

Algoritmo para decidir si una
forma unitaria es de tipo \mathbb{A}_n



UNIVERSIDAD AUTÓNOMA DEL
ESTADO DE MORELOS

Facultad
de Ciencias

TESIS

ALGORITMO PARA DECIDIR SI UNA FORMA UNITARIA ES DE TIPO \mathbb{A}_n

Presenta

Mario Alberto Abarca Sotelo

para obtener el título de
Licenciado en Ciencias (Computación)

Director de tesis:

Dr. Antonio Daniel Rivera López

Mario Alberto Abarca Sotelo

Algoritmo para decidir si una forma unitaria es de tipo \mathbb{A}_n

Licenciado en ciencias, Computación.

Esta tesis fue evaluada por el siguiente comité tutelar:



- Dra. Larissa Sbitneva, Presidente
- Dr. Antonio Daniel Rivera López, Secretario
- Dra. María Elena Lárraga Ramirez, Vocal

DEPARTAMENTO DE COMPUTACIÓN, FACULTAD DE CIENCIAS.



UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS. © 2011

Este trabajo está publicado bajo una licencia Creative Commons Atribución-CompartirIgual 3.0 Unported (cc BY-SA 3.0).

Usted es libre de:

-  Copiar, distribuir, ejecutar y comunicar públicamente la obra.
-  Hacer obras derivadas.

Bajo las condiciones siguientes:

-  **Atribución** — Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciante (pero no de una manera que sugiera que tiene su apoyo o que apoyan el uso que hace de su obra).
-  **Compartir bajo la Misma Licencia** — Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

Para más detalles consulte la siguiente página web:

<http://creativecommons.org/licenses/by-sa/3.0/legalcode>

*A la madre con mucho cariño,
y a los profesores también.*

PREFACIO

Muchos años atrás me sentía atraído por la idea de que la computadora pudiera realizar mi tarea de matemáticas. Ya adentrado en la carrera de ciencias de la computación caí en cuenta de que no era el único que había pensado de manera tan morbosa, y que de hecho hay toda una rama de las ciencias computacionales dedicada exclusivamente a eso. Admitidamente mis intereses actuales son más amplios y menos efímeros, pero la idea fundamental sigue detrás: es necesario enseñar a las computadoras cómo resolver varios de los problemas contemporáneos de la ciencia e ingeniería, y eso incluye, por supuesto, la resolución automática de varios problemas matemáticos.

Buscando satisfacer un poco de mi inquietud, accedí a realizar la tesis acerca de esta clasificación de ciertos polinomios llamados “formas cuadráticas unitarias”. Para mí fue un tema completamente nuevo, pero me esmeré mucho en que la explicación fuera clara y sencilla. Espero que el documento que tienes en tus manos (o monitor) pueda ser útil, no solo si estás interesado en el tema, sino también cuando necesites aprender las ideas básicas de las formas cuadráticas, repasar a detalle el algoritmo de recorrido en profundidad o entender qué es lo que hace un computólogo trabajando en matemáticas.

Para leer esta tesis solo es necesario tener un poco de conocimientos de álgebra lineal y teoría de gráficas, así como estar familiarizado con algoritmos o programación. He tratado de que el texto sea lo más fluido posible, enfatizando rigor más que formalismo. Así pues, casi toda la terminología ocupada en este documento se define explícitamente *al instante* utilizando **letras negritas** y se ilustran varios conceptos con ejemplos. También traté de que esta tesis fuera lo más autocontenida posible, de manera que el apéndice “A” repasa los conceptos básicos de matemáticas discretas, algoritmia y álgebra lineal. Si deseas referencias más detalladas sobre estos temas consulta Grimaldi (1998), Dasgupta

et al. (2008) y Lay (2001). El primer libro explica muy bien los fundamentos de teoría de gráficas, una breve introducción al recorrido en profundidad y las matemáticas necesarias para hacer análisis de algoritmos; el segundo es todo un curso de diseño y análisis de algoritmos; el tercero contiene una amigable introducción a las formas cuadráticas y álgebra lineal en general. Estos libros los recomiendo a título personal como estudiante.

Aprovecho para darle gracias a mi madre quien me apoyó en todo, pero especialmente en mis estudios; no solo los obligatorios, sino también aquellos pasatiempos como la programación de computadoras que hoy rinde frutos. Gracias a mis sinodales, que con mucho esmero trataron de formarme profesionalmente. Especialmente a mi asesor de tesis por su paciencia y dedicación. Gracias a los profesores que en más de una ocasión supieron darme una lección de vida. Gracias a mis amigos, que me dieron certidumbre en momentos inciertos. Gracias a mis lectores, o sea tú (porque realmente es muy raro que alguien se tome la molestia de leer un trabajo de tesis).

Este documento fue producido en el editor LYX sobre $\text{L}\text{A}\text{T}\text{E}\text{X} 2_{\epsilon}$ en una pc con ubuntu. Se usaron los tipos letra del paquete TX Fonts (eso incluye a Nimbus Roman y Nimbus Sans). Para el estilo tipográfico se usaron los paquetes **KOMA-Script**, `titlesec`, $\mathcal{A}\mathcal{M}\mathcal{S}\text{-L}\text{A}\text{T}\text{E}\text{X}$, `algorithm2e`, `Listings` y `BibTEX`. La bibliografía fue capturada en `JabRef` y las figuras fueron diseñadas en `Inkscape`. El diseño de la edición estuvo a cargo del autor.

—Mario

ÍNDICE GENERAL

Prefacio	vii
Nomenclatura	xi
1. Introducción	1
1.1. Formas cuadráticas	1
Cambio de variable	3
1.2. Formas unitarias	6
Cambio de variable entero	6
Gráficas de Dynkin	7
1.3. El problema que se aborda en esta tesis	15
2. Antecedentes	17
2.1. El método de las inflaciones	17
Inflaciones y deflaciones	17
Descripción y justificación	20
2.2. Ensamblaje por \mathbb{A} -bloques	27
Visión general del ensamble	27
Los \mathbb{A} -bloques	29
Ensamblaje de formas \mathbb{A}_n	33
3. Descomposición de gráficas	37
3.1. El algoritmo de recorrido en profundidad	37
Representación de gráficas	37
Descripción del recorrido	38
Una versión iterativa	40
Árboles recubridores en profundidad	41
3.2. Descomposición automática de gráficas	43
Caracterizando puntos de articulación	43

Uso del recorrido en profundidad	44
Algunos detalles de implementación	44
4. Caracterización algorítmica	48
4.1. Propiedad de la partición bicolor	48
4.2. Árboles recubridores para formas \mathbb{A}_n	50
Árboles recubridores de \mathbb{A} -bloques	50
Árboles recubridores en profundidad para formas \mathbb{A}_n	52
4.3. Resultado principal	54
5. El algoritmo, su implementación y conclusiones	57
5.1. Descripción, justificación y análisis	57
Descripción formal	60
Análisis de la complejidad temporal	61
5.2. Implementación y ejemplos	62
Implementando gráficas ponderadas	62
El programa	65
Ejemplos de ejecución	68
5.3. Conclusiones y trabajo a futuro	71
A. Conceptos preliminares	73
A.1. Elementos de matemáticas discretas	73
Conjuntos	73
Sucesiones y m -adas	74
Funciones y relaciones	76
Gráficas	80
Árboles	82
Lógica booleana	84
A.2. Elementos de algoritmia	84
Problemas computacionales	84
Algoritmos y pseudocódigo	85
Análisis de algoritmos	89
A.3. Elementos de álgebra lineal	92
Matrices	92
Espacios vectoriales	94
Transformaciones lineales	96
Operaciones elementales sobre matrices	97

NOMENCLATURA

- $i \text{---} j$ arista entre los vértices i y j , página 80
- $u \rightsquigarrow v$ v es alcanzable desde u , página 81
- $x \leftarrow f$ operador de asignación, página 87
- A^{-1} matriz inversa de A , página 94
- \mathbb{A}_n gráfica de Dynkin \mathbb{A}_n , página 8
- $\langle \alpha \rangle_T$ subgráfica inducida por los vértices que hay en el camino definido por los extremos de la arista α en el árbol T , página 53
- A^T transpuesta de A , página 93
- \mathbf{B}_q gráfica asociada a la forma q , página 7
- \wedge conjunción, página 84
- \vee disyunción, página 84
- $d(v)$ tiempo de descubrimiento del vértice v en un recorrido en profundidad, página 42
- $x \in C$ x elemento del conjunto C , página 73
- $E(G)$ conjunto de aristas de G , página 80
- $\mathbf{F}_{m,m'}$ \mathbb{A} -bloque de $m + m'$ vértices, página 27
- $f(v)$ tiempo de finalización del vértice v en un recorrido en profundidad, página 42
- $\ell(u)$ tiempo de descubrimiento del ancestro más remoto adyacente a los descendientes de u , página 43

\mathbb{N}	conjunto de números naturales, página 74
\neg	negación, página 84
$O(g)$	cota superior asintótica, página 89
q_G	forma unitaria asociada a la gráfica G , página 7
\mathbb{R}	conjunto de números reales, página 74
$A \subseteq B$	A subconjunto de B , página 73
$A \subset B$	A subconjunto propio de B , página 74
T_{ij}^-	inflación, página 19
T_{ij}^+	deflación, página 19
$V(G)$	conjunto de vértices de G , página 80
\mathbb{Z}	conjunto de números enteros, página 74

CAPÍTULO 1

INTRODUCCIÓN

La base para la poesía y el descubrimiento científico es la habilidad de comprender las semejanzas en lo diferente y las diferencias en lo semejante.

(Jacob Bronowski)

En este capítulo se presenta una introducción a las formas cuadráticas, sus representaciones y su clasificación. Esto con el fin permitir al lector un mejor entendimiento del problema que se plantea en este trabajo de tesis, y el cual se presenta al final de este capítulo.

1.1. FORMAS CUADRÁTICAS

Esta sección fue adaptada de Lay (2001). Algunas demostraciones aquí omitidas pueden consultarse en dicha referencia.

Fijemos un conjunto de variables $\{x_1, x_2, \dots, x_n\}$. Un **monomio** es un producto de la forma $x_1^{e_1} x_2^{e_2} \dots x_n^{e_n}$ donde cada e_i es un número natural. Un **término** se forma al multiplicar a un monomio por una constante, a la cual llamamos **coeficiente** del término. Un **polinomio** es una suma finita de términos.

Por ejemplo $3x^2y + 2xyz^3 - x$ es un polinomio sobre las variables x, y, z con tres monomios: (x^2y) , (xyz^3) y (x) cuyos respectivos coeficientes son 3, 2 y -1 .

Claramente todo polinomio sobre n variables es una función con dominio \mathbb{R}^n y rango \mathbb{R} . Una **forma cuadrática** es un polinomio $q : \mathbb{R}^n \rightarrow \mathbb{R}$ (con $n > 0$) si cada monomio del mismo es una variable al cuadrado

Consideramos al cero como un número natural (v. pag. 74).

Esta terminología fue tomada de von zur Gathen & Gerhard (2003).

o la multiplicación de dos variables. Esto es equivalente a decir que q se puede expresar como

$$q(x_1, x_2, \dots, x_n) = \sum_{i=1}^n q_{ii} x_i^2 + \sum_{j=2}^n \sum_{i=1}^{j-1} q_{ij} x_i x_j.$$

Los ejemplos más típicos aparecen al lado izquierdo del signo igual de las ecuaciones en las cónicas con centro en el origen

$$a x^2 + 2 b x y + c y^2 = d$$

y de las superficies cuadráticas con centro en el origen

$$a x^2 + 2 d x y + 2 e x z + b y^2 + 2 f y z + c z^2 = g$$

donde a, b, c, d, e, f y g son números reales. Este tipo de polinomios surgen de manera natural en diversas áreas de la ingeniería, procesamiento de señales, cinética, economía, geometría diferencial y estadística.

En algunos cursos básicos de álgebra lineal se suele definir el concepto de forma cuadrática como una función que se puede escribir como

$$q(\vec{x}) = \frac{1}{2} \vec{x}^T A \vec{x}$$

para alguna matriz simétrica A . En realidad esta representación matricial es equivalente a nuestra definición con monomios; a continuación veremos cómo pasar de una representación a otra. Primero notemos que $\frac{1}{2} \vec{x}^T A \vec{x}$ se puede reescribir como sigue:

$$\begin{aligned} \frac{1}{2} \vec{x}^T A \vec{x} &= \frac{1}{2} [x_1, x_2, \dots, x_n] \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \\ &= \frac{1}{2} \left[\sum_{i=1}^n a_{i1} x_i, \sum_{i=1}^n a_{i2} x_i, \dots, \sum_{i=1}^n a_{in} x_i \right] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \\ &= \frac{1}{2} \left(\sum_{i=1}^n a_{i1} x_i x_1 + \cdots + \sum_{i=1}^n a_{in} x_i x_n \right) \\ &= \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n a_{ij} x_i x_j \end{aligned}$$

Al desarrollar esta suma se puede ver que el coeficiente de $x_i x_j$ es $q_{ij} = \frac{1}{2} (a_{ij} + a_{ji})$ porque $x_i x_j = x_j x_i$, pero como habíamos supuesto que A es simétrica ($a_{ij} = a_{ji}$) entonces $q_{ij} = a_{ij}$. En particular cuando $i = j$ se tiene que el coeficiente de x_i^2 es $q_{ii} = \frac{1}{2} a_{ii}$. Por lo tanto tenemos la siguiente identidad:

$$\frac{1}{2} \vec{x}^T A \vec{x} = \sum_{i=1}^n \frac{1}{2} a_{ii} x_i^2 + \sum_{j=2}^n \sum_{i=1}^{j-1} a_{ij} x_i x_j. \quad (1.1)$$

La matriz simétrica A de esta identidad se conoce como **matriz asociada a la forma cuadrática q** . Nosotros la vamos a denotar por A_q , y según la ecuación 1.1 se puede calcular como sigue:

$$a_{ij} = \begin{cases} q_{ij} & \text{si } i < j \\ 2 q_{ij} & \text{si } i = j \\ q_{ji} & \text{si } i > j \end{cases} \quad (1.2)$$

Por ejemplo, para $q(x, y, z) = 5x^2 + 3y^2 + 2z^2 - xy + 8yz$ tenemos

$$q(x, y, z) = \frac{1}{2} \begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} 10 & -1 & 0 \\ -1 & 6 & 8 \\ 0 & 8 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}.$$

CAMBIO DE VARIABLE

Un **cambio de variable** es una ecuación de la forma

$$\vec{x} = P \vec{y} \quad \text{o bien} \quad \vec{y} = P^{-1} \vec{x} \quad (1.3)$$

donde P es una matriz invertible y \vec{y} es un nuevo vector variable en \mathbb{R}^n . Si se aplica el cambio de variable (1.3) sobre una forma cuadrática $q(\vec{x}) = \frac{1}{2} \vec{x}^T A \vec{x}$ se obtiene otra forma cuadrática $q'(\vec{y})$ cuya matriz asociada es $P^T A P$:

$$\frac{1}{2} \vec{x}^T A \vec{x} = \frac{1}{2} (P \vec{y})^T A (P \vec{y}) = \frac{1}{2} (\vec{y}^T P^T) A (P \vec{y}) = \frac{1}{2} \vec{y}^T (P^T A P) \vec{y}.$$

Garantizamos que $P^T A P$ es una matriz simétrica (y por tanto que en verdad corresponde a otra forma cuadrática) porque

$$\begin{aligned} (P^T A P)^T &= P^T A^T (P^T)^T \\ &= P^T A^T P \\ &= P^T A P. \end{aligned}$$

Mediante el cambio de variable $\vec{y} = P \vec{x}$ tenemos que $q(\vec{x}) = q'(\vec{y})$ y diremos que q y q' son **equivalentes** mediante la matriz invertible P . Si denotamos $\vec{y} = L(\vec{x})$ (es decir, $L(\vec{x}) = P \vec{x}$) entonces $q'(\vec{y}) = q'(L(\vec{x})) = (q' \circ L)(\vec{x})$, por lo tanto

$$q' = q \circ L.$$

Por ejemplo, consideremos la forma cuadrática $x^2 - 5y^2 - 8xy$ con matriz asociada

$$A = \begin{bmatrix} 2 & -8 \\ -8 & -10 \end{bmatrix}$$

y el cambio de variable definido por la ecuación

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2/\sqrt{5} & 1/\sqrt{5} \\ -1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix} \begin{bmatrix} z \\ w \end{bmatrix} = \begin{bmatrix} \frac{2z+w}{\sqrt{5}} \\ \frac{-z+2w}{\sqrt{5}} \end{bmatrix}$$

La matriz $P = \begin{bmatrix} 2/\sqrt{5} & 1/\sqrt{5} \\ -1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix}$ es invertible, y su inversa es

$$P^{-1} = \begin{bmatrix} 2/\sqrt{5} & -1/\sqrt{5} \\ 1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix} = P^T.$$

Calculamos $P^T A P = \begin{bmatrix} 6 & 0 \\ 0 & -14 \end{bmatrix}$ para concluir que

$$x^2 - 5y^2 - 8xy = 3z^2 - 7w^2.$$

Esto es precisamente lo que hubiéramos obtenido haciendo las sustituciones

$$\begin{aligned} x &\leftarrow \frac{2z+w}{\sqrt{5}} \\ y &\leftarrow \frac{-z+2w}{\sqrt{5}} \end{aligned}$$

sobre la expresión $x^2 - 5y^2 - 8xy$.

Vale la pena notar unas cuantas cosas sobre este ejemplo: En primera P resultó ser una **matriz ortogonal**, es decir, tal que $P^{-1} = P^T$. En segunda, que A es una matriz **ortogonalmente diagonalizable**, es decir, existe una matriz ortogonal P tal que $P^{-1} A P$ es una matriz diagonal.

TEOREMA 1.1. *Toda matriz es ortogonalmente diagonalizable si y solo si es simétrica.*

Este teorema se deja sin demostración. Lo importante es que, a partir de este teorema, y dado que las matrices asociadas a las formas cuadráticas son simétricas, se sigue inmediatamente el siguiente resultado:

TEOREMA 1.2 (de los ejes principales). *Toda forma cuadrática $q(\vec{x})$ es equivalente mediante una matriz ortogonal P a una forma cuadrática*

$$q'(\vec{y}) = \lambda_1 y_1^2 + \lambda_2 y_2^2 + \cdots + \lambda_n y_n^2 \quad (1.4)$$

Demostración. La matriz A asociada a q es simétrica, luego por teorema anterior existe una matriz P tal que

$$P^{-1} A P = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}$$

Haciendo $\vec{x} = P\vec{y}$ se obtiene $q'(\vec{y}) = \frac{1}{2} \vec{y}^T (P^T A P) \vec{y} = \lambda_1 y_1^2 + \lambda_2 y_2^2 + \cdots + \lambda_n y_n^2$. ■

Como veremos, esta representación es extremadamente útil para clasificar formas cuadráticas. Diremos que una forma cuadrática q es **definida positiva** si $q(\vec{x}) > 0$ para todo $\vec{x} \neq \vec{0}$.

LEMA 1.3. *Si dos formas cuadráticas $q(\vec{x})$ y $q'(\vec{y})$ son equivalentes mediante el cambio de variable $\vec{y} = P\vec{x}$, y si una de ellas es definida positiva entonces la otra también lo es.*

Demostración. Supongamos que $q(\vec{x})$ es definida positiva y defínase la transformación lineal $L(\vec{x}) = P\vec{x}$, entonces $L(\vec{0}) = L(\vec{0} + \vec{0}) = L(\vec{0}) + L(\vec{0})$, de donde $L(\vec{0}) = \vec{0}$. Como P es una matriz invertible entonces $L^{-1}(\vec{x}) = P^{-1}\vec{x}$. En particular L debe ser inyectiva y por tanto $L(\vec{x}) = \vec{0}$ implica $\vec{x} = \vec{0}$. Si q es definida positiva entonces $q(\vec{x}) = q'(\vec{y}) > 0$ para todo $\vec{x} \neq \vec{0}$; por lo tanto q' es definida positiva. Si en cambio hubiéramos supuesto que q' es definida positiva, entonces aplicando el mismo razonamiento con $L(\vec{y}) = P^{-1}\vec{y}$ se llega a la conclusión de que q también es definida positiva. ■

TEOREMA 1.4. *Sea q una forma cuadrática y supongamos que*

$$q(\vec{x}) = q'(\vec{y}) = \lambda_1 y_1^2 + \lambda_2 y_2^2 + \cdots + \lambda_n y_n^2$$

entonces q es definida positiva si y solo si todos los $\lambda_i > 0$.

Demostración. Por lema anterior q es definida positiva si y solo si q' es definida positiva. Por contradicción, si $\lambda_i \leq 0$ entonces defínase $\vec{y} = (y_1, y_2, \dots, y_n)$ donde todos los $y_k = 0$ excepto $y_i = 1$. Claramente $\vec{y} \neq \vec{0}$ pero $q(\vec{y}) = \lambda_i \leq 0$; por lo tanto q no es definida positiva. ■

Como las formas cuadráticas se pueden representar mediante matrices entonces podemos decir que una matriz A es definida positiva si su forma cuadrática asociada $q(\vec{x}) = \frac{1}{2} \vec{x}^T A \vec{x}$ es definida positiva.

A continuación enunciamos (sin demostración) un teorema típico de los cursos de análisis numérico que nos da un criterio computacionalmente eficiente para decidir cuándo una matriz simétrica es definida positiva:

TEOREMA 1.5. *Una matriz A es definida positiva si y sólo si tiene **factorización de Cholesky**, es decir, se puede escribir como $A = R^T R$ donde R es una matriz triangular superior con entradas positivas.*

1.2. FORMAS UNITARIAS

Esta sección fue adaptada de Ringel (1985) y Gabriel & Roĭter (1997).

Las **formas cuadráticas enteras** son un caso especial de las formas cuadráticas donde todos los coeficientes q_{ij} son todos números enteros. Si además exigimos que $q_{ii} = 1$ para $i = 1, 2, \dots, n$ entonces las llamamos **formas cuadráticas unitarias**. Dado que en el resto de este trabajo solamente se trabaja con formas cuadráticas enteras que son unitarias, las llamaremos simplemente **formas unitarias** y se sobreentiende que son cuadráticas y enteras.

CAMBIO DE VARIABLE ENTERO

Una **matriz entera** M es una matriz tal que todas sus entradas son números enteros, y es **\mathbb{Z} -invertible** si además su matriz inversa M^{-1} también es una matriz entera. Un **cambio de variable entero** es un cambio de variable $\vec{y} = P \vec{x}$ donde P es una matriz \mathbb{Z} -invertible. Los cambios de variable enteros tienen la propiedad de que transforman formas cuadráticas enteras en formas cuadráticas enteras (demostración: sean A y P matrices enteras, entonces $P^T A P$ es una matriz entera). Cuando $q(\vec{x}) = q'(\vec{y})$ mediante el cambio de variable entero $\vec{y} = P \vec{x}$, diremos que q y q' son **\mathbb{Z} -equivalentes** mediante la matriz \mathbb{Z} -invertible P .

GRÁFICAS DE DYNKIN

Si q es una forma unitaria entonces le asociaremos una multigráfica \mathbf{B}_q construida de la siguiente manera:

- Existe un vértice x_i por cada variable x_i
- Si $q_{ij} > 0$ entonces hay q_{ij} aristas punteadas entre los vértices x_i y x_j (cada una de ellas denotada como $x_i \cdots x_j$)
- Si $q_{ij} < 0$ entonces hay $|q_{ij}|$ aristas sólidas entre los vértices x_i y x_j (cada una de ellas denotada por $x_i \text{---} x_j$)

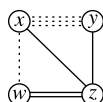


FIGURA 1.1: Gráfica asociada a $3xy - xz + xw - yz - 2zw$

Claramente podemos revertir este proceso y asociar a toda gráfica G (posiblemente con aristas punteadas) una forma unitaria que denotaremos por q_G . En efecto, toda la información de q_G está codificada en G : La existencia de un vértice x nos dice la forma cuadrática está definida sobre alguna variable x y que contiene el término x^2 (porque q es una forma cuadrática unitaria). El coeficiente del monomio xy es $c = a_p - a_s$ donde a_p es la cantidad de aristas punteadas entre x y y , y a_s es la cantidad de aristas sólidas entre estos mismos vértices. Más aún cabe recalcar que nosotros estamos descartando gráficas con lazos (v. pag. 80), por lo que cualquier gráfica en verdad define a una forma cuadrática unitaria.

Las gráficas de Dynkin se presentan en la figura 1.2. Note que las gráficas \mathbb{D}_n están definidas solamente para $n \geq 4$ mientras que las gráficas \mathbb{E}_n solo se definen para $n = 6, 7, 8$. La importancia de estas gráficas radica en que permite dar una caracterización elegante de las formas unitarias que son definidas positivas tal como se explica a continuación.

Decimos que la forma unitaria q es **positiva** si es definida positiva.

TEOREMA 1.6. *Toda forma unitaria q es positiva si y solamente si q es \mathbb{Z} -equivalente a otra forma unitaria q' donde cada componente conexa de $\mathbf{B}_{q'}$ es una gráfica de Dynkin.*

La definición de componente conexa aparece en la página 81.

Notación	Gráfica
A_n	
D_n	
E_6	
E_7	
E_8	

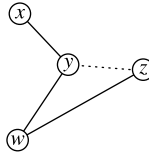
FIGURA 1.2: Gráficas de Dynkin. El subíndice n indica la cantidad de vértices que tiene la gráfica.

Antes de pasar a la demostración hay que comprender el enunciado del teorema. El teorema dice que la forma unitaria $q(\vec{x}) = \frac{1}{2} \vec{x}^T A \vec{x}$ es positiva si y solamente si se puede llevar, mediante un cambio de variable entero $\vec{y} = P \vec{x}$, a la forma $q'(\vec{y}) = \frac{1}{2} \vec{y}^T (P^T A P) \vec{y}$ donde $B_{q'}$ tiene la propiedad de cada una de sus componentes conexas es una gráfica de Dynkin. A dicha gráfica $B_{q'}$ se le llama el **tipo Dynkin** de q . Veamos un ejemplo:

Por ejemplo, la forma unitaria

$$q(x, y, z, w) = x^2 + y^2 + z^2 + w^2 - xy + yz - yw - zw \quad (1.5)$$

tiene asociada la gráfica



y su matriz asociada es

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & 1 & -1 \\ 0 & 1 & 2 & -1 \\ 0 & -1 & -1 & 2 \end{bmatrix}.$$

Consideremos la matriz

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

con inversa

$$P^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Ahora tenemos

$$\begin{aligned}
 P^T A P &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & 1 & -1 \\ 0 & 1 & 2 & -1 \\ 0 & -1 & -1 & 2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & 1 & 2 & -1 \\ 0 & -1 & -1 & 2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}
 \end{aligned}$$

Entonces podemos concluir que q es \mathbb{Z} -equivalente a la forma unitaria

$$q'(x, y, z, w) = x^2 + y^2 + z^2 + w^2 - xy - yz - zw,$$

con gráfica asociada



Esta gráfica es isomorfa a \mathbb{A}_4 , por lo tanto ese es el tipo Dynkin de q .

La demostración del teorema 1.6 se divide en dos partes: primero se demuestra que las gráficas de Dynkin son las únicas gráficas conexas de aristas sólidas que definen formas unitarias que son definidas positivas (tomando en cuenta inclusive a las gráficas de aristas múltiples, ver corolario 1.9 en la página 12); luego se demuestra que siempre es posible hacer cambios de variable enteros de tal manera que la gráfica resultante no contenga aristas sólidas. Este último tema será abordado en la sección 2.1.

LEMA 1.7. *Las gráficas de Dynkin son las únicas gráficas conexas de aristas sólidas que tienen asociadas formas unitarias positivas.*

Para comenzar necesitamos convencernos de que en verdad las gráficas de Dynkin definen formas unitarias positivas. Comenzamos con las gráficas \mathbb{A}_n :

$$x_1 \text{---} x_2 \text{---} \cdots \text{---} x_n$$

Consideremos la siguiente identidad:

$$\frac{1}{2} (x_i - x_j)^2 = \frac{1}{2} x_i^2 - x_i x_j + \frac{1}{2} x_j^2.$$

Podemos reescribir la forma unitaria asociada a \mathbb{A}_n como

$$\begin{aligned}
 q_{\mathbb{A}_n}(\vec{x}) &= \sum_{i=1}^n x_i^2 - \sum_{i=1}^{n-1} x_i x_{i+1} \\
 &= \frac{1}{2} x_1^2 + \sum_{i=1}^{n-1} \left(\frac{1}{2} x_i^2 + \frac{1}{2} x_{i+1}^2 \right) + \frac{1}{2} x_n^2 + \sum_{i=1}^{n-1} (-x_i x_{i+1}) \\
 &= \frac{1}{2} x_1^2 + \sum_{i=1}^{n-1} \left(\frac{1}{2} x_i^2 - x_i x_{i+1} + \frac{1}{2} x_{i+1}^2 \right) + \frac{1}{2} x_n^2 \\
 &= \frac{1}{2} x_1^2 + \sum_{i=1}^{n-1} \frac{1}{2} (x_i - x_{i+1})^2 + \frac{1}{2} x_n^2.
 \end{aligned}$$

Por el teorema 1.4 concluimos que $q_{\mathbb{A}_n}$ es definida positiva. Una demostración similar muestra que $q_{\mathbb{D}_n}$ es definida positiva; en este caso además usamos la identidad

$$\frac{1}{2} \left[(x_3 - x_2 - x_1)^2 + (x_2 - x_1)^2 \right] = x_1^2 + x_2^2 + \frac{1}{2} x_3^2 - x_1 x_3 - x_2 x_3;$$

de tal forma que

$$\begin{aligned}
 q_{\mathbb{D}_n}(\vec{x}) &= \sum_{i=1}^n x_i^2 - x_1 x_3 - \sum_{i=2}^{n-1} x_i x_{i+1} \\
 &= \frac{1}{2} \left[(x_3 - x_2 - x_1)^2 + (x_2 - x_1)^2 + \sum_{i=3}^{n-1} (x_i - x_{i+1})^2 + x_n^2 \right].
 \end{aligned}$$

Para las gráficas \mathbb{E}_6 , \mathbb{E}_7 y \mathbb{E}_8 usaremos el siguiente razonamiento: Supongamos que A_Δ es la matriz asociada a la gráfica Δ (con $\Delta = \mathbb{E}_6, \mathbb{E}_7, \mathbb{E}_8$); si existe una matriz R_Δ tal que $A_\Delta = R_\Delta^T R_\Delta$ entonces por teorema 1.5 se sigue que Δ define una forma unitaria definida positiva. En efecto:

$$A_{\mathbb{E}_6} = \begin{bmatrix} 2 & 0 & 0 & -1 & 0 & 0 \\ 0 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ -1 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

$$R_{\mathbb{E}_6} = \begin{bmatrix} \sqrt{2} & 0 & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & \sqrt{2} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 \\ 0 & 0 & \frac{\sqrt{3}}{\sqrt{2}} & -\frac{\sqrt{2}}{\sqrt{3}} & 0 & 0 \\ 0 & 0 & 0 & \frac{\sqrt{5}}{\sqrt{6}} & -\frac{\sqrt{6}}{\sqrt{5}} & 0 \\ 0 & 0 & 0 & 0 & \frac{2}{\sqrt{5}} & -\frac{\sqrt{5}}{2} \\ 0 & 0 & 0 & 0 & 0 & \frac{\sqrt{3}}{2} \end{bmatrix}$$

$$A_{\mathbb{E}_7} = \begin{bmatrix} 2 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ -1 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

$$R_{\mathbb{E}_7} = \begin{bmatrix} \sqrt{2} & 0 & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 \\ 0 & \sqrt{2} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\sqrt{3}}{\sqrt{2}} & -\frac{\sqrt{2}}{\sqrt{3}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\sqrt{5}}{\sqrt{6}} & -\frac{\sqrt{6}}{\sqrt{5}} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{2}{\sqrt{5}} & -\frac{\sqrt{5}}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{\sqrt{3}}{2} & -\frac{2}{\sqrt{3}} \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{\sqrt{2}}{\sqrt{3}} \end{bmatrix}$$

$$A_{\mathbb{E}_8} = \begin{bmatrix} 2 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

$$R_{\mathbb{E}_8} = \begin{bmatrix} \sqrt{2} & 0 & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & \sqrt{2} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\sqrt{3}}{\sqrt{2}} & -\frac{\sqrt{2}}{\sqrt{3}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\sqrt{5}}{\sqrt{6}} & -\frac{\sqrt{6}}{\sqrt{5}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{2}{\sqrt{5}} & -\frac{\sqrt{5}}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{\sqrt{3}}{2} & -\frac{2}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{\sqrt{2}}{\sqrt{3}} & -\frac{\sqrt{3}}{\sqrt{2}} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix}$$

Todavía no hemos acabado la demostración del lema 1.7. Falta demostrar que las gráficas de Dynkin son las únicas gráficas de aristas sólidas que definen formas unitarias positivas. Comenzamos con el siguiente lema, el cual muestra que toda gráfica que tenga aristas múltiples no corresponde a ninguna forma unitaria positiva.

LEMA 1.8. *Si la forma unitaria $q(\vec{x})$ es positiva entonces $q_{ij} \in \{-1, 0, 1\}$ para todo $1 \leq i < j \leq n$.*

Demostración. Denotemos con $\vec{e}_k = (d_1, d_2, \dots, d_n)$ al vector dado por $d_k = 1$ y $d_i = 0$ para toda $i \neq k$. Si $q_{ij} \geq 2$ entonces $q(\vec{e}_i - \vec{e}_j) = 2 - q_{ij} \leq 0$ pero $\vec{e}_i - \vec{e}_j \neq \vec{0}$. Si $q_{ij} \leq -2$ entonces $q(\vec{e}_i + \vec{e}_j) = 2 - q_{ij} \leq 0$ pero $\vec{e}_i + \vec{e}_j \neq \vec{0}$. De lo anterior, como q es definida positiva, entonces $|q_{ij}| < 2$ para todo $1 \leq i < j \leq n$. ■

Cada $|q_{ij}|$ nos dice la cantidad de aristas que hay entre los vértices x_i y x_j de la gráfica \mathbf{B}_q ; por tanto si $|q_{ij}| \leq 1$ tenemos que \mathbf{B}_q es una gráfica simple (v. pag. 80). Es decir que el lema anterior se puede reescribir como sigue:

COROLARIO 1.9. *Si q es una forma unitaria positiva entonces necesariamente \mathbf{B}_q es una gráfica simple.*

Con base en este lema diremos que una forma unitaria q es **simple** si su gráfica asociada \mathbf{B}_q es una gráfica simple.

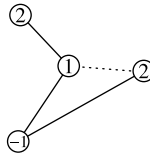
Ahora que hemos descartado a las gráficas de aristas múltiples (a las pseudográficas que tienen lazos sencillamente no las consideramos para este trabajo) el resto de la demostración es como sigue: primero demostraremos que toda gráfica que contenga a una gráfica Euclidiana (figura 1.3) no define una forma unitaria positiva; segundo, que toda gráfica

conexa que no sea de Dynkin necesariamente contiene una subgráfica Euclidiana.

Notación	Gráfica
\widetilde{A}_m	
\widetilde{D}_m	
\widetilde{E}_6	
\widetilde{E}_7	
\widetilde{E}_8	

FIGURA 1.3: Gráficas Euclidianas. La cantidad de vértices que tiene cada gráfica es $n = m + 1$.

Hasta ahora hemos asociado vértices con variables (por ejemplo \textcircled{x} se interpreta como la variable x), pero vamos a extender esta convención de la siguiente manera: si en lugar de etiquetar con variables etiquetamos con números reales (por ejemplo $\textcircled{2}$), convenimos que se está evaluando la variable correspondiente en dicho número real. Por ejemplo, consideremos a la forma q de la ecuación (1.5); entonces la gráfica



representa la evaluación $x = 2, y = 1, z = 2, w = -1$, es decir

$$\begin{aligned}
 q(2, 1, 2, -1) &= 2^2 + 1^2 + 2^2 + (-1)^2 - 2 \cdot 1 + 1 \cdot 2 - 1 \cdot (-1) - 2 \cdot (-1) \\
 &= 13
 \end{aligned}$$

Usando esta notación mostraremos que si \mathbf{B}_q contiene una subgráfica Euclidiana entonces existe un vector $\vec{x} \neq \vec{0}$ tal que $q(\vec{x}) = 0$, mostrando así que $q(\vec{x})$ no es definida positiva. En efecto, este vector se construye de la siguiente manera: se evalúan todas las variables que corresponden a la subgráfica Euclidiana de acuerdo a la figura 1.4 y todas las

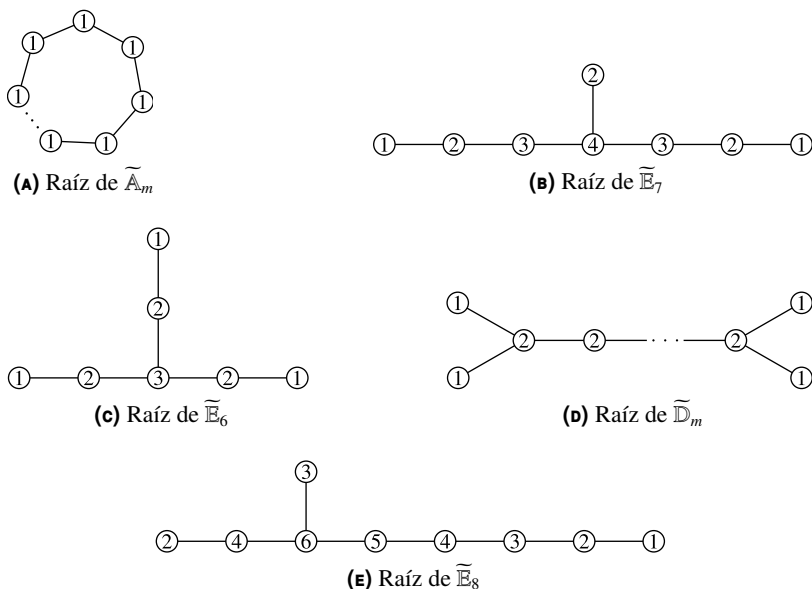


FIGURA 1.4: Raíces de las formas unitarias asociadas a las gráficas Euclidianas.

demás variables se evalúan a cero. Un simple cálculo nos muestra que esta evaluación produce un vector $\vec{x} \neq \vec{0}$ tal que $q(\vec{x}) = 0$.

Solamente falta demostrar que toda gráfica que no sea de Dynkin necesariamente contiene a una subgráfica Euclidiana. Sea G es una gráfica conexa de n vértices diferente de A_n , D_n , E_6 , E_7 y E_8 . Si G no es un árbol entonces G contiene un ciclo; es decir, contiene una subgráfica \tilde{A}_m para algún $m < n$. Si G es un árbol, y dado que $G \neq A_n$, entonces existe al menos un vértice v de grado 3 o más. Claramente v pertenece a una subgráfica D_r para algún $r \leq n$, pero habíamos supuesto que $G \neq D_n$ por lo tanto hay tres casos: si v tiene grado estrictamente mayor a 3 entonces G contiene a \tilde{D}_4 ; si G contiene otro vértice w de grado 3 o más, entonces G contiene a \tilde{D}_m para algún $m < n$; y si todos los demás vértices de G tienen grado menor a 3 entonces G debe contener a E_6 como subgráfica. Ahora bien, habíamos supuesto que $G \neq E_6$, por tanto $n > 6$. En este caso G debe contener a \tilde{E}_6 o E_7 . Si G contiene a $E_7 \neq G$ entonces $n > 7$, de donde G contiene a \tilde{E}_7 o E_8 , pero si G contiene a $E_8 \neq G$ entonces $n > 8$ y por lo tanto G contiene a \tilde{E}_8 .

Resumiendo, las gráficas de Dynkin definen formas unitarias positivas, y cualquier otra gráfica conexa y de aristas sólidas que no sea de Dynkin necesariamente contiene una gráfica Euclidiana que la vuelve

no positiva; por lo tanto las gráficas de Dynkin son las únicas gráficas conexas de aristas sólidas que definen formas unitarias positivas. Esto concluye la demostración del lema 1.7.

En la sección 2.1 se dará término a la demostración del teorema 1.6.

1.3. EL PROBLEMA QUE SE ABORDA EN ESTA TESIS

Es útil determinar cuando una forma unitaria es de tipo Dynkin \mathbb{A}_n en diversas áreas de las matemáticas como teoría de las álgebras de Lie, teoría de representaciones de álgebras y teoría de politopos. Este problema ha sido abordado previamente con enfoques completamente teóricos; en este trabajo el problema se aborda desde el punto de vista computacional. Un investigador de estas áreas podría beneficiarse de un programa de computadora eficiente que resuelva este problema. Formalmente se plantea el siguiente problema de decisión:

Instancia Una gráfica \mathbf{B}_q , con aristas posiblemente punteadas

Pregunta ¿ \mathbf{B}_q define una forma unitaria q de tipo Dynkin \mathbb{A}_n ?

El resto de este trabajo aborda el problema como sigue:

El capítulo 2 muestra los antecedentes a este trabajo. El primer antecedente se encuentra en la demostración del teorema 1.6, ya que al ser una demostración constructiva incluye de manera implícita un algoritmo que determina el tipo Dynkin de cualquier forma unitaria positiva. En este trabajo expone dicho algoritmo de forma explícita. El segundo antecedente es un teorema que permite construir cualquier forma unitaria de tipo \mathbb{A}_n ; aquí se reinterpreta dicho teorema usando terminología tomada de la teoría de gráficas y este será el antecedente principal de la tesis.

El capítulo 3 es un repaso del algoritmo de recorrido en profundidad y una adaptación que determina las componentes biconexas de una gráfica. Esta será la herramienta principal para analizar y atacar el problema.

El capítulo 4 presenta la relación existente entre el algoritmo de recorrido en profundidad y la caracterización mencionada en el capítulo 2. De esta manera será posible adaptar el recorrido en profundidad para obtener una caracterización computacionalmente eficiente de las formas unitarias tipo \mathbb{A}_n .

El capítulo 5 muestra cómo se puede implementar dicha caracterización de manera eficiente. Se hará un análisis de complejidad y se darán algunas conclusiones.

CAPÍTULO 2

ANTECEDENTES

Feliz es el hombre quien puede comprender
por qué suceden las cosas.

(Virgilio)

Este capítulo está dedicado a presentar los resultados principales de Gabriel & Roĭter (1997) y Barot (1999) que permiten caracterizar a las formas \mathbb{A}_n .

2.1. EL MÉTODO DE LAS INFLACIONES

La demostración del teorema 1.6 aparece en Gabriel & Roĭter (1997) y hace uso implícito de un algoritmo que describiremos en esta sección. La idea intuitiva es pasar mediante cambios de variable enteros de la forma unitaria q a la forma unitaria q' donde $\mathbf{B}_{q'}$ no contiene aristas punteadas. Si q es definida positiva entonces por lema 1.3 q' también es definida positiva; luego como q' solamente contiene aristas sólidas entonces por lema 1.7 se sigue que cada una de sus componentes conexas debe ser una gráfica de Dynkin (figura 1.2). El procedimiento para encontrar dicha forma cuadrática q' es el *método de las inflaciones*, y es el tema de esta sección.

INFLACIONES Y DEFLACIONES

En la **página 97** se discuten las matrices elementales E_{rs}^d , cuya transformación lineal

$$T_{rs}^d(A) = E_{rs}^d A$$

equivale a sumar d veces el renglón s de A al renglón r de A . Dado que la matriz inversa de E_{rs}^d es E_{rs}^{-d} , notamos que E_{rs}^d es \mathbb{Z} -invertible si y solamente si d es un número entero.

Consideremos al cambio de variable entero $\vec{y} = E_{rs}^d \vec{x}$, entonces

$$\begin{bmatrix} y_1 \\ \vdots \\ y_r \\ \vdots \\ y_n \end{bmatrix} = E_{rs}^d \begin{bmatrix} x_1 \\ \vdots \\ x_r \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_r + d x_s \\ \vdots \\ x_n \end{bmatrix}$$

y por tanto podemos concluir que T_{rs}^d como cambio de variable tiene el mismo efecto que sustituir $x_r \leftarrow x_r + d x_s$.

¿Qué efecto tiene T_{rs}^d sobre una forma unitaria q ? Supongamos que q tiene matriz asociada $A = [a_{ij}]$ y sea

$$q'(\vec{x}) = \frac{1}{2} (E_{rs}^d)^T A E_{rs}^d$$

Dado que E_{rs}^d es la matriz identidad salvo por d en la posición (r, s) -ésima, entonces $(E_{rs}^d)^T = E_{rs}^d$. Por lo tanto $(E_{rs}^d)^T$ tiene el efecto de sumar d veces el renglón r al renglón s . Por otra parte E_{rs}^d multiplicado por la derecha tiene el efecto de sumar d veces la columna r a la columna s , por lo tanto la matriz $C = (E_{rs}^d)^T A E_{rs}^d$ está dada por

$$c_{ij} = \begin{cases} a_{ij} & \text{si } i \neq s \text{ y } j \neq s \\ a_{sj} + d a_{rj} & \text{si } i = s \text{ pero } j \neq s \\ a_{is} + d a_{ir} & \text{si } j = s \text{ pero } i \neq s \\ d^2 a_{rr} + d(a_{sr} + a_{rs}) + a_{ss} & \text{si } i = s \text{ y } j = s \end{cases} \quad (2.1)$$

E_{rs}^d define un cambio de variable entero, por lo que C es la matriz asociada a alguna forma cuadrática entera, aunque no necesariamente unitaria. Para que $q'(\vec{x}) = \frac{1}{2} \vec{x}^T C \vec{x}$ sea una forma unitaria es necesario que $c_{ss} = 2$, o lo que es lo mismo (con $a_{rr} = a_{ss} = 2$ y $a_{rs} = a_{sr}$):

$$2d^2 + 2d a_{rs} + 2 = 2.$$

Esta ecuación tiene dos soluciones: $d = 0$ y $d = -a_{rs}$. Estas soluciones se resumen en el siguiente lema:

LEMA 2.1. Si q es una forma unitaria entonces $q \circ T_{rs}^d$ es una forma unitaria si y solo si $q_{rs} = q_{sr} = -d$.

COROLARIO 2.2. Si q es una forma unitaria simple entonces $q \circ T_{rs}^d$ es una forma unitaria para un $d \neq 0$ en cualquiera de estos dos casos y solamente en estos dos:

1. $d = 1$ y \mathbf{B}_q contiene una arista punteada entre los vértices x_r y x_s
2. $d = -1$ y \mathbf{B}_q contiene una arista sólida entre los vértices x_r y x_s

Este corolario no garantiza que $q \circ T_{rs}^d$ sea simple; en este caso por el corolario 1.9 vemos que esto ocurre solamente cuando q (que es \mathbb{Z} -equivalente a $q \circ T_{rs}^d$) no es definida positiva.

Con base en el corolario anterior se justifica que definamos la transformación de **inflación** como

$$T_{rs}^- = T_{rs}^{-1}$$

y la transformación de **deflación** como

$$T_{rs}^+ = T_{rs}^1.$$

El corolario anterior dice que la inflación T_{rs}^- se puede aplicar solamente cuando \mathbf{B}_q contiene la arista punteada $x_r \cdots x_s$, mientras que T_{rs}^+ solamente cuando \mathbf{B}_q contiene la arista sólida $x_r \text{---} x_s$. A partir de la ecuación (2.1), y del hecho de que la matriz es simétrica, Podemos interpretar las inflaciones y deflaciones como un algoritmo de reconexión sobre la gráfica \mathbf{B}_q (bajo las condiciones del corolario anterior) que consiste de cuatro etapas:

1. *Duplicar*: Generar una copia de las aristas que inciden en x_i excepto la arista que conecta x_i con x_j .
2. *Invertir*: En caso de una inflación reemplazar $x_i \cdots x_j$ por $x_i \text{---} x_j$ y para cada arista duplicada del paso anterior intercambiar las aristas punteadas con sólidas y viceversa. En caso de una deflación simplemente reemplazar $x_i \text{---} x_j$ por $x_i \cdots x_j$.
3. *Arrastrar*: Cada una de las aristas duplicadas se desconecta de su extremo x_i y se reconecta en x_j .
4. *Simplificar*: Si existen dos aristas $x_r \cdots x_j$ y $x_r \text{---} x_j$ incidentes en x_r y x_j , se borran.

Por ejemplo, consideremos la forma cuadrática q de la ecuación (1.5) y ordenemos a las variables como (x, y, z, w) . Entonces para aplicar T_{23}^{-1} podemos calcular $(E_{23}^{-1})^T A E_{23}^{-1}$ donde A es la matriz asociada a q .

$$\left| \begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 2 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 2 & 1 & -1 \\ 0 & -1 & 1 & 0 & 0 & 1 & 2 & -1 \\ 0 & 0 & 0 & 1 & 0 & -1 & -1 & 2 \end{array} \right| \left| \begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 2 & -1 & 1 & 0 \\ 0 & 1 & -1 & 0 & -1 & 2 & -1 & -1 \\ 0 & 0 & 1 & 0 & 1 & -1 & 2 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 & 2 \end{array} \right| = \left| \begin{array}{cccc|cccc} 2 & -1 & 1 & 0 & 2 & -1 & 1 & 0 \\ -1 & 2 & -1 & -1 & 1 & -1 & 2 & 0 \\ 1 & -1 & 2 & 0 & 0 & -1 & 0 & 2 \\ 0 & -1 & 0 & 2 & 0 & -1 & 0 & 2 \end{array} \right|$$

La matriz resultante nos describe la misma forma cuadrática que hubiéramos obtenido reemplazando $y \leftarrow y - z$; en efecto:

$$\begin{aligned} x^2 + (y - z)^2 + z^2 + w^2 - x(y - z) + (y - z)z - (y - z)w - zw \\ = x^2 + y^2 + z^2 + w^2 - xy + xz - yz - yw \end{aligned}$$

También podemos hacer el cambio de variable de manera gráfica tal como se ilustra en la figura 2.1.

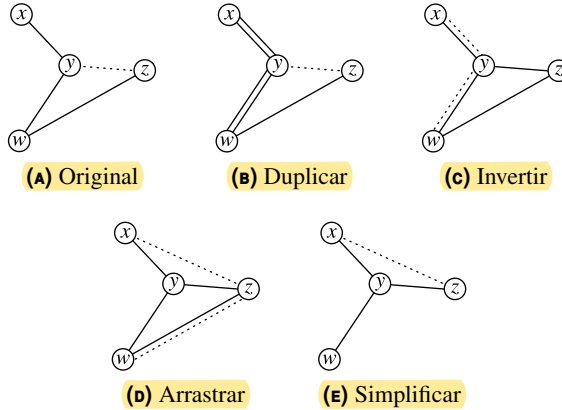


FIGURA 2.1: Ejemplo de una inflación.

DESCRIPCIÓN Y JUSTIFICACIÓN

El método que aparece en Gabriel & Roïter (1997) consiste en aplicar sucesivamente inflaciones como se indica en el algoritmo 2.1 tanto como sea posible, es decir, hasta que no haya más aristas punteadas.

ALGORITMO 2.1: INFLACIONES (q)

```

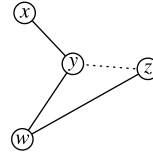
1 mientras exista una arista punteada  $x_r \cdots x_s$  con  $r < s$  en  $\mathbf{B}_q$  :
2    $q \leftarrow q \circ T_{rs}^-$ 
3   si para algún coeficiente  $q_{ij}$  se tiene  $|q_{ij}| > 1$  :
4     responder no es positiva
5 para cada componente conexa  $C$  de  $\mathbf{B}_q$  :
6    $n \leftarrow$  número de vértices que contiene  $C$ 
7   si  $C$  no es isomorfa a  $\mathbb{A}_n, \mathbb{D}_n, \mathbb{E}_6, \mathbb{E}_7$  o  $\mathbb{E}_8$  :
8     responder no es positiva
9 responder es positiva y es de tipo Dynkin  $\mathbf{B}_q$ 

```

Veamos cómo funciona este algoritmo sobre la forma cuadrática de la ecuación (1.5):

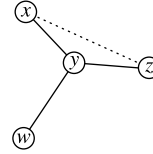
1. Comenzamos con la forma unitaria

$$x^2 + y^2 + z^2 + w^2 - xy + yz - yw - zw$$



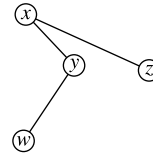
2. Notamos la arista $y \cdots z$ y aplicamos T_{23}^- para obtener

$$x^2 + y^2 + z^2 + w^2 - xy + xz - yz - wy$$



3. Notamos la arista $x \cdots z$ y aplicamos T_{13}^- para obtener

$$x^2 + y^2 + z^2 + w^2 - xy - xz - wy$$



4. Esta gráfica es isomorfa a \mathbb{A}_4 , por lo tanto este es el tipo Dynkin de la forma cuadrática (1.5).

Para terminar la demostración del teorema 1.6 es necesario mostrar que si q es una forma unitaria positiva entonces este algoritmo termina. El resto de esta sección está dedicada a ello. Podemos resumir lo

que resta de la demostración del teorema 1.6 de la siguiente manera (a continuación veremos los detalles):

- Decimos que $\vec{x} \in \mathbb{Z}^n$ es una **raíz** de la forma unitaria q si $q(\vec{x}) = 1$.
- Primero mostramos que toda forma cuadrática se puede poner en la forma

$$q(\vec{x}) = \lambda_1 y_1^2 + \lambda_2 y_2^2 + \cdots + \lambda_n y_n^2$$

$$y_i = x_i + \sum_{j=i+1}^n \rho_{ij} x_j$$

- Vamos a suponer que q es una forma unitaria positiva, y usando la expresión anterior mostraremos que q tiene una cantidad finita de raíces. Más específicamente mostraremos que cada x_i está acotada en el intervalo $[-\sigma_i, \sigma_i]$ para alguna constante σ_i .
- Encontraremos cierto subconjunto de raíces que crece con cada iteración del algoritmo 2.1. Como el conjunto de raíces es finito, la cantidad de iteraciones también.

La **reducción de Lagrange** es un método que nos permite llevar toda forma cuadrática a la forma (1.4). Primero necesitamos recordar la fórmula de **completar cuadrados**:

$$\begin{aligned} ax^2 + bx + c &= a \left[x^2 + \frac{b}{a}x \right] + c \\ &= a \left[x^2 + \frac{b}{a}x + \left(\frac{b}{2a} \right)^2 - \left(\frac{b}{2a} \right)^2 \right] + c \\ &= a \left(x + \frac{b}{2a} \right)^2 + \left(c - \frac{b^2}{4a} \right) \end{aligned} \quad (2.2)$$

Sea q una forma unitaria sobre las variables x_k, x_{k+1}, \dots, x_n . Para escribir $q(x_k, \dots, x_n)$ en la forma (1.4) hacemos lo siguiente:

1. Si $q = q_{nn}x_n^2$ es una forma cuadrática en una variable entonces se definen $\lambda_n = q_{nn}$, $y_n = x_n$ y terminamos (en caso contrario continuamos).
2. Sea $\ell = k + 1$. Apartamos todos los términos que contienen la variable x_k :

$$q = q_{kk}x_k^2 + \sum_{j=\ell}^n q_{kj}x_kx_j + r$$

donde $r = \sum_{i=\ell}^n q_{ii} x_i^2 + \sum_{j=\ell+1}^n \sum_{i=\ell}^{j-1} q_{ij} x_i x_j$.

3. Factorizamos $q_{k\ell} x_k$:

$$q = q_{kk} x_k^2 + q_{k\ell} x_k \left(\sum_{j=\ell}^n \frac{q_{kj}}{q_{k\ell}} x_j \right) + r$$

4. Completamos el cuadrado sustituyendo en la ecuación (2.2) los valores de $a = q_{kk}$, $b = q_{k\ell} \left(\sum_{j=\ell}^n \frac{q_{kj}}{q_{k\ell}} x_j \right)$ y $c = 0$:

$$q = q_{kk} \left(x + \sum_{j=\ell}^n \frac{q_{kj}}{2 q_{k\ell}} x_j \right)^2 - \frac{q_{k\ell}^2}{4 q_{kk}} \left(\sum_{j=\ell}^n \frac{q_{kj}}{q_{k\ell}} x_j \right)^2 + r$$

5. Se definen

$$\begin{aligned} \lambda_k &= q_{kk} \\ y_k &= x_k + \sum_{j=\ell}^n \frac{q_{kj}}{2 q_{k\ell}} x_j \\ q' &= -\frac{q_{k\ell}^2}{4 q_{kk}} \left(\sum_{j=\ell}^n \frac{q_{kj}}{q_{k\ell}} x_j \right)^2 + r \end{aligned}$$

Desarrollando el cuadrado en q' notamos que q' es una forma cuadrática sobre las variables $x_\ell, x_{\ell+1}, \dots, x_n$ (una variable menos que q) y además

$$q = \lambda_1 y_1 + q'$$

6. Repetimos el mismo procedimiento para q' .

Este método es en sí mismo una demostración constructiva (e inductiva) del siguiente lema:

LEMA 2.3. *Toda forma cuadrática $q(\vec{x})$ se puede reescribir como*

$$\begin{aligned} q(\vec{x}) &= \lambda_1 y_1^2 + \lambda_2 y_2^2 + \dots + \lambda_n y_n^2 \\ y_i &= x_i + \sum_{j=i+1}^n \rho_{ij} x_j \end{aligned} \tag{2.3}$$

Veamos un ejemplo. Sea $q(x_1, x_2, x_3) = x_1^2 + 2x_2^2 - 7x_3^2 - 4x_1x_2 + 8x_1x_3$. La primera iteración hace lo siguiente:

$$\begin{aligned}
 q(x_1, x_2, x_3) &= x_1^2 - 4x_1x_2 + 8x_1x_3 + \underbrace{2x_2^2 - 7x_3^2}_r \\
 &= \underbrace{1}_a x_1^2 + x_1 \underbrace{(-4)(x_2 - 2x_3)}_b + \underbrace{2x_2^2 - 7x_3^2}_r \\
 &= \underbrace{1}_{\lambda_1} \underbrace{(x_1 - 2x_2 + 4x_3)^2}_{y_1} \underbrace{-4(x_2 - 2x_3)^2 + 2x_2^2 - 7x_3^2}_{q'} \\
 &= \lambda_1 y_1^2 - \underbrace{2x_2^2 - 23x_3^2 + 16x_2x_3}_{q'}
 \end{aligned}$$

La segunda iteración funciona sobre $q'(x_2, x_3) = -2x_2^2 - 23x_3^2 + 16x_2x_3$:

$$\begin{aligned}
 q'(x_2, x_3) &= \underbrace{-2}_a x_2^2 + x_2 \underbrace{(16x_3)}_b - \underbrace{23x_3^2}_{r'} \\
 &= \underbrace{-2}_{\lambda_2} \underbrace{(x_2 - 4x_3)^2}_{y_2} + \underbrace{9x_3^2}_{q''} \\
 &= \lambda_2 y_2 + \underbrace{9x_3^2}_{q''}
 \end{aligned}$$

Finalmente en la tercera iteración opera sobre $q''(x_3) = 9x_3^2$, que es una forma cuadrática en una sola variable, por tanto $\lambda_3 = 9$ y $y_3 = x_3$. Así hemos reescrito a q como

$$q(x_1, x_2, x_3) = y_1^2 - 2y_2^2 + 9y_3^2$$

donde

$$\begin{aligned}
 y_1 &= x_1 - 2x_2 + 4x_3 \\
 y_2 &= x_2 - 4x_3 \\
 y_3 &= x_3
 \end{aligned}$$

Supongamos que q es una forma unitaria positiva y que \vec{x} es una raíz de q . Usando la reducción de Lagrange escribimos

$$\begin{aligned}
 q(\vec{x}) &= \lambda_1 y_1^2 + \lambda_2 y_2^2 + \cdots + \lambda_n y_n^2 \\
 y_i &= x_i + \sum_{j=i+1}^n \rho_{ij} x_j
 \end{aligned}$$

Como q es positiva entonces cada $\lambda_i > 0$. Más aún, como $\lambda_1 y_1^2 + \lambda_2 y_2^2 + \dots + \lambda_n y_n^2 = 1$ (porque habíamos supuesto que \vec{x} es una raíz) entonces $\lambda_i y_i^2 \leq 1$ para cada i .

Como $y_n = x_n$ entonces $\lambda_n x_n^2 \leq 1$, de donde $|x_n| \leq \frac{1}{\sqrt{\lambda_n}}$. Definiendo $\sigma_n = \frac{1}{\sqrt{\lambda_n}}$ hemos mostrado que $-\sigma_n \leq x_n \leq \sigma_n$.

A continuación mostraremos que si x_j está acotada por σ_j para toda j desde $k+1$ hasta n , entonces también x_k está acotada por alguna constante σ_k . En efecto, tenemos que

$$\lambda_k y_k^2 = \lambda_k \left(x_k + \sum_{j=k+1}^n \rho_{kj} x_j \right)^2 \leq 1$$

de donde

$$\left| x_k + \sum_{j=k+1}^n \rho_{kj} x_j \right| \leq \frac{1}{\sqrt{\lambda_k}}.$$

Para continuar necesitamos de la siguiente desigualdad:

$$\left| \sum_{i=1}^m a_i \right| \geq |a_1| - \sum_{i=2}^m |a_i|. \quad (2.4)$$

para cualesquiera números reales a_1, a_2, \dots, a_m . Esta desigualdad se demuestra fácilmente por inducción (el caso base $m = 2$ establece que $|a_1 + a_2| \geq |a_1| - |a_2|$ y se comprueba por casos; para el paso inductivo escribimos $|\sum_{i=1}^m a_i| \geq |\sum_{i=1}^{m-1} a_i| - |a_m|$ y aplicamos la hipótesis de inducción). Usando la desigualdad (2.4) deducimos que

$$|x_k| - \sum_{j=k+1}^n |\rho_{kj} x_j| \leq \left| x_k + \sum_{j=k+1}^n \rho_{kj} x_j \right| \leq \frac{1}{\sqrt{\lambda_k}}$$

de donde, despejando $|x_k|$ y usando que $x_j \leq \sigma_j$ con $\sigma_j > 0$, obtenemos

$$|x_k| \leq \frac{1}{\sqrt{\lambda_k}} + \sum_{j=k+1}^n |\rho_{kj} x_j| \leq \frac{1}{\sqrt{\lambda_k}} + \sum_{j=k+1}^n |\rho_{kj}| \sigma_j.$$

Por lo tanto $\sigma_k = \frac{1}{\sqrt{\lambda_k}} + \sum_{j=k+1}^n |\rho_{kj}| \sigma_j$ es una constante tal que $-\sigma_k \leq x_k \leq \sigma_k$. Por inducción concluimos que si \vec{x} es una raíz de q entonces todo x_i está acotado entre $-\sigma_i$ y σ_i para alguna constante σ_i . Como cada x_i debe ser un número entero entre $-\sigma_i$ y σ_i entonces concluimos el siguiente lema:

LEMA 2.4. Si q es una forma unitaria positiva entonces su conjunto de raíces es finito.

En lo que sigue denotemos por $S(q)$ al conjunto (finito) de raíces de q , y por $R(q)$ al conjunto de raíces no negativas de q , es decir:

$$\begin{aligned} S(q) &= \{\vec{x} \in \mathbb{Z}^n \mid q(\vec{x}) = 1\} \\ R(q) &= \{\vec{x} \in \mathbb{N}^n \mid q(\vec{x}) = 1\} \end{aligned}$$

Claramente $R(q) \subseteq S(q)$.

Supongamos que q es una forma unitaria positiva. Entonces \mathbf{B}_q es una gráfica simple y el algoritmo 2.1 busca valores de r y s tales que existe una arista punteada $x_r \cdots x_s$. Esto es lo mismo que decir que $q_{rs} = 1$ (ver corolario 1.9). Sea $q' = q \circ T_{rs}^-$. Como la transformación $T_{rs}^- : \mathbb{Z}^n \rightarrow \mathbb{Z}^n$ está definida por una matriz \mathbb{Z} -invertible entonces T_{rs}^- es invertible y su inversa es T_{rs}^+ .

Supongamos que \vec{x} es una raíz de q , entonces

$$q'(T_{rs}^+(\vec{x})) = q(T_{rs}^-(T_{rs}^+(\vec{x}))) = q(\vec{x}) = 1$$

y por tanto $T_{rs}^+(\vec{x})$ es una raíz de q' . Esto nos dice que de hecho T_{rs}^+ es una función biyectiva entre los conjuntos $S(q)$ y $S(q')$, e inyectiva en los conjuntos $R(q)$ y $R(q')$. De aquí concluimos que $|S(q)| = |S(q')|$ y que $|R(q)| \leq |R(q')|$; a continuación mostraremos que esta desigualdad es estricta.

Denotamos con $\vec{e}_k = (d_1, d_2, \dots, d_n)$ al vector tal que $d_k = 1$ y $d_i = 0$ para todo $i \neq k$. Sea $\vec{z} = \vec{e}_s - \vec{e}_r$. Notamos que

$$q(\vec{z}) = (-1)^2 + 1^2 + q_{rs}(-1)(1) = 1$$

por lo que $\vec{z} \in S(q)$ pero $\vec{z} \notin R(q)$ (porque tiene un -1 en la entrada r -ésima). Sin embargo $T_{rs}^+(\vec{z}) = \vec{e}_s$, es decir que $\vec{z} \in R(q')$. Como $\vec{z} \in R(q')$ pero $\vec{z} \notin R(q)$, entonces $|R(q)| < |R(q')|$.

Finalmente, si denotamos por q_1, q_2, q_3, \dots a la sucesión de valores que toma la variable q del algoritmo 2.1 entonces de lo anterior tenemos que

$$\begin{aligned} |S(q_1)| &= |S(q_2)| = |S(q_3)| = \dots \\ |R(q_1)| &< |R(q_2)| < |R(q_3)| < \dots \end{aligned}$$

Si estas sucesiones fuesen infinitas entonces eventualmente contradiríamos que $R(q_i) \subseteq S(q_i)$, por lo tanto el algoritmo 2.1 solamente puede

hacer un número finito de iteraciones del ciclo **mientras**. Una breve inspección al pseudocódigo nos muestra que esto ocurre cuando \mathbf{B}_q ya no contiene aristas punteadas, por lo tanto al término de este ciclo \mathbf{B}_q contiene solamente aristas sólidas. Como habíamos supuesto que q es positiva y como cada q_i es \mathbb{Z} -equivalente a q , entonces el lema 1.7 nos garantiza que cada componente conexa de \mathbf{B}_q es una gráfica de Dynkin.

Con esto se concluye la demostración del teorema 1.6.

2.2. ENSAMBLAJE POR \mathbb{A} -BLOQUES

VISIÓN GENERAL DEL ENSAMBLE

En Barot (1999) se aborda el problema de caracterizar las formas de tipo \mathbb{A}_n ; pero en lugar de basarse en transformaciones, se basa en la construcción de las gráficas Δ tales que q_Δ es de tipo \mathbb{A}_n . La idea es ir “pegando” ciertas gráficas como si fueran bloques de construcción. La estructura de estos bloques es muy sencilla: Se parte de dos conjuntos de vértices V_0 y V_1 ; entre los vértices (u, v) hay una arista punteada $u \cdots v$ si ambos $u, v \in V_0$ o ambos $u, v \in V_1$, y una arista sólida $u \text{---} v$ en otro caso. A las gráficas así obtenidas les llamaremos **\mathbb{A} -bloques** y las denotaremos por $\mathbf{F}_{m,m'}$ donde $m = |V_0|$ y $m' = |V_1|$ (pero acostumbramos a escribir $m \geq m'$). La figura 2.3 muestra cómo dibujar un \mathbb{A} -bloque.

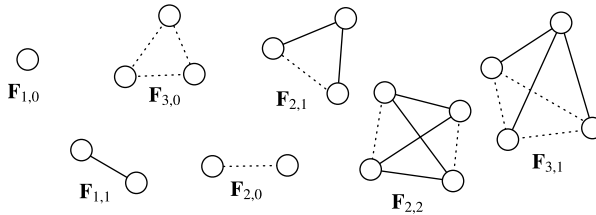


FIGURA 2.2: Algunos ejemplos de \mathbb{A} -bloques.

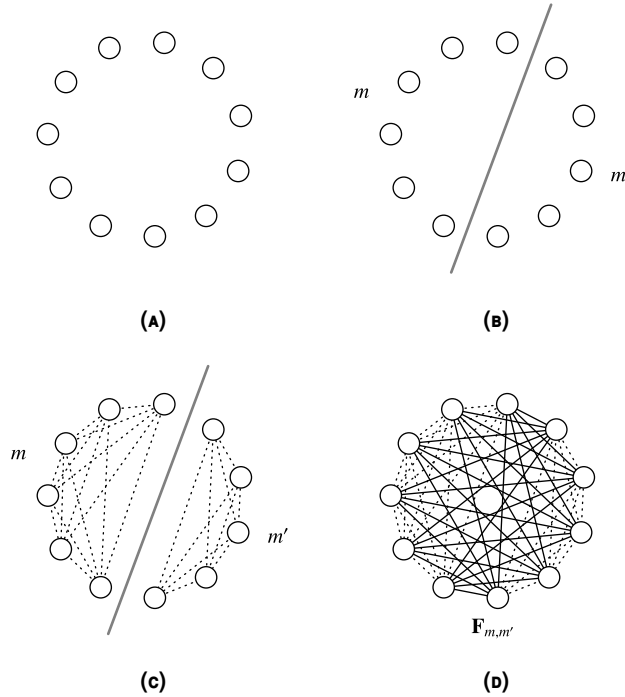
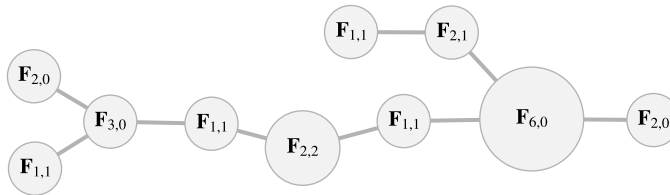


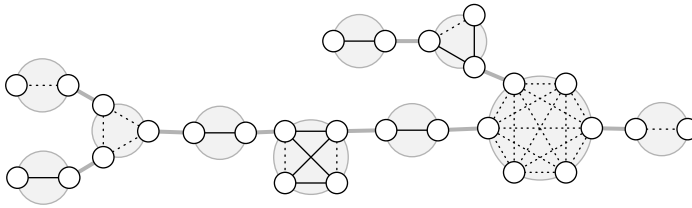
FIGURA 2.3: Cómo se construye un \mathbb{A} -bloque

El resultado principal de Barot (1999) es que todas las formas unitarias de tipo \mathbb{A}_n se *construyen* pegando \mathbb{A} -bloques sobre una gráfica de árbol como se explica a continuación:

1. Se comienza con un árbol T de vértices $\{1, 2, \dots, t\}$ y a cada vértice $i \in T$ se le asocia un \mathbb{A} -bloque \mathcal{B}_i . **Ejemplo:**



2. A cada arista $i-j \in T$ se le asocian los vértices $\sigma_i(i-j) \in \mathcal{B}_i$ y $\sigma_j(i-j) \in \mathcal{B}_j$ de manera que cada función σ_k sea inyectiva.



3. Para cada arista $i—j \in T$ se identifican los vértices $\sigma_i(i—j) \in \mathcal{B}_i$ y $\sigma_j(i—j) \in \mathcal{B}_j$, es decir, se “pegan” para volverse uno solo. Continuando con este ejemplo obtenemos la figura 2.4

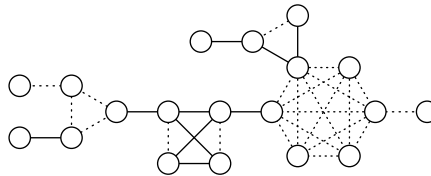


FIGURA 2.4: Esta gráfica define a una forma de tipo \mathbb{A}_{19}

LOS \mathbb{A} -BLOQUES

A continuación trataremos de justificar el ensamble de \mathbb{A} -bloques a partir del siguiente resultado tomado de Barot & de la Peña (1999):

TEOREMA 2.5. *Si q_G es de tipo \mathbb{A}_n , y si H es una subgráfica conexa inducida por m vértices de G , entonces q_H es de tipo \mathbb{A}_m .*

Comencemos con un caso especial de gráficas: las **gráficas circulares** son aquellas en las que cada vértice está conectado con exactamente otros dos vértices. Nos interesa saber cuáles gráficas circulares definen formas cuadráticas de tipo \mathbb{A}_n . Ciertamente $\mathbf{F}_{3,0}$ y $\mathbf{F}_{2,1}$ son unas de estas (lo podemos comprobar usando el algoritmo 2.1) pero ¿hay más?

Trataremos de simplificar un poco el problema. Las gráficas circulares de la figura 2.5 definen formas \mathbb{Z} -equivalentes: podemos pasar de una a otra mediante las matrices elementales E_i^{-1} de tamaño n (v. pág. 97). No es difícil comprobar que estas matrices tienen el efecto de intercambiar aristas sólidas con punteadas siempre que estas incidan en el vértice x_i . De hecho, este mismo razonamiento muestra que toda gráfica circular es equivalente a otra gráfica circular “estandarizada”, que consiste de aristas sólidas y posiblemente una sola arista punteada.

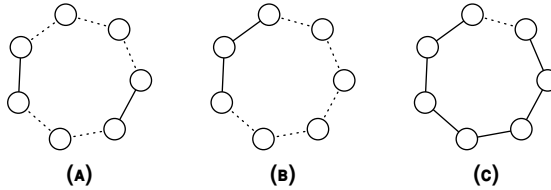
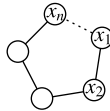


FIGURA 2.5: Ejemplos de gráficas circulares

Podemos descartar a las gráficas que no tienen ninguna arista punteada por que son de tipo $\widetilde{\mathbb{A}}_m$, es decir, no son positivas. Denotemos con $C(n)$ a la gráfica circular de n vértices que tiene exactamente una arista punteada $x_n \cdots x_1$ y las demás sólidas:



Con el algoritmo 2.1 podemos comprobar directamente que $C(4)$ tiene tipo Dynkin \mathbb{D}_4 . Si $n > 4$, aplicando la inflación T_{n-1}^- y borrando el vértice x_n de $C(n)$ se obtiene $C(n-1)$, de modo que por inducción y el teorema 2.5, todas las gráficas $C(n)$ con $n > 3$ no pueden ser de tipo \mathbb{A}_n . Hemos mostrado el siguiente lema:

LEMA 2.6. Las únicas gráficas circulares G tales que q_G es de tipo \mathbb{A}_n son $\mathbb{F}_{3,0}$ y $\mathbb{F}_{2,1}$.

Una gráfica es **k -conexa** si no existe ningún subconjunto de $k-1$ vértices tales que su borrado desconecte a la gráfica. Dicho de otro modo, si κ denota la cantidad mínima de vértices que es necesario borrar para obtener una gráfica desconexa, entonces $\kappa \geq k$. Por ejemplo, toda gráfica conexa es automáticamente 1-conexa y toda gráfica completa K_n (v. pag. 82) es k -conexa para todo $k \in \{1, 2, \dots, n\}$. De hecho, por definición tenemos que si G es una gráfica k -conexa entonces también es $(k-1)$ -conexa, $(k-2)$ -conexa, etc. Otro ejemplo: las gráficas circulares son gráficas **biconexas** (2-conexas). Ahora supongamos que \mathbf{B}_q es biconexa y que q es de tipo \mathbb{A}_n ; mostraremos que \mathbf{B}_q es un \mathbb{A} -bloque.

Primero mostraremos que \mathbf{B}_q debe ser una gráfica completa. Supongamos que no es así y sea S el conjunto de vértices más pequeño tal que al borrarlos de \mathbf{B}_q la gráfica se hace desconexa y defínase $\kappa = |S|$. Sabemos que \mathbf{B}_q es 2-conexa, y además que la ausencia de la arista $u-v$

(porque \mathbf{B}_q no es completa) nos permite obtener una gráfica disconexa borrando todos los vértices excepto u y v ; por lo tanto tenemos que $2 \leq \kappa \leq n - 2$. Seleccionamos cualesquiera $\kappa - 2$ vértices del conjunto S y los borramos de la gráfica \mathbf{B}_q para obtener otra gráfica \mathbf{B}'_q . Por la definición de S sabemos que \mathbf{B}'_q es una gráfica biconexa pero no **trico-nexa** (3-conexa). Es decir, en \mathbf{B}'_q existen dos vértices x e y tales que al borrarlos de \mathbf{B}'_q se obtiene una gráfica de dos componentes conexas \mathcal{B}_0 y \mathcal{B}_1 . Consideremos el camino más corto $x \rightsquigarrow y$ que inicia en x pasa solamente por vértices de \mathcal{B}_0 y termina en y , juntemos este camino con el camino más corto $y \rightsquigarrow x$ que inicia en y , pasa solamente por vértices de \mathcal{B}_1 y termina en x . Esta unión forma un ciclo $x \rightsquigarrow y \rightsquigarrow x$, pero habíamos supuesto que q es de tipo \mathbb{A}_n , por tanto el lema 2.6 nos dice que \mathbf{B}'_q contiene la siguiente subgráfica inducida (sin tomar en cuenta si las aristas son sólidas o punteadas):



Aquí $z \in \mathcal{B}_0$, $w \in \mathcal{B}_1$ y no existe la arista $z \text{---} w$ (en otro caso \mathbf{B}_q seguiría conectada después de quitar los vértices x e y). Otra vez mediante el uso del algoritmo 2.1 se puede demostrar que no importa cuáles aristas sean sólidas o punteadas, esta gráfica no es de tipo \mathbb{A}_n , en contradicción con el teorema 2.5; por lo tanto tenemos:

LEMA 2.7. Si $q : \mathbb{Z}^n \rightarrow \mathbb{Z}$ es una forma unitaria de tipo \mathbb{A}_n y si \mathbf{B}_q es una gráfica biconexa, entonces \mathbf{B}_q es una gráfica completa.

El siguiente lema se puede leer *entre líneas* en el artículo mencionado; aquí solamente se está recalcando su importancia porque haremos referencia a este lema en capítulos posteriores:

LEMA 2.8. Si toda subgráfica de \mathbf{B}_q inducida por tres vértices es $\mathbf{F}_{3,0}$ o $\mathbf{F}_{2,1}$, entonces \mathbf{B}_q es un \mathbb{A} -bloque.

Demostración. Fijemos un vértice u y definamos los conjuntos

$$\begin{aligned} V_0 &= \{u\} \cup \{v \mid \text{existe una arista punteada } u \text{-----} v\} \\ V_1 &= \{v \mid \text{existe una arista sólida } u \text{---} v\} \end{aligned}$$

Considérese a otros dos diferentes vértices v y w en \mathbf{B}_q . Por hipótesis tenemos que los vértices u , v y w inducen una subgráfica $\mathbf{F}_{3,0}$ o $\mathbf{F}_{2,1}$. Por la definición de V_0 y V_1 concluimos que si dos vértices v, w están en el mismo V_i entonces hay una arista punteada $v \text{-----} w$; y en otro caso

(están en conjuntos diferentes) hay una arista sólida $v \text{---} w$; por lo tanto \mathbf{B}_q es un \mathbb{A} -bloque. ■

Vamos a recapitular lo que hemos visto hasta ahora:

1. El lema 2.7 nos dice que si q es de tipo \mathbb{A}_n y si \mathbf{B}_q es biconexa, entonces \mathbf{B}_q es, de hecho, completa.
2. Pero por el teorema 2.5 cada subgráfica inducida de \mathbf{B}_q debe definir otra forma de tipo \mathbb{A}_n .
3. En particular, como \mathbf{B}_q es completa, el lema 2.6 nos dice que toda subgráfica inducida por cada tres vértices de \mathbf{B}_q es $\mathbf{F}_{3,0}$ o $\mathbf{F}_{2,1}$.
4. Entonces, por lema 2.8 concluimos que \mathbf{B}_q es un \mathbb{A} -bloque.

Ahora vamos a mostrar el recíproco: que todo \mathbb{A} -bloque es biconexo y que define una forma de tipo \mathbb{A}_n . La biconexidad es obvia puesto que todo \mathbb{A} -bloque $\mathbf{F}_{m,m'}$ es una gráfica completa. Ahora bien, renombraremos a los vértices en V_0 como x_1, x_2, \dots, x_m y a los de V_1 como $x_{m+1}, x_{m+2}, \dots, x_n$. Podemos “deshilar” a \mathbf{B}_q en dos etapas: en la primera quitamos todas las aristas punteadas que hay entre los vértices de V_1 usando las inflaciones T_{i+1}^- en orden $i = n-1, n-2, \dots, m+1$ y en la segunda las punteadas que hay en V_0 usando las inflaciones T_{i+1}^- en orden $i = 1, 2, \dots, m-1$.

Para mostrar que este método de “deshilado” funciona, supongamos que $m > 0$ y $m' > 0$ y sean $V_0 = \{x_1, \dots, x_m\}$ y $V_1 = \{x_{m+1}, \dots, x_n\}$ los conjuntos que definen a $\mathbf{F}_{m,m'}$. Sea x_r incidente en x_{n-1} .

- Si $x_r = x_n$ entonces la arista $x_{n-1} \cdots x_n$ se sustituye por $x_{n-1} \text{---} x_n$.
- Para todos los x_r tales que exista una arista punteada $x_r \cdots x_{n-1}$ se tiene que $x_r \in V_1$ y el arrastre forma una arista sólida $x_r \text{---} x_n$ que se cancela con la arista $x_r \cdots x_n$; por lo tanto luego de aplicar T_{n-1}^- en el vértice x_n no incide ninguna arista punteada.
- Para todos los x_r tales que exista una arista sólida $x_r \text{---} x_{n-1}$ se tiene que $x_r \in V_0$ y el arrastre forma una arista punteada $x_r \cdots x_n$ que se cancela con la arista $x_r \text{---} x_n$; por lo tanto luego de aplicar T_{n-1}^- en el vértice x_n no incide ninguna arista sólida.

De lo anterior concluimos que al aplicar T_{n-1}^- a $\mathbf{F}_{m,m'}$ se obtiene $\mathbf{F}_{m,m'-1}$ unida con una arista sólida $x_{n-1} \text{---} x_n$; luego entonces por inducción se

sigue la sucesión de inflaciones $(T_{i+1}^-)_{i=n-1}^{m+1}$ sobre la gráfica $\mathbf{F}_{m,m'}$ produce la gráfica $\mathbf{F}_{m,1}$ unida con $x_{m+1} \cdots x_n$. Un razonamiento similar muestra que la sucesión de inflaciones $(T_{i+1}^-)_{i=1}^{m-1}$ aplicadas a $\mathbf{F}_{m,1}$ produce $x_1 \cdots x_m$. Por lo tanto el método descrito transforma $\mathbf{F}_{m,m'}$ en $\mathbb{A}_{m+m'}$. Ver ejemplo en figura 2.6.

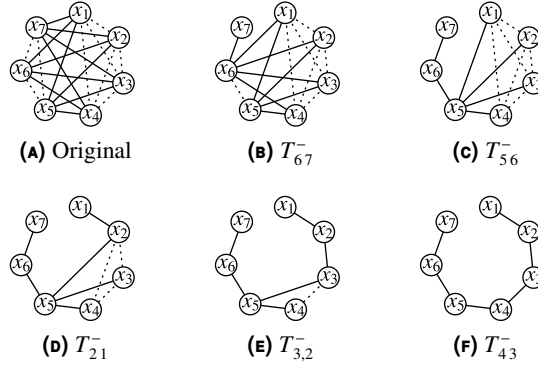


FIGURA 2.6: Deshilando a $\mathbf{F}_{4,3}$

Hemos demostrado el siguiente

LEMA 2.9. *q es una forma unitaria de tipo \mathbb{A}_n con \mathbf{B}_q biconexa si y solamente si \mathbf{B}_q es un \mathbb{A} -bloque.*

ENSAMBLAJE DE FORMAS \mathbb{A}_n

Ahora sí tenemos las herramientas necesarias para justificar el ensamble de \mathbb{A} -bloques. Primero veamos que todo ensamble de \mathbb{A} -bloques en verdad define una forma unitaria de tipo \mathbb{A}_n . Aquí vamos a demostrarlo de manera un poco diferente a la del artículo.

Supongamos que T es el árbol que subyace en un pegado de \mathbb{A} -bloques. Si este árbol tiene un solo vértice entonces ya acabamos por el lema 2.9 recién mostrado. En otro caso, como T es un árbol, debe haber un vértice t de grado 1 (podría decirse que t es una hoja del árbol). Entonces el \mathbb{A} -bloque asociado al vértice t , que es $\mathcal{B}_t = \mathbf{F}_{m,m'}$, comparte exactamente un vértice v con algún otro \mathbb{A} -bloque \mathcal{B}_s (figuras 2.7a y 2.7b). Ahora apliquemos la técnica deshilado explicada justo antes del lema 2.9, pero renombrando al vértice v como $x_{m+m'}$, de manera que este deshilado no afecta en absoluto a ningún otro vértice de \mathcal{B}_s . Lo que veremos (como en la figura 2.7c) es que ahora del vértice v “cuelga” la

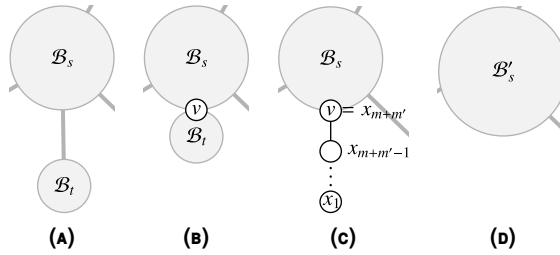


FIGURA 2.7: Esquemática de la fusión de dos \mathbb{A} -bloques.

siguiente gráfica:

$$x_{m+m'} \text{---} x_{m+m'-1} \text{---} \cdots \text{---} x_3 \text{---} x_2 \text{---} x_1.$$

Esto se parece a lo que se obtiene durante los pasos intermedios para deshilar un \mathbb{A} -bloque; esto sugiere hacer el proceso inverso, es decir, utilizar las deflaciones (v. pág. 19).

Podemos usar las deflaciones para “tejer” los vértices de \mathcal{B}_t en \mathcal{B}_s aplicando sucesivamente $T_{i+1,i}$ en orden $i = m + m' - 1, m + m' - 2, \dots, 3, 2, 1$ (porque precisamente este es el proceso inverso al de deshilarlo). Cada vez que hacemos esto estamos, en cierto sentido, agregando x_i a \mathcal{B}_s ; de manera que al término de estas deflaciones tendremos todos los vértices de \mathcal{B}_s y \mathcal{B}_t en un mismo \mathbb{A} -bloque \mathcal{B}'_s . Así hemos mostrado cómo fusionar dos \mathbb{A} -bloques del árbol en uno solo, repitiendo este proceso podemos fusionarlos todos, mostrando así que todo ensamble de \mathbb{A} -bloques es equivalente a un solo \mathbb{A} -bloque. Damos ejemplo de esta fusión en la figura 2.8.

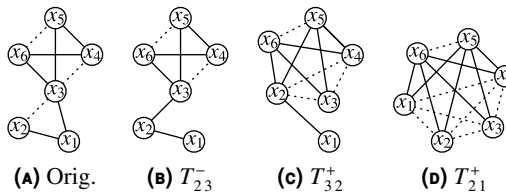


FIGURA 2.8: Ejemplo de fusión de dos \mathbb{A} -bloques

Hemos mostrado el siguiente:

LEMA 2.10. Si G es una gráfica construida por un ensamble de árbol de \mathbb{A} -bloques, entonces q_G es de tipo \mathbb{A}_n .

Falta mostrar el converso: que si G es una forma unitaria con tipo Dynkin \mathbb{A}_n entonces se puede construir mediante un ensamble de árbol de \mathbb{A} -bloques. Una **componente biconexa** de una gráfica, es una subgráfica biconexa maximal (no contenida propiamente en ninguna otra subgráfica biconexa). La manera más fácil de entender a las componentes biconexas es mediante los **puntos de articulación**, que son los vértices de la gráfica que al quitarlos la deja desconectada.

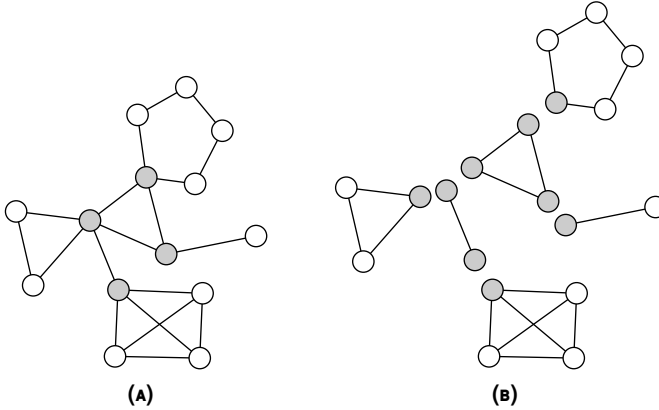


FIGURA 2.9: Una gráfica, sus puntos de articulación y componentes biconexas.

En la figura 2.9 se muestra en color gris los puntos de articulación de una gráfica y la misma gráfica separada en sus componentes biconexas.

LEMA 2.11. *Las componentes biconexas de una gráfica particionan al conjunto de aristas.*

Demostración. Cada arista es por sí misma una subgráfica biconexa, y por lo tanto pertenece a una subgráfica biconexa maximal; por otro lado ninguna arista puede pertenecer a dos componentes biconexas, por que si este fuera el caso entonces podríamos pegar ambas componentes por medio de la arista que tienen en común, mostrando así que no eran maximales. ■

Del teorema 2.5 y del lema 2.9 se concluye que si q es de tipo \mathbb{A}_n entonces las componentes biconexas de \mathbf{B}_q son \mathbb{A} -bloques. Supongamos que $\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_t$ son las componentes biconexas de \mathbf{B}_q , y formemos una nueva gráfica T de vértices $\{1, 2, \dots, t\}$, donde cada arista $i-j$ representa que \mathcal{B}_i y \mathcal{B}_j , con $i \neq j$, tienen un vértice en común (este debe

ser un punto de articulación de \mathbf{B}_q). Más aún, si \mathcal{B}_i y \mathcal{B}_j comparten el vértice v definiremos $\sigma_i(i \text{---} j) = \sigma_j(i \text{---} j) = v$.

Debido a la manera en que construimos las funciones σ_k y la gráfica T , tenemos que las siguientes tres afirmaciones son equivalentes:

1. Cada función σ_k es inyectiva
2. T es un árbol
3. Ningún punto de articulación pertenece a tres o más componentes biconexas

Por lo tanto basta mostrar cualquiera de estas afirmaciones. Mostraremos la tercera afirmación por inducción. Si \mathbf{B}_q no tiene puntos de articulación, por vacuidad ningún punto de articulación pertenece a tres o más componentes biconexas. Ahora supongamos que \mathbf{B}_q tiene al menos un punto de articulación v ; entonces $\mathbf{B}_q - v$ tiene componentes conexas C_1, C_2, \dots, C_ℓ . Defínase \mathcal{D}_i como la subgráfica de \mathbf{B}_q inducida por los vértices $V(C_i) \cup \{v\}$. Por teorema 2.5 cada q_{C_i} es de tipo \mathbb{A}_{n_i} y por hipótesis de inducción, cada C_i se construye haciendo un ensamble de árbol de \mathbb{A} -bloques (ningún C_i tiene puntos de articulación que pertenezcan a tres o más componentes biconexas). Ahora, mediante el proceso previamente descrito, deshilemos a cada C_i para convertirlo en una gráfica \mathbb{A}_{n_i} que tenga a v como extremo. Luego entonces v es el centro de una gráfica de estrella de ℓ picos y sin aristas punteadas; pero habíamos supuesto que \mathbf{B}_q es de tipo \mathbb{A}_n , de manera que el algoritmo 2.1 nos dice que $\ell = 2$. Por lo tanto v , que es un punto de articulación cualquiera, solamente puede pertenecer a exactamente dos componentes biconexas. Así tenemos el teorema principal del artículo:

TEOREMA 2.12. *Una forma unitaria q es de tipo \mathbb{A}_n si y solamente si \mathbf{B}_q se construye mediante un ensamble de árbol de \mathbb{A} -bloques.*

Sin embargo, para los fines de este trabajo más conveniente reinterpretar este resultado utilizando otras palabras (pero la demostración es la misma):

COROLARIO 2.13. *Una forma unitaria q es de tipo \mathbb{A}_n si y solo si \mathbf{B}_q es conexa, sus componentes biconexas son \mathbb{A} -bloques y todo punto de articulación pertenece a exactamente dos de estas componentes.*

DESCOMPOSICIÓN DE GRÁFICAS

Dentro de cada gran programa bien escrito
hay un pequeño programa bien escrito.

(Charles Hoare)

En el capítulo anterior mostramos la importancia de determinar las componentes biconexas y puntos de articulación en una gráfica. Aquí se presenta un famoso y eficiente algoritmo que lo hace.

3.1. EL ALGORITMO DE RECORRIDO EN PROFUNDIDAD

El tema de esta sección puede consultarse con mayor detalle en el capítulo Cormen *et al.* (2009).

REPRESENTACIÓN DE GRÁFICAS

Una gráfica se puede representar por medio de una **matriz de adyacencia**; si hay n vértices v_1, v_2, \dots, v_n formamos una matriz de tamaño $n \times n$ tal que su entrada (i, j) -ésima es

$$a_{ij} = \begin{cases} 1 & \text{si hay una arista de } v_i \text{ a } v_j \\ 0 & \text{en otro caso.} \end{cases}$$

Para gráficas no dirigidas (que son las que consideramos aquí) esta matriz es simétrica por que una arista se puede tomar en ambas direcciones.

Esta representación tiene la conveniencia de que la presencia de una arista determinada puede ser verificada en tiempo constante (es decir,

mediante un algoritmo de complejidad $\Theta(1)$), usando solo un acceso a memoria. Por otro lado, esta matriz ocupa $\Theta(n^2)$ de espacio, lo cual representa un gran desperdicio para gráficas que son poco densas.

Una representación alternativa es la **representación por listas de adyacencia**, que consiste de $|V|$ listas (una lista es una sucesión finita) donde la lista asociada al vértice v contiene los nombres de todos los vértices que son adyacentes a v . De esta forma, si $V = \{v_1, v_2, \dots, v_n\}$ tenemos que su representación es (L_1, L_2, \dots, L_n) donde

$$L_i = \{j \mid v_j \text{ es adyacente a } v_i\}$$

Así, la lista de adyacencia tiene tamaño $\Theta(|V| + |E|)$. Cada arista $u-v$ aparece dos veces en esta representación: una como u en la lista de v y otra como v en la lista de u .

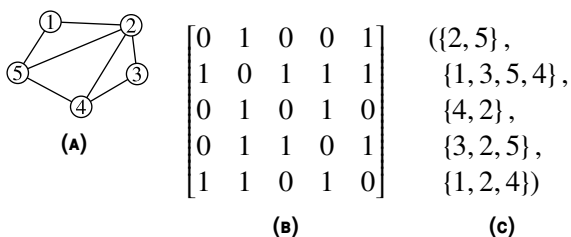


FIGURA 3.1: Una gráfica, su matriz y lista de adyacencia

En esta representación no se puede verificar la existencia de una arista en particular $u-v$ en un tiempo constante porque se necesita atravesar toda la lista de u . Sin embargo en esta representación es muy fácil iterar sobre los vecinos de un vértice (recorriendo la lista correspondiente). Esta operación resulta útil para varios algoritmos de gráficas, incluyendo el que veremos en esta sección.

DESCRIPCIÓN DEL RECORRIDO

Dada una gráfica G y un vértice u , quisiéramos saber cuáles son los vértices que se pueden alcanzar desde u , es decir, todos los vértices v tales que existe un camino $u \rightsquigarrow v$.

Explorar una gráfica es como explorar un laberinto: hay que visitar un conjunto de salones (vértices) unidos por pasadizos (aristas), pero es muy fácil perderse y dar vueltas sobre el mismo lugar. Para evitarlo, ponemos una marca en cada salón conforme lo vamos visitando y

usamos un cordón atado a la entrada para poder regresar siempre a ella. La estrategia del recorrido es adentrarse tanto como sea posible sin regresar al mismo lugar. Computacionalmente hablando, marcamos los vértices con FALSO si no han sido visitados y con VERDADERO si ya fueron visitados. Para simular la cuerda le asociamos a cada vértice v el vértice $\pi(v)$ desde donde se llegó por primera vez. Sin embargo una simulación implícita, pero más elegante de la cuerda consiste en utilizar recursión; por conveniencia usaremos una combinación de ambas estrategias en el algoritmo 3.1.

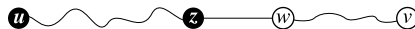
ALGORITMO 3.1: VISITAR (G, u)

```

1 visitado ( $u$ )  $\leftarrow$  VERDADERO
2 PREVISITAR ( $u$ )
3 para cada vértice  $v$  adyacente a  $u$  :
4   si no visitado ( $v$ ) :
5      $\pi(v) \leftarrow u$ 
6     VISITAR ( $G, v$ )
7 POSVISITAR ( $u$ )
```

PREVISITAR y POSVISITAR son métodos que todavía no han sido definidos, aquí podríamos poner cualquier cosa que queramos hacer con un vértice antes y después de visitarlo respectivamente. Siguiendo a Cormen *et al.* (2009), por el momento supongamos que PREVISITAR colorea al vértice de gris y POSVISITAR lo colorea de negro (suponemos que al principio los vértices son de color blanco y que ninguno tiene la marca de haber sido visitado).

Veamos como funciona el recorrido. El algoritmo va saltando de vértice en vértice usando solo las aristas de la gráfica, así que solamente puede marcar como visitado a los vértices v para los que exista un camino $u \rightsquigarrow v$. ¿Pero realmente marca a *todos*? Supongamos que esto no es así. Entonces tendríamos un camino de u a v donde, digamos, un vértice z en este camino fue el último en ser visitado pero w ya no.



Ahora bien, nuestro algoritmo claramente indica que después de marcar como visitado a un vértice se tienen que visitar a todos los vértices adyacentes que todavía no hayan sido visitados, por lo tanto después de visitar a z se tuvo que visitar a w ¡una contradicción!

Ahora que ya sabemos que VISITAR explora la parte de la gráfica alcanzable desde un punto inicial, podemos reiniciar el método en otro vértice que haya quedado sin visitar hasta explorar toda la gráfica. Esta es la idea detrás del recorrido en profundidad (algoritmo 3.2).

ALGORITMO 3.2: RECORRER-EN-PROFUNDIDAD (G)

```

1 para cada vértice  $v$  :
2    $visitado(v) \leftarrow \text{FALSO}$ 
3    $\pi(v) \leftarrow \text{NINGUNO}$ 
4 para cada vértice  $r$  :
5   si no  $visitado(r)$  : VISITAR( $G, r$ )
  
```

Esta técnica se
conoce como
análisis
amortizado.

¿Cuánto tiempo tarda un recorrido en profundidad cuando la entrada es una gráfica $G = (V, E)$? En lugar de analizar paso a paso usando complicadas relaciones de recurrencia, notemos que primero se ocupan $\Theta(|V|)$ pasos para marcar a los vértices como no visitados y luego, a lo largo de todo el recorrido, cada vértice se visita exactamente una vez (gracias a que los vamos marcando). Cada vez que se visita un vértice se requiere una cantidad fija de pasos para marcarlo, previsitarlo y pos-visitarlo, así que estas instrucciones ocupan un total de $\Theta(|V|)$ pasos. Las instrucciones para recorrer los vértices adyacentes hacen que a lo largo de todo el recorrido cada arista $u \rightarrow v$ se considere exactamente dos veces: una durante VISITA(u) y otra durante VISITA(v); o sea que estas instrucciones tardan $\Theta(|E|)$ pasos. Juntándolo todo, el tiempo total de ejecución es $\Theta(|V| + |E|)$.

UNA VERSIÓN ITERATIVA

Podemos aprovecharnos de que $\pi(u)$ apunta al vértice desde el cuál se llegó a u por primera vez para obtener la versión iterativa del recorrido en profundidad (algoritmo 3.3). Solamente hay que ser cuidadosos en cuanto a la forma que el algoritmo determina si hay un vértice v adyacente al vértice actual u que todavía no haya sido visitado, por que no queremos recorrer la misma lista de adyacencia una y otra vez. Para ello hacemos que el algoritmo “recuerde” la posición donde se quedó la última vez que recorrió la lista de adyacencia de cada vértice u . Veremos estos detalles en un capítulo posterior.

ALGORITMO 3.3: RECORRER-EN-PROFUNDIDAD (G)

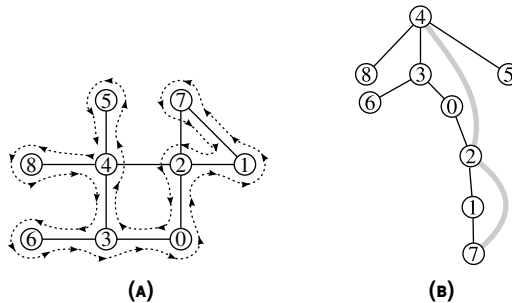
```

1 para cada vértice  $v$  :  $visitado(v) \leftarrow$  FALSO
2 para cada vértice  $r$  que todavía no haya sido visitado :
3    $visitado(r) \leftarrow$  VERDADERO
4    $\pi(r) \leftarrow$  NINGUNO
5   PREVISITAR( $r$ )
6    $u \leftarrow r$ 
7   mientras  $u \neq$  NINGUNO :
8     mientras haya un vértice  $v$  adyacente  $u$  sin visitar :
9        $visitado(v) \leftarrow$  VERDADERO
10       $\pi(v) \leftarrow u$ 
11      PREVISITAR( $v$ )
12       $u \leftarrow v$ 
13    POSVISITAR( $u$ )
14     $u \leftarrow \pi(u)$ 

```

ÁRBOLES RECUBRIDORES EN PROFUNDIDAD

Sea G es una gráfica decimos que H es un **bosque de recubrimiento** para G si H una subgráfica de G tal que $V(H) = V(G)$ y H es un bosque. En particular cuando H es un árbol decimos que es un **árbol de recubrimiento** para G . El recorrido en profundidad aplicado a una gráfica G forma un bosque de recubrimiento porque nunca se visita el mismo vértice dos veces, es decir que el recorrido nunca forma un ciclo. Si T es un árbol descubierto por el recorrido en profundidad, diremos que es un **árbol de recubrimiento en profundidad**.

**FIGURA 3.2:** Recorrido en profundidad sobre una gráfica.

Si $\text{PREVISITAR}(u)$ y $\text{POSVISITAR}(u)$ se definen como **escribir** (“(”, u) y **escribir** (u , “)”) respectivamente, entonces el recorrido de la figura 3.2 arroja, en 18 tiempos, el siguiente resultado:

$$\begin{array}{cccccccccccccccc} (& 4 & (& 8 & 8 &) & (& 3 & (& 6 & 6 &) & (& 0 & (& 2 & (& 1 & (& 7 & 7 &) & 1 &) & 2 &) & 0 &) & 3 &) & (& 5 & 5 &) & 4 &) \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 \end{array}$$

Note que los paréntesis están bien balanceados. Esto es algo natural del recorrido, por que si se visita el vértice u y luego v , entonces, o bien v se visitó mientras u se estaba visitando o bien después de haber terminado la visita de u . En el primer caso tenemos $(u \cdots (v \cdots v) \cdots u)$ y en el segundo $(u \cdots u) \cdots (v \cdots v)$. Denotamos con $d(v)$ al tiempo en que v fue **descubierto** (previsitado) por el algoritmo, y con $f(v)$ el tiempo en que fue **finalizado** (posvisitado). En nuestro ejemplo tenemos $d(3) = 4$ y $f(3) = 15$. De lo anterior podemos concluir el siguiente

TEOREMA 3.1 (de los paréntesis). *Sean u y v dos vértices de una gráfica G y sea T un árbol de recubrimiento en profundidad. Entonces se cumple una y solamente una de las siguientes afirmaciones:*

- a) *El intervalo $[d(v), f(v)]$ está contenido en $[d(u), f(u)]$ y v es un descendiente de u*
- b) *El intervalo $[d(u), f(u)]$ está contenido en $[d(v), f(v)]$ y v es un antecesor de u*
- c) *Los intervalos $[d(u), f(u)]$ y $[d(v), f(v)]$ son disjuntos y los vértices u y v pertenecen a ramas distintas de T*

COROLARIO 3.2. *En un árbol de recubrimiento en profundidad el vértice v es un descendiente de u si y solo si $d(u) < d(v) < f(v) < f(u)$.*

Si el recorrido descubre los vértices en el orden v_1, v_2, \dots, v_n , decimos que esta sucesión de vértices el **recorrido en preorden** del árbol de recubrimiento en profundidad. El **recorrido en postorden** se define de manera análoga bajo el orden en que se van finalizando los vértices.

Los árboles de recubrimiento en profundidad separan a las aristas de la gráfica en dos tipos: las que pertenecen al árbol y las que no. Las aristas que no forman parte del árbol tienen la propiedad de que siempre conectan a un vértice con un ancestro, veamos por qué: dada una arista $u-v$, supóngase sin pérdida de generalidad que u se descubrió antes que v , entonces, o bien v se descubrió mientras se visitaba algún descendiente de u o bien se visitó a v mediante esta arista; en cualquiera de los dos casos v descende de u . Estas aristas aparecen en la figura

3.2 de color gris. De lo anterior podemos decir que un árbol clasifica la aristas en **aristas de árbol** y **aristas de retroceso**, por que solo pueden conectar a un vértice con un antecesor suyo. Resumiendo:

LEMA 3.3. *Las aristas que no pertenecen a un árbol de recubrimiento en profundidad son aristas de retroceso.*

3.2. DESCOMPOSICIÓN AUTOMÁTICA DE GRÁFICAS

En esta sección se describe uno de los resultados más importantes de Hopcroft & Tarjan (1973).

En la página 35 se explicó qué son los puntos de articulación y las componentes biconexas de una gráfica. Veremos ahora cómo se puede aprovechar un árbol de recubrimiento en profundidad para su detección.

CARACTERIZANDO PUNTOS DE ARTICULACIÓN

El lema 3.3 implica que las aristas de una gráfica nunca cruzan de una rama a otra; esto tiene dos consecuencias importantes. En primera, si la raíz r de un árbol de recubrimiento en profundidad tiene al menos dos hijos c_1 y c_2 , entonces al borrar a r se separan c_1 y c_2 junto con sus descendientes, por lo que r es un punto de articulación. En segunda, si consideremos a cualquier otro vértice de ramificación x y a un hijo suyo c ; y si c ni ninguno de sus descendientes está conectado mediante una arista de retroceso a un ancestro de x , entonces al borrar a x se separan los ancestros de x de los descendientes de c (incluyendo a c), y por lo tanto x es un punto de articulación.

¿Cómo determinamos si c (hijo de x) o un descendiente de c está conectado mediante una arista de retroceso a un antecesor de x ? Si etiquetamos a cada vértice u con el ancestro más remoto que es adyacente a u o a uno de sus descendientes la pregunta se reduce a decidir si c está etiquetado con un ancestro de x . En este caso, si c está etiquetado con el ancestro y de x entonces $d(y) < d(x)$, así que, de hecho, nos debemos fijar únicamente en los tiempos de descubrimiento. Formalmente definimos

$$\ell(u) = \min \{d(v) \mid v \text{ es } u, \text{ adyacente a } u \text{ o a un descendiente de } u\}$$

Con esta definición lo anterior se traduce en que $\ell(c) = d(x)$ si y solo si el antecesor más remoto conectado con c o alguno de sus descendientes

es precisamente x ; dicho de otra forma, ni c ni sus descendientes tienen una arista de retroceso hacia un antecesor de x . Por lo tanto

TEOREMA 3.4. *Luego de un recorrido en profundidad, un vértice x es de articulación en estos dos casos y solamente en estos dos:*

1. x es la raíz del árbol y tiene al menos dos hijos;
2. x es un vértice de ramificación y tiene un hijo c con $\ell(c) = d(x)$.

USO DEL RECORRIDO EN PROFUNDIDAD

Nuestra tarea se ha reducido a calcular $\ell(u)$ para todo vértice u . Como ℓ depende de los descendientes de u , podemos calcular ℓ en dirección de las hojas hacia la raíz. Así, para cuando intentemos calcular $\ell(u)$ ya tendremos calculados los valores correspondientes a sus hijos y

$$\ell(u) = \min \begin{cases} d(u) \\ d(v) & \text{tal que } v \text{ es adyacente a } u; \\ \ell(w) & \text{tal que } w \text{ es hijo de } u. \end{cases}$$

Calcular ℓ de las hojas hacia la raíz tiene otra ventaja: en cuanto descubramos el primer par de padre x e hijo c tal que $\ell(c) = x$, tendremos la certeza de que no hay ningún punto de articulación en la descendencia de c (de otro modo, este punto de articulación hubiera sido descubierto antes); pero al no haber otros puntos de articulación, x , c , y los descendientes de c inducen una componente biconexa. Una vez que hemos determinado una componente biconexa podemos borrar (o ignorar) esta parte de la gráfica y continuar de la misma manera hasta encontrar todas las componentes biconexas.

ALGUNOS DETALLES DE IMPLEMENTACIÓN

A continuación trataremos de dar unos cuantos detalles de implementación que son interesantes. Para ello necesitamos comprender qué es una *pila* en la jerga computacional. Pensemos en un restaurante donde los platos sucios se colocan uno encima del otro conforme van llegando. Cuando se desea lavar un plato es necesario retirarlo desde la cima para evitar que el resto se caiga. En efecto, los platos sucios se van *apilando* conforme llegan y *desapilando* conforme son lavados. Así pues, una pila es el más claro ejemplo de la frase “los últimos serán los primeros”.

Formalmente entendemos por **pila** como un conjunto dinámico p que soporta las siguientes operaciones:

- CIMA: es el elemento más recientemente agregado a p
- APILAR: agregar un elemento a p , el cual pasa a ser la cima
- DESAPILAR: devuelve y borra el elemento cima de p , la cima pasa a ser el elemento que fue apilado más recientemente

¿QUÉ ES UN CONJUNTO DINÁMICO? En matemáticas los conjuntos son *estáticos*: una vez que son definidos ya no es posible alterar sus contenidos sin obtener una contradicción. Un *conjunto dinámico* es un objeto computacional que representa un conjunto y define operaciones que alteran su contenido. Algunas operaciones comunes son búsqueda, inserción, borrado, reemplazo y extracción, pero también hay otras más particulares como extraer el elemento que minimiza una función. *Estructuras de datos* es una rama de las ciencias computacionales dedicada al estudio de los conjuntos dinámicos: se desea que la representación de un conjunto dinámico ocupe poco espacio extra (además del espacio que ocupan sus elementos) y realizar las operaciones eficientemente.

Por definición, $d(u)$ y $f(u)$ se determinan durante $\text{PREVISITAR}(u)$ y $\text{POSVISITAR}(u)$ respectivamente. Usaremos una variable t (de tiempo) que se incrementa en estos eventos. Como $\text{POSVISITAR}(u)$ se manda a llamar desde las hojas hacia la raíz, es buen momento para calcular $\ell(u)$. El teorema 3.4 se traduce en que hay que buscar una componente biconexa inducida por $\pi(u)$, u , y los descendientes de u , cada vez que $\ell(u) = d(\pi(u))$. Por lema 2.11 solo debemos encontrar las aristas de la componente. Para ello, durante $\text{POSVISITAR}(u)$ agregamos a una pila las aristas $v \rightarrow u$ donde v es un antecesor de u (así evitamos listar dos veces la misma arista) y para cuando llegue el momento de extraer las aristas de una componente biconexa, estas estarán en la cima de B .

Así es como hemos obtenido al algoritmo 3.4 que determina las componentes biconexas de una gráfica G haciendo un recorrido en profundidad donde PREVISITAR y POSVISITAR se definen según los algoritmos 3.5 y 3.6 respectivamente.

Para encontrar los puntos de articulación adaptamos el algoritmo según el teorema 3.4. ¿Cuánto tardamos en encontrar las componentes biconexas? El recorrido en profundidad requiere $\Theta(|V| + |E|)$ pasos, pero ahora el tiempo que nos tardamos en POSVISITAR no es constante. A lo largo de todo el algoritmo cada vértice se previsa, visita y posvisita exactamente una vez, aportando $\Theta(|V|)$. La lista de adyacencia de

ALGORITMO 3.4: BICONEXAS (G)

```

1  $t \leftarrow 0$ 
2  $B \leftarrow$  pila vacía
3 RECORRER-EN-PROFUNDIDAD ( $G$ )

```

ALGORITMO 3.5: PREVISITAR (u)

```

1  $d(u) \leftarrow t$ 
2  $t \leftarrow t + 1$ 

```

ALGORITMO 3.6: POSVISITAR (u)

```

1  $f(u) \leftarrow t$ 
2  $t \leftarrow t + 1$ 
3  $\ell(u) \leftarrow d(u)$ 
4 para cada vértice  $v$  adyacente a  $u$  :
5   si  $d(v) < \ell(u)$  :  $\ell(u) \leftarrow d(v)$ 
6   si  $\pi(v) = u$  y  $\ell(v) < \ell(u)$  :  $\ell(u) \leftarrow \ell(v)$ 
7   si  $d(v) < d(u)$  : apilar la arista  $v \text{---} u$  en  $B$ 
8 si  $\ell(u) = d(\pi(u))$  :
9   mientras  $|B| > 0$  y alguno de los dos extremos de la arista
    cima( $B$ ) sea descendiente de  $u$  (con  $u$  inclusive) :
10    escribir cima( $B$ )
11    desapilar la cima de  $B$ 
12 escribir un separador

```

cada vértice se recorre dos veces: primero para buscar los vértices que no han sido visitados y luego para calcular $\ell(v)$; esto consume $\Theta(|E|)$ pasos. Cada arista se apila en B únicamente cuando el extremo más recientemente descubierto se está posvisitando, esto aporta $\Theta(|E|)$. Por tanto el tiempo de ejecución es $\Theta(|V| + |E|)$.

CAPÍTULO 4

CARACTERIZACIÓN ALGORÍTMICA

Bello es mejor que feo.
Explícito es mejor que implícito.
Simple es mejor que complejo.
Complejo es mejor que complicado.

(The Zen of Python)

Aquí veremos que la relación de las formas \mathbb{A}_n con su árbol recubridor en profundidad justifica un algoritmo eficiente para decidir si una forma unitaria es de tipo \mathbb{A}_n .

4.1. PROPIEDAD DE LA PARTICIÓN BICOLOR

Ahora que ya sabemos descomponer una gráfica en sus componentes biconexas y puntos de articulación, surge de manera natural un tentativo algoritmo: Para determinar si q es de tipo \mathbb{A}_n descomponemos a \mathbf{B}_q en sus componentes biconexas y puntos de articulación, y verificamos si estos cumplen con el corolario 2.13, sin embargo nos toparemos con un pequeño problema: ¿cómo determinamos eficientemente si una subgráfica \mathcal{B} es un \mathbb{A} -bloque?

Recordemos que los vértices de un \mathbb{A} -bloque $\mathbf{F}_{m,m'}$ se dividen en dos subconjuntos V_0 y V_1 (uno de m vértices y el otro de m' , ver figura 2.3). Si coloreamos de blanco a los vértices de V_0 y de negro a los de V_1 entonces veremos que solo hay dos tipos de aristas:

1. las punteadas que tienen ambos extremos del mismo color;



2. las sólidas que tienen ambos extremos de diferente color.



Notemos que si después de colorear una gráfica $\mathbf{F}_{m,m'}$ quitamos una arista, el coloreo sigue teniendo las mismas propiedades pero ya no es un \mathbb{A} -bloque. Para que una gráfica sea un \mathbb{A} -bloque es necesario que además tenga todas sus aristas. ¿Cuántas aristas hay en $\mathbf{F}_{m,m'}$? ciertamente una arista entre cada par de vértices, o sea $\binom{n}{2} = \frac{n(n-1)}{2}$ donde $n = m + m'$. Esto nos da la idea para el siguiente

LEMA 4.1. *Todas las componentes biconexas de una gráfica \mathbf{B}_q son \mathbb{A} -bloques si y solo si los vértices se pueden particionar como $V = B \cup N$ de manera que las aristas punteadas tienen ambos extremos en B o ambos en N , las sólidas con un extremo en B con el otro en N , y además cada componente biconexa de n_i vértices tiene $\binom{n_i}{2}$ aristas.*

Demostración. \Rightarrow : Sean $\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_t$ las componentes biconexas de una gráfica \mathbf{B}_q , cada una de ellas un \mathbb{A} -bloque. Podemos colorear los vértices de \mathcal{B}_1 según lo descrito antes de enunciar el teorema. Supóngase que \mathcal{B}_j tiene un vértice v en común con \mathcal{B}_1 . Coloreamos del mismo color que v a los vértices que son adyacentes a v por medio de una arista punteada, y del otro color a los que son adyacentes a v por medio de una arista sólida. Como \mathcal{B}_j es un \mathbb{A} -bloque, entonces este coloreo hace que las aristas punteadas tengan ambos extremos del mismo color y que las sólidas tengan extremos de color diferente. Podemos continuar este proceso hasta colorear toda la gráfica. Haciendo B el conjunto de los vértices blancos y de N el conjunto de vértices negros acabamos con esta implicación.

\Leftarrow : Sea \mathcal{B} una componente biconexa de la gráfica \mathbf{B}_q con n_i vértices. Como \mathcal{B} es una subgráfica de \mathbf{B}_q , entonces cumple las condiciones del teorema (su conjunto de vértices está partido en $V = B \cup N$, etc.). Como \mathcal{B} tiene $\binom{n_i}{2}$ aristas entonces tiene una arista entre cada par de vértices. Supongamos que m de estos vértices pertenecen a B y m' pertenecen a N . Por hipótesis tenemos que las aristas punteadas tienen ambos extremos en B o ambos en N y las sólidas tienen un extremo en B y otro en N , entonces por definición \mathcal{B} es la gráfica $\mathbf{F}_{m,m'}$ con $V_0 = B$ y $V_1 = N$. ■

La moraleja de este lema es que no es necesario determinar que cada componente biconexa sea un \mathbb{A} -bloque por separado siempre y cuando

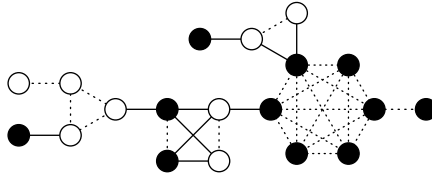


FIGURA 4.1: Coloreando los vértices según las aristas.

sepamos una manera adecuada de colorear la gráfica. En las secciones siguientes se explica cómo hacer esto con más detalle.

4.2. ÁRBOLES RECUBRIDORES PARA FORMAS \mathbb{A}_n

En esta sección veremos qué información nos aportan los árboles de recubrimiento sobre la gráfica de una forma unitaria de tipo \mathbb{A}_n . Primero ¿un árbol de recubrimiento determina sin ambigüedad la forma unitaria en cuestión, o qué información adicional hace falta?

ÁRBOLES RECUBRIDORES DE \mathbb{A} -BLOQUES

Pensemos en un \mathbb{A} -bloque cualquiera y consideremos al coloreo mencionado en la página 48, donde coloreamos de blanco a los vértices en V_0 y de negro a los vértices en V_1 . Sean u y v dos vértices de esta gráfica conectados por un camino

$$u = x_0 \text{---} x_1 \text{---} \cdots \text{---} x_n = v$$

Sin pérdida de generalidad supongamos que $u = x_0$ está coloreado de blanco. Como las aristas punteadas tienen ambos extremos del mismo color y las sólidas de diferentes colores, entonces solamente hay un cambio de color cada vez que el camino pasa por una arista sólida. Así, el color de v queda determinado por el color de u y del número de aristas sólidas que contiene este camino: si es par entonces v es blanco y en otro caso es negro. Así hemos mostrado el siguiente

LEMA 4.2. *Sean V_0 y V_1 los conjuntos que definen a un \mathbb{A} -bloque. Considérese a un camino $u \rightsquigarrow v$ en esta gráfica. Los vértices u y v pertenecen al mismo conjunto V_k si y solo si existe un número par de aristas sólidas en este camino.*

Podemos usar este lema para mostrar rápidamente el siguiente

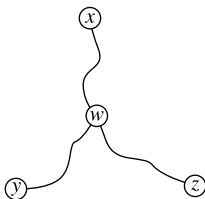
COROLARIO 4.3. *Una gráfica de aristas sólidas y punteadas con al menos dos vértices es un \mathbb{A} -bloque si y solo si es completa y todo ciclo tiene un número par de aristas sólidas.*

Demostración. \Rightarrow : Supongamos que $u \in V_0$ y que se tiene un ciclo $u \rightsquigarrow v \rightsquigarrow u$ donde v fue el último vértice en ser alcanzado mediante una arista sólida. Por contradicción, si este ciclo tiene un número impar de aristas sólidas entonces $v \in V_1$, pero como el subcamino $v \rightsquigarrow u$ tiene cero aristas sólidas —un número par— entonces $u \in V_1$.

\Leftarrow : Ver lema 2.8. ■

Estamos en posición para responder parte de la pregunta planteada al inicio de esta sección. Supongamos que T es un árbol recubridor de n vértices para un \mathbb{A} -bloque $\mathcal{B} = \mathbf{F}_{m,m'}$. Vamos a construir a \mathcal{B} . Como T es recubridor entonces $m + m' = n$ y en \mathcal{B} debe haber una arista entre cada par de vértices. Cada arista $u-v \in \mathcal{B}$ que no forma parte de T forma un ciclo en T , pero el corolario 4.3 afirma que ciclo tiene un número par de aristas sólidas, por lo tanto el color (punteado o sólido) de la arista $u-v$ queda completamente determinado por T . Si repetimos este procedimiento para cada arista, entonces \mathcal{B} queda completamente determinado por T .

Pero ¿en verdad la gráfica \mathcal{B} así construida es un \mathbb{A} -bloque? Si \mathcal{B} tiene solamente 1 o 2 vértices entonces es isomorfa a $\mathbf{F}_{1,0}$, $\mathbf{F}_{1,1}$ o $\mathbf{F}_{2,0}$ y la respuesta es que sí. Supongamos que \mathcal{B} tiene al menos tres vértices diferentes x , y y z . Sea w el último vértice que tienen en común los caminos (únicos) $x \rightsquigarrow y$ y $x \rightsquigarrow z$ en el árbol T , tal como se muestra en el siguiente esquema:



No estamos excluyendo la posibilidad de que $w = y$ o $w = z$. Denotemos con α el número de aristas sólidas en el camino $x \rightsquigarrow w$, con β las del camino $w \rightsquigarrow y$ y con γ las de $w \rightsquigarrow z$. Si los tres números α , β y γ son todos pares o todos impares, entonces $\alpha + \beta$, $\alpha + \gamma$ y $\beta + \gamma$ son todos pares, y luego entonces tenemos un ciclo $x \cdots y \cdots z \cdots x$ que es isomorfo a $\mathbf{F}_{3,0}$. En otro caso, (que no todos sean pares ni todos impares)

tendremos que exactamente uno de los números $\alpha + \beta$, $\alpha + \gamma$ y $\beta + \gamma$ es par y los otros dos impares, pero entonces x , y y z inducen a $\mathbf{F}_{2,1}$. Hemos mostrado que todo trío de vértices induce una gráfica $\mathbf{F}_{3,0}$ o $\mathbf{F}_{2,1}$, entonces por lema 2.8 hemos demostrado el

LEMA 4.4. *Todo árbol determina de manera única un \mathbb{A} -bloque y todo \mathbb{A} -bloque queda determinado por un árbol de recubrimiento.*

Ahora ya podemos resolver el problema de cómo encontrar el coloreo blanco-negro planteado al final de la sección anterior. Primero notemos que es muy fácil colorear un árbol: Iniciamos con la raíz de color blanco, y conforme vamos descendiendo por las ramas cambiamos entre blanco y negro cada vez que atravesemos alguna arista sólida. Entonces para encontrar la partición B/N de una gráfica \mathbf{B}_q basta con encontrar un árbol de recubrimiento y colorearlo; si este coloreo no define una partición válida (no cumple las condiciones del lema 4.1) entonces no existe tal partición. La justificación es que luego de haber coloreado el árbol recubridor, podemos aplicar el lema 4.4 a cada componente biconexa de \mathbf{B}_q .

ÁRBOLES RECUBRIDORES EN PROFUNDIDAD PARA FORMAS \mathbb{A}_n

De todos los posibles árboles de recubrimiento para una gráfica \mathbf{B}_q ¿qué tienen de interesante los árboles recubridores en profundidad?

Por **camino hamiltoniano** entendemos un camino pasa por todos y cada uno de los vértices de la gráfica exactamente una vez.

LEMA 4.5. *Todo árbol de recubrimiento en profundidad para la gráfica $\mathbf{F}_{m,m'}$ es un camino hamiltoniano.*

Demostración. Como $\mathbf{F}_{m,m'}$ es completa, durante un recorrido en profundidad todo vértice que esté sin visitar es adyacente al vértice actual, luego entonces se visitan todos los vértices antes de hacer el primer paso de retroceso. ■

Por **árbol binario** entendemos un árbol donde cada vértice puede tener dos hijos cuando mucho.

COROLARIO 4.6. *Sea q una forma de tipo \mathbb{A}_n . Entonces todo árbol de recubrimiento en profundidad de \mathbf{B}_q es un árbol binario.*

Demostración. Por lema anterior, el recorrido en profundidad genera un camino hamiltoniano en cada componente biconexa de \mathbf{B}_q ; por lo

tanto, los únicos vértices que pueden tener más de un hijo son los puntos de articulación. Por corolario 2.13 los puntos de articulación pertenecen a exactamente dos componentes biconexas. Entonces todo punto de articulación puede tener a lo mucho dos hijos: uno perteneciente a la componente biconexa actual (en caso de que todavía queden vértices sin visitar en la componente actual) y el otro hijo pertenece a la otra componente biconexa. ■

Si T es un árbol recubridor para \mathbf{B}_q , un **ciclo fundamental** de \mathbf{B}_q es un ciclo que se forma agregando a T una arista de \mathbf{B}_q . Diremos que un ciclo fundamental c es **maximal** si no existe ningún otro ciclo fundamental que contenga propiamente a los vértices de c .

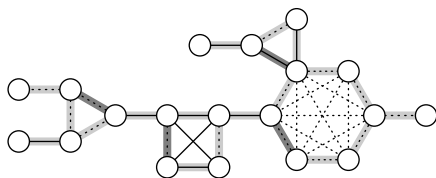


FIGURA 4.2: Un árbol de recubrimiento en profundidad (gris claro) y las aristas que definen los ciclos fundamentales maximales (gris oscuro).

Sea \mathcal{B} una componente biconexa de \mathbf{B}_q (q de tipo \mathbb{A}_n) con al menos tres vértices y sea c el camino hamiltoniano $u \rightsquigarrow v$ de \mathcal{B} definido por T . Por ser \mathcal{B} un \mathbb{A} -bloque existe la arista $u—v \in \mathcal{B}$ que no forma parte de c y que al agregarlo a T forma un ciclo fundamental. Este ciclo fundamental debe ser maximal por que, de hecho, es un ciclo hamiltoniano para \mathcal{B} . ¿Qué hay con las componentes biconexas que no tienen ni tres vértices? Como \mathbf{B}_q es conexa, estas componentes son **puentes**, es decir, aristas que al ser borradas la gráfica se vuelve disconexa. Para concluir:

LEMA 4.7. *Sea q una forma de tipo \mathbb{A}_n y sea T un árbol de recubrimiento en profundidad para \mathbf{B}_q . Si \mathcal{B} es una componente biconexa de \mathbf{B}_q entonces \mathcal{B} es un puente o bien está inducida por los vértices de un ciclo fundamental maximal.*

Sea T un árbol de recubrimiento para la gráfica \mathbf{B}_q . Si la arista $u—v$ define un ciclo fundamental maximal, o si es un puente, entonces denotaremos con $\langle u—v \rangle_T$ a la subgráfica de \mathbf{B}_q inducida por los los vértices que hay en el único camino $u \rightsquigarrow v$ de T . El lema anterior dice que cuando q es de tipo \mathbb{A}_n toda componente biconexa es una gráfica $\langle u—v \rangle_T$. Por corolario 2.13 tenemos el

COROLARIO 4.8. Sea q es una forma de tipo \mathbb{A}_n , T un árbol recubridor en profundidad para \mathbf{B}_q , y sean $\alpha_1, \alpha_2, \dots, \alpha_t$ todas las aristas que son puentes o que definen un ciclo fundamental maximal; entonces

1. $\langle \alpha_1 \rangle_T \cup \langle \alpha_2 \rangle_T \cup \dots \cup \langle \alpha_t \rangle_T = \mathbf{B}_q$;
2. $E(\langle \alpha_i \rangle_T \cap \langle \alpha_j \rangle_T) = \emptyset$ para toda $i \neq j$; y
3. todo $v \in V(\mathbf{B}_q)$ pertenece cuando mucho a dos gráficas $\langle \alpha_i \rangle_T$.

COROLARIO 4.9. Sean q , T , y α_i como en el corolario anterior; para cada $\alpha_i = u_i \text{---} v_i$ denotemos con n_i al número de vértices que contiene el camino $u_i \rightsquigarrow v_i$ de T . Entonces

$$|E(\mathbf{B}_q)| = \binom{n_1}{2} + \binom{n_2}{2} + \dots + \binom{n_t}{2}$$

Demostración. Cada $\langle \alpha_i \rangle_T$ es un \mathbb{A} -bloque, entonces $|E(\langle \alpha_i \rangle_T)| = \binom{n_i}{2}$. Las propiedades 1 y 2 del corolario anterior explican el resto. ■

4.3. RESULTADO PRINCIPAL

Para esta sección suponemos que se tiene una forma unitaria q y un árbol recubridor en profundidad T para la gráfica \mathbf{B}_q . Suponemos que los vértices fueron coloreados de blanco y negro, de tal manera que todas las aristas de T (solamente nos fijamos en las de T) si son punteadas entonces tienen extremos del mismo color, y si son sólidas entonces tienen extremos de color diferente (ver página 52). También suponemos que $\alpha_1, \alpha_2, \dots, \alpha_t$ son todas las aristas que definen un ciclo fundamental maximal o un puente, y denotamos con n_i a cantidad de vértices que hay en el camino $u_i \rightsquigarrow v_i$ en T para cada $\alpha_i = u_i \text{---} v_i$. Considere estas tres propiedades:

Propiedad 1 \mathbf{B}_q tiene $\binom{n_1}{2} + \binom{n_2}{2} + \dots + \binom{n_t}{2}$ aristas.

Propiedad 2 Toda arista punteada de \mathbf{B}_q tiene ambos extremos del mismo color, y toda arista sólida de \mathbf{B}_q tiene ambos extremos de color diferente.

Propiedad 3 Todo vértice de \mathbf{B}_q pertenece cuando mucho a dos gráficas $\langle \alpha_i \rangle_T$.

TEOREMA 4.10. *Una forma unitaria q es de tipo \mathbb{A}_n si y solo si su gráfica \mathbf{B}_q tiene estas tres propiedades.*

Idea de la demostración. Una implicación ya fue demostrada; falta mostrar el regreso. La propiedad 1 dice que las gráficas $\langle \alpha_i \rangle_T$ son gráficas completas que particionan al conjunto de aristas de \mathbf{B}_q ; como \mathbf{B}_q es conexa entonces las gráficas $\langle \alpha_i \rangle_T$ son las componentes biconexas de \mathbf{B}_q . Luego la propiedad 2 garantiza que estas componentes biconexas son \mathbb{A} -bloques. Finalmente la propiedad 3 dice que todo punto de articulación pertenece a exactamente dos componentes biconexas. Por corolario 2.13 ya acabamos. A continuación presentamos los detalles.

Demostración. \Rightarrow : Ver corolarios 4.9 y 4.8, y lema 4.1.

\Leftarrow : Primero mostraremos que que toda arista $u-v$ de \mathbf{B}_q pertenece a al menos una subgráfica $\langle \alpha_i \rangle_T$. Sin pérdida de generalidad supóngase que v descende de u . Si $u-v$ no pertenece a T entonces es una arista de retroceso que define un ciclo fundamental; por tanto u y v deben pertenecer a un ciclo fundamental maximal. En otro caso, si $u-v$ es un puente, entonces pertenece a $\langle u-v \rangle_T$. Finalmente, si $u-v$ pertenece al árbol pero no es un puente entonces por definición de puente, al quitar a $u-v$ de \mathbf{B}_q los vértices u y v siguen conectados por un camino simple $v \rightsquigarrow u$. Ya que v descende de u , este camino contiene al menos una arista de retroceso $v'-u'$ donde v' es v o un descendiente de v , y u' es u o un ancestro de u . Entonces podemos formar el ciclo fundamental $v \rightsquigarrow v'-u' \rightsquigarrow u-v$ que contiene a u y v ; por tanto u y v pertenecen a un ciclo fundamental maximal.

Sea $n'_i = |E(\langle \alpha_i \rangle_T)|$; entonces, como toda arista pertenece a una gráfica $\langle \alpha_i \rangle_T$ y como $n'_i \leq \binom{n_i}{2}$, se sigue que

$$|E(\mathbf{B}_q)| \leq \sum_{i=1}^t n'_i \leq \sum_{i=1}^t \binom{n_i}{2}$$

Por la propiedad 1 tenemos que $\sum_{i=1}^t n'_i = \sum_{i=1}^t \binom{n_i}{2}$. Como $n'_i \leq \binom{n_i}{2}$, entonces esta suma solamente puede darse cuando $n'_i = \binom{n_i}{2}$; por lo tanto las gráficas $\langle \alpha_i \rangle_T$ son completas y no comparten aristas (sí pueden compartir vértices).

$n'_i = \binom{n_i}{2}$; por lo tanto $\{E(\langle \alpha_i \rangle_T)\}_{i=1}^t$ es una partición de $E(\mathbf{B}_q)$ y $\Phi(\langle \alpha_i \rangle_T) = K_{n_i}$ para toda i .

Supongamos que p es un vértice compartido por $\langle \alpha_i \rangle_T$ y $\langle \alpha_j \rangle_T$; demostraremos que p es un punto de articulación. Por la definición de cada

$\langle \alpha_k \rangle_T$ sabemos que en T existe un camino $x \text{---} p \text{---} y$ tal que $x \text{---} p \in E(\langle \alpha_i \rangle_T)$ y $p \text{---} y \in E(\langle \alpha_j \rangle_T)$ para algunos vértices x, y de \mathbf{B}_q . Tenemos dos casos:

1. Si x es el padre de p entonces y debe ser un hijo de p . Sea v un descendiente de p , con p inclusive, y w cualquier vértice adyacente a v . Mostraremos por contradicción que w debe ser p o un descendiente de p . Supongamos que w no es p ni desciende de p ; como $v \text{---} w$ está inducida por $\langle \alpha_k \rangle_T$ para alguna k (corolario 4.8) entonces existe un ciclo fundamental maximal que pasa por el único camino $v \rightsquigarrow w$ de T , por lo tanto $x \text{---} p \text{---} y$ es una subgráfica de $\langle \alpha_k \rangle_T$, pero esto contradice que $\{E(\langle \alpha_i \rangle_T)\}_{i=1}^t$ era una partición de $E(\mathbf{B}_q)$. Por tanto p es un punto de articulación.
2. Si p no es el padre de x entonces x debe ser el padre de p . Si y es el padre de p entonces tenemos el caso anterior con los roles de x e y intercambiados, por lo tanto supongamos que p es el padre de y . Si p es la raíz de T entonces p es un punto de articulación porque tiene dos hijos. Si p no es la raíz de T entonces definiendo a x' como el padre de p este caso se reduce al anterior.

Dada la propiedad 2, y que $n'_i = \binom{n_i}{2}$, el lema 4.1 nos dice que estas componentes biconexas son \mathbb{A} -bloques. Finalmente, dado que las subgráficas $\langle \alpha_i \rangle_T$ son las componentes biconexas de \mathbf{B}_q , la propiedad 3 implica que cada punto de articulación pertenece a exactamente dos componentes biconexas. Por corolario 2.13 se sigue que q es de tipo \mathbb{A}_n . ■

EL ALGORITMO, SU IMPLEMENTACIÓN Y CONCLUSIONES

Ciencia es aquello que entendemos
suficientemente bien como para explicárselo
a una computadora. Arte es todo lo demás.

(Donald Knuth)

En este capítulo mostramos cómo se implementa la caracterización del teorema 4.10 de manera eficiente, se da justificación y se muestran ejemplos.

5.1. DESCRIPCIÓN, JUSTIFICACIÓN Y ANÁLISIS

Usaremos el teorema 4.10 para justificar un algoritmo eficiente que caracteriza a las formas A_n . Lo primero que necesitamos detallar es cómo encontramos las aristas α_i que definen ciclos fundamentales maximales o puentes. Mientras calculamos $\ell(v)$ es fácil identificar cuál vértice u hace que $\ell(v) = d(u) < d(v)$. Si v no tiene ningún hijo, o si $\ell(v) < \ell(w)$ para todo hijo w de v , entonces existe una arista $u \rightarrow v$ en B_q . Por la manera en que ℓ fue definido, no puede haber ninguna otra arista que conecte a un sucesor de v con un antecesor de u (inclusive u), por lo tanto tenemos dos posibilidades: si v es hijo de u entonces $v \rightarrow u$ es una arista de T que define un puente de B_q , y si v no es hijo de u entonces $v \rightarrow u$ es una arista de retroceso que define un ciclo fundamental maximal.

Para contar cuántos vértices hay en el camino definido por α_i ponemos la etiqueta $\gamma(u)$ que vale 1 cuando encontramos el primer vértice

de la arista α_i , a su padre $\pi(u)$ lo etiquetamos con $\gamma(\pi(u)) = \gamma(u) + 1$, y así sucesivamente hasta haber encontrado el extremo final de la arista. Si en \mathbf{B}_q los ciclos fundamentales maximales no se intersecan en las aristas (como ocurre cuando q es de tipo \mathbb{A}_n) entonces este extremo debe ser un punto de articulación que puede ser fácilmente identificado con el teorema 3.4. En caso de haber intersecciones las podemos identificar fácilmente haciendo uso del siguiente lema:

LEMA 5.1. *Dos ciclos fundamentales maximales para un árbol de recubrimiento en profundidad se intersecan en al menos una arista, en cualquiera de estos dos casos y solamente en cualquiera de estos dos:*

1. *Existe un vértice intermedio u con al menos un hijo v tal que $\ell(u) < \ell(v) < d(u)$.*
2. *Existe un vértice intermedio u con al menos dos hijos v y w tales que $\ell(v) < d(u)$ y $\ell(w) < d(u)$.*

Demostración. \Rightarrow : Dos ciclos fundamentales maximales solamente se pueden intersecar en un camino de T , digamos $x_1 \text{---} x_2 \text{---} \cdots \text{---} x_k$, donde $d(x_1) < d(x_2) < \cdots < d(x_k)$. Definimos $u = x_k$ como el último vértice que tienen en común estos ciclos. Si u fuese una hoja de T , entonces ambos ciclos son formados al agregar a T dos aristas de retroceso ancladas en u , pero entonces los vértices de uno de estos ciclos deben estar contenidos en los del otro contradiciendo que eran maximales; así, u tiene al menos un hijo v que pertenece a exactamente uno de estos dos. Sea $a \text{---} b$ la arista de retroceso tal que $\ell(u) = d(b)$ y sea $a' \text{---} b'$ la arista de retroceso tal que $\ell(v) = d(b')$. Claramente b y b' deben ser ancestros de u , y más aún, deben estar contenidos en el camino $r \rightsquigarrow x_1$ donde r es la raíz de T ; por lo tanto $\ell(v) < d(u)$ y $\ell(u) < d(u)$. Ahora tenemos dos casos: Si $a = u$ entonces la única posibilidad de que el camino $b \rightsquigarrow a$ no esté contenido en el camino $b' \rightsquigarrow a'$ es que b sea un antecesor de b' , y por lo tanto $\ell(u) = d(b) < d(b') = \ell(v) < d(u)$; para el segundo caso, si $a \neq u$ entonces u debe tener otro hijo w tal que $\ell(w) = d(b) < d(u)$.

\Leftarrow : Sea u un vértice con al menos un hijo v tal que $\ell(u) < \ell(v) < d(u)$. Como $\ell(u) \neq \ell(v)$ inferimos que hay dos aristas de retroceso diferentes $a \text{---} b$ y $a' \text{---} b'$ que definen ciclos fundamentales maximales. Entonces a es u , o un sucesor de u , pero no es un sucesor de v , y $u \text{---} \pi(u)$ es compartida por ambos ciclos fundamentales. Si en cambio u tiene dos hijos v y w tales que $\ell(v) < d(u)$ y $\ell(w) < d(u)$ entonces

con mayor razón la arista $u \text{---} \pi(u)$ está compartida por ambos ciclos fundamentales. ■

En caso de que la gráfica sí contenga estas intersecciones podemos detener nuestro análisis: q no es de tipo \mathbb{A}_n . En otro caso podemos calcular $\sum_{i=1}^t \binom{n_i}{2}$ para verificar la propiedad 1.

Bajo la misma suposición es fácil detectar si existe un vértice u que viola la propiedad 3: Sean c_1, c_2, \dots, c_k los hijos de u , y $\pi(u)$ el padre de u . Si $k > 2$ por corolario 4.6 sabemos que q no es de tipo \mathbb{A}_n . Si u es la raíz de T o si $k = 1$ entonces claramente u no puede violar la propiedad 3. En otro caso, si u es un vértice intermedio con $k = 2$ entonces dos de las aristas incidentes en u deben pertenecer a la misma subgráfica $\langle \alpha_i \rangle_T$; esto ocurre si y solo si $\ell(u) = \ell(c_j)$ para un hijo c_j .

Finalmente describimos cómo se verifica la propiedad 2: diremos que $h(u) = 1$ si u es blanco y $h(u) = -1$ si es negro. Recordemos que la arista $x_i \text{---} x_j$ es sólida si $q_{ij} = -1$ y punteada si $q_{ij} = 1$. Definimos $h(r) = 1$ para la raíz r , y durante el recorrido en profundidad hacemos lo siguiente: si x_i es el vértice actual y x_j es un vértice adyacente a x_i que todavía no ha sido visitado entonces marcamos $h(x_j) = q_{ij} h(x_i)$. De esta manera tenemos la garantía de que todas las aristas de T son solo de los tres tipos $\bigcirc \cdots \bigcirc$, $\bullet \cdots \bullet$ o $\bigcirc \text{---} \bullet$; las únicas que podrían no ser de esta forma son las aristas de retroceso de \mathbf{B}_q . Por lo tanto podemos detectar una violación a la propiedad 2 recorriendo a T en postorden; para cada vértice x_j adyacente al vértice actual x_i tal que $d(x_j) < d(x_i)$ tenemos que la arista $x_i \text{---} x_j$ viola la propiedad 2 si y sólo si $h(x_j) \neq q_{ij} h(x_i)$.

Para su implementación, el coloreo de la gráfica se hace durante VISITAR, y en POSVISITAR se realizan los test de coloreo, intersecciones de ciclos fundamentales maximales, si el árbol es binario, si los vértices pertenecen a más de una gráfica $\langle \alpha_i \rangle_T$, y se lleva la cuenta del total de aristas que debería tener toda la gráfica. Finalmente, al término del recorrido podemos comprobar si el total de aristas predicho es realmente el total de aristas de la gráfica, y si la gráfica es conexa (algo que hemos estado suponiendo desde el principio).

Que la arista $x_i \text{---} x_j$ sea sólida o punteada depende de q_{ij} , por lo tanto podemos adaptar la representación por listas de adyacencia para que sea (L_1, L_2, \dots, L_n) donde

$$L_i = \left\{ \left(j, q_{ij} \right) \mid q_{ij} \neq 0, i \neq j \right\}$$

DESCRIPCIÓN FORMAL

PREVISITAR(G, i) :

$d(i) \leftarrow t$
 $t \leftarrow t + 1$

POSVISITAR(G, i) :

$\ell(i) \leftarrow d(i)$

$C \leftarrow \emptyset$

para cada $(j, q_{ij}) \in L_i$:

si $h(j) \neq q_{ij} h(i)$: **responder** FALSO

►Propiedad 2

si $\pi(j) = i$:

$C \leftarrow C \cup \{j\}$

si $|C| > 2$: **responder** FALSO

►Corolario 4.6

si $\ell(j) \leq \ell(i)$:

$\ell(i) \leftarrow \ell(j)$

$\gamma(i) \leftarrow \gamma(j) + 1$

en otro caso, si $d(j) < \ell(i)$:

$\ell(i) \leftarrow d(j)$

$\gamma(i) \leftarrow 1$

para cada $j \in C$:

si $\ell(i) < \ell(j) < d(i)$: **responder** FALSO

►Lema 5.1

si $|C| = 2$:

 Sean j y k los elementos de C

$p_1 \leftarrow (\ell(j) < d(i) \wedge \ell(k) < d(i))$

►Lema 5.1

$p_2 \leftarrow (\ell(j) \neq \ell(i) \wedge \ell(k) \neq \ell(i))$

►Propiedad 3

si $p_1 \vee p_2$: **responder** FALSO

si $\pi(i) \neq \text{NINGUNO} \wedge \ell(i) = d(\pi(i))$:

$e \leftarrow e + \binom{\gamma(i)+1}{2}$

responder VERDADERO

VISITAR(G, i) :

$\text{visitado}(i) \leftarrow \text{VERDADERO}$

 PREVISITAR(i)

para cada $(j, q_{ij}) \in L_i$:

si $\neg \text{visitado}(j)$:

$\pi(j) \leftarrow i$

$h(j) \leftarrow q_{ij} h(i)$

si $\neg \text{VISITAR}(G, j)$: **responder** FALSO

si $\neg \text{POSVISITAR}(i)$: **responder** FALSO

responder VERDADERO

ES-DYNKIN-AN(G) :

para cada $i \in V(G)$:

$visitado(i) \leftarrow \text{FALSO}$

$\pi(i) \leftarrow \text{NINGUNO}$

$\gamma(i) = 0$

$e \leftarrow 0$

$t \leftarrow 0$

$h(1) \leftarrow 1$

VISITAR($G, 1$)

para cada $i \in V(G)$:

si $\neg visitado(i)$: **responder** FALSO

 ▶ G es desconexa

$e' \leftarrow 0$

para cada $L_i \in G$: $e' \leftarrow e' + |L_i|$

$e' \leftarrow e'/2$

 ▶ $e' = |E(G)|$

si $e \neq e'$: **responder** FALSO

 ▶ Propiedad 1

responder VERDADERO

ANÁLISIS DE LA COMPLEJIDAD TEMPORAL

En esta adaptación del recorrido en profundidad cada vértice se previsa, visita y posvisita exactamente una vez. PREVISITAR aporta una cantidad constante de pasos por cada vértice, es decir, $O(|V|)$. POSVISITAR recorre la lista de adyacencia de cada vértice una sola vez, y claramente para todo vértice x_i tenemos que $|C| \leq |L_i|$, por lo tanto esta subrutina contribuye en $O(|E|)$ al número total de pasos. Así pues, la complejidad es a lo mucho $O(|V| + |E|)$, sin embargo notemos que en esta adaptación estamos limitando el recorrido a una sola componente conexa. Para toda componente conexa de e aristas y m vértices se tiene que $e \geq m - 1$ (la igualdad se da cuando esta subgráfica es un árbol), por lo tanto en esta modificación solamente se visitan $O(|E|)$ cantidad de vértices, de manera que podemos acotar la complejidad de PREVISITAR por $O(|E|)$. Por lo tanto la complejidad temporal de este algoritmo es $O(|E|)$.

Para concluir, hemos demostrado el siguiente

COROLARIO 5.2. *El problema de decidir si una forma unitaria simple*

$$q(\vec{x}) = \sum_{i=1}^n x_i^2 + \sum_{i < j} q_{ij} x_i x_j$$

es tipo Dynkin \mathbb{A}_n se puede resolver en $O(|E(\mathbf{B}_q)|)$ operaciones.

5.2. IMPLEMENTACIÓN Y EJEMPLOS

A continuación se muestra la implementación iterativa de nuestro algoritmo en cuestión utilizando el lenguaje de programación Python 3 (www.python.org).

IMPLEMENTANDO GRÁFICAS PONDERADAS

Como se había mencionado, vamos a representar las gráficas mediante la pequeña variante de la lista de adyacencia donde cada

$$L_i = \{(j, q_{ij}) \mid q_{ij} \neq 0, i \neq j\}.$$

Claramente L_i se puede ver como una función que asocia a j con q_{ij} para todo x_j adyacente a x_i . Python ofrece una estructura de datos muy eficiente para representar funciones finitas llamada `dict` (de *dictionary* o *diccionario*).

¿QUÉ ES LA PROGRAMACIÓN ORIENTADA A OBJETOS? Analicemos la siguiente afirmación: “Firulais es mi perro; en este momento está un poco hambriento y está buscando algo de comer”. Todo **objeto** tiene su **clase** (*Firulais* es un objeto de la clase *perro*), se encuentra en un **estado** actual (*hambriento*) y al igual que todos los objetos de su clase, su **comportamiento** está determinado por las acciones que puede realizar (*comer*, *ladrar*, etcétera). Bajo el paradigma de la programación orientada a objetos todo programa se diseña como la interacción de varios objetos. Todo lenguaje orientado a objetos tiene las siguientes características:

Abstracción Los datos y los programas se definen con una representación similar a su significado.

Encapsulamiento La capacidad de agrupar a los datos con los métodos que operan sobre esos datos.

Polimorfismo La capacidad de que exista una interfase común para que un objeto pueda interactuar con objetos de otras clases.

Herencia La capacidad de reutilizar código para definir nuevos objetos basados en objetos previamente definidos.

Python admite varios paradigmas de programación. Por simplicidad se ha decidido implementar una clase que representa gráficas pondera-

das, pero el algoritmo para decir si una forma unitaria es de tipo \mathbb{A}_n se hace como una función con subrutinas. El programa 5.1 codifica una clase `WeightedGraph` que representa una gráfica ponderada mediante la estrategia antes descrita. Esta clase tiene dos atributos: `n` es la cantidad de vértices de la gráfica y `Adj` es una lista de diccionarios, uno por cada vértice, donde `Adj[u]` almacena toda la información de las aristas que inciden en el vértice `u`. `__init__` es el método constructor, `__getitem__` sobrecarga el operador de corchetes `[]` de tal forma que si `G` es de tipo `WeightedGraph` entonces `G[u]` es una colección de parejas ordenadas (v, w) que representan aristas ponderadas $u \xrightarrow{w} v$; finalmente `__len__` hará que sea posible interpretar `len(G)` como la cantidad de vértices de `G`.

Python implementa listas como arreglos, pero de tal forma que el costo de agregar un elemento a una lista está amortizado en $O(1)$.

El método `add_vertices(n)` añade otros n vértices a la gráfica `G`. En cambio, `add_edge(u, v, w)` agrega la arista $u \xrightarrow{w} v$ (o simplemente actualiza el valor de w) y `remove_edge(u, v)` borra la arista $u \xrightarrow{w} v$. Para facilitar la codificación de gráficas se proporciona `add_paths` que recibe como primer parámetro una lista de caminos que se desean dibujar en la gráfica (cada camino es a su vez una lista de vértices), y como segundo parámetro el peso que tiene cada arista de este camino. `remove_paths` es similar, pero es para borrar caminos.

`to_dot('foo')` transfiere toda la información de la gráfica `G` a un archivo `foo.dot` para que pueda ser visualizada en un programa externo, como los que incluye Graphviz (www.graphviz.org). Finalmente `copy()` genera una copia de `G`.

PROGRAMA 5.1: Graph.py

```

1 class WeightedGraph:
2     def __init__(self, n = 0):
3         self.Adj = [{ } for k in range(n)]
4         self.n = n
5
6     def __getitem__(self, key):
7         return self.Adj[key].items()
8
9     def __len__(self):
10        return self.n
11
12    def add_vertices(self, n = 1):
13        self.Adj.extend({ } for k in range(n))
14        self.n += n
15
16    def add_edge(self, u, v, weight = 1):
17        if 0 <= u < self.n and 0 <= v < self.n and u != v:
18            self.Adj[u][v] = weight

```

```

19         self.Adj[v][u] = weight
20     else:
21         raise IndexError
22
23     def remove_edge(self, u, v):
24         del self.Adj[u][v]
25         del self.Adj[v][u]
26
27     def add_paths(self, paths, weight = 1):
28         for path in paths:
29             path = list(path)
30             for i in range(len(path) - 1):
31                 self.add_edge(path[i], path[i + 1], weight)
32
33     def remove_paths(self, paths):
34         for path in paths:
35             path = list(path)
36             for i in range(len(path) - 1):
37                 self.remove_edge(path[i], path[i + 1])
38
39     def to_dot(self, graphname):
40         s = ''.join(['graph_ ', graphname,
41                     '\n\nnode_ [shape=plaintext, width=0, height=0\n']])
42         for u in range(self.n):
43             for (v, w) in self[u]:
44                 if v > u:
45                     s += ''.join(['_ ', str(u), '_--_', str(v)])
46                     if w == 1: s += '_ [style=dotted]'
47                     s += ';\n'
48         s += '}'
49         f = open(graphname + '.dot', 'w')
50         f.write(s)
51         f.close()
52
53     def copy(self):
54         other = WeightedGraph(self.n)
55         other.Adj = [S.copy() for S in self.Adj]
56         return other

```

A continuación se muestra un pequeño código de prueba. Lo que hace es generar una gráfica de tipo Dynkin A_n utilizando el teorema 2.12 y utilizar Graphviz para convertirla a formato PDF.

PROGRAMA 5.2: GraphExample.py

```

1  from random import randint
2  from os import system
3  from Graph import WeightedGraph
4
5  def paste_Ablock(G, m, n, p):
6      V0 = list(range(G.n - 1, G.n + m - 1))
7      V0[0] = p
8      V1 = list(range(G.n + m - 1, G.n + m + n - 1))
9      G.add_vertices(m + n - 1)

```

```

10     for i in range(m - 1):
11         for j in range(i + 1, m):
12             G.add_edge(V0[i], V0[j])
13     for i in range(n - 1):
14         for j in range(i + 1, n):
15             G.add_edge(V1[i], V1[j])
16     for i in range(m):
17         for j in range(n):
18             G.add_edge(V0[i], V1[j], -1)
19
20 def build_An(t, maxn):
21     S = set()
22     G = WeightedGraph(1)
23     for k in range(t):
24         n = 0
25         while n < 2:
26             n = randint(2, maxn)
27             m = randint(0, n - 1)
28             mp = n - m
29             if m < mp: (m, mp) = (mp, m)
30             p = randint(0, G.n - 1)
31             while p in S:
32                 p = randint(0, G.n - 1)
33             paste_Ablock(G, m, mp, p)
34             S.add(p)
35     return G
36
37
38 a = int(input('Cantidad de A-bloques que se van a pegar: '))
39 b = int(input('Número máximo de vértices por A-bloque: '))
40 f = input('Nombre del archivo donde se guardará la gráfica: ')
41 G = build_An(a, b)
42 G.to_dot(f)
43 system('neato' + f + '.dot' + ' -Tpdf -o' + f + '.pdf')

```

EL PROGRAMA

El programa 5.3 contiene la implementación iterativa de nuestro algoritmo conforme lo descrito en el algoritmo 3.3. Tenemos un objeto `iter` por cada vértice, de manera que el programa recuerda la última posición en la que se quedó mientras exploraba la lista de adyacencia del vértice actual; este es el propósito de la variable `remaining`. Para verificar si existe un vértice adyacente actual que esté sin visitar se utiliza la subrutina `find_next`, que funciona pidiendo repetidamente el siguiente elemento al iterador (de la lista de adyacencia) del vértice actual hasta que encuentra un vértice sin visitar o hasta que termina de recorrer la lista de adyacencia. El programa se ha adaptado de tal manera que siempre muestra los paréntesis del recorrido (ver página 42); para ello se ha definido una variable `history` que almacena la lista

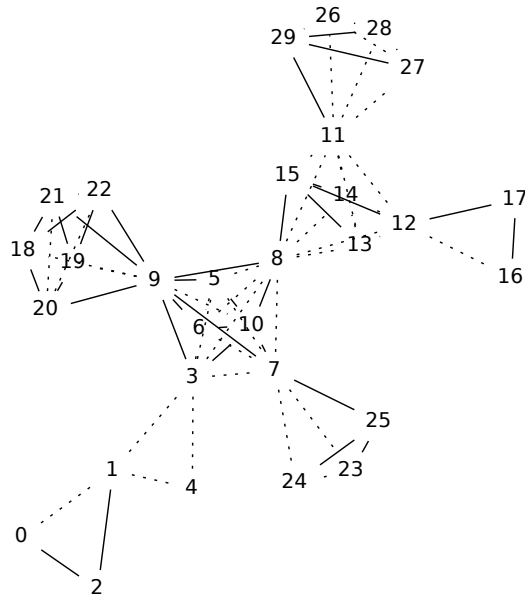


FIGURA 5.1: Gráfica aleatoria generada por el programa 5.2

de paréntesis. Otro detalle de implementación es que si la respuesta es `False` nos muestra la justificación mediante la subrutina `fail`.

PROGRAMA 5.3: `AnDecider.py`

```

1  from Graph import WeightedGraph
2
3  def is_Dynkin_An(L):
4
5      def previsit(i):
6          nonlocal history, d, t
7          history.append('(' + str(i))
8          d[i] = t
9          t += 1
10
11     def posvisit(i):
12         nonlocal L, ell, pi, d, gamma, h, e, history
13         history.append(str(i) + ')')
14         ell[i] = d[i]
15         C = []
16         for (j, q) in L[i]:

```

```

17         if h[j] != q*h[i]: return fail('Propiedad_2')
18         if pi[j] == i:
19             C.append(j)
20             if len(C) > 2: return fail('Corolario_4.6')
21             if ell[j] <= ell[i]:
22                 ell[i] = ell[j]
23                 gamma[i] = gamma[j] + 1
24             elif d[j] < ell[i]:
25                 ell[i] = d[j]
26                 gamma[i] = 1
27         for j in C:
28             if ell[j] < ell[i] < d[i]: return fail('Lema_5.1a')
29     if len(C) == 2:
30         [j, k] = C
31         if ell[j] < d[i] and ell[k] < d[i]:
32             return fail('Lema_5.1b')
33         if ell[j] != ell[i] and ell[k] != ell[i]:
34             return fail('Propiedad_3')
35     if pi[i] != None and ell[i] == d[pi[i]]:
36         m = gamma[i] + 1
37         e += (m*(m - 1))/2
38     return True
39
40 def find_next(i):
41     nonlocal visited, remaining
42     while True:
43         (j, q) = next(remaining[i], (None, None))
44         if j == None or not visited[j]: return (j, q)
45
46 def fail(arg):
47     nonlocal history, n
48     print('No es de tipo An por', arg)
49     print(' '.join(history))
50     print('e=', e)
51     print('|E|=', sum(len(L[i]) for i in range(n))/2)
52     return False
53
54 n = len(L)
55 if n == 0: return fail('n!=0')
56
57 history = []
58
59 visited = [False for i in range(n)]
60 pi = [None for i in range(n)]
61 gamma = [0 for i in range(n)]
62 h = [0 for i in range(n)]
63 d = [None for i in range(n)]
64 ell = [None for i in range(n)]
65 remaining = [iter(L[i]) for i in range(n)]
66 e = 0
67 t = 0
68 h[0] = 1
69
70 visited[0] = True
71 pi[0] = None
72 previsit(0)

```

```

73     i = 0
74     while i != None:
75         (j, q) = find_next(i)
76         while j != None:
77             visited[j] = True
78             pi[j] = i
79             h[j] = q*h[i]
80             previsit(j)
81             i = j
82             (j, q) = find_next(i)
83         if not posvisit(i): return False
84         i = pi[i]
85
86     if any(not visited[i] for i in range(n)):
87         return fail('Disconexidad')
88     if e != sum(len(L[i]) for i in range(n))/2:
89         return fail('Propiedad_1')
90     print(' '.join(history))
91     return True

```

EJEMPLOS DE EJECUCIÓN

A continuación se presenta un pequeño programa que pone a prueba el algoritmo sobre varias gráficas, incluyendo $F_{3,2}$, el ejemplo que aparece en Barot (1999) (figura 5.2), y varias alteraciones pequeñas. Algunos renglones del siguiente programa fueron quebrados en varias líneas porque no cabían en la página, favor de poner atención en los números de reglón que aparecen al lado izquierdo.

PROGRAMA 5.4: Test.py

```

1  from Graph import WeightedGraph
2  from AnDecider import is_Dynkin_An
3
4  #Ejemplo de una gráfica F[3,2]:
5  print('Comprobando_F[3,2]...')
6  G = WeightedGraph(5)
7  G.add_paths([[0, 1, 2, 0], [3, 4]], 1)
8  G.add_paths([[0, 3, 1, 4, 2, 3], [0, 4]], - 1)
9  G.to_dot('F32')
10 if is_Dynkin_An(G): print('Sí es de tipo_An')
11 print()
12
13 #Ejemplo de [Barot99]
14 print('Comprobando_[Barot99]...')
15 B = WeightedGraph(19)
16 B.add_paths([[0, 1, 2, 4, 1],
17             [16, 15, 17, 18, 9, 10, 14, 15, 18, 10, 17, 14, 9,
18             15],
19             [5, 8], [6, 7], [11, 12], [10, 15], [9, 17], [18,
19             14]], 1)

```



```

19 B.add_paths([[4, 5, 7, 8, 6, 5],
20             [13, 12, 10, 11],
21             [6, 9], [3, 2]], -1)
22 B.to_dot('Barot99')
23 if is_Dynkin_An(B): print('Sí es de tipo An')
24 print()
25
26 #El ejemplo de [Barot99] pero cambiando la arista (6)---(7) por
una sólida
27 print('Comprobando [Barot99]  $\square$ -(6)  $\cdots$  (7)  $\square$ +(6)---(7)  $\cdots$ ')
28 G = B.copy()
29 G.add_edge(6, 7, -1)
30 G.to_dot('C1')
31 if is_Dynkin_An(G): print('Sí es de tipo An')
32 print()
33
34 #El ejemplo de [Barot99] pero agregando la arista punteada (19)
 $\cdots$  (12)
35 print('Comprobando [Barot99]  $\square$ +(19)  $\cdots$  (12)  $\cdots$ ')
36 G = B.copy()
37 G.add_vertices(1)
38 G.add_edge(19, 12, 1)
39 G.to_dot('C2')
40 if is_Dynkin_An(G): print('Sí es de tipo An')
41 print()
42
43 #El ejemplo de [Barot99] pero agregando la arista sólida (6)
---(19)
44 print('Comprobando [Barot99]  $\square$ +(6)---(19)  $\cdots$ ')
45 G = B.copy()
46 G.add_vertices(1)
47 G.add_edge(6, 19, -1)
48 G.to_dot('C3')
49 if is_Dynkin_An(G): print('Sí es de tipo An')
50 print()
51
52 #El ejemplo de [Barot99] pero agregando la arista punteada (5)
 $\cdots$  (9)
53 print('Comprobando [Barot99]  $\square$ +(5)  $\cdots$  (9)  $\cdots$ ')
54 G = B.copy()
55 G.add_edge(5, 9, 1)
56 G.to_dot('C4')
57 if is_Dynkin_An(G): print('Sí es de tipo An')
58 print()
59
60 #El ejemplo de [Barot99] pero quitando la arista (9)---(14)
61 #Aquí falla el test de la cantidad de aristas
62 print('Comprobando [Barot99]  $\square$ -(9)---(14)  $\cdots$ ')
63 G = B.copy()
64 G.remove_edge(9, 14)
65 G.to_dot('C5')
66 if is_Dynkin_An(G): print('Sí es de tipo An')
67 print()
68
69 #El ejemplo de [Barot99] pero quitando la arista (15)---(16)
70 print('Comprobando [Barot99]  $\square$ +(15)---(16)')

```

```

71 | G = B.copy()
72 | G.remove_edge(15, 16)
73 | G.to_dot('C6')
74 | if is_Dynkin_An(G): print('Sí es de tipo An')
75 | print()

```

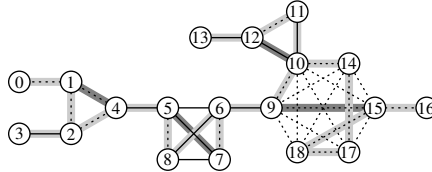


FIGURA 5.2: Recorrido realizado por computadora sobre el ejemplo de Barot (1999). Las aristas grises pertenecen al árbol, y las grises oscuro son las que definen los ciclos fundamentales maximales.

Al ejecutar este programa la pantalla arroja los siguientes resultados:

```

1 | Comprobando F[3,2]...
2 | (0 (1 (2 (3 (4 4) 3) 2) 1) 0)
3 | Sí es de tipo An
4 |
5 | Comprobando [Barot99]...
6 | (0 (1 (2 (3 3) (4 (5 (8 (6 (9 (10 (11 (12 (13 13) 12) 11) (14
   | (17 (18 (15 (16 16) 15) 18) 17) 14) 10) 9) (7 7) 6) 8) 5) 4)
   | 2) 1) 0)
7 | Sí es de tipo An
8 |
9 | Comprobando [Barot99] - (6)...(7) + (6)---(7)...
10 | No es de tipo An por Propiedad 2
11 | (0 (1 (2 (3 3) (4 (5 (8 (6 (9 (17 (18 (10 (11 (12 (13 13) 12)
   | 11) (14 (15 (16 16) 15) 14) 10) 18) 17) 9) (7 7)
12 | e = 22
13 | |E| = 33
14 |
15 | Comprobando [Barot99] + (19)...(12)...
16 | No es de tipo An por Propiedad 3
17 | (0 (1 (2 (3 3) (4 (5 (8 (6 (9 (17 (18 (10 (11 (12 (19 19) (13
   | 13) 12)
18 | e = 3
19 | |E| = 34
20 |
21 | Comprobando [Barot99] + (6)---(19)...
22 | No es de tipo An por Corolario 4.6
23 | (0 (1 (2 (3 3) (4 (5 (8 (6 (9 (17 (18 (10 (11 (12 (13 13) 12)
   | 11) (14 (15 (16 16) 15) 14) 10) 18) 17) 9) (19 19) (7 7) 6)
24 | e = 23
25 | |E| = 34
26 |
27 | Comprobando [Barot99] + (5)...(9)...
28 | No es de tipo An por Lema 5.1b
29 | (0 (1 (2 (3 3) (4 (5 (8 (6 (9 (17 (18 (10 (11 (12 (13 13) 12)
   | 11) (14 (15 (16 16) 15) 14) 10) 18) 17) 9) (7 7) 6)

```

```

30 e = 21
31 |E| = 34
32
33 Comprobando [Barot99] - (9)---(14)...
34 No es de tipo An por Propiedad 1
35 (0 (1 (2 (3 3) (4 (5 (8 (6 (9 (17 (18 (10 (11 (12 (13 13) 12)
      11) (14 (15 (16 16) 15) 14) 10) 18) 17) 9) (7 7) 6) 8) 5) 4)
      2) 1) 0)
36 e = 33
37 |E| = 32
38
39 Comprobando [Barot99] + (15)---(16)
40 No es de tipo An por Disconexidad
41 (0 (1 (2 (3 3) (4 (5 (8 (6 (9 (17 (18 (10 (11 (12 (13 13) 12)
      11) (14 (15 15) 14) 10) 18) 17) 9) (7 7) 6) 8) 5) 4) 2) 1)
      0)
42 e = 32
43 |E| = 32

```

5.3. CONCLUSIONES Y TRABAJO A FUTURO

Por supuesto que el teorema 4.10 y el corolario 5.2 son las obvias conclusiones que dan término y solución al problema que se planteó en esta tesis (ver sección 1.3), sin embargo conviene observar en perspectiva la manera en que se obtuvieron dichos resultados y reflexionar acerca del trabajo realizado y del que falta realizarse.

La tesis está fuertemente fundamentada en el estudio de la conectividad de gráficas (un tema sumamente amplio e interesante) sin embargo las herramientas principales, como las componentes biconexas, son de nivel básico. Esta es una muestra de la robustez y la utilidad que tiene un tema elemental de teoría de gráficas en problemas matemáticos contemporáneos.

El algoritmo de recorrido en profundidad ha hecho gala una vez más de su versatilidad para adaptarse a problemas concretos y de la cantidad de información que arroja sobre variadas estructuras. Es recalable el hecho de que la adaptación propuesta proporciona un algoritmo asintóticamente óptimo: $O(|E|)$ crece linealmente conforme el tamaño la representación de una forma cuadrática, cualquier algoritmo que funcione en menos tiempo sencillamente no puede leer todos los datos de la forma cuadrática.

En Barot (2001) se da una caracterización de las gráficas de \mathbb{D}_n ; sin embargo tratar de hacer una descomposición como la de este trabajo requeriría ahondar más a fondo en temas de conectividad, y en particular la triconexidad. Tal vez una extensión del algoritmo aquí propuesto

pueda ser suficiente para atacar el problema. En este mismo artículo se puede apreciar una lista de gráficas admisibles para \mathbb{E}_6 siempre y cuando sea ignorado el hecho de que una arista sea punteada; parece ser que hace falta una caracterización fundamentada en conectividad para las gráficas que definen formas \mathbb{E}_6 , \mathbb{E}_7 y \mathbb{E}_8 . De hecho sigue abierto un problema más general como es la \mathbb{Z} -equivalencia para formas no necesariamente positivas.

También queda abierto el problema de encontrar una implementación eficiente del algoritmo 2.1, encontrar su complejidad temporal, y posiblemente extender dicho algoritmo para que determine la \mathbb{Z} -equivalencia de las matrices cuasi-Cartan definidas positivas. Estas matrices tienen su fundamento en la teoría de Lie y comparten muchas similitudes con las matrices asociadas a las formas unitarias positivas. En este trabajo no se requirió hacer dicho análisis debido a que el problema fue delimitado a las gráficas que definen formas unitarias tipo \mathbb{A}_n y a que el algoritmo aquí encontrado resultó ser óptimo.

CONCEPTOS PRELIMINARES

Cuando uso una palabra, yo elijo su significado, ni más ni menos

(Alicia en el País de las Maravillas)

Muchas partes de este trabajo requieren conocimientos previos de matemáticas discretas, algoritmia y álgebra lineal. Este apéndice repasa las notaciones, definiciones y propiedades de conjuntos, relaciones, funciones, gráficas, árboles, algoritmos y matrices.

A.1. ELEMENTOS DE MATEMÁTICAS DISCRETAS

CONJUNTOS

Un **conjunto** es una agrupación de objetos representada como una unidad. Los conjuntos pueden contener cualquier tipo de objetos, incluyendo números, símbolos, e inclusive a otros conjuntos. Los objetos en los conjuntos se conocen como sus **elementos** o **miembros**. Los conjuntos se pueden describir formalmente en muchas maneras; una de ellas es listando los elementos del conjunto entre llaves. Por ejemplo, el conjunto

$$\{7, 21, 57\}$$

contiene a los elementos 7, 21 y 57. El símbolo \in se usa para representar la membresía, y su negación es \notin . Aquí, $7 \in \{7, 21, 57\}$ pero $8 \notin \{7, 21, 57\}$. Dados los conjuntos A y B , decimos que A es un **subconjunto** de B , escrito $A \subseteq B$, si todo miembro de A es también un

miembro de B . Un conjunto A es un **subconjunto propio** de B , escrito $A \subset B$ si A es un subconjunto de B , pero no es igual a B .

El orden para describir un conjunto no importa, ni tampoco las repeticiones de sus miembros. Obtenemos el mismo conjunto escribiendo $\{57, 7, 7, 7, 21\}$. Un **conjunto infinito** contiene una infinidad de elementos. No podemos escribir una lista de todos los elementos de un conjunto infinito, así que a veces usamos “...” como sinónimo de “y así sucesivamente”. Por ejemplo, podemos describir al conjunto infinito de los **números naturales** como

$$\mathbb{N} = \{0, 1, 2, 3, \dots\}$$

y al de los **números enteros** como

$$\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}.$$

Denotamos con \mathbb{R} al conjunto de todos los números reales y con \emptyset al conjunto de 0 elementos, llamado **conjunto vacío**.

Cuando queremos describir un conjunto que contiene sus elementos de acuerdo a una regla escribimos $\{n \mid \text{regla acerca de } n\}$. Así por ejemplo $\{n \mid n = m^2 \text{ para algún } m \in \mathbb{N}\}$ significa el conjunto de los números cuadrados perfectos y $[a, b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}$ es el **intervalo** que comprende todos los números reales desde a hasta b .

La **unión** de dos conjuntos A y B , denotada como $A \cup B$ es el conjunto que obtenemos al combinar a los elementos de A y de B en un solo conjunto. La **intersección** de A y B , denotada por $A \cap B$ es el conjunto que contiene a todos los elementos que están en ambos A y B . El **complemento** de A , escrito como \bar{A} , es el conjunto de todos los elementos en consideración que no pertenecen a A .

Como en casi todas las matemáticas, un dibujo siempre aclara el concepto. Para los conjuntos usamos un dibujo denominado **diagrama de Venn**. En los diagramas de Venn cada conjunto se representa como una región del plano.

SUCESIONES Y *m*-ADAS

Una **sucesión** de objetos es una lista de estos objetos en algún orden. Usualmente designamos una sucesión escribiendo su lista entre paréntesis. Por ejemplo, la sucesión 7, 21, 57 se escribiría como

$$(7, 21, 57).$$

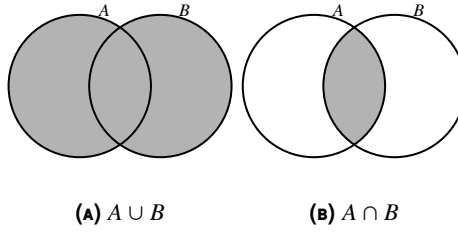


FIGURA A.1: Diagramas de Venn

En un conjunto el orden no importa, pero en una sucesión sí. Entonces $(7, 21, 57)$ no es lo mismo que $(57, 7, 21)$. Asimismo, la repetición también importa en las sucesiones, aunque no en los conjuntos. La sucesión $(7, 7, 21, 57)$ no es la misma que ninguna de las dos anteriores, mientras que los conjuntos $\{7, 21, 57\}$ y $\{7, 7, 21, 57\}$ sí son idénticos.

Como los conjuntos, las sucesiones pueden ser finitas o infinitas. Una ***m*-ada** es una sucesión finita de m elementos. Así, $(7, 21, 57)$ es una 3-ada —pronunciado *tríada*. A las 2-adas también se les llama **pares**.

Los conjuntos y sucesiones pueden aparecer como elementos de otros conjuntos o sucesiones. Por ejemplo, el conjunto potencia de un conjunto A es el conjunto de todos los subconjuntos de A . Si $A = \{0, 1\}$, el conjunto potencia de A es $\{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$. El conjunto de todos los pares de 0 y 1 es $\{(0, 0), (0, 1), (1, 0), (1, 1)\}$.

Si A y B son dos conjuntos, su **producto cartesiano**, escrito como $A \times B$, es el conjunto de todos los pares donde cada primer elemento es un miembro de A y cada segundo elemento es miembro de B . Por ejemplo, si $A = \{1, 2\}$ y $B = \{x, y, z\}$ entonces

$$A \times B = \{(1, x), (1, y), (1, z), (2, x), (2, y), (2, z)\}.$$

También podemos considerar el producto cartesiano de m conjuntos A_1, A_2, \dots, A_m , denotado como $A_1 \times A_2 \times \dots \times A_m$. Este conjunto consiste de todas las m -adas (a_1, a_2, \dots, a_m) donde cada $a_i \in A_i$.

Continuando con nuestro ejemplo anterior tenemos que

$$A \times B \times A = \{(1, x, 1), (1, y, 1), (1, z, 1), (2, x, 1), (2, y, 1), (2, z, 1), \\ (1, x, 2), (1, y, 2), (1, z, 2), (2, x, 2), (2, y, 2), (2, z, 2)\}.$$

Si tenemos el producto cartesiano de un conjunto consigo mismo po-

demos abreviar

$$\overbrace{A \times A \times \cdots \times A}^{k \text{ veces}} = A^k.$$

El conjunto \mathbb{N}^2 equivale a $\mathbb{N} \times \mathbb{N}$, todos los pares de números naturales. También lo podemos escribir como $\{(i, j) \mid i, j \geq 0\}$.

FUNCIONES Y RELACIONES

Las funciones son parte central de las matemáticas. Una **función** es un objeto que establece una relación de entrada y salida; es decir, una función recibe una entrada y produce una salida. En toda función la misma entrada produce la misma salida. Si f es una función cuyo valor de salida es b cuando su valor de entrada es a , escribimos

$$f(a) = b.$$

Por ejemplo, la función de valor absoluto *abs* toma como entrada un número x y devuelve x si x es positivo y $-x$ si x es negativo. Así, $abs(2) = abs(-2) = 2$. La adición es otro ejemplo de función, escrita *suma*. La entrada para la función de adición es un par de números, y la salida es la suma de esos números.

El **dominio** de una función es el conjunto de todas las entradas posibles para esa función; en cambio, al conjunto de salidas se le llama **rango**. Para decir que f es una función con dominio D y rango R escribimos

$$f : D \rightarrow R.$$

En el caso de la función *abs*, si estamos trabajando con enteros, el dominio y el rango son \mathbb{Z} , así que escribimos $abs : \mathbb{Z} \rightarrow \mathbb{Z}$. En el caso de la función suma de enteros, el dominio es el conjunto de los pares de enteros $\mathbb{Z} \times \mathbb{Z}$ y el rango es \mathbb{Z} , así que escribimos $suma : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$. Note que una función no necesariamente utiliza todos los elementos del rango especificado. Por ejemplo, la función *abs* nunca toma el valor -1 a pesar de que $-1 \in \mathbb{Z}$. Una función que sí usa todos los elementos del rango se dice que es **sobreyectiva**. Asimismo, cuando a entradas diferentes corresponden salidas diferentes decimos que la función es **inyectiva**. Una función es **biyectiva** cuando es inyectiva y sobreyectiva.

Podemos describir a una función en varias formas. Una de ellas es con un procedimiento para calcular la salida a partir de la entrada especificada. Por ejemplo la función **módulo** recibe dos argumentos a y m como entrada y arroja como salida el residuo de dividir a entre m . Esta

es la generalización del minuterio de un reloj, que cuenta los minutos módulo 60 (por ejemplo, a los 555 minutos después de la media noche el minuterio marca 15 min, que es precisamente 555 módulo 60).

Otra forma de describir una función es mediante una tabla que lista todas las posibles entradas y la salida asociada a cada entrada. Por ejemplo, podemos definir una función $f : \{0, 1, 2, 3, 4\} \rightarrow \{0, 1, 2, 3, 4\}$ mediante la tabla

n	$f(n)$
0	1
1	2
2	3
3	4
4	0

Esta función suma 1 a su entrada y devuelve el resultado módulo 5. Cuando hacemos aritmética modular definimos $\mathbb{Z}_m = \{0, 1, \dots, m-1\}$. Con esta notación la función antes mencionada es de la forma $f : \mathbb{Z}_5 \rightarrow \mathbb{Z}_5$.

A veces se usa una tabla bidimensional cuando el dominio es el producto cartesiano de dos conjuntos. Veamos un ejemplo de función $g : \mathbb{Z}_4 \times \mathbb{Z}_4 \rightarrow \mathbb{Z}_4$. La entrada que aparece en el renglón etiquetado con i y columna etiquetada con j es el valor de $g(i, j)$.

g	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

Esta función produce la suma módulo 4.

Cuando el dominio de una función f es $A_1 \times A_2 \times \dots \times A_m$ para algunos conjuntos A_1, A_2, \dots, A_m , la entrada para f es una m -ada (a_1, a_2, \dots, a_m) y llamamos a los a_i **argumentos** para f . Una función con m argumentos se llama **función m -aria**. Al número m se le conoce como la **aridad** de la función. Si m es 1, f tiene un solo argumento y f es una **función monaria**. Si m es 2, f es una **función binaria**. Ciertas funciones binarias que son muy familiares se escriben en **notación infija**, con el símbolo de la función colocado entre los dos argumentos, y no como en la **notación prefija**, donde el símbolo los precede. Por ejemplo, la función *suma* comúnmente se escribe en notación infija con el símbolo

+ entre los dos argumentos, así escribimos “ $a+b$ ” en vez de *suma* (a, b). Asimismo la función módulo se escribe como $a \bmod m$.

Un **predicado** es una función con rango {VERDADERO, FALSO}. A los predicados también se les conoce como **propiedades**. Por ejemplo definamos a *espar* como una propiedad que vale VERDADERO si la entrada es un número par y FALSO en otro caso. Aquí *espar* (4) = VERDADERO pero *espar* (5) = FALSO.

Una propiedad con dominio A^m se dice que es una **relación m -aria** sobre A . A una relación 2-aria se le conoce como **relación binaria**. En las relaciones binarias preferimos usar la notación infija. Por ejemplo, “menor que” es una relación comúnmente denotada con el signo $<$. La relación “igual que” es más familiar y se denota con el signo $=$. Si \mathcal{R} es una relación binaria entonces el enunciado $a\mathcal{R}b$ significa que $a\mathcal{R}b = \text{VERDADERO}$; y en general para cualquier relación \mathcal{R} el enunciado $\mathcal{R}(a_1, a_2, \dots, a_m)$ significa que $\mathcal{R}(a_1, a_2, \dots, a_m) = \text{VERDADERO}$. Por ejemplo, en el juego infantil *Piedra, Papel o Tijeras* dos jugadores escogen un elemento del conjunto {PIEDRA, PAPEL, TIJERAS} e indican su selección haciendo una seña con la mano. Si los dos contrincantes seleccionan el mismo elemento el juego vuelve a comenzar. Si la selección es diferente, un jugador gana de acuerdo a la relación *vence*.

<i>vence</i>	PIEDRA	PAPEL	TIJERAS
PIEDRA	FALSO	FALSO	VERDADERO
PAPEL	VERDADERO	FALSO	FALSO
TIJERAS	FALSO	VERDADERO	FALSO

Aquí se puede leer que PIEDRA *vence* TIJERAS y TIJERAS *vence* PAPEL, pero PAPEL *vence* PAPEL ES FALSO.

A veces conviene describir a los predicados como conjuntos. Todo predicado $P : D \rightarrow \{\text{VERDADERO, FALSO}\}$ se representa como el conjunto $\{a \mid P(a) = \text{VERDADERO}\}$. Así, la relación *vence* se puede reescribir como

$$\{(PIEDRA, TIJERAS), (PAPEL, PIEDRA), (TIJERAS, PAPEL)\}.$$

Un caso especial de una relación binaria, llamada **relación de equivalencia**, captura la noción de que dos objetos son equivalentes hasta cierta característica. Una relación binaria \mathcal{R} es una relación de equivalencia si satisface estas tres condiciones:

1. \mathcal{R} es **reflexiva** si para cada x , $x\mathcal{R}x$;
2. \mathcal{R} es **simétrica** si para cada x e y , $x\mathcal{R}y$ implica $y\mathcal{R}x$; y

3. \mathcal{R} es **transitiva** si para cada x, y y z , $x\mathcal{R}y$ y $y\mathcal{R}z$ implican $x\mathcal{R}z$.

Si \mathcal{R} es una relación de equivalencia entonces se define la **clase de equivalencia** de x , denotada $[x]_{\mathcal{R}}$, como el conjunto de todos los elementos y tales que $x\mathcal{R}y$.

Por ejemplo, definamos una relación de equivalencia \equiv_7 sobre los números naturales. Decimos que $i \equiv_7 j$ si $i - j$ es un múltiplo de 7. Esta relación satisface las tres condiciones: Es reflexiva porque $i - i = 0$ es un múltiplo de 7. Es simétrica porque $i - j$ es un múltiplo de 7 cuando $j - i$ es un múltiplo de 7. Es transitiva porque si $i - j$ y $j - k$ son múltiplos de 7, entonces $i - k = (i - j) + (j - k)$ es la suma de dos múltiplos de 7 y por lo tanto también es un múltiplo de 7. La clase de equivalencia de 9 es

$$\begin{aligned} [9]_{\equiv_7} &= \{y \mid y \equiv_7 9\} = \{y \mid y - 9 \text{ es múltiplo de } 7\} \\ &= \{y \mid y - 9 = 7k \text{ para algún entero } k\} \\ &= \{y \mid y = 7(k + 1) + 2 \text{ para algún entero } k\} \\ &= \{y \mid y - 2 \text{ es un múltiplo de } 7\} \\ &= [2]_{\equiv_7} \end{aligned}$$

Partir (o **particionar**) a un conjunto A significa dividirlo en subconjuntos B_1, B_2, B_3, \dots de manera que todo elemento x de A pertenece a exactamente un B_i . Decimos que $\{B_1, B_2, B_3, \dots\}$ es una **partición** de A .

Un teorema clásico de la teoría de conjuntos es que toda relación de equivalencia sobre un conjunto A lo particiona en sus clases de equivalencia. Veamos la demostración: por reflexividad todo x pertenece a su clase $[x]_{\mathcal{R}}$; ahora bien, si x pertenece a dos clases $[y]_{\mathcal{R}}$ y $[z]_{\mathcal{R}}$, entonces $y\mathcal{R}x$ y $z\mathcal{R}x$. Por simetría $x\mathcal{R}z$ y por transitividad $y\mathcal{R}z$. De aquí se aprecia que todo elemento equivalente con y es también equivalente con z y por lo tanto $[y]_{\mathcal{R}}$ y $[z]_{\mathcal{R}}$ son en realidad el mismo conjunto. Esto significa que x pertenece exclusivamente a una sola clase de equivalencia; por lo tanto las clases de equivalencia particionan al conjunto A . También podemos hacer el proceso inverso y así transformar toda partición en una relación de equivalencia: definimos $x\mathcal{R}y$ como “ x y y pertenecen al mismo subconjunto B_i ” y es fácil comprobar que \mathcal{R} satisface las tres propiedades de una relación de equivalencia.

GRÁFICAS

Una **gráfica no dirigida**, o simplemente **gráfica**, es un conjunto de puntos y líneas. Cada línea conecta a dos puntos diferentes. A los puntos se les llama **vértices** y a las líneas **aristas**.

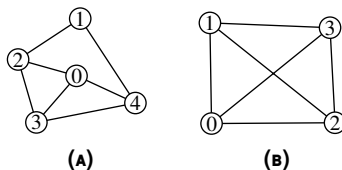


FIGURA A.2: Ejemplos de gráficas

Usualmente identificamos los vértices únicamente por su etiqueta, así por ejemplo, nos da lo mismo escribir “④” o simplemente “4”. Cuando entre cada par de vértices existe a lo mucho una sola arista entonces decimos que es una **gráfica simple**, y en otro caso **multigráfica**.

Algunos autores relajan la condición de que una arista conecte a dos vértices diferentes. En este caso se define **lazo** como una arista tal que ambos extremos conectan al mismo vértice. A las gráficas con lazos se les llama **seudográficas**, y no serán consideradas en este trabajo.

Usamos $i—j$ para representar una arista que conecta a i con j . El orden de i y j no importa, por eso muchos autores representan formalmente esta arista como $\{i, j\}$. Cuando existe una arista entre dos vértices u y v entonces decimos que u y v son **adyacentes** o **vecinos**. Si V es el conjunto de vértices y E el de aristas de una gráfica G entonces escribimos $G = (V, E)$. Podemos describir una gráfica con un diagrama o más formalmente especificando V y E . Por ejemplo, una descripción formal de la gráfica que aparece en la figura A.2a es

$$(\{0, 1, 2, 3, 4\}, \{0—3, 0—4, 1—2, 2—4, 2—0\}).$$

A veces es más conveniente denotar al conjunto de vértices de G como $V(G)$ y a su conjunto de aristas como $E(G)$.

El número de aristas que **inciden** en un vértice se llama el **grado** del vértice. En la figura A.2a el vértice ① tiene grado 2 pero los vértices ①, ②, ③ y ④ tienen grado 3. En la figura A.2b todos los vértices tienen grado 3. En toda gráfica $G = (V, E)$ se tiene que

$$\sum_{v \in V} \text{grado}(v) = 2 |E|$$

porque cada arista incide en dos vértices. A veces a esta ecuación se le conoce como el *lema del saludo de manos*.

Decimos que una gráfica G es una subgráfica de la gráfica H si los vértices de G son un subconjunto de los vértices de H y las aristas de G son un subconjunto de las aristas de H sobre los vértices correspondientes. La figura A.3 ilustra el concepto.

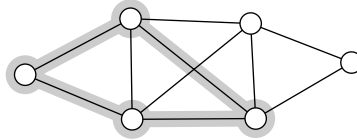


FIGURA A.3: Una subgráfica resaltada en gris

Si $G = (V, E)$ es una gráfica, la **subgráfica inducida** por el conjunto de vértices $V' \subseteq V$ es $G' = (V', E')$ donde $E' = \{u-v \in E \mid u, v \in V'\}$, es decir que se construye a partir de los vértices de V' colocando todas las aristas posibles de la gráfica original.

Un **camino** en una gráfica es una sucesión de vértices conectados por aristas. La **longitud** de este camino es la longitud de la sucesión. En un **camino simple** no se repite ningún vértice. Si hay un camino del vértice u al vértice v entonces decimos que v es **alcanzable desde u** y lo denotamos por $u \rightsquigarrow v$. Convenimos que todo vértice es alcanzable desde sí mismo; es decir, $u \rightsquigarrow u$ para todo vértice u . Una gráfica es **conexa** (o **conectada**) si todos los vértices son alcanzables entre sí, una gráfica que no es conexa es **disconexa** (o **desconectada**). Un camino de longitud tres o más que comienza y termina en el mismo vértice es un **ciclo**. En un **ciclo simple** no se repite ningún vértice excepto el primero. La relación $u \rightsquigarrow v$ define una relación de equivalencia; a sus clases de equivalencia se les llama **componentes conexas**.

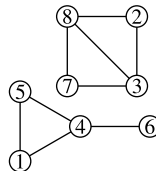


FIGURA A.4: Una gráfica disconexa (no existe ningún camino $1 \rightsquigarrow 2$ por ejemplo). Sus dos componentes conexas son $\{1, 4, 5, 6\}$ y $\{2, 3, 7, 8\}$.

Hay ciertos tipos de gráficas que son tan comunes que merecen su nombre propio. Una gráfica es...

- ... **completa** si existe una arista entre cada par de vértices. A la gráfica completa de n vértices se le denota por K_n .
- ... **bipartita** si su conjunto de vértices se puede partir en dos subconjuntos V_0 y V_1 de tal forma que cada arista tiene un extremo en V_0 y el otro en V_1 .
- ... **bipartita completa** si es una gráfica bipartita tal que para cada pareja $u \in V_0$ y $v \in V_1$ existe una arista $u-v$. Denotamos a la gráfica bipartita completa de $n + m$ vértices como $K_{n,m}$.

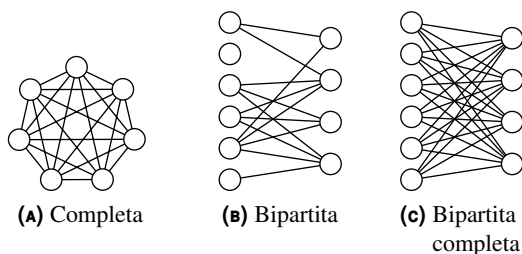


FIGURA A.5: Ejemplos de gráficas comunes

En muchas ocasiones no nos interesa la naturaleza de los vértices tanto como la manera en que están conectados. Diremos que dos gráficas G_1 y G_2 son **isomorfas** si existe una función

$$\phi : V(G_1) \rightarrow V(G_2)$$

tal que $u-v \in E(G_1)$ si y solo si $\phi(u)-\phi(v) \in E(G_2)$. Intuitivamente esto significa que G_1 y G_2 son esencialmente la misma gráfica salvo que cambian las etiquetas de los vértices (y la función ϕ nos dice cómo cambian las etiquetas).

ÁRBOLES

Una gráfica simple y sin ciclos se llama **bosque**; si además es conexa entonces diremos que es un **árbol**. Otras formas equivalentes de decir que T es un árbol son:

1. T no tiene ciclos, y al agregar cualquier arista se forma un ciclo simple (que no repite vértices)

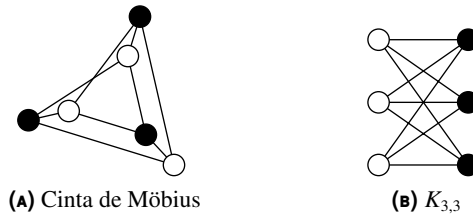


FIGURA A.6: Aparentemente distintas, en realidad estas gráficas son isomorfas. Los vértices fueron coloreados de blanco y negro para facilitar la comparación entre las dos gráficas.

2. Cualesquiera dos vértices de T están conectados por un único camino simple (sin repetir vértices)
3. T es conexa, tiene n vértices y $n - 1$ aristas
4. T no tiene ciclos, tiene n vértices y $n - 1$ aristas

Un **árbol enraizado** tiene un vértice distinguido llamado **raíz**. Cada vértice que aparece en camino de la raíz hasta un vértice v es un **ancestro** de v . Si u es un ancestro de v también decimos que v es un **descendiente** de u . El **subárbol enraizado** en v es la subgráfica inducida por v y todos sus descendientes. Si la última arista del camino simple que va de la raíz r a un vértice x es $y-x$, entonces y es el **padre** de x , o x es el **hijo** de y . La raíz es el único vértice que no tiene padre. Dos vértices con el mismo padre son **hermanos** y un vértice sin hijos es una **hoja**. A los vértices que no son hojas se les llama vértices **de ramificación** o vértices **intermedios**. Una **rama** de un árbol enraizado es un camino $r \rightsquigarrow h$ donde r es la raíz y h es una hoja. La **profundidad** de un vértice es la longitud del camino que hay entre la raíz y dicho vértice.

La terminología asemeja a un árbol familiar de bacterias: cada vértice tiene un padre pero no tiene madre.

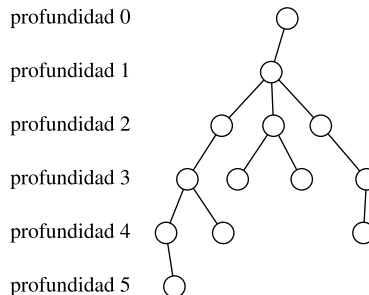


FIGURA A.7: Un árbol enraizado

LÓGICA BOOLEANA

La lógica booleana es un sistema matemático construido al rededor de los valores VERDADERO y FALSO (los cuales son denominados **valores Booleanos**). Originalmente fue concebida como matemática pura, pero actualmente es el fundamento de la electrónica digital y tiene mucha utilidad en programación de computadoras.

Podemos manipular los valores Booleanos con operaciones especialmente diseñadas llamadas **operaciones Booleanas**. La operación más sencilla es la **negación** o simplemente **no**, representada por el símbolo \neg . Así tenemos que $\neg \text{VERDADERO} = \text{FALSO}$ y $\neg \text{FALSO} = \text{VERDADERO}$. La **conjunción**, o **y**, se denota por \wedge . La conjunción de dos valores Booleanos es VERDADERO si ambos valores son VERDADERO. La **disyunción**, también llamada **o**, se denota por \vee . La disyunción de dos valores booleanos es VERDADERO si al menos uno de esos valores es VERDADERO. Resumimos esta información como sigue:

p	q	$p \wedge q$	$p \vee q$	$\neg q$
VERDADERO	VERDADERO	VERDADERO	VERDADERO	FALSO
VERDADERO	FALSO	FALSO	VERDADERO	VERDADERO
FALSO	VERDADERO	FALSO	VERDADERO	
FALSO	FALSO	FALSO	FALSO	

A.2. ELEMENTOS DE ALGORITMIA

PROBLEMAS COMPUTACIONALES

Un **problema computacional** Q es la relación binaria que hay entre un conjunto \mathcal{I} de **instancias** y otro \mathcal{S} de **soluciones**. Usualmente estos conjuntos se definen implícitamente mediante enunciados. Por ejemplo, el **problema de seleccionar el máximo** se define como sigue:

Instancia Un conjunto finito $C \neq \emptyset$ y una relación de orden \leq .

Solución El elemento $x \in C$ tal que $y \leq x$ para todo $y \in C$.

Aquí $\{1, 4, 6, 2, 3\}$ es una instancia con solución única 6, pero en general no hay ninguna restricción en cuanto a la cantidad de soluciones asociadas a una instancia.

Existen varios tipos de problemas computacionales. En un **problema de decisión** las únicas soluciones posibles son VERDADERO o FALSO.

Es común describir este tipo de problemas usando la palabra *pregunta* en lugar de *solución* (más adelante vemos un ejemplo). Asimismo los problemas de decisión también tienen una representación canónica en forma de un conjunto \mathcal{L} (usualmente denominado **lenguaje** en teoría de la computación) definido por

$$\mathcal{L} = \{i \in \mathcal{I} \mid i \text{ tiene solución VERDADERO}\}$$

Si las soluciones son tomadas de un conjunto arbitrario y solo nos interesa encontrar una solución cualquiera, entonces estamos hablando de un **problema de búsqueda**. Finalmente, en un **problema de optimización** se cuenta con una **función objetivo** $f : \mathcal{S} \rightarrow \mathbb{R}$ que mide la calidad $f(s)$ de la solución s . Dada una instancia i , se desea encontrar una **solución factible** s (*factible* significa que s es una solución de i) que maximiza el valor de $f(s)$, es decir

$$f(s) = \max \{f(s') \mid s' \text{ es solución de } i\}.$$

Un ejemplo clásico de problema de decisión es el de la primalidad:

Instancia Un número entero $n \geq 1$

Pregunta ¿ n tiene exactamente dos divisores diferentes 1 y n ?

La representación de este problema como lenguaje es

$$PRIMOS = \{n \mid n \text{ es un número primo}\}.$$

Este problema está muy relacionado con el de factorización, que es un problema de búsqueda:

Instancia Un número entero $n \geq 1$

Solución Un entero $q > 1$ que divide a n ; $q = 1$ si no existe tal divisor.

ALGORITMOS Y PSEUDOCÓDIGO

Casi todos los problemas interesantes tienen una infinidad de instancias. Sin embargo, tener una lista infinita de todas las soluciones es igual de bueno que tener un método para obtener una solución a partir de la instancia. Para ello nos valemos de **algoritmos**: procedimientos que se pueden describir de manera precisa mediante un texto finito, reciben datos de entrada, funcionan paso a paso, y posiblemente arrojan una salida.

Esta concepción de *algoritmo* fue adaptada de Dershowitz & Gurevich (2008).

Cuando decimos que un algoritmo funciona paso a paso nos referimos a que el algoritmo determina una sucesión de estados “computacionales” para cada entrada válida. Cada estado computacional queda determinado de manera única por los valores asociados a un conjunto de variables. Este conjunto de variables es fijo, finito, y solamente depende del algoritmo, no de la entrada.

Dado que los algoritmos pueden recibir datos de entrada y arrojar datos de salida, es común pensar en algoritmos como funciones. Si el algoritmo A se ejecuta sobre la entrada x , hace un número finito de operaciones y produce la salida y , nosotros escribimos

$$A(x) = y$$

y decimos que A **responde** (o **devuelve**) y ante la entrada x . Sin embargo A podría no terminar en un número finito pasos ante la entrada x . En este caso decimos que A entra a un **bucle infinito** y $A(x)$ queda sin definir. Una función $f : D \rightarrow R$ es **computable** (o **calculable**) si existe un algoritmo A tal que $A(x) = f(x)$ para todo $x \in D$.

Dado un problema computacional Q , decimos que un algoritmo A **resuelve** Q si A encuentra correctamente las soluciones de todas las instancias de Q , es decir que para toda instancia i el valor de $A(i)$ está bien definido y es una solución de i . Cuando Q se sobreentiende del contexto simplemente decimos que A es **correcto**.

La descripción que usaremos para los algoritmos es **pseudocódigo**, llamado así porque asemeja un lenguaje de programación. Hacemos la convención de que todo enunciado imperativo que sea evidentemente computable se considera un algoritmo. A continuación veremos cómo combinar algoritmos para formar otros. En lo que sigue, A y B son algoritmos, y P es alguna propiedad (que se puede evaluar a VERDADERO o FALSO):

- *Estructura secuencial.* La ejecución del algoritmo A va seguida de la ejecución del algoritmo B .

A
 B

- *Estructura condicional.* Si P entonces se ejecuta A y en caso contrario el algoritmo termina.

si P :
 A

- *Estructura repetitiva.* Se evalúa P ; si $P = \text{VERDADERO}$ entonces se ejecuta A ; al término de su ejecución se repite el algoritmo desde la evaluación de P y así sucesivamente hasta que P sea FALSO.

mientras P :

└ A

La instrucción de **asignación** sirve para cambiar el valor de una o más variables. Esta instrucción se representa como

$x \leftarrow f$

y significa “asigne a la variable x el valor que resulta de evaluar la expresión f ”. Las variables pueden tomar como valor cualquier objeto que resulte de evaluar f . Por ejemplo $\varphi \leftarrow \frac{1+\sqrt{5}}{2}$ asigna a la variable φ el número real $\frac{1+\sqrt{5}}{2}$ y luego la instrucción $S \leftarrow \{2, 3\} \cup \{\varphi\}$ asigna a la variable S el conjunto $\left\{2, 3, \frac{1+\sqrt{5}}{2}\right\}$.

El teorema fundamental de la programación estructurada nos garantiza que todo algoritmo se puede construir usando solamente las tres estructuras antes descritas y el operador de asignación, sin embargo por comodidad y legibilidad es muy común a definir nuevas operaciones, constantes, o introducir ligeras variantes de las estructuras como veremos a continuación.

La operación de **retorno** sirve para terminar un algoritmo y producir una salida; se denota por

responder f

y causa que el algoritmo arroje como salida el resultado de evaluar f . También introducimos una nueva constante especial NINGUNO que no representa ningún valor.

Una variante de la estructura condicional agrega una clausula “en otro caso” como sinónimo de “si no P ”, de tal forma que cuando P es FALSO el flujo de la ejecución pasa al algoritmo B :

si P :

└ A

en otro caso:

└ B

También es común usar una variante de la estructura repetitiva que es especialmente útil para examinar los datos de un conjunto S :

para cada x elemento de S :

└ A

Dicho conjunto S puede ser definido de manera explícita o implícita. Normalmente esperaríamos que el algoritmo A realice algo útil con la variable x . Esta extensión en realidad es solamente una abreviación del siguiente pseudocódigo:

```

 $x \leftarrow$  primer elemento de  $S$ 
mientras  $x \neq \text{NINGUNO}$  :
     $A$ 
     $x \leftarrow$  siguiente elemento de  $S$  o NINGUNO si no hay

```

La instrucción de retorno solo tiene sentido cuando el algoritmo tiene un nombre que lo identifica. Para aclarar el concepto veamos un ejemplo. La sucesión de Fibonacci f_0, f_1, f_2, \dots comienza con $f_0 = 0, f_1 = 1$, y a partir de ahí los demás elementos se calculan como la suma de los dos anteriores, es decir, $f_n = f_{n-1} + f_{n-2}$. El siguiente pseudocódigo define un algoritmo nombrado **FIB** que recibe el valor de la variable n como dato de entrada y produce como salida **FIB** (n) = f_n :

ALGORITMO A.1: **FIB**(n)

```

1  $(i, j) \leftarrow (1, 0)$ 
2 para cada  $k$  desde 1 hasta  $n$  :
3    $(i, j) \leftarrow (j, i + j)$ 
4 responder  $j$ 

```

Demostraremos que este algoritmo es correcto (en este caso necesitamos mostrar que **FIB** (n) = f_n para todo $n \in \mathbb{N}$). La primera instrucción asigna $(i, j) = (f_{-1}, f_0)$ donde $f_{-1} = 1$ es consistente con la definición de f_n . Ahora bien, si $(i, j) = (f_{k-2}, f_{k-1})$ son dos números consecutivos de la sucesión de Fibonacci entonces $(j, i + j) = (f_{k-1}, f_k)$ es el siguiente par de números consecutivos de la sucesión. Por lo tanto, al término de cada iteración k -ésima la variable $j = f_k$. La estructura repetitiva termina cuando $k = n$, por lo tanto el valor que devuelve este algoritmo es $j = f_n$.

Darle nombre a los algoritmos también nos permite describir **algoritmos recursivos**, que funcionan de la siguiente manera: si la instancia es muy pequeña se resuelve directamente, y en otro caso se define la solución como una función de instancias más pequeñas. Por ejemplo en el siguiente algoritmo **FIB'** (n) los casos $n = 0$ y $n = 1$ se resuelven directamente y los demás casos se resuelven calculando primero **FIB'** ($n - 1$) y **FIB'** ($n - 2$) y luego sumando sus resultados:

ALGORITMO A.2: $\text{FIB}'(n)$

```

1 si  $n = 0$  o  $n = 1$  :
2   responder  $n$ 
3 en otro caso:
4   responder  $\text{FIB}'(n - 1) + \text{FIB}'(n - 2)$ 

```

Este algoritmo calcula f_n porque es precisamente la definición de f_n , solamente que está escrita en pseudocódigo.

ANÁLISIS DE ALGORITMOS

El número de pasos que realiza un algoritmo ante una entrada dada puede depender de muchos parámetros; por ejemplo, si la entrada es una gráfica, entonces el número de pasos puede depender del número de vértices, del número de aristas, o una combinación de ambos. Por simplicidad describiremos la **complejidad temporal** de un algoritmo en función del **tamaño de la entrada**, es decir, de la cantidad de datos necesarios para describir una instancia dada. Hay varias posibles funciones, pero nosotros estamos interesados en el **análisis del peor caso**: para este análisis a cada algoritmo A le vamos a asociar la función de complejidad temporal $t_A : \mathbb{N} \rightarrow \mathbb{N}$, donde $t_A(n)$ es el número máximo de **operaciones elementales** que utiliza el algoritmo A ante una entrada cualquiera de tamaño n .

Formalmente un **dato** es una sucesión de bits de tamaño fijo.

Dado que el tiempo exacto de ejecución normalmente tiene una expresión bastante complicada, nos conformamos con estimarlo. En lugar de decir que un algoritmo realiza como máximo $5n^3 + 4n + 3$ operaciones ante una entrada de tamaño n , es más simple ignorar los términos de orden inferior como $4n$ y 3 (que se vuelven insignificantes conforme n crece), e inclusive el detalle del coeficiente 5 del término principal (de todas formas en unos años las computadoras serán 5 veces más rápidas) y diremos simplemente que el algoritmo toma un tiempo $O(n^3)$ (pronunciado “O-grande de n^3 ”).

Formalmente, dadas dos funciones $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$ (donde \mathbb{R}^+ es el conjunto de los números reales positivos) decimos que $f(n) = O(g(n))$ si hay enteros positivos c y n_0 tales que para todo $n \geq n_0$

$$f(n) \leq c g(n).$$

Cuando $f(n) = O(g(n))$ decimos que g es una **cota superior asintótica** de f , e intuitivamente significa que “ f crece no más rápido que g si descartamos las diferencias hasta un factor constante”.

En la expresión $f(n) = O(g(n))$ el signo igual no tiene el sentido usual de relación de equivalencia; más bien $O(g(n))$ representa a alguna función anónima, para la cual desconocemos su nombre o valor exacto (solamente nos interesa su crecimiento asintótico). En general una ecuación que involucra notación asintótica es correcta cuando podemos sustituir las funciones anónimas por funciones concretas de manera que la ecuación sea satisfecha. Por ejemplo, nos es correcto escribir $2n^2 + 3n + 1 = 2n^2 + O(n)$.

La notación O -grande es útil para comparar algoritmos conforme crece el tamaño de entrada. Por ejemplo, supongamos que deseamos elegir entre dos algoritmos A y B que resuelven el mismo problema, y que tienen complejidades $t_A(n) = n^2$ y $t_B(n) = 2n + 20$ (v. figura A.8). ¿Cuál es el mejor? Depende del valor de n . Si $n \leq 5$ entonces n^2 es más pequeño; de ahí en adelante $2n + 20$ es el contundente ganador. En este caso $2n + 20$ escala mejor conforme n crece, y por tanto es superior.

Esta superioridad está capturada por la notación O -grande: $t_B(n) = O(t_A(n))$ porque

$$\frac{t_B(n)}{t_A(n)} = \frac{2n + 20}{n^2} \leq 22$$

para todo n . Por otro lado $t_A(n) \neq O(t_B(n))$ dado que la razón $t_A(n)/t_B(n) = n^2/(2n+20)$ crece de manera arbitraria, de modo que ninguna constante c hará que funcione la definición.

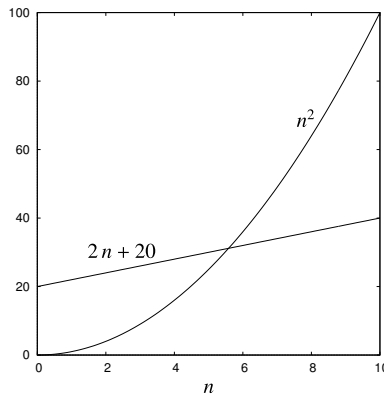


FIGURA A.8: Comparando tiempos de ejecución

La notación O -grande interactúa de manera especial con las funciones logarítmicas. Para cualquier $b > 1$ tenemos que

$$\log_b(n) = \frac{\log_2(n)}{\log_2(b)}$$

de forma que $\log_b(n) = O(\log_2(n))$. Por lo tanto es normal omitir la base y simplemente escribir $O(\log(n))$.

Así como $O(\cdot)$ es un análogo de “ \leq ”, podemos definir análogos de “ \geq ” y “ $=$ ” como sigue:

$$f(n) = \Omega(g) \text{ significa } g(n) = O(f(n))$$

$$f(n) = \Theta(g(n)) \text{ significa } f(n) = O(g(n)) \text{ y } g(n) = O(f(n)).$$

Como ejemplo mostraremos que $\log(n!) = \Theta(n \log n)$. Por un lado tenemos que $\log(n!) = O(n \log n)$ porque

$$\log(n!) = \log\left(\prod_{k=1}^n k\right) \leq \log\left(\prod_{k=1}^n n\right) = \sum_{k=1}^n \log(n) = n \log n.$$

Por otro lado $\log(n!) = \Omega(n \log n)$ porque

$$\begin{aligned} \log(n!) &= \log\left(\prod_{k=1}^n k\right) \geq \log\left(\prod_{k=1}^{n/2} k\right) \geq \log\left(\prod_{k=1}^{n/2} n/2\right) = \sum_{k=1}^{n/2} \log(n/2) \\ &= \frac{n}{2} \log \frac{n}{2} = \frac{1}{2} n (\log n - \log 2) = \frac{1}{2} n \log(n) - \frac{\log(2)}{2} n. \end{aligned}$$

Algunas funciones son tan comunes que tienen sus propios nombres según el cuadro A.1. Las funciones $f(n) = \Theta(1)$ no necesariamente son de la forma $f(n) = c$. Por ejemplo $1/n = \Theta(1)$ y $\sin(n) = \Theta(1)$.

Forma Theta	Nombre
$\Theta(1)$	<i>constante</i>
$\Theta(\log n)$	<i>logarítmica</i>
$\Theta(n)$	<i>lineal</i>
$\Theta(n^2)$	<i>cuadrática</i>
$\Theta(n^3)$	<i>cúbica</i>
$n^{\Theta(1)}$	<i>polinomial</i>
$b^{\Theta(n)}, b > 1$	<i>exponencial</i>

CUADRO A.1: Nombres de funciones comunes

En general hay reglas muy sencillas para manipular ecuaciones con notación asintótica (O-grande, Θ u Ω), descartando términos que son dominados por otros términos:

- Las constantes multiplicativas pueden ser omitidas: $14n^2$ pasa a ser n^2 .
- n^a domina n^b si $a > b$. Por ejemplo, n^2 domina n .
- Toda exponencial domina a cualquier polinomio. Por ejemplo 3^n domina n^5 (incluso domina a 2^n).
- De forma similar, todo polinomio domina a cualquier logaritmo. Por ejemplo n domina $(\log n)^3$ y n^2 domina $n \log n$.

A.3. ELEMENTOS DE ÁLGEBRA LINEAL

MATRICES

Una **matriz** A de tamaño $m \times n$ con entradas en un conjunto F es un arreglo rectangular de mn elementos de F organizados en m renglones y n columnas:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1j} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2j} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{ij} & \cdots & a_{in} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mj} & \cdots & a_{mn} \end{bmatrix}.$$

El i -ésimo renglón de A es

$$[a_{i1} \quad a_{i2} \quad \cdots \quad a_{in}]$$

mientras que la j -ésima columna es

$$\begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{bmatrix}.$$

Al conjunto de todas las matrices de tamaño $m \times n$ con entradas en F lo denotaremos por $\mathcal{M}_{m \times n}(F)$. Para el resto de esta sección vamos a restringir nuestra atención a las matrices de entradas reales $\mathcal{M}_{m \times n}(\mathbb{R})$.

El elemento que está en el renglón i y columna j se dice que es el **i, j -ésimo** elemento de A , o la **entrada** (i, j) . El símbolo a_{ij} es, en efecto, una función binaria

$$a : \{1, \dots, m\} \times \{1, \dots, n\} \rightarrow F$$

según se describió en la página 77, y denotamos $A = [a_{ij}]$. Por ejemplo, la **matriz identidad** de tamaño $n \times n$, denotada por I_n se define como $I_n = [a_{ij}]$ donde

$$a_{ij} = \begin{cases} 1 & \text{si } i = j, \\ 0 & \text{en otro caso.} \end{cases}$$

Es decir I_n tiene 1 en las entradas de la **diagonal** $a_{11}, a_{22}, \dots, a_{nn}$ y 0 en todas las demás. Diremos que A es una **matriz diagonal** si $a_{ij} = 0$ para todo $i \neq j$; la matriz I_n es un ejemplo de matriz diagonal.

Otro tipo de matrices de interés son las matrices triangulares:

- $A = [a_{ij}]$ es **triangular superior** si $a_{ij} = 0$ para todo $i > j$, es decir, solo tiene ceros debajo de la diagonal.
- $A = [a_{ij}]$ es **triangular inferior** si $a_{ij} = 0$ para todo $i < j$, es decir, solo tiene ceros encima de la diagonal.

En lo que sigue $A = [a_{ij}]$, $B = [b_{ij}]$, $C = [c_{ij}]$ y $r \in F$. Definimos cuatro operaciones sobre las matrices:

- Si A es de tamaño $m \times n$, su **matriz transpuesta** A^T es la matriz B de tamaño $n \times m$ que se obtiene intercambiando renglones con columnas, es decir:

$$b_{ij} = a_{ji}$$

- Si las matrices A y B son del mismo tamaño $m \times n$, su **suma** $C = A + B$ es otra matriz de tamaño $m \times n$ donde

$$c_{ij} = a_{ij} + b_{ij}.$$

- La **multiplicación por escalar** $rA = B$ se hace multiplicando por r a cada entrada de A , es decir

$$b_{ij} = r a_{ij}.$$

- Finalmente, si A es de tamaño $m \times p$ y B de $p \times n$ definimos la **multiplicación de matrices** $AB = C$ como la matriz de tamaño $m \times n$ dada por

$$c_{ij} = \sum_{k=1}^p a_{ik} b_{kj}.$$

Ilustramos estas operaciones con el siguiente ejemplo:

$$\begin{aligned} 2 \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^T \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} &= \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix} + \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \\ &= \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix} + \begin{bmatrix} 26 & 30 \\ 38 & 44 \end{bmatrix} \\ &= \begin{bmatrix} 28 & 34 \\ 44 & 52 \end{bmatrix} \end{aligned}$$

Note que la multiplicación de matrices no siempre es conmutativa:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix} \neq \begin{bmatrix} 23 & 34 \\ 31 & 46 \end{bmatrix} = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}.$$

Una matriz es **cuadrada** si tiene el mismo número de renglones que de columnas. Una matriz cuadrada A de tamaño $n \times n$ es **invertible** si existe una matriz B de tamaño $n \times n$ tal que $AB = I_n$. En este caso preferimos escribir A^{-1} en lugar de B . Asimismo una matriz cuadrada A es **simétrica** si $A^T = A$.

ESPACIOS VECTORIALES

Los vectores $(a_1, a_2, a_3) \in \mathbb{R}^3$ representan coordenadas en el espacio. A los vectores los podemos sumar unos con otros o escalar (dilatar o encoger) en algún factor r . La suma se realiza componente a componente como

$$(a_1, a_2, a_3) + (b_1, b_2, b_3) = (a_1 + b_1, a_2 + b_2, a_3 + b_3)$$

y el escalamiento se realiza multiplicando cada componente por el valor de escala (escalar):

$$r(a_1, a_2, a_3) = (ra_1, ra_2, ra_3).$$

Un **espacio vectorial real** V es una generalización de \mathbb{R}^3 . Formalmente todo espacio vectorial real debe cumplir estas tres leyes para todo $r, s \in \mathbb{R}$, $\vec{v}, \vec{w} \in V$:

- $r \cdot (\vec{v} + \vec{w}) = r \cdot \vec{v} + r \cdot \vec{w}$
- $(r + s) \cdot \vec{v} = r \cdot \vec{v} + s \cdot \vec{v}$
- $r \cdot (s \cdot \vec{v}) = (rs) \cdot \vec{v}$

A los elementos de V se les llama **vectores** y a los de \mathbb{R} **escalares**. Por conveniencia escribimos la multiplicación por escalar como $r\vec{v}$ en lugar de $r \cdot \vec{v}$.

De hecho, un concepto central del álgebra lineal es el de **espacio vectorial** en general (no necesariamente real). Su definición es similar, excepto que en lugar de \mathbb{R} se utiliza cualquier conjunto F que sea un campo (v. recuadro abajo).

¿QUÉ ES UN CAMPO? Un campo F es un conjunto cuyas propiedades generalizan a las de \mathbb{R} . En F deben estar dos elementos especiales z y u , y dos operaciones binarias $+$ y \cdot que cumplen las siguientes leyes para todos los elementos a, b y c de F :

- *Ley de identidad.* $a + z = a$, $a \cdot u = a$
- *Ley asociativa.* $(a + b) + c = a + (b + c)$, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
- *Ley de inversos.* Para todo a existe otro elemento $-a$ tal que $a + (-a) = z$ y para todo $a \neq z$ existe otro elemento a^{-1} tal que $a \cdot a^{-1} = u$
- *Ley conmutativa.* $a + b = b + a$, $a \cdot b = b \cdot a$
- *Ley distributiva.* $a \cdot (b + c) = a \cdot b + a \cdot c$

Por analogía es común llamar *uno* a u y *cero* a z así como *suma* y *multiplicación* a las operaciones $+$ y \cdot respectivamente, pero F no necesariamente tiene que ser un conjunto de números ni las operaciones tienen que ser las usuales.

Un subconjunto U de un espacio vectorial V es un **subespacio** de V si es cerrado bajo la suma y la multiplicación por escalar, es decir, $\vec{u} + \vec{v}$ y $r\vec{v}$ vuelven a ser elementos de U si $\vec{u}, \vec{v} \in U$. Los subespacios de \mathbb{R}^3 son conjuntos U de puntos que forman rectas o planos que pasan por el punto $\vec{0} = (0, 0, 0)$. Una sucesión finita de vectores $\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n$ son **linealmente dependientes** si existen escalares r_1, r_2, \dots, r_n , no todos cero, tales que $r_1 \vec{v}_1 + r_2 \vec{v}_2 + \dots + r_n \vec{v}_n = \vec{0}$; de otro modo decimos que

$\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n$ son **linealmente independientes**. El subespacio generado por los vectores $\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n$ es el conjunto de todas las combinaciones lineales

$$\langle \vec{v}_1, \vec{v}_2, \dots, \vec{v}_n \rangle = \{r_1 \vec{v}_1 + r_2 \vec{v}_2 + \dots + r_n \vec{v}_n \mid r_1, r_2, \dots, r_n \in F\}.$$

Un espacio vectorial V es **de dimensión finita** si es generado por una cantidad finita de vectores. Una n -ada $(\vec{v}_1, \dots, \vec{v}_n)$ de elementos en V es una **base** de V si los vectores son linealmente independientes y generan a V , es decir, $\langle \vec{v}_1, \dots, \vec{v}_n \rangle = V$. Un teorema central en álgebra lineal es que cualquier espacio vectorial V finitamente generado tiene una base finita, y que todas las bases tienen la misma cantidad de elementos, llamada **dimensión** $\dim V$ de V . Por ejemplo $\dim F^3 = 3$ y una base de F^3 está dada por los vectores unitarios $(1, 0, 0)$, $(0, 1, 0)$ y $(0, 0, 1)$. Con respecto a la base $(\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n)$, todo vector $\vec{v} \in V$ tiene una única representación $\vec{v} = r_1 \vec{v}_1 + \dots + r_n \vec{v}_n$ como combinación lineal de elementos de la base, con **coordenadas** $r_1, r_2, \dots, r_n \in F$.

TRANSFORMACIONES LINEALES

Una función $f : V \rightarrow W$ entre dos espacios vectoriales V y W sobre el mismo campo F es **F -lineal** si $f(\vec{v}_1 + \vec{v}_2) = f(\vec{v}_1) + f(\vec{v}_2)$ y $f(r \vec{v}_1) = r f(\vec{v}_1)$ para todos los $\vec{v}_1, \vec{v}_2 \in V$ y $r \in F$. A las funciones F -lineales también se les conoce como **transformaciones lineales** u **homomorfismos**.

Si V y W son espacios vectoriales sobre F con bases (v_1, v_2, \dots, v_n) y (w_1, w_2, \dots, w_m) respectivamente, entonces a toda transformación $f : V \rightarrow W$ corresponde la matriz $A = [a_{ij}]$ de tamaño $m \times n$ definida por

$$f(v_i) = a_{1i} w_1 + \dots + a_{mi} w_m$$

de manera que

$$A \begin{bmatrix} r_1 \\ \vdots \\ r_n \end{bmatrix} = \begin{bmatrix} t_1 \\ \vdots \\ t_m \end{bmatrix} \iff f(r_1 v_1 + \dots + r_n v_n) = t_1 w_1 + \dots + t_m w_m.$$

Si A_f y A_g son las matrices asociadas a las transformaciones lineales f y g respectivamente, entonces la matriz asociada a la transformación $f \circ g$ es $A_{f \circ g} = A_f A_g$.

OPERACIONES ELEMENTALES SOBRE MATRICES

Cualquiera de las siguientes son **operaciones elementales por renglones** sobre una matriz $A = [a_{ij}]$ de tamaño $n \times n$:

- Intercambiar los renglones r y s de A . Es decir,

para j desde 1 hasta n :
 $\quad \lfloor (a_{rj}, a_{sj}) \leftarrow (a_{sj}, a_{rj})$

- Multiplicar el renglón r por una constante $c \neq 0$. Es decir,

para j desde 1 hasta n :
 $\quad \lfloor a_{rj} \leftarrow c a_{rj}$

- Sumar d veces el renglón s de A al renglón r de A (con $r \neq s$). Es decir,

para j desde 1 hasta n :
 $\quad \lfloor a_{rj} \leftarrow a_{rj} + d a_{sj}$

A continuación veremos que las operaciones elementales sobre los renglones son transformaciones lineales. Defínase las **matrices elementales** E_{rs} , E_r^c y E_{rs}^d como las matrices que se obtienen de la matriz identidad de tamaño $n \times n$ aplicando respectivamente las operaciones de intercambiar del renglón r con el renglón s , multiplicar el renglón r por c y sumar d veces el renglón s al renglón r . Por ejemplo con $n = 4$ tenemos

$$E_{23} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad E_3^8 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad E_{24}^7 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 7 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Así pues, la transformación $T_{rs}(A) = E_{rs}A$ equivale a intercambiar el renglón r con el renglón s , la transformación $T_r^c(A) = E_r^cA$ multiplica el renglón r de A por c y finalmente la transformación $T_{rs}^d(A) = E_{rs}^dA$ suma d veces el renglón s de A al renglón r de A .

Las matrices elementales tienen las siguientes propiedades:

- La matriz identidad I es una matriz elemental (es E_r^1 para todo r)
- La matriz inversa de E_{rs} es ella misma, es decir $E_{rs}E_{rs} = I$
- La matriz inversa de E_r^c es $E_r^{1/c}$

- La matriz inversa de E_{rs}^d es E_{rs}^{-d}

Esto muestra que las operaciones elementales se pueden revertir usando otras operaciones elementales.

Se dice que una matriz de $m \times n$ es **equivalente por renglones** a una matriz B de $m \times n$, si B se puede obtener al aplicar a la matriz A una serie finita de operaciones elementales por renglones. La equivalencia por renglones se llama así porque en verdad define una relación de equivalencia:

- Toda matriz es equivalente por renglones a sí misma porque $IA = A$, pero I es una matriz elemental.
- Si A es equivalente por renglones a B entonces B es equivalente por renglones a A . En efecto, supongamos que $E_m \cdots E_2 E_1 A = B$ donde E_1, E_2, \dots, E_m son matrices elementales; sea E'_k la matriz inversa de E_k . Ya hemos visto que la inversa de toda matriz elemental existe y es otra matriz elemental. Por lo tanto despejando A tenemos que $E'_1 E'_2 \cdots E'_m B = A$, de donde B es equivalente por renglones a A .
- Si A es equivalente por renglones a B y B es equivalente por renglones a C entonces A es equivalente por renglones a C . En efecto, si $E_m \cdots E_2 E_1 A = B$ y $F_{m'} \cdots F_2 F_1 B = C$ donde cada E_k y cada F_k son matrices elementales, entonces sustituyendo la expresión para B se obtiene que $F_{m'} \cdots F_2 F_1 E_m \cdots E_2 E_1 A = C$

Un teorema muy importante del álgebra lineal es que toda matriz A de $n \times n$ es invertible si y solamente si es equivalente por renglones a la matriz identidad de $n \times n$.

BIBLIOGRAFÍA

- Barot, Michael. 1999. A characterization of positive unit forms of Dynkin type A_n . *Boletín de la Sociedad Matemática Mexicana*, **5**, 87–93.
- Barot, Michael. 2001. A characterization of positive unit forms, Part II. *Boletín de la Sociedad Matemática Mexicana*, **7**, 13–22.
- Barot, Michael, & de la Peña, José Antonio. 1999. The Dynkin type of a non-negative unit form. *Expositiones Mathematicae*, **17**, 339–348.
- Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., & Stein, Clifford. 2009. *Introduction to Algorithms*. MIT Press. Chap. Elementary Graph Algorithms.
- Dasgupta, Sanjoy, Papadimitriou, Christos, & Vazirani, Umesh. 2008. *Algorithms*. McGraw-Hill.
- Dershowitz, Nachum, & Gurevich, Yuri. 2008. A natural axiomatization of computability and proof of Church's Thesis. *Bulletin of Symbolic Logic*, **14:3**, 299–350.
- Gabriel, Peter, & Roĭter, Andreĭ Vladimirovich. 1997. *Representations of finite-dimensional algebras*. Springer. Chap. Roots.
- Grimaldi, Ralph P. 1998. *Matemáticas Discreta y Combinatoria*. Pearson.
- Hopcroft, John, & Tarjan, Robert. 1973. Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, **16**, 372–378.
- Lay, David C. 2001. *Álgebra lineal y sus aplicaciones*. Pearson.

Ringel, Claus Michael. 1985. *Tame Algebras and Integral Quadratic Forms*. Springer.

von zur Gathen, Joachim, & Gerhard, Jürgen. 2003. *Modern computer algebra*. Cambridge University Press. Chap. Gröbner bases.

ÍNDICE ALFABÉTICO

- árbol, 82
 - binario, 52
 - de recubrimiento, 41
 - de recubrimiento en profundidad, 41
 - enraizado, 83
- A-bloque, 27
- algoritmo, 85
 - recursivo, 88
- ancestro, 83
- arista, 80
 - de árbol, 43
 - de retroceso, 43
 - punto, 53
 - punteada, 7
 - sólida, 7
- bosque, 82
 - de recubrimiento, 41
- cambio de variable, 3
 - entero, 6
- camino, 81
 - hamiltoniano, 52
- ciclo, 81
 - fundamental, 53
 - fundamental maximal, 53
- coloreo de vértices, 48
- columna, 92
- complejidad temporal, 89
- componente
 - biconexa, 35
 - conexa, 81
- conjunto, 73
- deflación, 19
- descendiente, 83
- dominio, 76
- forma cuadrática, 1
 - definida positiva, 5
 - entera, 6
 - unitaria, 6
- forma unitaria, 6
 - simple, 12
- función, 76
 - m -aria, 77
 - biyectiva, 76
 - inyectiva, 76
 - sobreyectiva, 76
- gráfica, 80
 - k -conexa, 30
 - biconexa, 30
 - bipartita, 82
 - bipartita completa, 82
 - circular, 29
 - completa, 82
 - conexa, 81
 - de Dynkin, 7
 - disconexa, 81

- simple, 80
- triconexa, 31
- grado de un vértice, 80
- hijo, 83
- inflación, 19
- isomorfismo de gráficas, 82
- Lagrange, reducción de, 22
- lazo, 80
- lista de adyacencia, 38
- matriz, 92
 - asociada a una forma cuadrática, 3
 - cuadrada, 94
 - de adyacencia, 37
 - definida positiva, 6
 - elemental, 97
 - invertible, 94
 - simétrica, 94
 - triangular, 93
- multigráfica, 80
- padre, 83
- partición, 79
- pila, 44
- postorden, 42
- predicado, 78
- preorden, 42
- problema computacional, 84
 - de decisión, 84
- propiedad, 78
- pseudocódigo, 86
- punto de articulación, 35
- rama, 83
- rango, 76
- recorrido en profundidad, 40
- relación
 - m -aria, 78
 - binaria, 78
 - de equivalencia, 78
- renglón, 92
- seudográfica, 80
- subárbol enraizado, 83
- subconjunto, 73
- subgráfica, 81
 - inducida, 81
- sucesión, 74
- tipo Dynkin, 8
- transformación lineal, 96
- vértice, 80
 - adyacente, 80
 - de ramificación, 83
 - descubrimiento de, 42
 - finalización de, 42
 - intermedio, 83
- \mathbb{Z} -equivalencia, 6