Algoritmos y estructuras de datos

Recuperatorio segundo parcial 2021

Tema: 1
Alumno: Gavilondo Gonzalo
Nota:

Ejercicio A (3 pts)	Ejercicio B (3 pts)	Ejercicio C (4 pts)	Total (10 pts)

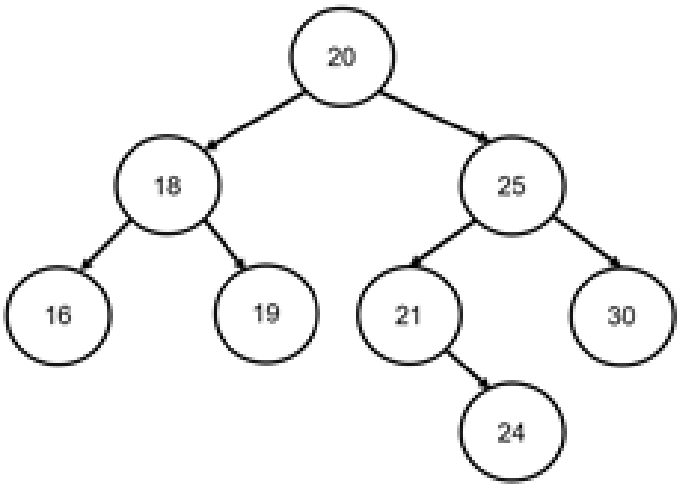
Se exige un mínimo del 40% de desarrollo correcto de cada tema para comenzar a tener puntaje en dicho tema.

Enunciados

Esta sección NO puede ser modificada por el alumno

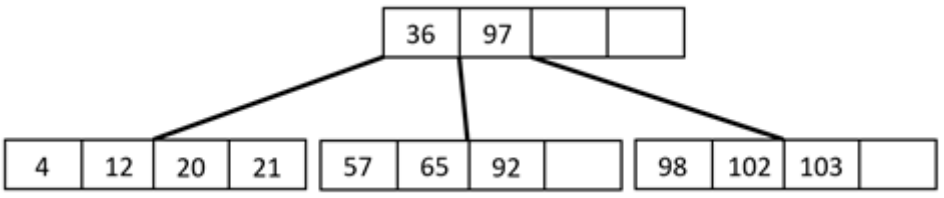
Ejercicio A

1) Dado el siguiente árbol AVL:



Eliminar 30 y balancear el árbol.

2) Dado el siguiente árbol B de orden 5:

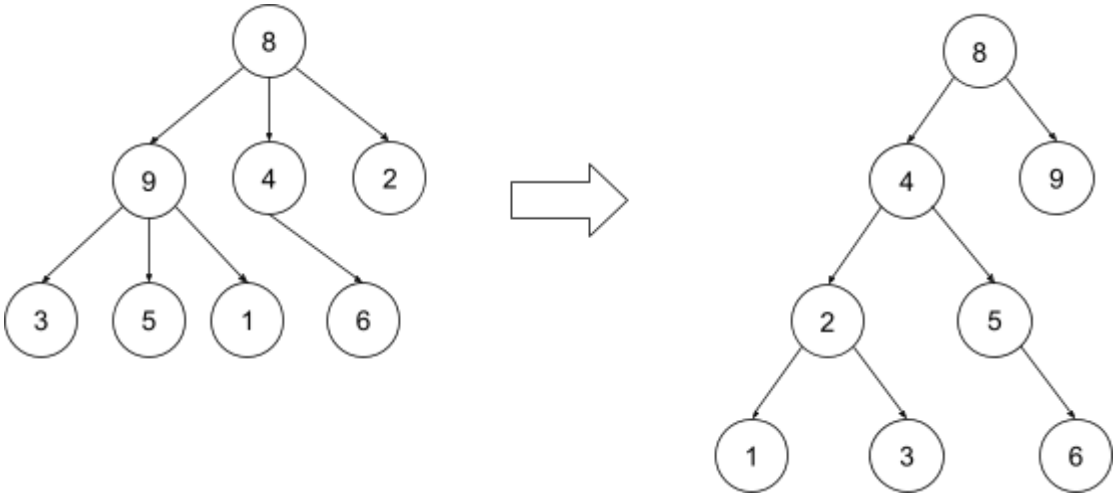


Insertar 104 y 110 verificando que se cumplan las propiedades de los árboles B.

Ejercicio B

Crear un árbol binario de búsqueda con los valores de todos los nodos de un árbol n-ario, siguiendo el recorrido mostrado en el ejemplo.
NOTA: Es parte de la evaluación reconocer el recorrido del árbol n-ario para formar el ABB, ya que diferentes recorridos forman árboles distintos.

Ejemplo:



La función debe seguir el siguiente protocolo:

```
btn *CrearABB (ntn *rootAN);
```

Teniendo en cuenta las siguientes estructura:

```
typedef struct ntNode ntn;
typedef struct ntList ntlist;

typedef struct ntList {
    ntn *node;
    ntlist *next;
} ntlist;

typedef struct ntNode {
    int value;
    ntlist *sons;
} ntn;
```

```
typedef struct btNode btn;
typedef struct btNode {
    int value;
    btn *left;
    btn *right;
} btn;
```

Consideraciones:

- En caso que la función llame a otras funciones debe incluir todas las funciones que utilice.
- **Si utiliza una estructura auxiliar debe estar definida.**

Ejercicio C

Desarrollar el código en C para:

Saber si un grafo dirigido y ponderado está incluido en otro. Teniendo en cuenta que los nombres de los vértices pueden estar en índices diferentes y que deben coincidir las aristas.

Entrada de la función principal:

- puntero a digrafo original (implementación a elección)
- puntero a digrafo a ser evaluado (implementación a elección)

Salida de la función principal:

- Valor entero 1 si el digrafo a ser evaluado está incluido en el original, o 0 si no lo está.

Consideraciones:

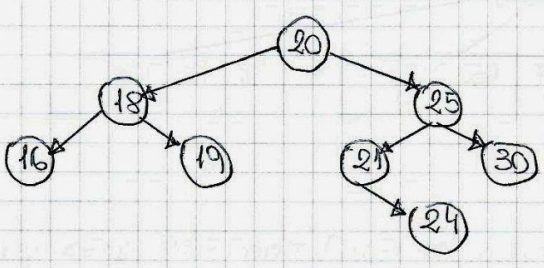
- Si el digrafo a ser evaluado NO está incluido se debe cortar el proceso al primer caso que no cumple la condición.
- NO debe utilizar **break** ni **return** en ciclos **for**.
- Puede utilizar cualquier implementación de grafos.
- Debe definir o transcribir todas las estructuras y algoritmos adicionales utilizados.
- Debe incluir todas las funciones utilizadas.

Soluciones

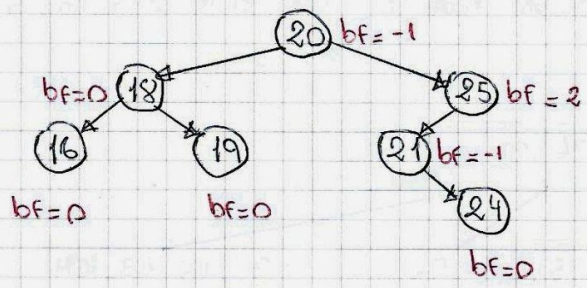
Aquí puede escribir el alumno

EJERCICIO A)

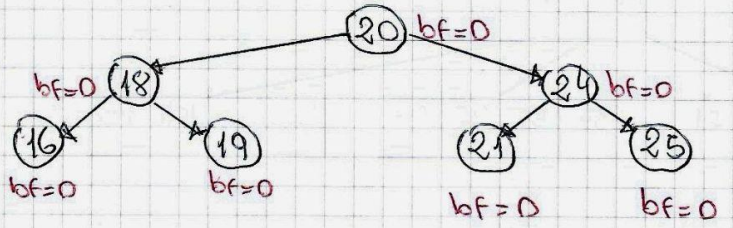
1)



Al eliminar el 30 nos queda el siguiente árbol:

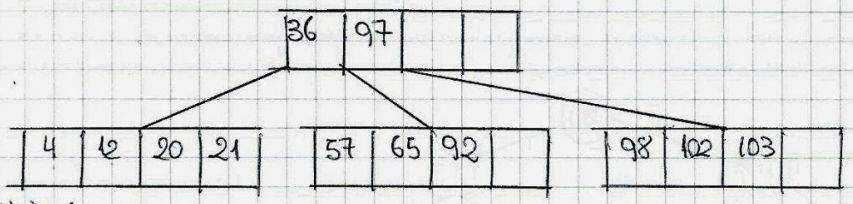


- Como vemos, el 25 está desbalanceado, como su hijo tiene bf=-1 (distinto signo) → hay que hacer una rotación previa, es decir, hay que hacer una rotación a izquierda entre el 24 y el 21, y luego una rotación a derecha entre el 24 y el 25.



ARBOL AVL RESULTANTE

2)



Propiedades:

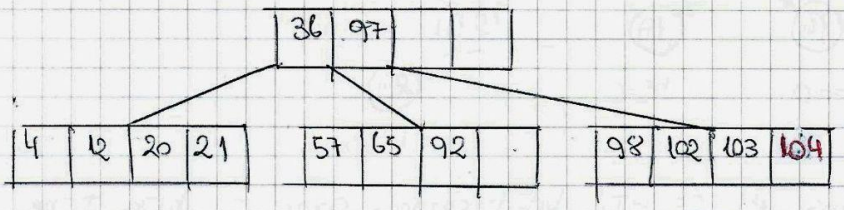
- Orden $m = 5$

a) Cada página (excepto la raíz y las hojas) tienen entre $\frac{m}{2} = 3$ y $m = 5$ hijos

b) Cada página contiene entre $\frac{m}{2} - 1 = 2$ y $m - 1 = 4$ elementos

c) La página raíz, o es una hoja o tiene entre 2 y $m = 5$ hijos.

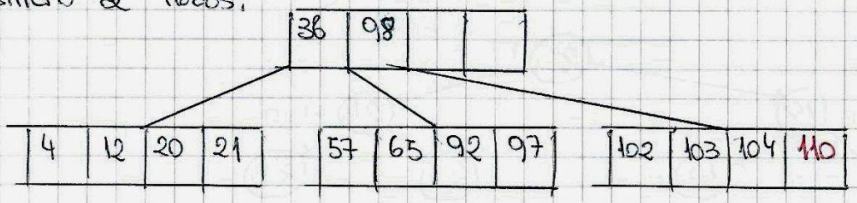
- Insertamos el 104



No hace falta cambiar nada, se siguen cumpliendo las propiedades.

- Insertamos el 110.

Tenemos que bajar el 97, subir el 98 e insertar el 110 en el último casillero de todos.



Se cumplen todas las propiedades.

EJERCICIO B)

```
/* Main */

int main()
{
    ntn *root = NULL;
    sample(&root);
    printf("ARBOL N ARIO \n\n");
    printNTN(root);
    btn *rootSBT = CrearABB(root);
    printf("\n\nARBOL SBT");
    printBTN(rootSBT);
    puts("\n");

    return 0;
}

/** Funciones implementadas **/
/**
      Nota: Utilice recorrido en amplitud, ya que me guio del ejemplo
utilizado,  si hubiese recorrido en profundidad, por ejemplo, a la
izquierda del 8 en vez de estar el “4”, se encontraria el “3”. Y asi se
formaria un arbol totalmente distinto.
**/
void _CrearABB(ntn *node, btn **nodeSBT)
 {
    if (node == NULL) return;

    insertValueSBT(nodeSBT, node->value); //Inserto el nodo
    ntq *q = createQueueNTN();
    enqueueNTN(q, node);
    while (!isEmptyQueueNTN(q))
    {
        ntn *aux = dequeueNTN(q);

        ntlist *l = aux->sons;
        while (l != NULL)
        {
            insertValueSBT(nodeSBT, l->node->value); //Inserto los hijos
            enqueueNTN(q, l->node);
            l = l->next;
        }
    }

 }
```

```
btn *CrearABB (ntn *rootAN)
{
    btn *rootSBT = NULL;
    _CrearABB(rootAN, &rootSBT);

    return rootSBT;
}
```

```
/** Codigo del arbol utilizado en el ejemplo **/
void sample (ntn **root)
{
    ntn *aux = NULL;
    ntn *aux2 = NULL;
    if (root != NULL)
    {
        *root = createNTN (8);
        aux = insertSonValueNTN(*root, 9);
        insertSonValueNTN(aux, 3);
        insertSonValueNTN(aux, 5);
        insertSonValueNTN(aux, 1);
        aux = insertSonValueNTN(*root, 4);
        aux2 = insertSonValueNTN(aux, 6);
        aux = insertSonValueNTN(*root, 2);

    }
}
```

Nota: Las funciones utilizadas después de arboles, ya sean binarios o n-arios son todas las vistas en la teoría.

EJERCICIO C)

```
/**Funcion main**/
int main()
{
    graph *g = ejemploDiGrafoA();
    graph *h = ejemploDiGrafoA();

    printf("Grafo 1: \n\n");
    graphPrint(g);
    puts("\n\n");
    printf("Grafo 2: \n\n");
    graphPrint(h);

    int resultado = grafoIncluidoenOtro(g, h);
    if(resultado == 1)
    {
```

```
        printf("El segundo grafo esta incluido en el primero.\n");
    }
    else
    {
        printf("El segundo grafo no esta incluido en el primero.\n");
    }

    return 0;
}

/**Funcion principal implementada**/
int grafoIncluidoenOtro(graph *g, graph *h)
{
    if(!isDigraph(g)) return 0;
    if(!isDigraph(h)) return 0;

    //Siendo g el grafo original y h el grafo que debe estar incluido.
    int result = 1;
    int vertex = 1; //Variable para controlar si son iguales los vertices
    int arco = 1;  //Variable para controlar que haya un arco en ambos
grafos entre los mismos vertices
    listStr *l = NULL;

    while(vertex == 1 && arco == 1)
    {

        for(int i = 0; i < graphSize(h); i++)
        {
            while(vertex != 0)
            {
                listStrAdd(&l, graphGetValueOf(g, i));
                if (listStrGetValue(l) != graphGetValueOf(h, i))
                {
                    vertex = 0;
                }
                listStrGetNext(l);
            }

        }

        int ii, jj;

        for(int i = 0; i < graphSize(h); i++)
        {
            for(int j = 0; j < graphSize(h); j++)
            {
                while(arco != 0)
```

```
                {
                    ii = graphGetVertexIndex(g, graphGetValueOf(h, i));
                    jj = graphGetVertexIndex(g, graphGetValueOf(h, j));
                    if((ii != NONE) && (jj != NONE) && (graphExistArc(g,
ii, jj) != 0) && (graphExistArc(h, i, j) != 0))
                    {
                        arco = 1;
                    }
                    else
                    {
                        arco = 0;
                    }
                }
            }
        }
        if(vertex == 0 || arco == 0)
        {
            result = 0;
        }
    }

    return result;
}


/**Estructuras y funciones adicionales **/

/*****************************************************************/
/*                      LISTA DE STRINGS                         */
/*****************************************************************

typedef struct ListStr listStr;
typedef struct ListStr
{
    char *value;
    listStr *next;

} listStr;


listStr *createListStrNode(char *value)
{
    listStr *result = (listStr*)malloc(sizeof(listStr));
    result->value = value;
    result->next = NULL;
    return result;
}
```

```
listStr *listStrGetNext(listStr *l)
{
    return l->next;
}

char *listStrGetValue(listStr *l)
{
    return l->value;
}

//A esta llama el usuario para agregar
void listStrAdd(listStr **l, char *value)
{
    if(!l) return;
    if((*l) == NULL)
    {
        (*l) = createListStrNode(value);
    }
    else
    {
        listStrAdd(&((*l)->next), value);
    }
}
int printListStr(listStr *l)
{
    if (!l) return 0;
    if (isEmptyListStr(l)) return 0;

    listStr *aux = l;
    while(aux != NULL)
    {
        printf("%s ", aux->value);
        aux = aux->next;
    }
    printf("\n");
    return 1;
}

int printListStrWay(listStr *l)
{
    if (!l) return 0;
    if (isEmptyListStr(l)) return 0;

    listStr *aux = l;
    if(aux != NULL)
    {
```

```
        printf("%s", listStrGetValue(aux));
        aux = listStrGetNext(aux);
    }
    while(aux != NULL)
    {
        printf(" --> %s", listStrGetValue(aux));
        aux = listStrGetNext(aux);
    }
    puts("\n");

    return 1;
}

/*************************************************************/
/*                    Funciones adicionales de grafos        */
/*************************************************************
int graphExistArc(graph *g, int indexV1, int indexV2)
{
    int result = 0;
    if(g->A[indexV1][indexV2] != 0 && g->A[indexV1][indexV2] != INF)
    {
        result = 1;
    }
    return result;
}

/**
    ¿Como saber si un grafo es Grafo o Digrafo?
    Nos tenemos que fijar si toda la matriz es simetrica.

        -Si no es simetrica -> Es Digrafo -> Devuelve un 1.
        -Si es simetrica -> Es Grafo -> Devuelve un 0.
**/
int isDigraph(graph *g)
{
    int result = 1;
    for(int i = 0; i < graphSize(g); i++)
    {
        for(int j = i; j < graphSize(g); j++)
        {
            if(graphExistArc(g, i, j) == graphExistArc(g, j, i))
            {
                if(graphCost(g, i, j) == graphCost(g, j, i))
                {
                    result = 0;
                }
            }
```

```
        }
    }
    return result;
}


/**
    Devuelve un valor del vertice donde se encuentra
**/
char *graphGetValueOf(graph *g, int index)
{
    char *result = NULL;
    if(graphValideIndex(g, index))
    {
        result = g->V[index]->value;
    }
    return result;
}

/**
    Busca el value de un grafo y devuelve el indice donde se encuentra
**/
int graphGetVertexIndex(graph *g, char *value)
{
    int i = 0;
    while( (i < graphSize(g)) && (strcmp(graphGetValueOf(g, i), value) !=
0) )
    {
        i++;
    }
    return (i < graphSize(g)) ? i : NONE;
}
```