ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

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

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

TEMA 1 (30 ptos)

Dos alumnos intentaron pasarse información en una prueba, el mensaje fue capturado y su contenido es el siguiente:

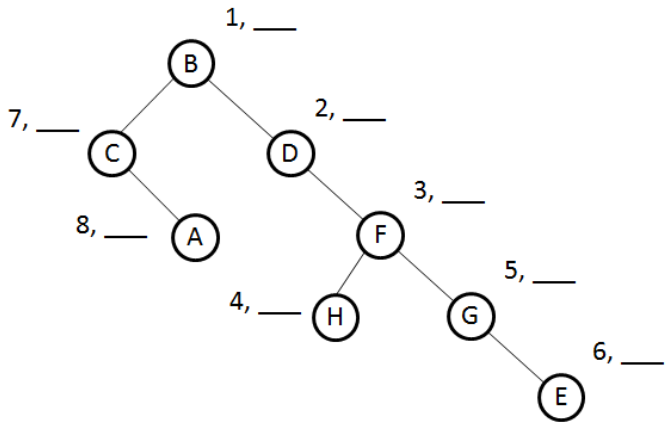
0	1	1	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	1	1	0	0	1	1	1	0	1	1	1	0	1	1	0

De las averiguaciones se sabe que utilizaron un algoritmo de codificación y compresión, además se tiene las letras y sus frecuencias de aparición en el mensaje: **D=3; U=3; Y=3; T=4; S=4**
Dependiendo del contenido del mensaje se esperan sanciones disciplinarias, usted ha sido asignado a las investigaciones y deberá:

a) **(15 ptos)** Identificar el algoritmo utilizado, indicar cuales fueron los códigos asignados a cada letra.
b) **(10 ptos)** Decodificar el mensaje y explicar cuál fue la estrategia para codificarlo.
c) **(5 ptos)** Calcular la tasa de compresión que se obtuvo con respecto del mensaje original, es decir, que porcentaje se redujo el mensaje tomando en cuenta el tamaño original del mismo.

TEMA 2 (20 ptos)

El siguiente es un árbol de expansión obtenido de un recorrido en profundidad a partir un grafo, se sabe que tiene arcos de retroceso entre los vértices A-B y E-D. Encuentre los puntos de articulación del grafo. *Justifique su respuesta.*



TEMA 3 (30 ptos)

Dada la siguiente expresión infija:

INFIJA: **((7 + 2 * 3 - 8) * 2 + 2) / 4 * 2 - 5 * (3 + 2 * 2 - 6)**

- **a) (10 ptos)** Escriba el algoritmo que permite cambiar la expresión de infija a posfija. Mediante diagramas realice el cambio de la expresión infija a postfija.
- **b) (10 ptos)** Escriba el algoritmo que construye el árbol de expresión. Mediante diagramas construya el árbol paso a paso.
- **c) (10 ptos)** Escriba el algoritmo que evalué el árbol de expresión. Mediante diagramas obtenga el resultado paso a paso.

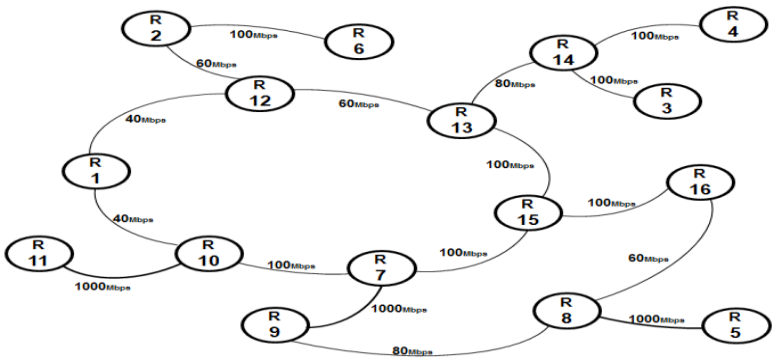
Nota: *Utilice pseudocódigo para los algoritmos.*

TEMA 4 (20 ptos)

Un mensaje de broadcast en una red de datos, se produce cuando desde un elemento de la red se envía un mensaje a todos los otros elementos de forma casi simultanea, no obstante, debido a las diferentes características que maneja una red de datos habrá algunos elementos reciban primero el mensaje y otros que lo reciban último.

a) **(15 ptos)** Escriba una **función en C**, que dado un elemento inicial retorne una *lista* con el orden en que fue recibido el mensaje de broadcast por cada uno de los elementos en la red. Considere únicamente el factor de proximidad con otros elementos.

b) **(5 ptos)** Usando la función del literal a), realice una *prueba de escritorio* y determine cual es el ultimo elemento en la red en recibir el mensaje, si el broadcast fue originado en el elemento R1.



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);
```