ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN
ESTRUCTURAS DE DATOS
SEGUNDA EVALUACIÓN - II TÉRMINO 2015

Nombre: _____ ***Matrícula:*** _____

TEMA 1 (15 PUNTOS)

Considere una tabla de dispersión de tamaño 7 con la siguiente función hash **h(k) = k%7**. Dibuje la tabla en memoria que resulte después de insertar los siguientes valores: 19, 26, 13, 48, 17 para cada uno de los siguientes escenarios:

a) Cuando las colisiones son manejadas con hashing abierto
b) Cuando las colisiones son manejadas con hashing cerrado usando rehash lineal
c) Cuando las colisiones son manejadas con hashing cerrado usando una segunda función **h'(k) = 5 - (k%5)**

TEMA 2 (20 puntos)

Escriba un método que se llame balance que determine si un árbol cumple la siguiente condición de balanceo: por cada subárbol, el número de nodos de su lado izquierdo y lado derecho difieren a lo mucho en un nodo. Asuma que el árbol está definido por una clase llamada NodoArbol que tiene las variables Dato, HijoIzquierdo e HijoDerecho.

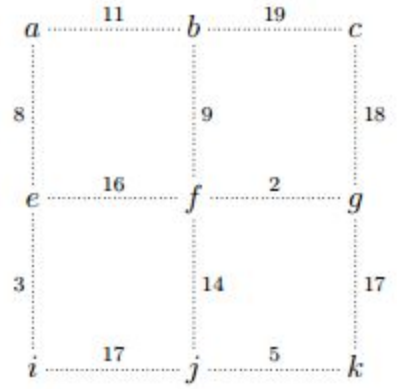
El método debe ser parte de la clase NodoArbol y tiene la siguiente forma:

```
boolean balance(NodoArbol nodo)
```

Devuelve verdadero si el subarbol izquierdo tiene a lo más un nodo más o menos que el subarbol izquierdo de ***nodo.*** Caso contrario devuelve falso.

TEMA 3 (10 puntos)

Identifique y dibuje un árbol de mínima expresión (árbol de expansión mínimo) del siguiente gráfico usando el algoritmo de Kruskal. Aparte escriba el orden en el cual el algoritmo de kruskal los agrega al árbol. Finalmente, indique el peso total del árbol que identificó.



TEMA 4 (25 puntos)

Escriba un método estático **vecinoEnComun** que reciba un grafo y un par de vértices (V1 y V2), y determine si tienen un vecino en común.

Dos vértices tiene un vecino en común si para un vértice V3, el grafo contiene un arco entre V1-V3, y un arco entre V2-V3. Asuma que los dos vértices están presente en el grafo y el grafo es no dirigido.

a) Resuelva utilizando una matriz de adyacencia
b) Resuelva utilizando una lista de adyacencia

TEMA 5 (30 puntos)

Usted es parte de un equipo de investigación dedicado al análisis de redes sociales. Durante este año están trabajando en el análisis de los mensajes (Tweets) que los usuarios escriben en Twitter, por ello a usted le entregan un conjunto de mensajes geolocalizados. De cada mensaje se tiene: texto (publicación) con los hashtags, la locación geográfica desde donde se envió el mensaje (latitud, longitud), el usuario que escribió el mensaje (autor) y el usuario que hace retweet del mensaje (vacío en caso de ser el mensaje original). Adicionalmente, se le indica que un usuario solamente tuiteó un solo mensaje.

Del conjunto de datos se quiere conocer:

a) Todos los hashtag usados por los usuarios.
b) Dado un hashtag, obtenga la red de mensajes que se forma.
c) Obtener cada una de las redes generadas por todos los hashtag

Su supervisor le provee clase Util que contiene el método

```
public static List<String> getHashtags(String text)
```

Recuerde que usted deberá crear los TDA necesarios (definir o heredar)

public enum Type { /*** * Grafo no dirigido.*/ UNDIRECTED, /*** * Grafo dirigido. */ DIRECTED}	
abstract class Graph	
Graph(Type type)	Constructor, Recibe el tipo de grafo
int getNumVertices()	Obtiene el número de vértices en el grafo
int getNumEdges()	Obtiene el número de arcos en el grafo
Object[] getVertices()	Agrega un nuevo vértice al grafo
void addVertex(Vertex v) void addEdge(Vertex source, Vertex destination, Object weight, Object info) void addEdge(Vertex source, Vertex destination) void addEdge(Vertex source, Vertex destination, int weight) void addEdge(Vertex source, Vertex destination, float weight) void addEdge(int v, int w, Object weight, Object info)	Agrega un nuevo arco entre 2 vértices. Si el grafo es no dirigido, genera el arco de destino a origen.
List<Vertex> getNeighbors(Vertex vertex) List<Vertex> getNeighbors(int v) List<Integer> getInNeighbors(Vertex v)	Obtiene los vecinos de un vértice
List<Vertex> depthFirstSearch(Vertex v, boolean visitAll)	Hace un recorrido en profundidad a partir del vértice v. Cuando visitAll es true recorre todas las componentes del grafo.
List<Vertex> breadthFirstSearch(Vertex v, boolean visitAll)	Hace un recorrido en anchura partir del vértice v. Cuando visitAll es true recorre todas las componentes del grafo.
void setUnVisited()	Establece todos los vértices del grafo como no visitados.
class Vertex<T>	
Vertex(Comparable<T> content, boolean visited) Vertex(Comparable<T> content)	Constructor
Comparable<T> getContent()	Obtiene el contenido del vértice
void setContent(Comparable<T> content)	Establece el contenido del vértice
boolean isVisited()	Obtiene si el vértice ha sido visitado o no
void setVisited(boolean visited)	Establece un vértice como visitado
class TreeNode<K,V>	
TreeNode(Comparable<K> key, V value)	Constructor
boolean isLeaf()	Determina si el nodo es una hoja
V getValue()	Devuelve el valor guardado en el nodo

Comparable<K> getKey()	Devuelve la clave guardada en el nodo
TreeNode getLeft()	Obtiene el hijo izquierdo
TreeNode getRight()	Obtiene el hijo derecho
void setKey(Comparable<K> key)	Establece la clave en el nodo.
void setValue (V value)	Establece el valor en el nodo
void setLeft(TreeNode leftNode)	Establece el hijo izquierdo
void setRight(TreeNode rightNode)	Establece el hijo derecho.