

GraphS

Graphs Specification Language

Procesadores de Lenguajes

Curso 2008 – 2009

Escuela Superior de Informática
Universidad de Castilla-La Mancha

Jose Domingo López López
josed.lopez1@alu.uclm.es

Ángel Escribano Santamarina
angel.escribano1@alu.uclm.es

Tabla de contenidos

1.	Definición del problema.....	3
2.	Diseño del Procesador. Diagramas en forma de "T"	4
3.	Descripción del lenguaje GraphS.....	5
4.	Especificación EBNF del lenguaje GraphS	6
5.	Diagramas de Conway	7
6.	Definición de la semántica en lenguaje natural	10
	• DEFINICIÓN DE GRAFO	10
	• DECLARACIÓN DE NODOS	10
	• DECLARACIÓN DE ARISTAS	10
	• CONEXIÓN DE NODOS	11
	• DEFINICIÓN DE OPERACIONES	11
7.	Ejemplos	12
	• GRAFO NO DIRIGIDO Y NO VALORADO	12
	• GRAFO NO DIRIGIDO Y VALORADO	13
	• GRAFO DIRIGIDO Y VALORADO	14
	• UNIÓN DE GRAFOS	15
8.	Futuras investigaciones.....	16

1. Definición del problema

Se pretende diseñar una aplicación que permita al usuario especificar grafos mediante el lenguaje GraphS y realizar operaciones con dichos grafos.

Informalmente, un grafo es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto.

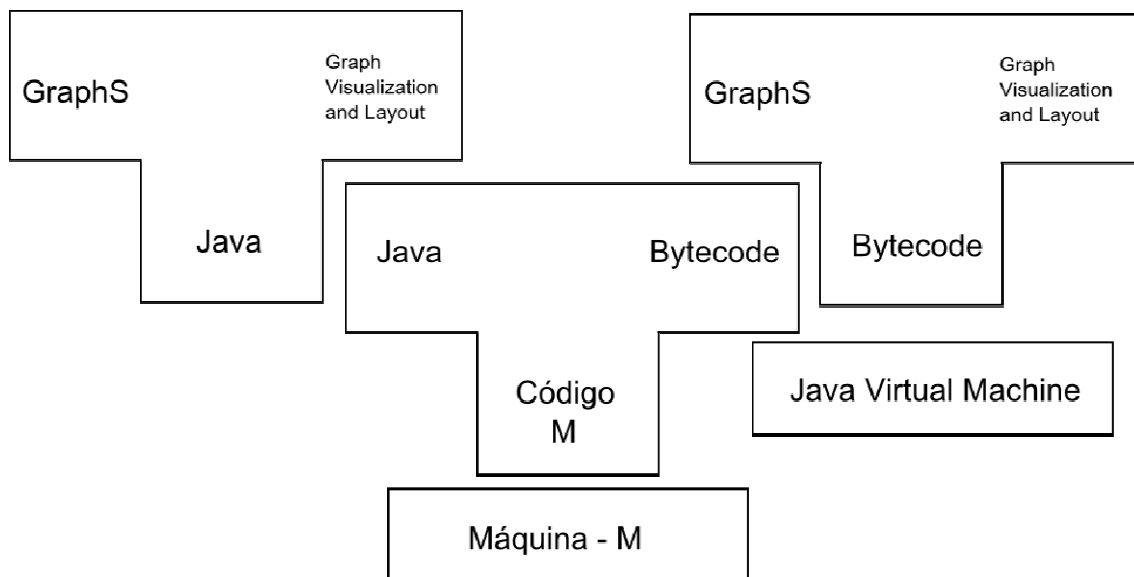
Desde un punto de vista práctico, los grafos permiten estudiar las interrelaciones entre unidades que interactúan unas con otras. Por ejemplo, una red de computadoras puede representarse y estudiarse mediante un grafo, en el cual los vértices representan terminales y las aristas representan conexiones (las cuales, a su vez, pueden ser cables o conexiones inalámbricas).

Prácticamente cualquier problema puede representarse mediante un grafo, y su estudio trasciende a las diversas áreas de las ciencias duras y las ciencias sociales.

La resolución del problema estará basada en un Procesador de Lenguajes que cogerá como entrada un fichero de texto, cuyo lenguaje se describe en el apartado 3, que definirá un conjunto finito de grafos sobre el cual se podrán aplicar una serie de algoritmos para resolver problemas como el camino mínimo entre dos nodos y el árbol de recubrimiento mínimo a partir de un nodo.

El resultado será la visualización de una imagen correspondiente al resultado de dichos algoritmos.

2. Diseño del Procesador. Diagramas en forma de “T”



Nuestra aplicación será implementada en Java y tomará como entrada un fichero de texto escrito en el lenguaje de especificación de grafos GraphS. La salida será la visualización del grafo y el resultado de las operaciones.

Para abordar la resolución del problema se hace uso de otro procesador de lenguajes (compilador *javac*) que tendrá como entrada el código fuente de nuestra aplicación y devolverá el *bytecode* perteneciente a ésta. Dicho *bytecode* será ejecutado sobre una máquina virtual de Java (Java Virtual Machina) dando lugar a nuestra aplicación.

3. Descripción del lenguaje GraphS

El lenguaje consiste en dos secciones básicas:

- La primera es opcional y está destinada a la declaración de nodos y aristas globales que podrán ser utilizados por todos los grafos.
- La segunda es obligatoria y está destinada a la definición de grafos.

Un grafo es un bloque que comienza con la palabra reservada *graph* seguida de un identificador de grafo. Su interior está compuesto por dos zonas: definición de grafo y definición de operaciones. Un grafo se puede definir mediante la declaración de sus nodos, aristas y conexiones o mediante una función de creación de grafos que implique a grafos ya definidos:

- Declaración de nodos: consta de una o más líneas que comienzan con la palabra reservada *node*, seguida de un identificador de nodo o una lista de identificadores de nodo separados por comas.
- Declaración de aristas: consta de una o más líneas que comienzan con la palabra reservada *edge*, seguida de un identificador de arista o una lista de identificadores de aristas separados por comas.
Es posible caracterizar una arista con un peso. Éste se indicará con un número entero entre paréntesis después del identificador de la arista. En caso de no indicarse un peso, se asumirá que el peso de la arista es nulo.
- Conexión de nodos: se representa mediante una asignación en la que la parte izquierda es un identificador de arista y la parte derecha son los dos identificadores de los nodos a unir con un símbolo entre ellos. Si queremos que la arista sea dirigida dicho símbolo será una flecha (->). Por el contrario si queremos que la arista sea no dirigida el símbolo será un guión (-).
- Definición de operaciones: consta de una sentencia para cada operación. Los parámetros de las operaciones se indican entre paréntesis y separados por comas. Pueden ser operaciones que engloben varios grafos o que sólo hagan referencia a uno.

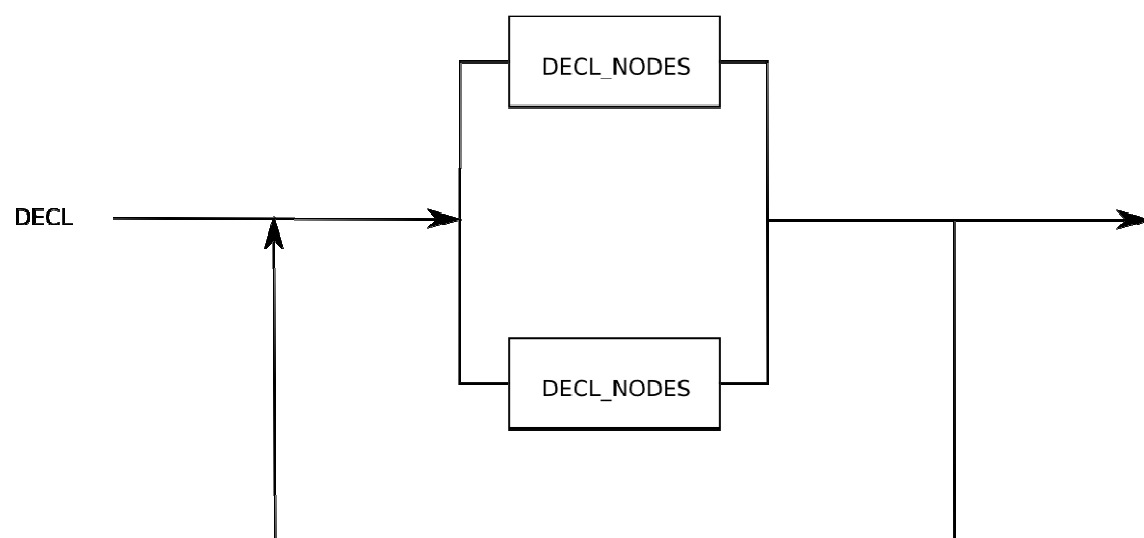
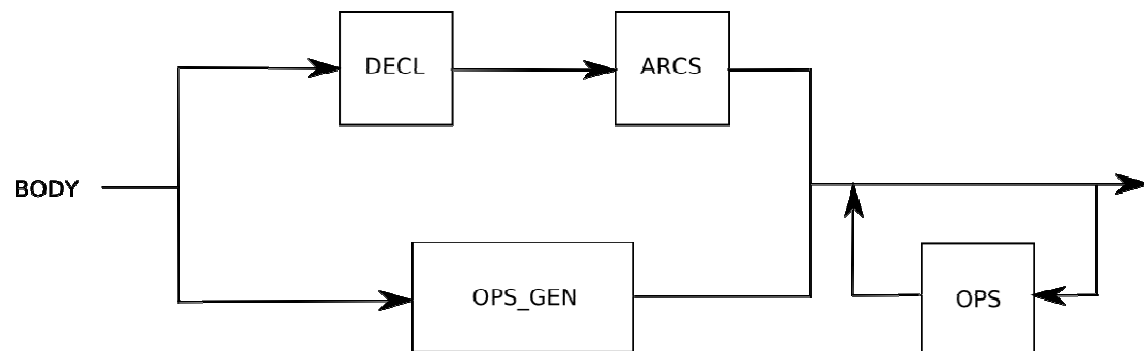
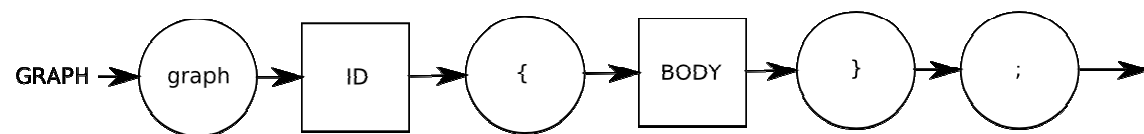
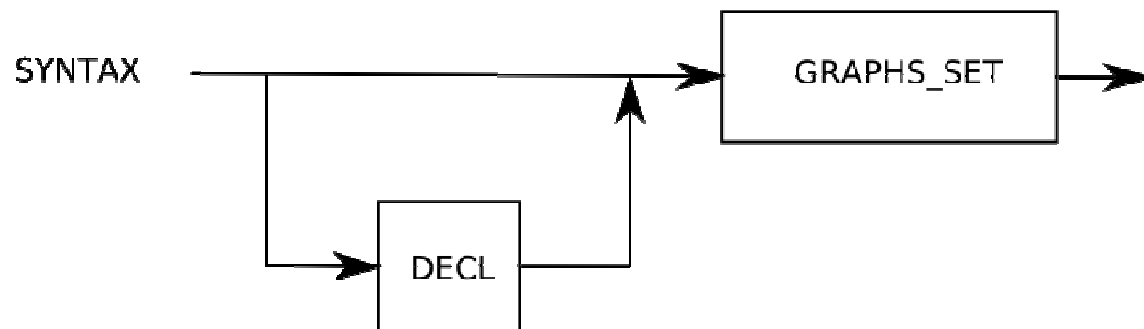
El final del bloque se indica con punto y coma una vez cerrada la llave.

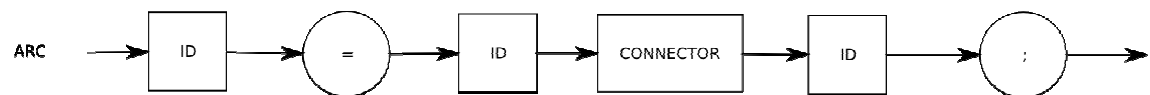
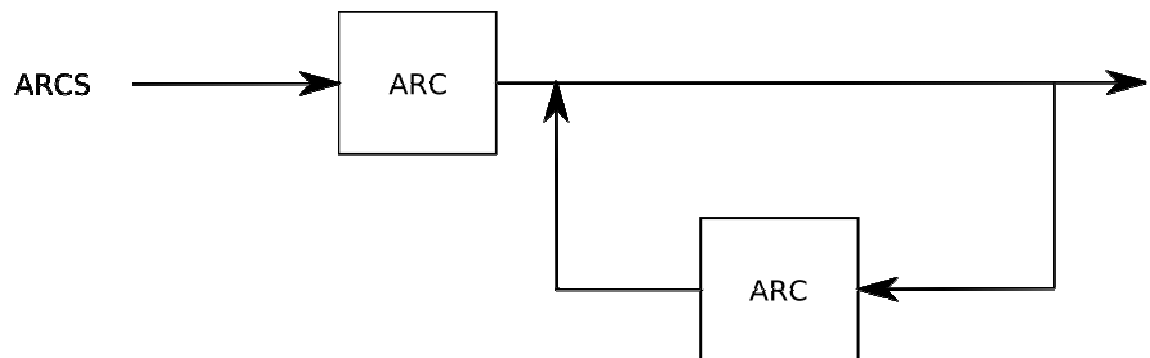
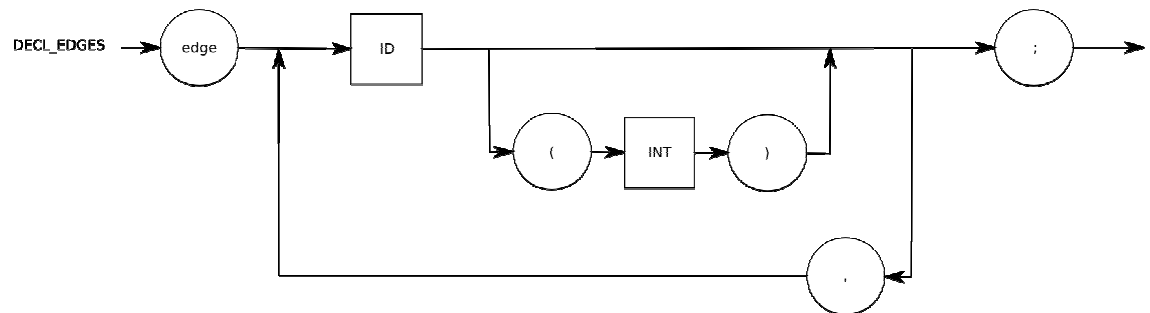
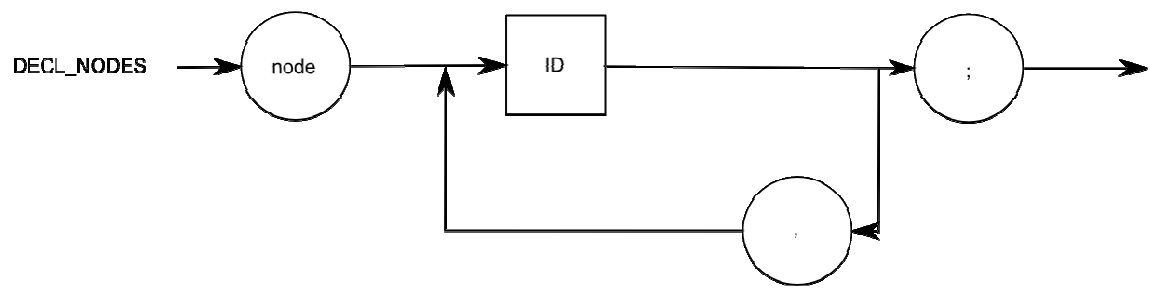
Cada sentencia termina en punto y coma.

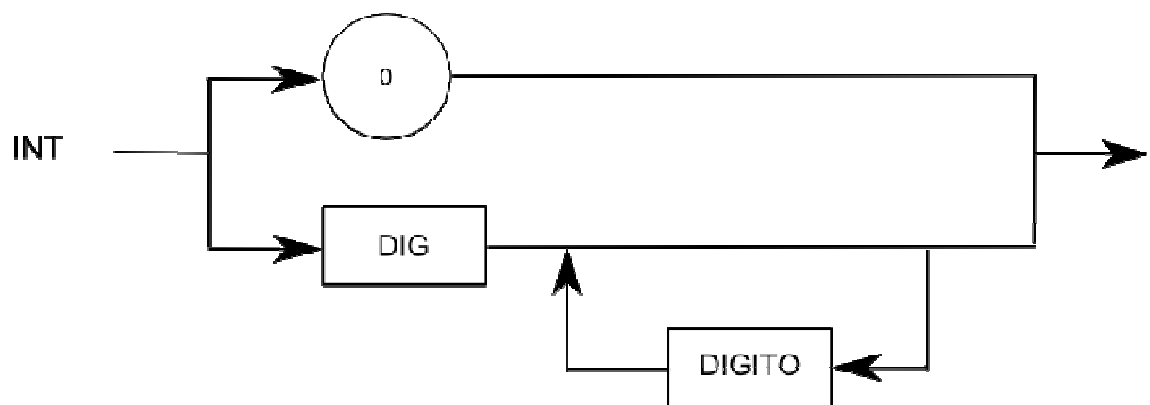
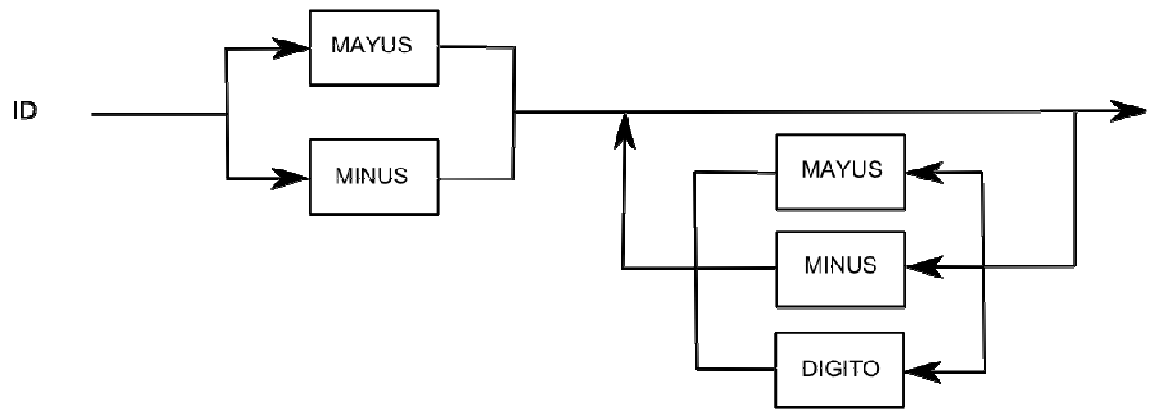
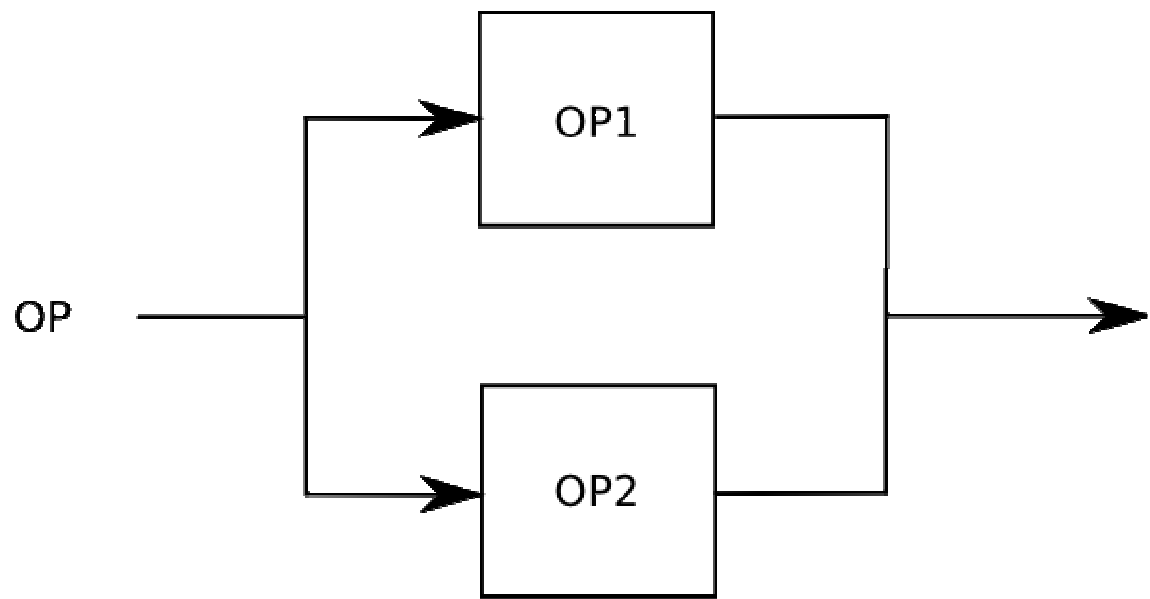
4. Especificación EBNF del lenguaje GraphS

```
SYNTAX ::= DECL GRAPHS_SET
GRAPHS_SET ::= GRAPH {GRAPH}
GRAPH ::= graph ID '{' BODY '}' ';'
BODY ::= (DECL ARCS | OPS_GEN) [OPS]
DECL ::= {DECL_NODES | DECL_EDGES}
DECL_NODES ::= node ID {' ID} ';'
DECL_EDGES ::= edge ID ['(' INT ')'] {' ID ['(' INT
')']}] ';'
ARCS ::= ARC {ARC}
ARC ::= ID '=' ID CONNECTOR ID ';'
CONNECTOR ::= '-' | '->'
OPS_GEN ::= OP_GEN
OP_GEN ::= OP5
OPS ::= {OP}
OP ::= (OP1 | OP2) ';'
OP1 ::= OPN1 '(' ID ')'
OPN1 ::= minimumSpanningTree
OP2 ::= OPN2 '(' ID ',' ID ')'
OPN2 ::= shortestPath
OP5 ::= OPN5 '(' ID ',' ID ',' ID ',' ID ',' ID ')'
OPN5 ::= union
ID ::= (MAYUS | MINUS) {MAYUS | MINUS | DIGITO}
INT ::= 0 | DIG {DIGITO}
MAYUS ::= A | B | C | D | E | F | G | H | I | J | K | L |
M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
MINUS ::= a | b | c | d | e | f | g | h | i | j | k | l |
m | n | o | p | q | r | s | t | u | v | w | x | y | z
DIG ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
DIGITO ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
COMMENT ::= '/' '/' {^LINETERMINATOR} LINETERMINATOR
COMMENT_M ::= '/' '*' {ANY_STRING | LINETERMINATOR} '*' '/'
LINETERMINATOR ::= \r | \n | \r\n
```

5. Diagramas de Conway







6. Definición de la semántica en lenguaje natural

- **Definición de grafo**

```
graph Grafo1 {  
...  
...  
};
```

Implica la definición de la estructura de datos *grafo*. En el interior del cuerpo del bloque se definirán los atributos de éste y las operaciones que podemos realizar con él.

- **Declaración de nodos**

```
node A;  
node B, C;
```

Sirve para declarar uno o más nodos de un grafo. No puede haber identificadores de nodo repetidos en una misma zona declarativa. Por el contrario, sí pueden declararse nodos locales que ya fueron declarados globalmente. En este caso, se consideraría el nodo de ámbito local.

- **Declaración de aristas**

```
edge X;  
edge Y(7), Z(5);
```

Implica la declaración de una o más aristas con sus respectivos pesos nulos o valorados. Todas las aristas deben ser del mismo tipo. Están sujetas a la mismas restricciones que los nodos en cuanto a repetición de identificadores y ámbitos de uso.

- **Conexión de nodos**

X = A – B; Y = C -> D;

Sirve para indicar que nodos une una arista y si ésta es dirigida o no dirigida.

No puede haber aristas dirigidas y no dirigidas en un mismo grafo, deber ser todas del mismo tipo.

El identificador que se encuentra a la izquierda de la asignación debe ser un identificador de arista declarado previamente. A la derecha de la asignación debe haber dos identificadores de nodo declarados previamente.

- **Definición de operaciones**

shortestPath (A, E); minimumSpanningTree (D); union (Graph1, Graph2, A, K, M);
--

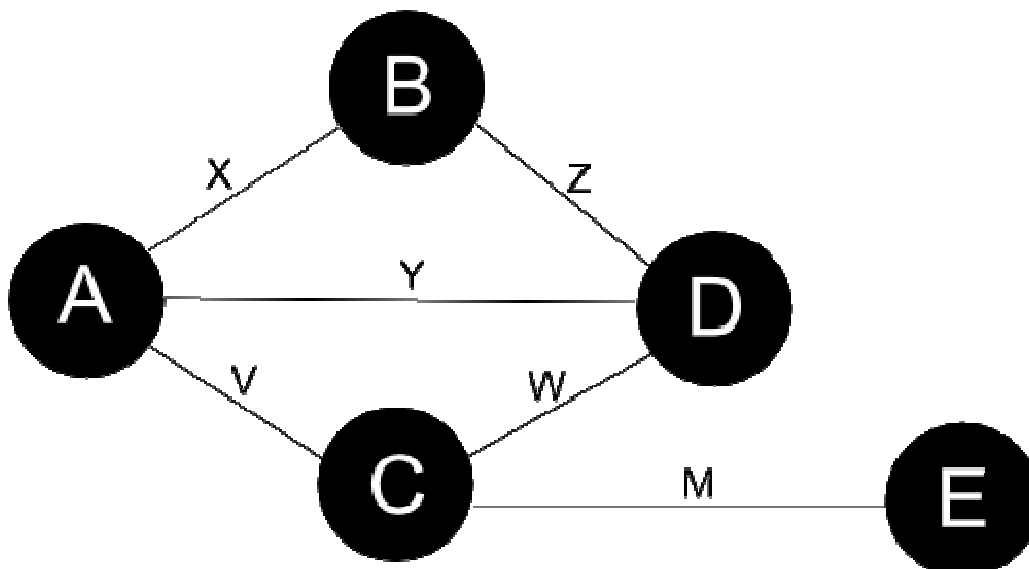
Los nodos, aristas y grafos implicados en las operaciones deben haber sido declarados anteriormente. Además, cada operación tiene sus propias restricciones semánticas:

- En la operación de camino mínimo (shortestPath) el grafo debe ser valorado y dirigido.
- En la operación del árbol de recubrimiento mínimo (minimumSpanningTree) el grafo debe ser valorado.
- En la operación de unión (union), ambos grafos deben ser del mismo tipo.

7. Ejemplos

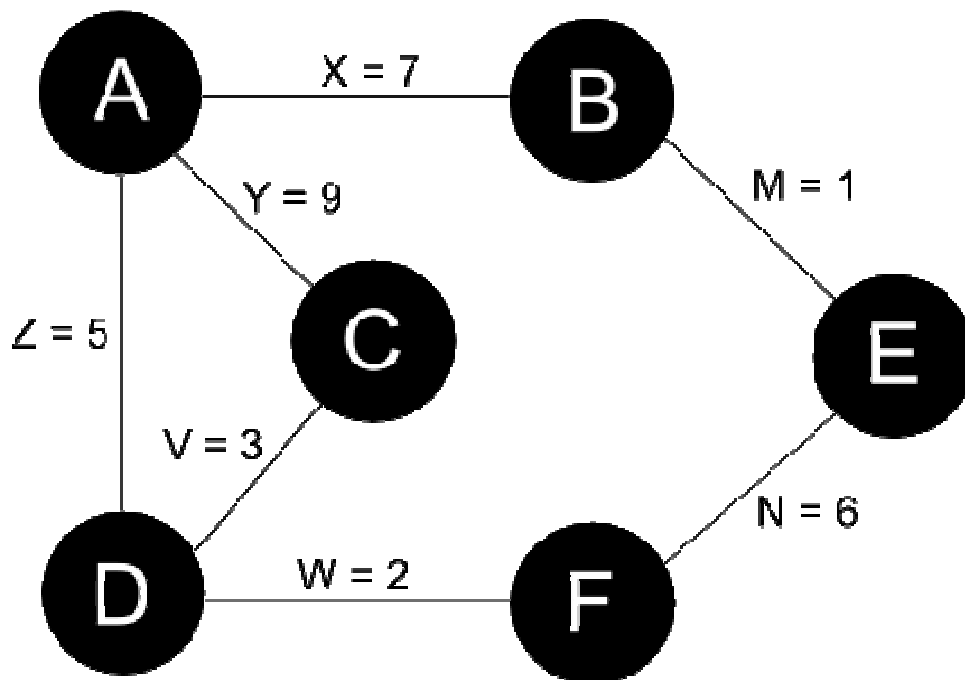
- Grafo no dirigido y no valorado

```
/* Esto es un comentario */  
//Esto también es un comentario  
  
/* Ejemplo de un grafo no dirigido y no valorado */  
graph Grafo1 {  
    //Declaración de nodos  
    node A, B, C, D;  
    node E;  
    //Declaración de aristas  
    edge X, Y, Z, V, W;  
    //no valoradas. Sería como poner X(0), Y(0)...  
    edge M;  
  
    //Conexión de nodos. El "-" indica una arista no  
    dirigida.  
    X = A - B;  
    Y = A - D;  
    V = A - C;  
    Z = B - D;  
    W = C - D;  
    M = C - E;  
  
    //Definición de operaciones  
    shortestPath (A, E);  
    //Dijkstra o Floyd -> weighted and directed graphs  
    minimumSpanningTree (D);  
    //Kruskal o Prim -> weighted graph  
};
```



- **Grafo no dirigido y valorado**

```
/* Ejemplo de un grafo no dirigido y valorado */  
graph Grafo2 {  
  node A, B, C, D, E, F;  
  edge X(7), Y(9), Z(5), V(3), W(2), M(1), N(6);  
  
  X = A - B;  
  Y = A - C;  
  Z = A - D;  
  V = D - C;  
  W = D - F;  
  M = B - E;  
  N = F - E;  
  
  shortestPath (A, F);  
  minimumSpanningTree (E);  
};
```

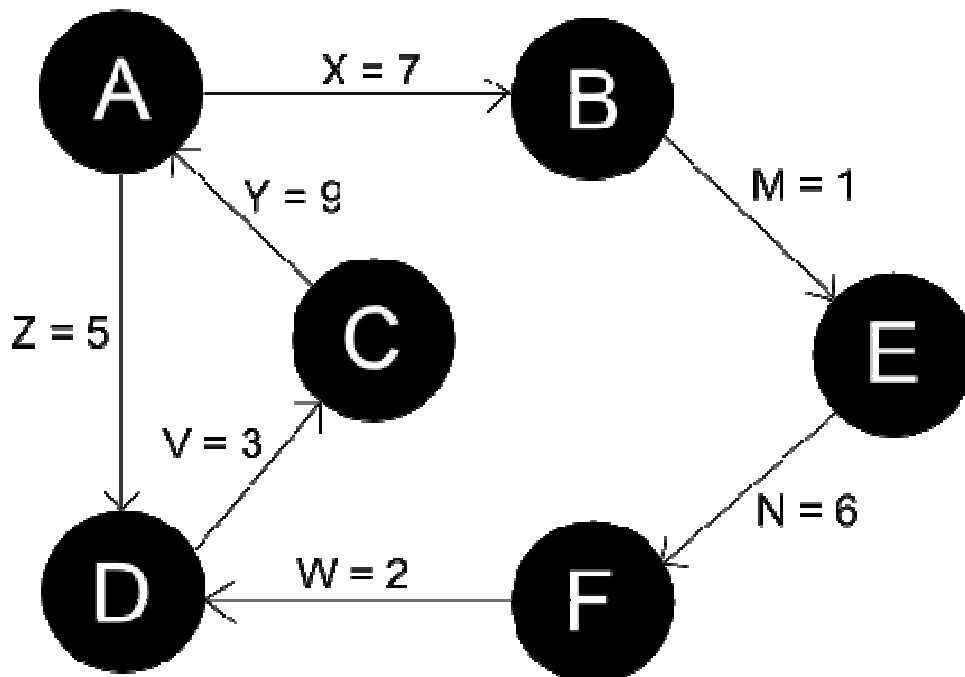


- **Grafo dirigido y valorado**

```
/* Ejemplo de un grafo dirigido y valorado */
graph Grafo3 {
  node A, B, C, D, E, F;
  edge X(7), Y(9), Z(5), V(3), W(2), M(1), N(6);

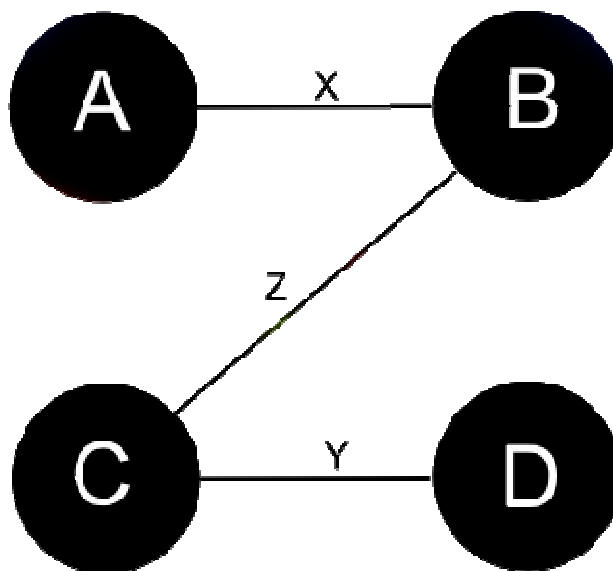
  X = A -> B; //"A -> B" indica una arista dirigida con
  origen en A y destino en B
  Y = C -> A;
  Z = A -> D;
  V = D -> C;
  W = F -> D;
  M = B -> E;
  N = E -> F;

  shortestPath (A, F);
  minimumSpanningTree (E);
};
```



- Unión de grafos

```
node A, B, C, D;  
edge X, Y, Z;  
  
graph Grafo1 {  
    node A, B;  
    edge X;  
  
    X = A - B;  
};  
  
graph Grafo2 {  
    node C, D;  
    edge Y;  
  
    Y = C - D;  
};  
  
graph Grafo3 {  
    union (Grafo1, Grafo2, B, C, Z);  
};
```



Grafo1

Grafo2

Grafo3

8. Futuras investigaciones

GraphS estará implementado en Java y será completamente portable a todo tipo de arquitecturas y sistemas operativos con una máquina virtual de Java instalada.

En principio la interfaz gráfica será desarrollada en Swing. No obstante, resulta interesante la realización de una interfaz gráfica Web para que los usuarios no tengan que instalar ningún software adicional y puedan utilizar la herramienta por medio de un navegador tradicional como Internet Explorer, Opera, Mozilla Firefox, Safari, etc.

Además, hemos planteado dos alternativas a la hora de mostrar los resultados de las operaciones:

1. Graphviz: librería de Java que genera imágenessvg de manera sencilla pero que no permiten la interacción del usuario con las partes del grafo. Se trata de un formato vectorial y libre.
2. JGraph: librería de Java que permite la visualización y la interacción con cada una de las partes del grafo.

Nos hemos decantado por JGraph ya que es más potente que una simple imagen y permite la posibilidad de implementar en un futuro la opción de **simulación de algoritmo**.