Algoritmos y estructuras de datos

Segundo parcial 2021

Tema: 3
Alumno: Gonzalo Gavilondo
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

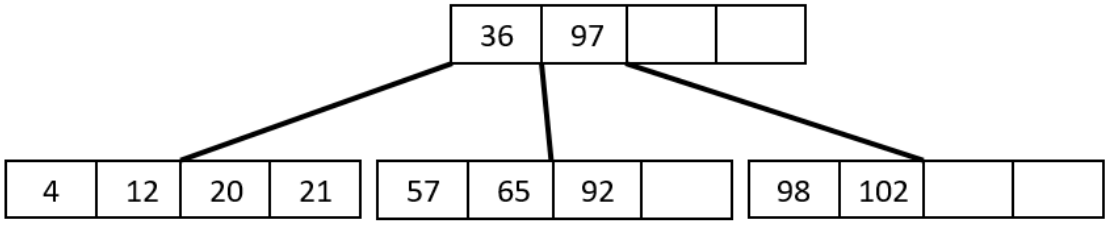
Ejercicio A

1) Dibujar un ABB con las siguientes claves: 9, 43, 58, 8, 3, 6, 1.

2) Balancear el árbol del punto 1) de modo de obtener un árbol AVL, especificando las rotaciones realizadas.

3) En el árbol AVL del punto 2) insertar las claves 60 y 7, balanceando el árbol en cada paso de ser necesario y especificando las rotaciones realizadas.

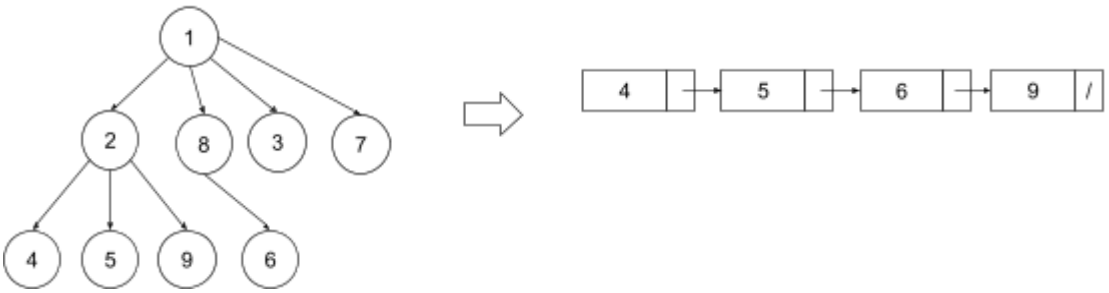
4) A partir del siguiente árbol B de orden 5, eliminar las claves 57 y 65. En cada paso dibujar el árbol resultante, indicando las operaciones realizadas.



Ejercicio B

Crear una lista ordenada con los valores de los nodos hijos de cada nodo par de un árbol n-ario.

Ejemplo:



La función debe seguir el siguiente protocolo:

```
list *hijosDePares (ntn *root);
```

Teniendo en cuenta las siguientes estructuras:

```
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 List list;
typedef struct List {
    int value;
    list *next;
} list;
```

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:

Crear un nuevo grafo a partir de otros 2 grafos, teniendo en cuenta los nombres (o claves) de los vértices.

Entrada de la función principal:

- puntero a grafo 1 (implementación a elección)
- puntero a grafo 2 (implementación a elección)

Salida de la función principal:

- puntero a nuevo grafo

Consideraciones:

- Los grafos no son dirigidos, ni ponderados.
- Los valores de las claves son “strings”, y en los grafos pueden encontrarse en índices diferentes. Por ejemplo: el grafo 1 tiene a “Madrid” con índice 0, pero en el grafo 2 “Madrid” está en el índice 3.
- Puede utilizar cualquier implementación.
- 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

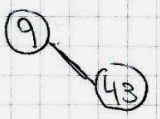
Gonzalo Gavilondo 40.666.672

1) Dibujar un arbol ABB con las siguientes claves: 9, 43, 58, 8, 3, 6, 1.

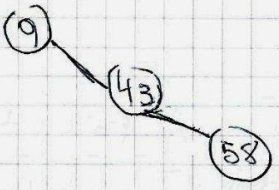
- Por propiedad de ABB, las claves menores al nodo raiz, se insertan a la izquierda y las mayores a la derecha.

- InserTamos el 9

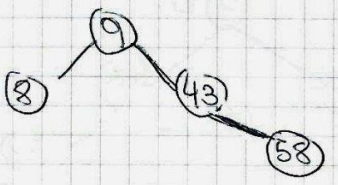
- InserTamos el 43, Como es 43>9 Va a la derecha



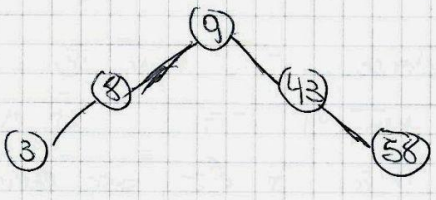
- InserTamos el 58, Como 58>9 y 58>43



- InserTamos el 8, Como 8<9 → inserTamos a la izquierda

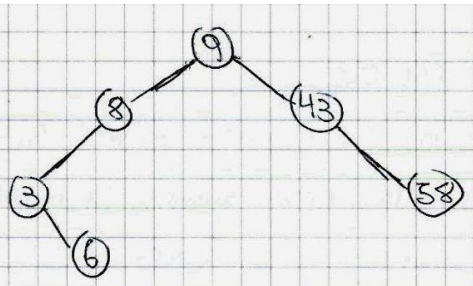


- InserTamos el 3, Como 3<9 y 3<8

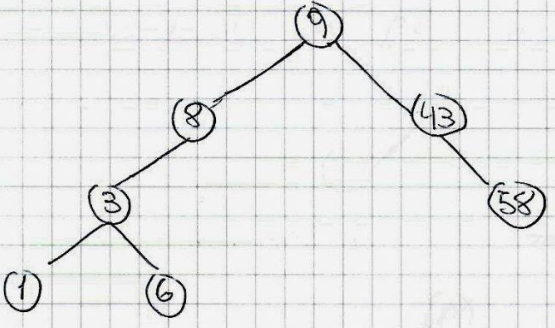


- InserTamos el 6, Como 6<9, 6<8 y 6>3 → inserTamos a la derecha de 3.

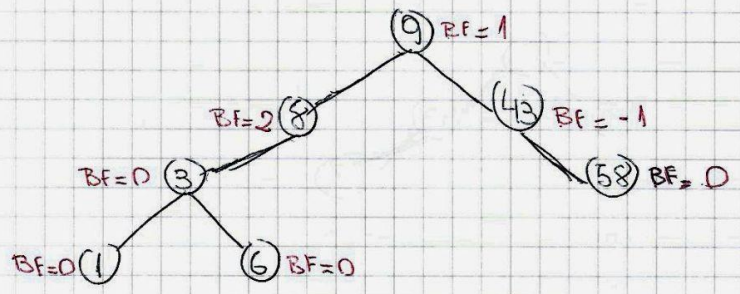
→



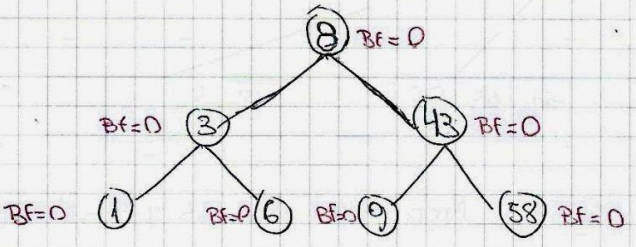
- InsertAmos el 1. Como 1<9, 1<8 y 1<3 → insertAmos a la izquierda del 3.



2) Primero vamos a ver el factor de balanceo de cada nodo para saber si el árbol está balanceado, $BF = |h(left) - h(right)| < 2$.



El único que está desbalanceado es el nodo (8) → hacemos una rotación a derecha donde el 8 pasa a ser el nodo raíz y el 9 pasa a ser hijo derecho de éste, pero va a quedar → desbalanceada la parte derecha → hacemos una rotación a izquierda entre el 43 y el 9. Resultando →

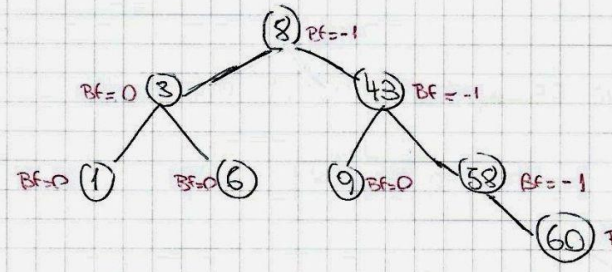


ARBOL AVL

- Rotamos a derecho el 8 con el 9
- Rotamos a izquierda el 9 con el 43.

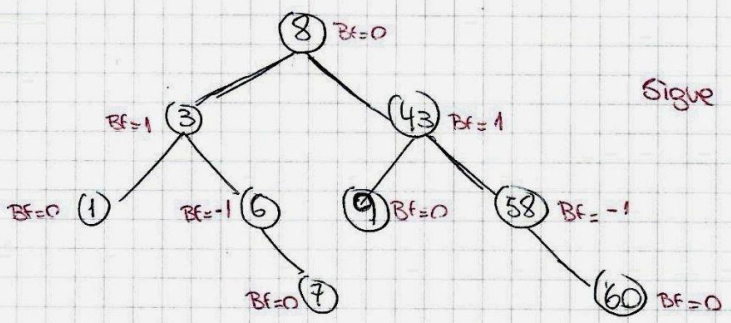
3) Ahora tenemos que insertar las claves: 60 y 7.

- InserTamos el 60



Sigue Balanceado

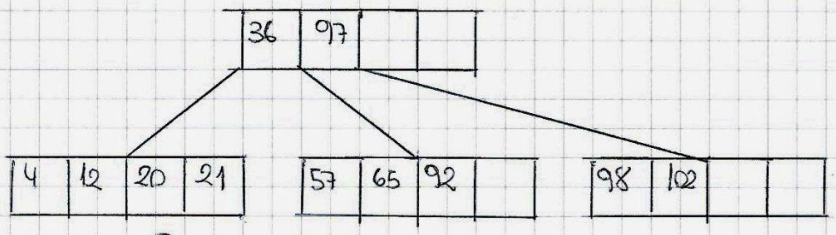
- InserTamos el 7.



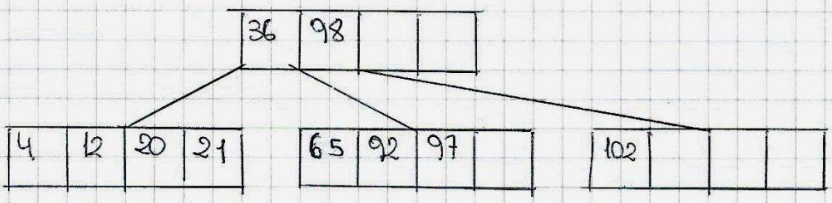
Sigue Balanceado

ARBOL AVL RESULTANTE.

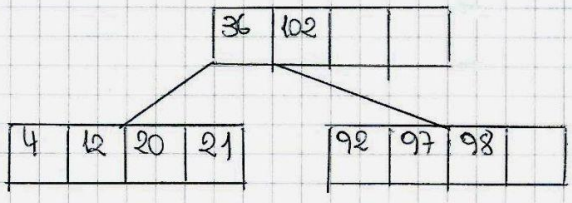
4)



- Si eliminamos el (57), bajo el (97) y subo el menor de la derecha, es decir, el (98). (Underflow).



- Si eliminamos el (65) → bajo el (98) y sube el menor de la derecha. (Underflow).



ARBOL B RESULTANTE.

Ejercicio B

Función principal: list *hijosDePares(ntn *root)

```
void _hijosDePares(ntn *nodo, list *result)
{
    if(!nodo) return;

    //Si el valor de los nodos hijos no es par, que siga buscando
    if(nodo->value % 2 != 0)
    {
        ntlist *l = nodo->sons;
        while(l != NULL)
        {
            _hijosDePares(l->node,  result);
            l = l->next;
        }
    }
    else
    {
        insertLastListNTN(result, nodo->value);
    }

}


list *hijosDePares(ntn *root)
{
    list *result = createListNTN();
    _hijosDePares(root, result);

//Función que ordene la lista de menor a mayor (No la tengo implementada
pero esa sería la idea)

    OrderList(result);

    return result;
}
```

Ejercicio C

Teniendo en cuenta las estructuras hechas en la cursada:

Función principal: graphFrom2Graph.
La idea que se me ocurrió es generar para cada grafo su árbol de cobertura correspondiente, y luego con una función que se llama: _graphFrom2Graphs. Rellenar el grafo con los vértices y setear los arcos provenientes del árbol n-ario. Todo esto tendrá la siguiente implementación:

```
graph *graphFrom2Graphs(graph *g1, graph *g2)
{
    graph *result = createGraph();

    //Hacemos un árbol de cobertura en profundidad del grafo g1 y g2.
    ntn *root1 = graphDFTree(g1, 0);
    ntn *root2 = graphDFTree(g2, 0);

    _graphFrom2Graphs(result, root1);
    _graphFrom2Graphs(result, root2);

    return result;
}

void _graphFrom2Graphs(graph *g, ntn *node)
{
    if (node != NULL)
    {
        char *aux = (char*)malloc(10*sizeof(char));
        sprintf(aux, "%d", node->value);
        graphAddNewVertex(g, aux); //Seteamos los vertices
        ntlist *l = node->sons;
        while (l != NULL)
        {
            _graphFrom2Graphs(g, l->node);
            //Seteamos los arcos en dos direcciones con un 1 de costo.
            graphSetDobleArc(g, node->value, l->node->value, 1);
            l = l->next;
        }
    }
}
```

Nota: La única función diferente que se utilizo es “graphSetDobleArc” que básicamente te hace el link entre dos vértices en ambas direcciones, para no tener que hacer por ejemplo graphSetArc(g, i, j) y graphSetArc(g, j, i)