Apellidos: _____ *Nombre:* _____

Ejercicio 1 *[5 puntos]*

Se quiere diseñar una clase llamada ***VacunasCovid*** para manejar información de los centros de administración de vacunas. Se necesita guardar el nombre del centro y una serie de pares de valores con los nombres de las vacunas y las dosis existentes de cada una. Las vacunas deben guardarse por orden alfabético. Se pide:

a. Proporcionar el área de datos más adecuada para esta clase. (0.5 ptos)

b. Escribir el constructor que recibe el nombre del centro y el de copia. (0.75 ptos)

c. Escribir un método que retorne el número de vacunas distintas disponibles. Indicar el coste (0.5 ptos)

d. Escribir un método que retorne el número total de dosis disponibles. Indicar el coste (0.75 ptos)

e. Implementar un método que añada a la base de datos un número de dosis de una cierta vacuna (la vacuna puede estar o no en la base de datos) (1.25 ptos)

f. Implementar un método que reste una dosis a una vacuna pasada como parámetro. Retorna un valor booleano indicando si se pudo restar o no la dosis (1.25 ptos)

Ejercicio 2 *[5 puntos]*

Dado el grafo de la figura, se pide:

a. Dibujar el bosque de extensión en profundidad (siempre se escogen los vértices, cuando haya que elegir, por orden alfabético de su etiqueta) NO SE PUNTUA SI NO SE PONEN LOS PASOS SEGUIDOS! (1.5 ptos)

b. Dibujar todos los tipos de arcos que existen e indicar el tipo de cada uno. (0.5 ptos)

c. Escribir una función estática que retorne el grado de salida máximo de todos los vértices de un grafo (1.5 ptos)

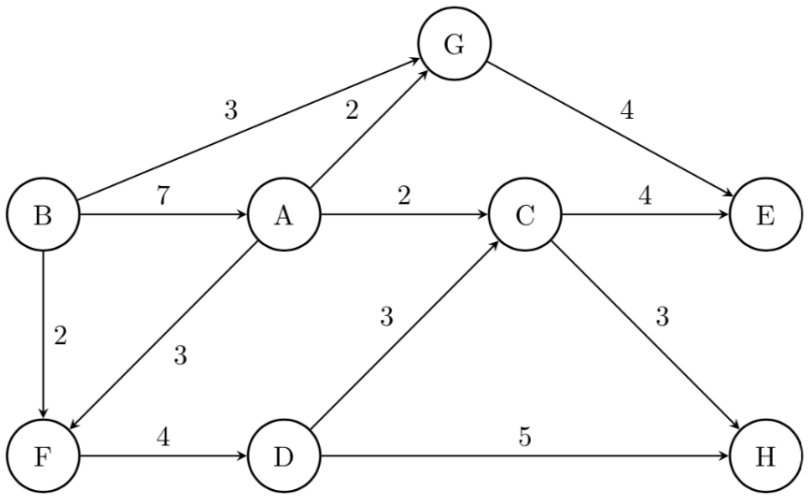
```
public static <Vertex> int gradoSalidaMax (Graph<Vertex> g);
```

Para el grafo de la figura devolvería 3 (grado de salida del vértice A y el B)

d. Escribir una función estática que retorne la suma de los pesos de los ejes del vértice que tiene grado de salida máximo (1.5 ptos)

```
public static <Vertex> double sumGradoSalidaMax (Graph<Vertex> g);
```

Para el grafo de la figura devolvería 12 (suma de los pesos de los ejes que salen de B: 7+2+3)



ANEXO

```
public interface Set<E> extends Collection<E>
{
   // Basic operations
   int size();
   boolean isEmpty();
   boolean contains(Object element);
   boolean add(E element);
   boolean remove(Object element);
   Iterator<E> iterator();
   // Bulk operations
   boolean containsAll(Collection<?> c);
   boolean addAll(Collection<? extends E> c);
   boolean removeAll(Collection<?> c);
   boolean retainAll(Collection<?> c);
   void clear
   // Array Operations
   Object[] toArray();
   <T> T[] toArray(T[] a);
}
```

```
public interface Map<K,V> {
    V put(K key, V value);
    V get(Object key);
    V remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    int size();
    boolean isEmpty();

    void putAll(Map<? extends K,
                    ? extends V> m);
    void clear();

    public Set<K> keySet();
    public Collection<V> values();
    public Set<Map.Entry<K,V>> entrySet();

    public interface Entry {
        K getKey();
        V getValue();
        V setValue(V value);
    }
}
```

```
public class Graph<Vertex> {
      private TreeMap<Vertex, TreeMap<Vertex, Double>> adjList;
      private int numVertex;        número de vértices
      private int numEdges;         número de arcos

      public Graph();
      public Graph(Graph<Vertex> g);
      public boolean hasVertex(Vertex v);
      public boolean addVertex(Vertex v);
      public boolean removeVertex(Vertex v);
      public int degreeIn(Vertex v);
      public int degreeOut(Vertex v);
      public boolean hasEdge(Vertex from, Vertex to);
      public boolean addEdge(Vertex from, Vertex to, Double weight);
      public boolean removeEdge(Vertex from, Vertex to);
      public Double weightEdge(Vertex from, Vertex to);
      public Collection<Vertex> adjacentsTo(Vertex v);
      public Collection<Vertex> getAllVertex();
      public int getNumVertex();
      public int getNumEdge(); }
```

Soluciones

Ejercicio 1

a)
```
private String nombre;
private TreeMap<String, Integer > vacunas;  //Tree pq se pide que estén en orden
```

b)
```
public VacunasCovid(String name){
     nombre=name;
     vacunas=new TreeMap<String, Integer> ();
}
public VacunasCovid (VacunasCovid v){
     this(v.nombre);
     data.putAll(v.vacunas);
}
```

c)
```
public int numVacunasDistinas(){   //O(1)
    return vacunas.size();
}
```

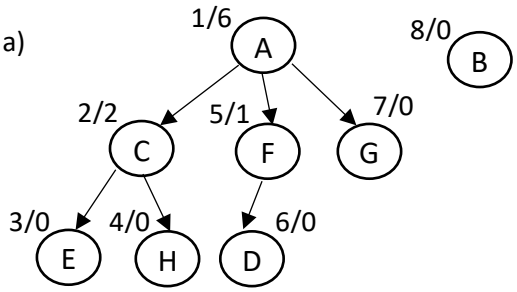
d)
```
public int numDosis(){
    int num=0;
    Collection<Integer> dosis=vacunas.values();   //O(N)
    for(Integer d:dosis)
         num+=d;
     return num;    }
```

e)
```
public int insertarDosis(String vacuna, int d){
    if (vacunas.containsKey(vacuna))
          vacunas.put(vacuna,vacunas.get(vacuna)+d;
    else
          vacunas.put(vacuna, d);
```

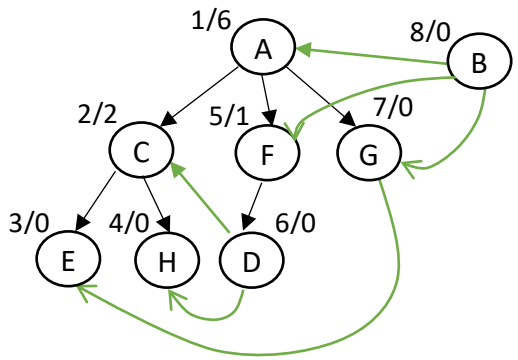
f)
```
public boolean restaDosis(String vacuna){
    if (vacunas.containsKey(vacuna)){
          int d=vacunas.get(vacuna);
          if (d==1)
                vacunas.remove(vacuna);
          else
                vacunas.put(vacuna,d-1);
          return true;
    }
    else
          return false;
```

Ejercicio 2

a)



b)



(Faltan las explicaciones basadas en las listas de adyacentes) azul: retroceso | naranja: avance | verde: cruzado

c)

```
public static <Vertex> int gradoSalidaMax (Graph<Vertex> g) {
	int gradoMax=-1;
	Collection<Vertex> vertices=g.getAllVertex();
	for (Vertex w: vertices){
		Collection<Vertex> adjs=g.adjacentsTo(w);
		int gradoSalida=adjs.size();
		if (gradoSalida>gradoMax)
			gradoMax=gradoSalida;
	}
	return gradoMax;
}
```

d)

```
public static <Vertex> double sumGradoSalidaMax (Graph<Vertex> g) {
	int gradoMax=-1;
	double sumaMax=0;
	Collection<Vertex> vertices=g.getAllVertex();
	for (Vertex w: vertices){
		Collection<Vertex> adjs=g.adjacentsTo(w);
		int gradoSalida=adjs.size();
		if (gradoSalida>=gradoMax) {
			//sumar los pesos de los arcos
			double suma=0;
			for (Vertex v: adjs){
				double peso=g.weightEdge(w,v);
				suma+=peso;
			}
			if ((gradoSalida==gradoMax && suma>sumaMax) || gradoSalida>gradoMax)
				sumaMax=suma;
			gradoMax=gradoSalida;
		}
	}
	return sumaMax;
}
```

NOTA: Como en el enunciado que os di puse como ejemplo de este método que salía 7 (debido a la suma de pesos del vértice A, que es el primero que se obtiene de grado 3) también se dará por buena la implementación correspondiente.