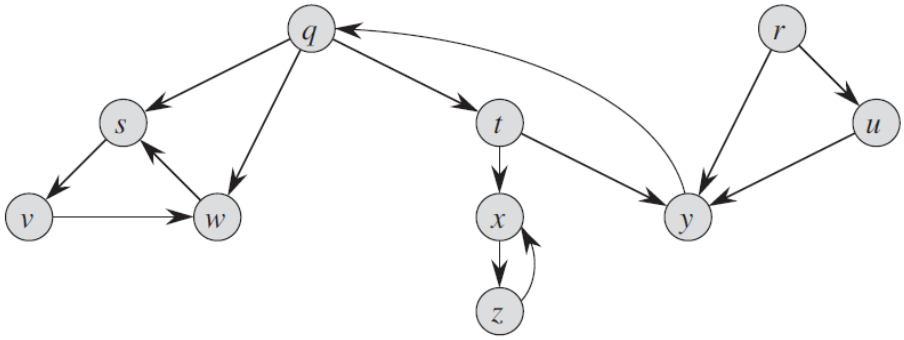
EJERCICIOS PARA LA CLASE

Ejercicio 1
Dada la representación utilizando lista de adyacencia de un grafo dirigido, que tiempo llevará calcular el grado-salida de cada vértice? Y el grado-entrada de cada vértice. El grado-entrada es la cantidad de aristas entrantes del vértice, y el grado-salida es la cantidad de aristas salientes del vértice

Ejercicio 2
Dibuje un grafo que represente todas las llamadas posibles que una subrutina puede hacer en un programa SL que tiene cinco subrutinas más el programa principal.
Nombre a sus subrutinas: *sub1, sub2, sub3, sub4, sub5* y *pp* al programa principal.

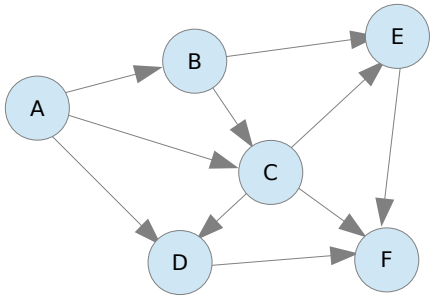
Ejercicio 3
Considere el siguiente grafo: Dibuje el árbol BFS y el árbol DFS considerando los algoritmos de CLRS mostrados en clase.



Ejercicio 4
Mostrar la representación de lista de adyacencia para un árbol binario completo de 7 vértices. Luego mostrar la representación como matriz de adyacencia. Asuma que los vértices están numerados del 1 al 7.

Ejercicio 5
Dada la siguiente representación de Grafo mediante lista de adyacencia donde cada vértice es un simple caracter. Implemente una función que imprima los vértices del grafo usando *búsqueda en profundidad*. Pruebe su algoritmo con el grafo mostrado a la derecha del código. Indique el costo asintótico de su implementación

```
/* Usamos todo public para simplificar la implementación que debe completar */
class Grafo {
    public  int  size ;
    public  Vertice [] TablaVertices ;

    public Grafo (int n ) {
       size = n;
       TablaVertices = new Vertice[size];
    }

    public static void main ( ) {
         Grafo g = new Grafo (10);
         g.insertArista ( 'A', 'B');
         g.insertArista ( 'A', 'C');
         g.insertArista ( 'A', 'D');
         ...
    }
}
```

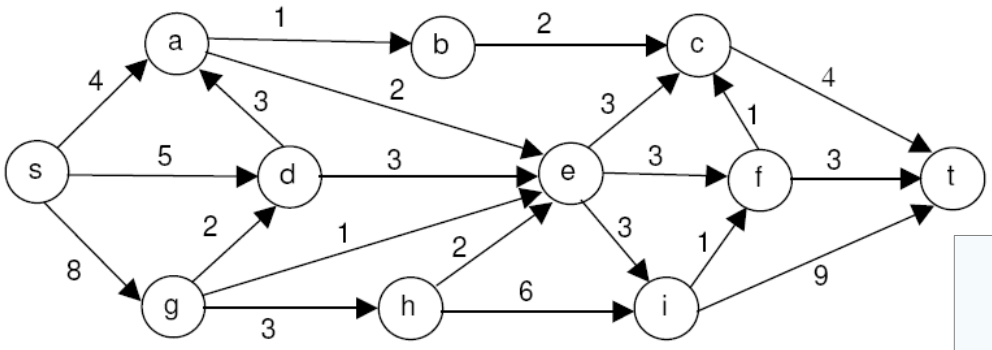



```
 class LinkArista {
      public int dest;  /* Destino */
      public LinkArista next = null; /* Siguiente adyacente*/

      public LinkArista ( int d, Link l ) {
           this.dest = d;
           this.next = l;
      }
 }
```

```
class Vertice {
     public char ident;        /* Identificador del nodo (A, B, C, etc.) */
     public int key;           /* Posición dentro de Vertices */
     public LinkArista ady ; /* Lista de aristas adyacentes */

     public Vertice ( int k, char d ) {
         this.ident = d;
         this.key   = k;
         ady        = null;
     }
}
```

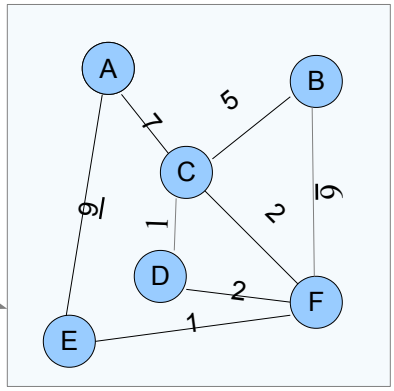
Ejercicio 6

Muestre la ordenación topológica para el grafo dirigido acíclico de abajo. ¿La ordenación es única? Si su respuesta es NO, justifíquela.



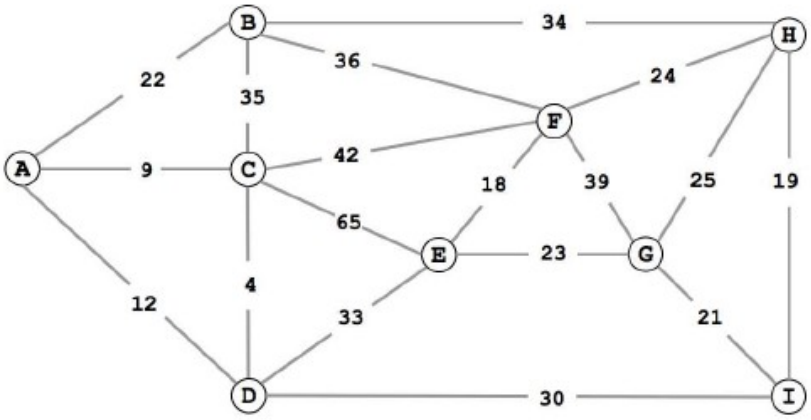
Ejercicio 7

Genere el MST (árbol de recubrimiento mínimo) del siguiente grafo utilizando el algoritmo de PRIM y luego de KRUSKAL. ¿Tiene el grafo más de un MST?



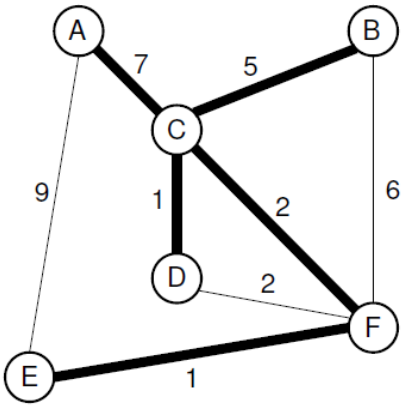
Ejercicio 8

Aplique el algoritmo de **Prim** sobre el grafo mostrado abajo , empezando por la arista A.

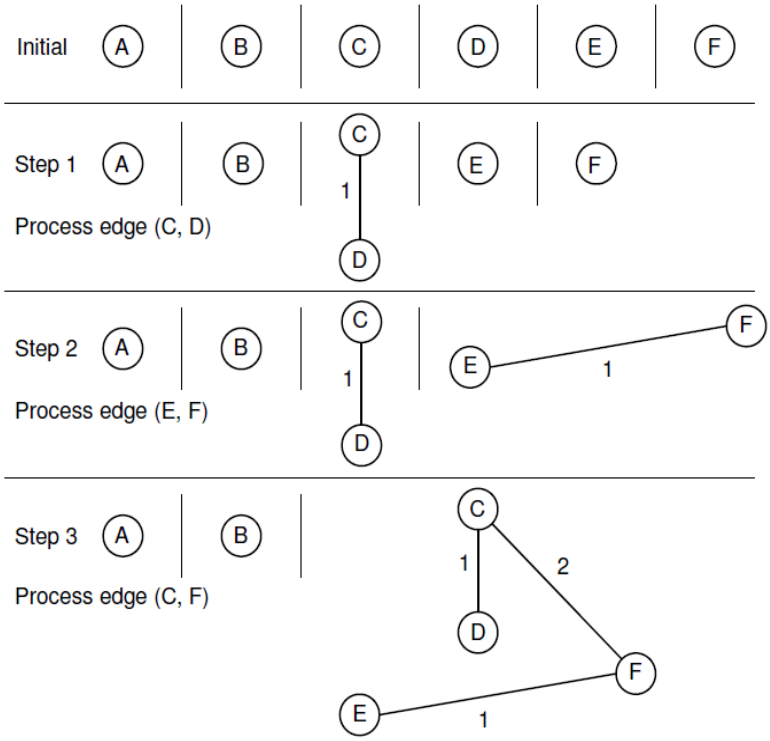


Luego aplique el algoritmo de **Kruskal** , siguiendo la siguiente descripción:
a) Partimos el conjunto de vértices en |V| clases equivalentes, cada una consistente de 1 vértice
b) Procesamos las aristas en orden de peso.
c) Una arista es agregada al MST y dos clases equivalentes son combinadas, si la arista conecta dos vértices en diferentes clases.
d) Este proceso es repetido hasta que se tenga una sola clase equivalente.

Ejemplo, dado el grafo de abajo:



El proceso de algoritmo de Kruskal aplicado al grafo de arriba (en sus primeros tres pasos)


Initial
A
B
C
D
E
F
Step 1
Process edge (C, D)
1
Step 2
Process edge (E, F)
1
Step 3
Process edge (C, F)
1
2
1

Ejercicio 9
El siguiente algoritmo, denominado ***Algoritmo de Bellman-Ford*** permite resolver el problema del camino mínimo desde un vértice considerando que las aristas pueden tener peso negativo.

Dado un grafo dirigido G=(V,E) con un vértice inicial *s* y una función de peso *w* : E → R, el algoritmo de *Bellman-Ford* retorna un valor lógico indicando si tiene o no un ciclo negativo que es alcanzable desde *s*. Si existe un ciclo, el algoritmo indica que no existe solución. Si no existe tal ciclo, el algoritmo produce los caminos mínimos y sus pesos.

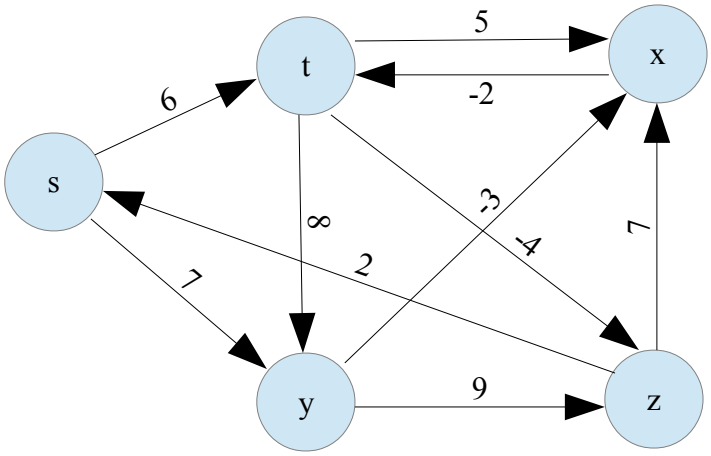
```
RELAX(u, v, w)
1  if d[v] > d[u] + w(u, v)
2     then d[v] ← d[u] + w(u, v)
3          π[v] ← u
```

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ V[G]
2       do d[v] ← ∞
3          π[v] ← NIL
4  d[s] ← 0
```

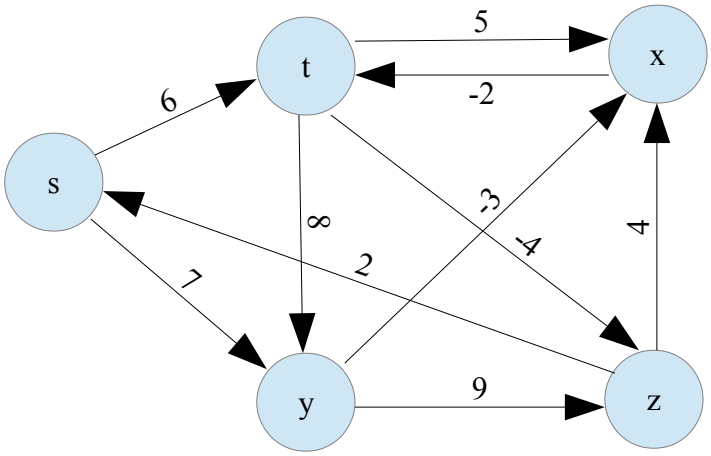
```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i ← 1 to |V[G]| − 1
3       do for each edge (u, v) ∈ E[G]
4            do RELAX(u, v, w)
5  for each edge (u, v) ∈ E[G]
6       do if d[v] > d[u] + w(u, v)
7            then return FALSE
8  return TRUE
```

Aplique el algoritmo para los siguientes grafos muestre los valores de d[] y π[]

a) Utilizando ***s*** como vértice inicial



b) Utilizando ***x*** como vértice inicial



Algoritmos y Estructura de datos III – Sección TR – 1er Semestre - Año 2018 - Tarea #8
Prof. Cristian Cappo **-** A presentar hasta el Viernes 4/Mayo/2018 hasta las 23:55 hs **TP=55P**

Utilice para su código la implementación base del Prof. Clifford Shaffer. Baje el código del sitio indicado en Educa o el proporcionado en el sitio del Prof. Shaffer http://people.cs.vt.edu/~shaffer/Book/JAVA/progs/

Recuerde que debe disponibilizar datos de prueba para que la corrección sea fácil y justa. Implemente métodos de impresión para verificar los resultados solicitados

Ejercicio 1 (10p) (Para el caso de implementaciones, incluya código y datos de prueba para verificar la solución**)**
a) Dado un grafo dirigido, implemente el algoritmo que permita encontrar los componentes fuertemente conectados [CLRS 2nd Edition – 22.5].

b) Resuelva el ejercicio 23.2-7 del [CLRS 2nd Edition].
Suppose that a graph G has a minimum spanning tree already computed. How quickly can the minimum spanning tree be updated if a new vertex and incident edges are added to G?

c) Resuelva el ejercicio 23.2-8 del [CLRS 2nd Edition].

> Professor Toole proposes a new divide-and-conquer algorithm for computing minimum spanning trees, which goes as follows. Given a graph $G = (V, E)$, partition the set V of vertices into two sets V_1 and V_2 such that $|V_1|$ and $|V_2|$ differ by at most 1. Let E_1 be the set of edges that are incident only on vertices in V_1, and let E_2 be the set of edges that are incident only on vertices in V_2. Recursively solve a minimum-spanning-tree problem on each of the two subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Finally, select the minimum-weight edge in E that crosses the cut (V_1, V_2), and use this edge to unite the resulting two minimum spanning trees into a single spanning tree.
>
> Either argue that the algorithm correctly computes a minimum spanning tree of G, or provide an example for which the algorithm fails.

d) Implemente el algoritmo de Kruskal indicado en el libro de CLRS, utilizando “disjoint-sets”. Debe implementar ese TDA en una clase separada con los métodos indicados en el algoritmo.

Ejercicio 2 (10p)
Dado un grafo no dirigido G con el conjunto de vértices $\{0,1,..,V-1\}$ y con E aristas. Denotamos con (i,j) la conexión entre el vértice i y el vértice j y con $d(i)$ el grado del vértice i.

Definimos las siguientes operaciones:
- `borrarArista(i,j)`: borrar de G la arista *(i,j)*.
- `borrarAristasIncidentes(i)`: borrar de G todas las *d(i)* aristas incidentes en el referido vértice.

Incluya todos los archivos necesarios para las pruebas de funcionamiento. Incluya métodos de impresión adecuados para la evaluación de la solución.

Por cada solución indicar el costo asintótico en términos de número de vértices y número de aristas.
Debe realizar la implementación para lista de adyacencia.

Ejercicio 3 (10p)
Se ha comentado en clase dos formas de representación de grafos: *Matriz de adyacencia* y *Lista de adyacencia*. Se mencionó que en algunos casos una forma es preferible que la otra y viceversa.

Dado los siguientes casos, ¿qué representación sería la más conveniente?. Justifique su respuesta.

a) Un grafo tiene diez mil vértices y veinte mil aristas y es fundamental usar la menor cantidad de espacio posible.

b) Un grafo tiene diez mil vértices y veinte millones de aristas.

c) Se necesita responder en tiempo constante la siguiente pregunta: ¿Esta la arista *(i,j)* en el grafo?

d) Se necesita responder en tiempo constante la siguiente consulta: ¿Existe un vértice conectado al vértice *i* o es *i* un vértice aislado?

Puede asumir que los vértices se identifican por enteros en el rango *1 .. V*

Ejercicio 4 (10p)

Considere un grafo dirigido acíclico (GDA) G = (V, E). Diseñe e implemente un algoritmo eficiente en Java que determine si el número de caminos en G, desde un vértice *s* a otro *t*, es par o impar. Analice el tiempo de ejecución de su algoritmo en términos de |V| y |E|. Si el grafo de entrada no es GDA entonces lanzar una excepción.

Utilice las implementaciones de los algoritmos conocidos del libro del Prof. Shaffer (BFS, DFS, orden topológico, etc). Implemente la solución utilizando lista de adyacencias.

Incluya código y datos de prueba para una mejor verificación.

Ejercicio 5 (5p)

Escriba un en Java que considerando un grafo no dirigido *G=(V,E)* con pesos, implementado como listas de adyacencias, encuentre el árbol de expansión MÁXIMO. Esto es, el árbol de expansión con el mayor costo posible.

Indique exactamente las estructuras auxiliares de datos que utiliza y el costo asintótico en términos de |V| y |E|.

La generación del árbol puede ser simplemente imprimir el árbol o retornar en una estructura de datos.

Ejercicio 6 (5p)

La transpuesta de un grafo dirigido G=(V,E) es el grafo $G^T = (V,E^T)$, donde $E^T = \{ (v,u) \in VxV : (u,v) \in E \}$. Por tanto. G^T es G pero con las aristas revertidas.

Diseñar e implementar un algoritmo eficiente para calcular G^T desde G. Hacerlo para ambas representaciones de grafos. Analice y describa el tiempo de ejecución de cada uno de los algoritmos.

Utilizar las implementaciones de algoritmos dadas.

Ejercicio 7 (5p)

Suponga que en vez de tener listas enlazadas, en una representación con listas de adyacencia, se tenga una tabla de dispersión por cada vértice. Suponiendo que todos los vértices tienen la misma probabilidad de aparición, ¿Cuál sería el tiempo estimado de calcular si una arista se encuentra en el grafo? ¿Qué desventajas tiene este esquema?