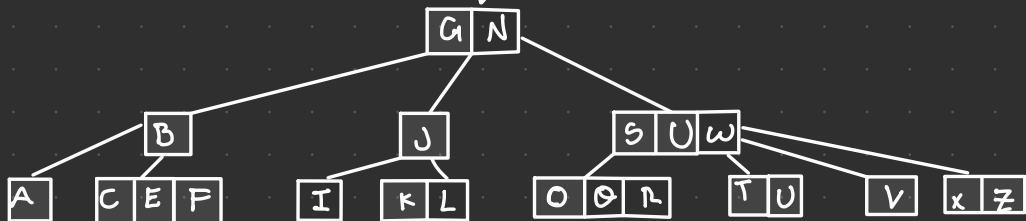
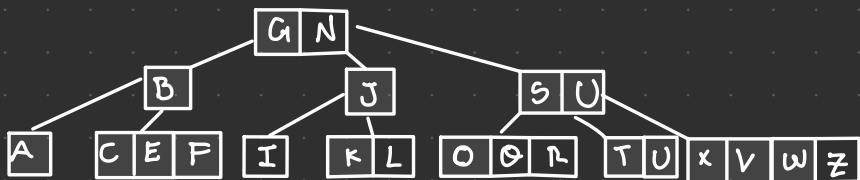
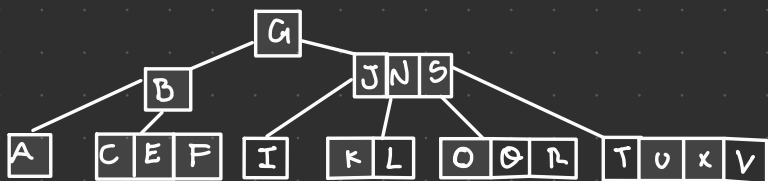
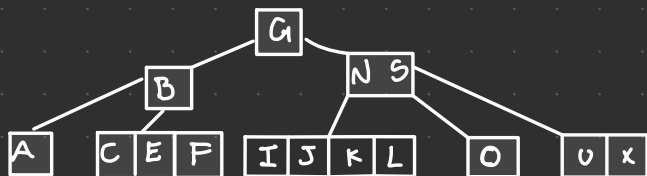
b)



c)



d)



## Ejercicio 5: conteste las siguientes preguntas:

- 1) Numero de Acceso al disco.
- 2)
  - \* Tiempo de búsqueda eficiente.
  - \* Distribución uniforme de datos.
  - \* Optimización del Espacio.
  - \* Mantenimiento de la estructura de Datos.
  - \* Reducción de Costos de Almacenamiento y Acceso.
- 3)
  - \* Balance Automatico.
  - \* Mejor Rendimiento en operaciones.
  - \* Garantia de tiempo logaritmico.
  - \* Menos Acceso al disco.
  - \* Menor Fragmentación.

### 4) Nodo hoja Árbol B

- \* Llaves de Datos.
- \* Punteros a Datos.
- \* Puntero Nodo Sig.

### Nodo interno Árbol B

- \* Llaves indice
- \* Punteros a nodos hijo
- \* No contiene datos reales.
- \* Ningun puntero al nodo sig.