Universidad Autónoma del Estado de Hidalgo
Instituto de Ciencias Básicas e Ingeniería
Centro de Investigación en Matemáticas

Implementación de Algoritmos en Teoría de Gráficas

Tesis que para obtener el título de

Licenciado en Matemáticas Aplicadas

presenta

Fidel Barrera Cruz

bajo la dirección de

Dr. Roberto López Hernández

y

Dr. Rubén A. Martínez Avendaño

PACHUCA, HIDALGO, SEPTIEMBRE DE 2007

Resumen

En este trabajo de tesis se abordan algunos problemas clásicos de la teoría de gráficas como determinación de circuitos eulerianos, determinación de circuitos hamiltonianos, planaridad y coloración; también se presentan algoritmos para resolver dichos problemas, en el caso de los circuitos hamiltonianos sólo se aborda cuando la gráfica cumple ciertas hipótesis. Los algoritmos fueron implementados en el programa computacional QGraphs, el cual fue producto de nuestro trabajo de tesis.

Abstract

In this thesis we work through some classical problems from graph theory such as finding Eulerian circuits, Hamiltonian cycles, planarity and graph coloring; we also present algorithms which solve these problems, the Hamiltonian cycle problem is only solved for graphs which satisfy certain hypothesis. The algorithms were implemented in the computer program QGraphs, which is product of our work.

Dedicatoria

De mis Padres y Abuelos he recibido grandes enseñanzas y el honor de ser uno de sus descendientes. Mis Hermanos y demás seres queridos, me han prodigado afecto fraternal verdadero. A todos ellos dedico este primer trabajo y les pido su invaluable apoyo para continuar con el ejemplo que me han inculcado. ¡Siempre estarán en mi mente!

Agradecimientos

A todos aquellos de quienes he aprendido que la adquisición de conocimiento es un proceso continuo, dentro y fuera de las aulas. Mis más sinceros y profundos agradecimientos, particularmente a: Dr. Fernando Barrera Mora, por el conocimiento que indujo en mí desde mi infancia y por todas sus enseñanzas; Dr. Rubén Alejandro Martínez Avendaño y Dr. Roberto López Hernández, por su gran apoyo al desarrollar este trabajo; Mtro. Andrés Aguilar Contreras, por haberme enseñado a apreciar el arte de programar.

También mi agradecimiento a todas las instituciones educativas públicas, con las que siempre estaré en deuda.

Índice general

Introducción

El origen de la teoría de gráficas se remonta al año de 1736, año en el que Leonhard Euler abordó y resolvió el célebre problema de los puentes de Königsberg [6]. Posteriormente, en las investigaciones de Kirchoff acerca de redes eléctricas él sustituyó los componentes de una red eléctrica por puntos y líneas para estudiarlas [6]. Cayley logró contar el número de isómeros (moléculas que tienen la misma fórmula química pero diferentes propiedades estructurales) de un cierto compuesto [2]. Mediante la utilización de técnicas similares se lograron resolver diferentes problemas en diversas áreas de la ciencia, entre ellos los recién mencionados. La semejanza entre las técnicas para resolver estos problemas dio origen a los conceptos fundamentales de la teoría de gráficas. Esto fue lo que consolidó dicha teoría [9].

En este trabajo se muestran diversos algoritmos que resuelven problemas clásicos de la teoría de gráficas así como una implementación de éstos en un programa computacional al cual hemos llamado QGraphs. En el primer capítulo se introducen los conceptos básicos de la teoría de gráficas así como la notación que se utilizará a lo largo de este escrito. En el segundo capítulo abordamos el problema de los puentes de Königsberg. Presentamos un modelo similar al que fue propuesto por Leonhard Euler para resolver el problema recién mencionado. Asimismo se presenta una caracterización de las gráficas eulerianas y un algoritmo que se encarga de encontrar un circuito euleriano en caso de existir. Cuando existe el circuito euleriano, el algoritmo se encarga de descomponer la gráfica en ciclos disjuntos para finalmente formar un circuito euleriano a partir de ellos.

En el Capítulo 3 se presenta un algoritmo que se encarga de encontrar circuitos hamiltonianos; los orígenes de este problema se remontan al siglo XIX, y se ha visto que es un problema considerablemente difícil de resolver [2]. En este capítulo se resuelve el caso particular del problema de los circuitos hamiltonianos en el que cada vértice es adyacente al menos a la mitad del total. La idea central del algoritmo consiste en encontrar un ciclo inicial y hacerlo crecer repetidamente hasta obtener un circuito hamiltoniano.

En el cuarto capítulo se muestra un algoritmo aplicable solamente a gráficas 2-conexas. El algoritmo recién mencionado obtiene una numeración st de una gráfica. Entenderemos por numeración st de una gráfica G con n vértices, una etiquetación de los vértices de G del 1 al n que cumple dos condiciones. La primera de las condiciones es que el vértice etiquetado con 1 sea adyacente al etiquetado con n y la segunda es que para todo vértice con etiqueta k con $1 < k < n$, deben de existir dos vértices adyacentes a él cuyas etiquetas sea una mayor y una menor. El algoritmo presentado en este capítulo será de gran utilidad en capítulos posteriores.

En el Capítulo 5 presentamos el problema de planaridad, es decir, dada una gráfica G, se comprueba si existe una representación de G en el plano sin que se crucen sus aristas. El algoritmo que se presenta en este capítulo requiere del algoritmo del Capítulo 4. Inicialmente requerimos de una gráfica 2-conexa con sus vértices etiquetados con una numeración st. A partir de lo anterior se verifica la planaridad de la gráfica. Posteriormente extendemos el algoritmo para gráficas en general, es decir, no solamente las 2-conexas, esto se logra mediante un algoritmo encargado de descomponer una gráfica dada en sus componentes 2-conexas.

Finalmente, en el Capítulo 6 se presenta un algoritmo que asigna una cinco coloración de una gráfica planar, esto es, dada una gráfica planar, se asigna a cada uno de sus vértices un color, de tal forma que vértices adyacentes tengan colores diferentes, lo anterior se logra con sólo cinco colores. El algoritmo anterior es relativamente sencillo e intuitivo y está basado en la demostración del teorema de Heawood acerca de 5 coloración de una gráfica planar.

Este trabajo se complementa con el programa QGraphs que implementa los algoritmos anteriormente mencionados. QGraphs fue escrito en el lenguaje de programación C++ utilizando el compilador `gcc` de GNU y la versión libre de la librería Qt para la interfaz gráfica.

Capítulo 1

Conceptos Básicos

En este capítulo se establecen las definiciones básicas de la teoría de gráficas. También se presenta el algoritmo *búsqueda primero en profundidad* (en inglés *depth first search*), el cual resultará de utilidad en capítulos posteriores.

1.1. Definiciones y Resultados Básicos

Definición. Una *gráfica* es una terna $G = (V, E, I)$, que consta de dos conjuntos V y E, y una función I donde:

1. El conjunto V es no vacío y a sus elementos los llamaremos *vértices* o *puntos*.
2. Los elementos del conjunto E serán llamados *aristas* o *líneas*.
3. La función I tiene dominio E y como contradominio el el conjunto de pares no ordenados de elementos (no necesariamente distintos) de V. La función I es llamada la *función de incidencia* de G.

Para nuestros fines sólo estudiaremos aquellas gráficas para las cuales su conjunto de vértices y de aristas son ambos finitos. Dada una gráfica $G = (V, E, I)$, usualmente denotaremos a V, E e I por V_G, E_G e I_G respectivamente, a menos que el contexto sea claro. También denotaremos por n y m al número de vértices y aristas de G respectivamente.

Ejemplo. Sea $G = (V, E, I)$, donde $V := \{v_1, v_2, v_3, v_4\}$, $E := \{e_1, e_2, \ldots, e_7\}$ e I está definida por:

$e_1 \mapsto \{v_1, v_1\}$

$e_2 \mapsto \{v_1, v_2\}$

$e_3 \mapsto \{v_1, v_4\}$

$e_4 \mapsto \{v_2, v_4\}$

$e_5 \mapsto \{v_4, v_4\}$

$e_6 \mapsto \{v_1, v_4\}$

$e_7 \mapsto \{v_2, v_4\}$

Un diagrama de esta gráfica se encuentra en la Figura 1.1

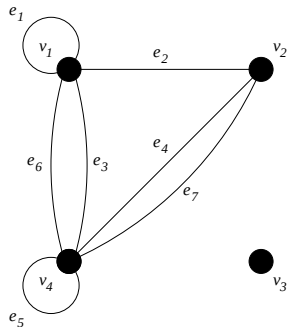

Figura 1.1: Diagrama de $G = (V, E, I)$

Como podemos observar en el ejemplo anterior, la función de incidencia no es necesariamente inyectiva, ya que $I(e_4) = I(e_7)$. A partir del diagrama de una gráfica G uno puede determinar si la función de incidencia de G es inyectiva: en caso de serlo, no habrá más de una arista entre cada par de vértices. A los elementos de E que son mapeados a un par de elementos iguales de V les llamaremos *lazos*, tal es el caso de e_5 y e_1 en el ejemplo anterior. Los lazos pueden ser detectados de manera sencilla en el diagrama, ya que son aquellas líneas que unen a algún punto consigo mismo.

Sea $G = (V, E, I)$ una gráfica. Diremos que G es *simple* si I es inyectiva y además E no contiene lazos, es decir, si el diagrama de G no tiene más de una línea entre cada par de vértices y no hay aristas que unen un vértice consigo mismo. Dados un vértice $v \in V$ y una arista $e \in E$, diremos que son *incidentes* si $v \in I(e)$. Sean $v_1, v_2 \in V$ y $e_1, e_2 \in E$, diremos que v_1 es *adyacente* a v_2, o que v_1 y v_2 son vecinos, si existe alguna arista en E que sea mapeada mediante I a $\{v_1, v_2\}$, y diremos que e_1 y e_2 son *adyacentes* si $I(e_1)$ e $I(e_2)$ tienen algún vértice en común. Dado un vértice v de G denotaremos al conjunto de vecinos de v por $N(v)$. Observemos que puede darse el caso en que $v \in N(v)$.

Definición. Sea G una gráfica simple. Diremos que G es *completa* si todos los puntos de G son adyacentes entre sí. Si G tiene n vértices entonces la gráfica completa de n vértices es denotada por K_n. A K_1 se le conoce como *gráfica trivial.*

Ejemplo. En la Figura 1.2 se muestran los diagramas de K_3 y K_5

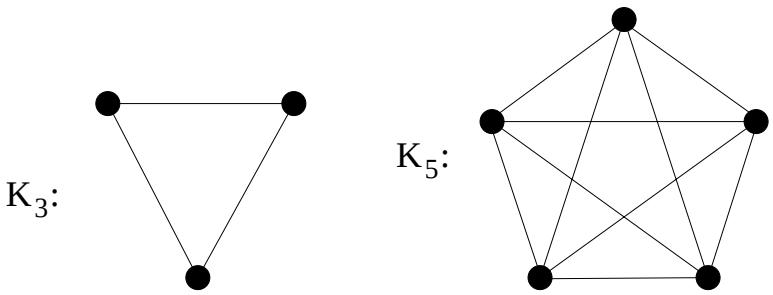

Figura 1.2: Diagramas de K_3 y K_5

Proposición 1.1. *La gráfica completa K_n tiene $\frac{n(n-1)}{2}$ aristas.*

Demostración. Cada vértice de K_n es adyacente a $n-1$ vértices, por lo tanto tenemos $n(n-1)$ relaciones de adyacencia, pero cada una fue contada dos veces, por lo cual tenemos $\frac{n(n-1)}{2}$ aristas. □

Definición. Dadas dos gráficas G_1 y G_2, decimos que G_1 es *isomorfa* a G_2, denotado por $G_1 \cong G_2$, si existen funciones biyectivas $\phi : V_1 \to V_2$ y $\theta : E_1 \to E_2$ tal que para cada $e \in E_1$ se tiene $I_1(e) = \{v_1, v_2\}$ si y sólo si $I_2(\theta(e)) = \{\phi(v_1), \phi(v_2)\}$

Ejemplo. Las gráficas G_1 y G_2 que se muestran en la Figura 1.3 son isomorfas. En realidad se trata de dos representaciones distintas de K_4.

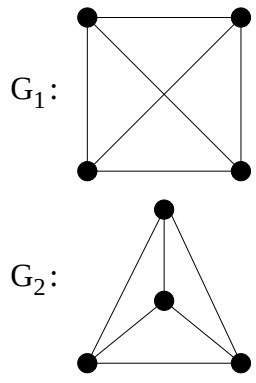

Figura 1.3: Dos diagramas de K_4

Definición. Sea G una gráfica. Decimos que la gráfica H es una *subgráfica* de G si:

1. Los vértices de H son vértices de G.
2. Las aristas de H son aristas de G.
3. La restricción de la función de incidencia de G a las aristas de H coincide con la función de incidencia de H.

En este mismo caso, cuando H es subgráfica de G, decimos que G es *supergráfica* de H.

Ejemplo.
En la Figura 1.4, la gráfica H es subgráfica de G.

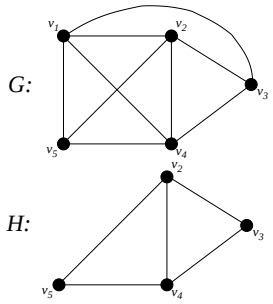

Figura 1.4: H es subgráfica de G

Diremos que una subgráfica H de G es *generadora* si H tiene los mismos vértices que G. Siendo G y H como en el ejemplo anterior, podemos observar que H no es subgráfica generadora de G, ya que v_1 no es vértice de H. Dada una gráfica G y un subconjunto no vacío $S \subseteq V_G$, la *gráfica inducida por S*, denotada por $G[S]$, es la subgráfica de G cuyo conjunto de vértices es S y cuyas aristas son todas las aristas de G que son incidentes a dos vértices de S. Sea G la gráfica de la Figura 1.4, y sea $S = \{v_2, v_3, v_4, v_5\}$; podemos observar que $H \cong G[S]$.

Sea G una gráfica y sea $S \subseteq V_G$, denotamos por $G \setminus S$ a $G[V_G \setminus S]$. Dados u y $v \in V_G$, definimos $G + uv$ como la gráfica G con una arista adicional entre u y v.

Sea G una gráfica y $v \in V_G$. Definimos el *grado* de v como:

$$\deg(v) = \Big|\big\{e \in E_G \mid e \text{ es incidente a } v\big\}\Big|,$$

es decir, el número de aristas que son incidentes a v. Nótese que los lazos son incidentes dos veces a un vértice. En el caso en que el grado del vértice v sea cero, diremos que v es un vértice aislado. Denotaremos por $\delta(G)$ y por $\Delta(G)$ al mínimo y al máximo de los grados de los vértices de G respectivamente.

Definición. Un *camino* en una gráfica G es una sucesión alternante de vértices y aristas empezando y terminando en vértices, tal que cada arista es incidente a los vértices anterior y posterior en la lista. Un *camino cerrado* en G es un camino en el que el vértice inicial es el mismo que el vértice final. Una *trayectoria* en G es un camino donde no se repiten aristas, mientras que un *paseo* en G es un camino donde no se repiten los vértices. Finalmente, un *ciclo* es una trayectoria cerrada en la que los únicos vértices que son iguales son el inicial y el final. Nótese que la definición permite que un camino tenga un solo vértice. Una gráfica que forma un paseo con todos sus vértices y todas sus aristas será llamada un paseo.

Cuando el contexto sea claro, omitiremos las aristas en las sucesiones que denotan caminos.

Ejemplo. Sea G la gráfica de la Figura 1.5. La sucesión $v_4v_5v_1v_5v_6v_2$ es un camino en G. Un ejemplo de un camino cerrado en G es la sucesión $v_7v_6v_3v_2v_6v_7$. Las sucesiones $v_4v_5v_6v_2v_3v_6$ y $v_1v_4v_5v_6v_7$ son una trayectoria y un paseo respectivamente. Un ciclo de G es el camino $v_2v_3v_6v_2$.

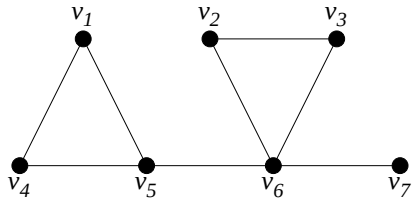

Figura 1.5: Caminos en una gráfica

Proposición 1.2. *Todo camino entre dos vértices u y v contiene un paseo entre u y v.*

Demostración. Sea $P = v_1v_2\dots v_k$ un camino entre u y v. Si P es un paseo, entonces P es el paseo buscado, de lo contrario, existen i y j tal que $i < j$ y $v_i = v_j$, redefinimos a P como $P := v_1v_2\dots v_{i-1}v_iv_{j+1}\dots v_k$. Mientras P no sea un paseo, la aplicación repetida de este argumento producirá eventualmente un paseo P entre u y v. □

Decimos que una gráfica G es *conexa* si dados dos vértices arbitrarios u y v de G, existe un camino, y por la Proposición 1.2 un paseo, entre u y v. Una *componente conexa* de G es una subgráfica conexa maximal de G. Diremos que una gráfica es *disconexa* si tiene al menos dos componentes conexas. Si una gráfica tiene tantas componentes conexas como vértices, diremos que dicha gráfica es *completamente disconexa*.

La *longitud* de un camino es el número de aristas del camino.

Proposición 1.3. *Sea G un paseo no trivial. Cualquier subconjunto S de vértices, de cardinalidad al menos $n/2+1$, debe de contener al menos un par de elementos que sean adyacentes en G.*

Demostración. Procedamos por inducción sobre el número de vértices n, lo cual es equivalente a proceder por inducción sobre el número de aristas m, ya que en el caso de los paseos $n = m+1$. Es claro que para $n = 2$ se cumple la proposición. Supongamos que se cumple para todo paseo con a lo más k vértices. Demostrémoslo para un paseo de longitud $k+1$. Sea G un paseo con $k+1$ vértices, y sea $S \subseteq V$ tal que $|S| \geq (k+1)/2+1$. Supongamos ahora que S contiene alguno de los dos vértices de G que son extremos del paseo y sea v este vértice. Si el vértice que es adyacente a v no se encuentra en S, entonces se sigue que los vértices restantes de S se encuentran entre los últimos $k-1$ vértices, como se muestra en la Figura 1.6.

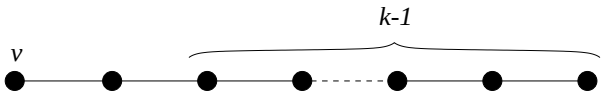

Figura 1.6: El subconjunto S contiene al menos un vértice del extremo.

Esto se traduce a que $S \setminus \{v\}$ es un subconjunto de los vértices de un paseo de longitud $k-1$, pero

$$\left|S \setminus \{v\}\right| \geq \frac{k+1}{2} + 1 - 1 = \frac{k-1}{2} + 1,$$

y por hipótesis de inducción se sigue que al menos dos vértices de $S \setminus \{v\}$ (y por lo tanto de S) son consecutivos.

Por otra parte, si S no contiene ninguno de los dos vértices extremos de G, entonces la demostración es completamente análoga al caso anterior, ya que, como se ilustra en la Figura 1.7, tomaremos un subconjunto con al menos $(k+1)/2$ vértices de una paseo con $k-1$ vértices, y nuevamente por hipótesis de inducción, se cumple que S contiene al menos un par de vértices consecutivos.

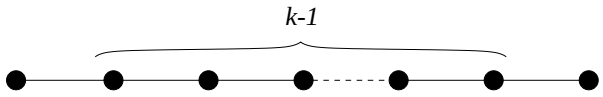

Figura 1.7: El subconjunto S no contiene ningún vértice extremo de G.

□

Corolario 1.4. *Sea G un ciclo con $n \geq 3$. Cualquier subconjunto S de vértices que tenga al menos $(n+1)/2$ elementos contiene al menos un par de vértices adyacentes en G.*

Demostración. Sea G un ciclo de longitud $n \geq 3$, podemos asumir que $|S| < n$, de lo contrario es evidente que se cumple el enunciado. Sea v un vértice de G que no está en S. Aplique la proposición anterior a $G - \{v\}$ □

Definición. Sea G una gráfica. Se define para dos vértices u y v en G la *distancia* entre u y v como:

$$d(u,v) = \text{mín}\{\text{longitudes de los paseos entre } u \text{ y } v\}$$

ó

$$d(u,v) = \infty \text{ (si } u \text{ y } v \text{ están en diferentes componentes conexas)}.$$

Definición. Una gráfica G que no tiene ciclos y es conexa se llama *árbol*. A toda gráfica acíclica se le llama *bosque*, en particular un árbol es un bosque. En ocasiones resulta conveniente distinguir un vértice r, llamado *raíz*, de un árbol T, en este caso a la pareja (T,r) le llamaremos un árbol T con raíz r; habrá ocasiones en que nos interese hablar de una árbol con raíz (T,r) sin hacer uso específico de r, en estas ocasiones nos referiremos a (T,r) simplemente como T.

Cuando consideramos un árbol con raíz, se induce una gradación de los vértices de la siguiente manera; diremos que un vértice u es *ancestro* de un vértice v si u está contenido en el paseo único entre la raíz y v, en este caso, también decimos que v es *descendiente* de u. Observemos que todo vértice es ancestro y descendiente de sí mismo. En caso de que u sea ancestro de v y se encuentren a distancia 1, diremos que u es *padre* de v y que v es *hijo* de u, notemos que, debido a que u no se encuentra a distancia 1 de sí mismo, u no es padre ni hijo de sí mismo. Este tipo de gráficas nos resultará de gran utilidad en los capítulos posteriores. El uso principal que le daremos a este tipo de gráficas radica en el algoritmo que presentaremos en la siguiente sección.

1.2. Algoritmos

En capítulos sucesivos habrá secciones en las que se enuncian algoritmos o procedimientos. En la descripción de estos se utilizarán estructuras de datos básicas, tales como *listas*, *pilas* y *colas*. Usualmente denotaremos a las estructuras de datos con letras mayúsculas. También haremos uso de procedimientos que involucran estructuras de datos, tales como remover elementos o añadirle elementos. En el caso de las pilas y colas, hacemos uso de las funciones *push* y *pop* que sirven para insertar y remover un elemento respectivamente. En el Apéndice A se muestran más detalles acerca de las estructuras de datos mencionadas y su funcionamiento.

A continuación presentamos un algoritmo, llamado *búsqueda primero en profundidad*, que nos será de gran utilidad en algoritmos que presentaremos en capítulos posteriores.

Algoritmo 1 Búsqueda Primero en Profundidad

Requiere: gráfica G, vértice $v \in V_G$
marque v como visitado
2: inicialice pila P a $\{v\}$
inicialice gráfica T con vértice v
4: **mientras** $|P| > 0$ **haga**
sea $u := pop(P)$
6: **si** $N(u) \setminus \{v \in V_G \mid v \text{ está marcado como visitado}\} \neq \emptyset$ **entonces**
sea $w \in N(u) \setminus \{v \in V_G \mid v \text{ está marcado como visitado}\}$
8: $push(u, P)$
marque w como visitado
10: agregue vértice w a T, agregar también arista entre u y w en T
$push(w, P)$
12: **fin si**
fin mientras
14: **regrese** T

Proposición 1.5. *El Algoritmo 1 obtiene un árbol generador de la componente conexa de v.*

Demostración. Primero demostraremos la proposición para una gráfica conexa y después para una gráfica disconexa.

Supongamos que G es conexa. Primero demostremos que T es una gráfica generadora. Supongamos, por contradicción, que algún vértice $u \in V_G$ no fue visitado por el Algoritmo 1. Como G es conexa, existe un paseo $v v_1 v_2 \dots v_k u$ entre v y u. Notemos que si un vértice w de G es visitado (y por lo tanto agregado a T junto con una arista incidente a él) durante la ejecución del ciclo **mientras**, entonces eventualmente todos los vecinos de w serán visitados. De lo anterior se sigue que partiendo de v, necesariamente fue visitado v_1. Al ser visitado v_1, necesariamente fue visitado v_2. A partir de la aplicación repetida de este argumento se concluye que necesariamente fue visitado v_k y por lo tanto también u, lo cual es una contradicción.

Demostramos ahora que T es un árbol, es decir, mostremos que G es conexa y acíclica. La gráfica T es conexa porque existe un camino de v a cualquier otro vértice, en consecuencia existe un camino entre cualquier par de vértices de T.

Ahora, demostremos que T es acíclica. Asumamos que T contiene un ciclo, digamos $u v_1 v_2 \dots v_k w$, denotamos por u al vértice que fue visitado más tempranamente por el algoritmo, además $u = w$. Esto significa que w fue visitado posteriormente por el algoritmo, pero esto no es posible debido a que en el ciclo **mientras** se evita que se visite más de una vez a un mismo vértice debido a la condición de la línea 6 y en consecuencia T no puede contener ciclos. Por lo tanto T es un árbol generador.

Si G es disconexa, aplique el caso anterior a la componente conexa de v. □

Capítulo 2

Gráficas Eulerianas

La teoría de gráficas surge al abordar el renombrado problema de los puentes de Königsberg [6]. Este problema fue resuelto por el matemático Leonhard Euler en el año de 1736. El problema consistía en encontrar un recorrido a través de la ciudad de Königsberg en el cual se pasara por cada puente exactamente una vez regresando al punto de partida. En el artículo titulado *Solutio problematis ad geometriam situs* se demuestra que no existe dicho recorrido a través de los puentes de Königsberg, para más detalles al respecto consulte [9]. En ese artículo, aunque no se utiliza la terminología moderna de teoría de gráficas, los métodos utilizados son escencialmente los de esta teoría. Este es el primer resultado del que se tiene conocimiento en esta rama de la matemática [9].

En este capítulo se estudiarán gráficas en las cuales existan caminos con propiedades similares a aquellas del recorrido buscado en el problema de los puentes de Königsberg. Se muestra un algoritmo que se encarga de encontrar el recorrido en una gráfica en caso de existir.

2.1. Caracterización de Gráficas Eulerianas

Definición. Sea G una gráfica, diremos que G es *euleriana* si existe un camino generador cerrado C que contiene exactamente una vez a cada arista de G. A C le llamaremos *circuito euleriano*.

Observemos que de la definición anterior se sigue directamente que G es necesariamente conexa. Además cómo $\delta(G) > 0$ la condición de ser generador es superflua.

Ejemplo. Sea G la gráfica de la Figura 2.1. Podemos observar que

$$v_1 v_2 v_3 v_2 v_1 v_4 v_3 v_4 v_1$$

es un circuito euleriano de G.

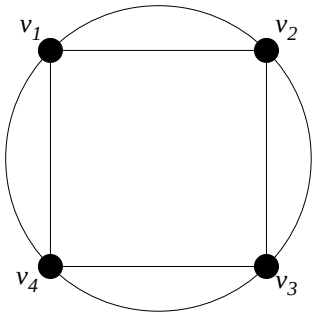

Figura 2.1: Gráfica euleriana

Recordando el problema de los puentes de Königsberg, este puede ser modelado con la gráfica de la Figura 2.2.

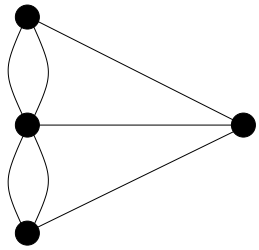

Figura 2.2: Los puentes de Königsberg

En la Figura 2.2 cada vértice representa una porción de tierra firme, y cada arista corresponde a un puente. En realidad el problema de los puentes de Königsberg se puede traducir a encontrar un circuito euleriano en la gráfica más recientemente mencionada.

El argumento que dio Euler para demostrar que no existía dicho recorrido consiste básicamente en que cada vez que se llegue a un vértice debe de ser posible abandonarlo mediante otra arista. Mientras que en el vértice inicial del recorrido, la condición es que una vez que lo abandone, tenga la posibilidad de regresar a él por medio de otra arista. Euler no sólo abordó el caso particular de la ciudad de Königsberg, sino que obtuvo un resultado más general, similar al presentado en [2], el cual se muestra a continuación.

Teorema 2.1. *Sea G una gráfica conexa. Entonces las siguientes afirmaciones son equivalentes:*

1. *G es euleriana.*
2. *Cada vértice de G tiene grado par.*
3. *Existe una partición de ciclos, disjuntos en aristas, de G.*

Demostración.

1) $\Rightarrow$ 2) Como se menciona en el párrafo anterior al enunciado del teorema, al recorrer el circuito euleriano, cada vez que se visite un vértice debe ser posible abandonarlo mediante otra arista; mientras que en el vértice extremo del circuito es visitado justo al inicio y al final del recorrido. De lo anterior se puede observar que el visitar cada vértice contribuye en dos a su grado, y por lo tanto cada vértice de G tiene grado par.

2) $\Rightarrow$ 3) Sea P un paseo maximal en G y sea v_k el vértice final de P. Necesariamente v_k es adyacente a algún vértice de P, esto se debe a que v_k tiene grado par, y no puede ser adyacente a algún otro vértice que no esté en P, porque esto último contradice la maximalidad de P. Sea v_j el vértice en P al que es adyacente v_k, por lo tanto $v_k v_j v_{j+1} \dots v_k$ es un ciclo en G. Ahora, considerando a la gráfica sin las aristas de este ciclo se tiene una nueva gráfica, posiblemente disconexa, en la que cada uno de sus vértices tiene grado par positivo o es aislado. La aplicación repetida de este argumento nos llevará a obtener una gráfica totalmente disconexa. Una vez llegado a este punto la partición en ciclos disjuntos ya estará construida.

3) $\Rightarrow$ 1) Si la partición en ciclos solo consta de uno solo, el circuito euleriano es el ciclo. Supongamos que la partición consiste exactamente de dos ciclos, digamos C_1 y C_2. La conexidad de G implica que hay algún vértice que tienen en común C_1 y C_2. El circuito euleriano es el camino que comienza en el vértice v que tienen en común los ciclos, recorriendo C_1 seguido de recorrer C_2. Una argumentación análoga nos da el circuito euleriano para cuando la partición tiene más ciclos.

□

2.2. Algoritmo

Una aplicación directa del teorema anterior resuelve el problema de los puentes de Königsberg, basta con un breve análisis de la gráfica en la Figura 2.2. Como en esta hay vértices que tienen grado impar, se sigue que dicho recorrido no existe.

La caracterización de este tipo de gráficas además de elegante nos permite dar un algoritmo que encuentra un circuito euleriano en una gráfica euleriana dada. A continuación se presenta un procedimiento que será una herramienta para obtener un circuito euleriano.

Procedimiento 2 Obtiene un ciclo

Requiere: gráfica G tal que $\forall\ v \in V(G)$, $\deg v$ es par y $\Delta(G) > 0$

sea $v \in V(G)$ tal que $\deg v > 0$
sea $w \in N(v)$
si $v = w$ **entonces**
  regrese $\{v, v\}$
fin si
inicialice cola P a $\{v, w\}$
mientras $N(w) \cap V(P)^c \neq \emptyset$ **haga**
  sea $u \in N(w) \cap V(P)^c$
  $push(u, P)$
  $w := u$
fin mientras
$v := pop(P)$
mientras $v \notin N(w)$ **haga**
  $v := pop(P)$
fin mientras
sea $C := \{v, P, v\}$
regrese C

Proposición 2.2. *El Procedimiento 2 regresa un ciclo C.*

Demostración. Sea C el resultado obtenido de aplicar el Procedimiento 2 a una gráfica que cumple con los requerimentos del procedimiento. Notemos que C nunca será vacío, ya que siempre se puede encontrar un vértice inicial, debido a que $\Delta(G) > 0$. En el segundo ciclo **mientras** de la línea 13 es necesario que w sea adyacente a algún vértice que se encuentre en la cola P porque $\deg(w)$ es par y P es maximal; de lo anterior tenemos que P es una trayectoria no vacía y por lo tanto C tendrá al menos tres elementos.

Probaremos que C es un ciclo, analizando primero el caso en el la longitud de C es 1 y posteriormente el caso en que la longitud de C es mayor que 1. Si $C = \{v, v\}$, $v \in V(G)$, claramente C es un ciclo. Notemos que la única manera de obtener un ciclo de longitud 1 es antes de haber ingresado al primer ciclo **mientras**, ya que una vez en éste se toman vértices que no están en la cola P.

Por otra parte, si $C = v_1, \ldots, v_k$, $k > 2$, basta probar que $v_1 = v_k$ y que $v_i \neq v_j$ si $1 \leq i < j < k$. Es claro que $v_1 = v_k$, por definición de C. Ahora demostremos por

contradicción que $v_i \neq v_j$ si $1 \leq i < j < k$, supongamos que $v_i = v_j$ con $1 \leq i < j < k$, entonces necesariamente en la iteración número j del ciclo **mientras**, el vértice i ya pertenecía a $V(P)$, lo cual contradice la instrucción en la línea 8, ya que $v_i \notin V(P)^c$. □

Otro procedimiento que será de utilidad es el que obtiene un circuito euleriano a partir de un vértice v, una lista de vértices l y una partición de ciclos $\mathscr{C}$ en la que ningún ciclo ha sido visitado. Este procedimiento es recursivo, y escencialmente lo que hace es aumentar a l los vértices de un ciclo que aún no ha sido visitado. Durante la ejecución del procedimiento es posible que se haga una llamada recursiva del procedimiento con ciclos que ya estén marcados como visitados, así que habrá ocasiones en que algunos ciclos de $\mathscr{C}$ ya estén etiquetados como visitados.

Procedimiento 3 Obtiene un circuito euleriano

Requiere: vértice v, lista de vértices l, lista de ciclos $\mathscr{C}$

marque el primer ciclo no visitado al que pertenece v como visitado
2: $u := v$
repita
4: **si** v pertenece a otro ciclo de $\mathscr{C}$ que no haya sido visitado **entonces**
Procedimiento 3(v, l, $\mathscr{C}$)
6: **fin si**
$insert(v, l)$
8: redefina v como el siguiente vértice del ciclo al que pertenece
hasta $v = u$
10: $insert(v, l)$
regrese l

Proposición 2.3. *El Procedimiento 3 obtiene un circuito euleriano dada una partición de ciclos disjuntos en aristas $\mathscr{C}$ de una gráfica conexa si ningún ciclo de $\mathscr{C}$ ha sido visitado.*

Demostración. La demostración se hará por inducción sobre el número de ciclos en $\mathscr{C}$. Si $\mathscr{C}$ sólo consta de un ciclo, nunca se cumplirá la condición de la línea 4. Al omitir la condición de la linea 4, es claro que l será el ciclo que contiene $\mathscr{C}$.

Supongamos ahora que el procedimiento funciona si $\mathscr{C}$ contiene a lo más n ciclos disjuntos. Demostremos que funciona si contiene $n+1$ ciclos. Sea c_k el último ciclo que se marcó como visitado durante la ejecución del Procedimiento 3 a $\mathscr{C}$ con $n+1$ ciclos. Por hipótesis de inducción, el procedimiento obtendrá un circuito euleriano de la gráfica formada por los vértices y aristas que se encuentran en $\mathscr{C} \setminus \{c_k\}$, digamos $l = v_1 v_2 \dots v_r$. Como G es conexa, existen vértices en común entre l y c_k. Sea v_j el primer vértice en común visitado por el Procedimiento 3 que tiene c_k con l. Sin

pérdida de generalidad, podemos suponer que $c_k = u_1 u_2 \dots u_s$, donde $u_1 = u_s = v_j$. Siguiendo la ejecución del procedimiento, al recorrer el ciclo c_k todos los ciclos en $\mathscr{C}$ ya están marcados como visitados. Por lo anterior, nunca se cumplirá la condición de la línea 4. Lo anterior implica que como resultado de recorrer c_k solamente se insertarán los vértices que lo conforman a l. Al final de la ejecución del procedimiento $l = v_1 v_2 \dots v_j u_2 \dots u_s v_{j+1} v_{j+2} \dots v_r$.

Para concluir la demostración probemos que l es un circuito euleriano. Es claro que l será un camino cerrado, ya que el elemento inicial y el final son el mismo por definición. Es claro también que cada arista de la gráfica aparece sólo una vez, ya que los ciclos son disjuntos y cada ciclo se recorre una sola vez. Por lo tanto el Procedimiento 3 obtiene circuitos eulerianos a partir de una partición en ciclos disjuntos. □

Ahora procedemos a enunciar un algoritmo que utiliza los dos procedimientos anteriores para obtener un circuito euleriano a partir de una gráfica euleriana. La manera en que se logra esto es similar a la demostración del teorema anterior, es decir, primero se encuentra una partición en ciclos disjuntos y finalmente se recorren de manera adecuada estos ciclos. En el siguiente algoritmo $\mathscr{C}$ denotará una lista cuyos elementos son ciclos, y denotaremos el conjunto de aristas de los ciclos de $\mathscr{C}$ por $E(\mathscr{C})$.

Algoritmo 4 Obtiene un circuito euleriano

Requiere: gráfica G euleriana
inicialice lista $\mathscr{C}$ a vacía
2: **mientras** $G \setminus E(\mathscr{C})$ no es totalmente disconexa **haga**
$insert$(Procedimiento 2$(G \setminus E(\mathscr{C})), \mathscr{C})$
4: **fin mientras**
sea $v \in V(G)$
6: inicialice lista l a vacía
marque todos los elementos de $\mathscr{C}$ como no visitados
8: Procedimiento 3$(v, l, \mathscr{C})$
regrese l

Proposición 2.4. *Sea G un gráfica euleriana. Entonces Algoritmo 4(G) produce un circuito euleriano.*

Demostración. Notemos que el remover las aristas de un ciclo en G produce una nueva gráfica cuyos vértices tienen grado par. Por lo anterior, la aplicación repetida de Procedimiento 2 produce una partición en ciclos disjuntos $\mathscr{C}$. Como Procedimiento 3 obtiene el circuito euleriano a partir de $\mathscr{C}$, entonces queda demostrado el funcionamiento del Algoritmo 4. □

Capítulo 3

Gráficas Hamiltonianas

En el año de 1859 salió a la venta un juego de mesa llamado *A Voyage Round the World*, que fue ideado por el matemático Sir William Rowan Hamilton. El juego tenía una representación del dodecaedro como una gráfica, en la que cada vértice representa una ciudad, y la presencia de una arista entre dos vértices indica que es posible viajar entre esas dos ciudades. El juego consistía en dar un paseo que visitara exactamente una vez cada ciudad, regresando a la ciudad de origen [9].

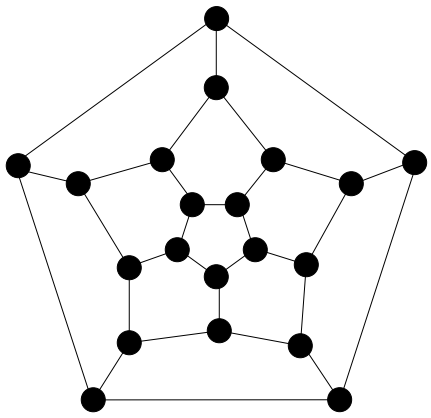

Figura 3.1: Dodecaedro.

Un problema que a simple vista no parece tener algo en común con el juego anterior, es el problema del *tour* del caballo en un tablero de ajedrez. El problema consiste en visitar exactamente una vez cada casilla de un tablero de ajedrez mediante movimientos legales de un caballo, finalizando dicho recorrido donde se inició. Nuevamente, el matemático Leonhard Euler fue quien presentó su solución a este problema en el año de 1759 [9].

En general, las gráficas que tienen recorridos del estilo de los ejemplos anterio-

res se llaman hamiltonianas. En el presente capítulo se enuncia el Teorema de Dirac concerniente a gráficas hamiltonianas. Posteriormente se mostrará un algoritmo que encuentra un circuito hamiltoniano en gráficas que satisfacen las hipótesis del teorema recién mencionado.

3.1. Algunos Resultados sobre Gráficas Hamiltonianas

Definición. Sea G una gráfica. Si en G existe un paseo generador P, diremos que P es un *paseo hamiltoniano*. Diremos G es *hamiltoniana* si existe un ciclo generador C en G. En este último caso a C le llamaremos *circuito hamiltoniano*.

Ejemplo. Sea G la gráfica de la Figura 3.2

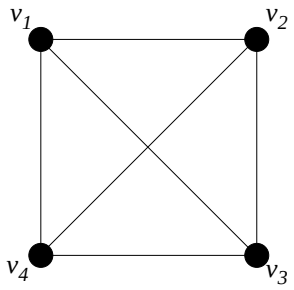

Figura 3.2: Gráfica hamiltoniana.

Podemos observar que $v_1 v_2 v_4 v_3 v_1$ es un circuito hamiltoniano de G

En el juego inventado por Hamilton, el jugador debía encontrar un circuito hamiltoniano en el dodecaedro. En el problema del tour del caballo podemos construir una gráfica de la siguiente manera. Habrá un vértice por cada casilla del tablero de ajedrez, y dos vértices serán adyacentes si es que se puede hacer un movimiento legal de caballo entre las casillas correspondientes a los vértices. Ahora el problema se reduce a encontrar un circuito hamiltoniano en esta gráfica, y es por esto que el problema del *tour* del caballo y *A Voyage Round the World* están relacionados.

A simple vista, el problema de hallar un circuito hamiltoniano parece ser casi tan sencillo como el de hallar un circuito euleriano. Pero esto no es así, este problema resulta mucho más complejo. Hasta donde tenemos conocimiento, no se sabe si existe un algoritmo que decida en "tiempo polinomial" (para más detalles véase [5]) si una gráfica dada es hamiltoniana o no. Tampoco existe un teorema que caracterize de manera elegante a estas gráficas [6].

A pesar de ser éste un problema difícil, existen diversos teoremas que dan condiciones suficientes para que una gráfica sea hamiltoniana. A continuación se enuncia uno de ellos; también puede ser encontrado en [2].

Teorema 3.1 (Ore)**.** *Sea G una gráfica simple y conexa con $n \geq 3$. Si para cada par de vértices no adyacentes u y v se cumple que*

$$\deg u + \deg v \geq n$$

entonces G es hamiltoniana.

Demostración. Por contradicción, sea G una gráfica que cumple con las hipótesis del teorema y que no es hamiltoniana.

Construyamos una supergráfica $G^\star$ de G de la siguiente manera: Agregaremos aristas entre vértices no adyacentes mientras esta gráfica siga siendo no hamiltoniana. La gráfica $G^\star$ existe, porque en un número finito de pasos se llega a K_n y K_n es hamiltoniana y necesariamente durante el proceso se obtiene la gráfica $G^\star$ deseada.

Afirmamos que $G^\star$ tiene un paseo hamiltoniano. Esto es porque al tomar dos vértices no adyacentes en $G^\star$, digamos v_1 y v_n, $G^\star + v_1 v_n$ es hamiltoniana. Sea

$$C = v_1 v_2 \dots v_{n-1} v_n v_1$$

el circuito hamiltoniano en $G^\star + v_1 v_n$. Notemos que la arista $v_1 v_n$ pertenece a C pero no a $G^\star$, ya que de lo contrario $G^\star$ sería hamiltoniana. Por lo tanto $v_1 v_2 \dots v_{n-1} v_n$ es un paseo hamiltoniano en $G^\star$. Es claro que $v_1 \notin N_{G^\star}(v_n)$, además, si $v_i \in N_{G^\star}(v_n)$, $1 \leq i < n$ entonces $v_{i+1} \notin N_{G^\star}(v_1)$, porque si ocurriera que $v_{i+1} \in N_{G^\star}(v_1)$ entonces $v_1 \dots v_i v_n v_{n-1} \dots v_{i+1} v_1$ sería un circuito hamiltoniano como se muestra en la Figura 3.3.

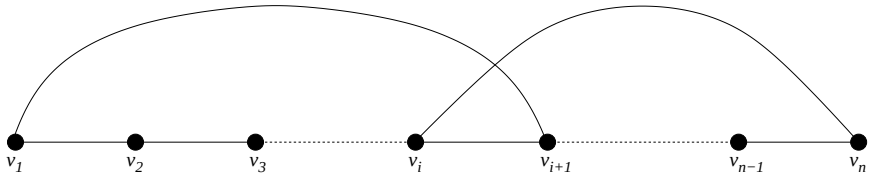

Figura 3.3: $G^\star$

Sabemos que en $G^\star$ el grado de v_1 es a lo más $n-1$, pero también sabemos que por cada vecino de v_n, v_1 tiene un vecino menos, así que tenemos

$$\deg_{G^\star} v_1 \leq n - 1 - \deg_{G^\star} v_n \tag{3.1}$$

Además sabemos que para cada $u \in V(G)$, $\deg_G u \leq \deg_{G^\star} u$. De lo anterior y de la ecuación (3.1) se sigue que

$$\deg_G v_1 + \deg_G v_n \leq n - 1$$

Lo cual es una contradicción. Por lo tanto G es hamiltoniana. □

Un corolario inmediato del teorema anterior es el siguiente.

Corolario 3.2 (Teorema de Dirac)**.** *Sea G una gráfica simple y conexa con $n \geq 3$. Si $\delta(G) \geq n/2$ entonces G es hamiltoniana.*

Demostración. La gráfica G cumple con las hipótesis del Teorema 3.1 □

3.2. Algoritmo

Dado que no se conoce algún algoritmo eficiente para la caracterización de gráficas hamiltonianas, el algoritmo de este capítulo encontrará circuitos hamiltonianos solamente en gráficas que cumplen con las hipótesis del Corolario 3.2.

Antes de enunciar algún procedimiento, procederemos a enunciar tres proposiciones que nos serán de utilidad.

Proposición 3.3. *Sea G una gráfica simple con $n \geq 4$ vértices, tal que $\delta(G) \geq n/2$. Si $C = v_1 v_2 \dots v_{n-2} v_{n-1} v_1$ es un ciclo en G de longitud $n-1$ y $w \notin V(C)$ entonces existen dos vértices consecutivos en C que son adyacentes a w y en consecuencia G tiene un circuito hamiltoniano de la forma $v_1 v_2 \dots v_k w v_{k+1} \dots v_{n-1} v_1$.*

Demostración. Se sigue directamente del Corolario 1.4 que w es adyacente a al menos dos vértices consecutivos en C debido a que $\deg w \geq n/2$, es decir, $|N(w)| \geq n/2$. Sean v_k y v_{k+1} estos vértices. Entonces la sucesión $v_1 v_2 \dots v_k w v_{k+1} \dots v_{n-2} v_{n-1} v_1$ forma un ciclo hamiltoniano de G.

□

Proposición 3.4. *Sea G una gráfica simple con $n \geq 5$ vértices tal que $\delta(G) \geq n/2$. Supongamos que G tiene un ciclo C de longitud $n-2$, digamos $v_1 v_2 \dots v_{n-3} v_{n-2} v_1$ y sean w_1 y w_2 los vértices de G que no están en C. Entonces alguno de los siguientes enunciados se cumple*

1. *Existe un par de vértices consecutivos en C que son adyacentes al mismo w_i, $i = 1$ ó 2 y por lo tanto hay un ciclo hamiltoniano.*
2. *El ciclo $v_1 w_i w_j v_3 \dots v_{n-2} v_1$ es de longitud $n-1$ en G.*
3. *El ciclo $v_1 w_i w_j v_2 \dots v_{n-2} v_1$ es hamiltoniano de G.*

Demostración. Si alguno de w_i, $i = 1$ ó 2 es adyacente a dos vértices consecutivos de C, entonces la sucesión $v_1 v_2 \dots v_k w_i v_{k+1} \dots v_{n-2} v_1$, donde $i = 1$ ó 2 y $1 \leq k \leq n-2$ es un ciclo de longitud $n-1$

Ahora supongamos que no ocurre eso, es decir, que para cada $i = 1$ ó 2 los vértices que son adyacentes a w_i no son adyacentes en C. Nótese que $w_1 w_2 \in E(G)$, ya

que de lo contrario, aplicando el Corolario 1.4, se sigue que w_i es adyacente a dos vértices consecutivos en C. Observemos que $\deg_{G-w_2}(w_1) \geq (n-2)/2$, por lo tanto w_1 es adyacente al menos a la mitad de los vértices de C, análogamente, se tiene el mismo argumento para w_2. Por otra parte si w_i es adyacente a más de la mitad de vértices de C, entonces hay al menos dos vértices consecutivos en C que son vecinos de w_i por lo tanto se sigue que w_i es adyacente a exactamente la mitad de vértices en C, y consecuentemente $N_{G-w_2}(w_1) = N_{G-w_1}(w_2)$ ó $N_{G-w_2}(w_1) = N_{G-w_1}(w_2)^c$ y n es par.

Supongamos que ocurre $N_{G-w_2}(w_1) = N_{G-w_1}(w_2)$, podemos asumir, sin pérdida de generalidad, que $v_1 \in N_{G-w_2}(w_1)$. De lo anterior se sigue que $N_{G-w_2}(w_1) = \{v_1, v_3, \ldots, v_{n-5}, v_{n-3}\}$, lo cual implica que $v_1 w_1 w_2 v_3 v_4 \ldots v_{n-2} v_1$ es un ciclo de longitud $n-1$ en G (esto se debe al hecho de que w_i es adyacente a la mitad de los vértices de C y además los vecinos de w_i son no adyacentes entre sí) y por lo tanto se sigue 2.

Por otra parte, si ocurre que $N_{G-w_2}(w_1) = N_{G-w_1}(w_2)^c$, y asumiendo que v_1 es vecino de w_1 en $G-w_2$ tenemos que $N_{G-w_2}(w_1) = \{v_1, v_3, \ldots, v_{n-5}, v_{n-3}\}$ y $N_{G-w_1}(w_2) = \{v_2, v_4, \ldots, v_{n-4}, v_{n-2}\}$, entonces $v_1 w_1 w_2 v_2 \ldots v_{n-2} v_1$ es un ciclo hamiltoniano de G, ya que si $v_k \in N_{G-w_2}(w_1)$, entonces $v_{k+1} \in N_{G-w_1}(w_2)$. Por lo tanto se cumple 3. □

Proposición 3.5. *Sean $n, l \in \mathbb{N}$, $l \geq 3$, tales que $n \geq l+3$ y $l \leq (n-1)/2$. Sea G una gráfica simple con n vértices, tal que $\delta(G) \geq n/2$. Supongamos que existe un ciclo C en G de longitud $n-l$ de la forma $v_1 v_2 \ldots v_{n-l-1} v_{n-l} v_1$, y sea $F := \{w_1, w_2, \ldots, w_l\}$ el conjunto de los vértices de G que no están en C. Entonces alguno de los siguientes enunciados es cierto*

1. *Existen w_i, con $1 \leq i \leq l$ y un par de vértices consecutivos en C tal que la sucesión $v_1 v_2 \ldots v_k w_i v_{k+1} \ldots v_{n-l} v_1$ es un ciclo de longitud $n-(l-1)$ en G.*

2. *Existe v_j, con $2 \leq j \leq l+1$ tal que, reetiquetando a los vértices w_i de ser necesario, $v_1 w_1 w_2 \ldots w_l v_j \ldots v_{n-l} v_1$ es un ciclo en G cuya longitud es mayor que $n-l$.*

Demostración. Si alguno de los w_i, $1 \leq i \leq l$ es adyacente a dos vértices consecutivos en C, se puede formar un ciclo de la forma $v_1 v_2 \ldots v_k w_i v_{k+1} \ldots v_{n-l} v_1$ y por lo tanto se cumple 1.

Ahora, supongamos que ningún elemento de F es adyacente a dos vértices consecutivos en C. De la suposición anterior se sigue que dado w_i un elemento de F,

$$\deg_{G[V(C)\cup\{w_i\}]}(w_i) \leq \frac{n-l}{2}$$

ya que de lo contrario w_i sería adyacente a dos vértices consecutivos en C, y por lo

tanto

$$\begin{aligned}\deg_{G[F]}(w_i) &\geq \frac{n}{2} - \deg_{G[V(C)\cup\{w_i\}]}(w_i)\\ &\geq \frac{n}{2} - \frac{n-l}{2} = \frac{l}{2}.\end{aligned}$$

En otras palabras, $\delta(G[F]) \geq |V(G[F])|/2$, y por el Teorema de Dirac, existe un ciclo hamiltoniano en $G[F]$. Asumamos, reetiquetando si es necesario, que este ciclo hamiltoniano es de la forma $w_1 w_2 \dots w_l w_1$.

Sea $w_i \in F$ fijo, demostremos que dos vecinos consecutivos de w_i en C, están a distancia a lo más l en C. El grado de w_i en $G[F]$ es a lo más $l-1$, así que

$$\begin{aligned}\deg_{G[V(C)\cup\{w_i\}]}(w_i) &\geq \frac{n}{2} - \deg_{G[F]}(w_i)\\ &\geq \frac{n}{2} - (l-1) = \frac{n-2l+2}{2}.\end{aligned}$$

Además nótese que, por ser $l \leq (n-1)/2$, se tiene que $(n-2l+2)/2 \geq 3/2$, es decir, hay al menos dos vecinos de w_i en el ciclo. Por contradicción, supongamos que la distancia entre dos vecinos consecutivos de w_i es mayor o igual que $l+1$, es decir, tendríamos que acomodar a al menos $(n-2l+2)/2 = (n-2l)/2+1$ vértices sobre un paseo de longitud $n-2l$. Pero por la Proposición 1.3 al acomodar $(n-2l)/2+1$ vértices sobre un paseo de longitud $n-2l$ necesariamente habrá al menos dos de ellos que son consecutivos, lo cual es una contradicción, ya que estamos asumiendo que eso no ocurre.

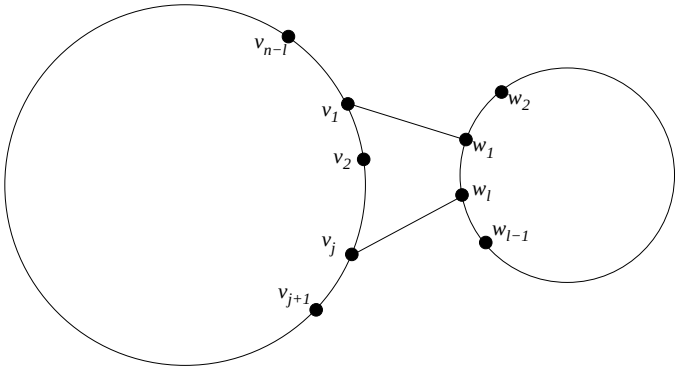

Figura 3.4: Reetiquetando podemos asumir que w_1 es adyacente a v_1

Volviendo a reetiquetar a F y a $\{v_1, v_2, \dots, v_{n-l}\}$ de ser necesario podemos asumir que w_1 es adyacente a v_1 como se muestra en la Figura 3.4. Por el párrafo anterior, los vecinos consecutivos de w_l en C están a distancia a lo más l, por lo tanto algún elemento de $\{v_2, v_3, \dots, v_l, v_{l+1}\}$ es vecino del vértice w_l. Por lo tanto la sucesión

$v_1 w_1 w_2 \dots w_l v_j v_{j+1} \dots v_{n-l} v_1$ es un ciclo en G, y su longitud es mayor que $n-l$ debido a que $2 \leq j \leq l+1$. Por lo tanto se cumple 2.

□

La idea fundamental del algoritmo de este capítulo es obtener un ciclo inicial y hacerlo crecer de alguna manera hasta obtener un circuito hamiltoniano. A continuación se enuncian un par de procedimientos que serán de utilidad para el algoritmo de este capítulo.

Procedimiento 5 Obtiene un ciclo

Requiere: gráfica simple G tal que $\delta(G) \geq n/2$ y $n \geq 3$

sea $v \in V(G)$
sea $w \in N(v)$
inicialice cola P a $\{v, w\}$
mientras $N(w) \cap V(P)^c \neq \emptyset$ **haga**
  sea $u \in N(w) \cap V(P)^c$
  $push(u, P)$
  $w := u$
fin mientras
$v := pop(P)$
mientras $v \notin N(w)$ **haga**
  $v := pop(P)$
fin mientras
sea $C := \{v, P, v\}$
regrese C

Proposición 3.6. *El Procedimiento 5 regresa un ciclo C de longitud al menos $n/2+1$.*

Demostración. Sea C el resultado de aplicar Procedimiento 5 a una gráfica apropiada. Demostremos primero que C es un ciclo y después que C tiene longitud al menos $n/2+1$. Por construcción (líneas 3–8), los vértices que se encuentran en la cola P forman un paseo. Las líneas 9–12 del Procedimiento 5 remueven elementos de P hasta encontrar algún vecino v de w, donde w es el último elemento insertado a P; v existe, ya que al finalizar el primer ciclo **mientras** se tiene que $N(w) \subseteq V(P)$. De lo anterior se sigue que $C := \{v, P, v\}$ es un ciclo.

Demostremos que C tiene longitud al menos $n/2+1$. Una vez que se termina de ejecutar el primer ciclo **mientras** w es un vértice cuyos vecinos están todos en P, en otras palabras $N(w) \subseteq V(P)$ y $|N(w)| \geq n/2$. El segundo ciclo **mientras** se encarga de sacar el primer vecino de w en P que fue insertado, digamos v_w. Al sacar a v_w de P, se tiene que $|P| \geq n/2$, porque $N(w) \setminus \{v_w\} \subseteq V(P)$ y además $w \in V(P)$. Así que el ciclo C regresado por el Procedimiento 5 tiene longitud al menos $n/2+1$. □

A continuación enunciamos el segundo procedimiento de este capítulo.

Procedimiento 6 Obtiene un ciclo de mayor longitud

Requiere: gráfica G tal que $\delta(G) \geq n/2$ y $n \geq 4$. Ciclo $C = u_1u_2\ldots u_ku_{k+1}$, donde $u_{k+1} = u_1$, $n/2+1 \leq k < n$

para cada $v \in V(G)$ tal que $v \notin V(C)$ **haga**
  para $i = 1\ldots k$ **haga**
    si $u_i \in N(v)$ y $u_{i+1} \in N(v)$ **entonces**
      regrese $\{u_1, \ldots, u_i, v, u_{i+1}, \ldots, u_k, u_1\}$
    fin si
  fin para
fin para
si $k = n-2$ **entonces**
  si $u_1 \in N(v_1)$ **entonces**
    si $u_1 \in N(v_2)$ **entonces**
      regrese $\{u_1, v_1, v_2, u_3, \ldots, u_k, u_1\}$
    si no
      regrese $\{u_1, v_1, v_2, u_2, \ldots, u_k, u_1\}$
    fin si
  si no
    si $u_1 \in N(v_2)$ **entonces**
      regrese $\{u_1, v_2, v_1, u_2, \ldots, u_k, u_1\}$
    si no
      regrese $\{u_1, u_2, v_2, v_1, u_4, \ldots, u_k, u_1\}$
    fin si
  fin si
si no
  sea $D :=$ Algoritmo 7$(G[V(C)^c])$
  sea $v_1 \in V(D)$ tal que $v_1 \in N(u_1)$
  sea $v_{n-k} \in N(v_1) \cap V(D)$
  para $i = 2\ldots n-k+1$ **haga**
    si $u_i \in N(v_{n-k})$ **entonces**
      regrese $\{u_1, v_1, v_2, \ldots, v_{n-k}, u_i, \ldots, u_k, u_1\}$
    fin si
  fin para
fin si

Proposición 3.7. *El Procedimiento 6 obtiene un ciclo de mayor longitud que C.*

Demostración. En las líneas 1–7 lo que se hace es una búsqueda exhaustiva de un vértice fuera de C que sea adyacente a dos vértices consecutivos en C. En las proposiciones 3.3–3.5 se establece que es posible, independientemente de la longitud del ciclo, que este vértice exista. Si este vértice existe necesariamente será encontrado y la longitud del ciclo regresado será mayor. Notemos que en el caso en que $k = n-1$, el vértice fuera de C tiene que ser adyacente a dos vértices consecutivos en C. En caso de que la búsqueda anterior no haya arrojado resultado alguno tenemos dos casos posibles, el de la línea 8 y el de la línea 22, que corresponden a los casos abordados en las proposiciones 3.4 y 3.5 respectivamente.

Si se cumple la condición de la línea 8, es decir, $k = n-2$ sólo es necesario encontrar el ciclo adecuado, el cual sólo puede tener una de cuatro formas. Esto sólo depende de si $N_{G-v_2}(v_1) = N_{G-v_1}(v_2)$ ó $N_{G-v_2}(v_1) = N_{G-v_1}(v_2)^c$ y del orden en que son adyacentes los vértices de C con v_1 y v_2. Lo anterior también produce un ciclo de mayor longitud.

En caso de no cumplirse la condición del párrafo anterior, tenemos necesariamente que $k \leq n-3$. Verifiquemos que C cumple con los requerimientos de la Proposición 3.5 para poder hacer crecer el ciclo. Denotemos por l al número de vértices en G que quedan fuera de C, observamos que $l = n-k$. Como $k \leq n-3$ es claro que $l \geq 3$ y que $n \geq l+3$. También tenemos que $n-k = l \leq (n-1)/2$ porque $k \geq n/2+1$. El método para hacer crecer el ciclo consiste en buscar entre los primeros $l+1$ vértices de C y unirlos a los extremos de un paseo generador en $G[V(C)^c]$. La manera en que se logra esto es haciendo una llamada recursiva al Algoritmo 7 que recibirá como argumento a $G[V(C)^c]$; lo anterior se puede hacer porque $\delta(G[V(C)^c]) \geq l/2$.

Una vez teniendo el circuito hamiltoniano $D = v_1 v_2 \dots v_l v_1$ de $G[V(C)^c]$, tomamos un vértice en $V(C)$ que sea adyacente a v_1, digamos u_k. Ahora sólo resta hacer la búsqueda de un vecino de v_l entre los siguientes l vértices de C partiendo de u_{k+1} para poder cerrar el ciclo. El ciclo obtenido es de longitud mayor a la de C por la Proposición 3.5. □

El siguiente algoritmo se encarga de encontrar un circuito hamiltoniano en una gráfica G con al menos tres vértices tal que $\delta(G) \geq n/2$

Algoritmo 7 Obtiene un circuito hamiltoniano

Requiere: gráfica G tal que $\delta(G) \geq n/2$ y $n \geq 3$
  $C :=$ Procedimiento 5(G)
  mientras $|C| < n$ **haga**
    $C :=$ Procedimiento 6(G, C)
  fin mientras
  regrese C

Proposición 3.8. *El Algoritmo 7 obtiene un circuito hamiltoniano.*

Demostración. Inicialmente C es un ciclo, posiblemente no sea un circuito hamiltoniano. Es posible que a partir de la ejecución del Procedimiento 5 obtengamos un circuito hamiltoniano, de hecho esto ocurre cuando G tiene 3 vértices. Así que éste será considerado como nuestro caso base para el paso recursivo. Después de la ejecución del Procedimiento 5 los vértices que quedan fuera de C son a lo más $n-(n/2+1) = (n-2)/2$. Se sigue de lo anterior que las llamadas recursivas se hacen con cada vez menos vértices fuera del ciclo, pero con al menos tres. Lo anterior se debe a que cuando tenemos menos de tres vértices fuera de C, el Procedimiento 6 no requiere de hacer llamadas recursivas. Es por esto que este algoritmo produce un circuito hamiltoniano.

□

Capítulo 4

Numeración *st*

En este capítulo presentamos el algoritmo que obtiene una numeración st de una gráfica dos conexa. La numeración st se aplica en las pruebas de planaridad, por lo que será de utilidad en capítulos posteriores.

4.1. Introducción

Definición. Sea G una gráfica simple y conexa. Diremos que G es *2-conexa*, si para todo $v \in V_G$, $G \setminus \{v\}$ es conexa. En caso de que para algún $v \in V_G$, $G \setminus \{v\}$ sea disconexa, a v se le llamará un *punto de corte* de G.

Ejemplo. Considere las gráficas G y H de la Figura 4.1. Claramente v_1 y v_2 son puntos de corte de H, mientras que la gráfica G es 2-conexa.

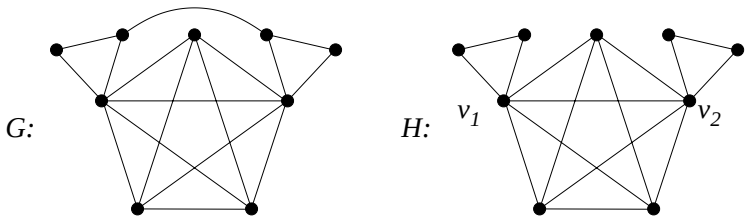

Figura 4.1: Ejemplos de gráfica 2-conexa y gráfica con puntos de corte.

El algoritmo que se enuncia en el presente capítulo se aplica solamente a gráficas 2-conexas. Por lo anterior, las gráficas que se considerarán en lo que sigue del capítulo serán únicamente gráficas 2-conexas, a menos que se especifique lo contrario.

Definición. Sea G una gráfica. Una *numeración st* de G es una etiquetación de los vértices de G con números del 1 al n en donde el vértice 1 es adyacente al vértice n y para todo $1 < i < n$, existen j y $k \in V_G$ tal que $j, k \in N(i)$ y $j < i < k$.

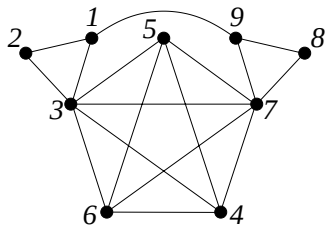

Figura 4.2: Numeración st de una gráfica.

Un resultado de Lempel, Even y Cederbaum [7] afirma que dada una gráfica 2-conexa G, se puede encontrar una numeración st de G. El algoritmo que se presenta en este capítulo obtiene una numeración st de una gráfica 2-conexa dada.

Ejemplo. En la Figura 4.2 se muestra una numeración st de una gráfica 2-conexa.

Dado un árbol T con raíz y dos vértices $u, v \in V_T$, $u \xrightarrow{\star} v$ denotará que v es descendiente de u. El símbolo $u \to v$ denotará que u es padre de v.

Diremos que un árbol con raíz T es un *árbol generador primero en profundidad* de una gráfica G si cumple el siguiente par de condiciones:

- T es subgráfica generadora de G
- para cada arista uv de G pero no en T, se tiene que $v \xrightarrow{\star} u$ ó $u \xrightarrow{\star} v$.

Notemos que el tipo de árboles recién definidos pueden ser obtenidos a partir del Algoritmo 1.

Ejemplo. En la Figura 4.3 se muestran dos subárboles generadores de G. Uno de ellos, T_1 *no* es un árbol generador primero en profundidad, ya que los vértices u y v son adyacentes en G pero no en T_1 y no se tiene que u es descendiente de v ni viceversa. El árbol T_2 si es generador primero en profundidad.

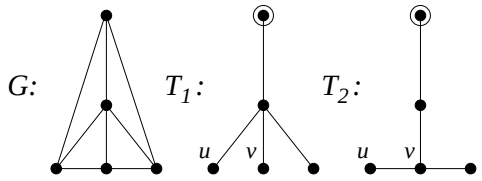

Figura 4.3: Gráfica G junto con dos árboles generadores, uno de los cuales es primero en profundidad.

Las aristas pertenecientes al conjunto E_T, el conjunto de aristas de T, serán llamadas *aristas de árbol*; mientras que las aristas de $(E_T)^c$, es decir, las aristas de G

que no están en T se llamarán *aristas ciclo*. Una arista ciclo uv será denotada por $u \leftrightsquigarrow v$.

Una etiquetación de los vértices de un árbol T con raíz r en *preorden*, es aquella producida por el siguiente algoritmo.

Algoritmo 8 Preorden

Requiere: árbol T y $r \in V_T$ la raíz
1: inicialice todas las etiquetas a vacías
2: aplique el Procedimiento 9 a r

Donde el Procedimiento 9 es el siguiente

Procedimiento 9

Requiere: vértice v de un árbol
1: asigne a v un número mayor a los asignados previamente
2: **para cada** w tal que $v \rightarrow w$ **haga**
3: aplique el Procedimiento 9 a w
4: **fin para**

Nótese que la etiquetación en preorden de los vértices de un árbol, puede ser obtenida durante la aplicación del Algoritmo Búsqueda Primero en Profundidad al asignar una etiqueta mayor a la asignada previamente cada que se marca como visitado cada uno de los vértices. De aquí en adelante, siempre que tengamos un árbol generador primero en profundidad etiquetado en preorden, supondremos que proviene de aplicar el Algoritmo 1.

Dado un árbol T con raíz, y $w \in V_T$, $w^\star$ denotará el conjunto de descendientes de w, en símbolos esto quiere decir

$$w^\star = \{u \in V_T | w \xrightarrow{\star} u\}.$$

A continuación enunciaremos un lema que nos será útil posteriormente.

Lema 4.1. *Sea G una gráfica 2-conexa y T un árbol generador primero en profundidad de G. Si $w \in V_T$, con $H_w = \{u \in V_T | w \rightarrow u\}$ el conjunto de hijos de w entonces $w^\star = \{w\} \cup \Big(\bigcup_{u \in H_w} u^\star \Big)$, donde para cada u y $v \in H_w$ distintos, tenemos que $u^\star \cap w^\star = \emptyset$. Además no es posible que $u_0 \leftrightsquigarrow v_0$ con u_0 y v_0 descendientes de dos hijos diferentes de w.*

Demostración. Es claro que $\{w\} \cup \Big(\bigcup_{u \in H_w} u^\star \Big) \subseteq w^\star$. Sea $s \in w^\star$, si $s = w$ no hay nada que demostrar. De suponer lo contrario, es decir, $s \neq w$, existe un paseo $v_1 v_2 \dots v_k$ en

T, de $w = v_1$ a $s = v_k$, pero $v_2 \in H_w$ y consecuentemente $s \in v_2^\star$, lo cual concluye la demostración de la igualdad.

Mostremos ahora que los conjuntos son disjuntos a pares. Como $w \notin H_w$, es claro que $\{w\} \cap u^\star = \emptyset$ con $u \in H_w$. Por lo anterior, basta demostrar que para dos vértices distintos u y $v \in H_w$ se tiene que $u^\star \cap v^\star = \emptyset$. Por contradicción, supongamos que para $u \neq v$ se tiene que $u^\star \cap v^\star \neq \emptyset$; sea $x \in u^\star \cap v^\star$, como $x \in u^\star$ existe un paseo $u_1 u_2 \dots u_k$ en T entre $u = u_1$ y $x = u_k$ y como $x \in v^\star$ existe un paseo $v_1 v_2 \dots v_l$ en T entre $v = v_1$ y $x = v_l$; note que ninguno de estos dos paseos pasa por w, pero el camino $u_1 u_2 \dots u_k v_{l-1} v_{l-2} \dots v_1$ junto al único paseo que une a u con v en T tiene al menos un ciclo como subgráfica, ya que se trata de un camino cerrado donde al menos dos aristas no se repiten, a decir uw y vw. Lo anterior contradice el hecho de que T es árbol, y por lo tanto $u^\star \cap v^\star = \emptyset$ para u y $v \in H_w$ distintos.

Finalmente probemos que para u y $v \in H_w$ distintos, no puede ocurrir que existen $u_0 \in u^\star$ y $v_0 \in v^\star$ tales que $u_0 \leftrightsquigarrow v_0$. Procedamos por contradicción, supongamos que existen $u_0 \in u^\star$ y $v_0 \in v^\star$ tales que $u_0 \leftrightsquigarrow v_0$. Como T es árbol generador primero en profundidad se sigue que $u_0 \xrightarrow{\star} v_0$ ó $v_0 \xrightarrow{\star} u_0$, lo cual implica que son descendientes del mismo hijo de w, contradiciendo la hipótesis y por lo tanto concluyendo lo que se deseaba demostrar. □

Sea G una gráfica y T un árbol generador primero en profundidad de G con raíz r cuyos vértices han sido etiquetados del 1 al n en preorden, e identifiquemos a los vértices por su etiqueta. Para cada vértice a de G definimos

$$L(a) = \text{mín}(\{a\} \cup \{p \mid \exists\, q \text{ tal que } a \xrightarrow{\star} q \text{ y } q \leftrightsquigarrow p\})$$

Ejemplo. Sean G y T las gráficas que se muestran en la Figura 4.4. Obsérvese que T es un árbol generador primero en profundidad de G con raíz r. Cada vértice de T está etiquetado en preorden, recuerde que en este caso $r = 1$. Claramente $L(1) = 1$. Por otra parte $L(2) = 1$, ya que $3 \leftrightsquigarrow 1$ y $3 \in 2^\star$, con un argumento análogo se concluye que $L(3) = 1$. Argumentando de manera similar, se concluye que $L(4) = L(5) = 1$, ya que $5 \leftrightsquigarrow 1$. Observando a los descendientes de 6 (incluyéndolo a él mismo), se concluye que $L(6) = 3$ y que $L(7) = 4$. Finalmente, $L(8) = 2$.

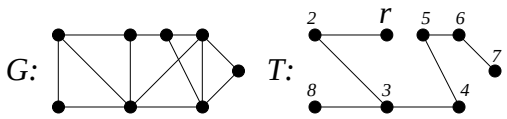

Figura 4.4: Gráficas G y T. El árbol T es generador primero en profundidad de G.

Como demostraremos más adelante (Corolario 5.3), en caso de que una gráfica G sea 2-conexa, en todo árbol generador primero en profundidad de G la raíz r de dicho árbol tendrá grado 1. En [8] Tarjan establece el siguiente resultado, aquí se presenta una demostración alternativa.

Lema 4.2. *Si G es una gráfica 2-conexa, T un árbol generador primero en profundidad de G y $v \to w$, entonces $v \neq 1$ implica $L(w) < v$, y $v = 1$ implica $L(w) = v = 1$.*

Demostración. Supongamos que $v = 1$, en este caso v es la raíz. Claramente el grado de v en G es mayor que 1 porque G es 2-conexa. Sea $u \in N(v) \setminus \{w\}$. Del Corolario 5.3 se tiene que $u \in w^\star$, y por lo tanto $L(w) = 1$.

Ahora, supongamos que $v \neq 1$ y procedamos por contradicción, es decir, supongamos que $L(w) \geq v$. Como T fue obtenido de aplicar Búsqueda Primero en Profundidad a G, para todo antecesor a de v se tiene que $a < v$, ya que durante la construcción de T, los ancestros de v fueron visitados más tempranamente que él y en consecuencia obtuvieron un número en preorden menor que v. De la suposición y observación anterior se sigue que ningún descendiente de w puede ser adyacente a algún antecesor de v, salvo al mismo v. Sea $w_0 \in w^\star$ y supongamos que existe $u_0 \in V_T$ tal que $w_0 \leftrightsquigarrow u_0$. Necesariamente ocurre que $u_0 \xrightarrow{\star} w_0$ ó $w_0 \xrightarrow{\star} u_0$. Si ocurre que $u_0 \xrightarrow{\star} w_0$, entonces u_0 no puede ser ancestro de v (salvo v), de lo contrario tendríamos que $L(w) < v$. En caso de que $w_0 \xrightarrow{\star} u_0$ se tiene que $u_0 \in w^\star$. De lo anterior se concluye que todas las aristas de G que inciden en algún vértice de $w^\star$ necesariamente inciden en otro vértice de $w^\star$ o en v, de no ser así tendríamos que $L(w) < v$; por lo tanto $H = G \setminus \{v\}$ es disconexa, ya que una de sus componentes conexas es $H[w^\star]$, es decir, la subgráfica de H generada por $w^\star$. Lo anterior contradice la hipótesis de que G es 2-conexa, y por lo tanto no puede ocurrir que $L(w) \geq v$, es decir, necesariamente ocurre que $L(w) < v$. □

4.2. Algoritmo

El objetivo de esta sección es presentar un algoritmo que obtiene una numeración st de una gráfica 2-conexa, este algoritmo junto con un procedimiento y sus respectivas demostraciones es presentado en [4]. Distinguiremos a uno de los vértices de la gráfica 2-conexa y lo llamaremos t. El algoritmo consta de tres partes fundamentales. La primera de ellas se encarga de obtener un árbol generador primero en profundidad T cuya raíz es t, durante este mismo proceso se etiquetan sus vértices en preorden y se obtiene el valor de $L(v)$ para cada $v \in V_T$.

La segunda parte del algoritmo es un procedimiento encargado de encontrar paseos en G llamado Pathfinder. Al inicio de nuestro algoritmo, se marcan como visitados la raíz t, su único hijo que denotaremos por s y la arista entre ellos. En la ejecución inicial Pathfinder(s) encuentra un paseo en G entre s y el vértice distinguido t que consta de aristas y vértices que inicialmente estaban marcados como no visitados y que después de la ejecución del procedimiento fueron marcados como visitados. En ejecuciones posteriores, Pathfinder se encarga de encontrar un paseo en G *no visitado* entre dos vértices v y w que ya habían sido visitados. A continuación se presenta el procedimiento mencionado.

Procedimiento 10 Pathfinder

Requiere: vértice v de un árbol primero en profundidad de una gráfica 2-conexa

1: inicialice paseo p a vacío
2: **si** hay una arista ciclo vw con $w \xrightarrow{\star} v$ **entonces**
3: etiquete la arista vw como visitada
4: agregue vw a p
5: **regrese** p
6: **si no si** hay una arista no visitada a algún hijo w de v **entonces**
7: etiquete la arista vw como visitada
8: agregue vw a p
9: **mientras** w es no visitado **haga**
10: encuentre una arista ciclo no visitada wx con $x = L(w)$ o una arista del árbol no visitada wx con $L(x) = L(w)$
11: etiquete el vértice w y la arista wx como visitados
12: agregue wx a p
13: $w := x$
14: **fin mientras**
15: **regrese** p
16: **si no si** hay una arista ciclo no visitada vw con $v \xrightarrow{\star} w$ **entonces**
17: etiquete la arista vw como visitada
18: agregue vw a p
19: **mientras** w es no visitado **haga**
20: encuentre la arista no visitada de árbol wx tal que $x \to w$
21: etiquete el vértice w y la arista wx como visitados
22: agregue wx a p
23: $w := x$
24: **fin mientras**
25: **regrese** p
26: **si no**
27: **regrese** p
28: **fin si**

Lema 4.3. *Suponga que los vértices s, t y la arista st fueron etiquetados como visitados. Entonces en aplicaciones sucesivas de* Pathfinder*, si algún vértice está etiquetado como visitado implica que todos sus ancestros están etiquetados de esa forma.*

Demostración. Procedamos por inducción sobre el número de aplicaciones de Pathfinder. Es claro que en la primera llamada a Pathfinder se cumple. Supongamos ahora que el argumento se cumple para las primeras k llamadas sucesivas a Pathfinder, mostremos que se cumple para la llamada $k+1$. En la llamada $k+1$ ocurrió alguno de los cuatro casos en que se divide el procedimiento Pathfinder. En caso de que se haya tratado del primer o último caso, no hay nada que demostrar, ya que el conjunto de vértices etiquetados como visitados no cambió. Supongamos ahora que se cumplió la segunda condición del procedimiento Pathfinder. En este caso, toda una cadena de descendientes de v fueron marcados como visitados, y como la condición para que cada vértice fuera marcado como visitado es que su padre ya estuviera marcado como visitado, pues se sigue inmediatamente que el padre de todo vértice marcado como visitado ya está marcado como visitado. Un argumento análogo demuestra el caso en el que se cumple la tercera condición en el procedimiento Pathfinder. □

Lema 4.4. *Suponga que los vértices s, t y la arista st inicialmente se etiquetaron como visitados. La llamada inicial* Pathfinder(s) *regresará un paseo de s a t que no contiene la arista st. Una llamada sucesiva a* Pathfinder(v)*, con v etiquetado como visitado, regresará un paseo cuyas aristas eran inicialmente no visitadas entre v y un vértice etiquetado como visitado w; lo anterior en caso de que haya alguna arista no visitada incidente a v, de lo contrario el procedimiento regresa el paseo vacío.*

Demostración. Recordemos primero que la etiqueta en preorden de s y t es 2 y 1 respectivamente. Claramente la llamada inicial Pathfinder(s) regresa un paseo de s a t que no contiene la arista st. Lo anterior es porque del Lema 4.2 se sigue que todos los hijos u de s son tales que $L(u) < s = 2$, es decir, $L(u) = 1$. Tomando en cuenta lo anterior y sabiendo que se cumplirá la segunda condición del Procedimiento 10, Pathfinder se encargará de encontrar un paseo de aristas inicialmente marcadas como no visitadas entre s y t.

Para demostrar la parte restante del lema consideremos los cuatro posibles casos en que se divide el Procedimiento 10. En el primer caso o en el último, el procedimiento en cuestión realiza la tarea prevista. Analicemos cada uno de los casos restantes por separado.

En caso de que se cumpla la segunda condición; tenemos, por el Lema 4.2, que $L(w) < v$, donde vw es la primera arista del paseo. Cada arista que se agrega al paseo, salvo la última, es una arista de árbol. La última arista del paseo es incidente a algún antecesor de v etiquetado como visitado, por el Lema anterior. Por lo tanto, Pathfinder obtiene un paseo como se había previsto.

Finalmente, si se cumplió la tercera condición en Pathfinder, quiere decir que no se cumplió la segunda condición. De lo anterior se sigue que todos lo hijos de v ya

están etiquetados como visitados. La arista inicial del paseo incide a algún descendiente de v, mientras que las demás son aristas de árbol que ascienden hasta algún descendiente de v, diferente de v. Lo anterior obtiene un paseo con las características deseadas y concluye la demostración. □

La tercera parte del algoritmo que produce una numeración st utiliza el Procedimiento 10 para lograr lo deseado. A continuación mostramos el algoritmo.

Algoritmo 11 Numeración st

Requiere: gráfica G 2-conexa

1: sea T un árbol generador primero en profundidad de G con raíz t
2: sea $s \in V_T$ tal que $t \rightarrow s$
3: etiquete a s, t y st como visitados y a todos los demás vértices y aristas como no visitados
4: sea P una pila vacía
5: $push(t, P)$
6: $push(s, P)$
7: $i := 1$
8: **mientras** $|P| > 0$ **haga**
9: $v := pop(P)$
10: sea $v_1 v_2 \dots v_k$ el paseo regresado por Pathfinder(v)
11: **si** $k > 0$ **entonces**
12: **para** $j = 1 \dots k-1$ **haga**
13: $push(v_{k-j}, P)$
14: **fin para**
15: **si no**
16: defina el número st de v como i
17: $i := i + 1$
18: **fin si**
19: **fin mientras**

Teorema 4.5. *El Algoritmo 11 anterior produce una numeración st de una gráfica 2-conexa G.*

Demostración. Observemos que ningún vértice v de G llega a aparecer más de una vez en la pila P. Esto es porque al agregar los vértices del paseo obtenido de la ejecución de Pathfinder no se agregan todos, sólo los que antes de la ejecución de Pathfinder estaban etiquetados como no visitados y el que fue retirado de la pila en la primera línea del ciclo **mientras**. Observemos también que ningún vértice v de G es removido permanentemente de P hasta que todas las aristas incidentes a él han sido etiquetadas como visitadas, esto resulta de las aplicaciones sucesivas de

Pathfinder a dicho vértice. Utilizaremos esos dos hechos para demostrar que todos los vértices w de G son colocados en P antes de que t sea removido de la pila y que la numeración asignada a los vértices de G es en efecto una numeración st.

Demostremos que todo vértice de G es colocado en P. Sea $w \in V_G$. Si $w = s$ es claro que s es colocado en P antes de que se remueva t. Si $w \neq s$ entonces, por la 2-conexidad de G, existe un paseo $u_1 u_2 \dots u_k$ que no pasa por t entre $s = u_1$ y $w = u_k$. Sea u_j el vértice del paseo con mayor j que es colocado dentro de la pila antes de que t sea removido. Supóngase que $u_k \neq u_j$, es decir, $u_k = w$ es colocado en P después de que t es removido. El vértice u_j será borrado permanentemente de P antes de t, pero esto no puede ocurrir sin que u_{j+1} sea agregado a la pila. Esta contradicción implica que $u_j = u_k$. Y por lo tanto cada vértice $w \in V_G$ es colocado en P antes de que se borre t.

Sabiendo que todo vértice de G es colocado en P concluimos que todo vértice recibe un número; probemos que ese número pertenece a una numeración st válida. Inicialmente s es colocado en P y también es el primer elemento de P que se borra de manera permanente y por lo tanto s recibe el número 1. También sabemos que t es el último vértice de G que permanece en P y por lo tanto recibe el número n, de esto se sigue que los vértices cuyos números st son el 1 y el n son adyacentes. Ahora sea $w \in V_G$, $w \neq s, t$. Cuando w es agregado a P, w forma parte de un paseo $u_1 u_2 \dots u_k$, es decir, $w = u_i$, para algún $u_i \neq u_1, u_k$. Pero los vértices u_{i-1} y u_{i+1} son colocados en la pila encima y debajo de w respectivamente, esto quiere decir que u_{i-1} y u_{i+1} recibirán un número st menor y mayor al de w respectivamente, concluyendo que la numeración st obtenida es válida. □

El algoritmo presentado en este capítulo formará parte del algoritmo que se presenta en el siguiente capítulo, siendo éste una de sus componentes cruciales.

A continuación desarrollaremos un ejemplo del funcionamiento del Algoritmo 11.

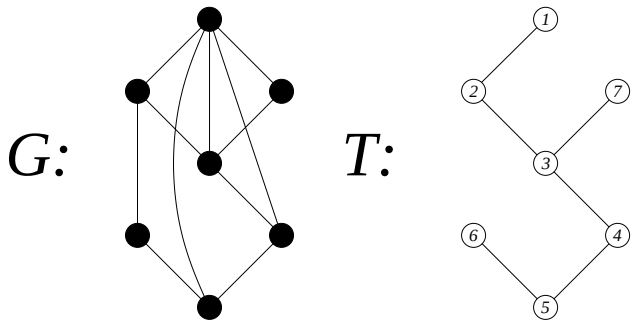

Figura 4.5: Gráficas G y T del ejemplo.

Ejemplo. Sean G y T las gráficas que se muestran en la Figura 4.5, en este caso, T es un árbol generador primero en profundidad de G, con raíz 1. Notemos que $L(i) = 1$ para todo $i \in \{1, 2, 3, 4, 5, 7\}$, pero $L(6) = 2$.

A continuación mostramos la sucesión de gráficas junto con la pila que genera la aplicación del Algoritmo 11. En cada una de las figuras, los vértices marcados como visitados aparecen rellenados con cuadrícula, mientras que las aristas visitadas aparecen con línea punteada. De aquí en adelante haremos referencia a los vértices por su numeración en preorden.

En la primera gráfica (Figura 4.6), se puede observar que los vértices s y t, junto con la arista entre ellos se marcan como visitados. También se incorporan s y t a la pila, es decir, 2 y 1. En la siguiente gráfica se ha marcado como visitado el paseo que se obtuvo de aplicar Pathfinder(s), a decir, 2 3 7 1. También se han incorporado los vértices del paseo recién obtenido a la pila en orden inverso, y excluyendo al último de ellos, como se especifica en la línea 12 del Algoritmo 11.

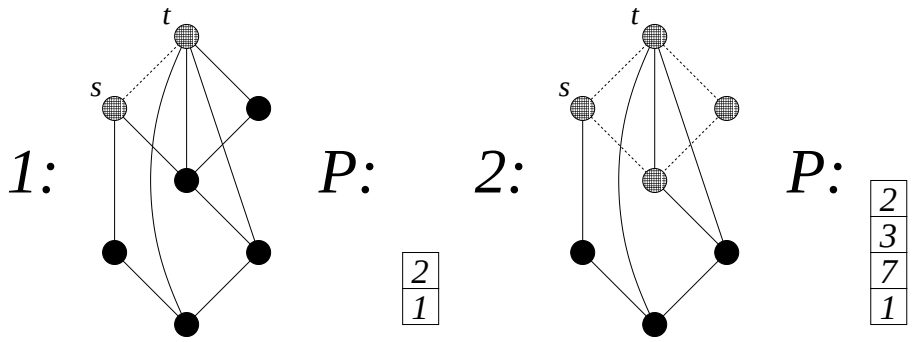

Figura 4.6: Ejemplo de aplicación del Algoritmo 11. Pasos 1 y 2.

En la tercera gráfica obtenida de aplicar el Algoritmo 11 se ha marcado como visitado el paseo obtenido de aplicar Pathfinder(s) por segunda vez. El paseo que es marcado como visitado es 2 6 5 4 3. Asimismo se incorporan, de manera adecuada, los vértices de este paseo a la pila. La gráfica recién descrita se muestra en la Figura 4.7. En la cuarta gráfica, el aplicar Pathfinder a s regresó el paseo vacío, teniendo como consecuencia que se asignara al vértice 2 el número st 1.

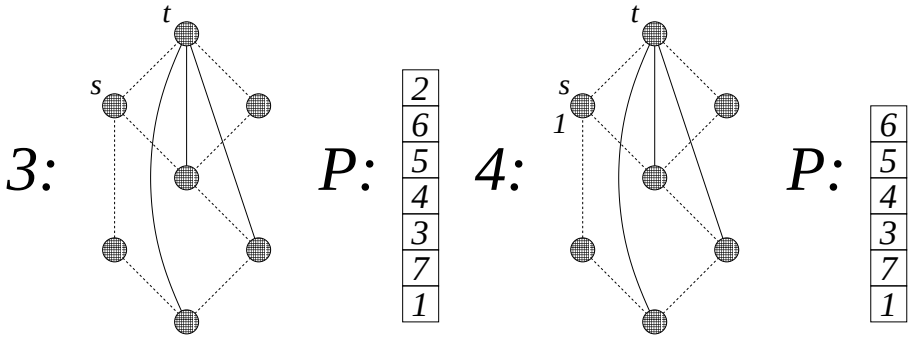

Figura 4.7: Ejemplo de aplicación del Algoritmo 11. Pasos 3 y 4.

En la Figura 4.8 se muestran la quinta y la sexta gráfica obtenida al continuar con la aplicación del algoritmo. En este caso, la quinta gráfica, el vértice número 6 recibe el número *st* 2, debido a que Pathfinder(6) regresa el camino vacío. En la sexta gráfica se marca como visitado la arista 5 1.

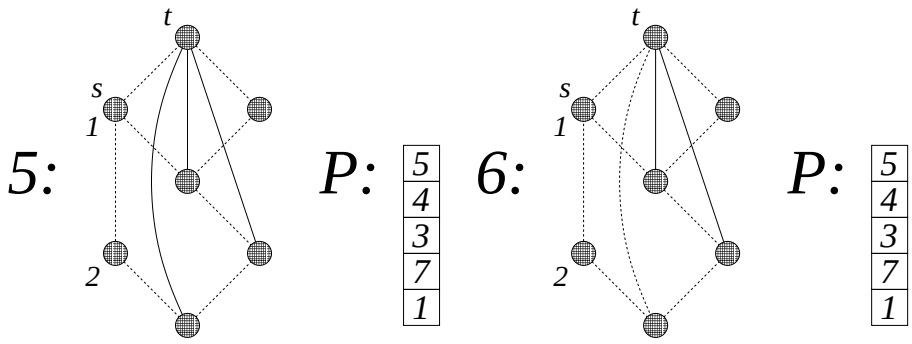

Figura 4.8: Ejemplo de aplicación del Algoritmo 11. Pasos 5 y 6.

Notemos que en la sexta gráfica todas las aristas incidentes a 5 ya están marcadas como visitadas, y por lo tanto, en la séptima gráfica el vértice 5 recibirá su número *st*, en este caso es el 3.

En la séptima y octava gráfica (Figura 4.9) se muestra que al aplicar Pathfinder(4) la primera vez, éste regresa el paseo 1 4; la segunda vez regresa el paseo vacío, obteniendo así el vértice 4 el número *st* 4, como se muestra en la Figura 4.10.

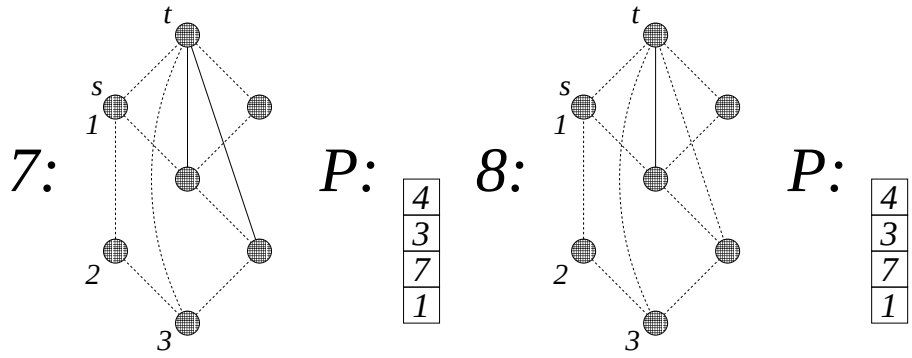

Figura 4.9: Ejemplo de aplicación del Algoritmo 11. Pasos 7 y 8.

En las gráficas de la Figura 4.10 se muestra, de manera similar a lo recién ocurrido con el vértice 4, que la primera aplicación de Pathfinder(3) regresa el paseo 1 3; mientras que la segunda aplicación de Pathfinder a 3 regresa el paseo vacío y por lo tanto, en la onceava gráfica, el vértice 3 obtiene el número *st* 5.

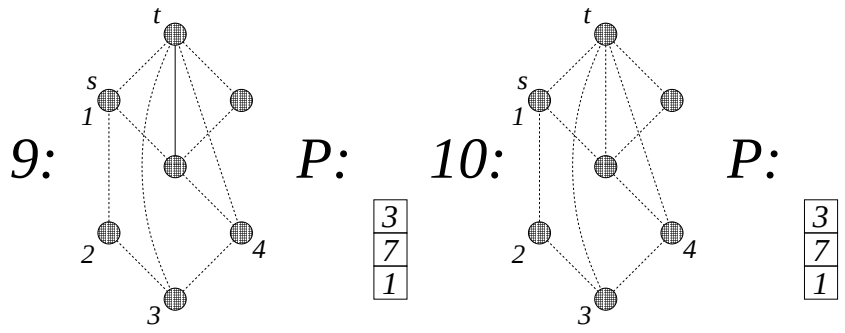

Figura 4.10: Ejemplo de aplicación del Algoritmo 11. Pasos 9 y 10.

Se puede observar en la onceava gráfica, Figura 4.11, que el vértice por procesar es el número 7. Todas las aristas incidentes a 7 ya fueron visitadas y por lo tanto el vértice 7 obtiene el número *st* 6.

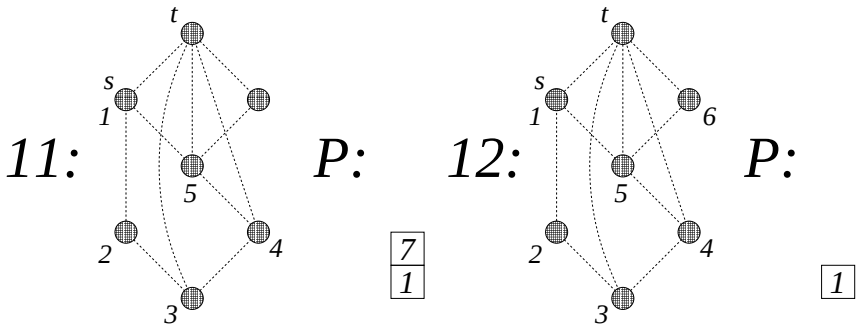

Figura 4.11: Ejemplo de aplicación del Algoritmo 11. Pasos 11 y 12.

Finalmente el vértice 1 obtiene el número *st* 7 y la numeración *st* queda como se muestra en la Figura 4.12.

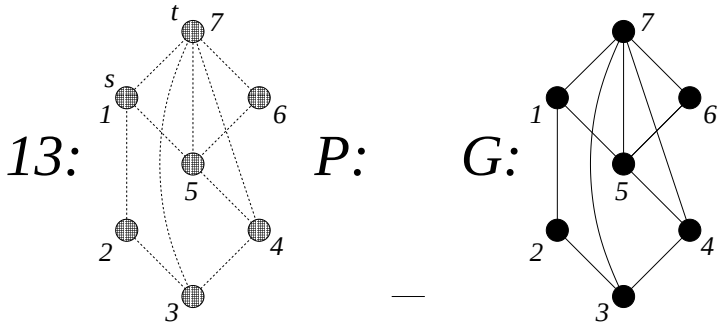

Figura 4.12: Ejemplo de aplicación del Algoritmo 11. Paso 13 y resultado final.

Capítulo 5

Planaridad

En algunas ocasiones resulta de cierto interés determinar si una gráfica dada se puede dibujar en el plano sin que sus aristas se corten. Por ejemplo, las personas encargadas del diseño de circuitos electrónicos deben crear dichos circuitos de tal forma que las franjas conductoras no pase una encima de otra ya que esto puede producir resultados no deseados. Una posible solución a este problema es el asociar una gráfica al circuito en cuestión y checar si existe una manera de dibujar la gráfica en el plano sin que se corten sus aristas [9]. La pregunta acerca de cómo se caracterizan este tipo de gráficas fue respondida por un teorema de Kasimir Kuratowski en el año de 1930 [6].

En este capítulo presentamos un algoritmo que determina planaridad de gráficas dos conexas. También se presenta un algoritmo basado en el recién mencionado que determina planaridad de una gráfica arbitraria.

5.1. Introducción

Definición. Diremos que una gráfica G es *planar* si existe una representación de G en el plano tal que sus aristas no se intersecan, donde cada arista es representada por una curva simple de Jordan. Diremos que G es una *gráfica plana* si G ya se encuentra representada por un diagrama en el que no se intersecan sus aristas.

Ejemplo. En la Figura 5.1 se muestran dos representaciones planas de K_4. En la Figura 1.3 (página 5) se puede observar otra representación plana de K_4

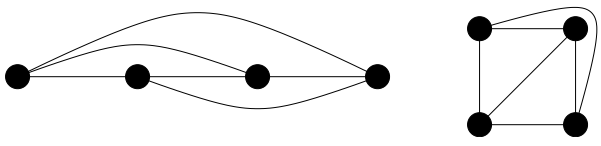

Figura 5.1: Dos representaciones planas de K_4

Observemos que si una gráfica *G* es planar entonces toda subgráfica de *G* lo será. Es también claro que *G* es planar si y sólo si cada una de sus componentes conexas lo es. Así que para determinar planaridad basta probar que cada componente conexa de una gráfica lo es. En *A Textbook of Graph Theory* [2, pág. 156] se puede encontrar una demostración del resultado que se enuncia a continuación.

Teorema 5.1. *Una gráfica G es planar si y solamente si cada una de sus componentes 2-conexas es planar.*

Utilizando el teorema anterior, el problema de checar planaridad de una gráfica se reduce a checar la planaridad de cada una de las componentes 2-conexas de una gráfica dada, es por esto que de aquí en lo que sigue de este capítulo, sólo consideraremos gráficas 2-conexas, a menos que se especifique lo contrario.

Para checar la planaridad de una gráfica 2-conexa utilizamos el algoritmo desarrollado por Booth y Lueker [3]. Este algoritmo requiere del uso de una estructura de dato especial llamada árbol *PQ* que describiremos a continuación.

5.2. Árboles *PQ*

Un árbol *PQ* es un árbol con una raíz, cuyos nodos pueden ser de uno de tres diferentes tipos; a decir, los nodos de tipo *P*, los de tipo *Q* y las hojas. Para lograr explicar de una manera más sencilla el funcionamiento de esta estructura de dato, representaremos los nodos de tipo *P* por círculos, los de tipo *Q* mediante rectángulos y las hojas simplemente tendrán una etiqueta. Todos los nodos internos del árbol serán de tipo *P* ó *Q*, mientras que todas los vértices colgantes serán hojas.

Ejemplo. En la Figura 5.2 se muestra el diagrama de un árbol *PQ*.

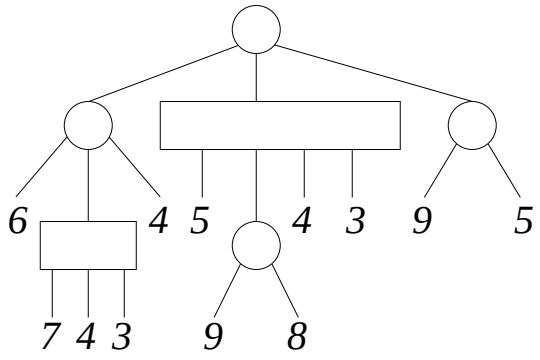

Figura 5.2: Un árbol *PQ*

La manera en que determinaremos planaridad de una gráfica será aplicando diez plantillas de reducción, que en realidad representan movimientos permitidos dentro de los nodos de un árbol PQ. Las plantillas de reducción serán aplicadas a un subconjunto dado de las hojas del árbol, así como a algunos de los ancestros de dichas hojas. Por definición, las hojas pertenecientes al conjunto dado son *llenas*, mientras que las que no pertenecen al conjunto son *vacías*. En el contexto de los árboles PQ, diremos que un conjunto dado de nodos es *adyacente* si forman un bloque de hijos consecutivos de un nodo. Por ejemplo, en la Figura 5.2, el nodo P que es padre de las hojas 9 y 8 junto con las hojas 5 y 4 que son hijas del nodo Q central, forman un bloque de hijos consecutivos y por lo tanto diremos que son adyacentes. Más adelante veremos que la propiedad de adyacencia sólo es relevante para los hijos de un nodo Q, ya que para los nodos P, cualquier subconjunto de hijos será adyacente.

El objetivo de las plantillas es lograr que todas estas hojas sean adyacentes mediante la aplicación repetida de diversas plantillas. Diremos que un nodo interno es *lleno* si todos sus descendientes son nodos llenos; de manera análoga si todos los hijos de un nodo son vacíos, diremos que el nodo es *vacío*. Si tenemos un nodo de tipo Q cuyos hijos llenos sean adyacentes, uno de ellos se encuentre en un extremo y además no todos sus hijos sean llenos diremos que dicho nodo Q es *parcial*, por definición sólo los nodos de tipo Q pueden ser de tipo parcial.

Definición. Llamaremos *raíz pertinente* del árbol con respecto a un subconjunto de hojas dado, a aquel nodo que sea el primer ancestro común de todas las hojas del subconjunto. Para los casos en que sea necesario distinguir a la raíz pertinente, el nodo correspondiente a la raíz pertinente estará marcado con una r.

Ejemplo. Sea T el árbol PQ de la Figura 5.3 y sea

$$S = \{\text{hojas de } T \text{ cuya etiqueta es 3, 4, 6 ó 7}\}$$

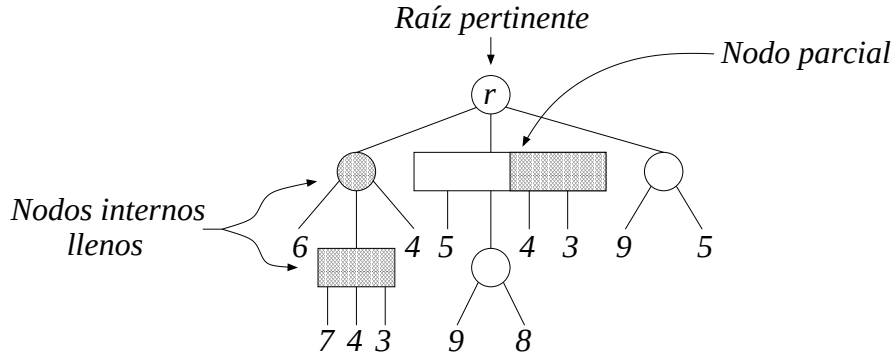

Figura 5.3: Ejemplo de raíz pertinente, nodo lleno y nodo parcial.

En la misma figura de la definición de T se pueden observar los nodos internos que son llenos, parciales y además la raíz pertinente con respecto a S.

Hay algunas reglas básicas que rigen el comportamiento de los árboles PQ. Una de ellas es que los hijos de los nodos de tipo P pueden permutarse de forma arbitraria (por esto, cualquier subconjunto de hijos de un nodo P es adyacente), mientras que los hijos de los nodos de tipo Q solamente pueden voltearse, es decir, el orden en el que aparecen no debe ser alterado. Otra de las reglas básicas es que un nodo P debe tener al menos dos hijos y un nodo Q debe tener al menos tres. Las reglas recién mencionadas serán utilizadas en la descripción de las plantilas sin ser mencionadas, es decir, supondremos que los nodos están acomodados de cierta forma salvo movimientos producidos por las reglas recién mencionadas.

En la Figura 5.3 se muestra como estarán representados gráficamente los nodos. Los nodos llenos estarán iluminados, mientras que los nodos vacíos no tendrán ningún tipo de relleno, asimismo, los nodos parciales tendrán un iluminado que los distinga. En lo que sigue del capítulo, los triángulos representarán subárboles, en caso de que algún triángulo esté iluminado, querrá decir que todo el subárbol consta de nodos llenos.

A continuación se procede a describir cada una de las plantillas.

Plantilla L_1 Esta plantilla es la que se aplica a las hojas llenas, el reemplazo es la misma hoja, es decir no se lleva a cabo ningún cambio en la estructura del árbol, solamente se marca como llena la hoja.

Plantilla P_1 Si en algún momento tenemos un nodo de tipo P cuyos descendientes sean todos llenos, este nodo P se vuelve nodo lleno, sin cambiar la estructura del árbol.

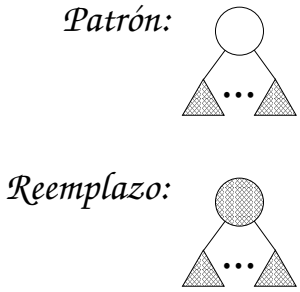

Figura 5.4: Plantilla P_1

Plantilla P_2 Analicemos ahora el caso en el que el nodo en cuestión sea un nodo de tipo P que además sea raíz pertinente y no todos sus hijos sean llenos, pero si al menos uno. En este caso se crea un nuevo nodo de tipo P que será hijo del nodo en cuestión, y cuyos hijos serán aquellos hijos llenos del nodo que se está procesando.

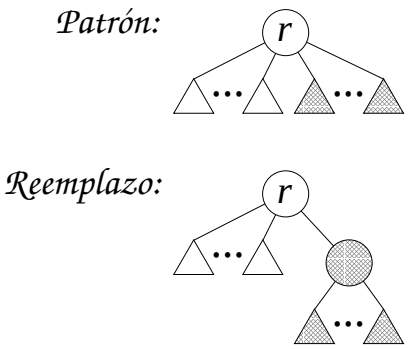

Figura 5.5: Plantilla P_2

Plantilla P_3 Veamos el caso cuando el nodo que se está procesando cumple con las mismas hipótesis que en el caso anterior, salvo que ahora no es raíz pertinente del árbol. Esta plantilla consiste en cambiar el tipo de nodo a Q y anexar a este exactamente dos hijos de tipo P, uno lleno y uno vacío. Los hijos de los nodos lleno y vacío serán los hijos llenos y vacíos del nodo en cuestión respectivamente.

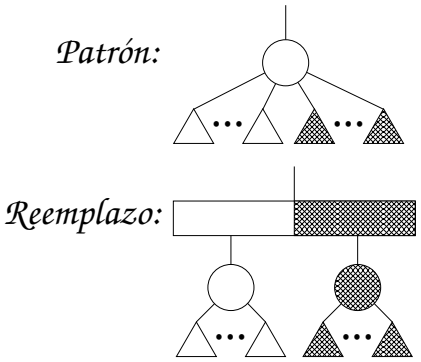

Figura 5.6: Plantilla P_3

En la plantilla anterior y en las posteriores, los nuevos nodos de tipo P que se agregarán sólo serán necesarios cuando vayan a tener al menos dos hijos, de lo contrario, no hará falta agregar un nuevo nodo de tipo P. En la Figura 5.7 se

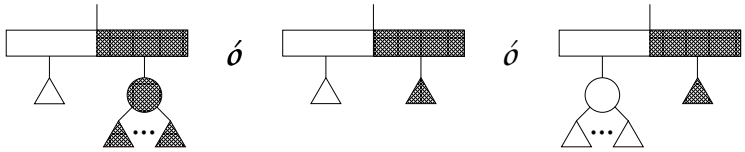

Figura 5.7: Reemplazos alternativos para la plantilla P_3

muestran los reemplazos alternativos para el caso de la plantilla P_3 según sea el caso.

Plantilla P_4 Esta plantilla se aplicará cuando el nodo procesado sea un nodo de tipo P que tenga exactamente un hijo parcial; y que además el nodo procesado sea la raíz pertinente. En este caso se procede a crear un nuevo nodo de tipo P agregándolo como hijo del nodo parcial incorporándolo en el extremo del nodo parcial en el que se encuentra un nodo lleno. Los hijos del nuevo nodo P serán los hijos llenos del nodo que se esta procesando actualmente, esto se muestra en la Figura 5.8.

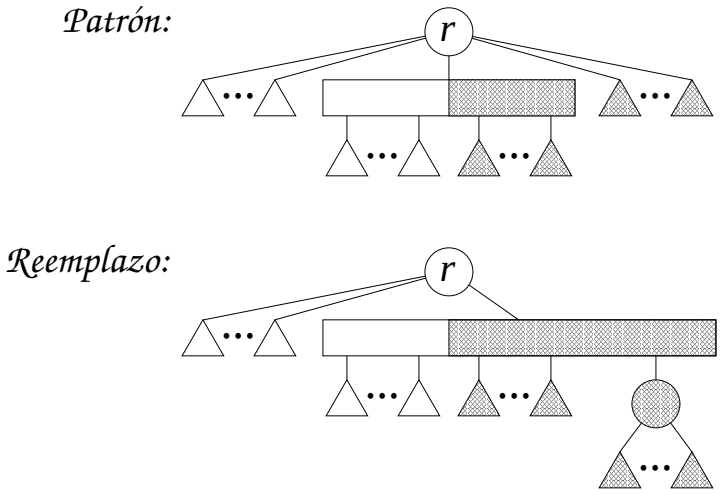

Figura 5.8: Plantilla P_4

Plantilla P_5 Para el caso en el que tenemos un caso similar al anterior, salvo que ahora el nodo procesado no sea raíz pertinente, aplicaremos esta plantilla. Lo que procede es sustituir al nodo procesado por el nodo parcial; agregando al nodo parcial dos nodos nuevos de tipo P. Uno de los nodos nuevos será lleno y tendrá como hijos a todos los hijos llenos del nodo que fue sustituido. El nuevo nodo P lleno se encontrará en el extremo del nodo parcial en el que se encuentra un nodo lleno. El otro nodo nuevo será vacío, y de forma análoga, sus hijos serán todos los hijos vacíos del nodo que fue reemplazado. De forma completamente análoga, el nuevo nodo vacío será agregado en el extremo del nodo parcial en el que se encuentra un nodo vacío.

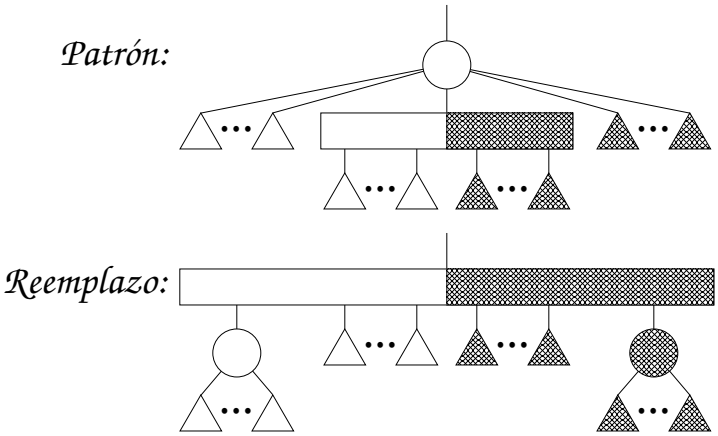

Figura 5.9: Plantilla P_5

Plantilla P_6 Esta es la última de las plantillas en las que el nodo procesado es de tipo P. Esta plantilla se aplica cuando el nodo en cuestión tiene exactamente dos hijos parciales y además es la raíz pertinente. En este caso lo que procede es fusionar los dos nodos parciales para formar uno solo. La manera en que se fusionan los nodos parciales es juntando los dos extremos de los nodos parciales correspondientes a nodos llenos. En caso de que el nodo que se está

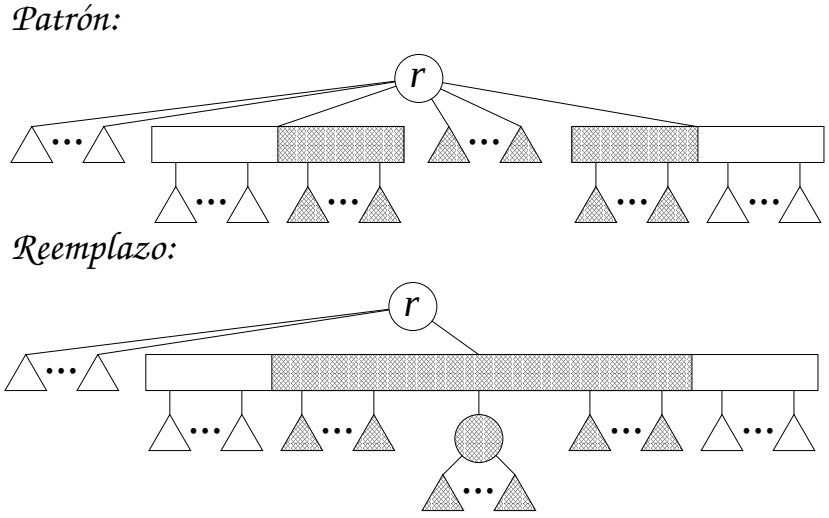

Figura 5.10: Plantilla P_6

procesando cuente con hijos llenos, estos se agregarán como hijos de un nuevo nodo P. Este nuevo nodo se incorporará como hijo del nodo resultante de la fusión, y se ubicará justamente en medio de los hijos de los dos nodos que se fusionaron previamente.

Ahora procedemos a describir las plantillas en las que el nodo procesado es de tipo Q.

Plantilla Q_1 Esta plantilla es una analogía completa a la P_1.

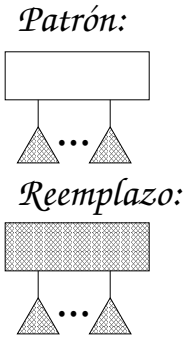

Figura 5.11: Plantilla Q_1

Plantilla Q_2 Esta plantilla se aplica cuando el nodo procesado tiene a lo más un hijo parcial. En caso de no tenerlo, no se realiza ninguna modificación. En caso de tener un hijo parcial, todos los nodos a la derecha (o izquierda) del nodo parcial deben ser llenos, y viceversa. Lo que sucede en este caso es que el nodo que se está procesando absorbe a su hijo parcial, es decir, los hijos del nodo parcial pasan a ser hijos del nodo actualmente procesado, de tal forma que todos los nodos llenos queden de un solo lado y todos los nodos vacíos queden del lado opuesto.

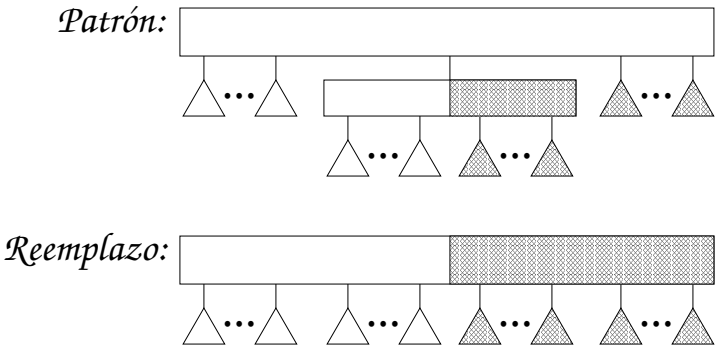

Figura 5.12: Plantilla Q_2

Plantilla Q_3 Esta plantilla sólo se aplica cuando el nodo en cuestión es de tipo Q, es raíz pertinente, tiene dos hijos parciales y en caso de tener hijos llenos, estos deben ser consecutivos y encontrarse entre los nodos parciales. La estructura

del nodo debe ser equivalente a la que se observa en la Figura 5.13 para poder aplicar esta plantilla. En este caso, el reemplazo consiste en que el nodo procesado absorbe a sus dos hijos parciales de tal forma que todos los nodos llenos sean adyacentes.

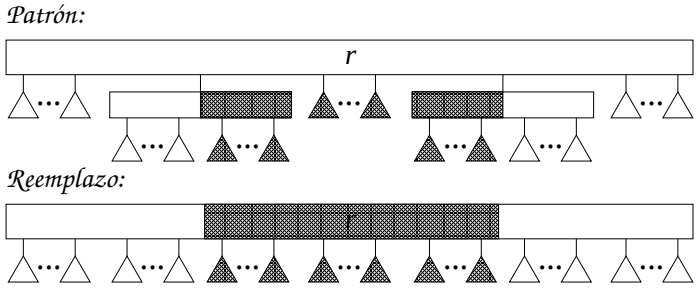

Figura 5.13: Plantilla Q_3

Si en algún momento durante el proceso de reducción sobre un conjunto dado y antes de haber procesado la raíz pertinente se encuentra un patrón que no coincide con el descrito en alguna de las plantillas, entonces la gráfica en cuestión resulta ser no planar.

5.3. Algoritmo

A continuación se presentan los procedimientos y el algoritmo que determina planaridad de una gráfica, para más detalles y la demostración de porqué el algoritmo es válido, véase [3].

Procedimiento 12 Construye una hoja o un nodo P para aplicar reducción

Requiere: gráfica 2-conexa G con numeración st, entero $1 \leq i \leq n$
sea v el vértice de G tal que $st(v) = i$
sea $NS := \{u \in N(v) : st(u) > st(v)\}$
si $|NS| = 1$ **entonces**
  regrese nueva hoja cuya etiqueta es el número st del vecino de v en NS
fin si
cree un nuevo nodo de tipo P
para cada $u \in NS$ **haga**
  anexe una nueva hoja, cuya etiqueta sea el número $st(u)$, como hijo del nodo P
fin para
regrese P

El Procedimiento 12 construye una hoja o un nodo P a partir de una gráfica G 2-conexa con numeración st conocida y un número dado. Primeramente se ubica el vértice v de G cuyo número st es el número dado. Después se considera el conjunto NS de vecinos de v cuyo número st es mayor que el de v. Si NS sólo contiene un elemento, entonces el procedimiento regresa una nueva hoja cuya etiqueta es el número st del vértice que se encuentra en NS. Si NS contiene al menos 2 elementos, se genera un nuevo nodo de tipo P y finalmente se le agregan a este nodo una hoja por cada elemento de NS, con etiqueta correspondiente al número st de cada vértice de NS. Notemos que separamos el caso en que NS solamente tiene un elemento, ya que el generar un nodo P con un solo hijo no tiene sentido, recordando que los nodos de tipo P deben tener al menos 2 hijos. Finalmente resaltemos que cuando el Procedimiento 12 es llamado con el número n, este regresará un nuevo nodo P sin hijos. Lo anterior se debe a que en este caso NS es el conjunto vacío. Esto solo se lleva a cabo para verificar que el proceso de reducción terminó de manera exitosa.

Ejemplo. Sea G la gráfica de la Figura 5.14. Al aplicar Procedimiento 12 a (G,1) obtenemos el nodo P que se observa en la misma figura donde se define G. En este caso $NS = \{2,3,4,8\}$ y por lo tanto el nodo P obtenido tiene como hijos a 4 hojas cuyas etiquetas son los elementos de NS. Notemos que en caso de haber aplicado Proce-

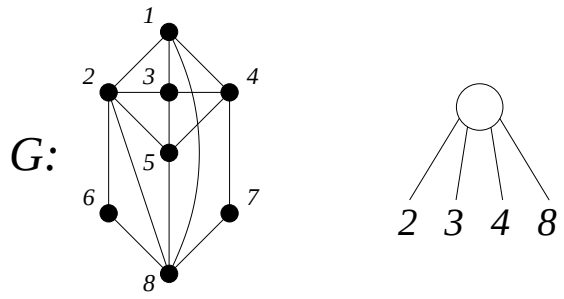

Figura 5.14: Resultado de aplicar Procedimiento 12 a (G,1).

dimiento 12 a (G,5) el resultado obtenido hubiera sido una sola hoja, dado que en este otro caso $NS = \{8\}$.

En el siguiente algoritmo, se hará referencia a los vértices de G mediante su número st. Recordemos que por definición, al aplicar reducción sobre un conjunto dado, las hojas que pertenecen al conjunto son llenas. A partir de esto, se puede observar que la aplicación repetida de las plantillas sobre los árboles resultantes genera nuevos nodos y modifica la estructura del árbol PQ.

Algoritmo 13 Checa Planaridad de una gráfica dada

Requiere: gráfica 2-conexa G

aplicando el Algoritmo 11 (página 34) obtenga una numeración st de G
sea $root$ el nodo P obtenido de aplicar Procedimiento 12 a $(G,1)$
sea T el árbol PQ obtenido de considerar a $root$ como raíz
para $i = 2 \ldots n$ **haga**
    sea S el conjunto de hojas de T etiquetadas con i
    aplique las plantillas de reducción sobre el conjunto S al árbol T
    sea $nuevo\_nodo$ el nodo obtenido de aplicar Procedimiento 12 a (G,i)
    si la raíz pertinente con respecto a S es un nodo lleno **entonces**
        reemplace la raíz pertinente por $nuevo\_nodo$
    si no si la raíz pertinente con respecto a S es un nodo parcial **entonces**
        reemplace todos los nodos llenos por $nuevo\_nodo$
    si no
        regrese Non Planar
    fin si
fin para
si el árbol PQ consta de un solo nodo P **entonces**
    regrese Planar
si no
    regrese Non Planar
fin si

El Algoritmo anterior determina planaridad dada una gráfica 2-conexa, a continuación se desarrollan un par de ejemplos de como aplicar dicho algoritmo.

Ejemplo. Sea G la gráfica de la Figura 5.15. Podemos observar que G es 2-conexa y además las etiquetas de los vértices forman una numeración st de G válida. En la misma figura donde se define G se muestra el resultado de aplicar Procedimiento 12 a $(G,1)$.

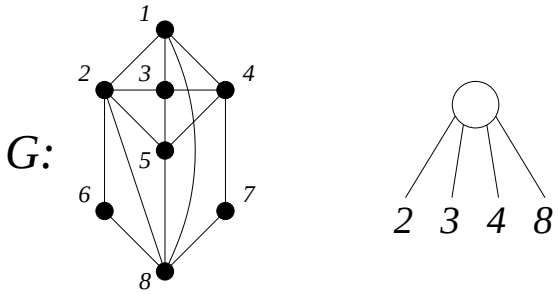

Figura 5.15: Gráfica 2-conexa con numeración st.

El primer paso es construir el conjunto S, en este caso $S = \{2\}$. Lo anterior quiere decir que aplicaremos plantillas de reducción sobre todas las hojas cuya etiqueta es 2. Al ser una única hoja, la única plantilla que es posible aplicar es la L_1, dando por terminado la aplicación de plantillas sobre $\{2\}$; ya que en este caso la raíz pertinente es la misma hoja. Ahora procedemos a aplicar el Procedimiento 12 a $(G, 2)$. El nodo recién obtenido sustituirá a la hoja 2, como se muestra en la Figura 5.16.

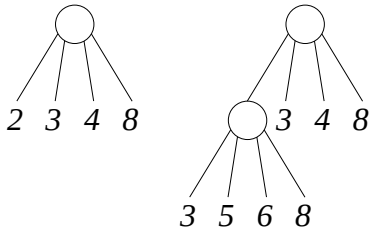

Figura 5.16: Procedimiento para checar planaridad de G (Figura 5.15). Fase 1 y 2.

El siguiente paso es construir un nuevo conjunto S, ahora $S = \{3\}$. De aquí en adelante no mencionaremos la aplicación de la plantilla L_1. La primera plantilla por aplicar ahora es la P_3, esta se aplica al nodo P que recién agregamos. Finalmente aplicamos la plantilla P_4 a la raíz del árbol, que en este caso también es la raíz pertinente con respecto a S. El proceso anterior se ilustra en la Figura 5.17

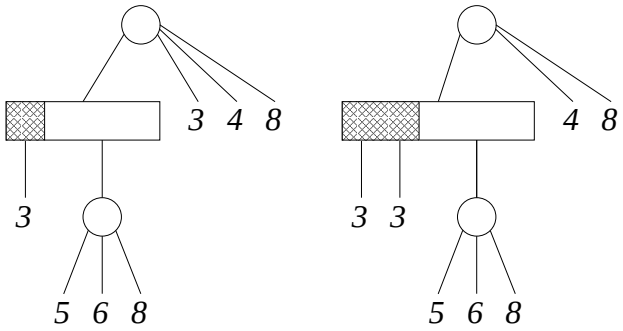

Figura 5.17: Procedimiento para checar planaridad de G (Figura 5.15). Fase 3 y 4.

Ahora procedemos a reemplazar el bloque de nodos adyacentes que son hijos del nodo Q por el nodo P obtenido de aplicar Procedimiento 12 a $(G, 3)$. Obteniendo como resultado el primer árbol PQ que se muestra en la Figura 5.18. Procedamos a aplicar reducción sobre el recién redefinido $S = \{4\}$. La primera plantilla que aplicamos es la P_3, al nodo recién agregado.

La siguiente plantilla que aplicaremos es la Q_2. Finalmente aplicamos la plantilla P_4 a la raíz pertinente con respecto a S, la cual es la raíz del árbol PQ. Lo anterior está ilustrado la Figura 5.19

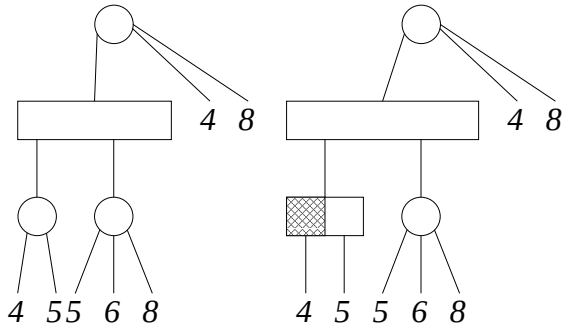

Figura 5.18: Procedimiento para checar planaridad de G (Figura 5.15). Fase 5 y 6.

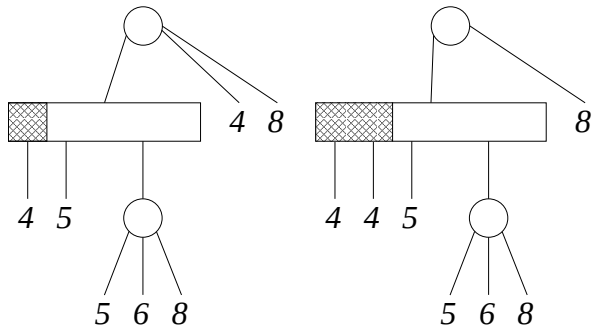

Figura 5.19: Procedimiento para checar planaridad de G (Figura 5.15). Fase 7 y 8.

Aplicamos Procedimiento 12 a $(G, 4)$, obteniendo un nodo P con dos hojas como hijos, cuyas etiquetas son 5 y 7. Definimos $S = \{5\}$. Ahora aplicamos la plantilla P_3 a los dos nodos P que tienen un hijo con etiqueta 5. El resultado de esto se muestra en la Figura 5.20

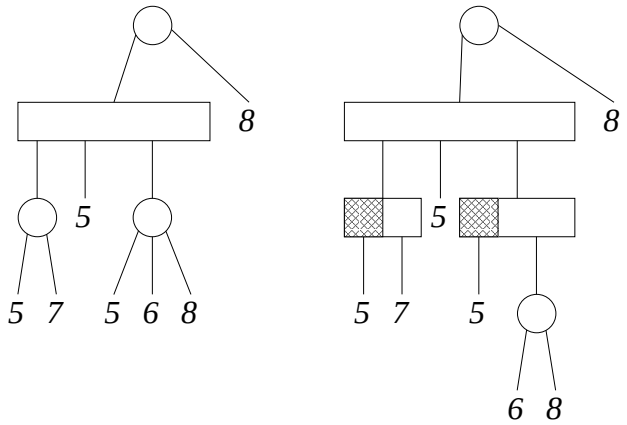

Figura 5.20: Procedimiento para checar planaridad de G (Figura 5.15). Fase 9 y 10.

A continuación aplicamos la plantilla Q_3 a la raíz pertinente del árbol. Esto da como resultado un nodo Q con un bloque de hijos llenos, el cual será reemplazado por el nodo obtenido de aplicar Procedimiento 12 a (G,5). El nodo que reemplazará el bloque será una hoja cuya etiqueta es 8. Esto se ilustra en la Figura 5.21.

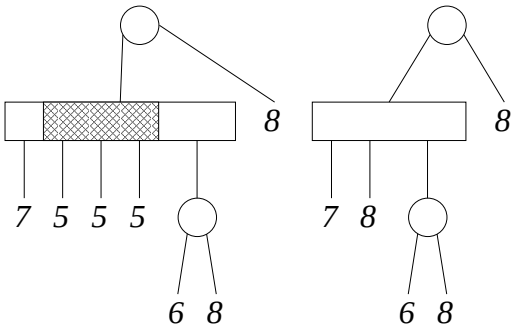

Figura 5.21: Procedimiento para checar planaridad de G (Figura 5.15). Fase 11 y 12.

En los árboles PQ de la Figura 5.22 se muestra como se reemplazan las hojas con etiqueta 6 y 7 respectivamente. El reemplazo es simple, ya que al definir $S = \{6\}$ directamente se substituye la hoja etiquetada con 6 por el resultado de aplicar Procedimiento 12 a (G, 6). Se procede de manera similar para procesar la hoja con etiqueta 7.

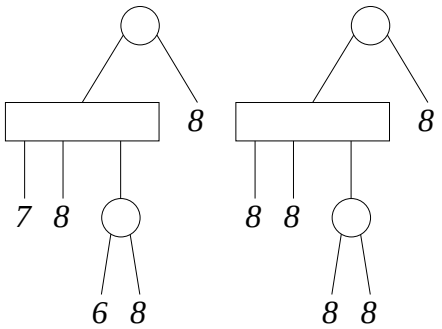

Figura 5.22: Procedimiento para checar planaridad de G (Figura 5.15). Fase 13 y 14.

A continuación se define $S = \{8\}$ y se procede a aplicar reducción sobre S. La primera plantilla que aplicaremos será la P_1 al nodo P que tiene dos hojas como hijos. Seguido de esto aplicamos la plantilla Q_1 al único nodo Q del árbol. Posteriormente aplicamos la plantilla P_1 a la raíz del árbol PQ. Finalmente reemplazamos la raíz pertinente por un nuevo nodo P, que se obtiene de aplicar Procedimiento 12 a (G, 8). El resultado final se puede observar en la Figura 5.23

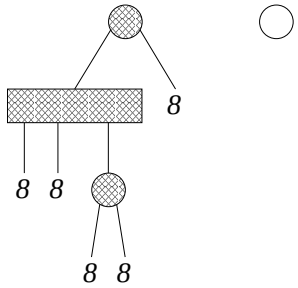

Figura 5.23: Procedimiento para checar planaridad de G (Figura 5.15). Fase 15 y 16.

El Algoritmo 13 regresa Planar; efectivamente, la gráfica G es planar como se muestra en la Figura 5.24

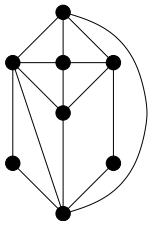

Figura 5.24: Representación plana de G.

A continuación presentamos un ejemplo con una gráfica no planar bien conocida.

Ejemplo. Consideremos a K_5 con la numeración st que se muestra en la Figura 5.25.

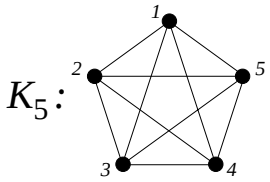

Figura 5.25: Gráfica K_5 con numeración st.

El resultado de aplicar el Procedimiento 12 a 1 se muestra en el primer árbol de la Figura 5.26, éste será nuestro árbol inicial. Aplicando reducción sobre $S = \{2\}$ obtenemos el segundo árbol que se muestra en la Figura 5.26.

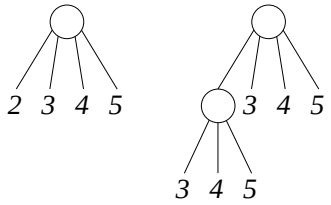

Figura 5.26: Fase 1 y 2 del proceso para checar planaridad de K_5.

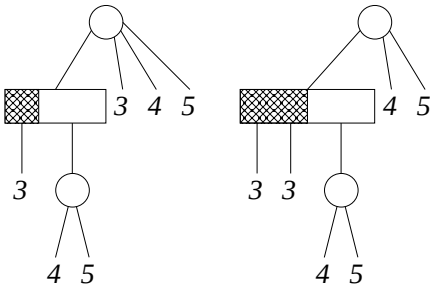

Figura 5.27: Fase 3 y 4 del proceso para checar planaridad de K_5.

Ahora, al aplicar reducción sobre $S = \{3\}$ obtenemos los árboles que se muestran en la Figura 5.27. Las plantillas aplicadas en este caso fueron la P_3 y P_4.

Al sustituir el bloque de hojas etiquetadas con 3 obtenemos el primero de los árboles de la Figura 5.28. En el segundo árbol de la Figura 5.28 se muestra el resultado de aplicar la plantilla P_3 a los dos hijos del nodo Q.

Después de aplicar dos veces la plantilla P_3 ya no es posible aplicar alguna otra. La plantilla que más se aproxima a ser aplicada es la Q_3, pero ésta sólo se puede aplicar en caso de que el nodo en cuestión sea la raíz pertinente. En este caso la raíz pertinente no es el nodo Q, sino la raíz del árbol completo. Por lo anterior se concluye que la gráfica en cuestión no es planar, es decir, K_5 no es planar.

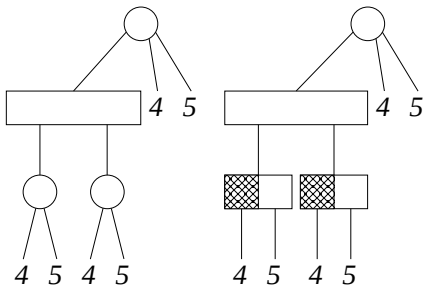

Figura 5.28: Fase 5 y 6 del proceso para checar planaridad de K_5

El Algoritmo 13 funciona sólo con gráficas 2-conexas, pero es posible hacer funcionar este algoritmo para gráficas arbitrarias. Basta un procedimiento que obtenga las componentes 2-conexas de una gráfica dada. Para eso, utilizaremos el siguiente lema.

Lema 5.2. *Sea G una gráfica conexa y $v \in V(G)$. Entonces* $\deg_{\mathrm{DFS}(v)}(v) \geq 2$ *si y sólo si v es punto de corte de G.*

Demostración. Sea $T := \mathrm{DFS}(v)$ y supongamos que $\deg_T(v) \geq 2$; tomemos u_1 y $u_2 \in N_T(v)$ y procedamos por contradicción, es decir, supongamos que v no es punto de corte de G. Como v no es punto de corte de G, entonces existe un paseo $u_1 v_2 v_3 \dots v_{k-1} u_2$ entre u_1 y u_2 que no pasa por v. Sin pérdida de generalidad, podemos asumir que u_1 fue visitado más tempranamente que u_2 durante la construcción de T. De lo anterior se sigue que los elementos de $N_T(u_1)$ fueron visitados más tempranamente que u_2, en particular v_2 lo fue. Una aplicación repetida del argumento anterior nos permite concluir que los elementos de $N_T(v_{k-1})$ fueron visitados más tempranamente que u_2, en particular u_2, lo cual es una contradicción. De lo anterior podemos concluir que no existe un paseo entre u_1 y u_2 que no pase por v y por lo tanto v es un punto de corte de G.

Sea $v \in V(G)$ un punto de corte en G y sea $T := \mathrm{DFS}(v)$. Por contradicción, supongamos que $\deg_T(v) \leq 1$, pero es claro que $\deg_T(v)$ no puede ser cero, dado que G es conexa. De lo anterior tenemos que $\deg_T(v) = 1$. Como v es punto de corte de G, entonces podemos tomar u_1 y $u_2 \in V(G)$ tales que todo camino entre u_1 y u_2 pase por v. Utilizando la conexidad de T se sigue que existe un paseo entre u_1 y u_2 en T. Pero v no forma parte de este paseo, ya que es un punto colgante de T. Ahora, como T es subgráfica de G, el paseo en T también es un paseo en G, es decir, es un paseo en G entre u_1 y u_2 que no pasa por v, contradiciendo que v es un punto de corte en G, por lo tanto $\deg_T(v) \geq 2$. □

Corolario 5.3. *Sea r la raíz de un árbol T generador primero en profundidad de una gráfica 2-conexa G. Entonces* $\deg_T(r) = 1$.

Demostración. Claramente $\deg_T(r) > 0$ porque G es 2-conexa. Del Lema anterior se sigue que $\deg_T(r) = 1$, ya que de otra forma r sería punto de corte de G, contradiciendo la hipótesis. □

Utilizando el lema anterior, podemos obtener el siguiente procedimiento, para el cual asumiremos que en la llamada inicial la lista $\mathscr{L}$ es vacía.

Procedimiento 14 Obtiene una lista con las componentes 2-conexas de una gráfica G

Requiere: gráfica G, lista $\mathscr{L}$
  si G es disconexa **entonces**
2: considere la colección $\mathscr{C} := \{C_1, \ldots, C_k\}$, donde cada C_i es el conjunto de vértices en la i-ésima componente conexa de G
  para cada $C \in \mathscr{C}$ **haga**
4: aplique este mismo procedimiento a $G[C]$, $\mathscr{L}$
  fin para
6: **si no si** $n = 1$ ó $n = 2$ ó G es 2-conexa **entonces**
  $insert(G, \mathscr{L})$
8: **si no**
  sea v un vértice de G tal que $\deg_{DFS(v)}(v) > 1$ y sea $T = DFS(v)$
10: considere la colección $\mathscr{C} = \{C_1, \ldots, C_k\}$, donde C_i es el conjunto de vértices en la i-ésima componente conexa de $T \setminus \{v\}$
  para cada $C \in \mathscr{C}$ **haga**
12: aplique este mismo procedimiento a $G[C \cup \{v\}]$, $\mathscr{L}$
  fin para
14: **fin si**

Proposición 5.4. *El Procedimiento 14 inserta las componentes 2-conexas de una gráfica G a una lista dada.*

Demostración. Sea G una gráfica. Si G es disconexa, entonces el Procedimiento será aplicado a cada una de sus componentes conexas, así que en lo que sigue de la demostración podemos asumir que la gráfica G dada es conexa. Si G es 2-conexa, o si es K_1 o K_2, el procedimiento agrega G a la lista, cumpliendo así su objetivo.

Por lo establecido en el párrafo anterior, hemos de suponer que G es conexa, pero que tiene puntos de corte. Por el Lema 5.2 se sigue que existe $v \in V_G$ tal que $\deg_{DFS(v)}(v) > 1$. Sea $T = DFS(v)$, se sigue también del Lema 5.2 que v es punto de corte en T y en G, por lo cual la colección $\mathscr{C}$ definida en la línea 10 tiene al menos dos elementos. Es claro que para cada $C \in \mathscr{C}$, $G[C \cup \{v\}]$ es una gráfica conexa con menos vértices que G. En caso de que $G[C \cup \{v\}]$ sea 2-conexa o sea isomorfa a K_1 ó K_2, el algoritmo habrá logrado su objetivo, de lo contrario, procederíamos a aplicar este procedimiento nuevamente. Este procedimiento genera una sucesión de subgráficas conexas de G con cada vez menos vértices, lo cual garantiza que eventualmente una de esas subgráficas sea 2-conexa o isomorfa a K_1 o K_2 □

Capítulo 6

Cinco Coloración

En el año de 1852 en Inglaterra, el joven Francis Guthrie tenía a la mano un mapa de ese país. Este mapa dividía por condados a Inglaterra. Francis se planteó la siguiente pregunta: ¿Cuál será el número mínimo de colores para colorear el mapa? Entenderemos que colorear un mapa consiste en asignar colores distintos entre regiones adyacentes. Este problema fue planteado al profesor de matemáticas de la universidad de Londres, Augustus De Morgan, por el hermano mayor de Francis. Al profesor De Morgan le pareció bastante razonable que cuatro colores eran suficientes para colorear un mapa, pero a pesar de eso no pudo demostrarlo [9].

En el año de 1878, durante una reunión de la sociedad matemática de Londres, Arthur Cayley planteó el problema y desde entonces se le conoció como el problema de los cuatro colores. Un año más tarde, el matemático amateur Alfred Kempe dio una "demostración" de este problema; pero once años más tarde, Percy Heawood encontró un error dentro de su demostración. A pesar de eso el esfuerzo de Kempe no fue en vano, ya que Heawood logró hacer pequeñas modificaciones a los argumentos de Kempe y así obtuvo el teorema de los 5 colores [9].

Esta pregunta permaneció abierta por más de un siglo. Sin embargo, el 22 de julio de 1976 Kenneth Appel y Wolfgang Haken anuncian públicamente la demostración de la conjetura de los cuatro colores. Esta demostración requería del uso de computadora, y en aquel entonces, la demostración requirió de más de 1000 horas de cómputo [2].

Pero en realidad, ¿qué tiene que ver el problema de los cuatro colores con la teoría de gráficas? La manera de abordar este problema utilizando esta teoría es asociando una gráfica a cada mapa, llamada la gráfica dual. La gráfica dual asociada a un mapa tendrá tantos vértices como regiones el mapa, y dos vértices serán adyacentes si las regiones a las que representan comparten frontera. Ahora el problema consiste en asignar colores a los vértices de esta gráfica de tal forma que cualquier par de vértices adyacentes tengan colores distintos.

En este capítulo se muestra un algoritmo que se encarga de encontrar una cinco coloración de una gráfica planar. El algoritmo esta basado en una demostración del Teorema de Heawood acerca de cinco coloración.

6.1. Algunos resultados sobre 5 colorabilidad

Definición. Sea G una gráfica, una *coloración* de G es una asignación de colores a sus puntos de tal forma que ningún par de vértices adyacentes tenga el mismo color. Diremos que G es *n-coloreable* si existe una *n-coloración* de G, es decir, si existe una coloración con n colores de G. El conjunto de todos los vértices de un solo color es llamado una *clase de color*.

Ejemplo. Sea G la gráfica de la Figura 6.1

Podemos observar que para colorear G requerimos de 5 colores, digamos c_1, c_2, c_3, c_4 y c_5. Una 5-coloración de G consistiría en asignar el color c_i al vértice v_i.

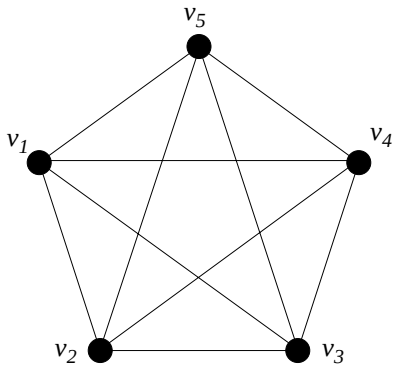

Figura 6.1: Gráfica 5-coloreable

Observemos que K_n es n-coloreable, y no se puede colorear con menos de n colores, ya que todos los vértices son adyacentes entre sí. En este capítulo nos restringiremos al estudio de la coloración de gráficas planares, como se mencionó previamente, Appel y Haken demostraron que toda gráfica planar es 4-coloreable. Nosotros utilizaremos el resultado de Heawood acerca de 5-coloración de una gráfica planar.

En el libro de Balakrishnan y Ranganathan [2, pág. 158] se puede encontrar una demostración del siguiente par de resultados.

Teorema 6.1. *Si G es planar, entonces $\delta(G) \leq 5$.*

Utilizando el teorema anterior, demostraremos el siguiente resultado.

Teorema 6.2 (Teorema de los 5 colores)**.** *Toda gráfica planar es 5 coloreable.*

Demostración. Procedemos por inducción sobre el número de vértices n de una gráfica planar. Es claro que para $n \leq 5$ el resultado se cumple.

Supongamos como hipótesis de inducción que para $n = k$ se puede encontrar una 5 coloración de cualquier gráfica planar con n vértices.

Sea G una gráfica planar con $k+1$ vértices, demostremos que existe una cinco coloración de G. Como G es planar se sigue del Teorema 6.1 que $\delta(G) \leq 5$, y por lo tanto existe un vértice u tal que $\deg u = \delta(G) \leq 5$. Es claro que $G \setminus \{u\}$ es una gráfica planar y con k vértices. Por hipótesis de inducción se sigue que $G \setminus \{u\}$ es 5 coloreable. Asignemos una 5 coloración a $G \setminus \{u\}$, y basándonos en esta coloración asignemos los mismos colores a los vértices correspondientes en G. Ahora solo falta asignarle color a u. Si los vecinos de u utilizan a lo más cuatro colores podemos asignar a u un quinto color sin mayor problema. Ahora, si los vecinos de u utilizan los 5 colores posibles, podemos permutarlos, de ser necesario, de tal forma que los vértices con colores c_1, c_2, c_3, c_4 y c_5 estén ordenados cíclicamente alrededor de u como se muestra en la Figura 6.2.

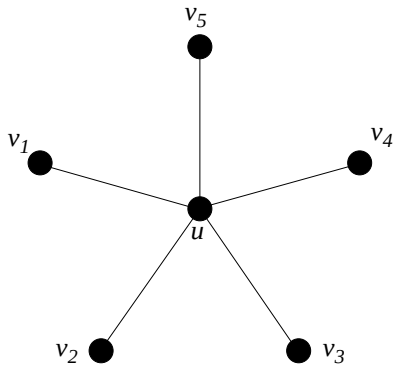

Figura 6.2: El vértice v_i tiene el color c_i

Llamemos $G_{i,j}$ a la subgráfica de G generada por las clases de color $[c_i]$ y $[c_j]$. Si el vértice v_1 y v_3 no se encuentran en la misma componente conexa en $G_{1,3}$, entonces podemos permutar los colores c_1 y c_3 del componente al que pertenece v_1, dejando disponible para u el color c_1. Por otra parte, si v_1 y v_3 se encuentran en la misma componente conexa en $G_{1,3}$, entonces existe un paseo entre ellos. Este paseo, junto con $v_1 u v_3$ forman un ciclo en G. Como G es planar, entonces puede que v_2 haya quedado dentro de este ciclo o puede que v_4 y v_5 hayan quedado dentro de este ciclo. En cualquiera de ambos casos, en $G_{2,4}$ los vértices v_2 y v_4 pertenecen a componentes conexas distintas, por lo que podemos intercambiar los colores c_2 y c_4 en el componente de v_2, dejando disponible así el color c_2 a u. □

6.2. Algoritmo

A continuación damos un algoritmo que produce una cinco coloración de una gráfica planar G.

Algoritmo 15 Obtiene una 5 coloración

Requiere: gráfica planar G
si $n \leq 5$ **entonces**
  asigne colores distintos a cada vértice de G
  regrese G
fin si
sea $u \in V(G)$ tal que $\deg u = \delta(G)$
G_u :=Algoritmo 15($G \setminus \{u\}$)
asigne los colores de G_u a los correspondientes vértices en G
si $N(u)$ utiliza a lo más cuatro colores **entonces**
  asigne un color no utilizado a u
si no
  para cada $u_1, u_2 \in N(u)$ **haga**
    sean i, j el color de u_1, u_2 respectivamente
    sea $T := \mathrm{DFS}_{G_{i,j}}(u_1)$
    si $u_2 \notin T$ **entonces**
      intercambie los colores i y j en T en la componente conexa de u_1
      asigne el color i a u
      regrese G
    fin si
  fin para
fin si

Proposición 6.3. *Sea G una gráfica planar. El Algoritmo 15 produce una 5-coloración de G.*

Demostración. Basta demostrar que para cada vértice u de G, el color de u es distinto al de cada uno de sus vecinos. Notemos que el Algoritmo 15 es recursivo, y el caso base es cuando la gráfica tiene a lo más 5 vértices. Es claro también que el algoritmo funciona correctamente para gráficas con a lo más 5 vértices.

Ahora, sea G una gráfica planar con más de 5 vértices y sea $u \in V(G)$. Supongamos, por contradicción, que existe $v \in N(u)$ tal que u y v tienen el mismo color. Esto no es posible si estos dos vértices nunca fueron removidos por el algoritmo, es decir, si estos dos vértices permanecieron en la subgráfica del caso base que consistía de 5 vértices; ya que a estos vértices siempre se les asigna inicialmente 5 colores distintos.

Supongamos ahora que alguno de los vértices adyacentes con el mismo color fue removido durante la ejecución del algoritmo. Podemos asumir, sin pérdida de

generalidad, que u fue removido de G antes que v. El paso recursivo de este algoritmo consiste en asignar una 5-coloración a la gráfica G menos un vértice. Sea G_u la primera gráfica generada por el algoritmo en la que no aparece el vértice u. La gráfica G_u recibe una cinco coloración. Justo después se agrega nuevamente el vértice u junto con las aristas incidentes a él; se sigue de esto que el grado de u en esta subgráfica es a lo más 5. Notemos además que $v \in N(u)$. En caso de que sus vecinos utilicen a lo más 4 colores, podemos asignar sin problema el quinto color a u, en cuyo caso v tiene inicialmente un color distinto al de u. El otro caso por analizar es cuando el grado de u es cinco y sus vecinos utilizan cada uno de los cinco colores disponibles. Pero en este caso, el Teorema 6.2 garantiza que podemos intercambiar dos colores en uno de los componentes de $G_{i,j}$ de tal forma que en G quede un color disponible para u, y esto se lleva a cabo mediante una búsqueda exhaustiva en el ciclo **para cada** de la línea 11, esto garantiza que v tendrá inicialmente un color distinto al de u.

Los colores de u y v permanecerán con colores distintos, ya que en el peor de los casos sus colores solo se llegan a intercambiar. En consecuencia observamos que el haber asumido que u y v tienen colores iguales nos lleva a una contradicción. Por lo tanto, no es posible que dado un vértice, alguno de sus vecinos tenga el mismo color que éste. □

Apéndice

En este apéndice describiremos de manera breve las estructuras de datos utilizadas a lo largo de este escrito.

Listas

Una *lista* L es una sucesión que consta de cero o más elementos de un cierto tipo. El número de elementos contenidos en L se denota por $|L|$. Denotaremos a los elementos contenidos en L por una sucesión de elementos separados por comas y encerrados entre llaves

$$\{v_1, v_2 \ldots, v_k\}$$

. En caso en que la sucesión no contenga elementos, la lista será llamada *vacía*. Si $k \geq 1$ diremos que v_1 es el primer elemento, mientras que v_k será llamado el último elemento.

Existen diferentes operaciones que se pueden llevar a cabo con listas, solo enunciaremos las que nos serán de utilidad para nuestros fines. Para más detalles puede consultar [1].

Sea L una lista. La función $insert(u, p, L)$ insertará el elemento u en la posición p de L, desplazando a los elementos cuyos índices son mayores que p, es decir, si $L = \{v_1, v_2 \ldots, v_k\}$ después de ejecutar la función recién mencionada obtenemos $L = \{v_1, v_2 \ldots, v_{p-1}, u, v_p, v_{p+1}, \ldots, v_k\}$. En caso de que el parámetro p se omita entenderemos que el elemento u se insertará al final de la lista, es decir, $insert(u, L)$ produciría $L = \{v_1, v_2 \ldots, v_k, u\}$. La función $retrieve(p, L)$ regresa el elemento que se encuentra en la posición p de L. La función $delete(p, L)$ elimina el elemento de L que se encuentra en la posición p, provocando que el resto de los elementos en L sean desplazados.

Pilas

Una *pila P* es un tipo especial de lista, donde solamente podremos ejecutar una versión restringida de las funciones para listas. Las funciones que utilizaremos serán $pop(P)$ y $push(u,P)$. Si $P = \{v_1, v_2 \ldots, v_k\}$ es una pila, ejecutar $pop(P)$ resulta en $P = \{v_2 \ldots, v_k\}$, regresando el elemento v_1; lo anterior es equivalente a ejecutar $retrieve(1,P)$ seguido de $delete(1,P)$. La aplicación de la función $push(u,P)$ resulta en $P = \{u, v_1, v_2 \ldots, v_k\}$; en términos de listas esta operación es equivalente a $insert(u,1,P)$. La función pop se encarga de regresar el primer elemento de la lista siendo eliminado de ella; por otra parte la función $push$ se encarga de insertar un elemento al inicio de la lista. Podemos imaginar una pila como una pila de libros. El apilar un libro produce que quede encima de todos los demás. Al remover un libro de la pila siempre tiene que ser el libro que se encuentra encima de los demás.

Colas

Una *cola C* es un tipo especial de lista también. Las funciones que operarán en las colas tendrán el mismo nombre que las de las pilas, solo que algunos de los efectos serán distintos. La función $pop(C)$ tiene el mismo efecto que en el caso de las pilas. La única función que difiere es $push(u,C)$. Si $C = \{v_1, v_2 \ldots, v_k\}$ es una cola, entonces $push(u,C)$ tiene el mismo efecto que $insert(u,C)$ (tratando a C como a una lista), es decir, el resultado sería $C = \{v_1, v_2 \ldots, v_k, u\}$. A diferencia de la pila, la cola puede imaginarse como una fila en la caja de un supermercado. La persona que se incorpora (inserta) a la fila es colocada al final de ella. Por otra parte, la persona a ser atendida (removida de la cola) es la que lleva más tiempo en ella, es decir, la primera.

Cuando se utilice alguna estructura de dato E que contenga vértices de gráficas, denotaremos al conjunto de vértices contenidos en E como $V(E)$.

Para más detalles acerca del funcionamiento de estructuras de datos véase [1].

Bibliografía

[1] A. AHO, J. HOPCROFT AND J. ULLMAN. *Data Structures and Algorithms.* Addison-Wesley, 1983.

[2] R. BALAKRISHNAN AND K. RANGANATHAN. *A Textbook of Graph Theory.* Springer, New York, 2000.

[3] K. S. BOOTH AND G. S. LUEKER. *Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms.* Journal of Computer and System Sciences, **Vol 13**, 335–379, 1976.

[4] S. EVEN AND R. TARJAN. *Computing an st-numbering.* Theoretical Computer Science, **Vol. 2**, 339–344, 1976.

[5] A. GIBBONS. *Algorithmic Graph Theory.* Cambridge University Press, 1985.

[6] F. HARARY. *Graph Theory.* Addison-Wesley, 1969.

[7] A. LEMPEL, S. EVEN, AND I. CEDERBAUM. *An algorithm for planarity testing of graphs.* Theory of Graphs International Symposium, pages 215–232, 1966.

[8] R. TARJAN. *Depth-first search and linear graph algorithms.* SIAM J. Comput., **Vol. 1**, 146–160, 1972.

[9] R. J. WILSON AND J. J. WATKINS. *Graphs: An Introductory Approach.* John Wiley and Sons, 1990.