ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN
ESTRUCTURAS DE DATOS
SEGUNDA EVALUACIÓN – II TÉRMINO 2012-2013

Nombre: _____ ***Matrícula:*** _____ ***Paralelo:*** ___

TEMA 1 (15 ptos)

Un árbol binario puede ser *par*, *impar* o *neutral*. Si la diferencia entre el *número de elementos* pares e impares es positiva entonces es un árbol PAR, en caso de ser negativa es un árbol IMPAR y de ser CERO es un árbol neutral.

Sea:

```
typedef struct {
        Generic cont;
        AB *left;
        AB *right;
}AB;
```

Implemente la función **`int CalcularArbol(AB *ab);`** que recibe un *árbol binario* con valores *enteros* como contenido y retorna la diferencia entre el número de nodos con contenido par e impar.

TEMA 2 (25 ptos)

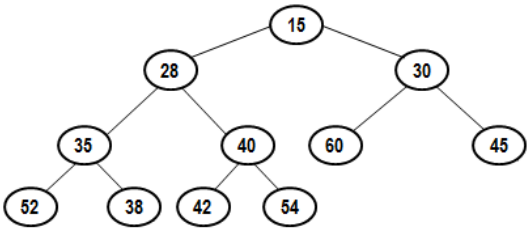
Considere el siguiente recorrido ***pre-orden*** de un *árbol binario de búsqueda*:

PRE-ORDEN: **1 0 10 2 7 6 3 4 5 9 8**

a) **(5 ptos)** Construya el árbol binario de búsqueda.
b) **(5 ptos)** Realice un recorrido pos-orden.
c) **(15 ptos)** En el orden con que fueron obtenidos los elementos del recorrido pos-orden, construya un árbol balanceado AVL, indicando cuantas rotaciones se hicieron para balancear el árbol. Grafique cada rotación.

TEMA 3 (20 ptos)

Represente con diagramas los pasos necesarios para extraer los 4 números menores del siguiente árbol usando *heapshort*.



TEMA 4 (30 ptos)

Se pretende dotar de agua potable por tubería a las comunidades de la amazonia ecuatoriana, para eso se ha instalado una red de tuberías de 5” y 10” de diámetro que conectan a cada comunidad entre si. Las de mayor diámetro son de poliuretano y se usan para distancias inferiores a los 1000mts entre comunidades, las otras son de fibra resistente y se utilizan para distancias superiores. Se desea instalar un reservorio de 10.000 galones en una de las comunidades, la que esté más céntrica de manera que el agua llegue con la mejor presión posible a todas las comunidades, se sabe también que por cada metro de tubería se pierde de 10 a 25 galones de agua dependiendo del diámetro de cada tubería, además cada comunidad consume una cantidad X de agua en función del número de familias.

a) **(10 ptos)** Defina la(s) TDA’s necesarias para representar el sistema de agua potable instalado en las comunidades.
b) **(20 ptos)** Escriba una función que determine cual será la ultima comunidad en recibir el liquido vital (agua), dada la comunidad donde se encuentra el reservorio.

TEMA 5 (10 ptos)

Conteste verdadero (V) o falso (F):

a. ( ) En un mapa (hashmap) los tiempos de búsquedas están en función del número de elementos
b. ( ) Si se recorre enorden a un ABB los elementos salen ordenados ascendentemente.
c. ( ) El árbol de expansión de un grafo se obtiene al realizar un recorrido en anchura.
d. ( ) Un árbol de búsqueda binaria siempre va a ser eficiente en sus búsquedas
e. ( ) En arbol de huffman en su raíz están los elementos de mayor frecuencia
f. ( ) Las colas de prioridad son implementadas con arboles de huffman
g. ( ) Un arbol de huffman admite huecos (nodos sin todos sus hijos) entre sus elementos
h. ( ) Un heap mantiene sus elementos ordenados: izq, raíz, derecha
i. ( ) En un heap de altura h>1, tiene todos sus nodos en el nivel h-1
j. ( ) En un heap basado en un arreglo, calculamos el nodo derecho desde una posición i como: Der(i) = (i+1)*2

Referencia

tree.h

```
NodeTree *nodeTreeNew(Generic key, Generic value);
int nodeTreeisLeaf(NodeTree *node);
Generic nodeTreeGetValue(NodeTree *node);
Generic nodeTreeGetKey(NodeTree *node);
NodeTree *nodeTreeGetLeft(NodeTree *node);
NodeTree *nodeTreeGetRight(NodeTree *node);
void nodeTreeSetValue (NodeTree *node , Generic value);
void nodeTreeSetKey(NodeTree *node , Generic key);
void nodeTreeSetLeft(NodeTree *node, NodeTree *newleft);
void nodeTreeSetRight(NodeTree *node, NodeTree *newright);
Tree treeNew();
int treeIsEmpty(Tree A);
void treeInsertNode(Tree *A, NodeTree *nuevo, cmpfn cmp);
NodeTree *treeRemoveNode(Tree *A, Generic key, cmpfn cmp);
NodeTree *treeRemoveRoot(Tree *A);
NodeTree *treeSearch(Tree A, Generic key, cmpfn fn );
void treePreOrder(Tree A, printfn fn);
void treePostOrder(Tree A, printfn fn);
void treeInOrder(Tree A, printfn fn);
Tree treeReadFromFile(char *filename, readfn read, getkeyfn getkey, cmpfn cmp);
```

heap.h

```
Heap * heapNew(int max, TSort type, cmpfn cmp);
int heapIsEmpty(Heap *H);
List *heapSort(Heap *H);
Generic heapDeQueue(Heap *H);
void heapEnQueue(Heap *H, Generic G);
Generic heapGetDatum(Heap *H, int idx);
int heapGetSize(Heap *H);
void heapPrint(Heap *H, printfn print);
void heapMake(Heap *H);
```

graph.h

```
GVertex *gVertexNew(Generic content);
void gVertexDelete(GVertex **V);
Generic gVertexGetContent(GVertex *v);
void gVertexSetContent(GVertex *v, Generic content);
List *gVertexGetAdjacents(GVertex *v);
int gVertexIsVisited(GVertex *v);
void gVertexVisit(GVertex *v);
void gVertexClearVisit(GVertex *v);
int gVertexCmpxContent(Gvertex *v1, Gvertex *v2);
GEdge *gEdgeNew(GVertex *vo, GVertex *vd, int weight, Generic extrainfo);
Generic gEdgeGetExtraInfo(GEdge *e);
void gEdgeSetExtraInfo(GEdge *e, Generic extraInfo);
int gEdgeGetWeight(GEdge *e);
void gEdgeSetWeight(GEdge *e, int weight);
GVertex *gEdgeGetSource(GEdge *e);
void gEdgeSetSource(GEdge *e, GVertex *ns);
GVertex *gEdgeGetDestination(GEdge *e);
void gEdgeSetDestination(GEdge *e, GVertex *nd);
int gEdgeCmpxDestination(GEdge *e, GVertex *vdestination);
Graph *graphNew();
int graphisEmpty(Graph *G);
void graphAddVertex(Graph *G, GVertex *v);
NodeList *graphRemoveVertex(Graph *G, GVertex *V, cmpfn fn);
void graphLinkVertices(Graph *G, GVertex *vsource, GVertex *vdestination, int weight, Generic extraInfo);
void graphUnlinkVertices(Graph *G, GVertex *vsource, GVertex *vdestination);
GEdge *graphGetLink(GVertex *vsource, GVertex *vdestination);
int graphCountVertices(Graph *G);
int graphCountEdges(Graph *G);
int gVertexCompare(GVertex *v, GVertex *ov);
GVertex *graphSearchVertex(Graph *G, Generic cont, cmpfn fn);
void graphInit(Graph *G);
void graphPrint(Graph *G, printfn prVertex, printfn prEdge);
```