Algunos algoritmos heurísticos para la coloración de vértices en grafos

María José Moreno Casallas
Vanessa Zea Parra

Universidad Pedagógica Nacional
Facultad de Ciencia y Tecnología
Licenciatura en Matemáticas
Bogotá D.C.
2026

Algunos algoritmos heurísticos para la coloración de vértices en grafos

María José Moreno Casallas
Vanessa Zea Parra

Trabajo de grado presentado como requisito para optar

por el título de Licenciadas en Matemáticas

Asesora:
Haydee Jiménez Tafur
Profesora del Departamento de Matemáticas

Universidad Pedagógica Nacional
Facultad de Ciencia y Tecnología
Licenciatura en Matemáticas
Bogotá D.C.
2026

Agradecimientos

Queremos agradecer a nuestras familias por su paciencia, comprensión y apoyo incondicional a lo largo de este proceso.

También agradecemos a la profesora Haydee, cuya guía, dedicación y acompañamiento fueron esenciales en el desarrollo de este proyecto; su disposición, consejos y compromiso hicieron que este proceso fuera enriquecedor tanto para nuestra formación académica como personal.

Además, extendemos nuestro agradecimiento al Seminario de Álgebra por brindarnos la oportunidad de participar, fortalecer nuestras habilidades y motivarnos a llevar a cabo nuestro trabajo de grado.

Índice general

Introducción

La teoría de grafos constituye una de las áreas fundamentales de la matemática discreta y desde sus orígenes en el siglo XVIII, con el célebre problema de los puentes de Königsberg planteado por Euler, ha experimentado un notable desarrollo. Los grafos se han constituido en una herramienta matemática para modelar situaciones relacionadas con redes de carreteras, redes de fibra óptica para conectar edificios de un campus universitario, representar la competencia por el alimento de diferentes especies en un nicho ecológico, modelar torneos entre equipos o relaciones entre personas (redes sociales), representar rutas de transportes, entre otras.

Dentro de la gran diversidad de problemas que existen en teoría de grafos, hay algunos relacionados con la optimización, como el problema de la coloración de vértices en grafos no dirigidos, donde buscamos asignar colores a los vértices de manera que dos vértices conectados por una arista deben tener un color diferente. El problema radica en encontrar la menor cantidad de colores necesarios para cumplir la condición; este valor se conoce como el número cromático del grafo.

El estudio de la coloración de grafos tiene su origen en el conocido problema de los cuatro colores, el cual plantea que cualquier mapa puede colorearse utilizando como máximo cuatro colores, de manera que las regiones vecinas no tienen el mismo color. Este problema puede modelarse dentro de la teoría de grafos, donde un mapa se representa mediante un grafo cuyos vértices se corresponden con las regiones y las aristas indican si dos regiones son vecinas. De esta forma, la coloración del mapa equivale a la coloración de los vértices del grafo asociado.

Durante muchos años, diversos investigadores intentaron solucionar el problema sin éxito; sin embargo, sus contribuciones permitieron enriquecer la teoría de coloración de grafos. Finalmente, en 1976, Kenneth Appel y Wolfgang Haken lograron demostrar el teorema de los cuatro colores con

ayuda de un ordenador; la demostración fue controvertida, ya que se cuestionó su validez debido a la imposibilidad de verificarla de manera manual.

A partir del estudio de la conjetura de los cuatro colores, se desarrolló una extensión del problema a grafos simples no dirigidos, llamado coloreo de vértices en un grafo. Aunque no se tiene una respuesta teórica para establecer el número cromático de cualquier grafo, se han presentado resultados para algunas familias de grafos, como los completos, bipartitos, lineales, ciclos, entre otros. Además, se cuenta con teoremas que proporcionan cotas inferiores y superiores para el número cromático como, por ejemplo, el Teorema de Brooks, el cual menciona que, en un grafo simple, conexo, no completo y con máximo grado mayor o igual a tres, el número cromático es menor o igual que el máximo grado de sus vértices (Gross et al., 2019, p. 360).

Aunque podemos pensar que para encontrar el número cromático de un grafo basta con enumerar las coloraciones y buscar una que utilice la cantidad mínima de colores, hacer esta tarea puede resultar inviable cuanto más grande sea la cantidad de vértices y aristas de un grafo. Es decir, este algoritmo desde el punto de vista computacional resulta no ser eficiente, esto es, el tiempo requerido para ejecutar operaciones elementales crece demasiado rápido a medida que el tamaño de los datos de entrada aumenta (cantidad de vértices y/o aristas), haciendo el problema de coloreado de grafos difícil (Guerequeta & Vallecillo, 2000). Por ello, se han diseñado algoritmos heurísticos, cuyos pasos están guiados por una heurística o reglas basadas en la experiencia, con las cuales se pretende resolver el problema planteado, pero sin tener necesariamente una demostración que garantice que la salida final del algoritmo resuelva satisfactoriamente el problema, es decir, no son exactos, pero nos brindan soluciones aceptables en tiempos adecuados.

Nuestro trabajo de grado retoma el estudio de la coloración de grafos desde un enfoque teórico con especial énfasis en el estudio de algunos algoritmos heurísticos para la coloración de vértices en grafos y lo desarrollamos en cuatro capítulos.

En el capítulo 1, presentamos nociones básicas de teoría de grafos que nos sirven para estudiar la coloración de vértices, como qué es un grafo, las relaciones básicas entre los vértices de los grafos, representaciones, subgrafos, operaciones entre grafos, la conectividad y algunas familias.

En el capítulo 2, nos enfocamos en el estudio de algunas cotas inferiores y superiores para el número cromático de un grafo y en determinar de forma teórica el número cromático asociado a algunas familias de grafos.

En el capítulo 3, nos centramos en el estudio de cuatro algoritmos heurísti-

cos que seleccionamos, iniciando con la descripción de cada algoritmo, proporcionando ejemplos de su empleo e indicando el código de su programación en Python. Además, realizamos un experimento computacional con el fin de comparar dichos algoritmos atendiendo a ciertos criterios establecidos.

Finalmente, en el capítulo 4, mencionamos algunas aplicaciones de la coloración de vértices en grafos con la solución de sudokus y la organización de horarios.

Además, resaltamos el uso de las tecnologías digitales que empleamos en el desarrollo de este trabajo, como *LaTeX* que nos permite tener una presentación adecuada de las expresiones matemáticas y diagramas. Asimismo, las representaciones gráficas de los grafos las realizamos manualmente en la página Graphonline (2015 – 2026). También usamos la plataforma Google Colab para ejecutar los códigos de Python para crear grafos, obtener información de estos, programar los algoritmos heurísticos y realizar el experimento computacional; para ello, estudiamos la librería *NetworkX* (Hagberg et al., 2008), consultando su página oficial NetworkX Developers (2025). A su vez, usamos la inteligencia artificial ChatGPT (OpenAI, 2026) como herramienta de apoyo para la programación de los algoritmos heurísticos en el lenguaje de Python, a partir de los pseudocódigos que creamos y la librería que le adjuntamos a la misma.

Objetivos

Objetivo General

Nuestro objetivo en el presente trabajo es estudiar algunos algoritmos heurísticos para la coloración de vértices en grafos.

Objetivos específicos

- Hacer una revisión del origen del problema de coloración de grafos, su extensión a grafos simples no dirigidos; además, establecer una notación y definiciones básicas.
- Estudiar algunos algoritmos heurísticos que han surgido en la historia.
- Estudiar algunas aplicaciones de la coloración de grafos.

Capítulo 1

Nociones Básicas

En este capítulo presentamos algunas definiciones, teoremas, lemas y notaciones necesarias para el estudio de algunos algoritmos heurísticos para la coloración de vértices en grafos. Los resultados que exponemos son adaptaciones de los presentados en los textos de Gross et al. (2019), West (2005) y Chartrand y Zhang (2012). Así mismo, complementamos los elementos teóricos con ejemplos, y al finalizar el capítulo, incluimos algunas líneas de código en Python para programar el ingreso de grafos, su representación gráfica y obtener elementos característicos del grafo.

Para resaltar nuestro estudio en este capítulo, presentamos un diagrama (Figura 1.1) en el que ubicamos todos los conceptos que introducimos para abordar tanto los algoritmos presentados en el capítulo 3, como algunos resultados teóricos relacionados con la coloración de grafos que planteamos en el capítulo 2, como lo es el Teorema de Brooks.

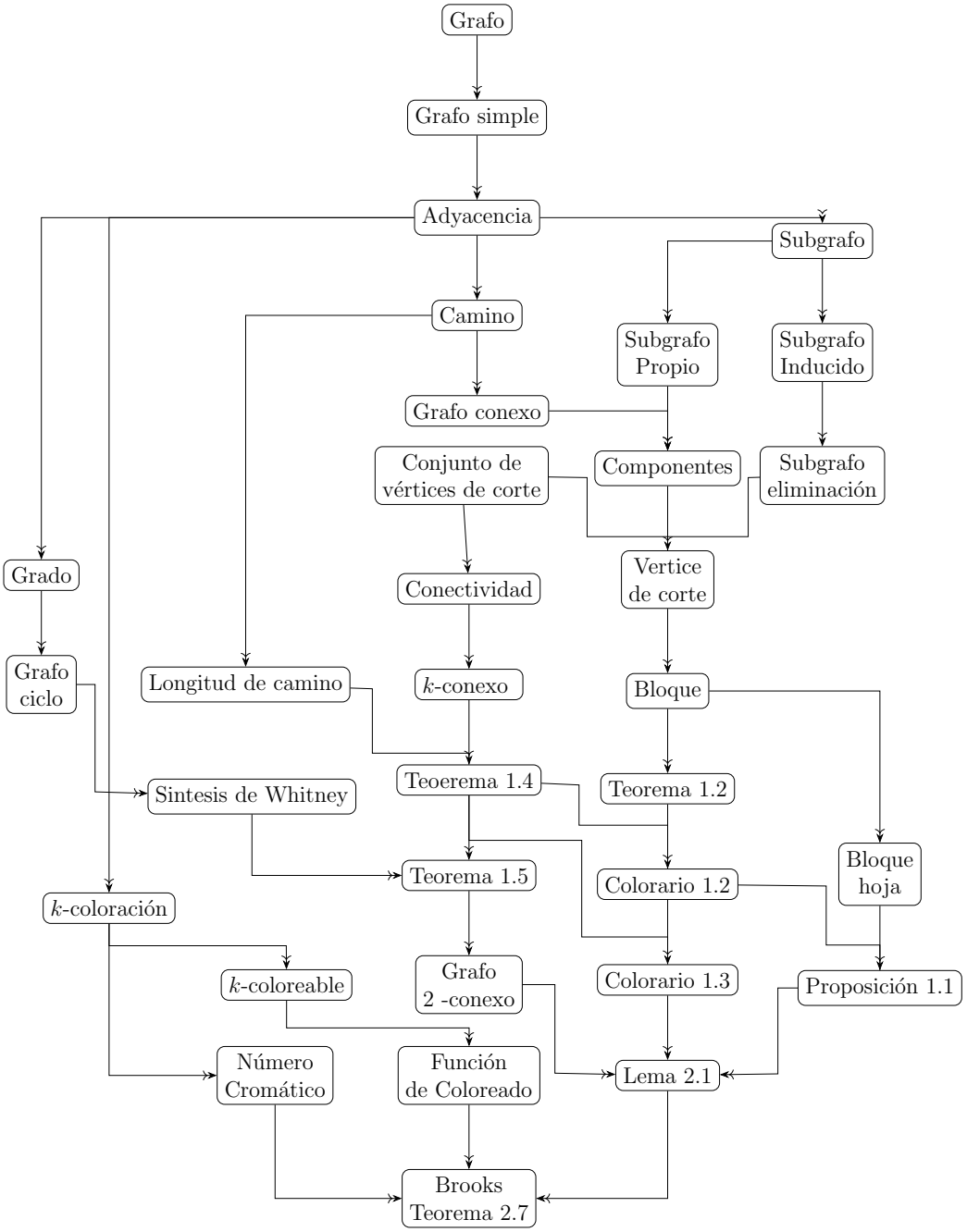


Figura 1.1: Recorrido del marco teórico

Iniciamos con la definición de grafo, señalando algunos tipos de grafos,

estableciendo la relación de adyacencia entre vértices:

Definición 1.1. Un **grafo** G está conformado por una tripleta, dos conjuntos finitos V_G, E_G y una función f_G, donde los elementos de V_G se conocen como vértices, referidos con letras mayúsculas; los elementos de E_G se llaman aristas; y la función f_G que le asigna a cada arista una par no ordenado de vértices, $f_G : E_G \to \{\{A, B\} : A, B \in V_G\}$. Dada la arista m tal que $f_G(m) = \{A, B\}$ diremos que A y B son extremos de la arista m.

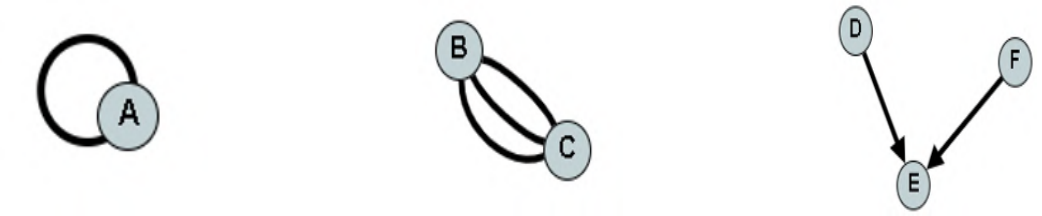
En este trabajo representamos los grafos de forma gráfica, donde los vértices son dibujados como puntos y las aristas como segmentos que conectan aquellos vértices determinados por la función f_G.

A partir de la Definición 1.1 podemos identificar varios tipos de aristas:

1. Dada una arista m y $A = B$, tal que $f_G(m) = \{A\}$, entonces a m la denominamos un bucle, es decir, un vértice conectado consigo mismo; como representamos en la Figura 1.2a.

2. Dadas dos aristas m y n, tal que $f_G(m) = \{A, B\}$ y $f_G(n) = \{A, B\}$, entonces m y n tienen los mismos vértices como extremos, lo que llamamos multiaristas; e ilustramos en la Figura 1.2b.

3. Si $f_G : E_G \to V_G \times V_G$, donde a cada arista le asignamos una pareja ordenada, decimos que las aristas se determinan a partir de un vértice de inicio (primera componente) y otro de fin (segunda componente). Estas aristas se llaman dirigidas y las representamos gráficamente como se muestra en la Figura 1.2c.



(a) Bucle (b) Multiaristas (c) Dirigidas

Figura 1.2: Tipos de aristas

La función f_G nos ofrece otra manera de referirnos a las aristas de un grafo G, señalando los vértices extremos asociados a estas y simplificando la notación, como ilustramos en el ejemplo a continuación:

Ejemplo 1.1. Un grafo G definido por los conjuntos $V_G = \{A, B, C\}$, $E_G = \{m, q\}$ y la función $f_G : E_G \to \{\{V_i, V_j\} : V_i, V_j \in V_G\}$, tal que $f_G(m) = \{C, B\}$ y $f_G(q) = \{A, C\}$; lo podemos definir como el grafo G con $V_G = \{A, B, C\}$ y $E_G = \{CB, AC\}$, cuya representación gráfica corresponde a la Figura 1.3.

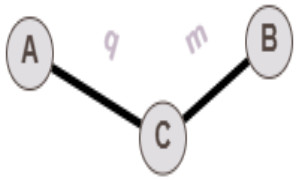
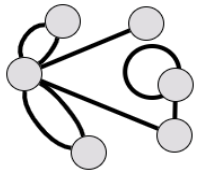
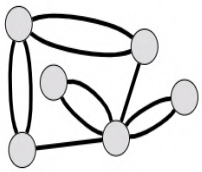


Figura 1.3: Vértices y aristas del grafo G

Los grafos pueden clasificarse en distintos tipos según sus aristas. En la Tabla 1.1 proporcionamos algunos tipos de grafos.

Tipos de grafos	**Tipos de aristas**	**Representación gráfica**
Pseudografo	Bucles y multiaristas	
Multigrafo	Multiarista	

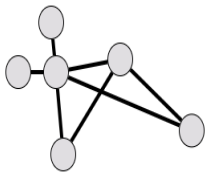
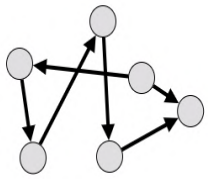
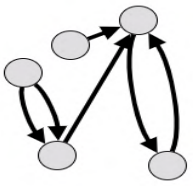
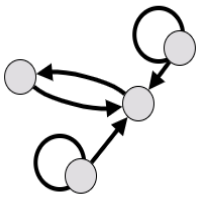
Tipos de grafos	**Tipos de aristas**	**Representación gráfica**
Simple	No tiene bucles ni multiaristas	
Digrafo	Dirigidas	
Multigrafo dirigido	Multiaristas dirigidas	
Pseudografo dirigido	Multiaristas dirigidas y bucles	

Tabla 1.1: Tipos de grafos

En este trabajo solo consideraremos grafos simples, que definimos como:

Definición 1.2. Un **grafo simple** es un grafo que no tiene bucles, multiaristas, ni aristas dirigidas; es decir, dos vértices están conectados por una sola arista.

La relación entre vértices y aristas de un grafo, nos permite definir algunos términos y atributos asociados:

Definición 1.3. Dos vértices A y B en el grafo G son **adyacentes** si están conectados por una arista, es decir, si son los extremos de una arista. Además:

1. Se dice que los vértices A y B son **incidentes** en la arista AB.
2. El conjunto de los vértices adyacentes a un vértice A, lo llamamos la **vecindad de** A y lo notamos como $N(A)$.
3. Dos vértices son **vecinos** si los dos son adyacentes, es decir, son elementos del conjunto de la vecindad de A o de B.

Ejemplo 1.2. En el grafo G con $V_G = \{A, B, C, D, E, F\}$ y $E_G = \{AC, BC, BD, DC, DF, EC, EF, CF\}$ y representación gráfica dada en la Figura 1.4, tenemos que: A es adyacente a C; B es adyacente a C y D. Y de manera general, tenemos las siguientes vecindades:

- $N(F) = \{C, D, E\}$
- $N(B) = \{C, D\}$
- $N(C) = \{A, B, D, E, F\}$
- $N(D) = \{C, F, B\}$
- $N(A) = \{C\}$
- $N(E) = \{C, F\}$

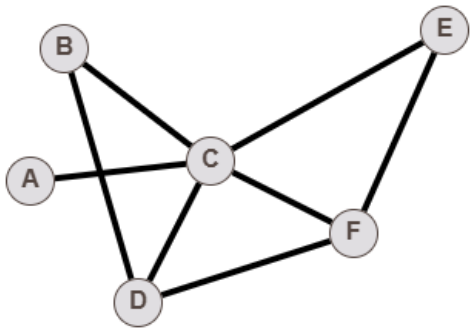


Figura 1.4: Grafo G

1.1. Representaciones de grafos

Existen otras formas alternativas para representar un grafo, ademas de su gráfica, por ejemplo usando matrices y listas de adyacencia.

Definición 1.4. Sea un grafo G con $V_G = \{V_1, V_2, \cdots, V_n\}$ **la matriz de adyacencia** de G, que notamos $A(G) = (a_{ij})$, es una matriz $n \times n$, cuyas filas y columnas están indexadas por el mismo orden de V_G, tal que $a_{ij} = 1$ si los vértices V_i y V_j son adyacentes, y $a_{ij} = 0$ en caso contrario, con $i, j = 1, 2, \cdots, n$.

Ejemplo 1.3. La representación con matriz de adyacencia del grafo G de la Figura 1.4 es:

$$A(G) = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

Teniendo en cuenta el orden de los vértices A, B, C, D, E, F, si dos vértices V_i y V_j son adyacentes entonces en la entrada a_{ij} se corresponde con el número 1, si no son adyacentes entonces se escribe un 0; si cambiamos el orden elegido la matriz de adyacencia cambia.

Definición 1.5. Sea un grafo G, las **listas de adyacencia** son una colección de listas, en las que a cada $V_i \in V_G$ se asocia una lista de los vecinos del vértice correspondiente.

Ejemplo 1.4. Las listas de adyacencia del grafo G, de la Figura 1.4, corresponde a:

$$\begin{array}{lllllll} A: & C & & & & & \\ B: & C & D & & & & \\ C: & A & B & D & E & F & \\ D: & B & C & F & & & \\ E: & C & F & & & & \\ F: & C & D & E & & & \end{array}$$

En seguida, vamos a definir algunos conceptos que surgen a partir de la adyacencia entre dos vértices.

Definición 1.6. Dado el grafo G, el **grado** de un vértice V de G, notado como $gr(V)$, es la cantidad de vecinos de V, o el cardinal del conjunto de la vecindad de V, $|N(V)|$. En un grafo G podemos encontrar:

- **Grado mínimo**, denotado como $\delta(G)$, es el mínimo grado de todos los vértices de G.
- **Grado máximo**, denotado como $\Delta(G)$, el cual es el máximo grado de todos los vértices de G.

Ejemplo 1.5. En el grafo G de la Figura 1.4 podemos observar que:

- $gr(A) = 1$
- $gr(C) = 5$
- $gr(E) = 2$
- $gr(B) = 2$
- $gr(D) = 3$
- $gr(F) = 3$

Entonces, el grafo G tiene $\delta(G) = 1$ y $\Delta(G) = 5$. También puede suceder que el grado mínimo y máximo de un grafo coincidan, como en el grafo W (Figura 1.5), en el que 3 es el grado máximo y mínimo.

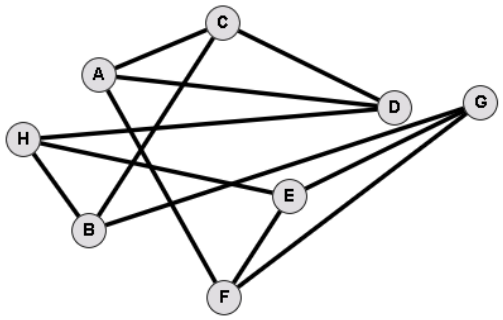


Figura 1.5: Grafo W

Definición 1.7. La **secuencia de grados** de un grafo G es la ordenación, creciente o decreciente, de los grados de los vértices del grafo, que representamos como una lista de los grados $(gr(V_1), gr(V_2), ..., gr(V_n))$, donde $\{V_1, ..., V_n\} = V_G$ y $gr(V_1) \geq gr(V_2) \geq \cdots \geq gr(V_N)$.

Ejemplo 1.6. En el grafo G (Figura 1.4) la secuencia de grados, en orden decreciente, es: $(gr(C), gr(D), gr(F), gr(E), gr(B), gr(A)) = (5, 3, 3, 2, 2, 1)$.

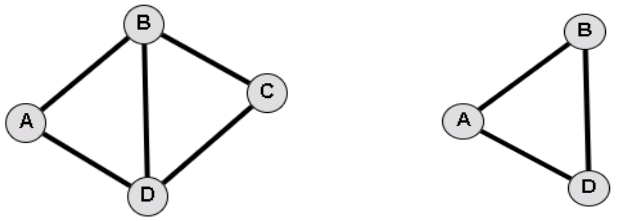
1.2. Subgrafos

El estudio de subgrafos nos permite descomponer un grafo en estructuras más simples, cuyo análisis nos posibilita identificar propiedades a nivel

local, que aportan a la comprensión general del grafo dado. En el caso de la coloración de vértices en un grafo G, los subgrafos nos ayudan a establecer cotas inferiores para el número de colores a usar.

Definición 1.8. Un **subgrafo** H del grafo G, es un grafo tal que sus conjuntos de vértices y aristas son subconjuntos de los conjuntos de vértices y aristas del grafo G, respectivamente.

Ejemplo 1.7. Dado el grafo G con los conjuntos $V_G = \{A, B, C, D\}$ y $E_G = \{AB, BC, CD, BD, DA\}$ (Figura 1.6a). El grafo H con los conjuntos $V_H = \{A, B, D\}$ y $E_H = \{AB, BD, DA\}$ (Figura 1.6b) es un subgrafo de G.



(a) Grafo G (b) Subgrafo H

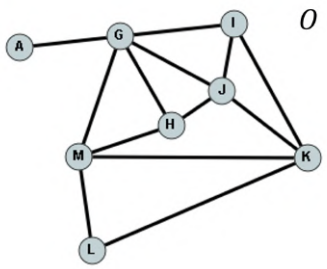
Figura 1.6: Subgrafo

De un grafo podemos obtener varios subgrafos, mediante diferentes estrategias:

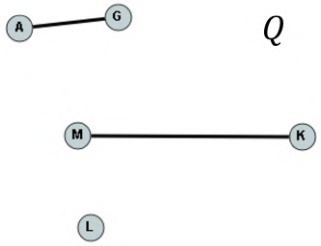
Definición 1.9. Un **subgrafo propio** H del grafo G es aquel cuyo conjunto de vértices es subconjunto propio del conjunto de vértices de G, o cuyo conjunto de aristas es un subconjunto propio del conjunto de aristas de G, esto es, $V_H \subset V_G$ o $E_H \subset E_G$.

Definición 1.10. Dado el grafo G y su conjunto de vértices V_G, un **subgrafo inducido por un subconjunto de vértices** de V_G , es el subgrafo H de G cuyo conjunto de vértices es $V_H \subseteq V_G$, y las aristas de H corresponden a las aristas que están presentes en el grafo G, tal que sus vértices extremos están en el conjunto V_H. Por lo tanto, el subgrafo inducido por V_H mantiene todas las adyacencias de G entre los vértices de V_H, sin añadir ni eliminar aristas existentes entre ellos en G.

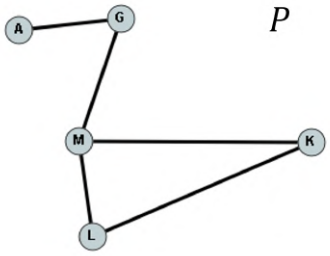
Ejemplo 1.8. A continuación presentamos el grafo O (Figura 1.7a), un subgrafo propio Q cuyo conjunto de vértices es $V_Q = \{A, G, M, L, K\}$ y conjunto de aristas $E_Q = \{AG, MK\}$ (Figura 1.7b); y un subgrafo P, inducido por el conjunto de vértices $V_Q = \{A, G, M, L, K\}$ (Figura 1.7c).



(a) Grafo O



(b) Subgrafo Propio Q

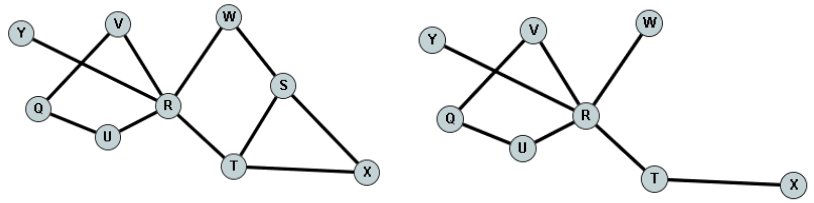


(c) Subgrafo inducido P

Figura 1.7: Algunos subgrafos de O

Definición 1.11. Sea el grafo G y el vértice $V \in V_G$. El **subgrafo de eliminación de un vértice**, que notamos como $G - \{V\}$, es el subgrafo inducido por el conjunto de los vértices del grafo G sin el vértice V, es decir, $V_G - \{V\}$.

Ejemplo 1.9. Sea el grafo M, que ilustramos en la Figura 1.8a, el subgrafo de eliminación del vértice $\{S\}$, es el grafo inducido por los vértices $\{Y, V, Q, U, R, W, T, X\}$. El subgrafo $M - \{S\}$ está representado en la Figura 1.8b.



(a) Grafo M (b) Subgrafo $M - \{S\}$

Figura 1.8: Subgrafo de eliminación de un vértice

Definición 1.12. Sea un grafo G y la arista $e \in E_G$. El **subgrafo de eliminación de una arista**, que notamos $G - \{e\}$, es aquel cuyo conjunto de vértices es igual al de G y su conjunto de aristas es $E_G - \{e\}$.

Ejemplo 1.10. En el grafo M (Figura 1.8a) si eliminamos la arista ST, obtenemos el subgrafo $M - \{ST\}$ como se ilustra en la Figura 1.9.

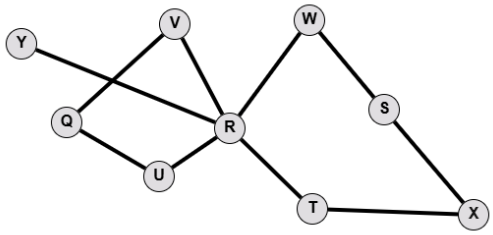


Figura 1.9: Subgrafo $M - \{ST\}$

1.3. Algunas operaciones entre grafos

Otra forma de estudiar los grafos es verlos como resultado de operaciones entre otros grafos dados. Este enfoque nos es de utilidad en problemas como la coloración de vértices de grafos, porque permite transformar un grafo con estructura compleja en grafos conocidos, lo que ayuda a comprender la estructura del grafo y mejorar las coloraciones para el grafo dado, es decir, reducir la cantidad de colores para colorear los vértices. En esta sección nos enfocamos en algunas operaciones con grafos.

Definición 1.13. Dados los grafos G y F, el **grafo unión** de los dos grafos, denotado $G \cup F$, es aquel cuyo conjunto de vértices y aristas son uniones disyuntas de los conjuntos de vértices y aristas de los grafos G y F, respectivamente.

Ejemplo 1.11. En la Figura 1.10 exhibimos el grafo unión entre los grafos O y P. Estos grafos no tienen vértices en común ni poseen aristas que los conecten entre sí. Además, cada uno, O y P, se considera un subgrafo del grafo $O \cup P$.

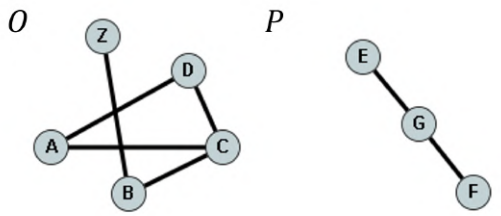


Figura 1.10: Grafo $O \cup P$

Definición 1.14. El **grafo suma** de los grafos disyuntos G y H, notado como $G + H$, se obtiene a partir del grafo $G \cup H$, añadiendo una arista entre cada vértice de G y cada vértice de H.

Ejemplo 1.12. En la Figura 1.11 mostramos el grafo suma entre los grafos O y P, que presentamos en la Figura 1.10.

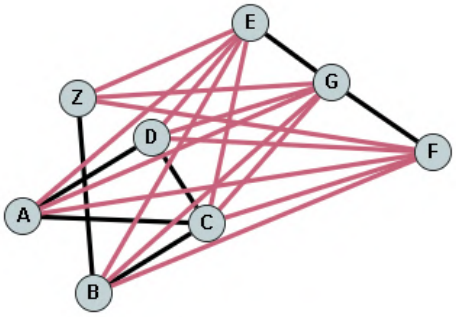


Figura 1.11: Grafo $O + P$

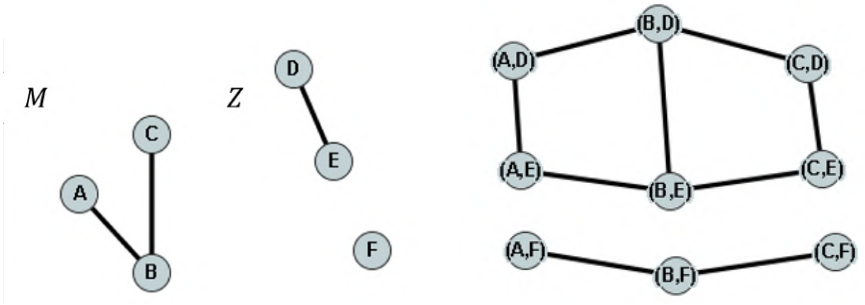
Definición 1.15. Dados dos grafos G y H, el **producto cartesiano** de los dos grafos, que notamos $G \times H$, tiene como conjunto de vértices al producto cartesiano entre los vértices de G y H, $V_{G\times H} = V_G \times V_H$. Además, la adyacencia de dos vértices (A, C) y (B, D), de $G \times H$ con $A, B \in V_G$ y $C, D \in V_H$, se determina si y solo si

1. $A = B$ y C, D son adyacentes en H

2. $C = D$ y A, B son adyacentes en G

Ejemplo 1.13. Dados los grafos M y Z (Figura 1.12a), el grafo $M \times Z$ resultante del producto cartesiano entre ellos, corresponde al grafo representado en la Figura 1.12b.

Como observamos en el grafo $M \times Z$, los vértices (A, E) y (B, E) son adyacentes porque tienen en común a E, y en el grafo M los vértices A y B están conectados. En cambio, los vértices (A, E) y (A, F) no son adyacentes en $M \times Z$, porque en el grafo Z no existe una arista que conecte a los vértices E y F.



(a) Grafos M y Z (b) Grafo $M \times Z$

Figura 1.12: Producto cartesiano

1.4. Algunas familias de grafos

A continuación estudiaremos algunas familias de grafos que también nos va a permitir simplificar en algunas ocasiones el problema de coloración de vértices, pues para estas familias podemos establecer de manera teórica su número cromático.

Definición 1.16. El grafo **trivial** consiste de un vértice sin aristas.

Definición 1.17. Un grafo $k-$**regular** es aquel en el que todos sus vértices tienen k vecinos, es decir, todos los vértices tienen el mismo grado.

Ejemplo 1.14. Uno de los grafos $k-$regulares más reconocidos por sus propiedades y usos para probar conjeturas y teoremas es el grafo **Petersen**, el cual es 3− regular.

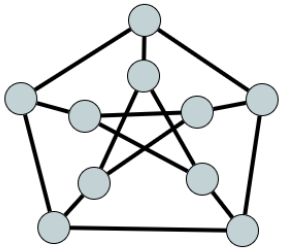


Figura 1.13: Grafo Petersen

Definición 1.18. Un grafo **completo** con n vértices, notado K_n, es aquel en que cada par de vértices está conectado entre sí.

Ejemplo 1.15. Un grafo completo, en este caso, con 6 vértices lo ilustramos en la Figura 1.14.

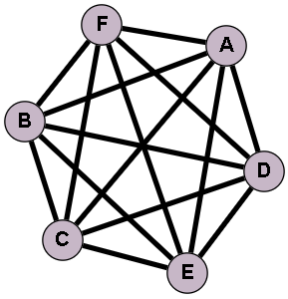


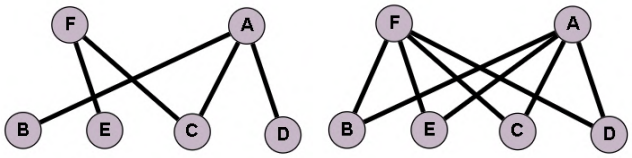
Figura 1.14: Grafo K_6

Definición 1.19. Un grafo **bipartito**, denotado como B_{mn}, es aquel cuyo conjunto de vértices $V_{B_{mn}}$ se puede dividir en dos subconjuntos U y W con m y n elementos, respectivamente, tales que los vértices de U no son adyacentes entre sí, así mismo para el subconjunto W.

Definición 1.20. Un grafo **bipartito completo**, con notación K_{mn}, es un grafo bipartito que cumple la condición que todos los vértices de U son adyacentes a todos los vértices de W.

Ejemplo 1.16. En la Figura 1.15 presentamos dos grafos bipartitos cuyos conjuntos de vértices se pueden dividir en subconjuntos $U = \{F, A\}$ y $W = \{B, C, D, E\}$, tales que no hay aristas entre los vértices de U ni entre los vértices de W, pero sí existen aristas que conecten a los vértices de U con los de W.

La diferencia entre ambos bipartitos radica en que el grafo bipartito completo $K_{2,4}$ (Figura 1.15b), todos los vértices de U están conectados con todos los vértices de W. En cambio, en el bipartito $B_{2,4}$ no sucede aquello (Figura 1.15a).



(a) Grafo $B_{2,4}$ (b) Grafo $K_{2,4}$

Figura 1.15: Grafos bipartitos

Definición 1.21. El grafo **lineal**, denotado como P_n, es un grafo donde cada vértice V_i está conectado únicamente con sus vecinos inmediatos V_{i+1} y V_{i-1}, siempre que existan; de manera que tiene dos vértices de grado 1 y los demás (si aplica) son de grado 2.

Ejemplo 1.17. En la Figura 1.16 ilustramos un grafo lineal G con 6 vértices.

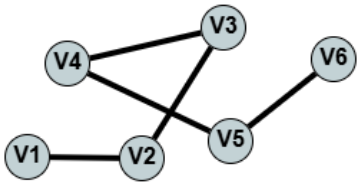


Figura 1.16: Grafo Lineal

Definición 1.22. Un grafo **ciclo**, notado C_n, es un grafo conexo [1] tal que el grado de todos sus vértices es dos.

Ejemplo 1.18. En la Figura 1.17 ilustramos un grafo conexo con 6 vértices, donde todos tienen grado 2, luego es un grafo ciclo.

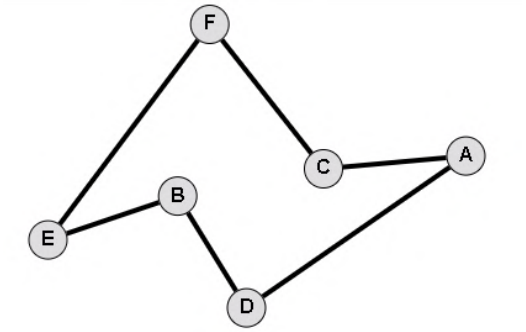


Figura 1.17: Grafo C_6

Definición 1.23. El grafo **rueda** con n vértices, denotado W_n, es el grafo ciclo C_{n-1} junto con un nuevo vértice x, el cual es adyacente a todos los vértices del ciclo.

Ejemplo 1.19. La Figura 1.18 representa el grafo rueda con siete vértices.

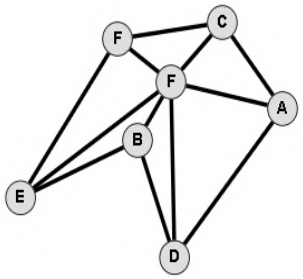


Figura 1.18: Grafo W_7

Definición 1.24. El grafo **árbol**, con n vértices, que notamos T_n, es un grafo conexo, en el que cualquier par de vértices están conectados por exactamente un camino y no tiene ciclos.

[1] En la sección 1.5 abordamos la Definición 1.30 de grafo conexo.

Ejemplo 1.20. La Figura 1.19 representa un grafo árbol con ocho vértices.

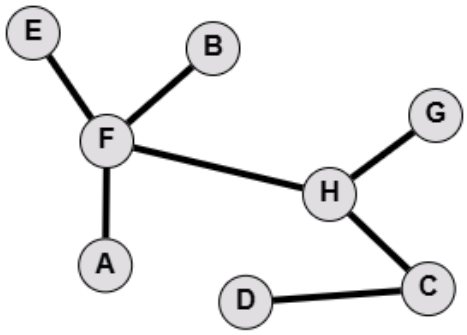


Figura 1.19: Grafo T_8

Definición 1.25. El grafo **hipercubo** es un grafo que notamos Q_n tiene 2^n vértices y se obtiene del producto cartesiano de n copias del grafo completo K_2. Además, Q_n es un grafo $n-$regular.

Ejemplo 1.21. La Figura 1.20 representa un grafo hipercubo denotado Q_3 que es $3-$regular.

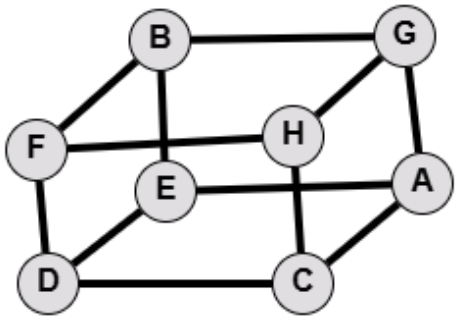


Figura 1.20: Grafo Q_3

1.5. Conectividad

La conectividad es un concepto importante en la teoría de grafos, ya que nos permite entender cómo están relacionados los vértices y qué tan conectado está un grafo; lo que nos ayuda a identificar si se puede dividir en partes independientes o si existen puntos críticos, como vértices de corte, que al eliminarse romperían la estructura y nos dejarían componentes aisladas.

El saber cómo están conectados los vértices nos ofrece ventajas en el problema de la coloración de vértices, pues podemos simplificar el problema general dividiendo el grafo en componentes más pequeñas y fáciles de colorear por separado. Además, la forma en que los vértices se agrupan y se relacionan entre sí nos da pistas claves sobre cuántos colores necesitaremos y cómo diseñar algoritmos más eficientes para asignar dichos colores.

Definición 1.26. Un **camino** h de V_1 a V_n es una secuencia finita de vértices y aristas de G, comenzando en el vértice V_1 y terminando en el vértice V_n, donde cada arista es incidente con el vértice que la precede y el que la sigue, esto es, $h = (V_1, V_1V_2, V_2, V_2V_3, ..., V_{n-1}, V_{n-1}V_n, V_n)$.

Un camino se puede escribir de varias maneras: enunciando únicamente las aristas $h = (V_1V_2, V_2V_3, ..., V_{n-1}V_n)$, o nombrando los vértices $h = (V_1, V_2, V_3, ..., V_{n-1}, V_n)$.

Definición 1.27. La **longitud de un camino** h es la cantidad de aristas pertenecientes al camino, se nota como l_h. El camino de longitud cero es el que no tiene aristas pero sí un vértice.

Definición 1.28. Dos vértices V_1 y V_2 de un grafo G se dice que están conectados si existe un camino entre ambos, es decir, cuando V_1 y V_2 son extremos de un camino.

De los caminos surgen varios tipos según sus características:

Definición 1.29. Algunas clases de caminos son:

1. Un **camino simple** es donde los vértices que aparecen en la secuencia no se repiten, esto es, todos son diferentes.

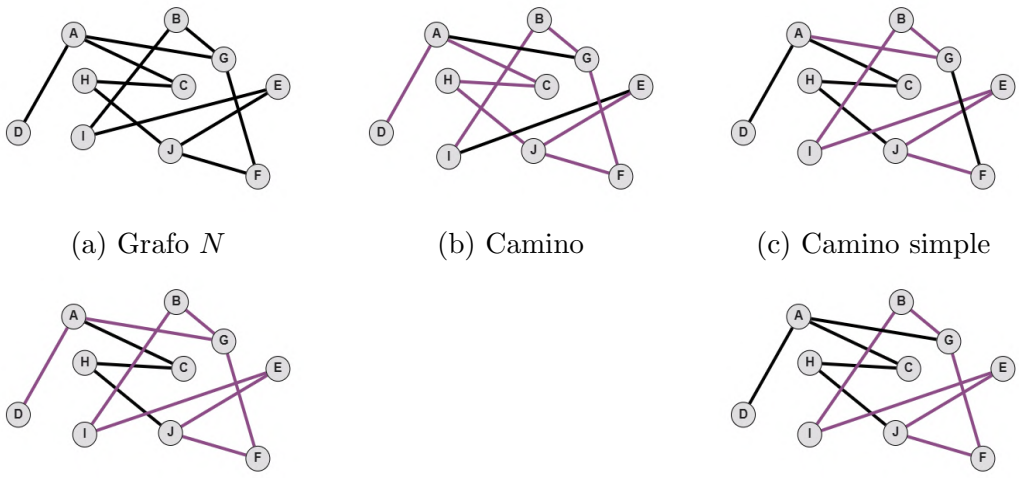
2. Un **camino cerrado** es aquel cuyos vértices de inicio y fin son el mismo.

3. Un **recorrido** es un camino sin aristas repetidas.

4. Un **camino ciclo**, es un camino cerrado donde los únicos vértices que coinciden son los de inicio y fin.

Ejemplo 1.22. En la Figura 1.21a presentamos el grafo N en el que se pueden determinar caminos simples, cerrados y ciclos, como:

- Un camino del vértice I al vértice A es $h = (IB, BG, GF, FJ, JE, EJ, JH, HC, CA, AD, DA)$, como está resaltado en morado en la Figura 1.21b.
- Un camino simple del vértice A al vértice F es $h_s = (AG, GB, BI, IE, EJ, JF)$, como se evidencia con las aristas de color morado en el grafo de la Figura 1.21c.
- Un camino cerrado del vértice D a si mismo, es $h_{ce} = (DA, AG, GB, BI, IE, EJ, JF, FG, GA, AD)$, como se visualiza resaltado en morado en la Figura 1.21d.
- Un camino ciclo del vértice F a si mismo, es $h_{ci} = (FJ, JE, EI, IB, BG, GF)$, como se resalta en morado en la Figura 1.21e.



(a) Grafo N (b) Camino (c) Camino simple

(d) Camino cerrado (e) Camino ciclo

Figura 1.21: Caminos de N

Teorema 1.1. Cualquier camino cerrado de longitud impar contiene un ciclo impar.

Demostración. La demostración la elaboramos por inducción sobre la longitud del camino, denotado por l.

1. Dado que en este trabajo utilizamos grafos simples, si tomamos un camino de longitud $l = 1$ generaría un bucle. Por lo tanto, la inducción la iniciamos desde $l = 3$. Sea h un camino cerrado de longitud 3, dicho camino debe involucrar tres vértices diferentes, pues de lo contrario tendríamos que considerar un bucle como una arista en la secuencia, pero estamos trabajando con grafos simples. Luego, este camino cerrado solo tiene los vértices extremos repetidos, esto es, un ciclo impar (Nota 1).

2. Sea h un camino cerrado de longitud impar l mayor que 3; supongamos que la afirmación se cumple para caminos cerrados de longitud impar menor que l, es decir, que todo camino cerrado de longitud impar menor que l contiene un ciclo impar.

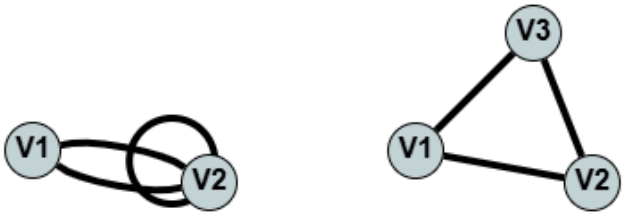
3. Consideremos el camino cerrado $h = (V_1V_2, V_2V_3..., V_{l-1}V_l, V_lV_1)$ que tiene longitud impar. Existen dos casos con aquel camino:

   a) Si todos los vértices de h son distintos, entonces h es un ciclo de longitud impar (Nota 2 ítem 1).

   b) Si existe un vértice V_i que se repite en en camino h, este camino puede dividirse en dos subcaminos que inician y terminan en V_i. Uno de los subcaminos necesariamente debe ser de longitud par, mientras que el otro es de longitud impar, pues h es de longitud impar. Por la hipótesis de inducción, como el subcamino de longitud impar es más corto que h, este contiene un ciclo de longitud impar (Nota 2 ítem 2).

□

Nota 1. Para construir un camino cerrado de longitud tres, por ejemplo de V_1 a V_1, debemos tener tres vértices diferentes, porque de lo contrario tendríamos que $h = (V_1, V_1V_2, V_2, V_2V_2, V_2, V_2V_1, V_1)$ y el grafo no es simple (Figura 1.22a):



(a) Camino cerrado con dos vértices

(b) Camino cerrado con tres vértices

Figura 1.22: Caminos cerrados de longitud 3

Nota 2. 1. En este caso, por ejemplo, tenemos el camino cerrado $h = (V_1, V_1V_2, V_2, V_2V_3, V_3, V_3V_4, V_4, V_4V_5, V_5, V_5V_1, V_1)$ cuya longitud es 5 y es un ciclo impar (Figura 1.23).

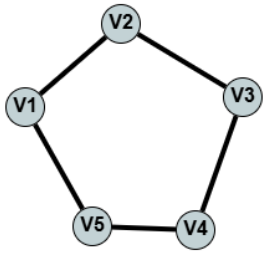
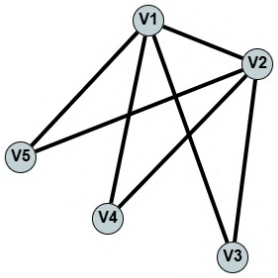
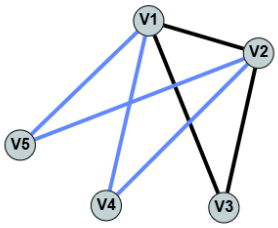


Figura 1.23: Camino cerrado y ciclo impar

2. En este caso, tenemos el camino $h = (V_1, V_1V_2, V_2, V_2V_3, V_3, V_3V_1, V_1, V_1V_4, V_4, V_4V_2, V_2, V_2V_5, V_5, V_5V_1, V_1)$ de longitud 7 (Figura 1.24a), el cual tiene los subcaminos cerrados $h_1 = (V_1, V_1V_2, V_2, V_2V_3, V_3, V_3V_1, V_1)$ y $h_2 = (V_1, V_1V_4, V_4, V_4V_2, V_2, V_2V_5, V_5, V_5V_1, V_1)$ de longitud 3 y 4, respectivamente, como se muestra en la Figura 1.24b.



(a) Camino h



(b) Caminos de longitud 3 y 4

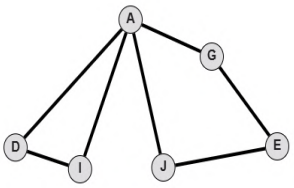
Figura 1.24: Camino cerrado y ciclo impar

Algunos conceptos que surgen a partir de la noción de camino son:

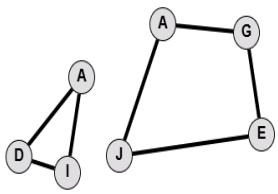
Definición 1.30. Un grafo **conexo** es aquel en el que cada par de vértices distintos están conectados por un camino.

Definición 1.31. Una **componente conexa** de un grafo G es un subgrafo de G conexo, que no es un subgrafo propio de otro subgrafo conexo de G, es decir, es un subgrafo conexo maximal de G.

Ejemplo 1.23. El grafo O representado en la Figura 1.25a es conexo , y el grafo de la Figura 1.25b no es conexo, pero tiene dos componentes conexas.



(a) Grafo conexo O



(b) Grafo M

Figura 1.25: Grafo conexo y componentes conexas

Definición 1.32. Un **clique** es un subconjunto de vértices de G donde cada par de vértices son adyacentes entre sí, además es un conjunto maximal de vértices de G mutuamente adyacentes.

Definición 1.33. El **número de clique** de un grafo G es el cardinal del clique más grande de G, y lo denotamos con $\omega(G)$.

Definición 1.34. Un subconjunto del conjunto de vértices del grafo G es un **conjunto independiente** si ningún par de vértices del subconjunto están conectados por una arista.

Definición 1.35. El **número de independencia** del grafo G es el cardinal del conjunto independiente más grande de G, y lo notamos $\alpha(G)$.

Ejemplo 1.24. En el grafo F (Figura 1.26), algunos subconjuntos cliques, donde $\omega(F) = 5$:

- $\{O, N, M, L, K\}$
- $\{P, O\}$
- $\{N, R\}$
- $\{P, S\}$
- $\{T, S\}$
- $\{K, S\}$
- $\{P, Q\}$
- $\{T, R\}$
- $\{Q, R\}$

Además, número de independencia es $\alpha(F) = 3$ y algunos conjuntos independientes son:

- $\{P, K, R\}$
- $\{P, L, R\}$
- $\{S, N, Q\}$
- $\{P, K, T\}$
- $\{P, L, T\}$
- $\{P, N, T\}$
- $\{S, L, R\}$
- $\{S, O, Q\}$
- $\{P, M, R\}$
- $\{S, Q, M\}$
- $\{P, M, T\}$
- $\{S, O, R\}$
- $\{L, S, Q\}$

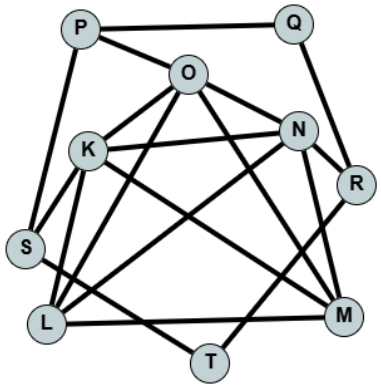
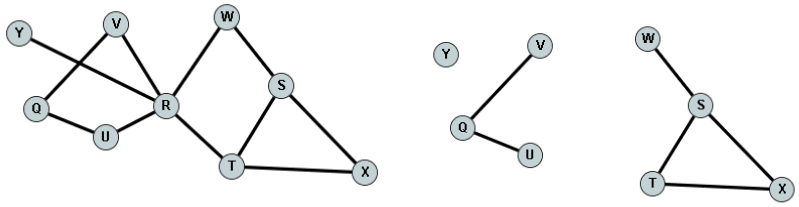


Figura 1.26: Grafo F

Con las operaciones entre grafos y algunos conceptos sobre conexidad, introducimos otros que permiten analizar los grafos.

Definición 1.36. Un **vértice de corte** del grafo G, es un vértice V, tal que al eliminarlo, el grafo $G - \{V\}$ tiene más componentes conexas que el grafo G.

Ejemplo 1.25. En el grafo M (Figura 1.27a), si eliminamos el vértice R, el grafo $M - \{R\}$ (Figura 1.27b) tiene tres componentes conexas, mientras que el grafo M solo tiene una. Esto es, R es un vértice de corte.

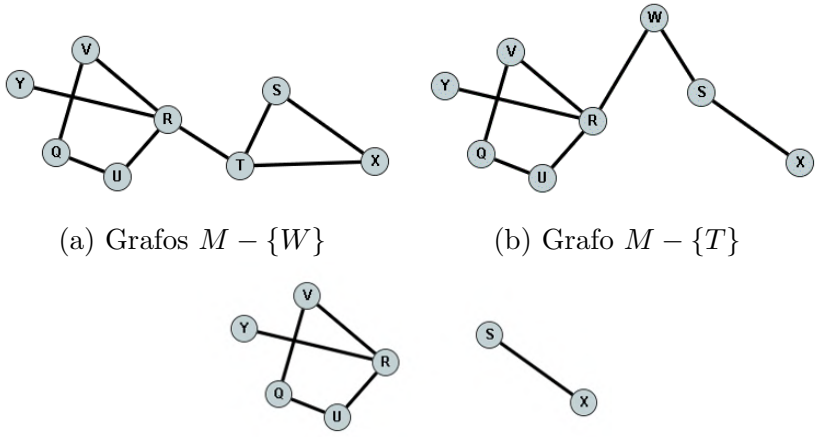


(a) Grafo M (b) Grafo $M - \{R\}$

Figura 1.27: Vértice de corte

Definición 1.37. Un **conjunto de vértices de corte** de un grafo G, es un conjunto de vértices de G, tal que al eliminarlos a la vez, el grafo obtenido presenta más componentes conexas que G .

Ejemplo 1.26. En el grafo M (Figura 1.27a) si eliminamos el vértice W, el grafo $M - \{W\}$ (Figura 1.28a) tiene una componente conexa al igual que M, lo mismo sucede con el grafo $M - \{T\}$ cuando eliminamos a T (Figura 1.28b). Pero si eliminamos los dos vértices al mismo tiempo, el grafo $M - \{W, T\}$ tiene dos componentes conexas (Figura 1.28c). Por lo tanto el conjunto de vértices $\{W, T\}$ es de corte.



(a) Grafos $M - \{W\}$ (b) Grafo $M - \{T\}$

(c) Grafo $M - \{W, T\}$

Figura 1.28: Conjunto de corte $\{W, T\}$

Definición 1.38. La **conectividad de vértices** de un grafo conexo G, notado como $\kappa(G)$ es la cantidad mínima de vértices que al removerlos, desconectan G o lo reducen a un vértice .

Definición 1.39. Un **bloque** de un grafo G, es un subgrafo conexo maximal tal que ninguno de sus vértices es un vértice de corte.

Ejemplo 1.27. Un bloque B del grafo G (Figura 1.29) es el subgrafo inducido por el conjunto de vértices $\{M, A, G, E, J, I\}$, el cual no tiene vértices de corte y su número de conectividad ($\kappa(B)$) es 2.

Otro bloque del grafo G es el subgrafo inducido por el conjunto de vértices $\{D, M\}$, que también es una arista. Y otro bloque es el vértice L.

Estos ejemplos de bloque muestran que aunque los vértices que los componen no son vértices de corte para los bloques, sí hay vértices de corte en el grafo G, como lo son M y E.

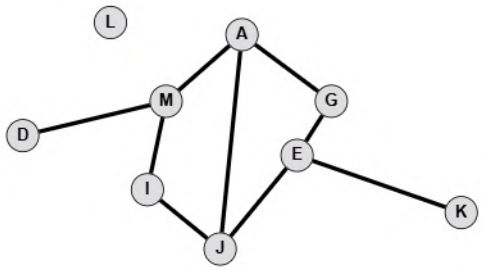


Figura 1.29: Grafo G con bloques

Los siguientes resultados nos permiten relacionar los bloques asociados a un grafo con sus puntos de corte:

Teorema 1.2. Sea G un grafo con dos bloques diferentes B_1 y B_2, los bloques pueden tener como máximo un vértice en común.

Demostración. Sean B_1 y B_2 dos bloques diferentes de un grafo G suponemos que ambos bloques tienen dos vértices en común X y Y, y consideramos el grafo $B_1 \cup B_2$.

Si eliminamos al vértice X del bloque B_1, el subgrafo $B_1 - \{X\}$ es conexo, por tanto, de B_1, existe un camino de cualquiera de los vértices de $B_1 - \{X\}$ al vértice Y. Así mismo sucede con el subgrafo $B_2 - \{X\}$ en el que contamos con un camino desde cualquiera de los vértices de $B_2 - \{X\}$ al vértice Y. Luego, en el grafo $B_1 \cup B_2 - \{X\}$ existe un camino entre cualquiera de los vértices del bloque $B_1 - \{X\}$ a cualquiera de los vértices del bloque $B_2 - \{X\}$ que pasa por el vértice Y, por lo tanto el vértice X no es vértice de corte para el subgrafo $B_1 \cup B_2$. Análogamente cuando eliminamos el vértice Y en los bloques B_1 y B_2, obtenemos que Y no es un vértice de corte para el subgrafo $B_1 \cup B_2$.

Ademas los vértices que solo están en uno de los bloques B_1 o B_2, no son vértices de corte para $B_1 \cup B_2$, pues si lo fueran tendrían que serlo para B_1 o B_2, pero esto contradice la Definición 1.39.

Como en el subgrafo $B_1 \cup B_2$ ninguno de sus vértices es de corte, este es un bloque que además contiene a los bloque B_1 y B_2, lo que contradice que B_1 y B_2 sean subgrafos conexos maximales. Entonces necesariamente los bloques deben tener como máximo un único vértice en común. □

Nota 3. Dados dos bloques B_1 con $V_{B_1} = \{A, B, C, D, X\}$ y B_2 con $V_{B_2} = \{X, Y, F, E, G\}$, si suponemos que $Y \in B_1$ debemos establecer una conexión entre los vértices C y Y (Figura 1.30).

De ahí que si eliminamos a X de los dos bloques sigue existiendo un camino entre los vértices $B_1 - \{X\}$ y $B_2 - \{X\}$ pasando por Y. Análogamente sucede si eliminamos al vértice Y, por lo tanto $B_1 \cup B_2$ sería un bloque, situación que ejemplifica la contradicción en la demostración del teorema.

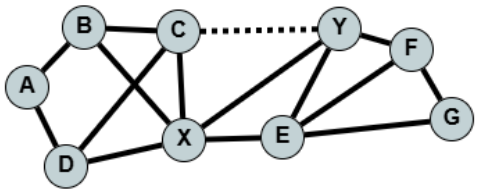


Figura 1.30: Bloques con dos vértices en común

Corolario 1.1. Toda arista de un grafo pertenece a un único bloque.

Demostración. Sea $e = UV$ una arista arbitraria de G. Sea $F = \{H \subseteq G : H$ es un subgrafo conexo, sin vértices de corte y que contiene a la arista $e\}$, F es diferente de vacío, pues la arista e es un elemento de F.

Luego consideramos un elemento máximal de F que llamamos B, este subgrafo es un bloque que contiene a e. Ahora vamos a probar que existe un solo bloque que contiene a e, para ello supongamos que existen dos bloques distintos B_1 y B_2 que contienen a e. Luego, los extremos de e, los vértices U y V, pertenecen tanto a B_1 como a B_2, pero esto contradice el Teorema 1.2, por lo tanto $B_1 = B_2$. □

Corolario 1.2. Sea X un vértice del grafo G. El vértice X es un vértice de corte de G si y solo si X pertenece a dos bloques diferentes B_1 y B_2 de G.

Demostración. Primero vamos a demostrar que si X es un vértice de corte de G entonces pertenece a dos bloques diferentes B_1 y B_2 de G. Supongamos que X es un vértice de corte de G, entonces $G - \{X\}$ tiene al menos dos componentes conexas distintas, que llamamos C_1 y C_2. Luego, existe un vértice V_1 en C_1 tal que XV_1 es una arista de G, y existe un vértice V_2 en C_2 tal que XV_2 es una arista de G; por el Corolario 1.1, la arista XV_1 pertenece a un bloque B_1 y la arista XV_2 pertenece a un bloque B_2, y en particular, X está en B_1 y en B_2.

Veamos que B_1 es diferente de B_2: Supongamos, que $B_1 = B_2$, entonces como los bloques son subgrafos sin vértices de corte y el vértice X no separa

a B_1, en $B_1 - \{X\}$ debe existir un camino que conecte a V_1 con V_2. Pero ese camino estaría contenido en $G - \{X\}$ lo que contradice que V_1 y V_2 están en componentes distintas C_1 y C_2 de $G - \{X\}$, luego los bloques B_1 y B_2 son diferentes.

Para la segunda parte, vamos a demostrar que si dos bloques diferentes de G, B_1 y B_2 tienen un vértice en común X, entonces ese vértice X es de corte para el grafo G.

Supongamos que X pertenece a dos bloques distintos B_1 y B_2, por el Teorema 1.2, tenemos que $B_1 \cap B_2 = \{X\}$. Luego, no existe un camino en G que conecte un vértice de $B_1 - \{X\}$ con uno de $B_2 - \{X\}$ sin pasar por X. Por tanto, al eliminar X de G, los subgrafos $B_1 - \{X\}$ y $B_2 - \{X\}$ quedan en componentes conexas distintas de $G - \{X\}$. Esto implica que estas componentes que aparecen en $G - \{X\}$ no estaban separadas antes, es decir, el número de componentes de $G - \{X\}$ es mayor que las de G, luego X es un vértice de corte de G. □

Corolario 1.3. Sea G un grafo conexo con dos bloques diferentes, B_1 y B_2, y sean los vértices V_1 y V_2 pertenecientes a B_1 y B_2, respectivamente, tal que ninguno es un vértice de corte en G. Entonces V_1 no es adyacente a V_2.

Demostración. Vamos a demostrar el corolario por contradicción:

Dado un grafo conexo G, con dos bloques diferentes B_1 y B_2 y los vértices $V_1 \in B_1$ y $V_2 \in B_2$. Suponemos que existe una arista que conecta a V_1 y V_2.

Con esta arista tenemos un subgrafo $B_1 \cup B_2 \cup \{V_1V_2\}$ que es conexo, porque los bloques en si son conexos y con la arista agregada sigue siendo conexo. De allí suceden tres casos:

1. Si existe un vértice $W \in B_1$ que también pertenece a $B_1 \cup B_2 \cup \{V_1V_2\}$, tal que ese vértice es de corte para el subgrafo $B_1 \cup B_2 \cup \{V_1V_2\}$ llegamos a una contradicción porque ninguno de los vértices del bloque B_1 puede ser vértice de corte para ese bloque por lo tanto, tampoco lo es para el subgrafo $B_1 \cup B_2 \cup \{V_1V_2\}$. Análogamente concluimos que ninguno de los vértices de B_2, puede ser un vértice de corte para $B_1 \cup B_2 \cup \{V_1V_2\}$.

2. Suponemos que $V_1 \in B_1 \cap B_2$, por el Corolario 1.2, si esto sucede el vértice V_1 es un vértice de corte del grafo G, lo cual contradice la hipótesis del enunciado que V_1 no puede ser vértice de corte para el grafo G. Así mismo sucede con el vértice V_2, entonces $V_2 \notin B_1 \cap B_2$.

3. Si existe un vértice en la intersección de los bloques sea $W \in B_1 \cap B_2$, ese vértice no es de corte para $B_1 \cup B_2 \cup \{V_1V_2\}$; porque si lo eliminamos, el subgrafo $B_1 \cup B_2 \cup \{V_1V_2\}$ sigue siendo conexo, puesto que la arista V_1V_2 conecta ambos bloques. Siendo así, el subgrafo $B_1 \cup B_2 \cup \{V_1V_2\}$ es un bloque contradiciendo la maximalidad de los bloque iniciales B_1 y B_2.

Es así como demostramos que no puede haber una arista que conecte dos vértices que pertenecen a dos bloques diferentes. □

Para ilustrar la demostración del Corolario 1.3, proponemos el siguiente ejemplo.

Ejemplo 1.28. En la Figura 1.32 mostramos el grafo G que tiene dos bloques, los subgrafos inducidos por los conjuntos de vértices $B_1 = \{A, B, D, C\}$ y $B_2 = \{I, E, H, F, C\}$, de allí se escogen los vértices $A \in B_1$ e $I \in B_2$. Si asumimos que estos vértices son adyacentes, no existe un vértice de corte en el grafo $G \cup \{AI\}$, por lo tanto el grafo $G \cup \{AI\}$ es un bloque, lo cual contradice la maximalidad de B_1 y B_2.

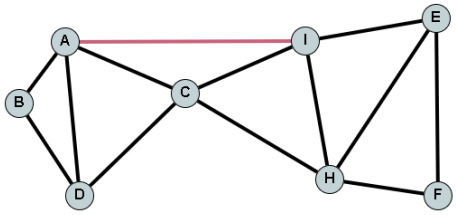
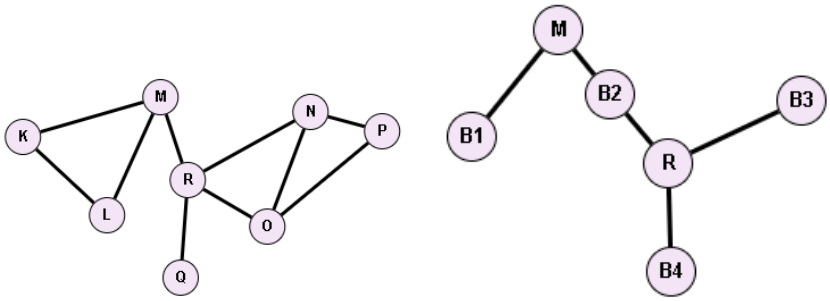


Figura 1.31: Grafo G con la arista AI

Definición 1.40. Un **grafo bloque** de un grafo G, notado como $BL(G)$, es aquel cuyo conjunto de vértices corresponde a bloques de G y vértices de corte de G, donde las aristas tienen como extremos un vértice bloque y un vértice de corte de G que pertenece a dicho bloque.

Ejemplo 1.29. El grafo N de la Figura 1.32a tiene dos vértices de corte M y R, con cuatro bloques $B_1 = \{L, K, M\}$, $B_2 = \{M, R\}$, $B_3 = \{R, N, P, O\}$ y $B_4 = \{Q, R\}$. En grafo bloque $BL(N)$ cada vértice representa un bloque de N y un vértice de corte de N, además si un vértice es un bloque, otro vértice de corte en N y el vértice pertenece al bloque entonces trazamos una arista entre ellos, como representamos en la Figura 1.32b.



(a) Grafo N (b) Grafo $BL(N)$

Figura 1.32: Grafo bloque del grafo N

Definición 1.41. Un **bloque hoja**, es un bloque que solo tiene un vértice de corte en el grafo G. Es decir, el vértice es de corte para G pero no lo es para el bloque.

Ejemplo 1.30. En el grafo G (Figura 1.33), el subgrafo inducido por los vértices $\{A, B, D, K\}$ es un bloque hoja, puesto que el vértice K es un vértice de corte en el grafo, pero no lo es en el subgrafo inducido.

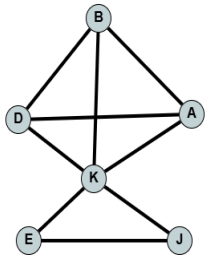


Figura 1.33: Ejemplo de Bloque y Bloque hoja

Ahora, una proposición que relaciona los vértices de corte con la existencia de bloques hoja.

Proposición 1.1. Sea G un grafo conexo, con al menos un vértice de corte. Entonces G tiene al menos dos bloques hoja.

Demostración. Sea G un grafo conexo con al menos un vértice de corte, consideremos su grafo bloque $BL(G)$; afirmamos que $BL(G)$ es un grafo árbol, porque es conexo y acíclico.

$BL(G)$ es conexo, pues si no lo fuera, existirían al menos dos componentes conexas en $BL(G)$, lo que implica que en G hay bloques que no están conectados entre sí, es decir, más de una componente conexa, contradiciendo que G es conexo; por lo tanto $BL(G)$ es conexo.

$BL(G)$ es acíclico; supongamos por contradicción que $BL(G)$ contiene un ciclo:

1. Si existiera un ciclo impar en $BL(G)$, necesariamente habrían dos vértices del mismo tipo adyacentes, esto es, dos bloques o dos vértices de corte adyacentes; sin embargo, por la Definición de grafo bloque 1.40, las aristas solo pueden existir entre un bloque y un vértice de corte. Por lo tanto, no pueden existir ciclos impares en $BL(G)$.

2. Supongamos que existe un ciclo par en $BL(G)$:

   - Si el ciclo tiene 4 vértices, esto implica que hay dos bloques distintos que están conectados a dos vértices de corte en común, lo que contradice el Teorema 1.2.
   - Si el ciclo tiene más de 4 vértices, entonces podemos encontrar, en $BL(G)$, dos bloques conectados mediante dos caminos distintos que pasan por vértices de corte diferentes; lo que implica que existe un camino alternativo entre los bloques que evita alguno de los vértices de corte, lo que contradice que ese vértice sea de corte en G, pues su eliminación no desconectaría al grafo G.

     En ambos casos llegamos a una contradicción, por lo tanto, $BL(G)$ no contiene ciclos pares.

   Concluimos que $BL(G)$ no tiene ciclos.

Dado que $BL(G)$ es conexo y acíclico, entonces es un árbol. En cualquier árbol hay al menos dos vértices de grado 1, en $BL(G)$, los vértices de grado 1 corresponden a bloques que están conectados con un único vértice de corte, es decir, son dos bloques hoja de G. □

Nota 4. Sea un grafo G conexo y con al menos un vértice de corte, veamos que el grafo bloque $BL(G)$ es un árbol. Si $BL(G)$ no fuera conexo, como en la Figura 1.34, implica que hay más de una componente conexa tanto en $BL(G)$ como en G, cada bloque sería una componente conexa, lo que contradice que G sea conexo. Entonces $BL(G)$ debe ser conexo.

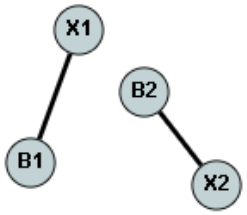
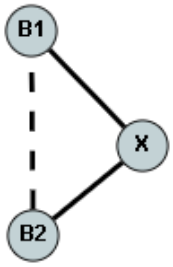


Figura 1.34: Grafo bloque disconexo

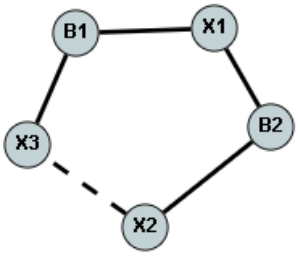
Si $BL(G)$ no fuera acíclico, entonces tendría ciclos impares o pares. Si los ciclos son impares se contradice la definición de grafo bloque, porque los bloques B_1 y B_2 son adyacentes (Figura 1.35a) o los vértices de corte X_3 y X_2 están conectados (Figura 1.35b).

Si los ciclos de $BL(G)$ son pares de 4 vértices (Figura 1.35c), se contradice que dos bloques distintos de un grafo pueden tener a lo sumo un vértice en común. Si son ciclos pares con más de 4 vértices, hay dos caminos diferentes de un bloque a otro, por ejemplo de B_1 a B_2 (Figura 1.35d), tenemos $h_1 = (B_1, B_1X_2, X_2, X_2B_2, B_2)$ y $h_2 = (B_1, B_1X_3, X_3, X_3B_3, B_3, B_3X_1, X_1, X_1B_2, B_2)$, entonces, si eliminamos el vértice X_2, tanto en $BL(G)$, como en el grafo G correspondiente, existe un camino entre los vértices de B_1 y los de B_2, por lo que X_2 no sería de corte y contradice su definición como vértice de corte.

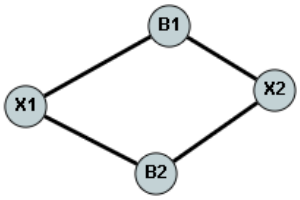
En los dos casos llegamos a contradicciones, por tanto $BL(G)$ no puede contener ciclos.



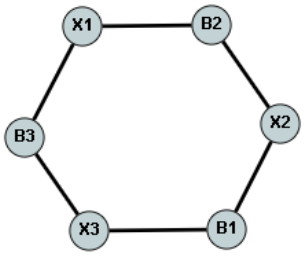
(a) $BL(G)$ con ciclo de 3 vértices



(b) $BL(G)$ con ciclo de 5 vértices



(c) $BL(G)$ con ciclo de 4 vértices



(d) $BL(G)$ con ciclo de 6 vértices

Figura 1.35: Grafo bloque con ciclos

Como $BL(G)$ es conexo y acíclico, entonces es un grafo árbol. Además, en los grafos hay al menos dos vértices de grado 1, como en la Figura 1.36, aquellos vértices son bloques que están conectados con un vértice de corte de G, por lo tanto son bloques hoja.

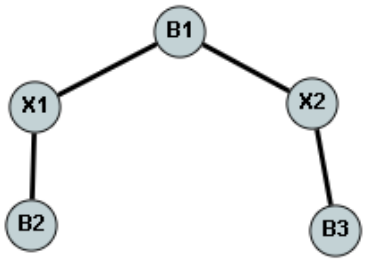


Figura 1.36: Grafo bloque $BL(G)$

Definición 1.42. Un grafo G es $k-$conexo, si el número de conectividad es mayor o igual k.

No siempre la representación gráfica de los grafos bipartitos nos permite identificar rápidamente los conjuntos U y W de vértices, por ello es siguiente teorema nos entrega otra caracterización. Además, la caracterización que se presenta con el teorema nos va a permitir determinar la mínima cantidad de colores para los vértices de algunas de las familias que estamos presentando.

Teorema 1.3. Un grafo es bipartito si y sólo si no tiene ciclos con un número impar de vértices.

Demostración. 1. Vamos a demostrar que si G es un grafo bipartito entonces no tiene ciclos impares.

Para ello, consideremos los subconjuntos de vértices que proporciona el grafo bipartito, $U = \{U_1, U_2, ..., U_m\}$ y $W = \{W_1, W_2, ..., W_n\}$. Si escogemos el vértice U_n, con $n \leq m$, y existe un camino cerrado de U_n a U_n, se cumple que ese camino tiene vértices que van alternando en los subconjuntos U y W, esto es, $h = (U_n, W_m, U_{n-1}, W_{m-1}, ..., W_{m-n}, U_n)$. De esta manera, creamos un camino cerrado que tiene la misma cantidad de vértices de cada subconjunto U y W (porque estamos trabajando con un grafo bipartito), es decir, hay p vértices de U, y hay p vértices de W, por lo tanto en el camino hay $2p$ vértices y aquel camino cerrado, que no tiene vértices repetidos salvo los extremos, es un ciclo par.

2. Ahora demostraremos que si G es un grafo sin ciclos con un número impar de vértices, entonces G es bipartito.

Primero asumimos que G es conexo, pues si no lo fuera basta con aplicar el mismo argumento a cada una de las componentes conexas de G. Luego, para este propósito, definimos dos subconjuntos de vértices U y W del conjunto de vértices del grafo V_G. Para ello seleccionamos un vértice U_1 que ubicamos en U, junto con todos aquellos vértices de V_G que tengan un camino hasta U_1, tal que ese camino sea el más corto y con un número par de aristas.

Por otro lado, en W incluimos a todos los vértices de V_G que tienen un camino con un número impar de aristas hasta U_1 y que además es el más corto. Notemos que U_1 no pertenece a W.

Con esta construcción, cualquier par de vértices que están en U, no son adyacentes; ya que si suponemos que en U existen dos vértices adyacentes U_i y U_j, implicaría que el camino cerrado $h_U = (U_1U_2, ..., U_{i-1}U_i, U_iU_j, U_jU_{j+1}, ..., U_1)$, que contiene los caminos más cortos de longitud par, los que van de U_i a U_1 y de U_j a U_1, y una arista demás (U_iU_j), tenga longitud impar; y por el Teorema 1.1 aquel camino es un ciclo impar, lo que contradice la hipótesis.

De manera análoga concluimos que cualquier par de vértices que están en W no son adyacentes.

Por lo tanto, el conjunto de vértices de G está dividido en dos subconjuntos U y W tales que, $V_G = U \cup W$, donde los vértices pertenecientes a cada subconjunto no son mutuamente adyacentes, entonces el grafo G es bipartito.

□

El siguiente ejemplo busca ilustrar los argumentos de la demostración del Teorema 1.3.

Ejemplo 1.31. Para la primera parte de la prueba, tomamos el grafo bipartito $B_{4,5}$ (Figura 1.37), cuyo conjunto de vértices es $V_{B_{4,5}} = W \cup U$ donde $W = \{W_1, W_2, W_3, W_4, W_5\}$ y $U = \{U_1, U_2, U_3, U_4\}$. Uno de los caminos cerrados que se puede evidenciar en el grafo es $h = (U_1W_2, W_2U_2, U_2W_3, W_3U_3, U_3W_4, W_4U_1)$, el cuál tiene tres vértices de U y tres vértices de W, por lo tanto aquel camino que no repite vértices, a excepción del de inicio y fin, tiene 6 aristas, entonces es un ciclo par.

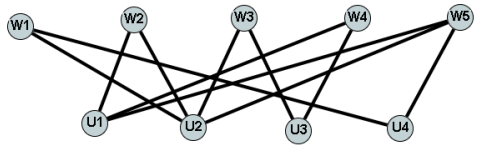


Figura 1.37: Grafo G para primer literal

Para la segunda parte, tenemos el grafo G sin ciclos impares (Figura 1.38), escogemos el vértice A y a partir de este los vértices se pueden dividir en dos subconjuntos:

Subconjunto U que incluye a A y aquellos cuyo camino más corto a A es de longitud par, es decir, $U = \{A, E, C\}$.

Y el subconjunto W que contiene a los vértices con un camino corto, impar hasta A, $W = \{F, H, B, D\}$.

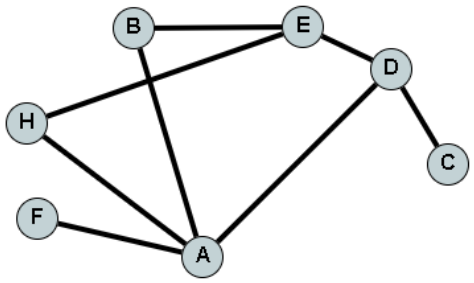


Figura 1.38: Grafo sin ciclos impares G

Como se evidencia en la figura, los vértices de cada subconjunto no son adyacentes entre si. Y si, por ejemplo, existiera una arista entre F y H, vértices de W, el camino cerrado $h_W = (AF, FH, HA)$ sería un ciclo impar, lo que contradice la definición del grafo G.

En el siguiente capítulo los grafos $2-$conexos (Definición 1.42) van a ser relevantes en las argumentaciones dadas, luego los siguientes teoremas nos ofrecen otra caracterización de ellos, vinculándolos con caminos y ciclos.

Teorema 1.4. Sea G un grafo conexo con tres o más vértices. G es $2-$conexo si y solo si para cada par de vértices de G existen dos caminos internamente disjuntos, es decir, los vértices en cada camino son distintos, a excepción de los vértices iniciales y finales.

Demostración. Como el teorema es una afirmación de "si y solo si", demostraremos ambas implicaciones por separado:

1. Probaremos que si tenemos dos caminos internamente disjuntos entonces G es $2-$conexo por contraposición [2].

   Suponemos que G no es $2-$conexo. Por la definición de $k-$conexo, el número de conectividad de G debe ser menor que 2, entonces existe al menos un vértice de corte C en G.

[2]Dada una afirmación p entonces q, se puede demostrar por medio de su equivalencia lógica, probando que $\neg q$ entonces $\neg p$ es verdadero

Si eliminamos a C del grafo G, se crean al menos dos componentes conexas C_1 y C_2 en $G-\{C\}$, luego existen dos vértices $V_1 \in C_1$ y $V_2 \in C_2$ tal que no hay un camino de V_1 a V_2 en $G-\{C\}$. Por lo tanto, todos los caminos que conectan a V_1 y V_2 en G necesariamente deben pasar por el vértice de corte C, es decir, que los caminos tienen un vértice en común. Por lo tanto, no existen dos caminos internamente disjuntos entre V_1 y V_1. Esto prueba la contraposición, en consecuencia que queda demostrado que si tenemos dos caminos internamente disjuntos entonces G es 2−conexo.

2. Para demostrar que si G es 2−conexo entonces tiene dos caminos internamente disyuntos, lo haremos por inducción sobre la longitud de un camino entre dos vértices cualesquiera de G. Sea un grafo G que es 2−conexo

   a) Si existe una arista entre los vértices V_1 y V_2 la longitud de ese camino va ser igual a 1; pero como G tiene por lo menos tres vértices y es conexo, existe un vértice V_3 tal que al eliminar el vértice V_1 de G, $G-\{V_1\}$ es conexo, pues G es 2−conexo; entonces existe un camino de V_3 a V_2, y de forma análoga, al eliminar V_2 de G, $G-\{V_2\}$ es conexo, luego existe un camino de V_3 a V_1, por lo tanto existen al menos dos caminos internamente disyuntos entre V_1 y V_2, el dado por la arista V_1, V_1V_2, V_2 y el que se obtiene de unir los caminos de V_1 a V_3 y de V_3 a V_2.

   b) Suponemos que para todo par de vértices cualesquiera de G cuyo camino más corto es de longitud menor a k, tal que $k \geq 2$, existe al menos dos caminos internamente disjuntos entre los vértices.

   c) Sean V_1 y V_2 dos vértices tales que la longitud de su camino mínimo entre ellos es k. Sea $h_1 = (V_1, \cdots, V_3, V_3V_2, V_2)$ un camino de longitud k, donde V_3 precede a V_2 en h_1, entonces en h_1 hay un subcamino de V_1 a V_3 cuya longitud es menor que k, por lo que, por la hipótesis del literal b, existen dos caminos internamente disjuntos entre V_1 y V_3, sean $h_2 = (V_1, \cdots, V_3)$ y $h_3 = (V_1, \cdots, V_3)$. Como G es 2−conexo, el grafo $G-\{V_3\}$ es conexo entonces encontramos un camino entre V_1 y V_2, por lo que existe un camino h_4 entre V_1 y V_2 que no contiene al vértice V_3 en el grafo G. Sea V_4 un vértice que precede a V_2 en $h_4 = (V_1, \cdots, V_4, V_4V_2, V_2)$; V_4 puede pertenecer a h_2 o h_3, para no perder generalidad, supongamos

que V_4 pertenece a h_2. Entonces podemos construir dos caminos internamente disjuntos entre V_1 y V_2:

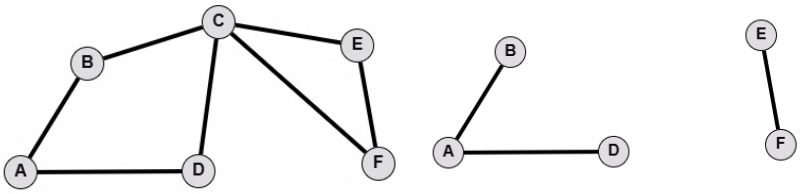
1) Un primer camino está entre el subcamino de h_2 desde V_1 hasta V_4 junto con el subcamino de h_4 desde V_4 hasta V_2, esto es $(V_1, \cdots, V_4, V_4V_2, V_2)$.
2) Para el segundo camino, consideramos dos casos:
   - Si V_2 pertenece al camino $h_3 = (V_1, \cdots, V_2, V_2V_3, V_3)$, entonces el subcamino de h_3 entre V_1 y V_2 es internamente disjunto al primer camino.
   - Si V_2 no pertenece al camino h_3, entonces tenemos un camino al unir h_3 con el subcamino de h_1 entre V_3 y V_2, esto es $(V_1, \cdots, V_3, V_3V_2, V_2)$ un camino internamente disjunto al primero.

Por lo tanto, hay al menos dos caminos internamente disjuntos entre V_1 y V_2

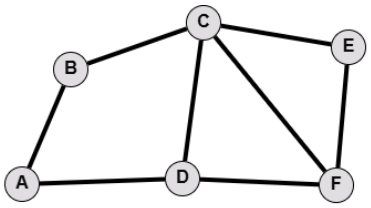
□

Ejemplo 1.32. Ilustramos los casos mencionados en la demostración del Teorema 1.4.

Suponemos que grafo G no es 2−conexo (Figura 1.39a), entonces el grafo tendrá un vértice de corte C. Al eliminar el vértice C del grafo, nuestro grafo $G - \{C\}$ (Figura 1.39b) tiene dos componentes conexas con vértices $C_1 = \{A, B, C, D\}$ y $C_2 = \{E, F\}$. En el grafo G todo camino de A a F necesariamente tiene que pasar por C. Por lo tanto, para que haya dos caminos disyuntos entre A y F el grafo G debe ser 2−conexo (Figura 1.39c), cuyos vértices de corte son C, D y cuyos caminos de A a F serían (A, AD, D, DF, F) y (AB, B, BC, C, CF, F).



(a) Grafo G (b) Grafo $G - \{C\}$



(c) Grafo G 2-conexo

Figura 1.39: Primera parte de la demostración

Para la segunda parte de la demostración suponemos que G es 2−conexo (Figura 1.40). Consideramos los vértices D y F cuyo camino mínimo entre ellos es $h = (D, DA, A, AC, C, CF, F)$ el cual tiene longitud 3. Sea C el vértice que precede a F en el camino h. Como la distancia entre D y C es menor que 3, existen dos caminos internamente disyuntos entre D y C $h_1 = (D, DA, A, AC, C)$ y $h_2 = (D, DE, E, EB, B, BC, C)$.

Además, como G es 2−conexo, existe un camino entre D y F que evita al vértice C; este camino es $h_3 = (D, DE, E, EB, B, BF, F)$.

A partir de estos caminos construimos dos caminos entre D y F: Primero h_1 de D a C y $h_4 = (C, CF, F)$, es decir, $h_1 \cup h_4 = (D, DA, A, AC, C, CF, F)$. Y segundo se obtiene tomando el subcamino h_2 de D hasta B y continuando por el camino h_3 desde B hasta F,es decir, $h_5 = (D, DE, E, EB, B, BF, F)$. Por lo tanto, obtenemos dos caminos internamente disyuntos entre D y F.

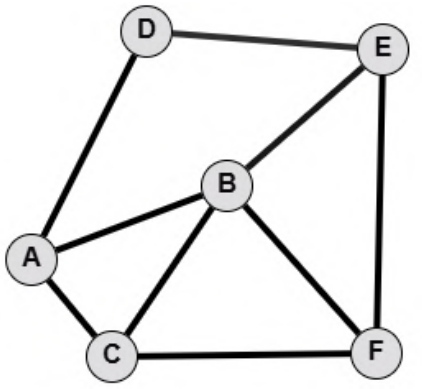


Figura 1.40: Segunda parte de la demostración

Definición 1.43. Una **adición de camino** a un grafo G es conectar a G un camino entre dos vértices existentes de G, de tal manera que las aristas y los vértices internos del camino no pertenezcan a G.

Definición 1.44. Una **adición de ciclo** es la unión a G de un ciclo que tiene exactamente un vértice en común con G.

Definición 1.45. Una **Síntesis de Whitney-Robbins** de un grafo G es una secuencia de grafos en la que se parte de un grafo conexo H_1, para añadirle sucesivamente, en cada paso, caminos o ciclos cuyos extremos son vértices existentes en el grafo anterior. Este proceso continúa hasta que en el paso n obtenemos el grafo G, esto es, obtenemos la secuencia de grafos $H_1, H_2, ..., H_{n-1}, H_n = G$. Si en cada paso H_i se obtiene unicamente mediante la adición de caminos, entonces se denomina una **Sintesis de Witney**

Ejemplo 1.33. Sea el grafo conexo H_1 de la Figura 1.41, con $V_{H_1} = \{A, D, C, B\}$ y $E_{H_1} = \{AD, DB, DC\}$ iniciamos un proceso de Síntesis de Whitney-Robbins del grafo G a partir del grafo H_1. Primero, agregamos la arista BC para obtener el grafo H_2. Luego, añadimos el ciclo con los vértices A, E, F, B, A para obtener el grafo H_3. Finalmente, agregamos el camino (E, G, H, I, C) para obtener a $H_4 = G$, resultado de este proceso de grafos.

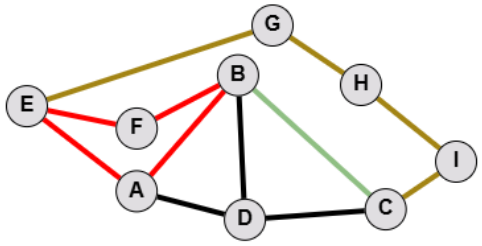


Figura 1.41: Sintesis de Whitney

Teorema 1.5. Un grafo G es $2-$conexo si y solo si G es un ciclo o es una síntesis de Whitney que parte desde un ciclo.

Demostración. El enunciado del teorema es un "si y solo si", por ello vamos a probar ambas implicaciones.

1. Si G es una síntesis de Whitney a partir de un ciclo entonces G es $2-$conexo.

   Supongamos que $C = H_1, H_2, H_3, ..., H_n = G$ es una síntesis de Whitney de G a partir de un ciclo C. Como el ciclo C es un grafo $2-$conexo, entonces el grafo que resulta de unir un camino a C es $2-$conexo, pues la propiedad de que para todo par de vértices existen dos caminos internamente disyuntos entre ellos, se mantiene bajo la unión de caminos. Por tanto, cada H_i es $2-$conexo, en particular G.

2. Si G es $2-$conexo entonces es una síntesis de Whitney.

   Como el grafo G es $2-$conexo, para cada par de vértices existen dos caminos internamente disyuntos, por tanto la unión de estos caminos forman un ciclo.

   Sea C un ciclo en el grafo G y consideramos H el conjunto de subgrafos S de G tales que cada S es una síntesis de Whitney a partir del ciclo C.

   Obtenemos que H es diferente de vacío pues el mismo ciclo C pertenece a H. Seleccionamos ahora un subgrafo de H con la mayor cantidad de aristas, al que notamos H^*. Demostraremos que $H^* = G$.

Supongamos que H^* es diferente de G. Como G es conexo, debe existir una arista $e = VW$ en $E_G - E_{H^*}$ tal que uno de sus vértices este en H^*, supongamos que es V.

Como G es $2-$conexo, entre los vértices V y W existen dos caminos internamente disyuntos por el Teorema 1.4, en particular el camino (V, VW, W), luego uniendo estos dos caminos tenemos un ciclo que contiene a la arista e.

Como V no es un vértice de corte, debe haber un ciclo C' que contiene a la arista e y que tiene vértices en H^* diferentes de V. Sea Z el primer vértice en C' tal que el ciclo retorna a H^*, es decir, el camino, contenido en C', de Z a V que no contiene a VW, esta en H^*. Por tanto, el camino de V a Z que contiene a VW es una unión que se hace a H^* y el grafo obtenido, contradice la maximalidad de H^*. Entonces $H^* = G$ por lo tanto G es una sísntesis de Whitney.

□

1.6. Coloración de grafos

Como hemos mencionado en el documento, nuestro interés se centra en colorear los vértices de un grafo; por tanto, en esta sección exponemos las definiciones básicas directamente relacionadas con este proceso.

Definición 1.46. Una $k-$**coloración** de los vértices de un grafo G es una función $f : V_G \to C$ que le asigna a cada vértice de V_G un color (elemento) de un conjunto C de k colores (elementos) diferentes.

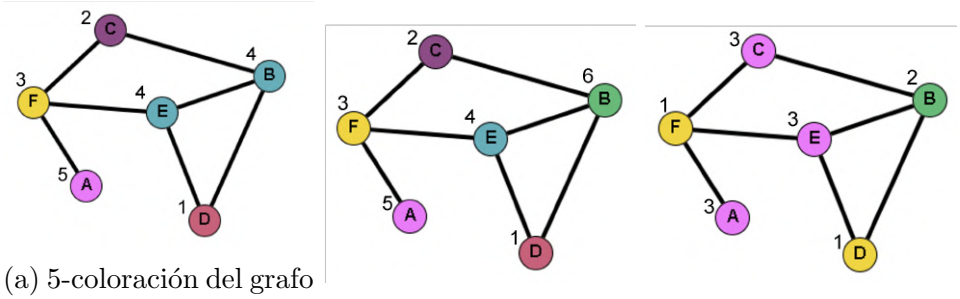
Definición 1.47. Un grafo es $k-$**coloreable** si tiene una $k-$coloración de vértices, tal que, dos vértices adyacentes no tienen asignado el mismo color.

Definición 1.48. El **número cromático** de un grafo es el mínimo número de colores diferentes que son necesarios para que el grafo sea coloreable, y lo denotamos $\chi(G)$.

Ejemplo 1.34. En la Figura 1.42 presentamos tres formas de colorear los vértices de un grafo G para hacer la distinción entre $k-$coloración, $k-$coloreable y el número cromático.

Los vértices del grafo G (Figura 1.42a) fueron coloreados con la lista de colores $C = (1, 2, 3, 4, 5)$, es decir, el grafo tiene una $5-$coloración. Pero en la

Figura 1.42b a los vértices de G que son adyacentes, como E y B, se les asigna un color distinto, por lo tanto con la lista de colores $C = (1, 2, 3, 4, 5, 6)$ el grafo es $6-$coloreable. Por su parte, en la Figura 1.42c mostramos que 3 es el mínimo de colores necesarios para que el grafo G sea coloreable.



(a) 5-coloración del grafo G (b) Grafo G 6-coloreable (c) Grafo G 3-coloreable

Figura 1.42: Coloreados de los vértices del grafo G

Definición 1.49. Una **clase de color** asociada a una $k-$coloración de los vértices de un grafo G, es un subconjunto de vértices de G los cuales tienen el mismo color.

Ejemplo 1.35. En el ejemplo anterior, asociado a la $3-$coloración del grafo G (Figura 1.42c), el conjunto de los vértices A, E y C son la clase de color 3; el conjunto de los vértices F y D son la clase de color 1; por último, el vértice B es la clase de color 2.

Definición 1.50. Dada una lista de colores $C = (1, 2, ..., n)$ y un grafo G cuyo conjunto de vértices es V_G con n elementos, la **función de coloración** de vértices es $f : V_G \to C$ donde se le asigna a cada vértice V_i el color de menor valor disponible de C, atendiendo al orden natural de la lista de colores, tal que no tenga el mismo color que sus vecinos.

Ejemplo 1.36. Para el grafo O representado en la Figura 1.43, primero ordenamos los vértices alfabéticamente, es decir, $A, B, C, D, E, F, G, H, I$. Luego, definimos una lista de colores.

Con esto vamos recorriendo cada vértice en orden para asignarle el menor color disponible que no haya sido usado por sus vecinos. Por ejemplo, al vértice A le corresponde el primer color (amarillo) porque es el primer vértice; después B también recibe el color 1, ya que A no es su vecino y 1 sigue estando

disponible. En cambio, C tiene asignado el color 2 (rojo), porque su vecino A quita la posibilidad de escoger el color 1; así el menor color disponible de la lista para el vértice E es 2, continuando la asignación de colores atendiendo a ese criterio obtenemos que el grafo O es $2-$coloreable.

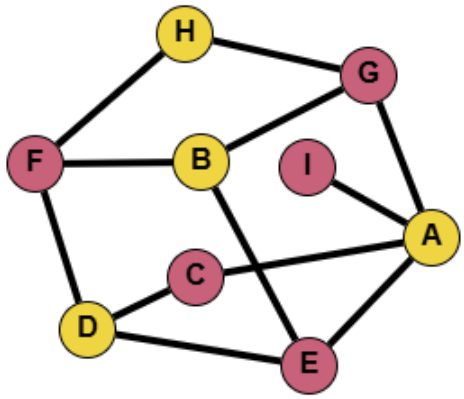


Figura 1.43: Función de coloración en el grafo O

1.7. Grafos en Python

Nosotras complementamos nuestro estudio con algunos códigos de Python que permiten crear, analizar y encontrar elementos asociados a los grafos.

En Python recurrimos las librerías Networkx y Matplotlib, para trabajar con grafos y generar sus representaciones gráficas.

En el Código 1.1 mostramos como generar un grafo a partir del conjunto de vértices y aristas que ingrese un usuario, la representación asociada e información sobre la adyacencia entre vértices, como la vecindad, los grados y la secuencia de grados del grafo.

```
1 #Librerías
2 import networkx as nx
3 import matplotlib.pyplot as plt
4
5 #Grafo simple vacío
6 G = nx.Graph()
7
```

```
#Solicitud de vértices y aristas
Cantidad_vértices = int(input("Ingrese la cantidad de vértices del grafo: "))
Vertices = input("Ingrese los vértices del grafo (separados por comas): ")
Vertices = list(Vertices.split(","))
while len(Vertices) != Cantidad_vértices:
  print("Los vértices ingresados no coinciden con la cantidad de vértices mencionada")
  Cantidad_vértices = int(input("Ingrese la cantidad de vértices del grafo: "))
  Vertices = input("Ingrese los vértices del grafo (separados por comas): ")
  Vertices = list(Vertices.split(","))
G.add_nodes_from(Vertices)

Cantidad_aristas = int(input("Ingrese la cantidad de aristas: "))
for j in range(Cantidad_aristas):
    arista = input("Ingrese los vértices de una arista (como A-B): ")
    arista = list(arista.split("-"))
    G.add_edge(arista[0], arista[1])

#Representación gráfica
print("Grafo: ")
fig, ax = plt.subplots(figsize=(4, 4))
pos = nx.spring_layout(G)

nx.draw_networkx_nodes(G, pos, node_color='white', edgecolors='black')
nx.draw_networkx_edges(G, pos, edge_color='black')
nx.draw_networkx_labels(G, pos)
ax.axis('off')
plt.show()

#Información sobre la adyacencia en el grafo
Vecinos = dict(G.adj)
for k in range(len(Vecinos)):
  print("Vecindad de", list(Vecinos.keys())[k], ": ",
```

```
    list(G.neighbors(list(Vecinos.keys())[k])))
40
41 Grados = dict(G.degree())
42 print("Los grados de los vértices del grafo son: ",
    Grados)
43 Grado_max = max(Grados.values())
44 print("El grado máximo del grafo es: ", Grado_max)
45 Grado_min = min(Grados.values())
46 print("El grado mínimo del grafo es: ", Grado_min)
47 Secuencia_grados = sorted(Grados.values())
48 print("La secuencia de grados es: ", Secuencia_grados)
```

Código 1.1: Elementos básicos de los grafos

Como nuestro estudio está enfocado en los grafos simples, para crear este tipo de grafos necesitamos de la línea de código `nx.Graph`, la cual inicia un grafo simple vacío al que se le agregan los vértices y aristas que ingresen los usuarios. Los vértices y aristas se pueden agregar de uno en uno con `Grafo.add_node(vértice)` y `Grafo.add_edge(vértice1, vértice2)` o desde listas `Grafo.add_node_from(lista de vértices)` y `Grafo.add_edge_from(lista de aristas)`.

En cuanto a la representación gráfica, usamos los códigos `nx.draw(Grafo ,opciones de edición)` y `plt.show()`. Además, para establecer estilos a los vértices, aristas y etiquetas utilizamos códigos como `nx.draw_networkx_nodes(G,atributos)`.

Para obtener información que surge a partir de la adyacencia de vértices en el grafo, usamos las líneas de código:

- Vecinos:
  - `Grafo.neighbors(Vértice)`: Entrega un iterador con los vecinos del vértice ingresado.
  - `Grafo.adj`: Es un diccionario de los vecinos de todos los vértices del grafo.
- Grados:
  - `Grafo.degree(Vértice)`: Entrega el grado del vértice ingresado.
  - En nuestro caso, con `max` y `min` obtuvimos el grado máximo y mínimo del grafo, al aplicarlos en los valores del diccionario que guardamos en la variable "Grados" (Código 1.1, linea 41).

Ejemplo 1.37. La Figura 1.44 muestra un ejemplo del Código 1.1 ejecutado en Python.

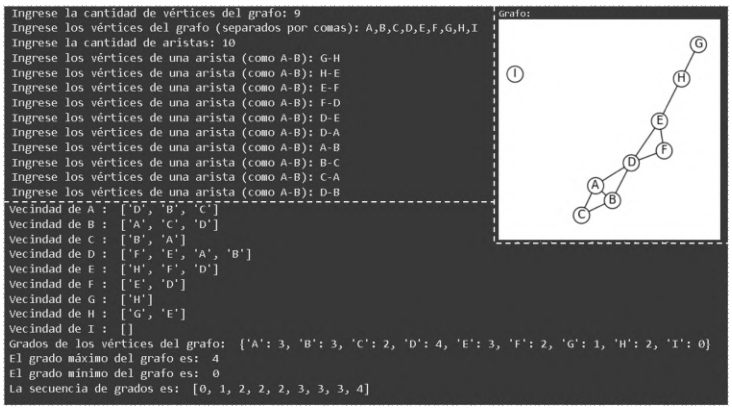


Figura 1.44: Ejemplo de elementos básicos en Python

Continuando con los subgrafos en el Código 1.2 le solicitamos al usuario un grafo inicial, el cual llamamos "original", para obtener de allí un subgrafo inducido, propio y de eliminación, dándole la opción al usuario de seleccionar el subconjunto de vértices requeridos para su definición.

```
#Librerías
import networkx as nx
import matplotlib.pyplot as plt
import random
#Función para las representaciones gráficas de los grafos
def representacion_grafica(G):
  fig, ax = plt.subplots(figsize=(4, 4))
  pos = nx.spring_layout(G)
  nx.draw_networkx_nodes(G, pos, node_color='white', edgecolors='black')
  nx.draw_networkx_edges(G, pos, edge_color='black')
  nx.draw_networkx_labels(G, pos)
  ax.axis('off')
  plt.show()

#Grafo original
G = nx.Graph()

```

```
print("Grafo original")
Cantidad_vértices = int(input("Ingrese la cantidad de vértices del grafo: "))
Vertices = input("Ingrese los vértices del grafo (separados por comas): ")
Vertices = list(Vertices.split(","))
while len(Vertices) != Cantidad_vértices:
  print("Los vértices ingresados no coinciden con la cantidad de vértices mencionada")
  Cantidad_vértices = int(input("Ingrese la cantidad de vértices del grafo: "))
  Vertices = input("Ingrese los vértices del grafo (separados por comas): ")
  Vertices = list(Vertices.split(","))
G.add_nodes_from(Vertices)

Cantidad_aristas = int(input("Ingrese la cantidad de aristas: "))
for j in range(Cantidad_aristas):
    arista = input("Ingrese los vértices de una arista (como A-B): ")
    arista = list(arista.split("-"))
    G.add_edge(arista[0], arista[1])

#Subgrafo inducido
print("Subgrafo inducido: ")
Vertices_subind = input("Ingrese los vértices del subgrafo inducido (separados por comas): ")
Vertices_subind = Vertices_subind.split(",")
subind = nx.induced_subgraph(G, Vertices_subind)

#Subgrafo propio
print("Subgrafo propio: ")
Vertices_subprop = input("Ingrese los vértices del subgrafo propio (separados por comas): ")
Vertices_subprop = Vertices_subprop.split(",")
subprop = nx.induced_subgraph(G,Vertices_subprop).copy()

Cantidad_aristas_subprop = int(input("Ingrese la cantidad de aristas que quiere en el subgrafo: "))
```

```
n = subprop.number_of_edges() - Cantidad_aristas_subprop
arista_prop = random.sample(list(subprop.edges),n)
subprop.remove_edges_from(arista_prop)

#Subgrafo eliminación de vértices
print("Subgrafo eliminación de vértices: ")
Vertices_Eliminados = input("Ingrese los vértices que
   quiere eliminar (separados por comas): ")
Vertices_Eliminados = Vertices_Eliminados.split(",")
subeliminacion = G.copy()
subeliminacion.remove_nodes_from(Vertices_Eliminados)

#Representaciones gráficas
print("Grafo original: ")
representacion_grafica(G)
print("Subgrafo inducido: ")
representacion_grafica(subind)
print("Subgrafo propio: ")
representacion_grafica(subprop)
print("Subgrafo eliminación de vértices: ")
representacion_grafica(subeliminacion)
```

Código 1.2: Subgrafos

Para crear un subgrafo inducido, empleamos la función `nx.induced_subgraph(G,Vertices_del_subgrafo)`. Para el subgrafo que corresponde a la eliminación de vértices basta con la línea `Subgrafo.remove_nodes_from (lista de vértices)`

Con los subgrafos propios no hay una función determinada, por ello, lo creamos a partir de un subgrafo inducido con los vértices que indique el usuario, luego se seleccionan unas aristas aleatorias, con la línea `random.sample(list(aristas_del_subgrafo_inducido),cantidad_de_aristas_a_eliminar)`, para eliminarlas del subgrafo con la instrucción `Subgrafo.remove_edges_from(lista_de_aristas)`.

Ejemplo 1.38. En la Figura 1.45 hay un ejemplo de lo que resulta cuando se ejecuta el Código 1.2.

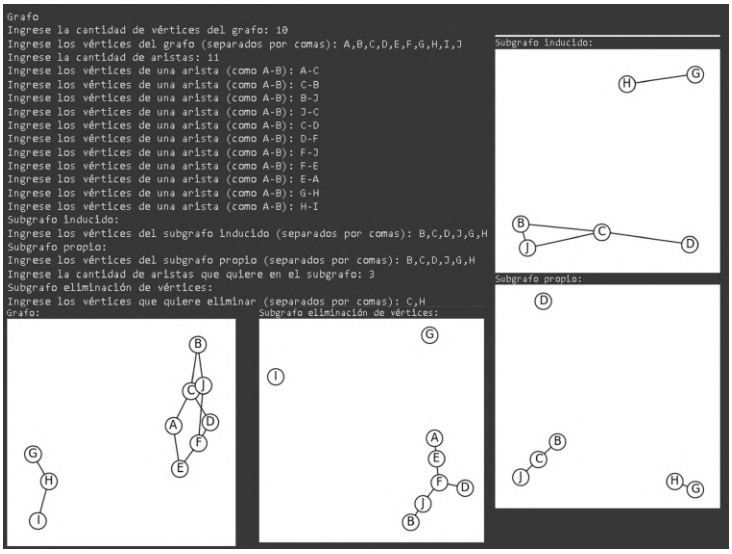


Figura 1.45: Subgrafos en Python

En el Código 1.3 ofrecemos las líneas correspondientes para operaciones entre dos grafos que ingresa el usuario.

```
#Librerías
import networkx as nx
import matplotlib.pyplot as plt

#Función para las representaciones gráficas
def representacion_grafica(G):
  fig, ax = plt.subplots(figsize=(4, 4))
  pos = nx.spring_layout(G)
  nx.draw_networkx_nodes(G, pos, node_color='white',
   edgecolors='black')
  nx.draw_networkx_edges(G, pos, edge_color='black')
  nx.draw_networkx_labels(G, pos)
  ax.axis('off')
  plt.show()

#Primer grafo
G1 = nx.Graph()

print("Grafo 1: ")
Cantidad_vérticesG1 = int(input("Ingrese la cantidad de
```

```
   vértices del grafo: "))
VerticesG1 = input("Ingrese los vértices del grafo (
   separados por comas): ")
VerticesG1 = list(VerticesG1.split(","))
while len(VerticesG1) != Cantidad_vérticesG1:
  print("Los vértices ingresados no coinciden con la
   cantidad de vértices mencionada")
  Cantidad_vérticesG1 = int(input("Ingrese la cantidad
   de vértices del grafo: "))
  VerticesG1 = input("Ingrese los vértices del grafo (
   separados por comas): ")
  VerticesG1 = list(VerticesG1.split(","))
G1.add_nodes_from(VerticesG1)

Cantidad_aristasG1 = int(input("Ingrese la cantidad de
   aristas: "))
for j in range(Cantidad_aristasG1):
    aristaG1 = input("Ingrese los vértices de una arista
    (como A-B): ")
    aristaG1 = list(aristaG1.split("-"))
    G1.add_edge(aristaG1[0], aristaG1[1])

#Segundo grafo
print("Grafo 2: ")
G2 = nx.Graph()
Cantidad_vérticesG2 = int(input("Ingrese la cantidad de
   vértices del grafo: "))
VerticesG2 = input("Ingrese los vértices del grafo (
   separados por comas): ")
VerticesG2 = list(VerticesG2.split(","))
while len(VerticesG2) != Cantidad_vérticesG2:
  print("Los vértices ingresados no coinciden con la
   cantidad de vértices mencionada")
  Cantidad_vérticesG2 = int(input("Ingrese la cantidad
   de vértices del grafo: "))
  VerticesG2 = input("Ingrese los vértices del grafo (
   separados por comas): ")
  VerticesG2 = list(VerticesG2.split(","))
G2.add_nodes_from(VerticesG2)

```

```
Cantidad_aristasG2 = int(input("Ingrese la cantidad de
    aristas: "))
for j in range(Cantidad_aristasG2):
    aristaG2 = input("Ingrese los vértices de una arista
    (como A-B): ")
    aristaG2 = list(aristaG2.split("-"))
    G2.add_edge(aristaG2[0], aristaG2[1])

#Unión
Union = nx.union(G1, G2)

#Suma
Suma = nx.full_join(G1, G2)

#Producto cartesiano
Producto_cartesiano = nx.cartesian_product(G1, G2)

#Representación grafica
print("Grafo 1: ")
representacion_grafica(G1)
print("Grafo 2: ")
representacion_grafica(G2)

print("Unión: ")
representacion_grafica(Union)
print("Suma: ")
representacion_grafica(Suma)
print("Producto cartesiano: ")
representacion_grafica(Producto_cartesiano)
```

Código 1.3: Operaciones entre grafos

Las líneas de códigos para las operaciones son intuitivas: para el grafo unión utilizamos `nx.union(Grafo 1, Grafo 2)`; para el grafo suma, `nx.full_join(Grafo 1, Grafo2)`, y el grafo resultante del producto cartesiano lo obtenemos con `nx.cartesian_product(Grafo 1, Grafo 2)`.

Ejemplo 1.39. La Figura 1.46 muestra un ejemplo de lo que surge al ejecutar el código correspondiente del Código 1.3.

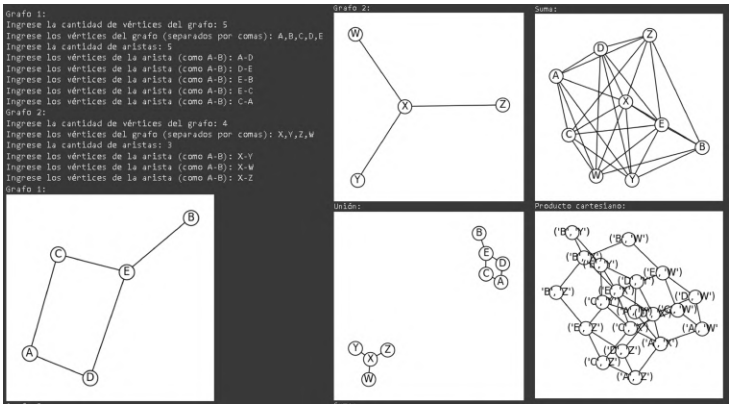


Figura 1.46: Operaciones en Python

En cuanto a la conectividad, hay algunos elementos como los cliques, el conjunto de independencia y los bloques que tienen varias funciones dependiendo de algoritmos heurísticos que permiten su búsqueda, o no hay funciones que permitan generar aquellos elementos. En el Código 1.4 presentamos algunas líneas simples que permiten estudiar la conectividad de un grafo.

```
#Librerías
import networkx as nx
import matplotlib.pyplot as plt
import random

#Función para las representaciones gráficas
def representacion_grafica(G):
  fig, ax = plt.subplots(figsize=(4, 4))
  pos = nx.spring_layout(G)
  nx.draw_networkx_nodes(G, pos, node_color='white',
   edgecolors='black')
  nx.draw_networkx_edges(G, pos, edge_color='black')
  nx.draw_networkx_labels(G, pos)
  ax.axis('off')
  plt.show()

#Grafo simple
G = nx.Graph()

Cantidad_vértices = int(input("Ingrese la cantidad de vé
   rtices del grafo: "))
```

```
Vertices = input("Ingrese los vértices del grafo (separados por comas): ")
Vertices = list(Vertices.split(","))
while len(Vertices) != Cantidad_vértices:
  print("Los vértices ingresados no coinciden con la cantidad de vértices mencionada")
  Cantidad_vértices = int(input("Ingrese la cantidad de vértices del grafo: "))
  Vertices = input("Ingrese los vértices del grafo (separados por comas): ")
  Vertices = list(Vertices.split(","))
G.add_nodes_from(Vertices)

Cantidad_aristas = int(input("Ingrese la cantidad de aristas: "))
for j in range(Cantidad_aristas):
    arista = input("Ingrese los vértices de una arista (como A-B): ")
    arista = list(arista.split("-"))
    G.add_edge(arista[0], arista[1])

#Camino simple
Vertice_inicio = input("Ingrese el vértice de inicio del camino: ")
Vertice_fin = input("Ingrese el vértice de fin del camino: ")
if nx.has_path(G, Vertice_inicio, Vertice_fin):
  camino_simple = list(nx.all_simple_paths(G, Vertice_inicio, Vertice_fin))
  print("Los caminos simples son:", camino_simple)
else:
  print("No existe un camino simple entre los vértices", Vertice_inicio, "y", Vertice_fin)

#Grafo conexo
if nx.is_connected(G):
  print("El grafo es conexo")
else:
  print("El grafo no es conexo")

```

```
#Componentes conexas
Componentes_conexas = list(nx.connected_components(G))
print("Las componentes conexas son: ",
   Componentes_conexas)

#Cliques y conjuntos de independencia*
Cliques = list(nx.find_cliques(G))
print("Cliques: ", Cliques)
Cardinal = []
for i in Cliques:
  Cardinal.append(len(i))
Número_clique = max(Cardinal)
print("El número clique del grafo es: ", Número_clique)

Independencia = list(nx.maximal_independent_set(G))
print("Un conjunto de independencia es: ", Independencia
   )

#Vértice de corte
Vertices_corte = list(nx.articulation_points(G))
print("Los vértices de corte son: ", Vertices_corte)

Conjunto_vertices_corte = list(nx.minimum_node_cut(G))
print("Los conjuntos de vértices de corte son: ",
   Conjunto_vertices_corte)

#Conectividad
Conectividad = nx.node_connectivity(G)
print("Conectividad: ", Conectividad)
print("El grafo es", Conectividad, "-conexo")

#Representación grafica
print("Grafo: ")
representacion_grafica(G)
```

Código 1.4: Conectividad

Para obtener los caminos simples entre dos vértices de un grafo, que ingresa el usuario, primero tenemos que verificar la existencia de al menos un camino entre los vértices utilizando la expresión evaluable `nx.has_path( Grafo, vértice de inicio, vértice final)`. Si existe un camino, em-

pleamos la función `nx.all_simple_paths(Grafo, vértice inicial, vértice final)` para obtener todos los caminos simples que existen entre esos dos vértices.

En Python sabemos si un grafo es conexo empleando la expresión evaluativa `nx.is_connected(Grafo)`. Además, las componentes conexas del grafo las listamos con la función `nx.connected_components(Grafo)`.

Para obtener los cliques de un grafo dado, en Networkx, existe la función `nx.find_cliques(Grafo, vértices (opcional))` la cual devuelve un iterador con todos los cliques máximales para cada vértice. Con la lista de los cliques del grafo, proponemos recorrer los elementos de la lista para calcular el cardinal de cada subconjunto de vértices y de allí encontrar el más grande para mostrar el número clique. Para el calculo de los conjuntos independientes, Networkx nos ofrecen la función `nx.maximal_independent_set(Grafo)` que entrega un conjunto independiente maximal aleatorio.

Para conseguir los vértices de corte del grafo, nos basta con la función `nx.articulation_points(Grafo)`. Y para el conjunto de vértices de corte usamos la función `nx.minimum_node_cut(Grafo)` que devuelve un conjunto de vértices mínimo que desconecta el grafo. Además, en la conectividad empleamos la función `nx.node_connectivity(Grafo)`.

Ejemplo 1.40. En la Figura 1.47 hay un ejemplo de lo que resulta cuando se ejecuta el Código 1.4.

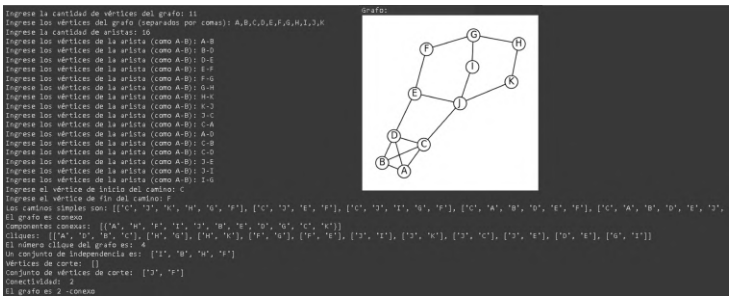


Figura 1.47: Conectividad en Python

Por último, para las familias de grafos Networkx tiene funciones determinadas para generar cada grafo. Aunque no hay una función para crear un grafo bipartito, si existe una que permite verificar si un grafo lo es con la línea `bipartite.is_bipartite(Grafo)` la cual se debe importar de la biblioteca de Networkx. En el Código 1.5 mostramos la generación de grafos asociados a algunas de las familias mencionadas en secciones anteriores.

```
# Librerías
import networkx as nx
import warnings
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")

#Función para las representaciones gráficas de los grafos
def representacion_grafica(G):
  fig, ax = plt.subplots(figsize=(4, 4))
  pos = nx.spring_layout(G)
  nx.draw_networkx_nodes(G, pos, node_color='white', edgecolors='black')
  nx.draw_networkx_edges(G, pos, edge_color='black')
  nx.draw_networkx_labels(G, pos)
  ax.axis('off')
  plt.show()

#Grafo trivial
print("Grafo trivial: ")
GT = nx.trivial_graph()
representacion_grafica(GT)

#Grafo k-regular
print("Grafo k-regular")
GK = nx.Graph()
Verticesk = int(input("Ingrese la cantidad de vértices: "))
Grado = int(input("Ingrese el grado de los vértices: "))
if (Verticesk * Grado) %2 != 0 :
  print("No se puede construir un grafo regular")
else:
  GK = nx.random_regular_graph(Grado, Verticesk)
  print("Grafo: ", Grado, "- regular")
  representacion_grafica(GK)

#Grafo completo
print("Grafo completo")
VerticesK = int(input("Ingrese la cantidad de vértices: "))
```

```
GK = nx.complete_graph(VerticesK)
representacion_grafica(GK)

#Grafo bipartito completo
print("Grafo bipartito completo:")
VerticesKn = int(input("Ingrese la cantidad de vértices
    del primer subconjunto: "))
VerticesKm = int(input("Ingrese la cantidad de vértices
    del segundo subconjunto: "))
GBK = nx.complete_bipartite_graph(VerticesKn, VerticesKm
    )
representacion_grafica(GBK)

#Grafo linea
print("Grafo linea:")
Verticesp = int(input("Ingrese la cantidad de vértices:
    "))
GP = nx.path_graph(Verticesp)
representacion_grafica(GP)

#Grafo ciclo
print("Grafo ciclo:")
Verticesc = int(input("Ingrese la cantidad de vértices:
    "))
GC = nx.cycle_graph(Verticesc)
representacion_grafica(GC)

#Grafo rueda
print("Grafo rueda:")
Verticesw = int(input("Ingrese la cantidad de vértices:
    "))
GW = nx.wheel_graph(Verticesw)
representacion_grafica(GW)
```

Código 1.5: Familias de grafos

Ejemplo 1.41. En la Figura 1.48 están ejemplos del Código 1.5 ejecutado.

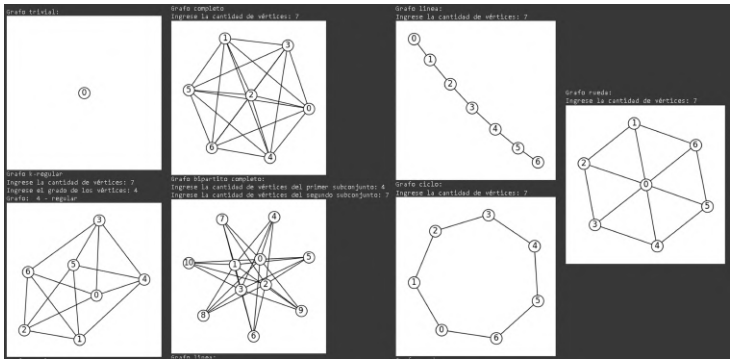


Figura 1.48: Familias de grafos en Python

Capítulo 2

Cotas para la coloración de grafos

El problema de encontrar una coloración para un grafo G donde se utilice la menor cantidad de colores posible, esto es, determinar el número cromático, $\chi(G)$, no tiene una solución teórica general para cualquier grafo; sin embargo, se pueden establecer algunas cotas inferiores y superiores. No obstante, en algunos casos, para algunas familias de grafos dicho número cromático si se puede establecer de forma teórica, como veremos en este capitulo. Siendo así, mostramos nuestro recorrido para este capítulo en la Figura 2.1.

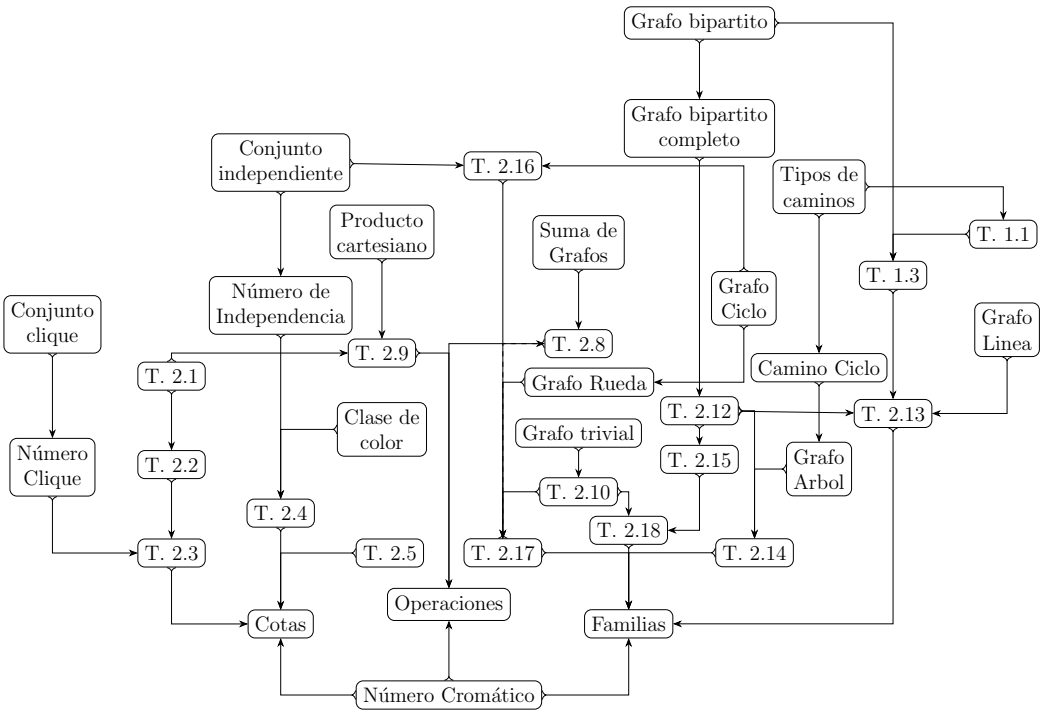


Figura 2.1: Recorrido de las cotas

2.1. Cotas inferiores

Los resultados que mostramos en esta sección se relacionan con la comparación del número cromático para G y el número cromático de sus subgrafos, en particular con los cliques. Asimismo, con los conjuntos independientes, ya que una $k-$coloración de G es una división del conjunto de vértices V_G en conjuntos independientes.

Teorema 2.1. Sea H un subgrafo del grafo G entonces el $\chi(G) \geq \chi(H)$.

Demostración. Si $k = \chi(G)$, es decir, G tiene una $k-$coloración, cuando restringimos dicha coloración al conjunto de vértices de H, dicha restricción es una coloración de H, pues cada arista de H es una arista en G y, por tanto, dos vértices adyacentes en H también lo son en G. Luego $\chi(H) \leq k$. □

Teorema 2.2. Sea G un grafo que tiene k vértices mutuamente adyacentes. Entonces $\chi(G) \geq k$.

Demostración. Sea el subgrafo H inducido por los k vértices mutuamente adyacentes del grafo G. Como todos los vértices son adyacentes entre sí, entonces a cada vértice se le asigna un color distinto, es decir, H es $k-$coloreable. Además, por el Teorema 2.1 tenemos que $\chi(G) \geq \chi(H)$ por lo tanto $\chi(G) \geq k$. □

Teorema 2.3. Sea G un grafo entonces el $\chi(G) \geq \omega(G)$.

Demostración. Dado un grafo G y un subconjunto clique Q con $\omega(G) = |Q|$ (el cardinal de Q) vértices, por el Teorema 2.2 tenemos que $\chi(G) \geq k = \omega(G)$. Entonces $\chi(G) \geq \omega(G)$. □

Teorema 2.4. Sea G un grafo entonces $\chi(G) \geq \lceil \frac{|V_G|}{\alpha(G)} \rceil$.

Demostración. Las clases de colores pueden tener una cantidad de vértices menor o igual $\alpha(G)$, ya que $\alpha(G)$ indica la cardinalidad del conjunto con mayor cantidad de elementos en los que sus vértices no son adyacentes. Por tanto, $\chi(G) \cdot \alpha(G) \geq |V_G|$. Y operando con las desigualdades obtenemos $\chi(G) \geq \lceil \frac{|V_G|}{\alpha(G)} \rceil$, debido a que el número cromático debe ser un número entero [1]. □

Ejemplo 2.1. Vamos a aplicar los teoremas de cotas inferiores para el grafo W (Figura 2.2a).

- Para aplicar el Teorema 2.1 escogemos el subgrafo H, inducido por los vértices $V_H = \{A, D, G, B\}$ (Figura 2.2b), cuyo número cromático es 3. Por lo tanto, $\chi(W) \geq \chi(H) = 3$; lo que significa que el grafo W va a usar al menos 3 colores diferentes.

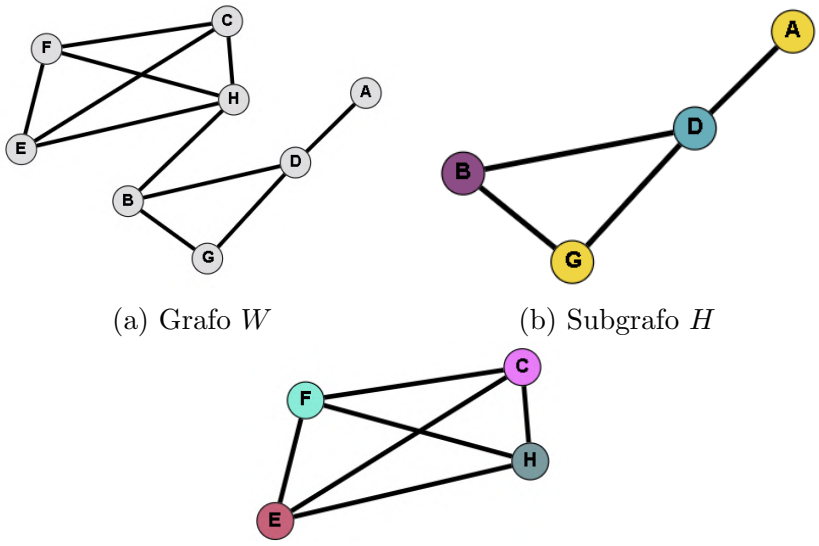
- Para ilustrar el Teorema 2.2 consideramos los vértices A y D, que son mutuamente adyacentes, es decir, $k = 2$. Esto implica que $\chi(W) \geq 2$, entonces W debe usar, por lo menos 2 colores diferentes.

- El número clique del grafo W es igual a 4 que corresponde al cardinal del clique $\{F, C, E, H\}$ (Figura 2.2c), por lo que $\chi(W) \geq 4$. Situación que ilustra el Teorema 2,3.

[1] En esta cota hacemos uso de la función techo $\lceil x \rceil$, que corresponde al menor entero que es mayor o igual que x.

- Por último, relacionado con el Teorema 2.4, tenemos que el número de independencia del grafo W es 3, la cantidad de vértices de W es 8, entonces $\chi(W) \geq \lceil \frac{8}{3} \rceil = 3$.

En este ejemplo observamos que cada teorema ofrece una cota inferior distinta para el número cromático del grafo W. Algunas cotas pueden ser más cercanas al número cromático, como en el tercer ítem, a diferencia de otras como en el segundo ítem (en este caso) es menos precisa, puesto que el grafo W contiene un subgrafo clique de 4 vértices, esto implica que necesariamente requerimos de al menos 4 colores.



(a) Grafo W

(b) Subgrafo H

(c) Subgrafo inducido por el clique del grafo W

Figura 2.2: Cotas inferiores en el grafo W

2.2. Cotas superiores

Los resultados que mostramos en esta sección se relacionan con la comparación del número cromático para G y el grado máximo de G.

El Teorema 2.5 se obtiene al aplicar la función de coloración 1.50 a un orden arbitrario de los vértices del grafo. Esta idea nos permite establecer una cota superior para el número cromático en función del grado máximo.

Teorema 2.5. Sea G un grafo simple entonces el número cromático $\chi(G) \leq \Delta(G) + 1$.

Demostración. Sea G un grafo, fijamos un orden arbitrario de sus vértices. Aplicamos la función de coloración asignando colores sucesivamente a cada vértice siguiendo dicho orden.

Cuando queremos colorear un vértice V_i, revisamos los vecinos de V_i, que tienen tantos grados como $gr(V_i)$ y que no son mayores que el grado máximo de G, es decir, $gr(V_i) \leq \Delta(G)$. Cada vecino ya coloreado de V_i puede impedir a lo sumo un color; por lo tanto, el número de colores prohibidos para el vértice V_i es a lo sumo el grado máximo de $\Delta(G)$. Luego, tenemos que disponer de un color más para colorear a V_i. Entonces, el número de colores requeridos para colorear los vértices de G nunca es superior $\Delta(G) + 1$.

Por los anterior, queda demostrado que $\chi(G) \leq \Delta(G) + 1$ para cualquier vértice del grafo. □

El Teorema 2.6 y su demostración se obtienen al aplicar la Definición de función de coloración 1.50 cuando se selecciona un orden particular de vértices.

Teorema 2.6. [Welsh - Powell, 1967] Si un grafo G tiene una secuencia de grados $gr(V_1) \geq ... \geq gr(V_n)$, entonces

$$\chi(G) \leq 1 + \max_{1 \leq i \leq n} \{\text{mín}(i - 1, gr(V_i))\}$$

Demostración. La siguiente demostración está basada en Cerón et al. (2005).

Dado un grafo G con n vértices que están ordenados en una secuencia decreciente según sus grados $gr(V_1) \geq ... \geq gr(V_n)$. Suponemos que debemos colorear el vértice V_k que pertenece a G; por tanto, tenemos los siguientes casos:

Caso 1: Si $gr(V_i) \geq i - 1$ para todo $1 \leq i \leq k$, entonces $\text{mín}(i - 1, gr(V_i)) = i - 1$. De allí tenemos que $\text{mín}(k - 1, gr(V_k)) = k - 1$, por lo tanto

$$\max_{1 \leq i \leq k} \{\text{mín}(i - 1, gr(V_i))\} = k - 1$$

porque estamos evaluando todos aquellos en los que los mínimos son dados por $i - 1$ para todo $1 \leq i \leq k$, entonces en últimas se estaría evaluando el

$$\max_{1 \leq i \leq k} \{i - 1\} = k - 1$$

Caso 2: Si $gr(V_i) \leq i-1$ para todo $k < i \leq n$, entonces mín$(i-1, gr(V_i)) = gr(V_i)$. De lo anterior tenemos que:

$$\max_{k<i\leq n} \{\text{mín}(i-1, gr(V_i))\} = gr(V_{k+1})$$

Dado que en la secuencia de grados $gr(V_k) \geq gr(V_{k+1}) \geq ... \geq gr(V_n)$, el grado máximo será $gr(V_{k+1})$ ya que es el grado mayor que le sigue a V_k.

Como consecuencia de los dos casos tenemos que:

$$\text{máx}(\text{mín}(i-1, gr(V_i))) = \text{máx}(k-1, gr(V_{k+1}))$$

Si

$$\text{máx}(k-1, gr(V_{k+1})) = k-1$$

por la Definición 1.50 todos los $k-1$ vértices anteriores a V_k pertenecen a la vecindad de V_k y ya han sido coloreados, por lo tanto, queda un color disponible para V_k, así necesitamos $(k-1)+1 = k$ colores para los vértices hasta V_k. Por último, $gr(V_i) \leq i-1$, entonces para pintar el vértice V_i cuando $i > k$ se utilizan a lo sumo k colores. Análogamente cuando

$$\text{máx}(k-1, gr(V_{k+1})) = gr(V_{k+1})$$

Es así que obtenemos:

$$\chi(G) \leq 1 + \max_{1\leq i\leq n} \{\text{mín}(i-1, gr(V_i)\}$$

□

A partir de los dos teoremas anteriores, podemos establecer una relación entre las cotas obtenidas para el número cromático. El Teorema de Welsh - Powell 2.6 mejora, o igual la cota dada en el Teorema 2.5, pero nunca es peor, pues mín$\{i-1, gr(V_i)\} \leq gr(V_i) \leq \Delta(G)$, y por consiguiente

$$1 + \{\text{mín}(i-1, gr(V_i))\} \leq \Delta(G) + 1$$

Ejemplo 2.2. Vamos a ilustrar los dos teoremas anteriores con el grafo N (Figura 2.3). Para el Teorema 2.6, primero organizamos los vértices según la secuencia de grados decreciente $(5, 4, 4, 4, 4, 3, 3, 2, 1, 0) = (D, A, B, H, C, I, F, E, G, J)$ y realizamos el procedimiento:

i	V_i	$gr(V_i)$	$i-1$	mín$(gr(V_i), i-1)$
1	D	5	0	0
2	A	4	1	1
3	B	4	2	2
4	H	4	3	3
5	C	4	4	4
6	I	3	5	3
7	F	3	6	3
8	E	2	7	2
9	G	1	8	1
10	J	0	9	0

Tabla 2.1: Proceso del Teorema de Welsh - Powell

Para obtener que

$$\chi(N) \leq 1 + \max_{1 \leq i \leq 10} \{0, 1, 2, 3, 4, 3, 3, 2, 1, 0\} = 1 + 4 = 5$$

Ahora, aplicando el Teorema 2.5 obtenemos que $\chi(N) \leq \Delta(N) + 1 = 5 + 1 = 6$.

En este caso, el Teorema de Welsh-Powell nos genera una mejor cota superior para el número cromático; sin embargo, para realizar el proceso del Teorema 2.6 a mano puede ser tedioso para grafos grandes.

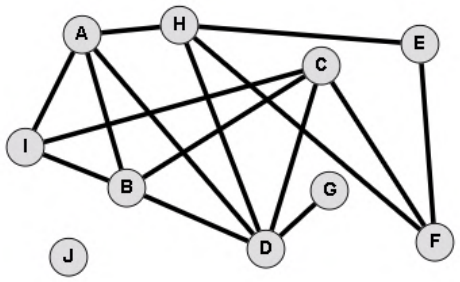


Figura 2.3: Grafo N

Otra cota superior para el número cromático de algunos grafos está dada por el Teorema de Brooks, el cual mejora la cota dada por el Teorema 2.5 para grafos conexos que no sean completos o que sean un ciclo impar. Antes de estudiar el Teorema de Brooks, vamos a introducir algunos resultados:

Lema 2.1. Sea un grafo G no completo, $k-$regular, con $k \geq 3$, $2-$conexo; entonces G tiene un vértice X con dos vecinos Y y Z tales que ellos dos no son adyacentes y el grafo $G - \{Y, Z\}$ es un grafo conexo.

Demostración. Sea G un grafo $2-$conexo.

Notemos que, aunque G es $2-$conexo, G no puede ser un ciclo puesto que los ciclos son $2-$regulares y el enunciado del lema exige que $k \geq 3$.

Seleccionamos un vértice A que pertenece al conjunto de vértices de G y consideramos el grafo $G - \{A\}$. Tenemos dos opciones, que $G - \{A\}$ sea $2-$conexo o no.

1. Si $G - \{A\}$ es $2-$conexo, como el grado de A es igual a k, pues G es $k-$regular, al eliminarlo del grafo G sus vecinos tienen grado $k - 1$. Luego seleccionamos en el grafo $G - \{A\}$ un vecino de A, que notamos M y como el grado de M es mayor o igual a 2, existe otro vértice en $G - \{A\}$, que notamos O, adyacente a M. Luego, tenemos un camino $h = (AM, MO)$ de longitud dos de A a O. Por lo tanto, el vértice M en el papel de X, A en el papel de Y y O en el de Z, cumplen la condición del lema, pues M tiene dos vecinos A y O que no son adyacentes entre si, además como $G - \{A\}$ es $2-$conexo al eliminar O del grafo $G - \{A\}$, el grafo resultante $G - \{A, O\}$ sigue siendo conexo.

2. Suponemos que el grafo $G - \{A\}$ no es $2-$conexo, entonces el grafo $G - \{A\}$ tiene un vértice de corte y por la Proposición 1.1, se pueden determinar dos bloques hojas B_1 y B_2 del grafo $G - \{A\}$.

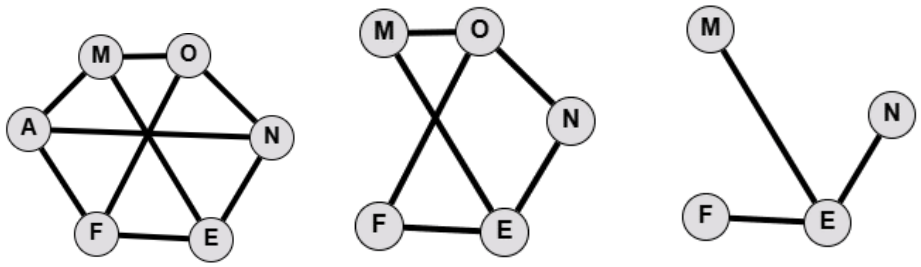
   Como G es $2-$conexo, esto es G no tiene vértices de corte entonces se pueden encontrar dos vértices, B_1 y B_2, adyacentes a A, pues si no existiera un vértice en B_1 adyacente a A, en el grafo G, cualquier camino que conecte un vértice de B_1 con un vértice fuera de B_1 no puede pasar por A, pero en $G - \{A\}$ la única forma de conectar cualquier vértice de B_1 con el resto del grafo es a través de su único vértice de corte C_1 (pues es un bloque hoja). Por tanto en el grafo G todo camino que conecta B_1 con el resto del grafo G pasa por dicho vértice C_1, lo que lo convierte en un vértice de corte de G, pero esto contradice que G es $2-$conexo. Un argumento análogo se aplica para el bloque B_2.

   Entonces M y O no son vértices de corte en $G - \{A\}$ pues cada bloque hoja tiene exactamente un vértice de corte, ademas cada bloque tiene al menos dos vértices, luego M y O se selccionan con esta condición.

Además, por el Colorario 1.3, M y O no son adyacentes entre si. Aunque G es 2−conexo el conjunto de vértices $\{M, O\}$ no es de corte, puesto que como M y O son adyacentes a A, entonces al eliminarlos al mismo tiempo del grafo G, este no se va a desconectar porque $gr(A) \geq 3$, es decir, al menos tendrá un vecino. Además como M y O no son vértices de corte en $G - \{A\}$ es conexo, por lo tanto $G - \{M, O\}$ es conexo, lo que cumple la condición del lema, teniendo a A como X, M como Y y O como Z.

□

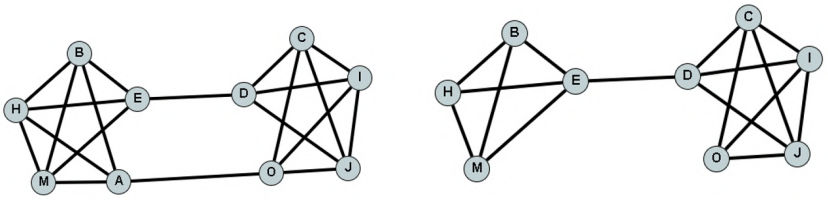
Ejemplo 2.3. Para ilustrar el primero caso del Lema 2.1 tomamos un grafo G (Figura 2.4a) donde $G - \{A\}$ sera 2−conexo (Figura 2.4b), como el grafo es 3−regular los vértices que eran adyacentes a A en el grafo $G - \{A\}$ tienen grado 2 y uno de los vecinos de A, en particular M, es adyacente a otro vértice O. Por lo tanto tenemos un camino $h = (AM, MO)$ de longitud 2. Entonces M tiene dos vecinos A y O que no son adyacentes entre si, además el grafo $G - \{A\}$ es 2−conexo y al eliminar O, el grafo resultante $G - \{A, O\}$ sigue siendo conexo (Figura 2.4c).



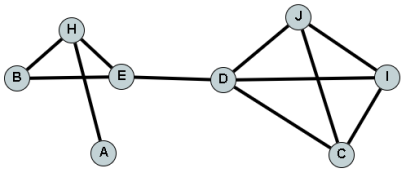
(a) Grafo G (b) Grafo $G - \{A\}$ (c) Grafo $G - \{A, O\}$

Figura 2.4: Primera parte de la demostración

Ilustramos el segundo caso con el grafo 2−conexo y 4−regular G en la Figura 2.5a, y el grafo $G-\{A\}$ que no es 2−conexo en la Figura 2.5b, de aquel grafo se reconocen dos bloques $B_1 = \{B, M, H, E\}$ y $B_2 = \{C, I, J, O, D\}$, en los que $M \in B_1$ y $O \in B_2$ no son adyacentes entre si ni son vértices de corte, entonces el grafo $G - \{M, O\}$ es conexo (Figura 2.5c).



(a) Grafo G (b) Grafo $G - \{A\}$



(c) Grafo $G - \{M, O\}$

Figura 2.5: Caso en el que $G - \{A\}$ no es 2-conexo

Teorema 2.7. (Teorema de Brooks) Sea G un grafo no completo, simple y conexo con grado máximo $\Delta(G) \geq 3$, entonces el $\chi(G) \leq \Delta(G)$.

Demostración. Esta demostración la elaboramos a partir de tres casos:

Caso 1. Dado un grafo G no regular con n vértices, elegimos un vértice V_n cuyo grado sea menor al máximo del grafo, $gr(V_n) < \Delta(G)$. A partir de V_n construimos un árbol, de manera que V_n sea el vértice de inicio y luego sus vecinos directos se añaden al árbol como los primeros vértices, asegurándose que entre ellos no creen ciclos. Seguido se incorporan los vértices de G que sean adyacentes a los ya agregados, asegurando que cada nuevo vértice esté vinculado únicamente a vértices previamente integrados. Repetimos el proceso sucesivamente hasta que todos sus vértices estén conectados al menos a un vértice del grafo.

Luego se organizan los vértices $V_1, V_2, ..., V_n$ de manera que V_n sea la raíz del árbol y para todo $i < n$, el vértice V_i está mas lejos de la raíz de V_j si $i < j$. Es decir, los indices decrecen a lo largo de cualquier camino que parte de V_n. Luego, definimos una función de coloración para G utilizando dicho orden de vértices. Analizando los vecinos ya coloreados que puede tener cada vértice, tenemos:

Si $i < n$: como $gr(V_i) \leq \Delta(G)$ y en el árbol, V_i tiene exactamente un vecino con indice mayor, entonces $gr(V_i) - 1 \leq \Delta(G) - 1$, es decir a lo sumo hay $\Delta(G) - 1$ vecinos con indices menores que i ya coloreados. Para V_n, como todos sus vecinos tienen indices menores y por hipótesis $gr(V_n) < \Delta(G)$, entonces a lo sumo hay $\Delta(G) - 1$ vecinos ya coloreados. Por tanto para colorear cualquier vértice como máximo $\Delta(G) - 1$ colores están prohibidos, luego se requiere uno mas.

Caso 2. Sea G un grafo $\Delta(G)-$regular, con un vértice de corte X. Si se elimina X de G, el grafo $G - \{X\}$ tiene más componentes conexas que G. Luego creamos los subgrafos inducidos por los conjuntos de vértices de estas componentes junto con el vértice X. Los subgrafos creados son no regulares puesto que a X le hace falta algunas aristas que estaban presentes en G, por lo tanto el $gr(X)$ en cada subgrafo inducido es menor que $\Delta(G)$. Siendo así, haciendo uso del caso 1, tenemos que cada subgrafo tiene a lo sumo una $\Delta(G)-$coloración.

Como los subgrafos inducidos solo tiene el vértice X en común y la unión de todos ellos es el grafo G, permutamos la organización de los vértices para que a X le corresponda el color 1 en todos los subgrafos existentes, para que al volver al grafo original la coloración sea válida.

Caso 3. Sea G un grafo $\Delta(G)-$regular y $2-$conexo. Por el Lema 2.1 se puede encontrar un vértice V_x tal que dos de sus vecinos, V_m y V_n, no son adyacentes entre sí y el grafo $G - \{V_m, V_n\}$ es conexo. Como los dos vértices no son adyacentes, se les asigna el mismo color 1. Y a partir del vértice V_x se crea un árbol como en el caso 1 con los vértices de $G - \{V_m, V_n\}$. Además, por medio de la función de coloración 1.50 obtenemos a lo sumo $\Delta(G)$ colores a los vértices del grafo, con $i < x$. El vértice V_x tiene $\Delta(G)$ vecinos pero dos de ellos tienen el mismo color, por lo tanto, a lo sumo el número de colores prohibidos para V_x es a lo sumo $\Delta(G) - 1$, luego de los $\Delta(G)$ colores siempre hay uno disponible para V_x.

□

Ejemplo 2.4. El grafo R es simple, conexo y no es completo, donde $\Delta(R) = 4$, entonces aplicando el Teorema de Brooks $\chi(R) \leq 4$.

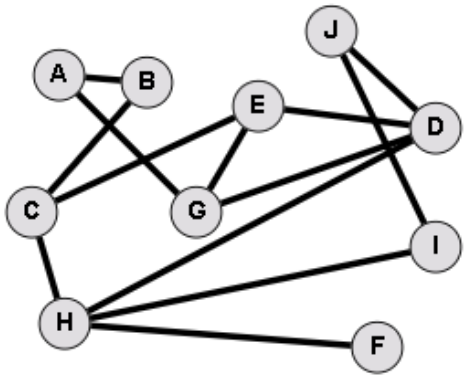


Figura 2.6: Grafo R

2.3. Número cromático asociado a operaciones entre grafos

Teorema 2.8. El grafo suma $G+H$ de los grafos G y H, tiene como número cromático $\chi(G+H) = \chi(G) + \chi(H)$

Demostración. Como el resultado del teorema es una igualdad, vamos a demostrar dos desigualdades.

1. En el grafo $G+H$ ningún color del subgrafo G puede ser usado en el subgrafo H, porque cada vértice G es adyacente a cada vértice de H. Entonces $\chi(G+H) \geq \chi(G) + \chi(H)$.

2. Dado que $\chi(G)$ es el mínimo de colores que se utiliza para colorear G y $\chi(H)$ es el mínimo de colores que se pueden utilizar para colorear H. Además, como todos los vértices H son adyacentes a todos los vértices G en $G+H$ entonces $G+H$ admite una coloración con $\chi(G)+\chi(H)$ colores. Por lo tanto $\chi(G+H) \leq \chi(G) + \chi(H)$.

3. Como demostramos que $\chi(G+H) \geq \chi(G)+\chi(H)$ y a su vez $\chi(G+H) \leq \chi(G) + \chi(H)$, concluimos que $\chi(G+H) = \chi(G) + \chi(H)$.

□

Ejemplo 2.5. El grafo suma $W + H$ (Figura 2.7) tiene un subgrafo inducido W con $V_W = \{A, B, C, D, F\}$ y un subgrafo inducido H con $V_H =$

$\{M, J, K, L\}$, en los que $\chi(W) = 3$ $\chi(H) = 2$, entonces por Teorema 2.8 $\chi(W + H) = 5$.

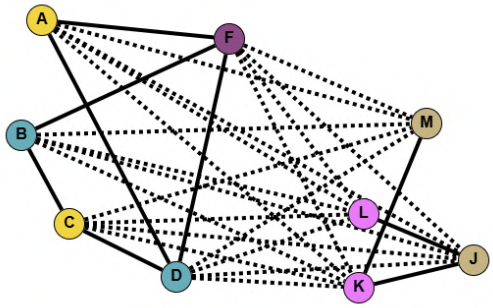


Figura 2.7: Grafo $W + H$

Teorema 2.9. El número cromático del producto cartesiano $G \times H$ de dos grafos G y H es el máximo del conjunto de los números cromáticos de dichos grafos, es decir $\chi(G \times H) = \text{máx}\{\chi(G), \chi(H)\}$

Demostración. La prueba la vamos a desarrollar por medio de desigualdades para concluir la igualdad.

1. Vamos demostrar que $\chi(G \times H) \geq \text{máx}\{\chi(G), \chi(H)\}$. En el grafo $G \times H$ se puede encontrar a los grafos G y H como subgrafos. Por el Teorema 2.1 tenemos que $\chi(G \times H) \geq \chi(G)$ y $\chi(G \times H) \geq \chi(H)$. Luego, podemos concluir que $\chi(G \times H) \geq \text{máx}\{\chi(G), \chi(H)\}$.

2. Demostraremos que $\chi(G \times H) \leq \text{máx}\{\chi(G), \chi(H)\}$. Para ello, asignamos $k = \text{máx}\{\chi(G), \chi(H)\}$. Vamos a definir una $k-$coloración para $G \times H$ a partir de la $\chi(G)-$coloración para G dada por g y la $\chi(H)-$coloración para H dada por h:

   a) Para el grafo G la función $g : V_G \to \{1, 2, ..., \chi(G)\}$

   b) Para el grafo H la función $h : V_H \to \{1, 2, ..., \chi(H)\}$

   c) Para el grafo $G \times H$ la función $f : V_{G \times H} \to \{1, 2, ..., k\}$. Donde el color del vértice (V_i, V_j) del grafo $G \times H$ será asignado así $f(V_i, V_j) = g(V_i) + h(V_j) \mod k$.
   La función $f(V_i, V_j) = g(V_i) + h(V_j) \mod k$ sirve para generar una $k-$coloración para los vértices del grafo producto cartesiano,

puesto que: Si (V_i, V_j) y (V_l, V_m) son dos vértices adyacentes en $G \times H$, por definición de producto cartesiano, significa que:

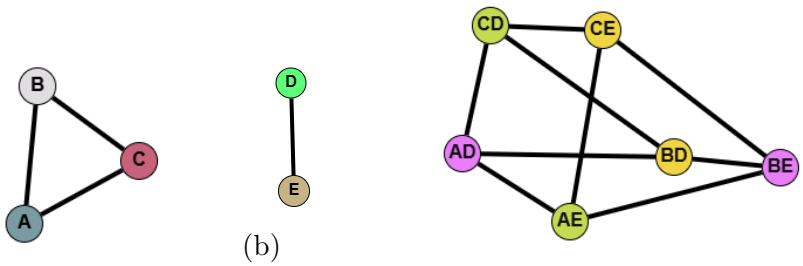
1) $V_i = V_l$ y V_j adyacente con V_m en el grafo H.
2) o $V_j = V_m$ y V_i adyacente con V_l en el grafo G.

Ahora en la función f para cada vértice se tendría: $f(V_i, V_j) = g(V_i) + h(V_j) \mod k$ y $f(V_l, V_m) = g(V_l) + h(V_m) \mod k$. Si $V_i = V_l$, el color asignado en G es el mismo, es decir, $g(V_i) = g(V_l)$. Por lo que, los colores $f(V_i, V_j)$ y $f(V_l, V_m)$ son distintos por el sumando determinada por la función h. De forma análoga se realiza el análisis para el caso 2. Debido a esa diferencia y al módulo k, tenemos que $f(V_i, V_j) \neq f(V_l, V_m)$ entonces los vértices (V_i, V_j) y (V_l, V_m) caen en distintas clases de color. Con lo que, la función definida funciona para obtener una $k-$ coloración del grafo $G \times H$.

Con ello, $\chi(G \times H) \leq k = \max\{\chi(G), \chi(H)\}$.

Demostradas ambas partes, concluimos que $\chi(G \times H) = \max\{\chi(G), \chi(H)\}$. □

Ejemplo 2.6. Los grafos H (Figura 2.8a) y O (Figura 2.8b) tienen que $\chi(G) = 3$ y $\chi(O) = 2$, aplicando el Teorema 2.9 resulta que $\chi(G \times H) = 3$ (Figura 2.8c).



(a) Grafo H (b) Grafo O (c) Grafo $H \times O$

Figura 2.8: Número cromático del grafo producto cartesiano

2.4. Número cromático en algunas familias de grafos

En esta sección presentamos algunos teoremas que nos permiten establecer el número cromático asociado a algunas familias de grafos.

Teorema 2.10. Un grafo G tiene $\chi(G) = 1$ si y solo si G no tiene aristas.

Demostración. Vamos a probar ambas implicaciones:

1. Si G no tiene aristas, significa que ninguno de los vértices del grafo son mutuamente adyacentes, por lo tanto, todos los vértices pueden tener el mismo color, es decir $\chi(G) = 1$.

2. Si $\chi(G) = 1$ significa que todos los vértices no son mutuamente adyacentes, entonces G no tiene aristas.

□

Teorema 2.11. Sea un grafo completo con n vértices entonces $\chi(K_n) = n$.

Demostración. Por la definición de grafo completo, todos los vértices son mutuamente adyacentes; entonces todos los vértices deben tener un color distinto es así como necesitamos n colores. □

Teorema 2.12. Un grafo bipartito tiene $\chi(B_{mn}) = 2$.

Demostración. Por la definición de grafo bipartito, el conjunto de vértices de $V_{B_{m,n}}$ se puede dividir en dos subconjuntos U y W de manera que los subconjuntos son independientes, es decir, ninguno de los vértices de U son mutuamente adyacentes, lo mismo para W. Entonces, a cada subconjunto le podemos asignar un color distinto, por lo tanto, solo es necesario de dos colores para colorear el grafo. □

A partir del Teorema 1.3 y del número cromático de los grafos bipartitos, podemos encontrar y demostrar el número cromático de otras familias de grafos.

Teorema 2.13. El grafo lineal P_n, para $n \geq 2$, tiene $\chi(P_n) = 2$.

Demostración. Para $n < 2$ se tiene $n = 1$, por lo que P_n es trivial y el número cromático es 1.

Ahora, para $n \geq 2$ y por la forma en la que se definió el grafo lineal, este no puede ser un ciclo, por lo tanto tampoco es un ciclo impar. Entonces, por el Teorema 1.3, P_n es un grafo bipartito y por el Teorema 2.12, el número cromático del grafo linea es 2. □

Teorema 2.14. Un grafo árbol tiene $\chi(T) = 2$.

Demostración. Por la definición de grafo árbol decimos que es conexo y no tiene ciclos, es decir no tiene ciclos impares y por el Teorema 1.3 el grafo árbol es bipartito. Además por el Teorema 2.12 el grafo árbol tiene número cromático 2. □

Teorema 2.15. Un grafo ciclo par tiene $\chi(C_{2n}) = 2$.

Demostración. El grafo es un ciclo que impar, entonces por el Teorema 1.3, el ciclo par es un grafo bipartito, por lo que aplica el Teorema 2.12 y el número cromático del ciclo par es igual a 2. □

Teorema 2.16. Un grafo ciclo impar tiene $\chi(C_{2n+1}) = 3$.

Demostración. El conjunto de vértices del ciclo impar se puede dividir en tres subconjuntos independientes, dos de n vértices y otro de un vértice. A cada subconjunto le podemos asignar un color distinto, siendo el número cromático 3. □

Teorema 2.17. Un grafo rueda par tiene $\chi(W_{2n}) = 4$.

Demostración. Los grafos rueda par son la suma de un grafo ciclo impar y un vértice que se puede considerar como el grafo completo K_1, haciendo uso del Teorema 2.8 tenemos

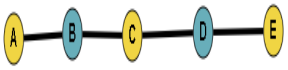
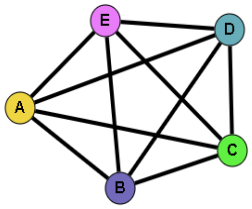
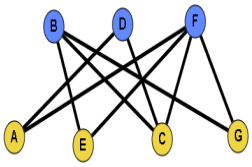
$$\chi(W_{2n}) = \chi(C_{2n+1} + K_1) = \chi(C_{2n+1}) + \chi(K_1) = 3 + 1 = 4$$

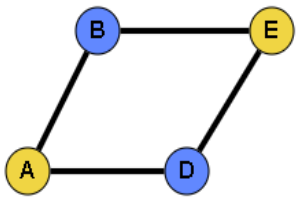
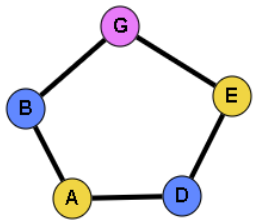
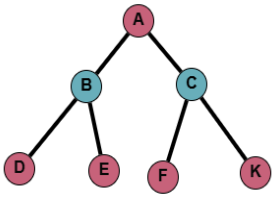
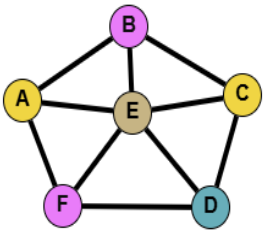
□

Teorema 2.18. El grafo rueda impar tiene $\chi(W_{2n+1}) = 3$.

Demostración. El grafo rueda impar lo consideramos como la unión de un ciclo par y un vértice, por lo que usamos el Teorema 2.8 para determinar: $\chi(W_{2n+1}) = \chi(C_{2n} + K_1) = \chi(C_{2n}) + \chi(K_1) = 2 + 1 = 3$. □

En la Tabla 2.2 presentamos un resumen de los números cromáticos de las familias de grafos mencionadas.

Familia de grafos	**Número cromático**	**Diagrama**	**Referencia**
Trivial	$\chi(G) = 1$		Teorema 2.10
Lineal	$\chi(G) = 2$		Teorema 2.13
Completo	$\chi(K_n) = n$		Teorema 2.11
Bipartito	$\chi(B_{m,n}) = 2$		Teorema 2.12

Familia de grafos	**Número cromático**	**Diagrama**	**Referencia**
Ciclo	Par: $\chi(C_{2n}) = 2$		Teorema 2.15
	Impar: $\chi(C_{2n+1}) = 3$		Teorema 2.16
Árbol no trivial	$\chi(G) = 2$		Teorema 2.14
Rueda	Par: $\chi(W_{2n}) = 4$		Teorema 2.17

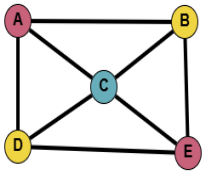
Familia de grafos	**Número cromático**	**Diagrama**	**Referencia**
	Impar: $\chi(W_{2n+1}) = 3$		Teorema 2.18

Tabla 2.2: Número cromático de algunas familias de grafos

2.5. Coloración de mapas

El origen de la coloración de grafos está estrechamente relacionado con problema de los cuatro colores. De acuerdo con Ramírez (2001), este problema comenzó en 1852, por Francis Guthrie, quien observó que con cuatro colores se podían colorear los condados de Inglaterra, de manera que dos regiones vecinas no pueden compartir el mismo color; hallazgo que se extendió a otros mapas. Luego, Francis le comunicó la conjetura a su hermano Frederick, quien se lo compartió a su profesor, Augustus de Morgan, que no logró dar una respuesta, pero a través de su carta a Hamilton logró divulgar el problema.

El problema de los cuatro colores puede ser modelado como un problema de grafos, de manera que un mapa corresponde a un grafo donde los vértices representan las regiones del país y las aristas muestran la conexión entre dos regiones que son fronterizas. Dicho grafo resulta ser plano, es decir, aquellos que se pueden dibujar en el plano donde ninguna de sus aristas se cruza entre sí (Gross et al., 2019, p. 298).

Durante varios años, diversos investigadores intentaron dar solución a este problema sin éxito; sin embargo, muchos de ellos aportaron resultados parciales que ayudaron a enriquecer la teoría. Este es el caso de Alfred Bray Kempe, quien se interesó en el tema y en 1879 dio a conocer la primera

prueba, publicada en el American Journal of Mathematics. En aquella demostración, Kempe propuso un método con regiones de colores alternados llamado cadenas de Kempe, que fue aceptada durante 11 años, hasta que en 1890 el matemático Percy John Heawood encontró una falla en el argumento, aportando un contraejemplo basado en el caso de regiones rodeadas por un anillo de cinco regiones.

Aunque la prueba de Kempe falló, este método le resultó útil a Heawood para demostrar el Teorema de los cinco colores, y su técnica fue fundamental para el desarrollo de las siguientes pruebas del problema de los cuatro colores.

Con el tiempo, varios matemáticos siguieron desarrollando avances para la solución al problema. Entre ellos, George David Birkhoff propusó el concepto de configuraciones reducibles, que consisten en conjuntos de regiones que no son contraejemplo mínimo al Teorema de los cuatro colores. Si se encontraba un conjunto de configuraciones reducibles que inevitablemente están presentes en cualquier mapa, el teorema quedaría demostrado. Con esta idea, matemáticos como Philip Franklin encontraron que los mapas con 25 regiones o menos, se podían colorear con cuatro colores. Luego, Oystein Ore y Joel Stemple en 1970 lo ampliaron a 39 regiones. Y Jean Mayer a 95 regiones en 1976 (Chartrand & Zhang, 2012, p. 249).

Tiempo después, Heinrich Heesch implementa un método sistemático para buscar configuraciones reducibles inevitables, quien a su vez fue el primero en usar la computadora para verificar la reducibilidad en las configuraciones. A este método se sumaron varios matemáticos y programadores, entre ellos Wolfgang Haken en asociación con Kenneth Appel y el estudiante de informática John Koch, crearon un algoritmo más eficiente, en términos de tiempo computacional, para verificar la reducibilidad de las configuraciones. En 1976, finalmente, Apple y Haken demostraron el teorema de los cuatro colores, al lograr encontrar un conjunto reducible inevitable con 1936 configuraciones; después de 1200 horas de ejecución en tres ordenadores.

La prueba de Appel y Haken fue y ha sido controversial, ya que dependía de un ordenador y generó dudas acerca de la validez de una demostración hecha con una computadora: “De hecho, muchos se mostraron muy escépticos e incómodos con ello. Esto inició numerosas discusiones sobre lo que es una demostración matemática. Un paso importante en la aceptación de su demostración; sin embargo, fue su aceptación por parte del distinguido matemático William Tutte”(Chartrand & Zhang, 2012, p. 250).

Como la coloración de mapas se puede estudiar asociado a la coloración de vértices de un grafo; en esta sección abordaremos esta idea.

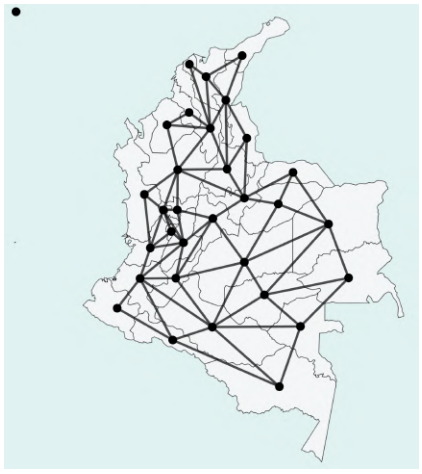
Definición 2.1. Un grafo es **plano** si su representación gráfica, en un plano, no tiene aristas que se intersecan.

Para colorear un mapa se debe tener en cuenta que si dos regiones son fronterizas sus colores deben ser diferentes. Entendiendo fronterizas por una curva, pues si se intersecan en un punto las dos regiones pueden tener un mismo color. Este problema lo trasladamos a la coloración de vértices de un grafo de manera que:

1. Cada región del mapa se representa con un vértice.
2. Si dos regiones son fronterizas los vértices que las representan se conectan por medio de una arista.

Ejemplo 2.7. En el mapa de los departamentos de Colombia (Figura 2.9a), cada departamento se representa con un vértice, y los vértices representan los departamentos que comparten una frontera curva son los extremos de una arista. (Figura 2.9b). Es así como el vértice que representa Casanare es adyacente a los vértices que representan Vichada, Meta, Boyacá y Arauca; pero no es adyacente al vértice que representa Cundinamarca, porque sus límites son un punto, por lo cual se descarta la existencia de una arista entre estos dos departamentos.

(a) Mapa de Colombia



(b) Grafo asociado al mapa de Colombia

De este modo, la coloración de mapas puede interpretarse como la coloración de vértices de un grafo asociado al mapa, el cual es un grafo plano.

En este contexto, un mapa puede tener una $k-$coloración por lo tanto es $k-$coloreable, entonces su número cromático también corresponde al número cromático de su grafo representativo.

A lo largo de la historia, se llegó a la idea que que cualquier mapa se puede colorear con mínimo 4 colores , bajo las condiciones mencionadas, es decir, es $4-$coloreable. Este enunciado fue uno los grandes problemas de la época (mitad del siglo XIX), que es referido como el teorema de los cuatro colores.

Teorema 2.19. Cualquier mapa plano puede ser coloreado usando como máximo 4 colores, tal que que ninguna región adyacente comparta el mismo color.

La demostración del teorema de los cuatro colores fue realizada por Kenneth Apple y Wolfgang Haken en el año 1976, la cual es extensa y compleja, por lo que vamos a describir la idea principal.

Los autores utilizan conceptos de los grafos planos, como las triangulaciones (divisiones del plano en triángulos sin superposición), y definen el concepto de configuración como: un subgrafo formado por un anillo (circuito cerrado) con su interior. Una configuración se considera reducible si no puede formar parte de un contraejemplo al teorema, es decir, si no puede aparecer en un grafo plano que requiera más de cuatro colores (West, 2005, p. 258).

El objetivo es construir un conjunto de configuraciones inevitable, es decir, un conjunto tal que toda triangulación del plano contenga al menos una de ellas. Si todas las configuraciones de ese conjunto también son reducibles, entonces no puede existir un contraejemplo al teorema, y por lo tanto, todo grafo plano puede colorearse con cuatro colores.

Por otro lado se utilizó el método de descarga donde a los vértices de cada triangulación se les asigna una carga positiva o negativa según su grado y luego redistribuir esas cargas respetando la conservación de carga basada en la carga total que brinda el uso de la fórmula de Euler. La idea es que esta redistribución permite identificar patrones o "vecindades"típicas que necesariamente deben aparecer en cualquier triangulación, y que pueden analizarse como configuraciones inevitables (West, 2005, p. 260).

Finalmente, la parte más controversial de esta prueba, fue el uso intensivo del ordenador para verificar la reducibilidad de todas las configuraciones del conjunto inevitable.

Con este enfoque, Appel y Haken lograron demostrar que toda triangulación contiene al menos una configuración reducible, lo que implica que todo

grafo plano es coloreable con, a lo más, cuatro colores. [2]

En 1879 Kempe presentó una demostración del teorema de los cuatro colores a partir de la estrategia conocida como las cadenas de Kempe; sin embargo, en 1890 Heawood encontró un error en la prueba, dando un contra-ejemplo a la prueba de Kempe. Aunque el método presentado por Kempe no tuvo éxito, Heawood pudo utilizar dicha técnica (cadenas de Kempe) para demostrar que todo mapa puede colorearse con cinco o menos colores.

Teorema 2.20. Dado un grafo plano simple G su número cromático es a lo sumo 5,es decir, G es $5-$coloreable.

La demostración del teorema se basa en considerar un grafo plano G y un vértice V cuyos vecinos están coloreados con cinco colores distintos. Como no es posible asignar a V uno de esos colores, entonces se analizan caminos en el grafo $G - \{V\}$ que están formados por vértices coloreados alternadamente con dos colores, conocidos como cadenas de Kempe; en particular, caminos entre dos vecinos de V.

Si no existe un camino entre dos vecinos de V usando dos colores, podemos intercambiar los colores de una componente del grafo, para que los dos vecinos tengan el mismo color, y liberar el otro color para el vértice V. En el caso que exista un camino entre dos vecinos de V, por ser G plano, no puede haber otro camino de colores alternados entre otro par de vecinos de V; lo que garantiza que podamos hacer el intercambio de colores para liberar uno para V. Al extender este proceso a todo el grafo G obtenemos una $5-$coloración. Para ver detalladamente la demostración del teorema de los cinco colores, recomendamos revisar el libro de Chartrand y Zhang (2012).

[2]Para ver el anuncio del teorema de los cuatro colores ver el artículo de Apple y Haken, 1976

Capítulo 3

Algoritmos Heurísticos

Calcular el número cromático de un grafo arbitrario constituye un problema complejo; si bien tenemos resultados teóricos para algunas familias de grafos, no disponemos de soluciones para grafos de manera general. Por eso, varios autores como Welsh y Powell (1979), Leighton (1979) y Matula et al. (1972), nos plantean algunos algoritmos heurísticos, que utilizan reglas o estrategias basadas en la experiencia para obtener soluciones rápidas, aunque no nos garantizan obtener el número cromático exacto del grafo.

En este capítulo vamos a presentar cuatro algoritmos heurísticos para encontrar una aproximación al número cromático de grafos. Para cada uno incluimos una descripción, ejemplos de su ejecución, el pseudogódigo correspondiente y el código de programación para Python que es generado por la inteligencia artificial ChatGPT a partir de los datos que le brindamos. Finalmente, vamos a presentar una prueba computacional que realizamos a los cuatro algoritmos, para caracterizarlos de acuerdo con tres criterios.

Antes de presentar los cuatro algoritmos, queremos aclarar que los métodos para colorear vértices de grafos los consideramos algoritmos. Bajo una definición más amplia de algoritmo, "dado un problema y un dispositivo donde resolverlo, es necesario proporcionar un método preciso que lo resuelva, adecuado al dispositivo" (Guerequeta & Vallecillo, 2000, p. 13). A partir de esta definición, nosotras consideramos como problema la coloración de vértices en grafos apuntando al número cromático, y como dispositivo un computador mediante la implementación en Python.

3.1. Algoritmo Greedy

La Definición de función de coloración 1.43 la podemos ver como un algoritmo heurístico para la coloración de vértices en grafos, el cual recibe el nombre de **Greedy, Voráz o Secuencial**. Dado un grafo G con n vértices, una lista de colores $C = (1, 2, ..., n)$ y un orden fijo para los vértices $V = (V_1, V_2, ..., V_n)$, el algoritmo consiste en recorrer la lista de vértices ordenados para asignarle a cada uno el menor color disponible que no haya sido usado por sus vecinos ya coloreados.

En el Algoritmo 1 ilustramos mediante pseudocódigo el funcionamiento del algoritmo Greedy.

Algoritmo 1 Greedy, Voráz o Secuencial

Entrada: : Grafo G con n vértices y una lista de los vértices con un orden fijo $V = (V_1, V_2, ..., V_n)$
Colores := (1, . . . , n)
Asignados := {V_1:1} Diccionario de vértices con colores asignados
Para $i \leftarrow 2$ **hasta** n **hacer**:
Vecinos := (Vecinos de V_i)
Para Vecinos$_j \leftarrow 1$ **hasta** Cantidad de vecinos **hacer**
Si Vecino $\in$ Asignados **entonces**
Asignado_vecinos := [Colores asignados a los vecinos de V_i]
end Si
Disponibles := Colores - Asignado_vecinos
end Para
Asignados[V_i] := min(Disponibles)
end Para
Salida: Diccionario Asignados.

Ejemplo 3.1. Vamos a ilustrar la ejecución del algoritmo Greedy con el grafo O de 9 vértices, que representamos en la Figura 3.1a.

1. Seleccionamos la siguiente ordenación de vértices $(C, L, I, A, H, J, B, K, M)$
2. Creamos la lista de colores $C = (1, 2, 3, 4, 5, 6, 7, 8, 9)$
3. Tomamos el primer vértice de nuestra lista ordenada, C y le asignamos el color 1.

4. Luego, escogemos el segundo vértice, L, miramos si tiene como vecino a C, como no es así, le asignamos el menor color disponible de la lista de colores, que es 1.

5. Elegimos el vértice I, miramos si I es vecino de C, como no son adyacentes, pasamos a revisar si I y L son vecinos, como son adyacentes descartamos el color de L en la lista de colores. Por lo que, el menor color disponible es 2, que es el que le asignamos.

6. Para el vértice A, los vecinos ya coloreados son C, I, entonces los colores disponibles son $(3, 4, 5, 6, 7, 8)$ y el color que le asignamos a A es 3.

7. En el caso del vértice H, sus vecinos ya coloreados son C, I, A, así que los colores disponibles son $(4, 5, 6, 7, 8)$ y el color de H es 4.

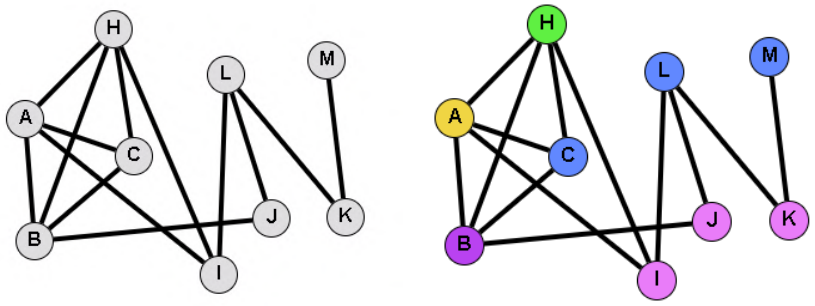
8. El vértice J tiene como vecino ya coloreado a L, con lo cual los colores disponibles son $(2, 3, 4, 5, 6, 7, 8)$, y el color de J es 2.

9. Para el vértice B, sus vecinos ya coloreados son C, A, H, J, luego los colores disponibles son $(5, 6, 7, 8)$, y el color de B es 5.

10. El vértice K tiene como vecino vecino ya coloreado a L, de este modo los colores disponibles son $(2, 3, 4, 5, 6, 7, 8)$, y el color de L es 2.

11. Finalmente, el vértice M tiene su vecino K con color 2, entonces los colores disponibles son $(1, 3, 4, 5, 6, 7, 8)$, y el color de M es 1.

En la Figura 3.1b representamos la coloración obtenida por el algoritmo, donde el color 1 se asocia con azul, 2 con rosado, 3 con amarillo, 4 con verde y 5 con purpura.



(a) Grafo N (b) Grafo N coloreado

Figura 3.1: Ejemplo algoritmo Greedy

A partir de la descripción del algoritmo Greedy y el pseudocodigo le pedimos a la Inteligencia Artificial ChatGPT que nos creara un código en Pyhton que después revisamos y ajustamos obteniendo el código 3.1.

```
def greedy_coloring(G, vertices):
    colores = {}
    for v in vertices:
        # Colores disponibles inicialmente
        disponibles = set(range(1, len(G.nodes())+1))
        # Remover colores de vecinos ya coloreados
        for vecino in G.neighbors(v):
            if vecino in colores:
                disponibles.discard(colores[vecino])
        # Asignar el menor color disponible
        colores[v] = min(disponibles)
    return colores
```

Código 3.1: Algoritmo Greedy en Python

Al ingresar un grafo de 14 vértices y 41 aristas en el Código 3.1, con un orden de vértices aleatorio, obtenemos la cantidad de colores, además del diccionario que contiene el vértice y su color correspondiente, como se ilustra en la Figura 3.2.

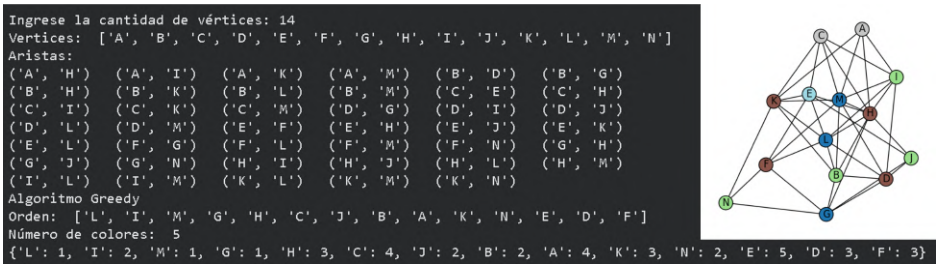
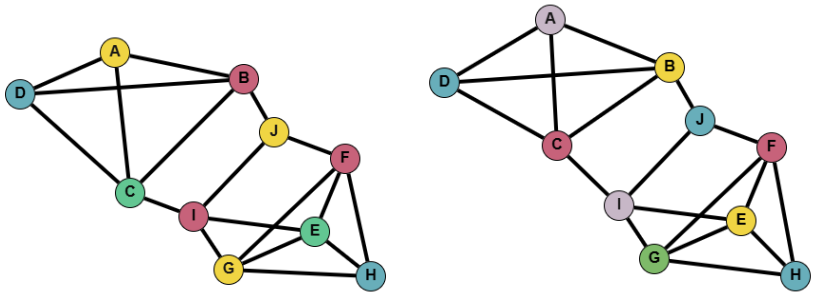


Figura 3.2: Resultado del algoritmo Greedy en Python

La coloración obtenida por el algoritmo Greedy puede variar dependiendo del orden que le demos a los vértices, además no nos garantiza el uso de la menor cantidad de colores, es decir, la obtención del número cromático. Por ejemplo, dado el grafo R (Figura 3.3) con 10 vértices y 18 aristas, al ejecutar el algoritmo en Python con el orden de vértices $(A, B, C, D, G, F, E, H, I, J)$, obtuvimos una $4-$coloración (Figura 3.3a); pero al ingresar los vértices en el orden $(D, H, C, F, J, E, B, I, G, A)$ obtuvimos una $5-$coloración (Figura 3.3b).



(a) Grafo R 4-coloreable (b) Grafo R 5-coloreable

Figura 3.3: Algoritmo Greedy en el grafo R

Con el algoritmo Greedy podemos obtener el número cromático, sin embargo esto implica tener una ordenación de vértices que nos permita conseguir una coloración ideal. Para ello, podemos considerar cada una de las ordenaciones posibles de $V(G)$ y aplicar una y otra vez el algoritmo hasta encontrar la indicada.

Pero si el grafo G tiene n vértices, tenemos $n!$ posibles ordenaciones, un calculo que aumenta el tiempo de forma excesiva. Por ejemplo, para un grafo

con 16 vértices tenemos que examinar 16! ordenaciones posibles. Si asumimos que podemos generar estas ordenaciones a un ritmo de 1 millón por segundo y que 1 día es igual 86400 segundos, tardaríamos, aproximadamente, 242 días para elaborarlas todas. Para un grafo con 20 vértices, debemos examinar 20! ordenaciones, y tardaríamos aproximadamente 77144 años para generarlos todos. Por esta razón, es necesario encontrar alternativas que nos permitan encontrar de manera más eficiente alguna coloración de un grafo, aunque no se garantice que se corresponda o se acerque más al número cromático.

3.2. Algoritmo Largest - First (LF)

El algoritmo Largest - First (LF) es una variante del Greedy propuesta por Welsh. Consiste en establecer una ordenación particular de los vértices en la cual aquellos más restrictivos se consideran primero. Esta ordenación se define atendiendo únicamente al grado de cada vértice de manera estática, es decir, considerando solamente sus conexiones en el grafo original. A partir de dicha ordenación, se establecen los conjuntos independientes y a los vértices de cada conjunto se le asigna un mismo color, de manera que cada color corresponde a un conjunto independiente.

Con base en lo anterior, describimos el proceso de aplicación del algoritmo LF. Sea G un grafo con su conjunto de vértices V_G; ordenamos sus vértices de manera decreciente según su grado, es decir, $gr(V_1) \geq gr(V_2) \geq gr(V_3) \geq ... \geq gr(V_n)$.

Posteriormente, construimos subconjuntos de vértices T_i en los que ningún elemento es adyacente a otro. Ubicamos el vértice de mayor grado, V_1 en el subconjunto T_1. Luego, consideramos el siguiente vértice en el orden establecido, V_2; si no es adyacente a ninguno de los elementos que se encuentran en T_1, entonces lo incorporamos a dicho subconjunto; en caso contrario, lo asignamos a un nuevo subconjunto T_2.

Continuamos este procedimiento siguiendo el orden definido, garantizando que dentro de cada T_i no existan adyacencias entre sus elementos. Finalmente, asignamos un color a cada subconjunto obtenido, con lo cual determinamos una coloración del grafo.

Siguiendo la idea del Algoritmo Largest - First proponemos el Pseudocodigo, en el Algoritmo 2, que muestra el paso a paso del algoritmo.

Algoritmo 2 Algoritmo Largest First

Entrada: : Grafo G y sus vértices $V = \{V_1, V_2, ..., V_n\}$
  Orden := (Decreciente de grados)
  Colores:=(1,...,n)
  Mientras Orden diferente de vacío **hacer**
    Para $k \leftarrow 1$ **hasta** n **hacer**
      $T_k := []$
      Para $i \leftarrow 1$ en orden **hasta** n **hacer**:
        Si V_i no es vecino de V_j en T_k **entonces**
          $V_i \in T_k$
        end Si
      end Para
      T_k Se le asigna el menor color de colores
      Orden := Orden-T_k
      Colores := Colores-Color de T_k
    end Para
  end Mientras
Salida: Subconjuntos T_k con los colores asignados.

Ejemplo 3.2. Para aplicar el algoritmo Largest - First vamos a ilustrar la ejecución con el grafo N de 8 vértices,que representamos en la Figura 3.4:

1. Ordenamos los vértices en forma decreciente según su grado, esto es $gr(D) \geq gr(C) \geq gr(B) \geq gr(A) \geq gr(F) \geq gr(H) \geq gr(E) \geq gr(G)$.

2. Ubicamos el vértice D en el subconjunto T_1. Luego consideramos el vértice C; como es adyacente a D, no puede pertenecer a T_1. Continuamos con B, que no es adyacente con D, entonces pertenece a T_1. Repitiendo este procedimiento con los demás vértices, obtenemos $T_1 = \{D, B, G\}$.

3. Con los vértices restantes $\{C, A, F, H, E\}$ repetimos el mismo proceso, obteniendo $T_2 = \{C, A, H, E\}$ y $T_3 = \{F\}$.

4. Finalmente, asignamos un color a cada subconjunto formado: T_1 recibe el color 1, T_2 recibe el color 2 y T_3 recibe el color 3.

En la Figura 3.4 representamos la coloración obtenida por el algoritmo LF, donde el color 1 se asocia al azul, 2 a rojo y 3 al amarillo.

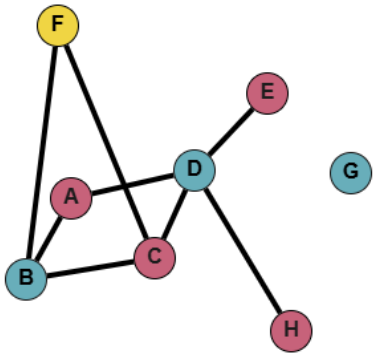


Figura 3.4: Ejemplo algoritmo Largest - First

Con base en la descripción del algoritmo Largest-First y su pseudocódigo, empleamos una herramienta de Inteligencia Artificial ChatGPT para generar un código en Python, el cual fue posteriormente revisada y ajustada por nosotras, obteniendo como resultado el Código 3.2.

```
# Largest - First
def welsh_powell_coloring(G):
    # Paso 1: ordenar vértices por grado (decreciente)
    orden_vertices = sorted(G.degree(), key=lambda item:
    item[1], reverse=True)
    orden_vertices = [v for v, d in orden_vertices]

    # Diccionario para asignar colores
    colores = {}
    color_actual = 1

    # Mientras queden vértices sin colorear
    while orden_vertices:
        # Crear un nuevo subconjunto Tk
        Tk = []
        for v in orden_vertices[:]:  # recorremos copia
   porque iremos quitando
            # Si no es adyacente a nadie en Tk, se
   agrega
            if all((vecino not in Tk) for vecino in G.
   neighbors(v)):
```

```
                Tk.append(v)
        print("Subconjunto es :",Tk)
        # Asignar color a todos los vértices de Tk
        for v in Tk:
            colores[v] = color_actual
            orden_vertices.remove(v)

        # Pasar al siguiente color
        color_actual += 1

    return colores
```

Código 3.2: Largest - First

Al ingresar un grafo de 11 vértices y 27 aristas en el Código 3.2, con el orden decreciente de grados, obtenemos como resultado los subconjuntos independientes de vértices, la cantidad de colores y el diccionario que contiene el vértice con su color correspondiente, como se ilustra en la Figura 3.5.

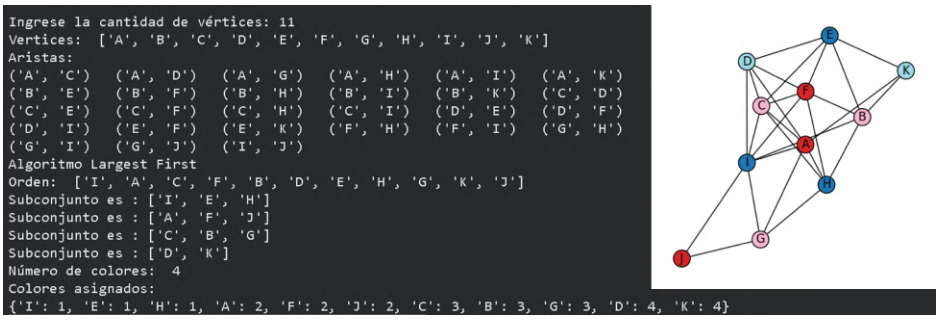


Figura 3.5: Resultado del algoritmo LF en Python

Una característica del algoritmo Largest - First es que el orden de los vértices queda determinado por la secuencia de grados decrecientes, lo cual reduce la variabilidad en la coloración obtenida para un mismo grafo. No obstante, cuando existen vértices con igual grado, el orden relativo entre ellos no queda completamente definido, situación que puede producir comportamientos similares a los observados en el algoritmo Greedy.

Por ejemplo, consideramos un grafo ciclo con 6 vértices en la Figura 3.6. Al ordenar los vértices según la secuencia (A, D, C, B, F, E) el algoritmo produce los subconjuntos independientes $T_1 = \{A, D\}$, $T_2 = \{C, F\}$, $T_3 = \{B, E\}$ en

consecuencia, se obtiene la asignación de colores T_1 con color 1(azul oscuro), T_2 con color 2 (café) y T_3 con color 3 (azul claro).

Sin embargo, sabemos que el número cromático de los ciclos pares es igual a 2; por lo tanto, la coloración obtenida difiere del valor teórico debido a la ordenación inicial de los vértices.

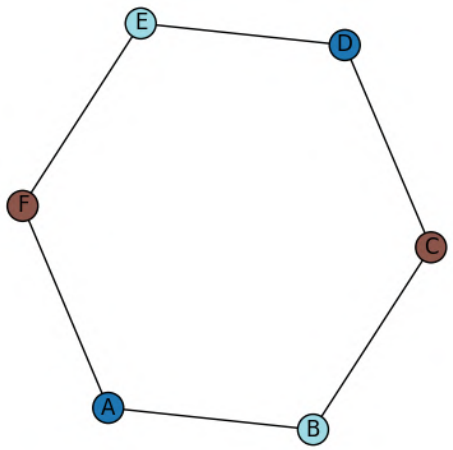


Figura 3.6: Grafo ciclo C_6

Una versión del algoritmo de Welsh - Powell, que en esencia es análoga, inicia con la lista de vértices ordenados con la secuencia de grados decreciente y le aplica el algoritmo Greedy. En el caso del ciclo con 6 vértices en la Figura 3.6, asignamos el color 1 al vértice A y también al vértice D, ya que no son adyacentes. Al considerar el vértice C, le asignamos el color 2 por ser adyacente a D, continuando el procedimiento se obtiene la coloración: B con color 3, F con color 2 y E con color 3.

La diferencia con el procedimiento descrito anteriormente radica en que, en el algoritmo Largest-First, primero completamos el subconjunto T_1 incorporando todos los vértices posibles que no presentan adyacencia entre sí antes de formar un nuevo subconjunto. En cambio, al aplicar Greedy sobre la lista ordenada, cada vértice recibe inmediatamente el primer color disponible; si resulta adyacente a algún vértice coloreado con el color 1, se le asigna directamente el color 2, sin esperar a completar un subconjunto independiente.

El algoritmo Largest-First combinado con Greedy lo podemos implementar mediante el siguiente Código 3.3, el cual representa una versión más com-

pacta del procedimiento. En esta implementación no construimos explícitamente los subconjuntos independientes ni realizamos múltiples verificaciones condicionales, lo que reduce el número de operaciones y nos permite obtener la coloración en menor tiempo de ejecución.

```
orden = sorted(G.degree(), key=lambda item: item[1],
   reverse=True)
orden = [v for v, d in orden]
colores = greedy_coloring(G, orden)
```

Código 3.3: Largest - First (2)

3.3. Algoritmo Recursive Largest First (RLF)

Una modificación del LF basada en la estructura del algoritmo conjunto independiente máximo aproximado (AMIS), es el algoritmo Recursive Largest First propuesto en Leighton, 1979. Este algoritmo busca una partición del conjunto de vértices en subconjuntos independientes cuyo número sea cercano al número cromático. Para ello, construimos conjuntos independientes a partir de una selección inicial del vértice más restrictivo, es decir, aquel con mayor grado en el subgrafo inducido por los vértices aún no coloreados.

Antes de describir detalladamente el algoritmo RLF; queremos mencionar que el algoritmo AMIS sirve para crear conjuntos independientes de gran tamaño, lo cual, en nuestro estudio nos permite generar una coloración en un grafo G. Esto lo logramos a partir de otro proceso llamado MIS que hace un conjunto independiente máximo, de la siguiente manera:

- Escogemos el vértice de grado menor en el grafo G, sea V_1 y lo guardamos en un subconjunto de vértices U.
- Hacemos el subgrafo inducido G_1 por el conjunto de vértice $V_G - \{V_1\} - N_1$, donde N_1 es el subconjunto de todos los vértices adyacentes a V_1, en G.
- En G_1, seleccionamos el vértice de grado menor, sea V_2 y lo añadimos al subconjunto U. Luego creamos el subgrafo inducido G_2 por el conjunto de vértices $V_G - \{V_1, V_2\} - N_1 - N_2$, donde N_2 es el subconjunto de todos los vértices adyacentes a V_2, en G_1.

- Agregamos en el subconjunto U el vértice de grado menor en G_{i-1}, para luego obtener el subgrafo inducido G_i por el conjunto de vértices $V_G - \{V_1, V_2, ..., V_i\} - N_1 - N_2 - ... - N_i$, donde N_i es el subconjunto de todos los vértices adyacentes a V_i, en G_{i-1}. Repetimos el proceso hasta que el subgrafo G_i no tenga vértices.

En el algoritmo AMIS, repetimos el proceso de MIS para obtener más de un conjunto independiente máximo, con la diferencia que la búsqueda se hace en el subgrafo inducido por los vértices que no pertenecen a un subconjunto independiente, además los vértices se eliminan del grafo cuando ya se completa un subconjunto independiente. Es decir, dado un grafo G

1. Empezamos con el color 1, para ello elegimos el vértice con menor grado de G, sea V_1 y a este le asignamos el color 1.

2. Luego creamos un subconjunto de vértices U con los vértices no coloreados y no adyacentes a V_1; en U buscamos el de menor grado en G, sea V_2 y le asignamos el color 1.

3. Volvemos a hacer el subconjunto U con los vértices no coloreados y no adyacentes a V_1 ni a V_2; volvemos a buscar en U el vértice de menor grado en G para darle el color 1.

4. Se repite el proceso de crear el subconjunto U con los vértices no coloreados y no adyacentes a ninguno de los ya coloreados, para seleccionar el de menor grado en G y asignarle el color 1. Hasta que $U = \varnothing$.

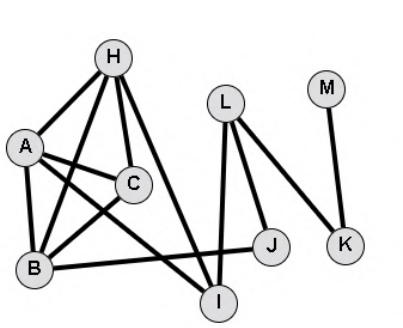
5. Continuamos con el siguiente color, por eso, hacemos el subgrafo inducido G' por los vértices no coloreados y repetimos el proceso de creación del subconjunto U y la selección de vértices de U buscando en el subgrafo inducido G' para asignarles el color en uso.

En el siguiente ejemplo, vamos a mostrar la diferencia entre los algoritmos MIS y AMIS.

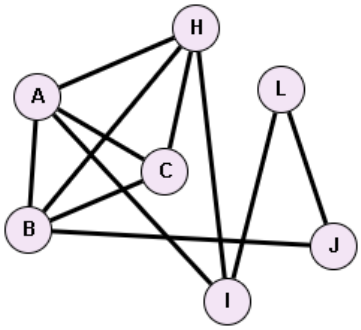
Ejemplo 3.3. Al grafo O que representamos en la Figura 3.7a, le vamos aplicar el procedimiento MIS:

- Escogemos el vértice de grado menor en O, sea M y lo agregamos al conjunto U. Hacemos el subconjunto cuyos vértices son adyacentes a M, es decir, $N_1 = \{K\}$.

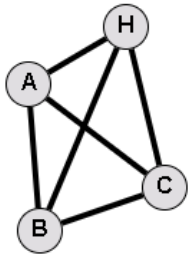
- Construimos el subgrafo inducido O_1 (Figura 3.7b) por los vértices $V_O - \{M\} - N_1 = \{L, J, I, H, C, A, B\}$. De O_1, elegimos el de menor grado, para añadirlo a U, esto es $U = \{M, L\}$, creamos un subconjunto de vértices adyacentes a L en O_1, es decir, $N_2 = \{J, I\}$.
- Repetimos los procesos. Siendo así, O_2 (Figura 3.7c) subgrafo inducido por los vértices $V_O - \{M, L\} - N_1 - N_2 = \{H, C, A, B\}$, seleccionamos a cualquiera de los de grado menor, sea H y hacemos $N_3 = \{A, C, B\}$.
- El subgrafo O_3 inducido por los vértices $V_O - \{M, L\} - N_1 - N_2 - N_3 = \{\}$, es nulo, por lo tanto se termina el algoritmo. Obteniendo un conjunto independiente máximo $U = \{M, L, H\}$.



(a) Grafo O



(b) Grafo O_1



(c) Grafo O_2

Figura 3.7: Ejemplo algoritmo MIS

Ahora vamos a aplicar el algoritmo AMIS al grafo O.

1. Iniciamos con el color 1. Al vértice de menor grado en O le asignamos el color 1, es decir, M.

2. Hacemos el subconjunto U cuyos vértices no son coloreados y no adyacentes a M, $U = \{A, B, C, H, I, J, L\}$. Del subconjunto U elegimos el de menor grado en G, sea I y le asignamos el color 1.

3. Volvemos a hacer el subconjunto U con vértices no son coloreados y no adyacentes a M ni a I, $U = \{A, B, C, J\}$. De los elementos de U seleccionamos a J para darle el color 1, porque es el de menor grado en O.

4. Repetimos, $U = \{A, C\}$, del subconjunto escogemos a A para ponerle el color 1.

5. Luego $U = \{\}$, entonces terminamos con el color 1. E iniciamos con el color 2 en el subgrafo inducido O' (Figura 3.8b) por los vértices no coloreados de O es decir, $\{H, B, C, L, K\}$

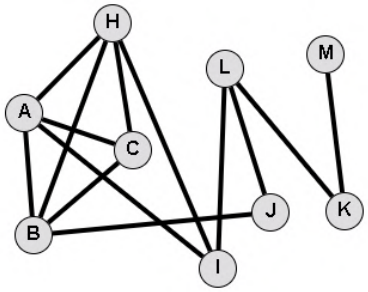
6. Volvemos a hacer todo el proceso. Escogemos a uno de los vértices de menor grado en O', queremos que sea L para darle el color 2.

7. En $U = \{H, B, C\}$ le asignamos a C el color 2.

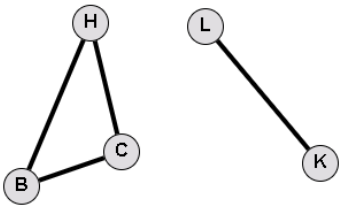
8. Luego $U = \{\}$, entonces terminamos con el color 2. Empezamos con el color 3 en el subgrafo inducido O'' (Figura 3.8c) por los vértices no coloreados de O es decir, $\{H, B, K\}$.

9. En O'' a K le proporcionamos el color 3. Y en $U = \{H, B\}$ a H también le corresponde el color 3.

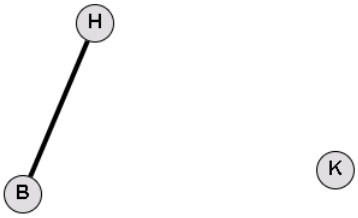
10. $U = \{\}$, terminamos con el color 3. Como el subgrafo inducido O''' (Figura 3.8d) por los vértices no coloreados de O, es decir, $\{B\}$ es el único vértice, a este la asignamos el color 4 para terminar con la coloración.



(a) Grafo O

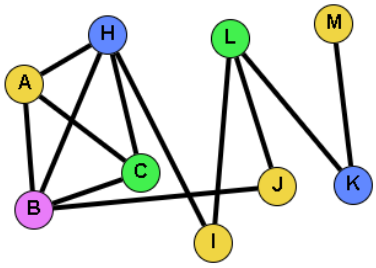


(b) Grafo O'



(c) Grafo O''

(d) Grafo O'''



(e) Grafo O coloreado

Figura 3.8: Ejemplo algoritmo AMIS

Con los algoritmos expuestos, nos permitimos presentar una descripción más detallada del algoritmo Recursive Largest First (RLF), en un grafo G:

1. Empezamos con el color 1. En G seleccionamos el vértice de mayor grado, sea V_1, para agregarlo en un conjunto independiente L_1 cuyo color asignado será 1. Creamos dos subconjuntos: U_1 cuyos vértices son no coloreados y no adyacentes a V_1 en G; y U_2 donde los vértices son no coloreados y adyacentes a V_1 en G.

2. Buscamos en U_1 el vértice que tenga más vecinos de los vértices de U_2

en G, a ese vértice, sea V_k, lo añadimos al conjunto independiente L_1.

- Si hay más de un vértice V_k consideramos empate y de esos vértices escogemos el que tenga menos vecinos en el subgrafo inducido por los elementos del subconjunto U_1.

3. Volvemos a hacer los subconjuntos U_1 y U_2, de manera que en U_1 los vértices son no coloreados y no adyacentes a ninguno de los que pertenecen a L_1 en G; y U_2 donde los vértices son no coloreados y adyacentes a alguno de los que pertenecen a L_1 en G. Para repetir el paso 2 hasta que el subconjunto U_1 quede vació.

4. Todos los vértices de L_1 tienen el color 1 y los vamos a eliminar del grafo G. Por lo tanto, tenemos el subgrafo inducido G_1 por el conjunto de vértices $V_G - L_1$ y vamos a hacer el conjunto independiente L_2 para el color 2. Para ello, repetimos los pasos 1, 2 y 3 sobre el subgrafo G_1.

5. Todos los vértices de las listas L_i tienen un color i asignado, entonces vamos a eliminar los vértices en el subgrafo correspondiente. Por lo tanto, tenemos el grafo inducido G_{i+1} por el conjunto de vértices $V_G - \{L_1 \cup L_2 \cup ... \cup L_i\}$ y seguimos con el conjunto independiente L_{i+1} para el color $i + 1$. Para ello repetimos los pasos 1,2,3 y 4 sobre el subgrafo G_{i+1}, hasta que todos los vértices de G esten coloreados.

Ejemplo 3.4. Al grafo O, que representamos en la Figura 3.9a, le vamos a aplicar el algoritmo RLF, para ello, vamos a hacer los siguientes pasos:

1. En nuestro grafo O elegimos el vértice con mayor grado, pueden ser A, B o H, en nuestro caso decidimos elegir A de manera arbitraria, para ponerlo en el conjunto de vértices independiente con el color 1, $L_1 = \{A\}$.

2. Construimos los subconjuntos de vértices $U_1 = \{L, K, M, J\}$ los cuales son no coloreados y no adyacentes a A y $U_2 = \{I, H, C, B\}$ que son no coloreados y adyacentes a A.

3. En U_1 los vértices J y L tienen un vecino en U_2, B e I, respectivamente; por lo tanto tenemos un empate. Para resolverlo identificamos los vecinos que tendrían en el subgrafo inducido por el conjunto de vértices U_1, en nuestro caso J tiene un vecino, que es L, mientras que L tiene dos

vecinos, J, K. En consecuencia elegimos el que tenga menor cantidad de vecinos, es decir, J y lo añadimos a $L_1 = \{A, J\}$.

4. Repetimos el proceso del paso 2, hasta que U_1 no tenga elementos, por lo que:

   a) $U_1 = \{K, M\}$ y $U_2 = \{B, H, C, I, K\}$, donde K tiene un vecino en U_2, por tanto lo agregamos a $L_1 = \{A, J, K\}$.

   b) Actualizamos los subconjuntos, y $U_1 = \{\}$, entonces terminamos con L_1, y el color 1.

   Construimos el subgrafo O_2 por el conjunto de vértices $V_O - L_1$ (Figura 3.9b).

5. Continuamos con el color 2 y el conjunto independiente L_2, para ello escogemos de los vértices no coloreados en el grafo O_2 el de mayor grado, H que va a pertenecer a L_2:

   a) Definimos $U_1 = \{L, M\}$ y $U_2 = \{B, I, C\}$; de los dos vértices en U_1, L es adyacente a I en el subgrafo O_2, y M no es adyacente a alguno de los vértices de U_2 en el subgrafo O_2, entonces escogemos a L, el que tiene más vértices vecinos en U_2, para agregarlo a $L_2 = \{H, L\}$.

   b) Con la actualización de L_2, hacemos $U_1 = \{M\}$ y $U_2 = \{B, I, C\}$, como M es el único vértice de U_1 lo agregamos al subconjunto de vértices L_2.

   c) Ahora, $U_1 = \{\}$; así, terminamos con el conjunto independiente $L_2 = \{H, L, M\}$ que tienen color 2.

   Hacemos el subgrafo inducido O_3 por el conjunto de vértices $O_2 - L_2$ (Figura 3.9c).

6. El siguiente color es 3 y el conjunto independiente L_3, para ello escogemos uno de los vértices no coloreados en el grafo O_3 con el de mayor grado, pueden ser C o B, de manera arbitraria escogemos C y hacemos los subconjuntos de acuerdo a $L_3 = \{C\}$:

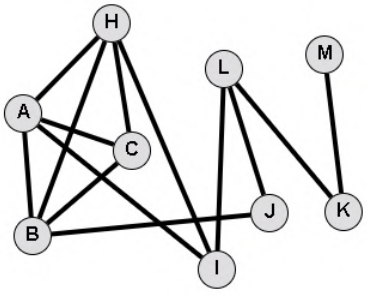
   a) $U_1 = \{I\}$ y $U_2 = \{B\}$, como I no es vecino de algún vértice en U_2 y es el único en U_1 entonces I está en L_3.

b) Actualizamos a $U_1 = \{\}$, terminamos con el conjunto independiente de color 3 $L_3 = \{C, I\}$.

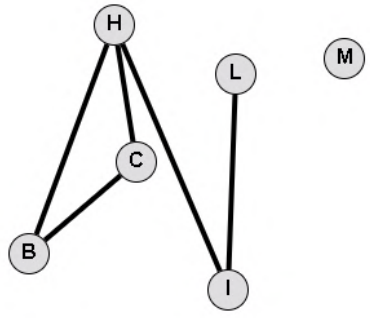
Hacemos el subgrafo inducido O_4 por el conjunto de vértices $O_3 - L_3$ (Figura 3.9d).

7. Como O_4 tiene un solo vértice, a B le asignamos el color 4, correspondiente al conjunto independiente L_4. Todos los vértices están coloreados, entonces se termina el algoritmo.

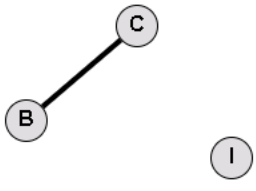
En el grafo representado en la Figura 3.9e, mostramos la coloración obtenida con el algoritmo RLF, donde los vértices de $L_1 = \{A, J, K\}$ tienen asignado 1 correspondiente al color azul, los elementos de $L_2 = \{H, L, M\}$ con color 2: rosado, los vértices de color 3 $L_3 = \{C, I\}$ representados con color amarillo y el vértice de $L_4 = \{B\}$ correspondiente al color 4 con verde.



(a) Grafo O

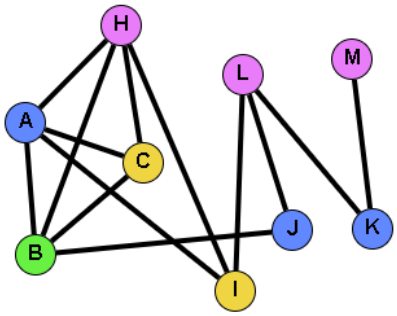


(b) Grafo O_2



(c) Grafo O_3

(d) Grafo O_4

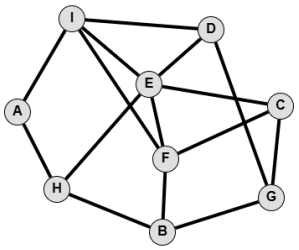


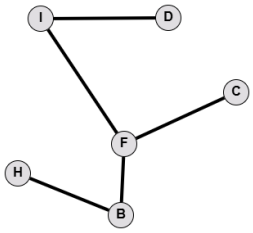
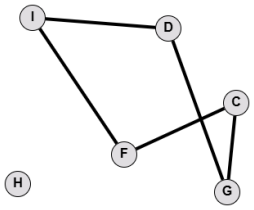
(e) Grafo O con RLF

Figura 3.9: Ejemplo algoritmo Recursive Largest - First

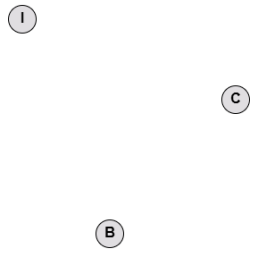
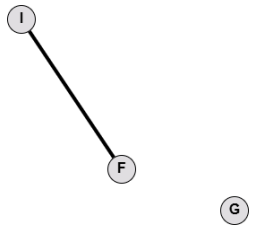
Debido a la naturaleza heurística del algoritmo RLF, este no nos asegura una única solución para un mismo grafo; incluso puede generar diferentes coloraciones dependiendo de su ejecución. Esta idea la vamos a desarrollar en el siguiente ejemplo:

Ejemplo 3.5. Al grafo W le vamos a aplicar el algoritmo RLF con el objetivo de identificar que pueden existir por lo menos dos coloraciones diferentes. Este proceso está en la Tabla 3.1.

El grafo W 
El vértice de mayor grado del grafo $\Delta(W) = 5$ es E, entonces este entra al conjunto independiente con color 1, $L_1 = \{E\}$
Construimos los subconjuntos de vértices de acuerdo con los elementos de L_1: $U_1 = \{A, B, G\}$ $U_2 = \{I, D, C, F, H\}$ Luego, en el grafo W, determinamos la vecindad de los vértices de U_1 que se estan en el subconjunto U_2, para encontrar el vértice que tenga más vecinos: $N(A) = \{I, H\}$, $N(B) = \{F, H\}$, $N(G) = \{C, D\}$ Como tenemos un empate entre los tres vértices, buscamos el que tenga menos vecinos en el subgrafo inducido por los vértices del subconjuto U_1, es decir, A y lo agregamos a $L_1 = \{E, A\}$.

<table>
<tr><td colspan="2">Con L_1 actualizado, volvemos a crear los
subconjuntos de vértices:
$U_1 = \{B, G\}$
$U_2 = \{I, D, C, F, H\}$
En el grafo W, buscamos el vértice de U_1 que tiene
más vecinos en U_2:
$N(B) = \{F, H\}$,
$N(G) = \{C, D\}$
Tenemos un empate entre ambos vértices, entonces
buscamos en el subgrafo inducido por los elementos de U_1
el de menor grafo, en este obtenemos otro empate,
entonces tenemos dos opciones para elegir:</td></tr>
<tr><th>El vértice G</th><th>El vértice B</th></tr>
<tr><td>Agregamos el vértice a
$L_1 = \{E, A, G\}$ a partir de este
creamos los subconjuntos:
$U_1 = \{\}$
$U_2 = \{I, D, C, F, H, B\}$
Como U_1 está vacío, finalizamos
el conjunto y el color 1.</td><td>Añadimos el vértice al conjunto
$L_1 = \{E, A, B\}$ y construimos los
subconjuntos:
$U_1 = \{\}$
$U_2 = \{I, D, C, F, H, G\}$
Como U_1 queda vacío,
terminamos el conjunto y color 1.</td></tr>
<tr><td>Eliminamos los vértices de L_1
del grafo W, obteniendo el grafo
W_1.
 
</td><td>Creamos el grafo inducido por el
conjunto $V(W) - L_1$, obteniendo
el grafo W_1'.
 
</td></tr>
<tr><td>Continuamos con el conjunto
independiente L_2 con el vértice
de mayor grado en W_1, es decir,
F.</td><td>Seguimos con el conjunto
independiente L_2 con uno de los
vértices de mayor grado en W_1',
arbitrariamente escogemos C.</td></tr>
</table>

A partir de L_2 creamos los subconjuntos: $U_1 = \{H, D\}$ $U_2 = \{B, I, C\}$ En el subgrafo W_1 buscamos el vértice de U_1 con más vecinos en U_2, para ello vemos sus vecindades: $N(H)_{U_2} = \{B\}$ $N(D)_{U_2} = \{I\}$ Para desempatar, elegimos el vértice de menor grado en el subgrafo inducido por los elementos de U_1, dado que tienen la misma, elegimos cualquiera, sea D y lo añadimos a $L_2 = \{F, D\}$.	Teniendo en cuenta L_2 construimos los subconjuntos: $U_1 = \{I, D, H\}$ $U_2 = \{F, G\}$ En W_1', buscamos en las vecindades de los vértices de U_1 en U_2 el de mayor grado: $N(I)_{U_2} = \{F\}$ $N(D)_{U_2} = \{G\}$ $N(H)_{U_2} = \{\}$ Como tenemos un empate, entre I y D, escogemos el de menor grado en el subgrafo inducido por los vértices de U_1; dado que ambos tienen el mismo grado, escogemos cualquiera, en nuestro caso D. Lo insertamos en $L_2 = \{C, D\}$
Con L_2 actualizado, volvemos a crear los subconjuntos: $U_1 = \{H\}$ $U_2 = \{B, I, C\}$ Al ser H, el único vértice de U_1, automáticamente lo agregamos a $L_2 = \{F, D, H\}$.	Actualizamos los subconjuntos, a partir de L_2: $U_1 = \{H\}$ $U_2 = \{F, G, I\}$ Como H es el único vértice que queda en U_1, lo agregamos a $L_2 = \{C, D, H\}$.

Al renovar los subconjuntos, teniendo en cuenta L_2; $U_1 = \{\}$, entonces terminamos el color y el conjunto independiente con $L_2 = \{F, D, H\}$. Además, los vértices de L_2 lo eliminamos del subgrafo W_1, obteniendo el subgrafo W_2 	Reestablecemos los subconjuntos desde L_2, donde $U_1 = \{\}$, por lo que terminamos el color y conjunto independiente con $L_2 = \{C, D, H\}$. También eliminamos los vértices correspondientes de W_1', teniendo el subgrafo W_2': 
En seguida, hacemos el conjunto independiente L_3, desde el subgrafo W_2. Aquí es evidente que ninguno de los vértices se conecta, por lo tanto los podemos agregar a $L_3 = \{I, CB\}$.	Seguimos con el conjunto independiente L_3. A partir del subgrafo W_2', elegimos el vértice de mayor grado, en nuestro caso tenemos empate entre dos, entonces seleccionamos F para agregarlo a $L_3 = \{F\}$.

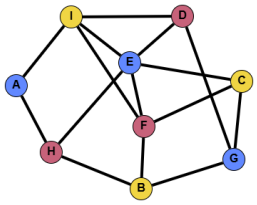
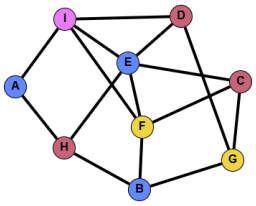
Como ya no quedan más vértices por colorear, finalizamos la coloración, teniendo el grafo W con 3 colores:  	Desde L_3, hacemos los subconjuntos: $U_1 = \{G\}$ $U_2\{I\}$ El vértice G es el único que queda en U_1, entonces lo añadimos a $L_3 = \{F, G\}$. Renovamos los subconjuntos con $U_1 = \{\}$, por eso acabamos el conjunto L_3 y eliminamos los vértices correspondientes del subgrafo W_2': 
	Concluimos, la coloración del grafo W, incluyendo el vértice I al conjunto independiente L_4. Siendo así, tenemos una 4-coloración del grafo: 

Tabla 3.1: Dos coloraciones del grafo W

En el ejemplo anterior evidenciamos que, dependiendo de la selección que hagamos ante los dobles empates, con el algoritmo podemos seguir diferentes caminos que producen diferentes coloraciones para el mismo grafo.

A continuación, en el Algoritmo 3 presentamos el funcionamiento del RLF, mediante un pseudocódigo.

Algoritmo 3 Recursive Largest First

Entrada: Grafo G y su conjunto de vértices $V = \{V_1, V_2, ..., V_n\}$
Color := 1
Vértices_colores := {} # Diccionario de vértices con sus colores
G_i := Copia de G
Mientras $V_{G_i} \neq \varnothing$ **hacer**
Si V_i es el de grado máximo de G_i **entonces**
Vértices_colores[V_i] := Color #El vértice V_i se agrega al diccionario con el color actual
end Si
L_i = [Vértices coloreados con Color]
U_2 = [Algún vértice no coloreado adyacentes a algún vértice de L_i]
U_1 = [Vértices no adyacentes no coloreados a ninguno vértice de L_i]
Mientras $U_1 \neq \varnothing$ **hacer**
Para cada V_j en U_1 **hacer**
Elegir V_j con máximo de vértices vecinos que pertencen a U_2 en el grafo G_i
Si Más de un V_j que tiene máximo de vecinos en U_2 **entonces**
Buscar entre los V_j el que tenga menos vecinos en el subgrafo inducido del conjunto de vértices U_1
Vértices_colores[V_j elegido]:= Color
Sino
Vertices_colores[V_j] := Color
end Si
En U_1 eliminar el vértice V_j
Para cada V_m en U_1 **hacer**
Si Si V_m es vecino de V_j **entonces**
Eliminar a V_m de U_1 y agregarlo a U_2
end Si
end Para
end Para
end Mientras
G_i le quitamos los vértices de L_i
Color = Color + 1
end Mientras
Salida: Diccionario de coloreado

A partir de la descripción del algoritmo y el pseudocódigo, empleamos una herramienta de Inteligencia Artificial ChatGPT para generar el código en Python, el cual fue posteriormente revisada y ajustada, obteniendo como resultado el Código 3.4.

```
def recursive_largest_first(G: nx.Graph):

    # Diccionario de colores
    vertex_colors = {}
    color = 1

    # Mientras haya vértices sin colorear
    while len(vertex_colors) < G.number_of_nodes():
        # Subgrafo inducido por los vértices no coloreados
        uncolored_nodes = [v for v in G.nodes if v not in vertex_colors]
        Gi = G.subgraph(uncolored_nodes)

        # Elegir vértice de mayor grado en Gi
        v_i = max(Gi.nodes, key=lambda v: Gi.degree(v))
        vertex_colors[v_i] = color

        # Inicializar Li: vértices coloreados con color actual
        Li = {v_i}

        # Construir U2 (adyacentes a Li) y U1 (no adyacentes a Li)
        U2 = {u for v in Li for u in Gi.neighbors(v)} - Li
        U1 = set(Gi.nodes) - Li - U2

        # Mientras aún haya candidatos en U1
        while U1:
            # Para cada v en U1, calcular vecinos en U2
            vecinos_en_U2 = {v: len(set(Gi.neighbors(v)) & U2) for v in U1}

            # Elegir v con máximo número de vecinos en
```

```
    U2
30              max_vecinos = max(vecinos_en_U2.values())
31              candidatos = [v for v, cnt in vecinos_en_U2.
    items() if cnt == max_vecinos]
32
33              if len(candidatos) > 1:
34                  # Entre los empatados, elegir el que
    tenga menos vecinos en U1
35                  vecinos_en_U1 = {v: len(set(Gi.neighbors
    (v)) & U1) for v in candidatos}
36                  elegido = min(vecinos_en_U1, key=
    vecinos_en_U1.get)
37              else:
38                  elegido = candidatos[0]
39
40              # Asignar color al elegido
41              vertex_colors[elegido] = color
42              Li.add(elegido)
43
44              # Actualizar U2 y U1
45              U2 = {u for v in Li for u in Gi.neighbors(v)
    } - Li
46              U1 = set(Gi.nodes) - Li - U2
47
48          # Al terminar, remover Li de Gi (en la práctica
    basta pasar al siguiente color)
49          color += 1
50
51      return vertex_colors
```

Código 3.4: Recursive Largest First

Al ingresar un grafo de 10 vértices y 18 aristas en el Código 3.4, obtenemos como resultado los subconjuntos independientes de vértices, la cantidad de colores y el diccionario que contiene el vértice con su color correspondiente, como se ilustra en la Figura 3.10.

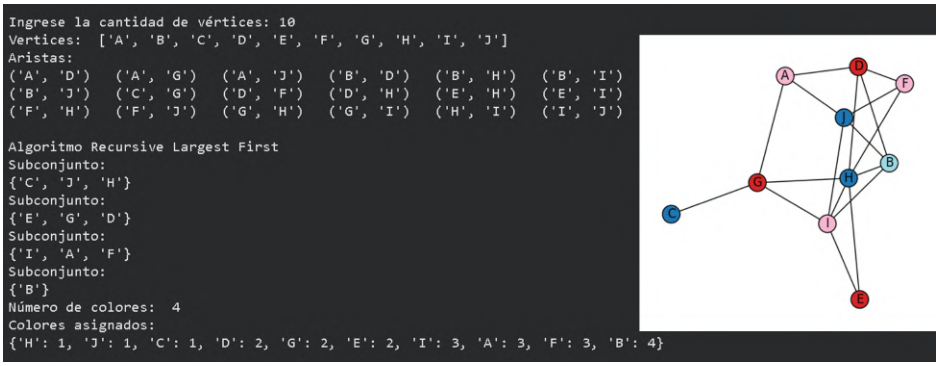


Figura 3.10: Resultado del algoritmo RLF en Python

3.4. Algoritmo Smallest Last (SL)

Otra propuesta para organizar los vértices y aplicar el algoritmo Greedy para obtener una coloración de los vértices en un grafo es la dada por Matula et al. (1972), llamada Smallest Last. Su propósito consiste en construir una ordenación de los vértices a partir de subgrafos resultantes de la eliminación sucesiva de vértices, por lo que la restricción considerada es dinámica.

En el procedimiento eliminamos, en cada paso, el vértice de menor grado del subgrafo inducido por los vértices restantes. Al eliminar primero los vértices menos conectados, aquellos con mayor conectividad relativa permanecen durante más tiempo en el proceso. En consecuencia, los últimos vértices en ser eliminados corresponden a la región más densa del grafo.

Los vértices retirados los incorporamos progresivamente al final de la lista, luego invertimos el orden de la lista y aplicamos el algoritmo Greedy con aquel orden. De esta forma, al momento de colorear, otorgamos prioridad a los vértices más restrictivos, es decir, a los más conectados, lo cual contribuye a disminuir los conflictos durante la asignación de colores.

De acuerdo con lo anterior, describimos el algoritmo Smallest Last (SL), en nuestro caso para ahorrar pasos decidimos insertar los vértices eliminados al principio de la lista creada, así no invertimos la lista resultante:

1. Dado un grafo G con n vértices y una lista vacía X, elegimos el vértice de menor grado en el grafo G, denotado por V_1, y lo agregamos a la lista X, es decir, $X = (V_1)$.

2. Creamos el subgrafo inducido G_1 por el conjunto de vértices $V_G - \{V_1\}$, que a partir de ahora denotamos como $V_G - X$.

3. Seleccionamos el vértice de menor grado en el subgrafo G_1, denotado por V_2, y lo añadimos al inicio de la lista X, obteniendo $X = (V_2, V_1)$.

4. Construimos el subgrafo inducido G_2 a partir del conjunto de vértices $V_G - X$. En este subgrafo volvemos a elegir el vértice de menor grado y lo incorporamos al inicio de la lista, quedando $X = (V_3, V_2, V_1)$.

5. Repetimos el procedimiento, construyendo sucesivamente los subgrafos inducidos G_i sobre el conjunto $V_G - X$, para escoger el vértice de menor grado en G_i y añadirlo al inicio de la lista X, hasta incluir todos los vértices de V_G.

6. Finalmente, utilizando el orden proporcionado por la lista X, aplicamos el algoritmo Greedy 1 para colorear los vértices del grafo G.

Ejemplo 3.6. Vamos aplicar el algoritmo Smallest Last al grafo W (Figura 3.11a), el cual tiene 9 vértices. Iniciamos con la lista vacía $X = ()$.

1. Escogemos el vértice de menor grado del grafo W (Figura 3.11a). Como $\delta(W) = 2$, seleccionamos el vértice A y lo agregamos a la lista X, quedando $X = (A)$.

2. Construimos el subgrafo inducido W_1 (Figura 3.11b) sobre el conjunto de vértices $V_W - X$. En este subgrafo identificamos nuevamente el vértice de menor grado en W_1, $\delta(W_1) = 2 = gr(H)$, seleccionamos el vértice H y lo añadimos al inicio de la lista X, obteniendo, $X = (H, A)$.

3. Construimos el subgrafo W_2 (Figura 3.11c) inducido por el conjunto de vértices $V_W - X$. Seleccionamos el vértice de menor grado en W_2; como $\delta(W_3) = 2 = gr(B)$, elegimos el vértice B y lo ubicamos al inicio de la lista X, obteniendo $X = (B, H, A)$.

4. Realizamos el subgrafo W_3 (Figura 3.11d) inducido por $V_W - X$. El vértice de menor grado cumple $\delta(W_3) = 2 = gr(G)$; por tanto, añadimos G al inicio de la lista X, quedando $X = (G, B, H, A)$.

5. Consideramos el subgrafo W_4 (Figura 3.11e) inducido por $V_W - X$. Aquí $\delta(W_4) = 2$, y se presenta un empate entre los vértices D y C. Elegimos

cualquiera de ellos; en nuestro caso seleccionamos D, obteniendo $X = (D, G, B, H, A)$.

6. Continuamos el mismo procedimiento hasta que la lista X contenga todos los vértices de W.

7. En el subgrafo W_5 (Figura 3.11f) inducido por $V_W - X$, se tiene $\delta(W_5) = 2$. Escogemos entre I o C; seleccionamos I, por lo que $X = (I, D, G, B, H, A)$.

8. Para el subgrafo W_6 (Figura 3.11g), con $\delta(W_6) = 2$, escogemos entre E, F o C. Tomamos el vértice E y obtenemos $X = (E, I, D, G, B, H, A)$.

9. En el subgrafo W_7 (Figura 3.11h),$\delta(W_7) = 1$, escogemos entre F y C; seleccionamos F, quedando $X = (F, E, I, D, G, B, H, A)$.

10. Finalmente, en el subgrafo W_8 (Figura 3.11i) $\delta(W_8) = 0$, por lo que añadimos el vértice C y obtenemos $X = (C, F, E, I, D, G, B, H, A)$.

11. Como la lista X contiene todos los vértices de W, terminamos la organización de vértices. Con este orden aplicamos el algoritmo Greedy (Figura 3.11j) para colorear el grafo. Usaremos la siguiente correspondencia de colores: color 1 (amarillo), color 2 (rojo) y color 3 (azul).

    a) Al vértice C le asignamos el color 1.

    b) El vértice F, es adyacente a C, recibe el color 2.

    c) El vértice E es adyacente a C y F, recibe el color 3.

    d) El vértice I es adyacente a E y F, recibe el color 1.

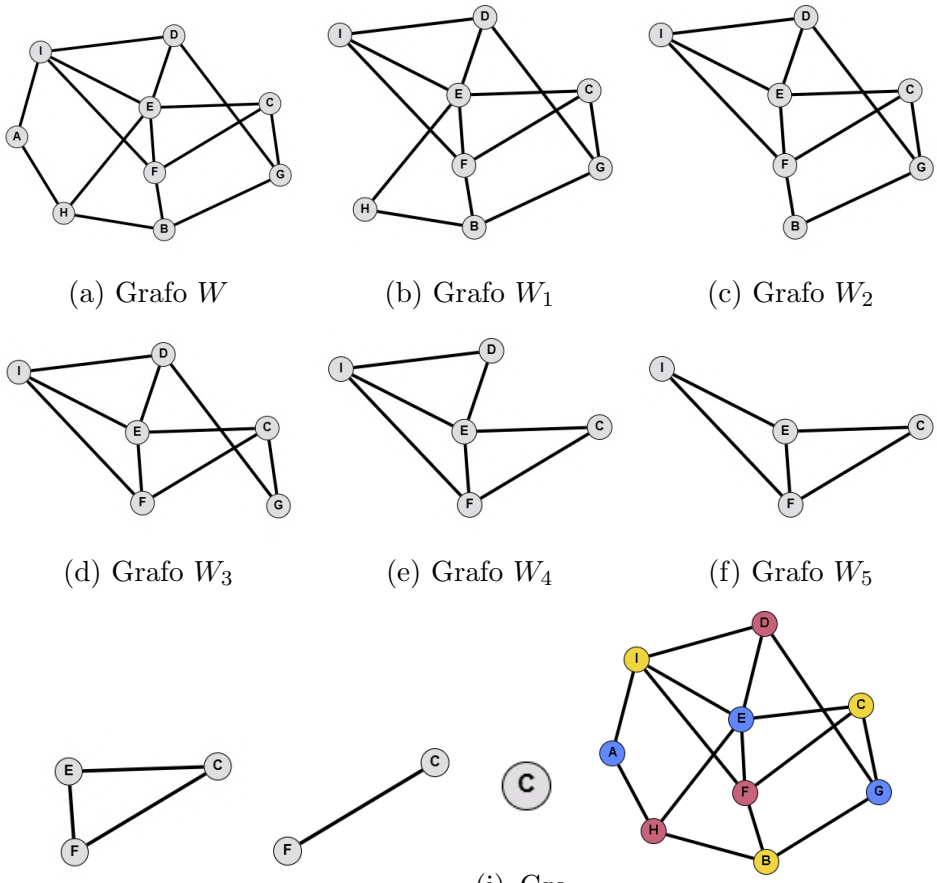
    e) El vértice D es adyacente a I y E, recibe el color 2.

    f) El vértice G es adyacente a D y C, recibe el color 3.

    g) El vértice B es adyacente a F y G, recibe el color 1.

    h) El vértice H es adyacente a E y B, recibe el color 2.

    i) El vértice A es adyacente a I y H, recibe el color 3.



(a) Grafo W (b) Grafo W_1 (c) Grafo W_2

(d) Grafo W_3 (e) Grafo W_4 (f) Grafo W_5

(g) Grafo W_6 (h) Grafo W_7 (i) Grafo W_8 (j) Grafo W coloreado

Figura 3.11: Grafo con coloración SL

Siguiendo esta idea del Algoritmo Smallest Last proponemos el siguiente Pseudocodigo en el Algoritmo 4.

Algoritmo 4 Algoritmo Smallest Last

Entrada: Grafo G con n vértices
X := () # Lista a la que se agregan vértices ordenados
V_G :={Conjunto de vértices de G}
G_i := Copia de G
Mientras $V_G \neq \varnothing$ **hacer**
V_i := Vértice de menor grado en G_i
X = (Agregar a V_i al final de la lista)
$V_G := V_G - \{X\}$
G_i := Subgrafo inducido por los vértices de V_G
end Mientras
Aplicar Algoritmo Greedy 1 con el grafo G y el orden de los vértices en X
Salida: Diccionario de coloreado

A partir de la descripción del algoritmo Smallest Last y el pseudocodigo empleamos una herramienta de Inteligencia Artificial ChatGPT para generar un código en Pyhton, el cual fue posteriormente revisado y ajustado por nosotras, obteniendo como resultado el Código 3.5.

```
def smallest_last_coloring(G):
    # Copiamos el grafo original para no modificarlo
    Gi = G.copy()
    X = []  # lista donde guardaremos el orden de
   eliminación

    # Mientras el grafo no esté vacío
    while len(Gi.nodes) > 0:
        # Obtener el vértice de menor grado
        v_min = min(Gi.nodes, key=lambda v: Gi.degree(v)
   )
        # Agregarlo a la lista X
        X.append(v_min)
        # Eliminarlo del grafo
        Gi.remove_node(v_min)

    print(" Orden: ", X )
    return X
orden = smallest_last_coloring(G)
```

```
18 colores = greedy_coloring(G, orden)
```

Código 3.5: Smallest Last

Al ingresar un grafo de 15 vértices y 40 aristas en el Código 3.5, obtenemos como resultado el orden de los vértices, luego se pasa aquel orden con el Algoritmo Greedy 3.1, para tener la cantidad de colores y el diccionario que contiene el vértice con su color correspondiente, como se ilustra en la Figura 3.12.

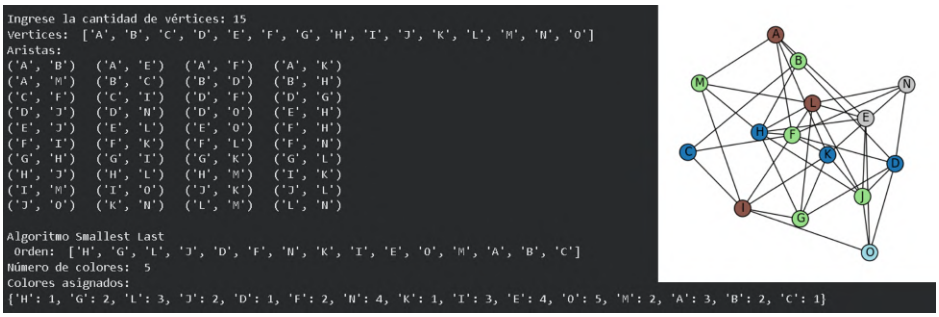
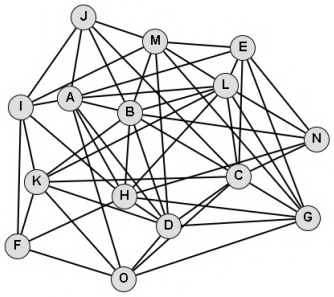


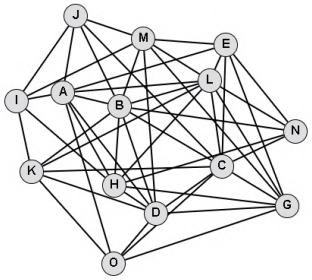
Figura 3.12: Resultado del algoritmo SL en Python

El algoritmo SL al igual que otros algoritmos heurísticos de coloración, no garantiza siempre la obtención del número cromático. Cuando existen vértices con el mismo grado mínimo en los subgrafos inducidos se presentan empates y, según el vértice que seleccionamos en cada iteración, la ordenación final puede cambiar. En consecuencia, también pueden variar la coloración obtenida y la cantidad de colores utilizados.

Ejemplo 3.7. En la tabla a continuación se presentan los resultados obtenidos al aplicar el algoritmo SL sobre el grafo W con 15 vértices Figura 3.2, considerando distintas elecciones de vértices en los casos de empate.

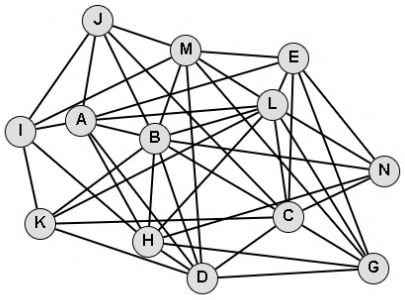
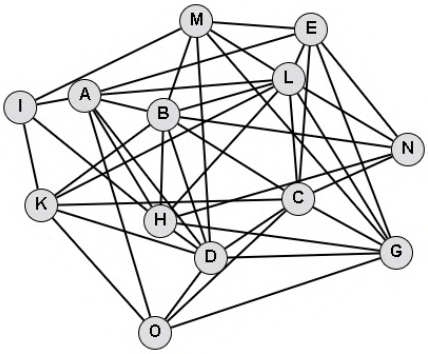
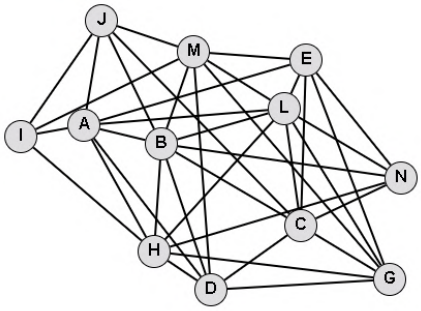
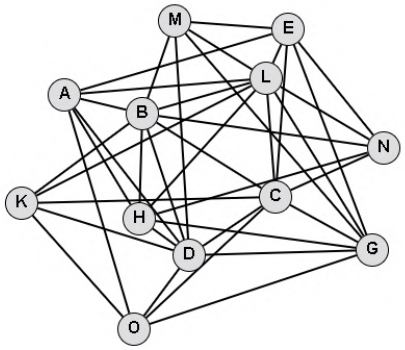


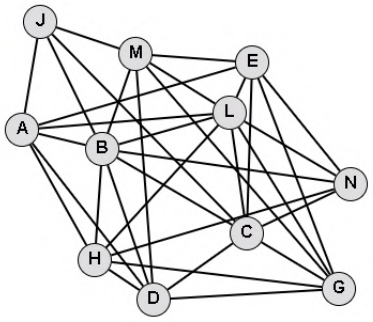
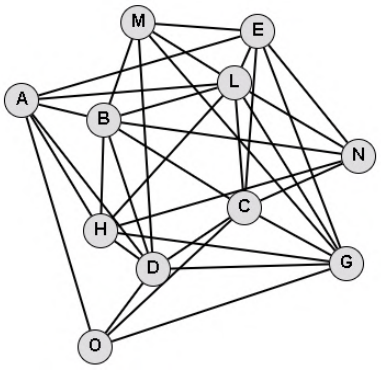
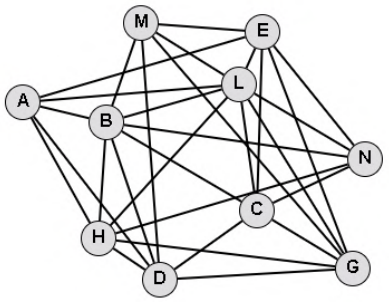
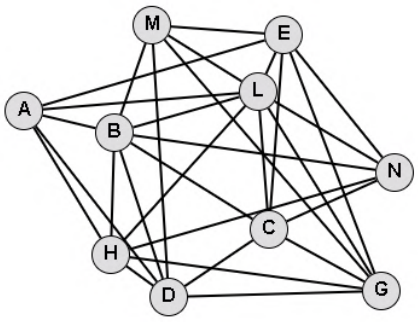
Eliminamos el vértice F que tiene el menor grado en el grafo W y lo agregamos a la lista $X = (F)$

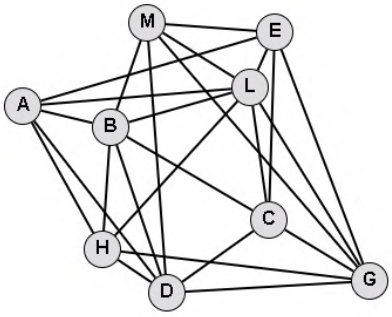
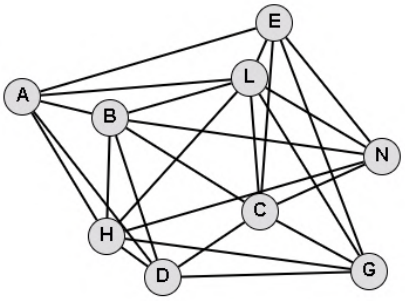
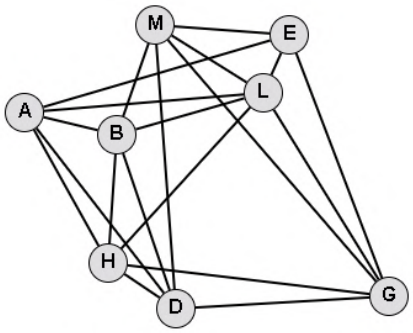
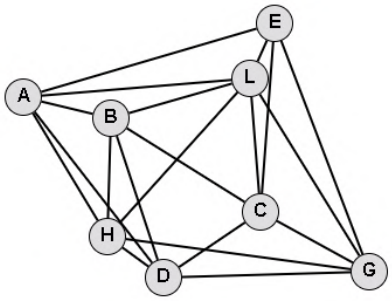


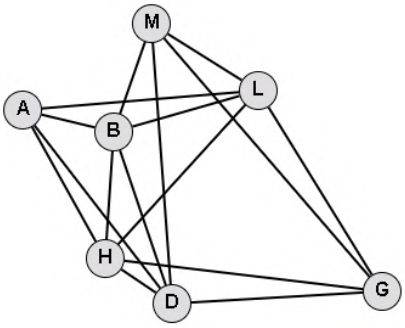
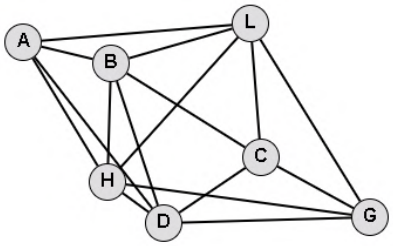
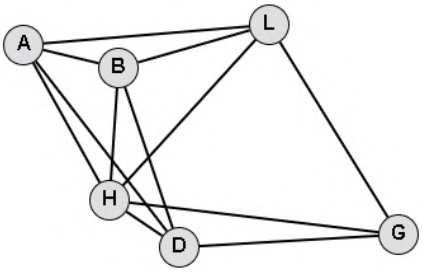
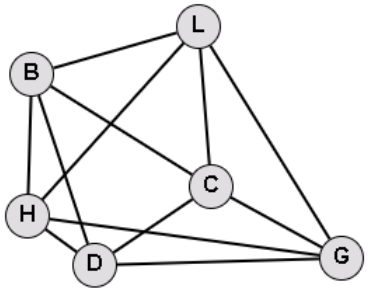
Buscamos el vértice de menor grado en el subgrafo inducido por los vértice $W - \{X\}$, aquí tenemos dos opciones

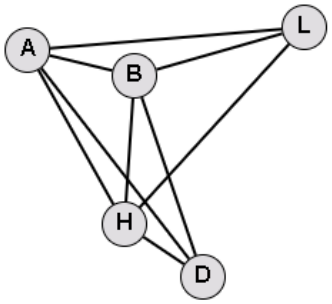
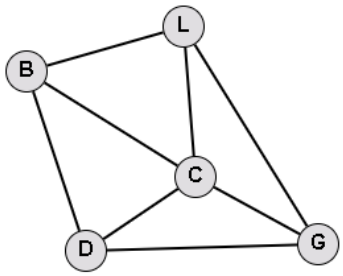
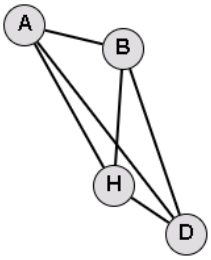
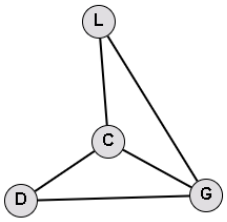
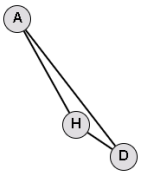
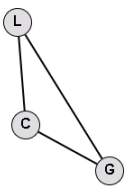
Orden de vértices 1	**Orden de vértices 2**
$\delta(W - X_1) = 5 = gr(O)$, entonces agregamos a O al inicio de la lista $X_1 = (O, F)$ y lo eliminamos del subgrafo, obteniendo:	$\delta(W - X_2) = 5 = gr(J)$, por lo que agregamos el vértice al inicio de la lista $X_2 = (J, F)$ y lo quitamos del subgrafo, generando:

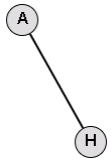
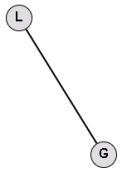
	
Buscamos el vértice de menor grado en el subgrafo $\delta(W - X_1) = 5 = gr(K)$ para insertarlo a la lista y retirarlo del subgrafo, $X_1 = (K, O, F)$, teniendo: 	Ubicamos el vértice de menor grado, $\delta(W - X_2) = 4 = gr(I)$, para ponerlo al inicio de la lista y borrarlo del subgrafo, $X_2 = (I, J, F)$, creando: 
Determinamos el vértice de menor grado, $\delta(W - X_1) = 4 = gr(I)$, lo incluimos a la lista $X_1 = (I, K, O, F)$ y quitamos del subgrafo:	Elegimos el vértice de menor grado, $\delta(W - X_2) = 5 = gr(K)$, para añadirlo a la lista $X_2 = (K, I, J, F)$ y suprimirlo del subgrafo:

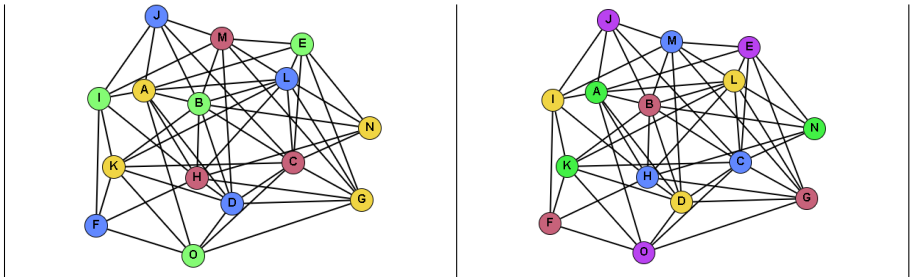
	
Volvemos a escoger el vértice con menor grado, $\delta(W - X_1) = 4 = gr(J)$, lo ponemos en la lista y quitamos del subgrafo, $X_1 = (J, I, K, O, F)$ 	Como $\delta(W - X_2) = 4 = gr(O)$, insertamos el vértice a la lista y lo eliminamos del subgrafo, $X_2 = (O, K, I, J, F)$ 
$\delta(W - X_1) = 5 = gr(N)$, por lo que agregamos el vértice al inicio de la lista $X_1 = (N, J, I, K, O, F)$ y lo quitamos del subgrafo:	$\delta(W - X_2) = 5 = gr(M)$, entonces agregamos el vértice al inicio de la lista $X_2 = (M, O, K, I, J, F)$ y lo eliminamos del subgrafo, obteniendo:

	
Ubicamos el vértice de menor grado, $\delta(W - X_1) = 5 = gr(C)$ para ponerlo al inicio de la lista y borrarlo del subgrafo, $X_1 = (C, N, J, I, K, O, F)$, creando: 	Buscamos el vértice de menor grado en el subgrafo, $\delta(W - X_2) = 5 = gr(N)$ para insertarlo a la lista y retirarlo del subgrafo, $X_2 = (N, M, O, K, I, J, F)$ 
Elegimos el vértice de menor grado, $\delta(W - X_1) = 4 = gr(E)$, para añadirlo a la lista $X_1 = (E, C, N, J, I, K, O, F)$ y suprimirlo del subgrafo:	Determinamos el vértice de menor grado, $\delta(W - X_2) = 4 = gr(E)$, lo incluimos a la lista $X_2 = (E, N, M, O, K, I, J, F)$ y quitamos del subgrafo:

	
Como $\delta(W - X_1) = 4 = gr(M)$, insertamos el vértice a la lista y lo eliminamos del subgrafo, $X_1 = (M, E, C, N, J, I, K, O, F)$ 	Volvemos a seleccionar el vértice con menor grado, $\delta(W - X_2) = 4 = gr(A)$, lo ponemos en la lista y quitamos del subgrafo, $X_2 = (A, E, N, M, O, K, I, J, F)$ 
$\delta(W - X_1) = 3 = gr(G)$, entonces agregamos el vértice al inicio de la lista $X_1 = (G, M, E, C, N, J, I, K, O, F)$ y lo eliminamos del subgrafo, obteniendo:	$\delta(W - X_2) = 4 = gr(H)$, por lo que agregamos el vértice al inicio de la lista, $X_2 = (H, A, E, N, M, O, K, I, J, F)$ y lo quitamos:

<table>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td>Buscamos el vértice de menor grado, $\delta(W - X_1) = 3 = gr(L)$ para insertarlo en la lista y retirarlo del subgrafo, $X_1 = (L, G, M, E, C, N, J, I, K, O, F)$, teniendo:
 
</td>
<td>Ubicamos el vértice de menor grado, $\delta(W - X_2) = 4 = gr(B)$, para ponerlo al inicio de la lista y borrarlo del subgrafo, $X_2 = (B, H, A, E, N, M, O, K, I, J, F)$, creando:
 
</td>
</tr>
<tr>
<td>Determinamos el vértice de menor grado, $\delta(W - X_1) = 4 = gr(B)$, lo incluimos a la lista $X_1 = (B, L, G, M, E, C, N, J, I, K, O, F)$ y quitamos del subgrafo:
 
</td>
<td>Elegimos $\delta(W - X_2) = 4 = gr(D)$, para añadirlo a la lista $X_2 = (D, B, H, A, E, N, M, O, K, I, J, F)$ y suprimirlo del subgrafo:
 
</td>
</tr>
</table>

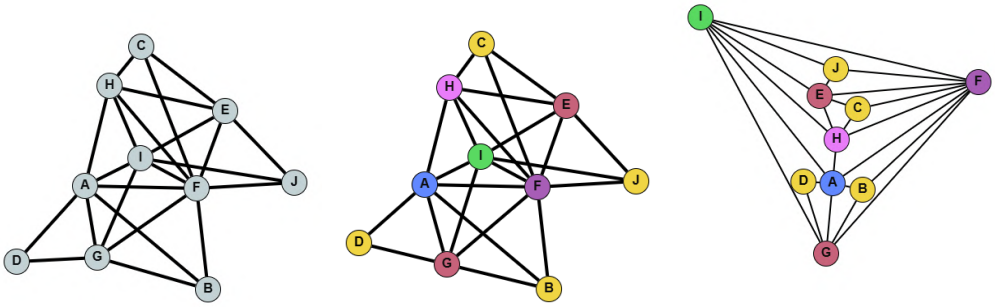
Volvemos a escoger el vértice de menor grado, $\delta(W - X_1) = 3 = gr(D)$, lo ponemos en la lista y quitamos del subgrafo $X_1 = (D, B, L, G, M, E, C, N, J, I, K, O, F)$ 	Como $\delta(W - X_2) = 3 = gr(C)$, lo insertamos en la lista y lo eliminamos del subgrafo, $X_2 = (C, D, B, H, A, E, N, M, O, K, I, J, F)$ 
Insertamos el vértice de menor grado, $\delta(W - X_1) = 2 = gr(H)$ en la lista $X_1 = (H, D, B, L, G, M, E, C, N, J, I, K, O, F)$, para eliminar y tener: 	Ubicamos $\delta(W - X_2) = 3 = gr(G)$ el vértice en la lista $X_2 = (G, C, D, B, H, A, E, N, M, O, K, I, J, F)$, para borrar y obtener: 
Finalizamos, agregando el vértice A en la lista $X_1 = (A, H, D, B, L, G, M, E, C, N, J, I, K, O, F)$. Aplicamos el algoritmo Greedy con el orden de X_1	Concluimos la lista agregando el vértice faltante a la lista $X_2 = (L, G, C, D, B, H, A, E, N, M, O, K, I, J, F)$. Además, procedemos con el algoritmo Greedy en el orden dado.



Grafo W con 4 colores | Grafo W con 5 colores

El algoritmo SL, propuesto por Matula et al. (1972), garantiza el uso de a lo sumo 6 colores al aplicarse sobre grafos planos. Observamos que en un grafo plano de 10 vértices el algoritmo requirió 6 colores, lo cual es consistente con la cota superior teórica establecida para este tipo de grafos.

Ejemplo 3.8. Dado un grafo plano P (Figura 3.13a) al aplicar el SL obtuvimos la siguiente ordenación de vértices $(D, J, B, G, C, E, H, A, I, F)$ (Figura 3.13b).



(a) Grafo P (b) Grafo P con 6 colores (c) Visualización plana del Grafo P

Figura 3.13: Grafo plano con coloración SL

3.5. Experimento computacional

En esta sección, nosotras vamos a hacer un estudio de los resultados de una prueba computacional para comparar los algoritmos Greedy, Largest First (LF), Recursive Largest First (RLF) y Smallest Last (SL). Para todos los algoritmos vamos a considerar la densidad del grafo, definida como la

proporción que relaciona la cantidad de aristas pertenecientes al grafo respecto al máximo de aristas posibles [1] ;es decir, mide que tan conectado está un grafo respecto al máximo de conexiones posibles. El análisis se realiza en función de tres criterios:

1. **Calidad cromática:** La medimos con el número de colores usados por cada algoritmo (que denotamos $\chi(G)_{alg}$) respecto a la densidad del grafo.

2. **Costo computacional:** Lo evaluamos, midiendo el tiempo de ejecución que tarda cada algoritmo para colorear un grafo con cierta densidad. Este tiempo, en segundos, corresponde al tiempo que tardó la CPU para ejecutar el algoritmo, excluyendo los tiempos de digitación, entrada/salida o espera, cuyo valor que fue obtenido mediante Python, el cual nos arrojaba automáticamente el número correspondiente al tiempo de ejecución.

3. **Desempeño:** Lo definimos como la razón entre la cantidad de colores usados por el algoritmo respecto a la cota teórica de $\Delta(G)+1$ (Teorema 2.5) [2]. Un valor cercano a 0 indica un mejor aprovechamiento de la estructura del grafo, porque tiene una mejor coloración frente al limite máximo de colores, pero si es más cercano a 1, significa que el algoritmo usa una cantidad de colores cercana a lo que exige la conectividad del grado máximo, al estar más próximos a la cota superior que utiliza un mayor número de colores. Sin embargo, este análisis cambia para los grafos completos y ciclos impares, ya que tener un desempeño de 1 significa usar el número de colores determinados por el número cromático porque para estos casos está descrito por la cota de $\Delta(G)+1$.

Para realizar la prueba computacional, escogimos grafos con tamaños $n = 50, 75, 100, 155, 200, 375, 435$ y 500, pertenecientes a las familias de grafos: completos, caminos, ciclos, bipartitos completos y $k-$regulares. Además, incluimos grafos aleatorios, los cuales se generaron a partir de una densidad previamente fijada, utilizando una función de la librería de *NetworkX*.

En cuanto a la densidad, definimos tres rangos: densidad baja: $(0, 0{,}2]$, densidad media: $(0{,}2, 0{,}6]$, densidad alta: $(0{,}6, 1]$. Posteriormente, considera-

[1] Densidad: $\frac{2m}{n(n-1)}$, donde n es la cantidad de vértices del grafo y m es la cantidad de aristas del grafo.

[2] Desempeño: $\frac{\chi(G)_{alg}}{\Delta(G)+1}$

mos las familias de los grafos involucrados atendiendo a estos rangos. Con el objetivo de mantener un balance en la cantidad de grafos por nivel de densidad, generamos los grafos en dos categorías; la primera corresponde a los grafos cuya densidad está determinada por su estructura, por ejemplo; los completos siempre pertenecen a densidad alta, y los caminos y ciclos corresponden a densidad baja. La segunda categoría incluye grafos cuya densidad puede ajustarse mediante parámetros, como los grafos $k-$regulares aleatorios y los aleatorios, lo que permite garantizar que para cada tamaño de n generamos al menos un grafo en cada rango de densidad. Al terminar con esta clasificación, y para asegurar la misma cantidad de grafos en cada rango de densidad, ajustamos el número de vértices en cada subconjunto de los grafos bipartitos completos, para poder incorporar dos tipos de grafos en la categoría de densidad media.

Teniendo en cuenta que n es la cantidad de vértices del grafo, la distribución final de los grafos según el nivel de densidad es:

- Densidad alta: Completos, $k-$regulares con $k = \frac{n}{15} + 11$ y aleatorios con densidad 0.8.
- Densidad media: Bipartitos completos, siendo p y q la cantidad de vértices de cada subconjunto de vértices, si $p < q$ con $\frac{n}{2} - 5$ y $n - (\frac{n}{2} - 5)$, bipartitos completos $p \geq q$ con $\frac{n}{2}$ y $n - (\frac{n}{2})$, $k-$regulares con $k = \frac{n}{2} + 1$ y aleatorios con densidad 0.4.
- Densidad Baja: Caminos, ciclos, $k-$regulares con $k = \frac{n}{10} + 1$ y aleatorios con densidad 0.1.

La cantidad de grafos que se pusieron a prueba, por cada tamaño, son: uno para completo, ciclo, camino, bipartitos completos con $p = q$ y $p < q$; y diez por cada $k-$regular y aleatorio de cada rango de densidad. Teniendo en total 65 grafos por cada tamaño y 520 en total. Todos ellos fueron procesados en un código de Python que implementa los cuatro algoritmos estudiados; el programa genera los grafos y ejecuta los algoritmos, registrando en un archivo de Excel la siguiente información: Tipo de grafo, su grado máximo, el algoritmo aplicado, el tiempo CPU, la densidad, el desempeño y la cantidad de colores usados. Este experimento se ejecutó en un entorno de computo de Google Colab / Google Compute Engine en Python 3, con una memoria RAM disponible de 12.67 GB y espacio en el disco disponible de 107.72 GB; el código se encuentra en el vinculo https://colab.research.google.com/drive/1PuwSgp0xC32qldYiBlfGkmPqeJxo8QWi?usp=sharing.

Para responder a los tres criterios, organizamos una tabla resumen que mostramos desde la Figura 3.14 hasta la Figura 3.22; en la que presentamos los resultados que nos brindó el programa agrupados según la densidad de los grafos (redondeada a tres cifras decimales). Para cada valor de densidad, reportamos el promedio del número de vértices de los grafos considerados y su grado máximo; además, para cada uno de los algoritmos que vamos a analizar presentamos los promedios del número de colores usados, el tiempo CPU y el desempeño; en los casos que las densidades agrupan varios grafos de distintas cantidades de vértices, resaltamos con color blanco las filas que corresponden al promedio de los datos considerados en aquel grupo.

3.6. Análisis de resultados

A partir de los datos presentados en la tabla de las figuras, vamos a analizar el comportamiento de los algoritmos respecto a las densidades de los grafos; buscando atender los tres criterios: calidad cromática, costo computacional y desempeño. Así mismo, vamos a presentar un análisis de los tres criterios comparados en cada familia de grafos que consideramos en el experimento computacional.

Calidad cromática

En densidades bajas, Greedy es el algoritmo que usa más colores, le siguen los algoritmos SL y LF con muy poca diferencia entre sus promedios calculados y el RLF es el que menos colores usa, a medida que los grafos son más densos, la diferencia entre colores usados entre el RLF y los otros algoritmos aumenta significativamente, a favor del RLF. Este comportamiento también se extiende a las densidades medias, donde RLF continua usando la menor cantidad de colores. En densidades altas, el SL resulta ser el algoritmo que usa más colores con poca diferencia del Greedy, LF y RLF, este cambio sucede porque en las densidades altas se encuentran grafos aleatorios, completos y sobre todo los k-regulares, tipo de grafos en los cuales el algoritmo SL tiene un comportamiento particular (más adelante ampliaremos esta idea).

Desempeño

En concordancia con la calidad cromática, en densidades bajas el algoritmo Greedy presenta el peor desempeño, es decir, el algoritmo usa un número

de colores muy cercano al limite teórico considerado. Le siguen los algoritmos SL y LF, mientras que RLF muestra un mejor rendimiento al aprovechar de manera más eficiente la estructura local del grafo. En densidades medias, todos los algoritmos mejoran su desempeño respecto a las densidades bajas, porque está más cerca a cero; sin embargo, el SL tiene el peor desempeño en este grupo de densidades, ya que a partir de la densidad 0.502 su desempeño es más cercano a uno, en comparación con el algoritmo Greedy. En densidades altas, los cuatro algoritmos aumentan su valor de desempeño respecto al de las densidades medias, estando más cerca a uno, además el algoritmo SL continua siendo el de peor desempeño hasta la densidad 0.775, a partir de dicho punto, el algoritmo Greedy vuelve a presentar el peor desempeño entre los algoritmos.

Costo computacional

En densidades bajas, los algoritmos LF y Greedy presentan un comportamiento similar, donde el algoritmo LF tiene menor tiempo de ejecución en el intervalo de densidades $[0{,}004, 0{,}096]$; después, el algoritmo Greedy pasa a ser el que consume menor tiempo CPU. Por su parte, RLF es el algoritmo que consume más tiempo entre los cuatro, con una diferencia bastante notable respecto a los demás. En densidades medias, RLF sigue siendo el algoritmo más costoso computacionalmente, alcanzando en algunos casos más de un segundo. En contraste, los otros tres algoritmos mantienen tiempos inferiores a un segundo, siendo SL el siguiente algoritmo con mayor consumo de tiempo, seguido del LF, mientras que Greedy resulta el menos costoso en términos computacionales. Este comportamiento general se mantiene para el rango de las densidades altas.

Familias de grafos

Ahora vamos a presentar algunos resultados que obtuvimos del análisis particular a las familias de grafos involucradas en el experimento computacional.

Bipartitos: Los cuatro algoritmos dan como resultado dos colores, correspondiente a su número cromático, por lo tanto, el desempeño es igual para todos los algoritmos. Sin embargo, para esta familia, resulta más conveniente usar el algoritmo Greedy, por lo que es el de menor costo computacional, ya que tarda en promedio 0.009 segundos. Esto se puede analizar en la tabla de

la Figura 3.19 cuando se observa en las densidades 0.49, 0.51, 0.496, 0.507, 0.5, 0.501, 0.505 que se corresponden con grafos bipartitos.

Caminos: De los cuatro algoritmos, Greedy es el que usa más colores, 3 en nuestro caso, a comparación con los otros tres algoritmos que resultan en dos colores; por ello el desempeño del algoritmo Greedy tiene valor 1, puesto que usa todos los colores disponibles de la cota $\Delta(G) + 1$, lo cual no es favorable en comparación con el desempeño de cualquiera de los otros algoritmos. Un ejemplo de esto se puede ver en la densidad 0.04, que corresponde unicamente a grafos caminos, de la tabla de la Figura 3.14. Para este tipo de familia observamos que al usar el algoritmo Largest First, da el número de colores correspondiente al número cromático, usando menos tiempo en comparación con los otros dos algoritmos.

Ciclo: El algoritmo Greedy se comporta deficiente para ciclos pares, porque nos da tres colores, que corresponden a la cota $\Delta(G) + 1$; sin embargo, para los ciclos impares, Greedy resulta dar la cantidad de colores acorde al número cromático. Cabe aclarar, que los ciclos, aunque sean pares o impares, siempre son de grado dos, es por ello que la cota $\Delta(G)+1$ resulta ser la mejor cota para los ciclos impares, porque se corresponde con el número cromático 3; esto se ve reflejado en el desempeño, porque aunque marque 1 esto no es el peor caso, al contrario es el mejor.

Para cualquiera de los casos, ciclo impar o par, Largest First es una buena opción para esta familia, porque en promedio se demora 0.0011 segundos de tiempo CPU y usa la menor cantidad de colores. Un ejemplo de los ciclos pares puede observarse en la densidad 0.041 de la tabla en la Figura 3.14.

Completos: En esta familia los algoritmos solo se diferencian en el tiempo CPU promedio, de manera que Greedy consume el menor tiempo con 0.011 segundos, seguido de Smallest Last con 0.17 segundos, después Largest First con 0.36 segundos y por último RLF que en promedio tarda 5.71 segundos. Esto se puede evidenciar en las filas que corresponden a la densidad 1 de la tabla en la Figura 3.22.

$k-$regulares: En calidad cromática, el algoritmo RLF es el que brinda la menor cantidad de colores en comparación con los otros algoritmos, a su vez RLF es el que más tiempo consume para hacer la coloración. Por otro lado, Smallest Last en la mayoría de los casos utiliza más colores que incluso Greedy y el Largest First. Sin embargo, LF brinda un mejor balance entre número de colores usados y tiempo computacional empleado. Esto se puede evidenciar, por ejemplo, en los datos correspondientes con la densidad 0.515 en la Figura 3.19.

Conclusiones

El algoritmo Greedy es muy rápido, pero su calidad cromática y desempeño no son suficientes en comparación con los otros algoritmos, en particular en densidades bajas. Recomendamos el Greedy si se quieren obtener resultados rápidos de coloración de vértices para cualquier densidad.

El algoritmo RLF ofrece la mejor calidad cromática, por lo tanto, su desempeño es el más cercano a cero, pero es más costoso computacionalmente. Es así, que recomendamos el RLF en grafos densos no completos, el algoritmo brinda la menor cantidad de colores para colorear los vértices de un grafo en comparación con los otros algoritmos.

El Smallest Last, para algunas densidades medias y altas de los grafos, usa menos colores que el algoritmo Greedy, y el tiempo computacional es mejor que el del RLF. Sin embargo, en el caso de los grafos k-regulares tiende a usar más colores que el LF y el RLF. Por lo tanto, no aseguramos que el Smallest Last pueda brindar un mejor comportamiento que los demás algoritmos, por lo que no podemos afirmar que sea un algoritmo que destaque para todo tipo de grafo.

El algoritmo Largest First, es el que presenta un mejor equilibrio entre calidad cromática y costo computacional, ya que, si bien no brinda la calidad del RLF, tampoco es igual de deficiente que Greedy o Smallest Last; así mismo su costo computacional es un poco mayor que el de Greedy pero no es tan alto como en RLF. Por eso, consideramos a LF el algoritmo que mejor funciona para la mayoría de las densidades que no cumplen las características que favorecen a los otros algoritmos, por ejemplo, la familia a la que pertenece el grafo.

Densidad	Vértices	Grado max	Greedy			Largest First			RLF			Smallest Last		
			Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño
0,004	500	2	3	0,0134	1	2	0,00305	0,66667	2	0,5907	0,66667	2	0,16595	0,66667
0,005	375	2	3	0,0071	1	2,5	0,0018	0,833335	2,5	0,34935	0,833335	2,5	0,0939	0,833335
	435	2	3	0,01	1	2,5	0,00265	0,833335	2,5	0,45235	1	2,5	0,1359	0,833335
	405	2	3	0,00855	1	2,5	0,002225	0,833335	2,5	0,40085	0,9166675	2,5	0,1149	0,833335
0,01	200	2	3	0,00255	1	2	0,00085	0,66667	2	0,09585	0,66667	2	0,02665	0,66667
0,013	155	2	3	0,0015	1	2,5	0,00065	0,833335	2,5	0,0633	0,833335	2,5	0,01805	0,833335
0,02	100	2	3	0,0008	1	2	0,0004	0,66667	2	0,027	0,66667	2	0,00785	0,66667
0,027	75	2	3	0,0005	1	2,5	0,0003	0,833335	2,5	0,01835	0,833335	2,5	0,00465	0,833335
0,04	50	2	3	0,0002	1	2	0,0001	0,66667	2	0,0062	0,66667	2	0,002	0,66667
0,041	50	2	3	0,0002	1	2	0,0001	0,66667	2	0,0062	0,66667	2	0,002	0,66667
0,089	50	10	5	0,0002	0,458335	4,5	0,0002	0,416665	4	0,00545	0,366665	4	0,0022	0,366665
0,091	50	9	4,5	0,0002	0,4596	4	0,00025	0,40404	4	0,0063	0,40404	4	0,0023	0,40404
0,092	100	17	6	0,0011	0,33333	6	0,0008	0,33333	5	0,0228	0,27778	6	0,0086	0,33333
0,093	75	13	7	0,0004	0,5	6	0,0003	0,42857	5	0,0114	0,35714	5	0,0047	0,35714
0,095	75	13	6,5	0,0004	0,464105	5	0,00035	0,358975	4,5	0,01535	0,32051	5	0,00485	0,358975
0,096	50	9	5	0,0002	0,5	4	0,0002	0,4	4	0,0063	0,4	5	0,003	0,5
	100	17,333	7	0,000967	0,38355	6,666667	0,000767	0,36797333	5	0,0261	0,27396667	6,333333	0,01	0,34836333
	87,5	15,25	6,5	0,000775	0,4126625	6	0,000625	0,37598	4,75	0,02115	0,305475	6	0,00825	0,3862725
0,097	50	11	5	0,0002	0,41667	4	0,0002	0,33333	4	0,0054	0,33333	4	0,0022	0,33333
	75	13	6	0,0005	0,43077	5	0,0005	0,358975	5	0,0132	0,358975	5,5	0,00485	0,397435
	100	19	7,5	0,0007	0,379795	6	0,0006	0,306905	5,5	0,0259	0,285165	6	0,01055	0,306905
	155	24	9	0,0021	0,36	8	0,0025	0,32	7	0,0567	0,28	9	0,0197	0,36
	375	53	15	0,0098	0,27778	14	0,0121	0,25926	12	0,4925	0,22222	15	0,1261	0,27778
	435	60	17	0,0142	0,27869	16	0,0172	0,2623	13	0,6887	0,21311	17	0,1567	0,27869
	170,625	26,5	9,125	0,003588	0,36928375	8	0,004275	0,31333125	7,125	0,165188	0,2921175	8,5	0,041938	0,33231

Figura 3.14: Densidad baja 1

Densidad	Vértices	Grado max	Greedy			Largest First			RLF			Smallest Last		
			Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño
0,098	50	10	5	0,0002	0,45455	4	0,0002	0,36364	4	0,00565	0,36364	4	0,00225	0,36364
	75	15	7	0,0004	0,4375	5	0,0003	0,3125	5	0,0117	0,3125	5	0,0046	0,3125
	155	27	10	0,0014	0,35714	8	0,0014	0,28571	7	0,0591	0,25	8	0,0203	0,28571
	200	32	11	0,0021	0,33333	9	0,0025	0,27273	8	0,105	0,24242	9	0,0503	0,27273
	500	68	20	0,0172	0,28986	18	0,0247	0,26087	15	0,9576	0,21739	17	0,2338	0,24638
	171,667	27	9,666667	0,003583	0,38782167	10,87179	0,004883	0,30984833	7,166667	0,190783	0,29159833	7,833333	0,05225	0,30743333
0,099	75	13,5	6	0,00065	0,414285	5	0,00055	0,345235	5	0,01805	0,345235	5	0,00665	0,345235
	155	26,25	9	0,0017	0,33354	8,25	0,001725	0,30541	7	0,067075	0,25942	8,25	0,0236	0,3048125
	200	31,25	11	0,002675	0,342115	10	0,00285	0,31101	8	0,1067	0,24881	10	0,0341	0,31168
	375	54,5	16	0,0125	0,28831	14	0,01695	0,252275	12	0,6414	0,216235	15,5	0,16965	0,27922
	435	65	18	0,0131	0,27273	15	0,0164	0,22727	13	0,6901	0,19697	17	0,1579	0,25758
	500	71	19,5	0,01465	0,271595	18,5	0,02695	0,2571	14	0,9824	0,194785	18,5	0,2172	0,25768
	247,459	38,2	12,06667	0,005747	0,32824867	10,86667	0,00824	0,293478	9	0,31126	0,24949333	11,2	0,07838	0,299188
0,1	155	26	9,666667	0,0014	0,35880667	8,666667	0,001433	0,32173	7	0,071667	0,25949667	8,333333	0,026033	0,30847333
	200	31,25	11	0,002925	0,34193	9,75	0,0034	0,3027775	8,25	0,1218	0,256485	10	0,0415	0,3108425
	375	55,5	22,89706	0,009475	0,287665	14,5	0,01215	0,2567825	12	0,492925	0,21257	14,5	0,120575	0,2567825
	435	64,2	17,6	0,0136	0,270412	16,2	0,02036	0,248782	13,4	0,87462	0,20588	16,8	0,20428	0,258356
	500	70,6	19,6	0,01622	0,273808	17,6	0,03136	0,245872	14,8	1,0993	0,206706	18,6	0,25814	0,259882
	354,286	58	15,42857	0,009662	0,30075714	13,90476	0,015481	0,27031905	11,57143	0,59731	0,22464952	14,28571	0,14469	0,27557667
0,101	50	11	5	0,41667	0,0002	5	0,0002	0,41667	4	0,0055	0,33333	4	0,0022	0,33333
	100	19	7	0,0011	0,35	7	0,001	0,35	5	0,0285	0,25	7	0,009	0,35
	375	54,333	16	0,013	0,28943333	15	0,0186	0,27134667	12	0,592633	0,21708	15	0,1549	0,27134667
	435	48,769	17,69231	0,013138	0,36359154	17,38462	0,020554	0,35947231	13,30769	0,836792	0,27476231	18,23077	0,175846	0,37825538
	500	71	20	0,0203	0,27864	18	0,0358	0,250775	15	1,3145	0,20898	18,5	0,233	0,258125
	396,5	48,45	16,5	0,012585	0,345947	15,95	0,01979	0,33777	12,4	0,76596	0,261222	16,5	0,161395	0,346547

Figura 3.15: Densidad baja 2

Densidad	Vértices	Grado max	Greedy			Largest First			RLF			Smallest Last		
			Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño
0,102	75	14	6	0,0006	0,4	5	0,0006	0,33333	5	0,0138	0,33333	5	0,0047	0,33333
	100	19	7	0,0007	0,35	7	0,0006	0,35	6	0,0234	0,3	6	0,0103	0,3
	155	25	9	0,0021	0,34615	8	0,0018	0,30769	7	0,0559	0,26923	9	0,0197	0,34615
	200	35	11	0,0021	0,30556	10	0,0026	0,27778	8	0,1056	0,22222	10	0,0342	0,27778
	375	38	16,3	0,01128	0,417952	16	0,01612	0,41026	12	0,61671	0,30769	17,2	0,19935	0,441028
	500	51	19,6	0,01676	0,376924	19,6	0,03255	0,376924	14,9	1,17479	0,286537	20,2	0,23322	0,388466
	386,667	40,958	16,33333	0,011913	0,38960292	16,08333	0,020513	0,38086	12,29167	0,754738	0,29446042	16,83333	0,183108	0,39800833
0,103	100	18	8	0,0011	0,42105	7	0,001	0,36842	5	0,0252	0,26316	6	0,0085	0,31579
0,104	100	23	7	0,0011	0,29167	7	0,0009	0,29167	6	0,0265	0,25	6	0,0089	0,25
	155	16	8,9	0,00183	0,523529	9,4	0,00196	0,552942	7,1	0,07283	0,417643	9,5	0,02201	0,558826
	150	16,636	8,727273	0,001764	0,50245091	9,181818	0,001864	0,52919	7	0,068618	0,40240273	9,181818	0,020818	0,53075091
0,106	75	14	6	0,0004	0,4	6	0,0003	0,4	5	0,0143	0,33333	5	0,0051	0,33333
	200	21	11	0,00298	0,5	11	0,00376	0,5	8,4	0,12816	0,38182	11,5	0,03753	0,522726
	188,636	20,364	10,54545	0,002745	0,49090909	10,54545	0,003445	0,49090909	8,090909	0,117809	0,37741182	10,90909	0,034582	0,50550818
0,108	75	8	6	0,00049	0,66667	6,1	0,00046	0,677781	5	0,01587	0,55556	5,9	0,00562	0,655559
0,111	100	11	7,3	0,00093	0,608332	7,3	0,00087	0,608332	5,9	0,02585	0,491667	7,4	0,00925	0,616666
0,114	50	10	5	0,0002	0,45455	5	0,0002	0,45455	4	0,0055	0,36364	5	0,0023	0,45455
0,122	50	6	5,1	0,00022	0,728575	5	0,00021	0,71429	4,1	0,00652	0,585716	5,1	0,00235	0,728574
0,371	50	26	11	0,0002	0,40741	9	0,0004	0,33333	8	0,0069	0,2963	9	0,0029	0,33333
0,38	50	27	10	0,0003	0,35714	10	0,0004	0,35714	8	0,0064	0,28571	10	0,0031	0,35714
0,385	50	26	11	0,0003	0,40741	10	0,0004	0,37037	9	0,007	0,33333	10	0,0031	0,37037
0,388	75	42	15	0,0009	0,34884	13	0,0015	0,30233	11	0,0184	0,25581	14	0,0064	0,32558
0,389	50	28	9	0,0003	0,31034	10	0,0004	0,34483	9	0,0063	0,31034	10	0,004	0,34483
0,39	75	37	14	0,0005	0,36842	13	0,0009	0,34211	12	0,0173	0,31579	12	0,0063	0,31579
0,391	75	40	13	0,0006	0,31707	13	0,001	0,31707	11	0,0175	0,26829	13	0,0068	0,31707
	100	49	17	0,001	0,34	15	0,0019	0,3	14	0,0486	0,28	16	0,0208	0,32
	87,5	44,5	15	0,0008	0,328535	14	0,00145	0,308535	12,5	0,03305	0,274145	14,5	0,0138	0,318535

Figura 3.16: Densidad baja y media

Densidad	Vértices	Grado max	Greedy			Largest First			RLF			Smallest Last		
			Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño
0,393	75	38,5	14	0,0005	0,35494	13	0,001	0,32959	11,5	0,01875	0,29204	14	0,00635	0,35494
	100	51	17	0,0009	0,32692	14	0,0016	0,26923	14	0,0556	0,26923	16	0,0117	0,30769
	155	75	25	0,0019	0,32895	23	0,005	0,30263	19	0,0989	0,25	21	0,0332	0,27632
	200	96	29	0,0034	0,29897	26	0,0114	0,26804	22	0,1893	0,2268	29	0,0499	0,29897
	121	59,8	19,8	0,00144	0,332944	17,8	0,004	0,299816	15,6	0,07626	0,266022	18,8	0,0215	0,318572
0,394	155	80	25	0,0035	0,30864	22	0,0093	0,2716	19	0,1814	0,23457	22	0,0536	0,2716
0,395	200	100	28	0,0031	0,27723	27	0,0095	0,26733	23	0,2029	0,22772	27	0,0533	0,26733
0,396	100	51	15	0,001	0,28846	16	0,0017	0,30769	14	0,035	0,26923	17	0,0117	0,32692
	155	76	25	0,0021	0,32468	23	0,0048	0,2987	19	0,1027	0,24675	23	0,0297	0,2987
	200	98	28	0,0043	0,28283	27	0,0095	0,27273	22	0,1898	0,22222	27	0,05	0,27273
	151,667	75	22,66667	0,002467	0,29865667	22	0,005333	0,29304	18,33333	0,109167	0,24606667	22,33333	0,030467	0,29945
0,398	200	96,5	28,5	0,00435	0,292455	27,5	0,00985	0,2822	23	0,1929	0,23595	27,5	0,05295	0,28204
	375	177,667	46	0,0161	0,25757333	45,66667	0,059933	0,25577	37,33333	1,109433	0,20900667	45	0,2217	0,25196
	435	202	51,33333	0,018	0,25295333	48	0,067967	0,23653	41,66667	1,875233	0,20535667	51	0,3329	0,25134333
	500	234	58	0,0215	0,24681	57	0,1037	0,24255	46	2,8345	0,19574	55	0,5777	0,23404
	370	174	45,22222	0,014722	0,26258889	43,66667	0,056344	0,25376111	36,55556	1,3527	0,21230333	44,22222	0,260822	0,25644778
0,399	100	52,667	17,33333	0,000933	0,32269667	16,33333	0,0019	0,30429	14	0,034033	0,26095	16,66667	0,012367	0,31034667
	375	177	45	0,014	0,25281	45	0,0445	0,25281	38	1,0844	0,21348	45	0,1998	0,25281
	435	204,667	51	0,022133	0,24811	50	0,093233	0,24324667	42,66667	1,8029	0,20760333	49,66667	0,2592	0,24164667
	500	229,6	57,6	0,02632	0,249818	54,8	0,11602	0,237664	47	2,55976	0,203846	56	0,37614	0,242858
	373,333	174,75	44,83333	0,0179	0,26786	43,16667	0,075833	0,25697833	36,91667	1,616167	0,21986417	43,66667	0,241267	0,26025667
0,4	75	39	14	0,0006	0,35	13	0,0009	0,325	12	0,018	0,3	13	0,0066	0,325
	155	77	23	0,0019	0,29487	22	0,005	0,28205	19	0,1149	0,24359	24	0,0292	0,30769
	375	179,75	46,75	0,0134	0,258925	44,75	0,049425	0,2477625	37	1,051625	0,20488	45,5	0,181075	0,2519825
	435	203,5	53	0,0194	0,25918	49,5	0,0716	0,242085	42	1,5898	0,205355	51	0,25325	0,2494
	500	232	58,5	0,02395	0,251075	56	0,10165	0,240345	47	2,3115	0,20172	56	0,3544	0,24034
	360	170,6	44,7	0,01428	0,270108	42,5	0,05501	0,256296	35,7	1,2142	0,217726	43,3	0,19754	0,26201

Figura 3.17: Densidad media 1

Densidad	Vértices	Grado max	Greedy			Largest First			RLF			Smallest Last		
			Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño
0,401	200	98	28,5	0,00495	0,287855	28	0,01375	0,282855	23	0,26985	0,232345	29	0,0824	0,29296
	375	178	47	0,0122	0,26257	45	0,0493	0,2514	38	1,0677	0,21229	47	0,1936	0,26257
	435	201	52	0,0179	0,25743	50	0,0706	0,24752	42	1,6118	0,20792	51	0,4557	0,25248
	500	232,5	58	0,0261	0,248405	56,5	0,1019	0,24198	47,5	2,3157	0,203425	56,5	0,34955	0,241955
	368,333	173,333	45,33333	0,015367	0,26542	44	0,058533	0,25809833	36,83333	1,308433	0,21529167	44,83333	0,2522	0,26414667
0,402	50	32	11	0,0003	0,33333	10	0,0004	0,30303	9	0,0081	0,27273	10	0,0031	0,30303
	155	78,667	23,66667	0,002367	0,29770333	22,33333	0,0064	0,28111	18,33333	0,127733	0,23080667	23	0,0376	0,28891333
	200	101	28	0,0031	0,27451	27	0,0093	0,26471	24	0,2007	0,23529	28	0,0571	0,27451
	435	202	50	0,0184	0,24631	50	0,0697	0,24631	43	1,5945	0,21182	51	0,2757	0,25123
	191,667	95,167	26,66667	0,004817	0,29121	25,66667	0,016433	0,27623	21,83333	0,364417	0,23537667	26,33333	0,074783	0,282585
0,403	50	27	11	0,0003	0,39286	10	0,0004	0,35714	9	0,0064	0,32143	10	0,003	0,35714
	200	101	28,5	0,0044	0,279615	27,5	0,01025	0,26981	23,5	0,20785	0,230385	27,5	0,04935	0,26981
	375	177	45	0,0203	0,25281	44	0,0892	0,24719	37	1,6291	0,20787	45	0,1976	0,25281
	206,25	101,5	28,25	0,00735	0,301225	27,25	0,027525	0,2859875	23,25	0,5128	0,2475175	27,5	0,074825	0,2873925
0,404	100	53	18	0,0009	0,33333	16	0,002	0,2963	15	0,0341	0,27778	17	0,0121	0,31481
	155	76	24	0,0019	0,31169	22	0,0049	0,28571	19	0,1012	0,24675	24	0,0293	0,31169
	127,5	64,5	21	0,0014	0,32251	19	0,00345	0,291005	17	0,06765	0,262265	20,5	0,0207	0,31325
0,405	75	38	14	0,0005	0,35897	13	0,001	0,33333	12	0,0183	0,30769	13	0,0066	0,33333
	100	54	19	0,0009	0,34545	17	0,0018	0,30909	15	0,0341	0,27273	16	0,012	0,29091
	155	76	27	0,002	0,35065	22	0,0051	0,28571	20	0,1021	0,25974	23	0,0329	0,2987
	110	56	20	0,001133	0,35169	17,33333	0,002633	0,30937667	15,66667	0,0515	0,28005333	17,33333	0,017167	0,30764667
0,406	50	28	11	0,0003	0,37931	9	0,0004	0,31034	10	0,007	0,34483	9	0,0033	0,31034
	155	78	25	0,0019	0,31646	23	0,0048	0,29114	19	0,1173	0,24051	22	0,0287	0,27848
	102,5	53	18	0,0011	0,347885	16	0,0026	0,30074	14,5	0,06215	0,29267	15,5	0,016	0,29441
0,407	50	28	10	0,0006	0,34483	10	0,0004	0,34483	9	0,0073	0,31034	11	0,003	0,37931
	75	40	14	0,0007	0,34146	14	0,001	0,34146	12	0,0172	0,29268	13	0,0067	0,31707
	100	52	19	0,0009	0,35849	17	0,0017	0,32075	14	0,0357	0,26415	17	0,0215	0,32075
	75	40	14,33333	0,000733	0,34826	13,66667	0,001033	0,33568	11,66667	0,020067	0,28905667	13,66667	0,0104	0,33904333
0,413	50	27	11	0,0003	0,39286	9	0,0007	0,32143	9	0,0076	0,32143	10	0,0031	0,35714

Figura 3.18: Densidad media 2

Densidad	Vértices	Grado max	Greedy			Largest First			RLF			Smallest Last		
			Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño
0,414	100	55	17	0,0009	0,30357	18	0,0021	0,32143	15	0,0342	0,26786	17	0,0116	0,30357
0,415	75	41	14	0,0009	0,33333	13	0,0019	0,30952	12	0,0347	0,28571	14	0,0126	0,33333
0,418	75	39	14	0,0005	0,35	14	0,0012	0,35	11	0,0174	0,275	14	0,0072	0,35
0,432	50	29	13	0,0003	0,43333	10	0,0004	0,33333	10	0,0072	0,33333	11	0,003	0,36667
0,49	50	30	2	0,0003	0,06452	2	0,0003	0,06452	2	0,017	0,06452	2	0,0035	0,06452
0,496	75	43	2	0,0007	0,04545	2	0,0011	0,04545	2	0,0552	0,04545	2	0,0076	0,04545
0,5	100	55	2	0,001	0,03571	2	0,0018	0,03571	2	0,1175	0,03571	2	0,013	0,03571
0,501	155	83	2	0,0035	0,02381	2	0,0062	0,02381	2	0,4305	0,02381	2	0,0315	0,02381
	200	105	2	0,0054	0,01887	2	0,0127	0,01887	2	0,9259	0,01887	2	0,0514	0,01887
	375	190,5	2	0,0155	0,010445	2	0,0755	0,010445	2	6,41795	0,010445	2	0,2044	0,010445
	435	220,5	2	0,0195	0,00903	2	0,1183	0,00903	2	9,88145	0,00903	2	0,3475	0,00903
	500	252,5	2	0,03095	0,00789	2	0,1709	0,00789	2	15,39505	0,00789	2	0,35175	0,00789
	371,875	189,375	2	0,0176	0,01217625	2	0,093538	0,01217625	2	8,093163	0,01217625	2	0,236275	0,01217625
0,502	435	218	65,3	0,02199	0,298173	65,5	0,11422	0,299085	53,2	2,14619	0,242923	68	0,33265	0,310503
0,503	155	78	2	0,0032	0,02532	2	0,0062	0,02532	2	0,4212	0,02532	2	0,0323	0,02532
	200	100	2	0,0049	0,0198	2	0,012	0,0198	2	0,918	0,0198	2	0,0576	0,0198
	375	188	58,4	0,01808	0,308996	57,7	0,08098	0,305293	47,2	1,39703	0,249738	60,3	0,24115	0,319047
	500	251	73,3	0,02957	0,290872	73,5	0,18215	0,291665	59,9	3,23616	0,237701	76	0,43088	0,301588
	413,864	207,636	60,04545	0,022027	0,27471818	59,81818	0,120432	0,27339545	48,86364	2,166868	0,22361409	62,13636	0,309555	0,28415773
0,505	100	50	2	0,0011	0,03922	2	0,0017	0,03922	2	0,1262	0,03922	2	0,0126	0,03922
0,506	155	78	29,3	0,00238	0,370888	29,1	0,00735	0,368356	23,8	0,13556	0,301268	30,9	0,03584	0,391141
0,507	75	38	2	0,0006	0,05128	2	0,0008	0,05128	2	0,0545	0,05128	2	0,0075	0,05128
0,508	200	101	35,9	0,00448	0,351963	35,9	0,01463	0,351962	29,1	0,26664	0,285291	37,7	0,06605	0,369609
0,51	50	25	2	0,0003	0,07692	2	0,0003	0,07692	2	0,0181	0,07692	2	0,0032	0,07692
0,514	75	38	17,6	0,00074	0,451284	17,3	0,00159	0,443592	14,3	0,02312	0,366665	18,5	0,00869	0,47436
0,515	100	51	21,8	0,00112	0,419234	21,6	0,00268	0,415388	17,4	0,04171	0,334612	22,6	0,01305	0,434618
0,531	50	26	13,2	0,00032	0,488888	13,7	0,0006	0,507408	11	0,00901	0,40741	13,5	0,00346	0,5
0,667	100	66	28,2	0,0011	0,420897	28,1	0,00347	0,419404	23,1	0,04962	0,344775	30,1	0,0152	0,449256
0,694	50	34	17	0,00033	0,485712	16,9	0,00082	0,482854	14,6	0,00968	0,417142	18,2	0,00376	0,520002

Figura 3.19: Densidad media y alta

Densidad	Vértices	Grado max	Greedy			Largest First			RLF			Smallest Last		
			Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño
0,714	155	110	43,5	0,00326	0,391894	43,6	0,01382	0,392795	36,1	0,18352	0,325224	45,5	0,04515	0,40991
0,724	200	144	55,1	0,00519	0,379999	54,6	0,0257	0,376551	45	0,33645	0,310342	56,5	0,07386	0,389657
0,729	500	364	116,4	0,03724	0,318906	116,2	0,33442	0,318358	97,9	4,66615	0,268216	120,4	0,56223	0,329866
0,737	435	320	105,2	0,02892	0,327725	105,1	0,23217	0,327414	87,9	3,2318	0,273832	109,3	0,40595	0,340497
0,738	375	276	92,3	0,02093	0,333213	93,5	0,15304	0,337545	77,8	2,08259	0,280868	97,3	0,29579	0,351263
0,757	75	56	26,7	0,00076	0,46842	26,2	0,00246	0,459649	22,2	0,03	0,389472	27,5	0,00966	0,482457
0,775	50	44	22	0,0003	0,48889	19	0,0009	0,42222	18	0,0109	0,4	22	0,0042	0,48889
0,783	50	44	21	0,0003	0,46667	19	0,0008	0,42222	18	0,0099	0,4	19	0,0038	0,42222
0,786	75	66	28	0,0011	0,41791	27	0,0033	0,40299	25	0,0296	0,37313	27	0,0088	0,40299
0,791	75	66	29	0,0011	0,43284	26	0,0025	0,38806	23	0,0298	0,34328	26	0,0091	0,38806
	100	89	36	0,0018	0,4	35	0,0053	0,38889	30	0,0558	0,33333	33	0,0175	0,36667
	87,5	77,5	32,5	0,00145	0,41642	30,5	0,0039	0,388475	26,5	0,0428	0,338305	29,5	0,0133	0,377365
0,792	50	48	21	0,0003	0,42857	19	0,0009	0,38776	18	0,011	0,36735	19	0,0039	0,38776
	75	65	29	0,0007	0,43939	28	0,0026	0,42424	25	0,031	0,37879	27	0,0087	0,40909
	62,5	56,5	25	0,0005	0,43398	23,5	0,00175	0,406	21,5	0,021	0,37307	23	0,0063	0,398425
0,794	155	134	52	0,0027	0,38519	49	0,0149	0,36296	44	0,1935	0,32593	48	0,0414	0,35556
	200	174	64	0,0043	0,36571	61	0,0332	0,34857	54	0,4028	0,30857	61	0,073	0,34857
	177,5	154	58	0,0035	0,37545	55	0,02405	0,355765	49	0,29815	0,31725	54,5	0,0572	0,352065
0,795	50	46	22	0,0006	0,46809	21	0,0016	0,44681	18	0,0105	0,38298	22	0,004	0,46809
	100	88	36	0,0019	0,40449	35	0,0061	0,39326	32	0,0621	0,35955	36	0,0163	0,40449
	155	135	51	0,0036	0,375	49	0,014	0,36029	42	0,1947	0,30882	50	0,0399	0,36765
	101,667	89,667	36,33333	0,002033	0,41586	35	0,007233	0,40012	30,66667	0,0891	0,35045	36	0,020067	0,41341
0,796	50	45	21	0,0006	0,45652	19	0,0017	0,41304	17	0,0112	0,36957	18	0,0043	0,3913
	75	65	29	0,0007	0,43939	27	0,0021	0,40909	24	0,0258	0,36364	29	0,0095	0,43939
	100	89,5	37,5	0,0012	0,414465	35,5	0,0049	0,39231	31	0,05575	0,34255	35,5	0,0157	0,39231
	155	135	54	0,005	0,39706	49	0,0272	0,36029	43	0,3372	0,31618	50	0,0709	0,36765
	200	173	63	0,006	0,36207	59	0,0263	0,33908	54	0,3736	0,31034	62	0,0715	0,35632
	113,333	99,5	40,33333	0,00245	0,413995	37,5	0,011183	0,38435333	33,33333	0,143217	0,340805	38,33333	0,031267	0,38988

Figura 3.20: Densidad alta 1

Densidad	Vértices	Grado max	Greedy			Largest First			RLF			Smallest Last		
			Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño
0,797	75	67	29	0,0007	0,42647	29	0,0024	0,42647	24	0,0267	0,35294	28	0,0102	0,41176
	155	133	51	0,0027	0,3806	49	0,0137	0,36567	43	0,1783	0,3209	51	0,0387	0,3806
	200	172	62,5	0,00645	0,36127	61	0,02775	0,3526	53	0,37795	0,30636	62	0,06735	0,35838
	157,5	136	51,25	0,004075	0,3824025	50	0,0179	0,374335	43,25	0,240225	0,32164	50,75	0,0459	0,37728
0,798	50	45	22	0,0003	0,47826	20	0,001	0,43478	18	0,0105	0,3913	20	0,0039	0,43478
	155	135	54	0,0029	0,39706	48	0,0146	0,35294	42	0,1824	0,30882	51	0,0424	0,375
	200	173	62	0,0064	0,35632	60	0,0274	0,34483	53	0,3588	0,3046	59	0,0719	0,33908
	375	322,5	105,5	0,0174	0,32612	102	0,14125	0,3153	91	2,0612	0,281295	105	0,2849	0,32456
	435	370,5	122,5	0,0319	0,329815	117	0,2337	0,314965	103,5	3,19895	0,27864	119,5	0,3731	0,321685
	500	423	133	0,0443	0,31368	133	0,3231	0,31368	114	4,6013	0,26887	133	0,4931	0,31368
	315,625	270,25	90,875	0,019063	0,35714875	87,375	0,1395	0,338345	77	1,959163	0,2991825	89	0,240913	0,34437875
0,799	75	67,5	30	0,0007	0,43798	27,5	0,00235	0,40143	24,5	0,0291	0,35763	29,5	0,0097	0,43084
	100	89	37,33333	0,001467	0,41481333	34,33333	0,0059	0,38148333	30,33333	0,06	0,33703333	36,33333	0,016233	0,40370333
	200	172	60	0,0062	0,34682	60	0,0267	0,34682	55	0,3698	0,31792	61	0,079	0,3526
	375	321,333	108	0,0216	0,33512667	104	0,1923	0,32267	90,66667	2,4413	0,28129667	104	0,383967	0,32265667
	435	374,75	119,25	0,024075	0,317405	118,25	0,2175	0,3147025	102	3,73645	0,271485	118	0,37065	0,314065
	500	429,5	136,5	0,05175	0,31709	131,5	0,59165	0,305475	116	5,08045	0,269465	133,5	0,5184	0,31012
	301	259,733	87,06667	0,01844	0,35842667	84,4	0,17862	0,34212667	73,8	2,202573	0,30086933	85,33333	0,25456	0,351324
0,8	200	173	66	0,0072	0,37931	61	0,0253	0,35057	54	0,3603	0,31034	63	0,0742	0,36207
	375	323,333	106,3333	0,018467	0,32786	101,6667	0,168533	0,31346333	90,66667	2,465967	0,27955333	104,3333	0,2713	0,32168667
	500	424,8	136,2	0,03158	0,319868	131,8	0,32218	0,309536	115	5,22844	0,270088	133,8	0,57368	0,314238
	425	363	118,4444	0,0245	0,32913667	113,8889	0,237978	0,31540444	100,1111	3,766711	0,27771556	116,1111	0,417389	0,32203556
0,801	155	133	53	0,0026	0,39552	50	0,0147	0,37313	44	0,1934	0,32836	49	0,0416	0,36567
	200	172	64	0,0082	0,36994	62	0,0516	0,35838	55	0,7083	0,31792	61	0,121	0,3526
	375	321	106	0,017	0,32919	105	0,1407	0,32609	91	2,1162	0,28261	101	0,2661	0,31366
	435	371,75	120,5	0,0284	0,3232725	116,5	0,27115	0,3125325	102,75	3,353225	0,2756625	119	0,3747	0,3192525
	500	427,5	135	0,03125	0,31506	130	0,32855	0,30338	115	5,16355	0,26838	132,5	0,5092	0,309215
	385,556	329,778	108,3333	0,022656	0,33531778	104,7778	0,216522	0,32383222	92,33333	2,9731	0,28536667	105,7778	0,327322	0,32526333
0,802	375	324	108	0,0166	0,33231	105	0,1547	0,32308	90	2,0616	0,27692	107	0,3091	0,32923

Figura 3.21: Densidad alta 2

Densidad	Vértices	Grado max	Greedy			Largest First			RLF			Smallest Last		
			Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño	Colores	Tiempo CPU	Desempeño
0,803	50	29,667	22	0,00035	0,483575	20,5	0,00095	0,45048	19	0,01115	0,41739	21	0,00415	0,461595
	75	67	30	0,0007	0,44118	26	0,0023	0,38235	25	0,0277	0,36765	28	0,0087	0,41176
	155	135	54	0,0043	0,39706	51	0,0274	0,375	45	0,348	0,33088	50	0,0727	0,36765
	200	176	64	0,0071	0,36158	60	0,0254	0,33898	53	0,6324	0,29944	62	0,1314	0,35028
	106	93,4	38,4	0,00256	0,433394	35,6	0,0114	0,399458	32,2	0,20608	0,36655	36,4	0,04422	0,410576
0,804	100	87	36,5	0,0012	0,41489	36	0,00515	0,40914	32	0,05905	0,36381	36	0,0165	0,40914
	155	135	54	0,0038	0,39706	51	0,0156	0,375	43	0,1787	0,31618	51	0,0419	0,375
	200	176	65	0,0043	0,36723	61	0,0279	0,34463	56	0,3826	0,31638	61	0,0731	0,34463
	138,75	121,25	48	0,002625	0,3985175	46	0,01345	0,3844775	40,75	0,16985	0,340045	46	0,037	0,3844775
0,805	75	66	29	0,0012	0,43284	27	0,0027	0,40299	24	0,0284	0,35821	28	0,0085	0,41791
	155	136	54	0,0027	0,39416	52	0,0155	0,37956	47	0,1954	0,34307	51	0,0426	0,37226
	115	101	41,5	0,00195	0,4135	39,5	0,0091	0,391275	35,5	0,1119	0,35064	39,5	0,02555	0,395085
0,806	50	45	22	0,0003	0,478015	20,5	0,001	0,4461	18,5	0,0107	0,40213	21	0,0039	0,45721
	155	136	52	0,0026	0,37956	51	0,0176	0,37226	45	0,2045	0,32847	52	0,0397	0,37956
	85	75,333	32	0,001067	0,44519667	30,66667	0,006533	0,42148667	27,33333	0,0753	0,37757667	31,33333	0,015833	0,43132667
0,812	75	67	29	0,0012	0,42647	28	0,0046	0,41176	26	0,0511	0,38235	29	0,0156	0,42647
0,813	100	89	36	0,0012	0,4	36	0,005	0,4	31	0,0575	0,34444	39	0,0158	0,43333
1	50	49	50	0,0006	1	50	0,0033	1	50	0,0253	1	50	0,0049	1
	75	74	75	0,0011	1	75	0,0089	1	75	0,0757	1	75	0,0106	1
	100	99	100	0,0018	1	100	0,0138	1	100	0,194	1	100	0,0192	1
	155	154	155	0,0039	1	155	0,0475	1	155	0,5873	1	155	0,0454	1
	200	199	200	0,0072	1	200	0,1627	1	200	1,2903	1	200	0,0898	1
	375	374	375	0,0174	1	375	0,5587	1	375	8,9002	1	375	0,2872	1
	435	434	435	0,0256	1	435	0,8521	1	435	13,5593	1	435	0,4024	1
	500	499	500	0,0306	1	500	1,2784	1	500	21,0956	1	500	0,5412	1
	236,25	235,25	236,25	0,011025	1	236,25	0,365675	1	236,25	5,715963	1	236,25	0,175088	1

Figura 3.22: Densidad alta 3

Capítulo 4

Aplicaciones

En el siguiente capítulo vamos a presentar dos aplicaciones de la coloración de vértices en grafos y su estudio con los algoritmos heurísticos estudiados en los capítulos anteriores.

4.1. Sudoku

El sudoku es un juego clásico que consiste en llenar una cuadrícula de 9 filas por 9 columnas (9×9) con subcuadrículas de 3×3, cuyo objetivo es ubicar los números del 1 al 9 en cada una de las casillas atendiendo a las condiciones:

1. Cada número debe estar solo una vez en cada fila.
2. Cada numero debe estar solo una vez en cada columna.
3. No se puede repetir un número en cada subcuadrícula.

Una versión más pequeña del sudoku es considerar una cuadricula de tamaño 4×4 (Figura 4.1) con las mismas reglas mencionadas anteriormente, solo que en este caso ubicamos los números del 1 al 4.

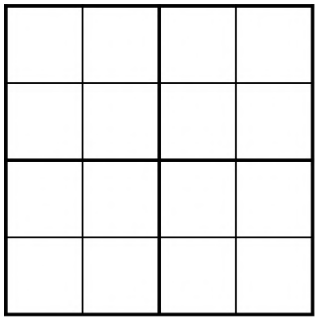


Figura 4.1: Sudoku 4x4

El sudoku se puede representar como un grafo, donde cada casilla de la cuadrícula es un vértice, cuyos nombres los vamos a designar de acuerdo a la posición de la fila y la columna, como en la Figura 4.2, con el fin de ejemplificar mejor las conexiones entre vértices, por ejemplo el vértice $A_{2,3}$ corresponde a la fila 2, columna 3:

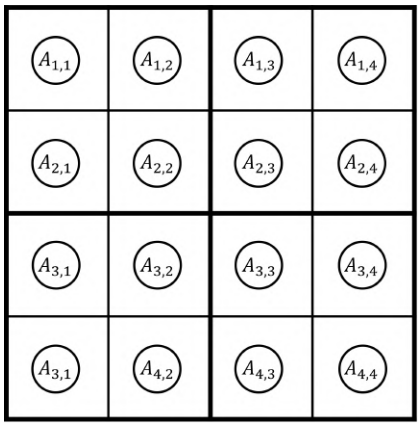


Figura 4.2: Vértices del sudoku

Además, las reglas del juego nos permiten determinar las aristas entre los vértices. La regla 1 indica que los números que aparecen en cada fila son diferentes por lo que lo expresamos a partir de la conexión, dos a dos, de los vértices que representan las casillas de esa fila. La Figura 4.3 ilustra las aristas trazadas entre los vértices de la fila 1, por ejemplo, $A_{1,1}$ se conecta con los vértices $A_{1,2}, A_{1,3}, A_{1,4}$.

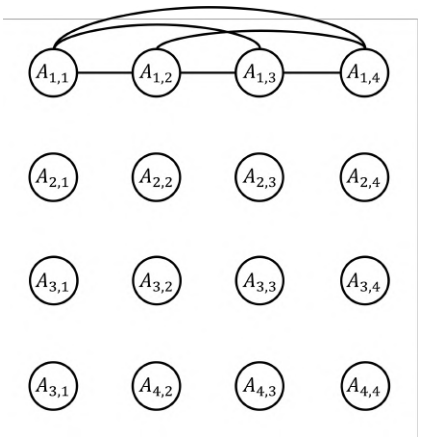


Figura 4.3: Conexión entre filas

Así mismo, los vértices se conectan con los de su misma columna, por ejemplo, $A_{1,1}$ se conecta con los vértices $A_{2,1}, A_{3,1}, A_{4,1}$ como se observa en la Figura 4.4.

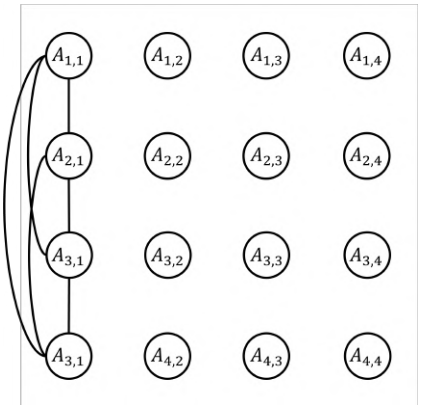


Figura 4.4: Conexión entre columnas

Dado que en la subcuadrícula tampoco se pueden repetir los números, también se establece una conexión entre aquellos que representan las casillas que hacen parte de dicha subcuadrícula, en nuestro ejemplo, la Figura 4.5 ilustra el caso de la subcuadrícula ubicada en la parte superior izquierda.

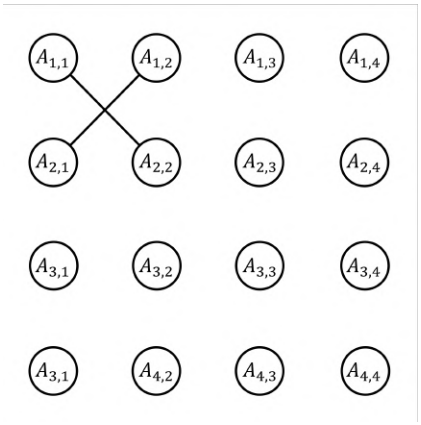


Figura 4.5: Conexión en subcuadrícula

Si extendemos estas conexiones a todos los vértices, tenemos un grafo $7-$regular que de ahora en adelante lo vamos a llamar Grafo Sudoku y lo representamos en la Figura 4.6.

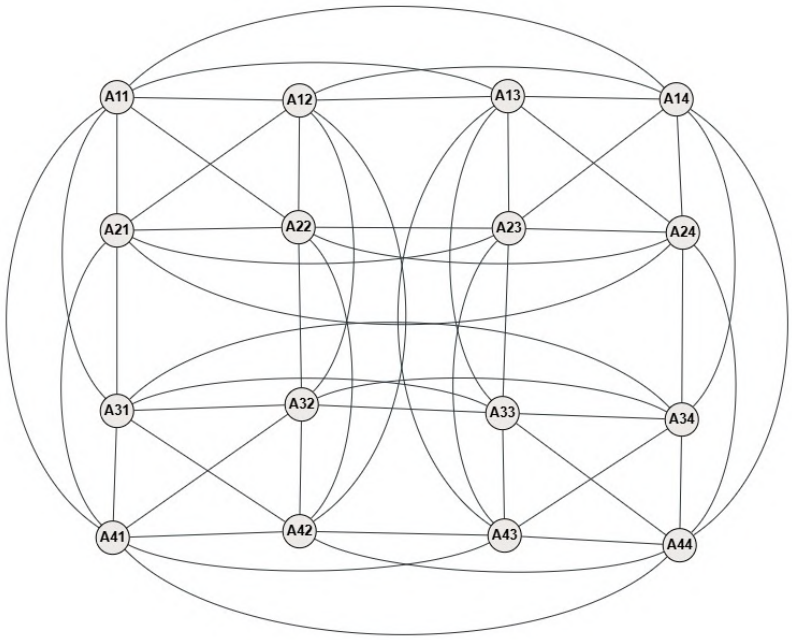


Figura 4.6: Grafo del sudoku

La solución a un sudoku, atendiendo a su representación como grafo, la podemos obtener realizando una $4-$coloración, es decir, interpretando cada número del 1 al 4 como un color, por ejemplo, vamos a resolver el sudoku de la Figura 4.7, para ello a cada número le vamos a asignar un color 1:amarillo, 2:rosado, 3:verde, 4:naranja.

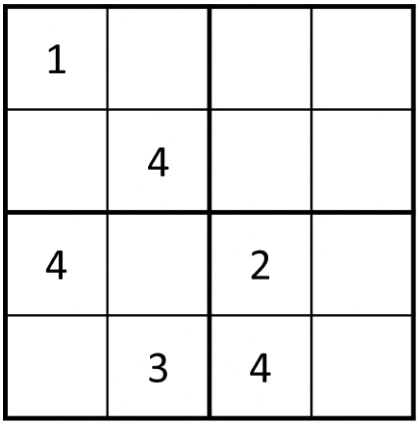


Figura 4.7: Sudoku con pistas

Sudoku vs Algoritmos Heurísticos

Para resolver un sudoku, basta con asignar a cada número un color y colorear el grafo correspondiente; en el caso de un sudoku de 4×4, esto puede realizarse con 4 colores. Esto se justifica porque existe un grafo $4-$completo que establece una cota inferior de 4; además, el algoritmo nos presenta una cota superior igual a 4. Por lo tanto, el número cromático del grafo es 4. Sin embargo, no todos los algoritmos resultan adecuados para este propósito, por lo que, con base en el análisis realizado y el estudio del capítulo anterior, revisaremos cuáles son los más apropiados para resolver este tipo de problemas.

Cuando lo intentamos con el algoritmo Greedy, dependiendo del orden inicial, nos puede dar más de cuatro colores:

En el Grafo sudoku (Figura 4.6), tomamos el orden de vértices: $A_{3,2}, A_{1,2}$, $A_{3,4}, A_{4,1}, A_{2,4}, A_{4,2}A_{1,3}, A_{2,2}, A_{1,4}, A_{3,1}, A_{2,3}, A_{4,3}, A_{1,1}, A_{4,4}, A_{3,3}, A_{2,1}$.

El número 1 se corresponde con el color amarillo; 2:rosado; 3:verde; 4:naranja, con el orden ya descrito, estos colores no son suficientes, porque en el vértice $A_{4,4}$ ya están gastados los cuatro, dando paso a un quinto color (azul),

así mismo sucede con el vértice $A_{3,3}$ que hace extender la lista de colores a seis (rojo).

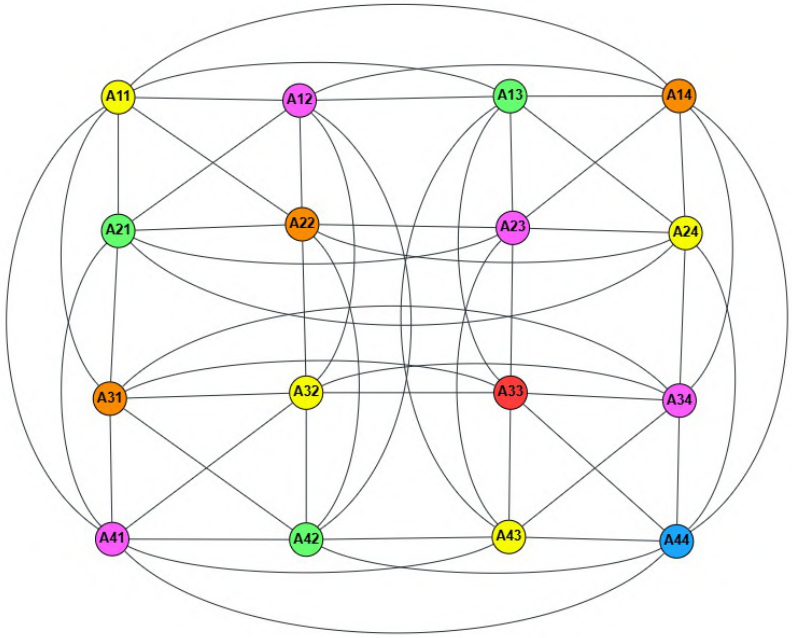


Figura 4.8: Sudoku con Greedy

Como se evidencia en el ejemplo, el algoritmo Greedy no es el más conveniente para resolver sudokus, algo que ya teníamos identificado por su comportamiento con los $k-$regulares. Esto mismo sucede con el algoritmo Largest First, ya que como es un grafo $k-$regular, todos los vértices tienen el mismo grado, por tanto, LF resulta ser casi igual a Greedy. Por otro lado, el Smallest Last, como se vio en la prueba computacional, es el que tiene peor comportamiento, ya que el numero de colores usados es mas cercano a la cota $\Delta(G)+1$ con los $k-$regulares, siendo poco efectivo para resolver sudokus.

Sin embargo, el RLF, tras el experimento computacional, es el que usa menos colores para los k-regulares, por lo tanto, es el más conveniente para este tipo de sudokus.

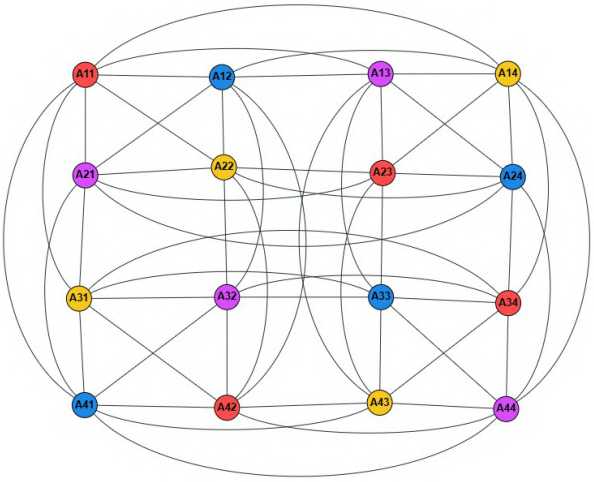
Para aplicar el algoritmo RLF en el Grafo sudoku (Figura 4.6), debemos iniciar con el vértice de mayor grado y ubicarlo en el conjunto L_1, como es un grafo $k-$regular escogemos cualquiera de los vértices, en nuestro caso $A_{4,4}$.

A partir de este conjunto hacemos los subconjuntos de vértices U_1 con los vértices no coloreados y no adyacentes a los que pertenecen al conjunto L_1, y U_2 con los vértices no coloreados y adyacentes a los que pertenecen al conjunto L_1; sean $U_1 = \{A_{1,1}, A_{1,2}, A_{1,3}, A_{2,1}, A_{2,2}, A_{2,3}, A_{3,1}, A_{3,2}\}$ y $U_2 = \{A_{4,3}, A_{4,2}, A_{4,1}, A_{3,4}, A_{2,4}, A_{1,4}, A_{3,3}\}$. De los vértices que pertenecen a U_1 seleccionamos el que tiene más vecinos en el subconjunto U_2, como tenemos un empate entre los vértices $A_{1,3}, A_{3,1}, A_{3,2}$ escogemos el que tiene menos vecinos en el conjunto U_1, pero también hay un triple empate entre estos vértices, entonces escogemos cualquiera de ellos, sea $A_{1,3}$, y lo agregamos al conjunto L_1.

Con los vértices del conjunto, volvemos a realizar los subconjuntos de vértices, U_1 y U_2 para ir actualizando el conjunto L_1 hasta que el subconjunto U_1 esté vacío. Siendo así, los conjuntos de vértices creados son:

- $L_1 : \{A_{4,4}, A_{1,3}, A_{3,2}, A_{2,1}\}$
- $L_2 : \{A_{4,2}, A_{1,1}, A_{2,3}, A_{3,4}\}$
- $L_3 : \{A_{2,2}, A_{1,4}, A_{4,3}, A_{3,1}\}$
- $L_4 : \{A_{1,2}, A_{2,4}, A_{4,1}, A_{3,3}\}$

Los conjuntos obtenidos, son subconjuntos de vértices independientes, por lo tanto a los vértices de cada conjunto les asignamos un color, es decir, los vértices del conjunto L_1 tienen el color morado; los de L_2 son de color rojo; con L_3 están de color amarillo; y en L_4 son de azul, como ilustramos en la Figura 4.9a. Obteniendo así, una de las soluciones del sudoku, ya que al remplazar los colores por los números del 1 al 4 tenemos el sudoku de la Figura 4.9b.



(a) Grafo Sudoku coloreado con RLF

2	4	1	3
1	3	2	4
3	1	4	2
4	2	3	1

(b) Solución del Sudoku correspondiente a colores

Figura 4.9: Una solución del sudoku

Habitualmente nos encontramos con sudokus que dan pistas para su solución, como al iniciar el algoritmo RLF seleccionamos el vértice de mayor grado y el grafo sudoku es $7-$regular, entonces escogemos uno de los vértices que ya tiene asignado un color, es decir, un vértice que representa una de las casillas en las que se encuentra una pista.

Por ejemplo, tenemos el sudoku de la Figura 4.10.

4		3	
	1		
		1	2

Figura 4.10: Sudoku con pistas

Acorde con los nombres de los vértices del grafo sudoku (Figura 4.6), determinamos que los vértices $A_{3,2}$ y $A_{4,3}$ tienen color 1-verde; el vértice $A_{4,4}$ color 2-amarillo; el vértice $A_{2,3}$ color 3-azul; y el vértice $A_{2,1}$ color 4-rosado.

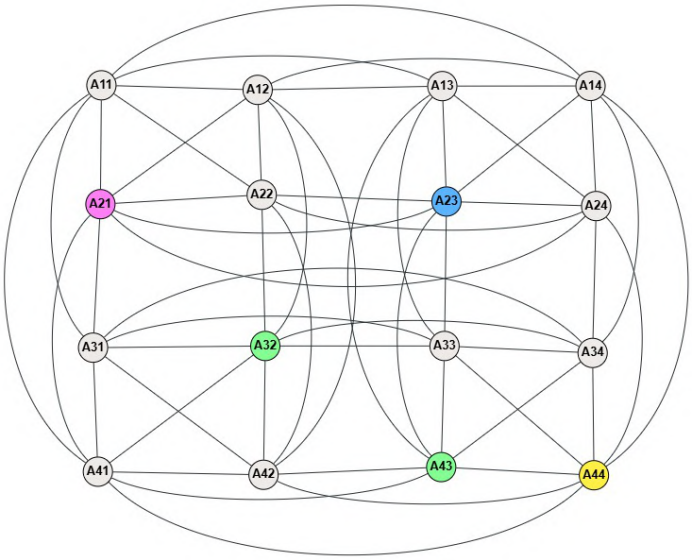


Figura 4.11: Grafo del sudoku con pistas coloreadas

Teniendo en cuenta estos datos, procedemos con el algoritmo RLF, de la siguiente manera:

1. Iniciamos al conjunto L_1, ubicando los vértices ya coloreados con el color 1, es decir, $L_1 = \{A_{3,2}, A_{4,3}\}$. A partir de este conjunto hacemos los subconjuntos de vértices U_1 con los *vértices no coloreados y no adyacentes* a los del conjunto L_1, y U_2 con los *vértices no coloreados y adyacentes* a los que pertenecen al conjunto L_1. De esta manera obtenemos:

   - $U_1 = \{A_{1,1}, A_{1,4}, A_{2,4}\}$
   - $U_2 = \{A_{1,2}, A_{1,3}, A_{2,2}, A_{3,1}, A_{3,3}, A_{3,4}, A_{4,1}, A_{4,2}\}$

2. De los vértices de U_1, escogemos el que tiene más vecinos en el subconjunto U_2, siendo $A_{1,1}$ y lo agregamos al conjunto $L_1 = \{A_{3,2}, A_{4,3}, A_{1,1}\}$.

3. A partir de la actualización de L_1, volvemos a crear los subconjuntos $U_1 = \{A_{2,4}\}$ y $U_2 = \{A_{1,2}, A_{1,3}, A_{2,2}, A_{3,1}, A_{3,3}, A_{3,4}, A_{4,1}, A_{4,2}, A_{1,4}\}$

4. Como el vértice $A_{2,4}$ es el único del subconjunto U_1, lo agregamos al conjunto $L_1 = \{A_{3,2}, A_{4,3}, A_{1,1}, A_{2,4}\}$. Al actualizar los subconjuntos, U_1 queda vacío; por lo tanto terminamos el conjunto L_1, de manera que los vértices de este estarán coloreados con el color 1. Además, los vértices del conjunto L_1 los eliminamos del grafo sudoku, como se ilustra en la Figura 4.12.

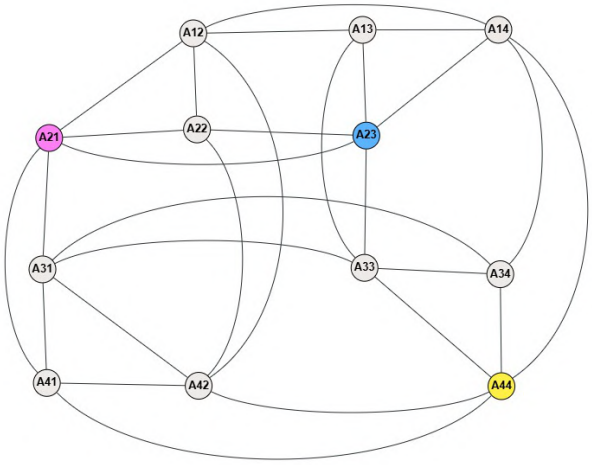


Figura 4.12: Subgrafo del sudoku - L_1

5. Con el subgrafo de la Figura 4.12 continuamos con el color 2, para ello en el conjunto L_2 incluimos el vértice que ya tiene asignado este color, $L_2 = \{A_{4,4}\}$, y creamos los subconjuntos:

   - $U_1 = \{A_{1,2}, A_{1,3}, A_{2,2}, A_{3,1}\}$
   - $U_2 = \{A_{1,4}, A_{3,3}, A_{3,4}, A_{4,1}, A_{4,2}\}$

6. En U_1, el vértice $A_{3,1}$ es el que tiene más vecinos en U_2, entonces lo agregamos al conjunto $L_2 = \{A_{4,4}, A_{3,1}\}$, para hacer los subconjuntos:

   - $U_1 = \{A_{1,2}, A_{1,3}, A_{2,2}\}$
   - $U_2 = \{A_{1,4}, A_{3,3}, A_{3,4}, A_{4,1}, A_{4,2}\}$

7. En U_1 tenemos un empate entre los vértices $A_{1,2}$ y $A_{1,3}$ con dos vecinos en U_2, y para escoger uno de ellos vemos cuál tiene menor cantidad de vecinos en el mismo subconjunto U_1, siendo $A_{1,3}$ el que ingresa al conjunto $L_2 = \{A_{4,4}, A_{3,1}, A_{1,3}\}$.

   - $U_1 = \{A_{2,2}\}$
   - $U_2 = \{A_{1,4}, A_{3,3}, A_{3,4}, A_{4,1}, A_{4,2}, A_{1,2}\}$

8. El vértice $A_{2,2}$ es el último que queda en el subconjunto U_1, entonces lo agregamos al conjunto $L_2 = \{A_{4,4}, A_{3,1}, A_{1,3}, A_{2,2}\}$. Al crear los subconjuntos de vértices con el conjunto L_2 actualizado, obtenemos que U_1 está vacío, entonces terminamos el proceso de incluir vértices en L_2 y estos vértices los eliminamos del subgrafo de la Figura 4.12, como se ilustra en el subgrafo de la Figura 4.13.

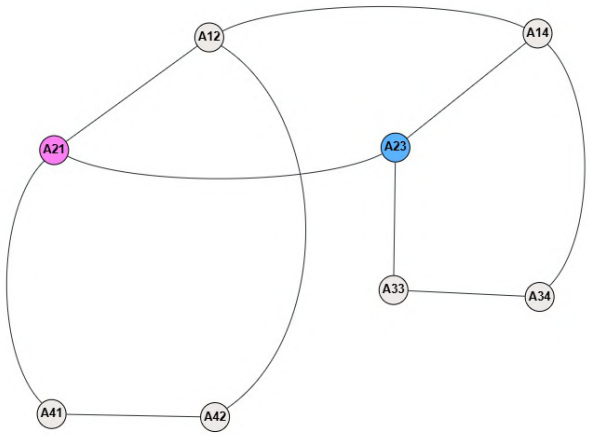


Figura 4.13: Subgrafo del sudoku - L_2

9. A partir del subgrafo de la Figura 4.13, continuamos con el color 3 y al conjunto correspondiente le agregamos el vértice que tiene asignado el color 3, $L_3 = \{A_{2,3}\}$, con este conjunto obtenemos los subconjuntos:

   - $U_1 = \{A_{1,2}, A_{3,4}, A_{4,1}, A_{4,2}\}$
   - $U_2 = \{A_{1,4}, A_{3,3}\}$

10. De U_1 escogemos el vértice $A_{3,4}$, que tiene dos vecinos en U_2, para incluirlo en el conjunto $L_3 = \{A_{2,3}, A_{3,4}\}$ y actualizar los subconjuntos:

- $U_1 = \{A_{1,2}, A_{4,1}, A_{4,2}\}$
- $U_2 = \{A_{1,4}, A_{3,3}\}$

11. En el subconjunto U_1 seleccionamos el vértice $A_{1,2}$, que tiene un vecino en U_2 a diferencia de $A_{4,1}$ y $A_{4,2}$, así renovamos $L_3 = \{A_{2,3}, A_{3,4}, A_{1,2}\}$:

    - $U_1 = \{A_{4,1}\}$
    - $U_2 = \{A_{1,4}, A_{3,3}, A_{4,2}\}$

12. Terminamos el color 3, agregando el único vértice que queda en U_1 a $L_3 = \{A_{2,3}, A_{3,4}, A_{1,2}, A_{4,1}\}$, los vértices de este conjunto los eliminamos del subgrafo de la Figura 4.13, como ilustramos en la Figura 4.14.

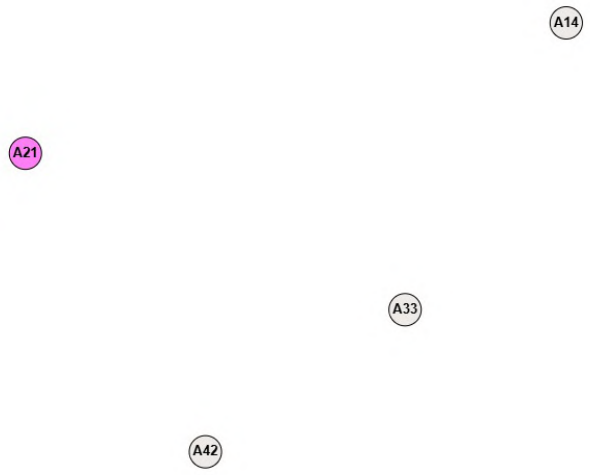
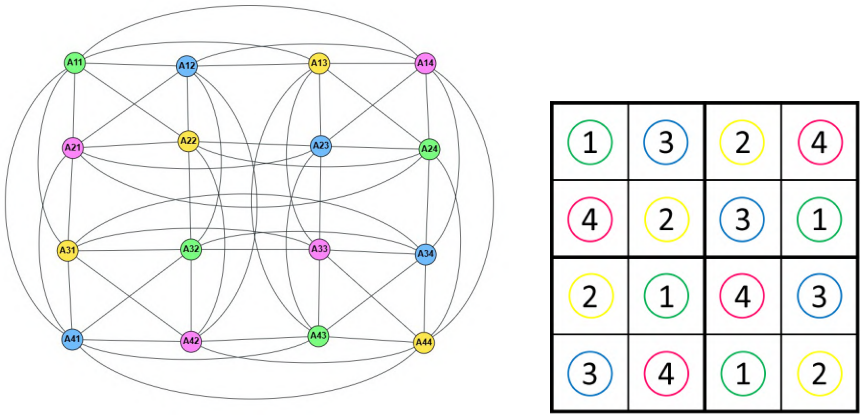


Figura 4.14: Subgrafo del sudoku - L_3

13. En el subgrafo de la Figura 4.14 ningún vértice está conectado con otro, por lo tanto podemos colorear aquellos vértices con el color 4 del vértice $A_{2,1}$. A partir de la $4-$coloración obtenida del grafo sudoku, procedemos a resolver el sudoku como se ilustra en la Figura 4.15b donde los vértices que tienen el color 1 se corresponden con el número 1 en el sudoku, así mismo con los otros conjuntos de vértices coloreados.



(a) Grafo Sudoku con pistas coloreado con RLF

(b) Sudoku resuelto con la 4-coloración

Figura 4.15: Solución del sudoku con pistas

Para los sudokus clásicos de 9×9 se puede hacer un proceso análogo, al crear el grafo que represente la cuadrícula y las reglas del juego, con un grafo $20-$regular de 81 vértices y 810 aristas, cuyos resultados con los algoritmos son semejantes al de 4×4.

4.2. Organizar horarios

Otra aplicación de la coloración de vértices en grafos consiste en la organización de horarios de exámenes, de clases, entre otros.

Nosotras decidimos ejemplificar la aplicación de la coloración de vértices en grafos con la organización de exámenes en una universidad.

En la Licenciatura de Matemáticas se deben programar los exámenes finales de varias asignaturas. Un total de 15 estudiantes presentan los exámenes de las asignaturas que cursaron durante el semestre. La universidad desea organizar los exámenes en franjas horarias de modo que ningún estudiante tenga dos exámenes al mismo tiempo. Las asignaturas para programar los exámenes y su letra representativa son:

- Aritmética (A)
- Elementos de Geometría (ED)
- Precálculo (PC)
- Sistemas Numéricos (SN)

- Cálculo Diferencial (CD)
- Geometría Plana (GP)
- Fundamentos de Programación (F)
- Álgebra Lineal (AL)
- Cálculo Integral (I)
- Geometría del Espacio (GE)
- Estadística (E)

Los estudiantes están inscritos a las siguientes asignaturas:

Estudiante	**Asignaturas**
1	A, PC, ED
2	SN, CD, GP, F
3	AL, I,GE, E
4	A, GD, GP, F
5	PC, SN, GP, F
6	ED, SN, CD, F
7	SN, I, GE, E
8	CD, AL, GE
9	GP, AL, I, E
10	F, AL, I, GE, E
11	A, AL, I, GE, E
12	PC, AL, GE, F
13	GP, F, AL ,SN, I
14	A, PC, ED, F, AL
15	SN, CD, GP, F, AL

Tabla 4.1: Materias inscritas

Las preguntas que nos queda por resolver para la organización de horarios son: ¿Cuál es el número mínimo de franjas horarias necesarias para programar todos los exámenes finales de modo que ningún estudiante tenga dos exámenes simultáneamente? y ¿Cómo se puede asignar cada examen a una franja horaria que satisfaga esta condición?

Para modelar este problema con un grafo, primero necesitamos determinar los conflictos entre asignaturas, esto es, listar las parejas de asignaturas en las que se encuentra inscrito de forma simultaneas, por lo menos, un estudiante.

Conflicto	Estudiantes
A - PC	E1, E14
A - ED	E1, E14
A - CD	E4
A - GP	E4
A - F	E4, E14
A - AL	E11, E14
A - I	E11
A - GE	E11
A - E	E11
PC - ED	E1, E14
PC - SN	E5
PC - GP	E5
PC - F	E5, E12, E14
PC - AL	E12, E14
PC - GE	E12
ED - SN	E6, E5
ED - CD	E6
ED - F	E6, E14
ED - AL	E14
SN - CD	E2, E6, E15
SN - GP	E2, E5, E13, E15
SN - F	E2, E5, E6, E13, E15
SN - AL	E13, E15
SN - I	E7, E13
SN - GE	E7
SN - E	E7
CD - GP	E2, E4, E15
CD - F	E2, E4, E6, E15
CD - AL	E8, E15
CD - GE	E8
GP - F	E2, E4, E5, E6, E13, E15
GP - AL	E9, E13, E5
GP - I	E9, E13
GP - E	E9
F - AL	E10, E12, E13, E14, E15
F - I	E10, E13
F - GE	E10, E12

Conflicto	Estudiantes
F - E	E10
ÁL - I	E3, E9, E10, E11, E13
ÁL - GE	E3, E8, E10, E11, E12
ÁL - E	E3, E9, E10, E11
I - GE	E3, E7, E10, E11
I - E	E3, E7, E9, E10, E11
GE - E	E3, E10, E11

Tabla 4.2: Conflictos entre asignaturas

Luego de definir los conflictos, vamos a establecer el grafo asociado a este problema, que llamaremos Grafo Horarios y representamos en la Figura 4.16, donde las asignaturas se corresponden con los vértices; y los conflictos entre exámenes determinan una arista. Como en el grafo cada vértice es una asignatura y cada arista representa que dos asignaturas comparten uno o más estudiantes, entonces dos vértices adyacentes representan dos asignaturas cuyos exámenes no se pueden programar en la misma franja horaria.

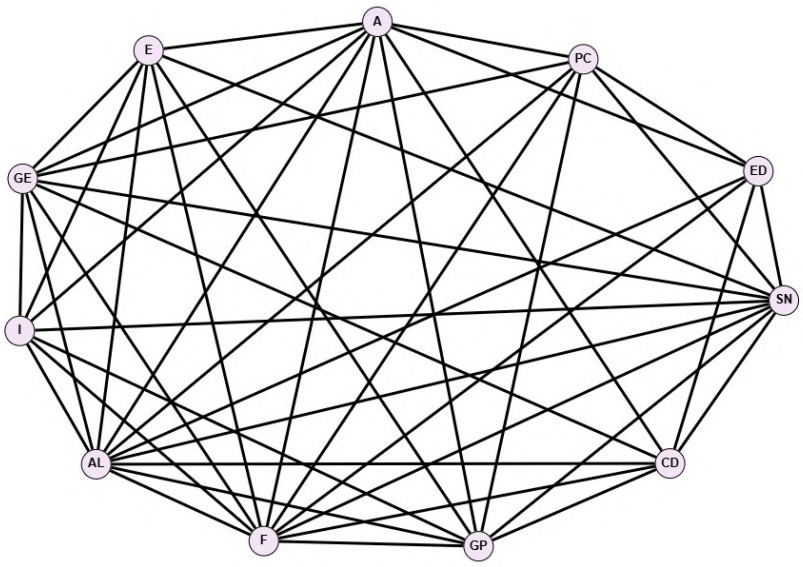


Figura 4.16: Grafo Horarios

La solución de la organización de los exámenes la podemos interpretar

como una coloración del grafo de la Figura 4.16, que utilice la menor cantidad de colores posibles, en la cual cada color representa una franja horaria.

La respuesta está asociada con la determinación del número cromático, sin embargo, esto no es fácil de determinar en todos los problemas. Desde el punto de vista teórico tenemos que la cota superior de $\Delta(G)+1$ del grafo es de 10 colores; sin embargo al aplicar los algoritmos estudiados en el capítulo 3, logramos mejorarla y obtener una $6-$coloración. Además, al determinar el número clique con la ayuda de python, obtenemos $\omega(G)=6$ lo que nos permite determinar que el número cromático del grafo es 6, gracias al Teorema 2.3. En la Figura 4.17 mostramos la $6-$coloración y el conjunto de vértices clique del grafo $\{AL, F, SN, E, GE, I\}$.

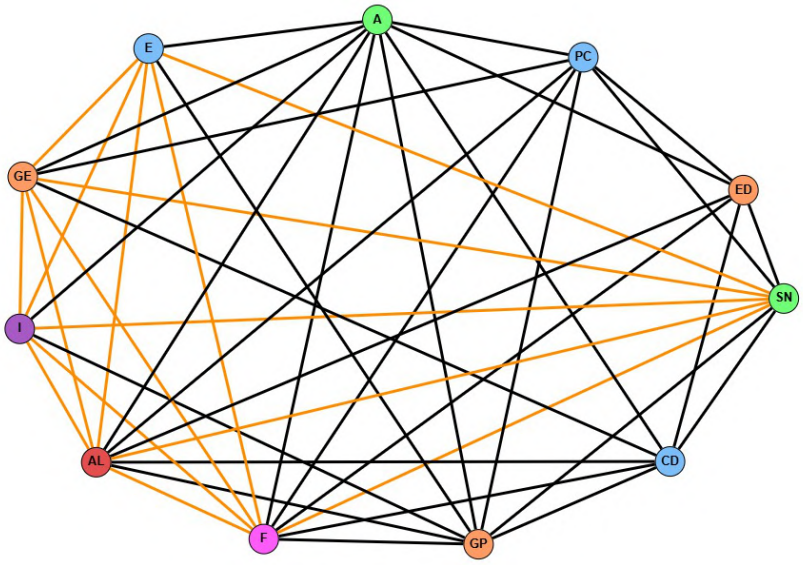


Figura 4.17: 6-coloración y conjunto clique del grafo horario

Con la coloración determinada, podemos establecer 6 franjas horarias para organizar los exámenes y las franjas horarias para cada examen, donde todos los vértices que están coloreados con el mismo color indican las asignaturas que van en la misma franja horaria. En particular, vamos a asociar el color naranja con la franja de 7 a 9 am; azul con 9 a 10 am; verde con 10am a 12m; morado con 12m a 2pm; rosado con 2pm a 4pm; y rojo con 4pm a 6pm. La información completa la consignamos en la Tabla 4.3.

Horario	**Asignaturas**	**Estudiantes**
7:00am - 9:00am	Espacio	E3, E7, E8, E10, E11, E12
	Elementos	E1, E6, E14
	Plana	E2, E4, E5, E9, E13, E15
9:00am - 10:00am	Precálculo	E1, E5, E12, E14
	Diferencial	E2, E4, E6, E8, E15
	Estadística	E3, E7, E9, E10, E11
10:00am - 12:00m	Aritmética	E1, E4, E11, E14
	Sistemas	E2, E5, E6, E7, E13, E15
12:00m - 2:00pm	Integral	E3, E7, E9, E10, E11, E13
2:00pm - 4:00pm	Fundamentos	E2, E4, E5, E6, E10, E12, E13, E14, E15
4:00pm - 6:00pm	Álgebra	E3, E8, E9, E10, E11, E12, E13, E14, E15

Tabla 4.3: Franjas horarias de los exámenes

A diferencia de lo que ocurre con el problema del sudoku, en estos problemas no tenemos asociado un único grafo, pues dependiendo del número de asignaturas y los conflictos existentes entre ellas, el grafo va a cambiar.

Capítulo 5

Conclusiones

La coloración de vértices en grafos es un problema que tiene su origen en la coloración de mapas, pero se extiende a grafos simples no dirigidos y consiste en asignar colores a cada vértice, de manera que, si dos vértices son adyacentes entonces deben tener colores diferentes; uno de los problemas es encontrar la coloración que nos da la menor cantidad de colores para colorear el grafo, dicho número es el número cromático.

Para abordar este problema, en el capítulo 1 establecimos nuestras definiciones y notaciones adoptadas en este trabajo a partir de la bibliografía consultada, dado que en la literatura no existen definiciones ni notaciones universalmente unificadas. Además, como la teoría de grafos es tan grande, escogimos lo necesario para estudiar la coloración de vértices en grafos.

Aunque el problema de encontrar el número cromático asociado a un grafo no tiene una solución teórica, en el capítulo 2, estudiamos algunos teoremas que nos permiten establecer cotas inferiores y superiores; además, para algunas familias de grafos establecimos de forma teórica el número cromático. También identificamos estrategias adicionales, como el estudio de los subgrafos asociados al grafo, para acotar el número cromático y los subgrafos completos maximales (cliques).

Otra estrategia para abordar el problema del número cromático fue el uso de algunos algoritmos, que, aunque al implementarlos no garantizan la obtención del número cromático, si proporcionan una mejor coloración o en el peor de los casos igual a la dada por la cota $\Delta(G)+1$. Entre los que revisamos en el capítulo 3, se encuentra el algoritmo Greedy, que consiste en fijar un orden cualquiera de vértices y asignar colores comparando la adyacencia con los vértices ya coloreados. Este algoritmo lo podemos aplicar a cada una de

las ordenaciones de los vértices posibles y al final seleccionar el resultado que involucra menos colores; sin embargo, dado un grafo con n vértices, tenemos $n!$ ordenes posibles; es decir, a medida que n crece, la estrategia descrita no es viable. Por esto, surgen otros algoritmos que pretenden mejorar la estrategia del algoritmo Greedy.

Entre estos, el algoritmo Largest First, fija un orden de vértices de mayor grado a menor grado y busca quitar los vértices más problemáticos, es decir, los que tienen mayor conexiones.

Por otro lado, el algoritmo Recursive Largest First, no fija una ordenación inicial, su objetivo es construir subconjuntos de vértices independientes en cada paso, en cada uno utilizando estrategias para buscar ordenaciones de vértices dinámicas, que dependen de subgrafos inducidos por los vértices no coloreados.

En cambio, el algoritmo Smallest Last tiene como objetivo dar una ordenación de vértices, primero quitando los vértices de menor grado en subgrafos inducidos por los vértices no eliminados, y agregando el eliminado al final de la lista que los ordena, para luego darle la vuelta a esta lista y así poder colorear, con el algoritmo Greedy, primero el vértice más problemático a nivel global y local, ya que tiene las mayores conexiones tanto en el grafo como en los subgrafos inducidos.

Además, realizamos un experimento computacional que nos permitió observar que: el algoritmo Greedy es bastante rápido en tiempo CPU, pero no es muy estable, ya que el orden en el que se consideran los vértices puede variar en cada ejecución, lo que implica la afectación del número de colores utilizados; en consecuencia, si aplicamos este algoritmo múltiples veces sobre un mismo grafo, no nos garantiza obtener siempre la misma cantidad de colores.

El algortimo RLF, nos presenta mejores coloraciones en los grafos, en especial, los grafos k-regulares; sin embargo este algoritmo es muy costoso computacionalmente. En cambio, el algortimo SL no siempre brinda la mejor coloración, ya que dependiendo de la estructura del grafo su desempeño varia; por ejemplo, en los grafos k-regulares este algoritmo presenta la peor coloración, además su tiempo CPU no es destacable en comparación con los presentados por los algoritmos Greedy y LF.

El algoritmo LF, es mejor si deseamos un menor costo computacional que el RLF y mejor coloración que el Greedy.

Dependiendo de las restricciones del usuario en términos de tiempo y recursos, un algoritmo se puede adaptar mejor que otro a sus necesidades;

en particular, un algoritmo que tenga un mayor tiempo de ejecución implica invertir más dinero. Por ello, si priorizamos obtener resultados rápidamente, aún sacrificando la calidad de la solución, es poco probable que se elija el algoritmo RLF.

Sin embargo, si podemos caracterizar cuál algoritmo es más conveniente de usar, dependiendo de la estructura del grafo y lo que el usuario desee, algunas son: menor costo computacional, según el tipo de grafo o menor cantidad de colores empleados, entre otros requerimientos.

Por otro lado, estudiamos algunas aplicaciones de la coloración de vértices en grafos, como la solución de sudokus y la organización de horarios. Encontramos que en el caso de los sudokus, usamos un único grafo que representa a todos los posibles juegos de sudokus en una cuadricula en blanco, el objetivo es asignar colores a los vértices que representan las casillas del juego de manera que los colores asignados se traducen con un número que da solución al sudoku. En nuestras exploración concluimos que el algoritmo RLF es el más adecuado para resolver estos problemas, por lo que el grafo de un sudoku es k-regular. Ademas nos surgió la duda de cómo se haría la solución de un sudoku que tiene algunos números ya escritos es decir pistas; encontramos que el algoritmo RLF sigue su esencia con la diferencia que en los vértices que representan las pistas ya se tiene un color asignado.

En la organización de horarios, la coloración de vértices apunta a dos objetivos, uno relacionado con la búsqueda de la menor cantidad de franjas horarias, y el otro sobre seleccionar qué ubicar en cada franja horaria donde cada color se relaciona con una franja horaria. De esta aplicación, resaltamos que cada grafo es diferente para cada tipo de problema, por lo tanto, la selección de un algoritmo adecuado para esta aplicación depende de la estructura del grafo.

Para concluir este trabajo de grado, queremos expresar nuestras reflexiones personales. En relación con el aporte a nuestra formación, destacamos la importancia de haber podido plasmar nuestras ideas a través de la escritura, ya que uno de nuestros objetivos personales es que cualquier persona tenga la oportunidad de leer este documento, incluso sin haber tenido un estudio previo sobre el tema. Además de brindarnos la oportunidad de experimentar por primera vez un proceso de investigación, en medio del desarrollo de este adquirimos habilidades en diferentes ámbitos como la programación, el uso de LaTex, el uso de las herramientas tecnológicas que tenemos hoy en día a nuestra disposición y el estudio de una rama de las matemáticas, que no siempre se aborda en la carrera.

Las futuras investigaciones que se podrían seguir con esta línea de trabajo son: ¿Cómo construir subconjuntos de vértices independientes?, ¿Cómo es la coloración de vértices en grafos que no son simples?, ¿Cuales son las variantes de los algoritmos u otros algoritmos escritos en la literatura?, ¿Cómo es la coloración de vértices de los grafos en las variantes de los sudokus o la clasificación de sudokus (principiante, intermedio, experto)?, ¿Qué pasaría si subimos las programaciones a una Inteligencia Artificial para que seleccione el mejor algoritmo para un grafo que inserte cualquier usuario?. Ademas, nos gustaría estudiar sobre los grafos aleatorios.

Bibliografía

Apple, K., & Haken, W. (1976). Every Planar Map is Four Colorable. *Bulletin of the American Mathematical Society*, *82*(5), 711-712.

Cerón, S. I., Martínez, W. A., & Palechor, L. L. (2005). *Fundamentos de Coloreado de Grafos y Teoría de Ramsey* [Tesis de pregrado]. Universidad del Cauca. http://repositorio.unicauca.edu.co:8080/xmlui/handle/123456789/7883

Chartrand, G., & Zhang, P. (2012). *A First Course in Graph Theory*. Dover Publications Inc.

Graphonline. (2015 – 2026). *Graph Online*. https://graphonline.top/es/

Gross, J., Yellen, J., & Anderson, M. (2019). *Graph Theory and Its Applications, Third Edition*. Taylor Francis Group.

Guerequeta, R., & Vallecillo, A. (2000). *Técnicas de Diseño de Algoritmos*. Universidad de Málaga.

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring Network Structure, Dynamics, and Function using NetworkX. En G. Varoquaux, T. Vaught & J. Millman (Eds.), *Proceedings of the 7th Python in Science Conference* (pp. 11-15).

Leighton, F. T. (1979). A Graph Coloring Algorithm for Large Scheduling Problems. *Journal of Research of the National Bureau of Standards*, *84*(6), 489-506.

Matula, D., Marble, G., & Isaacson, J. (1972). Graph Coloring Algorithms. *Graph Theory and Computing*, 104-122.

NetworkX Developers. (2025). *NetworkX Documentation*. https://networkx.org/documentation/stable/

OpenAI. (2026). *ChatGPT*. https://chatgpt.com

Ramírez, J. (2001). *Extensiones del Problema de Coloración de Grafos* [Tesis de doctorado]. Universidad Complutense de Madrid. https://docta.u

cm.es/rest/api/core/bitstreams/841c12a7-efca-4d9f-916b-c4d3a45746fe/content

Welsh, D., & Powell, M. B. (1979). An Upper Bound for the Chromatic Number of a Graph and Its Application to Timetabling Problems. *The Computer Journal*, *10*, 85-86.

West, D. (2005). *Introduction to Graph Theory, Second Edition*. Pearson Education Inc.