Modelización matemática con grafos en Maxima

Esperanza López Centella

SUMA+ núm. 99
pp. 73-82
Artículo solicitado por *Suma* en diciembre de 2021 y aceptado en febrero de 2022

En el marco del programa ESTALMAT, en este artículo presentamos una propuesta didáctica para el trabajo de la modelización matemática con grafos en el sistema de álgebra computacional Maxima (López, 2018). Se muestran distintas formas de definir un grafo en Maxima así como una variedad de nociones sobre grafos susceptibles de ser exploradas a través de dicho sistema en una sesión de enriquecimiento curricular. Se plantean actividades contextualizadas cuya resolución se simplifica a través de la construcción de modelos basados en grafos.

En efecto, los grafos y la teoría desarrollada en torno a ellos constituyen una poderosa herramienta matemática mediante la cual, a través de la programación, se modelizan gran cantidad de situaciones de nuestra vida cotidiana. En consonancia con esto, el currículo español de Educación Secundaria Obligatoria y Bachillerato (MECD, 2014) incluye una introducción al cálculo matricial en varias materias de Matemáticas, y al uso de lenguajes de programación y diseño de programas en distintas materias de Tecnología. Asimismo, el reciente informe sobre bases para la elaboración de un currículo de matemáticas en educación preuniversitaria del Comité Español de Matemáticas (Calvo y otros, 2021) refiere explícitamente al uso de grafos para la resolución de problemas, recalcando su importancia en la actividad de modelización matemática.

Maxima es un sistema de álgebra computacional de alta precisión para la manipulación de expresiones simbólicas y numéricas. Está basado en el sistema original de Macsyma desarrollado por el Instituto Tecnológico de Massachusetts en los años 70. Puede graficar funciones y datos en dos y tres dimensiones. El código fuente de Maxima puede ser compilado en varios sistemas operativos como Windows, Linux y MacOS X. El código fuente para todos los sistemas y los binarios precompilados para Windows y

Linux están disponibles *online* en el Administrador de archivos de SourceForge. Maxima funciona en modo consola (interfaz de línea de comandos), aunque también cuenta con las interfaces gráficas xMaxima y wxMaxima (entre otras opciones) para facilitar su uso. En la presente propuesta trabajamos con wxMaxima.

Definición de un grafo en Maxima

Para trabajar con grafos en Maxima hemos de cargar el paquete «graphs»:

```
→ load(graphs)$
```

Antes de abordar otras actividades, conviene revisar las distintas formas de definir grafos en este sistema. A continuación estudiamos algunas de ellas.

GRAFO A PARTIR DE SUS VÉRTICES Y SUS LADOS

La función create_graph nos permite definir un grafo a partir de las dos listas que recibe como argumento: la lista de sus vértices y la lista de sus lados. Nótese que cada lado es, a su vez, una lista (de longitud 2) de los vértices que une. La siguiente línea de comando crea un triángulo (grafo completo de tres vértices), denominado *C3*.

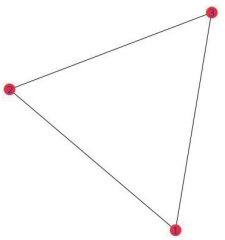
```
→ C3:create_graph([1,2,3],[[1,2],[2,3],[3,1]]);
Output:
GRAPH\(3 vertices, 3 edges\)
```

Mediante el uso de la función print_graph obtenemos información del grafo C3:

```
→print_graph(C3);
Output:
Graph on 3 vertices with 3 edges.
Adjacencies:
  1 : 3  2
  2 : 3  1
  3 : 1  2
```

Para representar gráficamente un grafo podemos emplear la función draw_graph. Si activamos la opción show_id en la función se mostrarán las etiquetas de los vértices:

```
→ draw_graph(C3, show_id=true);
Output:
```

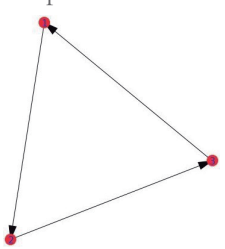


Usando la opción directed en la función create_graph, podemos obtener una versión dirigida del grafo anterior:

```
→ T3:create_graph([1,2,3],[[1,2],[2,3],[3,1]],
directed=true);
Output:
DIGRAPH\(3 vertices, 3 arcs\)

→ print_graph(T3);
Output:
Digraph on 3 vertices with 3 arcs.
Adjacencies:
  1 : 2
  2 : 3
  3 : 1

→ draw_graph(T3, directed=true,show_id=true);
Output:
```



GRAFOS A PARTIR DE SU MATRIZ DE ADYACENCIA

Otra de las posibilidades que ofrece Maxima para crear un grafo es hacerlo a partir de su matriz de adyacencia mediante la función from_adjacency_matrix. Las siguientes líneas de comandos crean y dibujan el grafo completo de 4 vértices.

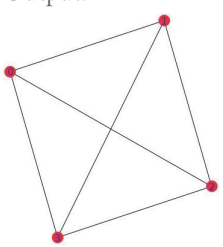
→ A:matrix([0,1,1,1],[1,0,1,1],[1,1,0,1],[1,1,1,0]);
Output:

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

→C4: from_adjacency_matrix(A);
Output:
GRAPH(4 vertices, 6 edges\)

→ print_graph(C4);
Output:
Graph on 4 vertices with 6 edges.
Adjacencies:
3 : 2 1 0
2 : 3 1 0
1 : 3 2 0
0 : 3 2 1

→ draw_graph(C4,show_id=true);
Output:



GRAFOS A PARTIR DE SUS VÉRTICES Y UNA FUNCIÓN BOOLEANA QUE DETERMINA SUS LADOS

Maxima también permite definir un grafo a partir de un conjunto de vértices V y una función booleana de dos variables definida sobre el producto cartesiano $V \times V$. Esta función determina las conexiones entre los vértices (es decir, las aristas), considerando arista del grafo aquella cuyos vértices satisfacen la condición de la función booleana. Las siguientes líneas de comandos muestran un ejemplo de esto, donde para expresar la condición en cuestión se han utilizado la función booleana **is** (devuelve *true* si la condición de su argumento es satisfecha y *false* en caso contrario) y la función módulo **mod** (devuelve el resto de la división de su primer argumento entre su segundo argumento).

→ b(u,v):=is(mod(u+v,3)=0)$

→ G:make_graph(makelist(i,i,0,9),b);
Output:
GRAPH\(10 vertices, 15 edges\)

→ print_graph(G);
Output:
Graph on 10 vertices with 15 edges.
Adjacencies:
0 : 3 6 9
1 : 2 5 8
2 : 1 4 7
3 : 0 6 9
4 : 2 5 8
5 : 1 4 7
6 : 0 3 9
7 : 2 5 8
8 : 1 4 7
9 : 0 3 6

Como vemos, el recién creado grafo G consta de 10 vértices, de forma que los vértices u y v están unidos por una arista si y solo si la suma $u+v$ módulo 3 es igual a 0 (es decir, si dicha suma es múltiplo de 3).

GRAFOS PREDEFINIDOS EN MAXIMA

Maxima cuenta con funciones predefinidas de algunos tipos de grafos. A continuación se listan algunos ejemplos.

— **complete_graph(n)**: crea el grafo completo de n vértices.
— **complete_bipartite_graph(m,n)**: define el grafo bipartito completo con particiones de m y n vértices.
— **complement_graph(G)**: establece el grafo complementario del grafo G.
— **path_graph(n)**: crea un grafo que es un camino de longitud $n-1$ (con n vértices).
— **path_digraph(n)**: define un grafo que es un camino dirigido de longitud $n-1$ (con n vértices).
— **cycle_graph(n)**: genera un grafo que es un ciclo de n vértices.
— **cycle_digraph(n)**: produce un grafo dirigido que es un ciclo de n vértices.
— **cube_graph(n)**: crea el grafo correspondiente al n-cubo (hipercubo n-dimensional).

La figura 1 muestra una representación gráfica de grafos definidos por algunas de las funciones anteriores.

Nociones sobre grafos en Maxima

Una vez definido un grafo en Maxima, podemos utilizar distintas funciones para obtener información acerca de este. He aquí algunas de ellas.

— **vertices(G)**: devuelve la lista de vértices del grafo *G*.
— **edges(G)**: proporciona la lista de aristas del grafo *G*.
— **remove_edge([u,v],G)**: elimina la arista [u,v] del grafo *G*.
— **induced_subgraph(V,G)**: define el subgrafo de *G* que tiene como conjunto de vértices *V*.
— **vertex_degree(n,G)**: devuelve el grado del vértice *n* del grafo *G*.
— **degree_sequence(G)**: proporciona una lista con los grados de los vértices del grafo *G*.
— **adjacency_matrix(G)**: provee la matriz de adyacencia del grafo *G*.
— **hamilton_cycle(G)**: devuelve un ciclo de Hamilton del grafo *G* o una lista vacía si el grafo *G* no es hamiltoniano.
— **hamilton_path(G)**: facilita una lista secuenciada de vértices que forman un camino de Hamilton en el grafo *G*.
— **shortest_path(u,v,G)**: devuelve el camino más corto entre los vértices *u* y *v* del grafo *G*.
— **is_isomorphic(G,G')**: informa si los grafos *G* y *G'* son isomorfos (por medio de los valores *true* y *false*).
— **isomorphism(G,G')**: devuelve un isomorfismo entre los grafos *G* y *G'* o una lista vacía si estos no son isomorfos.
— **is_bipartite(G)**: informa si el grafo *G* es bipartito (mediante los valores *true* y *false*).
— **bipartition(G)**: proporciona una partición de *G* o una lista vacía si *G* no es bipartito.
— **is_connected(G)**: informa si el grafo *G* es conexo (a través de los valores *true* y *false*).
— **connected_components(G)**: devuelve una lista con las listas de los vértices que forman las componentes conexas de *G*.
— **chromatic_number(G)**: devuelve el número cromático del grafo *G*.
— **vertex_coloring(G)**: proporciona una lista cuyo primer argumento es el número cromático del grafo *G* y su segundo argumento una lista representando una coloración del grafo (indicando el correspondiente «color» de cada vértice mediante un número).

Modelización con grafos

En esta sección construiremos un grafo que represente los posibles movimientos de un caballo de ajedrez en un tablero de juego cuadrado de 4 casillas

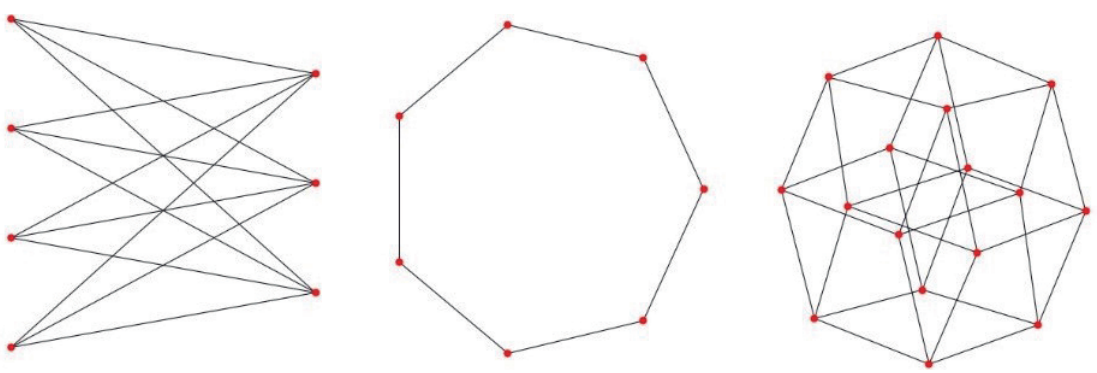


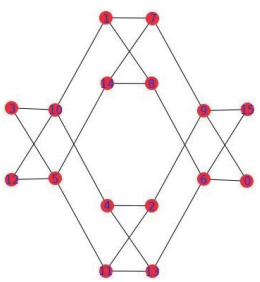
Figura 1. Representaciones gráficas del grafo bipartito completo de 3 y 4 vértices, del grafo cíclico de 7 vértices y del grafo correspondiente al teseracto (cubo tetradimensional en un espacio de cuatro dimensiones espaciales)

por lado (4×4). Para ello, enumeramos de 0 a 15 las 16 casillas del tablero como muestra la figura 2.

A cada casilla asociamos un punto etiquetado con el número de la casilla y lo consideramos como vértice del grafo. Dos vértices están unidos por un lado si están conectados por un movimiento de caballo de ajedrez entre las casillas correspondientes. Por ejemplo, en el tablero de ajedrez representado en la figura 2, el vértice 0 está unido a los vértices 6 y 9; el vértice 1 a los vértices 7, 8 y 10; y así sucesivamente. Las siguientes líneas de comando definen y dibujan el grafo:

```
→ caballo_ajedrez_4×4:create_graph
(makelist(i,i,0,15),[[0,6],[0,9],[1,7],[1,8],[1,10],[2,4],
[2,9],[2,11],[3,5],[3,10],[4,10],[4,13],[5,11],[5,12],
[5,14],[6,8],[6,13],[6,15],[7,9],[7,14],[8,14],[9,15],
[10,12],[11,13]]);
Output:
GRAPH\(16 vertices, 24 edges\)
```

```
→ draw_graph(caballo_ajedrez_4×4,show_id=true);
Output:
```



12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

Figura 2. Tablero de ajedrez 4×4 con casillas enumeradas

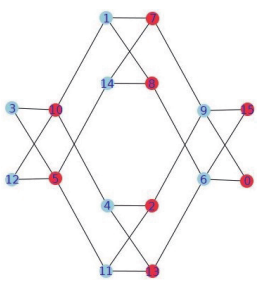
Habida cuenta de que cada casilla de un tablero de ajedrez es de uno de dos colores (típicamente, blanco y negro), que estos colores se manifiestan de forma alterna en las casillas y que en cada movimiento un caballo se desplaza desde una casilla de un color a una casilla del otro color, el número cromático del grafo **caballo_ajedrez_4×4** es 2:

```
→ vertex_coloring(caballo_ajedrez_4×4);
Output:
[2,[[15,2],[14,1],[13,2],[12,1],[11,1],[10,2],[9,1],[8,2],
[7,2],[6,1],[5,2],[4,1],[3,1],[2,2],[1,1],[0,2]]]
```

Es decir, el grafo es bipartito y la siguiente resulta una bipartición del mismo:

```
→ [x,y]:bipartition(caballo_ajedrez_4×4);
Output:
[[14,12,4,11,3,1,6,9],[15,7,2,13,5,10,8,0]]
```

```
→ draw_graph(ajedrez,show_vertices=
x,show_id=true)
Output:
```



Puede comprobarse cómo los vértices de grado 2 se corresponden con las 4 esquinas del tablero, los vértices de grado 4 con las 4 casillas centrales y los vértices de grado 3 son el resto de casillas. Definamos una función que devuelva los vértices de un grado específico *n* en un grafo dado G:

```
→ gr(G,n):=block([V],
f(x):=is(vertex_degree(x,G)=n),
V:setify(vertices(G)),
L:subset(V,f),
return(L));
```

Nótese que en la definición de la función se han usado las funciones auxiliares **setify** para convertir una lista en conjunto, y **subset**, para obtener los elementos de un conjunto que satisfacen una función booleana. Aplicando nuestra recién definida función sobre el grafo **caballo_ajedrez_4×4**, obtenemos:

```
→ gr(caballo_ajedrez_4×4,2);
Output:
{0,3,12,15}

→ gr(caballo_ajedrez_4×4,3);
Output:
{1,2,4,7,8,11,13,14}

→ gr(caballo_ajedrez_4×4,4);
Output:
{5,6,9,10}
```

No es posible para un caballo de ajedrez recorrer el tablero sin repetir casilla y regresar a la casilla inicial:

```
→ hamilton_cycle(caballo_ajedrez_4×4);
Output:
[]
```

Tampoco lo es omitiendo la condición de regreso a la casilla inicial:

```
→ hamilton_path(caballo_ajedrez_4×4);
Output:
[]
```

Actividades

Los movimientos del caballo de ajedrez

Construir un grafo que represente los movimientos de un caballo de ajedrez en un tablero de ajedrez generalizado $m \times n$ para cualesquiera números naturales m y n.

A continuación se sugieren algunos pasos orientativos para el alumnado en la realización de esta actividad.

1. Numerar las casillas del tablero. En el tablero representado en la figura 3, estas han sido numeradas de 0 a $mn-1$, de izquierda a derecha y de abajo a arriba. A cada casilla se asigna un punto y se etiqueta con el correspondiente número de casilla. Estos puntos constituyen el conjunto de vértices del grafo solicitado.
2. Definir una función booleana **caballo** que, aplicada a dos valores x e y, devuelva *true* si la casilla x puede ser conectada mediante un movimiento de caballo de ajedrez con la casilla y, y *false* en caso contrario. Para ello resulta de utilidad escribir en función de x los valores de las posibles casillas destino con origen en x (figura 3): $y_{1,x} = x-2+n$, $y_{2,x} = x-1+2n$, $y_{3,x} = x+1+2n$, $y_{4,x} = x+2+n$, $y_{5,x} = x-2-n$, $y_{6,x} = x-1-2n$, $y_{7,x} = x+1-2n$, $y_{8,x} = x+2-\mathrm{n}$. Para que x esté conectado con y es necesario que y sea alguno

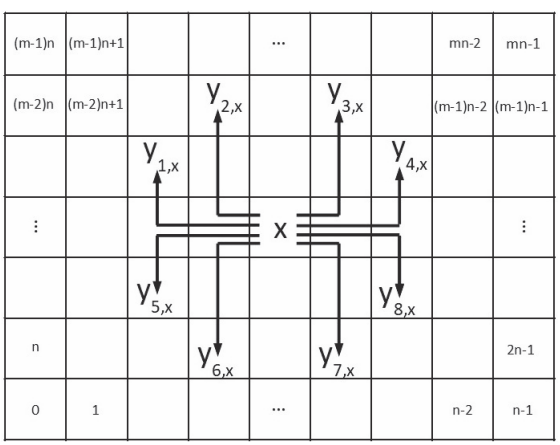


Figura 3. Representación de un tablero de ajedrez $m \times n$ numerado

de esos valores. Pero esta condición podría no ser suficiente: la posición de x también puede añadir restricciones. Si x no fuera ningún valor de las casillas del interior del tablero (representadas en color blanco en la figura 4), entonces algunos de esos valores $y_{i,x}$ podrían no tener sentido.

Para comprobar la posición de x en el tablero, podemos ayudarnos de la numeración del tablero, la función **mod** y las relaciones numéricas, tal y como se muestra en la tabla 1. Naturalmente, si x está en la intersección de dos de las bandas distinguidas en la tabla 1, sus posibles casillas destino son aquellas en común entre las correspondientes de estas dos bandas.

La función **caballo** debería incluir un conveniente diseño de filtros para identificar el argumento x en esta casuística, y entonces comparar el argumento y con los correspondientes valores $y_{i,x}$ según el caso. Si como fruto de esta comparación la función **caballo** detecta una igualdad, debería devolver *true*; en otro caso, debería devolver *false*.

3. A partir de la función **caballo**, definir la función **caballo_ajedrez** que, aplicada a cualesquiera dos números naturales m y n, devuelva el grafo de $m{\cdot}n$ vértices tal que dos vértices del grafo están unidos por una arista si y solo si pueden ser conectados a través de un movimiento de caballo de ajedrez. Este grafo podría ser definido aprovechando lo descrito en la sección anterior dedicada a los grafos a partir de sus vértices.

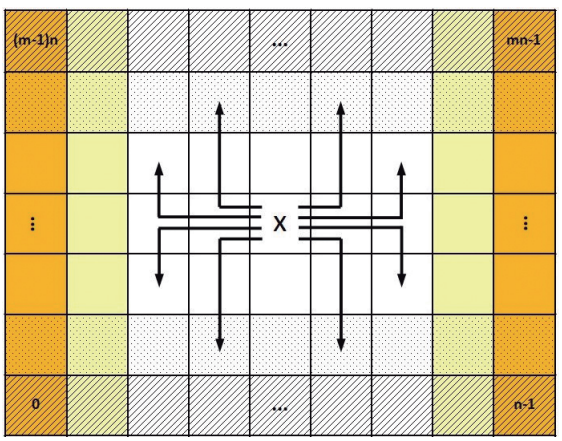


Figura 4. Zonas límite (naranja, amarilla, rayada y punteada) para los movimientos de un caballo de ajedrez

Posición de x	*Condición*	*Posibles casillas destino con origen x*
Banda naranja izquierda	$\mathrm{mod}(x,n)=0$	$y_{3,x}, y_{4,x}, y_{7,x}, y_{8,x}$
Banda naranja derecha	$\mathrm{mod}(x,n)=n-1$	$y_{1,x}, y_{2,x}, y_{5,x}, y_{6,x}$
Banda amarilla izquierda	$\mathrm{mod}(x,n)=1$	$y_{2,x}, y_{3,x}, y_{4,x}, y_{6,x}, y_{7,x}, y_{8,x}$
Banda amarilla derecha	$\mathrm{mod}(x,n)=n-2$	$y_{1,x}, y_{2,x}, y_{3,x}, y_{5,x}, y_{6,x}, y_{7,x}$
Banda rayada superior	$n\leq mn-1-x\leq 0$	$y_{5,x}, y_{6,x}, y_{7,x}, y_{8,x}$
Banda rayada inferior	$0\leq x\leq n-1$	$y_{1,x}, y_{4,x}, y_{5,x}, y_{6,x}, y_{7,x}, y_{8,x}$
Banda punteada superior	$2n\leq mn-1-x\leq n$	$y_{1,x}, y_{2,x}, y_{3,x}, y_{4,x}, y_{5,x}, y_{8,x}$
Banda punteada inferior	$n\leq x\leq 2n-1$	$y_{1,x}, y_{2,x}, y_{3,x}, y_{4,x}$

Tabla 1. Relaciones entre posición de x y condición satisfecha por x en un tablero $m\times n$ numerado como en la figura 3

Organización de equipos

A una prueba de duatlón acude un grupo de diez amigos deportistas, cinco de ellos atletas: Alicia (A), Berta (B), Celia (C), Darío (D) y Eloy (E); y cinco de ellos ciclistas: Felipe (F), Gema (G), Hugo (H), Ignacio (I) y Julia (J). Cada equipo participante en la prueba ha de estar formado por un atleta y un ciclista. De acuerdo con las fortalezas y debilidades de cada uno de los deportistas, consideran lo siguiente para formar equipos óptimos: A debe asociarse con F, G, H; B debe asociarse con G, I; C debe asociarse con F, G; D debe asociarse con G, I ,J; E debe asociarse con F, G, H.

a. Definir y representar gráficamente el grafo que modeliza la situación anterior, representando cada persona por un vértice.

b. ¿Es posible conformar cinco equipos óptimos para competir a la vez en la prueba? En caso afirmativo, indicar tales equipos. En caso contrario, encontrar el número máximo de equipos posibles cumpliendo las condiciones planteadas.

c. ¿Es posible la situación descrita en el punto (b) anterior si Darío forma equipo con Ignacio? En caso afirmativo, especificar el resto de equipos. En caso contrario, hallar el número máximo de equipos posibles satisfaciendo las condiciones planteadas.

d. Al grupo se incorporan tres nuevos atletas: Lucía (L), Marta (M) y Norberto (N), y tres nuevos ciclistas: Omar (O), Paula (P) y Quintín (Q) quedando el esquema del siguiente modo: A debe asociarse con F, G, H, O; B debe asociarse con G, I; C debe asociarse con F, G, O; D debe asociarse con G, I, J; E debe asociarse con F, G, H, O, P, Q; L debe asociarse con I, O; M debe asociarse con J; N debe asociarse con G, I, J, O. Resolver las tres cuestiones anteriores en esta situación.

e. ¿Cuál es el número mínimo de pruebas de duatlón precisas para que cada uno de los ocho atletas se asocie con todos y cada uno de los ciclistas con quienes forma un equipo óptimo?

Como se indica en el propio enunciado de la actividad, la situación descrita puede ser modelizada por medio de un grafo cuyos vértices son representados por cada uno de los deportistas, de forma que dos vértices están conectados por una arista si los dos deportistas correspondientes forman un equipo óptimo. En Maxima podríamos definir el grafo asociado mediante el uso de la función create_graph y representarlo gráficamente a partir de draw_graph. Efectivamente, con las condiciones impuestas, puede proporcionarse un emparejamiento completo en el grafo (bipartito) definido. En el caso de que Darío únicamente forme equipo con Ignacio, la situación queda modelizada por un nuevo grafo en que D solo tiene adyacencia con I. Este nuevo grafo no admite un emparejamiento completo, siendo 4 el máximo número de equipos óptimos posibles. Se puede proceder análogamente para razonar sobre la situación con las nuevas incorporaciones. Por último, para conocer el mínimo número de pruebas de duatlón precisas para que cada atleta se asocie con todos y cada uno de los ciclistas con quienes forma un equipo óptimo, recurrimos a una arista coloración del grafo y a su índice cromático.

Planificación de exámenes

Cada estudiante de un grupo de clase ha de realizar un examen de cada una de las materias en que está matriculado. Por normativa de evaluación, el examen de una materia ha de ser realizado a la vez por todo el alumnado matriculado; y el alumnado tiene derecho a no realizar dos exámenes el mismo día. Las materias (13) son: Arquitectura, Biología, Ciencias Naturales, Dibujo, Economía, Francés, Geografía, Historia, Inglés, Jardinería, Lengua y Literatura, Matemáticas y Neurología. Todo el alumnado está matriculado en las materias de Matemáticas y de Lengua y Literatura. Además, tienen alumnado en común las materias de Arquitectura y Dibujo; las de Biología y Ciencias Naturales; las de Biología y Neurología; las de Ciencias Naturales y Neurología; las de Economía e Historia; las de Francés e Inglés. ¿Cuál es la mínima cantidad de días de examen necesarios para satisfacer estas condiciones?

Podemos modelizar esta situación a través de un grafo cuyos vértices son las distintas materias, de manera que exista una arista de uno a otro vértice si en sus correspondientes materias hay alumnado matriculado en común. El color asignado a cada arista representará un día, de forma que el número cromático del grafo da respuesta a la cuestión del problema, es decir, informa de la cantidad mínima

de días de examen necesarios para satisfacer las condiciones de la normativa de evaluación. He aquí la correspondencia del modelo con el problema: dos materias (vértices) no deben celebrar su examen el mismo día (tener el mismo color asignado) si hay alumnado en común matriculado en ambas (son adyacentes, es decir, están conectados por una misma arista).

Glosario

En este apéndice se listan las definiciones de algunas nociones sobre grafos mencionadas en el texto (Burgos y otros, 2016; Deo, 1974).

Grafo: un grafo G es una terna (V, E, f_G), donde V y E son conjuntos y f_G una aplicación $E \to V \times V$. A V se le llama conjunto de vértices, a E conjunto de lados o aristas y a f_G aplicación de incidencia (que informa de las conexiones por aristas de los vértices).

Grafo dirigido (o *digrafo*): es aquel grafo cuyas aristas tienen un sentido definido de forma que las relaciones entre sus vértices no son simétricas como en un grafo general. Un grafo dirigido queda definido por sus conjuntos V y E de vértices y aristas respectivamente y dos aplicaciones $s, t: E \to V$ (*source* y *target*, que indican el origen y el destino de cada arista).

Grafo completo: grafo donde cada par de vértices está conectado por una arista.

Grafo complementario de un grafo G: grafo formado por el mismo conjunto de vértices que el grafo G y tal que entre cualesquiera dos vértices existe una arista si y solo si no la existe en el grafo G.

Grafo conexo: aquel en que existe al menos un camino entre cualesquiera dos de sus vértices.

Matriz de adyacencia: la matriz de adyacencia de un grafo con conjunto de vértices $V = \{v_1, v_2, \ldots, v_n\}$ es aquella matriz cuadrada de orden n cuyo coeficiente (i, j) es igual al número de aristas que unen v_i con v_j.

Grafos isomorfos: se dice que dos grafos $G = (V, E, f_G)$ y $G' = (V', E', f_{G'})$ son isomorfos si existen dos biyecciones $h_V: V \to V'$ y $h_E: E \to E'$ tales que para cada arista $e \in E$ se verifica que $f_{G'}(h_E(e)) = \{h_V(u), h_V(v)\}$ donde $\{u, v\} = f_G(e)$. En tal caso se dice que las aplicaciones h_V y h_E forman un isomorfismo de G a G'.

Grado de un vértice: se define el grado de un vértice v de un grafo G como el número de lados (no lazos) de G que son incidentes en v más 2 veces el número de lazos incidentes en v.

Grafo bipartito: aquel grafo $G = (V, E, f_G)$ tal que es posible descomponer V en dos subconjuntos disjuntos V_1 y V_2 de forma que todo lado de G incide en un vértice de V_1 y en un vértice de V_2.

Grafo bipartito completo: aquel grafo $G = (V, E, f_G)$ bipartito tal que para cada $v_1 \in V_1$ y $v_2 \in V_2$ (siendo $V = V_1 \cup V_2$ y $V_1 \cap V_2 = \varnothing$) existe $e \in E$ tal que $f_G(e) = \{v_1, v_2\}$.

Vértice coloración de un grafo: asignación de colores a los vértices de un grafo tal que dos vértices que compartan la misma arista tengan colores diferentes. Así, una coloración del grafo $G = (V, E, f_G)$ es una aplicación $c: V \to C$ donde C es un conjunto, de tal forma que para cualquier $e \in E$, si $f_G(e) = \{u, v\}$ con $u \neq v$ entonces $c(u) \neq c(v)$.

Arista coloración de un grafo: asignación de colores a las aristas de un grafo tal que las aristas incidentes en el mismo vértice tengan un color distinto.

Número cromático de un grafo: número mínimo de colores necesarios para una vértice coloración del grafo.

Índice cromático de un grafo: número mínimo de colores necesarios para una arista coloración del grafo.

Camino de longitud n: en un grafo $G = (V, E, f_G)$ es una sucesión de lados $e_1 e_2 \ldots e_n$ tales que $f_G(e_i) = \{v_{i-1}, v_i\}$ para una sucesión de vértices $v_0 v_1 \ldots v_n$. En tal caso

se dice que el camino $e_1 e_2 \ldots e_n$ es un camino del vértice v_0 al vértice v_n.

Ciclo: camino en que no aparecen lados repetidos ni vértices repetidos (salvo el primero y el último) en el que el primer vértice coincide con el último.

Camino de Hamilton: camino que recorre todos los vértices de un grafo una sola vez.

Agradecimientos

Agradezco sinceramente el apoyo de los compañeros del Departamento de Álgebra de la Universidad de Granada —y, en particular, el de Jesús García Miranda— quienes compartieron conmigo parte del material que inspira este trabajo, así como el apoyo del grupo de investigación FQM-193 «Pensamiento Numérico – Didáctica de la Matemática» del que soy miembro.

Referencias bibliográficas

Burgos, M., J. García, P. A. García y J. C. Rosales (2016), *Matemática Discreta y Álgebra Lineal*, Universidad de Granada, Granada.

Calvo, C., A. Carrillo de Albornoz, A. de La Fuente, M. de León, M. J. González, A. Gordaliza, I. Guevara, C. Lázaro, O. Monzó, A. J. Moreno, J. L. Rodríguez, J. Rodríguez y A. Serradó (2021), *Bases para la elaboración de un currículo de Matemáticas en Educación no Universitaria,* Comité Español de Matemáticas.

Deo, N. (1974), *Graph theory with applications to enginnering and computer sciences*, Prentice-Hall, New Jersey.

López, E. (2018), «Trabajando grafos con Maxima», en F. España (editor), *Actas del XVII Congreso de Enseñanza y Aprendizaje de las Matemáticas. Matemáticas en tierra de cine*, Universidad de Almería, Almería.

Real Decreto 1105/2014, de 26 de diciembre, por el que se establece el currículo básico de la Educación Secundaria Obligatoria y del Bachillerato, BOE.

Esperanza López Centella
Universidad de Granada
<esperanza@ugr.es>