UNIVERSIDAD AUTÓNOMA DE QUERÉTARO

FACULTAD DE INFORMÁTICA

ALGORITMOS PARA EL CÁLCULO DE INVARIANTES EN GRAFOS

T E S I S

QUE PARA OBTENER EL TÍTULO DE

LICENCIADO EN INFORMÁTICA

PRESENTA

MA. ELENA VÁZQUEZ HUERTA

DIRIGIDA POR

DR. JAIME RANGEL MONDRAGÓN

SANTIAGO DE QUERÉTARO, DICIEMBRE 2001.

PREFACIO

Actualmente, la informática ha sido un campo de estudio bastante amplio y desarrollado. Con ello se han abierto áreas de especialidad para poder abarcarla mayormente. Una de estas áreas importantes son las Ciencias Computacionales, estudio dirigido hacia la investigación, y con esto hacia estudios de posgrado. Estudiar Ciencias Computacionales distingue a un estudiante del resto, ya que no todos contamos con la capacidad de desempeñarnos en el área de investigación, y un buen principio para dirigirnos por ese camino es realizar un trabajo de tesis.

Realizar un trabajo de tesis en licenciatura no es una tarea fácil, esto se complica más cuando no se tienen las bases metodológicas para la investigación. Afortunadamente esa preparación la adquirí con la realización de los estudios de maestría que actualmente realizo. Una buena carta de presentación, es el desarrollo de un trabajo de tesis, lo cual habla de nuestra capacidad de análisis, investigación y logro.

Tampoco es tarea fácil analizar y entender temas avanzados que no se incluyen en cursos de licenciatura. Confieso que la mayor parte de los conocimientos adquiridos fueron nuevos y me costó trabajo asimilarlos. Sin embargo, es una experiencia bastante agradable, exponer nuestras ideas en foros nacionales y con gente especialista. Conocer y entender cada uno de esas pequeñas partes del tema, da una enorme satisfacción personal.

He mencionado algunas dificultades encontradas en la realización de esta tesis, pero gracias a las siguientes personas logré concluirla satisfactoriamente.

RESUMEN

Las nociones de isomorfismo y simetría son consideradas los conceptos más importantes de todas las áreas de la computación. La detección de estructuras isomorfas es importante en la construcción de algoritmos prácticos.

A finales de la década de los 70´s y principios de los 80´s hubo grandes avances en la solución del problema de isomorfismo de grafos, estos avances se basaron en la teoría de grupos. La teoría de grupos puede verse como el estudio algebraico de la simetría. Lo que relaciona las estructuras de grafos y grupos es el problema de determinar si dos grafos son los mismos conociendo únicamente la simetría que estos dos grafos poseen. Basándonos en este enfoque, se construyeron algoritmos eficientes (de orden polinomial) para determinar si dos grafos son isomorfos.

Para ilustrar esas ideas se describirá el enfoque de la teoría de grupos para isomorfismo de grafos, analizando la teoría grupos y sus algoritmos. Se discutirá el problema de isomorfismo de grafos, finalizando con un algoritmo que determine cuándo dos grafos son isomorfos. Se presentará un algoritmo general para isomorfismo de grafos que utiliza el método de invariantes o refinamiento repetitivo, posteriormente se presenta un algoritmo que es, en general, superior al anteriormente mencionado, el cual utiliza el método de certificados o etiquetado canónico. Finalmente se verán aplicaciones de los algoritmos en Árboles.

Palabras clave: Isomorfismo, Árboles, Certificados, Invariantes, Teoría de Grupos, Teoría de Grafos.

ABSTRACT

Among the most important concepts that are common to all areas of computer science are isomorphism and symmetry. The detection of isomorphic structures results essential in the construction of practical algorithms.

In the late 1970´s and early 1980´s tremendous progress was made towards the graph isomorphism problem using an approach based on group theory. Group theory can be thought of as an algebraic study of simmetry. The lovely insight that connects structures of graph and groups is the problem of determiny whether two graphs are the same using only the symmetries that the two graphs possess. By using this approach, efficient polynomial algorithms were obtained to determine whether graphs from several important classes are isomorphic.

To illustrate these ideas, in this work we describe the group-theoretical approach to graph isomorphism and cover some of the beautiful algorithms and ideas that arose from it. Then, we will discuss the graph isomorphism problem, culminating with an algorithm for determining when two graphs are isomorphic. We will also give a general graph isomorphism algorithm that uses the method of invariants or repeated refinement. An algorithm that is, in general, superior to this one is developed at the end, using the method of certificates or cannonical labeling. Finally, we present an algorithm to determine isomorphism of trees.

Keywords: Isomorphism, Trees, Certificate, Invariants, Group Theory, Graph Theory

ÍNDICE

Dios me acercó al **M.C. Arturo González Gutiérrez** y al **Dr. Jaime Rangel Mondragón**, de quienes he aprendido a ser una buena estudiante y un mejor ser humano. Ellos me enseñaron, con su ejemplo, a sobresalir y ser siempre la mejor luchando por lo que quiero pero sin olvidar los principios morales y éticos que mis padres me han inculcado. Ellos, como mis "padres académico", me mostraron el camino de la investigación advirtiéndome de los obstáculos que podía encontrarme, pero creyendo siempre en mi capacidad de vencerlos. A ellos mi mayor agradecimiento por su enseñanza, su ejemplo, su dedicación y, sobre todo, su amistad y cariño.

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Dentro de las abstracciones matemáticas y computacionales la estructura de grafo y sus algoritmos han tenido una amplia utilización en la solución de problemas tales como la optimización de servicios municipales, rutas óptimas, diseño de redes computacionales, entre otros. En la Química, por ejemplo, el problema de encontrar elementos químicos que poseen la misma estructura pero diferentes propiedades (es decir, los isómeros) es susceptible modelarlo y resolverlo utilizando grafos. Este problema pertenece a una clase más amplia en donde es necesario identificar estructuras isomorfas, es decir, aquellas que tengan la misma estructura.

De lo anterior surge la inquietud de conocer e investigar los algoritmos que calculan Isomorfismo de Grafos, y en particular de árboles, aplicando en ello habilidades combinatorias y conceptos de matemáticas computacionales.

El objetivo de este trabajo de investigación es conocer, analizar y explicar el método de invariantes para la detección de isomorfismo en grafos y árboles, especializándolo en certificados de grafos y árboles y haciendo uso de un lenguaje de programación para la verificación de dicho método.

Para la programación de estas estructuras se ha elegido el lenguaje funcional *Mathematica 4.0* por la facilidad de generar prototipos. Además el lenguaje funcional provee la metodología de programación de alto nivel siendo ésta la base de la nueva generación de lenguajes en paralelo.

En esta investigación se presentan brevemente los fundamentos de la teoría de grafos, árboles, funciones y complejidad. Estos fundamentos contribuyen con las herramientas metodológicas y de análisis del tema que se presenta. Se incluye un capítulo de Teoría de Grupos donde se presentan los conceptos básicos para el desarrollo de los invariantes. Uno de los dos capítulos de mayor realce en este trabajo de investigación es el de Isomorfismo, en él se detalla su importancia y se discutirá el problema del isomorfismo de grafos, esto es, dos grafos son estructuralmente iguales o isomorfos si son idénticos excepto por el nombre de los vértices. Asimismo se presenta un algoritmo general para isomorfismo de grafos que utiliza el método de invariantes o refinamiento repetitivo. Posteriormente se presenta un algoritmo que es, en general, superior al método de invariantes, el cual utiliza el método de certificados o etiquetado canónico. Finalmente se verán aplicaciones de los algoritmos a las estructuras de árboles y grafos. Se presenta también un capítulo referente a árboles, donde se muestra la aplicación de los conceptos analizados en el desarrollo de este trabajo de investigación, en el que se implementan funciones básicas de árboles hasta llegar a la generación de árboles no-isomorfos.

Como resultado de este trabajo de investigación, se presentó el artículo *"Técnicas de Programación Funcional en la Enumeración y generación de Grafos no-isomorfos"*, en el XVI Coloquio de Teoría de las Gráficas, Combinatoria y sus Aplicaciones organizado por la Sociedad Matemática Mexicana. Dicho artículo se incluye *in extenso* al final del trabajo.

CAPÍTULO 1
CONCEPTOS BÁSICOS

1.1. Teoría de Grafos

Muchas situaciones pueden ser descritas por medio de diagramas que consisten principalmente de puntos y líneas. En todos estos diagramas el estudio se centra en el hecho de que dos puntos están enlazados por una línea, más que en la manera de cómo ocurre este enlace. Una abstracción matemática de este tipo de situaciones da lugar al concepto de grafo.

Los grafos son herramientas poderosas para crear modelos matemáticos de una gran complejidad y variedad de situaciones; por lo que la teoría de grafos ha sido vital para analizar y resolver problemas en áreas tan diversas como el diseño de redes de computadoras, planeación urbana y biología molecular. Asimismo, la teoría de grafos ha sido utilizada para encontrar la mejor ruta en el tráfico aéreo y para inventar códigos secretos difíciles de descifrar [17]. En las ciencias sociales también ha sido ampliamente utilizada para representar relaciones interpersonales. Una de sus aplicaciones se describe detalladamente en [16]. Las moléculas químicas también pueden representarse con la ayuda de grafos, siendo útiles para la detección, clasificación y enumeración de isómeros. [3]

Los orígenes de la teoría de grafos surgen con el problema de los *Siete Puentes de Königsberg*, resuelto en 1736 por Leonard Euler. De ahí en adelante se han generado valiosos y significativos problemas dentro del mismo campo de la teoría de grafos con aplicaciones a problemas reales, tales como planaridad de grafos, la conjetura de los cuatro colores, optimización de rutas, el problema del agente viajero, entre algunas aplicaciones. [9]

Formalmente hablando, un grafo (G) es un conjunto de puntos llamados *vértices* o *nodos* (V) y un conjunto finito de líneas o curvas llamadas *aristas* (E). Cada arista conecta ya sea dos vértices diferentes o un vértice consigo mismo.

Si una arista a conecta a dos nodos u y v, se dice que a es *incidente* a u y v y que u y v son *adyacentes*. El *grado* de un vértice es el número de aristas que son incidentes a él. [17]

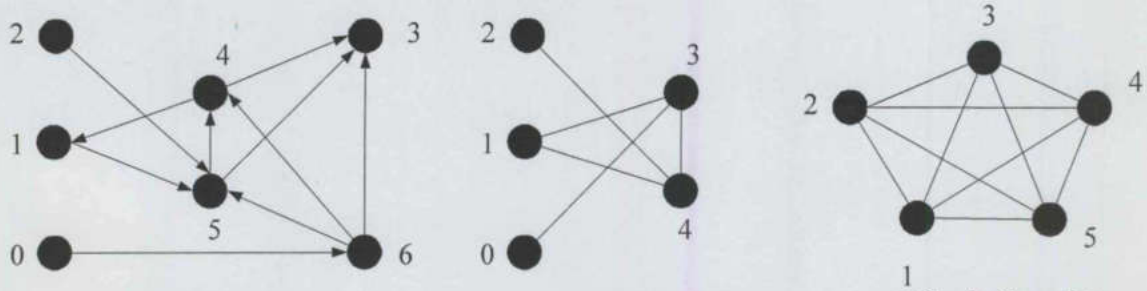
Un grafo *dirigido* es un conjunto V y una relación E asociada a él, esto es; $G=(V,E)$ donde $E \subseteq V \times V$

Un grafo es *no dirigido* si y sólo si E es simétrica, esto es: $(a,b) \in E \Rightarrow (b,a) \in E$ $\forall a,b \in V$

Un grafo es *simple* si y sólo si E es irreflexiva, esto es: $(a,a) \notin E \ \forall a \in V$

Un grafo $G=(V,E)$ es *Completo de Orden n* si y sólo si $|V|=n$ y $(v_i,v_j) \in E$ $\forall v_i,v_j \in V$, $v_i \neq v_j$

En la figura 1.1 se muestran algunos ejemplos de grafos, los cuáles particularmente son grafos simples.



a) Grafo dirigido b) Grafo no dirigido c) Grafo Completo

Figura 1.1 Ejemplos de Grafos

En [19] se ilustra una clasificación finita fundamental de grafos.

1.2. Árboles

Los árboles son un tipo especial de grafos y en computación constituyen la estructura no-lineal de árbol más importante para manipular información.

Los árboles tienen una gran variedad de aplicaciones. Por ejemplo, se pueden utilizar para representar fórmulas matemáticas, para organizar adecuadamente la información, para construir un árbol genealógico, para el análisis de circuitos eléctricos y para numerar los capítulos y secciones de un libro.

La terminología que por lo regular se utiliza para el manejo de árboles es la siguiente:

X es *hijo* de Y, sí y solo sí el nodo X es apuntado por Y. También se dice que X es *descendiente* directo de Y.

X es *padre* de Y sí y solo sí el nodo X apunta a Y. También se dice que X es *antecesor* de Y.

Dos nodos serán *hermanos* si son descendientes directos de un mismo nodo.

Se le llama *hoja* o terminal a aquellos nodos que no tienen ramificaciones (hijos).

Un *nodo interior* es aquel que no es raíz ni terminal.

El *grado* de un vértice, es el número de descendientes directos de un determinado nodo.

Un grafo $G=(V,E)$ es un árbol si y sólo si es un grafo simple y $\forall a,b \in V$, existe una sola trayectoria simple desde a hasta b.

$G=(V,E)$ es un árbol con raíz si y sólo si existe un vértice designado como raíz. Un árbol con raíz es dirigido si y sólo si existe una trayectoria dirigida desde la raíz hasta cada vértice del árbol.

El árbol más sencillo consta de solamente una línea conectando dos nodos. Tres nodos pueden ser unidos solo de una forma, las otras posibilidades son isomorfas, para formar un árbol, pero cuatro nodos pueden definir dos árboles diferentes y el número de árboles aumenta para cinco y seis nodos. En la figura 1.2 se muestran tales familias de árboles. trabajo de enumeración de árboles es analizado ampliamente en [16] [18]

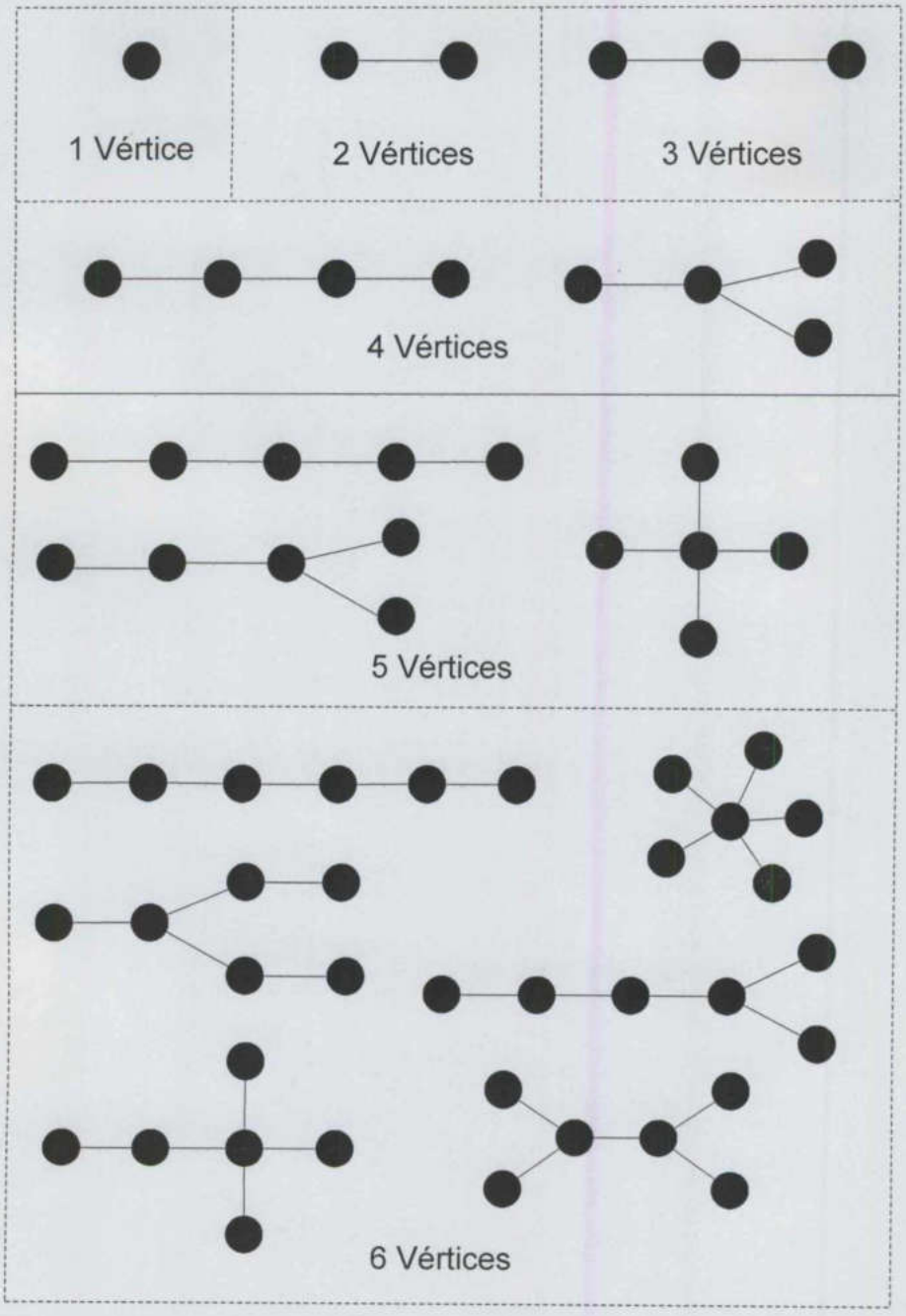


Figura 1.2 *Todos los árboles con 1, 2, 3, 4, 5, y 6 vértices*

La *distancia* entre un par de vértices de un grafo es igual al menor número de aristas en el camino entre ellos.

La *excentricidad* de un vértice *e(v)* en un grafo conexo $G = (V, E)$ se define como la distancia máxima entre u y v, donde $u \in V$.

El *radio* de un grafo conexo *r(G)* es igual a la excentricidad del vértice en G con menor excentricidad.

El *diámetro* de un grafo *d(G)* conexo es la excentricidad del vértice en G con mayor excentricidad.

Un vértice v en un grafo conexo se llama *central* si $e(v) = r(G)$.

El *centro* de un grafo conexo G es el conjunto de todos los vértices centrales en G.

El método para obtener el centro de un árbol es el siguiente:

1. Remover sucesivamente todos los vértices de grado 1 (hojas) junto con sus aristas incidentes. Este paso termina hasta obtener uno o dos vértices

2. Los vértices que no se eliminaron forman el centro del árbol

La figura 1.3 muestra dos ejemplos donde se aplica el método de identificación de centro de un árbol.

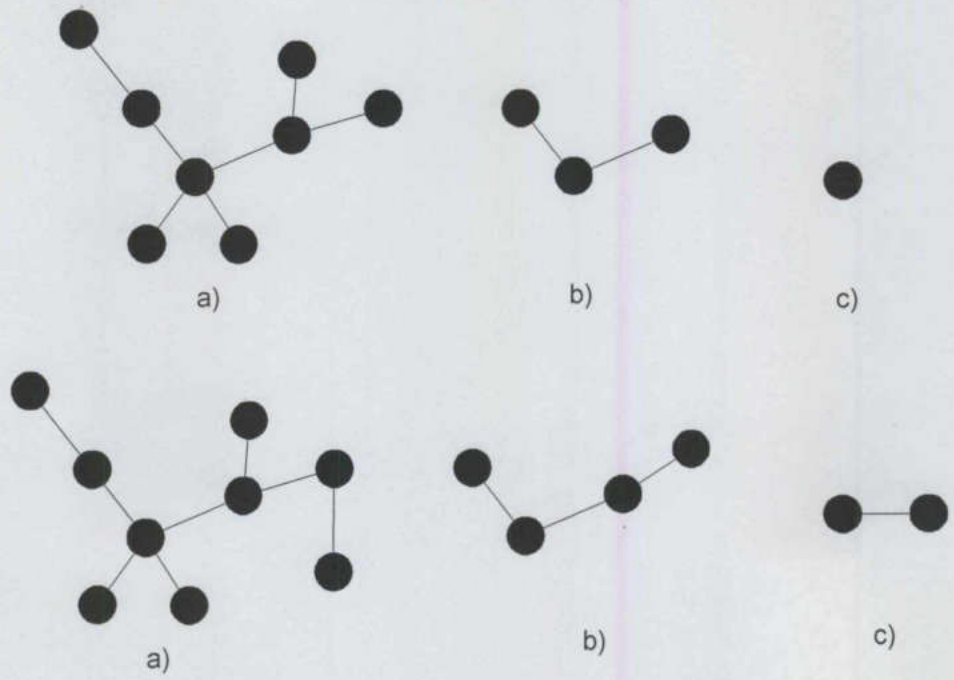


Figura 1.3 *Método para obtener el centro de un árbol*

Teorema: Cada árbol tiene un centro formado por uno o dos vértices adyacentes.

Demostración:

Sean T y T′ árboles con el mismo centro, donde T′ se obtiene al remover todos los vértices hojas de T. Considere la excentricidad de cada vértice de T, de modo que se puede afirmar que la excentricidad de cada vértice en T′ es menor en uno que la correspondiente a T. Por lo tanto, el vértice de mínima excentricidad en T tendrá mínima excentricidad en T′. Entonces T y T′ tienen el mismo centro. Si se remueven sucesivamente todos los vértices hoja de T, resultará una secuencia de árboles que tienen el mismo centro y este proceso llegará a dos posibles resultados: un árbol con 1 o 2 vértices, figura 1.4. □ [10]

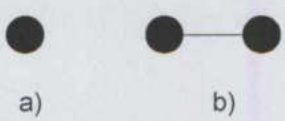


Figura 1.4 *Centro de un árbol de a) un vértice y b) dos vértices*

1.3. Funciones

Uno de los conceptos importante en la ciencia de la computación es el de una función o mapeo de un conjunto a otro. Uno podría decir con seguridad que esta noción juega un papel importante en la definición de paradigmas de programación, como la programación funcional, y en la actualidad se emplea en lenguajes de programación avanzados para describir procesos simulan paralelismo, como el lenguaje *Mathematica.*

Sean S y T conjuntos. Una *función* de S a T, escrita como $f: S \rightarrow T$, es una regla que asigna a cada elemento $s \in S$ un solo elemento $t \in T$. Al conjunto S se le denomina *dominio* de la función y al conjunto T *contradominio* de la función. $f(s) = t$ representa la relación (s, t) donde $s \in S$, recibe el nombre de pre-imagen de t; y t se llama la imagen de s bajo f.

Si S y T son finitos se puede utilizar un diagrama para describir la función $f: S \rightarrow T$. La figura 1.5 describe una función que asocia a cada elemento del conjunto {1, 2, 3, 4} con su correspondiente cuadrado, es decir $f(x) = x^2$.

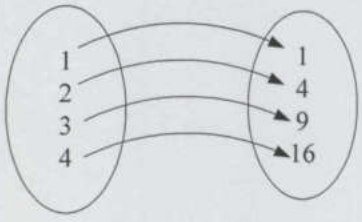
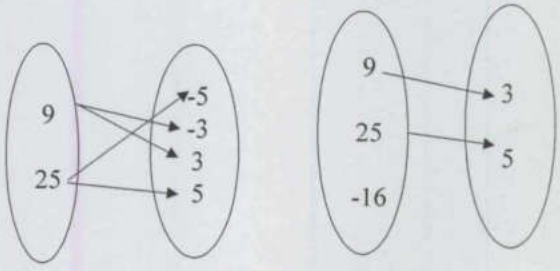


Figura 1.5 *Representación gráfica de funciones*



a) b)

Figura 1.6 *Ejemplos que no son funciones*

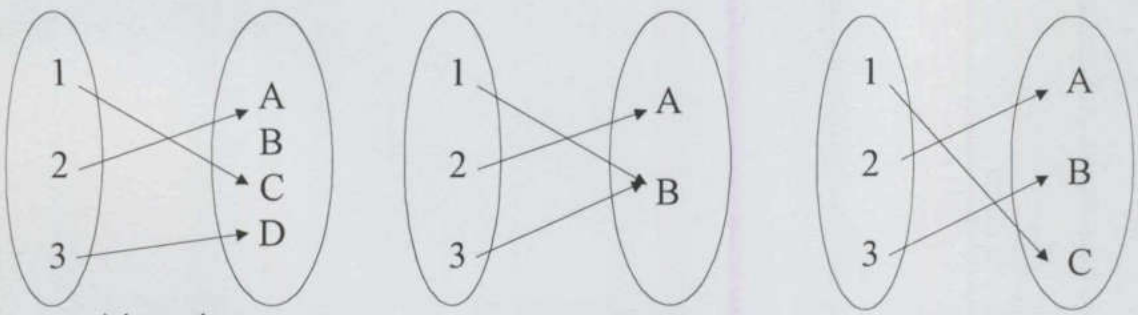
La figura 1.6 muestra gráficamente cuando dos conjuntos no forman una función. En a) un elemento del dominio es relacionado más de una vez con elementos del contradominio y en b) hay elementos del dominio que no están relacionados con algún elemento del contradominio.

El interés se centra en aquellas funciones que cumplen con restricciones en su contradominio, tipificando las funciones en inyectivas, suprayectivas y biyectivas.

La función f: $S \rightarrow T$ se dice *inyectiva* si $\forall a, b \in S, f(a) = f(b) \Rightarrow a = b$, es decir, cada elemento de T aparece a lo más una vez como contradominio de algún elemento de S.

La función f: $S \rightarrow T$ se dice *suprayectiva* si se cumple: $\forall t \in T \exists s \in S \ni, f(s) = t$.

La función f: $S \rightarrow T$ se dice *biyectiva* si es inyectiva y suprayectiva, como muestra la figura 1.7 [17]



a) inyectiva (pero no suprayectiva)

b) Suprayectiva (pero no inyectiva)

c) biyectiva

***Figura 1.7** Funciones inyectivas, suprayectivas y biyectivas*

1.4.Complejidad Algorítmica

El análisis de algoritmos se puede decir que consiste en predecir los recursos que requerirá un algoritmo, para obtener los resultados que se buscan, esto es, el tiempo computacional que consumirá el algoritmo. Analizar un algoritmo puede significar un gran reto ya que requiere habilidades matemáticas tales como combinatoria discreta, elementos de probabilidad, destreza algebraica y la habilidad para identificar los términos más significativos en una expresión matemática. Y si es importante llevar el análisis de matemático de un algoritmo a un análisis computacional, entonces es necesario que todas las pruebas se realicen en un mismo equipo de cómputo. [5]

El análisis de algoritmos es importante en programas de computadora porque frecuentemente existen varios algoritmos disponibles para una aplicación particular y se quisiera saber cuál es el mejor. El análisis usualmente consiste en encontrar valores mínimos (para personas optimistas), valores máximos (para personas pesimistas), valores promedios (para personas probabilísticas) y la desviación estándar (un indicador de qué tan cerca del promedio se espera que sea el valor).

Conociendo el número de veces que se realiza cada paso de un algoritmo se puede determinar el tiempo de ejecución en una computadora particular. Algunos de estos pasos dependen de la cantidad o el tipo de datos de entrada del usuario. [13]

El tiempo de ejecución puede ser expresado como una función *lineal o polinomial* $an + b$, una función *cuadrática o exponencial* $an^2 + bn + c$, una función *logarítmica* $\log n$, una función *factorial* $n!$ para constantes a y b que dependen de los costos de su expresión. Las funciones exponenciales crecen más rápido que las polinomiales. [1]

Paul Backmann introdujo la notación O en su libro *analytische Zahlentheorie (1894)* para comparar el desempeño de los algoritmos de un problema determinado. La notación $O(f(n))$ para la función $f(n)$, $n \in \mathbb{Z}^+$, representa una cantidad que no se conoce explícitamente pero que es finita.

La notación O es una gran ayuda en trabajos de aproximación, ya que describe ampliamente un concepto que ocurre frecuentemente y elimina detalles de información usualmente irrelevantes. Por lo tanto, puede ser manipulado algebraicamente con algunas sencillas consideraciones. [13]

La notación O es conocida como *notación asintótica* y hace referencia a la velocidad de crecimiento de los valores de una función. Para especificar un límite inferior de una velocidad de crecimiento se utiliza la notación Ω y para denotar un límite inferior de una velocidad de crecimiento que no es asintóticamente ajustado se usa la notación o ("o minúscula"). [5]

CAPÍTULO 2
TEORÍA DE GRUPOS

2.1. Antecedentes históricos

Las tres principales áreas de las matemáticas que dieron origen a la teoría de grupos son:

1. Geometría, a principios del siglo XIX
2. Teoría de números, al término del siglo XVIII
3. Teoría de ecuaciones algebraicas, a finales del siglo XVIII, dirigida al estudio de permutaciones.

En 1761 Euler estudió la aritmética modular y aunque el trabajo de Euler no se estableció en términos de teoría de grupos, dio un ejemplo de descomposición de un grupo Abeliano en cosets de un subgrupo. También probó un caso especial del orden de un subgrupo que es un divisor del orden del grupo.

Sin duda que la noción de grupo, en especial de grupo de substituciones que constituye el tema central de Galois, estaba ya esbozada en los trabajos de Lagrange y de Vandermonde del siglo XVIII, y en los de Gauss, Abel, Ruffini y Cauchy del XIX, implícita en problemas de teoría de las ecuaciones, teoría de números y de transformaciones geométricas, pero es Galois quien muestra una idea clara de la teoría general con las nociones de subgrupo y de isomorfismo.

Las permutaciones fueron estudiadas primeramente por Lagrange en su artículo de 1770 sobre teoría de ecuaciones algebraicas. Su principal objetivo era encontrar porqué las ecuaciones cúbicas y cuárticas podrían ser resueltas algebraicamente.

Se dedica Lagrange a lo que llama *calcul des combinaisons* que es el estudio de las permutaciones y aparece el más importante teorema de la teoría de grupos finitos, hoy conocido como teorema de Lagrange. Lagrange observó, en términos modernos, que el orden de un subgrupo del grupo simétrico S_n divide a $n!$. El resultado general lo probó Galois en 1830.

La primera persona que mostró que las ecuaciones de grado quinto no podían ser resueltas algebraicamente fue Ruffini. En 1799 publicó un trabajo cuyo propósito era demostrar la insolubilidad de la ecuación general quinta. El trabajo de Ruffini se basó en el trabajo de Lagrange, pero introdujo grupos de permutaciones. Ruffini dividió sus permutaciones en dos tipos: las permutaciones simples (grupos cíclicos en notación moderna) y las permutaciones compuestas, es decir los grupos no cíclicos. Las permutaciones compuestas las dividió en tres tipos: los grupos intransitivos, grupos imprimitivos transitivos y grupos primitivos transitivos.

En 1844 Cauchy publicó un trabajo original donde estableció la teoría de permutaciones. Introdujo la notación de potencias, positivas y negativas, de permutaciones, con la potencia 0 como identidad; definió el orden de una permutación e introdujo la notación de ciclos. Cauchy llamó a dos permutaciones *similares* si tienen la misma estructura de ciclo y demostró que es la misma cuando la permutación es conjugada.

En 1801 Gauss tomó el trabajo de Euler sobre aritmética modular e hizo una importante aportación a la teoría de grupos abelianos. Examinó ordenes de elementos y probó que hay un subgrupo para cada número que divide el orden de un grupo cíclico. Gauss examinó el comportamiento de formas bajo transformaciones y sustituciones. Particionó las formas en clases y definió una composición sobre las clases. Probó que el orden de composición de tres formas es inmaterial de modo que, en lenguaje moderno, la ley asociativa prevalece. De hecho, Gauss tiene un grupo finito abeliano y más tarde en 1869 Schering, quien editó el trabajo de Gauss, encontró una base para este grupo abeliano.

En 1824, Abel dio la primera demostración aceptada de la insolubilidad de la quíntica y usando las ideas existentes sobre permutaciones de raíces pero con pocas contribuciones al desarrollo de la teoría de grupos. El adjetivo abeliano es en honor a él, quien publicó en 1824 un artículo en donde demuestra la imposibilidad de resolver la ecuación de quinto grado por medio de radicales.

Möbius en 1827, a pesar de que estaba completamente alejado del concepto de grupos, comenzó a clasificar las geometrías utilizando el hecho de que una geometría particular estudia propiedades invariantes bajo un grupo particular. Steiner en 1832 estudió nociones de geometría sintética la cual eventualmente llegó a ser parte del estudio de grupos de transformaciones.

En 1831 Galois fue el primero en comprender realmente que la solución algebraica de una ecuación estaba relacionada a la estructura de un grupo de permutaciones relacionado con la ecuación. En 1832 descubrió que subgrupos especiales, ahora llamados subgrupos normales, son fundamentales. Llamó a la descomposición de un grupo en cosets de un subgrupo descomposición propia si las descomposiciones en coset izquierdo y derecho coinciden. Galois demostró que el grupo simple de orden más pequeño tiene orden 60.

Alrededor de 1850 Cayley publicó un artículo relacionando sus ideas de permutaciones con el trabajo de Cauchy. En 1854 Cayley definió un grupo abstracto y dio una tabla para mostrar la multiplicación de grupos. Dio las tablas de Cayley de un grupo de permutación especial, pero más significativamente para la introducción del concepto de grupo abstracto, se dio cuenta de que las matrices y quaterniones eran grupos. Cayley probó, entre muchos otros resultados, que cada grupo finito puede ser representado como un grupo de permutaciones.

Jordan en sus artículos de 1865, 1869 y 1870 muestra que se dio cuenta del significado de grupos de permutaciones. Definió isomorfismo de grupos de permutaciones y probó el teorema de Jordan-Hölder para grupos de permutaciones. Hölder lo probó en el contexto de grupos abstractos en 1889.

Combinando el desarrollo alcanzado por las geometrías no euclidianas y la geometría proyectiva con la teoría de los invariantes y la teoría de grupos, Klein expone en su Programa de Erlangen, en 1872, mediante grupo y subgrupos, una sistematización y jerarquización de todas las geometrías, concibiendo como objeto de cada geometría el descubrimiento de propiedades invariantes respecto de un determinado grupo de transformaciones y considerando cada geometría como subgeometría de otra a la que se adjunta cierta figura básica, que debe quedar invariante. Más tarde, en 1884, ofreció un ejemplo de dos grupos isomorfos: el de las rotaciones del icosaedro regular y el de la ecuación de quinto grado.

Mientras Klein estudia grupos discretos, Sophus Lie aborda, a partir de 1872, el estudio de los grupos continuos de transformaciones y su clasificación y aplicación a la integración de las ecuaciones diferenciales con derivadas parciales. Sus trabajos y los de sus discípulos aparecieron hacia fines de siglo. Klein intentó codificar todas las geometrías existentes de acuerdo con el grupo de transformaciones en las cuales eran invariantes las propiedades de la geometría. Esta idea se conoce como el programa de Erlangen.

Von Dyck, construyó en 1882 grupos libres y la definición de grupos abstractos en términos de generadores y relaciones haciendo uso de los grupos de permutaciones y las condiciones de asociatividad, identidad e inverso sobre la composición de funciones.

La teoría de grupos culmina hacia 1880 al aparecer la teoría de los grupos abstractos, que confiere a la teoría iniciada por Galois los caracteres de estructura algebraica. Como tal estructura, y en virtud del isomorfismo, un grupo puede entonces estudiarse ahora bajo el aspecto particular de uno de sus modelos o interpretaciones, ahora bajo su forma general puramente abstracta como resultado de un proceso de rasgos propios, específicos de la matemática de hoy, que evidencia que la misma abstracción, signo de la matemática de todas las épocas, ha cambiado o evolucionado a través de los tiempos. [6]

2.2. Grupos

Un *Grupo* $<G,*>$ es un conjunto G, y una operación binaria * definida sobre G, tal que satisface los siguientes axiomas:

- La operación binaria * es asociativa.
- Existe un elemento identidad único ε en G tal que $\varepsilon * x = x * \varepsilon = x$ para todas las $x \in G$.
- Para cada a en G existe un solo elemento a' en G con la propiedad de que $a' * a = a * a' = \varepsilon$. (éste elemento a' se denomina elemento inverso de a bajo *).

Por ejemplo, el conjunto $<\mathbb{Z}^+, +>$ *no* es un grupo pues no existe un elemento identidad en $\mathbb{Z}^+$ bajo +. Sin embargo el conjunto $<\mathbb{Q}^+, \bullet>$ *si es* un grupo.

Un *grupo* $<G,*>$ se dice que es *finito* si G es un conjunto finito. El conjunto más pequeño que puede dar lugar a un grupo es el conjunto $\{\varepsilon\}$. La única operación binaria * posible en $\{\varepsilon\}$ está definida por $\varepsilon * \varepsilon = \varepsilon$. En cada grupo, el elemento identidad es siempre su propio inverso (pero no viceversa).

Por ejemplo, $R = \left\{0, \frac{\pi}{3}, \frac{2\pi}{3}, \pi, \frac{4\pi}{3}, \frac{5\pi}{3}\right\}$ es el conjunto de seis posibles formas de rotar figuras geométricas dibujadas en un plano. Digamos que $\oplus$ es una operación binaria tal que para a y b en R, $a \oplus b$ es la rotación angular correspondiente a la rotación sucesiva por a y luego por b. $(R, \oplus)$ es un grupo con 0 como identidad, el inverso de $\frac{\pi}{3}$ es $\frac{5\pi}{3}$, el inverso de π es él mismo y así sucesivamente. [15]

El *orden* de un grupo finito $<G, * >$, $|G|$, es el número de elementos en G. Por ejemplo, el orden de $<\mathbb{Z}_5,+>$ es 5 y el de $<R, \oplus >$ es 6.

Un *grupo* $<G,*>$ es *infinito* si G es un conjunto infinito. Por ejemplo, consideremos el grupo $<\mathbb{Z}, +>$, donde + es la operación suma, ε es el número 0 y el inverso de n es $-n$.

Un *grupo* $<G, *>$ es llamado *grupo Abeliano*[1] o *grupo conmutativo* si su operación binaria * es conmutativa. El grupo $<\mathbb{Z}, +>$, donde + es la operación suma, *es* un grupo Abeliano. Sin embargo, el grupo <M, *>; donde M es el conjunto de todas las n x n matrices invertibles o que Det[M] $\neq 0$ con entradas en $\mathbb{R}$; no es un grupo Abeliano.

Un grupo es *cíclico* si existe un elemento $x \in G$ tal que para todo $a \in G$, $a = x^n$ para algún $n \in \mathbb{Z}$. El grupo $<\mathbb{Z}_4,+>$ es cíclico. En este caso, $x = 1$ y $x = 3$ generan el grupo. [9]

Para el caso del 3	Para el caso del 1
$3^0 = \varepsilon = 0$	$1^0 = \varepsilon = 0$
$3^1 = 3$	$1^1 = 1$
$3^2 = 2$	$1^2 = 2$
$3^3 = 1$	$1^3 = 3$

En general n es generador de $<\mathbb{Z}_m,+>$ si y solo si $(n, m) = 1$

Tablas de Cayley[2]

La estructura de un grupo finito puede ser determinada por una simple regla o por una tabla de grupo, llamada tabla de Cayley, donde se listan todos los resultados de la operación binaria sobre el conjunto.

Para la construcción de una tabla de Cayley de un grupo se pueden enumerar algunas condiciones que deben satisfacerse para definir una operación binaria sobre un conjunto finito.

- Se colocan los elementos del grupo en la parte superior hacia la derecha de la tabla, en el mismo orden en que se colocan del lado izquierdo hacia abajo, colocando en primer lugar la identidad.
- Es necesario que algún elemento del conjunto actúe como la identidad ε.

[1] En honor al Matemático Noruego Niels Henry Abel (1802-1829). Ver su biografía en Anexo II
[2] En honor al Matemático Inglés Arthur Cayley (1821-1895). Ver su biografía en Anexo II

- La condición ε * x = x significa que el renglón de la tabla que contiene a ε en el extremo izquierdo, debe contener exactamente los elementos que aparecen hasta arriba de la tabla en el mismo orden.
- De forma análoga, la condición x * ε = x significa que la columna de la tabla bajo ε, debe contener precisamente los elementos que aparecen en el extremo izquierdo en el mismo orden.
- El hecho de que cada elemento a tenga un inverso derecho y un inverso izquierdo, quiere decir que en el renglón frente a a debe aparecer un elemento ε y que en la columna bajo a debe aparecer ε en algun lugar. Así ε debe aparecer en cada renglón y en cada columna.
- Cada elemento del grupo debe aparecer una y sólo una vez en cada renglón y en cada columna de la tabla.
- Una operación binaria definida mediante una tabla es conmutativa si y solo si la tabla es simétrica con respecto a su diagonal principal.

Las siguientes cuatro tablas describen, de derecha a izquierda, el proceso de llenado de la tabla de Cayley para un grupo (necesariamente Abeliano) de orden 3 siguiendo los criterios anteriores.

*	ε	a	b
ε			
a			
b			

*	ε	a	b
ε	ε	a	b
a	a		
b	b		

*	ε	a	b
ε	ε	a	b
a	a	b	
b	b	ε	

*	ε	a	b
ε	ε	a	b
a	a	b	ε
b	b	ε	a

No todas las tablas pueden ser generadas de manera única puesto que existen varios grupos de ciertos ordenes. Cualesquiera dos grupos de orden tres son estructuralmente el mismo, es decir, estos dos grupos son isomorfos. Esto no sucede con los grupos de orden 4, consideremos el conjunto $\{\varepsilon, a, b, c\}$ con la identidad ε para la operación del grupo, entonces las tablas de grupo que se pueden generar son las que se muestran a continuación:

*	ε	a	b	c
ε	ε	a	b	c
a	a	ε	c	b
b	b	c	ε	a
c	c	b	a	ε

1

*	ε	a	b	c
ε	ε	a	b	c
a	a	b	c	ε
b	b	c	ε	a
c	c	ε	a	b

2

*	ε	a	b	c
ε	ε	a	b	c
a	a	c	ε	b
b	b	ε	c	a
c	c	b	a	ε

3

De estas tres tablas, la tabla 1 y 3 son estructuralmente la misma, para que ambas sean la misma se necesita cambiar el nombre de los elementos ε y *c* en las columnas y filas centrales. De lo anterior se concluye que hay dos tipos diferentes de estructuras de grupo de orden 4. La figura 2.1 muestra muestran estas dos tablas de grupo. [8]

V:

	ε	a	b	c
ε	ε	a	b	c
a	a	ε	c	b
b	b	c	ε	a
c	c	b	a	ε

$\mathbb{Z}_4$:

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

Figura 2.1 *Tablas de grupo de V y* $\mathbb{Z}_4$

El grupo *V* es el 4-grupo de Klein (La *V* se genera del alemán *vier-gruppe* o 4-grupo de Klein)

Representación Computacional de un Grupo

Para representar un grupo se utilizan las matrices, mismas que dan lugar a una representación computacional natural en forma de listas de listas. Por ejemplo, el grupo Abeliano $<\mathbb{Z}_7,+>$ se muestra en la figura 2.2

```
n = 7;
g = Table[ Mod[i + j, n], {i, n}, {j, n}]

Show[Graphics[RasterArray[Reverse[Table[ GrayLevel[Mod[i + j, n] / n], {i, n}, {j, n}]]]],
  AspectRatio → Automatic];
```

$$\begin{pmatrix} 2 & 3 & 4 & 5 & 6 & 0 & 1 \\ 3 & 4 & 5 & 6 & 0 & 1 & 2 \\ 4 & 5 & 6 & 0 & 1 & 2 & 3 \\ 5 & 6 & 0 & 1 & 2 & 3 & 4 \\ 6 & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 & 0 \end{pmatrix}$$

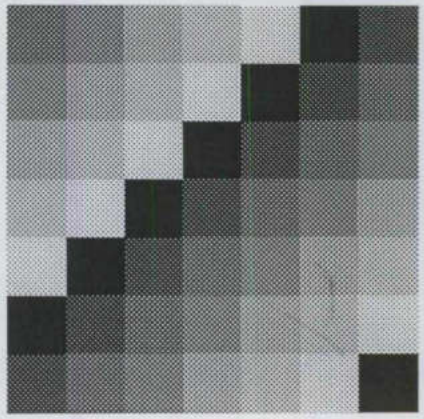


Figura 2.2 *El grupo Abeliano* $<\mathbb{Z}_7, +>$

El grupo abeliano $<\mathbb{Z}_{71}, *>$ y el no-abeliano $<\mathbb{Z}_{70}, *>$, respectivamente, se pueden observar en la figura 2.3

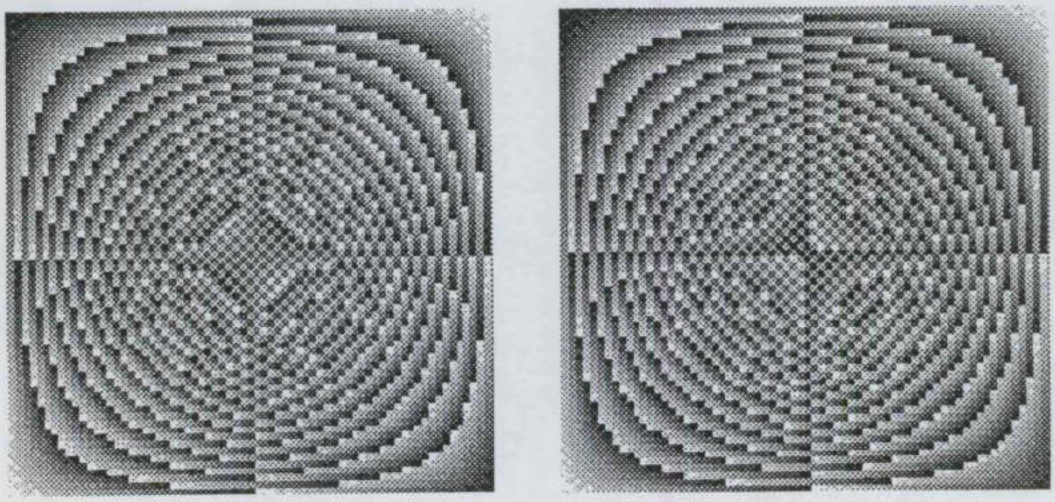


Figura 2.3 *El grupo Abeliano* $<\mathbb{Z}_{71}, *>$ *y el No- Abeliano* $<\mathbb{Z}_{70}, *>$

2.3. Subgrupos

Sean <G, *> y <H, *> grupos. Si H $\subseteq$ G entonces se dice que H es un subgrupo de G. Se denota por $H \leq G$ o $G \geq H$ el hecho de que H es un subgrupo de G, y $H < G$ o $G > H$ significa que $H \leq G$, pero $H \neq G$. La identidad ε de G está en H y para todos los elementos a de H, existe a^{-1} en H

Por ejemplo, <$\mathbb{Z}$, +> es un subgrupo del grupo <$\mathbb{R}$, +>

En $\mathbb{Z}_4$, {0, 2}es el único subgrupo no trivial; {0, 3} no es un subgrupo de $\mathbb{Z}_4$ pues no es cerrado. El grupo V= *{ε, a, b, c}*, de la anterior figura 2.1, tiene tres subgrupos propios no triviales *{ε, a}*, *{ε, b}* y *{ε, c}* en este caso *{ε, a, b}* no es cerrado bajo la operación de V.

Para representar gráficamente los subgrupos de un grupo, se utiliza un *diagrama reticular*. El grupo más abarcante se ubica en la parte posterior del diagrama y hacia abajo se localizan los subgrupos.

En la figura 2.4 se muestran los diagramas reticulares de las estructuras de grupo de orden 4: $\mathbb{Z}_4$ y V.

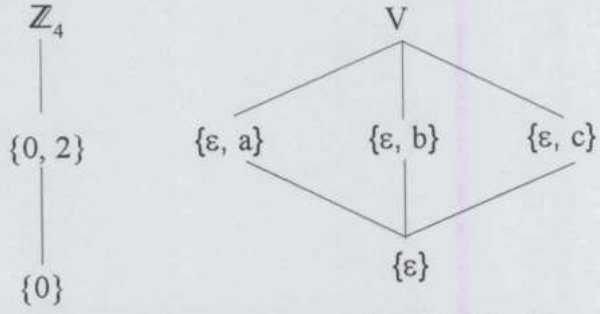


Figura 2.4 *Diagrama reticular de $\mathbb{Z}_4$ y V*

La forma de identificar los subgrupos en una tabla de grupo es verificando que hay filas con sus correspondientes columnas que forman también un grupo. Por ejemplo, $\mathbb{Z}_6$ tiene cuatro subgrupos {0}. {0, 3}, {0, 2, 4} y $\mathbb{Z}_6$, su tabla de grupo es la siguiente:

$\mathbb{Z}_6$:

+	**0**	**2**	**4**	**3**	**1**	**5**
0	0	2	4	3	1	5
2	2	4	0	5	3	1
4	4	0	2	1	5	3
3	3	5	1	0	4	2
1	1	3	5	4	2	0
5	5	1	3	2	0	4

Se puede observar que los elementos de la esquina superior izquierda, en color gris, forman una tabla de Cayley de grupo por ellos mismos. Lo mismo sucede si se colocan juntas las filas y columnas de los elementos $\{0, 3\}$.

Cosets (Grupos de clases laterales).

Supóngase que G es un grupo y $H \leq G$. Para cada $g \in G$ el coset izquierdo gH se define como el conjunto $\{gx \mid x \in H\}$ y el coset derecho Hg se define como el *conjunto* $\{xg \mid x \in H\}$. gH *y* Hg son los conjuntos de elementos de G obtenidos mediante la combinación de los elementos de G con elementos de H. Los cosets también son llamados grupos de clases laterales. Cada clase lateral izquierda o coset izquierdo deben ser una clase lateral derecha o coset derecho.

Si la operación en G es la suma, escribimos $a+h$ en vez de aH, donde $a+H = \{a+h \mid h \in H\}$ Se puede observar esta propiedad en el siguiente ejemplo

Para $<\mathbb{Z}_{12},+>$ y $H= \{0,4,8\}$, tenemos que

$$0 + H = \{0, 4, 8\} = 4 + H = 8 + H = H$$
$$1 + H = \{1, 5, 9\} = 5 + H = 9 + H$$
$$2 + H = \{2, 6, 10\} = 6 + H = 10 + H$$
$$3 + H = \{3, 7, 11\} = 7 + H = 11 + H$$

Se observa que la propiedad de los cosets es particionar el grupo en subconjuntos. Cada subconjunto tiene el mismo orden y cada elemento de G aparece en al menos un coset. Por

lo tanto G es igual a la unión de todos los cosets obtenidos. En el ejemplo, $\mathbb{Z}_{12}$ se particiona en cuatro subconjuntos, como se muestra en la figura 2.5

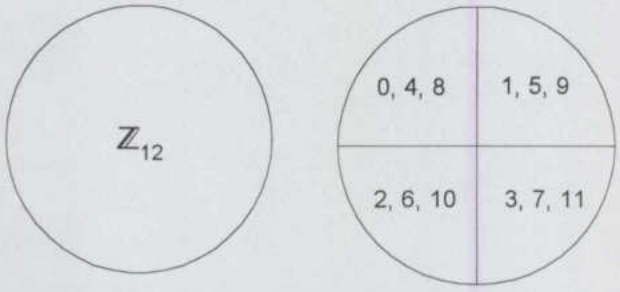


***Figura 2.5** Subgrupos de $\mathbb{Z}_{12}$*

Sea $H_1 = \{0, 4, 8\}$, $H_2 = \{1, 5, 9\}$, $H_3 = \{2, 6, 10\}$ y $H_4 = \{3, 7, 11\}$. Obsérvese que

$$G = H_1 \cup H_2 \cup H_3 \cup H_4 \text{ y } H_1 \cap H_2 \cap H_3 \cap H_4 = \varnothing$$

Si se reordenara la tabla de grupo agrupando cada subconjunto, se obtendría una tabla parecida a la siguiente.

*	$\mathbf{H_1}$	$\mathbf{H_2}$	$\mathbf{H_3}$	$\mathbf{H_4}$
$\mathbf{H_1}$	H_1	H_2	H_3	H_4
$\mathbf{H_2}$	H_2	H_3	H_4	H_1
$\mathbf{H_3}$	H_3	H_4	H_1	H_2
$\mathbf{H_4}$	H_4	H_1	H_2	H_3

De lo anterior se puede observar que cualquier elemento de H_i multiplicado por cualquier elemento de H_j produce siempre un elemento de H_k, donde H_i, H_j y H_k son subconjuntos de G.

Se utiliza el siguiente código para poder generar los cosets izquierdos y derechos de un grupo, a partir de un subgrupo dado.

```
coset[G_List, H_List] := Module[{g, i}, g = {};
  co = Map[Mod[Plus[#, H], 12] &, G];
  Union[Map[Sort[#] &, co]]]
```

```
coset[Z12, {0, 4, 8}]
{{0, 4, 8}, {1, 5, 9}, {2, 6, 10}, {3, 7, 11}}


coset[Z12, {6, 7, 4, 5}]
{{0, 1, 2, 3}, {0, 1, 2, 11}, {0, 1, 10, 11}, {0, 9, 10, 11},
{1, 2, 3, 4}, {2, 3, 4, 5}, {3, 4, 5, 6}, {4, 5, 6, 7}, {5, 6, 7, 8},
{6, 7, 8, 9}, {7, 8, 9, 10}, {8, 9, 10, 11}}


coset[Z12, {6, 7, 4, 5, 8}]
{{0, 1, 2, 3, 4}, {0, 1, 2, 3, 11}, {0, 1, 2, 10, 11}, {0, 1, 9, 10, 11},
{0, 8, 9, 10, 11}, {1, 2, 3, 4, 5}, {2, 3, 4, 5, 6}, {3, 4, 5, 6, 7},
{4, 5, 6, 7, 8}, {5, 6, 7, 8, 9}, {6, 7, 8, 9, 10}, {7, 8, 9, 10, 11}}
```

Como se ve en el último caso del ejemplo anterior, cuando el grado de H no es un divisor del grado de G se generan particiones que no son disjuntas, por lo tanto el grupo original no puede ser particionado en subconjuntos. Este resultado se encuentra fundamentado en el trabajo que desarrolló Lagrange.

Teorema de Lagrange

1. Si H es un subgrupo del grupo finito G, entonces G puede ser escrito como la unión disjunta de los grupos de clases laterales. Esto es

   $G = g_1 H \cup g_2 H \cup ... \cup g_r H$

Para algún $g_1, g_2, ...g_r \in G$

2. Si H es un subgrupo del grupo finito G, entonces $|H|$ divide $|G|$

Demostración

G es un grupo finito de orden n y H es un subgrupo de orden m. Si $H = G$ está demostrada la parte 2. En caso contrario, $m < n$ y existe un elemento $a \in G - H$. Como $a \notin H$, se sigue que $aH \neq H$, de modo que $aH \cap H = \emptyset$. Si $G = aH \cup H$, entonces $|G| = |aH| + |H| = 2|H|$ y se demuestra el teorema. En caso contrario, existe un elemento $b \in G\text{-}(H \cup aH)$, tal que $bH \cap H = \emptyset = bH \cap aH$ y $|bH| = |H|$. Si $G = bH \cup aH \cup H$, tenemos que $|G| = 3|H|$. En caso contrario, se obtiene un elemento $c \in G$ tal que $c \notin bH \cup aH \cup H$. El grupo G es finito, de

modo que la demostración termina y se observa que $G = a_1H \cup a_2H \cup ... \cup a_kH$. Por lo tanto $|G| = k|H|$ y m divide a n.■ [9]

También se puede demostrar el teorema del siguiente modo. Supóngase que H tiene m elementos. Considérese la colección de todos los cosets de H. Estos cosets son disjuntos, su grado es m y todo elemento de G pertenece a algún coset. Entonces, si hay r cosets, se tienen $n = rm$ de modo que m divide a n. ■ [8]

Teorema de Lagrange.
El orden de un subgrupo divide al orden de su grupo.

Este teorema nos dice que un grupo G solo puede tener subgrupos de orden n si n es un divisor de $|G|$; si n es un divisor de |G| entonces G no necesariamente tiene un subgrupo de orden n. El más pequeño ejemplo que muestra esta última afirmación es el grupo de orden 12. Este grupo no tiene un subgrupo de orden 6 a pesar de que 6 es divisor de 12. [4]. Considérese H que consiste de 12 permutaciones de $\{1, 2, 3, 4\}$. H = {1234, 2143, 3412, 4321, 2314, 3124, 2431, 4132, 3241, 1342, 1423, 4213}. H forma un subgrupo de S_4, S_4 es descrito en la siguiente sección. Este grupo de orden 12 no tiene un subgrupo de orden 6. [20]

2. 4. Grupos de permutaciones

Sea A un conjunto arbitrario y sean σ y τ permutaciones de A de modo que σ y τ son funciones biyectivas sobre A. Podemos suponer, sin pérdida de generalidad, que A = {1, 2, 3, ..., n} para alguna n positiva.

Supongamos que $n = 5$ y σ es la permutación {4,2,5,3,1} tomada del conjunto de las 120 permutaciones de A = {1, 2, 3, 4, 5}.

Una forma de representar a σ es:

$1 \rightarrow 4, 2 \rightarrow 2, 3 \rightarrow 5, 4 \rightarrow 3$ y $5 \rightarrow 1$.

Se escribe σ en una notación más común como sigue:

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 2 & 5 & 3 & 1 \end{pmatrix}$$

Ahora sea

$$\tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 4 & 2 & 1 \end{pmatrix}$$

haciendo uso de composición de funciones, se tiene

$$\tau\sigma 1 = \tau(\sigma 1) = \tau 4 = 2$$
$$\tau\sigma 2 = \tau(\sigma 2) = \tau 2 = 5$$
$$\tau\sigma 3 = \tau(\sigma 3) = \tau 5 = 1$$
$$\tau\sigma 4 = \tau(\sigma 4) = \tau 3 = 4$$
$$\tau\sigma 5 = \tau(\sigma 5) = \tau 1 = 3$$

lo que se representa como

$$\sigma\tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 2 & 5 & 3 & 1 \end{pmatrix}\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 4 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 1 & 4 & 3 \end{pmatrix}$$

Si se observan los elementos que forman las filas y las columnas de las tablas de grupo, se puede ver claramente que cada una de ellas es una permutación de los elementos del grupo. Por ello puede decirse que al menos todo grupo finito G es isomorfo a algún subgrupo del grupo S_G de todas las permutaciones de G. Basado en ello Arthur Cayley propuso que todo grupo, finito o infinito, es isomorfo a algún grupo formado por permutaciones bajo la multiplicación de permutaciones.

La forma de demostrar lo establecido por Cayley involucra tres pasos

1. Encontrar un conjunto G' de permutaciones que sea candidato a formar un grupo, bajo la multiplicación de permutaciones, isomorfo a G.

2. Probar que G' es un grupo bajo la composición de funciones, multiplicación de permutaciones.
3. Definir una función ϕ: $G \rightarrow G'$ y mostrar que ϕ es un isomorfismo entre G y G'.

Veamos estos pasos en el único grupo de orden 3.

Sea $G = \{1, 2, 3\}$

1. El primer paso consiste en buscar un conjunto G' de permutaciones que sea candidato a formar un subgrupo isomorfo a G. Pensemos en G simplemente como un conjunto y sea S_G el grupo de todas las permutaciones de G. Sin embargo, este conjunto es demasiado grande para ser isomorfo a G. Para solucionar esto, definase cierto subconjunto S_G. Para $a \in G$, sea ρ_a una permutación de G, esto es $\rho_a \in S_G$. Sea $G' = \{\rho_a \mid a \in G\}$.
   Cada ρ_a corresponde con la fila a en la tabla de grupo de G. Por lo tanto

$$\rho_1 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} \qquad \rho_2 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix} \qquad \rho_3 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}$$

2. Demostramos que G' es un grupo. Realizando la composición de funciones se tiene la siguiente tabla de grupo. Donde claramente se observa que es grupo.

*	ρ_1	ρ_2	ρ_3
ρ_1	ρ_1	ρ_2	ρ_3
ρ_2	ρ_2	ρ_3	ρ_1
ρ_3	ρ_3	ρ_1	ρ_2

3. Finalmente, para probar que G es isomorfo al grupo G' descrito, se define una función ϕ: $G \rightarrow G'$ por $\phi(a) = \rho_a$ para a $\in G$. Por lo que

   $\phi(1) = \rho_1$

   $\phi(2) = \rho_2$

   $\phi(3) = \rho_3$

entonces queda demostrado que G' es isomorfo a G. [8]

Dentro de los grupos de permutaciones hay dos ejemplos no triviales en el estudio de la teoría de grupos, ellos son el grupo simétrico S_n y el grupo dihédrico D_n, los cuales se explican en los párrafos que siguen.

El *grupo simétrico* S_n de $n!$ elementos es el conjunto de todas las permutaciones de *Range[n]* y es un grupo bajo la operación de composición de funciones, como se muestra más adelante con S_3.

Se pueden generar los grupos simétricos S_n mediante la siguiente línea de código.

```
S_n_ := Permutations[Range[n]]
```

Para $n = 4$, es decir, todas las permutaciones de cuatro elementos:

```
n = 4;
S_n

{{1, 2, 3, 4}, {1, 2, 4, 3}, {1, 3, 2, 4}, {1, 3, 4, 2},
{1, 4, 2, 3}, {1, 4, 3, 2}, {2, 1, 3, 4}, {2, 1, 4, 3},
{2, 3, 1, 4}, {2, 3, 4, 1}, {2, 4, 1, 3}, {2, 4, 3, 1},
{3, 1, 2, 4}, {3, 1, 4, 2}, {3, 2, 1, 4}, {3, 2, 4, 1},
{3, 4, 1, 2}, {3, 4, 2, 1}, {4, 1, 2, 3}, {4, 1, 3, 2},
{4, 2, 1, 3}, {4, 2, 3, 1}, {4, 3, 1, 2}, {4, 3, 2, 1}}
```

y para $n = 3$

```
n = 3;
S_n

{{1, 2, 3}, {1, 3, 2}, {2, 1, 3}, {2, 3, 1}, {3, 1, 2}, {3, 2, 1}}
```

Si se utilizara la siguiente línea de código se podría obtener el elemento que se encuentra en la posición $[i, j]$, este elemento es la composición de funciones de la permutación i con la permutación j.

```
op_G_,i_,j_ := Position[G, G[[i]][[G[[j]]]]][[1,1]]
```

El número que se les ha asignado a i y j corresponde al número de permutación en orden lexicográfico.

por ejemplo, para S_3, la composición de la 4ª y 5ª permutación es:

```
opSn,4,5
1
```

Esto es,

$$\sigma\tau = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}\begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$$

Con la ayuda de la función op_{S_n} se genera la tabla de composición de los grupos simétricos.

La tabla de composición de funciones para $n = 3$ es la que se muestra junto con su gráfica en escala de grises en la figura 2.6. Cada uno de los elementos de la tabla representan cada una de las permutaciones de n, el número que se les ha asignado corresponde a su posición en orden lexicográfico, como ya se había mencionado anteriormente.

```
Table[opSn,i,j, {i, n!}, {j, n!}] // MatrixForm

n = 3;

oSn = n!;

Show[Graphics[RasterArray[Reverse[Table[GrayLevel[opSn,i,j/oSn],
{i, oSn}, {j, oSn}]] ]],AspectRatio -> Automatic];
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 1 & 5 & 6 & 3 & 4 \\ 3 & 4 & 1 & 2 & 6 & 5 \\ 4 & 3 & 6 & 5 & 1 & 2 \\ 5 & 6 & 2 & 1 & 4 & 3 \\ 6 & 5 & 4 & 3 & 2 & 1 \end{pmatrix}$$

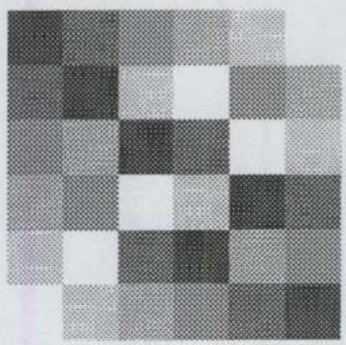


Figura 2.6 *Composición de funciones de* S_3

Todos los elementos de S_n pueden ser descritos en *notación cíclica*. Un ciclo es una lista de números encerrada en paréntesis, separada por espacios en blanco, esto es, $(a_1\ a_2\ \ldots\ a_m)$ lo cual indica que "a_1 va a a_2, a_2 va a a_3,... a_{m-1} va a_m y a_m va a a_1". No todas las permutaciones son cíclicas, pero todas son combinaciones de ciclos. El siguiente segmento de código muestra esta propiedad.

```
<< DiscreteMath`Combinatorica`
cicSn_ := ToCycles /@ Permutations[Range[n]]

Clear[t]
t[{x___, {y_Integer, z___}, u___}] := t[{x, FromDigits[{y, z}], u}]
t[x_] := StringJoin["(" <> ToString[#] <> ")" & /@ x]


t /@ cicSn
```

Un ejemplo es para n= 4

```
t /@ cicS4

{(1)(2)(3)(4), (1)(2)(43), (1)(32)(4), (1)(342), (1)(432),
(1)(42)(3), (21)(3)(4), (21)(43), (231)(4), (2341), (2431),
(241)(3), (321)(4), (3421), (31)(2)(4), (341)(2), (31)(42),
(3241), (4321), (421)(3), (431)(2), (41)(2)(3), (4231), (41)(32)}
```

y para $n = 3$

```
t /@ cicS3

{ (1)(2)(3), (1)(32), (21)(3), (231), (321), (31)(2)}
```

El grupo S_n, donde $n \geq 2$, puede ser dividido equitativamente en permutaciones pares e impares, esto es, hay $\frac{n!}{2}$ permutaciones pares y el mismo número de permutaciones impares. Una permutación es par o impar si puede expresarse como el producto de un número par o impar de transposiciones; una transposición es un ciclo de longitud 2, respectivamente.

El conjunto de permutaciones pares de S_n es llamado el *grupo Alternante* A_n de n letras. Los grupos A_n pueden obtenerse con el siguiente código.

```
An_ := FromCycles /@ Select[ToCycles /@ Sn, EvenQ[Length[#]] &]
```

El grupo A_4 es

```
A4
{{1, 2, 3, 4}, {1, 3, 4, 2}, {1, 4, 2, 3}, {2, 1, 4, 3},
{2, 3, 1, 4}, {2, 4, 3, 1}, {3, 1, 2, 4}, {3, 2, 4, 1},
{3, 4, 1, 2}, {4, 1, 3, 2}, {4, 2, 1, 3}, {4, 3, 2, 1}}
```

y su gráfica luciría como lo muestra la figura 2.7

```
oA = n! / 2;
Show[Graphics[RasterArray[Reverse[Table[GrayLevel[opAn,i,j/oA],
{i, oA}, {j, oA}]] ]],AspectRatio -> Automatic];
```

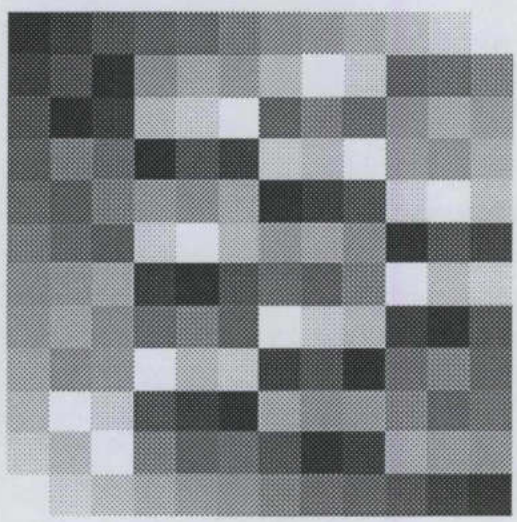


Figura 2.7 *Representación gráfica de* A_4

En geometría, las permutaciones de un conjunto de elementos son llamadas *transformaciones* de un conjunto. Una *rotación* es una transformación que rota el plano en sentido contrario al movimiento de las manecillas del reloj. Las rotaciones se representarán con el símbolo ρ y su obtención es a través del código siguiente:

```
rotn_ = NestList[RotateLeft, Range[n], n-1]
```

Las rotaciones de S_3 son

```
rot3

{{1, 2, 3}, {2, 3, 1}, {3, 1, 2}}
```

y las rotaciones de S_4 son

```
rot4

{{1, 2, 3, 4}, {2, 3, 4, 1}, {3, 4, 1, 2}, {4, 1, 2, 3}}
```

Las rotaciones de n corresponden con el grupo cíclico de n, esto se denota como C_n, como se muestra a continuación.

```
Cn_ := rotn

C3
{{1, 2, 3}, {2, 3, 1}, {3, 1, 2}}

C4

{{1, 2, 3, 4}, {2, 3, 4, 1}, {3, 4, 1, 2}, {4, 1, 2, 3}}
```

y la gráfica del grupo cíclico C_7 se ilustra en la figura 2.8

```
n = 7;
oC = n;
Show[Graphics[RasterArray[Reverse[Table[GrayLevel[opCn,i,j/oC],
{i, oC}, {j, oC}]] ]],AspectRatio -> Automatic];
```

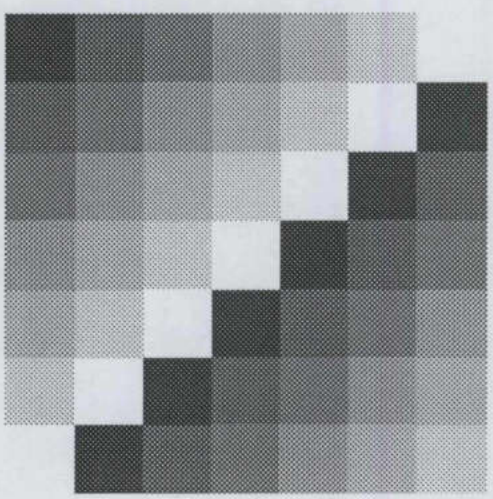


Figura 2.8 *Representación gráfica de* C_7

Una *reflexión* es una transformación que refleja el plano con respecto a un eje que tiene dos vértices opuestos. Si n es impar la línea se traza del centro y cualquiera de los vértices; si n es par la línea se traza del centro y el punto central de alguno de los lados del polígono. Las rotaciones se representarán con el símbolo μ y su obtención es a través del siguiente código

```
Ref_n_ := Join[{1},n+2-Range[2,n]]
```

Las reflexiones de S_3 son

```
ref3
{1, 3, 2}
```

y las reflexiones de S_4 son

```
ref4
{1, 4, 3, 2}
```

El *grupo dihedrico* D_n es el grupo de movimientos rígidos (rotaciones y reflexiones) de un polígono regular con n lados bajo composición. El grupo D_n es de orden 2n. Basado en eso, el grupo D_n se genera con el siguiente código:

```
Dn_:= Join[rotn, #[[refn]] & /@ rotn]
```

El grupo dihedrico D_3 es

```
D3
{{1, 2, 3}, {2, 3, 1}, {3, 1, 2}, {1, 3, 2}, {2, 1, 3}, {3, 2, 1}}
```

y D_4 es

```
D4
{{1, 2, 3, 4}, {2, 3, 4, 1}, {3, 4, 1, 2}, {4, 1, 2, 3},
{1, 4, 3, 2}, {2, 1, 4, 3}, {3, 2, 1, 4}, {4, 3, 2, 1}}
```

La gráfica del grupo dihedrico D_7 se muestra en la figura 2.9

```
n = 7;
oD = 2n;
```

```
Show[Graphics[RasterArray[Reverse[Table[GrayLevel[op_{Dn,i,j}/oD],
{i, oD}, {j, oD}]] ]],AspectRatio -> Automatic];
```

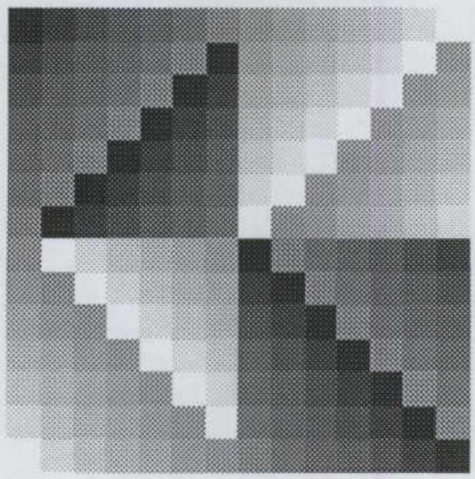


Figura 2.9 *Representación gráfica de* D_7

Hay una correspondencia entre los elementos de S_3 y las formas en que pueden colocarse, una sobre otra, dos copias de un triángulo equilátero con vértices 1, 2 y 3. Por lo tanto, S_3 denota el grupo D_3 de simetrías de un triángulo equilátero. Sin embargo $S_4 \neq D_4$ pues $|S_4| = 4! = 24$ y $|D_4| = 8$. [19]

2.5. Isomorfismo y automorfismo de grupos.

La propiedad de isomorfismo es una relación de equivalencia en una colección de grupos. Esto significa que dada una colección no vacía de grupos, siempre se puede partir la colección en celdas, clases de equivalencia o cosets, tales que cualesquiera dos grupos en la misma celda son isomorfos y no hay grupos en celdas distintas que sean isomorfos. Se ha visto que cualesquiera dos grupos de orden 3 son isomorfos. Se expresa diciendo que sólo hay un grupo de orden 3, salvo isomorfismo.

Hay un solo grupo de orden 1, uno de orden 2 y uno de orden 3, salvo isomorfismo. De orden 4 hay exactamente dos grupos diferentes, salvo isomorfismo: el grupo $\mathbb{Z}_4$ y el 4-

grupo V de Klein. Hay al menos dos grupos diferentes de orden 6, salvo isomorfismo, a saber, $\mathbb{Z}_6$ y S_3.

En el caso de que dos grupos posean una función biyectiva entre ellos, esto es, un isomorfismo; para demostrar que no son isomorfos, se suele exhibir una propiedad estructural que un grupo posee pero que el otro no posee. A esta propiedad se le da el nombre de *invariante de un grupo,* la cual debe compartir con cualquier grupo isomorfo. Esta propiedad no depende de los nombres o de cualquier otra característica no estructural de los elementos.

Invariantes posibles en un grupo	*Propiedades no estructurales posibles en un grupo*
El grupo es cíclico	El grupo contiene al 5
El grupo es abeliano	Todos los elementos del grupo son números
El grupo tiene orden n	Los elementos del grupo son permutaciones
El grupo es finito	La operación del grupo se llama *composición*
El grupo tiene exactamente x elementos de orden n	La operación del grupo se denota por yuxtaposición

Basado en lo anterior no se puede decir que $\mathbb{Z}$ y $\mathbb{Q}$, ambos bajo la suma, no sean isomorfos porque $\frac{1}{2} \in \mathbb{Q}$ y $\frac{1}{2} \notin \mathbb{Z}$. Pero si puede decirse que no son isomorfos porque $\mathbb{Z}$ es cíclico y $\mathbb{Q}$ no lo es.

Cuando hay un isomorfismo de un grupo con él mismo se le da el nombre de *automorfismo de grupo*.

Automorfismo de grupo de un grafo

Si $\{u, v\}$ es una arista del grafo $G = (V,E)$ y α es una permutación de V, entonces se define $\alpha(\{u, v\}) = \{\alpha(u), \alpha(v)\}$.Una permutación α de V es un automorfismo del grafo $G = (V,E)$ si $\alpha(\{u, v\}) \in E$, cuando $\{u, v\} \in E$.

El automorfismo de grupo de un grafo $G = (V,E)$ denotado por Aut(G) es el conjunto de todas las permutaciones de V que son automorfismos de G. Por lo tanto Aut(G) consiste de todas las permutaciones de V que componen, como un conjunto, las aristas E de G. [14]

Por ejemplo, la figura 2.10 muestra un grafo G junto con todos sus automorfismos

Aut (G) = {I, (1, 2), (0, 5), (0, 5)(1, 2), (0, 2)(1, 5)(3, 4), (0, 1)(2, 5)(3, 4), (0, 1, 5, 2)(3, 4), (0, 2, 5, 1)(3, 4)}

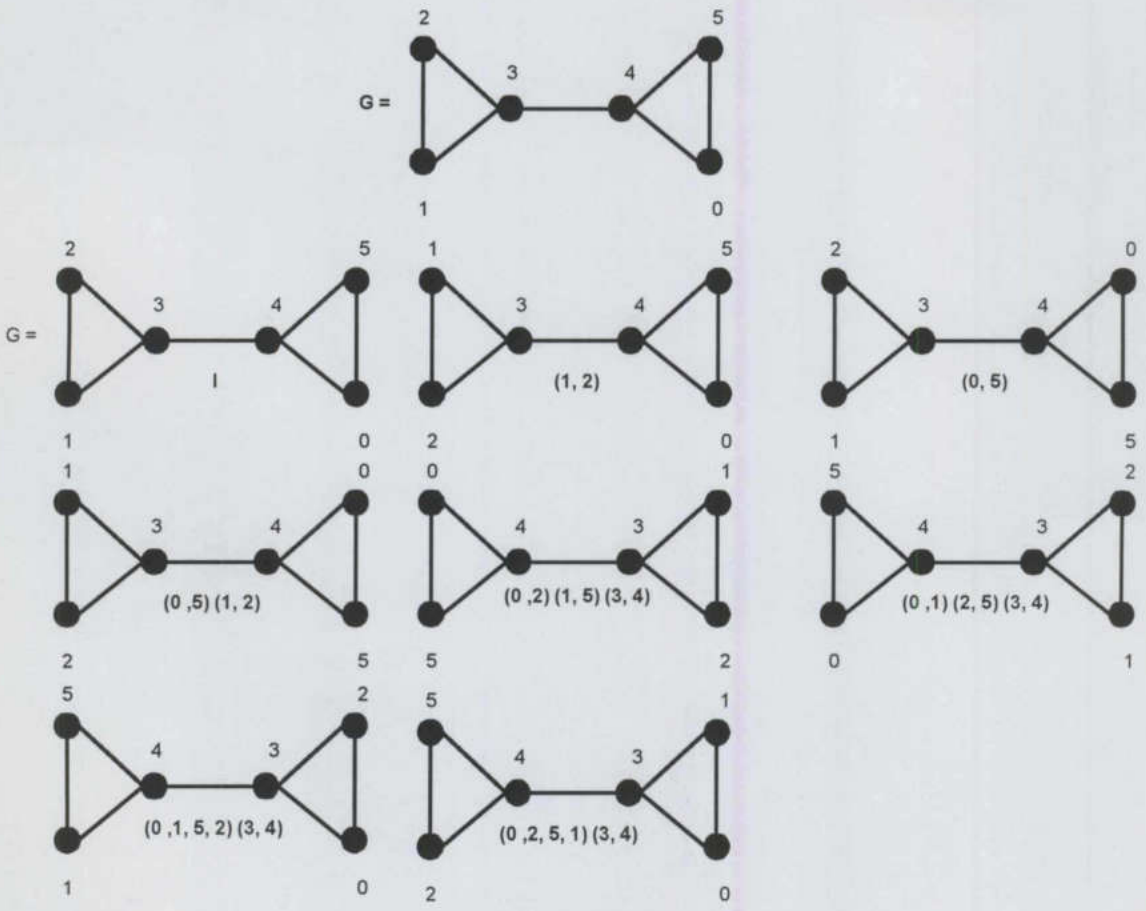


Figura 2.10 *Automorfismos de grafo*

CAPÍTULO 3
ISOMORFISMO

3. 1. Importancia

Las nociones de isomorfismo y simetría son consideradas como los conceptos más importantes de todas las áreas de la computación. El isomorfismo consiste en una prueba que se puede aplicar para detectar simetría entre objetos. Se entiende por simétrico algo bien proporcionado y bien equilibrado, aquel tipo de concordancia por la cual diversas partes se integran en un todo. La belleza va ligada a la simetría. La simetría tanto en su significado más amplio como el más restringido, es una idea a través de la cual el hombre, en todos los tiempos, ha tratado de captar y de crear el orden, la belleza y la perfección. La simetría significa reposo y ligazón, asimetría movimiento y soltura; una ley y orden, la otra arbitrariedad y accidente; la primera rigidez formal y restricción, la segunda vida, juego y libertad. La teoría de la relatividad es un aspecto más de la simetría. [22].

El isomorfismo es tema central de este trabajo de investigación. El isomorfismo emplea algoritmos combinatorios aplicados no sólo en la enumeración de clases de equivalencia y la selección de configuraciones representativas importantes, sino también en la eliminación de cómputo repetitivo. La detección de estructuras isomorfas es importante en la construcción de algoritmos prácticos. [14]

Los grafos fueron utilizados en 1984 para representar átomos en una molécula. Esa idea fue introducida por Alexander Crum Brown, quien explicó por primera vez el fenómeno de *Isomerismo*- la existencia de moléculas con la misma fórmula química pero diferentes propiedades químicas [21]

Aunque los diagramas parecidos a grafos fueron utilizados cerca de 1789 para representar moléculas químicas, no fue hasta 1850 que las ideas sobre átomos y la forma de combinarlos fueron suficientemente bien entendidas para que los diagramas principales se representaran.[7]

Si se proporcionan las siguientes instrucciones a dos personas, quienes no pueden ver lo que dibuja la otra persona: "trace y etiquete cinco vértices como *a, b, c, d y e*. Una *a* con *b*, *b* con *c*, *c* con *d*, *d* con *e y e* con *a*." Es seguro que estas figuras definen el mismo grafo aunque parezcan diferentes. Los grafos producidos son isomorfos. [12]

La palabra *isomorfo* deriva del griego *iso* y *morfo* que quiere decir "misma forma" y en términos matemáticos se refieres a dos o más estructuras que conservan la forma y con ello las propiedades, pero que su presentación difiere una de la otra.

El problema del isomorfismo de grafos consiste en determinar cuándo dos grafos son isomorfos. Aunque los algoritmos conocidos para resolver este problema son de orden exponencial o factorial en el peor de los casos, existen algoritmos de orden lineal que permiten determinar el isomorfismo entre un par de grafos pertenecientes a familias específicas. [14]

El primer intento para resolver el problema del isomorfismo de grafos consiste en analizar la matriz de adyacencia de cada grafo. Esta verificación requiere p^2 operaciones, donde p es el número de vértices del grafo. Pero al cambiar el orden de las filas y las columnas de la matriz de adyacencia se obtiene una matriz diferente y se tiene que verificar todas las posibles permutaciones de filas y columnas, este último cálculo es de orden factorial.

Posterior a ello surgieron dos métodos utilizados para resolver el problema de isomorfismo de grafos. El primer método, busca un mapeo directo entre los dos grafos. El algoritmo consiste en tomar nodos de cada uno de los grafos tales que conserven la adyacencia entre los nodos, este paso continúa hasta terminar con todos los nodos o hasta que un nodo en un grafo no tenga pareja con un nodo del otro grafo. Este método utiliza tiempo exponencial

en el peor de los casos y tiene la desventaja de que si dos grafos tienen varios isomorfismos sólo encontrará el primer isomorfismo.

El segundo método, obtiene un etiquetado canónico para cada uno de los grafos y los compara. La idea base es generar un código único, llamado *certificado*, para cada grafo, de forma tal que si son isomorfos tendrán el mismo certificado. Este segundo método se basa en el concepto de *invariante*, que es una propiedad de un grafo independiente del nombre de sus vértices, y resulta ser más efectivo ya que obtener un invariante de un grafo podría tomar un tiempo polinomial. Este último enfoque es precisamente la base del presente trabajo de investigación.

Para ilustrar estas ideas se discutirá el problema del isomorfismo de grafos precisando la idea de que dos grafos son estructuralmente iguales o isomorfos si son idénticos excepto por el nombre de los vértices. Se presenta un algoritmo general para isomorfismo de grafos que utiliza el método de invariantes o refinamiento repetitivo, posteriormente se presenta un algoritmo que es, en general, superior al método de invariantes, el cual utiliza el método de certificados o etiquetado canónico. Finalmente se verán aplicaciones de los mencionados algoritmos a árboles.

3.2. Invariantes

Dos grafos $G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$ son *isomorfos* si hay una biyección $f: V_1 \rightarrow V_2$ tal que

$$\{f(x), f(y)\} \in E_2 \text{ si y solo si } \{x, y\} \in E_1$$

Un isomorfismo para los grafos G_1 y G_2 de la figura 3.1 está dado por la función

$$f = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ i & a & d & e & g & f & c & b & h & j \end{pmatrix}$$

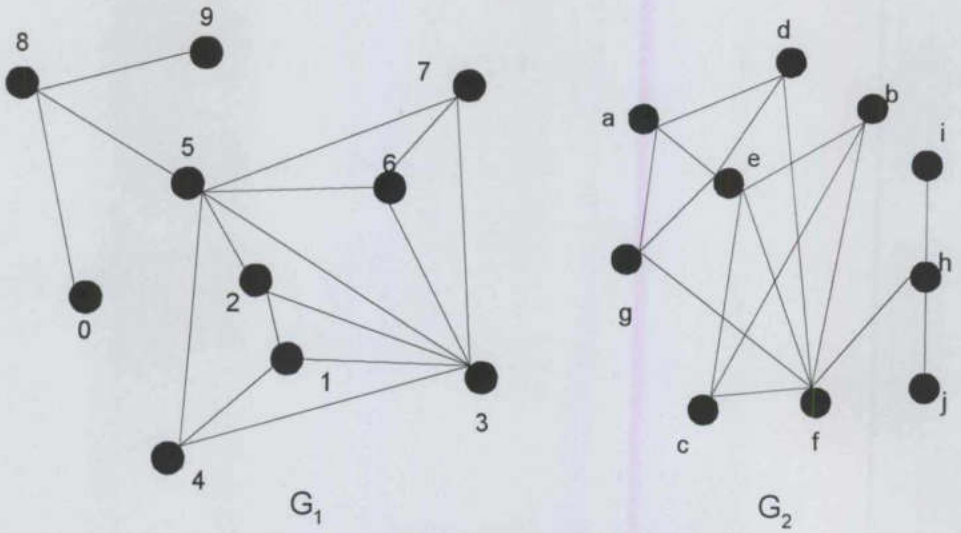


Figura 3.1 *Grafos Isomorfos*

Si f es un isomorfismo de un grafo consigo mismo, entonces f se llama *automorfismo*, como se vio en el capítulo 2.

Una función φ es un *invariante* del grafo $G = \{V, E\}$ si φ no depende de la forma como se etiqueta el grafo G, sino solamente de su estructura.

Algunos ejemplos de invariantes se observan en la figura 3.2

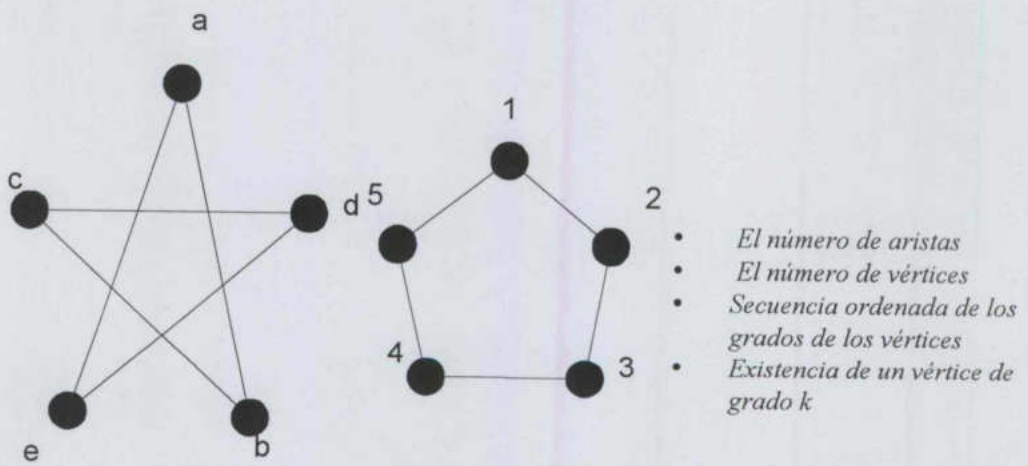


Figura 3.2 *Invariantes de un Grafo*

Considérese la familia de grafos F (por ejemplo la familia de todos los grafos con n vértices). Un invariante en F es una función φ con dominio F tal que

$$\varphi(G_1) = \varphi(G_2) \text{ si } G_1 \text{ es isomorfo a } G_2$$

Según la definición previa de isomorfismo existe una biyección de las aristas de G_1 sobre las aristas de G_2. Así G_1 y G_2 son isomorfas, y entonces ambos tienen el mismo número de aristas y el mismo número de vértices. Por lo tanto, la propiedad "número de aristas" y "número de vértices" son invariantes.

El determinante[1] de la matriz de adyacencia de un grafo es un invariante. Pero la matriz de adyacencia no es un invariante, puesto que un grafo puede dar lugar a matrices diferentes.

Teorema.
Si A' es la matriz que se obtiene al intercambiar dos renglones de A, entonces

$$Det(A') = -Det(A)$$

La matriz de los grafos analizados es una matriz cuadrada y simétrica. Por cada cambio de renglón se hace un cambio de columna, para conservar la simetría, esto quiere decir que se hacen siempre un número par de movimientos de la matriz y si se aplica el teorema anterior el determinante siempre es el mismo. Por lo tanto, el determinante de la matriz de adyacencia de un grafo es un invariante, mas aún, es un certificado. [2]

La figura 3.3 muestra dos grafos isomorfos.

Para obtener la matriz de G_2 a partir de G_1 hay que hacer los siguientes movimientos

1. Cambiar las columnas 2 con 5 y por lo tanto cambiar también las filas 2 con 5
2. Cambiar las columnas 2 con 3 y las filas 2 con 3

[1] El determinante de una matriz, denotado por *Det(A)*, es la suma de todos sus productos elementales, un producto elemental es cualquier producto de n elementos de A sin que dos cualesquiera de ellos provengan del mismo renglón o la misma columna. [2]

Este procedimiento se muestra después de las matrices de adyacencia correspondiente a los grafos de la figura 3.3.

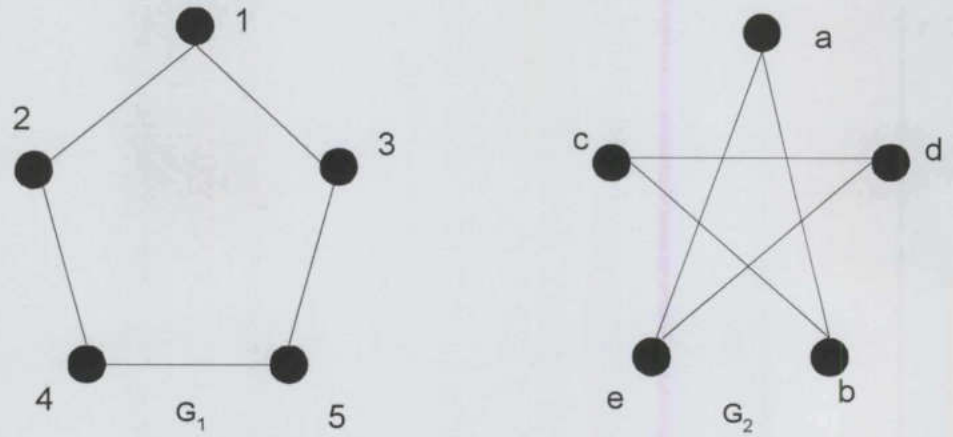


Figura 3.3 *Determinante de un grafo*

$$G_1 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \qquad G_2 = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$G_1 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \rightarrow G_1' = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \rightarrow G_1'' = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix} \qquad G_1'' = G_2$$

Estos pasos involucraron cuatro movimientos por lo tanto el determinante se expresaría de la siguiente forma

Det $(G_1) = 2$

Det$(G_2) = (-1)^4\, 2$

Det $(G_2) = 2$

En *Mathematica* existe la función *Det[ ]* para obtener el determinante de una matriz dada.

Con la definición de isomorfismo dada con anterioridad, se puede decir que si $\varphi(G_1) \neq \varphi(G_2)$ entonces G_1 y G_2 no son isomorfos, pero si $\varphi(G_1) = \varphi(G_2)$ no se puede concluir que G_1 y G_2 son isomorfos. Esta última conclusión se observa en la figura 3.4

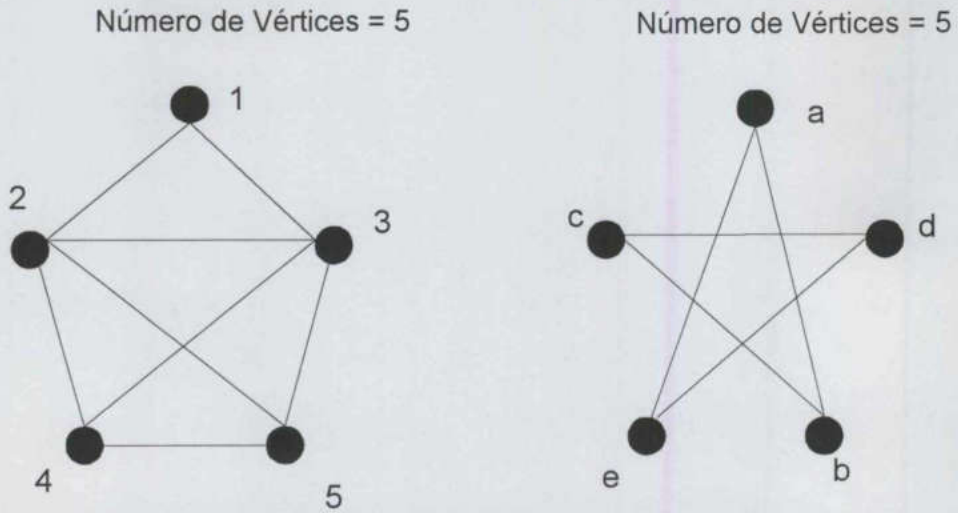


Figura 3.4 *Propiedad "número de vértices" que prueba que no es invariante suficiente de grafos.*

Del variado grupo de invariantes, se deben seleccionar aquellos que sean fáciles de obtener y que distinguen a varios grafos.

Un método para detectar isomorfismo con la ayuda de funciones invariantes, consiste en generar particiones de los vértices en cada uno de los grafos a analizar. Si se aplica la misma función invariante a dos grafos y el tamaño de las particiones es el mismo, lo único que se tiene que hacer es encontrar la correspondencia en las particiones equivalentes. Este método se explica a continuación.

Una *partición ordenada* B de V es una lista ordenada

$$B=[B[0], B[1], B[2], \ldots B[k-1]]$$

Donde $\{B[0], B[1], \ldots B[k-1]\}$ es una partición de V. El tamaño k de la partición es definido por $|B|$.

Por ejemplo, G es un grafo con $V=\{a, b, c, d, e, f\}$, como se muestra en la figura 3.5, y la función $deg_G(v)$, que determina el grado del vértice v en G, y $k= 6$, entonces

$B=[B[0], B[1], \ldots B[5]]$

definida por

$B[i]= \{v \in V \mid deg_G(v)=i\}$

Está dado por

$B[0] = \{\}, B[1] = \{\}, B[2] = \{a, d\}, B[3] = \{b, f\}, B[4] = \{c, e\}, B[5] = \{\}$

y entonces

$B= [\{\}, \{\}, \{a, d\}, \{b, f\}, \{c, e\}, \{\}]$ es una partición ordenada de los vértices de G.

Por lo tanto la función $\varphi(G) = [|B[0]|, |B[1]|, |B[2]|, \ldots |B[k-1]|]$ es un invariante para una familia de grafos. Esto se debe a que $|B[i]|$ es el número de vértices de grado i en G. Si dos grafos con n vértices son isomorfos, entonces tendrán el mismo número de vértices de grado i, para cada $i = 0, 1, 2, \ldots n-1$.

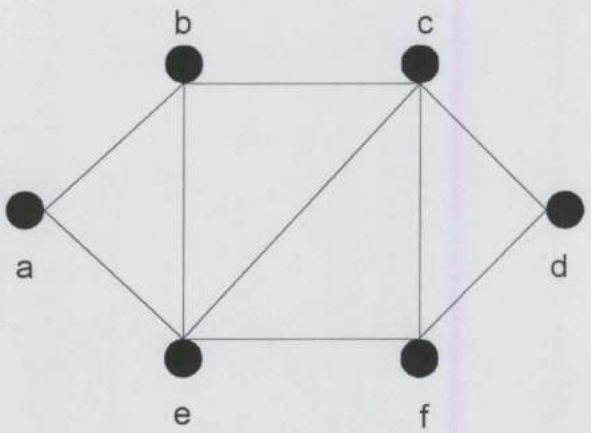


Figura 3.5 *Grafo G*

Para el ejemplo de la figura 3.5, $\varphi(G) = [0, 0, 2, 2, 2, 0]$.

Si F es una familia de grafos en el conjunto de vértices $V = \{v_1, v_2, \ldots v_n\}$ y D es una función con dominio $(F \times V)$, entonces la *partición inducida* por D es

$B=[B[0], B[1], B[2], \ldots B[n-1]]$

Donde $B[i] = \{v \in V \mid D(G, v) = i\}$

Si la función $\varphi_D(G) = [|B[0]|, |B[1]|, |B[2]|, \ldots |B[n-1]|]$ es un invariante, entonces se dice que D es una *función invariante inducida.*

Sería fácil determinar si un par de grafos son isomorfos o no, si se pudiera verificar rápidamente un número pequeño de invariantes, los cuáles son compartidos por los grafos isomorfos y únicamente por ellos. Este método es tardado y hasta la fecha no se ha encontrado ese "número pequeño de invariantes". Para poder solucionar más rápido el problema de isomorfismo, se utiliza el método de los certificados, este método es utilizado por los algoritmos más rápidos de búsqueda de isomorfismo entre grafos.[14]

Un *certificado Cert( )* para una familia de grafos F es una función tal que para algunos G_1, $G_2 \in F$,

$\text{Cert}(G_1) = \text{Cert}(G_2)$ si y solo si G_1 y G_2 son isomorfos.

De lo anterior se puede concluir que un certificado es un invariante.

3.3. Certificados de Árboles

El certificado de un árbol es una cadena de longitud 2n formada por ceros y unos, donde $n = |V|$. El método para obtener el certificado es el siguiente:

1. Etiquetar todos los vértices del árbol con la cadena "01"
2. Si hay más de dos vértices en el árbol
   Para cada vértice x que no es hoja hacer
   (a) Eliminar el 0 inicial y el 1 final de la etiqueta, el resto almacenarlo en un conjunto Y, junto con la etiqueta de las hojas adyacentes a x.
   (b) Ordenar lexicográficamente el conjunto Y creando una sola cadena
   (c) Actualizar la etiqueta de x con la cadena del conjunto Y. Concatenar el 0 inicial y el 1 final de x
   (d) Remover todas las hojas adyacentes a x.

3. Si hay sólo un vértice, es decir, si el árbol tiene un centro, el certificado es la etiqueta de ese vértice.
4. Si hay dos vértices, ésto es, un árbol que tiene dos centros, el certificado es la concatenación de sus etiquetas ordenadas lexicográficamente.

La figura 3.6, muestra la aplicación del algoritmo a un árbol con un centro, para n = 15, por lo que el certificado es de longitud 30. El centro del árbol es el vértice 7 y el certificado del árbol es 000001101100011110000110110111

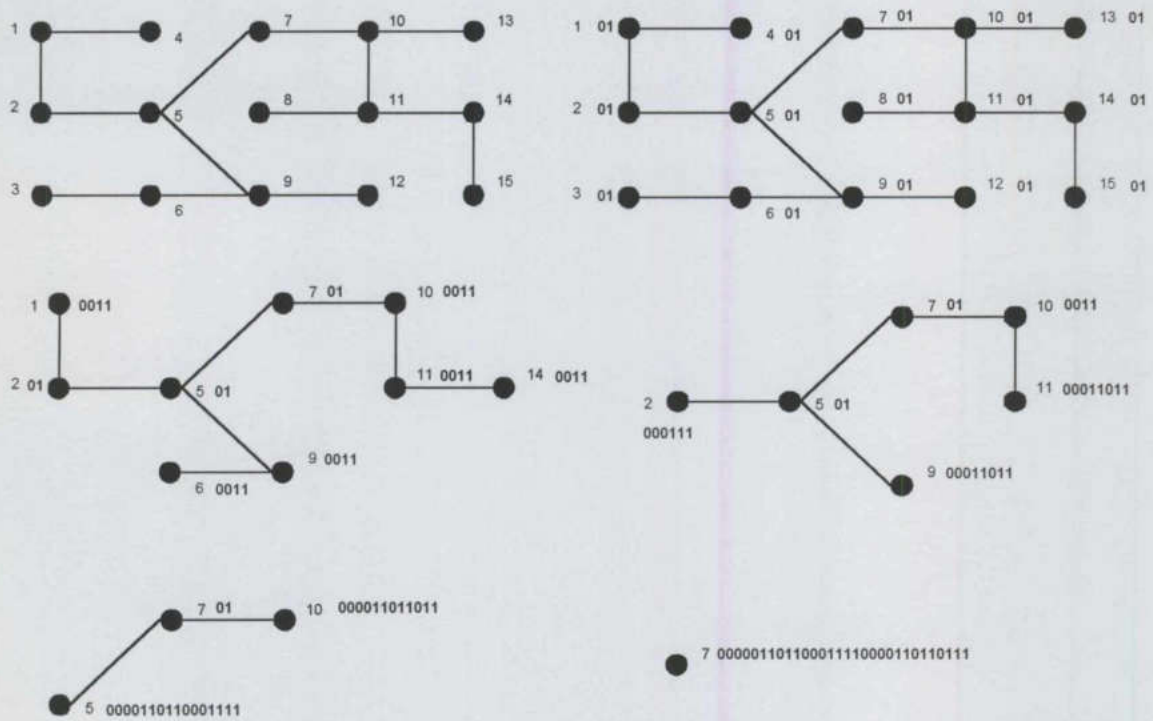


Figura 3.6 *Certificado de un árbol con un centro*

La figura 3.7, muestra ahora un árbol con dos centros. En este ejemplo, $n = 10$, por lo que el certificado es de longitud 20. El centro del árbol está formado por los vértices 1 y 4 y el certificado del árbol es 00010110011100011011

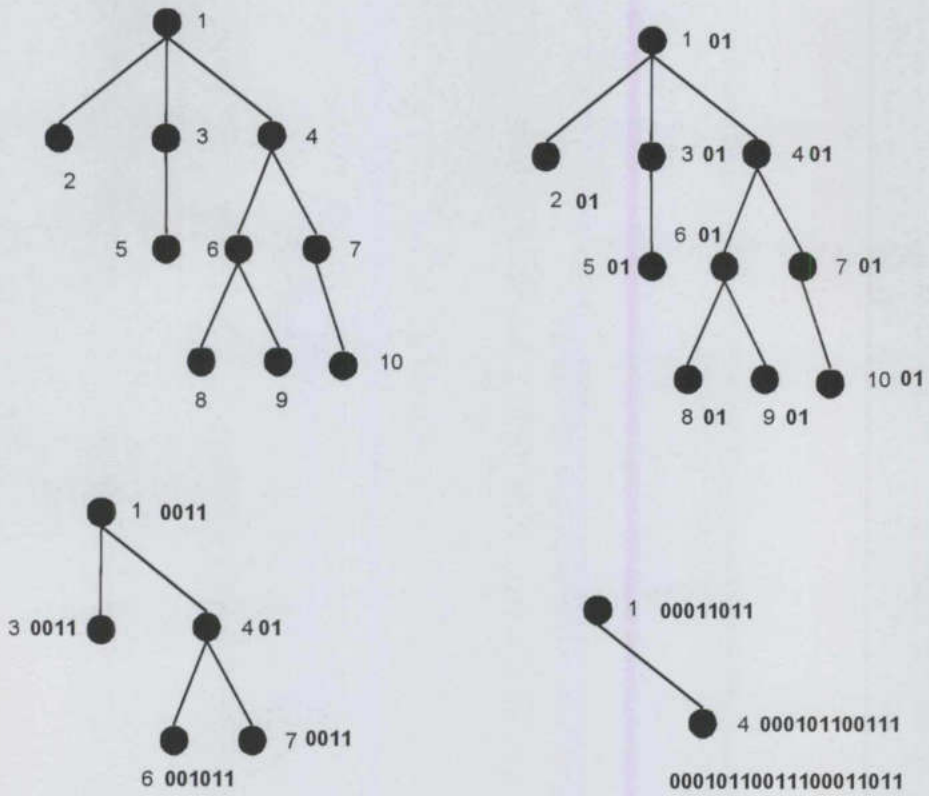


Figura 3.7. *Certificado de un árbol con dos centros.*

El siguiente paso es convertir un certificado válido en su árbol correspondiente. No se obtendrá el árbol exacto, se obtiene un árbol isomorfo a él.

Una cadena válida como certificado cumple con las siguientes características:

1. El número de 1´s y 0´s es el mismo y corresponde a la mitad de la longitud de la cadena, es decir, el número de vértices del árbol.

2. Todas las cadenas inician con un 0 y terminan con un 1.

3. La longitud de la primer secuencia de 1´s debe ser menor o igual que la longitud de la primer secuencia de 0´s.

4. Si el árbol tiene un centro de un vértice, el número de 1´s es igual al número de 0´s de la cadena hasta el término de la cadena.

5. Si el centro del árbol está compuesto por dos vértices, el número de 1´s es igual al número de 0´s en posiciones intermedias de la cadena.

Se analizaron dos métodos para la obtención del árbol a partir de un certificado válido dado. Uno de ellos es gráfico, lo que dificulta su implementación en la computadora, por ello, el segundo método es el que se implementó para su análisis.

El primer método, llamado "*Teorema del Sol Naciente*", obtiene una gráfica simulando unas montañas. Si trazamos estas montañas en el plano se tendrían subidas y bajadas desde la coordenada (0,0) hasta la coordenada (0, 2n). Los valores de Y simulan el "nivel del mar".

El algoritmo para crear esta gráfica es el siguiente:

1. Dibujar un eje, donde el plano de X son las posiciones de cada uno de los elementos del certificado y el plano de y son los valores acumulados de la cadena, dependiendo del valor que se haya leído
2. Iniciar en el punto (0,0)

3. Para i desde 1 hasta la longitud del certificado ($2n$)
   si certificado[i] = 0 incrementa en uno el valor de y,
   si certificado[i] = 1 disminuye en uno el valor de y.

4. Unir los puntos de la gráfica

Hasta este momento se tiene la gráfica de todos los elementos del certificado. Ahora se procede a obtener el árbol correspondiente. Para y = 0, primer nivel del mar, los inicios de las montañas chocarán con el nivel del mar dos o tres veces, dependiendo si el número de

vértices que tiene el centro del árbol es uno o dos vértices, respectivamente. Cada una de las montañas que hay en la gráfica corresponden a los vértices del árbol, si son dos o más montañas, serán vértices adyacentes. A medida que sube el nivel del mar, las montañas se van dividiendo en submontañas, estas submontañas corresponden a vértices hijos de la montaña que las ha generado.

5. Incrementar el valor de y y trazar los vértices de acuerdo a la anterior explicación.

La figura 3.8, muestra el algoritmo aplicado al árbol de la figura 3.6.

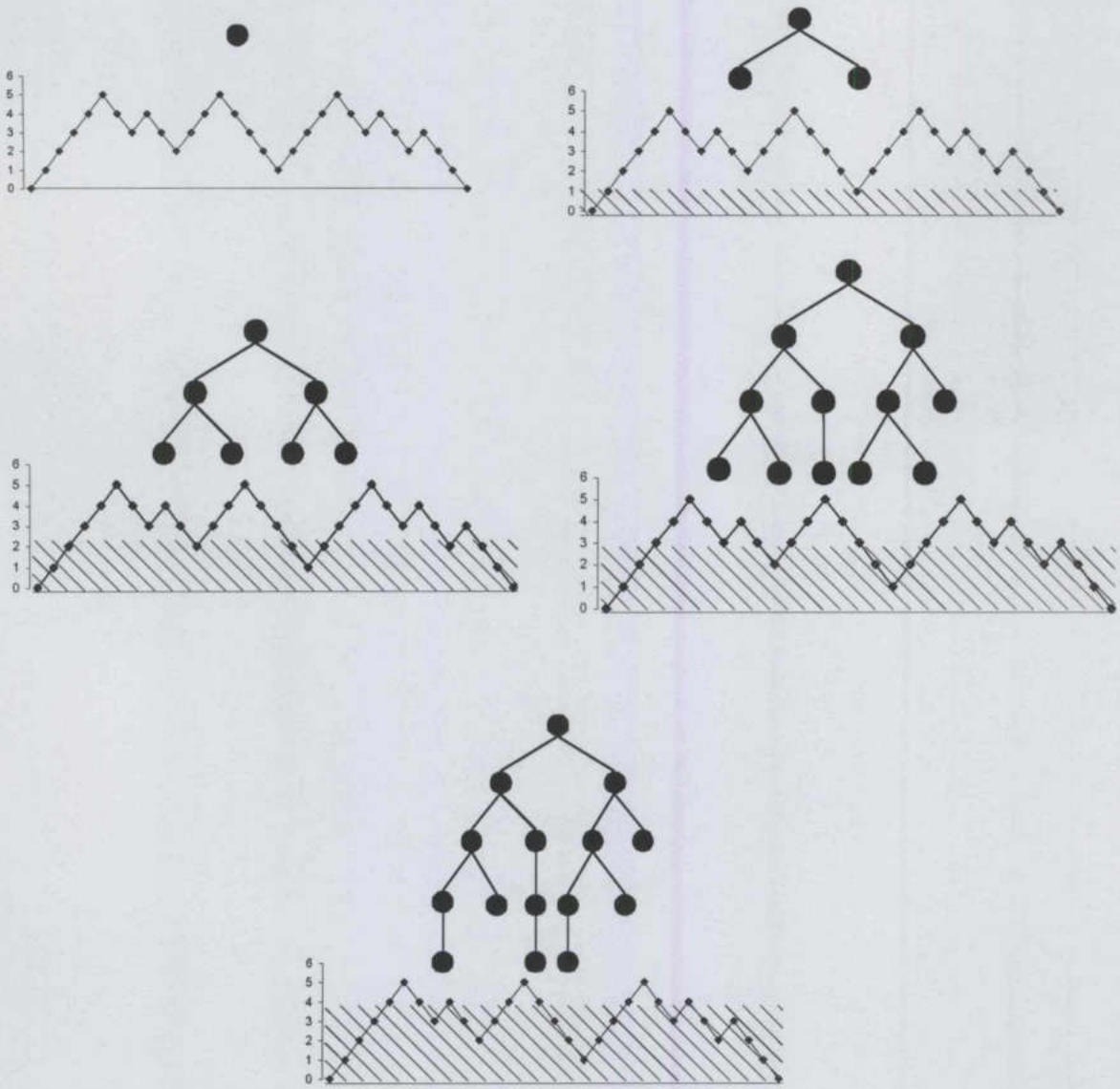


Figura 3.8 *Generación de un árbol de un centro a partir de un certificado*

La figura 3.9, muestra el algoritmo aplicado al árbol de la figura 3.7

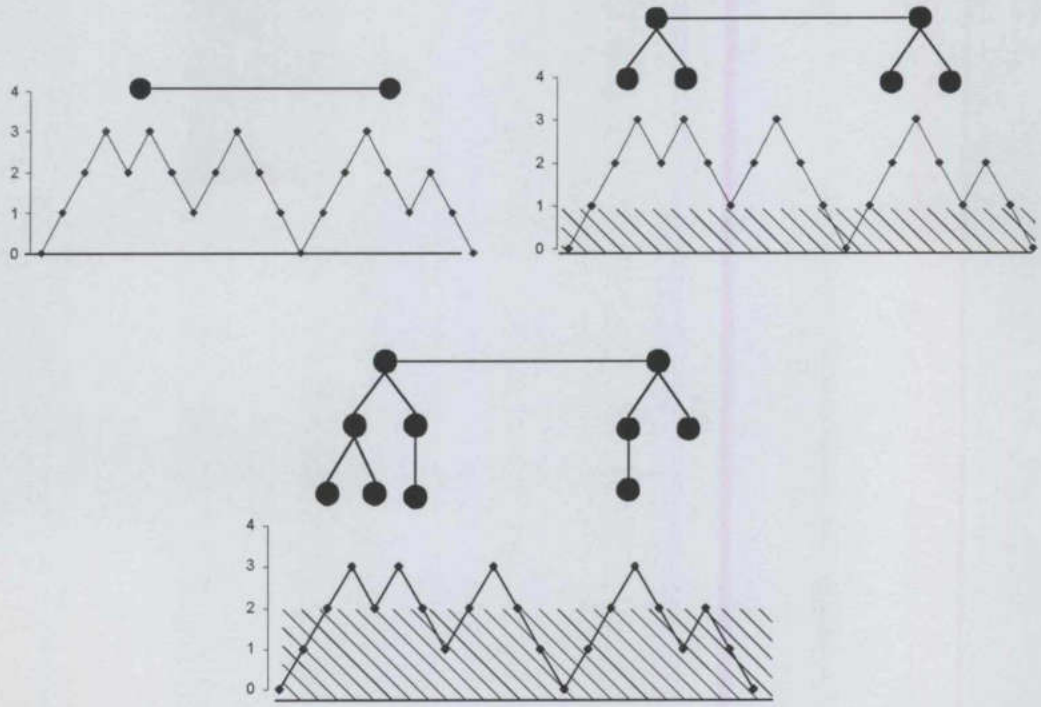


Figura 3.9 *Generación de un árbol de dos centros a partir de un certificado*

Este primer método parece ser bastante sencillo de entender y de utilizar para las personas, sin embargo, si quisiéramos implementarlo en la computadora, de hecho es lo ideal, sería un método bastante complejo y tardado. Para solucionar este problema, se diseño el siguiente método basado en la cadena que se obtiene como certificado y quitando la explicación gráfica.

El segundo método, crea el árbol en la medida en que se va leyendo la cadena que forma el certificado. Para este método se sigue el siguiente algoritmo:

1. Verificar cuantos vértices forman el centro del árbol, ya que el método es diferente para cada uno de los dos casos. Para ello se hace lo siguiente:

2. Repetir desde 1 hasta la longitud de la cadena

Si cadena[i]=0,

Incrementar número de 0´s

Si cadena[i]=1,

Incrementar número de 1´s

Si número de 0´s = número de 1´s y no se ha llegado a la longitud de la cadena, entonces el certificado corresponde a un árbol de dos centros.

En caso contrario, es un árbol de un centro.

3. Repetir recursivamente hasta que termine la cadena

Si el árbol tiene un vértice como centro

4. Remover certificado[i] y certificado [n], estos dos elementos forman un nodo padre f. Los nodos que se generen en los siguientes pasos son hijos de este nodo.

Si el árbol tiene dos vértices como centro

5. Dividir la cadena en igual numero de 0´s que de 1´s. Cada una de estas son los nodos que forman el centro del árbol, por lo que estarán unidas por una arista.
6. Dividir la cadena en igual número de 0´s que de 1´s. Cada una de estas subcadenas forma un hijo del nodo f.

7. Repetir los pasos 1 al 3 para cada una de las subcadenas.

Por ejemplo, dada la siguiente cadena 001010101011 de longitud $2n$=12, por lo tanto el árbol a obtener tendrá 6 vértices. Como el número de 0´s y de 1´s es igual hasta que termina la cadena, el árbol tiene un vértice como centro. El procedimiento se ilustra en la figura 3.10

En la figura 3.11 se muestra un segundo ejemplo, la cadena es 0010110011, por lo tanto el árbol a obtener tendrá 5 vértices. Como el número de 0´s y de 1´s es igual y la cadena no

ha terminado de ser leída, el árbol tiene dos vértice como centro. El procedimiento sería el que muestra la figura 3.11

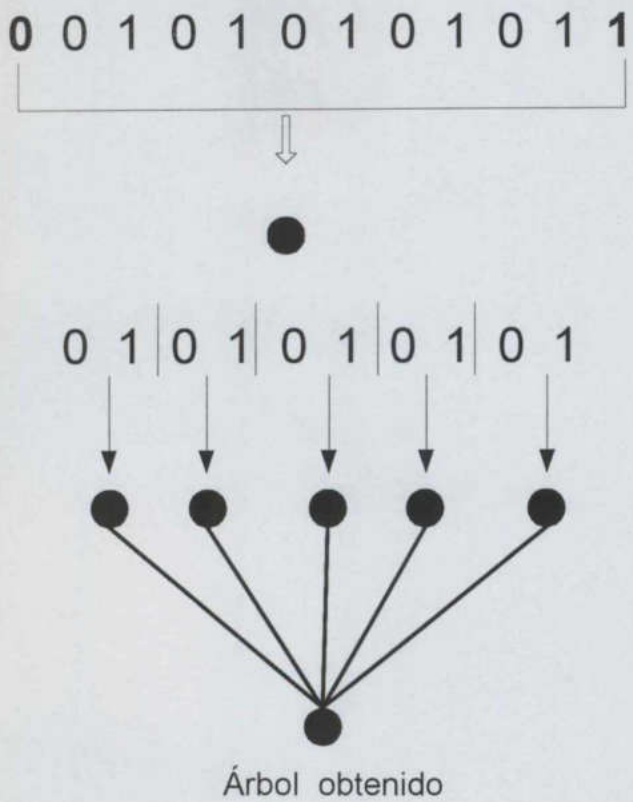


Figura 3.10 *Otra forma de obtener un árbol de un centro a partir de un certificado*

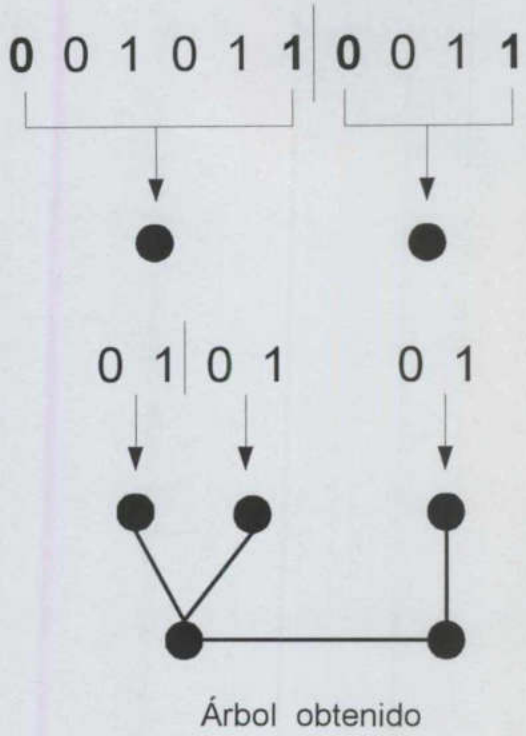


Figura 3.11 *Otra forma de obtener un árbol de dos centros a partir de un certificado*

En cada uno de los métodos de conversión de un certificado a su correspondiente árbol, es necesario verificar cuándo la cadena dada como certificado es válida. Para ello se toman en cuenta las siguientes condiciones de la cadena:

1. Existe el mismo número de 0's que de 1´s.
2. Inicia con 0 y termina con 1.
3. La longitud de la primera cadena de 1´s debe ser menor o igual que la longitud de la primera cadena de 0´s.
4. La longitud del certificado siempre es un número par.

Ejemplos de certificados inválidos:

00	No existen 1's
001	No hay igual número de 1's que de 0's
1010	No inicia con 0, ni termina con 1
0001111001	Hay tres 0's y después hay 4 1's

La programación de la técnica de obtención de certificados y árboles se explica a continuación.

Dado un certificado c, representando el 0 como 1 y el 1 como –1

```
c = {1,1,1,-1,1,-1,-1,1,1,-1,-1,-1,1,1,1,-1,-1,1,-1,-1};
```

se obtienen las sumas de cada uno de esos elementos, posición por posición. Éste es el paso que se hace para ir obteniendo los puntos en el "*Teorema del Sol Naciente*". El 0 indica que sube y el 1 indica que baja.

```
s = 0;
h = Map [If [# = =1, s++, s--;s] &,c]

{1, 2, 3, 2, 3, 2, 1, 2, 3, 2, 1, 0, 1, 2, 3, 2, 1, 2, 1, 0}
```

observemos que los elementos de *h* son mayores a 0. Esto se justifica si es un certificado válido, si no lo fuera como en el siguiente caso, se obtendrían sumas con valores menores a cero. Esto indicaría que hay valores por debajo del "nivel del mar", además no cumpliría con las características de un certificado válido.

```
c = {1,1,1,-1,-1,-1,-1,1,1,-1,-1,-1,1,1,1,-1,-1,1,-1,-1};
s = 0;
h = Map[(If[# == 1, s++, s--]; s) &, c]
{1, 2, 3, 2, 1, 0, -1, 0, 1, 0, -1, -2, -1, 0, 1, 0, -1, 0, -1, -2}
```

Se obtienen las posiciones en las cuales aparece el elemento 0, este resultado indica cuantos centros tiene el certificado y en que posiciones. Este cero indica en cuantos vértices se va a dividir el vértice padre. Posterior a ello se particiona el certificado en uno o dos subconjuntos si el certificado tiene dos o un centro, respectivamente.

```
p = Partition[Flatten[{1, Position[h, 0]}], 2, 1]
Map[Drop[Rest[Take[c, #]], -1] &, p]
```

```
{{1, 12}, {12, 20}}
```

```
{{1, 1, -1, 1, -1, -1, 1, 1, -1, -1}, {1, 1, 1, -1, -1, 1, -1}}
```

La función *cert2Tree* reúne en un módulo cada una de las funciones anteriores y obtiene el árbol para dicho certificado.

```
Clear[cert2Tree]
cert2Tree[{}] := 0
cert2Tree[c_] := Module[{s = 0, h, p},
    h = Join[{0}, Map[(If[# == 1, s++, s--]; s) &, c]];
    p = Partition[Flatten[{Position[h, 0]}], 2, 1];
    cert2Tree/@ Map[Drop[Rest[Take[c,{First[#],Last[#]-1}]],1]&,p]]
```

Por ejemplo *cert2Tree [c]*

```
t = cert2Tree[c]
{{{0, 0}, {0}}, {{0}, 0}}
```

La interpretación que se le da al resultado es que cada lista que contenga el resultado corresponde a un vértice, si el 0 esta en una lista quiere decir que es un vértice y un hijo, pero si es un 0 sin llaves, entonces es sólo un vértice, la correspondencia es como sigue:

0	Nodo
{0}	Nodo con un hijo
{}	Nodo
{0,0, ...}	Nodo con n hijos

En el ejemplo, efectivamente se construye el árbol de la figura 3.7

Aplicando el método al grafo de la figura 3.6 se obtiene

```
Cert2Tree[c3]
{{{{{0}, 0}, {{0}}}, {{{0}, 0}, 0}}}
```

Ahora se tiene el proceso inverso, dado el árbol se quiere obtener el certificado, esto se hace a través de la función *tree2Cert*

```
Clear[tree2Cert, t2C]
tree2Cert[0] := {1, -1}
tree2Cert[{t_}] := Flatten[{1, tree2Cert[t], -1}]
tree2Cert[{lT_, rT_}] := Module[{t1, t2},
    t1 = tree2Cert[lT];
    t2 = tree2Cert[rT];
    Flatten[{1, t1, t2, -1}]]
t2C[t_] := Drop[Rest[tree2Cert[t]], -1]
```

La función *tree2Cert* y la función *T2C* obtienen un certificado a partir del árbol generado por la función *cert2Tree*

```
tree2Cert[t]
{1,1,-1,1,-1,-1}
t2C[t]
{1,-1,1,-1}
c
{1,1,1,-1,1,-1,-1,1,1,-1,-1,-1,1,1,1,-1,-1,1,-1,-1,}
```

Aplicándolo al certificado *c3*

```
t2C[c3]
{1, 1, 1, 1, 1, -1, -1, 1, -1, -1, 1, 1, 1, -1, -1, -1, -1, 1, 1,
1, 1, -1, -1, 1, -1, -1, 1, -1, -1, -1}
```

Para verificarlo se hace el proceso inverso con la función *cert2Tree*

```
cert2Tree[{1, 1, -1, -1}]
{{0}}
t2C [{{0}}]
{1, 1, -1, -1}
```

Finalmente para generalizar el método se generan todos los posibles certificados de tamaño n que cumplan con las características dichas anteriormente.

Primero se obtienen todas las combinaciones de 0´s y 1´s

```
n = 4
IntegerDigits[Range[0, 2^n-1-2^(n-1)],2,n]
{{0, 0, 0, 0}, {0, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 1, 1},
{0, 1, 0, 0}, {0, 1, 0, 1}, {0, 1, 1, 0}, {0, 1, 1, 1}}
```

Luego, se seleccionan aquellos que cumplen con las características ya descritas anteriormente.

```
Table[
{n, s = Select[IntegerDigits[Range[0, 2^n-1-2^(n-1)],2,n] /. {0 →-1},
cert2Tree[#] ≠ {}&],Length[s]},{n,2,4,2}]
{{2, {{-1, 1}}, 1},
{4, {{-1, -1, 1, 1}, {-1, 1, -1, -1}, {-1, 1, -1, 1},
{-1, 1, 1, -1}, {-1, 1, 1, 1}}, 5}}
```

Finalmente se obtienen los certificados con las siguientes selecciones

```
Select[Select[Select[IntegerDigits[Range[0, 2^n - 1 - 2^n - 1], 2, n],
First[Reverse[#]] == 1 &] /. {0 →-1},
cert2Tree[#] != {} &], Plus @@ # == 0 &]
{{-1, -1, 1, 1}, {-1, 1, -1, 1}}
```

Ahora se procede a implementar las funciones para el isomorfismo

Para probar los algoritmos de esta programación se utilizarán las matrices *G1, G2 y G3*, estas matrices representan un grafo cada una de ellas.

$$G1 = \begin{pmatrix} 0&0&0&0&0&0&0&0&1&0\\ 0&0&1&1&1&0&0&0&0&0\\ 0&1&0&1&0&1&0&0&0&0\\ 0&1&1&0&1&1&1&1&0&0\\ 0&1&0&1&0&1&0&0&0&0\\ 0&0&1&1&1&0&1&1&1&0\\ 0&0&0&1&0&1&0&1&0&0\\ 0&0&0&1&0&1&1&0&0&0\\ 1&0&0&0&0&1&0&0&0&1\\ 0&0&0&0&0&0&0&0&1&0 \end{pmatrix};\quad G2 = \begin{pmatrix} 0&0&0&1&1&0&1&0&0&0\\ 0&0&1&0&1&1&0&0&0&0\\ 0&1&0&0&1&1&0&0&0&0\\ 1&0&0&0&1&1&0&0&0&0\\ 1&1&1&1&0&1&1&0&0&0\\ 0&1&1&1&1&0&1&1&0&0\\ 1&0&0&0&1&1&0&0&0&0\\ 0&0&0&0&0&1&0&0&1&1\\ 0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1&0&0 \end{pmatrix};\quad G3 = \begin{pmatrix} 0&1&0&1\\ 1&0&1&1\\ 0&1&0&0\\ 1&1&0&0 \end{pmatrix};$$

Se utilizaran dos invariantes *D1* y *D2*. Si al aplicar estos dos invariantes a dos o más grafos nos devuelve el mismo resultado, entonces los grafos son isomorfos y más que eso, la función devuelve todos los isomorfismos entre ambos grafos. De tal forma que si hacemos la prueba de isomorfismo con el mismo grafo se obtendrán todos sus automorfismos. Si al aplicar *D1* y *D2* los resultados son distintos los grafos no son isomorfos.

D1 es la lista de los grados de los vértices

```
deg[G_,v_]:= Plus @@ G[[v]]
D1[G_]:= deg [G, #]& /@ Range[Length[G]]

D1[G1]
{1, 3, 3, 6, 3, 6, 3, 3, 3, 1}

D1[G2]
{3, 3, 3, 3, 6, 6, 3, 3, 1, 1}

D1[G3]
{2, 3, 1, 2}
```

D2 es una función invariante más sofisticada. Se ejemplifica su descripción con la matriz *G1*.

Primero se construye una tabla del tamaño n, donde $n = |V|$. Posteriormente se obtienen $N_v(G)$ para $\forall v \in G$, este resultado es almacenado en *s1*.

```
s1 = Flatten[Position[#, 1]] & /@ G
{{9}, {3, 4, 5}, {2, 4, 6}, {2, 3, 5, 6, 7, 8}, {2, 4, 6},
{3, 4, 5, 7, 8, 9}, {4, 6, 8}, {4, 6, 7}, {1, 6, 10}, {9}}
```

La siguiente instrucción, *s2*, obtiene el grado de cada uno de los elementos de $N_v(G)$.

```
s2 = D1[G][[#]] & /@ s1
{{3}, {3, 6, 3}, {3, 6, 6}, {3, 3, 3, 6, 3, 3}, {3, 6, 6},
{3, 6, 3, 3, 3, 3}, {6, 6, 3}, {6, 6, 3}, {1, 6, 1}, {3}}
```

s3 obtiene la frecuencia de cada uno de los grados de los vértices de $N_v(G)$.

```
s3 = frequencies[#] & /@ s2
{{{1, 3}}, {{2, 3}, {1, 6}}, {{1, 3}, {2, 6}}, {{5, 3},
{1, 6}}, {{1, 3}, {2, 6}}, {{5, 3}, {1, 6}}, {{1, 3},
{2, 6}}, {{1, 3}, {2, 6}}, {{2, 1}, {1, 6}}, {{1, 3}}}
```

s4 contiene los resultados de *s3* en forma de lista.

```
s4 = (Plus @@ (ReplacePart [d, # [[1]], #[[2]] ]& /@ #)) & /@ s3
{{0, 0, 1, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 2, 0, 0, 1, 0, 0, 0, 0},
 {0, 0, 1, 0, 0, 2, 0, 0, 0, 0}, {0, 0, 5, 0, 0, 1, 0, 0, 0, 0},
 {0, 0, 1, 0, 0, 2, 0, 0, 0, 0}, {0, 0, 5, 0, 0, 1, 0, 0, 0, 0},
 {0, 0, 1, 0, 0, 2, 0, 0, 0, 0}, {0, 0, 1, 0, 0, 2, 0, 0, 0, 0},
 {2, 0, 0, 0, 0, 1, 0, 0, 0, 0}, {0, 0, 1, 0, 0, 0, 0, 0, 0, 0}}
```

Ahora se hace la unión de los elementos de *s4*, se ordenan lexicográficamente y a cada uno de ellos se les asigna un número de acuerdo a su orden. A los elementos de *s4*, se les sustituye por su correspondiente número dado en el ordenamiento. Este último resultado es el que se obtiene al aplicar *D2* a cada uno de los grafos.

```
D2[G1]
{1, 3, 2, 4, 2, 4, 2, 2, 5, 1}

D2[G2]
{3, 2, 2, 2, 4, 4, 2, 5, 1, 1}

D2[G3]
{2, 3, 1, 2}
```

Ahora se hace la prueba de isomorfismo, haciendo la comparación de sus invariantes, en caso de ser isomorfos se obtienen sus isomorfismos. El código que hace esta detección es el siguiente

```
isomorphisms[G1_, G2_] :=
Module[{d1G1 = D1[G1], d1G2 = D1[G2], d2G1, d2G2, A, B, bij,
bijections, ans = {}},

bij[{},z_]:=(If[(#[[z]] & /@ G2)[[z]]==G1, AppendTo[ans,z]]);
```

```
bij[{x_,y___},z_]:= Map[bij[{y},Append[z,#]]&,Complement[x,z]];

bijections[{A_, B_}] := bij[Flatten[Position[B, #]] & /@ A, {}];

If[Union[d1G1] ≠ Union[d1G2], {},
 d2G1 = D2[G1];
 d2G2 = D2[G2];
 If[Union[d2G1] ≠ Union[d2G2], {},
  A = Thread[List[d1G1, d2G1]];
  B = Thread[List[d1G2, d2G2]];
  If[Union[A] ≠ Union[B], {}, bijections[{A, B}]]]];
 ans]
```

Aplicando esta prueba a los grafos del ejemplo se obtienen los siguientes resultados

```
isomorphisms[G1, G2]
{{9, 1, 4, 5, 7, 6, 2, 3, 8, 10}, {9, 1, 4, 5, 7, 6, 3, 2, 8, 10},
{9, 1, 7, 5, 4, 6, 2, 3, 8, 10}, {9, 1, 7, 5, 4, 6, 3, 2, 8, 10},
{10, 1, 4, 5, 7, 6, 2, 3, 8, 9}, {10, 1, 4, 5, 7, 6, 3, 2, 8, 9},
{10, 1, 7, 5, 4, 6, 2, 3, 8, 9}, {10, 1, 7, 5, 4, 6, 3, 2, 8, 9}}
```

Lo cual indica que *G1* es Isomorfo a *G2* y los que aparecen son todos sus posibles isomorfismos.

```
isomorphisms[G1, G3]
{ }
```

Es obvio que *G3* no será isomorfo a ninguno de los dos por el simple hecho de que el invariante "numero de vértices no coincide". Utilizando el mismo método aplicado a *G3* se obtiene

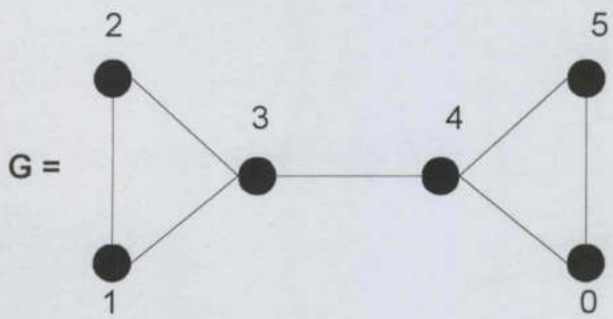
```
isomosphisms[G3, G3]
{{1, 2, 3, 4}, {4, 2, 3, 1}
```

El resultado obtenido son los automorfismos de *G3*, es decir, los isomorfismos de *G3* con *G3*.

3.4. Certificado de Grafos

El método común para obtener el certificado de un grafo es a través de su matriz de adyacencia. Cada orden posible de las permutaciones de los vértices que forman un árbol determina una matriz de adyacencia particular.

Una vez definida la matriz de adyacencia, las $\frac{n(n-1)}{2}$ entradas sobre la diagonal principal de la matriz forman un número binario de tamaño $\frac{n(n-1)}{2}$ llamado $Num_{\pi}(G)$. Este número se forma listando cada una de esas entradas columna por columna como se muestra en la figura 3.12



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

Figura 3.12 *Grafo* G_1

$Num_{\pi}(G_1) = \{001011100110001\}$

Es claramente observable que si n es grande $Num_{\pi}(G)$ será un número también muy grande y difícil de representar en la computadora, por lo tanto $Num_{\pi}(G)$ es particionado en un arreglo de varios enteros o se puede representar como una cadena de caracteres.

El más pequeño número que puede ser obtenido a través de los diferentes posibles ordenamientos define un certificado de un grafo. Este número pequeño se obtiene por

$$Cert(G) = min\{ Num_{\pi}(G) : \pi \in Sym(V)\}$$

Recordando que $Sym(V)$ es el grupo de simetría de los vértices de G y el orden se hace lexicográficamente.

Desafortunadamente este certificado es muy difícil de calcular. Tan solo para obtener *Sym* (8), es decir las 40,320 posibles permutaciones, se necesitaron 2.69 segundos. Este problema es semejante a resolver el problema del *Clique* Máximo de un grafo, el cual se sabe que es un problema NP-Completo.

Observamos que la dificultad de este problema se centra en conocer el orden de los vértices de un grafo que producirá un certificado mínimo. Para solucionar esto, se hará uso de una técnica que nos permita obtener un orden mínimo de los vértices de un grafo generando particiones del conjunto de vértices, posteriormente formar la matriz de adyacencia y obtener $Num_{\pi}(G)$.

Una partición B es una *partición discreta* si $|B[j]| = 1$ para todo j, $0 \leq j < k$. B es una *partición unitaria* si $|B| = 1$

Una partición B es *partición equitativa* con respecto al grafo $G = (V, E)$ si para todo i y j

$$|N_G(u) \cap B[j]| = |N_G(v) \cap B[j]|$$

para todo $u, v \in B[i]$, donde

$$N_G(u) = \{x \in V: \{u, v\} \in E\}$$

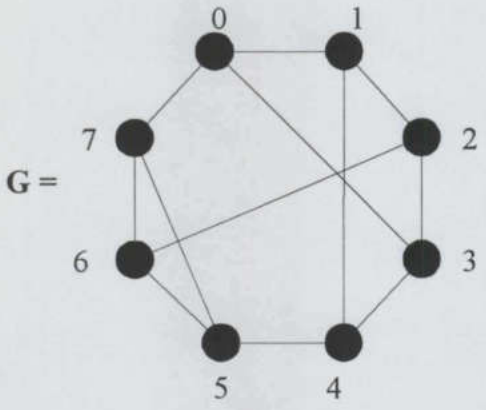
es el conjunto de vecinos de u en el grafo G

En una partición discreta, cada una de las particiones son de tamaño 1. En una partición unitaria, solamente existe una partición. En una partición equitativa, u y v tienen el mismo conjunto de elementos vecinos, de manera que escoger uno u otro es indistinto.

Supóngase que se tiene una partición B equitativa de tamaño k con respecto al grafo G. Se define, entonces, una matriz M_B de orden $k \times k$ y sus elementos estarían formados por

$$M_B[i, j] = |N_G(v) \cap B[j]|, \text{ donde } v \in B[i].$$

Por ser una partición equitativa $M_B[i, j]$ no depende de la elección de v en $B[i]$. Ahora se obtiene Num_B de la misma manera como se obtiene $Num_{\pi}(G)$ y se obtiene así el certificado para el grafo G. Este método se aplica a la figura 3.13.



$G_2 = (V, E)$

$V = \{0, 1, 2, 3, 4, 5, 6, 7\}$

$$A_{G_2} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Figura 3.13 *Grafo G2*

$Num_{\pi}(G_2)$ = [10110101010000100100110000011]

B es la partición equitativa de G_2.

$B = [\{0\}, \{2, 4\}, \{5, 6\}, \{7\}, \{1,3\}]$ y

$M_B [1, 1] = |N_G(0) \cap \{0\}| = |\{1, 3, 7\} \cap \{0\}| = 0$

$M_B [1, 2] = |N_G(0) \cap \{2, 4\}| = |\{1, 3, 7\} \cap \{2, 4\}| = 0$

$M_B [1, 3] = |N_G(0) \cap \{5, 6\}| = |\{1, 3, 7\} \cap \{5, 6\}| = 0$

$M_B [1, 4] = |N_G(0) \cap \{7\}| = |\{1, 3, 7\} \cap \{7\}| = 1$

$M_B [1, 5] = |N_G(0) \cap \{1, 3\}| = |\{1, 3, 7\} \cap \{1, 3\}| = 2$

...

por lo tanto $M_B = \begin{pmatrix} 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 2 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 \end{pmatrix}$ y Num_B = [0011012200]

De esta manera se obtiene un certificado mucho más pequeño. La dificultad en este método consiste en obtener la partición B. A continuación se explica este método.

Una partición B es un *refinamiento* de la partición ordenada A si

1. cada bloque $B[i]$ de B esta contenido en algún bloque $A[j]$ de A
2. si $u \in A[i_1]$ y $v \in A[j_1]$ con $i_1 \leq j_1$, entonces $u \in B[i_2]$ y $v \in B[j_2]$ con $i_2 \leq j_2$.

Por ejemplo, $B = [\{0\}, \{2, 4\}, \{5, 6\}, \{7\}, \{1,3\}]$ es un refinamiento de $A = [\{0\}, \{1, 2, 3, 4, 5, 6, 7\}]$, pero $B' = [\{2, 4\}, \{5, 6\}, \{0\}, \{7\}, \{1, 3\}]$ no es un refinamiento de A, porque los bloques no están ordenados con respecto a A.

Si A es una partición ordenada y T es algún bloque de T, entonces una función D_T: $V \rightarrow \{0, 1, ...n\text{-}1\}$ se define como $D_T(v) = |N_G(v) \cap T|$

Esta función D_T ayudará a refinar la partición ordenada A en una partición ordenada B. El método a seguir es el siguiente.

A es la partición original

B es la partición que se obtendrá a partir de A

S será una lista que contendrá los bloques de B

T será el último bloque tomado de S

L será una lista de los valores de D_T

1. Hacer $B = A$
2. S = bloques de B
3. Mientras S $\neq \varnothing$ hacer

   3.1 T = último bloque de S. Remover T de S

   3.2 Para cada bloque B[i] en B hacer

      3.2.1 Obtener D_T(v) para cada v $\in B$[i]

      3.2.2 L[h] = {v $\in$ B[i]: DT(v) = h}

Si hay más de un bloque no vacío en L

   3.2.3 Reemplazar el bloque B[i] con los bloques en L en el orden de los índices de h, h = 0, 1, ... , n-1

   3.2.4 Insertar todos los bloques no vacíos en L, en orden inverso, al final de la lista S

Ahora se aplica este método al grafo de la figura 3.13

$A = [\{0\}, \{1, 2, 3, 4, 5, 6, 7\}]$

$B = [\{0\}, \{1, 2, 3, 4, 5, 6, 7\}]$

$S = [\{1, 2, 3, 4, 5, 6, 7\}, \{0\}]$

Mientras $S \neq \varnothing$ hacer

$T = \{0\}$ y $S = [\{1, 2, 3, 4, 5, 6, 7\}]$

Para $B[1] = \{0\}$

$D_T(0) = |\{1, 3, 7\} \cap \{0\}| = 0$

$L[0] = \{0\}$

Pasos 8 y 9 no se realizan ya que no hay más de dos bloques no vacíos

Para $B[2] = \{1, 2, 3, 4, 5, 6, 7\}$

$D_T(1) = |\{2, 4, 0\} \cap \{0\}| = 1$ $\quad$ $D_T(5) = |\{4, 6, 7\} \cap \{0\}| = 0$

$D_T(2) = |\{1, 3, 6\} \cap \{0\}| = 0$ $\quad$ $D_T(6) = |\{2, 5, 7\} \cap \{0\}| = 0$

$D_T(3) = |\{2, 4, 0\} \cap \{0\}| = 1$ $\quad$ $D_T(7) = |\{0, 5, 6\} \cap \{0\}| = 1$

$D_T(4) = |\{1, 3, 5\} \cap \{0\}| = 0$

$L[0] = \{2, 4, 5, 6\}$ $\quad$ $L[1] = \{1, 3, 7\}$

Reemplazar $B[2] = \{1, 2, 3, 4, 5, 6, 7\}$ por $B[2] = [\{2, 4, 5, 6\}, \{1, 3, 7\}]$

$S = [\{1, 2, 3, 4, 5, 6, 7\}, \{1, 3, 7\}, \{2, 4, 5, 6\}]$

Como $S \neq \varnothing$

$T = \{2, 4, 5, 6\}$, $S = [\{1, 2, 3, 4, 5, 6, 7\}, \{1, 3, 7\}]$ y $B = [\{0\}, \{2, 4, 5, 6\}, \{1, 3, 7\}]$

Para $B[1] = \{0\}$

$D_T(0) = |\{1, 3, 7\} \cap \{2, 4, 5, 6\}| = 0$

$L[0] = \{0\}$

Pasos 8 y 9 no se realizan ya que no hay más de dos bloques no vacíos

Para $B[2] = \{2, 4, 5, 6\}$

$D_T(2) = |\{1, 3, 6\} \cap \{2, 4, 5, 6\}| = 1$

$D_T(4) = |\{1, 3, 5\} \cap \{2, 4, 5, 6\}| = 1$

$D_T(5) = |\{4, 6, 7\} \cap \{2, 4, 5, 6\}| = 2$

$D_T(6) = |\{2, 5, 7\} \cap \{2, 4, 5, 6\}| = 2$

$L[1] = \{2, 4\}$ $L[2] = \{5, 6\}$

Reemplazar $B[2] = \{2, 4, 5, 6\}$ por $B[2] = \{2, 4\}, \{5, 6\}$

$S = [\{1, 2, 3, 4, 5, 6, 7\}, \{1, 3, 7\}, \{5, 6\}, \{2, 4\}]$

Para $B[3] = \{1, 3, 7\}$

$D_T(1) = |\{2, 4, 0\} \cap \{2, 4, 5, 6\}| = 0$

$D_T(3) = |\{2, 4, 0\} \cap \{2, 4, 5, 6\}| = 0$

$D_T(7) = |\{0, 5, 6\} \cap \{2, 4, 5, 6\}| = 0$

$L[0] = \{1, 3, 7\}$

Pasos 8 y 9 no se realizan ya que no hay más de dos bloques no vacíos

Como $S \neq \varnothing$

$T = \{2, 4\}$, $S = [\{1, 2, 3, 4, 5, 6, 7\}, \{1, 3, 7\}, \{5, 6\}]$ y $B = [\{0\}, \{2, 4\}, \{5, 6\}, \{1, 3, 7\}]$

Para $B[1] = \{0\}$

$D_T(0) = |\{1, 3, 7\} \cap \{2, 4\}| = 0$

$L[0] = \{0\}$

Pasos 8 y 9 no se realizan ya que no hay más de dos bloques no vacíos

Para $B[2] = \{2, 4\}$

$D_T(2) = |\{1, 3, 6\} \cap \{2, 4\}| = 0$

$D_T(4) = |\{1, 3, 5\} \cap \{2, 4\}| = 0$

$L[0] = \{2, 4\}$

Pasos 8 y 9 no se realizan ya que no hay más de dos bloques no vacíos

Para $B[3] = \{5, 6\}$

$D_T(5) = |\{4, 6, 7\} \cap \{2, 4\}| = 1$

$D_T(6) = |\{2, 5, 7\} \cap \{2, 4\}| = 1$

$L[1] = \{5, 6\}$

Pasos 8 y 9 no se realizan ya que no hay más de dos bloques no vacíos

Para $B[4] = \{1, 3, 7\}$

$D_T(1) = |\{2, 4, 0\} \cap \{2, 4\}| = 2$

$D_T(3) = |\{2, 4, 0\} \cap \{2, 4\}| = 2$

$D_T(7) = |\{0, 5, 6\} \cap \{2, 4\}| = 0$

$L[0] = \{7\}$ $L[2] = \{1, 3\}$

Reemplazar $B[4] = \{1, 3, 7\}$ por $B[2] = \{7\}, \{1, 3\}$

$S = [\{1, 2, 3, 4, 5, 6, 7\}, \{1, 3, 7\}, \{5, 6\}, \{1, 3\}, \{7\}]$

A partir de este momento, la remoción de cada elemento último de S no produce cambio alguno en las particiones ya obtenidas, por ello se deja el procedimiento hasta este paso anterior.

Entonces $B = [\{0\}, \{2, 4\}, \{5, 6\}, \{7\}, \{1, 3\}]$, que es la partición con la que obtuvo el certificado del grafo. Se observa que esta partición no depende de la forma como se dibuje el grafo, finalmente cada vértice conservará sus vecinos y producirá la misma partición, con esto cumple la propiedad de ser un invariante y un certificado.

Si A empieza con la partición original $A = [\{0, 1, 2, 3, 4, 5, 6, 7\}]$ el primer refinamiento obtenido sería $B = [\{1, 3, 7\}, \{0, 2, 4, 5, 6\}]$ lo cual no cumple con las condiciones de ser una partición refinada porque no hay un orden con respecto a A. Este último método obtiene la primera partición basado en el grado de sus vértices para lo cual no se necesitaría un procedimiento tan largo como el que se explicó. Esta última variación se analizará al aplicar el método al grafo de la figura 3.14

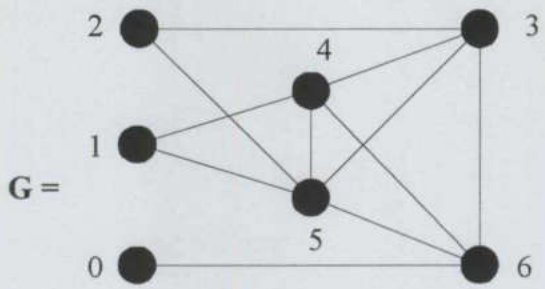


Figura 3.14 Grafo G_3

Aplicando el procedimiento visto anteriormente al grafo G_3 de la figura 3.14 se obtiene la partición

$B = [\{0\}, \{1, 2\}, \{3, 4\}, \{6\}, \{5\}]$

Una vez obtenida la partición equitativa y refinada se requiere que sea una partición discreta, es decir que $|B[i]| = 1$. Con esto se obtiene el orden de los vértices para formar la matriz de adyacencia y posteriormente obtener el certificado, como se explicó al principio de esta sección.

Para obtener la partición discreta se siguen los siguientes pasos sencillos.
Se inicia con una partición equitativa P de vértices V de un grafo, un bloque $P[i]$ de tamaño $m > 1$ debe ser dividida en dos bloques, el primero de tamaño 1 y el segundo de tamaño m - 1. Después de hacer la división se aplica el algoritmo para refinar particiones y se repite este algoritmo hasta que se obtenga una partición discreta.

Para la figura 3.13 se tiene $B = [\{0\}, \{2, 4\}, \{5, 6\}, \{7\}, \{1, 3\}]$ donde $P[2] = \{2, 4\}$, dividimos esta partición en $\{2\}$, $\{4\}$ y aplicando el algoritmo de refinamiento se obtiene la partición $B = [\{0\}, \{2\}, \{4\}, \{5\}, \{6\}, \{7\}, \{1, 3\}]$. Ahora se tiene $P[7] = \{1, 3\}$, dividimos esta partición en $\{1\}$ y $\{3\}$ y aplicando el algoritmo de refinamiento se obtiene la partición $B = [\{0\}, \{2\}, \{4\}, \{5\}, \{6\}, \{7\}, \{1\}, \{3\}]$, la cual es ya una partición discreta.

El orden de los vértices de la partición discreta indica orden en que deben ser acomodados los vértices para formar la matriz de adyacencia y así obtener el certificado mínimo del grafo.

En el grafo de la figura 3.13 el orden es $\pi[0, 2, 4, 5, 6, 7, 1, 3]$

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

$Num_{\pi 2}(G_2)$ = [00000101011001111100011100000] y $Num_{\pi}(G_2)$ = [10110101010000100100110000011] se ve claramente que $Num_{\pi 2}(G_2)$ es menor en orden lexicográfico.

Este resultado es producto de dividir $P[2] = \{2, 4\}$ en $\{2\}$ y $\{4\}$, pero que pasaría si se dividiera en $\{4\}$ y $\{2\}$, entonces se tendría $B = [\{0\}, \{4\}, \{2\}, \{5\}, \{6\}, \{7\}, \{1, 3\}]$, lo cual hubiera generado el mismo $Num_{\pi 2}(G_2)$, lo mismo hubiera pasado con $\{1, 3\}$. Esto es justificable porque las particiones originales son particiones equitativas y en ellas es indistinto tomar cualquier elemento que pertenece a dicha partición.

Sin embargo, que hubiera pasado con el grafo de la figura 3.14, donde
$B = [\{0\}, \{1, 2\}, \{3, 4\}, \{6\}, \{5\}]$ se obtiene el orden $\pi[0, 1, 2, 3, 4, 6, 5]$

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

$Num_{\pi 2}(G_3)$ = [0000010101100110111111] y $Num_{\pi}(G_3)$ = [0000010101011111100111] en este ejemplo $Num_{\pi}(G_3)$ es menor a $Num_{\pi 2}(G_3)$, entonces no hubiera sido necesario aplicar el método de refinamiento de particiones para obtener un nuevo orden de los vértices.

Sería más sencillo ordenar los vértices de acuerdo a su grado, el orden hubiera sido [0, 1, 2, 3, 4, 6, 5]. Por lo tanto empezar con la partición unitaria no es eficiente para el objetivo que se persigue.

CAPÍTULO 4
APLICACIÓN DEL USO DE CERTIFICADOS A ÁRBOLES

4.1. Estructura computacional de árboles

La estructura de árbol constituye una estructura de grafo de gran versatilidad que ha encontrado una multitud de aplicaciones y se considera entre las estructuras de datos más importantes en la ciencia de la computación. Dentro de la especie árbol se encuentran los llamados árboles con raíz, los cuales pueden definirse por la gramática:

Arbol ={} | {raíz, {árbol}}

Aunque es posible etiquetar los nodos de un árbol con datos de cualquier tipo, aun una mezcla de ellos, por simplicidad utilizaremos enteros positivos exclusivamente en esta sección puesto que siempre es posible diseñar una correspondencia biyectiva entre el tipo de dato que se tenga y esos enteros.

Este capítulo provee del diseño de varias operaciones fundamentales sobre árboles encaminadas hacia su enumeración y generación. Las últimas secciones especializarán este desarrollo a árboles binarios.

En el desarrollo subsecuente denotaremos por *t* un árbol genérico, *r* una raíz genérica y *sT* una lista genérica de subárboles.

La función *treeQ* valida una estructura dada que supuestamente representa un árbol.

```
treeQ[{leaf_, {}}] := True
treeQ[{_, {sT__}}] := And @@ treeQ /@ {sT}
treeQ[_] := False
treeQ[1]
False

treeQ[{1,{}}]
True
```

El siguiente árbol, denominado *smallTree*, es el primero de tres ejemplos que utilizaremos para probar todo nuestro desarrollo.

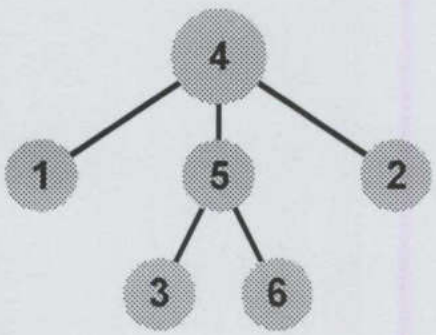


***Figura 4.1** SmallTree*

Su notación extendida es:

```
smallTree = {4, {{1, {}}, {5, {{3, {}}, {6, {}}}}, {2, {}}}};
treeQ[smallTree]
True
```

4.2. Notación Abreviada

La notación extendida utilizada para describir árboles resulta inadecuada. Con el fin de clarificar los resultados y acelerar la entrada de datos en algoritmos de procesamiento de árboles, diseñamos las funciones *shortNotation* y *longNotation*, las cuales actuaran respectivamente como post y preprocesadores de aquellos algoritmos donde la notación extendida es ventajosa y que supondremos será usada internamente de manera exclusiva.

En el ejemplo que sigue a la definición de la función *shortNotation* ilustraremos la reducción de la notación extendida en el caso de *smallTree* escribiendo las hojas simplemente como números y eliminando paréntesis externos y sustituyendo los restantes por paréntesis, formando de esta manera una estructura de cadena.

```
shortNotation[{leaf_, {}}] := ToString[leaf]
shortNotation[{r_, {sT__}}] :=
 ToString[r] <>
  StringReplace[
   ToString[shortNotation /@ {sT}],{"{"->"(","}"->")"," " ->""}]

smallTree
shortNotation[%]
{4, {{1, {}}, {5, {{3, {}}, {6, {}}}}, {2, {}}}}
4(1,5(3,6),2)"}
```

Una implementación directa de las transformaciones que tratan con la conversión de la notación extendida, que denominaremos *Notación Larga*, de un árbol, involucra el apareo de varios patrones y que el mecanismo interno de *Mathematica* facilita.

```
longNotation[t_String] := Module[{u},
    u = Map[If[DigitQ[#], ToExpression[#], #] &, Characters[t]];
    u = u //. {a___,i_Integer, j_Integer, b___} → {a, 10 i + j, b};
    ToExpression[sN[ToString /@ u]]]

sN[{r_,"(",sT__,")"}] := Module[{c = {sT}, i, j = 1, u, sons = {}},
    Do[
      If[(c[[i]] == ",") && balanced[u = Take[c, {j, i - 1}]],
         AppendTo[sons, u]; j = i + 1], {i, Length[c]}];
```

```
    AppendTo[sons, Take[c, {j, Length[c]}]];
    {r, sN /@ sons}]
sN[{leaf_}] := {leaf, {}}

balanced[s_] := Module[{i = 0},
    Map[
      If[i < 0, Return[False],

          If[(# == "(") || (# == "{"), i++,
            If[(# == ")") || (# == "}"), i--]]] &, s];
      (i == 0)]
```

Al ilustrar el uso de la función *longNotation*, introduciremos dos nuevos árboles de prueba *bigTree1* y *bigTree2*; Su representación gráfica se generara al final de esta sección. Tomaremos también la oportunidad de verificar el uso de las notaciones corta y larga sobre ella.

```
bigTree1 =
longNotation["5(10(8),3(11,12(2(13,14(15,16,17,18))),7),9(1,6),4)"]
shortNotation[%]
{5, {{10, {{8, {}}}}, {3, {{11, {}}, {12, {{2, {{13, {}},
{14, {{15, {}}, {16, {}}, {17, {}}, {18, {}}}}}}}}},
{7, {}}}}, {9, {{1, {}}, {6, {}}}}, {4, {}}}}

5(10(8), 3(11,12(2(13,14(15,16,17,18))),7),9(1,6),4)

bigTree2 =
  longNotation[
"1(5(11,10(18),12),4(9),2(6(13(19,20,21))),3(7(14,15),8(16,17)))"]
shortNotation[%]
{1, {{5, {{11, {}}, {10, {{18, {}}}}, {12, {}}}}, {4, {{9, {}}}},
{2, {{6, {{13, {{19, {}}, {20, {}}, {21, {}}}}}}}},
3, {{7, {{14, {}}, {15, {}}}}, {8, {{16, {}}, {17, {}}}}}}}

1(5(11,10(18),12),4(9),2(6(13(19,20,21))),3(7(14,15),8(16,17)))
```

4.3. Representando Árboles

4.3.1. Pares Ordenados

Una forma de representar árboles es a través del uso de pares ordenados, esto es, utilizando una relación sobre el conjunto de nodos. Las siguientes funciones proveen la codificación y decodificación necesarias a partir de esta representación. Los pares son provistos tales que (i, j) significa que i es el padre de j

```
toPairs[{r_, {sT__}}] :=
  Partition[Flatten[{Map[{r,#}&,First/@ sT}], toPairs /@ {sT}}], 2]
toPairs[_] := {}

toPairs[bigTree1]
{{5, 10}, {5, 3}, {5, 9}, {5, 4}, {10, 8},
{3, 11}, {3, 12}, {3, 7}, {12, 2}, {2, 13}, {2, 14},
{14, 15}, {14, 16}, {14, 17}, {14, 18}, {9, 1}, {9, 6}}

toPairs[bigTree2]
{{1, 5}, {1, 4}, {1, 2}, {1, 3}, {5, 11}, {5, 10}, {5, 12},
{10, 18}, {4, 9}, {2, 6}, {6, 13}, {13, 19}, {13, 20}, 13, 21}, {3,
7}, {3, 8}, {7, 14}, {7, 15}, {8, 16}, {8, 17}}
```

con el fin de diseñar su función inversa que toma un conjunto de pares y se calcula el árbol correspondiente, necesitamos considerar el problema de calcular la profundidad de un nodo dado. Este cálculo es efectuado por la función *dep* siguiente:

```
dep[r_, {r_, _}] := 0
dep[r_, {_, sT_}] := 1 + Max[dep[r, #] & /@ sT]
```

Por ejemplo, para obtener todas las profundidades correspondientes a *bigTree1* procedemos como sigue:

```
shortNotation[bigTree1]
Map[{#, dep[#, bigTree1]} &, Flatten[bigTree1]]
5(10(8), 3(11,12(2(13,14(15,16,17,18))),7),9(1,6),4)
{{5, 0}, {10, 1}, {8, 2}, {3, 1}, {11, 2}, {12, 2},
```

```
{2, 3}, {13, 4}, {14, 4}, {15, 5}, {16, 5}, {17, 5},
{18, 5}, {7, 2}, {9, 1}, {1, 2}, {6, 2}, {4, 1}}
```

En éste ejemplo se puede observar que el nodo 14 es de profundidad 4.

Un poco más difícil es deducir la profundidad de un nodo dado *n* cuando se dan los pares que forman el árbol:

```
dep2[n_, {a___, {b_, n_}, d___}] := 1 + dep2[b, {a, d}]
dep2[_, _] := 0

p = toPairs[bigTree1];
Map[{#, dep2[#, p]} &, Flatten[bigTree1]]
{{5, 0}, {10, 1}, {8, 2}, {3, 1}, {11, 2}, {12, 2},
{2, 3}, {13, 4}, {14, 4}, {15, 5}, {16, 5}, {17, 5},
{18, 5}, {7, 2}, {9, 1}, {1, 2}, {6, 2}, {4, 1}}
```

Donde podemos observar que la raíz es el 5 y sus hijos son los números 10, 3, 9 y 4.

```
Map[{#, dep2[#, toPairs[bigTree2]]} &, Flatten[bigTree2]]
{{1, 0}, {5, 1}, {11, 2}, {10, 2}, {18, 3}, {12, 2}, {4, 1},
{9, 2}, {2, 1}, {6, 2}, {13, 3}, {19, 4}, {20, 4}, {21, 4},
{3, 1}, {7, 2}, {14, 3}, {15, 3}, {8, 2}, {16, 3}, {17, 3}}
```

Finalmente, para construir una solución del problema inverso basada en el uso de patrones, necesitamos reordenar los pares dados a profundidad de tal forma que el árbol se construya de las hojas a la raíz. Este procedimiento se denomina *goodSort*.

```
goodSort[p_] :=
   Sort[p, (dep2[First[#1], p] >= dep2[First[#2], p]) &]
goodSort[toPairs[bigTree1]]
{{14, 15}, {14, 16}, {14, 17}, {14, 18}, {2, 13},
{2, 14}, {12, 2}, {10, 8}, {3, 11}, {3, 12}, {3, 7},
{9, 1}, {9, 6}, {5, 10}, {5, 3}, {5, 9}, {5, 4}}
```

```
goodSort[toPairs[bigTree2]]
{{13, 19}, {13, 20}, {13, 21}, {10, 18}, {6, 13}, {7, 14},
{7, 15}, {8, 16}, {8, 17}, {5, 11}, {5, 10}, {5, 12}, {4, 9},
{2, 6}, {3, 7}, {3, 8}, {1, 5}, {1, 4}, {1, 2}, {1, 3}}
```

La función *fromPairs* siguiente implementa un proceso de eliminación-construcción sobre el cual el árbol se construye. Procedemos a continuación con la verificación de las funciones principales de esta sección aplicados a nuestros arboles típicos.

```
fromPairs[p_] := Module[{q = goodSort[p], t = {}, i, j},
    Map[({i, j} = #;
       t = t /. {{a___, {i, {k__}}, b___} -> {a, {i, {k, {j, {}}}}},
      b}, {a___} -> {a, {i, {{j, {}}}}}}) &, q];
    First[
      t //. {a___, {b_, c_}, d___, {e_, {f___, {b_, {}}, g___}},
h___} -> {a, d, {e, {f, {b, c}, g}}, h}]]

shortNotation[fromPairs[{{1, 2}, {3, 1}}]]
3(1(2))

shortNotation[smallTree]
shortNotation[fromPairs[toPairs[smallTree]]]
4(1,5(3,6),2)
4(1,5(3,6),2)

shortNotation[bigTree1]
shortNotation[fromPairs[toPairs[bigTree1]]]
5(10(8),3(11,12(2(13,14(15,16,17,18))),7),9(1,6),4)
5(10(8),3(11,12(2(13,14(15,16,17,18))),7),9(1,6),4)

shortNotation[bigTree2]
shortNotation[fromPairs[toPairs[bigTree2]]]
1(5(11,10(18),12),4(9),2(6(13(19,20,21))),3(7(14,15),8(16,17)))
1(5(11,10(18),12),4(9),2(6(13(19,20,21))),3(7(14,15),8(16,17)))
```

4.3.2. Matrices

La conversión de una estructura arborescente en una estructura matricial correspondiente a los pares formando la relación *padre-de* necesita ser acompañada de una referencia explícita a la raíz. Inversamente, con el fin de recuperar el árbol que dio lugar a una matriz dada es necesario elegir el nodo que constituirá la raíz del árbol requerido. La representación matricial no resulta única tampoco en el orden jerárquico de los subárboles. La función *toMatrix* construye una matriz simétrica, con ceros en la diagonal principal, correspondiente a un árbol dado y, al realizar esto, mezcla dichos pares con su pareja simétricamente opuesta. La función *fromMatrix* dispone de pares innecesarios con el fin de recobrar el árbol "levantando" la raíz dada.

```
toMatrix[{r_, sT_}] := Module[{p = toPairs[{r, sT}], m, i, j},
    m = DiagonalMatrix[Table[0, {Length[p] + 1}]];
    Map[({i, j} = #;
    M[[i, j]] =
            M[[j, i]] = 1) &, p];
    {r, m}]

filterPairs[r_, p_] := {} /; Cases[p, {r, _}] == {}
filterPairs[r_, p_] := Module[{t = Cases[p, {r, _}], c, q},
    c = Cases[p, {_, r}];
    q = Complement[p, t, c];
    Partition[Flatten[{t, Map[filterPairs[#, q] &, Last /@ t]}],
2]]

fromMatrix[{r_, m_}] := fromPairs[filterPairs[r, Position[m, 1]]]

shortNotation[smallTree]
t = toMatrix[smallTree];
{First[t], Last[t] // MatrixForm}
shortNotation[fromMatrix[t]]
ListDensityPlot[-Reverse[Last[t]], Frame -> False];
4(1,5(3,6),2)
```

$$\left\{4,\begin{pmatrix}0&0&0&1&0&0\\0&0&0&1&0&0\\0&0&0&0&1&0\\1&1&0&0&1&0\\0&0&1&1&0&1\\0&0&0&0&1&0\end{pmatrix}\right\}$$

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

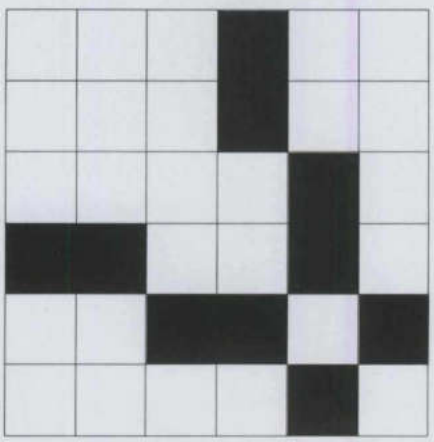


Figura 4.2 *Matriz correspondiente a smallTree*

4.4. Generación de Árboles

De la totalidad de 2^{n^2} relaciones posibles sobre un conjunto de n elementos, solamente una pequeña fracción son no-reflexivas (es decir, ningún elemento se relaciona consigo mismo) y simétrica, es decir $2^{\frac{n(n-1)}{2}}$ y de estas, solamente $\binom{\frac{n(n-1)}{2}}{n-1}$ corresponden a relaciones de tamaño n-1, una propiedad necesaria para todo árbol.

```
n = 6
```

$$2^{n^2}$$

$$2^{\frac{n(n-1)}{2}}$$

$$\text{Binomial}\left[\frac{n(n-1)}{2},n-1\right]$$

```
68719476736
32768
3003
```

De esas relaciones, es suficiente verificar su conexidad para seleccionar a todos los arboles posibles. También necesitamos filtrarlos subsecuentemente con el fin de obtener los no-isomorfos. Podemos eliminar matrices que posean al menos un renglón de ceros, disminuyendo la cota superior de su numero. Se tiene la siguiente conjetura: una condición necesaria y suficiente para que una matriz no-reflexiva-simetrica y que tenga $2(n\text{-}1)$ entradas positivas sea un árbol es que no posea un renglón nulo. Ya sea cierta o falsa, esta conjetura nos provee con una cota superior útil.

```
upperDiag[nodes_] :=
    Module[{m = nodes (nodes - 1) / 2, d,
        edges = nodes - 1},
      d = Map[IntegerDigits[#, 2, m] &, Range[2^m - 1]];
       Select[d, ((Plus\  @@ \ # == edges)\) &]]

  symMatrix[d_] := Module[{n, m, i, j, k = 1},
n = (1 + Sqrt[1 + 8 Length[d]]) / 2;
m = Table[0, {n}, {n}];
Do[
  Do[
    m[[i,j]] = m[[j,i]] = d[[k]]; k = k + 1, {j, i-1}], {i, 2, n}];
    m]

  allNonReflSym[n_] := symMatrix /@ upperDiag[n]
  someZeroRowQ[m_] := Position[(Plus  @@  #) & /@ m, 0] ≠ {}
```

Con el fin de obtener todos los candidatos a árboles no-isomorfos de, digamos, cuatro nodos, procedemos como sigue:

```
MatrixForm /@
(u = Select[allNonReflSym[4], Not[someZeroRowQ[#]] &])
```

$$\left\{\begin{pmatrix}0&0&0&1\\0&0&0&1\\0&0&0&1\\1&1&1&0\end{pmatrix},\begin{pmatrix}0&0&0&1\\0&0&1&0\\0&1&0&1\\1&0&1&0\end{pmatrix},\begin{pmatrix}0&0&0&1\\0&0&1&1\\0&1&0&0\\1&1&0&0\end{pmatrix},\begin{pmatrix}0&0&1&0\\0&0&0&1\\1&0&0&1\\0&1&1&0\end{pmatrix},\right.$$
$$\begin{pmatrix}0&0&1&1\\0&0&0&1\\1&0&0&0\\1&1&0&0\end{pmatrix},\begin{pmatrix}0&0&1&0\\0&0&1&0\\1&1&0&1\\0&0&1&0\end{pmatrix},\begin{pmatrix}0&0&1&0\\0&0&1&1\\1&1&0&0\\0&1&0&0\end{pmatrix},\begin{pmatrix}0&0&1&1\\0&0&1&0\\1&1&0&0\\1&0&0&0\end{pmatrix},$$
$$\begin{pmatrix}0&1&0&0\\1&0&0&1\\0&0&0&1\\0&1&1&0\end{pmatrix},\begin{pmatrix}0&1&0&1\\1&0&0&0\\0&0&0&1\\1&0&1&0\end{pmatrix},\begin{pmatrix}0&1&0&0\\1&0&1&0\\0&1&0&1\\0&0&1&0\end{pmatrix},\begin{pmatrix}0&1&0&0\\1&0&1&1\\0&1&0&0\\0&1&0&0\end{pmatrix},$$
$$\left.\begin{pmatrix}0&1&0&1\\1&0&1&0\\0&1&0&0\\1&0&0&0\end{pmatrix},\begin{pmatrix}0&1&1&0\\1&0&0&0\\1&0&0&1\\0&0&1&0\end{pmatrix},\begin{pmatrix}0&1&1&0\\1&0&0&1\\1&0&0&0\\0&1&0&0\end{pmatrix},\begin{pmatrix}0&1&1&1\\1&0&0&0\\1&0&0&0\\1&0&0&0\end{pmatrix}\right\}$$

```
v = (fromMatrix[{1, #}] & /@ u);
shortNotation /@ v
{1(4(2,3)), 1(4(3(2))), 1(4(2(3))), 1(3(4(2))), 1(3,4(2)),
1(3(2,4)), 1(3(2(4))), 1(3(2),4), 1(2(4(3))), 1(2,4(3)),
1(2(3(4))), 1(2(3,4)), 1(2(3),4), 1(2,3(4)), 1(2(4),3), 1(2,3,4)}
```

Podemos extraer de estos últimos aquellos candidatos finales a árboles no-isomorfos ignorando sus etiquetas:

```
shortNotation /@ Union[Map[(# /. _Integer -> 0) &, v]]
{"0(0(0(0))), 0(0(0,0)), 0(0,0(0)), 0(0(0),0), 0(0,0,0)}
```

Entonces, existen exactamente cinco árboles con cuatro vértices. Sin embargo, este procedimiento es de complejidad exponencial. En la siguiente sección se aplicará una estrategia más rápida encaminada a la enumeración de árboles no-isomorfos.

4.5. Características Estructurales

En esta sección proveemos de un número de funciones que efectúan tareas útiles en árboles, específicamente el cálculo del número de nodos, hojas, altura, ancho, reetiquetado, subárboles y conjunto de grados.

```
numNodes[t_] := Length[Flatten[t]]
numNodes[smallTree]
leaves[{r_, {}}] := {r}
leaves[{_, {sT__}}] := Flatten[leaves /@ {sT}]
leaves[smallTree]
leaves[bigTree1]
leaves[bigTree2]
height[{_, {}}] := 0
height[{_, sT_}] := 1 + Max[height /@ sT]
height[smallTree]
height[bigTree1]
height[bigTree2]
width[{_, {}}] := 1
width[{_, sT_}] := Plus @@ width /@ sT
width[smallTree]
width[bigTree1]
width[bigTree2]
rename[t_,newLabels_] :=
   t /. Thread[(#1->#2&)[Flatten[t], newLabels]]
6
{1, 3, 6, 2}
{8, 11, 13, 15, 16, 17, 18, 7, 1, 6, 4}
{11, 18, 12, 9, 19, 20, 21, 14, 15, 16, 17}
2
5
4
4
11
11

shortNotation[rename[smallTree, {10, 20, 17, 3, 0, 1}]]
10(20,17(3,0),1)
```

```
degreesTree[t_] := Module[{p = toPairs[t], d, i, j},
      d = Table[0, {numNodes[t]}];
      Map[({i, j} = #;
                  d[[i]] = d[[i]] + 1;
                  d[[j]] = d[[j]] + 1) &, p];
      d]

degreesTree[bigTree1]
{1, 3, 4, 1, 4, 1, 1, 1, 3, 2, 1, 2, 1, 5, 1, 1, 1, 1}

subTree[t_, r_] := t /; First[t] == r
subTree[{_, {}}, _] := {}
subTree[{_, sT_}, r_] := Flatten[Map[subTree[#, r] &, sT], 1]

shortNotation[bigTree1]
Table[shortNotation[subTree[bigTree1, i]], {i, 18}] // TableForm
5(10(8),3(11,12(2(13,14(15,16,17,18))),7),9(1,6),4)
1
2 (13,14(15,16,17,18))
3 (11,12(2(13,14(15,16,17,18))),7),
4
5 (10(8),3(11,12(2(13,14(15,16,17,18))),7),9(1,6),4)\\>},
6
7
8
9 (1,6)
10 (8)
11
12 (2(13,14(15,16,17,18)))
13
14 (15,16,17,18)
15
15
17
18
```

Finalmente, tomando la idea del *pumping lemma* para gramáticas de contexto libre, diseñamos la función *pump* que, dado un árbol t, nodos i y j y un conjunto de etiquetas,

"bombea" (substituye) el subárbol con raíz *i* con el subárbol con raíz *j* en el cual las etiquetas se renombran de acuerdo a este proceso. Esta resulta una forma útil de "crecer" árboles gigantes experimentales.

```
pump[t_, i_, j_, newLabels_] :=
  t /. subTree[t, i] -> rename[subTree[t, j], newLabels]
shortNotation[smallTree]
shortNotation[pump[smallTree, 1, 5, {1, 7, 8}]]
shortNotation[pump[smallTree, 6, 5, {6, 7, 8}]]
4(1,5(3,6),2)
4(1(7,8),5(3,6),2)
4(1,5(3,6(7,8)),2)
```

4.6. El Mínimo Común Ancestro

El mínimo común ancestro (*MCA*) de un par de nodos dados *i, j* es aquel subárbol de un árbol dado que tiene por raíz el nodo común más cercano a ellos. En otras palabras, *MCA*[*i, j*] es el menor subárbol que contiene a *i* y a *j*.

Comenzamos diseñando una versión alternativa no-recursiva de la función previa *subTree*, la cual selecciona un subárbol de un árbol dado que posee una raíz especifica.

```
subTree[t_, r_] := Module[{p = First[Position[t, r]], g = t},
      ((g = g[[#]]) &) /@ Drop[p, -1];
      g]

Table[shortNotation[subTree[smallTree, i]], {i, 6}]
{1, 2, 3, 4(1,5(3,6),2), 5(3,6), 6}
```

utilizando la función anterior construimos el procedimiento requerido:

```
MCA[t_, i_, j_] := Module[{p = toPairs[t], pI, pJ, v = 1},
      pI = FixedPointList
p /. {{___, {k_, #}, ___} -> k, {___} -> 0} &, i];
   pJ = FixedPointList[p /. {{___,{k_,#},___} -> k,{___}->0} &, j];
    While[!(MemberQ[pJ, pI[[v]]]), v++];
      subTree[t, pI[[v]]]]
```

```
shortNotation[MCA[bigTree1, 7, 1]]
5(10(8),3(11,12(2(13,14(15,16,17,18))),7),9(1,6),4)

shortNotation[MCA[bigTree2, 14, 17]]
3(7(14,15),8(16,17))
```

Uno de las aplicaciones de *MCA* es en el cálculo de la distancia entre dos nodos *i* y *j* en un árbol dado:

```
Clear[distance]
distance[t_, i_, j_] := Module[{q = MCA[t, i, j], r, p, pI, pJ},
      r = First[q];
      p = toPairs[q];
      pI = FixedPointList[
        p /. {{___,{r,#},___} -> r,{___,{k_,#},___} -> k, {___} ->
                r} &, i];
       pJ = FixedPointList[
        p /. {{___, {r, #}, ___} -> r, {___, {k_, #}, ___} -> k,
{___} -> r} &, j]; Length[pI] + Length[pJ] - 4]

distance[bigTree2, 19, 17]
7
```

Calculemos la matriz de distancias correspondiente a *smallTree*:

```
t = smallTree;
ListDensityPlot[-Reverse[Last[toMatrix[t]]], Frame -> False];
n = Length[Flatten[t]];
d = Table[0, {n}, {n}];
Do[
D[[i,j]] = d[[j,i]] = distance[t,i,j], {i,n-1}, {j,i+1,n}];
MatrixForm[d]
ListDensityPlot[-Reverse[d], Frame -> False];
```

$$\begin{pmatrix} 0 & 2 & 3 & 1 & 2 & 3 \\ 2 & 0 & 3 & 1 & 2 & 3 \\ 3 & 3 & 0 & 2 & 1 & 2 \\ 1 & 1 & 2 & 0 & 1 & 2 \\ 2 & 2 & 1 & 1 & 0 & 1 \\ 3 & 3 & 2 & 2 & 1 & 0 \end{pmatrix}$$

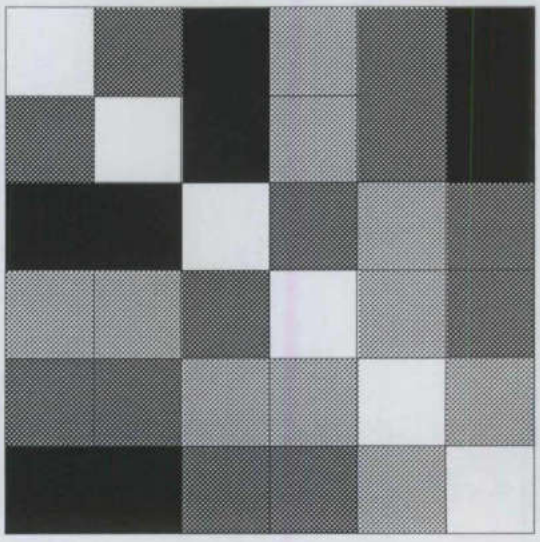


Figura 4.3 *Matriz de distancias correspondiente a smallTree*

Podemos reconocer el árbol original como la porción en gris en la figura anterior.

Con la ayuda de la función *distances* podemos calcular los centros (o centro) de un árbol dado, definido como aquellos nodos que minimizan el máximo de las distancias a todos los demás nodos.

```
centerTree[t_] := Module[{d, i, j, n = Length[Flatten[t]]},
      d = Table[0, {n}, {n}];
      Do[
            d[[i, j]] = d[[j, i]] = distance[t, i, j], {i, n - 1},
{j, i + 1, n}];
      d = Max /@ d;
      Flatten[Position[d, First[Union[d]]]]]

centerTree[smallTree]
centerTree[bigTree1]
centerTree[bigTree2]
{4, 5}
{3, 12}
{1, 2}
```

Notemos la verificación de la propiedad de que los centros son únicos o adyacentes.

4.7. Recorrido de Árboles

Presentamos cuatro formas de recorrer árboles. Dos de ellas corresponden al recorrido tradicional de pre-orden y post-orden. La tercera, denominada *naturalOrder*, se basa en la apariencia natural de las hojas presente en la notación (corta o larga) de un árbol. Finalmente, diseñamos la función *breathOrder*, la cual visita los nodos de manera transversal conforme nos trasladamos a niveles inferiores.

```
preOrder[r_] := r
preOrder[{r_, sT_}] := Flatten[{r, preOrder /@ sT}]

preOrder[smallTree]
preOrder[bigTree1]
preOrder[bigTree2]
{4, 1, 5, 3, 6, 2}
{5, 10, 8, 3, 11, 12, 2, 13, 14, 15, 16, 17, 18, 7, 9, 1, 6, 4}
{1, 5, 11, 10, 18, 12, 4, 9, 2, 6, 13, 19, 20, 21, 3, 7, 14, 15, 8,
16, 17}

postOrder[r_] := r
postOrder[{r_, sT_}] := Flatten[{postOrder /@ sT, r}]

postOrder[smallTree]
postOrder[bigTree1]
postOrder[bigTree2]
{1, 3, 6, 5, 2, 4}
{8, 10, 11, 13, 15, 16, 17, 18, 14, 2, 12, 7, 3, 1, 6, 9, 4, 5}
{11, 18, 10, 12, 5, 9, 4, 19, 20, 21, 13, 6, 2, 14, 15, 7, 16, 17,
8, 3, 1}

naturalOrder[t_] := Flatten[t]

naturalOrder[smallTree]
naturalOrder[bigTree1]
naturalOrder[bigTree2]
{4, 1, 5, 3, 6, 2}
{5, 10, 8, 3, 11, 12, 2, 13, 14, 15, 16, 17, 18, 7, 9, 1, 6, 4}
```

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

Notemos que este orden coincide con el de preOrder.

```
breathOrder[t_] := Module[{ans = {}, h},
      FixedPoint[(h = Flatten[#, 2];
                  AppendTo[ans, Select[h, AtomQ]];
                  Select[h, ListQ]) &, t];
      Flatten[ans]]

breathOrder[smallTree]
breathOrder[bigTree1]
breathOrder[bigTree2]
{4, 1, 5, 2, 3, 6}
{5, 10, 3, 9, 4, 8, 11, 12, 7, 1, 6, 2, 13, 14, 15, 16, 17, 18}
{1, 5, 4, 2, 3, 11, 10, 12, 9, 6, 7, 8, 18, 13, 14, 15, 16, 17, 19,
20, 21}
```

4.8. Dibujo de Árboles

La representación más común de un árbol, preferida sobre la corta o larga, es la gráfica. El propósito de esta sección será proveer de una función para el dibujo de arboles. Dos funciones adicionales se presentaran al final de esta sección.

La función auxiliar *drT* siguiente dibuja un árbol dado de tal manera que su raíz se ubica en el punto (x, y). La función principal *drawT* calcula la representación gráfica de un árbol. Sus etiquetas son dibujadas utilizando un color y tamaño especifico. Los nodos se dibujan como círculos de un color dado y el color de las aristas puede también ser elegido. Una variedad de ejemplos siguen al código.

```
"Graphics`Colors`"

drT[t : {r_, sT_}, {x_, y_}] :=
  Module[{w = width[t],wSons,n = Length[sT],sons,i,treeGraph = {}},
wSons = width /@ sT;
```

```
sons = Table[{x-w/2,y-1} + {wSons[[i]]/2 + Plus @@ Take[wSons,i-1],0},
{i, n}];
MapThread[AppendTo[treeGraph, {Line[{#1, {x, y}}], Disk[#1, 0.3],
  Text[First[#2], #1]}] &, {sons, sT}];
MapThread[ AppendTo[treeGraph, drT[#2, #1]] &, {sons, sT}];
Flatten[treeGraph]]
  drawT[t_, {textColor_, textSize_}, nodesColor_, edgesColor_] :=
    Module[{ts, d, dL, dD, dT}, ts = $TextStyle;
    TextStyle = {FontFamily -> "Arial", FontSize -> textSize,
FontWeight -> "Bold", FontColor -> textColor, Background -> None};
d = drT[t, {0, 0}];
dL = Cases[d, Line[_]];
dD = Cases[d, Disk[_, _]];
dT = Cases[d, Text[_, _]];
d = {{edgesColor, AbsoluteThickness[3], dL},
{nodesColor, dD, Disk[{0, 0}, 0.4]}, {dT, Text[First[t], {0, 0}]}};
Show[Graphics[{d}, AspectRatio -> Automatic],
PlotRegion -> {{0.1, 0.9}, {0.1, 0.9}}];
$TextStyle = ts;]

drawT[bigTree1, {Cobalt, 9}, Floral, EnglishRed];
```

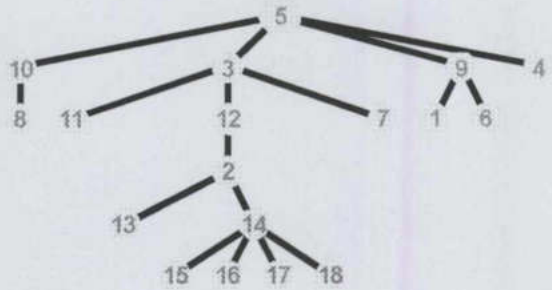


Figura 4.4 bigTree1

```
drawT[bigTree2, {VenetianRed, 10}, CyanWhite, Chocolate];
```

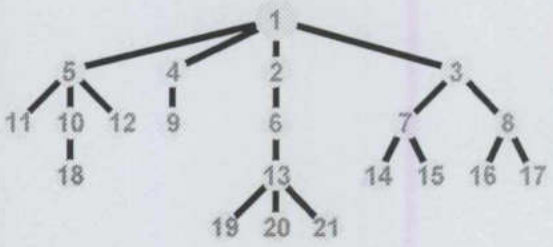


Figura 4.5 bigTree2

4. 9. Forma Normal

El objetivo de esta sección es generar todos los árboles no-isomorfos de un determinado número de nodos. Para tal fin se buscará un *invariante* de árboles. Claramente el *número de nodos* no es un invariante, pero es útil para distinguir de manera rápida los árboles no-isomorfos. Además del número de nodos, la *altura* de un árbol no puede ser un invariante, ya que se podrían encontrar pares de árboles isomorfos y no-isomorfos que tengan el mismo número de nodos y la misma altura, por ejemplo, 1(2(3),4) y 1(2,3(4)) ó 1(2(3),4) y 1(2,3,4) respectivamente. Agregando a estos dos últimos criterios la distinción de *anchura*, es posible entonces distinguir el último par pero no el primero. Estas tres cualidades, aunque útiles para diferenciar una gran cantidad de árboles no-isomorfos no constituyen el invariante completo que se busca.

Antes de extender la lista de propiedades, se diseñará la función *liftUp*, la cual se constituirá en un instrumento para la provisión de una cuarta propiedad que se pueda agregar al invariante que se construye. Esta función tomará un determinado árbol *t* y un entero asociado, y levantará el árbol de modo que el nodo correspondiente a la etiqueta dada por el entero asociado estará ahora en la parte superior de *t*, re-arreglándolo apropiadamente como se ilustra en los siguientes ejemplos:

```
liftUp[{r_, {sT__}}, r_] := {r, {sT}}
liftUp[t_, r_] := Module[{p = toPairs[t], q, i, j},
q = FixedPointList[p /. {{___,{k_,#}, ___} -> k, {___} -> 0} &, r];
  fromPairs[Map[({i, j} = #;
  If[MemberQ[q, j], {j, i}, #]) & , p]]]
```

```
drawT[liftUp[smallTree, 3], {Snow, 20}, Indigo, Firebrick];
```

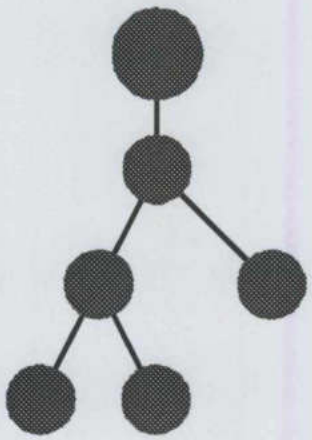


Figura 4.6 smallTree re-arreglado por el vértice 3

```
drawT[liftUp[bigTree1, 16], {Cobalt, 9}, Floral, EnglishRed];
```

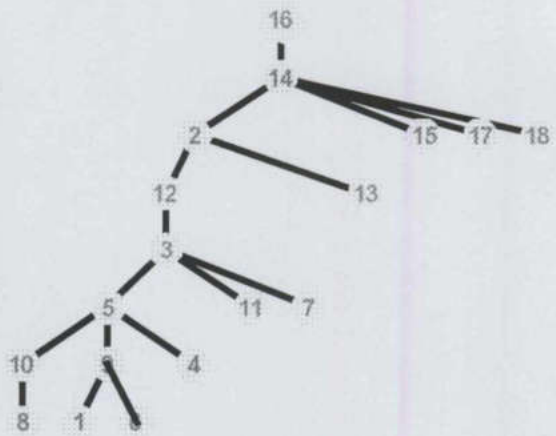


Figura 4.7 bigTree1 re-arreglado por el vértice 16

```
t = bigTree1;
drawT[liftUp[t,
      centerTree[t][[1]]], {Cobalt, 9},
    Floral, EnglishRed];
```

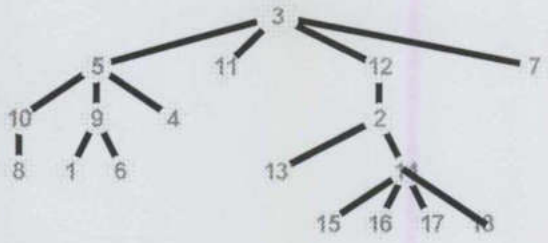


Figura 4.8 *bigTree1 re-arreglado por el vértice central 3*

Para controlar la manera en que se lleva a cabo el reacondicionamiento de un árbol, se diseñará enseguida la función *shake*. Acondicionar un árbol significa ordenar sus subárboles lexicográficamente de acuerdo a su altura, anchura y número de nodos, de tal forma que los subárboles más grandes se localicen a la izquierda.

```
shake[{r_, {}}] := {r, {}}
shake[{r_, {sT__}}] :=
  Module[{h = {#, height[#], width[#], numNodes[#]} & /@ {sT}},
      {r, shake /@ First /@ Sort[ h, ((#2[[2]] < #1 [[2]]) ||
((#2[[2]] == #1[[2]]) && (#2[[3]] < #1[[3]])) || ((#2[[2]] ==
#1[[2]]) && (#2[[3]] == #1[[3]]) && (#2[[4]] < #1[[4]]))) &]}]
```

Levantar un árbol a partir de cualquiera de sus centros produce árboles que tienen la misma profundidad (la mínima posible) y anchura, como se muestra enseguida:

```
t = bigTree2;
drawT[f1 = shake[liftUp[t, centerTree[t][[1]]]],
{VenetianRed, 10}, CyanWhite, Chocolate];
```

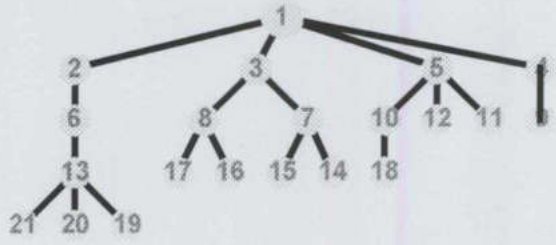


Figura 4.9 *bigTree2 re-arreglado por el vértice central 1*

```
t = bigTree2;
drawT[f2 =
      shake[liftUp[t, centerTree[t][[2]]]], {VenetianRed, 10},
CyanWhite, Chocolate];
```

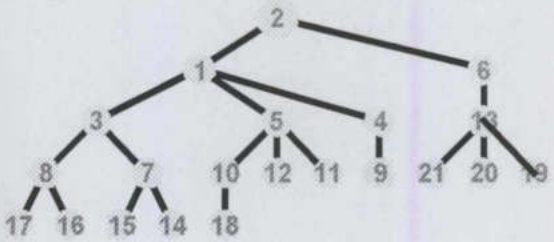


Figura 4.10 *bigTree2 re-arreglado por el vértice central 2*

Se busca asociar una forma única a un árbol de modo tal que dos árboles puedan calificarse de isomorfos si y solo si tienen la misma forma, la cual se llamará *forma normal*. La operación consistente en levantar un árbol desde cualquiera de sus centros, o centro, y arreglarlo de acuerdo a la función shake conduce a un buen candidato para la forma normal. Esta forma normal será una adición útil al invariante buscado en la medida que se comparen formas normales para distinguir dos árboles dados.

Sin embargo, en el caso de existir dos centros, el problema es como elegir uno de ellos. En este caso considere la función *energy* (por analogía con la energía potencial de un objeto, siendo los nodos de un árbol considerados como masas unitarias) de un árbol como la suma de las distancias de los nodos a la raíz.

```
energy[t : {r_, _}] := Module[{d, i, n = Length[Flatten[t]]},
      d = Table[distance[t, i, r], {i, n}];
      Plus @@ d]

energy[longNotation["1(2(3,4))"]]
energy[longNotation["1(2(3),4)"]]
5
4
```

```
energy[f1]
energy[f2]
48
57
```

así, elegimos el centro que provee mínima energía.

```
normalForm[t_] := Module[{c = centerTree[t], t1, t2, e1, e2},
      t1 = liftUp[t, c[[1]]];
      If[Length[c] == 2,
            t2 = liftUp[t, c[[2]];
            e1 = energy[t1];
            e2 = energy[t2];
            If[e1 > e2, t1 = t2]];
      shake[t1]]

drawT[normalForm[bigTree1], {Cobalt, 9}, Floral, EnglishRed];
drawT[normalForm[bigTree2], {VenetianRed, 10}, CyanWhite,
Chocolate];
```

El candidato para conformar la invariancia, por ejemplo, para la forma normal, incluirá todos los criterios anteriores más la igualdad de la forma normal (incluyendo las etiquetas). Sin embargo, existe un par de árboles isomorfos que tienen el mismo número de nodos, altura, anchura y forma normal diferente, es decir (1(2(3(4,5),6(7),8) y 1(2(3(4),5),6(7(8))). Se necesita encontrar una propiedad adicional. Para ello se iniciará diseñando las funciones *leftBranch*, la cual calcula el conjunto de nodos que comprende la ruta localizada más a la izquierda de un determinado árbol y *lexicographiQ*, que prueba si dos secuencias dadas están ordenadas lexicográficamente.

```
leftBranch[{r_, {}}] := {r}
leftBranch[{r_, {t_, ___}}] := Flatten[{r, leftBranch[t]}]
leftBranch[bigTree2]
leftBranch[normalForm[bigTree2]]
lexicographicQ[{}, {}] := True
lexicographicQ[{a_, ___}, {b_, ___}] := False /; a > b
```

```
lexicographicQ[{a_, ___}, {b_, ___}] := True /; a < b
lexicographicQ[{a_, b___}, {c_, d___}] := lexicographicQ[{b}, {d}]

LexicographicQ[{1, 2, 3, 1}, {1, 2, 2, 1}]
False

lexicographicQ[{2, 2, 2, 1}, {2, 3, 2, 4}]
True
```

La función *hull* que aparece enseguida selecciona las ramas ubicadas más a la izquierda de los dos primeros subárboles de un determinado árbol, los une a la raíz y calcula lexicográficamente la mínima secuencia de grados asociada.

Por ejemplo, los dos primeros subárboles de *bigTree2* inician en los nodos 5 y 4, y sus correspondientes ramas izquierdas son {5, 11} y {4, 9} respectivamente. Siempre se reservará la primera secuencia, se agregará a la raíz y a la segunda secuencia, dando en este caso la secuencia total de nodos 11, 5, 1, 4, 9 (para la forma normal de *bigTree2* se obtendría la secuencia 21, 13, 6, 2, 1, 3, 8, 17.) Sus grados correspondientes son 1, 4, 4, 2, 1. Al comparar esta secuencia con su inversa 1, 2, 4, 4, 1 se ve que la última es lexicográficamente la menor y entonces se puede elegir como la cubierta de *bigTree2* (para la forma normal de *bigTree2* se obtendría la secuencia de grados 1, 4, 2, 2, 4, 3, 3, 1).

```
hull[t : {r_, {{r1_, sT1_}, {r2_, sT2_}, ___}}] :=
  Module[{l1 = leftBranch[{r1, sT1}], l2 =leftBranch[{r2, sT2}], l,
      d = degreesTree[t], d1, d2},
      l = Join[Reverse[l1], {r}, l2];
      d1 = d[[l]];
      d2 = Reverse[d1];
      If[lexicographicQ[d1, d2], d1, d2]]

hull[bigTree2]
{1, 2, 4, 4, 1}

Timing[hull[normalForm[bigTree2]]]
{9.12 Second, {1, 3, 3, 4, 2, 2, 4, 1}}
```

```
Timing[hull[normalForm[bigTree1]]]
{5.98 Second, {1, 3, 4, 4, 2, 3, 5, 1}}
```

4. 10. Generación de árboles no isomorfos

Al efectuar la comparación de formas normales se ignoran las etiquetas de los arboles resultantes, sin embargo, estos son necesarios para servir de soporte al árbol, por esta razón adoptaremos una nueva forma de etiquetar árboles que utiliza el orden natural de las antiguas etiquetas. La función *skin* efectúa dicho etiquetado:

```
skin[t_] := Module[{q = Characters[shortNotation[t]], i, j = 1},
   Do[
     If[q[[i]] == "0",q [[i]] = ToString[j]; j++], {i, Length[q]}];
      longNotation[StringJoin[q]]]

shortNotation[bigTree1]
shortNotation[skin[bigTree1 /. _Integer -> 0]]
5(10(8),3(11,12(2(13,14(15,16,17,18))),7),9(1,6),4)
1(2(3),4(5,6(7(8,9(10,11,12,13))),14),15(16,17),18)
```

La siguiente función *grow* calcula la familia de todos los árboles que se obtienen podando una nueva rama (arista) a un árbol dado.

```
grow[t : {r_, {sT__}}] := Module[{n, v = Flatten[t], trees},
      n = Max[v] + 1;
      trees =
Map[normalForm[fromPairs[Append[toPairs[liftUp[t,#]], {n, #}]]] &,
 v];
skin /@ Union[trees  /. _Integer -> 0]]

shortNotation /@ grow[longNotation["1(2(3),4)"]]
{1(2(3),4(5)), 1(2(3),4,5)}

shortNotation /@ grow[longNotation["1(2,3,4)"]]
{1(2(3),4,5), 1(2,3,4,5)}
```

Finalmente, la función *nonIsomorphicTrees* obtiene todos los árboles no-isomorfos posibles de n nodos.

```
nonIsomorphicTrees[1] := {{{1, {}}, 1, 1, 0, {1}}}
nonIsomorphicTrees[2] := {{{1, {{2, {}}}}, 1, 2, 1, {1, 1}}}
nonIsomorphicTrees[n_] :=
  nonIsomorphicTrees[n] =
    Module[{u = First /@ nonIsomorphicTrees[n - 1], v, w, t},
      v = Union[Flatten[Map[grow[#] &, u], 1]];
      w = Map[{#, width[#], height[#], energy[#], hull[#]} &, v];
      w //. {a___, {b_, c_, d_, e_, f_}, g___, {_, c_, d_, e_, f_},
            h___} -> {a, {b, c, d, e, f}, g, h}]
```

Se muestra a continuación tiempos de corrida, los árboles y sus propiedades da hasta 7 nodos como se procesaron en una computadora PC de 200 MHz de velocidad (ver la sucesión de Sloan número 299 correspondiente a árboles no-etiquetados).

```
Do[
u = Timing[t = nonIsomorphicTrees[n];];
Print["Tiempo de Cómputo: ", First[u]/(60 Second),
"Minutos. Existen", Length[t], " Árboles de ", n, " nodos:"];
Print[Map[Join[{shortNotation[First[#]]}, Rest[#]] &, t] //
MatrixForm], {n, 3, 10}]
```

```
Tiempo de Cómputo: 0.0036666666666666475 Minutos. Existen 1 Árboles
de 3 nodos:
(1 (2, 3) 2 1 2 {1, 2, 1})
```

```
Tiempo de Cómputo: 0.0155 Minutos. Existen 2 Árboles de 4 nodos:
```

1(2(3),4)	2	2	4	{1,2,2,1}
1(2,3,4)	3	1	3	{1,3,1}

```
Tiempo de Cómputo: 0.0548333 Minutos. Existen 3 Árboles de 5nodos:
```

1(2(3),4(5))	2	2	6	{1,2,2,2,1}
1(2(3),4,5)	3	2	5	{1,2,3,1}
1(2,3,4,5)	4	1	4	{1,4,1}

```
Tiempo de Cómputo: 0.1365 Minutos. Existen 6 Árboles de 6 nodos:
```

1(2(3(4)),5(6))	2	3	9	{1,2,2,2,2,1}
1(2(3,4),5(6))	3	2	8	{1,2,2,3,1}
1(2(3),4(5),6)	3	2	7	{1,2,3,2,1}
1(2(3,4),5,6)	4	2	7	{1,3,3,1}
1(2(3),4,5,6)	4	2	6	{1,2,4,1}
1(2,3,4,5,6)	5	1	5	{1,5,1}

```
Tiempo de Cómputo: 0.4595 Minutos. Existen 11 Árboles de 7 nodos:
```

1(2(3(4)),5(6(7)))	2	3	12	{1,2,2,2,2,2,1}
1(2(3(4)),5(6,7))	3	3	11	{1,2,2,2,3,1}
1(2(3,4),5(6,7))	4	2	10	{1,3,2,3,1}
1(2(3,4,5),6(7))	4	2	10	{1,2,2,4,1}
1(2(3),4(5),6(7))	3	2	9	{1,2,3,2,1}
1(2(3(4)),5(6),7)	3	3	10	{1,2,2,3,2,1}
1(2(3,4),5(6),7)	4	2	9	{1,2,3,3,1}
1(2(3),4(5),6,7)	4	2	8	{1,2,4,2,1}
1(2(3,4),5,6,7)	5	2	8	{1,3,4,1}
1(2(3),4,5,6,7)	5	2	7	{1,2,5,1}
1(2,3,4,5,6,7)	6	1	6	{1,6,1}

Podemos desplegar todos estos árboles utilizando la función *drawT* como se muestra a continuación:

```
t = First /@ nonIsomorphicTrees[6];
Map[drawT[#, {White, 10}, Brown, Orange] &, t];
```

4.11. Árboles Binarios

Los árboles binarios constituyen una de las subfamilias más importantes de árboles. Dichos árboles están caracterizados por la propiedad de que cualquier nodo esta unido a otros dos. Nuestro propósito en ésta sección es la generación de todos los árboles binarios de una altura dada y que posean un número específico de nodos. Adoptaremos la siguiente notación: se utiliza la etiqueta y para denotar una hoja y la etiqueta x para denotar una arista ausente. Por ejemplo, la notación $\{\{x,\{y,y\}\},x\}$ representa el árbol de la figura 4.11

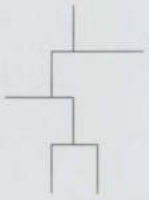


Figura 4.11 *Representación del árbol* $\{\{x,\{y,y\}\},x\}$

Notemos que no es posible usar la función *drawT* que se presentó en la sección 4.8 debido a que los árboles binarios están orientados (es decir, la rama izquierda es distinta de la derecha e inclusive puede estar ausente). Una representación gráfica más común y que será utilizada en la última parte de esta tesis es:

Figura 4.12 *Otra Representación del árbol* $\{\{x,\{y,y\}\},x\}$

Enumeración

En esta sección se obtendrá el número de árboles binarios, a los cuales nos referiremos de manera abreviada solamente como árboles. Como fue el caso para los árboles con raíz generales, adoptaremos un formato simplificado para los árboles binarios:

```
showTree[{tL_, tR_}] := "(" <> showTree[tL] <> showTree[tR] <> ")"
showTree[x_] := x
showTree[{{{"y", "x"}, "x"}, "x"}]
(((yx)x)x)
```

Con el fin de tener una versión extendida de la función *Outer* que será de utilidad posteriormente y que manipule listas de listas se diseñara a continuación la función *myOuter*:

```
myOuter[list1_, list2_] := Flatten[Outer[List, list1, list2, 1], 1]
myOuter[{1, 3, 2, 4}, {3, 4, 1}]
{{1, 3}, {1, 4}, {1, 1}, {3, 3}, {3, 4}, {3, 1},
```

```
{2, 3}, {2, 4}, {2, 1}, {4, 3}, {4, 4}, {4, 1}}

myOuter[{{1, 2}, {3, 1}}, {{1, 3}, {1, 2}}]
{{{1, 2}, {1, 3}}, {{1, 2}, {1, 2}}, {{3, 1},
{1, 3}}, {{3, 1}, {1, 2}}}
```

Dada una altura (profundidad) *d*, la función *ij* genera todos los pares de enteros positivos (i, j) tales que $max(i, j) = d$.

```
ij[d_] := Append[Flatten[Table[{{i, d}, {d, i}},
      {i, 0, d - 1}], 1], {d, d}]
ij[5]
{{0, 5}, {5, 0}, {1, 5}, {5, 1}, {2, 5},
{5, 2}, {3, 5}, {5, 3}, {4, 5}, {5, 4}, {5, 5}}
```

Todos los árboles de una altura dada y números de nodos especifico se pueden obtener utilizando la función *binaryTrees*:

```
binaryTrees[0, 1] := {"y"}
binaryTrees[0, _] := {}
binaryTrees[dep_, nodes_] :=
  binaryTrees[dep, nodes] =
    Module[{t, d = dep - 1, n = nodes - 1, case1, case2,
      case3 = {}, tih, tjk,h, u, v},
      t = binaryTrees[d, n];
      case1 = ({#, "x"} &) /@ t;
      case2 = ({"x", #} &) /@ t;
      Map[({u, v} = #;
       Do[
         tih = binaryTrees[u, h];
         tjk = binaryTrees[v, n - h];
         AppendTo[case3, myOuter[tih, tjk]], {h, n - 1}]) &, j[d]];
    Join[case1, case2, Flatten[case3, 1]]]

showTree /@ binaryTrees[2, 3]
{((yx)x), ((xy)x), (x(yx)), (x(xy))}
```

Para obtener, por ejemplo, todos los árboles de altura 2, procedemos como sigue. Notemos que el número de nodos de un árbol es igual al número de *x*'s más el doble del número de *y*'s menos uno.

```
Do[Print[n, "\t", b = binaryTrees[2, n]; showTree /@ b, Length[b]],
{n, 7}]

1       {}      0
2       {}      0
3       {((yx)x), ((xy)x), (x(yx)), (x(xy))}        4
4       {((yy)x), (x(yy)), (y(yx)), (y(xy)), ((yx)y), ((xy)y)}      6
5       {(y(yy)), ((yy)y), ((yx)(yx)), (yx)(xy)),
        ((xy)(yx)), ((xy)(xy))} 6
6       {((yx)(yy)), ((xy)(yy)), ((yy)(yx)), ((yy)(xy))}        4
7       {((yy)(yy))}        1
```

La distribución correspondiente a los 651 árboles de altura 3 se obtiene como se muestra a continuación. En ella es posible leer, por ejemplo, que existen 20 de tales árboles de 5 nodos.

```
Table[Length[binaryTrees[3, n]], {n, 2^4 - 1}]
  Plus @@ %
{0, 0, 0, 8, 20, 40, 68, 94, 114, 116, 94, 60, 28, 8, 1}
651
```

Con la ayuda de la función *ij*, la función *nT* calcula el número de árboles de una profundidad dada:

```
nT[0] := 1
nT[dep_] :=
nT[dep] =
   2 nT[dep-1] + Plus @@ (nT[First[#]] nT[Last[#]] &) /@ ij[dep-1]
Table[nT[d], {d, 0, 6}]
{1, 3, 21, 651, 457653, 210065930571, 44127887745696109598901}
```

La sucesión anterior se reporta en el libro de Sloane como la número 1251 correspondiente a una "Recurrencia no-lineal".

Consideremos ahora que dos árboles dados son isomorfos si uno puede ser obtenido del otro rotando (transponiendo) algunas de sus ramas. La función *nTred* cuenta cuantas de ellas que posean una profundidad dada son no-isomorfas.

```
nTRed[0] := 1
nTRed[dep_] :=
nTRed[dep] =
```

$$\text{nTRed[dep - 1]} \left(2 + \sum_{i=0}^{\text{dep}-2} \text{nTRed[i]} + \frac{1}{2}(\text{nTRed[dep - 1]} - 1) \right)$$

```
Table[nTRed[d], {d, 0, 6}]
{1, 2, 7, 56, 2212, 2595782, 3374959180831}
```

La anterior corresponde a la secuencia 718 en el libro de Sloane.

A partir del anterior desarrollo es posible ahora obtener la distribución de los árboles no-isomorfos de una altura y número de nodos específicos. El siguiente ejemplo muestra aquellas distribuciones de a lo más altura 4.

```
nTRed[0, 1] := 1
nTRed[0, _] := 0
nTRed[_, 1] := 0
nTRed[dep_, nodes_] :=
nTRed[dep, nodes] =
  Module[{d = dep - 1, n = nodes - 1, s1, s2, s3, i, h},
    s1 = nTRed[d, n];
```

$$\text{s2} = \sum_{i=0}^{d-1} \sum_{h=1}^{n-1} \text{nTRed[i,h]}\ \ \text{nTRed[d,n-h]};$$

$$\text{s3} = \frac{1}{2}\left(\sum_{h=1}^{n-1} \text{nTRed[d,h]}\ \ \text{nTRed[d,n-h]} + \text{If[OddQ[n],0,nTRed[d,}\frac{n}{2}\text{]]} \right);$$

```
    s1 + s2 + s3]
u = Table[nTRed[d, i], {d, 4}, {i, 2^(d + 1) - 1}];
  Do[Print[d, "  ", u[[d]], "  ", Plus @@ u[[d]]], {d, 4}]
```

```
1     {0, 1, 1}   2
2     {0, 0, 1, 2, 2, 1, 1}   7
3     {0, 0, 0, 1, 3, 5, 7, 8, 9, 7, 7, 4, 3, 1, 1}   56
4     {0, 0, 0, 0, 1, 4, 9, 17, 29, 45, 66, 89, 118,
      142, 171, 187, 205, 202, 201, 177, 158, 123,
      99, 66, 47, 26, 17, 7, 4, 1, 1}     2212
```

Generación

En ésta sección se realizará la generación de todos los árboles enumerados en la sección anterior. Primeramente se especializará la función *Outer*.

```
myOuter2[singleList_]:=Union[Sort /@myOuter[singleList,singleList]]
myOuter2[{1, 2, 3}]
{{1, 1}, {1, 2}, {1, 3}, {2, 2}, {2, 3}, {3, 3}}
myOuter2[{{1, 2}, {3, 1}}]
{{{1, 2}, {1, 2}}, {{1, 2}, {3, 1}}, {{3, 1}, {3, 1}}}
```

De manera similar al caso de la enumeración de árboles no-isomorfos, se presentan tres casos en el caso de su generación. La función *binTreesRed* prepara el cómputo utilizando las funciones desarrolladas previamente.

```
Clear[binTreesRed]
binTreesRed[dep_, nodes_] := {} /; Not[dep < nodes < 2^(dep + 1)]
binTreesRed[0, 1] = {"y"};
binTreesRed[dep_, nodes_] :=
binTreesRed[dep, nodes] =
  Module[{case1, case2 = case3 = case4 = {}, i, h, ih, tih,
    tOther, lim},
  case1 = Map[{#, "x"} &, binTreesRed[dep - 1, nodes - 1]];
  ih = myOuter[Range[0, dep - 2], Range[nodes - 2]];
  ih =
     Select[ih, (Max[First[#], nodes - 2^dep - 1] < Last[#] <
        Min[nodes - dep, 2^(First[#] + 1)]) &];
   Map[({i, h} = #;
        tih = binTreesRed[i, h];
        tOther = binTreesRed[dep - 1, nodes - 1 - h];
```

```
        AppendTo[case2, myOuter[tih, tOther]]) &, ih];
    case2 = Flatten[case2, 1];
    lim = (nodes - 1)/2;
    If[OddQ[nodes], lim = lim - 1];
    ih = myOuter[{dep - 1}, Range[lim]];
    ih = Select[ih, (Max[dep - 1, nodes - 2^dep - 1] < Last[#] <
        Min[nodes - dep, 2^dep]) &];
    Map[({i, h} = #;
      tih = binTreesRed[i, h];
      tOther = binTreesRed[dep - 1, nodes - 1 - h];
      AppendTo[case3, myOuter[tih, tOther]]) &, ih];
    case3 = Flatten[case3, 1];
    If[OddQ[nodes], tih = binTreesRed[dep - 1, lim + 1];
  AppendTo[ case4, myOuter2[tih]];
  case4 = Flatten[case4, 1]];
  Join[case1, case2, case3, case4]]
```

Por ejemplo, para obtener todos los árboles diferentes de altura 1 y que posean a lo más 4 nodos:

```
Table[showTree /@ binTreesRed[1, n], {n, 4}]
{{}, {(yx)}, {(yy)}, {}}
```

y para obtener todos los árboles diferentes de 5 nodos y altura 3:

```
showTree /@ binTreesRed[3, 5]
{(((yy)x)x), ((y(yx))x), (y((yx)x))}
```

La distribución completa de los árboles no-isomorfos de altura 2 se obtiene como se muestra a continuación:

```
h = 2;
Do[Print[n, "\t", bt = binTreesRed[h, n]; showTree /@ bt, "\t",
      Length[bt]], {n, h + 1, 2^(h + 1) - 1}]
3     {((yx)x)}   1
4     {((yy)x), (y(yx))}      2
```

```
5      {(y(yy)), ((yx)(yx))}   2
6      {((yx)(yy))}            1
7      {((yy)(yy))}            1
```

La longitud de *binTreesRed[h, n]* es positiva solamente en el rango $h < n < 2^{h+1}$. Solamente hay un único árbol reducido cuando $n \in \{h+1, 2^{h+1}-1\}$.

Representación Gráfica 1

Mostraremos a continuación la primera de dos formas diferentes de dibujar árboles binarios. Dada una lista de líneas y un punto v, la función *addLines* traslada todas las líneas una cantidad v como se muestra en el ejemplo.

```
addLines[li_, v_] := Module[{u = li /. Line[l_] -> l, h},
   (Line[#] &) /@ ((h = #; (# + v &) /@ h) &) /@ u]

addLines[{Line[{{1, 2}, {3, 4}}], Line[{{0, 0}, {0, 1},
{-1, -1}}]}, {1, 2}]

{Line[{{2, 4}, {4, 6}}], Line[{{1, 2}, {1, 3}, {0, 1}}]}
```

La siguiente función *treeShape* calcula una versión ortogonal de la representación gráfica de un árbol:

```
treeShape["x"] := {1, 1, 0, {}}
  treeShape["y"] := {1, 1, 0, {Line[{{1, 0}, {1, 1}}]}}
  treeShape[{tL_, tR_}] :=
    Module[{h, a, b, L, k, c, d, R, ahk, hp, ap, bp, ans,
        delta = 1},
    {h, a, b, L} = treeShape[tL];
    {k, c, d, R} = treeShape[tR];
    ahk = If[h > k, 0, Abs[h - k]];
    hp =  Max[h, k] + 1;
    ap = a + ((c + d) (delta (a + b + c + d) - a - d)) / (a + b + c + d);
    bp = delta(a + b + c + d) - ap;
    ans = addLines[L, {0, ahk}];
```

```
    AppendTo[ans, {Line[{{a, h + ahk}, {ap + bp - d, h + ahk}}],
        Line[{{ap, h + ahk}, {ap, h + ahk+1}}]}];
    AppendTo[ans, addLines[R, {ap + bp - c - d, h + ahk - k}]];
    {hp, ap, bp, Flatten[ans]}]
    drawAllTrees[t_] :=
    Module[{u = Last[treeShape[t]]},
    Show[Graphics[u], AspectRatio -> Automatic]]

drawTree[{{{"y", "x"}, "x"}, "x"}];
```

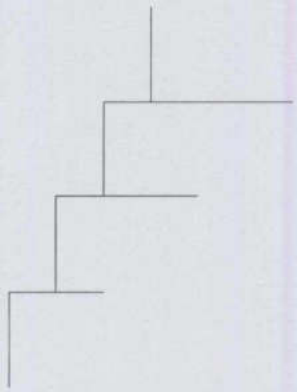


Figura 4.13 representación gráfica del árbol {{{y, x},x},x}

Utilizando lo anterior es posible generar un arreglo de gráficas que contenga todos los diferentes árboles de una altura dada.

```
drawAllTrees[trees_, nColumns_] :=
   Module[{u, n = Length[trees]},
   u = (Graphics[Last[treeShape[#]]] &)\) /@ trees;
   u = Join[u, Table[Graphics[Point[{0, 0}]],
        {nColumns Ceiling[n/nColumns] - n}]];
   Show[GraphicsArray[Partition[u, nColumns]],
    AspectRatio -> Automatic]]
t = Flatten[Table[binaryTrees[2, n], {n, 3, 2^3 - 1}], 1];
drawAllTrees[t, 5];
```

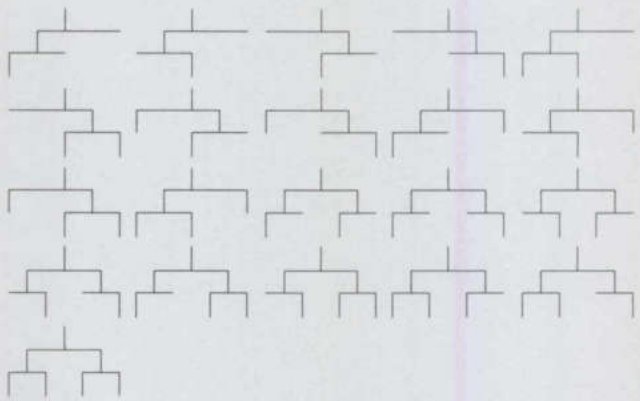


Figura 4.14 *Todos los árboles de altura 5*

Representación Gráfica 2

Una representación gráfica alternativa es aquella que utiliza líneas oblicuas para conectar nodos. Dicha representación es provista por la siguiente función *treeShape2* y es utilizada para generar el arreglo gráfico correspondiente en la función *drawAllTrees2*.

```
treeShape2["x"] := {1, 1, 0, {}}
  treeShape2["y"] := {1, 1, 0, {Line[{{1, 0}, {1, 1}}]}}
  treeShape2[{tL_, tR_}] :=
    Module[{h, a, b, L, k, c, d, R, ahk, hp, ap, bp, ans,
        delta = 1},
    {h, a, b, L} = treeShape2[tL];
    {k, c, d, R} = treeShape2[tR];
    ahk = If[h > k, 0, k - h];
    hp =  Max[h, k] + 1;
    ap = a + (c + d) (delta (a + b + c + d) - a - d) / (a + b + c + d);
    bp = delta(a + b + c + d) - ap;
    ans = addLines[L, {0, ahk}];
    AppendTo[ans, {Line[{{a, h + ahk}, {ap, h + ahk + 1},
        {ap + bp - d, h + ahk}}]}];
    AppendTo[ans, addLines[R, {ap + bp - c - d, h + ahk - k}]];
    {hp, ap, bp, Flatten[ans]}]
    drawAllTrees2[trees_, nColumns_] :=
    Module[{u, n = Length[trees]},
    u = (Graphics[Last[treeShape2[#]]] &)\) /@ trees;
```

```
u = Join[u, Table[Graphics[Point[{0, 0}]],
{nColumns Ceiling[ n / nColumns ] - n}]];
Show[GraphicsArray[Partition[u, nColumns]],
AspectRatio -> Automatic]]
```

A continuación mostramos el número de árboles no-isomorfos de profundidad 1, 2 y 3. Notemos que para realizar la generación de árboles no-isomorfos de profundidad *d*, generamos los casos de profundidad *d*-1, *d*-2, ..., 1 de manera recursiva y debido al uso de la función *binTreesRed* que almacena los valores que ha encontrado.

```
d = 1;
t = Flatten[Table[binTreesRed[d, n], {n, d+1, 2^(d+1) - 1}],1];
drawAllTrees2[t, 5];
  Length[t]

2
```

Figura 4.15 *Todos los árboles no-isomorfos de profundidad 1*

```
d = 2;
t = Flatten[Table[binTreesRed[d, n], {n, d+1, 2^(d+1) - 1}],1];
drawAllTrees2[t, 5];
  Length[t]
7
```

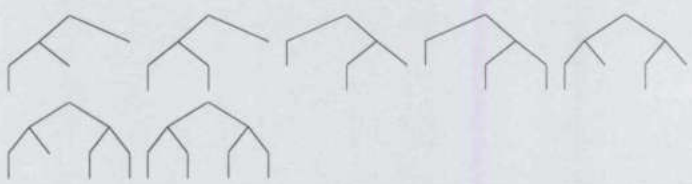


Figura 4.16 *Todos los árboles no-isomorfos de profundidad 2*

```
d = 3;
t = Flatten[Table[binTreesRed[d, n], {n, d+1, 2^(d+1) - 1}],1];
drawAllTrees2[t, 5];
  Length[t]

56
```

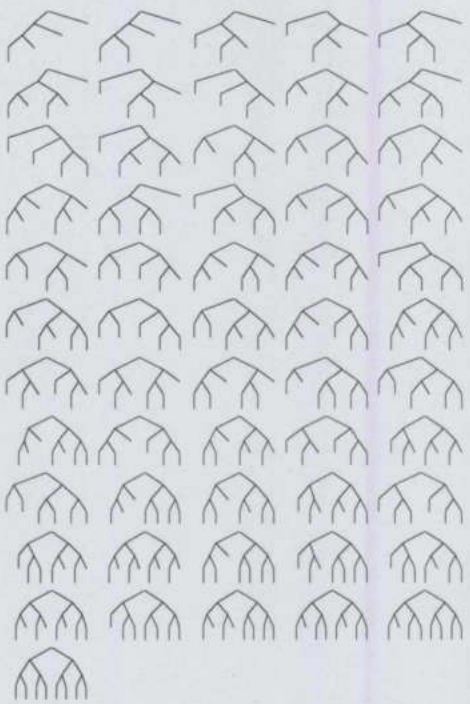


Figura 4.17 *Todos los árboles no-isomorfos de profundidad*

CONCLUSIÓN

Dentro de las abstracciones matemáticas y computacionales la estructura de grafo y sus algoritmos han tenido una amplia utilización en la solución de problemas. Los grafos son herramientas poderosas para crear modelos matemáticos de una gran complejidad y variedad de situaciones; por lo que la teoría de grafos ha sido vital para analizar y resolver problemas en áreas tan diversas.

El problema de isomorfismo de grafos es un problema algorítmico fundamental que involucra cálculos combinatorios de alto nivel. A pesar de los enormes esfuerzos invertidos en solucionar este problema, no se ha podido dar una solución computacionalmente factible. Se considera un algoritmo de complejidad entre P y NP-Completo, pero no se sabe con precisión cual es la complejidad de dicho algoritmo.

Las primeras investigaciones orientadas a solucionar el problema, realizaban pruebas exhaustivas a cada uno de los grafos posiblemente isomorfos, y se encontró que los grafos tenían propiedades totalmente independientes de la forma como se dibuja el grafo, a esta propiedad se le llamo invariante. El estudio de invariantes contribuyó significativamente en la detección de isomorfismo en grafos ya que cuando esos invariantes son iguales en ambos grafos, es posible que haya isomorfismo. Sin embargo, hay grafos con los mismos invariantes y no son isomorfos. Por ello, se buscó una cantidad adecuada de invariantes y se aplicaron a cada uno de los grafos. En este sentido surge el concepto de Certificado, que es una propiedad única de los grafos independientemente de la forma como es dibujado el grafo.

La teoría de grupos puede verse como un estudio algebraico de la simetría. Si se conoce la simetría que dos grafos poseen, entonces se puede decir si son o no isomorfos. De ahí la importancia de estudiar la teoría de grupos y sus algoritmos.

Los algoritmos para el cálculo de invariantes permite eliminar estructuras isomorfas de toda una familia de ellas. Estas pruebas han significado un ahorro de tiempo y esfuerzo en las aplicaciones donde es utilizada esta metodología. Entre las aplicaciones potenciales podemos mencionar la eliminación de trabajo redundante, ya que dada una colección de grafos y operaciones a realizar sobre esos grafos, disminuimos trabajo si antes de aplicar las operaciones eliminamos aquellos grafos que son isomorfos, esto es aplicable en pruebas de planaridad, por ejemplo. Otra aplicación, es la identificación de simetrías, las cuales se pueden obtener encontrando el conjunto de automorfismos de un grafo. Representando las estructuras químicas con grafos, se pueden identificar isómeros.

La investigación bibliográfica del problema de isomorfismo de grafos, únicamente proporciona información teórica, pero la gran mayoría de estas fuentes de información no presenta algoritmos prácticos para poder hacer las pruebas directamente a los grafos. Por ello se recurrió a diseñar posibles soluciones y concluir con una aplicación que nos permita darle una solución a este problema. El método para detectar isomorfismo a través del uso de certificados fue un método gráfico, bastante dificil de programar en la computadora. Por ello se recurrió a un método propio y factible de programas.

La aportación personal de este trabajo de investigación es el contenido temático ya que se hizo una investigación y un profundo análisis de los algoritmos hasta contar con su dominio y su implementación. El uso de *Mathematica* para las pruebas de los algoritmos implementados ayudó significativamente al logro del objetivo planteado.

Como productos generados de este trabajo de investigación, se presentó el artículo *"Técnicas de Programación Funcional en la Enumeración y generación de Grafos no-isomorfos"*. En este trabajo se mostraron algunos de los resultados analizados en la aplicación programada en lenguaje *Mathematica*. Los certificados de árboles programados

en lenguaje *Mathematica,* constituyen una herramienta fundamental y útil en el estudio de teoría de árboles. Precisamente, esta aplicación fue la parte más interesante del desarrollo de este trabajo, justificado por el hecho de usar técnicas de programación funcional.

El estudio de invariantes y certificados va mas allá de lo que aborda este trabajo de investigación. Durante el desarrollo de este trabajo surgen nuevas ideas e inquietudes y con ello se amplía el panorama que se adquiere en estudios de licenciatura. Además se incrementa la capacidad de investigación y análisis necesarias para el logro de estudios superiores. El trabajo futuro de esta tesis es la implementación gráfica de los algoritmos, por ejemplo el algoritmo del "Sol Naciente", y extender el trabajo hacia invariantes de grupos e invariantes partiendo de automorfismos. Un tema más que se puede profundizar teniendo los conocimientos adquiridos en este trabajo de investigación, es el estudio de las aplicaciones de los números de Catalán, que es donde se fundamenta el algoritmo de "Sol Naciente".

TÉCNICAS DE PROGRAMACIÓN FUNCIONAL EN LA ENUMERACIÓN Y GENERACIÓN DE GRAFOS NO-ISOMORFOS

Jaime Rangel Mondragón
Ma. Elena Vázquez Huerta

Universidad Autónoma de Querétaro
Facultad de Informática
División de Estudios de Posgrado

Resumen.
Se propone una implementación utilizando el paradigma funcional en el lenguaje *Mathematica*[1] de las funciones generadoras que permiten enumerar la totalidad de los grafos conexos no-isomorfos de un orden dado. Se describe también la generación explicita de estas estructuras a partir de un mecanismo de recursión doble.

Abstract
An implementation following the functional paradigm is proposed using the language Mathematica[1] of the generating functions that enumerate the whole set of non-isomorphic connected graphs of a given order. The explicit generation of these structures is also given following a doubly recursive mechanism.

Palabras clave
Grafos conexos. Programación funcional. Isomorfismo.

1.- Introducción

El presente trabajo tiene como objetivo la aplicación original de técnicas de programación funcional en la implementación de algoritmos de enumeración y generación de la totalidad de grafos no-isomorfos conteniendo un número específico de vértices. Con este fin se ha adoptado el lenguaje de programación *Mathematica* [15] como vehículo ideal para este tipo de aplicaciones debido a la facilidad con la que descripciones matemáticas complejas son traducidas a código inteligible por medio de la manipulación de estructuras computacionales de alto nivel [7, 8]. Aunque la metodología funcional no es de reciente invención [1], dicho paradigma ha tenido una extraordinaria difusión con el avance creciente en la velocidad de las computadoras y las recientes implementaciones de lenguajes de programación avanzados enfocados a problemas de cómputo científico, en los que su habilidad de manipulación algebraica y simbólica resulta fundamental para la construcción rápida de prototipos [6, 13].

A lo largo de este artículo supondremos que todos los grafos a considerar son no-dirigidos y simples, es decir, tales que no poseen aristas múltiples ni bucles. Dichos grafos pueden ser descritos por su matriz de adyacencia, una matriz simétrica de tamaño igual al número de vértices v que el grafo posee y cuya entrada i-j es igual a la unidad si los vértices i y j son adyacentes e igual a cero si no lo son. De esta manera, la cantidad de elementos positivos resulta coincidir con el número de aristas e siendo los elementos de la diagonal nulos. El número de tales matrices corresponden al número de relaciones simétricas e irreflexivas sobre v objetos; este número es igual a $\binom{\binom{v}{2}}{e}$, una cantidad de rápido crecimiento; por ejemplo, para v=6, y para e variando desde 0 hasta 15, dichos números son iguales a 1, 15, 105, 455, 1365, 3003, 5005, 6435, 6435, 5005, 3003, 1365, 455, 105, 15, 1, abarcando un total de 32768 posibles grafos de solo 6 vértices (la simetría en este tipo de listas es debida a la correspondencia uno-a-uno entre los grafos de v vértices y e aristas y sus grafos

[1] *Mathematica* es una marca registrada de *Wolfram Research Inc.*

complementarios de v vértices y $\binom{v}{2}$-e aristas). Los casos extremos corresponden al grafo vacío y al grafo completo. Para v=10 el total aumenta a 35184372088832, lo que descarta la idea de una enumeración exhaustiva de estos grafos con el fin de seleccionar a los no-isomorfos.

2.- Enumeración

Un resultado clásico de Harary (conocido pero no publicado por Polya) [4] es la derivación de una función generadora para la enumeración de grafos utilizando el índice de ciclo para S_n. Por medio de la teoría de la enumeración de Polya podemos calcular el número de grafos no-isomorfos sin recurrir a su generación explícita. El propósito de esta sección es ilustrar la programación funcional del resultado reportado en [3].

Como antecedente a la fórmula general, la siguiente función `lambdas` calcula todas las soluciones de la ecuación diofántica $\lambda_1 + 2\lambda_2 + \cdots n\lambda_n = n$, donde todas las variables toman valores enteros no-negativos.

```
lambdas[n_] := lambdas[n, n]

lambdas[1, n_] := {{n}}
lambdas[m_, n_] := Module[{i},
  Flatten[Table[Append[#, i] & /@ lambdas[m - 1, n - m i], {i, 0, ⌊n/m⌋}], 1]]
```

Por ejemplo, la activación `lambdas[6]` genera el conjunto solución:

```
{{6, 0, 0, 0, 0, 0}, {4, 1, 0, 0, 0, 0}, {2, 2, 0, 0, 0, 0}, {0, 3, 0, 0, 0, 0},
 {3, 0, 1, 0, 0, 0}, {1, 1, 1, 0, 0, 0}, {0, 0, 2, 0, 0, 0}, {2, 0, 0, 1, 0, 0},
 {0, 1, 0, 1, 0, 0}, {1, 0, 0, 0, 1, 0}, {0, 0, 0, 0, 0, 1}}
```

La primera de dichas definiciones utiliza el fenómeno de polimorfismo [5, 8] para subordinar el cómputo a una función auxiliar bidimensional. Dicha función instancia la ultima incógnita y obtiene recursivamente el resto de la lista solución final. El número de soluciones de dicha ecuación diofántica toma como primeros valores los números 1, 2, 3, 5, 7, 11, 15, 22, 30, 42, 56, 77, 101, que constituyen el número de particiones de n.

Definamos ahora las variables indexadas w y λ, donde u es uno de los vectores solución de lambdas mencionados anteriormente, así como la función `numGraphs` que calcula el número total de grafos no-isomorfos que poseen v vértices y e aristas y la función `allDiffGraphs` que comprende dicha totalidad para grafos de v vértices.

```
w_i_ := 1 + x^i
λ_i_ := u[[i]]

numGraphs[v_, e_] := Module[{i, q, r},
  Coefficient[Plus @@ Map[(u = #;
    1/(∏_{i=1}^{v} i^λi λi!) (∏_{i=0}^{(v-1)/2} w_{2i+1}^{i λ_{2i+1}}) (∏_{i=1}^{v/2} (w_i w_{2i}^{i-1})^{λ_{2i}}) (∏_{i=1}^{v} w_i^{i Binomial[λ_i, 2]})
    (∏_{q=1}^{v-2} ∏_{r=q+1}^{v-1} w_{LCM[q,r]}^{λq λr GCD[q,r]})) &, lambdas[v]], x, e]]

allDiffGraphs[v_] := ∑_{e=0}^{Binomial[v,2]} numGraphs[v, e]
```

Los siguientes activaciones ilustran el crecimiento de dichas funciones.

```
Table[numGraphs[v, e], {v, 5}, {e, 0, Binomial[v, 2]}] // TableForm
```

```
1
1    1
1    1    1    1
1    1    2    3    2    1    1
1    1    2    4    6    6    6    4    2    1    1
```

```
Table[allDiffGraphs[v], {v, 10}]
```

```
{1, 2, 4, 11, 34, 156, 1044, 12346, 274668, 12005168}
```

(ver sec. M1253 en [11] y notemos el hecho de que Harary reportó erróneamente el valor 12344 en lugar de 12346). Si consideramos la enumeración de solo aquellos grafos que posean v-1 aristas se tiene:

```
Table[numGraphs[v, v - 1], {v, 10}]
```

```
{1, 1, 1, 3, 6, 15, 41, 115, 345, 1103}
```

El valor 15 anterior correspondería al valor 6 en el caso de la enumeración de árboles; la diferencia reside en nuestra inclusión de grafos no-conexos.

Consideremos ahora la función generadora bidimensional g, en la que el coeficiente del término $x^e y^v$ es igual al número de grafos no-isomorfos de v vértices y e aristas. En el siguiente código incluimos su cálculo hasta un número dado de vértices `maxVer`.

```
g[x_, y_, maxVer_] := Module[{i, v, q, r},
  Plus @@ Table[y^v Expand[Plus @@ Map[(u = #;
      1/(Product_{i=1}^{v} i^λi λi!) (Product_{i=0}^{(v-1)/2} W2 i+1^(i λ2 i+1)) (Product_{i=1}^{v/2} (Wi W2 i-1)^λ2 i)
      (Product_{i=1}^{v} Wi^(i Binomial[λi, 2])) (Product_{q=1}^{v-2} Product_{r=q+1}^{v-1} WLCM[q,r]^(λq λr GCD[q,r]))) &, lambdas[v]]], {v, maxVer}]]
```

```
g[x, y, 6]
```

$$y + (1+x)\,y^2 + (1+x+x^2+x^3)\,y^3 + (1+x+2x^2+3x^3+2x^4+x^5+x^6)\,y^4 + (1+x+2x^2+4x^3+6x^4+6x^5+6x^6+4x^7+2x^8+x^9+x^{10})\,y^5 + (1+x+2x^2+5x^3+9x^4+15x^5+21x^6+24x^7+24x^8+21x^9+15x^{10}+9x^{11}+5x^{12}+2x^{13}+x^{14}+x^{15})\,y^6$$

(cf ec.11 en [1]). De esta expresión podemos identificar coincidencias con los términos generados previamente.

Obtengamos ahora el número de grafos conexos no-isomorfos. Harary [3] solamente especifica el resultado a partir de la comparación de dos funciones generadoras [12] que denominaremos A y B y que se utilizan para resolver un sistema de ecuaciones. Primeramente construiremos la estructura del polinomio de conteo buscado, incluyendo el conjunto de incógnitas a resolver.

```
c[x_, y_, maxV_] := y + Sum_{n=1}^{maxV} y^n Sum_{e=1}^{Binomial[n,2]} h_{e,n} x^e
maxV = 8;
c[x, y, maxV]
```

$$y + x\,y^2 h_{1,2} + y^3 (x h_{1,3} + x^2 h_{2,3} + x^3 h_{3,3}) + y^4 (x h_{1,4} + x^2 h_{2,4} + x^3 h_{3,4} + x^4 h_{4,4} + x^5 h_{5,4} + x^6 h_{6,4}) + y^5 (x h_{1,5} + x^2 h_{2,5} + x^3 h_{3,5} + x^4 h_{4,5} + x^5 h_{5,5} + x^6 h_{6,5} + x^7 h_{7,5} + x^8 h_{8,5} + x^9 h_{9,5} + x^{10} h_{10,5}) + y^6 (x h_{1,6} + x^2 h_{2,6} + x^3 h_{3,6} + x^4 h_{4,6} + x^5 h_{5,6} + x^6 h_{6,6} + x^7 h_{7,6} + x^8 h_{8,6} + x^9 h_{9,6} + x^{10} h_{10,6} + x^{11} h_{11,6} + x^{12} h_{12,6} + x^{13} h_{13,6} + x^{14} h_{14,6} + x^{15} h_{15,6}) + y^7 (x h_{1,7} + x^2 h_{2,7} + x^3 h_{3,7} + x^4 h_{4,7} + x^5 h_{5,7} + x^6 h_{6,7} + x^7 h_{7,7} + x^8 h_{8,7} + x^9 h_{9,7} + x^{10} h_{10,7} + x^{11} h_{11,7} + x^{12} h_{12,7} + x^{13} h_{13,7} + x^{14} h_{14,7} + x^{15} h_{15,7} + x^{16} h_{16,7} + x^{17} h_{17,7} + x^{18} h_{18,7} + x^{19} h_{19,7} + x^{20} h_{20,7} + x^{21} h_{21,7}) + y^8 (x h_{1,8} + x^2 h_{2,8} + x^3 h_{3,8} + x^4 h_{4,8} + x^5 h_{5,8} + x^6 h_{6,8} + x^7 h_{7,8} + x^8 h_{8,8} + x^9 h_{9,8} + x^{10} h_{10,8} + x^{11} h_{11,8} + x^{12} h_{12,8} + x^{13} h_{13,8} + x^{14} h_{14,8} + x^{15} h_{15,8} + x^{16} h_{16,8} + x^{17} h_{17,8} + x^{18} h_{18,8} + x^{19} h_{19,8} + x^{20} h_{20,8} + x^{21} h_{21,8} + x^{22} h_{22,8} + x^{23} h_{23,8} + x^{24} h_{24,8} + x^{25} h_{25,8} + x^{26} h_{26,8} + x^{27} h_{27,8} + x^{28} h_{28,8})$$

A continuación, la definición y computo de la expresión A:

```
A = (Sum_{n=1}^{maxV} 1/n c[x^n, y^n, maxV]) /. (y^i_ /; i > maxV → 0);
Collect[A, y]
```

$$y + y^2\left(\frac{1}{2} + x\,h_{1,2}\right) + y^3\left(\frac{1}{3} + x\,h_{1,3} + x^2\,h_{2,3} + x^3\,h_{3,3}\right) +$$
$$y^4\left(\frac{1}{4} + \frac{1}{2}\,x^2\,h_{1,2} + x\,h_{1,4} + x^2\,h_{2,4} + x^3\,h_{3,4} + x^4\,h_{4,4} + x^5\,h_{5,4} + x^6\,h_{6,4}\right) +$$
$$y^5\left(\frac{1}{5} + x\,h_{1,5} + x^2\,h_{2,5} + x^3\,h_{3,5} + x^4\,h_{4,5} + x^5\,h_{5,5} + x^6\,h_{6,5} + x^7\,h_{7,5} + x^8\,h_{8,5} + x^9\,h_{9,5} + x^{10}\,h_{10,5}\right) +$$
$$y^6\left(\frac{1}{6} + \frac{1}{3}\,x^3\,h_{1,2} + x\,h_{1,6} + x^2\,h_{2,6} + \frac{1}{2}\,(x^2\,h_{1,3} + x^4\,h_{2,3} + x^6\,h_{3,3}) + x^3\,h_{3,6} + x^4\,h_{4,6} + x^5\,h_{5,6} +\right.$$
$$\left. x^6\,h_{6,6} + x^7\,h_{7,6} + x^8\,h_{8,6} + x^9\,h_{9,6} + x^{10}\,h_{10,6} + x^{11}\,h_{11,6} + x^{12}\,h_{12,6} + x^{13}\,h_{13,6} + x^{14}\,h_{14,6} + x^{15}\,h_{15,6}\right) +$$
$$y^7\left(\frac{1}{7} + x\,h_{1,7} + x^2\,h_{2,7} + x^3\,h_{3,7} + x^4\,h_{4,7} + x^5\,h_{5,7} + x^6\,h_{6,7} + x^7\,h_{7,7} + x^8\,h_{8,7} + x^9\,h_{9,7} + x^{10}\,h_{10,7} +\right.$$
$$x^{11}\,h_{11,7} + x^{12}\,h_{12,7} + x^{13}\,h_{13,7} + x^{14}\,h_{14,7} + x^{15}\,h_{15,7} + x^{16}\,h_{16,7} + x^{17}\,h_{17,7} + x^{18}\,h_{18,7} + x^{19}\,h_{19,7} +$$
$$\left. x^{20}\,h_{20,7} + x^{21}\,h_{21,7}\right) +$$
$$y^8\left(\frac{1}{8} + \frac{1}{4}\,x^4\,h_{1,2} + x\,h_{1,8} + x^2\,h_{2,8} + x^3\,h_{3,8} + x^4\,h_{4,8} + x^5\,h_{5,8} +\right.$$
$$\frac{1}{2}\,(x^2\,h_{1,4} + x^4\,h_{2,4} + x^6\,h_{3,4} + x^8\,h_{4,4} + x^{10}\,h_{5,4} + x^{12}\,h_{6,4}) + x^6\,h_{6,8} + x^7\,h_{7,8} + x^8\,h_{8,8} + x^9\,h_{9,8} +$$
$$x^{10}\,h_{10,8} + x^{11}\,h_{11,8} + x^{12}\,h_{12,8} + x^{13}\,h_{13,8} + x^{14}\,h_{14,8} + x^{15}\,h_{15,8} + x^{16}\,h_{16,8} + x^{17}\,h_{17,8} + x^{18}\,h_{18,8} +$$
$$\left. x^{19}\,h_{19,8} + x^{20}\,h_{20,8} + x^{21}\,h_{21,8} + x^{22}\,h_{22,8} + x^{23}\,h_{23,8} + x^{24}\,h_{24,8} + x^{25}\,h_{25,8} + x^{26}\,h_{26,8} + x^{27}\,h_{27,8} + x^{28}\,h_{28,8}\right)$$

La función general de librería, `Expand` resulta muy lenta para nuestra aplicación. Con la redefinición específica de dicha función se consigue un incremento significativo en su cómputo como muestran las activaciones siguientes, tomadas de su activación en una PC a 700MHz.

```
myExpand[f_, 1, _] := f
myExpand[f_, n_, maxV_] :=
 myExpand[f, n, maxV] = Expand[f myExpand[f, n - 1, maxV]] /. (y^i_ /; i > maxV → 0)
```

```
Timing[Collect[Expand[u^maxV], y] /. (y^i_ /; i > maxV → 0)]
```

```
{34.710000000000036 Second, y^8}
```

```
Timing[myExpand[u, maxV, maxV]]
```

```
{0.4400000000005093 Second, y^8}
```

Continuando ahora con la definición de la función B:

```
u = g[x, y, maxV];
```

$$B = \sum_{n=1}^{maxV} \frac{(-1)^{n+1}}{n}\ \text{myExpand}[u, n, maxV];$$

```
B = (Collect[Expand[B] /. (y^i_ /; i > maxV → 0), y])
```

$$y + \left(\frac{1}{2} + x\right) y^2 + \left(\frac{1}{3} + x^2 + x^3\right) y^3 + \left(\frac{1}{4} + \frac{x^2}{2} + 2x^3 + 2x^4 + x^5 + x^6\right) y^4 +$$
$$\left(\frac{1}{5} + 3x^4 + 5x^5 + 5x^6 + 4x^7 + 2x^8 + x^9 + x^{10}\right) y^5 +$$
$$\left(\frac{1}{6} + \frac{x^3}{3} + \frac{x^4}{2} + 6x^5 + \frac{27x^6}{2} + 19x^7 + 22x^8 + 20x^9 + 14x^{10} + 9x^{11} + 5x^{12} + 2x^{13} + x^{14} + x^{15}\right) y^6 +$$
$$\left(\frac{1}{7} + 11x^6 + 33x^7 + 67x^8 + 107x^9 + 132x^{10} + 138x^{11} + 126x^{12} + 95x^{13} + 64x^{14} + 40x^{15} + 21x^{16} + 10x^{17} + 5x^{18} + 2x^{19} + x^{20} + x^{21}\right) y^7 +$$
$$\left(\frac{1}{8} + \frac{x^4}{4} + x^6 + 23x^7 + 90x^8 + 236x^9 + \frac{973x^{10}}{2} + 814x^{11} + \frac{2339x^{12}}{2} + 1454x^{13} + 1579x^{14} + 1515x^{15} + 1290x^{16} + 970x^{17} + 658x^{18} + 400x^{19} + 220x^{20} + 114x^{21} + 56x^{22} + 24x^{23} + 11x^{24} + 5x^{25} + 2x^{26} + x^{27} + x^{28}\right) y^8$$

y resolviendo el sistema A=B se tiene:

```
Bc = Coefficient[B, Table[x^i y^j, {i, Binomial[maxV, 2]}, {j, maxV}]];
Ac = Coefficient[A, Table[x^i y^j, {i, Binomial[maxV, 2]}, {j, maxV}]];
t = Solve[Ac == Bc, Drop[Variables[A], 2]]
c[x, y, maxV] /. t
```

```
{{h1,2 → 1, h1,3 → 0, h1,4 → 0, h1,5 → 0, h1,6 → 0, h1,7 → 0, h1,8 → 0, h2,3 → 1, h2,4 → 0,
  h2,5 → 0, h2,6 → 0, h2,7 → 0, h2,8 → 0, h3,3 → 1, h3,4 → 2, h3,5 → 0, h3,6 → 0, h3,7 → 0,
  h3,8 → 0, h4,4 → 2, h4,5 → 3, h4,6 → 0, h4,7 → 0, h4,8 → 0, h5,4 → 1, h5,5 → 5, h5,6 → 6,
  h5,7 → 0, h5,8 → 0, h6,4 → 1, h6,5 → 5, h6,6 → 13, h6,7 → 11, h6,8 → 0, h7,5 → 4, h7,6 → 19,
  h7,7 → 33, h7,8 → 23, h8,5 → 2, h8,6 → 22, h8,7 → 67, h8,8 → 89, h9,5 → 1, h9,6 → 20, h9,7 → 107,
  h9,8 → 236, h10,5 → 1, h10,6 → 14, h10,7 → 132, h10,8 → 486, h11,6 → 9, h11,7 → 138, h11,8 → 814,
  h12,6 → 5, h12,7 → 126, h12,8 → 1169, h13,6 → 2, h13,7 → 95, h13,8 → 1454, h14,6 → 1, h14,7 → 64,
  h14,8 → 1579, h15,6 → 1, h15,7 → 40, h15,8 → 1515, h16,7 → 21, h16,8 → 1290, h17,7 → 10,
  h17,8 → 970, h18,7 → 5, h18,8 → 658, h19,7 → 2, h19,8 → 400, h20,7 → 1, h20,8 → 220, h21,7 → 1,
  h21,8 → 114, h22,8 → 56, h23,8 → 24, h24,8 → 11, h25,8 → 5, h26,8 → 2, h27,8 → 1, h28,8 → 1}}

{y + x y^2 + (x^2 + x^3) y^3 + (2 x^3 + 2 x^4 + x^5 + x^6) y^4 + (3 x^4 + 5 x^5 + 5 x^6 + 4 x^7 + 2 x^8 + x^9 + x^10) y^5 +
  (6 x^5 + 13 x^6 + 19 x^7 + 22 x^8 + 20 x^9 + 14 x^10 + 9 x^11 + 5 x^12 + 2 x^13 + x^14 + x^15) y^6 +
  (11 x^6 + 33 x^7 + 67 x^8 + 107 x^9 + 132 x^10 + 138 x^11 + 126 x^12 + 95 x^13 + 64 x^14 + 40 x^15 + 21 x^16 +
     10 x^17 + 5 x^18 + 2 x^19 + x^20 + x^21) y^7 +
  (23 x^7 + 89 x^8 + 236 x^9 + 486 x^10 + 814 x^11 + 1169 x^12 + 1454 x^13 + 1579 x^14 + 1515 x^15 + 1290 x^16 +
     970 x^17 + 658 x^18 + 400 x^19 + 220 x^20 + 114 x^21 + 56 x^22 + 24 x^23 + 11 x^24 + 5 x^25 + 2 x^26 + x^27 + x^28) y^8}
```

Finalmente, encapsulando todo el procedimiento anterior, el siguiente código nos permite obtener la tabla sinóptica presentada al final de esta sección, en donde es posible leer, por ejemplo, que, de 32768 grafos con seis vértices, solo 156 son no-isomorfos y, de estos, solo 112 son conexos, hallándose entre éstos 6 árboles.

```
gr[v_, e_] := Binomial[Binomial[v, 2], e]

allGraphs[v_] := Sum[gr[v, e], {e, 0, Binomial[v, 2]}]
```

```
gC[x_, y_, maxVer_] := gC[x, y, maxVer] = Module[{n, A, B, Ac, Bc, i, j, t},
   A = Sum[1/n c[x^n, y^n, maxVer], {n, 1, maxVer}];
   A = (A /. (y^i_ /; i > maxVer -> 0));
   B = Sum[(-1)^(n+1)/n myExpand[g[x, y, maxVer], n, maxVer], {n, 1, maxVer}];
   B = (Collect[B, y] /. (y^i_ /; i > maxVer -> 0));
   Bc = Coefficient[B, Table[x^i y^j, {i, Binomial[maxVer, 2]}, {j, maxVer}]];
   Ac = Coefficient[A, Table[x^i y^j, {i, Binomial[maxVer, 2]}, {j, maxVer}]];
   t = Solve[Ac == Bc, Drop[Variables[A], 2]];
   c[x, y, maxVer] /. t]

numConnectedGraphs[v_, e_] :=
 numConnectedGraphs[v, e] = Coefficient[Coefficient[gC[x, y, v], y, v], x, e]

allDiffConnectedGraphs[v_] := Sum[numConnectedGraphs[v, e], {e, 1, Binomial[v,2]}]

numTrees[v_] := numConnectedGraphs[v, v - 1]

Table[{v, allGraphs[v], allDiffGraphs[v], allDiffConnectedGraphs[v], numTrees[v]},
  {v, 2, 10}] // TableForm
```

2	2	2	1	1
3	8	4	2	1
4	64	11	6	2
5	1024	34	21	3
6	32768	156	112	6
7	2097152	1044	853	11
8	268435456	12346	11117	23
9	68719476736	274668	261080	47
10	35184372088832	12005168	11716571	106

Las secuencias correspondientes a las dos últimas columnas son la M1657 y la M0791 en [11]

3.- Generación de todos los grafos conexos

En esta sección consideramos la generación de todos los grafos no-isomorfos conexos de v vértices. Dos grafos isomorfos tienen matrices de adyacencia similares (es decir, existe una matriz de permutaciones bajo la cual son similares) y, en consecuencia, poseen el mismo polinomio característico det (xI-M(G)), pero su cómputo resulta muy tardado; en consecuencia no resulta práctico la implementación de este criterio para eliminar todas las posibles parejas. En la sección 1 comentamos el hecho de que la obtención de grafos no-isomorfos a partir de una generación exhaustiva de todos los grafos resulta impráctica. Con este fin, presentamos un algoritmo que incluye la sincronización de dos funciones mutuamente recursivas, ambas bajo aceleración utilizando programación dinámica [14].

Primeramente consideremos una función estructuralmente análoga a `lambdas` que calcula todos los subconjuntos no-vacíos del conjunto {1, 2, ..., n}.

```
subSetsK[_, 0] := {{}}
subSetsK[0, _] := {}

subSetsK[n_, k_] := {} /; k > n
subSetsK[n_, k_] :=
 subSetsK[n, k] = Module[{s1 = subSetsK[n - 1, k], s2 = subSetsK[n - 1, k - 1]},
        Join[s1, Map[Append[#, n] &, s2]]]

subsets[n_] := Flatten[Table[subSetsK[n, k], {k, n}], 1]
```

Por ejemplo

```
subsets[5]

{{1}, {2}, {3}, {4}, {5}, {1, 2}, {1, 3}, {2, 3}, {1, 4}, {2, 4}, {3, 4},
 {1, 5}, {2, 5}, {3, 5}, {4, 5}, {1, 2, 3}, {1, 2, 4}, {1, 3, 4}, {2, 3, 4},
 {1, 2, 5}, {1, 3, 5}, {2, 3, 5}, {1, 4, 5}, {2, 4, 5}, {3, 4, 5}, {1, 2, 3, 4},
 {1, 2, 3, 5}, {1, 2, 4, 5}, {1, 3, 4, 5}, {2, 3, 4, 5}, {1, 2, 3, 4, 5}}
```

La siguiente función `allGraphs` construye, a partir de una lista de todos los grafos no-isomorfos de v-1 vértices, una nueva lista de candidatos con v vértices al añadir todas las posibles aristas a un nuevo vértice. La función `allDiffGraphs` construye, a partir de filtrar la lista de todos los grafos, una lista que incluya solamente los no-isomorfos. Dicho proceso hace uso de una función de ordenación canónica `ord` para la representación como lista de parejas de un grafo, en donde los vértices se identifican con los primeros n enteros positivos.

```
ord[lis_] := Sort[Sort /@ lis]

allGraphs[2] := {{{1, 2}}}
allGraphs[v_] := allGraphs[v] = Module[{ans = {}, c},
   Map[(c = #;
      Map[AppendTo[ans, Partition[Flatten[{c, Outer[List, #, {v}]}], 2]] &,
       subsets[v - 1]]) &, allDiffGraphs[v - 1]];
   ans]

allDiffGraphs[2] := {{{1, 2}}}
allDiffGraphs[v_] :=
 allDiffGraphs[v] = Module[{g = allGraphs[v], t, e, ans = {}},
   t = Map[Thread[Range[v] → #] &, Permutations[Range[v]]];
   While[g ≠ {},
    e = ord[First[g] /. #] & /@ t;
    AppendTo[ans, First[g]];
    g = ord /@ Complement[g, e]];
   ans]
```

Por ejemplo,

```
allGraphs[3]

{{{1, 2}, {1, 3}}, {{1, 2}, {2, 3}}, {{1, 2}, {1, 3}, {2, 3}}}

allDiffGraphs[3]

{{{1, 2}, {1, 3}}, {{1, 2}, {1, 3}, {2, 3}}}
```

La aplicación de las funciones anteriores nos permite encontrar la estructura fina de nuestros grafos. Por ejemplo, en la Figura 1 se muestran, de izquierda a derecha y de arriba abajo, las graficas correspondientes al número de grafos conexos no-isomorfos de 3, 4, 5 y 6 vértices y de aristas variables, superpuestas con las correspondientes biconexas. Supongamos que la notación (v, c, b) significa que, de los c grafos conexos de v vértices, solo existen b biconexos. Con ayuda de nuestro desarrollo podemos deducir entonces que (3, 2, 1), (4, 6, 3), (5, 21, 10) y (6, 112, 56) (ver sec. M2873 en [11]).

Finalmente, la siguiente función puede ser utilizada para dibujar las anteriores generaciones.

```
showGraph[listOfGraphs_, v_] := Module[{c, p, t},
  c = Table[{Cos[t], Sin[t]}, {t, 0, 2 π - 0.001, 2 π / v}];
  p = Disk[#, 0.2] & /@ c;
  Show[Graphics[{p, Line /@ (# /. Thread[Range[v] → c])}],
     DisplayFunction → Identity, AspectRatio → Automatic] & /@ listOfGraphs]
```

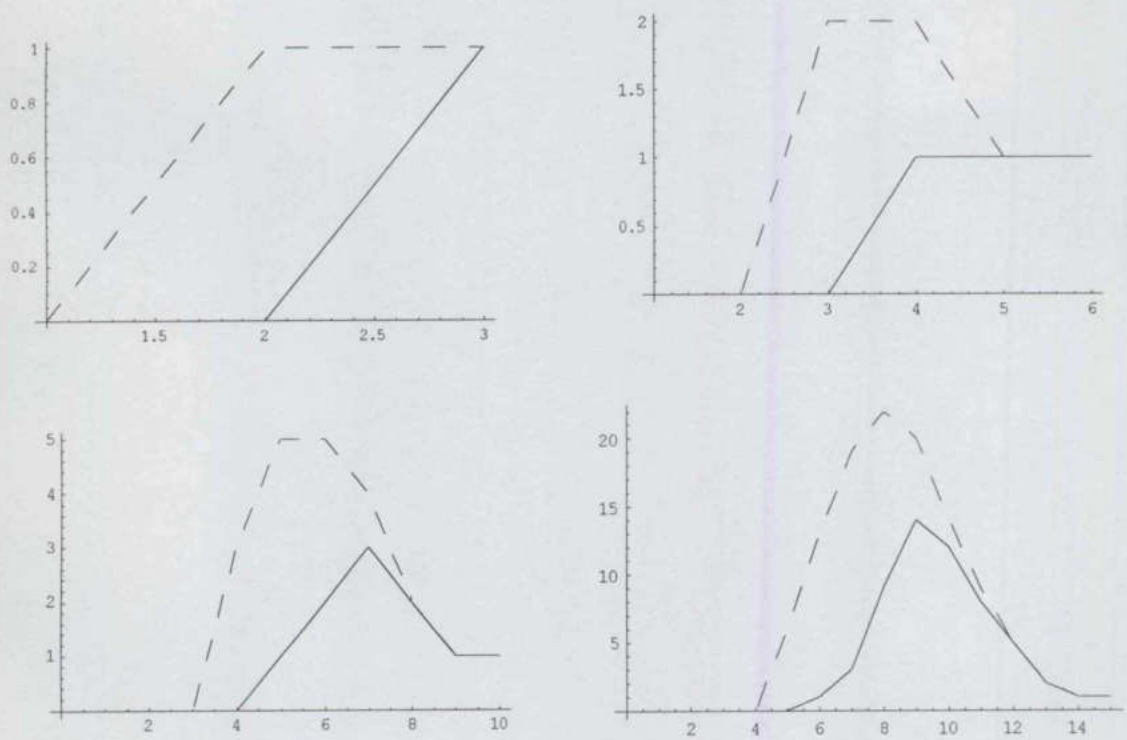
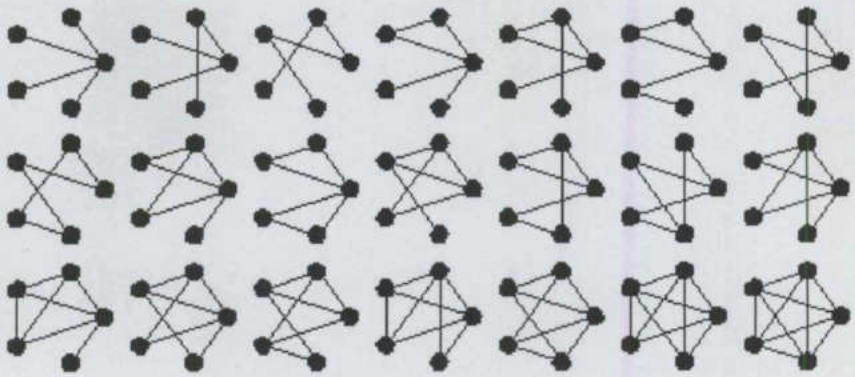


Figura 1

Por ejemplo:

```
v = 5;
u = allDiffGraphs[v];
Show[GraphicsArray[Partition[showGraph[u, v], 7]]];
```



4.- Conclusión

Un aspecto esencial de la teoría combinatoria y de la ciencia de la computación es la enumeración y construcción sistemática de estructuras matemáticas, que a su vez conduce a recurrencias y a comportamientos asintóticos y al diseño de algoritmos óptimos. Análogamente, la generación de estructuras no-isomorfas, de valor intrínseco en la compilación de catálogos, constituye un punto de partida para probar o refutar hipótesis generales y establecer conjeturas basadas en ellas. Las matemáticas experimentales como ayuda en el descubrimiento de conocimiento genuino matemático ha tenido un auge en sincronía con el desarrollo del poder de cómputo y este artículo plantea una contribución en esta dirección al presentar implementaciones bajo el paradigma funcional de formulas de enumeración de grafos conexos no-isomorfos.

La búsqueda de certificados e invariantes es importante para la generación y su incorporación futura resultará valiosa para los problemas que se mencionaron en este trabajo. Además de esto, como trabajos futuros podemos mencionar la clasificación de los grafos planares no-isomorfos, en donde la propiedad de no-separabilidad o biconectividad simplifica las estructuras a considerar y constituye un paso natural para extender nuestro trabajo.

5.-Referencias

[1] J. Backus. *Can Programming be liberated from the von Neumann style? A functional style and its algebra of programs.* Communications of the ACM 21 (8): 613-641, 1978

[2] C. Berge. *Principles of Combinatorics.* Academic Press, 1971

[3] F. Harary. *The Number of Linear, Directed, Rooted and Connected Graphs.* Trans. Amer. Math. Soc. 78: 445-463, 1955

[4] F. Harary, E.M. Palmer. *Graphical Enumeration.* Academic Press, 1973

[5] A. Hayes. *Experiments in efficient programming.* The Mathematica Journal 5(1): 24-31, 1995

[6] P. Henderson. *Functional Programming, Formal Specification and Rapid Prototyping*. IEEE Transactions on Software Engineering SE12 (2): 241-250, Febrero 1986

[7] R.E. Maeder. *The Mathematica Programmer II*. Academic Press, 1996

[8] R.E. Maeder. *Higher Order Functions*. The Mathematica Journal 5(3): 61-67, 1997

[9] R. Otter. *The Number of Trees*. Ann. of Math. 49: 583-599, 1948

[10] R.W. Robinson. Enumeration of non-separable graphs. J. Combin. Theory 9: 327-356, 1970

[11] N.J.A. Sloane, S. Plouffe. *The Encyclopedia of Integer Sequences*. Academic Press, 1995

[12] R. P. Stanley. *Generating Functions in Studies in Combinatorics*; G.C. Rota ed. The Math. Assoc. of America , 1990

[13] S.S. Skiena. *Implementing Discrete Mathematics*. Addison-Wesley, Reading, MA, 1990

[14] F. Rabbi, G. Lapalme. *Algorithms, a Functional Programming Approach*. Addison-Wesley, Reading MA, 1999

[15] S. Wolfram. *The Mathematica Book* (cuarta edición), Cambridge University Press, 1999

DÉCIMO SEXTO COLOQUIO DE TEORÍA DE LAS GRÁFICAS, COMBINATORIA Y SUS APLICACIONES

SAN LUIS POTOSÍ, SLP
25 DE FEBRERO AL 3 DE MARZO, 2001

El Comité Organizador extiende la presente
CONSTANCIA a

MA. ELENA VÁZQUEZ HUERTA

por su participación con la ponencia
Técnicas de Programación Funcional en la Enumeración y Generación de Grafos no-Isomorfos

en este coloquio, celebrado del 25 de febrero al 3 de marzo de 2001
en la ciudad de San Luis Potosí, S.L.P.

Víctor Neumann Lara

ANEXO II
BIOGRAFÍAS

Niels Henrik Abel

(1802 - 1829)

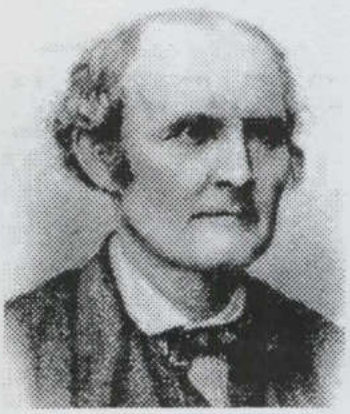
Matemático Noruego quien probó la imposibilidad de resolver algebraicamente ecuaciones de quinto grado. Abel publicó en 1823 escritos de ecuaciones funcionales e integrales . En esto Abel dio la primera solución de una ecuación integral. En 1824 probó que era imposible resolver algebraicamente ecuaciones de quinto grado y de su propio costo realizó publicaciones con la esperanza de obtener reconocimiento por su trabajo. Se distinguió por su contribución en el campo de las funciones e integrales elípticas, donde estudió las integrales llamadas abelianas e introdujo el concepto de género de una función algebraica. Demostró la irresolubilidad por radicales de ecuaciones de grado superior al cuarto. En su honor, se utiliza la expresión "grupo abeliano" con el mismo significado que "grupo conmutativo". El teorema del binomio fue formulado por Isaac Newton y el matemático suizo Leonhard Euler, pero Abel le dio una generalización más amplia, incluyendo los casos de exponentes irracionales e imaginarios.

Augustin Louis Cauchy

(1789 – 1857)

Matemático francés, considerado uno de los impulsores del análisis en el siglo XIX. Agustín Louis Cauchy pionero en el análisis y la teoría de permutación de grupos. También investigó la convergencia y la divergencia de las series infinitas, ecuaciones diferenciales,

determinantes, probabilidad y física matemática. Numerosos términos matemáticos llevan su nombre: el teorema integral de Cauchy, la teoría de las funciones complejas, las ecuaciones de Cauchy-Riemann y Secuencias de Cauchy. Probó (1811) que los ángulos de un Poliedro convexo estan determinados por sus caras (las superficies planas limitan a un sólido geométrico). Cauchy fue el primero en hacer un estudio cuidadoso de las condiciones para la convergencia de las series en el infinito; también dio una definición rigurosa de un integral independiente del proceso de diferenciación y desarrolló la teoría matemática de elasticidad. Sus textos *Cours d'analyse* (Curso en Análisis, 1821) y el *4-volume Exercises dánalyse et de physique mathematique* (Ejercicios en Análisis y en Físicas Matemáticas, 1840-47) fueron muy influyentes. Cauchy confirmó la existencia de funciones elípticas recurrentes, dio el primer movimiento a la teoría general de funciones, y puso la base para el tratamiento moderno de la convergencia de la serie infinita.



Arthur Cayley

(1821 - 1895)

Jurista y matemático inglés, propició con sus investigaciones el origen y desarrollo posterior del cálculo matricial. Junto con su coetáneo Hamilton, encabezaron la prestigiosa escuela de matemáticos ingleses del siglo XIX. Creador de la teoría de los invariantes y de la primera definición abstracta de un grupo finito. Introdujo la métrica proyectiva, formulando conceptos geométricos, luego desarrollados por Klein. En 1857 desarrolló el álgebra de matrices. Es considerado como el tercer escritor más prolífico de matemáticas, siendo sólo superado por Euler y Cauchy. Hizo importantes contribuciones en la Teoría de curvas y superficies, en la geometría analítica, en la teroria de los determinantes y el desarrollo de la teoría de los invariantes. Sus trabajos en geometría cuatridimensional, proporcionaron a los físicos del siglo XX, especialmente a Albert Einstein, la estructura para desarrollar la teoría de la relatividad.

Leonard Euler

(1707-1783)

Matemático suizo, cuyos trabajos más importantes se centraron en el campo de las matemáticas puras, campo de estudio que ayudó a fundar. Fue el matemático más prolífico en la historia. Sus 866 libros y artículos representan aproximadamente una tercera parte del cuerpo entero de la investigación en las matemáticas, física teórica, y la ingeniería mecánica, todos estos publicados entre 1726 y 1800.

En su Introducción al análisis de los infinitos (1748), Euler realizó el primer tratamiento analítico completo del álgebra, la teoría de ecuaciones, la trigonometría y la geometría analítica. En esta obra trató el desarrollo de series de funciones y formuló la regla por la que sólo las series convergentes infinitas pueden ser evaluadas adecuadamente. También abordó las superficies tridimensionales y demostró que las secciones cónicas se representan mediante la ecuación general de segundo grado en dos dimensiones. Otras obras trataban del cálculo (incluido el cálculo de variaciones), la teoría de números, números imaginarios y álgebra determinada e indeterminada. En la matemática pura, él integró el cálculo diferencial de Leibniz y el método de Newton de fluxiones dentro del análisis matemático; refinó la noción de función; hizo común muchas notaciones matemáticas, incluso *e, i,* el símbolo de *pi*, y el símbolo de sigma; y derribó los fundamentos de la teoría de funciones especiales, incluyendo las funciones transcendentales beta y gamma. También trabajó en los orígenes del cálculo de variaciones, pero detuvo su trabajo en deferencia a J. L. Lagrange. Fue un pionero en el campo de la topología e hizo la teoría de los números entrar en la ciencia, declaró el teorema del número primo y la ley de reciprocidad del bicuadrático. Realizó también aportaciones a la astronomía, la mecánica, la óptica y la acústica. En la física articuló la dinámica de Newton y derribó los fundamentos de la mecánica analítica, sobre todo en su Teoría de los Movimientos de Cuerpos Rígidos (1765). Como su maestro Johann Bernoulli, elaboró mecánicas contínuas, pero también puso adelante la teoría cinética de gases con el modelo molecular. Con Alexis Clairaut estudió la teoría lunar. También hizo investigación en el principio de elasticidad, acústica, la teoría de la onda de luz, y la hidromecánica de los barcos. Entre sus obras famosas se encuentran

Instituciones del cálculo diferencial (1755), Instituciones del cálculo integral (1768-1770) e Introducción al álgebra (1770).

Evariste Galois

(1811-1832)

Matemático francés, a quien suele otorgarse el título de fundador de la teoría de grupos. La resolución de ecuaciones fue el problema para el que Galois desarrolló la teoría de grupos. Los métodos generales de resolución de ecuaciones considerados aceptables deberían basarse únicamente en las operaciones de adición, sustracción, multiplicación, división y extracción de raíces, y tendrían que ser aplicables a cualquier ecuación de grado n, siendo n la máxima potencia a que viene elevada la incógnita. Galois demostró que no existe ningún método así a partir del caso n=5. Cada ecuación de grado n tiene asociado el grupo S(n) o algún subgrupo de S(n); hoy, al grupo asociado a una ecuación se le llama grupo de Galois de la ecuación. Galois demostró que solamente serán resolubles por medios aritméticos y extracción de raíces aquellas ecuaciones cuyo grupo de Galois sea soluble, noción definida por él. Un grupo se llama soluble cuando genera una serie de subgrupos normales maximales cuyos factores de composición (que se determina a partir de los números de elementos del grupo paterno y de los subgrupos) sean todos ellos primos. Los factores de composición generados por S(3) y su serie de subgrupos normales son todos números primos; por ello todas las ecuaciones de tercer grado son resolubles. Sin embargo, cuando n es mayor o igual que 5, puede demostrarse que el subgrupo normal maximal de A(n) es el grupo identidad, que contiene únicamente la permutación identidad. Como A(n) es el subgrupo normal maximal de S(n) los factores de composición de S(n) no son todos primos cuando n es mayor o igual que 5. Hay pues ecuaciones de grado 5 o superior no resolubles por los métodos permisibles. Es Galois quién muestra una idea clara de la teoría general con las nociones de subgrupo y de isomorfismo. En 1831 puso de manifiesto el valor del concepto de grupo, desarrollando la teoría de grupos algebraicos y la aplicación de éstos a la resolución de ecuaciones algebraicas. Con el concepto de grupo se abren las puertas a las más importantes ideas

matemáticas del mundo contemporáneo. La aportación de Evariste Galois a las matemáticas no es sencilla de entender por su complejidad y la novedad, incluso para los tiempos actuales, que encierra en su interior. No fue completamente comprendida por los matemáticos de su época, algunos sencillamente la ignoraron, y hasta finales del siglo XIX no se descubrió su profundidad y alcance. Para hacernos una idea de su importancia baste decir que las estructuras algebráicas llamadas Grupos de Galois son utilizadas asiduamente en los tiempos actuales en ramas de la técnica como la Criptografía, la Informática o las Telecomunicaciones.

Johann Karl Friedrich Gauss.

(1777-1855)

Matemático Alemán. Gauss formuló el método de los menos - cuadrados y una conjetura en la distribución de números primos entre todos los números; el más reciente fue probado por Jacques Hadamard en 1896. En 1795 Gauss descubrió el teorema fundamental de residuos cuadráticos, que tratan del concepto de congruencia en la teoría del número. En 1796 hizo su primera marca como un matemático serio por probar la posibilidad de construir un polígono regular de 17 lados usando sólo una regla y un compás. En 1801 publicó *Disquisitiones Arithmeticae*, su principal trabajo y uno de los más importantes en la historia de las matemáticas. Esta obra cubre temas de teoría de números, análisis matemático, teoría de probabilidades, geometría, fisicamatemática, astronomía y geodesia. El volumen XII contiene un Atlas magnético terrestre compilado y editado por Gauss, Weber y Goldschmidt Este libro fija bases para investigaciones futuras dándole un mayor reconocimiento entre los matemáticos de su tiempo. También en 1801, Gauss determinó correctamente las órbitas de los asteroides Ceres y Pallas usando el método de mínimos cuadrados que él mismo desarrolló en 1794, y que publicara en 1809 en *Theoria Motus Corporum Coelestium*. En probabilidad mostró que ésta puede representarse por una curva en forma de campana (distribución gaussiana), que es la base en la distribución estadística de datos. Contribuyó también al estudio del electromagnetismo: la teoría moderna del

potencial. Estudió el magnetismo terrestre y desarrolló, con Weber, la telegrafía eléctrica. En su trabajo teórico en topografía, Gauss desarrolló resultados que requirió de estadísticas y geometría diferencial. Hasta la fecha se han publicado 12 volúmenes de su trabajo. Gauss es considerado uno de los matemáticos más grandes de todos los tiempos: el Príncipe de las matemáticas.

Marie Ennemond Camille Jordan

(1838 – 1922)

Matemático francés y uno de los principales responsables de la consolidación de la teoría de los grupos. Jordan absorbió las ideas de Galois, desarrollando sobre esta base una rigurosa teoría sobre los grupos finitos e infinitos. Puso en conexión los grupos de permutación y el estudio de Galois acerca de la permutación de las raíces de las ecuaciones con el problema de la resolución de las ecuaciones polinómicas. Jordan publicó una obra ya clásica sobre la teoría de los grupos (1870). Impulsó el avance de los grupos simétricos y redujo las ecuaciones diferenciales lineales de orden n a un problema teórico de grupos. Generalizó los trabajos de Hermite sobre la teoría de las formas cuadráticas con coeficientes integrales. Jordan inspiró a Klein y Lie en la prosecución de nuevas investigaciones sobre la teoría de los grupos. La topología fue otra de las materias que captaron el interés de Jordan, terreno donde elaboró una topología homológica o combinatoria al investigar las simetrías de los poliedros.

Felix Klein

(1849 – 1925)

Matemático alemán. Introdujo una nueva concepción de la geometría como estudio de las propiedades de un espacio que son invariantes según un determinado grupo de transformaciones. El estudio de las transformaciones geométricas y su relación con las matrices se debe principalmente a este matemático. La importancia de su labor científica es indiscutible: en la tesis que presentó como docente -y que ha pasado a llamarse "el Programa de Erlanger". Sistematizó la clasificación de las distintas geometrías, con ayuda de la teoría de los grupos y de los invariantes de las transformaciones geométricas. Sus investigaciones sobre la teoría de las funciones son de suma importancia; sobresale su estudio de la función llamada modular, que es una generalización de las funciones elípticas. Combinando el desarrollo alcanzado por las geometrías no euclideanas y la geometría proyectiva con la teoría de los invariantes y la teoría de grupos, Klein expone en su Programa de Erlangen, en 1872, mediante grupo y subgrupos, una sistematización y jerarquización de todas las geometrías, concibiendo como objeto de cada geometría el descubrimiento de propiedades invariantes respecto de un determinado grupo de transformaciones y considerando cada geometría como subgeometría de otra a la que se adjunta cierta figura básica, que debe quedar invariante. Más tarde, en 1884, ofreció un ejemplo de dos grupos isomorfos: el de las rotaciones del icosaedro regular y el de la ecuación de quinto grado.

Joseph-Louis Lagrange

(1736 – 1813)

Astrónomo y matemático francés de origen italiano Fué uno de los científicos matemáticos y fisicos más importantes de finales del siglo XVIII. Inventó y maduró el cálculo de variaciones y más tarde lo aplicó a una nueva disciplina la Mecánica Celestial, sobre todo al hallazgo de mejores soluciones al Problema de tres cuerpos. También contribuyó significativamente con la solución numérica y algebraica de

ecuaciones y con la teoría numérica. En su clásica *Mecanique Analytique* (mecánicas Analíticas, 1788), transformó la mecánicas en una rama del análisis matemático. El tratado resumió los principales resultados sobre mecánicas que se saben del siglo XVIII y es notable por su uso de la teoría de ecuaciones diferenciales. Otra preocupación central de Lagrange fueron los fundamentos de cálculo. En un libro de 1797 él enfatizó la importancia de la serie de Taylor y el concepto de función. Su búsqueda por rigurosas fundaciones y generalizaciones le pone la base a Augustin Cauchy, Niels Henrik Abel, y Karl Weierstrass en el siguiente siglo. Demostró que tres cuerpos pueden estar situados en los vértices de un triángulo equilátero que gira en su plano. Si uno de los cuerpo tiene masa suficiente comparado con los otros dos, entonces la configuración triangular es estable aparentemente.

August Möbius

(1790 - 1868)

Matemático Alemán, autor de *El cálculo baricéntrico* (1827), obra que innovó la geometría proyectiva, aportando progresos en la orientación de los segmentos, áreas y volúmenes. Dio con el primer ejemplo de una superficie de una sola cara, la «banda de Möbius», que se consigue con una cinta plana, haciendo girar uno de sus extremos 180° y uniéndolo seguidamente al otro extremo. Esta banda se ha aplicado en filtro auto limpiante destinado a máquinas de limpieza en seco, que por tener la forma de banda de Mobius facilita el lavado por ambas caras; los artistas gráficos se han valido esta banda tanto para fines publicitarios como artísticos, la superficie de Mobius ha tenido también papel destacado en numerosos cuentos de ciencia ficción, No- died Professor, The Wall of darkness, (julio 1949). Se sabe de mecanógrafos muy rápidos que encontrando fastidioso tener que detenerse a meter en la máquina hojas nuevas en blanco, han optado por utilizar papel en rollo. Si hubieran usado un largo bucle de retorcido habrían podido ademas escribir por ambos lados del papel. Es recordado por haber planteado el problema de los cinco colores, consistente en hallar un mapa donde se requieran menos de cinco colores para identificar adecuadamente la totalidad de los países. Hasta la fecha nadie ha logrado dar con la solución del problema. Al parecer cuatro colores bastan para tal efecto.

Paolo Ruffini

(1765 – 1822)

Médico y matemático italiano. Demostró, aunque deficientemente, la imposibilidad de resolver por radicales la ecuación general de grado superior a cuatro. Es muy conocida su regla para la división de un polinomio en x por el binomio x - a. Ruffini fue el primero que realizó un intento, con éxito parcial (probablemente en 1803 o 1805), de demostrar la imposibilidad de resolver mediante procesos elementales de álgebra las ecuaciones generales de un grado superior a cuatro. Esta formulación, denominada teorema Abel-Ruffini, fue demostrada definitivamente por el matemático noruego Niels Henrik Abel.

Walther Von Dyck

(1856 - 1934)

Matemático Alemán, hizo importantes contribuciones a la teoría de grupos con la publicación de dos papers: *Gruppentheoretische Studien in Mathematische Annalen* el primero en 1882 y el segundo en el siguiente año. Hizo importantes contribuciones a la teoría de funciones, teoría de grupos (donde un resultado fundamental en presentación de grupos es nombrado despues de él), topología y teoría de potencia. Contribuyó significativamente en el teorema Gauss-Bonnet.

REFERENCIAS BIBLIOGRÁFICAS

[1] Aho, Alfred V; Hopcroft John E.; Ullman, *Jeffrey D. Estructura de datos y algoritmos.* Addison Wesley Iberoamericana, 1988.

[2] Anton, Howard. *Introducción al Álgebra Lineal.* 3ª edición, Limusa 1994

[3] Baumslag, Benjamin; Chandler, Bruce. *Theory and Problems of Group Theory.* McGraw Hill, 1968

[4] Cameron, Peter J. *Combinatorial Topics Techniques and Algorithms.* Cambridge University Press, 1994

[5] Cormen, Tomas H.; Leiserson, Charles E.; Rivest, Ronald L. *Introduction to Algorithms.* MIT PRESS, 1990.

[6] O´Connor, J.J.; Robertson, E.F. *The development of group theory* http://www-histori.mcs.st-andrews.ac.uk/History/HistTopics/Development_group_theory.html, May 1996

[7] Foulds, L. R. *Graph Theory applications.* Springer 1992

[8] Fraleigh, John B. *A First Course in Abstract Algebra.* 6th Edition. Addison Wesley, 1987

[9] Grimaldi, Ralph P. *Matematicas Discretas y Combinatoria. Una introducción con aplicaciones.* 3rd Edition.. Addison Wesley 1998

[10] Harary, Frank. *Graph Theory.* Perseus Book, 1969

[11] Hibbard, Allen C.; Levasseur, Kenneth M. *Exploring Abstract Algebra with Mathematica.* Springer Telos 1999

[12] Johnsongaugh, Richard. *Discrete Mathematics.* 4th Edition. Prentice Hall, 1999

[13] Knuth, Donald E. *The Art of Computer Programming. Volume 1. Fundamental Algorithms.* 3rd Edition., Addison Wesley 1997

[14] Kreher, Donald L.; Stinson, Douglas R. *Combinatorial Algorithms. Generation, Enumeration and Search.* CRC Press 1998

[15] Liu, Chung L. *Elementos de Matemáticas Discretas.* McGraw Hill, 1995

[16] López-Blancas, Verónica. *Enumeración de árboles.* Universidad Autónoma de Querétaro Facultad de Informática. 2001

[17] Rangel-Mondragón, Jaime; González-Gutiérrez, Arturo. *Apuntes de Matemáticas Discretas. Un punto de vista algoritmico. Parte V. Grupos y Árboles.* UAQ 1999

[18] Rangel-Mondragón, Jaime; Vázquez-Huerta, Ma. Elena. *Técnicas de Programación Funcional en la Enumeración y Generación de Grafos no-Isomorfos.* XVI Coloquio de Teoría de las Gráficas, Combinatoria y sus aplicaciones. SMM 2001

[19] Rossen, Keneth. *Handbook of Discrete and Combinatorial Mathematics.* CRC Press 2000.

[20] Slomson, Alan. *An Introductory Combinatorics.* Chapman And Hall/ CRC Press, 1991

[21] Wilson, Robin J.; Watkins, John J. *Graph An Introductory Approach.* John Wiley and Sons 1990

[22] Weyl, Hermann. *Lecturas Universitarias, Tomo 8. Antologías de Matemáticas.* Universidad Nacional Autónoma de México, 1971