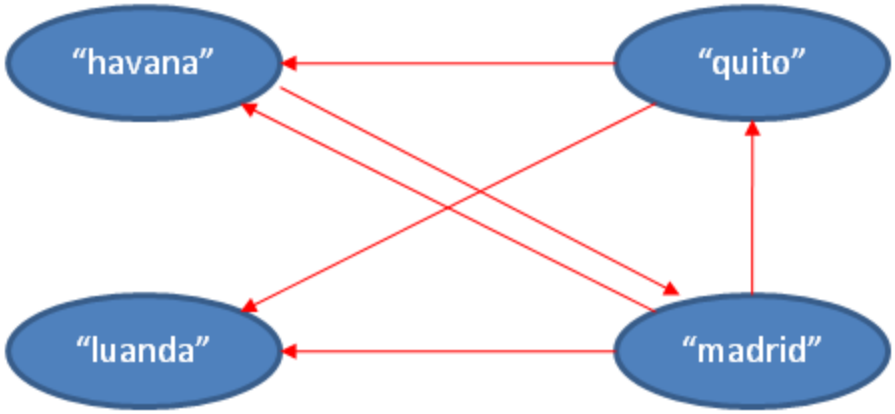
## Grafo dirigido

Un grafo dirigido está constituido por un conjunto de objetos (nodos) y aristas entre ellos. En la figura, los nodos del grafo son las cadenas "habana", "madrid", "quito" y "luanda" y las aristas son

- "madrid" → "luanda"
- "madrid" → "quito"
- "madrid" → "habana"
- "habana" → "madrid"
- "quito" → "habana"
- "quito" → "luanda"




Observe que no es lo mismo la arista A→B que la B→A.

Programe una biblioteca de clases EjercicioGrafo.dll donde defina la clase genérica `EjercicioGrafo.GrafoDirigido<T>` que implemente la interfaz `Examen.IGrafoDirigido<T>` suministrada en la biblioteca IGrafoDirigido.dll. Su clase `EjercicioGrafo.GrafoDirigido` debe tener un constructor sin parámetros.

A continuación se describen los miembros de la interfaz y la lógica esperada para cada uno de ellos.

| **Miembro** | **Descripción** |
|---|---|
| `void AgregarVertice(T vertice)` | Agregar un vértice al grafo. Si el grafo ya contenía a dicho vértice, lanzar `ArgumentException`. |

| **Miembro** | **Descripción** |
|---|---|
| `void EliminarVertice(T vertice)` | Eliminar un vértice del grafo. Implica la eliminación de todas las aristas que llegan o salen del vértice en cuestión. Si el objeto suministrado no es un vértice del grafo, lanzar `ArgumentException`. |
| `bool ContieneVertice(T vertice)` | Indica si determinado objeto constituye o no un vértice del grafo. |
| `void AgregarArista(T v1, T v2)` | Agrega al grafo una arista entre los vértices v1 y v2. Si alguno de los objetos suministrados no son vértices del grafo o si ya existía una arista v1→v2, lanzar `ArgumentException.` |
| `void EliminarArista(T v1, T v2)` | Elimina del grafo la arista entre los vértices v1 y v2. Si alguno de los objetos suministrados no son vértices del grafo o si la supuesta arista no existe, lanzar `ArgumentException.` |
| `bool ContieneArista(T v1, T v2)` | Indica si el grafo contiene la arista v1→v2. |
| `void Vaciar()` | Eliminar todos los vértices y aristas. |
| `int CantVertices { get; }` | Cantidad de vértices. |
| `int CantAristas{get;}` | Cantidad de aristas. |
| `IEnumerable<T> Vertices { get; }` | Enumeración de los vértices. Se garantiza que no se le tratará de hacer `Reset` al enumerador devuelto ni se modificará el grafo mientras haya un enumerador en uso. |
| `IEnumerable<Arista<T>> Aristas { get; }` | Enumeración de las aristas. Se garantiza que no se le tratará de hacer `Reset` al enumerador devuelto ni se modificará el grafo mientras haya un enumerador en uso. La clase `Arista<T>` está definida en el propio ensamblado IGrafo.dll |
| `int GradoDeSalida(T vertice)` | Cantidad de aristas que salen del vértice dado. Si el objeto suministrado no es un vértice del grafo, lanzar `ArgumentException` |

| Miembro | Descripción |
|---|---|
| `int GradoDeEntrada(T vertice)` | Cantidad de aristas que llegan al vértice dado. Si el objeto suministrado no es un vértice del grafo, lanzar `ArgumentException` |
| `IGrafoDirigido<T> Simetrico()` | Debe devolver un grafo con los mismos vértices y las aristas simétricas a las del grafo original, es decir, por cada arista x→y que exista en el grafo original, en el simétrico tiene que aparecer la arista y→x (y viceversa). El grafo original no se debe modificar. |
| `IGrafoDirigido<T> SubGrafo(IEnumerable<T> vertices)` | Devuelve un nuevo grafo con los vértices pasados como parámetro y todas las aristas del grafo original cuyos vértices inicial y final aparecen en esa secuencia. El grafo original no se debe modificar. Si alguno de los objetos de la enumeración no es un vértice del grafo, lanzar `ArgumentException.` |

A continuación le mostramos el código de la clase `Examen.Arista<T>` definida en IGrafoDirigido.dll (es solo para su conocimiento, no necesita ni debe incluir este código en su proyecto).

```
public class Arista<T>
{
    public Arista(T v1, T v2)
    {
        this.v1 = v1;
        this.v2 = v2;
    }
    private T v1, v2;
    public T V1 { get { return v1; } }
    public T V2 { get { return v2; } }

    public override bool Equals(object obj)
    {
        Arista<T> otra = (Arista<T>) obj;
        return v1.Equals(otra.v1) && v2.Equals(otra.v2);
    }
}
```

## *Ejemplos de uso de algunas de las funcionalidades de su clase.*

```
IGrafoDirigido<int> g = new Grafo<int>();
g.AgregarVertice(1);
g.AgregarVertice(2);
g.AgregarVertice(3);
g.AgregarArista(1, 2);
g.AgregarArista(2, 3);
```

```
Console.WriteLine(g.ContieneVertice(3));    // debe imprimir True
Console.WriteLine(g.ContieneVertice(9));    // debe imprimir False
Console.WriteLine(g.ContieneArista(2, 3));  // debe imprimir True
Console.WriteLine(g.ContieneArista(2, 1));  // debe imprimir False
Console.WriteLine(g.GradoDeSalida(1));      // debe imprimir 1
Console.WriteLine(g.GradoDeSalida(3));      // debe imprimir 0

IGrafoDirigido<int> sim = g.Simetrico();
Console.WriteLine(sim.ContieneVertice(3));    // debe imprimir True
Console.WriteLine(sim.ContieneVertice(9));    // debe imprimir False
Console.WriteLine(sim.ContieneArista(1, 2));    // debe imprimir False
Console.WriteLine(sim.ContieneArista(2, 1));    // debe imprimir True
```