Práctica 5

Teoría de Grafos con MatLab

En esta primera práctica vamos a explicar cómo dar los primeros pasos en la Teoría de Grafos utilizando el programa MatLab. Para ello comenzaremos explicando cómo se pueden definir grafos, hacer sus representaciones gráficas y solicitar información relevante sobre los vértices. A continuación estudiaremos el problema de la conexión en grafos tanto orientados como no orientados.

5.1. Primeros pasos con MatLab

5.1.1. Definir grafos

Comencemos explicando cómo podemos definir un grafo. MatLab nos permite definir grafos de dos maneras, de forma matricial y de forma vectorial. A continuación explicamos cómo hacerlo en cada uno de los casos.

Definir un grafo de forma matricial

Comenzamos definiendo una matriz cuadrada A, de manera que su dimensión coincida con el número de vértices del grafo. Si queremos definir simplemente un grafo y no una red, la matriz A es justamente la matriz de adyacencia asociada al grafo, de manera que para cada par de vértices i, j:

$$a_{ij} = \begin{cases} 0 & \text{si } i \longrightarrow j. \\ 1 & \text{si } i \not\longrightarrow j. \end{cases}$$

Ahora bien, si queremos definir una red, esto es, un grafo de manera que cada arco tenga asignado un valor numérico, la matriz A será la matriz de adyacencia con pesos. Esto significa que el valor a_{ij} será 0, si el arco $i \longrightarrow j$ no existe, y tomará un valor real no nulo, si el arco existe y tiene asignado dicho peso.

Una vez definida la matriz de adyacencia con pesos, tenemos que tener en cuenta si el grafo es orientado o no orientado. En el caso de grafos orientados, utilizaremos la instrucción:

```
>> G=digraph(A)
```

Por otra parte, si el grafo es no orientado, la matriz de incidencia A deberá de ser simétrica ($a_{ij} = a_{ji}$). En este caso, utizaremos la instrucción:

```
>> G=graph(A)
```

Si la matriz introducida es no simétrica y utilizamos la instrucción `graph`, MatLab dará un error, indicando que para construir un grafo no orientado es necesario utilizar una matriz simétrica.

Ejemplo 5.1. *Consideremos el grafo orientado G y su grafo no orientado asociado que denotaremos por F, definidos en la Figura 5.1. Como podemos ver, cada arco tiene asociado un peso. En el caso de los arcos $2 \longrightarrow 3$ y $3 \longrightarrow 2$, ambos tienen asociado el mismo peso 2.*

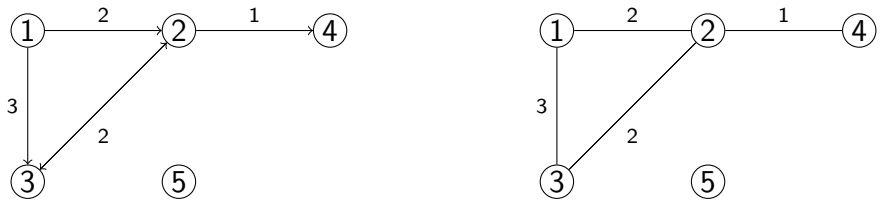

Figura 5.1: Grafo orientado G (izquierda) y su grafo no orientado asociado (derecha), denotado por F. Para el grafo orientado, el peso de los arcos $2 \longrightarrow 3$ y $3 \longrightarrow 2$ es en ambos casos de 2.

Las matrices de adyacencia con pesos de estos grafos, que denotaremos por A_1 y A_2, vienen dadas por:

$$A_1 = \begin{pmatrix} 0 & 2 & 3 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad A_2 = \begin{pmatrix} 0 & 2 & 3 & 0 & 0 \\ 2 & 0 & 2 & 1 & 0 \\ 3 & 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Como podemos observar, la matriz A_2, al estar asociada a un grafo no orientado, es simétrica. Para introducirlas en MatLab, usamos las siguientes órdenes:

```
>> A1=[0 2 3 0 0;0 0 2 1 0;0 2 0 0 0;0 0 0 0 0;0 0 0 0 0];
>> A2=[0 2 3 0 0;2 0 2 1 0;3 2 0 0 0;0 1 0 0 0;0 0 0 0 0];
```

Una vez introducidas las matrices de incidencia con pesos, podemos definir los grafos orientados y no orientados, respectivamente, mediante:

```
>> G=digraph(A1)
G=

  digraph with properties:

    Edges: [5x2 table]
    Nodes: [5x0 table]
>> G=graph(A2)
```

```
F=

  graph with properties:

    Edges: [4x2 table]
    Nodes: [5x0 table]
```

Como vemos, la salida nos informa de que el grafo orientado G tiene 5 arcos y 5 nodos, mientras que el grafo no orientado F sigue teniendo 5 vértices pero solamente 4 aristas (los arcos $2 \longrightarrow 3$ y $3 \longrightarrow 2$ dan lugar a la misma arista).

A la hora de definir un grafo, podemos estar interesados en dar un nombre a cada uno de los vértices. Por ejemplo, si cada vértice representa una ciudad, una localización, ... Para introducir dichos nombre, simplemente necesitamos definir un vector de caracteres que contenga los nombres de cada vértice. Por supuesto, el tamaño del vector deberá de coincidir con el número de vértices del grafo. Para un grafo con tres nodos la instrucción sería:

```
>> Nodos={'Nodo 1','Nodo 2','Nodo 3'};
>> G=digraph(A,Nodos)
```

En el caso de un grafo no orientado, simplemente sustituiríamos la instrución `digraph` por la instrucción `graph`.

Una vez definido el grafo, podemos solicitar por pantalla la información referida a los vértices, a los arcos/aristas o a los pesos mediante las instrucciones:

```
>> G.Nodes
>> G.Edges
>> G.Edges.Weight
```

Ejemplo 5.2. *Continuando con el Ejemplo 5.1, cada nodo hace referencia a un aeropuerto, mientras que los arcos hacen referencia a conexiones entre aeropuertos, y su peso sería el tiempo de vuelo. En el caso del grafo F no orientado, cada arista haría referencia a la presencia de una conexión entre ambos aeropuertos y en ambas direcciones, mientras que para el grafo G orientado cada arco $i \longrightarrow j$ hace referencia a la existencia de un vuelo desde el aeropuerto i hasta el aeropuerto j. En este caso, podríamos introducir esta información de la siguiente manera:*

```
>> A1=[0 2 3 0 0;0 0 2 1 0;0 2 0 0 0;0 0 0 0 0;0 0 0 0 0];
>> Nodos={'Aeropuerto 1','Aeropuerto 2','Aeropuerto 3',
'Aeropuerto 4','Aeropuerto 5'};
>> G=digraph(A1,Nodos)
G

  digraph with properties:

    Edges: [5x2 table]
    Nodes: [5x0 table]
```

```
>> G.Nodes

ans =

      Name
   --------------

   'Aeropuerto 1'
   'Aeropuerto 2'
   'Aeropuerto 3'
   'Aeropuerto 4'
   'Aeropuerto 5'
>> G.Edges

ans=

               EndNodes                Weight
   ------------------------------      ------

   'Aeropuerto 1'  'Aeropuerto 2'      2
   'Aeropuerto 1'  'Aeropuerto 3'      3
   'Aeropuerto 2'  'Aeropuerto 3'      2
   'Aeropuerto 2'  'Aeropuerto 4'      1
   'Aeropuerto 3'  'Aeropuerto 2'      2
```

De esta manera disponemos de la información relativa tanto a los nodos como a los arcos. Análogamente podemos proceder con el grafo no orientado F.

Definir un grafo a través de los arcos

Para definir grafos usando arcos/aristas, distinguiremos una vez más entre grafos orientados y no orientados:

a) **Grafos orientados:** En este caso, definimos dos vectores, un vector s y un vector t, ambos con tamaño igual a m (el número de arcos en el grafo), de manera que $s(i)$ y $t(i)$ indican los vértices inicial y final del i-ésimo arco, respectivamente. A continuación, definimos el grafo utilizando la siguiente instrucción:

```
>> G=digraph(s,t)
```

Además, si queremos añadir pesos a los arcos, podemos definir un vector `peso` de manera que en la posición i asigna el peso del arco $s(i) \longrightarrow t(i)$. La instrucción a utilizar sería en este caso:

```
>> G=digraph(s,t,peso)
```

b) **Grafos no orientados:** En este caso, volvemos a definir dos vectores s y t, ambos con tamaños igual a m (el número de aristas), de manera que $s(i)$ y $t(i)$ indican los vértices inicial y final de la i-ésima arista. En este caso, el orden en el que pongamos los vértices inicial y final es indiferente y solamente lo introduciremos una vez (por ejemplo, la arista $1-2$ se puede introducir como $1-2$ o $2-1$, pero solamente se introducirá en una ocasión). La orden que debemos utilizar es:

```
>> G=graph(s,t)
```

Al igual que en el caso de grafos orientados, si queremos añadir pesos, utilizaríamos la instrucción:

```
>> G=graph(s,t,peso)
```

Ejemplo 5.3. *Continuemos con los grafos G y F del Ejemplo 5.1 definidos en la Figura 5.1. Si queremos definir el grafo G utilizando los arcos, debemos definir los vectores s, t y* peso *de manera que la componente i-ésima contenga los vértices inicial y final del arco i-ésimo y el peso de dicho arco:*

```
>> s=[1 1 2 2 3];
>> t=[2 3 3 4 2];
>> peso=[2 3 2 1 2];
```

Con el orden introducido, estaríamos diciendo que el primer arco es $1 \longrightarrow 2$ con un peso de 2, el segundo arco $1 \longrightarrow 3$ con un peso de 3 y así sucesivamente. Por último, definiríamos el grafo G mediante la expresión:

```
>> G=digraph(s,t,peso)

G

  digraph with properties:

    Edges: [4x2 table]
    Nodes: [5x0 table]
```

En el caso del grafo no orientado F, hemos de introducir en los vectores s y t los vértices de cada arista. Una posibilidad para hacerlo sería la siguiente:

```
>> s=[1 1 2 2];
>> t=[2 3 3 4];
>> peso=[2 3 2 1];
```

Y definimos el grafo no orientado mediante:

```
>> F=graph(s,t,peso)

F

 graph with properties:

   Edges: [4x2 table]
   Nodes: [4x0 table]
```

Al igual que hicimos en el caso de grafos definidos a través de la matriz de incidencia con pesos, cuando definimos el grafo a través de los arcos también podemos asignar nombres a los vértices. En este caso, debemos introducir los vectores s y t que definen los arcos/aristas del grafo, a continuación un vector de pesos y un último vector de caracteres con los nombres de los vértices. En el caso de un grafo orientado con 3 vértices, la expresión sería:

```
>> Nodos={'Nodo 1','Nodo 2','Nodo 3'};
>> G=digraph(s,t,peso,Nodos)
```

En el caso de grafos no orientados, simplemente sustituiríamos la instrucción `digraph` por `graph`. Al igual que hicimos anteriormente, una vez definido el grafo podemos utilizar los comandos `G.Nodes` y `G.Edges` para obtener por pantalla la información sobre los vértices y los arcos/aristas, respectivamente.

Hemos visto cómo definir grafos usando o bien la matriz de incidencia o bien los arcos. En ambos casos, el grafo obtenido es el mismo, así que utilizaremos la manera que en cada caso nos resulte más cómoda.

5.1.2. Modificación de vértices y arcos/aristas

Una vez definido un grafo (tanto orientado como no orientado), podemos realizar ciertas operaciones como añadir o quitar vértices y/o arcos/aristas. Explicamos a continuación cómo realizar cada una de las operaciones:

a) **Añadir vértices.** Para añadir vértices a un grafo G, utilizaremos la instrucción `addnode`, donde debemos espcificar el nombre del grafo creado y, o bien el número de vértices a crear, o bien un vector de caracteres con los nombres de los nuevos vértices. Si por ejemplo deseamos añadir dos nodos, podríamos utilizar una de las siguientes instrucciones:

   ```
   >> H=addnode(G,2)
   >> H=addnode(G,{'Nodo 1','Nodo 2'})
   ```

   En el primer caso añadirá dos vértices que los numerará continuando con la numeración presente en G, mientras que con la segunda instrucción creará dos vértices a los que llamará `Nodo 1` y `Nodo 2`. En ambos casos, el nuevo grafo se almacenará con el nombre `H`.

b) **Eliminar vértices.** La eliminación de vértices en un grafo G se hace mediante el comando `rmnode`. En dicho comando se debe de especificar el grafo al que hacemos referencia así como los vértices a eliminar. Si éstos están simplemente numerados, se indicará el número correspondiente a dichos vértices; si por contra los vértices tienen un nombre en forma de caracteres, utilizaremos un vector de caracteres para indicar los vértices que deseamos eliminar. Por ejemplo, si se desea eliminar de un grafo G el vértice numerado como 3, usaremos la expresión:

```
>> H=rmnode(G,3)
```

Mientras que si de un grafo G deseamos eliminar el vértice llamado `'Nodo 3'`, usaremos la expresión:

```
>> H=rmnode(G,{'Nodo 3'})
```

Hay que decir que al eliminar un vértice, inmediatamente se eliminan también todos los arcos incidentes en él.

c) **Añadir arcos.** Para añadir arcos utilizaremos una expresión similar a la utilizada para definir los grafos a través de los arcos. El comando a utilizar es `addedge`, donde indicaremos el nombre del grafo y a continuación dos vectores s y t que contendrán, en el caso de grafos orientados, los vértices inicial y final de los arcos a añadir, respectivamente, mientras que en el caso de grafos no orientados s y t simplemente contendrán los vértices que definen los arcos así como un vector de pesos. Por ejemplo, si deseamos añadir los arcos $3 \longrightarrow 5$ y $2 \longrightarrow 1$, con los pesos 4 y 5, respectivamente, usaremos las siguientes instrucciones:

```
>> s=[3 2];
>> t=[5 1];
>> peso=[4 5];
>> H=addedge(G,s,t,peso)
```

En caso de que los vértices hayan sido nombrados con caracteres, los vectores s y t serán también vectores de caracteres.

d) **Eliminar arcos.** Para eliminar arcos de un grafo usaremos la expresión `rmedge`. En dicha expresión debemos de indicar el nombre del grafo y, al igual que en el apartado anterior, los vectores s y t que definan los arcos/aristas a eliminar. Por ejemplo, si se desea eliminar los arcos $3 \longrightarrow 1$ y $3 \longrightarrow 4$, se utilizarán los siguientes comandos:

```
>> s=[3 3];
>> t=[1 4];
>> H=rmedge(G,s,t)
```

Una vez más, si los vértices tienen un nombre en forma de caracter, los vectores s y t serán vectores de caracteres indicando los arcos/aristas a eliminar.

Ejemplo 5.4. *En el grafo orientado G definido en el Ejemplo 5.1, podemos observar que el* `Aeropuerto 5` *no tiene conexiones con ninguno de los otros aeropuertos, así que podemos eliminarlo de nuestro grafo:*

```
>> G2=rmnode(G,{'Aeropuerto 5'})
```

Además, dado que el vuelo desde el `Aeropuerto 1` *al* `Aeropuerto 2` *ha perdido un gran número de viajeros en los últimos meses, la compañía que operaba este vuelo lo ha cancelado. Hemos de eliminar por tanto este arco:*

```
>> G3=rmedge(G2,{'Aeropuerto 1'},{'Aeropuerto 2'})
```

Sin embargo, para no perder su presencia en el `Aeropuerto 1`*, la compañía ha decidido crear una nuevo conexión tanto de ida como de vuelta desde el* `Aeropuerto 1` *a un nuevo aeropuerto que llamaremos* `Aeropuerto 6`*. El tiempo de vuelo es de 1.5 horas, tanto para la ida y la vuelta:*

```
>> G4=addnode(G3,{'Aeropuerto 6'})
>> G5=addedge(G4,{'Aeropuerto 1','Aeropuerto 6'},
{'Aeropuerto 6','Aeropuerto 1'},[1.5 1.5])
```

Con esto, el nuevo grafo obtenido (que finalmente ha sido llamado `G5`*), se puede representar gráficamente como muestra la Figura 5.2.*

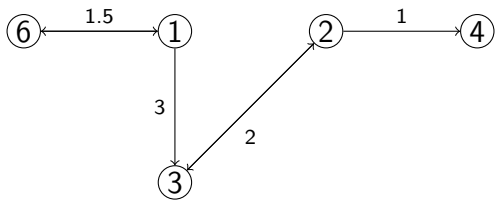

Figura 5.2: Grafo G modificado tras añadir y eliminar aeropuertos y conexiones.

Otros dos comandos que pueden ser de ayuda son `numnodes` y `numedges`, que nos indican el número de vértices y de arcos que hay el grafo, respectivamente. En el Ejemplo 5.4 anterior, si queremos saber el número de vértices y de arcos que hay en el grafo final obtenido (llamado `G5`), simplemente utilizaremos las siguiente instrucciones:

```
>> numnodes(G5)
    5
>> numedges(G5)
    6
```

5.1.3. Representación gráfica

Una vez definido un grafo en MatLab podemos hacer su representación gráfica. Ésto se puede hacer de manera sencilla mediante el comando `plot`, indicando el nombre del grafo que estamos trabajando. Si el grafo se ha almacenado con el nombre `G`, utilizaremos la instrucción:

```
>> plot(G)
```

La instrucción `plot` permite hacer una primera representación del grafo. Sin embargo, tenemos varias opciones para mejorar el gráfico obtenido.

a) **layout**. Podemos solicitar que MatLab dibuje los vértices y arcos de manera diferente. Para ello tenemos que utilizar el comando `layout` con alguna de las siguientes opciones: `circle, force, layered, subspace`. Por ejemplo:

```
>> h=plot(G)    # Almacenamos el gráfico con el nombre h para referirnos
                # a él a continuación.
>> layout(h,'force')
```

b) **labeledge.** En un gráfico podemos añadir un nombre a cada uno de los arcos. Por ejemplo, podemos estar interesados en añadir a cada arco su peso. En tal caso usaríamos el comando:

```
>> labeledge(h,1:numedges(G),peso)
```

De esta manera, a cada uno de los arcos les asignará el valor del vector `peso`.

c) **highlight.** Con este comando podemos remarcar nodos o arcos de un grafo no orientado, para dibujarlos en distinto color o tamaño. Por ejemplo para un grafo no orientado cuya figura se almacenó bajo el nombre de `h`, para remarcar en rojo los vértices 1 y 2 en color rojo, utilizamos la instrucción:

```
>> highlight(h,[1 2],'NodeColor','r')
```

Si por contra queremos remarcar varios arcos/aristas, indicaremos dos vectores s y t donde, al igual que hemos hecho en otras ocasiones, indicaremos los vértices inicial y final de los arcos/aristas. Por ejemplo, si en un grafo orientado queremos remarcar en rojo los arcos $1 \longrightarrow 2$ y $2 \longrightarrow 3$, usaríamos la instrucción:

```
>> highlight(h,[1 2],[2 3],'EdgeColor','r')
```

Una vez ejecutada la instrucción nos aparecerá una nueva pantalla con el gráfico solicitado.

Ejemplo 5.5. *Continuemos con el grafo modificado* `G5` *que hemos construído en el Ejemplo 5.4 y que hemos representado gráficamente en la Figura 5.2. Si queremos que* MatLab *lo represente gráficamente, utilizamos la instrucción:*

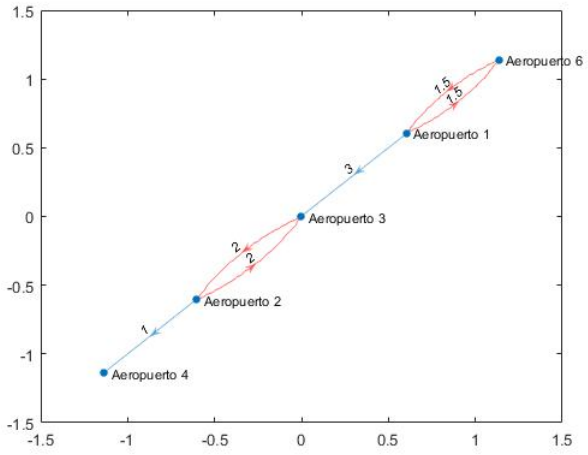

Figura 5.3: Representación gráfica de MatLab del grafo G5.

```
>> h=plot(G5)
```

Si por ejemplo queremos que en cada arco aparezca el tiempo de vuelo y además queremos remarcar en rojo las conexiones de ida y vuelta, es decir, los vuelos de ida y vuelta entre los aeropuertos 1 y 6 y entre los aeropuertos 2 y 3, usaremos la siguiente instrucción:

```
>> labeledge(h,1:numedges(G5),peso)
>> highlight(h,{'Aeropuerto 1','Aeropuerto 6','Aeropuerto 2','Aeropuerto 3'},
{'Aeropuerto 6','Aeropuerto 1','Aeropuerto 3','Aeropuerto 2'},'EdgeColor','r')
```

Obtendremos por pantalla la representación gráfica que podemos observar en la Figura 5.3.

5.1.4. Información sobre el grafo

Cuando hayamos definido el grafo, podemos solicitar información sobre él. Distinguiremos los casos de grafos orientados y no orientados.

Grafos orientados

Comencemos con un grafo orientado G. Sobre él podemos calcular los índices de incidencia de entrada y de salida, así como los antecesores y predecesores de cada vértice.

a) **Grado de incidencia de entrada.** Recordemos que, para cada vértice, este número indica el número de arcos que tiene a dicho vértice como vértice final. Para calcularlo, utilizaremos el comando `indegree`:

```
>> indegree(G)
```

b) **Grado de incidencia de salida.** El grado de incidencia de salida de un vértice dado es el número de arcos que tienen a dicho vértice como vértice de entrada. Para calcularlo utilizaremos la instrucción `outdegree`:

```
>> outdegree(G)
```

c) **Predecesores de un vértice.** Los predecesores de un vértice i son los vértices j de manera que $j \longrightarrow i$ es un arco. Para calcularlo utilizaremos la instrucción `predecessors`, indicando el nombre del grafo y el vértice sobre el que queremos calcular sus predecesores. Por ejemplo, para calcular los predecesores del vértice 1, usaremos la instrucción:

```
>> predecessors(G,1)
```

Si los vértices vienen nombrados como caracteres, llamaremos al vértice usando su nombre como caracter. Por ejemplo, para calcular los predecesores del vértice `'Nodo 1'`:

```
>> predecessors(G,'Nodo 1')
```

d) **Sucesores de un vértice.** Los sucesores de un vértice i son los vértices j de manera que $i \longrightarrow j$ es un arco. Para calcularlo utilizaremos la instrucción `sucessors`, indicando el nombre del grafo y el vértice sobre el que deseamos trabajar. Al igual que en el apartado anterior, para indicar el nodo lo haremos a través de su número o de su nombre caracter, según venga expresado. Por ejemplo, para calcular los sucesores del vértice 1, usaremos la instrucción:

```
>> sucessors(G,1)
```

e) **Determinar si hay ciclos.** En muchas ocasiones nos interesará saber si el grafo considerado es acíclico o si, por contra, tiene ciclos. Para ello, símplemente usaremos la instrucción `isdag`, indicando el nombre del grafo:

```
>> isdag(G)
```

El valor obtenido será un 1 si el grafo es acíclico, o 0 en el caso en el que sí haya ciclos.

Ejemplo 5.6. *Continuemos una vez más con el grafo que hemos modificado en el Ejemplo 5.4. Para calcular los grados de incidencia de entrada y de salida, podemos calcularlos mediante la instrucción:*

```
>> [indegree(G5),outdegree(G5)]

ans=

    1     2
    1     2
    2     1
    1     0
    1     1
```

Así, obtenemos los grados de incidencia de entrada (primera columna) y de salida (segunda columna) de cada uno de los vértices. En el lenguaje de nuestro ejemplo, los grados de incidencia de entrada y de salida de un aeropuerto i indica el número de aeropuertos desde los cuales se puede volar al aeropuerto i y el número de aeropuertos a los que se puede volar desde el aeropuerto i, respectivamente. Por otra parte, podemos calcular los predecesores y sucesores de cada vértice. En nuestro caso, por ejemplo para el `Aeropuerto 1`*, los predecesores son los aeropuertos desde los que se puede volar al* `Aeropuerto 1` *mientras que los sucesores son los aeropuertos a los que se puede volar desde el* `Aeropuerto 1`*:*

```
>> predecessors(G5,'Aeropuerto 1')
ans=
   'Aeropuerto 6'
>> successors(G5,'Aeropuerto 1')
ans=
   'Aeropuerto 3'
   'Aeropuerto 6'
```

Concluimos por tanto que al `Aeropuerto 1` *solamente se puede volar desde el* `Aeropuerto 6`*, pero desde el* `Aeropuerto 1` *se puede volar tanto al* `Aeropuerto 6` *como al* `Aeropuerto 3`*.*

Por último, nos interesa saber si en este grafo hay ciclos. Para ello usamos la instrucción:

```
>> isdag(G5)
ans =
     0
```

Eso significa que el grafo no es acíclico al tener algún ciclo. Esto no es sorprendente, puesto que ya sabemos que hay un vuelo de ida y vuelta entre los aeropuertos 1 y 6.

Grafos no orientados

En el caso de grafos no orientados, podemos calcular tanto el grado de incidencia como los precedesores/sucesores (que en el caso de grafos no orientados coinciden).

a) **Grado de un vértice.** El grado de un vértice i es el número de vértices j de manera que $i-j$ es una arista del grafo. En este caso, calcularemos los grados de los vértices de un grafo G mediante la instrucción `degree`:

   ```
   >> degree(G)
   ```

b) **Predecesores/Sucesores de un vértices.** Dado que estamos trabajando con grafos no orientados, los sucesores y predecesores son equivalentes. Para calcularlos, usaremos el comando `neighbors`, indicando el nombre del grafo y el vértice del que queremos calcular los predecesores/sucesores. Al igual que ocurría en el caso de grafos orientados, el vértice se indicará mediante número o caracter, en función de cómo estén definidos. Por ejemplo, para calcular los predecesores/sucesores del vértice 4, usaremos la instrucción:

   ```
   >> neighbors(G5,4)
   ```

5.1.5. Representación matricial

Como hemos visto en las clases de teoría, un grafo se puede representar de manera matricial de varias formas. Las representaciones matriciales son muy útiles porque nos permiten simplificar los cálculos. Con MalLab podemos calcular las matrices de adyacencia y de incidencia vértice-arco:

a) **Matriz de adyacencia.** Como vimos al inicio de la práctica, es la matriz cuadrada que tiene tantas filas y columnas como vértices hay en el grafo, y definida por $a_{ij} = 1$ is $i \longrightarrow j$ es un arco o $a_{ij} = 0$ en otro caso. Para calcular la matriz de adyacencia, usaremos el comando `adjacency`:

```
>> adjacency(G)
```

Usando esta expresión nos indicará los arcos que tiene el grafo. Si queremos que nos muestre la matriz completa, debemos añadir el comando `full`:

```
>> full(adjacency(G))
```

b) **Matriz de incidencia vértice-arco.** Esta matriz solamente se puede calcular para los grafos orientados. Esta matriz tiene tantas columnas como vértices y tantas filas como arcos, y en el elemento m_{ie} toma el valor 1, si i es el vértice inicial del arco e; m_{ie} toma el valor -1 si i es el vértice final del arco e, y toma el valor 0 en otro caso. Para calcularla, utilizaremos el comando `incidence`:

```
>> incidence(G)
```

Al igual que en el apartado anterior, por pantalla nos mostrará una lista con los arcos (i, j) y nos indicará el valor 1 o -1 según el vértice inicial sea i o j, respectivamente. Una vez más, si queremos la matriz completa debemos utilizar la instrucción `full`:

```
>> full(incidence(G))
```

Ejemplo 5.7. *Continuemos con el grafo del Ejemplo 5.4. Vamos a calcular sus matrices de adyacencia y de incidencia vértice-arco:*

```
>> adjacency(G5)

ans =

    (5,1)        1
    (3,2)        1
    (1,3)        1
    (2,3)        1
    (2,4)        1
    (1,5)        1
```

```
>> full(adjacency(G5))

ans =

      0       0       1       0       1
      0       0       1       1       0
      0       1       0       0       0
      0       0       1       1       0
      1       0       0       0       0

>> incidence(G5)

ans =

    (1,1)       -1
    (3,1)        1
    (1,2)       -1
    (5,2)        1
    (2,3)       -1
    (3,3)        1
    (2,4)       -1
    (4,4)        1
    (2,5)        1
    (3,5)       -1
    (1,6)        1
    (5,6)       -1

>> full(incidence(G5))

ans =

    -1      -1       0       0       0       1
     0       0      -1      -1       1       0
     1       0       1       0      -1       0
     0       0       0       1       0       0
     0       1       0       0       0      -1
```

Concluimos que las matrices de adyacencia (A) y de incidencia vértices-arco (M) vienen dadas por:

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad M = \begin{pmatrix} -1 & -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & -1 & 1 & 0 \\ 1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \end{pmatrix}$$

5.1.6. Conexión

Una vez que ya hemos explicado cómo definir grafos y hacer operaciones básicas con ellos, podemos estudiar la conexión. Recordemos brevemente el significado de la conexión en grafos orientados y no orientados:

a) **Conexión en grafos no orientados.** Un grafo no orientado es conexo cuando existe una cadena entre cada par de vértices.

b) **Conexión débil en grafos orientados.** Un grafo orientado es débilemente conexo cuando su grafo no orientado asociado es conexo.

c) **Conexión en grafos orientados.** Un grafo orientado es conexo cuando entre cada par de vértices i, j existe un camino de i a j y otro camino de j a i.

Cuando un grafo (orientado o no orientado) no es conexo (o débilmente conexo), es interesante conocer cuáles son las componentes conexas del grafo. Una componente conexa es un subgrafo conexo (o débilmente conexo) al que no es posible añadirle un vértice porque perdería esta propiedad.

Con el fin de conocer si un grafo es conexo, y conocer sus componentes conexas, utilizaremos la instrucción `conncomp`. Vamos a detallar su uso:

a) Si G es un grafo no orientado, utilizaremos la instrucción anterior en la que únicamente indicaremos el nombre del grafo:

```
>> conncomp(G)
```

Como resultado obtendremos un vector con tantos valores como vértices, de manera que para el vértice i indica a qué componente conexa pertenece. Si solamente hay una componente conexa, el grafo G será conexo.

b) Si G es un grafo orientado y queremos estudiar su posible conexión, seguiremos los mismos pasos del apartado anterior.

c) Si G es un grafo orientado y queremos ver si es débilmente conexo, esto es, si su grafo no orientado asociado es conexo, usaremos la siguiente instrucción:

```
>> conncomp(G,'Type','weak')
```

En este caso, indicamos que el *tipo* de conexión es *débil* (weak), y por pantalla nos volverá a sacar un vector indicando la componente conexa a la que pertenece cada vértice.

Ejemplo 5.8. *En el Ejemplo 5.4 teníamos un grafo orientado en el que teníamos varios aeropuertos e indicábamos las conexiones existentes entre ellos. Pasamos a continuación a estudiar la conexión en este grafo:*

```
>> conncomp(G5)

ans =

     1     2     2     3     1
```

De esta manera obtenemos que hay 3 componentes conexas en este grafo:
Componente conexa 1: Aeropuerto 1, Aeropuerto 6.
Componente conexa 2: Aeropuerto 2, Aeropuerto 3.
Componente conexa 3: Aeropuerto 4.

Así vemos que se pueden hacer viajes de ida y vuelta entre los aeropuertos 1 y 6, y entre los aeropuertos 2 y 3. Estudiemos a continuación la conexión débil de este grafo:

```
>> conncomp(G5,'Type','weak')

ans =

     1     1     1     1     1
```

Esto significa que el grafo es débilmente conexo. El significado es que, si todo vuelo de ida tuviese un vuelo de vuelta, el grafo sería conexo, es decir, se podrían hacer vuelos de ida y vuelta entre cada par de aeropuertos.

5.2. Ejemplo ilustrativo

En una gran ciudad se está planeando la construcción de un nuevo barrio en el que se construirán dos edificios, un parque, un centro comercial y unas instalaciones deportivas. Con el fin de hacer una planificación urbanística correcta, los técnicos del ayuntamiento están analizando una posible distribución del tráfico. En la Figura 5.4 tenemos un primer borrador donde se puede ver lo siguiente:

- Los dos edificios, pintados en gris.
- El parque, pintado en verde.
- Dos rotondas en las que se planea incluir una fuente, pintadas en azul.
- Un centro comercial, pintado en rojo.
- Unas instalaciones deportivas, pintadas en amarillo.

Además, cada uno de los edificios, tanto los habitables como el centro comercial y las instalaciones deportivas, disponen de un parking interior, marcado en negro. Las flechas indican la dirección del tráfico a lo largo de las distintas calles. En principio, lo técnicos planeaban no incluir calles de doble dirección para no congestionar el tráfico. Nuestro objetivo es por tanto analizar si el diseño del tráfico se ha realizado de forma correcta, de manera que desde cada punto del nuevo barrio sea posible llegar a todos los demás, tanto a pie como en vehículo.

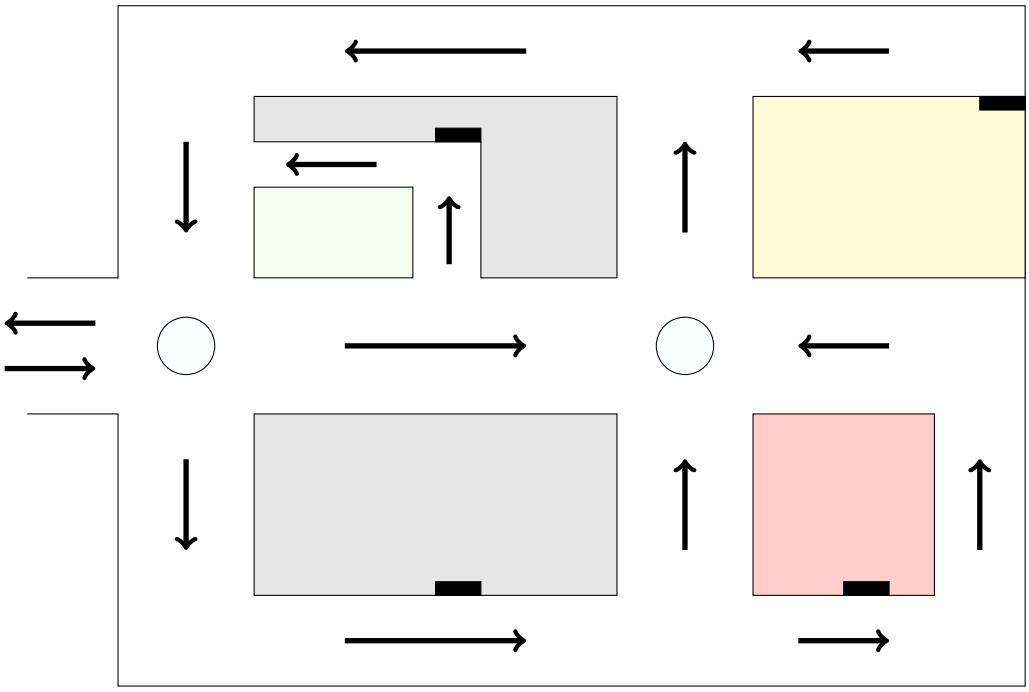

Figura 5.4: Distribución del tráfico en el nuevo barrio.

Para ello, vamos a expresar un problema de conectividad de un grafo. Lo primero que hemos de hacer es expresar nuestro plano como un grafo. Lo que haremos será definir un vértice para cada cruce, y entre dos vértices "consecutivos" definiremos un arco de acuerdo a la dirección del tráfico. En la Figura 5.5 hemos representado el mismo plano en el que hemos asignado un vértice a cada cruce.

A continuación, los arcos vendrán dados por las direcciones de tráfico. Por ejemplo, desde el cruce 8 se puede ir solamente al cruce 2, mientras que desde el cruce 10 se puede ir tanto al cruce 13 (si se continúa recto) como al cruce 9, la rotonda, si se gira a la izquierda. Con esta interpretación, y teniendo en cuenta que en este ejemplo los arcos no llevan asociado un peso, para introducir el grafo en MatLab bastará con introducir la matriz de adyacencia. Para definirla, comencemos definiendo una matriz de ceros y a continuación añadiremos los arcos:

```
>> A=zeros(13);
```

a) Desde el vértice 1, que podemos interpretar como el resto de la ciudad, podemos volver a 1, o adentrarnos en el barrio a través de la rotonda nombrada como cruce 4:

```
>> A(1,1)=1;
>> A(1,4)=1;
```

b) Desde el cruce 2 continuamos hasta el cruce 3:

```
>> A(2,3)=1;
```

c) Desde el cruce 3 únicamente podemos ir al cruce 4:

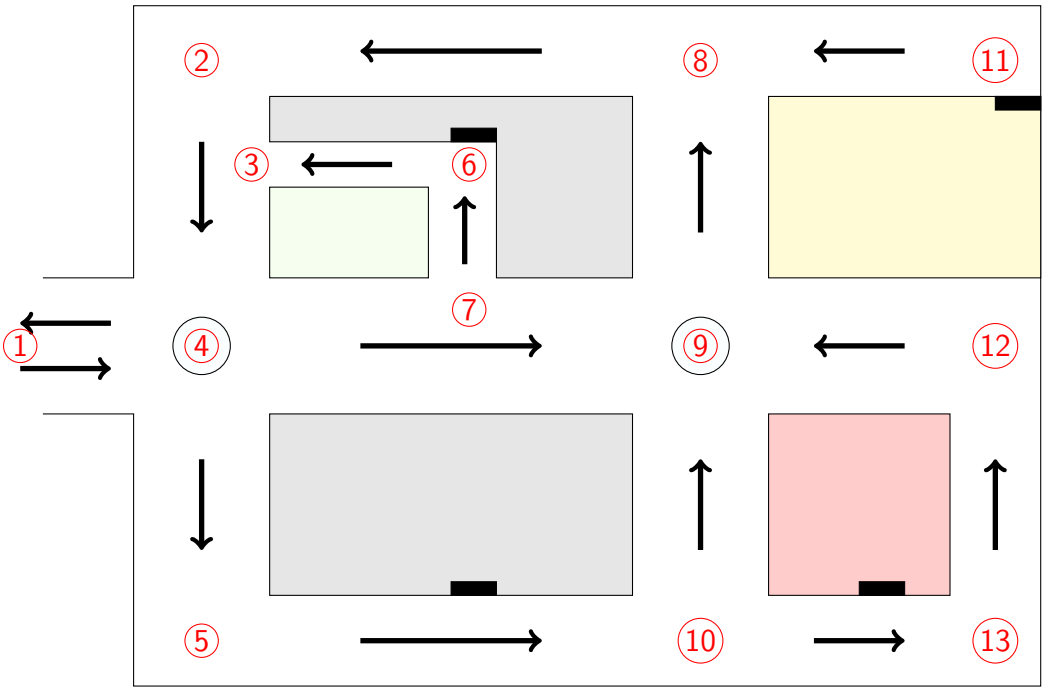

Figura 5.5: Definición de los vértices del grafo.

```
>> A(3,4)=1;
```

d) Desde la rotonda numerada como 4 podemos ir tanto al centro de la ciudad (vértice 1), como a los cruces 5 y 7:

```
>> A(4,1)=1;
>> A(4,5)=1;
>> A(4,7)=1;
```

e) Desde el cruce 5 solamente podemos ir hasta el cruce 10:

```
>> A(5,10)=1;
```

f) Desde el sexto vértice solamente podemos ir al tercer cruce:

```
>> A(6,3)=1;
```

g) Desde el cruce 7 podemos bien ir al cruce 6 o bien ir a la rotonda numerada como 9:

```
>> A(7,6)=1;
>> A(7,9)=1;
```

h) El vértice 8 tiene dirección única hasta el vértice 2:

```
>> A(8,2)=1;
```

i) Desde la rotonda 9 solamente podemos ir al cruce 8:

```
>> A(9,8)=1;
```

j) Desde el cruce 10 podemor ir tanto a la rotonda 9 como al cruce 13:

```
>> A(10,9)=1;
>> A(10,13)=1;
```

k) Desde el cruce 11, donde está situada la cochera de las instalaciones deportivas, podemos ir al cruce 8:

```
>> A(11,8)=1;
```

l) Desde el vértice 12 la única opción es entrar en la rotonda 9:

```
>> A(12,9)=1;
```

m) Desde el cruce 13 se debe de ir al cruce 12:

```
>> A(13,12)=1;
```

Dado que en este ejemplo cada arco no tiene asociado un peso, con la matriz de adyacencia podemos definir el grafo en MatLab:

```
>> G=digraph(A)

G =

  digraph with properties:

     Edges: [18x2 table]
     Nodes: [13x0 table]
```

Nos indica que tenemos 13 vértices (cruces) y 18 arcos (direcciones). Este grafo se puede representar gráficamente con la instrucción:

```
>> h=plot(G)
```

Obteniendo el dibujo de la Figura 5.6. Evidentemente, la representación gráfica realizada por MatLab es distinta de la representación en plano hecha por los técnicos del ayuntamiento, pero en términos de grafos la información que contiene es la misma.

Nuestro objetivo es estudiar si la planificación urbanística es correcta, es decir, si desde cada punto del barrio es posible acceder a todos los demás cruces tanto a pie como en vehículo. En términos de grafos, este problema se traduce en estudiar la conexión del grafo `G`. En el primer caso, un peatón a pie puede ir siempre en ambas direcciones, así que debemos de analizar si el grafo es débilmente conexo (cada arco se vuelve una arista y nos "olvidamos" de las direcciones):

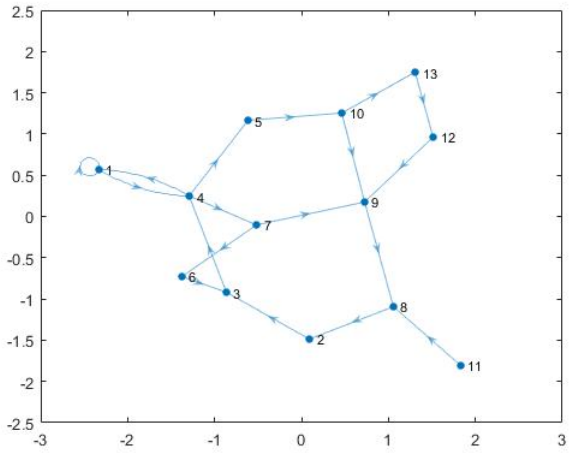

Figura 5.6: Representación gráfica de `MatLab` del grafo `G` correspondiente a la planificación urbanística.

```
>> conncomp(G,'Type','weak')

ans =
     1     1     1     1     1     1     1     1     1     1
     1     1     1
```

Esto significa que solamente hay una componente (débilmente) conexa, y por tanto el grafo es débilmente conexo. Concluimos que, a pie, la planificación urbanística es correcta.

Pasemos a analizar si la planificación en vehículo es también correcta. En este caso sí debemos de tener en cuenta las direcciones, y por lo tanto tendremos que ver si el grafo es conexo:

```
>> conncomp(G)

ans =
     2     2     2     2     2     2     2     2     2     2
     1     2     2
```

Vemos que en este caso hay dos componentes conexas: una de ellas formada únicamente por el cruce 11, y la segunda formada por el resto de cruces. Esto significa que, si no tenemos el cruce 11 en cuenta, es posible desplazarse entre cualquier par de puntos del barrio. Ahora bien, el problema surge con el cruce 11, que recordemos es el parking de las instalaciones deportivas. Vamos a calcular los predecesores y sucesores de dicho vértice para analizar la situación:

```
>> successors(G,11)

ans =
```

```
    8
>> predecessors(G,11)

ans =

    Empty matrix: 0-by-1
```

Vemos por tanto que desde el vértice 11 sí es posible ir al vértice 8, y consecuentemente a todos los demás, pero el vértice 11 no tiene ningún predecesor, es decir, no hay ninguna punto del barrio desde el cual sea posible de llegar al parking de las instalaciones deportivas.

¿Cómo podemos solventar este problema? A pesar de que los técnicos deseaban no incluir calles de doble sentido, la solución pasa por permitir que en la calle que une los cruces 8 y 11 sí se pueda cirular en dirección contraria. Vamos por tanto a añadir a nuestro grafo el vértice $8 \longrightarrow 11$ y estudiemos si con esta nueva dirección el grafo modificado es conexo:

```
>> G2=addedge(G,8,11,1)

G2 =

  digraph with properties:

    Edges: [19x2 table]
    Nodes: [13x0 table]
```

Con esta última instrucción hemos añadido el arco desde el vértice 8 hasta el vértice 11, asignándole a este arco un peso de 1, que es el peso que MatLab considera por defecto cuando no añadimos un peso. Veamos a continuación si este grafo modificado es conexo:

```
>> conncomp(G2)

ans =
     1     1     1     1     1     1     1     1     1     1
     1     1     1
```

En esta ocasión, vemos que solamente hay una componente conexa, y por tanto concluimos que, ahora sí, el grafo es conexo y por consiguiente el tráfico estaría bien planteado: es posible viajar en vehículo entre cada par de puntos del barrio.