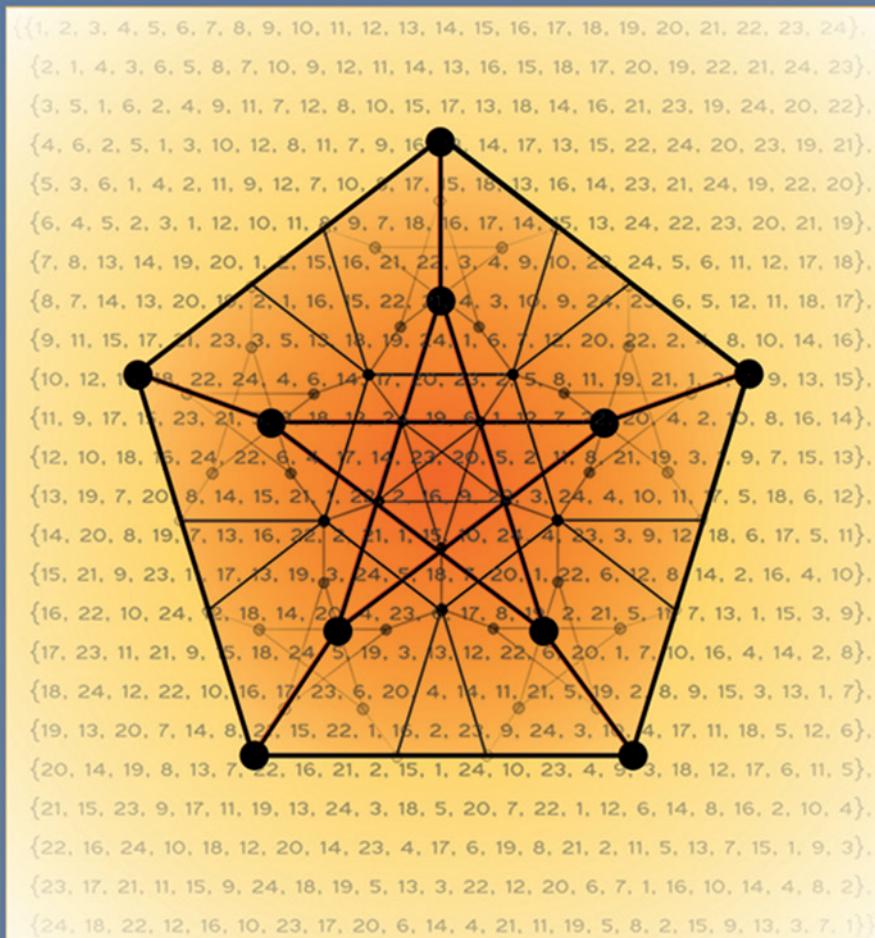


Métodos computacionales en álgebra

Matemática discreta: grupos y grafos
(2^a Edición Revisada)



Juan Francisco Ruiz Ruiz



UNIVERSIDAD DE JAÉN

METODOS COMPUTACIONALES EN ÁLGEBRA

Matemática discreta: grupos y grafos

2^a Edición Revisada

METODOS COMPUTACIONALES EN ÁLGEBRA

Matemática discreta: grupos y grafos

2^a Edición Revisada

Juan Francisco Ruiz Ruiz



UNIVERSIDAD DE JAÉN

© Autores
© Universidad de Jaén

DISEÑO Y MAQUETACIÓN
Servicio de Publicaciones

ISBN
978-84-8439-754-0

DEPÓSITO LEGAL
J-415-2013

COLECCIÓN
Techné, 39

EDITA
Publicaciones de la Universidad de Jaén
Vicerrectorado de Extensión Universitaria, Deportes y Proyección Institucional
Campus Las Lagunillas, Edificio Biblioteca
23071 Jaén (España)
Teléfono 953 212 355 – Fax 953 212 235
servpub@ujaen.es

“Cualquier forma de reproducción, distribución, comunicación pública o transformación de esta obra solo puede ser realizada con la autorización de sus titulares, salvo excepción prevista por la ley. Diríjase a CEDRO (Centro Español de Derechos Reprográficos, www.cedro.org) si necesita fotocopiar, escanear o hacer copias digitales de algún fragmento de esta obra”.

A Juan Claudio

CONTENIDOS

| | |
|--|-----|
| CONTENIDOS | V |
| PRÓLOGO | IX |
| 1. MATHEMATICA Y HERRAMIENTAS BÁSICAS DE PROGRAMACIÓN | 13 |
| 1. MATHEMATICA | 13 |
| 2. PROGRAMACIÓN | 15 |
| 3. OTRAS FUNCIONES | 19 |
| 4. LA VERSIÓN DE MATHEMATICA 5.2..... | 29 |
| 5. EJERCICIOS | 30 |
| 2. GRUPOS..... | 33 |
| 1. GRUPOS FINITOS | 34 |
| 1.1. OPERACIÓN INTERNA | 35 |
| 1.2. ELEMENTO NEUTRO..... | 37 |
| 1.3. ELEMENTO SIMÉTRICO..... | 39 |
| 1.4. ASOCIATIVA | 43 |
| 1.5. CONMUTATIVA..... | 44 |
| 1.6. TEST PARA GRUPOS FINITOS..... | 45 |
| 2. HOMOMORFISMOS DE GRUPOS | 45 |
| 3. OTROS EJEMPLOS Y GRUPOS INFINITOS..... | 53 |
| 3.1. GRUPOS DE ORDEN PEQUEÑO..... | 60 |
| 4. EJERCICIOS | 74 |
| 3. SUBGRUPOS, EL GRUPO COCIENTE Y GENERADORES..... | 81 |
| 1. SUBGRUPOS | 81 |
| 2. CLASES LATERALES. EL TEOREMA DE LAGRANGE..... | 85 |
| 3. SUBGRUPOS NORMALES Y GRUPOS COCIENTES..... | 87 |
| 4. CÁLCULO DE TODOS LOS SUBGRUPOS DE UN GRUPO FINITO | 91 |
| 5. SUBGRUPOS GENERADOS..... | 96 |
| 5.1. SUBGRUPO GENERADO POR UN ELEMENTO: EL ORDEN DE UN ELEMENTO | 96 |
| 5.2. SUBGRUPO GENERADO POR UN SUBCONJUNTO CUALQUIERA..... | 99 |
| 6. EFICACIA Y OPTIMIZACIÓN EN EL CÁLCULO DE SUBGRUPOS..... | 101 |
| 7. OTROS EJEMPLOS. CASO INFINITO | 134 |
| 8. EFICACIA Y OPTIMIZACIÓN EN EL CÁLCULO DE GRUPOS DE ORDEN PEQUEÑO..... | 135 |
| 9. EJERCICIOS | 142 |

| | |
|---|------------|
| 4. EL GRUPO SIMÉTRICO | 149 |
| 1. PERMUTACIONES | 149 |
| 1.1. PRIMER ALGORITMO PARA EL CÁLCULO DE PERMUTACIONES..... | 150 |
| 1.2. SEGUNDO ALGORITMO PARA EL CÁLCULO DE PERMUTACIONES | 152 |
| 1.3. TERCER ALGORITMO PARA EL CÁLCULO DE PERMUTACIONES | 154 |
| 1.4. FUNCIÓN DE MATHEMATICA | 155 |
| 1.5. EFICACIA Y OPTIMIZACIÓN..... | 157 |
| 2. EL GRUPO SIMÉTRICO | 159 |
| 2.1. IMPLEMENTACIÓN DE PERMUTACIONES | 160 |
| 2.2. COMPOSICIÓN O MULTIPLICACIÓN DE PERMUTACIONES | 163 |
| 2.3. CICLOS Y DESCOMPOSICIÓN EN PRODUCTO DE TRASPOSICIONES..... | 169 |
| 3. EL SUBGRUPO ALTERNADO | 174 |
| 3.1. SIGNATURA Y PARIDAD | 174 |
| 3.2. EL SUBGRUPO ALTERNADO..... | 177 |
| 4. EL SUBGRUPO DIÉDRICO | 184 |
| 5. LA MÁQUINA ENIGMA..... | 190 |
| 5.1. DESCRIPCIÓN DE LA MÁQUINA ENIGMA..... | 191 |
| 5.2. FUNCIONAMIENTO DE LA MÁQUINA ENIGMA..... | 192 |
| 6. EJERCICIOS | 196 |
| 5. GRAFOS. REPRESENTACIÓN E IMPLEMENTACIÓN..... | 201 |
| 1. DEFINICIÓN Y TIPOS DE GRAFOS | 201 |
| 1.1. IMPLEMENTACIÓN EN MATHEMATICA..... | 203 |
| 2. REPRESENTACIÓN MATRICIAL | 208 |
| 2.1. MATRIZ DE ADYACENCIA | 208 |
| 2.2. MATRIZ DE INCIDENCIA | 219 |
| 3. REPRESENTACIÓN GRÁFICA CON MATHEMATICA..... | 222 |
| 4. GRAFOS ISOMORFOS | 234 |
| 4.1. EFICACIA Y OPTIMIZACIÓN..... | 247 |
| 5. COMBINATORIA EN GRAFOS..... | 258 |
| 5.1. GRAFOS DE n VÉRTICES..... | 259 |
| 5.2. GRAFOS DE n VÉRTICES Y m LADOS..... | 263 |
| 5.3. CONSTRUCCIÓN DE GRAFOS | 265 |
| 5.4. EFICACIA Y OPTIMIZACIÓN..... | 274 |
| 6. EL GRUPO DE AUTOMORFISMOS DE UN GRAFO | 290 |
| 7. EJERCICIOS | 301 |
| 6. GRAFOS REGULARES Y COMPLETOS. SUBGRAFOS Y GRAFOS BIPARTITOS | 311 |
| 1. GRADO DE UN VÉRTICE | 311 |
| 1.1. EL GRADO EN GRAFOS NO DIRIGIDOS..... | 311 |
| 1.2. EL GRADO EN GRAFOS DIRIGIDOS..... | 313 |
| 2. GRAFOS REGULARES Y GRAFOS COMPLETOS | 317 |
| 2.1. GRAFOS REGULARES | 317 |
| 2.2. GRAFOS COMPLETOS | 321 |
| 3. SUBGRAFOS Y GRAFOS BIPARTITOS | 326 |

| | |
|--|------------|
| 3.1. SUBGRAFOS | 326 |
| 3.2. SUBGRAFOS INDUCIDOS | 327 |
| 3.3. GRAFOS BIPARTITOS Y GRAFOS BIPARTITOS COMPLETOS..... | 329 |
| 3.4. COMPLEMENTO DE UN GRAFO | 336 |
| 4. EJERCICIOS | 337 |
| 7. CAMINOS Y CICLOS | 341 |
| 1. DEFINICIÓN Y TIPOS DE CAMINOS..... | 341 |
| 1.1. CAMINOS EN GRAFOS NO ORIENTADOS | 341 |
| 1.2. CAMINOS EN GRAFOS DIRIGIDOS..... | 345 |
| 1.3. TIPOS DE CAMINOS..... | 350 |
| 2. TEOREMA DEL NÚMERO DE CAMINOS | 356 |
| 3. GRAFOS CONEXOS Y COMPONENTES CONEXAS | 362 |
| 3.1. GRAFOS CONEXOS. COMPONENTES CONEXAS..... | 362 |
| 3.2. GRAFOS DIRIGIDOS FUERTEMENTE CONEXOS..... | 367 |
| 4. DISTANCIA Y GEODÉSICAS | 370 |
| 5. GRAFOS DE EULER | 374 |
| 6. GRAFOS DE HAMILTON..... | 382 |
| 7. OTROS EJEMPLOS | 387 |
| 8. EJERCICIOS | 400 |
| 8. COLORACIÓN DE UN GRAFO, GRAFOS PLANOS Y ÁRBOLES..... | 413 |
| 1. COLORACIÓN DE UN GRAFO | 413 |
| 1.1. COLORACIONES ÓPTIMAS | 414 |
| 1.2. COLORACIÓN: OPTIMIZACIÓN Y EFICACIA..... | 421 |
| 1.3. EL POLINOMIO CROMÁTICO | 434 |
| 2. GRAFOS PLANOS..... | 445 |
| 2.1. EFICACIA | 455 |
| 2.2. REPRESENTACIONES PLANAS. REGIONES DE UN GRAFO PLANO..... | 466 |
| 3. ÁRBOLES Y BOSQUES | 472 |
| 3.1. ÁRBOLES Y BOSQUES. CICLOS ELEMENTALES | 473 |
| 3.2. ÁRBOLES Y BOSQUES. NÚMERO DE LADOS | 479 |
| 4. OTROS EJEMPLOS | 483 |
| 5. EJERCICIOS | 496 |
| REFERENCIAS | 507 |
| ÍNDICE DE PROGRAMAS Y FUNCIONES | 513 |
| ÍNDICE DE TABLAS E ILUSTRACIONES | 517 |
| BIBLIOGRAFÍA..... | 521 |

PRÓLOGO

La teoría de grupos y la teoría de grafos constituyen unas de las partes del Álgebra y en particular de la Matemática Discreta de mayor importancia dentro de las Ciencias de la Computación, suponen una herramienta matemática e informática con numerosas y variadas aplicaciones a multitud de disciplinas. Su impacto dentro de las nuevas tecnologías hace incuestionable su estudio desde un punto de vista tanto teórico como computacional. La resolución de problemas relacionados con la teoría de grafos o grupos computacionalmente es muy recurrente e imprescindible en la actualidad.

Fruto de la experiencia docente de los profesores que componen el área de Álgebra de la Universidad de Jaén, surgió la idea de utilizar métodos computacionales para acercar algunos conceptos algebraicos que por su nivel de abstracción suelen resultar distantes a un lector menos encaminado al estudio de fundamentos matemáticos, esta idea quedó plasmada en el libro “Métodos Computacionales en Álgebra para Informáticos: Matemática Discreta y Lógica” (García, M.A., Ordóñez, C. y Ruiz, J.F. [25]). En este texto se crearon de forma eficaz métodos computacionales capaces de resolver los problemas más habituales relacionados con los contenidos de la asignatura de Álgebra I de la Ingeniería Informática de Gestión de la Universidad de Jaén. Este libro surge como una continuación natural de aquél, el formato, las fuentes y el estilo son los mismos, la metodología empleada es muy parecida, también se ha elegido a Mathematica como aplicación sobre la que desarrollar los programas expuestos; si bien, viene marcado por sustanciales diferencias que pasamos a destacar. Aunque es un texto cuyos contenidos están relacionados con la asignatura de Álgebra II de la Ingeniería Informática de Gestión de la Universidad de Jaén y puede usarse como guía docente, durante el desarrollo del mismo se vio la necesidad de profundizar más sobre aspectos computacionales y algorítmicos, siempre desde un punto vista puramente matemático y en particular algebraico, si bien en ocasiones también se analicen aunque superficialmente, cuestiones de programación por las necesidades propias de las construcciones algorítmicas tratadas. Este extremo marcó el texto desde el principio, encaminándolo obligatoriamente hacia un lector más experto en cuestiones relacionadas con la programación. Parte en consecuencia, de una experiencia previa respecto a Mathematica u otros lenguajes de programación y así se presupone en casi todo el libro. También se suponen algunos conocimientos de matemáticas básicas, de teoría de conjuntos, del anillo \mathbb{Z} , del anillo de polinomios y de álgebra lineal.

En todo momento se establece como prioridad y es manifiesto en casi todo el texto, la transparencia en la construcción de los programas, utilizando herramientas básicas de programación para que el lector puede trasladar lo expuesto a la plataforma o el lenguaje deseado, traduciendo las funciones específicas de Mathematica y dando alternativas programables en otros lenguajes. Sacrificando eficacia en los métodos por una mejor

comprensión. Aunque la optimización es un tema tratado en este libro, siempre se hace, salvo en contadas ocasiones, a través de métodos o conceptos matemáticos, aunque no es una cuestión primordial del libro, queda patente su importancia, si bien es un tema que queda relegado a ámbitos de conocimiento distintos. Expresamente se han incluido un gran número de ejemplos que ilustren los distintos tipos de problemas que los métodos desarrollados permiten resolver. Con objeto de facilitar una mayor fluidez en la lectura, se han dejado algunos problemas, que por su similitud a otros lo permitían, propuestos como ejercicios. También se han propuesto intencionadamente, ejercicios que muestren directamente situaciones prácticas en las que aplicar los métodos desarrollados. No se han incluido demostraciones de los resultados usados por no ser su estudio un objetivo de este libro.

Como directriz general, se han desarrollado los métodos computacionales imprescindibles para cubrir los aspectos más básicos relacionados con la teoría de grupos y de grafos. En esta línea, este libro está destinado a cualquier lector o estudiante interesado en aprender grupos y grafos a través de técnicas informáticas, así como también sus posibles aplicaciones. Es obvia su fuerte relación con las Ciencias de la Computación y con aquellas otras que necesiten de estas herramientas matemáticas para modelar problemas concretos con el objeto de trasladarlos al ordenador, si bien se ha pretendido dar un punto de vista conceptual totalmente algebraico, se han buscado ejemplos y ejercicios lo más prácticos y cercanos a la vida cotidiana que ha sido posible. En él se intentan resolver la mayoría de problemas básicos relacionados con los grupos y los grafos: desde aquellos de baja dificultad como el cálculo del número de caminos, grado de un vértice, propiedades de los grupos finitos,...; hasta otros más duros, de dificultad media o alta como el cálculo de subgrupos, ciclos de Hamilton, grafos planos, la coloración de un grafo, polinomio cromático...

Tras un primer capítulo meramente introductorio, el libro está dividido en dos partes bien diferenciadas, por un lado tenemos tres capítulos dedicados a la teoría de grupos y otros cuatro dedicados a la teoría de grafos. En el capítulo 2, primero dedicado a los grupos, se hace un estudio de la estructura de grupo, en particular se estudia en profundidad los grupos finitos, fácilmente trasladables a la computadora. El tercer capítulo está dedicado al estudio de los subgrupos y el grupo cociente. Como ejemplo de las dificultades que se presentan, se analiza con detenimiento el problema del cálculo de todos los subgrupos de un grupo finito. Se termina la parte de grupos con el capítulo 4, que está dedicado al grupo simétrico. En la parte dedicada a grafos, se ilustran los conceptos más básicos pertenecientes a esta teoría, como preámbulo de un estudio más profundo y completo que se escapa de los objetivos de este libro, es por ello que no se progresan en cuestiones algorítmicas más complejas de sobra conocidas y estudiadas en el campo de la Informática. En el capítulo 5 se aprende a implementar y representar los grafos, también se estudian el isomorfismo entre grafos, el problema combinatorio y el grupo de automorfismos de un grafo. En el capítulo 6 se estudian conceptos básicos de la teoría de grafos: grado, regular, completo, subgrafo... En el 7 estudiamos los caminos y ciclos de un grafo, analizando el problema de los grafos de Euler y de Hamilton, por último en el capítulo 8 se analizan los problemas: coloración de un grafo, grafos planos y árboles.

Los programas y ejemplos que aparecen en el libro se adjuntan en un CD-ROM para facilitar su uso sin necesidad de tener que escribir personalmente el código de los

mismos. A pesar de las distintas revisiones realizadas, el lector puede encontrar problemas al ejecutar los programas en casos o situaciones no contempladas al diseñarlos o erratas propias de un libro de estas características. Cualquier errata o error encontrado, sugerencia o comentario, que el lector estime conveniente será bienvenida y considerada en futuras versiones.

J. F. Ruiz.

Prólogo 2^a edición

Respecto a la primera edición, además de corregir erratas o errores de todo tipo (ortográficas, gramaticales, de estilo, de redacción,...), se han retocado y clarificado algunos métodos para que puedan usarse de forma más cómoda y efectiva. Se han probado y corregido, si fuera necesario, todos los programas para su correcto funcionamiento en la versión 6 de Mathematica, también se han añadido y comentado las nuevas funciones que son interesantes para nuestros propósitos que por primera vez incorpora esta versión, en este sentido el manejo de grafos y su representación, ahora puede realizarse también directamente con las funciones que trae Mathematica 6 y que también incorporan las versiones 7 y 8. No existen diferencias importantes con las versiones 7 y 8, y podremos usar los métodos expuestos en el libro igualmente en estas versiones. También se han añadido programas para la resolución de algunos problemas que quedaron fuera de la primera edición, como el cálculo de grupos de orden pequeño, el grupo diédrico,...; y se han añadido nuevos ejemplos, aplicaciones prácticas, ejercicios y problemas con parámetros personalizables para su uso en el aula. También se han adaptado los programas y las funciones, tras la experiencia en clase, para acercarlos al alumno, facilitando su uso y haciéndolos más amigables.

Este libro y los programas que incluye, inicialmente fueron diseñados para la versión 5.2 de Mathematica, con posterioridad apareció la versión 6 que incluye profundos cambios e incompatibilidades que afectan a los contenidos de la primera edición de este libro y en particular a los capítulos de grafos. En la versión 6 se pueden definir los grafos como objetos de tipo “grafo”, y estos pueden estudiarse con una amplia biblioteca de funciones que en las versiones anteriores no existían. Algunas de estas funciones realizan tareas similares a funciones o programas incluidos en este texto, aunque en su mayoría, sólo funcionan sobre objetos de tipo grafo. En la medida de lo posible, se han incluido y estudiado la mayoría de las nuevas herramientas simultáneamente a las funciones y programas originales, discutiendo en cada caso la idoneidad de los mismos. Los programas originales de la versión 5.2 de Mathematica se han conservado en el disco de datos, que viene con el libro, sin embargo, el lector debe tener presente que los programas expuestos en el texto se han actualizado a las versiones posteriores, en particular a la versión 6 y siguientes, es probable en muchos casos, que tal cual vienen expuestos en el libro, no funcionen en versiones de Mathematica anteriores a la 6. Las versiones 7 y 8 de Mathematica apenas han cambiado respecto a los temas tratados en el libro, en particular las librerías y funciones que se usan no han sufrido cambios relevantes respecto a la versión 6, por lo que podremos usar Mathematica 7 y 8 sin problemas y tan sólo encontraremos

alguna diferencia de tipo visual sin demasiada importancia respecto a las salidas que se muestran en el texto, que se corresponden con las que proporciona la versión 6, o algún cambio en las librerías o paquetes de funciones. Respecto a los tiempos empleado en los cálculos tampoco se han observado diferencias de importancia entre la versión 6 y las versiones posteriores a ésta, por lo que podremos usar cualquiera de ellas. Es posible que con ordenadores o sistemas operativos más modernos, o en versiones posteriores de Mathematica, se puedan encontrar diferencias relevantes en la efectividad o incluso nuevas incompatibilidades.

Por último, dar las gracias y mostrar mi agradecimiento a la profesora, compañera y amiga, D^a Carmen Ordóñez, por sus sugerencias y consejos fruto de su experiencia en clase con la edición anterior del texto.

J. F. Ruiz.

1. MATHEMATICA Y HERRAMIENTAS BÁSICAS DE PROGRAMACIÓN

Sin ánimo de ser exhaustivos ni pretenciosos, introducimos en este primer capítulo las herramientas básicas de programación necesarias para la comprensión de los programas de los capítulos posteriores, las mostramos con la notación de Mathematica. Este capítulo no pretende constituir una guía de programación o de uso del Mathematica para el lector, por el contrario, tanto sólo es una revisión superficial, una exploración orientativa de las herramientas y métodos habituales de programación adaptadas a la plataforma elegida para la implementación posterior de los problemas planteados en este libro. Se presuponen unos conocimientos previos sobre programación, es por ello, que al lector no familiarizado con las técnicas y herramientas de programación básicas se le remite a un aprendizaje previo sobre las mismas, ya que el estudio detallado de éstas no es objetivo de este libro y en particular de este capítulo.

1. MATHEMATICA

La plataforma, aplicación o lenguaje de programación aconsejable para la resolución de los distintos problemas que se plantean en este libro es diferente según la complejidad, el tipo de respuesta que deseemos dar, la finalidad de dicha respuesta o el ambiente en el que lo planteemos. En principio, en este texto, nos planteamos la exploración de los distintos conceptos y problemas relacionados con la teoría de grupos y la teoría de grafos a través del ordenador con varios objetivos, criterios y prioridades. Enumeramos en orden de importancia, tal y como se han tenido en cuenta a la hora de diseñar y concebir cada uno de los métodos que aparecen en el texto los más interesantes:

- I. El primero y más importante, es el apartado didáctico y motivador, es por ello que la mayoría de las respuestas a los distintos problemas se han dado en base a herramientas de programación básicas, fácilmente comprensibles y trasladables a cualquier plataforma. Cuando obligatoriamente se ha tenido que optar por herramientas específicas y propias del programa elegido para la implementación de los algoritmos usados, se ha intentado ser lo más transparente posible en cuanto a las técnicas y métodos usados, y mostrar, en la medida de lo posible, paralelamente un planteamiento alternativo que evitase dichas técnicas o métodos.

- II. En segundo lugar, la necesidad de conseguir una colección lo más completa posible de algoritmos que den respuesta a la mayoría de los posibles problemas, sacrificando en ocasiones la facilidad en la comprensión a cambio de una respuesta completa al problema que proporcione a su vez un instrumento de autoevaluación, en conexión con el apartado anterior.
- III. Como tercera, la optimización del problema, en ocasiones se plantea y muestra al lector la importancia de la eficacia y optimización, pero siempre que ha sido necesario se ha sacrificado a favor de los dos objetivos anteriores.

En conjunto, teniendo en cuenta los objetivos del libro y el orden de las prioridades y criterios señalados, se ha elegido a Mathematica como la aplicación más cómoda y aconsejable. Si bien ha de tenerse en cuenta y así se refleja en las ocasiones oportunas, que no siempre ésta sería la plataforma más idónea para la resolución u optimización de algunos problemas. Otro punto fuerte a su favor viene impuesto por la referencia [25], germen de este libro.

Mathematica es una aplicación comercial extensamente implantada en ámbitos universitarios y profesionales que permite una cómoda interacción a la hora de realizar cálculos matemáticos. En este texto se hará una brevíssima descripción del mismo, remitiendo al lector poco familiarizado a otros textos más básicos que introducen el funcionamiento del programa Mathematica de forma más amplia y completa.

Mathematica dispone de una gran cantidad de funciones específicamente destinadas a la resolución de problemas de grupos y grafos, muchas de las cuales se analizan en el libro, otras pueden encontrarse en la red Internet, pero siempre que ha sido viable se han desarrollado al margen de ellas y con un carácter lo más transparente posible, alternativas programadas de tal forma que puedan llevarse a otros lenguajes de programación, evitando de esta forma la dependencia respecto a dicho programa.

| PROGRAMA | COMENTARIOS |
|--|------------------------------|
| Definimos variables y funciones previas; | Comentarios. |
| CÓDIGO | Comentarios y explicaciones. |
| Programa ... Nombre. | |

Ilustración 1.1. Esquema de un programa, función o procedimiento.

Mathematica se compone de dos partes bien diferenciadas: el interfaz de usuario a través del cual interactuamos con el programa y el núcleo, encargado de realizar todos los cálculos. El interfaz está compuesto por una barra de menú donde encontramos las opciones principales (archivo, edición, control del núcleo, ayuda,...), paletas opcionales para el control de ciertas herramientas específicas y el área de trabajo o documento de Mathematica en donde aparecerán los cálculos que vayamos realizando. A su vez, el área de trabajo está organizado en celdas (de entrada, salida, texto,...) que facilitarán los cálculos que realicemos e irán indexadas (entradas y salidas) para no crear conflictos cuando el número de cálculos sea elevado. En este libro distinguiremos entre entradas y salidas indicando “*In[]:=*” delante de la entrada y “*Out[]*=” precediendo a la salida, el

código de entrada lo destacaremos en negrita y el de salida irá en la fuente de texto “Courier New”; los programas los destacaremos y describiremos en un cuadro de dos columnas, en la primera se incluirá el código del mismo y en la segunda los comentarios y explicaciones pertinentes (véase la ilustración 1.1.), resaltaremos con distintas escalas de grises algunas partes del mismo: funciones y programas previos, código que el usuario deberá alterar para conseguir resultados particulares, bucles,...

2. PROGRAMACIÓN

Mathematica permite usar herramientas básicas de programación simultáneamente a la amplia librería de funciones de la que dispone, lo cual le da una gran potencia y versatilidad. A continuación enumeramos las herramientas de programación básicas de Mathematica que usaremos, para más detalle podemos consultar la ayuda de Mathematica o cualquiera de las referencias bibliográficas que introducen pormenorizadamente estas herramientas (por ejemplo, los primeros capítulos de [25]).

- a) Aritmética básica y leyes de precedencia. No haremos mención de ninguna circunstancia especial respecto a las operaciones usuales de suma, producto, multiplicación, división o potencia, pues el uso de las mismas coincide en Mathematica con el que se hace en cualquier otro lenguaje.

b) Variables y funciones. En Mathematica no es obligatoria la declaración del tipo de variable a usar; si no indicamos nada, ésta es automática. Los tipos de variables que se usan coinciden con los tipos habituales de datos numéricos usados en matemáticas: enteros (“Integer”), reales (“Real”), complejos (“Complex”),... Como nombres de variables o funciones podemos usar cualquier letra o palabra que no esté reservada por el programa¹ y su implementación será como sigue:

nombrevariable=definición de la variable

Es habitual utilizar abreviaturas del tipo **a++** o **++a** sobre todo en contadores, también es posible el uso de funciones recursivas. Para eliminar de la memoria una variable o función se dispone de la orden **Clear[]**.

- c) Listas. Podemos definir una lista directamente escribiendo los elementos que la compongan separados por comas y entre llaves:

lista={i₁,i₂,...,i_n}

Existen varias funciones para manejar listas:

¹ Es posible, aunque poco recomendable, desbloquear las palabras o letras reservadas por el programa. Normalmente protegidas porque son usadas como nombre de funciones prediseñadas de Mathematica (véase la función Unprotect[] en la ayuda de Mathematica).

- AppendTo[], Append[], PrependTo[], Prepend[], Insert[] para añadir elementos a la lista y Delete[] para eliminar elementos de una lista.
 - Join[] para pegar listas (unión disjunta).
 - Union[] e Intersection[] para unir e intersekar listas tratadas como conjuntos.
 - Complement[], para calcular complementos o diferencias lógicas como conjuntos.
 - First[], Last[], nombre_de_la_lista[[posición]] para referirnos a algún elemento particular de la lista.
 - Length[], nos devuelve la longitud de la lista.
 - Position[], permite encontrar elementos dentro de una lista.
 - Sort[], ordena una lista.
 - Take[], permite elegir un elemento o varios de la lista.
- d) Matrices o tablas. La implementación de matrices o tablas en Mathematica es muy sofisticada y, a diferencia de otros lenguajes, tiene una gran variedad de funciones disponibles para el manejo de estas estructuras. En el caso de pretender trasladar los métodos computacionales que se han desarrollado en este libro, desde Mathematica a otra aplicación, sería recomendable previamente construirse, si no existiera, una completa librería, similar a la disponible en Mathematica, de funciones o procedimientos que permitan manejar tablas o matrices, puesto que estas funciones son ampliamente utilizadas en este libro. Utilizaremos la función Table[] para definir o declarar una matriz o tabla de tantas entradas como deseemos:

```
tabla=Table[valorpordefecto,{i1,i1_inicio,i1_fin},  
{i2,i2_inicio,i2_fin},..., {in,in_inicio,in_fin}]
```

O bien directamente como una lista de listas.

Para referirnos a una coordenada particular, por ejemplo la (j_1, j_2, \dots, j_n) usaremos:

tabla[[j₁,j₂,...,j_n]]

También podemos referirnos a ella con cualquiera de las siguientes expresiones:

tabla[[j₁,j₂,...,j_{n-1}]][[j_n]],
tabla[[j₁,j₂,...,j_{n-2}]][[j_{n-1}]][[j_n]],
...
tabla[[j₁]][[j₂]][[j_n]]

Para manipular matrices dispondremos de gran cantidad de funciones y métodos, los que usaremos en este libro con mayor frecuencia son:

- Transpose[], calculará la matriz traspuesta.
- MatrixPower[], calculará la potencia de una matriz.

- Eigenvalues[], devolverá los valores propios.
 - SparseArray[], permite cambiar las coordenadas de una matriz de forma rápida y cómoda.
 - IdentityMatrix[], devolverá la matriz identidad.
 - Dimensions[], nos dará las dimensiones de la matriz.
 - Dadas dos matrices **A** y **B** y un escalar **x**, calcularemos el producto con **A.B**, la suma con **A+B** y el producto por escalar: **x A** o **x*A**.
 - MatrixForm[], mostrará en pantalla el argumento o entrada en forma de matriz.
 - TableForm[], mostrara en pantalla el argumento en forma de tabla, podemos incluir como segundo argumento opciones para una representación particular, tales como “TableAlignments”, “TableDepth”, “TableDirections”, “TableHeadings” o “TableSpacing”.
 - Det[], calculará el determinante de la matriz.
 - Minors[], calculará los determinantes de las submatrices (menores).
 - Take[], TakeRows[], TakeColumns[], SubMatrix[], entre otras, son algunas funciones que permiten manejar submatrices².
- e) Condicionales y operadores lógicos. Para comparar expresiones, se usarán: ==, ===, !=, <=, >= y <>. Uniremos expresiones lógicas con las funciones And[], Or[], que habitualmente usaremos también con la notación alternativa **&&** para el “y”, || para el “o”, negaremos una expresión con Not[] o simplemente pondremos a su izquierda !. Usaremos TrueQ[] para evaluar una expresión lógica. Los valores booleanos que tomarán dichas expresiones serán True o False y como condicional usaremos:

If[condición,proceso1,proceso2]

Equivalente a

Si condición entonces proceso1 en otro caso proceso2

También se dispone de la función Which[]:

Which[condición1,proceso1,condición2,proceso2,...]

Equivalente a

Si condición1 entonces proceso1 en otro caso si condición2 entonces proceso2...

- f) Bucles. En Mathematica existen tres tipos de bucles:

² TakeRows[], TakeColumns[] y SubMatrix[] en Mathematica 5.2 sólo están disponibles si previamente cargamos el paquete de funciones de Álgebra Lineal para la manipulación de matrices:

In[7]:= <<LinearAlgebra`MatrixManipulation`;

I. El bucle Do[]:

Do[cuerpo,{contador,inicio,fin,paso}]

Repetirá la ejecución del cuerpo para todos los valores que pueda tomar “contador”, desde “inicio” hasta “fin” avanzado a cada vuelta tanto como indique “paso”,

II. El bucle For[]:

For[contador=inicio,test,incremento,cuerpo]

Repetirá la ejecución del cuerpo para todos los valores que pueda tomar “contador”, desde “inicio”, mientras se verifique “test”, avanzado a cada vuelta tanto como indique “incremento”.

III. El bucle While[]:

While[test,cuerpo]

Repetirá la ejecución del cuerpo mientras se verifique “test”.

También disponemos de la función Break[] que interrumpirá súbitamente la ejecución del bucle. Puede observarse además que ninguno de estos bucles es indispensable, por el contrario, podemos usar en cada ocasión el que deseemos o nos dé la solución más cómoda.

Ejemplo 1.1. Generamos aleatoriamente una lista de 10 números enteros comprendidos entre 0 y 1000, para ello usamos la función Random[tipo_de_variable,rango]:

In[]:= lista=Table[Random[Integer,{0,1000}],{i,10}]

al ser listas aleatorias las salidas variarán cada vez que hagamos el cálculo, en este ejemplo:

Out[]={294,471,25,803,358,277,259,175,681,14}

con la función Sort[] podemos ordenarlos. Hacemos un programa que ordene de menor a mayor los números enteros de “lista” utilizando bucles y condicionales, esto es, que haga lo mismo que Sort[]. Para ello, sin ánimo de ser eficaces, ni plantearnos disquisiciones acerca de los algoritmos de ordenación o sobre órdenes distintos al usual de \mathbb{Z} , simplemente vamos comparando cada elemento con su consecutivo, si no están en orden los intercambiamos y damos pasadas a todos los elementos de la lista hasta que estén ordenados.

| PROGRAMA | COMENTARIOS |
|--|---|
| ordena=True; | Variabile auxiliar que indicará si debemos continuar dando pasadas. |
| While[ordena, ordena=False; | BUCLE 1 |
| For[i=1,i<Length[list]&&!ordena,i++, | BUCLE 2 El primer bucle dará |

| | | |
|---|--|---|
| If[lista[[i]]>lista[[i+1]], ordena=True;temp=lista[[i]]; lista[[i]]=lista[[i+1]];lista[[i+1]]=temp;];]; | Si un elemento y el consecutivo no están ordenados los intercambiamos. | vueltas hasta que la variable “ordena” alcance el valor “True”. |
| lista | Mostramos la lista de elementos ordenada. Programa 1.1. Ordenación de listas. | |

Lo aplicamos a nuestro ejemplo:

In[]:= **ordena=True;**

⋮ ⋮

Out[]=
{14, 25, 175, 259, 277, 294, 358, 471, 681, 803}



3. OTRAS FUNCIONES

En este epígrafe enumeraremos otras funciones que también usaremos.

- a) Para definir funciones o procedimientos particulares con variables de uso local se dispone de las funciones Module[] y Block[]:

Nombrefuncion[variable1_,variable2_,...]:=
Module[{variables_locales},cuerpo]

Nombrebloque=Block[{variables_locales},cuerpo]

- b) Existe una gran colección de funciones matemáticas ya implementadas en Mathematica, usaremos algunas de ellas como Sum[] para calcular sumatorias, Mod[] y Quotient[] para calcular el resto y cociente de dividir dos enteros, Divisors[] para calcular los divisores positivos de un número entero, IntergerDigits[] devuelve una lista de los dígitos en base b que forman un entero, Sqrt[] la raíz cuadrada,... Además también encontramos implementadas las constantes más usadas en matemáticas, la unidad compleja i se usará como **I**, el número e como **E**, π como **Pi**,...
- c) Mathematica permite utilizar expresiones simbólicas, para ello existen funciones específicas, como ejemplo, mostramos algunas de las que usaremos más adelante:
- Para resolver ecuaciones o sistemas de ecuaciones con tantas variables como queramos:

Solve[ecuaciones,variables]

- Para factorizar o expandir un producto:

Factor[polinomios] y Expand[expresión]

- Para simplificar una expresión:

Simplify[expresión]

- d) La representación gráfica es un apartado también ampliamente tratado en Mathematica, en este libro será indispensable para la representación de grafos. Además de las funciones específicas destinadas a la representación de grafos que se desarrollarán en el capítulo 5, tenemos la función,

Show[Graphics[lista_de_objetos]]

que dibujará todos los objetos que deseemos y hayamos incluido en la lista “lista_de_objetos”, como por ejemplo: círculos con Circle[], discos con Disk[], líneas con Line[], puntos con Point[], rectángulos con Rectangle[],... También dispondremos de funciones como RGB[] o GrayLevel[] para modificar el color y multitud de opciones para la representación, como “AspectRatio”, “AxesOrigin”, “Background”, “PlotRange”,... En Mathematica, también disponemos de una potente función: Animate[], que nos permite animar las representaciones gráficas o comprobar como ciertos cambios pueden afectar al resultado de nuestro problema.

- e) Para mostrar información, disponemos de la función Print[] cuyo argumento o entrada estará compuesto por todo aquello que queramos mostrar en pantalla, si son varios argumentos los separamos por comas. Si queremos cambiar el tipo de fuente, tamaño, color,... podemos usar StyleForm[] con las opciones “FontSize”, “FontWeight”, “FontSlant”, “FontFamily”, “FontColor” y “Background”.
f) Además de las ya nombradas encontraremos otras muchas órdenes con funcionalidades específicas que comentaremos conforme nos aparezcan.

Aparte de las funciones y herramientas que incluye Mathematica y que siempre están disponibles para su uso, también contaremos con otras incluidas en paquetes específicos, usaremos los siguientes:

Para teoría de grupos y grafos usaremos:

`<<Combinatorica`;`

Para teoría de grafos:

`<<GraphUtilities`;`

Todas las funciones que usaremos, incluidas en estos paquetes, serán desarrolladas y comentadas cuando las necesitamos. La mayoría de ellas y salvo en contados casos, son traducidas a herramientas básicas de programación, ya que estos paquetes de funciones, en

principio, son exclusivos de Mathematica y es probable que no podamos disponer de ellas en otros lenguajes.

Finalizamos este capítulo con algunos ejemplos encaminados a aclimatar al lector a la notación propia de Mathematica:

Ejemplo 1.2. A modo de ejemplo vamos a realizar un pequeño y vano programa cuyo único objetivo será mostrar el uso de algunas de las herramientas comentadas.

Vamos a representar en forma de calendario una fecha cualquiera, mostrando el mes completo. Para ello necesitaremos usar variables, funciones, tablas, listas, condicionales, bucles y también practicaremos el uso de la función Print[] y sus opciones. Hemos de tener en cuenta que no todos los meses tienen los mismos días y que algunos años son bisiestos³. Contaremos los días pasados desde el uno de enero de 1980 hasta el primer día del mes que representamos, para determinar qué día de la semana es y así dibujar correctamente la tabla que represente a dicho mes. Lógicamente podría modificarse para que funcionase para años anteriores a 1980 con bastante facilidad (ejercicio 1.3.).

| FUNCIÓN | COMENTARIOS |
|--|---|
| Calendario[dia_,mes_,año_]:=Module[| Definimos la función “Calendario” que tendrá 3 entradas, respectivamente, el dia, el mes y el año que compone una fecha particular. |
| {diasmes,diasmesbis,unoenero1980,nombremes, dias,i,diauno,Mes}, | Indicamos todas las variables locales que usará la función. |
| unoenero1980=2; | El 1 de enero de 1980 fue martes, por lo que lo indicamos en esta variable para posteriormente contar los días hasta el mes que indiquemos. |
| diasmes={31,28,31,30,31,30,31,31,30,31,30,31}; diasmesbis={31,29,31,30,31,30,31,31,30,31,30,31}; | Definimos dos listas que incluyen los días que tiene cada mes, distinguimos entre los años bisiestos y no bisiestos. |
| nombremes={"Enero","Febrero","Marzo","Abril", "Mayo","Junio","Julio","Agosto","Septiembre", "Octubre","Noviembre","Diciembre"}; | En esta otra lista almacenamos el nombre que tiene cada mes. |
| dias=0; | La variable “días” almacenará el número de días que hay entre el 01/01/1980 y el 01/mes/año del “mes” y año que indiquemos. |
| Do[If[Mod[i,4]==0,dias=dias+366,dias=dias+365] ,{i,1980,año-1}]; Do[If[Mod[i,4]==0, dias=dias+diasmesbis[[i]],dias=dias+diasmes[[i]]]; ,{i,1,mes-1}]; | Con dos bucles contamos los días, el primero que avanzará de año en año (teniendo en cuenta los bisiestos) y el segundo que contará los días de los meses restantes |
| diauno=Mod[dias+unoenero1980,7]; If[diaun==0,diauno=7]; | Determinamos que día de la semana es el 01/mes/año, para poder representar bien la tabla de este mes. |

³ Un año es bisiesto si es divisible por 4, excepto aquellos divisibles por 100, que para ser bisiestos, también deben ser divisibles por 400, por ejemplo, el año 2000 si es bisiesto porque es divisible por 400, sin embargo, el año 2100 o el 1900 no lo son. Este último extremo no se ha tenido en cuenta en la programación, aunque fácilmente podría añadirse un nuevo condicional que lo solucionase.

| | |
|--|--|
| <pre>If[Mod[año,4]==0,diasmes=diasmesbis];</pre> | <p>Tenemos cuidado, por si “año” es un año bisiesto.</p> |
| <pre>mes=Table[If[(j+((i-1)*7))-diauno+1>diasmes[[mes]] (j+((i-1)*7))-diauno+1<1 , "-" , If[(j+((i-1)*7))-diauno+1==dia, StyleForm[(j+((i-1)*7))-diauno+1, FontWeight->"Bold",FontSize→16] , (j+((i-1)*7))-diauno+1]] ,{i,6},{j,7}];</pre> | <p>Calculamos la tabla que representa al mes: “mes” de “año”. Destacamos en la tabla el día de la fecha concreta que representamos, cambiando su tamaño e indicando que la muestre en negrita.</p> |
| <pre>Print[StyleForm[nombremes[[mes]], FontWeight→"Bold",FontSize→18]," - ", StyleForm[año,FontWeight→"Bold", FontSize→18]];</pre> | <p>Mostramos en pantalla el nombre del mes y el año con una fuente más grande de la que el programa usa por defecto.</p> |
| <pre>Print[TableForm[Mes,TableHeadings→{None, {"L","M","X","J","V","S","D"}}, TableSpacing→{2,2}]];];</pre> | <p>Mostramos en pantalla la tabla, como cabecera de cada columna ponemos L, M, X, J, V, S y D para indicar el día de la semana.</p> |

Función 1.2. Calendario.

Introducimos la función 1.2.:

```
In]:= Calendario[dia_,mes_,año_]:=Module[
```

•
•
•

Hacemos el cálculo que deseamos:

In[]:= **Calendario[1,1,2006]**

Out[] = **Enero - 2006**

| L | M | X | J | V | S | D |
|----|----|----|----|----|----|----------|
| - | - | - | - | - | - | 1 |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| 30 | 31 | - | - | - | - | - |

Para representar la fecha actual podemos coger la fecha del sistema mediante la función Date[]:

Calendario[Date[[3]],Date[[2]],Date[[1]]]



Ejemplo 1.3. Con el mismo espíritu que en el ejemplo anterior y sin ánimo de ser exhaustivos, representamos el sistema orbital Tierra-Luna alrededor del Sol, supondremos órbitas totalmente circulares y en la eclíptica, los tamaños y distancias no son fieles, si bien se ha procurado que los periodos de rotación de la Tierra alrededor del Sol y de la Luna alrededor de la Tierra si lo sean aproximadamente.

En este programa usaremos listas, bucles, funciones numéricas y se practicará el apartado gráfico de Mathematica.

Con el programa 1.3. generamos los fotogramas que compondrán la animación que pretendemos crear y la mostramos usando la función Animate[]⁴de forma similar a un bucle Do[].

| PROGRAMA | COMENTARIOS |
|---|--|
| <code>periodolunar=27.3;</code> <code>periodoterrestre=365.25;</code> <code>velocidad=-0.01;</code> | Introducimos el periodo de traslación lunar alrededor de la Tierra, el de la Tierra alrededor del Sol y en velocidad indicamos la rapidez con la que queremos que vaya la animación. |
| <code>j=0;</code> | La variable “j” representará el ángulo de rotación de la Luna respecto a la Tierra. |
| <code>Animate[</code> | Con Animate[] mostramos los fotogramas que se van generando para cada “i”, siendo “i” el ángulo que determina la posición de la Tierra en el plano de la eclíptica. |
| <code>j=(2*Pi-i)*(periodoterrestre/periodolunar));</code> | Variamos “j” para cada fotograma. |
| <code>texto=StringJoin["Días: ",</code> <code>ToString[Round[periodoterrestre-</code> <code>i/(2*Pi)*periodoterrestre]],</code> <code>" Periodos lunares:",</code> <code>ToString[Round[Abs[(j-Pi)/(2*Pi)]]]];</code> | En la variable “texto” almacenamos la leyenda que aparecerá en la parte inferior de la animación. |
| <code>grafico={</code> | La variable “grafico” almacenará la lista de objetos a representar en cada fotograma. |
| <code>{GrayLevel[.5],Circle[{0,0},{1,2/3}]},</code> | La línea que marca la órbita terrestre alrededor del Sol. |
| <code>{RGBColor[1,1,0],Disk[{0,0},.2]},</code> | El disco que representa al Sol. |
| <code>{RGBColor[0,0,1],Disk[{Cos[i],Sin[i]^2/3},.06]},</code> | El disco que representa a la |

⁴ La función Animate[] sólo está disponible a partir de la versión 6 de Mathematica, en versiones anteriores podemos generar todos los frames que componen la animación con un bucle, por ejemplo con Do[].

| | |
|---|--|
| | Tierra. |
| {RGBColor[1,1,1],Disk[{(Cos[j]^*.2)+Cos[i], (Sin[j]^*.2*2/3)+Sin[i]^*.2/3}, .02]}, | El disco que representa a la Luna. |
| {RGBColor[1,1,1],BackGround->Black, Text[texto,{-.3,-1.1}]\}; | La leyenda de la parte inferior. |
| {RGBColor[1,0,0],Background->Yellow, Text["SOL",{0,0}\}}\}; | Escribimos "SOL" en rojo dentro del disco solar. |
| Show[Graphics[grafico],AspectRatio->1, AxesOrigin->\{0,0\},PlotRange->\{\{-1.3,1.3\}, {-1.3,1.3\}\},Background->GrayLevel[0]] | Representamos el gráfico completo, indicamos varias opciones para el dibujo. |
| ,{i,2*Pi,0,velocidad}] | Indicamos cómo los valores que debe tomar "i" para crear la animación. |

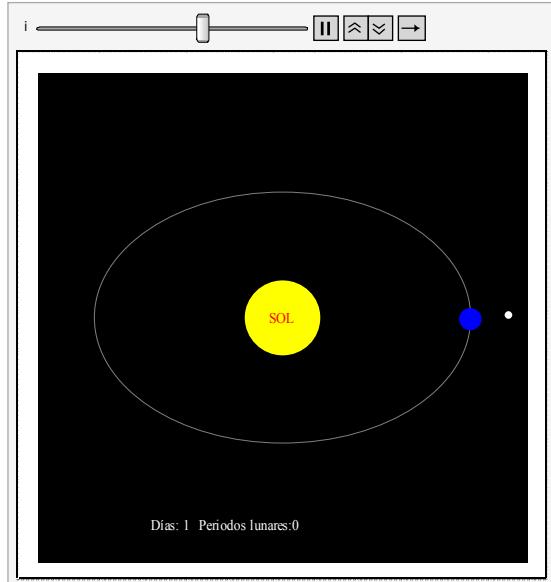
Programa 1.3. Sistema orbital Sol/Tierra/Luna.

Generamos la lista de fotogramas y mostramos con la función `Animate[]`:

In[]:= **periodolunar=27.3;**
periodoterrestre=365.25;
velocidad=-0.01;

⋮ ⋮

Out[]=



Ejemplo 1.4. En este ejemplo vamos a recrear el juego comúnmente conocido como “buscaminas”, practicaremos los bucles, los condicionales, las matrices y haremos uso de funciones recursivas. Para ello vamos a definir varias funciones con distintas finalidades:

- NuevoJuego[filas,columnas,bombas]**, generará un nuevo juego de forma aleatoria con un tablero formado por el número de filas, columnas y bombas que indiquemos.
- Levantar[i,j]**, descubrirá el casillero correspondiente a las coordenada (i, j) , usará la función **Mostrar[i,j]**, si hay una bomba perderemos, si no la hay nos indicará el número de bombas que hay alrededor y en el caso de ser cero, automáticamente descubrirá todas las casillas de alrededor, para ello usará **Mostrar[]** (1.5.) recursivamente.
- Marcar[i,j]**, cuando deduzcamos que en la posición (i, j) hay una bomba la marcaremos con esta función, cuando hayamos marcado un número de casillas igual al de bombas, esta función comprobará si hemos acertado en todas las bombas y nos dirá si hemos ganado.

Para jugar, primero generaremos un juego nuevo, luego usando las funciones **Levantar[]** y **Marcar[]** e iremos destapando las casillas del tablero y marcando aquellas donde creamos que haya bombas, si marcamos una bomba por error podemos desmarcarla de nuevo con **Marcar[]**.

El código de las funciones sería el siguiente:

| FUNCIÓN | COMENTARIOS |
|---|---|
| NuevoJuego[filas_,columnas_,bombas_]:=Module[{i,j}, | Como argumentos tendrá el número de filas, columnas del tablero y el número de bombas que queremos que se incluyan. |
| juego=Table[0,{i,filas},{j,columnas}]; | Generamos la tabla “juego” que almacenará el tablero y toda la información. |
| Do[| Generamos aleatoriamente tantas bombas como indique “bombas”, para ello usamos un bucle Do[] . |
| i=Random[Integer,{1,filas}]; j=Random[Integer,{1,columnas}]; While[juego[[i,j]]=="B", i=Random[Integer,{1,filas}]; j=Random[Integer,{1,columnas}];]; juego[[i,j]]="B"; | Generamos aleatoriamente las coordenadas de una bomba, comprobamos que no sean coordenadas donde ya hayamos considerado una bomba. |
| If[i-1>0, If[juego[[i-1,j]]!="B",juego[[i-1,j]]++]; If[j-1>0, If[juego[[i-1,j-1]]!="B",juego[[i-1,j-1]]++];]; If[j+1≤columnas, If[juego[[i-1,j+1]]!="B",juego[[i-1,j+1]]++];]; | Almacenamos en “juego” el número de bombas que hay alrededor de cada casilla, por cada bomba agregada sumamos una unidad al valor que haya en cada una de las |

| | |
|--|---|
| <pre>]; If[i+1<=filas, If[juego[[i+1,j]] != "B", juego[[i+1,j]]++]; If[j-1>0, If[juego[[i+1,j-1]] != "B", juego[[i+1,j-1]]++];]; If[j+1≤columnas, If[juego[[i+1,j+1]] != "B", juego[[i+1,j+1]]++];];]; If[j-1>0, If[juego[[i,j-1]] != "B", juego[[i,j-1]]++];]; If[j+1≤columnas, If[juego[[i,j+1]] != "B", juego[[i,j+1]]++];]; ,{k,bombas}];</pre> | <p>casillas de alrededor, recordemos que por defecto el valor es cero.</p> |
| <pre> , {k,bombas}]; ocultas=Table["X",{i,filas},{j,columnas}]; numbombas=bombas; marcadas=0;</pre> | <p>Contador del bucle.</p> |
| <pre> Print[TableForm[ocultas, TableHeadings→Automatic, TableSpacing→{1,1},TableAlignments→Center]]; Print["Bombas: ",numbombas, ". Marcadas: ",marcadas];</pre> | <p>Definimos un tablero “ocultas”, donde almacenaremos la información que vayamos descubriendo y que tendrá que mostrarse por pantalla. También almacenamos el número de bombas para recordárselo al jugador y el número de bombas supuestamente marcadas o encontradas hasta el momento.</p> |
| <pre>];</pre> | <p>Mostramos por pantalla el tablero, con la información oculta.</p> |

Función 1.4. Buscaminas I.

| FUNCIÓN | COMENTARIOS |
|---|---|
| Mostrar[i_,j_]:=Module[{}, | <p>Los argumentos son las coordenadas.</p> |
| $If[0 < i \leq Dimensions[juego][[1]] \&& 0 < j \leq Dimensions[juego][[2]],$ | <p>Comprobamos que las coordenadas sean correctas (también se podría haber comprobado que fuesen números enteros).</p> |
| $If[juego[[i,j]] \neq \text{ocultas}[[i,j]],$ | <p>Comprobamos que no se hubiese destapado previamente esta casilla.</p> |
| $If[juego[[i,j]] == "B", Print["Has perdido"]];$ $\text{ocultas} = \text{juego};$ | <p>Si hay una bomba, descubrimos todo el tablero o indicamos que se ha perdido.</p> |
| $,$ | <p>En otro caso.</p> |
| $If[juego[[i,j]] > 0,$ $\text{ocultas}[[i,j]] = \text{juego}[[i,j]];$ $,$ $\text{ocultas}[[i,j]] = \text{juego}[[i,j]];$ $If[i-1 > 0,$ $\text{Mostrar}[i-1,j];$ $If[j-1 > 0, \text{Mostrar}[i-1,j-1]];$ $If[j+1 \leq Dimensions[juego][[2]],$ $\text{Mostrar}[i-1,j+1]];$ $]$ $If[i+1 \leq Dimensions[juego][[1]],$ | <p>Si la coordenada (i, j) de “juego” es distinta de cero la destapamos, almacenándola en “ocultas”, si es cero la almacenamos y destapamos todas las de su alrededor, recursivamente.</p> |

| | |
|--|---|
| <pre> Mostrar[i+1,j]; If[j-1>0,Mostrar[i+1,j-1];]; If[j+1≤Dimensions[juego][[2]], Mostrar[i+1,j+1];];]; If[j-1>0,Mostrar[i,j-1];]; If[j+1≤Dimensions[juego][[2]], Mostrar[i,j+1];];];]; ,</pre> | |
| <pre> Print["Coordenadas erróneas: (",i,",",j,")"];]; ;</pre> | En el caso en que las coordenadas sean erróneas lo indicamos. |

Función 1.5. Buscaminas II.

| FUNCIÓN | COMENTARIOS |
|---|---|
| Levantar[i,j]:=Module[{}, | Indicamos las coordenadas de la casilla que vamos a descubrir. |
| Mostrar[i,j]; | Llamamos a la función recursiva Mostrar[] para que se descubra. |
| Print[TableForm[ocultas,TableHeadings→Automatic, TableSpacing→{1,1},TableAlignments→Center]]; Print["Bombas: ",numbombas, ". Marcadas: ",marcadas];]; | Mostramos en pantalla la información. |
| | |

Función 1.6. Buscaminas III.

| FUNCIÓN | COMENTARIOS |
|---|---|
| Marcar[i,j]:=Module[{k1,k2}, If[ocultas[[i,j]]=="B",ocultas[[i,j]]="X";marcadas--, If[ocultas[[i,j]]=="X", ocultas[[i,j]]="B"; marcadas++];], Print["Casilla destapada"]];]; | Los argumentos serán las coordenadas de la casilla. Si previamente la habíamos marcado con una “B”, la desmarcamos. En caso contrario, comprobamos que no esté destapada dicha casilla y la marcamos con una “B”. |
| Print[TableForm[ocultas,TableHeadings→Automatic, TableSpacing→{1,1},TableAlignments→Center]]; Print["Bombas: ",numbombas, ". Marcadas: ",marcadas]; | Mostramos en pantalla la información. |
| If[numbombas==marcadas, ganar=True; Do[Do[If[juego[[k1,k2]]=="B" && ocultas[[k1,k2]]!="B", ganar=False]; ,{k1,Dimensions[juego][[1]]}]; ,{k2,Dimensions[juego][[2]]}]; If[ganar,Print["Has ganado"]]; | Comprobamos si el número de bombas marcadas coincide con el número de bombas total, en caso afirmativo determinamos si las hemos marcado correctamente y mostramos en pantalla si nos hemos equivocado o por el contrario hemos ganado. |

```

    , Print["Bombas mal marcadas"];
  ];
];

```

Función 1.7. Buscaminas IV.

Pongamos un ejemplo, primero creamos el juego:

In[]:= **NuevoJuego[12,12,15]**

Out[]=

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|----|---|---|---|---|---|---|---|---|---|----|----|----|
| 1 | X | X | X | X | X | X | X | X | X | X | X | X |
| 2 | X | X | X | X | X | X | X | X | X | X | X | X |
| 3 | X | X | X | X | X | X | X | X | X | X | X | X |
| 4 | X | X | X | X | X | X | X | X | X | X | X | X |
| 5 | X | X | X | X | X | X | X | X | X | X | X | X |
| 6 | X | X | X | X | X | X | X | X | X | X | X | X |
| 7 | X | X | X | X | X | X | X | X | X | X | X | X |
| 8 | X | X | X | X | X | X | X | X | X | X | X | X |
| 9 | X | X | X | X | X | X | X | X | X | X | X | X |
| 10 | X | X | X | X | X | X | X | X | X | X | X | X |
| 11 | X | X | X | X | X | X | X | X | X | X | X | X |
| 12 | X | X | X | X | X | X | X | X | X | X | X | X |

Bombas: 15. Marcadas: 0

Ahora empezamos a levantar y marcar casilleros, por ejemplo la posición (5, 1):

In[]:= **Levantar[5,1]**

Out[]=

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|----|---|---|---|---|---|---|---|---|---|----|----|----|
| 1 | X | X | X | 1 | 0 | 0 | 0 | 0 | 1 | X | X | X |
| 2 | X | X | X | 1 | 0 | 0 | 0 | 0 | 1 | X | X | X |
| 3 | 1 | 2 | 2 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | X | X |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | X | X | X |
| 5 | 0 | 0 | 0 | 0 | 1 | 2 | 2 | 2 | X | X | X | X |
| 6 | 0 | 0 | 0 | 0 | 2 | X | X | X | X | X | X | X |
| 7 | 0 | 0 | 0 | 0 | 2 | X | X | X | X | X | X | X |
| 8 | 0 | 0 | 0 | 0 | 2 | X | X | X | X | X | X | X |
| 9 | 0 | 0 | 1 | 1 | 2 | X | X | X | X | X | X | X |
| 10 | 0 | 0 | 1 | X | X | X | X | X | X | X | X | X |
| 11 | 1 | 1 | 2 | X | X | X | X | X | X | X | X | X |
| 12 | X | X | X | X | X | X | X | X | X | X | X | X |

Bombas: 15. Marcadas: 0

Si tuviésemos la fortuna, como en el ejemplo, de elegir un casillero con cero bombas alrededor descubriríamos muchos otros casilleros y a la vista de la información que aquí tenemos, deducimos que al menos en (2, 2), (2, 3), (6, 6), (7, 6), (6, 7), (4, 9), (10, 4) seguro que hay una bomba, por otro lado también podemos descubrir al menos los casilleros (1, 1), (1, 2), (1, 3), (2, 1), (5, 9), (6, 8), (8, 6), (10, 5), (11, 4) porque no pueden ocultar bombas, esto es, escribiríamos: Marcar[2,2], Marcar[2,3], Marcar[6,6], Marcar[7,6],... y Levantar[1,1], Levantar[1,2], Levantar[1,3], Levantar[2,1],...; con la nueva información volveríamos a reiterar el proceso, y si conseguimos marcar bien todas las bombas, ganaremos.

□

4. LA VERSIÓN DE MATHEMATICA 5.2

Todos los programas y funciones de la primera edición de este libro fueron diseñados inicialmente para la versión 5.2 de Mathematica. Posteriormente apareció la versión 6, ésta presenta algunas diferencias e incompatibilidades respecto a la versión 5.2. El texto y los programas que aparecen en el libro se han adaptado para que puedan usarse con la versión 6 o posteriores de Mathematica sin problemas, pero en algunos casos esto ha provocado la incompatibilidad de los mismos métodos en las versiones anteriores. En el disco de datos que acompaña a este texto, se han incluido ambas versiones por si el lector prefiriera usar la versión 5.2. o sólo dispusiera de esta versión. Resumimos algunas de las diferencias más importantes entre ambas versiones que afectan al contenido de este libro:

- El paquete de funciones que cargamos con: <<Combinatorica`>, en la versión 5.2 lo cargaremos de forma distinta con: <<DiscreteMath`Combinatorica`>. Este paquete ha cambiado considerablemente, incluyendo multitud de nuevas funciones que no funcionarán en la versión 5.2 y que utilizaremos en este libro. Entre ellas encontraremos varias para manejar objetos de tipo grafo, que son exclusivos de la versión 6 o posteriores, por tanto, no podremos usar ninguna función que trabaje sobre este tipo de objetos en versiones anteriores a la 6, incluida la 5.2. Este cambio además afectará a todos aquellos métodos del libro que utilicen funciones de esta librería o paquete, aunque ya aparecieran en la antigua: KSubsets[], SymmetricGroup[], ToCycles[], FromCycles[], SignaturePermutation[], AlternatingGroup[], InversePermutation[],...
- El paquete de funciones: <<GraphUtilities, en la versión 5.2 se llamaba o lo cargamos con: <<DiscreteMath`GraphPlot`>. Afectará a la representación de grafos con GraphPlot[], esta función es muy distinta, aunque conserve el mismo nombre, han cambiado casi todos los nombres de las distintas opciones gráficas de representación. Hay otras diferencias que no merece la pena comentar, algunas muy sútiles y otras que afectan a funciones que no se usan en este libro o aparecen puntualmente y no son de uso generalizado. A partir de la versión 6, la función GraphPlot[], no está incluida en ningún paquete a diferencia de la 5.2. Otras funciones de este paquete que aparecen en el libro son: StrongComponents[], GraphDistance[], TreePlot[], GraphCoordinates[],...

5. EJERCICIOS

Ejercicio 1.1. Completar el programa 1.3. para que represente todos los astros del sistema solar interno, considerar órbitas circulares en la eclíptica y los datos de la siguiente tabla:

| Planeta | Distancia al Sol | Periodo de orbital |
|----------|--------------------------|----------------------|
| Mercurio | 57910000 Km. o 0,387 UA | 87 días y 23,3 horas |
| Venus | 108208930 Km. o 0,723 UA | 224,701 días |
| Tierra | 149597870 Km. o 1 UA | 365,2564 días |
| Marte | 227936640 Km. o 1,52 UA | 686,98 días |

Tabla 1.1. Órbitas planetarias.



Ejercicio 1.2. Observando la función 1.2. crear otra función que calcule el número de días comprendidos entre dos fechas cualesquiera.



Ejercicio 1.3. La función 1.2. sólo funciona para fechas posteriores al 01/01/1980, hacer las modificaciones oportunas para que funcione siempre.



Ejercicio 1.4. Programar una función que represente el calendario anual correspondiente a un año cualquiera que introduzcamos.



Ejercicio 1.5. Definir una matriz cuyos valores representen los píxeles de una pantalla de ordenador, sabiendo que la pantalla tiene una resolución de 1024×768 y una profundidad de color de 8 bits. Si la pantalla muestra un fondo de color negro y un círculo centrado con un radio de 300 píxeles de color blanco, ¿cómo será la matriz?



Ejercicio 1.6. Crear las funciones necesarias para recrear el juego de las tres en raya. ¿Cómo programaríamos el ordenador para que fuese uno de los jugadores?



Ejercicio 1.7. Representar gráficamente con Mathematica la siguiente ilustración:

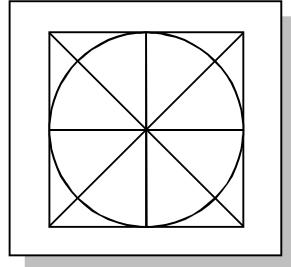


Ilustración 1.2.



Ejercicio 1.8. Representar gráficamente con Mathematica, utilizando la función Line[], un cubo y un octaedro.



2. GRUPOS

Son evidentes los problemas y las incompatibilidades que habitualmente presentan los conjuntos infinitos en la computadora. Aunque en ocasiones es posible describir estructuras algebraicas infinitas a partir de conjuntos o propiedades finitas, como es el caso de los espacios vectoriales finitos, donde para una base finita cualquiera de vectores del espacio vectorial, pueden describirse todos los vectores del espacio vectorial por sus coordenadas, únicas respecto de la base (véase con más detalle en [39] y su implementación en Mathematica, de forma muy básica, puede encontrarse en [43]). Para algunos grupos también puede hacerse, cuando un grupo puede ser descrito por un conjunto finito de elementos y un conjunto también finito de relaciones entre estos elementos, entonces se dice que el grupo es finitamente presentado y a los conjuntos de generadores y relaciones que lo determinan se les llama presentación del grupo. Cuando disponemos de un grupo finitamente presentado, aunque sea infinito, el ordenador sí que permite, con sus limitaciones, estudiar estos grupos infinitos, sin embargo este estudio desde el punto de vista planteado en este libro es bastante profuso y queda fuera de los objetivos y el perfil del mismo, si el lector lo desea puede profundizar más, por ejemplo en [11], [18], [30], [41] y [46].

Nos centramos en el estudio de los grupos finitos que no presentan problema alguno y se resuelven completamente, si bien aunque su implementación esté resuelta, dar respuesta a algunas preguntas para grupos con un número de elementos suficientemente elevado, implica una cantidad de proceso incontrolable. En el caso infinito nos limitaremos a presentar sólo algunas consideraciones sobre los mismos.

Este es el primer capítulo que dedicamos a la teoría de grupos, en él pretendemos analizar la estructura algebraica de grupo computacionalmente y profundizar sobre las nociones inherentes al concepto de grupo mismo ayudándonos de un ordenador. Obviamos la existencia de cualquier otro material preexistente acerca del tema de grupos y construimos desde cero una colección de pequeñas rutinas que permiten estudiar la estructura de grupo en conjuntos finitos, todas ellas desarrolladas con las herramientas informáticas habituales de cualquier lenguaje de programación, si bien, la notación está adaptada a Mathematica.

Recordemos la definición de grupo:

Un grupo es un par $(G, *)$ formado por un conjunto $G \neq \emptyset$ y una ley de composición interna $*: G \times G \rightarrow G$ verificando las siguientes propiedades:

- i. Elemento Neutro: Existe $e \in G$ tal que $e * a = a * e = a$ para cada $a \in G$.

- ii. Elemento simétrico: Para cada $a \in G$, existe $a' \in G$ tal que $a * a' = a' * a = e$.
- iii. Asociativa: $(a * b) * c = a * (b * c)$ para cada $a, b, c \in G$.

Se dirá que es un grupo abeliano o commutativo si además verifica:

- iv. Commutativa: $a * b = b * a$ para cada $a, b \in G$.

1. GRUPOS FINITOS

Sea $(G, *)$ un grupo, si el conjunto G es finito se dice que es un grupo finito, si tiene n elementos diremos que es de orden n y escribiremos: $|G| = n$.

Para un grupo finito con n elementos podemos representar la ley de composición interna por una tabla de doble entrada de n filas por n columnas:

Si $G = \{g_1, g_2, \dots, g_n\}$, tendríamos:

| * | g_1 | g_2 | \dots | g_n |
|----------|-------------|-------------|----------|-------------|
| g_1 | $g_1 * g_1$ | $g_1 * g_2$ | \dots | $g_1 * g_n$ |
| g_2 | $g_2 * g_1$ | $g_2 * g_2$ | \dots | $g_2 * g_n$ |
| \vdots | \vdots | \vdots | \ddots | \vdots |
| g_n | $g_n * g_1$ | $g_n * g_2$ | \dots | $g_n * g_n$ |

Tabla 2.1. Tabla de operaciones de un grupo.

Así, podríamos introducir un grupo finito en el ordenador de forma directa, el conjunto como una lista de elementos y la operación como una tabla de doble entrada:

In[]:= **G={g1,g2,...,gn};**
operacion= TABLA DE TODAS LAS OPERACIONES;

Ejemplo 2.1. Sea $G = \{a, b, c, d\}$ y $*$ la ley de composición interna dada por:

| * | a | b | c | d |
|-----|-----|-----|-----|-----|
| a | a | b | c | d |
| b | b | a | d | c |
| c | c | d | b | a |
| d | d | c | a | b |

Tabla 2.2.

Que introduciremos en el ordenador directamente:

In[]:= **G={a,b,c,d};**
operacion={a,b,c,d},{b,a,d,c},{c,d,b,a},{d,c,a,b}};

Obsérvese que, por ejemplo, $c * b$ vendría dado por $\text{operacion}[[3,2]]$ y $d * a$ vendría dado por $\text{operacion}[[4,1]]$. □

En general si $G = \{x_1, x_2, \dots, x_n\}$, entonces $x_i * x_j$ sería $G[[i]] * G[[j]]$, y lo calcularíamos con $\text{operacion}[[i,j]]$. Para facilitar el cálculo podemos definir funciones que operen dos elementos cualesquiera directamente y no a través de su índice. Damos dos definiciones, la opción 1 es más larga pero más intuitiva y la opción 2 se basa en la función `Position[]` de Mathematica:

| FUNCIÓN | COMENTARIOS |
|---|--|
| G=GRUPO; operacion=TABLA DE OPERACIONES DE G; | Introducimos el grupo y su tabla de operaciones. |
| op[x_,y_]:=Module[{CONTADORi,CONTADORj}, CONTADORi=1; While[Intersection[{x},{G[[CONTADORi]]}]=={}, CONTADORi++]; CONTADORj=1; While[Intersection[{y},{G[[CONTADORj]]}]=={}, CONTADORj++]; operacion[[CONTADORi,CONTADORj]]]; | La función tendrá dos entradas: dos elementos cualesquiera del grupo. Utilizamos bucles para encontrar los elementos del grupo. |
| | Salida de resultados. |

Función 2.1. Operación de un grupo.

| FUNCIÓN | COMENTARIOS |
|--|--|
| G=GRUPO; operacion=TABLA DE OPERACIONES DE G; | Introducimos el grupo y su tabla de operaciones. |
| op[x_,y_]:= operacion[[Position[G,x][[1]], Position[G,y][[1]]][[[1]][[1]]]; | La función tendrá dos entradas: dos elementos cualesquiera del grupo. Usamos la función <code>Position[]</code> . |

Función 2.1. bis. Operación de un grupo.

Ahora, estudiamos cada propiedad por separado y programamos rutinas que comprueben si se verifican cada una de ellas.

1.1. OPERACIÓN INTERNA

Aunque a la vista de la tabla “operacion” puede resultar obvio si la operación es **internal** o no, esto es, que $a * b \in G$ para cada $a, b \in G$, sin embargo es conveniente generar una pequeña rutina que haga esta comprobación para casos no evidentes o donde la tabla venga dada de forma algorítmica.

| PROGRAMA | COMENTARIOS |
|--|------------------------------|
| G=GRUPO; operacion=TABLA DE OPERACIONES DE G; | Introducimos el grupo. |
| op[x_,y_]:=... | Introducimos la función 2.1. |

| | |
|---|--|
| operacioninterna=True; | Suponemos de partida que sí se verifica. |
| CONTADORi=1; While[operacioninterna && CONTADORi<=Length[G], CONTADORj=1; While[operacioninterna &&CONTADORj<=Length[G], If[Intersection[{op[G][[CONTADORi]], G[[CONTADORj]]}],G]=={}, operacioninterna=False]; CONTADORj++;]; CONTADORi++;]; | Comprobamos si en algún caso no se verifica. |
| operacioninterna | Se muestran los resultados. |

Programa 2.2. Operación interna.

Nótese que $\text{operacion}[[i,j]]$ puede sustituirse por $\text{op}[G[[i]],G[[j]]]$, si bien, esta última expresión necesita de la función 2.1.

También podríamos realizar un test basado en operaciones entre conjuntos:

| FUNCIÓN | COMENTARIOS |
|---|---|
| INTERNA[G_,operacion_]:=Module[{CONTADORi,interna}, interna={}; Do[interna=Union[operacion[[CONTADORi]],internal] ,{CONTADORi,Length[operacion]}]; Length[Union[G,interna]]==Length[G]] | Comprobamos si es ley de composición interna. |

Función 2.3. Operación Interna.

En donde unimos todos los elementos de la tabla “operación” y comprobamos que están contenidos en G , este último método es más corto y directo pero menos intuitivo que el primero (programa 2.2.).

Ejemplo 2.2. Consideramos el mismo conjunto y operación interna del ejemplo 2.1. Comprobamos que, en efecto, dicha operación es interna.

Introducimos el grupo y la operación:

$$\text{In}[]:= \quad \text{G}=\{\text{a,b,c,d}\}; \\ \text{operacion}=\{\{\text{a,b,c,d}\},\{\text{b,a,d,c}\},\{\text{c,d,b,a}\},\{\text{d,c,a,b}\}\};$$

Introducimos 2.1. o 2.1.bis.

$$\text{In}[]:= \quad \text{op}[\text{x}_\text{,y}_\text{}]:=\text{operacion}[[\text{Position}[\text{G},\text{x}][[1]], \\ \text{Position}[\text{G},\text{y}][[1]]][[1]][[1]]];$$

Usamos el programa 2.2.:

In[]:= **operacioninterna=True;**

⋮ ⋮

Out[]= True

Análogamente, utilizando 2.3.:

In[]:= **INTERNA[G,_operacion_]:=Module[**

⋮ ⋮

In[]:= **INTERNA[G,operacion]**

Out[]= True

□

1.2. ELEMENTO NEUTRO

El elemento neutro, si existe, es único, luego bastará con buscar un elemento que verifique la propiedad del elemento neutro.

| PROGRAMA | COMENTARIOS |
|--|--|
| G=GRUPO; operacion=TABLA DE OPERACIONES DE G; | Introducimos el grupo. |
| op[x_,y_]:=... | Introducimos la función 2.1. |
| ElementoNeutro="No existe"; | Suponemos de partida que no existe. |
| Do[If[ElementoNeutro=="No existe", neutro=True; Do[If[TrueQ[op[G[[CONTADORi]],G[[CONTADORj]]] ==op[G[[CONTADORj]],G[[CONTADORi]]]] && TrueQ[op[G[[CONTADORi]],G[[CONTADORj]]] ==G[[CONTADORj]]] ,,neutro=False;Break[]]; {,CONTADORj,1,Length[G]}]; If[neutro, ElementoNeutro=G[[CONTADORi]];Break[]];]; {,CONTADORi,1,Length[G]}]; | <p>Se comprueba la propiedad que debe verificar un elemento para ser el neutro.</p> <p>Bucle que recorre todos los elementos de G para comprobar si alguno de ellos es el neutro.</p> |

| | |
|--|-----------------------------|
| Print["Elemento Neutro: ",ElementoNeutro] | Se muestran los resultados. |
| Programa 2.4. Elemento neutro. | |

Otra forma de determinar el elemento neutro es directamente fijándonos en la tabla de operaciones, la lista G debe de aparecer, respetando el orden original, como la fila y la columna i -ésima de la tabla, siendo entonces el i -ésimo elemento de G su elemento neutro. Para ello usaremos la función⁵ de Mathematica: Transpose[]. Integraremos estas ideas en la función NEUTRO[] (2.5.):

| FUNCIÓN | COMENTARIOS |
|---|---|
| NEUTRO[G_,operacion_]:=Module[{CONTADORi}, Neutro="No existe"; Do[If[TrueQ[operacion[[CONTADORi]]==G] && TrueQ[Transpose[operacion][[CONTADORi]]== operacion[[CONTADORi]]] ,Neutro=G[[CONTADORi]];Break[]]; ,{CONTADORi,Length[G]}]; | La función tendrá dos entradas: el conjunto y la tabla de operaciones que definen a un grupo. |
| Neutro]] | Se busca el neutro. |
| | Se muestran los resultados. |

Función 2.5. Elemento neutro.

Igual que ocurre en el apartado 1.1. (para la operación interna) este segundo método es más corto y directo, pero menos intuitivo que el primero (programa 2.4.) que es una fiel traslación del concepto.

Ejemplo 2.3. Consideramos el mismo conjunto y operación interna del ejemplo 2.1.

```
In]:= G={a,b,c,d};  
operacion={{a,b,c,d},{b,a,d,c},{c,d,b,a},{d,c,a,b}};  
op[x_,y_]:=operacion[[Position[G,x][[1]],  
Position[G,y][[1]]]][[1]][[1]];
```

Utilizamos el programa 2.4. y determinamos el elemento neutro.

```
In]:= ElementoNeutro="No existe";
```

⋮ ⋮

```
Out]= Elemento Neutro: a
```

Y usando 2.5.:

```
In]:= NEUTRO[G_,operacion_]:=Module[{CONTADORi},
```

⋮ ⋮

⁵ Esta función calcula la matriz traspuesta, esto es, la matriz que resulta de permutar filas por columnas o viceversa. Para más detalle sobre la función Transpose[] véase la ayuda de Mathematica.

In[]:= **NEUTRO[G,operacion]**

Out[*] =*

1.3. ELEMENTO SIMÉTRICO

Al igual que le ocurre al neutro, si existe el simétrico de un elemento, este es único. Calculamos, si existen, los elementos simétricos de todos los elementos de G .

| COMENTARIOS | |
|---|--|
| ELEMENTOSIMETRICO[G,_operacion_]:=Module[{CONTADORi,CONTADORj,Nosim,Neutro,op, ElementoSimetrico,salida}, op[x_,y_]:=operacion[[Position[G,x][[1]], Position[G,y][[1]]][[1]][[1]]]; Neutro="No existe"; Do[If[TrueQ[operacion[[CONTADORi]]==G] && TrueQ[Transpose[operacion][[CONTADORi]] ==operacion[[CONTADORi]]] ,Neutro=G[[CONTADORi]];Break[]]; ,{CONTADORi,Length[G]}]; salida=False; If[Neutro=="No existe", Print["Error: no se verifica la propiedad del elemento neutro."], Print["Elemento neutro: ",Neutro]; ElementoSimetrico={G,Table["-", {CONTADORi,1,Length[G]}]}; Nosim={}; Do[Do[If[TrueQ[op[G[[CONTADORi]],G[[CONTADORj]]]== op[G[[CONTADORj]],G[[CONTADORi]]]] && TrueQ[op[G[[CONTADORi]],G[[CONTADORj]]]== ElementoNeutro] , | Funcin cuyos argumentos sern: el conjunto y la tabla de operaciones del par candidato a grupo. Funcin 2.1. Calculamos, si existe, el elemento neutro del candidato a grupo con el mismo cdigo de la funcin 2.5. Inicializamos el valor de "salida" a "False", slo si existe el elemento neutro y verifica la propiedad de elemento simtrico, cambiar a "True". Comprobamos que exista el elemento neutro. Si el neutro no existe, se informa del error. Si el neutro existe, se informa del mismo y se continan los cculos. Generamos las tablas donde almacenaremos los simtricos. Bucle que recorre todos los elementos de G para calcular el simtrico de cada uno de ellos. |

| | |
|--|-----------------------------|
| <pre>]; ,{CONTADORj,1,Length[G]}; If[ElementoSimetrico[[2,CONTADORi]]=="-", AppendTo[Nosim,G[[CONTADORi]]]; ,{CONTADORi,1,Length[G]}]; Print[TableForm[ElementoSimetrico, TableHeadings→>{"Elementos:","Simétricos:", Automatic}]]; If[Nosim=={},salida=True;];]; salida]; </pre> | |
| | Se muestran los resultados. |

Función 2.6. Elemento simétrico.

Ejemplo 2.4. Consideramos el mismo conjunto y operación interna del ejemplo 2.1.

In[]:= $G=\{a,b,c,d\};$
operacion={ $\{a,b,c,d\}, \{b,a,d,c\}, \{c,d,b,a\}, \{d,c,a,b\}$ };

Calculamos, si existen, todos los simétricos con 2.6. y comprobamos si se verifica esta propiedad:

In[]:= **ELEMENTOSIMETRICO[G_,operacion_]:=Module[**

⋮ ⋮

In[]:= **ELEMENTOSIMETRICO[G,operacion]**

Out[] = Elemento neutro: a

| | 1 2 3 4 |
|------------|---------|
| Elementos: | a b c d |
| Simétricos | a b d c |

True

□

También programamos una función que determine el simétrico de un elemento en particular, esta función la usaremos frecuentemente más adelante y será, en ciertos casos, más cómoda de utilizar que la función 2.6.

| FUNCIÓN | COMENTARIOS |
|---|---|
| G=GRUPO; operacion=TABLA DE OPERACIONES DE G; | Introducimos el grupo “G”. |
| op[x_,y_]:=... | Introducimos la función 2.1. |
| ElementoNeutro=... | Calculamos el elemento neutro con el programa 2.4. |
| simetrico[x_]:=Module[{simetrico,CONTADORi}, simetrico=0; | La función tendrá como entrada un elemento “x” del grupo. |
| Do[| Opera “x” con todos los elementos de “G” |

| | |
|---|--|
| If[op[x,G[[CONTADORi]]]== ElementoNeutro, simetrico=G[[CONTADORi]]; Break[]]; ,{CONTADORi,1,Length[G]}]; simetrico]; | hasta encontrar su simétrico. Se muestran los resultados. |
|---|--|

Función 2.7. Cálculo de un elemento simétrico.

Para mayor comodidad del lector, podemos incluir el cálculo del elemento neutro y la función 2.1. dentro del cuerpo de la función y añadirle como argumentos el conjunto y la operación del grupo, así obtendremos una versión más cómoda de usar para el lector, que no dependerá de ninguna otra definición previa, aunque también será menos eficaz, porque siempre repetirá el cálculo del elemento neutro:

| FUNCIÓN | COMENTARIOS |
|--|---|
| simetrico[x_,G_,operacion_]:=Module[{simetrico, CONTADORi,op,Neutro}, op[x1_,y1_]:=operacion[[Position[G,x1]][[1]], Position[G,y1]][[1]]][[1]][[1]]; Neutro="No existe"; Do[If[TrueQ[operacion[[CONTADORi]]==G] && TrueQ[Transpose[operacion][[CONTADORi]] ==operacion[[CONTADORi]]] ,Neutro=G[[CONTADORi]];Break[]]; ,{CONTADORi,Length[G]}]; If[Neutro=="No existe", Print["No tiene elemento neutro."]; , | La función tendrá como entrada un elemento “x” del grupo y el conjunto y la tabla de operaciones del grupo. Función 2.1. |
| simetrico="No tiene simétrico"; Do[If[op[x,G[[CONTADORi]]]== Neutro, simetrico=G[[CONTADORi]]; Break[]]; ,{CONTADORi,1,Length[G]}]; simetrico] | Calculamos, si existe, el elemento neutro del candidato a grupo con el mismo código de la función 2.5. |
| | Si existe el elemento neutro procedemos a calcular el simétrico de “x”. |
| | Opera “x” con todos los elementos de “G” hasta encontrar su simétrico. |
| | Se muestran los resultados. |

Función 2.7.bis. Cálculo de un elemento simétrico.

Ejemplo 2.5. Consideramos el mismo conjunto y operación interna del ejemplo 2.1., y calculamos el elemento simétrico de a y d .

Introducimos el conjunto G , su operación interna y definimos la función 2.1. como hacemos siempre.

In[]:= $G=\{a,b,c,d\};$
 $\text{operacion}=\{\{a,b,c,d\},\{b,a,d,c\},\{c,d,b,a\},\{d,c,a,b\}\};$

```
op[x_,y_]:=operacion[[Position[G,x][[1]],
Position[G,y][[1]]][[1]][[1]];
```

Determinamos el elemento neutro con 2.4.

In[]:= ElementoNeutro="No existe";

⋮ ⋮

Definimos la función 2.7.

In[]:= simetrico[x_]:=Module[{simetrico,CONTADORi},

⋮ ⋮

La aplicamos a los elementos que nos interesan,

In[]:= simetrico[a]

Out[] = a

y

In[]:= simetrico[d]

Out[] = c

Y en efecto:

In[]:= op[a,a]

Out[] = a

In[]:= op[d,c]

Out[] = a

Por último, si queremos usar 2.7.bis., no será necesario definir 2.1. y usar 2.4., directamente escribiremos:

In[]:= simetrico[x_,G_,operación_]:=Module[{simetrico,

⋮ ⋮

In[]:= simetrico[a,G,operacion]

Out[] = a

In[]:= simetrico[d,G,operacion]

Out[]:= c

□

Al igual que en las propiedades anteriores, podríamos determinar el simétrico de un elemento fijándonos directamente en la tabla (ejercicio 2.9.).

1.4. ASOCIATIVA

Bastará con programar tres bucles anidados que recorran todas las posibles ternas de tres elementos de G y se compruebe si se verifica la propiedad asociativa para cada una de ellas.

| FUNCIÓN | COMENTARIOS |
|---|---|
| <pre>ASOCIATIVA[G_,operacion_]:=Module[{CONTADORi,CONTADORj,CONTADORk, Asociativa,op},</pre> | Funció n cuyos argumentos serán: el conjunto y la tabla de operaciones del par candidato a grupo. |
| <pre>op[x_,y_]:=operacion[[Position[G,x][[1]], Position[G,y][[1]]][[1]][[1]];</pre> | Función 2.1. |
| <pre>asociativa=True;</pre> | Suponemos de partida que sí se verifica. |
| <pre>CONTADORi=1; While[asociativa && CONTADORi≤Length[G], CONTADORj=1; While[asociativa && CONTADORj≤Length[G], CONTADORk=1; While[asociativa && CONTADORk≤Length[G], If[TrueQ[op[op[G[[CONTADORi]], G[[CONTADORj]],G[[CONTADORk]]] == op[G[[CONTADORi]],op[G[[CONTADORj]],G[[CONTADORk]]]] ,,,asociativa=False]; CONTADORk++;]; CONTADORj++;]; CONTADORi++;];</pre> | Comprobamos si en algún caso no se verifica. |
| <pre>asociativa</pre> | Se muestran los resultados. |

Función 2.8. Propiedad Asociativa.

Ejemplo 2.6. Consideramos el mismo conjunto y operación interna del ejemplo 2.1.

In[]:=

G={a,b,c,d};

operacion={{a,b,c,d},{b,a,d,c},{c,d,b,a},{d,c,a,b}};

Utilizamos el programa 2.8. para comprobar la asociatividad.

In[]:= ASOCIATIVA[G_,operacion]:=Module[

⋮ ⋮

In[]:= ASOCIATIVA[G,operacion]

Out[] = True

Luego como la operación es asociativa, por los ejemplos 2.2., 2.3. y 2.4., podemos asegurar que G es un grupo. \square

1.5. CONMUTATIVA

Por último, aunque puede que sea obvio observando la simetría de la tabla de operaciones, también queremos comprobar si es un grupo conmutativo. El siguiente programa es similar al 2.8.

| FUNCIÓN | COMENTARIOS | |
|---|--|--|
| CONMUTATIVA[G_,operacion]:=Module[{CONTADORi,CONTADORj,comutativa,op}, | Funció n cuyos argumentos serán: el conjunto y la tabla de operaciones del par candidato a grupo. | |
| op[x,y]:=operacion[[Position[G,x][[1]], Position[G,y][[1]]][[[1]][[1]]]; | Función 2.1. | |
| comutativa=True; | Suponemos de partida que sí se verifica. | |
| CONTADORi=1; While[comutativa && CONTADORi<=Length[G], CONTADORj=1; While[comutativa && CONTADORj<=Length[G], If[TrueQ[op[G[[CONTADORi]],G[[CONTADORj]]]== op[G[[CONTADORj]],G[[CONTADORi]]]] ,,comutativa=False]; CONTADORj++;]; CONTADORi++;]; | Comprobamos si en algún caso no se verifica. Equivalentemente podemos escribir únicamente: operacion== Transpose[operacion] | |
| comutativa | Se muestran los resultados. | |
|] | | |

Función 2.9. Propiedad Conmutativa.

Ejemplo 2.7. Consideramos el mismo conjunto y operación interna del ejemplo 2.1.

In[]:= G={a,b,c,d};

operacion={{{a,b,c,d},{b,a,d,c},{c,d,b,a},{d,c,a,b}}};

Usando el programa 2.9. comprobamos la propiedad conmutativa.

In[]:= CONMUTATIVA[G_,operacion_]:=Module[

⋮ ⋮

In[]:= CONMUTATIVA[G,operacion]

Out[] = True

Por tanto G es un grupo conmutativo. \square

También podríamos comprobar la propiedad conmutativa directamente, como se comenta en la función 2.9., de manera más sencilla, verificando la simetría de la tabla “operacion” con Mathematica,

Transpose[operacion]==operación

1.6. TEST PARA GRUPOS FINITOS

Por último juntamos las funciones anteriores para comprobar directamente si un conjunto con una operación es grupo o grupo abeliano.

**In[]:= GRUPO[Conjunto_,Operacion_]:=INTERNA[Conjunto,
Operacion] && ELEMENTOSIMETRICO[Conjunto,Operacion]
&& ASOCIATIVA[Conjunto,Operacion];**

In[]:= GRUPOABELIANO[Conjunto_,Operacion_]:=
INTERNA[Conjunto,Operacion] &&
ELEMENTOSIMETRICO[Conjunto,Operacion] &&
ASOCIATIVA[Conjunto,Operacion] &&
CONMUTATIVA[Conjunto,Operacion];

Observemos que, en las funciones anteriores, no hacemos uso de ninguno de los programas relativos al elemento neutro, porque la función 2.6. realiza esa comprobación previamente, ya que sin elemento neutro no podemos determinar los elementos simétricos.

2. HOMOMORFISMOS DE GRUPOS

Sean $(G_1, *_1)$ y $(G_2, *_2)$ dos grupos, una aplicación $f: G_1 \rightarrow G_2$ se dirá que es un homomorfismo de grupos si verifica:

$$f(x_1 *_1 x_2) = f(x_1) *_2 f(x_2), \text{ para cada } x_1, x_2 \in G_1.$$

Si el homomorfismo f es biyectivo, entonces se dirá que f es un isomorfismo de grupos y también que G_1 y G_2 son grupos isomorfos.

Las aplicaciones entre conjuntos y en particular los homomorfismos de grupos, en Mathematica pueden introducirse directamente:

F[x_]:=Definición del homomorfismo de grupos;

Y será así, siempre que tengamos los grupos bien definidos en Mathematica, la definición del homomorfismo lo permita y sea compatible con Mathematica⁶, esto es, pueda implementarse. Como nos sucede en las secciones anteriores, el cálculo computacional para casos infinitos, dependerá del ejemplo particular con el que trabajemos, sin embargo, para casos finitos sí que podremos resolver el problema en general. A continuación damos una solución para el caso finito.

Como vamos a usar dos grupos finitos distintos, alteraremos la notación habitual que venimos usando para trasladar la estructura de grupo al ordenador. Dos grupos $(G_1, *_1)$ y $(G_2, *_2)$ los definiremos en Mathematica como sigue:

*In[]:= G1=CONJUNTO G₁;
operacion1=TABLA DE OPERACIONES DE G₁;*

*In[]:= op1[x_,y_]:=operacion1[[Position[G1,x][[1]],
Position[G1,y][[1]]][[1]][[1]]];*

y

*In[]:= G2= CONJUNTO G₂;
operacion2= TABLA DE OPERACIONES DE G₂;*

*In[]:= op2[x_,y_]:=operacion2[[Position[G2,x][[1]],
Position[G2,y][[1]]][[1]][[1]]];*

Introducimos una aplicación entre ambos, según el caso, de la forma que nos resulte más cómoda (para más detalle, véase como se usan en Mathematica las aplicaciones entre conjuntos en el capítulo 7 de [25]), por ejemplo la podemos definir así:

*In[]:= ImagenF=LISTADO DE IMÁGENES DE LOS
ELEMENTOS DE G1;
Do[
 F[G1[[CONTADORi]]]:=ImagenF[[CONTADORi]];
,{ CONTADORi,1,Length[G1]}];*

Creamos un test que compruebe si es homomorfismo:

⁶ Problemas análogos se presentarían con otros programas.

| PROGRAMA | COMENTARIOS |
|--|--|
| <pre>G1=CONJUNTO G₁; operacion1=TABLA DE OPERACIONES DE G₁; op1[x ,y]:=operacion1[[Position[G1,x]][[1]], Position[G1,y]][[1]]][[1]][[1]]; G2= CONJUNTO G₂; operacion2= TABLA DE OPERACIONES DE G₂; op2[x ,y]:=operacion2[[Position[G2,x]][[1]], Position[G2,y]][[1]]][[1]][[1]]; ImagenF={IMAGENES}; Do[F[G1[[CONTADORi]]]:=ImagenF[[CONTADORi]]; ,{ CONTADORi,1,Length[G1]}]; Homomorfismo=True;</pre> | Introducimos los grupos y la aplicación con las notaciones comentadas. |
| <pre>CONTADORi=1; While[Homomorfismo && CONTADORi<=Length[G1], CONTADORj=1; While[Homomorfismo && CONTADORj<=Length[G1], If[TrueQ[F[op1[G1[[CONTADORi]], G1[[CONTADORj]]]]] ==op2[F[G1[[CONTADORi]]],F[G1[[CONTADORj]]]]] ,,Homomorfismo=False]; CONTADORj++;]; CONTADORi++;]; Homomorfismo</pre> | Suponemos de partida que es homomorfismo. |
| | Comprobamos si la aplicación “F” es homomorfismo de grupos. |
| | Salida de resultados. |

Programa 2.10. Homomorfismos de grupos.

Para mayor comodidad podemos construir una función, pero según como definamos la aplicación entre los grupos, podemos tener dificultades para usar dicha aplicación como argumento de la función. Para no tener problemas, utilizaremos como entrada de la función el grafo de la aplicación, que es un conjunto finito y podemos introducir directamente o calcular fácilmente desde la aplicación como sigue:

```
In//]:=      grafoF={};
Do[
 AppendTo[grafoF,{G1[[CONTADORi]],
F[G1[[CONTADORi]]]}];
,{CONTADORi,1,Length[G1]}];
```

Y la función se definiría de la siguiente forma:

| FUNCIÓN | COMENTARIOS |
|---|--|
| HOMOMORFISMO[G1_,operacion1_, G2_,operacion2_,grafoF_]:=Module[| La función tendrá 5 entradas: conjunto y tabla de operaciones del primer y segundo |

| | |
|---|---|
| {CONTADORi,CONTADORj,op1,op2,F}, | grupo y el grafo de la aplicación definida entre ambos conjuntos. |
| op1[x_,y_]:=operación1[[Position[G1,x][[1]], Position[G1,y][[1]]][[1]][[1]]; | Función para operar en el primer grupo. |
| op2[x_,y_]:=operación2[[Position[G2,x][[1]], Position[G2,y][[1]]][[1]][[1]]; | Función para operar en el segundo grupo. |
| F[X_]:=Module{CONTADORi}, CONTADORi=1; While[grafoF CONTADORi][[1]]!=X, CONTADORi++]; grafoF[[CONTADORi]][[2]]]; | Recuperamos la aplicación desde su grafo. |
| Homomorfismo=True; | Suponemos de partida que es homomorfismo. |
| CONTADORi=1; While[Homomorfismo && CONTADORi<=Length[G1], CONTADORj=1; While[Homomorfismo && CONTADORj<=Length[G1], If[TrueQ[F[op1[G1[[CONTADORi]], G1[[CONTADORj]]]]] ==op2[F[G1[[CONTADORi]]], F[G1[[CONTADORj]]]], ,Homomorfismo=False]; CONTADORj++;]; CONTADORi++;]; | Comprobamos si la aplicación es homomorfismo de grupos. |
| Homomorfismo | Salida de resultados. |

Función 2.11. Homomorfismos de grupos.

Podemos usar lo expuesto en el capítulo 7 de [25] para comprobar la biyectividad y determinar si son isomorfos. Aunque bastaría, por la finitud de los conjuntos, con comprobar que $|G_1| = |G_2| = |Im(f)|$.

Ejemplo 2.8. Sea G_1 el grupo del ejemplo 2.1. y sea $G_2 = \{1, 2, 3, 4\}$ con tabla de operaciones,

| $*_2$ | 1 | 2 | 3 | 4 |
|-------|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 |
| 2 | 2 | 4 | 1 | 3 |
| 3 | 3 | 1 | 4 | 2 |
| 4 | 4 | 3 | 2 | 1 |

Tabla 2.3.

Sea $f: G_1 \rightarrow G_2$ la aplicación definida por,

$$\begin{aligned} f(a) &= 1, & f(b) &= 4, \\ f(c) &= 2, & f(d) &= 3. \end{aligned}$$

Respondemos a las siguientes cuestiones:

- a) Comprobar que $(G_2, *_2)$ es un grupo commutativo.
 - b) Comprobar que f es un homomorfismo de grupos.
 - c) Determinar si G_1 y G_2 son isomorfos.
 - d) Comprobar si $g: \mathbb{Z} \rightarrow \mathbb{Z}_2$, definida por $g(x) := \bar{x}$ es un homomorfismo de grupos.
 - e) Comprobar si la aplicación $h: G_2 \rightarrow \mathbb{Z}_2$, definida por $h(x) := \bar{x}$ es un homomorfismo de grupos.
 - f) Comprobar si la aplicación $i: \mathbb{Z}_8 \rightarrow \mathbb{Z}_4$, definida por $i(\bar{x}) := \bar{x}$ es un homomorfismo de grupos utilizando la función 2.11.
- a) Comprobamos que $(G_2, *_2)$ es un grupo. Introducimos el conjunto, la operación y la función 2.1.:

```
In[]:=          G= {1,2,3,4};  
              operacion= {{1,2,3,4},{2,4,1,3},{3,1,4,2},{4,3,2,1}};  
  
In[]:=          op[x_,y_]:=operacion[[Position[G,x][[1]],  
           Position[G,y][[1]]][[1]][[1]]];
```

- Utilizamos 2.2. para determinar si la operación es interna:

```
In[]:=          operacioninterna=True;
```

```
          :          :
```

```
Out[]=          True
```

- Usamos 2.4. para comprobar la existencia y calcular el elemento neutro.

```
In[]:=          ElementoNeutro="No existe";
```

```
          :          :
```

```
Out[]=          Elemento Neutro: 1
```

- Con 2.6., comprobamos si verifica la propiedad de elemento simétrico y determinamos todos los simétricos.

```
In[]:=          ELEMENTOSIMETRICO[G_,operacion_]:=
```

```
          :          :
```

In[]:= **ELEMENTOSIMETRICO[G,operacion]**

Out[]:= Elemento Neutro: 1

| | |
|------------|---------|
| Elementos: | 1 2 3 4 |
| Simétricos | 1 2 3 4 |
| True | |

- Comprobamos la propiedad asociativa con 2.8.

In[]:= **ASOCIATIVA[G_,operacion_]:=Module[**

⋮ ⋮

In[]:= **ASOCIATIVA[G,operacion]**

Out[]:= True

- Por último, la propiedad comutativa:

In[]:= **Transpose[operaciones]==operaciones**

Out[]:= True

- b) Introducimos G_1 y G_2 con las variaciones de notación indicadas para no tener problemas.

In[]:= **G1={a,b,c,d};
operacion1={{a,b,c,d},{b,a,d,c},{c,d,b,a},{d,c,a,b}};**

In[]:= **op1[x_,y_]:=operacion1[[Position[G1,x][[1]],
Position[G1,y][[1]]][[1]][[1]]];**

In[]:= **G2= {1,2,3,4};
operacion2={{1,2,3,4},{2,4,1,3},{3,1,4,2},{4,3,2,1}};**

In[]:= **op2[x_,y_]:=operacion2[[Position[G2,x][[1]],
Position[G2,y][[1]]][[1]][[1]]];**

Definimos la aplicación:

In[]:= **F[a]:=1;
F[b]:=4;
F[c]:=2;
F[d]:=3;**

Utilizamos el programa 2.10.

In[]:= **Homomorfismo=True;**

⋮ ⋮

Out[] = True

Por tanto, se trata de un homomorfismo de grupos.

- c) Determinamos si son isomorfos, para ello tendríamos que encontrar un isomorfismo de grupos entre ellos, pero el homomorfismo f del apartado anterior es biyectivo ya que $Im(f) = \{1, 2, 3, 4\}$.
- d) En este caso nos encontramos con un grupo infinito ($\mathbb{Z}, +$), que habitualmente se suele encontrar implementado en cualquier lenguaje de programación. Si queremos trasladar la definición del homomorfismo g a Mathematica, podemos hacerlo de la siguiente forma:

In[]:= **g[x_]:=0;**
g[x_Integer]:=Mod[x,2];

Como las operaciones son la ‘+’, lo definimos como 0 en cualquier elemento no entero para que no de problemas, y como el resto de dividir por 2 en los número enteros que constituyen el dominio de la aplicación. Ahora comprobamos si es homomorfismo:

In[]:= **g[x+y]==g[x]+g[y]**

Out[] = True

Y en efecto, es un homomorfismo de grupos. El ejemplo ha sido elegido con cierta intencionalidad, pues observemos que trabajamos con un grupo infinito y el ordenador no ha dado problemas. Desafortunadamente, no siempre será así, en este caso particular, el grupo y el homomorfismo que se han elegido, están perfectamente implementados en Mathematica. (En el epígrafe 3 de este capítulo se tratan otros ejemplos de grupos infinitos con Mathematica).

- e) Introducimos los grupos,

In[]:= **G1= {1,2,3,4};**
operacion1=={{1,2,3,4},{2,4,1,3},{3,1,4,2},{4,3,2,1}};
op1[x_,y_]:=Module[{i,j},
i=1;While[Intersection[{x},{G1[[i]]}]=={},i++];
j=1;While[Intersection[{y},{G1[[j]]}]=={},j++];
operacion1[[i,j]]
];

In[]:= **G2={0,1};**

```

operacion2={\{0,1\},\{1,0\}};
op2[x_,y_]:=Module[\{i,j\},
  i=1;While[Intersection[\{x\},\{G2[[i]]\}]=={},i++];
  j=1;While[Intersection[\{y\},\{G2[[j]]\}]=={},j++];
  operacion2[[i,j]]
];

```

Definimos la aplicación h ,

```

In]:=          F[1]:=1;
              F[2]:=0;
              F[3]:=1;
              F[4]:=0;

```

o bien,

```

In]:=          F[x_]:=Mod[x,2];

```

y ahora usamos 2.10.,

```

In]:=          Homomorfismo=True;

```

```

          :
          :

```

```

Out]=          False

```

en efecto, observemos que:

```

In]:=          F[op1[3,4]]==op2[F[3],F[4]]

```

```

Out]=          False

```

f) Introducimos los grupos,

```

In]:=          G1=\{0,1,2,3,4,5,6,7\};
              operacion1=\{\{0,1,2,3,4,5,6,7\},\{1,2,3,4,5,6,7,0\},\{2,3,4,5,6,7,0,1\},
              \{3,4,5,6,7,0,1,2\},\{4,5,6,7,0,1,2,3\},\{5,6,7,0,1,2,3,4\},
              \{6,7,0,1,2,3,4,5\},\{7,0,1,2,3,4,5,6\}\};
              G2=\{0,1,2,3\};
              operacion2=\{\{0,1,2,3\},\{1,2,3,0\},\{2,3,0,1\},\{3,0,1,2\}\};

```

Calculamos el grafo de la aplicación:

```

In]:=          grafoF=\{};
              Do[
                  AppendTo[grafoF,\{G1[[CONTADORi]],
                  F[G1[[CONTADORi]]]\}],
                  ,\{CONTADORi,1,Length[G1]\}];

```

Definimos la función 2.11. y la utilizamos:

```
In[]:= HOMOMORFISMO[G1_,operacion1_,G2_,operacion2_,
grafoF_]:=Module[
;
;
]

In[]:= HOMOMORFISMO[G1,operacion1,G2,operacion2,grafoF]

Out[]= True
```

□

3. OTROS EJEMPLOS Y GRUPOS INFINITOS

En este capítulo no se ha hecho un estudio exhaustivo de todas las técnicas computacionales que analizan el concepto de grupo, sólo se ha dado una visión general y directa que permita al lector trasladar la definición de grupo al ordenador sin problemas, limitándose, en general, a casos finitos. En particular no se han estudiado los grupos finitamente presentados, se aconseja al lector interesado dirigirse a alguna de las referencias bibliográficas ya mencionadas. En este epígrafe, estudiamos algunos ejemplos que por sus características particulares pueden ser resueltos fácilmente con el programa Mathematica.

En algunas ocasiones los grupos finitos, si los elementos son numéricos y lo permiten, se pueden manejar de forma más cómoda respecto a cómo hemos visto en las secciones anteriores, como podemos ver en los siguientes ejemplos.

Ejemplo 2.9. \mathbb{Z}_n . Si queremos usar el grupo comutativo $(\mathbb{Z}_n, +)$ bastará con tener en cuenta que, $a + b = a + b$ y así podemos manejar fácilmente la operación y el conjunto:

```
In[]:= n=ENTERO MAYOR QUE UNO;
Zn=Table[i,{i,0,n-1}];
operacionn=Table[Mod[x+y,n],{x,0,n-1},{y,0,n-1}];
op[x_,y_]:=Mod[x+y,n];
```

Por ejemplo, para $n = 11$, comprobemos que en efecto es un grupo abeliano, para ello usaremos las funciones 2.3., 2.6., 2.8. y 2.9.:

```
In[]:= INTERNA[G_,operacion_]:=Module[
;
;

In[]:= ELEMENTOSIMETRICO[G_,operacion_]:=

;
;

In[]:= ASOCIATIVA[G_,operacion_]:=Module[
```

```

          :
          :

In]:=      CONMUTATIVA[G_,operacion_]:=Module[
          :
          :

In]:=      GRUPOABELIANO[Conjunto_,Operacion_]:=
          INTERNA[Conjunto,Operacion] &&
          ELEMENTOSIMETRICO[Conjunto,Operacion] &&
          ASOCIATIVA[Conjunto,Operacion] &&
          CONMUTATIVA[Conjunto,Operacion];

In]:=      n=11;
          Zn=Table[i,{i,0,n-1}];
          operacionn=Table[Mod[x+y,n],{x,0,n-1},{y,0,n-1}];
          op[x_,y_]:=Mod[x+y,n];

In]:=      GRUPOABELIANO[Zn,operación]

Out]=      Elemento Neutro: 0
          
```

| | | | | | | | | | | | |
|------------|---|----|---|---|---|---|---|---|---|----|----|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| Elementos: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Simétricos | 0 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

 True

Si queremos introducir la aplicación $f: \mathbb{Z}_n \rightarrow \mathbb{Z}_m$, definida por $f(\bar{x}) := \bar{x}$, escribiríamos:

```

In]:=      n=ENTERO MAYOR QUE UNO;
          m=ENTERO MAYOR QUE UNO;
          Zn=Table[i,{i,0,n-1}];
          Zm=Table[i,{i,0,m-1}];
          operacionn=Table[Mod[x+y,n],{x,0,n-1},{y,0,n-1}];
          operacionm=Table[Mod[x+y,m],{x,0,m-1},{y,0,m-1}];
          opn[x_,y_]:=Mod[x+y,n];
          opm[x_,y_]:=Mod[x+y,m];
          F[x_]:=Mod[x,m];
          grafoF=Table[{i,Mod[i,m]},{i,0,n-1}];

```

Y ahora con la función 2.11., sería fácil comprobar si es homomorfismo de grupos, para $n = 16$ y $m = 8$:

```

In]:=      n=16;
          m=8;
          :
          :

```

In]:= **HOMOMORFISMO[Zn,operacionn,Zm,operacionm,grafoF]**

Out]= True

Y para $n = 11$ y $m = 5$:

In]:= **n=7;**
m=5;

⋮ ⋮

In]:= **HOMOMORFISMO[Zn,operacionn,Zm,operacionm,grafoF]**

Out]= False

□

Ejemplo 2.10. Grupos Cíclicos. Un grupo finito $(G, *)$ de n elementos, se dice que es cíclico si existe un elemento $a \in G$ tal que

$$G = \{a * a * \dots * a = a^i \mid i \in \mathbb{Z}\},$$

esto es, si está generado por un único elemento a , $G = \langle a \rangle$. Obsérvese que $(\mathbb{Z}_n, +)$ es un grupo cíclico de n elementos generado por 1.

Podemos introducir un grupo cíclico de n elementos de la siguiente forma:

In]:= **G=Table[i,{i,n}];**
operacion=Table[Mod[i+j-2,n]+1,{i,n},{j,n}];

Por ejemplo, si consideramos el grupo cíclico generado por a de 7 elementos, $G = \{1, a, a^2, a^3, a^4, a^5, a^6\}$, podemos introducirlo de la siguiente forma:

In]:= **G=Table[i,{i,7}];**
operacion=Table[Mod[i+j-2,7]+1,{i,7},{j,7}];

Y para representar cada elemento con la notación que deseemos:

In]:= **G1={1,a,a²,a³,a⁴,a⁵,a⁶};**
operacion1=Table[G1[[operacion[[i,j]]]],{i,7},{j,7}];

In]:= **TableForm[operacion1]**

Out]=

$$\begin{array}{ccccccc}
 1 & a & a^2 & a^3 & a^4 & a^5 & a^6 \\
 a & a^2 & a^3 & a^4 & a^5 & a^6 & 1 \\
 a^2 & a^3 & a^4 & a^5 & a^6 & 1 & a \\
 a^3 & a^4 & a^5 & a^6 & 1 & a & a^2 \\
 a^4 & a^5 & a^6 & 1 & a & a^2 & a^3 \\
 a^5 & a^6 & 1 & a & a^2 & a^3 & a^4 \\
 a^6 & 1 & a & a^2 & a^3 & a^4 & a^5
 \end{array}$$

□

Por otra parte, existen otros conjuntos, como el de las matrices $n \times m$, los números reales, enteros, ...; todos ellos con la operación suma son grupos y dicha operación ya viene implementada en el programa Mathematica para estos conjuntos, por tanto, podremos usarla a nuestro antojo. Algo similar ocurre para otros conjuntos y operaciones, como en el siguiente ejemplo, con matrices cuadradas 2×2 con su producto.

Ejemplo 2.11. Comprobar que el conjunto $G = \left\{ \begin{pmatrix} a & b \\ 0 & c \end{pmatrix} \in M_{2 \times 2}(\mathbb{R}) \mid a, c \in \mathbb{R} - \{0\}, b \in \mathbb{R} \right\}$ con la operación producto de matrices es un grupo.

a) Comprobamos que es interna:

```
In[]:= A1={{a1,b1},{0,c1}};
A2={{a2,b2},{0,c2}};
MatrixForm[A1.A2]
```

$$Out[]= \begin{pmatrix} a_1 a_2 & a_1 b_2 + b_1 c_2 \\ 0 & c_1 c_2 \end{pmatrix}$$

en efecto es Ley de composición interna, pues $a_1 a_2, c_1 c_2 \neq 0$.

b) Calculamos el neutro:

```
In[]:= Solve[A1.A2==A1,{a2,b2,c2}]
Out[]= {{b2→0,a2→1,c2→1}}
```

Luego el elemento neutro es la matriz identidad.

c) Calculamos el elemento simétrico (inverso) de una matriz de G cualquiera no nula:

```
In[]:= Solve[A1.A2=={{1,0},{0,1}},{a2,b2,c2}]
```

$$Out[] = \left\{ b2 \rightarrow -\frac{b1}{a1c1}, a2 \rightarrow \frac{1}{a1}, c2 \rightarrow \frac{1}{c1} \right\}$$

Por tanto, la matriz $\begin{pmatrix} \frac{1}{a1} & -\frac{b1}{a1c1} \\ 0 & \frac{1}{c1} \end{pmatrix}$ es la inversa de $\begin{pmatrix} a1 & b1 \\ 0 & c1 \end{pmatrix}$, que existirá

siempre que $a1, c1 \neq 0$.

d) Comprobamos si es asociativa:

$$\begin{aligned} In[] := & \quad A1 = \{\{a1, b1\}, \{0, c1\}\}; \\ & A2 = \{\{a2, b2\}, \{0, c2\}\}; \\ & A3 = \{\{a3, b3\}, \{0, c3\}\}; \\ & \text{Expand}[(A1.A2).A3] == \text{Expand}[A1.(A2.A3)] \end{aligned}$$

$$Out[] = \text{True}$$

por tanto es asociativa.

□

Si el conjunto G , aunque no sea finito, es un conjunto de valores numéricos y la operación se basa en operaciones ya implementadas en Mathematica (suma, resta, producto,...), a pesar de la infinitud, algunos pueden resolverse con Mathematica, aunque la solución precise de ideas originales y particulares según el caso, sirvan como muestra los siguientes ejemplos:

Ejemplo 2.12. Comprobar que el conjunto $G = \mathbb{R} - \{-1\}$ con la operación:

$$x * y = xy + x + y$$

es un grupo conmutativo.

a) En primer lugar definimos la operación:

$$In[] := \text{op}[x_, y_] := x y + x + y;$$

b) Comprobamos que es interna:

$$In[] := \text{Solve}[\text{op}[x, y] == -1, \{x, y\}]$$

Solve::: svars : Equations may not give solutions for all "solve" variables. >>

$$Out[] = \{\{x \rightarrow -1\}, \{y \rightarrow -1\}\}$$

en efecto es Ley de composición interna, pues salvo para $x = -1$ o $y = -1$,

$$x * y \in \mathbb{R} - \{-1\}.$$

c) Calculamos el neutro:

In[]:= Solve[op[x,y]==x,y]

Out[]= { {y→0} }

por tanto el elemento neutro es el 0.

d) Calculamos el elemento simétrico de un x cualquiera:

In[]:= Solve[op[x,y]==0,y]

Out[]= { {y→-x/(1+x)} }

luego $y = -\frac{x}{1+x}$ es el simétrico de x que existirá para cualquier $x \neq -1$.

e) Comprobamos si es asociativa:

In[]:= TrueQ[Expand[op[x,op[y,z]]]==Expand [op[op[x,y],z]]]]

Out[]= True

por tanto es asociativa.

f) Por último, comprobamos si es commutativa:

In[]:= TrueQ[Expand[op[x,y]]==Expand[op[y,x]]]]

Out[]= True

Y así, concluimos que tenemos un grupo commutativo. □

Ejemplo 2.13. Comprobar si el conjunto $\mathbb{R} \times \mathbb{R}^*$ con la operación,

$$(a, b) * (c, d) = (ad + c, bd)$$

es un grupo commutativo.

En primer lugar introducimos la operación:

*In[]:= op[{a_,b_},{c_,d_}]:= {a*d+c,b*d};*

La ley de composición es interna, porque $bd \neq 0$ ya que b y d son elementos de \mathbb{R}^* .

Ahora comprobamos las propiedades de la operación, comprobamos si existe el elemento neutro y lo calculamos:

In]:= **Solve[$\text{op}[\{a,b\},\{c,d\}] == \{a,b\},\{c,d\}]$**

Out]:= $\{\{c \rightarrow 0, d \rightarrow 1\}\}$

La única solución de $\mathbb{R} \times \mathbb{R}^*$ es $(0, 1)$, que por tanto será el neutro. Ahora calculamos el simétrico:

In]:= **Solve[$\text{op}[\{a,b\},\{c,d\}] == \{0,1\},\{c,d\}]$**

Out]:= $\{\{c \rightarrow -\frac{a}{b}, d \rightarrow \frac{1}{b}\}\}$

Como b es distinto de cero, el simétrico de (a, b) siempre existe y es $(-(a/b), 1/b)$. Veamos la propiedad asociativa:

In]:= **Expand[$\text{op}[\text{op}[\{a,b\},\{c,d\}],\{e,f\}] ==$
 $\text{Expand}[\text{op}[\{a,b\},\text{op}[\{c,d\},\{e,f\}]]]$**

Out]:= **True**

Ya sabríamos que es un grupo, por último comprobamos la propiedad conmutativa:

In]:= **Expand[$\text{op}[\{a,b\},\{c,d\}] == \text{Expand}[\text{op}[\{c,d\},\{a,b\}]]$**

Out]:= $\{c+ad, bd\} == \{a+bc, bd\}$

Observamos que no responde “True”, y es porque no se puede asegurar que las primeras componentes de los pares ordenados, $ad + c$ y $a + bc$, sean iguales, por tanto concluimos que no verifica la propiedad conmutativa y aunque se trata de un grupo, no es abeliano. Podemos utilizar para comparar las expresiones y ver si son idénticas el test “==”:

In]:= **Expand[$\text{op}[\{a,b\},\{c,d\}] == \text{Expand}[\text{op}[\{c,d\},\{a,b\}]]$**

Out]:= **False**

□

Ejemplo 2.14. Consideramos el conjunto $G = \mathbb{R} \times \mathbb{R} - \{(0,0)\}$ con la operación:

$$(a, b) \cdot (c, d) = (ac - bd, ad + bc)$$

Comprobar que (G, \cdot) es un grupo. ¿Es conmutativo?

Definimos la operación y comprobamos que es interna.

*In[]:= op[{a_,b_},{c_,d_}]:=a*c - b*d,a*d + b*c};*

In[]:= Solve[op[{a,b},{c,d}]=={0,0},{a,b,c,d}]

Solve::: svars : Equations may not give solutions for all "solve" variables. >>

Out[]={ {c->0,d->0}, {a->-i b,c->i d}, {a->i b,c->-i d}, {a->0,b->0} }

A la vista de la salida es obvio que la operación es interna. Calculamos el neutro y el elemento simétrico de uno dado:

In[]:= Solve[op[{a,b},{c,d}]=={a,b},{c,d}]

Out[]={ {c->1,d->0} }

In[]:= Solve[op[{a,b},{c,d}]=={1,0},{c,d}]

Out[]={ {c->a/(a^2+b^2),d->-b/(a^2+b^2)} }

El elemento neutro es $(1, 0)$ y el simétrico de (a, b) es $\left(\frac{a}{a^2+b^2}, -\frac{b}{a^2+b^2}\right)$.

Comprobamos ahora la propiedad asociativa, con lo cual ya sería grupo y la propiedad conmutativa para concluir que es un grupo abeliano.

In[]:= Expand[op[op[{a,b},{c,d}],{e,f}]]==Expand[op[{a,b},op[{c,d},{e,f}]]]

Out[]=True

In[]:= Expand[op[{a,b},{c,d}]]==Expand[op[{c,d},{a,b}]]

Out[]=True

□

3.1. GRUPOS DE ORDEN PEQUEÑO

En este epígrafe vamos a realizar un estudio combinatorio sobre los grupos de orden pequeño, esto es, vamos a intentar calcular con el ordenador, salvo isomorfismo,

todos los grupos distintos que existen de un orden pequeño fijo. Como comprobaremos, se trata de un problema que exige una gran cantidad de cálculo al ordenador, por lo que usaremos algunas propiedades matemáticas para acelerar y optimizar estos cálculos. Recordemos que dos grupos son isomorfos si existe un homomorfismo biyectivo de grupos entre ellos.

Si tenemos dos grupos finitos de orden n : $G_1 = \{x_1, x_2, \dots, x_n\}$ y $G_2 = \{y_1, y_2, \dots, y_n\}$, entre los que existe un isomorfismo: $f: G_1 \rightarrow G_2$, y consideramos los elementos de G_2 ordenados de la siguiente forma: $G_2 = \{f(x_1), f(x_2), \dots, f(x_n)\}$, entonces las tablas de operaciones de G_1 y G_2 , salvo el nombre de los vértices son indistinguibles; de hecho, si identificamos los elementos por sus subíndices son idénticas. Pretendemos calcular todos los grupos de orden n que existen salvo isomorfismo y según acabamos de razonar, no habría ningún problema en suponer que los elementos del conjunto G (un candidato a grupo) sean: $\{1, 2, \dots, n\}$ y tampoco lo habría en que “1” sea el nombre del elemento neutro. Por tanto, los distintos grupos que nos puedan salir de orden n tendrán el mismo conjunto de elementos pero diferirán en la tabla de operaciones, que tendrán el siguiente aspecto:

| * | 1 | 2 | ... | n |
|-----|-----|---|-----|-----|
| 1 | 1 | 2 | ... | n |
| 2 | 2 | ? | ... | ? |
| : | : | : | ⋮ | ⋮ |
| n | n | ? | ... | ? |

Puesto que el elemento neutro hemos considerado de partida que es el “1”, la primera fila y columna podemos determinarla directamente. Para construir tablas de operaciones que puedan dar lugar a grupos, hemos de tener en cuenta que en cada fila y columna (al modo de un sudoku) no puede repetirse ningún elemento, puesto que si eso ocurriera (por ejemplo, si en la columna j se repitieran dos elementos en las filas i_1 e i_2 , entonces $i_1 * j = i_2 * j$, y en consecuencia $i_1 = i_2$), el grupo no tendría n elementos distintos. Por tanto, si llenamos los casilleros de la tabla que no conocemos (en los que hemos puesto una interrogante) bajo esta sencilla directriz (que no se repitan elementos en las filas o en las columnas) conseguiremos tablas de operaciones candidatas a dotar a G de estructura de grupo. Por otra parte, podemos asegurar que las tablas de operaciones obtenidas así obviamente pertenecerán a operaciones internas, verificarán la propiedad del elemento neutro porque así lo hemos forzado con la fila y columna primera que elegimos de partida, y además, al no repetirse los elementos de las filas y columnas, podemos asegurar que en cada fila y columna aparece el “1” (elemento neutro), por tanto si construimos las tablas de forma que, si ponemos un “1” en la coordenada (i, j) también lo pongamos en la (j, i) , entonces también podemos asegurar que verificará la propiedad del elemento simétrico. En consecuencia, por este método obtenemos tablas de operaciones que dotarían a G de estructura de grupo, a falta de la propiedad asociativa.

El siguiente, es un programa que para un cierto n calcula todas las posibles tablas de operaciones tal y como hemos descrito anteriormente, esto es, a falta de comprobar la propiedad asociativa:

| PROGRAMA | COMENTARIOS |
|--|--|
| tabla=TABLA DE OPERACIONES; | En la variable “tabla” almacenaremos la tabla de operaciones de partida, en ésta supondremos que “1” es el neutro, por tanto la primera y última fila son conocidas, si tenemos alguna información (relación) más, también la incluimos y en aquellas posiciones de la tabla que desconozcamos pondremos un “0”. |
| n=Length[table]; tiempo=TimeUsed[]; listadefinitiva={}; listatablas={tabla}; | “n” será el orden del grupo, “tiempo” nos servirá para medir el tiempo empleado en los cálculos, en “listadefinitiva” almacenaremos las tablas de operaciones cuando estén terminadas, y en “listatablas” se almacenarán las tablas conforme se vayan construyendo. |
| G=Table[i,{i,n}]; | “G” será el conjunto del grupo. |
| While[listatablas≠{}; listatablastemp={}; | Primer bucle, la condición de parada consistirá en comprobar que en “listatablas” no quede ninguna tabla sin terminar. |
| Do[| Segundo bucle que recorrerá todas las tablas a medio construir que hay en “listatablas”. |
| tabla=listatablas[[j]]; cero=Position[tabla,0]; fin=Length[cero]; cero=cero[[1]]; rellenos=Complement[G,Union[tabla[[cero[[1]]]], Transpose[tabla][[cero[[2]]]]]]; | Tomamos la tabla j -ésima de “listatablas” y buscamos la primera posición que contiene un “0” y comprobamos los posibles valores que puede tomar. |
| Do[| Tercer bucle que recorre los posibles valores que calculamos anteriormente. |
| nuevatabla=tabla; If[rellenos[[i]]==1, nuevatabla[[cero[[1]],cero[[2]]]]=1; nuevatabla[[cero[[2]],cero[[1]]]]=1; , nuevatabla[[cero[[1]],cero[[2]]]]=rellenos[[i]];]; | Creamos una tabla para cada valor y en caso de poner un “1”, tenemos la precaución de ponerlo también en la posición simétrica para que se verifique la propiedad del elemento simétrico. |
| If[fin==1, AppendTo[listadefinitiva,nuevatabla]; , AppendTo[listatablastemp,nuevatabla];]; | Si “fin” es 1, significará que no hay más ceros en la tabla y en consecuencia ya está completa. Si está terminada la introducimos en “listadefinitiva” y en |

| | | |
|---|--|--|
| | caso contrario en “listatablastemp”. | |
| ,{i,1,Length[rellenos]}]; ,{j,1,Length[listatablas]}]; listatablas=listatablastemp;]; | Actualizamos “listatablas”. | |
| Print["Tiempo empleado: ",TimeUsed[]-tiempo]; Length[listadefinitiva] | Mostramos el tiempo empleado en los cálculos y el número de tablas encontradas. En “listadefinitiva” están almacenadas estas tablas. | |

Programa 2.12. Grupos de orden pequeño.

El programa 2.12. calcula tablas de operaciones de posibles candidatos a grupos a falta de comprobar la asociatividad, con la función 2.8. hacemos esta última comprobación:

```
In]:= G=Table[i,{i,n}];  
grupos={};  
Do[  
  If[ASOCIATIVA[G,listadefinitiva[[i]]],  
    AppendTo[grupos,listadefinitiva[[i]]]  
,{i,1,Length[listadefinitiva]}];
```

Todas las tablas de operaciones obtenidas así, dotan a G de estructura de grupo, sin embargo, algunas de ellas pertenecerán a grupos isomorfos, por lo que ahora debemos comprobar cuántas hay distintas salvo isomorfismo. Utilizando la función Permutations[] (para más detalle véase el primer epígrafe del capítulo 4) construiremos las posibles biyecciones y usaremos la función HOMOMORFISMO[] para determinar si éstas son isomorfismos, lo implementamos en la siguiente función:

| FUNCIÓN | COMENTARIOS |
|---|--|
| HOMOMORFISMO[G1_,operacion1_, G2_,operacion2_,grafoF_]:=... | Introducimos la función 2.11. |
| QUITARISOMORFOS[tablasgrupos_]:=Module[{n,G,ISO,borrar,grupos,i,j,tiempo}, tiempo=TimeUsed[]; n=Length[tablasgrupos[[1]]]; permutar=Permutations[Table[i,{i,2,n}]]; G=Table[i,{i,n}]; ISO[G1_,operacion1_,G2_,operacion2_]:=Module[{i,j,grafo}, isomorfos=False; Do[grafo=Union[{{1,1}}, Table[{i,permutar[[j]][[i-1]]},{i,2,n}]]; If[HOMOMORFISMO[G1,operacion1, G2,operacion2,grafo],isomorfos=True; Break[]]; ,{j,Length[permutar]}]; isomorfos | Sólo tendrá una entrada: un listado de tablas de operaciones que dotan de estructura de grupo a $G = \{1, 2, \dots, n\}$. “n” será el orden del grupo y “G” el conjunto del grupo. En “permutar” almacenamos todas las posibles biyecciones que dejan al 1 (elemento neutro) fijo. |
| | Definimos la función ISO[] que comprobará si dos grupos G_1 y G_2 son isomorfos, para ello comprobará, una por una, todas las posibles biyecciones entre los elementos de los grupos (el neutro, esto es el 1, lo dejamos fijo). Hará uso de la función 2.11. para comprobar que sea un isomorfismo. |

| | |
|--|---|
| <code>];</code> | |
| grupos=tablasgrupos; | |
| i=1; While[i<Length[grupos], | Recorremos las tablas de operaciones. |
| borrar={}; | |
| Do[| Comprobamos qué grupos son isomorfos al grupo i -ésimo. |
| If[ISO[G.grupos[[i]],G.grupos[[j]]], AppendTo[borrar,{j}]]; | Testeamos si son isomorfos los grupos i -ésimo y j -ésimo. |
| ,{j,i+1,Length[grupos]}]; | |
| grupos=Delete[grupos,borrar]; i++; | Borramos de “grupos” aquellos que son isomorfos al i -ésimo grupo. |
|]; | |
| Print["Tiempo empleado: ",TimeUsed[]-tiempo]; grupos | Mostramos el tiempo empleado en los cálculos y las tablas de los grupos distintos que finalmente han quedado. |
|]; | |

Función 2.13. Grupos isomorfos.

Ahora utilizaremos el programa 2.12. y la función 2.13. en algunos ejemplos:

Ejemplo 2.15. En este ejemplo vamos a calcular todos los grupos que existen, salvo isomorfismo de 4 y 6 elementos.

Usamos la función 2.12. para la siguiente tabla de operaciones:

In[]:= **tabla=Table[If[i==1,j,If[j==1,i,0]],{i,4},{j,4}];**
TableForm[tabla]

Out[] =

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 2 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 |

In[]:= **tabla=Table[If[i==1,j,If[j==1,i,0]],{i,4},{j,4}];**

⋮ ⋮

Out[] = Tiempo empleado: 0.047

4

Ahora comprobamos cuáles de ellas verifican la propiedad asociativa, para ello definimos la función 2.8.:

In[]:= **ASOCIATIVA[G_,operacion_]:=Module[**

```

          :
          :

In]:=      G=Table[i,{i,n}];
grupos={};
Do[
  If[ASOCIATIVA[G,listadefinitiva[[i]]],
    AppendTo[grupos,listadefinitiva[[i]]]
  ,{i,1,Length[listadefinitiva]}];
]

Out]=      4

```

Y ahora eliminamos aquellas que doten a G de estructuras de grupos isomorfas, para ello utilizaremos la función 2.14. (previamente definimos la función 2.11.):

```

In]:=      HOMOMORFISMO[G1_,operacion1_,G2_,operacion2_,
grafoF_]:=Module[
          :
          :

In]:=      QUITARISOMORFOS[tablasgrupos_]:=Module[
          :
          :

In]:=      grupos=QUITARISOMORFOS[grupos];

```

```
Out]=      Tiempo empleado: 0.079
```

Y concluimos mostrando las tablas de operaciones de los dos únicos grupos, salvo isomorfismo, de 4 elementos que existen:

```

In]:=      TableForm[Table[
  {StyleForm[StringJoin["Grupo ",ToString[j],""]],"
  FontSize→22},TableForm[grupos[[j]],
  TableHeadings→{Table[StyleForm[i,
  FontWeight→"Bold"],{i,n}],
  Table[StyleForm[i,FontWeight→"Bold"],{i,n}]},
  TableSpacing→{2,2}],{j,Length[grupos]}],
  TableSpacing→{6,2}]

```

```
Out]=
          1   2   3   4
Grupo 1 : 2 | 2   1   4   3
          3 | 3   4   2   1
          4 | 4   3   1   2
```

```
Grupo 2 : 1 | 1   2   3   4
          2 | 2   1   4   3
```

| | | | | | |
|---|---|---|---|---|--|
| 3 | 3 | 4 | 1 | 2 | |
| 4 | 4 | 3 | 2 | 1 | |

Lo mismo hacemos para calcular los grupos de orden 6:

In[]:= tabla=Table[If[i==1,j,If[j==1,i,0]],{i,6},{j,6}];

⋮ ⋮

Out[] = Tiempo empleado: 2.359

1808

In[]:= G=Table[i,{i,n}];
grupos={};
Do[
If[ASOCIATIVA[G,listadefinitiva[[i]]],
AppendTo[grupos,listadefinitiva[[i]]]]
,{i,1,Length[listadefinitiva]}];

Out[] = 80

In[]:= grupos=QUITARISOMORFOS[grupos];

Out[] = Tiempo empleado: 7.469

In[]:= TableForm[Table[
{StyleForm[StringJoin["Grupo ",ToString[j],""]],
FontSize→22},TableForm[grupos[[j]],
TableHeadings→{Table[StyleForm[i,
FontWeight→"Bold"],{i,n}],
Table[StyleForm[i,FontWeight→"Bold"],{i,n}]},
TableSpacing→{2,2}}},{j,Length[grupos]}],
TableSpacing→{6,2}]

Out[] =

| | 1 | 2 | 3 | 4 | 5 | 6 | |
|----------|---|---|---|---|---|---|--|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | |
| 2 | 2 | 1 | 4 | 3 | 6 | 5 | |
| 3 | 3 | 4 | 5 | 6 | 1 | 2 | |
| 4 | 4 | 3 | 6 | 5 | 2 | 1 | |
| 5 | 5 | 6 | 1 | 2 | 3 | 4 | |
| 6 | 6 | 5 | 2 | 1 | 4 | 3 | |

| | 1 | 2 | 3 | 4 | 5 | 6 | |
|----------|---|---|---|---|---|---|--|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | |
| 2 | 2 | 1 | 4 | 3 | 6 | 5 | |
| 3 | 3 | 5 | 1 | 6 | 2 | 4 | |
| 4 | 4 | 6 | 2 | 5 | 1 | 3 | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 5 | 5 | 3 | 6 | 1 | 4 | 2 |
| 6 | 6 | 4 | 5 | 2 | 3 | 1 |

□

Ejemplo 2.16. Sea G un grupo de 8 elementos del que sabemos que su tabla de operaciones es de la forma:

| * | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 2 | 2 | 3 | 4 | 1 | ? | ? | ? | ? |
| 3 | 3 | 4 | 1 | 2 | ? | ? | ? | ? |
| 4 | 4 | 1 | 2 | 3 | ? | ? | ? | ? |
| 5 | 5 | ? | ? | ? | 1 | 4 | 3 | 2 |
| 6 | 6 | ? | ? | ? | 2 | 1 | 4 | 3 |
| 7 | 7 | ? | ? | ? | 3 | 2 | 1 | 4 |
| 8 | 8 | ? | ? | ? | 4 | 3 | 2 | 1 |

Utilizamos 2.12. para completar la tabla de operaciones que nos dan:

```
In]:= n=8;
tabla=Table[If[i==1,j,If[j==1,i,0]],{i,n},{j,n}];
tabla[[2,2]]=3;tabla[[2,3]]=4;tabla[[3,2]]=4;tabla[[3,3]]=1;
tabla[[2,4]]=1;tabla[[4,2]]=1;tabla[[4,3]]=2;tabla[[3,4]]=2;
tabla[[4,4]]=3;tabla[[5,5]]=1;tabla[[5,6]]=4;tabla[[6,5]]=2;
tabla[[6,6]]=1;tabla[[5,7]]=3;tabla[[7,5]]=3;tabla[[6,7]]=4;
tabla[[7,6]]=2;tabla[[7,7]]=1;tabla[[8,5]]=4;tabla[[5,8]]=2;
tabla[[6,8]]=3;tabla[[8,6]]=3;tabla[[7,8]]=4;tabla[[8,7]]=2;
TableForm[tabla]
```

```
Out]=
1 2 3 4 5 6 7 8
2 3 4 1 0 0 0 0
3 4 1 2 0 0 0 0
4 1 2 3 0 0 0 0
5 0 0 0 1 4 3 2
6 0 0 0 2 1 4 3
7 0 0 0 3 2 1 4
8 0 0 0 4 3 2 1
```

```
In]:= tiempo=TimeUsed[];
```

⋮ ⋮

```
Out]= Tiempo empleado: 0.235
```

Ahora comprobamos cuáles de ellas verifican la propiedad asociativa, para ello definimos la función 2.8.:

In[]:= ASOCIATIVA[G_,operacion_]:=Module[

⋮ ⋮

*In[]:= G=Table[i,{i,n}];
grupos={};
Do[
If[ASOCIATIVA[G,listadefinitiva[[i]]],
AppendTo[grupos,listadefinitiva[[i]]]]
,{i,1,Length[listadefinitiva]}];*

Out[] = 1

Por tanto, sólo encontramos un único grupo:

*In[]:= TableForm[grupos[[1]],
TableHeadings→{Table[StyleForm[i,
FontWeight→"Bold"],{i,n}],
Table[StyleForm[i,FontWeight→"Bold"],{i,n}]},
TableSpacing→{2,2}]*

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 2 | 2 | 3 | 4 | 1 | 6 | 7 | 8 | 5 |
| 3 | 3 | 4 | 1 | 2 | 7 | 8 | 5 | 6 |
| 4 | 4 | 1 | 2 | 3 | 8 | 5 | 6 | 7 |
| 5 | 5 | 8 | 7 | 6 | 1 | 4 | 3 | 2 |
| 6 | 6 | 5 | 8 | 7 | 2 | 1 | 4 | 3 |
| 7 | 7 | 6 | 5 | 8 | 3 | 2 | 1 | 4 |
| 8 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

□

Ejemplo 2.17. Buscar cualquier grupo G de 8 elementos: $a, b, c, d, e, f, g, h \in G$ distintos entre sí, sabiendo que h es el elemento neutro y que verifican: $a * b = b * a = c$, $b^2 = d^2 = e^2 = f^2 = g^2 = h$, $a^2 = b$ y $f * a = e$.

En primer lugar, observemos que con las relaciones anteriores pueden deducirse otras:

- $b * a = c$, entonces $b * c = b * (b * a) = h * a$ y por tanto $\underline{b * c = a}$. Análogamente, $a * b = c$, entonces $c * b = (a * b) * b = a * h$ y por tanto $\underline{c * b = a}$.
- $a * c = a * (a * b) = a^2 * b = b^2 = h$, por tanto $\underline{a * c = h}$. Análogamente, $c * a = (b * a) * a = b * a^2 = b^2 = h$, por tanto $\underline{c * a = h}$.
- $c^2 = (a * b)^2 = (a * b) * (a * b) = (a * b) * (b * a) = a * b^2 * a = a^2 = b$, por tanto $\underline{c^2 = b}$.

- $f * a = e$, entonces $f * e = f * (f * a) = h * a$ y por tanto $f * e = a$. Y además, como $f * e = a$, entonces $a * e = (f * e) * e = f * h$ y $f * a = f * (f * e) = h * e$ por tanto $a * e = f$ y $f * a = e$.
- $f * a = e$, entonces $(f * a) * a = e * a$, luego $f * a^2 = e * a$ y así $f * b = e * a$, por tanto $e * (f * b) = e * (e * a)$ y $(e * f) * b = a$, de donde $(e * f) * b^2 = a * b = c$, y $c = e * f$.
- $e * f = c$, luego $e * f^2 = c * f$, por tanto $e = c * f$ y análogamente $e^2 * f = e * c$, por tanto $e * c = f$.

Podemos suponer que $G = \{1, 2, 3, 4, 5, 6, 7, 8\}$, con $1 = h$, el elemento neutro; $a = 2$, $b = 3$, $c = 4$, $d = 5$, $e = 6$, $f = 7$ y $g = 8$. Teniendo en cuenta las relaciones de partida junto con las que hemos deducido, la tabla inicial de partida quedaría como sigue:

| * | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 2 | 2 | 3 | 4 | 1 | ? | 7 | ? | ? |
| 3 | 3 | 4 | 1 | 2 | ? | ? | ? | ? |
| 4 | 4 | 1 | 2 | 3 | ? | ? | 6 | ? |
| 5 | 5 | ? | ? | ? | 1 | ? | ? | ? |
| 6 | 6 | ? | ? | 7 | ? | 1 | 4 | ? |
| 7 | 7 | 6 | ? | ? | ? | 2 | 1 | ? |
| 8 | 8 | ? | ? | ? | ? | ? | ? | 1 |

La introducimos en el ordenador:

```
In[7]:= n=8;
tabla=Table[If[i==1,j,If[j==1,i,0]],{i,n},{j,n}];
tabla[[2,2]]=3;tabla[[2,3]]=4;tabla[[2,4]]=1;tabla[[2,6]]=7;
tabla[[3,2]]=4;tabla[[3,3]]=1;tabla[[3,4]]=2;
tabla[[4,2]]=1;tabla[[4,3]]=2;tabla[[4,4]]=3;tabla[[4,7]]=6;
tabla[[5,5]]=1;
tabla[[6,4]]=7;tabla[[6,6]]=1;tabla[[6,7]]=4;
tabla[[7,2]]=6;tabla[[7,6]]=2;tabla[[7,7]]=1;
tabla[[8,8]]=1;
TableForm[tabla]
```

```
Out[7]= 1 2 3 4 5 6 7 8
         2 3 4 1 0 7 0 0
         3 4 1 2 0 0 0 0
         4 1 2 3 0 0 6 0
         5 0 0 0 1 0 0 0
         6 0 0 7 0 1 4 0
         7 6 0 0 0 2 1 0
         8 0 0 0 0 0 0 1
```

Con 2.12. completamos la tabla:

```
In[8]:= tiempo=TimeUsed[];
```

⋮ ⋮

Out[7]:= Tiempo empleado: 0.063

8

Ahora comprobamos cuáles de ellas verifican la propiedad asociativa, para ello previamente definimos la función 2.8.:

In[8]:= ASOCIATIVA[G_,operacion_]:=Module[

⋮ ⋮

In[9]:= G=Table[i,{i,n}];

grupos={};

Do[

If[ASOCIATIVA[G,listadefinitiva[[i]]],

AppendTo[grupos,listadefinitiva[[i]]]]

,{i,1,Length[listadefinitiva]}];

Out[9]:= 2

Comprobamos que son isomorfos, para ello utilizaremos la función 2.14. (previamente definimos la función 2.11. y 2.13.):

In[10]:= HOMOMORFISMO[G1_,operacion1_,G2_,operacion2_,

grafoF_]:=Module[

⋮ ⋮

In[11]:= QUITARISOMORFOS[tablasgrupos_]:=Module[

⋮ ⋮

In[12]:= grupos=QUITARISOMORFOS[grupos];

Y por último mostramos la tabla del único grupo, salvo isomorfismo que verifica las relaciones:

In[13]:= TableForm[grupos[[1]],

TableHeadings→{Table[StyleForm[i,

FontWeight→"Bold"],{i,n}],

Table[StyleForm[i,FontWeight→"Bold"],{i,n}]],

TableSpacing→{2,2}]

```
Out[]=

$$\begin{array}{cccccccc}
& \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} \\
\mathbf{1} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\mathbf{2} & 2 & 3 & 4 & 1 & 6 & 7 & 8 & 5 \\
\mathbf{3} & 3 & 4 & 1 & 2 & 7 & 8 & 5 & 6 \\
\mathbf{4} & 4 & 1 & 2 & 3 & 8 & 5 & 6 & 7 \\
\mathbf{5} & 5 & 8 & 7 & 6 & 1 & 4 & 3 & 2 \\
\mathbf{6} & 6 & 5 & 8 & 7 & 2 & 1 & 4 & 3 \\
\mathbf{7} & 7 & 6 & 5 & 8 & 3 & 2 & 1 & 4 \\
\mathbf{8} & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1
\end{array}$$

```

□

En los ejemplos anteriores, comprobamos que las tablas verifiquen la propiedad asociativa al final. Si lo hacemos progresivamente mientras se van construyendo las tablas, se evitarán cálculos y la eficacia del algoritmo mejorará sustancialmente:

| FUNCIÓN | COMENTARIOS |
|---|--|
| <pre>GRUOSPEQ[tablaoperaciones_]:=Module[{n, listatablas,listadefinitiva,nuevatabla,cero,cont,fin, listatablastemp,tabla,asociativa},</pre> | La función tendrá como única entrada una tabla de n filas y n columnas con 0 en las posiciones que desconozcamos. Además supondremos que el “1” es el elemento neutro, siendo los elementos del candidato a grupo $G = \{1, 2, \dots, n\}$. |
| <pre>asociativa[operacion_,fila_]:=Module[{op,a,b,c,d,e}, G=operacion[[1]]; op[x_,y_]:=operacion[[Position[G,x]][[1]], Position[G,y][[1]]][[1]][[1]]; asociati=True; c=2; While[asociati && c<=fila, b=2; While[asociati && b<=fila, e=op[b,c]; If[e!=0, a=2; While[asociati && a<=fila, d=op[a,b]; If[d!=0 && op[d,c]!=0 && op[a,e]==0 && op[d,c]==op[a,e],asociati=False,]; a++]];]; b++; c++]; asociati];</pre> | Esta función, de uso local en este programa, hará comprobaciones parciales de la propiedad asociativa en las tablas que vayan a pareciendo. |
| <pre>Print["Calculando grupos que se ajusten a la tabla: ",TableForm[tablaoperaciones]];</pre> | Mostramos la tabla de partida. |
| <pre>n=Length[tablaoperaciones];</pre> | Definimos “n” el número de elementos del grupo. |
| <pre>tiempo=TimeUsed[]; cont=0;</pre> | La variable “tiempo” se utilizará para medir el tiempo empleado en los cálculos y “cont” nos ayudará a estimar los pasos restantes hasta terminar los cálculos. |
| <pre>listadefinitiva={};</pre> | En “listadefinitiva” almacenaremos las |

| | |
|---|---|
| <code>listatablas={tablaoperaciones};</code> | tablas de operaciones que se correspondan con tablas de grupos y “listatablas” contendrá a las tablas mientras se construyen. |
| <code>While[listatablas≠{}, listatablastemp={};</code> | Primer bucle, la condición de parada consistirá en comprobar que en “listatablas” no quede ninguna tabla sin terminar. |
| <code>Do[tabla=listatablas[[j]]; cero=Position[tabla,0]; fin=Length[cero]; cero=cero[[1]];</code> | Segundo bucle que recorrerá todas las tablas a medio construir qué hay en “listatablas”. |
| <code>Do[rellenos=Union[tabla[[cero[[1]]]], Transpose[tabla][[cero[[2]]]]]; If[Intersection[rellenos,{i}]≠{i}, nuevatabla=tabla; If[i==1, nuevatabla[[cero[[1]],cero[[2]]]]=1; nuevatabla[[cero[[2]],cero[[1]]]]=1; , nuevatabla[[cero[[1]],cero[[2]]]]=i;]; If[asociativa[nuevatabla,Min[cero]], If[fin==1, AppendTo[listadefinitiva,nuevatabla]; , AppendTo[listatablastemp,nuevatabla];];];]; ,{i,1,n}]; .,{j,1,Length[listatablas]}];</code> | De forma análoga al programa 2.12. vamos llenando las posiciones que desconocemos de las tablas con los posibles valores, ahora además hacemos una comprobación intermedia de la asociatividad. |
| <code>listatablas=listatablastemp; cont++; Print[(n-1)^2-cont," - ",TimeUsed[]-tiempo, " - ",Length[listatablas]];];</code> | Mostramos resultados intermedios y los tiempos empleados. |
| <code>Print["Tiempo empleado: ",TimeUsed[]-tiempo]; listadefinitiva];</code> | Mostramos el tiempo empleado y los resultados obtenidos. |

Función 2.14. Grupos de orden pequeño.

Ejemplo 2.18. Vamos a buscar todos los grupos de 8 elementos no conmutativos distintos salvo isomorfismo. No perdemos generalidad si suponemos que $G = \{1, 2, 3, 4, 5, 6, 7, 8\}$, 1 es el neutro y que $2 * 3 \neq 3 * 2$, tampoco perdemos generalidad si suponemos que $2 * 3 = 4$ y $3 * 2 = 5$. Por tanto, la tabla de operaciones de partida será:

```
In//]:= n=8;
tabla=Table[If[i==1,j,If[j==1,i,0]],{i,n},{j,n}];
tabla[[2,3]]=4;tabla[[3,2]]=5;
TableForm[tabla]
```

```
Out[]=
      1   2   3   4   5   6   7   8
      2   0   4   0   0   0   0   0
      3   5   0   0   0   0   0   0
      4   0   0   0   0   0   0   0
      5   0   0   0   0   0   0   0
      6   0   0   0   0   0   0   0
      7   0   0   0   0   0   0   0
      8   0   0   0   0   0   0   0
```

Ahora le aplicamos la función 2.14.:

```
In[]:= GRUPOSPEQ[tablaoperaciones_]:=Module[
```

```
: : :
```

```
In[]:= grupos=GRUPOSPEQ[tabla];
```

```
Out[]=
```

| | | | | | | | | |
|--------------------------|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| | 2 | 0 | 4 | 0 | 0 | 0 | 0 | 0 |
| | 3 | 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| Calculando grupos que se | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ajuste a la tabla: | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

```
48 - 0. - 4
47 - 0. - 20
```

```
: : :
```

Tiempo empleado: 54.328

Por último comprobamos cuáles de ellas dan lugar a grupos isomorfos y nos quedamos con un único representante:

```
In[]:= HOMOMORFISMO[G1_,operacion1_,G2_,operacion2_,
grafoF_]:=Module[
```

```
: : :
```

```
In[]:= QUITARISOMORFOS[tablasgrupos_]:=Module[
```

```
: : :
```

```
In[]:= grupos=QUITARISOMORFOS[grupos];
```

```
Out[]= Tiempo empleado: 142.156
```

Ahora mostramos los resultados:

In[7]:=

```
n=8;
TableForm[Table[
{StyleForm[StringJoin["Grupo ",ToString[j],":"]},
FontSize→22],TableForm[grupos[[j]],
TableHeadings→{Table[StyleForm[i,
FontWeight→"Bold"],{i,n}],
Table[StyleForm[i,FontWeight→"Bold"],{i,n}]},
TableSpacing→{2,2}},{j,Length[grupos]}],
TableSpacing→{6,2}]
```

Out[7]=

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|----------|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 2 | 2 | 6 | 4 | 7 | 3 | 8 | 5 | 1 |
| 3 | 3 | 5 | 6 | 2 | 8 | 7 | 1 | 4 |
| 4 | 4 | 3 | 8 | 6 | 1 | 5 | 2 | 7 |
| 5 | 5 | 7 | 2 | 1 | 6 | 4 | 8 | 3 |
| 6 | 6 | 8 | 7 | 5 | 4 | 1 | 3 | 2 |
| 7 | 7 | 4 | 1 | 8 | 2 | 3 | 6 | 5 |
| 8 | 8 | 1 | 5 | 3 | 7 | 2 | 4 | 6 |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|----------|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 2 | 2 | 1 | 4 | 3 | 6 | 5 | 8 | 7 |
| 3 | 3 | 5 | 1 | 7 | 2 | 8 | 4 | 6 |
| 4 | 4 | 6 | 2 | 8 | 1 | 7 | 3 | 5 |
| 5 | 5 | 3 | 7 | 1 | 8 | 2 | 6 | 4 |
| 6 | 6 | 4 | 8 | 2 | 7 | 1 | 5 | 3 |
| 7 | 7 | 8 | 5 | 6 | 3 | 4 | 1 | 2 |
| 8 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

□

Aun así, el tiempo necesario para calcular todos los grupos de 8 elementos o más sigue siendo excesivo, en parte podemos evitarlo usando otras propiedades matemáticas. En el capítulo 3 se estudian algunas de ellas.

4. EJERCICIOS

Ejercicio 2.1. Consideramos el par $(G, *)$, donde $G = \{a, b, c, d, e, f, g, h\}$ y la operación viene dada por la siguiente tabla:

| * | a | b | c | d | e | f | g | h |
|---|---|---|---|---|---|---|---|---|
| a | a | b | c | d | e | f | g | h |
| b | b | c | d | a | f | g | h | e |
| c | c | d | a | b | g | h | e | f |
| d | d | a | b | c | h | e | f | g |

| | | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| <i>e</i> | <i>e</i> | <i>f</i> | <i>g</i> | <i>h</i> | <i>a</i> | <i>b</i> | <i>c</i> | <i>d</i> |
| <i>f</i> | <i>f</i> | <i>g</i> | <i>h</i> | <i>e</i> | <i>b</i> | <i>c</i> | <i>d</i> | <i>a</i> |
| <i>g</i> | <i>g</i> | <i>h</i> | <i>e</i> | <i>f</i> | <i>c</i> | <i>d</i> | <i>a</i> | <i>b</i> |
| <i>h</i> | <i>h</i> | <i>e</i> | <i>f</i> | <i>g</i> | <i>d</i> | <i>a</i> | <i>b</i> | <i>c</i> |

Tabla 2.4.

Comprobar si es un grupo commutativo. □

Ejercicio 2.2. Consideramos el par $(G, *)$, donde $G = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$ y la operación viene dada por la siguiente tabla:

| * | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 2 | 2 | 1 | 4 | 3 | 6 | 5 | 8 | 7 | 10 | 9 | 12 | 11 |
| 3 | 3 | 5 | 1 | 6 | 2 | 4 | 9 | 11 | 7 | 12 | 8 | 10 |
| 4 | 4 | 6 | 2 | 5 | 1 | 3 | 10 | 12 | 8 | 11 | 7 | 9 |
| 5 | 5 | 3 | 6 | 1 | 4 | 2 | 11 | 9 | 12 | 7 | 10 | 8 |
| 6 | 6 | 4 | 5 | 2 | 3 | 1 | 12 | 10 | 11 | 8 | 9 | 7 |
| 7 | 7 | 8 | 9 | 10 | 11 | 12 | 1 | 2 | 3 | 4 | 5 | 6 |
| 8 | 8 | 7 | 10 | 9 | 12 | 11 | 2 | 1 | 4 | 3 | 6 | 5 |
| 9 | 9 | 11 | 7 | 12 | 8 | 10 | 3 | 5 | 1 | 6 | 2 | 4 |
| 10 | 10 | 12 | 8 | 11 | 7 | 9 | 4 | 6 | 2 | 5 | 1 | 3 |
| 11 | 11 | 9 | 12 | 7 | 10 | 8 | 5 | 3 | 6 | 1 | 4 | 2 |
| 12 | 12 | 10 | 11 | 8 | 9 | 7 | 6 | 4 | 5 | 2 | 3 | 1 |

Tabla 2.5.

Utilizar todas las funciones y métodos expuestos en el capítulo para:

- a) Comprobar si es un grupo.
- b) Determinar si es abeliano.

□

Ejercicio 2.3. Producto Cartesiano de Grupos Finitos. Dados $(G_1, *_1)$ y $(G_2, *_2)$ dos grupos finitos, el par $(G_1 \times G_2, *)$ con la operación:

$$(a_1, a_2) * (b_1, b_2) = (a_1 *_1 b_1, a_2 *_2 b_2)$$

es un grupo. Introducimos los dos grupos en el ordenador como en la sección 3 y cambiamos la notación para no tener conflictos:

| Grupo $(G_1, *_1)$ | Grupo $(G_2, *_2)$ |
|--|--|
| $G1=CONJUNTO G_1;$ $operacion1=TABLA DE OPERACIONES DE G_1;$ $op1[x,y]:=operacion1[[Position[G1,x][[1]],$ $Position[G1,y][[1]]][[[1]][[1]]];$ | $G2= CONJUNTO G_2;$ $operacion2= TABLA DE OPERACIONES DE G_2;$ $op2[x,y]:=operacion2[[Position[G2,x][[1]],$ $Position[G2,y][[1]]][[[1]][[1]]];$ |

El conjunto producto cartesiano podemos introducirlo como se indica en sección 2 del capítulo 7 de [25] y la operación anterior quedaría definida como sigue:

$$In[]:= \quad op[\{a,b\},\{c,d\}]:=\{op1[a,c],op2[b,d]\};$$

Comprobarlo para el producto cartesiano de los grupos de los ejercicios 2.1 y 2.2. y calcular:

- a) La tabla de operaciones.
- b) ¿Es abeliano?

□

Ejercicio 2.4. Calcular la tabla de operaciones del grupo conmutativo $\mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_2$.

□

Ejercicio 2.5. Comprobar que no existen grupos no conmutativos de 5 elementos o menos.

□

Ejercicio 2.6. Comprobar que los conjuntos con las operaciones dadas de los ejemplos 3.6., 3.7. y 3.11. son realmente grupos. ¿Cuáles son abelianos?.

□

Ejercicio 2.7. Dado un grupo $(G, *)$ se define el centro de G como el subconjunto:

$$Z(G) = \{g \in G \mid a * g = g * a \text{ para cada } a \in G\}.$$

Crear un programa que determine el centro de un grupo finito. Aplicarlo al grupo del ejemplo 3.7.

□

Ejercicio 2.8. Calcular $Z(G)$ de los grupos de los ejercicios 2.1. y 2.2.

□

Ejercicio 2.9. Escribir una rutina que determine, si existe, el simétrico de un elemento observando directamente la tabla de operaciones.

□

Ejercicio 2.10. En el ejemplo 2.13. se estudia la estructura de grupo de $G = \mathbb{R} \times \mathbb{R} - \{(0,0)\}$ con la operación $(a, b) \cdot (c, d) = (ac - bd, ad + bc)$, si identificamos un complejo cualquiera $a + bi$ con el par (a, b) , es evidente que (G, \cdot) es (\mathbb{C}^*, \cdot) . Mathematica es capaz de usar complejos directamente, la unidad compleja, $i = \sqrt{-1}$, está implementada y puede introducirse simplemente como **I**, de esta forma un complejo cualquiera puede ser introducido como:

$$\mathbf{a} + \mathbf{b}^*\mathbf{I}$$

La identificación es fácil de trasladar usando las funciones **Re[]** y **Im[]** que calculan la parte real e imaginaria respectivamente.

$$\{\mathbf{Re[a+b*I]}, \mathbf{Im[a+b*I]}\}$$

Aprovechado que Mathematica puede usar datos de tipo complejo, repetir el ejemplo 2.26. aprovechando este extremo.

□

Ejercicio 2.11. Sea $f: \mathbb{Z}_2 \times \mathbb{Z}_4 \rightarrow \mathbb{Z}_2$ la aplicación definida por:

$$f(\bar{x}, \bar{y}) = \overline{xy}$$

Comprobar si f es un homomorfismo de grupos. \square

Ejercicio 2.12. Sea G el grupo del ejemplo 3.16. definimos la aplicación $f: G \rightarrow \mathbb{Z}_2$ dada por $f(1) = f(4) = f(5) = \bar{0}$ y $f(2) = f(3) = f(6) = \bar{1}$.

Comprobar que f es un homomorfismo de grupos. \square

Ejercicio 2.13. Núcleo e imagen. Sean $(G_1, *_1)$ y $(G_2, *_2)$ dos grupos, y sea $f: G_1 \rightarrow G_2$ un homomorfismo de grupos, definimos el núcleo de f como

$$\ker(f) = \{g \in G \mid f(g) = e_2\},$$

y definimos la imagen de f como

$$Im(f) = \{f(g) \mid g \in G\}$$

donde e_2 es el elemento neutro de G_2 .

Escribir rutinas que calculen el núcleo y la imagen de un homomorfismo cualquiera entre grupos finitos. Aplicarlas a todos los homomorfismos que han ido apareciendo en los ejemplos y ejercicios del capítulo. \square

Ejercicio 2.14. Sea $G = \{\text{False}, \text{True}\}$, y consideremos la operación interna $\wedge: G \times G \rightarrow G$ definida por la siguiente tabla (conectiva AND):

| \wedge | False | True |
|----------|-------|-------|
| False | False | False |
| True | False | True |

- a) Comprobar si (G, \wedge) es un grupo conmutativo.
- b) Comprobar si G es grupo con alguna de las siguientes conectivas: $\vee, \uparrow, \downarrow, \rightarrow, \leftrightarrow, \oplus$.
- c) Sea $*$ una de las conectivas anteriores para la cual $(G, *)$ es un grupo. ¿Es isomorfo a $(\mathbb{Z}_2, +)$?
- d) ¿Qué podríamos decir sobre G^n con la operación inducida en el producto cartesiano por $*$? Calcular el elemento neutro y el simétrico si existe de un elemento cualquiera.

\square

Ejercicio 2.15. Anillos y cuerpos. Un anillo es una terna $(R, +, \cdot)$ formada por un conjunto y dos leyes de composición interna verificando las siguientes propiedades:

- a) $(R, +)$ es un grupo conmutativo (donde ‘0’ denotará al elemento neutro).
- b) Asociativa del producto: $a(bc) = (ab)c$ para cada $a, b, c \in R$.
- c) E. neutro del producto (lo denotaremos por ‘1’):

$$\begin{aligned} 1a &= a1 && \text{para cada } a \in R. \\ a(b+c) &= ab+ac \\ (b+c)a &= ba+ca && \text{para cada } a, b, c \in R. \end{aligned}$$
- d) Distributiva:

Si además el producto verifica la propiedad conmutativa: $ab = ba$ para cada $a, b \in R$, se dirá que es un anillo conmutativo.

Si además de la propiedad conmutativa, el producto verifica la propiedad de elemento inverso: para cada $a \in R - \{0\}$, existe $a^{-1} \in R$, tal que $aa^{-1} = a^{-1}a = 1$, entonces se dirá que es un cuerpo.

En el capítulo se crean distintos test para comprobar las propiedades que debe verificar una operación para dotar a un conjunto finito de estructura de grupo. De forma similar generar los programas necesarios que faltan para comprobar si un conjunto finito con dos operaciones es un anillo, anillo conmutativo o cuerpo.

□

Ejercicio 2.16. De forma análoga al ejercicio anterior programar rutinas que verifiquen las estructuras de retículo y álgebra de Boole de conjuntos finitos (en el capítulo 9 de [25] encontraremos algunos de estos test ya programados). Crear un test para espacios vectoriales finitos, álgebras finitas, módulos finitos,...

□

Ejercicio 2.17. Sea $U = \{(x, y) \in \mathbb{R} \times \mathbb{R} \mid x + y = 0\}$, demostrar que $(U, +)$ es un grupo abeliano.

□

Ejercicio 2.18. Sea $W = \{(x, y, z) \in \mathbb{C} \times \mathbb{C} \times \mathbb{C} \mid 2x + y - z = 0\}$, demostrar que $(W, +)$ es un grupo abeliano.

□

Ejercicio 2.19. Definir un grupo cíclico de 20 elementos, establecer un isomorfismo entre éste y \mathbb{Z}_{20} .

□

Ejercicio 2.20. Para el manejo y estudio de los grupos existen varios paquetes de funciones disponibles en la red Internet, por ejemplo podemos buscar algunos en la página de Wolfram Research, en estos paquetes encontramos funciones para realizar tareas similares a las que resolvemos en este capítulo. Encontrar y analizar el funcionamiento de alguno de estos paquetes, compararlo con lo desarrollado en el capítulo.

□

Ejercicio 2.21. Sea G el conjunto de todas las letras distintas que aparecen en su primer apellido. Definir en G una operación interna $*$ que le dote de estructura de grupo, comprobarlo explícitamente.

□

Ejercicio 2.22. Sea G el conjunto de todos los dígitos distintos de su DNI. Definir en G una operación interna $*$, si es posible, que dote a G de estructura de grupo no conmutativo, comprobarlo explícitamente, y si no existe razonar el porqué.

□

Ejercicio 2.23. Sea $X = \{1, 2, 3\}$ y sea $\mathcal{P}(X)$ el conjunto de todos los subconjuntos de X . En $\mathcal{P}(X)$ definimos la siguiente operación interna, para $A, B \in \mathcal{P}(X)$,

$$A + B = (A \cup B) - (A \cap B).$$

Comprobar que $(\mathcal{P}(X), +)$ es un grupo abeliano.

□

Ejercicio 2.24. Calcular todos los grupos distintos salvo isomorfismo con a lo sumo 5 elementos.

□

Ejercicio 2.25. Sea $X = \{1, 2, 3, 4, 5, 6\}$, encontrar, si existe, una operación \sqcup que dote a X de una estructura de grupo no conmutativo donde 1 sea el elemento neutro y 2 sea el inverso del 3.

□

Ejercicio 2.26. Encontrar, si existe, un grupo G de 8 elementos con elemento neutro e verificando: $x^2 = e$ para cada $x \in G$.

□

Ejercicio 2.27. Encontrar, si existe, un grupo G conmutativo formado por todos los dígitos distintos de su DNI.

□

Ejercicio 2.28. Encontrar, si existen, todos los grupos de ocho elementos: 1, 2, 3, 4, 5, 6, 7, 8, cuya tabla de operaciones sea de la forma:

| * | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 2 | 2 | 1 | ? | ? | ? | ? | ? | ? |
| 3 | 3 | ? | 1 | ? | ? | ? | ? | ? |
| 4 | 4 | ? | ? | 1 | ? | ? | ? | ? |
| 5 | 5 | ? | ? | ? | 1 | ? | ? | ? |
| 6 | 6 | ? | ? | ? | ? | 1 | ? | ? |
| 7 | 7 | ? | ? | ? | ? | ? | 1 | ? |
| 8 | 8 | ? | ? | ? | ? | ? | ? | 1 |

□

Ejercicio 2.29. Sea G el conjunto de todos los dígitos distintos de su DNI módulo 6 junto con la letra de su DNI. Definir, si existe, una operación interna $*$ en G , donde la letra de su DNI sea el elemento neutro, que dote a G de estructura de grupo no conmutativo, comprobarlo explícitamente. Si no existe dicha operación, definir en G una operación interna $*$, donde la letra de su DNI sea el elemento neutro, que verifique la propiedad de elemento simétrico y no sea conmutativa, comprobar explícitamente que las verifica y que no es un grupo.

□

Ejercicio 2.30. Grupo de los cuaternios. Consideramos las siguientes matrices cuadradas regulares 2×2 :

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, B = \begin{pmatrix} \sqrt{-1} & 0 \\ 0 & \sqrt{-1} \end{pmatrix}, C = \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix} \text{ y } D = \begin{pmatrix} 0 & \sqrt{-1} \\ \sqrt{-1} & 0 \end{pmatrix}.$$

Comprobar que el conjunto $Q_2 = \{A, -A, B, -B, C, -C, D, -D\}$ con el producto de matrices es un grupo no conmutativo.

□

3. SUBGRUPOS, EL GRUPO COCIENTE Y GENERADORES

Este capítulo está dedicado a profundizar en los conceptos básicos de la teoría de grupos: subgrupo, grupo cociente y grupo generado por un conjunto de elementos o generadores. En la mayoría de casos, en la misma línea del capítulo anterior, el estudio computacional está restringido al caso finito. La parte final la hemos dedicado a realizar un estudio más profundo sobre el cálculo de los subgrupos de un grupo finito, introduciendo de manera superficial aspectos relacionados con la eficacia y optimización, aunque con el ánimo de progresar en aspectos teóricos relacionados con la teoría de grupos, esto es, matemáticos más que computacionales. Conforme vamos analizando estos conceptos observamos que el Mathematica no es la herramienta más adecuada para realizar algunos cálculos y no por su complejidad, sino por las limitaciones propias de su lenguaje de programación en cuanto a eficacia, en el manejo de grupos de ordenes grandes⁷, tanto por el tiempo de proceso como por la gestión particular que hace de la memoria. Si queremos continuar con dichos cálculos y aspirar a analizarlos con más detalle, sería recomendable trasladarse a otras plataformas o aplicaciones de programación más apropiadas. También hemos progresado en la optimización del cálculo de grupos de orden pequeño teniendo en cuenta las cuestiones desarrolladas en el capítulo.

1. SUBGRUPOS

Sea $(G, *)$ un grupo, diremos que un subconjunto⁸ no vacío H de G es un subgrupo de G si verifica:

- Sea $e \in G$ el elemento neutro de G entonces $e \in H$.
- Para cada $h \in H$, su simétrico $h' \in H$.
- Es cerrado para la operación $*$, esto es, para cada $h_1, h_2 \in H$, $h_1 * h_2 \in H$.

⁷ Existen funciones en paquetes concretos que resuelven los problemas propuestos con gran eficacia, sin embargo, no es nuestro objetivo resolverlos con funciones prediseñadas, sino dar respuestas en la medida de lo posible, por medio de herramientas de programación habituales.

⁸ Aunque en las comprobaciones posteriores se suponen subconjuntos no vacíos de partida, es fácil determinar si un conjunto es subconjunto de otro con Mathematica:

`Sort[Intersection[A,B]]==Sort[B] o Sort[Union[A,B]]==Sort[A]`

Determinará si el conjunto A contiene al conjunto B . Por otra parte, un conjunto será vacío si: $A==\{\}$.

Obsérvese que en tal caso $(H, *)$ también tiene estructura de grupo.

Cuando tenemos un grupo finito podemos comprobar estas propiedades fácilmente como muestra el ejemplo 3.1. En el caso infinito y al igual que ocurre con los grupos infinitos todo dependerá del tipo de datos y operación dada, en muchos casos también podremos hacer comprobaciones directas (véase el epígrafe 7 de este mismo capítulo).

Ejemplo 3.1. Consideramos el mismo conjunto y operación interna del ejemplo 2.1., comprobamos si los subconjuntos $H_1 = \{a, b\}$ y $H_2 = \{b, c\}$ de G son subgrupos.

Introducimos el grupo, la operación y la función 2.1. como hacemos siempre,

```
In[]:= G={a,b,c,d};  
operacion={{a,b,c,d},{b,a,d,c},{c,d,b,a},{d,c,a,b}};  
op[x_,y_]:=operacion[[Position[G,x][[1]],  
Position[G,y][[1]]][[1]][[1]]];
```

Comprobamos las tres propiedades por separado:

- a) Calculamos el elemento neutro con el programa 2.4. y comprobamos si pertenece a H_1 y H_2 .

```
In[]:= ElementoNeutro="No existe";  
:  
:
```

```
Out[]=  
Elemento Neutro: a
```

Como $a \notin H_2$, ya podemos asegurar que H_2 no es subgrupo.

- b) Continuamos estudiando H_1 , que si contiene al neutro a , ahora calculamos los simétricos de los elementos de H_1 utilizando la función 2.7.

```
In[]:= simetrico[x_]:=Module[{simetrico,CONTADORi},  
:  
:]
```

La aplicamos a los elementos que nos interesan,

```
In[]:= simetrico[a]
```

```
Out[]=  
a
```

```
In[]:= simetrico[b]
```

```
Out[]=  
b
```

Luego la propiedad segunda también es verificada por H_1 .

c) Por último comprobamos que es cerrado para la operación:

$$In[]:= \quad \mathbf{op[a,a]}$$

$$Out[]:= \quad a$$

$$In[]:= \quad \mathbf{op[a,b]}$$

$$Out[]:= \quad b$$

$$In[]:= \quad \mathbf{op[b,a]}$$

$$Out[]:= \quad b$$

$$In[]:= \quad \mathbf{op[b,b]}$$

$$Out[]:= \quad a$$

En consecuencia H_1 si es subgrupo. □

No es difícil generar un test que junte todas las comprobaciones del ejemplo anterior como se propone en el ejercicio 3.5. Utilizando la siguiente caracterización también podemos implementar directamente un test:

Proposición 3.1. Equivalentemente, diremos que H es subgrupo de G , si para cada par de elementos $x, y \in H$, entonces $x * y' \in H$, donde y' es el simétrico de y . □

Implementamos la caracterización para grupos finitos con el siguiente programa:

| FUNCIÓN | COMENTARIOS |
|--|---|
| G=GRUPO; operacion=TABLA DE OPERACIONES DE G; | Introducimos el grupo “G”. |
| op[x_,y_]:=... | Introducimos la función 2.1. |
| ElementoNeutro=... | Calculamos el elemento neutro con el programa 2.4. |
| simetrico[x_]:=... | Definimos la función 2.7. |
| SUBGRUPO[H_]:=Module[{subgrupo,CONTADORi,CONTADORj}, subgrupo=True; CONTADORi=1; | La función tendrá como argumento a un subconjunto “H” de “G”. |
| While[subgrupo && CONTADORi<=Length[H], CONTADORj=1; While[subgrupo && CONTADORj<=Length[H], If[Intersection[{op[H][[CONTADORi]], simetrico[H[[CONTADORj]]]},H]=={}, subgrupo=False;]; | Comprobamos si en algún caso no se verifica la condición equivalente. |

| | |
|---|-----------------------------|
| CONTADORj++; }; CONTADORi++; }; subgrupo | |
| | Se muestran los resultados. |

Función 3.1. Subgrupos.

Aunque por razones de eficacia utilizaremos habitualmente la función anterior como test de subgrupo, para mayor comodidad del lector, podemos integrar todas las funciones previas necesarias para el correcto funcionamiento de 3.1. en una única:

| FUNCIÓN | COMENTARIOS |
|---|---|
| SUBGRUPO[H_G_,operacion_]:=Module[{subgrupo, CONTADORi,CONTADORj,op,simetrico, ElementoNeutro,neutro}, | La función tendrá como argumento a un subconjunto “H” de “G” y al grupo. |
| op[x_,y_]:=operacion[[Position[G,x][[1]], Position[G,y][[[1]]][[[1]]][[1]]]; | Introducimos la función 2.1.bis. |
| ElementoNeutro="No existe"; ⋮ ⋮ | Calculamos el elemento neutro con el programa 2.4. |
| simetrico[x_]:=Module[{simetrico,CONTADORi}, ⋮ ⋮ | Definimos la función 2.7. |
| subgrupo=True; ⋮ ⋮ subgrupo | Comprobamos si en algún caso no se verifica la condición equivalente con el mismo código de la función 3.1. |
| | . |

Función 3.1.bis. Subgrupos.

Ejemplo 3.2. Consideramos el mismo conjunto y operación interna del ejemplo 2.1. Comprobar si los subconjuntos $H_1 = \{a, b\}$ y $H_2 = \{b, c\}$ de G son subgrupos usando 3.1.

Como siempre definimos las funciones e introducimos los programas previos necesarios, 2.1., 2.4. y 2.7.

In[]:= **G={a,b,c,d};**
operacion={{a,b,c,d},{b,a,d,c},{c,d,b,a},{d,c,a,b}};
op[x_,y_]:=operacion[[Position[G,x][[1]],
Position[G,y][[[1]]][[[1]]][[1]]];

In[]:= **ElementoNeutro="No existe";**

⋮ ⋮

In[]:= **simetrico[x_]:=Module[{simetrico,CONTADORi},**

⋮ ⋮

Definimos la función 3.1.

In[]:= **SUBGRUPO[H_]:=Module[{subgrupo,CONTADORi,**
 ⋮ ⋮
 }

Comprobamos si H_1 y H_2 son subgrupos:

In[]:= **SUBGRUPO[{a,b}]**
Out[]= True
In[]:= **SUBGRUPO[{b,c}]**
Out[]= False

Con 3.1.bis. sería igual, pero ahora no será necesario definir o usar previamente 2.1., 2.4. y 2.7.:

In[]:= **SUBGRUPO[H_,G_,operacion_]:=Module[{subgrupo,**
 ⋮ ⋮
 }
In[]:= **SUBGRUPO[{a,b},G,operacion]**
Out[]= True
In[]:= **SUBGRUPO[{b,c},G,operacion]**
Out[]= False

□

2. CLASES LATERALES. EL TEOREMA DE LAGRANGE

Sea $(G, *)$ un grupo, diremos que H es un subgrupo propio de G , si $1 < |H| < |G|$. Los únicos subgrupos no propios o improperios de G son $\{e\}$ y el mismo G . Entonces para cada subgrupo propio $H \subseteq G$ podemos definir dos relaciones de equivalencia sobre G . Para $g_1, g_2 \in G$, diremos que,

$$g_1 R_1 g_2 \Leftrightarrow g_1' * g_2 \in H \quad (1)$$

$$g_1 R_2 g_2 \Leftrightarrow g_1 * g_2' \in H \quad (2)$$

para estas relaciones de equivalencia, las clases equivalencia de un $x \in G$ son:

$$\overline{x}^1 = \{y \in G \mid x R_1 y\} = \{y \in G \mid x' * y \in H\} = \{x * h \mid h \in H\} = x * H \quad (1)$$

$$\overline{x}^2 = \{y \in G \mid x R_2 y\} = \{y \in G \mid y * x' \in H\} = \{h * x \mid h \in H\} = H * x \quad (2)$$

a $x * H$ la llamaremos clase de equivalencia por la izquierda módulo H y a $H * x$ la llamaremos clase de equivalencia por la derecha módulo H. Obsérvese, que para grupos finitos, $|H|$ coincide con el número de elementos (cardinalidad) de $x * H$ o $H * x$ para cada $x \in G$, luego es evidente que el número de elementos de G/R_1 y G/R_2 coincide. En grupos finitos, llamaremos índice de H en G al número de clases de equivalencia de G/R_1 o G/R_2 y lo representaremos por $[G : H]$. En consecuencia tendremos el siguiente teorema, cuya demostración podemos encontrar en el epígrafe 4.2 del libro “Números, Grupos y Anillos” (Dorronsoro, J. y Hernández, E. [15]):

Teorema 3.1. Teorema de Lagrange. Sea $(G, *)$ un grupo finito y sea $H \subseteq G$ un subgrupo, entonces:

$$[G : H] = \frac{|G|}{|H|}.$$

□

Una consecuencia inmediata del teorema es que el cardinal de un subgrupo siempre debe dividir al cardinal del grupo (esto es, el número de elementos de un subgrupo H de G finito, será un divisor positivo de $|G|$).

Para un grupo finito es fácil programar una rutina que calcule las clases de equivalencia por ambos lados (laterales).

| FUNCIÓN | COMENTARIOS |
|--|---|
| <pre>Clases[x_,H_,G_,operacion]:=Module[{claseizquierda,clasederecha,CONTADORi,op},</pre> | Las entradas serán: un elemento “x” de “G”, un subgrupo “H” y el grupo junto con su tabla de operaciones. |
| <pre>op[a_,b_]:=operacion[[Position[G,a][[1]], Position[G,b][[1]]][[1]][[1]]];</pre> | Función 2.1.bis. |
| <pre>claseizquierda={}; clasederecha={}; Do[AppendTo[claseizquierda, op[x,H[[CONTADORi]]]]; AppendTo[clasederecha,op[H[[CONTADORi]],x]]; ,{CONTADORi,1,Length[H]}]; Print[x,"*H =",claseizquierda,"; H*",x, " = ",clasederecha]</pre> | Se calculan ambas clases. |
|] | Se muestran los resultados. |

Función 3.2. Clases laterales.

Ejemplo 3.3. Consideramos el mismo conjunto y operación interna del ejemplo 2.1., calculamos las clases laterales de a y c para el subgrupo $H = \{a, b\}$.

Introducimos el grupo, la operación:

```
In]:= G={a,b,c,d};  
operacion={{a,b,c,d},{b,a,d,c},{c,d,b,a},{d,c,a,b}};
```

Definimos la función 3.2.

```
In]:= Clases[x_,H_,G_,operacion_]:=Module[
```

```
];];
```

Calculamos las clases laterales de a :

```
In]:= Clases[a,{a,b},G,operacion]
```

```
Out]= a*H = {a,b}; H*a = {a,b}
```

Y las clases laterales de c :

```
In]:= Clases[c,{a,b},G,operacion]
```

```
Out]= c*H = {c,d}; H*c = {c,d}
```

□

3. SUBGRUPOS NORMALES Y GRUPOS COCIENTES

Un subgrupo H de G se dice que es normal y denotaremos $H \trianglelefteq G$, si $x * H = H * x$, para cada $x \in G$, esto es, coinciden las clases laterales de cada elemento de G . Podemos caracterizar el concepto de subgrupo normal por la siguiente proposición (en [15] encontraremos una demostración):

Proposición 3.2. Sea $(G, *)$ un grupo y H un subgrupo de G . Equivalen:

- i. H es normal.
- ii. $x * h * x' \in H$ para todo $x \in G$ y $h \in H$ (donde x' es el simétrico de x).
- iii. $(x * H) * (y * H) = (x * y) * H$ para cada $x, y \in G$.

□

Si un subgrupo es normal, coinciden los conjuntos cocientes $G/R_1 = G/R_2$ que denotaremos por G/H y dicho conjunto cociente con la operación interna:

$$(x * H) * (y * H) = (x * y) * H$$

es un grupo, que llamaremos grupo cociente de G sobre H .

Si G es conmutativo es evidente que cualquier subgrupo suyo es normal.

Usando la función 3.2., podemos determinar para grupos finitos cuándo un subgrupo es normal, y en caso afirmativo calcular el grupo cociente. Para ello, en primer lugar generamos un test para comprobar si un subgrupo es normal, lo hacemos desde la definición, calculamos las clases laterales de cada elemento y comprobamos si son iguales.

| FUNCIÓN | COMENTARIOS |
|---|---|
| <pre>NORMAL[H_,G_,operacion_]:=Module[{normal, clasederecha,claseizquierda,CONTADORi, CONTADORj,op}, op[x_,y_]:=operacion[[Position[G,x][[1]], Position[G,y][[1]]][[1]][[1]]]; normal=True; Do[claseizquierda={};clasederecha={}; Do[AppendTo[claseizquierda, op[G[[CONTADORj]],H[[CONTADORi]]]]; AppendTo[clasederecha, op[H[[CONTADORi]],G[[CONTADORj]]]]; ,{CONTADORi,Length[H]}]; If[!(Sort[claseizquierda]==Sort[clasederecha]) , normal=False;Break[]]; ,{CONTADORj,Length[G]}]; normal</pre> | Las entradas serán: el subgrupo que pretendemos testear y el grupo. |
| | Introducimos la función 2.1.bis. |
| | |
| | Se comprueba si es normal. |
| | Se muestran los resultados. |

Función 3.3. Subgrupos Normales.

En segundo lugar calculamos el grupo cociente suponiendo que el subgrupo sea normal.

| PROGRAMA | COMENTARIOS |
|---|--|
| <pre>G=GRUPO; operacion=TABLA DE OPERACIONES DE G;</pre> | Introducimos el grupo. |
| <pre>op[x_,y_]:=...</pre> | Introducimos la función 2.1. |
| <pre>H=SUBGRUPO NORMAL;</pre> | Introducimos un subgrupo normal H de G . |
| <pre>cociente={}; evaluados={}; Do[If[Intersection[{G[[CONTADORj]]},evaluados]=={}, clase={}; Do[AppendTo[clase,op[G[[CONTADORj]], H[[CONTADORi]]]]; ,{CONTADORi,1,Length[H]}]; AppendTo[cociente,clase]; evaluados=Union[clase,evaluados];];];</pre> | Calculamos todas las clases. |

| | |
|---------------------------------------|-----------------------------|
| ,{CONTADORj,1,Length[G]}; cociente | Mostramos el grupo cociente |
|---------------------------------------|-----------------------------|

Programa 3.4. Grupo cociente.

Una vez calculado el grupo cociente, es fácil operar entre sus elementos, pues recordemos que $(x * H) * (y * H) = (x * y) * H$, y con el ordenador no habría más que escribir:

Clases[op[x,y],H,G,operacion]

De esta forma, podemos determinar la tabla de operaciones del grupo cociente.

| PROGRAMA | COMENTARIOS |
|---|--|
| G=GRUPO; operacion=TABLA DE OPERACIONES DE G; | Introducimos el grupo. |
| op[x_,y_]:=... | Introducimos la función 2.1. |
| H=SUBGRUPO NORMAL; | Introducimos un subgrupo normal H de G . |
| operacioncociente=Table["-" ,{CONTADORi,Length[cociente]} ,{CONTADORj,Length[cociente]}]; | En “operacioncociente” introduciremos la tabla de operaciones del grupo cociente |
| Dol Do[Do[If[Intersection[{op[cociente][[CONTADORi]][[1]], cociente[[CONTADORj]][[1]]}, cociente[[CONTADORk]]]=={}, operacioncociente[[CONTADORi, CONTADORj]]= cociente[[CONTADORk]][[1]];Break[]];; ,{CONTADORk,Length[cociente]}]; ,{CONTADORi,Length[cociente]}]; ,{CONTADORj,Length[cociente]}]; | Calculamos la tabla de operaciones. |
| TableForm[operacioncociente,TableHeadings→ {Transpose[cociente][[1]],Transpose[cociente][[1]]}] | Salida de resultados. |

Programa 3.5. Tabla de operaciones del grupo cociente.

Ejemplo 3.4. Consideramos el mismo conjunto y operación interna del ejemplo 2.1., comprobamos si el subgrupo $H = \{a, b\}$ es normal y en caso afirmativo calcular el grupo cociente.

Introducimos el grupo y la operación:

In[]:= G={a,b,c,d};
operacion={{a,b,c,d},{b,a,d,c},{c,d,b,a},{d,c,a,b}};

Definimos la función 3.3.

In[]:= NORMAL[H_,G_,operacion_]:=Module[

$$\vdots \qquad \vdots$$

Comprobamos que en efecto H es normal:

In[]:= **NORMAL[{a,b},G,operacion]**

Out[]= True

Puesto que es normal, calculamos el grupo cociente usando 3.4., previamente definimos la función 2.1.:

In[]:= **op[x_,y_]:=operacion[[Position[G,x][[1]], Position[G,y][[1]]][[1]][[1]]];**

In[]:= **H={a,b};**

$$\vdots \qquad \vdots$$

Out[]= $\{\{a,b\}, \{c,d\}\}$

Por tanto $|G/H| = [G : H] = 2$ y las clases de equivalencia son:

$$a * H = b * H = \{a, b\} \quad \text{y} \quad c * H = d * H = \{c, d\}$$

Definimos la función 3.2.

In[]:= **Clases[x_,H_,G_,operacion_]:=Module[**

$$\vdots \qquad \vdots$$

y como

In[]:= **Clases[op[a,a],H,G,operacion]**

Clases[op[a,c],H, G,operacion]

Clases[op[c,a],H, G,operacion]

Clases[op[c,c],H, G,operacion]

Out[]= $a * H = \{a, b\} ; H * a = \{a, b\}$

$c * H = \{c, d\} ; H * c = \{c, d\}$

$c * H = \{c, d\} ; H * c = \{c, d\}$

$b * H = \{b, a\} ; H * b = \{b, a\}$

por tanto concluimos que la tabla de operaciones del grupo cociente es:

| G/H | $a * H$ | $c * H$ |
|---------|---------|---------|
| $a * H$ | $a * H$ | $c * H$ |
| $c * H$ | $c * H$ | $a * H$ |

O bien la calculamos usando 3.5.

In[]:= H={a,b};

⋮ ⋮

Out[]=

| | | |
|---|---|---|
| a | c | |
| a | a | c |
| c | c | a |

□

4. CÁLCULO DE TODOS LOS SUBGRUPOS DE UN GRUPO FINITO

Dado un grupo finito, podemos considerar el conjunto de sus partes⁹: $\mathcal{P}(G)$, y analizar cuáles de estos subconjuntos son subgrupos de G . De hecho y recordando el teorema de Lagrange, bastará con comprobarlo sólo para aquellos subconjuntos de G cuyo cardinal divida al cardinal de G . Para realizar el cálculo de todos los subgrupos de un grupo, primero calcularemos todos sus subconjuntos cuyo cardinal divida al del grupo y después comprobaremos uno por uno, cuáles de ellos son subgrupos. Para ello utilizaremos una función del paquete de Matemática Discreta de Mathematica: `<<Combinatorica` , y la función que nos interesará usar es:

KSubsets[conjunto,n]

que calcula todos los subconjuntos con “n” elementos de “conjunto”. Podemos aprender más sobre partes de un conjunto en la sección 3 del capítulo 7 de [25]. Esta función no es habitual en otras plataformas distintas a Mathematica, incluimos una rutina alternativa hecha con herramientas habituales de programación que hace lo mismo que KSubsets[] en el ejercicio 3.16.

Ejemplo 3.5. Sea $G = \{a, b, c, d, e, f\}$ un grupo, determinar todos los subconjuntos de G que podrían ser subgrupos de G .

Introducimos el grupo:

In[]:= G={a,b,c,d,e,f};

Calculamos todos los divisores positivos del cardinal de G . Para calcular los divisores positivos utilizaremos la función Divisors[] de Mathematica, (podemos encontrar más información sobre esta función en la ayuda de Mathematica). Aunque sería muy fácil simular el funcionamiento de Divisors[], bastaría con definir:

⁹ El conjunto de las partes de un conjunto es el conjunto de todos sus subconjuntos.

| FUNCIÓN | COMENTARIOS |
|--|---|
| <pre>DIVISORES[n_]:=Module[{i}, divisores={}; Do[If[Mod[n,i]==0,AppendTo[divisores,i]] ,{i,1,n}]; divisores]</pre> | Calculamos todos los divisores usando la función Mod[]. |

Función 3.6. Divisores.

In[]:= **divisores=Divisors[Length[G]]**

Out[] = {1, 2, 3, 6}

O bien,

In[]:= **DIVISORES[6];**

Usamos KSubsets[] para cada divisor:

In[]:= **<<Combinatorica`**

In[]:= **candidatos={};**
Do[AppendTo[candidatos,KSubsets[G,divisores[[i]]]],
,{i,Length[divisores]}];
candidatos

Out[] = { {{a}, {b}, {c}, {d}, {e}, {f}},
{{a, b}, {a, c}, {a, d}, {a, e}, {a, f}, {b, c}, {b, d},
{b, e}, {b, f}, {c, d}, {c, e}, {c, f}, {d, e}, {d, f},
{e, f}},
{{a, b, c}, {a, b, d}, {a, b, e}, {a, b, f}, {a, c, d},
{a, c, e}, {a, c, f}, {a, d, e}, {a, d, f}, {a, e, f},
{b, c, d}, {b, c, e}, {b, c, f}, {b, d, e}, {b, d, f},
{b, e, f}, {c, d, e}, {c, d, f}, {c, e, f}, {d, e, f}},
{{a, b, c, d, e, f}}}

□

En primer lugar vamos a calcular todos los subgrupos propios de G de n elementos para $1 < n < |G|$, para ello calcularemos todos los subconjuntos de n elementos de G e iremos comprobando si son subgrupos con la función 3.1.

| FUNCIÓN | COMENTARIOS |
|--|--|
| G=GRUPO; operacion=TABLA DE OPERACIONES DE G; | Introducimos el grupo “G”. |
| op[x_,y_]:=... | Introducimos la función 2.1. |
| ElementoNeutro=... | Calculamos el elemento neutro con el programa 2.4. |
| simetrico[x_]:=... | Definimos la función 2.7. |

| | |
|---|---|
| SUBGRUPO[H_]:=... | Definimos la función 3.1. |
| <<Combinatoria` | Introducimos el paquete de funciones. |
| SUBGRUPOS[N_]:=Module[{Subconjuntos,Subgrupos,CONTADORi}, | El argumento es el orden de los subgrupos que pretendemos calcular. |
| Subconjuntos=KSubsets[G,n]; | Calculamos todos los subconjuntos de "G" de "n" elementos. |
| Print["Se comprobarán ",Length[Subconjuntos], " subconjuntos de ", n," elementos"]; | Se informa sobre el número de subconjuntos que se van a comprobar. |
| Subgrupos={}; | |
| Do[If[SUBGRUPO[Subconjuntos[[CONTADORi]], AppendTo[Subgrupos, Subconjuntos[[CONTADORi]]]] ,{CONTADORi,1,Length[Subconjuntos]}]; | Se comprueban si son subgrupos todos los subconjuntos de "n" elementos. |
| Subgrupos] | Se muestran los resultados. |

Función 3.7. Subgrupos de "n" elementos¹⁰.

Aplicamos la función anterior a cada divisor del cardinal de G , y así obtendremos todos los subgrupos de G .

| PROGRAMA | COMENTARIOS |
|---|---|
| SUBGRUPOS[N_]:=... | Definimos la función 3.7. |
| total=0; cardinal=Divisors[Length[G]]; | Suponemos de partida que no hay ninguno y calculamos los divisores de $ G $. |
| Do[subg=SUBGRUPOS[cardinal[[CONTADORi]]]; total=total+Length[subg]; Print["Subgrupos de orden ", cardinal[[CONTADORi]],": ",subg] ,{CONTADORi,1,Length[cardinal]}]; | Calculamos los subgrupos de G con tantos elementos como indica el divisor i -ésimo. |
| Print["Número total de subgrupos: ",total] | Se muestran el número de subgrupos. |

Programa 3.8. Cálculo de todos los subgrupos.

Ejemplo 3.6. Determinar todos los subgrupos de $G = \{1, 2, 3, 4, 5, 6\}$ con la operación dada por la tabla que se define a continuación:

```
In]:= G=Table[CONTADORi,{CONTADORi,6}];  
In]:= operacion={{1,2,3,4,5,6},{2,1,4,3,6,5},{3,5,1,6,2,4},  
{4,6,2,5,1,3},{5,3,6,1,4,2},{6,4,5,2,3,1}};
```

Definimos todas las funciones que necesitamos, 2.1., 2.4., 2.7., 3.1 y 3.7.:

```
In]:= op[x_,y_]:=operacion[[Position[G,x][[1]],
```

¹⁰ Como en otras ocasiones, podríamos integrar todas las funciones y definiciones previas dentro del cuerpo de SUBGRUPOS[], resultando un poco más cómoda de usar, dejamos esta posibilidad para el lector interesado, porque esto supone cierta pérdida de eficacia y más adelante, se analizará y realizará un estudio más profundo de este tema, por lo que no sería lo más apropiado.

```

Position[G,y][[1]]][[1]][[1]];

In[]:= ElementoNeutro="No existe";
          :
          :

In[]:= simetrico[x_]:=Module[{simetrico,CONTADORi},
          :
          :

In[]:= SUBGRUPO[H_]:=Module[{subgrupo,
          :
          :

In[]:= <<“Combinatorica`”
In[]:= SUBGRUPOS[n_]:=Module[
          :
          :

```

Ahora usamos el programa 3.8.

```

In[]:= total=0;
cardinal=Divisors[Length[G]];
          :
          :

Out[]= Se comprobarán 6 subconjuntos de 1 elementos
           Subgrupos de orden 1: {{1}}
           Se comprobarán 15 subconjuntos de 2 elementos
           Subgrupos de orden 2: {{1,2},{1,3},{1,6}}
           Se comprobarán 20 subconjuntos de 3 elementos
           Subgrupos de orden 3: {{1,4,5}}
           Se comprobarán 1 subconjuntos de 6 elementos
           Subgrupos de orden 6: {{1,2,3,4,5,6}}
           Número total de subgrupos: 6

```

□

Ejemplo 3.7. Determinar todos los subgrupos de 1, 2, 3, 4 y 24 elementos de $G = \{1, 2, \dots, 24\}$ con la operación dada por la tabla que se define a continuación:

```

In[]:= G=Table[i,{i,24}];

In[]:= operacion={
{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24},
{2,1,4,3,6,5,8,7,10,9,12,11,14,13,16,15,18,17,20,19,22,21,24,23},
{3,5,1,6,2,4,9,11,7,12,8,10,15,17,13,18,14,16,21,23,19,24,20,22},
{4,6,2,5,1,3,10,12,8,11,7,9,16,18,14,17,13,15,22,24,20,23,19,21},
{5,3,6,1,4,2,11,9,12,7,10,8,17,15,18,13,16,14,23,21,24,19,22,20},
{6,4,5,2,3,1,12,10,11,8,9,7,18,16,17,14,15,13,24,22,23,20,21,19},

```

```
{7,8,13,14,19,20,1,2,15,16,21,22,3,4,9,10,23,24,5,6,11,12,17,18},
{8,7,14,13,20,19,2,1,16,15,22,21,4,3,10,9,24,23,6,5,12,11,18,17},
{9,11,15,17,21,23,3,5,13,18,19,24,1,6,7,12,20,22,2,4,8,10,14,16},
{10,12,16,18,22,24,4,6,14,17,20,23,2,5,8,11,19,21,1,3,7,9,13,15},
{11,9,17,15,23,21,5,3,18,13,24,19,6,1,12,7,22,20,4,2,10,8,16,14},
{12,10,18,16,24,22,6,4,17,14,23,20,5,2,11,8,21,19,3,1,9,7,15,13},
{13,19,7,20,8,14,15,21,1,22,2,16,9,23,3,24,4,10,11,17,5,18,6,12},
{14,20,8,19,7,13,16,22,2,21,1,15,10,24,4,23,3,9,12,18,6,17,5,11},
{15,21,9,23,11,17,13,19,3,24,5,18,7,20,1,22,6,12,8,14,2,16,4,10},
{16,22,10,24,12,18,14,20,4,23,6,17,8,19,2,21,5,11,7,13,1,15,3,9},
{17,23,11,21,9,15,18,24,5,19,3,13,12,22,6,20,1,7,10,16,4,14,2,8},
{18,24,12,22,10,16,17,23,6,20,4,14,11,21,5,19,2,8,9,15,3,13,1,7},
{19,13,20,7,14,8,21,15,22,1,16,2,23,9,24,3,10,4,17,11,18,5,12,6},
{20,14,19,8,13,7,22,16,21,2,15,1,24,10,23,4,9,3,18,12,17,6,11,5},
{21,15,23,9,17,11,19,13,24,3,18,5,20,7,22,1,12,6,14,8,16,2,10,4},
{22,16,24,10,18,12,20,14,23,4,17,6,19,8,21,2,11,5,13,7,15,1,9,3},
{23,17,21,11,15,9,24,18,19,5,13,3,22,12,20,6,7,1,16,10,14,4,8,2},
{24,18,22,12,16,10,23,17,20,6,14,4,21,11,19,5,8,2,15,9,13,3,7,1};
```

Definimos todas las funciones 2.1., 2.4., 2.7., 3.1. y 3.7.:

In[]:= **op[x_,y_]:=operacion[[Position[G,x][[1]],**
Position[G,y][[1]]][[1]]][[1]];

In[]:= **ElementoNeutro="No existe";**

⋮ ⋮

In[]:= **simetrico[x_]:=Module[{simetrico},**

⋮ ⋮

In[]:= **SUBGRUPO[H_]:=Module[{subgrupo,**

⋮ ⋮

In[]:= **<<“Combinatorica`”**

In[]:= **SUBGRUPOSN[n_]:=Module[**

⋮ ⋮

No tenemos problemas para calcular los subgrupos de 1, 2, 3 y 24 elementos:

In[]:= **SUBGRUPOSN[1]**
SUBGRUPOSN[2]
SUBGRUPOSN[3]
SUBGRUPOSN[24]

Out[]= Se comprobarán 24 subconjuntos de 1 elementos
 $\{\{1\}\}$

Out[]= Se comprobarán 276 subconjuntos de 2 elementos
 $\{\{1,2\}, \{1,3\}, \{1,6\}, \{1,7\}, \{1,8\}, \{1,15\}, \{1,17\}, \{1,22\}, \{1,24\}\}$

Out[]= Se comprobarán 2024 subconjuntos de 3 elementos
 $\{\{1,4,5\}, \{1,9,13\}, \{1,12,20\}, \{1,16,21\}\}$

Out[]= Se comprobarán 1 subconjunto de 24 elementos
 $\{\{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24\}\}$

Al calcular los subgrupos de 4 elementos, observamos que el cálculo se dilata unos segundos:

In[]:= **SUBGRUPOSN[4]**

Out[]= Se comprobarán 10626 subconjuntos de 4 elementos
 $\{\{1,2,7,8\}, \{1,3,22,24\}, \{1,6,15,17\}, \{1,8,17,24\}, \{1,8,18,23\}, \{1,10,17,19\}, \{1,11,14,24\}\}$

□

5. SUBGRUPOS GENERADOS

Dado un subconjunto A de un grupo $(G, *)$ definiremos el subgrupo generado por A como el menor de los subgrupos de G que contiene a A , lo denotaremos por $\langle A \rangle$.

Podemos escribir programas que determinen los subgrupos generados por un subconjunto de un grupo finito usando 3.7. (ejercicio 3.18.). Sin embargo este procedimiento sería poco efectivo, pues es necesario calcular todos los subgrupos.

5.1. SUBGRUPO GENERADO POR UN ELEMENTO: EL ORDEN DE UN ELEMENTO

Dado un elemento $x \in G$, denotaremos por $\langle x \rangle$ al subgrupo de G generado por el conjunto $\{x\}$. Llamaremos orden de x al número de elementos de $\langle x \rangle$ si éste es finito. Para G un grupo finito y $x \in G$, existe n un entero positivo tal que $x^n = e$, el elemento neutro de G , y en tal caso,

$$\langle x \rangle = \{e, x, x^2, \dots, x^{n-1}\}.$$

Para calcularlo bastaría con averiguar el orden del elemento y sus potencias del mismo:

| PROGRAMA | COMENTARIOS |
|-----------------|----------------------------|
| G=GRUPO; | Introducimos el grupo “G”. |

| | |
|--|--|
| operacion=TABLA DE OPERACIONES DE G; | |
| op[x_,y_]:=... | Introducimos la función 2.1. |
| ElementoNeutro=... | Calculamos el elemento neutro con el programa 2.4. |
| x=UN ELEMENTO DEL GRUPO; | Introducimos el elemento “x”. |
| Gx={x};atemp=x; While[atemp != ElementoNeutro, atemp=op[atemp,x]; AppendTo[Gx,atemp];]; | Se calculan las sucesivas potencias de “x” hasta obtener el elemento neutro. |
| Print["Orden de ",x,"":Length[Gx]] Print["<",x,"> = ",Gx] | Se muestran el número de subgrupos. |

Programa 3.9. Orden de un elemento.

Ejemplo 3.8. Consideramos el grupo de 24 elementos del ejemplo 3.7.:

```
In]:=          G=Table[i,{i,24}];  
  
In]:=          operación={{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,  
19,20,21,22,23,24},  
          :           :  
          :           :
```

Calculamos $<23>$ y su orden. Necesitaremos previamente determinar el elemento neutro con el programa 2.4. y definir la función 2.1.

```
In]:=          op[x_,y_]:=operacion[[Position[G,x][[1]],  
Position[G,y][[1]]][[1]][[1]]];  
  
In]:=          ElementoNeutro="No existe";  
          :           :  
          :           :
```

Out]= 1

Ahora utilizamos 3.9.

```
In]:=          x=23;  
          :           :  
          :           :  
Out]=      Orden de 23 : 4  
          < 23 > = {23, 8, 18, 1}
```

□

Podemos ayudarnos para los cálculos anteriores del siguiente teorema cuya demostración también podemos encontrarla en la introducción del capítulo 5 de [15]:

Teorema 3.3. Teorema de Cauchy para grupos finitos. Sea $(G, *)$ un grupo finito con n elementos. Si p es primo y divide a n entonces existen en G elementos de orden p (y por tanto subgrupos de p elementos).

□

Obsérvese que este teorema nos proporciona un recíproco del teorema de Lagrange, aunque nótese que sólo para divisores primos. Para grupos finitos abelianos, el recíproco del teorema de Lagrange, sí es cierto para cualquier divisor del orden del grupo. Usando 3.16. podremos encontrar fácilmente los elementos cuya existencia asegura el teorema.

Ejemplo 3.9. Consideramos el grupo de 24 elementos del ejemplo 3.7.:

```
In]:= G=Table[i,{i,24}];  
In]:= operación={{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,  
19,20,21,22,23,24},  
⋮  
⋮  
In]:= op[x_,y_]:=operación[[Position[G,x][[1]],  
Position[G,y][[1]]][[1]][[1]]];  
⋮  
⋮  
In]:= ElementoNeutro="No existe";  
⋮  
⋮  
Out]= 1
```

Es evidente que $|G| = 24$, y los divisores primos de 24 son 2 y 3. Calculamos el orden de todos los elementos de G usando 3.9., para ello utilizaremos un bucle que recorrerá todos los elementos de G , y determinará el subgrupo generado por cada elemento y su orden:

```
In]:= Do[VARx=CONTADORi;  
Gx={VARx};  
VARa=VARx;  
While[VARa != ElementoNeutro,  
VARa=op[VARa,VARx];  
AppendTo[Gx,VARa];];  
Print["Orden de ",VARx," : ",Length[Gx]];  
Print["<",VARx,"> = ",Gx];  
,{CONTADORi,1,24}]  
  
Out]=  
Orden de 1 : 1      Orden de 9 : 3      Orden de 17 : 2  
< 1 > = {1}      < 9 > = {9,13,1}      < 17 > = {17,1}  
Orden de 2 : 2      Orden de 10 : 4      Orden de 18 : 4
```

| | | |
|-----------------------------------|--|--|
| $\langle 2 \rangle = \{2, 1\}$ | $\langle 10 \rangle = \{10, 17, 19, 1\}$ | $\langle 18 \rangle = \{18, 8, 23, 1\}$ |
| Orden de 3 : 2 | Orden de 11 : 4 | Orden de 19 : 4 |
| $\langle 3 \rangle = \{3, 1\}$ | $\langle 11 \rangle = \{11, 24, 14, 1\}$ | $\langle 19 \rangle = \{19, 17, 10, 1\}$ |
| Orden de 4 : 3 | Orden de 12 : 3 | Orden de 20 : 3 |
| $\langle 4 \rangle = \{4, 5, 1\}$ | $\langle 12 \rangle = \{12, 20, 1\}$ | $\langle 20 \rangle = \{20, 12, 1\}$ |
| Orden de 5 : 3 | Orden de 13 : 3 | Orden de 21 : 3 |
| $\langle 5 \rangle = \{5, 4, 1\}$ | $\langle 13 \rangle = \{13, 9, 1\}$ | $\langle 21 \rangle = \{21, 16, 1\}$ |
| Orden de 6 : 2 | Orden de 14 : 4 | Orden de 22 : 2 |
| $\langle 6 \rangle = \{6, 1\}$ | $\langle 14 \rangle = \{14, 24, 11, 1\}$ | $\langle 22 \rangle = \{22, 1\}$ |
| Orden de 7 : 2 | Orden de 15 : 2 | Orden de 23 : 4 |
| $\langle 7 \rangle = \{7, 1\}$ | $\langle 15 \rangle = \{15, 1\}$ | $\langle 23 \rangle = \{23, 8, 18, 1\}$ |
| Orden de 8 : 2 | Orden de 16 : 3 | Orden de 24 : 2 |
| $\langle 8 \rangle = \{8, 1\}$ | $\langle 16 \rangle = \{16, 21, 1\}$ | $\langle 24 \rangle = \{24, 1\}$ |

Y en efecto comprobamos como existen elementos de orden 2 y 3, como nos adelantó el teorema:

- $|\langle 2 \rangle| = |\langle 3 \rangle| = |\langle 6 \rangle| = |\langle 7 \rangle| = |\langle 8 \rangle| = |\langle 15 \rangle| = |\langle 17 \rangle| = |\langle 22 \rangle| = |\langle 24 \rangle| = 2$.
- $|\langle 4 \rangle| = |\langle 5 \rangle| = |\langle 9 \rangle| = |\langle 12 \rangle| = |\langle 13 \rangle| = |\langle 16 \rangle| = |\langle 20 \rangle| = |\langle 21 \rangle| = 3$.

Además también nos ha proporcionado los elementos de orden 1 y 4, así como también se ha comprobado que este grupo no tiene elementos de orden superior a 4:

- $|\langle 1 \rangle| = 1$.
- $|\langle 10 \rangle| = |\langle 11 \rangle| = |\langle 14 \rangle| = |\langle 18 \rangle| = |\langle 19 \rangle| = |\langle 23 \rangle| = 4$.

□

5.2. SUBGRUPO GENERADO POR UN SUBCONJUNTO CUALQUIERA

Para calcular el subgrupo generado por un subconjunto podemos escribir una rutina que partirá de las ideas empleadas en la función 3.6.

| FUNCIÓN | COMENTARIOS |
|---|--|
| G=GRUPO; operacion=TABLA DE OPERACIONES DE G; | Introducimos el grupo “G”. |
| ElementoNeutro=... | Calculamos el elemento neutro con el programa 2.4. |
| simetrico[x_]:=... | Definimos la función 2.7. |
| GENERADO[A_]:=Module[{CONTADORx,CONTADORh,conjtemp,op}, op[x_,y_]:=operacion[[Position[G,x][[1]], Position[G,y][[1]]][[1]]][[1]]; conjunto=A; | Función cuyo argumento es un subconjunto de G . |
| conjunto=Union[{ElementoNeutro},A]; Do[conjunto=Union[conjunto, {simetrico[A][[CONTADORi]]}]; ,{CONTADORi,Length[A]}]; | Se completa el conjunto con el neutro y los simétricos de cada elemento. Esta parte no es indispensable y podemos quitarla aunque optimizará la función. |
| conjtemp={}; | |
| While[Length[conjtemp]!=Length[conjunto], | Se calculan todos los sucesivos resultados |

| | |
|---|---|
| <pre> conjtemp=conjunto; Do[Do[conjunto=Union[conjunto, {op[conjunto[[CONTADORx]], conjunto[[CONTADORh]]}], {op[conjunto[[CONTADORh]], conjunto[[CONTADORx]]}]]; ,{CONTADORh,CONTADORx+1, Length[conjunto]}]; conjunto=Union[conjunto, {op[conjunto[[CONTADORx]], conjunto[[CONTADORx]]}]]; ,{CONTADORx,Length[conjunto]}];] conjunto </pre> | de operar elementos del conjunto hasta hacerlo cerrado para la operación. |
| | Se muestran el menor subgrupo generado por “A”. |

Función 3.10. Subgrupo generado por un subconjunto.

Como ya se ha hecho en otras ocasiones, cabe la posibilidad de integrar el cálculo del elemento neutro y la función simetrico[] dentro del cuerpo de la función 3.10., resultado más cómoda de usar, aunque menos eficaz si dicha función se utiliza de manera reiterada.

Ejemplo 3.10. Para el grupo del ejemplo 3.7. calcular los subgrupos generados por los siguientes subconjunto:

- $A = \{3, 12\}$.
- $B = \{17, 18\}$.
- $C = \{11, 14\}$.

Introducimos el grupo, calculamos el neutro y definimos 2.1 y 2.7.:

```

In]:= G=Table[i,{i,24}];

In]:= operacion={{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,
19,20,21,22,23,24},
               :           :           :           :
               :           :           :           :

In]:= ElementoNeutro="No existe";
               :           :           :           :
               :           :           :           :

Out]= 1

In]:= simetrico[x_]:=Module[{simetrico,CONTADORi},
               :           :           :           :
               :           :           :           :

```

Definimos la función 3.10. y calculamos lo que nos proponen:

| | |
|---------------|---|
| <i>In[]:=</i> | GENERADO[A_]:=Module[{CONTADORx, |
| ⋮ | ⋮ |

Calculamos los subgrupos:

- *In[]:=* **GENERADO[{3,12}]**
- Out[]=* $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,$
 $17, 18, 19, 20, 21, 22, 23, 24\}$
- *In[]:=* **GENERADO[{11,14}]**
- Out[]=* $\{1, 11, 14, 24\}$
- *In[]:=* **GENERADO[{17,18}]**
- Out[]=* $\{1, 2, 7, 8, 17, 18, 23, 24\}$

□

6. EFICACIA Y OPTIMIZACIÓN EN EL CÁLCULO DE SUBGRUPOS.

La eficacia de un algoritmo se mide por la cantidad de recursos que consume, principalmente en cuanto al tiempo de proceso necesario y la memoria que le hace falta. Por lo general este problema es uno de los más complejos de analizar y solucionar en el caso de tener algoritmos poco eficientes. Además este problema constituye una parte muy importante dentro de las Ciencias de la Computación, no pretendemos en ningún caso salirnos de los objetivos del libro, por el contrario, en el desarrollo de los algoritmos de este libro ha primado la visión didáctica, la claridad y la sencillez, sobre la eficiencia de los mismos, se ha preferido mostrar traslaciones fieles del concepto teórico al ordenador. En el caso del cálculo de los subgrupos de un grupo finito, nos encontramos con la posibilidad de analizar en parte este problema a modo de ejemplo, sin entrar en detalles propios de la programación y sin perder la visión didáctica, vamos a optimizar el cálculo de subgrupos razonando sobre cuestiones puramente matemáticas y basadas en las propiedades algebraicas del concepto.

El tema de la efectividad tiene ciertos inconvenientes relacionados directamente con el lenguaje que usemos para programar. En Mathematica, por lo general, las funciones que incorpora son muy eficaces, por el contrario las rutinas que programemos con herramientas habituales de programación no lo son tanto comparativamente hablando respecto a otros lenguajes. Por ejemplo, KSubsets[] es una función muy rápida pero exclusiva de Mathematica, si la evitamos y la programamos (véase ejercicio 3.16.),

observaremos como el tiempo empleado por Mathematica en hacer los mismos cálculos es mucho mayor (incluso si analizamos y evitamos aquellos métodos que en Mathematica son poco eficaces), si bien esto no tendría que ser así en otros lenguajes o plataformas más adecuadas. Es por eso que en esta sección intentaremos evitar el uso de las funciones particulares de Mathematica y en particular de KSubsets[], con el fin de poder trasladar los métodos aquí expuestos a otras plataformas. Las funciones 3.13. (SUBGRUPOSN3[]), 3.14. (SUBGRUPOSN4[]) y 3.15. (SUBGRUPOSN5[]) que se construyen en este epígrafe, además de ser las más efectivas, no utilizan KSubsets[], lo cual permitiría trasladarlas fácilmente y eficazmente a otros lenguajes, donde los resultados obtenidos serían incluso mejores que los conseguidos con Mathematica.

Para facilitar la comprensión de los métodos o algoritmos, se desarrollarán de manera constructiva, motivándose la necesidad ineludible de los mismos por los dilatados tiempos de cálculo que se van obteniendo al aplicarlos, paralelamente, sobre un ejemplo particular y obligándonos en cada caso a la mejora del algoritmo en términos de su eficiencia.

Para medir la eficacia disponemos de dos funciones:

- a) Para medir el tiempo de proceso:

Timing[Expresion]

Función que devuelve una lista formada por el tiempo en segundos usado para realizar el cálculo junto con la salida correspondiente. El resultado lógicamente dependerá de la capacidad de proceso del ordenador y del número de tareas que en ese momento esté realizando, a modo orientativo incluimos las salidas que obtenemos en un ordenador en particular, estas salidas serán diferentes en máquinas distintas, aunque proporcionalmente¹¹ la diferencia del tiempo de proceso empleado en el cálculo será parecida.

- b) Para controlar la cantidad de memoria que en cada momento usa Mathematica , usaremos:

MemoryInUse[]

En el cálculo de subgrupos, nos centraremos en medir los tiempos necesarios para el cálculo, y no entraremos en detalle en la comprobación de la eficacia en términos de memoria necesaria. En otros algoritmos será imprescindible observar la memoria usada (véase por ejemplo, el cálculo de todos los grafos de un mismo número de vértices en el capítulo 5).

Comenzamos midiendo el tiempo necesario para realizar el cálculo de los subgrupos de 4 elementos del grupo del ejemplo 3.7.:

¹¹ Según el modelo y marca del ordenador y en particular del procesador, la cantidad de memoria instalada, la velocidad de ésta,...; los valores obtenidos pueden ser muy distintos, incluso algunos algoritmos pueden mostrarse menos eficaces en ordenadores aparentemente más rápidos por alguna de estas circunstancias.

In[]:= **SUBGRUPOSN[4]**

Out[] = Se comprobarán 10626 subconjuntos de 4 elementos
 $\{\{1, 2, 7, 8\}, \{1, 3, 22, 24\}, \{1, 6, 15, 17\}, \{1, 8, 17, 24\},$
 $\{1, 8, 18, 23\}, \{1, 10, 17, 19\}, \{1, 11, 14, 24\}\}$

In[]:= **Timing[SUBGRUPOSN[4]][[1]]**

Out[] = 3.859 Second

Intentaremos agilizar lo máximo posible estos cálculos, para ello aprovecharemos las propiedades de las definiciones de grupo y subgrupo antes expuestas, aunque existen otras propiedades o teoremas de estructura que podrían ser útiles para estos fines, no los usaremos porque no son objeto de estudio de este libro, limitándonos exclusivamente a los conceptos expuestos en el capítulo.

Podríamos acelerar este último cálculo y hacerlo más efectivo no considerando todos los subconjuntos. Por ejemplo, en vez de testear todos los subconjuntos de 4 elementos, podemos probar sólo con aquellos de 4 elementos que contengan al neutro, ya que es condición necesaria para ser subgrupo. Usamos la misma función 3.7., pero modificamos la definición de “Subconjuntos”:

| FUNCIÓN | COMENTARIOS |
|--|---|
| G=GRUPO; operacion=TABLA DE OPERACIONES DE G; | Introducimos el grupo “G”. |
| op[x_,y_]:=... | Introducimos la función 2.1. |
| ElementoNeutro=... | Calculamos el elemento neutro con el programa 2.4. |
| simetrico[x_]:=... | Definimos la función 2.7. |
| SUBGRUPO[H_]:=... | Definimos la función 3.1. |
| <<“Combinatoria” | Introducimos el paquete de funciones. |
| SUBGRUPOSN2[n_]:=Module[$\{\text{Subconjuntos}, \text{Subgrupos}, \text{CONTADORi}\},$ | Función cuyo argumento es el orden de los subgrupos que pretendemos calcular. |
| Subconjuntos=KSubsets[Complement[G,{ElementoNeutro}],n-1]; Do[Subconjuntos[[CONTADORi]]= Union[{ElementoNeutro}, Subconjuntos[[CONTADORi]]] $,\{\text{CONTADORi}, \text{Length}[\text{Subconjuntos}]\};$ | Se calculan los subconjuntos de “n” elementos que contienen al neutro. |
| Print["Se comprobarán ",Length[Subconjuntos], $" \text{subconjuntos de } ", n, " \text{ elementos}"];$ | Se informa sobre el número de subconjuntos que se van a comprobar. |
| Subgrupos={}; Do[If[SUBGRUPO[Subconjuntos[[CONTADORi]]], AppendTo[Subgrupos, Subconjuntos[[CONTADORi]]]] $,\{\text{CONTADORi}, 1, \text{Length}[\text{Subconjuntos}]\};$ | Se comprueban cuáles de los subconjuntos candidatos son subgrupos. |

| | |
|---|-----------------------------|
| Subgrupos | Se muestran los resultados. |
| Función 3.11. Subgrupos de "n" elementos. | |

In[]:= **SUBGRUPOSN2[4]**

Out[] = Se comprobarán 1771 subconjuntos de 4 elementos

$\{\{1, 2, 7, 8\}, \{1, 3, 22, 24\}, \{1, 6, 15, 17\}, \{1, 8, 17, 24\}, \{1, 8, 18, 23\}, \{1, 10, 17, 19\}, \{1, 11, 14, 24\}\}$

Medimos el tiempo de proceso,

In[]:= **Timing[SUBGRUPOSN2[4]][[1]]**

Out[] = 1.39 Second

La mejora es sustancial, pues sólo comprobamos 1771 subconjuntos frente a los 10626 de antes, de la misma forma podemos pensar en otras restricciones o propiedades que mejoren el rendimiento del programa.

Ahora nos planteamos el cálculo de todos los subgrupos de 6 y 8 elementos del grupo G del ejemplo 3.7. Obsérvese que el número de subconjuntos de 6 y 8 elementos es enorme:

In[]:= **Length[KSubsets[G,6]]**

Out[] = 134596

In[]:= **Length[KSubsets[G,8]]**

Out[] = 735471

Y si los calculamos con la función 3.7., el tiempo empleado es muy grande:

In[]:= **Timing[SUBGRUPOSN[6]]**

Out[] = Se comprobarán 134596 subconjuntos de 6 elementos

$\{52.063 \text{ Second}, \{\{1, 2, 3, 4, 5, 6\}, \{1, 2, 15, 16, 21, 22\}, \{1, 3, 7, 9, 13, 15\}, \{1, 6, 7, 12, 20, 22\}\}\}$

In[]:= **Timing[SUBGRUPOSN[8]]**

Out[] = Se comprobarán 735471 subconjuntos de 8 elementos

```
{320.515 Second,
{{1,2,7,8,17,18,23,24},{1,3,8,11,14,17,22,24},
{1,6,8,10,15,17,19,24}}}
```

Incluso si consideramos únicamente los subconjuntos de 6 y 8 elementos que contienen al neutro, el número de subconjuntos sigue siendo muy elevado:

In[]:= Length[KSubsets[Complement[G,{ElementoNeutro}],5]]

Out[] = 33649

In[]:= Length[KSubsets[Complement[G,{ElementoNeutro}],7]]

Out[] = 245157

Y el tiempo de proceso también:

In[]:= Timing[SUBGRUPOSN2[6]]

Out[] = Se comprobarán 33649 subconjuntos de 6 elementos

```
{27.703 Second,
{{1,2,3,4,5,6},{1,2,15,16,21,22},
{1,3,7,9,13,15},{1,6,7,12,20,22}}}
```

In[]:= Timing[SUBGRUPOSN2[8]]

*Out[] = Se comprobarán 245157 subconjuntos
de 8 elementos*

```
{215.297 Second,
{{1,2,7,8,17,18,23,24},{1,3,8,11,14,17,22,24},
{1,6,8,10,15,17,19,24}}}
```

Como el tiempo necesario para el cálculo sigue siendo excesivo, intentaremos hacerlo más eficaz. En un primer paso hemos reducido los subconjuntos a testear a aquellos que contienen al neutro, ahora vamos a pedirle adicionalmente que sean cerrados para simétricos, esto es, que si un elemento está, entonces su simétrico obligatoriamente también estará. De hecho lo que hacemos es construir conjuntos tales que verifiquen las dos propiedades: la de elemento neutro y la de elemento simétrico. Para ser subgrupo le faltaría ser cerrado para la operación, luego programamos una función que determine si un subconjunto es cerrado para la operación y una rutina que integre estas ideas, esto es, que construya subconjuntos que contengan al elemento neutro y sean cerrados para simétricos:

| FUNCIÓN | COMENTARIOS |
|--|---|
| G=GRUPO; operacion=TABLA DE OPERACIONES DE G; | Introducimos el grupo “G”. |
| op[x,y]:=... | Introducimos la función 2.1. |
| CERRADO[H_]:=Module[| La función tendrá como argumento un subconjunto de “G”. |

| | |
|---|---|
| {subgrupo,CONTADORi,CONTADORj}, | |
| <pre> subgrupo=True; CONTADORi=1; While[subgrupo && CONTADORi<=Length[H], CONTADORj=1; While[subgrupo && CONTADORj<=Length[H], If[Intersection[{op[H][[CONTADORi]], H[[CONTADORj]]}],H]=={} , subgrupo=False;]; CONTADORj++;]; CONTADORi++;]; </pre> | Comprobará que el subconjunto “H” es cerrado para la operación. |
| subgrupo | Salida de resultados. |

Función 3.12. Subconjuntos cerrados.

| FUNCIÓN | COMENTARIOS |
|---|---|
| G=GRUPO; operacion=TABLA DE OPERACIONES DE G; | Introducimos el grupo “G”, su operación interna, su elemento neutro, la función simetrico[] y la función CERRADO[]. |
| op[x_,y_]:=... | Introducimos la función 2.1. |
| ElementoNeutro=... | Calculamos el elemento neutro con el programa 2.4. |
| simetrico[x_]:=... | Definimos la función 2.7. |
| CERRADO[H_]:=... | Definimos la función 3.12. |
| SUBGRUPOSN3[n_]:=Module[{CONTADORi, CONTADORj,CONTADORk,Subconjuntos, Subconjuntostemp2,Gsin,sime,simetab}, | Función cuyo argumento es el orden de los subgrupos que pretendemos calcular. |
| If[Mod[Length[G],n]==0, | Hacemos cálculos sólo para los divisores del orden de “G” (teorema de Lagrange). |
| If[n==1 n==Length[G], If[n==1,Subgrupos={{ElementoNeutro}}]; If[n==Length[G],Subgrupos={G}]; | Si el orden 1 o el de “G” los cálculos son triviales. |
| , | |
| Subconjuntos={{ElementoNeutro}}; | Todos los subconjuntos contendrán como primer elemento al elemento neutro. |
| simetab=Table[simetrico[G[[CONTADORj]]],{CONTADORj,Length[G]}]; | Calculamos todos los simétricos y los almacenamos en la lista “simetab” para ganar en eficacia. |
| Do[Subconjuntostemp2={}; | Bucle que dará “(n – 1)” vueltas, en cada una se añadirán elementos nuevos a los subconjuntos candidatos a subgrupos. |
| Do[| Bucle que recorre todos los subconjuntos creados hasta el momento. |
| If[Length[Subconjuntos[[CONTADORj]]]<n, | Comprobamos que el subconjunto tenga menos de “n” elementos |
| Gsin=Complement[G, Subconjuntos[[CONTADORj]]]; Do[sime=simetab[[Position[G, | Si tiene menos de “n” elementos añadimos nuevos elementos, por cada nuevo elemento añadimos también su simétrico. |

| | |
|---|--|
| <pre> Gsin[[CONTADORi]] [1] [[1]]; If[Length[Subconjuntos[[CONTADORj]]] ==n-1 , If[sime==Gsin[[CONTADORi]], Subconjuntostemp2=Union[Subconjuntostemp2, {Union[Subconjuntos[[CONTADORj]], {Gsin[[CONTADORi]]}]}]; , Subconjuntostemp2=Union[Subconjuntostemp2, {Union[Subconjuntos[[CONTADORj]], {Gsin[[CONTADORi]]},{sime}]}];]; ,{CONTADORi,1,Length[Gsin]}]; </pre> | |
| <pre> , Subconjuntostemp2=Union[Subconjuntostemp2, {Subconjuntos[[CONTADORj]]}];]; ,{CONTADORj,Length[Subconjuntos]}]; </pre> | |
| <pre> Subconjuntos=Subconjuntostemp2; ,{CONTADORk,1,n-1}]; Print["Se comprobarán ",Length[Subconjuntos], " subconjuntos de ",n," elementos"]; Subgrupos={}; Do[</pre> | |
| <pre> If[CERRADO[Subconjuntos[[CONTADORi]]], AppendTo[Subgrupos, Subconjuntos[[CONTADORi]]]] ,{CONTADORi,1,Length[Subconjuntos]}];]; </pre> | Se comprueban cuáles de los subconjuntos candidatos son subgrupos. |
| <pre> , Print["No es divisor del orden del grupo."]; Subgrupos={};]; </pre> | |
| <pre> Subgrupos]</pre> | Se muestran los resultados. |

Función 3.13. Subgrupos de "n" elementos.

Como ya hemos comentado, esta última función construye todos los subconjuntos de G , que contienen al neutro y son cerrados para simétricos. Para ello parte del neutro y va añadiendo nuevos elementos, pero por cada elemento nuevo se añade también su simétrico (por tanto, verificará la propiedad de elemento neutro y de elemento simétrico). Finalmente no es necesario testear los subconjuntos candidatos a subgrupos con la función SUBGRUPO[] (3.1.), bastará con comprobar qué subconjuntos son cerrados para la operación, para ello se usa la función 3.12. (CERRADO[]).

La probamos para $n = 6$ y $n = 8$, medimos el tiempo de cálculo:

In[]:= **Timing[SUBGRUPOSN[6]]**

Out[] = Se comprobarán 903 subconjuntos de 6 elementos
 $\{3.625 \text{ Second}, \{\{1, 2, 3, 4, 5, 6\}, \{1, 2, 15, 16, 21, 22\}, \{1, 3, 7, 9, 13, 15\}, \{1, 6, 7, 12, 20, 22\}\}\}$

In[]:= **Timing[SUBGRUPOSN[8]]**

Out[] = Se comprobarán 2997 subconjuntos de 8 elementos
 $\{74.016 \text{ Second}, \{\{1, 2, 7, 8, 17, 18, 23, 24\}, \{1, 3, 8, 11, 14, 17, 22, 24\}, \{1, 6, 8, 10, 15, 17, 19, 24\}\}\}$

Podemos observar, que tanto el tiempo como el número de subconjuntos candidatos, se han reducido considerablemente, aunque ahora necesitamos una gran cantidad de proceso para la construcción de los subconjuntos. Nos falta determinar los subgrupos de 12 elementos, en este caso el número de subconjuntos es desorbitado:

In[]:= **Length[KSubsets[G,12]]**

Out[] = 2704156

Calculamos también el número de subconjuntos que contengan al neutro:

In[]:= **Length[KSubsets[Complement[G,{ElementoNeutro}],11]]**

Out[] = 1352078

Y éste sigue siendo muy elevado, si calculamos los subgrupos de 12 elementos con SUBGRUPOSN[] (3.7.) y SUBGRUPOSN2[] (3.11.), los tiempos empleados serán los siguientes:

In[]:= **Timing[SUBGRUPOSN[12]]**

Out[] = Se comprobarán 2704156 subconjuntos de 12 elementos

$\{1692.27 \text{ Second}, \{\{1, 4, 5, 8, 9, 12, 13, 16, 17, 20, 21, 24\}\}\}$

In[]:= **Timing[SUBGRUPOSN2[12]]**

Out[] = Se comprobarán 1352078 subconjuntos de 12 elementos

$\{1438.27 \text{ Second}, \{\{1, 4, 5, 8, 9, 12, 13, 16, 17, 20, 21, 24\}\}\}$

Aunque el número de candidatos a subgrupo que aparecen con `SUBGRUPOSN2[]` es la mitad que con `SUBGRUPOSN[]`, el tiempo empleado globalmente en su testeo no refleja esta proporción, esto se explica porque los candidatos que no consideramos en `SUBGRUPOSN2[]` (aquellos que no contienen al neutro) son fáciles de analizar, siendo para ellos el test de subgrupo más rápido. Si nos quedamos sólo con aquellos que también son cerrados para simétricos, el número de subconjuntos a testear se reduce extraordinariamente, pero el tiempo necesario para construir estos subconjuntos es incluso más elevado que el empleado por las funciones 3.7. y 3.11.:

```
In]:= Timing[SUBGRUPOSN3[12]]
```

```
Out]= Se comprobarán 8302 subconjuntos
de 12 elementos
```

```
{4292.3 Second,
{{1,4,5,8,9,12,13,16,17,20,21,24}}}
```

El tiempo empleado para el cálculo de los subconjuntos candidatos es muy grande, esto es debido a que el cálculo de los simétricos consume gran cantidad de recursos, es por ello, que aunque el número de subconjuntos candidatos que resultan es pequeño, la construcción es muy lenta. Por otro lado, con las funciones primeras el número de subconjuntos candidatos era excesivo. Para mejorar esta rutina, añadimos una nueva condición en la construcción que se basará en que el conjunto debe ser cerrado para la operación, con esta nueva condición, en realidad construimos directamente los subgrupos, pues construimos todos los subconjuntos que contienen al neutro, son cerrados para simétricos y para la operación. Recreamos esto último en otra función más, que llamaremos `SUBGRUPOSN4[]` (3.14.). Obsérvese que en esta función, por cada elemento añadido a un subconjunto candidato a subgrupo, también se amplía con todas las potencias del elemento, y al tratarse de un grupo finito, entonces existirá un $n \in \mathbb{N}$ tal que la potencia n -ésima sea el elemento neutro, por tanto la potencia $(n - 1)$ -ésima es el elemento simétrico, consiguiendo así que también sea cerrado para simétricos. Posteriormente y por cada elemento agregado se calculan todos los productos con el resto de elementos del subconjunto candidato y de nuevo también son añadidos, reiterándose el proceso:

| FUNCIÓN | COMENTARIOS |
|--|--|
| <code>G=GRUPO;</code> <code>operacion=TABLA DE OPERACIONES DE G;</code> | Introducimos el grupo “G”, su operación interna y su elemento neutro. |
| <code>op[x_,y_]:=...</code> | Introducimos la función 2.1. |
| <code>ElementoNeutro=...</code> | Calculamos el elemento neutro con el programa 2.4. |
| <code>SUBGRUPOSN4[n_]:=Module[{Subconjuntos,</code> <code>Subconjuntostemp2,CONTADORi,CONTADORj,</code> <code>CONTADORk,CONTADORh,Gsin,conjunto,longit,</code> <code>Nuevos,conjtemp,conjtemp2,CONTADORx,</code> <code>elemento},</code> | Función cuyo argumento es el orden de los subgrupos que pretendemos calcular. |
| <code>If[Mod[Length[G],n]==0,</code> | Hacemos cálculos sólo para los divisores del orden de “G” (teorema de Lagrange). |
| <code>If[n==1 n==Length[G],</code> <code>If[n==1,Subconjuntos={{ElementoNeutro}}];</code> | Si el orden 1 o el de “G” los cálculos son triviales. |

| | |
|--|---|
| If[n==Length[G],Subconjuntos={G}; , | |
| Subconjuntos={{ElementoNeutro}}; | Todos los subconjuntos contendrán como primer elemento al elemento neutro. |
| Do[Subconjuntostemp2={}; | Bucle que dará “(n – 1)” vueltas, en cada una se añadirán elementos nuevos a los subconjuntos candidatos a subgrupos. |
| Do[| Bucle que recorre todos los subconjuntos creados hasta el momento. |
| If[Length[Subconjuntos [CONTADORj]]]<n, | Comprobamos que el subconjunto tenga menos de “n” elementos. |
| Gsin=Complement[G, Subconjuntos [CONTADORj]]; Do[longit= Length[Subconjuntos [CONTADORj]]; conjunto=Union[Subconjuntos [CONTADORj]], {Gsin [CONTADORi]}]; | Si tiene menos de “n” elementos añadiremos elementos nuevos. |
| elemento=op[Gsin [CONTADORi]], Gsin [CONTADORi]]; While[!TrueQ[elemento!=ElementoNeutro], conjunto=Union[conjunto,{elemento}]; elemento=op[elemento, Gsin [CONTADORi]]];]; | Por cada elemento nuevo, se añaden también todas sus potencias, esto es, el subgrupo generado por el elemento. |
| conjtemp=Subconjuntos [CONTADORj]; While[longit!=Length[conjunto] && Length[conjunto]<n+1 , | Bucle que dará vueltas hasta que después de operar unos elementos con otros no resulte ninguno nuevo. |
| Nuevos=Complement[conjunto,conjtemp]; conjtemp=Complement[conjtemp, {ElementoNeutro}]; conjtemp2=conjunto; | |
| Do[If[Length[conjunto]<n+1, Do[If[Length[conjunto]<n+1, conjunto=Union[conjunto, {op[Nuevos [CONTADORx]], conjtemp [CONTADORh]}], {op[conjtemp [CONTADORh]], Nuevos [CONTADORx]}]];,Break[]];; ,{CONTADORh,1,longit-1}]; ,{CONTADORx,Length[Nuevos]}]; | Se operan todos los elementos nuevos con los antiguos (almacenados en “conjtemp”). |
| Do[If[Length[conjunto]<n+1, Do[If[Length[conjunto]<n+1, conjunto=Union[conjunto, {op[Nuevos [CONTADORx]], | Operamos todos los elementos nuevos entre ellos. |

| | |
|--|---|
| <pre> Nuevos [CONTADORh]]; {op[Nuevos [CONTADORh]], Nuevos [CONTADORx]]; ,Break[];}; ,{CONTADORh,CONTADORx+1, Length[Nuevos]}; ,Break[];}; ,{CONTADORx,Length[Nuevos]}; </pre> | |
| <pre> Nuevos=Complement[conjunto, conjtemp2]; Do[If[Length[conjunto]<n+1, elemento= op[Nuevos [CONTADORx]], Nuevos [CONTADORx]]; While[!TrueQ[elemento== ElementoNeutro] , conjunto=Union[conjunto, {elemento}]; elemento=op[elemento, Nuevos [CONTADORx]];]; ,Break[];}; ,{CONTADORx,Length[Nuevos}}; </pre> | Buscamos todos los elementos que han surgido al operar unos elementos con otros y calculamos todas sus potencias. |
| <pre> conjtemp=conjtemp2; longit=Length[conjtemp];]; </pre> | |
| <pre> If[Length[conjunto]<n+1, Subconjuntostemp2= Union[Subconjuntostemp2, {conjunto}];]; ,{CONTADORi,1,Length[Gsin]}; </pre> | |
| <pre> ,Subconjuntostemp2= Union[Subconjuntostemp2, {Subconjuntos [CONTADORj]}];]; </pre> | |
| <pre> ,{CONTADORj,Length[Subconjuntos]}; Subconjuntos=Subconjuntostemp2; ,{CONTADORk,1,n-1}];]; </pre> | |
| <pre> ,Print["No es un divisor del orden del grupo"]; Subconjuntos={};]; </pre> | |
| <pre> Subconjuntos </pre> | Se muestran los resultados. |

Función 3.14. Subgrupos de "n" elementos.

La efectividad en el cálculo de los subgrupos de 12 elementos de esta última función es comparativamente bastante buena:

```
In]:= Timing[SUBGRUPOSN4[12]]
```

```
Out]= {0.672 Second,
{{1, 4, 5, 8, 9, 12, 13, 16, 17, 20, 21, 24}}}
```

En resumen, hemos empleado 4 métodos distintos:

- Método 1. SUBGRUPOSN[] (3.7.), la función más burda de todas, calcula todos los subconjuntos de n elementos y determina cuáles de ellos son subgrupos con la función SUBGRUPO[] (3.1.).
- Método 2. SUBGRUPOSN2[] (3.11.), es igual que el método 1, pero sólo se queda con aquellos subconjuntos que contienen al neutro.
- Método 3. SUBGRUPOSN3[] (3.13.), nos quedamos sólo con aquellos subconjuntos que contienen al neutro y son cerrados para simétricos, finalmente usamos CERRADO[] (3.12.) para comprobar cuáles son cerrados para la operación y por tanto subgrupos.
- Método 4. SUBGRUPOSN4[] (3.14.), usando todo lo aprendido en 2 y 3, se hace una construcción de aquellos subconjuntos que son subgrupos y se integra todo en una nueva función.

Podemos obviar los cálculos de los subgrupos impropios de 1 y 24 elementos porque son evidentes, bastaría con un condicional que discriminara tales casos como se ha hecho en los métodos 3 y 4. El cálculo de los subgrupos de 2 y 3 elementos no presenta grandes problemas de proceso, y por cualquiera de los 4 métodos sería efectivo.

Con cada uno de ellos hemos obtenido resultados distintos en cuanto al número de subconjuntos¹² y tiempo empleado en el cálculo que reflejamos y resumimos en la siguiente tabla:

| Número de subconjuntos – Tiempos empleados | | | | | | | | | |
|--|----------------|-------------|----------------|-------------|--------------|-------------|-----------|----------|--|
| (Todos los tiempos están medidos en un mismo ordenador, estos variarán según la capacidad del ordenador en que se testeén) | | | | | | | | | |
| MÉTODO | 1 | | 2 | | 3 | | 4 | | |
| ELEM. ¹³ | Subconjuntos | Tiempo | Subconjuntos | Tiempo | Subconjuntos | Tiempo | Subgrupos | Tiempo | |
| 2 | 276 | 0.1 | 23 | <0.1 | 9 | <0.1 | 9 | <0.1 | |
| 3 | 2024 | 0.8 | 253 | 0.2 | 43 | <0.1 | 4 | <0.1 | |
| 4 | 10626 | 4 | 1771 | 1.5 | 147 | 0.5 | 7 | <0.1 | |
| 6 | 134596 | 52 | 33649 | 28 | 903 | 8 | 4 | 0.1 | |
| 8 | 735471 | 321 | 245157 | 215 | 2997 | 95 | 3 | 0.3 | |
| 12 | 2704156 | 1692 | 1352078 | 1438 | 8302 | 4292 | 1 | 0.6 | |
| TOTAL | 3587149 | 2070 | 1632931 | 1683 | 12401 | 4396 | 28 | 1 | |

Tabla 3.1. Efectividad en el cálculo de subgrupos impropios.

La siguiente tabla muestra todos los subgrupos obtenidos:

¹² El número de subconjuntos es importante desde el punto de vista de la cantidad de memoria que necesitamos, sobre todo para SUBGRUPOSN[] (3.7.) y SUBGRUPOSN2[] (3.11.) porque los calcula y guarda en memoria.

¹³ Sólo lo intentamos con los divisores de 24, pues por el teorema de Lagrange sabemos que es condición necesaria.

| ELEMENTOS | SUBGRUPOS |
|-----------|---|
| 1 | {1} |
| 2 | {1, 2}, {1, 3}, {1, 6}, {1, 7}, {1, 8}, {1, 15}, {1, 17}, {1, 22}, {1, 24} |
| 3 | {1, 4, 5}, {1, 9, 13}, {1, 12, 20}, {1, 16, 21} |
| 4 | {1, 2, 7, 8}, {1, 3, 22, 24}, {1, 6, 15, 17}, {1, 8, 17, 24}, {1, 8, 18, 23}, {1, 10, 17, 19}, {1, 11, 14, 24} |
| 6 | {1, 2, 3, 4, 5, 6}, {1, 2, 15, 16, 21, 22}, {1, 3, 7, 9, 13, 15}, {1, 3, 7, 9, 13, 15} |
| 8 | {1, 2, 7, 8, 17, 18, 23, 24}, {1, 3, 8, 11, 14, 17, 22, 24}, {1, 6, 8, 10, 15, 17, 19, 24} |
| 12 | {1, 4, 5, 8, 9, 12, 13, 16, 17, 20, 21, 24} |
| 24 | G |

Tabla 3.2. Subgrupos.

Ejemplo 3.11. Consideramos el conjunto $G = \{a, b, c, d, e, f, g, h, i, j, k, l\}$ y la operación interna definida por la siguiente tabla:

| | a | b | c | d | e | f | g | h | i | j | k | l |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | a | b | c | d | e | f | g | h | i | j | k | l |
| b | b | a | d | c | f | e | h | g | j | i | l | k |
| c | c | d | a | b | g | h | e | f | k | l | i | j |
| d | d | c | b | a | h | g | f | e | l | k | j | i |
| e | e | f | g | h | i | j | k | l | a | b | c | d |
| f | f | e | h | g | j | i | l | k | b | a | d | c |
| g | g | h | e | f | k | l | i | j | c | d | a | b |
| h | h | g | f | e | l | k | j | i | d | c | b | a |
| i | i | j | k | l | a | b | c | d | e | f | g | h |
| j | j | i | l | k | b | a | d | c | f | e | h | g |
| k | k | l | i | j | c | d | a | b | g | h | e | f |
| l | l | k | j | i | d | c | b | a | h | g | f | e |

Tabla 3.3.

Calculamos todos sus subgrupos por los cuatro métodos vistos.

Introducimos el conjunto G , la operación interna y la función 2.1.

```
In]:= G={a,b,c,d,e,f,g,h,i,j,k,l};
operacion={{a,b,c,d,e,f,g,h,i,j,k,l},{b,a,d,c,f,e,h,g,j,i,l,k},
{c,d,a,b,g,h,e,f,k,l,i,j},{d,c,b,a,h,g,f,e,l,k,j,i},
{e,f,g,h,i,j,k,l,a,b,c,d},{f,e,h,g,j,i,l,k,b,a,d,c},
{g,h,e,f,k,l,i,j,c,d,a,b},{h,g,f,e,l,k,j,i,d,c,b,a},
{i,j,k,l,a,b,c,d,e,f,g,h},{j,i,l,k,b,a,d,c,f,e,h,g},
{k,l,i,j,c,d,a,b,g,h,e,f},{l,k,j,i,d,c,b,a,h,g,f,e}};
```

$$\text{op}[x,y]:=\text{operacion}[[\text{Position}[G,x][[1]],$$

$$\text{Position}[G,y][[1]]][[1]][[1]]];$$

Utilizamos el programa 2.4. y determinamos el elemento neutro.

```
In]:= ElementoNeutro="No existe";
```

⋮ ⋮

Out[] = Elemento Neutro: a

Definimos las funciones 2.7., 3.1., 3.7., 3.11., 3.12., 3.13. y 3.14.

In[] := simetrico[x_]:=Module[{simetrico,CONTADORi},
 ⋮ ⋮]
In[] := SUBGRUPO[H_]:=Module[{subgrupo,CONTADORi},
 ⋮ ⋮]
In[] := <<Combinatorica`
In[] := SUBGRUPOSN[n_]:=Module[
 ⋮ ⋮]
In[] := SUBGRUPOSN2[n_]:=Module[
 ⋮ ⋮]
In[] := CERRADO[H_]:=Module[{subgrupo,CONTADORi},
 ⋮ ⋮]
In[] := SUBGRUPOSN3[n_]:=Module[{CONTADORi},
 ⋮ ⋮]
In[] := SUBGRUPOSN4[n_]:=Module[
 ⋮ ⋮]

Calculamos todos los subgrupos con los cuatro métodos y comparamos la eficacia de los mismos, para ello adaptamos 3.8.:

In[] := Timing[
total=0;cardinal=Divisors[Length[G]];
Do[subg=SUBGRUPOSN[cardinal][[CONTADORi]]];
 total=total+Length[subg];
 Print["Subgrupos de orden ",
 cardinal[[CONTADORi]],": ",subg
 ,{CONTADORi,1,Length[cardinal]}];
 Print["Número total de subgrupos: ",total]][[1]]

Out[] = Se comprobarán 12 subconjuntos de 1 elementos

```

Subgrupos de orden 1: {{a}}
Se comprobarán 66 subconjuntos de 2 elementos
Subgrupos de orden 2: {{a,b},{a,c},{a,d}}
Se comprobarán 220 subconjuntos de 3 elementos
Subgrupos de orden 3: {{a,e,i}}
Se comprobarán 495 subconjuntos de 4 elementos
Subgrupos de orden 4: {{a,b,c,d}}
Se comprobarán 924 subconjuntos de 6 elementos
Subgrupos de orden 6:
{{a,b,e,f,i,j},{a,c,e,g,i,k},{a,d,e,h,i,l}}
Se comprobarán 1 subconjuntos de 12 elementos
Subgrupos de orden 12:
{{a,b,c,d,e,f,g,h,i,j,k,l}}
Número total de subgrupos: 10
1.047 Second

```

In[]:=

```

Timing[
total=0;cardinal=Divisors[Length[G]];
Do[subg=SUBGRUPOSN2[[cardinal][[CONTADORi]]];
total=total+Length[subg];
Print["Subgrupos de orden ",
cardinal[[CONTADORi]],": ",subg
,{CONTADORi,1,Length[cardinal]}];
Print["Número total de subgrupos: ",total]][[1]]

```

Out[] =

```

Se comprobarán 1 subconjuntos de 1 elementos
Subgrupos de orden 1: {{a}}
Se comprobarán 11 subconjuntos de 2 elementos
Subgrupos de orden 2: {{a,b},{a,c},{a,d}}
Se comprobarán 55 subconjuntos de 3 elementos
Subgrupos de orden 3: {{a,e,i}}
Se comprobarán 165 subconjuntos de 4 elementos
Subgrupos de orden 4: {{a,b,c,d}}
Se comprobarán 462 subconjuntos de 6 elementos
Subgrupos de orden 6:
{{a,b,e,f,i,j},{a,c,e,g,i,k},{a,d,e,h,i,l}}
Se comprobarán 1 subconjuntos de 12 elementos
Subgrupos de orden 12:
{{a,b,c,d,e,f,g,h,i,j,k,l}}
Número total de subgrupos: 10
0.719 Second

```

In[]:=

```

Timing[
total=0;cardinal=Divisors[Length[G]];
Do[subg=SUBGRUPOSN3[[cardinal][[CONTADORi]]];
total=total+Length[subg];
Print["Subgrupos de orden ",
cardinal[[CONTADORi]],": ",subg
,{CONTADORi,1,Length[cardinal]}];
Print["Número total de subgrupos: ",total]][[1]]

```

Out[] =

```

Subgrupos de orden 1: {{a}}
Se comprobarán 3 subconjuntos de 2 elementos
Subgrupos de orden 2: {{a,b},{a,c},{a,d}}
Se comprobarán 7 subconjuntos de 3 elementos
Subgrupos de orden 3: {{a,e,i}}

```

Se comprobarán 13 subconjuntos de 4 elementos
 Subgrupos de orden 4: $\{\{a,b,c,d\}\}$
 Se comprobarán 22 subconjuntos de 6 elementos
 Subgrupos de orden 6:
 $\{\{a,b,e,f,i,j\}, \{a,c,e,g,i,k\}, \{a,d,e,h,i,l\}\}$
 Subgrupos de orden 12:
 $\{\{a,b,c,d,e,f,g,h,i,j,k,l\}\}$
 Número total de subgrupos: 10
 0.047 Second

In[]:=

```
Timing[
total=0;cardinal=Divisors[Length[G]];
Do[subg=SUBGRUPOSN4[cardinal][[CONTADORi]]];
total=total+Length[subg];
Print["Subgrupos de orden ",
cardinal[[CONTADORi]],": ",subg]
,{CONTADORi,1,Length[cardinal]}];
Print["Número total de subgrupos: ",total]][[1]]
```

Out[] =

Subgrupos de orden 1: $\{\{a\}\}$
 Subgrupos de orden 2: $\{\{a,b\}, \{a,c\}, \{a,d\}\}$
 Subgrupos de orden 3: $\{\{a,e,i\}\}$
 Subgrupos de orden 4: $\{\{a,b,c,d\}\}$
 Subgrupos de orden 6:
 $\{\{a,b,e,f,i,j\}, \{a,c,e,g,i,k\}, \{a,d,e,h,i,l\}\}$
 Subgrupos de orden 12:
 $\{\{a,b,c,d,e,f,g,h,i,j,k,l\}\}$
 Número total de subgrupos: 10
 0.031 Second

Finalmente, representamos el diagrama de orden (retículo de los subgrupos de G):

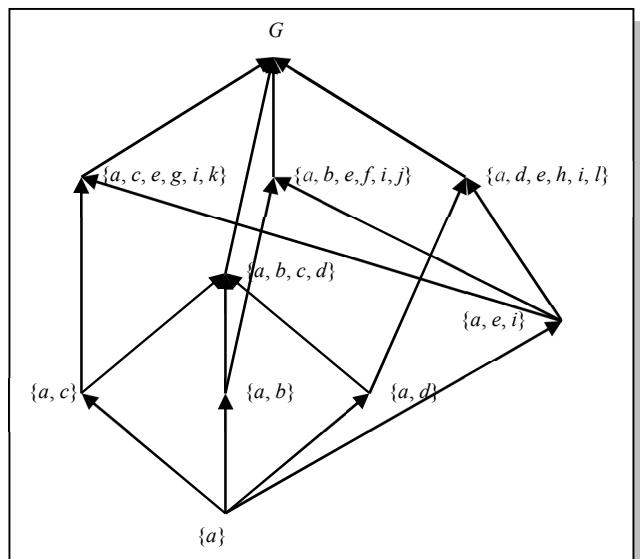


Ilustración 3.1.

□

Aún podemos mejorar más el tiempo de proceso, para ello vamos a usar la función GENERADO[] (3.10.) de este capítulo. Veámoslo a través del siguiente ejemplo:

Ejemplo 3.12. Queremos calcular todos los subgrupos de un grupo finito con suficientes elementos que nos permitan distinguir diferencias entre los lapsos de tiempo empleados en los cálculos, para ello cogeremos del capítulo siguiente a S_5 (grupo simétrico) un grupo de 120 elementos, podemos imaginar que su tabla de operaciones es enorme, de ahí la conveniencia y obligación de tomar prestado un grupo del capítulo siguiente ya que aunque podría definirse directamente por su conjunto y tabla de operaciones (tabla de 120 filas por 120 columnas), podemos hacerlo de forma algorítmica gracias a la función 4.6. Una vez calculado el conjunto y la tabla de operaciones podemos olvidarnos de cualquier otra consideración acerca del grupo (ya se estudiará en el capítulo 4) y limitarnos a usarlo como hasta ahora.

Calculamos la tabla de operaciones e introducimos el grupo simétrico identificando permutaciones con sus índices y usando las funciones 4.7, 4.9. y 4.11.:

```

In]:= sigma[k_,n_][m_]:=Permutations[Table[j,{j,n}]][[k,m]];
In]:= composicion2[k_,j_,n_]:=Module[{t,m,z},
Position[Permutations[Table[m,{m,n}]],
Table[sigma[k,n][sigma[j,n][m]],[m,1,n}}][[1]][[1]]]
]
In]:= cuentas2[n_]:=Module[{k,j,m},
operacion=Table[composicion2[j,k,n],{k,n!},{j,n!}];
TableForm[operacion,TableHeadings→{Table[m,{m,n!}],
Table[m,{m,n!}]},TableSpacing→{2,2}]
]
In]:= G=Table[i,{i,5!}];
cuentas2[5];
operacion=Table[composicion2[j,k,5],{k,5!},{j,5!}]
Out]= { {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,
19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,
34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,
49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,
64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,
79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,
94,95,96,97,98,99,100,101,102,103,104,105,106,
107,108,109,110,111,112,113,114,115,116,117,
118,119,120},
:
]:
In]:= op[x_,y_]:=operacion[[Position[G,x][[1]],
Position[G,y][[1]]][[1]][[1]]];

```

Utilizamos el programa 2.4. y determinamos el elemento neutro.

In[]:= **ElementoNeutro="No existe";**

⋮ ⋮

Out[] = Elemento Neutro: 1

Ahora calculamos todos los subgrupos, los posibles órdenes serán:

In[]:= **Divisors[5!]**

Out[] = {1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60, 120}

Vamos probando con todas las posibilidades la función 3.14. que es la más efectiva de las vistas hasta ahora. También determinamos los tiempos de cálculo:

In[]:= **Timing[SUBGRUPOSN4[2]]**

Out[] = {0.031 Second,
{ {1, 2}, {1, 3}, {1, 6}, {1, 7}, {1, 8}, {1, 15}, {1, 17},
{1, 22}, {1, 24}, {1, 25}, {1, 26}, {1, 27}, {1, 30},
{1, 55}, {1, 56}, {1, 61}, {1, 68}, {1, 81}, {1, 83},
{1, 87}, {1, 95}, {1, 106}, {1, 108}, {1, 112}, {1, 120}}}

In[]:= **Timing[SUBGRUPOSN4[3]]**

Out[] = {0.625 Second,
{ {1, 4, 5}, {1, 9, 13}, {1, 12, 20}, {1, 16, 21}, {1, 31, 49},
{1, 39, 75}, {1, 46, 100}, {1, 57, 79}, {1, 60, 104},
{1, 82, 105}}}

In[]:= **Timing[SUBGRUPOSN4[4]]**

Out[] = {0.984 Second,
{ {1, 2, 7, 8}, {1, 2, 25, 26}, {1, 2, 55, 56}, {1, 3, 22, 24},
{1, 3, 25, 27}, {1, 3, 106, 108}, {1, 6, 15, 17},
{1, 6, 25, 30}, {1, 6, 81, 83}, {1, 7, 81, 87},
{1, 7, 106, 112}, {1, 8, 17, 24}, {1, 8, 18, 23},
{1, 10, 17, 19}, {1, 11, 14, 24}, {1, 15, 55, 61},
{1, 15, 106, 120}, {1, 22, 55, 68}, {1, 22, 81, 95},
{1, 26, 95, 120}, {1, 26, 96, 119}, {1, 27, 61, 87},
{1, 27, 63, 85}, {1, 30, 68, 112}, {1, 30, 70, 110},
{1, 33, 61, 73}, {1, 36, 68, 98}, {1, 37, 51, 87},
{1, 40, 95, 99}, {1, 44, 54, 112}, {1, 45, 76, 120},
{1, 56, 83, 108}, {1, 56, 84, 107}, {1, 58, 83, 103},
{1, 59, 80, 108}}}

| | |
|----------------|---|
| <i>In[]:=</i> | Timing[SUBGRUPOSN4[5]] |
| <i>Out[] =</i> | <pre>{2.422 Second, {{1,34,65,91,97},{1,35,72,74,116}, {1,38,53,94,113},{1,42,69,86,101}, {1,43,52,90,117},{1,47,64,78,109}}}</pre> |
| <i>In[]:=</i> | Timing[SUBGRUPOSN4[6]] |
| <i>Out[] =</i> | <pre>{3.406 Second, {{1,2,3,4,5,6},{1,2,15,16,21,22}, {1,2,31,32,49,50},{1,2,81,82,105,106}, {1,3,7,9,13,15},{1,3,46,48,100,102}, {1,3,55,57,79,81},{1,4,5,25,28,29}, {1,4,5,26,27,30},{1,6,7,12,20,22}, {1,6,39,41,75,77},{1,6,55,60,104,106}, {1,7,25,31,49,55},{1,7,82,88,105,111}, {1,8,26,31,49,56},{1,8,82,87,105,112}, {1,9,13,106,114,118},{1,9,13,108,112,120}, {1,12,20,81,89,93},{1,12,20,83,87,95}, {1,15,25,39,75,81},{1,15,60,66,104,115}, {1,16,21,55,62,67},{1,16,21,56,61,68}, {1,17,30,39,75,83},{1,17,60,61,104,120}, {1,22,25,46,100,106},{1,22,57,71,79,92}, {1,24,27,46,100,108},{1,24,57,68,79,95}}}</pre> |
| <i>In[]:=</i> | Timing[SUBGRUPOSN4[8]] |
| <i>Out[] =</i> | <pre>{6.797 Second, {{1,2,7,8,17,18,23,24}, {1,2,25,26,95,96,119,120}, {1,2,55,56,83,84,107,108}, {1,3,8,11,14,17,22,24}, {1,3,25,27,61,63,85,87}, {1,3,56,59,80,83,106,108}, {1,6,8,10,15,17,19,24}, {1,6,25,30,68,70,110,112}, {1,6,56,58,81,83,103,108}, {1,7,27,37,51,61,81,87}, {1,7,30,44,54,68,106,112}, {1,15,26,45,76,95,106,120}, {1,15,27,33,55,61,73,87}, {1,22,26,40,81,95,99,120}, {1,22,30,36,55,68,98,112}}}}</pre> |
| <i>In[]:=</i> | Timing[SUBGRUPOSN4[10]] |
| <i>Out[] =</i> | <pre>{10.469 Second, {{1,8,27,38,53,68,83,94,113,120}, {1,8,30,43,52,61,90,95,108,117}, {1,17,26,47,64,68,78,87,108,109},</pre> |

```
{1,17,27,35,56,72,74,95,112,116},
{1,24,26,42,61,69,83,86,101,112},
{1,24,30,34,56,65,87,91,97,120}}}
```

In[]:= **Timing[SUBGRUPOSN4[12]]**

Out[]= {18.531 Second,
{ {1,2,3,4,5,6,25,26,27,28,29,30},
{1,2,7,8,25,26,31,32,49,50,55,56},
{1,2,7,8,81,82,87,88,105,106,111,112},
{1,2,15,16,21,22,55,56,61,62,67,68},
{1,3,7,9,13,15,106,108,112,114,118,120},
{1,3,22,24,25,27,46,48,100,102,106,108},
{1,3,22,24,55,57,68,71,79,81,92,95},
{1,4,5,8,9,12,13,16,17,20,21,24},
{1,4,5,56,57,60,79,82,83,104,105,108},
{1,6,7,12,20,22,81,83,87,89,93,95},
{1,6,15,17,25,30,39,41,75,77,81,83},
{1,6,15,17,55,60,61,66,104,106,115,120},
{1,9,13,27,31,39,49,57,61,75,79,87},
{1,12,20,30,31,46,49,60,68,100,104,112},
{1,16,21,26,39,46,75,82,95,100,105,120}}}}

In[]:= **Timing[SUBGRUPOSN4[15]]**

Out[]= {24.203 Second,{}}

In[]:= **Timing[SUBGRUPOSN4[20]]**

Out[]= {56.75 Second,
{ {1,8,18,23,27,36,38,45,53,58,63,68,76,83,85,94,
98,103,113,120},{1,8,18,23,30,33,40,43,52,59,
61,70,73,80,90,95,99,108,110,117},{1,10,17,19,
26,36,37,47,51,59,64,68,78,80,87,96,98,108,
109,119},{1,10,17,19,27,35,40,44,54,56,63,72,
74,84,85,95,99,107,112,116},{1,11,14,24,26,33,
42,44,54,58,61,69,73,83,86,96,101,103,112,
119},{1,11,14,24,30,34,37,45,51,56,65,70,76,
84,87,91,97,107,110,120}}}}

In[]:= **Timing[SUBGRUPOSN4[24]]**

Out[]= {113.781 Second,
{ {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,
19,20,21,22,23,24},{1,2,3,4,5,6,55,56,57,58,
59,60,79,80,81,82,83,84,103,104,105,106,107,
108},{1,2,15,16,21,22,25,26,39,40,45,46,75,
76,81,82,95,96,99,100,105,106,119,120},{1,3,
7,9,13,15,25,27,31,33,37,39,49,51,55,57,61,63,
73,75,79,81,85,87},{1,6,7,12,20,22,25,30,31,
36,44,46,49,54,55,60,68,70,98,100,104,106,

```

110,112} } }

In]:=      Timing[SUBGRUPOSN4[30]]

Out]= {127.172 Second, {}}

In]:=      Timing[SUBGRUPOSN4[40]]

Out]= {151.75 Second, {}}

In]:=      Timing[SUBGRUPOSN4[60]]

Out]= {709.844 Second,
{1,4,5,8,9,12,13,16,17,20,21,24,26,27,30,31,34,
35,38,39,42,43,46,47,49,52,53,56,57,60,61,64,
65,68,69,72,74,75,78,79,82,83,86,87,90,91,94,
95,97,100,101,104,105,108,109,112,113,116,117,
120} } }

```

En total se han empleado más de 20 minutos en el cálculo, podemos mejorar la eficacia con pequeños cambios en el programa atendiendo a algunos detalles relacionados con la programación.

- En SUBGRUPOSN4[], en las líneas:

| | |
|--|-------------------|
| <pre> elemento=op[Gsin [CONTADORi],Gsin [CONTADORi]]; While[elemento!=ElementoNeutro, conjunto=Union[conjunto,{elemento}]; elemento=op[elemento,Gsin [CONTADORi]];]; </pre> | (1) |
| \vdots \vdots | \vdots \vdots |
| <pre> Nuevos=Complement[conjunto,conjtemp2]; Do[If[Length[conjunto]<n+1, elemento=op[Nuevos [CONTADORx]], Nuevos [CONTADORx]]; While[!TrueQ[elemento==ElementoNeutro], conjunto=Union[conjunto,{elemento}]; elemento=op[elemento,Nuevos [CONTADORx]];]; ,Break[]]; ,{CONTADORx,Length[Nuevos]}]; </pre> | (2) |

Se calcula el subgrupo generado por un elemento, este cálculo se realiza varias veces en cada construcción y se repite a cada uso que hacemos de SUBGRUPOSN4[], para evitarlo, haremos el cálculo de todos los subgrupos

generados por un elemento previamente, almacenando estos cálculos en memoria, para no tener que volverlos a realizar:

```
In[]:=      Do[
  FUNCIÓN[CONTADORi]=
    Complement[GENERADO[{CONTADORi}],
    {ElementoNeutro}];
  ,{CONTADORi,Length[G]}];
```

Y sustituimos las líneas anteriores por:

| | |
|---|------------------|
| <code>conjunto=Union[Subconjuntos[{CONTADORj}], FUNCION[Gsin[{CONTADORj}]]];</code> | (1*) |
| <code>⋮ ⋮</code> | <code>⋮ ⋮</code> |
| <code>Nuevos=Complement[conjunto,conjtemp2]; Do[If[Length[conjunto]<n+1, conjunto=Union[conjunto, FUNCION[Nuevos[{CONTADORx}]]]; ,Break[];] ,{CONTADORx,Length[Nuevos]}];</code> | (2*) |

- Por otra parte en las construcciones de los subgrupos, para hacer a los subconjuntos candidatos cerrados para la operación, se realizan muchas operaciones entre elementos de G . Obsérvese que los elementos de G están identificados por su índice, a partir del cual también se construye la tabla de operaciones. Por tanto es claro que

$$\text{op}[i,j] \text{ es igual a } \text{operacion}[[i,j]]$$

y podemos evitar el uso de la función 2.1. (en este caso particular únicamente, salvo que identifiquemos previamente los elementos con sus índices), que al buscar la posición de los elementos para comprobar el resultado de operarlos en la tabla consume una gran cantidad de tiempo. Este cambio también puede hacerse en GENERADORES[] y también ganaríamos algo eficacia, aunque dejamos la exploración de esta opción al lector. Luego sustituiremos en SUBGRUPOS4[] las líneas:

| | |
|--|------------------|
| <code>conjunto=Union[conjunto, {op[Nuevos[{CONTADORx}], conjtemp[{CONTADORh}]], {op[conjtemp[{CONTADORh}], Nuevos[{CONTADORx}]}}];</code> | (3) |
| <code>⋮ ⋮</code> | <code>⋮ ⋮</code> |
| <code>conjunto=Union[conjunto,</code> | (4) |

| | |
|--|--|
| {op[Nuevos[[CONTADORx]], Nuevos[[CONTADORh]]], {op[Nuevos[[CONTADORh]], Nuevos[[CONTADORx]]]} }; | |
|--|--|

por estas otras,

| | |
|--|------|
| conjunto=Union[conjunto, {operacion[[Nuevos[[CONTADORx]], conjtemp[[CONTADORh]]]], {operacion[[conjtemp[[CONTADORh]], Nuevos[[CONTADORx]]]} }; ⋮ ⋮ | (3*) |
| conjunto=Union[conjunto, {operacion[[Nuevos[[CONTADORx]], Nuevos[[CONTADORh]]]], {operacion[[Nuevos[[CONTADORh]], Nuevos[[CONTADORx]]]} }; ⋮ ⋮ | (4*) |

Quedaría la función como sigue:

| FUNCIÓN | COMENTARIOS |
|--|--|
| G=GRUPO; operacion=TABLA DE OPERACIONES DE G; | Introducimos el grupo “G”. |
| op[x_,y_]:=... | Introducimos la función 2.1. |
| ElementoNeutro=... | Calculamos el elemento neutro con el programa 2.4. |
| GENERADO[A_]:=... | Introducimos la función 3.10. |
| Do[FUNCION[CONTADORi]=Complement[GENERADO[{CONTADORi},{ElementoNeutro}]; ,{CONTADORi,Length[G]}]; SUBGRUPOSN5[n_]:=Module[{Subconjuntos,Subconjuntostemp2,CONTADORi, CONTADORj,CONTADORk,CONTADORh, Gsin,conjunto,longit,Nuevos,conjtemp,conjtemp2, CONTADORx}, | Calculamos previamente y almacenamos en memoria los subgrupos generados por cada elemento. |
| If[Mod[Length[G],n]==0, If[n==1 n==Length[G], If[n==1,Subconjuntos={{ElementoNeutro}}]; If[n==Length[G],Subconjuntos={G}]; , | Función cuyo argumento es el orden de los subgrupos que pretendemos calcular |
| Subconjuntos={ElementoNeutro}}; | Hacemos cálculos sólo para los divisores del orden de “G” (teorema de Lagrange). |
| Do[| Si el orden 1 o el de “G” los cálculos son triviales. |

| | |
|---|--|
| Subconjuntostemp2={}; | una se añadirán elementos nuevos a los subconjuntos candidatos a subgrupos. |
| Do[| Bucle que recorre todos los subconjuntos creados hasta el momento. |
| If[Length[Subconjuntos[[CONTADORj]]]<n, | Comprobamos que el subconjunto tenga menos de "n" elementos. |
| Gsin=Complement[G, Subconjuntos[[CONTADORj]]]; Do[| Si tiene menos de "n" elementos añadiremos elementos nuevos. |
| longit=Length[Subconjuntos[[CONTADORj]]]; conjunto=Union[Subconjuntos[[CONTADORj]], FUNCION[Gsin[[CONTADORi]]]]; conjtemp=Subconjuntos[[CONTADORj]]; | Por cada elemento nuevo, se añaden también todas sus potencias, esto es, el subgrupo generado por el elemento. |
| While[longit!=Length[conjunto] && Length[conjunto]<n+1, Nuevos=Complement[conjunto, conjtemp]; conjtemp=Complement[conjtemp, {ElementoNeutro}]; conjtemp2=conjunto; | Bucle que dará vueltas hasta que después de operar unos elementos con otros no resulte ninguno nuevo. |
| Do[If[Length[conjunto]<n+1, Do[If[Length[conjunto]<n+1, conjunto=Union[conjunto, {operacion [Nuevos[[CONTADORx]], conjtemp[[CONTADORh]]]}], {operacion [conjtemp[[CONTADORh]], Nuevos[[CONTADORx]]]}]; ,Break[]]; ,{CONTADORh,1,longit-1}; ,Break[]]; ,{CONTADORx,Length[Nuevos]}]; | Se operan todos los elementos nuevos con los antiguos (almacenados en "conjtemp"). |
| Do[If[Length[conjunto]<n+1, Do[If[Length[conjunto]<n+1, conjunto=Union[conjunto, {operacion [Nuevos[[CONTADORx]], Nuevos[[CONTADORh]]]}], {operacion [Nuevos[[CONTADORh]], Nuevos[[CONTADORx]]]}]; ,Break[]]; ,{CONTADORh,CONTADORx+1, Length[Nuevos]}]; ,Break[]]; ,{CONTADORx,Length[Nuevos]}]; | Operamos todos los elementos nuevos entre ellos. |

| | |
|---|---|
| Nuevos=Complement[conjunto, conjtemp2]; Do[If[Length[conjunto]<n+1, conjunto=Union[conjunto, FUNCION] Nuevos[[CONTADORx]]]; ,Break[]];; {CONTADORx,Length[Nuevos]}]; conjtemp=conjtemp2; longit=Length[conjtemp];]; If[Length[conjunto]<n+1, Subconjuntostemp2=Union[Subconjuntostemp2,{conjunto}];; ,{CONTADORi,1,Length[Gsin]}]; , | Buscamos todos los elementos que han surgido al operar unos elementos con otros y calculamos todas sus potencias. |
| Subconjuntostemp2=Union[Subconjuntostemp2, {Subconjuntos[[CONTADORj]]};]; ,{CONTADORj,Length[Subconjuntos]}]; Subconjuntos=Subconjuntostemp2; ,{CONTADORk,1,n-1}];]; , | |
| Print["No es divisor del orden del grupo."]; Subconjuntos={};]; Subconjuntos | |
| | Se muestran los resultados. |

Función 3.15. Subgrupos de "n" elementos.

Ahora repetimos los cálculos, obtendremos las mismas salidas que con SUBGRUOSN4[] (3.14.) salvo los tiempos empleados, obviamos las salidas para no reiterarnos y resumimos los tiempos empleados en la siguiente tabla:

| ÓRDENES | Tiempos empleados en segundos | | | | | | | | | | | | | TOTAL | |
|----------------|--------------------------------------|----------|----------|----------|----------|----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|--------------|------|
| | 2 | 3 | 4 | 5 | 6 | 8 | 10 | 12 | 15 | 20 | 24 | 30 | 40 | | |
| SUBGRUOSN4[] | <0.1 | 0.6 | 0.9 | 2.4 | 3.4 | 7 | 10 | 19 | 24 | 58 | 114 | 128 | 152 | 710 | 1229 |
| SUBGRUOSN5[] | <0.1 | 0.1 | 0.2 | 0.5 | 0.9 | 1.8 | 2.7 | 4.5 | 5.5 | 12 | 25 | 28 | 33 | 203 | 317 |

Tabla 3.4. Efectividad.

Observemos que con estos pequeños cambios la mejora ha sido sustancial. Si hubiésemos pretendido hacer el cálculo con SUBGRUOSN[] (3.7.), se comprobarían alrededor de 10^{25} subconjuntos, suponiendo que tuviéramos un ordenador con la suficiente memoria como para gestionar tal número de subconjuntos en la forma en la que lo hace KSubsets[] y siendo muy optimistas podríamos suponer que se tardarían 10^{-4} segundos en comprobar con la función SUBGRUPO[] (3.1.) cada subconjunto (en realidad sería bastante más), lo cual supondría 10^{21} segundos, esto es, 3×10^{13} años (la edad del universo se estima en menos de 2×10^{10} años).

□

En términos de la cantidad de memoria usada, obsérvese que los primeros métodos comprobaban un gran número de subconjuntos, utilizábamos KSubsets[] y creábamos una lista de todos esos subconjuntos, lo que daba lugar a un uso elevado de memoria, por el contrario, los últimos métodos son constructivos, por lo que la cantidad de memoria necesaria es menor.

Obsérvese que algunos cálculos se repiten para cada orden que vamos considerando, por ejemplo para calcular los subgrupos de orden 60 de S_5 tenemos que repetir muchos de los cálculos que ya se había hecho para calcular el resto de subgrupos propios de S_5 . Si lo que deseamos es calcular todos los subgrupos de un grupo dado, podemos intentar evitar duplicar estos cálculos. Para ello partimos de todo lo aprendido en el diseño de la función SUBGRUPOSN5[] y construimos el siguiente programa, recordemos que las restricciones de SUBGRUPOSN5[] en cuanto al nombre de los elementos, identificados por sus subíndices, también las tendremos aquí, para evitarlas incluiremos unas líneas adicionales que cambiarán el nombre de los vértices para realizar los cálculos y los recuperará al final.

| FUNCIÓN | COMENTARIOS |
|--|---|
| G=GRUPO; operacion=TABLA DE OPERACIONES DE G; | Introducimos el grupo “G”. |
| op[x_,y_]:=... | Introducimos la función 2.1. |
| ElementoNeutro=... | Calculamos el elemento neutro con el programa 2.4. |
| GENERADO[A_]:=... | Introducimos la función 3.10. |
| SUBGRUPOS:=Block[{ciclicos,numero,tiempo, generado,CONTADORi,subgrupos,n, Subconjuntostemp2,Gsin,CONTADORj,longit, conjunto,conjtemp,Nuevos,conjtemp2, CONTADORx,CONTADORh,FUNCION, operacion2,G2,conjuntosubgrupos,i,j}, | Definimos el procedimiento y declaramos todas las variables locales que vamos a utilizar. |
| tiempo=TimeUsed[]; | En “tiempo” almacenamos el tiempo usado por Mathematica hasta este momento, al final volvemos a comprobar el tiempo usado y así determinamos con exactitud el tiempo usado por Mathematica en los cálculos intermedios. |
| n=Divisors[Length[G]][Length[Divisors[Length[G]]]-1]; | En “n” almacenamos el mayor orden posible de un subgrupo propio de “G”. |
| G2 = G; operacion2 = operacion; operacion = Table[Position[G, operacion[[i, j]]][[1]][[1]] ,{i, Length[G]}, {j, Length[G]}]; G = Table[i, {i, Length[G]}]; | Guardamos en “G2” y “operacion2” el grupo y la tabla de operaciones original y cambios los nombres de los elementos identificándolos por sus subíndices. |
| ciclicos={}; Do[generado=GENERADO[{CONTADORi}]; ciclicos=Union[ciclicos,{generado}]; FUNCION[CONTADORi]= Complement[generado,{ElementoNeutro}]; | Calculamos todos los subgrupos cíclicos y también simultáneamente almacenamos la información en “FUNCION” para poder usarla posteriormente. |

| | |
|---|--|
| <code>,{CONTADORi,Length[G]}];</code> | |
| <code>conjuntosubgrupos=ciclicos;</code> | La variable “conjuntosubgrupos” almacenará todos los subgrupos que vamos construyendo. |
| <code>subgrupos=Complement[ciclicos, {{ElementoNeutro}}];</code> | En “subgrupos” almacenaremos los subgrupos a los que posteriormente iremos añadiendo elementos. |
| <code>While[subgrupos!={}],</code> | Bucle principal, se añadirán nuevos elementos hasta que no resulte ningún subgrupo propio nuevo. |
| <code>Subconjuntostemp2={};</code> | En “Subconjuntostemp2” almacenaremos los subgrupos que vayamos construyendo. |
| <code>Do[</code> | |
| <code>Gsin=Complement[G, subgrupos [CONTADORj]]];</code> | Determinamos los elementos candidatos a añadir al subgrupo. |
| <code>Do[</code> | Bucle que recorrerá todos los elementos candidatos calculados y almacenados en “Gsin”. |
| <code>longit=Length[subgrupos [CONTADORj]]; conjunto=Union[subgrupos [CONTADORj]], FUNCION[Gsin [CONTADORi]]];</code> | Almacenamos el número de elementos del subgrupo, añadimos el nuevo elemento y todas sus potencias (almacenamos el nuevo conjunto en “conjunto”). |
| <code>conjtemp=subgrupos [CONTADORj];</code> | Bucle que recorre todos los subgrupos de “subgrupos” a los cuales queremos añadirle nuevos elementos. |
| <code>While[longit!=Length[conjunto] && Length[conjunto]<n+1,</code> | Bucle que sólo parará cuando ya no pueda dar lugar a un subgrupo propio o cuando sea cerrado para la operación y no se necesite añadirle más elementos. |
| <code>Nuevos=Complement[conjunto,conjtemp];</code> | Determinamos cuáles han sido los nuevos elementos añadidos |
| <code>conjtemp=Complement[conjtemp, {ElementoNeutro}]; conjtemp2=conjunto;</code> | |
| <code>Do[</code> <code>If[Length[conjunto]<n+1,</code> <code>Do[</code> <code>If[Length[conjunto]<n+1,</code> <code>conjunto=Union[conjunto,</code> <code>{operacion}[],</code> <code>Nuevos [CONTADORx]],</code> <code>conjtemp [CONTADORh]]],},</code> | Si el grupo es commutativo podemos simplificar estos cálculos y hacer la función más efectiva. En el caso de que los elementos no sean números o la tabla de operaciones |

| | |
|--|--|
| <pre>{operacion[] conjtemp[[CONTADORh]], Nuevos[[CONTADORx]]]];}; ,Break[];]; ,{CONTADORh,1,longit-1]; ,Break[];]; ,{CONTADORx,Length[Nuevos]}];</pre> | <p>no esté ordenada adecuadamente tendremos que usar la función op[].</p> |
| <pre>Do[If[Length[conjunto]<n+1, Do[If[Length[conjunto]<n+1, conjunto=Union[conjunto, {operacion[] Nuevos[[CONTADORx]], Nuevos[[CONTADORh]]]], {operacion[] Nuevos[[CONTADORh]], Nuevos[[CONTADORx]]}], ,Break[];]; ,{CONTADORh,CONTADORx+1, Length[Nuevos]}]; ,Break[];]; ,{CONTADORx,Length[Nuevos]}];</pre> | <p>Si el grupo es commutativo podemos simplificar estos cálculos y hacer la función más efectiva. En el caso de que los elementos no sean números o la tabla de operaciones no esté ordenada adecuadamente tendremos que usar la función op[].</p> |
| <pre>Nuevos=Complement[conjunto,conjtemp2]; Do[If[Length[conjunto]<n+1, conjunto=Union[conjunto, FUNCION[Nuevos[[CONTADORx]]]], , ,Break[];]; ,{CONTADORx,Length[Nuevos]}];</pre> | <p>Por cada elemento nuevo añadido, agregamos también todas sus potencias valiéndonos de “FUNCION”.</p> |
| <pre>conjtemp=conjtemp2; longit=Length[conjtemp];]; If[Length[conjunto]<n+1, Subconjuntostemp2=Union[Subconjuntostemp2,{conjunto}];]; ,{CONTADORi,1,Length[Gsin]}]; ,{CONTADORj,1,Length[subgrupos]}];</pre> | <p>Comprobamos si el nuevo subgrupo “conjunto” es propio y lo añadimos en caso afirmativo.</p> |
| <pre>subgrupos=Complement[Subconjuntostemp2, subgrupos]; conjuntosubgrupos=Union[conjuntosubgrupos, subgrupos];];</pre> | <p>Dejamos sólo aquellos subgrupos que han aparecido nuevos y que no estaban antes.</p> |
| | <p>Añadimos los nuevos subgrupos construidos.</p> |
| | <p>Cierre del bucle principal “While”.</p> |

| | |
|--|--|
| <pre> Print["Tiempo total: ", TimeUsed[]-tiempo]; operacion=operacion2; conjuntosubgrupos=Union[conjuntosubgrupos,{G}]; G=G2; cambiosubgrupos={}; Do[cambiosubgrupos=Union[cambiosubgrupos ,{Table[G[[conjuntosubgrupos[[i]][[j]]]] ,{j,Length[conjuntosubgrupos[[i]]]}]} ,{i,Length[conjuntosubgrupos]}]; cambiosubgrupos]; </pre> | <p>Se recupera el nombre original de los elementos y se muestran los resultados.</p> |
|--|--|

Procedimiento 3.16. Subgrupos.

La función SUBGRUPOS[] calcula en primer lugar todos los subgrupos cíclicos (generados por un único elemento) del grupo G , posteriormente a cada subgrupo de los ya construidos se le añade un nuevo elemento, dando lugar a todos los subgrupos generados por dos elementos, luego los de tres y así sucesivamente hasta que se dé la siguiente condición: no obtener ningún subgrupo propio nuevo después de añadir nuevos elementos a los subgrupos ya construidos.

Ejemplo 3.13. Calcular todos los subgrupos de S_5 usando el procedimiento 3.16. Definimos el grupo igual que en el ejemplo 3.12.

```

In]:= sigma[k_,n_][m_]:=Permutations[Table[j,{j,n}]][[k,m]];
In]:= composicion2[k_,j_,n_]:=Module[{t,m,z},
Position[Permutations[Table[m,{m,n}]],
Table[sigma[k,n][sigma[j,n][m]],[{m,1,n}]]][[1]][[1]]];
In]:= cuentas2[n_]:=Module[{k,j,m},
operacion=Table[composicion2[j,k,n],{k,n!},{j,n!}];
TableForm[operacion,TableHeadings→{Table[m,{m,n!}],
Table[m,{m,n!}]},TableSpacing→{2,2}]
];
In]:= G=Table[i,{i,5!}];
cuentas2[5];
operacion=Table[composicion2[j,k,5],{k,5!},{j,5!}];

```

Introducimos 3.16. y las funciones previas necesarias:

```

In]:= op[x_,y_]:=operacion[[Position[G,x][[1]],
Position[G,y][[1]]][[1]][[1]]];
In]:= ElementoNeutro="No existe";
⋮
⋮

```

Out[]:= Elemento Neutro: 1

In[]:= GENERADO[A_]:=Module[{CONTADORx,

⋮ ⋮

In[]:= SUBGRUPOS:=Block[

⋮ ⋮

In[]:= SUBGRUPOS

Out[]:= Tiempo total: 110.031
 $\{ \{1\}, \{1, 2\}, \{1, 3\}, \dots \}$

Mostramos en una tabla todos los subgrupos calculados:

| Órdenes | Nº | Subgrupos de S_5 |
|---------|----|--|
| 1 | 1 | {1} |
| 2 | 25 | {1,2},{1,3},{1,6},{1,7},{1,8},{1,15},{1,17},{1,22},{1,24},{1,25},{1,26},{1,27},{1,30},{1,55},{1,56},{1,61},{1,68},{1,81},{1,83},{1,87},{1,95},{1,106},{1,108},{1,112},{1,120} |
| 3 | 10 | {1,4,5},{1,9,13},{1,12,20},{1,16,21},{1,31,49},{1,39,75},{1,46,100},{1,57,79},{1,60,104},{1,82,105} |
| 4 | 35 | {1,2,7,8},{1,2,25,26},{1,2,55,56},{1,3,22,24},{1,3,25,27},{1,3,106,108},{1,6,15,17},{1,6,25,30},{1,6,81,83},{1,7,81,87},{1,7,106,112},{1,8,17,24},{1,8,18,23},{1,10,17,19},{1,11,14,24},{1,15,55,61},{1,15,106,120},{1,22,55,68},{1,22,81,95},{1,26,95,120},{1,26,96,119},{1,27,61,87},{1,27,63,85},{1,30,68,112},{1,30,70,110},{1,33,61,73},{1,36,68,98},{1,37,51,87},{1,40,95,99},{1,44,54,112},{1,45,76,120},{1,56,83,108},{1,56,84,107},{1,58,83,103},{1,59,80,108} |
| 5 | 6 | {1,34,65,91,97},{1,35,72,74,116},{1,38,53,94,113},{1,42,69,86,101},{1,43,52,90,117},{1,47,64,78,109} |
| 6 | 30 | {1,2,3,4,5,6},{1,2,15,16,21,22},{1,2,31,32,49,50},{1,2,81,82,105,106},{1,3,7,9,13,15},{1,3,46,48,100,102},{1,3,55,57,79,81},{1,4,5,25,28,29},{1,4,5,26,27,30},{1,6,7,12,20,22},{1,6,39,41,75,77},{1,6,55,60,104,106},{1,7,25,31,49,55},{1,7,82,88,105,111},{1,8,26,31,49,56},{1,8,82,87,105,112},{1,9,13,106,114,118},{1,9,13,108,112,120},{1,12,20,81,89,93},{1,12,20,83,87,95},{1,15,25,39,75,81},{1,15,60,66,104,115},{1,16,21,55,62,67},{1,16,21,56,61,68},{1,17,30,39,75,83},{1,17,60,61,104,120},{1,22,25,46,100,106},{1,22,57,71,79,92},{1,24,27,46,100,108},{1,24,57,68,79,95} |
| 8 | 15 | {1,2,7,8,17,18,23,24},{1,2,25,26,95,96,119,120},{1,2,55,56,83,84,107,108},{1,3,8,11,14,17,22,24},{1,3,25,27,61,63,85,87},{1,3,56,59,80,83,106,108},{1,6,8,10,15,17,19,24},{1,6,25,30,68,70,110,112},{1,6,56,58,81,83,103,108},{1,7,27,37,51,61,81,87},{1,7,30,44,54,68,106,112},{1,15,26,45,76,95,106,120},{1,15,27,33,55,61,73,87},{1,22,26,40,81,95,99,120},{1,22,30,36,55,68,98,112} |
| 10 | 6 | {1,8,27,38,53,68,83,94,113,120},{1,8,30,43,52,61,90,95,108,117},{1,17,26,47,64,68,78,87,108,109},{1,17,27,35,56,72,74,95,112,116},{1,24,26,42,61,69,83,86,101,112},{1,24,30,34,56,65,87,91,97,120} |
| 12 | 15 | {1,2,3,4,5,6,25,26,27,28,29,30},{1,2,7,8,25,26,31,32,49,50,55,56},{1,2,7,8,81,82,87,88,105,106,111,112},{1,2,15,16,21,22,55,61,62,67,68},{1,3,7,9,13,15,106,108,112,114,118,120},{1,3,22,24,25,27,46,48,100,102,106,108},{1,3,22,24,55,57,68,71,79,81,92,95},{1,4,5,8,9,12,13,16,17,20,21,24},{1,4,5,56,57,60,79,82,83,104,105,108},{1,6,7,12,20,22,81,83,87,89,93,95},{1,6,15,17,25,30,39,41,75,77,81,83},{1,6,15,17,55,60,61,66,104,106,115,120},{1,9,13,27,31,39,49,57,61,75,79,87},{1,12,20,30,31,46,49,60,68,100,104,112},{1,16,21,26,39,46,75,82,95,100,105,120} |
| 20 | 6 | {1,8,18,23,27,36,38,45,53,58,63,68,76,83,85,94,98,103,113,120},{1,8,18,23,30,33,40,43,52,59,61,70,73,80,90,95,99,108,110,117},{1,10,17,19,26,36,37,47,51,59,64,68,78,80,87,96,98,108,109,119},{1,10,17,19,27,35,40,44,54,56,63,72,74,84,85,95,99,107,112,116},{1,11,14,24,26,33,42,44,54,58,61,69,73,83,86,96,101,103,112,119},{1,11,14,24,30,34,37,45,51,56,65,70,76,84,87,91,97,107,110,120} |

| | | |
|-----|---|---|
| 24 | 5 | {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24}, {1,2,3,4,5,6,55,56,57,58,59,60,79,80,81,82,83,84,103,104,105,106,107,108}, {1,2,15,16,21,22,25,26,39,40,45,46,75,76,81,82,95,96,99,100,105,106,119,120}, {1,3,7,9,13,15,25,27,31,33,37,39,49,51,55,57,61,63,73,75,79,81,85,87}, {1,6,7,12,20,22,25,30,31,36,44,46,49,54,55,60,68,70,98,100,104,106,110,112} |
| 60 | 1 | {1,4,5,8,9,12,13,16,17,20,21,24,26,27,30,31,34,35,38,39,42,43,46,47,49,52,53,56,57,60,61,64,65,68,69,72,74,75,78,79,82,83,86,87,90,91,94,95,97,100,101,104,105,108,109,112,113,116,117,120} |
| 120 | 1 | {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119,120} |

Tabla 3.4. Subgrupos de S_5 .

□

Terminamos con un nuevo procedimiento, una versión simplificada del procedimiento 3.16. Si bien es bastante menos efectivo que 3.16., su estructura y funcionamiento es análogo pero con una mayor claridad, por lo que consideramos apropiada su inclusión ya que nos ayudará a entender mejor 3.15. y 3.16. En éste se ha suprimido parte del código y se ha usado directamente la función 3.10. para construir los nuevos subgrupos.

| FUNCIÓN/PROCEDIMIENTO | COMENTARIOS |
|--|--|
| G=GRUPO; operacion=TABLA DE OPERACIONES DE G; | Introducimos el grupo “G”. |
| op[x_,y_]:=... | Introducimos la función 2.1. |
| ElementoNeutro=... | Calculamos el elemento neutro con el programa 2.4. |
| GENERADO[A_]:=... | Introducimos la función 3.10. |
| SUBGRUPOS2:=Block[{ciclicos,tiempo,generado, CONTADORi,subgrupos,G2,operacion2,i,j, Subconjuntostemp2,Gsin,CONTADORj,conjunto, conjtemp,conjtemp2}, tiempo=TimeUsed[]]; | Definimos el procedimiento y declaramos todas las variables locales que vamos a utilizar. |
| G2 = G; operacion2 = operacion; operacion = Table[Position[G, operacion[[i, j]]][[1]][[1]] , {i, Length[G]}, {j, Length[G]}]; G = Table[i, {i, Length[G]}]; | Guardamos en “G2” y “operacion2” el grupo y la tabla de operaciones original y cambiamos los nombres de los elementos identificándolos por sus subíndices. |
| ciclicos={}; Do[generado=GENERADO[{CONTADORi}]; ciclicos=Union[ciclicos,{generado}]; ,{CONTADORi,Length[G]}]; | Calculamos todos los subgrupos cíclicos. |
| conjuntosubgrupos=ciclicos; | La variable “conjuntosubgrupos” almacenará todos los subgrupos que vamos construyendo. |
| subgrupos=Complement[ciclicos, {ElementoNeutro}]; | En “subgrupos” almacenaremos los subgrupos a los que añadiremos un elemento. |
| While[subgrupos!={}, | Bucle principal, se irán añadiendo elementos hasta que no resulte ningún subgrupo propio nuevo. |
| Subconjuntostemp2={}; Do[| En “Subconjuntostemp2” almacenaremos los subgrupos que vayamos construyendo. |

| | | |
|---|--|---|
| Gsin=Complement[G, subgrupos[[CONTADORj]]]; | Determinamos los elementos candidatos a añadir al subgrupo. | |
| Do[| | |
| conjunto=GENERADO[Union[subgrupos[[CONTADORj]], Gsin[[CONTADORi]]]]; | Calculamos el subgrupo generado por el nuevo conjunto. | Bucle que recorre todos los subgrupos de “subgrupos”. |
| Subconjuntostemp2=Union[Subconjuntostemp2,{conjunto}]; | Añadimos el nuevo grupo. | |
| ,{CONTADORi,1,Length[Gsin]}]; | | |
| ,{CONTADORj,1,Length[subgrupos]}]; | | |
| subgrupos=Complement[Subconjuntostemp2, subgrupos]; | Dejamos sólo aquellos subgrupos que han aparecido nuevos y que no estaban antes. | |
| conjuntosubgrupos=Union[conjuntosubgrupos, subgrupos]; | Añadimos los nuevos subgrupos construidos. | |
|]; | Cierre del bucle principal “While”. | |
| Print["Tiempo total: ", TimeUsed[]-tiempo]; operacion=operacion2; conjuntosubgrupos=Union[conjuntosubgrupos,{G}]; G=G2; cambiosubgrupos={}; Do[cambiosubgrupos=Union[cambiosubgrupos ,{Table[G][conjuntosubgrupos[[i]][[j]]]] ,{j,Length[conjuntosubgrupos[[i]]}]} ,{i,Length[conjuntosubgrupos]}]; cambiosubgrupos]; | Se recupera el nombre original de los elementos y se muestran los resultados. | |

Procedimiento 3.17. Subgrupos.

Ejemplo 3.14. Calcular y comprobar el tiempo empleado en el cálculo de todos los subgrupos de S_4 y S_5 usando el procedimiento 3.17. Definimos los grupos igual que en los ejemplos anteriores. Para ello tomamos prestadas del capítulo 4 las siguientes funciones:

```
In]:= sigma[k_,n_][m_]:=Permutations[Table[j,{j,n}]][[k,m]];
In]:= composicion2[k_,j_,n_]:=Module[{t,m,z},
Position[Permutations[Table[m,{m,n}]],
Table[sigma[k,n][sigma[j,n][m]],[{m,1,n}]]][[1]][[1]]];
In]:= cuentas2[n_]:=Module[{k,j,m},
operacion=Table[composicion2[j,k,n],{k,n!},{j,n!}];
TableForm[operacion,TableHeadings→{Table[m,{m,n!}],
Table[m,{m,n!}]},TableSpacing→{2,2}]];
]
```

Introducimos 3.17. junto con las funciones previas necesarias:

```
In]:= op[x_,y_]:=operacion[[Position[G,x][[1]],
```

```

Position[G,y][[1]]][[1]][[1]];

In[]:= ElementoNeutro="No existe";
          :
          :

Out[] = Elemento Neutro: 1

In[]:= GENERADO[A_]:=Module{CONTADORx,
          :
          :

In[]:= SUBGRUPOS2:=Block[
          :
          :


```

Definimos S_4 :

```

In[]:= G=Table[i,{i,4!}];
cuentas2[4];
operacion=Table[composicion2[j,k,4],{k,4!},{j,4!}];

```

y calculamos sus subgrupos:

```

In[]:= SUBGRUPOS2

Out[] = Tiempo total: 6.603
{ {1}, {1, 2}, {1, 3}, ... }

```

Igual con S_5 :

```

In[]:= G=Table[i,{i,5!}];
cuentas2[5];
operacion=Table[composicion2[j,k,5],{k,5!},{j,5!}];

In[]:= SUBGRUPOS2

Out[] = Tiempo total: 12270.6
{ {1}, {1, 2}, {1, 3}, ... }

```

□

Existen otras optimizaciones posibles basadas en teoría de grupos o en programación. Aquí sólo se ha dado una muestra de la complejidad que pueden tener ciertos cálculos y de la necesidad e importancia de estudiar la efectividad y optimización de los algoritmos usados, pensemos que se ha analizado un grupo de 120 elementos, en la sección 6 del capítulo 4 se utiliza un grupo de 403291461126605635584000000 elementos, en tal caso y si nuestro problema consistiera en calcular, si es que existen, los subgrupos de 13314502656 elementos, ¿cómo lo haríamos?

En conclusión podemos decir que la cuestión de la efectividad es muy compleja y que además de las propiedades matemáticas, depende de muchos otros factores, los resultados serían muy distintos en otras plataformas. Estarían en manos del lenguaje que usemos y de la efectividad de las funciones que lo integran, para un análisis completo habría que conocer en profundidad la efectividad de cada una de las instrucciones que usemos, para dejar de usarlas o usarlas con mayor o menor frecuencia. Nuestro punto de vista ha sido más algebraico que computacional y se han acelerado los procesos fijándonos sobre todo en las propiedades matemáticas de los conceptos analizados, con ello hemos conseguido acelerar el cálculo enormemente, podríamos seguir especulando¹⁴ y analizando para acelerar aún más el cálculo, si bien, consideramos que como muestra es suficiente con lo visto.

7. OTROS EJEMPLOS. CASO INFINITO

Resolvemos algunos ejemplos del caso infinito:

Ejemplo 3.15. $(\mathbb{R} \times \mathbb{R} \times \mathbb{R}, +)$ es un grupo abeliano, consideramos el subconjunto:

$$H = \{(x, y, z) \in \mathbb{R} \times \mathbb{R} \times \mathbb{R} \mid 2x + y = 0 \quad \wedge \quad x - z = 0\},$$

comprobamos que es un subgrupo. Consideramos $(x_1, y_1, z_1), (x_2, y_2, z_2) \in H$, tendremos que demostrar que $(x_1, y_1, z_1) - (x_2, y_2, z_2) \in H$, esto es,

$$2(x_1 - x_2) + (y_1 - y_2) = 0 \quad \text{y} \quad (x_1 - x_2) - (z_1 - z_2) = 0,$$

en efecto,

In[]:= **Solve**[{2x1+ y1==0,x1-z1==0,2 x2+y2==0,x2-z2==0,
a==2 (x1-x2)+(y1-y2),b==(x1-x2)-(z1-z2)}, {a,b}]

Out[] = { {a→0 ,b→0} }

□

Ejemplo 3.16. Sea G el grupo del ejemplo 2.14. y sea $H = \{(a, b) \in \mathbb{R} \times \mathbb{R} \mid a^2 + b^2 = 1\}$, comprobar que es un subgrupo de G .

Calculamos $(a, b) \cdot (c, d)^{-1}$ para dos elementos $(a, b), (c, d) \in H$.

Del ejemplo 2.13. sabemos que:

$$(c, d)^{-1} = \left(\frac{c}{c^2 + d^2}, -\frac{d}{c^2 + d^2} \right).$$

¹⁴ En el ejemplo 3.12. se hacen algunas consideraciones más sobre la eficacia del cálculo de subgrupos, relacionadas con la programación.

Y,

$$\text{In}[]:= \quad \text{Simplify}[\text{Expand}[(\text{op}[\{a,b\}, \{c/(c^2+d^2), \\ -d/(c^2+d^2)\}][[1]])^2 + (\text{op}[\{a,b\}, \{c/(c^2+d^2), \\ -d/(c^2+d^2)\}][[2]])^2]]$$

$$\text{Out}[] = \frac{a^2 + b^2}{c^2 + d^2}$$

Por tanto es subgrupo.

Obsérvese además que (\mathbb{C}^*, \cdot) es un grupo isomorfo a (G, \cdot) y que H es el subconjunto de los complejos con módulo 1, para más detalle véase el ejercicio 2.10. \square

8. EFICACIA Y OPTIMIZACIÓN EN EL CÁLCULO DE GRUPOS DE ORDEN PEQUEÑO

En el capítulo 2 calculábamos todos los grupos distintos salvo isomorfismo de un orden pequeño, además observábamos que incluso para órdenes muy pequeños (8 elementos o más) el tiempo de cálculo necesario era elevado. En este epígrafe intentaremos acelerar los cálculos utilizando las siguientes consecuencias de los teoremas de Cauchy para grupos finitos y de Lagrange:

Corolario 3.4. Si el orden de un grupo es un primo p , entonces tendrá un elemento de orden p y por tanto será cíclico e isomorfo a $(\mathbb{Z}_p, +)$. \square

Corolario 3.5. Si el orden de un grupo G es n de la forma p^k con p primo, entonces G tiene al menos un elemento de orden p^2 o todos los elementos de G son de orden p . \square

Corolario 3.6. Si el orden de un grupo G es n con factorización $p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ donde todos los p_i son primos distintos, entonces G tiene al menos k elementos distintos de órdenes p_1, p_2, \dots, p_k respectivamente. \square

Integraremos estos tres corolarios en el siguiente programa:

| PROGRAMA | COMENTARIOS |
|---|---|
| <code>HOMOMORFISMO[G1_operacion1_, G2_operacion2]:=...</code> | Definimos la función 2.11. |
| <code>QUITARISOMORFOS[tablasgrupos_]:=...</code> | Definimos la función 2.13. |
| <code>GRUPOSPEQ[tablaoperaciones_]:=...</code> | Definimos la función 2.14. |
| <code>TablaZ[n_]:=Table[Mod[x+y-2,n]+1,{x,n},{y,n}];</code> | Definimos la función "TablaZ[n]" que define la tabla de operaciones del grupo |

| | |
|---|---|
| | cíclico ($\mathbb{Z}_n, +$). |
| n=NUMERO DE ELEMENTOS; tiempo=TimeUsed[]; listatablas2={}; If[PrimeQ[n], Print[n," es primo"]; listatablas={TablaZ[n]};, factorizacion=FactorInteger[n]; If[Length[factorizacion]==1, Print[n," es de la forma ",factorizacion[[1]][[1]], " elevado a ",factorizacion[[1]][[2]]]; Print["Caso 1, Grupos que tienen al menos un elementos de orden ",factorizacion[[1]][[1]]^2]; If[factorizacion[[1]][[1]]^2==n, listatablas2={TablaZ[n]}; , tabla=Table[If[i==1,j,If[j==1,i,0]],{i,n},{j,n}]; subtabla=TablaZ[factorizacion[[1]][[1]]^2]; Do[Do[tabla[[j,k]]=subtabla[[j,k]]; ,{j,2,Length[subtabla]}];,{k,2,Length[subtabla]}]; listatablas2= QUITARISOMORFOS[GRUPOSPEQ[tabla]];]; | Introducimos el número de elementos. Comprobamos si "n" es primo Si "n" es primo aplicamos el corolario 3.3. Si "n" no es primo, lo factorizamos. Comprobamos si "n" es la potencia de un primo. En caso afirmativo aplicamos el corolario 3.4. y distinguimos los dos casos posibles. |
| Print["Caso 2, Grupos que no tienen elementos de orden ",factorizacion[[1]][[1]]^2]; tabla=Table[If[i==1,j,If[j==1,i,0]],{i,n},{j,n}]; l=0; While[l!=n-1, subtabla=TablaZ[factorizacion[[1]][[1]]]; Do[Do[If[subtabla[[j,k]]==1, tabla[[j+l,k+l]]=subtabla[[j,k]]; , tabla[[j+l,k+l]]=subtabla[[j,k]]+l;]; ,{j,2,Length[subtabla]}];,{k,2,Length[subtabla]}]; l+=Length[subtabla]-1;]; listatablas= QUITARISOMORFOS[GRUPOSPEQ[tabla]]; , | Caso 1: Los grupos que al menos tienen un elemento de orden p^2 . Caso 2: Todos los elementos son de orden p . |
| tabla=Table[If[i==1,j,If[j==1,i,0]],{i,n},{j,n}]; l=0; Do[subtabla=TablaZ[factorizacion[[i]][[1]]]; Do[Do[If[subtabla[[j,k]]==1, tabla[[j+l,k+l]]=subtabla[[j,k]]; , tabla[[j+l,k+l]]=subtabla[[j,k]]+l;]; ,{j,2,Length[subtabla]}];,{k,2,Length[subtabla]}]; | Si "n" no es primo, ni potencia de un primo, entonces aplicamos el corolario 3.5. Si p es un primo que aparece en la factorización de "n" entonces G tiene al menos un elemento de orden p . |

| | |
|---|--|
| <pre>]; ,{j,2,Length[subtabla]}];,{k,2,Length[subtabla]}]; l=+Length[subtabla]-1; ,{i,Length[factorizacion]}; listatablas= QUITARISOMORFOS[GRUPOSPEQ[tabla]];];]; Print["Tiempo empleado: ",TimeUsed[]-tiempo]; grupos=Union[listatablas,listatablas2]; Length[grupos] </pre> | |
| | En “grupos” almacenamos las tablas de operaciones de todos los grupos que hemos obtenido. Mostramos el número de grupos que hemos calculado. |

Programa 3.18. Grupos de orden pequeño.

Ejemplo 3.17. Calculamos usando el programa anterior todos los grupos distintos salvo isomorfismo de 8, 9 y 10 elementos. En primer lugar definimos todas las funciones previas necesarias:

*In//]:= **HOMOMORFISMO[G1_,operacion1_,G2_,operacion2_,grafoF_]:=Module[***

⋮ ⋮

*In//]:= **QUITARISOMORFOS[tablasgrupos_]:=Module[***

⋮ ⋮

*In//]:= **GRUPOSPEQ[tablaoperaciones_]:=Module[***

⋮ ⋮

*In//]:= **TablaZ[n_]:=Table[Mod[x+y-2,n]+1,{x,n},{y,n}];***

Y ahora utilizamos el programa 3.18. para $n = 8, 9$ y 10 , y mostramos las tablas de operaciones de los grupos calculados.

- Ocho elementos:

*In//]:= **n=8;***

⋮ ⋮

Out//=

⋮ ⋮

5

*In//]:= **TableForm[Table[***

{StyleForm[StringJoin["Grupo ",ToString[j],":"],

FontSize→22],TableForm[grupos[[j]],

```
TableHeadings→{Table[StyleForm[i,
FontWeight→"Bold"],{i,n}],
Table[StyleForm[i,FontWeight→"Bold"],{i,n}]},
TableSpacing→{2,2}],{j,Length[grupos]}],
TableSpacing→{6,2}]
```

Out[] =

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-----------------|----------|----------|----------|----------|----------|----------|----------|----------|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 2 | 2 | 1 | 4 | 3 | 6 | 5 | 8 | 7 |
| 3 | 3 | 4 | 1 | 2 | 7 | 8 | 5 | 6 |
| Grupo 1: | 4 | 4 | 3 | 2 | 1 | 8 | 7 | 6 |
| | 5 | 5 | 6 | 7 | 8 | 1 | 2 | 3 |
| | 6 | 6 | 5 | 8 | 7 | 2 | 1 | 4 |
| | 7 | 7 | 8 | 5 | 6 | 3 | 4 | 1 |
| | 8 | 8 | 7 | 6 | 5 | 4 | 3 | 2 |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 2 | 2 | 3 | 4 | 1 | 6 | 7 | 8 | 5 |
| 3 | 3 | 4 | 1 | 2 | 7 | 8 | 5 | 6 |
| Grupo 2: | 4 | 4 | 1 | 2 | 3 | 8 | 5 | 6 |
| | 5 | 5 | 6 | 7 | 8 | 1 | 2 | 3 |
| | 6 | 6 | 7 | 8 | 5 | 2 | 3 | 4 |
| | 7 | 7 | 8 | 5 | 6 | 3 | 4 | 1 |
| | 8 | 8 | 5 | 6 | 7 | 4 | 1 | 2 |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 2 | 2 | 3 | 4 | 1 | 6 | 7 | 8 | 5 |
| 3 | 3 | 4 | 1 | 2 | 7 | 8 | 5 | 6 |
| Grupo 3: | 4 | 4 | 1 | 2 | 3 | 8 | 5 | 6 |
| | 5 | 5 | 6 | 7 | 8 | 2 | 3 | 4 |
| | 6 | 6 | 7 | 8 | 5 | 3 | 4 | 1 |
| | 7 | 7 | 8 | 5 | 6 | 4 | 1 | 2 |
| | 8 | 8 | 5 | 6 | 7 | 1 | 2 | 3 |
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 2 | 2 | 3 | 4 | 1 | 6 | 7 | 8 | 5 |
| 3 | 3 | 4 | 1 | 2 | 7 | 8 | 5 | 6 |
| Grupo 4: | 4 | 4 | 1 | 2 | 3 | 8 | 5 | 6 |
| | 5 | 5 | 6 | 7 | 8 | 2 | 3 | 4 |
| | 6 | 6 | 7 | 8 | 5 | 3 | 4 | 1 |
| | 7 | 7 | 8 | 5 | 6 | 4 | 1 | 2 |
| | 8 | 8 | 5 | 6 | 7 | 1 | 2 | 3 |

| | | | | | | | | | |
|-----------------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| | 4 | 4 | 1 | 2 | 3 | 8 | 5 | 6 | 7 |
| | 5 | 5 | 8 | 7 | 6 | 1 | 4 | 3 | 2 |
| | 6 | 6 | 5 | 8 | 7 | 2 | 1 | 4 | 3 |
| | 7 | 7 | 6 | 5 | 8 | 3 | 2 | 1 | 4 |
| | 8 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| | 2 | 2 | 3 | 4 | 1 | 6 | 7 | 8 | 5 |
| | 3 | 3 | 4 | 1 | 2 | 7 | 8 | 5 | 6 |
| Grupo 5: | 4 | 4 | 1 | 2 | 3 | 8 | 5 | 6 | 7 |
| | 5 | 5 | 8 | 7 | 6 | 3 | 2 | 1 | 4 |
| | 6 | 6 | 5 | 8 | 7 | 4 | 3 | 2 | 1 |
| | 7 | 7 | 6 | 5 | 8 | 1 | 4 | 3 | 2 |
| | 8 | 8 | 7 | 6 | 5 | 2 | 1 | 4 | 3 |

El grupo 1 es isomorfo a $\mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_2$, el 2 lo es isomorfo a $\mathbb{Z}_4 \times \mathbb{Z}_2$ y el grupo 3 lo es a \mathbb{Z}_8 . Los grupos 4 y 5 no son comutativos, 4 es isomorfo al grupo diédrico de orden 4, D_4 (ver el epígrafe 4. del capítulo 4) y el grupo 5 es isomorfo al grupo de los cuaternios Q_2 (ver ejercicio 2.30).

2. Nueve elementos:

In[7]:= **n=9;**

⋮ ⋮

Out[7]=

⋮ ⋮

2

In[7]:=

```
TableForm[Table[
  {StyleForm[StringJoin["Grupo ", ToString[j], ":"], 
    FontSize -> 22], TableForm[grupos[[j]],
    TableHeadings -> {Table[StyleForm[i,
      FontWeight -> "Bold"], {i, n}],
      Table[StyleForm[i, FontWeight -> "Bold"], {i, n}]}, 
    TableSpacing -> {2, 2}], {j, Length[grupos]}]}, 
  TableSpacing -> {6, 2}]]
```

Out[7]=

| | | | | | | | | | | |
|-----------------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Grupo 1: | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| | 2 | 2 | 3 | 1 | 6 | 8 | 9 | 5 | 7 | 4 |

| | | | | | | | | | | |
|-----------------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| | 3 | 3 | 1 | 2 | 9 | 7 | 4 | 8 | 5 | 6 |
| | 4 | 4 | 6 | 9 | 5 | 1 | 8 | 3 | 2 | 7 |
| | 5 | 5 | 8 | 7 | 1 | 4 | 2 | 9 | 6 | 3 |
| | 6 | 6 | 9 | 4 | 8 | 2 | 7 | 1 | 3 | 5 |
| | 7 | 7 | 5 | 8 | 3 | 9 | 1 | 6 | 4 | 2 |
| | 8 | 8 | 7 | 5 | 2 | 6 | 3 | 4 | 9 | 1 |
| | 9 | 9 | 4 | 6 | 7 | 3 | 5 | 2 | 1 | 8 |
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | |
| 2 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 1 | |
| 3 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 1 | 2 | |
| Grupo 2: | 4 | 4 | 5 | 6 | 7 | 8 | 9 | 1 | 2 | 3 |
| | 5 | 5 | 6 | 7 | 8 | 9 | 1 | 2 | 3 | 4 |
| | 6 | 6 | 7 | 8 | 9 | 1 | 2 | 3 | 4 | 5 |
| | 7 | 7 | 8 | 9 | 1 | 2 | 3 | 4 | 5 | 6 |
| | 8 | 8 | 9 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| | 9 | 9 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Podemos fácilmente comprobar que son isomorfos a $\mathbb{Z}_3 \times \mathbb{Z}_3$ y \mathbb{Z}_9 , respectivamente.

3. Diez elementos:

In[]:= **n=10;**

⋮ ⋮

Out[]=

⋮ ⋮

2

In[]:=

```
TableForm[Table[
 {StyleForm[StringJoin["Grupo ",ToString[j],":"]], 
  FontSize→22},TableForm[grupos[[j]],
  TableHeadings→{Table[StyleForm[i,
  FontWeight→"Bold"],{i,n}],
  Table[StyleForm[i,FontWeight→"Bold"],{i,n}]},
  TableSpacing→{2,2}],{j,Length[grupos]}],
  TableSpacing→{6,2}]
```

Out[]=

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|----|----|----|----|----|----|----|----|----|----|----|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 2 | 2 | 1 | 7 | 8 | 9 | 10 | 3 | 4 | 5 | 6 |
| 3 | 3 | 7 | 4 | 5 | 6 | 1 | 8 | 9 | 10 | 2 |
| 4 | 4 | 8 | 5 | 6 | 1 | 3 | 9 | 10 | 2 | 7 |
| 5 | 5 | 9 | 6 | 1 | 3 | 4 | 10 | 2 | 7 | 8 |
| 6 | 6 | 10 | 1 | 3 | 4 | 5 | 2 | 7 | 8 | 9 |
| 7 | 7 | 3 | 8 | 9 | 10 | 2 | 4 | 5 | 6 | 1 |
| 8 | 8 | 4 | 9 | 10 | 2 | 7 | 5 | 6 | 1 | 3 |
| 9 | 9 | 5 | 10 | 2 | 7 | 8 | 6 | 1 | 3 | 4 |
| 10 | 10 | 6 | 2 | 7 | 8 | 9 | 1 | 3 | 4 | 5 |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|----|---|---|---|----|---|---|---|----|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 2 | 2 | 1 | 7 | 8 | 9 | 10 | 3 | 4 | 5 | 6 |
| 3 | 3 | 10 | 4 | 5 | 6 | 1 | 2 | 7 | 8 | 9 |

| | | | | | | | | | | | |
|----------|----|----|----|----|----|----|---|----|----|---|---|
| Grupo 1: | 4 | 4 | 8 | 5 | 6 | 1 | 3 | 9 | 10 | 2 | 7 |
| | 5 | 5 | 9 | 6 | 1 | 3 | 4 | 10 | 2 | 7 | 8 |
| | 6 | 6 | 10 | 1 | 3 | 4 | 5 | 2 | 7 | 8 | 9 |
| | 7 | 7 | 3 | 8 | 9 | 10 | 2 | 4 | 5 | 6 | 1 |
| | 8 | 8 | 4 | 9 | 10 | 2 | 7 | 5 | 6 | 1 | 3 |
| | 9 | 9 | 5 | 10 | 2 | 7 | 8 | 6 | 1 | 3 | 4 |
| | 10 | 10 | 6 | 2 | 7 | 8 | 9 | 1 | 3 | 4 | 5 |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|----|---|---|---|----|---|---|---|----|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 2 | 2 | 1 | 7 | 8 | 9 | 10 | 3 | 4 | 5 | 6 |
| 3 | 3 | 10 | 4 | 5 | 6 | 1 | 2 | 7 | 8 | 9 |

| | | | | | | | | | | | |
|----------|----|----|---|----|----|----|---|----|----|----|---|
| Grupo 2: | 4 | 4 | 9 | 5 | 6 | 1 | 3 | 10 | 2 | 7 | 8 |
| | 5 | 5 | 8 | 6 | 1 | 3 | 4 | 9 | 10 | 2 | 7 |
| | 6 | 6 | 7 | 1 | 3 | 4 | 5 | 8 | 9 | 10 | 2 |
| | 7 | 7 | 6 | 8 | 9 | 10 | 2 | 1 | 3 | 4 | 5 |
| | 8 | 8 | 5 | 9 | 10 | 2 | 7 | 6 | 1 | 3 | 4 |
| | 9 | 9 | 4 | 10 | 2 | 7 | 8 | 5 | 6 | 1 | 3 |
| | 10 | 10 | 3 | 2 | 7 | 8 | 9 | 4 | 5 | 6 | 1 |

Que son isomorfos a \mathbb{Z}_{10} y D_5 (ver el epígrafe 4. Del capítulo 4).

□

Existen otras propiedades matemáticas que permitirían acelerar y optimizar aún más los cálculos, sin embargo, estos escapan a los objetivos de este libro. Lo expuesto es una muestra representativa de cuánto y cómo, se aceleran los cálculos y pueden implementarse las propiedades matemáticas en el cálculo de grupos de orden pequeño.

La siguiente tabla resume los grupos distintos, salvo isomorfismo, que existen hasta orden 10:

| Orden | Comutativos | No comutativos | Total |
|-------|--|----------------|-------|
| 2 | \mathbb{Z}_2 | | 1 |
| 3 | \mathbb{Z}_3 | | 1 |
| 4 | $\mathbb{Z}_4, \mathbb{Z}_2 \times \mathbb{Z}_2$ | | 2 |
| 5 | \mathbb{Z}_5 | | 1 |
| 6 | \mathbb{Z}_6 | S_3 | 2 |
| 7 | \mathbb{Z}_7 | | 1 |
| 8 | $\mathbb{Z}_8, \mathbb{Z}_4 \times \mathbb{Z}_2, \mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_2$ | D_4, Q_2 | 5 |

| | | | |
|----|--|-------|---|
| 9 | $\mathbb{Z}_9, \mathbb{Z}_3 \times \mathbb{Z}_3$ | | 2 |
| 10 | \mathbb{Z}_{10} | D_5 | 2 |

Tabla 3.5. Grupos de orden pequeño distintos salvo isomorfismo.

Otro caso interesante por su sencillez, sería el cálculo de todos los grupos conmutativos de un orden pequeño, porque en tal caso disponemos de un teorema de estructura (cuya demostración podemos encontrar en el capítulo 5 de [15]), que nos resuelve el problema directamente:

Teorema 3.7. (Teorema de estructura para grupos abelianos finitos). Todo grupo abeliano finito es isomorfo a un producto cartesiano de grupos cíclicos cuyos órdenes son potencias de primos.

□

Bastaría con calcular la factorización del orden n de los grupos conmutativos que queremos buscar para tener el problema completamente resuelto (ejercicio 3.22).

9. EJERCICIOS

Ejercicio 3.1. Consideramos el grupo del ejercicio 2.1.

- a) Determinar el orden de todos los elementos.
- b) Calcular sus subgrupos.

□

Ejercicio 3.2. Consideramos el grupo del ejercicio 2.2.

- a) Calcular todos sus subgrupos.
- b) Determinar cuáles de sus subgrupos son normales y calcular el grupo cociente para cada uno de ellos.
- c) Determinar el orden de todos los elementos.

□

Ejercicio 3.3. Consideramos el grupo del ejercicio 2.3. Calcular:

- a) Todos los subgrupos de 6 elementos.
- b) El orden de cada elemento.
- c) Encontrar si existe algún subgrupo normal.

□

Ejercicio 3.4. Consideramos el grupo del ejercicio 2.4.

- a) Calcular los subgrupos de 4 elementos.
- b) Determinar el orden de todos sus elementos.

□

Ejercicio 3.5. Crear un programa que determine si un subconjunto es un subgrupo usando las ideas del ejemplo 3.1. Aplicarlo al subconjunto $A = \{1, 2, 3, 4, 5, 6\}$ del grupo G del ejercicio 2.2.

□

Ejercicio 3.6. Grupos cíclicos. Escribir un programa que determine si un grupo es cíclico, aplicarlo a $G = \mathbb{Z}_6, \mathbb{Z}_7$ o \mathbb{Z}_8 .

□

Ejercicio 3.7. Subgrupos cíclicos. Escribir un programa que determine todos los subgrupos cíclicos de un grupo finito.

□

Ejercicio 3.8. Dado un grupo cíclico G , encontrar los elementos $x \in G$ que verifican $\langle x \rangle = G$, para $G = \mathbb{Z}_6, \mathbb{Z}_7$ o \mathbb{Z}_8 con la operación suma.

□

Ejercicio 3.9. Sea p un número primo, el conjunto $\mathbb{Z}_p - \{0\}$ con la operación producto es un grupo abeliano. Determinar los subgrupos y los elementos, si existen, generadores de $\mathbb{Z}_7 - \{0\}$. ¿Es cíclico?

□

Ejercicio 3.10. Sea G un grupo y $Z(G)$ su centro (ver ejercicio 2.7.). Contestar a las siguientes preguntas:

- a) ¿Es $Z(G)$ subgrupo?
- b) Caso de ser subgrupo, ¿es normal?

□

Ejercicio 3.11. Usando cualquiera de las funciones que calcula los subgrupos de un grupo G , escribir un programa que calcule todos los subgrupos y compruebe cuáles de ellos son generados por un subconjunto fijo A de G .

□

Ejercicio 3.12. Calcular:

- a) Los subgrupos del grupo G del ejercicio 2.1. generados por: $\{a\}, \{b\}$ y $\{f, g\}$.
- b) Los subgrupos del grupo G del ejercicio 2.2. generados por: $\{2\}, \{3, 4\}$ y $\{2, 6, 7\}$.

□

Ejercicio 3.13. Consideremos $G = \{1, 2, 3, 4, 5, 6\}$ el grupo del ejemplo 3.9, esto es, con la operación dada por la tabla:

| * | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| 2 | 2 | 1 | 4 | 3 | 6 | 5 |
| 3 | 3 | 5 | 1 | 6 | 2 | 4 |
| 4 | 4 | 6 | 2 | 5 | 1 | 3 |
| 5 | 5 | 3 | 6 | 1 | 4 | 2 |
| 6 | 6 | 4 | 5 | 2 | 3 | 1 |

Tabla 3.5.

En el ejemplo 3.6. se vio que $H = \{1, 4, 5\}$ es subgrupo de G .

- Comprobar que H es normal.
- Calcular el grupo cociente G/H .

□

Ejercicio 3.14. Homomorfismos y subgrupos. Sean $(G_1, *_1)$ y $(G_2, *_2)$ dos grupos, y sea $f: G_1 \rightarrow G_2$ un homomorfismo de grupos, entonces (ver ejercicio 2.13.):

- $\ker(f)$ es un subgrupo normal de G_1 .
- f es inyectivo (monomorfismo) $\Leftrightarrow \ker(f) = \{e_1\}$, con e_1 el elemento neutro de G_1 .
- Si H es un subgrupo de G_1 entonces $f(H) = \{f(g) \mid g \in H\}$ es un subgrupo de G_2 , en particular $\text{Im}(f) = f(G_1)$ es un subgrupo de G_2 .
- Si I es un subgrupo de G_2 entonces $f^{-1}(I) = \{g \in G \mid f(g) \in I\}$ es un subgrupo de G_1 .
- Si f es sobreyectiva (epimorfismo) y H es un subgrupo normal de G_1 entonces $f(H)$ es un subgrupo normal de G_2 .

Comprobar los resultados anteriores para todos los homomorfismos que han ido apareciendo en los ejemplos y ejercicios del capítulo.

□

Ejercicio 3.15. Primer teorema de isomorfía para grupos. Sean $(G_1, *_1)$ y $(G_2, *_2)$ dos grupos, y sea $f: G_1 \rightarrow G_2$ un homomorfismo de grupos, entonces

$$G_1/\ker(f) \text{ es isomorfo a } \text{Im}(f).$$

Demostrar que el grupo cociente del ejercicio 3.13. y \mathbb{Z}_2 son isomorfos aplicando el primer teorema de isomorfía al homomorfismo de grupos del ejercicio 2.12.

□

Ejercicio 3.16. En el epígrafe 4 hemos usado la función KSubsets[] para calcular subconjuntos, esta función no es habitual en otros programas distintos de Mathematica, nos hemos limitado a su uso en el capítulo por razones de eficacia. Utilizando las mismas ideas que habitualmente se usan en la práctica para construir manualmente los subconjuntos de un conjunto de un determinado número de elementos, podemos programar una alternativa:

| PROGRAMA | COMENTARIOS |
|--|--|
| <pre>numero=NÚMERO DE ELEMENTOS DE LOS SUBCONJUNTOS; G=CONJUNTO;</pre> | Se indica el número de elementos que tendrán los subconjuntos y el conjunto. |
| <pre>subconjuntos=Table[{G CONTADORi }, {CONTADORi,Length[G]}];</pre> | Generamos todos los subconjuntos de 1 elemento. |
| <pre>Do[subconjuntostemp={}; Do[</pre> | Se van añadiendo elementos hasta llegar al valor que indique "numero". |

| | |
|--|-----------------------|
| <pre> conta=1; While[!TrueQ[G[[conta]]=== subconjuntos[[CONTADORj]] [[Length[subconjuntos[[CONTADORj]]]]]] , conta++); Gsin=Delete[G,Table[{CONTADOR}, {CONTADOR,conta}]]; Do[AppendTo[subconjuntostemp,Union[subconjuntos[[CONTADORj]], {Gsin[[CONTADORk]]}]]; ,{CONTADORk,Length[Gsin]}]; ,{CONTADORj,Length[subconjuntos]}]; subconjuntos=subconjuntostemp; ,{CONTADORi,numero-1}]; Subconjuntos </pre> | Salida de resultados. |
|--|-----------------------|

Programa 3.19. Subconjuntos de n elementos.

Ahora comprobemos su funcionamiento. Para ello consideremos un conjunto de 16 elementos y calculemos sus subconjuntos de 7 elementos utilizando el programa 3.19. y la función KSubsets[]. Medimos también en ambos casos el tiempo de proceso.

In[]:= **Timing[**
n= 7; **n=**
G={a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p};

⋮ ⋮

Out[] = {104.469 Second,
{ {a,b,c,d,e,f,g}, {a,b,c,d,e,f,h},
{a,b,c,d,e,f,i}, {a,b,c,d,e,f,j},
{a,b,c,d,e,f,k}, {a,b,c,d,e,f,l},
{a,b,c,d,e,f,m}, {a,b,c,d,e,f,n},
{a,b,c,d,e,f,o},

⋮ ⋮

In[]:= **Timing[Length[KSubsets[G,7]]]**

Out[] = {0.062 Second,
{ {a,b,c,d,e,f,g}, {a,b,c,d,e,f,h},
{a,b,c,d,e,f,i}, {a,b,c,d,e,f,j},
{a,b,c,d,e,f,k}, {a,b,c,d,e,f,l},
{a,b,c,d,e,f,m}, {a,b,c,d,e,f,n},
{a,b,c,d,e,f,o},

⋮ ⋮

Como era de esperar la función KSubsets[] es mucho más rápida.

La rutina está pensada para que sea efectiva en el cálculo de subconjuntos con un número de elementos menor o igual que el número de elementos del conjunto entre 2, pues el orden de un subgrupo propio debe dividir al del grupo y en consecuencia para los posibles órdenes es para los cuales el programa es más rápido.

- a) Programar una rutina alternativa que, por el contrario, sea más efectiva para un número de elementos mayor o igual que el número de elementos del conjunto entre 2.
- b) Evitar el uso de AppendTo[], generando de partida una lista subconjuntos de la longitud esperada y rellenándola después con los resultados obtenidos. Comprobar la eficacia respecto al programa 3.19.

□

Ejercicio 3.17. En el ejercicio 3.16. se construye una rutina que hace lo mismo que KSubsets[], calcular los subconjuntos de 4 elementos de los grupos de los ejercicios 2.1. y 2.2. usando KSubsets[] y el programa 3.19., comparar la eficacia en ambos casos.

□

Ejercicio 3.18. Comparar la efectividad de la función SUBGRUPOSN[] respecto a las demás aplicándola al grupo del ejemplo 3.7.

□

Ejercicio 3.19. $((\mathbb{B}_2)^n, \oplus)$ es un grupo conmutativo. Comprobarlo para $((\mathbb{B}_2)^4, \oplus)$ y calcular todos sus subgrupos. Obsérvese que para el cálculo de subgrupos de estos grupos basta con comprobar que el neutro pertenece y es cerrado para la operación, no es necesario comprobar que sea cerrado para simétricos pues el simétrico de todo elemento es el mismo.

□

Ejercicio 3.20. Podemos determinar los subgrupos normales del grupo G del ejemplo 3.7., aplicando la función 3.3. a todos los subgrupos del mismo (ver tabla 3.2.), resultando:

$$\{1, 4, 5, 8, 9, 12, 13, 16, 17, 20, 21, 24\} \text{ y } \{1, 8, 17, 24\}.$$

También podemos calcular el grupo cociente para cada uno de ellos usando 3.4. y obtenemos:

$$\{\{1, 8, 17, 24\}, \{2, 7, 18, 23\}, \{3, 11, 14, 22\}, \\ \{4, 12, 13, 21\}, \{5, 9, 16, 20\}, \{6, 10, 15, 19\}\}$$

y

$$\{\{1, 4, 5, 8, 9, 12, 13, 16, 17, 20, 21, 24\}, \{2, 3, 6, 7, 10, 11, 14, 15, 18, 19, 22, 23\}\}.$$

Crear una función que determine todos los subgrupos normales y los correspondientes grupos cociente de un grupo finito cualquiera. Aplicarla al grupo del ejemplo 3.7. y comprobar su correcto funcionamiento.

□

Ejercicio 3.21. Sea G el conjunto formado por los siguientes elementos: el día del mes en que nació módulo 5, el día de la semana en que cae su cumpleaños este año, su primer apellido, su DNI módulo 5 más 5, el número 11 y el 12.

- a) Encontrar una operación $*$ que dote a G de estructura de grupo no conmutativo y calcular sus subgrupos.
- b) Determinar cuáles son los subgrupos normales, elegir uno cualquiera H propio y determinar el grupo cociente.
- c) Calcular las clases laterales: $100 * H$ y $H * 101$.

□

Ejercicio 3.22. Escribir un programa que calcule todos los grupos abelianos distintos salvo isomorfismo de orden n .

□

4. EL GRUPO SIMÉTRICO

Los grupos simétricos son un caso particular de grupos finitos que, como podemos comprobar a lo largo del capítulo, son muy manejables por el ordenador. Comenzamos haciendo algunas valoraciones sobre las permutaciones y su posible implementación. Despu s analizamos con profundidad como utilizar las permutaciones y descomponerlas en composiciones de ciclos y trasposiciones, y terminamos calculando el subgrupo alternado. Muchos de los problemas aqu  expuestos vienen resueltos, y as  se explica en los casos donde se ha estimado oportuna su inclusi n, por funciones de Mathematica pertenecientes al paquete de Matem tica Discreta: <<Combinatorica` . Sin embargo y con el fin de dar respuestas did cticas, adem s de programables en otras plataformas o lenguajes, simult neamente vamos dando soluciones, que aunque escritas en el lenguaje de programaci n de Mathematica y menos efectivas, a diferencia de las funciones ya implementadas, son f cilmente trasladables a cualquier otro lenguaje, prim ndose ante todo la traducci n lo m s fiel y comprensible posible de los problemas cotidianos que suelen aparecer en relaci n a este tema.

Al igual que en el cap tulo anterior, en este tambi n llegaremos a puntos en la construcci n y desarrollo de algunos problemas, para los cuales el lenguaje de programaci n de Mathematica no es muy efectivo. Alcanzado este extremo ser a recomendable trasladarse a otras plataformas y por tanto indispensable tener una nocci n clara de cu les son los algoritmos necesarios y c mo pueden programarse.

En este cap tulo tambi n se hacen algunas consideraciones sobre la eficacia de los algoritmos propuestos, por lo general y como hasta ahora, se han hecho con una orientaci n ante todo did ctica. Aunque en algunas ocasiones, en el mismo texto se analizan y proponen posibles optimizaciones, no se ha realizado un estudio detallado sobre la efectividad de los mismos.

1. PERMUTACIONES

Dado un conjunto finito X , una permutaci n ser  una aplicaci n biyectiva de X en X . Dicho de otra forma, desde un punto de vista combinatorio, dado un conjunto de n elementos, llamaremos permutaci n simple (o sin repetici n) de n elementos a cada una de las listas que podamos formar que contenga a los n elementos y que cada una difiera de otra \'unicamente en el orden de colocaci n de los elementos. Si X tiene n elementos, el n mero de posibles aplicaciones biyectivas o permutaciones ser  $n!$.

Por ejemplo, todas las permutaciones para el conjunto $X = \{1, 2, 3\}$, serían:

$$\{1, 2, 3\}, \{1, 3, 2\}, \{2, 1, 3\}, \{2, 3, 1\}, \{3, 1, 2\} \text{ y } \{3, 2, 1\}.$$

Donde cada lista o conjunto anterior puede interpretarse como una aplicación biyectiva,

$$\begin{array}{llllll} \sigma_1 & \sigma_2 & \sigma_3 & \sigma_4 & \sigma_5 & \sigma_6 \\ X \rightarrow X & X \rightarrow X \\ 1 \mapsto 1 & 1 \mapsto 1 & 1 \mapsto 2 & 1 \mapsto 2 & 1 \mapsto 3 & 1 \mapsto 3 \\ 2 \mapsto 2 & 2 \mapsto 3 & 2 \mapsto 1 & 2 \mapsto 3 & 2 \mapsto 1 & 2 \mapsto 2 \\ 3 \mapsto 3 & 3 \mapsto 2 & 3 \mapsto 3 & 3 \mapsto 1 & 3 \mapsto 2 & 3 \mapsto 1 \end{array}$$

Un interesante ejercicio, consistiría en determinar algoritmos que calculen todas las posibles permutaciones de un conjunto de n elementos, en esta sección desarrollamos y explicamos tres, en la sección de ejercicios se propone alguno más, aunque la solución a este problema viene dada en Mathematica directamente por la función Permutations[] (véase el epígrafe 1.4.).

1.1. PRIMER ALGORITMO PARA EL CÁLCULO DE PERMUTACIONES

Una solución sería la propuesta en el siguiente programa:

| FUNCIÓN | COMENTARIOS |
|--|--|
| <pre>PERMUTACIONES[X_]:=Module[{n,temp,f,k,j,Z,i}, n=Length[X]; permutaciones={};</pre> | Se define la función “PERMUTACIONES” con un único argumento, el conjunto “X”. |
| <pre>Do[AppendTo[permutaciones,{X[[i]]}],{i,1,n}];</pre> | En primer lugar se añaden todos los conjuntos atómicos que aparecen a partir de los elementos de “X”. Si bien, luego serán eliminados. |
| <pre>While[Length[permutaciones[[1]]]<n,</pre> | Bucle principal. |
| <pre>temp=Length[permutaciones]; Do[Z=Complement[X,permutaciones[[k]]]; Do[AppendTo[permutaciones, Join[permutaciones[[k]],[Z[[j]]]]; ,{j,1,Length[Z]}]; ,{k,1,temp}];</pre> | Dos bucles anidados que realizan los cálculos. |
| <pre>permutaciones=Delete[permutaciones, Table[{f},{f,temp}]];];</pre> | Se eliminan las listas más pequeñas en longitud. |
| <pre>permutaciones</pre> | Se muestran los resultados. |

Función 4.1. Permutaciones I.

El funcionamiento del programa anterior se basa en la misma idea práctica con la que habitualmente construimos todas las permutaciones de un conjunto dado, por ejemplo, si quisieramos calcular las permutaciones del conjunto $X = \{1, 2, 3, 4\}$ lo haríamos siguiendo los siguientes pasos:

| PASO 1 | PASO 2 | PASO 3 | PASO 4 | Permutación |
|--------|--------|--------|--------|--------------|
| 1 | 2 | 3 | 4 | {1, 2, 3, 4} |
| | | 4 | 3 | {1, 2, 4, 3} |
| | 3 | 2 | 4 | {1, 3, 2, 4} |
| | | 4 | 2 | {1, 3, 4, 2} |
| | 4 | 2 | 3 | {1, 4, 2, 3} |
| | | 3 | 2 | {1, 4, 3, 2} |
| 2 | 1 | 3 | 4 | {2, 1, 3, 4} |
| | | 4 | 3 | {2, 1, 4, 3} |
| | 3 | 1 | 4 | {2, 3, 1, 4} |
| | | 4 | 1 | {1, 3, 4, 1} |
| | 4 | 1 | 3 | {2, 4, 1, 3} |
| | | 3 | 1 | {2, 4, 3, 1} |
| 3 | 1 | 2 | 4 | {3, 1, 2, 4} |
| | | 4 | 2 | {3, 1, 4, 2} |
| | 2 | 1 | 4 | {3, 2, 1, 4} |
| | | 4 | 1 | {3, 2, 4, 1} |
| | 4 | 1 | 2 | {3, 4, 1, 2} |
| | | 2 | 1 | {3, 4, 2, 1} |
| 4 | 1 | 2 | 3 | {4, 1, 2, 3} |
| | | 3 | 2 | {4, 1, 3, 2} |
| | 2 | 1 | 3 | {4, 2, 1, 3} |
| | | 3 | 1 | {4, 2, 3, 1} |
| | 3 | 1 | 3 | {4, 3, 1, 2} |
| | | 3 | 1 | {4, 3, 2, 1} |

Tabla 4.1. Cálculo de las permutaciones I.

Obsérvese que en el primer paso se toman todos los elementos del conjunto, ya que todos ellos pueden ser el primero de la permutación; en el segundo paso, para cada posibilidad, los tres restantes y así sucesivamente. El programa funciona de manera análoga, esto es:

- I. En un primer paso se define,

$$\text{permutaciones} = \{\{1\}, \{2\}, \{3\}, \{4\}\}.$$

- II. En el segundo paso añadimos un elemento distinto a los cuatro conjuntos que forman “permutaciones”, $\text{permutaciones} = \{\{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 1\}, \{2, 3\}, \{2, 4\}, \{3, 1\}, \{3, 2\}, \{3, 4\}, \{4, 1\}, \{4, 2\}, \{4, 3\}\}$ y se eliminan los conjuntos atómicos, quedando la lista “permutaciones” como sigue:

$$\{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 1\}, \{2, 3\}, \{2, 4\}, \{3, 1\}, \{3, 2\}, \{3, 4\}, \{4, 1\}, \{4, 2\}, \{4, 3\}\}.$$

- III. En el tercer paso se hace lo mismo que el paso 2, esto es, se añade otro elemento distinto a los conjuntos de dos elementos que en este momento forman la lista “permutaciones”, quedando:

$\{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 1\}, \{2, 3\}, \{2, 4\}, \{3, 1\}, \{3, 2\}, \{3, 4\}, \{4, 1\}, \{4, 2\}, \{4, 3\}, \{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 2\}, \{1, 3, 4\}, \{1, 4, 2\}, \{1, 4, 3\}, \{2, 1, 3\}, \{2, 1, 4\}, \{2, 3, 1\}, \{2, 3, 4\}, \{2, 4, 1\}, \{2, 4, 3\}, \{3, 1, 2\}, \{3, 1, 4\}, \{3, 2, 1\}, \{3, 2, 4\}, \{3, 4, 1\}, \{3, 4, 2\}, \{4, 1, 2\}, \{4, 1, 3\}, \{4, 2, 1\}, \{4, 2, 3\}, \{4, 3, 1\}, \{4, 3, 2\}\}$

como en II. eliminamos ahora los conjuntos de dos elementos y nos queda,

$\{\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 2\}, \{1, 3, 4\}, \{1, 4, 2\}, \{1, 4, 3\}, \{2, 1, 3\}, \{2, 1, 4\}, \{2, 3, 1\}, \{2, 3, 4\}, \{2, 4, 1\}, \{2, 4, 3\}, \{3, 1, 2\}, \{3, 1, 4\}, \{3, 2, 1\}, \{3, 2, 4\}, \{3, 4, 1\}, \{3, 4, 2\}, \{4, 1, 2\}, \{4, 1, 3\}, \{4, 2, 1\}, \{4, 2, 3\}, \{4, 3, 1\}, \{4, 3, 2\}\}.$

- IV. Por último y reiterando el proceso se añade el último elemento a cada conjunto de la lista “permutaciones” y se eliminan todos los de longitud 3,

$\{\{1, 2, 3, 4\}, \{1, 2, 4, 3\}, \{1, 3, 2, 4\}, \{1, 3, 4, 2\}, \{1, 4, 2, 3\}, \{1, 4, 3, 2\}, \{2, 1, 3, 4\}, \{2, 1, 4, 3\}, \{2, 3, 1, 4\}, \{2, 3, 4, 1\}, \{2, 4, 1, 3\}, \{2, 4, 3, 1\}, \{3, 1, 2, 4\}, \{3, 1, 4, 2\}, \{3, 2, 1, 4\}, \{3, 2, 4, 1\}, \{3, 4, 1, 2\}, \{3, 4, 2, 1\}, \{4, 1, 2, 3\}, \{4, 1, 3, 2\}, \{4, 2, 1, 3\}, \{4, 2, 3, 1\}, \{4, 3, 1, 2\}, \{4, 3, 2, 1\}\}.$

Observemos que en este algoritmo las permutaciones se obtienen en orden léxico-gráfico de izquierda a derecha, suponiendo la lista de partida como el primer elemento.

1.2. SEGUNDO ALGORITMO PARA EL CÁLCULO DE PERMUTACIONES

Otra solución más elegante podría ser la siguiente:

| FUNCIÓN | COMENTARIOS |
|--|---|
| PERMUTACIONES2[X_]:=Module[{n,K1,K2,i,j,k,h}, n=Length[X];Permutaciones={{}X [n]}]; | Se define la función “PERMUTACIONES” con un único argumento, el conjunto “X”. |
| Do[K1=Length[Permutaciones];K2=Length[Permutacione s[[1]]]; | Bucle principal. |
| Do[Do[AppendTo[Permutaciones, Insert[Permutaciones[[k]],X[[n-i]],j]]; ,{j,1,K2+1}]; ,{k,1,K1}]; | Algoritmo compuesto por dos bucles anidados que realizan los cálculos. |
| Permutaciones=Delete[Permutaciones, Table[{h},{h,K1}]]; ,{i,1,n-1}]; | Se eliminan las listas de longitud “K2”. |
| Permutaciones] | Se muestran los resultados. |

Función 4.2. Permutaciones II.

En este caso, el cálculo no es tan habitual en la práctica, un bucle recorre todos los elementos de la lista X , en el paso k -ésimo se calculan las permutaciones de los últimos k elementos, para ello se toman las permutaciones de los últimos $(k - 1)$ elementos y se construyen las nuevas permutaciones añadiendo el elemento nuevo en todas las posibles posiciones a las permutaciones que en ese momento se han calculado.

| PASO 1 | PASO 2 | PASO 3 | PASO 4 - Permutación |
|--------|--------|-----------|--|
| 4 | {3, 4} | {2, 3, 4} | {1, 2, 3, 4} {2, 1, 3, 4} {2, 3, 1, 4} {2, 3, 4, 1} |
| | | | {1, 3, 2, 4} {3, 1, 2, 4} {3, 2, 1, 4} {3, 2, 4, 1} |
| | | | {1, 3, 4, 2} {3, 1, 4, 2} {3, 4, 1, 2} {3, 4, 2, 1} |
| | | | {1, 2, 4, 3} {2, 1, 4, 3} {2, 4, 1, 3} {2, 4, 3, 1} |
| | | {4, 2, 3} | {1, 4, 2, 3} {4, 1, 2, 3} {4, 2, 1, 3} {4, 2, 3, 1} |
| | | | {1, 4, 3, 2} {4, 1, 3, 2} {4, 3, 1, 2} {4, 3, 2, 1} |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Tabla 4.2. Cálculo de las permutaciones II.

Veámoslo paso a paso con el mismo ejemplo de antes, si $X = \{1, 2, 3, 4\}$, se tomará:

$$\text{Permutaciones} = \{\{4\}\}$$

Y se calculará en 3 pasos:

- En un primer paso, se tomará el 3 y se calcularán todas las permutaciones del conjunto $\{3, 4\}$, para ello se toma el $\{4\}$ y se le añade el 3 por delante y por detrás, quedando,

$$\text{Permutaciones} = \{\{4\}, \{3, 4\}, \{4, 3\}\}.$$

Y se elimina $\{4\}$ de “Permutaciones”,

$$\text{Permutaciones} = \{\{3, 4\}, \{4, 3\}\}.$$

- II. Reiteramos el proceso y ahora tomamos el 2, generando para cada elemento de “Permutaciones”, otros tres, los que resultan de añadir 2 al principio, en medio y al final de la lista, esto es, quedará:

Permutaciones = $\{\{3, 4\}, \{4, 3\}, \{2, 3, 4\}, \{3, 2, 4\}, \{3, 4, 2\}, \{2, 4, 3\}, \{4, 2, 3\}, \{4, 3, 2\}\}$.

Se eliminan las listas de dos elementos y nos queda:

Permutaciones = $\{\{2, 3, 4\}, \{3, 2, 4\}, \{3, 4, 2\}, \{2, 4, 3\}, \{4, 2, 3\}, \{4, 3, 2\}\}$.

- III. Por último ahora generamos todas las permutaciones, para ellos añadimos 1 al principio, en las dos posiciones intermedias y al final de cada lista de las ya construidas:

Permutaciones = $\{\{2, 3, 4\}, \{3, 2, 4\}, \{3, 4, 2\}, \{2, 4, 3\}, \{4, 2, 3\}, \{4, 3, 2\}, \{1, 2, 3, 4\}, \{2, 1, 3, 4\}, \{2, 3, 1, 4\}, \{2, 3, 4, 1\}, \{1, 3, 2, 4\}, \{3, 1, 2, 4\}, \{3, 2, 1, 4\}, \{3, 2, 4, 1\}, \{1, 3, 4, 2\}, \{3, 1, 4, 2\}, \{3, 4, 1, 2\}, \{3, 4, 2, 1\}, \{1, 2, 4, 3\}, \{2, 1, 4, 3\}, \{2, 4, 1, 3\}, \{2, 4, 3, 1\}, \{1, 4, 2, 3\}, \{4, 1, 2, 3\}, \{4, 2, 1, 3\}, \{1, 4, 3, 2\}, \{4, 1, 3, 2\}, \{4, 3, 1, 2\}, \{4, 3, 2, 1\}\}$.

Se eliminan las listas de longitud tres y el cálculo ha concluido:

Permutaciones = $\{\{1, 2, 3, 4\}, \{2, 1, 3, 4\}, \{2, 3, 1, 4\}, \{2, 3, 4, 1\}, \{1, 3, 2, 4\}, \{3, 1, 2, 4\}, \{3, 2, 1, 4\}, \{3, 2, 4, 1\}, \{1, 3, 4, 2\}, \{3, 1, 4, 2\}, \{3, 4, 1, 2\}, \{3, 4, 2, 1\}, \{1, 2, 4, 3\}, \{2, 1, 4, 3\}, \{2, 4, 1, 3\}, \{2, 4, 3, 1\}, \{1, 4, 2, 3\}, \{4, 1, 2, 3\}, \{4, 2, 1, 3\}, \{4, 2, 3, 1\}, \{1, 4, 3, 2\}, \{4, 1, 3, 2\}, \{4, 3, 1, 2\}, \{4, 3, 2, 1\}\}$.

1.3. TERCER ALGORITMO PARA EL CÁLCULO DE PERMUTACIONES

Por último, la más simple, aunque menos comprensible, sería la siguiente solución basada en una función recursiva.

| FUNCIÓN | COMENTARIOS |
|---|--------------------|
| <pre>f[A]:= Which[Length[A]==1, Permutaciones2[A]={A};Permutaciones2[A], Length[A]!=1,Permutaciones2[A]={}; Do[G=Complement[A,{A[[i]]}];K=f[G]; Do[AppendTo[Permutaciones2[A], Join[K[[j]],[A[[i]]]]]; ,{j,Length[K]}];{i,Length[A]}]; Permutaciones2[A]];</pre> | Función recursiva. |

| | |
|---------------------------------------|--|
| PERMUTACIONES3[A_]:=Module[{}, | Se define la función “PERMUTACIONES3” con un único argumento, el conjunto “A”. |
| f[A]; | Se calculan las permutaciones. |
| Permutaciones2[A]] | Se muestran los resultados. |

Función 4.3. Permutaciones III.

El cálculo en este caso se sustenta en una función recursiva que calcula las permutaciones de los subconjuntos de “A”.

Lo analizamos para el mismo ejemplo, si $A = \{1, 2, 3, 4\}$, entonces $f[\{1, 2, 3, 4\}]$ calculará:

- I. Calculamos las permutaciones de $\{2, 3, 4\}$ con $f[\{2,3,4\}]$, y las unimos usando Join[] con {1}.
- II. Calculamos las permutaciones de $\{1, 3, 4\}$ con $f[\{1,3,4\}]$, y las unimos usando Join[] con {2}.
- III. Calculamos las permutaciones de $\{1, 2, 4\}$ con $f[\{1,2,4\}]$, y las unimos usando Join[] con {3}.
- IV. Calculamos las permutaciones de $\{1, 2, 3\}$ con $f[\{1,2,3\}]$, y las unimos usando Join[] con {4}.

Para terminar resaltar que $f[A]$ para “A” un conjunto de un único elemento es $\{A\}$.

1.4. FUNCIÓN DE MATHEMATICA

También disponemos de una función en Mathematica que directamente calcula las permutaciones:

Permutations[lista]

devuelve todas las permutaciones posibles que podamos realizar con los elementos del argumento “lista”.

Ejemplo 4.1. Calcular todas las permutaciones del conjunto $X = \{1, 2, 3, 4\}$.

Con la función 4.1.

In[]:= **PERMUTACIONES[X_]:=Module[{n,temporal,
temp,f,k,j,Z,i},
n=Length[X];**

⋮ ⋮

In[]:= **PERMUTACIONES[{1,2,3,4}]**

Out[]= $\{\{1, 2, 3, 4\}, \{1, 2, 4, 3\}, \{1, 3, 2, 4\}, \{1, 3, 4, 2\},$
 $\{1, 4, 2, 3\}, \{1, 4, 3, 2\}, \{2, 1, 3, 4\}, \{2, 1, 4, 3\},$
 $\{2, 3, 1, 4\}, \{2, 3, 4, 1\}, \{2, 4, 1, 3\}, \{2, 4, 3, 1\},$

$$\begin{aligned} & \{3, 1, 2, 4\}, \{3, 1, 4, 2\}, \{3, 2, 1, 4\}, \{3, 2, 4, 1\}, \\ & \{3, 4, 1, 2\}, \{3, 4, 2, 1\}, \{4, 1, 2, 3\}, \{4, 1, 3, 2\}, \\ & \{4, 2, 1, 3\}, \{4, 2, 3, 1\}, \{4, 3, 1, 2\}, \{4, 3, 2, 1\} \end{aligned}$$

Con la función 4.2.

*In[]]:= PERMUTACIONES2[X_]:=Module[{n,K1,K2,i,j,k,h},
n=Length[X];Permutaciones={X[[n]]};;*

⋮ ⋮

In[]]:= PERMUTACIONES2[{1,2,3,4}]

Out[]]= { {1, 2, 3, 4}, {2, 1, 3, 4}, {2, 3, 1, 4}, {2, 3, 4, 1},
 {1, 3, 2, 4}, {3, 1, 2, 4}, {3, 2, 1, 4}, {3, 2, 4, 1},
 {1, 3, 4, 2}, {3, 1, 4, 2}, {3, 4, 1, 2}, {3, 4, 2, 1},
 {1, 2, 4, 3}, {2, 1, 4, 3}, {2, 4, 1, 3}, {2, 4, 3, 1},
 {1, 4, 2, 3}, {4, 1, 2, 3}, {4, 2, 1, 3}, {4, 2, 3, 1},
 {1, 4, 3, 2}, {4, 1, 3, 2}, {4, 3, 1, 2}, {4, 3, 2, 1} }

Con la función 4.3.

In[]]:= f[A_]:=Which[

⋮ ⋮

In[]]:= PERMUTACIONES3[A_]:=Module[{},

⋮ ⋮

In[]]:= PERMUTACIONES3[{1,2,3,4}]

Out[]]= { {4, 3, 2, 1}, {3, 4, 2, 1}, {4, 2, 3, 1}, {2, 4, 3, 1},
 {3, 2, 4, 1}, {2, 3, 4, 1}, {4, 3, 1, 2}, {3, 4, 1, 2},
 {4, 1, 3, 2}, {1, 4, 3, 2}, {3, 1, 4, 2}, {1, 3, 4, 2},
 {4, 2, 1, 3}, {2, 4, 1, 3}, {4, 1, 2, 3}, {1, 4, 2, 3},
 {2, 1, 4, 3}, {1, 2, 4, 3}, {3, 2, 1, 4}, {2, 3, 1, 4},
 {3, 1, 2, 4}, {1, 3, 2, 4}, {2, 1, 3, 4}, {1, 2, 3, 4} }

Obsérvese que el orden en que aparecen las permutaciones es distinto como era de esperar, pues los algoritmos son totalmente distintos. Por último probamos a calcularlas con la función Permutations[]:

In[]]:= Permutations[{1,2,3,4}]

Out[]]= { {1, 2, 3, 4}, {1, 2, 4, 3}, {1, 3, 2, 4}, {1, 3, 4, 2},
 {1, 4, 2, 3}, {1, 4, 3, 2}, {2, 1, 3, 4}, {2, 1, 4, 3},
 {2, 3, 1, 4}, {2, 3, 4, 1}, {2, 4, 1, 3}, {2, 4, 3, 1},
 {3, 1, 2, 4}, {3, 1, 4, 2}, {3, 2, 1, 4}, {3, 2, 4, 1},
 {3, 4, 1, 2}, {3, 4, 2, 1}, {4, 1, 2, 3}, {4, 1, 3, 2},

$$\{4, 2, 1, 3\}, \{4, 2, 3, 1\}, \{4, 3, 1, 2\}, \{4, 3, 2, 1\}$$

Nótese que la salida es idéntica a la de la función PERMUTACIONES[] (4.1.), ambas se encuentran ordenadas léxico-gráficamente de izquierda a derecha a diferencia de la función PERMUTACIONES2[] (4.2.) y PERMUTACIONES3[] (4.3.). □

1.5. EFICACIA Y OPTIMIZACIÓN

Aunque desde un punto de vista práctico el primer algoritmo nos es más cercano, no es el más efectivo, para ello vamos a realizar una sencilla comprobación usando la función, Timing[].

Para realizar la prueba vamos a calcular las permutaciones de un conjunto de 8 elementos, por el primer algoritmo:

In]:= **Timing[PERMUTACIONES[{1,2,3,4,5,6,7,8}]]**[1]]

Out]= 362.766 Second

Utilizando el segundo:

In]:= **Timing[PERMUTACIONES2[{1,2,3,4,5,6,7,8}]]**[1]]

Out]= 83.625 Second

Usando el tercero:

In]:= **Timing[PERMUTACIONES3[{1,2,3,4,5,6,7,8}]]**[1]]

Out]= 72.953 Second

La efectividad de estos dos últimos algoritmos es bastante elevada pues observemos que si medimos el tiempo que tarda el ordenador en generar una lista de $8!$ elementos sin realizar ningún otro cálculo, el tiempo empleado es bastante grande¹⁵.

In]:= **Timing[list={};Do[AppendTo[list,i],{i,8!}]]**[1]]

Out]= 19.187 Second

También podemos medir el tiempo empleado por la función Permutations[]:

In]:= **Timing[Permutations[{1,2,3,4,5,6,7,8}]]**[1]]

Out]= 0.016 Second

¹⁵ Por las características propias del lenguaje de programación de Mathematica este cálculo es demasiado grande. En la mayoría de lenguajes de programación este cálculo sería mucho más corto.

A la vista de este resultado se podría pensar que los algoritmos usados son poco efectivos, en realidad, la función Permutations[] viene ya implementada en el programa y dicha implementación es a bajo nivel, por lo que no es comparable con los algoritmos que programamos, obsérvese como al programar algo mucho más simple como es la generación de una lista de longitud $8!$, el tiempo empleado es muy superior al de Permutations[] y en este código no hay algoritmo alguno que contemplar, lo cual no tendría sentido salvo por el extremo ya comentado, de hecho si cogemos las funciones PERMUTACIONES[] (4.1.) y PERMUTACIONES2[] (4.2.) y evitamos el uso de la función AppendTo[]:

| FUNCIÓN | COMENTARIOS |
|---|---|
| <pre>PERMUTACIONESv2[X_]:=Module[{n,temporal,temp,f,k,j,Z,i,tt,contk}, n=Length[X]; temporal={}; permutaciones= Table[{X[[i]]},{i,1,n}]; While[Length[permutaciones[[1]]]<n, temp=Length[permutaciones]; tt=Length[permutaciones[[1]]]; temporal=Table[0,{i,n!/(n-tt-1)!},{j,(tt+1)}]; contk=1; Do[Z=Complement[X,permutaciones[[k]]]; Do[temporal[[contk]]=Join[permutaciones[[k]],[Z[[j]]]]; contk++; ,{j,1,Length[Z]}]; ,{k,1,temp}]; permutaciones=temporal;]; permutaciones];</pre> | Igual a PERMUTACIONES[] (4.1.), evitando usar AppendTo[]. |

Función 4.1.bis. Permutaciones I.

| FUNCIÓN | COMENTARIOS |
|--|--|
| <pre>PERMUTACIONESv2[X_]:=Module[{n,K1,K2,i,j,k,h,temporal,contk}, n=Length[X]; Permutaciones={{X[[n]]}}; Do[K1=Length[Permutaciones]; K2=Length[Permutaciones[[1]]]; temporal=Table[0,{i,(K2+1)!},{j,K2+1}]; contk=1; Do[Do[</pre> | Igual a PERMUTACIONES2[] (4.2.), evitando usar AppendTo[]. |

```

temporal[[contk]]=Insert[
  Permutaciones[[k]],X[[n-i]],j];
  contk++;
,{j,1,K2+1};
,{k,1,K1}];
Permutaciones=temporal;
,{i,1,n-1}};
Permutaciones
]

```

Función 4.2.bis. Permutaciones II.

Encontraremos que estas nuevas funciones son sensiblemente más eficaces que las primeras:

In[]:= Timing[PERMUTACIONESv2[{1,2,3,4,5,6,7,8}]]

Out[] = 1.797 Second

In[]:= Timing[PERMUTACIONES2v2[{1,2,3,4,5,6,7,8}]]

Out[] = 0.516 Second

La razón por la cual un pequeño cambio en la programación (no en el algoritmo) provoca un cambio en la efectividad tan grande, entraría dentro del campo de las Ciencias Computacionales, del funcionamiento interno del Mathematica y no de la Matemática Discreta.

Aunque con motivaciones didácticas hemos desarrollado e implementado funciones que calculan las permutaciones, nos limitaremos en los programas posteriores a usar la función *Permutations[]* para evitar largos lapsos de tiempo en los cálculos, si bien, ha de tenerse en cuenta, que tal función no viene implementada en la mayoría de lenguajes. Por tanto, si quisieramos trasladar los algoritmos y cálculos posteriores a otros lenguajes, tendríamos que limitarnos a las soluciones de los epígrafes 1.1., 1.2. o 1.3.

2. EL GRUPO SIMÉTRICO

Para un entero $n > 1$, consideraremos el conjunto $X = \{1, 2, \dots, n\}$, entonces el grupo simétrico o de las permutaciones de n elementos, será el conjunto de todas las aplicaciones biyectivas de X en X (esto es, el conjunto de todas las permutaciones) con la operación composición:

$$S_n = \{f: X \rightarrow X \mid f \text{ es aplicación biyectiva}\}.$$

Los elementos de S_n o permutaciones, habitualmente las escribiremos así:

$$\sigma \in S_n, \quad \sigma = \begin{pmatrix} 1 & 2 & \dots & n \\ \sigma(1) & \sigma(2) & \dots & \sigma(n) \end{pmatrix}.$$

y hablaremos de composición o multiplicación de permutaciones para referirnos a la composición como aplicaciones de dos permutaciones σ y τ , además la denotaremos indistintamente por: $\sigma\tau$ o $\sigma \circ \tau$, y diremos que multiplicamos o componemos σ y τ . Existen varias posibilidades para manejar las permutaciones con el ordenador:

2.1. IMPLEMENTACIÓN DE PERMUTACIONES

2.1.1. ENTRADA DIRECTA DE PERMUTACIONES

Podemos introducir las permutaciones directamente como una aplicación entre conjuntos finitos como muestra el siguiente ejemplo.

Ejemplo 4.2. Consideremos $\sigma \in S_4$, $\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 4 \end{pmatrix}$ entonces podemos introducirla como sigue:

```
In[]:=          sigma[1]=2;
                sigma[2]=3;
                sigma[3]=1;
                sigma[4]=4;
```

o bien,

```
In[]:=          P={2,3,1,4};
                sigma2[x_]:=P[[x]];
```

□

Observemos que el ejemplo anterior muestra una fácil manera de intercambiar las dos visiones de permutación que se han comentado, esto es, entre una lista con todos los elementos del conjunto X y una aplicación biyectiva de X en X .

2.1.2. ENTRADA INDEXADA DE LAS PERMUTACIONES

Podemos calcular todas las permutaciones y representarlas de la forma habitual aprovechando la función¹⁶ PERMUTACIONES[] (4.1.) o bien Permutations[], para ello construimos la función:

¹⁶ Análogamente se podría trabajar con la función PERMUTACIONES2[] (4.2.), PERMUTACIONES3[] (4.3.) o cualquier otra que implementáramos, en tal caso el orden de las permutaciones es distinto y los índices también. Si bien, no es un grave problema el orden que asignemos a las permutaciones, para no dar lo lugar a ambigüedades supondremos que el conjunto de permutaciones vendrá ordenado por orden léxico-gráfico de izquierda a derecha, tal y como ocurre con las salidas de PERMUTACIONES[] (4.1.) y Permutations[], de esta forma la relación entre permutaciones e índices será biunívoca.

| FUNCIÓN | COMENTARIOS |
|--|---|
| <pre>S[n_]:=Table[MatrixForm[{Table[j,{j,n}],Permutations[Table[j,{j,n}]][[i]]}], {i,n!}] ,</pre> | Calculamos el grupo simétrico S_n para algún “n”. |

Función 4.4. El grupo simétrico S_n .

Función que calculará todas las permutaciones de S_n . Para referirnos a dichas permutaciones asignaremos un índice a cada una de ellas con la función:

| FUNCIÓN | COMENTARIOS |
|--|---|
| <pre>S[n_]:=...</pre> | Definimos la función 4.4. |
| <pre>NombreS[n_]:=Do[Print[\sigma_k, "=", S[n][[k]]], {k, 1, n!}]</pre> | Asignamos un índice a cada permutación. |

Función 4.5. Índices de las permutaciones.

Ejemplo 4.3. Calculamos todos los elementos de S_3 .

Definimos las funciones 4.4. y 4.5.

In[]:= $S[n_]:=Table[MatrixForm[\{Table[j,\{j,n\}],
 Permutations[Table[j,\{j,n\}]][[i]]\}],\{i,n!\}];$

In[]:= $\text{NombreS}[n_]:=Do[\text{Print}[\sigma_k, "=", S[n][[k]]], \{k, 1, n!\}];$

In[]:= $S[3]$

Out[]=

$$\left\{ \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}, \right. \\ \left. \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix} \right\}$$

In[]:= $\text{NombreS}[3]$

Out[]=

$$\sigma_1 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} \quad \sigma_2 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix} \quad \sigma_3 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}$$

$$\sigma_4 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix} \quad \sigma_5 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} \quad \sigma_6 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}$$

□

Podemos identificar una permutación concreta por el índice k que le asignamos,

$$S[n][[k]]$$

También podemos averiguar el índice asociado a una permutación cualquiera de S_n , damos dos definiciones alternativas de la misma función:

| FUNCIÓN | COMENTARIOS |
|---|--|
| <pre>Indice[permutacion_]:=Module[{i,n,j}, n=Length[permutacion];i=1; While[Permutations[Table[j,{j,1,n}]][[i]]!=permutacion , i++];]; </pre> | <p>El argumento será una lista.</p> <p>Se identifica la permutación.</p> |

Función 4.6. Índice de una permutación.

| FUNCIÓN | COMENTARIOS |
|--|---|
| <pre>Indice[permutacion_]:=Position[Permutations[Table[j, {j,1,Length[permutacion]}]],permutacion][[1]][[1]]</pre> | <p>Usamos directamente la función Position[].</p> |

Función 4.6.bis. Índice de una permutación.

Ejemplo 4.4. Identificar la permutación $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 1 & 5 \end{pmatrix}$.

Definimos la función 4.6.

In]:= **Indice[permutacion_]:=Module[{i,n,j},**

\vdots \vdots

In]:= **Indice[{2,3,4,1,5}]**

Out]:= 33

Y al revés, comprobamos:

In]:= **S[n_]:=Table[MatrixForm[{Table[j,{j,n}],
Permutations[Table[j,{j,n}]][[i]]}],{i,n!}];**

In]:= **S[5][[33]]**

Out:= $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 1 & 5 \end{pmatrix}$

□

Y así, podremos referirnos o aplicar cómodamente cualquier permutación. De esta forma, para calcular $\sigma_k(m)$ con $\sigma_k \in S_n$ y $m \in \{1, 2, \dots, n\}$, usaremos la función:

| FUNCIÓN | COMENTARIOS |
|--|--|
| <code>sigma[k_,n_][m_]:=Permutations[Table[j,{j,n}]][[k,m]]</code> | Definimos la permutación σ_k de S_n . |

Función 4.7. Permutación σ_k .

2.1.3. FUNCIONES DE MATHEMATICA

El paquete de Matemática Discreta que incorpora Mathematica y al cual accedemos escribiendo:

In[]:= <<Combinatorica`

incorpora las funciones:

- **SymmetricGroup[n]**, que funciona de forma similar a $S[n]$, si bien no usa la misma notación, en realidad sería equivalente a escribir:

Permutations[Table[i,{i,1,n}]

- **Index[]**, hace lo mismo que la función Indice[].

2.2. COMPOSICIÓN O MULTIPLICACIÓN DE PERMUTACIONES

También podemos componer o multiplicar las permutaciones, si $\sigma_k, \sigma_j \in S_n$ entonces la permutación $\sigma_k\sigma_j(m)$ vendrá dada (si usamos el método primero 2.1.1., la composición es evidente) por:

sigma[k,n][sigma[j,n][m]]

Habitualmente también nos interesará identificar a la permutación que resulta de componer otras dos, podemos programar una pequeña rutina que se encargue de esto:

| FUNCIÓN | COMENTARIOS |
|---|--|
| <code>sigma[k_,n_][m_]:=...</code> | Definimos la función 4.7. |
| <code>composicion[k_, j_,n_]:=Module[{t,m,z,tabla}, t=1; tabla=Table[sigma[k,n][sigma[j,n][m]],{m,1,n}];</code> | Llamaremos “composición” a la función y tendrá por argumentos a los índices de las permutaciones que vamos a multiplicar y el orden del grupo simétrico donde trabajamos |
| <code>While[tabla!=Permutations[Table[m,{m,n}]][[t]] ,t++];</code> | Bucle que recorre todas las permutaciones de S_n e identifica la composición. |
| <code>A=MatrixForm[{Table[z,{z,n}],tabla}];</code> | Se construye la matriz que representa a la permutación. |
| <code>Print[\sigma_k,\sigma_j,"=", \sigma_b, "=", A]</code> | Muestra por pantalla los resultados. |

| | |
|--|--|
| | |
|--|--|

Función 4.8. Producto de permutaciones.

Alternativamente, también podríamos programarla usando la función Position[], como hacemos en la siguiente función, que determina únicamente el índice de la permutación composición de las dos permutaciones de entrada:

| FUNCIÓN | COMENTARIOS |
|--|--|
| <code>sigma[k_,n_][m_]:=...</code> | Definimos la función 4.7. |
| <code>composicion2[k_,j_,n_]:=Module[{t,m,z},</code> <code>Position[Permutations[Table[m,{m,n}]],</code> <code>Table[sigma[k,n][sigma[j,n][m]},{m,1,n}]]][[1]][[1]]</code> <code>]</code> | Llamaremos “composición2” a la función y tendrá por argumentos a los índices de las permutaciones que vamos a multiplicar y el orden del grupo simétrico donde trabajamos. |
| <code>]</code> | Como salida tendrán el índice de la permutación composición. |
| <code>]</code> | |

Función 4.9. Producto de permutaciones 2.

Ejemplo 4.5. Calculamos la multiplicación de σ_2 y σ_4 de S_3 :

Definimos 4.7.:

In[]:= **sigma**[k_,n_][m_]:=Permutations[Table[j,{j,n}]][[k,m]];

Definimos 4.8.:

In[]:= **composicion**[k_, j_, n_]:=Module[{t,m,z},
t=1;

⋮ ⋮

Y la aplicamos a las permutaciones identificadas por los índices 2 y 4 de S_3 , obteniendo como resultado la permutación 6 de S_3 :

In[]:= **composicion**[2,4,3]

$$\text{Out}[] = \sigma_2\sigma_4 = \sigma_6 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}$$

Y con la función 4.9.:

In[]:= **composicion2**[k_,j_,n_]:=Module[{t,m,z},
Position[Permutations[Table[m,{m,n}]],
Table[sigma[k,n][sigma[j,n][m]},{m,1,n}]]][[1]][[1]]

|

In[]:= **composicion2**[2,4,3]

Out[] = 6

□

Para manejar los grupos simétricos igual que en el capítulo anterior, esto es, determinar la tabla de todos los productos posibles entre las permutaciones (tabla de operaciones), programamos la siguiente función:

| FUNCIÓN | COMENTARIOS |
|---|---|
| <code>sigma[k_,n_][m_]:=...</code> | Definimos la función 4.7. |
| <code>composicion2[k_,j_,n_]:=...</code> | Definimos la función 4.9. |
| <code>cuentas[n_]:= Module[{t,k,j,m}, tablacomp = Table[0, {k,n!}, {j, n!}]; Do[Do[t = composicion2[k,j,n]; tablacomp[[j, k]] = σ_t , {k, 1, n!}]; , {j, 1, n!}]; TableForm[tablacomp, TableHeadings -> {Table[σ_m, {m, n!}], Table[σ_m, {m, n!}]}, TableSpacing -> {2,2}]</code> | Función “cuentas” con un único argumento “n”. Bucles que recorren y calculan todos los posibles productos, |
| <code> </code> | Salida de resultados. |

Función 4.10. Tabla de productos entre permutaciones.

Ejemplo 4.6. Calculamos la tabla de todos los posibles productos de todas las permutaciones de S_3 , esto es, su tabla de operaciones para la operación composición del grupo de S_3 .

En primer lugar definimos las funciones 4.7. y 4.9.:

In[] := `sigma[k_,n_][m_]:=Permutations[Table[j,{j,n}]][[k,m]];`
In[] := `composicion2[k_,j_,n_]:=Module[{t,m,z},
Position[Permutations[Table[m,{m,n}]],
Table[sigma[k,n][sigma[j,n][m]],[{m,1,n}]]][[1]][[1]]]`

Introducimos la función 4.10.:

In[] := `cuentas[n_]:= Module[{t,k,j,m},
⋮
⋮`

Y calculamos la tabla :

In[] := `cuentas[3]`

Out[] =

| | | | | | |
|------------|------------|------------|------------|------------|------------|
| σ_1 | σ_2 | σ_3 | σ_4 | σ_5 | σ_6 |
| σ_1 | σ_1 | σ_2 | σ_3 | σ_4 | σ_5 |
| σ_2 | σ_2 | σ_1 | σ_4 | σ_3 | σ_6 |
| σ_3 | σ_3 | σ_5 | σ_1 | σ_6 | σ_2 |
| σ_4 | σ_4 | σ_6 | σ_2 | σ_5 | σ_1 |
| σ_5 | σ_5 | σ_3 | σ_6 | σ_1 | σ_4 |
| σ_6 | σ_6 | σ_4 | σ_5 | σ_2 | σ_3 |

□

La tabla que obtenemos con el programa anterior resulta muy útil desde el punto de vista práctico, para usarla con el ordenador previamente vamos a identificar cada permutación con su índice, esto es:

σ_i lo identificamos con i

entonces quedaría:

$$S_n = \{\sigma_1, \sigma_2, \dots, \sigma_{n!}\} = \{1, 2, \dots, n!\}$$

Y la operación vendría dada por una tabla, como se explica en la sección 1 del capítulo 3. Modificamos la función 4.10. para que nos devuelva la tabla de operaciones considerando esta identificación.

| FUNCIÓN | COMENTARIOS |
|---|--|
| <code>sigma[k_,n_][m]:=...</code> | Definimos la función 4.7. |
| <code>composicion2[k_,j_,n_]:=...</code> | Definimos la función 4.9. |
| <code>cuentas2[n_]:=Module[{k,j,m},</code> | Función “cuentas2” con un único argumento “n”. |
| <code>operacion=Table[composicion2[j,k,n],{k,n!},{j,n!}];</code> | Definimos la tabla operación. |
| <code>TableForm[operacion,</code> <code>TableHeadings→{Table[m,{m,n!}],</code> <code>Table[m,{m,n!}],TableSpacing→{2,2}]</code> | Salida de resultados. |
| <code>]</code> | |

Función 4.11. Tabla de productos entre permutaciones.

Ahora utilizando las herramientas y técnicas del capítulo anterior, podemos fácilmente comprobar que S_n es un grupo. Veámoslo como ejemplo para S_3 :

Ejemplo 4.7. Comprobar que S_3 es un grupo no conmutativo.

Hacemos la identificación entre índices y permutaciones, calculamos S_3 :

`In[1]:= S[n_]:=Table[MatrixForm[{Table[j,{j,n}],`
`Permutations[Table[j,{j,n}]][[i]]}],{i,n!}];`

`In[2]:= S[3]`

Out[] =

$$\left\{ \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}, \right.$$

$$\left. \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix} \right\}$$

Definimos la función 4.7., 4.9. y 4.11.

In[1]:= **G={1,2,3,4,5,6};**

```
In[7]:= sigma[k_,n_][m_]:=Permutations[Table[j,{j,n}]][[k,m]];
```

```
In[7]:= composicion2[k_,j_,n_]:=Module[{t,m,z},
  Position[Permutations[Table[m,{m,n}]],
  Table[sigma[k,n][sigma[j,n][m]},{m,1,n}]][[1]][[1]]
]
```

In]:= cuentas2[n]:=Module[{k,j,m},

• • • •

In[7]:= cuentas2[3]

Out[[]]=

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| 2 | 2 | 1 | 4 | 3 | 6 | 5 |
| 3 | 3 | 5 | 1 | 6 | 2 | 4 |
| 4 | 4 | 6 | 2 | 5 | 1 | 3 |
| 5 | 5 | 3 | 6 | 1 | 4 | 2 |
| 6 | 6 | 4 | 5 | 2 | 3 | 1 |

Introducimos la función 2.1.

In[*j*]:= **op[x,y]:=Module[{i,j},**

• • • • •

Ahora hacemos todas las comprobaciones. En primer lugar utilizamos el programa 2.2. para comprobar que es una operación interna.

In[7]:= **operacioninterna=True;**

• • • • •

Out[[*] = True*

Calculamos el elemento neutro con el programa 2.4.

In[]:= **ElementoNeutro="No existe";**

⋮ ⋮

Out[] = Elemento Neutro: 1

Por tanto el elemento neutro es $\sigma_1 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$. Con la función 2.6.

comprobamos la propiedad de elemento simétrico.

In[]:= **ELEMENTOSIMETRICO[G_,operacion_]:=Module[**

⋮ ⋮

In[]:= **ELEMENTOSIMETRICO[G,operacion]**

Out[] = Elemento Neutro: 1

1 2 3 4 5 6

Elemento 1 2 3 4 5 6

Simétrico 1 2 3 5 4 6

True

Con la función 2.8. determinamos si verifica la propiedad asociativa.

In[]:= **ASOCIATIVA[G_,operacion_]:=Module[**

⋮ ⋮

In[]:= **ASOCIATIVA[G,operacion]**

Out[] = True

Por último, comprobamos que no es commutativo con la función 2.9.

In[]:= **CONMUTATIVA[G_,operacion_]:=Module[**

⋮ ⋮

In[]:= **CONMUTATIVA[G,operacion]**

Out[] = False

□

En ocasiones necesitaremos calcular potencias de una permutación, generalizando lo visto en este epígrafe, resulta inmediata su resolución, por ejemplo $\sigma_i^k(x)$ para un entero positivo k , se calcularía:

```
In]:= m=x;
Do[m=sigma[i,n][m],{i,1,k}];
m
```

Aunque la inversa es una permutación que podemos calcular con las herramientas de los capítulos anteriores, por ejemplo con las funciones simetrico[] o ELEMENTOSIMETRICO[], no olvidemos que las permutaciones son aplicaciones biyectivas y por tanto, su inversa puede determinarse directamente fácilmente: si el grafo de una permutación $\sigma \in S_n$, como aplicación es $\{(i, \sigma(i)) \mid i = 1, 2, \dots, n\}$, el de su inversa σ^{-1} será $\{(\sigma(i), i) \mid i = 1, 2, \dots, n\}$. El grafo para una permutación cualquiera $\sigma_k \in S_n$ lo introduciremos así:

```
In]:= Grafo[k_,n_]:=Table[{j,sigma[k,n][j]},{j,n}];
```

Y el grafo de la permutación inversa:

```
In]:= Grafoinversa[k_,n_]:=Sort[Table[{sigma[k,n][j],j},{j,n}]];
```

Por último, si queremos conseguir la permutación inversa con la notación que habitualmente usamos:

```
In]:= Transpose[Grafoinversa[k,n]]//MatrixForm
```

2.3. CICLOS Y DESCOMPOSICIÓN EN PRODUCTO DE TRASPOSICIONES

Una permutación $\sigma \in S_n$ se dirá que es un ciclo si existe un subconjunto $\{i_1, i_2, \dots, i_k\} \subseteq X = \{1, 2, \dots, n\}$ para algún $1 \leq k \leq n$, tal que

$$\begin{cases} \sigma(i_k) = i_1, \\ \sigma(i_j) = i_{j+1}, & j = 1, 2, \dots, (k-1) \\ \sigma(t) = t, & \forall t \in X - \{i_1, i_2, \dots, i_k\}. \end{cases}$$

La denotaremos por

$$(i_1 \ i_2 \dots \ i_k)$$

k se dirá que es la longitud del ciclo y $\{i_1, i_2, \dots, i_k\} \subseteq X$ su soporte.

Dos ciclos $(i_1 \ i_2 \dots \ i_k), (j_1 \ j_2 \dots \ j_m) \in S_n$, diremos que son disjuntos si $\{i_1, i_2, \dots, i_k\} \cap \{j_1, j_2, \dots, j_m\} = \emptyset$. Es evidente que la composición de ciclos disjuntos es conmutativa.

Cualquier permutación puede escribirse como producto o composición de ciclos disjuntos, a esta notación la llamaremos notación cíclica. Si tenemos un ciclo de longitud 1, este será la aplicación identidad y no aparecerá en la notación cíclica.

Podemos comprobarlo con el siguiente programa:

| PROGRAMA | COMENTARIOS |
|---|---|
| <code>sigma[k_,n_ [m_]:=...</code> | Definimos la función 4.7. |
| <code>n=Nº DE ELEMENTOS DE LA PERMUTACIÓN;</code> <code>k=ÍNDICE DE LA PERMUTACIÓN;</code> | Se introduce “n” y “k” con los valores que deseemos. |
| <code>X=Table[i,{i,n}];</code> <code>ciclos={};</code> <code>While[X!={},</code> | Bucle que sigue dando vueltas mientras haya elementos en “X”. |
| <code> primero=X[[1]];</code> <code> imagen=primero;</code> <code> cadena={primero};</code> <code> While [sigma[k,n][imagen]≠primero,</code> <code> imagen=sigma[k,n][imagen];</code> <code> cadena=Append[cadena,imagen]</code> <code>];</code> <code> If[Length[cadena]>1,AppendTo[ciclos,cadena]];</code> | Bucle que calcula los ciclos y si no son triviales, los añade a “ciclos”. |
| <code>X=Complement[X,cadena];</code> | Se quita a “X” los elementos del soporte de cada ciclo. |
| <code>];</code> <code>ciclos</code> | Salida de resultados. |

Programa 4.12. Notación cíclica.

Su funcionamiento es similar al cálculo de la notación cíclica que habitualmente se hace de forma manual. Si consideramos permutaciones de n elementos ($X = \{1, 2, \dots, n\}$), y queremos encontrar la notación cíclica de una permutación cualquiera $\sigma \in S_n$, haremos la siguiente construcción:

- Tomamos un elemento cualquiera, por ejemplo $1 \in X$ y calculamos k_1 , el menor entero positivo tal que $\sigma^{k_1}(1) = 1$, entonces construimos:

$$c_0 = (1 \ \sigma(1) \ \sigma^2(1) \dots \ \sigma^{k_1-1}(1)), X_1 = X - \{1, \sigma(1), \sigma^2(1), \dots, \sigma^{k_1-1}(1)\};$$

- tomamos otro elemento $i_1 \in X_1 \subseteq X$ y de nuevo calculamos k_2 el menor entero positivo tal que $\sigma^{k_2}(i_1) = i_1$ y construimos:

$$c_1 = (i_1 \ \sigma(i_1) \ \sigma^2(i_1) \dots \ \sigma^{k_2-1}(i_1)), X_2 = X_1 - \{i_1, \sigma(i_1), \sigma^2(i_1), \dots, \sigma^{k_2-1}(i_1)\};$$

- reiteramos la construcción, $i_2 \in X_2$, k_3 el menor entero positivo tal que $\sigma^{k_3}(i_2) = i_2$, entonces:

$$c_2 = (i_2 \ \sigma(i_2) \ \sigma^2(i_2) \dots \ \sigma^{k_3-1}(i_2)), X_3 = X_2 - \{i_2, \sigma(i_2), \sigma^2(i_2), \dots, \sigma^{k_3-1}(i_2)\};$$

reiteramos el proceso t -veces, hasta que $X_{t+1} = \emptyset$, entonces tendremos una partición del conjunto $X = \{X_1, \dots, X_t\}$, compuesta por los soportes de los ciclos disjuntos de su notación cíclica, que vendrá dada por: $\sigma = c_0 \cdot c_1 \cdot \dots \cdot c_{t-1}$, y el orden en el que compongamos los ciclos no importa porque son disjuntos y en consecuencia conmutan.

También podemos conseguir la notación cíclica de una permutación con el paquete de Matemática Discreta: `Combinatorica`, que incorpora la función:

ToCycles[permutación]

cuyo argumento es una permutación y hace lo mismo. También disponemos la función inversa:

FromCycles[permutación_en_notación_cíclica].

Ejemplo 4.8. Descomponemos como producto de ciclos la permutación σ_8 de S_8 .

En primer lugar identificamos σ_8 :

```
In]:= S[n]:=Table[MatrixForm[{Table[j,{j,n}],  
Permutations[Table[j,{j,n}]][[i]]}],{i,n!}];  
  
In]:= S[4][[8]]
```

$$Out[] = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix}$$

Y ahora descomponemos en ciclos usando el programa 4.12.:

```
In]:= sigma[k_,n_][m_]:=Permutations[Table[j,{j,n}]][[k,m]];  
  
In]:= n=4;  
k=8;  
  
Out[] = {{1,2},{3,4}}
```

E interpretaremos que $\sigma_8 = (1\ 2)(3\ 4)$.

Análogamente podemos escribir:

```
In]:= <>Combinatorica`
```

In[]:= **ToCycles[{2,1,4,3}]**

Out[] = $\{\{2, 1\}, \{4, 3\}\}$

□

Un ciclo diremos que es una trasposición si su longitud es 2. Un ciclo $\sigma = (i_1 i_2 \dots i_k)$ se puede escribir como composición de trasposiciones como sigue:

$$\sigma = (i_1 i_2 \dots i_k) = (i_1 i_k) \dots (i_1 i_3)(i_1 i_2) \quad (1)$$

o bien,

$$\sigma = (i_1 i_2 \dots i_k) = (i_1 i_2)(i_2 i_3) \dots (i_{k-1} i_k) \quad (2)$$

En ambos casos escribimos el ciclo σ como producto de $(k - 1)$ trasposiciones, donde k es la longitud del ciclo. Es bastante inmediato realizar un pequeño programa que descomponga un ciclo en producto de trasposiciones como en (1) o como en (2). Lo hacemos para el primer caso y el segundo lo dejamos propuesto en el ejercicio 4.20.

| FUNCIÓN | COMENTARIOS |
|---|--|
| <code>trasposiciones[ciclo_]:=Module[{cadena,i}, cadena=""; Do[cadena=StringJoin[cadena,"(",ToString[ciclo[[1]]], " ",ToString[ciclo[[Length[ciclo]-i+1]]],")"] ,{i,1,Length[ciclo]-1}]; cadena]</code> | Se introduce el ciclo a descomponer |
| | Descompone el ciclo en producto de trasposiciones. |
| | Salida de resultados. |

Función 4.13. Descomposición de un ciclo en producto de trasposiciones.

Ejemplo 4.9. Escribimos como producto de trasposiciones el ciclo $\sigma = (2 \ 5 \ 3 \ 1)$.

Definimos la función 4.13.:

In[]:= **trasposiciones[ciclo_]:=Module[{cadena,i},**

⋮ ⋮

Y realizamos el calculo,

In[]:= **ciclo={2,5,3,1};
trasposiciones[ciclo]**

Out[] = $(2 \ 1)(2 \ 3)(2 \ 5)$

□

Ahora si unimos los programas 4.12. y 4.13, podemos construir otro que descomponga una permutación cualquiera como producto de trasposiciones:

| PROGRAMA | COMENTARIOS |
|--|--|
| <code>sigma[k_,n_][m_]:=...</code> | Definimos la función 4.7. |
| <code>S[n_]:=...</code> | Definimos la función 4.4. |
| <code>trasposiciones[ciclo_]:=...</code> | Definimos la función 4.13. |
| n=Nº DE ELEMENTOS DE LA PERMUTACIÓN; k=INDICE DE LA PERMUTACIÓN; | Se introduce el índice “k” de la permutación y número de elementos “n” |
| <code>X=Table[i,{i,n}];</code> ⋮ ⋮ <code>];</code> | Programa 4.12. |
| <code>descomposicion = "";</code> <code>Do[</code> <code>descomposicion = StringJoin[descomposicion,</code> <code>trasposiciones[ciclos[[i]]]]</code> <code>,{i, 1, Length[ciclos]}];</code> | Descompone en producto de trasposiciones cada ciclo. |
| <code>Print[σ_k, " = ", S[n][[k]], " = ", descomposicion]</code> | Salida de resultados. |

Programa 4.14. Descomposición de una permutación en producto de trasposiciones.

Ejemplo 4.10. Descomponemos como producto de trasposiciones la permutación $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 3 & 6 & 5 & 1 & 2 \end{pmatrix}$ utilizando 4.14.

Definimos las funciones previas que necesitamos, 4.6., 4.7. y 4.4.:

```

In]:= Indice[permutacion_]:=Module[{i,n,j},

$$\vdots \qquad \qquad \vdots$$


In]:= sigma[k_,n_][m_]:=Permutations[Table[j,{j,n}]][[k,m]];

In]:= S[n_]:=Table[MatrixForm[{Table[j,{j,n}],

$$\qquad \qquad \qquad \text{Permutations}[Table[j,{j,n}][[[i]]],{i,n!}];$$


```

Definimos la función 4.13.

En primer lugar identificamos el índice de la permutación usando la función 4.6.

```
In]:= Indice[{4,3,6,5,1,2}]
```

Y ahora descomponemos en producto de trasposiciones, usando el programa 4.14.

In[]:= n=6;

k=431;

⋮ ⋮

$$Out[J] = \sigma_{431} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 3 & 6 & 5 & 1 & 2 \end{pmatrix} = (1\ 5)(1\ 4)(2\ 6)(2\ 3)$$

□

3. EL SUBGRUPO ALTERNADO

En esta sección se analizará el problema de la paridad de una permutación y se determinará explícitamente el subgrupo alternado de cualquier grupo simétrico.

3.1. SIGNATURA Y PARIDAD

Dada una permutación $\sigma \in S_n$, $\sigma = \begin{pmatrix} 1 & 2 & \dots & n \\ \sigma(1) & \sigma(2) & \dots & \sigma(n) \end{pmatrix}$ tenemos una

inversión en σ si existen $i, j \in X = \{1, 2, \dots, n\}$ tales que $i < j$ y $\sigma(i) > \sigma(j)$. Denotaremos por $\gamma(\sigma)$ al número de inversiones que hay en σ , llamaremos signatura de σ a $(-1)^{\gamma(\sigma)}$ y escribiremos:

$$Sig(\sigma) = (-1)^{\gamma(\sigma)}.$$

Una permutación σ , se dirá que es par si el número de inversiones que hay en σ es par, esto es, si $Sig(\sigma) = 1$. Y se dirá que es impar si $\gamma(\sigma)$ es impar ($Sig(\sigma) = -1$).

Para resolver este problema directamente vamos a calcular el número de inversiones de una permutación. Desde un punto de vista práctico esta tarea puede resultar bastante engorrosa, sin embargo, desde el punto de vista computacional no lo es.

| PROGRAMA | COMENTARIOS |
|---|--|
| n=Nº DE ELEMENTOS DE LA PERMUTACIÓN; k=ÍNDICE DE LA PERMUTACIÓN; | Se introduce el índice “k” de la permutación y número de elementos “n” |
| permutacion=Table[sigma[k,n][i],{i,1,n}]; | Opcionalmente podemos directamente introducir la permutación que queramos y no necesariamente referirnos a ella por “n” y “k”. |
| inversion=0; | De partida suponemos que no hay ninguna inversión. |
| Do[Do[If[i<j && permutacion[[j]]< permutacion[[i]] ,inversion++]; ,{j,i,Length[permutacion]}]; | Se calcula el número de inversiones. |

| | |
|---|-----------------------|
| <code>,{i,1,Length[permutacion]}];</code> | |
| inversion | Salida de resultados. |

Programa 4.15. Inversiones.

Dentro del paquete, `Combinatorica`, encontramos

Inversions[permutación]

que también calcula el número de inversiones de una permutación.

Ejemplo 4.11. Calculamos el número de inversiones de la permutación

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 6 & 8 & 5 & 2 & 3 & 4 & 7 & 1 \end{pmatrix}.$$

Usando el programa 4.15. podemos calcularlas:

In[]:= **permutacion={6,8,5,2,3,4,7,1};**
 ⋮ ⋮

Out[]= 19

O bien,

In[]:= **<<Combinatorica`**
In[]:= **Inversions[{6,8,5,2,3,4,7,1}]**
Out[]= 19

□

Y conociendo el número de inversiones, es inmediato calcular la signatura de una permutación.

Ejemplo 4.12. Calculamos la signatura de la permutación

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 6 & 8 & 5 & 2 & 3 & 4 & 7 & 1 \end{pmatrix}.$$

Usamos 4.15.

In[]:= **permutacion={6,8,5,2,3,4,7,1};**
 ⋮ ⋮
Out[]= 19

In]:= (-1)^%

Out]= -1

□

Atendiendo a los siguientes resultados podemos (no son difíciles de demostrar) desde un punto de vista práctico, simplificar considerablemente el cálculo de la signatura de una permutación, si bien desde el punto de vista computacional no es así, como comprobaremos con los programas y ejemplos que a continuación se muestran.

Proposición 4.1. Las trasposiciones son impares.

□

Proposición 4.2. $\text{Sig}(\sigma\tau) = \text{Sig}(\sigma)\text{Sig}(\tau)$.

□

Dada una permutación cualquiera $\sigma \in S_n$, si llamamos $N(\sigma)$ al número de trasposiciones en que podemos descomponer σ , entonces $\text{Sig}(\sigma) = (-1)^{N(\sigma)}$. Luego si α es un ciclo de longitud k , su signatura será $\text{Sig}(\alpha) = (-1)^{k-1}$. Por tanto, modificando el programa 4.12, también podemos determinar la signatura de una permutación cualquiera.

| PROGRAMA | COMENTARIOS |
|---|---|
| <code>sigma[k_,n_][m_]:=...</code> | Definimos la función 4.7. |
| <code>n=Nº DE ELEMENTOS DE LA PERMUTACIÓN;</code> <code>k=ÍNDICE DE LA PERMUTACIÓN;</code> | Se introduce el índice “k” de la permutación y número de elementos “n” |
| <code>X=Table[i,{i,n}];</code> ⋮ ⋮ ⋮ | Programa 4.12. |
| <code>signatura=0;</code> <code>Do[</code> <code>signatura=signatura+Length[ciclos[[i]]]-1</code> <code>,{i,1,Length[ciclos]}]</code> <code>signatura=(-1)^signatura</code> | Cuenta el número de trasposiciones en que descompone la permutación. Salida de resultados. |

Programa 4.16. Cálculo de la signatura o paridad de una permutación.

Usando el paquete, `Combinatorica`, tendremos la función:

SignaturePermutation[permutación]

que nos proporcionará directamente la signatura de “permutación”.

Ejemplo 4.13. Calcular la signatura de la permutación $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 3 & 6 & 5 & 1 & 2 \end{pmatrix}$.

Primero definimos las funciones previas que necesitamos, 4.6 y 4.7.:

In]:= Indice[permutacion_]:=Module[{i,n,j},

⋮ ⋮

In[]:= sigma[k_,n_][m_]:=Permutations[Table[j,{j,n}]][[k,m]];

Identificamos el índice de la permutación usando la función 4.6. y calculamos su signatura usando el programa 4.16.

*In[]:= perm={4,3,6,5,1,2};
n=Length[perm];
k= Indice[perm];*

⋮ ⋮

Out[] = 1

O directamente,

In[]:= <<Combinatorica`

In[]:= SignaturePermutation[{4,3,6,5,1,2}]

Out[] = 1

□

3.2. EL SUBGRUPO ALTERNADO

Es inmediato comprobar usando que $Sig(\sigma\tau) = Sig(\sigma)Sig(\tau)$ que si σ y τ son dos permutaciones pares de S_n , entonces $\sigma\tau$, σ^{-1} y τ^{-1} también lo son, además es evidente que la permutación identidad carece de inversiones y en consecuencia también es par. Por tanto el subconjunto A_n de todas las permutaciones pares es un subgrupo de S_n , al que llamaremos subgrupo alternado de S_n . De hecho, A_n es un subgrupo normal de S_n y $|A_n| = |S_n|/2$.

Calculamos todas las permutaciones pares de S_n , esto es, el subgrupo alternado A_n :

| FUNCIÓN | COMENTARIOS |
|--|---|
| <i>sigma[k_,n_][m_]:=...</i> | Definimos la función 4.7. |
| <i>S[n_]:=...</i> | Definimos la función 4.4. |
| <i>A[n_]:=Module[{Alternado,Sn,imagen,primero, cadena,t,k,ciclos,añadir,l,i}, Alternado={}; Sn=S[n];</i> | Función “A” con un único argumento “n” que calculará A_n . |
| <i>Do[</i> | Bucle que recorre todas las permutaciones, comprobando cuáles de ellas son pares. |
| <i>X=Table[i,{i,n}]; ciclos={};</i> | Programa 4.12. que determina la |

| | |
|---|---|
| <pre> While[X≠{}]; primero=X[[1]]; imagen=primero; cadena={primero}; While[σ[k,n][imagen]≠primero, imagen=σ[k,n][imagen]; cadena=Append[cadena,imagen]]; If[Length[cadena]>1, AppendTo[ciclos,cadena]; X=Complement[X,cadena];] </pre> | notación cíclica de la permutación |
| <pre> signatura=0; Do[signatura=signatura+Length[ciclos[[i]]]-1 ,{i,1,Length[ciclos]}]; If[(-1)^signatura==1, AppendTo[Alternado,Sn[[k]]]; ,{k,1,n!}]; </pre> | Determinamos la signatura observando la notación cíclica, si es par, añadimos la permutación a "Alternado". |
| Alternado | Salida de resultados. |
|] | |

Función 4.17. Cálculo del subgrupo alternado por descomposición en producto de ciclos.

En la función 4.17. comprobamos la paridad de cada permutación descomponiéndolas en producto de ciclos (notación cíclica) y calculando la paridad de cada ciclo. Podemos hacer el mismo cálculo de otra forma, directamente contando el número de inversiones de cada permutación:

| PROGRAMA | COMENTARIOS |
|---|---|
| S[n]:=... | Definimos la función 4.4. |
| n=NUMERO DE ELEMENTOS DE LAS PERMUTACIONES; | Introducimos "n" para calcular A_n . |
| Alternado={}; Sn=S[n]; Permutaciones=Permutations[Table[j,{j,n}]]; | Calculamos S_n . |
| Do[| |
| permutacion=Permutaciones[[k]]; inversion=0; Do[Do[If[i<j && permutacion[[j]]<permutacion[[i]], inversion++]; ,{j,i,n}]; ,{i,1,n}]; inversion; If[Mod[inversion,2]==0, AppendTo[Alternado,Sn[[k]]]; | Se calcula el número de inversiones de cada permutación y se comprueba si es par. Bucle que recorre todas las permutaciones, comprobando cuáles son pares. |
| ,{k,1,n!}] | |
| Alternado | Salida de resultados. |

Programa 4.18. Cálculo del subgrupo alternado directamente contando el número de inversiones.

Por último podemos usar el paquete, `Combinatorica`, y directamente calcularlo:

AlternatingGroup[n]

Y así también obtendremos el subgrupo alternado A_n , si bien la notación difiere de la de la función A[] y del programa 4.18.

Ejemplo 4.14. Calculamos A_4 .

Definiremos las funciones que necesitamos previamente (4.4. y 4.7.) e introducimos la función 4.17.:

```

In]:= sigma[k_,n_][m_]:=Permutations[Table[j,{j,n}]][[k,m]];
In]:= S[n_]:=Table[MatrixForm[{Table[j,{j,n}],
Permutations[Table[j,{j,n}]][[i]],{i,n!}}];
In]:= A[n_]:=Module[{Alternado,Sn,imagen,primero,cadena},
⋮
⋮
In]:= A[4]

Out]= { $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}$ ,  $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 4 & 2 \end{pmatrix}$ ,  $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 2 & 3 \end{pmatrix}$ ,  $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix}$ ,
 $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 4 \end{pmatrix}$ ,  $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 3 & 1 \end{pmatrix}$ ,  $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 2 & 4 \end{pmatrix}$ ,  $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 4 & 1 \end{pmatrix}$ ,
 $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix}$ ,  $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 3 & 2 \end{pmatrix}$ ,  $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 2 & 1 & 3 \end{pmatrix}$ ,  $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{pmatrix}$ }

```

O bien usando el programa 4.18.,

```

In]:= n=4;
⋮
⋮
Out]= { $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}$ ,  $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 4 & 2 \end{pmatrix}$ ,  $\begin{pmatrix} 2 & 3 & 4 & 1 \\ 1 & 4 & 2 & 3 \end{pmatrix}$ ,  $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix}$ ,
 $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 4 \end{pmatrix}$ ,  $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 3 & 1 \end{pmatrix}$ ,  $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 2 & 4 \end{pmatrix}$ ,  $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 4 & 1 \end{pmatrix}$ ,
 $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix}$ ,  $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 3 & 2 \end{pmatrix}$ ,  $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 2 & 1 & 3 \end{pmatrix}$ ,  $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{pmatrix}$ }

```

Por último con el paquete de funciones:

```
In]:= <<Combinatorica`  

In]:= AlternatingGroup[4]  

Out]= {{1, 2, 3, 4}, {1, 3, 4, 2}, {1, 4, 2, 3}, {2, 1, 4, 3},  

{2, 3, 1, 4}, {2, 4, 3, 1}, {3, 1, 2, 4}, {3, 2, 4, 1},  

{3, 4, 1, 2}, {4, 1, 3, 2}, {4, 2, 1, 3}, {4, 3, 2, 1}}
```

□

Para concluir comprobemos cuál de los dos métodos que hemos implementado es más efectivo, para ello determinamos el tiempo necesario para calcular A_7 con ambos algoritmos (téngase en cuenta que las mediciones son particulares a una máquina concreta):

```
In]:= Timing[A[7]][[1]]  

Out]= 87.125 Second  

In]:= Timing[  

n=7;  

: : :  

][[1]]  

Out]= 12.438 Second
```

Como ya comentábamos computacionalmente es más rápido comprobar la paridad de cada permutación contando el número de inversiones, sin embargo cuando hacemos los cálculos manualmente es al revés, nos resultará más cómodo buscar la notación cíclica.

Por último y aunque no debe compararse con los métodos anteriores, también podemos ver el tiempo empleado por AlternatingGroup[]:

```
In]:= Timing[AlternatingGroup[7]][[1]]  

Out]= 1.072 Second
```

Ejemplo 4.15. Comprobamos que A_6 es un subgrupo normal de S_6 y determinamos la efectividad de las rutinas utilizadas en la resolución de este problema.

En primer lugar calculamos S_6 y su tabla de operaciones, para ellos usaremos las funciones 4.7., 4.9. y 4.11.

```
In]:= sigma[k_,n_][m_]:=Permutations[Table[j,{j,n}]][[k,m]];  

In]:= composicion2[k_,j_,n_]:=Module[{t,m,z},  

Position[Permutations[Table[m,{m,n}]],  

Table[sigma[k,n][sigma[j,n][m]],[{m,1,n}]]][[1]][[1]]]
```

```

]

In[]:= cuentas2[n_]:=Module[{k,j,m},
  :
  :

In[]:= n=6;
tiempo=TimeUsed[];
G=Table[i,{i,n!}];
cuentas2[n];
operacion=Table[composicion2[j,k,n],{k,n!},{j,n!}];
Print[TimeUsed[]-tiempo];

Out[=] 2862.17

```

Necesitaremos casi una hora para calcular la tabla de operaciones de S_6 , se trata de una tabla de 720 filas por 720 columnas, por lo que el ordenador deberá de realizar 518400 composiciones de permutaciones de 6 elementos para calcularla.

Ahora calculamos A_6 , para ello consideramos el programa 4.18. y hacemos unas ligeras modificaciones para obtener todos los elementos de A_6 identificados por los índices que proporciona la función 4.6.:

| PROGRAMA | COMENTARIOS |
|--|---|
| Indice[permutacion]:=... n=NUMERO DE ELEMENTOS DE LAS PERMUTACIONES; | Definimos la función 4.6. Introducimos "n" para calcular A_n . |
| Alternado={}; Permutaciones= Permutations[Table[i,{i,n}]]; | |
| Do[permutacion=Permutaciones[[k]]; inversion=0; Do[Do[If[i<j && permutacion[[j]]< permutacion[[i]], inversion++]; ,{j,i,n}]; ,{i,1,n}]; inversion; If[Mod[inversion,2]==0, AppendTo[Alternado,Indice[permutacion]]]; ,{k,1,n!}] Alternado | Se calcula el número de inversiones de cada permutación y se comprueba si es par. Bucle que recorre todas las permutaciones, comprobando cuáles son pares. |
| | Salida de resultados. |

Programa 4.18.bis. Cálculo del subgrupo alternado directamente contando el número de inversiones.

```

In[]:= Indice[permutacion_]:=Module[{i,n,j},
  :
  :

In[]:= tiempo=TimeUsed[];

```

```
n=6;
Alternado={};
```

```
⋮ ⋮
```

```
Print[TimeUsed[]-tiempo];
Alternado
```

```
Out[] = 0.265
```

```
Out[] = {1, 4, 5, 8, 9, 12, 13, 16, 17, 20, 21, 24, 26, 27, 30, 31, 34,
35, 38, 39, 42, 43, 46, 47, 49, 52, 53, 56, 57, 60, 61, 64, 65,
68, 69, 72, 74, 75, 78, 79, 82, 83, 86, 87, 90, 91, 94, 95, 97,
100, 101, 104, 105, 108, 109, 112, 113, 116, 117, 120, 122,
123, 126, 127, 130, 131, 134, 135, 138, 139, 142, 143, 145,
148, 149, 152, 153, 156, 157, 160, 161, 164, 165, 168, 170,
171, 174, 175, 178, 179, 182, 183, 186, 187, 190, 191, 193,
196, 197, 200, 201, 204, 205, 208, 209, 212, 213, 216, 218,
219, 222, 223, 226, 227, 230, 231, 234, 235, 238, 239, 241,
244, 245, 248, 249, 252, 253, 256, 257, 260, 261, 264, 266,
267, 270, 271, 274, 275, 278, 279, 282, 283, 286, 287, 289,
292, 293, 296, 297, 300, 301, 304, 305, 308, 309, 312, 314,
315, 318, 319, 322, 323, 326, 327, 330, 331, 334, 335, 337,
340, 341, 344, 345, 348, 349, 352, 353, 356, 357, 360, 362,
363, 366, 367, 370, 371, 374, 375, 378, 379, 382, 383, 385,
388, 389, 392, 393, 396, 397, 400, 401, 404, 405, 408, 410,
411, 414, 415, 418, 419, 422, 423, 426, 427, 430, 431, 433,
436, 437, 440, 441, 444, 445, 448, 449, 452, 453, 456, 458,
459, 462, 463, 466, 467, 470, 471, 474, 475, 478, 479, 481,
484, 485, 488, 489, 492, 493, 496, 497, 500, 501, 504, 506,
507, 510, 511, 514, 515, 518, 519, 522, 523, 526, 527, 529,
532, 533, 536, 537, 540, 541, 544, 545, 548, 549, 552, 554,
555, 558, 559, 562, 563, 566, 567, 570, 571, 574, 575, 577,
580, 581, 584, 585, 588, 589, 592, 593, 596, 597, 600, 602,
603, 606, 607, 610, 611, 614, 615, 618, 619, 622, 623, 625,
628, 629, 632, 633, 636, 637, 640, 641, 644, 645, 648, 650,
651, 654, 655, 658, 659, 662, 663, 666, 667, 670, 671, 673,
676, 677, 680, 681, 684, 685, 688, 689, 692, 693, 696, 698,
699, 702, 703, 706, 707, 710, 711, 714, 715, 718, 719}
```

El cálculo de A_6 se realiza en menos de un segundo. Por último comprobemos que en efecto A_6 es un subgrupo normal de S_6 . Para ello definimos previamente las funciones y programas 2.1., 2.4., 2.7., 3.1. y 3.3.

```
In[]:= op[x_,y_]:=Module[{i,j},
```

```
⋮ ⋮
```

```
In[]:= ElementoNeutro="No existe";
```

```
⋮ ⋮
```

```

Out[]:=      Elemento Neutro: 1
In[]:=      simetrico[x_]:=Module[{simetrico,CONTADORi},
                                :           :
In[]:=      SUBGRUPO[H_]:=Module[
                                :           :
In[]:=      NORMAL[H_]:=Module[{normal},
                                :           :

```

Y ahora hacemos las comprobaciones, medimos además el tiempo de proceso necesario para el cálculo:

```

In[]:=      Timing[SUBGRUPO[Alternado]]
Out[]:=     {7063.78 Second, True}
In[]:=      Timing[NORMAL[Alternado]]
Out[]:=     {81.531 Second, True}

```

Para concluir, le pedimos a Mathematica que nos muestre la cantidad de memoria que está utilizando:

```

In[]:=      MemoryInUse[]
Out[]:=     12006680

```

□

Prácticamente la totalidad de la memoria usada en el ejemplo anterior es usada para almacenar la tabla de operaciones del grupo, aunque para S_6 no supone un problema (aproximadamente medio millón de datos), en general para S_n la tabla de operaciones estaría compuesta por $(n!)^2$ datos, por ejemplo para $n = 10$, necesitaríamos almacenar 13168189440000 datos y las limitaciones de memoria de los ordenadores actuales hacen imposible almacenar este volumen de datos. En tal caso, tendríamos que evitar el cálculo de la tabla de operaciones ya que no podemos almacenarla, en principio esto no supone un problema porque se pueden realizar las mismas comprobaciones sin necesidad de calcular previamente dicha tabla. Este sacrificio dará lugar a un comportamiento peor de los métodos analizados desde el punto de vista de la velocidad de cálculo (al no disponer de la tabla, no podremos consultarla y nos veremos obligados a calcular la composición de permutaciones siempre que sea necesario). Por ello debemos siempre establecer un adecuado equilibrio entre la cantidad de memoria a usar y el tiempo necesario para dar una solución al problema, que nos lleve a una resolución satisfactoria del mismo. En el estudio

del grupo simétrico también existe la posibilidad de hacer un estudio más amplio sobre posibles optimizaciones, al modo en que se hizo en capítulo 3 para el cálculo de subgrupos.

4. EL SUBGRUPO DIÉDRICO

El grupo diédrico D_n se define como el conjunto de movimientos rígidos del plano que llevan un polígono de n lados en el mismo. Los movimientos rígidos de un plano pueden ser: traslaciones, rotaciones respecto a un punto (centro de rotación) y simetrías respecto de una recta (eje de simetría). Para $n = 4$, D_4 sería el conjunto formado por los siguientes movimientos rígidos, que además podemos identificar con permutaciones de los vértices del cuadrado:

| | | | |
|--|--|---|--|
| <p>Rotaciones:</p> <p>Giro de 90°</p> <p>Giro de 180°</p> | $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix}$ | <p>Giro de 270°</p> <p>Giro de 360° (identidad)</p> | $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 2 & 3 \end{pmatrix}$ |
| <p>Simetrías:</p> <p>Simetrías respecto a la horizontal y la vertical</p> | $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{pmatrix}$ | <p>Simetrías respecto a las diagonales</p> | $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix}$ |
| | $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 3 & 2 \end{pmatrix}$ | | $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 4 \end{pmatrix}$ |

Tabla 4.3. Elementos de D_4 .

En general podemos decir que D_n tendrá exactamente $2n$ elementos (n giros y n simetrías) y es evidente que la composición de dos de estos movimientos da lugar a otro elemento de D_n . Visto como subconjunto de S_n es cerrado para simétricos y contiene al neutro (el giro de 360°), por tanto D_n es subgrupo de S_n . Además obsérvese que el ciclo $\sigma = (1\ 2\ \dots\ n)$ es uno de los giros, es un elemento de orden n , y que $\{\sigma^k \mid k = 1, 2, \dots, n\}$ es el conjunto de todos los giros de D_n , cada simetría es de orden 2 y además dada una simetría τ podemos obtener el resto componiendo con los distintos giros: $\tau \circ \sigma$, $\tau \circ \sigma^2, \dots, \tau \circ \sigma^{n-1}$. Por tanto, podemos construir D_n fácilmente con Mathematica:

| FUNCIÓN | COMENTARIOS |
|--|--|
| <code>Indice[permutation_]:=...</code> | Definimos la función 4.6.bis. |
| <code>sigma[k_,n_ m_]:=...</code> | Definimos la función 4.7. |
| <code>composicion2[k_,j_,n_]:=...</code> | Definimos la función 4.9. |
| <code>Diedrico[n_]:=Module[{i, giro, simetria, comp}, conjunto = {};</code> | Función “Diedrico” con un único argumento “n” que calculará D_n . |
| <code>giro = Indice[Table[Mod[i, n] + 1, {i, 1, n}]]; simetria = Indice[Table[(n + 1) - i, {i, n}]]; AppendTo[conjunto, giro]; AppendTo[conjunto, simetria]; comp = giro;</code> | Definimos el giro dado por el ciclo $\sigma = (1\ 2\ \dots\ n)$ y la simetría: $\begin{pmatrix} 1 & 2 & \dots & n-1 & n \\ n & n-1 & \dots & 2 & 1 \end{pmatrix}$ |
| <code>Do[</code> | |
| <code>AppendTo[conjunto, composicion2[simetria, comp, n]]; comp = composicion2[comp, giro, n]; AppendTo[conjunto, comp];</code> | Calculamos las potencias de σ (los giros) y las componemos con la simetría “simetría” (las simetrías). |
| <code>,{i, n - 1}];</code> | |
| <code>Sort[conjunto]</code> | Salida de resultados. |
| <code>]</code> | |

Función 4.19. El grupo Diédrico.

Además, Mathematica dentro del paquete `<<Combinatorica`` incorpora una función que determina el grupo diédrico D_n :

DihedralGroup[n]

Ejemplo 4.16. Calculamos D_7 , comprobamos que es un grupo no commutativo y calculamos todos sus subgrupos.

Primero calculamos D_7 utilizando la función 4.19., previamente definimos 4.6.bis y 4.9.:

```
In[]:=      Indice[permutacion_]:=Position[Permutations[
Table[j,{j,1,Length[permutacion}]],permutacion][[1]][[1]]]

In[]:=      sigma[k_,n_|m_|]:=Permutations[Table[j,{j,n}]][[k,m]]

In[]:=      composicion2[k_,j_,n_|]:=Module[{t,m,z},
Position[Permutations[Table[m,{m,n}]],
```

```
Table[sigma[k,n][sigma[j,n][m]],{m,1,n}]]|[1]][[1]]]

In]:= Diedrico[n_]:=Module[{i, giro, simetria, comp},
  :
  :

In]:= Diedrico[7]

Out]= {1, 720, 840, 874, 1584, 1745, 2430, 2611, 3296, 3457,
 4167, 4201, 4321, 5040}
```

Identificamos las permutaciones usando la función 4.4.:

```
In]:= S[n_]:=Table[MatrixForm[{Table[j,{j,n}],
  Permutations[Table[j,{j,n}]]|[i]}],{i,n}]

In]:= listado=Diedrico[7];
S7=S[7];
Do[Print[\sigma_{listado[[i]]},",",S7[[listado[[i]]]],",",
\sigma_{listado[[i+1]]},",",S7[[listado[[i+1]]]]],{i,1,Length[listado],2}]

Out]=  $\sigma_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{pmatrix}$        $\sigma_{720} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 7 & 6 & 5 & 4 & 3 & 2 \end{pmatrix}$   

 $\sigma_{840} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 1 & 7 & 6 & 5 & 4 & 3 \end{pmatrix}$        $\sigma_{874} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 3 & 4 & 5 & 6 & 7 & 1 \end{pmatrix}$   

 $\sigma_{1584} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 2 & 1 & 7 & 6 & 5 & 4 \end{pmatrix}$        $\sigma_{1745} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 4 & 5 & 6 & 7 & 1 & 2 \end{pmatrix}$   

 $\sigma_{2430} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 4 & 3 & 2 & 1 & 7 & 6 & 5 \end{pmatrix}$        $\sigma_{2611} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 4 & 5 & 6 & 7 & 1 & 2 & 3 \end{pmatrix}$   

 $\sigma_{3296} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 4 & 3 & 2 & 1 & 7 & 6 \end{pmatrix}$        $\sigma_{3457} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 7 & 1 & 2 & 3 & 4 \end{pmatrix}$   

 $\sigma_{4167} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 6 & 5 & 4 & 3 & 2 & 1 & 7 \end{pmatrix}$        $\sigma_{4201} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 6 & 7 & 1 & 2 & 3 & 4 & 5 \end{pmatrix}$   

 $\sigma_{4321} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix}$        $\sigma_{5040} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 6 & 5 & 4 & 3 & 2 & 1 \end{pmatrix}$ 
```

Y si usamos la función específica de Mathematica, sería:

```
In]:= <<Combinatorica`;

In]:= D7=DihedralGroup[7]

Out]= {{1, 2, 3, 4, 5, 6, 7}, {7, 1, 2, 3, 4, 5, 6},
{6, 7, 1, 2, 3, 4, 5}, {5, 6, 7, 1, 2, 3, 4},
```

$$\{4, 5, 6, 7, 1, 2, 3\}, \{3, 4, 5, 6, 7, 1, 2\}, \\ \{2, 3, 4, 5, 6, 7, 1\}, \{7, 6, 5, 4, 3, 2, 1\}, \\ \{6, 5, 4, 3, 2, 1, 7\}, \{5, 4, 3, 2, 1, 7, 6\}, \\ \{4, 3, 2, 1, 7, 6, 5\}, \{3, 2, 1, 7, 6, 5, 4\}, \\ \{2, 1, 7, 6, 5, 4, 3\}, \{1, 7, 6, 5, 4, 3, 2\}\}$$

In[]]:= Sort[Table[Indice[D7[[i]]], {i, Length[D7]}]]

Out[]]= {1, 720, 840, 874, 1584, 1745, 2430, 2611, 3296, 3457, 4167, 4201, 4321, 5040}

Ahora, calculamos su tabla de operaciones y comprobamos que es un grupo no conmutativo:

In[]]:= G=listado;
operacion=Table[composicion2[listado[[i]],listado[[j]],7,
,{i,1,Length[listado]}, {j,1,Length[listado]}];
TableForm[operacion,TableHeadings→
{Table[StyleForm[listado[[i]],FontWeight→"Bold"],{i,14}],
Table[StyleForm[listado[[i]],FontWeight→"Bold"],{i,14}]],
TableSpacing→{2,2}]

Out[]]=

| | 1 | 720 | 840 | 874 | 1584 | 1745 | 2430 | 2611 | 3296 | 3457 | 4167 | 4201 | 4321 | 5040 |
|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 1 | 1 | 720 | 840 | 874 | 1584 | 1745 | 2430 | 2611 | 3296 | 3457 | 4167 | 4201 | 4321 | 5040 |
| 720 | 720 | 1 | 4321 | 5040 | 4201 | 4167 | 3457 | 3296 | 2611 | 2430 | 1745 | 1584 | 840 | 874 |
| 840 | 840 | 874 | 1 | 720 | 4321 | 5040 | 4201 | 4167 | 3457 | 3296 | 2611 | 2430 | 1584 | 1745 |
| 874 | 874 | 840 | 1584 | 1745 | 2430 | 2611 | 3296 | 3457 | 4167 | 4201 | 5040 | 4321 | 1 | 720 |
| 1584 | 1584 | 1745 | 874 | 840 | 1 | 720 | 4321 | 5040 | 4201 | 4167 | 3457 | 3296 | 2430 | 2611 |
| 1745 | 1745 | 1584 | 2430 | 2611 | 3296 | 3457 | 4167 | 4201 | 5040 | 4321 | 720 | 1 | 874 | 840 |
| 2430 | 2430 | 2611 | 1745 | 1584 | 874 | 840 | 1 | 720 | 4321 | 5040 | 4201 | 4167 | 3296 | 3457 |
| 2611 | 2611 | 2430 | 3296 | 3457 | 4167 | 4201 | 5040 | 4321 | 720 | 1 | 840 | 874 | 1745 | 1584 |
| 3296 | 3296 | 3457 | 2611 | 2430 | 1745 | 1584 | 874 | 840 | 1 | 720 | 4321 | 5040 | 4167 | 4201 |
| 3457 | 3457 | 3296 | 4167 | 4201 | 5040 | 4321 | 720 | 1 | 840 | 874 | 1584 | 1745 | 2611 | 2430 |
| 4167 | 4167 | 4201 | 3457 | 3296 | 2611 | 2430 | 1745 | 1584 | 874 | 840 | 1 | 720 | 5040 | 4321 |
| 4201 | 4201 | 4167 | 5040 | 4321 | 720 | 1 | 840 | 874 | 1584 | 1745 | 2430 | 2611 | 3457 | 3296 |
| 4321 | 4321 | 5040 | 720 | 1 | 840 | 874 | 1584 | 1745 | 2430 | 2611 | 3296 | 3457 | 4201 | 4167 |
| 5040 | 5040 | 4321 | 4201 | 4167 | 3457 | 3296 | 2611 | 2430 | 1745 | 1584 | 874 | 840 | 720 | 1 |

Para comprobar que es un grupo utilizaremos las funciones 2.3., 2.6. y 2.8.:

In[]]:= INTERNA[G,operacion]:=Module[{CONTADORi,

⋮ ⋮

In[]]:= INTERNA[G,operacion]

Out[]]= True

In[]:= ELEMENTOSIMETRICO[G_,operacion_]:=Module[{

⋮ ⋮

In[]:= ELEMENTOSIMETRICO[G,operacion]

Out[] = Elemento neutro: 1

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|-------------|---|-----|-----|------|------|------|------|------|------|------|------|------|------|------|
| Elementos: | 1 | 720 | 840 | 874 | 1584 | 1745 | 2430 | 2611 | 3296 | 3457 | 4167 | 4201 | 4321 | 5040 |
| Simétricos: | 1 | 720 | 840 | 4321 | 1584 | 4201 | 2430 | 3457 | 3296 | 2611 | 4167 | 1745 | 874 | 5040 |

True

In[]:= ASOCIATIVA[G_,operacion_]:=Module[{CONTADORi,

⋮ ⋮

In[]:= ASOCIATIVA[G,operacion]

Out[] = True

Y veamos si es commutativo:

In[]:= operacion==Transpose[operacion]

Out[] = True

Para calcular todos sus subgrupos usaremos 3.16., previamente definiremos las funciones 2.1., 3.10 y almacenaremos el neutro en “ElementoNeutro”.

In[]:= ElementoNeutro=1;
op[x_,y_]:=operacion[[Position[G,x][[1]],
Position[G,y][[1]]][[[1]][[1]]];
GENERADO[A_]:=Module[{CONTADORx,

⋮ ⋮

In[]:= SUBGRUPOS:=Block[{ciclicos,numero,tiempo,

⋮ ⋮

In[]:= SUBGRUPOS

Out[] = Tiempo total: 0.031

$\{\{1\}, \{1, 720\}, \{1, 840\}, \{1, 1584\}, \{1, 2430\},$
 $\{1, 3296\}, \{1, 4167\}, \{1, 5040\},$
 $\{1, 874, 1745, 2611, 3457, 4201, 4321\},$
 $\{1, 720, 840, 874, 1584, 1745, 2430, 2611, 3296\}$

$3457, 4167, 4201, 4321, 5040\} \}$

□

Ejemplo 4.17. Calculamos D_5 y comprobamos que es un subgrupo de S_5 y A_5 .

Utilizaremos la función 4.19., previamente definimos 4.6.bis y 4.9.:

```
In]:= Indice[permutacion_]:=Position[Permutations[
Table[j,{j,1,Length[permutacion]}]],permutacion][[1]][[1]]

In]:= sigma[k_,n_][m_]:=Permutations[Table[j,{j,n}]][[k,m]]

In]:= composicion2[k_,j_,n_]:=Module[{t,m,z},
Position[Permutations[Table[m,{m,n}]],
Table[sigma[k,n][sigma[j,n][m]],[{m,1,n}]]][[1]][[1]]]

In]:= Diedrico[n_]:=Module[{i, giro, simetria, comp},
⋮ ⋮

In]:= D5=Diedrico[5]
```

Out]= {1, 24, 30, 34, 56, 65, 87, 91, 97, 120}

Calculamos S_5 y su tabla, comprobamos que D_5 es un subgrupo suyo con 3.1.bis.

```
In]:= Diedrico[n_]:=Module[{i, giro, simetria, comp},
⋮ ⋮

In]:= S5=Table[i,{i,5!}];
operacion=Table[composicion2[j,k,5],{k,5!},{j,5!}];

In]:= SUBGRUPO[H_,G_,operacion_]:=Module[{subgrupo,
⋮ ⋮

In]:= SUBGRUPO[D5,S5,operación]

Out]= True
```

Ahora calculamos A_5 con 4.18.bis. y comprobamos que D_5 es un subconjunto suyo:

```
In]:= n=5;
⋮ ⋮
```

Out[]= 0.016
{1, 4, 5, 8, 9, 12, 13, 16, 17, 20, 21, 24, 26, 27, 30, 31,
34, 35, 38, 39, 42, 43, 46, 47, 49, 52, 53, 56, 57, 60, 61,
64, 65, 68, 69, 72, 74, 75, 78, 79, 82, 83, 86, 87, 90, 91,
94, 95, 97, 100, 101, 104, 105, 108, 109, 112, 113,
116, 117, 120}

In[]:= **Intersection[Alternado,D5]==D5**

Out[]= True

Aunque ya podemos concluir que D_5 es subgrupo¹⁷ de A_5 , lo comprobamos con la función 3.1.bis.

In[]:= **operacionA=Table[composicion2[Alternado[[i]],**
Alternado[[j]],5],
{i,Length[Alternado]}, {j,Length[Alternado]}];
SUBGRUPO[D5,Alternado,operacionA]

Out[]= True

□

5. LA MÁQUINA ENIGMA

Los grupos simétricos constituyen la base teórica de muchos sistemas de encriptación y cifrado de mensajes. En la Segunda Guerra Mundial los alemanes utilizaron la máquina Enigma para cifrar mensajes de todo tipo, muchos de ellos de especial importancia para el transcurso y posterior desarrollo del conflicto, la máquina Enigma utiliza un sistema de cifrado simétrico que se basa en el uso de permutaciones del grupo simétrico S_{26} por las 26 letras del abecedario latino (lógicamente, no el castellano). La clave cambiaba cada día y éstas eran distribuidas mediante un libro de claves a los operadores de radio, la misma clave permitía el cifrado y el descifrado de los mensajes y estaba compuesta por una terna de tres letras del abecedario. En un sistema de cifrado simétrico se utiliza la misma clave para cifrar y descifrar el mensaje, aunque se conociera el algoritmo de descifrado, no podría descifrarse un mensaje en ausencia de la clave necesaria, uno de los usos más conocidos y atrayentes de un sistema de cifrado simétrico fue la máquina Enigma.

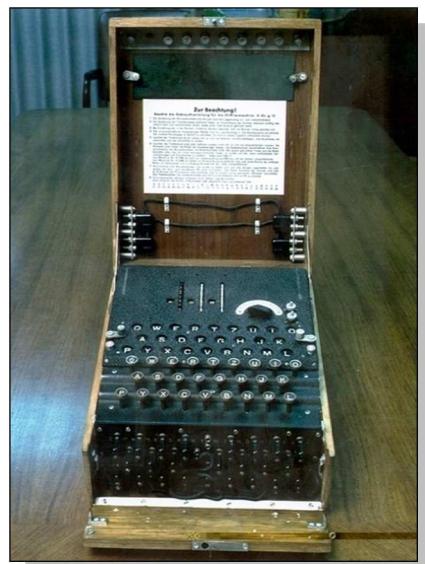


Ilustración 4.1. Máquina Enigma.

¹⁷ En general D_n no es subgrupo de A_n , de hecho si n es impar lo es y si es par no lo es.

5.1. DESCRIPCIÓN DE LA MÁQUINA ENIGMA

Sin ánimo de ser exhaustivos describimos brevemente el funcionamiento de la Máquina Enigma. En la máquina se distinguían las siguientes partes:

- **TECLADO:** Entrada de datos mediante un teclado con las 26 letras del abecedario.
- **PANEL DE SALIDA:** Salida de datos en un panel de 26 luces, cada una de las cuales representaba una letra del abecedario latino.
- **ROTORES:** Rotores “configurables” cada uno de los cuales realizaba una permutación, había tres rotors que simbolizaremos por α, β, γ , eran como el cuenta kilómetros (no digital) de un coche, tres ruedas o cilindros en cuya cara externa se encontraban las letras del abecedario, en su parte interior, en su núcleo, se encontraba un cableado con distintos contactos que se encargaban de realizar la permutación, al cambiar la posición del núcleo respecto al anillo exterior (abecedario) cambia la permutación. De esta forma según la clave del día se configuraban los tres rotors mediante una terna de tres letras del abecedario, por ejemplo “AAA”, que daría lugar a tres permutaciones distintas, que denotaremos α_A, β_A y γ_A . El número total de posibles claves sería 26^3 . Si identificamos al conjunto $\{A, B, C, \dots, Z\}$ con el conjunto $\{1, 2, 3, \dots, 26\}$, y consideramos la permutación (con notación cíclica)

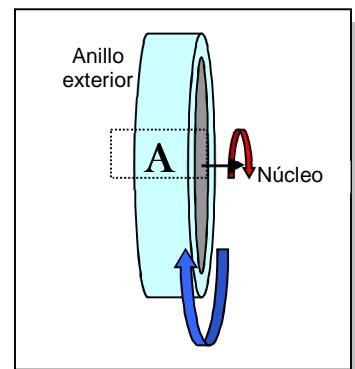


Ilustración 4.2. Máquina Enigma: Rotor.

$$\sigma = (1 \ 2 \ 3 \ 4 \ \dots \ 25 \ 26),$$

entonces se tiene que el avance del anillo exterior desde la letra A hasta la letra B, quedando el núcleo fijo, se traducirá de la siguiente forma:

$$\sigma \circ \alpha_A \circ \sigma^{-1} = \alpha_B,$$

y con la identificación del abecedario tendríamos,

$$\begin{aligned} \sigma^{p-1} \circ \alpha_1 \circ (\sigma^{-1})^{p-1} &= \alpha_p, \\ \sigma^{p-1} \circ \beta_1 \circ (\sigma^{-1})^{p-1} &= \beta_p, \\ \sigma^{p-1} \circ \gamma_1 \circ (\sigma^{-1})^{p-1} &= \gamma_p, \quad \forall p = 2, 3, \dots, 26 \end{aligned} \quad (1)$$

Los rotors avanzaban a cada letra del mensaje a cifrar, como ocurre en un cuenta kilómetros, si el mensaje era “HOLA” y empezamos con la clave “AAA”.

$$\begin{array}{lll} H & \text{se codificaría con las permutaciones asociadas a la terna: } AAA \equiv (1, 1, 1) \\ O & “ & “ \end{array} \quad \begin{array}{ll} & AAB \equiv (1, 1, 2) \end{array}$$

| | | | |
|---|---|---|------------------------|
| L | " | " | AAC \equiv (1, 1, 3) |
| A | " | " | AAD \equiv (1, 1, 4) |

Así sucesivamente, de forma que las permutaciones a usar eran distintas para cada letra del mensaje.

- REFLECTOR: El reflector era una nueva permutación δ , verificando: $\delta = \delta^{-1}$.
- PANEL DE CONEXIONES: Adicionalmente se añadía otra permutación más que actuaba antes y después de los rotores, esta dependía de un ínter conexionado exterior y que añadía más fuerza a la protección, la denotaremos por ϵ . (No todas las máquinas Enigma disponían de este panel).

5.2. FUNCIONAMIENTO DE LA MÁQUINA ENIGMA

La composición en S_{26} de todas las permutaciones descritas en 4.1. cifraba a cada letra introducida por el teclado, el orden en actuar era el siguiente:

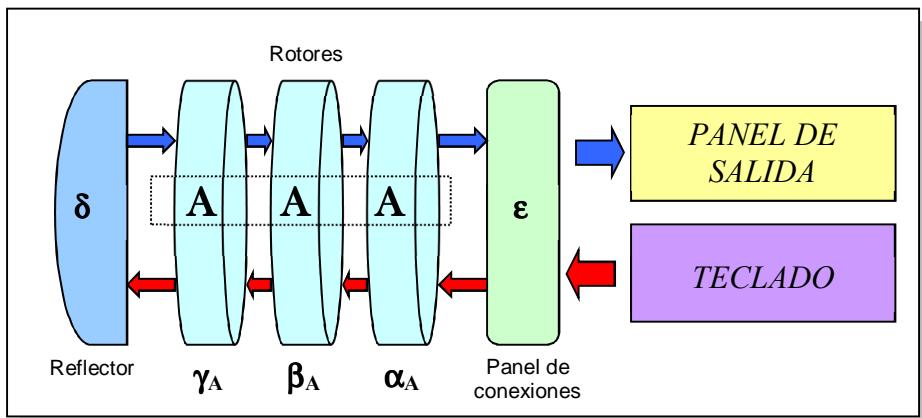


Ilustración 4.3. Máquina Enigma: Esquema de la composición de permutaciones.

1. ϵ , que representa al panel de conexionado.
2. α_i , el rotor más lento, configurado el anillo exterior con respecto al núcleo en la posición i , con $i \in \{1, 2, \dots, 26\}$, esto es, salvo la identificación en una letra del abecedario latino {A, B, ..., Z}.
3. β_j , el rotor intermedio, en la posición j con $j \in \{1, 2, \dots, 26\}$.
4. γ_k , el rotor más rápido, en la posición k con $k \in \{1, 2, \dots, 26\}$.
5. δ , el reflector.
6. $(\gamma_k)^{-1}$, después de pasar por el reflector vuelve por todos los rotores en sentido inverso, la permutación inversa del rotor más rápido.
7. $(\beta_j)^{-1}$, la inversa del rotor intermedio.
8. $(\alpha_i)^{-1}$, la inversa del rotor más lento.
9. ϵ^{-1} , inversa del panel de conexionado.

Luego si la clave inicial es “AAA” y el mensaje es “HOLA”, el carácter “H” quedaría codificado por:

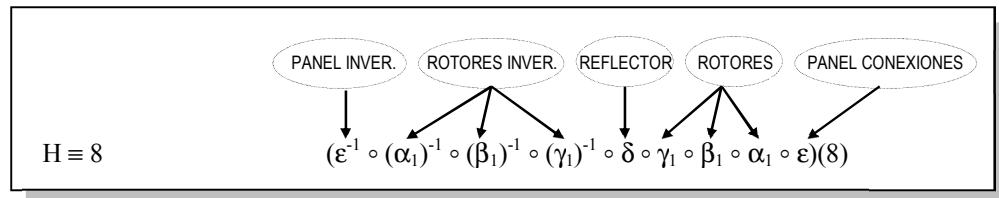


Ilustración 4.4.
Y teniendo en cuenta (1), el carácter “O”:

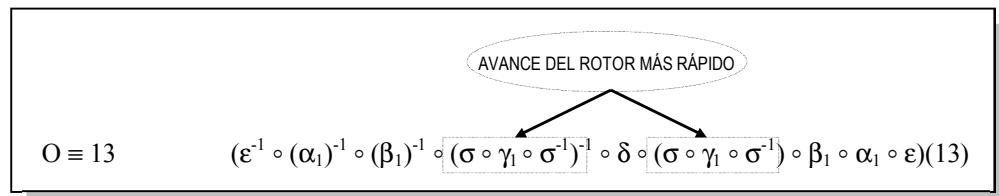


Ilustración 4.5.

Para los caracteres “L” y “A” avanzaría de nuevo el rotor más rápido:

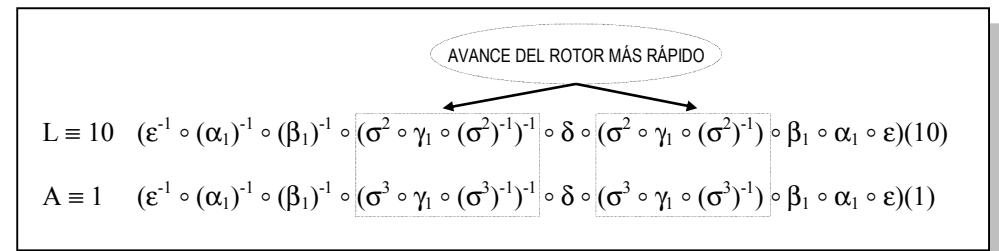


Ilustración 4.6

Obsérvese que el descifrado es idéntico porque las inversas de las permutaciones obtenidas son ellas mismas:

$$(\varepsilon^{-1} \circ (\alpha_i)^{-1} \circ (\beta_j)^{-1} \circ (\gamma_k)^{-1} \circ \delta \circ \gamma_k \circ \beta_j \circ \alpha_i \circ \varepsilon)^2 = I, \forall i, j, k \in \{1, 2, \dots, 26\}.$$

Ejemplo 4.18. A modo de ejemplo veamos cómo podríamos implementar el funcionamiento de la máquina Enigma limitándonos a un abecedario de sólo 5 letras: A, B, C, D y E, que identificaremos con el 1, 2, 3, 4 y 5 respectivamente. Necesitaremos 5 permutaciones de S_5 que representen los núcleos de los rotores, el reflector y el panel de conexiones, las elegimos al azar, usaremos la notación cíclica, por ejemplo:

$$\alpha_1 = (1 \ 2 \ 3) (4 \ 5),$$

$$\beta_1 = (2 \ 1 \ 4) (3 \ 5),$$

$$\gamma_1 = (2 \ 3) (5 \ 4 \ 1),$$

la permutación δ , del reflector debe verificar $\delta^2 = I$, por tanto será una composición de trasposiciones disjuntas:

$$\delta = (1\ 2)\ (4\ 5),$$

y por último el panel de conexiones, elegiremos por ejemplo:

$$\varepsilon = (4\ 1\ 2)\ (3\ 5).$$

Además necesitaremos de la permutación $\sigma = (1\ 2\ 3\ 4\ 5)$ y $\sigma^{-1} = (5\ 4\ 3\ 2\ 1)$ para obtener las permutaciones asociadas a las posiciones relativas del núcleo con el anillo exterior de cada rotor.

Implementamos todas las permutaciones y las identificamos por sus índices, ya que nos será más cómodo usarlas de esta forma, para ello usamos la función 4.6.:

```
In]:= Indice[permutacion_]:=Module[{i},
    :
    :
```

Identificamos los índices:

```
In]:= alpha=Indice[{2,3,1,5,4}];
beta= Indice[{4,1,5,2,3}];
gamma=Indice[{5,3,2,1,4}];
delta=Indice[{2,1,3,5,4}];
epsilon=Indice[{2,4,5,1,3}];
SIGMA=Indice[{2,3,4,5,1}];
inversasigma=Indice[{5,1,2,3,4}];
inversaalpha=Indice[{3,1,2,5,4}];
inversabeta=Indice[{2,4,5,1,3}];
inversagamma=Indice[{4,3,2,5,1}];
inversaepsilon=Indice[{4,1,5,2,3}];
```

Definimos la función:

```
In]:= sigma[k_,n_][m_]:=Permutations[Table[j,{j,n}]][[k,m]];
```

Programamos la potencia n -ésima de las permutaciones σ y σ^{-1}

```
In]:= potSIGMA[n_,k_]:=Module[{t},
    If[n==0,k,
     temp=k;
     Do[temp=sigma[SIGMA,5][temp],{t,1,n}];
     temp]
   ];
```

```
In]:= potinversaSIGMA[n_,k_]:=Module[{t},
    If[n==0,k,
     temp=k;
     Do[temp=sigma[inversasigma,5][temp],{t,1,n}];
```

temp]
];

que calcularían respectivamente, $\sigma^n(k)$ y $(\sigma^{-1})^n(k)$. Y ahora introducimos la composición de todas las permutaciones, primero simulamos el funcionamiento de los rotores, y después componemos:

```
In[]:= A1[t_,n_]:= potSIGMA[n-1,
    sigma[inversaalpha,5][potinversaSIGMA[n-1,t]]];
A2[t_,n_]:=potSIGMA[n-1,
    sigma[inversabeta,5][potinversaSIGMA[n-1,t]]];
A3[t_,n_]:=potSIGMA[n-1,
    sigma[inversagamma,5][potinversaSIGMA[n-1,t]]];
A4[t_,n_]:=potSIGMA[n-1,
    sigma[alpha,5][potinversaSIGMA[n-1,t]]];
A5[t_,n_]:=potSIGMA[n-1,
    sigma[beta,5][potinversaSIGMA[n-1,t]]];
A6[t_,n_]:=potSIGMA[n-1,
    sigma[gamma,5][potinversaSIGMA[n-1,t]]];

In[]:= COMPOSICION[X1_,X2_,X3_][t]:= 
    sigma[inversaepsilon,5][A1[A2[A3[sigma[delta,5][
        A6[A5[A4[sigma[epsilon,5][t],X1],X2],X3]],X3],X2],X1]];
```

Ciframos el mensaje “ACCEDA” para la clave inicial “DEB”. La clave inicial “DEB” se correspondería con la terna $(4, 5, 2)$, luego

| Mensaje original | Clave | Codificación con Mathematica | Mensaje cifrado |
|------------------|------------------------|--|-----------------|
| $A \equiv 1$ | $DEB \equiv (4, 5, 2)$ | $In[]:= \text{COMPOSICION}[4,5,2][1]$ $Out[] = 2$ | $2 \equiv B$ |
| $C \equiv 3$ | $DEC \equiv (4, 5, 3)$ | $In[]:= \text{COMPOSICION}[4,5,3][3]$ $Out[] = 5$ | $5 \equiv E$ |
| $C \equiv 3$ | $DED \equiv (4, 5, 4)$ | $In[]:= \text{COMPOSICION}[4,5,4][3]$ $Out[] = 2$ | $2 \equiv B$ |
| $E \equiv 5$ | $DEE \equiv (4, 5, 5)$ | $In[]:= \text{COMPOSICION}[4,5,5][5]$ $Out[] = 5$ | $5 \equiv E$ |
| $D \equiv 4$ | $EAA \equiv (5, 1, 1)$ | $In[]:= \text{COMPOSICION}[5,1,1][4]$ $Out[] = 2$ | $2 \equiv B$ |
| $A \equiv 1$ | $EAB \equiv (5, 1, 2)$ | $In[]:= \text{COMPOSICION}[5,1,2][1]$ $Out[] = 4$ | $4 \equiv D$ |

Tabla 4.4. Cifrado de un mensaje con ENIGMA.

Así el mensaje “ACCEDA” quedaría cifrado como “BEBEBD”. Ahora probemos a decodificar, recordemos que la clave inicial debe ser la misma,

| Mensaje cifrado | Clave | Codificación con Mathematica | Mensaje original |
|-----------------|------------------------|--|------------------|
| $B \equiv 2$ | $DEB \equiv (4, 5, 2)$ | $In[]:= \text{COMPOSICION}[4,5,2][2]$ $Out[] = 1$ | $1 \equiv A$ |

| | | | |
|--------------|------------------------|---|--------------|
| $E \equiv 5$ | $DEC \equiv (4, 5, 3)$ | $In[] := \text{COMPOSICION}[4,5,3][5]$ $Out[] = 3$ | $3 \equiv C$ |
| $B \equiv 2$ | $DED \equiv (4, 5, 4)$ | $In[] := \text{COMPOSICION}[4,5,4][2]$ $Out[] = 3$ | $3 \equiv C$ |
| $E \equiv 5$ | $DEE \equiv (4, 5, 5)$ | $In[] := \text{COMPOSICION}[4,5,5][5]$ $Out[] = 5$ | $5 \equiv E$ |
| $B \equiv 2$ | $EAA \equiv (5, 1, 1)$ | $In[] := \text{COMPOSICION}[5,1,1][2]$ $Out[] = 4$ | $4 \equiv D$ |
| $D \equiv 4$ | $EAB \equiv (5, 1, 2)$ | $In[] := \text{COMPOSICION}[5,1,2][4]$ $Out[] = 1$ | $1 \equiv A$ |

Tabla 4.5. Descifrado de un mensaje con ENIGMA.

□

Es bastante inmediato sistematizar el proceso anterior con bucles para realizar la codificación y descodificación de forma más rápida de cualquier mensaje.

En el ejercicio 4.29. se propone sistematizar el proceso y generalizar al uso de abecedarios más grandes, en particular al que usaron los alemanes.

6. EJERCICIOS

Ejercicio 4.1. Calcular todas las permutaciones de los elementos del conjunto $\{1, 2, 3, 4, 5, 6, 7, 8\}$ por todos métodos vistos en el capítulo.

□

Ejercicio 4.2. Realizar programas similares a las funciones 4.1., 4.2. y 4.3. que calculen variaciones y combinaciones a partir de un conjunto de n elementos.

□

Ejercicio 4.3. Usar las funciones $S[]$ (4.4.) y $\text{NombreS}[]$ (4.5.) de la sección 2.1.2. para calcular S_4 y comprobar cuáles son los índices asignados a cada permutación.

□

Ejercicio 4.4. Definir las permutaciones $\sigma, \tau \in S_4$, $\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix}$, $\tau = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 2 & 4 \end{pmatrix}$

por todos los métodos expuestos en el capítulo. Calcular $\sigma\tau$ y $\tau\sigma$. Identificar los índices asignados en el ejercicio 4.3. a $\sigma\tau$ y $\tau\sigma$.

□

Ejercicio 4.5. Determinar la tabla de operaciones de los grupos S_2 y S_4 .

□

Ejercicio 4.6. Definir el grupo S_4 usando su tabla de operaciones igual que en el capítulo 3. Comprobar que en efecto es un grupo no conmutativo usando los programas necesarios del capítulo 3.

□

Ejercicio 4.7. Comprobar que S_2 es un grupo conmutativo utilizando las técnicas del apartado 1. del capítulo 3. ¿Es isomorfo a \mathbb{Z}_2 ? □

Ejercicio 4.8. Crear un programa parecido a la función Indice[] (4.6.) o Index[], que utilice la notación para las permutaciones expuesta al principio de la sección 2. □

Ejercicio 4.9. Determinar el elemento simétrico (inverso¹⁸) de una permutación, calcular el inverso de una permutación cualquiera a partir de la tabla de operaciones, igual que en el capítulo 3. □

Ejercicio 4.10. Para cualquier permutación $\sigma \neq I$, sabemos que existe un entero positivo k tal que $\sigma^k = I$, por tanto σ^{k-1} es el inverso de σ , programar una función que determine el simétrico o inverso de cualquier permutación utilizando esta propiedad. □

Ejercicio 4.11. Comprobar que la composición de permutaciones no es conmutativa. □

Ejercicio 4.12. Crear un programa que detecte si una permutación es un ciclo. □

Ejercicio 4.13. Calcular la notación cíclica de las permutaciones:

- a) $\sigma, \tau \in S_4, \sigma = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix}, \tau = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 2 & 4 \end{pmatrix}.$
- b) $\sigma, \tau \in S_5, \sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 4 & 5 & 3 \end{pmatrix}, \tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 1 & 4 & 5 & 2 \end{pmatrix}.$
- c) $\sigma, \tau \in S_6, \sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 1 & 2 & 5 & 4 & 6 \end{pmatrix}, \tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 6 & 2 & 5 & 4 & 1 \end{pmatrix}.$

Ejercicio 4.14. Crear un programa que calcule el ciclo inverso de un ciclo cualquiera. □

Ejercicio 4.15. Crear un programa que haga lo mismo que la función FromCycles[]. □

¹⁸ Despu  s de introducir el paquete de Matem  tica Discreta:

`<<Combinatorica`

dispondremos de la func  n **InversePermutation[permutaci  n]**, cuyo argumento es una permutaci  n y realiza esta tarea.

Ejercicio 4.16. Descomponer en ciclos la aplicación identidad en $X = \{1, 2, 3, 4, 5\}$ con el programa 4.6. y con la función ToCycles[] e interpretar ambos resultados.

Ejercicio 4.17. Modificar el programa 4.12. que calcula la notación cíclica de una permutación para que la salida se muestre con paréntesis y sin comas, según la notación habitual de los ciclos.

Ejercicio 4.18. Descomponer como producto de trasposiciones los siguientes ciclos:

- a) $(1\ 3\ 5\ 7)$.
- b) $(2\ 3\ 5\ 4\ 1)$.
- c) $(2\ 4\ 6)$.



Ejercicio 4.19. Descomponer como producto de trasposiciones las permutaciones del ejercicio 4.13.

Ejercicio 4.20. Realizar un programa análogo al 4.13. que descomponga los ciclos en producto de trasposiciones como en el caso (2) de 2.3.

Ejercicio 4.21. Utilizar el programa que realizaste en el ejercicio 4.14 para crear un nuevo programa que determine el inverso de una permutación cualquiera a partir de su notación cíclica. Hacer lo mismo para trasposiciones.

Ejercicio 4.22. Comprobar explícitamente que el cardinal de A_6 es 360.

Ejercicio 4.23. Calcular la paridad y el número de inversiones de las permutaciones de los ejercicios 4.13. y 4.18.

Ejercicio 4.24. Calcular A_2 y A_3 .

Ejercicio 4.25. Crear un pequeño programa que usando la función AlternatingGroup[] tenga por salida lo mismo que la función A[] (4.17.).

Ejercicio 4.26. Comprobar que S_5 es un grupo no conmutativo y A_5 es un subgrupo normal.

Ejercicio 4.27. Calcular S_5 y A_5 . Determinar el grupo cociente S_5/A_5 y comprobar que es isomorfo al grupo $G=\{-1, 1\}$ con la operación:

| | | |
|----|----|----|
| . | 1 | -1 |
| 1 | 1 | -1 |
| -1 | -1 | 1 |

□

Ejercicio 4.28. Sistemas de numeración. En la referencia [25] de la bibliografía, podemos encontrar un estudio bastante completo de los sistemas de numeración y los posibles métodos computacionales que resuelven el problema. Ahora pretendemos obtener las permutaciones de un conjunto usando los sistemas de numeración, la idea es la siguiente, si $X = \{A, B, C, D\}$, vamos a identificar cada elemento con un número $A = 0, B = 1, C = 2$ y $D = 3$, y usando el sistema de numeración de base 4, las cuaternas que podríamos obtener serían:

$$\begin{aligned}\{A, A, A, A\} &\equiv AAAA \equiv (0000)_4 = 0 \\ \{A, A, A, B\} &\equiv AAAB \equiv (0001)_4 = 1 \\ \{A, A, A, C\} &\equiv AAAC \equiv (0002)_4 = 2 \\ \{A, A, A, D\} &\equiv AAAD \equiv (0003)_4 = 3 \\ \{A, A, B, A\} &\equiv AABA \equiv (0010)_4 = 4\end{aligned}$$

⋮ ⋮

$$\{D, D, D, D\} \equiv DDDD \equiv (3333)_4 = 255$$

se obtienen 256 listas, y de éstas, sólo son permutaciones aquellas en las que todos los “dígitos” son distintos.

Crear un programa que calcule las permutaciones de un conjunto usando los sistemas de numeración.

□

Ejercicio 4.29. La máquina Enigma. En el ejemplo 4.15. se implementa a modo de ejemplo el funcionamiento de la máquina Enigma para un abecedario de 5 letras. En el CDROM se ha incluido una implementación de la máquina Enigma original, con un abecedario de 26 letras, en formato *html* y usando *Javascript*, que puede trasladarse fácilmente a *Mathematica*.

- Crear funciones que cifren y descifre mensajes para el abecedario del ejemplo 4.15.
- Realizar un programa que simule el funcionamiento de la máquina Enigma para el abecedario latino estándar. Cifrar tu nombre para la clave “JFR” y comprobar que su descifrado es correcto.

□

Ejercicio 4.30.

- Calcular S_3 y determinar su tabla de operaciones. Calcular su elemento neutro y los elementos simétricos de todos sus elementos.
- Calcular A_4 . ¿Es normal?

- iii. Determinar el número de inversiones y la paridad de las siguientes permutaciones de S_7 :
- σ_n donde $n = \text{Mod}[\text{DNI}, 3040] + 1$.
 - σ_m donde $m = \text{mes} * \text{día} * (\text{Mod}[\text{año}, 13] + 1)$ y $\text{día}/\text{mes}/\text{año}$ es su fecha de nacimiento.
 - El producto o composición de σ_n y σ_m .

□

Ejercicio 4.31. Calcular todos los subgrupos de $D_4 \times A_3$ y D_8 .

□

Ejercicio 4.32. ¿Es D_5 un subgrupo normal de S_5 ?

□

5. GRAFOS. REPRESENTACIÓN E IMPLEMENTACIÓN

El estudio algorítmico, sobre la resolución de posibles problemas en todo lo relacionado con el tema de grafos, es bastante amplio y podemos encontrar una bibliografía al respecto muy completa. Nos restringiremos a describir los conceptos más importantes de la teoría de grafos y trasladarlos lo más fidedignamente posible al ordenador para su mejor comprensión y para facilitar su uso posterior. Utilizaremos algunas de las funciones específicas para la representación gráfica que incorpora Mathematica para grafos, en este capítulo nos limitaremos a mencionar tales funciones sin proporcionar alternativas a ellas trasladables a otras plataformas como se ha hecho en otros casos. Nuestra visión en este tema será algo más limitada de lo que desearíamos por los requisitos que encontramos fuera del campo de la Matemática Discreta y del Álgebra.

Partiendo de estas premisas, en este capítulo y los siguientes, resolvemos computacionalmente todos los problemas y conceptos básicos relacionados con la teoría de grafos, sin pretender solventarlos de la forma más óptima y sacrificando, en su caso, eficacia a cambio de comprensión, quedando para el lector interesado la ampliación en dichas cuestiones con la bibliografía oportuna (por ejemplo, algunas de las referencias incluidas en la sección de bibliografía). Algunos conceptos y problemas son más difíciles de trasladar al ordenador, en tales casos no se ha podido dar una sencilla respuesta como hubiera sido deseable, siendo en consecuencia las soluciones expuestas de un nivel más elevado en cuanto a su aprendizaje y comprensión se refiere.

En este capítulo aprendemos a representar grafos con el ordenador desde su definición, matricialmente y gráficamente. Estudiaremos el problema de la isomorfía entre grafos y daremos soluciones al mismo, revelaremos el gran problema combinatorio que hay detrás de las construcciones de grafos e intentaremos solucionarlo lo más eficazmente posible dentro de las limitaciones teóricas y computacionales que enmarcan este libro. Terminaremos el capítulo estudiando el grupo de automorfismos de un grafo, conectando así la teoría de grafos con la de grupos, y aprovechando lo estudiado en capítulos anteriores.

1. DEFINICIÓN Y TIPOS DE GRAFOS

La noción de grafo y los conceptos asociados pueden variar según la aplicación que pretendamos darle o el problema que queramos formalizar. En principio los definiremos de la forma más genérica posible, pero finalmente nos quedaremos con las

nociones de grafo más habituales y simples para ilustrar el resto de conceptos, tanto en este capítulo como en los siguientes. Distinguimos dos tipos básicos de grafos, orientados y no orientados:

- Un grafo orientado o digrafo G es una terna $(W, F; \gamma)$ formada por dos conjuntos no necesariamente finitos y una aplicación:
 - i. $W \neq \emptyset$ el conjunto de sus vértices o nodos, también denotado por $W(G)$;
 - ii. F el conjunto de sus flechas, arcos o aristas orientadas, también denotado por $F(G)$;
 - iii. la aplicación $\gamma: F \rightarrow W \times W$, que definiremos para cada flecha $\alpha \in F$ por $\gamma(\alpha) = (v, w) \in W \times W$, donde $v \in W$ es el vértice inicial u origen de la flecha α y $w \in W$ es el vértice final o destino de la flecha α .

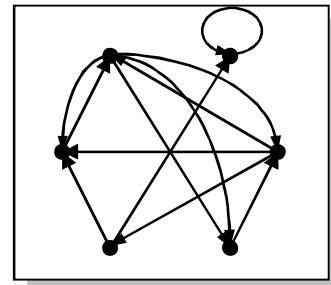


Ilustración 5.1.

A las flechas con el mismo vértice inicial y final ($v = w$), las llamaremos lazos o bucles.

- Un grafo no orientado G es una terna $(W, F; c)$ formada por dos conjuntos no necesariamente finitos y una aplicación:
 - i. $W \neq \emptyset$ el conjunto de sus vértices;
 - ii. F el conjunto de sus lados o aristas;
 - iii. la aplicación $c: F \rightarrow \{\{v, w\} \mid v, w \in W\}$ (aplicación del conjunto F en el conjunto de todos los subconjuntos de W con 1 o 2 elementos), que definiremos para cada lado $\alpha \in F$ por $c(\alpha) = \{v, w\} \subseteq W$ donde v y w son los vértices que conectan dicho lado, en tal caso diremos que v y w son vértices adyacentes, y además que el lado α es incidente con los vértices v y w .

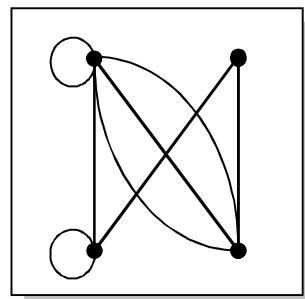


Ilustración 5.2.

Si $|c(\alpha)| = 1$ entonces α es un lazo ($v = w$).

También distinguiremos distintos modelos de grafos tanto en los orientados como no orientados:

- a) Un grafo se dice que es etiquetado, ponderado o con peso si a cada lado o flecha se le asocia un número real.
- b) Un grafo se dice que es un multigrafo si existe más de una flecha o lado incidentes con los mismos vértices.
- c) Un grafo $G = (W, F)$ se dice que es finito si los conjuntos W y F son finitos. Si $|W| = p$ y $|F| = q$, entonces diremos que G es un (p, q) -grafo.
- d) Un grafo se dice que es simple si no posee lazos y no es un multigrafo.

Como podemos observar se pueden considerar distintos tipos de grafos, la elección del modelo vendrá impuesto por el problema que queramos resolver. En este libro nos limitaremos a estudiar dos, que son los más elementales y habituales:

- 1) Grafos orientados, simples y finitos que llamaremos simplemente grafos dirigidos o dígrafos y que podemos entender que son pares $G = (W, F)$ formados por dos conjuntos finitos: $W \neq \emptyset$ el conjunto de sus vértices o nodos y $F \subseteq W \times W$ el conjunto de sus arcos o flechas, donde cada flecha $e \in F$ es un par ordenado de dos vértices $e = (v, w) \in W \times W$, que llamaremos respectivamente inicio y fin de la flecha.
- 2) Grafos no orientados, simples y finitos que llamaremos simplemente grafos no orientados, grafos no dirigidos o grafos y que podemos entender que son pares, $G = (W, F)$, formados por dos conjuntos finitos: $W \neq \emptyset$ el conjunto de sus vértices y $F \subseteq \{\{v, w\} \mid v, w \in W \wedge v \neq w\}$ (subconjunto del conjunto de todos los subconjuntos de dos elementos de W) el conjunto de sus aristas o lados, donde cada lado $e \in F$ es un subconjunto de dos vértices $e = \{v, w\} \subseteq W$.

1.1. IMPLEMENTACIÓN EN MATHEMATICA

1.1.1. DIRECTAMENTE COMO UN PAR DE CONJUNTOS

Si G es un (p, q) -grafo, el grafo vendrá dado por dos conjuntos $W = \{v_1, v_2, \dots, v_p\}$ y $F = \{e_1, e_2, \dots, e_q\}$, donde cada e_j será un lado o flecha, cada uno de ellos representado por un subconjunto de dos vértices o un par ordenado respectivamente. En cualquier caso lo introduciremos directamente en el ordenador de la misma forma:

In[]:= **W={TODOS LOS VÉRTICES SEPARADOS
POR COMAS};**
**F={TODOS LOS LADOS O FLECHAS SEPARADOS
POR COMAS};**

Un lado $\{v_i, v_j\}$ o flecha (v_i, v_j) lo introduciremos de la siguiente forma:

$$v_i \rightarrow v_j$$

en el caso de ser un lado no importa el orden de los vértices, pero si es una flecha obligatoriamente primero pondremos el origen y después el destino.

Si tenemos

$$\textbf{flecha} = \mathbf{v} \rightarrow \mathbf{w}$$

$$\textbf{lado} = \mathbf{x} \rightarrow \mathbf{y}$$

entonces: $\text{flecha}[1]$ es el origen “ v ” y $\text{flecha}[2]$ es el destino “ w ” de la flecha “flecha”; y el lado “lado” sería incidente con los vértices $\text{lado}[1]$ y $\text{lado}[2]$, esto es, “ x ” e “ y ”. Por comodidad o necesidad (como en el epígrafe siguiente), es frecuente identificar los vértices

por sus subíndices, en ese caso el lado o flecha se introduciría así: $i \rightarrow j$. De hecho en la mayoría de ejemplos expuestos en este libro se hace dicha identificación de forma automática, asignándoles a los vértices directamente como nombres las posiciones que en el conjunto de vértices, visto como lista, ocupan.

1.1.2. DEFINIENDO UN OBJETO DE TIPO GRAFO

A partir de la versión 6 de Mathematica, encontramos un nuevo tipo de objetos: "los grafos"; se identifican y manejan como tal y permiten usar los grafos de otra forma. Hasta esta versión los grafos eran manejados habitualmente sólo por su matriz de adyacencia (ver epígrafe 2.1.) o dos listas (la de sus vértices y otra de sus lados o flechas, ver 1.1.1.). Los objetos de tipo grafo pueden ser creados manualmente por el usuario o son las salidas de ciertas funciones. Dependiendo de nuestros intereses manejar así los grafos tendrá sus ventajas y sus inconvenientes, por tanto, es recomendable analizar su idoneidad según el tipo de problema que queramos resolver. Ahora resumiremos cómo pueden definirse objetos de tipo grafo y cómo se manejan.

Para definir un objeto de tipo grafo usaremos la función del paquete `<<Combinatorica``:

Graph[lados o flechas,vértices,opciones]

Función de tres argumentos:

- i. Conjunto de lados o flechas con sus opciones gráficas, cada lado o flecha: $v \rightarrow w$, será introducido como $\{\{v, w\}, \text{opciones}\}$ o directamente $\{\{v, w\}\}$ si no tenemos opciones que especificar, y el conjunto de todos los lados o flechas:

$$\{\{\{v_1, w_1\}, \text{opciones}_1\}, \{\{v_2, w_2\}, \text{opciones}_2\}, \dots, \{\{v_q, w_q\}, \text{opciones}_q\}\}$$

Las posibles opciones para los lados o flechas son:

- a) *EdgeWeight*, si queremos peso en las flechas o lados, por defecto su valor es 1.
 - b) *EdgeColor*, para asociar un color determinado a las flechas o lados.
 - c) *EdgeStyle*, para indicar el tamaño o la figura que se dibujará como lado o flecha.
 - d) *EdgeLabel*, sus valores son "True" o "False" y se utilizará para indicar si en la representación gráfica con `ShowGraph[]` debe aparecer el nombre de los lados o flechas, también se puede indicar directamente el nombre del lado o flecha. "*EdgeLabelColor*" y "*EdgeLabelPosition*", indicarán el color y la posición relativa.
- ii. Conjunto de las posiciones o coordenadas de los vértices en el plano con sus opciones gráficas, un vértice que es dibujado en las coordenadas (x, y) del plano será introducido como $\{\{x, y\}, \text{opciones}\}$ o directamente $\{\{x, y\}\}$ si no tenemos opciones que especificar, y el conjunto de todos los vértices:

$$\{\{\{x_1, y_1\}, \text{opciones}_1\}, \{\{x_2, y_2\}, \text{opciones}_2\}, \dots, \{\{x_q, y_q\}, \text{opciones}_q\}\}$$

Las posibles opciones para los vértices son:

- a) *VertexWeight*, si queremos peso en los vértices, por defecto su valor es 0.
- b) *VertexColor*, para asociar un color determinado a los vértices.
- c) *VertexStyle*, para indicar el tamaño o la figura que se dibujará como vértice.
- d) *VertexNumber*, sus valores son “True” o “False” y se utilizará para indicar si en la representación gráfica con `ShowGraph[]` (ver epígrafe 5.3.) debe aparecer el número o índice que identifica¹⁹ a los vértices. “*VertexNumberColor*” y “*VertexNumberPosition*”, indicarán el color y la posición relativa de esos números.
- e) *VertexLabel*, sus valores son “True” o “False” y se utilizará para indicar si en la representación gráfica con `ShowGraph[]` debe aparecer el nombre de los vértices, también se puede indicar directamente el nombre del vértice. “*VertexLabelColor*” y “*VertexLabelPosition*”, indicarán el color y la posición relativa.

iii. Por último las opciones globales del grafo. Éstas son:

- a) *LoopPosition*, para indicar la posición de los lazos (“*UpperLeft*”, “*UpperRight*”, “*LowerLeft*” o “*LowerRight*”), si el grafo los tuviera.
- b) *EdgeDirection*, que especificará si el grafo es orientado o no, por defecto su valor es “*False*”.

Si tenemos definido un objeto de tipo grafo “G” y queremos recuperar sus vértices y lados o flechas disponemos de las funciones:

Vertices[G] y Edges[G]

y las funciones **M[G]** y **V[G]** que respectivamente devolverán el número de lados o flechas y vértices del objeto de tipo grafo “G” que tengan por argumento, las cuatro forman parte del paquete de funciones: <<Combinatorica`>

También disponemos de otras funciones para comprobar el tipo de grafo con el que trabajamos (“G” siempre será un objeto de tipo grafo):

- **SimpleQ[G]**, comprobará si el grafo es simple.
- **UnweightedQ[G]**, comprobará si el peso de los lados del grafo es siempre 1.
- **SelfLoopsQ[G]**, comprobará si el grafo tiene lazos.
- **UndirectedQ[G]**, comprobará si el grafo es no orientado.
- **MultipleEdgeQ[G]**, comprobará si el grafo es un multigrafo.
- **EmptyQ[G]**, comprueba si el conjunto de lados o flechas del grafo es el vacío.

¹⁹ Téngase presente que Mathematica puede realizar una identificación distinta a la que el lector pueda presuponer.

Otras funciones dentro del mismo paquete que podemos usar para construir objetos de tipo grafo son las siguientes:

- **AddEdge[G, lado o flecha], AddEdges[G, lista de lados], ChangeEdges[G, lista de lados o flechas], DeleteEdge[G, lado o flecha], DeleteEdges[G, lista de lados o flechas]**, para añadir, modificar o quitar lados o flechas.
- **AddVertex[G, coordenadas de un vértice], ChangeVertex[G, lista de vértices dados por sus coordenadas], DeleteVertex[G, índice de un vértice]**, para añadir, modificar o borrar vértices.
- **RemoveSelfLoops[G], RemoveMultipleEdges[G], MakeDirected[grafo no orientado], MakeUndirected[grafo dirigido], MakeSimple[G]**; que usaremos respectivamente para quitar lazos, quitar lados o flechas múltiples, convertir en dirigido, convertir en no orientado y hacer simple.
- **RandomGraph[n, p]**, genera un grafo aleatorio de “n” vértices, donde la probabilidad de que dos vértices estén conectados es “p”, y **ExactRandomGraph[n, e]**, que genera un grafo aleatorio de “n” vértices y exactamente “e” lados o flechas.
- **FromOrderedPairs[lista de pares ordenados,opciones]**, construye un objeto de tipo grafo asociado al grafo que tiene por lados o flechas los representados por los pares ordenados del primer argumento y por vértices los que intervienen en la lista, como opción indicaremos si el grafo es orientado o no con: “Type->Directed” o “Type->Undirected”. **ToOrderedPairs[G]**, hace lo contrario, esto es, devuelve una lista de pares ordenados que representarán los lados o flechas del grafo.

En el siguiente ejemplo comprobaremos como podemos representar y utilizar los grafos en Mathematica partiendo directamente de su definición (su conjunto de vértices y de lados) o como objetos de tipo grafo:

Ejemplo 5.1.

- a) Sea $G_1 = (W_1, F_1)$ el grafo dirigido con conjunto de vértices:

$$W_1 = \{1, 2, 3\},$$

y de flechas:

$$F_1 = \{(2, 1), (1, 3), (3, 2)\},$$

donde $e_1 = (2, 1)$, $e_2 = (1, 3)$ y $e_3 = (3, 2)$ (ver ilustración 5.3.).

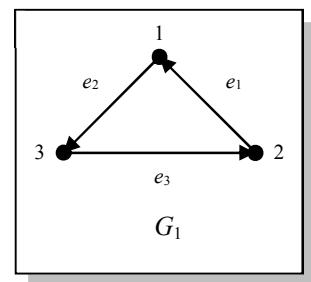


Ilustración 5.3.

En Mathematica lo introduciremos como sigue:

In[]:=

W1={1,2,3};

F1={2→1,1→3,3→2};

Y observemos:

- $F1[[i]]$ es e_i .
- $F1[[i,1]]$ y $F1[[i,2]]$ son el origen y el fin respectivamente de e_i .

Utilizando la función Graph[], definiríamos el objeto de tipo grafo asociado a G_1 de la siguiente manera:

```
In[]:= <<Combinatorica`  
In[]:= G1=Graph[{{2,1},{3,2},{1,3}},{{0,1},  
{1,0},{-1,0}},EdgeDirection->True]  
Out[]=- Graph:< 3,3,Directed >-
```

Donde:

```
In[]:= Edges[G1]  
Out[]={ {2,1}, {3,2}, {1,3} }
```

son las flechas y

```
In[]:= Vertices[G1]  
Out[]={ {0,1}, {1,0}, {-1,0} }
```

son las coordenadas en el plano de los tres vértices de G_1 . Observemos que $M[G1] = \text{Length}[Edges[G1]] = 3$ o que $V[G1] = \text{Length}[Vertices[G1]] = 5$.

b) Sea $G_2 = (W_2, F_2)$ el grafo de la ilustración 5.4.

Introducimos en Mathematica el conjunto de sus vértices y lados:

```
In[]:= W2={1,2,3,4,5};  
F2={1→2,2→3,3→4};
```

Observemos como el lado e_i es incidente con los vértices de $F2[[i]]$. Y los vértices $F2[[i,1]]$ y $F2[[i,2]]$ son adyacentes. Además el lado $e_1 = \{1, 2\}$ se puede introducir indistintamente como $1 \rightarrow 2$ o $2 \rightarrow 1$, $e_2 = \{3, 2\}$ como $3 \rightarrow 2$ o $2 \rightarrow 3$ y $e_3 = \{3, 4\}$ por $3 \rightarrow 4$ o $4 \rightarrow 3$.

Utilizando la función Graph[], definiríamos el objeto de tipo grafo asociado a G_2 de la siguiente manera:

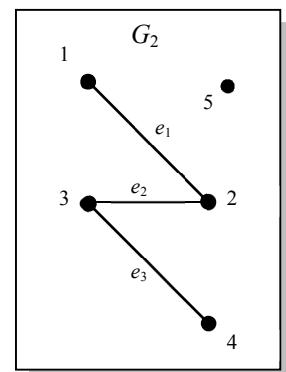


Ilustración 5.4.

```
In[]:= <<Combinatorica`
```

In[]:= **G2=Graph[{{1,2},{2,3},{3,4}},
{{0,2},{1,1},{0,1},{1,0},{1,2}}]**

Out[]=- Graph:< 3,5,Undirected >-

Donde:

In[]:= **Edges[G2]**

Out[]={ {1,2}, {2,3}, {3,4} *}*

son los lados y

In[]:= **Vertices[G2]**

Out[]={ {0,2}, {1,1}, {0,1}, {1,0}, {1,2} *}*

son las coordenadas en el plano de los 5 vértices de G_2 . □

Es importante resaltar que los lados de los grafos no orientados pueden representarse en Mathematica de dos formas distintas: $v \rightarrow w$ o $w \rightarrow v$ ($\{w, v\}$ o $\{v, w\}$ para los objetos de tipo grafo), aunque como lado del grafo que representan, estas dos notaciones son indistinguibles. Por tanto deberemos prestar atención a este detalle y tenerlo en consideración cuando deseemos resolver problemas asociados a grafos no orientados usando su definición. Además observemos que las dos formas de implementación de un grafo que hemos descrito son fácilmente intercambiables, esto es, podemos definir el objeto de tipo grafo desde los conjuntos de vértices y lados o flechas y viceversa (ejercicio 5.33).

2. REPRESENTACIÓN MATRICIAL

2.1. MATRIZ DE ADYACENCIA

Dado un (p, q) -grafo no orientado, llamaremos matriz de adyacencia a la matriz $A = (a_{ij})_{i,j \in \{1, \dots, p\}}$ con coordenadas:

$$a_{ij} = \begin{cases} 1 & \text{si los vértices } v_i \text{ y } v_j \text{ son adyacentes.} \\ 0 & \text{en otro caso.} \end{cases}$$

Si el grafo es dirigido entonces la matriz de adyacencia se define como sigue:

$$a_{ij} = \begin{cases} 1 & \text{si } (v_i, v_j) \in F. \\ 0 & \text{en otro caso.} \end{cases}$$

Si el grafo no es dirigido la matriz es simétrica y su diagonal principal se compondrá únicamente por ceros. Si el grafo es dirigido, la diagonal principal también se compone únicamente de ceros y por cada 1 en una posición (i, j) -ésima tendrá un 0 en la (j, i) -ésima puesto que no consideramos multigrafos. También podríamos generalizar el concepto a multigrafos, en donde la matriz de adyacencia ya no sería booleana, pero estos grafos no son objeto de estudio por nuestra parte. La matriz de adyacencia de un grafo no es única y depende del nombre u ordenación que asignemos a los vértices, esto se estudiará con más detalle más adelante en este mismo capítulo. Por otra parte, obsérvese que, aunque la matriz de adyacencia representa a un grafo determinado, ésta no contiene información alguna acerca de los nombres que tengan los vértices, para mayor comodidad y evitar ambigüedades identificaremos a los vértices por sus subíndices en vez de por sus nombres, esto es, si $W = \{v_1, v_2, \dots, v_p\}$ entonces entenderemos que el conjunto de vértices es $W = \{1, 2, \dots, p\}$ y si $v_i \rightarrow v_j$ es un lado o flecha, lo identificaremos con $i \rightarrow j$. Por tanto, para utilizar una expresión matricial de un grafo será imprescindible esta identificación. En cualquier función o programa en donde usemos la matriz de adyacencia del grafo, como forma de representación del mismo, obligatoriamente usaremos esta notación.

Calcularemos estas matrices con Mathematica definiendo la siguiente función para grafos no orientados:

| FUNCIÓN | COMENTARIOS |
|---|--|
| <pre>MATRIZADYACENCIA[W_,F_]:=Module[{i,j,k, CONTADORi,nuevoF}, nuevoF={}; Do[AppendTo[nuevoF, Position[W,F[[CONTADORi]][[1]][[1]][[1]]] →Position[W,F[[CONTADORi]][[2]][[1]][[1]]]; ,{CONTADORi,Length[F]}]; matrizadyacencia=Table[0,{i,Length[W]} ,{j,Length[W]}]; Do[matrizadyacencia[[nuevoF[[k,1]], nuevoF[[k,2]]]]=1; matrizadyacencia[[nuevoF[[k,2]], nuevoF[[k,1]]]]=1; ,{k,Length[F]}]; matrizadyacencia];</pre> | La función tendrá dos entradas: el conjunto de vértices “W” y el de lados “F”. Los vértices son identificados por sus subíndices utilizando la función Position[]. Si la identificación de vértices es previa a la aplicación de esta función, esta parte no es necesaria. |
| | Calcula la matriz de adyacencia. |
| | Salida de resultados. |

Función 5.1. Matriz de adyacencia (no orientado).

Y para grafos dirigidos:

| FUNCIÓN | COMENTARIOS |
|--|--|
| <pre>MATRIZADYACENCIADIR[W_,F_]:=Module[{i,j,k,CONTADORi,nuevoF}, nuevoF={}; Do[AppendTo[nuevoF, Position[W,F[[CONTADORi]][[1]][[1]][[1]]] →Position[W,F[[CONTADORi]][[2]][[1]][[1]]]; ,{CONTADORi,Length[F]}];</pre> | La función tendrá dos entradas: el conjunto de vértices “W” y el de flechas “F”. Los vértices son identificados por sus subíndices utilizando la función Position[]. Si la identificación de vértices es previa a la aplicación de esta función, esta parte no es necesaria. |

| | |
|---|---|
| <pre> matrizadyacencia=Table[0,{i,Length[W]} ,{j,Length[W]}]; Do[matrizadyacencia[[nuevoF[[k,1]]], nuevoF[[k,2]]]=1 ,{k,Length[F]}]; matrizadyacencia]; </pre> | Calcula la matriz de adyacencia. Salida de resultados. |
|---|---|

Función 5.2. Matriz de adyacencia de un grafo dirigido.

Más directamente usando la función SparseArray[] (no es habitual disponer de ella en otros lenguajes), podríamos realizar el cálculo como sigue, para los grafos no orientados:

| FUNCIÓN | COMENTARIOS |
|--|--|
| <pre> MATRIZADYACENCIA[W_,F_]:=Module[{i,sparsematriz,CONTADORi,nuevoF}, nuevoF={}; Do[AppendTo[nuevoF, Position[W,F][[CONTADORi]][[1]][[1]][[1]]] →Position[W,F][[CONTADORi]][[2]][[1]][[1]]; ,{CONTADORi,Length[F]}]; sparsematriz=SparseArray[Table[{nuevoF[[i]][[1]],nuevoF[[i]][[2]]}→1 ,{i,Length[F]}],{Length[W],Length[W]}; matrizadyacencia=sparsematriz+ Transpose[sparsematriz]; Normal[matrizadyacencia]]; </pre> | La función tendrá dos entradas: el conjunto de vértices “W” y el de lados “F”. Los vértices son identificados por sus subíndices utilizando la función Position[]. Si la identificación de vértices es previa a la aplicación de esta función, esta parte no es necesaria. Calcula la matriz de adyacencia. Salida de resultados. |
| | |
| | |

Función 5.1.bis. Matriz de adyacencia (no dirigidos).

Y para los grafos dirigidos:

| FUNCIÓN | COMENTARIOS |
|---|--|
| <pre> MATRIZADYACENCIADIR[W_,F_]:=Module[{i, CONTADORi,nuevoF}, nuevoF={}; Do[AppendTo[nuevoF, Position[W,F][[CONTADORi]][[1]][[1]][[1]]] →Position[W,F][[CONTADORi]][[2]][[1]][[1]]; ,{CONTADORi,Length[F]}]; matrizadyacencia=SparseArray[Table[{nuevoF[[i]][[1]],nuevoF[[i]][[2]]}→1 ,{i,Length[F]}],{Length[W],Length[W]}; Normal[matrizadyacencia]]; </pre> | La función tendrá dos entradas: el conjunto de vértices “W” y el de flechas “F”. Los vértices son identificados por sus subíndices utilizando la función Position[]. Si la identificación de vértices es previa a la aplicación de esta función, esta parte no es necesaria. Calcula la matriz de adyacencia. Salida de resultados. |
| | |
| | |

Función 5.2.bis Matriz de adyacencia de un grafo dirigido.

Observemos que en realidad, si los vértices son identificados por sus subíndices, la única información relevante de los vértices y necesaria para el cálculo de la matriz de adyacencia, es el número de vértices del grafo, esto es, es el número de elementos del conjunto W . Por tanto, con esta notación o identificación, podríamos definir la matriz de adyacencia directamente con el número de vértices y el conjunto de lados o flechas:

MATRIZADYACENCIA2[p_,F_]:=MATRIZADYACENCIA[Table[i,{i,p}],F];

Incluso podríamos simplificar las funciones 5.1. y 5.2. para que tuviesen por entradas el número de vértices y el conjunto de lados o flechas (siempre usando la notación de los subíndices para los vértices).

Las funciones relativas a grafos que incluye Mathematica, suelen aceptar como entrada, únicamente el conjunto de lados o flechas del grafo sin ningún otro dato adicional, sin embargo, en este planteamiento se escapa la posibilidad de vértices aislados, por lo que habitualmente nosotros utilizaremos para definir el grafo su conjunto de vértices y de lados o flecha. En consecuencia y a pesar de las bondades de otros planteamientos y notaciones, mantendremos la notación y la forma en que definimos el grafo en 5.1. y 5.2. (por su conjunto de vértices y su conjunto de lados o flechas) por cuestiones didácticas.

Recíprocamente también podemos determinar el conjunto de vértices y lados (resp. flechas) del grafo no orientado (resp. dirigido) desde su matriz de adyacencia, siempre será booleana y si es un grafo no orientado también deberá de ser simétrica. Obviamente y como comentábamos anteriormente, la matriz de adyacencia no contiene información alguna acerca de los nombres de los vértices, por tanto, desde la matriz de adyacencia recuperaremos el grafo, pero sus vértices estarán siempre identificados por sus subíndices.

| PROGRAMA | COMENTARIOS |
|--|--|
| matrizadyacencia=MATRIZ DE ADYACENCIA; | Introducimos la matriz de adyacencia de un grafo no orientado. |
| W=Table[i,{i,Dimensions[matrizadyacencia]][[1]]] | Se calcula el conjunto de vértices. |
| F={}; Do[Do[If[matrizadyacencia[[i,j]]==1, AppendTo[F,i→j]]; ,{i,j}]; ,{j,Dimensions[matrizadyacencia][[1]]}]; F | Se calcula el conjunto de lados. Se calcula el conjunto de flechas. |

Programa 5.3. Recuperamos un grafo no orientado desde la matriz de adyacencia.

| PROGRAMA | COMENTARIOS |
|---|--|
| matrizadyacencia=MATRIZ DE ADYACENCIA; | Introducimos la matriz de adyacencia de un grafo dirigido. |
| W=Table[i,{i,Dimensions[matrizadyacencia]][[1]]] | Se calcula el conjunto de vértices. |
| F={}; Do[Do[If[matrizadyacencia[[i,j]]==1, AppendTo[F,{i,j}]]; ,{i,Dimensions[matrizadyacencia][[1]]}]; ,{j,Dimensions[matrizadyacencia][[1]]}]; F | Se calcula el conjunto de flechas. |

Programa 5.4. Recuperamos un grafo dirigido desde la matriz de adyacencia.

Si deseamos recuperar el grafo desde la matriz de adyacencia con los nombre de los vértices originales, podremos hacerlo, pero deberemos de pasarlo al ordenador esta

información, si en el conjunto “antiguoW” tenemos almacenados estos nombres ordenados según los índices respectivos podremos deshacer la identificación:

```
In[7]:= antiguoW=CONJUNTO DE VÉRTICES;
antiguoF=Table[antiguoW[[F[[i]][[1]]]]
→antiguoW[[F[[i]][[2]]]],{i,Length[F]}];
```

En ocasiones, los lados o flechas también reciben nombre, sin embargo, por la forma en la que desde un principio hemos definido e introducido los grafos en el ordenador, no tratamos este extremo.

Desde la versión 6 de Mathematica, en el paquete <<GraphUtilities` disponemos de una función para calcular la matriz de adyacencia:

AdjacencyMatrix[F,n]

Donde “F” es el conjunto de lados o flechas y “n” es el número de vértices del grafo, aunque si no existen vértices aislados el segundo argumento puede ser omitido. Con esta función escribiremos

AdjacencyMatrix[F,n]+Transpose[AdjacencyMatrix[F,n]]

para obtener la matriz de adyacencia de un grafo no orientado²⁰. O únicamente:

AdjacencyMatrix[F,n]

si el grafo es dirigido. Es muy importante tener en cuenta que **AdjacencyMatrix[]** obligatoriamente identifica vértices con índices, como es de esperar, pero los índices se adjudican por criterios distintos a los comentados anteriormente en este libro: lo hace en el orden en el que aparecen en el conjunto de lados o flechas, es por eso que probablemente obtengamos una matriz de adyacencia distinta a la proporcionada por 5.1. o 5.2., aunque serán isomorfas (ver el epígrafe 4. de este mismo capítulo). La ventaja de este método reside en que no necesita el listado de vértices, el mayor inconveniente los tenemos en que si cogemos un conjunto *F* de lados o flechas y consideramos el mismo conjunto, pero cambiamos el orden en que aparecen estos lados o flechas, ambos son conjuntos iguales y representan al mismo grafo²¹, sin embargo, las matrices de adyacencia obtenidas con esta función, aunque son isomorfas, son distintas (véase la parte final del ejemplo 5.2.).

Además, la operación recíproca, esto es, recuperar el grafo desde la matriz de adyacencia, es muy distinta a como lo hemos hecho en los programas 5.3. o 5.4. y tenemos que tener muy presente cómo se eligieron los índices de los vértices con los que se han identificado. Para ello escribiremos:

²⁰ También funcionaría, si describimos cada lado del grafo por $v_i \rightarrow v_j$ y $v_j \rightarrow v_i$ en *F*, pero esto entra en conflicto con la descripción de grafo (no multigrafo) no orientado que se ha hecho en este libro.

²¹ Algo similar ocurre con las funciones 5.1. y 5.2. si cambiamos el orden de los vértices al definir el conjunto de vértices, aunque en este caso queda muy clara la permutación de vértices y en consecuencia el isomorfismo (ver epígrafe 4. de este capítulo) que relaciona las matrices de adyacencia obtenidas.

```
In]:= vertices={};  
Do[  
  añadir=Complement[{F[[i]][[1]],F[[i]][[2]]},  
    vertices];  
  vertices=Join[vertices,añadir];  
,{i,Length[F]}];  
W=vertices
```

Con este código obtenemos el conjunto de vértices ordenado según los índices adjudicados por `AdjacencyMatrix[]`, y ahora procedemos igual que se ha explicado antes con 5.3. y 5.4.

También podremos calcular las matrices de adyacencia de aquellos grafos que hayamos definido como objetos de tipo grafo (véase 1.1.2.), para este propósito disponemos de un par de funciones específicas dentro del paquete `<<Combinatorica``:

ToAdjacencyMatrix[Objeto de tipo grafo]

que calcula la matriz de adyacencia y:

FromAdjacencyMatrix[Matriz,Opciones]

cuya salida será el objeto de tipo grafo correspondiente. En opciones indicaremos el tipo de grafo con “*Type*”, tendrá dos posibles valores “*Directed*” o “*Undirected*”, aunque por defecto su valor será “*Undirected*”. De nuevo, con la matriz de adyacencia se pierde cualquier información relativa a los nombres de los vértices, lados o flechas y más particularmente de las posiciones en el plano de los vértices que tuviese el objeto de tipo grafo de partida.

Ejemplo 5.2. Las matrices de adyacencia de los grafos G_1 y G_2 del ejemplo 5.1. son respectivamente:

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad \text{y} \quad \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Definimos las funciones 5.1. y 5.2. y las calculamos con Mathematica:

```
In]:= MATRIZADYACENCIA[W_,F_]:=Module[{i,j,k},  
          :  
          :
```

```
In]:= MATRIZADYACENCIADIR[W_,F_]:=Module[{i,j,k},  
          :  
          :
```

⋮ ⋮

In[7]:= **W1={1,2,3};**
F1={2→1,1→3,3→2};
A=MATRIZADYACENCIADIR[W1,F1];
MatrixForm[A]

Out[7]=
$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

In[8]:= **W2={1,2,3,4,5};**
F2={1→2,2→3,3→4};
B=MatrixForm[MATRIZADYACENCIA[W2,F2]]

Out[8]=
$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Recíprocamente con el programa 5.4.,

In[7]:= **matrizadyacencia=A;**

⋮ ⋮

Out[7]= {1, 2, 3}
{2→1, 3→2, 1→3}

Y con el programa 5.3.,

In[7]:= **matrizadyacencia=B;**

⋮ ⋮

Out[7]= {1, 2, 3, 4, 5}
{1→2, 2→3, 3→4}

Probemos ahora el grafo no orientado $G_3 = (\{a, b, c, d, e\}, \{a \rightarrow b, b \rightarrow c, c \rightarrow d, d \rightarrow e, e \rightarrow a\})$:

In[7]:= **W={a,b,c,d,e};**
F={a→b,b→c,c→d,d→e,e→a};
matriz=MATRIZADYACENCIA[W,F]

```
Out]:= { {0,1,0,0,1}, {1,0,1,0,0}, {0,1,0,1,0},  
{0,0,1,0,1}, {1,0,0,1,0} }
```

Y si intentamos recuperar el grafo desde la matriz:

```
In]:= matrizadyacencia=matriz;
```

```
: : :
```

```
Out]:= {1,2,3,4,5}  
{1→2,2→3,3→4,1→5,4→5}
```

Y en efecto, como era de esperar, se ha perdido la información relativa al nombre de los vértices. Ahora podríamos recuperar, si aportamos de nuevo esa información perdida, el grafo original:

```
In]:= antiguoW={a,b,c,d,e}  
antiguoF=Table[antiguoW[[F[[i]][[1]]]]→  
antiguoW[[F[[i]][[2]]]],{i,Length[F]}]
```

```
Out]:= {a,b,c,d,e}  
{a→b,b→c,c→d,a→e,d→e}
```

Veámos cómo funciona `AdjacencyMatrix[]`, para ello vamos a considerar el grafo G_3 dado de otras dos formas distintas: (W, F_1) y (W, F_2) , donde F_1 y F_2 son conjuntos idénticos al de lados de G_3 , pero en el ordenador los introducimos ordenados de forma distinta a F :

```
In]:= F={a→b,b→c,c→d,d→e,e→a};  
F1={a→b,e→a,b→c,c→d,d→e};  
F2={d→e,b→c,a→b,c→d,e→a};
```

```
In]:= <<GraphUtilities`;
```

```
In]:= matriz1=MatrixForm[AdjacencyMatrix[F]+  
Transpose[AdjacencyMatrix[F]]]
```

Out]=
$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

```
In]:= matriz2=MatrixForm[AdjacencyMatrix[F1]+  
Transpose[AdjacencyMatrix[F1]]]
```

$$Out[] = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

In[]:= **matriz3=MatrixForm[AdjacencyMatrix[F2]+Transpose[AdjacencyMatrix[F2]]]**

$$Out[] = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Observemos que las tres matrices son distintas. Ahora recuperamos el grafo desde cada una de las tres matrices: “matriz1”, “matriz2” y “matriz3”, para ello tenderemos que comprobar previamente la asignación de índices que en cada caso realizó la función **AdjacencyMatrix[]**:

- Desde “matriz1”,

In[]:= **vertices={};
Do[
 añadir=Complement[{F[[i]][[1]],F[[i]][[2]]},vertices];
 vertices=Join[vertices,añadir];
 ,{i,Length[F]}];
 vertices**

Out[] = {a, b, c, d, e}

Ahora usamos el programa 5.3,

In[]:= **matrizadyacencia=matriz1;**

⋮ ⋮

Out[] = {1, 2, 3, 4, 5}
{1→2, 2→3, 3→4, 1→5, 4→5}

Asignamos a cada índice el nombre del vértice:

In[]:= **antiguoW=vertices;
antiguoF=Table[antiguoW[[F[[i]][[1]]]]->
antiguoW[[F[[i]][[2]]]],{i,Length[F]}]**

Out[] = {a → b, b → c, c → d, a → e, d → e}

- Desde “matriz2”,

```
In[]:= vertices={};  
Do[  
    añadir=Complement[{F1[[i]][[1]],F1[[i]][[2]]},vertices];  
    vertices=Join[vertices,añadir];  
,{i,Length[F]}];  
vertices
```

Out[] = {a, b, e, c, d}

Ahora usamos el programa 5.3,

```
In[]:= matrizadyacencia=matriz2;
```

⋮ ⋮

*Out[] = {1, 2, 3, 4, 5}
{1 → 2, 1 → 3, 2 → 4, 3 → 5, 4 → 5}*

Asignamos a cada índice el nombre del vértice:

```
In[]:= antiguoW=vertices;  
antiguoF=Table[antiguoW[[F[[i]][[1]]]]->  
    antiguoW[[F[[i]][[2]]]],{i,Length[F]}]
```

Out[] = {a → b, a → e, b → c, e → d, c → d}

- Desde “matriz3”,

```
In[]:= vertices={};  
Do[  
    añadir=Complement[{F2[[i]][[1]],F2[[i]][[2]]},vertices];  
    vertices=Join[vertices,añadir];  
,{i,Length[F]}];  
vertices
```

Out[] = {d, e, b, c, a}

Ahora usamos el programa 5.3,

```
In[]:= matrizadyacencia=matriz3;
```

⋮ ⋮

```
Out[] = {1,2,3,4,5}
           {1→2,1→4,3→4,2→5,3→5}
```

Asignamos a cada índice el nombre del vértice:

```
In[] := antiguoW=vertices;
antiguoF=Table[antiguoW[[F[[i]][[1]]]]->
antiguoW[[F[[i]][[2]]]],{i,Length[F]}]
```

```
Out[] = {d→e,d→c,b→c,e→a,b→a}
```

Por último definimos los tres grafos como objetos de tipo grafo, calculamos su matriz de adyacencia y recuperamos el grafo desde ésta, para G_3 las coordenadas de los vértices en el plano podemos asignarlas como queramos, ya que no partimos de una representación gráfica específica:

```
In[] := <<Combinatorica`>
In[] := G1=Graph[{{2,1},{3,2},{1,3}},
            {{0,1},{1,0},{-1,0}},EdgeDirection->True]
G2=Graph[{{1,2},{2,3},{3,4}},
            {{0,2},{1,1},{0,1},{1,0},{1,2}}]
G3=Graph[{{1,2},{2,3},{3,4},{4,5},{5,1}},
            {{0,1},{0,-1},{1,0},{-1,0},{0,0}}]
Out[] = - Graph:< 3,3,Directed >-
           - Graph:< 3,5,Undirected >-
           - Graph:< 5,5,Undirected >-
In[] := A1=ToAdjacencyMatrix[G1]
Out[] = {{0,0,1},{1,0,0},{0,1,0}}
In[] := A1=ToAdjacencyMatrix[G2]
Out[] = {{0,1,0,0,0},{1,0,1,0,0},{0,1,0,1,0},
           {0,0,1,0,0},{0,0,0,0,0}}
In[] := A1=ToAdjacencyMatrix[G2]
Out[] = {{0,1,0,0,1},{1,0,1,0,0},{0,1,0,1,0},
           {0,0,1,0,1},{1,0,0,1,0}}
```

Ahora recuperamos los conjuntos de vértices con **Vertices[]** (en realidad sus coordenadas en el plano porque están identificados por sus posiciones o por sus índices respectivos en el listado de vértices) y lados con **Edges[]** de los grafos, desde las matrices de adyacencia usando **FromAdjacencyMatrix[]**:

```
In[] := Vertices[FromAdjacencyMatrix[A1,Type->Directed]]
```

```

Edges[FromAdjacencyMatrix[A1,Type->Directed]]

Out[] = { {-0.5, 0.866025}, {-0.5, -0.866025}, {1., 0} }
{ {1, 3}, {2, 1}, {3, 2} }

In[] := Vertices[FromAdjacencyMatrix[A2]]
Edges[FromAdjacencyMatrix[A2]]

Out[] = { {0.309017, 0.951057}, {-0.809017, 0.587785},
{-0.809017, -0.587785}, {0.309017, -0.951057},
{1., 0} }
{ {1, 2}, {2, 3}, {3, 4} }

In[] := Vertices[FromAdjacencyMatrix[A1,Type->Directed]]
Edges[FromAdjacencyMatrix[A1,Type->Directed]]

Out[] = { {0.309017, 0.951057}, {-0.809017, 0.587785},
{-0.809017, -0.587785}, {0.309017, -0.951057},
{1., 0} }
{ {1, 2}, {1, 5}, {2, 3}, {3, 4}, {4, 5} }

```

□

2.2. MATRIZ DE INCIDENCIA

Dado un (p, q) -grafo no orientado G , llamaremos matriz de incidencia a la matriz

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1q} \\ a_{21} & a_{22} & \cdots & a_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ a_{p1} & a_{p2} & \cdots & a_{pq} \end{pmatrix},$$

con coordenadas:

$$a_{ij} = \begin{cases} 1 & \text{si el lado } e_j \text{ es incidente con el vértice } v_i, \\ 0 & \text{en otro caso.} \end{cases}$$

En la matriz de incidencia, cada columna tendrá dos unos y el resto ceros (puesto que nuestros grafos son simples), además no es única, y dependerá del orden o nombre que demos a los vértices y los lados del grafo. Si el grafo es dirigido no podríamos representarlo mediante una matriz de incidencia booleana. Nos limitaremos a estudiar las matrices de incidencia únicamente de los grafos no orientados. Como les ocurre a las matrices de adyacencia, las de incidencia tampoco contienen información relativa a los nombres de los vértices o lados, por tanto, de nuevo usaremos la identificación de los vértices con sus subíndices.

Para determinar la matriz de incidencia de un grafo utilizaremos la siguiente función:

| FUNCIÓN | COMENTARIOS |
|---|---|
| MATRIZINCIDENCIA[W_,F_]:=Module[{i,j, CONTADORi,nuevoF}, | La función tendrá dos entradas: el conjunto de vértices “W” y el de lados “F”. |
| nuevoF={}; Do[AppendTo[nuevoF, Position[W,F[[CONTADORi]][[1]][[[1]]][[1]]] →Position[W,F[[CONTADORi]][[2]][[[1]]][[1]]]; ,{CONTADORi,Length[F]}]; | Los vértices son identificados por sus subíndices utilizando la función Position[]. Si la identificación de vértices es previa a la aplicación de esta función, esta parte no es necesaria. |
| matrizincidencia=Table[0 ,{i,Length[W]},{j,Length[F]}]; Do[Do[matrizincidencia[[F[[j]][[i]],j]]=1; ,{i,2}];,{j,Length[F]}]; matrizincidencia]; | Calcula la matriz de incidencia. |
| | Salida de resultados. |

Función 5.5. Matriz de incidencia (no orientados).

Y recíprocamente, determinaremos el grafo desde su matriz de incidencia:

| PROGRAMA | COMENTARIOS |
|--|---|
| matrizincidencia=MATRIZ DE INCIDENCIA; | Introducimos la matriz de incidencia de un grafo. |
| W=Table[i,{i,Dimensions[matrizincidencia]][[1]]}] | Se calcula el conjunto de vértices. |
| F={}; Do[AppendTo[F, Position[Transpose[matrizincidencia][[i]],1][[1]][[1]]]→ Position[Transpose[matrizincidencia][[i]],1][[2]][[1]]] ,{i,Dimensions[matrizincidencia]][[2]]}]; F | Se calcula el conjunto de lados. |

Programa 5.6. Recuperamos un grafo desde la matriz de incidencia.

También podremos calcular las matrices de incidencia de aquellos grafos que hayamos definido como objetos (véase 1.1.2.), para este propósito disponemos una función específica a partir de la versión 6 de Mathematica:

IncidenceMatrix[Objeto de tipo grafo]

dentro del paquete `<<Combinatorica``.

Ejemplo 5.3. La matriz de incidencia del grafo G_2 del ejemplo 5.1. (ilustración 5.2.) será:

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

Con Mathematica la calculamos usando la función 5.5.:

In[]:= **MATRIZINCIDENCIA[W_,F_]:=Module[{i,j},**

⋮ ⋮

In[]:= **W2={1,2,3,4,5};**
F2={1→2,2→3,3→4};
F=MATRIZINCIDENCIA[W2,F2]
MatrixForm[F]

$$Out[] = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

Recíprocamente con el programa 5.6.,

In[]:= **matrizincidencia=F;**

⋮ ⋮

Out[] = **{1,2,3,4,5}**
{1→2,2→3,3→4}

Considerando al grafo como objeto de tipo grafo podemos hacer el mismo cálculo directamente usando la función IncidenceMatrix[]:

In[]:= **<<Combinatorica`**

In[]:= **G2=Graph[{{1,2},{2,3},{3,4}},**
{1,2},{1,1},{0,1},{1,0},{1,2}}];

In[]:= **MatrixForm[IncidenceMatrix[G2]]**

$$Out[7]= \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

□

3. REPRESENTACIÓN GRÁFICA CON MATHEMATICA

Los grafos podemos representarlos además de por su definición o matricialmente, como se ha visto, también gráficamente. Desafortunadamente la representación gráfica de un mismo grafo puede aparentemente resultar muy distinta. Existen distintos métodos habituales para la representación gráfica de un grafo, aunque siempre podremos representarlo como personalmente más nos apetezca: dibujando los vértices en las coordenadas que consideremos oportunas y los lados o flechas representarlos por líneas rectas o curvas cuyo recorrido decidiríamos arbitrariamente. En este epígrafe intentaremos resolver este problema de la forma más simple posible, sin meternos en detalles y resolviendo sólo los problemas que son interesantes para nuestros intereses. Dependiendo de la forma en que hayamos definido el grafo, representaremos gráficamente los grafos de dos formas distintas:

- I. Si partimos de un objeto de tipo grafo (sólo a partir de la versión 6 de Mathematica), entonces usaremos la función

ShowGraph[GRAFO,OPCIONES]

que se encuentra dentro del paquete de funciones: <<Combinatorica`.

La salida será la representación gráfica del grafo y tendrá dos argumentos, el primero será un objeto de tipo grafo y en el segundo incluiremos todas las opciones que consideremos oportunas para su representación: “*VertexColor*”, “*VertexStyle*”, “*VertexNumber*”, “*VertexNumberColor*”, “*VertexNumberPosition*”, “*VertexLabel*”, “*VertexLabelColor*”, “*VertexLabelPosition*”, “*EdgeColor*”, “*EdgeStyle*”, “*EdgeLabel*”, “*EdgeLabelColor*”, “*EdgeLabelPosition*”, “*LoopPosition*” y “*EdgeDirection*”; estas opciones pueden especificarse en ShowGraph[] para representar el grafo o directamente en la Graph[] al definir el objeto de tipo grafo, y podemos encontrar una descripción de todas ellas en 1.1.2., donde se describe la función Graph[].

- II. Si el grafo lo hemos definido directamente o por su matriz de adyacencia, utilizaremos la función de Mathematica²²

²² En la versión 5.2 de Mathematica, para la que originalmente fueron pensados todos los métodos expuestos en la primera edición de este libro, la función GraphPlot[] está disponible en el paquete de funciones: <<“DiscreteMath’GraphPlot” y además difiere de la incluida en la versión 6.0, donde además no es necesario incluir este paquete para poder utilizar dicha función, estando ésta disponible directamente.

GraphPlot[GRAFO,OPCIONES]

La función devolverá un objeto de tipo “Graphics”, esto es, una representación gráfica del grafo, y tiene dos argumentos que analizamos por separado:

- a) En el primer argumento incluiremos el grafo, por suerte las nociones de grafo no orientado y grafo dirigido que maneja Mathematica coinciden con los expuestos en el epígrafe 1 de este capítulo, por tanto serán fácilmente manejables. De las distintas representaciones de grafos que hemos visto, Mathematica admite tres:
 - 1) La matriz de adyacencia. El inconveniente de esta representación, como ya se ha comentado, es que se pierde el nombre de los vértices y de los lados, quedando los vértices identificados por su índice y los lados o flechas respectivamente como pares no ordenados u ordenados de vértices.
 - 2) La lista de lados o flechas F , esta representación presenta algunos problemas:
 - La representación de vértices aislados: por defecto estos no serán representados, ya que de hecho dicha información no está contenida en F . Podemos solventarlo incluyendo en F para cada vértice v aislado la lista $\{v, v\}$, pero atención, esto sólo debemos hacerlo con el fin de representar gráficamente el grafo, ya que tal par será interpretado como un lazo por el resto de rutinas y programas, por lo que deberemos suprimirlo para otros cálculos.
 - Al usar esta segunda opción también deberemos tener cuidado con la asignación de índices, pues Mathematica los asignará por el orden en el que aparezcan los vértices en la definición de los lados o flechas; por ejemplo si $F = \{\mathbf{b} \rightarrow \mathbf{c}, \mathbf{a} \rightarrow \mathbf{c}\}$, \mathbf{a} será el vértice 3, \mathbf{b} el 1 y \mathbf{c} el 2, y esto lo hará así siempre, independientemente del nombre que asignemos a los vértices, incluso en el caso en el que asignemos nombres numéricos.
 - No queda constancia de un nombre para los lados o flechas.
 - 3) ²³La lista formada por pares cuya primera componente son los lados o flechas de F , y cuya segunda es el nombre o etiqueta asociado al lado o flecha:

$$\{\{v_{i1} \rightarrow v_{j1}, \text{nombre1}\}, \{v_{i2} \rightarrow v_{j2}, \text{nombre2}\}, \dots, \{v_{ik} \rightarrow v_{jk}, \text{nombrek}\}\}$$

Esta representación presenta los mismos problemas que la anterior.

Como podemos observar, de las tres, la representación matricial es menos problemática, por tanto, siempre que podamos, por comodidad, nos inclinaremos preferentemente por esta representación.

²³ No está disponible en la versión 5.2 de Mathematica o anteriores.

- b) En el segundo argumento se incluirán las opciones²⁴ para la representación del grafo, éstas son múltiples y entre ellas se encuentran las que afectan a cualquier objeto gráfico. Mostramos las más interesantes para nuestros propósitos, pero no las analizamos con profundidad:
- PlotStyle*, indica propiedades gráficas a tener en cuenta para el dibujo de objetos gráficos, no es exclusiva de `GraphPlot[]` y es usada por cualquier función de representación gráfica. Entre otras, las opciones gráficas que habitualmente modificaremos son: la del color (“*Red*” para el rojo, “*Blue*” para el azul,...; o `RGB[]`) y el tamaño de los puntos (`PointSize[]`).
 - BaseStyle*, marca estilos básicos de representación como el color, la fuente o el tamaño del texto,...
 - VertexRenderingFunction*, es exclusiva de `GraphPlot[]` e indicará las propiedades de dibujo de los vértices, las más habituales que usaremos serán: la del color y la del objeto a dibujar como vértice (disco, círculo, nombre del vértice,...). Admite las siguientes opciones son:
 - Automatic*, etiquetará cada vértice con su índice.
 - $\{\}\&$, no hará nada.
 - Una función suministrada por el usuario con dos parámetros que serán respectivamente, las coordenadas del vértice y el índice o nombre, para ello usaremos la función de Mathematica:

Function[{parámetros},cuerpo]

o el método abreviado, donde los parámetros serán sustituidos por # ó #1, #2; donde # y #1 representarán las coordenadas y #2 el índice o nombre del vértice. Por ejemplo, si queremos que dibuje un disco azul en cada vértice con el índice o nombre en color amarillo dentro, escribiremos:

**VertexRenderingFunction→
Function[{i,j},{Blue,Disk[i,0.05],Yellow,Text[j,i]}]**

O, por el método abreviado:

**VertexRenderingFunction→
({Blue,Disk[#,0.05],Yellow,Text[#2,#1]}&)**

- VertexLabeling*, es exclusiva de `GraphPlot[]` y especifica si el nombre los vértices debe ser incluido en la representación gráfica, las opciones son: “*True*”, “*False*”, “*Tooltip*”, “*All*” y “*Automatic*”.
- VertexCoordinateRules*, indicará las coordenadas en las que los vértices deben ser dibujados. La usaremos de la siguiente manera:

VertexCoordinateRules→{v₁→{x₁,y₁}, v₂→{x₂,y₂},..., v_p→{x_p,y_p}}

²⁴ En la versión 5.2 o anteriores de Mathematica estas opciones difieren.

Donde (x_i, y_i) son las coordenadas del plano donde queremos que se dibuje el vértice v_i .

- f. *EdgeRenderingFunction*, es exclusiva de *GraphPlot[]*, indicará las propiedades de dibujo de los lados o flechas, normalmente incluiremos el color y el objeto a dibujar (línea, flecha, nombre del lado o flecha,...). Admite las siguientes opciones son:

- i. *Automatic*, etiquetará cada vértice con su índice.
 - ii. Una función suministrada por el usuario con tres parámetros que serán respectivamente, las coordenadas de los vértices inicial y final del lado o flecha, un par formado por los índices o nombres de los vértices inicial y final del lado o flecha y el nombre del lado.
- Para ello usaremos la función de Mathematica:

Function[{parámetros},cuerpo]

o el método abreviado, donde los parámetros serán sustituidos por # ó #1, #2, #3; donde # o #1, #2 y #3 representan los valores de los tres parámetros que definen cada lado o flecha. Por ejemplo, si queremos que dibuje una línea roja con el par de vértices que la definen como lado en el centro en color azul, escribiremos:

```
EdgeRenderingFunction→
Function[{i,j},{Red,Line[i],Blue,Text[j,
{(i[[1]][[1]]+i[[2]][[1]])/2,(i[[1]][[2]]+i[[2]][[2]])/2}]}]
```

O, por el método abreviado:

```
EdgeRenderingFunction→
({Red,Line[#1],Blue,Text[#2,
{(#1[[1]][[1]]+#1[[2]][[1]])/2,
(#1[[1]][[2]]+#1[[2]][[2]])/2}]}&)
```

- g. *EdgeLabeling*, es exclusiva de *GraphPlot[]* y funciones relacionadas, especifica si el nombre los lados o flechas debe ser incluido en la representación gráfica, las opciones son: “True”, “False” y “Automatic”.
- h. *MultiedgeStyle*, para representar multigrafos, en este libro estos grafos no son estudiados.
- i. *SelfLoopStyle*, para representar lazos (lados o flechas que empiezan y terminan en el mismo vértice), si bien, en este libro los grafos que contienen lazos no son estudiados.
- j. *Method*, indicará el método o algoritmo específico que se usará para la representación.”Automatic” elegirá el algoritmo de forma automática, estos algoritmos no son objeto de estudio por nuestra parte por lo que nos limitaremos a mencionar algunos:
- “CircularEmbedding”, “RandomEmbedding”,
 “HighDimensionalEmbedding”, “RadialDrawing”,
 “SpringEmbedding”, “SpringElectricalEmbedding”,...

- k. *PackingMethod*, para indicar cómo deben representarse las distintas componentes conexas (véase el capítulo 7) del grafo.
- l. *DirectedEdges*, puede tomar los valores “True” o “False” e indica si el grafo es dirigido o no orientado,
- m. Otras opciones interesantes son *Frame*, *FrameTicks*, *DataRange*, *PlotRange*, *PlotRangeMethod*, ...; para más detalle sobre éstas otras puede consultarse la ayuda de Mathematica.

Para familiarizarnos con ambas funciones, veamos algunos ejemplos:

Ejemplo 5.4. Representamos el grafo (no orientado) cuya matriz de adyacencia es:

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

Calculamos los vértices y lados usando el programa 5.3.:

In[]:= **matrizadyacencia=Table[If[i==j,0,1],{i,5},{j,5}];**

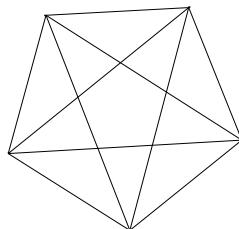
⋮ ⋮

Out[]= **{1, 2, 3, 4, 5}**
{1→2, 1→3, 2→3, 1→4, 2→4, 3→4, 1→5, 2→5, 3→5, 4→5}

Y lo representamos, indistintamente podemos usar como argumento “matrizadyacencia” o “F” (recordemos las ventajas e inconvenientes de cada uno):

In[]:= **GraphPlot[matrizadyacencia]**

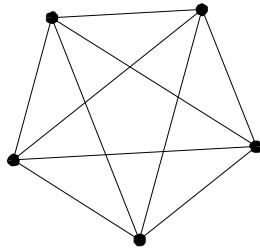
Out[]=



Podemos destacar los vértices:

In[]:= **GraphPlot[F, PlotStyle→{Black,PointSize[.03]}]**

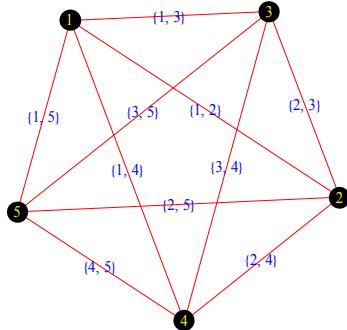
Out[]=



Dependiendo de las necesidades y gustos personales podemos dibujarlo como deseemos, aquí mostramos otra posibilidad más elaborada:

```
In]:= GraphPlot[F,
VertexRenderingFunction $\rightarrow$ {Black,Disk[#,0.04],
Yellow,Text[#2,#1]}&),
EdgeRenderingFunction $\rightarrow$ {Red,Line[#1],Blue,
Text[#2,{(#1[[1]][[1]]+#1[[2]][[1]])/2,
(#1[[1]][[2]]+#1[[2]][[2]])/2}}}&),
BaseStyle $\rightarrow$ {FontSize $\rightarrow$ 12}]
```

Out]=



Si optamos por definir un objeto de tipo grafo y usar la función ShowGraph[], entonces:

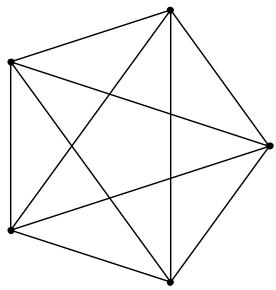
In]:= <<Combinatorica`

In]:= G=FromAdjacencyMatrix[matrizadyacencia]

Out]= - Graph:<10,5,Undirected>-

In]:= ShowGraph[G]

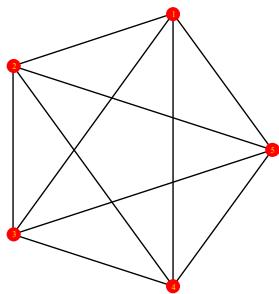
Out]=



Como antes podemos añadir opciones de representación para los vértices:

In[]:= **ShowGraph[G,VertexColor->Red,**
VertexStyle->PointSize[0.05],
VertexNumber->True,
VertexNumberColor->Yellow,
VertexNumberPosition->{+.01,+.0}]

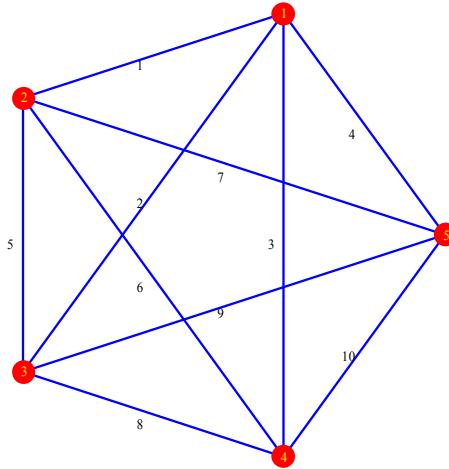
Out[]=



Y para los lados o flechas:

In[]:= **ShowGraph[G,VertexColor->Red,**
VertexStyle->PointSize[0.05],
VertexNumber->True,
VertexNumberColor->Yellow,
VertexNumberPosition->{+.01,+.0},
EdgeColor->Blue,EdgeLabel->True,
EdgeColor->Blue,EdgeLabel->True]

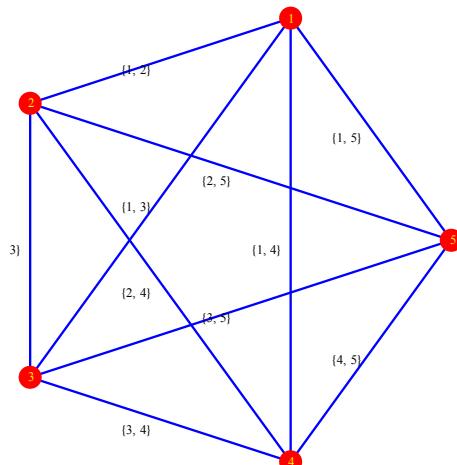
Out[]=



Si queremos mostrar la descripción de los lados como antes:

```
In[7]:= ShowGraph[
  Graph[Table[{Edges[G][[i]],
    EdgeLabel->Edges[G][[i]],{i,M[G]}],G[[2]]],
  VertexColor->Red, VertexStyle->PointSize[0.05],
  VertexNumber->True,VertexNumberColor->Yellow,
  VertexNumberPosition->{+.01,.0},
  EdgeColor->Blue,EdgeLabel->True]
```

Out[7]=



□

Ejemplo 5.5. Representamos el grafo dirigido cuya matriz de adyacencia es:

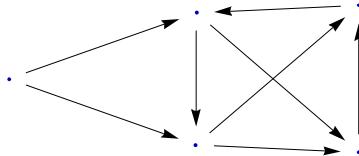
$$A = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

En primer lugar introducimos la matriz de adyacencia y determinamos los vértices y las flechas usando el programa 5.4.:

```
In[]:= matrizadyacencia={{0,0,0,1,1},{1,0,0,1,0},
{1,0,0,0,0},{0,0,1,0,1},{0,0,1,0,0}};
          :
          :
Out[]= {1,2,3,4,5}
{2→1,3→1,4→3,5→3,1→4,2→4,1→5,4→5}
```

Como primer argumento de `GraphPlot[]` usaremos indistintamente “matrizadyacencia” o “F” (salvo cuando tengamos vértices aislados que nos limitaremos a usar la matriz de adyacencia). Para representar un grafo dirigido tendremos que tener siempre la precaución de indicarle al ordenador que dibuje flechas, pues por defecto no lo hará, bastaría con indicárselo mediante “EdgeRenderingFunction”:

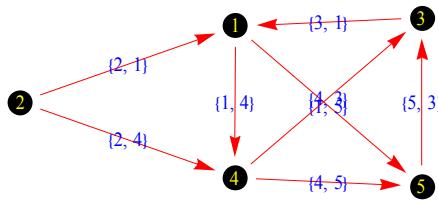
```
In[]:= GraphPlot[matrizadyacencia,
EdgeRenderingFunction→({Arrow[#1,0.1]}&)]
Out[=
```



Si queremos también podemos, como en el ejemplo anterior, ser más explícitos en nuestra representación y variar el color o tamaño de los vértices y de las flechas, también podemos indicarle que identifique los vértices y lados o flechas por su nombre:

```
In[]:= GRAFO=matrizadyacencia;
GraphPlot[GRAFO,
VertexRenderingFunction→({Black,Disk[#,0.06],
Yellow,Text[#,#1]}&),
EdgeRenderingFunction→({Red,Arrow[#1,0.1],Blue,
Text[#,{{#1[[1]][[1]]+#1[[2]][[1]]}/2,
(#1[[1]][[2]]+#1[[2]][[2]])/2}}&),
BaseStyle→{FontSize→12}]
```

Out[]=



Y si optamos por definir un objeto de tipo grafo y usar la función ShowGraph[], entonces:

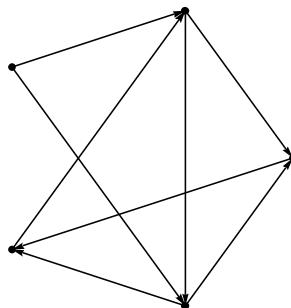
In[]:= <<Combinatorica`

In[]:= G=FromAdjacencyMatrix[matrizadyacencia,
Type->Directed]

Out[]= - Graph:<8,5,Directed>-

In[]:= ShowGraph[G]

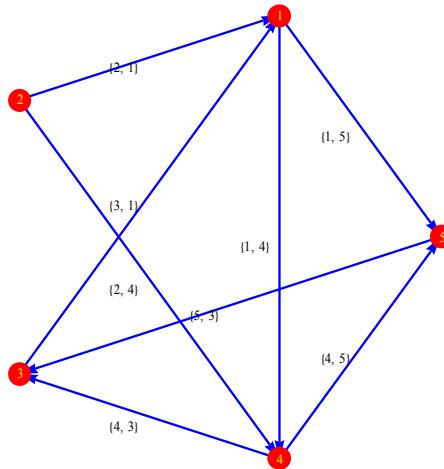
Out[]=



Y si queremos añadir algunas opciones:

In[]:= ShowGraph[
Graph[Table[{Edges[G][[i]],
EdgeLabel->Edges[G][[i]],{i,M[G]}},G[[2]]],
EdgeDirection->True],
VertexColor->Red, VertexStyle->PointSize[0.05],
VertexNumber->True, VertexNumberColor->Yellow,
VertexNumberPosition->{+.01,+.0},
EdgeColor->Blue, EdgeLabel->True]

Out[]=



□

Ejemplo 5.6. Representamos el grafo G_2 del ejemplo 5.1. (ilustración 5.2.). Partimos de la matriz de adyacencia y lo representamos:

Definimos la función 5.1. y el paquete de funciones de Matemática Discreta:

In[]:= **MATRIZADYACENCIA[W_,F_]:=Module[{i,j,k},**

⋮ ⋮

In[]:= **W={1,2,3,4,5};**
F={1→2,2→3,3→4};
GRAFO=MATRIZADYACENCIA[W,F];
GraphPlot[GRAFO, VertexRenderingFunction→
 $\{ \text{BlackDisk}[\#, .04], \text{Yellow}, \text{Text}[\#2, \#1] \} \&)]$

Out[]=

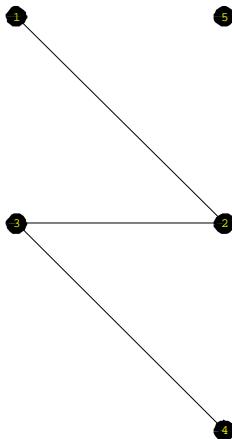


Como podemos observar la representación es muy distinta de la ilustración 5.2., si queremos que la representación gráfica se haga de una forma particular, por ejemplo, coincidiendo con la ilustración, podemos indicar al ordenador las coordenadas de los vértices manualmente:

In[]:= **GRAFO=MATRIZADYACENCIA[W,F];**
coordenadas={{0,2},{1,1},{0,1},{1,0},{1,2}};
GraphPlot[GRAFO,
 $\text{VertexRenderingFunction}\rightarrow(\{\text{BlackDisk}[\#, .04],$

**Yellow,Text[#, #1] &),
VertexCoordinateRules → coordenadas]**

Out[] =



Con ShowGraph[] no tenemos este problema puesto que los objetos de tipo grafo se definen especificando las coordenadas en el plano de los vértices. Si el grafo fuese dado desde la matriz de adyacencia y quisieramos concretar una representación gráfica en particular podríamos hacerlo de la siguiente manera:

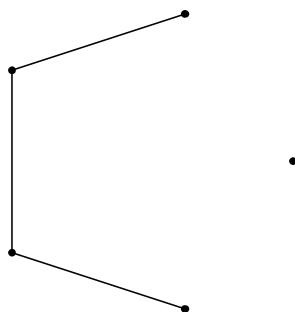
In[] := <<Combinatorica`

In[] := G=FromAdjacencyMatrix[matrizadyacencia]

Out[] = - Graph: <3, 5, Undirected> -

In[] := ShowGraph[G]

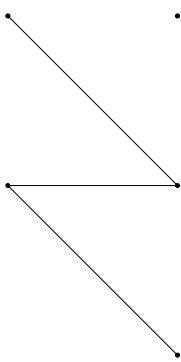
Out[] =



Y para representarlo gráficamente como en la ilustración 5.4.:

In[1]:=

```
NuevoG=Graph[G[[1]],
{{{0,2}},{{1,1}},{{0,1}},{{1,0}},{{1,2}}}];
ShowGraph[G]
```

Out[1]=

□

Existen otras funciones para la representación gráfica:

- **GraphPlot3D[GRAFO,OPCIONES]** y **GraphCoordinates3D[GRAFO,OPCIONES]**, análogas a las estudiadas que representan grafos en tres dimensiones.
- **ShowGraphArray[matriz]**, pertenece al paquete <<Combinatorica` y muestra la matriz de grafos “matriz” (de 1 y 2 dimensiones).
- **AnimateGraph[listado]**, pertenece al paquete <<Combinatorica` y se usa para generar una animación desde un listado de grafos.

Hay otras funciones dentro de los mismos paquetes relacionadas con el tema, pueden consultarse en la ayuda de Mathematica, no las analizaremos y las dejamos para un lector más interesado en la representación gráfica. Nos limitaremos al uso de las estudiadas y en particular a los métodos expuestos en los ejemplos, estos cubren todas las necesidades de representación gráfica que se nos presentarán en este libro.

4. GRAFOS ISOMORFOS

Como hemos comentado, tanto al representar grafos por su definición, como matricialmente, tenemos el problema de los índices o nombres que asignemos a cada vértice. Para un mismo grafo obtendremos expresiones matriciales distintas según la asignación de índices o el orden en que tomemos sus vértices, incluso los conjuntos de vértices y de lados o flechas aparecerán ser muy distintos. Por otra parte, las representaciones gráficas se realizan con un grado de libertad total, tanto a la hora de elegir la posición de los vértices como el trazado de los lados o flechas, incluso si asignamos los mismos nombres o índices para los vértices, pueden ser muy distintas. Por ejemplo, si

consideraremos G_1 y G_2 los dos grafos no orientados de la ilustración 5.5., sus representaciones gráficas son muy distintas:

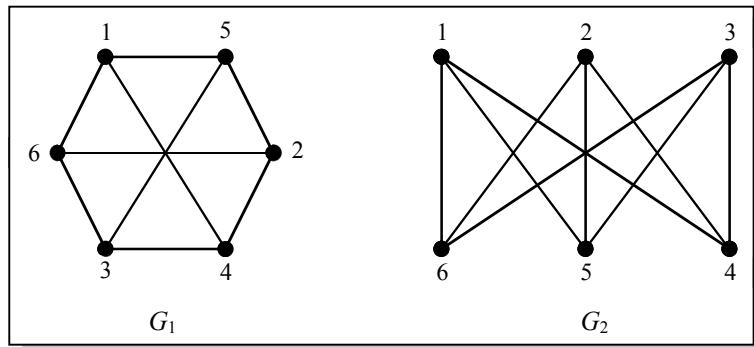


Ilustración 5.5.

Sin embargo, calculamos sus matrices de adyacencia y observamos que ambas coinciden con:

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

En resumen, estamos observando situaciones en las que salvo los nombres que asignemos a los vértices o lados, o la representación gráfica particular que hagamos, estamos tratando con los mismos grafos. Cuando tengamos dos o más grafos en una de estas situaciones diremos que son grafos isomorfos, que formalmente para evitar ambigüedades definiremos como sigue:

Dos grafos no orientados, $G_1 = (W_1, F_1)$ y $G_2 = (W_2, F_2)$ (resp. digrafos) diremos que son isomorfos si existe una aplicación $f: W_1 \rightarrow W_2$ biyectiva tal que, $\{v, w\} \in F_1$ (resp. $(v, w) \in F_1$) si y sólo si $\{f(v), f(w)\} \in F_2$ (resp. $(f(v), f(w)) \in F_2$).

Con Mathematica podemos comprobar si una aplicación entre grafos define un isomorfismo estudiando directamente si verifica la definición. Para grafos no orientados:

| PROGRAMA | COMENTARIOS |
|---|--|
| <pre> W1=CONJUNTO DE VÉRTICES DEL PRIMER GRAFO; F1=CONJUNTO DE LADOS DEL PRIMER GRAFO; W2=CONJUNTO DE VÉRTICES DEL SEGUNDO GRAFO; F2=CONJUNTO DE LADOS DEL SEGUNDO </pre> | Definición de los grafos G_1 y G_2 . |

| | |
|---|---|
| GRAFO; | |
| DEFINICIÓN DE LA APLICACIÓN <code>funcion[]</code> ENTRE LOS CONJUNTOS DE VERTICES; | Definición de la aplicación. |
| Isomorfismo=True; | Suponemos por defecto que si lo es. |
| If[<code>Length[W1]==Length[Table[funcion[W1[[i]],{i,Length[W1]}]]&& Length[W1]==Length[W2]&& Length[F1]==Length[F2],</code> | Comprobamos que “funcion” es biyectiva. |
| Do[<code>If[Length[Intersection[{funcion[F1[[i]][[1]]]>>funcion[F1[[i]][[2]]],funcion[F1[[i]][[2]]]>>funcion[F1[[i]][[1]]}],F2]==1</code> <code>,“Isomorfismo=False;];</code> <code>,{i,Length[F1]}];</code> | Comprueba que además es un isomorfismo de grafos. |
| , <code>Isomorfismo=False;</code>]; | Si no es biyectiva, no puede ser isomorfismo. |
| Isomorfismo | Salida de resultados. |

Programa 5.7. Isomorfismos de grafos no orientados.

Y para grafos dirigidos:

| PROGRAMA | COMENTARIOS |
|---|---|
| W1=CONJUNTO DE VERTICES DEL PRIMER GRAFO; F1=CONJUNTO DE FLECHAS DEL PRIMER GRAFO; W2=CONJUNTO DE VERTICES DEL SEGUNDO GRAFO; F2=CONJUNTO DE FLECHAS DEL SEGUNDO GRAFO; | Definición de los grafos. |
| DEFINICIÓN DE LA APLICACIÓN <code>funcion[]</code> ENTRE LOS CONJUNTOS DE VERTICES; | Definición de la aplicación. |
| Isomorfismo=True; | Suponemos por defecto que si lo es. |
| If[Length[W1]==Length[Table[funcion[W1[[i]],{i,Length[W1]}]]&& Length[W1]==Length[W2], | Comprobamos que “funcion” es biyectiva. |
| Do[<code>If[Length[Intersection[{funcion[F1[[i]][[1]]]>>funcion[F1[[i]][[2]]],funcion[F1[[i]][[2]]]>>funcion[F1[[i]][[1]]}],F2]==1</code> <code>,“Isomorfismo=False;];</code> <code>,{i,Length[F1]}];</code> | Comprueba que además es un isomorfismo de grafos. |
| , <code>Isomorfismo=False;</code>]; | Si no es biyectiva, no puede ser isomorfismo. |
| Isomorfismo | Salida de resultados. |

Programa 5.8. Isomorfismos de grafos dirigidos.

Ejemplo 5.7. Sean $G_1 = (W_1, F_1)$ y $G_2 = (W_2, F_2)$ dos grafos no orientados con $W_1 = W_2 = \{1, 2, 3, 4, 5\}$, $F_1 = \{1 \rightarrow 2, 1 \rightarrow 4, 1 \rightarrow 5, 2 \rightarrow 3, 2 \rightarrow 5, 3 \rightarrow 4, 3 \rightarrow 5, 4 \rightarrow 5\}$ y $F_2 = \{1 \rightarrow 2, 1 \rightarrow 3, 1 \rightarrow 4, 1 \rightarrow 5, 2 \rightarrow 3, 2 \rightarrow 5, 3 \rightarrow 4, 4 \rightarrow 5\}$. Comprobamos que la aplicación $f: W_1 \rightarrow W_2$ dada por, $f(1) = 4, f(2) = 5, f(3) = 2, f(4) = 3$ y $f(5) = 1$; define un isomorfismo de grafos.

Definimos en Mathematica los grafos y la aplicación:

```
In]:=          W1={1,2,3,4,5};
W2={1,2,3,4,5};
F1={1→2,1→4,1→5,2→3,2→5,3→4,3→5,4→5};
F2={1→2,1→3,1→4,1→5,2→3,2→5,3→4,4→5};
funcion[1]=4;
funcion[2]=5;
funcion[3]=2;
funcion[4]=3;
funcion[5]=1;
```

Usamos el programa 5.7.:

```
In]:=          Isomorfismo=True;
```

⋮ ⋮

```
Out]=          True
```

Supongamos ahora que G_1 y G_2 son grafos dirigidos, comprobemos usando el programa 5.8. si la aplicación “función” es un isomorfismo entre dígrafos:

```
In]:=          Isomorfismo=True;
```

⋮ ⋮

```
Out]=          False
```

□

Obsérvese que los dos grafos no orientados del ejemplo anterior son isomorfos y se diferencian en una permutación de sus vértices que conserva los lados (la aplicación biyectiva entre sus vértices que hemos definido: $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 5 & 2 & 3 & 1 \end{pmatrix}$).

Dados dos grafos $G_1 = (W_1, F_1)$ y $G_2 = (W_2, F_2)$ con el mismo número de vértices $|W_1| = |W_2| = n$ y de lados o flechas $|F_1| = |F_2|$, podemos determinar si son isomorfos comprobando si se diferencian en una permutación de los índices de sus vértices. Una buena forma de hacerlo sería comparar las matrices de adyacencia A_1 y A_2 , obviamente sólo tendremos que evidenciar que se diferencian en una permutación de los índices de los vértices, que se traducirá en la matriz de adyacencia en varias transformaciones elementales por filas (intercambiar las filas como se ha hecho con los índices) y las mismas por

columnas (intercambiar también las columnas igual que se ha hecho con los índices). Por tanto, obsérvese que en tal caso y si son isomorfos, existirá una matriz ortogonal y booleana P , tal que:

$$P \cdot A_1 \cdot P^t = A_2$$

La matriz P , será una permutación de las filas de la matriz identidad I_n que se corresponderá con la permutación de los índices de los vértices. (En particular las matrices de adyacencia de grafos isomorfos son congruentes). Véase [39] para más detalle.

Las distintas matrices P las calcularemos de la siguiente forma, para cada permutación $\sigma \in S_n$,

$$\sigma = \begin{pmatrix} 1 & 2 & \dots & n \\ \sigma(1) & \sigma(2) & \dots & \sigma(n) \end{pmatrix}$$

que podríamos entender que es la permutación de los índices de los vértices, la matriz correspondiente sería:

$$I_\sigma = \begin{pmatrix} 0 & \dots & 1 & \dots & 0 \\ 0 & \dots & 1 & \dots & 0 \\ \vdots & & \vdots & & \vdots \\ 0 & \dots & 1 & \dots & 0 \end{pmatrix}^{\sigma(1)-\text{ésima posición}} \\ \sigma(2)-\text{ésima posición} \\ \vdots \\ \sigma(n)-\text{ésima posición}$$

Con Mathematica utilizaremos

Permutations[IdentityMatrix[n]]

Nos dará como salida una lista de todas las posibles I_σ . Utilizando esto último podemos determinar el conjunto de todas las matrices de adyacencia asociadas a grafos isomorfos que llamaremos clase de isomorfía de matrices de adyacencia de grafos isomorfos.

| FUNCIÓN | COMENTARIOS |
|---|--|
| ISOMORFAS[A]:=Module[{j,matricesP}, isomorfas={}; | Se introduce la matriz de adyacencia. |
| matricesP= Permutations[IdentityMatrix[Dimensions[A][[1]]]]; | Calculamos todas las I_σ que nos permitirán determinar las matrices de adyacencia de la misma clase de isomorfía que “A”. |
| Do[isomorfas=Union[isomorfas, {matricesP[[j]].A.Transpose[matricesP[[j]]]}]; ,{j,Length[matricesP]}]; isomorfas]; | Calculamos $I_\sigma A(I_\sigma)^t$ para todas las I_σ . |
| | Se muestran los resultados. |

Funció 5.9. Grafos isomorfos a uno dado.

Usando estas matrices podemos generar un test que compruebe si dos grafos son isomorfos como sigue:

| FUNCIÓN | COMENTARIOS |
|---|---|
| <pre>ISOMORFOS[A1_,A2_]:=Module[{matricesP,i,j}, Isomorfos=False; If[Dimensions[A1]==Dimensions[A2], matricesP=Permutations[IdentityMatrix[Dimensions[A1][[1]]]]; Do[If[matricesP[[i]].A1.Transpose[matricesP[[i]]] ==A2 , Isomorfos=True; matrizP=matricesP[[i]]; permutacion=i; Break[]]; ,{i,Length[matricesP]}];]; If[Isomorfos, Print["Son isomorfos, la matriz P es:", MatrixForm[matrizP], ", una permutación es:", MatrixForm[{Table[j,{j,Dimensions[A1][[1]]}], Permutations[Table[j ,{j,Dimensions[A1][[1]]}][[permutacion]]}]], , Print["No son isomorfos"]];];];</pre> | <p>La función tendrá dos entradas, las matrices de adyacencia de los grafos que vamos a comprobar si son isomorfos.</p> <p>Calculamos todas las posibles I_σ.</p> <p>Comprobamos si existe alguna permutación de los índices que haga a los grafos isomorfos.</p> <p>Se muestran los resultados</p> |

Función 5.10. Test de isomorfía.

Ejemplo 5.8. Calculamos todas las matrices de adyacencia de todos los grafos isomorfos a G , cuya matriz de adyacencia es:

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

Usaremos la función 5.9.:

In[]:= **ISOMORFAS[A_]:=Module[{j,matricesP},**

⋮ ⋮

In[]:= **ISOMORFAS[{{0,1,0,0},{1,0,0,1},{0,0,0,1},{0,1,1,0}}]**

Out[]=

$$\left\{ \begin{array}{l} \{\{0,0,0,1\},\{0,0,1,0\},\{0,1,0,1\},\{1,0,1,0\}\}, \\ \{\{0,0,0,1\},\{0,0,1,1\},\{0,1,0,0\},\{1,1,0,0\}\}, \\ \{\{0,0,1,0\},\{0,0,0,1\},\{1,0,0,1\},\{0,1,1,0\}\}, \\ \{\{0,0,1,0\},\{0,0,1,1\},\{1,1,0,0\},\{0,1,0,0\}\}, \\ \{\{0,0,1,1\},\{0,0,0,1\},\{1,0,0,0\},\{1,1,0,0\}\}, \\ \{\{0,0,1,1\},\{0,0,1,0\},\{1,1,0,0\},\{1,0,0,0\}\}, \\ \{\{0,1,0,0\},\{1,0,0,1\},\{0,0,0,1\},\{0,1,1,0\}\}, \\ \{\{0,1,0,0\},\{1,0,1,0\},\{0,1,0,1\},\{0,0,1,0\}\}, \\ \{\{0,1,0,1\},\{1,0,0,0\},\{0,0,0,1\},\{1,0,1,0\}\}, \\ \{\{0,1,0,1\},\{1,0,1,0\},\{0,1,0,0\},\{1,0,0,0\}\}, \\ \{\{0,1,1,0\},\{1,0,0,0\},\{1,0,0,1\},\{0,0,1,0\}\}, \\ \{\{0,1,1,0\},\{1,0,0,1\},\{1,0,0,0\},\{0,1,0,0\}\} \end{array} \right\}$$

Utilizamos MatrixForm[] para mostrar las matrices apropiadamente:

In[]:= **Table[MatrixForm[isomorfas[[i]]],{i,Length[isomorfas]}]**

Out[]=

$$\left\{ \begin{array}{l} \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}, \\ \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}, \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \end{array} \right\}$$

Obsérvese que han resultado 12 matrices en la clase de isomorfía, y que en realidad tenemos 24 permutaciones distintas de los cuatro vértices, sin embargo, algunas permutaciones dan lugar a las mismas matrices de adyacencia, en consecuencia el número de matrices de adyacencia es menor que el de permutaciones.

□

En el paquete <<Combinatorica` de la versión 6 de Mathematica encontramos dos funciones para estudiar si dos objetos de tipo grafo son isomorfos, por un lado tenemos:

IsomorphicQ[Grafo1,Grafo2]

que devolverá “True” si los grafos que representan los objetos “Grafo1” y “Grafo2” son isomorfos y “False” en caso contrario; y por otro lado encontramos la función:

Isomorphism[Grafo1,Grafo2]

que nos devolverá un isomorfismo entre “Grafo1” y “Grafo2” si estos son isomorfos, si además incluimos la opción “All”, entonces devolverá todos los isomorfismos que existan entre ambos grafos. Si modificamos la función 5.10. también podemos mostrar todas las permutaciones de los índices de los vértices que definen los isomorfismos, si existen, entre dos grafos (ejercicio 5.32.).

Ejemplo 5.9. Comprobamos cuáles de los siguientes grafos no dirigidos son isomorfos:

a)

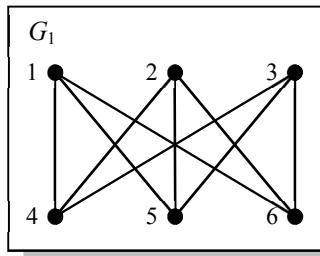


Ilustración 5.6.

b)

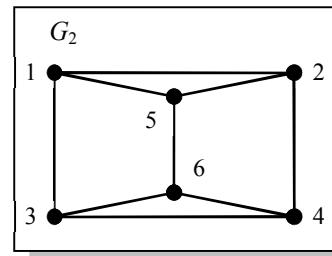


Ilustración 5.7.

c)

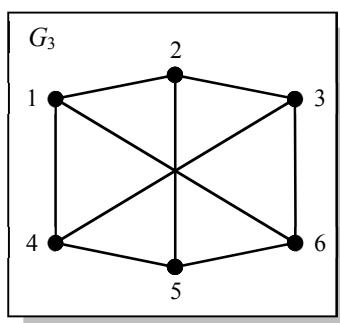


Ilustración 5.8.

d) G_4 , es el grafo con matriz de adyacencia:

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

e) G_5 , es el grafo con matriz de incidencia:

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

f) $G_6 = (W, F)$ con $W = \{1, 2, 3, 4, 5, 6\}$ y $F = \{1 \rightarrow 2, 1 \rightarrow 3, 1 \rightarrow 4, 2 \rightarrow 3, 2 \rightarrow 6, 3 \rightarrow 5, 4 \rightarrow 5, 4 \rightarrow 6, 5 \rightarrow 6\}$

Observemos que los grafos G_1 y G_3 parecen ser los mismos de la ilustración 5.3, sin embargo, a diferencia de aquellos si calculamos sus matrices de adyacencia observaremos que ahora son distintas porque aunque los dibujos sean los mismos, los vértices han cambiado de nombre.

Para comprobar cuáles de ellos son isomorfos determinamos sus matrices de adyacencia y buscamos las que pertenecen a la misma clase de isomorfía con 5.9. y en consecuencia representan a grafos isomorfos.

$$A_1 = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}, A_2 = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}, A_3 = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix},$$

$$A_4 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}, A_5 = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}, A_6 = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

Las introducimos en Mathematica:

```
In]:= A1={{{0,0,0,1,1,1},{0,0,0,1,1,1},{0,0,0,1,1,1},{1,1,1,0,0,0},
{1,1,1,0,0,0},{1,1,1,0,0,0}};
A2={{0,1,1,0,1,0},{1,0,0,1,1,0},{1,0,0,1,0,1},{0,1,1,0,0,1},
{1,1,0,0,1},{0,0,1,1,1,0}};
A3={{0,1,0,1,0,1},{1,0,1,0,1,0},{0,1,0,1,0,1},{1,0,1,0,1,0},
{0,1,0,1,0,1},{1,0,1,0,1,0}};
A4={{0,1,1,0,0,1},{1,0,1,0,1,0},{1,1,0,1,0,0},{0,0,1,0,1,1},
{0,1,0,1,0,1},{1,0,0,1,1,0}};
A5={{0,0,0,1,1,1},{0,0,0,1,1,1},{0,0,0,1,1,1},{1,1,1,0,0,0},
{1,1,1,0,0,0},{1,1,1,0,0,0}};
A6={{0,1,1,1,0,0},{1,0,1,0,0,1},{1,1,0,0,1,0},{1,0,0,0,1,1},
{0,0,1,1,0,1},{0,1,0,1,1,0}};
```

Definimos la función 5.10. y la usamos para comparar las matrices de adyacencia y comprobar si pertenecen a grafos isomorfos:

```
In]:= ISOMORFOS[A1_,A2_]:=Module[{matricesP,i,j},
```

⋮ ⋮

In[]:= **ISOMORFOS[A1,A2]**

Out[] = No son isomorfos

In[]:= **ISOMORFOS[A1,A3]**

Out[] = Son isomorfos, una matriz P es:

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

una permutación es: $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 4 & 2 & 5 & 3 & 6 \end{pmatrix}$

In[]:= **ISOMORFOS[A1,A4]**

Out[] = No son isomorfos

In[]:= **ISOMORFOS[A1,A5]**

Out[] = Son isomorfos, una matriz P es:

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

una permutación es: $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix}$

In[]:= **ISOMORFOS[A1,A6]**

Out[] = No son isomorfos

In[]:= **ISOMORFOS[A2,A4]**

Out[] = Son isomorfos, una matriz P es:

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

una permutación es: $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 5 & 6 & 4 & 3 \end{pmatrix}$

In[] := **ISOMORFOS[A2,A6]**

Out[] = Son isomorfos, una matriz P es:

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

una permutación es: $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 5 & 3 & 6 & 4 \end{pmatrix}$

Por tanto G_1 , G_3 y G_4 son grafos isomorfos y G_2 , G_4 y G_6 también son isomorfos.

Análogamente si definimos los objetos de tipo grafo asociados a los seis grafos:

In[] := **<<Combinatorica`**

In[] := **G1=FromAdjacencyMatrix[A1];**
G2=FromAdjacencyMatrix[A2];
G3=FromAdjacencyMatrix[A3];
G4=FromAdjacencyMatrix[A4];
G5=FromAdjacencyMatrix[A5];
G6=FromAdjacencyMatrix[A6];

comprobamos cuáles de ellos son isomorfos y calculamos un isomorfismo:

In[] := **IsomorphicQ[G1,G2]**

Out[] = **False**

In[] := **IsomorphicQ[G1,G3]**

Out[] = **True**

In[] := **Isomorphism[G1,G3]**

```

Out[] = {1, 3, 5, 2, 4, 6}
In[] := IsomorphicQ[G1,G5]
Out[] = True
In[] := Isomorphism[G1,G5]
Out[] = {1, 2, 3, 4, 5, 6}
In[] := IsomorphicQ[G2,G4]
Out[] = True
In[] := Isomorphism[G2,G4]
Out[] = {1, 2, 6, 5, 3, 4}
In[] := IsomorphicQ[G2,G6]
Out[] = True
In[] := Isomorphism[G2,G6]
Out[] = {1, 2, 4, 6, 3, 5}

```

Y llegamos a la misma conclusión que antes. Si quisieramos mostrar todas las permutaciones para dos grafos isomorfos escribiríamos:

```

In[] := Isomorphism[G2,G6,All]
Out[] = {{1, 2, 4, 6, 3, 5}, {1, 3, 4, 5, 2, 6}, {2, 1, 6, 4, 3, 5},
          {2, 3, 6, 5, 1, 4}, {3, 1, 5, 4, 2, 6}, {3, 2, 5, 6, 1, 4},
          {4, 5, 1, 3, 6, 2}, {4, 6, 1, 2, 5, 3}, {5, 4, 3, 1, 6, 2},
          {5, 6, 3, 2, 4, 1}, {6, 4, 2, 1, 5, 3}, {6, 5, 2, 3, 4, 1}}

```

También hubiésemos podido hacer el ejercicio usando la función 5.9., con ella calcularíamos todas las matrices de adyacencia pertenecientes a la clase de isomorfía de A_1 , y así podríamos comprobar cuáles de las restantes matrices están en ella y en consecuencia son matrices de adyacencia de grafos isomorfos, lo mismo para A_2 y así concluiríamos el ejercicio. □

Ejemplo 5.10. Comprobar cuáles de los siguientes grafos dirigidos son isomorfos:

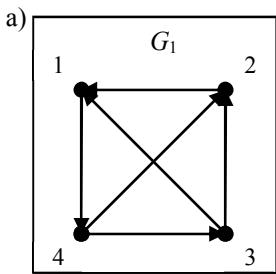


Ilustración 5.9.

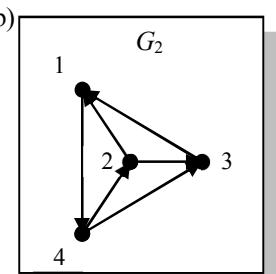


Ilustración 5.10.

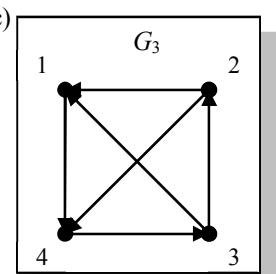


Ilustración 5.11.

Igual que el ejemplo anterior, calculamos sus matrices de adyacencia y aplicamos la función 5.10.

$$A_1 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}, A_2 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} \text{ y } A_3 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

Las introducimos en Mathematica:

```
In]:= A1={{0,0,0,1},{1,0,0,0},{1,1,0,0},{0,1,1,0}};
A2={{0,0,0,1},{1,0,1,0},{1,0,0,0},{0,1,1,0}};
A3={{0,0,0,1},{1,0,0,1},{1,1,0,0},{0,0,0,1}};
```

Definimos la función 5.10. y la utilizamos:

```
In]:= ISOMORFOS[A1_,A2_]:=Module[{matricesP,i,j},
```

```
⋮ ⋮
```

```
In]:= ISOMORFOS[A1,A2]
```

Out[]= Son isomorfos, una matriz P es: $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$,
la permutación es: $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 2 & 4 \end{pmatrix}$

```
In]:= ISOMORFOS[A1,A3]
```

Out[]= No son isomorfos

Luego sólo son isomorfos G_1 y G_3 . Y usando IsomorphicQ[]:

```
In]:= <<Combinatorica`  
In]:= G1=FromAdjacencyMatrix[A1,Type->Directed];  
G2=FromAdjacencyMatrix[A2,Type->Directed];  
G3=FromAdjacencyMatrix[A3,Type->Directed];  
In]:= IsomorphicQ[G1,G3]  
Out]= False  
In]:= IsomorphicQ[G1,G2]  
Out]= True  
In]:= Isomorphism[G1,G2]  
Out]= {1,3,2,4}
```

□

4.1. EFICACIA Y OPTIMIZACIÓN

Nos limitaremos a grafos no orientados, para dígrafos sería similar. El método para determinar si dos matrices se corresponden con grafos isomorfos resulta bastante ineficaz, tenemos que hacer multitud de cálculos y comprobaciones, además en ocasiones no es posible utilizarla como queda patente en algunos ejemplos de este epígrafe.

Podemos encontrar algunas sencillas condiciones necesarias, que en muchos casos nos eviten la comprobación con la función ISOMORFOS[] (5.9.), sirvan de ejemplo las siguientes²⁵:

- Puesto que dos matrices A_1 y A_2 de adyacencia asociadas a grafos no orientados son simétricas, entonces son diagonalizables, y además si pertenecen a grafos isomorfos, en particular también son semejantes, por tanto tendrán los mismos valores propios, de hecho, sabemos aun más, dichas matrices serán ortogonalmente semejantes. En definitiva, si dos grafos son isomorfos entonces sus matrices de adyacencia tendrán los mismos valores propios, pero desafortunadamente, al revés no es cierto, como podemos comprobar en el ejemplo 5.12. Podemos plasmar esta condición necesaria fácilmente, bastaría con escribir en Mathematica²⁶:

²⁵ Según el problema particular al que nos enfrentemos, necesitaremos propiedades o invariantes distintos que nos resulten favorables para su resolución. Aquí mostramos sólo un par de ejemplos factibles desde punto de vista teórico y computacional.

²⁶ En Mathematica disponemos de la función Eigenvalues[] para determinar los valores propios de una matriz directamente, ésta no es habitual en otros lenguajes, recordemos que los valores propios se corresponden con las raíces del polinomio característico: $p(\lambda) = \det(A - \lambda I)$.

Eigenvalues[A1]==Eigenvalues[A2]

- Otros propiedades²⁷ asociadas a cada vértice del grafo, que determinamos desde a las matrices de adyacencia y que permiten decidir cuando dos grafos no son isomorfos, son las diagonales principales de las potencias de las matrices de adyacencia. Si A_1 y A_2 son dos matrices de adyacencia de dos grafos isomorfos, las diagonales principales de $(A_1)^n$ y $(A_2)^n$ deberán iguales salvo el orden (permutación de la diagonal) para cada $n \geq 2$. Podemos comprobarlo con Mathematica:

```
n=EXPONENTE;
Sort[Table[MatrixPower[A1,n][[i,i]],{i,Dimensions[A1][[1]]}]]==
Sort[Table[MatrixPower[A2,n][[i,i]],{i,Dimensions[A2][[1]]}]]
```

o bien, desde el cuadrado hasta la potencia $(n + 1)$ -ésima de la matriz de adyacencia A , las dos listas de vectores de n coordenadas, donde el vector i -ésimo es el formado por las (i, i) -ésimas coordenadas de A^l para cada $l = 2, 3, \dots, (n + 1)$, coincidirán salvo el orden:

```
n=POTENCIA MÁXIMA QUE SE COMPROBARÁ MENOS UNO;
Sort[Table[
  Table[MatrixPower[A1,i+1][[j,j]],[i,n]},{j,Dimensions[A1][[1]]}]]==
Sort[Table[
  Table[MatrixPower[A2,i+1][[j,j]],[i,n]},{j,Dimensions[A2][[1]]}]]
```

Observemos que las propiedades anteriores son condiciones necesarias para que dos matrices de adyacencia pertenezcan a la misma clase de isomorfía, hablaremos por tanto de invariantes asociados a las matrices de adyacencia de una misma clase de isomorfía. En los ejemplos siguientes comprobaremos como los invariantes anteriores, en general, no constituyen una condición suficiente para determinar si dos grafos son isomorfos, siempre podremos encontrar grafos no isomorfos cuyas matrices de adyacencia tengan iguales valores propios (ejemplo 5.12.) o las diagonales de sus potencias n -ésimas coincidan salvo el orden para cualquier n fijo que consideremos (ejemplo 5.13.). Sin embargo, también comprobaremos que estas propiedades serán de gran relevancia desde el punto de vista de la optimización (ejemplos 5.13. y 5.15.).

Ejemplo 5.11. Dados los siguientes grafos:

²⁷ En el capítulo 7 se estudia el teorema del número de caminos, con él entenderemos el significado de las diagonales de las potencias de las matrices de adyacencia y podremos interpretar esta propiedad.

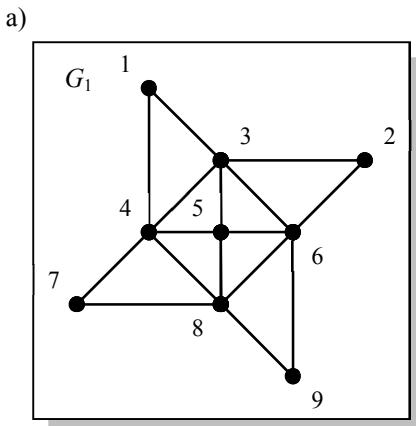


Ilustración 5.12.

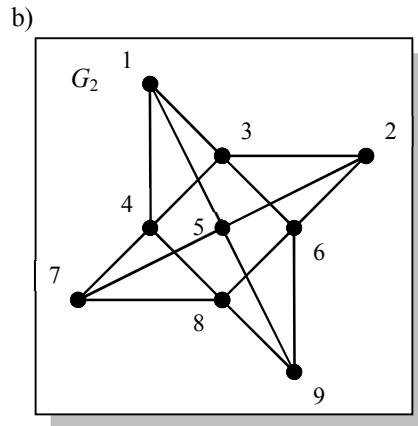


Ilustración 5.13.

Vamos a comprobar si son isomorfos.

Primero usamos directamente la función 5.10. y medimos el tiempo empleado:

In[]:=

ISOMORFOS[A1_,A2_]:=Module[{matricesP,i,j},

\vdots

\vdots

In[]:=

A1={{0,0,1,1,0,0,0,0,0},{0,0,1,0,0,1,0,0,0},{1,1,0,1,1,1,0,0,0},
{1,0,1,0,1,0,1,1,0},{0,0,1,1,0,1,1,0,0},{0,1,1,0,1,0,1,0,1},
{0,0,0,1,1,1,0,1,1},{0,0,0,1,0,0,1,0,0},{0,0,0,0,0,1,1,0,0};
A2={{0,0,1,1,1,0,0,0,0},{0,0,1,0,1,1,0,0,0},{1,1,0,1,0,1,0,0,0},
{1,0,1,0,0,0,1,1,0},{1,1,0,0,0,0,1,0,1},{0,1,1,0,0,0,0,1,1},
{0,0,0,1,1,0,0,1,0},{0,0,0,1,0,1,1,0,1},{0,0,0,0,1,1,0,1,0};
Timing[ISOMORFOS[A1,A2]]][[1]]

Out[]=

No son isomorfos

17.297 Second

Aunque es un grafo de tan sólo 9 vértices el tiempo empleado en el cálculo es bastante elevado. Además téngase en cuenta que se han calculado todas las permutaciones de las nueve filas que componen la matriz identidad 9×9 , esto es, un total de 362880 matrices, lo que también supone un uso muy elevado de memoria. Obsérvese como cualquiera de las condiciones necesarias sobre las matrices de adyacencia expuestas anteriormente es más rápida y no necesita tal volumen de memoria:

In[]:=

Timing[Eigenvalues[A1]==Eigenvalues[A2]]

Out[]=

{0.015 Second, False}

*In[]:=***Timing[****n=2;****B1=MatrixPower[A1,n];B2=MatrixPower[A2,n];****Sort[Table[B1[[i,i]],{i,Dimensions[B1][[1]]}]]==****Sort[Table[B2[[i,i]],{i,Dimensions[B2][[1]]}]]]****]***Out[] =*

{0. Second, False}

□

Ejemplo 5.12. Calculamos los valores propios de las matrices de adyacencia de los siguientes grafos:

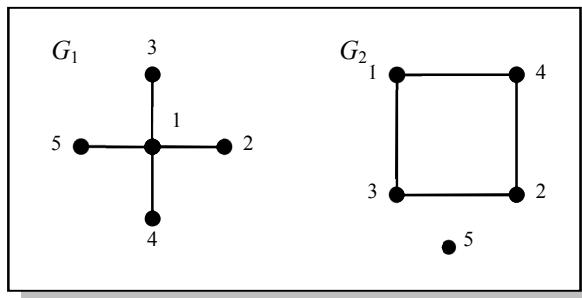


Ilustración 5.14.

Introducimos las matrices de adyacencia y calculamos sus valores propios:

In[]:=

$$\text{Eigenvalues}\left[\{\{0,1,1,0,0\}, \{1,0,0,1,0\}, \{1,0,0,1,0\}, \{0,1,1,0,0\}, \{0,0,0,0,0\}\}\right]$$
Out[] =

{-2, 2, 0, 0, 0}

In[]:=

$$\text{Eigenvalues}\left[\{\{0,1,1,1,1\}, \{1,0,0,0,0\}, \{1,0,0,0,0\}, \{1,0,0,0,0\}, \{1,0,0,0,0\}\}\right]$$
Out[] =

{-2, 2, 0, 0, 0}

y es obvio que no son isomorfos. □

Ejemplo 5.13. Dadas las siguientes matrices de adyacencia:

Veamos si pertenecen a la misma clase de isomorfía, esto es, representan grafos isomorfos.

```

In[]:= A1={\{0,0,0,0,0,1,1,0,0,0,0,0,1,0,0,0\},\{0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0\},
\{0,1,0,1,0,0,0,0,1,0,0,0,0,0,0,0\},\{0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0\},
\{0,0,0,1,0,0,0,0,0,0,0,0,0,1,0\},\{1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0\},
\{1,0,0,0,0,0,0,0,0,0,1,0,0,0\},\{0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0\},
\{0,0,1,0,0,0,1,0,1,0,0,0,0,0,0,0\},\{0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0\},
\{0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1\},\{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1\},
\{1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0\},\{0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0\},
\{0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0\},\{0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0\}};
A2={\{0,0,0,0,0,1,1,0,0,0,0,0,1,0,0,0\},\{0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0\},
\{0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0\},\{0,0,1,0,1,0,0,0,0,1,0,0,0,0,0,0\},
\{0,0,0,1,0,0,0,0,0,0,0,0,0,1,0\},\{1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0\},
\{1,0,0,0,0,0,0,0,0,0,1,0,1,0,0\},\{0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0\},
\{0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0\},\{0,0,0,1,0,0,0,0,1,0,1,0,0,0,0,0\},
\{0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1\},\{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1\},
\{1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0\},\{0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0\},
\{0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0\},\{0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0\}};

```

No podemos usar ISOMORFOS[] (5.10.) porque implica el uso de 2092278988000 posibles permutaciones, por lo que utilizaremos las propiedades. Podemos comprobar que:

```
In[7]:= Sort[Table[MatrixPower[A1,2][[i1,i1]],{i1,16}]]==
          Sort[Table[MatrixPower[A2,2][[i1,i1]],{i1,16}]];
Sort[Table[MatrixPower[A1,3][[i1,i1]],{i1,16}]]==
          Sort[Table[MatrixPower[A2,3][[i1,i1]],{i1,16}]];
Sort[Table[MatrixPower[A1,4][[i1,i1]],{i1,16}]]==
          Sort[Table[MatrixPower[A2,4][[i1,i1]],{i1,16}]];
Sort[Table[MatrixPower[A1,5][[i1,i1]],{i1,16}]]==
          Sort[Table[MatrixPower[A2,5][[i1,i1]],{i1,16}]]
```

```
Out]:=      True
           True
           True
           True
```

Observemos como las diagonales de las potencias 2^a , 3^a , 4^a y 5^a coinciden, pero no por ello podemos asegurar que los grafos sean isomorfos, en efecto:

```
In]:=      Sort[Table[MatrixPower[A1,6][[i1,i1]],[i1,16]]]==
           Sort[Table[MatrixPower[A2,6][[i1,i1]],[i1,16]]]
```

```
Out]=      False
```

y

```
In]:=      Eigenvalues[A1]==Eigenvalues[A2]
```

```
Out]=      False
```

Por tanto no son isomorfos. Analicemos, siempre a modo de ejemplo y sin pretender ser exhaustivos, la situación que se ha presentado en el ejemplo, obsérvese la representación gráfica de los grafos:

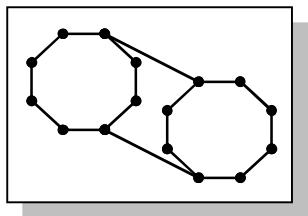


Ilustración 5.15.

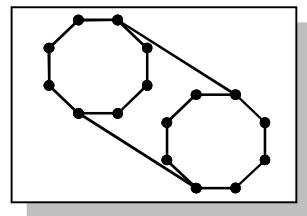


Ilustración 5.16.

Los grafos están compuestos por tres ciclos (véase el capítulo 7, donde se estudian con detalle los ciclos) de cierta longitud, dos en los extremos y otro que surge al conectarlos, se ha elegido esta configuración para que las primeras potencias de las matrices de adyacencia tengan iguales diagonales (salvo el orden) y nos veamos obligados a llegar a la potencia sexta para diferenciarlos. Configuraciones similares pueden obtenerse siempre que lo deseemos y de forma que coincidan las diagonales de todas las potencias A^i con $2 \leq i \leq n$ para un n cualquiera.

□

El ejemplo 5.13. revela el grave problema de eficacia que presenta la función ISOMORFOS[] (5.10.) y la necesidad de buscar una alternativa, utilizando los invariantes analizados anteriormente, vamos a construir un nuevo algoritmo que compruebe si dos grafos son isomorfos de forma más eficaz. Para ello en primer lugar comprobaremos si ambas matrices de adyacencia tienen los mismos invariantes; en caso afirmativo, buscaremos todas las posibles permutaciones que pasen de una matriz a otra, para ello construiremos las permutaciones (isomorfismos) de forma que cada vértice con sus invariantes (previamente calculados), sólo pueda ir a otro vértice con los mismos

invariantes y a cada paso que demos en la correspondencia de vértices que vamos construyendo, además comprobaremos que se vaya conservando la adyacencia entre los mismos para que sea isomorfismo:

| FUNCIÓN | COMENTARIOS |
|---|--|
| ISOMORFOS2[A1_,A2_]:=Module[{potencias,n,permutaciones2,adyacentes1,tabla1, tabla2,condicion,imagenes,i,j,k,h,kk}, | La función tendrá como argumentos las matrices de adyacencia de los grafos que queremos comprobar si son isomorfos. |
| potencias=3; | Indicaremos cuantas potencias de matrices empezando en la segunda queremos utilizar en la comprobación. |
| n=Dimensions[A1][[1]]; tabla1=Table[Table[MatrixPower[A1,i+1][[j,j]], ,{i,potencias}],{j,Dimensions[A1][[1]]}]; tabla2=Table[Table[MatrixPower[A2,i+1][[j,j]], ,{i,potencias}],{j,Dimensions[A2][[1]]}]; | Calculamos las tablas con los listados de invariantes asociados a cada vértice. |
| If[Sort[tabla1]==Sort[tabla2] && Eigenvalues[A1]==Eigenvalues[A2], | Comprobamos que tengan iguales invariantes, en caso contrario ya podremos asegurar que no son isomorfos. |
| Permutaciones=Position[tabla2,tabla1[[1]]]; | Al primer vértice podremos asignarle otro cualquiera del segundo grafo que tenga iguales invariantes asociados. |
| Do[| Bucle que recorre los $n - 1$ vértices que quedan por asignarle su imagen. |
| permutaciones2={}; adyacentes1=Position[A1[[i]],1]; | En “adyacentes1” almacenamos la información sobre los vértices adyacentes al vértice i -ésimo. |
| Do[| Bucle que recorre las posibles permutaciones que se están construyendo. |
| imagenes=Complement[Position[tabla2,tabla1[[i]]], Table[{permutaciones[[i]][[kk]]} ,{kk,Length[permutaciones[[i]]]}]]; | En “imagenes” almacenamos los vértices que pueden ser imagen por los invariantes y porque previamente no ha sido asignado. |

| | |
|---|---|
| <pre> Do[condicion=True; Do[If[adyacentes1[[k]][[1]]<i, If[A2[[permutaciones[[j]]][[adyacentes1[[k]][[1]]]]] , imagenes[[h]][[1]]]]!=1 , condicion=False;Break[]];];]; ,{k,Length[adyacentes1]};] If[condicion , permutaciones2=Join[permutaciones2 ,{Join[permutaciones[[j]] ,{imagenes[[h]][[1]]}]}];]; ,{h,Length[imagenes]}]; </pre> | <p>Recorremos las posibles imágenes y determinamos cuales de ellas conservan la adyacencia.</p> |
| <pre> If[Mod[j,1000]==0, Print["Van ",j," ", Length[permutaciones2]]; ,{j,Length[permutaciones]}]; </pre> | <p>Mostramos información parcial sobre las posibles permutaciones.</p> |
| <pre> permutaciones=permutaciones2; </pre> | <p>Actualizamos las posibles permutaciones después de esta pasada.</p> |
| <pre> ,{i,2,n}]; Print["PERMUTACIONES:"]; Print[Table[MatrixForm[{Table[j,{j,n}], permutaciones[[i]]}],{i,Length[permutaciones]}]]; , permutaciones={};]; </pre> | <p>Mostramos todas las permutaciones calculadas.</p> |
| <pre> If[permutaciones=={},False,True]]; </pre> | <p>Si no se han encontrado permutaciones entenderemos que no son isomorfos.</p> |

Función 5.11. Test de isomorfía II.

Podríamos añadir más velocidad si paramos el algoritmo una vez determinada una permutación, pues en tal caso ya podríamos asegurar que se trata de grafos isomorfos, por el momento la usaremos así y mostraremos ejemplos donde encontraremos muchas permutaciones entre grafos isomorfos.

Ejemplo 5.14. Comprobamos si los siguientes grafos son isomorfos y en caso afirmativo, determinamos los isomorfismos de grafos que existen entre ambos.

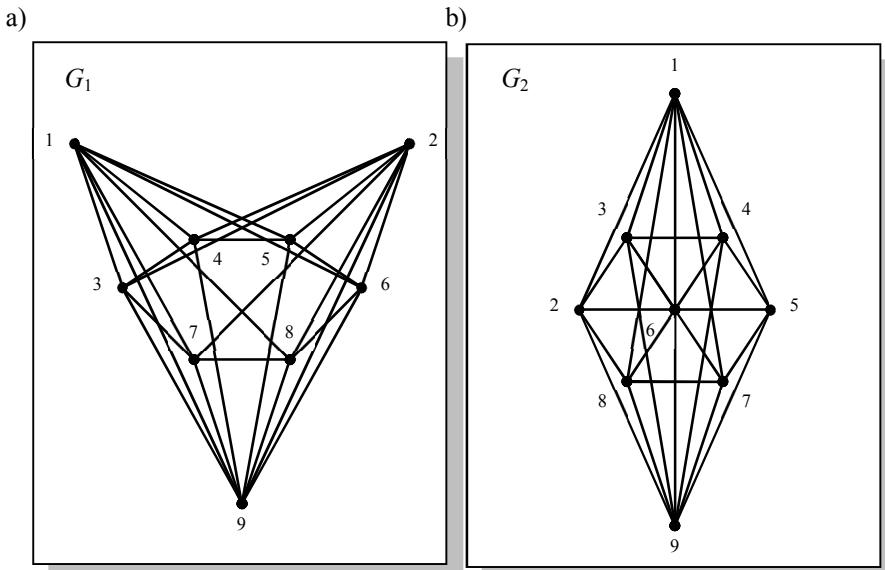


Ilustración 5.17.

Ilustración 5.18.

Calculamos las matrices de adyacencia de ambos y utilizamos la función 5.11., mediremos también su eficacia.

```
In]:= A1={{0,0,1,1,1,1,1,1,1},{0,0,1,1,1,1,1,1,1},{1,1,0,1,0,0,1,0,1},
{1,1,1,0,1,0,0,0,1},{1,1,0,1,0,1,0,0,1},{1,1,0,0,1,0,0,1,1},
{1,1,1,0,0,0,0,1,1},{1,1,0,0,0,1,1,0,1},{1,1,1,1,1,1,1,1,0}};
A2={{0,1,1,1,1,1,1,1,0},{1,0,1,0,0,1,0,1,1},{1,1,0,1,0,1,0,0,1},
{1,0,1,0,1,1,0,0,1},{1,0,0,1,0,1,1,0,1},{1,1,1,1,0,1,1,1,1},
{1,0,0,0,1,1,0,1,1},{1,1,0,0,0,1,1,0,1},{0,1,1,1,1,1,1,1,0}};

In]:= ISOMORFOS2[A1_,A2_]:=Module[
];
;
;

In]:= Timing[ISOMORFOS2[A1,A2]]

Out]:= PERMUTACIONES:


$$\left\{ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 9 & 2 & 3 & 4 & 5 & 8 & 7 & 6 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 9 & 2 & 8 & 7 & 5 & 3 & 4 & 6 \end{pmatrix}, \right.$$


$$\left. \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 9 & 3 & 2 & 8 & 7 & 4 & 5 & 6 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 9 & 3 & 4 & 5 & 7 & 2 & 8 & 6 \end{pmatrix}, \right.$$


$$\left. \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 9 & 4 & 3 & 2 & 8 & 5 & 7 & 6 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 9 & 4 & 3 & 7 & 8 & 3 & 2 & 6 \end{pmatrix} \right\}$$

```

$$\left(\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 9 & 5 & 4 & 3 & 2 & 7 & 8 & 6 \end{array} \right), \quad \left(\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 9 & 5 & 7 & 8 & 2 & 4 & 3 & 6 \end{array} \right),$$

$$\left(\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 9 & 7 & 5 & 4 & 3 & 8 & 2 & 6 \end{array} \right), \quad \left(\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 9 & 7 & 8 & 2 & 3 & 5 & 4 & 6 \end{array} \right),$$

$$\left(\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 9 & 8 & 2 & 3 & 4 & 7 & 5 & 6 \end{array} \right), \quad \left(\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 9 & 8 & 7 & 5 & 4 & 2 & 3 & 6 \end{array} \right),$$

$$\left(\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 9 & 1 & 2 & 3 & 4 & 5 & 8 & 7 & 6 \end{array} \right), \quad \left(\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 9 & 1 & 2 & 8 & 7 & 5 & 3 & 4 & 6 \end{array} \right),$$

$$\left(\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 9 & 1 & 3 & 2 & 8 & 7 & 4 & 5 & 6 \end{array} \right), \quad \left(\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 9 & 1 & 3 & 4 & 5 & 7 & 2 & 8 & 6 \end{array} \right),$$

$$\left(\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 9 & 1 & 4 & 3 & 2 & 8 & 5 & 7 & 6 \end{array} \right), \quad \left(\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 9 & 1 & 4 & 3 & 7 & 8 & 3 & 2 & 6 \end{array} \right),$$

$$\left(\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 9 & 1 & 5 & 4 & 3 & 2 & 7 & 8 & 6 \end{array} \right), \quad \left(\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 9 & 1 & 5 & 7 & 8 & 2 & 4 & 3 & 6 \end{array} \right),$$

$$\left(\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 9 & 1 & 7 & 5 & 4 & 3 & 8 & 2 & 6 \end{array} \right), \quad \left(\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 9 & 1 & 7 & 8 & 2 & 3 & 5 & 4 & 6 \end{array} \right),$$

$$\left(\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 9 & 1 & 8 & 2 & 3 & 4 & 7 & 5 & 6 \end{array} \right), \quad \left(\begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 9 & 1 & 8 & 7 & 5 & 4 & 2 & 3 & 6 \end{array} \right) \}$$

{0.047 Second, True}

In]:= **MemoryInUse[]**

Out]= 2363624

Para comparar la función 5.11. con la 5.10., descargamos previamente cualquier información de la memoria para no falsear el resultado, por ejemplo quitamos el núcleo, y comprobemos su eficacia comparándola con ISOMORFOS[] (5.10.).

In]:= **ISOMORFOS[A1_,A2_]:=Module[{matricesP,i,j},**

⋮ ⋮ ⋮

In]:= **Timing[ISOMORFOS[A1,A2]]**

Out]= {0.816 Second, True}

Para comprobar cuánta memoria ha usado aproximadamente la función 5.10. definimos nuevamente la variable local “matricesP”:

In]:= **matricesP=Permutations[**

IdentityMatrix[Dimensions[A1][[1]]]
];

In[]:= **MemoryInUse[]**

Out[] = 23924720

□

Ejemplo 5.15. Comparamos el funcionamiento con las matrices de adyacencia del ejemplo 5.13.:

In[]:= **A1={**{0,0,0,0,0,1,1,0,0,0,0,1,0,0,0},
 {0,0,1,0,0,0,0,0,0,0,0,1,0,0,0},

⋮ ⋮

A2={{0,0,0,0,0,1,1,0,0,0,0,1,0,0,0},
 {0,0,1,0,0,0,0,0,0,0,0,1,0,0,0},

⋮ ⋮

In[]:= **ISOMORFOS2[A1_,A2_]:=Module[**

⋮ ⋮

In[]:= **Timing[ISOMORFOS2[A1,A2]]**

Out[] = {0.063 Second, False}

In[]:= **MemoryInUse[]**

Out[] = 2294752

In[]:= **ISOMORFOS[A1_,A2_]:=Module[{matricesP,i,j},**

⋮ ⋮

In[]:= **Timing[ISOMORFOS[A1,A2]]**

Out[] = No more memory available.
Mathematica kernel has shut down.
Try quitting other applications and then retry.

Al usar la función 5.10. con estos grafos, los recursos de memoria que se precisan son demasiados y Mathematica nos devuelve un error.

□

Con este ejemplo, queda patente la obligatoriedad de considerar propiedades como las analizadas para el estudio del problema del isomorfismo entre grafos.

Por último, podemos comparar el rendimiento de ISOMORFOS2[] e Isomorphism[], de ésta podemos encontrar una descripción en la sección 8.6 de [42], en realidad ambas parecen funcionar de manera similar, aunque considerando otros invariantes distintos en Isomorphism[], a los que hemos considerados para construir ISOMORFOS2[]. Antes de evaluar los tiempos obtenidos en cada caso realizando los mismos cálculos, recordemos lo visto en la sección 1.5. del capítulo 4 de este libro, y aun así, comprobaremos que los tiempos obtenidos con ambas funciones son similares. Podemos generar grafos isomorfos de un número relativamente elevado de vértices fácilmente como sigue:

```
In]:= <<Combinatorica`  
In]:= n=="Número de vértices"  
A1=ToAdjacencyMatrix[RandomGraph[n,.5]];  
P=Permute[IdentityMatrix[n],RandomSample[Range[n]]];  
A2=P.A1.Transpose[P];  
G1=FromAdjacencyMatrix[A1];  
G2=FromAdjacencyMatrix[A2];
```

y comparar el rendimiento:

```
In]:= Timing[Isomorphism[G1,G2,All]]  
Timing[ISOMORFOS2[A1,A2]]
```

Comprobaremos que con unos grafos, una de ellas es más rápida que la otra y viceversa (aunque los tiempos siempre son similares), también dependerá de la cantidad de invariantes que en ISOMORFOS2[] consideremos, ya que esta cuestión en esta función está abierta a las modificaciones que cada usuario quiera realizar, a diferencia de Isomorphism[]. Por otro lado, según lo expuesto en [42], Isomorphism[] parece usar funciones menos frecuentes en otros lenguajes y por tanto está más alejada del espíritu de este libro.

5. COMBINATORIA EN GRAFOS

Encontrar grafos específicos es una tarea de gran complejidad, que exige una gran cantidad de proceso y en ocasiones también de memoria por parte del ordenador. En este epígrafe pretendemos mostrar el problema combinatorio que supone calcular todos los grafos existentes para un número de vértices fijo y estudiarlos salvo isomorfismo. Este estudio permitirá encontrar grafos concretos con las propiedades que deseemos y experimentar conjeturas sobre grafos finitos.

Nos limitaremos al estudio de grafos no orientados, dejando para el lector la resolución análoga del problema para dígrafos (ejercicio 5.22.). También estudiaremos

aquellos grafos con un número fijo de lados y vértices o que además contengan a un grafo dado.

5.1. GRAFOS DE n VÉRTICES

En primer lugar determinamos todas las posibles matrices de adyacencia para un número de vértices fijo n , nos limitaremos a grafos no dirigidos (el caso dirigido sería muy parecido, ejercicio 5.22.). Al final del programa pedimos a Mathematica que nos muestre la memoria que en ese momento está en uso con el objeto de discernir la eficacia y si el problema en cuestión que intentamos resolver está a nuestro alcance.

| PROGRAMA | COMENTARIOS |
|--|---|
| <code>n=NÚMERO DE VÉRTICES;</code> | Introducimos el número de vértices. |
| <code>comb=n*(n-1)/2;</code> <code>listamatrices=Table[0,{indice,2^comb},{k1,n},{k2,n}];</code> | Se define la lista de las matrices necesarias. |
| <code>Do[lista=IntegerDigits[i-1,2];</code> <code>Do[PrependTo[lista,0];{f,comb-Length[lista]}];</code> <code>gg=1;</code> <code>Do[</code> <code>Do[</code> <code>If[lista[[gg]]==1,listamatrices[[i,k2,k1]]=1];</code> <code>gg++;</code> <code>{k1,k2+1,n};,{k2,n-1};</code> <code>listamatrices[[i]]=listamatrices[[i]]</code> <code>+Transpose[listamatrices[[i]]];</code> <code>,{i,2^comb}];</code> | Se calculan todas las matrices y se almacenan en “listamatrices”. |
| <code>Print["FIN"];</code> <code>MemoryInUse[]</code> | Se muestra la memoria usada. |

Programa 5.12. Grafos de n vértices.

El programa construye todas las matrices de adyacencia posibles, para ello y puesto que estas matrices son simétricas se fija sólo en las coordenadas de la matriz por encima de la diagonal principal (compuesta sólo por ceros, porque no consideramos lazos), en total para n vértices habrá $n(n - 1)/2$ posibles lados o unos por encima de la diagonal, por tanto tendremos $2^{n(n - 1)/2}$ matrices. Para construirlas disponemos todas las coordenadas por encima de la diagonal en una única lista o vector de $n(n - 1)/2$ posiciones o coordenadas y conseguimos todas las posibilidades considerando los números enteros que van desde 0 hasta $2^{n(n - 1)/2} - 1$ en base 2, esto es:

| Entero | Base 2 | Matriz de adyacencia |
|--------|------------------|--|
| 0 | $(00\dots 00)_2$ | $\begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix}$ |

| | | |
|--------------------|-----------------|--|
| 1 | $(00\dots01)_2$ | $\begin{pmatrix} 0 & \dots & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & 0 \\ 0 & \dots & 0 & 0 & 1 \\ 0 & \dots & 0 & 1 & 0 \end{pmatrix}$ |
| 2 | $(0\dots010)_2$ | $\begin{pmatrix} 0 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & \dots & 0 & 0 & 0 & 1 \\ 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & \dots & 0 & 1 & 0 & 0 \end{pmatrix}$ |
| 3 | $(0\dots011)_2$ | $\begin{pmatrix} 0 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & \dots & 0 & 0 & 0 & 1 \\ 0 & \dots & 0 & 0 & 0 & 1 \\ 0 & \dots & 0 & 1 & 1 & 0 \end{pmatrix}$ |
| ⋮ | ⋮ | ⋮ |
| $2^{n(n-1)/2} - 1$ | $(11\dots11)_2$ | $\begin{pmatrix} 0 & 1 & 1 & \dots & 1 \\ 1 & 0 & 1 & \dots & 1 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 1 & \dots & 1 & 0 & 1 \\ 1 & \dots & 1 & 1 & 0 \end{pmatrix}$ |

Tabla 5.1.

En la versión 6 de Mathematica y dentro del paquete `Combinatorica` se incorporó una nueva función:

ListGraph[n]

que calcula todos los objetos de tipo grafo asociados a grafos distintos salvo isomorfismo de “n” vértices, como opción podemos indicar si los queremos orientados con “Directed”. También, y dentro del mismo paquete, se tienen las funciones **NumberOfGraphs[n]** y **NumbersOfDirectedGraphs[n]** que proporcionan el número de grafos distintos salvo isomorfismo de “n” vértices.

Ejemplo 5.16. Determinamos cuántos grafos distintos no isomorfos existen de 5 vértices.

Usamos el programa 5.12.:

In[1]:= n=5;

⋮ ⋮

Out[]= FIN
6357792

En “listamatrices” están todas las matrices de adyacencia. Eliminamos los grafos isomorfos, previamente definimos la función 5.9.:²⁸

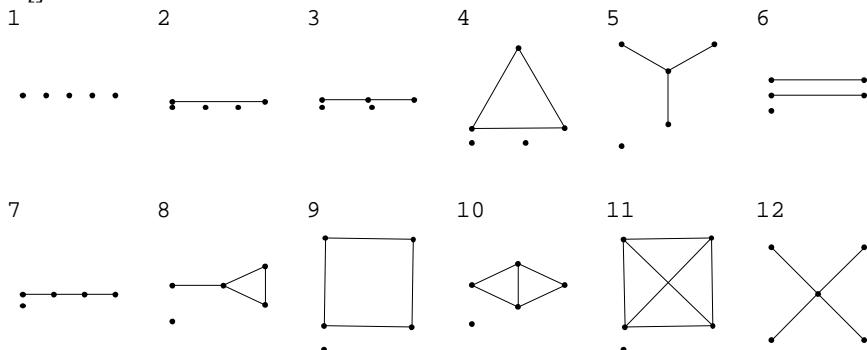
```
In[ ]:= ISOMORFAS[A_]:=Module[{j,matricesP},  
; ;  
In[ ]:= listamatrices2={};  
While[listamatrices!={}],  
AppendTo[listamatrices2,listamatrices[[1]]];  
listamatrices=Complement[listamatrices,  
ISOMORFAS[listamatrices[[1]]]];  
];  
Length[listamatrices2]
```

Out[]= 34

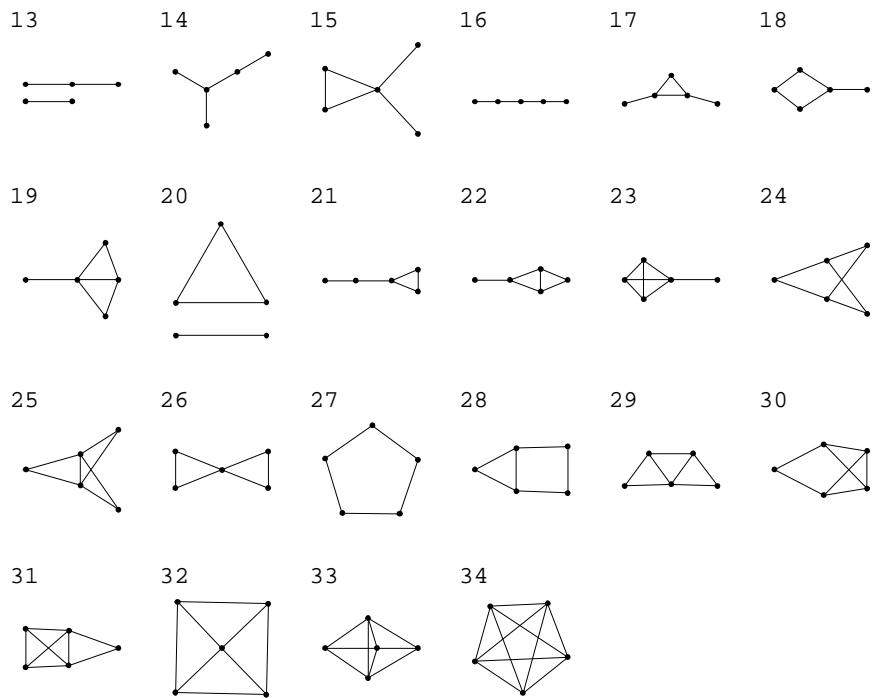
Finalmente los dibujamos:

```
In[ ]:= Do[Print[i];  
Print[GraphPlot[listamatrices2[[i]],  
PlotStyle→{Black,PointSize[.05]}]];  
,{i,Length[listamatrices2]}];
```

*Out[]=*²⁸



²⁸ Mathematica no dibujará los grafos en forma de tabla como se han mostrado aquí por comodidad, sin embargo dispone de una función GraphicsArray[] que podría usarse para que así lo hiciera.



El mismo cálculo con ListGraph[]:

In[]:= **<<Combinatorica`**

In[]:= **ShowGraphArray[ListGraphs[5]]**

que tendrá una salida muy similar a la anterior y que no incluiremos. □

Si elevamos el número de vértices comprobaremos que el cálculo se complica considerablemente:

Ejemplo 5.17. Hacemos el cálculo para 7 vértices, el número de matrices de adyacencia distintas es $2^{21} = 2097152$, luego la variable “listamatrices” almacena una lista de más de 2 millones de matrices 7×7 .

Usamos el programa 5.12.

In[]:= **n=7;**

: **:**

Out[]= **FIN**
824225560

Observemos que para únicamente 7 vértices el número de matrices es enorme y la memoria usada también. Si quisieramos eliminar como en el ejemplo 5.13. los grafos isomorfos y dejar un único representante por clase, además de necesitar una gran cantidad de memoria el proceso se prolongaría por bastantes horas.

Incluso si usamos ListGraph[], el tiempo empleado se dispara a más de ocho minutos, siendo el costo en tiempo para los 8 o más vértices prohibitivo:

```
In[7]:= Timing[ListGraph[7]]
Out[7]= {511.907, {-Graph:<0,7,Undirected>-,
:
:
}
```

□

5.2. GRAFOS DE n VÉRTICES Y m LADOS

El siguiente programa es una variación del 5.12. y consigue calcular todos los grafos de un número fijo de vértices y de lados. Para ello discriminamos aquellas matrices de adyacencia de grafos de n vértices y m lados, esto es, matrices $n \times n$ con $2m$ “unos” (dos “unos” por cada lado $\alpha = \{v_i, v_j\}$, uno en la posición (i, j) y otro en la posición (j, i) de la matriz simétrica de adyacencia), modificamos 5.12. y conforme calculamos todas las matrices de adyacencia nos quedamos sólo con aquellas que representen a grafos del número de lados que deseemos, bastará con contar el número de “unos” que tiene.

| PROGRAMA | COMENTARIOS |
|--|---|
| n=NÚMERO DE VÉRTICES; m=NÚMERO DE LADOS; <pre>comb=(n*(n-1))/2; numeromatrices=comb!/((comb-m)!*m!); listamatrices=Table[0,{indice,numeromatrices},{k1,n}, {k2,n}]; kc=1; Do[lista=IntegerDigits[i-1,2]; If[Sum[lista[[j]],[j,Length[lista]]]==m, matriztemp=Table[0,{k1,n},{k2,n}]; Do[PrependTo[lista,0];,{f,comb-Length[lista]}]; gg=1; Do[Do[If[lista[[gg]]==1,matriztemp[[k2,k1]]=1]; gg++; ,{k1,k2+1,n}];,{k2,n-1}]; listamatrices[[kc]]=matriztemp +Transpose[matriztemp]; kc++;]; ,{i,2^comb}];</pre> | Introducimos el número de vértices. |
| | Se define la lista de las matrices necesarias previamente para ganar en eficacia. |

| | |
|---|--|
| Print["Nº de matrices: ",Length[listamatrices]]; | Se muestra el número de matrices calculadas. |
|---|--|

Programa 5.13. Grafos de n vértices y m lados.

En la versión 6 de Mathematica, la misma función del epígrafe anterior **ListGraph[]** del paquete `<<Combinatorica`` en la forma:

ListGraph[n,m]

calcula todos los objetos de tipo grafo asociados a grafos distintos, salvo isomorfismo, de “n” vértices y “m” lados, como opción podemos indicar si los queremos que sean orientados con “*Directed*”. También, podemos usar las funciones **NumberOfGraphs[n,m]** y **NumbersOfDirectedGraphs[n,m]** que proporcionan el número de grafos distintos salvo isomorfismo de “n” vértices y “m” lados.

Ejemplo 5.18. Calculamos todos los grafos de 6 vértices y 8 lados distintos salvo isomorfismo.

Utilizamos el programa 5.13.:

In[1]:= n=6;
m=8;

⋮ ⋮

Out[1]= Nº de matrices: 6435

Eliminamos los grafos isomorfos, de nuevo usamos la función 5.9.:

In[2]:= ISOMORFAS[A_]:=Module[{j,matricesP},
⋮ ⋮
In[3]:= listamatrices2={};
While[listamatrices!={}],
AppendTo[listamatrices2,listamatrices[[1]]];
listamatrices=Complement[listamatrices,
ISOMORFAS[listamatrices[[1]]]];
};
Length[listamatrices2]

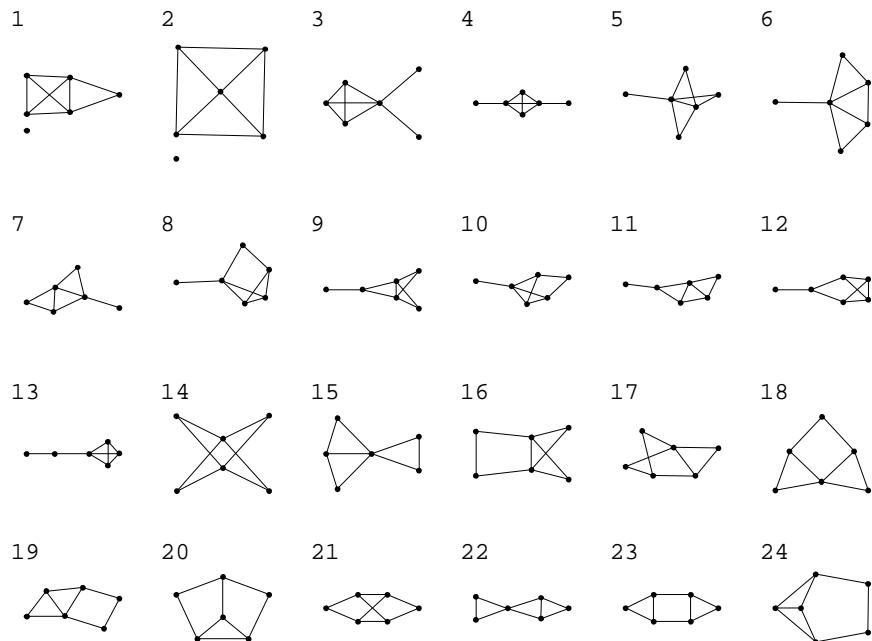
Out[3]= 24

Finalmente los dibujamos:

In[4]:= Do[Print[i];
Print[GraphPlot[listamatrices2[[i]],
PlotStyle→{Black,PointSize[.05]}]];

```
,{i,Length[listamatrices2]}];
```

Out[]=



El mismo cálculo con ListGraph[]:

In[]:= <<Combinatorica`

In[]:= ShowGraphArray[ListGraphs[6,8]]

que tendrá una salida muy similar que no incluimos por cuestiones de espacio. □

5.3. CONSTRUCCIÓN DE GRAFOS

Los programas anteriores son difícilmente utilizables para un número de vértices elevado, tanto por la cantidad de proceso necesaria, como por la memoria que usamos para almacenar las numerosas matrices de adyacencia que van apareciendo. Como hemos podido comprobar en los ejemplos, en realidad el número de grafos salvo isomorfismo no es elevado, y además al final esta información es la que nos interesa, por tanto podríamos intentar quedarnos sólo con un único representante por clase de isomorfía, o matriz de adyacencia de entre todas aquellas que representan a grafos isomorfos. De esta forma al menos evitaremos el problema relacionado con la cantidad de memoria necesaria.

Para determinar todos los grafos distintos salvo isomorfismo, vamos a pensar en un método algo más elegante y efectivo que el anterior, en vez de considerar todas las

matrices posibles, vamos a construir todos los grafos distintos salvo isomorfismo, la idea que seguiremos consistirá en ir añadiendo los lados de uno en uno y a cada paso eliminar los que generemos y que sean isomorfos.

En primer lugar construimos una función que tenga por entrada un grafo cualquiera y por salida todos los grafos que resultan al añadirle un lado al de entrada:

| FUNCIÓN | COMENTARIOS |
|---|--|
| Añadirlado[matrizadyacencia_]:=Module[{i,j,matriz}, | Función cuya única entrada será la matriz de adyacencia del grafo. |
| listamatrices={}; Do[Do[If[matrizadyacencia[[j,i]]==0, matriz=matrizadyacencia; matriz[[j,i]]=1; matriz[[i,j]]=1; AppendTo[listamatrices,matriz];]; ,{j,i-1}]; ,{i,Dimensions[matrizadyacencia][[1]]}]; listamatrices]; | Se añade un lado más (o dos unos a la matriz de adyacencia) al grafo, se consideran todas las posibilidades. |
| | Se muestran los resultados. |

Función 5.14. Añadir un lado a un grafo.

Ejemplo 5.19. Calculamos todos los grafos que resultan, salvo isomorfismo, al añadirle un lado al grafo G cuya matriz de adyacencia es:

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

Introducimos la matriz y definimos la función 5.14.:

In[]:= matrizadyacencia={{0,1,0,0,0},{1,0,1,0,1},{0,1,0,1,0},
 {0,0,1,0,0},{0,1,0,0,0}};

In[]:= Añadirlado[matrizadyacencia_]:=Module[{i,j,matriz},

⋮ ⋮

Añadimos un nuevo lado:

In[]:= Añadirlado[matrizadyacencia]

Out[]=
{{{{0,1,1,0,0},{1,0,1,0,1},{1,1,0,1,0}},

```

{ { 0,0,1,0,0 }, { 0,1,0,0,0 } },
{ { 0,1,0,1,0 }, { 1,0,1,0,1 }, { 0,1,0,1,0 } },
{ { 1,0,1,0,0 }, { 0,1,0,0,0 } },
{ { 0,1,0,0,0 }, { 1,0,1,1,1 }, { 0,1,0,1,0 } },
{ { 0,1,1,0,0 }, { 0,1,0,0,0 } },
{ { 0,1,0,0,1 }, { 1,0,1,0,1 }, { 0,1,0,1,0 } },
{ { 0,0,1,0,0 }, { 1,1,0,0,0 } },
{ { 0,1,0,0,0 }, { 1,0,1,0,1 }, { 0,1,0,1,1 } },
{ { 0,0,1,0,0 }, { 0,1,1,0,0 } },
{ { 0,1,0,0,0 }, { 1,0,1,0,1 }, { 0,1,0,1,0 } },
{ { 0,0,1,0,1 }, { 0,1,0,1,0 } }
}

```

Nos quedamos sólo con un representante por clase de isomorfía, para ello usamos 5.9.:

```

In]:= ISOMORFAS[A_]:=Module[{j,matricesP},
];
;
;

In]:= listamatrices2={};
While[listamatrices!={},
AppendTo[listamatrices2,listamatrices[[1]]];
listamatrices=Complement[listamatrices,
ISOMORFAS[listamatrices[[1]]]];
];
Length[listamatrices2]

```

Out]= 4

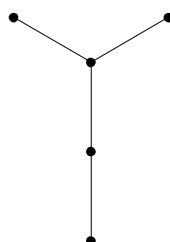
Por último, representamos gráficamente tanto G , como a los grafos obtenidos:

```

In]:= GraphPlot[matrizadyacencia,
PlotStyle→{Black,PointSize[.05]}]

```

Out]=

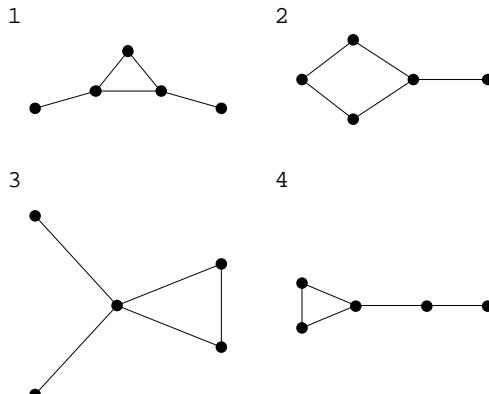


```

In]:= Do[Print[i];
Print[GraphPlot[listamatrices2[[i]],
PlotStyle→{Black,PointSize[.05]}]];
,{i,Length[listamatrices2]}];

```

Out[]=



□

Usando la función anterior reiteradamente podemos construir:

- Todos los grafos que contienen a uno dado de un número de vértices fijo mayor o igual al número de vértices que tiene el grafo de partida (reiterando el proceso, podremos variar el número de vértices).
- Todos los grafos que contienen a uno dado con un número de lados y vértices fijado de antemano.
- Todos los grafos de un número de vértices fijo.

Para ello programamos la siguiente rutina que va añadiendo lados: un lado al grafo de partida, otro a cada uno de los resultantes y así sucesivamente, para agilizar el proceso también va eliminando los isomorfos a cada paso.

| PROGRAMA | COMENTARIOS |
|---|--|
| ISOMORFAS[A_]:=... | Definimos la función 5.9. |
| Añadirlado[matrizadyacencia]:=... | Definimos la función 5.14. |
| nlados=NÚMERO DE LADOS A AÑADIR; matrizadyacencia=MATRIZ DE ADYACENCIA; n=Dimensions[matrizadyacencia][[1]]; (NÚMERO DE VÉRTICES) | Introducimos el número de vértices, el número de lados que añadimos y la matriz de adyacencia del grafo de partida que estará contenido en todos los que construyamos. |
| tiempo=TimeUsed[]; listamatrices1={matrizadyacencia}; listamatrices2=listamatrices1; Do[| |
| listamatrices3={}; Do[listamatrices3=Union[listamatrices3, Añadirlado[listamatrices2[[k2]]]; ,{k2,Length[listamatrices2]}]; Print["De ",k," lados: ",Length[listamatrices3]]; | Añadimos un lado a todos los grafos, consideramos todas las posibilidades Añadimos un lado a cada vuelta del bucle a todos los grafos construidos hasta el momento. |

| | |
|--|---|
| <pre>listamatrices2={}; While[listamatrices3≠{}, AppendTo[listamatrices2,listamatrices3[[1]]]; listamatrices3=Complement[listamatrices3, ISOMORFAS[listamatrices3[[1]]]];]; listamatrices1=Join[listamatrices1,listamatrices2]; Print["Quedan : ",Length[listamatrices2], ". Total: ",Length[listamatrices1]]; ,{k,nlados}]; Print["Totales: ",Length[listamatrices1], " Tiempo empleado: ",TimeUsed[]-tiempo];</pre> | <p>Eliminamos los isomorfos.</p> |
| <pre>listamatrices1=Join[listamatrices1,listamatrices2]; Print["Quedan : ",Length[listamatrices2], ". Total: ",Length[listamatrices1]]; ,{k,nlados}];</pre> | <p>Mostramos información parcial.</p> |
| <pre>Print["Totales: ",Length[listamatrices1], " Tiempo empleado: ",TimeUsed[]-tiempo];</pre> | <p>Se muestra el número de matrices calculadas.</p> |

Programa 5.15. Grafos que contienen a uno dado con un número máximo de lados.

Ejemplo 5.20. Determinamos todos los grafos, salvo isomorfismo, de 5 vértices que contienen al pentágono.

La matriz de adyacencia del pentágono es:

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

Puesto que el pentágono tiene 5 lados y el número máximo de lados de un grafo con 5 vértices es 10, podremos añadir hasta otros 5 lados. Luego usando el programa 5.15., primero definimos las funciones 5.9. y 5.14. e introducimos los lados a añadir “nlados=5”, los vértices “n=5” y matriz de adyacencia del pentágono:

```
In]:= ISOMORFAS[A_]:=Module[{j,matricesP},
  ];
In]:= Añadirlado[matrizadyacencia_]:=Module[{i,j,matriz},
  ];
In]:= nlados=5;
n=5;
matrizadyacencia={{0,1,0,0,1},{1,0,1,0,0},
  {0,1,0,1,0},{0,0,1,0,1},{1,0,0,1,0}};
  ];
Out]:= De 1 lados: 5
Quedan : 1. Total: 2
De 2 lados: 4
```

```

Quedan : 2. Total: 4
De 3 lados: 5
Quedan : 2 . Total: 6
De 4 lados: 3
Quedan : 1. Total: 7
De 5 lados: 1
Quedan : 1. Total: 8
Totales: 8 Tiempo empleado: 0.078

```

Por último los dibujamos:

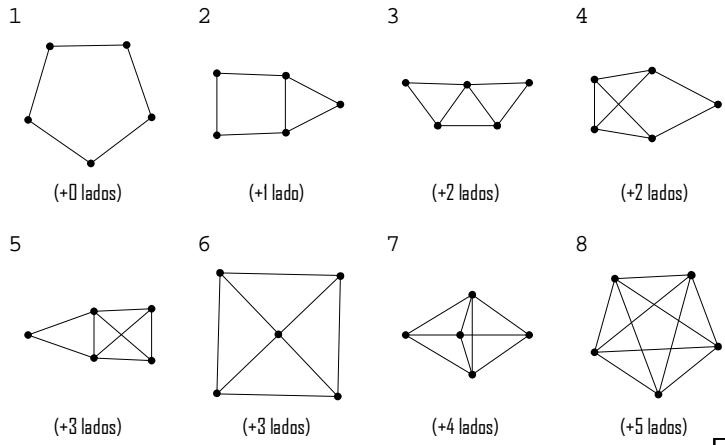
In[]:=

```

Do[Print[i];
Print[GraphPlot[listamatrices1[[i]],
PlotStyle→{Black,PointSize[.05]}]];
,{i,Length[listamatrices1]}];

```

Out[]:=



□

Ejemplo 5.21. Determinamos todos los grafos, salvo isomorfismo:

- De 6 vértices y 11 lados que contienen al hexágono.
- De 6 vértices y 10 lados que contienen al pentágono.
- La matriz de adyacencia del es hexágono es:

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

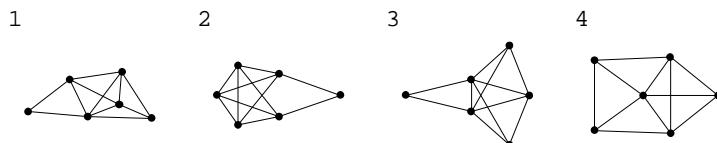
Usamos el programa 5.15., añadimos a los 6 lados que ya tiene el hexágono otros 5 más:

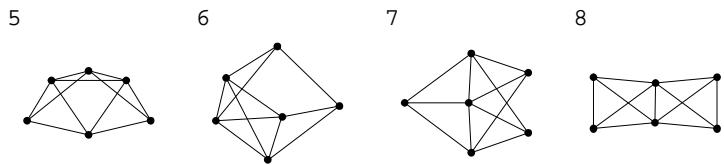
```
In[]:= ISOMORFAS[A_]:=Module[{j,matricesP},
                                ];
In[]:= AñadirLado[matrizadyacencia_]:=Module[{i,j,matriz},
                                ];
In[]:= nlados=5;
n=6;
matrizadyacencia={{0,1,0,0,0,1},{1,0,1,0,0,0},{0,1,0,1,0,0},
                   {0,0,1,0,1,0},{0,0,0,1,0,1},{1,0,0,0,1,0}};
In[]:=
Out[]=
De 1 lados: 9
Quedan: 2. Total: 3
De 2 lados: 15
Quedan: 6. Total: 9
De 3 lados: 31
Quedan: 11. Total: 20
De 4 lados: 46
Quedan: 11. Total: 31
De 5 lados: 36
Quedan: 8. Total: 39
Totales: 39 Tiempo empleado: 3.64
```

Observemos que en “listamatrices1” se almacenan el total de grafos y en “listamatrices2” los últimos 10 que se han calculado y que resultan al añadir el último lado. Luego los grafos que dan respuesta al problema propuesto son:

```
In[]:= Do[Print[i];
          Print[GraphPlot[listamatrices2[[i]],
                         PlotStyle→{Black,PointSize[.05]}]];
          ,{i,Length[listamatrices2]}];
```

Out[] =





- b) En este caso tendremos que considerar la matriz de adyacencia del pentágono y añadir un sexto vértice aislado:

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

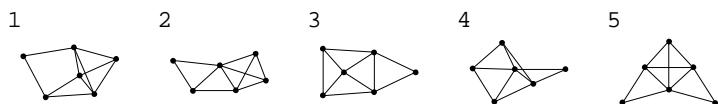
```
In[7]:= nlados=5;
n=6;
matrizadyacencia={{0,1,0,0,1,0},{1,0,1,0,0,0},{0,1,0,1,0,0},
{0,0,1,0,1,0},{1,0,0,1,0,0},{0,0,0,0,0,0}};
```

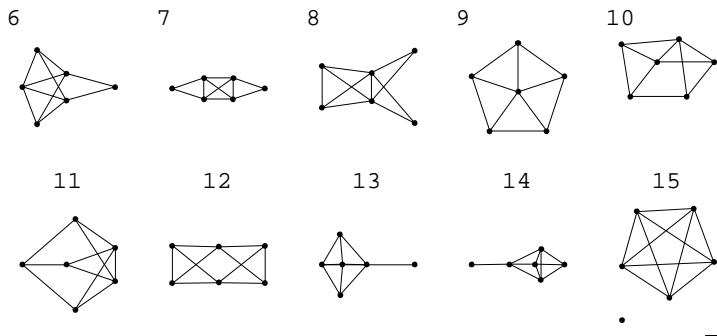
⋮ ⋮

```
Out[7]= De 1 lados: 10
Quedan: 2. Total: 3
De 2 lados: 17
Quedan: 7. Total: 10
De 3 lados: 43
Quedan: 15. Total: 25
De 4 lados: 72
Quedan: 18. Total: 43
De 5 lados: 75
Quedan: 15. Total: 58
Totales: 58 Tiempo empleado: 3.141
```

```
In[8]:= Do[Print[i];
Print[GraphPlot[listamatrices2[[i]],
PlotStyle→{Black,PointSize[.05]}]];
,{i,Length[listamatrices2]}];
```

Out[8]=





Ejemplo 5.22. Determinamos salvo isomorfismo todos los grafos de 5 vértices como hicimos en el ejemplo 5.14. usando el programa 5.15.

In[]:=

ISOMORFAS[A_]:=Module[{j,matricesP},

⋮

⋮

In[]:=

AñadirLado[matrizadyacencia_]:=Module[{i,j,matriz},

⋮

⋮

In[]:=

n=5;
nlados=n*(n-1)/2;
matrizadyacencia=Table[0,{i,n},{j,n}];

⋮

⋮

Out[]=

De 1 lados: 10
 Quedan : 1. Total: 2
 De 2 lados: 9
 Quedan: 2. Total: 4
 De 3 lados: 15
 Quedan: 4. Total: 8
 De 4 lados: 25
 Quedan: 6. Total: 14
 De 5 lados: 32
 Quedan: 6. Total: 20
 De 6 lados: 23
 Quedan: 6. Total: 26
 De 7 lados: 19
 Quedan: 4. Total: 30
 De 8 lados: 9
 Quedan: 2. Total: 32
 De 9 lados: 3
 Quedan: 1. Total: 33
 De 10 lados: 1
 Quedan: 1. Total: 34
 Totales: 34 Tiempo empleado: 0.219

□

Observemos como la efectividad es considerablemente mayor, tanto en lo referente a memoria usada como en el tiempo empleado, que con el método directo que vimos en el epígrafe 5.1. de este capítulo.

El problema contrario, esto es, determinar todos los grafos contenidos en uno dado, es un problema más sencillo de resolver que los anteriores, estaríamos hablando en realidad de los subgrafos de un grafo dado, por lo que remitimos su estudio al capítulo 6.

5.4. EFICACIA Y OPTIMIZACIÓN

En el ejemplo 5.17. pudimos comprobar cómo incluso para 7 vértices el cálculo de todos los grafos salvo isomorfismo se complicaba mucho, incluso si usamos la función `ListGraph[]`, nueva en la versión 6 de Mathematica, comprobamos en el mismo ejemplo que con 7 o más vértices el tiempo de cálculo se dispara, por lo que ésta no será una opción razonable a tener en cuenta. Además obsérvese que la variable “`matricesP`” (en 5.10.) almacena una lista de $n!$ matrices para n vértices. Luego si manejáramos por ejemplo 12 vértices entonces almacenaría 12! matrices, esto es, unos 500 millones de matrices 12×12 , lo que provocaría un uso de memoria superior al estándar del que en la actualidad disponen los ordenadores domésticos. Aunque evitemos utilizar “`matricesP`” y así, una cantidad limitada de memoria nos permitiese abordar los cálculos, seguiríamos enfrentándonos a los casi 500 millones de permutaciones posibles, lo que haría prácticamente imposible hacer comprobaciones en este sentido. En 4.1. se mejoró considerablemente la función `ISOMORFOS[]` (5.10.), creando una nueva función `ISOMORFOS2[]` (5.11.) que no utilizaba “`matricesP`” y que construía, caso de existir los isomorfismos. En esta misma línea, en este epígrafe vamos a optimizar los métodos anteriores, mejoraremos su eficacia, tanto en el tiempo de proceso como en el uso de la memoria. Como comprobaremos la optimización de los métodos lleva consigo gran cantidad de código asociado, veremos que los invariantes asociados a las clases de isomorfía de matrices de adyacencia, en ocasiones particulares, constituyen una condición suficiente para que dos matrices de adyacencia pertenezcan a la misma clase de isomorfía, esto es, representen a grafos isomorfos. En los ejemplos 5.12. y 5.13. pudimos comprobar cómo los invariantes no siempre son condición suficiente para que dos matrices de adyacencia representen a grafos isomorfos, luego el problema en ocasiones puede ser bastante complejo en el sentido siguiente: averiguar cuáles son los invariantes suficientes para que dos matrices de adyacencia pertenezcan a la misma clase de isomorfía.

El problema de los métodos anteriores radica en determinar grafos únicos salvo isomorfía (con ciertas propiedades particulares según el problema que nos planteemos), esto es, una vez construidos los grafos tendremos que distinguir cuáles de ellos son isomorfos y debido a la cantidad de posibles permutaciones cuando el número de vértices es elevado, nos vemos obligados a realizar multitud de comprobaciones. Para evitarlo intentaremos construir los grafos bajo cierto criterio que permita eliminar a cada paso los

isomorfos con cierta comodidad, de manera que no acumulemos²⁹ demasiados grafos isomorfos.

Distinguiremos dos posibilidades que trataremos de forma distinta según dispongamos o no de invariantes asociados a las matrices de adyacencia de grafos isomorfos que supongan una condición suficiente.

5.4.1. NO DISPONEMOS DE UNA CONDICIÓN SUFICIENTE

Lo dividimos en tres pasos:

- a) Modificamos la función ISOMORFOS2[] (5.11.) y construimos una que simplemente testeé si dos matrices de adyacencia se corresponden con grafos isomorfos, construyendo al menos un isomorfismo, usará los invariantes asociados a los vértices de cada grafo, que se corresponderán con “tabla1” y “tabla2”, la comprobación se hace de forma constructiva; esto es, se construye, si es posible, un isomorfismo entre los grafos:

| FUNCIÓN | COMENTARIOS |
|---|---|
| ISOMORFOSPARTE2[A1_,A2_,tabla1_,tabla2_]:= Module[{n,permutaciones2,adyacentes1,condicion, imagenes,i,j,k,h,kk,permutaciones}, | La función tendrá 4 argumentos, las matrices de adyacencia y los invariantes asociados a los vértices que permiten agilizar la construcción del isomorfismo en el caso en que exista. |
| sonisomorfos=False; n=Dimensions[A1][[1]]; permutaciones=Position[tabla2,tabla1[[1]]]; | Buscamos las posibles imágenes del primer vértice. |
| Do[| Bucle que busca correspondencia entre los $(n - 1)$ vértices restantes. |
| permutaciones2={}; adyacentes1=Position[A1[[i]],1]; | Almacenamos la información respecto a los adyacentes al vértice i -ésimo. |
| Do[| Bucle que recorre las posibilidades construidas hasta el momento. |
| imagenes=Complement[Position[tabla2,tabla1[[i]]], Table[{permutaciones[[j]][[kk]]} ,{kk,Length[permutaciones[[j]]]}]; | Posibles imágenes del vértice i -ésimo. |
| Do[condicion=True; Do[| Recorremos las posibles imágenes y comprobamos si conservan la adyacencia. |
| If[adyacentes1[[k]][[1]]<i, If[A2[[permutaciones[[j]]] adyacentes1[[k]][[1]] $,$ | |

²⁹ Aunque mejoremos sustancialmente con estas ideas la eficacia respecto a los métodos anteriores, al elevar el número de vértices, volveremos a topar con una cantidad de proceso y complejidad enorme.

| | |
|--|-----------------------|
| <pre> imagenes[[h]][[1]]!=1 ,condicion=False;Break[];};]; ,{k,Length[adyacentes1]}]; If[condicion, permutaciones2=Join[permutaciones2, {Join[permutaciones[[j]], {imagenes[[h]][[1]]}]}]; If[i==n,sonisomorfos=True;Break[]];]; ,{h,Length[imagenes]}]; If[Mod[j,1000]==0,Print["Van ",j," ",Length[permutaciones2]]; If[sonisomorfos,Break[]]; ,{j,Length[permutaciones]}]; permutaciones=permutaciones2; ,{i,2,n}]; sonisomorfos]; </pre> | |
| | |
| | |
| | Salida de resultados. |

Función 5.16. Grafos isomorfos.

- b) Construimos otra función que tendrá por entrada una lista de matrices de adyacencia, y por salida la lista de las matrices de adyacencia asociadas a grafos no isomorfos que resulta de la de entrada, esto es, después de eliminar las matrices asociadas a grafos isomorfos y dejar un único representante por clase de isomorfía.

| FUNCIÓN | COMENTARIOS |
|---|---|
| ISOMORFOSPARTE2[A1_,A2_,tabla1_,tabla2_]:=... | Definimos la función 5.16. |
| QUITARISOMORFOS[listamatrices_]:= Module[{i,i1,i2,invariantes,potencias,posiciones,kk,j, invariantesordenados,n,tabla,possibleisomorfos, clase,listamat,isomorfos,k3}, invariantes={}; invariantesordenados={}; possibleisomorfos={}; n=Dimensions[listamatrices[[1]][[1]]]; potencias=3; Do[tabla=Table[Table[MatrixPower[listamatrices[[i]],kk+1][[j,j]] ,{kk,potencias}],{j,n}]; AppendTo[invariantes,tabla]; AppendTo[invariantesordenados,Sort[tabla]]; {i,Length[listamatrices]}]; | La función tendrá por entrada una lista de matrices de adyacencia de grafos con igual número de vértices. |
| | Invariantes que vamos a usar para acelerar la comprobación, estos son opcionales y podremos variarlos según nuestros intereses. |
| posiciones=Table[{i},{i,Length[listamatrices]}]; While[posiciones!={}, clase=Position[invariantesordenados, invariantesordenados[[posiciones[[1]][[1]]]]]; AppendTo[possibleisomorfos,clase]; posiciones=Complement[posiciones,clase];]; Print["DIVIDIMOS EN ", | Partimos el conjunto de todas las matrices de adyacencia en clases con los mismos invariantes. |

| | |
|---|-----------------------|
| <pre>Length[posiblesisomorfos], " CLASES"];</pre> | |
| <pre>listamat2={};</pre> | |
| <pre>Do[</pre> | |
| <pre> listamat=Table[</pre> | |
| <pre> {listamatrices[[posiblesisomorfos[[j]][[k1]][[1]]]],</pre> | |
| <pre> invariantes[[posiblesisomorfos[[j]][[k1]][[1]]]]}</pre> | |
| <pre> ,{k1,Length[posiblesisomorfos[[j]]]}];</pre> | |
| <pre> If[Length[listamat]==1,</pre> | |
| <pre> AppendTo[listamat2,listamat[[1]][[1]]];</pre> | |
| <pre> ,</pre> | |
| <pre> kk=0;</pre> | |
| <pre> While[listamat!={}>,</pre> | |
| <pre> AppendTo[listamat2,listamat[[1]][[1]]];</pre> | |
| <pre> kk++;</pre> | |
| <pre> isomorfos={};</pre> | |
| <pre> If[Length[listamat]>1,</pre> | |
| <pre> Do[</pre> | |
| <pre> If[</pre> | |
| <pre> ISOMORFOSPARTE2[</pre> | |
| <pre> listamat[[1]][[1]],</pre> | |
| <pre> listamat[[k3]][[1]],</pre> | |
| <pre> listamat[[1]][[2]],</pre> | |
| <pre> listamat[[k3]][[2]]],</pre> | |
| <pre> AppendTo[isomorfos,{k3}];</pre> | |
| <pre>];</pre> | |
| <pre> ,{k3,2,Length[listamat]}];</pre> | |
| <pre>];</pre> | |
| <pre> listamat=Delete[listamat,</pre> | |
| <pre> Union[{{1}},isomorfos]];</pre> | |
| <pre>];</pre> | |
| <pre> If[kk!=1,</pre> | |
| <pre> Print["En la clase ", j, " hay ",kk,</pre> | |
| <pre> "que no son isomorfos"];</pre> | |
| <pre>];</pre> | |
| <pre>];</pre> | |
| <pre>,{j,Length[posiblesisomorfos]}];</pre> | |
| <pre>listamat2</pre> | |
| <pre>];</pre> | Salida de resultados. |

Función 5.17. Grafos isomorfos.

- c) Sea G un (p, q) -grafo, usando a) y b) programamos la siguiente rutina que construye todos los grafos que contienen al grafo G que introducimos en “matrizadyacencia”, con p vértices y a lo sumo $(q + k)$ lados, introduciendo p en la variable “n” y k en “nlados”:

| PROGRAMA | COMENTARIOS |
|---|--|
| Añadirlado[matrizadyacencia]:=... | Definimos la función 5.14. |
| ISOMORFOSPARTE2[A1,A2,tabla1,tabla2]:=... | Definimos la función 5.16. |
| QUITARISOMORFOS[listamatrices]:=... | Definimos la función 5.17. |
| nlados=NÚMERO DE LADOS; | Introducimos el número de lados máximo que se añadirán al grafo con matriz de adyacencia “matrizadyacencia”. |
| n=NÚMERO DE VÉRTICES; | Introducimos el número de vértices o número de filas y columnas de la matriz |

| | |
|--|---|
| | “matrizadyacencia”. |
| matrizadyacencia=MATRIZ DE ADYACENCIA; | Matriz de adyacencia del grafo que queremos que esté contenido en todos los grafos que se construyan. |
| tiempo=TimeUsed[]; | Fijamos el tiempo de proceso usado por Mathematica hasta el momento para determinar al final el tiempo total empleado en este método. |
| listamatrices1={matrizadyacencia}; | En “listamatrices1” almacenaremos todos los grafos calculados. |
| listamatrices2=listamatrices1; | En “listamatrices2” almacenaremos los grafos resultantes tras añadir el último lado. |
| Do[| Bucle que añade a cada ciclo un nuevo lado a los grafos resultantes. |
| listamatrices3={}; | |
| Print["-----> Añadimos nuevos lado a ", Length[listamatrices2]," grafos"]; | Indicamos los cálculos a realizar en este paso. |
| Do[| Se añade un lado a cada uno de los grafos de “listamatrices2”. |
| listamatrices3=Union[listamatrices3, | |
| AñadirLado[listamatrices2[[k2]]]; | Podemos utilizar la función alternativa AñadirLado2[] (5.20.) que se definirá más adelante. |
| If[Mod[k2,100]==1,Print["Van ",k2];]; | Indicamos cuantos van, cada cien grafos nuevos añadidos. |
| ,{k2,Length[listamatrices2]}; | |
| Print["***** DE ",k," LADOS: ", Length[listamatrices3]," *****"; | Mostramos resultados parciales. |
| listamatrices2= QUITARISOMORFOS[listamatrices3]; | Si así lo decidíramos podemos usar QUITARISOMORFOS2[] (5.19.) como se explica más adelante. |
| listamatrices1=Join[listamatrices1,listamatrices2]; Print[TimeUsed[]-tiempo, ". Total: ",Length[listamatrices1]]; | Mostramos resultados parciales y tiempo empleado hasta el momento. |
| ,{k,nlados}]; | |
| Print["Totales: ",Length[listamatrices1], " Tiempo empleado: ",TimeUsed[]-tiempo]; | Resultado final |

Programa 5.18. Combinatoria en grafos.

Los siguientes ejemplo ilustran como podemos usar 5.16., 5.17. y 5.18.:

Ejemplo 5.23. Buscamos todos los grafos de 7 vértices distintos salvo isomorfismo. En primer lugar definimos las funciones 5.14., 5.16. y 5.17., usamos el programa 5.18.:

In//]:= AñadirLado[matrizadyacencia_]:=Module[{i,j,matriz},

⋮ ⋮

In//]:= ISOMORFOSPARTE2[A1_,A2_,tabla1_,tabla2_]:=Module[

⋮ ⋮

In//]:= QUITARISOMORFOS[listamatrices_]:=Module[

In[]:=

```
n=7;
nlados=(n*(n-1))/2;
matrizadyacencia=Table[0,{i,n},{j,n}];
```

*⋮**Out[] =*

```
-----> Añadimos nuevo lado a 1 grafos
Van 1 ***** DE 1 LADOS: 21 *****
DIVIDIMOS EN 1 CLASES
0.531. Total: 2
-----> Añadimos nuevo lado a 1 grafos
Van 1 ***** DE 2 LADOS: 20 *****
DIVIDIMOS EN 2 CLASES
0.672. Total: 4
-----> Añadimos nuevo lado a 2 grafos
Van 1 ***** DE 3 LADOS: 37 *****
DIVIDIMOS EN 5 CLASES
0.797. Total: 9
-----> Añadimos nuevo lado a 5 grafos
Van 1 ***** DE 4 LADOS: 86 *****
DIVIDIMOS EN 10 CLASES
1.062. Total: 19
-----> Añadimos nuevo lado a 10 grafos
Van 1 ***** DE 5 LADOS: 157 *****
DIVIDIMOS EN 21 CLASES
1.468. Total: 40
-----> Añadimos nuevo lado a 21 grafos
Van 1 ***** DE 6 LADOS: 306 *****
DIVIDIMOS EN 41 CLASES
2.265. Total: 81
-----> Añadimos nuevo lado a 41 grafos
Van 1 ***** DE 7 LADOS: 533 *****
DIVIDIMOS EN 65 CLASES
3.718. Total: 146
-----> Añadimos nuevo lado a 65 grafos
Van 1 ***** DE 8 LADOS: 759 *****
DIVIDIMOS EN 97 CLASES
5.656. Total: 243
-----> Añadimos nuevo lado a 97 grafos
Van 1 ***** DE 9 LADOS: 1033 *****
DIVIDIMOS EN 131 CLASES
8.562. Total: 374
-----> Añadimos nuevo lado a 131 grafos
Van 1
Van 101 ***** DE 10 LADOS: 1271 *****
DIVIDIMOS EN 148 CLASES
12.156. Total: 522
-----> Añadimos nuevo lado a 148 grafos
Van 1
Van 101 ***** DE 11 LADOS: 1293 *****
DIVIDIMOS EN 148 CLASES
16.062. Total: 670
-----> Añadimos nuevo lado a 148 grafos
Van 1
Van 101 ***** DE 12 LADOS: 1152 *****
DIVIDIMOS EN 131 CLASES
19.468. Total: 801
-----> Añadimos nuevo lado a 131 grafos
Van 1
Van 101 ***** DE 13 LADOS: 900 *****
DIVIDIMOS EN 97 CLASES
22.187. Total: 898
-----> Añadimos nuevo lado a 97 grafos
Van 1 ***** DE 14 LADOS: 591 *****
DIVIDIMOS EN 65 CLASES
23.984. Total: 963
-----> Añadimos nuevo lado a 65 grafos
Van 1 ***** DE 15 LADOS: 339 *****
DIVIDIMOS EN 41 CLASES
24.937. Total: 1004
-----> Añadimos nuevo lado a 41 grafos
Van 1 ***** DE 16 LADOS: 182 *****
DIVIDIMOS EN 21 CLASES
25.656. Total: 1025
-----> Añadimos nuevo lado a 21 grafos
Van 1 ***** DE 17 LADOS: 79 *****
DIVIDIMOS EN 10 CLASES
25.968. Total: 1035
-----> Añadimos nuevo lado a 10 grafos
Van 1 ***** DE 18 LADOS: 30 *****
DIVIDIMOS EN 5 CLASES
26.172. Total: 1040
-----> Añadimos nuevo lado a 5 grafos
Van 1 ***** DE 19 LADOS: 11 *****
DIVIDIMOS EN 2 CLASES
26.281. Total: 1042
-----> Añadimos nuevo lado a 2 grafos
Van 1 ***** DE 20 LADOS: 3 *****
DIVIDIMOS EN 1 CLASES
26.375. Total: 1043
-----> Añadimos nuevo lado a 1 grafos
Van 1 ***** DE 21 LADOS: 1 *****
DIVIDIMOS EN 1 CLASES
26.375. Total: 1044
Totales: 1044 Tiempo empleado: 26.375
```

Si comparamos la eficacia respecto método usado en el ejemplo 5.17., observaremos que la mejora es sustancial

□

Ejemplo 5.24. Buscamos todos los grafos de 9 vértices distintos salvo isomorfismo que contengan al eneágono de 11 lados. Definimos las funciones 5.14., 5.16. y 5.17., usamos el programa 5.18.:.

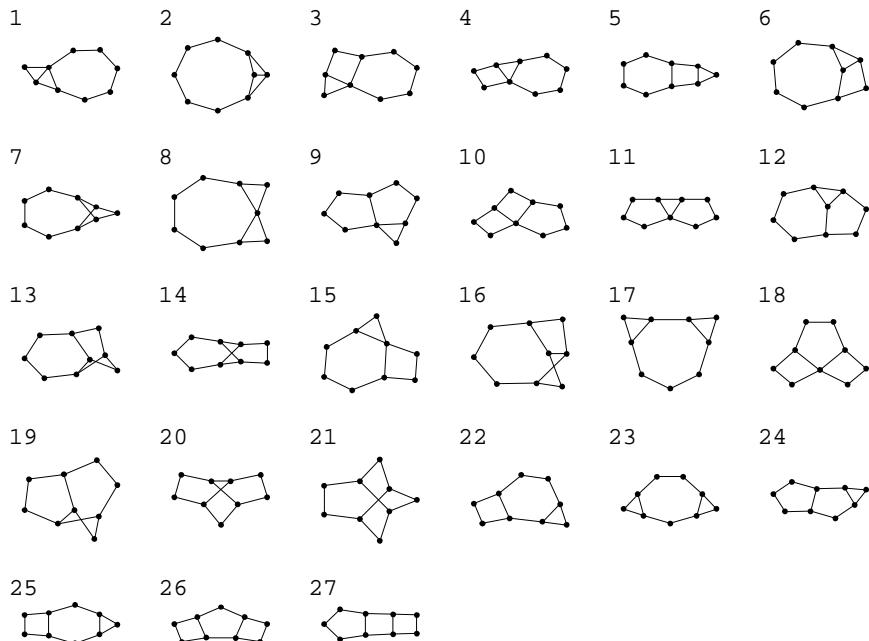
```
In]:= AñadirLado[matrizadyacencia_]:=Module[{i,j,matriz},
 $\vdots$   $\vdots$ 
In]:= ISOMORFOSPARTE2[A1_,A2_,tabla1_,tabla2_]:=Module[
 $\vdots$   $\vdots$ 
In]:= QUITARISOMORFOS[listamatrices_]:=Module[
 $\vdots$   $\vdots$ 
In]:= n=9;
nlados=2;
matrizadyacencia={{0,1,0,0,0,0,0,0,1},{1,0,1,0,0,0,0,0,0},
{0,1,0,1,0,0,0,0,0},{0,0,1,0,1,0,0,0,0},
{0,0,0,1,0,1,0,0,0},{0,0,0,0,1,0,1,0,0},
{0,0,0,0,0,1,0,1,0},{0,0,0,0,0,0,1,0,1},
{1,0,0,0,0,0,1,0,0}};
 $\vdots$   $\vdots$ 
Out]= -----> Añadimos nuevo lado a 1 grafos
Van 1
***** DE 1 LADOS: 27 *****
DIVIDIMOS EN 3 CLASES
0.11. Total: 4
-----> Añadimos nuevo lado a 3 grafos
Van 1
***** DE 2 LADOS: 75 *****
DIVIDIMOS EN 27 CLASES
0.329. Total: 31
Totales: 31 Tiempo empleado: 0.329
```

En “listamatrices1” están almacenados los grafos de 9, 10 y 11 lados que contienen al eneágono y “listamatrices2” contiene los grafos de 11 lados.

Los representamos:

```
In]:= Do[Print[i];
Print[GraphPlot[listamatrices2[[i]],
PlotStyle→{Black,PointSize[.05]}]];
,{i,Length[listamatrices2]}];
```

Out]=



□

Ejemplo 5.25. Buscamos todos los grafos de 12 vértices distintos salvo isomorfismo que contengan al dodecágono y tengan a lo sumo 14 lados. Definimos las funciones 5.14., 5.16. y 5.17., usamos el programa 5.18.:

```

In[]:= AñadirLado[matrizadyacencia_]:=Module[{i,j,matriz},
];
;

In[]:= ISOMORFOSPARTE2[A1_,A2_,tabla1_,tabla2_]:=Module[
];
;

In[]:= QUITARISOMORFOS[listamatrices_]:=Module[
];
;

In[]:= nlados=2;
n=12;
matrizadyacencia={

{0,1,0,0,0,0,0,0,0,0,1},{1,0,1,0,0,0,0,0,0,0,0},

{0,1,0,1,0,0,0,0,0,0,0},{0,0,1,0,1,0,0,0,0,0,0},

{0,0,0,1,0,1,0,0,0,0,0},{0,0,0,0,1,0,1,0,0,0,0},

{0,0,0,0,0,1,0,1,0,0,0},{0,0,0,0,0,0,1,0,1,0,0},

{0,0,0,0,0,0,1,0,1,0,0},{0,0,0,0,0,0,0,1,0,1,0},

{0,0,0,0,0,0,0,1,0,1,0},{1,0,0,0,0,0,0,0,1,0,0}};
```

```

          :
          :

Out]:=      -----> Añadimos nuevo lado a 1 grafos
Van 1
***** DE 1 LADOS: 54 *****
DIVIDIMOS EN 3 CLASES
En la clase 3 hay 3 que no son isomorfos
0.5 . Total: 6
-----> Añadimos nuevo lado a 5 grafos
Van 1
***** DE 2 LADOS: 255 *****
DIVIDIMOS EN 46 CLASES
En la clase 9 hay 4 que no son isomorfos
En la clase 12 hay 2 que no son isomorfos
En la clase 14 hay 3 que no son isomorfos
En la clase 16 hay 5 que no son isomorfos
En la clase 20 hay 3 que no son isomorfos
En la clase 21 hay 3 que no son isomorfos
En la clase 22 hay 2 que no son isomorfos
En la clase 25 hay 4 que no son isomorfos
En la clase 26 hay 3 que no son isomorfos
En la clase 31 hay 2 que no son isomorfos
En la clase 32 hay 5 que no son isomorfos
En la clase 33 hay 5 que no son isomorfos
En la clase 36 hay 2 que no son isomorfos
En la clase 37 hay 2 que no son isomorfos
En la clase 38 hay 2 que no son isomorfos
En la clase 39 hay 2 que no son isomorfos
2.5 . Total: 85
Totales: 85 Tiempo empleado: 2.5

```

En unos 2.5 segundos el programa a calculado los 85 grafos no isomorfos que existen, obsérvese como la rutina también nos informa de que la partición en clases según los invariantes considerados no ha sido suficiente, esto es, los invariantes no constituyen una condición suficiente en este ejemplo. Si subimos de 3 a 5 las potencias de matrices de adyacencia a considerar (esto es, escribimos “potencias=5” en la función QUITARISOMORFOS[]) entonces si tendremos una condición suficiente:

In]:= **QUITARISOMORFOS[listamatrices_]:=Module[**

```

          :
          :
```

potencias=5;

```

          :
          :
```

In]:= **nlados=2;**
n=12;
matrizadyacencia={
{0,1,0,0,0,0,0,0,0,0,1},{1,0,1,0,0,0,0,0,0,0},

```

{0,1,0,1,0,0,0,0,0,0,0},{0,0,1,0,1,0,0,0,0,0,0},
{0,0,0,1,0,1,0,0,0,0,0}, {0,0,0,0,1,0,1,0,0,0,0},
{0,0,0,0,0,1,0,1,0,0,0},{0,0,0,0,0,0,1,0,1,0,0},
{0,0,0,0,0,0,0,1,0,1,0,0},{0,0,0,0,0,0,0,0,1,0,1},
{0,0,0,0,0,0,0,1,0,1,0,1};{1,0,0,0,0,0,0,0,0,1,0};

          :           :

Out[7]=      -----> Añadimos nuevo lado a 1 grafos
Van 1
***** DE 1 LADOS: 54 *****
DIVIDIMOS EN 5 CLASES
0.688 Total: 6
-----> Añadimos nuevo lado a 5 grafos
Van 1
***** DE 2 LADOS: 255 *****
DIVIDIMOS EN 79 CLASES
3.797 Total: 85
Totales: 85 Tiempo empleado: 3.797

```

□

5.4.2. DISPONEMOS DE UNA CONDICIÓN SUFFICIENTE

En ocasiones particulares los invariantes supondrán una condición suficiente para que las matrices de adyacencia representen a grafos isomorfos, en tales casos podemos ganar en eficacia, reprogramamos la función QUITARISOMORFOS[] para esta situación:

| FUNCIÓN | COMENTARIOS |
|---|--|
| QUITARISOMORFOS2[listamatrices_]:=Module[{i,i1,i2}, invariantes={}; posiblesisomorfos={}; Do[| Por entrada la función tendrá una lista de matrices de adyacencia. |
| A1=listamatrices[[i]].listamatrices[[i]]; tabla1=Sort[Table[A1[[i1,i1]],{i1,n}]]; A2=A1.listamatrices[[i]]; tabla2=Sort[Table[A2[[i1,i1]],{i1,n}]]; A3=A1.A1; tabla3=Sort[Table[A3[[i1,i1]],{i1,n}]]; | Invariantes que suponen una condición suficiente para que dos matrices de adyacencia representen a grafos isomorfos. |
| AppendTo[invariantes,{tabla1,tabla2,tabla3}]; ,{i,Length[listamatrices]}]; posiciones=Table[{i},{i,Length[listamatrices]}]; While[posiciones!={}, clase=Position[invariantes,invariantes [[posiciones[[1]][[1]]]]]; AppendTo[posiblesisomorfos,clase]; | Bucle que recorre todas las matrices de adyacencia de "listamatrices" y calcula los invariantes que queramos, en este caso: diagonales principales de las tres primeras potencias de la matriz de adyacencia. Calculamos el número de clases de matrices de grafos isomorfos en que podemos dividir "listamatrices" según los invariantes calculados. |

| | |
|---|---|
| <pre> posiciones=Complement[posiciones,clase];]; Print["DIVIDIMOS EN ",Length[posiblesisomorfos], " CLASES"]; listamat2={}; Do[listamat=Table[listamatrices[[posiblesisomorfos[[j]][[[k1]]]][[1]] ,{k1,Length[posiblesisomorfos[[j]]]}]; AppendTo[listamat2,listamat[[1]]]; ,{j,Length[posiblesisomorfos]}]; listamat2]; </pre> | <p>Cogemos un representante de cada clase.</p> <p>Como salida se da la lista de las matrices de adyacencia no isomorfas según los invariantes propuestos.</p> |
|---|---|

Función 5.19. Grafos isomorfos.

Ejemplo 5.26. Buscamos todos los grafos de 7 vértices distintos salvo isomorfismo. En este caso es suficiente con estudiar las diagonales principales de la potencia segunda, tercera y cuarta de las matrices de adyacencia. Usamos 5.18. pero cambiamos QUITARISOMORFOS[] (5.17.) por QUITARISOMORFOS2[] (5.19.).

```

In]:= Añadirlado[matrizadyacencia_]:=Module[{i,j,matriz},
  :
  :

In]:= QUITARISOMORFOS2[listamatrices_]:=Module[
  :
  :

In]:= n=7;
nlados=(n*(n-1))/2;
matrizadyacencia=Table[0,{i,n},{j,n}];

  :
  :

Out]= -----> Añadimos nuevo lado a 1 grafos
Van 1 **** DE 1 LADOS: 21 *****
DIVIDIMOS EN 1 CLASES
  :
  :

Totales: 1044 Tiempo empleado: 4.953

```

Si comparamos su eficacia respecto al ejemplo 5.23., observamos de nuevo una mejora sustancial.

Si nos quedáramos sólo con la potencia segunda y tercera no tendríamos una condición suficiente. Lo podemos comprobar de dos formas:

- Usamos el mismo método del ejemplo 5.23. pero cambiamos el valor de la variable “potencia” (de la función QUITARISOMORFOS[] (5.17.)) de 3 a 2, entonces comprobaremos dos cosas: que el tiempo empleado es mucho mayor y además que el mismo programa nos indica que sólo con los invariantes indicados no es suficiente.
- También podemos comprobarlo usando el mismo método de este ejemplo pero quitando como invariante la consideración de la diagonal de la cuarta potencia, en tal caso comprobaremos como calcula sólo 859 de los 1044 grafos que sabemos que existen.

□

Ejemplo 5.27. Buscamos todos los grafos de 9 vértices distintos salvo isomorfismo que contengan al eneágono como en el ejemplo 5.24. Definimos las funciones 5.14., 5.19. y usamos el programa 5.18.:.

```
In]:= Añadirlado[matrizadyacencia_]:=Module[{i,j,matriz},

$$\vdots \qquad \vdots$$

In]:= QUITARISOMORFOS2[listamatrices_]:=Module[

$$\vdots \qquad \vdots$$

In]:= n=9;
nlados=2;
matrizadyacencia={{0,1,0,0,0,0,0,0,1},{1,0,1,0,0,0,0,0,0},
{0,1,0,1,0,0,0,0,0},{0,0,1,0,1,0,0,0,0},{0,0,0,1,0,1,0,0,0},
{0,0,0,0,1,0,1,0,0},{0,0,0,0,0,1,0,1,0},{0,0,0,0,0,0,1,0,1},
{1,0,0,0,0,0,1,0,0}};

$$\vdots \qquad \vdots$$

Out]= -----> Añadimos nuevo lado a 1 grafos
Van 1
***** DE 1 LADOS: 27 *****
DIVIDIMOS EN 3 CLASES
0.031 . Total: 4
-----> Añadimos nuevo lado a 3 grafos
Van 1
***** DE 2 LADOS: 75 *****
DIVIDIMOS EN 27 CLASES
0.078 . Total: 31
Totales: 31 Tiempo empleado: 0.078
```

Como se comentó en el ejemplo 5.24., en “listamatrices1” están almacenados los grafos de 9, 10 y 11 vértices y “listamatrices2” contiene los grafos de 11 vértices. El tiempo empleado es considerablemente mejor.

□

Podríamos acelerar más el algoritmo si eliminamos los grafos isomorfos que resultan al añadir un lado a un grafo fijo, para ello reprogramamos la función AñadirLado[] (5.14.):

| FUNCIÓN | COMENTARIOS |
|---|--|
| <pre>AñadirLado2[matrizadyacencia_]:=Module[{i,j,grados,Isomorfas,Isomorfas2,tabla3,tabla4, tabla5,tabla6,A1,A2,A3,A4,A5,A6,n,i1,k1,parte, parte2,posiciones}, listamatrices={}; grados={}; A=matrizadyacencia.matrizadyacencia; Do[Do[If[matrizadyacencia[[j,i]]==0, matriz=matrizadyacencia; matriz[[j,i]]=1; matriz[[i,j]]=1; AppendTo[listamatrices,matriz]; AppendTo[grados,{A[[i,i]],A[[j,j]]}];]; ,{j,i-1}];</pre> | La función tendrá por entrada la matriz de adyacencia de un grafo. |
| <pre>,{i,Dimensions[matrizadyacencia][[1]]}]; posiciones=Table[{i}, {i,Length[grados]}]; Isomorfas={};Isomorfas2={}; While[posiciones!={}, parte=Union[Position[grados,grados [[posiciones[[1,1]]]], Position[grados,{grados[[posiciones[[1,1]],2]], grados[[posiciones[[1,1]],1]]}]; parte2=parte; If[Length[parte2]>1, Do[A1=listamatrices[[parte2[[1,1]]]]; listamatrices[[parte2[[1,1]]]]; A2=listamatrices[[parte2[[k1,1]]]]; listamatrices[[parte2[[k1,1]]]]; A3=A1.listamatrices[[parte2[[1,1]]]]; A4=A2.listamatrices[[parte2[[k1,1]]]]; tabla3=Sort[Table[A3[[i1,i1]],[i1,n]]]; tabla4=Sort[Table[A4[[i1,i1]],[i1,n]]]; A5=A1.A1; A6=A2.A2; tabla5=Sort[Table[A5[[i1,i1]],[i1,n]]]; tabla6=Sort[Table[A6[[i1,i1]],[i1,n]]]; If[tabla3==tabla4 && tabla5==tabla6, AppendTo[Isomorfas2,parte2[[k1]]]];]</pre> | Se añade un lado que no exista previamente al grafo y guarda la información sobre los nuevos vértices adyacentes y sus grados. |
| <pre>,{k1,2,Length[parte2]}]; parte2=Complement[parte2,Union[{parte[[1]]}, Isomorfas2]]; Isomorfas=Union[Isomorfas,Isomorfas2]; Isomorfas2={};</pre> | Introducimos invariantes suficientes que aseguren el isomorfismo |

```

];
posiciones=Complement[posiciones,parte];
];
Isomorfas;
listamatrices=Delete[listamatrices,Isomorfas];
];

```

Función 5.20. Añadir un lado a un grafo.

Ejemplo 5.28. Calcular todos los grafos de 8 vértices y a lo sumo 11 lados usando:

- a) QUITARISOMORFOS[] (5.17.) y Añadirlado[] (5.14.).
 - b) QUITARISOMORFOS2[] (5.19.) y Añadirlado[] (5.14.).
 - c) QUITARISOMORFOS2[] (5.19.) y Añadirlado2[] (5.20.).
 - d) La función ListGraph[] del paquete de Mathematica: <<Combinatoria`.

Compararemos los tiempos empleados e indicaremos en cada caso todas las funciones previas necesarias.

- a) Usamos QUITARISOMORFOS[] y AñadirLado[], además necesitaremos de la función 5.16.

b) Usamos QUITARISOMORFOS2[] y Añadirlado[].

In]:= **Añadirlo[matrizadyacencia_]:=Module[**

```

In]:=          QUITARISOMORFOS2[listamatrices_]:=Module[
 $\vdots$             $\vdots$ 

In]:=          n=8;
nlados=11;
matrizadyacencia=Table[0,{i,n},{j,n}];

 $\vdots$             $\vdots$ 

Out]=
 $\vdots$             $\vdots$ 

Totales: 2481 Tiempo empleado: 99.828

```

- c) Usamos QUITARISOMORFOS2[] y Añadirlado2[].

```

In]:=          Añadirlado2[matrizadyacencia_]:=Module[
 $\vdots$             $\vdots$ 

In]:=          QUITARISOMORFOS2[listamatrices_]:=Module[
 $\vdots$             $\vdots$ 

In]:=          n=8;
nlados=11;
matrizadyacencia=Table[0,{i,n},{j,n}];

 $\vdots$             $\vdots$ 

Out]=
 $\vdots$             $\vdots$ 

Totales: 2481 Tiempo empleado: 78.782

```

- d) Con Listgraph[] en realidad no se puede resolver el ejercicio, vayamos calculando poco a poco los grafos y midiendo los tiempos empleados en el cálculo para grafos de 8 vértices, desde 0 lados en adelante, hasta que la función falle:

```

In]:=          <<Combinatorica`;

In]:=          Timing[ListGraphs[8,0]][[1]]

Out]=          22.656

In]:=          Timing[ListGraphs[8,1]][[1]]

```

```

Out[] = 23.172
In[] := Timing[ListGraphs[8,2]][[1]]

Out[] = 26.012
In[] := Timing[ListGraphs[8,3]][[1]]

Out[] = 34.
In[] := Timing[ListGraphs[8,4]][[1]]

Out[] = 50.908
In[] := Timing[ListGraphs[8,5]][[1]]

Out[] = 87.725
In[] := Timing[ListGraphs[8,6]][[1]]

Out[] = 185.735
In[] := Timing[ListGraphs[8,7]][[1]]

Out[] = 421.172
In[] := Timing[ListGraphs[8,8]][[1]]

Out[] = 1101.5
In[] := Timing[ListGraphs[8,9]][[1]]

Out[] =
No more memory available.
Mathematica kernel has shut down.
Try quitting other applications and then retry.

```

Como vemos a partir de 8 lados la función nos da un error.

□

Estos últimos métodos serán más eficaces respecto a los anteriores cuanto mayor sea el número de vértices que consideremos.

Como podemos observar con una buena condición suficiente el cálculo es bastante más eficaz, si bien el problema reside en disponer y conocer dicha condición suficiente y que además ésta sea fácilmente trasladable al ordenador.

6. EL GRUPO DE AUTOMORFISMOS DE UN GRAFO

A un isomorfismo de un grafo en si mismo se le llama automorfismo. La situación sería la siguiente, tenemos un grafo $G = (W, F)$ con p vértices ($W = \{v_1, \dots, v_p\}$) y una permutación $\sigma = \begin{pmatrix} 1 & \dots & p \\ \sigma(1) & \dots & \sigma(p) \end{pmatrix}$ tal que aplicada a los índices de los vértices da lugar a un automorfismo, esto es, $f_\sigma: W \rightarrow W$ definida por $f_\sigma(v_i) = v_{\sigma(i)}$ es un isomorfismo de grafos. Además, todos los automorfismos de G son de esta forma y vienen dados por una permutación de S_p . El conjunto de todos los automorfismos de un grafo G se denota por $Aut(G)$ y cómo podemos identificar cada automorfismo con una permutación de S_p , concluimos que $Aut(G) \subseteq S_p$. Obsérvese además que la composición de automorfismos vuelve a ser automorfismo, la permutación identidad también es automorfismo trivialmente y para cada permutación que sea un automorfismo, su inversa también lo es, por tanto $Aut(G)$ es un subgrupo de S_p y se le llama grupo de automorfismos de G .

Aprovechando lo aprendido en 5.7. y 5.8., podemos programar el cálculo del grupo de automorfismos de un grafo para grafos no orientados y dígrafos, como son subgrupos del grupo simétrico S_p , aprovecharemos la notación del capítulo 4 y lo que se aprendió sobre subgrupos en el capítulo 3 para comprender mejor la situación:

| FUNCIÓN | COMENTARIOS |
|---|---|
| <pre>AUTOMORFISMOS[W_,F_]:=Module[{Permutaciones,Isomorfismo,automor, CONTADORi,CONTADORj,permutacion}, Permutaciones=Permutations[W]; automor={}; indices={}; Do[permutacion=Permutations[[CONTADORj]]; Isomorfismo=True; Do[If[Length[Intersection] {permutacion[[F][[CONTADORi]][[1]]]]-> permutacion[[F][[CONTADORi]][[2]]]], permutacion[[F][[CONTADORi]][[2]]]]-> permutacion[[F][[CONTADORi]][[1]]]},F]==1 ,Isomorfismo=False]; ,{CONTADORi,Length[F]}]; If[Isomorfismo, AppendTo[automor, MatrixForm[{W,permutación}]]; AppendTo[indices,CONTADORj];]; ,{CONTADORj,Length[Permutaciones]}; Print[automor]; indices];</pre> | Funció n con dos argumentos, el primero será el conjunto de vértices y el segundo el conjunto de lados del grafo (identificados los vértices por sus subíndices). |
| <pre>Do[permutacion=Permutations[[CONTADORj]]; Isomorfismo=True; Do[If[Length[Intersection] {permutacion[[F][[CONTADORi]][[1]]]]-> permutacion[[F][[CONTADORi]][[2]]]], permutacion[[F][[CONTADORi]][[2]]]]-> permutacion[[F][[CONTADORi]][[1]]]},F]==1 ,Isomorfismo=False]; ,{CONTADORi,Length[F]}]; If[Isomorfismo, AppendTo[automor, MatrixForm[{W,permutación}]]; AppendTo[indices,CONTADORj];]; ,{CONTADORj,Length[Permutaciones]}; Print[automor]; indices];</pre> | Comprobamos todas las permutaciones y nos quedamos con aquellas que originan isomorfismos del grafo en si mismo. |
| <pre>Print[automor]; indices];</pre> | Salida de resultados, mostramos las permutaciones con la notación habitual y también identificadas por su índice (véase el capítulo 4 para más detalle). |

Función 5.21. Grupo de automorfismos de un grafo no orientado.

| FUNCIÓN | COMENTARIOS |
|---|--|
| <pre>AUTOMORFISMOS[W_,F_]:=Module[{Permutaciones,Isomorfismo,automor, CONTADORi,CONTADORj,permutacion}, Permutaciones=Permutations[W]; automor={}; indices={};]</pre> | Función con dos argumentos, el primero será el conjunto de vértices y el segundo el conjunto de flechas del grafo (identificados los vértices por sus subíndices). |
| <pre>Do[permutacion=Permutaciones[[CONTADORj]]; Isomorfismo=True; Do[If[Length[Intersection] {permutacion[[F][[CONTADORi]][[1]]]]> permutacion[[F][[CONTADORi]][[2]]],F]==1 ,Isomorfismo=False;]; ,{CONTADORi,Length[F]}]; If[Isomorfismo, AppendTo[automor, MatrixForm[{W,permutación}]]; AppendTo[indices,CONTADORj];]; ,{CONTADORj,Length[Permutaciones]}];</pre> | Comprobamos todas las permutaciones y nos quedamos con aquellas que originan isomorfismos del grafo en si mismo. |
| <pre>Print[automor]; indices];</pre> | Salida de resultados, mostramos las permutaciones con la notación habitual y también identificadas por su índice (véase el capítulo 4 para más detalle). |

Función 5.22. Grupo de automorfismos de un digrafo.

La función `Isomorphism[]` del paquete `<<Combinatorica`` de la versión 6 de Mathematica también permite calcular el grupo de automorfismos de un grafo, si el objeto de tipo grafo asociado es “G”, entonces `Isomorphism[G]` calculará el grupo de automorfismos de “G”.

No debemos confundir el grupo de automorfismos de un grafo G , compuesto por todas las permutaciones que dan lugar a un automorfismo en G , con la clase de isomorfía de matrices de adyacencia de grafos isomorfos a G , que estará formada por todas las matrices de adyacencia que representan a grafos isomorfos y que obtenemos cambiando el nombre de los vértices del grafo de partida, aunque en ambos casos permutemos vértices para conseguirlos. Veámoslo más claro en el siguiente ejemplo:

Ejemplo 5.29. Calculamos el grupo de automorfismos y la clase de isomorfía de matrices de adyacencia de grafos isomorfos al grafo G , siendo G un pentágono y su matriz de adyacencia:

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

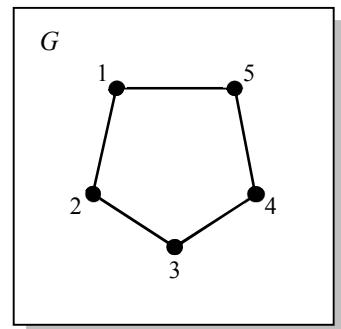


Ilustración 5.19.

Introducimos su matriz de adyacencia en Mathematica:

```
In]:= A={\{0,1,0,0,1},\{1,0,1,0,0},\{0,1,0,1,0},\{0,0,1,0,1},\{1,0,0,1,0}\};
```

Y su clase de isomorfía la calculamos con la función 5.9. Rescribimos su código para mostrar directamente las matrices:

```
In]:= Clase={MatrixForm[A]};  
matricesP=Permutations[IdentityMatrix[  
Dimensions[A][[1]]]];  
Do[  
Clase=Union[Clase,  
{MatrixForm[matricesP[[i]].A.Transpose[matricesP[[i]]]]}];  
,{i,Length[matricesP]}];  
Clase
```

Out[]=

$$\left\{ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \right\}$$

Ahora calculemos su grupo de automorfismos, para ello en primer lugar describimos su conjunto de vértices y de lados:

In[]:= **W={1,2,3,4,5};**
F={1→2,2→3,3→4,4→5,5→1};

Definimos la función 5.21.

In[]:= **AUTOMORFISMOS[W,F]:=Module[**
 ⋮ ⋮
]

Y calculamos el grupo de automorfismos:

In[]:= **AUTOMORFISMOS[W,F]**
Out[]= $\left\{ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 5 & 4 & 3 & 2 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 5 & 4 & 3 \end{pmatrix}, \right.$
 $\left. \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 2 & 1 & 5 & 4 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 1 & 2 \end{pmatrix}, \right.$
 $\left. \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 3 & 2 & 1 & 5 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 5 & 1 & 2 & 3 \end{pmatrix}, \right.$
 $\left. \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 1 & 2 & 3 & 4 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 4 & 3 & 2 & 1 \end{pmatrix} \right\}$

Out[]= {1, 24, 30, 34, 56, 65, 87, 91, 97, 120}

Si consideramos que el grafo es orientado, entonces el resultado será el siguiente:

In[]:= **AUTOMORFISMOSOR[W,F]:=Module[**
 ⋮ ⋮
]

Y calculamos el grupo de automorfismos:

In[]:= **AUTOMORFISMOSOR[W,F]**
Out[]= $\left\{ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 1 & 2 \end{pmatrix}, \right.$
 $\left. \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 5 & 1 & 2 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 1 & 2 & 3 & 4 \end{pmatrix} \right\}$

Out[]= {1, 34, 65, 91, 97}

Si observamos la tabla 3.4., podremos identificarlos como subgrupos de S_5 .

Con la función Isomorfism[], los haríamos así:

```
In]:= <<Combinatorica`  
In]:= Isomorphism[FromAdjacencyMatrix[A]]  
Out]= { {1,2,3,4,5}, {1,5,4,3,2}, {2,1,5,4,3},  
{2,3,4,5,1}, {3,2,1,5,4}, {3,4,5,1,2},  
{4,3,2,1,5}, {4,5,1,2,3}, {5,1,2,3,4},  
{5,4,3,2,1} }
```

y para el grafo orientado:

```
In]:= B={{{0,1,0,0,0},{0,0,1,0,0},{0,0,0,1,0},{0,0,0,0,1},{1,0,0,0,0}};  
Isomorphism[FromAdjacencyMatrix[B,Type->Directed]]  
Out]= { {1,2,3,4,5}, {2,3,4,5,1}, {3,4,5,1,2},  
{4,5,1,2,3}, {5,1,2,3,4} } □
```

Si G_1 es isomorfo a G_2 como grafo entonces es evidente que $Aut(G_1)$ y $Aut(G_2)$ son isomorfos como grupos, de hecho serán dos subgrupos isomorfos (no necesariamente iguales) del mismo grupo simétrico. Por tanto, el grupo de automorfismos constituye un invariante de la clase de isomorfía de matrices de adyacencia de grafos isomorfos que podría ser usada en los métodos del epígrafe 5 de este capítulo (ejercicio 5.27.). Desafortunadamente, de nuevo nos encontramos con una condición necesaria para que dos grafos sean isomorfos (tener grupos de automorfismos isomorfos) pero no suficiente. En el siguiente ejemplo calculamos varios grafos no isomorfos que tienen el mismo grupo de automorfismos.

Ejemplo 5.30. Vamos a calcular todos los grafos no orientados de 6 vértices cuyo grupo de automorfismos sea el subgrupo de S_6 formado por la identidad y la trasposición (5.6).

Lo primero que debemos de tener en cuenta es que según nombremos a los vértices de un grafo, este tendrá por grupo de automorfismo un subgrupo u otro de S_6 , aunque podremos asegurar que serán isomorfos. En el caso que nos ocupa en este ejemplo, podemos observar que si un grafo tienen por grupo de automorfismos a la identidad y una trasposición cualquiera, si renombramos los vértices de forma apropiada, conseguiremos otro grafo isomorfo al de partida pero que tendrá por grupo de automorfismos $\{I, (5\ 6)\}$. Por tanto vamos a calcular las clases de isomorfía de matrices de adyacencia que tengan por grupo de automorfismos, un subgrupo de S_6 de dos elementos formado por la identidad y una trasposición. Dentro de cada una de estas clases existirán grafos cuyo grupo de automorfismos sea el que buscamos. Usamos 5.16., 5.18. y 5.19.:

```
In]:= AñadirLado2[matrizadyacencia_]:=Module[{i,j,matriz},  
⋮ ⋮
```

```

In]:= QUITARISOMORFOS2[listamatrices_]:=Module[

$$\vdots \qquad \vdots$$


In]:= n=6;
nlados=15;
matrizadyacencia=Table[0,{i,n},{j,n}];


$$\vdots \qquad \vdots$$


Out]=

$$\vdots \qquad \vdots$$


Totales: 156 Tiempo empleado: 0.187

```

Ahora buscamos aquellos que tengan como grupo de automorfismos un subgrupo de S_6 con la identidad y una trasposición, después identificamos cuáles deberían ser los vértices 5 y 6, destacándolos con otro color y tamaño, los otros 4 vértices podemos nombrarlos como queramos. Aprovechamos el código del programa 4.12.

```

In]:= sigma[k_,n_][m_]:=Permutations[Table[j,{j,n}]][[k,m]];

In]:= AUTOMORFISMOS[W_,F_]:=Module[

$$\vdots \qquad \vdots$$


In]:= n=6;
Do[
  matrizadyacencia=listamatrices1[[i]];
  W=Table[i,{i,Dimensions[matrizadyacencia][[1]]}];
  F={};
  Do[
    Do[
      If[matrizadyacencia[[i,j]]==1,
        AppendTo[F,i->j]];
      ,{i,j}];
      ,{j,Dimensions[matrizadyacencia][[1]]}];
    automorfismo=Sort[AUTOMORFISMOS[W,F]];
    If[Length[automorfismo]==2,
      k=automorfismo[[2]];
      X=Table[i,{i,n}];
      ciclos={};
      While[X!={},
        primero=X[[1]];
        imagen=primero;
        cadena={primero};
        While[sigma[k,n][imagen]!=primero,

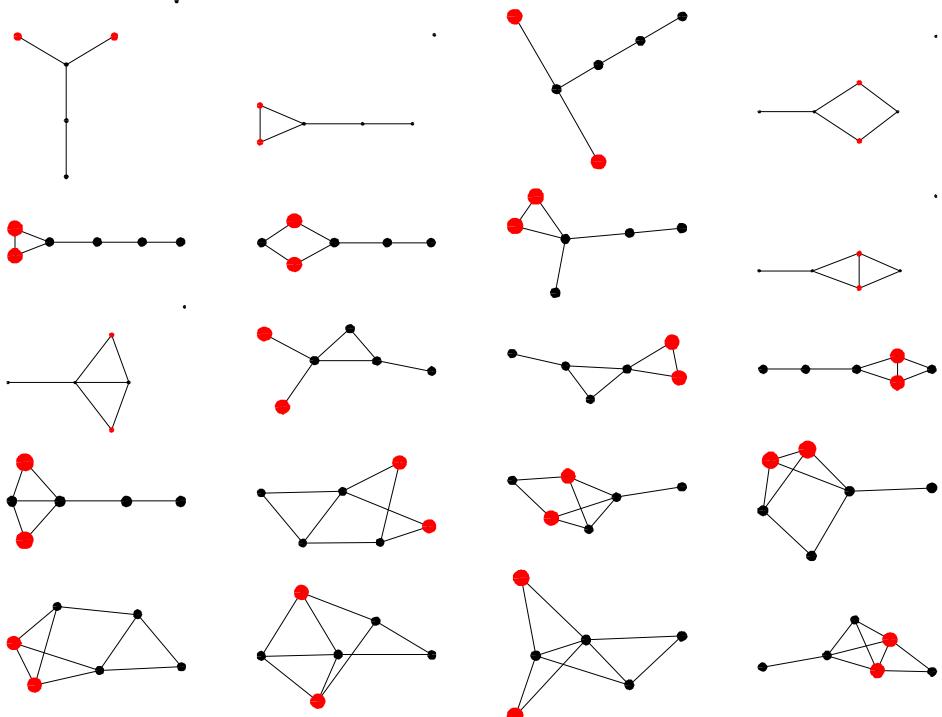
```

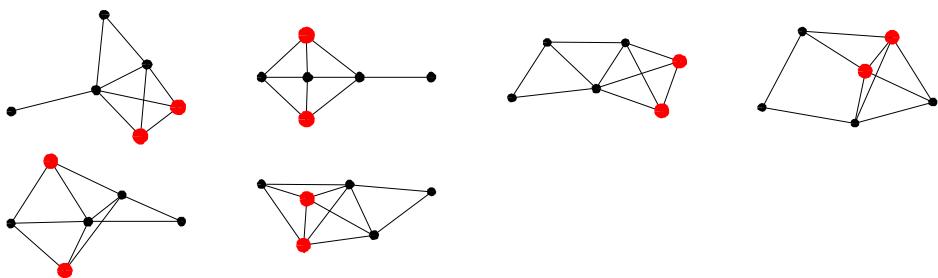
```

imagen=sigma[k,n][imagen];
cadena=Append[cadena,imagen]];
If[Length[cadena]>1,AppendTo[ciclos,cadena]
];
X=Complement[X,cadena];
];
If[Length[ciclos]==1,
colorear={Black,Black,Black,Black,Black,Black};
discos={0.05,0.05,0.05,0.05,0.05,0.05};
colorear[[ciclos[[1]][[1]]]]=Red;
colorear[[ciclos[[1]][[2]]]]=Red;
discos[[ciclos[[1]][[1]]]]=0.08;
discos[[ciclos[[1]][[2]]]]=0.08;
Print[GraphPlot[listamatrices1[[i]],
VertexRenderingFunction→({colorear[[#2]],
Disk[#,discos[[#2]]]}&)];
];
];
,{i,Length[listamatrices1]}

```

Out[] =





Por cada grafo de los representados encontraremos $2! \cdot 4! = 48$ grafos isomorfos de los $6! = 720$ que tiene la clase de isomorfía con el grupo de automorfismos que buscamos, esto es, en total encontraremos 26 grafos no isomorfos y 1248 grafos con matriz de adyacencia distinta y con dicho grupo de automorfismos.

□

A la vista del ejemplo anterior, debemos subrayar que todos los grupos de automorfismos de los grafos con matrices de adyacencia de la misma clase de isomorfía son isomorfos como grupos, por tanto, si queremos hacer un estudio combinatorio bastará con estudiar un único representante por clase y finalmente, si así lo queremos, permutando los índices de los vértices iremos obteniendo todos los grupos de automorfismos.

Ejemplo 5.31. En el siguiente ejemplo vamos a calcular los grupos de automorfismos de todos los grafos no orientados de 5 vértices, estudiamos un representante por clase de isomorfía, para ello construimos todos los grafos de cinco vértices (salvo isomorfismo) usando 5.16., 5.18. y 5.19., cualquier otro grafo será isomorfo como grafo a uno de los analizados y en consecuencia su grupo de automorfismo también será isomorfo pero como grupo.

In[]:= AñadirLado2[matrizadyacencia_]:=Module[{i,j,matriz},

⋮ ⋮

In[]:= QUITARISOMORFOS2[listamatrices_]:=Module[

⋮ ⋮

In[]:= n=6;
nlados=15;
matrizadyacencia=Table[0,{i,n},{j,n}];

⋮ ⋮

Out[]=

⋮ ⋮

Total: 35 Tiempo empleado: 0.016

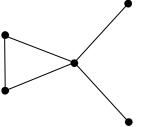
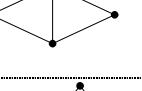
In[]:= AUTOMORFISMOS[W_,F_]:=Module[

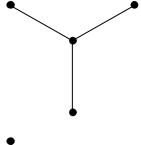
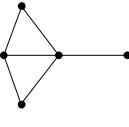
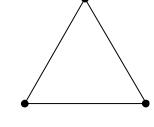
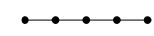
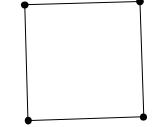
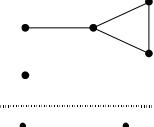
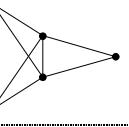
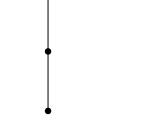
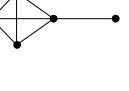
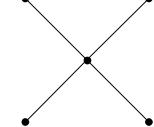
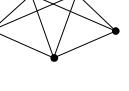
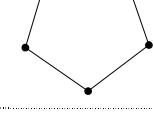
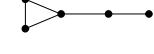
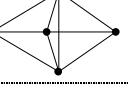
⋮ ⋮

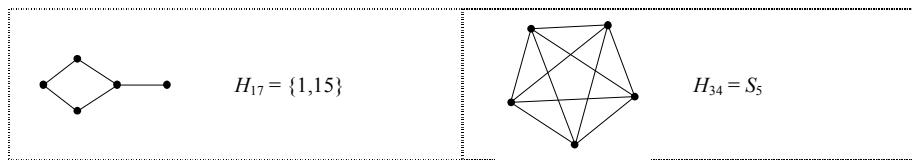
Ahora calculamos el subgrupo de S_5 que es el grupo de automorfismos de cada uno de los representantes de cada clase de isomorfía, si hubiésemos elegido otros representantes, estos subgrupos sería distintos aunque isomorfos:

```
In[7]:= Do[
  matrizadyacencia=listamatrices1[[i]];
  W=Table[i,{i,Dimensions[matrizadyacencia][[1]]}];
  F={};
  Do[Do[
    If[matrizadyacencia[[i,j]]==1,AppendTo[F,i→j]];
    ,{i,j};,{j,Dimensions[matrizadyacencia][[1]]}];
  Print[Hi, " ",AUTOMORFISMOS[W,F]];
  Print[GraphPlot[listamatrices1[[i]],
    PlotStyle→{Black,PointSize[.05]}];
  ,{i,Length[listamatrices1]}]
```

Out[7]=

| | | | |
|---|--|---|---|
|  | $H_1 = S_5$ |  | $H_{18} = \{1, 2, 25, 26\}$ |
|  | $H_2 = \{1, 2, 3, 4, 5, 6, 25, 26, 27, 28, 29, 30\}$ |  | $H_{19} = \{1, 7, 81, 87\}$ |
|  | $H_3 = \{1, 3, 25, 27, 61, 63, 85, 87\}$ |  | $H_{20} = \{1, 56\}$ |
|  | $H_4 = \{1, 2, 7, 8\}$ |  | $H_{21} = \{1, 3, 56, 59, 80, 83, 106, 108\}$ |
|  | $H_5 = \{1, 3, 22, 24\}$ |  | $H_{22} = \{1, 2, 15, 16, 21, 22, 55, 56, 61, 62, 67, 68\}$ |
|  | $H_6 = \{1, 61\}$ |  | $H_{23} = \{1, 27\}$ |
|  | $H_7 = \{1, 2, 7, 8, 25, 26, 31, 32, 49, 50, 55, 56\}$ |  | $H_{24} = \{1, 15\}$ |

| | | | |
|---|--|---|---|
|  | $H_8 = \{1, 3, 7, 9, 13, 15\}$ |  | $H_{25} = \{1, 81\}$ |
|  | $H_9 = \{1, 3, 22, 24, 25, 27, 46, 48, 100, 102, 106, 108\}$ |  | $H_{26} = \{1, 3, 7, 9, 13, 15, 25, 27, 31, 33, 37, 39, 49, 51, 55, 57, 61, 63, 73, 75, 79, 81, 85, 87\}$ |
|  | $H_{10} = \{1, 56\}$ |  | $H_{27} = \{1, 22, 55, 68\}$ |
|  | $H_{11} = \{1, 7, 27, 37, 51, 61, 81, 87\}$ |  | $H_{28} = \{1, 83\}$ |
|  | $H_{12} = \{1, 25\}$ |  | $H_{29} = \{1, 2, 7, 8, 81, 82, 87, 88, 105, 106, 111, 112\}$ |
|  | $H_{13} = \{1, 22\}$ |  | $H_{30} = \{1, 3, 55, 57, 79, 81\}$ |
|  | $H_{14} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24\}$ |  | $H_{31} = \{1, 2, 55, 56, 83, 107, 108\}$ |
|  | $H_{15} = \{1, 24, 30, 34, 56, 65, 87, 91, 97, 120\}$ |  | $H_{32} = \{1, 7, 81, 87\}$ |
|  | $H_{16} = \{1, 106\}$ |  | $H_{33} = \{1, 2, 7, 8, 25, 26, 31, 32, 49, 50, 55, 56\}$ |



Ayudándonos de la tabla 3.4. podemos identificar los distintos subgrupos de S_5 . Si quisieramos realizar un estudio pormenorizado de todos los grafos e identificar exactamente el grupo de automorfismos (subgrupo de S_5), tendríamos que estudiar una por una las 2^{10} matrices de adyacencia posibles. Lo haríamos como sigue, aprovechamos 5.12.,

```
In[]:= n=5;
comb=n*(n-1)/2;
```

```
⋮ ⋮
```

y con 5.3.:

```
In[]:= Do[
matrizadyacencia=listamatrices[[CONTADORi]];
W=Table[i,{i,Dimensions[matrizadyacencia][[1]]}];
F={};
Do[Do[
If[matrizadyacencia[[i,j]]==1,AppendTo[F,i→j]];
,{i,j}];,{j,Dimensions[matrizadyacencia][[1]]}];
Print[MatrixForm[matrizadyacencia]];
Print[AUTOMORFISMOS[W,F]];
,{CONTADORi,Length[listamatrices}}]
```

Out[] =

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120\}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \quad \{1, 2, 7, 8, 25, 26, 31, 32, 49, 50, 55, 56\}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \quad \{1, 6, 15, 17, 25, 30, 39, 41, 75, 77, 81, 83\}$$

⋮

⋮

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix} \quad \{\{1, 2, 7, 8, 25, 26, 31, 32, 49, 50, 55, 56\}$$

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix} \quad \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120\}$$

□

7. EJERCICIOS

Ejercicio 5.1. ¿Cómo debería ser la matriz de incidencia de un digrafo? ¿Y de un multigrafo?

□

Ejercicio 5.2. Diseñar un programa que determine la matriz de incidencia de un grafo a partir de su matriz de adyacencia. Análogamente, construir otro que haga justo lo contrario.

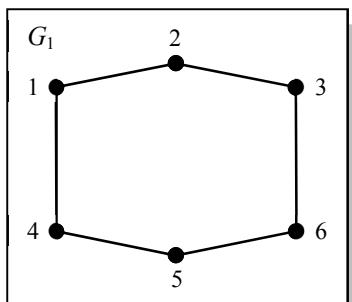
□

Ejercicio 5.3. Determinar todos los grafos dirigidos que existen con 5 vértices salvo isomorfismos.

□

Ejercicio 5.4. Determinar si los siguientes grafos son isomorfos:

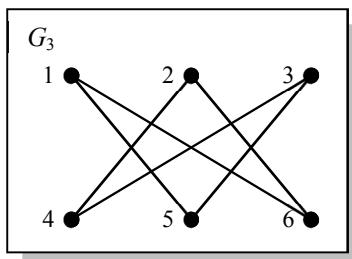
a)

b) G_2 , el grafo con matriz de adyacencia:

$$A_2 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Ilustración 5.20.

c)

d) G_4 , el grafo con matriz de adyacencia:

$$A_4 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Ilustración 5.21.

e) G_5 , el grafo con matriz de incidencia:

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

f) $G_6 = (W, F)$ con $W = \{1, 2, 3, 4, 5, 6\}$ y $F = \{1 \rightarrow 2, 1 \rightarrow 3, 2 \rightarrow 3, 3 \rightarrow 5, 4 \rightarrow 5, 4 \rightarrow 6, 5 \rightarrow 6\}$

□

Ejercicio 5.5. Determinar si los siguientes digrafos son isomorfos:

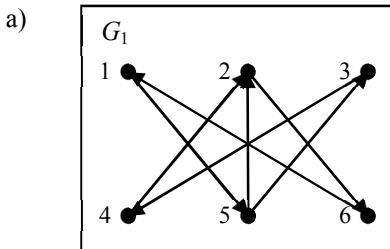


Ilustración 5.22.

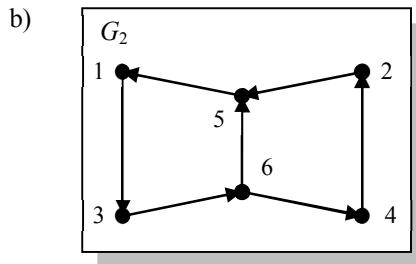


Ilustración 5.23.

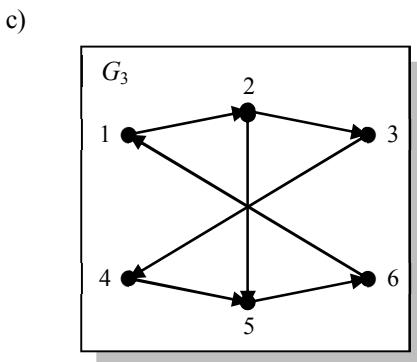


Ilustración 5.24.

d) G_4 , el grafo con matriz de adyacencia:

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

e) $G_6 = (W, F)$ con $W = \{1, 2, 3, 4, 5, 6\}$ y $F = \{1 \rightarrow 4, 2 \rightarrow 3, 6 \rightarrow 1, 3 \rightarrow 5, 4 \rightarrow 6, 5 \rightarrow 6\}$

□

Ejercicio 5.6. Los diagramas de orden son grafos dirigidos sin ciclos orientados. Utilizar las herramientas de grafos para representar diagramas de orden. Dado un conjunto ordenado, calcular su diagrama de orden utilizando las herramientas de representación gráfica que hemos visto. Representar los diagramas de orden de las álgebras de Boole³⁰: $(B_2)^3$ y $(B_2)^4$.

□

Ejercicio 5.7. Las moléculas suelen representarse usando multigrafos no dirigidos donde los átomos son los vértices y los lados son los enlaces. Si nos limitamos a enlaces simples podremos representar distintas moléculas con las herramientas que hemos visto. Los hidrocarburos saturados o alcanos tienen por fórmula general C_nH_{2n+2} y su estructura sería la siguiente:

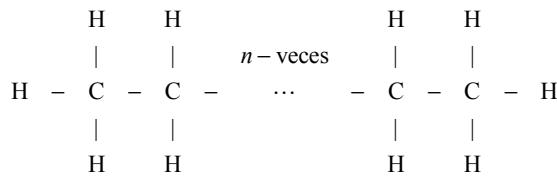
³⁰ En la versión 6 de Mathematica y dentro del paquete de Matemática Discreta: `Combinatorica`, tenemos funciones específicas para la representación de diagramas de orden (o diagramas de Hasse) y en particular de álgebras de Boole:

BooleanAlgebra[n]

que devuelve el objeto de tipo grafo asociado al diagrama de orden del álgebra de Boole $(\mathbb{B}_2)^n$, y

HasseDiagram[G]

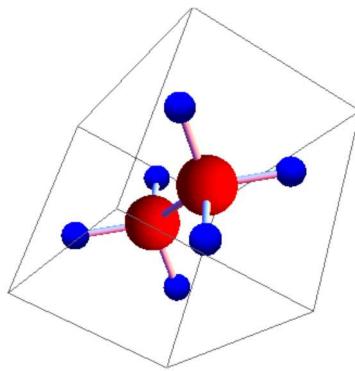
que construye el diagrama de Hasse (o de orden) del grafo dirigido acíclico asociado al objeto de tipo grafo “G”.



Dado un n , determinar la matriz de adyacencia y el grafo necesario para que su representación sea la de una molécula del alcano $\text{C}_n\text{H}_{2n+2}$. Encontrar y representar los grafos no orientados asociados a: el propano ($n = 3$), el butano ($n = 4$) y el octano ($n = 8$). Representar la molécula asignando un color a los átomos de hidrógeno y otro a los de carbono. Para la representación de moléculas es interesante la función `GraphPlot3D[]` que permite colocar los vértices o átomos en el espacio y representar los vértices como esferas y los enlaces como cilindros o la figura que nos interese, por ejemplo, podemos representar el etano:

```
In[7]:=      coordenadas={2->{-8,0,0},5->{8,0,0},1->{-1.3,Cos[0],Sin[0]},  
            3->{-1.3,Cos[2*Pi/3],Sin[2*Pi/3]},4->{-1.3,Cos[4*Pi/3],  
            Sin[4*Pi/3]},6->{1.3,Cos[Pi/3],Sin[Pi/3]},7->{1.3,Cos[Pi],  
            Sin[Pi]},8->{1.3,Cos[5*Pi/3],Sin[5*Pi/3]}};  
tamano={.2,.4,.2,.2,.4,.2,.2};  
colores={Blue,Red,Blue,Blue,Red,Blue,Blue,Blue};  
GraphPlot3D[{1->2,2->3,2->4,2->5,5->6,5->7,5->8},  
EdgeRenderingFunction-(Cylinder[#1,.05]&),  
VertexRenderingFunction-({colores[[#2]],Sphere[#1,  
tamano[[#2]]]}&),VertexCoordinateRules->coordenadas]
```

Out[7]=



Representar el propano, el butano y el octano en el espacio con `GraphPlot3D[]`.

□

Ejercicio 5.8. En la tabla 3.2. de la sección 6. del capítulo 3 encontramos todos los subgrupos del grupo G del ejemplo 3.7. Podemos construir el diagrama de orden (retículo) de los subgrupos con la relación de orden “inclusión”:

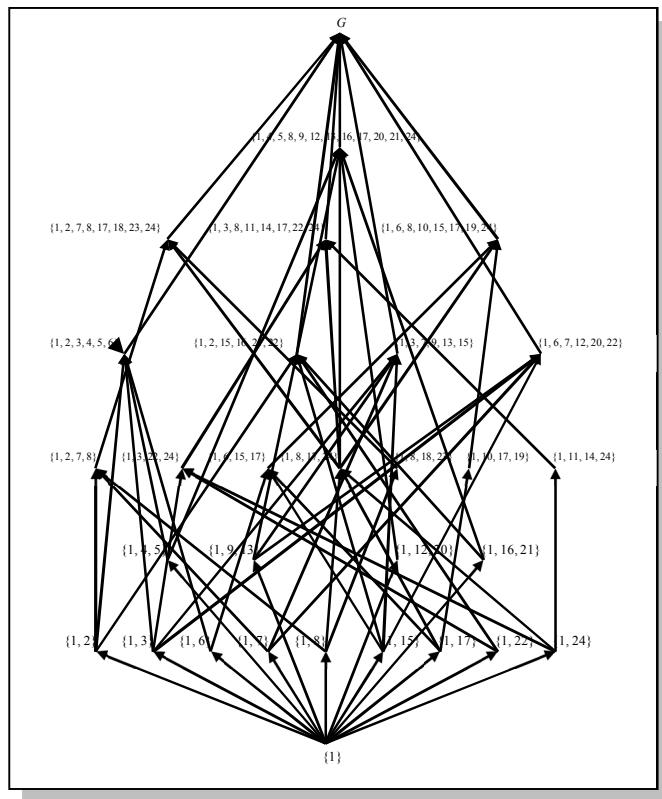


Ilustración 5.25.

Utilizando lo aprendido en el ejercicio 5.5. y el capítulo 3, construir una rutina que represente el diagrama de orden de todos los subgrupos de un grupo finito G .

□

Ejercicio 5.9. En 5.4.2. mejoramos la función Añadirlado[] (5.9.) y se comparan los grafos que tienen más posibilidades de ser isomorfos. Sin embargo la construcción se hace bajo la suposición de que disponemos de invariantes que supongan una condición suficiente para que dos matrices de adyacencia pertenezcan a grafos isomorfos. Programar de forma análoga otra función sin tener en cuenta esta suposición.

□

Ejercicio 5.10. Calcular todos los grafos distintos salvo isomorfismo de 8 vértices y 15 lados, utilizar el método que consideremos más eficaz.

□

Ejercicio 5.11. Calcular todos los grafos distintos salvo isomorfismo de 8 vértices. Determinar propiedades o invariantes de las matrices de adyacencia suficientes para que dos grafos sean isomorfos, utilizarlos para determinar todos los grafos de 8 vértices distintos salvo isomorfismo. Comprobar la eficacia respecto al primer cálculo.

□

Ejercicio 5.12. Calcular la clase de isomorfía, esto es, todas las matrices de adyacencia de grafos isomorfos de las siguientes matrices de adyacencia:

a)

$$A_1 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

b)

$$A_2 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

c)

$$A_3 = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

d)

$$A_4 = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

□

Ejercicio 5.13. Determinar todos los grafos distintos salvo isomorfismo de 8 vértices que contienen al grafo:

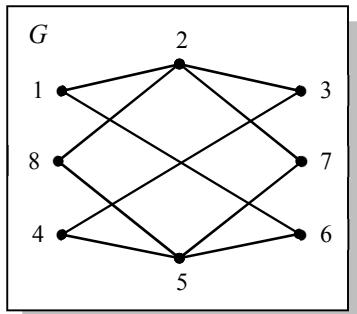


Ilustración 5.26.

□

Ejercicio 5.14. Determinar todos los grafos distintos salvo isomorfismo de 8 vértices que contengan un octógono y que a lo sumo tengan 15 lados.

□

Ejercicio 5.15. Determinar dos grafos no isomorfos tales que las diagonales principales de las potencias 2^a a 4^a de sus matrices de adyacencia coincidan salvo el orden y que además tengan los mismos valores propios.

□

Ejercicio 5.16. Representar gráficamente todos los grafos distintos salvo isomorfismo de 4 vértices.

□

Ejercicio 5.17. ¿Cuántas clases de isomorfía de matrices de adyacencia de grafos de 8 vértices existen? Determinar la matriz de adyacencia de un representante de cada clase de isomorfía de matrices de adyacencia de grafos de 10 lados.

□

Ejercicio 5.18. Calcular todos los grafos distintos salvo isomorfismo de 6 vértices y 10 lados.

□

Ejercicio 5.19. Calcular todas las posibles matrices de adyacencia y de incidencia del siguiente grafo:

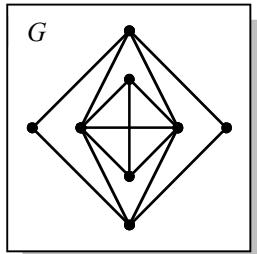


Ilustración 5.27.

□

Ejercicio 5.20. Representar gráficamente los grafos de las ilustraciones 5.19., 5.20., 5.23. y 5.24. con Mathematica. Volver a representarlas pero igual que aparecen dibujadas en dichas ilustraciones. Asignar un nombre a cada vértice y lado, y representarlos mostrando dichos nombres.

□

Ejercicio 5.21. Representar gráficamente, calcular la matriz de adyacencia, calcular la matriz de incidencia y calcular el conjunto de vértices y lados o flechas de cada uno de los grafos de los ejercicios 5.4. y 5.5.

□

Ejercicio 5.22. Hacer un estudio combinatorio en dígrafos:

- Generar todas las matrices de adyacencia para un número fijo de vértices.
- Crear funciones que calculen si dos matrices de adyacencia se corresponden con grafos isomorfos y la clase de isomorfía.

□

Ejercicio 5.23. Calcular los grupos de automorfismos de los grafos del ejercicio 5.5.

□

Ejercicio 5.24. Determinar salvo isomorfismo (de grupos) los grupos de automorfismos de todos los grafos de 6 vértices. □

Ejercicio 5.25. Determinar salvo isomorfismo (de grafos) todos los grafos de 7 vértices cuyo grupo de automorfismos es $\{I, (1\ 2\ 3), (1\ 3\ 2)\}$. □

Ejercicio 5.26. La técnica seguida para construir grafos en el epígrafe 5.3. de este capítulo, ha consistido en añadir progresivamente lados a un grafo dado, esta podría invertirse, es decir, en vez de añadir, quitar lados de un grafo dado hasta llegar al grafo trivial sin lados. Por ejemplo si cogemos K_n y le vamos quitando lados, seremos capaces de construir todos los grafos de n vértices. Programar nuevas rutinas que hagan la construcción como se ha descrito. Razonar las ventajas e inconvenientes que conlleva cambiar la técnica de construcción. □

Ejercicio 5.27. El grupo de automorfismos constituye un invariante de la clase de isomorfía de matrices de adyacencia de grafos isomorfos. Usarlo en los métodos del epígrafe 5 de este capítulo y comprobar su eficacia. □

Ejercicio 5.28. Determinar dos grafos no isomorfos tales que las diagonales principales de las potencias 2^a a 8^a de sus matrices de adyacencia coincidan salvo el orden. □

Ejercicio 5.29.

- a) Sea W el conjunto formado por todas las cifras distintas que componen su DNI junto con todas las letras distintas de su primer apellido.
- b) Consideramos el grafo no orientado G_1 cuyo conjunto de vértices es W y donde dos vértices cualesquiera serán adyacentes si y sólo si son:
 - I. Un número par y una vocal.
 - II. Un número impar y una consonante.
 - III. Dos números pares distintos.
- c) Consideramos el grafo dirigido G_2 cuyo conjunto de vértices es W y donde el conjunto de flechas está formado por todas aquellas flechas que podemos considerar de la forma:
 - I. Origen en un número impar y fin en una vocal.
 - II. Origen en una vocal y fin en una consonante.
 - III. Origen en un número par y fin en un número impar.

Se pide:

- i. Definir G_1 y G_2 directamente por sus conjuntos de vértices y lados o flechas.
 - ii. Calcular las matrices de adyacencia de G_1 y G_2 .
 - iii. Calcular la matriz de incidencia de G_1 .
 - iv. Representar gráficamente G_1 y G_2 .
-

Ejercicio 5.30. Encontrar tres grafos distintos no orientados de al menos 6 vértices y 8 lados, de forma que dos de ellos sean isomorfos y el tercero no lo sea. Calcular sus matrices de adyacencia e incidencia. Representarlos gráficamente.

□

Ejercicio 5.31. Construir si es posible, utilizando el ordenador, todos los grafos no orientados, distintos salvo isomorfismo, que verifiquen a la vez una propiedad de cada bloque.

| | |
|---|---|
| A | Tenga 8 vértices. Tenga 7 vértices. Tenga 6 vértices. Tenga 6 o 7 vértices. Tenga a lo sumo 8 vértices. Tenga a lo sumo 7 vértices. Tenga a lo sumo 6 vértices. Tenga a lo sumo 5 vértices. |
| B | Contenga un triángulo. Contenga un cuadrado. Contenga un triángulo y un cuadrado. Contenga un triángulo o un cuadrado. Contenga un pentágono. Contenga un hexágono Contenga un heptágono. Contenga un polígono regular de tantos lados como vértices tenga el grafo. |
| C | Tenga igual número de lados que de vértices. A lo sumo tenga 10 lados. A lo sumo tenga 12 lados. A lo sumo tenga 14 lados. A lo sumo tenga 16 lados. Como mínimo tenga 8 lados. Como mínimo tenga 7 lados. Como mínimo tenga 6 lados. |

□

Ejercicio 5.32. Modificar la función 5.10. para que muestre todas las permutaciones de los índices de los vértices que dan lugar a isomorfismos, si existen, entre dos grafos con igual número de vértices; igual que la función Isomorphism[] con la opción “All” o como la función ISOMORFOS2[].

□

Ejercicio 5.33. Escribir dos programas: uno que construya un objeto de tipo grafo desde el conjunto de vértices y el de lados o flechas de un grafo dado, y el otro que haga justo lo contrario.

□

6. GRAFOS REGULARES Y COMPLETOS. SUBGRAFOS Y GRAFOS BIPARTITOS

En el capítulo anterior aprendimos a implementar y construir grafos, representarlos gráficamente y distinguir grafos isomorfos. En éste empezamos a estudiar conceptos y propiedades particulares, comenzaremos por definir el grado de un vértice y estudiaremos distintos tipos de grafos: regulares, completos, bipartitos, bipartitos completos. También estudiaremos el concepto de subgrafo, grafo inducido y complemento de un grafo.

1. GRADO DE UN VÉRTICE

1.1. EL GRADO EN GRAFOS NO DIRIGIDOS

Sea $G = (W, F)$ un grafo (no dirigido), llamaremos grado de un vértice $v \in W$ al número de lados incidentes con dicho vértice, lo denotaremos por $gr(v)$. Podremos calcularlo con Mathematica utilizando la siguiente rutina:

| FUNCIÓN | COMENTARIOS |
|--|---|
| <code>GRADO[n_,F_]:=Module[{i},</code> | Función con dos argumentos, el primero será el vértice y el segundo el conjunto de lados. |
| <code>grado=0;</code> <code>Do[</code> <code>If[Intersection[{F[[i]][[1]],F[[i]][[2]]},{n}]=={n}</code> <code>,grado++]</code> <code>,{i,Length[F]}];</code> | Calcula el grado del vértice “n”. |
| <code>grado</code>]; | Salida de resultados. |

Función 6.1. Grado de un vértice de un grafo no dirigido.

También es sencillo determinar el grado de un vértice desde la matriz de adyacencia. Obsérvese que éste viene dado por la suma de todos los unos de la fila o columna correspondiente, esto es, si A es la matriz de adyacencia, el grado de v_i viene dado por la coordenada (i, i) -ésima de A^2 .

| FUNCIÓN | COMENTARIOS |
|--|---|
| GRADO2[n_,matrizadyacencia_]:=MatrixPower[matrizadyacencia,2][[n,n]]; | La función tiene dos entradas, el vértice (identificado por su índice) y la matriz de adyacencia del grafo. |

Función 6.2. Grado de un vértice de un grafo no dirigido.

Análogamente, también podemos determinar el grado de un vértice desde la matriz de incidencia, éste coincide con el número de unos de la fila correspondiente.

| FUNCIÓN | COMENTARIOS |
|--|---|
| GRADO3[n_,matrizincidencia_]:=Sum[matrizincidencia[[n,i]],{i,Dimensions[matrizincidencia]][[2]]}; | La función tiene dos entradas, el vértice (identificado por su índice) y la matriz de incidencia. |

Función 6.3. Grado de un vértice de un grafo no dirigido.

En el paquete `Combinatorica` de la versión 6 de Mathematica, disponemos de la función:

Degrees[G]

que nos devuelve el grado de todos los vértices del grafo asociado al objeto de tipo grafo “G”.

Ejemplo 6.1. Sea G un (p, q) -grafo, es fácil concluir que la suma de los grados de todos los vértices de un grafo ha de ser forzosamente el doble del número de lados:

$$\sum_{i=1}^p gr(v_i) = 2q$$

Definimos la función 6.1.:

In[1]:= GRADO[n_,F_]:=Module[{i},

⋮ ⋮

Esta fórmula respondería a la siguiente igualdad en Mathematica:

Sum[GRADO[i,F],{i,Length[W]}]== Length[F]*2

Lo comprobamos para el grafo del ejemplo 5.4.

*In[2]:= W={1,2,3,4,5};
F= {1→2,1→3,2→3,1→4,2→4,3→4,1→5,2→5,3→5,4→5};*

*In[3]:= Sum[GRADO[i,F],{i,Length[W]}]== Length[F]*2*

Out[3]= True

Análogamente lo podemos comprobar con las funciones 6.2. y 6.3., definimos para ello las funciones 5.1. y 5.5.:

```
In]:= MATRIZADYACENCIA[W_,F_]:=Module[{k,i,j},
```

• • •

In[7]:= MATRIZINCIDENCIA[W,F]:=Module[{i,j},

•
•
•

```
In[]:= GRADO2[n_,matrizadyacencia_]:= MatrixPower[matrizadyacencia,2][[n,n]];
```

```
In[7]:= GRADO3[n_,matrizincidencia_]:= Sum[matrizincidencia[[n,i]] ,{i,Dimensions[matrizincidencia][[2]]}];
```

In[7]:= **Sum[GRADO2[i,MATRIZADYACENCIA[W,F]]
,{i,Length[W]}]==Length[F]*2**

Out[[*] = True*

In[7]:= **Sum[GRADO3[i,MATRIZINCIDENCIA[W,F]]
,{i,Length[W]}]==Length[F]^2**

Out[[*] = True*

Y con la función Degrees[]:

In[7]:= <<Combinatorica`

```
In]:= grados=Degrees[FromAdjacencyMatrix[MATRIZADYACENCIA[W,F]]]
```

```
Out[7]= {4, 4, 4, 4, 4}
```

```
In[7]:= Sum[grados[[i]],[i,Length[grados]]]==Length[F]*2
```

Out[[]]= True

1

1.2. EL GRADO EN GRAFOS DIRIGIDOS

Sea G un grafo dirigido, llamaremos grado de entrada (resp. de salida) de un vértice v al número de flechas cuyo destino (resp. origen) es v , lo denotaremos por $gr^-(v)$ (resp. $gr^+(v)$).

Podemos realizar pequeñas rutinas que identificasen el grado de entrada o salida de cada vértice:

- El grado de entrada en un grafo dirigido:

| FUNCIÓN | COMENTARIOS |
|--|--|
| GRADOENTRADA[n_,F_]:=Module[{i}, | Por entradas tendrá al vértice y el conjunto de flechas del digrafo. |
| grado=0; Do[If[F[[i,2]]==n,grado++],{i,Length[F]}]; | Calcula el grado de entrada del vértice “n”. |
| grado]; | Salida de resultados. |

Función 6.4. Grado de entrada de un vértice de un grafo dirigido.

- El grado de salida en un grafo dirigido:

| FUNCIÓN | COMENTARIOS |
|--|--|
| GRADOSALIDA[n_,F_]:=Module[{i}, | Por entradas tendrá al vértice y el conjunto de flechas del digrafo. |
| grado=0; Do[If[F[[i,1]]==n,grado++],{i,Length[F]}]; | Calcula el grado de salida del vértice “n”. |
| grado]; | Salida de resultados. |

Función 6.5. Grado de salida de un vértice de un grafo dirigido.

Al igual que para grafos no dirigidos, no es difícil encontrar el grado de entrada o salida de un vértice desde la matriz de adyacencia del grafo, como se propone en el ejercicio 6.1.

En el paquete <<Combinatorica` (desde la versión 6 de Mathematica), disponemos de las funciones:

InDegree[G,n] y OutDegree[G,n]

que respectivamente calculan el grado de entrada y el de salida del vértice “n” del grafo asociado al objeto de tipo grafo “G”. Si escribimos: InDegree[G] o OutDegree[G], entonces nos devolverá una lista de todos los grados de salida o de entrada.

Ejemplo 6.2. Sea G un grafo dirigido, un vértice v se dirá que es una fuente si $gr^-(v) = 0$, se dirá que es un sumidero si $gr^+(v) = 0$ y será aislado si $gr^-(v) = 0 = gr^+(v)$. De manera análoga en un grafo no orientado se dirá que un vértice es aislado si $gr(v) = 0$.

- Determinamos si existe algún vértice aislado en el grafo G_2 de la ilustración 5.2.

$$\begin{aligned} In[]:= & \quad W=\{1,2,3,4,5\}; \\ & F=\{1 \rightarrow 2, 2 \rightarrow 3, 3 \rightarrow 4\}; \end{aligned}$$

Utilizamos la función 6.1.:

```
In]:=          GRADO[n_,F_]:=Module[{i},
                                ];
In]:=          Do[If[GRADO[i,F]==0,
                    Print["El vértice ",i," es aislado"],{i,Length[W]}]
Out]=          El vértice 5 es aislado
```

- b) Consideramos el grafo dirigido cuya matriz de adyacencia es:

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

Determinamos, si existen, los sumideros y las fuentes.

```
In]:=          matrizadyacencia={{0,1,1,1,0},{0,0,1,0,0},{0,0,0,0,0},
                           {0,0,1,0,0},{0,1,1,1,0}};
```

Calculamos con el programa 5.4. los conjuntos de vértices y flechas:

```
In]:=          W=Table[i,{i,Dimensions[matrizadyacencia][[1]]}]
F={};
Do[
  Do[
    If[matrizadyacencia[[i,j]]==1,AppendTo[F,i→j]];
    ,{i,Dimensions[matrizadyacencia][[1]]}];
  ,{j,Dimensions[matrizadyacencia][[1]]}];
F

Out]=          {1,2,3,4,5}
{1→2,5→2,1→3,2→3,4→3,5→3,1→4,5→4}
```

Usaremos las funciones 6.4. y 6.5.:

```
In]:=          GRADOENTRADA[n_,F_]:=Module[{i},
                                ];
In]:=          GRADOSALIDA[n_,F_]:=Module[{i},
                                ]
```

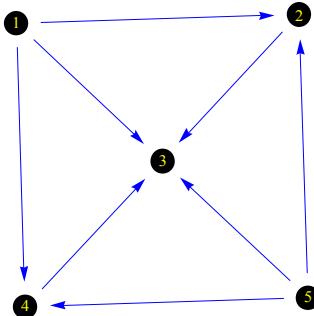
```
In]:= Do[If[GRADOENTRADA[i,F]==0,
          Print["El vértice ",i," es una fuente"]]
    ,{i,Length[W]}]
Do[If[GRADOSALIDA[i,F]==0,
          Print["El vértice ",i," es un sumidero"]]
    ,{i,Length[W]}]

Out]= El vértice 1 es una fuente
      El vértice 5 es una fuente
      El vértice 3 es un sumidero
```

Lo comprobamos gráficamente:

```
In]:= GRAFO=matrizadyacencia;
GraphPlot[GRAFO,
VertexRenderingFunction→({Black,Disk[#,0.05],
Yellow,Text[#2,#1]&}),EdgeRenderingFunction→
({Blue,Arrow[{#1,0.1}]&},
BaseStyle→{FontSize→14})]

Out]=
```



Y con las funciones InDegree[] y OutDegree[]:

```
In]:= <<Combinatorica`

In]:= InDegrees[FromAdjacencyMatrix[
matrizadyacencia,Type->Directed]]

Out]= {0, 2, 4, 2, 0}
```

Por tanto, los vértices 1 y 5 son fuentes.

```
In]:= OutDegrees[FromAdjacencyMatrix[
matrizadyacencia,Type->Directed]]
```

Out[] = {3, 1, 0, 1, 3}

Y el vértice 3 es un sumidero.



2. GRAFOS REGULARES Y GRAFOS COMPLETOS

En este epígrafe sólo estudiaremos grafos no orientados.

2.1. GRAFOS REGULARES

Un grafo se dice que es *k*-regular si todos sus vértices son de grado *k*.

Comprobar con el ordenador si un grafo es regular es inmediato. Podemos hacerlo, por ejemplo, directamente desde la matriz de adyacencia comprobando que todas las filas o columnas tengan el mismo número de “unos” o bien que las coordenadas de la diagonal principal del cuadrado de la matriz de adyacencia sean todas iguales. También es fácil comprobarlo con la matriz de incidencia (ejercicio 6.2).

| FUNCIÓN | COMENTARIOS |
|--|--|
| <code>REGULAR[matrizadyacencia_]:=Module[{i,j},</code> | La función tendrá como entrada a la matriz de adyacencia del grafo. |
| <code>grado=Sum[matrizadyacencia[[1,j]],{j,Dimensions[matrizadyacencia][[1]]}];</code> <code>regular=True;</code> <code>Do[</code> <code>If[grado!=Sum[matrizadyacencia[[i,j]],{j,1,Dimensions[matrizadyacencia][[1]]}],</code> <code>,regular=False;Break[];];</code> <code>,{i,2,Dimensions[matrizadyacencia][[1]]}];</code> <code>regular</code> <code>];</code> | Se comprueba el grado de cada vértice desde la matriz de adyacencia. |
| | Salida de resultados. |

Función 6.6. Grafos regulares.

| FUNCIÓN | COMENTARIOS |
|--|--|
| <code>REGULAR[matrizadyacencia_]:=Module[{i,j},</code> | La función tendrá como entrada a la matriz de adyacencia del grafo. |
| <code>A=MatrixPower[A,2];</code> <code>grado=A[[1,1]];</code> <code>regular=True;</code> <code>Do[</code> <code>If[grado!=A[[i,i]]]</code> <code>,regular=False;Break[];];</code> <code>,{i,2,Dimensions[matrizadyacencia][[1]]}];</code> <code>regular</code> <code>];</code> | Se comprueba el grado de cada vértice desde la matriz de adyacencia. |
| | Salida de resultados. |

Función 6.6.bis. Grafos regulares.

O usando los conjuntos W y F y la función GRADO[] (6.1.).

| FUNCIÓN | COMENTARIOS |
|---|--|
| GRADO[n_,F_]:=... | Definimos la función 6.1. |
| REGULAR2[W_,F_]:=Module[{i,j}, $\text{grado}=\text{GRADO}[W[[1]],F];$ $\text{regular}=\text{True};$ Do[$\text{If}[\text{grado}\neq\text{GRADO}[W[[i]],F],$ $\text{regular}=\text{False};\text{Break}[],];$ $,\{i,2,\text{Length}[W]\}];$ $\text{If}[\text{regular},\text{Print}["El grafo es ",\text{grado},",-regular"],$ $\text{Print}["El grafo no es regular"]];$]; | .La función tendrá dos argumentos: el conjunto de vértices y el conjunto de lados. Se comprueba el grado de cada vértice Salida de resultados. |
| | Función 6.7. Grafos regulares. |

En el paquete `Combinatorica` de la versión 6 de Mathematica, disponemos de las funciones:

RegularQ[G] y **RegularGraph[k,n]**,

la primera comprueba si el grafo asociado al objeto de tipo grafo “G” es regular y la segunda construye un grafo “k”-regular de “n” vértices.

Ejemplo 6.3. Determinamos si los siguientes grafos son regulares y los representamos gráficamente:

1) $G_1 = (W_1, F_1)$ con $\begin{cases} W_1 = \{1,2,3,4,5\} \\ F_1 = \{\{1,2\}, \{1,3\}, \{1,5\}, \{2,3\}, \{2,4\}, \{3,4\}, \{3,5\}, \{4,5\}\} \end{cases}$.

2) G_2 el grafo cuya matriz de adyacencia es:

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

- 1) Definimos las funciones 6.1 y 6.7.:

In[]:= **GRADO[n_,F_]:=Module[{i},**

⋮ ⋮

In[]:= **REGULAR2[W_,F_]:=Module[{i,j},**

⋮ ⋮

Comprobamos si G_1 es regular:

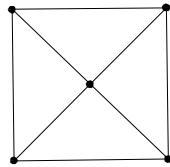
In[]:= **W1={1,2,3,4,5};**
F1={1→2,1→3,1→5,2→3,2→4,3→4,3→5,4→5};
REGULAR2[W1,F1]

Out[]:= El grafo no es regular

Lo vemos gráficamente:

In[]:= **GraphPlot[F, PlotStyle→{Black,PointSize[.05]}]**

Out[]:=



2) Analizamos G_2 usando la función 6.6.

In[]:= **REGULAR[matrizadyacencia]:=Module[{i,j},**

⋮ ⋮

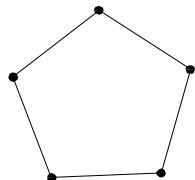
In[]:= **matrizadyacencia={{0,1,0,0,1},{1,0,1,0,0},{0,1,0,1,0},**
{0,0,1,0,1},{1,0,0,1,0}};
REGULAR[matrizadyacencia]

Out[]:= True

Lo dibujamos:

In[]:= **GraphPlot[matrizadyacencia,**
PlotStyle→{Black,PointSize[.05]}]

Out[]:=



Y con la función RegularQ[]:

```
In]:= <<Combinatorica`  

In]:= RegularQ[FromAdjacencyMatrix[  

matrizadyacencia]]  

Out]= True
```

□

Ejemplo 6.4. Representamos gráficamente todos los grafos 4-regulares de 7 vértices distintos salvo isomorfismo.

Al ser 4-regular el número de lados $q = (\sum_{i=1}^p gr(v_i)) / 2 = 14$. Usando las funciones

5.19., 5.20. (consideraremos las diagonales principales de las potencias segunda, tercera y cuarta para comprobar si dos matrices de adyacencia pertenecen a la misma clase de isomorfía) y el programa 5.18. calculamos todos los grafos de 7 vértices y 14 lados.

```
In]:= Añadirlado2[matrizadyacencia_]:=Module[{i,j,matriz},  

: : :  

In]:= QUITARISOMORFOS2[listamatrices_]:=Module[  

: : :  

In]:= n=7;  

nlados=14;  

matrizadyacencia=Table[0,{i,n},{j,n}];  

: : :  

Out]=  

: : :  

DIVIDIMOS EN 97 CLASES  

25.407 . Total: 898  

-----> Añadimos nuevo lado a 97 grafos  

Van 1  

***** DE 14 LADOS: 591 *****  

DIVIDIMOS EN 65 CLASES  

27.344 . Total: 963  

Totales: 963 Tiempo empleado: 5.344
```

Obtenemos 963, que es la longitud de la lista “listamatrices2”, donde están los grafos de 7 vértices y 14 lados, comprobamos cuáles de ellos son regulares con la función 6.6. y los representamos:

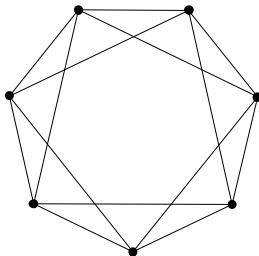
In[]:= **REGULAR[matrizadyacencia_]:=Module[{i,j},**

⋮ ⋮

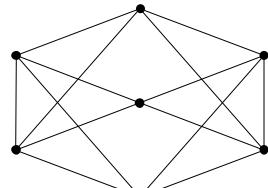
In[]:= **Do[**
If[REGULAR[listamatrices2[[i]]],
Print[GraphPlot[listamatrices2[[i]],
PlotStyle→{Black,PointSize[.04]}]];
];
 $\{i, \text{Length}[\text{listamatrices2}]\}$

Out[]=

62



63



□

2.2. GRAFOS COMPLETOS

Un (p, q) -grafo no orientado se dirá que es completo si cada vértice es adyacente con todos los demás, esto es, el grado de cada vértice es $p - 1$ o lo que es lo mismo es $(p - 1)$ -regular, lo denotaremos por K_p .

Podemos comprobar si un grafo es completo con Mathematica:

| FUNCIÓN | COMENTARIOS |
|---|---|
| COMPLETO[matrizadyacencia_]:= $(\text{Sum}[\text{MatrixPower}[\text{matrizadyacencia}, 2][[i, i]], \{i, \text{Dimensions}[\text{matrizadyacencia}][[1]]\}] == \text{Dimensions}[\text{matrizadyacencia}][[1]]^* (\text{Dimensions}[\text{matrizadyacencia}][[1]] - 1))$ | La función tendrá la matriz de adyacencia del grafo por entrada. |
| | Comprobamos si es completo calculando los grados de todos los vértices. |

Función 6.8. Grafos completos.

La matriz de adyacencia de un grafo completo está compuesta sólo por unos excepto la diagonal principal que tiene ceros, es fácil de definir con Mathematica:

In[]:= **n=NUMERO DE VÉRTICES;**
matrizadyacencia=Table[If[i==j,0,1},{i,n},{j,n}];

También podemos dar directamente los conjuntos de vértices y lados W y F , con Mathematica:

```
In[7]:= n=NUMERO DE VÉRTICES;
W=Table[i,{i,n}];
F={};Do[Do[AppendTo[F,i→j],{j,i+1,n}];,{i,n-1}];
```

Por otra parte comprobar si $G = (W, F)$ un (p, q) -grafo es o no completo resulta muy sencillo pues el número de lados que debe de tener es obligatoriamente $\frac{p(p-1)}{2}$:

$$\text{Length}[F] == \text{Length}[W] * (\text{Length}[W]-1)/2$$

En el paquete `Combinatorica` (versión 6 o posterior de Mathematica), disponemos de las funciones:

CompleteQ[G] y **CompleteGraph[n]**,

la primera comprueba si el grafo asociado al objeto de tipo grafo “G” es completo y la segunda construye el grafo completo de “n” vértices.

Ejemplo 6.5. Comprobar si alguno de los siguientes grafos es completo:

- a) $G_1 = (W_1, F_1)$ con $W_1 = \{1, 2, 3, 4, 5, 6\}$ y $F_1 = \{1 \rightarrow 2, 2 \rightarrow 3, 3 \rightarrow 4, 4 \rightarrow 5, 5 \rightarrow 6, 6 \rightarrow 1, 2 \rightarrow 4, 2 \rightarrow 5, 4 \rightarrow 6, 2 \rightarrow 6, 3 \rightarrow 6, 3 \rightarrow 5\}$.
- b) $G_2 = (W_2, F_2)$ con $W_2 = \{1, 2, 3, 4\}$ y $F_2 = \{1 \rightarrow 2, 2 \rightarrow 3, 3 \rightarrow 4, 4 \rightarrow 1, 2 \rightarrow 4, 3 \rightarrow 1\}$.
- c) G_3 , con matriz de adyacencia:

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

- d) G_4 , con matriz de incidencia:

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

Introducimos todos los grafos.

```
In]:=          W1={1,2,3,4,5,6};
F1={1→2,2→3,3→4,4→5,5→6,6→1,2→4,2→5,4→6,2→6,
     3→6,3→5};
W2={1,2,3,4};
F2={1→2,2→3,3→4,4→1,2→4,3→1};
matrizadyacencia3={{0,1,1,1,1},{1,0,1,1,1},{1,1,0,1,1},
{1,1,1,0,1},{1,1,1,0,0}};
matrizincidencia4={{1,0,0,0,0,1,0,0,0,1,0,0,1,0,1},
{1,1,0,0,0,0,1,0,0,0,1,0,0,1,0,0},{0,1,1,0,0,1,0,1,0,0,0,1,0,0,0,0},
{0,0,1,1,0,0,1,0,1,0,0,0,1,0,0,0},{0,0,0,1,1,0,0,1,0,1,0,0,0,1,0,0},
{0,0,0,0,1,0,0,0,1,0,1,1,0,0,1}};
```

a) y b) podemos comprobarlo rápidamente contando el número de vértices y lados:

```
In]:= Length[F1]==Length[W1]*(Length[W1]-1)/2
```

```
Out]= False
```

```
In]:= Length[F2]==Length[W2]*(Length[W2]-1)/2
```

```
Out]= True
```

También podríamos determinar si son regulares, usamos la función 6.7. (necesita de la función 6.1.):

```
In]:= GRADO[n_,F_]:=Module[{i},
```

```
⋮ ⋮
```

```
In]:= REGULAR2[W_,F_]:=Module[{i,j},
```

```
⋮ ⋮
```

```
In]:= REGULAR2[W1,F1]
```

```
Out]= El grafo no es regular
```

```
In]:= REGULAR2[W2,F2]
```

```
Out]= El grafo es 3-regular
```

```
In]:= grado==Length[W2]-1
```

```
Out]= True
```

c) podemos comprobarlo con la función 6.8.

In[]:= **COMPLETO[matrizadyacencia_]:=**

⋮ ⋮

In[]:= **COMPLETO[matrizadyacencia3]**

Out[]= True

o con CompleteQ[]:

In[]:= <<Combinatorica`

In[]:= **CompleteQ[FromAdjacencyMatrix[**
Matrizadyacencia3]**]]**

Out[]= True

También podríamos comprobarlo determinando si es 4-regular con la función 6.6.:

In[]:= **REGULAR[matrizadyacencia_]:=Module[{i,j},**

⋮ ⋮

In[]:= **REGULAR[matrizadyacencia3]**

Out[]= El grafo es 4-regular

Por último d) podemos comprobarlo de varias formas por ejemplo, contando el número de vértices y lados:

In[]:= **p=Dimensions[matrizincidencia4][[1]];**
q=Dimensions[matrizincidencia4][[2]];

In[]:= **q==p*(p-1)/2;**

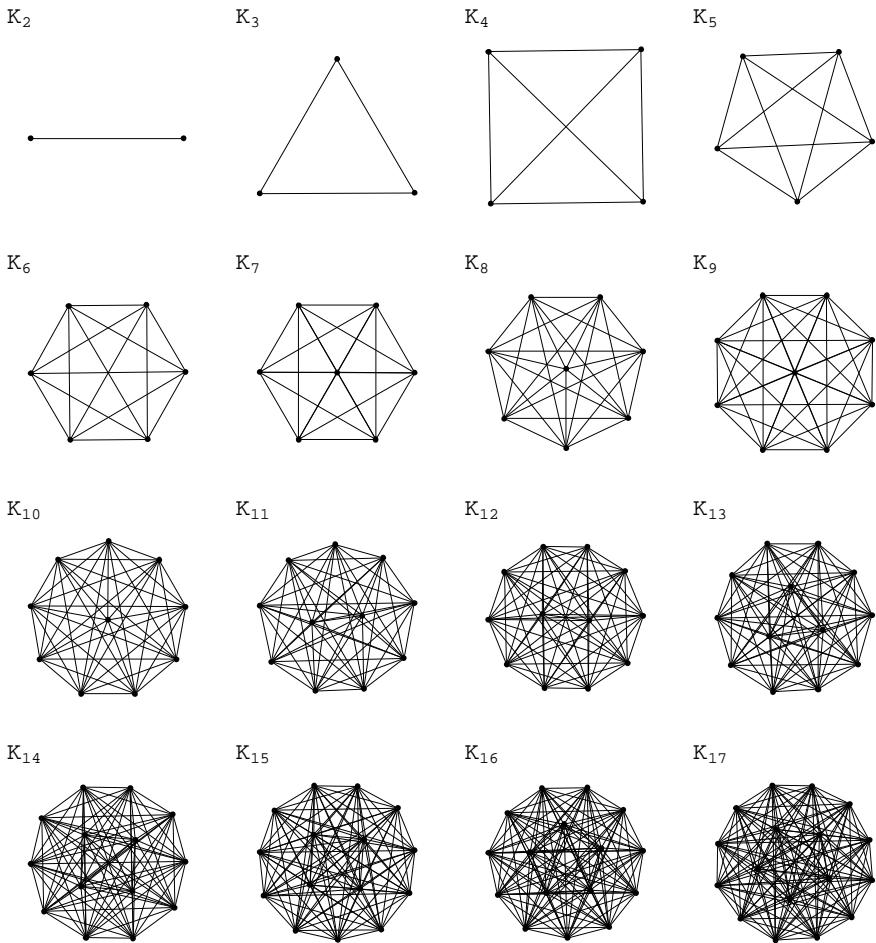
Out[]= True

□

Ejemplo 6.6. Representamos gráficamente los grafos completos de 2 a 17 vértices.

In[]:= **Do[**
Print["K"_n];
Print[GraphPlot[Table[If[i== j, 0, 1], {i, n}, {j, n}],
PlotStyle→{Black,PointSize[.05]}]];
,{n, 2, 17}]

Out[]=



Y con CompleteGraph[]:

In[1]:= <<Combinatorica`

In[2]:= ShowGraphArray[Table[CompleteGraph[i], {i, 2, 17}]];

Obtendremos una salida similar que omitimos. □

3. SUBGRAFOS Y GRAFOS BIPARTITOS

3.1. SUBGRAFOS

Dado un grafo no orientado (resp. grafo dirigido) $G = (W, F)$, diremos que otro grafo no orientado (resp. grafo dirigido) $G_1 = (W_1, F_1)$ es un subgrafo de G si $\emptyset \neq W_1 \subseteq W$ y $F_1 \subseteq F$, donde cada lado (resp. flecha) de F_1 es incidente con vértices de W_1 . Si un subgrafo es tal que $W_1 = W$ se dirá que es un subgrafo maximal.

Comprobar si un grafo (dirigido o no) es subgrafo de otro o subgrafo maximal con Mathematica es inmediato:

```
Length[Intersection[W,W1]]==Length[W1] &&
Length[Intersection[F,F1]]==Length[F1]
```

Y subgrafo maximal:

```
Sort[W]==Sort[W1] && Length[Intersection[F,F1]]==Length[F1]
```

Podemos definir las operaciones de unión e intersección en el conjunto de todos los subgrafos de un grafo dado; para $G_1 = (W_1, F_1)$, $G_2 = (W_2, F_2)$, dos subgrafos de G :

$$G_1 \cup G_2 = (W_1 \cup W_2, F_1 \cup F_2) \quad \text{y} \quad G_1 \cap G_2 = (W_1 \cap W_2, F_1 \cap F_2)$$

Con Mathematica bastará con usar las funciones `Union[]` e `Intersection[]`:

```
In]:= Union[W1,W2];
Union[F1,F2];
```

```
In]:= Intersection[W1,W2];
Intersection[F1,F2];
```

Para más detalle sobre estas funciones véase el capítulo 7 de [25], donde también se analiza la diferencia lógica entre conjuntos. También es aconsejable revisar el epígrafe 5.3. del capítulo 5 y relacionar lo allí descrito con los conceptos que acabamos de introducir.

En el paquete `<<Combinatorica`` (versión 6 de Mathematica o posterior), disponemos de funciones para realizar éstas y otras operaciones entre grafos: `GraphUnion[G1,G2,...]`, `GraphIntersection[G1,G2,...]`, `GraphJoin[G1,G2,...]`, `GraphSum[G1,G2,...]`, `GraphProduct[G1,G2,...]` y `GraphDifference[G1,G2]`; todas ellas aplicables sobre objetos de tipo grafo. Dejamos para el lector interesado el estudio de su funcionamiento en detalle (ejercicio 6.21).

3.2. SUBGRAFOS INDUCIDOS

Sea $G = (W, F)$ un grafo no orientado o grafo dirigido y sea $W' \subseteq W$, llamaremos subgrafo de G inducido por W' y lo denotaremos por $\langle W' \rangle$ al subgrafo de G que tiene por conjunto de vértices a W' y por conjunto de lados o flechas al subconjunto de F formado por todos los lados o flechas de F incidentes con vértices de W' . Con Mathematica podemos obtenerlo con la siguiente función:

| FUNCIÓN | COMENTARIOS |
|--|---|
| <code>IND[W1_,F_]:=Module[{i},</code> | Las entradas serán: el subconjunto de vértices y el conjunto de lados o flechas. |
| <code> F1={}; Do[If[Sort[Intersection[{F[[i]][[1]],F[[i]][[2]]},W1]== Sort[{F[[i]][[1]],F[[i]][[2]]}],AppendTo[F1,F[[i]]]], ,{i,Length[F]}]; F1]</code> | Se comprueban los qué lados o flechas de “F” son incidentes con vértices de “W1”. |
| | Salida de resultados. |

Función 6.9. Subgrafos inducidos.

| FUNCIÓN | COMENTARIOS |
|--|--|
| <code>IND2[W1_,matrizadyacencia_]:=Module[{n,i,j},</code> | Las entradas serán: el subconjunto de vértices (identificados por sus subíndices) y la matriz de adyacencia. |
| <code>n=Length[W1]; matrizadyacencia2=Table[0,{i,n},{j,n}]; Do[Do[matrizadyacencia2[[i,j]]= matrizadyacencia[[W1[[i]],W1[[j]]]]; ,{i,n}]; ,{j,n}]; matrizadyacencia2];</code> | Se comprueban los qué lados o flechas son incidentes con vértices de “W1”. |
| | Salida de resultados. |

Función 6.10. Subgrafos inducidos.

Ambos programas funcionan de forma similar, el primero hace los cálculos desde el conjunto de lados o flechas y el segundo desde la matriz de adyacencia.

En el paquete `<<Combinatorica`` (versión 6 y siguientes de Mathematica), disponemos de la función:

InduceSubgraph[G,S]

que calcula el subgrafo inducido del grafo asociado al objeto de tipo grafo “G” para el subconjunto de vértices “S” (dados por sus índices).

Ejemplo 6.7. Calculamos usando las funciones 6.9., 6.10. y `InduceSubGraph[]`:

- a) El subgrafo inducido de K_5 por $\{2, 3, 4\}$.

- b) El subgrafo inducido por $\{2, 3, 4\}$ del grafo dirigido del apartado b) del ejemplo 6.2.
- a) Definimos la función 6.9.:

```
In]:=          IND[W1_,F_]:=Module[{i},
                                ];
                                
```

Y la aplicamos:

```
In]:=          n=5;
W=Table[i,{i,n}];F={};
Do[Do[AppendTo[F,i->j],{j,i+1,n}];,{i,n-1}];
IND[{2,3,4},F]
```

```
Out]= {2->3, 2->4, 3->4}
```

Usando la matriz de adyacencia con la función 6.10.:

```
In]:=          IND2[W1_,matrizadyacencia_]:=Module[{n,i,j},
                                ];
                                
```

```
In]:=          IND2[{2,3,4}, Table[If[i==j,0,1],{i,n},{j,n}]]
```

```
Out]= {{0,1,1},{1,0,1},{1,1,0}}
```

Y con InduceSubGraph[]:

```
In]:=          <<Combinatorica`
```

```
In]:=          InduceSubgraph[CompleteGraph[5],{2,3,4}]
```

```
Out]= -<3,3,Undirected>-
```

- b) Tomamos la matriz de adyacencia del ejemplo 6.2. y le aplicamos la función 6.10.:

```
In]:=          matrizadyacencia={{0,1,1,1,0},{0,0,1,0,0},{0,0,0,0,0},
{0,0,1,0,0},{0,1,1,1,0}};
```

```
In]:=          IND2[{2,3,4}, matrizadyacencia]
```

```
Out]= {{0,1,0},{0,0,0},{0,1,0}}
```

Ahora calculamos los conjuntos de vértices y flechas y le aplicamos 6.9.:

```
In]:=          W=Table[i,{i,Dimensions[matrizadyacencia][[1]]}];
```

```

F={};
Do[
 Do[
 If[matrizadyacencia[[i,j]]==1,AppendTo[F,i->j]];
 ,{i,Dimensions[matrizadyacencia]][[1]]];
 ,{j,Dimensions[matrizadyacencia]][[1]]];
 F

Out[]=
 {1->2, 5->2, 1->3, 2->3, 4->3, 5->3, 1->4, 5->4}

In[]:= IND[{2,3,4},F]

Out[]=
 {2->3, 4->3}

```

□

Sean $G_1 = (W_1, F_1)$ y $G_2 = (W_2, F_2)$ dos subgrafos (resp. subgrafos dirigidos) del grafo G , entonces denotaremos por $G_1 - G_2$ al subgrafo $G_3 = (W_3, F_3)$ de G cuyo conjunto de vértices es $W_3 = W_1 - W_2$ y cuyo conjunto de lados (resp. de flechas) es $F_3 \subseteq F_1 - F_2$, el conjunto formado por todos los lados (resp. flechas) de $F_1 - F_2$ incidentes con vértices de $W_1 - W_2$. Con Mathematica:

```

In[]:= W3=Complement[W1,W2];
F3=IND[W3,Complement[F1,F2]];

```

3.3. GRAFOS BIPARTITOS Y GRAFOS BIPARTITOS COMPLETOS

Un grafo $G = (W, F)$ (no dirigido) se dirá bipartito si existen

$$W_1 \neq \emptyset \text{ y } W_2 \subseteq W$$

dos subconjuntos de vértices disjuntos tales que:

- i. $W = W_1 \cup W_2$;
- ii. $\langle W_1 \rangle = G_1 = (W_1, \emptyset)$;
- iii. $\langle W_2 \rangle = G_2 = (W_2, \emptyset)$;

esto es, todos los lados son incidentes con un vértice de W_1 y otro de W_2 . Si además cada vértice de W_1 es adyacente a todos los vértices de W_2 , entonces se dirá que es bipartito completo y lo denotaremos por $K_{n,m}$ donde $|W_1| = n$ y $|W_2| = m$. Observemos que el número de lados de un grafo bipartito completo es nm y además $gr(w_1) = m$ y $gr(w_2) = n$ para cada $w_1 \in W_1$ y $w_2 \in W_2$.

Para determinar si grafo es bipartito utilizaremos la función $IND[]$ (6.9.) que hemos definido anteriormente y la función $KSubsets[]$ del paquete de Matemática Discreta: `Combinatorica'. Analizaremos parejas de subconjuntos de vértices tales que verifiquen las propiedades que definen a un grafo bipartito, esto es, que sean partición del conjunto de

vértices e induzcan grafos sin lados. Finalmente, si es bipartito, comprobando el número de lados podemos determinar si es completo o no, porque el número de lados de un bipartito completo es obligatoriamente el producto del número de vértices que tengan los subconjuntos W_1 y W_2 .

| FUNCIÓN | COMENTARIOS |
|---|---|
| <code><<Combinatoria`;</code> | Introducimos este paquete de funciones para utilizar la función KSubsets[]. |
| <code>IND[W1_,F_]:=...</code> | Introducimos la función 6.9. |
| <pre>BIPARTITO[W_,F_]:=Module[{i,subconjuntos,j,fin}, bipartito=False;i=1; fin=Quotient[Length[W],2]; While[!bipartito && i<=fin, subconjuntos=KSubsets[W,i]; j=1; While[!bipartito && j<=Length[subconjuntos], If[IND[subconjuntos[[j]],F]=={}, If[IND[Complement[W,subconjuntos[[j]]],F] =={}, bipartito=True; W1=subconjuntos[[j]]; W2=Complement[W,subconjuntos[[j]]];]; j++;]; i++];]; If[bipartito, Print["Es bipartito: W1=",W1," y W2=",W2]; If[Length[F] == Length[W1]*Length[W2], Print["Es bipartito completo: ", "K" Length[W1],Length[W2]], Print["No es bipartito completo"]]; ,Print["No es bipartito"]];]</pre> | <p>Se comprueba si es bipartito.</p> <p>Salida de resultados.</p> |

Función 6.11. Grafos bipartitos y bipartitos completos.

Ejemplo 6.8. Comprobamos si son bipartitos K_6 y el grafo

$$G = (W, F) \text{ con } \begin{cases} W = \{1, 2, 3, 4, 5, 6\} \\ F = \{\{1, 4\}, \{1, 5\}, \{1, 6\}, \{2, 4\}, \{2, 5\}, \{2, 6\}, \{3, 4\}, \{3, 5\}, \{3, 6\}\} \end{cases}$$

Para K_6 , introducimos el grafo:

```
In[]:=      n=6;
           W=Table[i,{i,n}]
           F={};
           Do[Do[AppendTo[F,i->j],{j,i+1,n}],{i,n-1}];
           F
```

```
Out[]= {1, 2, 3, 4, 5, 6}
```

$$\{1 \rightarrow 2, 1 \rightarrow 3, 1 \rightarrow 4, 1 \rightarrow 5, 1 \rightarrow 6, 2 \rightarrow 3, 2 \rightarrow 4, 2 \rightarrow 5, 2 \rightarrow 6, \\ 3 \rightarrow 4, 3 \rightarrow 5, 3 \rightarrow 6, 4 \rightarrow 5, 4 \rightarrow 6, 5 \rightarrow 6\}$$

Definimos las funciones necesarias:

```
In]:=      IND[W1_,F_]:=Module[{i},  
          :           :  
          ;           ;  
In]:=      <<Combinatorica`;  
In]:=      BIPARTITO[W_,F_]:=Module[{i,subconjuntos,j,fin},  
          :           :  
          ;           ;
```

Y lo comprobamos:

```
In]:=      BIPARTITO[W,F]  
Out]= No es bipartito
```

Análogamente para G :

```
In]:=      BIPARTITO[{1,2,3,4,5,6},  
          {1 \rightarrow 4,1 \rightarrow 5,1 \rightarrow 6,2 \rightarrow 4,2 \rightarrow 5,2 \rightarrow 6,3 \rightarrow 4,3 \rightarrow 5,3 \rightarrow 6}]  
Out]= Es bipartito: W1={1,2,3} y W2={4,5,6}  
Es bipartito completo: K3,3
```

□

También podemos definir cualquier grafo $K_{n,m}$, para ello consideramos $W = \{1, 2, \dots, n+m\}$ y F sería el conjunto de todos los posibles lados incidentes en un vértice de $W_1 = \{1, 2, \dots, n\}$ y otro de $W_2 = \{n+1, n+2, \dots, n+m\}$. Con Mathematica:

| FUNCTION | COMENTARIOS |
|---|--|
| BC[n_,m_]:=Module[{i,j}, W=Table[i,{i,n+m}]; F={}; Do[Do[AppendTo[F,i \rightarrow j],{i,n}],{j,n+1,n+m}];] | Por entrada tendrán dos enteros positivos mayores que cero. Definimos el conjunto de vértices. Definimos el conjunto de lados. |

Función 6.12. Grafos bipartitos completos.

En el paquete `Combinatorica` (versión 6 de Mathematica o posterior), disponemos de las funciones:

BipartiteQ[G] y **CompleteKPartiteGraph[a,b]**

La primera comprueba si el grafo asociado al objeto de tipo grafo “G” es bipartito y la segunda crea grafos bipartitos completos, donde “a” y “b” son dos enteros positivos.

Ejemplo 6.9. Representamos gráficamente $K_{2,3}$, $K_{3,3}$ y $K_{5,6}$.

Utilizamos la función 6.12. para conseguir W y F , pero los representaremos usando la matriz de adyacencia porque la asignación de índices que hace Mathematica es distinta de la nuestra y puede dar lugar a confusión, para ello usaremos la función 5.1.:

In[]:= **MATRIZADYACENCIA[W,F]:=Module[{k,i,j},**

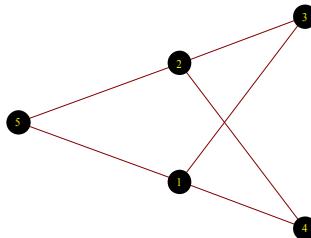
⋮ ⋮

In[]:= **BC[n,m]:=Module[{i,j},**

⋮ ⋮

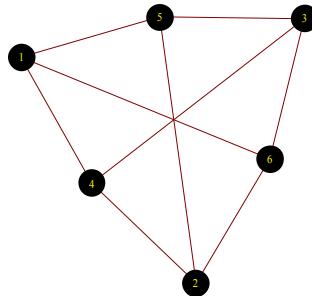
In[]:= **BC[2,3];**
MA=MATRIZADYACENCIA[W,F];
GraphPlot[MA,VertexRenderingFunction→
 $\{ \text{Black,Disk}[\#, .07], \text{Yellow,Text}[\#2,\#1]\} \&)]$

Out[] =



In[]:= **BC[3,3];**
MA=MATRIZADYACENCIA[W,F];
GraphPlot[MA,VertexRenderingFunction→
 $\{ \text{Black,Disk}[\#, .07], \text{Yellow,Text}[\#2,\#1]\} \&)]$

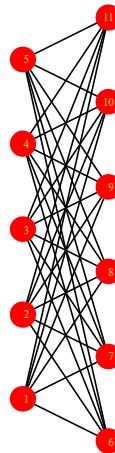
Out[] =



El último lo haremos con CompleteKPartiteGraph[]:

```
In[]:= <<Combinatorica`  
In[]:= G=CompleteKPartiteGraph[5,6];  
ShowGraph[G,VertexColor->Red,  
VertexStyle->PointSize[0.08],  
VertexNumber->True,  
VertexNumberColor->Yellow,  
VertexNumberPosition->{+.02,+.0}]
```

Out[]=



Y efectivamente:

```
In[]:= BipartiteQ[G]
```

Out[]= True

□

También podemos determinar directamente la matriz de adyacencia de $K_{n,m}$:

$$\left(\begin{array}{cccccc} 0 & \cdots & 0 & 1 & \cdots & 1 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & \cdots & 1 \\ 1 & \cdots & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 1 & \cdots & 1 & 0 & \cdots & 0 \end{array} \right)$$

n columnas m columnas

n filas m filas

Con Mathematica:

| FUNCIÓN | COMENTARIOS |
|---|--|
| <code>BCadyacencia[n_,m_]:=Table[If[(i≤n &&j≤n) (i>n &&j>n),0,1],{i,n+m},{j,n+m}];</code> | Las entradas serán dos enteros positivos mayores que cero. |

Función 6.13. Grafos bipartitos completos.

Ejemplo 6.10. Determinamos la matriz de adyacencia de $K_{5,5}$ con la función 6.13.:

In[]:= `BCadyacencia[n_,m_]:=Table[If[(i≤n &&j≤n)||
(i>n &&j>n),0,1],{i,n+m},{j,n+m}];`

In[]:= `MatrixForm[BCadyacencia[5,5]]`

Out[]=

$$\left(\begin{array}{cccccccccc} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{array} \right)$$

□

Ejemplo 6.11. Vamos a representar gráficamente todos los grafos bipartitos completos de la forma $K_{n,n}$ con $n \in \mathbb{N}$ y $2 \leq n \leq 8$.

Usaremos la función 6.13.:

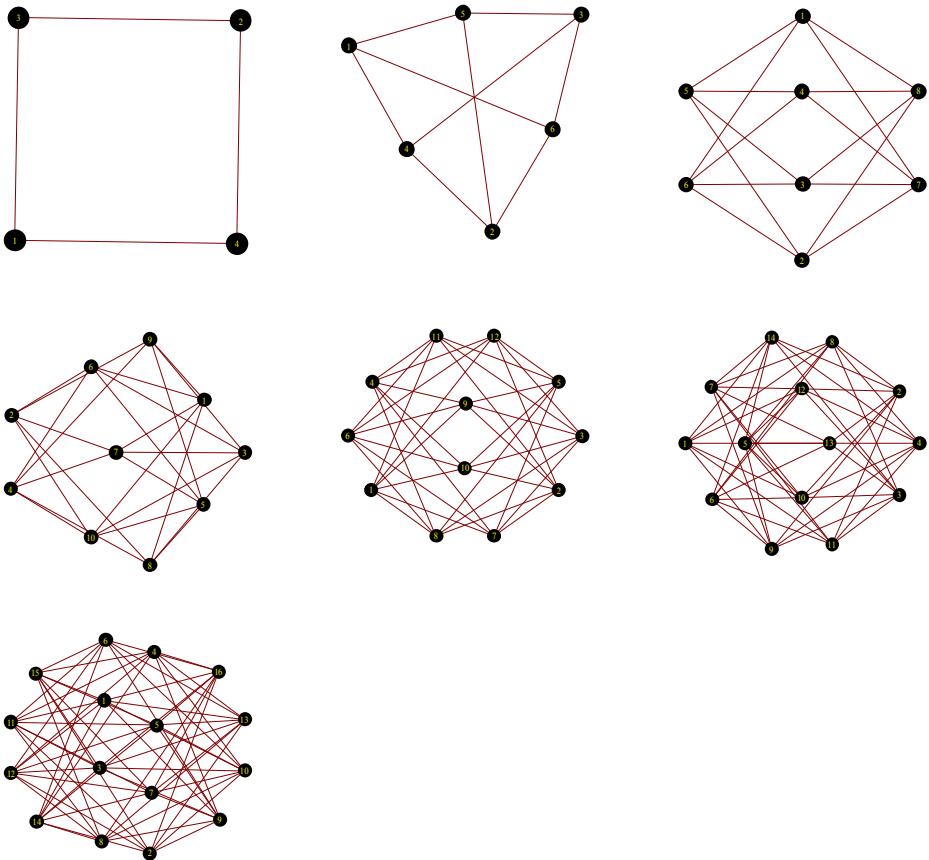
In[]:= `BCadyacencia[n_,m_]:=Table[If[(i≤n &&j≤n)||`

$(i > n \ \&\& j > n), 0, 1], \{i, n+m\}, \{j, n+m\}\};$

In[7]:=

```
listado={};
Do[
A=BCadyacencia[i,i];
AppendTo[listado,
GraphPlot[A,VertexRenderingFunction→
({Black,Disk[#,0.07],Yellow,Text[#2,#1]}&)],
{i,2,8}];
TableForm[listado]
```

Out[7]=



□

3.4. COMPLEMENTO DE UN GRAFO

Sea G un (p, q) -grafo (no orientado), entonces G podemos verlo como un subgrafo de K_p (renombrando sus vértices si fuese necesario), llamaremos complemento de G y lo denotaremos por \overline{G} , al subgrafo maximal de K_p tal que $G \cup \overline{G} = K_p$.

Determinarlo es inmediato, si $G = (W, F_1)$ y $K_p = (W, F)$, entonces

$$\overline{G} = (W, F - F_1)$$

y con Mathematica sería análogo:

```
Wc=W
Fc=Complement[F,F1]
```

En el paquete `Combinatorica` de la versión 6 y siguientes de Mathematica, disponemos de la función:

```
GraphComplement[G]
```

que calcula el completo del grafo asociado al objeto de tipo grafo “G”.

Ejemplo 6.12. Representamos gráficamente el complemento de $K_{3,3}$.

Introducimos $K_{3,3}$, podemos usar la función 6.12. o directamente:

```
In]:= W={1,2,3,4,5,6};
F1={1→4,1→5,1→6,2→4,2→5,2→6,3→4,3→5,3→6};
```

Determinamos K_6 :

```
In]:= n=6;
F={};
Do[Do[AppendTo[F,i→j],{j,i+1,n}],{i,n-1}];
```

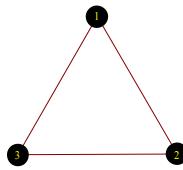
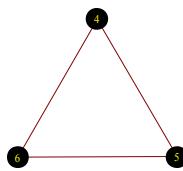
Calculamos el complemento y lo representamos gráficamente:

```
In]:= Fc=Complement[F,F1]
```

```
Out]= {1→2, 1→3, 2→3, 4→5, 4→6, 5→6}
```

```
In]:= GraphPlot[Fc,VertexRenderingFunction→
({Black,Disk[#,07],Yellow,Text[#2,#1]}&)]
```

```
Out]=
```



Y con GraphComplement[]:

In[]:= **<<Combinatorica`**

In[]:= **G=FromOrderedPairs[Table[{F1[[i]][[1]],F1[[i]][[2]]}, {i,Length[F1]}],Type->Undirected];**

Out[]=-<9,6,Undirected>-

In[]:= **GraphComplement[G]**

Out[]=-<6,6,Undirected>-

□

4. EJERCICIOS

Ejercicio 6.1. Realizar una rutina que determine el grado de entrada y de salida de un vértice desde la matriz de adyacencia de un grafo dirigido.

□

Ejercicio 6.2. Determinar si un grafo no orientado es regular o completo desde su matriz de incidencia.

□

Ejercicio 6.3. Determinar si un vértice es aislado, sumidero o fuente directamente desde la matriz de adyacencia.

□

Ejercicio 6.4. La siguiente función determina si existe algún vértice de grado menor que 2:

| FUNCIÓN | COMENTARIOS |
|---|--|
| GRADO2[A_]:=Module[{j,B}, | Por entrada tendrá a la matriz de adyacencia. |
| B=MatrixPower[A,2]; grado2=True; Do[If[B[[j,j]]<2, grado2=False;Break[];] ,{j,Dimensions[A][[1]]}; | Comprueba si existe algún vértice de grado 0 (aislado) o 1 (colgante). |
| grado2] | Salida de resultados. |

Función 6.14. Vértices aislados o colgantes.

La función anterior comprueba si existe algún vértice aislado o colgante (de grado 1) en un grafo no orientado. Aplicarla a los grafos de las ilustraciones 5.2., 5.3., 5.10. y 5.12.

□

Ejercicio 6.5. Determinar si un grafo es regular directamente desde W y F sin utilizar la función GRADO[] (6.1.).

□

Ejercicio 6.6. Calcular todos los grafos distintos, salvo isomorfismo, 3-regulares de 7 vértices.

□

Ejercicio 6.7. Comprobar cuántos grafos completos no isomorfos existen de 7 vértices.

□

Ejercicio 6.8. Determinar cuáles de los siguientes grafos son regulares, completos, bipartitos o bipartitos completos:

- a) $G_1 = (W_1, F_1)$ con $W_1 = \{1, 2, 3, 4, 5, 6\}$ y $F_1 = \{1 \rightarrow 2, 3 \rightarrow 1, 4 \rightarrow 5, 6 \rightarrow 1\}$.
 b) G_2 con matriz de adyacencia:

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

- c) G_3 con matriz de incidencia:

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

d) Los grafos de la siguiente ilustración:

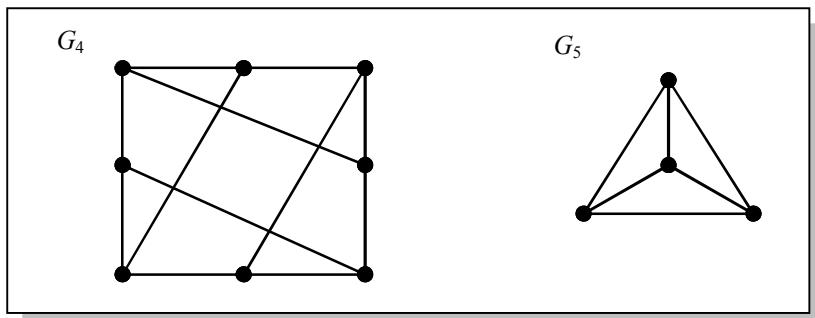


Ilustración 6.1.

□

Ejercicio 6.9. Calcular todos los grafos de 8 vértices que tengan como subgrafo a K_5 o $K_{3,3}$.

□

Ejercicio 6.10. Determinar todos los grafos dirigidos distintos salvo isomorfismo de 4 vértices con 2 sumideros y 2 vértices.

□

Ejercicio 6.11. Calcular todos los subgrafos de $K_{3,3}$ no isomorfos.

□

Ejercicio 6.12. Calcular todos los subgrafos de K_5 distintos salvo isomorfismo, inducidos por subconjuntos de vértices.

□

Ejercicio 6.13. Calcular el complemento de los grafos de la ilustración 6.1.

□

Ejercicio 6.14. Calcular la unión e intersección de los subgrafos de K_5 inducidos por los conjuntos de vértices $\{1,2,3\}$ y $\{3,4,5\}$. ¿Son bipartitos o bipartitos completos? Determinar el complemento de ambos.

□

Ejercicio 6.15. Podemos observar que $Aut(K_n) = Aut(G) = S_n$ con $G = (\{v_1, \dots, v_n\}, \emptyset)$, esto es, un grafo de n vértices sin lados. Comprobarlo para K_4 y K_6 .

□

Ejercicio 6.16. Es fácil comprobar que $Aut(G) = Aut(\overline{G})$. Elegir varios grafos de 5 y 6 vértices y comprobarlo con Mathematica.

□

Ejercicio 6.17. Calcular todos los grafos de 7 vértices, salvo isomorfismo, que tengan a $K_{3,3}$ como subgrafo.

□

Ejercicio 6.18. Calcular todos los grafos, salvo isomorfismo, que tengan a $K_{4,4}$ como subgrafo maximal.

□

Ejercicio 6.19. Calcular todos los grafos de 8 vértices, salvo isomorfismo, que tengan a K_5 o $K_{3,3}$ como subgrafo.

□

Ejercicio 6.20. Calcular y representar gráficamente $K_{n,m}$ para cada $n, m \in \mathbb{N}$ con $2 \leq n, m \leq 8$.

□

Ejercicio 6.21. Estudiar el funcionamiento de las siguientes funciones del paquete <<Combinatorica` de la versión 6 de Mathematica:

- a) **GraphUnion[G1,G2,...]**
- b) **GraphIntersection[G1,G2,...]**
- c) **GraphJoin[G1,G2,...]**
- d) **GraphSum[G1,G2,...]**
- e) **GraphProduct[G1,G2,...]**
- f) **GraphDifference[G1,G2]**

□

7. CAMINOS Y CICLOS

Este capítulo lo dedicamos enteramente al estudio de los caminos de un grafo, las preguntas más recurrentes que suelen aparecer en problemas formalizados matemáticamente con grafos suelen estar relacionados con los caminos: conectividad entre vértices, el tipo de caminos que podamos encontrar, la longitud de los mismos, si son ciclos,... Usamos para su estudio el teorema del número de caminos y los calculamos haciendo barridos sobre todas las posibilidades. Terminamos comprobando cuándo un grafo contiene un ciclo de Euler o un ciclo de Hamilton y si existen los calculamos.

1. DEFINICIÓN Y TIPOS DE CAMINOS

Analizaremos el concepto distinguiendo entre grafos dirigidos y no orientados:

1.1. CAMINOS EN GRAFOS NO ORIENTADOS

Sea $G = (W, F)$ un grafo no orientado, un camino c es una sucesión finita de vértices

$$c = \{v_1, v_2, \dots, v_{n+1}\}, \text{ con } v_i \in W \text{ para cada } i = 1, 2, \dots, n+1;$$

tales que dos vértices consecutivos son adyacentes, esto es, verificando: $\{v_i, v_{i+1}\} = e_i \in F$ para cada $i = 1, 2, \dots, n$; también podemos denotar a un camino como la sucesión de lados

$$c = \{e_1, e_2, \dots, e_n\}, \text{ con } e_i \in F \text{ para cada } i = 1, 2, \dots, n;$$

o incluso también suele ser habitual denotarlos como la sucesión alternada de los vértices y los lados anteriores:

$$c = \{v_1, e_1, v_2, e_2, v_3, \dots, v_n, e_n, v_{n+1}\}.$$

Diremos que n es la longitud del camino c y la denotaremos por $l(c)$. También diremos que el camino c conecta los vértices v_1 y v_{n+1} , siendo v_1 el origen y v_{n+1} el fin del camino, que denotaremos por $s(c)$ y $e(c)$ respectivamente. Si $n = 0$, entonces entenderemos que tenemos un camino trivial de longitud cero cuya sucesión de vértices asociada tendrá longitud 1 y será de la forma $\{v_1\}$ sin ningún lado. Si un lado o vértice pertenece al camino diremos que dicho camino pasa por ese lado o vértice.

Por lo general un camino lo introduciremos en el ordenador como una lista de vértices y en Mathematica escribiremos:

listavertices={v₁,v₂,...,v_{n+1}};

Es claro que “Length[listavertices] – 1)” es la longitud del camino, “listavertices[[1]]” es el inicio del camino y “listavertices[[n+1]]” es el fin, esto es, conecta a “listavertices[[1]]” con “listavertices[[n+1]]” y pasa por todo los vértices: “listavertices[[i]]” para $1 < i < (n + 1)$.

En Mathematica también podemos introducir caminos como una lista de lados:

listalados={v₁→v₂,v₂→v₃,...,v_k→v_{k+1}};

pero atención, esta notación en grafos no orientados es ambigua, una sucesión de lados puede representar a varios caminos distintos sin que podamos distinguirlos, por tanto la evitaremos siempre que podamos y nos limitaremos a denotar los caminos en grafos no orientados como listas de vértices. En cuanto a la última notación que hemos comentado, sucesión alternada de vértices y lados, aunque es la más explícita, no aporta demasiado respecto a la sucesión de vértices, pues el camino y los lados que los constituyen están obligados a estar biunívocamente determinados en los grafos con los que trabajamos (porque no son multigrafos³¹).

Ejemplo 7.1. Consideremos el grafo no orientado $G = (W, F)$ dado por $W = \{1, 2, 3, 4, 5\}$ y $F = \{1 \rightarrow 2, 3 \rightarrow 2, 4 \rightarrow 3, 5 \rightarrow 1, 5 \rightarrow 4, 2 \rightarrow 4, 3 \rightarrow 5\}$, entonces un camino denotado como sucesión de vértices $c = \{1, 2, 3, 4, 5, 1, 5, 4, 2\}$ lo introduciríamos en el ordenador directamente:

In]:= **caminoververtices=**{1,2,3,4,5,1,5,4,2};

o como sucesión de lados:

In]:= **caminolados=**{1→2,2→3,3→4,4→5,5→1,1→5,5→4,4→2};

Recordemos que en un grafo no orientado un mismo lado {v, w} puede representarse de dos formas: $v \rightarrow w$ o $w \rightarrow v$, por tanto podríamos también introducirlo por ejemplo como:

In]:= **caminolados2=**{1→2,3→2,4→3,5→4,5→1,5→1,5→4,2→4};

Y como sucesión alternada de vértices y lados:

In]:= **caminoververticeslados=**{1,1→2,2→3,3→4,4→5,5→1,1,

³¹ En el caso de multigrafos, la sucesión de vértices no es suficiente para determinar al camino ya que entre dos vértices pueden existir varios lados incidentes. La forma más correcta de denotar a los caminos en multigrafos sería como la sucesión alternada de vértices y lados.

$$1 \rightarrow 5, 5 \rightarrow 4, 4 \rightarrow 2, 2\};$$

Ahora consideremos el siguiente camino denotado como sucesión de lados $d = \{1 \rightarrow 5, 1 \rightarrow 5\}$, observemos que este podría representar a cualquiera de los dos siguientes:

$$d1 = \{1, 5, 1\} \text{ o } d2 = \{5, 1, 5\}.$$

Por tanto en grafos no orientados deberemos tener presente la ambigüedad de la notación como sucesión de lados y evitarla cuando podamos. \square

Podemos generar unos pequeños tests que comprueben si una sucesión de vértices es un camino:

| FUNCIÓN | COMENTARIOS |
|---|--|
| <pre>CAMINO[camino_F]:=Module[{i}, cam=True; Do[If[Intersection[{camino[[i]]→camino[[i+1]], camino[[i+1]]→camino[[i]]},F]=={} , cam=False;Break[]]; ,{i,Length[camino]-1}]; cam]</pre> | Función cuyas entradas serán: el camino expresado como sucesión de vértices y el conjunto de lados del grafo. Determina si la lista “camino” de vértices es realmente un camino en el grafo cuyo conjunto de lados es “F”. Salida de resultados. |

Función 7.1. Caminos en grafos no dirigidos.

| FUNCIÓN | COMENTARIOS |
|--|---|
| <pre>CAMINO2[camino_matrizadyacencia]:=Module[{i}, cam=True; Do[If[matrizadyacencia[[camino[[i]],camino[[i+1]]]]==0, cam=False;Break[]]; ,{i,Length[camino]-1}]; cam]</pre> | Función cuyas entradas serán: el camino expresado como sucesión de vértices (identificados por sus subíndices) y la matriz de adyacencia del grafo. Determina si la lista “camino” de vértices es realmente un camino en el grafo cuya matriz de adyacencia es “matrizadyacencia” Salida de resultados. |

Función 7.2. Caminos en grafos no dirigidos.

También podemos calcular la sucesión de lados asociada un camino que denotemos como sucesión de vértices (téngase en cuenta siempre la ambigüedad de esta notación):

| FUNCIÓN | COMENTARIOS |
|---------------------------------|---|
| SUCLADOS[vertices]:=Module[{i}, | Función que tendrá por entrada un camino expresado como sucesión de vértices. |

| | |
|---|--|
| <code>lados={}; Do[AppendTo[lados,vertices[[i]]→vertices[[i+1]]]; ,{i,Length[vertices]-1}]; lados]</code> | Calculamos la sucesión de lados que denota al camino. Salida de resultados. |
|---|--|

Función 7.3. Caminos en grafos no dirigidos.

Y si tenemos una sucesión de lados que denota a un camino del grafo no orientado, también podemos recuperar desde esta última la sucesión de vértices (de nuevo debemos tener presente la ambigüedad de la notación):

| FUNCIÓN | COMENTARIOS |
|--|---|
| <code>SUCVERTICES[lados_]:=Module[{i}, vertices={lados[[1]][[1]]}; Do[AppendTo[vertices,lados[[i]][[2]]]; ,{i,Length[lados]}]; vertices]</code> | El argumento de la función será un camino expresado como sucesión de lados. |
| <code>vertices={lados[[1]][[1]]}; Do[AppendTo[vertices,lados[[i]][[2]]]; ,{i,Length[lados]}]; vertices]</code> | Calculamos la sucesión de vértices que denota al camino. |
| | Salida de resultados. |

Función 7.4. Caminos en grafos no dirigidos.

Nos quedarían por realizar programas análogos para la notación como sucesión alternada de vértices y lados, como no vamos a utilizar multigrafos relegamos este estudio al ejercicio 7.24.

Ejemplo 7.2. Consideramos la sucesión de vértices $\{1, 2, 3, 4, 5, 1\}$ de K_5 , comprobar si es un camino, expresarlo como sucesión de lados y viceversa.

Definimos la función 7.2.:

In[]:= CAMINO2[camino_,matrizadyacencia_]:=Module[{i},

$\vdots \qquad \vdots$

*In[]:= K5=Table[If[i==j,0,1],{i,5},{j,5}];
CAMINO2[\{1,2,3,4,5,1\},K5]*

Out[] = True

Determinamos la sucesión de lados con la función 7.3.:

In[]:= SUCLADOS[vertices_]:=Module[{i},

$\vdots \qquad \vdots$

In[]:= lados=SUCLADOS[\{1,2,3,4,5,1\}]

Out[] = {1→2, 2→3, 3→4, 4→5, 5→1}

Téngase presente para estos casos que en grafos no orientados, los lados $v_i \rightarrow v_j$ y $v_j \rightarrow v_i$ los identificamos. Por último calculamos la sucesión de vértices desde la de lados con la función 7.4.:

In[]:= SUCVERTICES[lados_]:=Module[{i},

$\vdots \qquad \vdots$

In[]:= SUCVERTICES[lados]

Out[] = {1, 2, 3, 4, 5, 1}

□

1.2. CAMINOS EN GRAFOS DIRIGIDOS

En los grafos dirigidos debemos de distinguir entre dos tipos de caminos, los caminos orientados y los no orientados que definimos a continuación:

Sea $G = (W, F)$ un grafo dirigido, entonces:

- a) Un camino c es una sucesión finita de vértices $\{v_1, v_2, \dots, v_{n+1}\} \subseteq W$ verificando:

$$(v_i, v_{i+1}) \in F \text{ ó } (v_{i+1}, v_i) \in F \text{ para cada } i = 1, 2, \dots, n;$$

si llamamos f_i a la correspondiente flecha, también podemos denotarlo como la sucesión de flechas $\{f_1, f_2, \dots, f_n\} \subseteq F$ o como una sucesión alternada de vértices y flechas $\{v_1, f_1, v_2, f_2, v_3, \dots, v_n, f_n, v_{n+1}\}$. (Si obviamos la orientación del grafo, un camino no orientado, conceptualmente coincide con un camino del grafo no orientado).

- b) Un camino orientado o camino dirigido c es una sucesión finita de flechas $\{f_1, f_2, \dots, f_n\} \subseteq F$ o de vértices $\{v_1, v_2, \dots, v_{n+1}\} \subseteq W$ verificando:

- 1) $e(f_i) = s(f_{i+1}) = v_{i+1};$
- 2) $s(f_1) = v_1$ y $e(f_n) = v_{n+1};$

es decir, $(v_i, v_{i+1}) = f_i \in F$ para cada $i = 1, 2, \dots, n$.

Al igual que para grafos no dirigidos, diremos que n es la longitud del camino c (orientado o no) y la denotaremos por $l(c)$, diremos que conecta los vértices v_1 y v_{n+1} , siendo v_1 el origen y v_{n+1} el fin del camino, que también denotaremos por $s(c)$ y $e(c)$ respectivamente. Si $n = 0$, al igual que antes, entenderemos que tenemos un camino trivial de longitud cero cuya sucesión de vértices asociada tendrá longitud 1 y será de la forma $\{v_1\}$ sin ninguna flecha. Si una flecha o vértice pertenece al camino diremos que dicho camino pasa por esa flecha o vértice.

1.2.1. CAMINOS NO ORIENTADOS EN GRAFOS DIRIGIDOS

En Mathematica podemos implementar caminos no orientados como una lista de vértices. La sucesión de vértices la introduciremos en Mathematica de la misma forma que para grafos no orientados:

$$\text{listavertices} = \{v_1, v_2, \dots, v_{n+1}\}$$

De nuevo, evitaremos denotar los caminos no orientados como sucesión de flechas, por resultar ambigua. La notación como sucesión alternada de vértices y lados evitará esta ambigüedad pero lo más cómodo será denotarlos siempre como sucesión de vértices, puesto que no consideramos multigrafos. Los caminos no orientados de un grafo dirigido pueden estudiarse directamente con las funciones 7.1. y 7.2. olvidándonos de la orientación.

1.2.2. CAMINOS ORIENTADOS EN GRAFOS DIRIGIDOS

La misma función 7.1. con ligeros cambios nos vale para comprobar si una sucesión de vértices es un camino orientado del grafo, mientras que 7.2. no es necesario modificarla, pues está basada en la matriz de adyacencia y ésta refleja correctamente la orientación:

| FUNCIÓN | COMENTARIOS |
|---|---|
| CAMINOORI [camino_,F_]:=Module[{i}, cam=True; Do[If[Intersection[{camino[[i]]→camino[[i+1]]},F]=={}, cam=False;Break[]]; ,{i,Length[camino]-1}]; cam] | Funció n con dos entradas: un camino expresado como lista de vértices y el conjunto de todos los lados del grafo. |
| | Determina si la lista de vértices: "camino" de vértices es realmente un camino no orientado en el grafo. |
| | Salida de resultados. |

Función 7.5. Caminos orientados en grafos dirigidos.

Sin embargo en el caso de caminos orientados la sucesión de flechas resulta perfectamente válida:

$$\text{CAMINO} = \{v_1 \rightarrow v_2, v_2 \rightarrow v_3, \dots, v_k \rightarrow v_{k+1}\}$$

Completamos el estudio de los caminos orientados con las siguientes funciones que determinarán, para una sucesión de flechas o vértices, en primer lugar si verdaderamente es un camino y en segundo la otra notación, esto es, la sucesión de vértices o flechas respectivamente:

| FUNCIÓN | COMENTARIOS |
|--|---|
| CAMINODIRIGIDO [lados_,F_]:=Module[{i}, | Funció n con dos entradas: un camino expresado como una sucesión de flechas y el conjunto de todas las flechas del grafo. |

| | |
|--|--|
| <pre> camino=True; listavertices={}; If[Length[Intersection[lados,F]]!=Length[lados], camino=False;; If[Length[lados]!=1, listavertices={lados[[1]][[1]]}; Do[If[lados[[i]][[2]]!=lados[[i+1]][[1]], camino=False;Break[]]; AppendTo[listavertices,lados[[i]][[2]]]; ,{i,Length[lados]-1}]; AppendTo[listavertices, lados[[Length[lados]]][[2]]]; , listavertices={lados[[1]][[1]],lados[[1]][[2]]};];]; </pre> | <p>Se comprueban las dos condiciones necesarias para que una sucesión de flechas sea un camino y se construye la sucesión de vértices.</p> |
| <pre> If[camino, Print["Es un camino orientado"]; Print[listavertices]; Print[lados];Print["No es un camino orientado"]]] </pre> | <p>Salida de resultados.</p> |

Función 7.6. Caminos orientado en grafos dirigidos.

Análogamente, dada una lista de vértices, podemos comprobar si es un camino orientado y la lista o sucesión de lados asociada:

| FUNCIÓN | COMENTARIOS |
|---|--|
| <pre> CAMINODIRIGIDO2[vertices_,F_]:=Module[{i}, camino=True; listalados={}; Do[If[Intersection[{vertices[[i]]→vertices[[i+1]]},F]=={}, camino=False;Break[]] , AppendTo[listalados, Intersection[{vertices[[i]]→vertices[[i+1]], vertices[[i+1]]→vertices[[i]]},F][[1]]];]; ,{i,Length[vertices]-1}]; </pre> | <p>Funció con dos entradas: un camino expresado como una sucesión de vértices y el conjunto de todas las flechas del grafo.</p> |
| <pre> If[camino, Print["Es un camino orientado"]; Print [listalados]; Print[vertices];Print["No es un camino orientado"]]] </pre> | <p>Se comprueban las dos condiciones necesarias para que una sucesión de vértices sea un camino y se construye la sucesión de lados.</p> |
| | <p>Salida de resultados.</p> |

Función 7.7. Caminos orientados en grafos dirigidos.

De nuevo dejamos como ejercicio el mismo estudio para la notación como sucesión de alternada de vértices y lados, (ejercicio 7.25.).

Ejemplo 7.3. Consideramos el dígrafo $G = (W, F)$ con $W = \{1, 2, 3, 4, 5\}$ y $F = \{1 \rightarrow 2, 3 \rightarrow 2, 5 \rightarrow 2, 2 \rightarrow 4, 4 \rightarrow 5, 5 \rightarrow 1, 1 \rightarrow 4, 3 \rightarrow 4, 3 \rightarrow 1, 5 \rightarrow 3\}$. Determinar si alguna de las siguientes sucesiones es un camino, comprobar si es un camino orientado o no orientado:

- a) $\{1 \rightarrow 4, 4 \rightarrow 5, 5 \rightarrow 3, 2 \rightarrow 3, 1 \rightarrow 2\}$.
- b) $\{1 \rightarrow 4, 4 \rightarrow 5, 5 \rightarrow 3, 3 \rightarrow 2, 2 \rightarrow 4\}$.
- c) $\{3 \rightarrow 2, 2 \rightarrow 4, 4 \rightarrow 5, 5 \rightarrow 1, 1 \rightarrow 2\}$.
- d) $\{1, 3, 4, 2, 5\}$.
- e) $\{2, 4, 5, 1, 4, 5, 2\}$.

Introducimos el grafo en el ordenador:

In[]:= **W={1,2,3,4,5};**
F={1→2,3→2,5→2,2→4,4→5,5→1,1→4,3→4,3→1,5→3};

- a) Comprobamos si $\{1 \rightarrow 4, 4 \rightarrow 5, 5 \rightarrow 3, 2 \rightarrow 3, 1 \rightarrow 2\}$ es un camino orientado con la función 7.6.:

In[]:= **CAMINODIRIGIDO[lados_,F_]:=Module[{i},**
 ⋮ **⋮**

In[]:= **CAMINODIRIGIDO[\{1→4,4→5,5→3,2→3,1→2\},F]**

Out[]= No es un camino orientado

Para comprobar si es un camino no orientado, en primer lugar se comprueba si todas las flechas pertenecen al grafo:

In[]:= **Length[Intersection[\{1→4,4→5,5→3,2→3,1→2\},F]]**
==Length[\{1→4,4→5,5→3,2→3,1→2\}]

Out[]= False

Luego tampoco es un camino no orientado.

- b) Hacemos lo mismo para $\{1 \rightarrow 4, 4 \rightarrow 5, 5 \rightarrow 3, 3 \rightarrow 2, 2 \rightarrow 4\}$.

In[]:= **CAMINODIRIGIDO[\{1→4,4→5,5→3,3→2,1→2\},F]**

Out[]= No es un camino orientado

Para comprobar si es un camino no orientado, en primer lugar se comprueba si todas las flechas pertenecen al grafo:

In[]:= **Length[Intersection[\{1→4,4→5,5→3,3→2,1→2\},F]]**

==Length[{1→4,4→5,5→3,3→2,1→2}]

Out[[*] = True*

La dificultad ahora radica en que la notación de un camino no orientado como sucesión de lados es ambigua, tendríamos que pasar a una de las dos notaciones alternativas, el problema es que debido a la ambigüedad que se tiene, es una tarea fútil crear un método para este propósito ya que la salida puede no ser única y además errónea (véase ejemplo 7.1.). En este caso, es casi evidente que la intencionalidad es la de considerar el camino no orientado cuya sucesión de vértices es $\{1, 4, 5, 3, 2, 1\}$, pero téngase en cuenta que no es la única y que en otros casos puede no ser tan evidente. Comprobamos si es camino no orientado usando directamente la función 7.1.

In[7]:= CAMINO[camino_,F_]:=Module[{i},

• • • •

In[]:= CAMINO[{1,4,5,3,2,1},F]

Out[[*] = True*

- c) Ahora lo comprobamos para $\{3 \rightarrow 2, 2 \rightarrow 4, 4 \rightarrow 5, 5 \rightarrow 1, 1 \rightarrow 2\}$.

```
In[7]:= CAMINODIRIGIDO[{3→2,2→4,4→5,5→1,1→2},F]
```

Out[]= Es un camino orientado

$$\{3, 2, 4, 5, 1, 2\}$$

$$\{3 \rightarrow 2, 2 \rightarrow 4, 4 \rightarrow 5, 5 \rightarrow 1, 1 \rightarrow 2\}$$

O bien, podemos comprobarlo con la función 7.5. (la sucesión de vértices desde un camino dirigido está determinada de forma biunívoca, por lo que no existe ambigüedad).

```
In[7]:= CAMINOORI[camino_,F_]:=Module[{i},
```

• • • • •

In[7]:= CAMINOORI[{3,2,4,5,1,2},F]

Out[*] = True*

- d) Para el camino $\{1, 3, 4, 2, 5\}$ usaremos la función 7.7.

```
In[7]:= CAMINODIRIGIDO2[vertices ,F ]:=Module[{i},
```

⋮ ⋮

In[]:= **CAMINODIRIGIDO2[{1,3,4,2,5},F]**

Out[]:= No es un camino orientado

O bien con 7.5.:

In[]:= **CAMINOORI[{1,3,4,2,5},F]**

Out[]:= False

Veamos si es un camino no orientado, usaremos directamente 7.1.:

In[]:= **CAMINO[{1,3,4,2,5},F]**

Out[]:= True

- e) Y por último, el camino {2, 4, 5, 1, 4, 5, 2}.

In[]:= **CAMINODIRIGIDO2[{2,4,5,1,4,5,2},F]**

Out[]:= Es un camino orientado

{ 2→4 , 4→5 , 5→1 , 1→4 , 4→5 , 5→2 }

{ 2, 4, 5, 1, 4, 5, 2 }

□

Ahora cabría modificar las funciones 7.6. y 7.7. para que funcionasen con las matrices de adyacencia, para no ser reiterativos dejamos esta tarea propuesta en el ejercicio 7.12.

1.3. TIPOS DE CAMINOS

Un camino cualquiera se dirá simple si no se repite ningún lado o flecha, esto es, no pasa dos veces por el mismo lado o flecha, se dirá que es elemental si además de ser simple tampoco pasa más de una vez por el mismo vértice (aunque si puede empezar y terminar en el mismo vértice).

Un camino c se dirá que es cerrado si $s(c) = e(c)$, los caminos cerrados se llamarán ciclos. Un camino simple y cerrado se llama círculo.

En Mathematica, es inmediato hacer estas comprobaciones.

- a) Para un grafo no orientado, si tenemos un camino $c = \{v_1, v_2, \dots, v_{n+1}\}$ que introduciremos:

camino= {v₁,v₂,...,v_{n+1}};

Comprobaremos si es simple calculando la sucesión de lados y comprobando que todos sean distintos:

| PROGRAMA | COMENTARIOS |
|--|---|
| F=CONJUNTO DE LADOS DEL GRAFO; | Introducimos el conjunto de lados del grafo. |
| camino=LISTA DE VÉRTICES; | Introducimos un camino expresado como sucesión de vértices en la variable "camino". |
| SUCLADOS[vértices]:=...; | Definimos la función 7.3. |
| lados=SUCLADOS[camino]; | |
| n=Length[lados]; Do[AppendTo[lados,lados[[i]][[2]]->lados[[i]][[1]]],{i,n}]; If[Length[Intersection[lados,F]]==n, simple=True,simple=False]; | Calculamos la sucesión de vértices que denota al camino. |
| simple | Salida de resultados. |

Programa 7.8. Caminos simples en grafos no dirigidos.

O bien comprobamos que no haya ninguno repetido utilizando herramientas de conjuntos:

| PROGRAMA | COMENTARIOS |
|--|---|
| F=CONJUNTO DE LADOS O FLECHAS DEL GRAFO; | Introducimos el conjunto de lados del grafo. |
| camino=LISTA DE VÉRTICES; | Introducimos un camino expresado como sucesión de vértices en la variable "camino". |
| SUCLADOS[vértices]:=...; | Definimos la función 7.3. |
| lados=SUCLADOS[camino]; If[Length[Intersection[lados,F]]==Length[lados], simple=True,simple=False]; | |
| simple | Salida de resultados. |

Programa 7.9. Caminos simples en grafos.

Para comprobar que también sea elemental lo hacemos con:

| PROGRAMA | COMENTARIOS |
|---|---|
| W=CONJUNTO DE VÉRTICES DEL GRAFO; | Introducimos el conjunto de vértices del grafo. |
| camino=LISTA DE VÉRTICES; | Introducimos un camino expresado como sucesión de vértices en la variable "camino". |
| If[camino[[1]]==camino[[Length[camino]]], long=Length[camino]-1,log=Length[camino]]; If[Length[Intersection[camino,W]]==long, elemental=True,elemental=False]; | |
| elemental | Salida de resultados. |

Programa 7.10. Caminos elementales en grafos.

Veremos si es cerrado usando:

camino[[1]]==camino[[Length[camino]]]

Y obviamente será circuito si:

(camino[[1]]==camino[[Length[camino]]]) && simple

- b) Para un grafo dirigido, si tenemos un camino dirigido $c = \{v_1, v_2, \dots, v_{n+1}\}$ que introduciremos:

camino= {v₁,v₂,...,v_{n+1}};

Comprobaremos si es simple calculando la sucesión de lados y comprobando que todos sean distintos con el programa 7.8., igual que para grafos no orientados y análogamente comprobaremos si es elemental con el programa 7.9.

Veremos si es cerrado,

camino[[1]]==camino[[Length[camino]]]

Y obviamente será circuito si,

(camino[[1]]==camino[[Length[camino]]]) && simple

- c) Por último, si se trata de un camino no orientado de un grafo dirigido,

camino= {v₁,v₂,...,v_{n+1}};

Determinamos la sucesión de flechas como sigue y el resto será igual.

| PROGRAMA | COMENTARIOS |
|---|---|
| F=CONJUNTO DE FLECHAS DEL GRAFO; | Introducimos el conjunto de lados del grafo. |
| camino=LISTA DE VÉRTICES; | Introducimos un camino expresado como sucesión de vértices en la variable "camino". |
| lados={}; Do[AppendTo[lados, Intersection[{vertices[[i]]->vertices[[i+1]], vertices[[i+1]]->vertices[[i]]},F][[1]]]; ,{i,Length[vertices]-1}]; lados | |
| | Salida de resultados. |

Programa 7.11. Sucesión de lados de un camino no orientado en un grafo dirigido.

Ejemplo 7.4. Comprobar cuáles de los siguientes caminos son simples en el grafo no orientado K_5 .

- a) $C_1 = \{1, 2, 3, 4, 5\}$.

- b) $C_2 = \{2, 3, 2, 3, 2\}$.
c) $C_3 = \{1, 2, 3, 4, 3\}$.

Definimos la matriz de adyacencia de K_5 , calculamos su conjunto de lados y vértices con 5.3. e introducimos en Mathematica los caminos:

```
In]:= matrizadyacencia=Table[If[i==j,0,1],{i,5},{j,5}];  

W=Table[i,{i,Dimensions[matrizadyacencia][[1]]}];  

F={};  

Do[Do[If[matrizadyacencia[[i,j]]==1,AppendTo[F,i->j]],{i,j}];  

,{j,Dimensions[matrizadyacencia][[1]]}];  

In]:= c1={1,2,3,4,5};  

c2={2,3,2,3,2};  

c3={1,2,3,4,3};
```

Comprobamos cuáles son simples con programa 7.8., previamente definimos la función 7.3.:

```
In]:= SUCLADOS[vertices_]:=Module[{i},
```

```
⋮ ⋮
```

a)

```
In]:= lados=SUCLADOS[c1];
```

```
⋮ ⋮
```

```
Out]= True
```

b)

```
In]:= lados=SUCLADOS[C2];
```

```
⋮ ⋮
```

```
Out]= False
```

c)

```
In]:= lados=SUCLADOS[C3];
```

```
⋮ ⋮
```

```
Out]= False
```

Veamos que C_1 además es elemental con el programa 7.10.:

```
In]:= If[Length[Intersection[c1,W]]==Length[W],  

elemental=True,elemental=False];
```

elemental

Out[]:= True

□

Ejemplo 7.5. Consideramos el dígrafo $G = (W, F)$ con $W = \{1, 2, 3, 4, 5\}$ y $F = \{1 \rightarrow 2, 3 \rightarrow 2, 5 \rightarrow 2, 2 \rightarrow 4, 4 \rightarrow 5, 5 \rightarrow 1, 1 \rightarrow 4, 3 \rightarrow 4, 3 \rightarrow 1, 5 \rightarrow 3\}$. Estudiar de qué tipo son los siguientes caminos no orientados:

- a) $C_1 = \{1, 2, 3, 4, 5\}$.
- b) $C_2 = \{2, 3, 2, 3, 2\}$.
- c) $C_3 = \{1, 2, 3, 4, 3\}$.

Introducimos el grafo y los caminos:

```
In[]:=      W={1,2,3,4,5};
F={1→2, 3→2, 5→2, 2→4, 4→5, 5→1, 1→4, 3→4, 3→1,
      5→3};
c1={1,2,3,4,5};
c2={2,3,2,3,2};
c3={1,2,3,4,3};
```

Determinamos la sucesión de flechas asociada a cada uno de ellos:

a)

```
In[]:=      lados1={};
Do[AppendTo[lados1,Intersection[{c1[[i]]→c1[[i+1]],
      c1[[i+1]]→c1[[i]]},F][[1]]];
,{i,Length[c1]-1}];
lados1
```

Out[]:= {1→2, 3→2, 3→4, 4→5 }

b)

```
In[]:=      lados2={};
Do[AppendTo[lados2,Intersection[{c2[[i]]→c2[[i+1]],
      c2[[i+1]]→c2[[i]]},F][[1]]];
,{i,Length[c2]-1}];
lados2
```

Out[]:= {3→2, 3→2, 3→2, 3→2 }

c)

```
In[]:=      lados3={};
Do[AppendTo[lados3,Intersection[{c3[[i]]→c3[[i+1]],
      c3[[i+1]]→c3[[i]]},F][[1]]];
,{i,Length[c3]-1}];
lados3
```

Out[] = {1 → 2, 3 → 2, 3 → 4, 3 → 4}

Igual que en el ejemplo 7.4. ahora aplicamos el programa 7.8. y determinaremos si son simples, análogamente hacemos para comprobar si son elementales, resultando C_1 simple y elemental, mientras que C_2 y C_3 no son simples.

Ahora estudiamos de qué tipo son los siguientes caminos dirigidos:

- a) $C_4 = \{1, 2, 4, 5, 3\};$
- b) $C_5 = \{5, 2, 4, 5\};$
- c) $C_6 = \{2, 4, 5, 1, 4, 5\};$

Los introducimos,

*In[] := c4={1,2,4,5,3};
c5={5,2,4,5};
c6={2,4,5,1,4,5};*

Comprobamos si son simples:

a)
*In[] := lados4=SUCLADOS[c4];
If[Length[Intersection[lados4,F]]==Length[lados4],
simple=True
,*
*simple=False];
simple*

Out[] = True

b)
*In[] := lados5=SUCLADOS[c5];
If[Length[Intersection[lados5,F]]==Length[lados5],
simple=True
,*
*simple=False];
simple*

Out[] = True

c)
*In[] := lados6=SUCLADOS[c6];
If[Length[Intersection[lados6,F]]==Length[lados6],
simple=True
,*
*simple=False];
simple*

Out[] = False

Ahora determinamos si C_4 y C_5 son elementales:

*In[] := If[c4[[1]] == c4[[Length[c4]]],
 long = Length[c4] - 1, long = Length[c4]];
If[Length[Intersection[c4, W]] == long,
 elemental = True, elemental = False];
elemental*

Out[] = True

*In[] := If[c5[[1]] == c5[[Length[c5]]],
 long = Length[c5] - 1, long = Length[c5]];
If[Length[Intersection[c5, W]] == long,
 elemental = True, elemental = False];
elemental*

Out[] = True □

2. TEOREMA DEL NÚMERO DE CAMINOS

Analizar computacionalmente el concepto de camino puede aparentar ser muy complicado, sin embargo, se dispone de una herramienta fundamental que, como veremos más adelante, proporcionará directamente la solución a muchos problemas. Podemos encontrar una demostración de este teorema en el epígrafe 6.5 del libro “Matemáticas Discretas” (Johnsonbaugh, R. [31]).

Teorema 7.1. Teorema del número de caminos. Sea $G = (W, F)$ un grafo (resp. grafo dirigido) con matriz de adyacencia A para $W = \{v_1, \dots, v_n\}$. Entonces el número de caminos (resp. caminos orientados) de longitud l que conectan los vértices v_i y v_j coincide con la coordenada (i, j) -ésima de A^l .

□

Son muchas las aplicaciones del teorema del número de caminos. Desde el punto de vista computacional este teorema constituye una herramienta eficaz, capaz de resolver multitud de problemas. En realidad, ya hemos usado el teorema del número de caminos en algunos de los métodos expuestos y lo usaremos más adelante en otros. Entre otras, éstas son algunas de sus aplicaciones más directas:

- i. Se ha usado para determinar el grado de cada vértice del grafo (función 6.2.). Obsérvese que la diagonal principal del cuadrado de la matriz de adyacencia, según el teorema, representaría el número de los ciclos de longitud 2 que existen para cada vértice, pero en un vértice existirán tantos ciclos de longitud 2 que empiecen y terminen en él como lados incidentes con él, esto es, la coordenada $(i,$

- i -ésima del cuadrado de la matriz de adyacencia es el grado del vértice v_i . (Véase el epígrafe 1 del capítulo 6).
- ii. Análogamente, podemos razonar que la coordenada (i, i) -ésima diagonal principal de la tercera potencia de la matriz de adyacencia es el doble del número de triángulo (ciclos de longitud 3) que contienen al vértice v_i .
 - iii. Podemos reconocer los grafos regular y completos comprobando la diagonal principal del cuadrado de la matriz de adyacencia (funciones 6.6.bis. y 6.8.). (Véase el epígrafe 2 del capítulo 6).
 - iv. Conseguiremos hallar la distancia entre dos vértices y el número de geodésicas (función 7.16.). (Véase el epígrafe 4 de este capítulo).
 - v. Permitirá determinar si un grafo no orientado es conexo o si un digrafo es conexo o fuertemente conexo y además calcular sus respectivas componentes conexas o fuertemente conexas (funciones 7.13., 7.14. y 7.15.). (Véase el epígrafe 3 de este capítulo).

Ejemplo 7.6. Dado el grafo dirigido, cuya matriz de adyacencia es:

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

Responder a las siguientes cuestiones:

- a) ¿Cuántos caminos orientados de longitud 3 existen?
- b) ¿Cuántos caminos de longitud 10 conectan los vértices v_1 y v_6 ?
- c) ¿Qué vértices pertenecen a ciclos orientados?

En primer lugar introducimos la matriz de adyacencia:

$$In[]:= \quad A=\{\{0,1,0,0,0,0\},\{0,0,1,0,0,1\},\{0,0,0,1,0,0\},\{0,0,0,0,1,0\},\\ \{1,0,0,0,0,1\},\{1,0,1,1,0,0\}\};$$

- a) Calculamos A^3 ,

$$In[]:= \quad \text{MatrixForm}[\text{MatrixPower}[A,3]]$$

$$Out[] =$$

$$\begin{pmatrix} 1 & 0 & 1 & 2 & 0 & 0 \\ 0 & 1 & 0 & 1 & 2 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 2 \end{pmatrix}$$

Sumamos todas las coordenadas:

```
In]:= B=MatrixPower[A,3];  
Sum[Sum[B[[i,j]],{i,6}],{j,6}]
```

Out]= 24

- b) Calculamos las coordenadas $(1, 6)$ -ésima y $(6, 1)$ -ésima de A^{10} ,

```
In]:= MatrixPower[A,10][[1,6]]  
MatrixPower[A,10][[6,1]]
```

Out]= 11
30

Por tanto existen 11 caminos orientados de v_1 a v_6 y 30 de v_6 a v_1 .

- c) Para comprobar qué vértices pertenecen a ciclos nos fijamos en las diagonales principales, por el apartado a) sabemos que v_1, v_2, v_4, v_5 y v_6 pertenecen a ciclos de longitud 3. Si calculamos A^l , las diagonales principales nos indicarán si existen ciclos que pasan por los vértices de longitud l .

Por ejemplo:

```
In]:= MatrixPower[A,4][[3,3]]
```

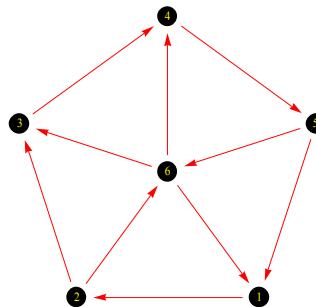
Out]= 1

Nos indicará que existe un ciclo de longitud 4 que pasa por el vértice v_3 . Por tanto, finalmente concluimos que todos los vértices pertenecen a ciclos.

Para terminar representamos gráficamente el grafo dirigido:

```
In]:= GraphPlot[A,VertexRenderingFunction->  
({Black,Disk[#,0.06],Yellow,Text[#2,#1]}&),  
EdgeRenderingFunction->  
({Red,Arrow[{#1,0.1}]&},BaseStyle->{FontSize->12})
```

Out]=



□

Conociendo la existencia de caminos de cierta longitud l que conecten dos vértices, el problema que nos planteamos ahora sería encontrar dichos caminos. Para ello partimos del vértice origen, vamos cogiendo lados o flechas hasta tener caminos de longitud l , y nos quedamos sólo con aquellos cuyo último lado o flecha termine en el vértice final. Lo implementamos en Mathematica, lo hacemos para caminos de grafos no orientados y para caminos orientados de grafos dirigidos (los caminos no orientados de grafos dirigidos se reducen al estudio de caminos de grafos no orientados como ya se ha comentado):

| FUNCIÓN | COMENTARIOS |
|--|---|
| <pre>CAMINOS[vertice1_,vertice2_,longitud_, matrizadyacencia_]:=Module[{i,ultvertex,caminos, adyacentes,caminostemp,k,j,t},</pre> | La función tendrá cuatro entradas, el inicio (“vertice1”) del camino, el fin (“vertice2”), la longitud (“longitud”) y el grafo que lo introducimos por su matriz de adyacencia (“matrizadyacencia”). Los vértices estarán identificados por sus subíndices. |
| <pre>caminos={{vertice1}}; Do[caminostemp={}; Do[adyacentes={}; ultvertex=caminos[[k]][[i]]; Do[If[matrizadyacencia[[ultvertex,j]]==1, AppendTo[adyacentes,j]]; ,{j,Dimensions[matrizadyacencia][[1]]}]; If[adyacentes!={}, Do[AppendTo[caminostemp, Append[caminos[[k]],adyacentes[[t]]]]; ,{t,Length[adyacentes]}];]; ,{k,Length[caminos]}]; caminos=caminostemp; ,{i,longitud}];</pre> | Progresivamente por cada lista de “caminos” (“caminos[[k]]”) añadimos nuevas listas (posibles caminos), una por cada vértice adyacente al último de la lista (“ultvertex”), repetimos el proceso tantas veces como indique “longitud”, de esta forma construimos todos los caminos de longitud “longitud” que empiezan en “vertice1”. |
| <pre>caminostemp={}; Do[If[caminos[[k]][[longitud+1]]==vertice2, AppendTo[caminostemp,caminos[[k]]]; ,{k,Length[caminos]}];</pre> | Nos quedamos sólo con aquellos caminos resultantes cuyo fin sea “vertice2”. |

| | |
|--------------------------|-----------------------|
| caminostemp]; | Salida de resultados. |
|--------------------------|-----------------------|

Función 7.12. Caminos de longitud l .

Ejemplo 7.7. Calculamos todos los caminos de longitud 4 que empiezan en el vértice 1 y terminan en el vértice 4 de $K_{3,4}$.

En primer lugar calculamos la matriz de adyacencia de $K_{3,4}$, usamos la función 6.13.:

In]:= BCadyacencia[n_,m_]:=Table[If[(i≤n &&j≤n) ||
(i>n &&j>n),0,1],{i,n+m},{j,n+m}];

In]:= A=BCadyacencia[3,4];

Out]= { { 0, 0, 0, 1, 1, 1, 1}, { 0, 0, 0, 1, 1, 1, 1},
{ 0, 0, 0, 1, 1, 1, 1}, { 1, 1, 1, 0, 0, 0, 0},
{ 1, 1, 1, 0, 0, 0, 0}, { 1, 1, 1, 0, 0, 0, 0},
{ 1, 1, 1, 0, 0, 0, 0} }

Comprobamos, con el teorema del número de caminos, el número de caminos que existen:

In]:= MatrixPower[matrizadyacencia,4][[1,3]]

Out]= 48

Calculamos todos los caminos con la función 7.12.:

In]:= CAMINOS[vertice1_,vertice2_,longitud_,
matrizadyacencia_]:=

⋮ ⋮

In]:= caminos=CAMINOS[1,3,4,A]

Out]= { { 1, 4, 1, 4, 3}, { 1, 4, 1, 5, 3}, { 1, 4, 1, 6, 3},
{ 1, 4, 1, 7, 3}, { 1, 4, 2, 4, 3}, { 1, 4, 2, 5, 3},
{ 1, 4, 2, 6, 3}, { 1, 4, 2, 7, 3}, { 1, 4, 3, 4, 3},
{ 1, 4, 3, 5, 3}, { 1, 4, 3, 6, 3}, { 1, 4, 3, 7, 3},
{ 1, 5, 1, 4, 3}, { 1, 5, 1, 5, 3}, { 1, 5, 1, 6, 3},
{ 1, 5, 1, 7, 3}, { 1, 5, 2, 4, 3}, { 1, 5, 2, 5, 3},
{ 1, 5, 2, 6, 3}, { 1, 5, 2, 7, 3}, { 1, 5, 3, 4, 3},
{ 1, 5, 3, 5, 3}, { 1, 5, 3, 6, 3}, { 1, 5, 3, 7, 3},
{ 1, 6, 1, 4, 3}, { 1, 6, 1, 5, 3}, { 1, 6, 1, 6, 3},
{ 1, 6, 1, 7, 3}, { 1, 6, 2, 4, 3}, { 1, 6, 2, 5, 3},
{ 1, 6, 2, 6, 3}, { 1, 6, 2, 7, 3}, { 1, 6, 3, 4, 3},
{ 1, 6, 3, 5, 3}, { 1, 6, 3, 6, 3}, { 1, 6, 3, 7, 3},
{ 1, 7, 1, 4, 3}, { 1, 7, 1, 5, 3}, { 1, 7, 1, 6, 3},
{ 1, 7, 1, 7, 3}, { 1, 7, 2, 4, 3}, { 1, 7, 2, 5, 3},

```
{1,7,2,6,3},{1,7,2,7,3},{1,7,3,4,3},
{1,7,3,5,3},{1,7,3,6,3},{1,7,3,7,3}}
```

Por último comprobamos el teorema del número de caminos:

In[]:= Length[caminos]

Out[]= 48

□

La misma función nos vale para caminos orientados de grafos dirigidos, pues como podemos comprobar en el código, la adyacencia de vértices se comprueba desde la matriz de adyacencia y ésta refleja la orientación de los caminos.

Ejemplo 7.8. Consideramos el grafo dirigido del ejemplo 5.5. e introducimos su matriz de adyacencia en ordenador:

*In[]:= matrizadyacencia={\{0,0,0,1,1\},\{1,0,0,1,0\},\{1,0,0,0,0\},
{0,0,1,0,1\},\{0,0,1,0,0\}};*

Calculamos todos los caminos de longitud 12 con origen en 2 y fin en 3 con la función 7.12..

*In[]:= CAMINOS[vertice1_,vertice2_,longitud_,
matrizadyacencia_]:=*

⋮ ⋮

In[]:= caminos=CAMINOS[2,3,12,matrizadyacencia]

Out[]= {{2,1,4,3,1,4,3,1,4,3,1,4,3}, {2,1,4,3,1,4,3,1,4,3,1,5,3}, {2,1,4,3,1,4,3,1,5,3,1,4,3}, {2,1,4,3,1,4,3,1,5,3,1,5,3}, {2,1,4,3,1,5,3,1,4,3,1,4,3}, {2,1,4,3,1,5,3,1,4,3,1,5,3}, {2,1,4,3,1,5,3,1,5,3,1,4,3}, {2,1,4,3,1,5,3,1,5,3,1,5,3}, {2,1,4,3,1,5,3,1,5,3,1,5,3}, {2,1,4,5,3,1,4,5,3,1,4,5,3}, {2,1,5,3,1,4,3,1,4,3,1,4,3}, {2,1,5,3,1,4,3,1,4,3,1,5,3}, {2,1,5,3,1,4,3,1,5,3,1,4,3}, {2,1,5,3,1,5,3,1,4,3,1,4,3}, {2,1,5,3,1,5,3,1,4,3,1,5,3}, {2,1,5,3,1,5,3,1,5,3,1,4,3}, {2,1,5,3,1,5,3,1,5,3,1,5,3}, {2,4,3,1,4,3,1,4,3,1,4,5,3}, {2,4,3,1,4,3,1,4,5,3,1,4,3}, {2,4,3,1,4,3,1,4,5,3,1,5,3}, {2,4,3,1,4,3,1,5,3,1,4,5,3}},

```
{2,4,3,1,4,5,3,1,4,3,1,4,3},  

{2,4,3,1,4,5,3,1,4,3,1,5,3},  

{2,4,3,1,4,5,3,1,5,3,1,4,3},  

{2,4,3,1,4,5,3,1,5,3,1,5,3},  

{2,4,3,1,4,5,3,1,4,3,1,4,5,3},  

{2,4,3,1,5,3,1,4,3,1,4,5,3},  

{2,4,3,1,5,3,1,4,5,3,1,4,3},  

{2,4,3,1,5,3,1,4,5,3,1,5,3},  

{2,4,3,1,5,3,1,5,3,1,4,5,3},  

{2,4,3,1,5,3,1,4,3,1,4,3},  

{2,4,5,3,1,4,3,1,4,3,1,4,3},  

{2,4,5,3,1,4,3,1,4,3,1,5,3},  

{2,4,5,3,1,4,3,1,5,3,1,4,3},  

{2,4,5,3,1,4,3,1,5,3,1,5,3},  

{2,4,5,3,1,4,3,1,4,3,1,4,3},  

{2,4,5,3,1,4,3,1,5,3,1,5,3},  

{2,4,5,3,1,5,3,1,4,3,1,4,3},  

{2,4,5,3,1,5,3,1,4,3,1,5,3},  

{2,4,5,3,1,5,3,1,5,3,1,4,3},  

{2,4,5,3,1,5,3,1,5,3,1,5,3}}
```

Por último comprobamos el teorema del número de caminos:

In[]:= Length[caminos]==MatrixPower[matrizadyacencia,12][[2,3]]

Out[] = True

□

También se podrían analizar los caminos simples y elementales (véase el ejercicio 7.4.).

3. GRAFOS CONEXOS Y COMPONENTES CONEXAS

Un grafo no orientado (resp. grafo dirigido) se dirá conexo (o débilmente conexo) si para cada par de vértices existe un camino (resp. camino no orientado) que los conecta. Un grafo dirigido se dirá que es fuertemente conexo si para cada par de vértices existe un camino orientado que los conecta para ambos sentidos, esto es, para cualesquier par de vértices existe un ciclo que pasa o incluye a ambos.

Los mayores subgrafos conexos de un grafo respecto de la inclusión se llamarán componentes conexas. Análogamente, los mayores subgrafos fuertemente conexos de un grafo dirigido se llamarán componentes fuertemente conexas.

3.1. GRAFOS CONEXOS. COMPONENTES CONEXAS

Dado un (p, q) -grafo (no dirigido) cuya matriz de adyacencia sea A , entre dos vértices conectados existirá al menos un camino que los conecta de longitud menor o igual que $p - 1$. Luego teniendo en cuenta el teorema del número de caminos, si pretendemos averiguar si dos vértices están o no conectados, bastará con ver si existe algún camino de cualquier longitud que los conecta, pero sabemos que de estar conectados, al menos existirá uno de longitud menor o igual que $p - 1$. Si calculamos,

$$B = A + A^2 + \dots + A^{(p-1)}$$

la coordenada (i, j) -ésima de B representará el número de caminos que conectan los vértices v_i y v_j de longitud menor o igual a $p - 1$, por tanto, si ambos vértices están conectados, esta coordenada será distinta de 0. Luego el grafo será conexo si y sólo si $b_{ij} \neq 0$ para cada $i \neq j$, con b_{ij} la coordenada (i, j) -ésima de B . Con Mathematica, primero calculamos B :

```
B=Sum[MatrixPower[A,j],{j,Dimensions[A][[1]]}];
```

Y después comprobamos si el grafo es conexo estudiando cada b_{ij} para cada $i \neq j$. En realidad bastaría con comprobarlo sólo para cada $i > j$ por la simetría de la matriz de adyacencia, sin embargo haremos el barrido completo (apenas supone tiempo de proceso) y así podremos aprovechar la misma función para digrafos. Lo integramos en una función:

| FUNCIÓN | COMENTARIOS |
|--|--|
| <code>CONEXO[A_]:=Module[{i,j,B},</code> | La función tendrá un único argumento, la matriz de adyacencia “A”. |
| <code>B=Sum[MatrixPower[A,j],{j,Dimensions[A][[1]]}];</code> | Calculamos la matriz B . |
| <code>conexo=True;</code> <code>Do[</code> <code> Do[If[i!=j && B[[i,j]]==0,conexo=False];</code> <code> ,{i,Dimensions[B][[1]]}];</code> <code> ,{j,Dimensions[B][[1]]}];</code> | Comprobamos si es conexo con el teorema del número de caminos. |
| <code>conexo</code> <code>]</code> | Salida de resultados. |

Función 7.13. Grafos conexos.

Si el grafo es dirigido, para comprobar si es conexo (débilmente), nos vale el mismo razonamiento, pues para este concepto no importa que sean lados o flechas, aunque tendríamos que considerar, en vez de la matriz de adyacencia A , la matriz $A + A^t$.

En el paquete `<<Combinatorica`` (versión 6 y siguientes de Mathematica), disponemos de la función:

ConnectedQ[G, opciones]

que comprueba si el grafo asociado al objeto de tipo grafo “G” es conexo, como opciones podremos indicar, sólo para grafos dirigidos: “Weak” o “Strong” (que analizaremos en esta sección más adelante), y si no indicamos ninguna opción funcionará como 7.13.

Ejemplo 7.9. Comprobamos que K_5 y $K_{3,3}$ son conexos.

Utilizaremos la función 7.13.

In[]:= **CONEXO[A_]:=Module[{i,j,B},**

⋮ ⋮

In[]:= **CONEXO[matrizadyacencia=Table[If[i==j,0,1],{i,5},{j,5}]]**

```

Out[7]=      True

In[7]:=      n=3;m=3;
A=Table[If[(i<=n && j<=n) ||
            (i>n && j>n),0,1],{i,n+m},{j,n+m}]
CONEXO[A]

```

Out[7]= True

Y con ConnectedQ[]:

```

In[7]:=      <<Combinatorica`

In[7]:=      ConnectedQ[FromAdjacencyMatrix[A]]

Out[7]=      True

```

□

Ejemplo 7.10. Comprobar si es débilmente conexo el grafo dirigido G cuya matriz de adyacencia es:

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

Introducimos la matriz de adyacencia, calculamos $A^t + A$ y usamos la función 7.13.

```

In[7]:=      A={\{0,1,0,0,0\},\{0,0,1,0,0\},\{1,0,0,0,0\},\{0,0,0,0,1\},\{0,0,0,0,0\}};

In[7]:=      A2=A+Transpose[A];

In[7]:=      CONEXO[A2]

Out[7]=      False

```

Y con ConnectedQ[] lo podremos hacer de dos formas:

```

In[7]:=      <<Combinatorica`

In[7]:=      ConnectedQ[FromAdjacencyMatrix[A2]]

Out[7]=      False

```

o bien con la opción “Weak”,

```
In]:= <<Combinatorica`  

In]:= ConnectedQ[FromAdjacencyMatrix[A,  

Type->Directed],Weak]  

Out]= False
```

□

Desde la matriz B también se puede determinar el número de componentes conexas del grafo y quiénes son estas componentes conexas, para ello analizamos qué vértices están conectados entre si y calculamos una partición de W , cuyos elementos serán los subconjuntos de vértices que inducen una componente conexa del grafo.

| FUNCIÓN | COMENTARIOS |
|---|--|
| <pre>COMPONENTESCONEXAS[matrizadyacencia_]:= Module[{j,B,conjunto,temp,i}, B=Sum[MatrixPower[matrizadyacencia,j] ,{j,Dimensions[matrizadyacencia][[1]]}]; componentesconexas={}; conjunto=Table[i,{i,Dimensions[B][[1]]}]; While[conjunto!={}, temp={conjunto[[1]]}; Do[If[B[[conjunto[[1]],conjunto[[i]]]]!=0, AppendTo[temp,conjunto[[i]]];] ,{i,2,Length[conjunto]}]; AppendTo[componentesconexas,temp]; conjunto=Complement[conjunto,temp];]; componentesconexas]</pre> | <p>La función tendrá como única entrada la matriz de adyacencia del grafo.</p> <p>Determinamos los subconjuntos de vértices conectados entre si.</p> |

Función 7.14. Componentes conexas.

En el paquete de Matemática Discreta de Mathematica, <<“GraphUtilities”>>; se dispone de la función:

WeakComponents[G]

que funciona sobre grafos no orientados de forma parecida a 7.14.

Y en el paquete <<Combinatorica` (versión 6 de Mathematica o posterior), disponemos de las funciones:

ConnectedComponents[G1] y WeaklyConnectedComponents[G2]

que calculan respectivamente las componentes conexas del grafo no orientado asociado al objeto de tipo grafo “G1” y del digrafo asociado al objeto de tipo grafo “G2”.

Ejemplo 7.11. Sea G un grafo cuya matriz de adyacencia es:

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

Calculamos sus componentes conexas con la función 7.14.:

In[1]:= **COMPONENTESCONEXAS[matrizadyacencia_]:=Module[**
 ⋮ ⋮

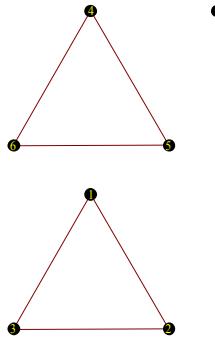
In[2]:= **A={{{0,1,1,0,0,0,0},{1,0,1,0,0,0,0},{1,1,0,0,0,0,0},{0,0,0,0,1,1,0},**
 {0,0,0,1,0,1,0},{0,0,0,1,1,0,0},{0,0,0,0,0,0,0}};
 COMPONENTESCONEXAS[A]

Out[2]= **{ {1, 2, 3}, {4, 5, 6}, {7} }**

Lo representamos gráficamente:

In[3]:= **GraphPlot[A,**
 VertexRenderingFunction->({Black,Disk[#,0.04],
 Yellow,Text[#2,#]}&),BaseStyle->{FontSize->12}]

Out[3]=



Y con WeakComponents[] y ConnectedComponents[] lo podremos hacer de dos formas:

In[4]:= **<<GraphUtilities`;**

```

In]:=          WeakComponents[A]
Out]= { {1,2,3}, {4,5,6}, {7} }

O bien:

In]:=      <<Combinatorica` 

In]:=      ConnectedComponents[FromAdjacencyMatrix[A]]
Out]= { {1,2,3}, {4,5,6}, {7} }

```

□

Ejemplo 7.12. Calcular las componentes conexas del digrafo del ejemplo 7.10.

Aplicamos la función 7.14. a la matriz A2 del ejemplo 7.10.

```

In]:=      COMPONENTESCONEXAS[matrizadyacencia_]:=Module[
          :
          :
In]:=      COMPONENTESCONEXAS[A2]
Out]= { {1,2,3}, {4,5} }

Y con WeaklyConnectedComponents[] lo podremos hacer de dos formas:

In]:=      <<Combinatorica` 

In]:=      WeaklyConnectedComponents[FromAdjacencyMatrix[A]]
Out]= { {1,2,3}, {4,5} }

```

□

3.2. GRAFOS DIRIGIDOS FUERTEMENTE CONEXOS

Por fortuna el teorema del número de caminos es válido también para grafos dirigidos, si bien, en este caso los caminos a los que se refiere son orientados. Por tanto analizaremos también la matriz B , que resultaba en el epígrafe anterior:

$$B = A + A^2 + \dots + A^{(p-1)}$$

y nos vale el mismo test para comprobar si es fuertemente conexo, que creamos para grafos no dirigidos, puesto que observemos que analiza todas las coordenadas de la matriz sin tener en cuenta la simetría del caso no orientado. Por tanto, comprobaremos si un grafo dirigido es fuertemente conexo usando 7.13.:

```
FUERTEMENTECONEXO[matrizadyacencia_]:=CONEXO[matrizadyacencia];
```

Por otra parte, para determinar las componentes fuertemente conexas, partiremos del mismo código que para caminos no orientados, pero tendremos cuidado y contemplaremos la posibilidad de estar conectados en un sentido y no en otro.

| FUNCIÓN | COMENTARIOS |
|---|---|
| COMPONENTESFUERTEMENTECONEXAS[matrizadyacencia_]:=Module[{j,B,conjunto,temp,i}, B=Sum[MatrixPower[matrizadyacencia,j] .,{j,Dimensions[matrizadyacencia][[1]]}]; | La función tiene un único argumento, la matriz de adyacencia del grafo. |
| componentesconexas={}; conjunto=Table[i,{i,Dimensions[B][[1]]}]; While[conjunto!={}, temp={conjunto[[1]]}; Do[If[B[[conjunto[[1]],conjunto[[i]]]]!=0 && B[[conjunto[[i]],conjunto[[1]]]]!=0, AppendTo[temp,conjunto[[i]]]; ,{i,2,Length[conjunto]}]; AppendTo[componentesconexas,temp]; conjunto=Complement[conjunto,temp];]; componentesconexas] | Calculamos B . |
| | Determinamos los subconjuntos de vértices conectados entre si. |
| | Salida de resultados. |

Función 7.15. Componentes fuertemente conexas.

En el paquete de Matemática Discreta de Mathematica, <<“GraphUtilities`”; se dispone de la función:

StrongComponents[GRAFO]

que funciona igual que la función 7.15. que acabamos de definir.

En el paquete <<Combinatorica` de la versión 6 o posterior de Mathematica, disponemos de las funciones:

ConnectedQ[G,Strong] y StronglyConnectedComponents[G]

la primera comprueba si el digrafo asociado al objeto de tipo grafo “G” es fuertemente conexo y la segunda calcula las componentes fuertemente conexas.

Ejemplo 7.13. Calcular las componentes fuertemente conexas de los grafos dirigidos cuyas matrices de adyacencia son:

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \quad \text{y} \quad B = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Introducimos ambas matrices en Mathematica:

```
In]:= A={{0,0,0,1,1},{1,0,0,1,0},{1,0,0,0,0},{0,0,1,0,1},{0,0,1,0,0}};
B={{0,1,0,0,1,0,0,0},{0,0,1,0,0,0,0,0},{0,0,0,1,0,0,0,0},
{0,0,0,0,0,0,0,0},{0,0,0,0,0,1,0,0},{1,1,0,0,0,0,0,0},
{0,0,0,0,1,0,0,0},{0,0,1,1,0,0,1,0}};
```

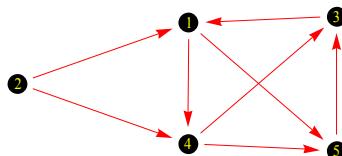
Calculamos las componentes fuertemente conexas con la función 7.15. y los representamos gráficamente:

```
In]:= COMPONENTESFUERTEMENTECONEXAS[
matrizadyacencia_]:=
```

⋮ ⋮

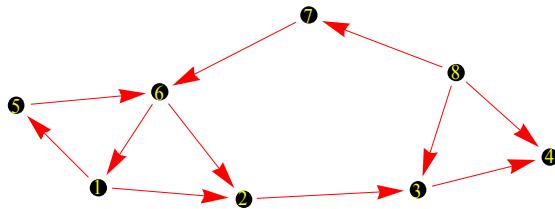
```
In]:= COMPONENTESFUERTEMENTECONEXAS[A]
GraphPlot[A,VertexRenderingFunction->({Black,
Disk[#,06],Yellow,Text[#2,#1]}&),
EdgeRenderingFunction->({Red,Arrow[#1,0.1]}&),
BaseStyle->{FontSize->12}]
```

```
Out]= { {1, 3, 4, 5}, {2} }
```



```
In]:= COMPONENTESFUERTEMENTECONEXAS[B]
GraphPlot[B,VertexRenderingFunction->({Black,
Disk[#,06],Yellow,Text[#2,#1]}&),
EdgeRenderingFunction->({Red,Arrow[#1,0.1]}&),
BaseStyle->{FontSize->10}]
```

Out[] = { {1, 5, 6}, {2}, {3}, {4}, {7}, {8} }



Igual podríamos hacerlo con `StrongComponents[]`:

```
In[]:= <<GraphUtilities`;  
In[]:= StrongComponents[B]
```

Out[] = { {4}, {3}, {2}, {1, 5, 6}, {7}, {8} }

Y con `StronglyConnectedComponents[]`:

```
In[]:= <<Combinatorica`  
In[]:= StronglyConnectedComponents[  
FromAdjacencyMatrix[B, Type -> Directed]]  
Out[] = { {4}, {3}, {2}, {1, 5, 6}, {7}, {8} }
```

□

4. DISTANCIA Y GEODÉSICAS

Sea G un grafo no orientado (resp. dirigido), llamaremos distancia entre dos vértices $d(v, u)$ a la longitud de cualquiera de los caminos (resp. caminos orientados) más cortos (de longitud mínima) que conectan v con u , a estos caminos, los de menor longitud, los llamaremos geodésicas.

Ambos conceptos pueden ser analizados también con el teorema del número de caminos, la distancia entre dos vértices conectados $d(v_i, v_j)$ se corresponde con el menor l tal que la coordenada (i, j) -ésima de A^l es distinta de cero, dicha coordenada indicará también el número de geodésicas que existen.

Podemos trasladar esto último directamente a Mathematica:

| FUNCTION | COMENTARIOS |
|--|--|
| <code>distancia[u_,v_,matrizadyacencia_]:=Module[{B},</code> | Funció n con tres entradas: dos vértices (identificados por sus subíndices) y la |

| | | |
|---|--|---------------------------------|
| | | matriz de adyacencia del grafo. |
| <pre>n=1;B=matrizadyacencia; While[B[[u,v]]==0 && n<Dimensions[B][[1]], n++; B=MatrixPower[matrizadyacencia,n];]; If[n==Dimensions[B][[1]], Print["No están conectados"]; Print[B[[u,v]]," geodésica/s. Distancia: ",n];];</pre> | Determinamos los subconjuntos de vértices conectados entre si. | |
| | | Salida de resultados. |

Función 7.16. Número de geodésicas y distancia.

También podemos usar las funciones del paquete de Matemática Discreta `<<GraphUtilities``:

GraphDistance[GRAFO,VÉRTICE_i,VÉRTICE_j]

y

GraphPath[GRAFO,VÉRTICE_i,VÉRTICE_j]

que calcularán la distancia y una geodésica entre los vértices “VÉRTICE_i” y “VÉRTICE_j”.

Ejemplo 7.14. Sea G el grafo cuya matriz de adyacencia es la matriz B del ejemplo anterior. Calcular las distancias y número de geodésicas existentes entre los vértices:

- a) 1 y 3.
- b) 8 y 2.
- c) 8 y 5.
- d) 3 y 1.

Definimos la función 7.16.:

*In[7]:= **distancia[u_,v_,matrizadyacencia_]:=Module[{B},***

⋮ ⋮

- a) *In[7]:= B={{{0,1,0,0,1,0,0,0},{0,0,1,0,0,0,0,0},{0,0,0,1,0,0,0,0},
{0,0,0,0,0,0,0,0},{0,0,0,0,0,1,0,0},{1,1,0,0,0,0,0,0},{0,0,0,0,1,0,0},{0,0,1,1,0,0,1,0}};*
- distancia[1,3,B]**

Out[7]= 1 geodésica/s. Distancia: 2

- b) *In[7]:= **distancia[8,2,B]***

Out[7]= 1 geodésica/s. Distancia: 3

c) $In[]:=$ **distancia[8,1,B]**

$Out[] =$ 1 geodésica/s. Distancia: 3

d) $In[]:=$ **distancia[3,1,B]**

$Out[] =$ No están conectados

Alternativamente podríamos usar:

$In[]:=$ <<“GraphUtilities`”;

$In[]:=$ **GraphDistance[B,3,1]**

$Out[] =$ ∞

Como los vértices 3 y 1 no están conectados, la función nos da como salida ∞ . □

Puesto que disponemos de herramientas para determinar los caminos que conectan dos vértices de longitud fija y la distancia, combinándolas podemos obtener fácilmente un nuevo método para calcular todas las geodésicas.

| FUNCIÓN | COMENTARIOS |
|--|--|
| CAMINOS[vertice1_,vertice2_,longitud_, matrizadyacencia_]:=... | Definimos la función 7.12. |
| GEODESICAS[u_,v_,matrizadyacencia_]:= Module[{B}, | Tendrá tres entradas, los dos vértices “u” y “v” (identificados por sus subíndices), entre los que buscamos geodésicas, y la matriz de adyacencia del grafo. |
| n=1; B=matrizadyacencia; While[B[[u,v]]==0 && n<Dimensions[B][[1]], n++; B=MatrixPower[matrizadyacencia,n];]; | Determinamos la distancia. |
| If[n==Dimensions[B][[1]], Print["No están conectados"]; , Print[CAMINOS[u,v,n,matrizadyacencia]]];] | Calculamos las geodésicas usando CAMINOS[] y las mostramos. |

Función 7.17. Geodésicas.

En el paquete <<Combinatorica` (versión 6 y siguientes de Mathematica), disponemos de la función:

ShortestPath[GRAFO,VÉRTICE_i,VÉRTICE_j]

que calcula una geodésica entre los vértices “VÉRTICE_i” y “VÉRTICE_j” del grafo asociado al objeto de tipo grafo “G”.

Ejemplo 7.15. Calculamos todas las geodésicas que parten del vértice 1 y llegan al vértice 2 del grafo $K_{3,4}$.

Definimos $K_{3,4}$ con la función 6.13.y usamos la función 7.17. para determinar las geodésicas que nos piden, previamente introducimos también 7.12.:

```
In]:= BCadyacencia[n_,m_]:=Table[If[(i≤n &&j≤n)||  
(i>n &&j>n),0,1],{i,n+m},{j,n+m}];  
  
In]:= CAMINOS[vertice1_,vertice2_,longitud_,  
matrizadyacencia_]:=  
⋮ ⋮  
  
In]:= GEODESICAS[u_,v_,matrizadyacencia_]:=Module[{B},  
⋮ ⋮  
  
In]:= A=BCadyacencia[3,4];  
  
In]:= GEODESICAS[1,2,A]  
  
Out]= { {1, 4, 2}, {1, 5, 2}, {1, 6, 2}, {1, 7, 2} }
```

Y con GraphPath[] y ShortestPath[]:

```
In]:= <<GraphUtilities`;  
  
In]:= GraphPath[A,1,2]  
  
Out]= {1, 4, 2}  
  
In]:= <<Combinatorica`  
  
In]:= ShortestPath[FromAdjacencyMatrix[A],1,2]  
  
Out]= {1, 7, 2}
```

□

5. GRAFOS DE EULER

Sea $G = (W, F)$ un grafo (resp. grafo dirigido) sin vértices aislados, un camino simple (resp. un camino orientado simple) que pase por todos los lados (resp. flechas) del grafo se dice de que es de Euler. Un camino (resp. orientado) de Euler cerrado, se dirá que es un ciclo de Euler o círculo de Euler. Un grafo que posea al menos un ciclo de Euler se llamará grafo de Euler.

Obsérvese que los grafos de Euler (resp. dígrafos de Euler) son conexos (resp. fuertemente conexos).

Los grafos de Euler son fácilmente detectables a partir de las siguientes caracterizaciones que podemos encontrar demostradas en el epígrafe 11.3 del libro “Matemáticas Discreta y Combinatoria” (Grimaldi, R.P., [29]).

Teorema 7.2. Sea $G = (W, F)$ un grafo (no dirigido) conexo, entonces G es de Euler si y sólo si el grado de cada vértice es par.

□

Teorema 7.3. Sea $G = (W, F)$ un grafo dirigido conexo, entonces G es de Euler si y sólo si el grado de entrada de cada vértice coincide con el de salida.

□

Con Mathematica nos limitaremos a trasladar estas caracterizaciones:

| FUNCIÓN | COMENTARIOS |
|--|--|
| <code>CONEXO[A_]:=...</code> | Definimos la función 7.13. |
| <code>EULER[matrizadyacencia_]:=Module[{A,i},</code> | La función tendrá por argumento la matriz de adyacencia del grafo. |
| <code>Euler=True;</code> <code>If[CONEXO[matrizadyacencia],</code> | Comprobamos que sea conexo. |
| <code>A=matrizadyacencia.matrizadyacencia;</code> <code>Do[</code> <code> If[Mod[A[[i,i]],2]==1,Euler=False;Break[],];</code> <code> ,{i,Length[matrizadyacencia]}];</code> <code>,</code> <code> Euler=False;</code> <code>];</code> | Utilizamos el teorema 7.2. |
| <code>Euler</code> <code>]</code> | Salida de resultados |

Función 7.18. Grafo de Euler no dirigido.

| FUNCIÓN | COMENTARIOS |
|--|--|
| <code>CONEXO[A_]:=...</code> | Definimos la función 7.13. |
| <code>FUERTEMENTECONEXO[A_]:=CONEXO[A];</code> | Definimos la función FUERTEMENTECONEXO[]. |
| <code>EULERDIR[matrizadyacencia_]:=</code> <code>Module[{i,gentrada,grsalida},</code> | La función tendrá por argumento la matriz de adyacencia del grafo. |
| <code>Euler=True;</code> <code>If[FUERTEMENTECONEXO[matrizadyacencia],</code> | |

| | |
|--|----------------------------|
| <pre> Do[grentrada=0;grsalida=0; Do[If[matrizadyacencia[[i,j]]==1,grsalida++]; If[matrizadyacencia[[j,i]]==1,grentrada++]; ,{j,Length[matrizadyacencia]}]; If[grentrada!=grsalida,Euler=False;Break[]]; ,{i,Length[matrizadyacencia]}]; , Euler=False;]; Euler] </pre> | Utilizamos el teorema 7.3. |
| | Salida de resultados |

Función 7.19. Grafo de Euler dirigido.

En el paquete `Combinatoria` (versión 6 de Mathematica o posterior), disponemos de la función:

EulerianQ[G]

que comprueba si el grafo asociado al objeto de tipo grafo “G” es de Euler.

Ejemplo 7.16. Determinar cuáles de los siguientes grafos y digrafos son de Euler:

a)

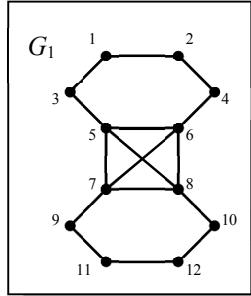


Ilustración 7.1.

b)

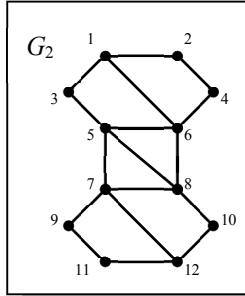


Ilustración 7.2.

c)

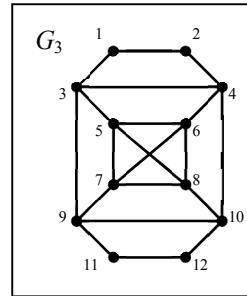


Ilustración 7.3.

d)

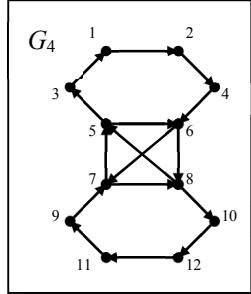


Ilustración 7.4.

e)

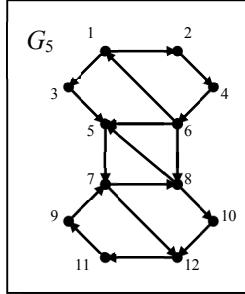


Ilustración 7.5.

f)

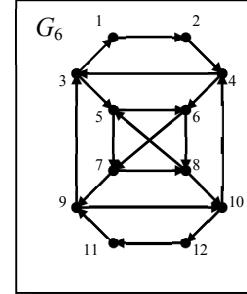


Ilustración 7.6.

Calculamos sus matrices de adyacencia y las introducimos en Mathematica:

```
In[]:= A1={{0,1,1,0,0,0,0,0,0,0,0},{1,0,0,1,0,0,0,0,0,0,0},  

{1,0,0,0,1,0,0,0,0,0,0},{0,1,0,0,0,1,0,0,0,0,0},  

{0,0,1,0,0,1,1,1,0,0,0},{0,0,0,1,1,0,1,1,0,0,0},  

{0,0,0,0,1,1,0,1,1,0,0},{0,0,0,0,1,1,1,0,0,1,0},  

{0,0,0,0,0,1,0,0,0,1,0},{0,0,0,0,0,0,1,0,0,0,1},  

{0,0,0,0,0,0,0,1,0,0,1},{0,0,0,0,0,0,0,0,1,1,0}};  

A2={{0,1,1,0,0,1,0,0,0,0,0},{1,0,0,1,0,0,0,0,0,0,0},  

{1,0,0,0,1,0,0,0,0,0,0},{0,1,0,0,0,1,0,0,0,0,0},  

{0,0,1,0,0,1,1,1,0,0,0},{1,0,0,1,1,0,0,1,0,0,0},  

{0,0,0,0,1,0,0,1,1,0,0,1},{0,0,0,0,1,1,1,0,0,1,0},  

{0,0,0,0,0,0,1,0,0,0,1,0},{0,0,0,0,0,0,0,1,0,0,1},  

{0,0,0,0,0,0,0,1,0,0,1},{0,0,0,0,0,0,1,0,1,1,0}};  

A3={{0,1,1,0,0,0,0,0,0,0,0},{1,0,0,1,0,0,0,0,0,0,0},  

{1,0,0,1,1,0,0,0,1,0,0,0},{0,1,1,0,0,1,0,0,0,1,0},  

{0,0,1,0,0,1,1,1,0,0,0},{0,0,0,1,1,0,1,1,0,0,0},  

{0,0,0,0,1,1,0,1,1,0,0,0},{0,0,0,0,1,1,1,0,0,1,0},  

{0,0,1,0,0,0,1,0,0,1,0},{0,0,0,1,0,0,0,1,1,0,0,1},  

{0,0,0,0,0,0,0,1,0,0,1},{0,0,0,0,0,0,0,0,1,1,0}};  

A4={{0,1,0,0,0,0,0,0,0,0,0},{0,0,0,1,0,0,0,0,0,0,0},  

{1,0,0,0,0,0,0,0,0,0,0},{0,0,0,0,0,1,0,0,0,0,0},  

{0,0,1,0,0,1,0,0,0,0,0},{0,0,0,0,0,0,1,1,0,0,0},  

{0,0,0,0,1,0,0,1,0,0,0},{0,0,0,0,1,0,0,0,0,1,0},  

{0,0,0,0,0,0,1,0,0,0,0},{0,0,0,0,0,0,0,0,0,0,1},  

{0,0,0,0,0,0,0,0,1,0,0},{0,0,0,0,0,0,0,0,0,1,0}};  

A5={{0,1,1,0,0,0,0,0,0,0,0},{0,0,0,1,0,0,0,0,0,0,0},  

{0,0,0,0,1,0,0,0,0,0,0},{0,0,0,0,0,1,0,0,0,0,0},  

{0,0,0,0,0,0,1,1,0,0,0},{1,0,0,0,1,0,0,0,0,0,0},  

{0,0,0,0,0,0,0,1,0,0,1},{0,0,0,0,1,0,0,0,0,1,0},  

{0,0,0,0,0,0,1,0,0,0,0},{0,0,0,0,0,0,0,0,0,0,1},  

{0,0,0,0,0,0,0,0,1,0,0},{0,0,0,0,0,0,0,0,0,1,0}};  

A6={{0,1,0,0,0,0,0,0,0,0,0},{0,0,0,1,0,0,0,0,0,0,0},  

{1,0,0,0,1,0,0,0,0,0,0},{0,0,1,0,0,1,0,0,0,0,0},  

{0,0,0,0,0,1,1,0,0,0,0},{0,0,0,0,0,0,1,1,0,0,0},  

{0,0,0,0,0,0,0,1,1,0,0,0},{0,0,0,0,1,0,0,0,0,1,0},  

{0,0,1,0,0,0,0,0,0,1,0,0},{0,0,0,1,0,0,0,0,0,0,1},  

{0,0,0,0,0,0,0,0,1,0,0},{0,0,0,0,0,0,0,0,0,1,0}};
```

Comprobamos cuáles de ellos son de Euler, para los grafos no orientados usamos la función 7.18., previamente definimos la función 7.13.:

```
In[]:= CONEXO[A_]:=Module[{i,j,B},
```

```
⋮ ⋮
```

```
In[]:= EULER[matrizadyacencia_]:=Module[{A,i},
```

```
⋮ ⋮
```

```
In[]:= EULER[A1]
```

```

Out[]:=      True
In[]:=      EULER[A2]
Out[]:=      False
In[]:=      EULER[A3]
Out[]:=      True

```

Y para los orientados la función 7.19.:

```

In[]:=      FUERTEMENTECONEXO[A_]:=CONEXO[A_];
In[]:=      EULERDIR[matrizadyacencia_]:=
            Module[{i,grentrada,grsalida},

```

⋮ ⋮

```

In[]:=      EULERDIR[A4]

```

```

Out[]:=      True

```

```

In[]:=      EULERDIR[A5]

```

```

Out[]:=      False

```

```

In[]:=      EULERDIR[A6]

```

```

Out[]:=      True

```

Y con EulerianQ[]:

```

In[]:=      <<Combinatorica`  

In[]:=      EulerianQ[FromAdjacencyMatrix[A1]]  

          EulerianQ[FromAdjacencyMatrix[A2]]  

          EulerianQ[FromAdjacencyMatrix[A3]]  

          EulerianQ[FromAdjacencyMatrix[A4,Type->Directed]]  

          EulerianQ[FromAdjacencyMatrix[A5,Type->Directed]]  

          EulerianQ[FromAdjacencyMatrix[A6,Type->Directed]]

```

```

Out[]=
      True
      False
      True
      True
      False
      True

```

□

Un interesante problema que podemos analizar, consistiría en encontrar un ciclo de Euler en un grafo de Euler, lo resolvemos para grafos no orientados, para ello vamos a utilizar el siguiente resultado que es fácil de demostrar:

Teorema 7.4. Sea $G = (W, F)$ un grafo no orientado y conexo, entonces G es de Euler si y sólo si existe una partición de F formada por subconjuntos de lados que forman ciclos elementales.

□

Para implementarlo en Mathematica, seguiremos el siguiente proceso:

- Elegimos un vértice al azar, por ejemplo v_1 , buscamos un ciclo c_1 que contenga a v_1 , donde no se repitan lados y eliminamos todos los lados que hemos usado en el ciclo c_1 del conjunto F , llamando al conjunto de lados resultante F_1 .
- Elegimos otro vértice v_2 , también al azar, pero de forma tal que exista algún lado en F_1 incidente en él. Buscamos un nuevo ciclo c_2 que contenga a v_2 formado por lados de F_1 y donde no se repitan lados, eliminamos todos los lados usados en c_2 de F_1 quedando el conjunto de lados F_2 .
- Reiteramos el proceso hasta un paso $(n + 1)$ -ésimo donde el conjunto resultante F_{n+1} sea el vacío.
- Cogemos todos los ciclos resultantes: c_1, c_2, \dots, c_n , y si dos de ellos tienen algún vértice en común, intercalamos uno en otro, resultando un nuevo ciclo más largo, por ejemplo si $c_i = \{v_{i1}, v_{i2}, \dots, v_{ik}, \dots, v_{i1}\}$ y $c_j = \{v_{j1}, v_{j2}, \dots, v_{jh}, \dots, v_{j1}\}$ con $v_{ik} = v_{jh}$, creamos un nuevo ciclo:

$$\{v_{i1}, v_{i2}, \dots, v_{ik} = v_{jh}, v_{j(h+1)}, \dots, v_{j1}, v_{j2}, \dots, v_{jh} = v_{ik}, \dots, v_{i1}\}$$

- Reiteramos la operación hasta que sólo quede un ciclo, que será el ciclo de Euler que caracteriza al grafo.

Con Mathematica (representaremos los caminos como listas de vértices):

| PROGRAMA | COMENTARIOS |
|--|--|
| F = CONJUNTO DE LADOS DE UN GRAFO NO ORIENTADO DE EULER <pre> tempF=F; listaciclos={}; While[tempF!={}, camino={}; inicio=tempF[[1,1]]; otro=tempF[[1,2]]; AppendTo[camino,tempF[[1,1]]]; AppendTo[camino,tempF[[1,2]]]; tempF=Complement[tempF,{tempF[[1]]}]; While[otro!=inicio, j=1; ultimo=Last[camino]; While[tempF[[j,1]]!=ultimo && tempF[[j,2]]!= </pre> | Introducimos el conjunto de lados de un grafo de Euler. |
| | Construimos una partición de F formada por subconjuntos de lados que forman un ciclo (no necesariamente simple) pero sin que repita ningún lado. |

| | |
|--|--|
| <pre> ultimo ,j++]; otro=Complement[{tempF[[j]][[1]], tempF[[j]][[2]]},{ultimo}][[1]]; AppendTo[camino,otro]; tempF=Complement[tempF,{tempF[[j]]}];]; AppendTo[listaciclos,camino];]; While[Length[listaciclos]!=1, i=2; While[Intersection[listaciclos[[1]],listaciclos[[i]]]=={}, i++]; opcion=Intersection[listaciclos[[1]], listaciclos[[i]][[1]]]; k1=Position[listaciclos[[1]],opcion][[1]][[1]]; k2=Position[listaciclos[[i]],opcion][[1]][[1]]; nuevociclo=listaciclos[[1]]; t=1; Do[nuevociclo=Insert[nuevociclo, listaciclos[[i]][[cont]],k1+t];t++; ,{cont,k2+1,Length[listaciclos[[i]]]}]; Do[nuevociclo=Insert[nuevociclo, listaciclos[[i]][[cont]],k1+t]; t++; ,{cont,2,k2}]; listaciclos=Delete[listaciclos,i]; listaciclos=Delete[listaciclos,1]; AppendTo[listaciclos,nuevociclo];]; Nuevociclo </pre> | Con la partición anterior construimos el ciclo de Euler. |
| Nuevociclo | Mostramos el ciclo de Euler calculado. |

Programa 7.20. Ciclo de Euler.

En el paquete `Combinatorica` de la versión 6 de Mathematica, disponemos de la función:

EulerianCycle[G]

que calcula un ciclo de Euler del grafo asociado al objeto de tipo grafo “G”.

Ejemplo 7.17. Determinar ciclos de Euler para todos los grafos de Euler no orientados del ejemplo 7.15.

Los grafos de Euler no orientados son el a) y el c),

a)

```

In[]:=      matrizadyacencia=A1;
F={};
Do[Do[If[matrizadyacencia[[i,j]]==1,AppendTo[F,i→j]],[i,j]];
,{j,Dimensions[matrizadyacencia][[1]]}];F

```

Out[] = {1→2, 1→3, 2→4, 3→5, 4→6, 5→6, 5→7, 6→7, 5→8, 6→8, 7→8, 7→9, 8→10, 9→11, 10→12, 11→12}

Con el programa 7.20.:

In[] := tempF=F;

⋮ ⋮

Out[] = {7, 6, 8, 5, 3, 1, 2, 4, 6, 5, 7, 8, 10, 12, 11, 9, 7}

c)

In[] := matrizadyacencia=A3;
F={};
Do[Do[If[matrizadyacencia[[i,j]]==1,AppendTo[F,i→j]],{i,j}];
,{j,Dimensions[matrizadyacencia][[1]]}];
F

Out[] = {1→2, 1→3, 2→4, 3→5, 4→6, 5→6, 5→7, 6→7, 5→8, 6→8, 7→8, 7→9, 8→9, 8→10, 9→11, 10→12, 11→12}

Usamos el programa 7.20.:

In[] := tempF=F;

⋮ ⋮

Out[] = {9, 3, 1, 2, 4, 3, 5, 6, 4, 10, 8, 5, 7, 6, 8, 7, 9, 10, 12, 11, 9}

Y con EulerianCycle[]:

In[] := <<Combinatorica`

In[] := EulerianCycle[FromAdjacencyMatrix[A3]]

Out[] = {10, 8, 5, 6, 4, 3, 5, 7, 6, 8, 7, 9, 10, 12, 11, 9, 3, 1, 2, 4, 10} □

También podríamos analizar el cálculo de un ciclo de Euler haciendo un barrido sobre los caminos, de esta forma podríamos incluso determinar todos los ciclos de Euler de un grafo, si bien, el procedimiento sería muy ineficaz (véase ejercicio 7.15.), este procedimiento además también funcionaría con digrafos.

Ejemplo 7.18. Representar gráficamente todos los grafos no dirigidos de Euler de 7 vértices distintos, salvo isomorfismo.

Usando las funciones 5.19., 5.20. (consideraremos las diagonales principales de las potencias segunda, tercera y cuarta para comprobar si dos matrices de adyacencia

pertenecen a la misma clase de isomorfía) y el programa 5.18. calculamos las 1044 clases de isomorfía de matrices de adyacencia de grafos de 7 vértices.

```
In]:=          AñadirLado2[matrizadyacencia_]:=Module[{i,j,matriz},
                                              :
                                              :
                                              ]
```

```
In]:=          QUITARISOMORFOS2[listamatrices_]:=Module[
                                              :
                                              :
                                              ]
```

```
In]:=          n=7;
nLados=21;
matrizadyacencia=Table[0,{i,n},{j,n}];

                                              :
                                              :
```

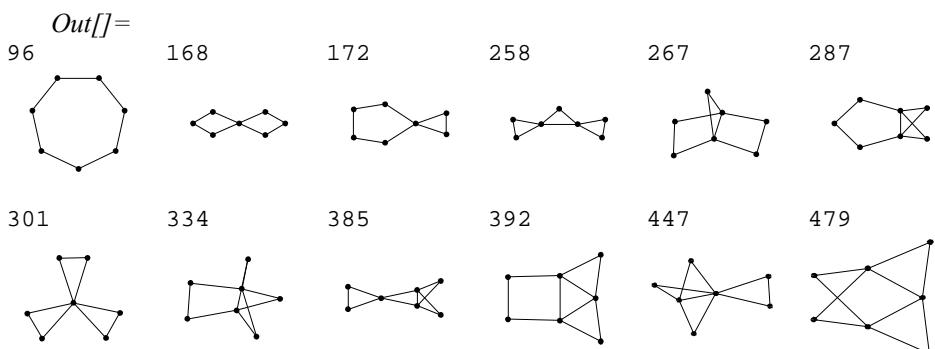
```
Out]=          :
                                              :
                                              :
```

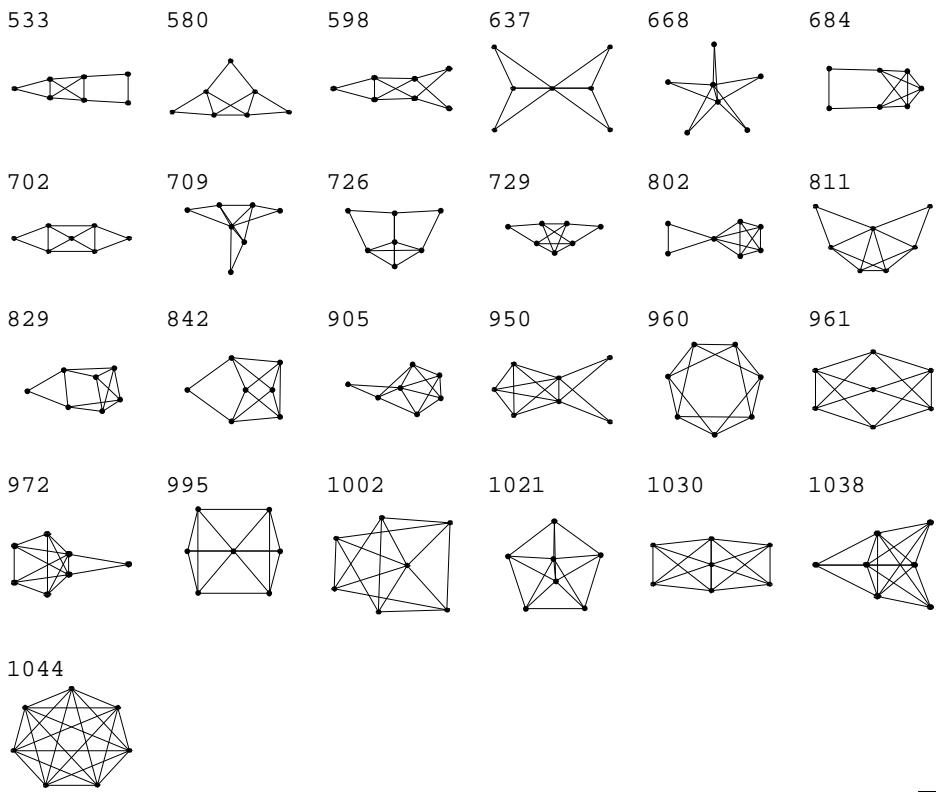
Totales: 1044 Tiempo empleado: 5.765

Ahora comprobamos cuáles de ellos son de Euler con la función 7.18. y los representamos gráficamente:

```
In]:=          EULER[matrizadyacencia_]:=Module[{A,i},
                                              :
                                              :
```

```
In]:=          Do[
If[EULER[listamatrices1[[i]]],
Print[i];
Print[GraphPlot[listamatrices1[[i]],
PlotStyle→{Black,PointSize[.05]}]];
,{i,Length[listamatrices1]}]
```





6. GRAFOS DE HAMILTON

Sea $G = (W, F)$ un grafo con más de tres vértices (resp. grafo dirigido), un camino elemental (resp. un camino orientado elemental) que pase por todos los vértices del grafo, se dice de que es de Hamilton. Un camino (resp. orientado) de Hamilton cerrado, se dirá que es un ciclo de Hamilton o círculo de Hamilton. Un grafo que posea al menos un ciclo de Hamilton se llamará grafo de Hamilton.

Desafortunadamente no existen caracterizaciones de los grafos de Hamilton como ocurre con los de Euler, por lo que su estudio con o sin Mathematica se limitará al uso de algunas condiciones suficientes o necesarias. A continuación enumeramos algunas:

- Sea $G = (W, F)$ un (n, m) -grafo (no digrafo) conexo con más de tres vértices tal que $gr(v) \geq n/2$ para cada $v \in W$, entonces G es un grafo de Hamilton.
- Si un grafo posee un vértice que sea articulación (ver ejercicio 7.1.), entonces no es de Hamilton.
- Sea $G = (W, F)$ un (n, m) -grafo (no digrafo) conexo con más de tres vértices tal que $gr(v) + gr(w) \geq n$ para cada $v, w \in W$ dos vértices no adyacentes, entonces G es un grafo de Hamilton.

- iv. Sea $G = (W, F)$ un (n, m) -grafo (no digrafo) conexo con más de tres vértices tal que $m \geq (n-1)(n-2)/2 + 2$, entonces G es un grafo de Hamilton.

Estas condiciones pueden trasladarse a Mathematica rápidamente (ver ejercicio 7.21.), pero sólo resolverán algunos grafos, en muchos casos, no aclararán si el grafo es de Hamilton.

Podemos crear un algoritmo para buscar todos los ciclos de Hamilton, si existen, de un grafo; evidentemente, sólo es operativo para grafos pequeños en cuanto al número de vértices y lados, ya que su funcionamiento se basa en realizar un barrido de todos los posibles caminos elementales que sean ciclos y con longitud el número de vértices del grafo, si el grafo es grande en cuanto al número de vértices y lados necesitará una elevada cantidad de proceso, nos limitamos al caso no orientado, dejando propuesta la implementación de una rutina similar para grafos dirigidos en el ejercicio 7.30.

| FUNCIÓN | COMENTARIOS |
|--|---|
| HAMILTON[A]:=Module[{ciclos,ultvertice,ciclostemp,adyacentes,k,t,j,i}, | Como única entrada la función tendrá la matriz de adyacencia "A" del grafo. |
| ciclos={{1}}; Do[ciclostemp={}; Do[adyacentes={}; ultvertice=ciclos[[k]][[i]]; Do[If[A[[ultvertice,j]]==1,AppendTo[adyacentes,j]]; ,{j,Dimensions[A][[1]]}; adyacentes=Complement[adyacentes,ciclos[[k]]]; If[adyacentes!={}, Do[AppendTo[ciclostemp, Append[ciclos[[k]],adyacentes[[t]]]; ,{t,Length[adyacentes]}];]; ,{k,Length[ciclos]}]; ciclos=ciclostemp; ,{i,Dimensions[A][[1]]-1}; ciclostemp={}; | Calculamos todos los caminos simples que empiezan en 1 de longitud el orden de la matriz "A", esto es, el número de vértices del grafo. |
| Do[If[A[[1,ciclos[[k]][[Dimensions[A][[1]]]]]]==1, AppendTo[ciclostemp,Append[ciclos[[k]],1]]; ,{k,Length[ciclos]}]; | Nos quedamos sólo con los caminos que terminan en vértices adyacentes al vértice 1. |
| ciclostemp]; | Salida de resultados. |

Función 7.21. Grafo de Hamilton no dirigido.

La función anterior determina todos los ciclos de Hamilton si existen de grafo que empiezan y terminan en el primer vértice, para usarlo como test bastaría con comprobar que dicha lista de ciclos de Hamilton es el vacío. Cualquier otro ciclo de Hamilton será uno de los obtenidos por 7.21., salvo por el vértice inicial, esto es, tendrá el mismo recorrido pero comenzando y terminando en otro vértice intermedio del ciclo.

En el paquete `Combinatorica` de la versión 6 y siguientes de Mathematica, disponemos de las funciones:

HamiltonianQ[G] y **HamiltonianCycle[G,opciones]**

la primera comprueba si el grafo asociado al objeto de tipo grafo “G” es de Hamilton y la segunda calcula un ciclo de Hamilton, y con la opción “All” calculará todos los ciclos de Hamilton.

Ejemplo 7.19. Determinar cuáles de los grafos no dirigidos del ejemplo 7.15. son de Hamilton y calcular algún ciclo de Hamilton si existe.

Utilizamos la función 7.21. y con ello determinaremos también si existen ciclos de Hamilton:

```
In]:= HAMILTON[A]:=Module[
          {ciclos,ultvertice,ciclostemp,adyacentes,k,t,j,i},
          ⋮ ⋮
In]:= HAMILTON[A1]
Out]= { {1,2,4,6,7,9,11,12,10,8,5,3,1},
         {1,2,4,6,8,10,12,11,9,7,5,3,1},
         {1,3,5,7,9,11,12,10,8,6,4,2,1},
         {1,3,5,8,10,12,11,9,7,6,4,2,1} }

In]:= HAMILTON[A2]
Out]= { {1,2,4,6,8,10,12,11,9,7,5,3,1},
         {1,3,5,7,9,11,12,10,8,6,4,2,1} }

In]:= HAMILTON[A3]
Out]= { {1,2,4,6,5,7,8,10,12,11,9,3,1},
         {1,2,4,6,7,5,8,10,12,11,9,3,1},
         {1,2,4,6,7,9,11,12,10,8,5,3,1},
         {1,2,4,6,8,10,12,11,9,7,5,3,1},
         {1,2,4,10,12,11,9,7,6,8,5,3,1},
         {1,2,4,10,12,11,9,7,8,6,5,3,1},
         {1,3,5,6,8,7,9,11,12,10,4,2,1},
         {1,3,5,7,9,11,12,10,8,6,4,2,1},
         {1,3,5,8,6,7,9,11,12,10,4,2,1},
         {1,3,5,8,10,12,11,9,7,6,4,2,1},
         {1,3,9,11,12,10,8,5,7,6,4,2,1},
         {1,3,9,11,12,10,8,7,5,6,4,2,1} }
```

Y si usamos **HamiltonianQ[]** y **HamiltonianCycle[]**:

```
In]:= <<Combinatorica`
```

```
In]:= HamiltonianQ[FromAdjacencyMatrix[A1]]  

HamiltonianCycle[FromAdjacencyMatrix[A2]]  

HamiltonianCycle[FromAdjacencyMatrix[A3],All]  

Out]= True  

{1,2,4,6,8,10,12,11,9,7,5,3,1}  

{{1,2,4,6,5,7,8,10,12,11,9,3,1},  

 {1,2,4,6,7,5,8,10,12,11,9,3,1},  

 {1,2,4,6,7,9,11,12,10,8,5,3,1},  

 {1,2,4,6,8,10,12,11,9,7,5,3,1},  

 {1,2,4,10,12,11,9,7,6,8,5,3,1},  

 {1,2,4,10,12,11,9,7,8,6,5,3,1},  

 {1,3,5,6,8,7,9,11,12,10,4,2,1},  

 {1,3,5,7,9,11,12,10,8,6,4,2,1},  

 {1,3,5,8,6,7,9,11,12,10,4,2,1},  

 {1,3,5,8,10,12,11,9,7,6,4,2,1},  

 {1,3,9,11,12,10,8,5,7,6,4,2,1},  

 {1,3,9,11,12,10,8,7,5,6,4,2,1}}
```

□

Ejemplo 7.20. Representar gráficamente todos los grafos no dirigidos de Hamilton de 6 vértices distintos, salvo isomorfismo.

Usando las funciones 5.19., 5.20. (consideraremos las diagonales principales de las potencias segunda, tercera y cuarta para comprobar si dos matrices de adyacencia pertenecen a la misma clase de isomorfía) y el programa 5.18. calculamos las 156 clases de isomorfía de matrices de adyacencia de grafos de 6 vértices.

```
In]:= AñadirLado2[matrizadyacencia_]:=Module[{i,j,matriz},  

 $\vdots$                     $\vdots$   

In]:= QUITARISOMORFOS2[listamatrices_]:=Module[  

 $\vdots$                     $\vdots$   

In]:= n=6;  

nlados=15;  

matrizadyacencia=Table[0,{i,n},{j,n}];  

 $\vdots$                     $\vdots$   

Out]=  

 $\vdots$                     $\vdots$   

Totales: 156 Tiempo empleado: 0.219
```

Ahora comprobamos cuáles de ellos son de Hamilton con la función 7.21. y los representamos gráficamente:

In[]:=

```
HAMILTON[A_]:=Module[
{ciclos,ultvertice,ciclostemp,adyacentes,k,t,j,i},
```

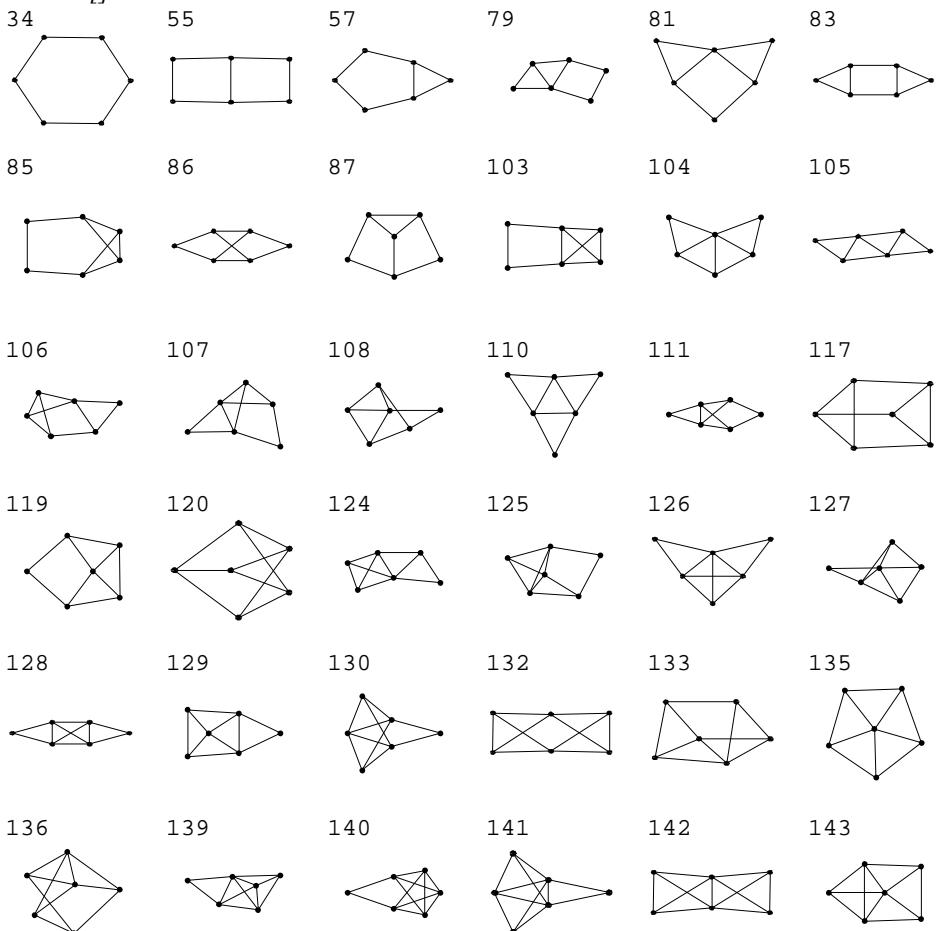
⋮

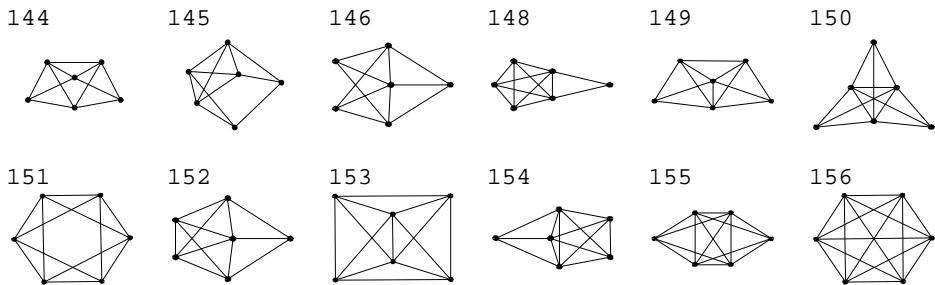
⋮

In[]:=

```
Do[
If[HAMILTON[listamatrices1[[i]]]!={},
Print[i];
Print[GraphPlot[listamatrices1[[i]],
PlotStyle→{Black,PointSize[.05]}]];
,{i,Length[listamatrices1]}]
```

Out[] =





Hemos calculado todos los grafos de 6 vértices y después hemos comprobado cuáles de ellos son de Hamilton. Podríamos haberlo hecho directamente de otra forma, puesto que somos capaces de determinar todos los grafos que contienen a uno dado, y como un grafo de 6 vértices será de Hamilton si y sólo si contiene a un ciclo de Hamilton, esto es, si contiene al hexágono, entonces simplemente podríamos haberlos determinado con el programa 5.18.:

```
In[]:= n=6; nlados=9; matrizadyacencia={{0,1,0,0,0,1},{1,0,1,0,0,0},{0,1,0,1,0,0}, {0,0,1,0,1,0},{0,0,0,1,0,1},{1,0,0,0,1,0}};
```

⋮ ⋮

```
Out[]=
```

⋮ ⋮

Totales: 48 Tiempo empleado: 0.031

Y en “listamatrices1” tendríamos almacenados los 48 grafos de Hamilton. \square

Como se ha comprobado en el ejemplo anterior el problema combinatorio en relación a los grafos de Hamilton lo tenemos resuelto. Los grafos de Hamilton de p vértices serán aquellos que contengan como subgrafo al polígono de p lados, y con el programa 5.18. el problema se resuelve directamente.

7. OTROS EJEMPLOS

Concluimos el capítulo con otros ejemplos y aplicaciones de algunas de las propiedades más interesantes que hemos visto.

Ejemplo 7.21. Comprobar si el dodecaedro es de Hamilton y representar gráficamente todos los ciclos de Hamilton.

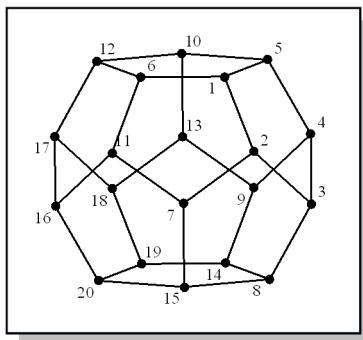


Ilustración 7.7. Dodecaedro.

Introducimos la matriz de adyacencia:

```
In]:= A={ {0,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0},  
      {1,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0},  
      {1,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0},  
      {0,0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0},  
      {0,1,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0},  
      {0,0,1,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0},  
      {0,0,0,0,0,1,0,1,0,0,0,1,0,0,0,0,0,0},  
      {0,0,0,1,0,0,1,0,1,0,0,0,0,0,0,0,0,0},  
      {0,0,0,0,1,0,0,1,0,0,0,1,0,0,0,0,0,0},  
      {0,0,0,0,0,0,1,0,1,0,0,0,0,1,0,0,0,0},  
      {0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0},  
      {0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0},  
      {1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1},  
      {0,0,0,0,0,1,0,0,0,1,0,0,0,0,1,0,0,0},  
      {0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,1,0,0},  
      {0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,1,0},  
      {0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,0},  
      {0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0},  
      {0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1},  
      {0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,1},  
      {0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,0,1} };
```

Definimos la función 7.21. y la aplicamos a la matriz A :

```
In]:= HAMILTON[A_]:=Module[  
  {ciclos,ultvertice,ciclostemp,adyacentes,k,t,j,i},  
   :  
   :
```

```
In]:= HAMILTON[A]
```

```
Out]= {{1,2,5,4,3,6,7,8,9,10,15,18,20,19,17,14,13,16,12,11,1},  
       {1,2,5,4,3,6,12,16,19,17,14,13,7,8,9,10,15,18,20,11,1},
```

```
{1,2,5,4,8,7,13,16,19,17,14,9,10,15,18,20,11,12,6,3,1},
{1,2,5,4,8,9,10,15,18,20,11,12,16,19,17,14,13,7,6,3,1},
{1,2,5,10,9,8,4,3,6,7,13,14,17,15,18,20,19,16,12,11,1},
{1,2,5,10,9,14,13,7,8,4,3,6,12,16,19,17,15,18,20,11,1},
```

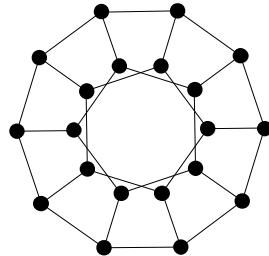
⋮ ⋮

Obtenemos un total 60 ciclos hamiltonianos distintos que empiezan y terminan en el vértice 1, el resto de ciclos podríamos obtenerlo con estos rotando el ciclo obtenido. Si observamos con atención, cada recorrido aparece dos veces como era de esperar, puesto que como hemos calculado todos los ciclos de Hamilton que parten y terminan en el vértice 1, por cada camino aparecerá otro que consistirá en realizar el recorrido inverso (desde el último vértice al primero). Nos quedamos sólo con uno de los ciclos, quedando en total 30 ciclos:

```
In]:= hamilton2={hamilton[[1]]};
Do[
  caminoinv=Table[
    hamilton[[CONTADORi]][[21-i]],{i,0,20}];
  If[Intersection[hamilton2,{caminoinv}]=={},AppendTo[hamilton2,hamilton[[CONTADORi]]]];
  ,{CONTADORi, 2,Length[hamilton]}];
hamilton=hamilton2;
```

Intentamos dibujar el dodecaedro directamente:

```
In]:= GraphPlot[A,PlotStyle->{Black,PointSize[.05]}]
Out]=
```



Como Mathematica no dibuja el grafo como en la ilustración 7.7., lo introducimos manualmente, indicando sus coordenadas:

```
In]:= coordenadas=Table[{0,0},{i,20}];
coordenadas[[1]]={Cos[1/5*2*Pi]*3,Sin[1/5*2*Pi]*3};
coordenadas[[11]]={Cos[2/5*2*Pi]*3,Sin[2/5*2*Pi]*3};
coordenadas[[20]]={Cos[3/5*2*Pi]*3,Sin[3/5*2*Pi]*3};
coordenadas[[18]]={Cos[4/5*2*Pi]*3,Sin[4/5*2*Pi]*3};
coordenadas[[2]]={Cos[2*Pi]*3,Sin[2*Pi]*3};
coordenadas[[3]]={Cos[1/5*2*Pi]*2,Sin[1/5*2*Pi]*2};
```

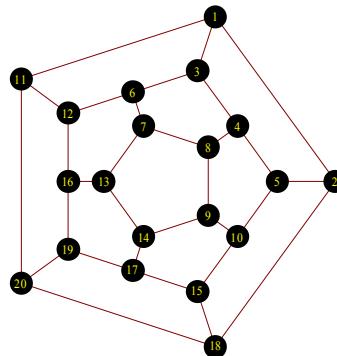
```

coordenadas[[12]]={Cos[2/5*2*Pi]*2,Sin[2/5*2*Pi]*2};
coordenadas[[19]]={Cos[3/5*2*Pi]*2,Sin[3/5*2*Pi]*2};
coordenadas[[15]]={Cos[4/5*2*Pi]*2,Sin[4/5*2*Pi]*2};
coordenadas[[5]]={Cos[2*Pi]*2,Sin[2*Pi]*2};
coordenadas[[7]]={Cos[15/50*2*Pi],Sin[15/50*2*Pi]};
coordenadas[[13]]={Cos[25/50*2*Pi],Sin[25/50*2*Pi]};
coordenadas[[14]]={Cos[35/50*2*Pi],Sin[35/50*2*Pi]};
coordenadas[[9]]={Cos[45/50*2*Pi],Sin[45/50*2*Pi]};
coordenadas[[8]]={Cos[55/50*2*Pi],Sin[55/50*2*Pi]};
coordenadas[[4]]=((coordenadas[[3]][[1]]+coordenadas[[5]]
[[1]])/2,(coordenadas[[3]][[2]]+coordenadas[[5]][[2]])/2);
coordenadas[[6]]=((coordenadas[[3]][[1]]+coordenadas[[12]]
[[1]])/2,(coordenadas[[3]][[2]]+coordenadas[[12]][[2]])/2);
coordenadas[[10]]=((coordenadas[[15]][[1]]+coordenadas[[5]]
[[1]])/2,(coordenadas[[15]][[2]]+coordenadas[[5]][[2]])/2);
coordenadas[[16]]=((coordenadas[[12]][[1]]+coordenadas[[19]]
[[1]])/2,(coordenadas[[12]][[2]]+coordenadas[[19]][[2]])/2);
coordenadas[[17]]=((coordenadas[[15]][[1]]+coordenadas[[19]]
[[1]])/2,(coordenadas[[15]][[2]]+coordenadas[[19]][[2]])/2);

```

In[]:= **GraphPlot[A,VertexRenderingFunction->({Black,Disk[#,2],
Yellow,Text[#2,#1]}&),
VertexCoordinateRules->coordenadas]**

Out[]=



Por último dibujamos los ciclos de Hamilton que hemos obtenido:

In[]:= **W=Table[i,{i,Dimensions[A][[1]]}];
F={};
Do[Do[
If[A[[i,j]]==1,AppendTo[F,Sort[{i,j}]]];
,{i,j}];{j,Dimensions[A][[1]]}];
Do[
camino=Table[Sort[{hamilton[[CONTADORj]][[i]],
hamilton[[CONTADORj]][[i+1]]}]]**

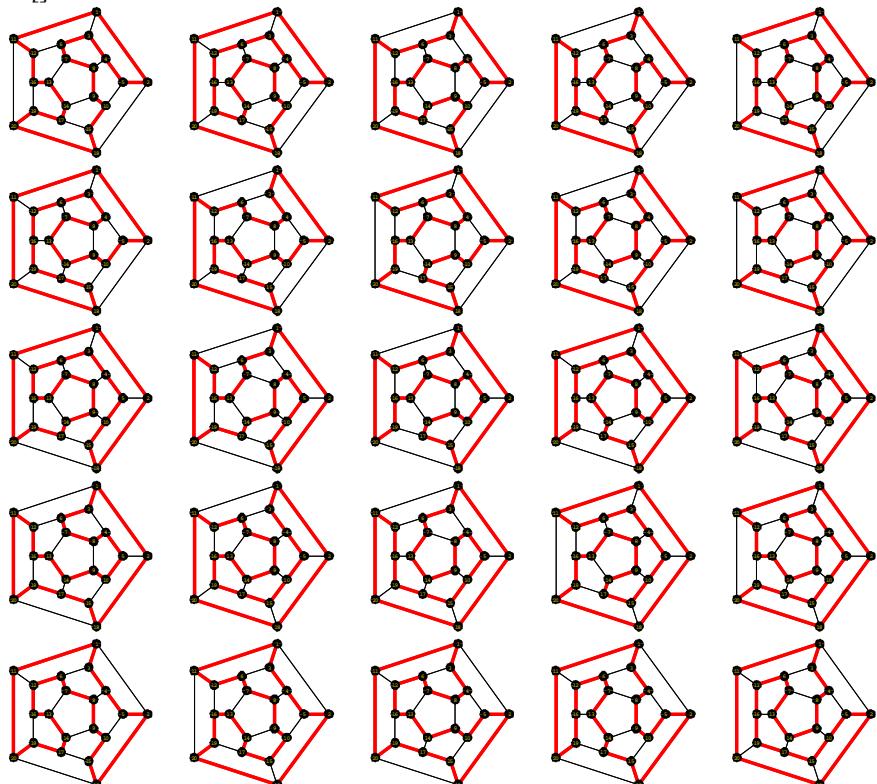
```

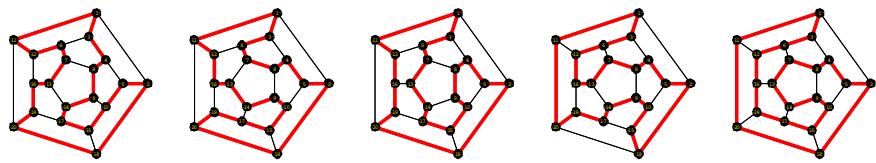
,{i,Length[hamilton][[CONTADORj]]-1}];

Do[
  If[Intersection[camino,{F[[CONTADORi]]}]=={},
    colores[F[[CONTADORi]]]=Black;
    grosor[F[[CONTADORi]]]=0.005;
  ,
    colores[F[[CONTADORi]]]=Red;
    grosor[F[[CONTADORi]]]=0.025;
  ];
,{CONTADORi,Length[F]}];
Print[GraphPlot[A,
  VertexRenderingFunction->({Black,Disk[#,2],Yellow,
    Text[#2,#1]&},VertexCoordinateRules->coordenadas,
  EdgeRenderingFunction->Function[{i,j},
    {colores[Sort[j]],Thickness[grosor[Sort[j]]],Line[i]}]];
,{CONTADORj,Length[hamilton]}];

```

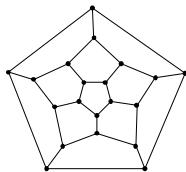
Out[]=





En la versión 6 de Mathematica y dentro del paquete `<<Combinatorica`` encontramos algunos de los grafos finitos más conocidos ya definidos como objetos de tipo grafo, en particular el dodecaedro, nos referiremos a él con “`DodecahedralGraph`”:

```
In]:= <<Combinatorica`  
In]:= ShowGraph[DodecahedralGraph]  
Out]=
```



Es obvia la ventaja, aunque sólo encontramos predefinidos³² algunos de los grafos más conocidos. Podemos realizar los mismos cálculos:

```
In]:= Length[HamiltonianCycle[DodecahedralGraph,All]]  
Out]= 60
```

Si comparamos la efectividad de la función `HAMILTON[]` con `HamiltonianCycle[]`, es sorprendente que la efectividad de la construida en este libro sea mayor, con `Timing[]` podemos comprobar que el tiempo empleado por `HamiltonianCycle[]` en calcular todos los ciclos de Hamilton de este ejemplo es el triple del empleado por `HAMILTON[]`.

Y por último, comprobamos que el grafo predefinido en Mathematica y el que hemos definido manualmente, son isomorfos:

```
In]:= IsomorphicQ[FromAdjacencyMatrix[A],  
DodecahedralGraph]  
Out]= True
```

□

Ejemplo 7.22. Podemos analizar el problema de un “laberinto” utilizando grafos simples. Consideremos el siguiente laberinto:

³² El tetraedro (“`TetrahedralGraph`”), el cubo (“`CubicalGraph`”), el octaedro (“`OctahedralGraph`”) y el icosaedro (“`IcosahedralGraph`”), entre otros...

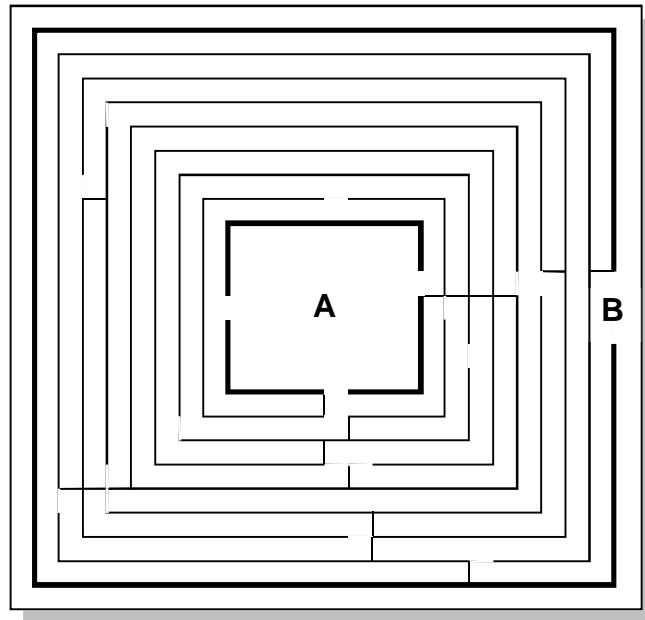


Ilustración 7.8.

Asociado a cualquier laberinto podemos encontrar un grafo simple, pondremos un punto o vértice en cada intersección, en los caminos sin salida y en el punto inicial y final del laberinto:

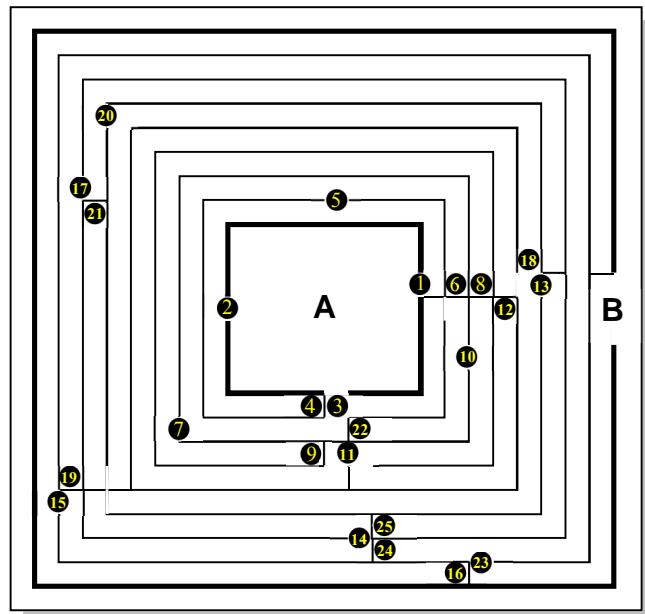


Ilustración 7.9.

Si de un vértice podemos llegar a otro sin pasar por ningún otro vértice, los conectaremos por un lado:

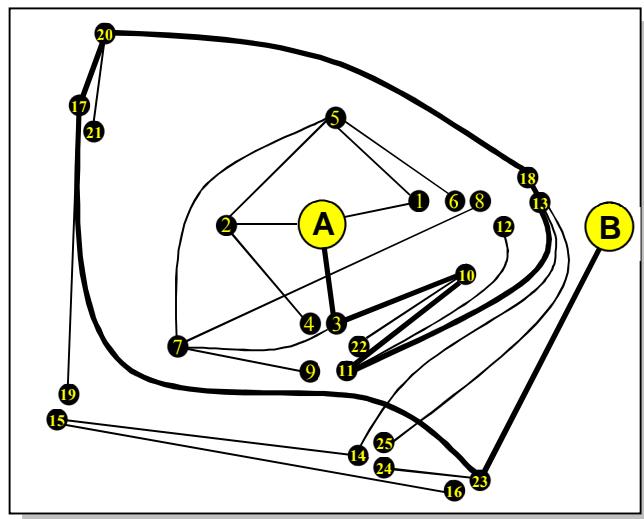


Ilustración 7.10.

Introducimos el grafo en el ordenador:

```
In[]:= W={A,B,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
21,22,23,24,25};  
F={A->1,A->2,A->3,2->4,1->5,2->5,5->6,5->7,7->8,7->9,7->3,
3->10,10->22,10->11,11->12,11->13,13->25,13->14,14->15,
15->16,13->18,18->20,20->21,20->17,17->19,17->23,
23->24,23->B};
```

Cualquier camino que conecta los vértices “A” y “B” representará una solución al problema, buscamos los más cortos, esto es, las geodésicas:

In[]:= A=MATRIZADYACENCIA[W,F];

Recordemos que al calcular la matriz de adyacencia los vértices son identificados por sus subíndices, perdiéndose cualquier información sobre sus nombres, por tanto el vértice “A” será el uno y el vértice “B” el dos, ya que ocupan respectivamente las posiciones primera y segunda en “F”, por tanto, para calcular las geodésicas escribiremos:

```
In]:= CAMINOS[vertice1_,vertice2_,longitud_,  
matrizadyacencia_]:=
```

In[]:= **GEODESICAS[u_,v_,matrizadyacencia_]:=Module[{B},**

⋮ ⋮

In[]:= **GEODESICAS[1,2,A]**

Out[] = { {1,5,12,13,15,20,22,19,25,2} }

Ahora recuperaremos el nombre original de los vértices:

In[]:= **geo={1,5,12,13,15,20,22,19,25,2};**
Table[W[[geo[[i]]]],{i,Length[geo]}]

Out[] = {A,3,10,11,13,18,20,17,23,B}

Si redibujamos el grafo, lo veremos más claro:

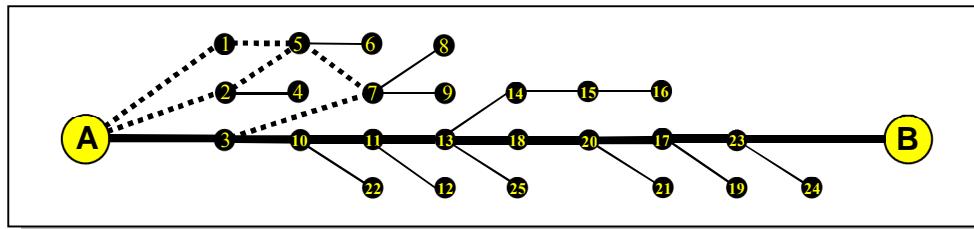


Ilustración 7.11.

□

En el siguiente ejemplo utilizaremos varias funciones particulares de Mathematica y en particular de la versión 6, si quisieramos trasladar los cálculos y las representaciones gráficas del mismo a otros lenguajes o programas distintos a Mathematica, en la mayoría nos encontraríamos con la ausencia de librerías o paquetes de funciones tan específicos por lo que supondría un gran esfuerzo llegar a los mismos resultados.

Ejemplo 7.24. En este ejemplo calcularemos ciclos de Euler y Hamilton, si existen, de los sólidos platónicos (el tetraedro, el octaedro, el cubo, el icosaedro y el dodecaedro³³) y del hipercubo 4-dimensional. Para ello utilizaremos los objetos de tipo grafo predefinidos e incluidos en el paquete <<Combinatoria> de la versión 6. También usaremos las funciones incluidas en el mismo paquete para realizar estos cálculos.

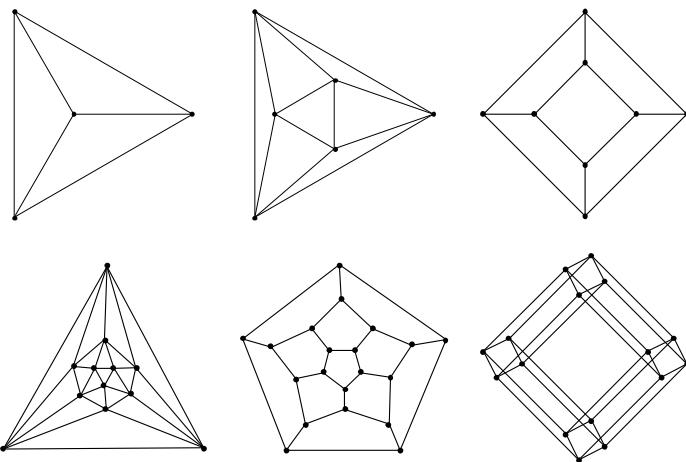
Empecemos por definirlos y representarlos con ShowGraphArray[]:

In[]:= <<Combinatoria`

³³ En el ejemplo 7.21. calculamos todos los ciclos de Hamilton del dodecaedro usando la función 7.21. (HAMILTON[]), ahora realizaremos esta comprobación junto con otras, utilizando funciones específicas de la versión 6 de Mathematica del paquete de funciones: <<Combinatoria>`.

In[1]:= **grafos={TetrahedralGraph,OctahedralGraph,CubicalGraph,
IcosahedralGraph,DodecahedralGraph,Hypercube[4]};
ShowGraphArray[{grafos[[1;;3]],grafos[[4;;6]]}]**

Out[1]=



Ahora comprobamos cuáles son de Euler y cuáles de Hamilton:

In[2]:= **Do[Print[i," Euler: ",EulerianQ[grafos[[i]]],
" Hamilton: ",HamiltonianQ[grafos[[i]]]],
{i,Length[grafos]}];**

Out[2]=

| | | |
|---|--------------|----------------|
| 1 | Euler: False | Hamilton: True |
| 2 | Euler: True | Hamilton: True |
| 3 | Euler: False | Hamilton: True |
| 4 | Euler: False | Hamilton: True |
| 5 | Euler: False | Hamilton: True |
| 6 | Euler: True | Hamilton: True |

Buscamos un ciclo de Euler del octaedro y del hipercubo 4-dimensional y creamos unas animaciones que representamos usando *Animate[]*, para el octaedro:

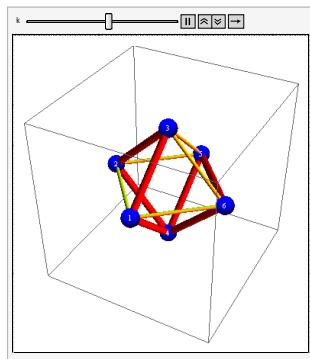
In[3]:= **grafo=OctahedralGraph;
camino=EulerianCycle[grafo];
lados=Edges[grafo];
Do[
 ciclo=camino[[1;k+1]];
 ladosciclo=Table[Sort[{ciclo[[i]],ciclo[[i+1]]}],
 {i,Length[ciclo]-1}];
 Do[
 If[Intersection[{Sort[lados[[i]]]},ladosciclo]=={},
 color[Sort[lados[[i]]]]=Yellow;**

```

grosor[Sort[lados[[i]]]]=.02;
,
color[Sort[lados[[i]]]]=Red;
grosor[Sort[lados[[i]]]]=.04;
];
,{i,Length[lados]}];
framegrafo[k]=GraphPlot3D[ToAdjacencyMatrix[grafo],
VertexRenderingFunction->({Blue,Sphere[#,1],White,
Text[#2,#1]&}),EdgeRenderingFunction->Function[{i,j},
{color[Sort[j]],Cylinder[i,grosor[Sort[j]]]}]];
,{k,Length[camino]-1}];
Animate[framegrafo[k],{k,1,Length[camino]-1,1}]

```

Out[]=

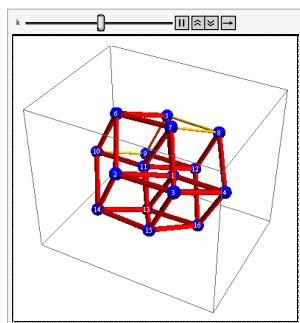


Para el hipercubo usaremos el mismo código:

In[]:= **grafo=Hypercube[4];**

⋮ ⋮

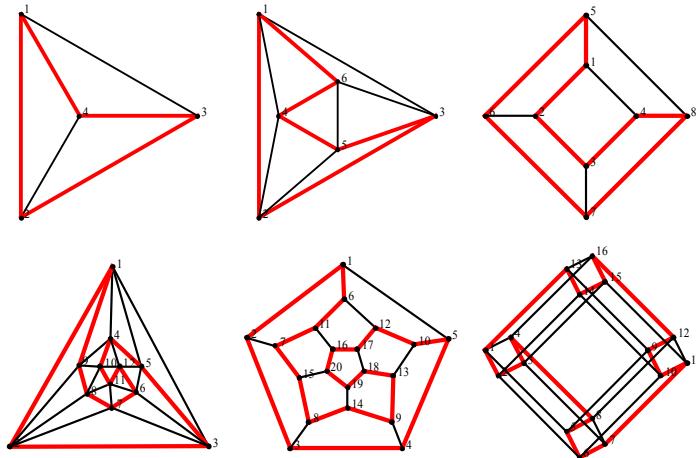
Out[]=



Ahora buscamos un ciclo de Hamilton de cada uno de los grafos y lo representamos:

```
In]:= grafo={};  
Do[  
  ciclo=HamiltonianCycle[grafos[[k]]];  
  ladosciclo=Table[Sort[{ciclo[[i]],ciclo[[i+1]]}],  
   {i,Length[ciclo]-1}];  
  lados=Edges[grafos[[k]]];  
  vertices=Vertices[grafos[[k]]];  
  color={};  
  grosor={};  
  Do[  
    If[Intersection[{Sort[lados[[i]]]},ladosciclo]=={},  
     AppendTo[color,Black];AppendTo[grosor,.01];  
     ,  
     AppendTo[color,Red];AppendTo[grosor,.02];]  
    ,{i,Length[lados]}];  
  AppendTo[grafo,Graph[Table[{lados[[i]],  
   EdgeColor->color[[i]],EdgeStyle->Thickness[grosor[[i]]]},  
   {i,Length[lados]}],  
   Table[{vertices[[i]]},{i,Length[vertices]}]]];  
  ,{k,Length[grafos]}];  
 ShowGraphArray[{grafo[[1;;3]],grafo[[4;;6]]}  
 ,VertexLabel->True]
```

Out]:=



Y los representamos en 3 dimensiones:

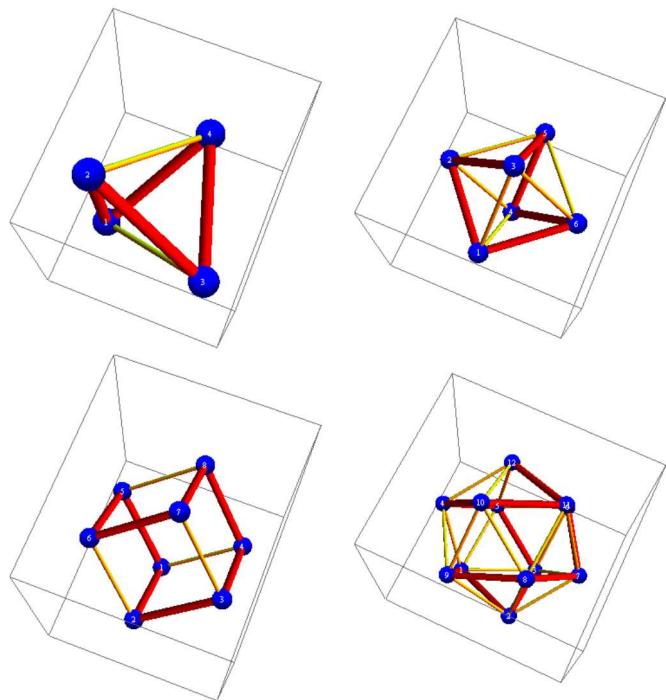
```
In]:= Clear[color,grosor];  
Do[  
  ciclo=HamiltonianCycle[grafos[[k]]];  
  ladosciclo=Table[Sort[{ciclo[[i]],ciclo[[i+1]]}],  
   {i,Length[ciclo]-1}];  
  lados=Edges[grafos[[k]]];  
  vertices=Vertices[grafos[[k]]];
```

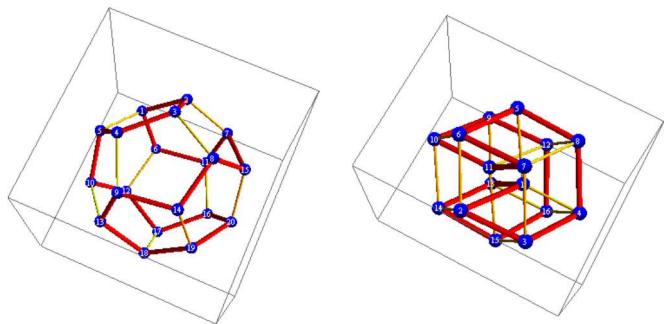
```

Do[
  If[Intersection[{Sort[lados[[i]]]},ladosciclo]=={},
    color[Sort[lados[[i]]]]=Yellow;
    grosor[Sort[lados[[i]]]]=.02;
  ,
    color[Sort[lados[[i]]]]=Red;
    grosor[Sort[lados[[i]]]]=.04;
  ];
,{i,Length[lados]}];
Print[GraphPlot3D[ToAdjacencyMatrix[grafos[[k]]],
  VertexRenderingFunction->({Blue,Sphere[#,1],White,
  Text[#2,#1]&},EdgeRenderingFunction->Function[{i,j},
  {color[Sort[j]],Cylinder[i,grosor[Sort[j]]]}])];
,{k,Length[grafos]}]

```

Out[]=





□

8. EJERCICIOS

Ejercicio 7.1. Puentes y articulaciones. Una articulación es un vértice v de un grafo conexo, tal que si consideramos el subgrafo que resulta con $W - \{v\}$ y el subconjunto de todos los lados no incidentes en v , éste no es conexo. Análogamente un punto es un lado e de un grafo conexo tal que si consideramos el subgrafo $(W, F - \{e\})$, éste no es conexo. Diseñar rutinas que detecten puentes y articulaciones en un grafo.

□

Ejercicio 7.2. Excentricidad y Diámetro. Sea G un (p, q) -grafo no orientado y conexo, definiremos la excentricidad de un vértice v , $ex(v)$, como la mayor de las distancias de los caminos elementales (no cerrados) que empiezan en v . Es obvio que $ex(v) \leq (p - 1), q$ para cada vértice v . Al vértice con menor excentricidad se le llama central y la excentricidad de este se le llama radio del grafo, mientras que al conjunto de todos los vértices centrales se le llama centro del grafo. Al máximo de las excentricidades de todos los vértices se le llama diámetro del grafo.

- Realizar una función que determine la excentricidad de un vértice cualquiera.
- Usando la función anterior realizar un programa que determine el radio, el centro y el diámetro de un grafo.
- Aplicarlos a K_8 y a $K_{4,4}$.

□

Ejercicio 7.3. Podemos modificar la función 7.12. para calcular todos los caminos de longitud fija que conectan dos vértices que además son elementales, para ello bastaría con añadir la siguiente línea al código de la función 7.12:

```
CAMINOS2[vertice1_,vertice2_,longitud_,matrizadyacencia_]:=
```

```
: : :
```

```
,{j,Dimensions[matrizadyacencia][[1]]}};
```

```
adyacentes=Complement[adyacentes,caminos[[k]]];
```

←

If[adyacentes $\neq\{\}$,

⋮ ⋮

];

Con esta modificación podemos calcular todos los caminos elementales que conectan “vertice1” con “vertice2” suponiendo que sean distintos, si son iguales entonces calcularíamos los ciclos elementales, y este último problema lo encontraremos resuelto en el capítulo 8.

De forma análoga hacer las modificaciones oportunas para que se calculen los caminos simples. \square

Ejercicio 7.4. Aprovechando la función 7.12. crear otras que calculen:

- a) Todos los ciclos.
- b) Todos los circuitos.
- c) Todos los caminos simples.
- d) Todos los caminos elementales.

\square

Ejercicio 7.5. Determinar todos los grafos no orientados conexos de 7 vértices y con 18 lados o menos. \square

Ejercicio 7.6. Determinar las componentes fuertemente conexas de los grafos de las ilustraciones 7.4, 7.5. y 7.6. \square

Ejercicio 7.7. Dado el siguiente grafo:

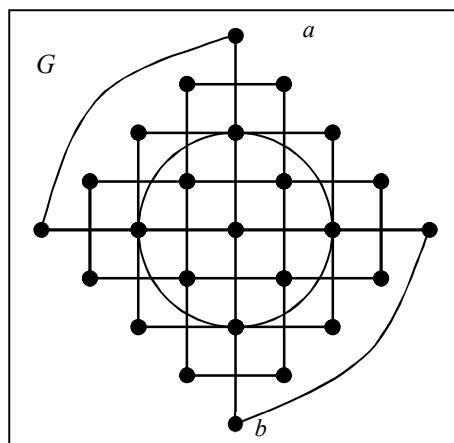


Ilustración 7.12.

- a) Determinar si es un grafo de Euler.
 b) Calcular en caso afirmativo un ciclo de Euler.
 c) Calcular la distancia entre el vértice a y b .

□

Ejercicio 7.8. Dado el siguiente grafo:

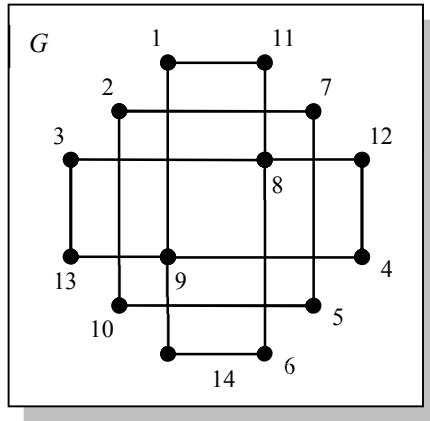


Ilustración 7.13.

- a) Calcular las componentes conexas del grafo.
 b) Determinar la distancia entre los vértices 11 y 13. Determinar las geodésicas que los conectan.
 c) Igual que el apartado b) para los vértices 7 y 9.
 d) Calcular el número de caminos de longitud 8 que conectan los vértices 3 y 6.
 e) Determinar todos los caminos de longitud 6 que conectan los vértices 12 y 14.

□

Ejercicio 7.9. Dado el siguiente digrafo:

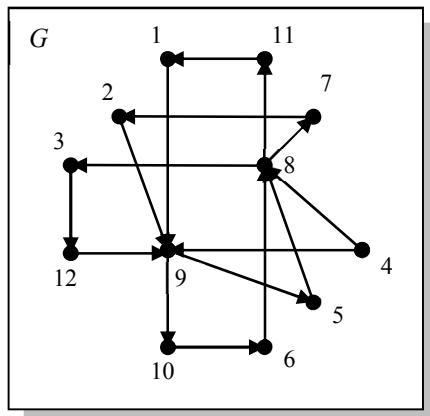


Ilustración 7.14.

- a) Calcular las componentes fuertemente conexas del grafo

- b) Calcular los sumideros y las fuentes.
- c) ¿Es un grafo de Euler?
- d) Determinar la distancia entre los vértices 1 y 8. Determinar las geodésicas que los conectan.
- e) Calcular el número de caminos de longitud 8 que conectan los vértices 3 y 6.
- f) Determinar todos los caminos de longitud 6 que conectan los vértices 2 y 9.

□

Ejercicio 7.10. Calcular, salvo isomorfismo, todos los grafos dirigidos de 5 vértices de Euler.

□

Ejercicio 7.11. Calcular, salvo isomorfismo, todos los grafos no orientados de 6 vértices de Euler.

□

Ejercicio 7.12. Hacer las alteraciones necesarias en las funciones 7.6. y 7.7. para que funcionen con las matrices de adyacencia.

□

Ejercicio 7.13. Escribir programas que determinen si un grafo dirigido es conexo (no necesariamente fuertemente conexo) y determinar sus componentes conexas. Aplicarlo a los digrafos que aparecen en el capítulo.

□

Ejercicio 7.14. Crear un algoritmo que determine un ciclo de Euler de un grafo dirigido de Euler cualquiera. Aplicarlo a los grafos dirigidos de Euler que aparecen en el capítulo.

□

Ejercicio 7.15. Crear un algoritmo que determine si un grafo es de Euler haciendo un barrido sobre los posibles ciclos, de forma similar a 7.21. Utilizarlo para calcular todos los ciclos de Euler de los grafos de Euler que aparecen en el capítulo.

□

Ejercicio 7.16. Dado el siguiente grafo:

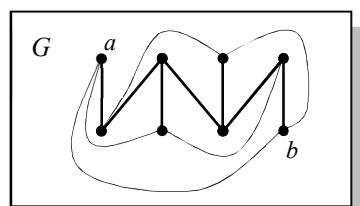


Ilustración 7.15.

- a) Determinar si es de Hamilton.
- b) Calcular los caminos de longitud 5 que conectan los vértices a y b .
- c) Calcular la distancia y las geodésicas que conectan los vértices a y b .

□

Ejercicio 7.17. Generar una función similar a la 7.21. para grafos dirigidos que compruebe si un digrafo es de Hamilton.



Ejercicio 7.18. Calcular todos los grafos no orientados de Hamilton de 8 vértices.



Ejercicio 7.19. Calcular todos los grafos no orientados de Hamilton y de Euler de 8 vértices.



Ejercicio 7.20. Calcular todos los grafos dirigidos de Hamilton de 8 vértices.



Ejercicio 7.21. En el epígrafe 6 se estudian algunas condiciones suficientes o necesarias para que un grafo sea de Hamilton. Implementar las cuatro condiciones en una rutina cuya entrada sea un grafo cualquiera y por salida tenga la información que podamos obtener del grafo desde las condiciones anteriores.



Ejercicio 7.22. Vías de un tren. Como pudimos apreciar en el capítulo 5, la noción de grafo puede moldearse según nuestros intereses, de hecho en este libro nos hemos limitado a las nociones de grafo y digrafo que vienen implementadas en Mathematica 5.2, olvidándonos de los multigrafos o la posibilidad de lazos en un vértice. Imaginemos una situación en la que, si bien, el problema es fácilmente modelable por un grafo no orientado tal y como lo hemos definido, los caminos que nos interesan en él, verifican una propiedad adicional:

- $|e_i \cap e_{i+1}| = 1$, para cada $i = 1, 2, \dots, n - 1$, esto es, dos lados consecutivos son distintos y tienen un único vértice en común.

En tal caso, tendríamos que limitarnos sólo a los caminos que verifican tal propiedad, por tanto el teorema del número de caminos no sería útil para resolver problemas relacionados con este tipo de caminos.

Obsérvese que dicha propiedad no llega a ser la de un camino elemental o simple, sin embargo, si que formaliza algunos problemas en donde, caso de no exigir esta propiedad daría lugar a caminos triviales o no válidos para nuestras pretensiones, por ejemplo, la vía de un tren, donde con un único tramo de vía que una dos puntos no podemos definir un ciclo que permita dar la vuelta el tren (ésta es la razón por la que el diseño de muchas máquinas locomotoras y trenes están pensados para no dar la vuelta y únicamente invertir la marcha) .

Si queremos estudiar este tipo de problemas necesitaremos estudiar estos caminos. Con la siguiente rutina comprobamos si una sucesión de lados dada es un camino de este estilo y caso de serlo nos proporcionará el listado de vértices que también define al camino:

| FUNCIÓN | COMENTARIOS |
|--|--|
| <pre> PATH[CAMINO,_F_]:=Module[{i,inicio}, camino=True; If[Length[Intersection[Union[CAMINO, Table[CAMINO[[i]][[2]]→CAMINO[[i]][[1]], ,{i,Length[CAMINO]}]],F]]!= Length[CAMINO], camino=False] </pre> | Comprobamos que todos los lados pertenezcan al grafo, como es un grafo no dirigido identificamos los lados $a \rightarrow b$ y $b \rightarrow a$. |
| <pre> ', If[Length[CAMINO]!=1, Do[If[Length[Intersection[{CAMINO[[i]][[1]],CAMINO[[i]][[2]]}, {CAMINO[[i+1]][[1]],CAMINO[[i+1]][[2]]}]]!=1, camino=False;Break[]]; ,{i,Length[CAMINO]-1}]; If[camino, inicio=Complement[{CAMINO[[1]][[1]],CAMINO[[1]][[2]]}, {CAMINO[[2]][[1]],CAMINO[[2]][[2]]}][[1]]; listavertices={inicio}; Do[AppendTo[listavertices, Complement[{CAMINO[[i]][[1]],CAMINO[[i]][[2]]}, {listavertices[[i]]}][[1]]]; If[Length[Intersection[{CAMINO[[i]][[1]], CAMINO[[i]][[2]]}], {listavertices[[i]],listavertices[[i+1]]}]!=2, camino=False;Break[]]; ,{i,Length[CAMINO]}];]; ', listavertices={CAMINO[[1]][[1]], CAMINO[[1]][[2]]};];]; If[camino, Print["Es un camino"]; Print[listavertices]; Print[CAMINO]; , Print["No es un camino"]]] </pre> | Se comprueban las dos condiciones necesarias para que una sucesión de lados sea un camino y se construye la sucesión de vértices. |
| | Salida de resultados. |

Función 7.22. Caminos* en grafos no orientados.

- a) Consideraremos que el grafo de la ilustración 7.12. represente un circuito de vías de tren en donde circulan trenes que no pueden invertir la marcha por la

configuración de los mismos. Calcular las posibles rutas que unan Andalucía con Cataluña pasando por Madrid y Extremadura.

- b) Diseñar rutinas que comprueben si un camino no orientado expresado como una sucesión de vértices es del tipo anterior para grafos no orientados.
- c) Modificar la función 7.12. para que calcule todos los caminos así definidos de longitud l , que conecten dos vértices.
- d) Obsérvese que estos caminos pueden representarse como sucesión de lados sin que ello implique ambigüedad alguna. En el caso de digrafos los caminos siempre verifican una propiedad similar.

□

Ejercicio 7.23. Queremos dar respuesta a la siguiente pregunta, ¿cuál es el menor número de Comunidades Autónomas españolas por las que debemos pasar si queremos ir por carretera de Granada a San Sebastián?

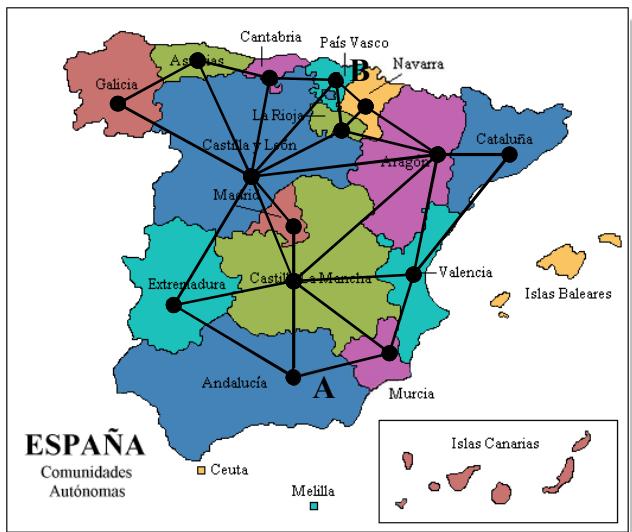


Ilustración 7.16.

Para responder a esta pregunta consideraremos el grafo no orientado cuyos vértices son las Comunidades Autónomas españolas de la Península Ibérica, siendo adyacentes aquellos vértices asociados a Comunidades con frontera común, entonces nos quedaría el grafo de la ilustración 7.16.

Tendríamos que hallar la distancia entre el vértice A y el vértice B.

- a) Introducir el grafo en Mathematica y dar respuesta a la cuestión.
- b) Dadas dos provincias españolas de la península, crear un programa que determine el menor número de provincias españolas por las que habría que pasar para ir por carretera de una a otra.
- c) De forma análoga calcular el grafo cuyos vértices sean los países de la Unión Europea, siendo adyacentes aquellos vértices que representen a países con frontera común.

□

Ejercicio 7.24. Crear una rutina que determine si una sucesión alternada de vértices y lados es un camino en un grafo no orientado, y en tal caso, que calcule la sucesión de vértices y la de lados que también representan al camino.

Crear una función que determine si un camino en un multigrafo es simple o elemental.

□

Ejercicio 7.25. Crear una rutina que determine si una sucesión alternada de vértices y lados es un camino en un digrafo, y en tal caso, calcule la sucesión de vértices y la de lados que también representan al camino.

Crear una función que determine si un camino en un multigrafo dirigido es orientado, simple o elemental.

□

Ejercicio 7.26. El problema de los siete puentes de Königsberg dio origen a la teoría de los grafos y fue resuelto por Leonhard Euler en 1736. El problema consistía en encontrar un camino que permitiera pasar por todos los puentes de la ciudad una única vez terminando en el punto de partida. Esquemáticamente representamos los puentes:

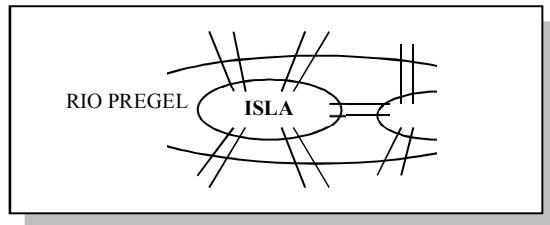


Ilustración 7.17.

Lo interpretamos como el multigrafo:

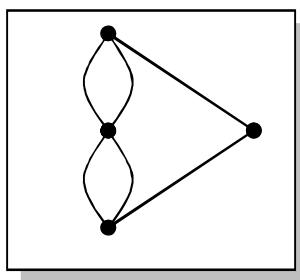


Ilustración 7.18.

Sabiendo que la caracterización de grafos de Euler, teorema 7.2, también es válida para multigrafos, encontrar la solución al problema.

□

Ejercicio 7.27. En un tablero de ajedrez los movimientos del caballo son en forma de ele, si numeramos cada casilla del tablero de ajedrez y los asociamos a vértices, no todas las casillas del tablero serían adyacentes para un caballo. Analizamos el siguiente problema, ¿cuántos movimientos serían necesarios para ir de uno a otro punto del tablero de ajedrez?

Observamos en la ilustración 7.15. que el vértice b1 tendría grado tres y es adyacente a los vértices a3, c3 y d2. La solución vendrá dada por un grafo con 64 vértices, por tanto su matriz de adyacencia tendrá 64 filas y 64 columnas.

- Construir el grafo que represente los movimientos del caballo.
- ¿Es conexo?
- Calcular el número de movimientos necesarios para ir del vértice c1 al vértice a8.
- De la misma forma estudiar los movimientos de la torre, el alfil, la dama, el rey y los peones.

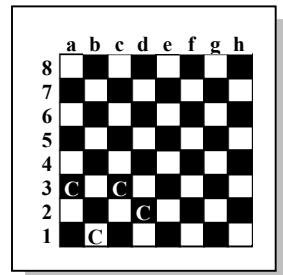


Ilustración 7.19.

□

Ejercicio 7.28. Problema del cartero chino. Consiste en encontrar un camino cerrado que pase por cada lado al menos una vez y que sea de longitud mínima (en el caso de ser de Euler, sería el ciclo de Euler).

Crear una función que resuelva el problema para grafos no orientados y otra para dígrafos.

□

Ejercicio 7.29. En un grafo conexo no orientado existe un camino de Euler (no ciclo) entre dos vértices distintos si y sólo si todos los vértices excepto esos dos son de grado par.

- Crear una función que compruebe si un grafo contiene un camino de Euler.
- Crear una función que determine, si existe, un camino de Euler.

□

Ejercicio 7.30. Comprobar, usando la función HamiltonianQ[], qué grafos de 8 vértices son de Hamilton.

□

Ejercicio 7.31. Un grafo es bipartito si y sólo si no contiene un ciclo de longitud impar. Crear un programa que determine si un grafo es bipartito a partir de esta equivalencia. Comprobar su eficacia respecto al dado en el capítulo.

□

Ejercicio 7.32. En el ejemplo 7.21. se representan gráficamente los recorridos de los ciclos de Hamilton calculados. Aprovechar el código allí usado para realizar un programa que dibuje el recorrido de cualquier camino, tanto para grafos no orientados como dirigidos.

□

Ejercicio 7.33. La función 7.21. determina todos los ciclos de Hamilton, si existen, de un grafo que empiezan y terminan en el vértice 1.

- a) Modificar la función 7.21. para que calcule todos los ciclos de Hamilton.
- b) Modificar la función 7.21. para que calcule todos los ciclos de Hamilton tales que la representación gráfica de su recorrido (como se hace en el ejemplo 7.21.) sea distinta.

□

Ejercicio 7.34. Un sistema de posicionamiento global o navegador GPS, la gestión del tráfico de una ciudad o una red telemática, entre otros, son ejemplos de situaciones reales en las que usamos la teoría de grafos.

- a) Analizar para estos ejemplos, qué tipo de grafo se precisa para modelarlos y posteriormente razonar cuál es la utilidad que tienen en estos ejemplos las geodésicas, los ciclos de Euler y los ciclos de Hamilton.
- b) Encontrar nuevos ejemplos en los que los caminos jueguen un papel importante.

□

Ejemplo 7.35. Consideremos varios pueblos comunicados de la siguiente forma:

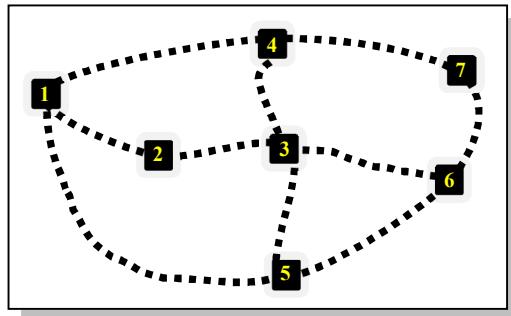


Ilustración 7.20.

Se pide:

- a) Partiendo y terminando en el pueblo número 5, ¿será posible recorrer todos los pueblos pasando por cada uno de ellos una vez? En caso afirmativo, de cuántas formas podemos hacerlo.
- b) ¿Cuántas rutas podemos seguir para ir desde el pueblo número 2 al 7 pasando a lo sumo por otros dos pueblos?
- c) Si quisieramos inspeccionar las carreteras que unen los pueblos, partiendo y terminando en el pueblo número 5, ¿podríamos hacerlo pasando una única vez por cada carretera?

□

Ejercicio 7.36. Una red de metro representa un grafo no dirigido (los trenes del metro están diseñados para invertir la marcha, por lo que no se presenta el problema del ejercicio 7.20., además las redes de metro suelen ser de doble vía), cuyos vértices son las estaciones, en la

ilustración 7.21. se muestra la red de metro de Madrid, utilizar las herramientas del capítulo para encontrar la ruta más corta entre dos estaciones cualesquiera.



Ejercicio 7.37. Utilizando la función 7.12. (CAMINOS[]), programar dos rutinas que determinen todos los ciclos de Euler y de Hamilton de un grafo cualquiera. Comparar la eficacia de las mismas respecto a las incluidas en el capítulo.



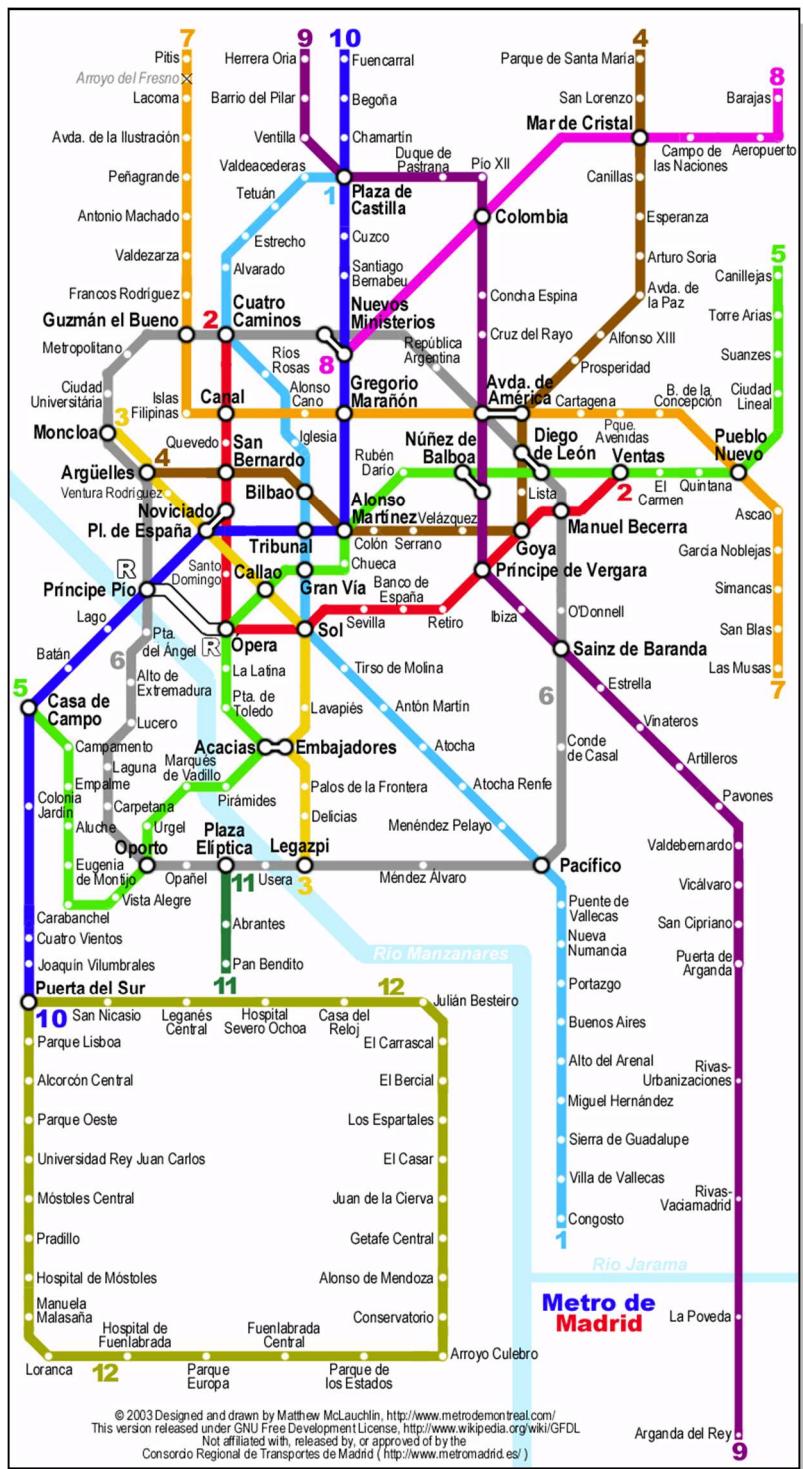


Ilustración 7.21.

8. COLORACIÓN DE UN GRAFO, GRAFOS PLANOS Y ÁRBOLES

En este último capítulo estudiaremos las cuestiones más relevantes sobre la coloración de grafos, los grafos planos y los árboles. La resolución de algunos de los problemas de este capítulo necesitará de una programación más compleja por lo que su seguimiento puede resultar algo más complicado. Determinaremos una coloración óptima de un grafo de forma combinatoria, también exploraremos algunos algoritmos de coloración que no calculan necesariamente una coloración óptima pero que resultarán interesantes por su eficacia. Calcularemos el polinomio cromático de un grafo directamente o usando el teorema de descomposición para polinomios cromáticos (8.1.). Los grafos planos los resolvemos usando el teorema de Kuratowski (8.2.) y analizaremos la eficacia de la solución propuesta. Estudiaremos las regiones de un grafo plano y su coloración. Distinguiremos los árboles o bosques comprobando la existencia de ciclos o directamente usando el teorema 8.4.

Estudiaremos todos estos conceptos sólo para grafos no orientados, si bien no resultaría difícil generalizar el estudio a grafos dirigidos, incluso en algunos casos bastaría con usar las mismas rutinas con pequeños cambios por basarse éstas en las matrices de adyacencia (como la coloración de vértices) u olvidarnos de la orientación (por ejemplo cogiendo la matriz de adyacencia más su traspuesta) y de nuevo aplicar las mismas rutinas (como plano o árbol); obviamente para introducir estos y otros aspectos sobre estos conceptos en grafos dirigidos previamente hemos de analizar y definir claramente los correspondientes conceptos y variantes de los mismos en este tipo de grafos, sin embargo, por claridad y porque el caso más interesante es el no orientado, ya que es el habitual y además desde este, el dirigido se estudia con relativa facilidad, nos limitaremos al estudio en grafos no orientados.

1. COLORACIÓN DE UN GRAFO

Dado un grafo $G = (W, F)$ (no dirigido), colorear un grafo G consiste en asignar colores a cada vértice de forma que a dos vértices adyacentes no se les asigne un mismo color. Si el número de colores usado en una coloración de G es n entonces diremos que tenemos una n -coloración de G . Al menor n para el cual conseguimos una n -coloración se

le llama número cromático de G , en tal caso se dice que G es n -cromático. Una coloración óptima será una n -coloración donde n es el número cromático del grafo.

Evidentemente el número de colores que podemos usar para colorear un grafo estará comprendido entre el número cromático y el número de vértices del grafo. Nuestro principal objetivo será conseguir una coloración óptima. Para asegurar el cálculo del número cromático y de una coloración óptima utilizaremos métodos combinatorios, también veremos algunos algoritmos interesantes por su eficacia pero que no siempre lograrán encontrar una coloración óptima.

1.1. COLORACIONES ÓPTIMAS

Resolveremos el problema de forma combinatoria, esto es, calcularemos todas las posibles coloraciones del grafo para un número fijo de colores. Para ello utilizaremos el siguiente programa que determinará todas las coloraciones que existen del grafo usando a lo sumo λ colores:

| FUNCIÓN | COMENTARIOS |
|---|---|
| <code>NCOLORACIONES[matrizadyacencia_,lambda_]:=Module[{coloracionestemp,t,h,i,j,colorady,k},</code> | La función tendrá dos argumentos: la matriz de adyacencia del grafo identificando los vértices con sus subíndices y el número de colores: λ . |
| <code>coloraciones=Table[{k},{k,lambda}];</code> | La variable “coloraciones” almacenará todas las posibles coloraciones que construyamos. Directamente asignamos un color al primer vértice entre 1 y “lambda”. |
| <code>Do[</code> | Bucle que recorrerá todos los vértices del grafo. |
| <code>coloracionestemp={};</code> <code>Do[</code> | Bucle que recorre todas las posibles coloraciones conforme se van construyendo |
| <code>colorady={};</code> <code>Do[</code> <code>If[matrizadyacencia[[t,i]]==1,</code> <code>colorady=Union[colorady,</code> <code>{coloraciones[[h]][[t]]}];</code> <code>,{t,Length[coloraciones[[h]]]}];</code> | Comprobamos los vértices que son adyacentes al vértice “ i ” y los colores que tienen asignados (si ya han sido coloreados), los almacenamos en “colorady”. |
| <code>Do[</code> <code>If[Intersection[{j},colorady]=={},</code> <code>AppendTo[coloracionestemp,</code> <code>Append[coloraciones[[h]],j]];</code> <code>,{j,lambda}];</code> | Coloreamos el vértice “ i ” con todos los posibles colores que podamos. |
| <code>,{h,Length[coloraciones]}];</code> <code>coloraciones=coloracionestemp;</code> <code>,{i,2,Dimensions[matrizadyacencia][[1]]}];</code> | |
| <code>coloraciones</code> | Salida de resultados. |

| | |
|---|--|
| ; | |
|---|--|

Función 8.1. Las n -coloraciones de un grafo.

La función anterior construye de forma combinatoria todas las posibles coloraciones del grafo que a lo sumo utilicen λ colores, para ello identifica los colores con los naturales: 1, 2, ..., λ ; y nos devolverá listas de longitud p (donde p es el número de vértices del grafo) compuestas por naturales comprendidos entre 1 y λ , donde cada una de las listas se corresponderá con una coloración. Es decir, supongamos que aplicamos el programa a un grafo y un λ cualquiera, el programa tendrá por salida todas las μ -coloraciones posibles con $\mu \leq \lambda$, si son k :

$$\{l_1, l_2, \dots, l_k\}$$

Donde $l_i = \{a_{i1}, a_{i2}, \dots, a_{ip}\}$ es una coloración para cada i y donde $1 \leq a_{ij} \leq \lambda$ es un color para cada i, j . Interpretaremos la asignación de colores como sigue:

Vértice 1 (o vértice v_1) \Rightarrow color a_{i1} ,

Vértice 2 (o vértice v_1) \Rightarrow color a_{i2} ,

⋮

Vértice p (o vértice v_p) \Rightarrow color a_{ip} ,

Con la función 8.1. podemos calcular el número cromático fácilmente:

| FUNCIÓN | COMENTARIOS |
|--|---|
| NCOLORACIONES[matrizadyacencia_, lambda]:=... | Definimos la función 8.1. |
| NUMEROCROMATICO[matrizadyacencia_]:=Module[{}, | La función tendrá por entrada la matriz de adyacencia del grafo. |
| t=1; While[NCOLORACIONES[matrizadyacencia,t]=={}, t++]; | Buscamos el primer λ para el cual existe alguna coloración del grafo. |
| t | Salida de resultados. |

Función 8.2. Número cromático de un grafo.

Para representar gráficamente una de las coloraciones óptimas usaremos la siguiente función, donde identificaremos cada entero positivo con un color³⁴ (1 con negro, 2 con rojo, 3 con verde,...).

| FUNCIÓN | COMENTARIOS |
|--|--|
| NCOLORACIONES[matrizadyacencia_, lambda]:=... | Definimos la función 8.1. |
| NUMEROCROMATICO[matrizadyacencia_]:=... | Definimos la función 8.2. |
| DIBUJAR[matrizadyacencia_]:=Module[{colores,i}, | La función tendrá por entrada la matriz de |

³⁴ La identificación y la elección de colores es aleatoria, nos valdría cualquier otra.

| | |
|--|---|
| <pre> colorear,COLORES}, NUMEROCROMATICO[matrizadyacencia]; colorear=coloraciones[[1]]; colores={"Negro", "Rojo", "Verde", "Azul", "Amarillo", "Cyan", "Marrón", "Naranja", "Púrpura", "Rosa", "Gris"}; COLORES={Black,Red,Green,Blue, Yellow,Cyan,Brown,Orange,Purple,Pink,Gray}; Print["Colores asignados: "]; Do[Print["Vértice ", i, ",", color ",colores[[colorear[[i]]]]]; ,{i,Length[colorear]}]; </pre> | adyacencia del grafo. Identificamos los colores. |
| <pre> GraphPlot[matrizadyacencia, VertexRenderingFunction->({COLORES[[colorear[[#2]]]], Disk[#1,0.06`],Black, Text[#2,{#1[[1]]+.1,#1[[2]]+.1}]\}&)]] </pre> | Salida de resultados. |

Función 8.3. Coloración óptima de un grafo.

En el paquete `Combinatorica` de la versión 6 de Mathematica, disponemos de las funciones:

ChromaticNumber[G] , **MinimumVertexColoring[G]** y
MinimumVertexColoring[G,k]

la primera calcula el número cromático del grafo asociado al objeto de tipo grafo “G”, la segunda una coloración óptima y la tercera, si existe, una “k”-coloración.

Ejemplo 8.1. Calculamos una coloración óptima y el número cromático del grafo con matriz de adyacencia A .

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

Introducimos la matriz en Mathematica:

In[]:= **matrizadyacencia=**{ {0,1,1,0,0,0,0,0}, {1,0,1,1,1,0,0,0},
{1,1,0,0,1,1,0,0}, {0,1,0,0,1,0,1,0}, {0,1,1,1,0,1,0,0},
{0,0,1,0,1,0,0,1}, {0,0,0,1,0,0,0,1}, {0,0,0,0,1,1,0} };

Definimos las funciones 8.1, 8.2. y 8.3.:

In[]:= NCOLORACIONES[matrizadyacencia_,lambda_]:=Module[{coloracionestemp,t,h,i,j,colorady,k},

⋮ ⋮

In[]:= NUMEROGRAMATICO[matrizadyacencia_]:=Module[{},

⋮ ⋮

In[]:= DIBUJAR[matrizadyacencia_]:=Module[{i},

⋮ ⋮

Calculamos el número cromático:

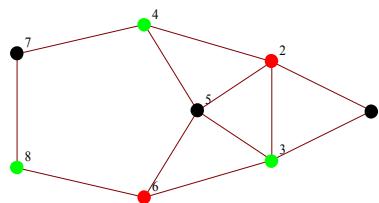
In[]:= NUMEROGRAMATICO[matrizadyacencia]

Out[] = 3

Calculamos una coloración óptima:

In[]:= DIBUJAR[matrizadyacencia]

*Out[] = Colores asignados:
 Vértice 1, color Negro
 Vértice 2, color Rojo
 Vértice 3, color Verde
 Vértice 4, color Verde
 Vértice 5, color Negro
 Vértice 6, color Rojo
 Vértice 7, color Negro
 Vértice 8, color Verde*



Y si usamos ChromaticNumber[] y MinimumVertexColoring[]:

In[]:= <<Combinatoria`

*In[]:= ChromaticNumber[
 FromAdjacencyMatrix[matrizadyacencia]]*

```

Out[]=
3

In[]:= MinimumVertexColoring[
FromAdjacencyMatrix[matrizadyacencia]]

Out[]=
{1,2,3,3,1,2,1,3}

```

que coincide con la calculada antes si identificamos el color “1” con el negro, el “2” con el rojo y el “3” con el verde.

□

Ejemplo 8.2. Calcular el número de 4-coloraciones que existen del grafo del ejemplo anterior y determinar todas sus coloraciones óptimas.

```

In[]:= matrizadyacencia={{0,1,1,0,0,0,0,0},{1,0,1,1,1,0,0,0},
{1,1,0,0,1,1,0,0},{0,1,0,0,1,0,1,0},{0,1,1,1,0,1,0,0},
{0,0,1,0,1,0,0,1},{0,0,0,1,0,0,0,1},{0,0,0,0,1,1,0}};

```

Definimos la función 8.1.:

```

In[]:= NCOLORACIONES[matrizadyacencia_,lambda_]:=Module[
{coloracionestemp,t,h,i,j,colorady,k},
⋮
⋮

```

Calculamos el número de 4-coloraciones:

```

In[]:= Length[Complement][
NCOLORACIONES[matrizadyacencia,4],
NCOLORACIONES[matrizadyacencia,3]]]

```

```
Out[]=
1278
```

Determinamos las coloraciones óptimas:

```

In[]:= coloraciones=NCOLORACIONES[matrizadyacencia,3]

Out[]=
{{1,2,3,3,1,2,1,3},{1,2,3,3,1,2,2,1},
{1,2,3,3,1,2,2,3},{1,3,2,2,1,3,1,2},
{1,3,2,2,1,3,3,1},{1,3,2,2,1,3,3,2},
{2,1,3,3,2,1,1,2},{2,1,3,3,2,1,1,3},
{2,1,3,3,2,1,2,3},{2,3,1,1,2,3,2,1},
{2,3,1,1,2,3,3,1},{2,3,1,1,2,3,3,2},
{3,1,2,2,3,1,1,2},{3,1,2,2,3,1,1,3},
{3,1,2,2,3,1,3,2},{3,2,1,1,3,2,2,1},
{3,2,1,1,3,2,2,3},{3,2,1,1,3,2,3,1}}

```

Para interpretar más cómodamente los resultados podemos escribir:

```
In[]:=      colores={"Negro","Rojo", "Verde", "Azul",
           "Amarillo","Cyan","Marrón","Naranja",
           "Púrpura","Rosa","Gris"};
Do[
 Print["COLORACIÓN ",j,":"];
 Print[
  TableForm[
   Table[{Table[i,{i,Dimensions[matrizadyacencia][[1]]}],
          Table[colores[[coloraciones[[j]][[i]]]],
          {i,Dimensions[matrizadyacencia][[1]]}]}]];
 ,{j,Length[coloraciones]}]

Out[]=
COLORACIÓN 1 :
 1     2     3     4     5     6     7     8
 Negro  Rojo  Verde  Verde  Negro  Rojo  Negro  Verde
COLORACIÓN 2 :
 1     2     3     4     5     6     7     8
 Negro  Rojo  Verde  Verde  Negro  Rojo  Negro  Verde
:
:
```

□

Obviamente el método combinatorio no es el más eficaz, aunque permite calcular siempre una coloración óptima, como comprobamos en el siguiente ejemplo:

Ejemplo 8.3. Calcular el número cromático de K_8 .

Introducimos las funciones 8.1. y 8.2.:

```
In[]:=      NCOLORACIONES[matrizadyacencia_,lambda_]:=Module[
 {coloracionestemp,t,h,i,j,colorady,k},
 :
]

In[]:=      NUMEROOCROMATICO[matrizadyacencia_]:=Module[{},
 :
]
```

Calculamos el número cromático:

```
In[]:=      n=8;
K=Table[If[i==j,0,1],{i,n},{j,n}];
Timing[NUMEROOCROMATICO[K]]

Out[]={146.625 Second, 8}
```

Obsérvese que el grafo es de tan sólo 8 vértices y el tiempo empleado en el cálculo es de más de 2 minutos.

□

Ejemplo 8.4. Representar gráficamente todos los grafos 4-cromáticos de 6 vértices distintos salvo isomorfismo.

Usando las funciones 5.19., 5.20. (consideraremos las diagonales principales de las potencias segunda, tercera y cuarta para comprobar si dos matrices de adyacencia pertenecen a la misma clase de isomorfía) y el programa 5.18. calculamos las 156 clases de isomorfía de matrices de adyacencia de grafos de 6 vértices.

In[]:= AñadirLado2[matrizadyacencia_]:=Module[{i,j,matriz},

⋮ ⋮

In[]:= QUITARISOMORFOS2[listamatrices_]:=Module[

⋮ ⋮

In[]:= n=6;
nLados=15;
matrizadyacencia=Table[0,{i,n},{j,n}];

⋮ ⋮

Out[]=

⋮ ⋮

Totales: 156 Tiempo empleado: 0.219

Ahora comprobamos cuáles de ellos son 4-cromáticos con las funciones 8.1. y 8.2. y los representamos gráficamente:

In[]:= NCOLORACIONES[matrizadyacencia_,lambda_]:=Module[
{coloracionestemp,t,h,i,j,colorady,k},

⋮ ⋮

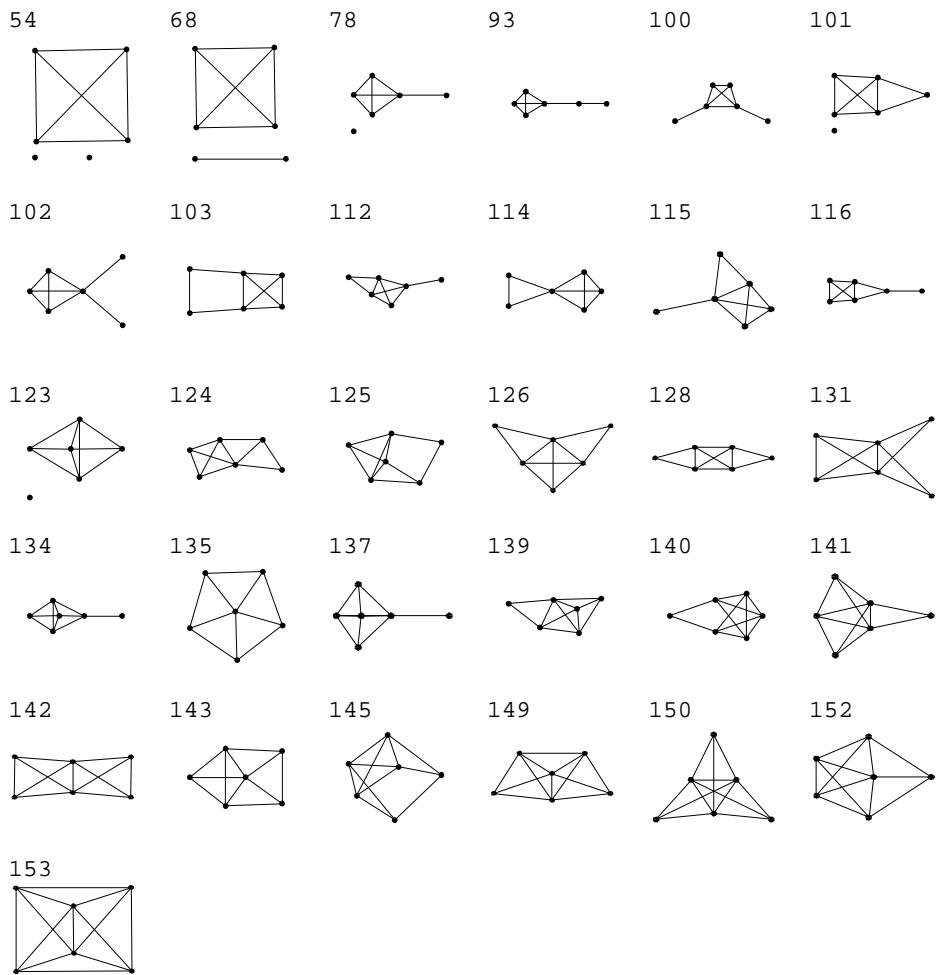
In[]:= NUMEROGRAMATICO[matrizadyacencia_]:=Module[{},

⋮ ⋮

In[]:= Do[
If[NUMEROGRAMATICO[listamatrices1[[i]]]==4,
Print[i];
Print[GraphPlot[listamatrices1[[i]],
PlotStyle->{Black,PointSize[.05]}]];];

,{i,Length[listamatrices1]}]

Out[*i*]=



□

1.2. COLORACIÓN: OPTIMIZACIÓN Y EFICACIA

Las nuevas funciones que trae versión 6 de Mathematica: ChromaticNumber[] y MinimumVertexColor[], son más rápidas que las expuestas en 1.1. y para ciertos casos son recomendables. En el ejercicio 8.49. se proponen algunas ideas para acelerar el algoritmo combinatorio. Ahora estudiamos otros algoritmos más eficaces que los de la sección anterior, pero que no aseguran una coloración óptima.

1.2.1. ALGORITMO DE COLORACIÓN I

Este algoritmo suele llamarse algoritmo de coloración secuencial y es de tipo voraz. La idea del mismo es la siguiente: definimos una lista de colores identificados por números enteros positivos, vamos cogiendo vértices (en principio al azar, aunque en la función 8.4. en realidad se hace por el orden de sus índices), comprobamos los colores usados en los vértices adyacentes coloreados hasta el momento y el primer color de la lista de colores no usado por estos será el que elegiremos para colorearlo. Obviamente según la asignación de índices que hagamos la coloración obtenida será distinta.

| FUNCIÓN | COMENTARIOS |
|---|--|
| <pre>COLOREAR[matrizadyacencia_]:=Module[{colores,coloreartemp,COLORES,color,i,j}, colores={"Negro","Rojo","Verde","Azul", "Amarillo","Cyan","Marrón","Naranja", "Púrpura","Rosa","Gris"}; COLORES={Black,Red,Green,Blue,Yellow, Cyan,Brown,Orange,Purple,Pink,Gray};</pre> | Se seguirá el orden de asignación de colores que introduzcamos, paralelamente se introducen los códigos de color para la representación gráfica. |
| <pre>colorear={1};color=1; Do[coloreartemp=colorear; Do[If[matrizadyacencia[[j,i]]==1, coloreartemp=Complement[coloreartemp, {colorear[[j]]}]; ,{j,i-1}]; If[coloreartemp=={}, color++;AppendTo[colorear,color]; , AppendTo[colorear,coloreartemp[[1]]]; ,{i,2,Dimensions[matrizadyacencia][[1]]}];</pre> | Recorremos todos los vértices y asignamos el menor número de colores. El orden de recorrido de los vértices será el de sus índices. |
| <pre>Print["Número cromático: ",color]; Print["Colores asignados: "]; Do[Print["Vértice ", i, ", color ", colores[[colorear[[i]]]]],{i,Length[colorear]}]; GraphPlot[matrizadyacencia, VertexRenderingFunction-> ({COLORES[[colorear[[#2]]]],Disk[#1,0.06`], Black,Text[#2,{#1[[1]]+.1,#1[[2]]+.1}]})&]</pre> | Salida de resultados. |

Función 8.4. Coloración de un grafo.

Los siguientes ejemplos muestran su funcionamiento, en 8.5. vemos algunos casos donde el algoritmo calcula una coloración óptima, por otra parte 8.6. nos da una idea de los grafos donde el algoritmo no calcula una coloración óptima y la razón de que éste falle.

Ejemplo 8.5. Analizamos el número cromático usando la función 8.4. de los siguientes grafos: $K_{3,3}$, K_5 y el pentágono.

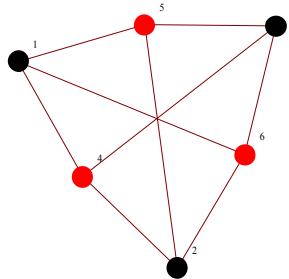
In//:= COLOREAR[matrizadyacencia_]:=Module[
{colores,coloreartemp,COLORES,color,i,j},

⋮ ⋮

a) $K_{3,3}$:

```
In[]:= n=3;m=3;
matrizadyacencia=Table[If[(i≤n &&j≤n)|| (i>n &&j>n),0,1]
,{i,n+m},{j,n+m}];
COLOREAR[matrizadyacencia]
```

```
Out[]= Colores asignados:
Vértice 1, color Negro
Vértice 2, color Negro
Vértice 3, color Negro
Vértice 4, color Rojo
Vértice 5, color Rojo
Vértice 6, color Rojo
```

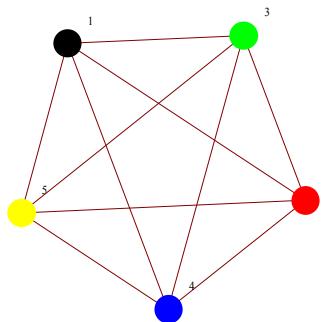


Al obtener una 2-coloración podemos concluir que su número cromático en 2 porque es evidente que existen al menos dos vértices adyacentes.

b) K_5 :

```
In[]:= matrizadyacencia=Table[If[i==j,0,1],{i,5},{j,5}];
COLOREAR[matrizadyacencia]
```

```
Out[]= Colores asignados:
Vértice 1, color Negro
Vértice 2, color Rojo
Vértice 3, color Verde
Vértice 4, color Azul
Vértice 5, color Amarillo
```

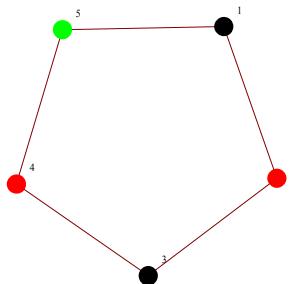


Un grafo completo K_n tiene como número cromático al número n de vértices del mismo, ya que al ser todos los vértices adyacentes nos vemos obligados a usar para colorearlo tantos colores como vértices tenga el grafo.

c) El pentágono:

```
In[7]:= matrizadyacencia={\{0,1,0,0,1\},\{1,0,1,0,0\},\{0,1,0,1,0\},
{\0,0,1,0,1\},\{1,0,0,1,0\}};
COLOREAR[matrizadyacencia]
```

```
Out[7]= Colores asignados:
Vértice 1, color Negro
Vértice 2, color Rojo
Vértice 3, color Negro
Vértice 4, color Rojo
Vértice 5, color Verde
```

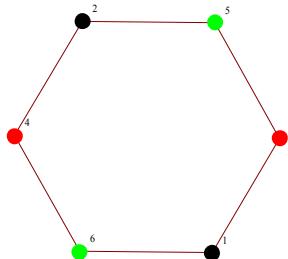


Los polígonos de un número par de vértices serán 2-cromáticos, mientras que los de un número impar serán 3-cromáticos.

□

Ejemplo 8.6. Colorearemos un hexágono utilizando la función 8.4., y comprobaremos que esta función no siempre determina una coloración óptima.

```
In[]:= COLOREAR[matrizadyacencia_]:=Module[{colores,coloreartemp,COLORES,color,i,j},  
⋮  
⋮  
In[]:= matrizadyacencia={{0,0,1,0,0,1},{0,0,0,1,1,0},{1,0,0,0,1,0},  
{0,1,0,0,0,1},{0,1,1,0,0,0},{1,0,0,1,0,0}};  
COLOREAR[matrizadyacencia]  
  
Out[]= Colores asignados:  
Vértice 1, color Negro  
Vértice 2, color Negro  
Vértice 3, color Rojo  
Vértice 4, color Rojo  
Vértice 5, color Verde  
Vértice 6, color Verde
```



Es claro que su número cromático es 2 y no 3. □

1.2.2. ALGORITMO DE COLORACIÓN II

Partimos del algoritmo anterior, pero no elegimos los vértices al azar, seguimos un orden. Coloreamos un vértice elegido al azar y después vamos coloreando los vértices adyacentes a los ya coloreados, necesitamos que el grafo sea conexo, lo cual no es restrictivo, pues podremos aplicarlo a todas las componentes conexas. Aunque mejora 1.2.1. seguimos sin poder garantizar la obtención de una coloración óptima.

| FUNCIÓN | COMENTARIOS |
|---|--|
| <pre>COLOREAR2[matrizadyacencia_]:=Module[{colores, vertices,color,cont,COLORES,adyacen,colorusado, vert,coloronoady,j,k,i}, colores={"Negro","Rojo", "Verde", "Azul", "Amarillo","Cyan", "Marrón","Naranja", "Púrpura","Rosa","Gris"}; COLORES={Black,Red,Green,Blue,Yellow, Cyan,Brown,Orange,Purple,Pink,Gray};</pre> | La función tendrá por entrada la matriz de adyacencia de un grafo conexo. |
| | Se seguirá el orden de asignación de colores que introduzcamos, paralelamente se introducen los códigos de color para la representación gráfica. |

| | |
|---|---|
| <pre> vertices={1}; colorear={1}; color=1; cont=1; While[Length[vertices]< Dimensions[matrizadyacencia][[1]], adyacen=Position[matrizadyacencia[[vertices[[cont]]]],1]; adyacen=Complement[adyacen, Table[{vertices[[k]]},{k,Length[vertices]}]]; Do[If[Intersection[adyacen[[i]],vertices]=={}, vert=adyacen[[i]][[1]]; colorusado={}; Do[If[matrizadyacencia[[adyacen[[i]]][[1]], vertices[[j]]]==1, colorusado=Union[colorusado, {colorear[[j]]}]; ,{j,Length[vertices]}]; coloresnoady=Complement[Table[cc,{cc,color}],colorusado]; If[coloresnoady=={}, color++;AppendTo[colorear,color]; , AppendTo[colorear,Sort[coloresnoady][[1]]];]; AppendTo[vertices,vert];]; ,{i,Length[adyacen]}]; cont++;]; coloreartemp=Table[0,{i,Length[colorear]}]; Do[coloreartemp[[vertices[[i]]]]=colorear[[i]], ,{i,Length[colorear]}]; colorear=coloreartemp; Print["Colores asignados: "]; Do[Print["Vértice ", i, ", color ", colores[[colorear[[i]]]]] ,{i,Length[colorear]}]; Print[GraphPlot[matrizadyacencia, VertexRenderingFunction-> ({COLORES[[colorear[[#2]]]],Disk[#1,0.06'], Black,Text[#2,{#1[[1]]+.1,#1[[2]]+.1}]}&)]]; </pre> | <p>Igual que la función 8.4. pero alteramos el orden en que vamos coloreando, elegimos vértices adyacentes a los ya coloreados.</p> |
| | <p>Salida de resultados.</p> |

Función 8.5. Coloración de un grafo.

Aunque mejora a 8.4. y es capaz de calcular coloraciones óptimas donde 8.4. fallaba (ejemplo 8.7.) el algoritmo sigue sin asegurarnos la coloración óptima (ejemplo 8.8.).

Ejemplo 8.7. Para comprobar que funciona mejor que la función 8.4. vamos a colorear usando la función 8.5. (COLOREAR2[]) un grafo de 10 vértices, compuesto por dos componentes conexas, una de ellas será un hexágono y la otra un cuadrado:

```
In]:= matrizadyacencia={{0,0,1,0,0,1,0,0,0,0},{0,0,0,1,1,0,0,0,0,0},
{1,0,0,0,1,0,0,0,0,0},{0,1,0,0,0,1,0,0,0,0},{0,1,1,0,0,0,0,0,0,0},
{1,0,0,1,0,0,0,0,0,0},{0,0,0,0,0,0,0,1,1},{0,0,0,0,0,0,0,1,1},
{0,0,0,0,0,1,1,0,0},{0,0,0,0,0,1,1,0,0}};
```

Determinamos si es conexo y calculamos sus componentes conexas con las funciones 7.13. y 7.14.

```

In[]:= CONEXO[A_]:=Module[{i,j,B},
                           :           :
                           :           :];
In[]:= COMPONENTESCONEXAS[matrizadyacencia_]:=Module[
                           :           :
                           :           :];

```

A cada componente conexa le aplicamos la función 8.5.

```

In[]:= COLOREAR2[matrizadyacencia_]:=  

          ;           ;  

In[]:= If[CONEXO[matrizadyacencia],  

        COLOREAR2[matrizadyacencia];  

      , comp=COMPONENTESCONEXAS[matrizadyacencia]  

      Do[  

        COLOREAR2[  

          Table[matrizadyacencia[[comp[[h]][[i]],comp[[h]][[j]]],  

                {i,Length[comp[[h]]]}, {j,Length[comp[[h]]]}];  

          ,{h,Length[comp]}];  

      ]  

Out[]= { {1,2,3,4,5,6}, {7,8,9,10} }  

Colores asignados:  

Vértice 1, color Negro  

Vértice 2, color Rojo  

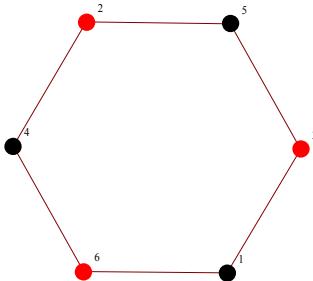
Vértice 3, color Rojo  

Vértice 4, color Negro  

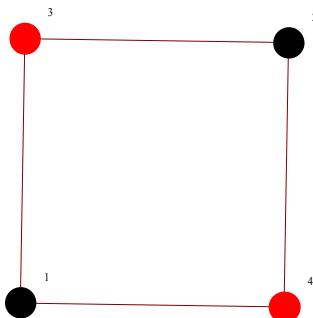
Vértice 5, color Negro  

Vértice 6, color Rojo

```



Colores asignados:
 Vértice 1, color Negro
 Vértice 2, color Negro
 Vértice 3, color Rojo
 Vértice 4, color Rojo



La primera componente conexa se corresponde con el grafo del ejemplo 8.6. y la función 8.5. (COLOREAR2[]) si ha obtenido una coloración óptima, por otra parte a la vista de las coloraciones obtenidas es obvio que el número cromático es 2.

□

Ejemplo 8.8. En este ejemplo comprobaremos como la función COLOREAR2[] (8.5.) no siempre obtendrá una coloración óptima. La aplicamos al grafo del ejemplo 8.1.

In]:= matrizadyacencia={{0,1,1,0,0,0,0,0},{1,0,1,1,1,0,0,0},
{1,1,0,0,1,1,0,0},{0,1,0,0,1,0,1,0},{0,1,1,1,0,1,0,0},
{0,0,1,0,1,0,0,1},{0,0,0,1,0,0,0,1},{0,0,0,0,0,1,0}};

Comprobamos si el grafo es conexo con 7.13.:

In]:= CONEXO[A_]:=Module[{i,j,B},

⋮

⋮

In[]:= CONEXO[matrizadyacencia]

Out[]:= True

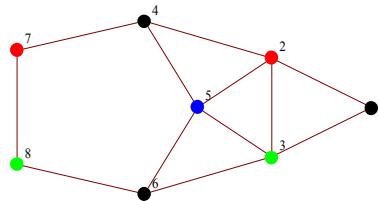
Calculamos la coloración con la función 8.5.:

In[]:= COLOREAR2[matrizadyacencia]:=

⋮ ⋮

In[]:= COLOREAR2[matrizadyacencia]

In[]:= Colores asignados:
 Vértice 1, color Negro
 Vértice 2, color Rojo
 Vértice 3, color Verde
 Vértice 4, color Negro
 Vértice 5, color Azul
 Vértice 6, color Negro
 Vértice 7, color Rojo
 Vértice 8, color Verde



La coloración no es óptima, pues este grafo es 3-cromático según pudimos comprobar en el ejemplo 8.1.

□

1.2.3. ALGORITMO DE COLORACIÓN III

También es un algoritmo voraz, esta vez coloreamos de un mismo color todos los vértices posibles, pasamos al siguiente color y hacemos lo mismo, así sucesivamente hasta que queden todos los vértices coloreados.

| FUNCIÓN | COMENTARIOS |
|--|--|
| <pre>COLOREAR3[matrizadyacencia_]:=Module[{color,j,t,k,adyacentes,i,nocolorear}, colorear=Table[0 ,{i,Dimensions[matrizadyacencia][[1]]}]; colorear[[1]]=1; color=1; t=1;</pre> | La función tendrá por argumento a la matriz de adyacencia del grafo. |

| | |
|--|---|
| While[t<Dimensions[matrizadyacencia][[1]], | Mientras queden vértices sin colorear el bucle seguirá dando vueltas. |
| Do[| Recorremos todos los vértices del grafo. |
| If[colorear[[k]]==0, nocolorear=False; adyacentes=Position[matrizadyacencia[[k]],1]; Do[If[colorear[[adyacentes[[j]][[1]]]]==color, nocolorear=True;Break[]]; ,{j,Length[adyacentes]}]; If![nocolorear,colorear[[k]]=color;t++];]; | Coloreamos del nuevo color todos los vértices que podamos y no estén ya coloreados. |
| ,{k,Dimensions[matrizadyacencia][[1]]}]; color++;]; | |
| colores={"Negro","Rojo", "Verde", "Azul", "Amarillo","Cyan", "Marrón","Naranja", "Púrpura","Rosa","Gris"}; COLORES={Black,Red,Green,Blue,Yellow, Cyan,Brown,Orange,Purple,Pink,Gray}; Print["Colores asignados: "]; Do[Print["Vértice ", i, ", color ", colores[[colorear[[i]]]]] ,{i,Length[colorear]}]; GraphPlot[matrizadyacencia, VertexRenderingFunction-> ({COLORES[[colorear[[#2]]]],Disk[#1,0.06'], Black,Text[#2,{#1[[1]]+.1,#1[[2]]+.1}]})&]]; | Salida de resultados. |

Función 8.6. Coloración de un grafo.

En este tercer algoritmo seguimos una metodología distinta a 8.5. y 8.6., pero por desgracia sigue sin asegurarnos una coloración óptima (ejemplo 8.10.).

Ejemplo 8.9. Calcular una coloración óptima de $K_{3,4}$ utilizando la función 8.6.

Calculamos con la función 6.13., el grafo bipartito completo $K_{3,4}$ y lo coloreamos con la función 8.6.:

In]:= BCadyacencia[n_,m_]:=Table[If[(i≤n &&j≤n)||
(i>n &&j>n),0,1],{i,n+m},{j,n+m}];

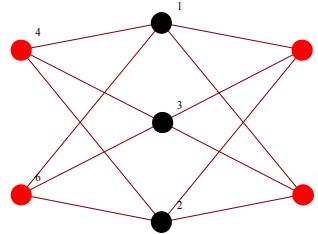
In]:= COLOREAR3[matrizadyacencia_]:=Module[
{color,j,t,k,adyacentes,i,nocolorear},

⋮ ⋮

In]:= A=BCadyacencia[3,4];
COLOREAR3[matrizadyacencia]

Out[]=

Colores asignados:
 Vértice 1, color Negro
 Vértice 2, color Negro
 Vértice 3, color Negro
 Vértice 4, color Rojo
 Vértice 5, color Rojo
 Vértice 6, color Rojo



Por tanto es una coloración óptima y su número cromático es 2.

□

Ejemplo 8.10. Igual que ocurrió con 8.4. y 8.5., este algoritmo no siempre determinará una coloración óptima. Lo comprobamos aplicándolo al grafo del ejemplo 8.1.:

In[]:=

COLOREAR3[matrizadyacencia]:=Module[
{color,j,t,k,adyacentes,i,nocolorear},

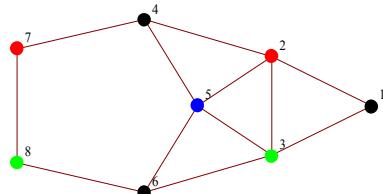
⋮ ⋮

In[]:=

matrizadyacencia={{0,1,1,0,0,0,0,0},{1,0,1,1,1,0,0,0},
{1,1,0,0,1,1,0,0},{0,1,0,0,1,0,1,0},{0,1,1,1,0,1,0,0},
{0,0,1,0,1,0,0,1},{0,0,0,1,0,0,0,1},{0,0,0,0,1,1,0}};
COLOREAR3[matrizadyacencia]

In[]:=

Colores asignados:
 Vértice 1, color Negro
 Vértice 2, color Rojo
 Vértice 3, color Verde
 Vértice 4, color Negro
 Vértice 5, color Azul
 Vértice 6, color Negro
 Vértice 7, color Rojo
 Vértice 8, color Verde



1

En el ejercicio 8.7. se proponen nuevas mejoras sobre estos algoritmos que pretenden acercarse lo máximo posible a una coloración óptima.

1.2.4. LA FUNCIÓN `VerxtexColoring[]`:

En la versión 6 y siguientes de Mathematica y dentro del paquete `<<Combinatorica``, también disponemos de una función que calcula de forma eficaz una buena coloración, aunque no necesariamente óptima del grafo asociado al objeto de tipo grafo “G”:

VertexColoring[G]

Concluimos el epígrafe con otro ejemplo en donde aplicamos todos los métodos de coloración vistos a un mismo grafo.

Ejemplo 8.11. Calcular las coloraciones por todos los métodos estudiados y comprobar su eficacia para el grafo de la ilustración 8.1.:

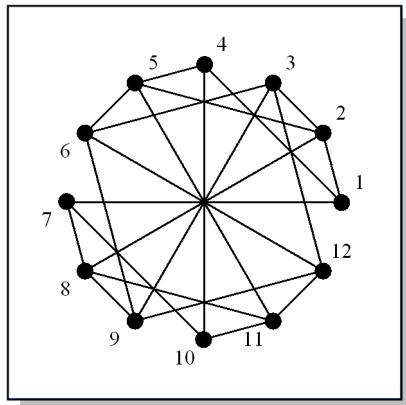


Ilustración 8.1.

Introducimos su matriz de adyacencia:

```
In[]:= A={{0,1,0,1,0,0,1,0,0,0,0,0},{1,0,1,0,1,0,0,1,0,0,0,0},  
{0,1,0,0,0,1,0,0,1,0,0,1},{1,0,0,0,1,0,0,0,0,1,0,0},  
{0,1,0,1,0,1,0,0,0,1,0},{0,0,1,0,1,0,0,0,1,0,0,1}}
```

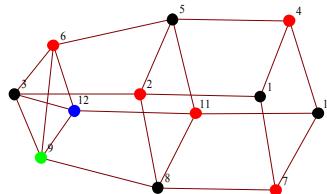
```
{1,0,0,0,0,0,1,0,1,0,0},{0,1,0,0,0,0,1,0,1,0,1,0},  

{0,0,1,0,0,1,0,1,0,0,0,1},{0,0,0,1,0,0,1,0,0,0,1,0},  

{0,0,0,0,1,0,0,1,0,1,0,1},{0,0,1,0,0,1,0,0,1,0,1,0}};
```

Con los cuatro métodos DIBUJAR[] (8.3.), COLOREAR[] (8.4.), COLOREAR2[] (8.5.) y COLOREAR3[] (8.6.) obtenemos la misma coloración:

Colores asignados:
 Vértice 1, color Negro
 Vértice 2, color Rojo
 Vértice 3, color Negro
 Vértice 4, color Rojo
 Vértice 5, color Negro
 Vértice 6, color Rojo
 Vértice 7, color Rojo
 Vértice 8, color Negro
 Vértice 9, color Verde
 Vértice 10, color Negro
 Vértice 11, color Rojo
 Vértice 12, color Azul



Usamos Timing[] para comprobar el tiempo empleado por cada función:

In[]:= NCOLORACIONES[matrizadyacencia_,lambda_]:=Module[

{coloracionestemp,t,h,i,j,colorady,k},

⋮ ⋮

In[]:= NUMEROGRAMATICO[matrizadyacencia_]:=Module[{},

⋮ ⋮

In[]:= DIBUJAR[matrizadyacencia_]:=Module[{i},

⋮ ⋮

In[]:= Timing[DIBUJAR[A]][[1]]

Out[]= 23.734 Second

In[]:= COLOREAR[matrizadyacencia_]:=Module[

```

{colores,coloreartemp,COLORES,color,i,j},

⋮ ⋮

In]:=      Timing[COLOREAR[A]][[1]]

Out]=      0.016 Second

In]:=      COLOREAR2[matrizadyacencia_]:=Module[{colores,
⋮ ⋮

In]:=      Timing[COLOREAR2[A]][[1]]

Out]=      0.047 Second

In]:=      COLOREAR3[matrizadyacencia_]:=Module[
{color,j,t,k,adyacentes,i,nocolorear},
⋮ ⋮

In]:=      Timing[COLOREAR3[A]][[1]]

Out]=      0. Second

In]:=      <<Combinatorica`

In]:=      Timing[VertexColoring[FromAdjacencyMatrix[A]]][[1]]

Out]=      0. Second

```

Y podemos observar como todos los algoritmos no combinatorios son muy eficaces y emplean tiempos similares.

□

1.3. EL POLINOMIO CROMÁTICO

Sea G un grafo plano (no dirigido), llamaremos polinomio cromático de G , a la función polinomial en la variable λ o polinomio $p(\lambda)$ que determina el número total de formas distintas que hay de colorear el grafo G con un máximo de $\lambda \in \mathbb{N}$ colores, esto es:

$$p(\lambda) = \text{número de } \mu\text{-coloraciones distintas de } G \text{ con } \mu \leq \lambda.$$

Es habitual denotarlo por $p(G, \lambda)$.

Para el grafo:

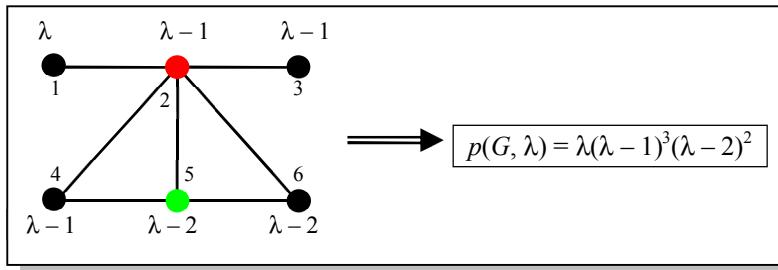


Ilustración 8.2.

Obsérvese la construcción del polinomio cromático en la ilustración 8.2., si empezamos por colorear el vértice 1, éste podrá ser coloreado con cualquiera de los λ colores, si continuamos por 2, éste se podrá colorear con $\lambda - 1$ colores, todos menos el usado en 1 con el que es adyacente, continuamos por 3 que sólo podrá ser coloreado por $\lambda - 1$, de nuevo todos menos el usado en 2 y así sucesivamente obtenemos el polinomio cromático.

Para un (n, m) -grafo G con k componentes conexas G_1, G_2, \dots, G_k , existen ciertas restricciones y propiedades del polinomio característico que nos pueden ayudar en su cálculo y que resumimos a continuación:

- i. El grado de $p(G, \lambda)$ es n .
- ii. El coeficiente líder de $p(G, \lambda)$ es 1.
- iii. El coeficiente de λ^{n-1} es $-m$.
- iv. El término independiente de $p(G, \lambda)$ es 0.
- v. $p(G, \lambda) = \prod_{i=1}^k p(G_i, \lambda)$.
- vi. Si m es distinto de 0, entonces la suma de los coeficientes de $p(G, \lambda)$ es 0.

Usando 8.1. el polinomio cromático vendría dado directamente por:

```
p[matrizadyacencia ,lambda]:=
Length[NCOLORACIONES[matrizadyacencia,lambda]];
```

Podemos conseguir la expresión del polinomio utilizando la función de Mathematica:

InterpolatingPolynomial[]³⁵

Si G es un (n, m) -grafo, entonces bastará con determinar $n + 1$ valores del polinomio cromático para poder encontrarlo por interpolación, por tanto, calcularemos el número de coloraciones para $0, 1, \dots, (n + 1)$ colores:

³⁵ La función **InterpolatingPolynomial[]** calcula el polinomio cromático por interpolación, esta función no es frecuente en otros lenguajes, por tanto para trasladar este método a otro lenguaje previamente deberemos programarla.

| FUNCIÓN | COMENTARIOS |
|--|--|
| NCOLORACIONES[matrizadyacencia_,lambda_]:=... | Definimos la función 8.1. |
| p[matrizadyacencia_,lambda_]:=Length[NCOLORACIONES[matrizadyacencia,lambda_]]; | |
| POLINOMIOCROMATICO[matrizadyacencia_]:=Module[{datos,j}, $\text{datos} = \{\};$ $\text{Do}[$ $\quad \text{AppendTo}[\text{datos}, \{j, p[\text{matrizadyacencia}, j]\}];$ $\quad , \{j, 0, \text{Dimensions}[\text{matrizadyacencia}][[1]]\};$ $\quad \text{Factor}[\text{InterpolatingPolynomial}[\text{datos}, x]]$ $\quad]$ | Por entrada tendrá la matriz de adyacencia del grafo. Calculamos el número de coloraciones para 0, 1, ..., "Dimensions[matrizadyacencia][[1]]" colores. |
| | Salida de resultados. |

Función 8.7. Polinomio cromático.

En la versión 6 o posterior, y dentro del paquete `Combinatorica`, disponemos de una nueva función:

ChromaticPolynomial[G, λ]

que determina el polinomio cromático $p(\lambda)$ del grafo asociado al objeto de tipo grafo "G".

Ejemplo 8.12. Calculamos el polinomio cromático del grafo de la ilustración 8.1.

In[]:= **NCOLORACIONES[matrizadyacencia_,lambda_]:=Module[**
 $\quad \{\text{coloracionestemp}, t, h, i, j, \text{colorady}, k\},$

⋮ ⋮

In[]:= **p[matrizadyacencia_,lambda_]:=**
 $\quad \text{Length}[\text{NCOLORACIONES}[\text{matrizadyacencia}, \text{lambda}]]$

In[]:= **POLINOMIOCROMATICO[matrizadyacencia_]:=Module[**
 $\quad \{\text{datos}, j\},$

⋮ ⋮

In[]:= **matrizadyacencia={{0,1,0,0,0,0},{1,0,1,1,1,1},{0,1,0,0,0,0},**
 $\quad \quad \quad \{0,1,0,0,1,0\}, \{0,1,0,1,0,1\}, \{0,1,0,0,1,0\}};$
POLINOMIOCROMATICO[matrizadyacencia]

Out[] = $(-2+x)^2 (-1+x)^3 x$

Y con ChromaticPolynomial[]:

In[]:= **<<Combinatorica`**

In[]:= **Factor[ChromaticPolynomial[
FromAdjacencyMatrix[matrizadyacencia],x]]**

Out[] = $(-2+x)^2 (-1+x)^3 x$

□

El polinomio cromático de un grafo no conexo es el producto de los polinomios cromáticos de cada una de las componentes conexas, podemos acelerar la función POLINOMIOCROMATICO[] (8.7.) si calculamos el polinomio cromático de cada componente conexa y posteriormente los multiplicamos, obviamente la siguiente función no tiene sentido para grafos conexos.

| FUNCIÓN | COMENTARIOS |
|--|---|
| <code>NCOLORACIONES[matrizadyacencia_, lambda_]:=...</code> | Definimos la función 8.1. |
| <code>p[matrizadyacencia_,lambda_]:=</code> <code>Length[</code> <code>NCOLORACIONES[matrizadyacencia,lambda]</code> <code>];</code> | |
| <code>POLINOMIOCROMATICO[matrizadyacencia_]:=...</code> | Definimos la función 8.7. |
| <code>COMPONENTESCONEXAS[matrizadyacencia_]:=...</code> | Definimos la función 7.14. |
| <code>POLINOMIOCROMATICO2[matrizadyacencia_]:=</code> <code>Module[</code> <code>{comp,h,i,j},</code> <code>comp=COMPONENTESCONEXAS[</code> <code>matrizadyacencia];</code> <code>polinomio=1;</code> <code>Do[</code> <code>polinomio=polinomio*</code> <code>(POLINOMIOCROMATICO[</code> <code>Table[matrizadyacencia[[comp[[h]][[i]],</code> <code>comp[[h]][[j]]]],</code> <code>,{i,Length[comp[[h]]}],{j,Length[comp[[h]]}]],</code> <code>,{h,Length[comp]}];</code> <code>polinomio</code> <code>];</code> | La función tendrá como único argumento la matriz de adyacencia del grafo. |
| | |
| | Salida de resultados. |

Función 8.8. Polinomio cromático.

Ejemplo 8.13. Calculamos el polinomio cromático del grafo no conexo del ejemplo 8.6.

In[]:= **matrizadyacencia={{0,0,1,0,0,1,0,0,0,0},{0,0,0,1,1,0,0,0,0,0},
{1,0,0,0,1,0,0,0,0,0},{0,1,0,0,0,1,0,0,0,0},{0,1,1,0,0,0,0,0,0,0},
{1,0,0,1,0,0,0,0,0,0},{0,0,0,0,0,0,0,1,1},{0,0,0,0,0,0,0,1,1,1},
{0,0,0,0,0,1,1,0,0},{0,0,0,0,0,1,1,0,0}};**

Definimos las funciones 8.1., 8.7., 7.14. y 8.8.:

In[]:= **NCOLORACIONES[matrizadyacencia_,lambda_]:=Module[
{coloracionestemp,t,h,i,j,colorady,k},**

```

          :
          :

In]:=      p[matrizadyacencia_,lambda]:=Length[NCOLORACIONES[matrizadyacencia,lambda]]

In]:=      POLINOMIOCROMATICO[matrizadyacencia_]:=Module[
          {datos,j},
          :
          :

In]:=      COMPONENTESCONEXAS[matrizadyacencia_]:=Module[
          :
          :

In]:=      POLINOMIOCROMATICO2[matrizadyacencia_]:=Module[
          :
          :

In]:=      Timing[POLINOMIOCROMATICO2[matrizadyacencia]]
```

□

Al basarse en las soluciones combinatorias del problema de la coloración, este método no es muy eficaz. Podemos conseguir un algoritmo más eficaz que no esté basado en la solución combinatoria usando el siguiente resultado cuya demostración podemos encontrar en el epígrafe 11.6 de [29]:

Teorema 8.1. Teorema de descomposición para polinomios cromáticos. Sea $G = (W, F)$ un grafo conexo y sea $f = \{w_1, w_2\} \in F$ un lado, entonces:

$$p(G_f, \lambda) = p(G, \lambda) + p(G'_f, \lambda)$$

donde $G_f = (W, F - \{f\})$ y G'_f es el grafo que resulta de G al identificar los vértices w_1 y w_2 .

□

Podemos construir G_f y G'_f con Mathematica:

| FUNCIÓN | COMENTARIOS |
|--|--|
| D1[matrizadyacencia_,i_,j_]:= | La función tendrá tres entradas: la matriz de adyacencia del grafo y los dos vértices incidentes (identificados por sus subíndice) con el lado que pretendemos eliminar. |
| Table[If[(k1==i && k2==j) (k1==j&&k2==i), | Ponemos un cero en las coordenada (i, j) , (j, i) -ésimas de la matriz de adyacencia. |

| | |
|---|--|
| <pre> 0 , matrizadyacencia[[k1,k2]] ,{k1,Dimensions[matrizadyacencia][[1]]} ,{k2,Dimensions[matrizadyacencia][[2]]}] </pre> | |
|---|--|

Función 8.9. Eliminar un lado.

| FUNCIÓN | COMENTARIOS |
|--|--|
| <pre> D2[matrizadyacencia,_i,_j]:=Module[{k1,W,F,W1,F1,k,CONTi,CONTj}, W=Table[k1 ,{k1,Dimensions[matrizadyacencia][[1]]}]; F={}; Do[Do[If[matrizadyacencia[[k1,k2]]==1, AppendTo[F,k1→k2]]; ,{k1,k2}]; ,{k2,Dimensions[matrizadyacencia][[1]]}]; F1={}; Do[If[F[[k1]][[1]]≠j &&F[[k1]][[2]]≠j, F1=Union[F1,{F[[k1]]}]; , If[F[[k1]][[1]]≠i &&F[[k1]][[2]]≠i, If[F[[k1]][[1]]==j, F1=Union[F1,{i→F[[k1]][[2]]}]; , F1=Union[F1,{F[[k1]][[1]]→i}];]];]; ,{k1,Length[F]}]; W1=Complement[W,{j}]; F1=Table[Position[W1,F1[[k1]][[1]]][[1]]→ Position[W1,F1[[k1]][[2]]][[1]] ,{k1,Length[F1]}]; </pre> | <p>La función tendrá tres entradas: la matriz de adyacencia del grafo y los dos vértices que pretendemos identificar (identificados por sus subíndices).</p> <p>Calculamos el conjunto de vértices y el de lados.</p> <p>Identificamos los vértices.</p> |
| <pre> matrizadyacencia1=Table[0,{CONTi,Length[W1]}, {CONTj,Length[W1]}]; Do[matrizadyacencia1[[F1[[k,1]],F1[[k,2]]]]=1; matrizadyacencia1[[F1[[k,2]],F1[[k,1]]]]=1; ,{k,Length[F1]}]; matrizadyacencia1 </pre> | <p>Determinamos la matriz de adyacencia del grafo (“W1”, “F1”).</p> |
| <pre>] </pre> | <p>Salida de resultados.</p> |

Función 8.10. Identificación de dos vértices.

Ejemplo 8.14. Comprobamos el teorema 8.1, para el grafo de la ilustración 8.2., lo hacemos para el lado {2, 5}.

```

In]:=      matrizadyacencia={{0,1,0,0,1},{1,0,1,0,0},{0,1,0,1,0},
{0,0,1,0,1},{1,0,0,1,0}};

Definimos las funciones 8.1., 8.7., 8.9. y 8.10.:

In]:=      D1[matrizadyacencia_,i_,j_]:=  

          :  

          :  

In]:=      D2[matrizadyacencia_,i_,j_]:=Module[{k1,W,F,W1,F1,k,i,j},  

          :  

          :  

In]:=      NCOLORACIONES[matrizadyacencia_,lambda_]:=Module[  

          {coloracionestemp,t,h,i,j,colorady,k},  

          :  

          :  

In]:=      p[matrizadyacencia_,lambda_]:=  

          Length[NCOLORACIONES[matrizadyacencia,lambda]]  

In]:=      POLINOMIOCROMATICO[matrizadyacencia_]:=Module[  

          {datos,j},  

          :  

          :  


```

Calculamos el polinomio cromático usando el teorema 8.1.:

```

In]:=      Factor[  

          POLINOMIOCROMATICO[D1[matrizadyacencia,2,5]]-  

          POLINOMIOCROMATICO[D2[matrizadyacencia,2,5]]]

```

```

Out]=      (-2+x) (-1+x) x (2 - 2x + x^2)

```

En efecto,

```

In]:=      Factor[POLINOMIOCROMATICO[matrizadyacencia]]

```

```

Out]=      (-2+x) (-1+x) x (2 - 2x + x^2)

```

□

Ejemplo 8.15. Consideremos el grafo conexo G :

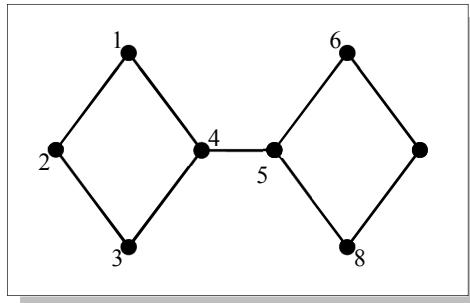


Ilustración 8.3.

Lo introducimos en Mathematica:

```
In]:= A1={{0,1,0,1,0,0,0,0},{1,0,1,0,0,0,0,0},{0,1,0,1,0,0,0,0},
{1,0,1,0,1,0,0,0},{0,0,0,1,0,1,0,1},{0,0,0,0,1,0,1,0},
{0,0,0,0,0,1,0,1},{0,0,0,0,1,0,1,0}};
```

El tiempo necesario para calcular el polinomio cromático directamente con **POLINOMIOCROMATICO[]** (8.7.) es muy elevado por tanto lo vamos a calcular usando el teorema 8.1. aplicándolo al lado {4, 5}, utilizaremos las funciones 8.9.y 8.10.:

```
In]:= D1[matrizadyacencia_,i_,j_]:=  
⋮ ⋮  
In]:= D2[matrizadyacencia_,i_,j_]:=Module[{k1,W,F,W1,F1,k,i,j},  
⋮ ⋮  
In]:= A2=D1[A1,4,5];  
B2=D2[A1,4,5];
```

“A2” es la matriz de adyacencia de un grafo no conexo, podemos calcular su polinomio cromático usando la función 8.8. (definimos también todas las funciones previas):

```
In]:= NCOLORACIONES[matrizadyacencia_,lambda_]:=Module[
{coloracionestemp,t,h,i,j,colorady,k},  
⋮ ⋮  
In]:= p[matrizadyacencia_,lambda_]:=Length[NCOLORACIONES[matrizadyacencia,lambda]]  
In]:= POLINOMIOCROMATICO[matrizadyacencia_]:=Module[
{datos,j},
```

$$\vdots \qquad \vdots$$

In[]:= **COMPONENTESCONEXAS[matrizadyacencia_]:=Module[**

$$\vdots \qquad \vdots$$

In[]:= **POLINOMIOCROMATICO2[matrizadyacencia_]:=**

$$\vdots \qquad \vdots$$

In[]:= **POLINOMIOCROMATICO2[A2]**

Out[] = $(-1 + x)^2 x^2 (3 - 3x + x^2)^2$

El grafo cuya matriz de adyacencia es “B2” sigue siendo demasiado complejo para calcular su polinomio cromático con 8.7., por lo que volvemos a aplicar el teorema 8.1. al lado $\{3, 4\}$.

In[]:= **A3=D1[B2,3,4];**
B3=D2[B2,3,4];

Podemos calcular el polinomio cromático de “B3”, pero a “A3” debemos aplicarle de nuevo el teorema:

In[]:= **POLINOMIOCROMATICO[B3]**

Out[] = $(-2 + x) (-1 + x^2) x (3 - 3x + x^2)$

In[]:= **A4=D1[A3,1,4];**
B4=D2[A3,1,4];
POLINOMIOCROMATICO2[A4]
POLINOMIOCROMATICO[B4]

Out[] = $(-1 + x)^3 x^2 (3 - 3x + x^2)$
 $(-1 + x)^3 x (3 - 3x + x^2)$

Por tanto, el polinomio cromático del grafo de partida es:

In[]:= **Factor[POLINOMIOCROMATICO2[A2]-**
 $((POLINOMIOCROMATICO2[A4]-$
POLINOMIOCROMATICO[B4])-
POLINOMIOCROMATICO[B3])]

Out[] = $(-1 + x)^3 x (3 - 3x + x^2)^2$

□

El teorema 8.1. puede ser usado en el sentido contrario, esto es, en vez de quitar lados, añadirlos, puesto que el polinomio cromático de K_n es conocido y además a mayor número de lados, menor número de combinaciones de coloraciones y mayor efectividad de POLINOMIOCROMATICO[]. Definimos otra función que en esta ocasión añada un lado:

| FUNCIÓN | COMENTARIOS |
|--|---|
| D3[matrizadyacencia _i,_j]:= | La función tendrá tres entradas: la matriz de adyacencia del grafo y los dos vértices incidentes (identificados por sus subíndices) con el lado que pretendemos añadir. |
| <pre data-bbox="251 419 634 457"> Table[If[(k1==i && k2==j) (k1==j&&k2==i), 1 , matrizadyacencia[[k1,k2]] ,{k1,Dimensions[matrizadyacencia][[1]]} ,{k2,Dimensions[matrizadyacencia][[2]]}] </pre> | Ponemos un “uno” en las coordenada (i, j) , (j, i) -ésimas de la matriz de adyacencia. |

Función 8.11. Añadir un lado.

Ejemplo 8.16. Calculamos el polinomio cromático del grafo:

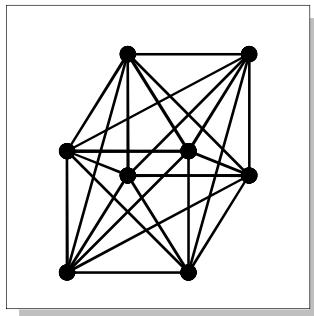


Ilustración 8.4.

```
In[]:= A1={{{0,1,1,1,1,1,0},{1,0,1,1,1,1,0,1},{1,1,0,1,0,1,1,1},  
{1,1,1,0,1,0,1,1},{1,1,0,1,0,1,1,1},{1,1,1,0,1,0,1,1},  
{1,0,1,1,1,1,0,1},{0,1,1,1,1,1,1,0}}};
```

Observemos que tiene muchos lados, son 8 vértices y es 6-regular, por tanto usaremos el teorema 8.1. y añadiremos más lados para que el cálculo sea más sencillo:

- Añadimos el lado $\{1, 8\}$, usamos las funciones 8.10. y 8.11.:.

```
In[]:= D2[matrizadyacencia_i,j]:=Module[{k1,W,F,W1,F1,k,i,j},
```

• • • • •

In[7]:= D3[matrizadyacencia ,i ,j]:=

$$\vdots \qquad \vdots$$

In[]:= **A2=D3[A1,1,8];**
B2=D2[A1,1,8];

Calculamos el polinomio cromático de “B2” con la función 8.7. (definimos también las funciones previas necesarias) y seguimos añadiendo lados a “A2”.

In[]:= **NCOLORACIONES[matrizadyacencia_,lambda_]:=Module[**
{coloracionestemp,t,h,i,j,colorady,k},

$$\vdots \qquad \vdots$$

In[]:= **p[matrizadyacencia_,lambda_]:=**
Length[NCOLORACIONES[matrizadyacencia,lambda_]]

In[]:= **POLINOMIOCROMATICO[matrizadyacencia_]:=Module[**
{datos,j},

$$\vdots \qquad \vdots$$

In[]:= **POLINOMIOCROMATICO[B2]**

Out[] = $(-3 + x)(-2 + x)x(-71 + 50x - 12x^2 + x^3)$

- Añadimos el lado $\{2, 7\}$ a “A2”:

In[]:= **A3=D3[A2,2,7];**
B3=D2[A2,2,7];

Calculamos el polinomio cromático de “B3” y seguimos añadiendo lados a “A3”.

In[]:= **POLINOMIOCROMATICO[B3]**

Out[] = $(-4 + x)(-3 + x)(-2 + x)(-1 + x)x(21 - 9x + x^2)$

- Añadimos el lado $\{3, 5\}$ a “A3”:

In[]:= **A4=D3[A3,3,5];**
B4=D2[A3,3,5];

Calculamos el polinomio cromático de “B4” y seguimos añadiendo lados a “A4”.

In[]:= **POLINOMIOCROMATICO[B4]**

Out[] = $(-5 + x)^2(-4 + x)(-3 + x)(-2 + x)(-1 + x)x$

- Añadimos el lado $\{4, 6\}$ a “A4”:

In[]:= **A5=D3[A4,4,6];**
B5=D2[A4,4,6];

Calculamos el polinomio cromático de “B5” y de “A5” que es K_8 .

In[]:= **POLINOMIOCROMATICO[B5]**

Out[]= $(-6 + x) (-5 + x) (-4 + x) (-3 + x) (-2 + x)$
 $(-1 + x) x$

In[]:= **POLINOMIOCROMATICO[A5]**

Out[]= $(-7 + x) (-6 + x) (-5 + x) (-4 + x) (-3 + x)$
 $(-2 + x) (-1 + x) x$

Por tanto el polinomio cromático será la suma de todos:

$$(-3 + x) (-2 + x) (-1 + x) x (465 - 392x + 125x^2 - 18x^3 + x^4)$$

□

Cualquiera de las dos técnicas anteriores facilita el cálculo del polinomio cromático, podríamos reiterar recursivamente la aplicación del teorema hasta llegar a grafos manejables desde el punto de vista del cálculo computacional del polinomio cromático, todo ello puede integrarse en una rutina y así se propone en el ejercicio 8.43.

2. GRAFOS PLANOS

Un grafo se dirá plano si podemos representarlo gráficamente en el plano de forma que no se corten sus lados.

Obsérvese que para determinar si un grafo es plano, podemos restringirnos al estudio de grafos no dirigidos, pudiendo también obviar la condición de multigrafo o la existencia de lazos (véase el ejercicio 8.45.).

Sea $G = (W, F)$ un grafo no orientado y sea $e = \{w_1, w_2\} \in F$ un lado, una subdivisión elemental de G , es otro grafo $G_1 = (W_1, F_1)$ donde $W_1 = W \cup \{w\}$ y $F_1 = (F - \{e\}) \cup \{\{w_1, w\}, \{w, w_2\}\}$ para algún $w \notin W$.

Dos grafos G_1 y G_2 diremos que son homeomorfos si existen dos grafos isomorfos H_1 y H_2 tales que de H_1 llegamos a G_1 y de H_2 a G_2 por sucesivas subdivisiones o bien son isomorfos.

Podemos caracterizar los grafos planos con el siguiente teorema cuya demostración podemos encontrar en el capítulo 8 del libro “Introduction to Combinatorial Mathematics” (Liu C.L., [47]).

Teorema 8.2. Teorema de Kuratowski. Un grafo es plano si y sólo si no contiene un subgrafo homeomorfo a K_5 o $K_{3,3}$.

□

Para comprobar si un grafo es plano con el ordenador, aplicaremos directamente el teorema de Kuratowski, esto es, primero eliminamos cualquier subdivisión posible (vértices de grado 2) y después comprobamos si los subgrafos del grafo son homeomorfos a K_5 o $K_{3,3}$. Como la comprobación es larga de construir, lo hacemos en varios pasos:

- Vamos a eliminar las subdivisiones, para ello quitaremos cualquier vértice v de grado dos y posteriormente usando D3[] (8.11.) añadiremos un lado que conecte los vértices adyacentes a v . También podemos eliminar los de grado 1 (vértices colgantes) y los de grado 0 (aislados) que no aportan nada para discernir si el grafo es plano, con ello ganaremos en eficacia.

| FUNCIÓN | COMENTARIOS |
|---|---|
| D3[matrizadyacencia,i,j]:=... | Definimos la función 8.11. |
| QUITARVERTICESGRADO210[matrizadyacencia_]:=Module[{grado2,t,W1,A,i,k,m,adya,k1,k2,k3,k4,adyacentes,j}, Nueva=matrizadyacencia; A=Nueva.Nueva; grado2=True;t=1; While[grado2, W1={}; grado2=False; If[Dimensions[A][[1]]>2, Do[If[A[[i,i]]==2 A[[i,i]]==1 A[[i,i]]==0, AppendTo[W1,i];grado2=True;] ,{i,Dimensions[A][[1]]}]; If[grado2, Sort[W1]; Do[m=W1[[Length[W1]-k+1]]; If[Dimensions[Nueva][[1]]>2, If[A[[m,m]]==2, adyacentes={}; Do[If[Nueva[[m,adya]]==1, AppendTo[adyacentes,adya]] ,{adya,Dimensions[Nueva][[1]]}]; Nueva= D3[Nueva,adyacentes[[1]],adyacentes[[2]]]]; Nueva=Table[If[k1≥m,k3=k1+1,k3=k1]; If[k2≥m,k4=k2+1,k4=k2]; Nueva[[k3,k4]] ,{k1,Dimensions[Nueva][[1]]-1}] | Por entrada tendrá la matriz de adyacencia del grafo a simplificar. Bucle que dará vueltas mientras existan vértices de grado 2, 1 o 0. Si el orden de la matriz es dos o menos paramos la simplificación. Comprobamos la existencia de vértices de grado 2, 1 o 0. Recorremos todos los vértices de grado 2, 1 y 0. Continuamos hasta orden 2. Si el vértice es de grado 2, añadimos un lado incidente con los vértices adyacentes al que hemos eliminado. Quitamos la fila y columna correspondientes al vértice que quitamos. |

| | |
|----------------------------------|---|
| ,{k2,Dimensions[Nueva][[2]]-1}]; | |
| A=Nueva.Nueva; | Actualizamos el valor de "A". |
|]; | |
| ,{k,Length[W1]}]; | Cierre del bucle que recorre los vértices a simplificar. |
|]; | |
|]; | |
| t++; | Contamos las vueltas necesarias. |
|]; | Cierre del bucle principal. |
| Nueva | Salida de la matriz de adyacencia del grafo simplificado. |
|] | |

Función 8.12. Grafos planos: eliminar subdivisiones y vértices de grado 1 o 0.

Podemos hacer esto mismo de otras formas, por ejemplo, pudimos identificar cada vértice de grado 2 con cualquiera de los adyacentes a él usando D2[] (8.10.). O bien, también podemos considerar el grafo inducido por todos los vértices, excepto los de grado 2, 1 o 0 y posteriormente añadir los lados como en la función 8.12. con D3[]. La eliminación de vértices también puede hacerse buscando la submatriz correspondiente de la matriz de adyacencia, existen funciones de Mathematica específicas para ello, por ejemplo Take[] (véase ayuda de Mathematica para más detalle). En el ejercicio 8.44. se propone la resolución de este problema eligiendo estos caminos alternativos.

- b) Una vez eliminadas todas las subdivisiones, pretendemos comprobar si algún subgrafo es homeomorfo a K_5 o $K_{3,3}$. Para ello, en primer lugar construimos una función que determine si un grafo de 6 vértices contiene a $K_{3,3}$.

| FUNCIÓN | COMENTARIOS |
|---|--|
| <<"Combinatorica`"; | Introducimos el paquete de Matemática Discreta que incluye la función KSubsets[]. |
| K33[W_,matrizadyacencia_]:=Module[{subconjuntos,nok33,i,k1,k2}, | Como entrada tendrá un subconjunto de 6 vértices (identificados por sus subíndices). En el subgrafo inducido queremos comprobar si $K_{3,3}$ está contenido. |
| contiene=False; subconjuntos=KSubsets[Complement[W,{W[[1]]}],2]; | Generamos los subconjuntos de 3 vértices que puedan dar lugar al grafo bipartito completo $K_{3,3}$. |
| Do[nok33=False; A=Join[subconjuntos[[i]},{W[[1]]}]; B=Complement[W,A]; Do[If[Sum[matrizadyacencia[[A[[k1]],B[[k2]]]], {k2,3}]<3, nok33=True;Break[]]; ,{k1,3}]; If[!nok33,contiene=True;Break[]]; ,{i,Length[subconjuntos]}]; contiene | Comprobamos si los tres vértices de cada subconjunto son adyacentes a otros tres vértices de "W". |
|] | Salida de resultados. |

Función 8.13. Grafos planos: test para $K_{3,3}$.

- c) Con el siguiente programa determinaremos si un grafo es homeomorfo a otro que contiene a K_5 u homeomorfo a un grafo que contiene a $K_{3,3}$ como subgrafo.

| PROGRAMA | COMENTARIOS |
|---|---|
| <code><<“Combinatorica”;</code> | Introducimos el paquete de funciones que incluye la función KSubsets[]. |
| <code>D3[matrizadyacencia ,i ,j]:=...</code> | Definimos la función 8.11. |
| <code>QUITARVERTICESGRADO210[matrizadyacencia]:=...</code> | Definimos la función 8.16. |
| <code>K33[W ,matrizadyacencia]:=...</code> | Definimos la función 8.17. |
| <code>HOMEOMORFOK5K33[matrizadyacencia_]:=Module[{Subconjuntos,i,j,cont1,matrizadyacenciaK5,Nueva},</code> | La función tendrá como único argumento la matriz de adyacencia del grafo. |
| <code>Nueva=QUITARVERTICESGRADO210[matrizadyacencia];</code> <code>homeo=True;</code> | Eliminamos los vértices de grado 1 y 0, quitamos las subdivisiones. |
| <code>If[Dimensions[Nueva][[1]]==6,</code> <code>If[K33[{1,2,3,4,5,6},Nueva],homeo=False];];</code> | Comprobamos, en el caso de tener 6 vértices, si contiene a $K_{3,3}$. |
| <code>matrizadyacenciaK5=Table[If[i==j,0,1],{i,5},{j,5}];</code> <code>If[Dimensions[Nueva][[1]]==5,</code> <code>If[Nueva==matrizadyacenciaK5,homeo=False];];</code> | Comprobamos si es K_5 . |
| <code>homeo</code>] | Salida de resultados |

Función 8.14. Grafos homeomorfos a K_5 o $K_{3,3}$.

Podemos evitar el uso de la función K33[] (8.17.) y en su lugar utilizar cualquiera de las funciones que comprueban si dos matrices de adyacencia pertenecen a la misma clase de isomorfía, esto es, son grafos isomorfos, para ello podríamos usar la función 5.11. (ISOMORFOS2[]) y aplicarla a la matriz “Nueva” y a la matriz de adyacencia de $K_{3,3}$, que podríamos calcular con la función 6.13. (BCadyacencia[]). Sin embargo, la eficacia sigue siendo la misma, por lo que relegamos el estudio de esta alternativa al ejercicio 8.38.

- d) Terminaremos comprobando si existe algún subgrafo homeomorfo a K_5 o $K_{3,3}$, bastará con comprobar los subgrafos maximales, ya que estos incluyen el resto de casos. Completamos la comprobación con la función:

| PROGRAMA | COMENTARIOS |
|--|---|
| <code><<“Combinatorica”;</code> | Introducimos el paquete de funciones que incluye la función KSubsets[]. |
| <code>D3[matrizadyacencia ,i ,j]:=...</code> | Definimos la función 8.11. |
| <code>QUITARVERTICESGRADO210[matrizadyacencia]:=...</code> | Definimos la función 8.12. |
| <code>K33[W ,matrizadyacencia]:=...</code> | Definimos la function 8.13. |
| <code>HOMEOMORFOK5K33[matrizadyacencia_]:=</code> <code>MATRIZADYACENCIA[W ,F]:=...</code> | Definimos la función 8.14. |
| <code>PLANO[matrizadyacencia_]:=Module[{Nueva,k2,k3,subclados,lados},</code> | Definimos la función 5.1. |
| <code>Nueva=QUITARVERTICESGRADO210 </code> | La función tendrá como único argumento la matriz de adyacencia del grafo. |
| | Quitamos las subdivisiones y los vértices |

| | |
|---|--|
| matrizadyacencia]; | aislados o colgantes. |
| tiempo=TimeUsed[]; plano=True; | Almacenamos el tiempo usado hasta el momento y suponemos de partida que el grafo es plano. |
| vertices=Table[i,{i,Dimensions[Nueva][[1]]}]; | Determinamos el conjunto de vértices del grafo. |
| lados={}; Do[Do[If[Nueva[[t1,t2]]==1,AppendTo[lados,t1→t2]]; , {t1,t2}; , {t2,Dimensions[Nueva][[1]]};] | Calculamos todos los lados del grafo y los almacenamos en la lista “lados”. |
| Do[| Bucle que recorrerá el número de lados que podrán tener los subgrafos a comprobar, como mínimo 9 y como máximo “Length[lados]”. |
| subclados=KSubsets[lados,k2]; | Calculamos todos los subconjuntos de “k2” elementos de “lados”. |
| Do[| Bucle que recorrerá todos los subconjuntos de lados calculados. |
| If[HOMEOMORFO5K33 MATRIZADYACENCIA[vertices, subclados[[k3]]]==False, plano=False; subgrafo=subclados[[k3]]; Break[]] | Comprobamos si los subgrafos con conjunto de vértices “vértices” y por tanto maximales y conjunto de lados cada subconjunto considerado de más de 9 elementos es homeomorfo a un grafo que contiene a K_5 o homeomorfo a un grafo que contiene a $K_{3,3}$, en el caso de encontrar un subgrafo almacenamos en “subgrafo” el conjunto de lados. |
| If[TimeUsed[]-tiempo>5, Print[k2-9," ",Length[subclados]-k3]; tiempo=TimeUsed[];] | Cada 5 segundos mostramos información sobre los cálculos restantes. |
| ,{k3,Length[subclados]}; If[!plano,Break[]]; ,{k2,Length[lados],9,-1}; | |
| plano] | Salida de resultados |

Función 8.15. Grafos planos.

La versión 6 y siguientes de Mathematica incorpora una nueva función en el paquete `<<Combinatorica``:

PlanarQ[G]

que comprueba si el grafo asociado al objeto de tipo grafo “G” es plano.

Exploraremos el funcionamiento de estos programas con algunos ejemplos:

Ejemplo 8.17. Determinar si $K_{2,10}$ es plano.

Introducimos su matriz de adyacencia, podemos usar `BCAdyacencia[]` (6.13) o directamente:

In[]:= **matrizadyacencia={{0,0,1,1,1,1,1,1,1,1},**

```
{0,0,1,1,1,1,1,1,1,1,1},{1,1,0,0,0,0,0,0,0,0,0},  

{1,1,0,0,0,0,0,0,0,0,0},{1,1,0,0,0,0,0,0,0,0,0},  

{1,1,0,0,0,0,0,0,0,0,0},{1,1,0,0,0,0,0,0,0,0,0},  

{1,1,0,0,0,0,0,0,0,0,0},{1,1,0,0,0,0,0,0,0,0,0},  

{1,1,0,0,0,0,0,0,0,0,0},{1,1,0,0,0,0,0,0,0,0,0},  

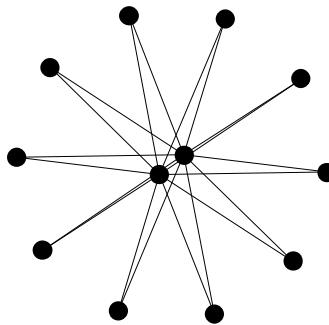
{1,1,0,0,0,0,0,0,0,0,0};
```

Lo representamos gráficamente:

```
In]:= GraphPlot[matrizadyacencia,  

PlotStyle->\{Black,PointSize[.05]\}]  

Out]=
```



Introducimos las funciones previas necesarias (8.11., 8.12., 8.13., 8.14. y 5.1.) y con la función 8.15.:

```
In]:= <<Combinatorica`;  

In]:= D3[matrizadyacencia_,i_,j_]:=Table[  

          : , : ]  

In]:= QUITARVERTICESGRADO210[matrizadyacencia_]:=   

          : , : ]  

In]:= K33[W_,matrizadyacencia_]:=Module[  

          : , : ]  

In]:= HOMEOGORFOK5K33[matrizadyacencia_]:=Module[  

          {Subconjuntos,i,j,cont1,matrizadyacenciaK5,Nueva},  

          : , : ]  

In]:= MATRIZADYACENCIA[W_,F_]:=Module[{i,j,k},
```

```

          :
          :

In]:=      PLANO[matrizadyacencia_]:=Module[
          {Nueva,k2,k3,subclados, lados },

          :
          :

In]:=      PLANO[matrizadyacencia]

Out]=      True

Y con PlanarQ[]:

In]:=      PlanarQ[FromAdjacencyMatrix[matrizadyacencia]]

Out]=      True

```

□

Ejemplo 8.18. Determinar si el grafo dado por la matriz de adyacencia siguiente es plano:

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

Introducimos la matriz de adyacencia en Mathematica y representamos el grafo:

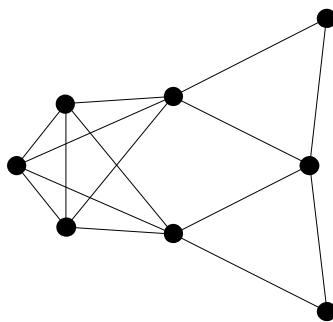
```

In]:=      matrizadyacencia={\{0,1,1,1,0,0,0,1},\{1,0,1,1,0,0,0,1},
          \{1,1,0,1,0,0,0,1},\{1,1,1,0,1,1,0,0},\{0,0,0,1,0,1,0,0},
          \{0,0,0,1,1,0,1,1},\{0,0,0,0,0,1,0,1},\{1,1,1,0,0,1,1,0}\};

In]:=      GraphPlot[matrizadyacencia,
          PlotStyle->\{Black,PointSize[.05]\}]

Out]=

```



Introducimos todas las funciones previas y con 8.15. comprobamos si es plano:

En efecto, obsérvese que la matriz de adyacencia del grafo, después de eliminar subdivisiones, es la de K_5 . La función 8.15. almacena en la variable “subgrafo” el conjunto de lados del subgrafo del grafo con matriz de adyacencia “Nueva”, el que se ha encontrado homeomorfo a otro que contiene a K_5 o a $K_{3,3}$. Podemos mostrar el valor de “subgrafo”:

```
In]:= subgrafo
Out]= {1→2, 1→3, 2→3, 1→4, 2→4, 3→4, 1→5, 2→5, 3→5, 4→5} □
```

Ejemplo 8.19. Representar gráficamente todos los grafos planos de 6 vértices distintos salvo isomorfismo que contienen al hexágono.

Usando las funciones 5.19., 5.20. (consideraremos las diagonales principales de las potencias segunda, tercera y cuarta para comprobar si dos matrices de adyacencia pertenecen a la misma clase de isomorfía) y el programa 5.18. determinamos todos los grafos de 6 vértices que contienen al hexágono (esto es, de Hamilton).

```
In]:= Añadirlado2[matrizadyacencia]:=Module[{i,j,matriz},
 $\vdots$   $\vdots$ 
In]:= QUITARISOMORFOS2[listamatrices]:=Module[
 $\vdots$   $\vdots$ 
In]:= n=6;
nlados=9;
matrizadyacencia={{0,1,0,0,0,1},{1,0,1,0,0,0},{0,1,0,1,0,0},
 $\quad\quad\quad\{0,0,1,0,1,0\},{0,0,0,1,0,1},{1,0,0,0,1,0}};$ 
 $\vdots$   $\vdots$ 
Out]=
 $\vdots$   $\vdots$ 
Total: 48 Tiempo empleado: 0.031
```

Ahora comprobamos cuáles de ellos son planos, para ello necesitaremos las funciones:

```
In]:= <<Combinatorica`;
In]:= D3[matrizadyacencia_,i_,j_]:=Table[
 $\vdots$   $\vdots$ 
In]:= QUITARVERTICESGRADO210[matrizadyacencia]:=
```

```

          :
          :

In]:=      K33[W_,matrizadyacencia_]:=Module[
          :
          :

In]:=      HOMEOMORFOK5K33[matrizadyacencia_]:=Module[
          {Subconjuntos,i,j,cont1,matrizadyacenciaK5,Nueva},

          :
          :

In]:=      MATRIZADYACENCIA[W_,F_]:=Module[{i,j,k},

          :
          :

In]:=      IND2[W1_,matrizadyacencia_]:=Module[{n,i,j},

          :
          :

In]:=      PLANO[matrizadyacencia_]:=Module[
          {Nueva,k2,k3,subclados,lados},

          :
          :

```

Comprobamos cuáles de ellos son planos y los almacenamos en “listaplanos”:

```

In]:=      listaplanos={};
Do[
  If[PLANO[listamatrices1[[i]]]
   ,AppendTo[listaplanos,listamatrices1[[i]]]];
,{i,Length[listamatrices1]}];
Length[listaplanos]
```

```

Out]=
          :
          :
```

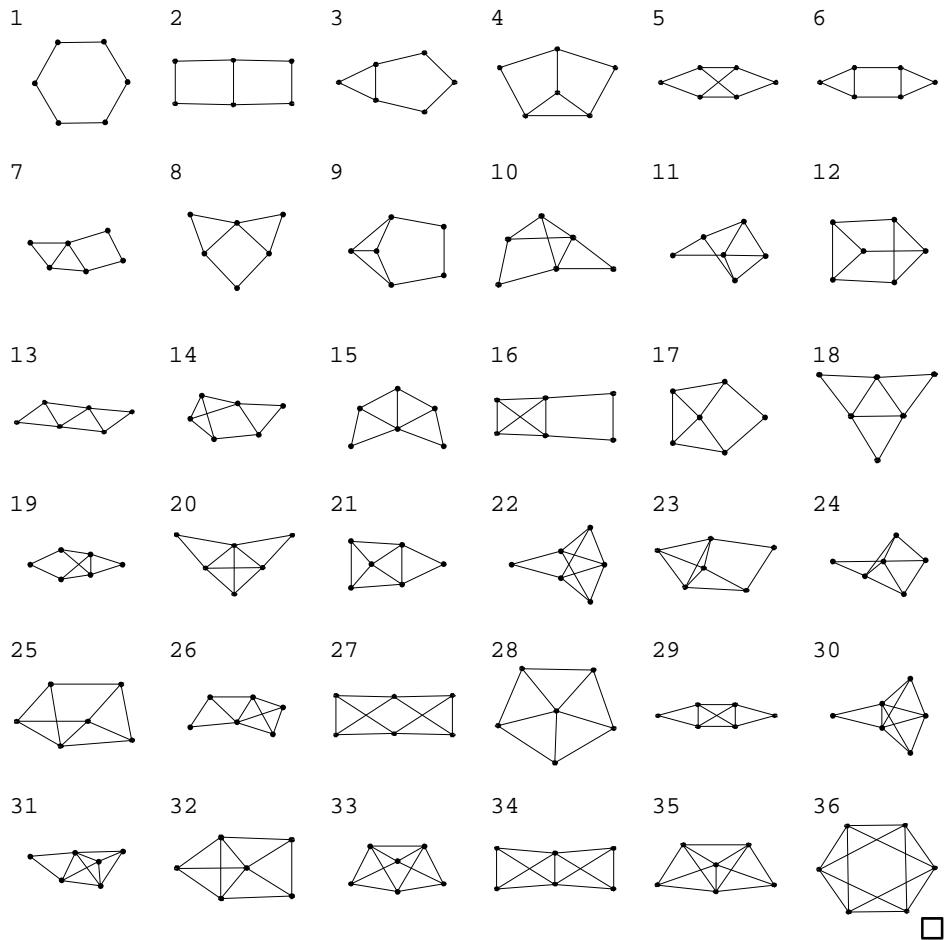
Out]:= 36

Por último representamos los 36 grafos de “listaplanos”:

```

In]:=      Do[
  Print[i];
  Print[GraphPlot[listaplanos[[i]],
   PlotStyle->{Black,PointSize[.05]}]]
,{i,Length[listaplanos]}]
```

Out]=



2.1. EFICACIA

Para determinar si un grafo es plano hemos usado un método combinatorio (igual que hicimos para determinar la coloración óptima de un grafo), comprobamos todos los subgrafos del grafo, aunque sea suficiente con probarlo sólo para los subgrafos maximales, tenemos que considerar todos los subconjuntos de lados del grafo de 9 o más lados. Comprobemos con el siguiente ejemplo lo que ocurre si consideramos un grafo con un número de lados relativamente elevado:

Ejemplo 8.20. Intentaremos comprobar si K_{10} es plano.

```
In[]:= matrizadyacencia=Table[If[i==j,0,1],{i,10},{j,10}];
```

Definimos todas las funciones necesarias y usamos 8.19.:

```
In[]:= <<Combinatorica`;
```

Si usamos la versión 5.2, el ordenador nos informa de que no hay suficiente memoria para realizar el cálculo solicitado, la versión 6 también, aunque con suficiente memoria, sí lo realiza, el cálculo se dilata enormemente.

□

No sería difícil evitar el uso de `KSubsets[]` que provoca el error del ejemplo 8.20. (si bien, en la versión 6, parece estar optimizada y utiliza menos memoria), pero aunque lo evitáramos seguiríamos enfrentándonos a un número muy elevado de subgrafos que tendríamos que comprobar, luego el método combinatorio que hemos desarrollado en la sección anterior, sólo es eficaz si el número de lados del grafo es pequeño.

En este epígrafe proponemos una solución basada en condiciones necesarias que han de cumplir los grafos planos, en particular, programamos una rutina que comprobará si el grafo o un homeomorfo a él contiene como subgrafo a K_5 o $K_{3,3}$.

| PROGRAMA | COMENTARIOS |
|---|--|
| <<Combinatorica`; | Introducimos el paquete de funciones que incluye la función KSubsets[]. |
| D3[matrizadyacencia_,i_,j_]:=... | Definimos la función 8.11. |
| QUITARVERTICESGRADO210[matrizadyacencia_]:=... | Definimos la función 8.12. |
| K33[W_,matrizadyacencia_]:=... | Definimos la función 8.13. |
| IND2[W1_,matrizadyacencia_]:=... | Definimos la función 6.10. |
| CDNECESARIAPLANO[matrizadyacencia_]:=Module[{Subconjuntos,i_,cont1,matrizadyacenciaK5}, Nueva=QUITARVERTICESGRADO210[matrizadyacencia]; Print["Comprobamos si puede ser plano"]; Print["Comprobamos si $K_{3,3}$ está contenido"]; plano=True; Subconjuntos=KSubsets[Table[i,{i,Dimensions[Nueva]][[1]]}],6]; Do[If[K33[Subconjuntos[[cont1]],Nueva],plano=False;Break[]]; Print["Está contenido"];,{cont1,Length[Subconjuntos]}]; If[plano, Print["No está contenido"]; Print["Comprobamos si K_5 está contenido"]; matrizadyacenciaK5=Table[If[i==j,0,1],{i,5},{j,5}]; Subconjuntos=KSubsets[Table[i,{i,Dimensions[Nueva]][[1]]}],5]; Do[If[IND2[Subconjuntos[[cont1]],Nueva]==matrizadyacenciaK5,plano=False;Break[]];Print["Está contenido"];,{cont1,Length[Subconjuntos]}];]; If[plano,Print["No está contenido"]]; plano] | La función tendrá como único argumento la matriz de adyacencia del grafo. Eliminamos los vértices de grado 1 y 0, quitamos las subdivisiones. Comprobamos si contiene a $K_{3,3}$. Comprobamos si contiene a K_5 . |
| | Salida de resultados |

Función 8.16. Condición necesaria para grafos planos.

Ejemplo 8.21. Intentaremos comprobar de nuevo si K_{10} es plano.

In]:= matrizadyacencia=Table[If[i==j,0,1],{i,10},{j,10}];

Definimos todas las funciones necesarias y usamos 8.20.:

In]:= <<Combinatorica`;

In]:= D3[matrizadyacencia_,i_,j_]:=Table[

: :

In[]:= **QUITARVERTICESGRADO210[matrizadyacencia]:=**

⋮ ⋮

In[]:= **K33[W_,matrizadyacencia]:=Module[**

⋮ ⋮

In[]:= **IND2[W1_,matrizadyacencia]:=Module[{n,i,j},**

⋮ ⋮

In[]:= **CDNECESARIAPLANO[matrizadyacencia]:=Module[**
{Subconjuntos,i,,cont1,matrizadyacenciaK5},

⋮ ⋮

In[]:= **CDNECESARIAPLANO[matrizadyacencia]**

Out[]= Comprobamos si puede ser plano
 Comprobamos si $K_{3,3}$ está contenido
 Está contenido
 False

Por tanto no es plano. □

Desafortunadamente, es una condición necesaria y no suficiente, por lo que existirán grafos que la verifiquen y sin embargo no sean planos:

Ejemplo 8.22. Consideramos el siguiente grafo:

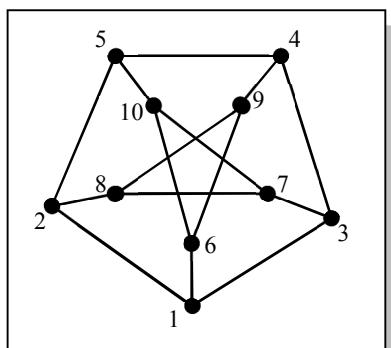


Ilustración 8.5. Grafo de Petersen³⁶

³⁶ El grafo de Petersen y otros grafo especiales están incorporados a la versión 6 de Mathematica en el paquete <<Combinatorica>>, podemos referirnos a él directamente con: **PetersenGraph**. Para conocer más detalles sobre otros grafos especiales incluidos podemos dirigirnos a la guía de referencia del programa Mathematica o escribir:

Calculamos su matriz de adyacencia y la introducimos:

```
In]:= matrizadyacencia={\{0,1,1,0,0,1,0,0,0,0,\},\{1,0,0,0,1,0,0,1,0,0,\},
\{1,0,0,1,0,0,1,0,0,0,\},\{0,0,1,0,1,0,0,0,1,0,\},\{0,1,0,1,0,0,0,0,0,1\},
\{1,0,0,0,0,0,0,1,1,\},\{0,0,1,0,0,0,0,1,0,1,\},\{0,1,0,0,0,0,1,0,1,0\},
\{0,0,0,1,0,1,0,1,0,0,\},\{0,0,0,0,1,1,1,0,0,0\}};
```

En primer lugar comprobamos si puede ser plano con 8.20., introducimos todas las funciones necesarias:

```
In]:= <<Combinatorica`;
In]:= D3[matrizadyacencia_i,j_]:=Table[
      \vdots \vdots
];
In]:= QUITARVERTICESGRADO210[matrizadyacencia_]:=[
      \vdots \vdots
];
In]:= K33[W_,matrizadyacencia_]:=Module[
      \vdots \vdots
];
In]:= IND2[W1_,matrizadyacencia_]:=Module[\{n,i,j\},
      \vdots \vdots
];
In]:= CDNECESARIAPLANO[matrizadyacencia_]:=Module[
      \{Subconjuntos,i,,cont1,matrizadyacenciaK5\},
      \vdots \vdots
];
In]:= CDNECESARIAPLANO[matrizadyacencia]
```

```
Out]= True
```

Según 8.20. podría serlo, sin embargo con la función 8.19.:

```
In]:= HOMEOMORFOK5K33[matrizadyacencia_]:=Module[
      \{Subconjuntos,i,j,cont1,matrizadyacenciaK5,Nueva\},
```

```
In]:= ShowGraphArray[FiniteGraphs]
```

O bien referirnos a él por su “nombre” y escribir:

```
GraphData[nombre]
```

```
In[7]:= MATRIZADYACENCIA[W_,F_]:=Module[{i,j,k},
```

•
•
•

In]:= **PLANO[matrizadyacencia_]:=Module[**
{Nueva,k2,k3,subclados,lados},

• • • • •

In[]:= **PLANO[matrizadyacencia]**

Out[*] =* False

En efecto, el subgrafo encontrado, si quitamos el vértice 10, es homeomorfo a $K_{3,3}$:

In[]:= **subgrafo**

```
Out[7]= {1→2, 1→3, 3→4, 2→5, 4→5, 1→6, 3→7, 2→8, 7→8, 4→9, 6→9, 8→9, 5→10}
```

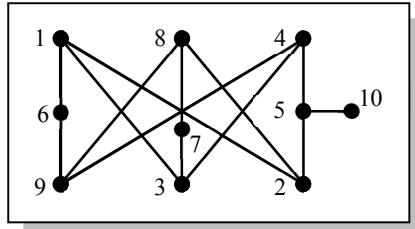


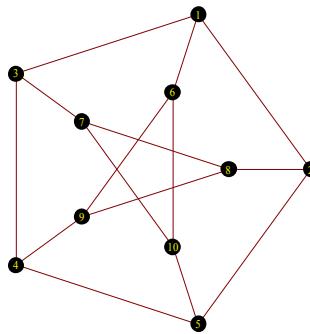
Ilustración 8.6.

Lo podemos evidenciar usando Mathematica, aprovecharemos el código que empleamos en el ejemplo 7.21., representamos el grafo de Petersen como en la ilustración 8.5.:

```
In[7]:= coordenadas=Table[{0,0},{i,10}];  
coordenadas[[1]]={Cos[1/5^2*Pi]^2,Sin[1/5^2*Pi]^2};  
coordenadas[[3]]={Cos[2/5^2*Pi]^2,Sin[2/5^2*Pi]^2};  
coordenadas[[4]]={Cos[3/5^2*Pi]^2,Sin[3/5^2*Pi]^2};  
coordenadas[[5]]={Cos[4/5^2*Pi]^2,Sin[4/5^2*Pi]^2};  
coordenadas[[2]]={Cos[2*Pi]^2,Sin[2*Pi]^2};  
coordenadas[[6]]={Cos[1/5^2*Pi],Sin[1/5^2*Pi]};  
coordenadas[[7]]={Cos[2/5^2*Pi],Sin[2/5^2*Pi]};  
coordenadas[[9]]={Cos[3/5^2*Pi],Sin[3/5^2*Pi]};  
coordenadas[[10]]={Cos[4/5^2*Pi],Sin[4/5^2*Pi]};  
coordenadas[[8]]={Cos[2*Pi],Sin[2*Pi]};  
GraphPlot[matrizadyacencia,
```

```
VertexRenderingFunction->({Black,Disk[#,1],
Yellow,Text[#,#1]}&),
VertexCoordinateRules->coordenadas|
```

Out[]=



Ahora buscamos el subgrafo encontrado y almacenado en “subgrafo”, destacamos los lados que lo componen:

```
In[ ]:= A=matrizadyacencia;
W=Table[i,{i,Dimensions[A][[1]]}];
F={};
Do[Do[
  If[A[[i,j]]==1,AppendTo[F,i->j]];
,{i,j}];{j,Dimensions[A][[1]]}];
subg=Table[If[Intersection[{subgrafo[[i]],F}=={},  

  subgrafo[[i]][[2]]->subgrafo[[i]][[1]],
  subgrafo[[i]][[1]]->subgrafo[[i]][[2]]];
,{i,Length[subgrafo]}];
Do[
  If[Intersection[subg,{F[[CONTADORi]]}]=={},  

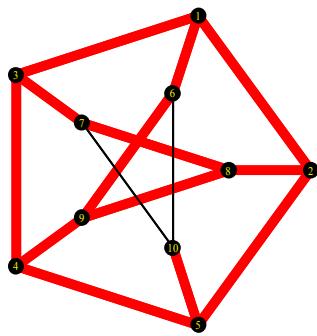
  colores[{F[[CONTADORi]][[1]],
  F[[CONTADORi]][[2]]}]=Black;
  colores[{F[[CONTADORi]][[1]],
  F[[CONTADORi]][[2]]}]=Black;
  grosor[{F[[CONTADORi]][[1]]},
  F[[CONTADORi]][[2]]]=0.005;
  grosor[{F[[CONTADORi]][[2]],
  F[[CONTADORi]][[1]]}]=0.005;
  colores[{F[[CONTADORi]][[1]],
  F[[CONTADORi]][[2]]}]=Red;
  colores[{F[[CONTADORi]][[2]],
  F[[CONTADORi]][[1]]}]=Red;
  grosor[{F[[CONTADORi]][[1]]},
  F[[CONTADORi]][[2]]]=0.025;
```

```

grosor[{F[[CONTADORi]][[2]],
  F[[CONTADORi]][[1]]}=0.025;
];
,{CONTADORi,Length[F]}];
GraphPlot[A,VertexRenderingFunction->
  ({Black,Disk[#,1],Yellow,Text[#2,#1]}&),
  VertexCoordinateRules->coordenadas,
  EdgeRenderingFunction->({colores[{#2[[1]],#2[[2]]}],
    Thickness[grosor[{#2[[1]],#2[[2]]}]],Line[#1]}&)]

```

Out[]=



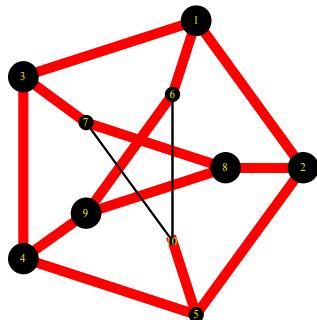
Destacamos los vértices que no son colgantes o subdivisiones del subgrafo:

```

In[ ]:= disco={0.2,0.2,0.2,0.2,0.1,0.1,0.1,0.2,0.2,0.05};
GraphPlot[A,VertexRenderingFunction->({Black,
  Disk[#,disco[[#2]]],Yellow,Text[#2,#1]}&),
  VertexCoordinateRules->coordenadas,
  EdgeRenderingFunction->({colores[{#2[[1]],#2[[2]]}],
    Thickness[grosor[{#2[[1]],#2[[2]]}]],Line[#1]}&)]

```

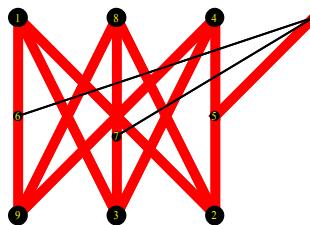
Out[]=



Por último, cambiamos únicamente las coordenadas de los vértices, y volvemos a representar:

```
In[]:=      coordenadas[[9]]={1,0};
            coordenadas[[3]]={2,0};
            coordenadas[[2]]={3,0};
            coordenadas[[1]]={1,2};
            coordenadas[[8]]={2,2};
            coordenadas[[4]]={3,2};
            coordenadas[[6]]={1,1};
            coordenadas[[7]]={2,8};
            coordenadas[[5]]={3,1};
            coordenadas[[10]]={4,2};
GraphPlot[A,VertexRenderingFunction->({Black,
Disk[#],disco[[#2]],Yellow,Text[#2,#1]}&),
VertexCoordinateRules->coordenadas,
EdgeRenderingFunction->({colores[{{#2[[1]],#2[[2]]}}],
Thickness[grosor[{{#2[[1]],#2[[2]]}}]],Line[#1]}&)]
```

Out[] =



Obsérvese, que ahora es evidente el “subgrafo”, salvo subdivisiones, que contiene a $K_{3,3}$, ya que la matriz de adyacencia del grafo de Petersen no la hemos tocado, tan sólo hemos alterado las coordenadas de los vértices y hemos destacado unos vértices y lados sobre otros.

□

El siguiente ejemplo nos muestra la complejidad que envuelve al problema, nos cogemos un grafo plano y que en consecuencia verificará 8.20., para asegurar que es plano tendremos que utilizar forzosamente 8.19. y aunque no es un grafo muy complejo, observaremos que el tiempo de proceso necesario es muy elevado.

Ejemplo 8.23. Comprobar que el grafo siguiente es plano.

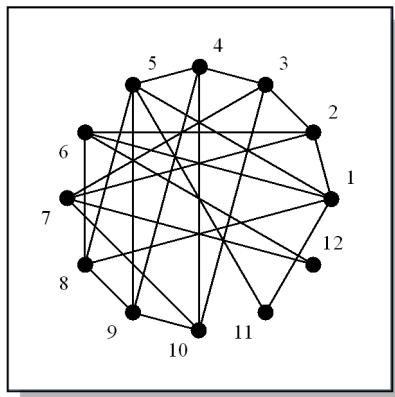


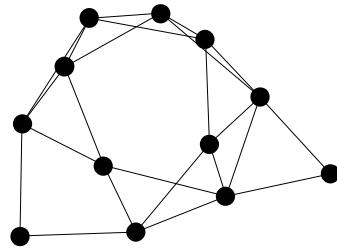
Ilustración 8.7.

Introducimos la matriz de adyacencia y lo representamos con Mathematica:

```
In]:= matrizadyacencia={{0,1,0,0,1,1,0,1,0,0,1,0},
{1,0,1,0,0,1,1,0,0,0,0,0},{0,1,0,1,0,0,1,0,0,1,0,0},
{0,0,1,0,1,0,0,0,1,1,0,0},{1,0,0,1,0,0,0,1,1,0,1,0},
{1,1,0,0,0,0,1,0,0,0,1},{0,1,1,0,0,0,0,0,0,1,0,1},
{1,0,0,0,1,1,0,0,1,0,0,0},{0,0,0,1,1,0,0,1,0,1,0,0},
{0,0,1,1,0,0,1,0,1,0,0,0},{1,0,0,0,1,0,0,0,0,0,0,0,0},
{0,0,0,0,0,1,1,0,0,0,0,0}};
```

```
In]:= GraphPlot[matrizadyacencia,
PlotStyle->{Black,PointSize[.05]}]
```

Out]=



Tras eliminar subdivisiones el grafo que quedará será:

```
In]:= D3[matrizadyacencia_,i_,j_]:=Table[
```

⋮ ⋮

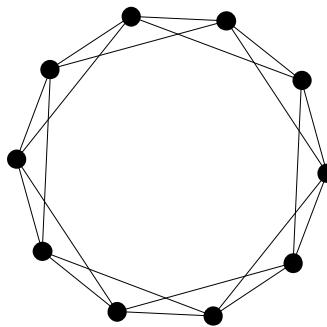
```
In]:= QUITARVERTICESGRADO210[matrizadyacencia_]:=
```

⋮ ⋮

In[]:= **Nueva=QUITARVERTICESGRADO210[matrizadyacencia];**

In[]:= **GraphPlot[Nueva,PlotStyle->{Black,PointSize[.05]}]**

Out[]=



Definimos todas las funciones necesarias y aplicamos la función 8.20.:

In[]:= **<<Combinatorica`;**

In[]:= **K33[W_,matrizadyacencia_]:=Module[**

⋮ ⋮

In[]:= **IND2[W1_,matrizadyacencia_]:=Module[{n,i,j},**

⋮ ⋮

In[]:= **CDNECESARIAPLANO[matrizadyacencia_]:=Module[**
{Subconjuntos,i,,cont1,matrizadyacenciaK5},

⋮ ⋮

In[]:= **CDNECESARIAPLANO[matrizadyacencia]**

Out[]= **True**

Según la función 8.20. el grafo podría ser plano, ahora lo comprobamos con 8.19.:

In[]:= **HOMEOMORFOK5K33[matrizadyacencia_]:=Module[**
{Subconjuntos,i,j,cont1,matrizadyacenciaK5,Nueva},

⋮ ⋮

In[]:= **MATRIZADYACENCIA[W_,F_]:=Module[{i,j,k},**

```

          :
          :

In]:=      PLANO[matrizadyacencia_]:=Module|
{Nueva,k2,k3,subclados,lados},

          :
          :

```

Comprobamos si el grafo es plano con la función 8.19. (el cálculo se prolongará por varias horas).

```

In]:=      PLANO[matrizadyacencia]

Out]=      :
          :
          :

Out]=      True

```

□

Como ocurre en la mayoría de problemas tratados en este libro, existen más propiedades, otras técnicas y atajos o trucos que permitirían mejorar el test y que no vamos a desarrollar. La nueva función PlanarQ[] incluida en la versión 6 es bastante eficaz y en particular lo es más que PLANO[], aunque con ella nos perdemos los pasos intermedios y algunos problemas interesantes como los de los ejemplos 8.18. o 8.22.

2.2. REPRESENTACIONES PLANAS. REGIONES DE UN GRAFO PLANO

Un grafo plano divide al plano en regiones o caras, todas ellas delimitadas por un ciclo que las rodea, excepto una de ellas que es ilimitada o infinita y que se corresponde con el resto del plano. Nos limitaremos al estudio de grafos conexos, quedando para el lector el resto de casos (ejercicio 8.40.). El número de caras de un grafo conexo viene determinado por el siguiente teorema (podemos encontrar una demostración en el epígrafe 6.7 de [31]).

Teorema 8.3. Fórmula de Euler para grafos. Sea G un (p, q) -grafo conexo³⁷, entonces si llamamos r al número de regiones, tendremos:

$$r = q - p + 2.$$

□

Podemos determinar el número de regiones fácilmente con el teorema anterior.

$$\text{Regiones}=\text{Length}[F]-\text{Length}[W]+2$$

³⁷ Si el grafo no es conexo, es fácil determinar otra fórmula a partir del teorema 8.3., teniendo en cuenta sus componentes conexas (ejercicio 8.39.).

Llamaremos grado de una región al número de lados del ciclo que rodea a dicha región (algunos lados podrán contarse dos veces).

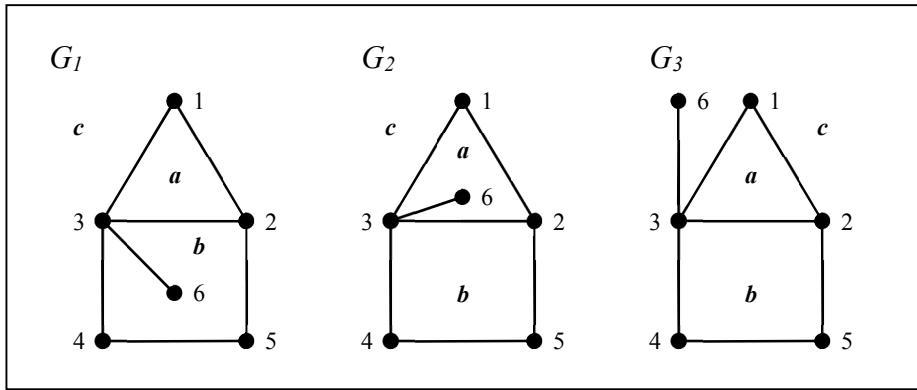


Ilustración 8.8.

Obsérvese que en la ilustración 8.8. tenemos tres grafos conexos, planos e isomorfos que dividen al plano en tres regiones a , b y c , dependiendo de la representación gráfica particular que hagamos del grafo, el grado de las regiones es distinto, nótense en consecuencia que este concepto resulta un poco ambiguo en situaciones como las de la ilustración 8.8. Incluso si evitamos situaciones como las de 8.8. (vértices aislados o colgantes que no aportan información respecto a la representación plana) encontraremos distintas formas de dibujar el mismo grafo que darán lugar a grados distintos de las regiones:

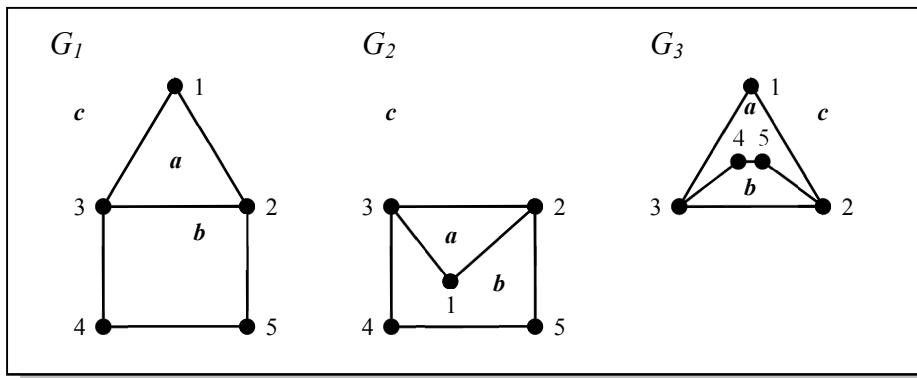


Ilustración 8.9.

Observemos la ilustración 8.9., la región a puede tener grado 3 o 5, la b tiene grado 4 o 5 y la c tiene grado 5, 4 o 3 según como dibujemos el grafo.

Comprobada la ambigüedad del concepto es claro que el grado las regiones de un mismo grafo determinado a partir de su definición o matriz de adyacencia no es único, éste dependerá del modo en que representemos gráficamente a dicho grafo. Por tanto para estudiar estos conceptos y usar Mathematica, tendremos que tener en cuenta la

representación gráfica plana particular del grafo que estemos considerando. Para ello, además del conjunto de vértices y lados o la matriz de adyacencia, tendremos que tener en cuenta el conjunto de todos los ciclos que delimitan las regiones de la representación particular que consideremos. El número de regiones r que existen es constante para cualquier representación plana que consideremos. En Mathematica introduciremos una lista formada por r ciclos, esto es, todos aquellos que bordean a las regiones (incluida la región ilimitada o infinita):

```
In]:= W={CONJUNTO DE VÉRTICES};  

F={CONJUNTO DE LADOS O FLECHAS};  

REGIONES={REGIÓN1, REGIÓN2,..., REGIÓNr};
```

Introduciremos los ciclos que bordean las regiones como una sucesión de vértices (de forma análoga podrían introducirse como sucesión de lados). Podremos determinar el grado de cada región fácilmente con:

```
GRADOREG[REGION_]:=Length[REGIONES[[REGION]]]-1;
```

Ejemplo 8.24. Consideraremos el grafo plano G_1 de la ilustración 7.1. y consideraremos una de las múltiples representaciones planas de este grafo conexo:

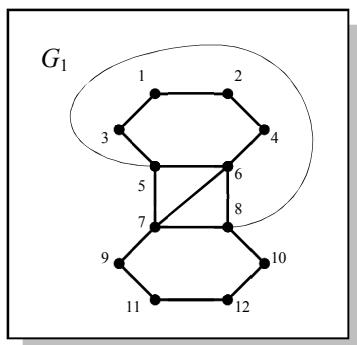


Ilustración 8.10.

Introduciremos en Mathematica esta representación plana de la siguiente forma:

```
In]:= REGIONES={\{1,2,4,6,5,3,1\},\{5,8,6,4,2,1,3,5\},\{5,6,7,5\},  

\{6,7,8,6\},\{7,8,10,12,11,9,7\},\{5,7,9,11,12,10,8,5\}\};
```

El grado de una región cualquiera es fácilmente calculable:

```
In]:= Length[REGIONES[[2]]]-1
```

```
Out]= 7
```

□

De esta forma podemos introducir en el ordenador una representación plana cualquiera de un grafo plano, obsérvese que se ha incluido la región ilimitada (vendría dada

por el ciclo que bordea a todo el grafo), en realidad ésta no es imprescindible³⁸ para determinar la representación gráfica del grafo porque puede deducirse de las demás (véase ejercicio 8.14.).

Ahora queremos colorear las regiones de un grafo conexo y plano, esto es, queremos asignar un color a cada región de forma que dos regiones adyacentes no estén coloreadas por el mismo color, entendiendo por regiones adyacentes, aquellas que tienen algún lado en común. Observemos con la siguiente ilustración que este problema puede reducirse al estudio de la coloración de los vértices de un grafo:

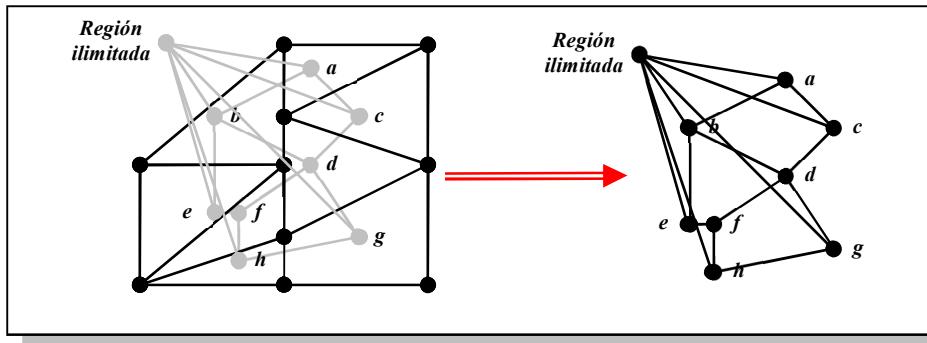


Ilustración 8.11.

Por tanto, para colorear las regiones de un grafo bastará con construir otro grafo cuyos vértices representen las regiones y estos serán adyacentes cuando dichas regiones lo sean. Programamos una función que tendrá por entrada los ciclos que bordean las regiones de la representación plana particular que queramos considerar, tendremos que añadir también (si queremos tenerla en cuenta) la región ilimitada. Lo hacemos en dos pasos, en primer lugar programamos una función que devuelva la sucesión de lados que componen cada ciclo, tendremos la precaución de ordenar los vértices que componen cada lado (recordemos que los lados $v \rightarrow w$ y $w \rightarrow v$ son iguales en un grafo no orientado, pero para Mathematica no están identificados y por tanto son distintos).

| FUNCIÓN | COMENTARIOS |
|---|--|
| SUCLADOS2[vertices_]:=Module[{i}, | La función tendrá por entrada una sucesión de vértices identificados por sus subíndices. |
| <pre> lados={}; Do[If[vertices[[i]]<vertices[[i+1]], AppendTo[lados,vertices[[i]]\[Rightarrow]vertices[[i+1]]] , AppendTo[lados,vertices[[i+1]]\[Rightarrow]vertices[[i]]]; ,{i,Length[vertices]-1}]; </pre> | Calculamos la sucesión de lados, ordenamos los vértices de cada lado. |
| lados | Salida de resultados. |

³⁸ Un caso particular de grafos planos son los árboles y bosques que se estudian en el siguiente epígrafe, en tal caso sólo existe una región: la ilimitada, se trata de un caso trivial desde este punto de vista, pero no entra en conflicto con el análisis propuesto, porque en tal caso sólo hay una única posible representación plana cuya única región sería la ilimitada.

| | |
|--|--|
| | |
|--|--|

Función 8.17. Coloración de regiones, paso 1.

En segundo lugar, calculamos con la función 8.18. las sucesiones de lados que componen cada uno de los ciclos que bordean las regiones y comprobamos cuáles de ellos tienen algún lado en común, representando en tal caso regiones adyacentes.

| FUNCIÓN | COMENTARIOS |
|--|--|
| SUCLADOS2[vertices]:=... | Definimos la función 8.17. |
| <pre>GRAFOREGIONES[REGIONES_]:=Module[{i,j,reginoeslados,ladosext,resto,regioneslados},</pre> <pre>matrizadyacencia=Table[0,{i,Length[REGIONES]} ,{j,Length[REGIONES]}]; regioneslados={}; Do[AppendTo[regioneslados, SUCLADOS2[REGIONES[[i]]]] ,{i,Length[REGIONES]}]; ladosext={}; Do[resto={}; Do[If[i≠j,resto=Union[resto,regioneslados[[j]]]; ,{j,Length[regioneslados]}]; ladosext=Union[ladosext, Complement[regioneslados[[i]],resto]]; ,{i,Length[regioneslados]}]; Do[Do[If[i≠j, If[Intersection[regioneslados[[i]], regioneslados[[j]]]≠{}, matrizadyacencia[[i,j]]=1; matrizadyacencia[[j,i]]=1;];]; ,{j,Length[REGIONES]}]; ,{i,Length[REGIONES]}]; matrizadyacencia];</pre> | <p>Como argumento introduciremos todos los ciclos que delimitan las regiones de la representación gráfica plana que consideremos (incluida, si queremos, la región ilimitada).</p> <p>Calculamos los lados que componen los ciclos que delimitan a cada región.</p> <p>Comprobamos que regiones son adyacentes y lo traducimos en un lado el grafo que estamos construyendo.</p> <p>Salida de resultado.</p> |

Función 8.18. Coloración de regiones: Grafo de las regiones.

Ejemplo 8.25. Calcular las coloraciones óptimas de las regiones de los grafos planos del ejemplo 8.24. y de la ilustración 8.9. con las representaciones planas que allí se indican, incluyendo la región ilimitada o infinita.

$$In//:= \quad \text{REGIONES}=\{\{5,7,9,11,12,10,8,5\},\{1,2,4,6,5,3,1\}, \\ \{5,8,6,4,2,1,3,5\},\{5,6,7,5\},\{6,7,8,6\},\{7,8,10,12,11,9,7\}\};$$

Utilizamos la función 8.18. para determinar el grafo asociado a las regiones de la representación plana que consideramos, previamente definimos también la función 8.17.:

```
In]:= SUCLADOS2[vertices_]:=Module[{i},
  :
  :
  In]:= GRAFOREGIONES[REGIONES_]:=Module[
    {i,j,reginoeslados,ladosext,resto,regioneslados},
    :
    :
    In]:= A=GRAFOREGIONES[REGIONES]
    Out]=={{{0,0,1,1,0,1},{0,0,1,1,0,0},{1,1,0,0,1,0},
      {1,1,0,0,1,0},{0,0,1,1,0,1},{1,0,0,0,1,0}}}
```

Ahora comprobamos si existen 2-coloraciones con 8.1.:

```
In]:= NCOLORACIONES[matrizadyacencia_,lambda_]:=Module[
  {coloracionestemp,t,h,i,j,colorady,k},
  :
  :
  In]:= NCOLORACIONES[A,2]
  Out]={{{1,1,2,2,1,2},{2,2,1,1,2,1}}}
```

Por tanto es 2-cromático.

Antes de calcular una coloración óptima de las regiones del grafo de la ilustración 8.11. asignamos un índice a cada vértice:

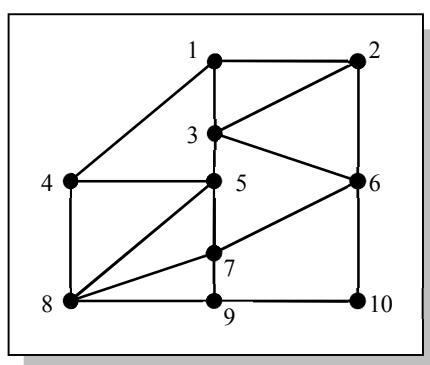


Ilustración 8.12.

Introducimos las regiones, incluida la ilimitada:

In[]:= **REGIONES2=**{ {1,2,6,10,9,8,4,1}, {1,3,5,4,1}, {1,2,3,1}, {2,3,6,2}, {3,5,7,6,3}, {4,5,8,4}, {5,7,8,5}, {7,8,9,7}, {6,7,9,10,6} };

Calculamos el grafo asociado y el número cromático.

In[]:= **A2=GRAFOREGIONES[R2]**

Out[]:= { {0,1,1,1,0,1,0,1,1}, {1,0,1,0,1,1,0,0,0}, {1,1,0,1,0,0,0,0,0}, {1,0,1,0,1,0,0,0,0}, {0,1,0,1,0,0,1,0,1}, {1,1,0,0,0,0,1,0,0}, {0,0,0,0,1,1,0,1,0}, {1,0,0,0,0,0,0,1,0,1}, {1,0,0,0,1,0,0,1,0} }

In[]:= **NCOLORACIONES[A2,3]**

Out[]:= { {1,2,3,2,1,3,2,3,2}, {1,2,3,2,3,3,1,3,2}, {1,2,3,2,3,3,2,3,2}, {1,3,2,3,1,2,3,2,3}, {1,3,2,3,2,2,1,2,3}, {1,3,2,3,2,2,3,2,3}, {2,1,3,1,2,3,1,3,1}, {2,1,3,1,3,3,1,3,1}, {2,1,3,1,3,3,2,3,1}, {2,3,1,3,1,1,2,1,3}, {2,3,1,3,1,1,3,1,3}, {2,3,1,3,2,1,3,1,3}, {3,1,2,1,2,2,1,2,1}, {3,1,2,1,2,2,3,2,1}, {3,1,2,1,3,2,1,2,1}, {3,2,1,2,1,1,2,1,2}, {3,2,1,2,1,1,3,1,2}, {3,2,1,2,3,1,2,1,2} }

En el grafo hay triángulos podemos asegurar que es 3-cromático y por tanto cualquiera de las coloraciones anteriores es óptima. □

3. ÁRBOLES Y BOSQUES

Un árbol es un grafo conexo y sin ciclos elementales (circuitos). Un bosque es un grafo sin ciclos elementales, esto es, es un grafo donde cada componente conexa es un árbol. Los árboles son fácilmente caracterizables por el siguiente resultado que podemos encontrar demostrado en la introducción del capítulo 8 del libro “Matemática Discreta 2^a Edición” (García Merayo, F., [19]):

Teorema 8.4. Sea G un (p, q) -grafo conexo, entonces:

$$G \text{ es un árbol} \Leftrightarrow p = q + 1.$$
□

Podemos determinar si un grafo es un árbol o un bosque de dos formas:

1. Directamente desde la definición comprobando la existencia de ciclos elementales y determinando si es conexo con la función CONEXO[] (7.13.) estudiada en el capítulo 7.

2. Utilizando el teorema 8.4. y aplicándolo a cada componente conexa del grafo, para ello utilizaremos la función COMPONENTESCONEXAS[] (7.14.) del capítulo 7.

3.1. ÁRBOLES Y BOSQUES. CICLOS ELEMENTALES

Con la siguiente función podemos determinar los ciclos elementales de un grafo:

| FUNCIÓN | COMENTARIOS |
|--|--|
| <pre>CICLOSELEMENTALES[vertice1_,longitud_,A_]:=Module[{i,ultvertice,caminos,adyacentes, caminostemp,k,j,t}, caminos={{vertice1}}; Do[caminostemp={}; Do[adyacentes={}; ultvertice=caminos[[k]][[i]]; Do[If[A[[ultvertice,j]]==1, AppendTo[adyacentes,j]; ,{j,Dimensions[A][[1]]}]; adyacentes=Complement[adyacentes, caminos[[k]]]; If[adyacentes!={}, Do[AppendTo[caminostemp, Append[caminos[[k]],adyacentes[[t]]]]; ,{t,Length[adyacentes]}];]; ,{k,Length[caminos]}]; caminos=caminostemp; ,{i,longitud-1}]; caminostemp={}; Do[If[A[[caminos[[k]][[longitud]],vertice1]]==1, AppendTo[caminostemp, Append[caminos[[k]],vertice1]]; ,{k,Length[caminos]}]; caminostemp];];];]</pre> | Los argumentos de la función son: un vértice (inicio del ciclo e identificado por su subíndice), la longitud que pretendemos que tenga el ciclo y la matriz de adyacencia. |
| | Determinamos todos los caminos elementales de longitud: (“longitud” – 1). |
| | Se comprueba si alguno de los caminos elementales obtenidos se puede completar hasta un ciclo terminando en “vertice1”. |
| | Salida de resultados. |

Función 8.19. Ciclos elementales de longitud l .

En la versión 6 y siguientes de Mathematica y dentro del paquete <<Combinatorica` disponemos de varias funciones para encontrar o manejar ciclos elementales y en particular para comprobar si un grafo es un árbol, todas ellas se aplican sobre objetos de tipo grafo:

| | |
|---------------------|--|
| TreeQ[G] | Comprueba si “G” es un árbol. |
| AcyclicQ[G] | Comprueba si “G” contiene ciclos elementales. |
| FindCycle[G] | Encuentra, si existe, un ciclo elemental en “G”. |

| | |
|-----------------------------|---|
| ExtractCycles[G] | Devuelve una lista maximal de ciclos disjuntos de “G”. |
| Cycle[n] | Construye un objeto de tipo grafo que consiste en el grafo 2-regular que constituye un polígono de “n” lados. |
| DeleteCycle[G,ciclo] | Elimina el ciclo “ciclo” del grafo “G”. |

Tabla 8.1.

Ejemplo 8.26. Calcular los ciclos que comienzan en el vértice 1 de longitud 4 de K_6 .

Definimos la función 8.19. y la aplicamos a K_6 :

```
In]:= CICLOSELEMENTALES[vertice1_,longitud_,A_]:=
          :
          :
In]:= CICLOSELEMENTALES[1,4, Table[If[i==j,0,1],{i,6},{j,6}]]

Out]:= {{1,2,3,4,1},{1,2,3,5,1},{1,2,3,6,1},
        {1,2,4,3,1},{1,2,4,5,1},{1,2,4,6,1},
        {1,2,5,3,1},{1,2,5,4,1},{1,2,5,6,1},
        {1,2,6,3,1},{1,2,6,4,1},{1,2,6,5,1},
        {1,3,2,4,1},{1,3,2,5,1},{1,3,2,6,1},
        {1,3,4,2,1},{1,3,4,5,1},{1,3,4,6,1},
        {1,3,5,2,1},{1,3,5,4,1},{1,3,5,6,1},
        {1,3,6,2,1},{1,3,6,4,1},{1,3,6,5,1},
        {1,4,2,3,1},{1,4,2,5,1},{1,4,2,6,1},
        {1,4,3,2,1},{1,4,3,5,1},{1,4,3,6,1},
        {1,4,5,2,1},{1,4,5,3,1},{1,4,5,6,1},
        {1,4,6,2,1},{1,4,6,3,1},{1,4,6,5,1},
        {1,5,2,3,1},{1,5,2,4,1},{1,5,2,6,1},
        {1,5,3,2,1},{1,5,3,4,1},{1,5,3,6,1},
        {1,5,4,2,1},{1,5,4,3,1},{1,5,4,6,1},
        {1,5,6,2,1},{1,5,6,3,1},{1,5,6,4,1},
        {1,6,2,3,1},{1,6,2,4,1},{1,6,2,5,1},
        {1,6,3,2,1},{1,6,3,4,1},{1,6,3,5,1},
        {1,6,4,2,1},{1,6,4,3,1},{1,6,4,5,1},
        {1,6,5,2,1},{1,6,5,3,1},{1,6,5,4,1}}
```

Obviamente no es un árbol, para responder esta última cuestión también podemos aplicar directamente las funciones de la tabla 8.1.:

```
In]:= <<Combinatorica`
```

```
In]:= FindCycle[CompleteGraph[6]]
```

```
Out]:= {3,1,2,3}
```

```
In]:= ExtractCycles[CompleteGraph[6]]
```

```
Out]:= {{5,3,4,5},{5,1,4,2,5},{3,1,2,3}}
```

In[]:= **AcyclicQ[CompleteGraph[6]]**

Out[]:= **False**

In[]:= **TreeQ[CompleteGraph[6]]**

Out[]:= **False**

El grafo es conexo y con el teorema 8.4. ($p = 6$ y $q = 15$) o con cualquiera de las funciones concluimos que no es un árbol.

□

De forma similar se puede generar una rutina que determine caminos elementales (ejercicio 8.35.).

Con la función CICLOSELEMENTALES[] (8.19.) y CONEXO[] (7.13.) podemos determinar si un grafo es un árbol o bosque, buscaremos ciclos elementales de longitud mayor o igual que 3 y menor o igual que el número de vértices, ya que estos son los únicos posibles. Por otra parte y para que el algoritmo sea más eficaz, sólo testeamos esta posibilidad en aquellos vértices en donde sabemos si existen o no ciclos (no necesariamente elementales) aprovechando el teorema del número de caminos.

| FUNCIÓN | COMENTARIOS |
|--|---|
| CICLOSELEMENTALES[vertice1_,longitud_,A_]:= ... | Definimos la función 8.19. |
| BOSQUE[matrizadyacencia_]:=Module[{ n, longitud, i, bosque=True; n=Dimensions[matrizadyacencia][[1]]; | Test para determinar si un grafo es un bosque desde la matriz de adyacencia. |
| Do[potencia= MatrixPower[matrizadyacencia, longitud]; Do[If[potencia[[i,i]]>0, If[CICLOSELEMENTALES[i, longitud, matrizadyacencia]≠{}, bosque=False;Break[];];]; ,{i,n}; If[bosque==False,Break[]]; ,{longitud,3,n}]; | Suponemos de partida que sí lo es. |
| bosque]; | Comprobamos la existencia de ciclos elementales de cualquier longitud. Salida de resultados. |

Función 8.20. Árboles y bosques.

Aunque para árboles o bosques el test no ganaría en eficacia, para grafos que no sean árboles o bosques, si integramos 8.19. y 8.20. en una única función y paramos nada más encontrar un ciclo, optimizaríamos el test (el ejercicio 8.36. propone esta mejora), si bien comprobaremos más adelante que para árboles y bosques podremos programar un test muy rápido basado en la simplicidad del teorema 8.4.

Ejemplo 8.27. Consideramos el grafo G cuyo conjunto de lados F viene generado por el siguiente código:

```
In[]:= n=4;
t=1;t1=1;F={};
Do[Do[
  Do[t++;AppendTo[F,t1→t],{k,i}];
  t1++;
,{j,(i-1)!}],{i,1,n}];F
Out[]={1→2,2→3,2→4,3→5,3→6,3→7,4→8,4→9,4→10,
5→11,5→12,5→13,5→14,6→15,6→16,6→17,6→18,
7→19,7→20,7→21,7→22,8→23,8→24,8→25,8→26,
9→27,9→28,9→29,9→30,10→31,10→32,10→33,
10→34}
```

Al menos debería tener 34 vértices:

```
In[]:= W=Table[i,{i,34}];
```

Comprobamos que es un árbol, en primer lugar usando la función 5.1. determinamos su matriz de adyacencia, comprobamos si es conexo con 7.13. y finalmente comprobamos si es un árbol con las funciones 8.19. y 8.20.

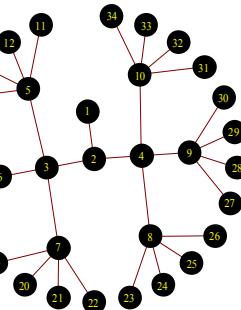
```
In[]:= MATRIZADYACENCIA[W_,F_]:=Module[{i,j,k},
];
In[]:= matriz=MATRIZADYACENCIA[Table[i,{i,34}],F];
In[]:= CONEXO[A_]:=Module[{i,j,B},
];
In[]:= CICLOSELEMENTALES[vertice1_,longitud_,A_]:=Module[{i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z},BOSQUE];
In[]:= BOSQUE[matrizadyacencia_]:=Module[{n, longitud, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z},CICLOSELEMENTALES];
In[]:= CONEXO[matriz] && BOSQUE[matriz]
Out[]= True
```

Ahora lo dibujamos:

In[7]:=

```
GraphPlot[F, VertexRenderingFunction->
  {Black, Disk[#, .2], Yellow, Text[#2, #]}&)]
```

Out[7]=

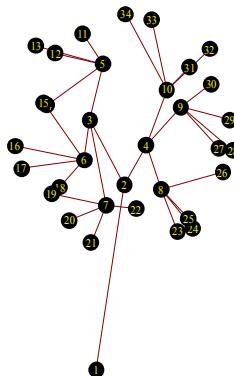


Para terminar vamos a cambiar las coordenadas de los vértices introduciendo un factor aleatorio, además el vértice primero lo vamos a representar más bajo de forma que represente la base del tronco de un árbol:

In[7]:=

```
GraphPlot[GRAFO, VertexRenderingFunction->
  {Black, Disk[#, .2], Yellow, Text[#2, #]}&,
  VertexCoordinateRules->coordenadas]
```

Out[7]=



□

Ejemplo 8.28. Representar gráficamente todos los árboles de 7 vértices distintos salvo isomorfismo.

Usando las funciones 5.19., 5.20. (consideraremos las diagonales principales de las potencias segunda, tercera y cuarta para comprobar si dos matrices de adyacencia pertenecen a la misma clase de isomorfía) y el programa 5.18 determinamos los grafos distintos salvo isomorfismo de 7 vértices y a lo sumo 8 lados.

In[]:= AñadirLado2[matrizadyacencia_]:=Module[{i,j,matriz},

⋮ ⋮

In[]:= QUITARISOMORFOS2[listamatrices_]:=Module[

⋮ ⋮

In[]:= n=7;
nLados=8;
matrizadyacencia=Table[0,{i,n},{j,n}];

⋮ ⋮

Out[]=

⋮ ⋮

Total: 243 Tiempo empleado: 0.86

Ahora comprobamos cuáles de ellos son árboles, para ello necesitaremos las funciones: 8.19., 8.20. y 7.13:

In[]:= CICLOSELEMENTALES[vertice1_,longitud_,A_]:=

⋮ ⋮

In[]:= BOSQUE[matrizadyacencia_]:=Module[{n,longitud,i},

⋮ ⋮

In[]:= CONEXO[A_]:=Module[{i,j,B},

⋮ ⋮

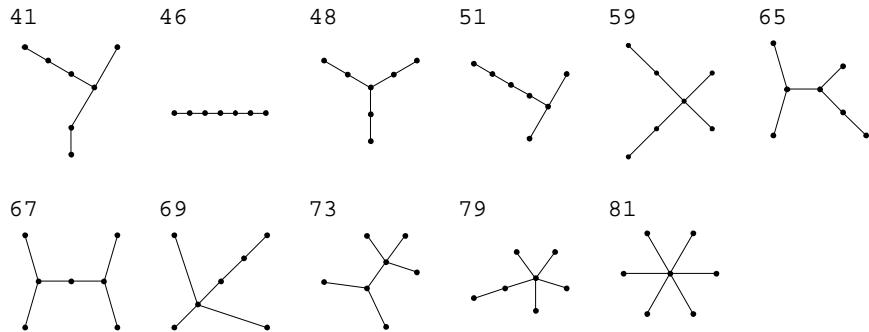
Comprobamos cuáles de ellos son árboles y los representamos:

In[]:= Do[
If[CONEXO[listamatrices1[[i]]],
If[BOSQUE[listamatrices1[[i]]],
Print[i];
Print[GraphPlot[listamatrices1[[i]],
PlotStyle->{Black,PointSize[.05]}]];

```

];];
,{i,Length[listamatrices1]}}
Out[]=

```



□

3.2. ÁRBOLES Y BOSQUES. NÚMERO DE LADOS

Observando el número de lados de un grafo conexo es fácil determinar si el grafo es un árbol por el teorema 8.4., podemos implementarlo directamente:

| FUNCIÓN | COMENTARIOS |
|--|---|
| CONEXO[A_]:=... | Definimos la función 7.13. |
| ARBOL[matrizadyacencia_]:=Module[{i,j,lados}, | Test para determinar si un grafo es un árbol desde la matriz de adyacencia. |
| arbol=False; | Suponemos que no lo es. |
| If[CONEXO[matrizadyacencia], | Comprobamos si es conexo. |
| arbol=(Dimensions[matrizadyacencia][[1]]== (Sum[matrizadyacencia[[i,j]], {i,Dimensions[matrizadyacencia][[1]]}, {j,Dimensions[matrizadyacencia][[2]]}]/2)+1); | Comprobamos la igualdad del teorema 8.4. |
| arbol] | Salida de resultados. |

Función 8.21. Árboles.

En el caso de que el grafo no sea conexo, estudiaremos si el grafo es un bosque, para ello tendremos que comprobar si cada componente conexa es un árbol.

| FUNCIÓN | COMENTARIOS |
|---|--|
| IND2[W1_,matrizadyacencia_]:=... | Definimos la función 6.10. |
| COMPONENTESCONEXAS[matrizadyacencia_]:=... | Definimos la función 7.14. |
| CONEXO[A_]:=... | Definimos la función 7.13. |
| ARBOL[matrizadyacencia_]:=... | Definimos la función 8.21. |
| BOSQUE2[matrizadyacencia_]:=Module[{componentes,i}, | Test para determinar si un grafo es un bosque desde la matriz de adyacencia. |
| bosque=True; | Partimos de que lo es. |

| | |
|--|--|
| componentes= COMPONENTESCONEXAS[matrizadyacencia]; | Calculamos las componentes conexas. |
| Do[If[ARBOL[IND2[componentes[[i]], matrizadyacencia]]==False, bosque=False]; ,{i,Length[componentes]}]; bosque] | Comprobamos que cada componente conexa sea un árbol. |
| | Salida de resultados. |
| | Función 8.22. Bosques. |

Ejemplo 8.29. Determinar usando la función 8.22. todos los bosques de 8 vértices.

Usando las funciones 5.19., 5.20. (consideraremos las diagonales principales de las potencias segunda, tercera y cuarta para comprobar si dos matrices de adyacencia pertenecen a la misma clase de isomorfía) y el programa 5.18 determinamos todos los grafos distintos salvo isomorfismo de 8 vértices y 7 lados o menos (no calculamos de más lados, puesto que no podrían ser bosques).

In[]:= AñadirLado2[matrizadyacencia_]:=Module[{i,j,matriz},

⋮ ⋮

In[]:= QUITARISOMORFOS2[listamatrices_]:=Module[

⋮ ⋮

In[]:= n=8;
nLados=7;
matrizadyacencia=Table[0,{i,n},{j,n}];

⋮ ⋮

Out[] =

⋮ ⋮

Total: 215 Tiempo empleado: 0.875

Ahora determinamos cuáles de los 215 grafos son bosques, previamente definimos todas las funciones necesarias (6.10., 7.13., 7.14., 8.21. y 8.22.):

In[]:= CONEXO[A_]:=Module[{i,j,B},

⋮ ⋮

In[]:= COMPONENTESCONEXAS[matrizadyacencia_]:=
Module[{j,B,conjunto,temp},

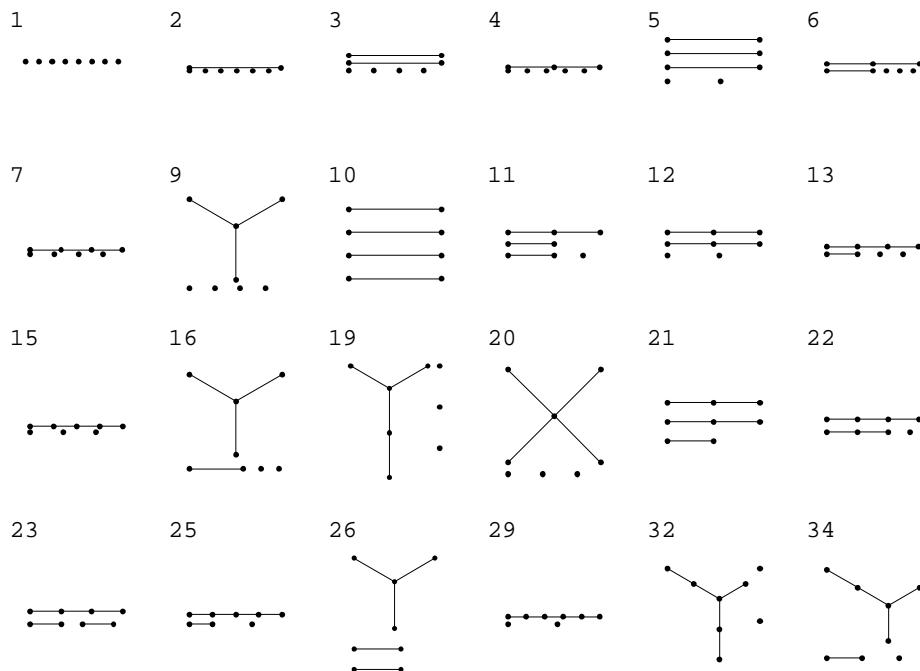
⋮ ⋮

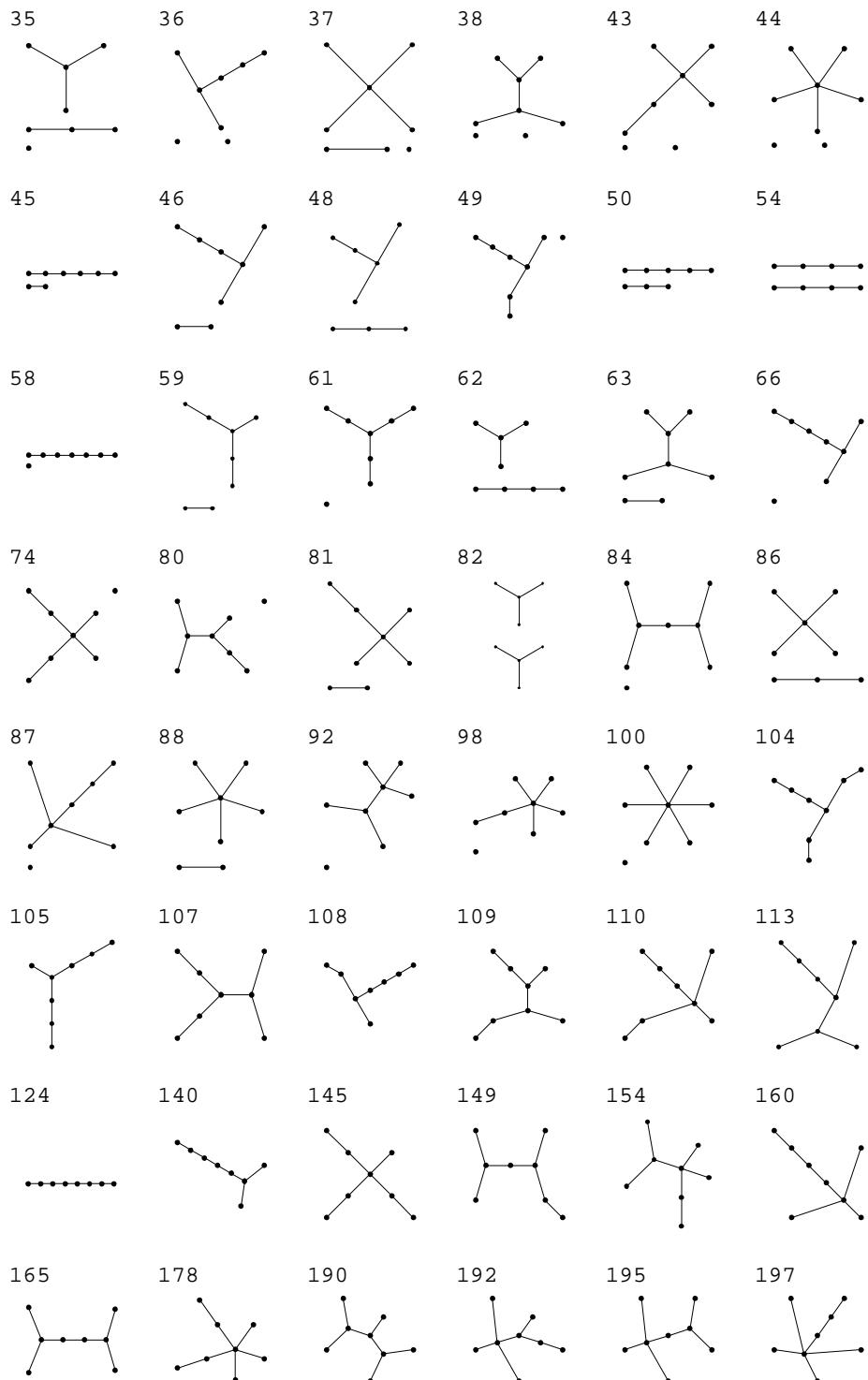
```

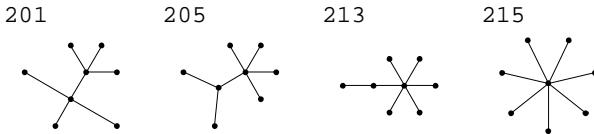
In]:= IND2[W1,matrizadyacencia]:=Module[{n,i,j},
                                         ...
                                         ];
In]:= ARBOL[matrizadyacencia]:=Module[{i,j,lados},
                                         ...
                                         ];
In]:= BOSQUE2[matrizadyacencia]:=Module[{componentes,i},
                                         ...
                                         ];
In]:= Do[
      If[BOSQUE2[listamatrices1[[i]]],
         Print[i];
         Print[GraphPlot[listamatrices1[[i]]],
                     PlotStyle->{Black,PointSize[.05]}]];
      ];
      ,{i,Length[listamatrices1]}]

```

Out[]=







Observemos que desde el 104 al 215 son los árboles de 8 vértices, luego serían los grafos que son bosques de 8 vértices y 7 lados.

Vistos ambos métodos, podríamos buscar varios ejemplos y medir su eficacia (ejercicio 8.46.).

4. OTROS EJEMPLOS

Concluimos el capítulo con otros ejemplos donde estudiaremos conjuntamente algunas de las propiedades más interesantes que hemos visto.

Ejemplo 8.30. Determinar si el siguiente grafo es plano, conexo, árbol, determinar los ciclos elementales que parten del vértice 1 de longitud 5, colorearlo y colorear sus regiones.

In[1]:=

matrizadyacencia={

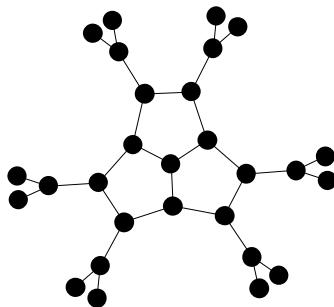
```
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},  

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},  

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
```

In[]:= **GraphPlot[matrizadyacencia,**
PlotStyle->{Black,PointSize[.05]}]

Out[]:=



a) Plano:

Introducimos todas las funciones necesarias:

In[]:= <<“Combinatorica`”;

In[]:= **D3[matrizadyacencia_,i_,j_]:=Table[**

⋮ ⋮

In[]:= **QUITARVERTICESGRADO210[matrizadyacencia_]:=**

⋮ ⋮

In[]:= **K33[W_,matrizadyacencia_]:=Module[**

⋮ ⋮

In[]:= **HOMEOMORFOK5K33[matrizadyacencia_]:=Module[**
{Subconjuntos,i,j,cont1,matrizadyacenciaK5,Nueva},

⋮ ⋮

In[]:= **MATRIZADYACENCIA[W_,F_]:=Module[{i,j,k},**

⋮ ⋮

In[]:= **PLANO[matrizadyacencia_]:=Module[**
{Nueva,k2,k3,subcladoslados},

⋮ ⋮

Comprobamos si es plano:

In[]:= **PLANO[matrizadyacencia]**

Out[] = True

b) Conexo:

In[]:= **CONEXO[A_]:=Module[{i,j,B},**

⋮ ⋮

In[]:= **CONEXO[matrizadyacencia]**

Out[] = True

c) No es bosque, comprobamos si es Árbol:

In[]:= **CICLOSELEMENTALES[vertice1_,longitud_,A_]:=**

⋮ ⋮

In[]:= **BOSQUE[matrizadyacencia_]:=Module[{n,longitud,i},**

⋮ ⋮

In[]:= **BOSQUE[matrizadyacencia]**

Out[] = False

O directamente con la función 8.21.:

In[]:= **ARBOL[matrizadyacencia_]:=Module[{i,j,lados},**

⋮ ⋮

In[]:= **ARBOL[matrizadyacencia]**

Out[] = False

d) Calculamos los ciclos elementales que parten del vértice 1 de longitud 5:

In[]:= **CICLOSELEMENTALES[1,5,matrizadyacencia]**

Out[]= {1,2,5,6,3,1},{1,2,10,9,4,1},{1,3,6,5,2,1},
{1,3,7,8,4,1},{1,4,8,7,3,1},{1,4,9,10,2,1}}

- e) Lo coloreamos, alteramos ligeramente la función COLOREAR[] para que la representación sea más clara:

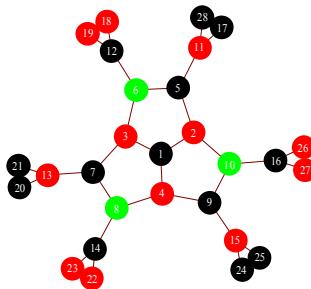
In[]:= **COLOREAR[matrizadyacencia_]:=Module[**
{colores,coloreartemp,COLORES,color,i,j},
 $\vdots \qquad \vdots$
GraphPlot[matrizadyacencia,
VertexRenderingFunction->({COLORES[[colorear[[#2]]]],
Disk[#1,0.3],White,Text[#2,{#1[[1]],#1[[2]]}]}&)]]

In[]:= **COLOREAR[matrizadyacencia]**

Out[]= Colores asignados:
Vértice 1, color Negro

$\vdots \qquad \vdots$

Vértice 28, color Negro



Como contiene pentágonos y hemos encontrado 3-coloraciones, entonces podemos asegurar que es 3-cromático. Obsérvese que por el número de vértices implicados, el uso del método combinatorio para el cálculo de una coloración óptima, implicaría una gran cantidad de proceso, quedando patente su ineficacia y la necesidad de estudiar otros métodos como los vistos.

- d) Por último calculamos una coloración de las regiones, observemos los nombres de los vértices (o la asignación de índices) del grafo y con ella obtenemos los ciclos que determinan la representación gráfica plana del mismo que deseemos tener en cuenta, en este caso:

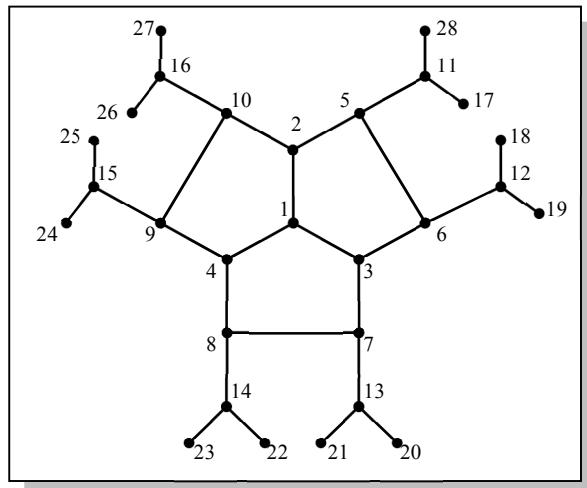


Ilustración 8.13.

Introducimos las regiones (incluida la ilimitada o infinita):

```
In]:= REGIONES={{1,2,10,9,4,1},{1,2,5,6,3,1},{1,3,7,8,4,1},
{2,5,11,28,11,17,11,5,6,12,18,12,19,12,6,3,7,13,20,13,21,13,7,
8,14,22,14,23,14,8,4,9,15,24,15,25,15,9,10,16,,26,16,
27,16,10,2}};
```

Calculamos el grafo asociado a las regiones:

```
In]:= A=GRAFOREGIONES[REGIONES]
Out]= {{0,1,1,1},{1,0,1,1},{1,1,0,1},{1,1,1,0}}
```

Observamos que la salida es la matriz de adyacencia de K_4 , por tanto será 4-cromático, en efecto, buscamos las 4-coloraciones y 3-coloraciones o directamente podemos usar la función 8.2.:

```
In]:= NCOLORACIONES[matrizadyacencia_,lambda_]:=Module[
{coloracionestemp,t,h,i,j,colorady,k},
```

⋮ ⋮

```
In]:= NCOLORACIONES[A,3]
```

```
Out]= {}
```

```
In]:= NCOLORACIONES[A,4]
```

```
Out]= {{1,2,3,4},{1,2,4,3},{1,3,2,4},{1,3,4,2},
{1,4,2,3},{1,4,3,2},{2,1,3,4},{2,1,4,3},
{2,3,1,4},{2,3,4,1},{2,4,1,3},{2,4,3,1},
{3,1,2,4},{3,1,4,2},{3,2,1,4},{3,2,4,1}},
```

```
{3,4,1,2},{3,4,2,1},{4,1,2,3},{4,1,3,2},
{4,2,1,3},{4,2,3,1},{4,3,1,2},{4,3,2,1}}
```

In[]:= **NUMEROYCROMATICO[matrizadyacencia_]:=Module[{},**

⋮ ⋮

In[]:= **NUMEROYCROMATICO[matrizadyacencia]**

Out[]= 4

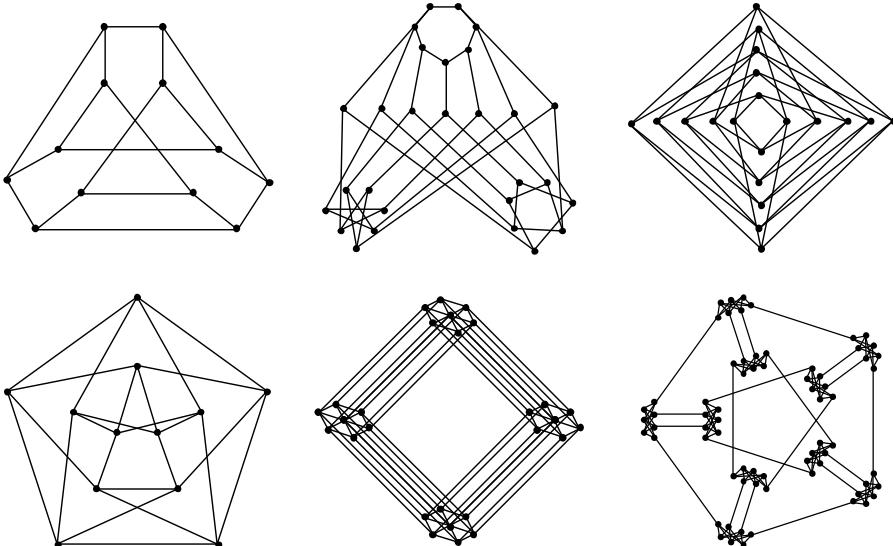
□

Ejemplo 8.31. En este ejemplo estudiaremos algunos de los objetos de tipo grafo asociados a grafos especiales incluidos en el paquete <<Combinatorica` (versión 6 o posterior de Mathematica), utilizando para ello las funciones incluidas en el mismo paquete.

In[]:= **<<Combinatorica`**

In[]:= **grafos={FranklinGraph,CoxeterGraph,FolkmanGraph,
ChvatalGraph,Hypercube[5],MeredithGraph};
ShowGraphArray[{grafos[[1;;3]],grafos[[4;;6]]}]**

Out[]=



Comprobamos que ninguno de ellos es plano o árbol:

In[]:= **Do[Print[i," Plano: ",PlanarQ[grafos[[i]]],
" Árboles: ",TreeQ[grafos[[i]]]],{i,Length[grafos]}];**

Out[]= 1 Plano: False Árbol: False

```

2   Plano: False    Árbol: False
3   Plano: False    Árbol: False
4   Plano: False    Árbol: False
5   Plano: False    Árbol: False
6   Plano: False    Árbol: False

```

Calculamos los números cromático y coloraciones óptimas:

```

In]:=      numeroscromaticos=
            Table[ChromaticNumber[grafos[[i]]],{i,Length[grafos]}]

Out]=      {2,3,2,4,2,3}

In]:=      optimas=Table[MinimumVertexColoring[grafos[[i]]]
            ,{i,Length[grafos]}]

Out]=      {{1,2,2,1,1,2,2,1,1,2,1,2},
            {1,2,2,1,1,2,3,2,1,2,1,1,2,1,1,2,3,3,3,1,2,2,1,
            3,3,2,1,3},
            {1,1,1,1,1,2,2,2,2,2,2,2,2,2,1,1,1,1,1},
            {1,2,1,1,3,1,2,3,2,3,4,2},
            {1,2,1,2,2,1,2,1,2,1,2,1,2,1,2,1,2,2,1,2,
            1,2,1,2,1,1,2,1,2},
            {1,1,1,1,2,2,2,1,1,1,1,2,2,2,2,2,2,1,1,1,2,2,
            2,2,1,1,1,2,1,1,1,3,3,3,1,1,1,1,2,2,2,2,2,2,2,
            1,1,1,1,2,2,1,3,3,3,2,2,2,2,1,1,1,1,1,3,2,2,
            2}}

```

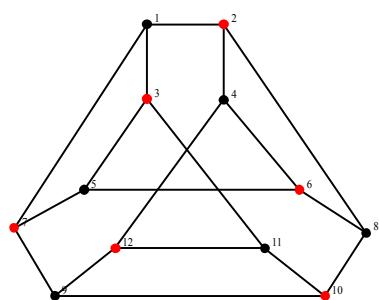
Representamos las coloraciones óptimas con ShowGraph[]:

```

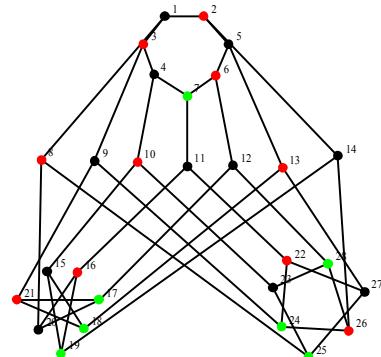
In]:=      Do[
            optima=optimas[[k]];
            numerocromatico=numeroscromaticos[[k]];
            coloracion={};
            colores={Black,Red,Green,Blue,Pink,Yellow,Orange};
            Do[
                color={VertexColor->colores[[i]]};
                Do[
                    If[optima[[j]]==i,PrependTo[color,j]];
                    ,{j,Length[optima]}];
                    AppendTo[coloracion,color];
                    ,{i,numerocromatico}];
                    Print>ShowGraph[grafos[[k]],Table[coloracion[[i]],
                    ,{i,numerocromatico},VertexLabel->True]]
                    ,{k,Length[grafos]}];

```

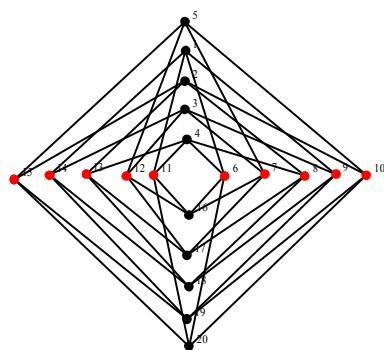
```
Out]=
```



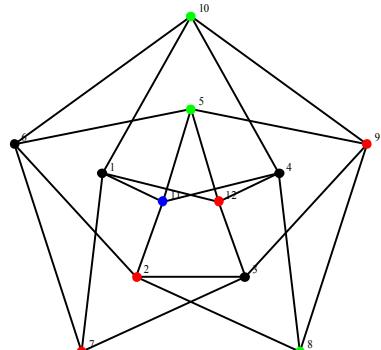
Negro: 1, 4, 5, 8, 9 y 11.
Rojo: 2, 3, 6, 7, 10 y 12.



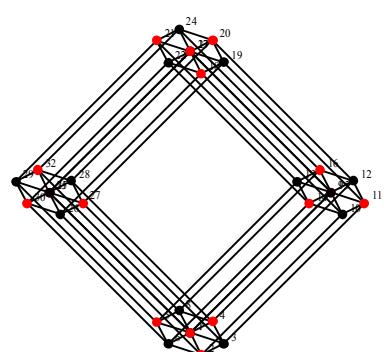
Negro: 1, 4, 5, 9, 11, 12, 14, 15, 20, 23 y 27.
Rojo: 2, 3, 6, 8, 10, 13, 16, 21, 22 y 26.
Verde: 7, 17, 18, 19, 24, 25 y 28.



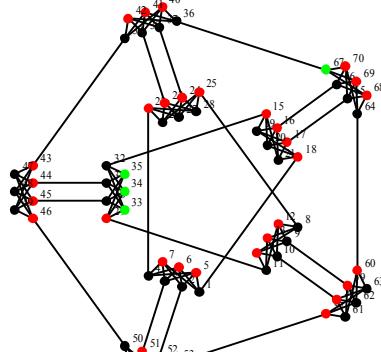
Negro: 1, 2, 3, 4, 5, 16, 17, 18, 19 y 20.
Rojo: 6, 7, 8, 9, 10, 11, 12, 13, 14 y 15.



Negro: 1, 3, 4 y 6.
Rojo: 2, 7, 9 y 12.
Verde: 5, 8 y 10.
Azul: 11.



Negro: 1, 3, 6, 8, 10, 13, 15, 17, 19, 22, 24, 26, 28, 29 y 31.
Rojo: 2, 4, 5, 7, 9, 11, 14, 16, 18, 20, 21, 23, 25, 27, 30, 32.



Negro: 1, 2, 3, 4, 8, 9, 10, 11, 19, 20, 21, 26, 27, 28, 30, 31, 32, 36, 37, 38, 39, 47, 48, 49, 50, 53, 61, 62, 63, 64, 65 y 66.
Rojo: 5, 6, 7, 12, 13, 14, 15, 16, 17, 18, 22, 23, 24, 25, 29, 40, 41, 42, 43, 44, 45, 46, 51, 52, 57, 58, 59, 60, 68, 69 y 70.

Verde: 33, 34, 35, 54, 55, 56 y 67.



Ejemplo 8.32. La gestión del tráfico urbano controlada por semáforos puede modelarse mediante teoría de grafos y reducirse a un problema de coloración de grafos que puede tratarse y resolverse computacionalmente.

Uno de los mayores problemas del tráfico urbano consiste en encontrar una regulación de los semáforos óptima que permita circular al mayor número de vehículos por unidad de tiempo de manera eficaz y sin generar cuellos de botella. Antes de entrar en un análisis más técnico y profundo del problema, exploremos un par de situaciones reales que vamos a modelar usando teoría de grafos y que clarifiquen la situación, dándonos las claves para la resolución del problema.

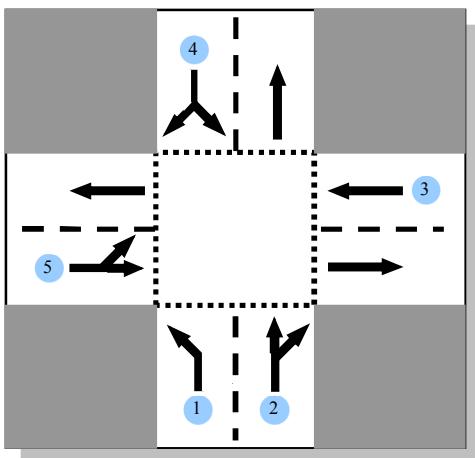


Ilustración 8.14.

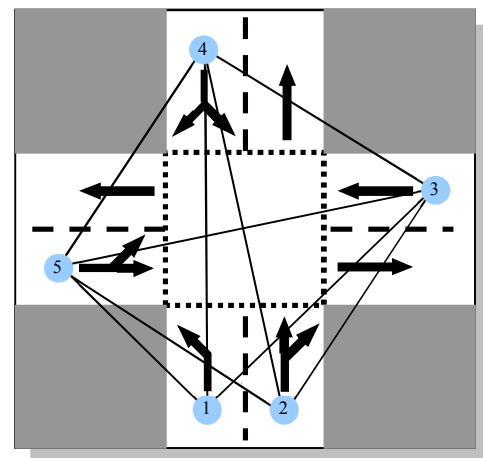


Ilustración 8.15.

Por ejemplo, si pretendemos regular el tráfico en un determinado cruce mediante semáforos, en primer lugar, debemos averiguar cuántos turnos son necesarios para que todos los coches puedan pasar por ese punto y describir qué coches son los que pueden circular en cada turno. Supongamos un cruce como el descrito por la ilustración 8.14., donde damos nombre a cada una de las direcciones que entran en el cruce y que pueden colisionar.

La solución más burda al problema consistiría en dividirlo en cinco turnos, uno por cada una de las direcciones que hemos marcado en la ilustración 8.14. Sin embargo, podemos dar una respuesta mejor, si nos fijamos en la ilustración 8.14., notaremos que 1 y 2 pueden circular simultáneamente. Analicemos el mismo problema usando la teoría de grafos, para ello vamos a unir con una línea aquellas direcciones que tienen

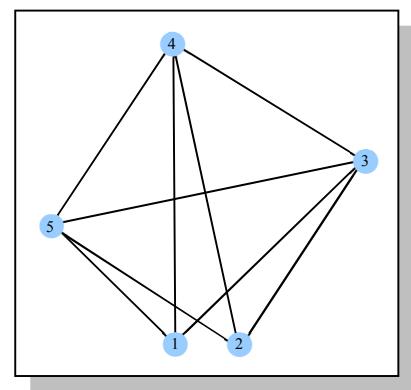


Ilustración 8.16.

posibilidad de colisionar en el cruce si circularan por él simultáneamente, ilustración 8.15.

Si nos quedamos únicamente con los puntos y las líneas, tendremos el grafo $G = (W, F)$ con: $W = \{1, 2, 3, 4, 5\}$ el conjunto de vértices y $F = \{\{1, 3\}, \{1, 4\}, \{1, 5\}, \{2, 3\}, \{3, 4\}, \{2, 5\}, \{3, 4\}, \{3, 5\}, \{4, 5\}\}$ el de lados.

Y el problema se reduciría a colorear los vértices de dicho grafo de forma que dos vértices adyacentes no estén coloreados del mismo color, y donde cada turno vendría dado por un color. Cada coloración del grafo sería una solución al problema, si queremos determinar el menor número de turnos necesarios, éste vendría dado por el número cromático (menor número de colores necesario para colorear el grafo) del grafo. La 5-coloración del grafo se correspondería con la solución más burda del problema y que describíamos en primer lugar. Lo resolvemos con las herramientas de la sección 1:

```
In]:= matrizadyacencia={\{0,0,1,1,1\},\{0,0,1,1,1\},\{1,1,0,1,1\}
,\{1,1,1,0,1\},\{1,1,1,1,0\}\};

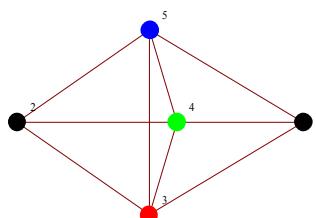
In]:= NCOLORACIONES[matrizadyacencia_lambda_]:=Module[
{coloracionestemp,t,h,i,j,colorady,k},
:
:
]

In]:= NUMERO CROMATICO[matrizadyacencia_]:=Module[{},
:
:
]

In]:= DIBUJAR[matrizadyacencia_]:=Module[{i},
:
:
]

In]:= DIBUJAR[matrizadyacencia]

Out]= Colores asignados:
Vértice 1, color Negro
Vértice 2, color Negro
Vértice 3, color Rojo
Vértice 4, color Verde
Vértice 5, color Azul
```



Ahora nos planteamos el problema con otro cruce algo más complicado (ilustración 8.17.). Un análisis directo parece más confuso, por ello indicamos las trayectorias que describen los coches según la dirección de la que provengan y a las que se dirijan. Describimos el grafo, uniendo aquellos vértices que se correspondan con las direcciones que en la ilustración 8.17. se tocan o cruzan y por tanto tienen posibilidades de colisionar.

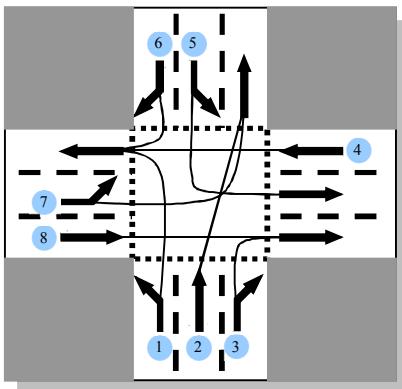


Ilustración 8.17.

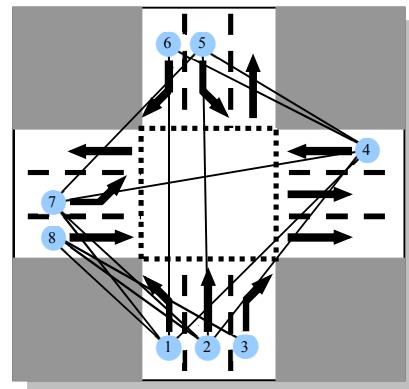


Ilustración 8.18.

Por último, nos quedamos con el grafo $G = (W, F)$ de la ilustración 8.19.: $W = \{1, 2, 3, 4, 5, 6, 7, 8\}$ y $F = \{\{1, 4\}, \{1, 6\}, \{1, 7\}, \{1, 8\}, \{2, 4\}, \{2, 5\}, \{2, 7\}, \{2, 8\}, \{3, 8\}, \{4, 5\}, \{4, 6\}, \{4, 7\}, \{5, 7\}\}$.

Cualquier coloración del grafo soluciona el problema, las más interesantes serán obviamente las coloraciones óptimas (con menor número de colores), en este caso el número cromático del grafo es 4, por tanto el menor número de turnos será 4. Lo hacemos con el ordenador:

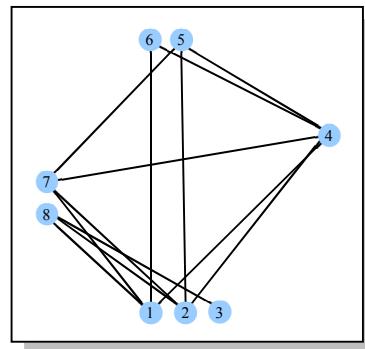
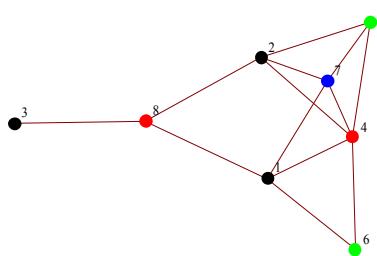


Ilustración 8.19.

```
In[]:= W={1,2,3,4,5,6,7,8};  
F={1->4,1->6,1->7,1->8,2->4,2->5,2->7,2->8,3->8,4->5,4->6,  
4->7,5->7};  
  
In[]:= MATRIZADYACENCIA[W_,F_]:=Module[  
    :  
    :  
];  
  
In[]:= matrizadyacencia=MATRIZADYACENCIA[W,F];  
DIBUJAR[matrizadyacencia]  
  
Out[]=  
Colores asignados:  
Vértice 1, color Negro  
Vértice 2, color Negro
```

Vértice 3, color Negro
 Vértice 4, color Rojo
 Vértice 5, color Verde
 Vértice 6, color Verde
 Vértice 7, color Azul
 Vértice 8, color Rojo



Con NCOLORACIONES[] podemos calcular todas las posibles coloraciones con 4 colores, podemos comprobar:

In[]:= Length[NCOLORACIONES[matrizadyacencia,4]]

Out[]= 720

que hay 720 posibles coloraciones, por cada coloración hay otras 24 que resultan al permutar los 4 colores y que son equivalentes para el problema, por tanto bastaría con analizar sólo 30, y en la realidad nos quedaríamos con la coloración más favorable según las condiciones particulares del tráfico, por ejemplo del número de coches que circulan por cada dirección. De lo anterior se deduce que habría que tener en cuenta el volumen de coches que avanza por cada dirección para distribuir los turnos de la manera más eficaz, otro factor a tener en cuenta es el tiempo que se le dará a cada turno, que en la realidad es variable y de nuevo dependerá de la cantidad de coches que circulan por cada dirección. Por ejemplo, imaginemos que el tiempo medio necesario para que pase un coche por el cruce es de 5 segundos y que el número de coches a cierta hora que circula por cada dirección cada 3 minutos viene dado por la siguiente tabla:

| Dirección | Nº de coches | Tiempo mínimo cada 3 minutos necesario por turno |
|-----------|--------------|--|
| 1 | 8 | 40 |
| 2 | 9 | 45 |
| 3 | 3 | 15 |
| 4 | 7 | 35 |
| 5 | 2 | 10 |
| 6 | 3 | 15 |
| 7 | 5 | 25 |
| 8 | 1 | 5 |

Tabla 8.2.

Con la coloración que hemos obtenido, precisamos de:

$$45 \text{ (turno negro)} + 35 \text{ (turno rojo)} + 15 \text{ (turno verde)} + 25 \text{ (turno azul)} = 120 \text{ segundos,}$$

para desalojar el tráfico que se acumula cada 3 minutos, debemos añadir un tiempo que se pierde por cada cambio de turno, supongamos por ejemplo que es de 6 segundos, esto es, +24 segundos por ciclo, por tanto el tiempo mínimos será de 144 segundos por ciclo.

Buscamos una coloración en la que las direcciones con peso similar (tráfico similar) estén en el mismo turno:

| Dirección | Nº de coches | Tiempo mínimo necesario por dirección |
|-----------|--------------|---------------------------------------|
| 1 | 8 | 40 (ROJO) |
| 2 | 9 | 45 (ROJO) |
| 3 | 3 | 15 (VERDE) |
| 4 | 7 | 35 (VERDE) |
| 5 | 2 | 10 (AMARILLO) |
| 6 | 3 | 15 (AZUL) |
| 7 | 5 | 25 (AZUL) |
| 8 | 1 | 5 (AMARILLO) |

Tabla 8.3.

Ahora, el tiempo mínimo necesario es de $45 + 25 + 35 + 10 + 24 = 139$ segundos. Para las condiciones del ejemplo, podemos asegurar que este es el menor tiempo, puesto que dentro del grafo distinguimos que los vértices 2, 4, 5 y 7 están conectados entre sí por un lado dos a dos. Debemos optimizar la duración de cada turno teniendo en cuenta además que cada cambio supone una pérdida de tiempo (a turnos más largos menos tiempo se pierde), pero por otra parte si los turnos son muy largos la espera media de cada coche hasta que llegue su turno también sube. Supongamos que nos limitamos a que el ciclo dure exactamente los 3 minutos, el tiempo restante hasta completar los 3 minutos sería distribuido entre los turnos, proporcionalmente al número de coches:

| Turno | Nº de coches | Tiempo asignado |
|--------------|--------------|---------------------------------------|
| Rojo | 85 | $45 + 18.33$ |
| Verde | 50 | $35 + 10.80$ |
| Azul | 40 | $25 + 8.63$ |
| Amarillo | 15 | $10 + 3.23$ |
| TOTAL | 190 | $155.99 + 24 = 180$ |

Tabla 8.4.

Según la hora del día, el volumen de tráfico por cada dirección es variable, por tanto estos turnos deberían ajustarse según las necesidades propias de la franja horaria, optando por la configuración más favorable para cada momento.



5. EJERCICIOS

Ejercicio 8.1. Encontrar una coloración óptima para los siguientes grafos:

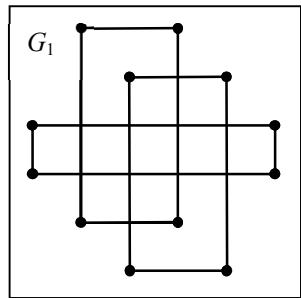


Ilustración 8.20.

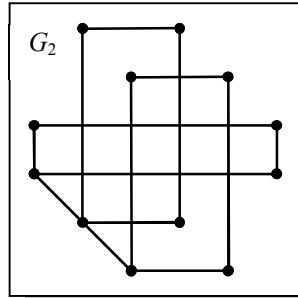


Ilustración 8.21.

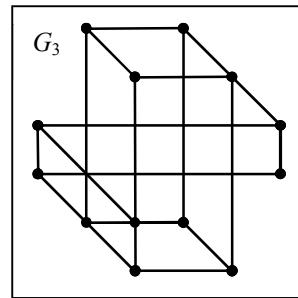


Ilustración 8.22.

¿Cómo serían el resto de coloraciones óptimas de los grafos anteriores?

□

Ejercicio 8.2. Calcular el polinomio cromático de los grafos anteriores. Si fuese necesario utilizar el teorema de descomposición para polinomios cromáticos.

□

Ejercicio 8.3. Se va a realizar un examen en un aula donde los pupitres tienen la siguiente disposición:

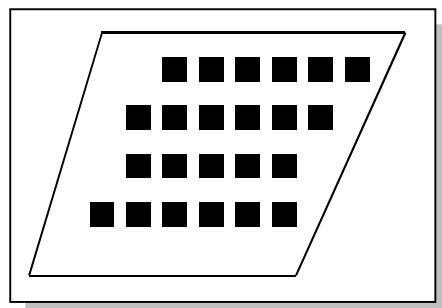


Ilustración 8.23

Sabiendo que el aula está completamente llena y que los alumnos que se encuentren cerca (esto es, uno al lado de otro en vertical, horizontal o diagonal) no podrán hacer el mismo examen, ¿cuántos modelos de examen son necesarios?

□

Ejercicio 8.4. Un zoológico tiene la estructura del grafo de la ilustración 8.11. donde los vértices representan las zonas del zoológico, siendo adyacentes aquellos vértices que representan a zonas que están cerca. Algunos animales no pueden tener cerca otros, los clasificamos en rumiantes, grandes depredadores, pequeños depredadores y aves, los grandes depredadores no pueden tener cerca de los rumiantes y pequeños depredadores no

pueden tener cerca de las aves, ¿cómo podríamos organizar el zoológico?, ¿de cuántas formas podemos organizarlo? □

Ejercicio 8.5. Existen multitud de problemas que pueden resolverse usando la coloración de vértices, además de los vistos en los ejercicios anteriores, por ejemplo: la situación de los medicamentos en una farmacia, las sustancias químicas en un laboratorio, las secciones de un supermercado... Diseñar un grafo que represente las distintas secciones de una gran superficie teniendo en cuenta que algunas secciones no pueden estar al lado de otras, por ejemplo carnicería y perfumería. □

Ejercicio 8.6. Coloración de los lados de un grafo. En algunas ocasiones resulta interesante la coloración de los lados de un grafo, entendiendo por “coloración de lados”, la asignación de colores a los lados del grafo de forma que dos lados incidentes con un mismo vértice no tengan asignado el mismo color, este problema es fácilmente reducible a otro de coloración de vértices, definiendo un nuevo grafo cuyos vértices se identifican con los lados y estos serán adyacentes cuando estén asociados a lados incidentes con un mismo vértice. Hacer un estudio y programar funciones que coloren los lados de un grafo. En la versión 6 y siguientes de Mathematica y dentro del paquete <<Combinatorica` disponemos de funciones que hacen esto mismo sobre un grafo asociado a un objeto de tipo grafo “G”:

EdgeColoring[G] y EdgeChromaticNumber[G] □

Ejercicio 8.7. Las funciones 8.4, 8.5. y 8.6., calculan coloraciones de un grafo de forma eficaz, si bien, no necesariamente óptima. Podemos proponer nuevas ideas para generar otros algoritmos de coloración:

- Diseñar un programa que coloree grafos empezando la asignación de colores por aquellos vértices de mayor grado y terminando por los de menor (para más detalle, véase el libro: “Matemática Discreta: Problemas y Ejercicios Resueltos”, García, C., López, J.M. y Puigjaner, D. [23]) ¿Se calcula por este método siempre una coloración óptima?
 - Algoritmo de Welch-Powell. Combinar la técnica del apartado anterior con el algoritmo voraz (8.6.). ¿Se calcula por este método siempre una coloración óptima?
 - Combinar la técnica del apartado a) con el algoritmo usado en la función 8.5. ¿Se calcula por este método siempre una coloración óptima?
-

Ejercicio 8.8. Los grafos bipartitos son siempre 2-coloreables³⁹, mientras que un grafo completo K_n es n -cromático. Comprobarlo para $K_{2,3}$, $K_{4,4}$, K_6 y K_7 . □

³⁹ Desde la versión 6 de Mathematica y dentro del paquete <<Combinatorica` , disponemos de la función:

TwoColoring[G]

ésta encuentra una 2-coloración de un objeto de tipo grafo “G” que sea bipartito.

Ejercicio 8.9. Determinar cuáles de los siguientes grafos son planos:

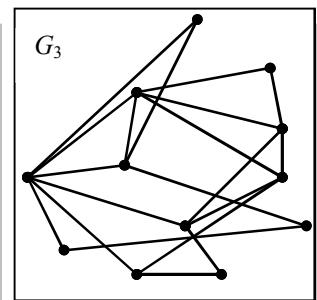
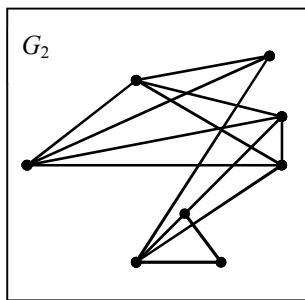
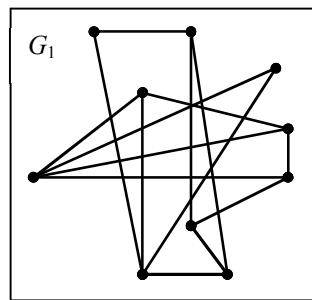


Ilustración 8.24.

Ilustración 8.25.

Ilustración 8.26.

Ejercicio 8.10. Calcular una representación gráfica plana de los grafos del ejercicio anterior que son planos. Introducir dicha representación en el ordenador, representarla gráficamente y calcular una coloración de las regiones.

□

Ejercicio 8.11. Crear una función que calcule la matriz de adyacencia de un grafo plano a partir de todos los ciclos que delimitan sus regiones.

□

Ejercicio 8.12. Programar una pequeña rutina que determine el número de regiones de un grafo plano desde su matriz de adyacencia.

□

Ejercicio 8.13. Coloración de un grafo plano. Los grafos planos son 4-coloreables (teorema de los cuatro colores⁴⁰). Calcular una 4-coloración y el número cromático de todos los grafos planos que han aparecido en el capítulo.

□

Ejercicio 8.14. La región ilimitada podemos calcularla a partir de las otras. Nos restringiremos a una única componente conexa (en el caso de tener un grafo no conexo relegamos su estudio al ejercicio 8.39.), calcularemos el ciclo (no orientado) que bordea a todo el grafo y con él el grado de dicha región, este ciclo estará compuesto por aquellos lados que sólo aparezcan en un único ciclo de los que definen a las regiones:

| FUNCIÓN | COMENTARIOS |
|---|---|
| <code>SUCLADOS2[vertices]:=...</code> | Definimos la función 8.17. |
| <code>REGILIM[REGIONES_]:=Module[{regioneslados,i,j,k,ladosext,resto},</code> | Por entrada tendrá la lista de todas las regiones de un grafo conexo excepto la infinita. |

⁴⁰ Este teorema afirma que el número cromático de cualquier grafo plano es menor o igual que cuatro. Fue conjecturado en 1850 por Francis Guthrie y demostrado en 1976 por K. Appel y W. Haken utilizando un ordenador, para más detalle ver su artículo: *The solution of the four color problem*, Scientific American, 237, octubre 1977, págs.. 108-121.

| | |
|--|--|
| <pre> regioneslados={}; Do[AppendTo[regioneslados, SUCLADOS2[REGIONES[[i]]]] ,{i,Length[REGIONES]}]; ladosext={}; Do[resto={}; Do[If[i!=j,resto=Union[resto,regioneslados[[j]]]]; ,{j,Length[regioneslados]}]; ladosext=Union[ladosext, Complement[regioneslados[[i]],resto]]; ,{i,Length[regioneslados]}]; ciclo={ladosext[[1]][[1]],ladosext[[1]][[2]]}; ladosext=Complement[ladosext,{ladosext[[1]]}]; While[ladosext!={}, k=1; While[ladosext[[k]][[1]]!=Last[ciclo] && ladosext[[k]][[2]]!=Last[ciclo],k++]; If[ladosext[[k]][[1]]==Last[ciclo], AppendTo[ciclo,ladosext[[k]][[2]]]; , AppendTo[ciclo,ladosext[[k]][[1]]]; ladosext=Complement[ladosext,{ladosext[[k]]}];]; ciclo]; </pre> | Calculamos los lados que delimitan a la región externa. |
| <pre>]; </pre> | Con los lados que hemos calculado determinados el ciclo que bordea al grafo completamente. |
| <pre>]; </pre> | Salida de resultado. |

Función 8.23. Región ilimitada de un grafo plano conexo.

Esta última función calculará la región ilimitada desde las anteriores, siempre y cuando no tenga caminos o vértices colgantes, en cuyo caso no funcionará porque no los tendrá en cuenta, si bien para la coloración de regiones este extremo no es relevante y además podrían eliminarse fácilmente.

Por ejemplo, si la aplicamos a la representación plana del grafo de la ilustración 8.8.:

In[]:= **REGIONES**= $\{\{1,2,4,6,5,3,1\}, \{5,8,6,4,2,1,3,5\}, \{5,6,7,5\}, \{6,7,8,6\}, \{7,8,10,12,11,9,7\}\}$;

Y podemos determinar la región ilimitada:

In[]:= **REGIONILIM[REGIONES]**

Out[] = {5, 7, 9, 11, 12, 10, 8, 5}

- a) Programar una función que determine de forma correcta la región ilimitada, incluso cuando existen vértices colgantes.
- b) Aplicarla al grafo de la ilustración 8.11.

- c) ¿Cómo podemos interpretar que los árboles o bosques sólo tengan una única región? ¿cómo funciona en tal caso el programa 8.25.? □

Ejercicio 8.15. En el diseño de circuitos electrónicos sobre una placa impresa, una de las cuestiones imprescindibles a tener en cuenta en su diseño es que no se corten y por tanto cortocircuiteen los lados del grafo que formaliza a dicho circuito, en consecuencia, dicho grafo deberá ser plano.

- a) Considerar la representación del circuito lógico de la función booleana $f(x, y, z) = (x \wedge (y \vee z), ((\sim y) \oplus z) \downarrow x)$, ¿podrán integrarse en una placa impresa de una cara?
 b) ¿Podemos diseñar un circuito que conecte tres puntos distintos, cada uno de ellos con otros tres en una misma tarjeta impresa y sólo por una cara?
-

Ejercicio 8.16. Determinar el menor número de colores necesarios para colorear las Comunidades Autónomas españolas de la Península Ibérica (véase la ilustración 7.12.). □

Ejercicio 8.17. Coloración de un árbol. Calcular el número cromático de todos los árboles que aparecen en el capítulo. ¿Qué puede decirse sobre el número cromático de un árbol cualquiera?

□

Ejercicio 8.18. Diámetro de un árbol. En un árbol dos vértices cualesquiera están conectados por un único camino simple cuya longitud será la distancia entre ambos vértices, el diámetro de un árbol coincidirá con la distancia de los dos vértices más alejados entre si. Programar una rutina que determine el diámetro de un árbol.

□

Ejercicio 8.19. Árboles de expansión. Entenderemos por árbol de expansión de un grafo conexo G como el subgrafo G conexo con menor número de aristas. Programar una función que determine el árbol de expansión de un grafo conexo.

□

Ejercicio 8.20. En Mathematica, se dispone de la función TreePlot[], especialmente diseñada para la representación de árboles. Probar dicha función con los árboles del capítulo.

□

Ejercicio 8.21. Los directorios o carpetas de una unidad de ordenador (disco duro, CDROM, pen drive, discos flexibles,...) se pueden representar mediante un árbol que se suele llamar árbol de directorios. Si desde el símbolo del sistema de Windows XP escribimos el comando Tree, podremos comprobarlo. Introducir la matriz de adyacencia de este árbol en Mathematica y demostrar que no contiene ciclos.

□

Ejercicio 8.22. Grado de parentesco. Uno de los temas más importantes, por ejemplo legalmente o genéticamente a la hora de heredar, es el grado de parentesco. Es obvio que un árbol genealógico es un grafo, donde los individuos (ascendientes y descendientes) representan los vértices y dos vértices serán adyacentes si entre ellos existe una relación de

padre/madre–hijo/a. El grado de parentesco entre dos individuos se define como la distancia entre los dos vértices que los representan en el grafo que representa al árbol genealógico. ¿Cuál será el grado de parentesco entre dos primos hermanos?

□

Ejercicio 8.23. Sintaxis de oraciones, torneos deportivos, estructuras jerárquicas,...; también puede representarse mediante árboles. Buscar otras situaciones análogas que puedan modelarse matemáticamente o formalizarse mediante un árbol o bosque. Analizar un ejemplo particular con las herramientas del capítulo.

□

Ejercicio 8.24. Los árboles y bosques son grafos planos. Calcular una coloración de las regiones de los árboles y bosques que han aparecido en el capítulo. ¿Qué puede decirse de la coloración de las regiones de un árbol o un bosque?

□

Ejercicio 8.25. Consideramos el siguiente mosaico:

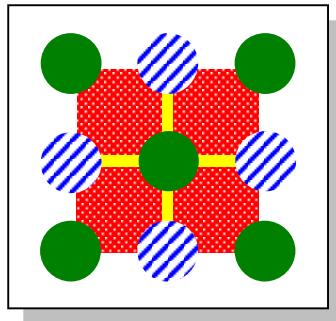


Ilustración 8.27.

¿De cuántas formas podremos colorearla con a lo sumo 4 colores? ¿Y con 5 colores?

□

Ejercicio 8.26. Dados los siguientes grafos:

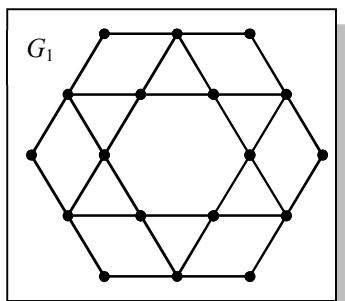


Ilustración 8.28.

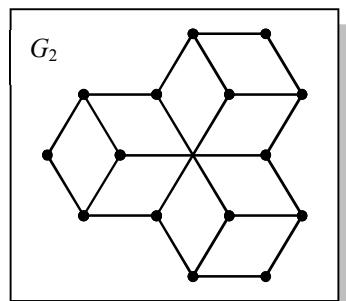


Ilustración 8.29.

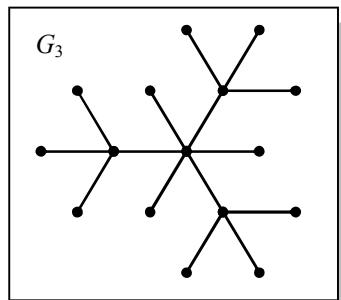


Ilustración 8.30.

- a) Comprobar que los tres verifican las condiciones necesarias propuestas en la función 8.20. para ser planos. Estimar el tiempo necesario para comprobar si son planos realmente con la función 8.19.
- b) Calcular una coloración óptima de sus vértices.
- c) Calcular una coloración óptima de regiones.
- d) ¿Cuáles son de Euler? ¿Y de Hamilton?
- e) Calcular, si es posible, un ciclo de Euler y otro de Hamilton.

Ejercicio 8.27. Determinar todos los grafos distintos salvo isomorfismo, de 8 vértices y 2-cromáticos.

Ejercicio 8.28. Calcular todos los grafos distintos salvo isomorfismo 3-coloreables de 7 vértices y con más de 12 lados.

Ejercicio 8.29. Comprobar que todos los grafos excepto K_5 de 5 vértices son planos.

Ejercicio 8.30. Calcular todos los grafos planos de 7 vértices distintos salvo isomorfismo.

Ejercicio 8.31. Utilizando lo obtenido en el ejercicio anterior, calcular todos los grafos planos y de Hamilton de 7 vértices.

Ejercicio 8.32. Comprobar el teorema de los cuatro colores (ejercicio 8.13.) para todos los grafos planos de 6 vértices.

Ejercicio 8.33. Calcular todos los árboles distintos salvo isomorfismo, de 6 vértices y 5 lados.

Ejercicio 8.34. Determinar todos los árboles y bosques de 7 y 9 vértices distintos salvo isomorfismo.

Ejercicio 8.35. Modificar 8.23 para que calcule caminos elementales entre dos vértices. □

Ejercicio 8.36. Integrar las funciones 8.23. y 8.24. en una única función que pare nada más encontrar un ciclo. Comprobar si mejora su eficacia respecto a la función 8.24. aplicando ambas funciones y midiendo los lapsos de tiempo empleados en algunos ejemplos. □

Ejercicio 8.37. En el ejemplo 8.22. comprobamos que las funciones programadas para distinguir si un grafo es plano son poco eficaces, pensar mejoras sobre las mismas y aplicarlas de nuevo al grafo del ejemplo 8.22. □

Ejercicio 8.38. La función 8.18 puede reprogramarse sin utilizar la función 8.17. y usar en su lugar una de las funciones que determina si dos grafos son isomorfos.

- a) Reprogramar 8.18 sin usar la función 8.17. (K33[]).
- b) Reprogramar la función 8.19. para que compruebe todos los subgrafos y no sólo los maximales. □

Ejercicio 8.39. Crear una función que calcule el número de regiones de un grafo cualquiera (incluso no conexo). □

Ejercicio 8.40. En el caso de grafos planos no conexos, las representaciones planas han de tener en cuenta que una componente conexa puede representarse dentro de la región ilimitada o infinita o de otra región de cualquier otra componente conexa, una forma de solucionar el problema es añadir lados ficticios (puentes) entre las componentes conexas, de esta forma, nos reduciríamos al caso conexo como podemos observar en las siguientes ilustraciones:

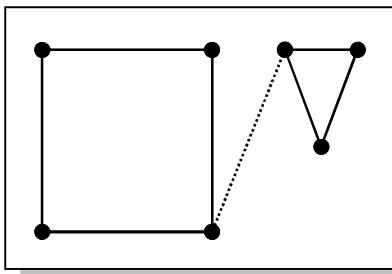


Ilustración 8.31.

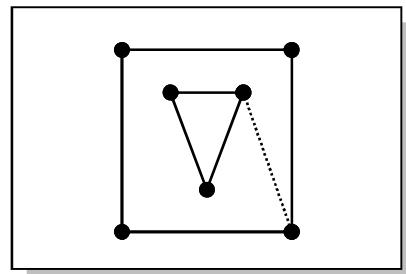


Ilustración 8.32.

- a) Analizar el problema de la coloración de regiones de un grafo plano no conexo.
- b) Programar una función que coloree las regiones de un grafo plano no conexo. □

Ejercicio 8.41. Comprobar si son planos el tetraedro, el cubo, el hipercubo 4-dimensional, el octaedro, el dodecaedro y el icosaedro. En caso afirmativo, buscar representaciones

planas y calcular una coloración óptima de las regiones para las representaciones planas calculadas.

□

Ejercicio 8.42. Crear un programa que determine una representación plana de un grafo plano. Completarlo representando gráficamente la salida obtenida.

□

Ejercicio 8.43. Implementar las ideas expuestas en el teorema 8.1. en una única función que determine eficazmente el polinomio cromático de cualquier grafo no orientado.

□

Ejercicio 8.44. Crear funciones alternativas a 8.12., que hagan los mismos cálculos utilizando:

- a) La funciones que tiene Mathematica para el cálculo de submatrices.
- b) La función D2[] (8.10.).

□

Ejercicio 8.45. La matriz de adyacencia de un grafo dirigido, multigrafo y con lazos no será necesariamente una matriz booleana y su diagonal principal no estará compuesta únicamente por ceros.

- a) ¿Cómo determinaríamos si un grafo así es plano?
- b) Crear una función que tenga por entrada la matriz de adyacencia de un grafo dirigido, multigrafo y con lazos y por salida la matriz de adyacencia del grafo no orientado que contenga toda la información necesaria para determinar si el grafo de partida es plano.

□

Ejercicio 8.46. Comprobar qué método es el más eficaz para comprobar si un grafo es un árbol o bosque: el empleado en el epígrafe 3.1. o el empleado en 3.2.

□

Ejercicio 8.47. Optimizar el tráfico del siguiente cruce como en el ejemplo 8.33.

□

Ejercicio 8.48. Para un grafo no orientado, si encontramos subgrafos que sean o contengan a K_n entonces sabremos que el número cromático será mayor o igual que n , esto es relativamente fácil de comprobar porque se traduce en la existencia en la matriz de adyacencia de una submatriz cuya diagonal principal está formada por ceros pero el resto son todos unos. Incluso podemos razonar con el teorema del número de caminos, pues cada coordenada de la diagonal principal de la potencia tercera de la matriz de adyacencia representa el doble del número

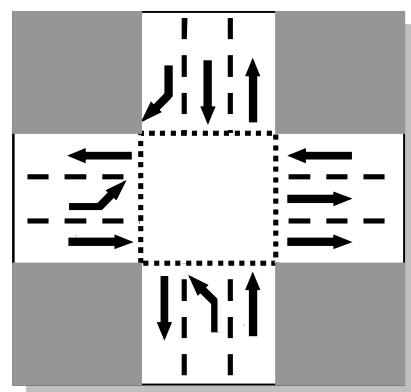


Ilustración 8.33.

de triángulos que contienen al vértice de índice dicha coordenada, podemos comprobar si el grafo contiene algún triángulo con:

In[]:= **Sum[Diagonal[MatrixPower[matrizadyacencia,3]][[i]]
,{i,Dimensions[matrizadyacencia]][[1]]}]**

Si contiene alguno, el número cromático será mayor o igual a tres y si no lo contiene, entonces el grafo no tendría ciclos de longitud 3. También podemos razonar con los ciclos de longitud superior observando la diagonal principal de las sucesivas potencias de la matriz de adyacencia.

Implementar estas ideas y comprobar cómo funcionan los nuevos algoritmos para distintos grafos. □

REFERENCIAS

A

- álgebra de Boole · 78
- anillo · 78
 - comutativo · 78
- aplicación
 - biyectiva · 149
- árbol · 472
- árbol de expansión · 500
- arco · *Véase* flecha
- arista · *Véase* lado
- arista orientada · *Véase* flecha
- aritmética básica · 15
- articulación · 400
- asociativa · 34, 43
- automorfismo · 290

B

- bipartito · 329
- bipartito completo · 329
- biyección · 46
- bosque · 472
- bucle · *Véase* lazo
- bucles · 17

C

- camino · 341, 345
 - fin · 341
 - longitud · 341
 - origen · 341
 - trivial · 341
- camino dirigido · 345
- camino trivial · 345
- centro · 400
- ciclo de euler · 374

- ciclo de Hamilton · 382
- clases laterales · 86
- coloración
 - lados de un grafo · 497
 - regiones · 471
 - vértices · 413
- coloración óptima · 414
- colorear un grafo · 413
- componentes conexas · 362
- componentes fuertemente conexas · 362
- condicional · 17
- conexo · 362
 - débilmente · 362
 - fuertemente · 362
- comutativa · 34, 44
- constante
 - E · 19
 - I · 19
 - Pi · 19
- cromático · *Véase* número cromático
- cuaternios · 80

D

- datos numéricos
 - Complex · 15
 - Integer · 15
 - Real · 15
- diámetro · 400
- digrafo · 203
- distancia · 370
- distributiva · 78
- divisor · 91
- dodecaedro · 388

E

- elemento neutro · 33, 37
- elemento simétrico · 34, 39

excentricidad · 400

F

False · 17
 flecha · 202
 lazo · 202
 Fórmula de Euler para grafos · 466
 función
 ! · Véase Not[]
 && · Véase And[]
 || · Véase Or[]
 AcyclicQ[] · 473
 AddEdge[] · 206
 AddEdges[] · 206
 AddVertex[] · 206
 AdjacencyMatrix[] · 212
 AlternatingGroup[] · 179
 And[] · 17
 Animate[] · 23
 AnimateGraph[] · 234
 Append[] · 16
 AppendTo[] · 16
 BipartiteQ[] · 331
 Block[] · 19
 Break[] · 18
 ChangeEdges[] · 206
 ChangeVertex[] · 206
 ChromaticNumber[] · 416
 ChromaticPolynomial[] · 436
 Circle[] · 20
 Clear[] · 15
 Complement[] · 16
 CompleteGraph[] · 322
 CompleteKPartiteGraph[] · 331
 CompleteQ[] · 322
 ConnectedComponents[] · 365
 ConnectedQ[] · 363
 CubicalGraph · 392
 Cycle[] · 474
 Degrees[] · 312
 DeleteCycle[] · 474
 DeleteEdge[] · 206
 DeleteEdges[] · 206
 DeleteVertex[] · 206
 Det[] · 17
 Dimensions[] · 17
 Disk[] · 20
 Divisors[] · 19, 91
 Do[] · 18
 DodecahedralGraph · 392
 EdgeChromaticNumber[] · 497

EdgeColoring[] · 497
 Edges[] · 205
 Eigenvalues[] · 17, 248
 EmptyQ[] · 205
 EulerianCycle[] · 379
 EulerianQ[] · 375
 ExactRandomGraph[] · 206
 ExtractCycleQ[] · 474
 Factor[] · 20
 FindCycleQ[] · 473
 First[] · 16
 For[] · 18
 FromAdjacencyMatrix[] · 213
 FromCycles · 171
 FromOrderedPairs[] · 206
 Function[] · 224, 225
 Graph[] · 204
 GraphComplement[] · 336
 GraphCoordinates3D[] · 234
 GraphDifference[] · 326
 GraphDistance[] · 371
 Graphics[] · 20
 GraphIntersection[] · 326
 GraphJoin[] · 326
 GraphPath[] · 371
 GraphPlot[] · 223
 GraphPlot3D[] · 234
 GraphProduct[] · 326
 GraphSum[] · 326
 GraphUnion[] · 326
 GrayLevel[] · 20
 HamiltonianCycle[] · 384
 HamiltonQ[] · 384
 IcosahedralGraph · 392
 IdentityMatrix[] · 17
 If[] · 17
 IncidenceMatrix[] · 220
 InDegree[] · 314
 Index[] · 163
 InduceSubgraph[] · 327
 Insert[] · 16
 InterpolatingPolynomial[] · 435
 Intersection[] · 16
 InversePermutation[] · 197
 Inversions[] · 175
 IsomorphicQ[] · 240
 Isomorphism[] · 241
 Join[] · 16
 KSubsets[] · 91, 329
 Last[] · 16
 Length[] · 16
 Line[] · 20
 ListGraph[] · 264
 M[] · 205

- MakeDirected[] · 206
 MakeSimple[] · 206
 MakeUndirected[] · 206
 MatrixForm[] · 17
 MatrixPower[] · 16
 MemoryInUse[] · 102
 MinimumVertexColoring[] · 416
 Minors[] · 17
 Mod[] · 19
 Module · 19
 MultipleEdgeQ[] · 205
 Not[] · 17
 NumberOfDirectedGraphs[] · 264
 NumberOfGraphs[] · 264
 OctahedralGraph · 392
 Or[] · 17
 OutDegree[] · 314
 Permutations[] · 155, 238
 PlanarQ[] · 449
 Point[] · 20
 Position[] · 16, 35
 Prepend[] · 16
 PrependTo[] · 16
 Print[] · 20
 Quotient[] · 19
 Random[] · 18
 RandomGraph[] · 206
 Rectangule[] · 20
 RegularGraph[] · 318
 RegularQ[] · 318
 RemoveMultipleEdges[] · 206
 RemoveSelfLoops[] · 206
 RGB[] · 20
 SelfLoopsQ[] · 205
 ShortestPath[] · 372
 Show[] · 20
 ShowGraph[] · 222
 ShowGraphArray[] · 234
 SignaturePermutation[] · 176
 SimpleQ[] · 205
 Simplify[] · 20
 Solve[] · 19
 Sort[] · 16, 18
 SparseArray[] · 17
 Sqrt[] · 19
 StrongComponents[] · 368
 StronglyConnectedComponents[] · 368
 StyleForm[] · 20
 SubMatrix[] · 17
 Sum[] · 19
 SymmetricGroup[] · 163
 Table[] · 16
 TableForm[] · 17
 Take[] · 17
 TakeColumns[] · 17
 TetrahedralGraph · 392
 Timing[] · 102
 ToAdjacencyMatrix[] · 213
 ToCycles[] · 171
 ToOrderedPairs[] · 206
 Transpose[] · 16, 38
 TreePlot[] · 500
 TreeQ[] · 473
 TrueQ[] · 17
 UndirectedQ[] · 205
 Union[] · 16
 Unprotect[] · 15
 UnweightedQ[] · 205
 V[] · 205
 VertexColoring[] · 432
 Vertices[] · 205
 WeakComponents[] · 365
 WeaklyConnectedComponents[] · 365
 Which[] · 17
 While[] · 18
-
- G**
- geodésica · 370
 grado de entrada de un vértice · 313
 grado de salida de un vértice · 313
 grado de un vértice · 311
 grado de una región · 467
 grafo
 (p, q)-grafo · 202
 complemento · 336
 completo · 321
 de Euler · 374
 de Hamilton · 382
 dirigido · *Véase* digrafo
 etiquetado · 202
 finito · 202
 isomorfo · 234, 235
 no orientado · 202
 orientado · 202
 ponderado · 202
 regular · 317
 simple · 202
 grafo plano · 445
 grafos
 intersección · 326
 unión · 326
 Graph[], opción
 EdgeColor · 204
 EdgeDirection · 205
 EdgeLabel · 204

EdgeLabelColor · 204
 EdgeLabelPosition · 204
 EdgeStyle · 204
 EdgeWeight · 204
 LoopPosition · 205
 VertexColor · 205
 VertexLabel · 205
 VertexLabelColor · 205
 VertexLabelPosition · 205
 VertexNumber · 205
 VertexNumberColor · 205
 VertexNumberPosition · 205
 VertexStyle · 205
 VertexWeight · 205
GraphPlot[], opción
 DataRange · 226
 DirectedEdges · 226
 EdgeLabeling · 225
 EdgeRenderingFunction · 225
 Frame · 226
 FrameTicks · 226
 Method · 225
 MultiEdgeStyle · 225
 PackingMethod · 226
 PlotRange · 226
 PlotRangeMethod · 226
 PlotStyle · 224
 SelfLoopStyle · 225
 VerteCoordinateRules · 224
 VertexLabeling · 224
 VertexRenderingFunction · 224
grupo · 33
 abeliano o commutativo · 34
 alternado · 177
 centro · 76
 cíclico · 55, 143
 cuaternios · 80
 diédrico · 184
 epimorfismo · 144
 finito · 34
 grupos de orden pequeño · 60
 homomorfismo · 45
 isomorfismo · 45
 monomorfismo · 144
 producto cartesiano · 75
 simétrico · 159
 tabla de operaciones · 34
grupo cociente · 87
grupo de automorfismos · 290
grupos isomorfos · 46

H

homeomorfo · 445

I

imagen · 77
índice · 86

L

lado · 202
 incidente · 202
 lazo · 202
 ley de composición interna · 33
 leyes de precedencia · 15
 lista · 15

M

Máquina Enigma · 190, 199
 panel de conexiones · 192
 panel de salida · 191
 reflector · 192
 rotor · 191
 teclado · 191
 matrices de adyacencia
 clase de isomorfía · 238
 matriz · 16
 dimensiones · 17
 identidad · 17
 producto · 17
 producto por escalar · 17
 suma · 17
 traspuesta · 16, 38
 matriz de adyacencia · 208
 matriz de incidencia · 219
 multigrafo · 202

N

n-coloración · 413
 n-cromático · 414
 nodo · *Véase* vértice
 núcleo · 77
 número cromático · 414

O

operación interna · *Véase* ley de composición interna
 operador lógico · 17
 orden · 34
 orden de un elemento · 96

P

paquete de funciones
 `<<Combinatorica` · 20
 `<<GraphUtilities` · 20
 paquete de funciones (versiones antiguas)
 `<<DiscreteMath`Combinatorica` · 20
 `<<DiscreteMath`GraphPlot` · 20
 permutación · 149
 ciclo · 169
 ciclos disjuntos · 169
 impar · 174
 inversión · 174
 longitud de un ciclo · 169
 notación cíclica · 170
 par · 174
 signatura · 174
 simple · 149
 trasposición · 172
 plano · 445
 Primer Teorema de isomorfía · 144
 puente · 400

R

radio · 400
 región · 466
 ilimitada · 499
 regiones adyacentes · 469
 retículo · 78
 rotación · 184

S

Show[], opción
 AspectRatio · 20
 AxesOrigin · 20
 Background · 20
 PlotRange · 20
 simetría · 184

sistemas de numeración · 199

soporte · 169

StyleForm[], opción

- Background · 20
- FontColor · 20
- FontFamily · 20
- FontSize · 20
- FontSlant · 20
- FontWeight · 20

subdivisión elemental · 445

subgrafo · 326

subgrafo maximal · 326

subgrafos inducidos · 327

subgrupo · 81, 83

- generado por un subconjunto · 96

- impropio · 85

- normal · 87

- propio · 85

subgrupo alternado · 177

subgrupo diédrico · 184

T

tabla · 16

TableForm[], opción

- TableAlignments · 17
- TableDepth · 17
- TableDirections · 17
- TableHeadings · 17
- TableSpacing · 17

Teorema de Cauchy · 98

Teorema de descomposición para polinomios cromáticos · 438

Teorema de estructura para grupos abelianos finitos · 142

Teorema de Kuratowski · 446

Teorema de Lagrange · 86

Teorema de los cuatro colores · 498

Teorema del número de caminos · 356

traslación · 184

True · 17

V

valor propio · 17

variable · 15

vértice · 202

- adyacentes · 202

- aislado · 314

- colgante · 338

- conectar · 345

final · 202, *Véase* destino
fuente · 314
inicial · 202

origen · *Véase* incial
sumidero · 314

ÍNDICE DE PROGRAMAS Y FUNCIONES

| | |
|---|-----|
| PROGRAMA 1.1. ORDENACIÓN DE LISTAS | 19 |
| FUNCIÓN 1.2. CALENDARIO | 22 |
| PROGRAMA 1.3. SISTEMA ORBITAL SOL/TIERRA/LUNA..... | 24 |
| FUNCIÓN 1.4. BUSCAMINAS I..... | 26 |
| FUNCIÓN 1.5. BUSCAMINAS II..... | 27 |
| FUNCIÓN 1.6. BUSCAMINAS III..... | 27 |
| FUNCIÓN 1.7. BUSCAMINAS IV..... | 28 |
| FUNCIÓN 2.1. OPERACIÓN DE UN GRUPO | 35 |
| FUNCIÓN 2.1. BIS. OPERACIÓN DE UN GRUPO..... | 35 |
| PROGRAMA 2.2. OPERACIÓN INTERNA..... | 36 |
| FUNCIÓN 2.3. OPERACIÓN INTERNA..... | 36 |
| PROGRAMA 2.4. ELEMENTO NEUTRO..... | 38 |
| FUNCIÓN 2.5. ELEMENTO NEUTRO | 38 |
| FUNCIÓN 2.6. ELEMENTO SIMÉTRICO | 40 |
| FUNCIÓN 2.7. CÁLCULO DEL ELEMENTO SIMÉTRICO | 41 |
| FUNCIÓN 2.7.BIS. CÁLCULO DEL ELEMENTO SIMÉTRICO..... | 41 |
| FUNCIÓN 2.8. PROPIEDAD ASOCIATIVA | 43 |
| FUNCIÓN 2.9. PROPIEDAD CONMUTATIVA | 44 |
| PROGRAMA 2.10. HOMOMORFISMOS DE GRUPOS | 47 |
| FUNCIÓN 2.11. HOMOMORFISMOS DE GRUPOS | 48 |
| PROGRAMA 2.12. GRUPOS DE ORDEN PEQUEÑO..... | 63 |
| FUNCIÓN 2.13. GRUPOS ISOMORFOS..... | 64 |
| FUNCIÓN 2.14. GRUPOS DE ORDEN PEQUEÑO..... | 72 |
| FUNCIÓN 3.1. SUBGRUPOS | 84 |
| FUNCIÓN 3.1.BIS. SUBGRUPOS | 84 |
| FUNCIÓN 3.2. CLASES LATERALES | 86 |
| FUNCIÓN 3.3. SUBGRUPOS NORMALES | 88 |
| PROGRAMA 3.4. GRUPO COCIENTE | 89 |
| PROGRAMA 3.5. GRUPO COCIENTE | 89 |
| FUNCIÓN 3.6. DIVISORES..... | 92 |
| FUNCIÓN 3.7. SUBGRUPOS DE N ELEMENTOS | 93 |
| PROGRAMA 3.8. CÁLCULOS DE TODOS LOS SUBGRUPOS | 93 |
| PROGRAMA 3.9. ORDEN DE UN ELEMENTO | 97 |
| FUNCIÓN 3.10. SUBGRUPO GENERADO POR UN SUBCONJUNTO..... | 100 |
| FUNCIÓN 3.11. SUBGRUPOS DE N ELEMENTOS | 104 |

| | |
|--|-----|
| FUNCIÓN 3.12. SUBCONJUNTOS CERRADOS. | 106 |
| FUNCIÓN 3.13. SUBGRUPOS DE N ELEMENTOS. | 107 |
| FUNCIÓN 3.14. SUBGRUPOS DE N ELEMENTOS. | 111 |
| FUNCIÓN 3.15. SUBGRUPOS DE N ELEMENTOS. | 125 |
| PROCEDIMIENTO 3.16. SUBGRUPOS. | 129 |
| PROCEDIMIENTO 3.17. SUBGRUPOS. | 132 |
| PROGRAMA 3.18. GRUPOS DE ORDEN PEQUEÑO. | 137 |
| PROGRAMA 3.19. SUBCONJUNTOS DE N ELEMENTOS. | 145 |
| FUNCIÓN 4.1. PERMUTACIONES I. | 150 |
| FUNCIÓN 4.2. PERMUTACIONES II. | 152 |
| FUNCIÓN 4.3. PERMUTACIONES III. | 155 |
| FUNCIÓN 4.1.BIS. PERMUTACIONES. | 158 |
| FUNCIÓN 4.2.BIS. PERMUTACIONES II. | 159 |
| FUNCIÓN 4.4. EL GRUPO SIMÉTRICO S_N . | 161 |
| FUNCIÓN 4.5. ÍNDICES DE LAS PERMUTACIONES. | 161 |
| FUNCIÓN 4.6. ÍNDICE DE UNA PERMUTACIÓN. | 162 |
| FUNCIÓN 4.6.BIS. ÍNDICE DE UNA PERMUTACIÓN. | 162 |
| FUNCIÓN 4.7. PERMUTACIÓN σ_k . | 163 |
| FUNCIÓN 4.8. PRODUCTO DE PERMUTACIONES. | 164 |
| FUNCIÓN 4.9. PRODUCTO DE PERMUTACIONES 2. | 164 |
| FUNCIÓN 4.10. TABLA DE PRODUCTOS ENTRE PERMUTACIONES. | 165 |
| FUNCIÓN 4.11. TABLA DE PRODUCTOS ENTRE PERMUTACIONES. | 166 |
| PROGRAMA 4.12. NOTACIÓN CÍCLICA. | 170 |
| FUNCIÓN 4.13. DESCOMPOSICIÓN DE UN CICLO EN PRODUCTO DE TRASPOSICIONES. | 172 |
| PROGRAMA 4.14. DESCOMPOSICIÓN DE PERMUTACIONES EN PRODUCTO DE TRASPOSICIONES. | 173 |
| PROGRAMA 4.15. INVERSIONES. | 175 |
| PROGRAMA 4.16. CÁLCULO DE LA SIGNATURA O PARIDAD DE UNA PERMUTACIÓN. | 176 |
| FUNCIÓN 4.17. CÁLCULO DEL SUBGRUPO ALTERNADO POR DESCOMPOSICIÓN EN PRODUCTO DE CICLOS. | 178 |
| PROGRAMA 4.18. CÁLCULO DEL SUBGRUPO ALTERNADO DIRECTAMENTE CONTANDO EL NÚMERO DE INVERSIONES. | 178 |
| PROGRAMA 4.18.BIS. CÁLCULO DEL SUBGRUPO ALTERNADO DIRECTAMENTE CONTANDO EL NÚMERO DE INVERSIONES. | 181 |
| FUNCIÓN 4.19. EL GRUPO DIÉDRICO. | 185 |
| FUNCIÓN 5.1. MATRIZ DE ADYACENCIA (NO DIRIGIDOS). | 209 |
| FUNCIÓN 5.2. MATRIZ DE ADYACENCIA DE UN GRAFO DIRIGIDO. | 210 |
| FUNCIÓN 5.1.BIS. MATRIZ DE ADYACENCIA (NO DIRIGIDOS). | 210 |
| FUNCIÓN 5.2.BIS. MATRIZ DE ADYACENCIA DE UN GRAFO DIRIGIDO. | 210 |
| PROGRAMA 5.3. RECUPERAMOS UN GRAFO NO ORIENTADO DESDE LA MATRIZ DE ADYACENCIA. | 211 |
| PROGRAMA 5.4. RECUPERAMOS UN GRAFO DIRIGIDO DESDE LA MATRIZ DE ADYACENCIA. | 211 |
| FUNCIÓN 5.5. MATRIZ DE INCIDENCIA (NO DIRIGIDOS). | 220 |
| PROGRAMA 5.6. . RECUPERAMOS UN GRAFO DESDE LA MATRIZ DE INCIDENCIA. | 220 |
| FUNCIÓN 5.7. ISOMORFISMOS DE GRAFOS NO ORIENTADOS. | 236 |
| FUNCIÓN 5.8. ISOMORFISMOS DE GRAFOS DIRIGIDOS. | 236 |
| FUNCIÓN 5.9. GRAFOS ISOMORFOS A UNO DADO. | 238 |
| PROGRAMA 5.10. TEST DE ISOMORFÍA. | 239 |

| | |
|---|-----|
| PROGRAMA 5.11. TEST DE ISOMORFÍA II | 254 |
| PROGRAMA 5.12. GRAFOS DE N VÉRTICES | 259 |
| PROGRAMA 5.13. GRAFOS DE N VÉRTICES Y M LADOS | 264 |
| FUNCIÓN 5.14. AÑADIR UN LADO A UN GRAFO | 266 |
| PROGRAMA 5.15. GRAFOS QUE CONTIENEN UNO DADO CON UN NÚMERO MÁXIMO DE LADOS | 269 |
| | |
| FUNCIÓN 5.16. GRAFOS ISOMORFOS..... | 276 |
| FUNCIÓN 5.17. GRAFOS ISOMORFOS..... | 277 |
| FUNCIÓN 5.18. COMBINATORIA EN GRAFOS..... | 278 |
| FUNCIÓN 5.19. GRAFOS ISOMORFOS..... | 284 |
| FUNCIÓN 5.20. AÑADIR UN LADO A UN GRAFO | 287 |
| FUNCIÓN 5.21. GRUPO DE AUTOMORFISMOS DE UN GRAFO NO ORIENTADO | 290 |
| FUNCIÓN 5.22. GRUPO DE AUTOMORFISMOS DE UN DIGRAFO | 291 |
| FUNCIÓN 6.1. GRADO DE UN VÉRTICE DE UN GRAFO NO DIRIGIDO..... | 311 |
| FUNCIÓN 6.2. GRADO DE UN VÉRTICE DE UN GRAFO NO DIRIGIDO..... | 312 |
| FUNCIÓN 6.3. GRADO DE UN VÉRTICE DE UN GRAFO NO DIRIGIDO..... | 312 |
| FUNCIÓN 6.4. GRADO DE ENTRADA DE UN VÉRTICE DE UN GRAFO DIRIGIDO..... | 314 |
| FUNCIÓN 6.5. GRADO DE SALIDA DE UN VÉRTICE DE UN GRAFO DIRIGIDO | 314 |
| FUNCIÓN 6.6. GRAFOS REGULARES | 317 |
| FUNCIÓN 6.6.BIS. GRAFOS REGULARES | 317 |
| FUNCIÓN 6.7. GRAFOS REGULARES | 318 |
| FUNCIÓN 6.8. GRAFOS COMPLETOS | 321 |
| FUNCIÓN 6.9. SUBGRAFOS INDUCIDOS | 327 |
| FUNCIÓN 6.10. SUBGRAFOS INDUCIDOS | 327 |
| FUNCIÓN 6.11. GRAFOS BIPARTITOS Y BIPARTITOS COMPLETOS | 330 |
| FUNCIÓN 6.12. GRAFOS BIPARTITOS COMPLETOS | 331 |
| FUNCIÓN 6.13. GRAFOS BIPARTITOS COMPLETOS | 334 |
| FUNCIÓN 6.14. VÉRTICES AISLADOS O COLGANTES | 338 |
| FUNCIÓN 7.1. CAMINOS EN GRAFOS NO DIRIGIDOS | 343 |
| FUNCIÓN 7.2. CAMINOS EN GRAFOS NO DIRIGIDOS | 343 |
| FUNCIÓN 7.3. CAMINOS EN GRAFOS NO DIRIGIDOS | 344 |
| FUNCIÓN 7.4. CAMINOS EN GRAFOS NO DIRIGIDOS | 344 |
| FUNCIÓN 7.5. CAMINOS ORIENTADOS EN GRAFOS DIRIGIDOS | 346 |
| FUNCIÓN 7.6. CAMINO ORIENTADO EN GRAFOS DIRIGIDOS | 347 |
| FUNCIÓN 7.7. CAMINOS ORIENTADOS EN GRAFOS DIRIGIDOS | 347 |
| PROGRAMA 7.8. CAMINOS SIMPLES EN GRAFOS NO DIRIGIDOS | 351 |
| PROGRAMA 7.9. CAMINOS SIMPLES EN GRAFOS NO DIRIGIDOS | 351 |
| PROGRAMA 7.10. CAMINOS ELEMENTALE EN GRAFOS NO DIRIGIDOS | 351 |
| PROGRAMA 7.11. SUCESIÓN DE LADOS DE UN CAMINO NO ORIENTADO EN UN GRAFO DIRIGIDO | 352 |
| | |
| FUNCIÓN 7.12. CAMINOS DE LONGITUD L | 360 |
| FUNCIÓN 7.13. GRAFOS CONEXOS | 363 |
| FUNCIÓN 7.14. COMPONENTES CONEXAS | 365 |
| FUNCIÓN 7.15. COMPONENTES FUERTEMENTE CONEXAS | 368 |
| FUNCIÓN 7.16. NÚMERO DE GEODÉSICAS Y DISTANCIA | 371 |
| FUNCIÓN 7.17. GEODÉSICAS | 372 |
| FUNCIÓN 7.18. GRAFO DE EULER NO DIRIGIDO | 374 |
| FUNCIÓN 7.19. GRAFO DE EULER DIRIGIDO | 375 |

| | |
|---|-----|
| PROGRAMA 7.20. CICLO DE EULER | 379 |
| FUNCIÓN 7.21. GRAFO DE HAMILTON NO DIRIGIDO | 383 |
| FUNCIÓN 7.22. CAMINOS* EN GRAFOS NO ORIENTADOS | 405 |
| FUNCIÓN 8.1. LAS N -COLORACIONES DE UN GRAFO | 415 |
| FUNCIÓN 8.2. NÚMERO CROMÁTICO DE UN GRAFO | 415 |
| FUNCIÓN 8.3. COLORACIÓN ÓPTIMA DE UN GRAFO | 416 |
| FUNCIÓN 8.4. COLORACIÓN DE UN GRAFO | 422 |
| FUNCIÓN 8.5. COLORACIÓN DE UN GRAFO | 426 |
| FUNCIÓN 8.6. COLORACIÓN DE UN GRAFO | 430 |
| FUNCIÓN 8.7. POLINOMIO CROMÁTICO | 436 |
| FUNCIÓN 8.8. POLINOMIO CROMÁTICO | 437 |
| FUNCIÓN 8.9. ELIMINAR UN LADO | 439 |
| FUNCIÓN 8.10. IDENTIFICACIÓN DE DOS VÉRTICES | 439 |
| FUNCIÓN 8.11. AÑADIR UN LADO | 443 |
| FUNCIÓN 8.12. GRAFOS PLANOS: ELIMINAR SUBDIVISIONES Y VÉRTICES DE GRADO 1 O 0 | 447 |
| FUNCIÓN 8.13. GRAFOS PLANOS: TEST PARA $K_{3,3}$ | 448 |
| FUNCIÓN 8.14. GRAFOS HOMEOMORFOS A K_5 O $K_{3,3}$ | 448 |
| FUNCIÓN 8.15. GRAFOS PLANOS | 449 |
| FUNCIÓN 8.16. CONDICIÓN NECESARIA PARA GRAFOS PLANOS | 457 |
| FUNCIÓN 8.17. COLORACIÓN DE REGIONES, PASO 1 | 470 |
| FUNCIÓN 8.18. COLORACIÓN DE REGIONES: GRAFO DE LAS REGIONES | 470 |
| FUNCIÓN 8.19. CICLOS ELEMENTALES DE LONGITUD L | 473 |
| FUNCIÓN 8.20. ÁRBOLES Y BOSQUES | 475 |
| FUNCIÓN 8.21. ÁRBOLES | 479 |
| FUNCIÓN 8.22. BOSQUES | 480 |
| FUNCIÓN 8.23. REGIÓN ILIMITADA DE UN GRAFO PLANO CONEXO | 499 |

ÍNDICE DE TABLAS E ILUSTRACIONES

| | |
|--|-----|
| ILUSTRACIÓN 1.1. ESCHEMA DE UN PROGRAMA, FUNCIÓN O PROCEDIMIENTO. | 14 |
| TABLA 1.1. ÓRBITAS PLANETARIAS. | 30 |
| ILUSTRACIÓN 1.2. | 30 |
| TABLA 2.1. TABLA DE OPERACIONES DE UN GRUPO. | 34 |
| TABLA 2.2. | 34 |
| TABLA 2.3. | 48 |
| TABLA 2.4. | 75 |
| TABLA 2.5. | 75 |
| TABLA 3.1. EFECTIVIDAD EN EL CÁLCULO DE SUBGRUPOS IMPROPIOS. | 112 |
| TABLA 3.2. SUBGRUPOS. | 113 |
| TABLA 3.3. | 113 |
| ILUSTRACIÓN 3.1. | 116 |
| TABLA 3.4. EFECTIVIDAD. | 125 |
| TABLA 3.4. SUBGRUPOS DE S_5 . | 131 |
| TABLA 3.5. GRUPOS DE ORDEN PEQUEÑO DISTINTOS SALVO ISOMORFISMO. | 142 |
| TABLA 3.5. | 143 |
| TABLA 4.1. CÁLCULO DE LAS PERMUTACIONES I. | 151 |
| TABLA 4.2. CÁLCULO DE LAS PERMUTACIONES II. | 153 |
| TABLA 4.3. ELEMENTOS DE D_4 . | 184 |
| ILUSTRACIÓN 4.1. MÁQUINA ENIGMA. | 190 |
| ILUSTRACIÓN 4.2. MÁQUINA ENIGMA: ESCHEMA DEL ROTOR. | 191 |
| ILUSTRACIÓN 4.3. MÁQUINA ENIGMA: ESCHEMA DE LA COMPOSICIÓN DE PERMUTACIONES. | 192 |
| ILUSTRACIÓN 4.4. | 193 |
| ILUSTRACIÓN 4.5. | 193 |
| ILUSTRACIÓN 4.6. | 193 |
| TABLA 4.4. CIFRADO DE UN MENSAJE CON ENIGMA. | 195 |
| TABLA 4.5. DESCIFRADO DE UN MENSAJE CON ENIGMA. | 196 |
| ILUSTRACIÓN 5.1. | 202 |
| ILUSTRACIÓN 5.2. | 202 |
| ILUSTRACIÓN 5.3. | 206 |
| ILUSTRACIÓN 5.4. | 207 |
| ILUSTRACIÓN 5.5. | 235 |
| ILUSTRACIÓN 5.6. | 241 |

| | |
|-----------------------------|-----|
| ILUSTRACIÓN 5.7. | 241 |
| ILUSTRACIÓN 5.8. | 241 |
| ILUSTRACIÓN 5.9. | 246 |
| ILUSTRACIÓN 5.10. | 246 |
| ILUSTRACIÓN 5.11. | 246 |
| ILUSTRACIÓN 5.12. | 249 |
| ILUSTRACIÓN 5.13. | 249 |
| ILUSTRACIÓN 5.14. | 250 |
| ILUSTRACIÓN 5.15. | 252 |
| ILUSTRACIÓN 5.16. | 252 |
| ILUSTRACIÓN 5.17. | 255 |
| ILUSTRACIÓN 5.18. | 255 |
| TABLA 5.1. | 260 |
| ILUSTRACIÓN 5.19. | 292 |
| ILUSTRACIÓN 5.20. | 302 |
| ILUSTRACIÓN 5.21. | 302 |
| ILUSTRACIÓN 5.22. | 303 |
| ILUSTRACIÓN 5.23. | 303 |
| ILUSTRACIÓN 5.24. | 303 |
| ILUSTRACIÓN 5.25. | 305 |
| ILUSTRACIÓN 5.26. | 306 |
| ILUSTRACIÓN 5.27. | 307 |
| ILUSTRACIÓN 6.1. | 339 |
| ILUSTRACIÓN 7.1. | 375 |
| ILUSTRACIÓN 7.2. | 375 |
| ILUSTRACIÓN 7.3. | 375 |
| ILUSTRACIÓN 7.4. | 375 |
| ILUSTRACIÓN 7.5. | 375 |
| ILUSTRACIÓN 7.6. | 375 |
| ILUSTRACIÓN 7.7. DODECAEDRO | 388 |
| ILUSTRACIÓN 7.8. | 393 |
| ILUSTRACIÓN 7.9. | 393 |
| ILUSTRACIÓN 7.10. | 394 |
| ILUSTRACIÓN 7.11. | 395 |
| ILUSTRACIÓN 7.12. | 401 |
| ILUSTRACIÓN 7.13. | 402 |
| ILUSTRACIÓN 7.14. | 402 |
| ILUSTRACIÓN 7.15. | 403 |
| ILUSTRACIÓN 7.16. | 406 |
| ILUSTRACIÓN 7.17. | 407 |
| ILUSTRACIÓN 7.18. | 407 |
| ILUSTRACIÓN 7.19. | 408 |
| ILUSTRACIÓN 7.20. | 409 |
| ILUSTRACIÓN 7.21. | 411 |
| ILUSTRACIÓN 8.1. | 432 |
| ILUSTRACIÓN 8.2. | 435 |
| ILUSTRACIÓN 8.3. | 441 |
| ILUSTRACIÓN 8.4. | 443 |

| | |
|---|-----|
| ILUSTRACIÓN 8.5. GRAFO DE PETERSEN..... | 458 |
| ILUSTRACIÓN 8.6..... | 460 |
| ILUSTRACIÓN 8.7..... | 464 |
| ILUSTRACIÓN 8.8..... | 467 |
| ILUSTRACIÓN 8.9..... | 467 |
| ILUSTRACIÓN 8.10..... | 468 |
| ILUSTRACIÓN 8.11..... | 469 |
| ILUSTRACIÓN 8.12..... | 471 |
| TABLA 8.1..... | 474 |
| ILUSTRACIÓN 8.13..... | 487 |
| ILUSTRACIÓN 8.14..... | 491 |
| ILUSTRACIÓN 8.15..... | 491 |
| ILUSTRACIÓN 8.16..... | 491 |
| ILUSTRACIÓN 8.19..... | 493 |
| ILUSTRACIÓN 8.18..... | 493 |
| ILUSTRACIÓN 8.17..... | 493 |
| TABLA 8.2..... | 494 |
| TABLA 8.3..... | 495 |
| TABLA 8.4..... | 495 |
| ILUSTRACIÓN 8.20..... | 496 |
| ILUSTRACIÓN 8.21..... | 496 |
| ILUSTRACIÓN 8.22..... | 496 |
| ILUSTRACIÓN 8.23..... | 496 |
| ILUSTRACIÓN 8.24..... | 498 |
| ILUSTRACIÓN 8.25..... | 498 |
| ILUSTRACIÓN 8.26..... | 498 |
| ILUSTRACIÓN 8.27..... | 501 |
| ILUSTRACIÓN 8.28..... | 501 |
| ILUSTRACIÓN 8.29..... | 501 |
| ILUSTRACIÓN 8.30..... | 502 |
| ILUSTRACIÓN 8.31..... | 503 |
| ILUSTRACIÓN 8.32..... | 503 |
| ILUSTRACIÓN 8.33..... | 504 |

BIBLIOGRAFÍA

- [1] **Abellanas, M. y Lodares, D.** “*Análisis de algoritmos y teoría de grafos*”. Ed. RA-MA. Madrid, 1990. ISBN: 84-7897-000-2.
- [2] **Abellanas, M. y Lodares, D.** “*Matemática Discreta*”. Ed. RA-MA. Madrid, 1990. ISBN: 84-86381-99-1.
- [3] **Alegre Gil, C.** “*Problemas de Matemática Discreta*” Ed. Universidad Politécnica de Valencia, 1997. ISBN 84-7721-495-6.
- [4] **Anzola, M. y otros.** “*Problemas de Álgebra: Anillos. Polinomios. Ecuaciones.*” (tomo 2). Ed. Autores, 1981/82. ISBN 84-300-6417-6.
- [5] **Anzola, M. y otros.** “*Problemas de Álgebra: Conjuntos. Grupos.*” (tomo 1). Ed. Autores, 1981/82. ISBN 84-300-4073-0.
- [6] **Biggs, N.L.** “*Matemática Discreta*”. Ed. Vicens Vives, 1998. ISBN: 84-316-3311-5.
- [7] **Blachman, N.** “*Mathematica. Un Enfoque Práctico*”. Ariel Informática, 1993. ISBN 84-344-0478-8.
- [8] **Blachman, N.** “*Mathematica*”. Ed. Addison-Wesley, 1992. ISBN 0-201-62880-5.
- [9] **Bujalance, E. y otros.** “*Elementos de Matemática Discreta*”. Sanz Torres, 2001. ISBN: 84-88667-35-3.
- [10] **Bujalance, E. y otros.** “*Problemas de Matemática Discreta*”. Sanz Torres, 2000. ISBN: 84-88667-03-5.
- [11] **Chartrand, G. y Oellermann, O. R.** “*Applied and Algorithmic Graph Theory*”. McGraw-Hill, 1993. ISBN: 0-07-557101-3.
- [12] **Cohn, P.M.** “*Álgebra*”. Volumen I, J. Wiley & Sons, 1974. ISBN: 0471164305.
- [13] **Diestel, R.** “*Graph Theory*”. Series: *Graduate Text in Mathematics, Vol. 173*. Springer. 2006. ISBN: 978-3-540-26183-4.
- [14] **Domínguez Pérez, J.A. y otros.** “*Álgebra Lineal. Planteamiento y Resolución de Problemas con Mathematica*”. Ed. Plaza Universitaria, Salamanca, 1995. ISBN: 84-89109-06-0.
- [15] **Dorronsoro, J. y Hernández, E.** “*Números, Grupos y Anillos*”. Addison Wesley. Universidad Autónoma de Madrid, 1996. ISBN: 0-201-65359-8.
- [16] **Dubreil, P. y otros.** “*Lecciones de Álgebra Moderna*”. Ed. Reverté, 1971. ISBN: 84-291-5070-6.
- [17] **Fernández – Ferreiros, A. y otros.** “*Álgebra Lineal. Prácticas con Mathematica*”. Ed. Prensas Universitarias de Zaragoza, Zaragoza, 1995. ISBN: 84-773-3452-8.
- [18] **Foulds, L.R.** “*Graph Theory Applications*”. Springer-Verlag, 1992. ISBN 0-387-97599-3.
- [19] **García Merayo, F.** “*Matemática Discreta 2ª Edición*”. Ed. Thomson, 2005. ISBN: 84-9732-367-X.
- [20] **García Merayo, F. y Rodríguez Gómez, F.J.** “*Fundamentos y Aplicaciones de*

- Mathematica* ". Ed. Paraninfo, 1998. ISBN: 84-283-2485-9.
- [21] **García Merayo, F., Hernández, G. y Nevot, A.** "Problemas Resueltos de Matemática Discreta". Ed. Thomson, 2003. ISBN: 84-973-2210-X.
- [22] **García Valle, J.L.** "Matemáticas Especiales para Computación ". Serie Informática de Gestión. Ed. McGraw Hill, 1991. ISBN: 84-7615-267-1.
- [23] **García, C., López, J. M. y Puigjaner, D.** "Matemática Discreta: Problemas y Ejercicios Resueltos ". Prentice Hall D. L., 2002. ISBN: 84-205-3439-0.
- [24] **García-Muñoz, M.A., Ordóñez, C. y Ruiz, J.F.** "Divisibilidad de los Números Enteros: El Secreto de los Números Primos ". I Jornadas de Innovación y Mejora Docente en la Universidad de Jaén. Jaén, 2005.
- [25] **García-Muñoz, M.A., Ordóñez, C. y Ruiz, J.F.** "Métodos computacionales en Álgebra para informáticos. Matemática Discreta y lógica". Ed. Servicio de publicaciones de la UJA, 2006. ISBN: 84-8439-316-X.
- [26] **García-Muñoz, M.A., Ordóñez, C. y Ruiz, J.F.** "Orden y Estructuras Algebraicas Mediante Nuevas Tecnologías ". I Jornadas de Innovación y Mejora Docente en la Universidad de Jaén. Jaén, 2005.
- [27] **García-Muñoz, M.A., Ordóñez, C. y Ruiz, J.F.** "Teoría de Números con Mathematica ". VIII Jornadas Andaluzas de Educación Matemática "Thales". Jaén, 1998. Pág. 163-168. ISBN: 84-89869-37-5.
- [28] **Grassmann, W.K. y Tremblay, J.P.** "Matemática Discreta y Lógica ". Prentice Hall D. L., 1996. ISBN: 84-89660-04-2.
- [29] **Grimaldi, R.P.** "Matemáticas Discreta y Combinatoria ". Addison Wesley Iberoamericana, 1989. ISBN: 0-201-64406-1.
- [30] **Holt, D. F., Eick, B. y O'Brien, E. A.** "Handbook of Computational Group Theory (Discrete Mathematics and Its Applications)". Chapman & Hall/CRC, 2005. ISBN 1584883723.
- [31] **Johnsonbaugh, R.** "Matemáticas Discretas ". Ed. Pearson Prentice Hall, 1999. ISBN: 9701702530.
- [32] **Joyce, J. y Moon, M.** "Microsoft® Windows® XP Plain & Simple ". Microsoft Press, 2001. ISBN 0-7356-1525-X.
- [33] **Knuth, T.E.** "Algoritmos Fundamentales ". El Arte de Programar Ordenadores Vol. I. Ed. Reverté, 1980. ISBN: 84-291-2662-7.
- [34] **Kolman, B., Bugsby, R.C. y Ross, S.** "Estructuras de Matemáticas Discretas para la Computación ". 3^a Edición. Ed. Pearson Prentice Hall. 1997. ISBN: 0-13-320912-1.
- [35] **Lipschutz, S.** "Teoría y Problemas de Matemática Discreta ". Ed. McGraw-Hill, 1990. ISBN: 84-7615-450-X.
- [36] **Liu, C. L.** "Introduction to Combinatorial Mathematics ". New York, McGraw-Hill, 1968.
- [37] **McKay, B. D.** "Practical Graph Isomorphism", Congressus Numerantium, Vol 30 (1981), 45-87.
- [38] **McKay, B. D.** "NAUTY". 2007. (<http://cs.anu.edu.au/~bdm/nauty/>)
- [39] **Merino L. y Santos E.** "Álgebra Lineal con Métodos Elementales ". Ed. Thomson, 2006. ISBN 84-8498-461-3.
- [40] **OTSI.** "Microsoft® Windows® XP Step by Step ". Microsoft Press, 2001. ISBN 0-7356-1383-4.
- [41] **Paulsen, W.** "Group Presentations Using Mathematica ". Mathematica in Education and Research, 1995, volumen 4, issue 4, 21-24.

- [42] **Pemmaraju, S. y Skiena S.** “*Computational Discrete Mathematics. Combinatorics and Graph Theory with Mathematica*”. Cambridge University Press. ISBN: 0-521-80686-0.
- [43] **Ramírez González, V. y otros.** “*Matemáticas con Mathematica*”. Granada: Proyecto Sur de Ediciones, 1996. ISBN: 84-8254-084-X.
- [44] **Rosen H. K.** “*Discrete Mathematics and Its Applications*”. McGraw-Hill, 2003. ISBN: 0-07-242434-6.
- [45] **Sigler, L.G.** “*Álgebra*” Ed. Reverté. ISBN: 84-291-5129-X.
- [46] **The GAP Group.** “*GAP -- Groups, Algorithms, and Programming, Version 4.4.9*”. 2006. (<http://www.gap-system.org>)
- [47] **Vera López, A. y otros.** “*Álgebra Abstracta Aplicada*”. Ed. Vera López. Murcia 1992. ISBN 84-604-3850-3.
- [48] **Vera López, A. y otros.** “*Problemas y Ejercicios de Matemática Discreta*”. Ed. Vera López. Bilbao 1995. ISBN: 84-605-4351-X.
- [49] **Wolfram, S.** “*Mathematica Reference Guide*”. Addison-Wesley, 1992. ISBN 0-201-51012-X.
- [50] **Wolfram, S.** “*Mathematica. A System for Doing Mathematics by Computer*”. Addison-Wesley, 1991.. ISBN: 0-201-51507-5.
- [51] **Wolfram, S.** “*Mathematica: The Student Book*”. Addison-Wesley, 1994. ISBN 0-201-55479-8.
- [52] **Wolfram, S.** “*The Mathematica Book, Fifth Edition*”. Wolfram Media, 2003. ISBN 1-57955-022-3.

