Algoritmos para representar conexiones.

Muchas motivaciones…

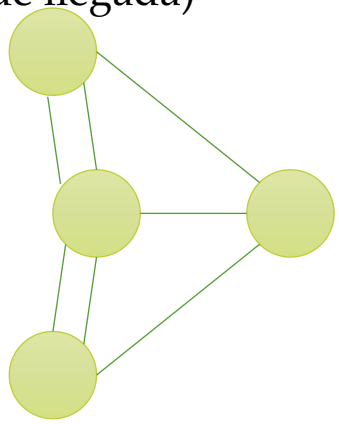
- Mapas

En el siglo 18, en Prusia (Königsberg) había zonas separadas por el río Pregolya unidas por 7 puentes. Se puede pasear por la ciudad pasando **una y una sola vez** por cada puente? Cómo?

Hay camino Euleriano posible?

Idea: modelar la situación con un grafo. Las 4 zonas son vértices y los puentes son los ejes. Los ejes no son dirigidos porque puedo atravesar el puente en cualquier dirección.

La teoría de grafos tiene muchos teoremas. Ej: para que haya un camino euleriano la cantidad de nodos con grado impar deben ser 0 o 2. (si fueran 2 serían el nodo de salida y el de llegada)



Tiene solución la visita a los puentes de Prusia?

Graph

- **Caso de Uso: “Flujo/Transporte”.**

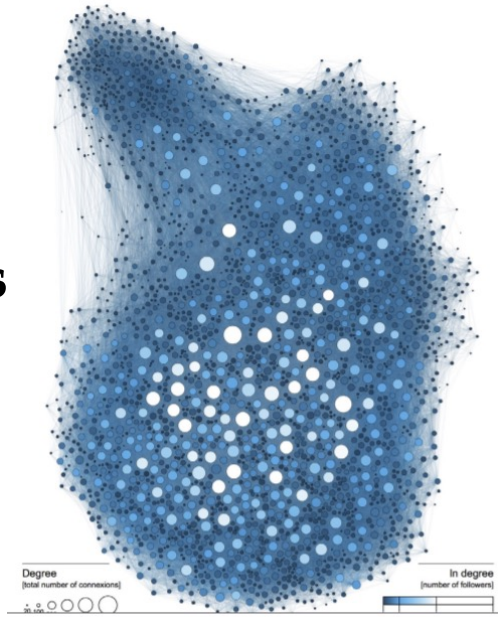
Para representar rutas, conexiones y tráfico. Ej: rutas áreas, formas alternativas para ir de un lugar a otro, etc.

Ej: Actualmente OpenstreetMap sigue usando grafos dirigidos para representar calles, avenidas, puentes, POIs, etc

- **Casos de Uso: “Redes sociales, y el análisis de la comunidad digital”**

Ej: 2500 usuarios de twitter

Y su interconexión



Cuando se precisa representar los vínculos entre objetos. Ej: Twitter, Facebook, DBLP (autores y publicaciones científicas), Netflix, etc, etc, etc.

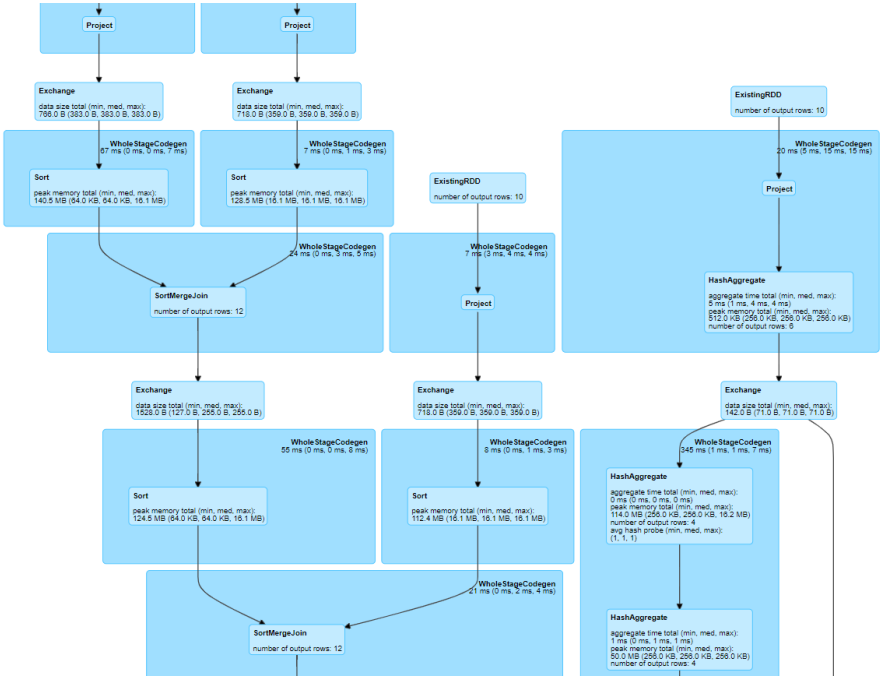
Dada la complejidad de las interrelaciones entre participantes, existe mucha información que puede extraerse al analizar este tipo de redes: el más influyente, las comunidades, recomendaciones, entre otros.

- **Casos de Uso:**
  “Instaladores/Compiladores/Optimizadores”

Precisan saber el orden conveniente en que hay que instalar/configurar/ejecutar paquetes.

Ej: manejador de proyectos de software como maven. Con tantas dependencias, ¿En qué orden debe instalarse un paquete y todas sus dependencias? ¿Quién depende de quién?

- Ej: un optimizador de SQL distribuido para ser ejecutado en un cluster de computadoras organiza las tareas

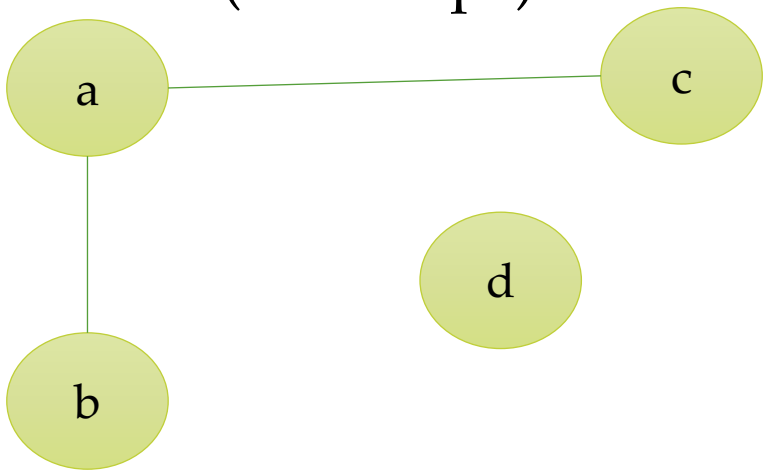


Ej: Al fabricar un elemento (ej: un auto). ¿Por donde se empieza?

Ej: Al armar un cronograma para cierta actividad, como ser ¿cuál es el plan para recibirse de Ing. en Informática en ITBA?

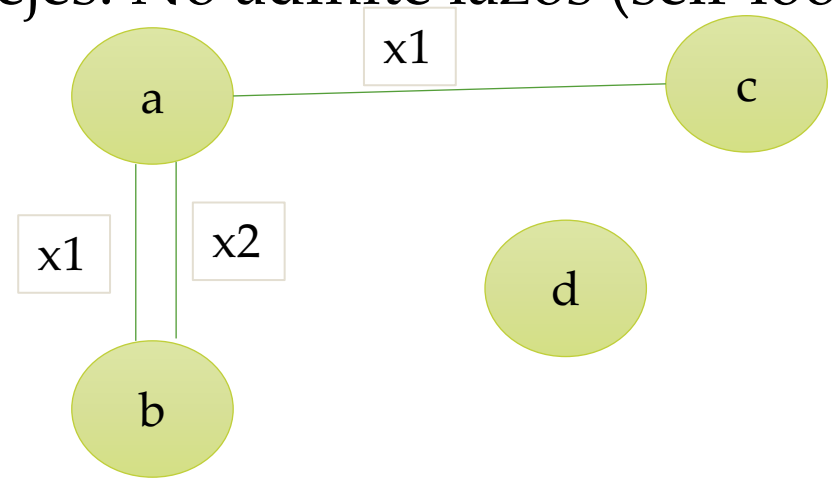
Grafos y sus tipos

- **Con ejes no dirigidos**
  - Simple: entre cada par de nodos hay a lo sumo un eje. No admite lazos (self-loops)



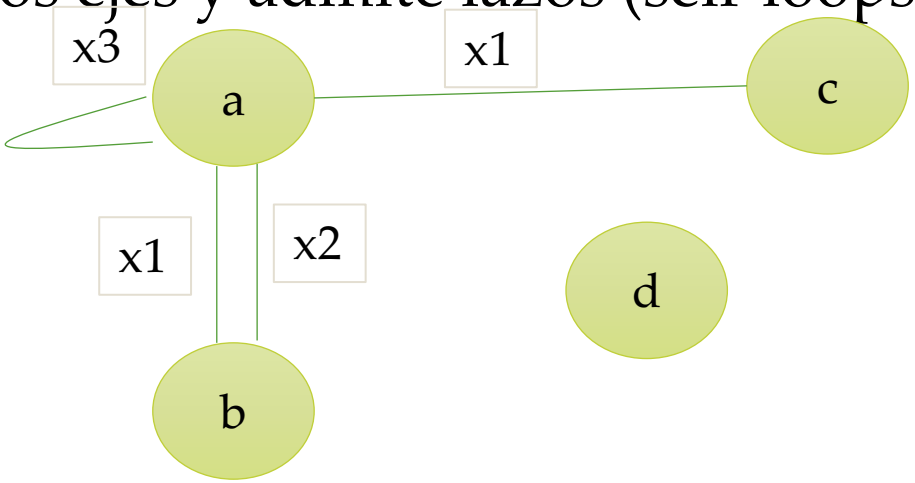
Tipos de Grafos

- **Con ejes no dirigidos**
  - Multigrafo: entre cada par de nodos puede haber varios ejes. No admite lazos (self-loops)



Tipos de Grafos

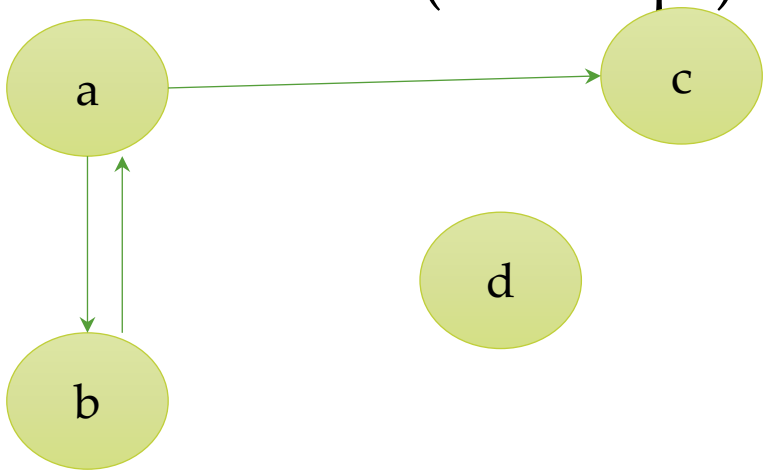
- **Con ejes no dirigidos**
  - Pseudografo: entre cada par de nodos puede haber varios ejes y admite lazos (self-loops)



No hay clasificación para simple con lazo (self-loop)

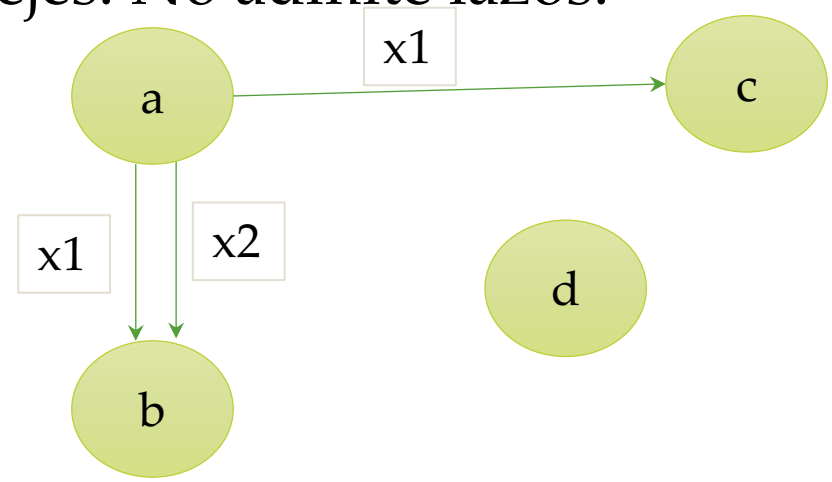
Tipos de Grafos

- **Con ejes dirigidos (digrafos)**
  - Simple digrafo: entre cada par de nodos hay a lo sumo un eje. No admite lazos (self-loops).



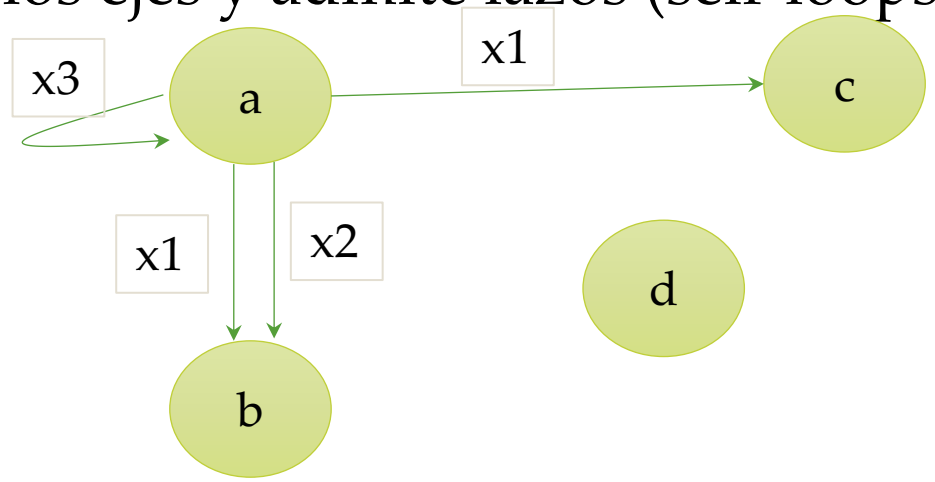
Tipos de Grafos

- **Con ejes dirigidos (digrafos)**
  - Multi digrafo: entre cada par de nodos puede haber varios ejes. No admite lazos.



Tipos de Grafos

- **Con ejes dirigidos (digrafos)**
  - Pseudo digrafo: entre cada par de nodos puede haber varios ejes y admite lazos (self-loops).



No hay clasificación para simple digrafo con lazo (self-loop)

Tipos de Grafos

Depende lo que quiera modelar, el tipo de grafo que me conviene.

Ej. Para rutas áreas, ¿hay algún vuelo que salga de una ciudad y llegue a la misma ciudad?

Rta: No. Entonces, no elegiría ninguno de los casos que acepten lazos.

Ej: Para correlatividades entre materias. ¿Puede una materia ser correlativa a ella misma? ¿Serviría no poner quien va antes que quien?

Rta: No. Entonces, no elegiría ninguno de los casos que acepten lazos. Tampoco elegiría uno no dirigido.

Tipos de Grafos

Además de todos los casos que expusimos tenemos las variantes donde los **ejes aceptan pesos**.

Ej: en el caso de rutas áreas esto puede ser muy útil porque podría colocar en dichos pesos la duración del vuelo, etc.

Conclusión

Por eso hay tantos tipos de grafos. Porque me permiten modelar situaciones diversas.

El siguiente cuadro sumariza los **8 tipos de grafos**:

Dirigido?	Multiplicidad?	Lazos?	Nombre
✗	✗	✗	Simple
✗	✗	✓	*Simple con lazos*
✗	✓	✗	Multi Grafo
✗	✓	✓	Pseudo Grafo
✓	✗	✗	(Simple) Digrafo
✓	✗	✓	*Digrafo con lazos*
✓	✓	✗	Multi Digrafo
✓	✓	✓	Pseudo digrafo

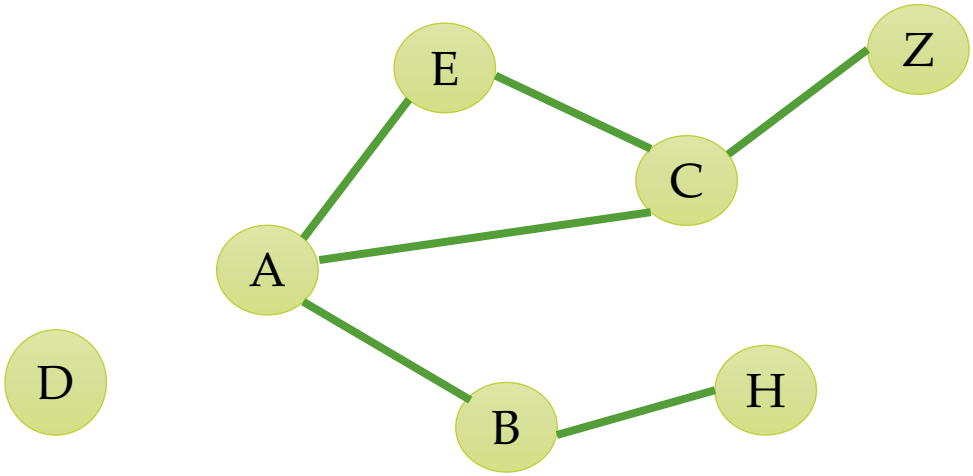
Si además de estas 8 combinaciones les permitimos manejar peso en los ejes => **tenemos 16 TIPOS !!!!**

A veces, algunas definiciones/algoritmos dependen del tipo.

Ej: indegree y outdegree solo aplica a grafos dirigidos.
Ej: calcular el camino mínimo solo aplica a grafos con peso en los ejes, etc.

Caso de Uso

```
g.dump();

System.out.println(String.format("#vertices: %d", g.getVertices().size() ));

System.out.println(String.format("#edges: %d", g.getEdges().size() ));

// degree de cada nodo
for(Character aV: g.getVertices()) {
   System.out.println(
   String.format("vertex %s has degree %d", aV, g.degree(aV)));
 }
```




```
Vertexes:
(A) (B) (C) (D) (E) (H) (Z)
Edges:
(A) -- (B)
(A) -- (C)
(A) -- (E)
(B) -- (A)
(B) -- (H)
(C) -- (A)
(C) -- (E)
(C) -- (z)
(E) -- (A)
(E) -- (C)
(H) -- (B)
(Z) -- (C)

#vertices: 7
#edges: 12
vertex A has degree 3
vertex B has degree 2
vertex C has degree 3
vertex D has degree 0
vertex E has degree 2
vertex H has degree 1
vertex z has degree 1
```

Para ejemplificar las posibles representaciones vamos a tomar como ejemplo: Grafo Simple

Formalmente un grafo simple

Sea G=(V, E)

Representamos a ambos conjuntos:

1) Vértices V

2) Y hay cuatro propuestas típicas para representar a los ejes E

Respecto a la implementación de los ejes, hay 4 implementaciones:

2.1) Matriz de adyacencia. Ideal grafos densos

2.2) Lista de adyacencia. Ideal grafos esparcidos (sparse)

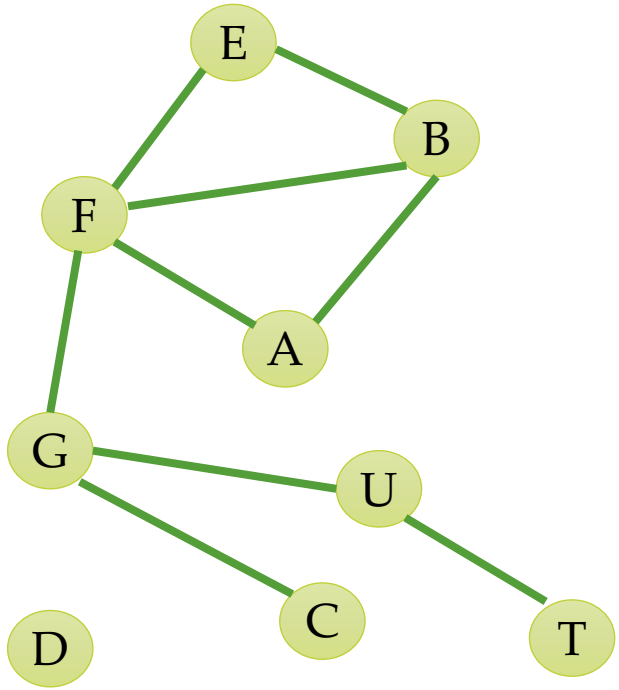
2.3) Matriz de incidencia. Ídem.

2.4) Lista de incidencia. Ídem.

Empecemos con la representación.

1) Para los vértices:

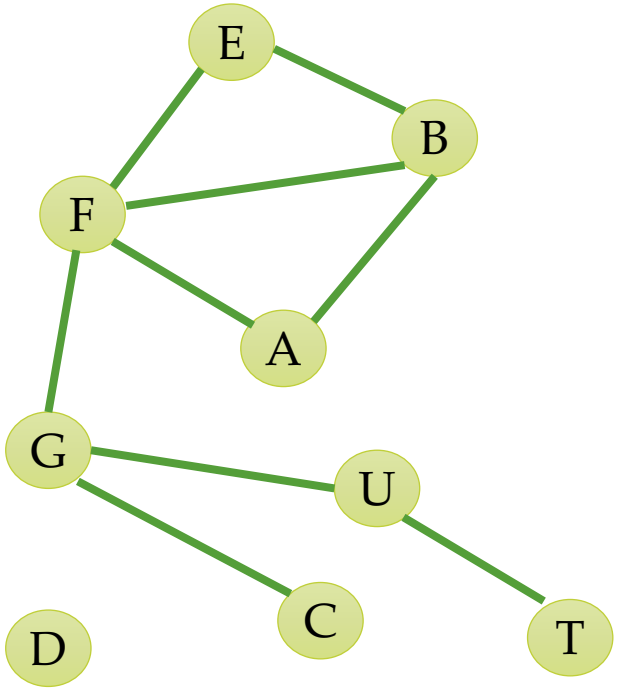
```
g.addEdge('E', 'B', new EmptyEdgeProp());
g.addEdge('A', 'B', new EmptyEdgeProp());
g.addEdge('F', 'B', new EmptyEdgeProp());
g.addVertex('D');
g.addVertex('G');
g.addEdge('E', 'F', new EmptyEdgeProp());
g.addEdge('F', 'A', new EmptyEdgeProp());
g.addEdge('F', 'G', new EmptyEdgeProp());
g.addEdge('U', 'G', new EmptyEdgeProp());
g.addEdge('T', 'U', new EmptyEdgeProp());
g.addEdge('C', 'G', new EmptyEdgeProp());
```



E	B	A	F	D	G	U	T	C

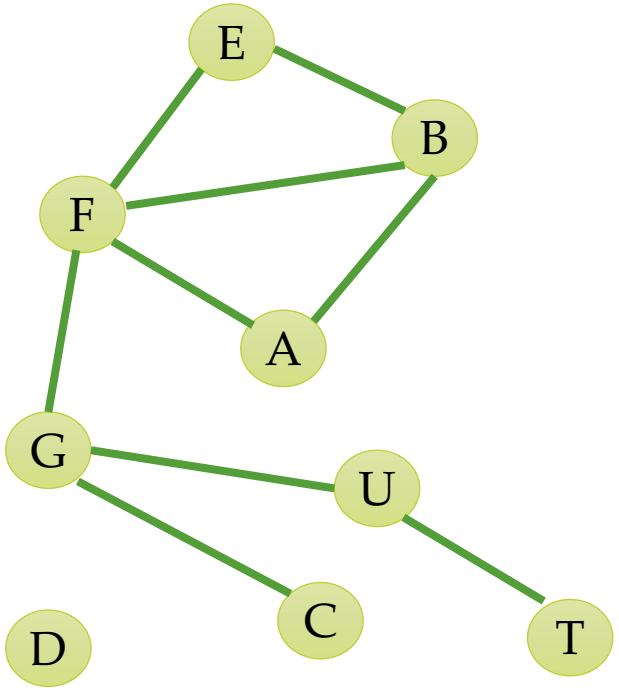
2.1) Para los ejes: “matriz de adyacencia”

		0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F
		E	B	A	F	D	G	U	T	C



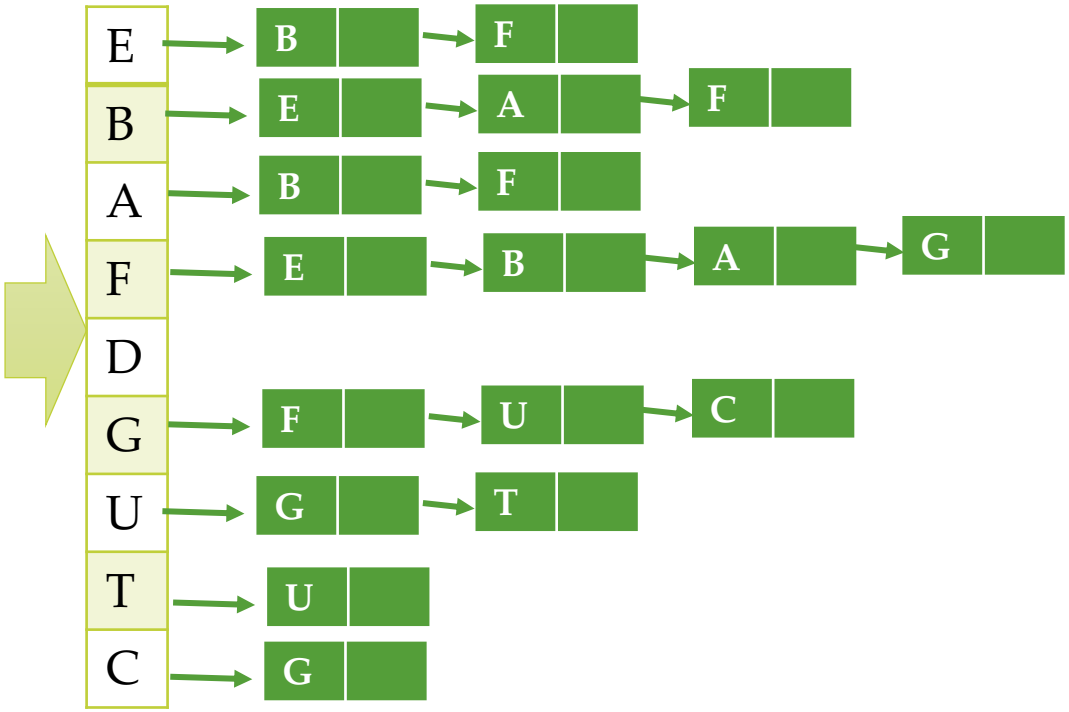
2.1) Para los ejes: “matriz de adyacencia”

	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F
		E	B	A	F	D	G	U	T	C

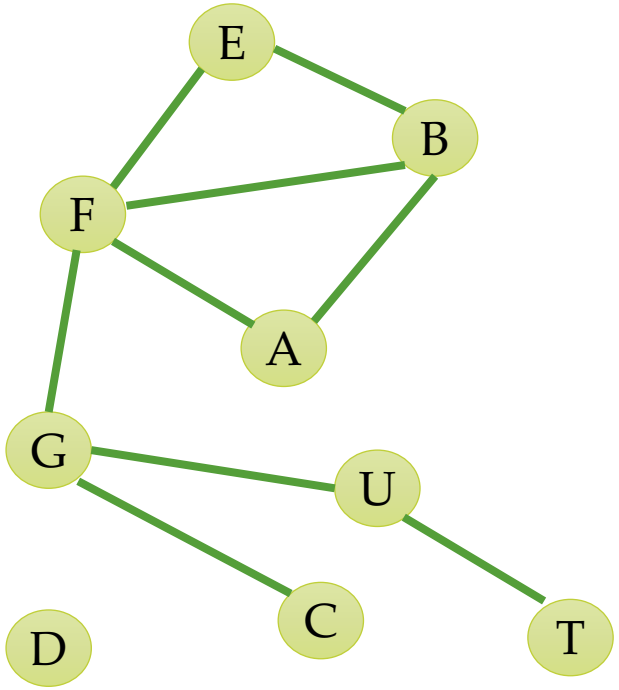


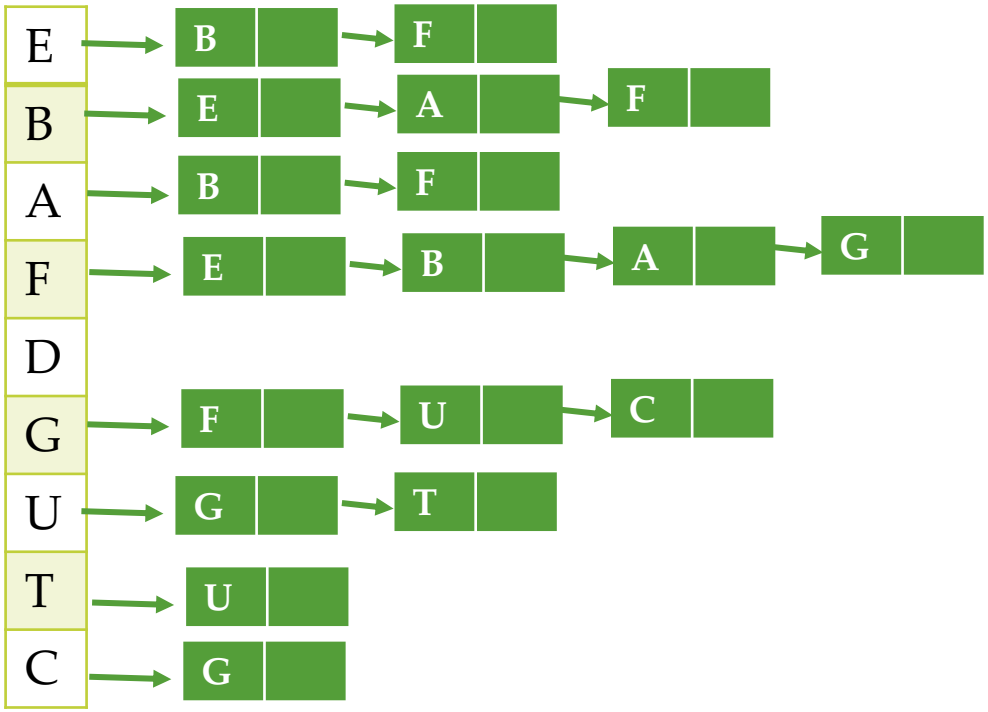
2.2) Para los ejes: "lista de adyacencia" si es muy esparcida

	0	1	2	3	4	5	6	7	8
E	0		T		T				
B	1	T		T	T				
A	2		T		T				
F	3	T	T	T			T		
D	4								
G	5				T			T		T
U	6						T		T	
T	7							T		
C	8						T			
		E	B	A	F	D	G	U	T	C



2.2) Para los ejes: “lista de adyacencia” si es muy esparcida

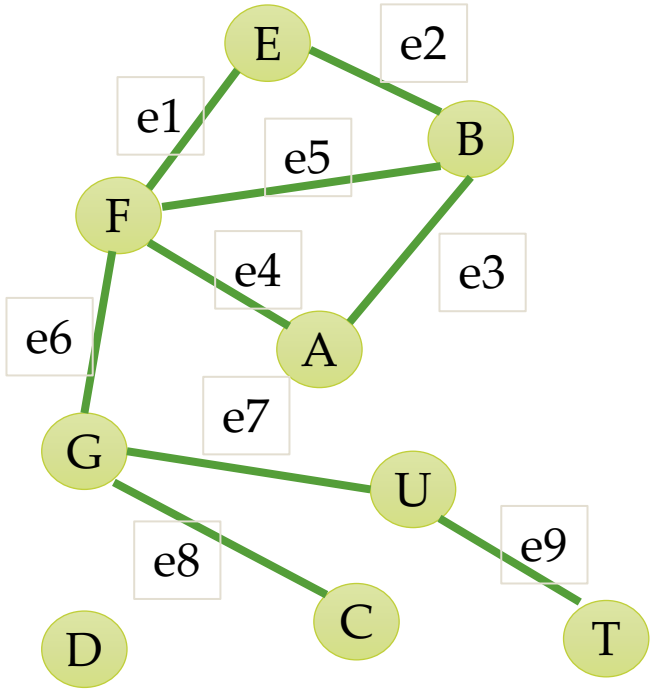




2.3) Para los ejes: "matriz de incidencia". Se colocan vértices y ejes en filas y columnas

		0	1	2	3	4	5	6	7	8
E	0	T	T	F	F	F	F	F	F	F
B	1	F	T	T	F	T	F	F	F	F
A	2	F	F	T	T	F	F	F	F	F
F	3	T	F	F	T	T	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	F	F	T	T	T	F
U	6	F	F	F	F	F	F	T	F	T
T	7	F	F	F	F	F	F	F	F	T
C	8	F	F	F	F	F	F	F	T	F

e1	e2	e3	e4	e5	e6	e7	e8	e9



2.4) Para los ejes: “lista de incidencia”. Idem. Se representa una lista asociada que compacta

Queremos que el usuario genere varios tipos de grafos posibles * 2 (por el peso en los ejes) * 4 (por las implementaciones posibles), pero no vamos a esperar que el usuario conozca el nombre de un montón de clases, para los casos.

Buena idea: escribir un Factory.

Esta idea es la que implementa la clase JGraphT, una biblioteca de código abierto sobre grafos muy importante.

La clase GraphFractory es abstracta. Create es método static. No me de una instancia de GraphFactory. Me da una instancia de la clase que corresponda a través de un servicio

Se parametrizan las clases que representan las propiedades de los vértices y las propiedades de los ejes

Modo de uso:

```
GraphService<Character,EmptyEdgeProp> g =
    GraphFactory.create(Multiplicity.SIMPLE,
                        EdgeMode.UNDIRECTED,
                        SelfLoop.NO,
                        Weight.NO,
                        Storage.SPARSE);
```

Características del tipo de grafo y del tipo de almacenamiento

La clase GraphFractory es abstracta. Create es método static. No me de una instancia de GraphFactory. Me da una instancia de la clase que corresponda a través de un servicio

Modo de uso:

```
GraphService<Character,EmptyEdgeProp> g =
    GraphFactory.create(Multiplicity.SIMPLE,
                        EdgeMode.UNDIRECTED,
                        SelfLoop.NO,
                        Weight.NO,
                        Storage.SPARSE);
```

Se parametrizan las clases que representan las propiedades de los vértices y las propiedades de los ejes

```
GraphService<Flight,WeightedEdgeProp> g =
    GraphFactory.create(Multiplicity.MULTI,
                        EdgeMode.DIRECTED,
                        SelfLoop.NO,
                        Weight.YES,
                        Storage.SPARSE);
```

Características del tipo de grafo y del tipo de almacenamiento

```
abstract public class GraphFactory<V, E> {

    public static <V, E> GraphService<V, E> create(Multiplicity edgeMultiplicity, EdgeMode theEdgeMode,
            SelfLoop acceptSelfLoops, Weight hasWeight, Storage theStorage) {

        if (theStorage== Storage.SPARSE)  // manejando 8 tipos con 2 clases concretas
            if (edgeMultiplicity== Multiplicity.SIMPLE)
                return new SimpleOrDefault<V,E>(theEdgeMode==EdgeMode.DIRECTED,
                                                acceptSelfLoops==SelfLoop.YES,
                                                hasWeight==Weight.YES );
            else
                return new Multi<V,E>(theEdgeMode==EdgeMode.DIRECTED,
                                      acceptSelfLoops==SelfLoop.YES,
                                      hasWeight==Weight.YES );

        // todavia no lo hemos implementado en forma Densa Matriz
        //  return new AdjacencymatrixGraph<V,E>(isSimple, isDirected, isWeighted );


        throw new RuntimeException("Not yet implemented");
    }

    private GraphFactory() {
    }
}
```

Instanciamos una clase u otra. Ambas implementan el GraphService (sparse).

Si es SimpleOrDefault sabemos que tiene máximo un eje entre 2 nodos.

Un Factory expone el orden de los parámetros en forma fija:

```
public static <V, E> GraphService<V, E> create(
Multiplicity edgeMultiplicity,
EdgeMode theEdgeMode,
SelfLoop acceptSelfLoops,
Weight hasWeight, Storage theStorage)
```

Otra buena idea es ofrecer una clase Builder que permita que en cualquier orden se pasen parámetros en forma aislada (si no se proporcionan asumen algún default) y cuando lo decida invoque build() que finalmente invoca al GraphFactory. Es un caso de postergación.

El usuario también podría crear grafos de diferentes formas:

```
GraphService <Character,EmptyEdgeProp> g =
new GraphBuilder<Character,EmptyEdgeProp> ().
withMultiplicity(Multiplicity.SIMPLE).
withDirected(EdgeMode.UNDIRECTED).
withAcceptSelfLoop(SelfLoop.NO).
withAcceptWeight(Weight.YES).
withStorage(Storage.SPARSE).
build();
```

```
GraphService <Character,EmptyEdgeProp> g =
new GraphBuilder<Character,EmptyEdgeProp> ().
withStorage(Storage.SPARSE).
withAcceptSelfLoop(SelfLoop.NO).
withAcceptWeight(Weight.YES).
withMultiplicity(Multiplicity.SIMPLE).
withDirected(EdgeMode.UNDIRECTED).
build();
```

```
// equivalente a la de orden PREDETERMINADO
GraphService<Character,EmptyEdgeProp> g =
GraphFactory.create(Multiplicity.SIMPLE,
    EdgeMode.UNDIRECTED,
    SelfLoop.NO,
    Weight.YES,
    Storage.SPARSE);
```

Es muy fácil implementar esta técnica.

```
public class GraphBuilder<V,E> {
    private Multiplicity multiplicity= Multiplicity.SIMPLE;
    private EdgeMode edgeMode= EdgeMode.DIRECTED;
    private SelfLoop acceptSelfLoops= SelfLoop.NO;
    private Weight hasWeight= Weight.NO;
    private Storage implementation= Storage.SPARSE;

    public GraphBuilder<V,E> withMultiplicity(Multiplicity param) {
        this.multiplicity= param;
        return this;
    }

    public GraphBuilder<V,E> withDirected(EdgeMode param) {
        this.edgeMode= param;
        return this;
    }

    public GraphBuilder<V,E> withAcceptSelfLoop(SelfLoop param) {
        this.acceptSelfLoops= param;
        return this;
    }

    public GraphService<V,E> build() {
      return GraphFactory.create(multiplicity, edgeMode, acceptSelfLoops, hasWeight, implementation);
    }
}
```

GraphBuilder es una clase instanciable y tiene vbles de instancia para cada estado posible, con algún default.

Hay un método de instancia por cada propiedad. Setea la propiedad y devuelve this para poder encadenar invocaciones.

El método build() no devuelve this. Es lo último a invocar. Reusa al GraphFactory de parámetros ordenados

GraphFactory (sea que lo invoca directamente el usuario o sea que se invoca a partir del GraphBuilder) genera una instancia de **SimpleOrDefault y Multi.**

Ambas clases tienen **mucho en común**.

Sin embargo, difieren en algo:

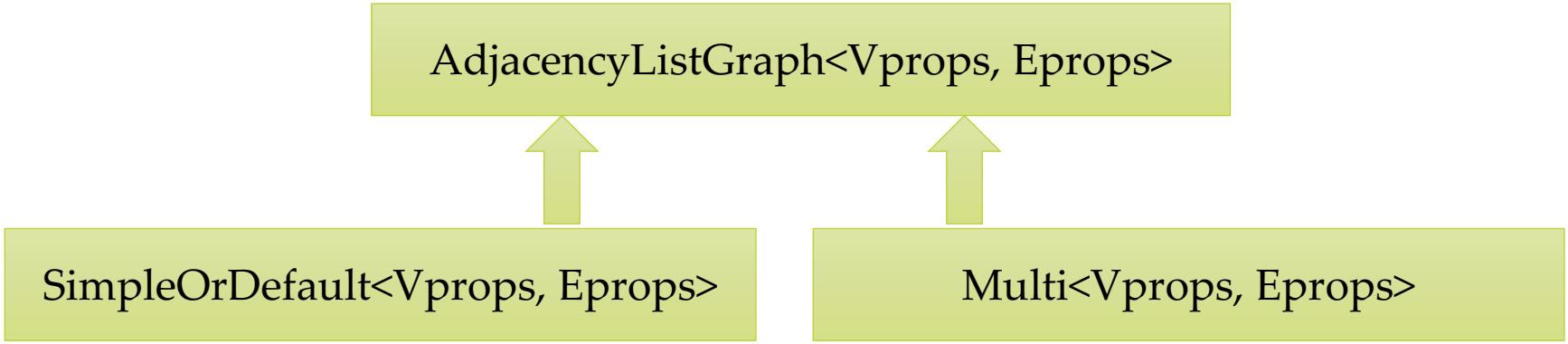
- **addEdge**: en el caso de SimpleOrDefault, si se vuelve a crear otro eje entre el mismo par de vértices, se ignora. En Multi no se ignora, se crea.

- **removeEdge**: en el caso de SimpleOrDefault, si se indica un par de vértices, con sus propiedades y se lo encuentra, se borra el único eje encontrado. En el caso de Multi se borran todas las apariciones de ese eje con mismas propiedades entre esos vértices.

Tip: si no se especifican properties se borra en el caso de SimpleOrDefault el único eje que pudiera existir entre dichos vértices. Si es Multi lanza exception (para evitar ambigüedad)

Es responsabilidad del usuario definir equals/hash en la clase que represente las propiedades de los vértices y las propiedades de los ejes.

Como ambas clases tienen mucho en común, podemos hacerlas especializar de la clase abstracta AdjacencyListGraph



Sea SimpleOrDefault o Multi

Como cada vértice tiene una lista de adyacencia asociada, podemos armar un Map de vértice a su lista de adyacencia.

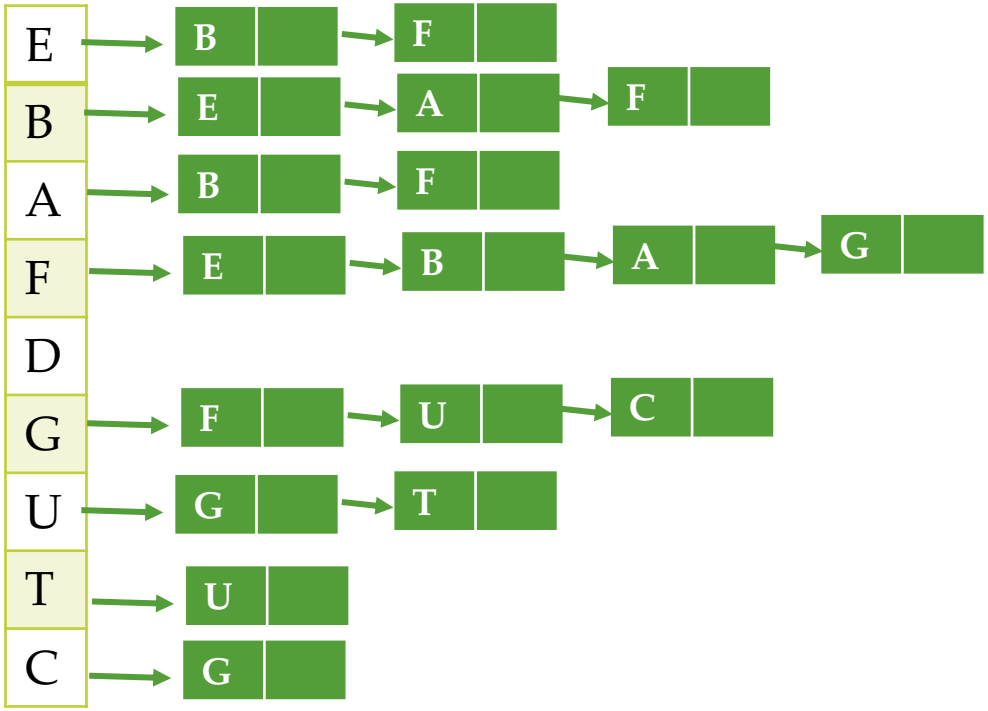
Claro que esa "lista de adyacencia" será diferente si estamos con un SimpleOrDefault o Multi.

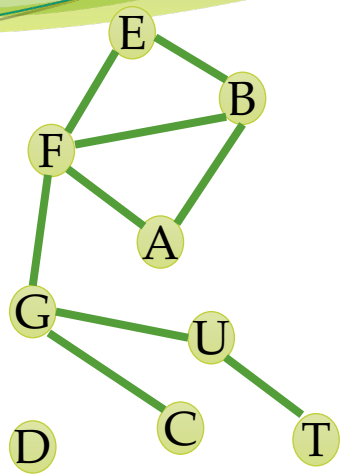
```
abstract public class AdjacencyListGraph<V, E> implements GraphService<V, E> {
private boolean isSimple;
protected boolean isDirected;
private boolean acceptSelfLoop;
private boolean isWeighted;
protected String type;

// HashMap no respeta el orden de insercion. En el testing considerar eso
private Map<V,Collection<InternalEdge>> adjacencyList=
            new HashMap<>();

// respeta el orden de llegada y facilita el testing. Asocia una lista por llegada.
//private Map<V,Collection<InternalEdge>> adjacencyList= new LinkedHashMap<>();
```

Opción 1 (la que usamos para la discusión): los vértices están en cualquier lado.
No requiere que los vértices implementen Comparable.

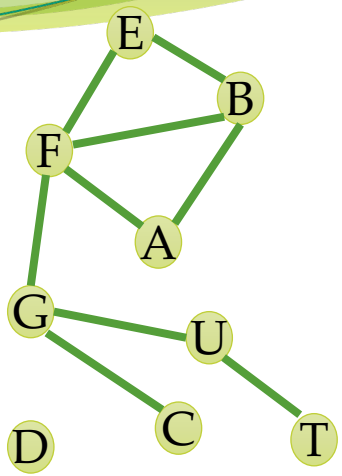
Opción 2: además del arreglo interno para el hash, cada slot indica quien llegó antes y quien después armando una lista doblemente encadenada. No requiere que los vértices implementen Comparable. Ocupa mucho espacio.

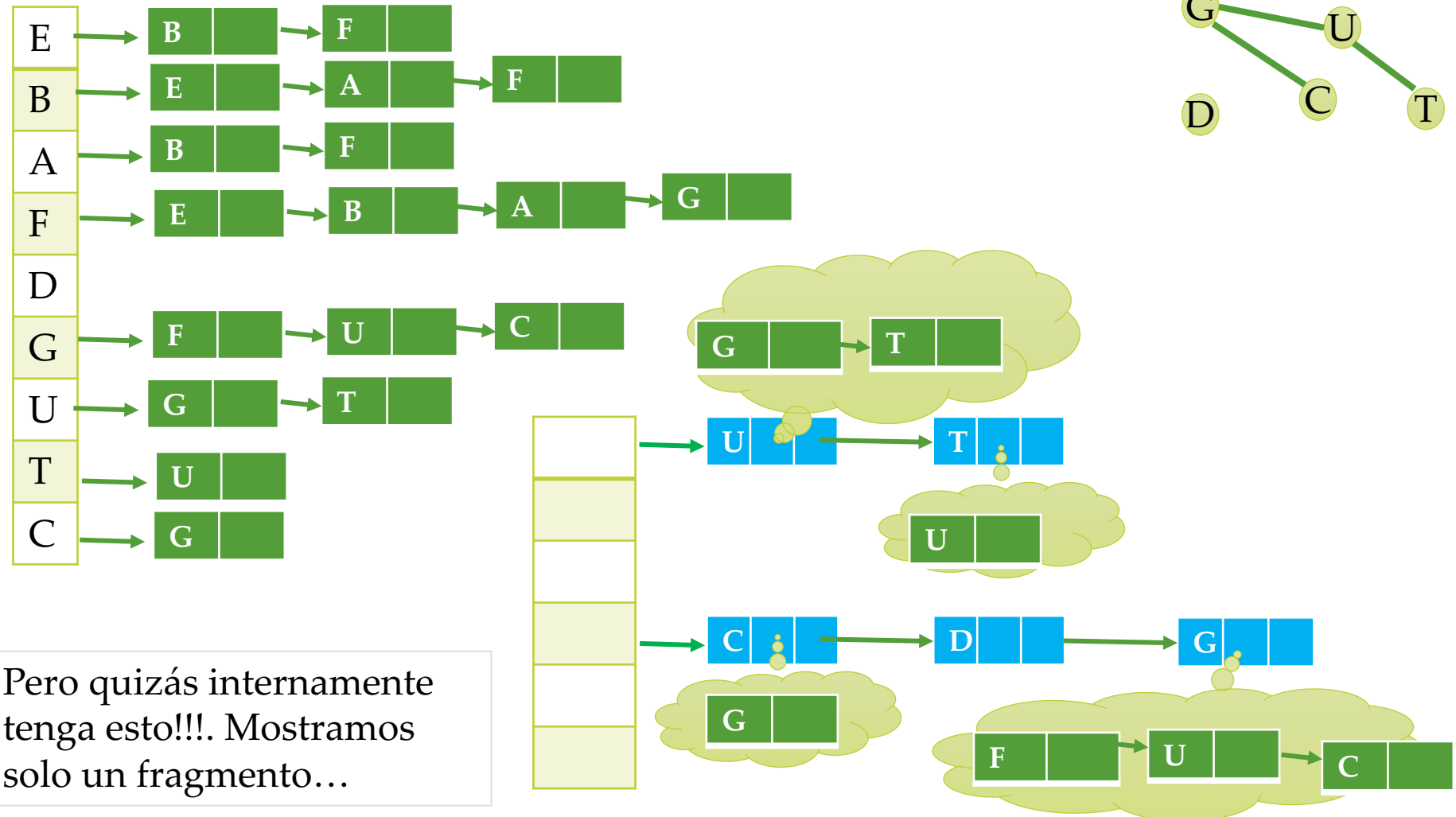
Con un HashMap yo quiero esto





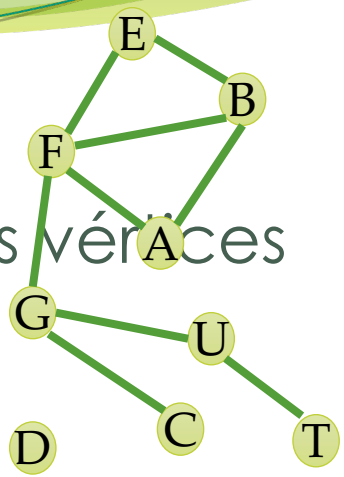
Con un HashMap yo quiero esto





Pero quizás internamente tenga esto!!!. Mostramos solo un fragmento…

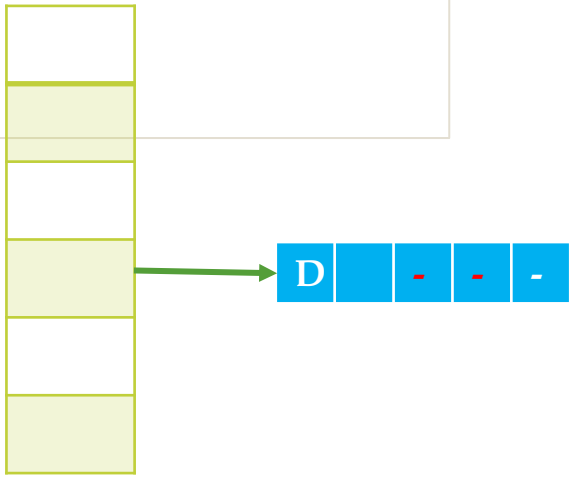
Con un LinkedHashMap con el orden de llegada a la inserción de los vértices



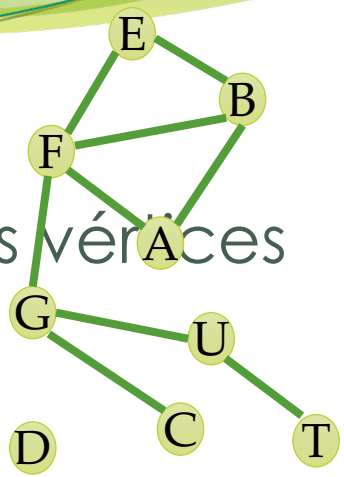
Si se hizo esto:
g.addVertex('D');
g.addEdge('U', 'G', new EmptyEdgeProp());
…

Internamente
tenga esto!!!. Mostramos solo un fragmento…
First: ptr a D

Cada nodo tiene:
Dato, Value, Prev, Next, Ptr



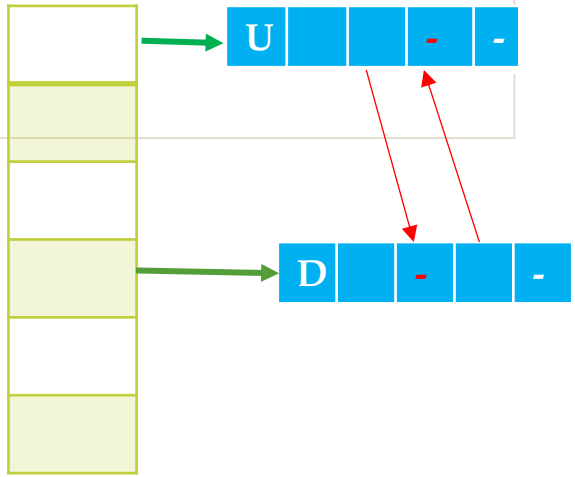
Con un LinkedHashMap con el orden de llegada a la inserción de los vértices



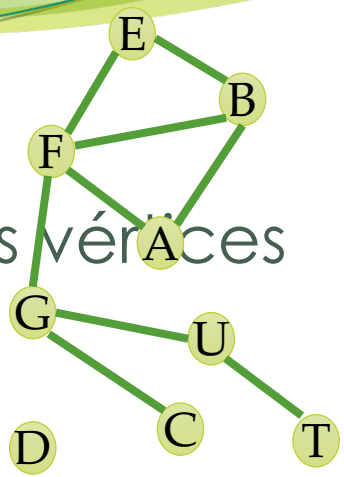
Si se hizo esto:
g.addVertex('D');
g.addEdge('U', 'G', new EmptyEdgeProp());
…

Internamente
tenga esto!!!. Mostramos solo un fragmento…
First: ptr a D

Cada nodo tiene:
Dato, Value, Prev, Next, Ptr



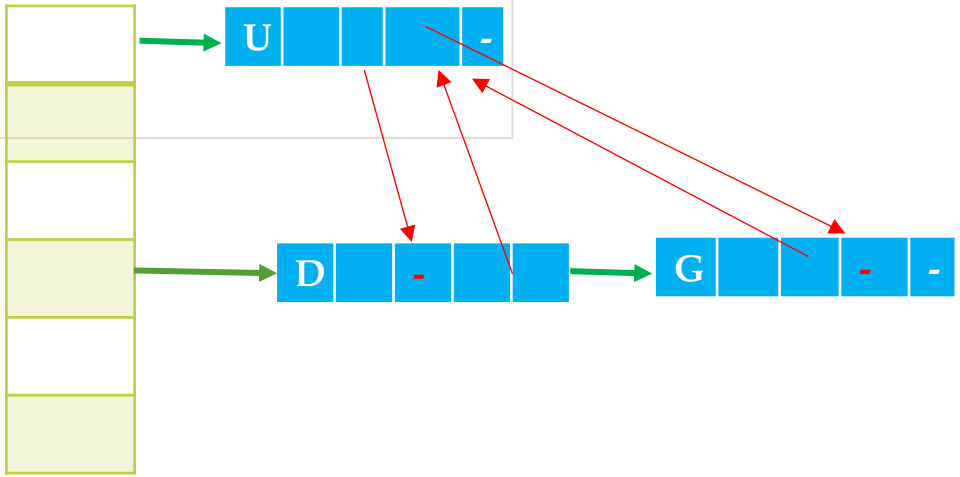
Con un LinkedHashMap con el orden de llegada a la inserción de los vértices



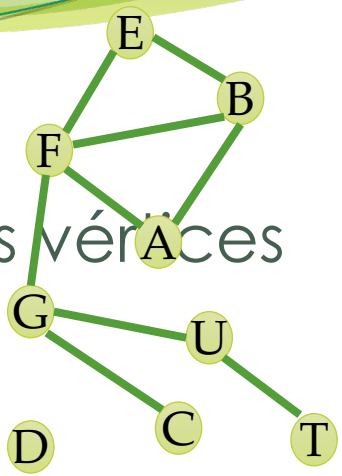
Si se hizo esto:
g.addVertex('D');
g.addEdge('U', 'G', new EmptyEdgeProp());
…

Internamente
tenga esto!!!. Mostramos solo un fragmento…
First: ptr a D

Cada nodo tiene:
Dato, Value, Prev, Next, Ptr



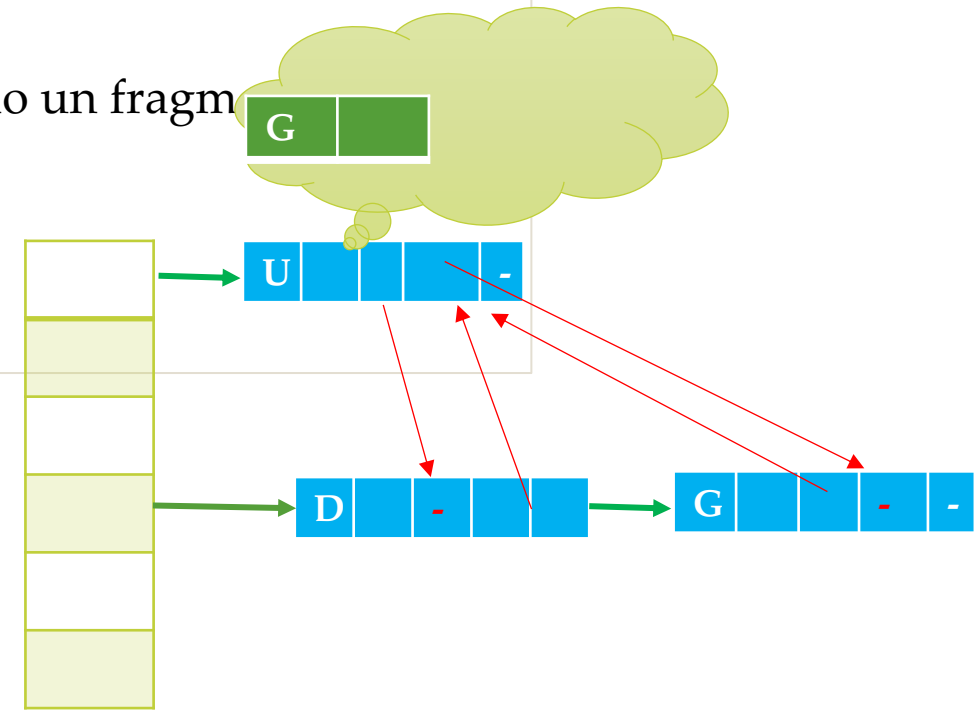
Con un LinkedHashMap con el orden de llegada a la inserción de los vértices



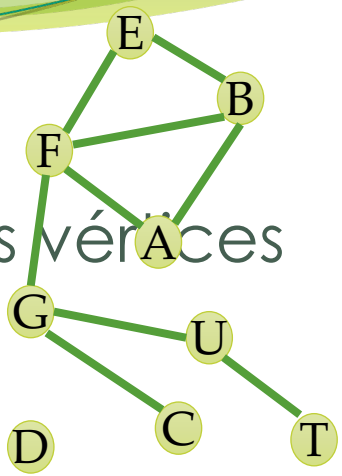
Si se hizo esto:
g.addVertex('D');
g.addEdge('U', 'G', new EmptyEdgeProp());
…

Internamente
tenga esto!!!. Mostramos solo un fragm
First: ptr a D

Cada nodo tiene:
Dato, Value, Prev, Next, Ptr



Con un LinkedHashMap con el orden de llegada a la inserción de los vértices

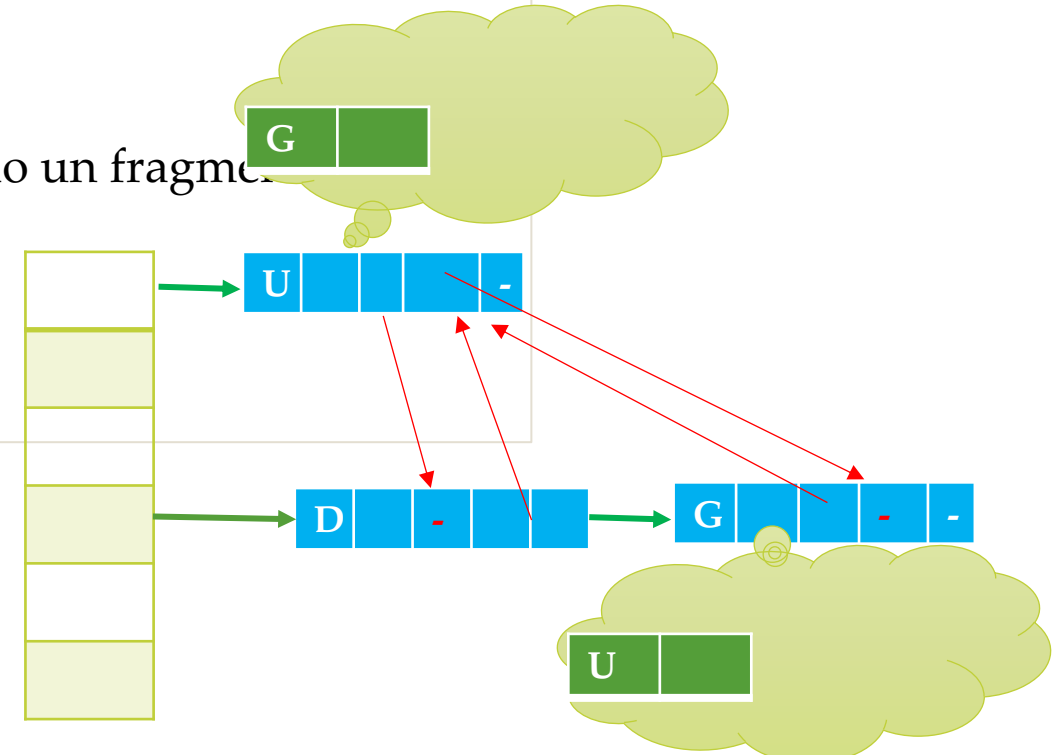


Si se hizo esto:
g.addVertex('D');
g.addEdge('U', 'G', new EmptyEdgeProp());

…

Internamente
tenga esto!!!. Mostramos solo un fragme
First: ptr a D

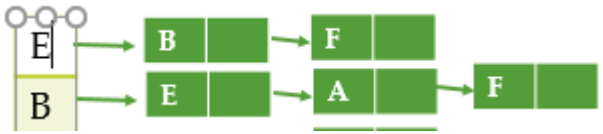
Cada nodo tiene:
Dato, Value, Prev, Next, Ptr



Nosotros usamos HashMap (no esperar ningún orden específico)

```
abstract public class AdjacencyListGraph<V, E> implements GraphService<V, E> {
private boolean isSimple;
protected boolean isDirected;
private boolean acceptSelfLoop;
private boolean isWeighted;
protected String type;

// HashMap no respeta el orden de insercion. En el testing considerar eso
private Map<V,   Collection<InternalEdge>> adjacencyList=
                new HashMap<>();
```

```
class InternalEdge {
    E edge;
    V target;
```



¿ Cómo implementamos **addVertex()** en **AdjacencyListGraph<V, E>** ?

```
@Override
public void addVertex(V aVertex) {

   if (aVertex == null )
      throw new IllegalArgumentException(…);

   // no edges yet
   getAdjacencyList().putIfAbsent(aVertex,
         new ArrayList<InternalEdge>());
}
```

TP 6 – Ejer 1.1

Implementar para todos los casos el metodo

public int addEdge(V v1, V v2, E ege)

¿ Cómo implementamos **addEdge()** en ?

- En SimpleOrDefault:
  - Invocar el método superclase que valida y crea vértices si es necesario
  - Lanzar exception si había eje entre ambos vértices.
  - Crear efectivamente el eje

- En Multi:
  - Invocar el método superclase que valida y crea vértices si es necesario
  - Crear efectivamente el eje.

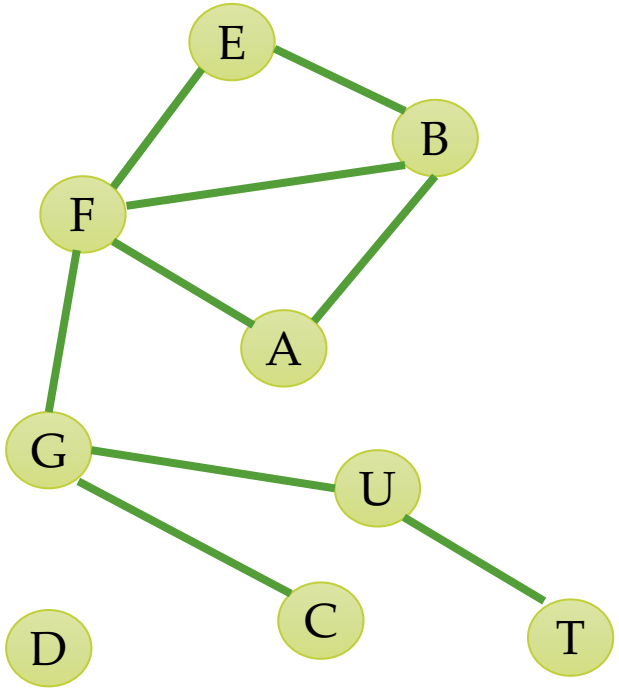
TP 6 – Ejer 1.2

Implementar para todos los casos el metodo

public int numberOfEdges()

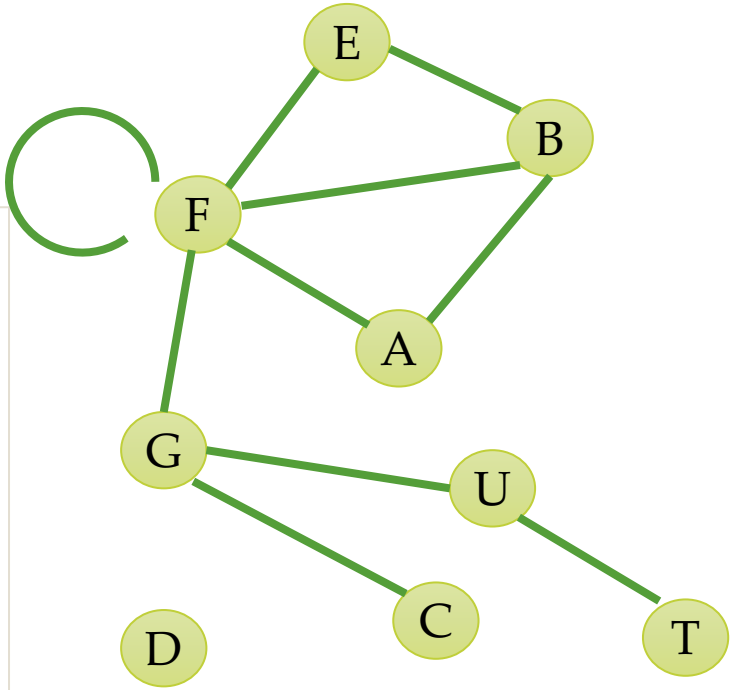
Caso de Uso A (no dirigido)

```
g.addEdge('E', 'B', new EmptyEdgeProp());
g.addEdge('A', 'B', new EmptyEdgeProp());
g.addEdge('F', 'B', new EmptyEdgeProp());
g.addVertex('D');
g.addVertex('G');
g.addEdge('E', 'F', new EmptyEdgeProp());
g.addEdge('F', 'A', new EmptyEdgeProp());
g.addEdge('F', 'G', new EmptyEdgeProp());
g.addEdge('U', 'G', new EmptyEdgeProp());
g.addEdge('T', 'U', new EmptyEdgeProp());
g.addEdge('C', 'G', new EmptyEdgeProp());

System.out.println( g.numberOfEdges() );  // 9
```



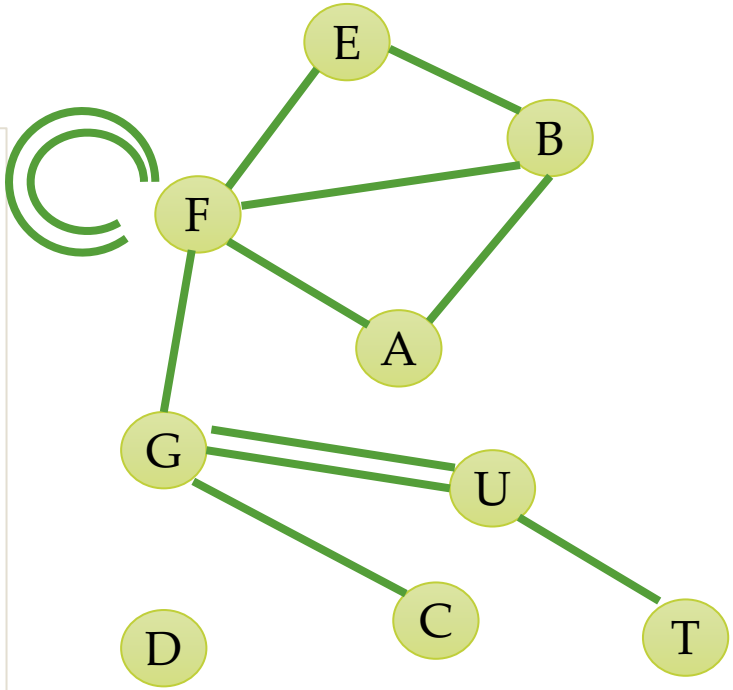
Caso de Uso B (no dirigido)

```
g.addEdge('E', 'B', new EmptyEdgeProp());
g.addEdge('A', 'B', new EmptyEdgeProp());
g.addEdge('F', 'B', new EmptyEdgeProp());
g.addVertex('D');
g.addVertex('G');
g.addEdge('E', 'F', new EmptyEdgeProp());
g.addEdge('F', 'A', new EmptyEdgeProp());
g.addEdge('F', 'G', new EmptyEdgeProp());
g.addEdge('U', 'G', new EmptyEdgeProp());
g.addEdge('T', 'U', new EmptyEdgeProp());
g.addEdge('C', 'G', new EmptyEdgeProp());
g.addEdge('F', 'F', new EmptyEdgeProp());

System.out.println( g.numberOfEdges() );  // 10
```



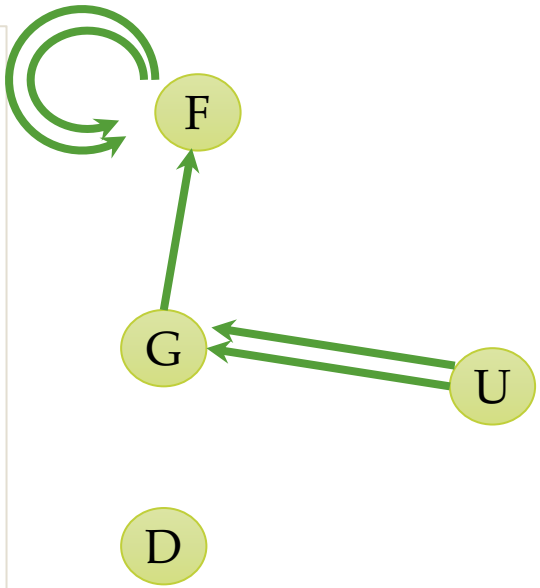
Caso de Uso C (no dirigido)

```
g.addEdge('E', 'B', new WeightedEdge(3));
g.addEdge('A', 'B', new WeightedEdge(1));
g.addEdge('F', 'B', new WeightedEdge(2));
g.addVertex('D');
g.addVertex('G');
g.addEdge('E', 'F', new WeightedEdge(-2));
g.addEdge('F', 'A', new WeightedEdge(8));
g.addEdge('F', 'G', new WeightedEdge(2));
g.addEdge('U', 'G', new WeightedEdge(-10));
g.addEdge('T', 'U', new WeightedEdge(8));
g.addEdge('C', 'G', new WeightedEdge(1));
g.addEdge('G', 'U', new WeightedEdge(0));
g.addEdge('F', 'F', new WeightedEdge(3));
g.addEdge('F', 'F', new WeightedEdge(2));

System.out.println( g.numberOfEdges() );  // 12
```



Caso de Uso D (dirigido)

```
g.addVertex('D');
g.addVertex('G');
g.addEdge('G', 'F', new WeightedEdge(2));
g.addEdge('U', 'G', new WeightedEdge(-10));
g.addEdge('U', 'G', new WeightedEdge(0));
g.addEdge('F', 'F', new WeightedEdge(3));
g.addEdge('F', 'F', new WeightedEdge(2));

System.out.println( g.numberOfEdges() );  // 5
```



TP 6 – Ejer 1.3

Atención: en un self-loop el degree cuenta doble

Implementar (ver especificaciòn)

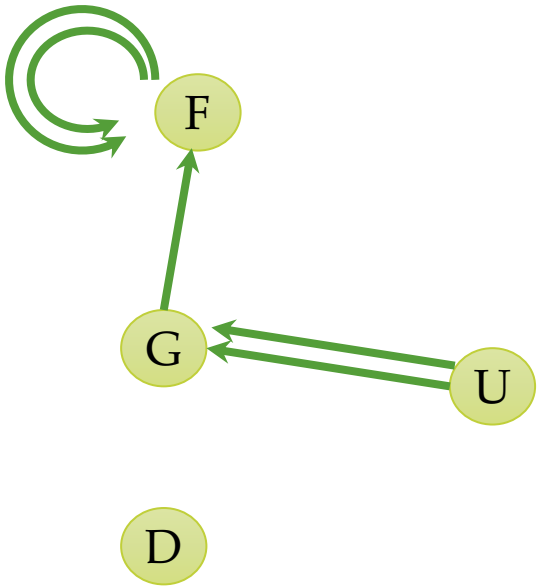
public int degree(V)

public int inDegree(V)

public int outDegree(V)

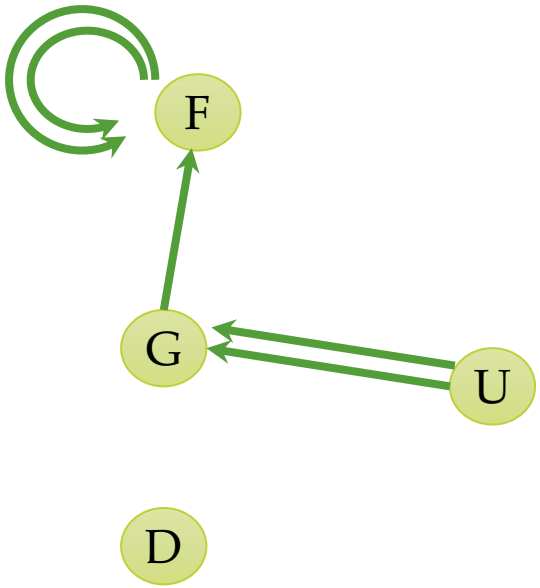
Caso de Uso A (dirigido)

```
g.addVertex('D');
g.addVertex('G');
g.addEdge('G', 'F', new WeightedEdge(2));
g.addEdge('U', 'G', new WeightedEdge(-10));
g.addEdge('U', 'G', new WeightedEdge(0));
g.addEdge('F', 'F', new WeightedEdge(3));
g.addEdge('F', 'F', new WeightedEdge(2));

System.out.println( g.degree('G') );

//  excepcion:  degree aplica para no dirigido
```



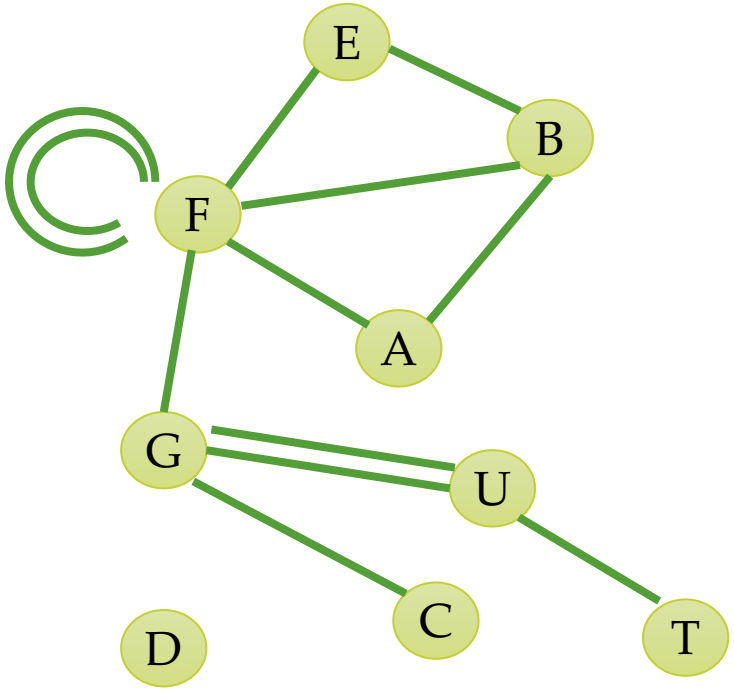
Caso de Uso B (dirigido)

```
g.addVertex('D');
g.addVertex('G');
g.addEdge('G', 'F', new WeightedEdge(2));
g.addEdge('U', 'G', new WeightedEdge(-10));
g.addEdge('U', 'G', new WeightedEdge(0));
g.addEdge('F', 'F', new WeightedEdge(3));
g.addEdge('F', 'F', new WeightedEdge(2));

System.out.println( g.inDegree('G') );     // 2
System.out.println( g.outDegree('G') );  // 1


System.out.println( g.inDegree('F') );     // 3
System.out.println( g.outDegree('F') );  //  2
```



Caso de Uso C (no dirigido)

```
g.addEdge('E', 'B', new WeightedEdge(3));
g.addEdge('A', 'B', new WeightedEdge(1));
g.addEdge('F', 'B', new WeightedEdge(2));
g.addVertex('D');
g.addVertex('G');
g.addEdge('E', 'F', new WeightedEdge(-2));
g.addEdge('F', 'A', new WeightedEdge(8));
g.addEdge('F', 'G', new WeightedEdge(2));
g.addEdge('U', 'G', new WeightedEdge(-10));
g.addEdge('T', 'U', new WeightedEdge(8));
g.addEdge('C', 'G', new WeightedEdge(1));
g.addEdge('G', 'U', new WeightedEdge(0));
g.addEdge('F', 'F', new WeightedEdge(3));
g.addEdge('F', 'F', new WeightedEdge(2));

System.out.println( g.degree('G') );    // 4
```



TP 6 – Ejer 1.4

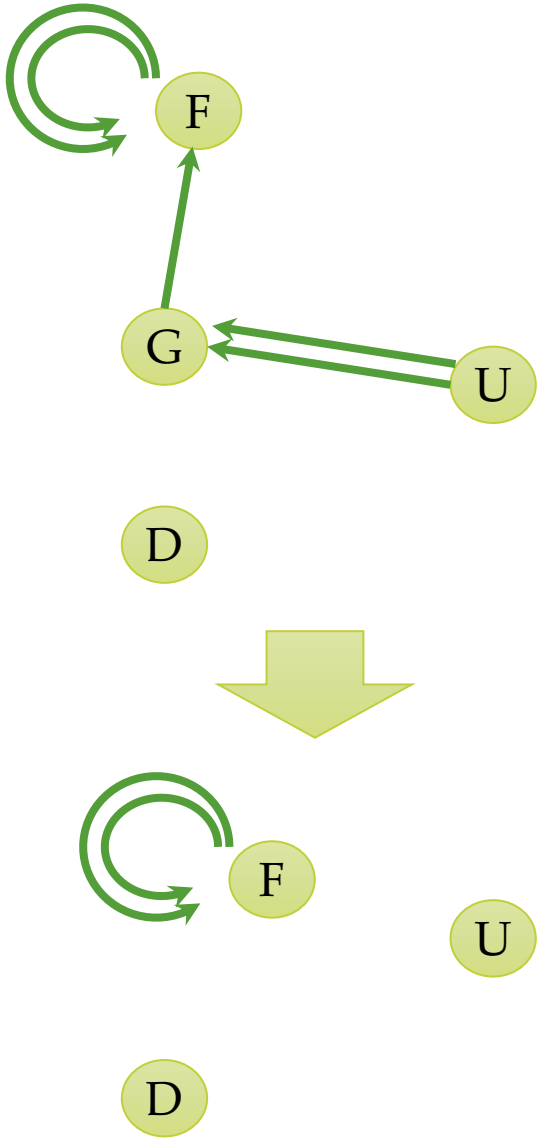
Aclaracion: recorrer lo menos posible!!!

Implementar (ver especificaciòn)

public boolean removeVertex(V)

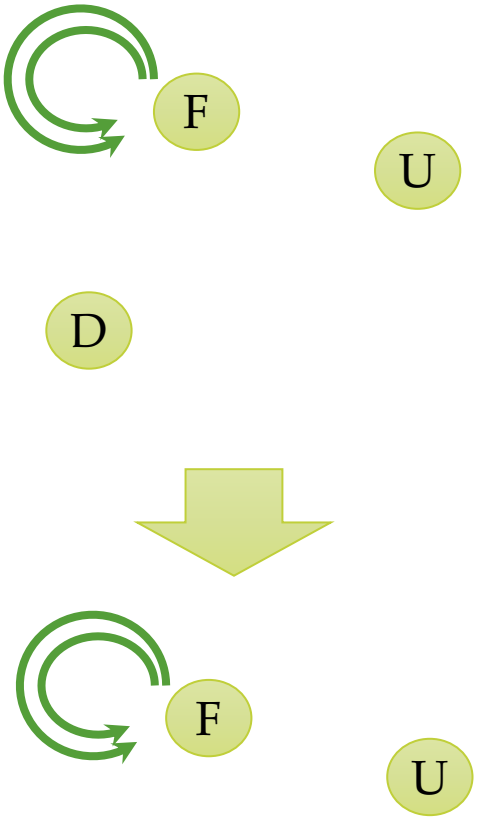
Caso de Uso A (dirigido)

```
g.addVertex('D');
g.addVertex('G');
g.addEdge('G', 'F', new WeightedEdge(2));
g.addEdge('U', 'G', new WeightedEdge(-10));
g.addEdge('U', 'G', new WeightedEdge(0));
g.addEdge('F', 'F', new WeightedEdge(3));
g.addEdge('F', 'F', new WeightedEdge(2));

g.removeVertex('G' );
```



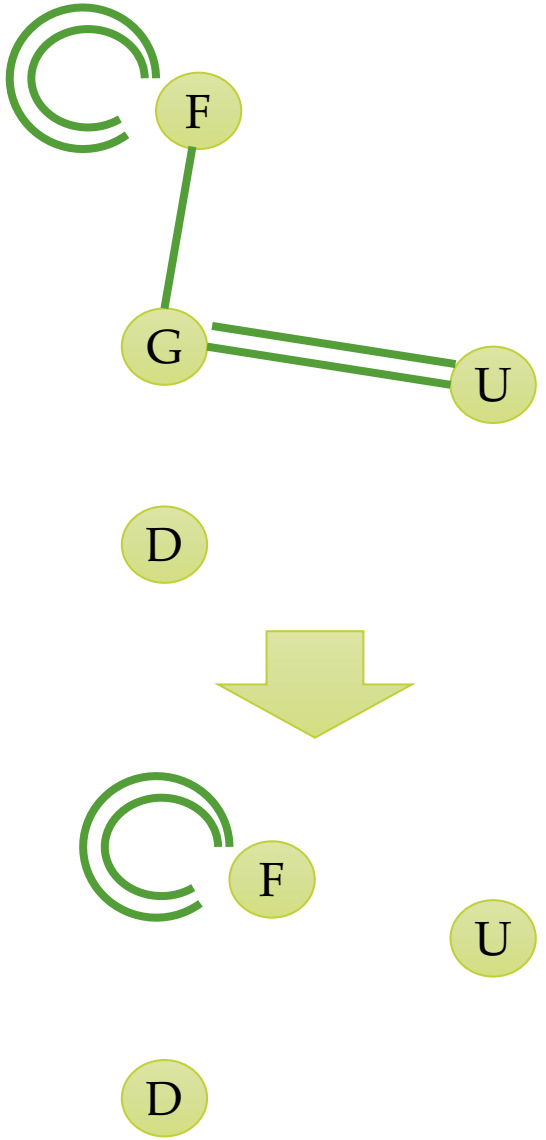
Caso de Uso B (dirigido)

```
g.addVertex('D');
g.addVertex('U');
g.addEdge('F', 'F', new WeightedEdge(3));
g.addEdge('F', 'F', new WeightedEdge(2));

g.removeVertex('D' );
```



Caso de Uso C (no dirigido)

```
g.addVertex('D');
g.addVertex('G');
g.addEdge('G', 'F', new WeightedEdge(2));
g.addEdge('U', 'G', new WeightedEdge(-10));
g.addEdge('U', 'G', new WeightedEdge(0));
g.addEdge('F', 'F', new WeightedEdge(3));
g.addEdge('F', 'F', new WeightedEdge(2));

g.removeVertex('G' );
```



TP 6 – Ejer 1.5

Implementar (ver especificaciòn)

public boolean removeEdge(V aVertex, V otherVertex, E theEdge)

public boolean removeEdge(V aVertex, V otherVertex)

Recorridos en Grafos

Recorridos (traversal) en grafos se usan para ver los nodos alcanzables a partir de un nodo y se muestra dicho camino.

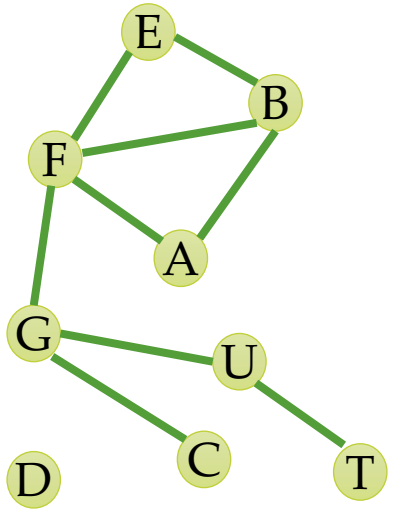
Aclaración: Si fuera un multigrafo (para que no haya ambigüedades) podría indicar cual eje tome en cada caso para la navegación. Igual, los vértices los sigo visitando una sola vez.

Los métodos son:

1) Breadth-First Search (BFS)
2) Depth-First Search (DFS)

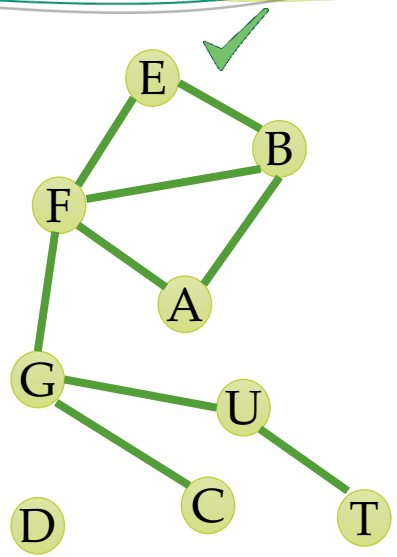
g.printBFS('E')

	0	1	2	3	4	5	6	7	8
E	F	T	F	T	F	F	F	F	F
B	T	F	T	T	F	F	F	F	F
A	F	T	F	T	F	F	F	F	F
F	T	T	T	F	F	T	F	F	F
D	F	F	F	F	F	F	F	F	F
G	F	F	F	T	F	F	T	F	T
U	F	F	F	F	F	T	F	T	F
T	F	F	F	F	F	F	T	F	F
C	F	F	F	F	F	T	F	F	F



E	B	A	F	D	G	U	T	C

g.printBFS('E')



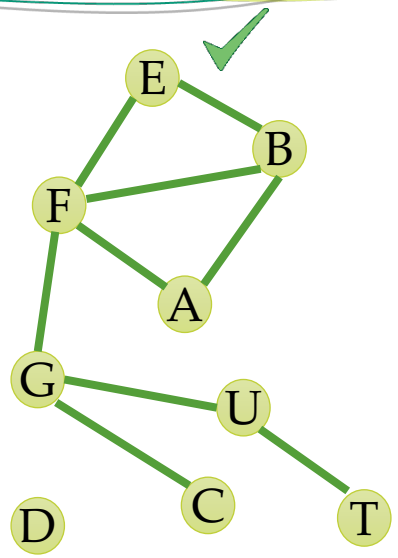
	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

E	B	A	F	D	G	U	T	C

E

g.printBFS('E')

	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F



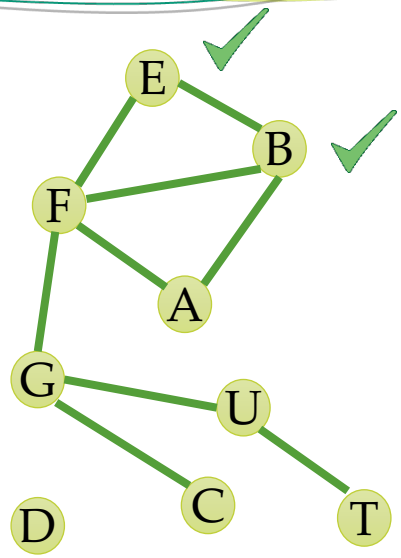
E	B	A	F	D	G	U	T	C

B	F

Vecinos de E no marcados como visitados aun.

g.printBFS('E')

	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

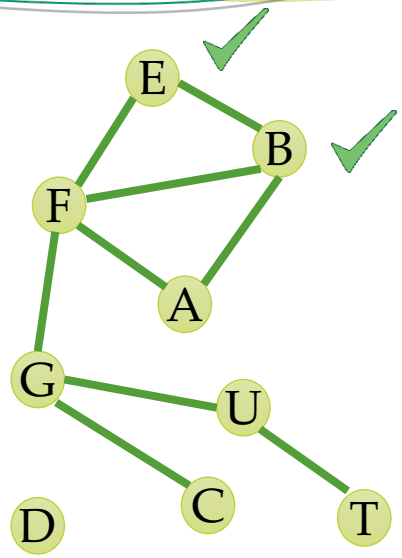


E
B

F

E	B	A	F	D	G	U	T	C

g.printBFS('E')



	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

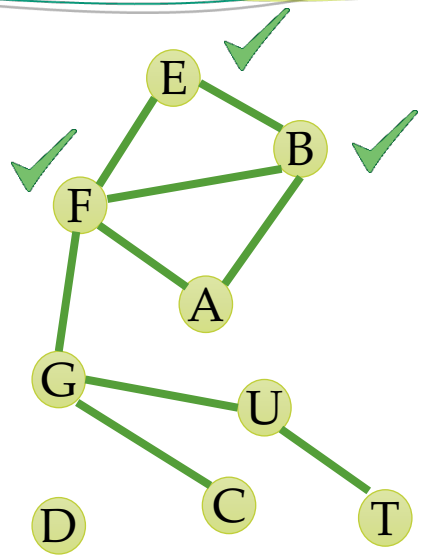
E	B	A	F	D	G	U	T	C

E
B

F	A	F

Vecinos de B no marcados como visitados aun.

g.printBFS('E')



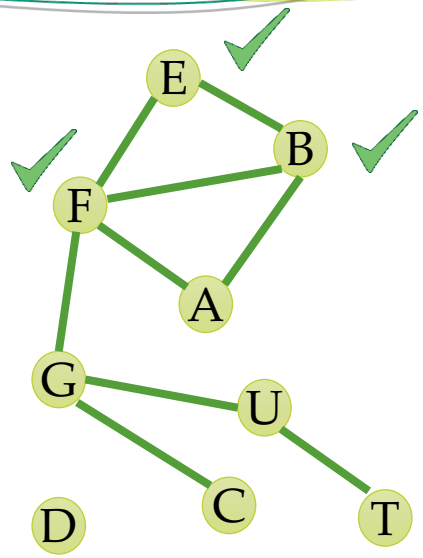
	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

E	B	A	F	D	G	U	T	C

E
B
F

A	F

g.printBFS('E')



	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

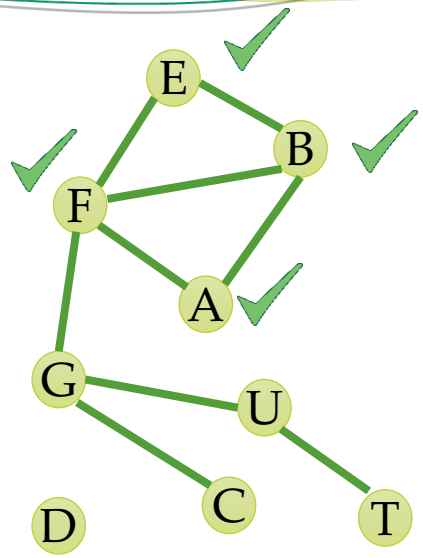
E	B	A	F	D	G	U	T	C

E
B
F

A	F	A	G

Vecinos de F no marcados como visitados aun.

g.printBFS('E')



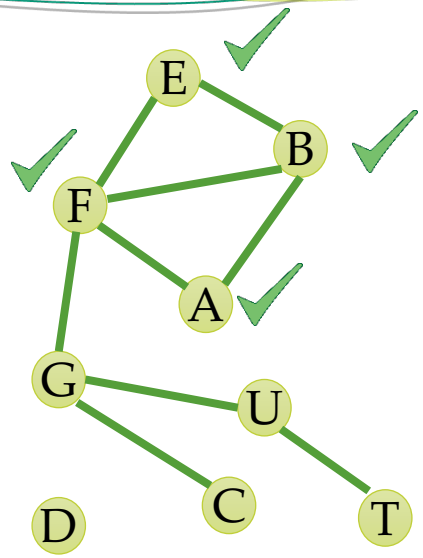
	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

E	B	A	F	D	G	U	T	C

E
B
F
A

F	A	G

g.printBFS('E')



	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

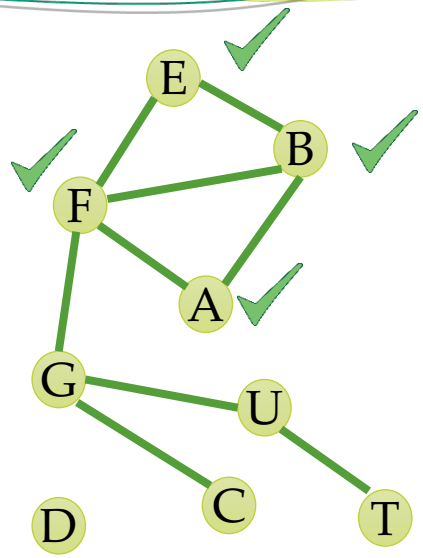
E	B	A	F	D	G	U	T	C

E
B
F
A

F	A	G

A no tiene vecinos no visitados. No agrega nada

g.printBFS('E')



	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

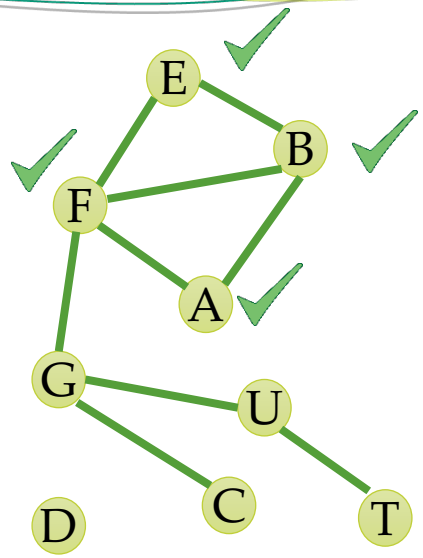
E	B	A	F	D	G	U	T	C

E
B
F
A

A	G

A no se procesa (se ignora) porque ya fue visitado

g.printBFS('E')



	0	1	2	3	4	5	6	7	8
E	**0**	F	T	F	T	F	F	F	F	F
B	**1**	T	F	T	T	F	F	F	F	F
A	**2**	F	T	F	T	F	F	F	F	F
F	**3**	T	T	T	F	F	T	F	F	F
D	**4**	F	F	F	F	F	F	F	F	F
G	**5**	F	F	F	T	F	F	T	F	T
U	**6**	F	F	F	F	F	T	F	T	F
T	**7**	F	F	F	F	F	F	T	F	F
C	**8**	F	F	F	F	F	T	F	F	F

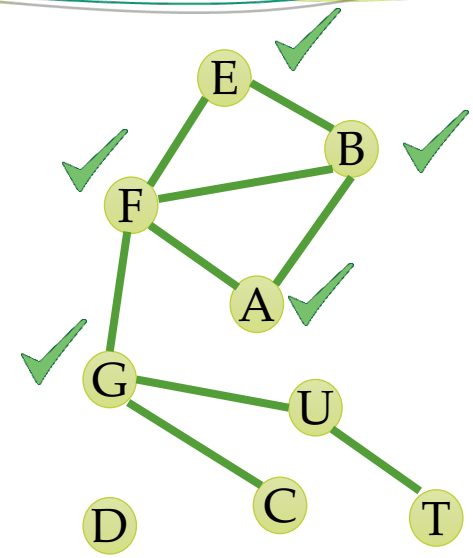
E	B	A	F	D	G	U	T	C

E
B
F
A

G

G no se procesa (se ignora) porque ya fue visitado

g.printBFS('E')



	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

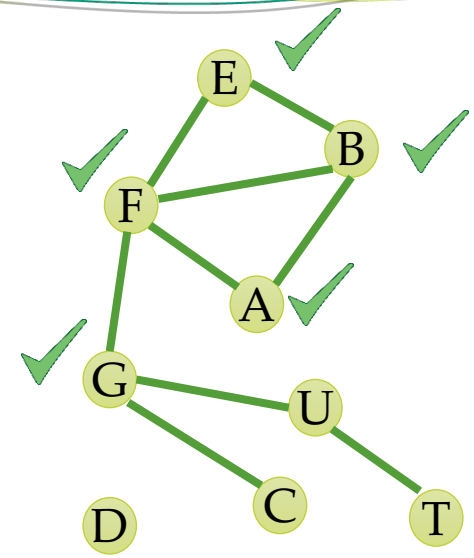
E	B	A	F	D	G	U	T	C

E
B
F
A
G

g.printBFS('E')

	0	1	2	3	4	5	6	7	8
E	F	T	F	T	F	F	F	F	F
B	T	F	T	T	F	F	F	F	F
A	F	T	F	T	F	F	F	F	F
F	T	T	T	F	F	T	F	F	F
D	F	F	F	F	F	F	F	F	F
G	F	F	F	T	F	F	T	F	T
U	F	F	F	F	F	T	F	T	F
T	F	F	F	F	F	F	T	F	F
C	F	F	F	F	F	T	F	F	F

E	B	A	F	D	G	U	T	C

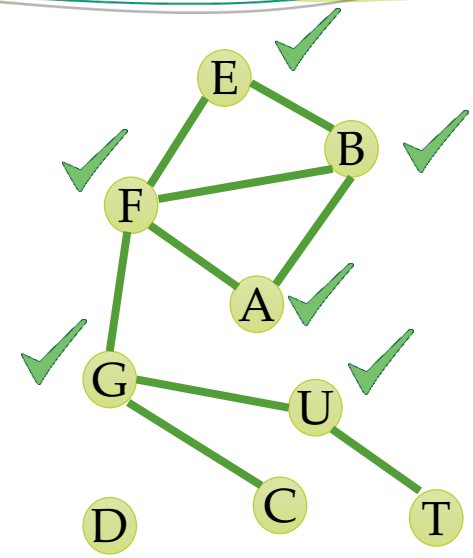


E
B
F
A
G

U	C

Vecinos de G no marcados como visitados aun.

g.printBFS('E')



	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

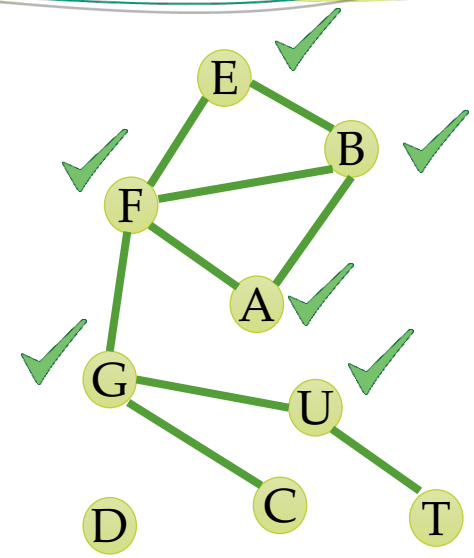
E	B	A	F	D	G	U	T	C

E
B
F
A
G
U

C

g.printBFS('E')

	0	1	2	3	4	5	6	7	8
E	F	T	F	T	F	F	F	F	F
B	T	F	T	T	F	F	F	F	F
A	F	T	F	T	F	F	F	F	F
F	T	T	T	F	F	T	F	F	F
D	F	F	F	F	F	F	F	F	F
G	F	F	F	T	F	F	T	F	T
U	F	F	F	F	F	T	F	T	F
T	F	F	F	F	F	F	T	F	F
C	F	F	F	F	F	T	F	F	F



E	B	A	F	D	G	U	T	C

E
B
F
A
G
U

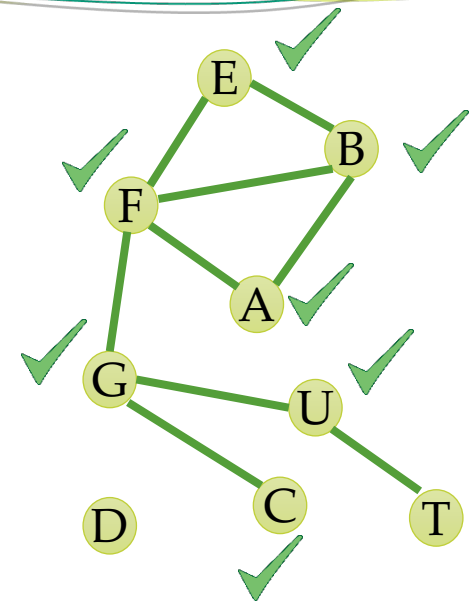
C	T

Vecinos de U no marcados como visitados aun.

g.printBFS('E')

	0	1	2	3	4	5	6	7	8
E	F	T	F	T	F	F	F	F	F
B	T	F	T	T	F	F	F	F	F
A	F	T	F	T	F	F	F	F	F
F	T	T	T	F	F	T	F	F	F
D	F	F	F	F	F	F	F	F	F
G	F	F	F	T	F	F	T	F	T
U	F	F	F	F	F	T	F	T	F
T	F	F	F	F	F	F	T	F	F
C	F	F	F	F	F	T	F	F	F

E	B	A	F	D	G	U	T	C



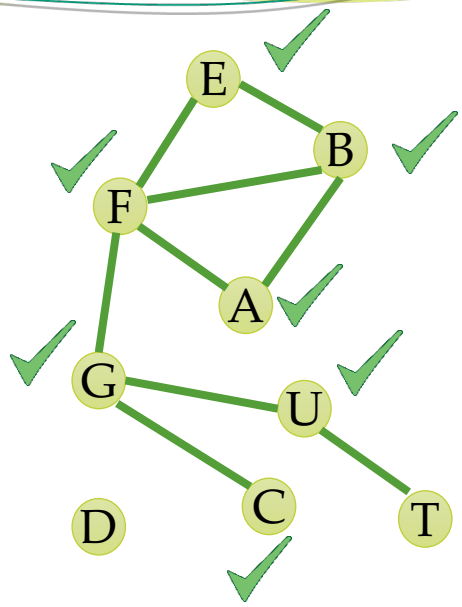
E
B
F
A
G
U
C

T

g.printBFS('E')

	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

E	B	A	F	D	G	U	T	C



E
B
F
A
G
U
C

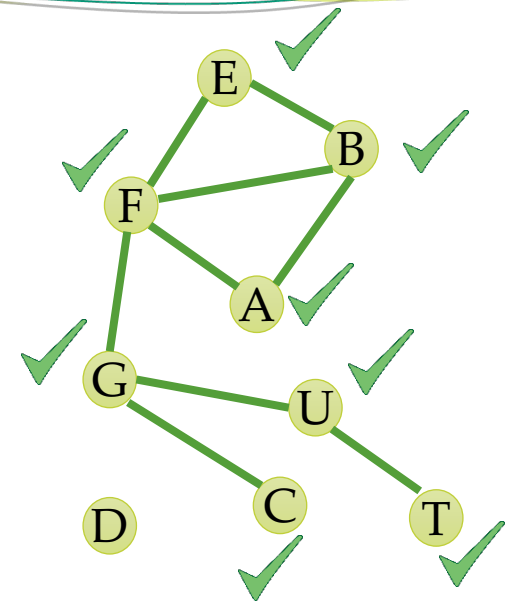
T

C no tiene vecinos no visitados. No agrega nada

g.printBFS('E')

	0	1	2	3	4	5	6	7	8
E	F	T	F	T	F	F	F	F	F
B	T	F	T	T	F	F	F	F	F
A	F	T	F	T	F	F	F	F	F
F	T	T	T	F	F	T	F	F	F
D	F	F	F	F	F	F	F	F	F
G	F	F	F	T	F	F	T	F	T
U	F	F	F	F	F	T	F	T	F
T	F	F	F	F	F	F	T	F	F
C	F	F	F	F	F	T	F	F	F

E	B	A	F	D	G	U	T	C

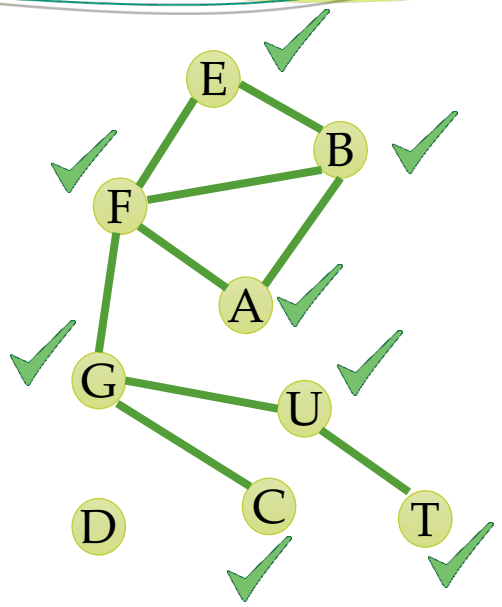


E
B
F
A
G
U
C
T

g.printBFS('E')

	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

E	B	A	F	D	G	U	T	C



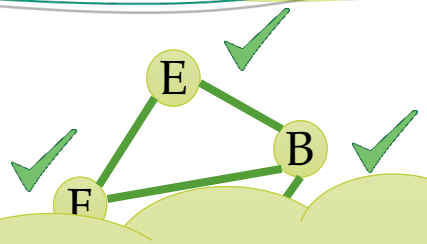
E
B
F
A
G
U
C
T

T no tiene vecinos no visitados. No agrega nada

g.printBFS('E')

	0	1	2	3	4	5	6	7	8
E	F	T	F	T	F	F	F	F	
B	T	F	T	T	F	F	F	F	
A	F	T	F	T	F	F	F	F	F
F	T	T	T	F	F	T	F	F	F
D	F	F	F	F	F	F	F	F	F
G	F	F	F	T	F	F	T	F	T
U	F	F	F	F	F	T	F	T	F
T	F	F	F	F	F	F	T	F	F
C	F	F	F	F	F	T	F	F	F

E	B	A	F	D	G	U	T	C

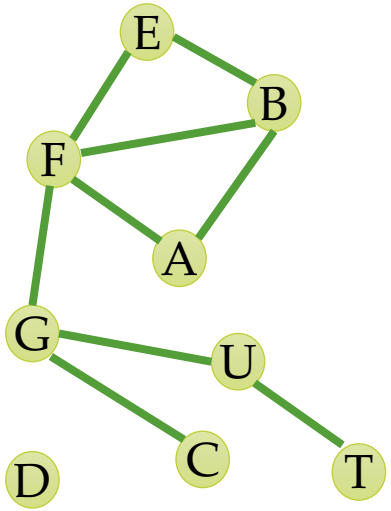


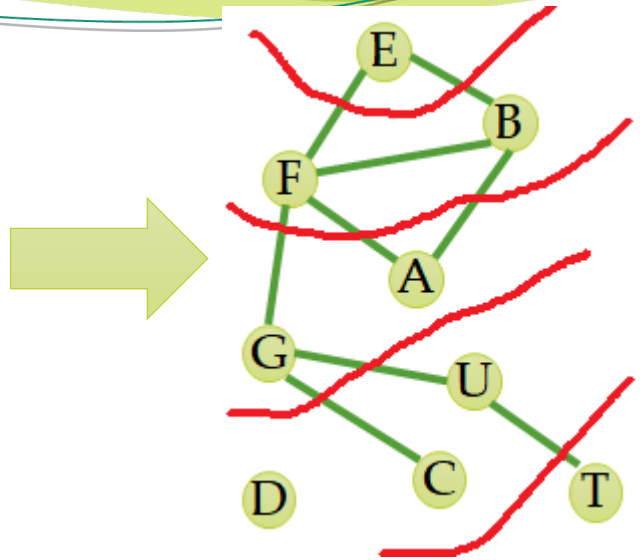
¿A qué algoritmo que vimos en arboles se parece?
¿Qué estructura auxiliar hay que usar?

E
B
F
A
G
U
C
T

g.printBFS('E')

E
B
F
A
G
U
C
T





Atención: Dependen de cómo se almacena todo, el orden obtenido puede variar. Enunciar TODOS los otros posibles recorridos BFS desde E válidos.

Ej:

E B F A G C U T
E B F G A U C T
E B F G A C U T
E F B A G U C T

printBFS(V) clásico. Tiene que estar OK para todo tipo de grafo.

Respecto a la implementación. ¿Qué estructura auxiliar usa????

Rta: Queue. Es como navegar por niveles.

Y además (es un grafo!!!) precisa marcar los vértices por los que pasó:

Opción 1: estructura paralela a los vértices booleana

Opción 2: representar el vértice con un tag booleano

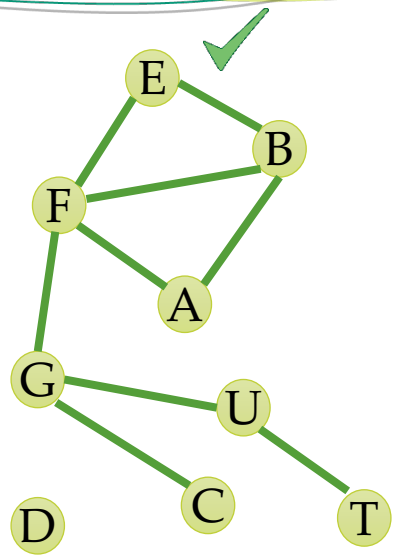
Escribamos el código Java para lista de adyacencia.

Agregamos en la interface.

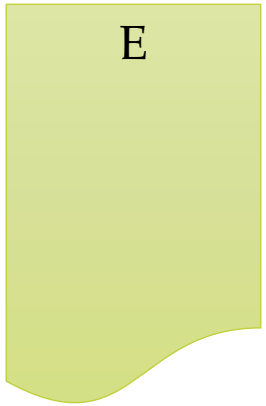
public void printBFS(V source);

Ahora el otro recorrido

g.printDFS('E')



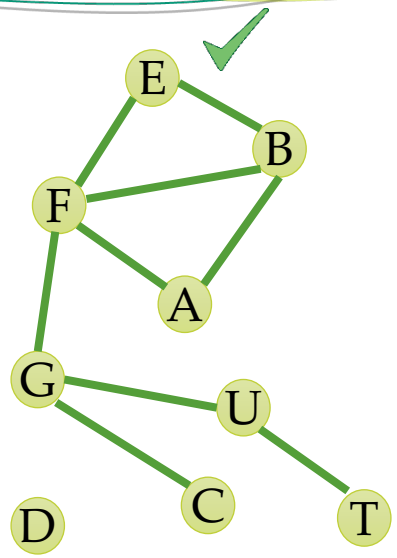
	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F



E	B	A	F	D	G	U	T	C

g.printDFS('E')

	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F



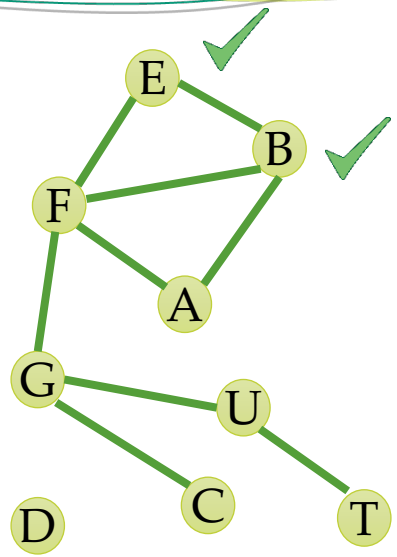
E	B	A	F	D	G	U	T	C

B
F

Vecinos de E no marcados como visitados aun.

g.printDFS('E')

	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F



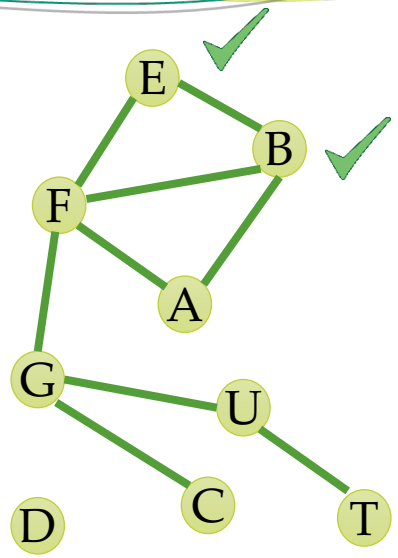
E
B

F

E	B	A	F	D	G	U	T	C

g.printDFS('E')

	0	1	2	3	4	5	6	7	8
E	F	T	F	T	F	F	F	F	F
B	T	F	T	T	F	F	F	F	F
A	F	T	F	T	F	F	F	F	F
F	T	T	T	F	F	T	F	F	F
D	F	F	F	F	F	F	F	F	F
G	F	F	F	T	F	F	T	F	T
U	F	F	F	F	F	T	F	T	F
T	F	F	F	F	F	F	T	F	F
C	F	F	F	F	F	T	F	F	F



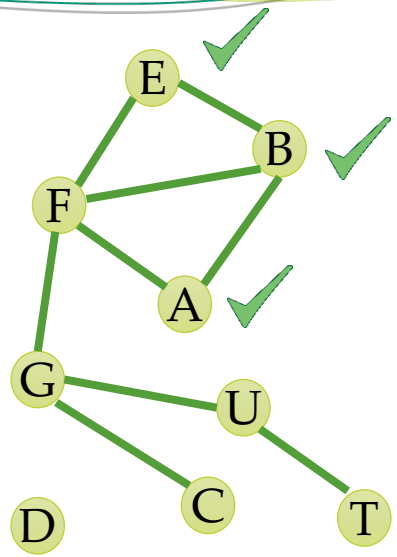
E	B	A	F	D	G	U	T	C

E
B

A
F
F

Vecinos de B no marcados como visitados aun.

g.printDFS('E')



	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

E	B	A	F	D	G	U	T	C

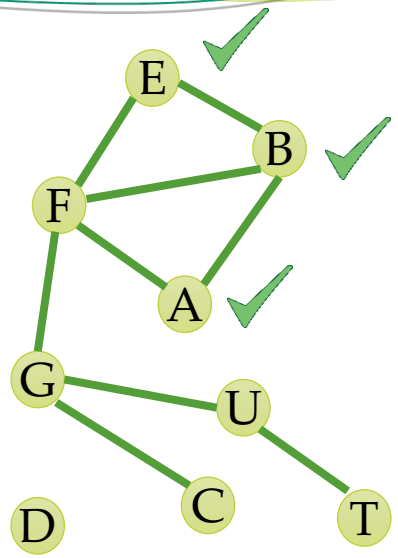
E
B
A

F
F

g.printDFS('E')

	0	1	2	3	4	5	6	7	8
E	F	T	F	T	F	F	F	F	F
B	T	F	T	T	F	F	F	F	F
A	F	T	F	T	F	F	F	F	F
F	T	T	T	F	F	T	F	F	F
D	F	F	F	F	F	F	F	F	F
G	F	F	F	T	F	F	T	F	T
U	F	F	F	F	F	T	F	T	F
T	F	F	F	F	F	F	T	F	F
C	F	F	F	F	F	T	F	F	F

E	B	A	F	D	G	U	T	C

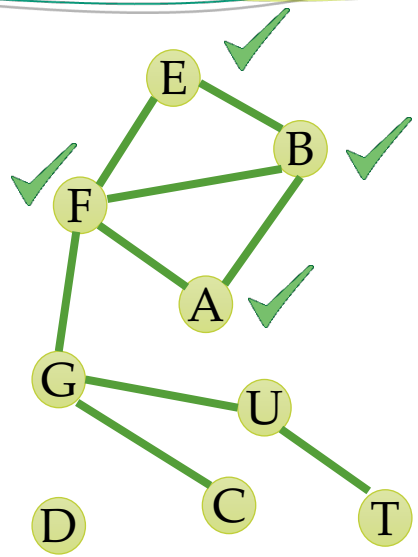


E
B
A

F
F
F

Vecinos de A no marcados como visitados aun.

g.printDFS('E')



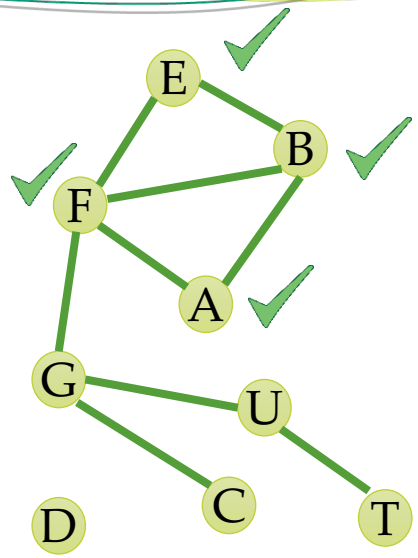
	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

E	B	A	F	D	G	U	T	C

E
B
A
F

F
F

g.printDFS('E')



	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

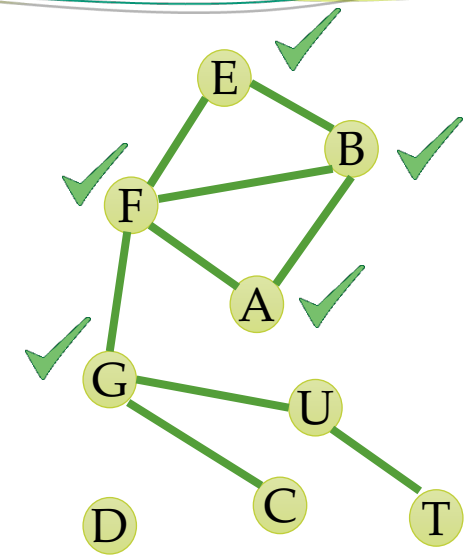
E	B	A	F	D	G	U	T	C

E
B
A
F

G
F
F

Vecinos de F no marcados como visitados aun.

g.printDFS('E')



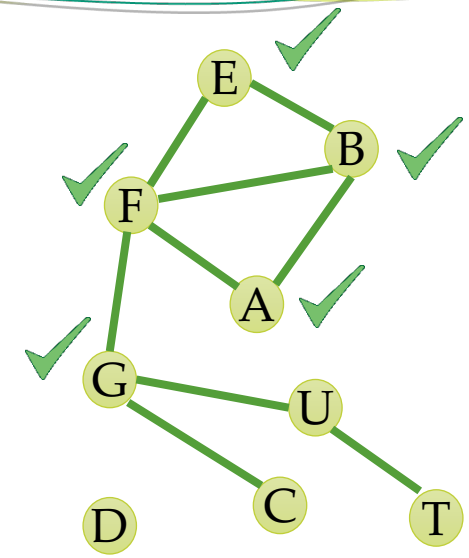
	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

E	B	A	F	D	G	U	T	C

E
B
A
F
G

F
F

g.printDFS('E')



	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

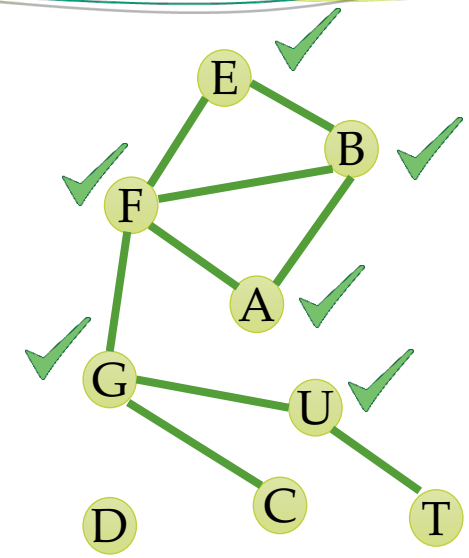
E	B	A	F	D	G	U	T	C

E
B
A
F
G

U
C
F
F

Vecinos de G no marcados como visitados aun.

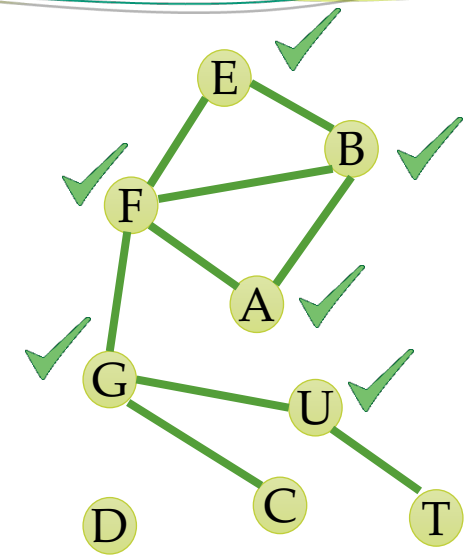
g.printDFS('E')



	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

E	B	A	F	D	G	U	T	C

g.printDFS('E')



	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

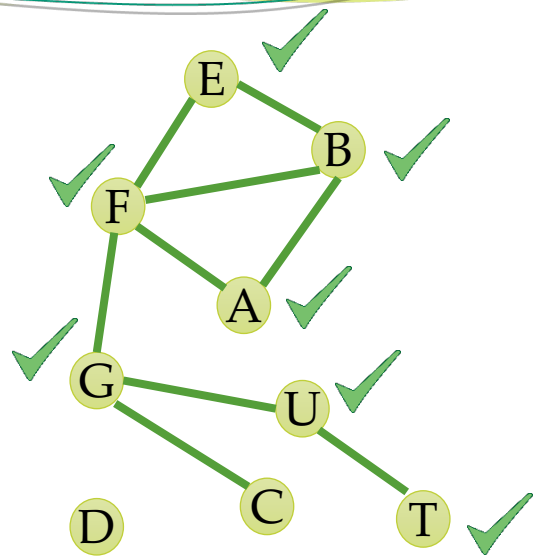
E	B	A	F	D	G	U	T	C

E
B
A
F
G
U

T
C
F
F

Vecinos de U no marcados como visitados aun.

g.printDFS('E')



	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

E	B	A	F	D	G	U	T	C

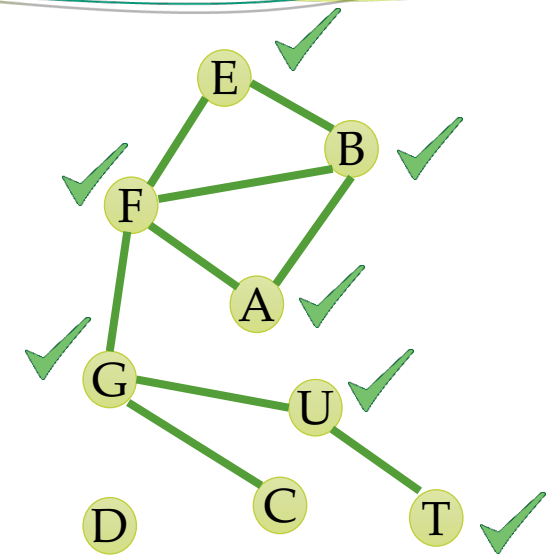
E
B
A
F
G
U
T

C
F
F

g.printDFS('E')

	0	1	2	3	4	5	6	7	8
E	F	T	F	T	F	F	F	F	F
B	T	F	T	T	F	F	F	F	F
A	F	T	F	T	F	F	F	F	F
F	T	T	T	F	F	T	F	F	F
D	F	F	F	F	F	F	F	F	F
G	F	F	F	T	F	F	T	F	T
U	F	F	F	F	F	T	F	T	F
T	F	F	F	F	F	F	T	F	F
C	F	F	F	F	F	T	F	F	F

E	B	A	F	D	G	U	T	C



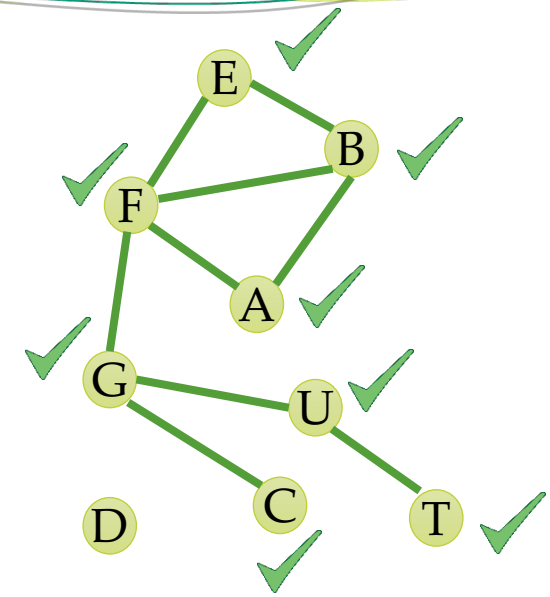
T no tiene vecinos no visitados. No agrega nada

E
B
A
F
G
U
T

C
F
F

g.printDFS('E')

	0	1	2	3	4	5	6	7	8
E	F	T	F	T	F	F	F	F	F
B	T	F	T	T	F	F	F	F	F
A	F	T	F	T	F	F	F	F	F
F	T	T	T	F	F	T	F	F	F
D	F	F	F	F	F	F	F	F	F
G	F	F	F	T	F	F	T	F	T
U	F	F	F	F	F	T	F	T	F
T	F	F	F	F	F	F	T	F	F
C	F	F	F	F	F	T	F	F	F



E	B	A	F	D	G	U	T	C

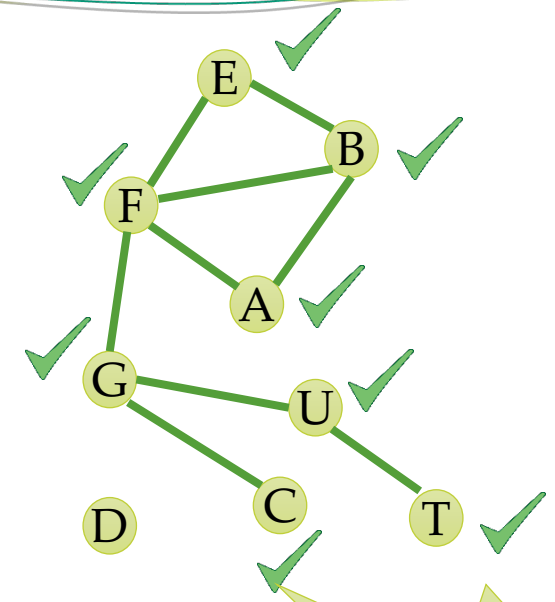
E
B
A
F
G
U
T
C

F
F

g.printDFS('E')

	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

E	B	A	F	D	G	U	T	C



C no tiene vecinos no visitados. No agrega nada

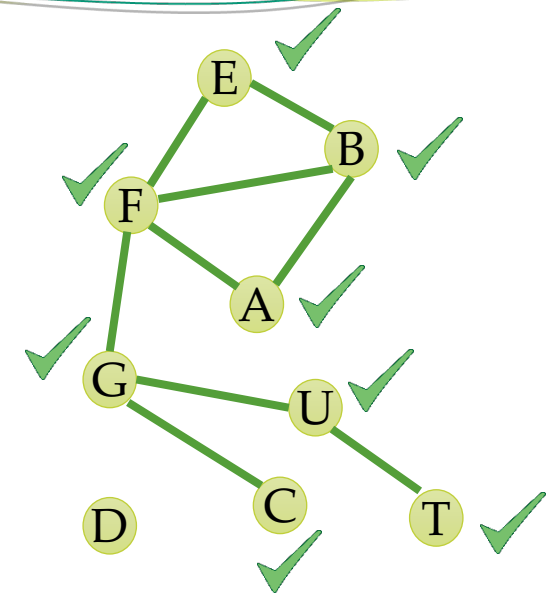
E
B
A
F
G
U
T
C

F
F

g.printDFS('E')

	0	1	2	3	4	5	6	7	8
E	0	F	T	F	T	F	F	F	F	F
B	1	T	F	T	T	F	F	F	F	F
A	2	F	T	F	T	F	F	F	F	F
F	3	T	T	T	F	F	T	F	F	F
D	4	F	F	F	F	F	F	F	F	F
G	5	F	F	F	T	F	F	T	F	T
U	6	F	F	F	F	F	T	F	T	F
T	7	F	F	F	F	F	F	T	F	F
C	8	F	F	F	F	F	T	F	F	F

E	B	A	F	D	G	U	T	C



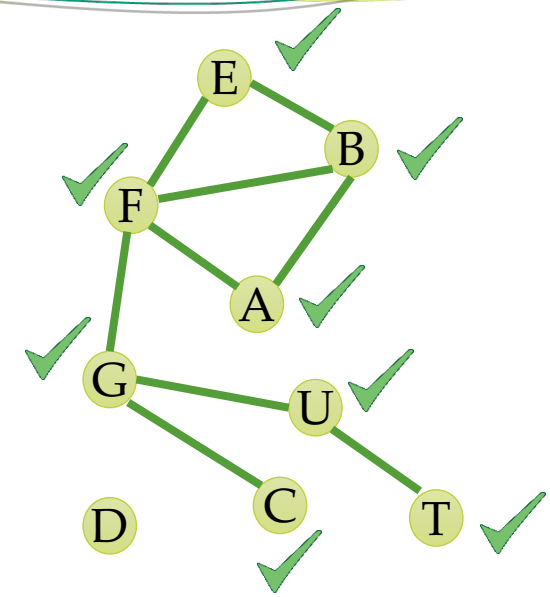
E
B
A
F
G
U
T
C

F
F

F no se procesa (se ignora) porque ya fue visitado

g.printDFS('E')

	0	1	2	3	4	5	6	7	8
E	F	T	F	T	F	F	F	F	F
B	T	F	T	T	F	F	F	F	F
A	F	T	F	T	F	F	F	F	F
F	T	T	T	F	F	T	F	F	F
D	F	F	F	F	F	F	F	F	F
G	F	F	F	T	F	F	T	F	T
U	F	F	F	F	F	T	F	T	F
T	F	F	F	F	F	F	T	F	F
C	F	F	F	F	F	T	F	F	F



E	B	A	F	D	G	U	T	C

E
B
A
F
G
U
T
C

printDFS(V) clásico. Tiene que estar OK para todo tipo de grafo.

Respecto a la implementación. ¿Qué estructura auxiliar usa????

Rta: Stack. (quizás por recursión)

Y además (es un grafo!!!) precisa marcar los vértices por los que pasó:

Opción 1: estructura paralela a los vértices booleana

Opción 2: representar el vértice con un tag booleano

Escribamos el código Java para lista de adyacencia.

Agregamos en la interface.

public void printDFS(V source);

TP 6 – Ejer 2.3

Implementar version iterable de PrintBFS

Pasemos a versión Iterable para usar el **foreach**

Caso de Uso

```
System.out.println("Recorrido BFS desde C con iterador");
for (Character aVertex : g.getBFS('C')) {
    System.out.println(aVertex);
}
```

Pasemos a versión Iterable para usar el **foreach**

Observación:

Iterable e Iterator están relacionadas!

```
public Iterable<V> getBFS(V startNode) {
    if (startNode == null || !existsVertex(startNode) )
        throw new IllegalArgumentException(Messages.getString("vertexParamError"));


    return new Iterable<V>() {
        @Override
        public Iterator<V> iterator() {
            return new myIteratorBFS(startNode);
        }
    };
}
```

```java
@Override
public void printBFS(V startNode) {
    if (startNode == null || !existsVertex(startNode) )
        throw new IllegalArgumentException(Messages.getString("vertexParamError"));

    Set<V> visited= new HashSet<V>();

    Queue<V> theQueue = new LinkedList<>();
    theQueue.add(startNode);

    while(! theQueue.isEmpty()) {
        V current = theQueue.poll();

        if ( visited.contains(current) )
            continue;

        visited.add(current);
        System.out.println(current);

        Collection<InternalEdge> adjListOther = getAdjacencyList().get(current);
        for (InternalEdge internalEdge : adjListOther) {
            if (! visited.contains(internalEdge.target)) {
                theQueue.add(internalEdge.target);
            }
        }
    }

}
```

```
Set<V> visited= new HashSet<V>();
Queue<V> theQueue = new LinkedList<>();

myIteratorBFS(V startNode) {
    theQueue.add(startNode);
}
```

```
@Override
public void printBFS(V startNode) {
    if (startNode == null || !existsVertex(startNode) )
        throw new IllegalArgumentException(Messages.getString("vertexParamE       ));

    Set<V> visited= new HashSet<V>();

    Queue<V> theQueue = new LinkedList<>();
    theQueue.add(startNode);

    while(! theQueue.isEmpty()) {
        V current = theQueue.poll();

        if ( visited.contains(current) )
            continue;

        visited.add(current);
        System.out.println(current);

        Collection<InternalEdge> adjListOther = getAdjacencyList().get(current);
        for (InternalEdge internalEdge : adjListOther) {
            if (! visited.contains(internalEdge.target)) {
                theQueue.add(internalEdge.target);
            }
        }
    }

}
```

```java
@Override
public boolean hasNext() {
    while ( ! theQueue.isEmpty() ) {
        V current = theQueue.peek();  // NO LO SACO!!!

        if ( ! visited.contains(current) )
            return true;

        // sino, buscar otro
        theQueue.poll();   // descartado
    }

    return false;  // no hubo manera
}
```

```java
@Override
public void printBFS(V startNode) {
    if (startNode == null || !exists
        throw new IllegalArgumentExc

    Set<V> visited= new HashSet<V>();

    Queue<V> theQueue = new LinkedList<>();
    theQueue.add(startNode);

    while(! theQueue.isEmpty()) {
        V current = theQueue.poll();

        if ( visited.contains(current) )
            continue;

        visited.add(current);
        System.out.println(current);

        Collection<InternalEdge> adjListOther = getAdjacencyList().get(current);
        for (InternalEdge internalEdge : adjListOther) {
            if (! visited.contains(internalEdge.target)) {
                theQueue.add(internalEdge.target);
            }
        }
    }

}
```

```java
@Override
public V next() {
    V current= theQueue.poll();
    visited.add(current);

    Collection<InternalEdge> adjListOther = getAdjacencyList().get(current);
    for (InternalEdge internalEdge : adjListOther) {
        if (! visited.contains(internalEdge.target)) {
            theQueue.add(internalEdge.target);
        }
    }

    return current;
}
}
```

```java
@Override
public void printBFS(V
    if (startNode == nu
        throw new Illeg

    Set<V> visited= new

    Queue<V> theQueue = new LinkedList<>();
    theQueue.add(startNode);

    while(! theQueue.isEmpty()) {
        V current = theQueue.poll();

        if ( visited.contains(current) )
            continue;

        visited.add(current);
        System.out.println(current);

        Collection<InternalEdge> adjListOther = getAdjacencyList().get(current);
        for (InternalEdge internalEdge : adjListOther) {
            if (! visited.contains(internalEdge.target)) {
                theQueue.add(internalEdge.target);
            }
        }
    }

}
```

TP 6 – Ejer 2.4

Implementar version iterable de PrintDFS (no recursiva)

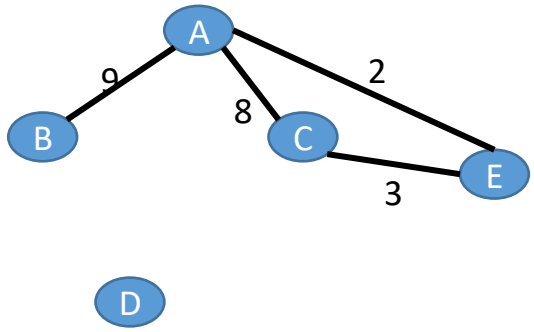
(con esto correcto podremos hacer la version iteradora de DFS)

TP 6 – Ejer 2.5

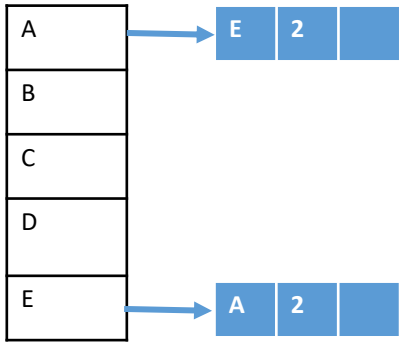
Implementar version iterable de PrintDFS

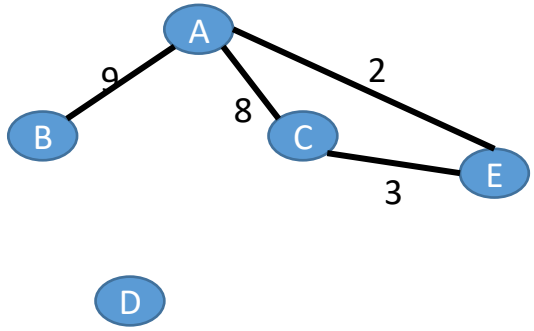
(usar lo anterior)

```
GraphService<Character,WeightedEdge> g =
GraphFactory.create(
Multiplicity.SIMPLE,
    EdgeMode.UNDIRECTED,
    SelfLoop.NO,
    Weight.YES,
    Storage.SPARSE);

g.addEdge('A', 'E', new WeightedEdge(2));
g.addEdge('A', 'C', new WeightedEdge(8));
g.addEdge('A', 'B', new WeightedEdge(9));
g.addEdge('C', 'E', new WeightedEdge(3));
g.addVertex('D');
```



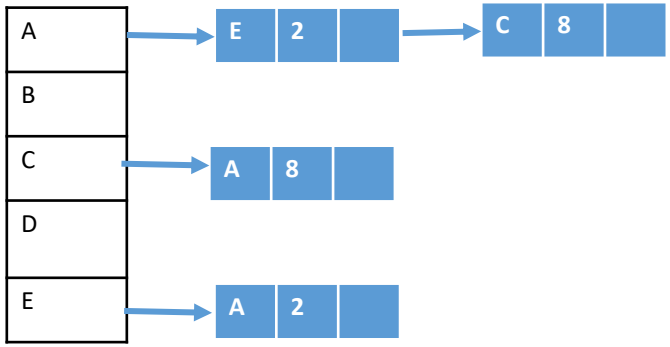
```
GraphService<Character,WeightedEdge> g =
GraphFactory.create(
Multiplicity.SIMPLE,
    EdgeMode.UNDIRECTED,
    SelfLoop.NO,
    Weight.YES,
    Storage.SPARSE);
```



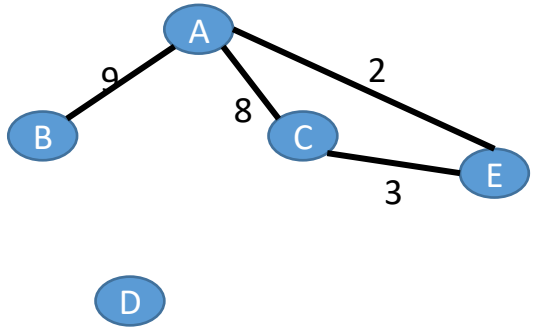



```
g.addEdge('A', 'E', new WeightedEdge(2));
g.addEdge('A', 'C', new WeightedEdge(8));
g.addEdge('A', 'B', new WeightedEdge(9));
g.addEdge('C', 'E', new WeightedEdge(3));
g.addVertex('D');
```

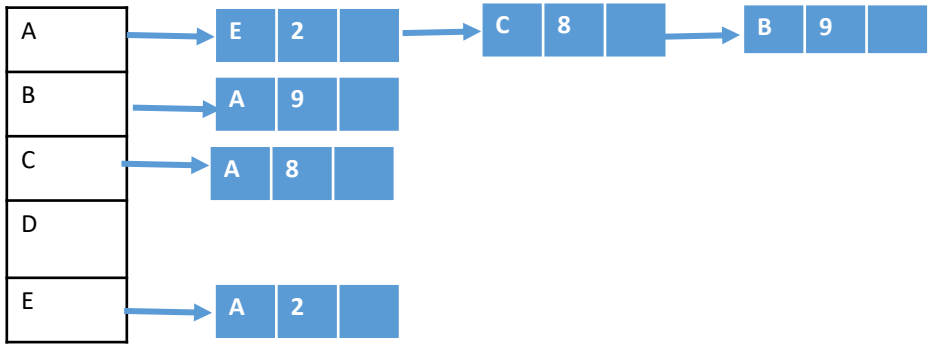
```
GraphService<Character,WeightedEdge> g =
GraphFactory.create(
Multiplicity.SIMPLE,
    EdgeMode.UNDIRECTED,
    SelfLoop.NO,
    Weight.YES,
    Storage.SPARSE);
```



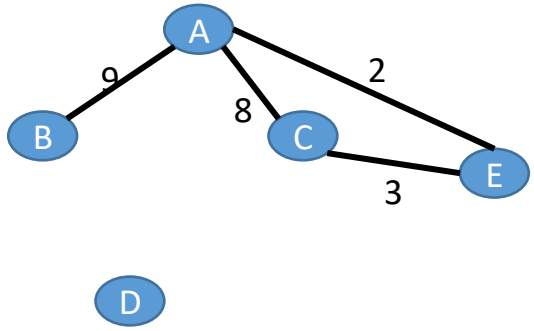

```
g.addEdge('A', 'E', new WeightedEdge(2));
g.addEdge('A', 'C', new WeightedEdge(8));
g.addEdge('A', 'B', new WeightedEdge(9));
g.addEdge('C', 'E', new WeightedEdge(3));
g.addVertex('D');
```



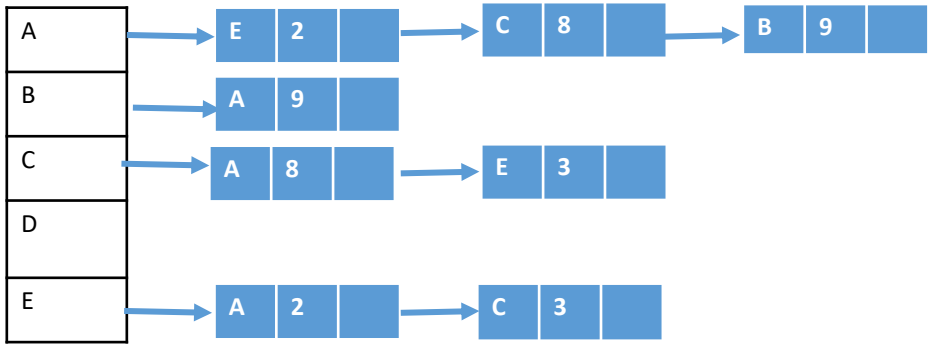
```
GraphService<Character,WeightedEdge> g =
GraphFactory.create(
Multiplicity.SIMPLE,
    EdgeMode.UNDIRECTED,
    SelfLoop.NO,
    Weight.YES,
    Storage.SPARSE);
```

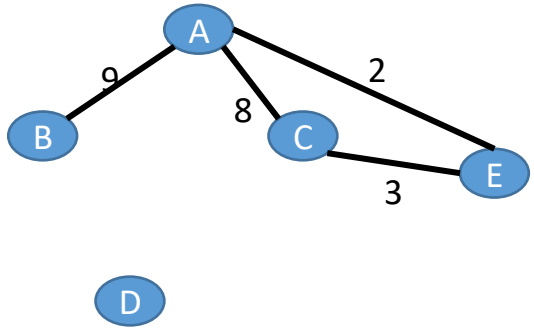



```
g.addEdge('A', 'E', new WeightedEdge(2));
g.addEdge('A', 'C', new WeightedEdge(8));
g.addEdge('A', 'B', new WeightedEdge(9));
g.addEdge('C', 'E', new WeightedEdge(3));
g.addVertex('D');
```



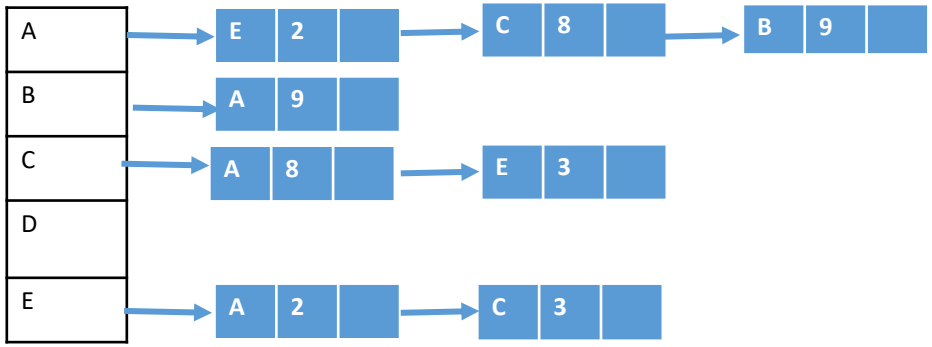
```
GraphService<Character,WeightedEdge> g =
GraphFactory.create(
Multiplicity.SIMPLE,
    EdgeMode.UNDIRECTED,
    SelfLoop.NO,
    Weight.YES,
    Storage.SPARSE);
```



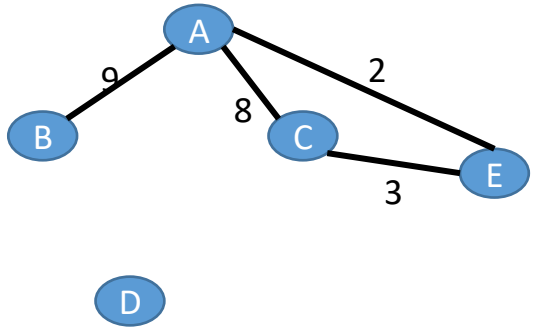



```
g.addEdge('A', 'E', new WeightedEdge(2));
g.addEdge('A', 'C', new WeightedEdge(8));
g.addEdge('A', 'B', new WeightedEdge(9));
g.addEdge('C', 'E', new WeightedEdge(3));
g.addVertex('D');
```

```
GraphService<Character,WeightedEdge> g =
GraphFactory.create(
Multiplicity.SIMPLE,
    EdgeMode.UNDIRECTED,
    SelfLoop.NO,
    Weight.YES,
    Storage.SPARSE);
```

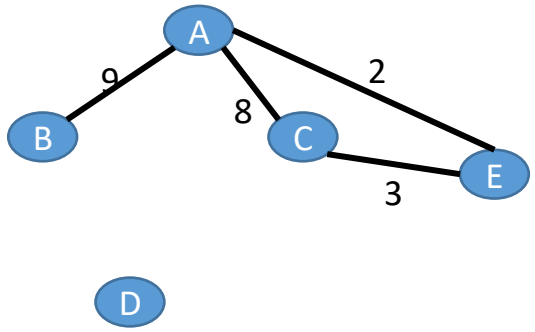



```
g.addEdge('A', 'E', new WeightedEdge(2));
g.addEdge('A', 'C', new WeightedEdge(8));
g.addEdge('A', 'B', new WeightedEdge(9));
g.addEdge('C', 'E', new WeightedEdge(3));
g.addVertex('D');
```



```
GraphService<Character,WeightedEdge> g =
GraphFactory.create(
Multiplicity.SIMPLE,
    EdgeMode.UNDIRECTED,
    SelfLoop.NO,
    Weight.YES,
    Storage.DENSE);
```

	A	B	C	D	E
A		9	8		2
B	9				
C	8				3
D					
E	2		3		



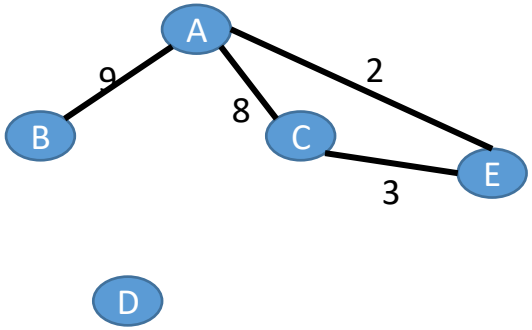

```
g.addEdge('A', 'E', new WeightedEdge(2));
g.addEdge('A', 'C', new WeightedEdge(8));
g.addEdge('A', 'B', new WeightedEdge(9));
g.addEdge('C', 'E', new WeightedEdge(3));
g.addVertex('D');
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	Inf	Inf	0	Inf	inf

Visited = { }

costosConocidos= { **(C,0)** }



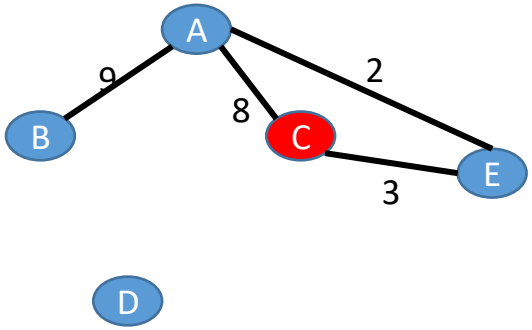

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	Inf	Inf	0	Inf	inf

Visited = { **C** }

costosConocidos= { }



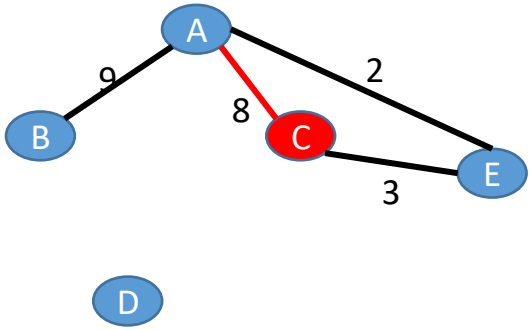

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	8	Inf	0	Inf	inf

Visited = { **C** }

costosConocidos= { (A, 8) }



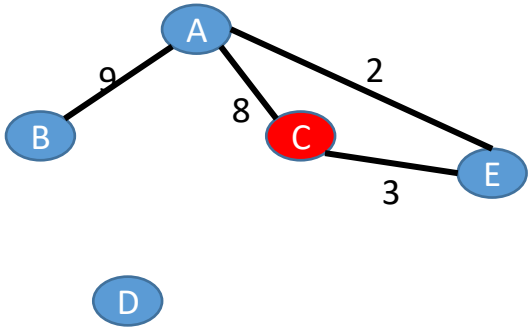

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	8	Inf	0	Inf	inf

Visited = { **C** }

costosConocidos= { (A, 8) }



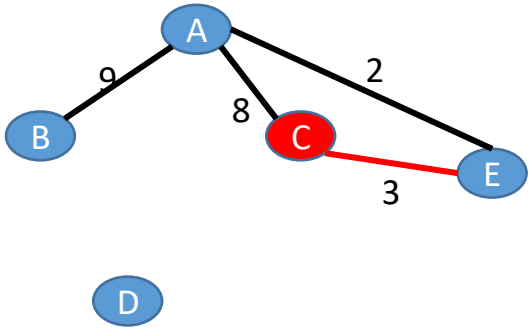

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	8	Inf	0	Inf	3

Visited = { **C** }

costosConocidos= { (E, 3), (A, 8) }



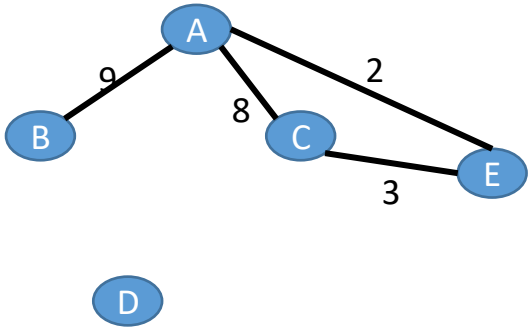

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	8	Inf	0	Inf	3

Visited = { **C** }

costosConocidos= { (E, 3), (A, 8) }



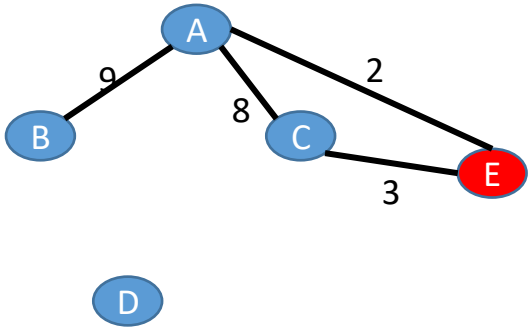

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	8	Inf	0	Inf	3

Visited = { **C, E** }

costosConocidos= { (A, 8) }



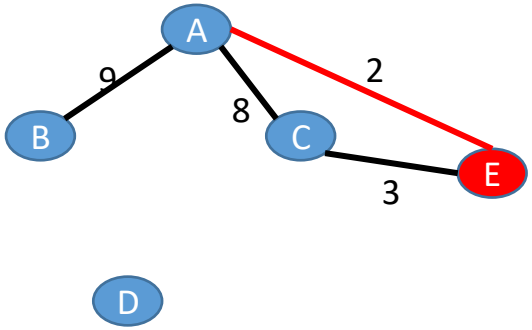

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	5	Inf	0	Inf	3

Visited = { **C, E** }

costosConocidos= { (A, 5), (A, 8) }



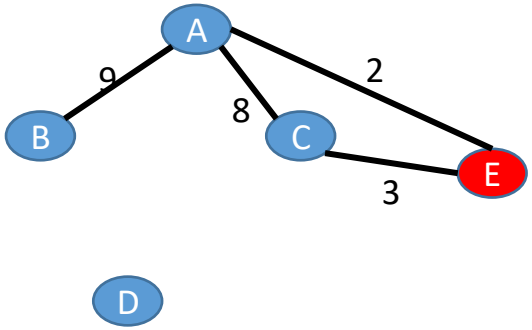

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	5	Inf	0	Inf	3

Visited = { **C, E** }

costosConocidos= { (A, 5), (A, 8) }



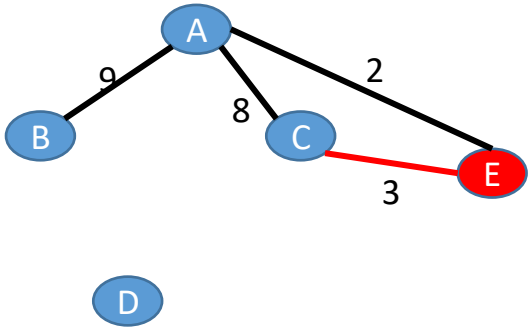

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	5	Inf	0	Inf	3

Visited = { **C, E** }

costosConocidos= { (A, 5), (A, 8) }



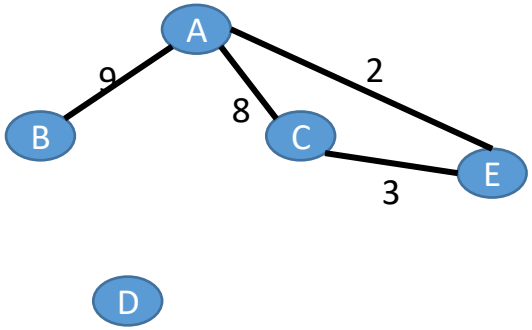

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	5	Inf	0	Inf	3

Visited = { **C, E** }

costosConocidos= { (A, 5), (A, 8) }



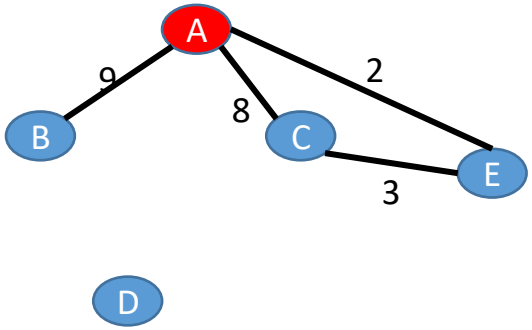

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	5	Inf	0	Inf	3

Visited = { **C, E, A** }

costosConocidos= { (A, 8) }



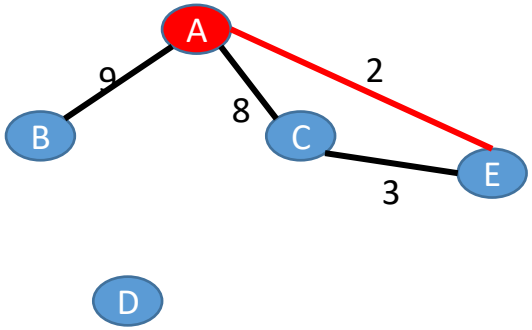

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	5	Inf	0	Inf	3

Visited = { **C, E, A** }

costosConocidos= { (A, 8) }



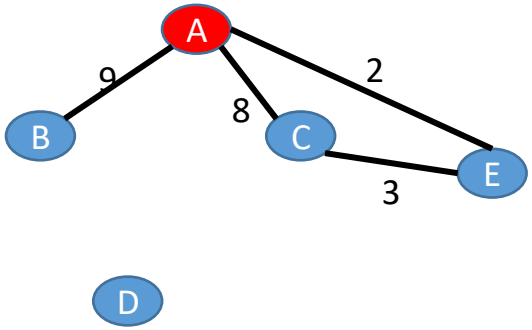

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	5	Inf	0	Inf	3

Visited = { **C, E, A** }

costosConocidos= { (A, 8) }



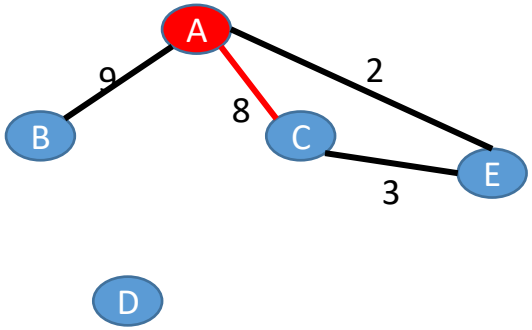

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	5	Inf	0	Inf	3

Visited = { **C, E, A** }

costosConocidos= { (A, 8) }



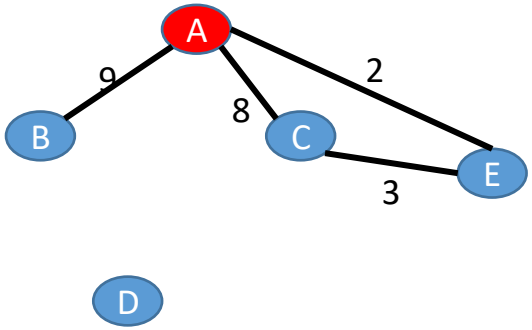

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	5	Inf	0	Inf	3

Visited = { **C, E, A** }

costosConocidos= { (A, 8) }



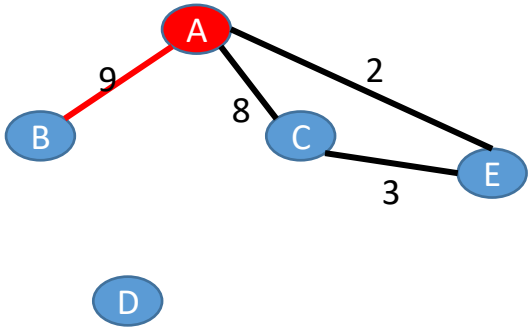

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	5	14	0	Inf	3

Visited = { **C, E, A** }

costosConocidos= { (A, 8), (B, 14) }



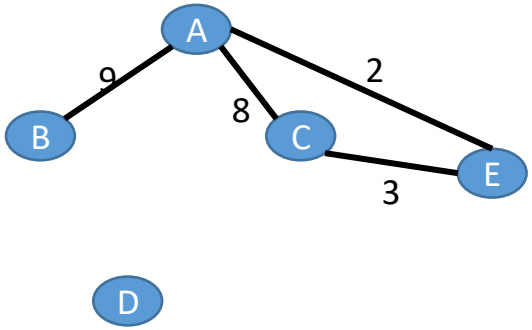

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	5	14	0	Inf	3

Visited = { **C, E, A** }

costosConocidos= { (A, 8), (B, 14) }



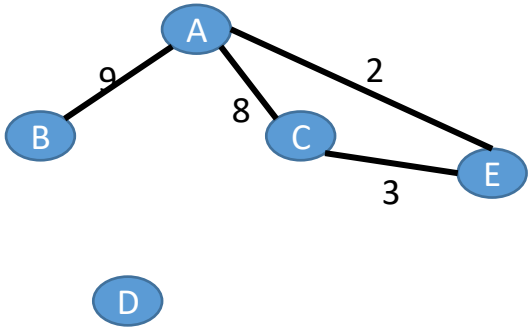

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	5	14	0	Inf	3

Visited = { **C, E, A** }

costosConocidos= { (B, 14) }



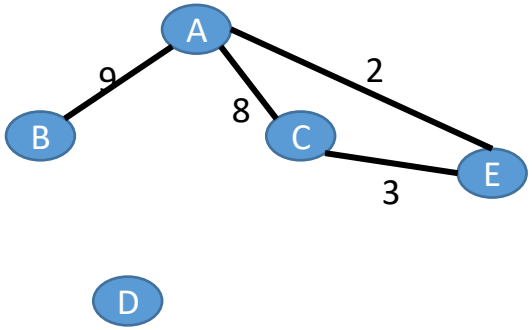

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	5	14	0	Inf	3

Visited = { **C, E, A** }

costosConocidos= { (B, 14) }



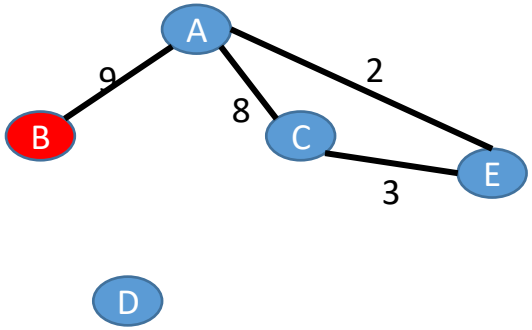

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	5	14	0	Inf	3

Visited = { **C, E, A** }

costosConocidos= { }



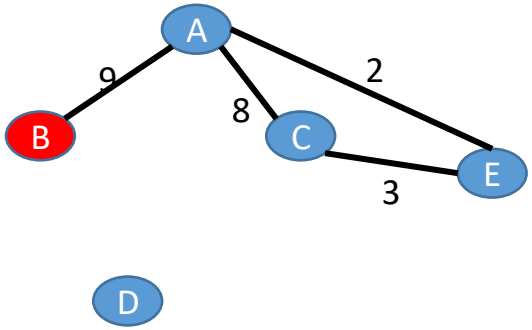

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	5	14	0	Inf	3

Visited = { **C, E, A, B** }

costosConocidos= {}



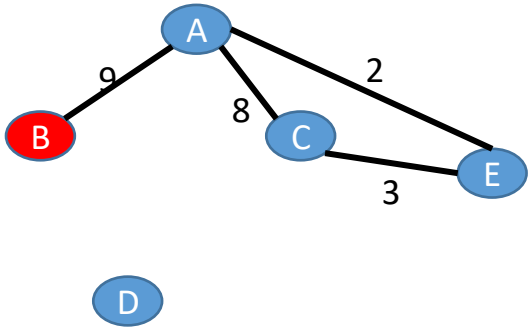

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	5	14	0	Inf	3

Visited = { **C, E, A, B** }

costosConocidos= {}



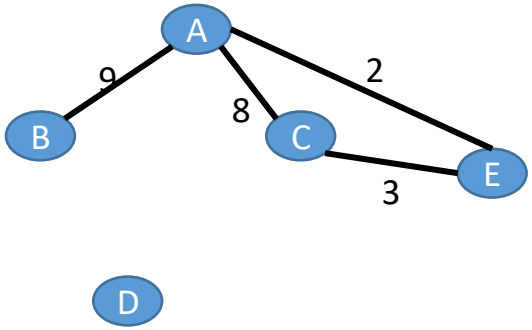

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Dijkstra de C a los demás?

	A	B	C	D	E
Costo?	5	14	0	Inf	3

Visited = { **C, E, A, B** }

costosConocidos= {}

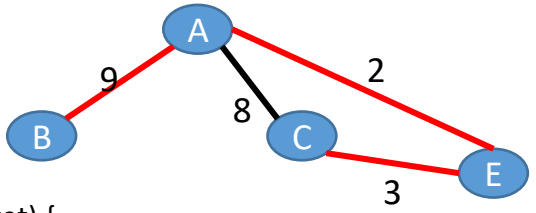



```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```

Y si quiero el camino???

Cuando actualizo por un costo menor, podría indicar quien es su nuevo “previo”

```
…
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                              saltear;
                else
                              si (suma costo de current + e.weight  <  costo e.target) {
                                            actualizar el costo de e.target;
                                            agregarloCostosConocidos(e.target, new Co
                                            prev.put(  e.target,   current.vertex);    // hashing
                              }
}
```



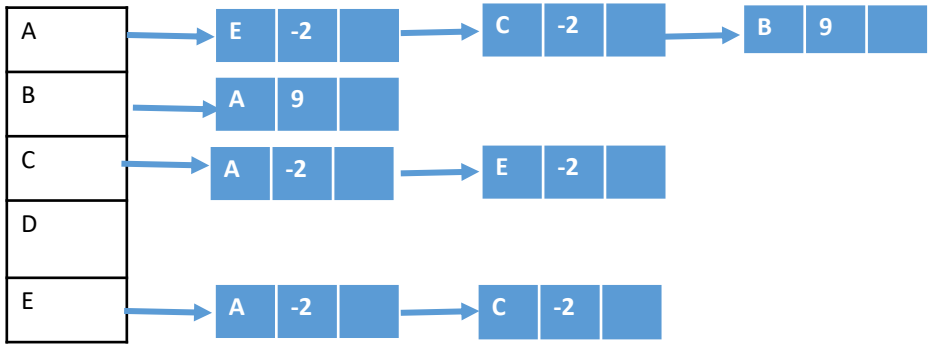
	A	B	C	D	E
Costo?	5	14	0	Inf	3
Previo	E	A	Null	Null	C

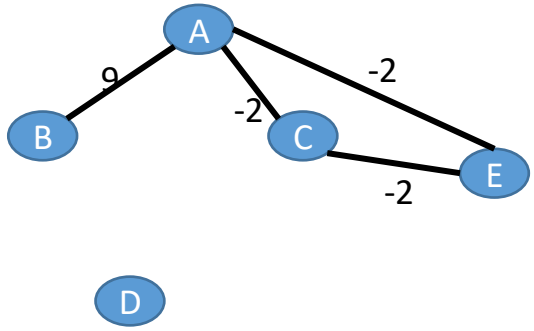
```
5: [C, E, A]
14: [C, E, A, B]
0: [C]
INF: []
3: [C, E]
```

Dijkstra tiene una precondición: los ejes no puede tener peso negativo

¿Por qué?

```
GraphService<Character,WeightedEdge> g =
GraphFactory.create(
Multiplicity.SIMPLE,
    EdgeMode.UNDIRECTED,
    SelfLoop.NO,
    Weight.YES,
    Storage.SPARSE);
```






```
g.addEdge('A', 'E', new WeightedEdge(-2));
g.addEdge('A', 'C', new WeightedEdge(-2));
g.addEdge('A', 'B', new WeightedEdge(9));
g.addEdge('C', 'E', new WeightedEdge(-2));
g.addVertex('D');
```

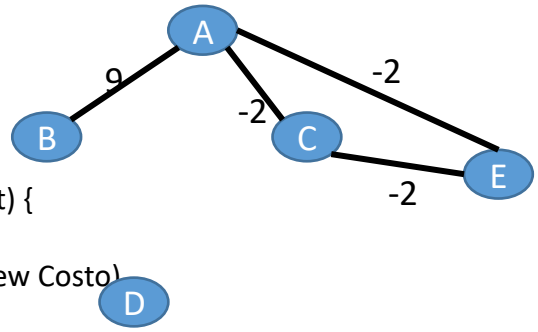
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	Inf	Inf	Inf	Inf	0

Visited = { }

costosConocidos= { **(E,0)** }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



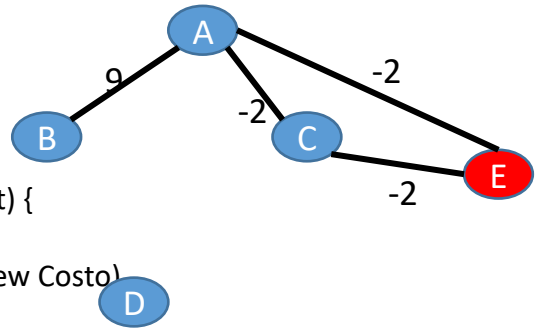
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	Inf	Inf	Inf	Inf	0

Visited = { **E** }

costosConocidos= {}

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



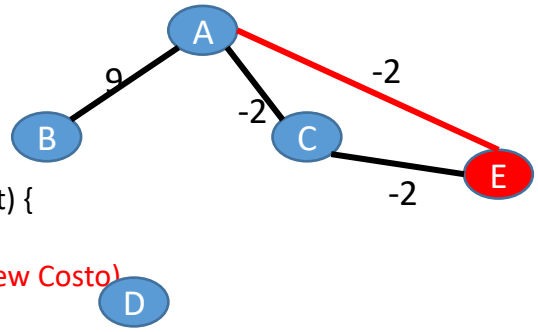
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	Inf	Inf	Inf	0

Visited = { **E** }

costosConocidos= { (A, -2) }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



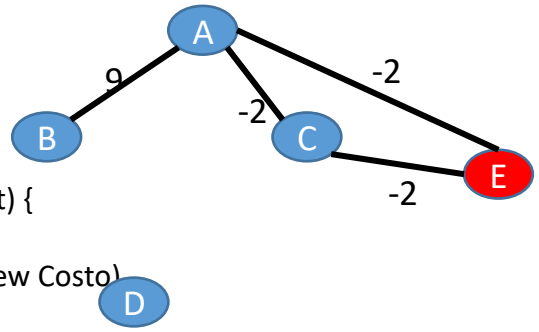
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	Inf	Inf	Inf	0

Visited = { **E** }

costosConocidos= { (A, -2) }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



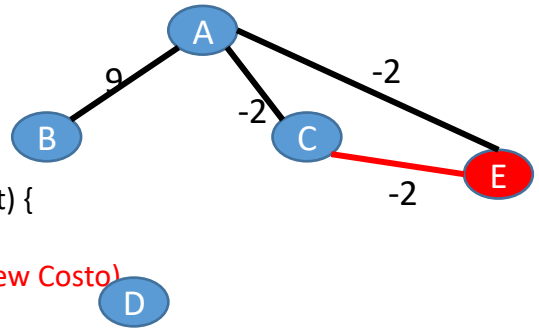
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	Inf	-2	Inf	0

Visited = { **E** }

costosConocidos= { (A, -2), (C, -2) }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



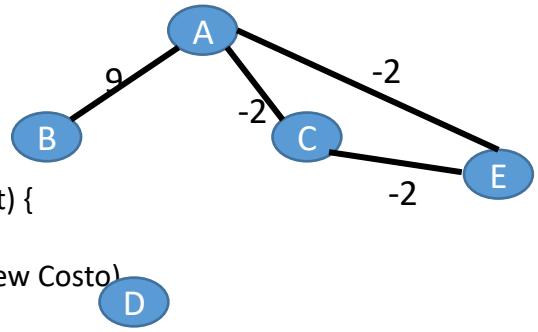
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	Inf	-2	Inf	0

Visited = { **E** }

costosConocidos= { (A, -2), (C, -2) }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



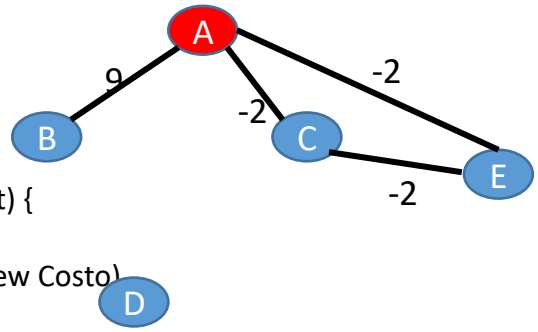
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	Inf	-2	Inf	0

Visited = { **E, A** }

costosConocidos= { (C, -2) }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



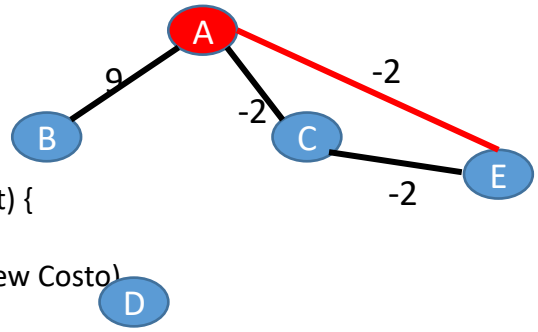
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	Inf	-2	Inf	0

Visited = { **E, A** }

costosConocidos= { (C, -2) }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



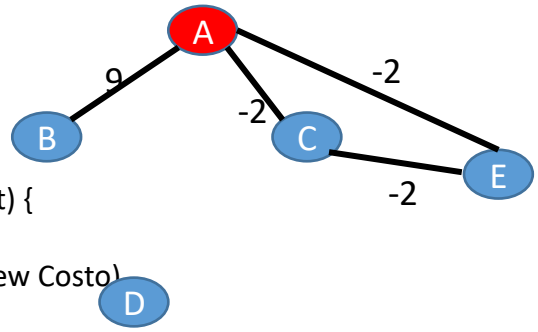
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	Inf	-2	Inf	0

Visited = { **E, A** }

costosConocidos= { (C, -2) }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



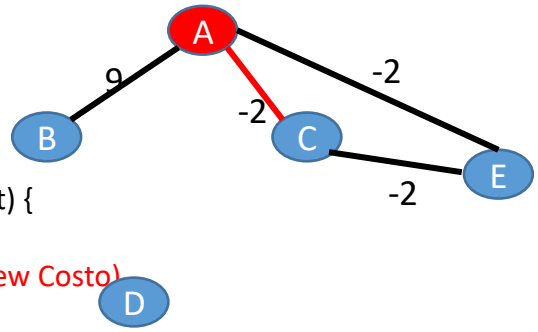
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	Inf	-4	Inf	0

Visited = { **E, A** }

costosConocidos= { (C,-4) , (C, -2) }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



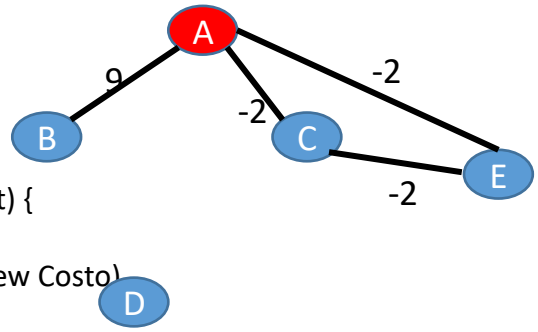
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	Inf	-4	Inf	0

Visited = { **E, A** }

costosConocidos= { (C, -4) ,(C, -2) }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



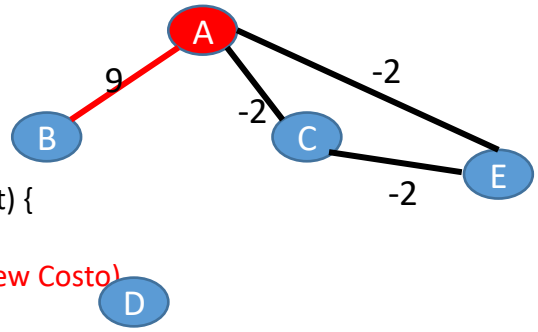
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	7	-4	Inf	0

Visited = { **E, A** }

costosConocidos= { (C, -4) ,(C, -2) (B,7) }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



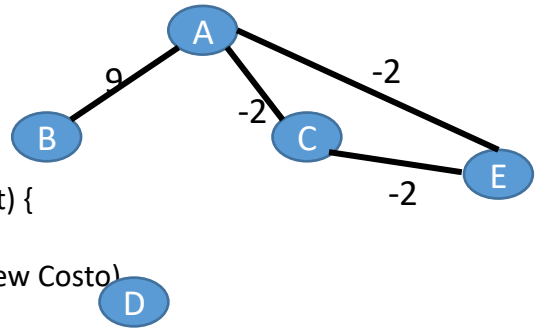
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	7	-4	Inf	0

Visited = { **E, A** }

costosConocidos= { (C, -4) ,(C, -2) (B,7) }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



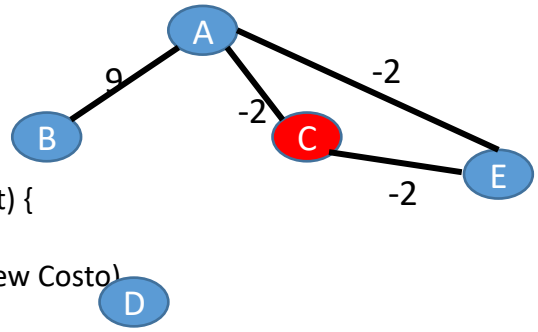
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	7	-4	Inf	0

Visited = { **E, A, C** }

costosConocidos= {(C, -2) (B,7) }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



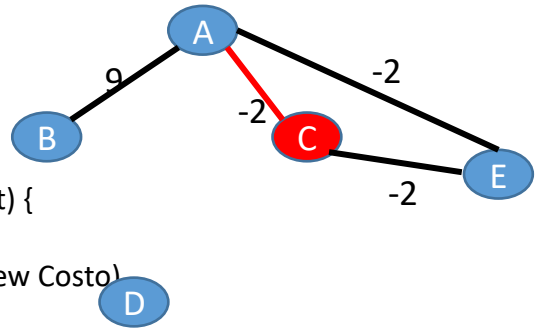
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	7	-4	Inf	0

Visited = { **E, A, C** }

costosConocidos= { (C, -2) (B,7) }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



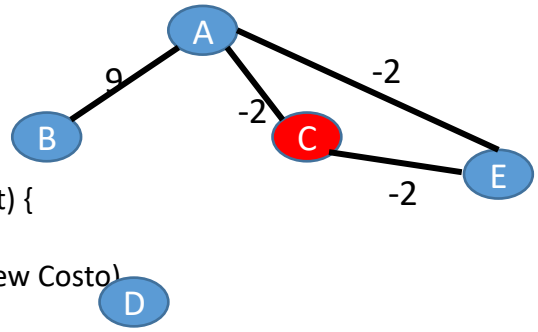
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	7	-4	Inf	0

Visited = { **E, A, C** }

costosConocidos= {(C, -2) (B,7) }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



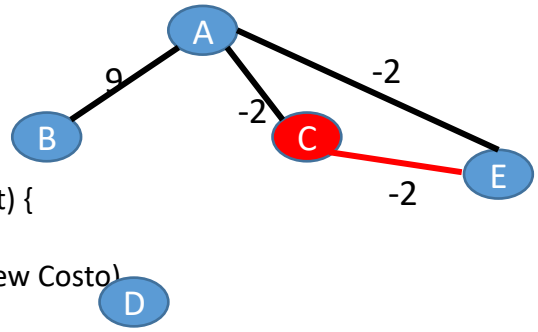
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	7	-4	Inf	0

Visited = { **E, A, C** }

costosConocidos= { (C, -2) (B,7) }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



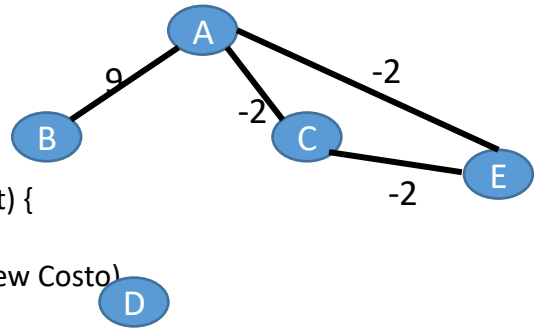
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	7	-4	Inf	0

Visited = { **E, A, C** }

costosConocidos= { (C, -2) (B,7) }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



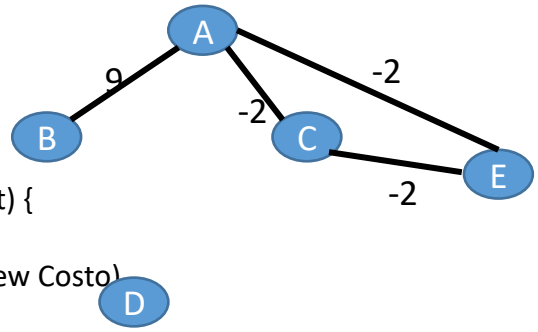
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	7	-4	Inf	0

Visited = { **E, A, C** }

costosConocidos= { (C, -2) (B,7) }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



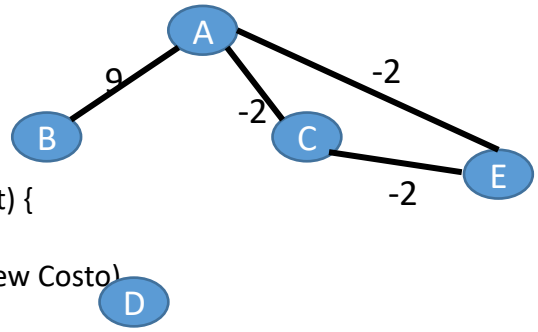
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	7	-4	Inf	0

Visited = { **E, A, C** }

costosConocidos= { (B,7) }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



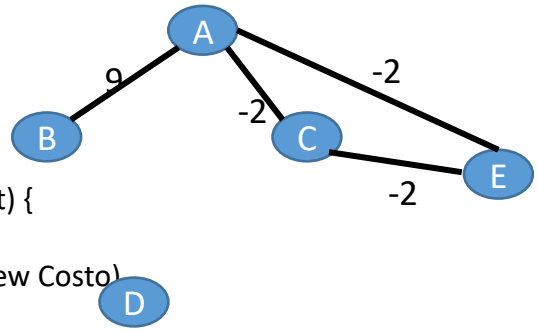
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	7	-4	Inf	0

Visited = { **E, A, C** }

costosConocidos= { (B,7) }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



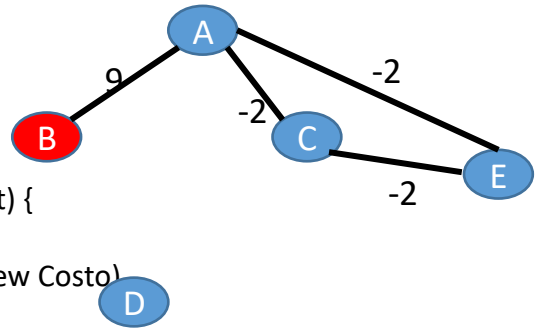
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	7	-4	Inf	0

Visited = { **E, A, C, B** }

costosConocidos= { }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



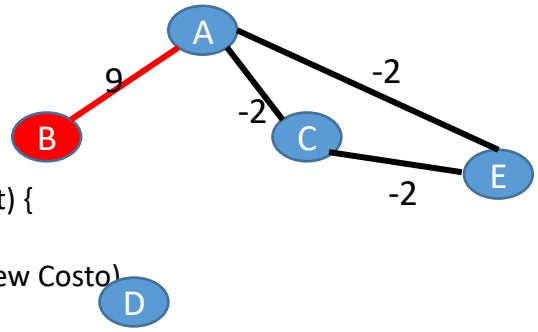
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	7	-4	Inf	0

Visited = { **E, A, C, B** }

costosConocidos= { }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                                saltear;
                else
                                si (suma costo de current + e.weight  <  costo e.target) {
                                                actualizar el costo de e.target;
                                                agregarloCostosConocidos(e.target, new Costo)
                                }
}
```



DISPARATE!!!!

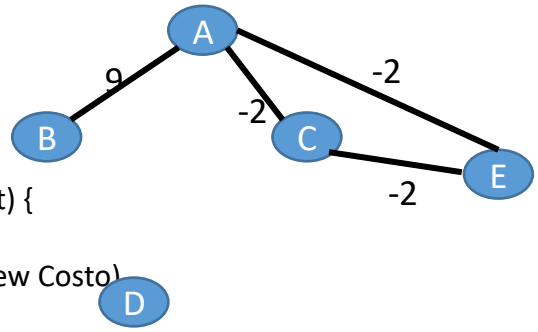
Dijkstra de E a los demás?

	A	B	C	D	E
Costo?	-2	7	-4	Inf	0

Visited = { **E, A, C, B** }

costosConocidos= { }

```
While (! costosConocidos.isEmpty() ) {
   current= Sacar el de menor costo de  costosConocidos.
  if ( Visited.contains( current.vertex)
    continue;

  Visisted.add( current.vertex);
  Foreach (    e   in  ejes incidentes de current )
                if ( e.target ya estaba en visitado)
                              saltear;
                else
                              si (suma costo de current + e.weight  <  costo e.target) {
                                            actualizar el costo de e.target;
                                            agregarloCostosConocidos(e.target, new Costo)
                              }
}
```



Otra forma de implementarlo correctamente (hay varias) consiste en:

En vez de agregar nodos posiblemente repetidos a la estructura:

si el costo mejora **agregoOActualizo** (si el elemento no estaba en la estructura lo **agrego**, sino **saco a ese y agrego**). Así, la estructura nunca tiene mas de |V| elementos.

Si hiciéramos eso, para calcular complejidad tenemos:

Times

=

$$\sum_{u \in V} ((Times(sacarMin) + \sum_{v \, vecino \, de \, u} Times(agregoOactualizo))$$

=

$$\sum_{u \in V} Times(sacarmin) + \sum_{u \in V} \sum_{v \, vecino \, de \, u} Times(agregoOactualizo)$$

= |V| * Times(sacarMin) + |E| * Times(agregoOActualizo)

Como Times= |V| * Times(sacarMin) + |E| * Times(agregoOActualizo)

a) Si como estructura elegimos un AVL o RedBlackTree cuanto es O?

Rta O (|V| * log2|V| + |E|* c * log2|V|)

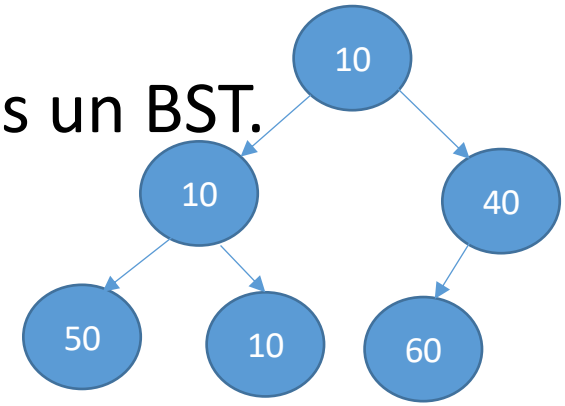
o sea O ((|V| + |E|) * log2 |V|)

b) Hay otra estructura que se usa mucho y java tiene implementada: PriorityQueue que implementa un Binary Heap

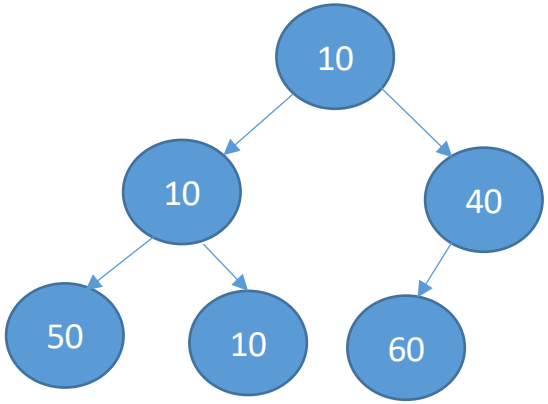
Un Binary Heap es un BT completo (se acuerdan?) tal que cada nodo es menor o igual que todos los elementos de sus subárboles.

Claramente no es un BST.

Ej:

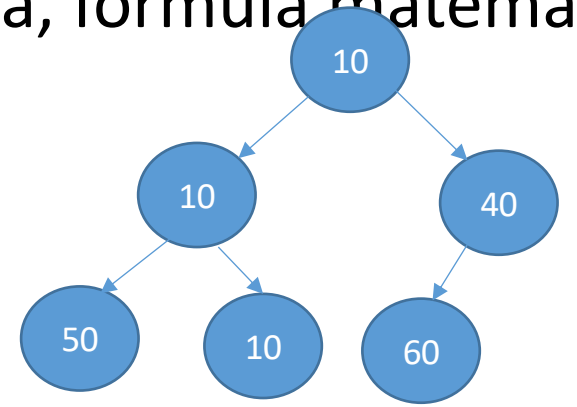


Se pueden representar muy eficientemente con arreglos (si me pongo a cubierto para no quedarme corto con el espacio). El arreglo surge del recorrido “por niveles” del Binary Heap



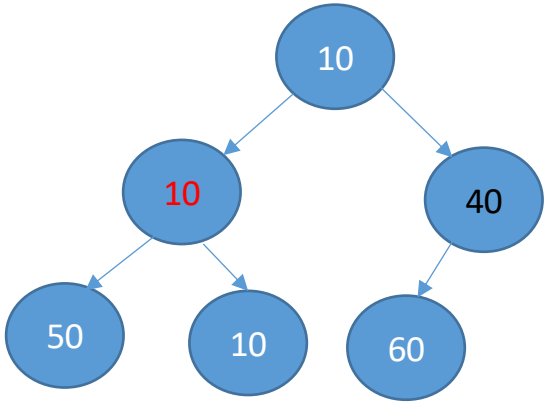
10	10	40	50	10	60		
0	1	2	3	4	5	6	7

Cada elemento en el arreglo tiene la siguiente característica de indización: para pos=i su antecesor está en pos $\lfloor (i-1) / 2 \rfloor$, su hijo izq está en pos 2*i+1 y su hijo derecho está en pos 2*i+1+1 (o sea, fórmula matemática!)

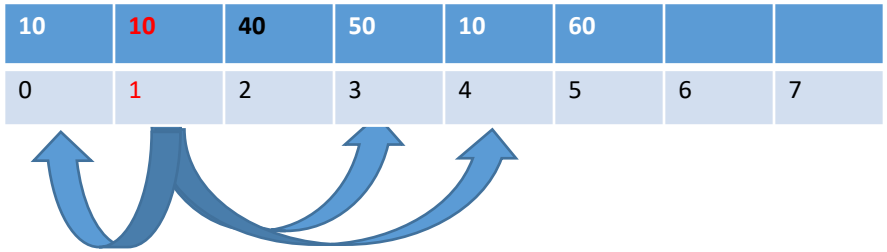


10	10	40	50	10	60		
0	1	2	3	4	5	6	7

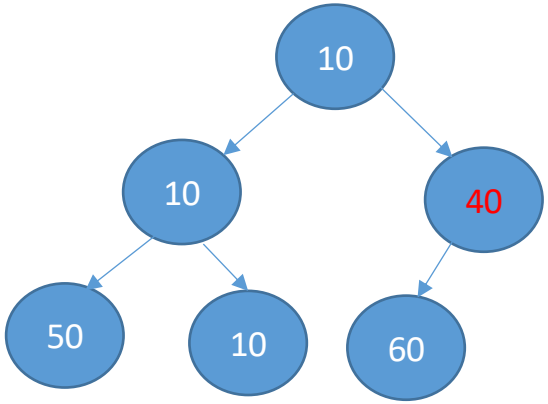
Pos=1



Antecesor $\lfloor (1-1) / 2 \rfloor$ => 0
Hijo izq 2*1 + 1 => 3
Hijo der 2*1 + 1 +1 => 4



Pos=2

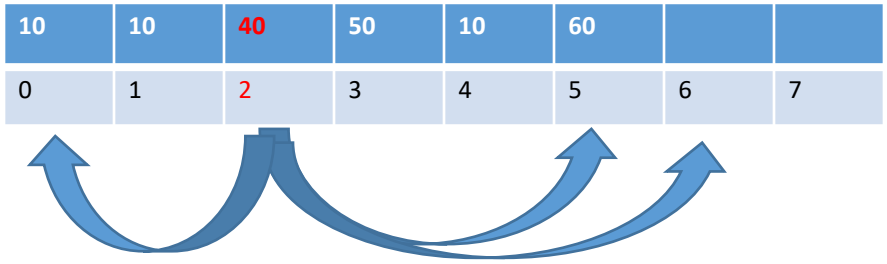


Antecesor $\lfloor (2-1) / 2 \rfloor$ => 0

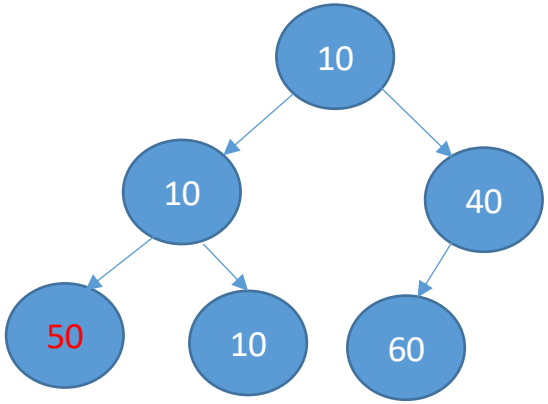
Hijo izq 2*2 + 1

=> 5

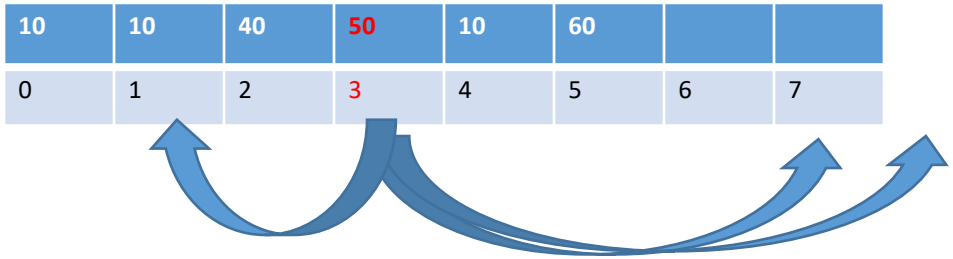
Hijo der 2*2 + 1 + 1 => 6



Pos=3



Antecesor $\lfloor (3-1) / 2 \rfloor$ => 1
Hijo izq 2*3 + 1 => 7 (cuando estén)
Hijo der 2*3 + 1 + 1 => 8 (cuando estén)



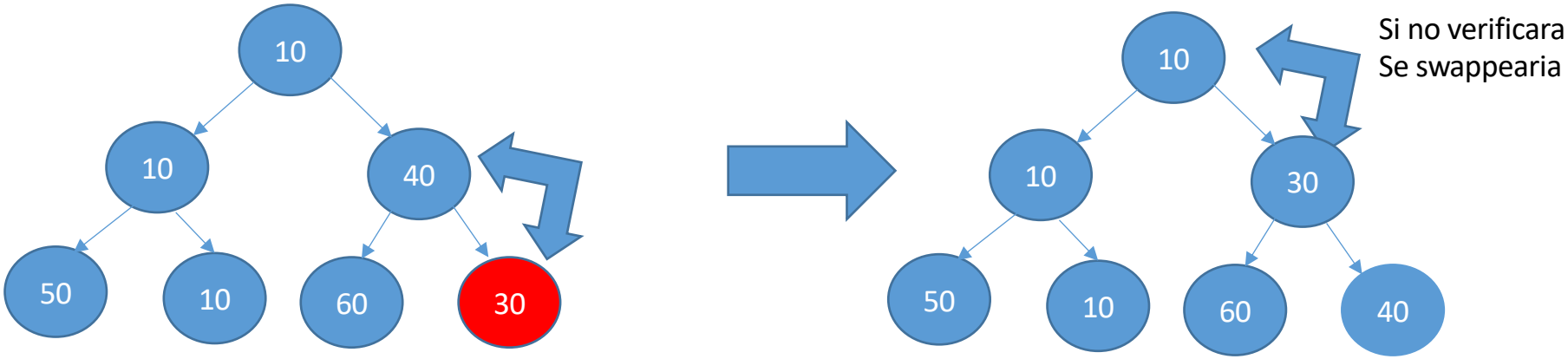
Que hay de las operaciones que precisamos?

Consultar Min => O(1)

Add => lo inserto al final (trivial) pero tengo que solo swappear para garantizar la propiedad.

SacarMin => lo reemplazo por ultimo y luego swappeo para garantizar la propiedad

Ej: inserto el 30. Como viola, empiezo a swappear (en el arreglo) con el antecesor (que se donde esta…) hasta que se verifique prop.



A lo sumo, cuantos swappeos hago?

Rta: la altura del árbol, o sea O(log n)

EL borrado del min es análogo: se reemplaza la raíz por la ultima hoja y se swappea hasta garantizar propiedad. Es O(log n)

Como Times= |V| * Times(sacarMin) + |E| * Times(agregoOActualizo)

a) Si como estructura elegimos un AVL o RedBlackTree cuanto es O?

Rta O (|V| * log2|V| + |E|* c * log2|V|)

o sea O ((|V| + |E|) * log2 |V|)

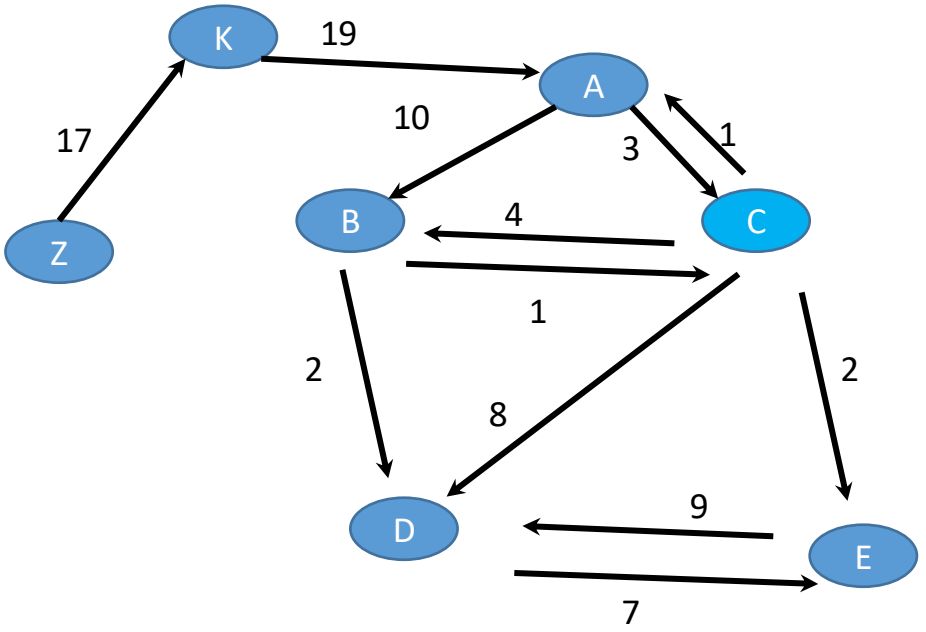
b) Para PriorityQueue la complejidad es:

Rta O (|V| * log2 | V| + |E| * c * log2|V|)

o sea O ((|V| + |E|) * log2 |V|)

Caso de Uso

```
GraphService<Character,WeightedEdge> g =  GraphFactory.create(Multiplicity.SIMPLE, EdgeMode.DIRECTED,  SelfLoop.NO,
Weight.YES,  Storage.SPARSE);

g.addEdge('A', 'B', new WeightedEdge(10));

g.addEdge('A', 'C', new WeightedEdge(3));

g.addEdge('B', 'C', new WeightedEdge(1));

g.addEdge('B', 'D', new WeightedEdge(2));

g.addEdge('C', 'A', new WeightedEdge(1));

g.addEdge('C', 'B', new WeightedEdge(4));

g.addEdge('C', 'D', new WeightedEdge(8));

g.addEdge('C', 'E', new WeightedEdge(2));

g.addEdge('D', 'E', new WeightedEdge(7));

g.addEdge('E', 'D', new WeightedEdge(9));

g.addEdge('Z', 'K', new WeightedEdge(17));

g.addEdge('K', 'A', new WeightedEdge(19));

DijkstraPath<Character, WeightedEdge> pathRta = g.dijsktra('A');

System.out.println(pathRta);
```




```
0: [A]
7: [A, C, B]
3: [A, C]
9: [A, C, B, D]
5: [A, C, E]
```

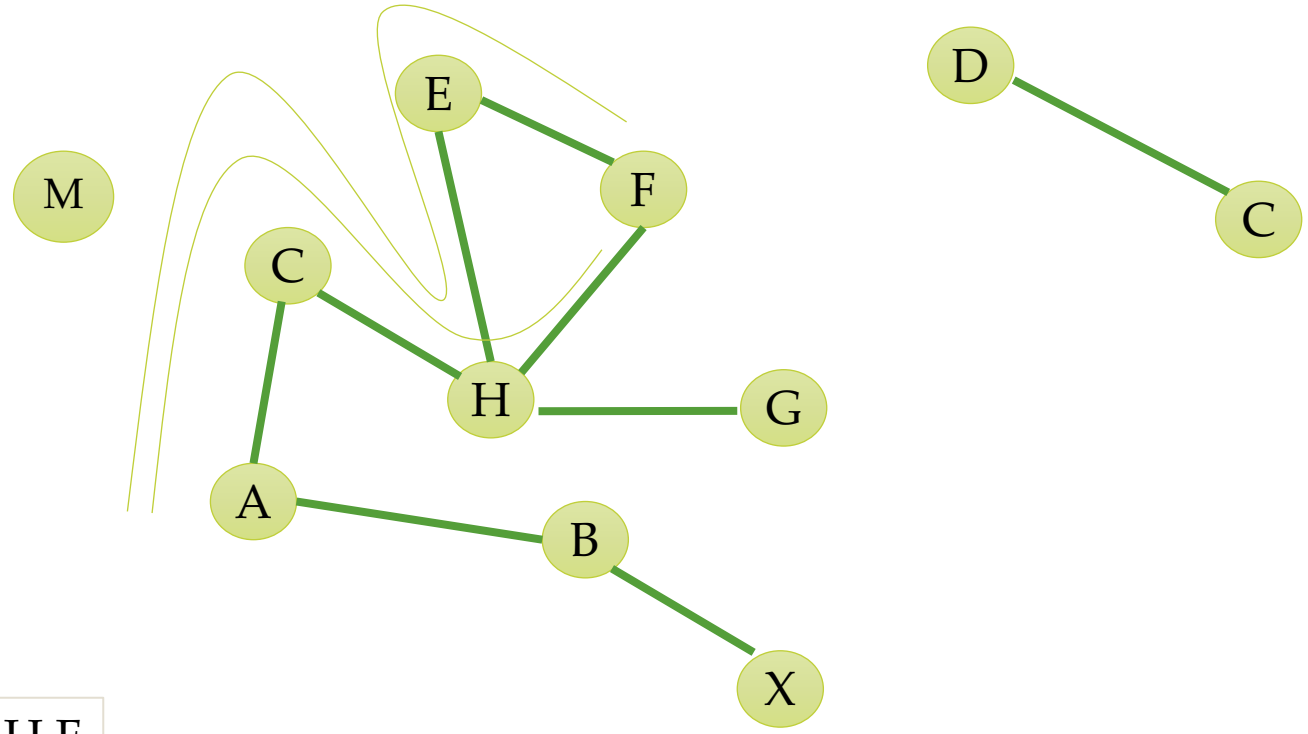
Ejercicio

Solo para grafo SimpleOrDefault, sin lazos

Agregar el siguiente método a la interface

void printAllPaths(V startNode, V endNode);

g.printAllPaths('A', 'F');



A C H F

A C H E F

```java
public void printAllPaths(V startNode, V endNode) {
    if (startNode == null || !existsVertex(startNode) || endNode == null || !existsVertex(endNode) )
        throw new IllegalArgumentException(Messages.getString("vertexParamError"));

    if (acceptSelfLoop)
        throw new IllegalArgumentException(Messages.getString("getAllPathsError"));

    Set<V> visited= new HashSet<V>();
    ArrayList<V> path= new ArrayList<>();

    printAllPaths(startNode, endNode, visited, path);

}
```

```java
public void printAllPaths(V startNode, V endNode, Set<V> visited, ArrayList<V> path ) {

    // proceso el nodo
    path.add(startNode);
    visited.add(startNode);

    if (startNode.equals(endNode)) {
        System.out.println(path);

        // deshago porque no voy a pasar por flujo normal
        visited.remove(endNode);
        path.remove(endNode);

        return;
    }

    Collection<InternalEdge> adjListOther = getAdjacencyList().get(startNode);
    for (InternalEdge internalEdge : adjListOther) {
        if (! visited.contains(internalEdge.target)) {
            printAllPaths(internalEdge.target, endNode, visited, path);
        }
    }

    // lo deshago
    visited.remove(startNode);
    path.remove(startNode);

}
```