
Matemáticas discretas

Favio E. Miranda
Elisa Viso G.
Facultad de Ciencias, UNAM

Índice general

I	Lógica Matemática	1
1.	Introducción	3
1.1.	Expresiones	4
1.2.	Mecanismos formales para descripción de expresiones	6
1.3.	Gramáticas y árboles de derivación	10
2.	Lógica proposicional	19
2.1.	El lenguaje de la lógica proposicional	19
2.1.1.	Argumentos lógicos	19
2.1.2.	Proposiciones	21
2.1.3.	Sintaxis de la lógica proposicional	24
2.1.4.	Semántica de la lógica proposicional	26
2.1.5.	Tautologías y contradicciones	35
2.1.6.	Argumentos correctos	37
2.2.	Evaluación de expresiones	42
2.2.1.	Estados y evaluación	42
2.2.2.	Precedencia y asociatividad	44
2.2.3.	Sustitución textual	46
2.3.	Análisis sintáctico de expresiones lógicas	50
2.3.1.	Esquemas	51
2.3.2.	Rango y conectivo principal	53
2.3.3.	Análisis de proposiciones compuestas	55
2.3.4.	Tautologías y sustitución	57
2.4.	Equivalencia lógica	61
2.4.1.	Razonamiento ecuacional	63
2.4.2.	Álgebra de equivalencias lógicas	68
2.5.	Conceptos semánticos importantes	74
2.5.1.	Interpretaciones	75
2.5.2.	Consecuencia lógica	77
2.6.	Análisis de argumentos	80
2.6.1.	Tablas de Verdad	80

2.6.2.	Uso de interpretaciones	82
2.6.3.	Derivaciones	88
2.7.	Tableaux en cálculo proposicional	98
2.7.1.	El concepto de tableau	98
2.7.2.	Eliminación de ramas del tableau	103
2.7.3.	Reglas para los tableaux	105
2.7.4.	Modelo de una fórmula	107
2.7.5.	Algoritmos para la lógica proposicional	108
3.	Lógica de predicados	113
3.1.	Introducción	113
3.1.1.	Predicados	114
3.1.2.	Variables y cuantificadores	116
3.2.	Sintaxis de la lógica de predicados	118
3.2.1.	Términos	118
3.2.2.	Fórmulas	119
3.2.3.	Fórmulas cuantificadas	121
3.2.4.	Variables libres y ligadas	123
3.3.	Especificación formal	128
3.3.1.	Juicios aristotélicos	130
3.3.2.	Negaciones	131
3.3.3.	Contando objetos	132
3.3.4.	Micromundos	133
3.4.	Semántica informal	139
3.4.1.	Dominios de interpretación	139
3.4.2.	Noción informal de verdad	142
3.4.3.	Verdad en micromundos	144
3.4.4.	Algunas equivalencias lógicas	147
3.4.5.	Algunos argumentos correctos	155
3.5.	Predicados y tipos	156
II	Inducción y recursión	161
4.	Inducción y recursión	163
4.1.	Introducción	163
4.2.	Los números naturales	164
4.2.1.	Axiomas de Peano	165
4.3.	Inducción en los números naturales	166
4.3.1.	Cambio de la base de la inducción	170
4.3.2.	Inducción completa	172
4.4.	Definiciones recursivas	178

4.4.1.	Definición de funciones recursivas	181
4.5.	Inducción estructural	186
4.5.1.	Inducción en listas	187
4.5.2.	Inducción en fórmulas	190
4.5.3.	Inducción en árboles	192

III Teoría de Gráficas 199

5. Conceptos de teoría de gráficas 201

5.1.	Motivación	201
5.1.1.	Tiempo para completar un proyecto	203
5.1.2.	Asignación óptima de recursos	210
5.2.	Conceptos y formalización	215
5.3.	Representación de gráficas para su manipulación	228
5.3.1.	Matriz de adyacencias	228
5.3.2.	Matriz de incidencias	230
5.3.3.	Listas de adyacencias	231
5.3.4.	Listas de incidencias	233
5.4.	Isomorfismo entre gráficas	236

6. Exploración en gráficas 243

6.1.	Circuitos eulerianos	243
6.2.	Trayectorias hamiltonianas	261
6.3.	Distancias en una gráfica	269
6.4.	Trayectorias más cortas	275
6.5.	Número de caminos	284
6.5.1.	Matrices de adyacencias	284
6.5.2.	Colofón	291

7. Modelado con gráficas 295

7.1.	Coloración	295
------	----------------------	-----

8. Árboles 311

8.1.	Caracterización	311
8.2.	Árboles generadores	318
8.3.	Búsqueda en profundidad (DFS)	322
8.4.	Árboles generadores de peso mínimo	328
8.4.1.	Algoritmo de Prim para árboles de peso mínimo	329
8.4.2.	Algoritmo de Kruskal para árboles de peso mínimo	346

9. Multigráficas y gráficas dirigidas	353
9.1. Multigráficas	353
9.2. Gráficas dirigidas	356
9.3. Circuitos eulerianos	361
9.4. Distancias en una gráfica dirigida	366
9.4.1. BFS	366
9.4.2. Algoritmo de Dijkstra para trayectorias dirigidas más cortas	367
9.4.3. Número de caminos	368
9.4.4. Árboles	369
Índice	375

Parte I

Lógica Matemática

Introducción | 1

El libro está dividido fundamentalmente en tres partes: *Lógica Matemática*, *Inducción y Recursión*, y *Teoría de Gráficas*. De la inducción y recursión tal vez no hemos oído pero de lógica y gráficas sí, cuando por ejemplo hemos hecho gráficas desde la secundaria (aunque esta interpretación de “gráficas” no es la que vamos a atacar en este curso) y el término *lógica* lo usamos de manera bastante liberal en nuestra vida diaria en frases como las que siguen:

- No es lógico lo que estás diciendo.
- No entiendo la lógica de este asunto.
- Presentas un argumento que no es coherente.
- Es falso lo que estás suponiendo.

Todos nosotros sabemos que existe más precisión cuando estamos en el terreno matemático que cuando estamos hablando de experiencias de la vida común. Realmente, en el lenguaje *natural*¹ dejamos mucho a la subjetividad de los hablantes y al contexto que se supone conocen ambos. Decimos que este tipo de lenguaje es *informal* mientras que el lenguaje que se usa en matemáticas o los lenguajes de programación son *lenguajes formales*.

Distinguimos entre un objeto informal y uno formal porque este último está claramente definido y especificado por un conjunto de reglas. Uno de los atractivos del formalismo es el poder expresar ideas de forma concreta, breve y precisa. Pero no nada más nos interesan estos aspectos del formalismo sino su aplicación, la cual nos obliga a formalizar nuevas

¹Llamaremos así al lenguaje que habla cualquier ser humano, en nuestro caso el español.

ideas o experiencias y, en este proceso, precisar y encontrar contradicciones que pudieran causar mucho daño, como en el caso de un programa de computadora que pudiese contener ambigüedades.

Si nos referimos a un objeto de manera formal, podemos construir un *modelo* de ese objeto. Algunas de las ventajas de los modelos matemáticos son las siguientes:

- Un modelo matemático es, por lo general, más preciso, entendible, conciso y riguroso que una descripción informal escrita en lenguaje natural.
- A través de un modelo matemático podemos calcular directamente respuestas a problemas sobre el objeto modelado.
- Las matemáticas, y en particular la lógica, nos proporcionan métodos de razonamiento: para manipular objetos, para demostrar propiedades de y sobre objetos, y para obtener resultados nuevos a partir de resultados ya conocidos, lo cual genera una extensión del conocimiento.

Este último punto es, tal vez, uno de los aspectos más importantes de los modelos matemáticos, que tiene una enorme utilidad en ciencias de la computación: tener la seguridad científica de que algo funciona como esperamos; poder extender las posibilidades de una computadora, un lenguaje de programación o un algoritmo para usos distintos que para los que fue creado; en fin, para poder continuar con el impresionante desarrollo que han tenido las ciencias de la computación en este siglo.

1.1. Expresiones

La lógica, y en particular la lógica matemática, juega un papel muy importante en el desarrollo de las matemáticas en general y de las ciencias de la computación en particular. En el ámbito de las ciencias de la computación es importante poder distinguir entre argumentos válidos o inválidos, es decir, entre aquellos que son sólidos lógicamente hablando y los que no lo son.

La lógica presenta ciertos elementos de estudio que no son tan distintos a los que estamos acostumbrados en matemáticas. Durante muchos años hemos manipulado expresiones numéricas con incógnitas, es decir, ecuaciones con variables y constantes, para obtener un resultado final. Mientras que en el álgebra estamos trabajando con números fundamentalmente, en lógica trabajamos con *proposiciones lógicas* o simplemente *proposiciones*. Al igual que en álgebra, utilizamos variables (*símbolos* que nos sirven para representar a los objetos elementales), constantes (true, false) y operadores, que también son símbolos pero con un significado especial.

Cuando tenemos una expresión aritmética hablamos de los operadores y operandos, entendiendo a los operadores como operaciones que se tienen que realizar, utilizando para

ello a los operandos. Los operadores pueden ser *unarios*, cuando utilizan o actúan sobre un único operando; *binarios* cuando actúan sobre dos operandos y, en general, *n-arios*² cuando actúan sobre n operandos. Entre los operadores unarios aritméticos que conocemos está el signo de menos $-$ y en algunas ocasiones también podemos considerar el signo de más $+$ (nos tendremos que poner de acuerdo antes de empezar a cuál aceptamos y a cuál no). Entre los operadores binarios podemos mencionar a la multiplicación (que la podemos representar con \cdot , \times o con $*$ como se acostumbra en los lenguajes de programación), la división (\div , $/$). Un ejemplo de operador ternario puede ser $\text{entre}(a, b, c)$ que decide cuál de los tres números a, b, c se encuentra entre los otros dos o el operador $\text{raíces}(a, b, c)$ que devuelve las raíces del polinomio cuadrático $ax^2 + bx + c$.

Hay tres estilos para escribir expresiones, los cuales se distinguen por cómo se coloca al operador en relación a sus operandos. Si el operador es unario o n-ario únicamente tenemos dos opciones:

Notación prefija: El operador se coloca antes del operando

$$-a \quad (+ \ a \ b \ c)$$

Notación sufija o polaca: El operador se coloca después del operando

$$a \uparrow \quad (a \ b \ *)$$

apuntador en Pascal

Si el operador es binario, además de estas dos formas tenemos la que es más usual:

Notación infija: Es aquella donde el operador se encuentra *entre* sus operandos.

$$a + b \quad 3 \cdot (7 + 5)$$

Las diferencias entre estas tres notaciones no son nada más de forma, pues cada una de ellas tiene propiedades particulares que veremos después. Por lo pronto, trabajaremos con la notación infija que, como ya mencionamos, es la más usual.

Estas maneras de escribir expresiones tienen que ver con su *sintaxis*, término que se refiere a la *forma* que deben tener las cadenas de letras y símbolos para que califiquen como expresiones bien construidas. Aún no hemos hablado del *significado* de una expresión, aunque pronto lo haremos. Los aspectos relacionados con el significado conforman la *semántica* de las expresiones.

²Se lee “enario”.

1.2. Mecanismos formales para descripción de expresiones

Cuando describimos una expresión aritmética podemos hacerlo de varias maneras. Una de ellas es dando tantos ejemplos como podamos y dejando al lector que encuentre patrones que describan al mayor número posible de expresiones. Es claro que con este método no vamos a poder describir a todas las expresiones aritméticas posibles, ya que tenemos un número infinito de ellas. Una segunda manera es dando reglas para *construir* expresiones aritméticas correctas. Estas reglas de formación pertenecen, queremos insistir, a la sintaxis de las expresiones aritméticas.

A continuación formalizamos reglas para construir expresiones aritméticas sencillas, para posteriormente contrastar con lo que es una expresión lógica:

Definición 1.1 Una *expresión aritmética* es alguna de las que siguen:

1. Un *objeto elemental*: un número (una constante) o una variable.
2. Si E es una expresión aritmética, (E) es una expresión aritmética.
3. Si \triangleright es un operador unario y E es una expresión aritmética, entonces $\triangleright E$ es una expresión aritmética.
4. Si \diamond es un operador binario infijo y E y F son dos expresiones aritméticas, entonces $E \diamond F$ es una expresión aritmética.
5. Si \star es un operador n-ario y E_1, E_2, \dots, E_n son expresiones aritméticas, entonces $\star(E_1, E_2, \dots, E_n)$ es una expresión aritmética.
6. Éstas y sólo éstas son expresiones aritméticas válidas.

A primera vista esta definición parece incorrecta pues en algunas partes se utiliza el mismo concepto de expresión que se está definiendo. Esta clase de definiciones, llamadas definiciones recursivas, son omnipresentes en ciencias de la computación. Más adelante las estudiaremos con detalle. Estamos suponiendo que sabemos cuáles son los operadores unarios: $+$ (positivo), $-$ (negativo); cuáles los binarios: $-$ (resta), $+$ (suma), \times , \cdot , $*$ (multiplicación), \div , $**$ (\wedge , exponenciación); y conocemos algunos n-arios: $f(x_1, x_2, \dots)$, $\max(\dots)$, $\min(\dots)$...

Veamos a continuación algunos ejemplos de construcción de expresiones aritméticas enfatizando su proceso de construcción.

Ejemplo 1.1. Cada uno de los siguientes es un objeto elemental:

a i \times 3.0 1

Por lo tanto también son expresiones aritméticas.

Ejemplo 1.2. Considérese los siguientes operadores unarios, que representan el signo de un número en aritmética: $+$ y $-$:

-17 $-$ reemplaza \triangleright y 17 es una expresión, por ser un número.

$+a$ $+$ reemplaza \triangleright y a es una expresión, por ser una variable.

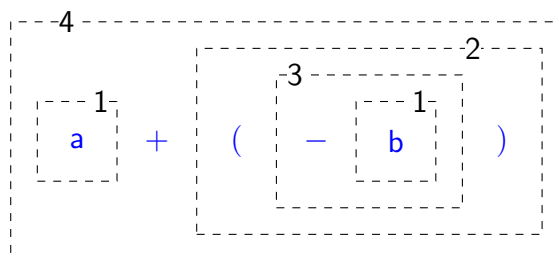
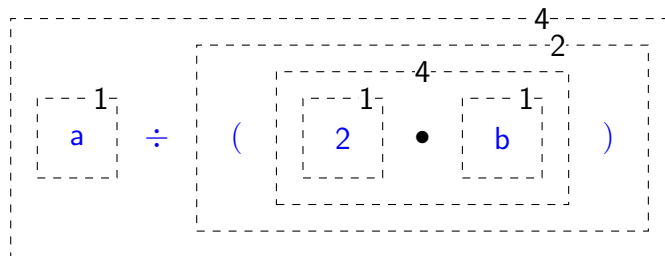
Podemos también ir encerrando en cuadrados contenidos uno dentro del otro, con una anotación de la regla a la que corresponden, a las distintas expresiones que conforman la expresión original. Cada rectángulo encierra a una expresión. Veamos a continuación:



Ejemplo 1.3. Considérese los operadores binarios \cdot y \div .

$a \div (2 \cdot b)$ \div y \cdot son operadores binarios; a y $(2 \cdot b)$ son expresiones.

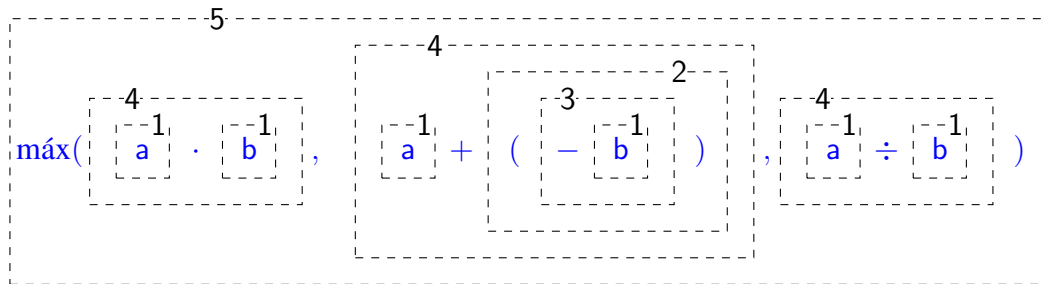
$a + (-b)$ $+$ es un operador binario; a y $(-b)$ son expresiones.



Ejemplo 1.4. Supongamos que tenemos dos operadores, máx y mín .

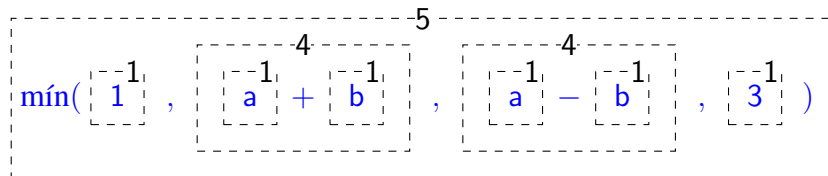
$\text{máx}(a \cdot b, a + (-b), a \div b)$

máx es un operador n -ario con $n = 3$; $a \cdot b$ es una expresión; $a + (-b)$ es una expresión. $a \div b$ es una expresión.



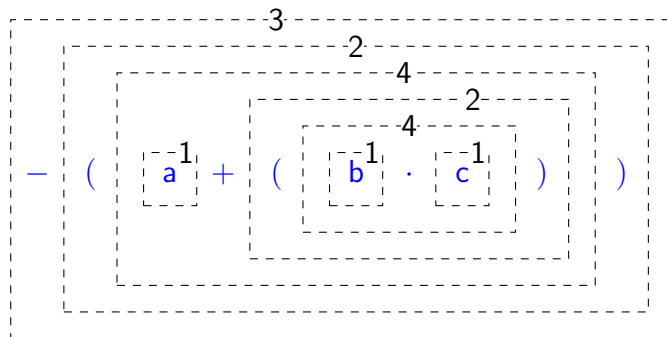
$\text{mín}(1, \text{máx}(a + b, a - b, 3))$

mín es un operador n -ario con $n = 2$, es decir, binario; 1 es una expresión; $\text{máx}(a + b, a - b, 3)$ es una expresión.



Ejemplo 1.5. La expresión $-(a + (b \cdot c))$ es una expresión aritmética porque:

- Como b y c son expresiones, por ser variables y \cdot es un operador binario, entonces $b \cdot c$ es una expresión.
- Como $b \cdot c$ es una expresión, entonces $(b \cdot c)$ es una expresión.
- Como a y $(b \cdot c)$ son expresiones y $+$ es un operador binario entonces $a + (b \cdot c)$ es una expresión.
- Como $a + (b \cdot c)$ es una expresión, entonces $(a + (b \cdot c))$ es una expresión.
- Como $(a + (b \cdot c))$ es una expresión y $-$ es un operador unario, entonces $-(a + (b \cdot c))$ es una expresión.



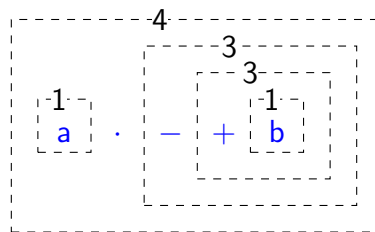
Ejemplo 1.6. Observemos la expresión $(a \cdot b) - (4 \cdot a \cot b - 2 \cdot b)$.

- a y b son variables, por lo que a su vez son expresiones.
- \cdot es un operador binario, por lo que, junto con el inciso anterior, $a \cdot b$ es una expresión.
- Como $a \cdot b$ es una expresión, también lo es $(a \cdot b)$.
- Como 4 es una constante y a una variable, $4 \cdot a$ es una expresión.

Hasta acá vamos bien. Pero ninguna de nuestras reglas indica que \cot sea un operador binario y no existe la posibilidad de que haya una sucesión de variables sin nada entre ellas. Por lo que ésta no es una expresión aritmética bien construida.

Ejemplo 1.7. La expresión $a \cdot - + b$ es una expresión aritmética que utiliza el uso de las siguientes reglas para su construcción:

- a es una expresión
- b es una expresión
- $+b$ es una expresión pues b es una expresión y $+$ es un operador unario.
- $- + b$ es una expresión pues $+b$ es una expresión y $-$ es un operador unario.
- $a \cdot - + b$ es una expresión pues tanto a como $- + b$ son expresiones y \cdot es un operador binario.



Ejemplo 1.8. La sucesión de símbolos $\cdot - + a b$ no es una expresión aritmética correcta, pues no se puede obtener a partir de las reglas anteriores. ¿Por qué?

De los ejemplos anteriores se observa que para mostrar que una sucesión dada de símbolos s es una expresión aritmética, debemos identificar cada uno de sus componentes y asociarlos con alguna de las reglas de formación de expresiones. Este método puede resultar demasiado tedioso de aplicar por lo que debemos buscar métodos más sencillos y susceptibles de ser automatizados.

1.3. Gramáticas y árboles de derivación

Otra manera de mostrar cómo se construyen las expresiones es mediante lo que se conoce como una *gramática formal*, que es un mecanismo sencillo de especificación de reglas de construcción, llamadas *producciones o reglas de reescritura*, con las cuales se pueden generar expresiones de un lenguaje. Las formas que pueden tomar estas reglas de reescritura son muy variadas, pero nos ocuparemos sólo de una de éstas. Las reglas de reescritura de las que nos ocuparemos tienen la siguiente forma:

$$\boxed{\text{símbolo}} ::= \boxed{\text{cadena}}$$

El símbolo “ $::=$ ” se lee “se puede reescribir como” y al aplicar una regla particular sustituimos el lado izquierdo de este símbolo por la cadena que se encuentra del lado derecho.

Para las expresiones aritméticas, por ejemplo, tendríamos las reglas de reescritura que aparecen en la tabla 1.1. En ellas, los símbolos que aparecen en gris o azul (con este tipo de letra) son aquellos que ya no pueden ser reescritos, pues no aparecen del lado izquierdo de ninguna regla de reescritura. A estos símbolos les llamamos *símbolos terminales*, pues son los que *terminan* las cadenas de reescritura. A los símbolos que pueden ser reescritos les llamamos *no terminales* o *variables*. El símbolo “ $|$ ” juega el papel de separador de opciones, para ahorrar trabajo.

Tabla 1.1 Reglas de reescritura para expresiones aritméticas

$$S ::= E \quad (1.1)$$

$$E ::= \text{var} \quad (1.2)$$

$$E ::= \text{const} \quad (1.3)$$

$$E ::= \triangleright E \quad (1.4)$$

$$E ::= E \diamond E \quad (1.5)$$

$$E ::= (E) \quad (1.6)$$

$$\text{var} ::= \text{a} \mid \text{b} \mid \dots \quad (1.7)$$

$$\text{const} ::= 0 \mid 1 \mid 2 \mid 17 \mid 3.5 \mid \dots \quad (1.8)$$

$$\triangleright ::= + \mid - \quad (1.9)$$

$$\diamond ::= + \mid - \mid \cdot \mid \div \quad (1.10)$$

A una colección de reglas de reescritura como la anterior le llamamos *gramática* porque nos describe la *forma o reglas sintácticas* que deben tener las expresiones aritméticas bien construidas. Para producir “oraciones” (cadenas, palabras, expresiones) correctas de acuerdo a las reglas de la gramática, empezamos con el símbolo a la izquierda del $::=$ de la

primera regla, y utilizamos las reglas de reescritura, que nos dicen que en cualquier momento podemos sustituir parte de (o toda) la expresión, si en ella localizamos una subexpresión³ que corresponda al lado izquierdo de cualquiera de las reglas y la sustituimos por el lado derecho. Cada vez que hacemos una sustitución, encontramos en la frase el primer símbolo desde el extremo izquierdo que aparece a la izquierda de $::=$ de alguna de las reglas y lo sustituimos por la cadena a la derecha de $::=$ en esa regla. En cada paso únicamente sustituimos a un símbolo. Decimos entonces que estamos haciendo una sustitución *por la izquierda*. Es importante mencionar que el orden en que se hagan las sustituciones no es obligatoriamente por la izquierda y no afecta el resultado final, aunque es bueno convenir en hacerlo por la izquierda en aras de obtener un método único de construcción, es decir un método *determinista*. Veamos algunos ejemplos en la figura 1.1 a continuación.

Figura 1.1 Proceso de generación de expresiones aritméticas

(1/2)

$-(a \cdot (b + c))$				
Frase	Regla usada			
S	inicio	(—)		
E	$S ::= E$	(1.1)		
$\triangleright E$	$E ::= \triangleright E$	(1.4)		
$-E$	$\triangleright ::= -$	(1.9)		
$-(E)$	$E ::= (E)$	(1.6)		
$-(E \diamond E)$	$E ::= E \diamond E$	(1.5)		
$-(var \diamond E)$	$E ::= var$	(1.2)		
$-(var \cdot E)$	$\diamond ::= \cdot$	(1.10)		
$-(var \cdot (E))$	$E ::= (E)$	(1.6)		
$-(var \cdot (E \diamond E))$	$E ::= E \diamond E$	(1.5)		
$-(a \cdot (E \diamond E))$	$var ::= a$	(1.7)		
$-(a \cdot (var \diamond E))$	$E ::= var$	(1.2)		
$-(a \cdot (var + E))$	$\diamond ::= +$	(1.10)		
$-(a \cdot (var + var))$	$E ::= var$	(1.2)		
$-(a \cdot (b + var))$	$var ::= b$	(1.7)		
$-(a \cdot (b + c))$	$var ::= c$	(1.7)		

Gramática:

$S ::= E$	(1.1)
$E ::= var$	(1.2)
$E ::= const$	(1.3)
$E ::= \triangleright E$	(1.4)
$E ::= E \diamond E$	(1.5)
$E ::= (E)$	(1.6)
$var ::= a \mid b \mid \dots$	(1.7)
$const ::= 0 \mid 1 \mid 2 \mid 17 \mid 3.5 \mid \dots$	(1.8)
$\triangleright ::= + \mid -$	(1.9)
$\diamond ::= + \mid - \mid \cdot \mid \div$	(1.10)

³Una *subexpresión* es una expresión que aparece dentro de otra. Esto implica que debe estar bien construida.

Figura 1.1 Proceso de generación de expresiones aritméticas

(2/2)

a			(3.0 + 21)		
Frase	Regla usada		Frase	Regla usada	
<i>S</i>	inicio	(—)	<i>S</i>	inicio	(—)
<i>E</i>	$S ::= E$	(1.1)	<i>E</i>	$S ::= E$	(1.1)
<i>var</i>	$E ::= var$	(1.2)	(<i>E</i>)	$E ::= (E)$	(1.6)
a	$var ::= a$	(1.7)	(<i>E</i> \diamond <i>E</i>)	$E ::= E \diamond E$	(1.5)
			(<i>const</i> \diamond <i>E</i>)	$E ::= const$	(1.3)
			(<i>const</i> + <i>E</i>)	$\diamond ::= +$	(1.10)
			(<i>const</i> + <i>const</i>)	$E ::= const$	(1.3)
			(3.0 + <i>const</i>)	$const ::= 3.0$	(1.8)
			(3.0 + 21)	$const ::= 21$	(1.8)

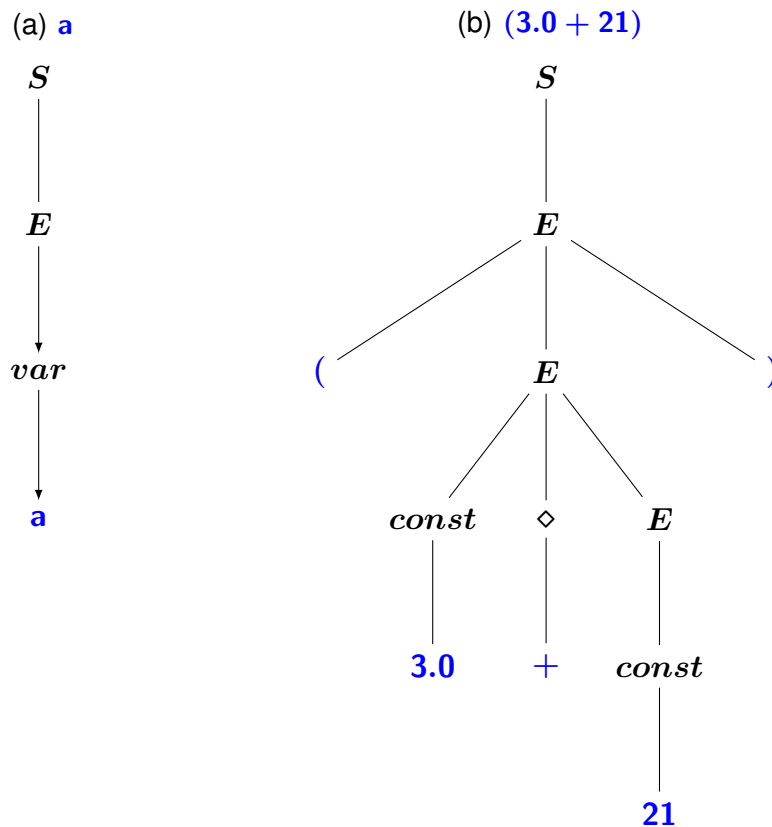
Una secuencia de aplicación de reglas de reescritura, como las que se muestran en la figura 1.1 se conoce como una *derivación* de una expresión; esta expresión es la que figura en el último de sus renglones. Estas derivaciones pueden presentarse así o gráficamente utilizando un *árbol*. Un árbol es una gráfica que remeda a un árbol biológico, excepto que elegimos pintarlo “de cabeza”. Nuestro árbol tiene un nodo inicial llamado raíz, que corresponde al origen de las sustituciones, y va creciendo de la siguiente manera:

- Cada nodo tiene un símbolo asociado.
- Para que de un nodo salgan flechas (o simplemente líneas o aristas, ya que la dirección es siempre hacia abajo) se requiere que el símbolo que está en ese nodo aparezca del lado izquierdo de alguna producción.
- Las flechas (o líneas) apuntan a cada uno de los símbolos que aparecen del lado derecho de la producción utilizada.
- Es importante observar que un nodo tiene un único símbolo asociado.
- Si el símbolo en un nodo se reescribe en una sucesión de tres símbolos, entonces deberán salir tres líneas de él, una por cada símbolo.
- Aquellos nodos de los que no salen líneas les llamamos *hojas*.
- Para determinar cuál es la expresión que corresponde a un determinado árbol, vamos listando las hojas del árbol de izquierda a derecha. A la cadena que se obtiene de esta manera le vamos a llamar el *resultado del árbol*.

En el árbol se pierde el orden en que se hacen las sustituciones; en cada nivel se muestran las sustituciones elegidas, de entre las posibles, en la frase que se encuentra en ese

nivel. Aquellos elementos que aparecen en gris (o en azul o con este tipo de letra) son los que no aparecen del lado izquierdo de ninguna regla y se encuentran colocados en las hojas del árbol. A estos símbolos les llamamos *símbolos terminales* y son los únicos que pueden aparecer en una expresión correcta. Los niveles intermedios no corresponden a expresiones, sino a *descripciones* de lo que puede convertirse en una expresión si se hacen los reemplazos necesarios. Veamos este proceso en las figuras 1.2 en esta página y 1.3 en la siguiente.

Figura 1.2 Ejemplos de árboles de derivación

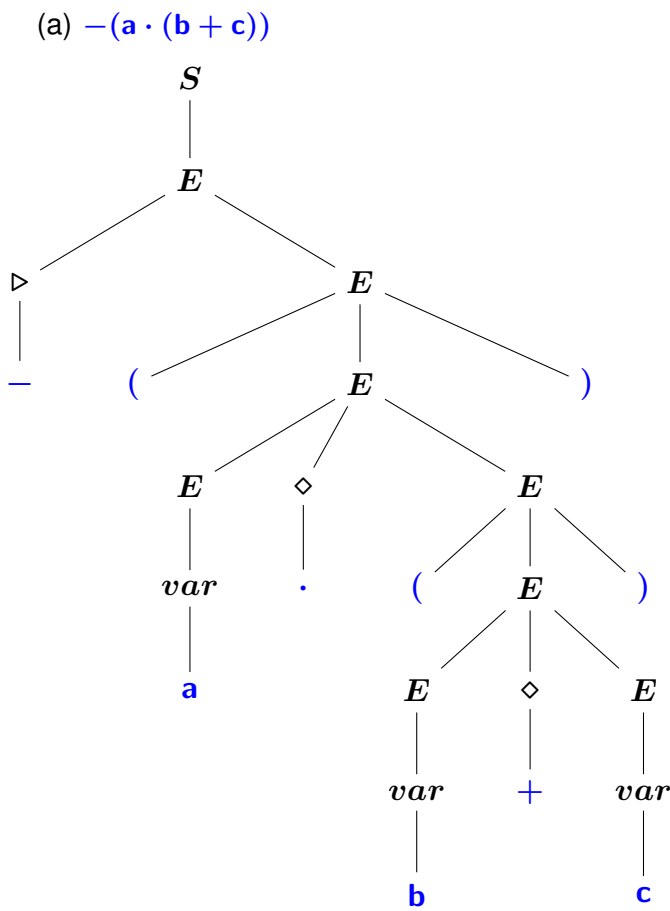


En este contexto, decimos que una expresión aritmética está *bien construida* o que es *correcta* si podemos construir el árbol que representa a su derivación. Este proceso de construcción consiste de las sustituciones que fuimos realizando en cada nivel hasta llegar a un árbol donde todas sus hojas son símbolos terminales. En otras palabras, una expresión aritmética es válida si es el resultado de algún árbol de derivación.

Una vez que tenemos bien escrita una expresión aritmética, queremos obtener su valor. Éste se obtiene reemplazando cada una de las variables que aparecen en la expresión por algún valor permitido y realizando las operaciones correspondientes. Si la expresión es

aritmética, el resultado consistirá de algún número.

Figura 1.3 Otro ejemplo de árbol de derivación



Expresiones de paréntesis balanceados

Como otro ejemplo del uso de gramáticas y árboles de derivación presentamos las expresiones de paréntesis balanceados. Este lenguaje es parte esencial de cualquier lenguaje de programación. La gramática que lo define se encuentra en la tabla 1.2.

Tabla 1.2 Gramática que define a paréntesis bien balanceados

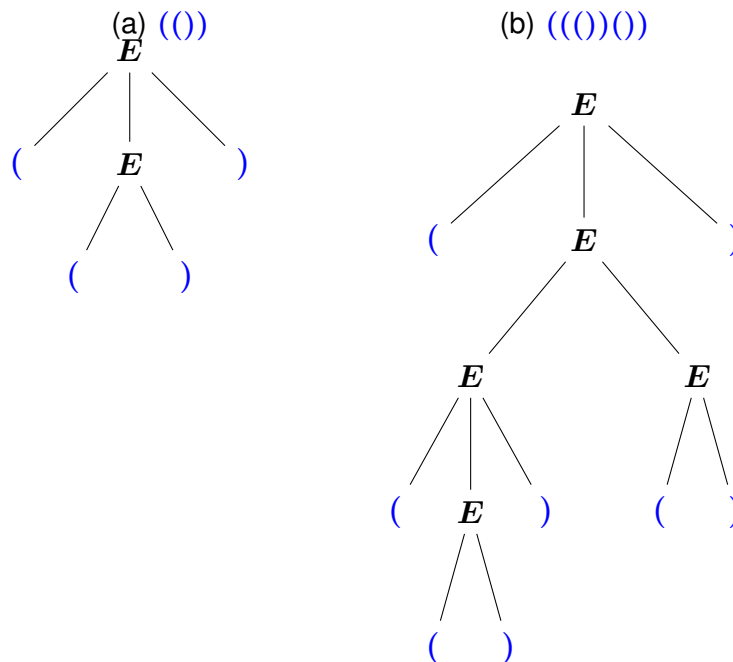
- $E ::= ()$
- (1.11)
- $E ::= (E)$
- (1.12)
- $E ::= EE$
- (1.13)

Obsérvese que esta gramática produce expresiones que constan únicamente de paréntesis balanceados, no hay números ni otra clase de objetos que no sean paréntesis. La regla 1.11 corresponde a la expresión más simple de paréntesis balanceados $()$ mientras que la regla 1.12 corresponde a encerrar entre paréntesis una expresión anterior. Por otra parte la regla 1.13 representa la generación de una nueva expresión con paréntesis balanceados al “pegar” o concatenar dos expresiones previas. Veamos un par de ejemplos.

Figura 1.4 Expresiones de paréntesis balanceados

()			(())		
Frase	Regla usada		Frase	Regla usada	
E	inicio	(—)	E	inicio	(—)
(E)	$E ::= (E)$	(1.12)	(E)	$E ::= (E)$	(1.12)
$(())$	$E ::= ()$	(1.11)	(EE)	$E ::= EE$	(1.13)
			$(()E)$	$E ::= ()$	(1.11)
			$(())()$	$E ::= ()$	(1.11)

Figura 1.5 Ejemplos de árboles de derivación para expresiones de paréntesis balanceados



Ejercicios

1.3.1.- Dadas las producciones para construir expresiones aritméticas, para cada una de las siguientes expresiones decir si se pueden o no construir con esas producciones. Justifica tu respuesta.

a) $- + -a$

b) $2(b \cdot b)$

c) $\frac{1}{2}(a + b)$

1.3.2.- Usando la gramática que dimos para expresiones aritméticas, dibujar los árboles que corresponden a cada una de las expresiones que siguen:

a) $-a \cdot b + c$

b) $(-b + (b \cdot b - 4 \cdot a \cdot c)) \div (2 \cdot a)$

c) $-a + b$

1.3.3.- Dadas las expresiones aritméticas del ejercicio 1.3.2, da la secuencia de producciones que usas, haciendo siempre la sustitución por la izquierda.

1.3.4.- Dada las siguientes producciones:

$$S ::= aSb \quad (1.14)$$

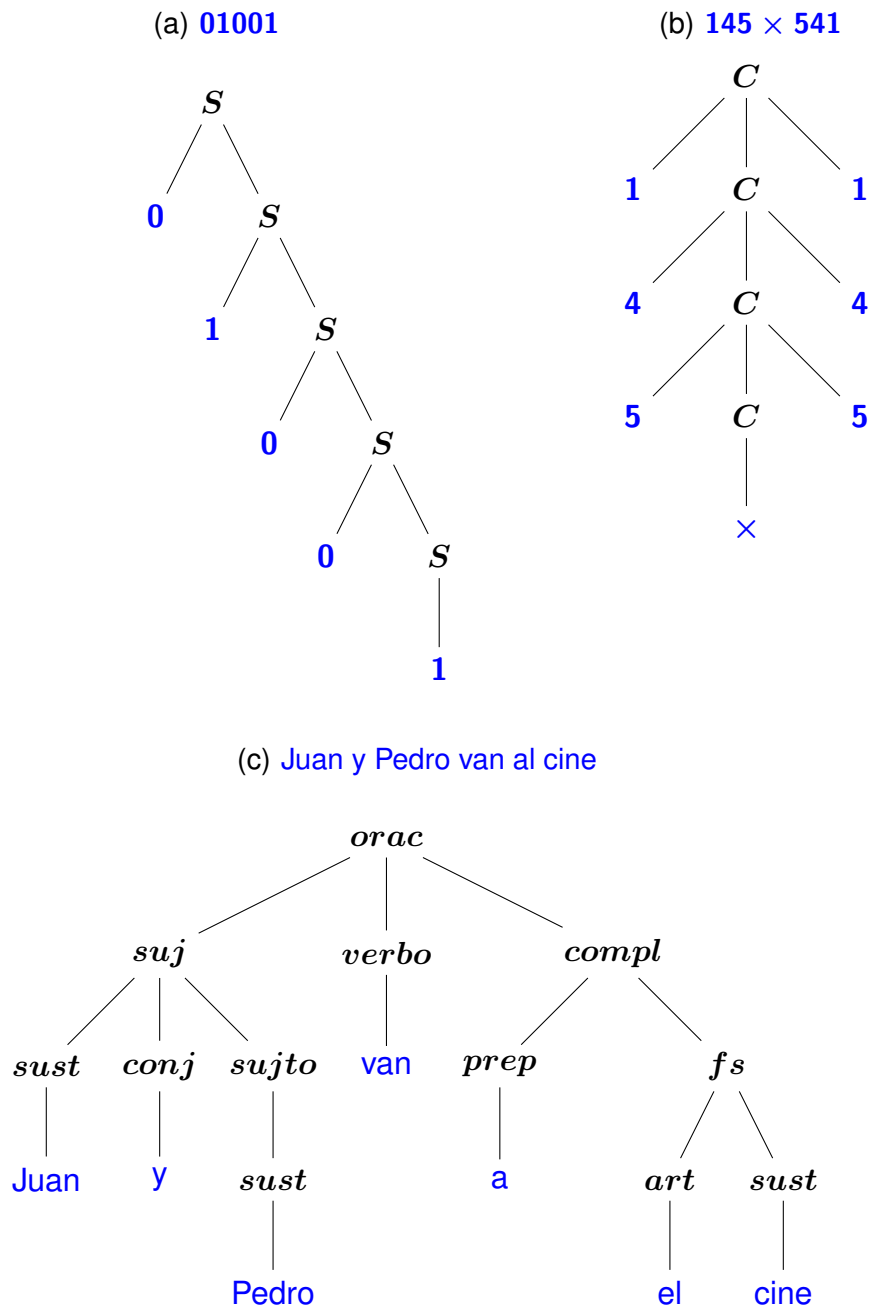
$$S ::= ab \quad (1.15)$$

$$S ::= bSa \quad (1.16)$$

$$S ::= ba \quad (1.17)$$

Da 3 expresiones que se puedan derivar de esta gramática.

1.3.5.- Para cada uno de los árboles de la figura 1.6, da las producciones que tuvieron que utilizarse para construirlos:

Figura 1.6 Ejercicio 1.3.5

1.3.6.- Para cada uno de los árboles del ejercicio 1.3.5, da otras dos expresiones o frases distintas a la dada que se puedan construir usando las mismas producciones.

Lógica proposicional | 2

2.1. El lenguaje de la lógica proposicional

En esta sección nos dedicamos a definir la sintaxis y semántica del lenguaje formal de la lógica proposicional, empezando con una discusión acerca de argumentos lógicos.

2.1.1. Argumentos lógicos

Uno de los aspectos más importantes de la lógica matemática es el decidir si un argumento es correcto o no. Entre nuestros objetivos está el de utilizar la lógica como una herramienta para evidenciar o (*deducir*) la solidez o correctud de un argumento lógico. Pero para empezar debemos contestar ¿qué es un argumento lógico? En general un argumento o argumentación se da en lenguaje natural presentando ciertos hechos – “alegatos”, verdades, situaciones – así como una *conclusión* que, si la argumentación es correcta, debe ser evidente de los hechos anteriores a los cuales llamamos *premisas*. Veamos algunos ejemplos:

Si llueve, me quedo en casa. Si me quedo en casa, leo un libro. Por lo tanto, si llueve, leo un libro

Si me gusta el curso, pongo atención; si pongo atención, entiendo el material. Luego entonces, si me gusta el curso, entiendo el material

x es mayor o igual que y o bien x es menor que y . x no es mayor o igual que y . De manera que x es menor que y

Ahora bien, ¿cómo distinguimos entre las premisas y la conclusión del argumento? Esto depende de ciertas frases del lenguaje natural que nos dan la pauta para hacer la distinción, frases como *por lo tanto*, *luego entonces*, *de manera que*, etc.

Una vez identificadas la conclusión y las premisas se puede reescribir el argumento de una forma estructurada omitiendo ciertas frases del lenguaje natural, como en los siguientes ejemplos:

- (a)
 - 1. Si llueve, me quedo en mi casa
 - 2. Si me quedo en mi casa, leo un libro
 -
 - 3. Si llueve, leo un libro
- (b)
 - 1. Si me gusta el curso, pongo atención
 - 2. Si pongo atención, entiendo el material
 -
 - 3. Si me gusta el curso, entiendo el material
- (c)
 - 1. x es mayor o igual que y o bien x es menor que y
 - 2. x no es mayor o igual que y
 -
 - 3. x es menor que y
- (d)
 - 1. Los libros son baratos o son caros
 - 2. Los libros no son caros
 -
 - 3. Los libros son baratos
- (e)
 - 1. Este programa funciona mal o los datos son incorrectos
 - 2. Los datos son correctos
 -
 - 3. Este programa funciona mal

Obsérvese que la conclusión está separada de las premisas mediante una línea horizontal. Además, de acuerdo a nuestra intuición, todos los argumentos anteriores parecen correctos, pero formalmente ¿cuándo un argumento es correcto? o ¿cómo decidimos si un argumento es correcto?. Para responder a esta pregunta nos serviremos de la lógica matemática, la cual nos proporcionará un conjunto de reglas operacionales, que en particular permitirán obtener – deducir, encontrar, conformar, derivar – un nuevo hecho a partir de ciertos hechos dados. Un argumento lógico será *correcto o sólido* si la verdad de sus premisas causan necesaria y obligatoriamente la verdad de su conclusión, lo cual puede

mostrarse mediante las reglas lógicas de operación.

Aristóteles fue el primero que para poder manipular argumentos lógicos optó por asignarles letras a ciertas frases consideradas de estructura lógica simple, llamadas proposiciones o fórmulas atómicas. De esta manera podemos ver en forma concisa los argumentos lógicos. Procedamos a hacer esto con los argumentos anteriores para poderlos mostrar a la manera aristotélica.

<p>(a)</p> <p>p llueve</p> <p>q me quedo en mi casa</p> <p>r leo un libro</p> <p>1. Si p, q</p> <p>2. Si q, r</p> <hr/> <p>3. Si p, r</p>	<p>(b)</p> <p>p Me gusta el curso</p> <p>q pongo atención</p> <p>r entiendo el material</p> <p>1. Si p, q</p> <p>2. Si q, r</p> <hr/> <p>3. Si p, r</p>
<p>(c)</p> <p>p x es mayor y</p> <p>q x es igual a y</p> <p>r x es menor que y</p> <p>1. $p \vee q \vee r$</p> <p>2. no $(p \vee q)$</p> <hr/> <p>3. r</p>	<p>(d)</p> <p>p Los libros son baratos</p> <p>q Los libros son caros</p> <p>1. $p \vee q$</p> <p>2. no q</p> <hr/> <p>3. p</p>
<p>(e)</p> <p>p Este programa funciona mal</p> <p>q Los datos son correctos</p> <p>1. $p \vee \text{no } q$</p> <p>2. q</p> <hr/> <p>3. p</p>	

Se observa que el uso de letras deja ver patrones o esquemas comunes, aunque aún tenemos algunas palabras del español. Para deshacernos de ellas y formalizar completamente el estudio de argumentos lógicos introducimos ahora el lenguaje formal de la lógica proposicional. Observen que en el inciso (c), cuando decimos “no $(p \vee q)$ ” estamos manifestando “ni p ni q ”.

2.1.2. Propositiones

De manera similar a como construimos expresiones aritméticas, vamos ahora a definir y construir expresiones lógicas. Empecemos por ver cuáles son los objetos elementales de la lógica. En la aritmética teníamos valores numéricos (constantes) y de forma similar

la lógica tiene constantes, pero sólo dos: 0 (*falso*) y 1 (*verdadero*). Estas constantes se conocen como valores lógicos o booleanos. Usar los valores de 0 y 1 como sinónimos de *falso* y *verdadero* es una libertad que nos damos las personas dedicadas a computación. También podemos hablar de los valores F y T (*false* y *true* respectivamente), aunque a lo largo de este texto usaremos 0 y 1 pues es así como se van a representar en la computadora.

Las expresiones lógicas se conocen también como *proposiciones* y son enunciados u oraciones a las que les podemos asociar un valor lógico (tienen valor de 0 o 1). En general las proposiciones se dan en lenguaje natural; un enunciado es una proposición solamente si se puede decir, en un contexto dado, si es falso o verdadero. En el ejemplo (a) que acabamos de dar, la proposición $p = \text{llueve}$ es falsa o verdadera dependiendo del momento en que se diga, de si en ese momento está lloviendo o no.

Cuando decimos que una proposición es falsa o verdadera, estamos *determinando* el valor de dicha proposición. De manera similar para las expresiones aritméticas, podemos hablar del *valor* de la expresión aritmética, que nos va a dar una constante numérica calculada a partir de los valores de cada una de las variables y constantes involucradas en la expresión. A este conjunto de valores le llamamos el *estado* en el que se evalúa la expresión – a lo que anteriormente llamamos el contexto de una proposición –. Podemos definir entonces un *estado* como un conjunto de parejas, donde cada pareja tiene el nombre de una variable y el valor de esa variable. Un ejemplo de cómo especificamos un estado se encuentra a continuación.

$$\text{estado} = \{(x, 5), (y, 7), (p, \text{falso})\}$$

En este estado tenemos los valores para tres variables, dos numéricas y la tercera lógica. Cada elemento del conjunto es una pareja ordenada, donde primero se da el nombre de la variable y después el valor de esa variable en el estado.

Regresando a las expresiones lógicas, son proposiciones:

- ☞ Está lloviendo
- ☞ Juan es más grande que Pedro
- ☞ $x \geq z$
- ☞ El libro es rojo
- ☞ Roberto es el asesino
- ☞ Esta materia es fácil

No son proposiciones:

- ☞ ¡Mario, llévate esto!
- ☞ ¿Estás seguro?
- ☞ $x + y$
- ☞ Ni modo

☞ ¡Viva Pancho Villa!

Intuitivamente a las proposiciones se les puede evaluar, es decir, decidir si son falsas o verdaderas. Pero como mencionamos antes, este valor depende del *estado* que tomen sus variables. Por ejemplo, la tercera proposición de la lista que dimos es verdadera si el estado es $\{(x, 5.6), (z, 3.0)\}$. En este estado particular, la proposición tiene el valor de verdadero. Evaluada en el estado $\{(x, 2.3), (y, 4.0)\}$ la proposición tiene el valor de falso. La cuarta proposición tendrá el valor de verdadero en el caso de que el libro de que estemos hablando sea rojo, es decir cuando estemos en el estado $\{(color\ del\ libro, rojo)\}$.

Más adelante hablaremos formalmente de estados y del proceso de evaluación. Por ahora sigamos con el estudio de las proposiciones

Definición 2.1 Una **proposición** es un enunciado que puede calificarse como falso (0) o verdadero (1), dependiendo del estado en que se evalúe.

Decimos que una proposición es *atómica* si no puede subdividirse en proposiciones más simples. Las proposiciones anteriores son todas atómicas. En contraste, las siguientes proposiciones no son atómicas:

☞ Juan y Pedro están hambrientos

☞ Está nublado, por lo que va a llover, entonces no saldremos

☞ $0 \leq x \leq 10$

☞ El libro es rojo o azul

Estas proposiciones se llaman *compuestas* pues cada una de ellas se puede descomponer en dos o más proposiciones atómicas como a continuación se muestra:

- Juan y Pedro están hambrientos

☞ Juan está hambriento

y

☞ Pedro está hambriento

- Está nublado, por lo que va a llover; entonces no saldremos

☞ Está nublado,

por lo que

☞ va a llover

entonces

☞ **no** saldremos

- $0 \leq x \leq 10$
 - ☞ $0 \leq x$
 - y
 - ☞ $x \leq 10$
- El libro es rojo o azul
 - ☞ el libro es rojo
 - o
 - ☞ el libro es azul

Las proposiciones atómicas son aquellas que están a continuación de ☞ y hasta el final del renglón. Encerramos en un marco a la palabra o frase que relaciona a la primera proposición atómica con la siguiente y así sucesivamente. A estas palabras les llamamos *conectivos*.

A continuación vamos a pasar de las proposiciones en lenguaje natural al estudio de un lenguaje formal de expresiones lógicas. En el proceso de traducción o especificación de lenguaje natural al formal se acostumbra asociar identificadores (letras) a las proposiciones atómicas, para poder escribir de manera más fluida y así representar y manipular adecuadamente a las proposiciones. Obsérvese que esto ya lo hicimos en la semi formalización de argumentos en la introducción de este capítulo. A estos identificadores se les conoce como *variables proposicionales*. Ya tenemos entonces variables, pero para construir expresiones más complejas necesitamos de constantes y operadores lógicos y que corresponden estos últimos a las frases en lenguaje natural que hemos llamado conectivos.

2.1.3. Sintaxis de la lógica proposicional

En esta sección definimos un lenguaje formal para la lógica proposicional mediante una gramática para expresiones lógicas.

Las reglas para construir proposiciones son las siguientes:

$$P ::= VarProp \quad (2.1)$$

$$P ::= ConstLog \quad (2.2)$$

$$P ::= \neg P \quad (2.3)$$

$$P ::= P \diamond P \quad (2.4)$$

$$P ::= (P) \quad (2.5)$$

$$VarProp ::= a, b, \dots, p, q, \dots \quad \text{variables proposicionales} \quad (2.6)$$

$$ConstLog ::= \text{false}, \text{true} \quad \text{constantes lógicas} \quad (2.7)$$

$$\neg ::= \neg \quad \text{negación (not)} \quad (2.8)$$

$$\diamond ::= \wedge, \quad \text{y, además, pero} \quad (and) \quad (2.10)$$

$$\vee, \quad \text{o} \quad (or) \quad (2.11)$$

$$\rightarrow, \quad \text{implica,} \quad (2.12)$$

si ... entonces, por lo que,
de... se sigue (*implies*)

$$\leftrightarrow \quad \text{si y sólo si, sii, syss, iff} \quad (2.13)$$

(*if and only if*)

Veamos ahora el paso del español al lenguaje formal de proposiciones mediante algunos ejemplos. Considérese la siguiente asignación de significados a variables proposicionales:

<i>Proposición atómica</i>	<i>Variable proposicional</i>
Juan está hambriento	a
Pedro está hambriento	b
está nublado	c
va a llover	d
saldremos	e
$0 < x$	p
$x < 10$	q
el libro es rojo	r
el libro es azul	s

Las proposiciones no atómicas de los ejemplos anteriores son representadas de la siguiente manera:

- Juan y Pedro están hambrientos

$$a \wedge b$$

- Está nublado por lo que va a llover; entonces no saldremos

$$(c \rightarrow d) \rightarrow \neg e$$

$$0 < x < 10$$

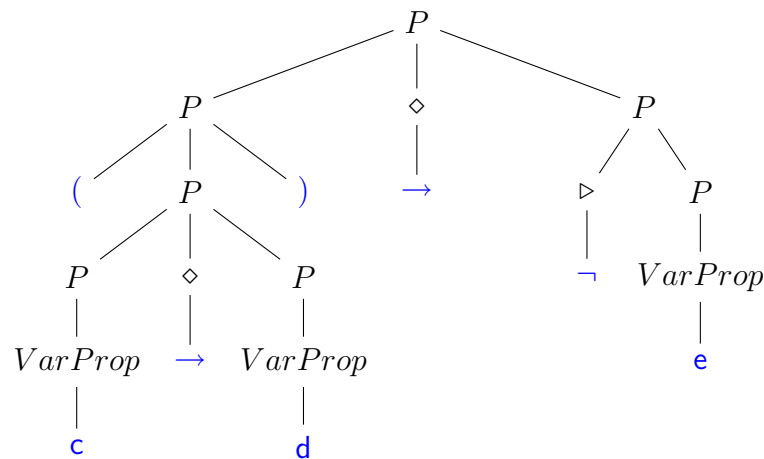
$$p \wedge q$$

$$\text{El libro es rojo o el libro es azul}$$

$$r \vee s$$

Veamos ahora el árbol de derivación para alguna de estas expresiones, digamos $(c \rightarrow d) \rightarrow \neg e$, en la figura 2.1.

Figura 2.1 Derivación de $(c \rightarrow d) \rightarrow \neg e$



Nuevamente, los símbolos terminales están en distinto tipo y color.

2.1.4. Semántica de la lógica proposicional

Una vez que hemos discutido informalmente qué es una proposición así como la sintaxis de un lenguaje formal para proposiciones, es momento de hablar de su significado. Los aspectos relacionados con el significado de cualquier clase de expresiones forman lo que se conoce como la *semántica* del lenguaje. En nuestro caso ya conocemos el significado intuitivo de las proposiciones, de hecho le hemos dado a los operadores lógicos un nombre relativo a su significado. Por ejemplo la proposición $\neg p$ se lee “no p ” y representa a la negación de la información especificada por p . En analogía a las expresiones aritméticas cuyo significado es un número, calculado al hacer las operaciones dadas en la expresión de acuerdo a un estado particular de sus variables, cada proposición tiene como significado

un valor booleano que depende tanto del valor particular de sus variables proposicionales como del significado de las constantes y operadores lógicos. De manera que para poder entender el significado de una proposición debemos empezar por definir el significado o funcionamiento de cada constante u operador lógico. El significado de las constantes lógicas debe ser claro, la constante *true* significa verdadero (1) y la constante *false* significa falso (0). La manera más fácil para definir el significado de un operador lógico es mediante lo que se conoce como *tablas de verdad*. En lo que sigue se usan mayúsculas para denotar proposiciones que pueden ser compuestas. A continuación analizamos cada operador lógico.

La negación

La negación de una proposición P se denota de alguna de las siguientes formas:

$$\neg P, \sim P, \overline{P}, P'$$

Nosotros usaremos $\neg P$ exclusivamente.

Su significado en español es:

$\neg P$

no P

no es cierto que P

es falso que P

Su tabla de verdad es:

	negación
P	$\neg P$
1	0
0	1

Este tipo de tablas merece algunas observaciones. Para calcular la tabla de verdad de una proposición cualquiera E es necesario considerar todos los *estados* posibles de los operandos de la expresión E . Cada operando puede estar en uno de dos estados posibles, 1 para verdadero y 0 para falso. Cada renglón de la tabla corresponde a un estado particular de los operandos. En este caso nuestra expresión es $\neg P$, que tiene como único operando a P , que independientemente de que sea una proposición atómica o no, sólo puede estar en dos estados posibles, por lo que la tabla de verdad sólo tiene dos renglones. En esta tabla, la primera columna es la que indica el estado del operando P mientras que la segunda nos indica el resultado de la evaluación de la expresión deseada, en este caso $\neg P$. Como se ve en la tabla anterior, el operador \neg lo que hace es “invertir” o negar el valor original de la proposición dada.

Veamos a continuación la semántica de los operadores lógicos binarios.

La conjunción

La conjunción de dos proposiciones P y Q se denota de alguna de las siguientes formas:

$$P \wedge Q, P \& Q, P \cdot Q, PQ$$

No significando $P \wedge Q$ exclusivamente.

Su significado en español es:

$P \wedge Q$
P y Q
P además de Q
P pero Q

Puede observarse aquí cierta incapacidad de la lógica para representar al español: ciertamente al usar la palabra *pero* se le está dando cierta intensión a una afirmación que no corresponde a la simple conjunción, como en la frase *Te llevo al cine, pero haces la tarea*, la cual sólo puede representarse con una conjunción que corresponde a *Te llevo al cine y haces la tarea*. Desafortunadamente, en lógica la única posibilidad para representar un *pero* es la conjunción.

Su tabla de verdad es:

P	Q	Conjunción $P \wedge Q$
1	1	1
1	0	0
0	1	0
0	0	0

En esta ocasión, al haber dos operandos (P y Q), tenemos cuatro posibles estados para el sistema:

- Que ambas proposiciones valgan 1
- Que P valga 0 y Q valga 1
- Que P valga 1 y Q valga 0
- Que ambas proposiciones valgan 0

La disyunción

La disyunción de dos proposiciones P y Q se denota de alguna de las siguientes formas:

$$P \vee Q, P \mid Q, P + Q$$

Nosotros usaremos $P \vee Q$ exclusivamente.

Su significado en español es:

$$P \vee Q$$

$$P \text{ o } Q$$

$$\text{o } P \text{ o } Q$$

Su tabla de verdad es:

P	Q	Disyunción $P \vee Q$
1	1	1
1	0	1
0	1	1
0	0	0

Observando el primer renglón de la tabla de verdad nos damos cuenta de que este uso de la disyunción es inclusivo, es decir, la disyunción es cierta también en el caso en que ambos operandos sean ciertos.

La implicación

La implicación o condicional de dos proposiciones P y Q se denota de alguna de las siguientes formas:

$$P \rightarrow Q, P \Rightarrow Q, P \supset Q$$

Nosotros usaremos $P \rightarrow Q$ exclusivamente. Su significado en español es:

$P \rightarrow Q$
si P entonces Q
P implica Q
P es (condición) suficiente para Q
Q , si P
P sólo si Q
Q se sigue de P
Q es (condición) necesaria para P

Su tabla de verdad es la que sigue:

P	Q	Implicación o condicional $P \rightarrow Q$
1	1	1
1	0	0
0	1	1
0	0	1

Nos sorprende en esta tabla la evaluación del primer y segundo renglones, pues parece, a primera vista, contrario a la intuición. Veamos un ejemplo:

Ejemplo 2.1.

p	es	una botella contiene ácido
q	es	la botella tiene una calavera en la etiqueta
$p \rightarrow q$	es	si una botella tiene ácido, entonces tiene una calavera en la etiqueta

Como se ve en este ejemplo, la verdad de p (que la botella contenga ácido) nos permite garantizar la verdad de q (que hay una calavera en la etiqueta). Pero si la botella no contiene ácido, pudiera ser que la botella contenga algún otro compuesto venenoso y que de todos modos tenga una calavera en la etiqueta, estado representado por el tercer renglón de la tabla; pero también pudiera ser que la botella no tenga ácido y que no tenga calavera en la etiqueta, estado representado por el último renglón de la tabla. Lo que no puede suceder (el resultado es 0) es que la botella, conteniendo ácido **no** tenga una calavera en la etiqueta, estado representado por el segundo renglón de la tabla.

Veamos otro ejemplo, esta vez en matemáticas. Considérese la siguiente proposición¹:

$$((x > y) \wedge (y > z)) \rightarrow (x > z)$$

Evaluemos esta expresión en el estado $\{(x, 8), (y, 6), (z, 4)\}$. En este estado, el antecedente de la implicación es verdadero $((8 > 6) \text{ y } (6 > 4))$, por lo que podemos garantizar que $x > z$, pues en efecto, $8 > 4$. Sin embargo, veamos que sucede en el estado $\{(x, 7), (y, 8), (z, 6)\}$. En este caso el antecedente es falso pero el consecuente es verdadero. El valor de la proposición es, de acuerdo a la definición en su tabla de verdad, verdadero. Otro estado que ilustra el primer caso es $\{(x, 4), (y, 6), (z, 5)\}$. También este estado hace que la proposición se evalúe a verdadero, porque una vez que el antecedente es falso, el estado del consecuente puede ser cualquiera. Por otra parte, si el antecedente es verdadero, no puede suceder que el consecuente sea falso, es decir, no existe un estado en el cual $(x > y)$ y $(y > z)$ y que sin embargo tengamos $(x \leq z)$.

Los valores de verdadero y falso de la implicación simplemente nos dicen cuáles estados pueden presentarse y cuáles no. En el primer ejemplo que dimos, si llueve es seguro que me quedo en casa, pero si no llueve, el estado del consecuente puede ser cualquiera. Recordemos que sólo hay dos estados posibles para las proposiciones lógicas, falso o verdadero.

Cada implicación $P \rightarrow Q$ tiene asociadas otras implicaciones que involucran a las mismas proposiciones P y Q que a continuación definimos:

- La *recíproca* o *inversa* de $P \rightarrow Q$ es la fórmula $Q \rightarrow P$.
- La *contrapositiva* de $P \rightarrow Q$ es la fórmula $\neg Q \rightarrow \neg P$.
- La *contrarrecíproca* de $P \rightarrow Q$ es la fórmula $\neg P \rightarrow \neg Q$.

Ejemplo 2.2. Considérese la oración *si tengo un triángulo entonces tengo un polígono*, formalizada como $t \rightarrow p$. Sus implicaciones asociadas son:

- Recíproca: $p \rightarrow t$ que significa *si tengo un polígono entonces tengo un triángulo*.
- Contrapositiva: $\neg p \rightarrow \neg t$ que significa *si no tengo un polígono entonces no tengo un triángulo*.
- Contrarrecíproca: $\neg t \rightarrow \neg p$ que significa *si no tengo un triángulo entonces no tengo un polígono*

Más adelante veremos la relación existente entre una implicación y sus implicaciones asociadas.

¹Usamos aquí tantos paréntesis como se requieran para definir sin ambigüedades la estructura de la expresión lógica.

La equivalencia

La equivalencia o bicondicional de dos proposiciones P y Q se denota de alguna de las siguientes formas:

$$P \leftrightarrow Q, P \Leftrightarrow Q, P \equiv Q$$

Nosotros usaremos $P \leftrightarrow Q$ exclusivamente.

Su significado en español es:

$$P \leftrightarrow Q$$

P si y sólo si Q

P es equivalente a Q

P es (condición) necesaria y suficiente para Q

Su tabla de verdad es:

P	Q	Equivalencia o bicondicional $P \leftrightarrow Q$
1	1	1
1	0	0
0	1	0
0	0	1

En este caso, la equivalencia es verdadera si ambas proposiciones se evalúan a lo mismo: ambas se evalúan a falso o ambas se evalúan a verdadero.

Tablas de verdad para proposiciones compuestas

Al conocerse el significado de cada conectivo lógico mediante su tabla de verdad es posible obtener el significado de cualquier fórmula mediante una tabla de verdad que combine las tablas de cada subfórmula componente de la fórmula original. Veamos un ejemplo.

P	Q	R	$(P \rightarrow \neg Q)$	\vee	$(Q \wedge \neg R)$	\rightarrow	$(\neg P \leftrightarrow R)$
1	1	1	0	0	0	1	0
1	1	0	0	1	1	1	1
1	0	1	1	1	0	0	0
1	0	0	1	1	0	1	1
0	1	1	1	1	0	1	1
0	1	0	1	1	1	0	0
0	0	1	1	1	0	1	1
0	0	0	1	1	0	0	0

Como se observa, las tablas de verdad crecen tanto en columnas como en renglones, al volverse más compleja la fórmula en cuestión. ¿Cuántos renglones tiene la tabla de verdad de una fórmula que tiene n variables proposicionales?

Propiedades de los conectivos lógicos

Vimos en las secciones anteriores lo que constituye una proposición, así como el significado de los principales operadores o conectivos lógicos. De conocer las tablas de verdad para estos conectivos, podemos observar algunas de sus propiedades importantes.

Conmutatividad: Esta propiedad nos dice que el orden en que aparecen las proposiciones relacionadas por el conectivo lógico no afecta el resultado de la operación. Por ejemplo, la evaluación de $p \wedge q$ da siempre el mismo resultado que la evaluación de $q \wedge p$. Esta propiedad la tienen asimismo los operadores aritméticos de suma y multiplicación. De las expresiones aritméticas sabemos, por ejemplo, que ni la resta ni la división son operadores conmutativos: No es lo mismo $7 - 5$ que $5 - 7$; tampoco se evalúa a lo mismo $8 \div 2$ que $2 \div 8$. También en el caso de los conectivos lógicos no todos son conmutativos. Los conectivos \vee , \wedge , \leftrightarrow son conmutativos pues:

El valor de:	es el mismo que el de:
$p \vee q$	$q \vee p$
$p \wedge q$	$q \wedge p$
$p \leftrightarrow q$	$q \leftrightarrow p$

De su tabla de verdad es fácil ver que la implicación (\rightarrow) no es conmutativa.

Asociatividad: En aritmética es claro que $(a + b) + c = a + (b + c)$. Decimos entonces que la suma es *asociativa*. En el caso de los conectivos lógicos no todos tienen esta

propiedad llamada *asociatividad*. Mientras que en la aritmética la suma y la multiplicación son asociativos, esto no es así con la resta y la división. Por ejemplo, en el estado $\{(a, 5), (b, 7), (c, 3)\}$, $a - (b - c) = 1$, mientras que $(a - b) - c = -5$. También, $(a \div b) \div c = 5/21 \approx 0.24$, mientras que $a \div (b \div c) = 15/7 \approx 2.1$. En el caso de la lógica matemática los conectivos que son asociativos son la conjunción (\wedge), la disyunción (\vee) y la equivalencia (\leftrightarrow). Nuevamente, la condicional (\rightarrow) tampoco presenta esta propiedad.

El valor de:	es el mismo que el de:
$(p \wedge q) \wedge r$	$p \wedge (q \wedge r)$
$(p \vee q) \vee r$	$p \vee (q \vee r)$
$(p \leftrightarrow q) \leftrightarrow r$	$p \leftrightarrow (q \leftrightarrow r)$

Elemento identidad: En general, un *elemento identidad* para un operador \star es aquel valor que al operarlo con una expresión el resultado es esa misma expresión, es decir e es una identidad para \star si $e \star x = x = x \star e$ para cualquier expresión x . (Noten que estamos suponiendo la conmutatividad del operador con respecto al elemento identidad y cualquier otro elemento.) En el caso de la suma, el elemento identidad es el 0 puesto que $a + 0 = a = 0 + a$, mientras que en el caso de la multiplicación el elemento identidad es el 1 ya que $a \cdot 1 = a = 1 \cdot a$. Como se ve, el elemento identidad va a depender del operador o conectivo particular. En el caso de los conectivos lógicos, los elementos identidad de cada operador se dan a continuación. Para ver que, en efecto, son elementos identidad, sugerimos desarrollar las tablas de verdad correspondientes.

Operador	Identidad	El valor de	es el valor de
\wedge	true	$p \wedge \text{true}$	p
\vee	false	$p \vee \text{false}$	p
\leftrightarrow	true	$p \leftrightarrow \text{true}$	p

Elemento neutro: También conocido como *dominante*, es aquella constante que al operar con cualquier otro valor, el resultado es la constante misma. Es decir, e es un elemento neutro para el operador \star si $x \star e = e = e \star x$ para cualquier expresión x . En el caso de la aritmética, el 0 (cero) con el producto tiene ese efecto. Hay que notar que la suma, la resta y la división no tienen elemento neutro (el elemento nulo tiene que ser el mismo para todos los valores que puedan participar en la operación). En el caso de las proposiciones lógicas, el elemento neutro de la disyunción (\vee) es la constante true y de la conjunción (\wedge) es la constante false.

El valor de:	es el valor de:
$p \vee \text{true}$	true
$p \wedge \text{false}$	false

Idempotencia: Esta propiedad habla de un operador que al tener dos operandos iguales el resultado es el operando mismo. Por ejemplo, si tenemos la proposición $p \wedge p$ podemos observar de la tabla de verdad, que su valor es el mismo que el de p . Los operadores \wedge y \vee son idempotentes:

p	$p \wedge p$	$p \vee p$
1	1	1
0	0	0

Para la implicación hay otras proposiciones interesantes que vale la pena notar. Se caracterizan porque al operar con la constante false o true dan siempre como resultado el valor de 1:

p	$\text{false} \rightarrow p$	$p \rightarrow \text{true}$
1	1	1
0	1	1

2.1.5. Tautologías y contradicciones

Las tablas de verdad nos permiten observar el valor de una fórmula en *todos* sus posibles estados. Esto nos permite clasificar a las fórmulas de la siguiente manera:

tautologías

Aquellas fórmulas que se evalúan a *verdadero* en todos los estados posibles

contradicciones

Aquellas fórmulas que se evalúan a *falso* en todos los posibles estados

fórmulas contingentes o contingencias

Aquellas fórmulas que no son ni tautologías ni contradicciones

Conocemos ya varias tautologías, como es el caso de $p \vee \neg p$, $p \rightarrow p \vee q$, $p \wedge p \leftrightarrow p$. Para convencernos, veamos sus tablas de verdad en la siguiente página:

p	q	$p \vee \neg p$		$p \rightarrow p \vee q$		$p \wedge p \leftrightarrow p$	
		\vee	\neg	\rightarrow	\vee	\wedge	\leftrightarrow
1	1	1	0	1	1	1	1
1	0	1	0	1	1	1	1
0	1	1	1	1	1	0	1
0	0	1	1	1	0	0	1

Como las tautologías son muy importantes, se elige una notación especial para representarlas. Para ello utilizamos un *metalenguaje*, el cual nos sirve para decir algo respecto al lenguaje formal que estamos utilizando. Ya nos encontramos con metalenguajes con anterioridad. Por ejemplo, nuestras gramáticas con sus producciones corresponden a un metalenguaje, ya que si bien nos describen perfectamente lo que es una expresión, las producciones en sí *no* son expresiones. Podemos pensar también en los esquemas de fórmula que utilizamos (E , $P \vee Q$, $A \rightarrow B$, etc.) como *metaexpresiones*, ya que los usamos para describir a objetos de nuestro lenguaje particular, pero ellos no forman parte del lenguaje. Más adelante hablaremos de esquemas con más detalle.

Volviendo al cálculo proposicional, si A es una proposición que es tautología, escribimos $\models A$. Insistimos en que el símbolo \models **no** es un operador de la lógica proposicional y la expresión $\models P$ **no** es una proposición, sino que nos habla *acerca* de la proposición P , diciéndonos que P es una tautología.

Como ejemplos de tautologías de gran importancia tenemos:

- $p \vee \neg p$ *Ley del tercero excluido*, nos dice que toda proposición tiene que evaluarse a falso o verdadero, que no hay ningún otro valor posible.
- $\text{false} \rightarrow p$ *Falso implica cualquier cosa*. Cuando el antecedente es falso, se puede concluir cualquier proposición.
- $p \rightarrow \text{true}$ Cuando el consecuente es verdadero, cualquier proposición lo implica (lo “justifica”).

Contradicciones

Una *contradicción* es una expresión que se evalúa a falso en todos los estados posibles. Podemos cotejar que una expresión es una contradicción utilizando para ello tablas de verdad, como en el caso de las tautologías. Por ejemplo, $P \leftrightarrow \neg P$ y $P \wedge \neg P$ son ambas contradicciones, como se muestra en las tablas de verdad correspondientes.

P	$\neg P$	$P \leftrightarrow \neg P$	$P \wedge \neg P$
1	0	0	0
0	1	0	0

Las contradicciones están íntimamente relacionadas con las tautologías. Si A es una tautología, entonces $\neg A$ es una contradicción y viceversa.

2.1.6. Argumentos correctos

Una vez que hemos definido la sintaxis y la semántica de las fórmulas de la lógica proposicional, así como el concepto de tautología, podemos dar la definición formal de argumento lógico e introducir formalmente la noción de argumento correcto.

Definición 2.2 Un argumento lógico es una sucesión de fórmulas A_1, \dots, A_n llamadas *premisas* y una fórmula B llamada *conclusión*. Dicha sucesión se escribe usualmente como

$$\begin{array}{c} A_1 \\ \vdots \\ A_n \\ \hline \therefore B \end{array} \quad \text{o bien} \quad A_1, \dots, A_n / \therefore B$$

Nuestro problema fundamental es decidir cuándo un argumento es correcto o válido, lo cual sucederá, como ya mencionamos anteriormente, si y sólo si **suponiendo** que sus premisas son verdaderas, entonces necesariamente la conclusión también lo es. Obsérvese que esta definición corresponde a los llamados *argumentos deductivos*. En contraste, en un *argumento inductivo* se aceptan como válidas conclusiones basadas en observación o probabilidad. Nosotros nos dedicaremos sólo a los argumentos deductivos.

Como ya tenemos a nuestra disposición la definición de tautología, nos servimos de ésta para dar una definición formal de argumento correcto.

Definición 2.3 El argumento

$$A_1, A_2, \dots, A_n / \therefore B$$

es correcto si y sólo si

$$\models A_1 \wedge A_2 \dots A_n \rightarrow B.$$

A la fórmula $A_1 \wedge A_2 \dots A_n \rightarrow B$ se le llama fórmula asociada al argumento lógico.

Por lo tanto, verificar la correctud de un argumento es equivalente a verificar que su fórmula asociada es tautología, para lo cual basta construir su tabla de verdad.

Veamos algunos ejemplos

Ejemplo 2.3. El argumento $p \rightarrow q, p / \therefore q$, es correcto. La fórmula a analizar es $p \wedge (p \rightarrow q) \rightarrow q$.

p	q	p	\wedge	$(p \rightarrow q)$	\rightarrow	q
1	1	1	1	1	1	1
1	0	1	0	0	1	0
0	1	0	0	1	1	1
0	0	0	0	1	1	0

Como muestra la tabla, tenemos una tautología y el argumento es correcto.

Ejemplo 2.4. Analizar el siguiente argumento.

Si hoy es viernes entonces mañana es sábado; mañana es sábado, por lo tanto hoy es viernes.

Frases como “por lo tanto”, “así que”, “luego entonces”, “de forma que”, entre otras, señalan la conclusión del argumento.

La representación formal del argumento es:

$$\frac{v \rightarrow s \quad s}{\therefore v}$$

De manera que el argumento es correcto si y sólo si $\models (v \rightarrow s) \wedge s \rightarrow v$. La tabla de verdad es:

v	s	$(v \rightarrow s)$	\wedge	s	\rightarrow	v
1	1	1	1	1	1	1
1	0	0	0	0	1	1
0	1	1	1	1	0	0
0	0	1	0	0	1	0

El tercer renglón de la tabla muestra que la fórmula no es una tautología por lo que el argumento es incorrecto.

Ejemplo 2.5. Mostrar la correctud del siguiente argumento:

$$\frac{p \wedge q \rightarrow r \quad p}{\therefore q \rightarrow r}$$

La tabla de verdad de la fórmula asociada al argumento es:

p	q	r	$(p \wedge q \rightarrow r)$	\wedge	p	\rightarrow	$(q \rightarrow r)$
1	1	1	1	1	1	1	1
1	1	0	0	0	1	1	0
1	0	1	1	1	1	1	1
1	0	0	1	1	1	1	1
0	1	1	1	0	0	1	1
0	1	0	1	0	0	1	0
0	0	1	1	0	0	1	1
0	0	0	1	0	0	1	1

Por lo que $\models ((p \wedge q) \rightarrow r) \wedge p \rightarrow (q \rightarrow r)$ y el argumento es correcto.

El ejemplo anterior deja ver que el método de tablas de verdad para mostrar la correctud de un argumento puede resultar complicado al crecer el número de variables involucradas en el mismo. Por esta razón resulta mandatorio buscar métodos alternativos, cosa que haremos más adelante.

Ejercicios

2.1.1.- ¿Cuáles de las siguientes oraciones son proposiciones atómicas, cuáles proposiciones no atómicas y cuáles no son proposiciones? Justifica tu respuesta.

- a) El cielo está nublado
- b) Por favor ven a verme

c) $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

d) $0 \leq x \leq 10$

e) Juan y Pedro van al cine

f) Estoy a dieta porque es necesario para bajar de peso

2.1.2.- Expresa los siguientes enunciados en el lenguaje de la lógica proposicional:

- a) Un triángulo equilátero tiene sus tres ángulos iguales.
- b) Siempre que come fresas le da alergia.
- c) $0 \leq x \leq y \leq 15$
- d) Todo número par es divisible entre 2.
- e) Para que vayas al cine tienes que hacer tu tarea.

2.1.3.- Usa variables proposicionales p , q y r para formalizar los siguientes argumentos lógicos. Lista cómo asignas las variables a las proposiciones atómicas.

- a) Si hay exámenes cada semana, los estudiantes se quejan; y si no hay exámenes cada semana, los estudiantes se quejan; de cualquier forma los estudiantes se quejan.
- b) Si n es número primo, no puede ser divisible entre 2; sabemos que 24 es divisible entre 2, por lo que no es número primo.
- c) Si lo mató, fue un crimen pasional; y si es un crimen pasional, el asesino sale corriendo; sabemos que ella no salió corriendo; entonces no lo mató.
- d) No hay otra manera de pasar la materia más que estudiando.
- e) Hay que llegar temprano para agarrar buen lugar.

2.1.4.- Usando las variables proposicionales ℓ y s para denotar a las proposiciones atómicas *Juan es muy listo* y *Juan está satisfecho* respectivamente, denota con estas variables proposicionales y los conectivos lógicos a las siguientes proposiciones:

- a) Juan es muy listo y está satisfecho.
- b) Si Juan no fuera listo, no estaría satisfecho.
- c) Juan es listo o está satisfecho.
- d) Juan está satisfecho únicamente si es listo.
- e) Si Juan es listo entonces está satisfecho.
- f) Juan es listo pero no está satisfecho.

2.1.5.- En los siguientes enunciados, identifica las proposiciones atómicas y asígnale variables proposicionales. Una vez hecho esto, convierte los enunciados a proposiciones lógicas.

- a) Si Juan fue al cine, seguro que Lupe fue también.
- b) Las noticias no son buenas.
- c) Te darán clave para la red sólo si estás inscrito en el curso.
- d) Si asistió a las clases, debió pasar la materia.
- e) El asesino era de tez blanca o clara.

2.1.6.- Formaliza las siguientes implicaciones y construye sus implicaciones asociadas.

- a) Si un número es divisible entre 2 entonces es par.
- b) Si Elke es austriaca entonces es europea.

- c) Una condición necesaria para que Lourdes lleve el curso de algoritmos es que apruebe matemáticas discretas.
- d) El programa es legible sólo si está bien estructurado.
- e) La audiencia dormirá si el ponente diserta sobre lógica medieval.

2.1.7.- Para el siguiente enunciado, asigna variables proposicionales a las proposiciones atómicas y escribe la proposición completa usando esas variables proposicionales.

- (a) María fue al teatro el lunes en la noche sólo en el caso de que no tuviera clase el martes temprano.
- (b) Si Juan llevó su Mustang al desfile es porque le cambió el amortiguador el día anterior.
- (c) Si los tres lados de un triángulo son congruentes, entonces los tres ángulos del triángulo son congruentes.
- (d) Si x es mayor que 3 entonces también es mayor que 2.
- (e) Nunca ha nevado en Cuernavaca.
- (f) Si n es un entero, entonces $n^3 - n$ es par.

2.1.8.- Para cada pareja de enunciados que se listan, escribe las fórmulas para la disyunción de ambos y la conjunción de ambos. Para cada fórmula, indica si es verdadera o no.

- (a) p : Uno es un entero par q : Nueve es un entero positivo
- (b) p : Chihuahua está en la frontera con EEUU q : Brasil está en África
- (c) p : La naranja es una fruta q : La papa es una verdura
- (d) p : Los pájaros tienen cuatro patas q : Los conejos vuelan
- (e) p : Los cardenales son rojos q : Los ruiséñores son azules

2.1.9.- Para cada uno de los siguientes enunciados, asigna variables proposicionales y escribe la fórmula o argumento lógico correspondiente al enunciado.

- (a) Si hoy es viernes, iré al cine.
- (b) Si termino la tarea voy a tomar un descanso.
- (c) Si Pepito compite en natación va a ganar el primer lugar. Si Juanito compite en natación va a ganar el primer lugar. Alguno de los dos no va a quedar en primer lugar en la competencia de natación. Por lo tanto o Pepito no compite o Juanito no compite.
- (d) Los perros son mamíferos. Los mamíferos no tienen agallas. Por lo tanto los perros no tienen agallas.

(e) Voy a comer tacos o quesadillas. Decidí no comer quesadillas. Entonces comeré tacos.

2.1.10.- Elabora la tabla de verdad para el operador $\boxed{\text{nand}}$, donde $p \boxed{\text{nand}} q$ está definido como $\neg (p \wedge q)$.

2.1.11.- Elabora las tablas de verdad para $p \wedge p$, $p \vee \neg p$, $p \vee p$, $p \wedge \text{true}$, $p \wedge \text{false}$, $p \vee \text{true}$ y $p \vee \text{false}$. Observa cada una de estas tablas de verdad y di la relación que tienen con la variable p original.

2.1.12.- Construye la tabla de verdad para cada una de las siguientes fórmulas, clasificando si se trata de una tautología, contradicción o contingencia.

a) $(p \wedge q) \rightarrow \neg(r \wedge q)$

b) $(p \wedge (r \wedge q)) \rightarrow r$

c) $((p \rightarrow q) \wedge \neg q) \rightarrow p$

d) $(s \vee t) \leftrightarrow (s \wedge t)$

e) $(r \rightarrow s) \wedge \neg t$

f) $(q \vee p) \rightarrow (\neg p \rightarrow q)$

2.1.13.- Analizar mediante tablas de verdad la correctud de los siguientes argumentos.

a) $p, q / \therefore p \wedge q$

b) $q, r, s / \therefore q \wedge t$

c) $p \rightarrow q, \neg q / \therefore \neg p$

d) $p \rightarrow q \vee r, \neg q / \therefore p \rightarrow r$

e) $p \rightarrow q, \neg p / \therefore \neg q$

2.2. Evaluación de expresiones

Si bien el proceso de evaluación de una expresión nos queda intuitivamente claro y en muchos casos es un proceso mental completamente automático, vamos a formalizarlo en esta sección. El objetivo de esta formalización radica principalmente en la necesidad de la misma para un estudio en abstracto de la evaluación y sus propiedades mediante el cual se podrá automatizar el proceso de evaluación más fácilmente.

2.2.1. Estados y evaluación

Regresamos al concepto de *estado* que, como dijimos en la sección anterior, está íntimamente relacionado con la evaluación de expresiones (o, en nuestro caso, de proposiciones).

Definición 2.4 Un **estado** es una función que asigna a una variable dada x un valor v elegido de entre aquellos que pueden asignarse a esa variable. Un estado se representa usualmente mediante un conjunto de parejas ordenadas (x, v) (o bien $x = v$), donde en cada pareja el primer elemento x es una variable y el segundo elemento v es un valor.

Definición 2.5 La *evaluación de una expresión E en un cierto estado* se logra reemplazando todas las variables en E por los valores que éstas tienen en ese estado y calculando después el valor de la expresión resultante, dictada por el significado de los operadores que figuran en la expresión.

Esta definición, si bien debe ser intuitivamente clara, no es completamente formal; más adelante definiremos formalmente qué significa reemplazar una variable por un valor o una expresión. Veamos algunos ejemplos en la siguiente página.

Expresión	Estado	Evaluación
$m \div n$	$\{ m = 63, n = 7 \}$	9
$m \div n$	$\{ m = 8, n = 48 \}$	$\frac{1}{6}$
$i = 1$	$\{ i = 2, j = 1 \}$	0
$i = 1$	$\{ i = 1 \}$	1
$i = 1$	$\{ j = 1 \}$	$i = 1$
$a + (b \cdot c)$	$\{ a = 3, b = 5, c = 2 \}$	13
$a + (b \cdot c)$	$\{ a = 4, b = 5, c = 7, d = 8 \}$	39
$a + (b \cdot c)$	$\{ a = 13, b = 11, d = 2 \}$	$13 + (11 \cdot c)$
$(p \wedge q) \vee r$	$\{ p = 0, q = 1, r = 1 \}$	1
$(p \wedge q) \vee r$	$\{ p = 1, q = 0, r = 0 \}$	0
$(p \rightarrow q) \rightarrow r$	$\{ p = 0, q = 0, r = 0 \}$	0
$(p \rightarrow q) \rightarrow r$	$\{ p = 1, q = 0, r = 0 \}$	1

Si sucede que hay variables en la expresión que no aparecen en el estado (es decir, que no tienen un valor asignado), entonces la evaluación de la expresión incluirá presencias de esas variables a las que no les podemos asignar valor (quedarán con *incógnitas*, como les hemos llamado a este tipo de variables). En estos casos lo más común es que la expresión obtenida mediante esta evaluación parcial interactúe más adelante con otro estado para terminar su evaluación. Sin embargo, en algunos casos se puede evaluar completamente una expresión aun cuando el valor de alguna de sus variables no esté definido en el estado. Estos casos son aquellos en los que el valor de la expresión no depende de dicha variable. Por ejemplo, si llegamos a una expresión como $0 \cdot (a + b)$ es irrelevante el valor ya sea de a o de b , porque esta expresión se evalúa a 0 (cero); lo mismo para las expresiones $0 \rightarrow p$ o bien $p \rightarrow 1$, pues sabemos que el resultado de ambas expresiones es verdadero (1). Es útil, entonces, para ahorrarnos algo de trabajo, conocer las propiedades de los operadores y de algunas expresiones en las que están involucradas constantes.

2.2.2. Precedencia y asociatividad

Hemos utilizado paréntesis en expresiones. Los paréntesis nos indican *agregación*. Por ejemplo, en la expresión $3 + (4 \cdot 5)$ los paréntesis agregan la expresión $4 \cdot 5$ como el segundo operando de la suma para indicar que la operación que queremos realizar es la suma de 3 con el producto de 4 y 5, cuyo resultado es 23. Si la expresión tuviera los paréntesis $(3 + 4) \cdot 5$, se estaría agregando la suma de 3 y 4 como operando del producto, dando como resultado 35. Para reducir el número de paréntesis en una expresión se asignan *precedencias* a los operadores. En particular, los lenguajes de programación hacen esto, pues el uso excesivo de paréntesis resulta ser una carga para el programador y obscurece el significado de la expresión para el lector humano. Si el operador op_1 tiene mayor precedencia que el operador op_2 , eso quiere decir que primero evaluamos la operación de op_1 y después la de op_2 . Por ejemplo, como usualmente la multiplicación tiene mayor precedencia que la suma, en la expresión $3 + 4 \cdot 7$ se debe evaluar primero el producto $4 \cdot 7$, y ese resultado usarlo para la suma con 3. En otras palabras, es como si los paréntesis aparecieran alrededor del producto, $3 + (4 \cdot 7)$ y, de hecho, una vez definido el orden de precedencia, es posible restaurar los paréntesis originales siguiendo este orden.

Otro concepto, que se relaciona en particular con el orden de evaluación de una expresión, es el de *asociatividad*. Esta propiedad nos permite decidir, si tenemos al mismo operador más de una vez en una expresión y en ausencia de paréntesis para indicar el orden de evaluación, cuál de las presencias del operador debe evaluarse primero. Por ejemplo, en la expresión $p \rightarrow q \rightarrow r$, ¿cuál de los dos debe evaluarse primero, el de la izquierda o el de la derecha? El resultado de la evaluación es distinta, dependiendo de la asociatividad que tengamos:

p	q	r	$(p \rightarrow q) \rightarrow r$	Valor	$p \rightarrow (q \rightarrow r)$	Valor
0	0	0	$1 \rightarrow 0$	0	$0 \rightarrow 1$	1

Como se puede ver, tanto la precedencia como la asociatividad determinan, en ausencia de paréntesis, el orden de evaluación de las subexpresiones. Los paréntesis se usan, como ya dijimos, para alterar la precedencia y asociatividad natural o bien para que quede explícita la precedencia y asociatividad que deseamos. A continuación damos una tabla de precedencias y asociatividades de los operadores aritméticos y lógicos más comunes. En el orden en que aparecen, la precedencia va de mayor a menor. Los operadores que tienen la misma precedencia aparecen en el mismo renglón de la tabla.

Operador	Descripción	Asociatividad
$+$ $-$ \neg	operadores unarios prefijos	izquierda
$**$	exponenciación	derecha
\cdot $/$ \div mod gcd	producto, división, módulo y máximo común divisor	izquierda
$+$ $-$	suma y resta binarias	izquierda
$=$ $<$ $>$	comparadores	izquierda
\wedge \vee	conjunción y disyunción	izquierda
\rightarrow	implicación	derecha
\leftrightarrow	bicondicional	izquierda

Como podemos observar de la tabla anterior, en ausencia de paréntesis la evaluación de $p \rightarrow q \rightarrow r$ debe realizarse asociando $p \rightarrow (q \rightarrow r)$, ya que el operador \rightarrow asocia a la derecha. Esto quiere decir que evaluamos de derecha a izquierda, como si hubiera paréntesis alrededor de $q \rightarrow r$. En el caso de la expresión $3 + 4 \cdot 7$, la precedencia de \cdot es mayor que la de $+$ binario, por lo que se evalúa a 31 ($3 + (4 \cdot 7)$). Sin embargo, esta tabla no nos ayuda a determinar los paréntesis implícitos en expresiones como $P \wedge Q \vee R$, ya que \wedge y \vee tienen la misma precedencia, pero no son el mismo operador, por lo que no podemos utilizar la asociatividad para dirimir el conflicto. En este tipo de expresiones es costumbre poner **siempre** paréntesis para indicar la precedencia, ya que de otra manera la evaluación de la expresión es ambigua. Por lo tanto debemos escribir $(P \wedge Q) \vee R$ o bien $P \wedge (Q \vee R)$, dependiendo de cuál es la precedencia deseada.

Puede haber estados en los que la evaluación sea la misma. Veamos la evaluación de estas dos asociatividades en un estado en el que no se obtiene el mismo valor, para corroborar que en ese estado no producen el mismo resultado y por lo tanto las dos expresiones no son equivalentes.

P	Q	R	$(P \wedge Q) \vee R$	valor	$P \wedge (Q \vee R)$	valor
0	0	1	0 1 1	1	0 0 1	0

Insistimos en que el concepto de asociatividad sólo se puede aplicar cuando se trata de dos o más presencias *consecutivas* del mismo operador; no son los niveles de precedencia los que definen la asociatividad.

2.2.3. Sustitución textual

Supongamos que tenemos dos expresiones² E y R , y sea x una variable (usualmente presente en E). Usamos la notación $E[x := R]$ o E_R^x para denotar la expresión que es la misma que E , pero donde cada presencia (*ocurrencia*) de x en la expresión E ha sido sustituida por la expresión (R) . Llamamos *sustitución textual* al acto de sustituir todas las presencias de x en E por (R) . Cuál de las dos notaciones utilizar no es relevante, excepto que se debe elegir una de ellas y mantener esa elección. La notación $E[x := R]$ es más apropiada para computación, pero la notación E_R^x es la utilizada por los profesionales de la lógica matemática.

Tabla 2.4 Ejemplos de sustitución textual

(1/2)

Expresado como $E[x := R]$	Expresado como E_R^x	Resultado
1. $a + b[a := x + y]$	$a + b_{x+y}^a$	$a + b$
2. $(a + b)[a := x + y]$	$(a + b)_{x+y}^x$	$((x + y) + b)$
3. $(x \cdot y)[x := z + 2]$	$(x \cdot y)_{z+2}^x$	$((z + 2) \cdot y)$

²Nos referimos a expresiones de cualquier tipo.

Tabla 2.4 Ejemplos de sustitución textual

(2/2)

Expresado como $E[x := R]$	Expresado como E_R^x	Resultado
4. $(4 \cdot a \cdot b)[a := b]$	$(4 \cdot a \cdot b)_b^a$	$(4 \cdot (b) \cdot b)$
5. $(p \rightarrow q)[p := 0]$	$(p \rightarrow q)_0^p$	$((0) \rightarrow q)$
6. $(p \rightarrow p \vee q)[p := p \vee q]$	$(p \rightarrow p \vee q)_{p \vee q}^p$	$((p \vee q) \rightarrow (p \vee q) \vee q)$
7. $(5 \cdot x)[x := 2 + 6]$	$(5 \cdot x)_{2+6}^x$	$(5 \cdot (2 + 6))$

Es conveniente notar que la sustitución textual es una operación y podemos considerar a $[x := R]$, o bien $_R^x$, como el operador. En este caso no es una operación de números a números como la suma y el producto, o de proposiciones a proposiciones como la negación o conjunción, sino que se trata de una operación de expresiones cualesquiera en expresiones cualesquiera. A este operador se le asigna la precedencia más alta de todos los operadores y su asociatividad es a la izquierda. Esto debe tomarse en cuenta cuando veamos a cuál expresión es a la que afecta la sustitución: no es lo mismo $a + b[a := x + y]$ ($a + b_{x+y}^a$) que $(a + b)[a := x + y]$ ($(a + b)_{x+y}^a$), pues en el primer caso la única expresión a la que se refiere la sustitución es b , mientras que en el segundo caso es $(a + b)$. En ambos casos, y dado que la sustitución textual es lo primero que se va a ejecutar, toma como operando al grupo que se encuentra a su izquierda, que en el primer caso consiste únicamente de b mientras que en el segundo caso, dado que se usaron paréntesis, consiste de $(a + b)$.

Podemos ver algunos ejemplos en la tabla 2.4 de la página anterior, utilizando ambas notaciones por el momento, aunque después usaremos la que indicamos como la más adecuada para computación.

Deseamos hacer hincapié sobre los siguientes puntos:

- Debe quedar claro el porqué la sustitución se define poniendo entre paréntesis a R dentro de E : si no lo hiciésemos así correríamos el riesgo de alterar los paréntesis implícitos de la expresión. En la sustitución 7 del ejemplo, si evaluamos la expresión resultante obtenemos 40, pero si no pusiéramos los paréntesis alrededor de $2 + 6$, la expresión se evaluaría a 16, de acuerdo con la precedencia de los operadores en la expresión resultante.
- R , la expresión por la que vamos a sustituir, puede o no tener presencias de x , la variable a la que vamos a sustituir.
- Si E no tiene ninguna presencia de x , la expresión queda exactamente igual a como estaba, es decir $E[x := R] = E$.

- Si hay varias presencias de x en E , como es el caso del ejemplo 6, se piensa en la sustitución hecha *simultáneamente* a cada presencia de x en E . Es como si marcáramos las posiciones de x en E , después ponemos una caja en lugar de la variable y después colocamos en esas cajas a (R) . Si es que x aparece en R , no regresamos a sustituir estas presencias de x en el resultado.
- Es común que en el resultado queden paréntesis que no aportan nada, por ejemplo aquellos que rodean a una variable sola. En este caso, y cuando la eliminación de los paréntesis no afecte la precedencia y asociatividad del resultado, éstos pueden eliminarse. Esto también se refiere a los paréntesis que utilizamos para rodear a la expresión sobre la que queremos hacer la sustitución. En adelante mantendremos los paréntesis sólo en aquellos casos en que sean estrictamente necesarios, es decir, cuando quitarlos altere la precedencia y asociatividad de la expresión resultante.

Si tenemos una lista de variables $\mathbf{x} : x_1, x_2, \dots, x_n$ distintas y una lista de expresiones $\mathbf{R} : R_1, R_2, \dots, R_n$ (no forzosamente distintas), podemos definir la sustitución textual simultánea $E[\mathbf{x} := \mathbf{R}]$ ($E_{\mathbf{R}}^{\mathbf{x}}$) como el reemplazo simultáneo de cada una de las variables de la lista \mathbf{x} por su correspondiente expresión en la lista \mathbf{R} . Esto es, x_1 se reemplaza con R_1 , x_2 con R_2 , y así sucesivamente. Por ejemplo, $(p \wedge q)[p, q := 1, 0]$ es $((1) \wedge (0))$, cuyo valor es 0, mientras que $(p \wedge q)[p, q := 1, p]$ es $(1 \wedge p)$, ya que no se puede “regresar” a hacer la sustitución textual de la variable x que aparece en la expresión R , en la expresión resultante.

Un punto mucho muy importante a notar es que la sustitución textual se utiliza únicamente para sustituir presencias de variables, **no** de expresiones ni de constantes.

Como ya mencionamos, la asociatividad de la sustitución textual es izquierda, por lo que $E[x := R][z := S]$ se asocia $(E[x := R])[z := S]$, donde E , R y S son expresiones y x y z son variables; esta operación se define como una copia de E en la que las presencias de x fueron sustituidas por R , y en esa copia las presencias de z fueron sustituidas por S . Es importante notar que, en general, $E[x := R][z := S]$ es distinto a $E[x, z := R, S]$, como se puede ver en las siguientes sustituciones:

$$\begin{aligned} (p \rightarrow p \vee q)[p := q][q := p] & \text{ es } p \rightarrow p \vee p \\ (p \rightarrow p \vee q)[p, q := q, p] & \text{ es } q \rightarrow q \vee p \end{aligned}$$

Variables escondidas en la sustitución textual

Es usual asignar una variable a una expresión para que sea más sencillo manipularla. Por ejemplo, podemos decidir

$$Q : \frac{-b + \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

y utilizar esta asociación para, en lugar de escribir

$$x = (-b + \sqrt{b^2 - 4 \cdot a \cdot b}) / (2 \cdot a)$$

podamos escribir $x = Q$. Pero entonces Q tiene tres variables escondidas, a , b y c , y una sustitución de la forma $Q[a := 3]$ se debe interpretar como

$$\left((-b + \sqrt{b^2 - 4 \cdot a \cdot b}) / (2 \cdot a) \right) [a := 3]$$

cuyo resultado es $(-b + \sqrt{b^2 - 4 \cdot 3 \cdot b}) / (2 \cdot 3) = (-b + \sqrt{b^2 - 12 \cdot b}) / 6$

Queremos hacer notar que la evaluación de una expresión consiste en, simplemente, hacer una sustitución textual en la expresión, donde por cada variable definida en el estado en que se desea evaluar a la expresión se le sustituye por el valor de la variable en ese estado. Después, si es posible, se ejecutan las operaciones necesarias para obtener el valor de la expresión en ese estado.

Ejercicios

2.2.1.- Coloca los paréntesis en las siguientes expresiones de acuerdo a la precedencia y asociatividad de los operadores, sin preocuparte por la evaluación de la expresión:

a) $-b + b * 2 - 4 \cdot a \cdot c / 2 \cdot a$

b) $p \wedge q \vee r \rightarrow s \leftrightarrow p \vee q$

c) $a < b \wedge b < c \rightarrow a < b$

d) $a \cdot b < a \cdot c \leftrightarrow a > 0 \wedge b > c$

2.2.2.- Para cada expresión que se da a continuación, evalúa la expresión en cada uno de los estados que se proporcionan:

Expresión	Estados	Evaluación
a) $a^2 + (b \cdot c)$	$\{a = 5, b = 3, c = 6\}$	
	$\{a = -2, b = 1, c = 11, d = 3\}$	
	$\{d = 3, b = 4, c = 10\}$	
	$\{a = 3, b = 0\}$	
b) $p \rightarrow q \leftrightarrow q \rightarrow r$	$\{p = 1, q = 0, r = 1\}$	
	$\{p = 0, r = 1\}$	
	$\{p = 1, r = 0\}$	
	$\{p = 1, q = 1, r = 1\}$	

2.2.3.- Ejecuta las siguientes sustituciones textuales, fijándote bien en la colocación de los paréntesis. Quita los paréntesis que no sean necesarios.

- a) $x[x := b + 2]$
- b) $(x + y \cdot x)[x := b + 2]$
- c) $(x + x \cdot 2)[y := x \cdot y]$
- d) $(x + x \cdot y + x \cdot y \cdot z)[x := x + y]$

2.2.4.- Ejecuta las siguientes sustituciones textuales simultáneas, fijándote bien en la colocación de los paréntesis. Quita los paréntesis que no sean necesarios.

- a) $x + y \cdot x[x, y := b + 2, x + 2]$
- b) $(x + y \cdot x)[x, y := x \cdot y, x \cdot y]$
- c) $(x + y \cdot 2)[y, x := x \cdot y, x \cdot x]$
- d) $(x + x \cdot y + x \cdot y \cdot z)[x, y := y, x]$

2.2.5.- Ejecuta las siguientes sustituciones textuales, fijándote bien en la colocación de los paréntesis. Quita los paréntesis que no sean necesarios.

- a) $x + y \cdot x[x := y + 2][y := y \cdot x]$
- b) $(x + y \cdot x)[x := y + 2][y := y \cdot x]$
- c) $(x + x \cdot 2)[x, y := x, z][x := y]$
- d) $(x + x \cdot y + x \cdot y \cdot z)[x, y := y, x][y := 2 \cdot y]$

2.2.6.- Expresa la evaluación de las expresiones en la pregunta 2.2.2 utilizando sustitución textual simultánea.

2.3. Análisis sintáctico de expresiones lógicas

En general, una expresión es una cadena o palabra construida mediante símbolos de un alfabeto dado. Sin embargo no todas las cadenas que construyamos simplemente pegando símbolos van a ser expresiones útiles, sino únicamente aquellas construidas de acuerdo a una gramática diseñada con ese propósito particular. El proceso de evaluación descrito anteriormente requiere que la expresión a evaluar sea sintácticamente válida; por ejemplo, no podemos ni debemos intentar evaluar una cadena de símbolos como $p \neg q$, puesto que ésta no es una expresión válida y el intento de evaluarla fracasará.

En nuestro caso a las expresiones generadas de manera legítima por la gramática de la lógica proposicional les llamamos *expresiones lógicas*, *proposiciones* o bien *fórmulas*. Por ejemplo, $P \wedge Q$ es una fórmula si es que garantizamos que P y Q son, a su vez, fórmulas. El proceso de evaluación de una expresión debe ser precedido por el proceso de reconocer cuándo una cadena de símbolos es una fórmula bien construida o formada; este proceso se conoce como *análisis sintáctico*. En nuestro caso particular la pregunta que nos interesa responder es ¿cuándo una cadena de símbolos es una expresión lógica? Hasta ahora la

única manera de responder es derivando dicha cadena mediante las reglas de la gramática; sin embargo, este proceso puede ser largo y tedioso, y si bien esta es la manera usual de implementar el proceso de análisis sintáctico, nos gustaría tener un proceso más simple y directo para nuestro uso. A continuación nos serviremos de la operación de sustitución textual para verificar cuándo una cadena de símbolos es una fórmula bien formada.

2.3.1. Esquemas

En matemáticas es común asociar *identificadores* a ciertas expresiones con el propósito de abreviar su escritura; podemos escribir por ejemplo A para denotar a la fórmula $p \vee q$. Sin embargo, A no es una variable proposicional, pues para obtener su valor es necesario evaluar la fórmula $p \vee q$, a partir de los valores de las variables proposicionales p y q . Un identificador es entonces una especie de variable informal, conocida entre los lógicos como *metavariante*. A continuación fijamos una definición de esquema.

Definición 2.6 Un *esquema* es una expresión construida de manera similar a las fórmulas pero usando, en algunos casos, identificadores en vez de variables proposicionales.

Si bien esta definición es informal pues el concepto de identificador no ha sido definido con precisión, con ella nos basta.

Ejemplo 2.6. Si A y B son identificadores, entonces $A \wedge B$ es un *esquema*.

Ejemplo 2.7. La expresión $(A \rightarrow B)$ es un esquema, y si $A = (p \wedge q)$ y $B = (p \vee q)$ entonces nos proporciona una forma más concisa de escribir

$$((p \wedge q) \rightarrow (p \vee q))$$

Ejemplo 2.8. Si p es una variable proposicional y $A = (p \rightarrow q)$, la fórmula $(p \wedge \neg A)$ es un esquema que proporciona una forma más concisa de escribir $(p \wedge \neg (p \rightarrow q))$.

La sustitución textual en combinación con el concepto de esquema proporcionan una manera simple para decidir si una expresión es una fórmula bien formada. Por ejemplo, ¿cómo podemos verificar si la expresión $p \wedge \neg q \rightarrow r \wedge s$ es una implicación?; basta ver que dicha fórmula se obtiene a partir del esquema de implicación $A \rightarrow B$, en el caso particular en que los identificadores se sustituyan (*instancien*) con $A = p \wedge \neg q$ y $B = r \wedge s$.

Definición 2.7 *Instanciar* un esquema consiste en hacer una sustitución textual simultánea de cero o más identificadores en el esquema, por fórmulas bien construidas, que pueden o no involucrar a identificadores que aparecen originalmente en el esquema.

Un esquema tiene tantas instancias como fórmulas bien formadas podamos usar en la sustitución textual simultánea, esto es, un número infinito de instancias. Todo esquema es una instancia de sí mismo, ya que resulta de la sustitución textual simultánea de cero identificadores en el esquema o, visto de otra manera, donde cada identificador que aparece en el esquema es sustituido por sí mismo.

Si bien existen una infinidad de esquemas, basta identificar con nombre a los siguientes, llamados *básicos*:

	Llamamos	a una expresión de la forma
1.	negación	$(\neg A)$
2.	conjunción	$(A \wedge B)$
3.	disyunción	$(A \vee B)$
4.	condicional	$(A \rightarrow B)$
5.	equivalencia	$(A \leftrightarrow B)$

Obsérvese que toda fórmula debe ser atómica, o bien corresponder a una o varias sustituciones textuales simultáneas de uno de estos cinco esquemas.

Ahora veamos ejemplos de fórmulas bien construidas. Utilizaremos paréntesis para presentar las distintas fórmulas y procederemos a comprobar que están bien construidas mediante esquemas. Haremos uso de la precedencia y asociatividad para eliminar paréntesis, cuando esto no afecte el significado de la fórmula.

Ejemplo 2.9. La expresión $((p \wedge q) \rightarrow (p \vee q))$ es una *condicional*. Para ver por qué se le asigna este nombre, veamos la sucesión de sustituciones textuales que se fueron realizando:

$$(p \rightarrow q)[p, q := p \wedge q, p \vee q] = ((p \wedge q) \rightarrow (p \vee q))$$

que quitando los paréntesis superfluos queda

$$p \wedge q \rightarrow p \vee q.$$

Como el esquema original del que partimos es el de la implicación, la instanciación dada es por ende una implicación.

Ejemplo 2.10. El esquema $\neg A \rightarrow P \vee Q$ es una condicional, porque al restaurar los paréntesis implícitos en la expresión, dada la precedencia y asociatividad de los distintos operadores

que aparecen, obtenemos $((\neg A) \rightarrow (P \vee Q))$.

$$\begin{aligned} (P \rightarrow Q)[P, Q := A, P \vee Q][A := \neg A] &= \\ &= ((A) \rightarrow (P \vee Q))[A := \neg A] \\ &= ((\neg A) \rightarrow (P \vee Q)), \end{aligned}$$

donde quitando los paréntesis superfluos, queda $\neg A \rightarrow P \vee Q$.

Como el esquema original del que partimos es una condicional, decimos que el esquema $\neg A \rightarrow P \vee Q$ también es una condicional.

Ejemplo 2.11. La fórmula $(p \leftrightarrow q) \wedge (r \leftrightarrow p) \leftrightarrow (p \leftrightarrow q) \wedge (r \leftrightarrow q)$ es una equivalencia. Nuevamente veamos los paréntesis implícitos, de acuerdo a las reglas de precedencia y asociatividad:

$$\left(((p \leftrightarrow q) \wedge (r \leftrightarrow p)) \leftrightarrow ((p \leftrightarrow q) \wedge (r \leftrightarrow q)) \right)$$

y veamos la sucesión de sustituciones textuales a partir del esquema $(A \leftrightarrow B)$.

$$\begin{aligned} (A \leftrightarrow B)[A, B := P \wedge Q, P \wedge R][P, Q, R := p \leftrightarrow q, r \leftrightarrow p, r \leftrightarrow q] &= \\ &= ((P \wedge Q) \leftrightarrow (P \wedge R))[P, Q, R := p \leftrightarrow q, r \leftrightarrow p, r \leftrightarrow q] \\ &= (((p \leftrightarrow q) \wedge (r \leftrightarrow p)) \leftrightarrow ((p \leftrightarrow q) \wedge (r \leftrightarrow q))), \end{aligned}$$

donde quitando los paréntesis superfluos, nos lleva a:

$$(p \leftrightarrow q) \wedge (r \leftrightarrow p) \leftrightarrow (p \leftrightarrow q) \wedge (r \leftrightarrow q).$$

Obsérvese que en este ejemplo primero transformamos el esquema básico de implicación en un esquema más cercano a la fórmula original, para después instanciar con las fórmulas adecuadas y obtener el resultado deseado.

Del último ejemplo podemos concluir que el proceso de análisis mediante sustituciones textuales empieza a resultar complicado, por lo que nos gustaría dar una definición del proceso, susceptible de aplicarse mecánicamente, algo que desarrollamos a continuación.

2.3.2. Rango y conectivo principal

Para mecanizar el proceso de análisis sintáctico de una expresión nos serviremos, además de la sustitución textual y el uso de esquemas, de un proceso de descomposición en expresiones sintácticamente más simples, las cuales son más sencillas de analizar. Dicha descomposición utiliza los conceptos de rango de un conectivo lógico y conectivo principal de una fórmula que a continuación definimos.

Definición 2.8 El concepto de *rango o alcance* de un conectivo en una fórmula o esquema E se define, con base en los esquemas básicos, como sigue:

- Si E es instancia de $\neg A$, entonces el rango de \neg en E es A .
- Si E es instancia de uno de los esquemas básicos binarios $A \star B$, donde \star es un conectivo lógico binario, entonces el conectivo \star en E tiene un *rango izquierdo* que es A y un *rango derecho* que es B .

Obsérvese que el rango o rangos de un conectivo (operador) en una expresión corresponden a los operandos; en caso de que no estén explícitamente indicados se obtienen tomando en cuenta las reglas de asociatividad y precedencia ya estudiadas. Por ejemplo:

- En el esquema $\neg A \wedge B$ el rango del operador \neg es únicamente el identificador A . Si queremos que el rango sea $A \wedge B$ debemos encerrar este esquema entre paréntesis, obteniendo $\neg (A \wedge B)$.
- En la fórmula $A \wedge B \wedge C$ el rango izquierdo del segundo conectivo \wedge es la fórmula $(A \wedge B)$, ya que como no hay paréntesis, las reglas de precedencia y asociatividad hacen que la colocación de los paréntesis implícita de la fórmula sea $((A \wedge B) \wedge C)$.

Otro concepto importante es el de *conectivo principal*. Si una expresión E resulta ser instancia de uno de los esquemas básicos, entonces el conectivo que observamos en el esquema correspondiente será también el conectivo principal de E . Por ejemplo, si $E = (p \vee q) \wedge C$, entonces el conectivo principal de E es \wedge , puesto que $E = (A \wedge B)[A, B := p \vee q, C]$.

Veamos un ejemplo más elaborado.

Ejemplo 2.12. Consideremos el esquema $(A \wedge B \wedge C) \vee (A \rightarrow B \rightarrow C)$. El análisis sintáctico de este esquema es el siguiente:

- Para el esquema original:
 - El conectivo principal es \vee .
 - El rango izquierdo es $(A \wedge B \wedge C)$.
 - El rango derecho es $(A \rightarrow B \rightarrow C)$.
- Para el rango izquierdo:
 - Los paréntesis implícitos son $((A \wedge B) \wedge C)$.
 - El conectivo principal es el segundo \wedge .
 - El rango izquierdo corresponde a $(A \wedge B)$.
 - El rango derecho corresponde a C .
- Para el rango derecho, podemos observar que:

- Los paréntesis implícitos son $(A \rightarrow (B \rightarrow C))$.
- El conectivo principal es el primer \rightarrow .
- El rango izquierdo es A .
- El rango derecho es $(B \rightarrow C)$.

Este proceso puede seguir hasta que ya tengamos esquemas o fórmulas que no correspondan a los esquemas básicos, es decir esquemas que consistan de un único identificador o bien variables proposicionales, en las que no tienen ningún significado los conceptos de conectivo principal o rango. Estos casos corresponden al fin del proceso de análisis sintáctico. Como toda fórmula consiste de una combinación de conectivos y proposiciones atómicas, la descomposición en rangos no puede durar para siempre.

2.3.3. Análisis de proposiciones compuestas

Existen dos clases de métodos para el análisis de una expresión, los métodos *generadores* que construyen la expresión deseada a partir de símbolos o esquemas iniciales utilizando ciertas reglas u operaciones; y los métodos *analíticos* que consisten en partir de la supuesta expresión dada y realizar un proceso de descomposición hasta llegar a expresiones básicas, donde el proceso de análisis es directo. Los métodos de gramáticas y árboles de derivación y de instanciación de esquemas básicos son generadores.

A continuación veremos un método analítico basado en la descomposición de una expresión utilizando su conectivo principal y rangos correspondientes. Haremos explícita esta descomposición usando un árbol, cuya raíz consistirá de la fórmula completa. En cada nivel que bajemos del árbol, identificaremos al conectivo principal de la fórmula y procederemos a colgar de la fórmula al conectivo y a su(s) rango(s). La idea principal es que si E es una expresión compuesta, los rangos del conectivo principal son expresiones, a las que les podemos aplicar el mismo procedimiento. Veamos un ejemplo:

Ejemplo 2.13. Si el equipo mexicano llega a cuartos de final del Mundial, todo mundo lo admirará y los jugadores se volverán ricos; pero si no llega, nada pasará.

Hagamos una asignación a variables proposicionales:

p : el equipo mexicano llega a cuartos de final

q : todo mundo admira al equipo mexicano

r : los jugadores se vuelven ricos

s : nada pasará

Hagamos la traducción a una fórmula con paréntesis:

$$((p \rightarrow (q \wedge r)) \wedge ((\neg p) \rightarrow s))$$

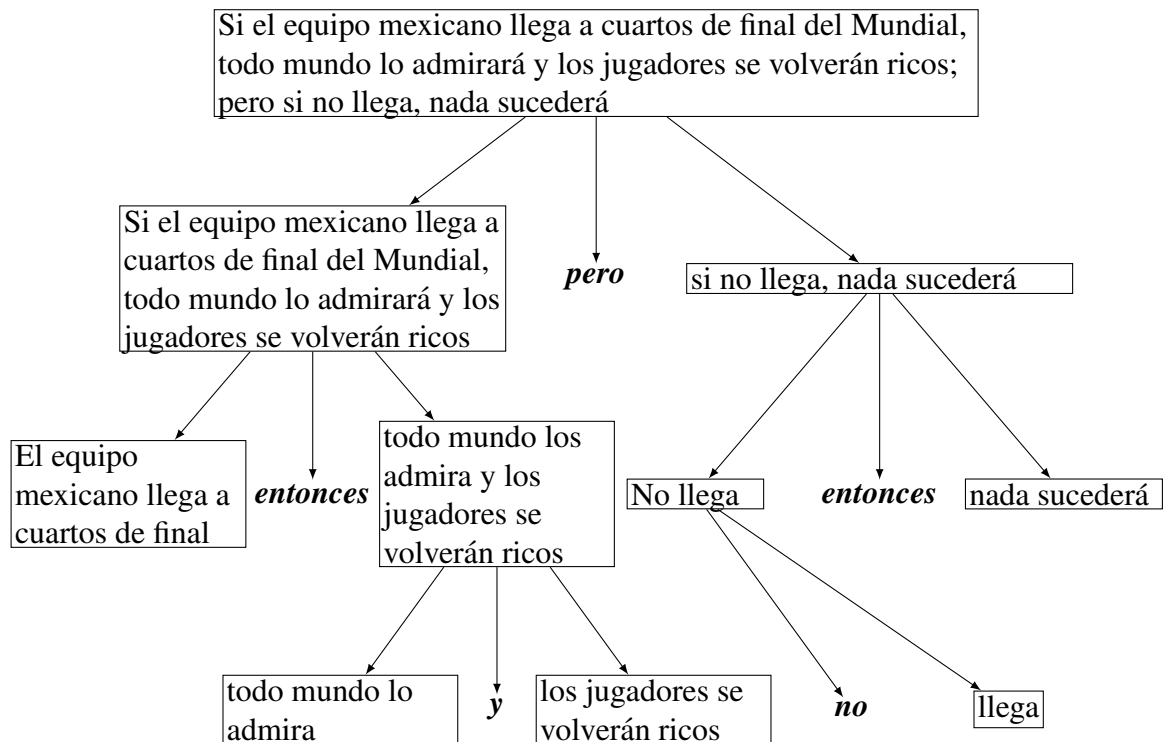
y veamos cómo queda el árbol producto del análisis sintáctico de esta fórmula en la figura 2.2 de la siguiente página.

En este caso hemos elegido presentar el árbol con las frases en español de manera que se pueda observar la descomposición directamente con enunciados más simples. Obsérvese que las hojas de este árbol corresponden a proposiciones atómicas que ya no pueden descomponerse.

El proceso de analizar una expresión es un proceso *recursivo*, que consiste de los siguientes pasos:

1. Si la proposición es atómica, el análisis termina.
2. Si la proposición no es atómica:
 - a) Definir el conectivo principal
 - b) Si el conectivo es unario, analizar la proposición que corresponde al rango derecho.
 - c) Si el conectivo es binario, analizar la proposición que corresponde al rango izquierdo y la proposición que corresponde al rango derecho.

Figura 2.2 Análisis de proposición compuesta



Veamos otro ejemplo, esta vez sin remitirnos en el árbol a las frases en español.

Ejemplo 2.14. Si el anuncio tiene éxito, toda la producción se va a vender y el dueño se volverá rico; pero si el anuncio no tiene éxito, la inversión se habrá perdido.

Variables proposicionales:

p : el anuncio tiene éxito

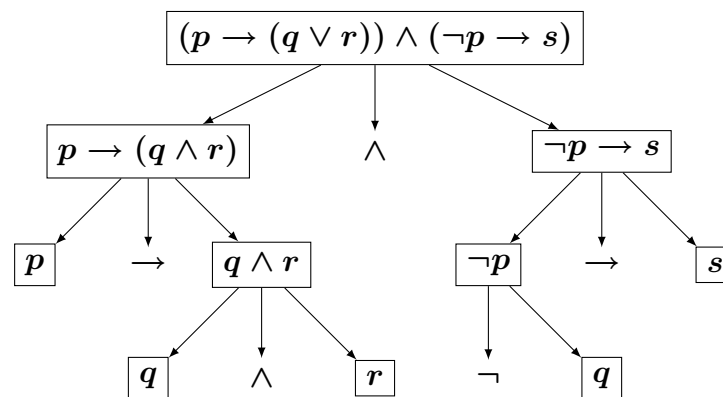
q : toda la producción se vende

r : el dueño se vuelve rico

s : la inversión se pierde

Podemos ver el árbol, usando las variables proposicionales y los conectivos lógicos, en la figura 2.3, que se encuentra en la siguiente página.

Figura 2.3 Análisis de una proposición



En este momento resulta claro que dada una expresión sintácticamente válida, se puede construir el árbol de análisis sintáctico partiendo directamente de ella; si la expresión está completamente expresada con paréntesis (todos los paréntesis que definen precedencia y asociatividad son explícitos), el proceso es inmediato, mientras que si no es así habrá que usar los criterios de precedencia y asociatividad. Los niveles del árbol se van construyendo de adentro hacia afuera (de abajo hacia arriba) y de izquierda a derecha para aquellos operadores que asocien a la izquierda, y de derecha a izquierda para aquellos que asocien a la derecha.

Más aún, el proceso de análisis sintáctico facilita el proceso de evaluación puesto que una vez construido el árbol, las hojas corresponden a fórmulas atómicas, las cuales se pueden evaluar directamente continuando el proceso de evaluación según lo dictado por las tablas de verdad de los conectivos principales.

2.3.4. Tautologías y sustitución

Hasta ahora la única manera de verificar si una fórmula dada A es una tautología es construyendo su tabla de verdad. Sin embargo, al crecer el número de variables la tabla de

verdad contiene cada vez más renglones y su construcción se vuelve complicada, ineficiente y eventualmente imposible. Como ejemplo considérese el esquema $A \wedge B \rightarrow B$. Es fácil ver mediante una tabla de verdad de cuatro renglones que $\models A \wedge B \rightarrow B$. Por otra parte considérese la expresión $p_1 \wedge p_2 \wedge \dots \wedge p_{99} \wedge p_{100} \rightarrow p_{100}$, ¿cómo mostrar que se trata de una tautología? La tabla de verdad tendrá 2^{100} renglones, así que resulta imposible construirla. Afortunadamente la operación de sustitución permite generar más tautologías a partir de tautologías conocidas.

Una vez que se conoce que $\models A$, no importa si en A sustituimos cualquier variable proposicional por una expresión, el resultado va a seguir siendo una tautología. Esto se formaliza en el siguiente teorema, cuya demostración omitimos.

Teorema 2.1 (Propiedad de sustitución) *Sea A una fórmula o esquema tal que $\models A$ y sean p_1, p_2, \dots, p_n variables proposicionales. Si B_1, B_2, \dots, B_n son expresiones lógicas o esquemas arbitrarios, entonces*

$$\models A[p_1, p_2, \dots, p_n := B_1, B_2, \dots, B_n];$$

es decir, las sustituciones textuales en tautologías generan tautologías.

Usando este resultado y observando que

$$(A \wedge B \rightarrow B)[A, B := p_1 \wedge p_2 \wedge \dots \wedge p_{99}, p_{100}] = p_1 \wedge p_2 \wedge \dots \wedge p_{100} \rightarrow p_{100}$$

concluimos que $\models p_1 \wedge p_2 \wedge \dots \wedge p_{100} \rightarrow p_{100}$.

Veamos otros ejemplos.

Ejemplo 2.15. Demostrar que $\models (p \wedge q) \vee \neg (p \wedge q)$.

Identificamos en el ejemplo una disyunción de una expresión y su negación, por lo que buscamos algún esquema tautológico que tenga esta misma forma. Sabemos que $p \vee \neg p$ es una tautología. Entonces

$$(p \vee \neg p)[p := p \wedge q] = (p \wedge q) \vee \neg (p \wedge q)$$

por lo que esta expresión es también una tautología.

Ejemplo 2.16. Demostrar que $\models R \rightarrow (P \vee Q) \vee R$.

Debemos buscar un esquema para “deshacer” las sustituciones que se hayan hecho. En el nivel más alto el esquema es

$$A \rightarrow B.$$

Busquemos ahora una tautología que involucre implicación y que en el rango derecho tenga una conjunción, sabemos que $\models p \rightarrow p \vee q$ es una tautología (mostramos su tabla de verdad

al inicio de la sección). Como la disyunción tiene la propiedad de conmutatividad, tenemos que $p \rightarrow p \vee q$ es lo mismo que $p \rightarrow q \vee p$. Por el Teorema de sustitución, tenemos:

$$(p \rightarrow q \vee p)[p, q := q, p] = q \rightarrow p \vee q.$$

Este último esquema tautológico nos sirve, pues lo que buscamos es que el rango derecho de la disyunción coincida con el rango izquierdo de la condicional.

A continuación observamos que el rango izquierdo de la disyunción es una subexpresión compuesta, no nada más una fórmula atómica, por lo que ahí también se llevó a cabo una sustitución textual, que si la “deshacemos” queda como sigue:

$$\begin{aligned} (q \rightarrow p \vee q)[q, p := R, P \vee Q] &= \\ &= (R) \rightarrow (P \vee Q) \vee (R) \\ &= R \rightarrow (P \vee Q) \vee R \end{aligned}$$

Reglas de inferencia

Una vez que se ha mostrado la correctud de un argumento lógico, éste se convierte en un esquema de argumento que sigue siendo correcto al sustituir algunos de sus identificadores por fórmulas arbitrarias, puesto que el esquema correspondiente a su fórmula asociada es una tautología, que se preserva bajo sustituciones como lo asegura el teorema 2.1. En tal caso hablamos ya no de un argumento correcto sino de una *regla de inferencia*.

Definición 2.9 Una *regla de inferencia* es un esquema de argumento correcto.

Por ejemplo, dado que los argumentos de los ejemplos 2.3 y 2.5 – el primero conocido como *modus ponens* – son correctos podemos enunciarlos como esquemas:

$$\begin{array}{c} A \rightarrow B \\ A \\ \hline B \end{array} \qquad \begin{array}{c} A \wedge B \rightarrow C \\ A \\ \hline B \rightarrow C \end{array}$$

Obsérvese que una vez que un argumento correcto se transforma en regla de inferencia, al ser correcto, el símbolo \therefore desaparece en la conclusión.

Ejercicios

2.3.1.- Clasifica a las siguientes proposiciones en alguna de las siguientes categorías, justificando la respuesta mediante el uso de esquemas:

- (a) negación
- (b) disyunción
- (c) conjunción
- (d) condicional
- (e) bicondicional

Fórmula	Categoría
$\neg P \rightarrow \neg Q$	
$P \leftrightarrow Q \leftrightarrow (P \rightarrow Q) \wedge (Q \rightarrow P)$	
$Q \wedge P \rightarrow Q \rightarrow P$	
$(P \rightarrow Q) \wedge (\neg P \rightarrow Q) \rightarrow Q$	
$P \rightarrow Q \leftrightarrow \neg Q \rightarrow \neg P$	

2.3.2.- Para las siguientes proposiciones, di a cuál esquema básico corresponden, rehaciendo las sustituciones textuales que se hayan llevado a cabo. En caso de ambigüedad respecto a la asociatividad de dos operadores distintos con la misma precedencia, se debe asociar desde la izquierda.

- (a) $p \rightarrow q \wedge q \rightarrow p$
- (b) $r \wedge \neg q \leftrightarrow \neg r \wedge q$
- (c) $p \rightarrow q \rightarrow r$
- (d) $p \vee q \wedge \neg p \wedge q \rightarrow q \wedge \neg q$
- (e) $\neg (p \vee \neg q \wedge p)$
- (f) $\neg p \rightarrow q$
- (g) $\neg (p \wedge \neg q)$
- (h) $\neg p \wedge (\neg p \wedge q) \vee (p \wedge (p \wedge \neg q))$

2.3.3.- De los siguientes enunciados, define el conectivo principal. Para cada operador: si el operador es binario especifica su rango izquierdo y su rango derecho; si el operador es unario, especifica su rango (derecho).

- (a) $p \vee (\neg p \wedge q) \rightarrow p \vee q$
- (b) $\neg (p \wedge q \rightarrow p \vee q)$

- (c) $\neg p \wedge (q \vee p) \wedge \neg q$
 (d) $(p \rightarrow q) \rightarrow p \vee q \rightarrow q$
 (e) $\neg(p \vee q \rightarrow r)$

2.3.4.- Da el árbol de análisis sintáctico de cada una de los siguientes esquemas:

- a) $P \wedge Q \wedge R \rightarrow P$
 b) $P \rightarrow Q \leftrightarrow \neg Q \rightarrow \neg P$
 c) $P \rightarrow Q \rightarrow R \vee S \vee P$
 d) $P \rightarrow Q \wedge R \rightarrow S \rightarrow P \rightarrow S$

2.3.5.- Construye el árbol de análisis sintáctico para cada una de las siguientes fórmulas

- a) $\neg\neg p \wedge \neg q \rightarrow s \leftrightarrow \neg s \rightarrow \neg p \vee q$
 b) $\neg p \vee q \rightarrow p \wedge \neg q \rightarrow \neg p \vee \neg q \rightarrow \neg p \wedge \neg q$
 c) $p \wedge q \rightarrow p \vee q \rightarrow \neg p \wedge q$

2.4. Equivalencia lógica

El concepto de expresiones equivalentes es imprescindible para todo tipo de razonamiento. Decimos que dos expresiones son equivalentes si y sólo si en todos y cada uno de sus posibles estados se evalúan a lo mismo.

Por ejemplo, podemos comprobar usando una tabla de verdad, que las expresiones $\neg\neg P$ y P son equivalentes:

P	$\neg P$	$\neg(\neg P)$
1	0	1
0	1	0

Lo que debemos observar es que, *renglón por renglón*, el valor correspondiente a P es el mismo que el valor correspondiente a $\neg\neg P$. No se interprete esta definición como que estamos exigiendo tener el mismo valor en todos los renglones, esto es, que todos los renglones valieran 0 o todos los renglones valieran 1.

En el caso de expresiones lógicas el concepto de equivalencia está relacionado con un tipo particular de tautología. Si tenemos una bicondicional ($A \leftrightarrow B$) que es una tautología, entonces decimos que tenemos una *equivalencia lógica* :

Definición 2.10 (Equivalencia lógica) Sean A, B dos fórmulas. Si $A \leftrightarrow B$ es una tautología, entonces decimos que A y B son *lógicamente equivalentes* y lo denotamos por $A \equiv B$. Esto es lo mismo que decir

$$A \equiv B \quad \text{si y sólo si} \quad \models A \leftrightarrow B.$$

La tabla 2.5 resume algunas equivalencias lógicas de importancia, las cuales pueden comprobarse mediante el uso de tablas de verdad.

Tabla 2.5 Leyes de equivalencia de la lógica proposicional

Asociatividad:	$(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$	(2.14)
	$(P \vee Q) \vee R \equiv P \vee (Q \vee R)$	(2.15)
Identidad:	$P \vee \text{false} \equiv P$	(2.16)
	$P \wedge \text{true} \equiv P$	(2.17)
Idempotencia:	$P \vee P \equiv P$	(2.18)
	$P \wedge P \equiv P$	(2.19)
Dominación	$P \vee \text{true} \equiv \text{true}$	(2.20)
(o elemento nulo):	$P \wedge \text{false} \equiv \text{false}$	(2.21)
Conmutatividad:	$P \vee Q \equiv Q \vee P$	(2.22)
	$P \wedge Q \equiv Q \wedge P$	(2.23)
Tercero excluido:	$P \vee \neg P \equiv \text{true}$	(2.24)
Contradicción:	$P \wedge \neg P \equiv \text{false}$	(2.25)
Doble negación:	$\neg \neg P \equiv P$	(2.26)
Distributividad:	$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$	(2.27)
	$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$	(2.28)
De Morgan:	$\neg (P \wedge Q) \equiv \neg P \vee \neg Q$	(2.29)
	$\neg (P \vee Q) \equiv \neg P \wedge \neg Q$	(2.30)
Eliminación de operadores:	$P \rightarrow Q \equiv \neg P \vee Q$	(2.31)
	$P \leftrightarrow Q \equiv (\neg P \vee Q) \wedge (P \vee \neg Q)$	(2.32)
	$P \leftrightarrow Q \equiv (P \wedge Q) \vee (\neg P \wedge \neg Q)$	(2.33)
	$P \leftrightarrow Q \equiv (P \rightarrow Q) \wedge (Q \rightarrow P)$	(2.34)

A continuación mostraremos el uso de equivalencias lógicas en particular como herramienta auxiliar imprescindible en el análisis de un argumento lógico.

2.4.1. Razonamiento ecuacional

Consideremos la igualdad aritmética $x + y + x + z = y + 2x + z$. Probablemente ninguno de nosotros dudaría de su validez, debido a la experiencia con números que tenemos desde nuestra educación básica. Más aún, si se nos pidiera una demostración formal tal vez daríamos la siguiente:

$$x + y + x + z = y + x + x + z = y + 2x + z;$$

y si se nos pidiera nuevamente una justificación tal vez apelaríamos a las igualdades

$$x + y = y + x \quad \text{y} \quad x + x = 2x.$$

Este tipo de razonamiento se conoce como razonamiento ecuacional y será parte importante del proceso de análisis de un argumento lógico.

Por lo general las fases de razonamiento ecuacional nos son tan familiares que no se mencionan explícitamente dentro del análisis de un argumento; de hecho, nosotros respetaremos esta costumbre. Sin embargo, en nuestro curso nos conciernen no sólo los aspectos puramente matemáticos de un tema, sino también el proceso de implementación, el cual es esencialmente sintáctico dado que las computadoras no entienden de significados ni son capaces de razonar como nosotros. A continuación discutimos las propiedades de la igualdad, en particular la llamada regla de Leibniz que involucra a la sustitución textual, y que nos brindará una manera posible de implementar el razonamiento ecuacional.

Si consideramos a la igualdad como un operador (cuyo resultado es 0 o 1), podemos observar que tiene las siguientes propiedades:

Reflexividad	$X = X$
Conmutatividad	$X = Y$
	$Y = X$
Transitividad	$X = Y$
	$Y = Z$
	$X = Z$

Las últimas dos propiedades las dimos como reglas de inferencia, puesto que corresponden a argumentos correctos. Finalmente, veamos una propiedad, conocida con el nombre de regla de Leibniz, que nos va a permitir sustituir expresiones iguales en expresiones que

resultarán iguales nuevamente y proporciona una manera de implementar nuestro razonamiento ecuacional usual.

$$\frac{X = Y}{E[z := X] = E[z := Y]} \quad \text{Leibniz}$$

Lo que esta regla de inferencia nos dice es que si suponemos que $X = Y$, entonces es posible tomar dos copias de la expresión E (en la que tenemos presencias de una variable z), en una de ellas sustituir a la variable z por la expresión X , y en la otra copia sustituir a la misma variable z por la expresión Y , obteniendo que las expresiones $E[z := X]$ y $E[z := Y]$ son iguales nuevamente. Es decir, la sustitución de expresiones iguales en expresiones iguales genera expresiones iguales.

Es importante notar que en el caso de expresiones lógicas el concepto de igualdad que se utiliza es el de equivalencia lógica, es decir, si decimos que dos expresiones lógicas A y B son iguales, queremos decir que $A \equiv B$. De manera que en este caso podemos reescribir el argumento de Leibniz de la siguiente forma:

$$\frac{X \equiv Y}{E[z := X] \equiv E[z := Y]} \quad \text{Leibniz}$$

Veamos unos ejemplos de la aplicación de esta regla de inferencia.

Ejemplo 2.17. Supongamos que $b + 3 = c + 5$, y sea E la expresión aritmética $d + e$. Entonces, tenemos la siguiente instancia de la regla de Leibniz:

$$\frac{b + 3 = c + 5}{(d + e)[e := b + 3] = (d + e)[e := c + 5]},$$

lo que nos permite concluir que $d + (b + 3) = d + (c + 5)$ es verdadero en aquellos estados en los que $b + 3 = c + 5$ se evalúe a verdadero. Como la suma es asociativa y podemos eliminar paréntesis superfluos, esto es lo mismo que decir $d + b + 3 = d + c + 5$.

Las situaciones en las que usualmente se usa la regla de Leibniz se dan como sigue:

- Tenemos una expresión $E[z := X] = G$. Esto quiere decir que dada una expresión cualquiera G , localizamos en ella una subexpresión a la que denotamos con X . Esta subexpresión puede aparecer más de una vez, ya que la variable “original” z también puede ocurrir más de una vez en E .
- Buscamos una expresión Y que nos convenga, tal que $X = Y$.
- Podemos entonces obtener una nueva expresión $G' = E[z := Y]$.

- La regla de Leibniz nos permite concluir que $G = G'$

A continuación discutimos el ejemplo introductorio de esta sección.

Ejemplo 2.18. Sabemos que

- $x + y = y + x$ (2.35)

- $x + x = 2 \cdot x$ (2.36)

Sea $E = x + y + x + z$. Si deseamos simplificar esta expresión, debemos poder aplicar los dos hechos que sabemos – equivalencias (2.35) y (2.36) –. Por lo pronto, únicamente podemos aplicar la equivalencia (2.35), con dos lugares (en la expresión que queremos manipular) donde podemos hacerlo, considerando que tratamos de localizar a cualquiera de los dos lados de la igualdad:

- $\boxed{x+y} + x + z$ (primer acomodo)

- $x + \boxed{y+x} + z$ (segundo acomodo)

Si utilizamos el primer acomodo, entonces $X = x + y$ e $Y = y + x$, y sustituimos lo que está en la caja por la expresión equivalente:

$$\boxed{y+x} + x + z$$

Pero ahora tenemos la siguiente expresión, en la que, nuevamente, podemos localizar varias subexpresiones:

- $\boxed{y+x} + x + z$ (tercer acomodo)

- $y + \boxed{x+x} + z$ (cuarto acomodo)

Pero si elegimos el tercer acomodo, regresamos a donde estábamos, por lo que no nos conviene. Mejor elegimos el cuarto acomodo, utilizando la equivalencia (2.36) y tenemos $X = x + x$, $Y = 2 \cdot x$, quedándonos nuestra expresión de la siguiente forma:

$$y + \boxed{2 \cdot x} + z$$

El lector puede comprobar que también eligiendo el segundo patrón que reconocimos en la expresión original hubiésemos podido llegar al mismo resultado.

Veamos otro ejemplo aritmético en detalle.

Ejemplo 2.19.

Supongamos que queremos demostrar

$$(a + b) - b = a$$

y que conocemos las siguientes equivalencias:

$$(x + y) - z = x + (y - z) \quad (2.37)$$

$$y - y = 0 \quad (2.38)$$

$$x + 0 = x \quad (2.39)$$

Entonces, podemos pensar en la siguiente demostración, utilizando la propiedad de sustitución (teorema 2.1), la regla de Leibniz, y lo que ya conocemos. Como queremos demostrar que $(a + b) - b = a$, y dado que el lado izquierdo de la igualdad presenta más estructura, lo indicado es “salir” de ese lado y tratar, mediante la aplicación de la propiedad de sustitución y la regla de Leibniz, llegar a a . Es obvio que cada vez que pasamos a una nueva instancia de una regla de inferencia cualquiera, estamos utilizando la propiedad de transitividad para “encadenar” las igualdades:

Paso 1: Aplicar el Teorema 2.1.

$$\begin{aligned} ((x + y) - z = x + (y - z))[x, y, z := a, b, b] &= \\ &= ((a + b) - b = a + (b - b)) \end{aligned}$$

La propiedad que estamos utilizando es la de sustitución: sabemos que la premisa es una igualdad válida, una tautología, $((x + y) - z = x + (y - z))$ y elegimos las sustituciones que necesitamos para obtener la expresión con la que queremos trabajar $((a + b) - b)$. De la regla de inferencia tenemos lo siguiente:

$$\begin{array}{ll} E & \text{es} \quad (x + y) - z = x + (y - z) \\ E[x, y, z := a, b, b] & \text{es} \quad (a + b) - b = a + (b - b) \end{array}$$

También utilizamos este mismo teorema de sustitución para pasar de la expresión que tenemos $(y - y = 0)$ a la forma que queremos $(b - b = 0)$.

Paso 2: Volver a aplicar el Teorema 2.1.

$$\begin{array}{c} y - y = 0 \\ \hline (y - y = 0)[y := b] \end{array}$$

Y como

$$(y - y = 0)[y := b] \quad \text{es} \quad (b - b = 0),$$

tenemos ya:

$$(a + b) - b = a + (b - b) \quad (\text{por la aplicación del paso 1})$$

$$b - b = 0 \quad (\text{por la aplicación del paso 2})$$

Podemos ahora utilizar la regla de Leibniz de la siguiente manera:

Paso 3: Aplicar la regla de Leibniz.

$$\frac{b - b = 0}{a + (b - b) = a + 0}$$

donde:

$$\begin{array}{lll} X & \text{es} & b - b \\ Y & & 0 \\ E & & a + z \\ E[z := X] & & (a + z)[z := b - b] = (a + (b - b)) \\ E[z := 0] & & (a + z)[z := 0] = (a + (0)), \end{array}$$

que cuando quitamos los paréntesis superfluos nos dejan

$$a + (b - b) = a + 0$$

También sabemos que $x + 0 = x$ es una igualdad válida. Entonces podemos aplicarle sustitución textual y seguir teniendo una igualdad válida:

$$\textbf{Paso 4:} \quad \frac{x + 0 = x}{(x + 0 = x)[x := a]} .$$

Pero como $(x + 0 = x)[x := a]$ es $a + 0 = a$, tenemos la siguiente sucesión de igualdades válidas:

$$(a + b) - b = a + (b - b)$$

$$b - b = 0$$

$$a + (b - b) = a + 0$$

$$a + 0 = a$$

$$\frac{a + 0 = a}{(a + b) - b = a}$$

Decimos entonces que hemos demostrado que $(a + b) - b = a$ es una igualdad válida.

2.4.2. Álgebra de equivalencias lógicas

Análogamente al hecho de que el razonamiento aritmético ecuacional es la base del álgebra que conocemos desde hace tiempo, en el caso de las expresiones lógicas se genera un álgebra que manipula variables y constantes que representan valores de verdad; en particular podemos emplear equivalencias lógicas para deducir o simplificar nuevas expresiones a partir de otras ya conocidas. Ilustremos esto mediante algunos ejemplos.

Ejemplo 2.20. Sabemos que

$$\bullet P \wedge P \equiv P \quad (2.40)$$

$$\bullet P \wedge Q \equiv Q \wedge P \quad (2.41)$$

Supongamos que queremos “simplificar” la siguiente expresión:

$$q \wedge r \wedge q \wedge s$$

Para poder aplicar el argumento de Leibniz, hagamos primero sustitución textual sobre las variables, para tener los mismos términos:

$$(q \wedge r \wedge q \wedge s)[q, r, s := P, Q, R] = P \wedge Q \wedge P \wedge R.$$

Ahora tratemos de identificar alguno de los lados de las equivalencias dentro de la expresión que tenemos. Existen dos posiciones que podemos reconocer:

$$\bullet \boxed{P \wedge Q} \wedge P \wedge R \quad \text{– lado izquierdo de (2.41)}$$

$$\bullet P \wedge \boxed{Q \wedge P} \wedge R \quad \text{– lado derecho de (2.41)}$$

Si aplicamos a la primera elección la igualdad, $X = Y$ con $X = P \wedge Q$ y $Y = Q \wedge P$, la regla de Leibniz nos lleva a la expresión:

$$\frac{P \wedge Q \equiv Q \wedge P}{\boxed{P \wedge Q} \wedge P \wedge R \equiv \boxed{Q \wedge P} \wedge P \wedge R}.$$

Enseguida localizamos el otro esquema que corresponde a la equivalencia dada en (2.40), al principio de esta sección, donde $X = P \wedge P$ y $Y = P$. La sustitución se hace como sigue:

$$\frac{P \wedge P \equiv P}{Q \wedge \boxed{P \wedge P} \wedge R \equiv Q \wedge \boxed{P} \wedge R},$$

por lo que terminamos con la siguiente expresión:

$$Q \wedge P \wedge R \equiv P \wedge Q \wedge R;$$

de las dos aplicaciones de Leibniz y usando la regla de transitividad podemos concluir que

$$P \wedge Q \wedge P \wedge R \equiv P \wedge Q \wedge R.$$

Si nos quedamos con la expresión de la derecha y hacemos la sustitución de las variables de regreso a q , r y s , tenemos:

$$(P \wedge Q \wedge R)[P, Q, R := q, r, s] = q \wedge r \wedge s$$

y esta última es la simplificación final de la original.

Ejemplo 2.21. Consideremos ahora la siguiente expresión lógica $(P \wedge Q) \wedge \neg Q$. El objetivo es simplificarla lo más posible. Tenemos que:

$$1. \quad (A \wedge B) \wedge C \equiv A \wedge (B \wedge C) \quad \text{Propiedad asociativa de } \wedge$$

$$2. \quad (P \wedge Q) \wedge \neg Q \equiv P \wedge (Q \wedge \neg Q) \quad \text{Sustitución textual en 1)}$$

$$3. \quad P \wedge \neg P \equiv \text{false} \quad X = P \wedge \neg P \text{ e } Y = \text{false}$$

$$4. \quad P \wedge (Q \wedge \neg Q) \equiv P \wedge \text{false} \quad \text{Leibniz}$$

y como

$$P \wedge \text{false} \equiv \text{false} \quad \text{Elemento nulo}$$

ya terminamos.

De esta manera hemos demostrado que $(P \wedge Q) \wedge \neg Q \equiv \text{false}$, con la siguiente sucesión de equivalencias, utilizando la propiedad de transitividad de la equivalencia lógica:

$$\begin{aligned} (P \wedge Q) \wedge \neg Q &\equiv P \wedge (Q \wedge \neg Q) \\ &\equiv P \wedge \text{false} \\ &\equiv \text{false} \end{aligned}$$

En la tabla 2.5 (página 62) mostramos la lista inicial de equivalencias que vamos a utilizar para nuestro razonamiento ecuacional. Sin embargo existen muchas otras equivalencias que se pueden derivar de las anteriores y son de gran importancia. A continuación obtenemos algunas de ellas.

Leyes de absorción:

$$P \vee (P \wedge Q) \equiv P \quad (2.42)$$

$$P \wedge (P \vee Q) \equiv P \quad (2.43)$$

Leyes de simplificación:

$$(P \wedge Q) \vee (\neg P \wedge Q) \equiv Q \quad (2.44)$$

$$(P \vee Q) \wedge (\neg P \vee Q) \equiv Q \quad (2.45)$$

Debemos demostrar estas nuevas leyes, ya que no aparecen en nuestro conjunto inicial de equivalencias. Lo haremos con cuidado y detalle en uno de los casos, dejando el otro como ejercicio.

Ejemplo 2.22. Absorción frente a \vee : $P \vee (P \wedge Q) \equiv P$.

Utilizaremos el método de tomar a uno de los equivalentes y derivar, a partir de él, al otro. Como el de la izquierda tiene más estructura, es el que tomamos como punto de partida.

Punto de partida. Localizaremos este esquema en alguno de los axiomas o teoremas que ya hayamos demostrado. En este momento únicamente contamos con (2.14) a (2.33).

Usando Identidad (2.17) y Leibniz.

$$P \equiv P \wedge \text{true}$$

$$\boxed{P} \vee (P \wedge Q) \equiv \boxed{(P \wedge \text{true})} \vee (P \wedge Q)$$

Distributividad de \wedge (2.28)

$$(P \wedge Q) \vee (P \wedge R) \equiv P \wedge (Q \vee R)$$

Usando sustitución $[Q, R := \text{true}, Q]$ tenemos:

$$P \vee (P \wedge Q)$$

$$\equiv (P \wedge \text{true}) \vee (P \wedge Q)$$

$$\equiv P \wedge (\text{true} \vee Q)$$

(Continúa en la siguiente página)

	(Continúa de la página anterior)
(de la página anterior)	$\equiv P \wedge (\text{true} \vee Q)$
Usando dominación (2.20) y Leibniz:	
$\frac{Q \vee \text{true} \equiv \text{true}}{P \wedge (Q \vee \text{true}) \equiv P \wedge \boxed{\text{true}}}$	$\equiv P \wedge \text{true}$
Usando identidad de \wedge (2.17)	$\equiv P$

Ejemplo 2.23. Simplificación: $(P \vee Q) \wedge (\neg P \vee Q) \equiv Q$.

Nuevamente tenemos que demostrar una equivalencia lógica, por lo que trataremos de transformar a uno de los equivalentes en el otro. Como el equivalente de la izquierda tiene mayor estructura, partiremos de él. Dado que el número que le corresponde a este teorema es el (2.44), podemos utilizar en este caso las leyes ((2.14) a (2.43)).

Punto de partida. Localizaremos este esquema en alguno de los axiomas o teoremas que ya hayamos demostrado. Vemos un esquema similar en el rango derecho de (2.28):	$(P \vee Q) \wedge (\neg P \vee Q)$
Usando Conmutatividad (2.22).	
$(P \vee Q) \wedge (\neg P \vee Q) \equiv (Q \vee P) \wedge (Q \vee \neg P)$	$\equiv (Q \vee P) \wedge (Q \vee \neg P)$
Instanciando (2.28)	
$\frac{(P \vee Q) \wedge (P \vee R) \equiv P \vee (Q \wedge R)}{(Q \vee P) \wedge (Q \vee \neg P) \equiv Q \vee (P \wedge \neg P)}$	$\equiv Q \vee (P \wedge \neg P)$
Contradicción: (2.25)	
$P \wedge \neg P \equiv \text{false}$	$\equiv Q \vee \text{false}$

(Continúa en la siguiente página)

(Continúa de la página anterior)	
(de la página anterior)	$\equiv Q \vee \text{false}$
Identidad: (2.16)	
$Q \vee \text{false} \equiv Q$	$\equiv Q$

Se deja como ejercicio la demostración de (2.43).

En todos los ejemplos de esta sección marcamos e hicimos explícitos todos los usos de las reglas. Sin embargo, en la práctica muchas de estas reglas se usan de manera implícita. A continuación damos algunos atajos que se pueden tomar al hacer álgebra de equivalencias lógicas.

1. La Ley de Conmutatividad se aplica directamente, “sin avisar”.
2. La Ley de Asociatividad se aplica directamente, “sin avisar”.
3. Se puede desechar directamente lo siguiente:
 - a) Copias duplicadas de una subexpresión en una expresión que es una disyunción o una conjunción (Ley de Idempotencia).
 - b) La constante true en una conjunción (Ley de Identidad para \wedge).
 - c) La constante false en una disyunción (Ley de Identidad para \vee).
4. De igual manera, se puede simplificar haciendo lo siguiente:
 - a) Sustituir el esquema $A \wedge \neg A$ por false (Ley de Contradicción).
 - b) Sustituir el esquema $A \vee \neg A$ por true (Ley del Tercero Excluido).
 - c) Sustituir el esquema $\neg\neg A$ por A (Ley de Doble Negación).

En esta sección hemos mostrado cómo es posible justificar formalmente el razonamiento ecuacional usual. Esta justificación, que se hizo apelando al uso de la regla de Leibniz, además de proporcionar un fundamento matemático formal a un razonamiento al que estamos acostumbrados desde hace mucho, nos da una pauta para una posible automatización del proceso.

En adelante el uso de razonamiento ecuacional será, por lo general, intuitivo, sin requerir el uso explícito de la regla de Leibniz.

Para terminar probaremos la equivalencia lógica entre una implicación y su contrapositiva, usando algunos de los atajos anteriores. Esto justifica el método de demostración por contrapositivo, usual en Matemáticas.

Ejemplo 2.24. Contrapositiva: $P \rightarrow Q \equiv \neg Q \rightarrow \neg P$. Usaremos la ley de eliminación de la

implicación mediante disyunción, así como las leyes de De Morgan.

$$\begin{aligned}
 P \rightarrow Q &\equiv \neg P \vee Q \\
 &\equiv \neg(P \wedge \neg Q) \\
 &\equiv \neg(\neg(Q \vee \neg P)) \\
 &\equiv Q \vee \neg P \\
 &\equiv \neg Q \rightarrow \neg P
 \end{aligned}$$

Ejercicios

2.4.1.- Para las siguientes expresiones E , dadas z , X e Y , obtener $E[z := X]$ y $E[z := Y]$.

	z	E	X	Y
(a)	p	p	$p \wedge q$	$q \wedge p$
(b)	p	$(p \vee q) \wedge (p \vee r)$	true	$p \leftrightarrow p$
(c)	p	$p \wedge p \leftrightarrow p$	$p \vee q$	$p \vee \neg q \leftrightarrow p$
(d)	q	$p \wedge (\neg p \wedge q)$	$p \vee (q \wedge r)$	$(p \vee q) \wedge (p \vee r)$

2.4.2.- La regla de Leibniz se refiere a cualquier combinación de expresiones E , X e Y y a cualquier variable z . A continuación damos varios razonamientos que siguen el patrón de Leibniz y que están incompletos. El orden no es forzosamente el dado por la expresión, esto es, abajo de X no forzosamente está $E[z := X]$. Llena las partes que faltan y escribe en qué consiste la expresión E . Los últimos dos ejercicios tienen tres respuestas. Dadas todas.

$$a) \frac{p \leftrightarrow p \vee 0}{p \vee 0 \vee q \leftrightarrow ?}$$

$$b) \frac{7 = y + 1}{7 \cdot x + 7 \cdot y = ?}$$

$$c) \frac{p \rightarrow q \leftrightarrow \neg q \rightarrow \neg p}{p \rightarrow q \rightarrow p \leftrightarrow ?}$$

$$d) \frac{x = b + c}{x + y + w = ?}$$

$$e) \frac{x + 1 = y}{3 \cdot (x + 1) + 3 \cdot x + 1 = ?}$$

$$f) \frac{b \cdot c = y + w}{x + y + w = ?}$$

$$g) \frac{x = y}{x + x = ?}$$

2.4.3.- El objetivo de este ejercicio es reforzar las habilidades en el uso del argumento de Leibniz para demostrar que dos expresiones son iguales. Vamos a dar las expresiones $E[z := X]$ y $E[z := Y]$ y deberás localizar respectivamente a X y a Y .

	$E[z := X]$	$E[z := Y]$
(a)	$(x + y) \cdot (x + y)$	$(x + y) \cdot (y + x)$
(b)	$(x + y) \cdot (x + y)$	$(y + x) \cdot (y + x)$
(c)	$x + y + w + x$	$x + y \cdot w + x$
(d)	$x \cdot y \cdot x$	$(y + w) \cdot y \cdot x$
(e)	$x \cdot y \cdot x$	$y \cdot x \cdot x$

2.4.4.- Elimina los operadores \rightarrow y \leftrightarrow de cada una de las siguientes proposiciones:

$$a) (P \rightarrow Q \wedge R) \vee ((R \leftrightarrow S) \wedge (Q \vee S))$$

$$b) (P \rightarrow Q) \wedge (Q \rightarrow R)$$

$$c) \neg P \rightarrow \neg Q$$

$$d) (P \rightarrow Q) \leftrightarrow ((P \wedge Q) \leftrightarrow Q)$$

2.5. Conceptos semánticos importantes

Una vez que hemos estudiado el análisis sintáctico de una fórmula lógica pasamos a estudiar ciertos conceptos de importancia relacionados con su semántica.

2.5.1. Interpretaciones

La noción de interpretación presentada en esta sección será de gran importancia para evitar el uso de tablas de verdad en las pruebas de correctud.

Definición 2.11 Un *estado* de las variables proposicionales es una función \mathcal{I} que asigna a cada variable proposicional el valor de falso o verdadero:

$$\mathcal{I} : \text{Variables proposicionales} \rightarrow \{0, 1\}$$

Cada estado genera una función de interpretación sobre todas las fórmulas, definida como se explica a continuación.

Definición 2.12 Cada estado \mathcal{I} determina una *interpretación* de las fórmulas – denotada también por \mathcal{I} – definida como sigue:

$$\begin{aligned} \mathcal{I}(\text{true}) &= 1 \\ \mathcal{I}(\text{false}) &= 0 \\ \mathcal{I}(\neg P) &= 1 && \text{si y sólo si} && \mathcal{I}(P) = 0 \\ \mathcal{I}(P \vee Q) &= 0 && \text{si y sólo si} && \mathcal{I}(P) = 0 = \mathcal{I}(Q) \\ \mathcal{I}(P \wedge Q) &= 1 && \text{si y sólo si} && \mathcal{I}(P) = 1 = \mathcal{I}(Q) \\ \mathcal{I}(P \rightarrow Q) &= 0 && \text{si y sólo si} && \mathcal{I}(P) = 1 \text{ e } \mathcal{I}(Q) = 0 \\ \mathcal{I}(P \leftrightarrow Q) &= 1 && \text{si y sólo si} && \mathcal{I}(P) = \mathcal{I}(Q) \end{aligned}$$

Si $\mathcal{I}(P) = 1$ entonces decimos que

- \mathcal{I} *satisface a* P , o bien
- P *es satisfacible en* \mathcal{I} , o bien
- P *se satisface en* \mathcal{I} , o bien
- \mathcal{I} es un *modelo* de P .

Ejemplo 2.25. Si tenemos la fórmula $A = p \rightarrow q \vee r$, la siguiente asignación de estado

$$\mathcal{I}_1(p) = 1, \mathcal{I}_1(q) = 0, \mathcal{I}_1(r) = 0,$$

hace $\mathcal{I}_1(p \rightarrow q \vee r) = 0$, por lo que \mathcal{I}_1 no es un modelo para la fórmula. Por otro lado, el estado

$$\mathcal{I}_2(p) = 1, \mathcal{I}_2(q) = 0, \mathcal{I}_2(r) = 1$$

hace que $\mathcal{I}_2(p \rightarrow q \vee r) = 1$, por lo que sí es un modelo para la fórmula.

Dada una fórmula P podemos preguntarnos ¿cuántas interpretaciones hacen verdadera a P ? Las posibles respuestas llevan a las siguientes definiciones:

Definición 2.13 Sea P una fórmula. Entonces

- Si $\mathcal{I}(P) = 1$ para toda interpretación \mathcal{I} , decimos que P es una tautología o fórmula válida y escribimos $\models P$.
- Si $\mathcal{I}(P) = 1$ para alguna interpretación \mathcal{I} , decimos que P es satisfacible, que P es verdadera en \mathcal{I} o que \mathcal{I} es modelo de P y escribimos $\mathcal{I} \models P$.
- Si $\mathcal{I}(P) = 0$ para alguna interpretación \mathcal{I} , decimos que P es falsa o insatisfacible en \mathcal{I} o que \mathcal{I} no es modelo de P y escribimos $\mathcal{I} \not\models P$.
- Si $\mathcal{I}(P) = 0$ para toda interpretación \mathcal{I} , decimos que P es una contradicción o fórmula no satisfacible.

Similarmente, si Γ es un conjunto de fórmulas decimos que:

- Γ es satisfacible si tiene un modelo, es decir, si existe una interpretación \mathcal{I} tal que $\mathcal{I}(P) = 1$ para toda $P \in \Gamma$, lo cual denotamos a veces, abusando de la notación, con $\mathcal{I}(\Gamma) = 1$.
- Γ es insatisfacible o no satisfacible si no tiene un modelo, es decir, si no existe una interpretación \mathcal{I} tal que $\mathcal{I}(P) = 1$ para toda $P \in \Gamma$.

Para el último ejemplo se cumple lo siguiente, de acuerdo a la definición anterior, $\mathcal{I}_1 \not\models A$, $\mathcal{I}_2 \models A$, $\not\models A$. Veamos otro ejemplo.

Ejemplo 2.26. Sean $\Gamma_1 = \{p \rightarrow q, r \rightarrow s, \neg s\}$, $\Gamma_2 = \{p \rightarrow q, \neg(q \vee s), s \vee p\}$. Entonces

- Si $\mathcal{I}(s) = \mathcal{I}(r) = \mathcal{I}(p) = 0$, entonces $\mathcal{I}(\Gamma_1) = 1$ por lo que Γ_1 es satisfacible.
- Γ_2 resulta insatisfacible pues supóngase que existe una interpretación \mathcal{I} tal que $\mathcal{I}(\Gamma_2) = 1$. Entonces, se tiene que $\mathcal{I}(\neg(q \vee s)) = 1$ por lo que $\mathcal{I}(\neg q) = \mathcal{I}(\neg s) = 1$. Además como $\mathcal{I}(p \rightarrow q) = 1$ entonces $\mathcal{I}(p) = 0$ puesto que el antecedente de la implicación es falso. De esto último se tiene $\mathcal{I}(s) = 1$ dado que $\mathcal{I}(s \vee p) = 1$. De manera que se tiene $\mathcal{I}(\neg s) = 1 = \mathcal{I}(s)$ lo cual es imposible. Por lo tanto no puede existir una interpretación \mathcal{I} que satisfaga a Γ_2 .

Con respecto a las tablas de verdad tenemos las siguientes observaciones:

- Una fórmula P es satisfacible si en alguna línea de la tabla de verdad, P toma el valor 1. En caso contrario, es decir si en **todas** las líneas toma el valor 0, entonces es insatisfacible (contradicción).
- Un conjunto de fórmulas Γ es satisfacible si existe alguna línea de la tabla de verdad en la que **todas** las fórmulas de Γ toman el valor 1.

2.5.2. Consecuencia lógica

La definición matemática formal de argumento deductivo correcto se sirve del concepto de consecuencia o implicación lógica que discutimos aquí.

Definición 2.14 (consecuencia lógica) Sean $\Gamma = \{A_1, \dots, A_n\}$ un conjunto de fórmulas y B una fórmula. Decimos que B es consecuencia lógica de Γ si toda interpretación \mathcal{I} que satisface a Γ también satisface a B . Es decir, si todo modelo de Γ es modelo de B . En tal caso escribimos $\Gamma \models B$.

Nótese que la relación de consecuencia lógica está dada por una implicación de la forma

$$\text{Si } \mathcal{I}(\Gamma) = 1 \text{ entonces } \mathcal{I}(B) = 1.$$

De manera que no se afirma nada acerca de la satisfacibilidad del conjunto Γ , sino que simplemente se supone que es satisfacible y, en tal caso, se prueba que la fórmula B también lo es con la misma interpretación.

Obsérvese la sobrecarga del símbolo \models que previamente utilizamos para denotar satisfacibilidad $\mathcal{I} \models A$ y tautologías $\models A$.

Ejemplo 2.27. Considerese el siguiente conjunto $\Gamma = \{q \rightarrow p, p \leftrightarrow t, t \rightarrow s, s \rightarrow r\}$. Muestre que $\Gamma \models q \rightarrow r$. Sea \mathcal{I} un modelo de Γ . Tenemos que demostrar que $\mathcal{I}(q \rightarrow r) = 1$. Si $\mathcal{I}(q) = 0$ entonces $\mathcal{I}(q \rightarrow r) = 1$ y terminamos. En otro caso se tiene $\mathcal{I}(q) = 1$ de donde $\mathcal{I}(p) = 1$ pues $\mathcal{I}(q \rightarrow p) = 1$. Entonces se tiene $\mathcal{I}(t) = 1$, pues \mathcal{I} es modelo de $p \leftrightarrow t$, de donde $\mathcal{I}(s) = 1$ dado que \mathcal{I} también es modelo de $t \rightarrow s$. Finalmente, como $\mathcal{I}(s \rightarrow r) = 1$ e $\mathcal{I}(s) = 1$ entonces $\mathcal{I}(r) = 1$. Por lo tanto $\mathcal{I}(q \rightarrow r) = 1$.

Para terminar la sección discutimos algunas propiedades importantes de la relación de consecuencia lógica.

Proposición 2.1 La relación de consecuencia lógica cumple las siguientes propiedades:

- (a) Si $A \in \Gamma$ entonces $\Gamma \models A$.
- (b) Principio de refutación: $\Gamma \models A$ si y sólo si $\Gamma \cup \{\neg A\}$ es insatisfacible.
- (c) $\Gamma \models A \rightarrow B$ si y sólo si $\Gamma \cup \{A\} \models B$.

- (d) *Insatisfacibilidad implica trivialidad: Si Γ es insatisfacible entonces $\Gamma \models A$ para toda fórmula A .*
- (e) *Si $\Gamma \models \text{false}$ entonces Γ es insatisfacible.*
- (f) *$A \equiv B$ si y sólo si $A \models B$ y $B \models A$.*
- (g) *$\models A$ (es decir A es tautología) si y sólo si $\emptyset \models A$ (es decir A es consecuencia lógica del conjunto vacío).*

Demostración.

Procedemos a justificar algunos de los incisos:

- (a) Si $\mathcal{I}(\Gamma) = 1$ quiere decir que existe un modelo para Γ y, por lo tanto, para cada una de las fórmulas de Γ , en particular para A .
- (b) Supongamos que toda interpretación que satisface a Γ también satisface a A (definición de $\Gamma \models A$). Si una interpretación satisface a Γ , dado que satisfacía también a A , entonces no satisface a $\neg A$. Por lo tanto, es imposible satisfacer a Γ y a $\neg A$ al mismo tiempo, lo cual implica que $\Gamma \cup \{\neg A\}$ es insatisfacible.

En sentido contrario, supongamos que $\Gamma \cup \{\neg A\}$ es insatisfacible. Para mostrar que $\Gamma \models A$, consideremos \mathcal{I} una interpretación cualquiera tal que $\mathcal{I}(\Gamma) = 1$. En tal caso, necesariamente tenemos que $\mathcal{I}(A) = 1$ puesto que de lo contrario $\mathcal{I}(A) = 0$, por lo que $\mathcal{I}(\neg A) = 1$ y así $\Gamma \cup \{\neg A\}$ sería satisfacible mediante \mathcal{I} , lo cual por hipótesis no puede suceder.

- (c) Supongamos $\Gamma \models A \rightarrow B$. Por la definición de consecuencia lógica, tenemos que si $\mathcal{I}(\Gamma) = 1$ entonces $\mathcal{I}(A \rightarrow B) = 1$. Para mostrar que $\Gamma \cup \{A\} \models B$ sea \mathcal{I} una interpretación tal que $\mathcal{I}(\Gamma \cup \{A\}) = 1$; en esta interpretación se tiene que $\mathcal{I}(A) = 1$; como además $\mathcal{I}(A \rightarrow B) = 1$ por hipótesis, porque estamos suponiendo $\mathcal{I}(\Gamma) = 1$, entonces por definición de la interpretación de una implicación, dado que para el antecedente A se tiene $\mathcal{I}(A) = 1$, entonces necesariamente $\mathcal{I}(B) = 1$. Por lo tanto $\Gamma \cup \{A\} \models B$.

En sentido contrario, supongamos que $\Gamma \cup \{A\} \models B$. Esto es que si $\mathcal{I}(\Gamma \cup \{A\}) = 1$ entonces $\mathcal{I}(B) = 1$. Sea \mathcal{I} una interpretación tal que $\mathcal{I}(\Gamma) = 1$. Tenemos los siguientes casos:

- $\mathcal{I}(A) = 1$. Entonces $\mathcal{I}(B) = 1$, pues se cumple que $\mathcal{I}(\Gamma \cup \{A\}) = 1$; con lo que $\mathcal{I}(A \rightarrow B) = 1$ y tenemos que $\Gamma \models A \rightarrow B$.
- $\mathcal{I}(A) = 0$. En este caso, independientemente de cuál sea el valor de $\mathcal{I}(B)$, tenemos $\mathcal{I}(A \rightarrow B) = 1$, por lo que nuevamente $\Gamma \models A \rightarrow B$.

Por lo tanto, $\Gamma \models A \rightarrow B$ si y sólo si $\Gamma \cup \{A\} \models B$.

- (d) Si Γ es insatisfacible, quiere decir que para toda interpretación \mathcal{I} , se tiene $\mathcal{I}(\Gamma) = 0$. Si esto es así, se cumple trivialmente que si $\mathcal{I}(\Gamma) = 1$ entonces $\mathcal{I}(A) = 1$. Es decir $\Gamma \models A$.

- (e) Si $\Gamma \models \text{false}$, por la definición de consecuencia lógica tenemos que $\mathcal{I}(\Gamma) = 1$ implica $\mathcal{I}(\text{false}) = 1$. Sin embargo, $\mathcal{I}(\text{false}) = 0$ siempre sucede; por lo que, como $\Gamma \models \text{false}$ tenemos necesariamente que $\mathcal{I}(\Gamma) = 0$ para toda posible interpretación de Γ , es decir, Γ es insatisfacible.

Se deja la justificación de los incisos restantes al lector.

Es importante disponer de métodos algorítmicos para decidir la consecuencia lógica, que nos permitirán, en particular, analizar argumentos del lenguaje natural y establecer su correctud formalmente. En las siguientes secciones presentaremos algunos de estos métodos.

Ejercicios

2.5.1.- Para cada una de las fórmulas que siguen, determina si son o no *satisfacibles*. Si lo son, muestra un *modelo* para cada una de ellas.

- (a) $p \wedge q \leftrightarrow \neg p \wedge q$
- (b) $(\neg p \vee q) \wedge p$
- (c) $p \wedge q \wedge \neg p$
- (d) $(p \rightarrow q) \wedge (q \rightarrow p)$

2.5.2.- Usa interpretaciones para determinar si las siguientes fórmulas son tautologías, contradicciones o contingentes. Si son contingentes, da una interpretación en la que la fórmula no se evalúa a verdadero.

- (a) $\left(((p \vee q) \vee r) \wedge (p \vee (q \vee r)) \right) \rightarrow p \vee q$
- (b) $(p \wedge (q \wedge r)) \rightarrow (p \rightarrow (q \rightarrow r))$
- (c) $p \vee q \rightarrow p \vee r$
- (d) $(p \rightarrow (p \rightarrow q)) \rightarrow p$

2.5.3.- Decide si los siguientes conjuntos son satisfacibles.

- a) $\Gamma = \{(\neg q \wedge r) \vee p \vee q, p \wedge r\}$
- b) $\Gamma = \{p \wedge \neg q, \neg(q \vee \neg p), (q \wedge p) \vee q \vee \neg p\}$
- c) $\Gamma = \{q \vee r \vee s, \neg(q \vee r), \neg(r \vee s), \neg(s \vee q)\}$
- d) $\Gamma = \{\neg(p \wedge q) \wedge \neg(p \wedge r), q \vee r, \neg(p \vee \neg r)\}$
- e) $\Gamma = \{p \leftrightarrow q, q \leftrightarrow s, p, \neg s\}$

2.5.4.- Demuestra la consecuencia lógica en cada caso.

- a) $\{p, q\} \models p \wedge q$
- b) $\{p, \neg q\} \models \neg(p \rightarrow q)$
- c) $\{p \vee q, p \rightarrow r, q \rightarrow r\} \models r$
- d) $\{p \rightarrow q, p \rightarrow \neg q\} \models \neg p$
- e) $\{r \wedge s \rightarrow t, \neg t\} \models t \rightarrow q$
- f) $\{\neg q \rightarrow \neg r, \neg r \rightarrow \neg p, \neg p \rightarrow \neg q\} \models q \leftrightarrow r$

2.6. Análisis de argumentos

En esta sección aplicamos todos los conocimientos previos de lógica matemática estudiados hasta ahora para cumplir con nuestro propósito fundamental: el análisis de correctud de un argumento lógico proposicional.

2.6.1. Tablas de Verdad

Como ya discutimos antes, un argumento es correcto si y sólo si su fórmula asociada es una tautología; para decidir esta situación podemos construir la tabla de verdad correspondiente tal y como lo hicimos en la sección 2.1.6. Veamos un ejemplo más.

Ejemplo 2.28. El argumento $P \rightarrow Q, Q \rightarrow R / \therefore P \rightarrow R$ es correcto.

Basta ver que $\models (P \rightarrow Q) \wedge (Q \rightarrow R) \rightarrow (P \rightarrow R)$. La tabla de verdad se muestra en la tabla 2.8 a continuación.

Tabla 2.8 $P \rightarrow Q, Q \rightarrow R / \therefore P \rightarrow R$

P	Q	R	$(P \rightarrow Q)$	\wedge	$(Q \rightarrow R)$	\rightarrow	$(P \rightarrow R)$
1	1	1	1	1	1	1	1
1	1	0	1	0	0	1	0
1	0	1	0	0	1	1	1
1	0	0	0	0	1	1	0
0	1	1	1	1	1	1	1
0	1	0	1	0	0	1	1
0	0	1	1	1	1	1	1
0	0	0	1	1	1	1	1

Como se observa de los valores en la quinta columna, en negritas, la fórmula es una tautología, por lo que este argumento, conocido como *silogismo hipotético*, es correcto.

Este ejemplo, junto con los de la sección 2.1.6, deja ver que la tabla de verdad se vuelve más complicada al aumentar el número de variables proposicionales involucradas. La construcción de la tabla de verdad completa, aunque plausible desde el punto de vista teórico, es de “fuerza bruta”. En la práctica pues nos obliga, en los casos interesantes y no triviales, a evaluar un número muy grande de estados para determinar si tenemos o no una tautología (o una contradicción). Aun si lo hiciésemos con una computadora, y suponiendo que a la computadora le llevara un milisegundo evaluar cada estado, si la expresión es muy grande tenemos el crecimiento en el número de estados que vemos en la tabla 2.9.

Tabla 2.9 Crecimiento en el número de estados con respecto a número de variables

Número de variables	Número de estados	Tiempo (segundos)
1	2	.002
2	4	.004
3	8	.008
⋮	⋮	⋮
10	1,024	1
11	2,048	2
⋮	⋮	⋮
20	1,048,576	1,048 (= 17min)
⋮	⋮	⋮

Como se puede observar en la tabla 2.9, cada vez que se agrega una variable a la expresión, el tiempo que lleva calcular todos sus estados se duplica, siguiendo, como ya mencionamos, a la función 2^n , donde n es el número de variables³. Esta ineficiencia surge en la práctica, por ejemplo, en problemas de calendarización o búsqueda de rutas donde ciertas fórmulas lógicas involucradas tienen usualmente cientos de variables. Para estimar la ineficiencia considérese una fórmula con 500 variables, cuya tabla de verdad tendrá 2^{500} renglones, número aproximadamente igual a 10^{150} , los cuales, de acuerdo a nuestra suposición anterior respecto a la velocidad de la computadora, se calcularían en 10^{147} milisegundos. Dado que en un año hay 3.1536×10^{10} milisegundos, la tabla terminaría de calcularse en aproximadamente 3.2×10^{139} años; considerando que la edad de nuestro planeta es

³Cuando tenemos este tipo de cálculo, decimos que la función crece con 2^n , o que tiene un *crecimiento exponencial*. Este tipo de cálculos, en la práctica, no pueden ser evaluados en una computadora cuando la n no es pequeña.

aproximadamente 10^9 años, podemos corroborar que el tiempo estimado del método es inadmisibile.

Dada esta situación, vamos a utilizar tablas de verdad únicamente para verificar expresiones pequeñas y cuando no podamos recurrir a otras técnicas.

Obsérvese que el método de tablas de verdad puede evitarse al usar esquemas: una vez que se prueba que un argumento es correcto, él mismo genera un esquema, llamado regla de inferencia y cada instancia de estaregla será, a su vez, un argumento correcto.

Ejemplo 2.29. Mostrar la correctud del argumento

$$\frac{r \rightarrow s \vee \neg t \quad (r \rightarrow s \vee \neg t) \rightarrow \neg p \wedge (q \vee w)}{\therefore \neg p \wedge (q \vee w)}$$

La tabla de verdad para este análisis tendría $2^6 = 64$ renglones, dado que tenemos seis variables. Sin embargo, no es necesario el análisis puesto que el argumento corresponde al esquema del modus ponens que ya mostramos que es correcto. Formalmente tenemos que

$$\begin{aligned} (P \wedge (P \rightarrow Q) \rightarrow Q) [P, Q := r \rightarrow s \vee \neg t, \neg p \wedge (q \vee w)] &= \\ = ((r \rightarrow s \vee \neg t) \wedge ((r \rightarrow s \vee \neg t) \rightarrow (\neg p \wedge (q \vee w))) &\rightarrow (\neg p \wedge (q \vee w))) \end{aligned}$$

y como $\models P \wedge (P \rightarrow Q) \rightarrow Q$ podemos concluir que

$$\models ((r \rightarrow s \vee \neg t) \wedge ((r \rightarrow s \vee \neg t) \rightarrow (\neg p \wedge (q \vee w))) \rightarrow (\neg p \wedge (q \vee w))).$$

Este método es útil en algunos casos en los que ya se conoce de antemano un esquema de argumento correcto; sin embargo no es siempre efectivo ni fácil de implementar.

2.6.2. Uso de interpretaciones

Ya estamos convencidos de que el uso de una tabla de verdad para analizar la correctud de un argumento es una muy mala idea en general.

Construir la tabla de verdad para una fórmula de la forma $A_1 \wedge \dots \wedge A_n \rightarrow B$, en su totalidad, resulta, en la mayoría de los casos, innecesario. Por ejemplo, al observar nuevamente la tabla 2.8, podemos darnos cuenta de que sólo nos interesa la mitad de ésta, a saber los renglones donde la conjunción de las premisas es verdadera. El resto de la tabla puede desecharse puesto que si la conjunción de las premisas no es verdadera, la implicación será verdadera automáticamente. El concepto de consecuencia lógica toma en cuenta esta observación al suponer que las premisas son ciertas y bajo este supuesto mostrar que, bajo la misma interpretación, la conclusión también lo es.

Para mostrar la correctud del argumento lógico $A_1, \dots, A_n / \therefore B$ mediante el uso de interpretaciones, nos servimos de la siguiente proposición cuya demostración dejamos como ejercicio.

Proposición 2.2 *El argumento $A_1, \dots, A_n / \therefore B$ es lógicamente correcto si y sólo si*

$$\{A_1, \dots, A_n\} \models B,$$

es decir, si la conclusión es consecuencia lógica de las premisas.

De acuerdo a las propiedades de la consecuencia lógica, existen básicamente dos formas para demostrar la correctud de un argumento, el método directo y el indirecto.

Método directo: Probar la consecuencia $A_1, \dots, A_n \models B$. Para esto se supone la existencia de una interpretación \mathcal{I} que sea modelo de todas las premisas y se argumenta, usando esta información y la definición de interpretación, que la conclusión B también se satisface con \mathcal{I} .

Método indirecto (refutación o contradicción): Probar que es insatisfacible el conjunto $\{A_1, \dots, A_n, \neg B\}$. Para esto se supone que hay una interpretación \mathcal{I} que hace verdaderas a todas las premisas y a la negación de la conclusión $\neg B$ o bien, equivalentemente, hace falsa a la conclusión B . Apelando a este supuesto y a la definición de interpretación, se trata de mostrar que tal interpretación no puede existir; esto se logra mostrando que cierta fórmula está forzada a ser verdadera y falsa al mismo tiempo.

Es de importancia observar que estos métodos son la base de los métodos usuales de demostración en matemáticas. En un curso cualquiera de matemáticas, cuando se dice que la demostración de un teorema de la forma $A \rightarrow B$ es directa es porque estamos probando la consecuencia $A \models B$ con el método directo. Similarmente si hablamos de una demostración indirecta o por contradicción o reducción al absurdo es porque estamos probando $A \models B$ con el método indirecto.

Veamos algunos ejemplos.

Ejemplo 2.30. Mostrar la correctud del argumento $\{p, s \vee \neg s, \neg p \vee q, \neg q \leftrightarrow r\} / \therefore \neg r$.

Sean $\Gamma = \{p, s \vee \neg s, \neg p \vee q, \neg q \leftrightarrow r\}$; debemos mostrar que $\Gamma \models \neg r$, para lo cual tomamos una interpretación \mathcal{I} tal que \mathcal{I} es modelo de Γ . Debemos mostrar que $\mathcal{I}(\neg r) = 1$.

Como \mathcal{I} es modelo de Γ entonces $\mathcal{I}(p) = 1$ e $\mathcal{I}(\neg p \vee q) = 1$, de donde $\mathcal{I}(q) = 1$ puesto que $\mathcal{I}(\neg p) = 0$. Como $\mathcal{I}(q) = 1$ e $\mathcal{I}(\neg q \leftrightarrow r) = 1$ entonces $\mathcal{I}(r) = 0$, de donde finalmente se obtiene $\mathcal{I}(\neg r) = 1$. Obsérvese que la prueba no determina un valor para s ya que con esta interpretación el argumento es correcto independientemente del valor de s . En particular, la única fórmula que involucra a s es la tautología $s \vee \neg s$.

Este método puede resultar tedioso o intrincado pero puede escribirse de manera más clara enunciando cada paso de razonamiento, como en el siguiente ejemplo.

Ejemplo 2.31. Mostrar la correctud del argumento $p \rightarrow q, \neg q / \therefore \neg p$, conocido como *Modus Tollens*, al que se hace referencia más adelante.

Para lograr esto mostramos la consecuencia lógica $p \rightarrow q, \neg q \models \neg p$.

1. $\mathcal{I}(p \rightarrow q) = 1$ Hipótesis
2. $\mathcal{I}(\neg q) = 1$ Hipótesis
3. $\mathcal{I}(q) = 0$ por 2, ya que $\mathcal{I}(\neg q) = 1$
4. $\mathcal{I}(p) = 0$ por 1 y 3, ya que si $\mathcal{I}(p \rightarrow q) = 1$ e $\mathcal{I}(q) = 0$,
 $\therefore \mathcal{I}(p)$ no puede ser 1.

De manera que el argumento es correcto. El razonamiento paso a paso permite una mayor claridad en el proceso de análisis. Por supuesto que cada paso debe tener una justificación exacta. El análisis terminó aquí al llegar a que la conclusión es verdadera, por lo que se probó la consecuencia lógica de manera directa.

Ejemplo 2.32. Si hoy tirila y Chubaka es kismi entonces Chubaka es borogrove y si hoy no tirila entonces hay fefos. Más aún sabemos que no hay fefos y que Chubaka es kismi, luego entonces Chubaka es borogrove.

La formalización es:

Variable Proposicional	Enunciado
t	hoy tirila
k	Chubaka es kismi
b	Chubaka es borogrove
f	hay fefos

y el argumento queda como sigue:

$t \wedge k \rightarrow b$	Si hoy tirila y Chubaka es kismi entonces Chubaka es borogrove
$\neg t \rightarrow f$	si hoy no tirila entonces hay fefos
$\neg f \wedge k$	sabemos que no hay fefos y que Chubaka es kismi
$\therefore b$	de donde Chubaka es borogrove

Queremos demostrar que $\{t \wedge k \rightarrow b, \neg t \rightarrow f, \neg f \wedge k\} \models b$.

1. $\mathcal{I}(t \wedge k \rightarrow b) = 1$ Hipótesis.
2. $\mathcal{I}(\neg t \rightarrow f) = 1$ Hipótesis.
3. $\mathcal{I}(\neg f \wedge k) = 1$ Hipótesis.
4. $\mathcal{I}(b) = 0$ Refutación.
5. $\mathcal{I}(k) = 1$ por 3, $\mathcal{I}(p \wedge q) = 1$ si y sólo si $\mathcal{I}(p) = 1$ e $\mathcal{I}(q) = 1$
6. $\mathcal{I}(t \wedge k) = 0$ por 4 y 1. Como $\mathcal{I}(b) = 0$ y la implicación en 1 es verdadera, entonces la única posibilidad para $t \wedge k$ es que valga 0.
7. $\mathcal{I}(t) = 0$ por 5 y 6. Por 5, $\mathcal{I}(k) = 1$; si $\mathcal{I}(t \wedge k) = 0$ (por 6) es porque $\mathcal{I}(t) = 0$
8. $\mathcal{I}(\neg t) = 1$ por 7.
9. $\mathcal{I}(f) = 1$ por 2 y 8. Como el antecedente es verdadero en (2), para que la implicación sea verdadera el consecuente tiene que serlo.
10. $\mathcal{I}(\neg f) = 1$ por 3, Tenemos que $\mathcal{I}(\neg f \wedge k) = 1$ y esta interpretación exige $\mathcal{I}(\neg f) = 1$ e $\mathcal{I}(k) = 1$.
11. $\mathcal{I}(f) = 0$ por 10, lo que nos lleva a una contradicción con 9.

Los pasos 9 y 11 generan una contradicción explícita, de manera que por el principio de refutación el conjunto $\Gamma \cup \{\neg b\}$ es insatisfacible y el argumento es correcto.

Ejemplo 2.33. Mostrar la correctud del siguiente argumento conocido como *dilema constructivo simple*: $p \rightarrow r, \neg p \rightarrow r / \therefore r$.

1. $\mathcal{I}(p \rightarrow r) = 1$ Hipótesis
2. $\mathcal{I}(\neg p \rightarrow r) = 1$ Hipótesis
3. $\mathcal{I}(r) = 0$ Refutación
4. $\mathcal{I}(p) = 0$ por 3 y 1. Como $\mathcal{I}(p \rightarrow r) = 1$ e $\mathcal{I}(r) = 0$, $\mathcal{I}(p)$ tiene que ser 0.
5. $\mathcal{I}(\neg p) = 0$ por 3 y 2, argumento similar a 4
6. $\mathcal{I}(p) = 1$ por 5, pero hay contradicción con 4

Por lo tanto el argumento es correcto.

Es importante observar lo siguiente acerca del uso del método de interpretaciones para analizar argumentos:

- Si se usa el método directo, el análisis termina una vez que se logra asignar a la conclusión el valor de verdadero.
- Si se usa el método indirecto, el análisis termina una vez que se logre forzar a que una fórmula tome los dos valores posibles de verdad. Esta fórmula es generalmente una variable proposicional, aunque esto no es la única opción.
- Forzar un valor v para una fórmula A significa que, de acuerdo a la definición de interpretación y a los valores previamente obtenidos de variables o fórmulas, el valor para A es *necesariamente y sin lugar a dudas* el valor v , que puede ser 1 o 0. Por ejemplo, si sabemos que $\mathcal{I}(p \rightarrow q) = 1$ e $\mathcal{I}(q) = 0$, entonces necesariamente $\mathcal{I}(p) = 0$, puesto que si tuviésemos $\mathcal{I}(p) = 1$, la definición de interpretación para la implicación nos llevaría a $\mathcal{I}(p \rightarrow q) = 0$, lo cual sabemos que no sucede. De esta manera el valor de p *está forzado* a ser 0.
Es error común asignar valores que no están forzados; por ejemplo, si sólo sabemos que $\mathcal{I}(r \rightarrow s) = 1$, entonces es un error decir que el valor $\mathcal{I}(s) = 0$ está forzado puesto que no hay suficiente información para descartar la posibilidad de que $\mathcal{I}(r) = 0$, en cuyo caso s podría ser verdadero sin afectar el valor conocido de $r \rightarrow s$.
- Si al usar el método indirecto no es posible hallar una contradicción o si en el método directo no se forzó a que la conclusión sea verdadera, entonces el argumento resulta incorrecto y la interpretación asignada será un *contraejemplo* a la correctud del argumento, puesto que las premisas serán ciertas y la conclusión falsa.

Analizaremos ahora un par de argumentos incorrectos.

Ejemplo 2.34. Analizar el argumento $q \rightarrow p, r \vee s / \therefore r \rightarrow p$.

Procedemos directamente:

1. $\mathcal{I}(q \rightarrow p) = 1$ Hipótesis
2. $\mathcal{I}(r \vee s) = 1$ Hipótesis

En este momento no hay manera de forzar ningún valor puesto que tanto la implicación como la disyunción son verdaderas en tres estados. Esta libertad nos permite asignar valores que causen que la conclusión sea falsa, lo que sucede como sigue:

3. $\mathcal{I}(r) = 1$ Supuesto
4. $\mathcal{I}(p) = 0$ Supuesto

Aún no terminamos, puesto que debemos dar valores a q y s , los cuales pueden obtenerse como sigue:

5. $\mathcal{I}(q) = 0$ por 1 y 4
6. $\mathcal{I}(s) = 0$ Supuesto

De manera que la interpretación dada por $\mathcal{I}(p) = \mathcal{I}(q) = \mathcal{I}(s) = 0$ e $\mathcal{I}(r) = 1$ es un contraejemplo al argumento, pues con esta interpretación $\mathcal{I}(r \rightarrow p) = 0$, ya que $1 \rightarrow 0$ es 0. Esto es, en el estado $\{p = 0, q = 0, s = 0, r = 1\}$, tenemos que

$$((q \rightarrow p) \wedge (r \vee s)) \rightarrow (r \rightarrow p)$$

se evalúa a 0.

				(2)	(3)	(1)	(5)	(4)
p	q	r	s	$q \rightarrow p$	\wedge	$r \vee s$	\rightarrow	$r \rightarrow p$
0	0	1	0	1	1	1	0	0

Obsérvese que s también pudo haber sido verdadero, lo cual habría generado otro contraejemplo.

El método indirecto puede ser de más ayuda en algunos casos, pues obliga desde el principio a forzar algunos valores como en el siguiente ejemplo.

Ejemplo 2.35. Analizar el argumento $q \rightarrow p, r \rightarrow p / \therefore r \vee s$.

Procedemos indirectamente:

1. $\mathcal{I}(q \rightarrow p) = 1$ Hipótesis
2. $\mathcal{I}(r \rightarrow p) = 1$ Hipótesis
3. $\mathcal{I}(r \vee s) = 0$ Refutación
4. $\mathcal{I}(r) = 0$ por 3
5. $\mathcal{I}(s) = 0$ por 3

Obsérvese que falta asignar los valores de p y q . Puede ser que con la asignación $\mathcal{I}(r) = 0$ ya aseguramos que la segunda premisa se mantiene cierta, por lo que el valor de p está libre. Asimismo, el valor de q sólo afecta a la primera premisa y puede elegirse libremente. Un contraejemplo es entonces $\mathcal{I}(r) = \mathcal{I}(s) = \mathcal{I}(q) = \mathcal{I}(p) = 0$. Con estos valores aseguramos que las premisas son verdaderas pero que la conclusión es falsa, por lo que el argumento no es correcto. Otro contraejemplo es $\mathcal{I}(r) = \mathcal{I}(s) = \mathcal{I}(q) = 0, \mathcal{I}(p) = 1$, como se puede verificar de manera muy sencilla.

Algunas observaciones son pertinentes.

- Al usar valores supuestos – no forzados – no es posible afirmar la correctud del argumento al llegar al valor verdadero para la conclusión o al llegar a una contradicción. En este caso esto sólo indica que el valor supuesto debe reconsiderarse. Si se llega al mismo resultado para todos los posibles valores supuestos entonces podremos afirmar la correctud del argumento y sólo hasta ese momento.

- En el caso de llegar a un contraejemplo con un valor supuesto, con éste basta. No es necesario reconsiderar valores supuestos pues el contraejemplo ya está construido.

El método de interpretaciones, si bien es más eficiente en general que el uso de tablas de verdad, requiere de una gran interacción con el usuario, por lo que se antoja difícil de automatizar; es un método muy cercano al razonamiento humano. Más aún, los pasos de razonamiento no siempre son únicos, por ejemplo al usar supuestos, lo cual añade una dificultad más, la elección o no determinismo.

La noción de consecuencia lógica es un concepto semántico de gran importancia que permite analizar argumentos lógicos y además puede generalizarse a otros sistemas lógicos, en contraste con las tablas de verdad. Más aún, el uso de interpretaciones proporciona la base para la búsqueda de contraejemplos a argumentos incorrectos. Sin embargo, no es un método eficiente para encontrar consecuencias dado un conjunto de premisas. Para este propósito es más conveniente construir pruebas o derivaciones de manera sintáctica, es decir, sin apelar al concepto de interpretaciones. Haremos esto en la siguiente sección.

2.6.3. Derivaciones

Muchos argumentos lógicos correctos pueden obtenerse mediante composición de otros argumentos correctos previamente obtenidos, en el sentido de que la conclusión de un argumento previo puede servir como premisa para un siguiente argumento, y así sucesivamente, hasta llegar a una conclusión deseada. Obsérvese que esta composición de argumentos es un mecanismo puramente sintáctico, al no apelar a la noción de verdad o interpretación. Veamos un par de ejemplos.

Ejemplo 2.36. Queremos demostrar que el siguiente fragmento de programa deja el valor de la variable x de tal forma que después de la ejecución es imposible que $x > \text{Max}$, esto es $(x > \text{Max}) \equiv \text{false}$.

```
if  $x > \text{Max}$  then  $x := \text{Max}$ ;
```

Formalizamos con las siguientes variables proposicionales:

p	:	$x > \text{Max}$	antes de la ejecución
q	:	$x = \text{Max}$	después de la ejecución
r	:	$x > \text{Max}$	después de la ejecución

Tenemos que distinguir entre $x > \text{Max}$ antes y después de la ejecución, pues la asignación modifica el valor de la variable x , es decir, x tiene un valor distinto antes y después de la ejecución del programa.

Vamos a hacer primero un análisis intuitivo del problema: hay dos casos, correspondientes a p y $\neg p$. Si p sucede entonces la asignación se lleva a cabo y q se vuelve válida, es decir la implicación $p \rightarrow q$ se cumple. Además, si q es válida entonces $\neg r$ también, pues si los dos

números x y Max son iguales entonces $x > \text{Max}$ es falso, así que la implicación $q \rightarrow \neg r$ es válida. Por otro lado, si $\neg p$ es válida, entonces la asignación no se lleva a cabo y claramente $\neg r$ es cierta, pues en este caso p es equivalente a r , por lo que la implicación $\neg p \rightarrow \neg r$ es válida. Formalmente queremos concluir que $\neg r$, lo cual es posible usando como hipótesis las implicaciones anteriores y aplicando los esquemas de silogismo hipotético (SH) y dilema constructivo simple (DCS), (ver ejemplos 2.4 y 2.33. Procedemos paso a paso como sigue:

Fórmula	Justificación	Comentario
1. $p \rightarrow q$	Hipótesis	Si $x > \text{Max}$ antes de la ejecución entonces $x = \text{Max}$ después de la ejecución
2. $q \rightarrow \neg r$	Hipótesis	Si $x = \text{Max}$ después de la ejecución entonces $x > \text{Max}$ no es cierta después de la ejecución.
3. $\neg p \rightarrow \neg r$	Hipótesis	Si $x > \text{Max}$ no es cierta antes de la ejecución entonces tampoco después de la ejecución
4. $p \rightarrow \neg r$	SH 1,2	Si $x > \text{Max}$ antes de la ejecución entonces $x > \text{Max}$ no es cierta después de la ejecución.
5. $\neg r$	DCS 3,4	Por lo tanto, sin importar si $x > \text{Max}$ es cierta o falsa antes de la ejecución, después de la ejecución $x > \text{Max}$ es falsa.

Se observa que el paso 4, que es la conclusión de una instancia del silogismo hipotético, fue usado además como premisa para lograr una instancia del dilema constructivo simple. Más aún, en ningún momento se apela a la noción de interpretación.

Ejemplo 2.37. Uno de los más reconocidos pensadores “lógicos” es Sherlock Holmes, el detective creado por Arthur Conan Doyle. Veamos una de sus argumentaciones más famosas, que aparece en el libro “Estudio en Escarlata”:

Y ahora llegamos a la gran pregunta del motivo. El robo no fue la razón del asesinato, ya que nada fue sustraído. Entonces, ¿fue la política o fue una mujer? Esta es la pregunta a la que me enfrenté. Me incliné desde un principio a la segunda suposición. Los asesinos políticos hacen su trabajo lo más rápido posible y huyen en cuanto terminan. Este asesinato, en cambio, fue hecho de manera deliberada y el asesino dejó sus huellas en todo el cuarto, mostrando que permaneció ahí mucho tiempo.

Para expresar esta cita, utilizaremos las siguientes variables proposicionales:

- r : fue un robo
 s : algo fue sustraído
 p : fue la política (motivos políticos)
 m : fue una mujer
 h : el asesino huyó inmediatamente
 c : el asesino dejó sus huellas en todo el cuarto

Veamos la derivación que llevó a cabo Sherlock Holmes, y que lo llevó a concluir que fue una mujer, en la tabla 2.11.

Tabla 2.11 Análisis dado por Sherlock Holmes

Derivación	Regla	Comentario
1. $r \rightarrow s$	Premisa	Si fue un robo entonces algo debió ser sustraído
2. $\neg s$	Premisa	Nada fue sustraído
3. $\neg r$	Modus Tollens 1, 2	No fue un robo
4. $\neg r \rightarrow p \vee m$	Premisa	Si no fue un robo, debió ser motivo político o una mujer
5. $p \vee m$	Modus Ponens 3, 4	Fue motivo político o una mujer
6. $p \rightarrow h$	Premisa	Si fue motivo político, el asesino debió huir inmediatamente
7. $c \rightarrow \neg h$	Premisa	Si el asesino dejó huellas en todo el cuarto, no huyó inmediatamente
8. c	Premisa	El asesino dejó huellas en todo el cuarto
9. $\neg h$	Modus Ponens 7, 8	El asesino no huyó inmediatamente
10. $\neg p$	Modus Tollens 6, 9	El motivo no fue político
11. m	Silogismo Disyuntivo 5, 10	Por lo tanto debió ser una mujer

La secuencia de argumentos utilizados se muestra en la lista a continuación. En ella se puede observar claramente como las conclusiones que se van obteniendo de los argumentos, se pueden utilizar como premisas en argumentos sucesivos.

1. $r \rightarrow s$	
2. $\neg s$	
<hr/>	
3. $\neg r$	Modus Tollens
3. $\neg r$	
4. $\neg r \rightarrow p \vee m$	
<hr/>	
5. $p \vee m$	Modus Ponens
7. $c \rightarrow \neg h$	
8. c	
<hr/>	
9. $\neg h$	Modus Ponens
6. $p \rightarrow h$	
9. $\neg h$	
<hr/>	
10. $\neg p$	Modus Tollens
5. $p \vee m$	
10. $\neg p$	
<hr/>	
11. m	Silogismo Disyuntivo

Las secuencias de composición de argumentos que acabamos de mostrar en los ejemplos anteriores se llaman *derivaciones*, *pruebas* o *deducciones formales*. A continuación las estudiamos de manera formal.

Sistemas para derivaciones

Los aspectos de la lógica relacionados con el estudio de las derivaciones conforman lo que se llama *teoría de la demostración* en contraste con los aspectos semánticos cuyo estudio se conoce como *teoría de modelos*. En esta sección describimos formalismos para desarrollar pruebas o derivaciones en lógica proposicional de manera sistemática, los cuales se conocen como cálculos deductivos o sistemas para derivaciones.

Aunque existen diversos sistemas para desarrollar derivaciones, todos tienen las siguientes características en común:

1. Hay un conjunto de argumentos lógicos admisibles, que definimos ya como reglas de inferencia. Nos referiremos a este conjunto con \mathcal{L} . Formalmente cada elemento de \mathcal{L} es en realidad un esquema de argumento, el cual debe ser un argumento correcto. En algunos casos se aceptan argumentos sin premisas los cuales se llaman *axiomas*.

2. La derivación es en sí misma una lista de expresiones lógicas. Originalmente, la lista está vacía y una expresión puede agregarse a la lista si es una premisa, o si se puede obtener como conclusión de alguna de las reglas de inferencia de \mathcal{L} a partir de expresiones que se encuentran previamente en la lista. Este proceso continúa hasta que se llega a la fórmula B que se desea obtener como conclusión. En tal caso decimos que la lista completa es una derivación de B .

Estas características describen el conocido método axiomático introducido por Euclides en sus “*Elementos*” donde están las bases de la geometría euclidea.

La siguiente definición es de importancia.

Definición 2.15 Sean $\Gamma = \{A_1, \dots, A_n\}$ un conjunto de fórmulas. Si existe una derivación de B a partir de Γ , es decir, donde las premisas son fórmulas del conjunto Γ , entonces decimos que B es derivable a partir de Γ y escribimos $\Gamma \vdash_{\mathcal{L}} B$, o simplemente $\Gamma \vdash B$ si el conjunto de reglas de inferencia válidas ya es conocido.

Por lo general el conjunto de reglas de inferencia \mathcal{L} está fijo desde un principio, de manera que únicamente pueden usarse reglas de inferencia que figuran en él. En nuestro caso no seremos tan estrictos y permitiremos usar cualquier regla previamente derivada, aunque esencialmente usaremos las siguientes:

Tabla 2.13 Principales reglas de inferencia

(1/2)

Regla	Nombre	Notación
$A \quad B / A \wedge B$	Introducción de \wedge	$I\wedge$
$A \wedge B / B$	Eliminación de \wedge	$E\wedge$
$A \wedge B / A$	Eliminación de \wedge	$E\wedge$
$A / A \vee B$	Introducción de \vee	$I\vee$
$B / A \vee B$		$I\vee$
$A \quad A \rightarrow B / B$	Modus Ponens	MP
$\neg B \quad A \rightarrow B / \neg A$	Modus Tollens	MT
$A \rightarrow B \quad B \rightarrow C / A \rightarrow C$	Silogismo Hipotético	SH

(Continúa en la siguiente página)

Tabla 2.14 Principales reglas de inferencia

(2/2)

(Continúa de la página anterior)

Regla	Nombre	Notación
$A \vee B \quad \neg A / B$	Silogismo Disyuntivo	SD
$A \vee B \quad \neg B / A$		SD
$A \rightarrow B \quad \neg A \rightarrow B / B$	Casos simple	CS
$A \leftrightarrow B / A \rightarrow B$	Eliminación de equivalencia	$E\leftrightarrow$
$A \leftrightarrow B / B \rightarrow A$		$E\leftrightarrow$
$A \rightarrow B \quad B \rightarrow A / A \leftrightarrow B$	Introducción de Equivalencia	$I\leftrightarrow$
$A, \neg A / B$	Inconsistencia	Inc

Es momento de desarrollar algunos ejemplos.

Ejemplo 2.38. Mostrar la correctud del siguiente argumento:

$$p \rightarrow r, r \rightarrow s, t \vee \neg s, \neg t \vee u, \neg u / \therefore \neg p.$$

Vamos a desarrollar una derivación de $\neg p$ con premisas

$$\Gamma = \{p \rightarrow r, r \rightarrow s, t \vee \neg s, \neg t \vee u, \neg u\}.$$

Derivación:

1. $p \rightarrow r$ Premisa
2. $r \rightarrow s$ Premisa
3. $t \vee \neg s$ Premisa
4. $\neg t \vee u$ Premisa
5. $\neg u$ Premisa
6. $p \rightarrow s$ (SH) Silogismo hipotético con 1, 2
7. $\neg t$ (SD) Silogismo disyuntivo con 4, 5
8. $\neg s$ (SD) Silogismo disyuntivo con 7, 3
9. $\neg r$ (MT) Modus tollens con 2, 8
10. $\neg p$ (MT) Modus tollens con 9, 1

Ejemplo 2.39. Mostrar la correctud del siguiente argumento

$$p \rightarrow q, q \rightarrow r \wedge s, \neg r \vee \neg t \vee u, p \wedge t / \therefore u$$

1.	$p \rightarrow q$	Premisa
2.	$q \rightarrow r \wedge s$	Premisa
3.	$\neg r \vee \neg t \vee u$	Premisa
4.	$p \wedge t$	Premisa
5.	$p \rightarrow r \wedge s$	SH 1,2
6.	p	E \wedge 4
7.	$r \wedge s$	MP 5,6
8.	r	E \wedge 7
9.	$\neg t \vee u$	SD 8,3
10.	t	E \wedge 4
11.	u	SD 9,10

Ejemplo 2.40. Mostrar la correctud del siguiente argumento:

Si la banda no puede tocar cumbia o las cervezas no llegan temprano, entonces la fiesta de fin de semestre se cancelaría y Menelao montaría en cólera. Si la fiesta se cancela, hay que devolver las entradas. No se devolvieron las entradas. Luego entonces la banda pudo tocar cumbia.

Se asignan las siguientes variables proposicionales:

- b : La banda pudo tocar cumbia
- c : Las cervezas llegan temprano
- f : La fiesta se cancela
- m : Menelao monta en cólera
- d : Hubo que devolver el dinero

El argumento a verificar es: $\neg b \vee \neg c \rightarrow f \wedge m, f \rightarrow d, \neg d / \therefore b$.

1.	$\neg b \vee \neg c \rightarrow f \wedge m$	Premisa
2.	$f \rightarrow d$	Premisa
3.	$\neg d$	Premisa
4.	$\neg f$	MT 2, 3
5.	$\neg f \vee \neg m$	I \vee 4
6.	$\neg(f \wedge m)$	RE 5
7.	$\neg(\neg b \vee \neg c)$	MT 6,1
8.	$\neg\neg b \wedge \neg\neg c$	RE 7
9.	$b \wedge c$	RE 8
10.	b	E \wedge 9

Se observa en los pasos 6, 8 y 9 el uso de razonamiento ecuacional (RE); muchas veces éste se da por sobreentendido y no se menciona, por lo que podríamos haber pasado del paso 5 al 7 o del paso 7 al 9 directamente.

Estrategias para la construcción de derivaciones

En esta sección presentamos algunas estrategias o métodos para la derivación de argumentos correctos. La meta es construir una derivación $\Gamma \vdash B$. De acuerdo al conectivo principal de la conclusión B de un argumento, podemos simplificar la derivación del mismo.

Conjunción

Para derivar una conjunción $\Gamma \vdash P \wedge Q$ basta derivar ambos operandos por separado. Es decir,

- Si $\Gamma \vdash P$ y $\Gamma \vdash Q$, entonces $\Gamma \vdash P \wedge Q$

Esta propiedad es inmediata de la regla de inferencia ($\wedge I$). Obsérvese que la afirmación recíproca también es cierta.

Disyunción

De acuerdo a la regla de introducción de la disyunción ($\vee I$), para mostrar $\Gamma \vdash P \vee Q$ basta mostrar alguno de los dos operandos. Es decir,

- Si $\Gamma \vdash P$ o bien $\Gamma \vdash Q$, entonces $\Gamma \vdash P \vee Q$.

En este caso la afirmación recíproca no es necesariamente cierta; por ejemplo, tenemos $p \vee q \vdash p \vee q$ pero no es posible derivar $p \vee q \vdash p$ ni $p \vee q \vdash q$.

Implicación

Cuando tratamos de derivar una implicación basta suponer como premisa adicional el antecedente y derivar a partir de ello el consecuente. Esto se debe a que para mostrar la verdad de una implicación basta examinar aquellos casos en que el antecedente es verdadero y corroborar que de ese antecedente se infiere el consecuente; si el antecedente es falso, la implicación es verdadera no importando el valor del consecuente. Esto se expresa en la siguiente propiedad conocida como el *metateorema de la deducción* :

- Si $\Gamma, P \vdash Q$ entonces $\Gamma \vdash P \rightarrow Q$.

Obsérvese que esta regla se usa prácticamente siempre en las demostraciones matemáticas en general.

Ejemplo 2.41. Supongamos que deseamos demostrar

$$\vdash P \rightarrow P \vee Q$$

Utilizando la propiedad anterior basta encontrar una derivación

$$P \vdash P \vee Q$$

la cual es inmediata de la regla de introducción de la disyunción ($\vee I$).

Equivalencia

Para derivar una equivalencia $P \leftrightarrow Q$ basta probar ambas implicaciones.

- Si $\Gamma \vdash P \rightarrow Q$ y $\Gamma \vdash Q \rightarrow P$, entonces $\Gamma \vdash P \leftrightarrow Q$.

Nuevamente esta propiedad es muy común en demostraciones matemáticas.

Negación

Para derivar una negación no hay estrategia general. En algunos casos podemos usar equivalencias lógicas, por ejemplo si deseamos $\Gamma \vdash \neg(P \wedge Q)$ entonces basta mostrar $\Gamma \vdash \neg P \vee \neg Q$; para demostrar esto último podemos usar la estrategia para la disyunción y probar alguna de $\Gamma \vdash \neg P$ o bien $\Gamma \vdash \neg Q$.

Un sistema de derivación \mathcal{L} debe ser tal que no se puedan derivar resultados que no son sólidos. Esto es, \mathcal{L} no debe contener ninguna *falacia*, una regla de inferencia que permite concluir algo que no está implicado por las premisas y que por lo tanto no es válido.

Un sistema de derivación también debe ser *completo*, esto es, que sea posible derivar absolutamente a todas las conclusiones que se puedan inferir de las premisas. Por ejemplo, la tabla 2.13 no nos da un sistema completo, pues hay leyes, como la del Tercero Excluido, que no se puede derivar de ellas. Y como no hay forma de derivar esta ley a partir de las que se dan en la tabla, debemos agregarla como premisa:

$$\vdash P \vee \neg P$$

Ejercicios

2.6.1.- Usa los identificadores P y Q para formalizar los siguientes argumentos. Además indica de cuál de las reglas de inferencia son instancia.

- a) Si 10 es primo, 10 no puede ser igual a 2 veces 5. 10 es 2 veces 5. Por lo tanto, 10 no puede ser primo.
- b) Si llueve frecuentemente, los agricultores se quejan; si no llueve frecuentemente, los agricultores se quejan. En conclusión, los agricultores se quejan.

2.6.2.- Para los siguientes argumentos decide si son correctos y en caso de no serlo da un interpretación que haga verdaderas a las premisas y falsa a la conclusión.

- (a) $(p \rightarrow q) \wedge (p \rightarrow r) / \therefore q \rightarrow r$
- (b) $p \vee q \rightarrow r, s \rightarrow p, s / \therefore r$
- (c) $p \vee q, \neg(p \wedge r), \neg q / \therefore r \rightarrow s$
- (d) $p \rightarrow q, p \vee r, \neg(r \wedge s) / \therefore (p \rightarrow q) \rightarrow (q \vee \neg s)$

2.6.3.- Da un ejemplo, en español, para cada uno de las siguientes reglas de inferencia

- a) Silogismo hipotético.
- b) Silogismo disyuntivo.
- c) Eliminación de \wedge .
- d) Introducción de \vee .
- e) Inconsistencia.

2.6.4.- Identifica qué regla de inferencia corresponde a los siguientes argumentos en español.

- a) Si vamos al cine, nos desvelamos. No me quiero desvelar. Entonces no vamos al cine.
- b) ¡Me pagas la deuda o te quito la televisión! No me pagaste la deuda. Entonces te quito la televisión.
- c) Si el número de visitas es a lo más 15, estarán todos en la sala. Hay visitas en la recámara. Es porque vinieron más de 15.
- d) Ese muchacho se llama Juan o Pedro. No se llama Juan. Entonces se llama Pedro.

2.6.5.- Construye las siguientes derivaciones

- (a) $p \rightarrow q, r \rightarrow s, \neg q \vee \neg s \vdash \neg p \vee \neg r$
- (b) $\vdash p \vee (p \wedge \neg q \rightarrow r)$
- (c) $\vdash p \vee (\neg p \wedge q) \rightarrow p \vee q$
- (d) $\vdash (p \rightarrow q) \rightarrow (p \vee q \rightarrow q)$
- (e) $\vdash (\neg p \wedge (\neg p \wedge q)) \vee (p \wedge (p \wedge \neg q)) \leftrightarrow (\neg p \wedge q) \vee (p \wedge \neg q)$

2.7. Tableaux semánticos para el cálculo proposicional

Una de las preocupaciones de la lógica proposicional (y de la de predicados que veremos más adelante) es la de determinar si una fórmula bien formada⁴ (*fbf*) es o no *razonable*. Esto último quiere decir que deseamos determinar si existe algún estado en el que la fórmula se evalúe a verdadero; o dicho de otra manera, si hay alguna asignación posible a las variables proposicionales que participan en la fórmula de tal manera que ésta se evalúa a verdadera (dicho de una tercera forma, si la fórmula tiene modelo).

Uno de los mecanismos que podemos utilizar para determinar si una fórmula es razonable es la de elaborar la tabla de verdad de la misma. Sin embargo, como ya hemos mencionado, la tarea de elaborar tablas de verdad cuando estamos hablando de fórmulas de más de tres o cuatro variables se vuelve un problema intratable, ya que tendremos que examinar 2^n posibles estados.

Un mecanismo que permite de manera eficiente y segura determinar si una fórmula es tautología, contradicción o contingencia, y encontrar un estado para el cual la fórmula se evalúa a verdadera son los *tableaux*.

2.7.1. El concepto de tableau

Un *tableau* corresponde a un árbol cuya función es buscar una interpretación para determinada fórmula. Los tableaux toman la forma de un árbol, parecido a los árboles de derivación. Las fórmulas que van a ser representadas en un tableau deben consistir únicamente de conjunciones y disyunciones de literales, que son fórmulas atómicas (*true*, *false*, p , q , r , ...) o negaciones de ellas (\neg *true*, \neg *false*, $\neg p$, $\neg q$, $\neg r$, ...). Estas fórmulas no pueden tener ningún otro operador, pero esto no nos debe preocupar ya que vimos que es posible eliminar la implicación y la bicondicional sustituyéndolas por disyunciones y conjunciones. También podemos eliminar la negación de una fórmula disyuntiva o conjuntiva ($\neg(p \wedge q)$) utilizando

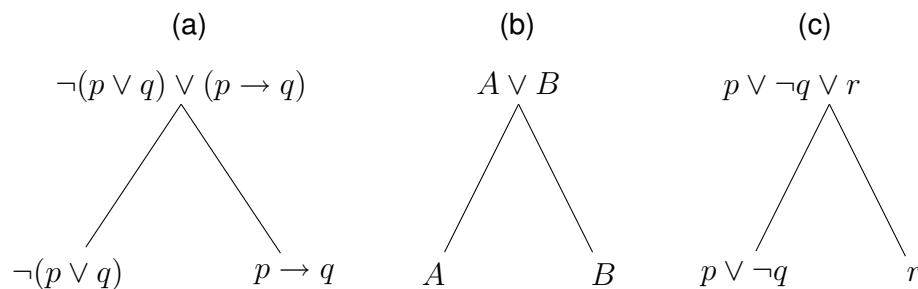
⁴En inglés *well-formed formula* (*wff*)

las leyes de De Morgan ($\neg(p \wedge q) \equiv \neg p \vee \neg q$). Es importante, sin embargo, mantener la asociatividad de los operadores dada por la fórmula original (preservar la precedencia original o sea trabajar con fórmulas donde todos los paréntesis que indican precedencia son explícitos).

La construcción de tableaux tiene realmente muy pocas reglas. Veámoslas:

1. La fórmula para la que deseamos construir el tableau aparece como raíz del árbol.
2. Si el esquema de la fórmula es una disyunción ($A \vee B$), de la raíz del subárbol se abren dos ramas, una para la fórmula A y otra para la fórmula B , como podemos ver en la figura 2.4.

Figura 2.4 Construcción de tableau para la disyunción

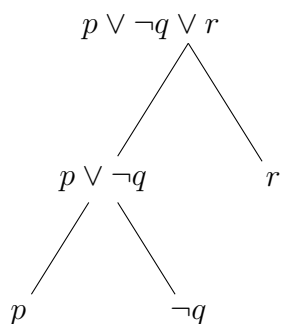


Como el operador \vee es conmutativo y asociativo, se pueden intercambiar el orden de las ramas de los árboles. También utilizamos la propiedad asociativa de la disyunción en el caso de la fórmula del tableau 2.4(c) y decidimos “abrir” primero la segunda disyunción.

Por lo pronto, dejamos a los tableaux desarrollados únicamente en el primer nivel, lo que deja ramas que deben ser expandidas en el primer y tercer caso. Más adelante veremos cuándo y cómo conviene extender un tableau. Conforme se avanza en la fórmula, se va “componiendo” con el árbol que se tiene hasta ese momento. Lo que debe quedar claro es que en la fórmula 2.4(a) no podemos extender, tal como están, a ninguna de las fórmulas en el segundo nivel del árbol, ya que no corresponden a esquemas de disyunción o conjunción; en esta expresión, la fórmula de la izquierda corresponde a un esquema de negación, mientras que la segunda es una condicional; así que por lo pronto posponemos su extensión hasta que demos las reglas de transformación para tableaux. En cambio, en la fórmula 2.4(c) sí tenemos en la rama izquierda un esquema de disyunción, por lo que ya podemos expandirla, quedando el

tableau como se muestra en la figura 2.5.

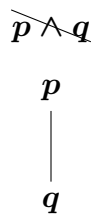
Figura 2.5 Desarrollo completo del tableau de la fórmula 2.4(c)



3. Si el esquema de la fórmula es una conjunción ($A \wedge B$) se pone a uno de los operandos como hijo del otro (como el operador \wedge es conmutativo, el orden no importa). Podemos ver tres ejemplos en las figuras 2.6 a 2.8.

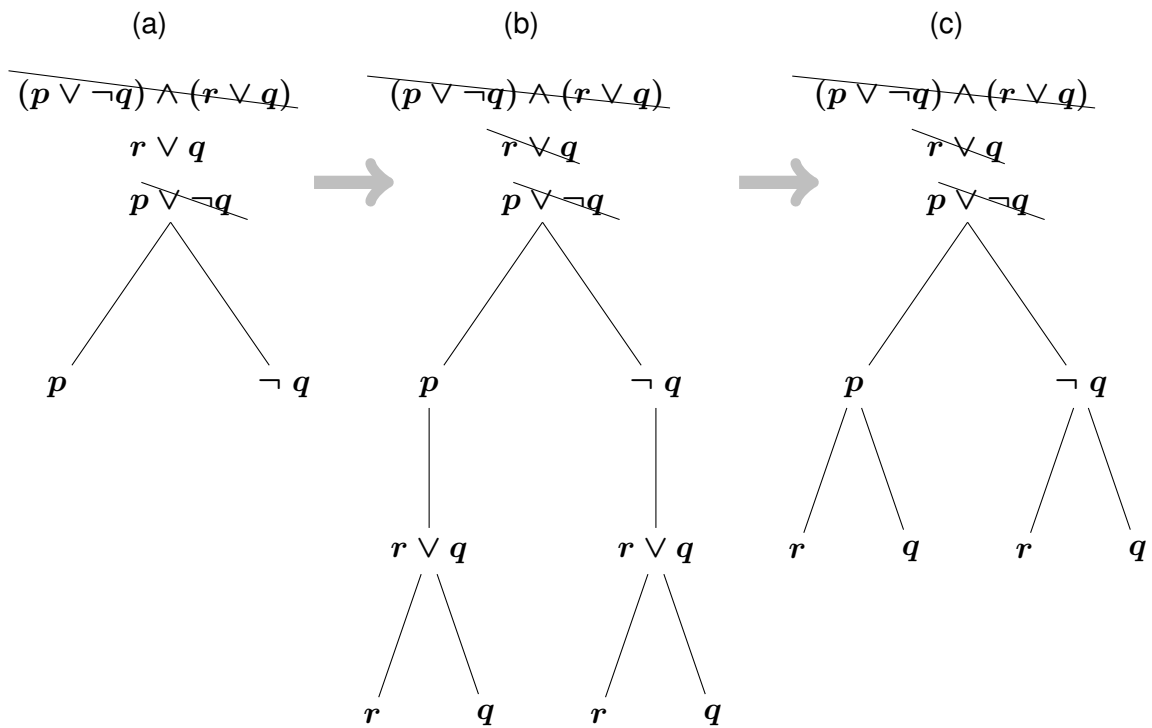
En la fórmula de la figura 2.6 tenemos un esquema de conjunción, donde cada uno de los operandos es una variable proposicional.

Figura 2.6 Primer ejemplo de tableau para representar conjunciones

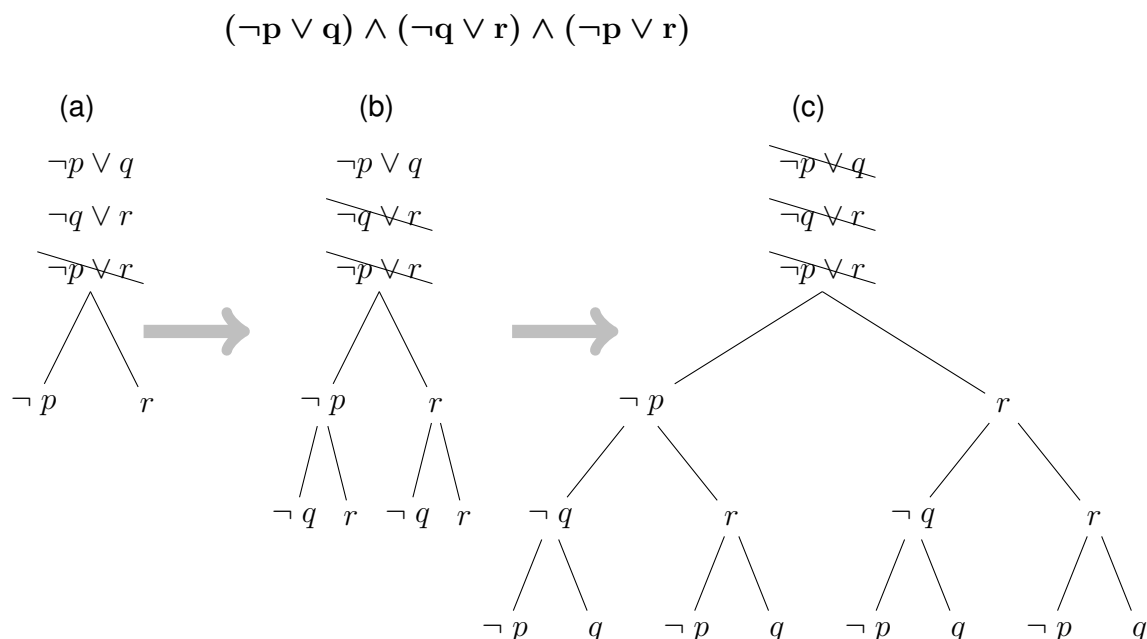


En la fórmula de la figura 2.7 abajo, tenemos un esquema de conjunción donde cada operando es, a su vez, una disyunción. Entonces, listamos los dos operandos, uno abajo del otro (el orden no importa) y procedemos a construir el tableau para uno de ellos, en este caso el primero. Una vez que tenemos en el tableau como hojas únicamente variables proposicionales que ya no pueden descomponerse más, colgamos de cada una de las ramas al otro operando y procedemos a abrirlo. Mostramos en el tableau de la figura 2.7(b) el nivel intermedio para la fórmula $r \vee q$, aunque esto no es necesario, sino que podríamos haber colgado directamente la conjunción, como se ve en el tercer tableau de esta fórmula.

Figura 2.7 Segundo ejemplo de tableau para representar conjunciones



Para la tercera fórmula tenemos también un esquema de conjunción. Como la conjunción es asociativa, podemos asociar $((\neg p \vee q) \wedge (\neg q \vee r)) \wedge (\neg p \vee r)$, que es como lo hicimos, o pudiéramos usar también la conmutatividad de este operador. Listamos los tres operandos uno abajo del otro y desarrollamos el tableau del último $(\neg p \vee r)$ como primer paso. A continuación colgamos de todas las ramas de este tableau al segundo operando $(\neg q \vee r)$ y lo tachamos – ya no pusimos la subfórmula original explícitamente en el árbol –. Una vez que tenemos únicamente variables proposicionales como hojas del tableau, como tercer paso colgamos de cada una de las ramas a la primera fórmula $(\neg p \vee q)$. El tableau construido de esta manera es el último en la figura 2.8.

Figura 2.8 Tercer ejemplo de tableaux con disyunción

Este caso es, claramente, un poco más complicado que el caso de la bifurcación. La intención con la que se construyeron los árboles (tableaux) es la de que, como se trata de una conjunción, cualquier “camino” en el árbol debe contemplar a todos los operandos de la conjunción. En el primer ejemplo, simplemente tenemos dos variables proposicionales, por lo que las ponemos en el árbol a una de ellas como descendiente de la otra. En el segundo ejemplo, desarrollamos uno de los operandos de la disyunción y de cada hoja, en la que hay únicamente variables proposicionales, “colgamos” a la otra proposición desarrollada como tableau. Como el tableau para $r \vee q$ es un tableau con dos ramas, éste se cuelga tanto de p como de $\neg q$.

El tercer ejemplo consiste de dos operadores \wedge (tres operandos). En el primer nivel colocamos (es arbitraria esta elección) al tercer operando. Una vez que lo desarrollamos completo, colgamos de cada una de las ramas el segundo operando, a su vez desarrollado ya en un tableau; por último, tenemos que colocar el operando que nos falta, $\neg p \vee q$, colgándolo de cada una de las ramas que llevamos hasta el momento. El orden no es importante, siempre y cuando hayamos incluido para desarrollar a todas las subfórmulas en la manera en que indicamos.

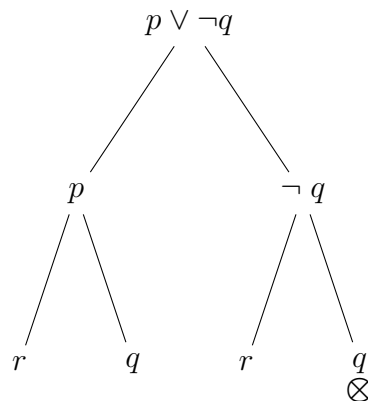
Dado que únicamente tenemos reglas de construcción para la disyunción y la conjunción, debemos decidir qué hacer con aquellas fórmulas que involucren otros operadores. Tenemos dos opciones: transformar la fórmula antes de construir el tableau, usando propiedades de los operadores, asociatividad, conmutatividad y las Leyes de De Morgan, y

proceder después a desarrollar el tableau de la fórmula resultante. Otra opción es ir transformando las subfórmulas durante la construcción del tableau. Esta estrategia nos puede ahorrar trabajo por razones que no tardaremos en explicar. Además, es la que más se beneficia del uso de tableaux.

2.7.2. Eliminación de ramas del tableau

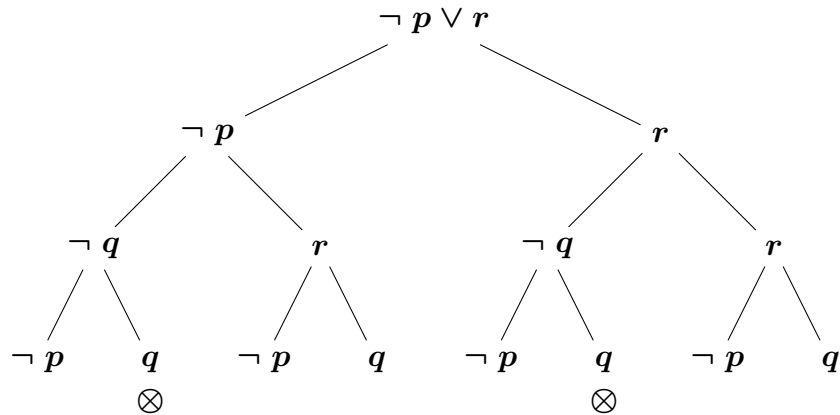
Como dijimos antes, vamos a utilizar los tableaux para determinar si una fórmula es satisfacible o no. Por como están contruidos, si seguimos un camino dentro del tableau vamos a tener la conjunción de variables proposicionales. Por ejemplo, en el caso del tableau de la figura 2.6 simplemente tenemos que el único camino en el árbol es salir de p y llegar a q . Pero en el caso del tableau de la figura 2.7, el camino $(p \vee \neg q) \wedge (r \vee q)$ nos indica la subfórmula $\neg q \wedge q$, que es una contradicción, por lo que “siguiendo” esa rama ya no va a satisfacer a la fórmula (la conjunción será evaluada a falso). Cada vez que encontramos, en un camino (una rama) dentro del árbol, una literal y su literal complementaria⁵, podemos *cerrar* esa rama y ya no extenderla más, pues no importa qué fórmulas le colguemos a ese camino tendremos una conjunción con falso, lo que hace a la fórmula representada por ese camino falsa. Denotamos que un camino está cerrado colocando el símbolo \otimes . En la figura 2.7 se habría eliminado una rama, como se puede observar en la figura 2.9.

Figura 2.9 Cierre de ramas en la figura 2.7

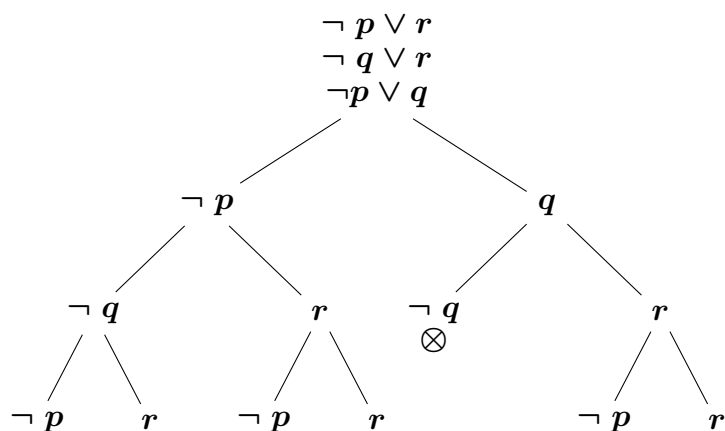


Por ejemplo, el tableau de la figura 2.8 no presenta ninguna rama cerrada que nos ahorre trabajo – ver figura 2.10 –.

⁵La literal complementaria de una literal dada L se define como A si L es $\neg A$ y como $\neg A$, si L es A .

Figura 2.10 Cierre de ramas en un tableau

Sin embargo, hasta ahora únicamente hemos podido cerrar ramas que ya están totalmente desarrolladas, y lo que queremos es *ahorrar* trabajo, esto es, cerrar ramas lo antes posible. Si hubiésemos seguido otro orden en la doble conjunción en la fórmula de la figura 2.8, buscando que aparezcan lo antes posible una literal y su complementaria, habríamos podido llevar a cabo menos trabajo. Por ejemplo, si el orden en que colgamos del tableau es $(\neg p \vee q)$, $(\neg q \vee r)$ y por último $(\neg p \vee r)$, tenemos lo antes posible la contradicción, como se muestra en la figura 2.11.

Figura 2.11 Orden de armado del tableau para cerrar lo más pronto posible

Las ramas que cerremos ya no tiene sentido seguir expandiéndolas y eso nos va a ahorrar trabajo. Si todas las ramas quedan cerradas la fórmula es una contradicción; sin em-

bargo, el que todas las ramas queden abiertas **no** significa que tenemos una tautología. Como ejemplo veamos la fórmula de la figura 2.6, donde todas sus ramas (exactamente una) quedaron abiertas y, sin embargo, esta fórmula sólo será verdadera en el caso en que $\mathcal{I}(p) = \mathcal{I}(q) = 1$; en caso de que algunas ramas queden abiertas y otras cerradas se trata de una fórmula contingente. Para determinar si una fórmula es tautología tenemos que construir el tableau para su negación; si en este tableau se cierran todas las ramas, tenemos que la negación de la fórmula es contradicción y por lo tanto la original es tautología.

2.7.3. Reglas para los tableaux

Vamos a optar por ir “abriendo” las fórmulas conforme las vamos incluyendo en el tableau; la razón para ello es que si nos encontramos con ramas cerradas antes de agregar alguna conjunción, nos ahorramos el trabajo de transformar la regla. Como en el caso del razonamiento ecuacional y de la sustitución textual, es muy importante determinar cuál es el esquema principal que estamos procesando: cuál es el operador que domina. Las reglas que podemos usar para transformar las fórmulas y poderlas agregar al tableau en desarrollo se encuentran a continuación.

- α -reglas:

1. De $A \wedge B$ se deduce A y B . $\alpha(1)$
2. De $\neg(A \vee B)$ se deduce $\neg A$ y $\neg B$. $\alpha(2)$
3. De $\neg(A \rightarrow B)$ se deduce A y $\neg B$. $\alpha(3)$

- β -reglas:

1. De $A \vee B$ se deduce A y, en una rama separada, B . $\beta(1)$
2. De $\neg(A \wedge B)$ se deduce $\neg A$ y, en una rama separada, $\neg B$. $\beta(2)$
3. De $A \rightarrow B$ se deduce $\neg A$ y, en una rama separada, B . $\beta(3)$

- σ -reglas:

1. De $\neg\neg A$ se deduce A . $\sigma(1)$
2. De $\neg\text{false}$ se deduce true . $\sigma(2)$
3. De $\neg\text{true}$ se deduce false . $\sigma(3)$

Las reglas σ son auxiliares y pueden evitarse usando razonamiento ecuacional.

- Reglas de cierre:

1. Cerrar cualquier rama que tenga A y $\neg A$ (para cualquier A),
o bien tenga $\neg \text{true}$, o false . (*cierre*)

Veamos algunos ejemplos de construcción de tableaux.

Ejemplo 2.42.

Demostrar $\vdash ((p \rightarrow q) \rightarrow p) \rightarrow p$, construyendo el tableau correspondiente a su negación. De lo anterior, usando $\alpha(3)$ tenemos:

$$\neg \left(((p \rightarrow q) \rightarrow p) \rightarrow p \right) = \left(((p \rightarrow q) \rightarrow p) \wedge \neg p \right),$$

por lo que pasamos a desarrollar el tableau de esta conjunción en la figura 2.12.

Figura 2.12 Construcción del tableau para el ejemplo 2.42

Fórmula:

Regla usada:

$$\neg \left(((p \rightarrow q) \rightarrow p) \rightarrow p \right) \equiv \left(((p \rightarrow q) \rightarrow p) \wedge \neg p \right)$$

$\alpha(3)$

$$(p \rightarrow q) \rightarrow p$$

$\alpha(1)$

$$\neg p$$

$$(p \rightarrow q) \rightarrow p$$

$\beta(3)$

$$\neg (p \rightarrow q) \vee p$$

$$\neg (p \rightarrow q) \quad p$$

$\alpha(3)$

cierre

$$p \wedge \neg q$$

$$p$$

$$\otimes$$

cierre

Vemos que no queda ninguna rama abierta, lo que denota a una contradicción. Como el tableaux se armó para la negación de la fórmula original y tenemos una contradicción para esta fórmula, podemos deducir que la fórmula original es una tautología.

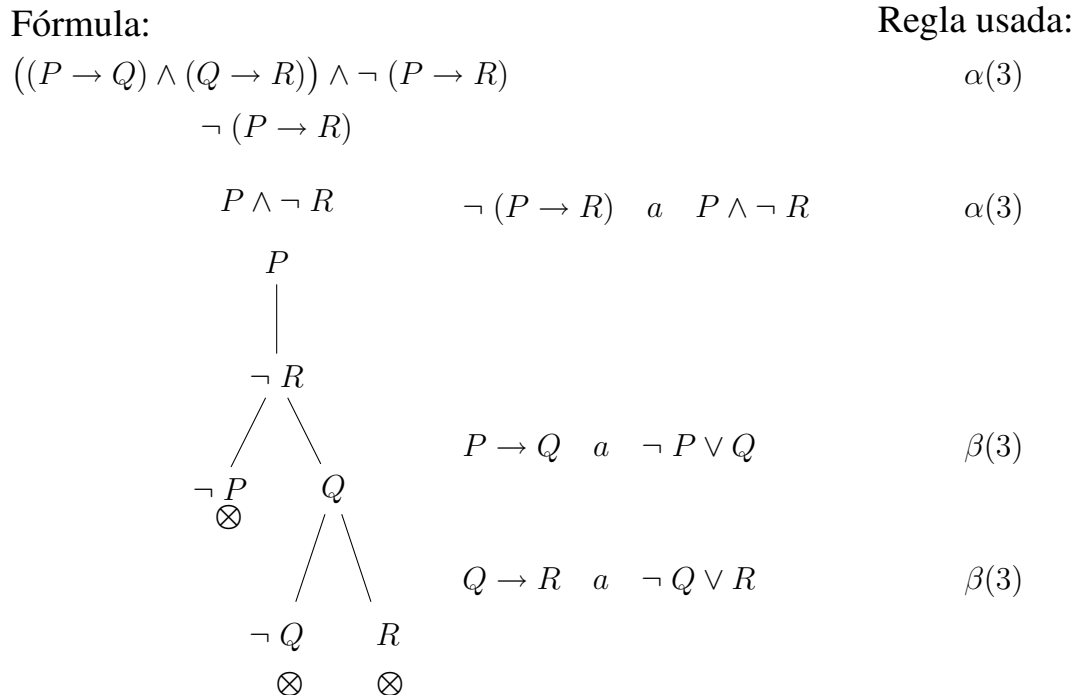
Ejemplo 2.43. Demostrar que el silogismo hipotético es una tautología:

$$\left(((P \rightarrow Q) \wedge (Q \rightarrow R)) \rightarrow (P \rightarrow R) \right)$$

Para demostrar que esta fórmula es tautología trabajamos con su negación:

$$\neg \left(((P \rightarrow Q) \wedge (Q \rightarrow R)) \rightarrow (P \rightarrow R) \right)$$

Ver figura 2.13 en la siguiente página para el desarrollo del tableau correspondiente.

Figura 2.13 Construcción del tableau para el ejemplo 2.43

Como todas las ramas están cerradas la fórmula es una contradicción y, por lo tanto, la fórmula original es tautología (lo que ya sabíamos).

2.7.4. Modelo de una fórmula

Al desarrollar un tableau para una fórmula dada trataremos de trabajar lo menos posible, esto es, abrir el menor número de ramas posibles. Ya vimos que una rama cerrada no tiene sentido seguirla extendiendo; las estrategias usadas deberán ir en la dirección de cerrar lo antes posible una rama. Estas estrategias las podemos resumir de la siguiente manera:

1. Descomponer primero las fórmulas que no abran ramas; es decir, usar las α -reglas y las σ -reglas antes que las β -reglas.
2. Dar prioridad a la descomposición de fórmulas que cierren ramas.
3. Parar cuando el problema esté resuelto (para demostrar satisfacibilidad basta con encontrar una rama abierta completa).
4. Cuando no sirvan las estrategias anteriores, empezar por las fórmulas más complejas (habrá luego menos ramas en las que desarrollar la fórmula compleja).

El tableau de una fórmula también nos proporciona una interpretación para la fórmula que es modelo de la misma. De hecho, cada rama completa que queda abierta corresponde a una interpretación de la fórmula. Por lo tanto, para encontrar un modelo de una fórmula basta encontrar una rama abierta completa. La interpretación que corresponde a esa rama es como sigue:

1. Las variables que aparecen negadas en esa rama se les asigna el valor 0.
2. Las variables que aparecen sin negar en esa rama se les asigna el valor 1.
3. Aquellas variables que aparecen en la fórmula pero no en esa rama pueden tener cualquier asignación.

Nótese que no puede haber ninguna variable a la que se tuviera que asignar 0 y 1 en una misma rama, porque esto querría decir que aparece negada y sin negar, en cuyo caso la rama se habría cerrado.

2.7.5. Algoritmos para la lógica proposicional

Los tableaux son muy útiles en lógica proposicional pues proporcionan diversos algoritmos de decisión, algunos de los cuales enunciamos a continuación.

- ¿Es A una tautología?

Objetivo: Definir si A es tautología.

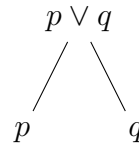
Entrada: A .

Salida: La decisión de si A es o no tautología.

Método:

- Construir el tableau \mathcal{T} para $\neg A$.
- Si \mathcal{T} se cierra entonces A es tautología.
- En otro caso existe una rama abierta y completa en \mathcal{T} la cual genera un modelo de $\neg\varphi$, por lo que φ no es tautología.

Para saber si una fórmula es tautología (o para demostrarlo), construimos el tableau de la negación de la fórmula; si este tableau corresponde a una contradicción, entonces la fórmula es tautología. Este algoritmo se puede adaptar fácilmente para obtener uno que decida si A es una contradicción, observando que una fórmula es una contradicción si y sólo si todas las ramas de su tableau se cierran. Sin embargo, obsérvese que no podemos decir que una fórmula es tautología si todas sus ramas quedan abiertas, porque pudiera haber interpretaciones que no fueran modelo. Por ejemplo, la fórmula $p \vee q$, cuyo tableau aparece a continuación,



es un tableau (muy simple) en el que todas las ramas quedaron abiertas y sin embargo no es tautología. Lo único que podemos concluir, respecto a interpretaciones, es que $\mathcal{I}_1(p) = 1$ e $\mathcal{I}_2(q) = 1$ son modelos de esta fórmula (la variable que no se menciona en cada una de las interpretaciones puede tomar cualquier valor).

- Si se desea clasificar una fórmula en tautología, contradicción o contingencia se usa el siguiente algoritmo.

Objetivo: Clasificar una fórmula A .

Entrada: Una fórmula A que deseamos clasificar como tautología, contradicción o contingencia.

Salida: El dictamen de si la fórmula es tautología, contradicción o contingencia.

Método:

- Construir el tableau \mathcal{T} de A .
- Si \mathcal{T} se cierra entonces A es contradicción.
- En otro caso, existe una rama abierta y completa en \mathcal{T} que proporciona un modelo \mathcal{I} de A .
 - Construir el tableau \mathcal{T}' para $\neg A$.
 - Si \mathcal{T}' se cierra entonces A es tautología.
 - En otro caso \mathcal{T}' tiene una rama abierta y completa que proporciona un modelo \mathcal{I}' de $\neg A$.
- Las interpretaciones \mathcal{I} e \mathcal{I}' muestran que A es contingente.

Con respecto a conjuntos de fórmulas tenemos los siguientes algoritmos.

- Satisfacibilidad de un conjunto de fórmulas Γ .

Objetivo: Decidir la satisfacibilidad de un conjunto de fórmulas Γ

Entrada: Un conjunto de fórmulas $\Gamma = \{A_1, \dots, A_n\}$.

Salida: La decisión de si Γ es o no satisfacible.

Método:

- Construir el tableau \mathcal{T} para $A_1 \wedge A_2 \wedge \dots \wedge A_n$.
 - Si \mathcal{T} se cierra entonces Γ es insatisfacible.
 - En otro caso existe una rama abierta y completa en \mathcal{T} , la cual genera un modelo de Γ , por lo que este conjunto es satisfacible.
- ¿Es A consecuencia lógica de Γ ?

Objetivo: Decidir la consecuencia lógica $\Gamma \models A$.

Entrada: Un conjunto de fórmulas Γ y una fórmula A .

Salida: La decisión de si A es consecuencia lógica de Γ (sí o no).

Método:

- Construir el tablero \mathcal{T} para el conjunto $\Gamma \cup \{\neg A\}$.
- Si \mathcal{T} se cierra entonces la consecuencia lógica $\Gamma \models A$ se da y el argumento que representa es correcto.
- En otro caso, existe una rama abierta y completa en \mathcal{T} por lo que la consecuencia es inválida y se genera un modelo de las premisas Γ donde la conclusión A es falsa.

Como ya se ha visto, los tableaux son un mecanismo para derivación de fórmulas y demostración de teoremas que resulta mucho más económico, en términos de trabajo, que el razonamiento ecuacional, por interpretaciones o mediante derivaciones; adicionalmente, es algorítmico, ya que siempre termina y no hay que ser creativos en el orden de abrir los tableaux; en el peor de los casos, haremos un poco más de trabajo, pero está garantizado que terminamos con la respuesta correcta.

Ejercicios

2.7.1.- Construye el tableau correspondiente a cada una de las fórmulas, sin cerrar ramas. Para poder hacerlo, primero transforma a la fórmula para que tenga únicamente conjunciones y disyunciones de literales.

- $((p \vee q) \wedge (r \rightarrow \neg p)) \rightarrow (r \rightarrow q)$
- $((p \rightarrow q) \rightarrow p) \rightarrow (q \rightarrow p)$
- $((p \vee q) \wedge (p \vee r)) \rightarrow p$
- $((p \rightarrow q) \wedge (q \rightarrow r)) \rightarrow (\neg r \rightarrow \neg p)$
- $((r \vee \neg s) \vee t) \wedge \neg ((p \vee \neg q) \wedge (\neg q \vee \neg p))$
- $\neg ((p \rightarrow q) \wedge (q \rightarrow p))$
- $(p \rightarrow q) \vee r$

2.7.2.- Usando tableaux, determina cuál de las siguientes fórmulas es tautología, contradicción o contingente.

- $((p \vee q) \vee r) \wedge (p \vee (q \vee r)) \rightarrow p \vee q$
- $(p \wedge (q \wedge r)) \rightarrow (p \rightarrow (q \rightarrow r))$

$$(c) \quad p \vee q \rightarrow p \vee r$$

$$(d) \quad (p \rightarrow (p \rightarrow q)) \rightarrow p$$

2.7.3.- Demuestra que las siguientes fórmulas son tautologías usando tableaux:

$$(a) \quad (p \rightarrow q) \wedge (r \rightarrow s) \wedge (\neg q \vee \neg s) \rightarrow \neg p \vee \neg r$$

$$(b) \quad p \vee (p \wedge \neg q \rightarrow r)$$

$$(c) \quad p \vee (\neg p \wedge q) \rightarrow p \vee q$$

$$(d) \quad (p \rightarrow q) \rightarrow (p \vee q \rightarrow q)$$

$$(e) \quad (\neg p \wedge (\neg p \wedge q)) \vee (p \wedge (p \wedge \neg q)) \leftrightarrow (\neg p \wedge q) \vee (p \wedge \neg q)$$

2.7.4.- Para los ejercicios 2.7.2 y 2.7.3, usando los tableaux construidos, da un modelo para las fórmulas y para sus negaciones en el caso de que sean contingentes, así como un modelo contraejemplo para los argumentos incorrectos.

2.7.5.- Determinar si los siguientes conjuntos de fórmulas Γ son satisfacibles en cuyo caso dar un modelo.

$$a) \quad \Gamma = \{\neg p \wedge q, (r \rightarrow p \leftrightarrow \neg q) \vee \neg r, \neg(r \vee \neg p)\}$$

$$b) \quad \Gamma = \{r \rightarrow \neg(p \wedge \neg q), ((p \rightarrow r) \rightarrow (\neg q \leftrightarrow r)) \wedge \neg r, \neg(q \wedge q)\}$$

$$c) \quad \Gamma = \{(p \wedge r) \vee (\neg r \rightarrow q), (q \leftrightarrow r) \rightarrow (\neg q \rightarrow r), \neg p \wedge q \wedge \neg r\}$$

$$d) \quad \Gamma = \{p \wedge \neg q \rightarrow \neg r, (\neg p \rightarrow \neg q) \wedge \neg r, r \wedge q \wedge \neg r\}$$

$$e) \quad \Gamma = \{\neg q \rightarrow \neg r, p, (\neg q \rightarrow p) \rightarrow q, \neg r \rightarrow p, s \rightarrow q, r \vee p\}$$

2.7.6.- Usando tableaux, determina la correctud de los siguientes argumentos.

$$(a) \quad (p \rightarrow q) \wedge (p \rightarrow r) / \therefore q \rightarrow r$$

$$(b) \quad p \vee q \rightarrow r, s \rightarrow p, s / \therefore r$$

$$(c) \quad p \vee q, \neg(p \wedge r), \neg q / \therefore r \rightarrow s$$

$$(d) \quad p \rightarrow q, p \vee r, \neg(r \wedge s) / \therefore (p \rightarrow q) \rightarrow (q \vee \neg s)$$

2.7.7.- ¿Por qué es que se pueden cerrar ramas si es que aparece una literal y la literal complementaria en un camino dentro del tableau?

2.7.8.- Explica en tus propias palabras por qué los tableaux no se pueden construir para fórmulas que tienen otros operadores que no sean la conjunción y la disyunción.

Lógica de predicados | 3

3.1. Introducción

Hemos utilizado fórmulas de la lógica proposicional siempre que se trata de representar proposiciones en español, esto es, enunciados que son falsos o verdaderos. Analicemos los siguientes enunciados:

- Todo plátano es amarillo.
- Algunas especies de pájaros migran.
- Todos los vaqueros usan sombrero.
- Ningún perro maúlla.
- Baja California Sur es el único estado de la República Mexicana con mar en tres de sus cuatro bordes.

Se puede observar que todos y cada uno de los enunciados anteriores es una proposición pues tiene un valor de falso o verdadero. Sin embargo difieren de las estudiadas anteriormente pues no reconocemos en los enunciados palabras correspondientes a conectivos lógicos, por lo que la única manera que tenemos, por el momento, de formalizarlos es simplemente con una sola variable proposicional asignada a todo el enunciado.

El lenguaje de la lógica proposicional estudiado en el capítulo anterior no tiene suficiente poder expresivo para analizar proposiciones y argumentos que requieren de una clase de

enunciados como los anteriores, que contienen referencias a colectividades de objetos.

Considérese por ejemplo el siguiente razonamiento:

Algunas personas van al teatro. Todos los que van al teatro se divierten. De manera que algunas personas se divierten.

La intuición dice que el argumento es correcto. Sin embargo la representación correspondiente en lógica proposicional es:

$p, q / \therefore r$ ¡Incorrecto!

Esta situación nos permite concluir únicamente que el argumento en lógica proposicional es incorrecto en el sentido de que la conclusión no es consecuencia lógica de las premisas. Sin embargo, a partir de este hecho no es posible concluir que el argumento en lenguaje natural sea incorrecto, pues podría ser que lo sea en base a ciertos principios lógicos más fuertes. Tal posibilidad requiere el desarrollo de un lenguaje de especificación formal más poderoso, así como una lógica adecuada al mismo, llamada lógica de predicados.

3.1.1. Predicados

Consideremos los siguientes enunciados:

- Cualquier empleado tiene un jefe.
- Algunos programas usan ciclos.
- Hay una lista que está ordenada.

Como acabamos de argumentar, para representar a cada uno de estos enunciados la única forma de hacerlo, con las herramientas que tenemos hasta el momento, es mediante fórmulas proposicionales atómicas, es decir, mediante una simple variable proposicional para cada una de ellos. De los dos ejemplos anteriores vemos que esta representación no es adecuada, ya que no es capaz de reflejar la estructura interna del enunciado, algo de lo que no debemos sustraernos. Buscamos una herramienta lógica que tome en cuenta, de alguna manera, a esa estructura interna. Por ejemplo, el enunciado *algunos programas usan ciclos* trata acerca de programas, ciclos y la acción de usar. Estas componentes de la estructura interna de un enunciado se clasifican como *individuos* (u objetos) y propiedades (o relaciones) atribuibles a los individuos; a estas últimas es a las que llamamos *predicados*. Tanto los individuos como los predicados se definen en un contexto particular dependiendo del problema que queramos representar. Este contexto se conoce como *universo de discurso*, el cual es una colección de todas las personas, ideas, cosas, estructuras de datos, etcétera, necesarios para analizar una fórmula o argumento lógico.

Veamos algunos ejemplos para hacer la distinción entre predicados e individuos en universos de discurso. En cada caso los individuos se encuentran encerrados en una caja y los predicados son las partes del enunciado que describen las relaciones entre ellos, así como las acciones que los individuos llevan a cabo; por ejemplo, *ser colegas*; *ser padre de*; *ser canario*; *ser la suma de*; *usar*; *visitar*; *ir*; *jugar*; etcétera.

- El universo de discurso son personas:

- `Isabel` y `María` son colegas.
- `Pedro` es el padre de `Juan`.

- El universo de discurso son los animales:

- `Piolín` es un canario.
- `Claudio` es un gallo.

- El universo son números:

La suma de `2` y `3` es `5`.

El producto de `10` y `-2` es negativo.

- El universo consta de lenguajes de programación, algoritmos y programas:

- `Haskell` es un lenguaje funcional con el que se puede escribir el algoritmo `quicksort` en una línea.
- Este `programa` en Java usa `clases`.

- El universo puede constar de diversas clases de individuos, como en el caso de que los siguientes enunciados se usen en un mismo contexto:

- La infanta `Christina` visita `museos`.
- `El teatro` al que la condesa `Karla Michelle` fue ayer tiene asientos cómodos.
- Su majestad `Martha Elena III` y el perro imperial `Bu` juegan en el `jardín` de palacio.

En el caso de estos enunciados, el universo tiene al menos personas, animales y lugares.

Aunque parezca que podemos utilizar lógica proposicional para representar a los individuos y relaciones, esto no es así. Por ejemplo, no tiene sentido decir que el primer enunciado se formaliza como $p \wedge q$ donde p significa *Isabel es colega* y q significa *María es colega*, ya que la conjunción de estos dos enunciados no consigue explicar la relación de colegas *entre* Isabel y María.

En *lógica de predicados* utilizamos la notación

$$P(t_1, t_2, \dots, t_n)$$

para describir que la propiedad o relación P se da entre los individuos t_1, t_2, \dots, t_n . Expresemos algunos de los ejemplos que dimos arriba con esta nueva notación:

- $Colegas(Isabel, María)$, con $P = \text{ser colegas}$, $t_1 = \text{Isabel}$ y $t_2 = \text{María}$.
- $Padre(Pedro, Juan)$, con $P = \text{padre de}$, $t_1 = \text{Pedro}$ y $t_2 = \text{Juan}$.
- $Canario(Piolín)$, con $P = \text{ser canario}$ y $t_1 = \text{Piolín}$.
- $Suma(2, 3, 5)$, con $P = \text{suma}$, $t_1 = 2$ y $t_2 = 3$ son los sumandos y $t_3 = 5$ es el resultado.

Podemos ver de estos ejemplos que cada predicado P recibe un número distinto de argumentos de entrada t_1, \dots, t_n – escribimos el nombre de los predicados con mayúsculas para distinguirlos de las funciones –. Al número de argumentos de un predicado le llamamos el *índice* o *aridad* del predicado. También vemos que el orden, en muchos de ellos, es importante. Por ejemplo, el predicado $Padre$ tiene un significado muy distinto si cambiamos el orden de los argumentos, lo mismo que el predicado $Suma$. Una vez que se ha definido un predicado con un determinado índice, queda prohibido cambiarle el índice posteriormente. Por ejemplo, en el primer predicado, $Colegas$, el índice es 2, lo cual impide formar expresiones como $Colegas(\text{Juan}, \text{Lupe}, \text{Rosa})$, aun cuando esto tenga sentido desde nuestra intuición. Si se desea utilizar un número de argumentos distinto al definido inicialmente por el índice, es necesario definir otro predicado, por ejemplo $Colegas'(\text{Juan}, \text{Lupe}, \text{Rosa})$. Los predicados de índice uno, es decir de un solo argumento, reciben el nombre específico de *propiedades*.

3.1.2. Variables y cuantificadores

Hasta ahora el uso de predicados en lugar de variables proposicionales podría parecer simplemente otra manera de escribir enunciados. Por ejemplo, la proposición

Anastasia recita poesía nórdica,

se representa con predicados como $Recita(\text{Anastasia}, \text{poesía nórdica})$, lo cual parece ser simplemente una manera distinta de escribir el mismo enunciado en español. La principal diferencia es que el predicado puede cambiar de argumentos, como en

$Recita(\text{Licantro}, \text{odas en sánscrito})$.

Más aún, podemos sustituir individuos por variables, como en $Recita(x, y)$. De esta manera podemos definir predicados de manera más formal, como los que siguen:

- $F(x, y)$ significa que x es padre de y .
- $E(x)$ significa que x es un estudiante
- $J(x, y)$ significa que x es más joven que y .

Es de importancia remarcar que los nombres de las variables no importan siempre y cuando se usen de forma consistente. Sin embargo, obsérvese que las expresiones anteriores no corresponden a proposiciones, puesto que los valores de x e y están indeterminados, por lo que resulta imposible verificar si el predicado *Recita* se cumple. Las variables juegan el papel de representantes de valores concretos, como un estudiante, un número o un programa particular. Obsérvese entonces que un mismo predicado puede representar un número potencialmente infinito de proposiciones, una por cada individuo que pueda tomar el lugar de cada una de las variables del predicado.

Consideremos ahora los siguientes enunciados:

- *Hay un gato rayado.*
- *Algunas personas van al teatro.*
- *Todos los programas en Java usan clases.*
- *Todos los estudiantes trabajan duro.*
- *Algunos estudiantes se duermen en clase.*
- *Ocho de cada diez gatos lo prefieren.*
- *Nadie es más tonto que yo.*
- *Al menos seis estudiantes están despiertos.*
- *Hay una infinidad de números primos.*
- *Hay más computadoras PC que Mac.*

Todos estos enunciados tienen en común el hecho de que no involucran a ningún individuo en particular. Aun cuando tenemos predicados a nuestra disposición, necesitamos un mecanismo para formalizar las partes de los enunciados que se refieren a una cantidad, como *todos*, *algunos*, *hay*, *nadie*, *cualquiera*, A estas cantidades es a lo que llamamos *cuantificadores*.

Por ejemplo, para el enunciado *Todos los estudiantes son más jóvenes que algún profesor*, entendiendo que el universo de discurso son las personas de la Facultad de Ciencias, sería inoperante escribir todos los posibles predicados para estudiante, profesor y ser más joven, $E(\text{Karla})$, $E(\text{Hugo})$, . . . , $P(\text{Elisa})$, $P(\text{Favio})$, $J(\text{Karla}, \text{Favio})$, Más aún, en algunos casos esto resulta imposible, como con la frase *hay una infinidad de números primos*.

Este problema se soluciona al emplear operadores de cuantificación sobre individuos indeterminados, \forall (se lee *para todo*) y \exists (se lee *existe*), los cuales siempre van seguidos de una variable que representa a los individuos de la colectividad que se está especificando. Por ejemplo, para decir *todos hablan español* escribimos $\forall x E(x)$ donde $E(x)$ significa que x habla español. Similarmente, si $C(x)$ significa que x es cuervo, entonces para especificar que *hay un cuervo* escribimos $\exists x C(x)$. Más aún, usando cuantificadores en combinación con la lógica proposicional, podemos representar enunciados más complejos, como por ejemplo *todos los estudiantes son más jóvenes que algún profesor* cuya especificación es

como sigue:

$$\forall x(E(x) \rightarrow \exists y(P(y) \wedge J(x, y))),$$

donde queda claro que $P(x)$ significa *x es profesor*; $E(x)$ significa que *x es estudiante* y $J(x, y)$ significa que *x es más joven que y*.

A continuación vamos a definir el lenguaje formal de la lógica de predicados que incluye todos los elementos discutidos hasta ahora, para después volver al tema de especificación formal.

3.2. Sintaxis de la lógica de predicados

En esta sección definimos formalmente lo que se conoce como un lenguaje de la lógica de predicados de primer orden, el cual, a diferencia del caso proposicional, varía dependiendo del universo de discurso particular y de las relaciones y propiedades de los individuos que se deseen especificar.

3.2.1. Términos

Los *términos* son la categoría sintáctica que representa individuos del universo de discurso.

Definición 3.1 Un *término* es una constante, una variable o bien un símbolo de función aplicado a otros términos.

Los términos se generan mediante la siguiente gramática. En los casos en que aparezca una coma (“,”), ésta es parte de la sintaxis. El metasímbolo “...” significa “más de los anteriores” y no forma parte de los símbolos terminales de la gramática.

term ::= var	(3.1)	const ::= b	(3.9)
term ::= const	(3.2)	const ::= c	(3.10)
term ::= func(lista-de-term)	(3.3)	const ::= ...	(3.11)
var ::= x	(3.4)	func ::= f	(3.12)
var ::= y	(3.5)	func ::= g	(3.13)
var ::= z	(3.6)	func ::= h	(3.14)
var ::= ...	(3.7)	func ::= ...	(3.15)
const ::= a	(3.8)	lista-de-term ::= term	(3.16)
		lista-de-term ::= term, lista-de-term	(3.17)

Cada símbolo de la categoría func tiene asociado un número fijo de argumentos (el índice o aridad del símbolo). A veces escribimos $f^{(n)}$ para indicar que el símbolo f tiene índice n .

Veamos a continuación algunos ejemplos.

Ejemplo 3.1. Supongamos que el universo consta de los países del mundo.

- Las variables x e y denotan a países cualesquiera.
 - La constante a denota a Alemania y la constante b a Brasil.
 - El símbolo funcional f de índice 1 denota a la operación que recibe un país y devuelve su ciudad capital. Es decir, $f(x)$ es la capital de x . Esto es posible dado que cada país tiene una única capital de manera que dicha asociación es funcional. En particular $f(a)$ denota a Berlín y $f(b)$ a Brasilia.
-

Ejemplo 3.2. Si el universo consta de números naturales, entonces:

- La constante a denota al individuo 0 y la constante b al individuo 1.
 - Los términos funcionales $f^{(2)}(x, y)$ y $g^{(2)}(x, y)$ denotan a los individuos $x + y$ y $x * y$ respectivamente.
 - En tal caso, los individuos 2 y 4 se representan mediante $f(b, b)$ y $g(f(b, b), f(b, b))$ respectivamente.
-

3.2.2. Fórmulas

Una vez definidos los términos podemos construir las fórmulas del lenguaje, las cuales representan a las relaciones entre individuos, así como a los enunciados generales del lenguaje. Empecemos con las fórmulas más simples, las atómicas.

Definición 3.2 Una *fórmula atómica* es una expresión de la forma $P(t_1, \dots, t_n)$, donde P es un símbolo de predicado de índice n y t_1, \dots, t_n son términos.

Ejemplo 3.3. Definimos los símbolos de predicado $P^{(2)}$, $R^{(3)}$, $Q^{(1)}$, los símbolos de función $f^{(1)}$ y $g^{(2)}$ y las constantes a, b y c . Las siguientes son fórmulas atómicas:

- $P(b, f(y))$
 - $Q(g(f(a), c))$
 - $R(z, f(g(a, c)), b)$
-

Ahora que tenemos fórmulas atómicas, podemos combinarlas con los conectivos proposicionales para obtener fórmulas más complejas.

Ejemplo 3.4.

En el universo de discurso de los números naturales, si $a + b = c + b$ entonces $a = c$. Definimos las constantes a, b, c y los siguientes símbolos de función:

$$\begin{array}{lll} f(x, y) & \text{para representar} & x + y \\ \text{igual}(x, y) & \text{para representar} & x = y \end{array}$$

Y la especificación queda como sigue:

$$\text{igual}(f(a, b), f(c, b)) \rightarrow \text{igual}(a, c)$$

Ejemplo 3.5.

En la expresión *Bombón es un gato que araña* tenemos lo siguiente:

- a) El universo de discurso son los animales (los mamíferos, los felinos, cualquier conjunto, raza o familia que incluya a los gatos).
- b) Los predicados que definimos son:

$$\begin{array}{ll} G(x) & x \text{ es un gato} \\ A(x) & x \text{ araña} \end{array}$$

Siendo *Bombón* uno de los individuos concretos del universo de discurso, estará representado por una constante, su propio nombre. La expresión lógica queda como sigue:

$$G(\text{Bombón}) \wedge A(\text{Bombón})$$

Otros ejemplos son:

$$\begin{array}{l} \text{Perro}(x) \rightarrow \text{Tienecola}(x) \\ \text{Madre}(x, y) \wedge \text{Madre}(x, z) \rightarrow \text{Hermanos}(y, z) \\ \text{Calif}(x) \rightarrow x \geq 0 \wedge x \leq 10 \end{array}$$

Obsérvese que en el último ejemplo los predicados \geq, \leq se usan de manera infija como es usual en matemáticas. ¿Cual es el universo de discurso en cada caso?

De los ejemplos anteriores se observa que las fórmulas con predicados se generan de la misma manera que las fórmulas de la lógica proposicional, sólo que las fórmulas atómicas

han cambiado de simples variables proposicionales a predicados que involucran términos. Veamos la gramática formal, en la que usamos el metasímbolo “|” para separar alternativas de sustitución para un mismo símbolo no terminal de manera abreviada, y el metasímbolo “...” para denotar “más como los anteriores”.

$$E ::= \text{pred}(\text{lista-de-term}) \quad (3.18)$$

$$E ::= \neg E \quad (3.19)$$

$$E ::= E \rightarrow E \quad (3.20)$$

$$E ::= E \vee E \quad (3.21)$$

$$E ::= E \wedge E \quad (3.22)$$

$$E ::= E \leftrightarrow E \quad (3.23)$$

$$E ::= (E) \quad (3.24)$$

$$\text{pred} ::= P \mid Q \mid R \mid \dots \quad (3.25)$$

$$\text{lista-de-term} ::= \text{term} \quad (3.26)$$

$$\text{lista-de-term} ::= \text{term}, \text{lista-de-term} \quad (3.27)$$

Nuevamente cada símbolo de la categoría *pred* tiene asociado un número fijo de argumentos.

3.2.3. Fórmulas cuantificadas

Finalmente definimos las fórmulas que involucran cuantificadores, las cuales proporcionan una gran expresividad al lenguaje.

Definición 3.3 Sea E una fórmula. La expresión $\forall x E$ es la *cuantificación universal* de E con respecto a x y representa al enunciado *para todo x se cumple E* . Análogamente, la expresión $\exists x E$ es la *cuantificación existencial* de E con respecto a x y representa al enunciado *existe un x que cumple E* . En ambos casos la fórmula E se conoce como el *alcance de la cuantificación* y la variable que figura inmediatamente después del cuantificador se conoce como *variable del cuantificador*.

Una pregunta común es el porqué en la cuantificación universal se prefiere el formato $P(x, \dots) \rightarrow Q(x, \dots)$, mientras que en la cuantificación existencial se prefiere la forma $P(x) \wedge Q(x)$. Cuando tenemos una cuantificación universal, examinaremos a todo el universo de discurso para comprobar que todo aquel que cumple $P(x, \dots)$, también cumple $Q(x, \dots)$. Si el individuo que estamos examinando no cumple $P(x, \dots)$ no nos interesa qué pasa con $Q(x, \dots)$, por lo que queremos que la cuantificación sea verdadera fijándonos únicamente en aquellos individuos que cumplen $P(x, \dots)$. Si usáramos el esquema

$P(x, \dots) \wedge Q(x, \dots)$, la cuantificación sería falsa si en el universo de discurso hubiese individuos que no cumplen con $P(x, \dots)$.

En el caso de la cuantificación existencial deseamos encontrar en el universo de discurso al menos a un individuo que cumpla con $P(x, \dots)$ y $Q(x, \dots)$. Si usamos la fórmula $P(x, \dots) \rightarrow Q(x, \dots)$, al examinar al primer individuo que no cumpla con $P(x, \dots)$ daríamos por buena la cuantificación, pues en el caso de que el antecedente sea falso, la condicional es verdadera; no seguiríamos revisando el universo de discurso para ver si encontramos a un individuo que cumpla con $P(x, \dots)$; la cuantificación se evaluaría a verdadero aun cuando no existiese ningún individuo como el que queremos. Con la conjunción garantizamos que tiene que existir al menos un individuo que cumpla con ambos predicados.

Pasemos a ver algunos ejemplos.

Ejemplo 3.6. Supongamos que el universo de discurso es el universo astronómico. Sea $S(x)$ el predicado *ser sol* y $P(x)$ el predicado *ser planeta*. Vamos a traducir algunos enunciados sencillos que involucran cuantificadores.

- Todo es un sol: $\forall x S(x)$
- Todo es un planeta: $\forall y P(y)$
- Existe un planeta y un sol: $\exists x \exists y (P(x) \wedge S(y))$
- Cualquiera es sol o planeta: $\forall z (P(z) \vee S(z))$

En la sección 3.3 discutiremos el proceso de especificación más ampliamente. A continuación ejemplificamos el concepto de *alcance*.

Ejemplo 3.7. Veamos un ejemplo de los alcances de cuantificaciones. Encerraremos en un cuadro los alcances, marcando a la variable de las cuantificaciones correspondientes.

$$\forall x \left(\left(x > i \wedge i > j \right) \rightarrow \exists i \exists j \left(\boxed{x > i \wedge x > j} \right) \right)$$

El recuadro de guiones marca el alcance de la cuantificación $\forall x$ mientras que el recuadro de línea sólida marca el alcance de las cuantificaciones $\exists i$ y $\exists j$.

Agregamos a la gramática que acabamos de dar para la lógica de predicados las reglas que describen a la cuantificación universal y existencial como fórmulas lógicas:

$$E ::= \forall \text{ var } E \quad (3.28)$$

$$E ::= \exists \text{ var } E \quad (3.29)$$

con lo cual nuestra gramática para fórmulas de la lógica de predicados queda completa.

3.2.4. Variables libres y ligadas

Consideremos el enunciado *todos son blancos*; si deseamos especificarlo formalmente, primero debemos definir un predicado $B(x)$ cuyo significado es *x es blanco*, para después cuantificar universalmente, obteniendo la fórmula $\forall x B(x)$. Ahora bien, consideremos la fórmula $\forall y B(y)$, ¿qué enunciado en español se especifica ahora? Fácilmente nos damos cuenta de que su significado es nuevamente *todos son blancos*, es decir, el nombre particular de la variable utilizada, en este caso x o y , es irrelevante para entender el significado de la fórmula. Por esta razón a la variable de un cuantificador se le conoce también como variable artificial o monigote¹, porque únicamente marca el lugar o la posición. En contraste, consideremos las fórmulas $B(x)$ y $B(y)$, y supongamos que el universo son los números naturales, siendo B el predicado *ser par*. Con esta información no es posible entender el significado² de estas fórmulas, pues hace falta saber el valor de sus variables. Por ejemplo, si x es 3 entonces $B(x)$ significa *3 es par*, mientras que si y vale 8 entonces $B(y)$ significa *8 es par*, de donde claramente $B(x)$ y $B(y)$ no significan lo mismo, pues su significado depende del valor particular de sus variables. La pregunta inmediata es: ¿cuándo es relevante el valor particular de una variable para conocer el significado y valor de verdad de una fórmula? Para responderla introducimos los conceptos de *variable libre* y *ligada*.

Definición 3.4 Se dice que una presencia **específica** de una variable x en una fórmula A está *libre* si no es la variable artificial de un cuantificador ni figura dentro del alcance de una cuantificación cuya variable artificial también es x .

En la siguiente tabla presentamos algunas fórmulas y la lista de variables libres de cada una de ellas.

Cuantificación	Variables libres
$\forall x((x > i \wedge i > j) \rightarrow (x > j))$	i, j
$\exists x(x > i \wedge i > j)$	i, j
$\forall i \forall j((x > i \wedge i > j) \rightarrow (x > j))$	x
$\forall i((x > i \wedge i > j) \rightarrow (x > j))$	x, j
$\exists i \exists j(x > i \wedge i > j)$	x
$\exists j(x > i \wedge i > j)$	i, x

Las variables que no figuran libres en una fórmula se denominan *ligadas* o *acotadas*. Veamos una definición más detallada.

¹En inglés *dummy*

²El valor de verdadero o falso

Definición 3.5 Decimos que una presencia determinada de una variable x en una fórmula A es *ligada* o *acotada* si x es una variable artificial de A o cae dentro del alcance de un cuantificador con la misma variable artificial x .

Los enunciados en español se formalizan mediante fórmulas que no tienen variables libres, a las cuales llamamos también enunciados, sentencias o fórmulas cerradas.

Definición 3.6 Un enunciado es una fórmula A que no contiene presencias libres de variables.

Ejemplo 3.8. En la expresión

$$\overset{(1)}{i} > 0 \vee \forall \overset{(2)}{i} (0 \leq \overset{(3)}{i} \rightarrow x \cdot \overset{(4)}{i} = 0)$$

tenemos cuatro presencias de i . La primera presencia de i , anotada con (1), es una presencia libre, pues no se encuentra dentro de ninguna cuantificación. El valor que contribuya a la expresión dependerá del estado en el que se la evalúe. La segunda presencia es la variable artificial de un cuantificador por lo que es ligada. Las otras dos presencias de i también son ligadas, el valor de la cuantificación no depende del valor particular que tenga i . Finalmente, la presencia de x es una presencia *libre* y su contribución a la expresión dependerá también del estado en el que se evalúe la expresión.

Ejemplo 3.9. En la expresión

$$\overset{(1)}{k} + \overset{(2)}{j} > 0 \wedge \exists \overset{(3)}{j} (0 \leq \overset{(4)}{j} \leq 5 \wedge \overset{(5)}{k} < \overset{(6)}{j})$$

las presencias (1) y (5) de k son presencias libres, pues en el primer caso la k se encuentra fuera de la cuantificación y aunque en el segundo caso se encuentra dentro de una cuantificación, la variable artificial es j , no k .

La presencia (2) de j es distinta que las presencias (3), (4) y (6), pues mientras la primera se encuentra fuera de una cuantificación y es, por lo tanto, presencia libre, las otras tres se encuentran dentro de una cuantificación donde la variable artificial es ella misma, por lo que son presencias acotadas.

El valor de esta fórmula dependerá del estado en el que se evalúen los valores libres de j y k .

Puede suceder que el usar una misma variable (j en el ejemplo anterior) para dos papeles distintos – el papel de presencia libre en (2) y de presencia acotada en (4) y (6) – lleve al lector a confusión. En estos casos es recomendable cambiar el nombre a todas las presencias *ligadas* de la variable en cuestión que participan directamente en la cuantificación para eliminar confusiones. De hacer esto, la expresión que dimos arriba quedaría como sigue:

$$\overset{(1)}{k} + \overset{(2)}{j} > 0 \wedge \exists \overset{(3)}{i} (0 \leq \overset{(4)}{i} \leq 5 \wedge \overset{(5)}{k} < \overset{(6)}{i})$$

El concepto de variables libres o acotadas es similar al que presentan los lenguajes de programación con estructura de bloques. En ellos tenemos la posibilidad de *anidar* pedazos de código de la siguiente manera:

```
1  var i : integer ;
2  procedure p(var x: integer) ;
3      var i : integer ;
4  begin
5      i := x * x ;
6      x := 2 * i ;
7  end
```

La presencia de i en la línea 1 es libre, ya que no se encuentra dentro de un bloque. La presencia de x en 2 es ligada y hace el papel de una declaración, pues le da nombre a una variable artificial. Las presencias de x en 5 y 6, así como las presencias de i dentro del procedimiento son acotadas (líneas 5 y 6), ya que estos identificadores sólo tienen significado dentro del procedimiento. Si cambiáramos los identificadores de x a y en todas las presencias acotadas, y de i a k también en las presencias acotadas, el procedimiento obtenido es

```
1  var i : integer ;
2  procedure p(var y: integer) ;
3      var i : integer ;
4  begin
5      k := y * y ;
6      y := 2 * k ;
7  end
```

y hace exactamente lo mismo que el original.

Para terminar esta sección deseamos hacer hincapié en lo siguiente:

- Al trabajar con predicados es muy importante que el universo de discurso esté bien definido y sea claro.
- Los términos y las fórmulas son ajenos, es decir ningún término es fórmula y ninguna fórmula es término.
- Los términos denotan exclusivamente a individuos u objetos.
- Las fórmulas atómicas (predicados) denotan únicamente proposiciones o propiedades acerca de los términos.
- Únicamente los individuos u objetos son cuantificables. Esta característica justifica la denominación *primer orden* que se le da a la lógica de predicados que estamos estudiando.

Ejercicios

3.2.1.- Sean $f^{(2)}$ y $g^{(3)}$ símbolos de función y d una constante. ¿Cuáles de las siguientes expresiones son términos? Justifica tu respuesta.

- a) $g(d, d)$
- b) $f(x, g(y, z), d)$
- c) $g(x, f(y, z), d)$
- d) $f(d, x)$
- e) $g(d, g(x, y, f(z, d)), f(f(d, x), w))$
- f) $g(g(y, y, f(d, d)), f(w, g(d, x, y)))$

3.2.2.- Sean a una constante, $f^{(1)}$ un símbolo de función y $P^{(2)}$, $Q^{(2)}$ y $R^{(1)}$ símbolos de predicado. ¿Cuáles de las siguientes expresiones son fórmulas? Justifica tu respuesta.

- a) $P(a, x)$
- b) $Q(a, f(a))$
- c) $f(a)$
- d) $Q(Q(a, x), x)$
- e) $R(Q(x, x))$
- f) $P(f(y), a)$
- g) $R(f(w))$
- h) $Q(\neg R(x), w)$
- i) $P(x, y) \rightarrow \exists z Q(z, y)$
- j) $\forall f Q(f(y), y)$
- k) $\neg R(f(z)) \vee \neg Q(a, f(w))$
- l) $\forall x \exists y Q(x, y) \wedge R(w)$

3.2.3.- Sean c y d constantes, $f^{(1)}$, $g^{(2)}$ y $h^{(3)}$ símbolos de función y $P^{(3)}$ y $R^{(2)}$ símbolos de predicado. ¿Cuáles de las siguientes expresiones son fórmulas? Justifica tu respuesta.

- a) $Q(c, d, c)$
- b) $\forall x P(f(d), h(g(c, x), d, y))$
- c) $\forall x P(f(d), h(R(y, y), d))$
- d) $\exists w P(g(h(x, f(d), x), g(w, w)), h(x, x, x), c)$

- e) $\exists u(Q(z, z, z) \rightarrow R(y, y))$
 f) $\forall x \forall y(g(x, y) \rightarrow P(x, y, x))$

3.2.4.- Para cada una de las siguientes fórmulas, clasifica todas las presencias de variables en libres o ligadas. Además da el alcance de cada cuantificador.

- a) $R(x, y) \wedge L(y)$
 b) $\forall x R(x, f(y, z)) \wedge L(y)$
 c) $R(f(x, y), z) \wedge \exists y R(f(y, x), z) \rightarrow \forall w I(x, y)$
 d) $\forall x(C(x, z) \wedge R(f(y, x), f(x, y))) \rightarrow C(y, z)$
 e) $\exists x \exists y R(x, y) \wedge C(x, y)$
 f) $C(f(x, y), z) \wedge \exists y C(f(y, x), z) \rightarrow \forall x(L(x) \wedge L(y) \wedge I(x, y))$
 g) $\exists y C(x, y) \vee \exists z R(x, z)$
 h) $\forall x(L(x) \rightarrow R(f(x, a), x) \wedge C(f(x, a), a))$
 i) $\exists y C(a, y) \vee L(a)$
 j) $\exists y(I(f(x, y), f(y, x)) \wedge D(r(x)))$.
 k) $D(f(x, y)) \vee \forall z C(z, r(y))$.
 l) $\exists y(C(x, f(y, z)) \wedge D(y) \wedge \forall x I(z, r(y)))$.
 m) $\forall x \exists z I(z, r(x)) \rightarrow C(z, y) \wedge D(y)$.

3.2.5.- Para cada una de las siguientes fórmulas, clasifica todas las presencias de variables en libres o ligadas. Además da el alcance de cada cuantificador.

- a) $\forall x \exists z(Q(z, y) \wedge \exists y R(x, f(x)))$
 b) $P(x, a, y) \vee \exists y(P(x, y, a) \wedge R(a, z))$
 c) $W(f(x, a), g(y)) \wedge \forall x \exists y S(f(x, a), g(z))$
 d) $\neg R(f(x, x), w, g(x)) \wedge \forall x \exists y T(x, y, g(z))$
 e) $\forall w T(w, x, g(y)) \rightarrow \neg \exists z R(x, f(w, y))$

3.2.6.- Construye para cada inciso una fórmula A que cumpla las condiciones dadas.

- a) A es un enunciado que es una cuantificación de una implicación donde el consecuente es una fórmula existencial.
 b) A es un enunciado y es una cuantificación existencial de una conjunción.
 c) A es una fórmula que incluye cuantificadores pero tiene al menos tres presencias libres de exactamente dos variables.

- d) A es un enunciado y una disyunción de un enunciado atómico con una fórmula existencial cuyo alcance es un predicado ternario.
- e) A es una fórmula que no es un enunciado y tiene dos cuantificadores universales con variables distintas, un cuantificador existencial y además se convierte en un enunciado al cuantificarla universalmente.

3.3. Especificación formal

El proceso de especificación o traducción del español a la lógica formal no es siempre sencillo. Algunas frases del español no se pueden traducir de una manera completamente fiel a la lógica de predicados, como veremos en algunos ejemplos. Sin embargo, el proceso es de gran importancia pues es la base de muchos métodos de especificación formal utilizados en inteligencia artificial o ingeniería de software (como ejemplo tenemos el lenguaje de especificación Z). A continuación presentamos algunos consejos, observaciones y ejemplos que pretenden facilitar la especificación del español en términos de la lógica.

- Únicamente podemos especificar afirmaciones o proposiciones; no es posible traducir preguntas, exclamaciones, órdenes, invitaciones, etcétera.
- La idea básica es extraer predicados a partir de los enunciados dados en español, de manera que el enunciado completo se construya al combinar fórmulas atómicas mediante conectivos y cuantificadores. Por ejemplo, la frase *me gustan los tacos y las pizzas* debe traducirse como *me gustan los tacos y me gustan las pizzas*. Análogamente *iré de vacaciones a la playa o a la montaña* significa *iré de vacaciones a la playa o iré de vacaciones a la montaña*.
- La conjunción “y” se traduce como \wedge . La palabra “pero” también, aunque el sentido original del español se pierde. Por ejemplo *te doy dulces pero haces la tarea* sólo puede traducirse en *Te doy dulces y haces la tarea*, lo cual es diferente en el lenguaje español.
- La disyunción es incluyente: *comeremos pollo o vegetales* incluye el caso en que se coman ambos.
- Con la implicación hay que ser cautelosos, sobre todo en el caso de frases de la forma *A sólo si B* lo cual es equivalente con *Si no B entonces no A*, que a su vez es equivalente con *Si A entonces B*. Es un error común intentar traducir dicha frase inicial mediante $B \rightarrow A$.
- Si en el español aparecen frases como *para todos*, *para cualquier*, *todos*, *cualquiera*, *los*, *las*, debe usarse el cuantificador universal \forall .

- Si en el español hay frases como *para algún, existe un, alguno, alguna, uno, una, alguien*, generalmente se usa el cuantificador existencial \exists .
Importante: En ciertas ocasiones, frases en español que involucran *alguien, algo* deben especificarse con un cuantificador universal y no un existencial. Por ejemplo, el enunciado *si hay alguien demasiado alto entonces se pegará con el marco de la puerta* se puede reescribir en español como *cualquiera demasiado alto chocará con el marco de la puerta*, lo cual nos lleva a $\forall x(A(x) \rightarrow C(x))$. El lector debe convenirse de que no es posible traducir esta oración usando un cuantificador existencial.
- Pronombres como *él, ella, eso* no se refieren a un individuo particular sino que se usan como referencia a algo o alguien mencionado previamente, por lo que obtienen significado del contexto particular. Cuando un pronombre aparezca en un enunciado debe uno averiguar primero a quién o qué se refiere. Por ejemplo, en el enunciado *Martha es amiga de Lupita pero ella no es amiga de Karla* debe traducirse como *Martha es amiga de Lupita y Lupita no es amiga de Karla*. Similarmente, cuando necesitamos de variables, como en *hay un perro con manchas y él ladra en las mañanas* es un error tratar de traducir por separado *hay un perro con manchas* y *él ladra en las mañanas* puesto que lo que existe está ligado con lo que ladra por la conjunción, de manera que debe utilizarse una variable que modele esta conexión.
- Las variables no se mencionan en español sino que son sólo un formalismo para representar individuos. Por ejemplo, la fórmula $\forall x(M(x) \rightarrow T(x))$ puede traducirse como *Cualquier minotauro es troyano* y no como *para cualquier x , si x es minotauro entonces x es troyano*. Enunciados de esta forma sólo sirven como pasos intermedios en el proceso de traducción pues no forman parte del español correcto ni de la lógica formal. A este mismo respecto es prácticamente imposible que en una traducción del español figuren variables libres.
- Los esquemas $\forall x(A \rightarrow B)$ y $\exists x(A \wedge B)$ son de gran utilidad y bastante comunes. Menos comunes, aunque también adecuados, son los esquemas $\forall x(A \wedge B)$, $\forall x(A \vee B)$ y $\exists x(A \vee B)$.
- El esquema $\exists x(A \rightarrow B)$, si bien es una fórmula sintácticamente correcta, es extremadamente raro que figure en una traducción del español.
- El hecho de que se usen dos o más variables distintas no implica que éstas representen a elementos distintos del universo, de manera que para especificar dos individuos distintos no es suficiente contar simplemente con variables distintas. Las fórmulas $\exists xP(x)$ y $\exists x\exists y(P(x) \wedge P(y))$ expresan ambas lo mismo, a saber que algo cumple P . Se debe agregar explícitamente que x e y tienen la propiedad de ser distintos, es decir $x \neq y$.

3.3.1. Juicios aristotélicos

Una gran parte de las especificaciones en lenguaje natural pueden formalizarse mediante instancias de alguno de los cuatro juicios aristotélicos básicos, los cuales se refieren a dos relaciones y expresan las posibilidades de que éstas se cumplan o no en ciertos individuos.

Ejemplo 3.10. Tomemos como universo de discurso al reino animal. Vamos a construir los llamados *juicios aristotélicos fundamentales* a partir de las propiedades *ser perico* y *ser feo*. Primero definimos los predicados necesarios:

$P(x)$ x es perico

$F(x)$ x es feo

(a) Juicio universal afirmativo: *Todos los pericos son feos*,

$$\forall x(P(x) \rightarrow F(x)).$$

(b) Juicio existencial afirmativo: *Algunos pericos son feos*,

$$\exists x(P(x) \wedge F(x)).$$

(c) Juicio existencial negativo: *Algunos pericos no son feos*,

$$\exists x(P(x) \wedge \neg F(x)).$$

(d) Juicio universal negativo: *Ningún perico es feo*, lo cual es equivalente a decir que cualquier perico no es feo o bien todos los pericos no son feos; de manera que las dos siguientes especificaciones son correctas:

$$\neg \exists x(P(x) \wedge F(x)),$$

$$\forall x(P(x) \rightarrow \neg F(x)).$$

En el siguiente ejemplo nos servimos de juicios aristotélicos para obtener especificaciones más complejas.

Ejemplo 3.11. Tenemos los siguientes predicados en el universo de discurso de los habitantes de la Ciudad de México:

$I(x)$ x es inteligente

$E(x)$ x es estudiante de la Facultad de Ciencias

$M(x)$ a x le gusta la música

Especificar con cuantificaciones los siguientes enunciados:

- Todos los estudiantes de la Facultad de Ciencias son inteligentes.

$$\forall x(E(x) \rightarrow I(x))$$

- A algunos estudiantes inteligentes les gusta la música.

$$\exists x(E(x) \wedge I(x) \wedge M(x))$$

- Todo aquel a quien le gusta la música es un estudiante que no es inteligente.

$$\forall x(M(x) \rightarrow E(x) \wedge \neg I(x))$$

Ejemplo 3.12. En este ejemplo observamos el significado de las distintas combinaciones de dos cuantificaciones. Sea $Q(x, y)$ el predicado x quiere a y .

• Todos quieren a alguien:	$\forall x \exists y Q(x, y)$
• Alguien quiere a todos:	$\exists x \forall y Q(x, y)$
• Alguien es querido por todos:	$\exists x \forall y Q(y, x)$
• Todos se quieren, o bien, todos quieren a todos:	$\forall x \forall y Q(x, y)$
• Algunos se quieren entre sí, o bien alguien quiere a alguien:	$\exists x \exists y Q(x, y)$
• Alguno no es querido por nadie:	$\exists x \forall y \neg Q(y, x)$
• Alguien no quiere a nadie:	$\exists x \forall y \neg Q(x, y)$
• Todos no quieren a alguien:	$\forall x \exists y \neg Q(x, y)$
• Nadie quiere a todos:	$\neg \exists x \forall y Q(x, y)$
• Nadie quiere a nadie:	$\forall x \forall y \neg Q(x, y)$

3.3.2. Negaciones

Con frecuencia necesitaremos traducir la negación de una cuantificación, lo cual ejemplificamos a continuación.

Ejemplo 3.13. La negación de una cuantificación puede obtenerse simplemente anteponiendo el operador de negación, por ejemplo:

- *No todos son leones* se traduce como $\neg \forall x L(x)$.
- *No existen leones* se traduce como $\neg \exists x L(x)$.

Sin embargo, estas traducciones no proporcionan información suficiente y pueden mejorarse usando equivalencias intuitivas del español, como sigue:

- *No todos son leones* es lo mismo que *existe algo que no es león* cuya traducción es:

$$\exists x \neg L(x)$$

- *No existen leones* es lo mismo que *cualquiera no es león* cuya traducción es:

$$\forall x \neg L(x)$$

Por supuesto que en los dos casos ambas traducciones deben ser equivalentes en la lógica pues lo son en español. Analizaremos esto con más detalle en la subsección 3.4.4.

Veamos un ejemplo más elaborado.

Ejemplo 3.14. Traducir el enunciado *no todos los planetas tienen una luna*. Definimos los predicados $P(x)$, $L(x)$, $T(x, y)$ para *ser planeta*, *ser luna* y x *tiene a* y respectivamente.

- Lo más simple es especificar primero la cuantificación universal y anteponer la negación, obteniendo

$$\neg \forall x (P(x) \rightarrow \exists y (L(y) \wedge T(x, y))).$$

- Otra opción es transformar la frase a una equivalente en español que permita una estructura lógica que nos dé más información. En este caso el enunciado original es equivalente a *existe un planeta que no tiene lunas*, cuya especificación es:

$$\exists x (P(x) \wedge \neg \exists y (L(y) \wedge T(x, y))).$$

- Es posible refinar aún más la traducción si observamos que la frase *no existe una luna tal que x la tenga* se puede reescribir como *para toda luna, x no la tiene*, obteniendo así la especificación más refinada posible.

$$\exists x (P(x) \wedge \forall y (L(y) \rightarrow \neg T(x, y))).$$

3.3.3. Contando objetos

Como ya mencionamos al principio de esta sección, el hecho de usar variables diferentes no implica que se refieran necesariamente a individuos distintos, de manera que para representar cantidades particulares se requiere especificar explícitamente que ciertos individuos no son iguales. Veamos algunos ejemplos.

Ejemplo 3.15. En las siguientes especificaciones se utiliza el predicado binario de igualdad = de manera infija. Además las fórmulas del esquema $\neg(t = s)$ se escriben como $t \neq s$, como es usual en matemáticas.

- *Hay al menos una luna*, esto resulta equivalente a *hay una luna*:

$$\exists x L(x).$$

- *Hay más de una luna, es decir, existen al menos dos lunas:*

$$\exists x \exists y (L(x) \wedge L(y) \wedge x \neq y).$$

Obsérvese que se hace explícito el hecho de que las lunas denotadas por x e y son diferentes.

- *Hay al menos tres lunas.* De manera similar al enunciado anterior usamos tres variables y hacemos explícito el hecho de que denotan a tres individuos diferentes:

$$\exists x \exists y \exists z (L(x) \wedge L(y) \wedge L(z) \wedge x \neq y \wedge x \neq z \wedge y \neq z).$$

En general es posible definir el enunciado *hay al menos n objetos* de manera análoga. Sin embargo es imposible especificar que existe una infinidad de objetos. ¿Por qué?

- *Existe un único sol.* Lo usual aquí es especificar que hay un sol y que cualesquiera dos soles en realidad son iguales:

$$\exists x (S(x) \wedge \forall y \forall z (S(y) \wedge S(z) \rightarrow y = z)).$$

Este esquema es de gran utilidad en matemáticas y suele abreviarse como $\exists! x P(x)$ para cualquier predicado P .

- *Hay a lo más un sol, lo cual es equivalente a Cualesquiera dos soles son el mismo.*

$$\forall x \forall y (S(x) \wedge S(y) \rightarrow x = y).$$

Obsérvese que esta especificación incluye el caso en que no haya soles. Otra posibilidad es especificar que *No es cierto que existen al menos dos soles*.

3.3.4. Micromundos

En inteligencia artificial un micromundo es un modelo artificialmente simple de una situación real; por ejemplo, si se desea programar un robot para que mueva objetos de manera inteligente, basta modelar los movimientos deseados sin tomar en cuenta sus dimensiones reales ni la cantidad total de objetos en juego, para lo cual basta considerar una idealización del mundo real con pocos objetos. A continuación especificamos algunas descripciones para dos micromundos similares a los utilizados en inteligencia artificial.

El micromundo de cubos

En este micromundo hay cubos de color amarillo, azul o rojo. Un cubo puede estar sobre otro o en el piso. Definimos los predicados $S(x, y)$ representando que el cubo x

está sobre el cubo y ; $A(x)$, $Az(x)$ y $R(x)$ que representan que un cubo puede ser de color amarillo, azul o rojo respectivamente; $L(x)$ significa que el cubo x está libre, es decir que ningún cubo está sobre el cubo x ; y la constante p representa al piso. Veamos algunas especificaciones.

- Ningún cubo amarillo está libre:

$$\forall x(A(x) \rightarrow \neg L(x)).$$

- Hay un cubo azul libre y un cubo rojo libre:

$$\exists x \exists y (Az(x) \wedge L(x) \wedge R(y) \wedge (y)).$$

- Cualquier cubo amarillo tiene un cubo sobre él:

$$\forall x \left(A(x) \rightarrow \exists y (S(y, x) \wedge x \neq y) \right).$$

- No todos los cubos azules están libres:

$$\exists x (Az(x) \wedge \neg L(x)).$$

- Hay un cubo azul sobre el piso con un cubo amarillo sobre él y un cubo rojo sobre el amarillo:

$$\exists x \exists y \exists w \left(Az(x) \wedge A(y) \wedge R(w) \wedge S(x, p) \wedge S(y, x) \wedge S(w, y) \right).$$

Un mundo de triángulos, círculos y cuadrados

El micromundo consta de una cuadrícula de cualquier tamaño donde en cada cuadro puede haber figuras que son círculos, cuadrados o triángulos, las cuales pueden ser pequeñas, medianas o grandes. También se tienen las relaciones dadas por la posición: sur, norte, este, oeste; y las relaciones dadas por estar en la misma columna y en el mismo renglón. Los predicados para las figuras son: $T(x)$, $C(x)$ y $S(x)$ para triángulo, círculo y cuadrado respectivamente; para tamaño tenemos $P(x)$, $M(x)$ y $G(x)$ para pequeño, mediano y grande. Para la posición tenemos $Su(x, y)$, $N(x, y)$, $E(x, y)$ y $O(x, y)$; por ejemplo $N(x, y)$ significa x está al norte de y . Finalmente tenemos $Co(x, y)$ y $R(x, y)$ para indicar que x está en la misma columna o renglón que y , respectivamente. Hagamos algunas descripciones para este micromundo.

- Hay círculos medianos y cuadrados grandes:

$$\exists x (C(x) \wedge M(x)) \wedge \exists y (S(y) \wedge G(y)).$$

- No hay cuadrados pequeños:

$$\forall x (S(x) \rightarrow \neg P(x)).$$

- Hay un triángulo al sur de todos los círculos:

$$\exists x (T(x) \wedge \forall y (C(y) \rightarrow Su(x, y))).$$

- No hay dos triángulos en el mismo renglón:

$$\neg \exists y \exists x (T(x) \wedge T(y) \wedge R(x, y)).$$

- Hay un círculo tal que todos los círculos al oeste de él son grandes:

$$\exists x (C(x) \wedge \forall y (C(y) \wedge O(y, x) \rightarrow G(y))).$$

Con esto terminamos esta sección y a continuación nos ocupamos brevemente de algunos aspectos semánticos de la lógica de predicados. Regresaremos a los micromundos después de revisar algunos conceptos relacionados.

Ejercicios

3.3.1.- Para los siguientes predicados propón un universo de discurso adecuado:

- | | | |
|-----|---------------------------|---------------------------------|
| (a) | $A(x)$ | x tiene los pétalos amarillos |
| (b) | $menor(x, \text{MÍNIMO})$ | x es menor que el mínimo |
| (c) | $P(x, y)$ | x es padre de y |
| (d) | $R(x)$ | x ruge |
| (e) | $mayor(x, 0)$ | x es mayor que 0 |

3.3.2.- Considera los siguientes enunciados donde se usan predicados. Determina cuáles son los predicados necesarios y escribe cada uno de los enunciados en cálculo de predicados. El universo de discurso es el conjunto de todas las personas.

- Los enemigos de mis enemigos son mis amigos.
- Los amigos van y vienen; los enemigos se acumulan.
- Juan aprecia a María y María aprecia a Lupita; entonces Juan aprecia a Lupita.
- Juan es familiar de Rosa; Rosa es familiar de Guillermo; entonces Juan es familiar de Guillermo.

3.3.3.- Considera los siguientes enunciados donde se usan predicados. Determina cuáles son los predicados necesarios y escribe cada uno de los enunciados en cálculo de predicados. El universo de discurso es el conjunto de todos los animales.

- (a) Los leones comen carne cruda.
- (b) Sólo los leones rugen.
- (c) El piquete de abejas duele mucho.
- (d) La boa constrictora no es venenosa.
- (e) La víbora de cascabel es venenosa.
- (f) No hay mamíferos venenosos.

3.3.4.- Considera los siguientes enunciados donde se usan predicados. Determina cuáles son los predicados necesarios y escribe el argumento lógico usándolos. El universo de discurso son los animales de la selva.

- (a) Los leones son feroces.
- (b) Los elefantes asustan a los leones.
- (c) Los ratones asustan a los elefantes.
- (d) De esto, los ratones asustan a animales feroces.

3.3.5.- Considera los siguientes enunciados donde se usan predicados. Determina cuáles son los predicados necesarios y escribe el argumento lógico usándolos. El universo de discurso son las computadoras asignadas a Ciencias de la Computación.

- (a) La computadora x ha sido invadida (*hackeada*) desde la computadora y .
- (b) La computadora x funciona con el sistema operativo Linux.
- (c) La computadora x funciona con el sistema operativo Windows.
- (d) El servidor del taller de Lenguajes de Programación, que funciona con Linux, no fue *hackeado*.
- (e) El servidor del taller de Ingeniería de Software, que funciona con Windows, sí fue *hackeado*.
- (f) Si una computadora tiene el sistema Linux no puede ser *hackeada*.

3.3.6.- Traduce los siguientes enunciados a cuantificaciones universales y (o) existenciales, donde el universo de discurso son los días de la semana y suponiendo los predicados y constantes que siguen:

- $S(x)$ el día x está soleado
- $N(x)$ el día x está nublado
- L la constante "Lunes"
- M la constante "Martes"

- (a) Todos los días están soleados.
- (b) Algunos días no están nublados.
- (c) Todo día que está soleado no está nublado.
- (d) Algunos días están soleados y nublados.
- (e) Ningún día es al mismo tiempo soleado y nublado.
- (f) Siempre está soleado sólo si está nublado.
- (g) Ningún día es soleado.
- (h) El lunes estuvo soleado, por lo que todos los días estarán soleados.
- (i) Se nubló el lunes y el martes.
- (j) Si algún día está nublado, entonces todos los días estarán soleados.

3.3.7.- Escribe las fórmulas con cuantificadores de los siguientes enunciados, suponiendo que el universo de discurso son las personas y usando la siguiente asignación para los predicados:

- $J(x)$ x es juez
- $A(x)$ x es abogado
- $M(x)$ x es mujer
- $Q(x)$ x es químico
- $R(x, y)$ x respeta a y

- (a) Hay algunas abogadas mujeres que son químicas.
- (b) Ninguna mujer es abogado y químico.
- (c) Algunos abogados sólo respetan jueces.
- (d) Los jueces respetan sólo a los jueces.
- (e) Todas las abogadas mujeres respetan a algún juez.
- (f) Algunas mujeres no respetan a ningún abogado.

3.3.8.- Sea $T(x, y)$ el predicado x puede tomarle el pelo a y , donde el dominio consiste de todos los seres humanos. Usa cuantificadores para expresar cada uno de los siguientes enunciados:

- (a) Todo mundo puede tomarle el pelo a Juan.
- (b) María puede tomarle el pelo a cualquiera.
- (c) Cualquiera puede tomarle el pelo a alguien.
- (d) Nadie puede tomarle el pelo a cualquiera.
- (e) Siempre hay alguien que le puede tomar el pelo a cualquiera.
- (f) Hay exactamente una persona a quien cualquiera puede tomarle el pelo.

- (g) Hay alguien que puede tomarle el pelo a exactamente una persona distinta de sí mismo.

3.3.9.- Sea $S(x)$ el predicado *x es un estudiante*, $P(x)$ el predicado *x es un maestro* y $Q(x, y)$ el predicado *x le hace una pregunta a y*, donde el dominio consiste de toda la comunidad de la Facultad de Ciencias. Traduce los siguientes enunciados a cuantificaciones:

- (a) Luisa le preguntó al Profesor Miguel una pregunta.
- (b) Cada estudiante le hizo una pregunta al Profesor García.
- (c) Todo profesor ha hecho una pregunta al Profesor López o bien el Profesor Pérez les ha hecho una pregunta.
- (d) Algún estudiante no le ha hecho preguntas a ningún profesor.
- (e) Hay un profesor a quien ningún estudiante le ha hecho nunca ninguna pregunta.
- (f) Un estudiante le ha hecho preguntas a todos los profesores.
- (g) Hay un profesor que le ha hecho preguntas a cada uno de los profesores.
- (h) Hay un estudiante al que ningún profesor le ha hecho preguntas.

3.3.10.- Hacer las siguientes traducciones, definiendo previamente el universo de discurso y los predicados necesarios.

- (a) Los perros muerden a los carteros.
- (b) Existe un perro que muerde a los carteros.
- (c) Existe un cartero que es mordido por todos los perros.
- (d) Hay un perro que no muerde carteros.
- (e) Hay un cartero que no es mordido por perros.
- (f) Hay un perro que es cartero y se muerde a sí mismo.

3.3.11.- Traduce las siguientes oraciones a fórmulas, donde el universo de discurso son las novelas, usando los siguientes predicados:

- $S(x)$ x es una novela de espías
- $M(x)$ x es una novela de misterio
- $L(x)$ x es larga
- $M(x, y)$ x es mejor que y

- (a) Todas las novelas de espías son largas.
- (b) No todas las novelas de misterio son una novela de espías.
- (c) Sólo las novelas de misterio son largas.
- (d) Algunas novelas de espías son de misterio.
- (e) Las novelas de espías son mejores que las de misterio.

(f) Sólo las novelas de espías son mejores que las de misterio.

3.3.12.- Traduce los siguientes argumentos a la lógica de predicados. Especifica el universo de discurso y explica el significado de cada predicado usado.

- (a) A algunos pacientes les caen bien todos los doctores. A ningún paciente le cae bien una enfermera. Por lo tanto ningún doctor es enfermera.
- (b) Todos los empleados de la empresa INC deben de saber usar Cobol. Todos los empleados de INC que escriben aplicaciones deben de saber Excel. Roxana trabaja para la empresa INC, pero ella no sabe Excel. Ingrid sabe Excel pero no Cobol. Por lo tanto Roxana no escribe aplicaciones e Ingrid no trabaja para INC.

3.3.13.- Expresa las siguientes especificaciones de un sistema de cómputo usando lógica de predicados. Declara previamente los predicados que vas a utilizar.

- (a) Si hay menos de 30 megabytes libres en el disco duro, se envía un mensaje de advertencia a todos los usuarios.
- (b) Ningún directorio puede abrirse ni ningún archivo puede cerrarse si se han detectado errores en el sistema.
- (c) El sistema de archivos no puede respaldarse si hay algún usuario con sesión activa.
- (d) Pueden recibirse archivos de video cuando hay al menos 8 megabytes de memoria disponible y la velocidad de conexión es al menos de 56 kilobits por segundo.

3.4. Semántica informal

En esta sección nos dedicaremos a dar ciertas ideas acerca de la semántica de la lógica de predicados. Lo haremos de manera informal e intuitiva, dado que la semántica formal requiere de mecanismos matemáticos sofisticados que caen fuera del alcance de este libro.

3.4.1. Dominios de interpretación

Antes de poder determinar cuándo una fórmula de la lógica de predicados es verdadera debemos formalizar el concepto de universo de discurso; eso se hace mediante un dominio de interpretación, el cual es un conjunto no vacío en el que se definirán matemáticamente los significados de los símbolos de constante, predicado y función usados en las especificaciones formales, de manera que un símbolo de constante será un individuo y los predicados y funciones serán operadores entre elementos del universo, que devuelven otro elemento

del universo en el caso de una función, o bien un valor booleano en el caso de un predicado. Veamos algunos ejemplos.

Tabla 3.1 Distintos universos y dominios

Con:	Se representa a	Operadores	Comentarios:
\mathbb{N}	Los números naturales	0, 1 + × <i>mod</i> <i>div</i> >, <, <i>par?</i> , <i>primo?</i>	Los números naturales tienen definidas la suma y el producto. También tienen residuo entero (<i>mod</i>) y cociente entero (<i>div</i>). Como interpretación de constantes tenemos la identidad 1 y elemento nulo 0, así como los operadores de decisión para orden, par y primo que corresponden a predicados.
\mathbb{Z}	Los enteros	+ − × <i>mod</i> <i>div</i> <i>neg?</i> ,	Los operadores para los números naturales siguen siendo válidos y agregamos la operación de resta, el operador de decisión <i>ser negativo</i> y el operador de divisibilidad.
\mathbb{Q}	Números racionales	÷, <i>num</i> <i>den</i> <i>simp</i> <i>etc.</i>	Aquí ya es posible usar la división; tenemos además operadores que devuelven el numerador o denominador de un racional y el operador de simplificación que elimina factores comunes.
\mathbb{R}	Los números reales	$\sqrt{\quad}$ [] [] <i>rac?</i> π e	Agregamos las operaciones de raíz cuadrada (válida sólo para reales positivos), mayor entero menor o igual, menor entero mayor o igual, el predicado ser racional y las constantes π y e
\mathbb{B}	Los booleanos {1, 0}	\wedge , \vee \rightarrow , \leftrightarrow \neg	\mathbb{B} es el tipo booleano, que recibe su nombre del matemático inglés George Boole (1815-1864) que creó las bases algebraicas para la lógica. Los operadores son los mismos definidos en el capítulo dos.

Continúa en la siguiente página

Tabla 3.1 Distintos universos y dominios

Continúa de la página anterior

Con:	Se representa a	Operadores	Comentarios:
\mathcal{MB}	Micromundo de cubos	piso <i>azul?</i> <i>verde?</i> <i>libre?</i> <i>sobre?</i> <i>arriba?</i> mover etc.	El universo es heterogéneo pues tiene al piso. Aquí se tienen predicados para los colores; si se desea tener una función que devuelva el color de un objeto es necesario añadir los colores al dominio y también predicados para decidir si un individuo es color o cubo.
\mathcal{H}	Los seres vivos que pertenecen al reino Fungi	<i>comestible?</i> <i>venenoso?</i> <i>=familia?</i> <i>mismo?</i> etc.	Si se desean operaciones que devuelvan lugar de origen o dimensiones, por ejemplo, el dominio debe volverse heterogéneo para incluir lugares y números.
\mathcal{FC}	La Facultad de Ciencias	inscribir estudiar calificar <i>estudiante?</i> <i>profesor?</i> <i>reprueba?</i> etc.	Un dominio donde hay personas, salones, clases, números de cuenta, libros, apuntes, etc..
\mathcal{MF}	Micromundo de figuras	<i>cuadrado?</i> <i>círculo?</i> <i>triángulo?</i> <i>pequeño?</i> <i>al-norte?</i> etc.	En este dominio sólo hay figuras pero no es completamente homogéneo pues las figuras son de tres clases diferentes, por lo que necesitamos de los predicados para cuadrado, círculo y triángulo. Sin embargo, y a diferencia de otros mundos heterogéneos, el dominio está claramente partido en tres clases distintas de objetos.
<i>MiBiblio</i>	Una biblioteca	elegir prestar ordenar <i>está?</i> etc.	El dominio es heterogéneo y debe contener libros, libreros, personas, etc.

3.4.2. Noción informal de verdad

Si el dominio de interpretación (universo de discurso) que esté en uso es finito, entonces podemos asignar el valor de falso o verdadero a cada predicado analizando todas las posibles combinaciones de individuos en dicho universo de discurso. Por ejemplo, si tenemos el predicado $>$ (*mayor que*) y nuestro universo de discurso consiste de los enteros 1, 2, 3 y 4, entonces podemos hacer una tabla que asigne los valores de falso o verdadero a cada pareja de individuos, como podemos observar en la tabla 3.2 en la siguiente página.

Tabla 3.2 Asignación para el predicado $>$ (mayor que)

$>$	1	2	3	4
1	0	0	0	0
2	1	0	0	0
3	1	1	0	0
4	1	1	1	0

Por supuesto que esto resulta más complicado si el universo es demasiado grande, como en el caso en que el universo conste de los enteros $1, \dots, 1000$. Por otra parte, si el universo en cuestión consta de todos los números naturales, resulta imposible construir la tabla pues tendría un número infinito de columnas y un número infinito de renglones:

$>$	1	2	3	4	5	6	7	8	10	11	12	13	14	15	16	17	18	19	20
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
.....																						
.....																						

Si el índice de un predicado es uno o dos y el universo finito, también es fácil ver la asignación en una tabla. Sin embargo, si el índice es tres o más, aunque el universo sea relativamente pequeño, ya no es fácil visualizar dicha asignación. De manera que el uso de

tablas de verdad para la lógica de predicados es inadecuado. Esto es de esperarse, ya que la noción de verdad en lógica de predicados es mucho más complicada puesto que depende de un mundo en particular, al contrario de lo que sucedía en la lógica de proposiciones, donde en el fondo el único mundo o universo de discurso es el de los valores booleanos *cierto* o *falso*. En esta nueva lógica, el universo de discurso puede incluir literalmente cualquier cosa: números, conjuntos, piedras, flores, árboles, palabras, galaxias, etcétera. De manera que la noción de verdad dependerá del mundo que hayamos fijado de antemano. Por supuesto que al cambiar éste, el valor de verdad de una fórmula también puede hacerlo.

Antes de dar una definición de verdad analicemos el caso de los cuantificadores con un ejemplo sencillo en el micromundo de figuras:

- *Todos son círculos*: $\forall x C(x)$. Esto será cierto si y sólo si al revisar cada objeto del micromundo, el objeto resulta ser un círculo. Si suponemos que hay n objetos, denotados por las constantes a_1, \dots, a_n , entonces $\forall x C(x)$ será cierto si y sólo si $C(a_1)$ es cierta y $C(a_2)$ es cierta y \dots y $C(a_n)$ es cierta; es decir, si y sólo si la conjunción $C(a_1) \wedge \dots \wedge C(a_n)$ es cierta. Obsérvese que esto no puede ser una definición, pues en el caso en que el universo sea infinito es imposible formar la conjunción de todos los objetos.
- *Existe algo pequeño*: $\exists x P(x)$. Similarmente a la cuantificación universal, esta fórmula es cierta si y sólo si alguno de los objetos a_1, \dots, a_n resulta ser pequeño, es decir, si la disyunción $P(a_1) \vee \dots \vee P(a_n)$ es cierta.

Esta idea intuitiva nos lleva a una definición informal de verdad para cualquier fórmula de la lógica de predicados, la cual damos a continuación.

Definición 3.3.16 Dada una fórmula A de la lógica de predicados, definimos cuándo A es verdadera en un mundo o universo de discurso dado \mathcal{M} , de acuerdo a su forma sintáctica, como sigue:

- Si A es una fórmula atómica, digamos $P(t_1, \dots, t_n)$, entonces A es verdadera en \mathcal{M} si y sólo si los valores de los términos t_1, \dots, t_n como individuos de \mathcal{M} están en la relación del universo definida por P .
- Si A es una fórmula proposicional³, entonces usamos los criterios de verdad de la lógica proposicional.
- Si $A = \forall x B$ es una fórmula universal, entonces A es verdadera en \mathcal{M} si y sólo si B es verdadera en \mathcal{M} para todos los posibles valores de x como individuo de \mathcal{M} .
- Si $A = \exists x B$ es una fórmula existencial, entonces A es verdadera en \mathcal{M} si y sólo si B es verdadera en \mathcal{M} para algún valor de x como individuo de \mathcal{M} .

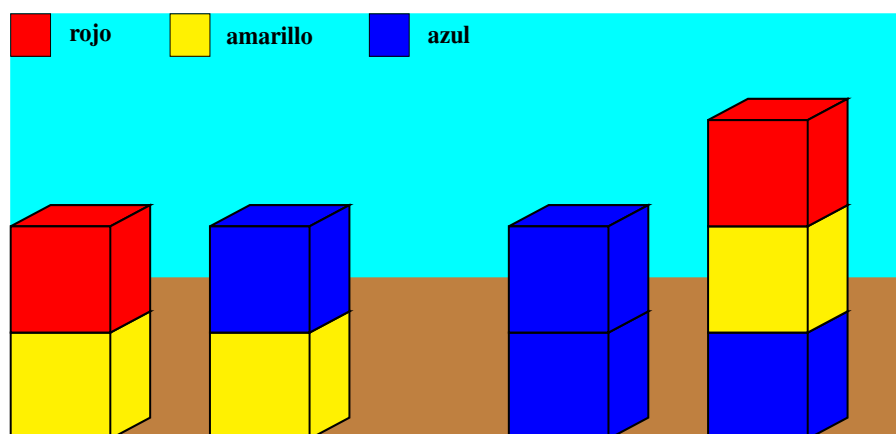
³Es decir una fórmula que pertenece a algún esquema de la lógica proposicional, con predicados en lugar de variables proposicionales

Esta definición es informal puesto que en el caso en que el universo de discurso sea infinito, no queda claro en general cómo mostrar que la fórmula $\forall x B$ es cierta para todos los valores posibles de x .

3.4.3. Verdad en micromundos

A continuación regresamos a nuestros dos micromundos particulares empezando con el mundo de los cubos para ejemplificar la definición informal de semántica que acabamos de enunciar. Nos referiremos al mundo particular que se encuentra en la figura 3.1, a continuación.

Figura 3.1 Micromundo de cubos



Queremos ahora determinar la semántica de algunas fórmulas en este micromundo.

- Cualquier cubo rojo está libre:

$$\forall x (R(x) \rightarrow L(x)).$$

Verdadero, pues los cubos rojos en la primera y cuarta torre, que son todos los cubos rojos en este micromundo, están libres.

- Todos los cubos sobre el piso son azules:

$$\forall x (S(x, p) \rightarrow Az(x)).$$

Falso, pues la primera y segunda torre tienen cubos amarillos sobre el piso, por lo que no todos los cubos sobre el piso son azules.

- Cualquier cubo que esté sobre un cubo amarillo es rojo o azul:

$$\forall x (\exists y (A(y) \wedge S(x, y)) \rightarrow R(x) \vee Az(x)).$$

Cierto, ya que los cubos amarillos de la primera y cuarta torre tienen a un cubo rojo encima; y el cubo amarillo de la segunda torre tiene encima a un cubo azul.

- Hay un cubo rojo sobre un cubo rojo:

$$\exists x \exists y (R(x) \wedge R(y) \wedge S(x, y)).$$

Falso. Los dos cubos rojos, en la primera y cuarta torre, son libres, por lo que no tienen encima a ningún cubo, en particular a uno rojo.

- Hay un cubo amarillo libre sobre el piso:

$$\exists x (A(x) \wedge L(x) \wedge S(x, p)).$$

Falso. No hay ningún cubo libre sobre el piso, en particular que sea amarillo, por lo que la fórmula es falsa.

- Ningún cubo está sobre el piso:

$$\forall x \neg S(x, p).$$

Falso, pues el cubo amarillo en la primera torre sí está sobre el piso.

- Hay un cubo amarillo que está sobre uno azul y hay un cubo azul sobre él:

$$\exists x \exists y (A(x) \wedge Az(y) \wedge S(x, y) \wedge \exists w (Az(w) \wedge S(w, x))).$$

Falso. No hay una torre que contenga una secuencia de cubo azul, cubo amarillo y cubo azul.

- Todos los cubos están sobre algo:

$$\forall x \exists y S(x, y).$$

Verdadera. Todos los cubos están o sobre el piso o sobre algún otro cubo.

Veamos ahora un micromundo particular de figuras geométricas, en la figura 3.2 de la siguiente página, con los predicados que ya definimos para los mundos de figuras geométricas, y decidamos cuáles de las fórmulas que le siguen son falsas o verdaderas. Arriba del mundo observamos los tres tamaños de figuras disponibles. Además usaremos coordenadas para renglón y columna como un auxiliar para señalar cada cuadro en particular.

- Hay círculos medianos y cuadrados grandes:

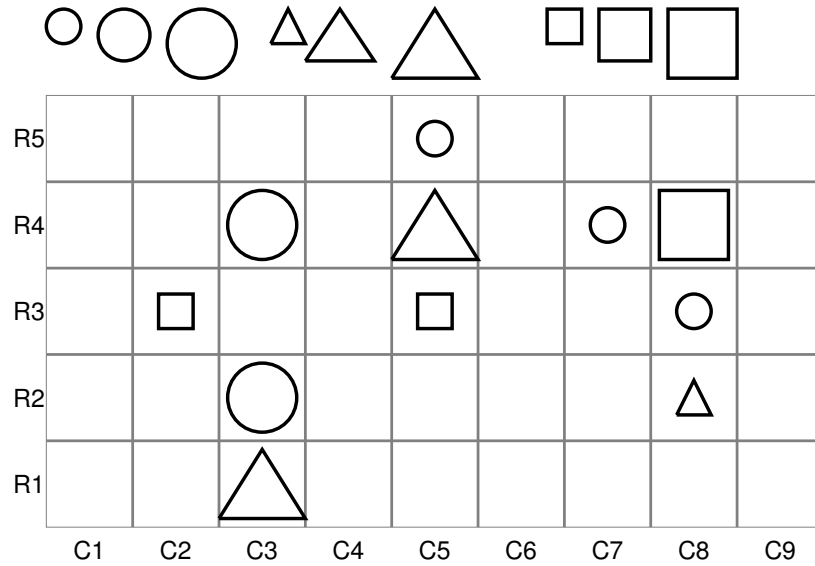
$$\exists x (C(x) \wedge M(x)) \wedge \exists y (S(y) \wedge G(y)).$$

Falso, pues no hay ningún círculo mediano.

- No hay cuadrados pequeños:

$$\forall x (S(x) \rightarrow \neg P(x)).$$

En (R3C2) hay un cuadrado pequeño, por lo tanto la fórmula es falsa.

Figura 3.2 Mundo particular de figuras geométricas

- Ningún cuadrado está al norte de un círculo grande:

$$\neg \exists x \left(S(x) \wedge \exists y (C(y) \wedge G(y) \wedge N(x, y)) \right).$$

Falso, pues el cuadrado en (R3C5) sí está al norte del círculo grande en (R2C3).

- Todos los círculos medianos están al oeste de un mismo triángulo grande:

$$\exists x \left(T(x) \wedge G(x) \wedge \forall y (C(y) \wedge M(y) \rightarrow O(y, x)) \right).$$

Como sí hay un triángulo grande en (R4C5) pero no hay círculos medianos, la implicación tiene antecedente falso y la fórmula se evalúa a verdadera.

- Todos los cuadrados pequeños están al sur de cualquier triángulo:

$$\forall x \left(S(x) \wedge P(x) \rightarrow \forall y (T(y) \rightarrow Su(x, y)) \right)$$

Los cuadrados pequeños están en (R3), pero no están al sur del triángulo en (R2C8), por lo que la fórmula es falsa.

- Si dos cuadrados están en el mismo renglón, entonces cualquier triángulo al sur de ambos es mediano:

$$\forall x \forall y \left(S(x) \wedge S(y) \wedge R(x, y) \rightarrow \forall z (T(z) \wedge Su(z, x) \wedge Su(z, y) \rightarrow M(z)) \right).$$

La fórmula es falsa pues para los dos cuadrados en (R3), ninguno de los triángulos al sur es mediano.

- No hay dos triángulos medianos en la misma columna; y si un triángulo es grande, entonces hay un círculo pequeño al este de él:

$$\neg \exists x \exists y (T(x) \wedge T(y) \wedge Co(x, y)) \wedge \forall z (T(z) \wedge G(z) \rightarrow \exists w (C(w) \wedge P(w) \wedge E(w, z))).$$

El primer operando de la conjunción es verdadero, porque como no hay triángulos medianos, cualquier cosa que se diga de ellos es verdadera; tanto para el triángulo grande en (R4C5) como el de (R1C3), hay dos círculos pequeños, (R4C7) y (R3C8), que se encuentran al este de cualquiera de ellos. Por lo tanto, el segundo operando de la conjunción es verdadero y la fórmula completa también.

3.4.4. Algunas equivalencias lógicas

Con frecuencia un enunciado en español puede reescribirse de manera que la estructura lógica sea más clara, tal como lo hicimos en algunos ejemplos de la sección 3.3. En este caso ambos enunciados deben ser equivalentes, en el sentido de que cualquier conclusión obtenida con uno de ellos debe seguir siendo válida usando el otro. Esta situación se formaliza mediante el concepto de equivalencia lógica, que ya estudiamos para la lógica proposicional, y que en la lógica de predicados tiene el mismo significado: dos fórmulas A y B son lógicamente equivalentes, denotado $A \equiv B$, si y sólo si ambas son verdaderas exactamente en los mismos mundos o interpretaciones. A continuación discutimos algunas equivalencias lógicas de utilidad.

Equivalencias proposicionales

Todas las equivalencias lógicas para la lógica proposicional siguen siendo válidas en la lógica de predicados y pueden usarse también dentro de una cuantificación.

Ejemplo 3.17. Las siguientes fórmulas son equivalentes debido al uso de una ley proposicional de equivalencia lógica.

- *Hay un círculo grande es equivalente a hay alguna figura grande que es círculo:*

$$\exists x (C(x) \wedge G(x)) \equiv \exists x (G(x) \wedge C(x)).$$

- *Cualquier figura o es triángulo o es mediana equivale a que toda figura que no es triángulo es mediana:*

$$\forall x (T(x) \vee M(x)) \equiv \forall x (\neg T(x) \rightarrow M(x)).$$

- *No es cierto que hay un cuadrado y que todas las figuras sean pequeñas significa lo mismo que o bien no hay cuadrados o bien no todas las figuras son pequeñas:*

$$\neg (\exists x S(x) \wedge \forall y P(y)) \equiv \neg \exists x S(x) \vee \neg \forall y P(y).$$

- Si todas las figuras son cuadrados entonces no hay figuras grandes equivale a si existen figuras grandes entonces no todas son cuadrados:

$$\forall x S(x) \rightarrow \neg \exists y G(y) \equiv \exists y G(y) \rightarrow \neg \forall x S(x).$$

Negación de cuantificadores

Volviendo a la idea de que una cuantificación puede entenderse como una conjunción o disyunción en el caso de un universo finito, podemos analizar de qué forma interactúan los cuantificadores con la negación. Por ejemplo, la fórmula $\neg \forall x C(x)$ (*no todos son círculos*) es cierta si y sólo si la negación de la conjunción de todos los objetos del universo, dada por $\neg (C(a_1) \wedge \dots \wedge C(a_n))$, es cierta; es decir, usando las leyes de De Morgan, $\neg C(a_1) \vee \dots \vee \neg C(a_n)$ es cierta, lo cual equivale a la fórmula existencial $\exists x \neg C(x)$; o lo que es lo mismo, *a existe algo que no es círculo*. Similarmente podemos analizar la negación de una fórmula existencial. Por esta razón es que las leyes de negación, que enunciamos enseguida, también se conocen como leyes de De Morgan generalizadas.

Leyes de negación:

$$\neg \forall x A \equiv \exists x \neg A \quad (3.30)$$

$$\neg \exists x A \equiv \forall x \neg A \quad (3.31)$$

Obsérvese que estas equivalencias permiten mover una negación hacia el alcance de un cuantificador, intercambiando cuantificadores.

Ejemplo 3.18. Mostramos aquí el uso de las leyes de negación para transformar una fórmula de manera que la negación se aplique únicamente a predicados.

- *No es cierto que si hay un triángulo entonces todas las figuras son medianas.*

$$\begin{aligned} \neg (\exists x T(x) \rightarrow \forall y M(y)) &\equiv \exists x T(x) \wedge \neg \forall y M(y) \\ &\equiv \exists x T(x) \wedge \exists y \neg M(y). \end{aligned}$$

Hay un triángulo y no todas las figuras son medianas, lo que equivale asimismo a *hay un triángulo y hay una figura que no es mediana*.

En lo que sigue, el dominio de interpretación son los habitantes de la Ciudad de México, los lapsos de tiempo y los exámenes; utilizaremos los siguientes predicados:

$F(x)$:	x es estudiante de la Facultad de Ciencias	$I(x)$:	x es inteligente
$A(x)$:	x es alumno	$T(x)$:	x es un tiempo
$E(x, y)$:	x estudia en el tiempo y	$R(x)$:	x reprueba
$C(x)$:	el examen x fue calificado	$P(x)$:	x es un examen

- *No es cierto que todos los estudiantes de la Facultad de Ciencias sean inteligentes:*

$$\begin{aligned}\neg \forall x (F(x) \rightarrow I(x)) &\equiv \exists x \neg (F(x) \rightarrow I(x)) \\ &\equiv \exists x (F(x) \wedge \neg I(x))\end{aligned}$$

Hay un estudiante inscrito en la Facultad de Ciencias que no es inteligente.

- *No hay alumnos que estudien todo el tiempo:*

$$\begin{aligned}\neg \exists x (A(x) \wedge \forall y (T(y) \rightarrow E(x, y))) &\equiv \forall x \neg (A(x) \wedge \forall y (T(y) \rightarrow E(x, y))) \\ &\equiv \forall x (\neg A(x) \vee \neg \forall y (T(y) \rightarrow E(x, y))) \\ &\equiv \forall x (\neg A(x) \vee \exists y \neg (T(y) \rightarrow E(x, y))) \\ &\equiv \forall x (\neg A(x) \vee \exists y (T(y) \wedge \neg E(x, y))) \\ &\equiv \forall x (A(x) \rightarrow \exists y (T(y) \wedge \neg E(x, y)))\end{aligned}$$

Para cualquier alumno hay un tiempo en el que no estudia.

- *No es cierto que o algún examen no se calificó o todos los alumnos reprobaron el curso:*

$$\begin{aligned}\neg (\exists x (P(x) \wedge \neg C(x)) \vee \forall y (A(y) \rightarrow R(y))) &\equiv \neg \exists x (P(x) \wedge \neg C(x)) \wedge \neg \forall y (A(y) \rightarrow R(y)) \\ &\equiv \forall x \neg (P(x) \wedge \neg C(x)) \wedge \exists y \neg (A(y) \rightarrow R(y)) \\ &\equiv \forall x (\neg P(x) \vee C(x)) \wedge \exists y (A(y) \wedge \neg R(y)) \\ &\equiv \forall x (P(x) \rightarrow C(x)) \wedge \exists y (A(y) \wedge \neg R(y))\end{aligned}$$

Todos los exámenes se calificaron y algún alumno no reprobó.

Distributividad

Una vez que hemos visto como interactúan los cuantificadores con la negación, resulta natural preguntarse qué sucede con los demás conectivos proposicionales frente a los cuantificadores. Para esto presentamos algunas leyes distributivas entre cuantificadores y conectivos.

$$\forall x (A \wedge B) \equiv \forall x A \wedge \forall x B \quad (3.32)$$

El lado izquierdo nos dice que para todo objeto x se cumple $A \wedge B$, lo cual equivale a que para todo individuo se cumplen tanto A como B . ¿Qué sucede si cambiamos la conjunción por disyunción?

Para el cuantificador existencial tenemos la siguiente equivalencia:

$$\exists x(A \vee B) \equiv \exists xA \vee \exists xB \quad (3.33)$$

Si un individuo cumple $A \vee B$, entonces o cumple A o cumple B , de donde la disyunción de la derecha es válida. ¿Qué sucede si cambiamos la disyunción por conjunción?

Cuantificación vacua

Consideremos el siguiente enunciado: *para cualquier individuo, Berlín es la capital de Alemania*, el cual se especifica como $\forall xC(b, a)$; consideremos también el enunciado *existe un individuo tal que todos son leones*, representado con $\exists x\forall yL(y)$ o inclusive con $\exists x\forall xL(x)$, donde la variable x de la cuantificación existencial es ocultada por la de la cuantificación universal, lo que hace que $L(x)$ haga referencia a la variable de la cuantificación universal. Este tipo de cuantificaciones, donde la variable cuantificada no figura libre en el alcance de la cuantificación, se conoce como cuantificación vacua o nula. Con respecto a su valor de verdad, de acuerdo a nuestra definición, $\forall xC(b, a)$ es verdadera si y sólo si $C(b, a)$ es verdadera para cualquier valor de x como un individuo particular, pero como x no figura en $C(b, a)$, basta mostrar la verdad de esta última fórmula, es decir, la cuantificación no aporta nada a la evaluación de la fórmula original y por lo tanto puede eliminarse mediante las siguientes equivalencias:

Cuantificadores vacuos: si x no figura libre en A entonces

$$\forall xA \equiv A, \quad (3.34)$$

$$\exists xA \equiv A, \quad (3.35)$$

donde A puede ser, a su vez, una cuantificación con la misma variable cuantificadora o con una distinta. En particular estas equivalencias permiten eliminar cuantificadores múltiples, puesto que $\forall x\forall xA \equiv \forall xA$ y $\exists x\exists xA \equiv \exists xA$.

Prenexación

El proceso de prenexación permite manipular un esquema proposicional binario, donde uno de los operandos es una cuantificación y la variable cuantificada en este operando no figura libre en el otro operando. El objetivo de la manipulación es “factorizar” el cuantificador, sumergiendo al operando proposicional en la cuantificación, de manera que la fórmula resultante ya no corresponde a un esquema proposicional sino a un esquema de cuantificación. Las equivalencias para el proceso de prenexación son:

Prenexación de cuantificadores: si x no figura libre en A entonces,

$$A \wedge \forall x B \equiv \forall x (A \wedge B) \quad (3.36)$$

$$A \vee \forall x B \equiv \forall x (A \vee B) \quad (3.37)$$

$$A \wedge \exists x B \equiv \exists x (A \wedge B) \quad (3.38)$$

$$A \vee \exists x B \equiv \exists x (A \vee B) \quad (3.39)$$

Prenexación de cuantificadores: si x no figura libre en A entonces,

$$A \rightarrow \forall x B \equiv \forall x (A \rightarrow B) \quad (3.40)$$

$$A \rightarrow \exists x B \equiv \exists x (A \rightarrow B) \quad (3.41)$$

Prenexación de cuantificadores: si x no figura libre en B entonces,

$$\forall x A \rightarrow B \equiv \exists x (A \rightarrow B) \quad (3.42)$$

$$\exists x A \rightarrow B \equiv \forall x (A \rightarrow B) \quad (3.43)$$

Veamos un ejemplo del proceso de prenexación.

Ejemplo 3.19.

- Para cualquier objeto, es azul o Múnchen es la capital de Baviera :

$$\forall x (A(x) \vee C(m, b)) \equiv \forall x A(x) \vee C(m, b).$$

Todos los objetos son azules o Múnchen es la capital de Baviera.

- Hay algo tal que los gorriones son bonitos y ese algo es la capital de Francia:

$$\exists x (\forall y (G(y) \rightarrow B(y)) \wedge C(x, f)) \equiv \forall y (G(y) \rightarrow B(y)) \wedge \exists x C(x, f).$$

Los gorriones son bonitos y algo es la capital de Francia.

Ejercicios

3.4.1.- Si $P(x)$ denota al enunciado $x > 4$, di cuál es el valor de

- (a) $P(0)$ (b) $P(4)$ (c) $P(6)$

3.4.2.- Sea $C(x, y)$ el enunciado x es la capital de y . Di cuál es el valor de verdad de:

- (a) $C(\text{Toluca}, \text{México})$ (b) $C(\text{Grenoble}, \text{Francia})$
 (c) $C(\text{Quito}, \text{Bolivia})$ (d) $C(\text{Cd. Juárez}, \text{Nuevo León})$

3.4.3.- Encuentra el valor de verdad de las siguientes fórmulas si el universo son los números reales \mathbb{R} y los predicados se interpretan como sigue:

- $Q(x)$ x es un número par
- $P(x)$ x es un número primo
- $R(x)$ x es divisible por 6
- $G(x)$ x es menor o igual a 5
- $L(x, y)$ x es menor que y

- (a) $\exists x(R(x) \wedge P(x))$
- (b) $\exists x R(x) \wedge \exists x P(x)$
- (c) $\forall x(P(x) \rightarrow \neg Q(x))$
- (d) $\forall x(R(x) \rightarrow \exists y(L(x, y) \wedge R(y)))$
- (e) $\forall x \exists y(L(x, y) \wedge L(y, x))$
- (f) $\exists x P(x) \rightarrow \exists x(P(x) \wedge R(x))$

3.4.4.- Da un micromundo de triángulos, cuadrados y círculos donde todas las fórmulas lógicas que siguen sean verdaderas:

- $\exists x \exists y \exists z \left(T(x) \wedge C(y) \wedge S(z) \right. \\ \wedge ((G(x) \wedge G(y) \wedge G(z)) \\ \vee (M(x) \wedge M(y) \wedge M(z)) \\ \vee (P(x) \wedge P(y) \wedge P(z))) \\ \left. \wedge R(x, y) \wedge R(y, z) \right)$.
- $\exists x \exists y (C(x) \wedge P(x) \wedge C(y) \wedge M(y) \wedge E(x, y))$.
- $\exists x \exists y \exists z (T(x) \wedge P(x) \wedge T(y) \wedge M(y) \wedge T(z) \wedge G(z))$.

3.4.5.- Para cada una de las siguientes fórmulas, da un micromundo de figuras donde estas fórmulas sean verdaderas o, en su defecto, justificar por qué no existe tal mundo.

- (a) $\forall x \exists y (T(x) \rightarrow S(y) \wedge O(x, y))$.
- (b) $\forall x \exists y (C(x) \wedge (T(y) \vee S(y)) \wedge (Co(x, y)))$.
- (c) $\exists x \exists y (R(x, y) \vee Co(x, y))$.
- (d) $\exists x \exists y \exists z (C(x) \wedge M(x) \wedge N(z, x) \wedge O(y, x))$.

$$(e) \forall x \exists y N(y, x) \wedge \exists z C(z).$$

3.4.6.- Para cada fórmula da dos micromundos de figuras, uno donde la fórmula sea verdadera y otro donde sea falsa.

$$(a) \neg \forall x (C(x) \rightarrow G(x)) \wedge \exists z (P(z) \wedge \neg \exists y (T(y) \wedge O(y, z))).$$

$$(b) \forall x \forall y (T(x) \wedge C(y) \wedge N(x, y) \rightarrow \exists z (S(z) \wedge P(z) \wedge Z(z, x) \wedge Z(y, z))).$$

$$(c) \exists w (S(w) \vee G(w)) \wedge \forall x (T(x) \wedge M(x) \wedge \exists y Z(y, x) \rightarrow \exists z (G(z) \wedge N(z, x))).$$

$$(d) \forall w (G(w) \rightarrow \exists y (P(y) \wedge N(y, w))) \vee \exists x \exists z (T(z) \wedge M(x) \wedge O(z, x)).$$

$$(e) \exists x (T(x) \wedge \forall y (N(y, x) \rightarrow P(y) \vee S(y))) \wedge \forall w (C(w) \rightarrow \exists y (G(y) \wedge E(y, w))).$$

3.4.7.- Da un micromundo de cubos donde las siguientes fórmulas sean verdaderas al mismo tiempo.

- $\exists x \exists y \exists z (Az(x) \wedge L(x) \wedge A(y) \wedge L(y) \wedge R(z) \wedge L(z)).$
- $\exists x \exists y \exists z (Az(x) \wedge S(x, p) \wedge A(y) \wedge S(y, p) \wedge R(z) \wedge S(z, p)).$
- $\exists x \exists y (Az(x) \wedge A(y) \wedge S(x, y)) \wedge \exists x \exists y (R(x) \wedge Az(y) \wedge S(x, y)).$

3.4.8.- Considera las siguientes fórmulas e interpretaciones para los predicados:

- $F(x)$ x está fuera de servicio
- $O(x)$ x está ocupada
- $P(x)$ x se ha perdido
- $C(x)$ x está en la cola
- $I(x)$ x es impresora
- $T(x)$ x es trabajo

Construye micromundos de impresoras y trabajos que hagan verdaderas a las fórmulas. La descripción de un micromundo puede ser mediante constantes y tablas de verdad para predicados.

$$(a) \exists x (I(x) \wedge F(x) \wedge O(x)) \rightarrow \exists y (T(y) \wedge P(y)).$$

$$(b) \forall x (I(x) \rightarrow O(x)) \rightarrow \exists y (T(y) \wedge C(y)).$$

$$(c) \exists y (T(y) \wedge C(y) \wedge P(y)) \rightarrow \exists y (I(y) \wedge F(y)).$$

$$(d) \forall x (I(x) \rightarrow O(x)) \wedge \forall y (T(y) \rightarrow C(y)) \rightarrow \exists z (T(z) \wedge P(z)).$$

3.4.9.- Da las negaciones de las siguientes cuantificaciones de manera que el símbolo de negación sólo afecte a predicados. Por ejemplo, la negación de $\forall x (P(x) \wedge Q(x))$ es $\exists x (\neg P(x) \vee \neg Q(x))$, donde puedes notar que no hay negación frente al cuantificador ni frente a una fórmula que consista de más de un predicado.

- (a) $\forall x(x^2 > x)$.
- (b) $\exists x(x^2 = x)$.
- (c) $\forall x(P(x) \rightarrow Q(x))$.
- (d) $\forall x(x^3 < x \rightarrow x < 0)$.
- (e) $\exists x(P(x) \wedge Q(x) \rightarrow R(x))$.

3.4.10.- Para los siguientes enunciados, di cuál o cuáles son las negaciones correctas de los predicados:

- (a) *A todo el mundo le gusta el helado.*
 - i. A nadie le gusta el helado
 - ii. A todo mundo le disgusta el helado
 - iii. Alguien no adora el helado
- (b) *Algunas fotografías están viejas y deslavadas.*
 - i. Todas las fotografías ni están viejas ni están deslavadas
 - ii. Algunas fotografías no están viejas o deslavadas
 - iii. Todas las fotografías no son viejas ni deslavadas

3.4.11.- Muestra que $\forall x(P(x) \wedge Q(x))$ y $\forall x(P(x)) \wedge \forall x(Q(x))$ son lógicamente equivalentes.

3.4.12.- Muestra que $\neg \forall x(P(x) \rightarrow Q(x))$ y $\exists x(P(x) \wedge \neg Q(x))$ son lógicamente equivalentes.

3.4.13.- Transforma las siguientes fórmulas mediante equivalencias lógicas, de manera que las negaciones sólo figuren frente a predicados.

- (a) $\forall x \exists y \neg \forall z \exists w (P(x, w) \vee Q(z, y)) \rightarrow \neg \exists v \forall u \neg R(u, v)$
- (b) $\neg \forall x \exists y \neg \forall w \exists z (P(x, y) \vee \neg Q(x) \rightarrow \exists w \neg T(a, w))$
- (c) $\neg \exists x \forall y \neg \exists w \forall z \neg (\neg P(x, y) \wedge Q(x) \rightarrow \forall w T(x, w))$
- (d) $\neg \left(\neg \exists x \forall y (\neg T(y) \wedge R(z, x) \rightarrow G(x, z)) \rightarrow \forall w \neg \exists v P(v, a, w) \right)$
- (e) $\neg \left(\forall x \exists w \neg (\neg P(a, x) \vee R(c, w)) \wedge \exists z \neg \forall y (T(b, z) \wedge \neg Q(y, a)) \right)$

3.4.5. Algunos argumentos correctos

Ya hemos mencionado con anterioridad que nuestro propósito principal para estudiar lógica consiste en obtener métodos formales para mostrar la correctud de argumentos lógicos. Si bien podemos generalizar algunos de los métodos estudiados en la lógica de proposiciones para la lógica de predicados, estos métodos no son infalibles, debido a un importante resultado de la Lógica Matemática, demostrado por Alonzo Church, que nos dice que no puede existir un algoritmo para decidir si un argumento dado es correcto o no. A pesar de este resultado, el problema de analizar un argumento de la lógica de predicados para intentar decidir su correctud sigue siendo de gran importancia en la práctica, puesto que la lógica de predicados es una herramienta de gran importancia para la especificación formal en computación.

Si bien no hay un algoritmo general, la correctud de un argumento puede decidirse en muchos casos mediante métodos sintácticos o semánticos que quedan fuera del alcance de este libro. Sin embargo, dado que el proceso de argumentación es relevante en la práctica tanto en matemáticas como en ciencias de la computación, enunciamos a continuación algunos argumentos correctos de la lógica de predicados, los cuales surgen naturalmente en matemáticas.

- *Generalización Universal*: Sea A una fórmula y x una variable que no figura libre en la argumentación actual. Entonces el siguiente argumento es correcto:

$$\frac{A}{\forall x A}$$

Este argumento permite concluir la validez de la fórmula $\forall x A$ al mostrar la validez de A , cerciorándonos de que x no figura libre en ninguna de las premisas usadas para llegar a A . Esta restricción implica que en la argumentación no se usó ninguna propiedad particular de x , por lo que ésta denota a cualquier individuo posible del universo de discurso, lo cual permite realizar la generalización. Este argumento es indispensable en pruebas por inducción, como se verá en el siguiente capítulo.

- *Instanciación Universal*:

$$\frac{\forall x A}{A[x := t]}$$

La correctud de este argumento es intuitivamente clara: si la fórmula $\forall x A$ se supone verdadera, entonces uno debería poder concluir $A[x := t]$ para cualquier individuo particular del universo de discurso, denotado por el término t . Sin embargo, hay que

tener cuidado, puesto que en A pueden figurar otros cuantificadores y en t otras variables, se corre el peligro de capturar alguna variable de t , que por supuesto estaba libre, mediante algún cuantificador de A causando un problema semántico importante. Es por esta razón que la sustitución en lógica de predicados no es una sustitución textual como la estudiada antes en este libro, sino que debe vigilar no cambiar presencias libres por ligadas y viceversa.

- *Generalización Existencial:*

$$\frac{A[x := t]}{\exists x A}$$

Nuevamente, la correctud de este argumento es intuitivamente clara: si sabemos que un individuo particular t cumple la propiedad A entonces podemos concluir que alguien cumple A , es decir, podemos concluir $\exists x A$.

- *Instanciación Existencial:* Sea A una fórmula y c una constante nueva en la argumentación actual. Entonces el siguiente argumento es correcto:

$$\frac{\exists x A}{A[x := c]}$$

Este argumento permite concluir la validez de A para un individuo particular c del universo de discurso a partir de la verdad de $\exists x A$. La restricción acerca de que c sea una constante nueva se debe al hecho de que no es posible saber cuál individuo particular es el que cumple A a partir de la única información que tenemos, que es $\exists x A$.

3.5. Predicados y tipos

Frecuentemente un dominio de interpretación se compone de diversas clases bien determinadas de objetos, por ejemplo círculos y cuadrados, alumnos y profesores, animales y vegetales. En estos casos, cuando se quiere especificar algo acerca de todos los individuos de cierta clase de objetos del universo, es conveniente hacer explícita su pertenencia a dicha clase particular mediante el uso de predicados llamados *calificadores* o *tipos*, los cuales denotan clases de objetos. Para expresar propiedades de un tipo específico de objeto se usa un juicio universal afirmativo. Por ejemplo, si el universo son los mamíferos y queremos hablar de una propiedad universal $P(x)$ de los felinos, como pudiese ser maullar o ser cuadrúpedo, la especificación $\forall x P(x)$ no da la suficiente información y es preferible usar un tipo $F(x)$ para felinos, con lo que la especificación sería $\forall x (F(x) \rightarrow P(x))$. Similarmente, si la especificación es existencial, utilizamos un juicio existencial afirmativo, como en *algunos felinos beben leche* que se formaliza con $\exists x (F(x) \wedge BL(x))$.

El uso de tipos permite restringir o dirigir el rango de valores de una variable dada mediante el uso de un juicio afirmativo. Sin embargo, dado que su uso resulta muy frecuente y útil, es conveniente introducir una notación especial para tipos como sigue:

$$\begin{aligned} \forall x:A. P(x) & \text{ en lugar de } \forall x(A(x) \rightarrow P(x)) \\ \exists x:A. P(x) & \text{ en lugar de } \exists x(A(x) \wedge P(x)) \end{aligned}$$

A esta notación la denominamos de *tipos abreviados*. Por ejemplo, si el universo son los números reales, el enunciado *para todo número real existe un natural mayor que él* puede expresarse como sigue:

- Sin tipos: $\forall x \exists y (x < y)$ (inconveniente pues no da suficiente información).
- Con juicios afirmativos: $\forall x (R(x) \rightarrow \exists y (N(y) \wedge x < y))$.
- Con tipos abreviados: $\forall x : R. \exists y : N. x < y$.

Cuando un lenguaje tiene reglas sintácticas que manejen los tipos de las variables decimos que tenemos un lenguaje *fuertemente tipificado o tipado*. Entre los lenguajes de programación que son fuertemente tipificados tenemos a Pascal, C++, Java, C#, C y Haskell. En general, un lenguaje fuertemente tipificado nos da reglas muy estrictas respecto a cómo podemos combinar distintos tipos en una misma expresión. Otros lenguajes, como Lisp, Prolog y Scheme, son lenguajes que no observan el concepto de tipo, de manera que las variables pueden tener distintos tipos durante la ejecución, dependiendo del estado de las mismas.

El concepto de tipo que se presenta corresponde únicamente a los tipos primitivos de un lenguaje. La lógica de predicados que estudiamos aquí es una lógica sin tipos en el sentido de que los símbolos funcionales y de predicado, que se interpretan como operadores o relaciones, no tienen tipos explícitos. En nuestro caso, el uso de tipos es un mecanismo de ayuda y simplificación en la escritura de ciertas especificaciones.

Terminamos esta sección con otros ejemplos.

Ejemplo 3.20. Vamos a especificar el tipo de los números naturales $N(x)$ con sus operaciones más comunes.

- | | |
|---|---|
| • El cero es un número natural: | $N(0)$ o bien $0:N$ |
| • El sucesor de un natural es un natural: | $\forall x:N. (N(s(x)))$ o bien $\forall x:N. (s(x):N)$ |
| • La suma de dos naturales es un natural: | $\forall x:N. \forall y:N. (N(x + y))$ |
| • El producto de dos naturales es un natural: | $\forall x:N. \forall y:N. (N(x \cdot y))$ |
| • El sucesor es una función inyectiva: | $\forall x:N. \forall y:N. (s(x) = s(y) \rightarrow x = y)$ |
| • Hay un natural menor o igual que todos los naturales. | $\exists y:N. \forall x:N. y \leq x$ |
-

Por último un ejemplo más cercano a las especificaciones usuales en computación.

Ejemplo 3.21. Se desean especificar propiedades de un sistema de archivos de computadora. Consideremos que el universo consta de archivos y directorios para lo cual definimos los tipos $A(x)$ y $D(x)$ (o simplemente A y D), con el predicado $C(x, y)$ como *el objeto x está contenido en el objeto y* y la función $n(x)$ que devuelve el nombre del objeto.

- Ningún directorio se contiene a sí mismo:

$$\forall x:D. \neg C(x, x)$$

- Si un directorio está contenido en otro, entonces el segundo no puede estar contenido en el primero (es decir, no hay directorios cíclicos):

$$\forall x:D. \forall y:D. (C(x, y) \rightarrow \neg C(y, x))$$

- Existe un directorio que no está contenido en ningún otro directorio (el directorio raíz):

$$\exists x:D. \forall y:D. \neg C(x, y)$$

- Existe un directorio vacío:

$$\exists x:D. \forall y. \neg C(y, x)$$

- Todo archivo está contenido en algún directorio:

$$\forall x:A. \exists y:D. C(x, y)$$

- Si dos archivos están en el mismo directorio entonces deben tener nombres distintos.

$$\forall x:A. \forall y:A. (\exists z:D. (C(x, z) \wedge C(y, z)) \rightarrow n(x) \neq n(y))$$

Ejercicios

3.5.1.- Formaliza las siguientes especificaciones acerca de un tipo $A(x)$ y el tipo de listas de elementos de A , denotado $L(x)$. Debes agregar cualquier predicado, función o constante necesaria.

- La lista vacía es una lista de elementos de A .
- La operación de agregar un elemento de A al inicio de una lista dada es nuevamente una lista.

- (c) La concatenación de dos listas es nuevamente una lista.
- (d) La cabeza de una lista es un elemento de A .
- (e) La cola de una lista es nuevamente una lista.
- (f) La longitud de una lista es un número natural.

3.5.2.- Formaliza las siguientes especificaciones acerca de un tipo $A(x)$ y el tipo de pilas de elementos de A , denotado $P(x)$. Debes agregar cualquier predicado, función o constante necesaria.

- (a) Hay una pila vacía.
- (b) La operación de agregar un elemento de A al tope de una pila es una pila.
- (c) El tope de la pila es un elemento de A .
- (d) La operación de eliminar el elemento en el tope de la pila devuelve una pila.

Parte II

Inducción y recursión

Inducción y recursión | 4

4.1. Introducción

Existen muchos universos o dominios que contienen un número ilimitado de elementos que sin embargo pueden ser contados. Por ejemplo, el universo de números naturales, el dominio de expresiones lógicas (tomadas con las variables proposicionales de un alfabeto) y el dominio de programas escritos en ciertos lenguajes de programación. Estos universos se conocen como conjuntos infinitos numerables y son de gran utilidad en Ciencias de la Computación y Matemáticas Discretas. *Numerable* en este caso significa que se pueden contar en el sentido de que dado un elemento del conjunto, es posible determinar cuál es el elemento siguiente. Sin embargo, por ser infinitos, no es posible describirlos elemento por elemento pues nunca terminaríamos, ni tampoco podemos probar alguna propiedad acerca de ellos tratando de mostrarla para cada elemento particular. En este capítulo tratamos dos técnicas muy relacionadas entre sí, la inducción y la recursión, las cuales sirven para probar y definir propiedades sobre dominios infinitos numerables.

Iniciamos el capítulo definiendo de manera formal a los números naturales, mostrando algunas definiciones recursivas de funciones sobre los mismos y discutiendo el llamado método de inducción matemática y algunas de sus variantes. Posteriormente nos ocuparemos de las definiciones recursivas de conjuntos y funciones en un ámbito más general. Estas definiciones recursivas son generalizaciones de las utilizadas en números naturales a cual-

quier dominio infinito numerable y que además esté bien fundado¹. En la última sección nos ocupamos en generalizar el principio de inducción matemática mediante la llamada inducción estructural en algunas estructuras de datos muy necesarias en programación como son árboles y cadenas o listas finitas.

4.2. Los números naturales

El conjunto de números naturales² $\mathbb{N} = \{0, 1, 2, \dots\}$ es quizás el ejemplo más sencillo de un conjunto infinito numerable, pero siendo infinito, ¿cómo podemos justificar su construcción y manejo en computación?

Empecemos con su construcción. En la vida diaria utilizamos los símbolos $0, \dots, 9$ para representar los primeros diez números naturales, llamados dígitos, mientras que los siguientes números se definen a partir de los dígitos mediante ciertas reglas. Formalmente sólo utilizaremos el dígito 0 ya que los demás números se construirán utilizando la función sucesor. El sucesor de un número n , escrito $s(n)$, es simplemente el número que le sigue a n en la sucesión de números naturales o, equivalentemente, $s(n) = n + 1$, pero como aún no definimos la suma evitaremos su uso. Obsérvese que la función sucesor es general y no depende del dominio de los números naturales; por ejemplo los días y meses tienen sucesor.

La definición de números naturales será nuestro primer ejemplo de definición recursiva.

- 0 es un número natural.
- Si n es un número natural, entonces $s(n)$ es un número natural.
- Éstos y sólo éstos.

Esta definición es recursiva pues en la segunda cláusula se está usando a n , que suponemos es un natural, para poder concluir que $s(n)$ también lo es, es decir, estamos usando lo definido en la misma definición; en la siguiente sección trataremos con detalle este tipo de definiciones. La tercera cláusula puede parecer extraña y con frecuencia se omite en las definiciones. Sin embargo es necesaria para garantizar que un objeto es un número natural únicamente si fue construido usando las cláusulas anteriores. Esto es necesario para que funcionen los principios de inducción.

Según la definición anterior el conjunto de números naturales es

$$\mathbb{N} = \{0, s(0), s(s(0)), \dots\}$$

De esta manera hemos construido un conjunto infinito en el sentido de que siempre podremos construir cualquier número de sus elementos y en particular cualquier elemento.

¹dominio bien fundado se refiere, en términos muy generales, a que podemos encontrar un primer elemento

²La inclusión del 0 en los naturales no es aceptada universalmente, especialmente por matemáticos; sin embargo, aquellos académicos que cultivan la investigación en lógica, postulan que $0 \in \mathbb{N}$.

Diferencia entre sintaxis y semántica

Estructuralmente es claro que el conjunto de números naturales recién definido es infinito; sin embargo, si le damos cierto significado a la función sucesor pudiera darse el caso de que los elementos $s(s(\dots s(n) \dots))$ no sean todos distintos. Por ejemplo, si hablamos de los días de la semana,

$$s(s(s(s(s(s(s(lunes))))))) = lunes.$$

Para indicar que el conjunto es infinito es necesario postular dos propiedades más que garanticen que todos los naturales son distintos.

- $\forall n(s(n) \neq 0).$
- $\forall n \forall m(s(n) = s(m) \rightarrow n = m).$

Estas dos propiedades aseguran que el 0 no es sucesor de nadie y que la función sucesor es inyectiva.

A continuación nos gustaría definir las operaciones básicas suma y producto; esto se hará nuevamente usando recursión. Para la suma tenemos la siguiente definición:

- $\forall n(n + 0 = n).$
- $\forall n \forall m(m + s(n) = s(m + n)).$

La importancia de una definición recursiva es que podemos extraer de ella un programa para calcular dicha función; veamos un ejemplo sencillo:

$$\begin{aligned} 3 + 2 &= s(s(s(0))) + s(s(0)) \\ &= s(s(s(s(0))) + s(0)) \\ &= s(s(s(s(s(0))) + 0)) \\ &= s(s(s(s(s(s(0)))))) \\ &= 5 \end{aligned}$$

Finalmente, el producto de dos naturales se define recursivamente como sigue:

- $\forall n(n \times 0 = 0).$
- $\forall n \forall m(n \times s(m) = n \times m + n).$

Más adelante daremos más ejemplos de funciones definidas recursivamente.

4.2.1. Axiomas de Peano

Las fórmulas lógicas definidas anteriormente constituyen los llamados *axiomas de Peano*; éstos fueron propuestos por el matemático italiano Giuseppe Peano en 1889 y cons-

tituyen una definición abstracta del conjunto de los números naturales. A continuación los resumimos.

• 0 es un número natural. (P-1)

• Si n es un número natural entonces $s(n)$ es un número natural. (P-2)

• $\forall n (s(n) \neq 0)$. (P-3)

• $\forall m \forall n ((s(n) = s(m)) \rightarrow (n = m))$. (P-4)

También contamos, en este mismo formato, con las definiciones recursivas de las operaciones de suma y producto de los números naturales recién discutidas y que recapitulamos a continuación:

• $\forall m (m + 0 = m)$. (D-1)

• $\forall m \forall n (m + s(n) = s(m + n))$. (D-2)

• $\forall n (n \times 0 = 0)$. (D-3)

• $\forall m \forall n (m \times s(n) = m \times n + m)$. (D-4)

El último axioma de Peano es el llamado *axioma de inducción* y nos dice que para cualquier predicado P la siguiente expresión es válida:

$$P(0) \wedge \forall n (P(n) \rightarrow P(s(n))) \rightarrow \forall n (P(n)). \quad (\text{P-5})$$

Esta expresión formaliza el principio de inducción para números naturales. Este principio es muy conocido y de gran importancia en matemáticas discretas y ciencias de la computación y, en general, en todas las matemáticas. A continuación discutimos su validez y desarrollamos algunos ejemplos de su uso.

4.3. Inducción en los números naturales

Dada una propiedad P acerca de números naturales, tal que $P(n)$ ha sido probada para un natural cualquiera n , es fácil cerciorarse de la validez de la propiedad para el siguiente número, es decir la validez de $P(s(n))$; si además podemos probar $P(0)$, entonces el axioma (P-5) nos permite concluir que nuestra propiedad es válida para todos los números naturales. Esto se justifica al existir para cada número natural n_0 una derivación de $P(n_0)$ construida como sigue, usando $1, 2, 3, \dots$ en lugar de $s(0), s(s(0)), \dots$:

- | | | |
|-------|---------------------------------------|------------------------------|
| 1. | $P(0)$ | Hipótesis. |
| 2. | $\forall n(P(n) \rightarrow P(s(n)))$ | Hipótesis. |
| 3. | $P(0) \rightarrow P(1)$ | Instanciación $n := 0$ en 2. |
| 4. | $P(1)$ | Modus Ponens 1, 3. |
| 5. | $P(1) \rightarrow P(2)$ | Instanciación $n := 1$ en 2. |
| 6. | $P(2)$ | Modus Ponens 4, 5. |
| 7. | $P(2) \rightarrow P(3)$ | Instanciación $n := 2$ en 2. |
| 8. | $P(3)$ | Modus Ponens 6, 7. |
| | \vdots | |
| k . | $P(n_0)$ | |

Estas derivaciones generan la siguiente regla de inferencia, la cual también se deriva del axioma (P-5):

$$\frac{\begin{array}{l} P(0) \\ \forall n(P(n) \rightarrow P(s(n))) \end{array}}{\forall n(P(n))}$$

donde P es un predicado acerca de números naturales.

Veamos algunos ejemplos de su uso:

Ejemplo 4.1. Mostrar que 0 es identidad por la izquierda de la suma; esto es

$$\forall n(0 + n = n).$$

Demostración.

Base: Demostrar $P(0)$: $(0 + 0) = 0$.

Esto se cumple por (D-1).

Hipótesis de inducción: Suponemos $P(n)$: $0 + n = n$.

Paso inductivo: Demostrar $P(s(n))$: $0 + s(n) = s(n)$.

$$\begin{aligned} (0 + s(n)) &= s(0 + n) && \text{(D-2)} \\ &= s(n) && \text{(hipótesis de inducción)} \end{aligned}$$

Ejemplo 4.2. Mostrar que la suma es conmutativa, esto es:

$$\forall m(\forall n(n + m = m + n)).$$

Demostración. Demostrar $n + m = m + n$.

Haremos inducción sobre m (no se puede hacer sobre ambas variables).

Base: Demostrar $P(0)$: $0 + n = n + 0$.

$$\begin{aligned} 0 + n &= n && \text{(ejemplo (4.1))} \\ &= n + 0 && \text{(D-1)} \end{aligned}$$

Hipótesis de inducción: Suponemos $P(m)$: $m + n = n + m$.

Paso inductivo: Demostrar $P(s(m))$: $s(m) + n = n + s(m)$.

Tomando el lado derecho:

$$\begin{aligned} n + s(m) &= s(n + m) && \text{(D-2)} \\ &= s(m + n) && \text{(hipótesis de inducción)} \end{aligned}$$

Quisiéramos que el siguiente paso fuera

$$s(m + n) = s(m) + n.$$

Pero esto no es consecuencia de los axiomas ni de resultados anteriores. Por lo tanto, lo tenemos que demostrar. Lo haremos usando inducción natural sobre n ahora.

Base: Demostrar $P(0)$: $s(m + 0) = s(0 + m) = 0 + s(m) = s(m) + 0$.

Esto se cumple porque ambos lados son iguales a $s(m)$.

Hipótesis de inducción: Suponemos $P(n)$: $s(m + n) = s(m) + n$.

Paso inductivo: Demostrar $P(s(n))$: $s(m + s(n)) = s(m) + s(n)$.

$$\begin{aligned} s(m + s(n)) &= s(s(m + n)) && \text{(D-2)} \\ &= s(s(m) + n) && \text{(hipótesis de inducción)} \\ &= s(m) + s(n) && \text{(D-2)} \end{aligned}$$

Generalización Universal: $\forall n(s(m + s(n)) = s(m) + s(n))$

Generalización universal sobre m : $\forall m(\forall n(m + n = n + m))$.

Ejemplo 4.3. Sea $H_n = 0$ para $n = 0$, y $H_{n+1} = 1 + 2H_n$ para $n > 0$. Demostrar que $H_n = 2^n - 1$.

Demostración. Verificamos primero para la base, que en este caso es 0:

Base: Demostrar, usando la definición dada, $P(0)$: $H_0 = 0 = 2^0 - 1$.

$$\begin{aligned} H_0 &= 2^0 - 1 && \text{(por la definición de } H_n \text{ con } n = 0 \text{)} \\ &= 1 - 1 && \text{(por aritmética)} \\ &= 0 && \checkmark \end{aligned}$$

Hipótesis de inducción: Suponemos $P(n)$: $H_n = 1 + 2H_{n-1} = 2^n - 1$.

Paso inductivo: Verificar que $H_{n+1} = 2^{n+1} - 1$.

$$\begin{aligned}
 H_{n+1} &= 1 + 2H_n && \text{(definición de } H_{n+1}) \\
 &= 1 + 2(2^n - 1) && \text{(hipótesis de inducción)} \\
 &= 1 + 2 \cdot 2^n - 2 \cdot 1 && \text{(aritmética)} \\
 &= 1 + 2^{n+1} - 2 && \text{(aritmética)} \\
 &= 2^{n+1} - 1 && \text{(aritmética)} \quad \checkmark
 \end{aligned}$$

Ejemplo 4.4. Muestra que para toda n , $2(n+2) \leq (n+2)^2$.

Demostración.

Base: Para $n = 0$,

$$2(0+2) = 2+2 = 4 \leq 4 = 2^2 = (0+2)^2$$

Hipótesis de inducción: Suponemos $P(n)$: $2(n+2) \leq (n+2)^2$.

Paso inductivo: Corroborar que se cumple $P(n+1)$:

$$\begin{aligned}
 2((n+1)+2) &= 2n+2+4 && \text{(definición y aritmética)} \\
 &= 2(n+2)+2 && \text{(aritmética)} \\
 &< (n+2)^2+2 && \text{(hipótesis de inducción)} \\
 &= n^2+4n+4+2 && \text{(aritmética)}
 \end{aligned}$$

Buscamos acercarnos al lado derecho: $(n+3)^2 = n^2+6n+9$

$$\begin{aligned}
 &< n^2+4n+6+2n+3 && n > 0 \text{ (por lo que al agregarlo se mantiene la desigualdad)} \\
 &= (n+3)^2 && \text{(aritmética)} \\
 &= ((n+1)+2)^2 && \checkmark
 \end{aligned}$$

Ejemplo 4.5. Demuestra que $n^3 + 2n$ es divisible por 3.

Demostración.

Que $n^3 + 2n$ sea divisible entre 3 quiere decir que se puede expresar como $n^3 + 2n = 3 \cdot k$ para algún entero k .

Base: Para $n = 0$, $0^3 + 2n = 0 + 0 = 0 = n$, por lo que $n^3 + 2n$ es divisible por 3.

Hipótesis de inducción: Suponemos $P(n)$: $n^3 + 2n = 3k$ para alguna k .

Paso inductivo: Tomemos $n+1$ y veamos cómo se expresa $(n+1)^3 + 2(n+1)$.

$$\begin{aligned}
 (n+1)^3 + 2(n+1) &= n^3 + 3n^2 + 3n + 1 + 2n + 2 && \text{(álgebra)} \\
 &= (n^3 + 2n) + 3n^2 + 3n + 3 && \text{(asociatividad y} \\
 & && \text{conmutatividad)} \\
 &= 3k + 3(n^2 + n + 1) && \text{(hipótesis de inducción)}
 \end{aligned}$$

$$\text{sea } k' = n^2 + n + 1$$

$$= 3(k + k') \quad \text{(factorización)}$$

Conclusión: De esto, $\forall n(n^3 + 2n \text{ es múltiplo de } 3)$.

4.3.1. Cambio de la base de la inducción

En algunos casos la base de la inducción no es necesariamente el cero o el uno; esto no es una falla en el método de inducción, sino que la propiedad utilizada es válida a partir de cierto número n_0 , lo cual genera un principio similar, presentado aquí como regla de inferencia:

$$\frac{P(n_0) \quad \forall n(n \geq n_0 \rightarrow P(n) \rightarrow P(s(n)))}{\forall n(n \geq n_0 \rightarrow P(n))}$$

Ejemplo 4.6. Mostrar que $2^n < n!$, para $n \geq 4$.

Demostración.

Base: $P(4)$: $2^4 = 16 < 24 = 4!$.

Hipótesis de inducción: Suponer $P(n)$: $2^n < n!$.

Paso inductivo: Demostrar $P(n+1)$: $2^{n+1} < (n+1)!$.

$$\begin{aligned}
 2^{n+1} &= 2 \times 2^n && \text{(aritmética)} \\
 &< 2 \times n! && \text{(hipótesis de inducción)} \\
 &< (n+1) \times n! && 2 < n+1, \text{ (pues } n \geq 4) \\
 &= (n+1)! && \text{(definición de } (n+1)! \text{)}
 \end{aligned}$$

Ejemplo 4.7. Mostrar que cualquier cantidad mayor a 3 pesos puede pagarse usando únicamente monedas de 2 y 5 pesos.

Demostración.

Base: $P(4)$: $4 = 2 \cdot 2$ de manera que \$4 puede pagarse con dos monedas de \$2.

Hipótesis de inducción: $P(n)$: Suponemos que \$ n pueden pagarse con monedas de \$2 y \$5.

Paso inductivo: $P(n+1)$: Demostrar que \$($n+1$) pueden pagarse con monedas de \$2 y \$5.

Por la hipótesis de inducción tenemos que $\$n = k \cdot 2 + m \cdot 5$. Es decir, \$ n se pagan con k monedas de \$2 y m monedas de \$5. Tenemos dos casos:

- $m = 0$. Es decir, \$ n se pagaron solamente con monedas de \$2. En este caso,

$$n + 1 = k \cdot 2 + 1 = (k - 2) \cdot 2 + 2 \cdot 2 + 1 = (k - 2) \cdot 2 + 5.$$

de donde si \$ n se pagaron con k monedas de \$2, tenemos que \$($n+1$) se pagan con $k - 2$ monedas de \$2 y una moneda de \$5. Obsérvese que estamos separando dos monedas de \$2 para completar \$5; esto puede hacerse debido a que $k \geq 2$ ya que $n \geq 4$.

- $m > 0$. Es decir, \$ n se pagaron con al menos una moneda de \$5.

$$n + 1 = k \cdot 2 + m \cdot 5 + 1 = k \cdot 2 + (m - 1) \cdot 5 + 5 + 1 = (k + 3) \cdot 2 + (m - 1) \cdot 5$$

de donde \$($n+1$) se pagan con $k + 3$ monedas de \$2 y $m - 1$ monedas de \$5. Obsérvese que separamos una moneda de \$5 para obtener \$6 que se pagan con tres monedas de \$2; esto puede hacerse pues $m \geq 1$.

De los ejemplos anteriores podemos obtener un esquema general para una prueba por inducción:

1. Enunciar el uso del principio de inducción. De esta manera el lector comprenderá de qué tipo de prueba se trata.
2. Definir un predicado apropiado $P(n)$, de manera que la meta a probar sea $\forall n P(n)$. Con frecuencia este predicado puede extraerse de la afirmación en español que se desea probar.
3. Mostrar que la base de la inducción $P(0)$ (o $P(n_0)$) es cierta.
4. Enunciar la hipótesis de inducción $P(n)$.
5. Probar la implicación $P(n) \rightarrow P(n+1)$; esto se conoce como paso inductivo.
6. Invocar el principio de inducción y concluir que $\forall n P(n)$.

Cualquier prueba por inducción debe tener todos estos pasos y en este orden.

4.3.2. Inducción completa

Si pensamos en una prueba por inducción de acuerdo al principio original (P-5) y a la derivación lógica dada en la página 167 para justificar el método, al probar $P(m)$ para un número cualquiera m tuvimos que probar antes $P(0), P(1), \dots, P(m-1)$, es decir, la propiedad P tuvo que verificarse para todos los números anteriores a m . Esta información podría ser útil y necesaria para probar $P(m+1)$, ya que en algunos casos no basta con la información inmediata anterior $P(m)$. Esta observación da lugar al *principio de inducción fuerte o completa* que enunciamos aquí como regla de inferencia.

$$\frac{\forall n \left(\forall m (m < n \rightarrow P(m)) \rightarrow P(n) \right)}{\forall n (P(n))}$$

Obsérvese que en este caso no hay una base explícita de la inducción. Si instanciamos $n = 0$ entonces la premisa de la regla resulta equivalente a $P(0)$, puesto que la fórmula $\forall m (m < 0 \rightarrow P(m))$ es cierta al tratarse de una implicación con antecedente falso ($m < 0$) con $m \in \mathbb{N}$. Al probar el paso inductivo para $n = 0$ no hay hipótesis disponible para usarse, por lo que $P(0)$ debe ser probado como en casos anteriores. Sin embargo, esto no es necesario en la mayoría de los casos.

Este principio permite partir la prueba del paso inductivo en dos o más casos más pequeños, cualesquiera que éstos sean.

Ejemplo 4.8. Sea d el cero del operador \circ , es decir $\forall x (x \circ d = d \circ x = d)$. Mostrar que cualquier expresión que contenga una o más presencias de d debe ser igual a d .

Sea $P(n)$ la proposición de que cualquier expresión con n presencias de \circ y al menos una presencia de d es igual a d .

Base: Veamos las posibles expresiones con una presencia de \circ y al menos una presencia de d :

$$(a) \quad x \circ d \qquad (b) \quad d \circ x.$$

Por la definición del operador \circ tenemos

$$\forall x (x \circ d = d \circ x = d).$$

por lo que $P(1)$ se cumple.

Hipótesis de inducción: Supongamos $P(m)$ para $m < n$. Es decir, cualquier expresión con $m < n$ presencias de \circ y al menos una presencia de d es igual a d .

Paso inductivo: Sea x una expresión con $n > 0$ operadores que contiene al menos una presencia de d ; entonces $x = x_1 \circ x_2$ donde x_1, x_2 son expresiones con menos de n operadores \circ y alguna de x_1, x_2 contiene una presencia de d , digamos que es x_1 . En tal caso, por la hipótesis de inducción se tiene $x_1 = d$, de donde tenemos $x = x_1 \circ x_2 = d \circ x_2$. Como $d \circ x$ tiene menos de n presencias de \circ (eliminamos todas las presencias de \circ en x_1) presencia de d , tenemos que $d \circ x_2 = d$, lo cual completa el paso inductivo. Obsérvese que la prueba es totalmente análoga si es x_2 quien contiene una presencia de d .

Ejemplo 4.9. Demostrar que cualquier $n \geq 2$ es primo o es producto de primos.

Sea $P(n)$ la proposición: n es primo o producto de primos. Queremos probar que $\forall n(n \geq 2 \rightarrow P(n))$.

Base: $P(2)$: Para $n = 2$ tenemos que 2 es primo, por lo que se cumple $P(2)$.

Hipótesis de inducción: Supongamos $P(m)$ para $m < n$. Es decir, cualquier número $m < n$ es primo o producto de primos.

Paso inductivo: Si n es primo hemos terminado. Si no lo es, n se puede escribir como $n = m \cdot q$ con $1 < m, q < n$ y por la hipótesis de inducción, ambos, m y q , son primos o producto de primos, de donde $n = m \cdot q$ también lo es.

Obsérvese que en este ejemplo se combinan la inducción completa y el cambio de base al iniciar en $n = 2$.

Ejercicios

4.3.1.- Demuestra, usando las definiciones de suma y producto dadas al inicio de esta sección, que $s(0)$ es la identidad para la multiplicación; esto es

$$\forall m(m \times s(0) = m)$$

4.3.2.- Demuestra las siguientes propiedades de la suma y el producto:

- Asociatividad de la suma: $\forall m \forall n \forall k(m + (n + k) = (m + n) + k)$.
- Asociatividad del producto: $\forall m \forall n \forall k(m \times (n \times k) = (m \times n) \times k)$.
- Neutro izquierdo del producto: $\forall n(0 \times n = 0)$.
- Conmutatividad del producto: $\forall m \forall n(m \times n = n \times m)$.

4.3.3.- Demuestra, usando las definiciones de suma y producto dadas al inicio de esta sección, que

$$\forall m \forall n (s(m) \times s(n) = m \times n + s(m) + n)$$

4.3.4.- Demuestra que para toda n ,

$$\sum_{k=1}^n k = \frac{n(n+1)}{2}.$$

4.3.5.- Demuestra que para toda n ,

$$\sum_{k=1}^n k^3 = \left(\frac{n(n+1)}{2} \right)^2.$$

4.3.6.- Demuestra que para toda n ,

$$5+8+11+\cdots+(3n+2) = \frac{1}{2} (3n^2 + 7n).$$

4.3.7.- Usa inducción así como las leyes de conmutatividad y asociatividad para demostrar que

$$a_1 + (a_2 + (a_3 + \cdots + (a_{n-1} + a_n) \cdots)) = a_n + (a_{n-1} + (\cdots + (a_2 + a_1) \cdots))$$

4.3.8.- Sea $n > 3$ un número natural. Sea m el entero mayor que es menor o igual que $(n+2)/2$ (o sea, $m = \lfloor (n+2)/2 \rfloor$). Veamos una pequeña tabla con los valores de n y m :

n	(n + 2)/2	m
2	2	2
5	3.5	3
6	4	4
7	4.5	4

Entonces, dados más de m enteros en el conjunto $\{1, 2, \dots, n\}$, tres de los enteros en este conjunto tienen la propiedad de que alguno de los tres es la suma de los otros dos.

4.3.9.- Demuestra que para toda $n \geq 0$,

$$\sum_{k=0}^n 9 \cdot 10^k = 10^{n+1} - 1.$$

4.3.10.- Usa inducción matemática para demostrar que para todo entero n , $n < 2^n$.

4.3.11.- Demuestra que para todo entero positivo n existe un entero positivo con n dígitos que es divisible entre 5^n y tal que todos sus dígitos son impares. Que un entero p sea

divisible entre otro entero q , denotado $p \mid q$, quiere decir que al dividir p entre q , el residuo es 0, o dicho de otra manera:

$$\text{Dados } p, q \in \mathbb{Z}^+, p \mid q \rightarrow \exists m \in \mathbb{Z}^+ \text{ tal que } p = q \cdot m$$

Veamos algunos ejemplos de la proposición:

n	Entero con n dígitos	$p = q \cdot m$
$P(1)$	5	$5 = 5^1(1)$
$P(2)$	75	$75 = 5^2(3) = 25 \cdot 3$
$P(3)$	375	$375 = 5^3(3) = 125 \cdot 3$
$P(4)$	9375	$5^4(15) = 625 \cdot 15$

4.3.12.- Demuestra que para todo entero $n \geq 0$ y $z \neq 1$,
$$\sum_{k=0}^n z^k = \frac{z^{n+1} - 1}{z - 1}$$

4.3.13.- Demostrar que para todo entero $n > 6$, $3^n < n!$.

4.3.14.- Para todo natural n ,
$$\sum_{k=1}^n k(k!) = (n+1)! - 1$$

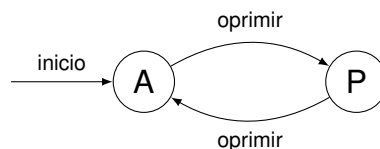
4.3.15.- Los autómatas finitos son un modelo muy útil para dispositivos en software o hardware. Un autómata finito es un dispositivo que se puede encontrar, en un momento dado, en un número finito de *estados*. El objetivo de los estados es *recordar* una porción relevante de la historia del sistema. Como sólo hay un número finito de estados, la historia completa no puede ser registrada, por lo que se deberá diseñar con cuidado para recordar los aspectos relevantes.

En cada estado, el autómata recibe posibles señales, que lo pueden hacer cambiar de estado. El autómata inicia siempre en un estado designado como *inicial*, y dependiendo del estado en el que está, puede emitir una señal.

Podemos modelar un apagador muy sencillo con un autómata finito. El autómata tiene dos estados, el de *apagado* y el de *prendido*, que es lo que el autómata tiene que recordar. Cuando se oprime el apagador, dependiendo en cuál de los dos estados esté, va a pasar al otro: Si está en *apagado* pasa a *prendido* y si está en *prendido* pasa a *apagado*. El estado inicial es *apagado*. Podemos modelar el autómata con lo que se conoce como un *diagrama de transiciones*, como se muestra en la figura 4.1. Como se puede ver en esta figura, los estados están representados por círculos, mientras que el resultado de oprimir el apagador, que corresponde a una *transición*, se representa

con una flecha que va de un estado al otro. El estado inicial es al que llega la flecha identificada con inicio.

Figura 4.1 Autómata correspondiente a un apagador



Debemos demostrar que los siguientes enunciados para describir el comportamiento del autómata se cumplen:

$S_1(n)$: El autómata está en el estado **A** (de *apagado*) después de haber oprimido el botón n veces, si y sólo si n es par.

$S_2(n)$: El autómata está en el estado **P** (de *prendido*) si y sólo si n es impar.

Se tiene que hacer una demostración doble de inducción, ya que hay que hacer inducción sobre los dos casos posibles de la definición.

4.3.16.- Un *poliominó* es una pieza formada por cuadrados iguales unidos entre sí por al menos una arista (se excluyen los que estén unidos sólo por un vértice). Los poliominós se clasifican según el número de cuadrados que los forman; así, tenemos los monominós son aquéllos formados por un cuadrado, los dominós por dos cuadrados, triminós por tres, tetraminós por cuatro, pentaminós con cinco, los hexaminós con seis, los heptaminós con siete, . . . En ciencias de la computación a este tipo de uniones donde se requiere que cuadrados adyacentes compartan un lado se conoce también como 4-conectividad.

Al número de cuadrado que tiene el poliominó se le llama el *orden* de la figura. Según el número de cuadrados en el poliominó tendremos un número distinto de figuras con ese número de cuadrados. En la siguiente tabla consideraremos el número de poliominós *libres*, donde dos poliominós son diferentes si uno no es el reflejo, la rotación o la traslación del otro. En la tabla 4.1 en la siguiente página mostramos los poliominós libres de orden 1 a 5.


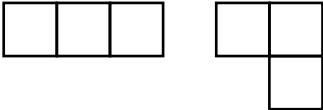
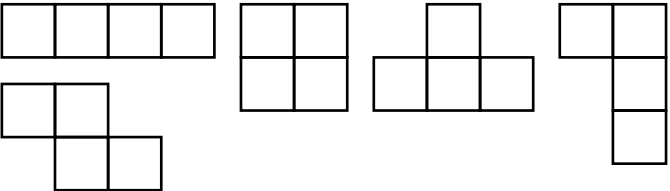
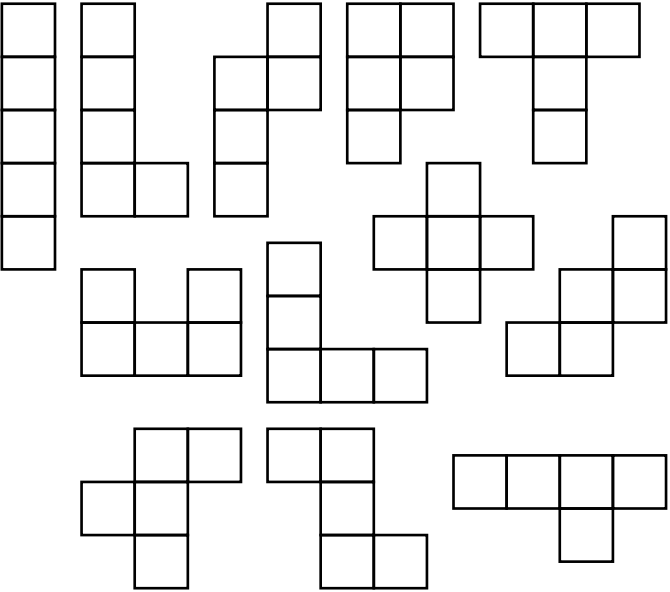
Tabla 4.1 Poliominós libres de orden 1 a 5

Nombre	Número	Figuras
monominó	1	

(Continúa en la siguiente página)

Tabla 4.1 Poliomínos libres de orden 1 a 5

(Continúa de la página anterior)

Nombre	Número	Figuras
dominós	1	
triminós	2	
tetraminós	5	
pentaminós	12	

Consideremos el triminó en forma de L. Consideremos un tablero de $2^n \times 2^n$ cuadros en el que eliminamos un cuadro. Demostrar que el resto del tablero puede ser cubierto con triminós en forma de L.

4.4. Definiciones recursivas

Una definición recursiva es aquella en la cual el concepto definido figura en la definición misma. Esto puede parecer problemático y de hecho introduce problemas matemáticos profundos si dicho uso o autoreferencia se utiliza sin cuidado. Sin embargo, usado bajo ciertas restricciones, este principio de autoreferencia, al que llamaremos en adelante *recursión*, proporciona un método de definición sumamente útil tanto en matemáticas como en ciencias de la computación. En particular, todos los tipos de datos usuales en programación como listas o árboles, así como diversas funciones sobre los mismos, pueden definirse recursivamente.

Para que una definición recursiva sea válida, en el sentido de que genere tipos de datos o funciones que no causen ciclos infinitos de evaluación, debe constar de dos partes:

- Un conjunto de casos base, los cuales son casos simples donde la definición se da directamente, es decir, sin usar autoreferencia.
- Un conjunto de reglas recursivas donde se define un nuevo elemento de la definición en términos de anteriores ya definidos.

Además de estas dos partes, la definición debe constar de una cláusula que asegure que las dos anteriores son las únicas formas de obtener el concepto, objeto o función definida. Esta cláusula puede omitirse en el entendido de que siempre está presente.

La definición en los casos base nos da un punto de partida al proporcionar una definición directa, mientras que las reglas recursivas nos permiten construir nuevos casos a partir de los básicos de una manera iterativa.

Es muy importante observar que las únicas definiciones recursivas que consideramos válidas son aquellas donde las reglas recursivas se definen en términos de elementos anteriores. Por ejemplo, la siguiente definición de una función

$$f(0) = 1$$

$$f(n + 1) = f(n + 2)$$

no es válida, puesto que la definición en $n + 1$ está dada en términos de un elemento posterior a $n + 1$, a saber $n + 2$. En particular, f resulta indefinida en cualquier valor distinto de cero. Definiciones como la anterior se llaman *recursivas generales* y por lo general causan ciclos infinitos en programación.

Ya hemos visto definiciones recursivas del conjunto de números naturales, así como de algunas funciones sobre este mismo tipo de datos como la suma o el producto. Veamos algunos ejemplos más

Ejemplo 4.10. Dada una persona x , la relación *ser descendiente de* x en el dominio de las personas se define como sigue:

- i. Si y es hijo de x entonces y es descendiente de x .
 - ii. Si y es descendiente de x y z es hijo de y entonces z es descendiente de x .
 - iii. Nadie más es descendiente de x .
-

Ejemplo 4.11. Dados dos números naturales n y m , la relación n es menor que m , denotada $n < m$, se define como sigue:

- i. $0 < s(k)$.
 - ii. $s(n) < s(k)$, si $n < k$
 - iii. Ningún otro par de números está en la relación $<$.
-

Obsérvese que en el ejemplo anterior la recursión se hace sobre el número n dejando a m fijo y declarándolo explícitamente como un número sucesor $s(k)$, puesto que la relación $n < 0$ no sucede nunca.

Ejemplo 4.12. El conjunto de fórmulas bien construidas de la lógica proposicional se define como sigue:

- i. Una variable proposicional es una fórmula bien construida.
 - ii. Las constantes lógicas true y false son fórmulas bien construidas.
 - iii. Si A y B son fórmulas bien construidas, entonces $(\neg A)$, $(A \vee B)$, $(A \wedge B)$ y $(A \rightarrow B)$ son fórmulas bien construidas.
 - iv. Ninguna expresión que no sea construida con estas reglas es una fórmula bien construida.
-

Ejemplo 4.13. El conjunto de expresiones aritméticas se define como sigue:

- i. Todos los enteros y todos los nombres de variables son expresiones aritméticas.
 - ii. Si A y B son expresiones aritméticas entonces $(-A)$, $(A + B)$, $(A - B)$, $(A \times B)$ y (A/B) son expresiones aritméticas.
 - iii. Sólo éstas son expresiones aritméticas.
-

Ejemplo 4.14. El tipo de datos de listas finitas $[a_1, \dots, a_n]$ con elementos a_i en un conjunto A se define de la siguiente forma:

- i. La lista vacía es una lista y se denota por $[]$.
- ii. Si $a \in A$ y ℓ es una lista entonces $\text{cons}(a, \ell)$ es una lista. A a se le llama la cabeza y a ℓ la cola de la lista.
- iii. Sólo éstas son listas.

Frecuentemente se usa la notación $(a : \ell)$ para $\text{cons}(a, \ell)$. Por ejemplo, si consideramos al conjunto $A = \{1, 3, 6, 10, 15, 21, 28\}$, la lista $[10, 6, 1, 6]$ que contiene a los elementos 10, 6, 1, 6 en ese orden, se representa de la siguiente manera:

$$(10 : (6, (1, (6, []))))$$

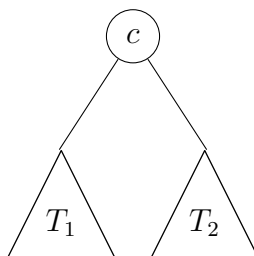
Es conveniente notar que en las listas se admiten repeticiones, a diferencia de lo que sucede con los conjuntos.

Tenemos varias opciones para representar a una lista con un único elemento. $\text{cons}(a, [])$ corresponde a una lista con un primer elemento a y donde la cola de la lista es la lista vacía. También podemos denotar a una lista con un solo elemento, de manera abreviada, como $[a]$, cosa que haremos más adelante.

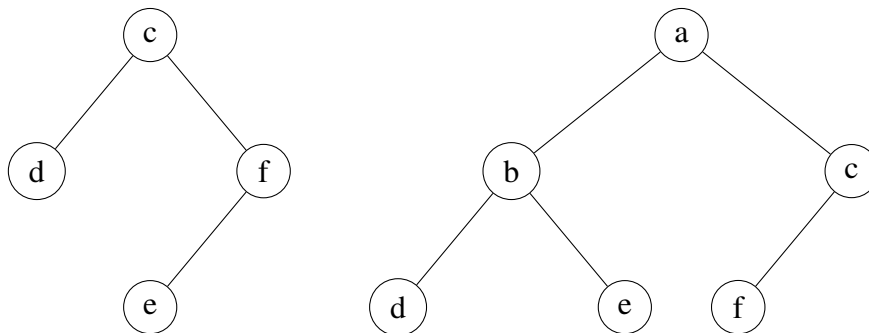
Ejemplo 4.15. El tipo de datos de árboles binarios con todos los nodos etiquetados por elementos de un conjunto A se define como sigue:

- i. Un árbol vacío es un árbol binario y se denota por void .
- ii. Si T_1 y T_2 son árboles binarios y a es un elemento de A , entonces $\text{tree}(T_1, a, T_2)$ es un árbol binario, donde T_1 es el *subárbol izquierdo* y T_2 es el *subárbol derecho*. Al nodo etiquetado con a se le llama la *raíz* del árbol.
- iii. Nada más es un árbol binario.

Por ejemplo, la expresión $\text{tree}(T_1, c, T_2)$ corresponde a la siguiente figura



¿A qué expresión corresponde cada uno de los siguientes árboles?



Los ejemplos anteriores muestran definiciones recursivas de tipos de datos usuales en computación como números naturales, expresiones lógicas o aritméticas, listas o árboles, o bien de relaciones como el orden usual entre números. En la siguiente sección mostramos definiciones recursivas de funciones que involucran a los tipos de datos recién definidos.

4.4.1. Definición de funciones recursivas

La definición recursiva de tipos de datos permite definir funciones sobre los mismos utilizando la técnica de *casamiento o apareamiento de patrones*³: cada cláusula de la definición del tipo de datos introduce un patrón, el cual es un esquema sintáctico bien definido que se utiliza para definir un caso de la función en cuestión. Listamos a continuación los patrones básicos de cada tipo de datos definido previamente:

- Números naturales: $0, s(n)$.
- Expresiones lógicas: $p, \text{true}, \text{false}, \neg A, A \wedge B, A \vee B, A \rightarrow B$
- Expresiones aritméticas: $n, x, (-A), (A + B), (A - B), (A \times B)$ y (A/B)
- Listas: $[], (a : \ell)$
- Árboles binarios: $\text{void}, \text{tree}(T_1, c, T_2)$

De esta manera, para definir, por ejemplo, una función f sobre las listas, es suficiente definir los casos para $f([])$ y para $f((a : \ell))$.

Veamos a continuación algunos ejemplos de funciones definidas sobre los tipos de datos recién definidos y cuyas implementaciones se dan mediante apareamiento de patrones. En cada caso se da primero una especificación que proporciona una definición directa, seguida de una implementación mediante una función recursiva f definida mediante patrones. En algunos ejemplos nos puede resultar claro que la definición recursiva de f cumple con la especificación dada en cada caso. Sin embargo, debemos mostrar esto formalmente para cada ejemplo, proceso que discutiremos en la siguiente sección.

Ejemplo 4.16. Exponenciación de números naturales.

Especificación: $\text{pot}(n, m) = n^m$

Implementación recursiva:

- $f(n, 0) = 1$
- $f(n, s(m)) = f(n, m) \cdot n$

Ejemplo 4.17. Factorial de un número natural.

Especificación: $\text{fac}(n) = n \cdot (n - 1) \cdot \dots \cdot 2 \cdot 1$, donde además $\text{fac}(0) = 1$.

³En inglés *pattern matching*

Implementación recursiva:

- $f(0) = 1$
 - $f(s(n)) = s(n) \cdot f(n)$
-

Ejemplo 4.18.

Especificación: suma de los elementos de una lista de números.

$$\text{suml}([a_1, \dots, a_n]) = a_1 + a_2 + \dots + a_n$$

Implementación recursiva:

- $f([]) = 0$
 - $f((a : \ell)) = a + f(\ell)$
-

Ejemplo 4.19.

Especificación: producto de los elementos de una lista de números.

$$\text{prodl}([a_1, \dots, a_n]) = a_1 \cdot a_2 \cdot \dots \cdot a_n$$

Implementación recursiva:

- $f([]) = 1$
 - $f((a : \ell)) = a \cdot f(\ell)$
-

Ejemplo 4.20.

Especificación: longitud de una lista.

$$\text{long}([a_1, \dots, a_n]) = n$$

Implementación recursiva:

- $f([]) = 0$
 - $f((a : \ell)) = 1 + f(\ell)$
-

Ejemplo 4.21.

Especificación: el operador binario \sqcup devuelve la concatenación de dos listas.

$$[a_1, \dots, a_k] \sqcup [b_1, \dots, b_j] = [a_1, \dots, a_k, b_1, \dots, b_j]$$

Implementación recursiva:

- $f([], \ell_2) = \ell_2$
 - $f((a : \ell_1), \ell_2) = (a : f(\ell_1, \ell_2))$
-

Ejemplo 4.22.

Especificación: reversa de una lista.

$$rev([a_1, \dots, a_k]) = [a_k, \dots, a_1]$$

Implementación recursiva:

- $f([]) = []$
 - $f((a : \ell)) = f(\ell) \sqcup [a]$
-

En algunos casos, la especificación de una función no puede darse de forma directa mediante una ecuación como en los casos anteriores, sino que tiene que darse con palabras como en los siguientes ejemplos.

Ejemplo 4.23.

Especificación: nc es la función que calcula el número de conectivos en una fórmula de la lógica proposicional. Por ejemplo $nc(p \rightarrow \neg q \vee r) = 3$.

Implementación recursiva:

- $f(p) = 0$
 - $f(\text{true}) = f(\text{false}) = 0$
 - $f(\neg A) = 1 + f(A)$
 - $f(A \wedge B) = 1 + f(A) + f(B)$
 - $f(A \vee B) = 1 + f(A) + f(B)$
 - $f(A \rightarrow B) = 1 + f(A) + f(B)$
-

Ejemplo 4.24.

Especificación: *ccd* es la función que recibe una fórmula proposicional *A* y devuelve la fórmula obtenida a partir de *A* al intercambiar los conectivos \wedge y \vee en *A*. Por ejemplo $ccd(\neg p \vee q \rightarrow r \wedge s) = \neg p \wedge q \rightarrow r \vee s$.

Implementación recursiva:

- $f(p) = p$
 - $f(\text{true}) = \text{true}$
 - $f(\text{false}) = \text{false}$
 - $f(\neg A) = \neg f(A)$
 - $f(A \wedge B) = f(A) \vee f(B)$
 - $f(A \vee B) = f(A) \wedge f(B)$
 - $f(A \rightarrow B) = f(A) \rightarrow f(B)$
-

Ejemplo 4.25.

Especificación: *at* es la función que calcula el número de presencias de fórmulas atómicas que figuran en una fórmula. Por ejemplo:

$$at(q \wedge \neg p \rightarrow r \vee p) = 4.$$

$$at(q \wedge (\text{true} \vee \neg(r \rightarrow \text{false} \wedge t))) = 5.$$

Implementación recursiva:

- $f(p) = 1$
 - $f(\text{true}) = f(\text{false}) = 1$
 - $f(\neg A) = f(A)$
 - $f(A \wedge B) = f(A) + f(B)$
 - $f(A \vee B) = f(A) + f(B)$
 - $f(A \rightarrow B) = f(A) + f(B)$
-

El ejemplo en la siguiente pagina involucra a dos tipos de datos, el de las fórmulas proposicionales y el de listas de fórmulas.

Ejemplo 4.26.

Especificación: sf es la función que devuelve la lista de subfórmulas de una fórmula proposicional A . Por ejemplo,

$$sf(\neg(p \rightarrow q) \wedge r) = [\neg(p \rightarrow q) \wedge r, \neg(p \rightarrow q), p \rightarrow q, p, q, r].$$

$$sf(p \vee \neg(p \rightarrow s)) = [p \vee \neg(p \rightarrow s), p, \neg(p \rightarrow s), p \rightarrow s, p, s]$$

Implementación recursiva:

- $f(p) = [p]$
 - $f(\text{true}) = [\text{true}]$
 - $f(\text{false}) = [\text{false}]$
 - $f(\neg A) = (\neg A : f(A))$
 - $f(A \wedge B) = ((A \wedge B) : (f(A) \sqcup f(B)))$
 - $f(A \vee B) = ((A \vee B) : (f(A) \sqcup f(B)))$
 - $f(A \rightarrow B) = ((A \rightarrow B) : (f(A) \sqcup f(B)))$
-

Ejemplo 4.27.

Especificación: nn es la función que recibe un árbol binario t y calcula el número de nodos que hay en t .

Implementación recursiva:

- $f(\text{void}) = 0$
 - $f(\text{tree}(T_1, c, T_2)) = 1 + f(T_1) + f(T_2)$
-

Ejemplo 4.28.

Especificación: la profundidad o altura de un nodo x en un árbol binario T se define como la distancia (número de líneas) existente entre x y la raíz de T en la representación gráfica de T . La profundidad o altura de un árbol T se define como la altura máxima de un nodo de T más uno. ht es la función que calcula la profundidad de un árbol binario.

Implementación recursiva:

- $f(\text{void}) = 0$

$$\bullet f(\text{tree}(T_1, c, T_2)) = 1 + \max\{f(T_1), f(T_2)\}$$

Como ya mencionamos, en cada caso debemos cerciorarnos formalmente que la definición recursiva dada por f realmente cumple con la especificación dada. Para el caso de funciones que involucren a los números naturales esto puede lograrse mediante el principio de inducción matemática. Como ejemplo, veamos que la definición recursiva del factorial en verdad cumple la especificación.

Ejemplo 4.29. La definición recursiva de f en el ejemplo 4.17 calcula a la función factorial. Es decir, para todo número natural n , se cumple $f(n) = \text{fac}(n)$.

Base: $n = 0$. Tenemos $f(0) = 1 = \text{fac}(0)$.

Hipótesis de inducción: $f(n) = \text{fac}(n)$.

Paso inductivo: queremos demostrar que $f(s(n)) = \text{fac}(s(n))$.

$$\begin{aligned} f(s(n)) &= s(n) \cdot f(n) && \text{(definición de } f) \\ &= s(n) \cdot \text{fac}(n) && \text{(hipótesis de inducción)} \\ &= (n+1) \cdot n \cdot (n-1) \cdot \dots \cdot 2 \cdot 1 && \text{(definición de } \text{fac}(n)) \\ &= \text{fac}(s(n)) && \text{(definición de } \text{fac}(s(n))) \end{aligned}$$

En conclusión $f(n) = \text{fac}(n)$ para todo número natural n .

Ahora bien, para el caso de funciones definidas sobre otro tipo de datos ¿Cómo podemos probar que la especificación se satisface con la implementación recursiva?

De las definiciones y pruebas por inducción de las operaciones de suma y producto, así como de la prueba del ejemplo anterior, se observa una fuerte relación entre el principio de inducción matemática y las definiciones recursivas que involucran números naturales. Cada propiedad de la definición recursiva, como cumplir con una especificación dada, puede mostrarse mediante el principio de inducción. Esta relación puede generalizarse a distintas estructuras o tipos de datos definidos recursivamente, lo que haremos a continuación.

4.5. Inducción estructural

Para demostrar propiedades acerca de estructuras definidas recursivamente en el sentido descrito en la página 178, es posible recurrir a la inducción matemática, definiendo una medida en la estructura en cuestión, lo que se hace mediante un número natural. Entre

las medidas que podemos mencionar están la longitud de una lista, la profundidad de un árbol o el número de conectivos en una fórmula proposicional. Esto es posible debido a que las reglas recursivas de la definición en cuestión se dan en términos de elementos estructuralmente más simples, por lo que su medida será menor y la hipótesis de inducción podrá emplearse. Sin embargo, en la mayoría de los casos, el uso de una medida complica las pruebas, además de que la elección de una medida incorrecta podría resultar en una prueba fallida. Otra posibilidad es generalizar el principio de inducción completa mediante la definición de un orden en los tipos de datos, el cual debe ser bien fundado, es decir, no debe contener sucesiones descendentes infinitas. Sin embargo, el problema de decidir si una estructura particular es bien fundada no siempre es fácil de resolver.

En lugar de las alternativas anteriores es posible utilizar los llamados principios de inducción estructural, basados en las reglas base y recursivas de la definición de un tipo de datos, así como en el análisis de los patrones básicos introducidos por éstas.

El esquema general del principio de inducción estructural es el siguiente: Sean A un conjunto o tipo de datos definido recursivamente y P una propiedad acerca de los elementos de A . Para probar que $P(x)$ es válida para todo elemento de A deben seguirse los siguientes pasos:

- **Base de la inducción:** Si a es un elemento de A generado por una regla básica, entonces debemos probar directamente la validez de $P(a)$.
- Si x es un elemento de A construido mediante alguna regla recursiva a partir de elementos anteriores⁴ x_1, \dots, x_n , entonces procedemos como sigue:

Hipótesis de inducción: Suponer $P(x_1), \dots, P(x_n)$.

Paso inductivo: Probar $P(x)$.

- En este caso, el principio de inducción estructural permite concluir que $\forall x P(x)$.

Este principio debe adaptarse a cada conjunto o tipo de datos en particular. En las siguientes secciones lo ejemplificamos para los casos de listas, árboles y fórmulas proposicionales.

4.5.1. Inducción en listas

El tipo de datos lista es uno de los más comunes en ciencias de la computación. Este tipo de datos se definió recursivamente en el ejemplo 4.14, y genera el siguiente principio de inducción estructural:

Sea P una propiedad acerca de listas; si se desea probar $P(xs)$ para toda lista xs , basta proceder como sigue:

Base de la inducción: Probar $P([])$ directamente.

⁴Es decir, elementos estructuralmente más simples.

Hipótesis de inducción: Suponer $P(xs)$.

Paso inductivo: Probar $P((a : xs))$.

Si este es el caso, el principio de inducción para listas permite concluir $P(xs)$ para cualquier lista xs .

Para ilustrar el uso de la inducción en listas probamos enseguida algunas propiedades de las operaciones en listas.

Proposición 4.1 *La función recursiva f dada en el ejemplo 4.21 calcula la concatenación de dos listas. Es decir, para cualesquiera listas xs, ys , $f(xs, ys) = xs \sqcup ys$.*

Demostración. Inducción sobre xs .

Base de la inducción: $xs = []$. Tenemos $[] \sqcup ys = ys = f([], ys)$.

Hipótesis de inducción: $f(xs, ys) = xs \sqcup ys$.

Paso inductivo: Debemos mostrar que $f((a : xs), ys) = (a : xs) \sqcup ys$.

$$\begin{aligned}
 (a : xs) \sqcup ys &= a : (xs \sqcup ys) && \text{(razonamiento directo)} \\
 &= a : f(xs, ys) && \text{(hipótesis de inducción)} \\
 &= f((a : xs), ys) && \text{(definición recursiva de } f)
 \end{aligned}$$

En conclusión $f(xs, ys) = xs \sqcup ys$ para cualesquiera listas xs, ys .

De manera similar podemos probar la correctud de todas definiciones recursivas dadas en los ejemplos de la sección 4.4.1.

Como ya probamos que la implementación de la concatenación \sqcup es correcta, podemos usarla de ahora en adelante, como en el caso de la siguiente proposición.

Proposición 4.2 *La operación de concatenación \sqcup en listas cumple las siguientes propiedades:*

- *Asociatividad:* $xs \sqcup (ys \sqcup zs) = (xs \sqcup ys) \sqcup zs$
- *Longitud:* $\text{long}(xs \sqcup ys) = \text{long}(xs) + \text{long}(ys)$

Demostración. Probamos la asociatividad mediante inducción sobre la lista xs .

Sea $P(xs)$ la propiedad $xs \sqcup (ys \sqcup zs) = (xs \sqcup ys) \sqcup zs$

Base de la inducción: $xs = []$, debemos mostrar que

$$[] \sqcup (ys \sqcup zs) = ([] \sqcup ys) \sqcup zs$$

$$\begin{aligned} [] \sqcup (ys \sqcup zs) &= ys \sqcup zs && \text{(def. rec. de } \sqcup \text{)} \\ &= ([] \sqcup ys) \sqcup zs && (ys = [] \sqcup ys) \end{aligned}$$

Hipótesis de inducción: $xs \sqcup (ys \sqcup zs) = (xs \sqcup ys) \sqcup zs$.

Paso inductivo: sea a un elemento de A , debemos mostrar que

$$(a : xs) \sqcup (ys \sqcup zs) = ((a : xs) \sqcup ys) \sqcup zs$$

$$\begin{aligned} (a : xs) \sqcup (ys \sqcup zs) &= a : (xs \sqcup (ys \sqcup zs)) && \text{(def.rec. de } \sqcup \text{)} \\ &= a : ((xs \sqcup ys) \sqcup zs) && \text{(hipótesis de inducción)} \\ &= (a : (xs \sqcup ys)) \sqcup zs && \text{(def.rec. de } \sqcup \text{)} \\ &= ((a : xs) \sqcup ys) \sqcup zs && \text{(def.rec. de } \sqcup \text{)} \end{aligned}$$

Así que por el principio de inducción para listas, concluimos que la operación *app* es asociativa.

La propiedad de longitud se demuestra similarmente.

Proposición 4.3 *La operación reversa rev en listas cumple las siguientes propiedades:*

- *Longitud:* $\text{long}(\text{rev}(xs)) = \text{long}(xs)$
- *Concatenación:* $\text{rev}(xs \sqcup ys) = \text{rev}(ys) \sqcup \text{rev}(xs)$
- *Idempotencia:* $\text{rev}(\text{rev}(xs)) = xs$

Demostración. Mostramos la propiedad de idempotencia mediante inducción sobre la lista xs , dejando las restantes como ejercicio.

Base de la inducción: $xs = []$. Como $\text{rev}([]) = []$ entonces

$$\text{rev}(\text{rev}([])) = \text{rev}([]) = [].$$

Hipótesis de inducción: $\text{rev}(\text{rev}(xs)) = xs$

Paso inductivo: Sea a un elemento de A ; mostraremos que

$$\text{rev}(\text{rev}((a : xs))) = (a : xs).$$

$$\begin{aligned} \text{rev}(\text{rev}((a : xs))) &= \text{rev}(\text{rev}(xs) \sqcup [a]) && \text{(definición recursiva de rev)} \\ &= \text{rev}([a]) \sqcup \text{rev}(\text{rev}(xs)) && \text{(proposición anterior)} \\ &= \text{rev}([a]) \sqcup xs && \text{(hipótesis de inducción)} \\ &= [a] \sqcup xs && (\text{rev}([a]) = [a]) \\ &= (a : []) \sqcup xs && ([a] = (a : [])) \\ &= a : ([] \sqcup xs) && \text{(definición recursiva de } \sqcup \text{)} \\ &= (a : xs) && \text{(definición recursiva de } \sqcup \text{)} \end{aligned}$$

Conclusión: Así que por el principio de inducción para listas se cumple

$$\text{rev}(\text{rev}(xs)) = xs \text{ para toda lista } xs.$$

Pasamos ahora a ilustrar la inducción estructural en fórmulas proposicionales.

4.5.2. Inducción en fórmulas

El conjunto de fórmulas de la lógica proposicional se definió ya mediante una gramática, así como mediante la definición recursiva del ejemplo 4.12. Esta última forma de definirlo habilita un principio de inducción estructural de gran utilidad en lógica matemática.

El principio de inducción estructural para fórmulas es el siguiente:

Sea P una propiedad acerca de fórmulas proposicionales. Si se desea probar $P(A)$ para toda fórmula A , basta proceder como sigue:

Base de la inducción: probar $P(q)$ directamente para cada variable proposicional q ; probar $P(\text{true})$ y probar $P(\text{false})$

Hipótesis de inducción: suponer $P(A)$ y $P(B)$.

Paso inductivo: probar $P(\neg A)$, $P(A \wedge B)$, $P(A \vee B)$ y $P(A \rightarrow B)$.

Conclusión: En tal caso el principio de inducción para fórmulas permite concluir $P(A)$ para cualquier fórmula A .

En este caso, debido a nuestros conocimientos de equivalencias lógicas, el paso inductivo puede simplificarse a probar $P(\neg A)$ y alguno de los casos para un operador binario, el cual se elige dependiendo de la propiedad P particular.

Demostramos a continuación algunas propiedades de las fórmulas proposicionales.

Proposición 4.4 Sea $comp$ la siguiente función recursiva:

$$comp(p) = \neg p \quad (i)$$

$$comp(true) = false \quad (ii)$$

$$comp(false) = true \quad (iii)$$

$$comp(\neg A) = \neg comp(A) \quad (iv)$$

$$comp(A \vee B) = comp(A) \wedge comp(B) \quad (v)$$

$$comp(A \wedge B) = comp(A) \vee comp(B) \quad (vi)$$

Entonces, para toda fórmula C , se cumple $comp(C) \equiv \neg C$.

Demostración. Inducción sobre las fórmulas.

Base: C es atómica. Si $C = p$ entonces hay que mostrar $comp(p) \equiv \neg p$.

$$\begin{aligned} comp(p) &= \neg p && \text{(por (i))} \\ &\equiv \neg p && \text{(reflexividad de } \equiv \text{)} \end{aligned}$$

Los casos para $C = true$ y $C = false$ son similares.

Hipótesis de inducción Supongamos que $comp(A) \equiv \neg A$ y $comp(B) \equiv \neg B$.

Paso inductivo: Dado nuestro conocimiento de las equivalencias lógicas, basta mostrar la propiedad para $\neg A$ y $A \wedge B$.

$$\begin{aligned} comp(\neg A) &= \neg comp(A) && \text{(por (iv))} \\ &\equiv \neg \neg A && \text{(hipótesis de inducción y equivalencia lógica)} \\ comp(A \wedge B) &= comp(A) \vee comp(B) && \text{(por (vi))} \\ &\equiv \neg A \vee \neg B && \text{(hipótesis de inducción y equivalencia lógica)} \\ &\equiv \neg(A \wedge B) && \text{(De Morgan)} \end{aligned}$$

Conclusión: Por el principio de inducción para fórmulas, podemos concluir que $comp(C) \equiv \neg C$, para cualquier fórmula C .

Por último demostramos una propiedad que relaciona a las funciones definidas en los ejemplos 4.20, 4.23 y 4.25.

Proposición 4.5 Si A es una fórmula proposicional, entonces la longitud de la lista de subfórmulas de A es igual a la suma del número de presencias de variables proposicionales de A con el número de conectivos que figuran en A . Es decir,

$$long(sf(A)) = at(A) + nc(A)$$

Demostración. Inducción sobre la fórmula A .

Base de la inducción: Sea $A = p$. Tenemos en A una presencia de la fórmula atómica p y ningún conector. Por lo tanto,

$$\text{long}(sf(p)) = \text{long}([p]) = 1 = 1 + 0 = \text{at}(p) + \text{nc}(p)$$

Para $A = \text{true}$ o $A = \text{false}$, la prueba es similar.

Hipótesis de inducción: Supongamos que

$$\text{long}(sf(A)) = \text{at}(A) + \text{nc}(A) \quad \text{long}(sf(B)) = \text{at}(B) + \text{nc}(B).$$

Paso inductivo: Probamos la propiedad para $\neg A$ y $A \rightarrow B$.

$$\begin{aligned} \text{long}(sf(\neg A)) &= \text{long}((\neg A : sf(A))) && \text{(definición de } sf) \\ &= 1 + \text{long}(sf(A)) && \text{(definición recursiva de } long) \\ &= 1 + (\text{at}(A) + \text{nc}(A)) && \text{(hipótesis de inducción)} \\ &= \text{at}(A) + (1 + \text{nc}(A)) && \text{(aritmética)} \\ &= \text{at}(\neg A) + (1 + \text{nc}(A)) && \text{(definición recursiva de } at) \\ &= \text{at}(\neg A) + \text{nc}(\neg A) && \text{(definición recursiva de } nc) \end{aligned}$$

$$\begin{aligned} \text{long}(sf(A \rightarrow B)) &= \text{long}(((A \rightarrow B) : sf(A) \sqcup sf(B))) && \text{(definición de } sf) \\ &= 1 + \text{long}(sf(A) \sqcup sf(B)) && \text{(definición recursiva de } long) \\ &= 1 + \text{long}(sf(A)) + \text{long}(sf(B)) && \text{(proposición de } \sqcup) \\ &= 1 + (\text{at}(A) + \text{nc}(A)) + (\text{at}(B) + \text{nc}(B)) && \text{(hipótesis de inducción)} \\ &= (\text{at}(A) + \text{at}(B)) + (1 + \text{nc}(A) + \text{nc}(B)) && \text{(aritmética)} \\ &= \text{at}(A \rightarrow B) + (1 + \text{nc}(A) + \text{nc}(B)) && \text{(definición recursiva de } at) \\ &= \text{at}(A \rightarrow B) + \text{nc}(A \rightarrow B) && \text{(definición recursiva de } nc) \end{aligned}$$

Conclusión: Por lo tanto, por el principio de inducción para fórmulas, para cualquier fórmula A se cumple

$$\text{long}(sf(A)) = \text{at}(A) + \text{nc}(A)$$

Para finalizar este capítulo discutimos el principio de inducción para árboles binarios.

4.5.3. Inducción en árboles

La definición recursiva del tipo de datos de árboles binarios dada en el ejemplo 4.15 genera la siguiente versión del principio de inducción estructural:

Sea P una propiedad acerca de árboles binarios. Si se desea probar $P(T)$ para todo árbol T , basta proceder como sigue:

Base de la inducción: Probar $P(\text{void})$ directamente.

Hipótesis de inducción: Suponer $P(T_1)$ y $P(T_2)$.

Paso inductivo: Probar $P(\text{tree}(T_1, c, T_2))$.

Conclusión: En tal caso el principio de inducción para árboles permite concluir $P(T)$ para cualquier árbol T .

Veamos a continuación un par de ejemplos de pruebas mediante este principio de inducción.

Proposición 4.6 *Cualquier árbol binario T con n nodos contiene exactamente $n + 1$ subárboles vacíos.*

Demostración. Inducción sobre T .

Base: $T = \text{void}$. El número de nodos de T es 0, y el número de subárboles vacíos es $0 + 1 = 1$, pues T mismo es un subárbol binario vacío.

Hipótesis de inducción: Si los árboles binarios T_1 y T_2 tienen n_1 y n_2 nodos respectivamente, entonces tienen $n_1 + 1$ y $n_2 + 1$ subárboles vacíos respectivamente.

Paso inductivo: Sea $T = (T_1, c, T_2)$ un árbol binario no vacío. El número de nodos de T es $1 + n_1 + n_2$. Queremos demostrar que T tiene $n_1 + n_2 + 2$ árboles vacíos.

Es claro que los subárboles vacíos de T son subárboles de T_1 o de T_2 , por lo que se tiene que el número de subárboles vacíos de T es igual a la suma de los números de subárboles vacíos de T_1 y de T_2 ; pero por la hipótesis de inducción dicha suma es igual a $(n_1 + 1) + (n_2 + 1) = n_1 + n_2 + 2$.

Conclusión: Todos los árboles binarios con n nodos tienen $n + 1$ subárboles vacíos.

Una hoja a de un árbol es aquel nodo del que cuelgan únicamente árboles vacíos, representado por $\text{tree}(\text{void}, a, \text{void})$. Definimos la altura de un árbol general como uno más de la distancia de la raíz a la hoja más lejana, donde la distancia es el número de aristas que se tienen que recorrer desde la raíz para llegar al nodo en la representación gráfica del árbol.

Proposición 4.7 *Si T es un árbol binario con altura n , entonces tiene a lo más $2^n - 1$ nodos. Es decir, $nn(T) \leq 2^n - 1$*

Demostración. Inducción sobre T .

Base: $T = \text{void}$. En este caso la altura de T es 0 y $nn(T) = 0$ pues T no tiene nodos; por otro lado, $2^0 - 1 = 1 - 1 = 0$, con lo que queda demostrada la base de la inducción.

Hipótesis de inducción: Si el árbol T_i tiene altura n_i entonces tiene a lo más $2^{n_i} - 1$ nodos, donde $i = 1, 2$.

Paso inductivo: Sea $T = \text{tree}(T_1, c, T_2)$. Recordemos que la altura de T es igual a $1 + \max\{n_1, n_2\}$, por lo que debemos demostrar que el máximo número de nodos en T es $2^{1+\max\{n_1, n_2\}} - 1$, es decir que $nn(T) \leq 2^{1+\max\{n_1, n_2\}} - 1$.

$$\begin{aligned}
 nn(T) &= nn(T_1) + nn(T_2) + 1 && \text{(Definición recursiva de nn)} \\
 &\leq 2^{n_1} - 1 + 2^{n_2} - 1 + 1 && \text{(hipótesis de inducción)} \\
 &\leq 2^{\max\{n_1, n_2\}} - 1 + 2^{\max\{n_1, n_2\}} - 1 + 1 && \text{(aritmética)} \\
 &= 2 \cdot 2^{\max\{n_1, n_2\}} - 1 && \text{(aritmética)} \\
 &= 2^{1+\max\{n_1, n_2\}} - 1 && \text{(leyes de exponentes)}
 \end{aligned}$$

Este resultado particular es muy utilizado en computación.

Ejercicios

4.5.1.- Para cada ejemplo de la sección 4.4.1 demuestra mediante el principio de inducción estructural correspondiente que la función especificada cumple con la implementación recursiva.

4.5.2.- Considera las siguientes especificaciones de dos funciones *spar* y *simp* cuyo dominio y codominio son los números naturales.

$$spar(n) = 2 + 4 + 6 + \dots + 2n \quad simp(n) = 1 + 3 + 5 + \dots + (2n + 1)$$

- Propone implementaciones recursivas f y g para *spar* y *simp*, respectivamente.
- Muestra que $f(n) = n(n + 1)$
- Muestra que $g(n) = (n + 1)^2$.

4.5.3.- Definimos al conjunto de cadenas $a^m b a^m$ de la siguiente manera:

- b , la cadena representada por $a^0 b a^0$, está en el conjunto.
- Si w es una cadena en este conjunto, entonces awa también está en el conjunto.
- Éstas son las únicas formas de construir cadenas que cumplan con ser $a^m b a^m$.

Demuestra que todas las cadenas que pertenecen a este conjunto tienen un número impar de caracteres, utilizando los siguientes métodos:

- Inducción sobre la longitud de las cadenas.
- Definiendo y utilizando un principio de inducción estructural adecuado.

4.5.4.- Considera la definición recursiva de las expresiones aritméticas dada en el ejemplo 4.13. Enuncia el principio de inducción estructural correspondiente y utilízalo para demostrar que toda expresión aritmética tiene el mismo número de paréntesis izquierdos que derechos.

4.5.5.- Una cadena de caracteres es palíndroma si es de la forma ww^R , donde w^R es w escrita de atrás hacia adelante. Algunos ejemplos son 0110 y *aabaabaa*. Define al conjunto de las cadenas palíndromas en forma recursiva y demuestra mediante inducción estructural que todas las cadenas palíndromas de este tipo tiene un número par de símbolos.

4.5.6.- Demuestra mediante inducción para listas lo siguiente:

- a) La propiedad de longitud enunciada en la proposición 4.2
- b) Las propiedades de concatenación e idempotencia para la reversa, enunciadas en la proposición 4.3

4.5.7.- La función `snoc` en listas se define como sigue:

$$\text{snoc } c [x_1, \dots, x_n] = [x_1, \dots, x_n, c]$$

- a) Da una definición recursiva para `snoc`.
- b) Demuestra que:

$$\text{snoc } c (xs \sqcup ys) = xs \sqcup (\text{snoc } c ys)$$

- c) Demuestra la siguiente propiedad que relaciona a `snoc` con la operación reversa `rev`:

$$\text{rev } (\text{snoc } c xs) = c : (\text{rev } xs)$$

4.5.8.- Considera la siguiente función misteriosa `mist`:

$$\begin{aligned} \text{mist } [] ys &= ys \\ \text{mist } (x : xs) ys &= \text{mist } xs (x : ys) \end{aligned}$$

- a) ¿Qué hace `mist`?
- b) Muestra que $\text{rev } xs = \text{mist } xs []$

4.5.9.- Este ejercicio concierne a la operación de sustitución textual para las fórmulas de la lógica proposicional.

- a) Define recursivamente la operación de sustitución textual $A[p := B]$.
- b) Demuestra las siguientes propiedades mediante inducción para fórmulas:
 - Si p no figura en A , entonces $A[p := B] = A$.
 - Si $p \neq q$ y p no figura en C , entonces

$$A[p := B][q := C] = A[q := C][p := B[q := C]].$$

- Si $p \neq q$ y p no figura en B , entonces

$$A[q, p := B, C] = A[q := B][p := C].$$

4.5.10.- Sea A una fórmula de la lógica proposicional cuyos únicos conectivos son \wedge, \vee, \neg . Construimos la fórmula dual de A , denotada A_D , intercambiando \wedge con \vee , y reemplazando cada variable p por su negación $\neg p$. Por ejemplo, si $A = (r \vee p) \wedge \neg q$, entonces $A_D = (\neg r \wedge \neg p) \vee \neg \neg q$.

- Define recursivamente una función *dual* tal que $dual(A) = A_D$.
- Muestra que $\neg A \equiv A_D$ mediante inducción sobre las fórmulas.

4.5.11.- Define recursivamente al conjunto de términos de la lógica de predicados y enuncia el principio de inducción estructural correspondiente.

4.5.12.- Define recursivamente al conjunto de fórmulas de la lógica de predicados y enuncia el principio de inducción estructural correspondiente. Observa que este principio debe incluir al dado en la sección 4.5.2 para la lógica de proposiciones.

4.5.13.- Define recursivamente las siguientes funciones para términos de la lógica de predicados:

- $ctes(t)$ que devuelva el conjunto de constantes que figuran en t . Por ejemplo, $ctes(f(a, g(x, b))) = \{a, b\}$
- $vars(t)$ que devuelva el conjunto de variables que figuran en t . Por ejemplo, $vars(g(x, f(y), h(b))) = \{x, y\}$.
- $func(t)$ que devuelva el conjunto de símbolos de función que figuran en t . Por ejemplo, $func(f(a, g(x, b))) = \{f, g\}$.

4.5.14.- La operación de sustitución textual puede extenderse a los términos de la lógica de predicados. Si t y r son términos y x es una variable entonces $t[x := r]$ denota a la sustitución textual de x por r en t . Por ejemplo $f(a, x, g(y, x))[x := h(w)] = f(a, h(w), g(y, h(w)))$. Realiza lo siguiente:

- Formula una definición recursiva de $t[x := r]$.
- Muestra que si $x \notin vars(t)$, entonces $t[x := r] = t$.
- Muestra que:

$$vars(t[x := r]) = (vars(t) \setminus \{x\}) \cup vars(r).$$

4.5.15.- Define recursivamente las siguientes funciones para fórmulas de la lógica de predicados:

- $fv(A)$ que devuelva el conjunto de variables libres de A . Por ejemplo, $fv(\forall x P(x, y) \wedge \exists w Q(z, w)) = \{y, z\}$

- b) $bv(A)$ que devuelva el conjunto de variables ligadas de A . Por ejemplo, $bv(\forall xP(x, y) \wedge \exists wQ(z, w)) = \{x, w\}$
- c) $nq(A)$ que devuelva el número de cuantificadores que figuran en A . Por ejemplo, $bv(\forall x\exists yP(x, y) \wedge \exists w\forall zQ(z, w)) = 4$

4.5.16.- Demuestra que el mínimo número de nodos en un árbol de altura n es n .

4.5.17.- Demuestra que el número máximo de hojas en un árbol de altura n es 2^{n-1} y que el máximo número de nodos internos es $2^{n-1} - 1$

4.5.18.- Demuestra que el siguiente recorrido en un árbol binario reporta a todos los nodos del árbol y siempre termina.

Reglas para reportar un árbol binario.

- a) Si el árbol es un árbol vacío, reporta void y regresa.
- b) Si el árbol es $tree(A, c, B)$, donde A y B son árboles binarios, entonces:
 - i. Reporta c .
 - ii. Reporta A .
 - iii. Reporta B .
 - iv. Termina.

4.5.19.- Define recursivamente una función *aplana* que tome un árbol binario y devuelva la lista de sus nodos empezando por la raíz y siguiendo con los nodos del subárbol izquierdo y derecho recursivamente.

Por ejemplo, si $T = tree(tree(hoja(1), 6, hoja(2)), 5, tree(hoja(4), 9, void))$, donde $hoja(n) = tree(void, n, void)$, entonces $aplana(T) = [5, 6, 1, 2, 9, 4]$.

Muestra que para cualquier árbol t , se cumple $nn(t) = long(aplana(t))$.

4.5.20.- Queremos representar árboles binarios cuyas únicos nodos etiquetados son las hojas. Para eso tenemos la siguiente definición:

- Si $a \in A$, entonces $hoja(a)$ es un árbol.
- Si t_1, t_2 son árboles, entonces $mk(t_1, t_2)$ es un árbol.
- Son todos.

Observa que en esta definición no existe el árbol vacío.

- a) Define funciones recursivas nh , nni que calculen el número de hojas y el número de nodos internos de un árbol (es decir los nodos que no son hojas).
- b) Enuncia el principio de inducción estructural correspondiente y utilízalo para mostrar que:

$$nh(t) = nni(t) + 1.$$

Parte III

Teoría de Gráficas

Conceptos de teoría de gráficas | 5

5.1. Motivación

Una de las actividades más importantes de todo científico, y en particular de los dedicados a Computación es la del *modelado*. *Modelar* un problema quiere decir traducirlo del lenguaje natural a un lenguaje matemático en el que podamos expresar de manera más precisa las características de lo que estamos modelando, y poder manipularlo también de manera precisa y válida.

Hemos estado manejando ya modelos cuando trabajamos con Cálculo Proposicional y con Cálculo de Predicados, tratando de modelar estados de la vida real. Pasaremos ahora a modelar otro tipo de problemas.

Una vez que tenemos un modelo apropiado, podemos usar la computadora para manipularlo. En general, estamos preocupados con tres problemas:

- i. ¿Existe una solución para el problema? Nos interesa una solución que pueda ser calculada (o encontrada) por una computadora en un tiempo razonable.
- ii. ¿Existe una *solución óptima* para el problema? Esto es, podemos calcular una solución que sea mejor que cualquier otra solución dada para ese problema.
- iii. Por último, ¿Cuántas soluciones distintas existen para el problema dado?

Veamos algunos ejemplos.

Existencia de solución: Cuatro parejas casadas juegan tenis de dobles mixtos en dos canchas cada domingo en la noche. Juegan durante dos horas, pero intercambian parejas y oponentes al final de cada período de media hora. ¿Existe una programación de tal manera que cada hombre juegue con y contra cada mujer exactamente una vez, y juega contra cada hombre al menos una vez?

Contar el número de soluciones: Un grupo de inversionistas decide rotar los puestos de presidente y tesorero cada año. ¿Cuántos años van a transcurrir antes de que tenga que repetir alguno de los socios en alguna de las dos posiciones?

Optimización: Un empresario tiene tres empleados, Patty, Enrique y Roque, a quienes les paga \$60, \$70 y \$80 pesos la hora respectivamente. El empresario tiene tres trabajos por asignar. La siguiente tabla muestra cuánto tiempo requiere cada trabajador para hacer cada uno de los trabajos. ¿Cuál es la manera de asignar el trabajo para que salga tan barato como sea posible?

	Patty	Enrique	Roque
Trabajo 1	7.5 hr.	6 hr.	6.5 hr
Trabajo 2	8 hr.	8.5 hr.	7 hr.
Trabajo 3	5 hr.	6.5 hr.	5.5 hr.

Estos tres problemas, de alguna manera, tienen que ver con combinatoria, las distintas maneras que se tienen de resolver el problema y cómo elegir la mejor de ellas. La solución para muchos de estos problemas está dada por algoritmos. Un algoritmo es un método de solución que cumple con:

Entradas: El algoritmo trabaja a partir de cero o más datos. Cuando son cero datos, es porque trabaja a partir de constantes. Por ejemplo, tenemos un algoritmo que elabora una tabla de senos, y empieza a producir valores empezando con un valor constante.

Salidas: El algoritmo produce un resultado.

Finitud: El número de pasos del algoritmo es finito.

Definición: Cada paso está bien definido y susceptible de ser ejecutado por un hombre con papel y lápiz.

Terminación: El algoritmo siempre debe terminar.

No para todos los problemas hay algoritmos que los resuelvan. Más adelante, en su contacto con las ciencias de la computación estudiarán que hay más problemas que algoritmos, por lo que algunos problemas se tendrán que quedar sin solución algorítmica.

En este capítulo nos haremos las tres preguntas que acabamos de plantear, revisando aquellos problemas que se pueden modelar con gráficas, donde una gráfica es un modelo matemático.

5.1.1. Tiempo para completar un proyecto

El problema:

La Sociedad Mexicana de Ciencias de la Computación está organizando un Encuentro para llevarse a cabo a principios del mes de marzo, y tiene que mandar propaganda (un folleto de 8 páginas) para avisar del evento. Esta propaganda debe ser enviada al menos 10 días antes de la fecha del evento para que sea efectivo, pero se deben hacer varias tareas y tomar algunas decisiones antes de elaborar el folleto. El comité organizador del evento debe decidir que temas se van a tratar en el encuentro, y el comité académico debe decidir a quienes invitar para que sean árbitros de los trabajos. Entonces un comité local debe decidir a quién invitar para conferencias magistrales sobre los temas decididos.

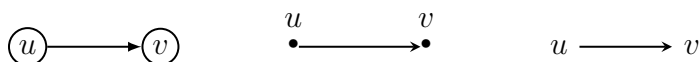
El comité organizador debe preparar dibujos alusivos a los temas a tratar, y alguien tiene que redactar las descripciones cortas de las conferencias magistrales. Finalmente se junta toda la información y se elabora la propaganda requerida, para que se envíe por correo.

El comité de propaganda elabora una lista de correo de a quienes enviar la propaganda. Una vez hecho esto se elaboran las etiquetas para los sobres a enviar. Una vez que se termina de imprimir el folleto, se le pega a cada uno una etiqueta, se organizan por código postal y se llevan a la oficina de correos.

Todas estas actividades toman un cierto tiempo; algunas de ellas se pueden llevar a cabo de manera simultánea, pero otras tienen que esperar a que actividades previas se terminen. La SMCC quiere saber cuál es el tiempo que requiere para preparar el encuentro, para saber cuál es la fecha más tarde en la que pueden empezar las distintas tareas.

Para calcular el tiempo total del proyecto necesitamos dos tipos de información: el tiempo, en días, que se toma cada actividad, y las actividades que tienen que estar terminadas para que ésta se pueda llevar a cabo. En la tabla 5.1 se encuentra esa información.

Este problema se presta para modelarlo con una *gráfica dirigida* (o *digráfica*). Una digráfica es un conjunto de vértices o nodos y una relación entre ellos, que llamamos los arcos de la digráfica. Si u y v son nodos de la digráfica, decimos que (u, v) ($u \rightarrow v$) es un arco que sale del nodo u y llega al nodo v y que se representa de alguna de las siguientes formas:

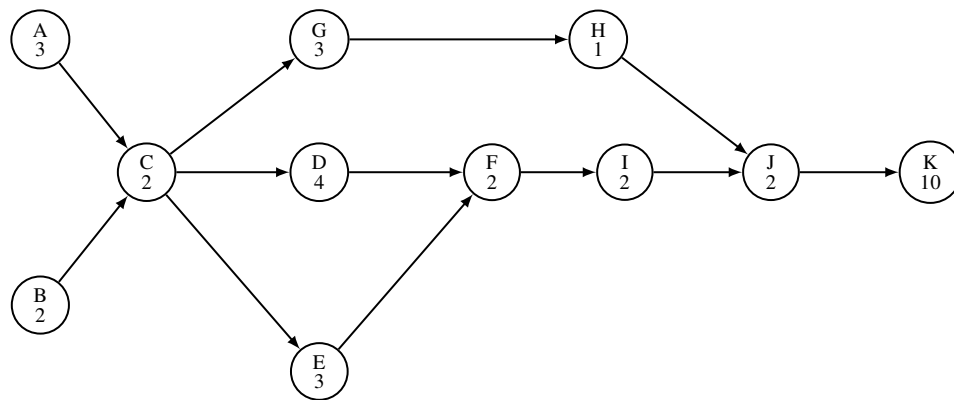


En nuestro problema particular, el arco (u, v) significa que la actividad u se tiene que terminar antes de que inicie la actividad v .

En cada vértice colocaremos el tiempo requerido para que la actividad se lleve a cabo, además del identificador de la actividad. El resultado se puede observar en la figura 5.1.

Tabla 5.1 Tiempos requeridos y predecesores por actividad

<i>Id.</i>	<i>Tarea</i>	<i>Tiempo</i>	<i>Tareas precedentes</i>
A.	Elegir temas.	3	ninguna
B.	Elegir árbitros	2	ninguna
C.	Elegir conferencias magistrales	2	A y B
D.	Dibujos alusivos	4	C
E.	Redacción de resúmenes	3	C
F.	Elaborar el folleto	2	D, E
G.	Elaborar la lista de correo	3	C
H.	Imprimir las etiquetas	1	G
I.	Imprimir el folleto	5	F
J.	Pegar las etiquetas	2	H, I
K.	Repartir la propaganda	10	J

Figura 5.1 Digráfica que corresponde a la organización del encuentro

Tratemos de calcular cuál es el mínimo tiempo requerido para ejecutar todas las tareas y terminar lo antes posible.

Si simplemente tomamos la suma de los tiempos que se requieren para terminar todas las tareas, obtendremos un tiempo total de 47 días. Sin embargo, muchas de esas tareas se pueden hacer de manera simultánea. ¿Cómo determinar, entonces, cuál es el menor tiempo en el que todas las tareas pueden ser completadas?

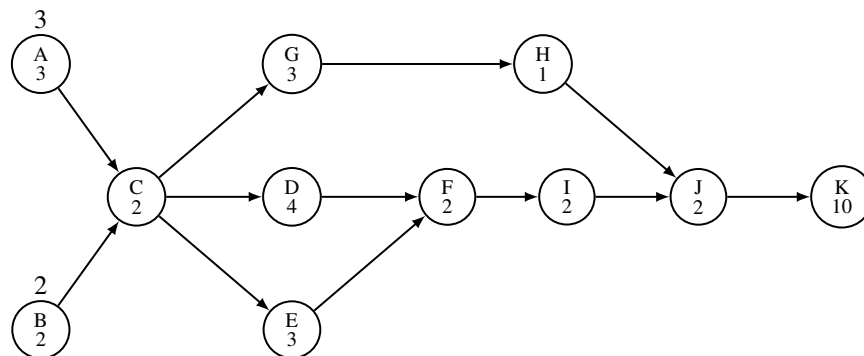
En 1958 se desarrolló una técnica, llamada *PERT (Program Evaluation and Review Technique)*, que calcula, entre otras cosas, lo que se conoce como *ruta crítica*. Una *ruta crítica* consiste de ordenar en el tiempo los eventos que se tienen que llevar a cabo y encontrar una “programación” de dichos eventos, de tal manera que se desarrollen todos ellos en el menor tiempo posible, respetando las precedencias especificadas. El algoritmo para ruta crítica es muy sencillo y consiste de lo siguiente.

1. Programar a todas las tareas que no tienen ninguna tarea que las preceda. Esto quiere decir asignarles como tiempo de programación (o terminación) igual al tiempo que se lleva la tarea.
2. Mientras queden tareas por programar:
 - a) La tarea t se programa si todas las tareas que preceden a t ya están programadas.
 - b) Su tiempo de programación se calcula como la suma del tiempo que se lleva la tarea t más el máximo tiempo de programación de entre las tareas que la preceden.
3. El tiempo total mínimo que se lleva el proceso es el máximo asignado a cualquiera de los nodos.

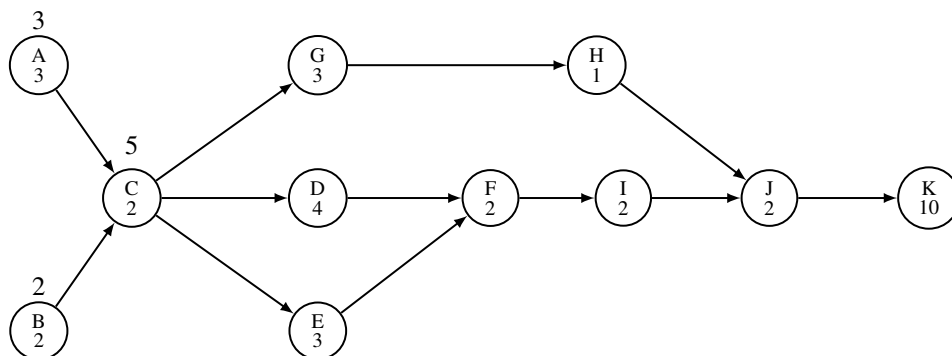
Si trabajamos sobre la digráfica de la figura 5.1, podemos ver la ejecución de cada uno de los pasos. Colocamos encima del nodo el valor asignado para la programación.

El Paso 1 nos indica que localicemos a las tareas que no tienen predecesores, y que son la tarea A y la B . Procedemos a asignarles como tiempo de programación el que corresponde a su tiempo de ejecución.

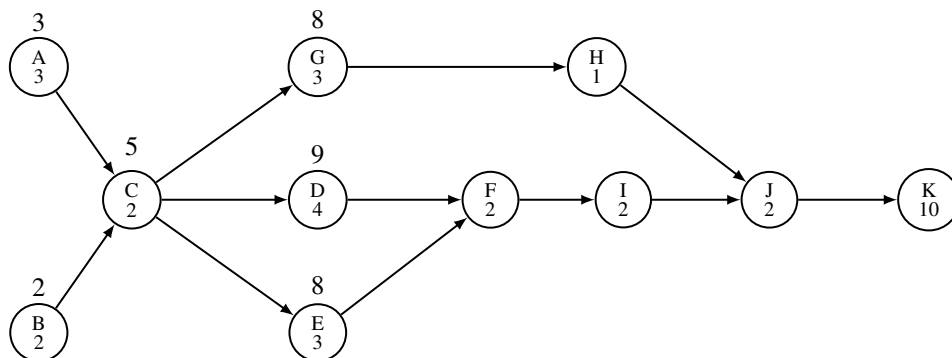
Figura 5.2 Programación de los eventos A y B de la digráfica 5.1



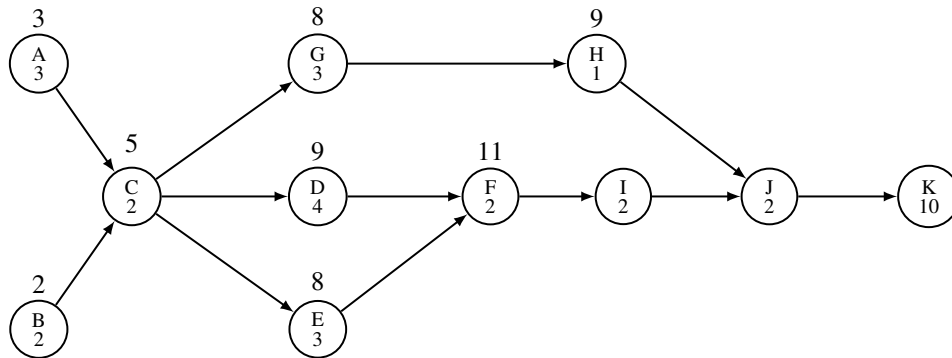
Al entrar a la iteración del algoritmo, el único nodo que cumple con las condiciones dadas es el nodo etiquetado con C , al que le corresponde el valor de 5, que es el máximo para programación de sus predecesores (3) sumado al tiempo que toma C (2).

Figura 5.3 Programación del evento *C* de la digráfica

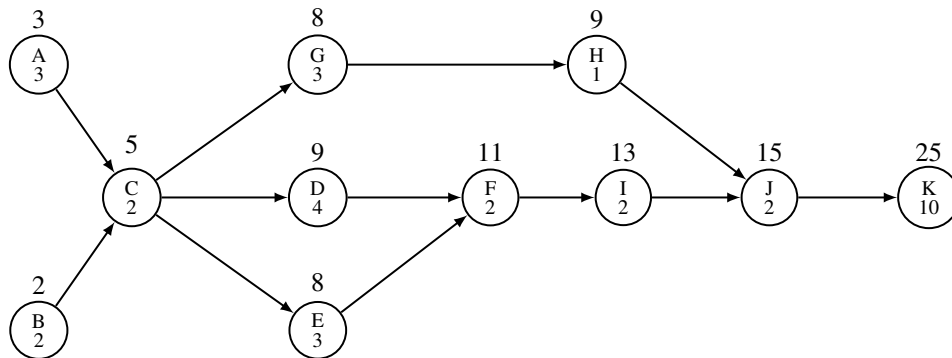
Como *C* es el único predecesor de *G*, *D* y *H*, estamos en condiciones de resolver el tiempo requerido, desde el inicio del proyecto, para estas tres tareas:

Figura 5.4 Programación de los eventos *D*, *G* y *E* de la digráfica

La siguiente “capa” es la que corresponde a las tareas *H* y *F*, cuyos antecesores ya están resueltos:

Figura 5.5 Programación de los eventos F y H de la digráfica

A continuación resolvemos el evento I , al que le asignamos un valor de 15, y con este evento, y con el evento H podemos resolver el evento J y a continuación el evento K . Los tiempos quedan asignados como se muestra en la figura 5.6.

Figura 5.6 Programación del resto de los eventos de la digráfica

De manera similar podemos obtener el tiempo de inicio de cada proyecto mediante el siguiente algoritmo:

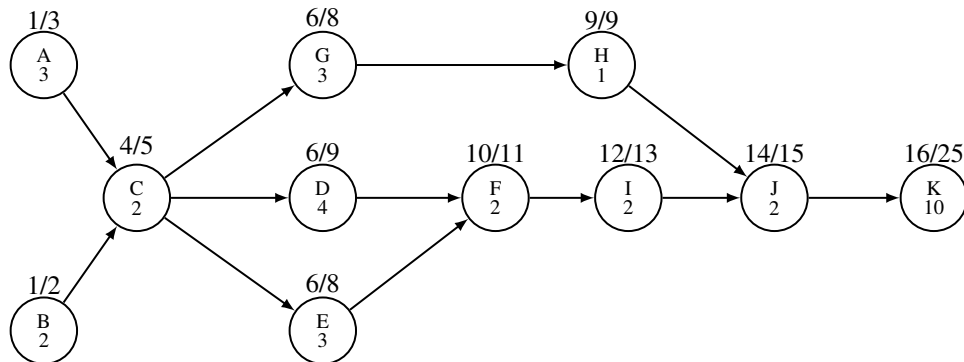
1. Marcar con 1 a todas las tareas que no tienen otra tarea que las preceda.
2. Se asigna tiempo de inicio a la tarea t si todas las tareas que la preceden tienen ya tiempo de inicio asignado, usando la siguiente fórmula:

$$Inicio(t) = \max\{Inicio(p) + Tiempo(p)\} \quad \text{con } p \text{ predecesor de } t$$

3. Termina cuando no haya tareas sin tiempo inicial asignado.

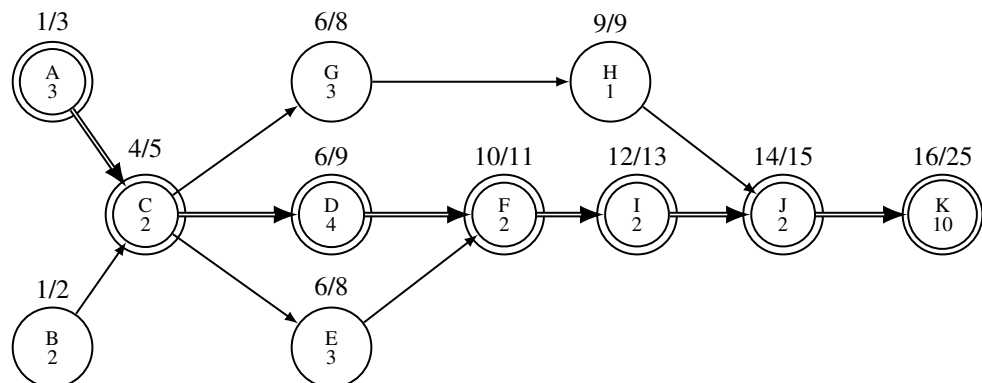
Colocaremos el tiempo inicial antes del tiempo final, separándolos con una diagonal, en la figura 5.7.

Figura 5.7 Tiempos de inicio/fin de cada tarea



A continuación queremos obtener aquellas tareas que son *críticas* para el proyecto, esto es, que deben iniciar exactamente en cuanto sus predecesores terminan – de acá viene el nombre de *ruta crítica* para este problema. Para ello, procedemos de la última tarea, la que le da el tiempo total al proyecto, y regresamos marcando a las tareas que aportaron el máximo. Conforme vamos regresando, marcamos los arcos de la digráfica con línea doble, para identificar la ruta, y los nodos también con línea doble. El resultado se puede ver en la figura 5.8.

Figura 5.8 Una de las rutas críticas del proyecto

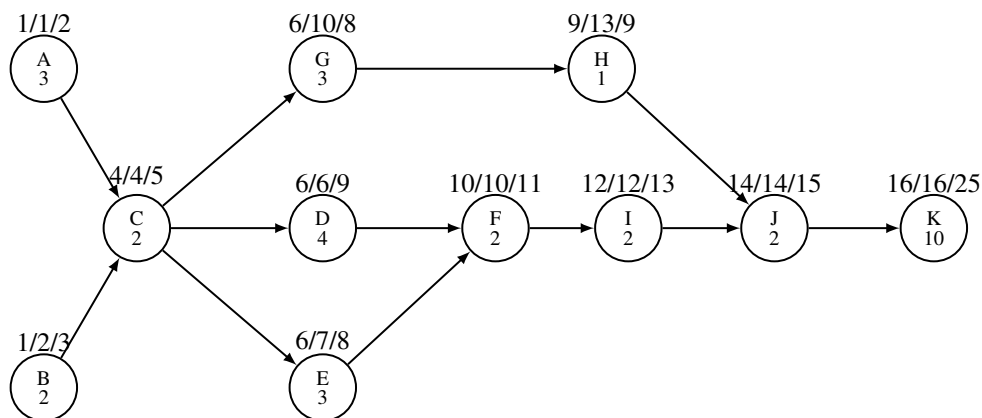


Otra información que podemos sacar de esta digráfica es aquellas actividades que tienen tiempo “de sobra” para empezar sus actividades. Es claro que las actividades que están sobre la ruta crítica no pueden perder tiempo y deben empezar exactamente cuando se les tiene programadas, pues son las que más tiempo se llevan. Pero aquellas actividades que no están en la ruta crítica pudiesen tener alguna holgura en su tiempo de inicio. Por ejemplo, la actividad *B* podría no empezar el primer día, sino hacerlo hasta el segundo sin retrasar el tiempo final del proyecto. La actividad *H*, asimismo, pudiera empezar el noveno día o bien esperarse hasta el día 13, y con eso estar lista en el día 14, que es cuando la actividad *J* necesita que la actividad *H* esté terminada. Este tiempo de holgura lo podemos calcular como el menor tiempo de inicio de sus sucesores. El cálculo empieza en las tareas que no tienen sucesores y va hacia atrás. En la figura 5.9 se muestra a continuación del tiempo de inicio, mediante el siguiente algoritmo:

1. Calcula el máximo tiempo de inicio de las tareas que no tienen sucesores como el tiempo de inicio que tienen ya marcado.
2. El máximo tiempo de inicio de una tarea que tiene sucesores se calcula como:

$$mInicio(t) = \min\{Inicio(s) - Tiempo(t)\} \quad \text{con } s \text{ tarea sucesora de } p$$

Figura 5.9 Tiempos de inicio/último-inicio/fin de cada tarea



En la tabla 5.2 en la siguiente página ponemos el último día en que cada actividad puede empezar.

Tabla 5.2 Margen de tiempos para el inicio de las actividades

<i>Id.</i>	<i>Tarea</i>	<i>Tiempo</i>	<i>Tareas precedentes</i>	<i>Día 1</i>	<i>Día últ.</i>
A.	Elegir temas.	3	ninguna	1	1
B.	Elegir árbitros	2	ninguna	1	2
C.	Elegir conferencias magistrales	2	A y B	4	4
D.	Dibujos alusivos	4	C	6	6
E.	Redacción de resúmenes	3	C	6	7
F.	Elaborar el folleto	2	D y E	10	10
G.	Elaborar la lista de correo	3	C	6	10
H.	Imprimir las etiquetas	1	G	9	13
I.	Imprimir el folleto	2	F	12	12
J.	Pegar las etiquetas	2	H, I	14	14
K.	Repartir la propaganda	10	J	16	16

5.1.2. Asignación óptima de recursos

Supongamos que tenemos una aerolínea que vuela a 7 distintas ciudades y cuenta con 7 pilotos. Cada piloto tiene sus preferencias respecto a dónde quiere volar. ¿Cuál es la manera de asignar a cada piloto a un vuelo, de tal manera que todos los pilotos vuelen a alguna ciudad de su preferencia? A este problema se le conoce con el nombre de *apareamiento perfecto* o *acoplamiento perfecto* (*perfect matching* en inglés) y se representa también con una gráfica que se conoce como bipartita.

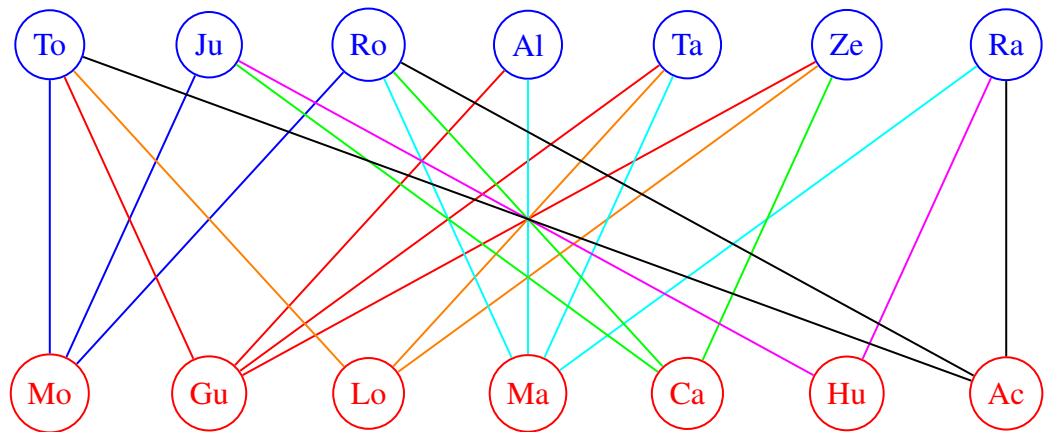
Veamos las preferencias de los pilotos respecto a las ciudades a las que quieren volar en la tabla 5.3.

Tabla 5.3 Relación entre ciudades y pilotos

<i>Ciudad</i>	<i>Pilotos:</i>			
Monterrey	Torres,	Juárez,	Robles	
Guadalajara	Albarrán,	Torres,	Tamariz,	Zepeda
Londres	Torres,	Tamariz,	Zepeda	
Manzanillo	Albarrán,	Tamariz,	Robles,	Ramírez
Cancún	Juárez,	Zepeda,	Robles	
Huatulco	Juárez,	Ramírez		
Acapulco	Torres,	Robles,	Ramírez	

Una gráfica bipartita es aquella en la que los vértices están partidos en dos conjuntos ajenos y las aristas van siempre de vértices en un conjunto a vértices en el otro. En el caso de acoplamiento, es natural que un conjunto sea, por ejemplo, el de los pilotos, mientras que el otro conjunto es el de las ciudades a las que deben volar. La gráfica que corresponde a la tabla 5.3 se encuentra en la figura 5.10

Figura 5.10 Gráfica bipartita que corresponde a la tabla 5.3



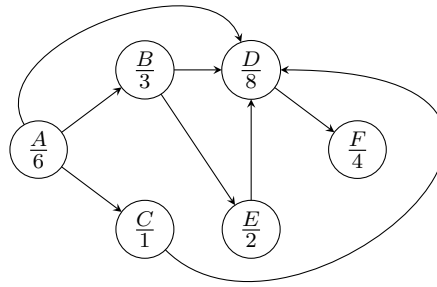
No todos los problemas de asignación óptima de recursos tienen solución. Por ejemplo, si se limita a que cada elemento de una de las partes quede relacionado únicamente con sólo uno de la otra parte, empezamos por pedir que el número de vértices en una partición sea el mismo que en la otra. Aun así podría no tener solución.

Un algoritmo nos dice que tratemos de encontrar un camino que toque a cada vértice exactamente una vez y da otras pistas de dónde iniciar y cuáles son las características que debe tener la gráfica bipartita para intentar armar el camino.

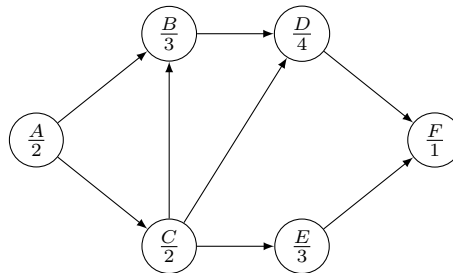
Habiendo dado ya una idea de por qué las gráficas son un mecanismo de modelado útil, pasamos a explorar este concepto.

Ejercicios

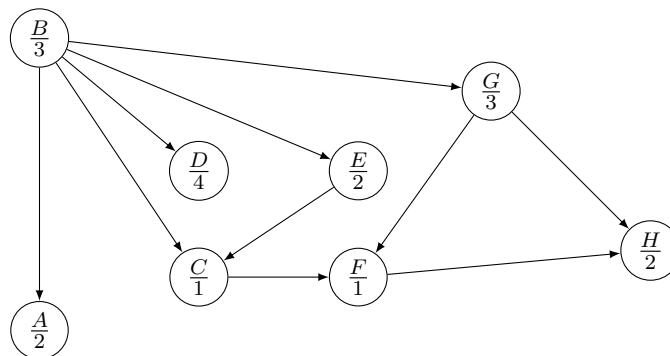
5.1.1.- Encuentra la ruta crítica en la siguiente digráfica. Para ello, elabora la tabla de las dependencias dadas por la digráfica y anota en dicha tabla el tiempo de inicio, el tiempo final y el tiempo disponible de holgura (*slack* en inglés) para cada una de las actividades del proyecto. Indica también cuál es el evento que se puede llevar a cabo el primero y cuál es el último evento en llevarse a cabo.



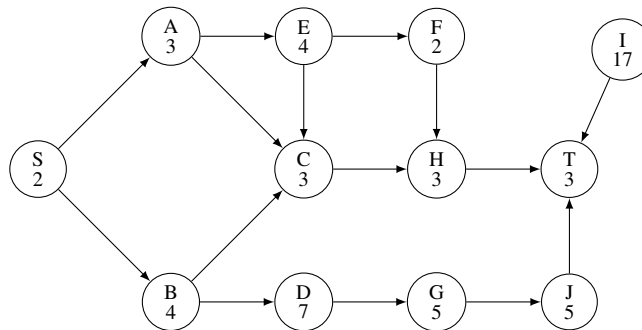
5.1.2.- Encuentra la ruta crítica en la siguiente digráfica. Para ello, elabora la tabla de las dependencias dadas por la digráfica y anota en dicha tabla el tiempo de inicio, el tiempo final y el tiempo disponible de holgura para cada una de las actividades del proyectos. Indica también cuál es el evento que se puede llevar a cabo el primero y cuál es el último evento en llevarse a cabo?



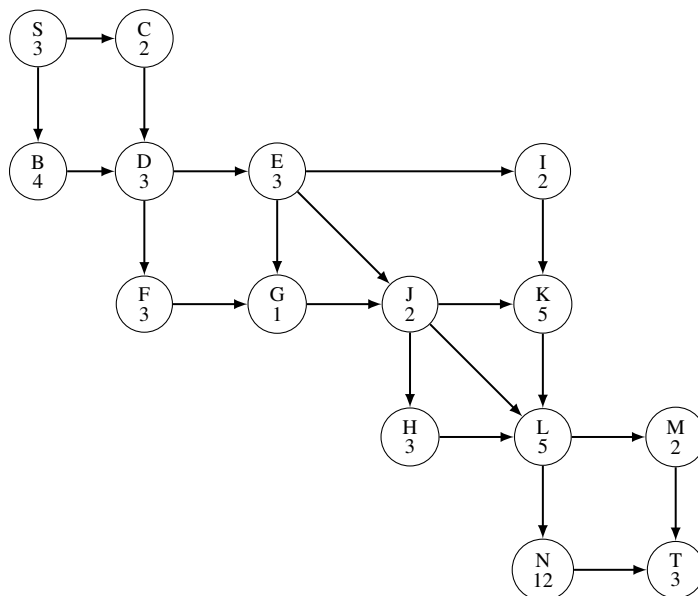
5.1.3.- Encuentra la ruta crítica en la siguiente digráfica. Para ello, elabora la tabla de las dependencias dadas por la digráfica y anota en dicha tabla el tiempo de inicio, el tiempo final y el tiempo disponible de holgura para cada una de las actividades del proyectos. Indica también cuál es el evento que se puede llevar a cabo el primero y cuál es el último evento en llevarse a cabo?



5.1.4.- Encuentra la ruta crítica en la siguiente digráfica. Para ello, elabora la tabla de las dependencias dadas por la digráfica y anota en dicha tabla el tiempo de inicio, el tiempo final y el tiempo disponible de holgura para cada una de las actividades del proyectos. Indica también cuál es el evento que se puede llevar a cabo el primero y cuál es el último evento en llevarse a cabo?



5.1.5.- Encuentra la ruta crítica en la siguiente digráfica. Para ello, elabora la tabla de las dependencias dadas por la digráfica y anota en dicha tabla el tiempo de inicio, el tiempo final y el tiempo disponible de holgura para cada una de las actividades del proyectos. Indica también cuál es el evento que se puede llevar a cabo el primero y cuál es el último evento en llevarse a cabo?



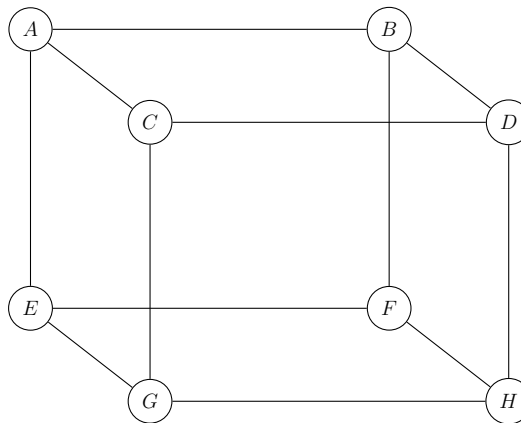
5.1.6.- En un taller de herrería se cuenta con dos máquinas de tipo A, una máquina de tipo B y tres máquinas de tipo C. El taller cuenta con 7 empleados con distinto manejo de cada una de las máquinas, como se muestra en la tabla siguiente:

Empleado Num.	Sabe usar máquina(s):
1	A
2	A,C
3	C
4	A,B
5	A,B,C
6	B
7	B,C

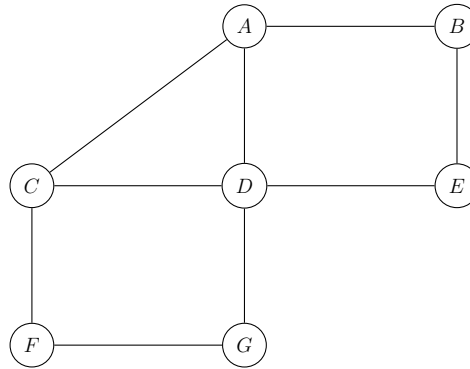
Construye la gráfica bipartita correspondiente a la tabla anterior.

5.1.7.- Da una asignación posible para el problema anterior que ocupe el mayor número de máquinas, un trabajador por máquina. ¿Cuántos turnos tendrían que cubrirse para que todos los trabajadores trabajen el mismo número de turnos en las máquinas? Da la organización necesaria para que esto último suceda.

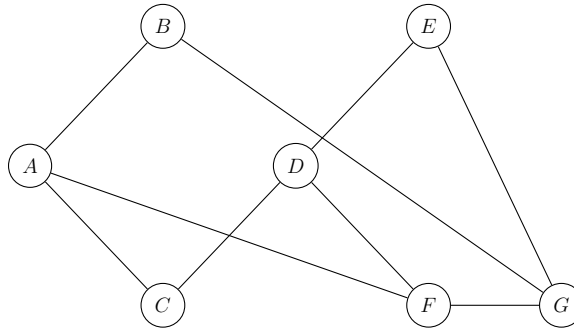
5.1.8.- La siguiente gráfica es bipartita, aunque no se note. Dibújala de tal manera que se vea a simple vista que es bipartita.



5.1.9.- Dada la siguiente gráfica, argumenta por qué NO es bipartita.



5.1.10.- Dada la siguiente gráfica, decide si es bipartita o no. Si es bipartita, da la partición de los vértices y si no lo es, explica por qué esa partición no se puede dar.



5.2. Conceptos y formalización

Como ya vimos, las gráficas nos sirven para modelar un sinnúmero de problemas de la vida real, como lo son la programación de eventos, el si se puede llegar de una ciudad a otra, los mapas, las relaciones presentes entre personas de un cierto grupo, la asignación de tareas, y muchos más. En este capítulo revisaremos con más cuidado el concepto matemático de gráfica y algunos de sus usos.

Definición 5.1 Una *gráfica* $G = (V, E)$ es una pareja que consiste de un conjunto no vacío de puntos V , llamados los *vértices* o *nodos* de la gráfica, y un segundo conjunto E de *aristas*, que corresponden a una relación de parejas entre los vértices, subconjunto de $V \times V$.

Los elementos de E reciben nombres, dependiendo de la forma que toma la pareja:

Definición 5.2 Una *arista* es una pareja no ordenada de vértices $e = uv$. Los vértices u y v son los extremos de la arista e y la representamos como $uv = vu$.

Definición 5.3 Un *lazo* es una arista cuyos extremos son el mismo vértice (uu).

Definición 5.4 Un *arco* es una pareja *ordenada*, que representamos como $u \rightarrow v$, \overrightarrow{uv} o, simplemente, (u, v) . En el caso que estemos trabajando con arcos, al subconjunto del producto cartesiano lo identificamos con A .

Definición 5.5 Decimos que una gráfica tiene *aristas múltiples* si es que hay más de dos aristas con los mismos extremos.

Podemos asignar nombres a las aristas o arcos. Dependiendo de las restricciones que pudiese tener E (o A), las gráficas se clasifican de la siguiente manera:

Definición 5.6 Una *gráfica simple* es aquella que no tiene aristas múltiples ni lazos.

Definición 5.7 Una *multigráfica* es aquella que permite aristas múltiples y lazos.

Definición 5.8 Una *gráfica dirigida* o *digráfica* es aquella que se define como $G = (V, A)$, donde el subconjunto del producto cartesiano $V \times V$ corresponde a arcos; en este caso es claro que $(u, v) \neq (v, u)$.

Los términos *gráfica simple* o *multigráfica* se pueden aplicar a digráficas, sustituyendo simplemente a las aristas por arcos. Utilizaremos el término genérico de *gráfica* cuando no distingamos entre gráficas simples o multigráficas; en el mismo sentido utilizaremos *digráficas* cuando hablemos de los distintos tipos de gráficas dirigidas.

Solemos representar a las gráficas pintando un punto o círculo pequeño por cada vértice y una línea que une a dos de éstos para las aristas. Podemos ver distintas representaciones de gráficas en la figura 5.11. En ella se muestran tres de las posibles variaciones que podemos darle a la representación, que incluyen nombrar o no a las aristas o vértices de la gráfica y la manera como elegimos representar a los vértices.

Figura 5.11 Algunas representaciones con figuras (visuales) de gráficas

(1/2)

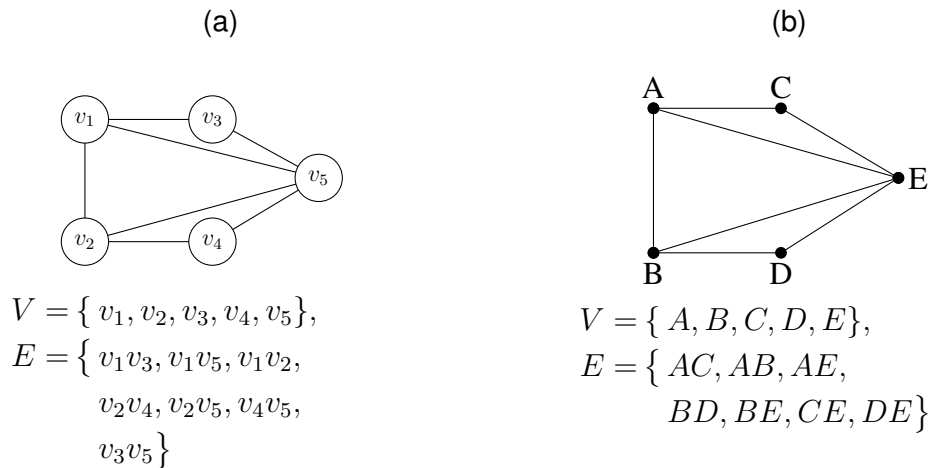
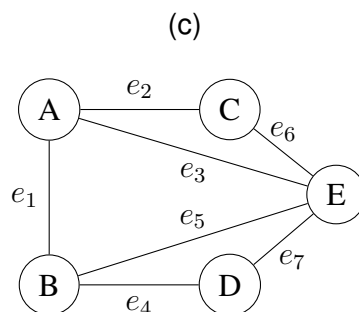


Figura 5.11 Algunas representaciones con figuras (visuales) de gráficas

(2/2)



$$V = \{A, B, C, D, E\},$$

$$E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$$

El que la pareja sea *no ordenada* quiere decir que las parejas uv y vu representan a la misma arista, pues aparecen los mismos vértices, no importa en qué orden. En la figura 5.11, tenemos a las aristas sin nombre en las gráficas de las figuras 5.11(a) y 5.11(b), mientras que las aristas en la subfigura 5.11(c) sí tienen nombre (en los tres casos están listadas abajo de la gráfica).

Definición 5.9 Dos vértices u y v son *adyacentes* si la arista $e = uv \in E$; dicho de otra manera, si existe una arista en E cuyos extremos son u y v .

En las gráficas de la figura 5.11 los vértices A y B , por ejemplo, son adyacentes, lo mismo que los vértices A y C ; y A y E . Los vértices B y C , en cambio, no son adyacentes ya que no hay ninguna arista que los conecte.

Definición 5.10 Una arista e es *incidente* en un vértice v si v es uno de los extremos de la arista.

En la gráfica de la figura 5.11(c) la arista e_4 es incidente en los vértices B y D .

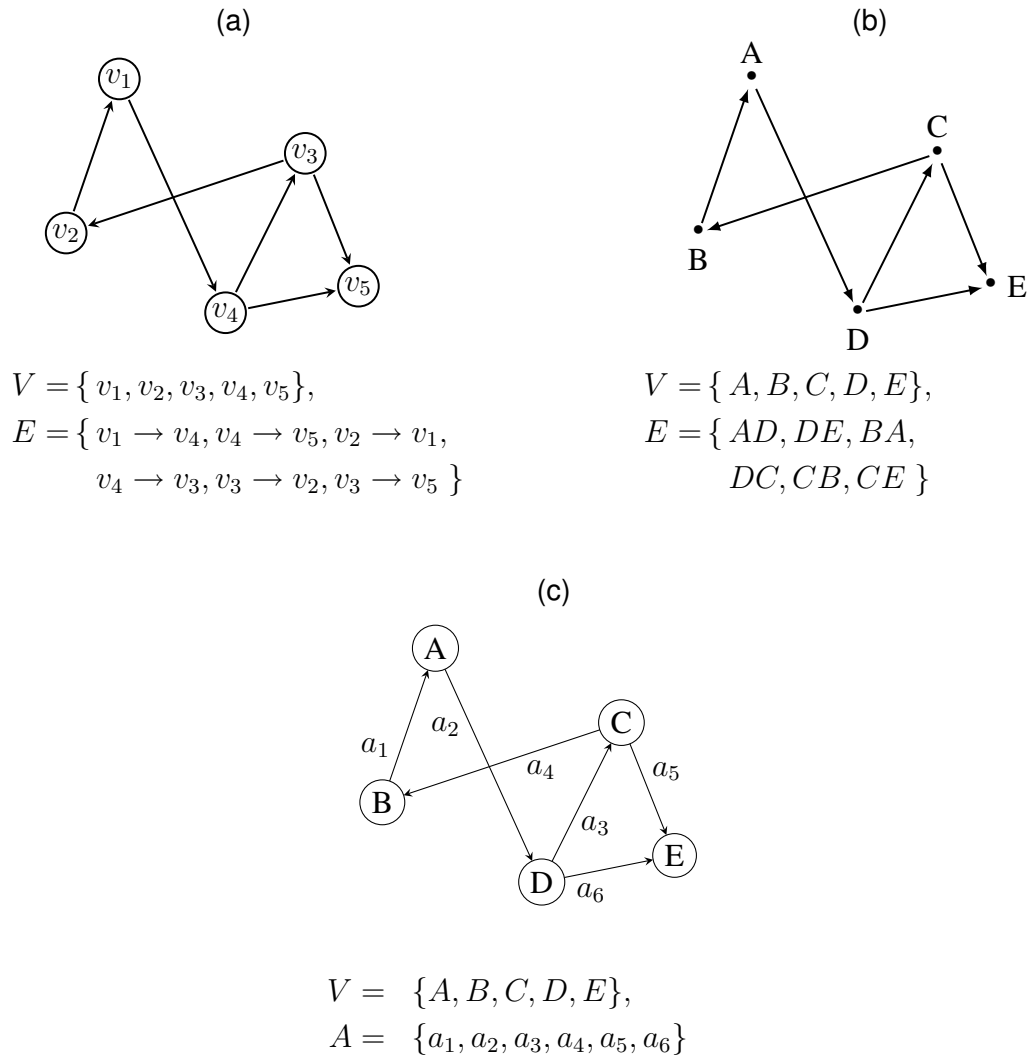
En la figura 5.12 presentamos distintas maneras de dibujar una digráfica. En el caso de las digráficas los arcos se representan en los dibujos como flechas con la dirección deseada.

La relación entre aristas y vértices está dada por el *grado* de un vértice:

Definición 5.11 El *grado de un vértice*, denotado por $\text{grado}(v)$, es el número de aristas incidentes en el vértice.

En la gráfica de la figura 5.11(c) el grado del vértice A es 3, que corresponde a las aristas e_1 , e_2 y e_3 , mientras que el grado del vértice D es 2 que corresponde a las aristas e_4 y e_7 .

En digráficas hablamos del $\text{exgrado}(v)$ e $\text{ingrado}(v)$ que corresponde, respectivamente, al número de arcos que salen de v y al número de arcos que entran a v .

Figura 5.12 Representación de gráficas dirigidas

El número de aristas de una gráfica está íntimamente relacionado con el grado de los vértices. Esta relación se presenta en el teorema 5.1.

Teorema 5.1 *En una gráfica, la suma de los grados de los vértices es igual a dos veces el número de aristas:*

$$\sum_{v \in V} \text{grado}(v) = 2 \cdot |E|$$

donde $|E|$ denota al número de elementos de E , la cardinalidad de E .

Demostración. Cada arista incide en dos vértices, por lo que si se cuentan los vértices en los que inciden las aristas ($\sum \text{grados}$), cada arista será contada dos veces. \square

Otra demostración de este teorema, y que utiliza la inducción estructural que ya vimos y que es muy común en teoría de gráficas, se plantea como sigue:

- (a) Demostramos primero el caso para una gráfica con 1 vértice y 0 aristas (ver en la figura 5.13 el caso $G = (\{v\}, \emptyset)$). El grado de este vértice es 0; el número de aristas también es 0, por lo que se cumple

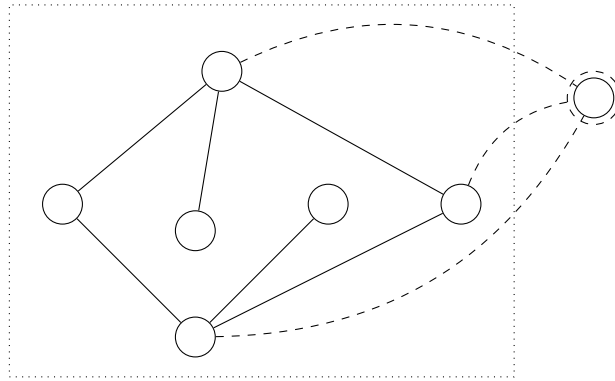
$$0 = 2 \cdot |E| = \sum_{v \in V} \text{grado}(v) = 0$$

Figura 5.13 $G = (\{v\}, \emptyset)$



- (b) Observemos una gráfica con $n \geq 2$ vértices. Quitémosle un vértice (con todas las aristas que inciden en él) – ver figura 5.14 –.

Figura 5.14 Inducción estructural en gráficas



La gráfica resultante (encuadrada con línea punteada) cumple con la hipótesis de inducción de que la suma de los grados de los vértices es dos veces el número de aristas. Repongamos el vértice que quitamos, junto con sus aristas. Por cada arista que agregamos, dos vértices ven incrementado sus grados en una unidad – el que estamos reinsertando y aquel al que llega la arista –. Supongamos que el grado del vértice reinsertado es k . Entonces tenemos

$$\sum_{i=1}^n \text{grado}(v_i) = \left(\sum_{i=1}^{n-1} \text{grado}(v_i) \right) + 2 \cdot k = 2(|E| - k) + 2 \cdot k = 2 \cdot |E|$$

Teorema 5.2 En una gráfica cualquiera, existe un número par de vértices de grado impar.

Demostración. Por el teorema anterior sabemos que $\sum_{v \in V} \text{grado}(v) = 2|E|$. Separemos esta suma entre los vértices de grado par y los de grado impar:

$$\begin{aligned} \text{Sea } V_p &= \{v \in V \mid \text{grado}(v) \text{ es par}\} \\ \text{y } V_i &= \{v \in V \mid \text{grado}(v) \text{ es impar}\} \end{aligned}$$

Entonces

$$\sum_{v \in V} \text{grado}(v) = \sum_{v \in V_p} \text{grado}(v) + \sum_{v \in V_i} \text{grado}(v) = 2|E|$$

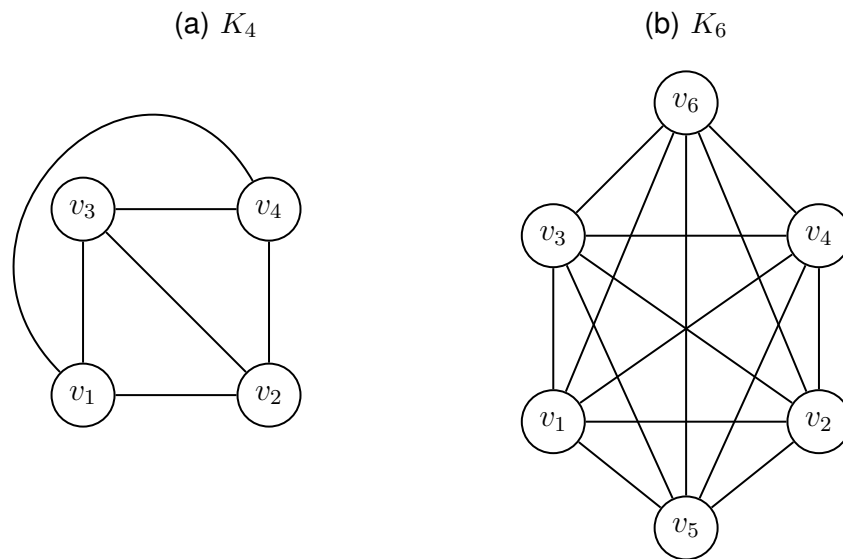
La suma total es par; la suma de pares es par (de la primera parte) y, para que el resultado sea par, la segunda parte también tiene que ser par. Si el número de vértices de grado impar fuera impar, el resultado sería impar, por lo que el número de vértices de grado impar tiene que ser par. \square

Tenemos ciertos tipos de gráficas que tienen una determinada relación entre sus vértices y sus aristas.

Definición 5.12 Una gráfica completa con n vértices, denotada por K_n , es aquella donde cada vértice es adyacente a cualquier otro vértice de la misma gráfica.

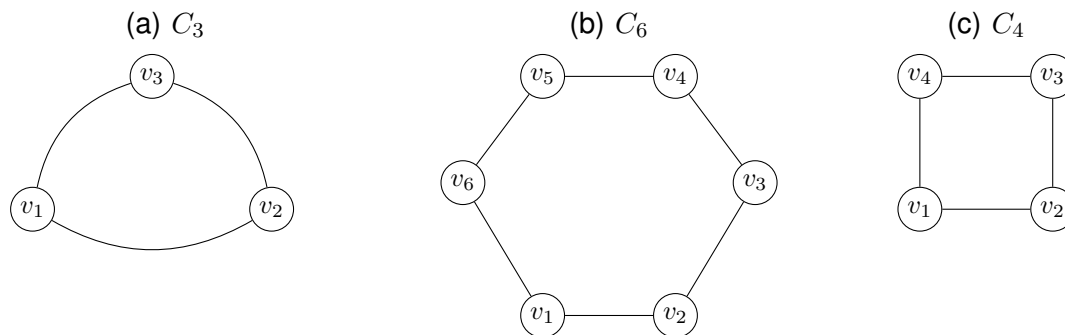
En la figura 5.15 vemos algunos ejemplos de gráficas completas.

Figura 5.15 Ejemplos de gráficas completas



Otro tipo de gráficas distinguidas son aquellas que forman una especie de anillo, donde cada vértice tiene grado 2 y se puede llegar de cualquier vértice a cualquier otro simplemente recorriendo el anillo. A estas gráficas se les conoce como C_n , donde la C está por “ciclo” y la n denota el número de vértices. En la figura 5.16 se muestran varias de estas gráficas.

Figura 5.16 Ejemplos de ciclos



Las gráficas C_n presentan también algunas propiedades, como son que el grado de todos sus vértices es 2 y que el número de aristas en una gráfica C_n es n .

Cuando definamos lo que es un camino entre dos vértices regresaremos brevemente a las gráficas C_n .

Otro tipo de gráfica que ya vimos y que resulta muy importante para toda una clase de problemas, la optimización de asignación de recursos, son las gráficas *bipartitas*.

Definición 5.13 (Gráfica bipartita) Una *gráfica bipartita* es aquella en la que podemos partir al conjunto de los vértices en dos subconjuntos ajenos, de tal manera que todas las aristas van de vértices de un conjunto a vértices del otro subconjunto (esto es, no hay aristas entre los vértices de un mismo subconjunto).

De esta definición queda claro que no importa cómo se dibuje la gráfica, siempre se puede redibujar de tal manera que tengamos dos hileras de vértices, una por cada subconjunto. En la figura 5.17 se muestran varias gráficas bipartitas.

Figura 5.17 Gráficas bipartitas

(1/2)

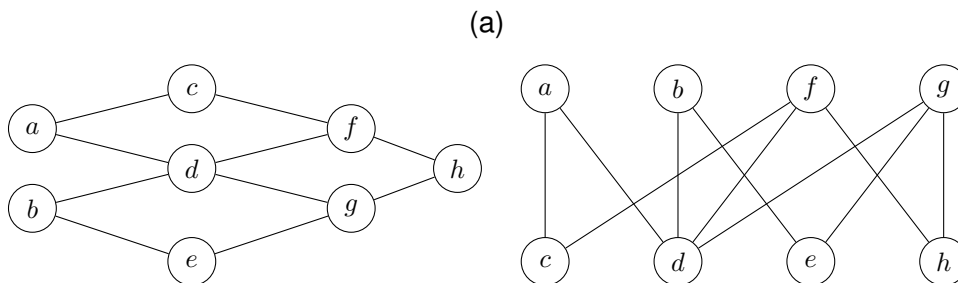
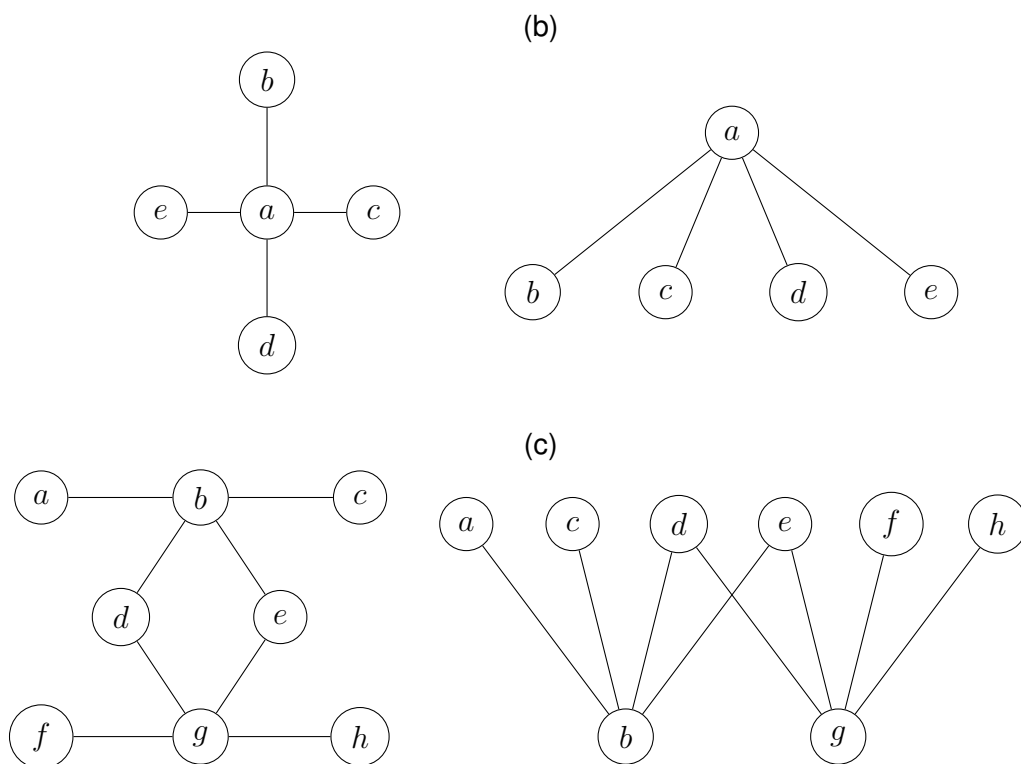


Figura 5.17 Gráficas bipartitas

(2/2)



También entre las gráficas bipartitas podemos tener gráficas completas y se representan como $K_{m,n}$, donde la m y la n nos denotan el número de vértices en cada subconjunto. En el caso de gráficas bipartitas, lo completo se refiere a que cada vértice de un subconjunto es adyacente a todos los vértices del otro subconjunto. En la figura 5.18 podemos ver algunas gráficas bipartitas completas.

Figura 5.18 Gráficas bipartitas completas

(1/2)

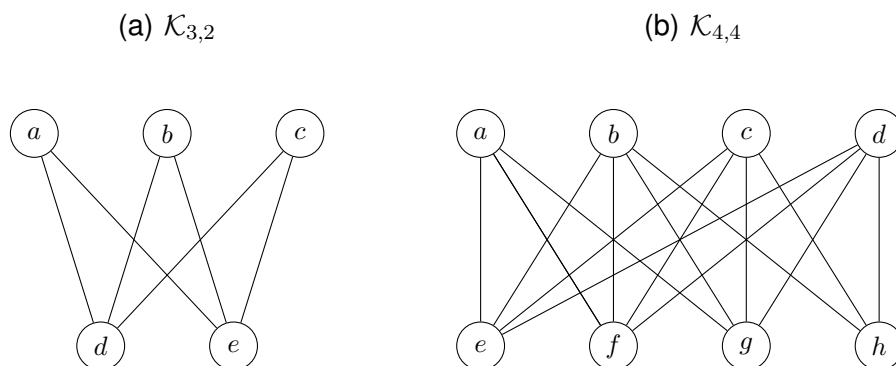
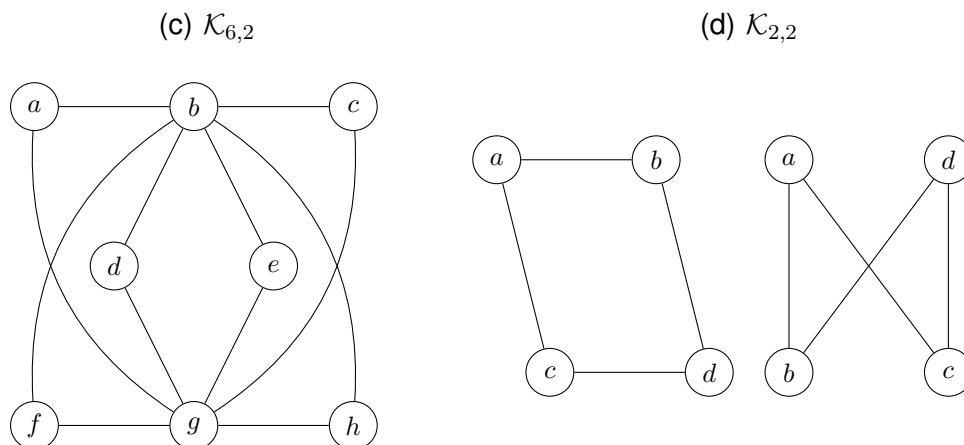


Figura 5.18 Gráficas bipartitas completas

(2/2)



Definimos también lo que es una *subgráfica*, concepto que utilizaremos ampliamente en el resto de este capítulo.

Definición 5.14 (Subgráfica) Sea $G = (V, E)$ una gráfica. Una *subgráfica* $G' = (V', E')$ de G es una gráfica tal que $V \subseteq V'$ y $E \subseteq E'$. En otras palabras, se eliminan cero o más vértices y cero o más aristas de la gráfica original. Entre las aristas que se eliminan tienen que estar aquellas aristas incidentes en los vértices eliminados.

Esto es, una subgráfica G' de una gráfica G es aquella que resulta de quitar de la gráfica original algunos vértices, junto con todas las aristas que inciden en estos vértices, y algunas aristas más. Ya utilizamos el concepto de subgráfica en la demostración por inducción estructural del teorema 5.1. En la figura 5.19 presentamos varias gráficas y a su derecha algunas de sus subgráficas.

Figura 5.19 Gráficas con algunas de sus subgráficas

(1/2)

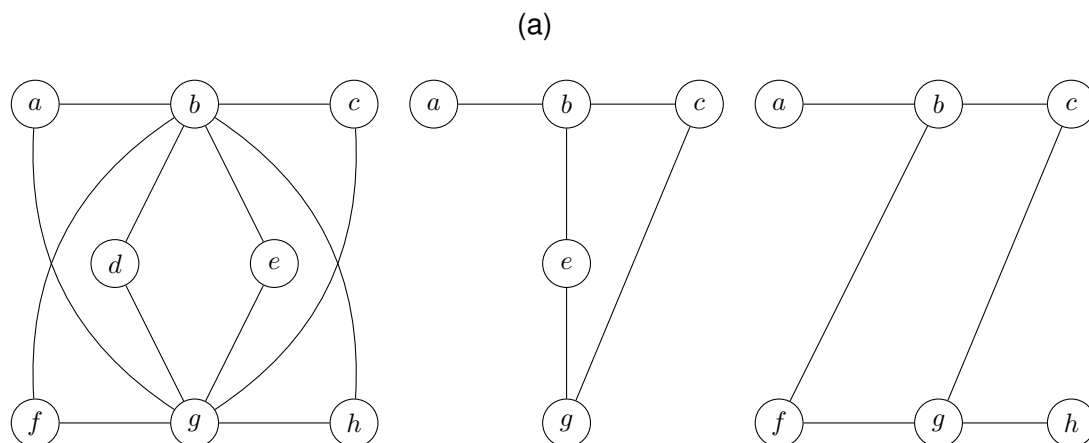
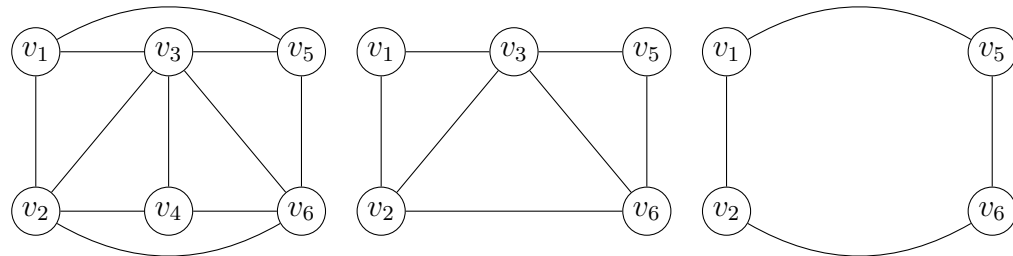


Figura 5.19 Gráficas con algunas de sus subgráficas

(2/2)

(a)



Nos van a interesar ciertos tipos de subgráficas, que definimos a continuación:

Definición 5.15 Una *subgráfica de una gráfica G inducida por un subconjunto de vértices A* es aquella que resulta de eliminar algunos vértices de la gráfica y, junto con esos vértices, eliminar exclusivamente las aristas incidentes en los vértices eliminados; se denota como $G[A]$ (o más raramente $G\langle A \rangle$). Similarmente, una *subgráfica inducida por un subconjunto de aristas* es aquella que resulta de eliminar algunas aristas y, junto con esas aristas, aquellos vértices que resultan no ser adyacentes a ningún otro vértice de la gráfica una vez eliminadas las aristas seleccionadas.

En la figura 5.20 presentamos varias gráficas y a su derecha algunas de sus subgráficas inducidas. En la subfigura 5.20(a) vemos subgráficas generadas por un subconjunto de los vértices originales. Se eliminan exclusivamente las aristas incidentes en los vértices eliminados. En la subfigura 5.20(a) vemos subgráficas inducidas por un subconjunto de aristas, donde se eliminan aquellos vértices que quedan sin ninguna arista incidente en ellos.

Figura 5.20 Gráficas con algunas de sus subgráficas inducidas

(1/2)

(a) Subgráficas inducidas por subconjuntos de vértices

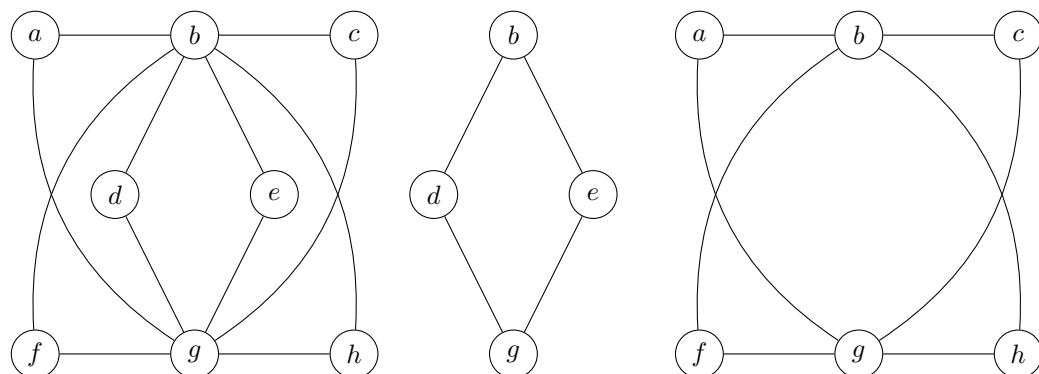
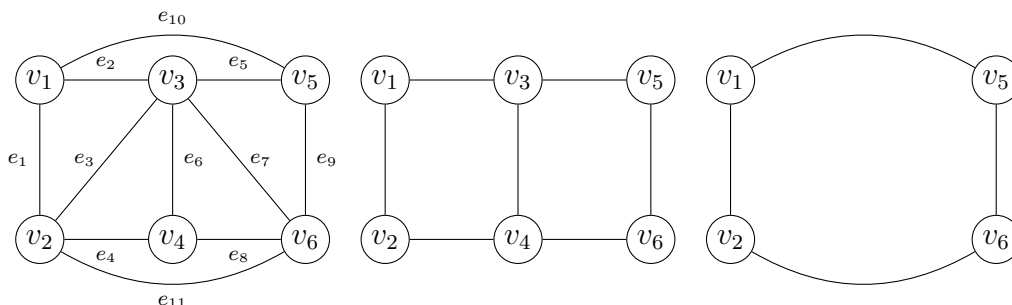


Figura 5.20 Gráficas con algunas de sus subgráficas inducidas

(2/2)

(a) Subgráficas inducidas por subconjuntos de aristas

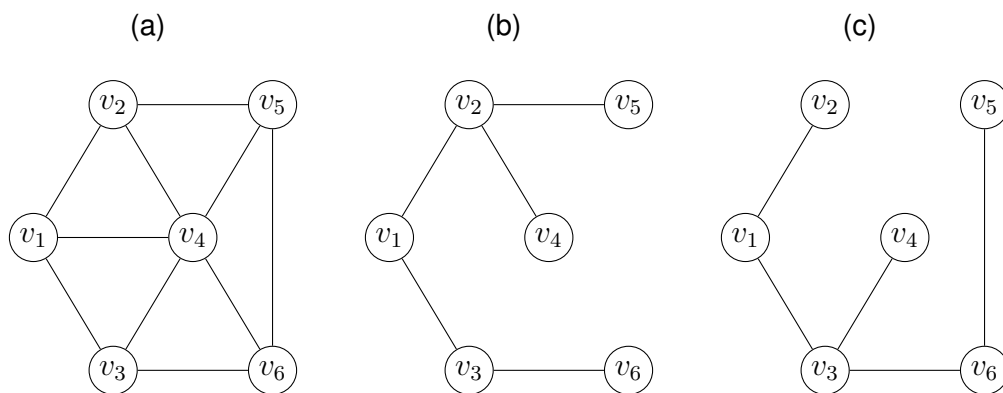


En la segunda gráfica de la subfigura 5.20(a), al eliminar a las aristas e_3 , e_6 , e_{10} y e_{11} , ningún vértice queda aislado, por lo que no se elimina ninguno de la subgráfica. En cambio, en la tercera gráfica de esta misma subfigura, al eliminar a todas las aristas incidentes en v_3 y v_4 , se eliminan también estos dos vértices.

Otro concepto importante relacionado con el de subgráficas es el de *subgráfica generadora*¹ que se define como sigue:

Definición 5.16 (Subgráfica generadora) Sea $G = (V, E)$ una gráfica no dirigida. Una *subgráfica generadora* es una subgráfica $G' = (V, E')$ de G tal que contiene a todos los vértices de G y $E' \subseteq E$.

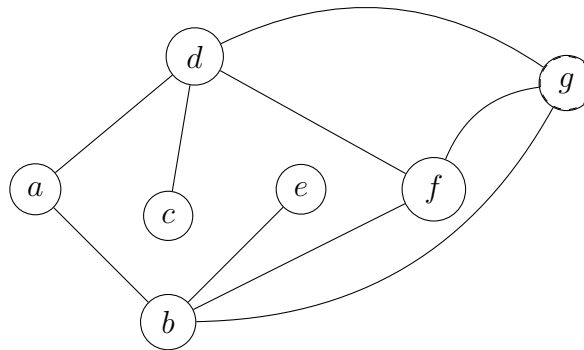
La idea con una subgráfica generadora es que estén presentes todos los vértices de la gráfica original. En la figura 5.21 podemos ver a una gráfica con dos de sus gráficas generadoras.

Figura 5.21 Gráfica con subgráficas generadoras

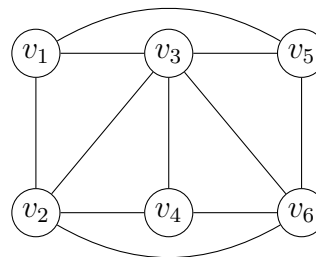
¹En inglés *spanning subgraph*.

Ejercicios

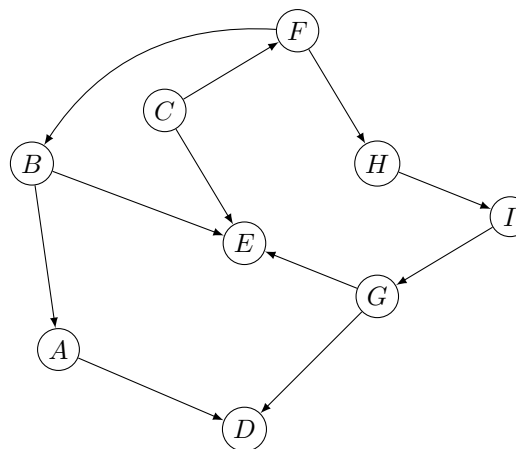
5.2.1.- Dada la siguiente gráfica, especifica cuáles son sus conjuntos V y E .



5.2.2.- Dada la siguiente gráfica, especifica cuáles son sus conjuntos V y E .



5.2.3.- Dada la siguiente gráfica dirigida, especifica cuáles son sus conjuntos V y A .



5.2.4.- Dados los siguientes conjuntos, dibuja la gráfica a la que corresponden.

$$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$$

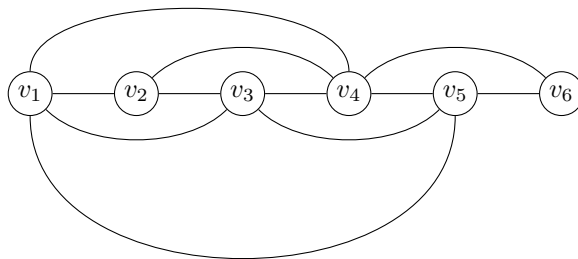
$$E = \{v_1v_2, v_2v_4, v_2v_5, v_2v_3, v_2v_6, \\ v_4v_3, v_6v_3\}$$

5.2.5.- Dibuja una gráfica tal que $|V| = 5$ y $|E| = 6$.

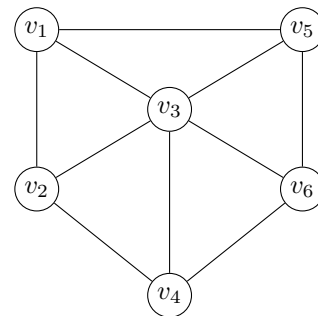
5.2.6.- Dibuja una gráfica dirigida tal que $|V| = 6$ y $|A| = 9$.

5.2.7.- Determina el grado de cada vértice en las siguientes gráficas:

(a)

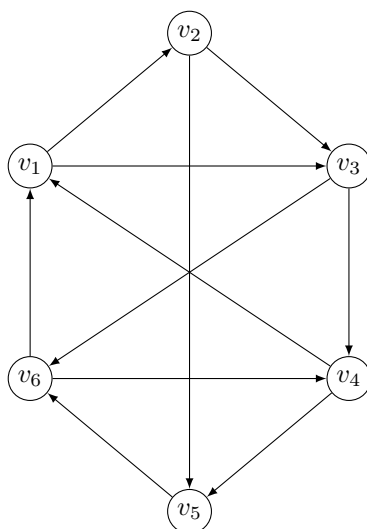


(b)

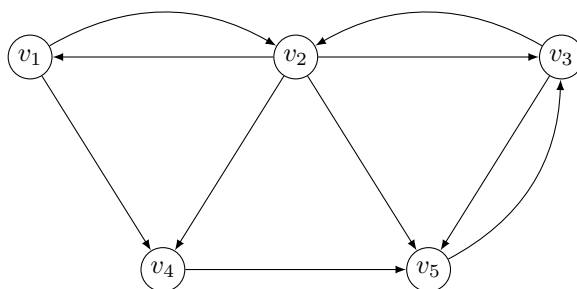


5.2.8.- Determina el exgrado e ingrado de cada vértice en las siguientes gráficas dirigidas:

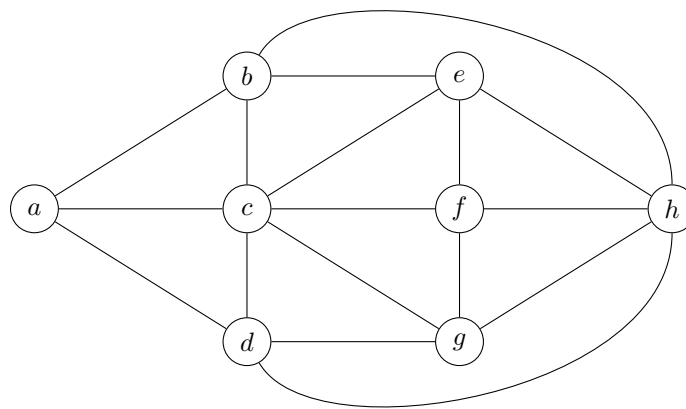
(a)



(b)



- 5.2.9.- Explica por qué no puede haber una gráfica con un número impar de vértices, cada uno de ellos con grado impar.
- 5.2.10.- Dibuja las gráficas que corresponden a K_2 , K_3 y K_5 .
- 5.2.11.- Usando la propiedad de que los vértices en las gráficas C_n , $n > 2$, tienen todos grado 2, demuestra que el número de aristas en una gráfica C_n es n .
- 5.2.12.- Dada la siguiente gráfica, dibuja dos subgráficas generadoras distintas.



5.3. Representación de gráficas para su manipulación

En esta sección trabajaremos únicamente con gráficas simples y nos referiremos a ellas simplemente como gráficas. Para poder manipular las gráficas desde alguna aplicación en una computadora tenemos que contar con alguna representación más descriptiva que la de los dibujos. Si bien los dibujos funcionan muy bien para el ojo humano, las computadoras no trabajan bien con imágenes “de conjunto”. La definición matemática de lo que es una gráfica resulta ser mucho más manejable. En esta sección presentaremos varias opciones de representación que se prestan bien para la manipulación de gráficas a través de algoritmos.

5.3.1. Matriz de adyacencias

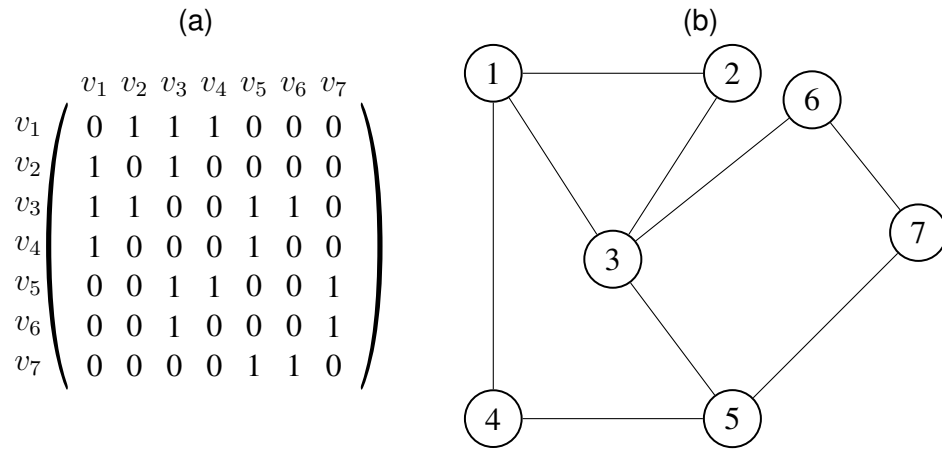
La matriz de adyacencias para una gráfica con n vértices es una matriz cuadrada de $n \times n$, tal que el elemento $m_{i,j}$ de la matriz, que denota al renglón i , columna j , está definido

de la siguiente manera:

$$m_{i,j} = \begin{cases} 1 & \text{si } e = v_i v_j \in E \\ 0 & \text{si } e = v_i v_j \notin E \end{cases}$$

Veamos en la figura 5.22 una gráfica con su correspondiente matriz de adyacencias.

Figura 5.22 Representación de gráficas con matriz de adyacencias



Esta representación particular presenta propiedades que pudiesen contestar rápidamente a preguntas que quisiésemos hacernos respecto a una gráfica particular.

Teorema 5.3 La suma de las entradas en el renglón i de una matriz de adyacencias corresponde al grado del vértice v_i .

Demostración. Observemos el renglón de cualquier vértice, digamos v_i . Cada 1 que aparece es porque el vértice de la columna j , v_j , es adyacente a él. Por lo tanto, el número de 1's en el renglón i indica el número de vértices distintos de v_i que son adyacentes a él. Como cada entrada vale 1, la suma de estas entradas es precisamente el número de vértices adyacentes a él, o sea $\text{grado}(v_i)$. \square

La matriz de adyacencias de una gráfica presenta las siguientes propiedades:

Definición 5.17 Decimos que una matriz es *cuadrada* si tiene el mismo número de renglones que de columnas.

Las matrices de adyacencias son cuadradas porque tienen un renglón para cada vértice y una columna para cada vértice, por lo que el número de renglones es el mismo que el número de columnas.

Definición 5.18 Decimos que una matriz es *simétrica* si

$$\forall i, j, 0 \leq i, j < |V|, m_{i,j} = m_{j,i}.$$

Para que una matriz sea simétrica debe ser cuadrada.

Definición 5.19 La *diagonal principal* de una matriz (cuadrada) es el vector que corresponde a los elementos $m_{i,i}$ de la matriz.

Como estamos trabajando con gráficas simples, la diagonal principal de una matriz de adyacencias es de ceros, ya que no tenemos ningún lazo en la gráfica. También es importante notar que las matrices de adyacencias son siempre simétricas: si tenemos una arista que va del vértice i al vértice j ($m_{i,j} = 1$), también tendremos a esa misma arista (recordemos que estamos trabajando con gráficas simples) que va del vértice j al vértice i (por lo que $m_{i,j} = m_{j,i} = 1$).

Es conveniente mencionar que, dada esta simetría en las matrices de adyacencias, el teorema 5.3 se cumple también si sustituimos “columna” donde dice “renglón”.

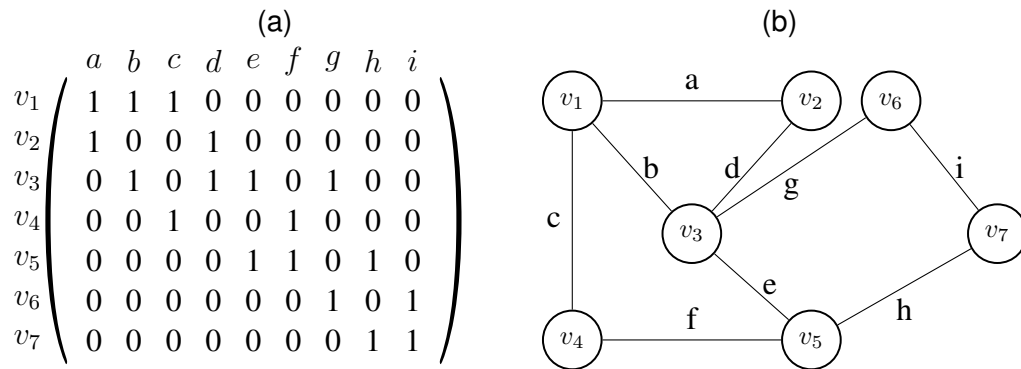
5.3.2. Matriz de incidencias

En esta representación tenemos también una matriz, pero ahora es de $n = |V|$ renglones por $m = |E|$ columnas, donde en cada posición (i, j) se anota si la arista j es incidente en el vértice i . Las ventajas que tiene esta representación es que podemos tener más de una arista entre dos vértices (multigráfica). Las entradas de la matriz quedan de la siguiente forma:

$$m_{i,j} = \begin{cases} 0 & \text{si } e_j \text{ no incide en } v_i \\ 1 & \text{si } e_j \text{ incide en } v_i \end{cases}$$

Si le ponemos nombres a las aristas de la gráfica anterior, la matriz de incidencias quedaría como se ve en la figura 5.23.

Figura 5.23 Representación de gráficas con matriz de incidencias



En esta representación es un poco más complicado conocer cuáles son los vértices adyacentes a un vértice dado. Si queremos saber, por ejemplo, cuáles son los vértices adyacentes al vértice v_3 , observamos cada columna del renglón correspondiente a v_3 en la que haya un 1, indicando que la arista e_j es incidente en ese vértice. Las columnas correspondientes son la b , d , e y g . A continuación revisamos cada una de estas columnas y aquel renglón distinto del correspondiente a v_3 en el que haya un 1, corresponde a un vértice adyacente al vértice v_3 ; en la columna b es v_1 ; en la columna d es v_2 ; en la columna e es v_5 y en la columna g es v_8 .

Sin embargo, a la pregunta de si una arista es o no incidente en un vértice la respuesta se obtiene simplemente revisando la entrada correspondiente de la matriz.

De esta representación obtenemos rápidamente la información respecto al grado de un vértice, ya que corresponde al número de aristas incidentes en él.

Lema 5.1 *En una gráfica representada por una matriz de incidencias, el grado del vértice v_i está dado por la suma de las entradas del renglón i en la matriz.*

La demostración de este lema se deja como ejercicio.

El número de aristas en la gráfica también se obtiene de manera inmediata de esta representación, ya que corresponde al número de columnas de la matriz.

5.3.3. Listas de adyacencias

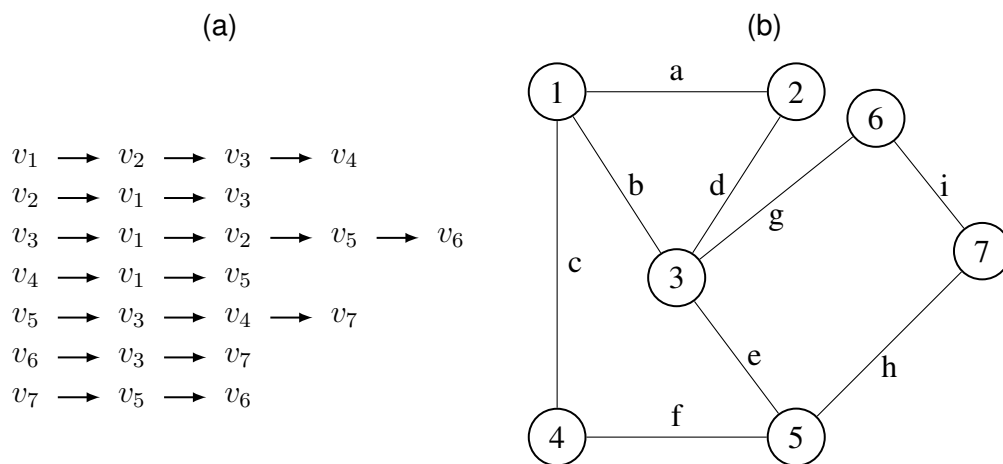
Cuando representamos a una gráfica a través de una matriz, sea ésta de adyacencias o incidencias, se tiene la sensación de que se desperdicia mucho lugar en la matriz. Por ejemplo, si los vértices, en general, tienen un grado menor a la mitad del número de vértices ($\forall v \in V, \text{grado}(v) < |V|$), la matriz va a tener más ceros que unos. Si estamos hablando de una matriz de incidencias, cada columna tiene a lo más dos entradas con 1 y el resto con 0. Las entradas que realmente tienen información son las que tienen 1, por lo que decimos que la densidad de la matriz es baja (a matrices con esta distribución de información se les conoce como *matrices ralas* y a las gráficas correspondientes también se les llama *gráficas ralas* porque tienen pocas aristas).

Si en lugar de una posición para cada combinación posible, listamos únicamente aquellas entradas que sean distintas de 0, y si la matriz es rala, podemos ahorrar muchísimo espacio. Por ejemplo, si la gráfica tiene 1,000 vértices que representan, digamos, a todos los pueblos por los que pasan todas las carreteras del país, pero cada pueblo está conectado a, digamos, un máximo de 6 pueblos distintos a él, con las listas de adyacencias ocuparíamos algo así como 12,000 lugares², mientras que con la matriz de adyacencias ocuparíamos 1,000,000 de lugares.

²En una lista se ocupan dos espacios por cada elemento, uno para el elemento mismo y otro como referencia al que sigue en la lista.

La representación con listas de adyacencias es como sigue: se usan $n = |V|$ listas, una para cada vértice. Estas listas pueden estar en un vector, de tal manera que la lista correspondiente al i -ésimo vértice se encuentre a partir de la posición i del vector. A partir de ahí, vamos colocando en la lista del vértice i a cada uno de los vértices adyacentes a él. Para que el tamaño de cada lista pueda ser variable, cada elemento de la lista tiene una referencia (apuntador, liga) al siguiente vértice adyacente, o bien una referencia nula si éste es el último vértice de la lista.

Figura 5.24 Representación de gráficas con listas de adyacencias



En el caso de la gráfica que nos ocupa, para cada vértice v_i , armamos una sucesión (una lista) con todos los vértices que son adyacentes a él. Nuevamente utilizamos a la gráfica de la figura 5.23 para construir la lista de adyacencias en la figura 5.24.

Esta representación resulta apropiada para recorrer gráficas, pues se pueden ir eliminando de las listas para cada vértice a aquellos vértices que se van alcanzando. Asimismo, es fácil recorrer aristas pues se va de vértice en vértice de manera prácticamente directa.

Las respuestas a las preguntas relacionadas con los grados de los vértices se obtienen de manera muy eficiente, pues se cuentan los elementos presentes en cada una de las listas. Averiguar si dos vértices son o no adyacentes puede ser un poco más costoso, pues hay que recorrer toda la lista de uno de los vértices antes de decidir que el otro vértice no aparece en esa lista, mientras que en la representación de matriz de adyacencias la respuesta se obtiene simplemente viendo la entrada correspondiente de la matriz.

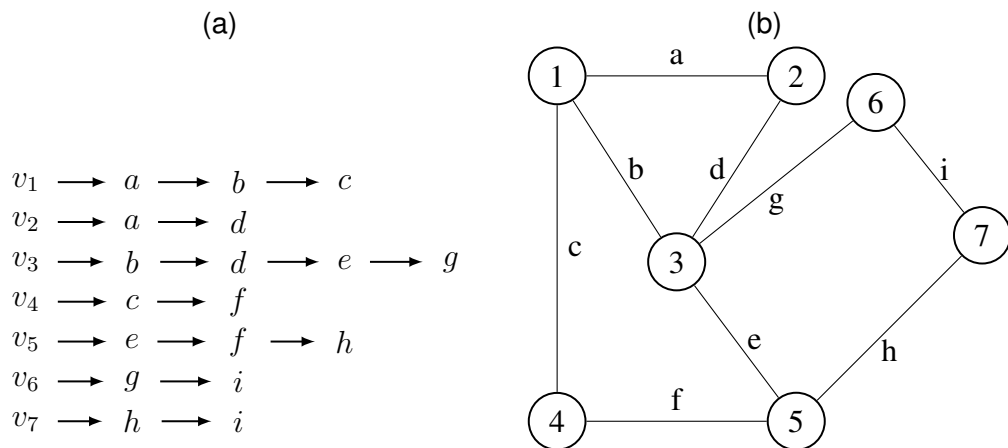
Esta representación es la preferida para la mayoría de los algoritmos que pretenden explorar una gráfica.

5.3.4. Listas de incidencias

En la representación de gráficas mediante listas de incidencias seguimos una lógica parecida a la que usamos para las listas de adyacencias: únicamente vamos a listar, para cada vértice, aquellas aristas que son incidentes a ese vértice. La diferencia principal es que en lugar de que aparezcan los vértices adyacentes aparecen las aristas incidentes.

En la figura 5.25 mostramos la codificación de la gráfica que hemos estado manejando.

Figura 5.25 Representación de gráficas con listas de incidencias

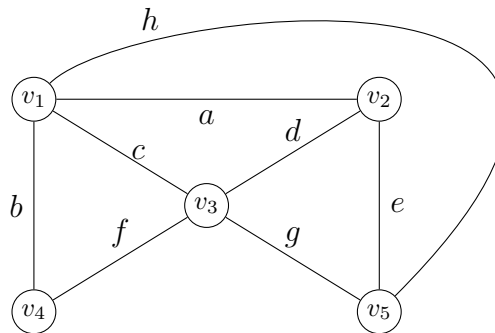


Nuevamente en esta representación obtenemos información de manera eficiente respecto a los grados de los vértices de la gráfica. Para saber cuáles son los vértices adyacentes a un vértice dado deberemos contar con otra estructura de datos que nos indique cuáles son los extremos de las aristas listadas. Si bien esto requiere de más espacio, el tiempo requerido para responder a esta pregunta también es constante, por lo que se considera una operación eficiente.

Cada una de estas representaciones tiene ventajas y desventajas. Es claro que la elección de representación va a depender del tamaño de la gráfica (podemos pensar en gráficas con un número muy grande de vértices), de qué tan conectados estén los vértices entre sí (si hay una gran cantidad de conexiones entre los vértices o la gráfica es completa, la matriz resultará densa y por lo tanto con poco desperdicio), de cuáles sean las preguntas que nos hagamos respecto a ella y, finalmente, de cuáles son los algoritmos que queramos ejecutar sobre la gráfica.

Ejercicios

- 5.3.1.- Demuestra el lema 5.1.
- 5.3.2.- Cuenta el número de observaciones que se tienen que hacer en una gráfica representada por una matriz de adyacencias para determinar el grado de cada uno de los vértices de la gráfica.
- 5.3.3.- Explica cómo se obtienen los extremos de una arista dada en cada una de las representaciones que dimos.
- 5.3.4.- Para la siguiente gráfica, da su representación como matriz de adyacencias, matriz de incidencias, listas de adyacencias y listas de incidencias.



- 5.3.5.- Explica por qué las listas de adyacencias y las listas de incidencias tienen exactamente el mismo número de elementos renglón por renglón.
- 5.3.6.- ¿Con cuál de las codificaciones se pueden representar *lazos*, o sea aristas que salen y llegan al mismo vértice?
- 5.3.7.- Dada las siguientes listas de adyacencias, dibuja una gráfica que corresponda a estas listas.

Listas de adyacencias:

$v_1 \longrightarrow v_2 \longrightarrow v_5 \longrightarrow v_6$
 $v_2 \longrightarrow v_1 \longrightarrow v_3 \longrightarrow v_6$
 $v_3 \longrightarrow v_2 \longrightarrow v_4 \longrightarrow v_6$
 $v_4 \longrightarrow v_3 \longrightarrow v_5 \longrightarrow v_6$
 $v_5 \longrightarrow v_1 \longrightarrow v_4 \longrightarrow v_6$
 $v_6 \longrightarrow v_1 \longrightarrow v_2 \longrightarrow v_3 \longrightarrow v_4 \longrightarrow v_5$

5.3.8.- Dada la siguiente matriz de adyacencias, dibuja una gráfica que tenga a esa matriz como representación.

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

5.3.9.- Dibuja una gráfica que corresponda a la siguiente matriz de adyacencias:

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

5.3.10.- ¿Cómo identificas en una matriz de adyacencias a una gráfica C_n ?

5.3.11.- ¿Cómo identificas en una matriz de adyacencias a una gráfica K_n ?

5.3.12.- En una gráfica no dirigida bastaría con registrar la mitad de la matriz de adyacencias, ya que la matriz es simétrica: lo que está bajo la diagonal se repite arriba de la diagonal. Justifica esta aseveración.

5.3.13.- ¿Cómo identificas en una matriz de adyacencias una gráfica que tiene al menos un lazo?

5.3.14.- En las listas de incidencias que corresponde a una gráfica no dirigida, ¿cuántas veces aparece cada arista?

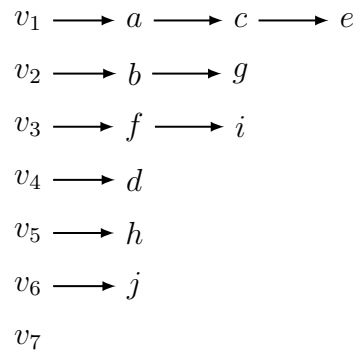
5.3.15.- ¿Qué cambios hay que hacer a una matriz de adyacencias para poder representar a una gráfica dirigida?

5.3.16.- Dada la siguiente matriz de adyacencias que corresponde a una gráfica dirigida, dibuja una gráfica que corresponda a esta matriz.

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

5.3.17.- Dadas las siguientes listas de incidencias, dibuja una gráfica dirigida que corresponda a estas listas.

arco	vértice origen	vértice destino
a	v_1	v_2
b	v_2	v_4
c	v_1	v_4
d	v_4	v_7
e	v_1	v_3
f	v_3	v_1
g	v_2	v_5
h	v_5	v_4
i	v_3	v_6
j	v_6	v_4



5.3.18.- Revisa cada una de las representaciones posibles y discute la manera en que se logra invertir la dirección de todos los arcos en la gráfica dirigida del ejercicio 17.

5.4. Isomorfismo entre gráficas

Como pudimos observar en los ejercicios de la sección anterior, hay más de una manera de dibujar una misma gráfica – recordemos que el término *gráfica* corresponde al ente matemático –. ¿Cómo podemos determinar que dos dibujos corresponden a la misma gráfica? ¿Por qué el interés de determinar esto?

Muchas veces tenemos dos problemas aparentemente distintos, pero que denotan a gráficas similares (o a la misma gráfica). En estos casos, la solución a uno de los problemas

de manera automática nos proporciona la solución a todos los problemas que tengan a la misma gráfica, aunque sus gráficas respectivas estén dibujadas o especificadas de manera distinta. Revisemos un caso donde esto sucede.

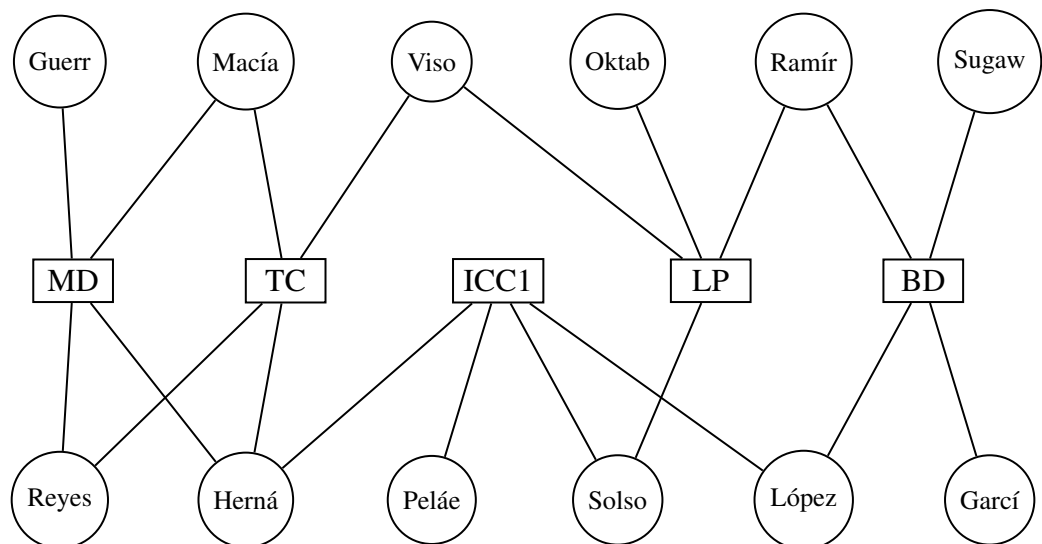
Por ejemplo, supongamos que tenemos cinco materias en la carrera de Ciencias de la Computación; de cada materia se reciben cuatro solicitudes cada semestre y existen colisiones entre los profesores que desean impartir estas materias, como se muestra en la tabla 5.4:

Tabla 5.4 Grupos para las materias de CC

<i>Materia</i>	<i>Profesores:</i>			
Matemáticas Discretas	Macías	Hernández	Guerrero	Reyes
Teoría de la Computación	Viso	Reyes	Macías	Hernández
Lenguajes de Programación	Solsona	Ramírez	Viso	Oktaba
ICC1	Solsona	Peláez	Hernández	López
Bases de datos	García	López	Ramírez	Sugawara

Podemos pensar en una gráfica donde cada vértice es una de las materias o uno de los profesores; la materia a es adyacente al profesor B si es que el profesor B solicita la materia a . La gráfica queda como se muestra en la figura 5.26.

Figura 5.26 G_1 para las materias de CC



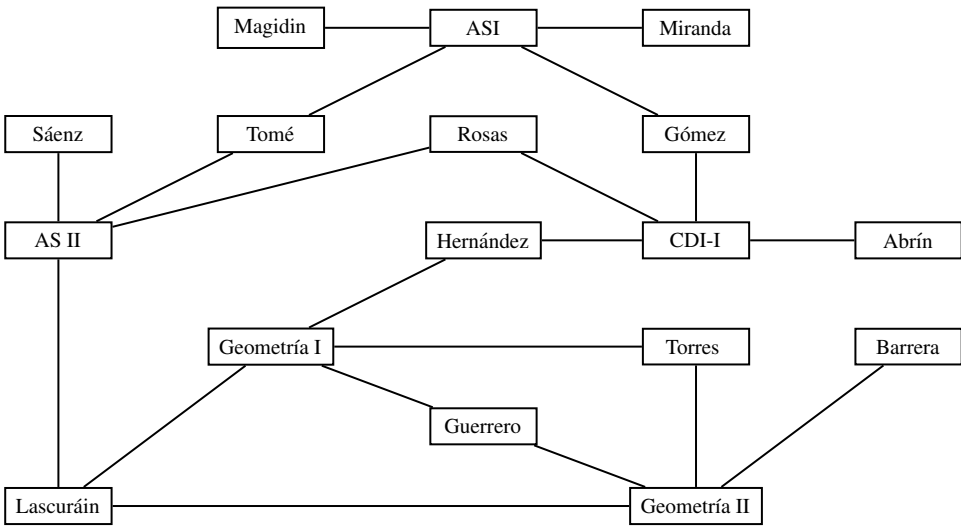
Es conveniente notar que no hay aristas entre profesores, ya que, como está dibujado, las materias y los profesores corresponden a categorías distintas.

También en matemáticas se consideran cinco materias y las solicitudes para ellas. La relación está dada por la tabla 5.5 y la gráfica correspondiente se encuentra en la figura 5.27. La gráfica que corresponde a esta tabla se encuentra en la figura 5.27.

Tabla 5.5 Materias en Matemáticas

Materia	Profesores:			
Álgebra Superior I	Gómez	Magidin	Miranda	Tomé
Álgebra Superior II	Lascuráin	Sáenz	Rosas	Tomé
Geometría Analítica I	Guerrero	Hernández	Torres	Lascuráin
Geometría Analítica II	Barrera	Guerrero	Lascuráin	Torres
Cálculo Dif e Int I	Abrín	Gómez	Hernández	Rosas

Figura 5.27 Gráfica correspondiente a las materias de matemáticas



Aunque estas dos gráficas, a simple vista, no se parecen, podemos encontrar, por ejemplo, que las podríamos dibujar igual. En general, podemos encontrar que el número de vértices y aristas es el mismo, y que por cada vértice con grado k en una de las gráficas existe un vértice con el mismo grado en la otra. Si verificamos otra condición más, que consiste en que las relaciones de adyacencia se mantengan, decimos entonces que podemos definir un *isomorfismo* entre ambas gráficas.

Definición 5.20 Una gráfica $G_1 = (V_1, E_1)$ es *isomorfa* a una gráfica $G_2 = (V_2, E_2)$ si es que podemos hacer una correspondencia 1 a 1 entre los vértices (una función biyectiva de los vértices de G_1 a los vértices de G_2), $f : V_1 \mapsto V_2$ de tal manera que

$$uv \in E_1 \iff f(u)f(v) \in E_2$$

Es fácil corroborar que existen el mismo número de vértices y de aristas en las dos gráficas de las figuras 5.26 y 5.27; asimismo, también se preserva el número de vértices con grado 1, 2 y así sucesivamente, como se puede ver en la tabla 5.6.

Tabla 5.6 Condiciones necesarias para que haya isomorfismo entre gráficas

Gráfica	Número de vértices:	Número de aristas:
G_1	17	20
G_2	17	20

Gráfica	grado=1	grado=2	grado=3	grado=4	Total de vértices
G_1	5	6	1	5	17
G_2	5	6	1	5	17

Dado que se cumplen las condiciones básicas, no podemos descartar que haya un isomorfismo entre estas dos gráficas. Deberemos verificar, sin embargo, que las relaciones de adyacencia se mantengan. Para ello definimos un isomorfismo para este caso en la tabla 5.7.

Tabla 5.7 Isomorfismo entre las gráficas de las figuras 5.26 y 5.27

f(MD)	=	Geometría II
f(TC)	=	Geometría I
f(ICC1)	=	Álgebra Superior II
f(LP)	=	Cálculo I
f(BD)	=	Álgebra Superior I
f(Sugawara)	=	Magidin
f(García)	=	Miranda
f(Peláez)	=	Sáenz
f(López)	=	Tomé
f(Solsona)	=	Rosas
f(Ramírez)	=	Gómez
f(Viso)	=	Hernández
f(Oktaba)	=	Abrín
f(Reyes)	=	Torres
f(Guerrero)	=	Barrera
f(Macías)	=	Guerrero
f(Hernández)	=	Lascuráin

Podemos verificar que esta función preserva la adyacencia entre vértices en una gráfica y la imagen de esos vértices en la otra. Por ejemplo, en la primera gráfica tenemos las siguientes aristas:

$(MD, Guerrero), (LP, Ramírez), (ICC1, Peláez)$

que en la segunda gráfica, bajo el isomorfismo, tenemos:

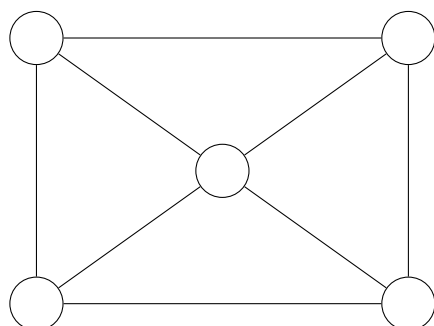
$(f(MD), f(Guerrero)) \quad (Geometría II, Barrera)$
 $(f(LP), f(Ramírez)) \quad (Cálculo I, Gómez)$
 $(f(ICC1), f(Peláez)) \quad (Álgebra Superior II, Sáenz)$

que como podemos verificar con la gráfica, están todas presentes en la segunda gráfica. El lector puede verificar que la relación de adyacencia se preserva dada esta asociación. Sin embargo, no es la única asociación posible.

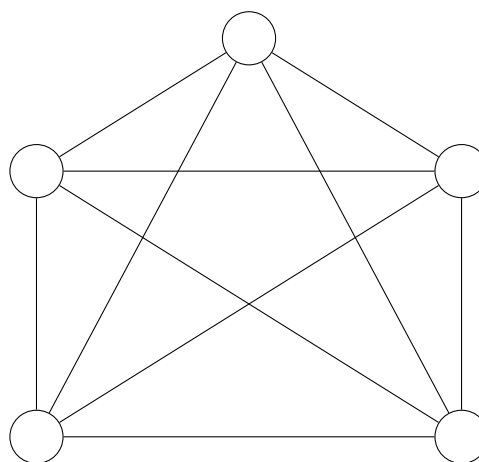
Ejercicios

5.4.1.- Determina si las dos gráficas que se encuentran a continuación son isomorfas. Si lo son, da el isomorfismo que mantiene las relaciones de adyacencia. Si no son isomorfas, explica qué es lo que hace que no lo sean.

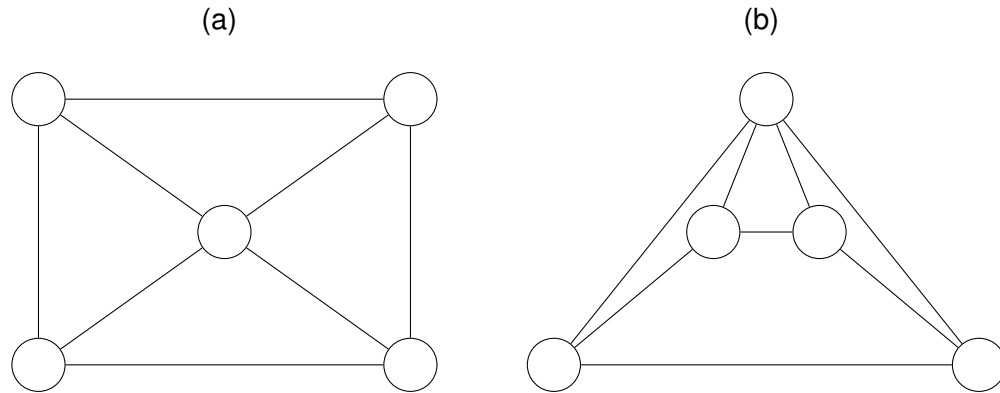
(a)



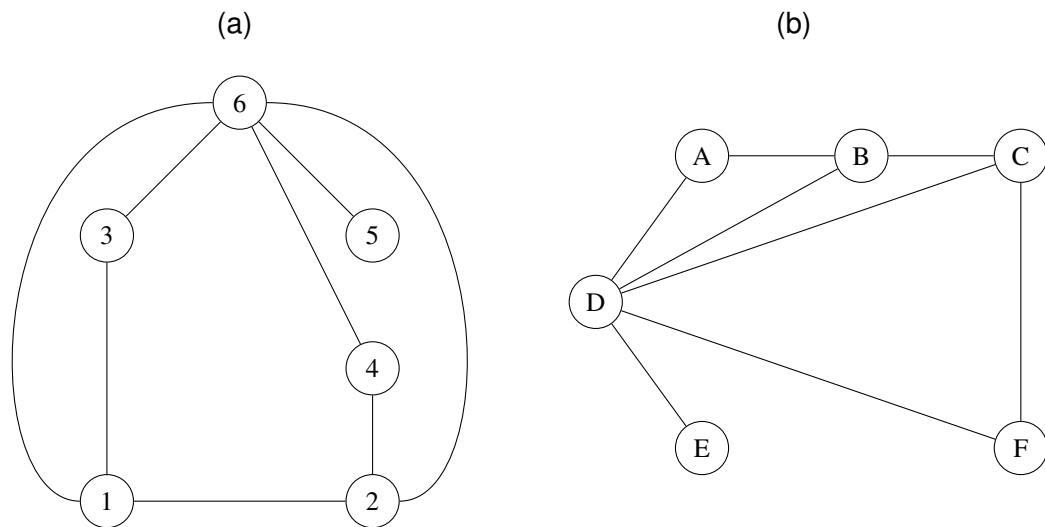
(b)



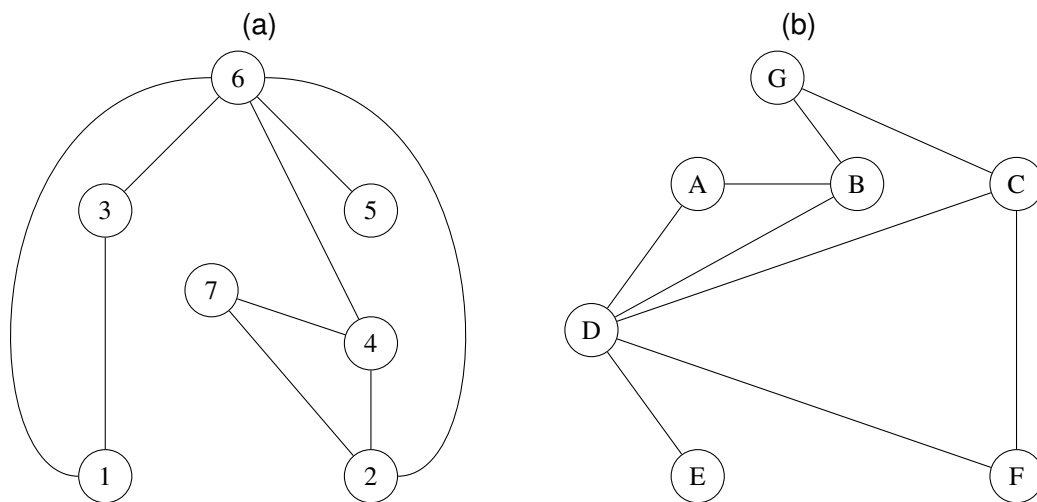
5.4.2.- Determina si las dos gráficas que se encuentran a continuación son isomorfas. Si lo son, da el isomorfismo que mantiene las relaciones de adyacencia. Si no son isomorfas, explica qué es lo que hace que no lo sean.



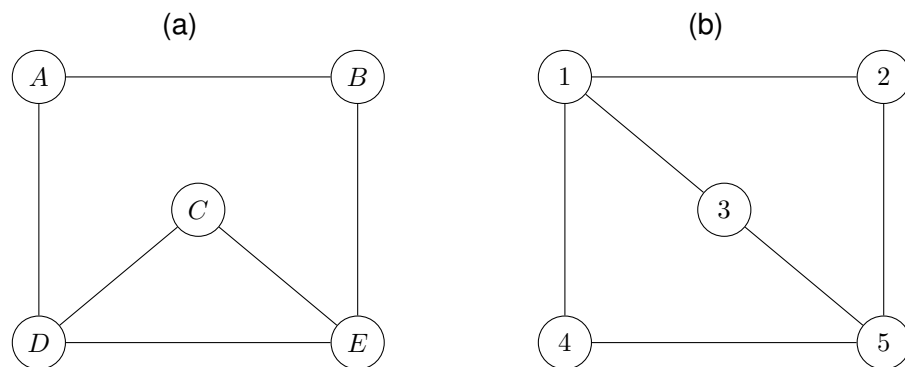
5.4.3.- Determina si las dos gráficas que se encuentran a continuación son isomorfas. Si lo son, da el isomorfismo que mantiene las relaciones de adyacencia. Si no son isomorfas, explica qué es lo que hace que no lo sean.



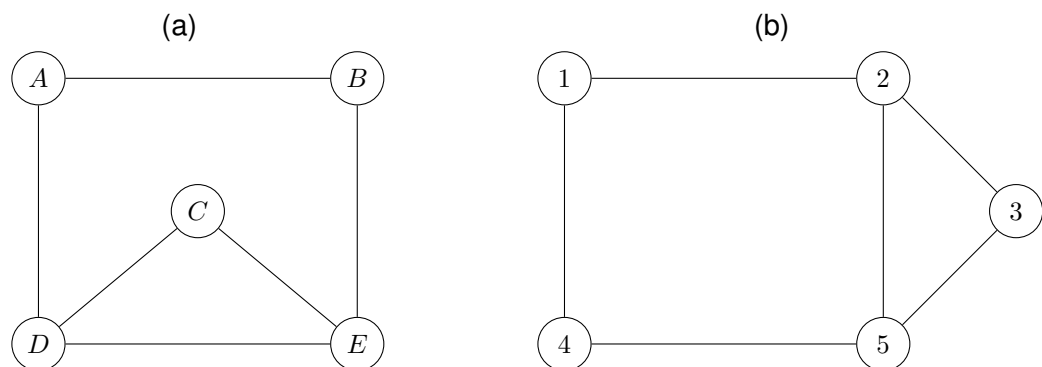
5.4.4.- Determina si las dos gráficas que se encuentran a continuación son isomorfas. Si lo son, da el isomorfismo que mantiene las relaciones de adyacencia. Si no son isomorfas, explica qué es lo que hace que no lo sean.



5.4.5.- Di si es posible un isomorfismo entre las siguientes dos gráficas. Si es posible, propón uno; si no es posible definir un isomorfismo, explica por qué.



5.4.6.- Di si es posible un isomorfismo entre las siguientes dos gráficas. Si es posible, propón uno; si no, explica por qué.



Exploración en gráficas | 6

6.1. Circuitos eulerianos

Una exploración en una gráfica implica recorrerla de alguna manera para descubrir propiedades de la misma. Para explorar una gráfica debemos visitar los vértices de la misma usando las aristas para pasar de un vértice a otro; en otras palabras, definimos *caminos* en ellas. Visitaremos primero la exploración en gráficas no dirigidas (o simplemente gráficas).

Definición 6.1 (camino) Un *camino* en una gráfica es una sucesión de vértices intercalados con aristas, donde el primero y último elemento de la sucesión son vértices. Si el primer vértice de la sucesión es u y el último vértice es v , decimos que tenemos un camino de u a v . Podemos denotar a ese camino con una letra mayúscula (usualmente P) y damos los vértices u y v en sus extremos, $u \rightsquigarrow_P v$ (o $u \overset{P}{\rightsquigarrow} v$); o si no le ponemos nombre al camino, simplemente lo denotamos con $u \rightsquigarrow v$.

Por ejemplo, observemos la gráfica de la figura 6.1. En esta gráfica tenemos los siguientes caminos entre el vértice v y el vértice t :

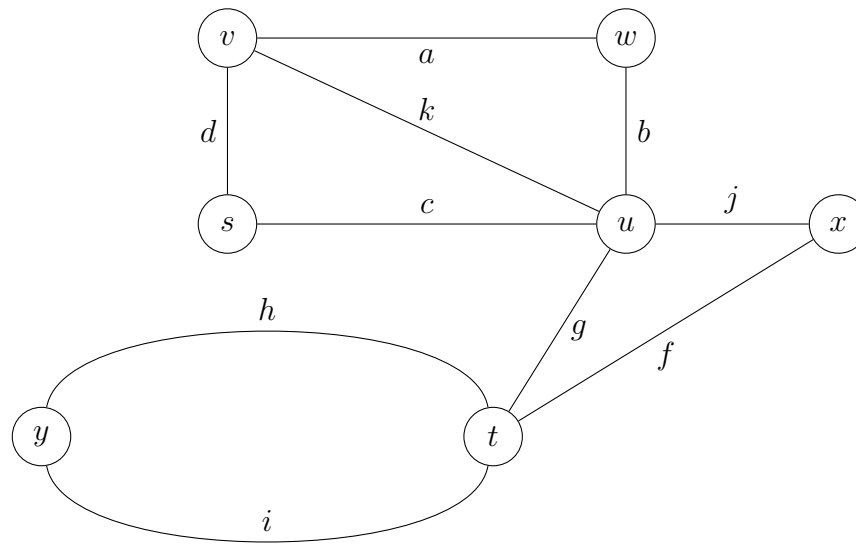
$$P_1 = v - a - w - b - u - g - t$$

$$P_2 = v - d - s - c - u - b - w - a - v - k - u - g - t - f - x - j - u - g - t$$

$$P_3 = v - a - w - b - u - j - x - f - t - h - y - i - t$$

y muchos más. Noten que los nombres de los vértices corresponden a letras cerca del final del alfabeto mientras que los nombres de las aristas corresponden a letras cerca del principio del alfabeto. En ambos casos usaremos letras minúsculas.

Figura 6.1 Gráfica no dirigida con caminos definidos



Si es que no hay confusión (que la gráfica no tenga más de una arista entre cualesquiera dos vértices) el camino puede representarse únicamente por la sucesión de vértices. En cualquier caso un camino está totalmente representado por la sucesión de aristas.

Los primeros dos caminos que dimos pueden ser representados por sus vértices o aristas. El tercero no puede ser representado por la sucesión de vértices, pues no quedaría claro cuál arista fue la utilizada entre los vértices y y t .

$$P_1 = v, w, u, t = a, b, g$$

$$P_2 = v, s, u, w, v, u, t, u, t = d, c, b, a, k, g, f, j, g$$

$$P_3 = v, w, u, x, t, y, t = a, b, j, f, h, i$$

Podemos observar en estos tres caminos que algunas veces regresamos a un vértice ya visitado o a una arista ya recorrida. Nos interesa el camino más sencillo que podamos encontrar, donde *sencillo* lo interpretamos como con el menor número de vértices o aristas que llevan del vértice inicial al final.

Definición 6.2 Una *trayectoria* entre u y v ($u - v$) es aquella en el que no se repiten vértices – y por lo tanto tampoco se repiten aristas –.

La única trayectoria que dimos en la gráfica de la figura 6.1 es P_1 . Otras trayectorias en esa misma gráfica son v, u, t y v, s, u, x, t .

Íntimamente relacionado con el concepto de trayectoria es el de una *trayectoria cerrada*.

Definición 6.3 (Trayectoria cerrada) Una *trayectoria cerrada* es una sucesión de vértices y aristas intercalados, donde el primer y último vértice son el mismo vértice.

Definición 6.4 (Ciclo) Un *ciclo* es una trayectoria cerrada.

Al igual que con caminos, podemos tener un ciclo dentro de otro ciclo. Por ejemplo, en la figura 6.2 tenemos los siguientes ciclos, con ciclos dentro de ellos:

$$C_1 = a, b, i, k, d, c, a$$

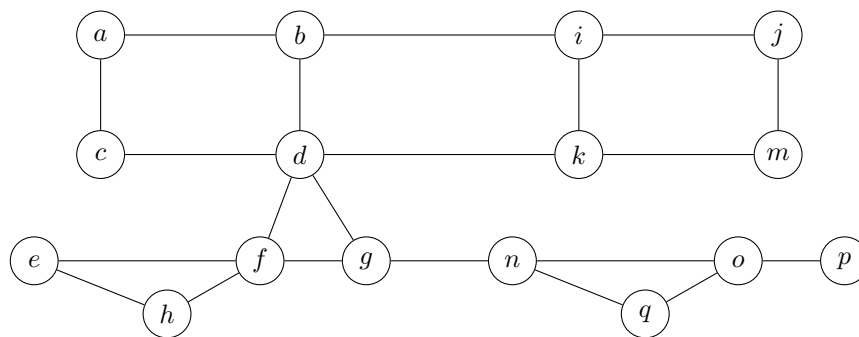
$$C_2 = d, f, e, h, f, g, d$$

$$C_3 = a, b, d, g, f, d, c, a$$

$$P = g, d, f, g, n, o, q, n, o, p$$

En el caso de C_1 tenemos un ciclo que empieza y termina en el vértice a . En C_2 podemos identificar el ciclo “externo” que empieza y termina en el vértice d . Dentro de ese ciclo podemos identificar otro ciclo, el que corresponde a f, e, h, f . En C_3 también tenemos al ciclo d, g, f, d contenido en el ciclo exterior que empieza y termina en el vértice a . Por último, en P , que no es un ciclo, podemos encontrar el ciclo n, o, q, n contenido en el camino; también podemos encontrar el ciclo o, q, n, o que se intersecta con el ciclo n, o, q, n .

Figura 6.2 Ciclos en una gráfica



Regresando a caminos en general, nos interesa, por supuesto, comparar qué tan grande es un camino con respecto a otro:

Definición 6.5 La *longitud* de un camino P , denotada por $|P|$, es el número de aristas en el camino (uno menos que el número de vértices).

En la gráfica de la figura 6.1, el camino P_1 tiene longitud 3; el camino P_2 tiene longitud 9; y el camino P_3 tiene longitud 6:

$$|P_1| = 3; \quad |P_2| = 9; \quad |P_3| = 6.$$

Por definición, todo vértice tiene un camino a sí mismo de longitud 0.

Teorema 6.1 *Todo camino entre u y v (camino $u-v$) contiene a una trayectoria.*

Demostración.

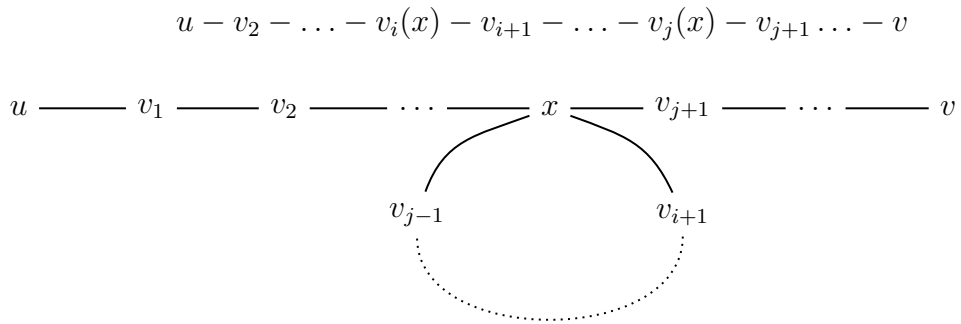
Si el camino $u-v$ es una trayectoria, se contiene a sí misma y queda demostrado el teorema.

Supongamos que el camino $u-v$ no es una trayectoria y veamos la sucesión de vértices que lo denotan (anotando la posición que ocupa cada uno de ellos):

$$v_1, v_2, \dots, v_n$$

Como no es trayectoria existe algún vértice en esta sucesión que se repite. Si $u = v$ (se sale y llega al mismo vértice), eliminamos toda la sucesión y dejamos únicamente el primer vértice, que tiene una trayectoria de tamaño cero a sí mismo.

Supongamos que $u \neq v$. Entonces tenemos dos vértices en la sucesión, digamos en las posiciones i y j , que son el mismo vértice ($v_i = x = v_j$). Por lo tanto, tenemos un camino de la siguiente forma:



Si procedemos a quitar del camino el subcamino

$$x = v_i - v_{i+1} - \dots - v_{j-1} - v_j = x,$$

nos quedaremos con el camino más corto dado por

$$u - v_2 - \dots - v_i - v_{j+1} \dots - v.$$

Si este camino todavía no es trayectoria, volvemos a hacer lo mismo hasta que ya no queden vértices repetidos en la sucesión. □

Queremos explorar cuál es la relación entre los vértices de una gráfica, como por ejemplo si hay algún camino entre ellos. Esto nos lleva a la siguiente definición:

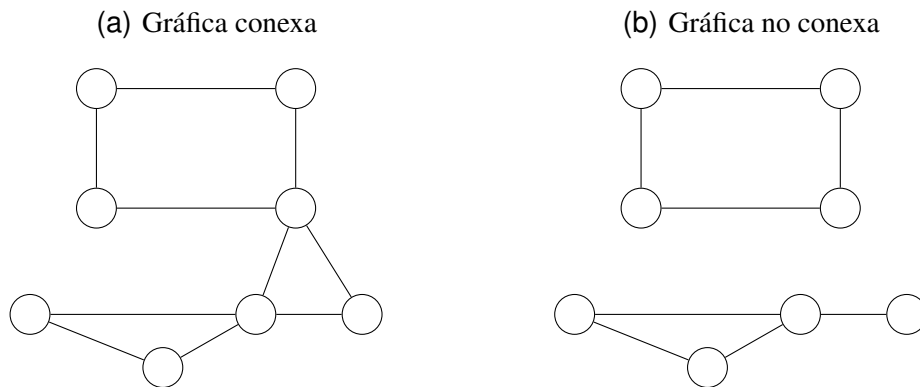
Definición 6.6 Una gráfica $G = (V, E)$ es *conexa* si para cualesquiera dos vértices u y $v \in V$ existe un camino entre ellos.

Definición 6.7 Una gráfica que no es conexa está compuesta por dos o más *componentes conexas*, donde en cada componente hay un camino entre cualesquiera dos vértices.

En la figura 6.3 mostramos una gráfica conexa y una que no lo es. Si bien hasta el momento hemos trabajado casi exclusivamente con gráficas conexas, es importante notar que la gráfica en 6.3(b) cumple perfectamente con la definición que dimos para una gráfica: conjunto de vértices y conjunto de aristas denotadas por la relación entre los vértices.

En la figura 6.3(a) tenemos una gráfica donde no importa qué pareja de vértices tomemos, siempre hay una trayectoria entre ellos. En cambio, en la figura 6.3(b) tenemos dos componentes conexas, pero no hay trayectoria entre algún vértice en una de las componentes y un vértice en la otra componente.

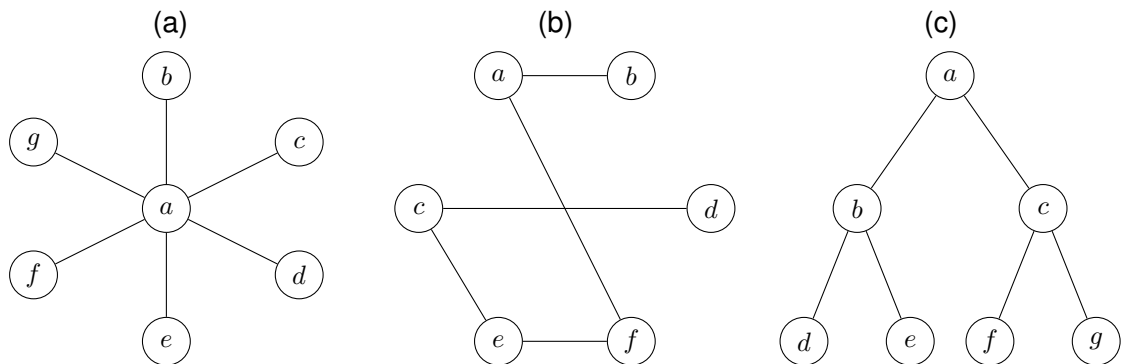
Figura 6.3 Conexidad en gráficas



Otro tipo de gráfica que no hemos introducido en este capítulo, y que será importante en el futuro, son los *árboles*.

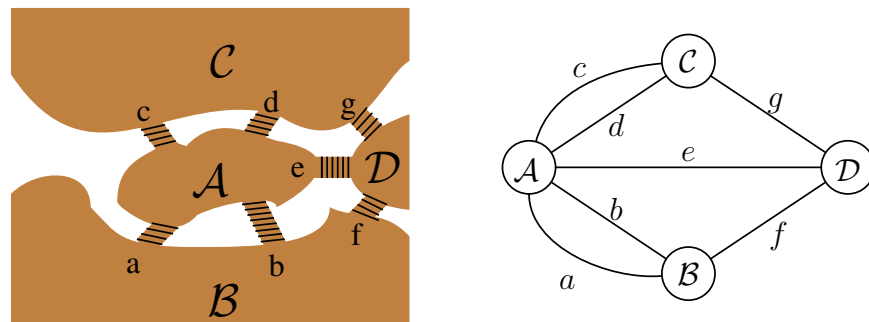
Definición 6.8 (Árbol) Un *árbol* es una gráfica acíclica y conexa.

Daremos más adelante varias definiciones equivalentes de lo que es un árbol y demostraremos propiedades que nos serán muy útiles. Por lo pronto nos conformamos con esta definición, que se ilustra en las gráficas de la figura 6.4.

Figura 6.4 Gráficas que son árboles

Circuitos eulerianos

Los problemas relacionados con caminos en gráficas se aplican a muchas situaciones en la vida real. El primero de ellos (considerado como el problema *fundador* de la teoría de gráficas) fue planteado por Leonhard Euler en 1736 y surge como la solución a un problema sencillo, conocido hoy en día como *circuito euleriano*.

Figura 6.5 Problema de los Puentes de Königsberg

Euler nació el 15 de abril de 1707 en Basilea, Suiza. En la década de 1730 a sus manos llegó un problema relativo a la ciudad de Königsberg en Prusia (actualmente la ciudad se llama Kaliningrado y se encuentra en Rusia); esta ciudad bordeaba al río Pregel e incluía dos grandes islas que estaban conectadas entre sí y a tierra firme por siete puentes¹. El acertijo consistía en definir un paseo por esa parte del río de tal manera de recorrer cada puente exactamente una vez y regresar al punto desde el que se inició el paseo. Euler modeló este problema con lo que conocemos hoy en día como una multigráfica, donde las aristas eran los puentes y había un vértice por cada posición de tierra. No se debería

llegar a una isla más que recorriendo alguno de los puentes y todos los puentes deberían ser recorridos de extremo a extremo. El problema consistía en encontrar si la gráfica tenía un paseo de este tipo. La gráfica resultante fue la que se muestra en la figura 6.5.

Definición 6.9 (Circuito euleriano) Un *circuito euleriano* sobre una gráfica (o multigráfica) es aquel ciclo en la gráfica donde cada arista aparece exactamente una vez.

Similar a este problema se definió el de encontrar si una gráfica tiene un *paseo euleriano*, que se define como sigue:

Definición 6.10 (Paseo euleriano) Un *paseo euleriano* en una gráfica (o multigráfica) es una trayectoria que incluye a cada arista de la gráfica exactamente una vez, empezando y terminando en vértices distintos.

No sólo pudo Euler dar respuesta a esta pregunta, sino que caracterizó a aquellas gráficas que sí tienen un circuito euleriano de la siguiente forma:

Teorema 6.2 Una gráfica conexa $G = (V, E)$ tiene un circuito euleriano si y sólo si todos los vértices tienen grado par. G tiene un paseo euleriano si y sólo si todos los vértices de G tienen grado par, excepto exactamente dos vértices que tienen grado impar.

Demostración.

Supongamos que la gráfica tiene un circuito euleriano y supongamos que iniciamos el circuito en un cierto vértice v_0 recorriendo una arista. v_0 tiene al menos grado 1 porque la gráfica es conexa. A cada vértice que llegamos por una arista volvemos a salir por otra, por lo que cada vértice tiene un número par de aristas incidentes: aquella por la que se llega y aquella por la que se sale. Cuando ya terminamos de recorrer todas las aristas regresamos a v_0 , por lo que le sumamos 1 al grado (impar hasta ahora) de v_0 . De donde todos los vértices tienen grado par.

Supongamos ahora que en $G = (V, E)$ todos los vértices tienen grado par. Elegimos a un vértice v como origen del circuito y vamos recorriendo aristas. A cada vértice que llegamos podemos salir, pues el grado es par. Como únicamente hemos utilizado un número impar de aristas incidentes en v (por la que salimos más dos por cada vez que pasemos por v), tenemos que usar una última arista que regrese a v y ya no podremos salir de él. Tomamos entonces cualquier arista que no ha sido recorrida y cualquiera de los vértices en uno de sus extremos, digamos u . Volvemos a repetir el ejercicio hasta que tengamos un circuito que empiece y termine en u . Como la gráfica es conexa, existe un camino entre cualquier arista no usada todavía y algún vértice en el circuito construido hasta el momento, por lo que el último circuito se conecta al anterior en un vértice. Esto es, toda arista es finalmente incluida en el circuito.

Para el caso de paseos eulerianos, se sigue el mismo razonamiento, excepto que el vértice del que se sale y al que se llega finalmente no son el mismo. □

¹Desgraciadamente los puentes fueron destruidos durante la Segunda Guerra Mundial.

En el algoritmo 6.1 se formalizan las ideas dadas en la demostración, en cuanto a que si el grado de cada vértice es par, tenemos un circuito euleriano.

Este algoritmo construye el circuito euleriano correctamente, si es que la gráfica cumple con las condiciones necesarias para ello (grado par en todos sus vértices). Veamos un ejemplo:

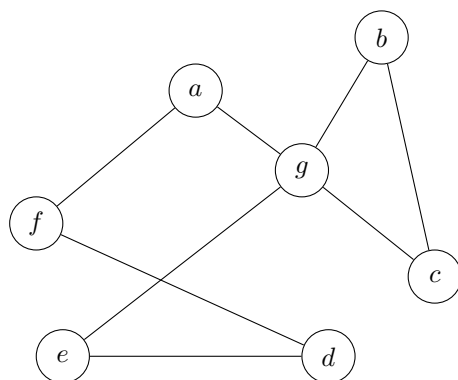
Listado 6.1 Algoritmo para obtener un circuito euleriano

```

1  Sea  $C = [x]$ ,  $x \in V$ ;
2  Sea  $\mathcal{E} = E$ ;
3  Mientras (haya aristas sin usar en  $\mathcal{E}$ )
4      Sea  $v$  un vértice arbitrario en  $C$  que tiene alguna
5      arista disponible.
6      Sea  $P = [v]$ 
7      Sea  $u = v$ ;
8      Mientras ( $\exists e = ux$  disponible)
9           $\mathcal{E} = \mathcal{E} - e$ ;
10          $P = P + e + x$ ;
11          $u = x$ ;
12     /* Ciclo: Mientras ( $\exists e = ux$  disponible) */
13     Sustituir a  $v$  por  $P$  en  $C$ ;
14 /* Ciclo: Mientras (haya aristas sin usar en  $\mathcal{E}$ ) */
15  $C$  es el circuito euleriano;
```

Ejemplo 6.1.

Tenemos la siguiente gráfica que cumple con las condiciones necesarias y suficientes para tener un circuito euleriano. Representamos a la gráfica con listas de adyacencia. A la derecha de la lista de adyacencias se encuentra la lista de aristas, identificadas en orden alfabético, esto es, el primer vértice es el nombre menor. Cada arista aparece en esta lista únicamente una vez.



Listas de adyacencias

$a \rightarrow f \rightarrow g \rightarrow \emptyset$
 $b \rightarrow c \rightarrow g \rightarrow \emptyset$
 $c \rightarrow b \rightarrow g \rightarrow \emptyset$
 $d \rightarrow e \rightarrow f \rightarrow \emptyset$
 $e \rightarrow d \rightarrow g \rightarrow \emptyset$
 $f \rightarrow a \rightarrow d \rightarrow \emptyset$
 $g \rightarrow a \rightarrow b \rightarrow c \rightarrow e \rightarrow \emptyset$

Aristas

$a - f$
 $a - g$
 $b - c$
 $b - g$
 $c - g$
 $d - e$
 $d - f$
 $e - g$

Procedemos a construir un circuito euleriano siguiendo el algoritmo dado. Elegimos al

azar cualquiera de los vértices. El vértice natural a elegir es el primero, a .

Inicialización

$$1 \quad \mathcal{C} = [a];$$

$$2 \quad \mathcal{E} = \{a-f, a-g, b-c, b-g, c-g, d-e, d-f, e-g\};$$

Entramos al ciclo de la línea 3 porque \mathcal{E} tiene todavía aristas sin usar. Tomamos al vértice a en las líneas 4 y 5.

$$4 \quad v = a;$$

$$6 \quad \mathcal{P} = [a];$$

$$7 \quad u = a;$$

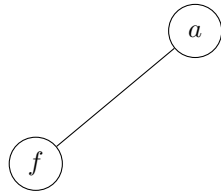
Entramos al ciclo de la línea 8, ya que la lista de adyacencias del vértice a no está vacía, y elegimos al primer vértice adyacente al que se alcanza con la arista $e = a-f$, por lo que eliminamos a esta arista de \mathcal{E} y agregamos a la arista y al vértice en el otro extremo al camino que estamos construyendo.

$$9 \quad \mathcal{E} = \{a-g, b-c, b-g, c-g, d-e, d-f, e-g\};$$

$$10 \quad \mathcal{P} = [a, a-f, f];$$

$$11 \quad u = f;$$

Como usamos la arista $e = a-f = f-a$, eliminamos a f de la lista de adyacencias de a y a a de la lista de adyacencias de f :



Listas de adyacencias

$$a \longrightarrow g \longrightarrow \emptyset$$

$$b \longrightarrow c \longrightarrow g \longrightarrow \emptyset$$

$$c \longrightarrow b \longrightarrow g \longrightarrow \emptyset$$

$$d \longrightarrow e \longrightarrow f \longrightarrow \emptyset$$

$$e \longrightarrow d \longrightarrow g \longrightarrow \emptyset$$

$$f \longrightarrow d \longrightarrow \emptyset$$

$$g \longrightarrow a \longrightarrow b \longrightarrow c \longrightarrow e \longrightarrow \emptyset$$

Seguimos en el ciclo de la línea 8 tomando a la siguiente arista disponible desde f que es $f-d = d-f$, ya que d es el primer vértice en la lista de adyacencias de f .

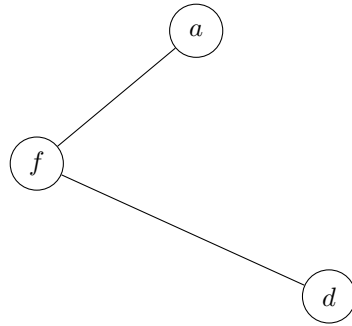
$$8 \quad e = f-d;$$

$$9 \quad \mathcal{E} = \{a-g, b-c, b-g, c-g, d-e, e-g\};$$

$$10 \quad \mathcal{P} = [a, a-f, f, f-d, d];$$

$$11 \quad u = d;$$

Como usamos la arista $e = f-d = d-f$, eliminamos a d de la lista de adyacencias de f y a f de la lista de adyacencias de d :

**Listas de adyacencias**

$a \rightarrow g \rightarrow \emptyset$
 $b \rightarrow c \rightarrow g \rightarrow \emptyset$
 $c \rightarrow b \rightarrow g \rightarrow \emptyset$
 $d \rightarrow e \rightarrow \emptyset$
 $e \rightarrow d \rightarrow g \rightarrow \emptyset$
 $f \rightarrow \emptyset$
 $g \rightarrow a \rightarrow b \rightarrow c \rightarrow e \rightarrow \emptyset$

Seguimos en el ciclo de la línea 8 pues d tiene todavía vértices en su lista de adyacencias, el primero de los cuales es e ; así que tomamos la arista $d-e = e-d$.

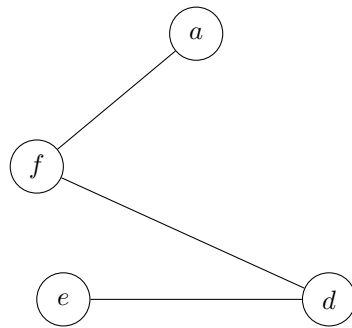
$$8 \quad e = d-e;$$

$$9 \quad \mathcal{E} = \{a-g, b-c, b-g, c-g, e-g\};$$

$$10 \quad \mathcal{P} = [a, a-f, f, f-d, d, d-e, e];$$

$$11 \quad u = e;$$

Como usamos la arista $e = d-e = e-d$, eliminamos a d de la lista de adyacencias de e y a e de la lista de adyacencias de d :

**Listas de adyacencias**

$a \rightarrow g \rightarrow \emptyset$
 $b \rightarrow c \rightarrow g \rightarrow \emptyset$
 $c \rightarrow b \rightarrow g \rightarrow \emptyset$
 $d \rightarrow \emptyset$
 $e \rightarrow g \rightarrow \emptyset$
 $f \rightarrow \emptyset$
 $g \rightarrow a \rightarrow b \rightarrow c \rightarrow e \rightarrow \emptyset$

Seguimos en el ciclo de la línea 8 pues e tiene todavía vértices en su lista de adyacencias, el primero de los cuales es g , así que tomamos la arista $e-g = g-e$.

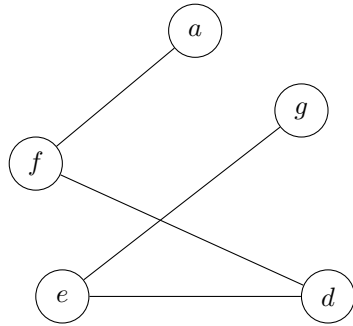
$$8 \quad e = e-g;$$

$$9 \quad \mathcal{E} = \{a-g, b-c, b-g, c-g\};$$

$$10 \quad \mathcal{P} = [a, a-f, f, f-d, d, d-e, e, e-g, g];$$

$$11 \quad u = g;$$

Eliminamos a e de la lista de adyacencias de g y a g de la lista de e :

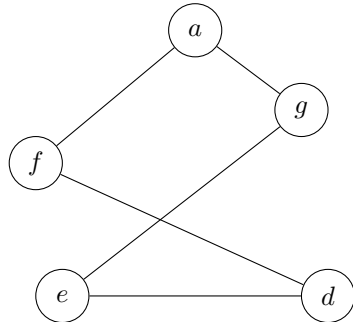
**Listas de adyacencias**

$a \rightarrow g \rightarrow \emptyset$
 $b \rightarrow c \rightarrow g \rightarrow \emptyset$
 $c \rightarrow b \rightarrow g \rightarrow \emptyset$
 $d \rightarrow \emptyset$
 $e \rightarrow \emptyset$
 $f \rightarrow \emptyset$
 $g \rightarrow a \rightarrow b \rightarrow c \rightarrow \emptyset$

Seguimos en el ciclo de la línea 8 pues g tiene todavía vértices en su lista de adyacencias, el primero de los cuales es a , por lo que tomamos la arista $g-a = a-g$.

- 8 $e = g - a;$
 9 $\mathcal{E} = \{b-c, b-g, c-g\};$
 10 $\mathcal{P} = [a, a-f, f, f-d, d, d-e, e, e-g, g, g-a, a];$
 11 $u = a;$

Eliminamos a a de la lista de adyacencias de g y a g de la lista de a :

**Listas de adyacencias**

$a \rightarrow \emptyset$
 $b \rightarrow c \rightarrow g \rightarrow \emptyset$
 $c \rightarrow b \rightarrow g \rightarrow \emptyset$
 $d \rightarrow \emptyset$
 $e \rightarrow \emptyset$
 $f \rightarrow \emptyset$
 $g \rightarrow b \rightarrow c \rightarrow \emptyset$

Como el vértice a ya no tiene a ningún vértice en su lista de adyacencias, no volvemos a entrar en el ciclo de la línea 8 y pasamos a ejecutar la línea 13. En esa línea tenemos que v es a , por lo que sustituimos a a en \mathcal{C} por todo el camino \mathcal{P} que se construyó:

$\mathcal{C} = [a]$ antes
 $\mathcal{C} = [a, a-f, f, f-d, d, d-e, e, e-g, g, g-a, a]$ después

y regresamos a la iteración de la línea 3. Como \mathcal{E} todavía tiene aristas, elegimos un vértice en \mathcal{C} que tenga aristas disponibles; este es el caso del vértice g , por lo que la ejecución se da como sigue:

- 4 $v = g;$
 6 $\mathcal{P} = [g];$
 7 $u = g;$

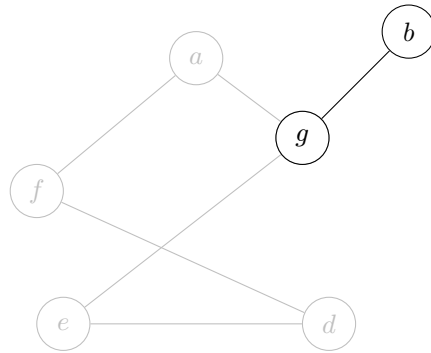
Llegamos nuevamente a la línea 8 del algoritmo, donde el primer vértice en la lista de adyacencias de g es b , por lo que elegimos a la arista $e = g - b = b - g$ y procedemos a ejecutar la iteración:

```

8   $e = g - b;$ 
9   $\mathcal{E} = \{b - c, c - g\};$ 
10  $\mathcal{P} = [g, g - b, b];$ 
11  $u = b;$ 

```

Eliminamos a g de la lista de adyacencias de b y a b de la de g :



Listas de adyacencias

```

 $a \rightarrow \emptyset$ 
 $b \rightarrow c \rightarrow \emptyset$ 
 $c \rightarrow b \rightarrow g \rightarrow \emptyset$ 
 $d \rightarrow \emptyset$ 
 $e \rightarrow \emptyset$ 
 $f \rightarrow \emptyset$ 
 $g \rightarrow c \rightarrow \emptyset$ 

```

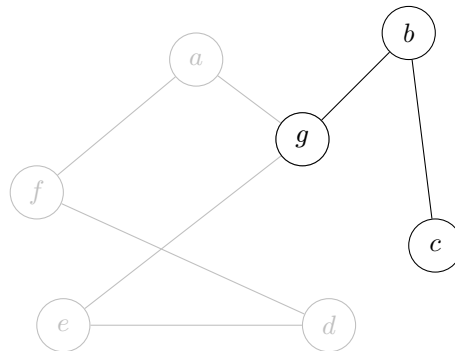
Regresamos a la línea 8 y como tenemos a c en la lista de adyacencias de b , entramos nuevamente a la iteración:

```

8   $e = b - c;$ 
9   $\mathcal{E} = \{c - g\};$ 
10  $\mathcal{P} = [g, g - b, b, b - c, c];$ 
11  $u = c;$ 

```

Eliminamos a b de la lista de adyacencias de c y a c de la de b :



Listas de adyacencias

```

 $a \rightarrow \emptyset$ 
 $b \rightarrow \emptyset$ 
 $c \rightarrow g \rightarrow \emptyset$ 
 $d \rightarrow \emptyset$ 
 $e \rightarrow \emptyset$ 
 $f \rightarrow \emptyset$ 
 $g \rightarrow c \rightarrow \emptyset$ 

```

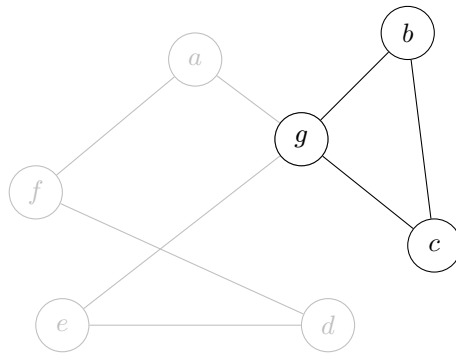
Regresamos a verificar la condición en la línea 8 y vemos que en la lista de adyacencias de c está el vértice g , por lo que tomamos la arista $e = c - g = g - e$ y entramos al ciclo nuevamente.

```

8   e = c - g;
9   E = { };
10  P = [g, g - b, b, b - c, c, c - g, g];
11  u = g;

```

Eliminamos a g de la lista de adyacencias de c y a c de la de g :



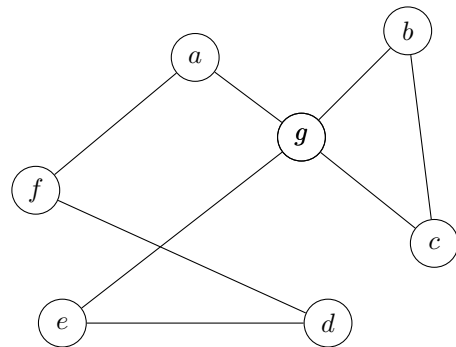
Listas de adyacencias

$a \rightarrow \emptyset$
 $b \rightarrow \emptyset$
 $c \rightarrow \emptyset$
 $d \rightarrow \emptyset$
 $e \rightarrow \emptyset$
 $f \rightarrow \emptyset$
 $g \rightarrow \emptyset$

Al regresar a la línea 8 vemos que ya no hay ningún vértice en la lista de adyacencias de g , por lo que pasamos a ejecutar la línea 13 del algoritmo. En esa línea tenemos que v es g , por lo que sustituimos a g en C por todo el camino P que se construyó:

$C = [a, a - f, f, f - d, d, d - e, e, e - g, \boxed{g}, g - a, a]$ antes
 $C = [a, a - f, f, f - d, d, d - e, e, e - g, \boxed{g, g - b, b, b - c, c, c - g, g}, g - a, a]$ después

y regresamos a la iteración de la línea 3. Como \mathcal{E} ya está vacía, salimos de la iteración en la línea 15 y damos el circuito euleriano.

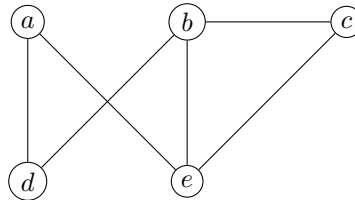


Listas de adyacencias

$a \rightarrow \emptyset$
 $b \rightarrow \emptyset$
 $c \rightarrow \emptyset$
 $d \rightarrow \emptyset$
 $e \rightarrow \emptyset$
 $f \rightarrow \emptyset$
 $g \rightarrow \emptyset$

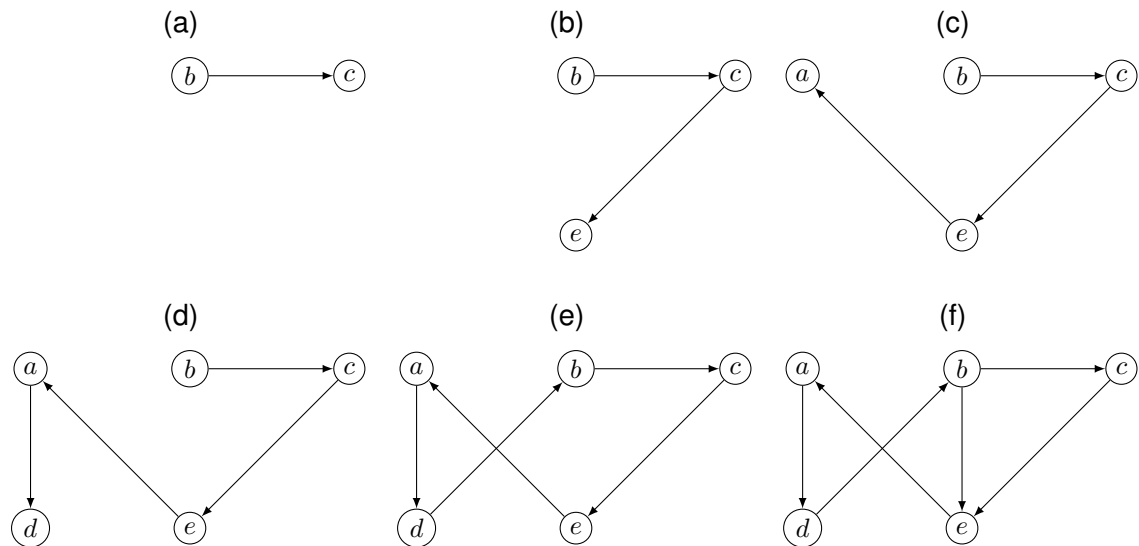
Si deseamos que el algoritmo construya un paseo euleriano, suponiendo que la gráfica tiene grado par en todos sus vértices excepto en exactamente dos, tenemos que modificar ligeramente el algoritmo para que el primer vértice seleccionado en las líneas 4 y 5 sea uno de los de grado impar. De otra manera va a tratar de construir un circuito que no existe. Veamos un ejemplo.

Ejemplo 6.2. Obtener el paseo euleriano de la siguiente gráfica.



Verificamos primero que la gráfica cumpla con la condición necesaria y suficiente para que tenga un paseo euleriano y así es. En esta ocasión únicamente iremos dibujando el paseo que se va formando. Lo mostraremos en la figura 6.6. Tenemos que empezar con el vértice b o con el e porque son los que tienen grado impar. Empezaremos en el vértice b . Suponemos que las listas de adyacencias están ordenadas lexicográficamente, por lo que la primera arista que vamos a agregar al paseo es $b—c$.

Figura 6.6 Construcción de un paseo euleriano



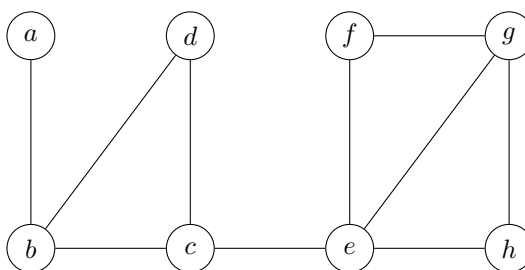
Exactamente hay dos vértices, b y e que tienen grado impar; el resto de los vértices tiene grado par. El camino empieza en el vértice b , que es uno de los que tiene grado impar;

recorre todas las aristas exactamente una vez; termina en el vértice e , que es el otro vértice con grado impar.

Podemos notar que el algoritmo que construye el circuito euleriano (paseo euleriano) le asigna una orientación a las aristas por el orden en que las va usando. La eficiencia del algoritmo va a depender de la representación interna de la gráfica y de cómo determinemos que una arista ya fue usada: quitarla de ambas listas de adyacencias no es muy eficiente y tal vez convendría más tener una lista de aristas en la que marcáramos aquellas que ya fueron usadas y no se pueden volver a usar.

El algoritmo del listado 6.1 obtiene un circuito euleriano. Si se le da una gráfica que no cumpla con los requisitos de tener grado par en todos sus vértices el algoritmo de todos modos va a trabajar, pero no va a encontrar un circuito euleriano. Veamos el siguiente ejemplo para ilustrar este aspecto.

Ejemplo 6.3. Tomemos la siguiente gráfica, que no cumple con tener grado par en todos sus vértices, por lo que no tiene un circuito euleriano; tampoco tiene exactamente dos vértices con grado impar, por lo que tampoco tiene un paseo euleriano.



Veremos el progreso del paseo en la figura 6.7. Tomamos como primer vértice a uno de los que tienen grado impar, por ejemplo a , y de ahí continuamos eligiendo aristas disponibles.

Figura 6.7 Construcción incorrecta de un paseo euleriano

1/2

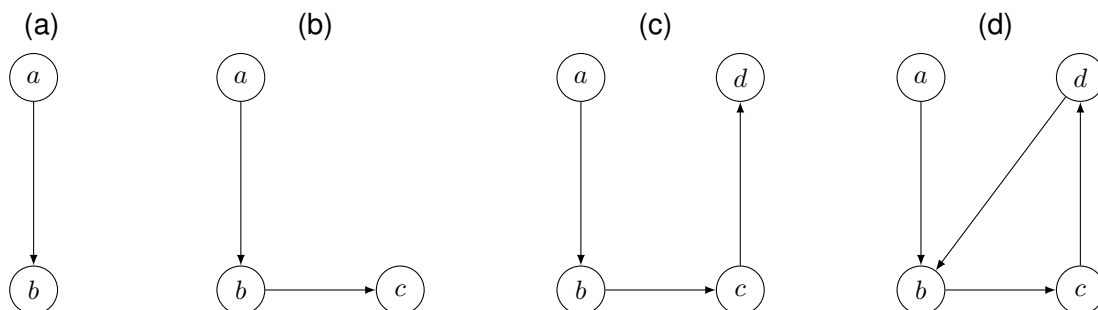
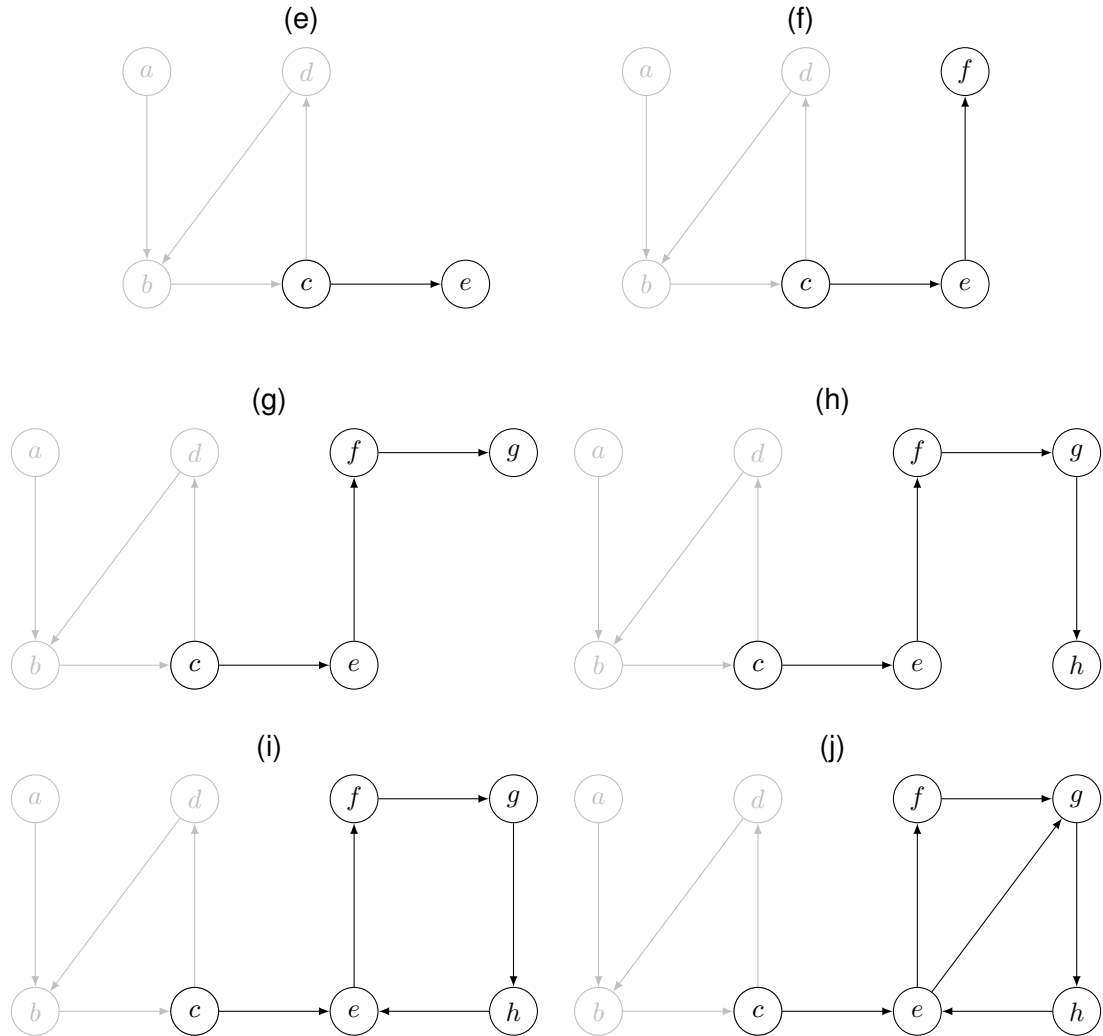


Figura 6.7 Construcción incorrecta de un paseo euleriano

2/2

En este momento ya no podemos seguir con el mismo paseo, así que la “guardamos” y tomamos al vértice c que es el único en el camino actual que tiene aristas disponibles.



En este punto ya no tenemos aristas disponibles en g por lo que debemos intentar juntar esta trayectoria a la primera; tenemos lo siguiente:

$$\mathcal{C} = \{a, a-b, b, b-c, \boxed{c}, c-d, d, d-b\}$$

$$\mathcal{P} = \{c, c-e, e, e-f, f, f-g, g, g-h, h, h-e, e, e-g, g\}$$

Sin embargo, si tratamos de sustituir al vértice c en \mathcal{C} por la trayectoria \mathcal{P} , nos vamos a

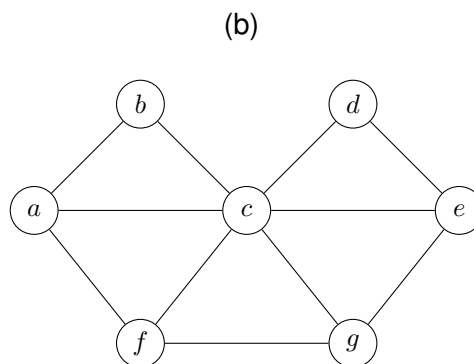
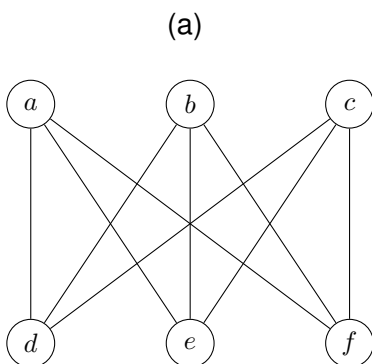
encontrar que lo que resulta no es un camino:

$$\dots b-c, c, c-e, \dots, e, \boxed{e-g, g, c-d} \dots;$$

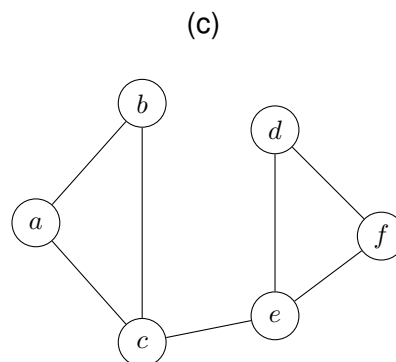
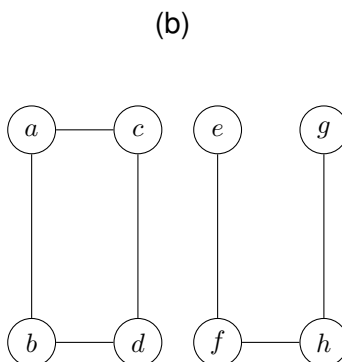
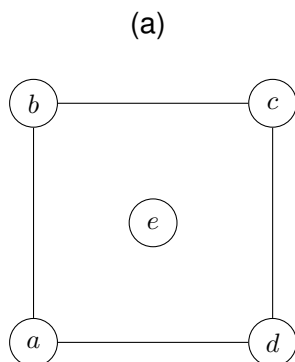
esto se debe a que el primer paseo euleriano que construimos termina en un vértice de grado impar (b), pero entonces requerimos que lo que se construya para ocupar el lugar de c sea un circuito euleriano que empiece y termine en c , lo que no sucede, pues en esa subgráfica tenemos tres vértices de grado impar. Si bien se recorrieron todas las aristas exactamente una vez, lo que construimos fueron dos caminos ajenos, no un paseo euleriano.

Ejercicios

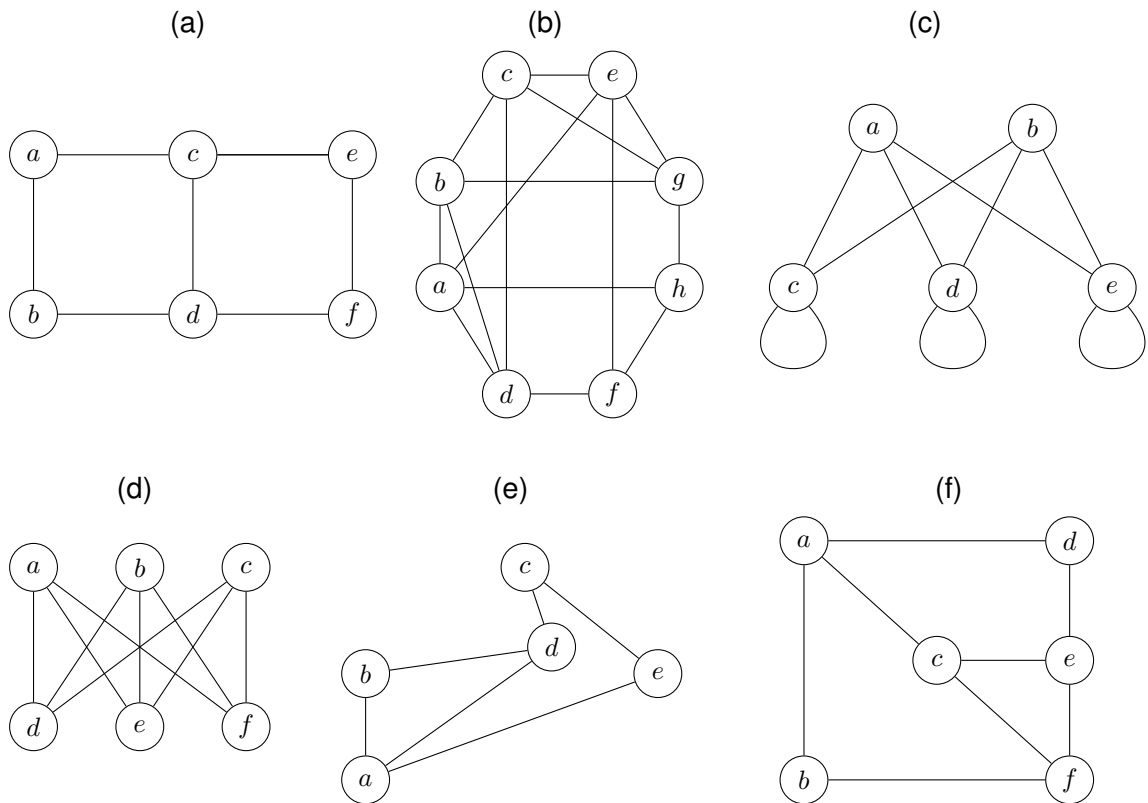
6.1.1.- En las gráficas que siguen, di cuántas trayectorias distintas hay entre los vértices a y d .



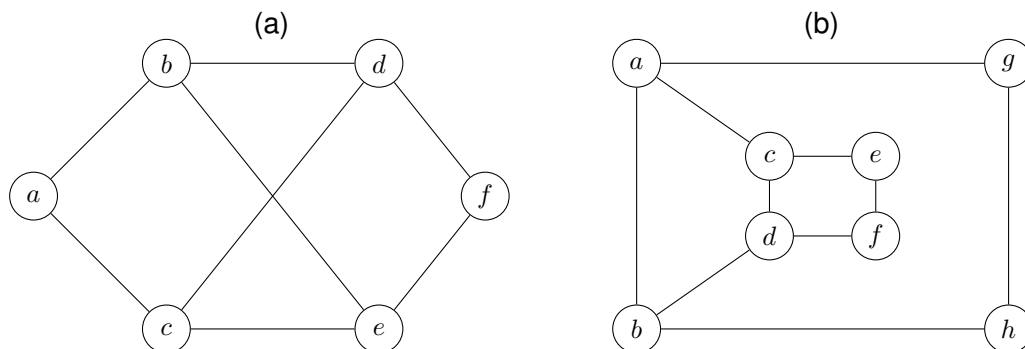
6.1.2.- Para las siguientes gráficas, determina si son gráficas conexas.

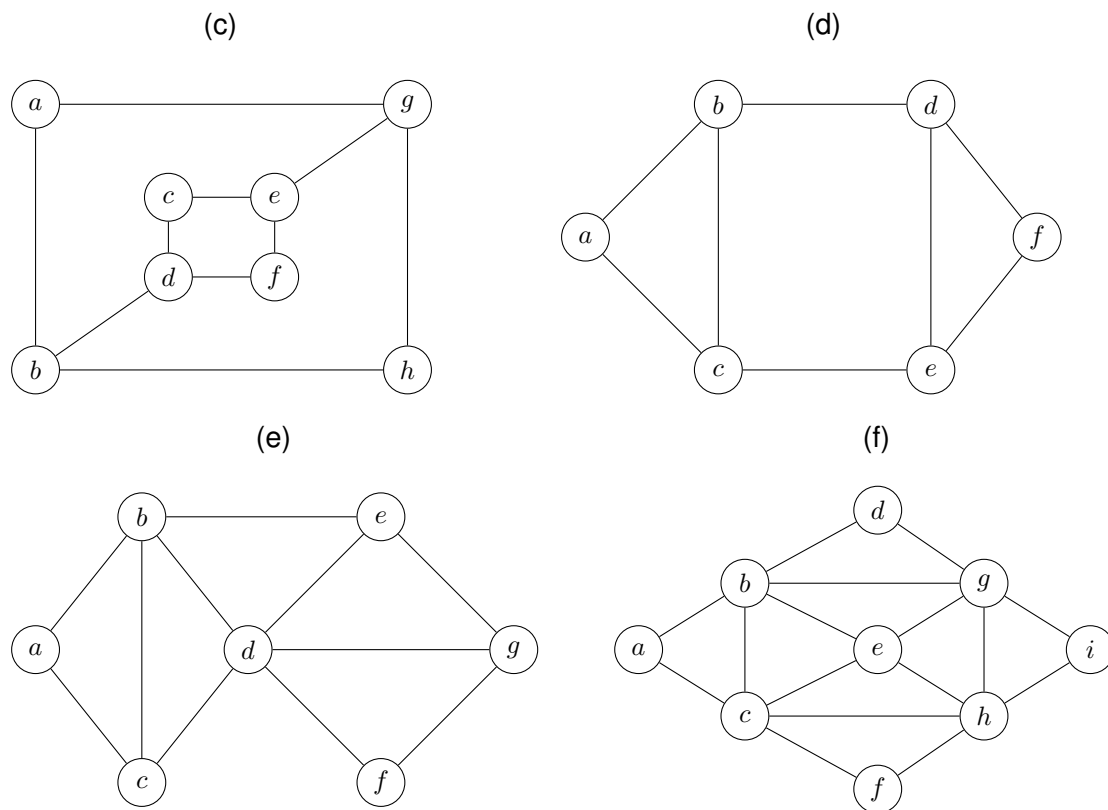


6.1.3.- ¿Cuáles de las siguientes gráficas tienen circuito/paseo euleriano? Si es que tienen alguno de los dos, aplicar el algoritmo 6.1 para encontrarlo, dando cómo se va armando el circuito/paseo durante la aplicación del algoritmo.



6.1.4.- Para cada una de las siguientes gráficas, que no tienen circuito euleriano ni paseo euleriano, di cuántas aristas tendrías que agregar y dónde para que la gráfica tuviera un circuito euleriano, si es que esto es posible. Di cuántas aristas tendrías que agregar y dónde para que la gráfica tuviera un paseo euleriano, si es que esto es posible.





6.2. Trayectorias hamiltonianas

Veamos ahora un problema muy parecido, el de explorar una gráfica con la restricción de que cada vértice sea visitado exactamente una vez. Bajo *visitado* queremos decir llegar y salir de él. Tenemos la restricción adicional de empezar y terminar el ciclo en el mismo vértice en el que empezamos. A este tipo de recorrido es a lo que se conoce como *ciclo hamiltoniano*.

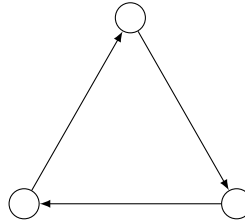
El problema es, entonces, dada una gráfica conexa determinar si existe o no un ciclo hamiltoniano en la gráfica. A pesar del gran parecido entre éste y el problema de los ciclos (o paseos) eulerianos, mientras que las condiciones necesarias y suficientes para que exista un ciclo euleriano están perfectamente definidas, éste no es el caso de los ciclos hamiltonianos. Inclusive, mientras que para construir el ciclo euleriano existe un algoritmo muy eficiente, para el ciclo hamiltoniano la única manera conocida de obtenerlo es calculando todos los posibles ciclos y ver cuál de ellos es hamiltoniano.

Sin embargo sí existen algunos resultados que nos pueden ayudar a determinar si una gráfica *pudiese* tener un ciclo o trayectoria hamiltoniana. Una condición *suficiente* para ello

se enuncia en el teorema 6.3, que se debe a Dirac.

Teorema 6.3 (Teorema de Dirac) Sea $G = (V, E)$ una gráfica conexa tal que $|V| = n$, $n > 2$. Si $\text{grado}(v) \geq \frac{n}{2}$, $\forall v \in V$, entonces G tiene un ciclo hamiltoniano.

Demostación. Si $n = 3$, como estamos pidiendo $\text{grado}(v) \geq \frac{n}{2} = 2$, G tiene que ser K_3 y tiene un ciclo hamiltoniano que empieza en cualquiera de los vértices y prosigue al siguiente.



Veamos que pasa con $n \geq 4$. Construyamos una trayectoria P tan larga como sea posible.



Observemos lo siguiente:

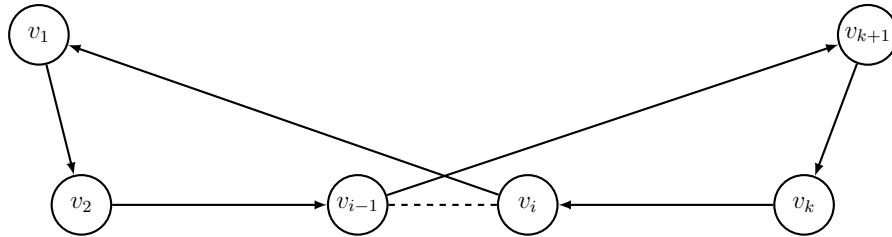
- Tenemos como hipótesis que $\text{grado}(v_1) \geq \frac{n}{2}$.
- No puede haber otro vértice fuera de P adyacente a v_1 , porque haría una trayectoria P' mayor que P , y lo mismo sucede con v_{k+1} , y supusimos que P era la más larga (o que no hay en G una trayectoria más larga).



- Quiere decir que todos los vértices a los que es adyacente v_1 están en P (lo mismo para v_{k+1}).
- Por lo que en P debe haber al menos $\frac{n}{2} + 1$ vértices, donde al menos $\frac{n}{2}$ son los vértices adyacentes a v_1 distintos de v_1 .

Si todos los vértices de V están en P , ya terminamos, pues P los toca a todos exactamente una vez.

Si no es así, procedemos de la siguiente manera. Supongamos ahora que en P existe un vértice v_i , $2 \leq i \leq k+1$, tal que v_i es adyacente a v_1 y v_{i-1} es adyacente a v_{k+1} . Si esto es así, tenemos el siguiente ciclo:



Demostraremos por contradicción que v_i y v_{i-1} deben existir con estas características.

- Supongamos que para toda $i \geq 2$ tal que v_i es adyacente a v_1 , tenemos que v_{i-1} no es adyacente a v_{k+1} .
- Como $\text{grado}(v_1) \geq \frac{n}{2}$ y ya demostramos que no puede haber ningún vértice adyacente a v_1 que no esté en P , hay al menos $\frac{n}{2}$ vértices adyacentes a v_1 en P ; para cada uno de estos vértice adyacentes a v_1 hay un vértice que lo precede; como estamos demostrando por contradicción, estamos suponiendo que ninguno de estos vértices v_{i-1} es adyacente a v_{k+1} . Por lo tanto en P hay $\frac{n}{2} + 1$ vértices no adyacentes a v_{k+1} (los inmediatos anteriores a v_i junto con v_{k+1}).
- También demostramos que todos los vértices adyacentes a v_{k+1} tienen que estar en P .
- Como la gráfica tiene n vértices, de los cuales $n - 1$ no son v_{k+1} , si contamos los vértices que quedan en P y que pueden ser adyacentes a v_{k+1} nos vamos a encontrar con que son $n - (\frac{n}{2} + 1) = \frac{n}{2} - 1 < \frac{n}{2}$. Por lo que tenemos:

$$\text{grado}(v_{k+1}) \leq \frac{n}{2} - 1.$$

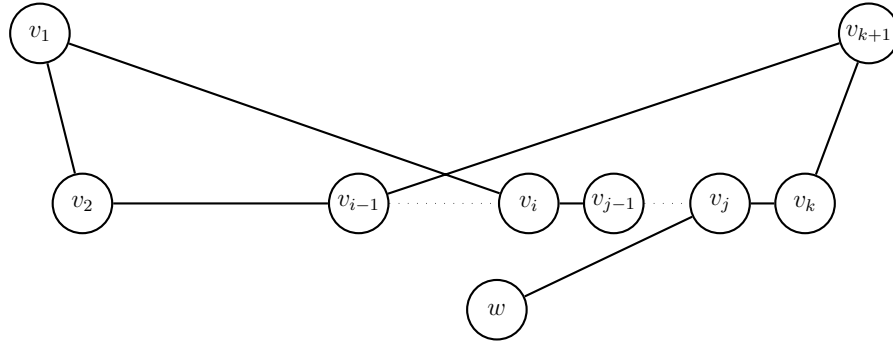
Pero esto es una contradicción respecto a que $\text{grado}(v) \geq \frac{n}{2}$. Por lo que **sí** existen v_i y v_{i-1} .

Dada esta situación, podemos construir un ciclo

$$C = v_1, v_2, \dots, v_{i-1}, v_{k+1}, v_k, \dots, v_i, v_1,$$

que contiene a todos los vértices de P . Si P contiene a todos los vértices de V , este ciclo nos da el ciclo hamiltoniano. Y debe ser así.

Supongamos que no y que existe un vértice $w \notin P$. A lo más hay $\frac{n}{2} - 1$ vértices de G fuera de P . Como $\text{grado}(w) \geq \frac{n}{2}$, w tiene que ser adyacente a algún vértice de P . Sea ese vértice v_j .



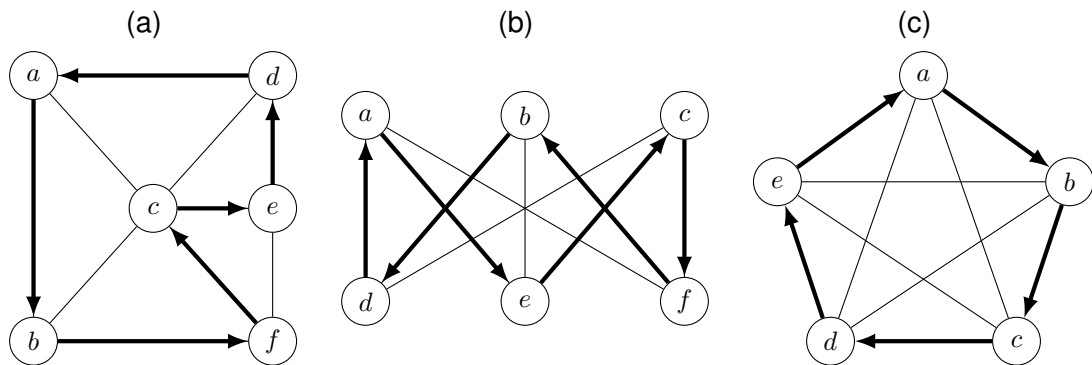
Pero entonces, la trayectoria

$$P' = w, v_j, v_{j+1}, \dots, v_k, v_{k+1}, v_{i-1}, \dots, v_2, v_1, v_i, \dots, v_{j-1}$$

es una trayectoria de longitud mayor que P (tiene una arista más, ya que sustituimos la arista $v_{i-1}v_i$ por las dos aristas $v_{i-1}v_{k+1}$ y v_1v_i ; también sustituimos a la arista $v_{j-1}v_j$ por la arista wv_j), a quien supusimos de longitud máxima. Por lo que no puede haber vértices de V fuera de P . \square

Veamos un ejemplo sencillo de gráficas que cumplen con la condición del teorema 6.3 y que, por lo tanto, tiene un ciclo hamiltoniano, en la figura 6.8.

Figura 6.8 Algunas gráficas con ciclos hamiltonianos que cumplen la condición de Dirac

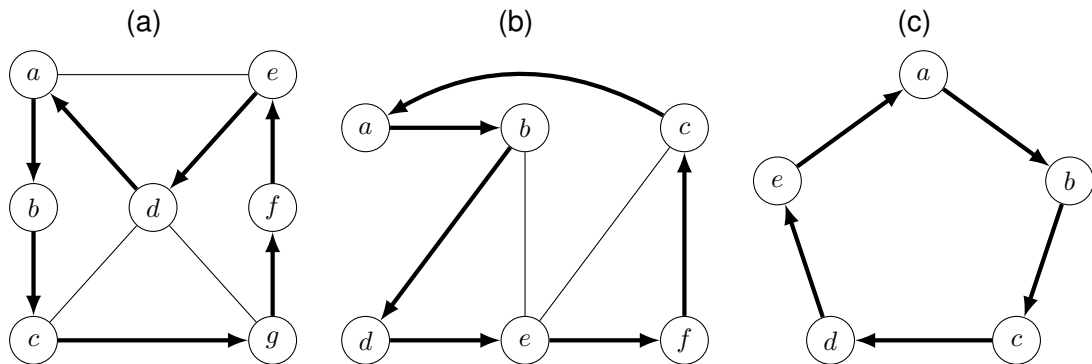


Si bien el teorema 6.3 garantiza la existencia de un ciclo hamiltoniano si el grado de cada vértice es mayor o igual a la mitad del número de vértices, el teorema no nos dice cómo encontrar ese camino. Esto fue evidente al construir los ciclos hamiltonianos de las primeras dos gráficas de la figura 6.8. En cambio, en el caso de los ciclos eulerianos, al demostrar el teorema dimos un método de construcción del ciclo. Es claro que todas las

gráficas completas tienen un ciclo hamiltoniano pues cumplen con la condición dada por el teorema.

Esta es una condición suficiente; esto es, podemos tener gráficas que no cumplan esta condición y que sin embargo sí tengan ciclos hamiltonianos, como las que se muestran en la figura 6.9.

Figura 6.9 Gráficas con ciclo hamiltoniano que no cumplen con la condición de Dirac



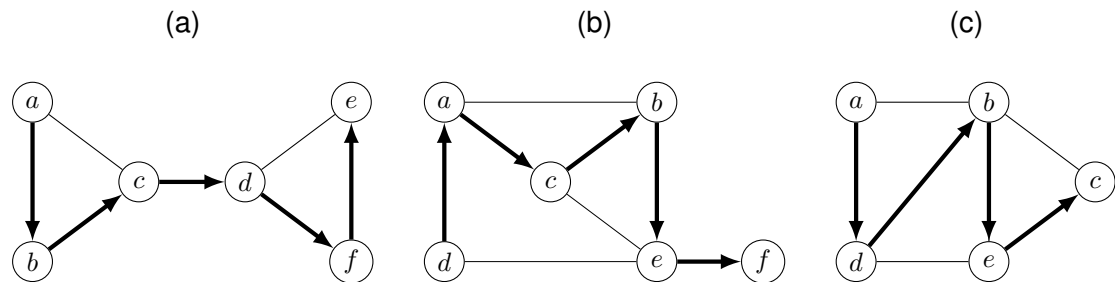
En estas tres gráficas hay vértices con grado menor a $\frac{n}{2}$ y sin embargo sí tienen un ciclo hamiltoniano. Las dos primeras, de hecho, tienen más de uno.

Otro teorema que da condiciones suficientes para que una gráfica tenga un ciclo hamiltoniano, pero que no demostraremos acá, es el Teorema de Ore:

Teorema 6.4 (Teorema de Ore) *Si G es una gráfica con n vértices, $n \geq 3$, tal que para todo par de vértices no adyacentes u y v en G , $\text{grado}(u) + \text{grado}(v) \geq n$, entonces G tiene un ciclo hamiltoniano.*

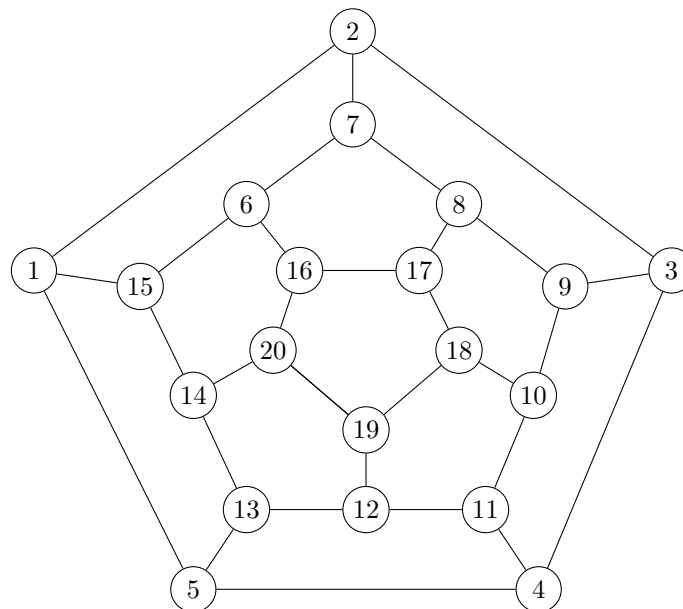
Nuevamente este teorema nos da una condición suficiente. Ninguna de las gráficas de la figura 6.9 cumple con la condición de Ore y sin embargo todas tienen un ciclo hamiltoniano.

De manera similar a como definimos un paseo euleriano podemos definir una trayectoria hamiltoniana como aquella que recorre a cada vértice exactamente una vez, pero a diferencia de un ciclo hamiltoniano, empieza y termina en distintos vértices. Es claro que toda gráfica que tiene un ciclo hamiltoniano tiene, asimismo, una trayectoria hamiltoniana: simplemente no usamos la última arista recorrida para el ciclo hamiltoniano. Pero podríamos tener una trayectoria hamiltoniana y que no hubiera arista entre el primer vértice de la trayectoria y el último. En la figura 6.10 tenemos varias gráficas que tienen una trayectoria hamiltoniana.

Figura 6.10 Gráficas con trayectorias hamiltonianas

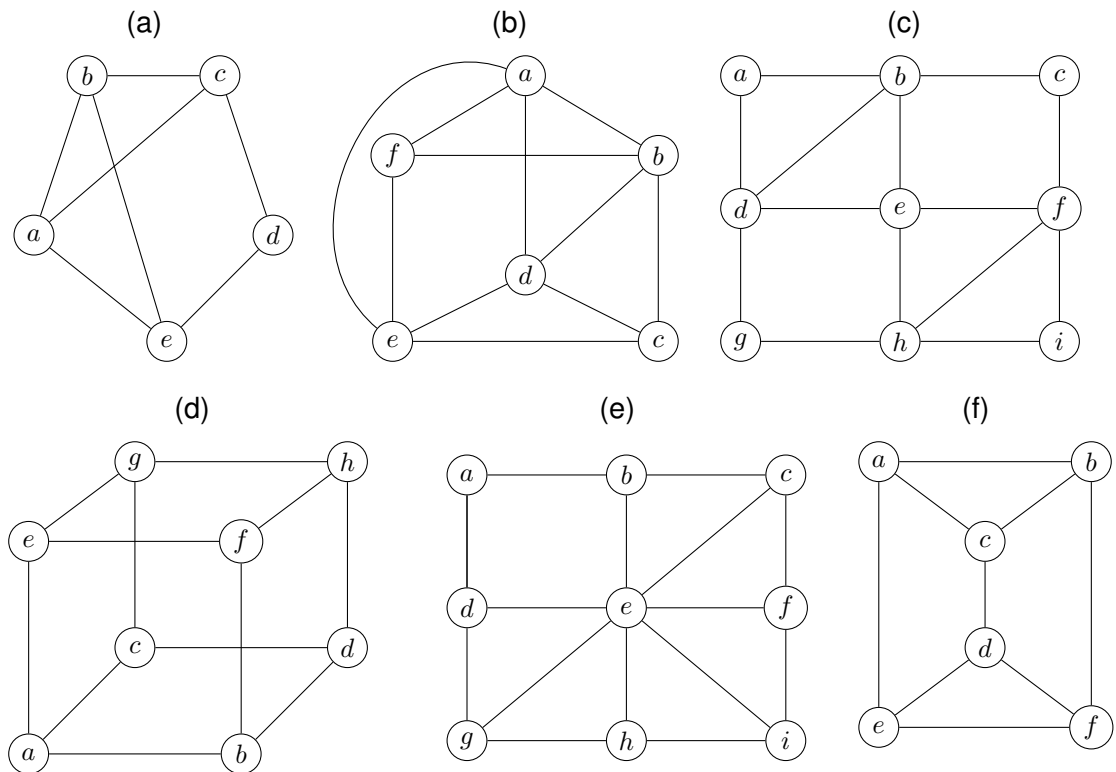
Ejercicios

6.2.1.- El matemático irlandés, Sir William R. Hamilton, a quien se debe la introducción del concepto de ciclo hamiltoniano, trató de comercializar un juego que consistía de un dodecaedro hecho de madera, donde cada esquina representaba a una ciudad famosa. El juego consistía en encontrar una ruta que visitara todas las ciudades exactamente una vez y regresara a la ciudad de origen o, lo que es lo mismo, encontrar un ciclo hamiltoniano en el dodecaedro. En el plano, el juego se pinta como sigue:

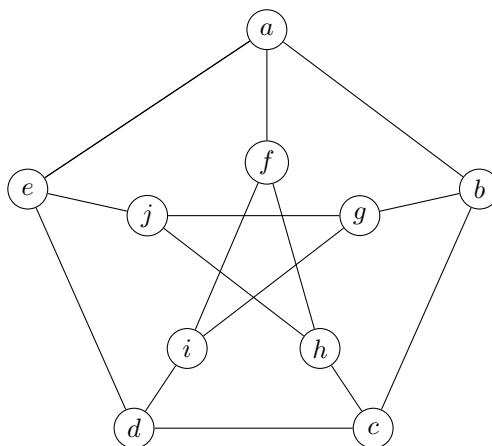


Encontrar un ciclo hamiltoniano (o todos) en el dodecaedro.

6.2.2.- De las siguientes gráficas di cuáles tienen un ciclo hamiltoniano o una trayectoria hamiltoniana. Si tiene alguno de estos dos, dibújalo.

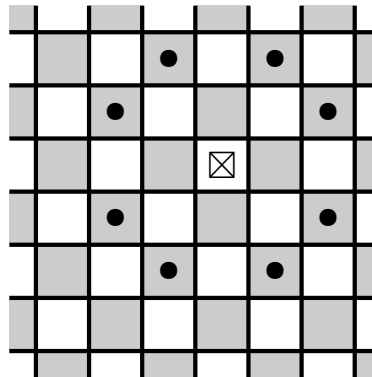


6.2.3.- La siguiente gráfica, conocida como la gráfica de Petersen, no tiene un ciclo hamiltoniano, pero si se le quita cualquiera de sus vértices y las aristas incidentes en él, la subgráfica así obtenida sí tiene un ciclo hamiltoniano. Muestra que estas dos aseveraciones son verdaderas.



6.2.4.- Un *caballo* es una pieza de ajedrez que se puede mover dos cuadros horizontales y uno vertical, o un cuadro horizontal y dos verticales. Esto es, un caballo en el

cuadro (x, y) puede moverse a cualquiera de las siguientes posiciones $(x \pm 2, y \pm 1)$, $(x \pm 1, y \pm 2)$, si estos cuadrados se encuentran en el tablero. Veamos un ejemplo en un tablero de 6×6 . La posición del caballo la mostraremos con \otimes y las posibles posiciones a las que se puede mover con \bullet .



Un *paseo equino* es una secuencia de movimientos legales de un caballo que empieza en alguna posición y visita cada cuadro del tablero exactamente una vez. Un paseo equino es *reentrante* si hay un movimiento legal que lleva al caballo desde el último cuadro del paseo a donde inició el mismo. Podemos modelar paseos equinos usando una gráfica que tiene un vértice para cada cuadro en el tablero, con una arista entre dos vértices si el caballo se puede mover legalmente entre los cuadrados representados por estos vértices.

- (a) Dibuja la gráfica que representa los movimientos válidos de un caballo en un tablero de 3×3 .
- (b) Dibuja la gráfica que representa los movimientos válidos de un caballo en un tablero de 3×4 .
- (c) Muestra que encontrar un paseo equino en un tablero de $m \times n$ es equivalente a encontrar una trayectoria hamiltoniana en la gráfica que representa los movimientos legales del caballo en el tablero.
- (d) Muestra que encontrar un paseo equino reentrante en un tablero de $m \times n$ es equivalente a encontrar un ciclo hamiltoniano en la gráfica que representa los movimientos legales del caballo en el tablero.

6.2.5.- Se dice que una gráfica es euleriana (hamiltoniana) si tiene un ciclo euleriano (hamiltoniano, respectivamente). Da un ejemplo de gráfica que exactamente sea

- (a) euleriana y hamiltoniana,
- (b) euleriana y no hamiltoniana,

- (c) no euleriana y hamiltoniana,
- (d) no euleriana y no hamiltoniana.

6.3. Distancias en una gráfica

En esta sección exploraremos una gráfica para obtener caminos entre vértices y ver cuál es su longitud. Primero veremos gráficas no dirigidas sencillas, donde suponemos que todas las aristas tienen un costo (o peso) uniforme de una unidad. En la siguiente sección exploraremos el concepto de una gráfica con pesos, que es aquella donde cada arista tiene un costo asociado. A continuación definimos el concepto de *distancia*, que es el que nos ocupará por lo pronto.

Definición 6.11 (distancia) Sea $G = (V, E)$ una gráfica no dirigida. La distancia entre dos vértices u y v en V , denotada por $\delta(u, v)$, es la longitud de la trayectoria más corta entre u y v . Definimos $\delta(u, v) = \infty$ si no existe trayectoria entre u y v .

Hay tres modalidades bajo las cuáles podemos querer obtener distancias en una gráfica:

- i. Trayectoria más corta entre dos vértices s y t .
- ii. Trayectoria más corta desde un vértice origen a todos y cada uno de los vértices en la gráfica.
- iii. Trayectorias más cortas entre todas las posibles parejas de vértices u y v en una gráfica.

El algoritmo que presentaremos a continuación ejecuta una exploración en amplitud (*Breadth First Search*) y por esta característica se le conoce como *BFS*. El algoritmo elige a un vértice como origen de la exploración, y “cuelga” la gráfica de ese vértice, obteniendo la distancia desde él a todos y cada uno de los vértices a los que se puede llegar desde s . Una vez elegido (o determinado) el vértice origen s , procede a avanzar de un vértice a otro por capas de adyacencia: primero a los vértices adyacentes a s – que están a una arista de s – definiendo una primera capa de vértices a distancia 1; después a los que están adyacentes a los vértices de la primera capa y que no se encuentran ya en ella y que están a distancia 2; y así sucesivamente. Al ir calculando las distancias registra también la trayectoria más corta que lleva a ese vértice mediante un atributo π que se refiere al vértice desde el cual se le visitó por primera vez.

Al construir estas capas de adyacencia, la distancia desde s a un vértice está dada por la capa que le corresponde la primera vez que se le alcanza (cuando se le descubre).

Esta exploración calcula naturalmente la distancia del vértice seleccionado como origen a cada uno de los vértices de la gráfica, por lo que resuelve la primera y segunda modalidad. Para resolver la tercera modalidad deberíamos ejecutar *BFS* desde cada uno de los vértices de la gráfica.

En los algoritmos que siguen usaremos notación orientada a objetos para referirnos a los distintos atributos (campos) que queremos tenga cada arista o vértice de la gráfica. Así, para referirnos a la distancia δ del vértice v al origen, lo denotaremos con $v.\delta$.

Definición 6.12 (Vértice alcanzable) Un vértice $v \in V$ es *alcanzable desde* u si es que existe alguna trayectoria entre u y v .

En una componente conexa todos los vértices son alcanzables desde cualquier otro vértice, mientras que si tenemos más de una componente conexa en una gráfica, ningún vértice de una de las componentes es alcanzable desde cualquier vértice en una componente conexa distinta.

Algoritmo BFS

Objetivo: Dada una gráfica $G = (V, E)$ y un vértice $s \in V$, encontrar $v.\delta$, que corresponde a la distancia del vértice s a v . Dejar marcada en la gráfica la trayectoria que corresponde a cómo se obtuvo esa distancia.

Datos: La gráfica $G = (V, E)$ y $s \in V$.

Salida: $\forall v \in V$, tal que haya una trayectoria $s \rightsquigarrow v$, reportar $v.\delta$ y $s \rightsquigarrow v$.

Estructuras de datos:

- i. La gráfica estará representada por listas de adyacencias.
- ii. Cada vértice tendrá un atributo $v.\pi$ que indica cuál es el vértice predecesor y un atributo $v.\delta$ que indica cuál es la distancia a s .

Método: Se encuentra en el listado 6.2.

Listado 6.2 Algoritmo BFS ($\delta(s, v), \forall v \in V$ alcanzable desde s)

(1/2)

```

firstnumber
1  /* ( Inicialización : ) */
2   $\forall v \in V : \{v.\pi \leftarrow \text{nulo}; v.\delta \leftarrow \infty\}$ 
3   $s.\delta \leftarrow 0; s.\pi \leftarrow \text{nulo}$ 
4   $\mathcal{C} \leftarrow \langle s \rangle$ 

```

Listado 6.2 Algoritmo BFS ($\delta(s, v), \forall v \in V$ alcanzable desde s)

(2/2)

```

5  /*(Procesar al que esté al frente de la cola.)*/
6  Mientras  $\mathcal{C} \neq \emptyset$ ,
7       $u = \text{frente}(\mathcal{C})$ 
8       $\forall v \in u.\text{adyacencias}$ :
9           $v \leftarrow$  siguiente vértice en la lista de adyacencias de  $u$ 
10         Si  $v.\delta == \infty$  /* Primera vez que se llega a él */
11              $v.\delta \leftarrow u.\delta + 1$ 
12              $v.\pi \leftarrow u$ 
13              $\mathcal{C} \leftarrow \mathcal{C} + v$ 
14         /* Fin:  $v.\delta == \infty$  */
15     /* Ciclo:  $\forall v \in u.\text{adyacencias}$  */
16      $\mathcal{C} \leftarrow \mathcal{C} - u$  /* Se saca a  $u$  de la cola */
17 /* Ciclo: Mientras  $\mathcal{C} \neq \emptyset$  */
18 /* Se reportan los resultados */
19  $\forall v \in V$ 
20     /*(Reporta distancia:)*/
21     Escribe: "La_distancia_a_v es " +  $v.\delta$ 
22     /*(Reporta camino)*/
23     Repite:
24         Reporta  $v$ 
25          $v \leftarrow v.\pi$ 
26     hasta que  $v == \text{nulo}$ 
27 /* Ciclo:  $\forall v \in V$  */

```

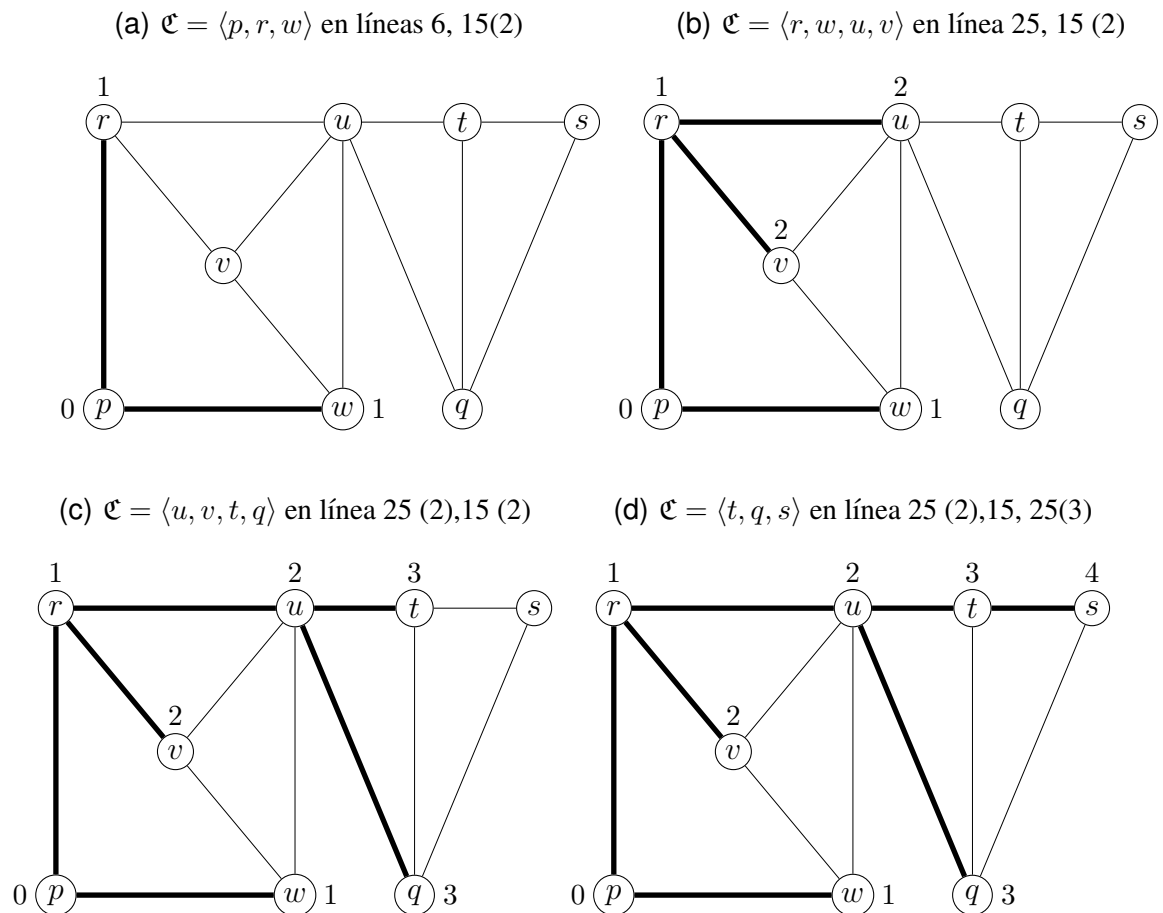
Ejemplo 6.4. Si seguimos el algoritmo BFS en la gráfica de la figura 6.11, veremos que los vértices de la gráfica van llegando a la cola \mathcal{C} en el orden en que se muestra en esa misma figura. Este orden es el que define la exploración.

Podemos observar en esta figura cómo se van definiendo las distancias de los vértices.

- En la subfigura 6.11(a) se calculan las distancias de p , el origen de la exploración, a r y w y se determina que su distancia es 1.
- En ese momento al frente de la cola se encuentra r , que es adyacente a u y v . Se calcula la distancia para estos dos vértices (2) y se quita de la cola a r .
- En ese momento se encuentra w al frente de la cola, con sus dos aristas sin usar que van a u y v ; pero como estos dos vértices ya fueron alcanzados antes (su atributo δ es distinto de ∞), estas aristas no se incluyen en la subgráfica que estamos armando.
- Quitamos a w del frente de la cola y surge u al frente de la misma. Las aristas de u que no han sido usadas son las que van a t y a q . A ambos vértices se les anota la distancia, se meten a la cola y se saca a u .
- Queda v al frente de la cola, pero como ya fueron exploradas sus aristas a u y a w , simplemente se saca de la cola a v .

- A continuación queda t al frente de la cola y tiene dos aristas sin usar, la que va a s y la que va a q . La que va a q se desecha porque a q se le asignó ya su distancia; se registra la distancia a s , que es 4, y se mete a s a la cola.
- Se saca a t de la cola, quedando q al frente de la cola. La única arista que tiene q sin usar es la que va a s , que ya fue procesado, por lo que simplemente se quita a q de la cola. Llega s al frente de la cola y como ya no tiene aristas incidentes sin usar, se le saca de la cola y la cola queda vacía.

Figura 6.11 Exploración BFS de una gráfica conexa desde el vértice p



Una característica de la exploración BFS en gráficas no dirigidas es que determina si la gráfica tiene o no ciclos. Si durante el recorrido alguna arista que se está explorando termina en un vértice que ya fue visitado, quiere decir que la gráfica contiene un ciclo. Esto se puede ver claramente porque hay dos trayectorias distintas que llevan desde el vértice origen hasta el vértice de que se trata. Por ejemplo, en la gráfica de la figura 6.11, cuando

se tiene en la cabeza de la cola al vértice w , al tratar de explorar la arista hacia u , encuentra este vértice ya marcado – se descubrió desde el vértice r –. Podemos ver que la trayectoria $u-r-p-w$ formaría una trayectoria que se cerraría formando un ciclo si se agregara la arista wu .

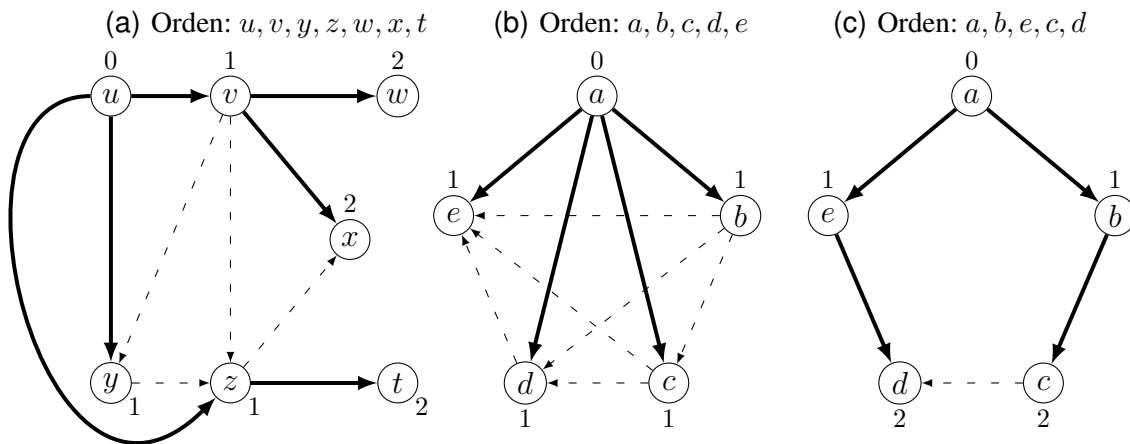
Podemos observar que la subgráfica generadora de la gráfica (incluye a todos los vértices) que se construye durante la ejecución de BFS es un árbol, pues es una subgráfica conexa y no tiene ciclos.

Cada recorrido BFS genera una subgráfica generadora $G' = (V', E')$, que contiene a todos los vértices de la gráfica original, si es que ésta es conexa (V' corresponde a todos los vértices alcanzables desde el vértice origen).

En el caso del recorrido BFS nunca se tiene que deshacer una trayectoria encontrada, pues éste queda determinado por la primera arista que descubra a un vértice.

Veamos algunos ejemplos más en la figura 6.12. En las gráficas de esta figura damos dirección a las aristas para mostrar el sentido de la exploración.

Figura 6.12 Ejemplos de recorridos BFS



Ejercicios

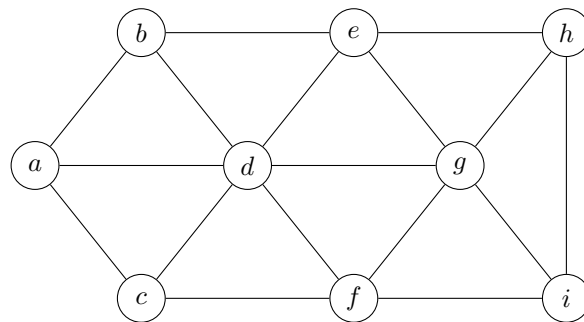
6.3.1.- Demostrar que la distancia definida para gráficas cumple con las tres propiedades que debe cumplir una medida de distancia y que son, a saber:

No negatividad: $\delta(x, y) \geq 0$; $\delta(x, y) = 0$ si y sólo si $x = y$.

simetría: $\delta(x, y) = \delta(y, x)$

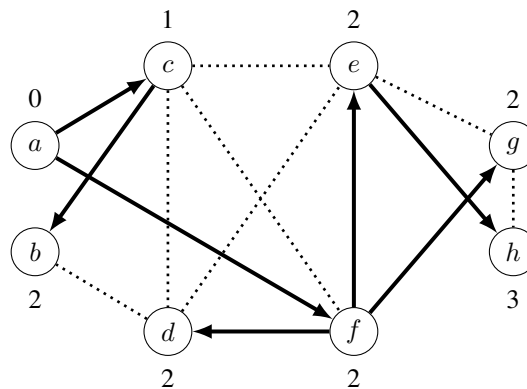
Desigualdad del triángulo: $\delta(x, z) \leq \delta(x, y) + \delta(y, z)$.

6.3.2.- Tenemos la siguiente gráfica, con su codificación en listas de adyacencias a su derecha. Da el recorrido BFS de la gráfica que se produce usando el algoritmo del listado 6.2 y la representación dada para la gráfica en listas de adyacencias.

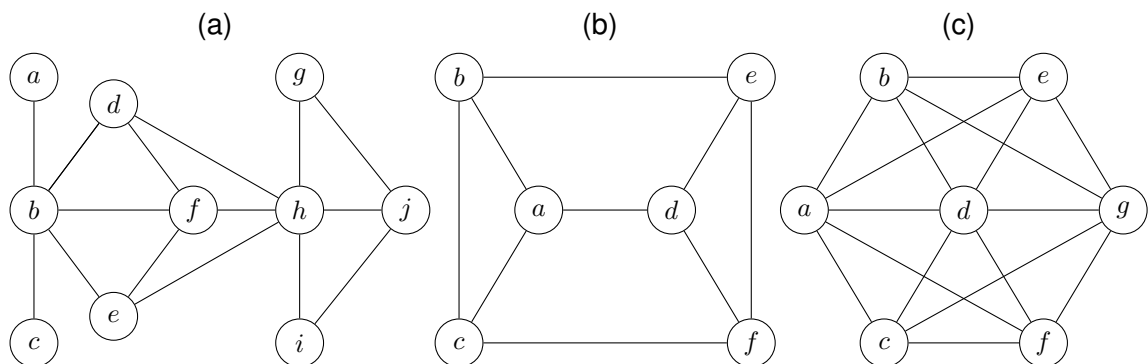


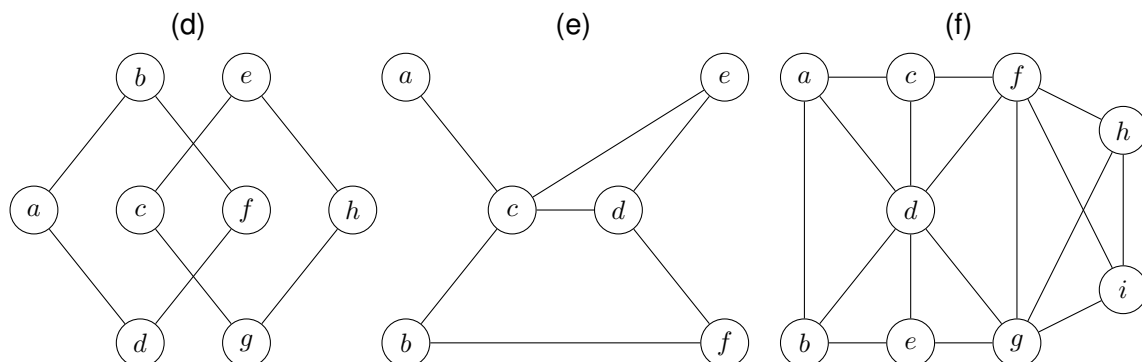
$a \rightarrow b \rightarrow c \rightarrow d$
 $b \rightarrow a \rightarrow e \rightarrow d$
 $c \rightarrow a \rightarrow d \rightarrow f$
 $d \rightarrow a \rightarrow b \rightarrow c \rightarrow e \rightarrow f \rightarrow g$
 $e \rightarrow b \rightarrow d \rightarrow g \rightarrow h$
 $f \rightarrow c \rightarrow d \rightarrow g \rightarrow i$
 $g \rightarrow e \rightarrow f \rightarrow h \rightarrow i \rightarrow d$
 $h \rightarrow e \rightarrow g \rightarrow i$
 $i \rightarrow h \rightarrow f \rightarrow g$

6.3.3.- Dado el recorrido BFS marcado con aristas sólidas (el resto de las aristas son punteadas), determinar las listas de adyacencias que hicieron posible este recorrido.

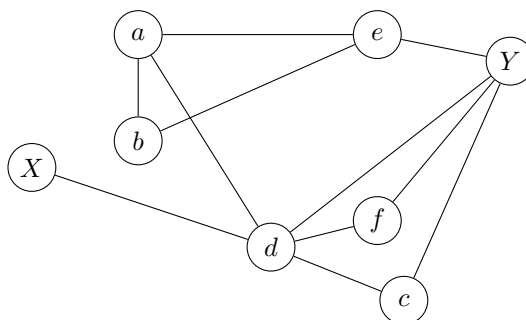


6.3.4.- Da las distancias de cada uno de los vértices al vértice a , en las gráficas que siguen.





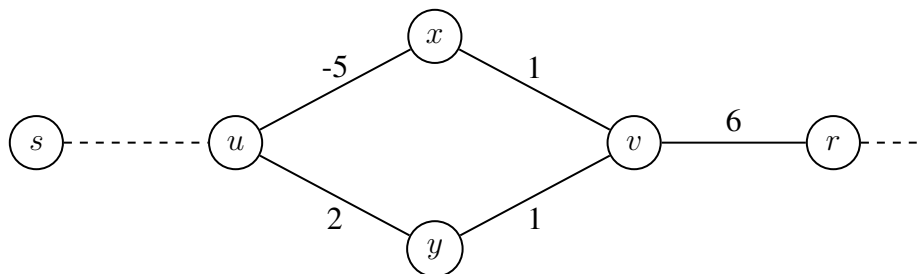
6.3.5.- La siguiente gráfica representa una ciudad en la que existen dos depósitos de gas, marcados con los símbolos X e Y . El dueño de los depósitos desea saber desde cuál depósito puede alcanzar a todos los clientes, representados por vértices, al menor costo posible. Las aristas representan caminos entre los tanques de gas que se deben llenar. Proporciona al dueño del depósito la respuesta.



6.4. Trayectorias más cortas

Muchas veces las aristas de una gráfica tienen costos asociados, como pudiera ser la distancia entre dos ciudades, el tiempo que se lleva recorrerla o el precio que hay que pagar por acceder a ella. En este caso tenemos lo que se conoce como una *gráfica con pesos* (en inglés, *weighted graph*) $G = (V, E; w)$, donde w es una función que asigna a cada arista (o arco) un valor. Este valor puede ser entero, real, o cualquier otro tipo que se pueda agregar. Cuando este es el caso, la distancia entre el vértice u y el vértice v se define como la menor suma posible de los pesos de las aristas de las trayectorias entre u y v .

Sin embargo, como w puede ser cualquier valor, podríamos encontrarnos con un caso como el que se muestra en la Figura 6.13.

Figura 6.13 Ciclos negativos en una gráfica con pesos

En el caso de esta gráfica con pesos, la distancia de u a r , por el camino u, x, v, r es 2; por el camino u, y, v, r es 9. Sin embargo, si tomamos el camino u, x, v, y, u, x, v, r la distancia es 1. Si recorremos el ciclo n veces la distancia se va acortando, y no hay una cota inferior para ello. Decimos entonces que no podemos calcular la distancia de u a r y le asignamos el valor ∞ .

Tenemos algoritmos que logran detectar si existe algún ciclo negativo en la gráfica y entonces no calculan las distancias. En el caso del algoritmo de Dijkstra para trayectorias más cortas, el algoritmo supone que la gráfica no tiene ningún ciclo negativo; más aún, que todas las aristas tienen pesos positivos o cero. Si se presentan pesos negativos en las aristas, aunque no excluyen por sí solos que la distancia entre los vértices esté definida, no es posible demostrar la correctud del algoritmo de Dijkstra, demostración que no enfrentaremos en este texto.

Algoritmo de Dijkstra para trayectorias más cortas

El algoritmo de Dijkstra para trayectorias más cortas encuentra la distancia entre un vértice origen s y todos aquellos vértices alcanzables desde s . Como acabamos de mencionar, este algoritmo trabaja únicamente con gráficas cuyas aristas tengan pesos no negativos. Es un algoritmo bastante eficiente, conocido como glotón o ávido (*greedy*), ya que en cada momento toma la decisión local más conveniente. Para ello mantiene lo que se conoce como una cola de prioridades, manteniendo a la cabeza de la cola aquel vértice ya descubierto cuya distancia al origen sea la menor de entre los que se encuentran en la cola. De cierta manera trabaja muy parecido al algoritmo BFS, pues va estableciendo capas de vértices a una cierta distancia del vértice origen. La diferencia principal es que mientras que en BFS el número de aristas determina la distancia, en el algoritmo de distancias de Dijkstra lo que determina las distancias es la suma de los pesos de las aristas. De esta manera, un vértice que pudiese estar adyacente al origen, este camino bien pudiera pesar más que uno que estuviese a dos aristas del origen. A continuación se encuentra el algoritmo para trayectorias más cortas de Dijkstra.

Objetivo: Dada una gráfica $G = (V, E; w)$ tal que todos los pesos en las aristas son no negativos, y un vértice origen s , determinar $v.\delta$, la distancia del vértice origen a cada uno de los vértices alcanzables desde s .

Datos: La gráfica $G = (V, E; w)$ y $s \in V$.

Salida: El valor de $v.\delta, \forall v \in V$ tal que $\exists s \rightsquigarrow v..$

Estructuras de datos: Las mismas que para BFS, excepto que cada arista tiene asociado un peso. La cola ahora es una *cola de prioridades* (\mathcal{C}_P), donde podemos elegir al que tenga la mayor prioridad (la menor distancia anotada).

Método: Se encuentra en el listado 6.3.

Listado 6.3 Algoritmo de Dijkstra para distancias

```

1  /* Inicialización: */
2   $\forall v \in V$  :
3       $v.\delta \leftarrow \infty$ 
4       $v.\pi \leftarrow \text{nulo}$ 
5  /* Ciclo:  $\forall v \in V$  :
6   $s.\delta \leftarrow 0$ 
7   $\mathcal{C}_P \leftarrow \langle s \rangle$ 
8  /* Procesar al que esté al frente de la cola. */
9  Mientras  $\mathcal{C} \neq \emptyset$ 
10      $u \leftarrow v$  tal que  $v.\delta \leq x.\delta$ , para toda  $x \in \mathcal{C}$ 
11      $\forall v \in$  lista de adyacencias de  $u$ :
12          $v \leftarrow u.\text{primero}; w \leftarrow (u, v).\text{peso}$ 
13         Si  $v.\delta == \infty$ 
14              $v.\delta \leftarrow u.\delta + w$ 
15              $v.\pi \leftarrow u$ 
16              $\mathcal{C}_P \leftarrow \mathcal{C}_P + v$ 
17         /* Fin:  $v.\delta == \infty$  */
18         Si  $v.\delta > u.\delta + w$ 
19              $v.\delta \leftarrow u.\delta + w$ 
20              $v.\pi \leftarrow u$ 
21         /* Fin:  $v.\delta > u.\delta + w$  */
22     /* Ciclo:  $\forall v \in$  lista de adyacencias de  $u$  */
23      $\mathcal{C}_P = \mathcal{C}_P - u$ 
24 /* Ciclo: Mientras  $\mathcal{C} \neq \emptyset$  */
25
26 /* Reporta Resultados: */
27  $\forall v \in V$ 
28     /* (Reporta distancia) */
29     Escribe "Distancia_de_" +  $s$  + "_a_" +  $v$  + "_es_" +  $v.\delta$ 
30     /* (Reporta camino) */
31     Repite:
32         Reporta  $v$ 
33          $u \leftarrow u.\pi$ 
34     hasta que  $u == \text{nulo}$ 
35 /* Ciclo:  $\forall v \in V$  */

```

Ejemplo 6.5. Apliquemos el algoritmo a la gráfica de la figura 6.14 cuya representación con listas de adyacencias se encuentra en la tabla 6.1.

Figura 6.14 Algoritmo de Dijkstra para trayectorias más cortas

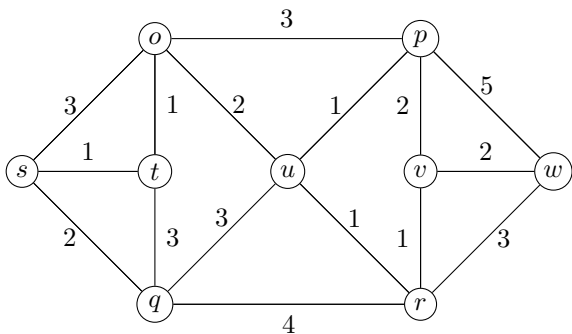
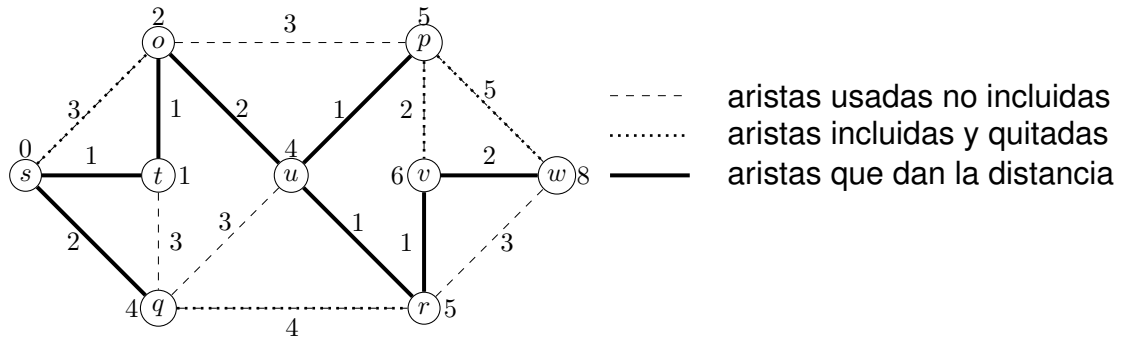


Tabla 6.1 Listas de adyacencias para la gráfica en la figura 6.14

vértice → vértices adyacentes: (peso(u, v), v)	
s	→ (3, o) → (1, t) → (2, q) → ∅
o	→ (3, s) → (1, t) → (2, u) → (3, p) → ∅
t	→ (1, o) → (1, s) → (3, q) → ∅
q	→ (2, s) → (3, t) → (3, u) → (4, r) → ∅
u	→ (2, o) → (3, q) → (1, p) → (1, r) → ∅
p	→ (3, o) → (2, v) → (5, w) → ∅
v	→ (2, p) → (1, r) → (2, w) → ∅
r	→ (4, q) → (1, u) → (1, v) → (3, w) → ∅
w	→ (5, p) → (2, v) → (3, r) → ∅

En la figura 6.15 mostramos a la derecha de la gráfica, arriba, el significado del tipo de línea que une a dos vértices y, abajo, en una tabla, la distancia con la que quedó cada vértice, medida desde el vértice *s*, y el orden en que fue llegando cada vértice al frente de la cola de prioridades. Este orden depende del orden de los vértices (o aristas) en las listas de adyacencias (incidencias) de cada vértice.

Figura 6.15 Distancias con el algoritmo de Dijkstra

v_i	s	o	t	q	u	p	r	w	v
δ	0	3, 2	1	2	4	5	6 , 5	10 , 8	7 , 6
orden	1	3	2	4	5	6	7	9	8

Las aristas se van tomando en el orden en que están en la lista de adyacencias de cada vértice cuando éste llega al frente de la cola de prioridades. La gráfica resultante es producto de este orden y el peso específico de las aristas.

En ocasiones podemos tener otra gráfica como resultado y esta situación se da cuando se puede llegar a un vértice desde dos vértices distintos pero con la misma distancia. Si las aristas se toman en orden distinto o cambiamos el orden de selección entre vértices que tienen la misma distancia y ésta los coloca al frente de la cola, podemos terminar con trayectorias del mismo tamaño pero que visitan diferentes vértices. Esto sucede, por ejemplo, si al elegir entre o y q , ambos con distancia 2, se elige al vértice que no se eligió en una ejecución anterior. También sucede entre los vértices p y r .

Veamos algunos ejemplos más del uso del algoritmos de distancias de Dijkstra. La cola de prioridades aparecerá en orden de prioridad, no en el orden en que ingresaron los vértices a la misma. Las listas de adyacencias se van a considerar ordenadas alfabéticamente.

Ejemplo 6.6. En la figura 6.16 de la siguiente página vemos nuevamente la ejecución del algoritmo de Dijkstra para trayectorias más cortas.

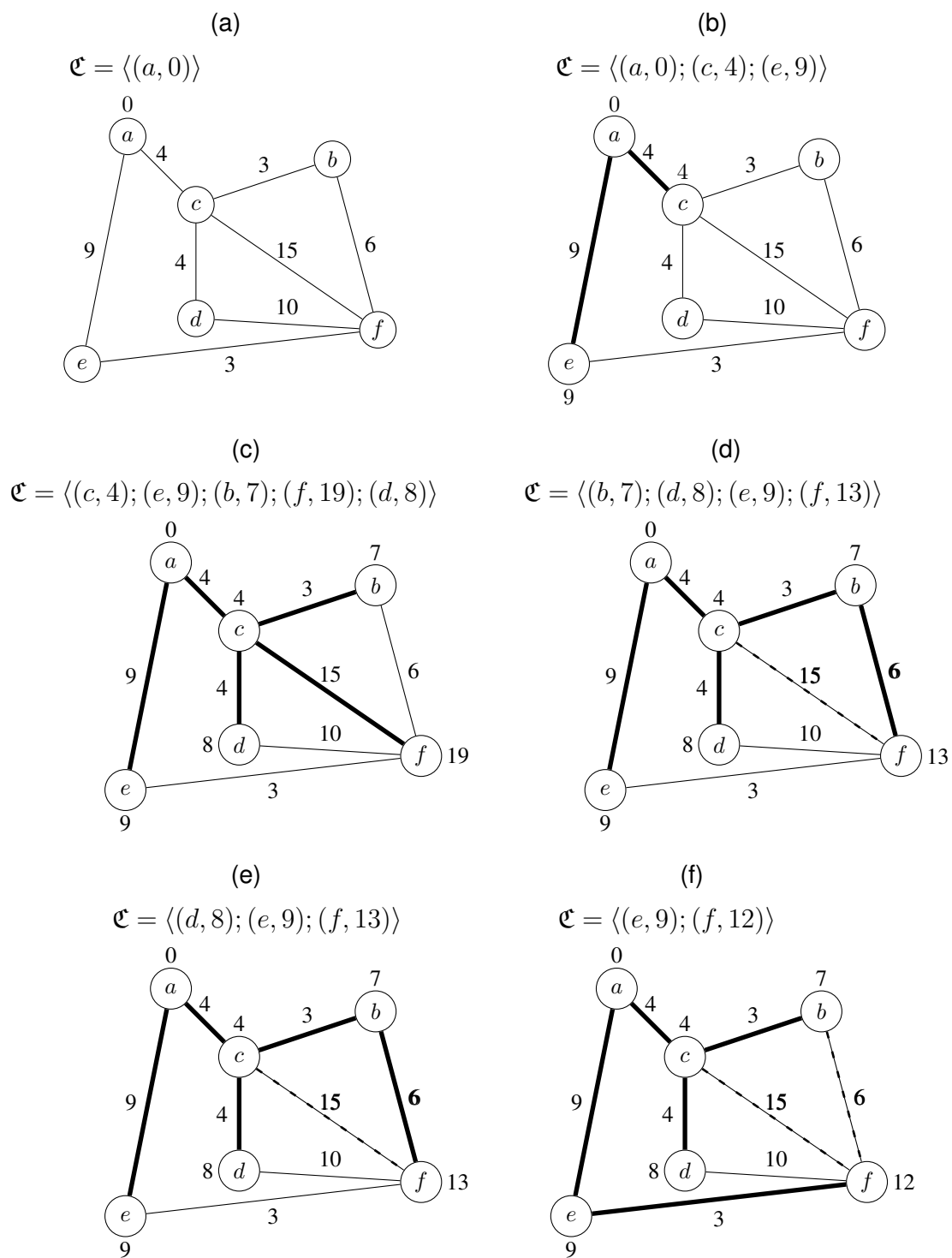
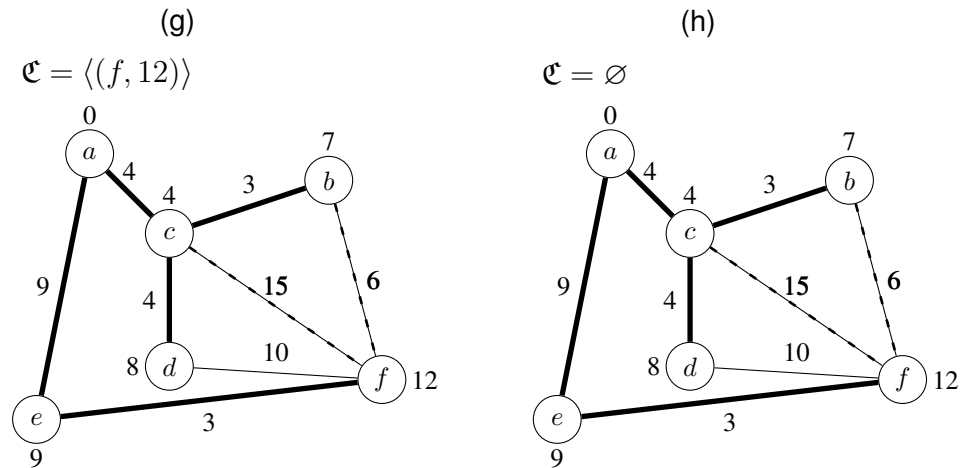
Figura 6.16 Trayectorias más cortas

Figura 6.16 Trayectorias más cortas

(continúa...)



En la figura 6.16 mostramos el progreso del algoritmo de Dijkstra para trayectorias más cortas.

- En la subfigura 6.16(a) se encuentra el estado de la gráfica y la cola de prioridades al iniciarse el ciclo, después de la inicialización. El vértice a , que es el origen, se encuentra como único elemento de la cola y su distancia es 0.
- Los vértices c y e entran a la cola de prioridades ya que son descubiertos desde a . Entran a la cola con su distancia a a . Esto se muestra en la subfigura 6.16(b).
- Se elimina a a de la cola y queda c al frente de la cola, ya que es el vértice con la distancia menor registrada que se encuentra en la cola. Desde c se descubre a b, d y f , que entran a la cola con sus distancias al vértice a , pero pasando por el vértice c . Este estado se muestra en la subfigura 6.16(c).
- En la subfigura 6.16(d) se muestra la salida del vértice c de la cola porque ya fue procesado; el vértice al frente de la cola es b , porque es el que presenta la menor distancia al vértice a de entre los que se encuentran en la cola; como el camino desde a a f que pasa por b es más corto que el que pasa por c , se corrige la distancia a f y la trayectoria más corta hasta el momento de a a f .
- Se elimina a b de la cola y queda como primero en la misma el vértice d . No se descubre a ningún vértice que no está ya en la cola, ni cambia ninguna de las trayectorias.
- Se elimina a d de la cola, quedando al frente de la misma el vértice e . La trayectoria de a a f que pasa por d es más corta que la que pasa por b , por lo que se corrige la distancia a f y la trayectoria, obligando a que ahora pase por e .
- Sale e de la cola, quedando f al frente de la misma y como único elemento. Desde f no hay ninguna trayectoria que corregir y no queda ya ningún vértice por descubrir que sea adyacente a f , por lo que en esta iteración no se hace nada.

- (h) Se saca a f de la cola, quedando ésta vacía, por lo que ya no se vuelve a entrar al ciclo.

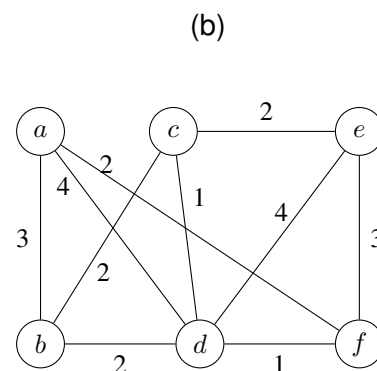
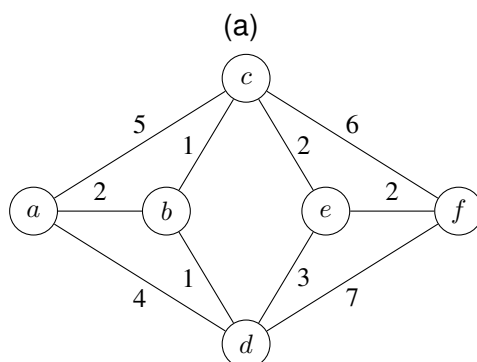
A diferencia del algoritmo BFS, el algoritmo de Dijkstra para trayectorias más cortas sí debe deshacer trabajo ya hecho. Sin embargo, este trabajo no es significativo, pues el peor caso se va a presentar si cada arista que se explora obliga a cambiar las distancias y la trayectoria. Pero tanto la distancia como la trayectoria representan un cambio en los campos δ y π de cada vértice, por lo que la construcción de la trayectoria y la definición de la distancia únicamente involucra, en el peor caso, dos pasos por cada arista en la gráfica.

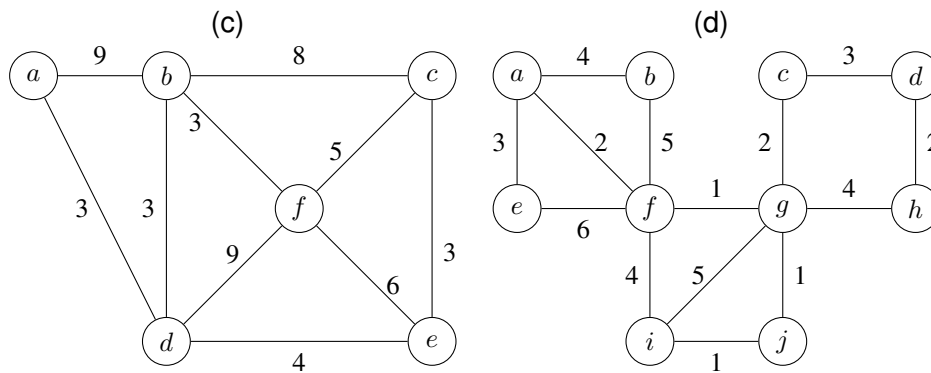
Pero notarán que en cada ciclo se tiene que salir del mismo con la cola de prioridades actualizada, esto es, con el vértice de menor distancia al frente. Podemos pensar en mantener la lista ordenada, pero esto es muy costoso, por lo que nos vamos a conformar con mover al vértice con la menor distancia al frente de la cola. Hay algoritmos eficientes que logran esto y depende de la estructura de datos que se elija para almacenar a la cola de prioridades. De cualquier forma, al costo de procesar los caminos y las distancias le debemos agregar el costo de mantener la cola de prioridades, que puede ser tan bajo como $n \log n$. Por lo que el costo de encontrar el camino con pesos en las aristas es más costoso que el problema similar con pesos homogéneos en las aristas.

Es importante notar también que una vez que un vértice llega al frente de la cola es porque ya alcanzó su trayectoria más corta. Por lo tanto, ninguna iteración va a modificar su distancia. Esto se demuestra en un curso de análisis de algoritmos (y es la razón por la que los pesos en las aristas deben ser no negativos).

Ejercicios

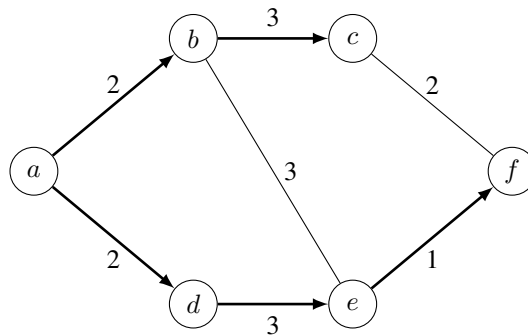
- 6.4.1.- Si alguna de las aristas tuviese peso negativo, ¿en qué punto fallaría el algoritmo de Dijkstra para trayectorias más cortas?
- 6.4.2.- En las siguientes gráficas, encuentra las trayectorias más cortas dadas por el algoritmo de Dijkstra para distancias.





6.4.3.- Modifica el algoritmo de distancias de Dijkstra para que produzca la distancia entre dos vértices dados en una gráfica con pesos en las aristas.

6.4.4.- Dado el siguiente árbol de distancias de Dijkstra que corresponde a una gráfica con pesos en las aristas, reconstruye las listas de adyacencias que fueron usadas al ejecutarse el algoritmo.



6.4.5.- Supongamos que queremos movernos en el sistema de transporte público de la ciudad. Describe a la gráfica con pesos en las aristas que modelan a los siguientes problemas:

- ¿Cuál es el menor tiempo requerido para viajar entre dos puntos?
- ¿Cuál es la menor distancia que se tiene que recorrer para viajar de un punto a otro?
- Suponiendo que cada uno de los tipos de transporte tiene un costo distinto, ¿cuál es la manera más económica de llegar de un punto a otro de la ciudad?

6.4.6.- ¿Se podría modificar el algoritmo de Dijkstra para que encuentre la trayectoria más larga entre cualesquiera dos puntos? Justifica tu respuesta.

6.4.7.- Si tenemos aristas con pesos negativos en una gráfica con pesos en las aristas, pero de tal manera que no haya ciclos negativos, ¿podemos modificar la gráfica de alguna manera para poder usar el algoritmo de Dijkstra para distancias? Justifica tu respuesta.

6.5. Número de caminos

Queremos saber el número de caminos de una cierta longitud que hay entre cualesquiera dos vértices o, simplemente, si existen caminos de una cierta longitud entre dos vértices. Esta información es importante cuando queremos operar sobre una red y deseamos saber qué tan conectados están unos vértices con otros.

6.5.1. Matrices de adyacencias

El siguiente teorema nos dice cómo obtener el número de caminos de una cierta longitud entre dos vértices cualesquiera.

Teorema 6.5 Sea $G = (V, E)$ con $|V| = n$ y los vértices etiquetados v_1, v_2, \dots, v_n . Sea A la matriz de adyacencias de G , con $a_{i,j}$ la entrada que corresponde a los vértices i y j . Entonces, el número de caminos de longitud m entre los vértices v_i y v_j está dado por $a_{i,j}$ en A^m , donde A^m corresponde a la multiplicación de la matriz A por sí misma m veces.

Demostración.

Demostraremos el teorema por inducción sobre m .

Base: Para $m = 1$, el número de caminos entre el vértice v_i y el vértice v_j de tamaño uno es 0 o 1, dependiendo de si hay una arista entre v_i y v_j . Esta es precisamente la definición de matriz de adyacencias.

Hipótesis de inducción: Supongamos que la matriz A^{m-1} contiene para cada pareja i, j el número de caminos distintos entre v_i y v_j de tamaño $m - 1$.

Paso inductivo: Veamos qué y cómo se calcula A^m , representándola como $a_{i,j}^m$.

$$a_{i,j}^m = \sum_{k=1}^n a_{i,k} \cdot a_{k,j}^{m-1}$$

$a_{i,k}$ nos dice si hay un camino de longitud 1 de v_i a v_k , y $a_{k,j}^{m-1}$ nos dice cuántos caminos hay de v_k a v_j de longitud $m - 1$. Para calcular el número de caminos de longitud m tratamos de extender los caminos de longitud $m - 1$, de la siguiente manera:

Para ir de v_i a v_j con un camino de longitud m , vemos cuáles vértices v_k tienen caminos de longitud $m - 1$ hacia v_j , y aumentamos el camino con todos los vértices v_k que tienen caminos de longitud 1 desde v_i . Si no hay arista de v_i a v_k el camino no puede ser extendido. Si hay, vamos a sumar todos los posibles caminos que cumplan con esto.

∴ En A^m queda el número de caminos de longitud m entre cualesquiera dos vértices. \square

Ejemplo 6.7. Fijémonos en la figura 6.17, donde la matriz A^1 corresponde simplemente a la matriz de adyacencias de la gráfica.

Debemos notar que esta forma de calcular el número de caminos entre cualesquiera dos vértices de una gráfica también nos puede llevar a decidir fácilmente si existe un camino entre cualesquiera dos vértices.

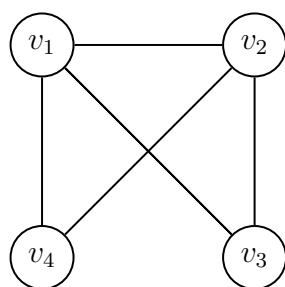
Para este último punto, podemos observar que si una gráfica tiene n vértices, lo más alejados que pueden estar dos vértices entre sí es por $n - 1$ aristas; por lo tanto, si tenemos una gráfica $G = (V, E)$ con n vértices y la matriz de adyacencias A para esa gráfica, la pregunta de si la gráfica es o no conexa se puede resolver obteniendo sucesivamente A^k , para $k = 1, 2, \dots, n - 1$, lo que nos daría, para cada k , si existen o no caminos de longitud k entre dos vértices. Como a lo más tenemos que calcular la potencia $n - 1$, si alguna entrada de la matriz se mantiene en 0 en todas estas matrices, quiere decir que esos vértices no están conectados.

Ejemplo 6.8. Si bien la matriz A^k (la matriz de adyacencias multiplicada por sí misma k veces) nos da el número de caminos de longitud k , pudiera ser que para una pareja particular de vértices no existiera ningún camino de longitud 2, por ejemplo, pero que sí existieran caminos de longitud 1 y 3. Tal es el caso de los vértices v_1 y v_2 en la gráfica de la figura 6.18.

Para obtener si hay una trayectoria cualquiera entre dos vértices hay que obtener, de alguna manera, la suma de todas estas matrices. Por ejemplo, A nos dice si están conectados por un camino de longitud 1, $A + A^2$ nos indica si hay caminos de longitud 2 o menores entre cualesquiera dos vértices; $A + A^2 + A^3$ si hay algún camino de longitud menor o igual a 3; y así sucesivamente. Podríamos definir a estas matrices de la siguiente forma:

$$\begin{aligned} A_1 &= A; \\ A_n &= A_{n-1} + A^n; \end{aligned}$$

donde A^n tiene el significado que le dimos en los párrafos anteriores. De esta manera, la entrada (i, j) de la matriz A_n (m_{ij}^n) nos indicaría el número de caminos de longitud menor o igual a n entre los vértices v_i y v_j de la gráfica.

Figura 6.17 Número de caminos de longitud 3

$$A^1: \begin{matrix} & \begin{matrix} v_1 & v_2 & v_3 & v_4 \end{matrix} \\ \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{matrix} & \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \end{matrix}$$

$$A^2: \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 3 & 2 & 1 & 1 \\ 2 & 3 & 1 & 1 \\ 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \end{pmatrix}$$

$$A^3: \begin{pmatrix} 3 & 2 & 1 & 1 \\ 2 & 3 & 1 & 1 \\ 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 4 & 5 & 5 & 5 \\ 5 & 4 & 5 & 5 \\ 5 & 5 & 2 & 2 \\ 5 & 5 & 2 & 2 \end{pmatrix}$$

Las matrices para este caso (corroborar si se desea) serían las que se encuentran en la figura 6.19.

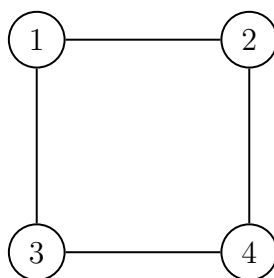
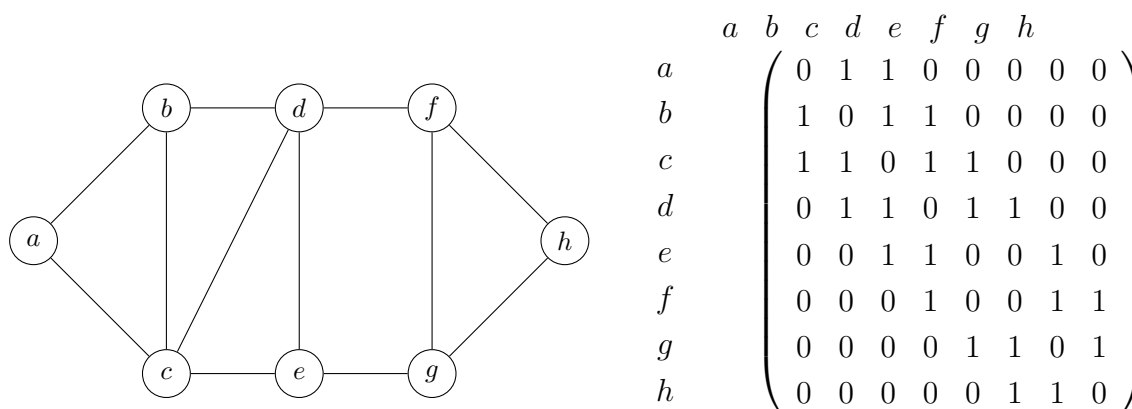
Figura 6.18 No existe camino de longitud 2 entre 1 y 2

Figura 6.19 Resultado de multiplicar matrices de adyacencias por sí mismas

$$A^1: \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \quad A^2: \begin{pmatrix} 2 & 0 & 0 & 2 \\ 0 & 2 & 2 & 0 \\ 0 & 2 & 2 & 0 \\ 2 & 0 & 0 & 2 \end{pmatrix} \quad A^3: \begin{pmatrix} 0 & 4 & 4 & 0 \\ 4 & 0 & 0 & 4 \\ 4 & 0 & 0 & 4 \\ 0 & 4 & 4 & 0 \end{pmatrix}$$

Ejemplo 6.9. Veamos la gráfica de la figura 6.20. Vamos a tratar de determinar si la gráfica es conexa, aplicando la suma y multiplicación de las matrices de adyacencias.

Figura 6.20 Conexidad en gráficas mediante número de caminos

El número de caminos de longitud 2 entre cualesquiera dos vértices está dado por la matriz de adyacencias multiplicada por sí misma, mientras que el número de caminos de longitud 3 entre cualesquiera dos vértices está dado por A^3 :

$$A^2 = \begin{pmatrix} 2 & 1 & 1 & 2 & 1 & 0 & 0 & 0 \\ 1 & 3 & 2 & 1 & 2 & 1 & 0 & 0 \\ 1 & 2 & 4 & 2 & 1 & 1 & 1 & 0 \\ 2 & 1 & 2 & 4 & 1 & 0 & 2 & 1 \\ 1 & 2 & 1 & 1 & 3 & 2 & 0 & 1 \\ 0 & 1 & 1 & 0 & 2 & 3 & 1 & 1 \\ 0 & 0 & 1 & 2 & 0 & 1 & 3 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 2 \end{pmatrix} \quad A^3 = \begin{pmatrix} 2 & 5 & 6 & 3 & 3 & 2 & 1 & 0 \\ 5 & 4 & 7 & 8 & 3 & 1 & 3 & 1 \\ 6 & 7 & 6 & 8 & 7 & 3 & 2 & 2 \\ 3 & 8 & 8 & 4 & 8 & 7 & 2 & 2 \\ 3 & 3 & 7 & 8 & 2 & 2 & 6 & 2 \\ 2 & 1 & 3 & 7 & 2 & 2 & 6 & 4 \\ 1 & 3 & 2 & 2 & 6 & 6 & 2 & 4 \\ 0 & 1 & 2 & 2 & 2 & 4 & 4 & 2 \end{pmatrix}$$

Revisemos algunas parejas de vértices para ver que, en efecto, el número de caminos dado por las potencias de la matriz es correcto. Por ejemplo, el número de caminos de longitud 2 que salen de a y regresan a a está dado por la entrada $m_{aa}^2 = 2$, y corresponden a $a-b-a$ y $a-c-a$. Sólo hay un camino de longitud 2 entre d y e , que corresponde a $d-c-e$. Cualquier otro camino entre d y e es de longitud distinto a 2.

Para los caminos de longitud 3, especificados en la matriz A^3 , podemos observar que hay 3 caminos de longitud 3 entre c y f , a saber, $c-b-d-f$, $c-e-d-f$ y $c-e-g-f$. Cualquier otro camino tiene longitud distinta a 3.

Mostramos a continuación las matrices obtenidas de las distintas potencias. Lo único realmente interesante es que en A^3 todavía no hay camino entre a y h ($m_{ha}^3 = m_{ah}^3 = 0$), aunque en A^4 deberemos poder ya alcanzar todos los vértices desde cualquier otro vértice, pues el camino simple más largo entre dos vértices tiene longitud 4.

$$A^4 = \begin{pmatrix} 11 & 11 & 13 & 16 & 10 & 4 & 5 & 3 \\ 11 & 20 & 20 & 15 & 18 & 12 & 5 & 4 \\ 13 & 20 & 28 & 23 & 16 & 12 & 12 & 5 \\ 16 & 15 & 23 & 31 & 14 & 8 & 17 & 9 \\ 10 & 18 & 16 & 14 & 21 & 16 & 6 & 8 \\ 4 & 12 & 12 & 8 & 16 & 17 & 8 & 8 \\ 5 & 5 & 12 & 17 & 6 & 8 & 16 & 8 \\ 3 & 4 & 5 & 9 & 8 & 8 & 8 & 8 \end{pmatrix} \quad A^5 = \begin{pmatrix} 24 & 40 & 48 & 38 & 34 & 24 & 17 & 9 \\ 40 & 46 & 64 & 70 & 40 & 24 & 34 & 17 \\ 48 & 64 & 72 & 76 & 63 & 40 & 33 & 24 \\ 38 & 70 & 76 & 60 & 71 & 57 & 31 & 25 \\ 34 & 40 & 63 & 71 & 36 & 28 & 45 & 22 \\ 24 & 24 & 40 & 57 & 28 & 24 & 41 & 25 \\ 17 & 34 & 33 & 31 & 45 & 41 & 22 & 24 \\ 9 & 17 & 24 & 25 & 22 & 25 & 24 & 16 \end{pmatrix}$$

A^4 dice que hay 3 caminos de longitud 4 entre a y h . Ellos son: $a-b-d-f-h$, $a-c-e-g-h$ y $a-c-d-f-h$. Como vemos en A^3 , A^2 y A , no hay caminos de longitud menor a 4; como vemos de A^5 hay 9 caminos de longitud 5. Estos 9 caminos se obtienen cuando, para cada uno de los caminos de longitud 4 (que son 3) en lugar de seguir hacia h pasamos al vértice “de enfrente”. Tomemos el camino $a-c-e-g-h$ por ejemplo. De este camino obtenemos 3 de longitud 5, si desde c subimos a b y seguimos hacia h ($a-c-b-d-f-h$), desde e subimos a d ($a-c-e-d-f-h$) o desde g subimos a h ($a-c-e-g-f-h$).

En resumen, si nos interesa saber si hay alguna trayectoria entre dos vértices tenemos que observar si para alguna de las potencias de la matriz hay una entrada distinta de cero en esa posición de la matriz. No obtenemos esta información simplemente de observar A^k , pues pudiese haber caminos de longitudes distintas a k , pero ninguno exactamente de longitud k .

Para obtener la información de si hay o no camino, o de cuántos caminos hay de cualquier longitud (menores a n) hay que sumar las entradas de las potencias de las matrices. Por ejemplo, para saber si dos vértices están conectados por algún camino de longitud menor a 3, basta sumar A , A^2 y A^3 . Podemos denotar con S_3 al resultado de sumar $A + A^2 + A^3$

y con s_{ij}^3 al elemento en el renglón i columna j de S^k . Cada s_{ij}^k , $i = 1, \dots, n$, $j = 1, \dots, n$ se calcula como sigue:

$$s_{ij}^r = m_{ij} + m_{ij}^2 + \dots + m_{ij}^r = \begin{cases} s_{ij}^{r-1} + \sum_{k=1}^n m_{ik}^{r-1} * m_{kj} & \text{si } r > 1 \\ m_{ij} & \text{si } r = 1 \end{cases}$$

Ejemplo 6.10. Apliquemos esta operación a la gráfica de la figura 6.20.

$$A + A^2 + A^3 = \begin{pmatrix} 4 & 6 & 7 & 5 & 4 & 2 & 1 & 0 \\ 6 & 7 & 9 & 9 & 5 & 2 & 3 & 1 \\ 7 & 9 & 10 & 10 & 8 & 4 & 3 & 2 \\ 5 & 9 & 10 & 8 & 9 & 7 & 4 & 3 \\ 4 & 5 & 8 & 9 & 5 & 4 & 6 & 3 \\ 2 & 2 & 4 & 7 & 4 & 5 & 7 & 5 \\ 1 & 3 & 3 & 4 & 6 & 7 & 5 & 5 \\ 0 & 1 & 2 & 3 & 3 & 5 & 5 & 4 \end{pmatrix}$$

En esta matriz podemos observar que no hay camino, de longitud menor o igual que 3, entre a y h . Por ejemplo, entre los vértices d y f no hay caminos de longitud 2, pero sí de longitud 1 y 3. Al observar la matriz de las sumas vemos que en la matriz que representa a las sumas, la posición M_{df} es distinta de cero.

Si únicamente queremos saber si hay algún camino entre dos vértices, podemos modificar ligeramente el algoritmo y hacer una operación lógica de disyunción en lugar de la suma. Llamemos a esta matriz C^r y a cada elemento de la misma c_{ij}^r . La fórmula para el cálculo de esta matriz queda como sigue:

$$c_{ij}^r = m_{ij} \oplus m_{ij}^2 \oplus \dots \oplus m_{ij}^r = \begin{cases} c_{ij}^{r-1} \oplus \bigvee_{k=1}^n m_{ik}^{r-1} \wedge m_{kj} & \text{si } r > 1 \\ m_{ij} & \text{si } r = 1 \end{cases}$$

La interpretación de \oplus corresponde al resultado de hacer una disyunción lógica, interpretando a 0 como falso y a 1 como verdadero. El significado de la fórmula es que va a haber un camino entre el vértice v_i y el vértice v_j , si es que para alguna k existe un camino entre

$$C^5 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \quad C^6 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

De los productos obtenidos del algoritmo de Warshall podemos observar lo siguiente:

- i. En C^3 la entrada para $c_{ah} = c_{ha}$ es 0, lo que quiere decir que no hay ningún camino de longitud menor o igual a 3 entre los vértices a y h , lo que podemos observar directamente en la gráfica.
- ii. En C^4 podemos observar que todas las entradas en la matriz son 1, lo que quiere decir que hay algún camino de longitud menor o igual a 4 entre cualesquiera dos vértices.
- iii. A partir de C^4 el contenido de la matriz ya no cambia, pues ya se encontró un camino entre cualesquiera dos vértices. No mostramos ya C^7 para ahorrar espacio, aunque es claro que va a ser igual a las tres matrices anteriores.

6.5.2. Colofón

El algoritmo de Warshall es lo que se conoce como un algoritmo de *punto fijo*, esto es, a partir de una determinada iteración el resultado de la ejecución ya no cambia. En el caso de este algoritmo, la iteración corresponde, a más tardar, a una menos que el número de vértices, aunque como ya vimos, esto puede suceder antes. Nos vamos a encontrar con muchos algoritmos de punto fijo en ciencias de la computación. Este atributo es importante porque nos garantiza que el algoritmo va a terminar.

Es claro que la representación más conveniente de una gráfica va a depender de los procesos que queramos aplicarle o de las preguntas que deseamos hacer respecto a la gráfica. Si la pregunta se refiere a la conexidad de una gráfica, podemos aplicar cualquiera de los algoritmos de exploración que vimos (BFS o ciclos eulerianos) y en un número relativamente bajo de operaciones (proporcional a la suma del número de vértices con el número

de aristas) determinar si existe un camino desde alguno de los vértices a cualquier otro. Al final del algoritmo simplemente tenemos que observar si todos los vértices quedaron con una distancia definida, y si es así la gráfica es conexa. Si algún vértice quedó con distancia indefinida (∞), entonces no hay camino a él desde el vértice origen. Si usamos el algoritmo BFS podemos averiguar la distancia entre el vértice origen y cualquiera de los otros vértices en la gráfica, pero para saber la distancia entre cualesquiera dos vértices tendríamos que ejecutar el algoritmo tomando como origen a cada uno de los vértices de la gráfica, lo que nos llevaría un número de operaciones proporcional a n^2 , donde n es el número de vértices en la gráfica. El algoritmo de Warshall ejecuta n^2 operaciones para obtener cada matriz de caminos, por lo que ejecuta un número de operaciones proporcional a n^3 para obtener la conexidad de la gráfica. Sin embargo, como estamos hablando de matrices simétricas, se pueden usar representaciones específicas para matrices triangulares y disminuir el número de operaciones a ejecutar.

Encontrar el número de caminos de determinada longitud en una gráfica representada por listas, ya sea de adyacencias o incidencias, se ve bastante más complicado. Habría que seguir todos los posibles caminos para ver si alguno llega al vértice deseado, lo que supondría muchos caminos fallidos. También es difícil responder eficientemente en esta representación a la pregunta de si existe algún camino entre dos vértices cualesquiera. Esto se deberá repetir para cualquier pareja de vértices y no sirve el trabajo hecho para otra pareja cualquiera. En cambio, con la matriz de adyacencias, todos los caminos se calculan simultáneamente.

En contrapartida, ejecutar un algoritmo como BFS o DFS sobre una gráfica representada en una matriz involucra revisar todo el renglón o la columna correspondiente para encontrar el siguiente vértice a explorar, o para determinar si quedan aristas sin usar. Esto agregaría muchas operaciones, por cada operación que se realiza en una lista de adyacencias.

Si representamos una relación de equivalencia con una gráfica no dirigida, el algoritmo de Warshall encuentra lo que se conoce como la *cerradura transitiva* de la relación, esto es, cuáles vértices están relacionados, usando transitividad y simetría.

Para terminar con este tema deseamos remarcar que cuando se va a procesar una gráfica usando una computadora es importante tener claro el tipo de preguntas que se desean hacer, ya que de ello depende la representación interna que se le dé a la gráfica y que va a llevar a procesos más eficientes si esta representación es la adecuada.

Ejercicios

- 6.5.1.- Supongamos que tenemos una gráfica con pesos en las aristas sin ciclos negativos. El algoritmo de Floyd trabaja de manera similar a como lo hace el algoritmo de Warshall; lo presentamos a continuación:

Algoritmo de Floyd para distancias

Objetivo: Encontrar la distancia entre cualesquiera dos vértices para una gráfica con pesos en las aristas.

Datos: La gráfica representada con una matriz de pesos, esto es, en cada entrada de la matriz se encuentra el peso de la arista que conecta a esos vértices. Para el caso de que no haya arista, se registra como distancia ∞ .

Salida: Una matriz D donde en la entrada $d_{ij} = d_{ji}$ se encuentra la distancia entre los vértices v_i y v_j .

Método: Se multiplica la matriz por sí misma; en cada operación se selecciona la trayectoria más corta hasta el momento. El algoritmo se encuentra en el listado 6.4.

Listado 6.4 Algoritmo para distancias de Floyd

```

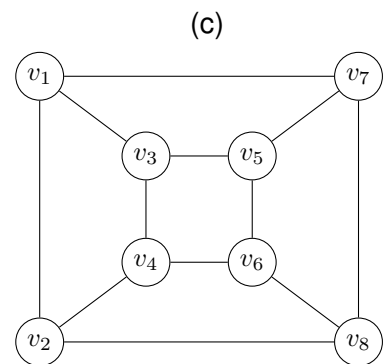
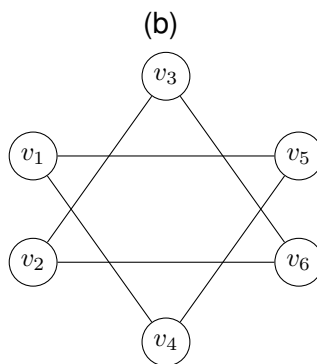
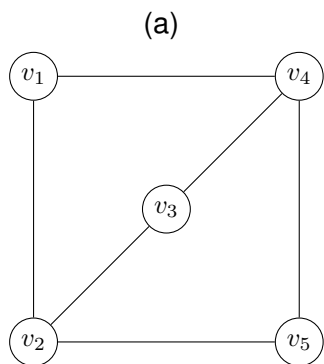
1  /* Inicializacion */
2   $\forall i = 1, \dots, n$ 
3     $\forall j = 1, \dots, n$ 
4       $d_{ij} = \text{matriz}_{ij}$ 
5   $\forall i = 1, \dots, n$ 
6     $\forall j = 1, \dots, n$ 
7       $\forall k = 1, \dots, n$ 
8         $dist = d_{ik} + d_{kj}$ 
9        if  $dist < d_{ij}$ 
10          $d_{ij} = dist$ 

```

Supongamos que la gráfica de la figura 6.20 tenga pesos homogéneos en sus aristas (pensemos que es 1).

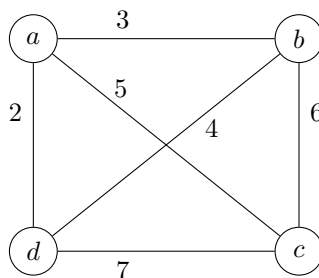
- (a) ¿Cuál es la matriz inicial de distancias?
- (b) ¿Cuál es el resultado de aplicar el algoritmo de Floyd a esta gráfica?

6.5.2.- Calcula el número de caminos de longitud menor o igual a 3 en cada una de las siguientes gráficas:

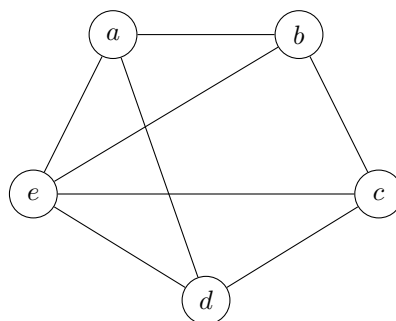


6.5.3.- Para cada una de las gráficas del ejercicio anterior determina las distancias entre todas las parejas de vértices.

6.5.4.- Encuentra las distancias entre cualquier pareja de vértices en la siguiente gráfica:



6.5.5.- Determina cuál es la trayectoria más corta que hace que todos los vértices de la siguiente gráfica queden conectados, usando para ello la versión adecuada del algoritmo de Warshall.



6.5.6.- Prueba que el elemento a_{ii} de la matriz A^2 , donde A es la matriz de adyacencias de una gráfica, contiene el grado del vértice i .

Modelado con gráficas | 7

7.1. Coloración

Supongamos que tenemos que organizar el calendario de exámenes finales para los estudiantes de primer semestre. Tenemos el problema que mientras que los cursos únicamente se llevan una hora al día (dos para Cálculo), los exámenes finales tienen que durar dos horas. Se desea que el examen inicie a la hora que le corresponde al curso. Deseamos también, de preferencia, que a los alumnos de un semestre dado no les toque en un mismo día dos exámenes del mismo semestre. Por lo tanto, debemos evitar que cursos llevados por el mismo estudiante tengan como fecha de examen final períodos o salones que se intersecten.

Podemos modelar este problema con gráficas: cada curso corresponde a un vértice de la gráfica; existe una arista entre dos vértices si y sólo si esas materias se intersectan en horario. El máximo número de materias para las tres carreras del departamento de matemáticas es cinco, por lo que si repartimos las materias en los cinco días no debería haber problemas. La restricción debiera ser que dos materias que están relacionadas entre sí por el horario no les toque el mismo día.

Veamos el primer semestre para la carrera de Ciencias de la Computación. La lista de materias que se llevan en primer semestre, con los horarios, se encuentra en la tabla 7.1. La gráfica que corresponde a estas incompatibilidades se encuentra en la figura 7.1.

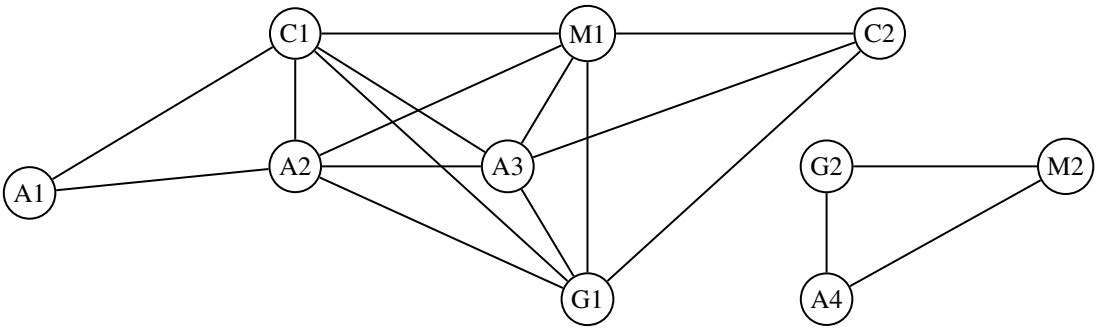
Lo que debemos hacer es asignar días distintos a vértices adyacentes; a cada día lo

podemos pensar como un color distinto y pintar (*colorear*) los vértices de la gráfica de tal manera que no haya dos vértices adyacentes con el mismo color. En general, si contamos con un número infinito de colores podríamos dar a cada vértice su propio color, pero esto no es así: contamos con un número finito de colores, muchas veces acotado – como es este caso en que los exámenes se deben realizar todos en 5 días – por lo que el problema en realidad se puede describir como la coloración de una gráfica con el menor número posible de colores.

Tabla 7.1 Incompatibilidades en exámenes finales

Vértice	Materia	Grupos	Horario
A1	Álgebra Superior I	4000	7-9
A2	Álgebra Superior I	4001	8-10
A3	Álgebra Superior I	4002 a 4007	9-11
A4	Álgebra Superior I	4008 a 4012	12-14
C1	Cálculo Diferencial e Integral I	4018	8-10
C2	Cálculo Diferencial e Integral I	4019 a 4029	10-12
G1	Geometría Analítica I	4037 a 4042	9-11
G2	Geometría Analítica I	4043 a 4048	12-14
M1	Matemáticas Discretas	7000	9-11
M2	Matemáticas Discretas	7001	13-15

Figura 7.1 Gráfica de incompatibilidades para exámenes



Tratemos de asignar colores de izquierda a derecha. A A1 le podemos asignar cualquiera de los colores, digamos rojo. A continuación, a C1 y A2 no les podemos asignar rojo,

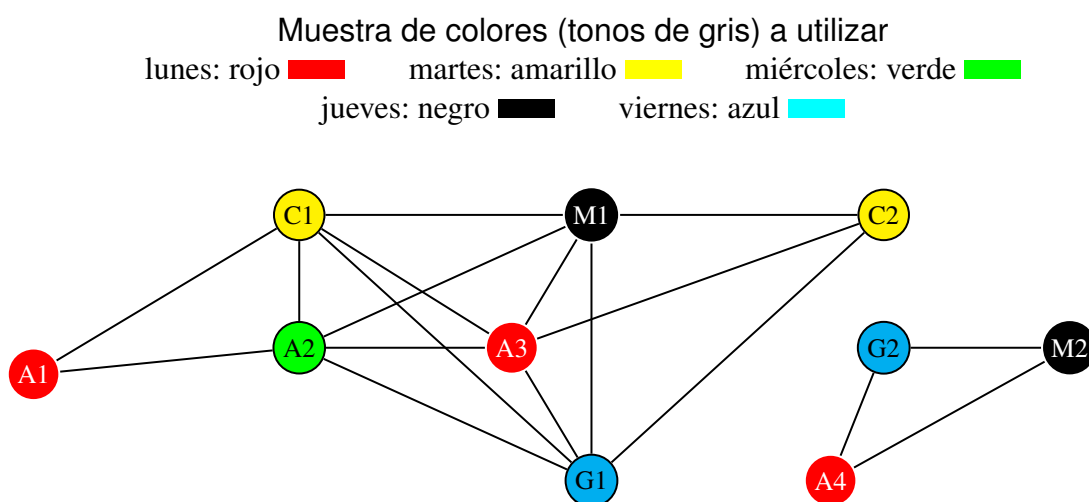
porque son adyacentes a $A1$, ni tampoco el mismo color a ambos porque son adyacentes entre sí; por lo tanto, asignémosles amarillo y verde respectivamente.

En el siguiente nivel, podemos volver a asignar a $M1$ el rojo, pero como tenemos disponibles todavía 2 colores sin usar, asignemos el negro; podemos usar el rojo para $G1$ o $A3$, y como $A1$ es la misma materia que $A3$, se lo asignamos a $A3$. A $G1$ le podemos asignar el quinto color que tenemos todavía disponible, el azul.

A $C2$ no le podemos asignar ninguno de negro, rojo o azul; de igual manera en que lo hicimos con $A1$ y $A3$ le asignamos amarillo, que es el color asignado a $C1$ que corresponde a la misma materia.

Para los tres vértices del componente del lado derecho, bajo el razonamiento que hemos dado hasta ahora, asignamos rojo a $A4$, azul a $G2$ y negro a $M2$. Con esto terminamos de colorear la gráfica y queda como se muestra en la figura 7.2.

Figura 7.2 Coloración de la gráfica de incompatibilidades para exámenes



Prevalece la pregunta de si 5 es el menor número de colores con el que se puede colorear esta gráfica. Si revisamos con cuidado esta gráfica veremos que no se puede colorear con menos de 5 colores, ya que $M1$, por ejemplo, es adyacente a 4 vértices, que son adyacentes dos a dos entre sí, por lo que no se puede repetir ningún color entre ellos ni con $M1$.

Veamos otro ejemplo. Supongamos que estamos tratando de programar presentaciones de libros en la Feria Internacional del Libro. Cada compañía editora tiene un determinado número de libros que presentar, y los autores de los libros deben estar presentes, así como el representante de la editorial. Aunque no es común, podríamos tener más de un libro por autor, por lo que un mismo autor no puede estar en dos presentaciones a la vez.

Para modelar este problema tendremos un vértice por cada libro a presentar, y habrá una arista entre dos vértices si es que son de la misma editorial, del mismo autor o tienen

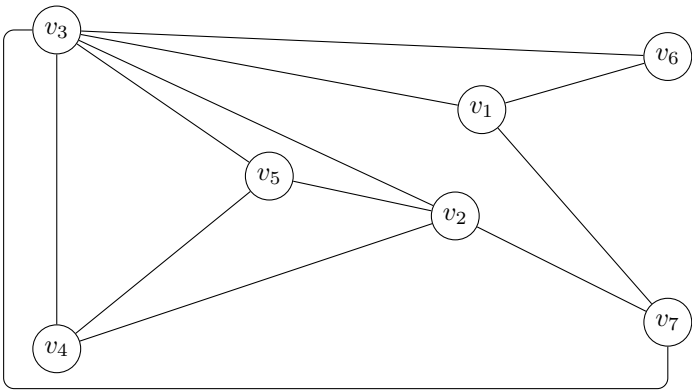
presentadores en común. La lista de libros a presentar se encuentra en la tabla 7.2.

Tabla 7.2 Presentaciones de libros en la Feria del libro

Vértice	Autor	Editorial
v_1	Magidin	Trillas
v_2	Dehesa	Siglo XXI
v_3	Viso	Trillas
v_4	Miranda	Fondo de Cultura
v_5	Galaviz	Fondo de Cultura
v_6	López	Trillas
v_3	Viso	Fondo de Cultura
v_7	Kuri	Siglo XXI
v_2	Dehesa	Fondo de Cultura
v_7	Kuri	Trillas

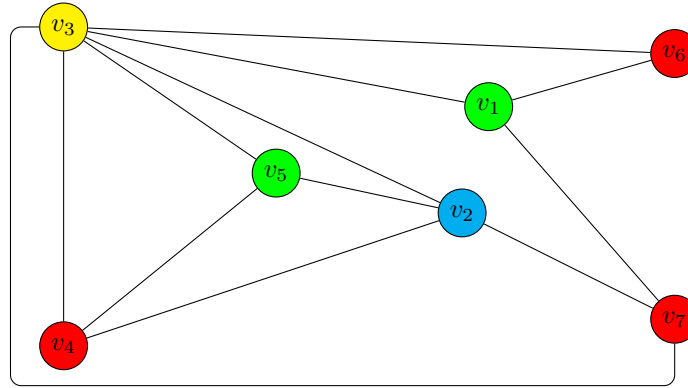
La gráfica que corresponde a este problema se encuentra en la figura 7.3.

Figura 7.3 Relación entre autores y editoriales



Veamos cuál es el menor número de colores que requerimos para colorear esta gráfica. Si asignamos a v_3 un color, tenemos que asignar a v_4 y v_5 colores distintos entre sí y distintos del primero. Para v_2 requerimos también un cuarto color, ya que v_2 es adyacente a los tres vértices anteriores. Por otro lado, v_1 o v_6 pueden ser coloreados con el mismo color que v_2 , pero no el mismo para ambos. Como v_6 no es adyacente a v_5 podría recibir el mismo color, lo mismo que v_7 . Con esto, la gráfica quedaría coloreada como se muestra en la figura 7.4.

En este caso si hubiésemos usado un color por vértice (un horario distinto para cada presentación) hubiésemos requerido de 7 horarios. De esta manera, sólo requerimos 4 horarios distintos.

Figura 7.4 Coloración de la gráfica correspondiente a la Feria del Libro

Como se habrá notado el problema de coloración de gráficas es importante y se presenta suficientemente seguido como para que se trate de encontrar un algoritmo para que asigne los colores de manera eficiente. Tal algoritmo no existe, pero tenemos algunos resultados que nos pueden ayudar en esta asignación. Precisemos primero algunos conceptos:

Definición 7.1 (número cromático) El *número cromático* de una gráfica es el número mínimo de colores que se requieren para colorearla, de forma tal que cualquier par de vértices adyacentes reciban distinto color; el número cromático de una gráfica G se denota con $\chi(G)$.

Supongamos que tenemos un subconjunto de vértices $V_i \subseteq V$ pintados todos del mismo color. Entonces, la subgráfica inducida por este subconjunto ($G[V_i]$) no contiene ninguna arista, pues no puede haber aristas entre vértices pintados del mismo color.

El problema es, entonces, determinar el número cromático de distintas gráficas. A continuación mencionamos algunas propiedades que conocemos acerca del número cromático de una gráfica.

Lema 7.1 $\chi(G) = 1$ si y sólo si G no tiene aristas, esto es $G = (V, \emptyset)$.

Demostración.

\Rightarrow : Por contrapositivo, supongamos que $G = (V, E)$, con $E \neq \emptyset$. Entonces existe $e = uv \in E$ tal que el vértice u es adyacente al vértice v (y viceversa). Si esto es así, u y v tienen que estar coloreados con distinto color, por lo que $\chi(G)$ es al menos 2 ($\neg(\chi(G) = 1) \equiv (\chi(G) \neq 1)$).

\Leftarrow : Supongamos ahora que $G = (V, \emptyset)$. En este caso no hay ningún par de vértices que sean adyacentes, por lo que todos los vértices pueden estar coloreados con el mismo color; esto es, $\chi(G) = 1$. □

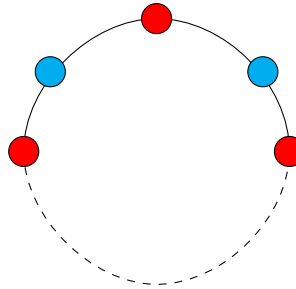
Lema 7.2 ($\chi(C_n)$) *El número cromático de un ciclo con n vértices y n par es 2; si n es impar entonces $\chi(C_n) = 3$:*

$$\chi(C_n) = \begin{cases} 2 & \text{si } n \text{ es par} \\ 3 & \text{si } n \text{ es impar} \end{cases}$$

Demostración.

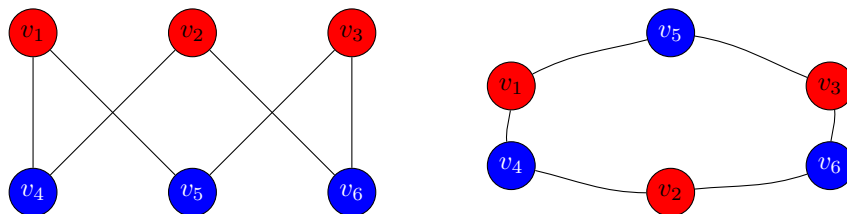
Caso n par: Tomemos un vértice cualquiera del ciclo y lo coloreamos con color 1; el siguiente vértice (en cualquier dirección) lo coloreamos con color 2; el siguiente con 1 y así sucesivamente. A los vértices pares se les asigna 1 y a los vértices impares 2. El último vértice es impar y el primero par, por lo que no se asigna el mismo color a vértices adyacentes – ver figura 7.5.

Figura 7.5 Coloración de círculo con un número par de vértices



Caso n impar: En cambio, si el número de vértices en el ciclo es impar, necesitaremos al menos 3 colores. Si asignamos los colores de la misma forma que lo hicimos en el ciclo de longitud par, al llegar al último vértice por colorear va a resultar que es par, lo mismo que el primero; si le asignamos el color 1 estará coloreado igual que el primero, que es adyacente a él; si le asignamos el color 2 estará coloreado igual que el último que coloreamos, que también es adyacente a él. Por lo tanto, debemos colorearlo con un nuevo color. \square

Ejemplo 7.1. Veamos la siguiente gráfica, que consiste de un ciclo con 6 vértices:



Nótese que la gráfica está pintada como si fuera una gráfica bipartita. De hecho, cualquier gráfica que se puede colorear con exactamente dos colores corresponde a una gráfica bipartita.

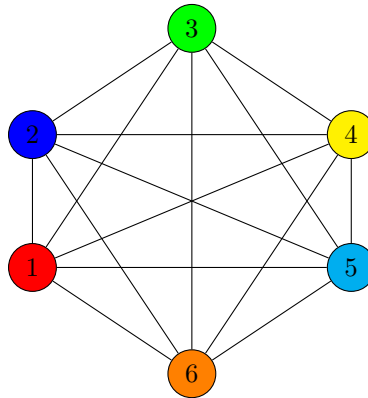
También tenemos resultados referentes a las gráficas completas, K_n :

Lema 7.3 *El número cromático de K_n es n .*

Demostración.

Como todas y cada uno de los vértices de K_n tiene $n - 1$ vecinos, y cada uno de ellos es vecino de todos y cada uno de los otros vértices, no se puede asignar el mismo color a ningún par de vértices vecinos del primero, ya que son vecinos entre sí. Por lo tanto necesitamos que cada vértice tenga su propio color. \square

Ejemplo 7.2. Veamos la coloración de la gráfica K_6 . Como cada vértice es adyacente a todos los otros vértices de la gráfica, ninguna pareja de vértices puede estar pintado del mismo color. De esto, el número cromático es precisamente n , usando un color distinto para cada vértice.



Si tenemos una gráfica que no tiene ciclos, entonces su número cromático será 2, como lo enuncia el lema 7.4.

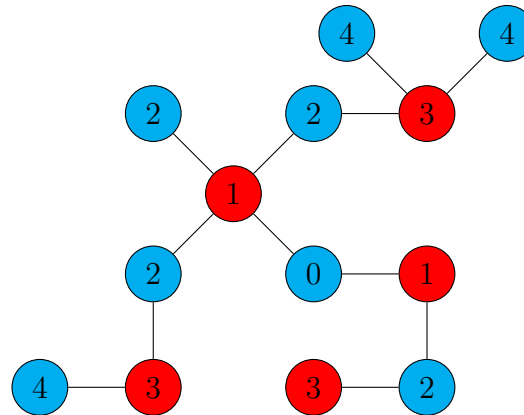
Lema 7.4 *El número cromático de una gráfica sin ciclos es 2.*

Demostración.

Ejecutamos en la gráfica BFS y asignamos los colores de la siguiente manera: a los vértices que están a distancia impar de s les asignamos el color 1, y a los que están a distancia par les asignamos el color 2. Como no hay ciclos en la gráfica, ningún vértice a distancia impar es adyacente a otro vértice a distancia impar, por lo que la coloración es correcta. \square

Veamos un ejemplo en la gráfica de la figura 7.6, donde la distancia al vértice origen se encuentra dentro del vértice.

Figura 7.6 Coloración de gráfica sin ciclos usando BFS



Podemos decir algo respecto al número cromático que tiene que ver con los distintos grados de los vértices de la gráfica. Podemos notar que si un vértice tiene grado k , a lo más requerimos de $k + 1$ colores distintos. Esto es una cota superior para el número cromático, ya que si los vértices vecinos no son adyacentes entre sí bastaría con dos colores.

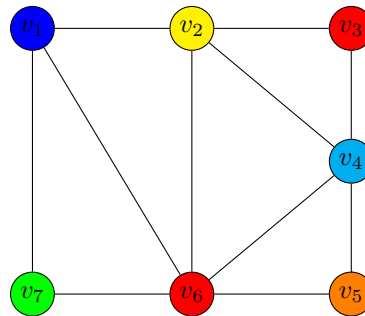
Teorema 7.1 *El número cromático de una gráfica no excede el máximo grado de sus vértices más 1.*

Demostración.

Supongamos que tenemos una gráfica cualquiera cuyo grado máximo es k . Por lo tanto contamos con los colores C_0, C_1, \dots, C_k . Tomemos al primer vértice v y lo coloreamos con cualquiera de los colores.

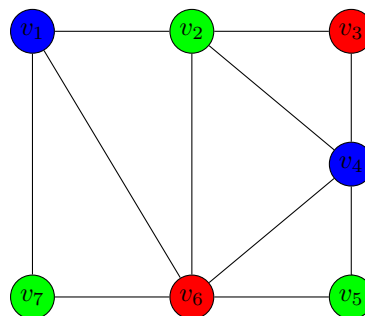
Tomamos cualquier vértice u que no ha sido coloreado; como $\text{grado}(u) \leq k$, hay al menos 1 color que no ha sido usado en ninguno de sus vecinos (pudieran ser más o que los vecinos no estén coloreados todavía); usamos ese color para colorear a u . Continuamos de esa manera hasta que no quede ningún vértice sin colorear. En ningún momento requerimos de más de $k + 1$ colores. \square

Ejemplo 7.3. Veamos la gráfica en la figura 7.7. La manera sencilla de colorear la gráfica es empezando con un color arbitrario para el vértice de mayor grado, que en este caso es v_6 , al que coloreamos de rojo. Después usamos el resto de los colores para colorear a cada uno de los vértices adyacentes a v_6 . En este punto hemos coloreado a todos los vértices menos a v_3 , que no es adyacente a v_6 , por lo que lo podemos colorear con el mismo color que a v_6 , o sea rojo.

Figura 7.7 Coloración siguiendo el grado mayor

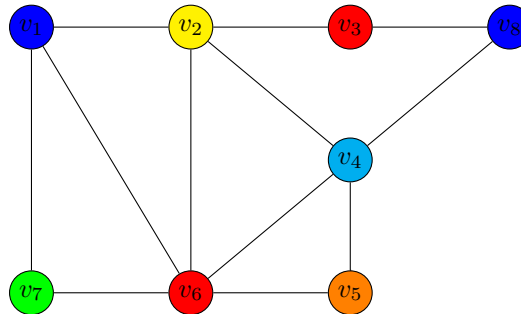
Sin embargo, podemos colorear esta gráfica con menos colores, de la siguiente manera:

- Empezamos con el vértice v_6 , que es el de mayor grado.
- Observamos que los vértices adyacentes a v_6 no son todos adyacentes entre sí, así que aprovechamos esta situación para asignar colores alternados, siguiendo el ciclo $v_6-v_7-v_1-v_2-v_6$. Como es un ciclo de longitud par, bastaría con dos colores para colorearlo. Sin embargo, tenemos la arista v_1-v_6 que cierra dos ciclos impares, por lo que requerimos tres colores distintos para v_6 , v_7 y v_1 , digamos rojo, verde y azul respectivamente. v_2 también forma parte de un ciclo impar, pero no es adyacente a v_7 , por lo que lo podemos colorear también con verde.
- Pasamos a ver el ciclo formado por v_2 , v_3 y v_4 , que también es de longitud impar, por lo que hay que colorear los vértices en el ciclo con tres colores distintos; v_4 no puede ser verde ni rojo, pues es adyacente a vértices con estos colores, por lo que le asignamos azul. v_3 no puede ser verde ni azul, por lo que le asignamos rojo.
- Por último sólo nos falta v_5 , que también conforma un ciclo impar con v_6 y v_4 , por lo que tiene que pintarse de verde.



Si bien la primera coloración es correcta – no hay dos vértices adyacentes con el mismo color – el número cromático de la gráfica es menor, como lo pudimos constatar con la segunda coloración que dimos. Esta segunda coloración corresponde al número cromático de la gráfica.

Ejemplo 7.4. Observemos, sin embargo, una gráfica muy similar a la anterior, con el mismo grado máximo:



En este caso, colorando de manera simple, al terminar de pintar los vértices adyacentes a v_6 nos quedan dos vértices sin colorear, v_3 y v_8 , a los que no podemos colorear con el mismo color, pues son adyacentes entre sí. Conviene en este caso simplemente asignar colores siguiendo la estrategia dada para el vértice de mayor grado, parándose en los vértices adyacentes a los que quedaron sin asignar color, trabajando uno por uno. Una coloración más eficiente se logra siguiendo el criterio de los ciclos impares que dimos en el ejemplo anterior, cuidando a los vértices adyacentes en ciclos distintos.

Otro tipo de gráficas muy fáciles de colorear son las gráficas bipartitas. Al respecto enunciamos un resultado en el teorema 7.2.

Teorema 7.2 Sea $G = (V, E)$ una gráfica. Entonces $\chi(G) = 2$ si y sólo si G es bipartita.

Demostración.

Supongamos que $G = (V, E)$ es bipartita. Por la definición de gráfica bipartita, sabemos que $V = V_1 \cup V_2$ con $V_1 \cap V_2 = \emptyset$. También sabemos que no hay ninguna arista entre cualesquiera dos vértices de V_1 (y lo mismo para cualquier pareja de vértices en V_2). Por lo tanto, si asignamos un color a los vértices de V_1 y otro color a los de V_2 , no tendremos ninguna pareja de vértices adyacentes pintados con el mismo color y sólo requerimos de 2 colores, por lo que $\chi(G) = 2$ (una gráfica con al menos una arista tiene número cromático al menos 2).

Ahora supongamos que $\chi(G) = 2$. Sea V_1 el conjunto de vértices pintados con uno de los colores y V_2 el de los vértices pintados con el otro color. Ningún vértice en V_1 es adyacente a otro en el mismo subconjunto, porque si así fuera no tendríamos una coloración correcta. Lo mismo sucede para cualesquiera dos vértices en V_2 . De esto, tenemos que $V = V_1 \cup V_2$; $V_1 \cap V_2 = \emptyset$, pues ningún vértice puede estar pintado con los dos colores; ningún vértice es adyacente a otro vértice en el mismo subconjunto: por la

definición de número cromático, cada una de las subgráficas inducidas $G[V_i]$ ($i = 1, 2$) no contiene aristas. Por lo tanto, todas las aristas en G tienen un vértice en V_1 y otro vértice en V_2 . De donde G es bipartita. \square

Teorema 7.3 Sea $G = (V, E)$ una gráfica conexa tal que $|V| \geq 3$. Entonces G es bipartita si y sólo si G no contiene ciclos de longitud impar.

Demostración.

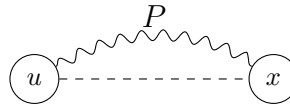
Sea $G = (V_1, V_2, E)$ una gráfica bipartita. Si G contiene un ciclo C , los vértices de C alternan entre los conjuntos V_1 y V_2 , por lo que el número de vértices es par y por lo tanto también el número de aristas; de donde C es de longitud par.

En sentido inverso, supongamos que G no contiene ciclos de longitud impar. Demostraremos que podemos partir a V en dos conjuntos ajenos, tales que toda arista vaya de un vértice en uno de los conjuntos a un vértice en el otro.

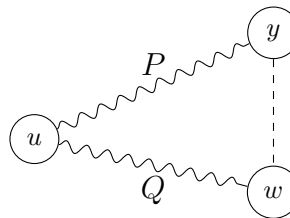
Elegimos un vértice arbitrario u en V y ejecutamos el algoritmo BFS con u como origen. A los vértices que están a distancia impar los asignamos a un conjunto V_1 , mientras que a los vértices que están a distancia par (incluyendo a u que está a distancia 0) los asignamos a un conjunto V_2 . Es claro que $V_1 \cap V_2 = \emptyset$ y que, si G es conexa, $V = V_1 \cup V_2$, pues todos los vértices serán alcanzados.

Debemos demostrar ahora que no hay aristas entre vértices del mismo conjunto. Consideremos los siguientes cuatro casos:

Caso 1. Sea $x \in V_2$ tal que $x \neq u$. Como x está a distancia par de u , existe una trayectoria P de longitud par de u a x . Sea $2n$ la longitud de P . Si la arista $ux \in E$, tenemos un ciclo de longitud impar – la longitud de $u \xrightarrow{P} x \xrightarrow{ux} u$ es $2n + 1$ – lo que contradice la hipótesis de que G no contiene ningún ciclo de longitud impar. Por lo que ux no puede estar en E .



Caso 2. Sean $y, w \in V_1$, con $y \neq u$ y $w \neq u$. Supongamos que $yw \in E$. Sea P la trayectoria entre u e y , y Q la trayectoria entre u y w , y supongamos que P y Q no tienen ningún vértice en común:



Como BFS encuentra las distancias entre el vértice u y cualquier otro vértice en G , sabemos que P y Q son trayectorias lo más cortas posibles, con P de longitud $2m$ y Q de longitud $2n$. Si $yw \in E$, el ciclo

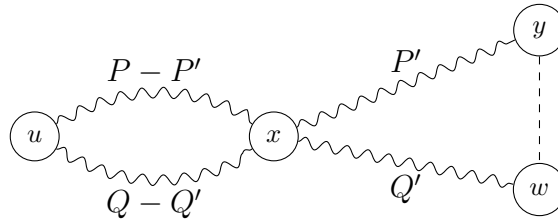
$$u \overset{P}{\rightsquigarrow} y \xrightarrow{yw} w \overset{Q}{\rightsquigarrow} u$$

tiene longitud impar ($2m + 2n + 1 = 2(m + n) + 1$), lo que contradice la hipótesis de que G no contiene ciclos de longitud impar, por lo que $yw \notin E$.

Supongamos ahora que las trayectorias se intersectan en un vértice x , con

$$P' = x \rightsquigarrow y \quad \text{y} \quad Q' = x \rightsquigarrow w.$$

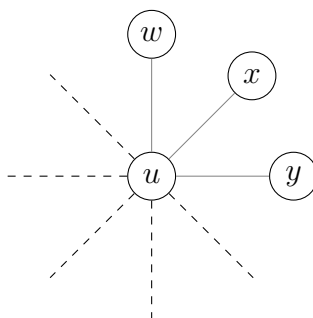
Además, P' y Q' no tienen ningún otro vértice en común además de x :



Como tanto P como Q son trayectorias más cortas, $P - P'$ y $Q - Q'$ tienen la misma longitud: si por ejemplo $P - P'$ tuviera menor longitud que $Q - Q'$, entonces la trayectoria $u \overset{P-P'}{\rightsquigarrow} x \overset{Q'}{\rightsquigarrow} w$ sería una trayectoria más corta que Q , contradiciendo el hecho de que BFS encuentra trayectorias más cortas. Por lo tanto, si suponemos que la longitud de $P - P'$ ($Q - Q'$) es r , la longitud de P' es $2m - r$ y la longitud de Q' es $2n - r$. Por lo tanto, el ciclo que se forma, si es que $yw \in E$, es de longitud $2m - r + 2n - r + 1 = 2m + 2n - 2r + 1 = 2(m + n - r) + 1$ que es impar, otra vez contradiciendo la hipótesis de que no existe ningún ciclo de longitud impar en G , por lo que $yw \notin E$.

Caso 3. Tanto y como w están a distancia impar, con $y, w \in V_1$, $y \neq u$ y $w \neq u$. Nuevamente sean P y Q trayectorias de longitud más corta entre u e y y entre u y w respectivamente (tenemos un diagrama como el del caso 2). En este caso, las longitudes de P y Q son ambas impares, por lo que podemos expresarlas como $2m - 1$ y $2n - 1$ respectivamente. Si $yw \in E$, el ciclo formado por $u \overset{P}{\rightsquigarrow} y \xrightarrow{yw} w \overset{Q}{\rightsquigarrow} u$ tiene longitud $2m - 1 + 2n - 1 + 1 = 2m + 2n - 2 + 1 = 2(m + n - 1) + 1$ que es un número impar, nuevamente contradiciendo la hipótesis de que G no contiene ciclos de longitud impar, por lo que $yw \notin E$. El caso de que P y Q se intersecten en algún vértice distinto de u sigue el mismo razonamiento que en el caso 2, por lo que ya no lo presentamos.

Caso 4. No hay ningún vértice distinto de u en V_1 . Si este es el caso, todos los vértices en V_2 son adyacentes a u y están a distancia 1 – como no hay ningún vértice a distancia par de u , en particular a distancia 2, ningún camino que empiece en u puede tener longitud mayor que 1 –.



Si hubiese alguna arista entre dos vértices de V_2 , por ejemplo $xw \in E$, tendríamos el ciclo $u-x-w-u$, que sería un ciclo impar, contradiciendo la hipótesis de que G no tiene ciclos de longitud impar, por lo que $xw \notin E$. \square

Si bien una de las hipótesis de estos dos teoremas es que G es conexa, se puede aplicar el mismo razonamiento para cada componente conexa de una grafica, lo que resulta en el siguiente corolario:

Corolario 7.4 *Sea G una gráfica. Entonces $\chi(G) = 2$ si y sólo si G no contiene ciclos de longitud impar.*

Se deja la demostración como ejercicio.

Con lo anterior hemos dado algunas pistas de cómo proceder a colorear una gráfica, identificando qué tipo de gráfica es y asignando el menor número de colores posibles. Hay que aclarar, sin embargo, que fuera de las gráficas que están plenamente identificadas, no hay receta (algoritmo) para colorear una gráfica.

Para terminar esta sección enunciaremos un teorema muy famoso, conocido como el *Teorema de los cuatro colores*, que hasta 1976 seguía como una conjetura. Para enunciarlo requerimos del concepto de una *gráfica plana*:

Definición 7.2 Una *gráfica plana* es aquella que se puede pintar en el plano sin que se crucen aristas.

Tomemos un mapa donde todos los países están totalmente contenidos en una sola región; cada país está representado por un vértice y hay una arista entre un país y otro si y sólo si los países tienen una frontera en común que corresponda a algún segmento de recta (si la frontera es únicamente en un punto, no se consideran adyacentes). Este tipo de gráficas son planas pues nunca hay fronteras en común que “brinquen” por encima de otras fronteras. Las aristas siempre aparecen como radios que salen de un país a los países con frontera común y se pueden ir dibujando de izquierda a derecha y de norte a sur. La especificación original del Teorema de los cuatro colores, planteado como conjetura a mitad

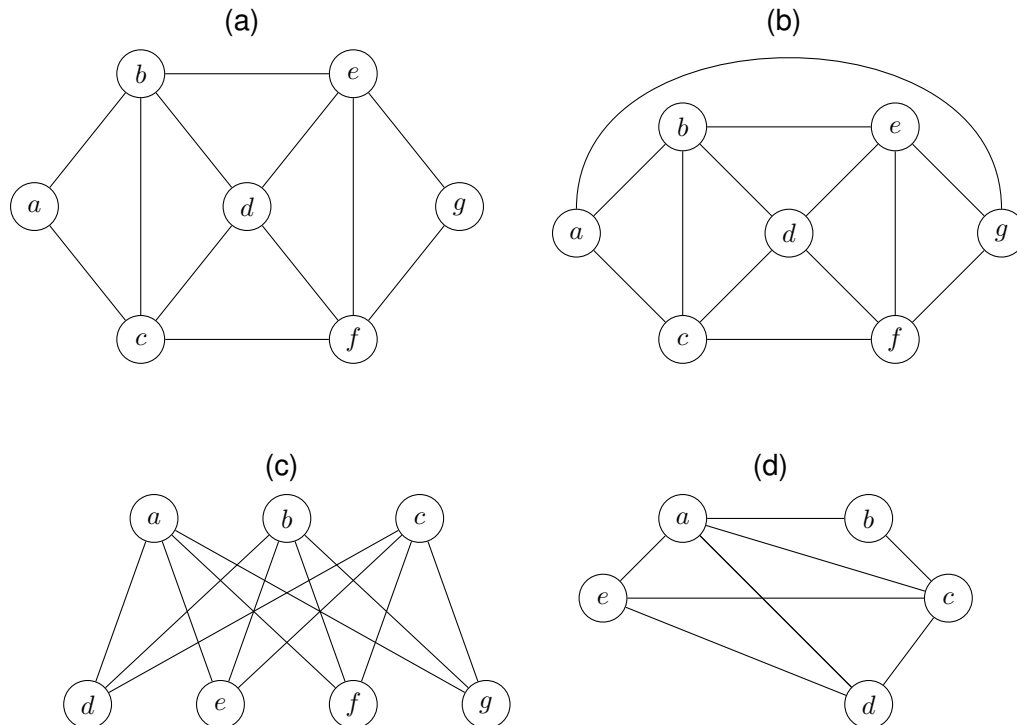
del siglo XIX; fue demostrado por Appel y Haken en 1976, haciendo un estudio caso por caso, usando para ello una computadora¹. Pasamos a enunciar el teorema:

Teorema 7.5 (Teorema de los cuatro colores) *El número cromático de una gráfica plana no excede a 4.*

Ejercicios

7.1.1.- Demuestra el corolario 7.4.

7.1.2.- Colorea las siguientes gráficas de acuerdo a su número cromático. Justifica el número cromático determinado.



7.1.3.- Programa los exámenes finales para Cálculo I, Cálculo II, Cálculo III, Cálculo IV, Discretas, ICC1, ICC2 y Probabilidad y Estadística, usando el mínimo número de horarios y considerando que no hay estudiantes cursando al mismo tiempo²:

¹Por el hecho de que el teorema fue demostrado con la ayuda de una computadora, para listar los casos posibles, muchos matemáticos sostienen que ésta no es una demostración matemática elegante. Inclusive, hay matemáticos que sostienen que ni siquiera es una demostración.

²A este tipo de consideraciones se les llama *incompatibilidades*.

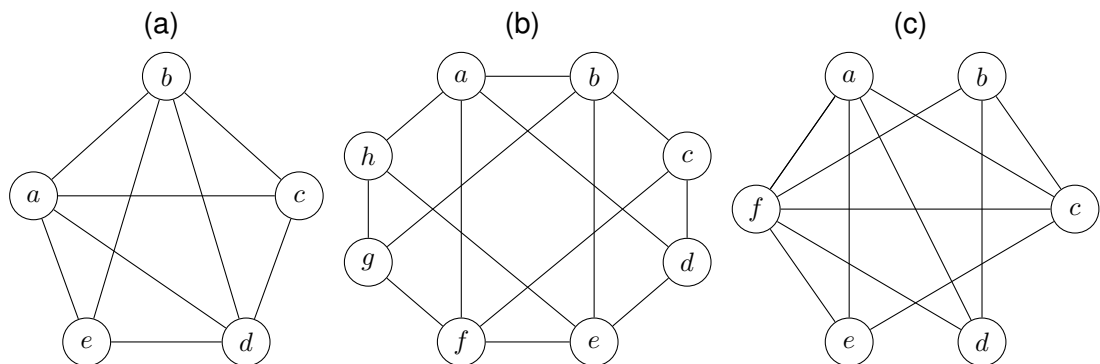
- Cálculo I y Probabilidad y Estadística;
- Cálculo II y Probabilidad y Estadística;
- Cálculo IV y Discretas;
- Cálculo IV e ICC1;
- Cálculo I y Cálculo II;
- Cálculo III y Cálculo IV;

pero hay estudiantes comunes en cualquier otra combinación de cursos.

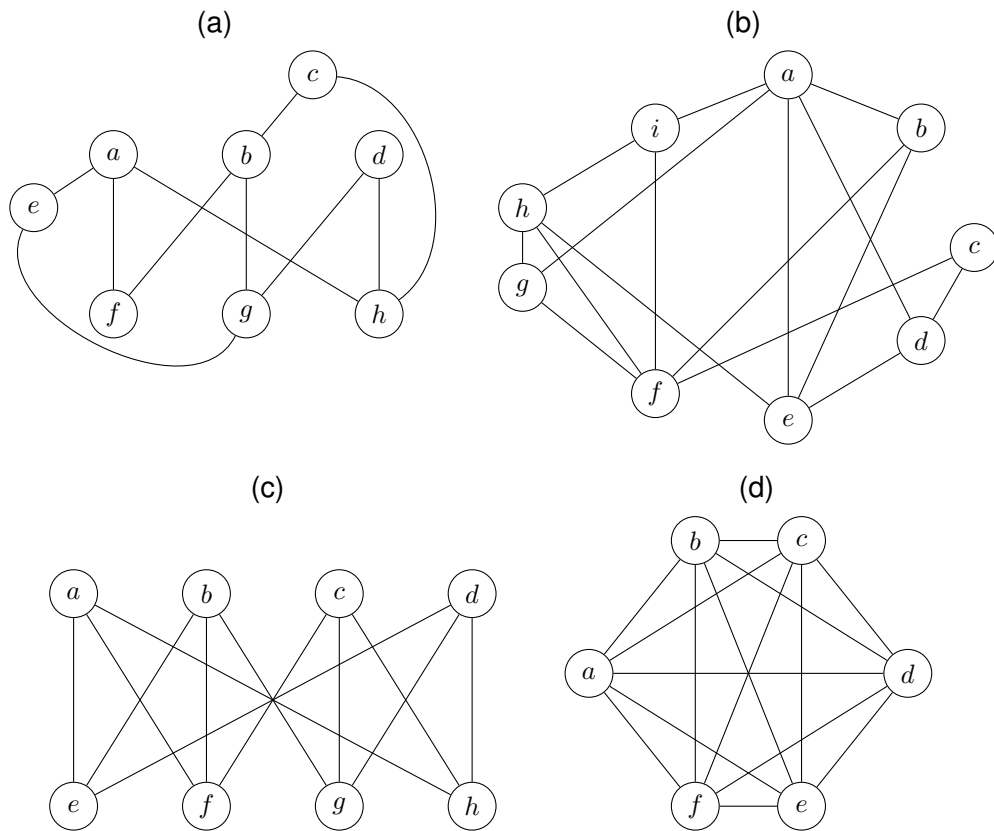
7.1.4.- ¿Cuál es el menor número de policías que debemos apostar en las escuelas de una colonia para cubrir a todas las escuelas de la zona, si no podemos poner policías que estén a menos de 15 cuadras de distancia, porque entonces se juntan y se ponen a platicar, descuidando la vigilancia?

	E1	E2	E3	E4	E5	E6
E1	–	8	17	20	5	10
E2	8	–	12	17	10	16
E3	17	12	–	10	20	25
E4	20	17	10	–	21	22
E5	5	10	20	21	–	10
E6	10	16	25	22	10	–

7.1.5.- Podemos observar que $\mathcal{K}_{3,3}$ y \mathcal{K}_5 no son gráficas planas, pues no hay manera de dibujarlas sin cruzar aristas. El Teorema de Kuratowski nos dice que si una gráfica contiene alguna subgráfica isomorfa a $\mathcal{K}_{3,3}$ o a \mathcal{K}_5 entonces la gráfica no es plana – el Teorema de Kuratowski es más fuerte e incluye el concepto de *homeomorfismo* que no es parte de este texto, por lo que sólo veremos esta versión de este teorema –. En las siguientes gráficas, usando este teorema, determina si las gráficas son planas o no. Si lo son, dibújalas como gráficas planas.



7.1.6.- Usando todos los resultados presentados en esta sección, determina el número cromático de las siguientes gráficas y colorea las gráficas de acuerdo al número cromático determinado.

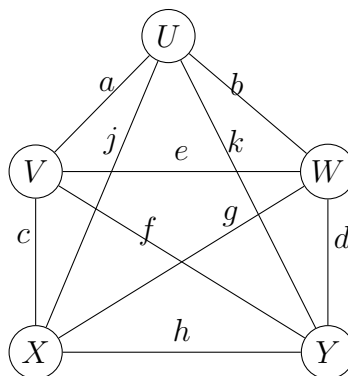


Árboles | 8

8.1. Caracterización

Los árboles son gráficas que modelan adecuadamente diversos problemas. En particular son muy útiles cuando queremos cubrir a un conjunto de nodos de la manera más eficientemente posible, con el menor número posible de conexiones. Por ejemplo, si queremos poner un conjunto de teléfonos que comuniquen a un cierto número de comunidades pero queremos tener la menor cantidad de líneas posibles; y estamos dispuestos a canalizar llamadas aunque no sea directamente, la manera más económica de hacerlo es mediante un árbol. Supongamos que tenemos cinco pueblos y queremos que todos estén comunicados con cualquier otro. La gráfica que deseamos tener es una gráfica completa (K_5), pero sin que tengamos que tener a todas las aristas presentes.

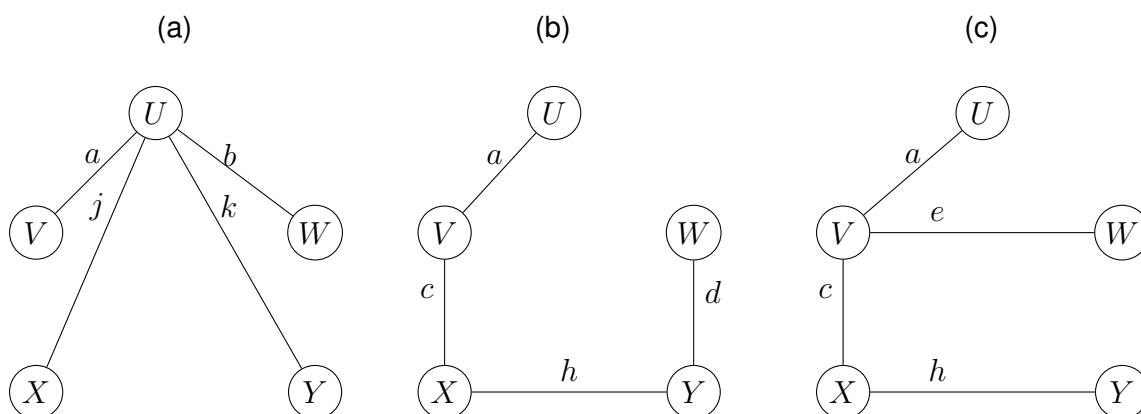
Figura 8.1 Gráfica completa K_5



Podemos encontrar varios conjuntos de aristas tales que cubramos con ellas a todas las ciudades, como se muestra en las gráficas de la figura 8.2. Lo que podemos notar en estas tres gráficas es que todas son conexas y no tienen ciclos.

Definición 8.1 (árbol) Un *árbol* es una gráfica conexa y acíclica.

Figura 8.2 Maneras de cubrir a todos los vértices de K_5



Las gráficas de la figura 8.3 son todas ellas árboles, mientras que las de la figura 8.4 no lo son.

Figura 8.3 Gráficas que son árboles

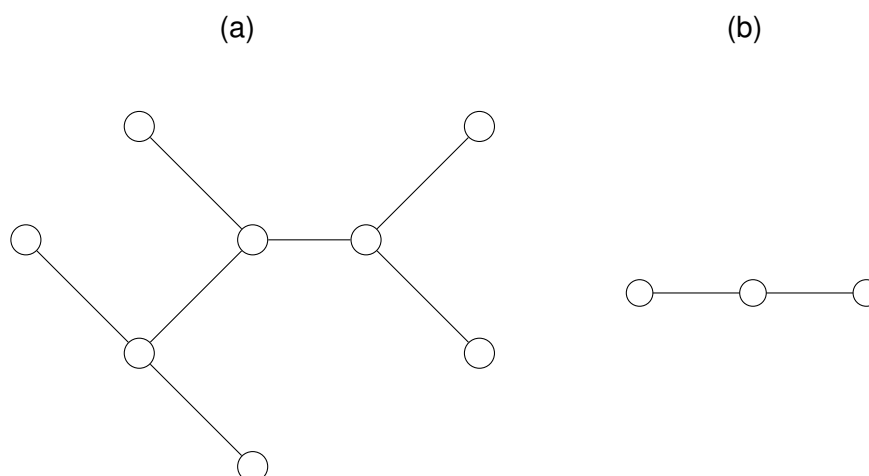
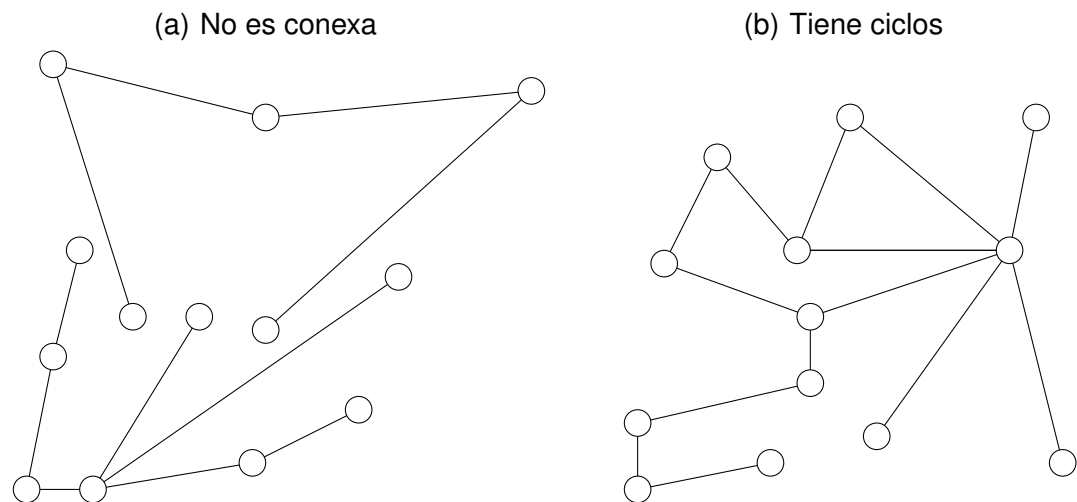


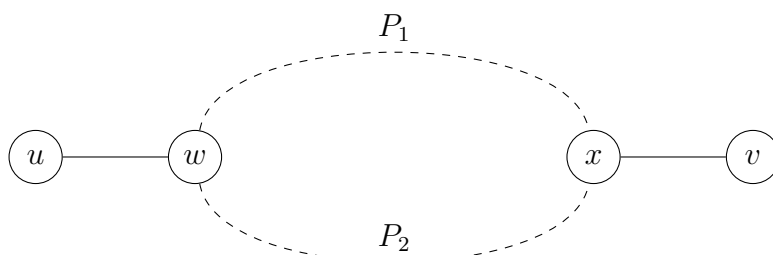
Figura 8.4 Gráficas que no son árboles

Teorema 8.1 Sean u y v vértices en un árbol T con al menos dos vértices. Entonces hay exactamente una trayectoria de u a v .

Demostración.

Como un árbol es una gráfica conexa, existe al menos un camino entre u y v en T ; por el teorema 6.1 éste contiene a una trayectoria entre u y v . Ahora mostraremos que no puede haber más de una.

Supongamos que hay más de una trayectoria entre u y v – al menos dos –; si el árbol tiene sólo dos vértices, la única trayectoria entre ellos es la arista que los une, y como no permitimos aristas múltiples, ésa es la única trayectoria entre los dos vértices. Supongamos entonces $|V| \geq 3$ y hay más de una trayectoria entre dos de esos vértices, u y v ; tenemos la situación que se muestra en la figura 8.5.

Figura 8.5 Existencia de más de una trayectoria entre dos vértices en un árbol

Supongamos que las dos trayectorias comparten una porción inicial y final, $u \rightsquigarrow w$ y $x \rightsquigarrow v$ (w pudiera ser u y x pudiera ser v). Sean esas dos trayectorias P_1 y P_2 . Pero entonces, la trayectoria $w \rightsquigarrow P_1 \rightsquigarrow x \rightsquigarrow P_2 \rightsquigarrow w$ forma un ciclo, lo que contradice que T es un árbol (conexo y acíclico). De donde no puede haber dos trayectorias entre dos vértices. □

Teorema 8.2 *En un árbol T con más de un vértice hay al menos dos vértices de grado 1.*

Demostración.

Como T es conexa con más de un vértice, hay al menos una trayectoria con dos vértices distintos. Tomemos una pareja de vértices u y v cuyo camino entre ellos sea tan grande como cualquier otra trayectoria entre dos vértices en T (de tamaño máximo). Esta trayectoria tiene un número máximo de aristas respecto a cualquier trayectoria en T . Aseveramos que tanto u como v tienen grado uno. Si u tiene grado mayor que uno, como T no tiene ciclos, existiría una trayectoria más larga en T ; lo mismo para v . De donde u y v tienen grado uno. □

Teorema 8.3 *Un árbol con n vértices tiene exactamente $n - 1$ aristas.*

Demostración.

Haremos la demostración por inducción en n , el número de vértices en T .

Para $n = 1$, el único árbol con un solo vértice no puede tener ninguna arista, pues como no hay ciclos no podemos tener lazos. Por lo que se cumple que el árbol con un vértice tiene cero aristas.

Nuestra hipótesis de inducción es que todo árbol con $k < n$ vértices tiene $k - 1$ aristas.

Supongamos ahora un árbol con n vértices. Como tenemos al menos dos de ellos con grado 1, quitemos uno de ellos y su correspondiente arista. Esta gráfica también es un árbol (sigue siendo conexa y acíclica). Nos queda una gráfica con $n - 1$ vértices que, por la hipótesis de inducción, tiene $n - 2$ aristas. Por lo que al agregarle el vértice que quitamos con su arista correspondiente, tendremos n vértices con $n - 1$ aristas. □

Teorema 8.4

- a. *Cuando se elimina una arista de un árbol, la gráfica se desconecta y deja de ser árbol.*
- b. *Cuando se agrega una arista a un árbol (sin agregar vértices) la gráfica resultante tiene exactamente un ciclo y por lo tanto deja de ser árbol.*

Demostración.

- a. Sea T un árbol con n vértices. Por el teorema 8.3, T tiene $n - 1$ aristas y existe exactamente una trayectoria entre cualesquiera dos vértices. Supongamos que quitamos la arista $u-v$. Esta arista era la única trayectoria entre u y v , por lo que al quitar la arista ya no hay trayectoria entre u y v y la gráfica queda desconectada.

- b. Nuevamente, como T es un árbol, existe exactamente una trayectoria entre cualesquiera dos vértices. Si la gráfica únicamente tiene dos vértices y una arista, la única arista que podemos agregar es un lazo, lo que hace un ciclo, o una arista múltiple, lo que también formaría un ciclo. De esto, T dejaría de ser acíclica y por lo tanto árbol.

Supongamos que $|V| > 2$ y tomemos dos vértices u y v que no son adyacentes en T . Como T es árbol, existe una trayectoria entre u y v . Al agregarle la arista $u-v$ se forma exactamente un ciclo, por lo que T deja de ser árbol.

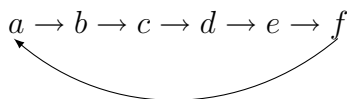


Teorema 8.5 *Los siguientes enunciados respecto a una gráfica T son equivalentes:*

- (a) T es un árbol.
- (b) T es conexa y el número de vértices es uno más que el número de aristas.
- (c) T es acíclica y el número de vértices es uno más que el número de aristas.
- (d) Existe una única trayectoria entre cualesquiera dos vértices de T .
- (e) T es conexa y al quitar cualquier arista T se desconecta.
- (f) T es acíclica y al agregar cualquier arista se forma un ciclo.

Demostración.

Para demostrar estas equivalencias, tenemos que seguir el siguiente orden en las demostraciones:



$a \rightarrow b$ Sea T un árbol (gráfica acíclica y conexa). Por definición, es conexa, por lo que la primera parte del inciso (b) ya está. También demostramos en el teorema 8.3 que un árbol con n vértices tiene $n - 1$ aristas, que corresponde a la segunda parte del inciso (b).

$b \rightarrow c$ Que el número de vértices es uno más que el número de aristas está en el antecedente, por lo que está en el consecuente. Nos falta demostrar que es acíclica.

Como es conexa, existe una trayectoria entre cualesquiera dos vértices. Por contradicción, supongamos que existe un ciclo con k vértices en la gráfica. Como es un ciclo, hay el mismo número de vértices que de aristas en él. Fuera del ciclo se encuentran $n - k$ vértices y, como la gráfica es conexa, debemos poderlos alcanzar desde cualquier vértice del ciclo. Pero necesitamos al menos $n - k$ aristas para alcanzar a esos vértices, lo que nos da un total de al menos n aristas; y por el antecedente sabemos que la gráfica tiene exactamente $n - 1$ aristas; de donde la suposición de que existe al menos un ciclo no se cumple, por lo que la gráfica es acíclica.

$c \rightarrow d$ Sabemos que T es acíclica y con n vértices y $n - 1$ aristas. Debemos demostrar que es conexa. Como no hay ciclos no hay más de una trayectoria entre cualesquiera dos vértices. Ahora tenemos que demostrar que al menos hay una.

Haremos la demostración por inducción en el número de vértices. Si el número de vértices es uno, T es conexa y acíclica y tiene cero aristas. Por vacuidad se cumple que hay una trayectoria entre cualesquiera dos vértices distintos.

Supongamos que se cumple para gráficas con $2 \leq k < n$ vértices y veamos para n vértices.

Tomemos la gráfica con n vértices. Como es acíclica, tiene al menos dos vértices de grado 1. Quitemos a uno de ellos y a la arista que lo conecta con la gráfica. En la subgráfica que nos queda tenemos un vértice menos, una arista menos y no hay ciclos. Por la hipótesis de inducción, en esta gráfica existe una única trayectoria entre cualesquiera dos vértices. Ahora le agregamos el vértice que le quitamos, junto con su arista. La única trayectoria a este vértice es pasando por la arista que lo une al resto de la gráfica, por lo que se cumple para toda la gráfica que exista una única trayectoria entre cualesquiera dos vértices.

$d \rightarrow e$ Sabemos que existe exactamente una trayectoria entre cualesquiera dos vértices, que es la definición de que T es conexa. Como esa trayectoria es única, al quitar cualquier arista deja de haber trayectoria entre los extremos de esa arista.

$e \rightarrow f$ Como T es conexa, quiere decir que hay una trayectoria entre cualesquiera dos vértices. Si hubiese algún ciclo, podríamos quitar una arista de ese ciclo sin que la gráfica se desconectara, pero no es así, ya que en el antecedente decimos que si se quita una arista se desconecta. De donde es acíclica. Si agregamos una arista entre cualesquiera dos vértices, como ya había una trayectoria entre ellos, se forma una nueva trayectoria, y por lo tanto un ciclo.

$f \rightarrow a$ Debemos demostrar que T es acíclica y conexa, suponiendo que es acíclica y que al agregarle cualquier arista se forma un ciclo. Que es acíclica ya está.

Si al agregar una arista $x-y$, para x e y cualesquiera, se forma un ciclo, quiere decir que ya había una trayectoria entre x e y . Y la definición de que es conexa es que para cualesquiera dos vértices haya siempre una trayectoria entre ellos. \square

Con este teorema dejamos ya varias caracterizaciones de árboles con las que podemos jugar para encontrar otros resultados.

Ejercicios

8.1.1.- Demuestra directamente que $d \rightarrow c$.

8.1.2.- ¿Cuántas aristas hay en un árbol con 15 vértices?

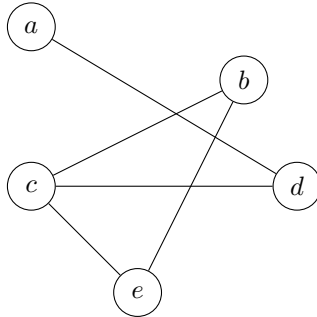
8.1.3.- ¿Cuántos vértices hay en un árbol con 20 aristas?

8.1.4.- Determina si las siguientes gráficas son o no árboles. Justifica tu respuesta.

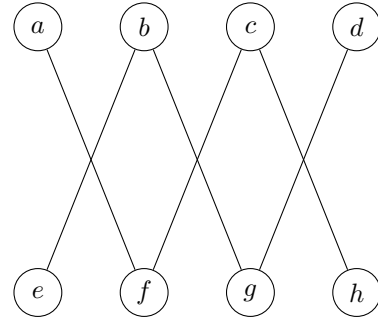
(a)



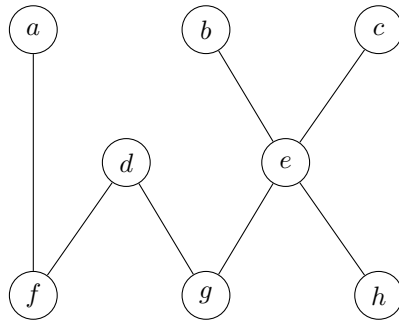
(b)



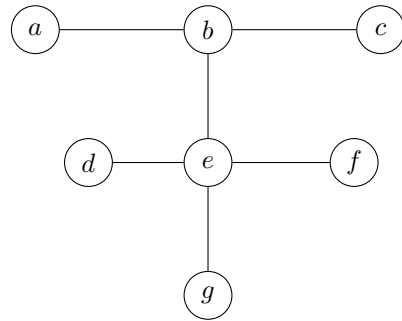
(c)



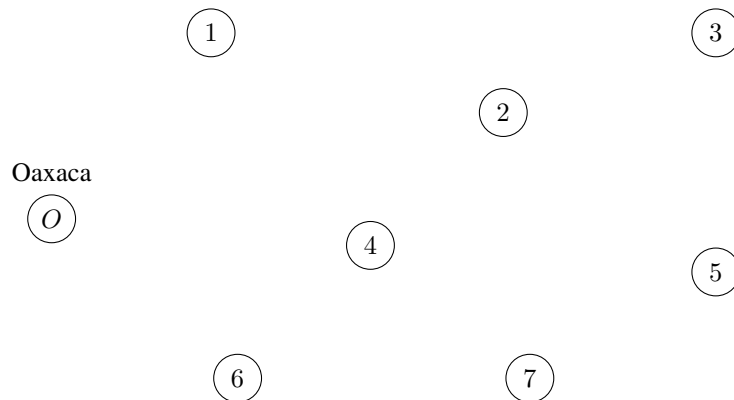
(d)



(e)



8.1.5.- La Secretaría de comunicaciones quiere construir una red ferroviaria que una a varias comunidades madereras en el estado de Oaxaca para que puedan transportar la madera a la Ciudad de Oaxaca. Los puntos de acopio van a estar situados equidistantes y se pueden ver en el siguiente mapa. Diseña las líneas del ferrocarril para que se construya lo más económico posible. ¿Hay una única solución?



- 8.1.6.- Dibuja una gráfica que **no sea un árbol** para la cual el número de vértices es uno más que el número de aristas.
- 8.1.7.- Dibuja una gráfica que **no sea un árbol** que tenga exactamente dos vértices de grado 1.
- 8.1.8.- ¿Cuál es el máximo número de vértices en una gráfica conexa con n aristas?
- 8.1.9.- ¿Cuál es el mínimo número de vértices en una gráfica conexa con n aristas?

8.2. Árboles generadores

Recordemos la definición 5.14 de la página 223, que repetimos acá:

Definición 8.2 (subgráfica) Una gráfica $G' = (V', E')$ es *subgráfica* de una gráfica $G = (V, E)$ si es que $V' \subseteq V$ y $E' \subseteq E$.

Es claro que toda gráfica es subgráfica de sí misma.

Muchas veces queremos conectar un conjunto de puntos entre sí de la manera más económica posible. Es decir, queremos encontrar una subgráfica de la gráfica original cuyas aristas toquen todos los vértices y que no tenga ciclos. Los ciclos, de cierta manera, son un gasto redundante porque tenemos más de una manera de alcanzar a los vértices en el ciclo.

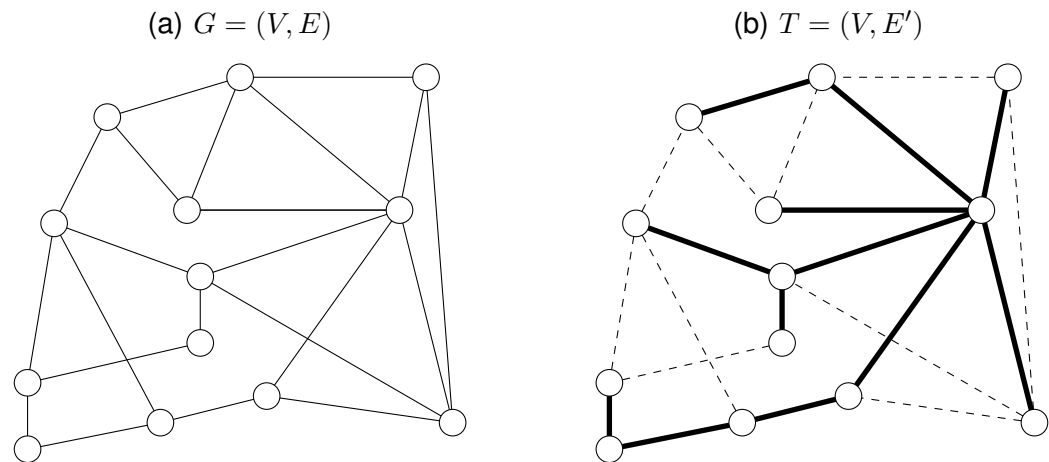
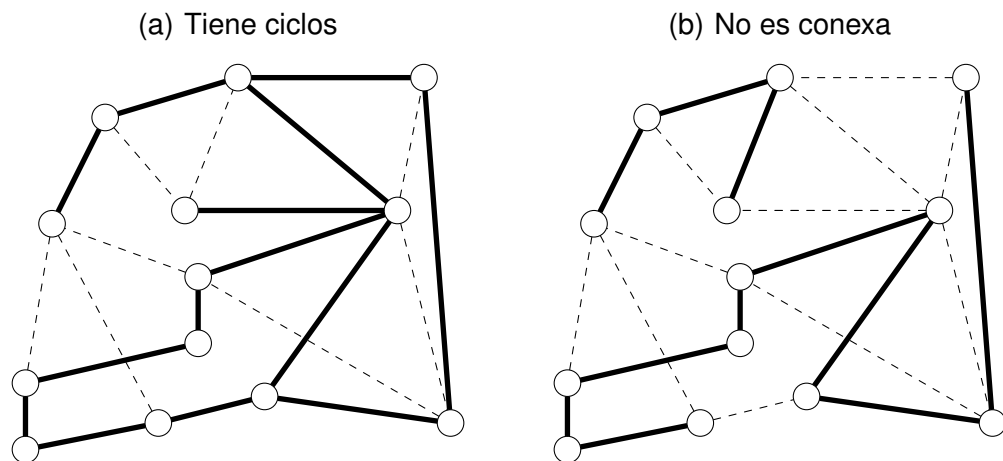
Definición 8.3 (árbol generador) Un *árbol generador* de una gráfica $G = (V, E)$ es una subgráfica conexa y acíclica de G , $T = (V', E')$, tal que $V' = V$, $E' \subseteq E$.

Veamos las gráficas en las figuras 8.6 y 8.7.

En la gráfica de la figura 8.6, la subgráfica 8.6(b) corresponde a un árbol generador de la gráfica en 8.6(a), ya que es conexa, acíclica y cubre todos los vértices de 8.6(a). Las dos gráficas de la figura 8.7, en cambio, no corresponden a árboles generadores de la gráfica 8.6(a) en la figura 8.6.

La gráfica 8.7(a) no es acíclica y por lo tanto no es árbol. La gráfica 8.7(b), aunque es acíclica, no es conexa, por lo que tampoco corresponde a un árbol generador.

Hay varias maneras de construir un árbol generador. Si la gráfica no tiene pesos, por ejemplo, podemos usar el algoritmo que dimos para BFS para determinar un árbol generador de la gráfica. Recuerden que en la subgráfica de distancias resultante, cada vértice tiene únicamente un predecesor, que es el padre en el árbol generador. No tiene ciclos, pues cuando se llega a un vértice y no es la primera vez, esta arista nos define una trayectoria al vértice en cuestión, distinta de la que se había determinado antes; pero esta arista no se

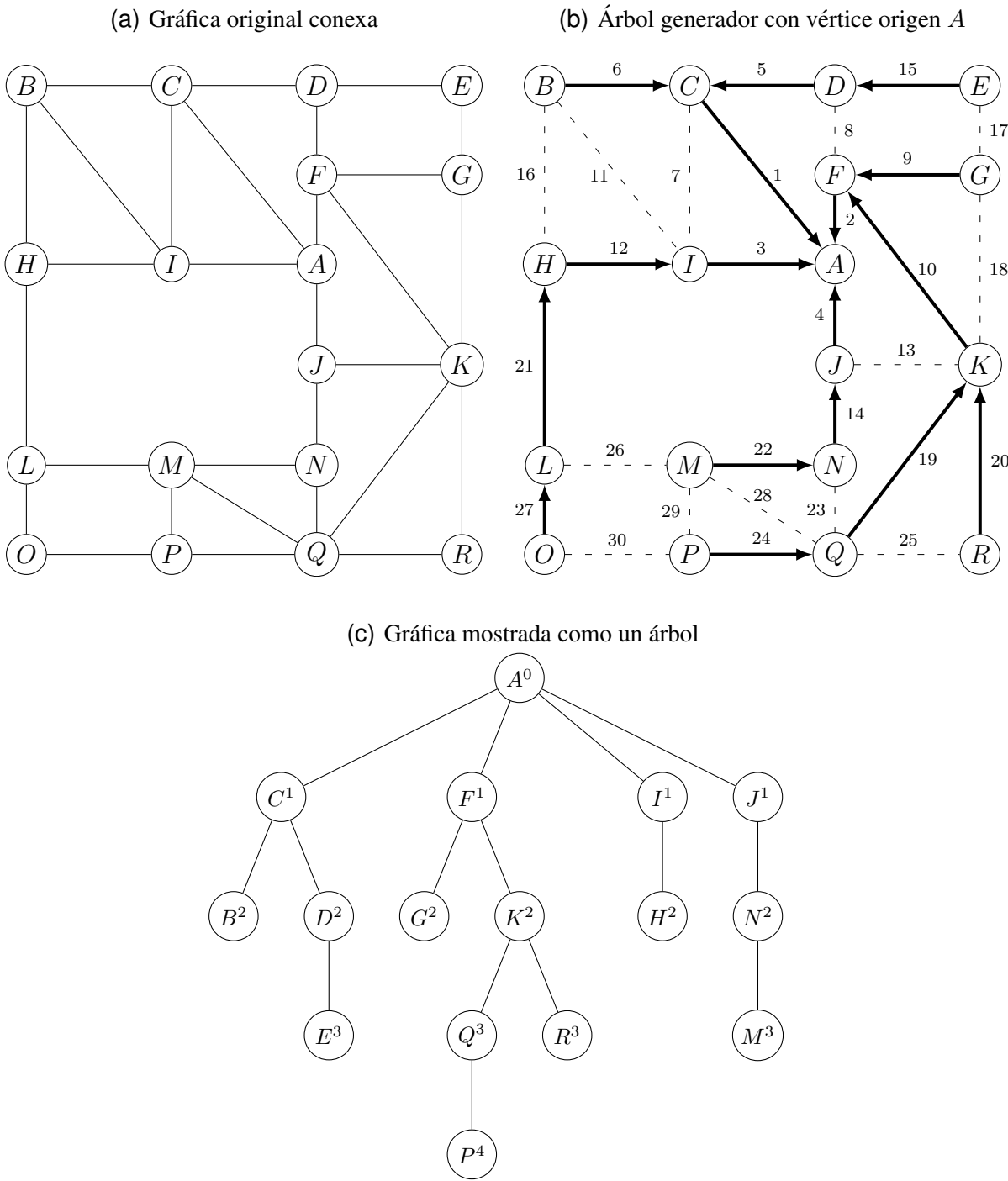
Figura 8.6 Árbol generador en una gráfica G **Figura 8.7** Subgráficas que no son árboles generadores de G 

incluye en el árbol, por lo que no se cierra el ciclo. Este árbol incluye a todos los vértices de la gráfica, si la gráfica originalmente es conexa, pues llega a los vértices adyacentes al inicio, a los adyacentes a éstos, y así sucesivamente. Y es conexa ya que siempre estamos recorriendo caminos desde el origen. En la figura 8.8 en la siguiente página mostramos un recorrido BFS en una gráfica y el árbol generador que produce. Las aristas están etiquetadas con el ordinal en que fueron usadas.

Si la gráfica tiene pesos, lo mismo podemos decir del algoritmo de Dijkstra, que construye un árbol generador de trayectorias más cortas. Hay otros algoritmos que también son muy famosos, como el algoritmo de exploración en profundidad (*Depth First Search, DFS*)

que también determinan árboles generadores de las gráficas sobre las cuales se ejecutan. Lo veremos en la siguiente sección.

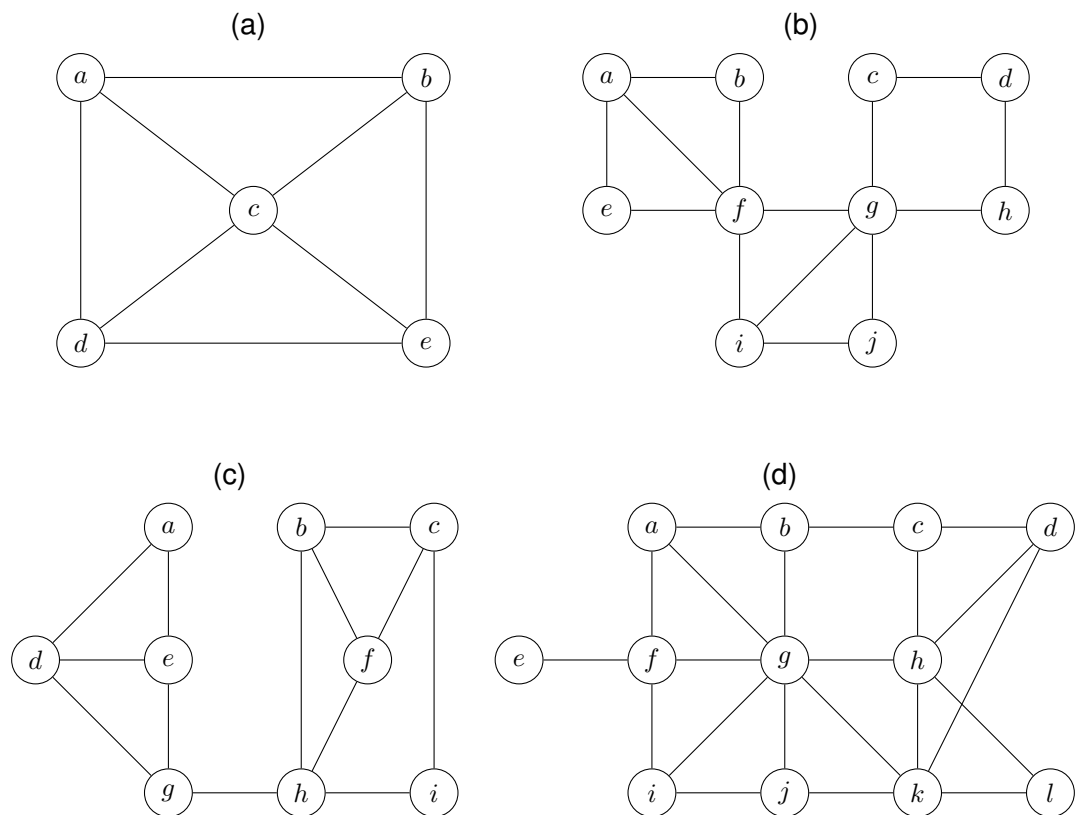
Figura 8.8 Árbol generador determinado por BFS, con origen en el vértice A



Ejercicios

8.2.1.- ¿Cuántas aristas se tienen que eliminar de una gráfica conexa con n vértices y m aristas para producir un árbol generador?

8.2.2.- Para las siguientes gráficas, dibuja un árbol generador para cada una de ellas.

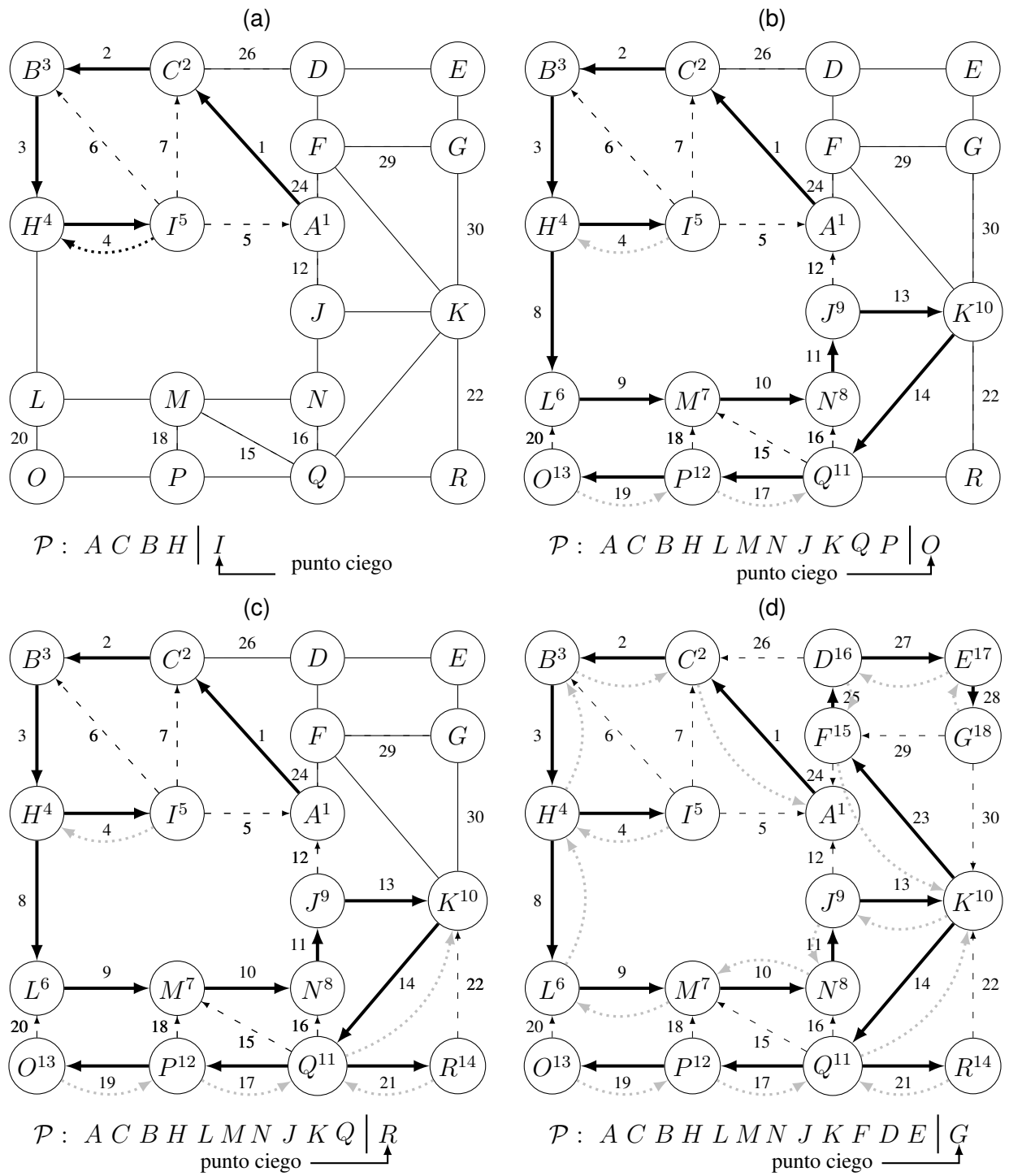


8.2.3.- Supongamos que tenemos dos árboles generadores para una misma gráfica. ¿Estos árboles tienen algún vértice en común? Si sí, justifica. Si no, da un contraejemplo.

8.2.4.- ¿Cuántos árboles generadores distintos hay para un ciclo con n vértices, $n \geq 3$? ¿Cuántos hay si consideramos árboles isomorfos entre sí?

8.2.5.- Muestra que una arista que al removerla desconecta a una gráfica conexa forma parte de todo árbol generador de la gráfica.

8.2.6.- Dibuja el árbol generador que se forma aplicando el algoritmo BFS a \mathcal{K}_n .

Figura 8.10 Árbol generador determinado por DFS, con origen en el vértice A

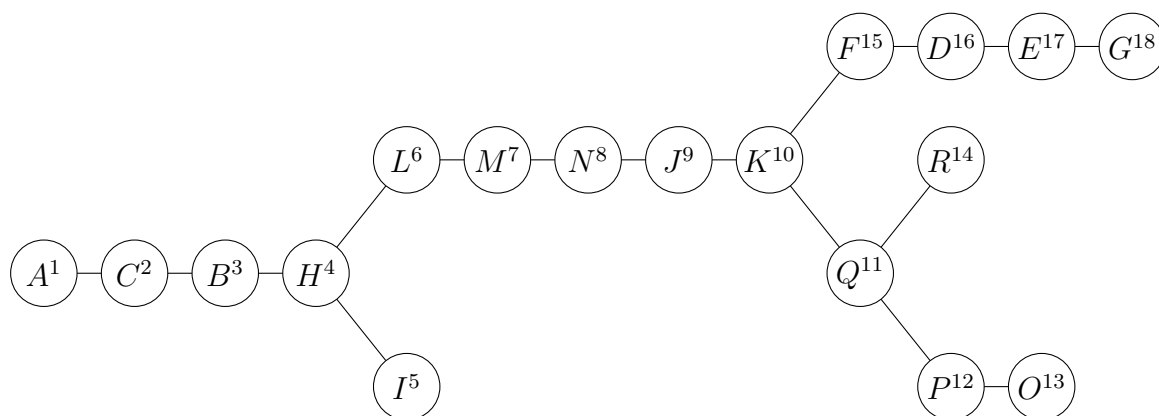
Como ya mencionamos, en la figura 8.10 mostramos la ejecución del algoritmo DFS. Cada subfigura corresponde al avance de la exploración hasta que se encuentra un vértice por el que ya no se puede continuar – marcado al final de la lista –. En este punto, retrocede – marcado con flecha punteadas en gris claro – hasta un vértice desde el que pueda volver a salir.

Durante el recorrido iremos colocando a los vértices una etiqueta que corresponderá al orden en que el vértice fue descubierto. También anotaremos en las aristas el orden en el que fueron consideradas. Asimismo, mantendremos un conjunto con aquellos vértices que ya han sido visitados y otro con las aristas que se van eligiendo para el árbol generador.

Para decidir el orden en que se van visitando los vértices usaremos lo que se conoce como una pila: es una estructura en la que el último que entra es el primero que sale (*Last In First Out, LIFO*), que llamaremos \mathcal{P} . Las operaciones válidas en una pila son **pop**, que elimina al último vértice que se agregó, **push(v)** que ingresa al vértice v en la pila y **top** que informa cuál es el elemento que se encuentra en el tope, el último que fue agregado. Si al buscar vértices adyacentes sin marcar hay varios vértices que se pueden seleccionar, se elegirá al que tenga la etiqueta menor en orden alfabético.

El árbol generado por este recorrido se encuentra en la figura 8.11. Procederemos a explicar cada una de las subfiguras que representan al recorrido.

Figura 8.11 Árbol generado por DFS en la gráfica de la figura 8.11



En la gráfica de la subfigura 8.10(a), se recorre desde A hasta llegar al vértice I ; para este vértice, todas las aristas tienen como destino a un vértice ya visitado, por lo que es un punto ciego. Se retrocede hasta el vértice H que tiene todavía aristas disponibles.

En la gráfica de la subfigura 8.10(b), se recorre desde H hasta el vértice O , siguiendo siempre el orden dado por el alfabeto. En este punto, la única arista disponible de O va hacia L , que ya fue visitado, por lo que O es un punto ciego y se procede a regresar. En P

sucede lo mismo, esto es, que la única arista disponible es a un vértice ya visitado, por lo que se continúa el retroceso. En Q , si bien dos de sus aristas disponibles son a vértices ya visitados (M y N), la arista hacia R aún no ha sido usada y R no ha sido descubierto, por lo que la exploración se continúa desde Q .

En la gráfica de la subfigura 8.10(c), se recorre la arista que va de Q a R , pero como la única arista disponible de R es hacia K que ya fue descubierto, R corresponde a un punto ciego, por lo que hay que retroceder para encontrar algún vértice que tenga aristas disponibles a vértices aún no descubiertos. Este retroceso se hace sobre la pila, en orden inverso a como fueron ingresando los vértices, y nos deja en el vértice K que aún tiene aristas disponibles a vértices que todavía no han sido visitados.

En la gráfica de la subfigura 8.10(d) se encuentra la pila, antes de retroceder. Como ya todas las aristas fueron usadas y los vértices visitados, el retroceso se lleva a cabo hasta que se llega al vértice origen, que es el vértice A , que ya no tiene aristas disponibles.

Veamos a continuación más detenidamente el algoritmo.

Algoritmo de búsqueda a profundidad (DFS)

Objetivo: Dada una gráfica $G = (V, E)$ conexa y un vértice de salida s , construir un árbol generador para G , que realice la exploración a profundidad.

Datos: La gráfica $G = (V, E)$ – representada por su lista de adyacencias – y un vértice s que pertenezca a G , que será el vértice inicial en la exploración.

Salida: Una gráfica $G' = (V, E')$ que corresponda a un árbol generador de G . Esta gráfica deberá tener anotado lo siguiente:

- (a) El orden en que los vértices son alcanzados ($v.etiq$).
- (b) Una referencia al vertice desde el que se les alcanzó por primer vez ($v.\pi$).
- (c) La marca de que ya fueron alcanzados ($v.visitado$).

Estructuras de datos:

1. La gráfica representada con listas de incidencia.
2. Una pila \mathcal{P} para dar el orden en que se usan los vértices.
3. Cada vértice quedará apuntando a su predecesor y de esta forma se va armando el árbol.
4. La lista de aristas donde se marcará a cada una cuando ya haya sido usada.

Método: Se encuentra en el listado 8.1.

Listado 8.1 Algoritmo de búsqueda a profundidad (DFS)

1/2

```

1  /* Inicio : */
2   $\forall v \in V$  :
3       $v.visitado \leftarrow no$ 
4       $v.\pi \leftarrow \emptyset$ 
5       $v.etiq \leftarrow 0$ 
6   $\forall e \in E$ 
7       $e.usada \leftarrow no$ 

```

Listado 8.1 Algoritmo de búsqueda a profundidad(DFS)

2/2

```

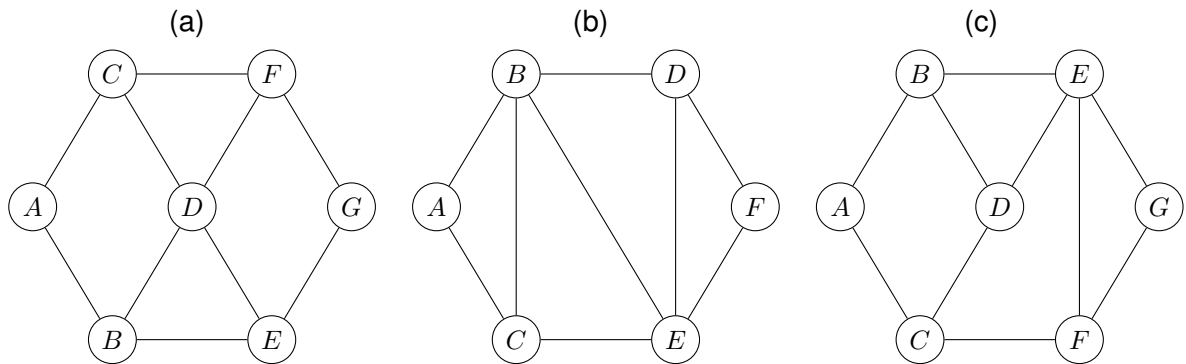
8      /* Marcar al vértice inicial */
9      s.etiq ← 1
10     s.visitado ← ya
11     k ← 2 // iniciar el contador
12     P.push(s)
13     /* Etiquetar el resto de los vértices */
14     Mientras que  $P \neq \emptyset$ :
15         x ← P.tope
16         Mientras haya algún vértice v adyacente a x sin marcar:
17             v.visitado ← ya
18             P.push(v)
19             e = xv
20             e.usada ← ya
21             v.π ← x
22             v.etiq ← k++
23             x ← P.tope
24         /* Fin ciclo Mientras haya vértices .\ .\ . */
25         /* Ya no hay vértices adyacentes a x sin visitar */
26         P.pop
27     // Ya se procesaron todos los vértices alcanzables desde s
28     Todos los vértices están marcados:
29         El árbol generador está dado por las referencias a
30         v.π y tenemos un árbol generador de G
31     No todos los vértices están marcados:
32         G no es conexa y no tiene árbol generador

```

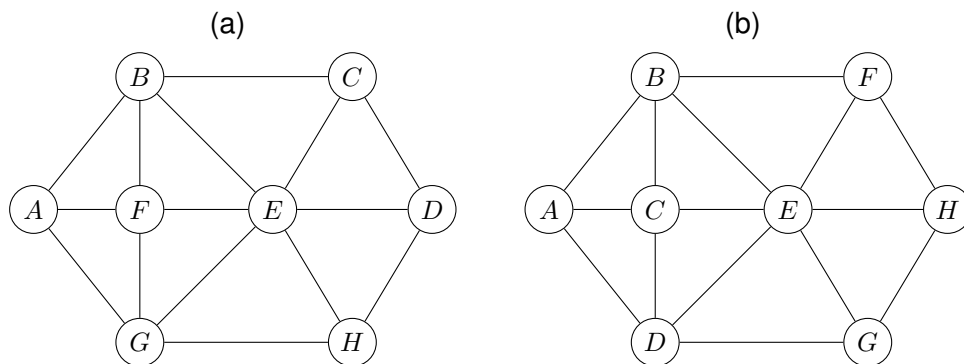
Es interesante observar (en la figura 8.11) la forma que toma el árbol generador construido por DFS. Notemos que las ramas son mucho más profundas que en BFS; por eso este algoritmo recibe el nombre de búsqueda en profundidad.

Ejercicios

8.3.1.- Para las siguientes gráficas, siguiendo los algoritmos respectivos, construye los árboles generadores dados por BFS y DFS.



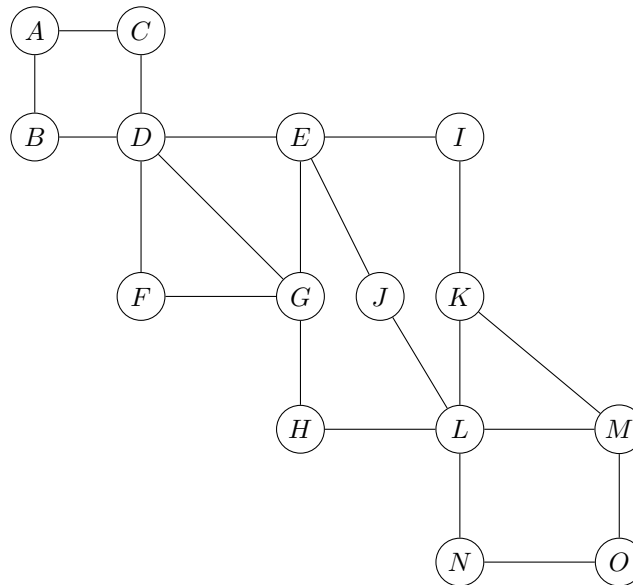
8.3.2.- Encuentra el árbol generador DFS de las siguientes gráficas:



8.3.3.- Explica por qué en el ejercicio anterior, a pesar de que son gráficas isomorfas entre sí, el árbol generado por DFS no es el mismo.

8.3.4.- Usando el algoritmo DFS en los dos ejercicios anteriores, para cada una de las gráficas lista las aristas hacia atrás que se van generando.

8.3.5.- El Jefe de Gobierno de la Ciudad de México tiene identificada una colonia en la que las calles son muy angostas, pero tienen tráfico en ambos sentidos. Quiere hacer las calles de un solo sentido pero no sabe cómo asignar los sentidos de tal manera que se pueda llegar de cualquier lugar a cualquier otro siguiendo el sentido de las calles. A continuación se encuentra un mapa de las calles de la colonia.



Un estudiante de ciencias de la computación le comentó al Jefe de Gobierno que se podía asignar dirección a las calles usando DFS. Determina la dirección de las calles usando este algoritmo y ve si es posible llegar de cualquier punto a cualquier otro usando esta dirección. Nota: se tiene que asignar dirección también a las aristas que cierran ciclos, aunque se lleve a cabo el regreso al vértice desde el que se explora una arista que lleva a un vértice ya descubierto.

- 8.3.6.- ¿Cómo caracterizarías a la gráfica que produce DFS para poder determinar que se puede llegar de cualquier vértice a cualquier otro?
- 8.3.7.- Para una misma gráfica, ¿pueden coincidir los árboles generadores creados por DFS y BFS? ¿Cuáles son las condiciones en las que esto sucede?

8.4. Árboles generadores de peso mínimo

Al ver exploración en gráficas, en particular los algoritmos BFS y de Warshall, revisamos también el caso de gráficas con peso en las aristas y vimos varios algoritmos para resolver este caso. Tanto BFS como DFS construyen un árbol generador de la gráfica. Pasamos ahora a observar árboles generadores sobre gráficas con peso en las aristas. Cabe aclarar que el árbol de distancias que construye el algoritmo de Dijkstra es también un árbol generador. La diferencia con los árboles generadores de peso mínimo es que en este último caso lo que buscamos minimizar es la suma total de pesos de las aristas que conforman el árbol y no el peso de la única trayectoria a cada uno de los vértices. Este algoritmo

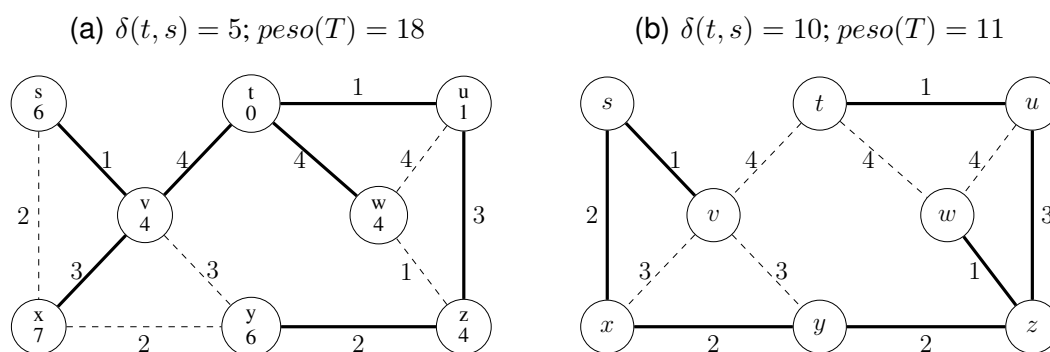
es muy útil, por ejemplo, si tratamos de conectar una red de computadoras buscando que el costo total de la red sea mínimo.

En el caso de BFS por ejemplo, donde todas las aristas tienen peso uniforme, el árbol construido por BFS (o para el caso, también por DFS) es un árbol de peso mínimo, ya que el peso total es $n - 1$. Sin embargo, en el caso del algoritmo de Dijkstra el árbol de trayectorias más cortas construidas puede no ser un árbol de pesos mínimos, como se puede observar en la gráfica de la figura 8.12, que se encuentra abajo.

Cabe notar que el árbol de peso mínimo tiene un peso mayor que la trayectoria del origen al destino. Esto se debe a que en la trayectoria más corta pueden no incluirse todos los vértices, mientras que en el árbol generador de peso mínimo sí tienen que aparecer todos los vértices. Adicionalmente, en el árbol de peso mínimo no importa cuál es el origen de la exploración.

En la gráfica de la figura 8.12 se contrastan un árbol de trayectorias más cortas con uno de peso mínimo. El peso del árbol en la gráfica 8.12(a) es 18, mientras que el árbol de peso mínimo tiene peso total 11. En cambio, la longitud de la trayectoria de t a s en la gráfica 8.12(a) es 5, mientras que en la gráfica 8.12(b) es 10. Se puede ver que cada árbol generador cumple su cometido, aunque pudiera darse el caso de que coincidieran.

Figura 8.12 Árbol generador de trayectorias más cortas vs. de peso mínimo



8.4.1. Algoritmo de Prim para árboles de peso mínimo

El algoritmo de Prim para árboles de peso mínimo es un algoritmo ávido (*greedy*) que toma decisiones localmente. Inicia con un vértice cualquiera de la gráfica y mete a una cola de prioridades a las aristas incidentes en ese vértice que conecten con un vértice que todavía no esté en la cola, con el orden dado por el peso de la arista. En todo momento se encuentra al frente de la cola aquel vértice – con la arista con la que se llegó a él – que tenga el menor peso asociado a la arista. Toma al vértice al frente de la cola e inserta a la cola de prioridades todos aquellos vértices en el extremo opuesto de cada arista incidente al vértice elegido de la cola. Una vez incluidos los nuevos vértices, saca al que acaba de ser

el centro de acción. Prosigue con este proceso hasta que no haya ya vértices sin incluir en la cola.

Conforme se revisan los vértices adyacentes, se sustituyen aristas si es que se encuentra alguna de menor peso que incida en algún vértice que todavía está en la cola.

Se puede demostrar, aunque no es materia de este material, que este algoritmo construye un árbol generador (incluye a todos los vértices en la lista) y que éste es de peso mínimo.

El algoritmo formal se encuentra a continuación.

Objetivo: Dada una gráfica $G = (V, E; w)$, encontrar un árbol generador de peso mínimo $T = (V, E' \subseteq E; w)$, esto es, si $T' = (V, E'' \subseteq E; w)$ es *cualquier otro árbol generador de G* , tenemos

$$\sum_{e \in E'} w(e) \leq \sum_{e \in E''} w(e)$$

Datos: La gráfica $G = (V, E; w)$.

Salida: El árbol generador de peso mínimo.

Estructuras de datos:

1. La gráfica, representada con listas de adyacencia.
2. Una tabla donde apuntaremos, para cada vértice, quién es su predecesor y cuál es la arista que lo incluyó.
3. Una lista donde iremos colocando las aristas que van conformando el árbol.
4. Una cola de prioridades donde iremos colocando los vértices conforme los vamos alcanzando. Ingresa a la cola un vértice con el peso de la arista que lo hizo ingresar. La cola es una *cola de prioridades*, donde podemos elegir al que tenga la mayor prioridad (el menor peso para la arista). Conforme se van examinando aristas, si se encuentra una de peso menor para un vértice en la cola, se modifica este peso. Se elige al vértice que tiene el menor peso anotado.

Método: El algoritmo como se describió al inicio de esta sección se encuentra en el listado 8.2 de la siguiente página.

Listado 8.2 Algoritmo de Prim para árbol generador de peso mínimo

```

1      /* Inicialización: */
2       $\forall v \in V :$ 
3           $v.\pi \leftarrow \text{nulo}$ 
4           $v.\text{peso} \leftarrow \infty$ 
5           $v.\text{arista} \leftarrow \text{nulo};$ 
6       $s.\text{peso} \leftarrow 0; s.\pi \leftarrow \text{nulo}$ 
7       $C_p \leftarrow \langle s \rangle$ 
8
9      /* Procesar al que esté al frente de
10         la cola de prioridades */
11      Mientras  $C_p \neq \emptyset$ 
12           $u \leftarrow C_p.\text{frente}$ 
13          Si  $u.\text{arista} \neq \text{nulo}$ 
14               $T \leftarrow T \cup u.\text{arista}$ 
15          Mientras  $u.\text{listaAdy} \neq \emptyset$ 
16               $v \leftarrow u.\text{listaAdy}.\text{siguiente}$ 
17               $e = uv$ 
18               $w \leftarrow e.\text{peso}$ 
19               $u.\text{listaAdy} \leftarrow u.\text{listaAdy} - \{v\}$ 
20              Si  $v.\text{peso} = \infty$ 
21                   $C_p \leftarrow C_p + v$ 
22              Si  $v.\text{peso} > w$ 
23                   $v.\text{peso} \leftarrow w$ 
24                   $v.\pi \leftarrow u$ 
25                   $v.\text{arista} \leftarrow e$ 
26          /* Fin  $u.\text{listaAdy} == \emptyset$  */
27
28       $C_p \leftarrow C_p - \{u\}$ 
29      /*  $C_p == \emptyset$  */
30      Si  $|T| = |V| - 1$ : entonces  $T$  es el árbol generador de  $G$ 
31                      de peso mínimo
32      Si no:          la gráfica no es conexa.

```

Éste es un algoritmo sencillo de implementar y corresponde, como ya dijimos, a un algoritmo glotón. Veamos la ejecución del algoritmo sobre la misma gráfica que hemos utilizado hasta ahora para ver árboles generadores, pero con pesos asignados a las aristas.

Figura 8.14 Árbol generador de peso mínimo (algoritmo de Prim)

(2/8)

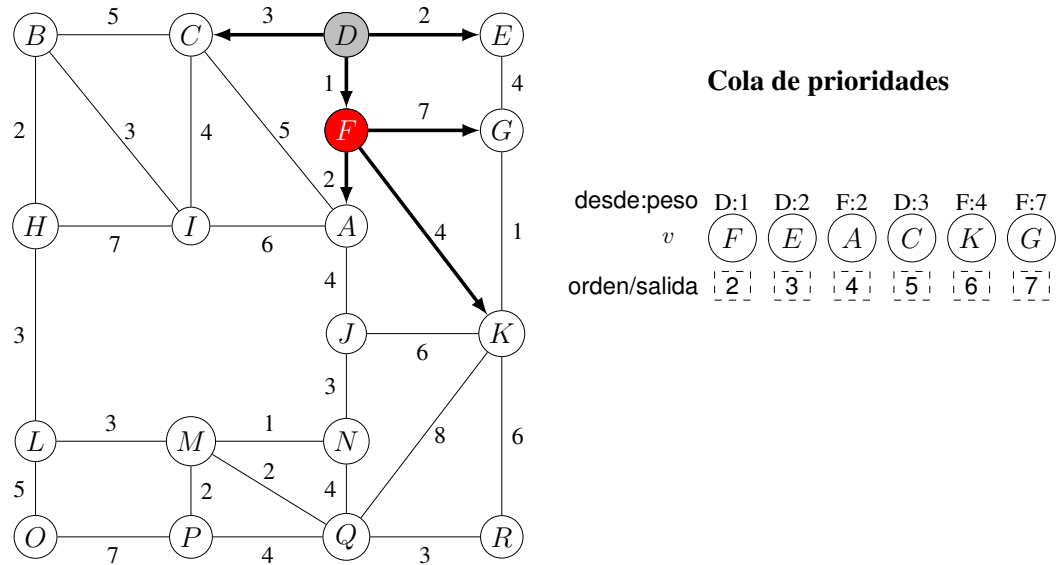
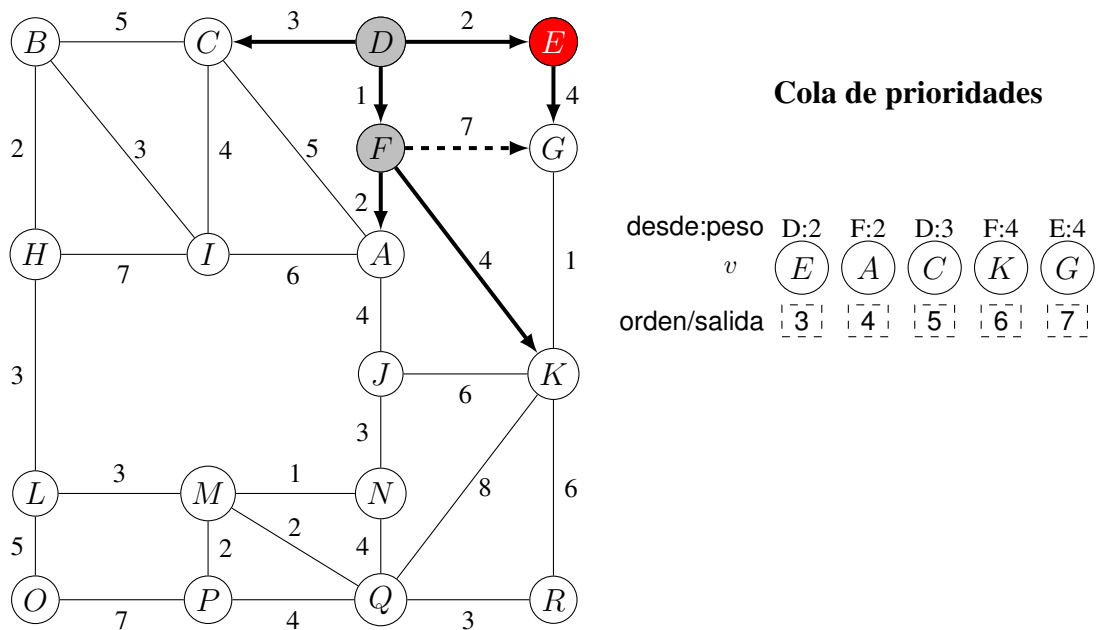
(b) Centro de acción: F (c) Centro de acción: E 

Figura 8.14 Árbol generador de peso mínimo (algoritmo de Prim)

(3/8)

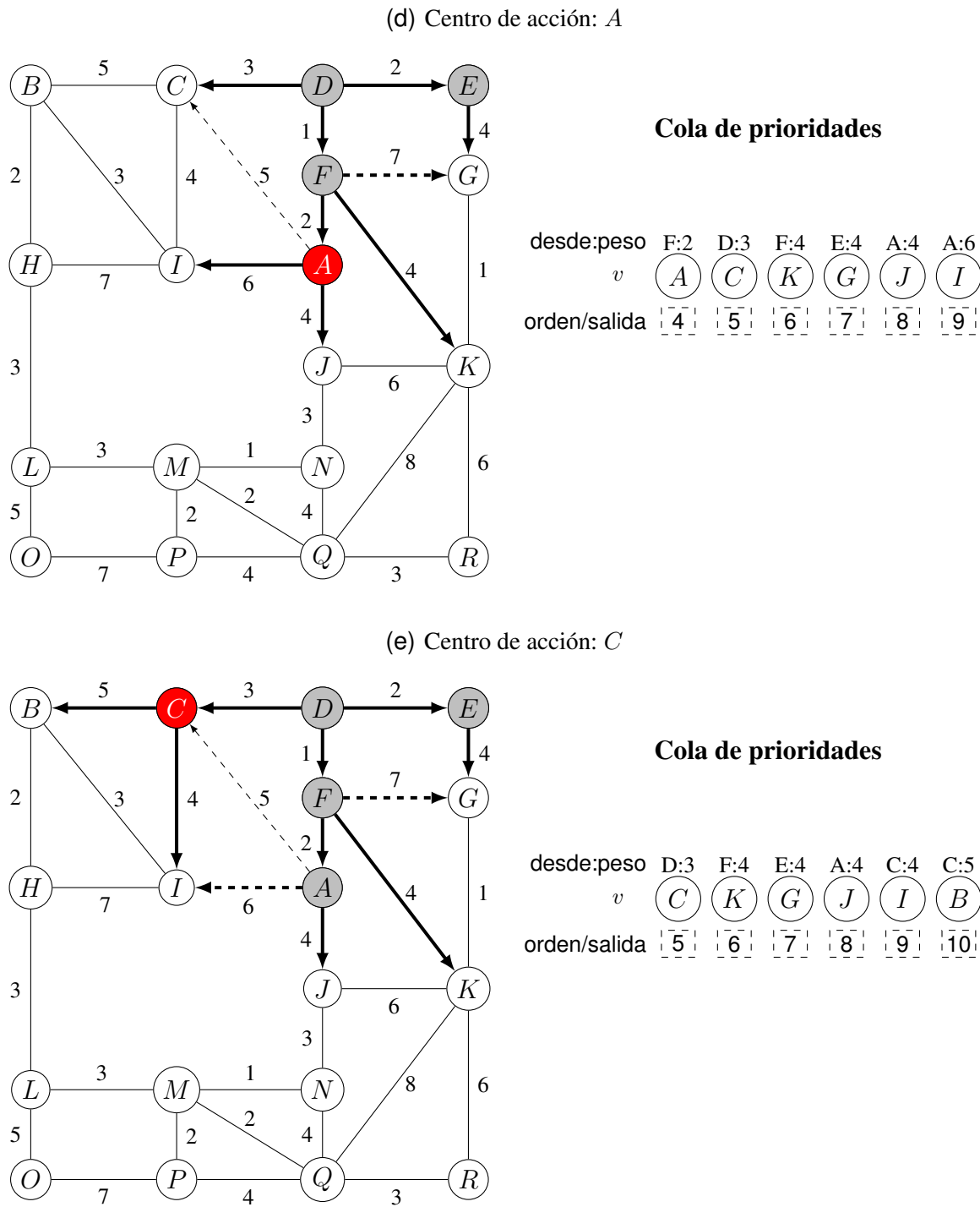
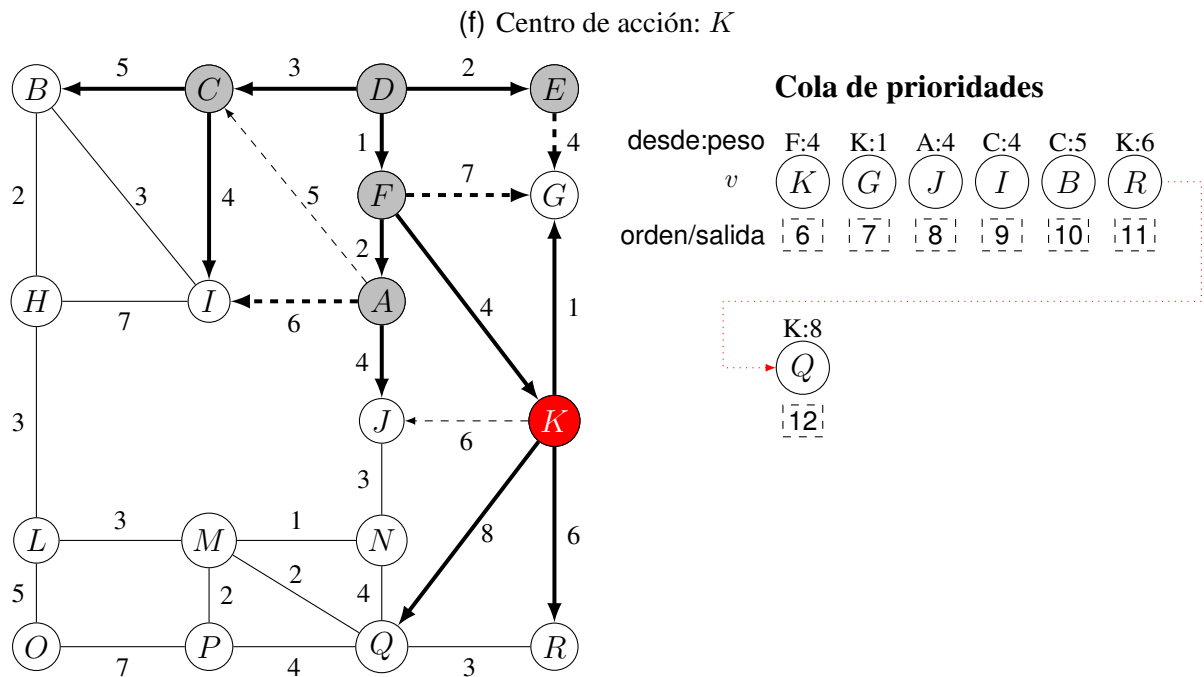


Figura 8.14 Árbol generador de peso mínimo (algoritmo de Prim)

(4/8)



El siguiente centro de acción es G , pero como no tiene aristas sin usar incidentes en él, simplemente se le saca de la cola, así que el siguiente centro de acción es el vértice J .

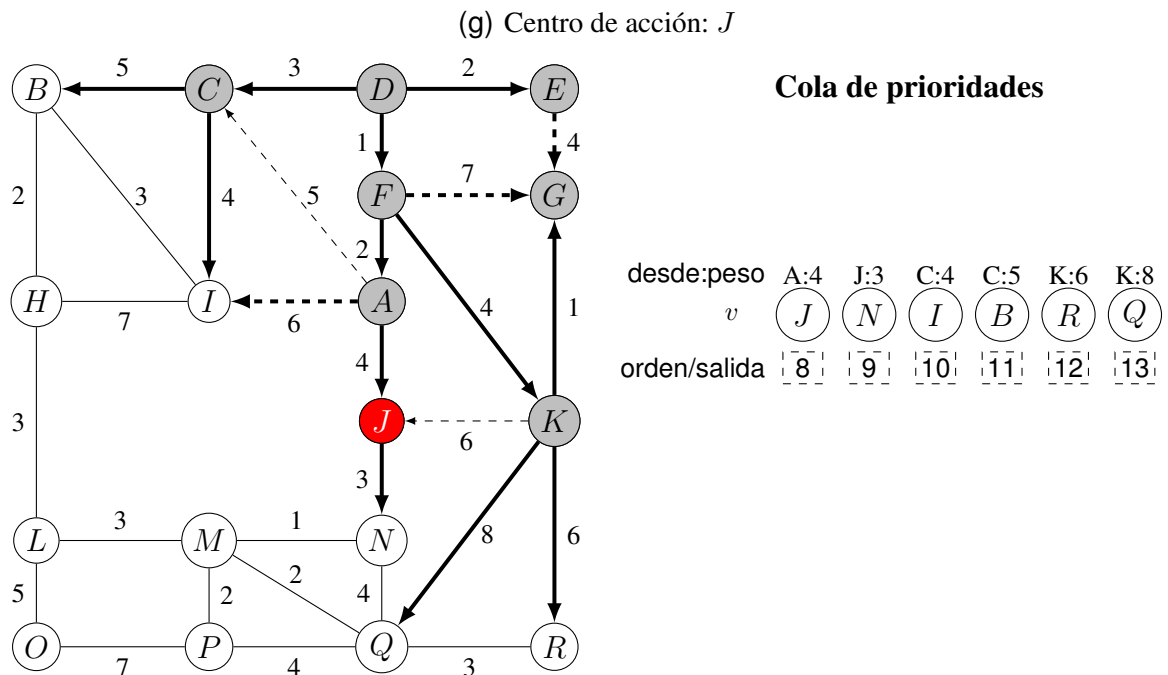


Figura 8.14 Árbol generador de peso mínimo (algoritmo de Prim)

(5/8)

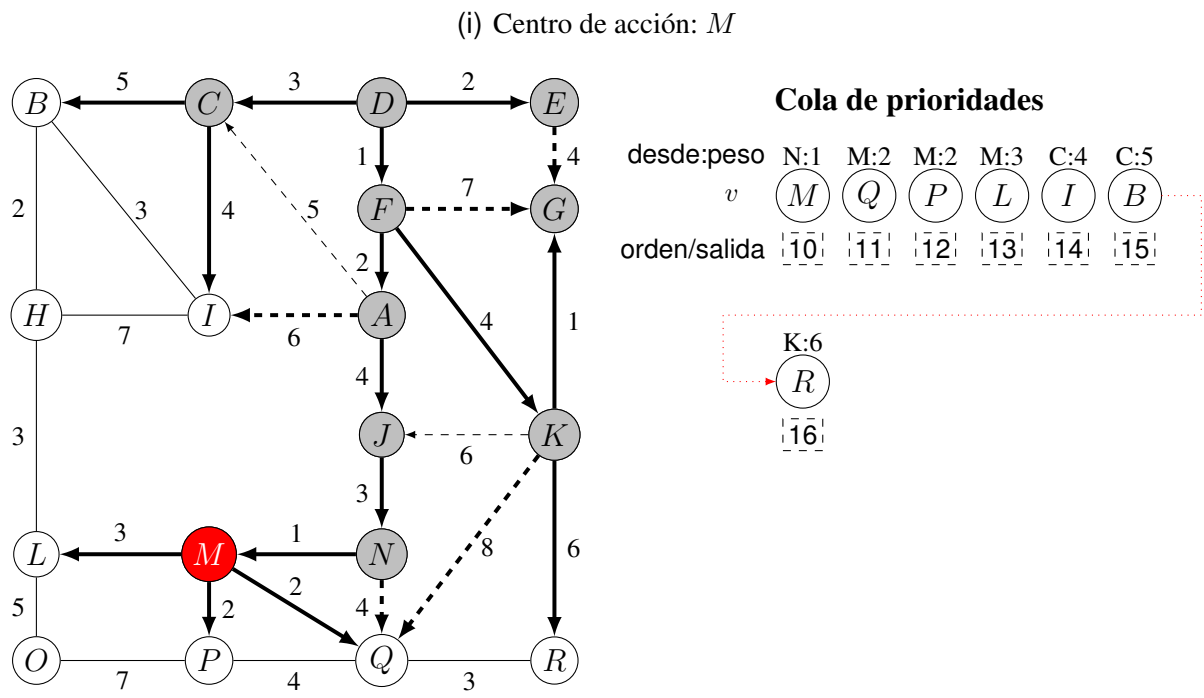
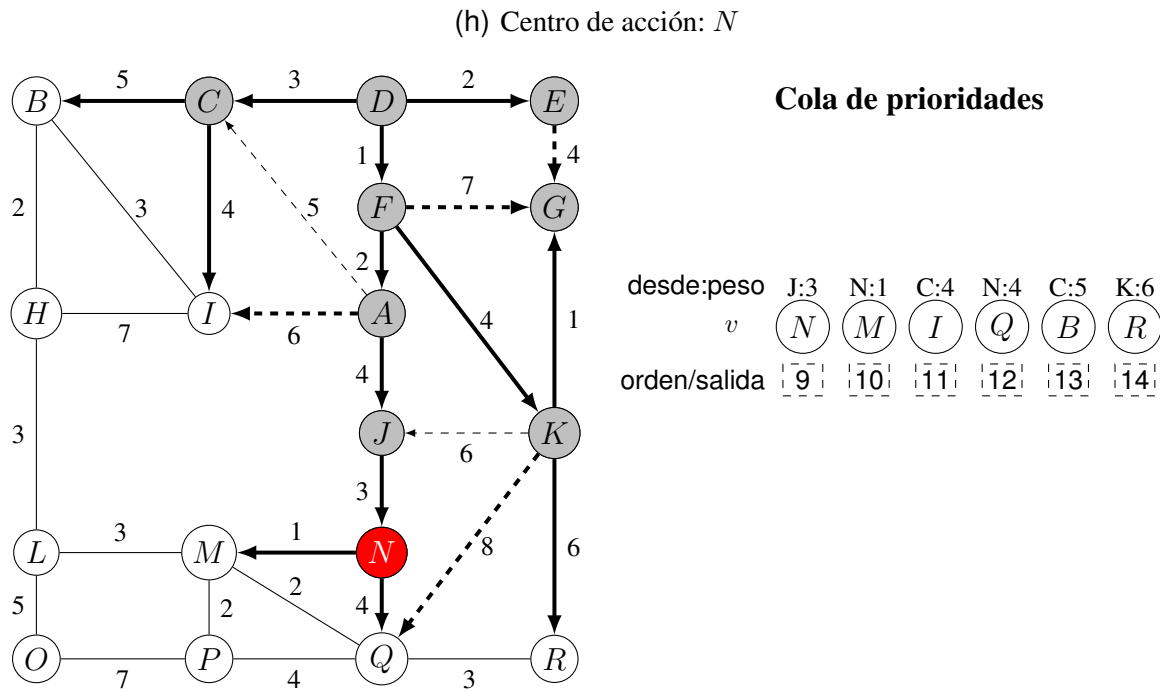


Figura 8.14 Árbol generador de peso mínimo (algoritmo de Prim)

(6/8)

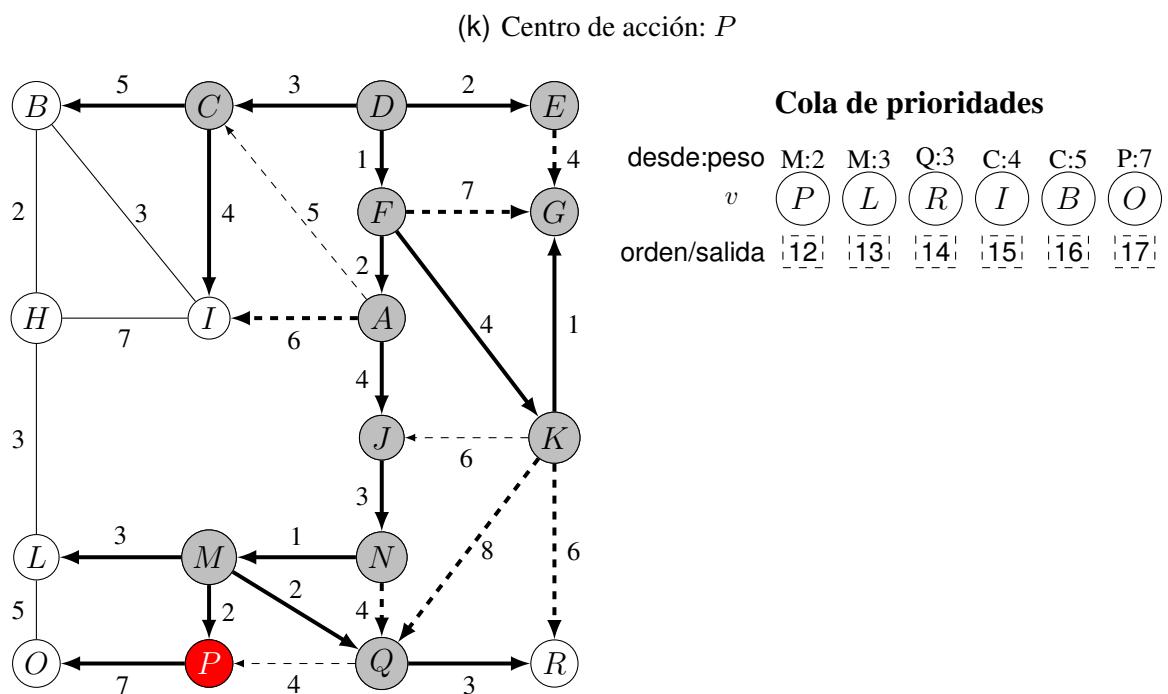
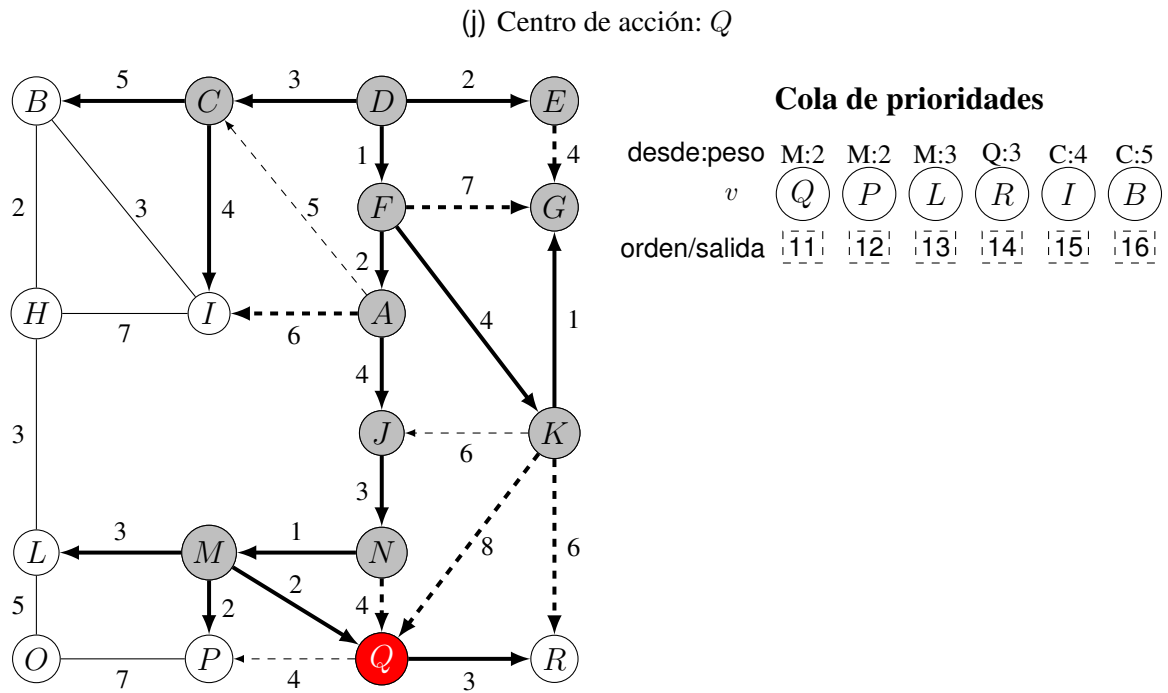
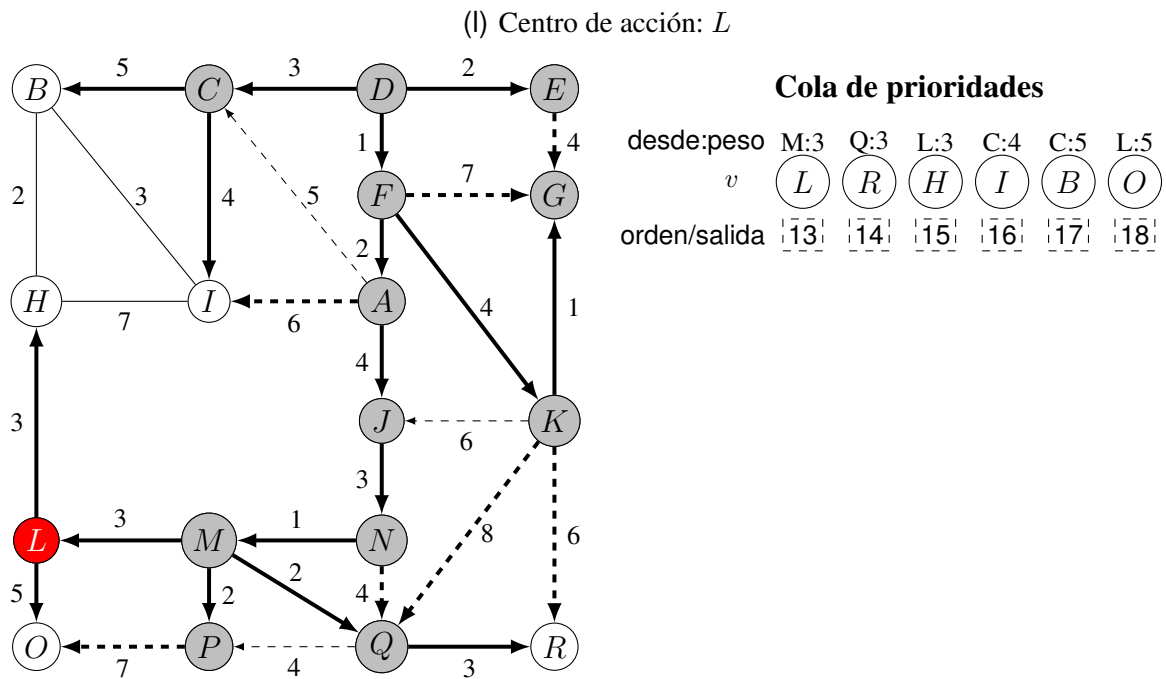


Figura 8.14 Árbol generador de peso mínimo (algoritmo de Prim)

(7/8)



El siguiente centro de acción es *R*, pero como ya no tiene aristas disponibles, simplemente se le saca de la cola y queda *H* al frente de la cola.

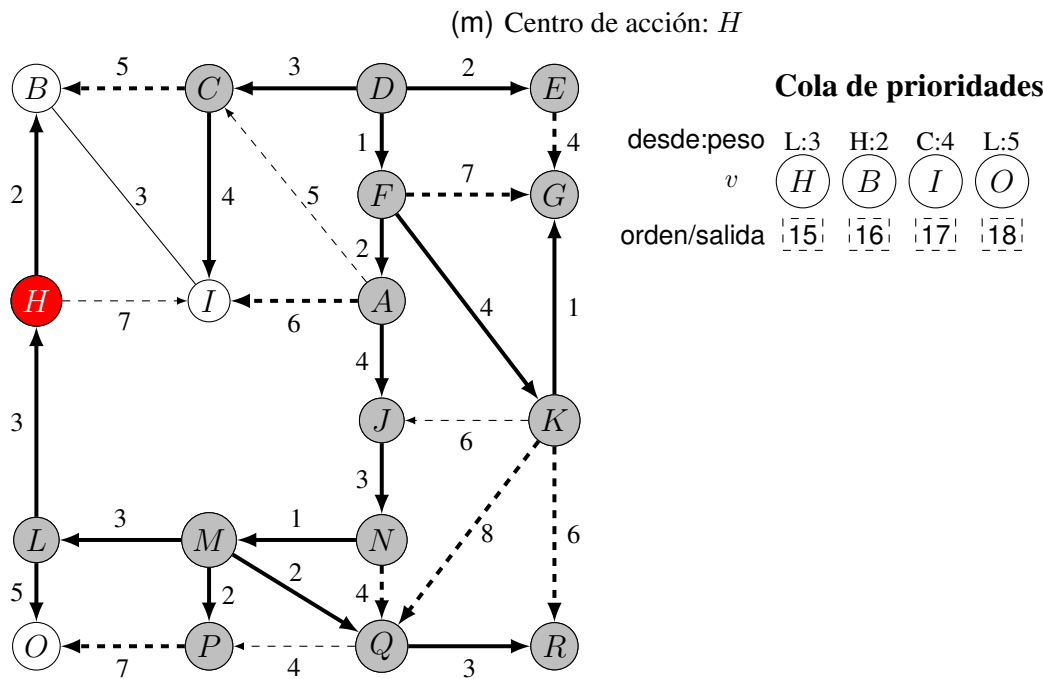
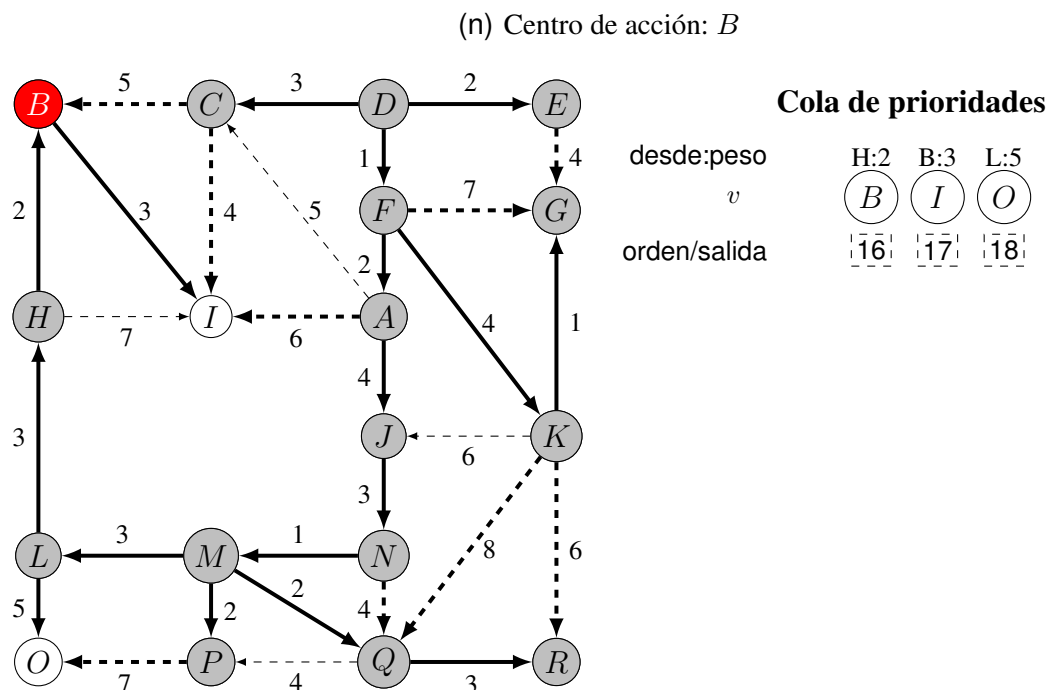


Figura 8.14 Árbol generador de peso mínimo (algoritmo de Prim)

(8/8)



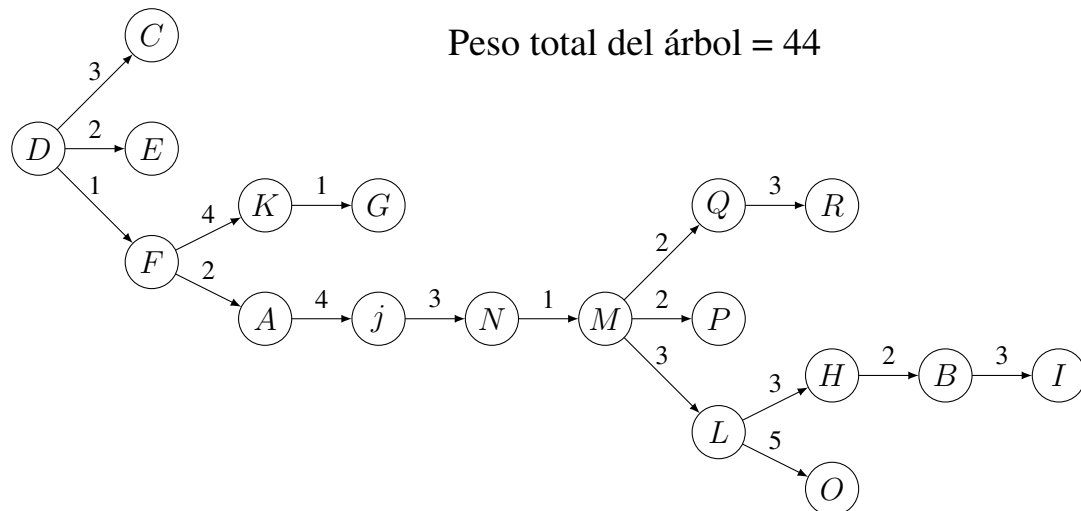
El siguiente centro de acción es *I*, pero como ya no hay aristas disponibles no modifica ya nada en el árbol; lo mismo sucede cuando *O* es centro de acción. En este momento se vacía la cola de prioridades y el algoritmo de Prim termina.

En todas las subgráficas se presentó sombreado el centro de acción, que es aquel vértice que se encuentra al frente de la cola de prioridades y desde el que se van a explorar aquellas aristas que no hayan sido ya exploradas. La simbología que se utilizó es la siguiente:

- Arista aún no considerada.
- Arista del árbol generador de peso mínimo.
- > Arista incluida y después eliminada.
- > Arista no incluida en el árbol generador de peso mínimo.

La gráfica, presentada como árbol y únicamente con las aristas que quedaron en el mismo, queda como se ve en la figura 8.15 (en la siguiente página) y tiene un peso total de 44 unidades.

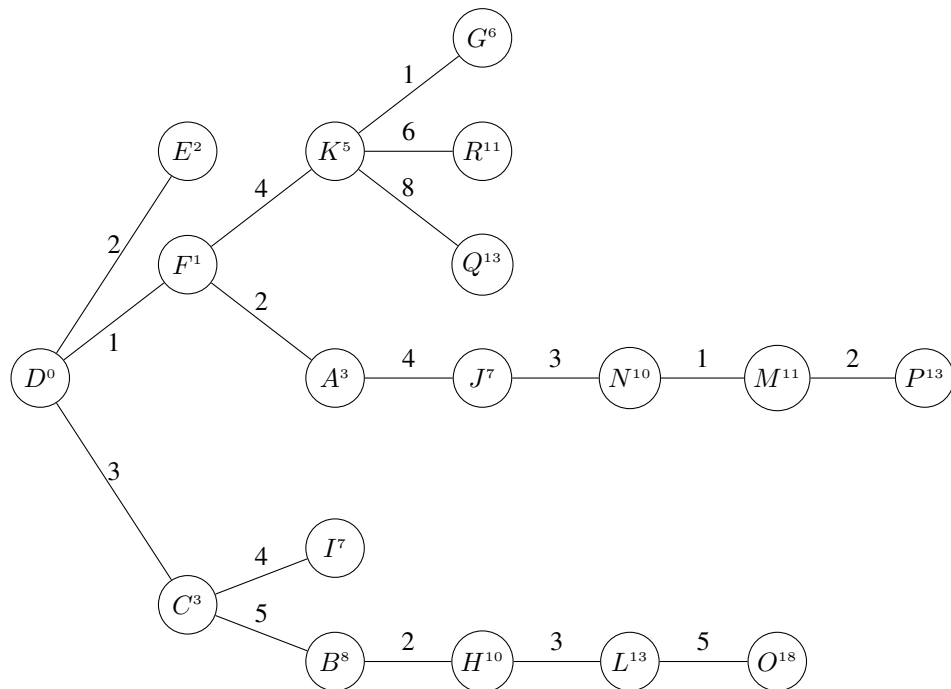
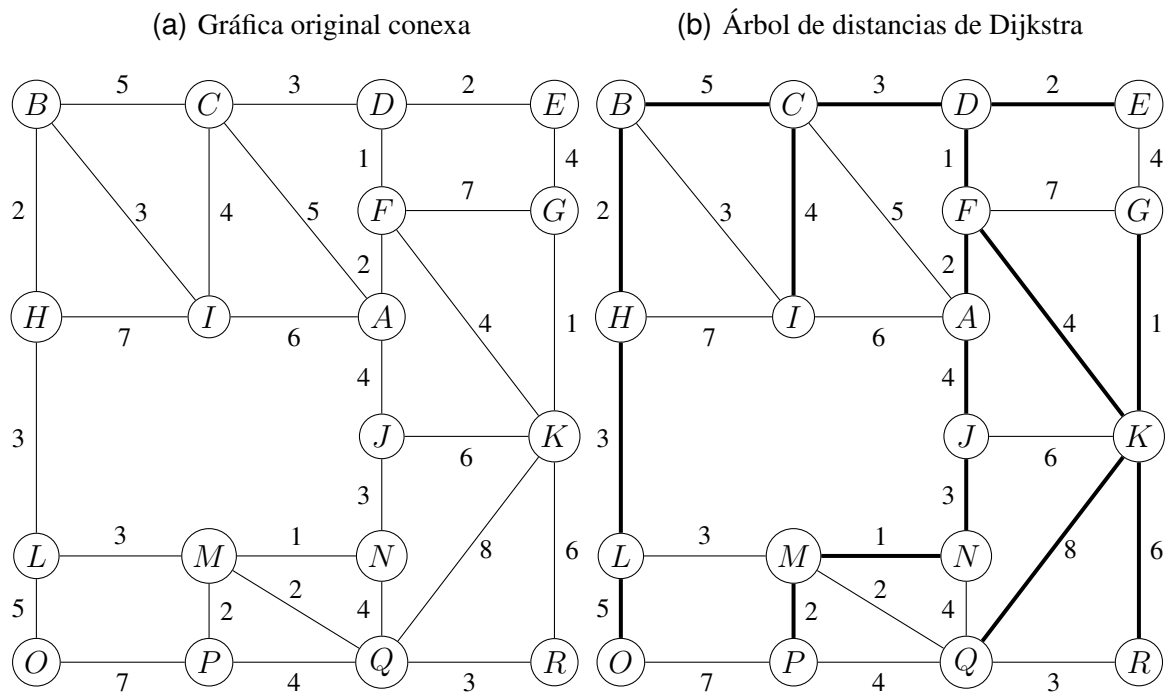
Es importante mencionar que éste no es el único árbol posible de peso mínimo, ya que existen otras combinaciones posibles para árboles en los cuales la suma de los pesos de las

Figura 8.15 Árbol generador de peso mínimo (Prim)

aristas también dé el mínimo de 44 unidades. Esto depende en gran medida del algoritmo que se utilice para mantener la cola de prioridades. Cuando en la cola de prioridades se encuentran dos vértices con el mismo peso, el algoritmo puede respetar el orden en que entraron a la cola; puede optar por el vértice con el nombre lexicográficamente menor; o simplemente elegir aleatoriamente cuál de los vértices con peso igual colocar al frente de la cola. En el ejemplo que vimos se eligió el criterio de respetar el orden de llegada a la cola, además de que cuando se modificó el peso de un vértice para moverlo hacia el frente de la cola, se le colocaba después de cualquier otro vértice que ya tuviera ese mismo peso.

Un ejercicio interesante es observar cómo se comparan un árbol generador de peso mínimo y uno de distancias, generado por el algoritmo de Dijkstra. Para hacer esta comparación mostramos en la figura 8.16 el árbol generador de distancias de Dijkstra aplicando el algoritmo a la misma gráfica. Como se puede ver, los árboles son muy distintos, aun teniendo el mismo vértice como origen. Inmediatamente se puede observar que el árbol de distancias es más bajo y ancho, porque, precisamente, el algoritmo de Dijkstra busca minimizar la distancia desde la raíz a cualquiera de los vértices de la gráfica: busca que todas las trayectorias sean lo más cortas posibles. Por ejemplo, la distancia del vértice *D* al vértice *I* es de 7, mientras que si seguimos la trayectoria dada por el árbol generador de peso mínimo, la longitud de la trayectoria del vértice *D* al vértice *I* es de 22 unidades.

Otro comentario interesante es el hecho de que aunque consideremos otro vértice origen para el árbol generador de peso mínimo, existirá un árbol generador de peso mínimo con origen en *D* que contenga a las mismas aristas. Esto es, si le quitamos la dirección a las aristas del árbol generador de peso mínimo, podemos usar a cualquier otro vértice y “colgar” de ahí el árbol. Eso, en cambio, no sucede con el árbol de distancias, pues éstas están determinadas para un origen particular.

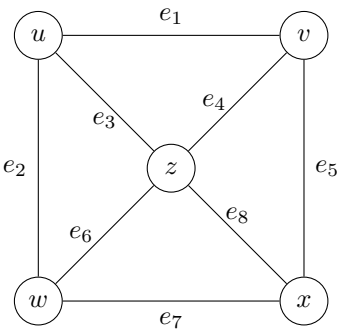
Figura 8.16 Árbol generador de trayectorias más cortas (algoritmo de Dijkstra)

Como ya mencionamos, el árbol de trayectorias más cortas tiene menor profundidad que el de peso mínimo, mientras que éste tiene mayor amplitud que aquél. Es importante notar que el peso total del árbol es de 56 unidades, bastante más que el árbol de peso mínimo. Sin embargo, el vértice más lejano del origen, está a una distancia de 18 unidades (O), mientras que ese mismo vértice está a distancia de 19 unidades del origen en el árbol generador de peso mínimo.

El algoritmo de Prim para árboles generadores de peso mínimo es bastante eficiente. El único costo es, realmente, el de mantener la cola de prioridades, ya que siempre debe tener al frente al vértice al que se llega con la arista de menor peso. Veamos paso a paso la ejecución del algoritmo de Prim con una gráfica más sencilla que las que hemos visto en esta sección.

Ejemplo 8.1. Apliquemos el algoritmo de Prim para árboles generadores de peso mínimo en la gráfica de la figura 8.17. Usaremos como vértice origen al vértice u .

Figura 8.17 Otro ejemplo para el algoritmo de Prim

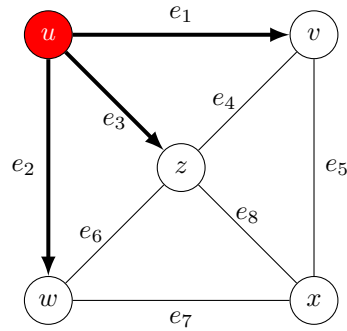


Pasos 2 a 7:

v	$v.\pi$	$v.peso$	$v.arista$	lista de adyacencias
u	<i>nulo</i>	0	<i>nulo</i>	$\rightarrow v \rightarrow z \rightarrow w$
v	<i>nulo</i>	∞	<i>nulo</i>	$\rightarrow u \rightarrow z \rightarrow x$
w	<i>nulo</i>	∞	<i>nulo</i>	$\rightarrow u \rightarrow z \rightarrow x$
x	<i>nulo</i>	∞	<i>nulo</i>	$\rightarrow v \rightarrow z \rightarrow w$
z	<i>nulo</i>	∞	<i>nulo</i>	$\rightarrow u \rightarrow v \rightarrow w \rightarrow x$

Uso	e_i	u	v	$e.peso$
	e_1	u	v	1
	e_2	u	w	4
	e_3	u	z	2
	e_4	v	z	3
	e_5	v	x	3
	e_6	w	z	3
	e_7	w	x	1
	e_8	x	z	2

$$\mathcal{C}_p = \langle (u, 0, \emptyset) \rangle$$

Pasos 12 a 14**Pasos 16 a 27 para v, z y w**

v	$v.\pi$	$v.peso$	$v.arista$	lista de adyacencias
u	<i>nulo</i>	0	<i>nulo</i>	\emptyset
v	u	1	e_1	$\rightarrow u \rightarrow z \rightarrow x$
w	u	4	e_2	$\rightarrow u \rightarrow z \rightarrow x$
x	<i>nulo</i>	∞	<i>nulo</i>	$\rightarrow v \rightarrow z \rightarrow w$
z	u	2	e_3	$\rightarrow u \rightarrow v \rightarrow w \rightarrow x$

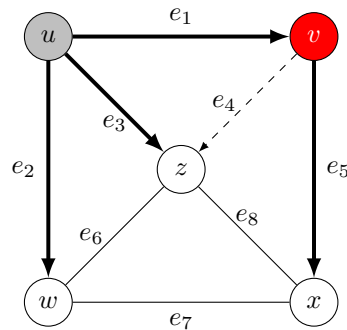
Uso	e_i	u	v	$e.peso$
✓	e_1	u	v	1
✓	e_2	u	w	4
✓	e_3	u	z	2
	e_4	v	z	3
	e_5	v	x	3
	e_6	w	z	3
	e_7	w	x	1
	e_8	x	z	2

$$\mathcal{C}_p = \langle (u, 0, \emptyset), (v, 1, u), (z, 2, u), (w, 4, u) \rangle$$

Paso 28

$$\mathcal{C}_p = \langle (v, 1, u), (z, 2, u), (w, 4, u) \rangle$$

Pasos 12 a 14 $(v, 1)$ está al frente de la cola, por lo que tenemos:



Pasos 15 a 27 2 veces Tomamos los dos vértices adyacentes a v con aristas que no han sido usadas, y las estructuras de datos quedan como sigue:

v	$v.\pi$	$v.peso$	$v.arista$	lista de adyacencias
u	<i>nulo</i>	0	<i>nulo</i>	\emptyset
v	u	1	e_1	\emptyset
w	u	4	e_2	$\rightarrow u \rightarrow z \rightarrow x$
x	v	3	e_5	$\rightarrow v \rightarrow z \rightarrow w$
z	u	2	e_3	$\rightarrow u \rightarrow v \rightarrow w \rightarrow x$

Uso	e_i	u	v	$e.peso$
✓	e_1	u	v	1
✓	e_2	u	w	4
✓	e_3	u	z	2
✓	e_4	v	z	3
✓	e_5	v	x	3
	e_6	w	z	3
	e_7	w	x	1
	e_8	x	z	2

$$C_p = \langle (v, 1, u), (z, 2, u), (x, 3, v), (w, 4, u) \rangle$$

Terminamos la lista de adyacencias de v , lo quitamos de la cola y queda z al frente de la cola.

Pasos 12 a 14 Quitamos a v de la cola y queda z al frente de la misma.

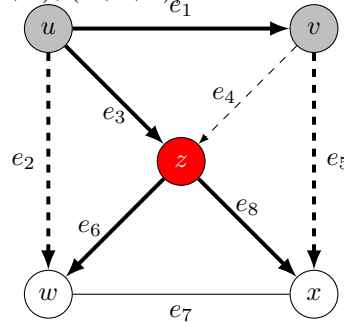
$$C_p = \langle (z, 2, v), (x, 3, v), (w, 4, u) \rangle$$

Pasos 15 a 27 para las aristas e_8 y e_6 Con z como centro de acción verificamos a la arista a w (e_6) y a x (e_8). Como la arista e_8 pesa menos que e_5 , cambia los atributos de x y la sustituye en el árbol; lo mismo que pasa con la arista e_6 a w , que pesa menos que e_2 , por lo que también corrige los atributos de w .

v	$v.\pi$	$v.peso$	$v.arista$	lista de adyacencias
u	<i>nulo</i>	0	<i>nulo</i>	\emptyset
v	u	1	e_1	\emptyset
w	z	3	e_6	$\rightarrow u \rightarrow z \rightarrow x$
x	z	2	e_8	$\rightarrow v \rightarrow z \rightarrow w$
z	u	2	e_3	\emptyset

Uso	e_i	u	v	$e.peso$
✓	e_1	u	v	1
✓	e_2	u	w	4
✓	e_3	u	z	2
✓	e_4	v	z	3
✓	e_5	v	x	3
✓	e_6	w	z	3
	e_7	w	x	1
✓	e_8	x	z	2

$$C_p = \langle (z, 2, u), (x, 2, z), (w, 3, z) \rangle$$

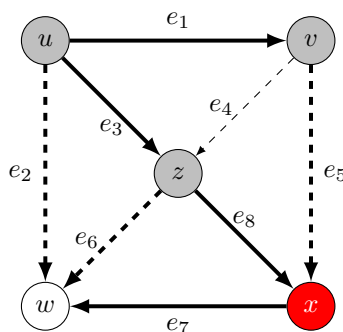


Pasos 12 a 14 con x al frente de la cola: La única arista que queda por explorar es e_7 , que lleva a w con peso 1; como este peso es menor que el que tenía w , se sustituye a la arista e_6 por e_7 y se ajusta a w para que sea alcanzado por esta arista. Las estructuras de datos quedan como sigue:

v	$v.\pi$	$v.peso$	$v.arista$	lista de adyacencias
u	<i>nulo</i>	0	<i>nulo</i>	\emptyset
v	u	1	e_1	\emptyset
w	x	1	e_7	\emptyset
x	z	2	e_8	\emptyset
z	u	2	e_3	\emptyset

Uso	e_i	u	v	$e.peso$
✓	e_1	u	v	1
✓	e_2	u	w	4
✓	e_3	u	z	2
✓	e_4	v	z	3
✓	e_5	v	x	3
✓	e_6	w	z	3
✓	e_7	w	x	1
✓	e_8	x	z	2

$$C_p = \langle (x, 2, z), (w, 1, x) \rangle$$

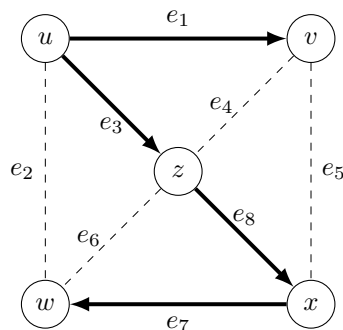


Paso 28 Se quita a x de la cola, quedando ya únicamente w .

$$C = \langle (w, 1, x) \rangle$$

Pasos 15 a 28 Como w ya no tiene ninguna arista sin usar, no entra al ciclo y en la línea 28 saca a w de la cola y deja la cola vacía. Esto hace que ya no vuelva a entrar al ciclo en la línea 11 y salta hasta la línea 30.

Pasos 30 a 32 Como $T = \langle e_1, e_3, e_8, e_7 \rangle$, que consiste de 4 aristas, una menos que el número de vértices, la gráfica original es conexa y T es un árbol generador de peso mínimo.



El algoritmo de Prim para árboles generadores de peso mínimo es un algoritmo eficiente donde el único problema es mantener la cola de prioridades de tal manera que siempre se pueda elegir fácilmente al siguiente vértice que va a ser el centro de acción – desde el que se van a explorar aristas que no han sido usadas. Como los valores que dan la prioridad pueden cambiar en cada paso, la cola debe actualizarse constantemente. Si esto no se hace de manera adecuada, el costo de elegir el vértice con mayor prioridad en la línea 12 puede subir el costo del algoritmo de manera notoria.

8.4.2. Algoritmo de Kruskal para árboles de peso mínimo

Otro algoritmo que también obtiene un árbol generador de peso mínimo es el algoritmo de Kruskal. En él, las aristas se ordenan por peso y se van tomando en orden (de menor a mayor para peso mínimo). A los vértices se les coloca a cada uno en un conjunto que contiene únicamente a ese vértice. Para cada arista que se va tomando (en orden) se verifica si sus extremos están o no en el mismo conjunto. Si es así, al agregar la arista se estaría cerrando un ciclo, por lo que la arista se desecha. Si los vértices no están en el mismo conjunto, entonces se agrega la arista al árbol que se está construyendo. De esta manera, los vértices en un conjunto forman una subgráfica conexa y acíclica. Al terminar, si todos los vértices quedaron en el mismo subconjunto, esas aristas conforman el árbol generador de peso mínimo.

Supongamos que la arista $e = uv$ es la siguiente sin usar en la lista con menor peso de entre las que quedan en ella. Si u y v están en el mismo conjunto, quiere decir que hay un camino entre ellos, y la arista se descarta, pues al agregarla se formaría un ciclo. Si u y v están en distintos conjuntos, entonces se agrega la arista y se construye un nuevo conjunto que es la unión del conjunto que contiene a u y el que contiene a v . En el caso en que la gráfica sea conexa, el algoritmo termina cuando todos los vértices están en un único conjunto, pues al agregar cualquier otra arista se formaría un ciclo. Si la gráfica no es conexa, el algoritmo alcanzará un punto fijo en el que ya no será posible agregar aristas y quedarán tantos subconjuntos como componentes conexas tenga la gráfica. En el listado 8.3 se encuentra a detalle el algoritmo de Kruskal.

Algoritmo de Kruskal

Objetivo: Encontrar el árbol generador de peso mínimo de una gráfica usando el algoritmo de Kruskal.

Datos: Una gráfica representada por la lista de sus aristas. Cada arista tiene los siguientes atributos: vértices en los extremos y peso.

Salida: El árbol generador de peso mínimo de la gráfica, si es que la gráfica original es conexa.

Estructuras de datos:

- i. Una lista de las aristas ordenadas por peso.
- ii. Subconjuntos de vértices, donde inicialmente cada vértice pertenece a un subconjunto que sólo lo contiene a él.

Método: El algoritmo va tomando arista por arista en el orden en que vienen por peso. Si los extremos de la arista se encuentran en conjuntos ajenos, se incluye la arista en el árbol generador y se hace la unión de los subconjuntos que contienen a los vértices. Si ambos vértices se encuentran en el mismo conjunto, entonces la arista se desecha.

Dos vértices se encuentran en el mismo subconjunto si hay un camino en el árbol generador de uno al otro. Si en ese momento se agrega la arista que se está inspeccionando, se formaría un ciclo. Los pasos precisos se encuentran en el listado 8.3.

Listado 8.3 Algoritmo de Kruskal para árbol generador de peso mínimo

```

1  /* Inicio: ordenar lista de aristas */
2   $\mathcal{C} = \langle \text{lista ordenada de aristas} \rangle$ 
3  /* Construir los subconjuntos para cada vértice */
4   $\forall v \in V$  construir  $S_v$ 
5  /* Iniciar el árbol generador */
6   $T = \emptyset$ 
7  /* Procesar las aristas en orden */
8  Mientras  $\mathcal{C} \neq \emptyset$  y  $|T| < |V| - 1$ 
9       $e \leftarrow$  arista al frente de la cola
10      $\mathcal{C} \leftarrow \mathcal{C} - \{e\}$ 
11     Sean  $e = uv$ 
12     Si  $S_u \neq S_v$  // Si están en distinto subconjunto
13          $S_{uv} \leftarrow S_u \cup S_v$ 
14          $T \leftarrow T \cup \{e\}$ 
15     /* ... Mientras */
16     Si  $|T| = |V| - 1$ 
17          $T$  es el árbol generador de peso mínimo
18     Si no
19          $G$  no es conexa.

```

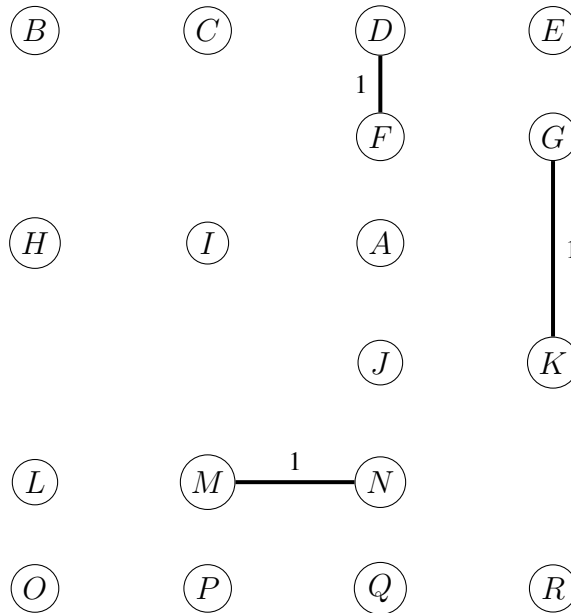
Ejemplo 8.2. Veamos el árbol generador que se forma con el algoritmo de Kruskal en la gráfica con la que hemos estado trabajando. Mostraremos también cómo se van formando los conjuntos.

Figura 8.18 Lista ordenada de aristas al empezar

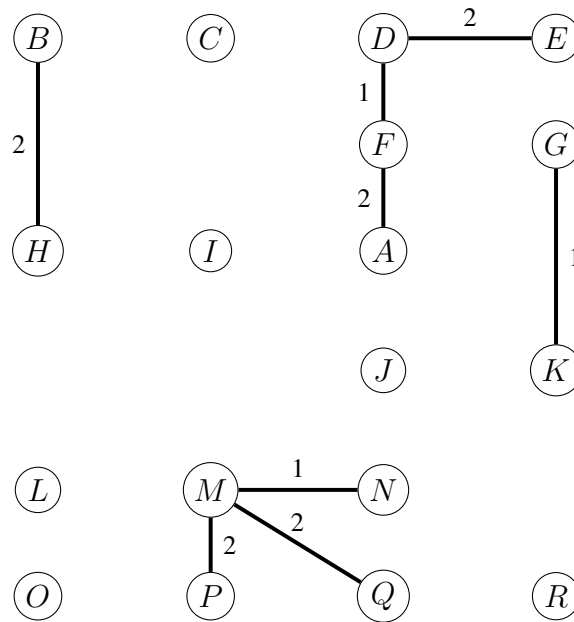
$D \overset{1}{-} F, G \overset{1}{-} K, M \overset{1}{-} N, B \overset{2}{-} H, M \overset{2}{-} P, A \overset{2}{-} F, M \overset{2}{-} Q, D \overset{2}{-} E, C \overset{3}{-} D, B \overset{3}{-} I, Q \overset{3}{-} R,$
 $J \overset{3}{-} N, L \overset{3}{-} M, H \overset{3}{-} L, C \overset{4}{-} I, E \overset{4}{-} K, P \overset{4}{-} Q, A \overset{4}{-} J, N \overset{4}{-} Q, B \overset{5}{-} C, L \overset{5}{-} O, A \overset{5}{-} C,$
 $A \overset{6}{-} I, J \overset{6}{-} K, K \overset{6}{-} R, F \overset{7}{-} G, H \overset{7}{-} I, O \overset{7}{-} P, K \overset{8}{-} Q,$

Formemos cada vértice en su propio conjunto y veamos el resultado de incluir al árbol todas las aristas con peso 1. como se muestra en la figura 8.19:

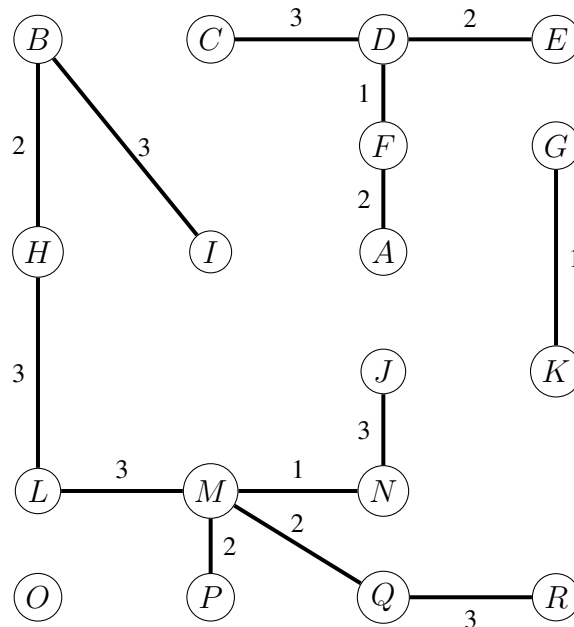
Figura 8.19 Inclusión de aristas con peso 1



Ahora agreguemos las aristas de peso 2 que no cierren ciclos, como se muestra en la figura 8.20 de la siguiente página.

Figura 8.20 Inclusión de aristas con peso 2

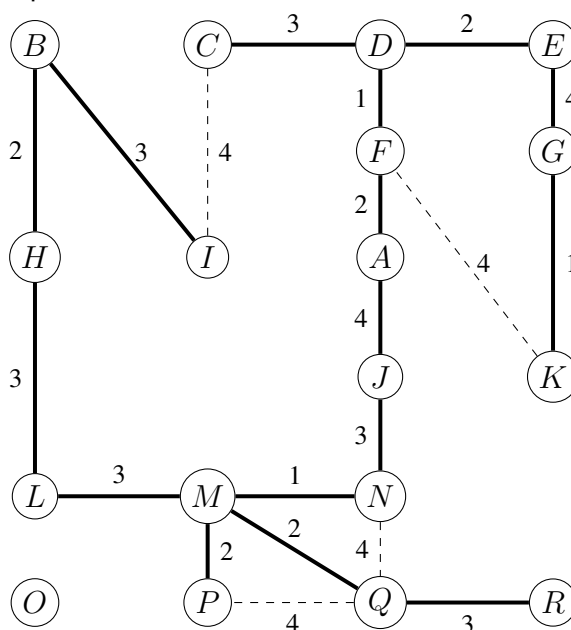
Agregamos ahora las aristas de peso 3 que no cierren ningún ciclo, como se muestra en la figura 8.21. Los subconjuntos están formados por cada una de las componentes conexas.

Figura 8.21 Inclusión de aristas con peso 3

Agregamos ahora aquellas aristas de peso 4 que no cierren ciclos. Marcaremos con línea

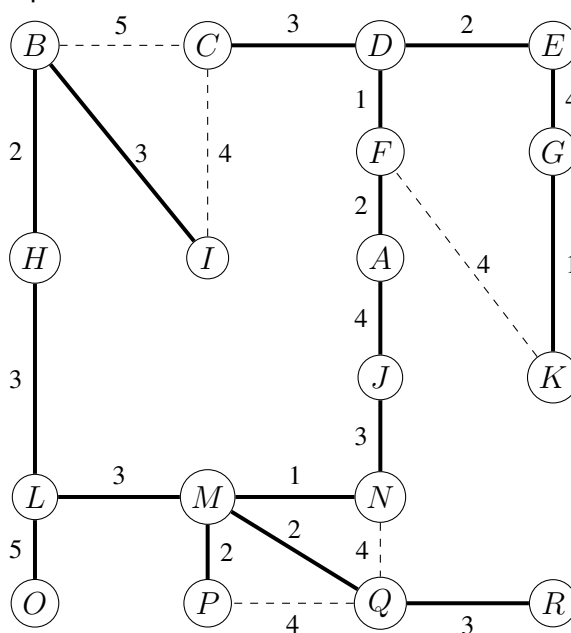
punteada las que no pueden ser agregadas porque cierran un ciclo. El resultado se puede ver en la figura 8.22.

Figura 8.22 Inclusión de aristas con peso 4



Agreguemos ahora las aristas de peso 5, sin incluir aquellas que cierran ciclos, como se muestra en la figura 8.23.

Figura 8.23 Inclusión de aristas con peso 5



La última arista que incluimos es OL , pues al agregarla se completa el número de aristas necesarias para el árbol. El resto de las aristas ni siquiera se revisan. El peso total del árbol es de 44 unidades, exactamente el mismo peso que el obtenido con el algoritmo de Prim.

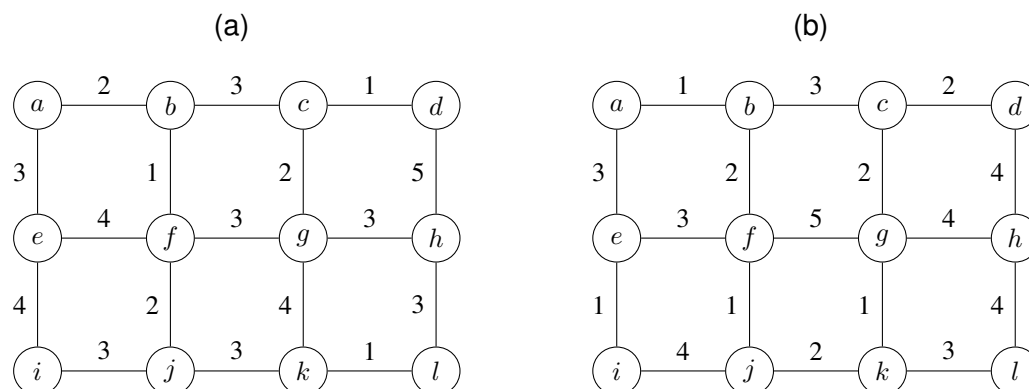
Aunque pudiera suponerse que el árbol generador producido por el algoritmo de Kruskal fuese, si no idéntico, casi idéntico al obtenido por el algoritmo de Prim, esto no es forzosamente cierto, sobre todo si existen más de un árbol generador de peso mínimo para una gráfica. Si no son el mismo será porque se sustituyen aristas por otras de exactamente el mismo peso. Por ejemplo, mientras que con el algoritmo de Prim teníamos el subcamino $F - K - G$ con peso total de 5 unidades, en el árbol construido por el algoritmo de Kruskal tenemos a G colgando de E y a K colgando de G , lo que da un costo también de 5 unidades para incluir a G y K al árbol.

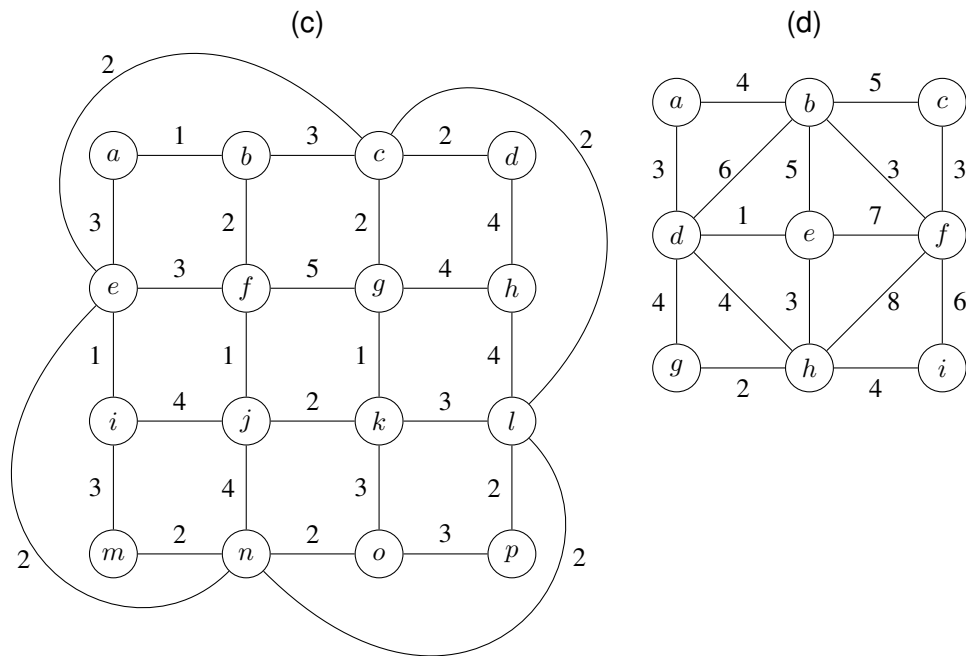
Deseamos insistir en que hemos utilizado hasta ahora aristas con pesos positivos. En especial el algoritmo de Dijkstra no puede utilizarse si hay pesos negativos, porque no podríamos garantizar que una vez que un vértice está al frente de la cola de prioridades es porque ya se le alcanzó por el camino más corto. Las aristas con peso negativo pudiesen encontrar un camino más corto aun después de haberse terminado de procesar al vértice. Los algoritmos de Prim y Kruskal no se ven afectados por esta situación.

Ejercicios

8.4.1.- ¿Puede el algoritmo de Kruskal producir más de un árbol generador distinto para una misma gráfica? Si es así, ¿cuándo se podría dar esta situación?

8.4.2.- Para las gráficas que siguen construye los árboles generadores bajo el algoritmo de Prim y después bajo el algoritmo de Kruskal. En el caso del algoritmo de Prim, cada vez que haya más de una opción para elegir al siguiente vértice que será el centro de acción, deberás elegir en orden lexicográfico.





- 8.4.3.- Construye una gráfica simple con pesos en las aristas con el menor número posible de aristas que tenga al menos dos árboles generadores de peso mínimo distintos.
- 8.4.4.- Un bosque generador de peso mínimo es un conjunto de árboles, donde cada árbol es un árbol generador de peso mínimo para la componente conexa de la gráfica. Habrá tantas componentes conexas como árboles generadores de peso mínimo. Explica cómo modificarías los algoritmos de Prim y Kruskal para que pudieran calcular el bosque generador de peso mínimo de una gráfica cualquiera, con posiblemente más de una componente conexa.
- 8.4.5.- Argumenta por qué una arista que tiene el peso mínimo de entre los pesos presentes en la gráfica debe estar presente en el árbol generador de peso mínimo.
- 8.4.6.- Supongamos que deseamos un árbol generador de peso mínimo para una gráfica, pero hay ciertas aristas que, independientemente de su peso, tienen que estar presentes en el árbol. ¿Cómo sugieres que esto se pueda hacer?

Multigráficas y gráficas dirigidas | 9

Hasta ahora hemos trabajado exclusivamente con gráficas donde entre dos vértices a lo más hay una arista – excepto cuando revisamos la formalización dada por Euler para el problema de los Puentes de Königsberg – y no hemos admitido aristas de la forma uu , conocidas como *lazos*¹. Tampoco hemos trabajado con gráficas dirigidas (llamadas *digráficas*). En esta sección abordaremos ambos temas.

9.1. Multigráficas

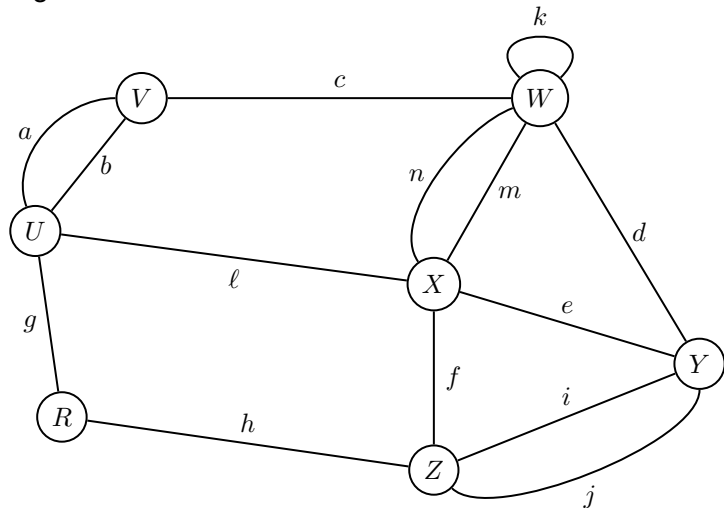
Una multigráfica es, como ya mencionamos, una gráfica que admite varias aristas entre dos vértices – llamadas *aristas paralelas* – y lazos. En realidad una gráfica es un caso particular de una multigráfica, por lo que todos los conceptos que vimos en las gráficas se aplican a multigráficas. La diferencia fundamental es que los extremos de una arista ya no determinan unívocamente a la arista. Por ello se usa identificar a las aristas de una multigráfica asociando nombres para cada arista. Veamos una multigráfica en la figura 9.1.

En el caso de multigráficas las trayectorias se especifican usando los nombres de las aristas, no las parejas de vértices, ya que esto último lleva a confundir de cuál arista se trata. Podemos observar, por ejemplo, que la gráfica del problema de los Puentes de Königsberg es una multigráfica. Por lo tanto encontrar un ciclo euleriano o un paseo euleriano se aplica también a multigráficas. o de componente conexa de una gráfica, que se interpreta de

¹En inglés *loop*.

manera directa para una multigráfica.

Figura 9.1 Ejemplo de multigráfica



La representación de multigráficas en matrices de adyacencias no es posible, pues no tendríamos la noción de cuál de las aristas es la que está en juego. Lo mismo podemos decir de las listas de adyacencias. En cambio, las matrices o listas de incidencias son una representación adecuada que preserva la distinción entre las aristas múltiples.

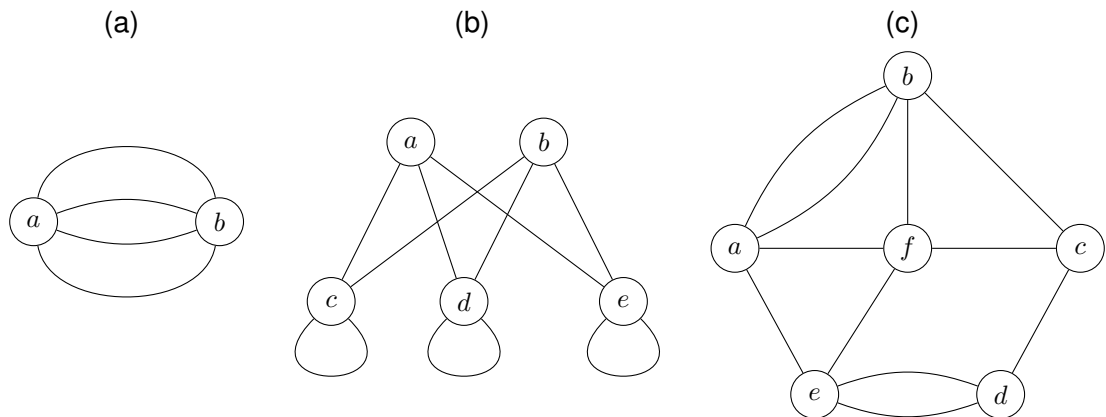
Ejemplo 9.1. Veamos la matriz de incidencias y las listas de incidencias para la gráfica en la figura 9.1.

	Matriz de incidencias														Listas de incidencias	
	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>	<i>m</i>	<i>n</i>		
<i>R</i>	0	0	0	0	0	0	1	1	0	0	0	0	0	0	$R \rightarrow g \rightarrow h$	
<i>U</i>	1	1	0	0	0	0	0	0	0	0	0	1	0	0	$U \rightarrow a \rightarrow b \rightarrow l$	
<i>V</i>	1	1	1	0	0	0	0	0	0	0	0	0	0	0	$V \rightarrow a \rightarrow b \rightarrow c$	
<i>W</i>	0	0	1	1	0	0	0	0	0	0	1	0	1	1	$W \rightarrow c \rightarrow d \rightarrow k \rightarrow m \rightarrow n$	
<i>X</i>	0	0	0	0	1	1	0	0	0	0	0	1	1	1	$X \rightarrow e \rightarrow f \rightarrow l \rightarrow m \rightarrow n$	
<i>Y</i>	0	0	0	1	1	0	0	0	1	1	0	0	0	0	$Y \rightarrow d \rightarrow e \rightarrow i \rightarrow j$	
<i>Z</i>	0	0	0	0	0	1	0	1	1	1	0	0	0	0	$Z \rightarrow f \rightarrow h \rightarrow i \rightarrow j$	

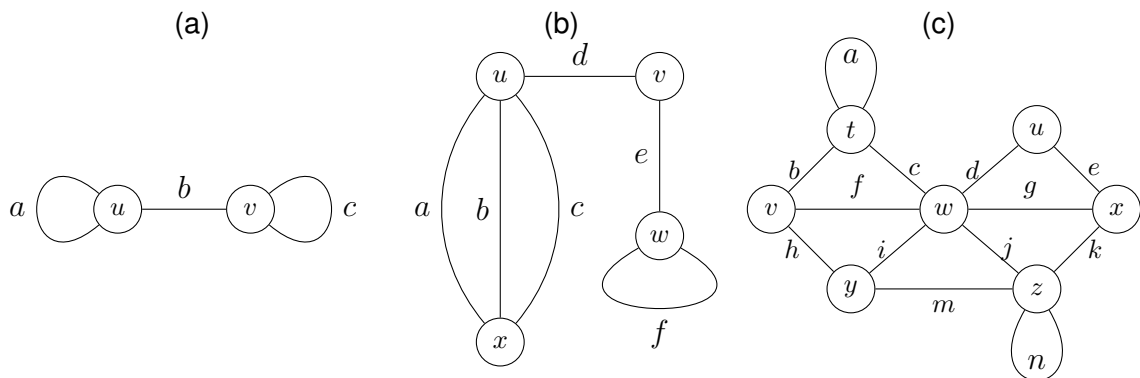
Ejercicios

9.1.1.- Demuestra que una multigráfica siempre tiene una subgráfica simple que preserva adyacencias.

9.1.2.- En las siguientes gráficas determina si la gráfica es o no una multigráfica. Justifica tu respuesta.



9.1.3.- Lista los lazos y las aristas múltiples en las siguientes multigráficas.



9.1.4.- Dibuja ejemplos de multigráficas que satisfagan cada una de las siguientes condiciones:

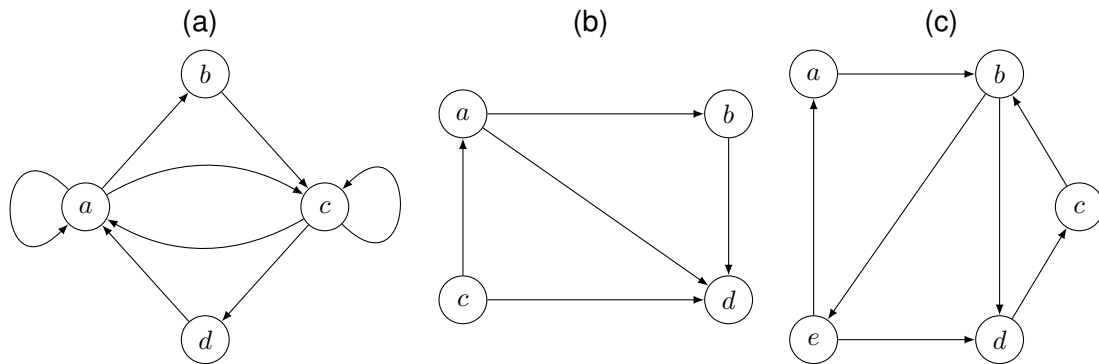
- (a) Hay exactamente dos ciclos.
- (b) Hay un ciclo de tamaño 1.
- (c) Hay un ciclo de tamaño 2.

9.2. Gráficas dirigidas

Como mencionamos al principio de este capítulo, una gráfica dirigida o digráfica $D = (V, A)$ es una pareja compuesta por un conjunto de vértices, como en el caso de las gráficas, y un conjunto de *arcos* que consisten de parejas *ordenadas* de vértices. Recordamos que a las parejas ordenadas de vértices se les conoce como *arcos* para subrayar el hecho de que el orden de los vértices en la pareja importan: $(u, v) \neq (v, u)$. En el caso de las digráficas, podemos tener también multigráficas dirigidas, aunque el concepto de arco múltiple entre dos vértices se aplica únicamente a la presencia más de una vez de la misma pareja de vértices ordenados. Es frecuente la presencia de lazos en el caso de gráficas dirigidas.

Ejemplo 9.2. En la figura 9.2 vemos la ilustración de algunas digráficas.

Figura 9.2 Ejemplos de digráficas



En la digráfica de la figura 9.2(a) podemos observar que aunque hay dos arcos entre a y c , uno corresponde a la pareja ordenada (a, c) mientras que el otro corresponde a (c, a) , por lo que no hay posibilidad de confundirlos. En la figura 9.2(b) podemos observar que no hay manera de llegar del vértice a al vértice c , siguiendo la dirección de los arcos. En cambio, en la 9.2(c) se puede llegar desde cualquier vértice a cualquier otro siguiendo la dirección de los arcos.

En el caso de digráficas, al definir una trayectoria deberemos exigir que la dirección de los arcos sea la correcta:

Definición 9.1 (trayectoria dirigida) Una *trayectoria dirigida* en una digráfica es una sucesión de arcos a_1, a_2, \dots, a_k tal que $a_i = (u, v)$, $a_{i-1} = (x, u)$ y $a_{i+1} = (v, y)$, $2 \leq i \leq k - 1$.

Ejemplo 9.3. En la digráfica de la figura 9.2(a) la sucesión $(a, c), (c, d), (d, a), (a, b), (b, c), (c, a)$ forman una trayectoria dirigida; pero la sucesión $(c, a), (c, d), (d, a), (a, b), (b, c), (c, a)$ no corresponde a una trayectoria dirigida, pues no coinciden el segundo elemento de la primera pareja con el primer elemento de la segunda pareja.

En ocasiones es conveniente observar una digráfica sin tomar en cuenta la dirección de sus arcos. Tenemos entonces el concepto de *gráfica subyacente*:

Definición 9.2 (gráfica subyacente) La *gráfica subyacente* de una digráfica D es la que resulta de quitar la dirección a los arcos de la digráfica; esto es, considerar a las parejas de vértices que representan a los arcos como parejas no ordenadas.

Si consideramos a las subgráficas subyacentes, tenemos la siguiente definición:

Definición 9.3 (trayectoria no dirigida) Una *trayectoria no dirigida* en una digráfica D es aquella que es una trayectoria en la gráfica subyacente.

Otro tipo de caminos que podemos identificar son los *caminos antidirigidos*:

Definición 9.4 (trayectorias antidirigidas) Una *trayectoria antidirigida* es una sucesión de arcos o flechas que alternan direcciones.

Veamos dos definiciones más relacionadas con digráficas:

Definición 9.5 (exgrado) El *exgrado* de un vértice $v \in V$ – denotado por $d^+(v)$ – corresponde al número de arcos que salen de v , aquéllos en los que el primer componente es v .

Definición 9.6 (ingrado) El *ingrado* de un vértice $v \in V$ – denotado por $d^-(v)$ – corresponde al número de arcos que llegan o entran a v , aquéllos cuya segunda componente es v .

Ejemplo 9.4. Veamos los ingrados y exgrados de las gráficas de la figura 9.2.

gráfica 9.2(a)		
v	d^+	d^-
a	3	3
b	1	1
c	3	3
d	1	1

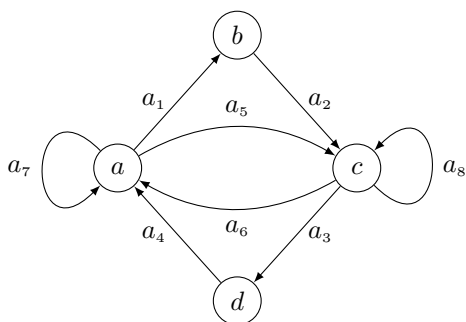
gráfica 9.2(b)		
v	d^+	d^-
a	2	1
b	1	1
c	2	0
d	0	3

gráfica 9.2(c)		
v	d^+	d^-
a	1	1
b	2	2
c	1	1
d	1	2
e	2	1

En cuanto a la representación de las gráficas dirigidas lo haremos de la misma manera que lo hicimos con gráficas en general, siempre y cuando no se repita ninguna pareja ordenada de vértices. Cabe mencionar que en este caso las matrices de adyacencias no van a ser simétricas, ya que el arco (u, v) se registra únicamente en el renglón u , columna v . Por otro lado, las matrices de incidencias no nos van a dar información respecto al ingrado o exgrado de los vértices, ya que si, para un vértice dado, registramos si es extremo de un arco, de la matriz, y sin ver a la pareja que conforma al arco, no sabemos si el arco sale o llega al vértice. Tampoco tenemos realmente forma de representar a un lazo en las matrices de incidencias. Una opción es la de anotar con 1 los vértices a los que llega esa arista y con -1 los vértices de los que sale, pero esta representación tampoco nos va a dar oportunidad de representar lazos.

Para el caso de las listas de adyacencias seguiremos la regla de colocar en la lista de un vértice dado sólo aquellos vértices a los que se puede llegar desde ese vértice recorriendo un único arco; similarmente, para las listas de incidencias únicamente registraremos en la lista de un vértice a aquellos arcos que salen de ese vértice.

Ejemplo 9.5. Veamos las distintas representaciones para la digráfica de la figura 9.2(a), a la que le agregamos nombres a los arcos:



Matriz de adyacencias

$$\begin{matrix} & a & b & c & d \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{matrix}$$

Listas de adyacencias

$a \longrightarrow a \longrightarrow b \longrightarrow d$
 $b \longrightarrow c$
 $c \longrightarrow c \longrightarrow a \longrightarrow d$
 $d \longrightarrow a$

Matriz de incidencias

$$\begin{matrix} & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{matrix}$$

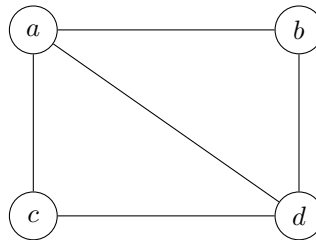
Listas de incidencias

$a \longrightarrow a_1 \longrightarrow a_5 \longrightarrow a_7$
 $b \longrightarrow a_2$
 $c \longrightarrow a_3 \longrightarrow a_6 \longrightarrow a_8$
 $d \longrightarrow a_4$

El concepto de conexidad en gráficas se traslada a digráficas de manera un poco distinta. Por un lado tenemos el concepto de *digráfica fuertemente conexa*, donde se exige que haya

un camino dirigido entre cualesquiera dos vértices. Aunque en ocasiones esta exigencia resulta demasiado fuerte, es común que se desee una digráfica con estas características para garantizar el acceso de cualquier vértice desde cualquier otro. El concepto de *digráfica (débilmente) conexa* es un poco más relajado e involucra a la gráfica subyacente. Entonces, decimos que una digráfica es *conexa* si su gráfica subyacente es conexa.

Ejemplo 9.6. Si bien en la digráfica de la figure 9.2(b) no hay trayectorias dirigidas entre cualesquiera dos vértices de la digráfica – por ejemplo, no se puede llegar desde el vértice d al vértice a – la gráfica subyacente es conexa, como se puede observar en la figura.



Pasemos a ver algunas de las propiedades que vimos en gráficas simples y que, de alguna manera, se adaptan a gráficas dirigidas.

Lema 9.1 En una gráfica dirigida $D = (V, A)$, las siguientes cantidades son iguales:

$$\sum_{v \in V} d^+(v) = \sum_{v \in V} d^-(v) = |A|$$

Demostración.

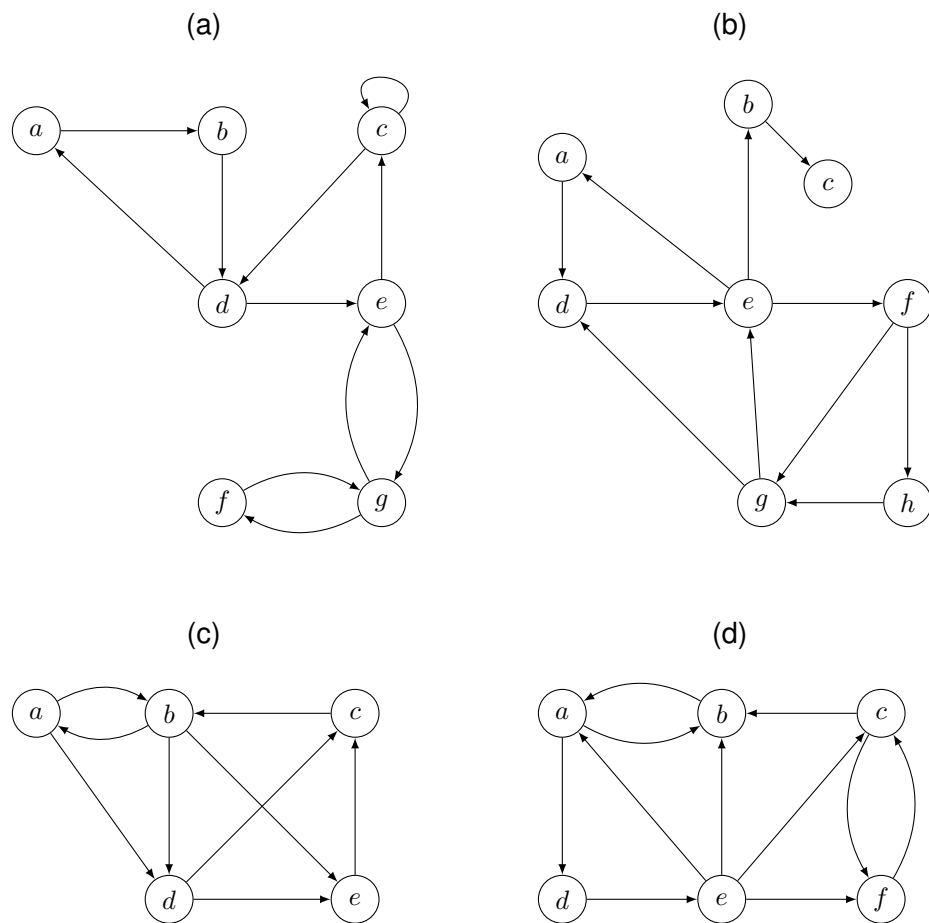
Claramente por cada unidad en el ingrado de un vértice que corresponde a un arco que llega al vértice, este arco proviene de algún (otro) vértice, donde corresponde, a su vez, a una unidad en el exgrado de ese vértice. Por lo que por cada unidad en el ingrado, existe una unidad en el exgrado y la primera igualdad se cumple. Ahora, el número de arcos los podemos contar ya sea cuando salen de un vértice (el exgrado) o cuando llegan a un vértice, que corresponde al ingrado. Por lo tanto, el número total de arcos corresponde a la suma de los ingrados o bien a la suma de los exgrados. □

Ejercicios

9.2.1.- Para las siguientes matrices de adyacencias, dibuja las gráficas dirigidas correspondientes.

$$\begin{array}{ccc}
 \text{(d)} & \text{(e)} & \text{(f)} \\
 \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}
 \end{array}$$

9.2.2.- Para las siguientes gráficas dirigidas, construye las matrices de adyacencias y las listas de adyacencias e incidencias correspondientes.

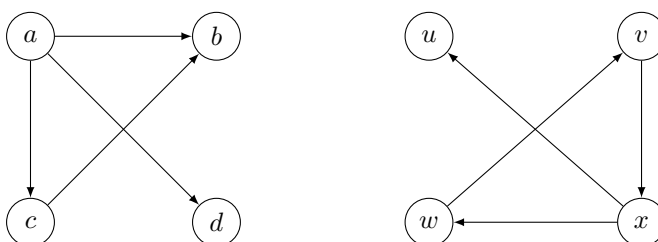


9.2.3.- ¿Hay alguna característica que determine, *a priori*, que una gráfica dirigida no tenga caminos desde un vértice origen a cualquiera de los vértices de la gráfica?

9.2.4.- Da un algoritmo para determinar si desde cualquier vértice se puede llegar a un vértice dado en una gráfica dirigida.

9.2.5.- Escribe una definición de lo que es un isomorfismo entre gráficas dirigidas.

9.2.6.- Determina si las gráficas dirigidas a continuación son o no isomorfas.



9.3. Circuitos eulerianos

Para el caso de digráficas, se definen condiciones un poco distintas para que haya un circuito euleriano dirigido (en adelante abreviamos a circuito euleriano simplemente); básicamente, se exige que a cada vértice al que se llega mediante un arco dirigido hacia él, haya un arco que salga de él para poder abandonarlo. Esta condición queda clara en el teorema 9.1.

Teorema 9.1 Una digráfica $D = (V, A)$ cuya gráfica subyacente es conexa, tiene un circuito euleriano dirigido si y sólo si para cada $v \in V$, $d^+(v) = d^-(v)$.

Demostración.

\Rightarrow Supongamos que $D = (V, A)$ tiene un circuito euleriano. Mantengamos la cuenta de $d^+(v)$ y $d^-(v)$, $\forall v \in V$. Sea v_0 el vértice en el que iniciamos; sumamos 1 a $d^+(v_0)$. A partir de ese momento, a cada vértice al que llegamos, salimos de él, de donde por cada arco que llega tenemos un arco que sale del vértice, por lo que incrementamos en 1 tanto a $d^+(v)$ como a $d^-(v)$. Pero para el último vértice en el circuito únicamente tenemos el arco que llega. Pero ese vértice es v_0 , el mismo desde el cual iniciamos, por lo que ese último arco se aparea con el que se usó para empezar e incrementa a $d^-(v)$ en 1.

$$\therefore \forall v \in V, d^+(v) = d^-(v)$$

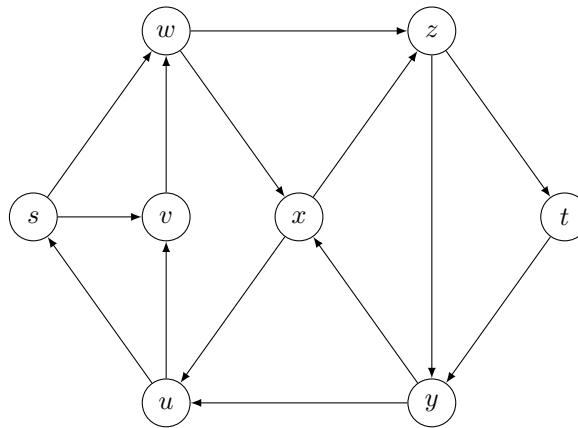
\Leftarrow Supongamos ahora que $\forall v \in V$, $d^+(v) = d^-(v)$. Seguimos el algoritmo dado para gráficas no dirigidas, pero en cada vértice elegimos algún arco que salga y que no haya sido todavía utilizado; como en el caso de digráficas $\sum_{v \in V} d^+(v) = \sum_{v \in V} d^-(v)$, cada uno de estos arcos, llega a otro vértice. Como tenemos para cada vértice que $d^-(v) = d^+(v)$, cada vez que llegamos a un vértice, tenemos uno libre por el cual salir. Por lo que si seguimos el algoritmo 6.1, pero eligiendo para salir un arco que tenga como primer

componente al vértice del que deseo salir, construiremos un circuito euleriano. □

En cuanto a los paseos eulerianos dirigidos (en adelante, paseos eulerianos simplemente), también podemos pensar en condiciones similares. Mientras que en gráficas no dirigidas se exigía que hubiese exactamente dos vértices de grado impar y el resto de grado par, en el caso de digráficas debemos pedir que exactamente un vértice tenga un arco más de salida que los que tiene de entrada, mientras que exactamente un vértice tenga un arco más de entrada que los que tiene de salida. El resto de los vértices tendrá que cumplir la condición de que su exgrado sea igual a su ingrado. El paseo euleriano debe iniciar en el vértice de exgrado mayor al ingrado y debe terminar en el vértice con ingrado mayor a su exgrado.

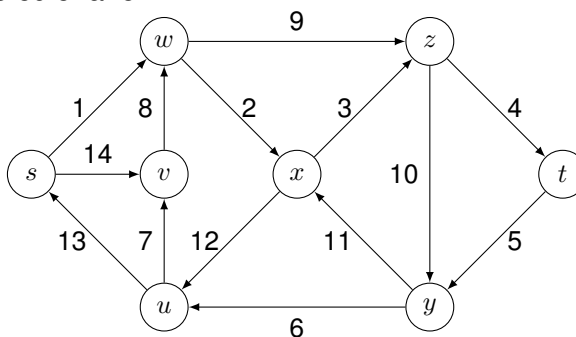
Ejemplo 9.7. Dada la gráfica dirigida de la figura 9.3, decidir si la digráfica tiene o no un paseo euleriano, y si lo tiene, encontrarlo.

Figura 9.3 Digráfica para paseo dirigido euleriano



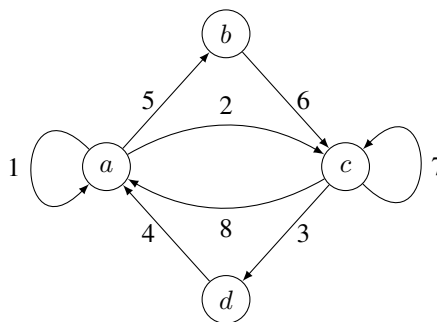
Respuesta: La digráfica tiene una trayectoria dirigida euleriana porque $d^+(v) = d^-(v)$, $\forall v \in V$, excepto para s que tiene $d^-(s) = 1$ mientras que $d^+(s) = 2$, lo que indica que se debe empezar en ese vértice. Para el vértice donde se debe terminar tenemos $d^-(v) = 2$ y $d^+(v) = 1$; de donde los vértices cubren las condiciones necesarias y suficientes para que la gráfica tenga un paseo euleriano.

En la gráfica de la figura 9.4 anotamos los arcos con el orden en que pueden recorrerse para obtener un paseo euleriano.

Figura 9.4 Digráfica con el paseo euleriano

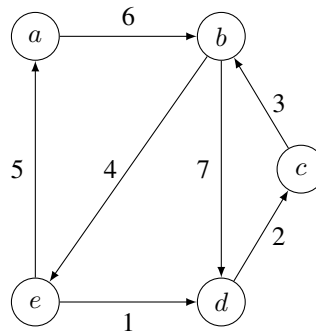
Ejemplo 9.8. Examinemos las digráficas que presentamos en la figura 9.2 para ver si tienen o no circuito o paseo euleriano.

- (a) En la digráfica de esta figura, la gráfica subyacente es conexa y nos encontramos con que para todos los vértices se cumple que su ingrado sea igual a su exgrado, por lo que tiene un circuito euleriano. A continuación mostramos la digráfica con las aristas anotadas con el orden en que son tomadas para el circuito euleriano.



Hay muchos otros posibles circuitos eulerianos en esta digráfica. Invitamos al lector a encontrarlos.

- (b) En esta digráfica tres de los cuatro vértices tienen distintos su ingrado y su exgrado, por lo que no hay posibilidad ni de circuito euleriano ni de paseo euleriano.
- (c) En esta digráfica vemos que se cumplen las condiciones para un paseo euleriano, empezando en el vértice e que es el que tiene el exgrado mayor a su ingrado, y terminando en el vértice d que es el que tiene el ingrado mayor que el exgrado. Un paseo euleriano posible se muestra a continuación.



Hay al menos otro paseo euleriano sobre esta misma digráfica. Invitamos al lector a encontrarlo.

Como se puede observar, la única diferencia entre los algoritmos que se aplican a gráficas no dirigidas o a digráficas, en el caso de paseos eulerianos, es que hay que cuidar de cuál de los dos vértices se sale; la representación más útil para manipular digráficas va a ser la de listas de adyacencias porque esta representación es la que nos va a dar más información. Sin embargo, si queremos saber, por ejemplo, el ingrado de un vértice, tendremos que contar las veces que ese vértice aparece en las distintas listas, lo que tiene un costo proporcional al número de arcos.

Ejercicios

9.3.1.- Para las digráficas del ejercicio 9.3.2, determina si tienen circuito euleriano, paseo euleriano o ninguno de los dos. Si es alguno de los primeros dos casos, encuentra un circuito euleriano o un paseo euleriano, según corresponda.

9.3.2.- Queremos modelar con una digráfica las preferencias de un profesor para impartir cursos en la carrera de Ciencias de la Computación. El profesor ha establecido su preferencia entre cada dos materias de cinco – Matemáticas discretas (MD), Análisis lógico (AL), Teoría de la computación (TC), Complejidad computacional (CC), Lenguajes de programación (LP) – que él puede impartir. Las preferencias son las siguientes:

Su materia favorita es TC.

Prefiere dar AL que cualquier otra materia excepto TC.

Prefiere dar CC que LP.

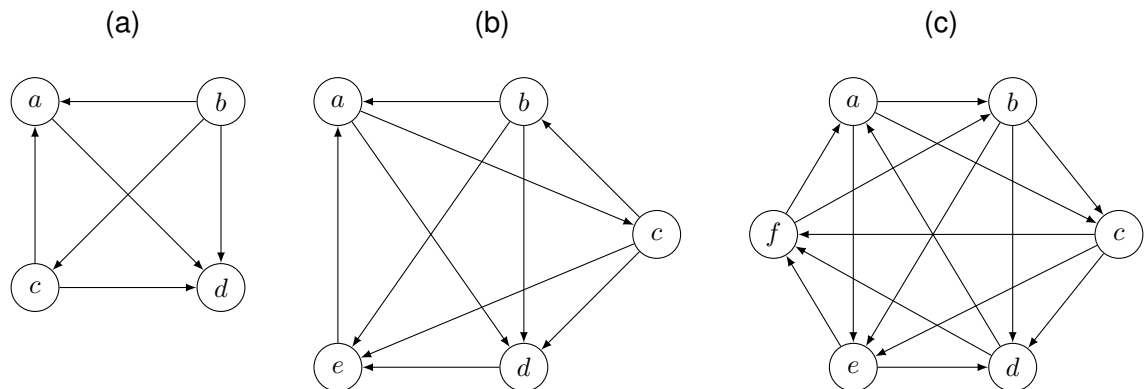
Prefiere dar MD que CC o LP.

Dibuja la gráfica dirigida que corresponde a estas preferencias. ¿Existe algún orden en estas materias que refleje estas preferencias? ¿Cuántos posibles ordenamientos

existen?

(Este problema se asemeja un poco al de trayectorias hamiltonianas, pues nos pide un camino que visite a todos los vértices exactamente una vez, siguiendo la dirección de los arcos, donde hay un arco del vértice u al vértice v si es que u debe ir antes que v en la lista ordenada.)

9.3.3.- Un *torneo* es una digráfica que se obtiene asignando dirección a cada arista de una gráfica completa no dirigida. Es decir, un torneo es una digráfica en la que cada pareja de vértices está conectado por un solo arco. En un torneo, el exgrado de un vértice nos indica el *marcador* (número de juegos que ganó). El *rey* del torneo es aquel vértice con marcador mayor. En los siguientes torneos encuentra al rey del torneo y muestra que hay una trayectoria dirigida de longitud 1 o 2 desde ese vértice a cualquier otro:



9.3.4.- ¿Puede un torneo tener dos equipos que siempre pierdan?

9.3.5.- En un torneo con 4 equipos, América, Pumas, Chivas y Cruz Azul, cada uno de los equipos juega contra el resto exactamente una vez. El resultado de los juegos es el siguiente:

- i. Los Pumas les ganaron al resto de los equipos.
- ii. Las Chivas perdieron contra el resto de los equipos.
- iii. El América les ganó a todos menos a los Pumas.

¿Puedes asignar lugares a cada uno de los equipos? ¿Cuántos posibles órdenes hay?

9.3.6.- Escribe el algoritmo para encontrar un circuito euleriano en una digráfica.

9.4. Distancias en una gráfica dirigida

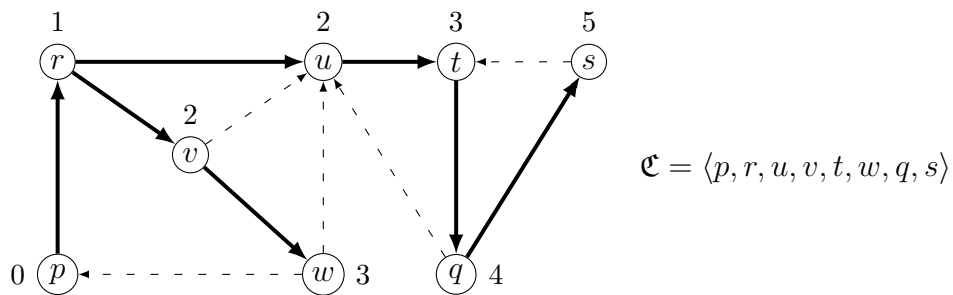
Adaptamos el concepto de distancia entre dos vértices u y v – $\delta(u, v)$ – en una digráfica a que sea la longitud de la trayectoria dirigida simple más corta que va de u a v , o bien $\delta(u, v) = \infty$ si tal trayectoria dirigida no existe. Nuevamente, el orden de los vértices es importante, pues en una digráfica puede haber una trayectoria dirigida de u a v y que no haya una de v a u .

9.4.1. BFS

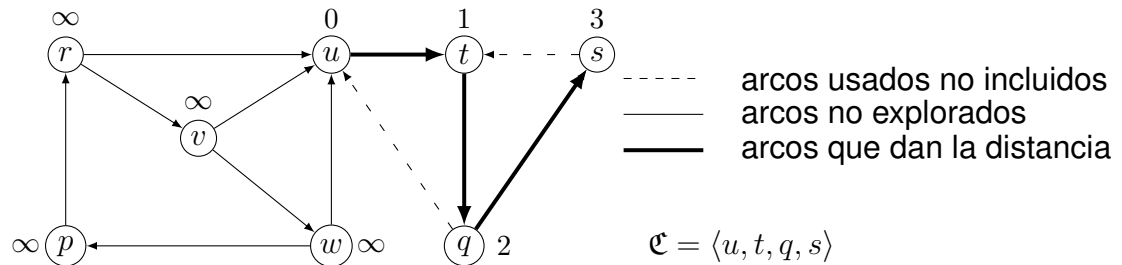
Para encontrar $\delta(u, v)$ para todo vértice v en V , usamos el algoritmo BFS, excepto que las listas de adyacencias respetan la dirección de los arcos – ver el algoritmo BFS en la página 270.

Similarmente, si deseamos encontrar la distancia entre cualquier pareja de vértices o entre dos vértices (u, v) , usamos el algoritmo BFS con las listas de adyacencias respetando la dirección de los arcos. Debemos tener claro que aunque la gráfica subyacente sea conexa, pudiese suceder que no haya trayectoria entre dos vértices dados, ya que ésta depende del vértice origen y que los arcos se encuentren en la dirección correcta para seguir una trayectoria. Veamos, por ejemplo, la misma gráfica que se usó para ejemplificar BFS en la figura 9.5, pero con dirección definida para los arcos.

Figura 9.5 Exploración BFS de una gráfica dirigida desde el vértice p



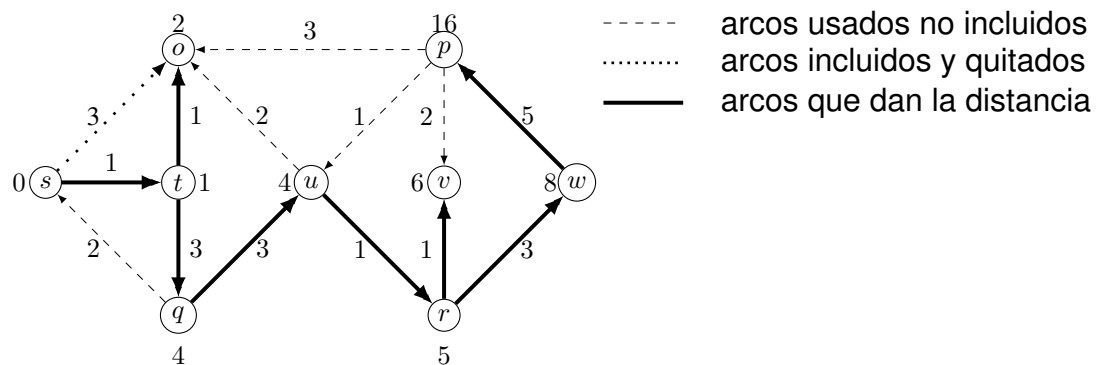
Si la distancia se mide desde el vértice p , entonces todos los vértices quedan con la distancia definida ($< \infty$). En cambio, si intentamos medir la distancia desde el vértice u nos vamos a encontrar que únicamente la parte derecha de la gráfica es alcanzable, mientras que la parte izquierda no; esto sucede a pesar de que la gráfica subyacente es conexa – ver figura 9.6.

Figura 9.6 Exploración BFS de una gráfica dirigida desde el vértice u 

9.4.2. Algoritmo de Dijkstra para trayectorias dirigidas más cortas

También en el caso del algoritmo de Dijkstra para trayectorias dirigidas más cortas, todo sucede igual que en el caso de gráficas no dirigidas, excepto que los arcos se toman con la dirección adecuada – ver página 276 – donde en cada vértice se pregunta por los arcos no usados que salen de ese vértice.

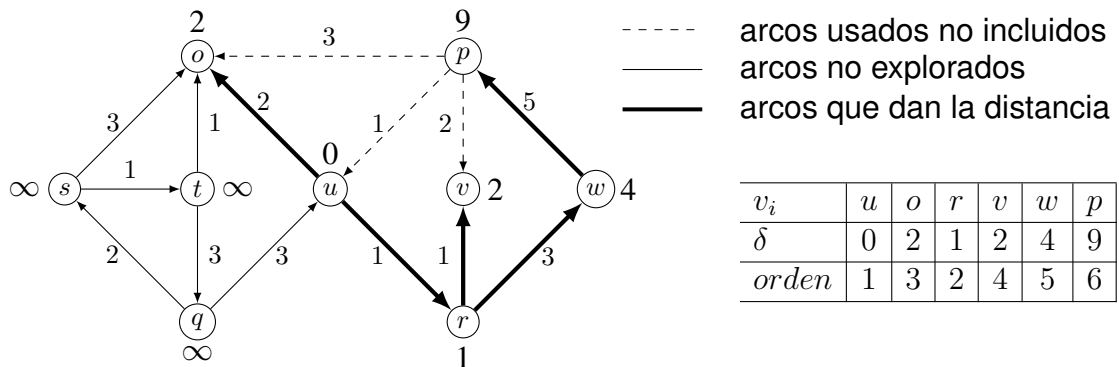
En el caso de este algoritmo, cuando se utilizan arcos en lugar de aristas las cosas pueden cambiar mucho, pues aunque la gráfica subyacente sea conexa, en ocasiones no habrá trayectoria dirigida entre dos vértices. Veamos la misma gráfica de la figura 6.14 pero con dirección en las aristas, para ver cómo se definen las distancias, en la gráfica de la figura 9.7.

Figura 9.7 Algoritmo de Dijkstra para trayectorias dirigidas más cortas con origen en s 

v_i	s	o	t	q	u	r	v	w	p
δ	0	16	1	2	4	5	6	8	16
$orden$	1	3	2	4	5	6	7	8	9

Sin embargo, si el vértice origen es, por ejemplo, u , no todos los vértices van a tener una distancia definida, como se puede ver en la figura 9.8.

Figura 9.8 Algoritmo de Dijkstra para trayectorias dirigidas más cortas con origen en u



9.4.3. Número de caminos

Nuevamente utilizamos el método de multiplicar las matrices de adyacencias, excepto que estas matrices deben reflejar la dirección de los arcos. En A^k encontraremos el número de caminos dirigidos del vértice u al vértice v de tamaño k .

Veamos lo que pasa con la matriz de adyacencias de la gráfica en la figura 9.8 sin pesos en las aristas. Obtendremos únicamente los caminos de longitud 2, sólo para ilustrar.

$$\begin{matrix} & A & & A & & A^2 \\ & o \ p \ q \ r \ s \ t \ u \ v \ w & & o \ p \ q \ r \ s \ t \ u \ v \ w & & o \ p \ q \ r \ s \ t \ u \ v \ w \\ \begin{matrix} o \\ p \\ q \\ r \\ s \\ t \\ u \\ v \\ w \end{matrix} & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} & \cdot & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} & = & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}
 \end{matrix}$$

En la matriz correspondiente a A^2 , tenemos que hay dos caminos del vértice q al vértice o . En efecto, en la gráfica de la figura 9.8 podemos observar que esos dos caminos son $q \rightarrow u \rightarrow o$ y $q \rightarrow s \rightarrow o$. El camino de longitud 2 que va del vértice t al vértice s es $t \rightarrow q \rightarrow s$. El resto de los caminos los puede verificar el lector de así desearlo.

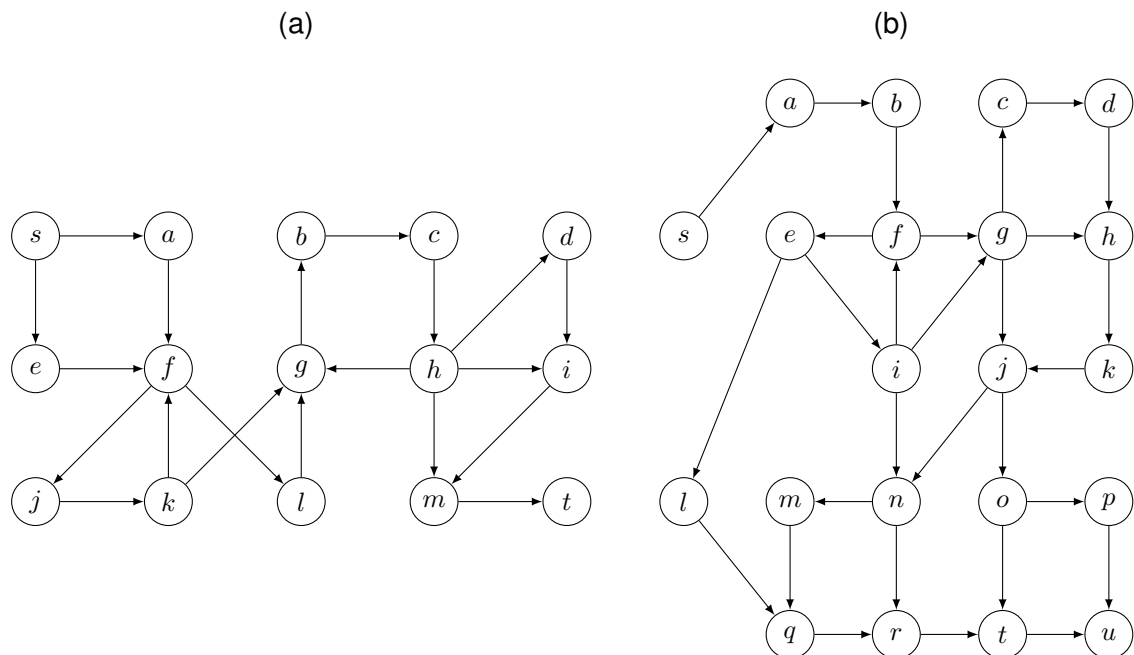
9.4.4. Árboles

En el caso de árboles dirigidos los cambios que se presentan son más significativos que para el resto de los algoritmos que vimos. Por falta de espacio y tiempo hemos decidido no revisarlo en esta ocasión.

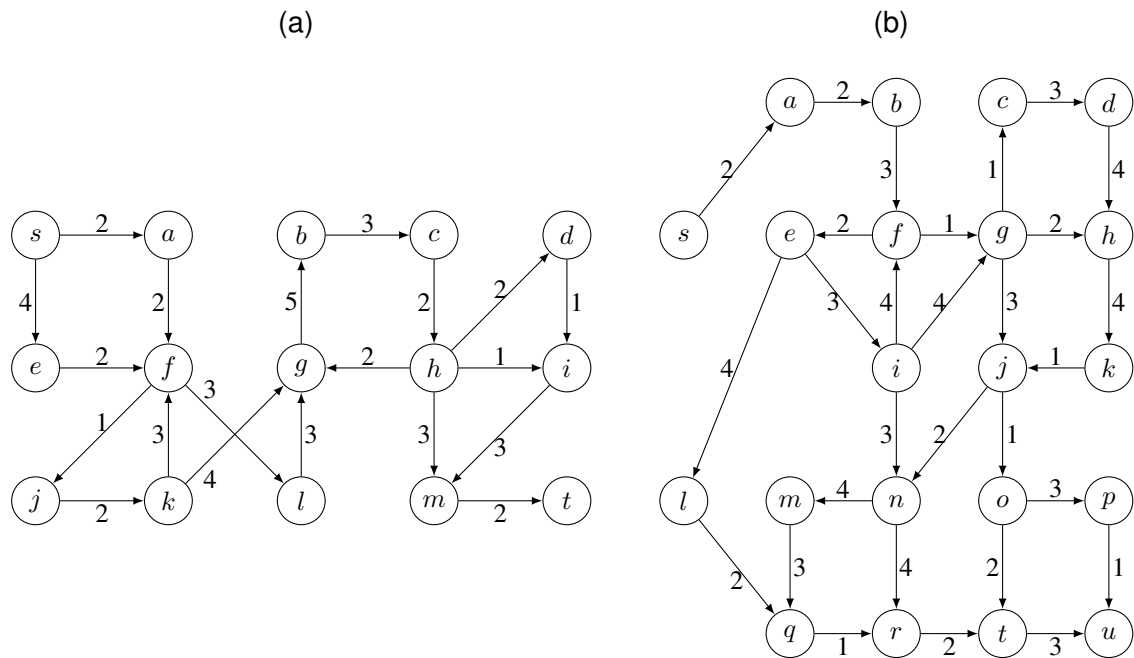
Ejercicios

9.4.1.- Escribe la versión del algoritmo BFS para encontrar las distancias en una gráfica dirigida.

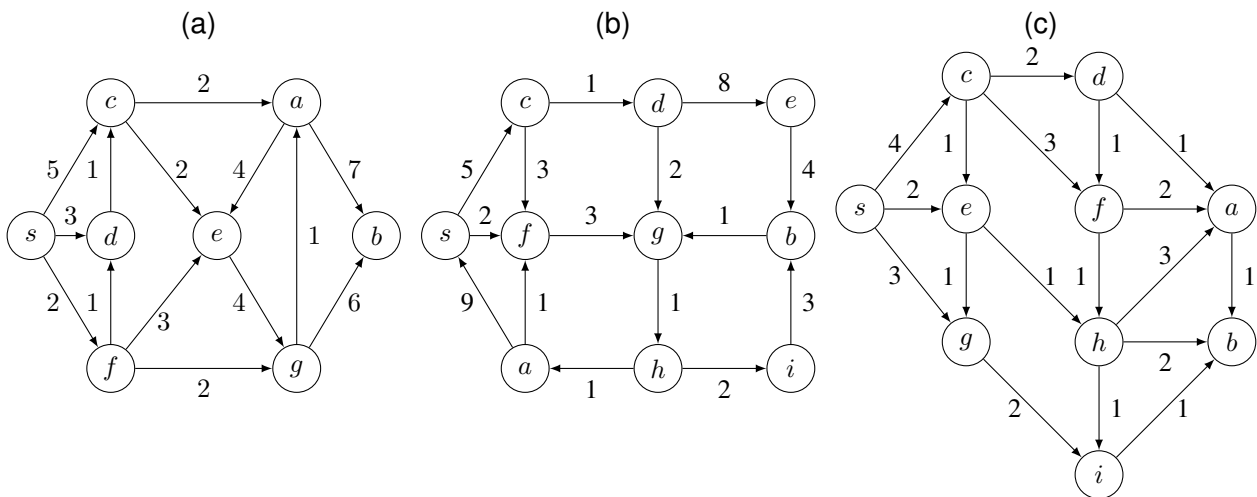
9.4.2.- Usa el algoritmo BFS para encontrar el árbol de distancias de las siguientes digráficas.



9.4.3.- Encuentra los árboles de distancias para las gráficas dirigidas con pesos en los arcos que se encuentran a continuación.



9.4.4.- Encuentra la trayectoria dirigida más corta del vértice s al vértice a en las siguientes digráficas utilizando el algoritmo de Dijkstra.



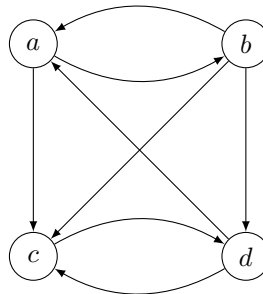
9.4.5.- Explica por qué para una digráfica \mathcal{D} donde $V = \{v_1, v_2, \dots, v_n\}$ con su matriz de adyacencias A , el número de caminos dirigidos de tamaño m desde v_i hasta v_j está dado por la entrada $[i, j]$ de A^m .

9.4.6.- Da una definición de digráficas isomorfas con pesos en sus arcos. Da un ejemplo de

dos digráficas con pesos en sus arcos que **no** sean isomorfas, aun cuando sus gráficas subyacentes sí lo son.

9.4.7.- Muestra que todo torneo tiene una trayectoria dirigida hamiltoniana.

9.4.8.- Para la siguiente gráfica determina el número de caminos de tamaño 1, 2, 3 y 4 desde a hasta d y desde d hasta a .



Bibliografía

- [1] K. Doets and J. van Eijck. *The Haskell Road to Logic, Maths and Programming*. King's Coll. Pub., London, 2004.
- [2] John A. Dossey, Albert D. Otto, Lawrence E. Spence, and Charles Vanden Eynden. *Discrete Mathematics*. Pearson/Addison-Wesley, 5-th edition, 2006.
- [3] Judith L. Gersting. *Mathematical Structures for Computer Science*. Computer Science Press, W.H. Freeman and Company, third edition, 1993.
- [4] Winifried Karl Grassman and Jean-Paul Tremblay. *LOGIC AND DISCRETE MATHEMATICS, A Computer Science Perspective*. Prentice-Hall Inc., 1996.
- [5] David Gries and Fred B. Schneider. *A Logical Approach to Discrete Mathematics*. Springer-Verlag, 1994.
- [6] Jerold W. Grossman. *DISCRETE MATHEMATICS, an introduction to concepts, methods and applications*. Macmillan Publishing Company, 1990.
- [7] Thomas Koshy. *Discrete Mathematics with Applications*. Elsevier Academic Press, 2004.
- [8] K.H. Rossen. *Discrete Mathematics and its Applications*. McGraw Hill, 6-th edition, 2006.

Índice

- ::=, 10
- acoplamiento perfecto, 210
- adyacencia
 - en vértices, 217
 - preservación, 240
- alcance, 54, 122
- alcanzable
 - vertice, 270
- algoritmo, 202
 - BFS, 269
 - circuito euleriano, 250
 - de Dijkstra, 276
 - en digráficas, 367
 - de Floyd, 292
 - de Prim, 329
 - de Warshall, 290
 - DFS, 322
 - glotón o ávido, 276
 - para clasificar fórmulas, 109
 - para consecuencia lógica, 110
 - para ruta crítica, 205
 - para satisfacibilidad, 109
 - para tautologías, 108
- análisis sintáctico, 50
- apareamiento perfecto, 210
- árbol, 12, 247, 312, 315
 - binario, 180, 197
 - de distancias, 340, 369
 - de peso mínimo, 339
 - generador, 318
 - de peso mínimo, 328
 - por BFS, 320
 - por DFS, 322, 326
 - por Dijkstra, 328
 - por Prim, 330
 - resultado, 12
- archivo
 - directorio, 158
- arco, 203, 216, 356
- argumento
 - correcto, 37, 77, 155
 - deductivo, 37
 - inductivo, 37
 - lógico, 19, 37
- aridad, 119
- arista, 215, 216
 - dirección, 257
 - múltiple, 216
- aristas
 - hacia atrás, 327
 - paralelas, 353
- asociatividad, 34, 44
- autómatas finitos, 175
- ávido
 - algoritmo de Prim, 329
- axioma, 91
 - de inducción, 166
 - de Peano, 165
- BFS
 - algoritmo, 269
 - en digráficas, 366
 - exploración en digráfica, 366
 - gráficas dirigidas, 369
- bicondicional, 32, 61
- busqueda
 - en profundidad, 322
- búsqueda
 - en profundidad, 322

- cadenas, 10
- cálculos deductivos, 91
- camino, 243
 - longitud o tamaño, 245
- casación de patrones, 181
- cerradura transitiva, 292
- $\chi(G)$, 299
- ciclo, 245
 - BFS, 272
 - hamiltoniano, 261, 266
- circuito
 - euleriano, 248, 249, 362–364
 - algoritmo, 250
 - dirigido, 361
- C_n , 221
 - ciclo, 221
- cola
 - de prioridades, 276, 329
 - estructura de datos, 276
- coloración, 295, 296
 - BFS, 302
 - ciclos, 299, 301
 - gráfica
 - bipartita, 304
 - completa, 301
- colorear, 296
- completo, 96
- componente
 - conexa, 270
- conclusión, 19, 37
- condicional, 29
- conectivo, 24
 - principal, 54
- conexidad
 - digráficas, 358
- conjunción, 28
- conjuntos infinitos numerables, 163
- conmutatividad, 33, 63
- consecuencia lógica, 77, 82, 88
- constante, 4
- contingencia, 35
- contradicción, 35, 36, 76
- contraejemplo, 86, 88
- contrapositiva, 31
- contrarrecíproca, 31
- correcto, 20
- cuantificación
 - alcance, 121
 - existencial, 121
 - universal, 121
 - vacua, 150
 - variable de la, 121
- cuantificador, 117
- cuatro colores
 - teorema de los, 307
- deducción formal, 91
- definición
 - recursiva, 178
 - de funciones, 181
 - general, 178
- $\delta(u, v)$
 - distancia en digráficas, 366
- derivable, 92
- derivación, 12, 91
- descendiente
 - relación, 178
- DFS
 - algoritmo, 325
 - ejecución, 323
- diagonal principal, 230
- digráfica, 203, 216, 353, 356
 - conexa, 359
 - débilmente conexa, 359
 - fuertemente conexa, 358
 - listas de adyacencias, 358
 - matriz de adyacencias, 358
 - matriz de incidencias, 358
- Dijkstra
 - algoritmo de, 276
- dilema constructivo simple, 85, 89
- Dirac
 - teorema de, 262

- dirección
 - asignar, DFS, 327
- distancia, 269
 - en digráficas, 366
- distributividad
 - en lógica de predicados, 149
- disyunción, 29
- $d^+(v)$
 - exgrado, 357
- $d^-(v)$
 - ingrado, 357
- dominante, 34
- dominio
 - bien fundado, 164
 - de interpretación, 139
- E_R^x , 46
- $E[x := R]$, 46
- elemento identidad, 34
- elemento neutro, 34
- eliminación
 - de \leftrightarrow , 93
 - de \wedge , 92
- emparejamiento de patrones, 181
- enunciado, 124
- equivalencia, 32, 61
 - lógica, 61, 147
 - álgebra de, 72
- esquema, 51
 - básico, 55
 - instanciar un, 52
- estado, 22, 42, 75
 - evaluación en un, 43
- evaluación
 - de una expresión, 42, 49
- exgrado, 217, 227, 357, 362, 363
- exploración, 243
- expresión
 - aritmética, 6, 179
 - lógica, 50
- extremo
 - de una arista, 215
- $f^{(n)}$, 119
- factorial, 181
- falacia, 96
- Floyd
 - algoritmo de, 292
- fórmula, 50
 - atómica, 21
 - bien construida, 179
 - cerrada, 124
 - cuantificada, 121
 - insatisfacible, 76
 - no satisfacible, 76
 - razonable, 98
 - válida, 76
- fórmula atómica
 - con predicados, 119
- fuertemente tipificado, 157
- función sucesor, 164
- $G[V_i]$, 299
- generalización
 - existencial, 156
 - universal, 155
- grado, 217
- gráfica, 202, 215
 - acíclica, 247, 315, 318
 - bipartita, 210, 221
 - completa, 222
 - completa, 220
 - coloración, 301
 - componente de una, 247
 - con pesos, 275, 319
 - conexa, 247, 313, 315, 318
 - digráfica, 203
 - dirigida, 203
 - exploración, 243
 - generadora
 - BFS, 273
 - isomorfismo, 236
 - multigráfica, 216

- número de caminos, 284
- plana, 307
- rala, 231
- simple, 216
- subyacente, 357, 359
- gramática, 10
 - de la lógica
 - de predicados, 121, 122
 - proposicional, 24
 - de los términos, 118
 - formal, 10
- hojas, 12
- holgura, 209, 211
 - slack*, 211
- homeomorfismo, 309
- \mathcal{I} , 75
- idempotencia, 35
- identificador, 51
- implicación, 29
 - contrapositiva, 31
 - contrarrecíproca, 31
 - recíproca, 31
- incidencia
 - en vértices, 217
- incógnitas, 44
- índice, 119
- inducción, 163
 - cambio de base, 170
 - completa, 172
 - en fórmulas, 190
 - en listas, 187
 - en árboles, 192
 - estructural, 186
 - fuerte, 172
 - matemática, 163
- inferencia
 - regla de, 59
- ingrado, 217, 227, 357, 362, 363
- instanciación
 - existencial, 156
 - universal, 155
- instanciar
 - un esquema, 52
- interpretación, 75, 82, 88
 - función de, 75
- introducción
 - de \vee , 92
 - de \wedge , 92
- inversa, 31
- isomorfismo
 - de digráficas, 361
 - de digráficas con pesos, 371
 - de gráficas, 238
- juicio
 - afirmativo, 157
 - aristotélico, 130
 - existencial
 - afirmativo, 130, 156
 - negativo, 130
 - universal
 - afirmativo, 130, 156
 - negativo, 130
- \mathcal{K}_5 , 309
- $K_{m,n}$
 - bipartita completa, 222
- K_n , 220
- Kruskal
 - algoritmo de, 346
 - árbol de peso mínimo, 346
 - subgráfica conexa acíclica, 346
- $\mathcal{K}_{3,3}$, 309
- Kuratowski
 - teorema de, 309
- \mathcal{L} , 91
- laberinto
 - DFS, 322
- lazo, 216, 230, 353
- Leibniz
 - regla de, 63, 72

- lenguaje
 - de la lógica
 - de predicados, 118
 - de la lógica proposicional, 19
 - formal, 3
 - fuertemente tipificado, 157
 - natural, 3
- LIFO
 - pila, 324
- lista
 - concatenación, 183
 - de adyacencias, 231
 - de incidencias, 233
 - en multigráficas, 354
 - finita, 179
 - longitud, 182
 - reversa, 183
 - tipo, 158
- literal, 98, 103
 - complementaria, 103
- lógica
 - de predicados de primer orden, 118
 - proposicional, 21
- mapa, 307
- matriz
 - cuadrada, 229
 - de adyacencias, 228
 - de distancias, 293
 - de incidencias, 230
 - en multigráficas, 354
 - de pesos, 293
 - rala, 231
 - simétrica, 230
 - triangular, 235
- metaexpresión, 36
- metalenguaje, 36
- método
 - analítico, 55
 - generador, 55
- micromundo, 133
 - de cubos, 133
 - de figuras geométricas, 134
- modelo, 4, 75, 98
 - matemático, 4
- \models , 36
- modus
 - ponens, 59, 82
 - tollens, 84
- multiarco, 356
- multigráfica, 230, 248, 249, 353
 - ciclo euleriano en, 353
 - componente conexa, 354
 - paseo euleriano en, 353
 - trayectoria en, 353
- nand, 42
- naturales
 - definición recursiva, 164
 - exponenciación, 181
 - producto, 165
 - suma, 165
- negación, 27
 - en lógica de predicados, 131
- notación
 - infija, 5
 - prefija, 5
 - sufija o polaca, 5
- numerable, 163
- número
 - cromático, 299
 - en gráfica plana, 308
 - de caminos, 368
- números naturales, 163
- operador, 4
 - binario, 5
 - de cuantificación, 117
 - n-ario, 5
 - unario, 5
- Ore
 - teorema de, 265

- \mathcal{P} , 324
- palabras, 10
- palíndroma, 195
- paseo
 - equino, 268
 - reentrante, 268
 - euleriano, 249, 256, 362–364
 - dirigido, 362
- patrón, 181
- Peano
 - axioma de inducción, 166
 - axiomas de, 165
- PERT, 205
- pesos
 - homogéneos, 293
- Petersen
 - gráfica de, 267
- pila, 324
 - tipo, 159
- poliominó, 176
- pop, 324
- precedencia, 44
- predicado, 114
 - calificador, 156
 - tablas para, 142
- premisa, 19, 37
- prenexación, 150
- producciones, 10
- propiedades, 116
- proposición, 4, 22, 50
 - atómica, 21, 23
 - compuesta, 23, 55
 - definición, 23
 - lógica, 4
- prueba, 91
- punto ciego, 322
 - en DFS, 324
- punto fijo
 - en algoritmos, 291
- push, 324
- raíz, 12
- rama
 - de un tableau, 103
- rango, 54
- razonamiento
 - ecuacional, 63, 72
- recíproca, 31
- recursión, 163
- reflexividad, 63
- regla
 - de reescritura, 10
 - recursiva, 178
 - sintáctica, 10
- regla de inferencia, 59
 - α , 105
 - β , 105
 - casos simple, 93
 - de Leibniz, 64
 - eliminación
 - de \leftrightarrow , 93
 - de \wedge , 92
 - generalización
 - existencial, 156
 - universal, 155
 - inconsistencia, 93
 - instanciación
 - existencial, 156
 - universal, 155
 - introducción
 - de \vee , 92
 - de \wedge , 92
 - introducción de \leftrightarrow , 93
 - modus
 - ponens, 92
 - tollens, 92
 - para tableaux, 105
 - σ , 105
 - silogismo
 - disyuntivo, 93
 - hipotético, 92
- ruta crítica, 205
- satisface, 75

- satisfacible, 75
- semántica, 5, 26, 74, 165
- silogismo
 - disyuntivo, 93
 - hipotético, 81, 89, 92
- símbolo
 - no terminal, 10
 - terminal, 10, 13
- sintaxis, 5, 165
- sólido, 20
- subconjuntos
 - de aristas, 346
- subexpresión, 11
- subgráfica, 223, 318
 - generadora, 225
 - inducida, 224
- sustitución
 - por la izquierda, 11
 - propiedad de, 58
 - textual, 46, 63
 - variables escondidas en, 48
- tablas de verdad, 27
- tableaux, 98
 - algoritmos con, 108
 - contradicción, 104
 - fórmula contingente, 105
 - semánticos, 98
 - tautología, 105, 108
- tautología, 35, 57, 61, 76
- teorema
 - de la deducción, 95
 - de los cuatro colores, 307
- teoría
 - de la demostración, 91
 - de modelos, 91
- tercero excluido, 36
- término, 118
- tipado, 157
- tipo, 156
 - abreviado, 157
- top, 324
- torneo
 - digráficas, 365
- transitividad, 63
- trayectoria, 244
 - antidirigida, 357
 - cerrada, 245
 - dirigida, 356
 - más corta, 367
 - en multigráficas, 353
 - hamiltoniana
 - en digráficas, 365
 - más corta, 293
 - no dirigida, 357
- universo
 - de discurso, 114, 139
- valor, 22
 - forzar un, 86
- variable, 4, 10
 - acotada, 124
 - libre, 123
 - ligada, 123
 - proposicional, 24
- verdad
 - noción de, 143
- vértice, 203, 215
 - alcanzable, 270
- Warshall
 - algoritmo de, 290