Nombre y Apellido: ….…… N° Legajo: ……

Segundo Parcial de Estructuras de Datos y Algoritmos 72.34

Segundo Cuatrimestre de 2014 – 14/11/2014

Ejercicio 1	*Ejercicio 2*	*Ejercicio 3*	*Nota*

Condición Mínima de Aprobación: Tener por lo menos dos ejercicios con B-

Consideraciones a tener en cuenta. MUY IMPORTANTE

- **El ejercicio que no respete estrictamente el enunciado será anulado.**
- **Se puede entregar el examen escrito en lápiz.**
- **Se tendrán en cuenta la eficiencia y el estilo de programación.**
- **Los teléfonos celulares deben estar apagados.**
- **Entregar los ejercicios en hojas separadas**

Ejercicio 1

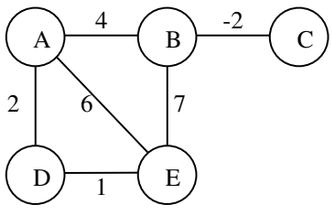
Se cuenta con la siguiente implementación parcial de un grafo no dirigido con peso en las aristas:

```
public class Graph<V> {

	private HashMap<V, Node> nodes = new HashMap<V, Node>();
	private List<Node> nodeList = new ArrayList<Node>();

	public void addVertex(V vertex) {
		if (!nodes.containsKey(vertex)) {
			Node node = new Node(vertex);
			nodes.put(vertex, node);
			nodeList.add(node);
		}
	}

	public void addArc(V v, V w, Double d) {
		Node origin = nodes.get(v);
		Node dest = nodes.get(w);
		if (origin != null && dest != null && !origin.equals(dest)) {
			for (Arc arc : origin.adj) {
				if (arc.neighbor.info.equals(w)) {
					return;
				}
			}
			origin.adj.add(new Arc(dest, d));
			dest.adj.add(new Arc(origin, d));
		}
	}

	private class Node {
		V info;
		boolean visited = false;
		int tag = 0;
		List<Arc> adj = new ArrayList<Arc>();

		public Node(V info) {
			this.info = info;
		}
		public int hashCode() {
			return info.hashCode();
		}
		public boolean equals(Object obj) {
			if (obj == null || (obj.getClass() != getClass())) {
				return false;
			}
			return info.equals(((Node)obj).info);
		}
	}
	private class Arc {
		Node neighbor;
		Double weight;

		public Arc(Node neighbor, Double weight) {
			this.neighbor = neighbor;
			this.weight = weight;
		}
	}
}
```

Implementar un algoritmo que dado cierto peso encuentre un camino en el grafo (sin repetir nodos) donde la suma de las aristas que lo conforman sea exactamente el peso buscado. Si existe, debe retornar los nodos que lo conforman (si existe más de un camino basta con retornar uno solo). Si no existe, debe retornar null.

El peso de los ejes puede ser negativo.

Ejemplo: para el siguiente grafo, si se ejecuta el algoritmo con peso 7, podría retornar la lista {B, A, D, E}. Si se ejecuta con peso 5 podría retornar {E, B, C}. Con peso 12 no puede retornar {C,B,E,D,A,B} porque repite B, debe retornar null



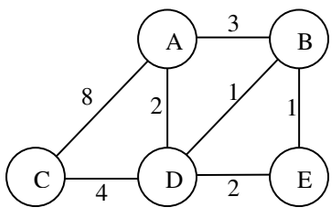
Ejercicio 2

Se tiene un grafo con pesos enteros **positivos** en las aristas. Implementar un algoritmo que dados dos nodos retorne la cantidad de caminos de menor peso entre ellos.

Ejemplo: en el siguiente grafo, si se ejecuta el algoritmo para los nodos A y B la respuesta es 2, ya que el peso del camino mínimo entre ambos es 3, y existen dos caminos con este peso: {A, B}y {A, D, B}.

Si se ejecuta para los nodos A y C la respuesta es 1 (el peso del camino mínimo es 6, y está dado por los nodos {A, D, C}.

Si se ejecuta el algoritmo para los nodos A y E la repuesta es 3. El peso del camino mínimo es 4, y existen tres caminos con este peso: {A, B, E}, {A, D, E} y {A, D, B, E}.



Ejercicio 3

Se trata de resolver el rompecabezas ABC donde se tiene un tablero de 5x5 donde cada celda puede estar vacía o haber una ‘A’, una ‘B’ o una ‘C’.
En cada fila y cada columna debe haber exactamente una ocurrencia de cada letra (no puede haber dos iguales, no puede faltar una)
Para algunas filas y columnas se puede tener indicada una letra al principio y/o al final. Estas letras son las primeras que se encuentran si se comienza el recorrido desde ese punto.

En el ejemplo siguiente, la primer letra que se encuentra recorriendo la primer fila desde la columna cero debe ser una A, pero recorriéndola desde “la izquierda” puede ser cualquiera.
Si se recorre la segunda columna “desde abajo” la letra que se encuentre debe ser una A, pero si se la recorre desde arriba puede ser cualquiera.

Ejemplo:

Tablero inicial

A						
						C
B						
B						
		A	**C**		**C**	

Solución única

A	**A**	**C**			**B**	
		B	**A**	**C**		**C**
	C			**B**	**A**	
B			**B**	**A**	**C**	
B	**B**	**A**	**C**			
		A	**C**		**C**	

Tiene muchas soluciones

A					
B					
C					
B					
B					

No tiene solución

		B				
A						**C**
						C
B						
B						
		A	**C**		**C**	

Se pide: implementar una clase que soporte el juego y el método isValid que dado el juego sin comenzar retorne:

- 0 si no tiene solución
- 1 si tiene solución única
- 2 si tiene más de una solución