BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

Algoritmo para determinar la *K-Coloración* de un grafo

TESIS

QUE PARA OBTENER EL TITULO DE

INGENIERO EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA

OLGA LIDIA GARCIA LIMON

ASESOR

DR. PEDRO BELLO LÓPEZ

Noviembre, 2023

Agradecimientos

Quiero agradecer a Dios y al Universo por permitirme llegar a este punto de mi vida y poder culminar este ciclo, por las grandes oportunidades que me ha puesto en mi camino, por lograr cada uno de mis metas.

Agradezco a mi familia por apoyarme en cada uno de mis metas y mis sueños en especial a mi madre: Francisca Limón Ambrosio la que está en cada momento y me ha dado su apoyo incondicional. Y siempre me da lo mejor de ella.

A mis hermanos María Eugenia, Javier, Andrés, Leonardo que han estado conmigo en todo este proceso y que han sido parte de mi crecimiento,

Agradezco a mi abuela Teresa Ambrosio la cual ha sido una segunda madre y siempre cuido de mí, a mi tía Yolanda Robles por sus consejos de que siempre se puede lograr las cosas si uno se lo propone

Agradezco a mis amigos Elena Reyes, Jaqueline González, Adrián Maceda. los cuales siempre motivaron a seguir adelante y dándome sus mejores consejos, habilidades y conocimientos.

Agradezco a mi asesor al Dr. Pedro Bello López por su paciencia y dedicación, el motivarme y creer en mí, el darme la oportunidad en pertenecer al grupo de proyectos de investigación

Agradecimiento especial a la Vicerrectoría de Investigación y Estudios de Posgrado (VIEP) por el apoyo económico brindado para culminar este trabajo de tesis dentro del proyecto VIEP 00238. Aplicando Ciencias de Datos para analizar cadenas genómicas 2023 del cuerpo académico PRODEP Algoritmos Combinatorios y Aprendizaje.

Contenido

Resumen 4
Capítulo 1. Introducción....6
1.1 Trabajos relacionados 8
1.1 Objetivos Generales y Específicos del Proyecto 11
1.3 Metodología.....11
Capítulo 2. Coloreo de grafos 13
2.1 Teoría de grafos 13
2.2 Algoritmos de coloreo de grafos....16
2.3 Algoritmo de coloración secuencial básico....17
2.4 Algoritmo de coloración de Welsh y Powell 18
2.5 Algoritmo de coloración de Matula, Marble, Isaacson....19
2.6 Algoritmo de coloración de Brelaz....20
2.7 Algoritmo de independencia MCI (máximo conjunto independiente)....21
2.8 El coloreo de grafos y los conjuntos independientes 22
2.9 La conjetura de los cuatro colores....23
2.10 Número Cromático $\chi(G)$....24
Capítulo 3. Algoritmo de coloración implementado....26
3.1 Pseudocódigo del algoritmo de coloreo 26
3.2 Prueba de escritorio del algoritmo 27
3.3 Análisis del Algoritmo propuesto....30
Capítulo 4. Sistema de coloreo de grafos 33
4.1. Herramientas y Lenguaje de desarrollo....33
4.2 Formato de los archivos de entrada 36
4.3 Pantallas y pruebas del sistema....37
4.4 Ejemplos de coloreo de grafos....40
4.5 Estructuras de datos del sistema 43
Conclusiones 47
Bibliografía 48

Resumen

En la teoría de grafos, la coloración de grafos es un caso especial de etiquetado de grafos; es una asignación de etiquetas llamadas colores a los nodos o vértices del grafo. Es decir, una coloración (o color del grafo) de los vértices de un grafo es una asignación tal que ningún vértice adyacente comparta el mismo color.

Si en la coloración se usan *k* (*k*=1,2, …, *n*) colores distintos diremos que es una *k-coloración*. Una coloración siempre es posible, dado que podemos asignar a cada vértice del grafo un color diferente si fuera necesario, por ejemplo, para grafos completos.

Si existe una *k-coloración* de *G* se dice que el grafo *G* es *k*-coloreable. El valor mínimo *k* (de colores) para el que un grafo *G* es *k*-coloreable se denomina número cromático de *G*, y se designa por *χ(G).* De esta forma la coloración de los vértices se basa en encontrar grupos de vértices en el grafo que no sean adyacentes entre sí para así poder asignarles el mismo color.

En este proyecto de tesis se propone diseñar e implementar un algoritmo para determinar con cuantos colores se puede colorear un grafo. El sistema debe estar en línea con el fin de que personas interesadas puedan realizar pruebas del *k*-coloreo de grafos.

El trabajo de tesis está organizado de la siguiente forma:

- En el primer capítulo se realiza la introducción del trabajo de tesis donde se expone el problema de coloreo de grafo y la teoría asociada a este tipo de problemas, así como los principales trabajos que han dado propuestas de algoritmos para el problema de coloración de grafos y por tanto obtener el numero cromático de un grafo.

- En el segundo capítulo se describe la teoría general de grafos y las diversas propuestas de algoritmos de coloreo de grafos donde se describe la estrategia utilizada en cada caso.

- En el tercer capítulo se describe la propuesta del algoritmo que se usara en este trabajo de tesis para el coloreo de grafos y para obtener el

numero cromático de un grafo. Se expondrán también diversos ejemplos de coloreo de un grafo.

- En el cuarto capítulo se presenta la implementación de la propuesta algorítmica realizada y se describe el desarrollo de un prototipo funcional en web para realizar pruebas de coloreo y se incluyen diversas instancias de grafos clásicos y su coloración para obtener el numero cromático.

- Finalmente se presentan las conclusiones del trabajo de tesis, los aportes y las posibles mejoras al sistema web que permite colorear un grafo para obtener el numero cromático del mismo.

Capítulo 1. Introducción

El origen del estudio de los grafos se remonta a mediados de 1736, cuando el matemático suizo Leonhard Euler escribió un artículo científico en el cual se dedicaba a analizar el problema de los puentes de Konisberg. Este problema consiste en el estudio de una hipótesis que consistía en recorrer la ciudad atravesando por todos y cada uno de los siete puentes de la ciudad y volver al punto de partida sin pasar más de una vez por un mismo puente [1].

Leonhard Euler planteó este problema desde los orígenes de la teoría de grafos. Después de su análisis Euler llegó a la conclusión. para este problema planteado desde la teoría de grafos, es que debía de pasar al menos 2 veces por alguno de los puntos para hacer todo el recorrido deseado. Con este experimento se inició todo el desarrollo de una teoría que hoy forma parte fundamental de la computación, la denominada teoría de grafos [1].

Existen problemas muy complicados para los que:

- No sabemos cómo encontrar su solución óptima, o bien.
- Conocemos el algoritmo que permite encontrar la solución óptima, pero requiere una cantidad excesiva de tiempo.

Cuando revisamos los algoritmos heurísticos, nos referimos a un procedimiento o estrategia que puede producir una buena solución el problema planteado, incluso se puede dar el caso de obtener una solución óptima si somos afortunados, pero también puede darse el caso que le algoritmo heurístico no produzca ninguna solución, y en algunos casos el algoritmo heurístico puede generar una solución no sea precisamente óptima. Una forma en la que son muy usados es con los algoritmos voraces.

En estos casos utilizaremos algoritmos heurísticos:

- Proporcionan soluciones más o menos próximas a la óptima, en un tiempo de ejecución polinómico.
- Permiten resolver ejemplares del problema de tamaños que serían inabordables utilizando el algoritmo óptimo.
- Muchos de los algoritmos heurísticos y aproximados son algoritmos voraces.

Una ventaja adicional de los algoritmos heurísticos es el tiempo de ejecución donde podremos probar instancias grandes de prueba y obtener buenas soluciones.

Colorear un grafo *G* es una asignación de colores a los vértices de *G*, a cada vértice le corresponde un color, de forma que a los vértices adyacentes del nodo coloreado reciban colores distintos. Podemos decir para simplificar, que la coloración de los vértices de un grafo se basa en encontrar grupos de vértices que no sean adyacentes entre sí (ver Figura 1.1). El número mínimo de colores necesarios para colorear el grafo representa el número cromático del grafo.

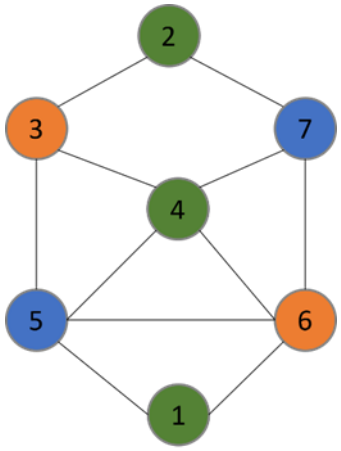

Figura 1.1 Grafo coloreado con 3 colores, numero cromático=3.

El coloreo de grafos es un concepto fundamental en la teoría de grafos y tiene una amplia importancia en diversas aplicaciones y campos, entre las más importantes podemos mencionar los siguientes puntos:

1. Planificación de horarios: En la planificación de horarios escolares, la asignación de aulas, profesores y cursos se puede modelar como un problema de coloreo de grafos. Garantizar que ningún par de cursos con horarios superpuestos se imparten en la misma aula al mismo tiempo es esencial para la eficiencia y la gestión adecuada de los recursos.

2. Asignación de recursos: En la asignación de recursos, como la asignación de frecuencias de radio, canales de comunicación, asignación de trabajadores a tareas, etc., el coloreo de grafos puede utilizarse para garantizar que los recursos compartidos no se utilicen simultáneamente por entidades incompatibles.

3. Optimización de redes: En la optimización de redes, como la asignación de colores a las regiones de un mapa para que las regiones vecinas tengan colores diferentes, el coloreo de grafos se utiliza para minimizar la interferencia y asegurar un flujo eficiente en la red.

4. Resolución de problemas NP-duros: El coloreo de grafos es un problema NP-duro, lo que significa que es computacionalmente difícil encontrar una solución óptima en tiempo polinómico para casos generales. Esto lo convierte en un problema de referencia para la investigación en algoritmos y complejidad computacional.

5. Grafos en la vida cotidiana: Muchos problemas cotidianos pueden ser representados y resueltos como problemas de coloreo de grafos. Por ejemplo, la programación de asignación de tareas domésticas o la organización de eventos con restricciones de disponibilidad de personas son ejemplos de situaciones en las que el coloreo de grafos puede ser útil.

6. Teoría de la computación y algoritmos: El coloreo de grafos se utiliza en algoritmos de optimización, en la resolución de problemas de programación lineal y en la toma de decisiones en la teoría de la computación. También tiene aplicaciones en la demostración de propiedades en la teoría de la complejidad computacional.

Así, el coloreo de grafos es una herramienta poderosa y versátil que se utiliza en una variedad de aplicaciones prácticas y es un problema importante en la teoría de grafos. Su resolución eficiente es esencial para abordar problemas de asignación y programación en la vida real, así como para avanzar en la investigación en algoritmos y complejidad.

1.1 Trabajos relacionados

Desde un punto de vista práctico, los grafos permiten estudiar las interrelaciones entre unidades que interactúan unas con otras. Por ejemplo, una red de computadoras puede representarse y estudiarse mediante un grafo, en el cual los vértices representan terminales y las aristas representan conexiones (las cuales, a su vez, pueden ser cables o conexiones inalámbricas).

Existen diversas herramientas que me permiten visualizar los grafos como son:

- Gephi
- Graphviz
- Sigma
- Cytoscape
- Jung
- Igraph
- Linkurious

La descripción de cada herramienta se encuentra en https://www.grapheverywhere.com/herramientas-de-visualizacion-de-grafos/ (recuperado el 6/09/2022).

Existen algunos esfuerzos para colorear grafos como en http://pedrogonzalezruiz.net/coloreado/coloreado.html (recuperado 6/09/202) donde se dispone de un programa para colorear cualquier grafo y es muy útil para resolver muchos problemas. Se presentan varios ejemplos para que pueda entenderse. Debe ejecutarse en un emulador de terminal o dentro de Emacs bajo el sistema operativo Linux. Está hecho en stklos, una implementación de Scheme, un dialecto de Lisp, dispone de un apartado dentro de éste artículo, a modo de pequeño tutorial, para comprender la sintaxis y se-mántica de ésta versión de Lisp. En https://www.makingcode.dev/2015/08/algoritmos-de-coloracion-de-grafos.html, (recuperado el 7/09/2022) se presenta otro esfuerzo individual de un programa para colorear un grafo, se explica el código para que los usuarios lo implementen.

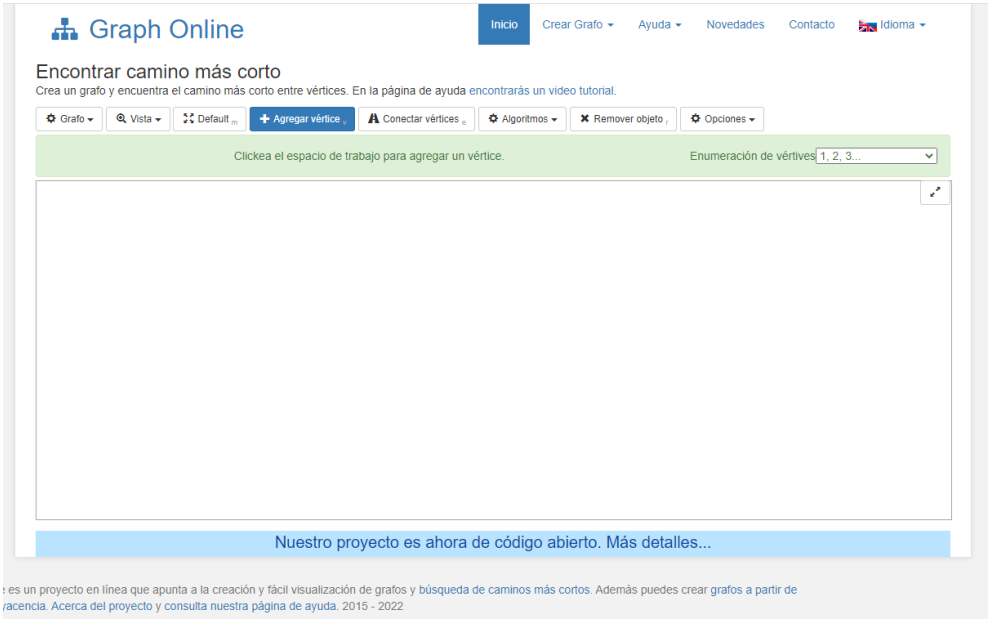

Figura 1.2. Pantalla de inicio de Graph Online

Graph Online disponible en https://graphonline.ru/es/ (consultado el 7/09/2022) es un proyecto de Rumania en línea que apunta a la creación y fácil visualización de grafos y búsqueda de caminos más cortos. Además, puedes crear grafos a partir de matrices de adyacencia (Figura 1.2).

Grafos: disponible en http://personales.upv.es/arodrigu/grafos/ (recuperado el 8/09/2022) es un Software para la construcción, edición y análisis de grafos. Este software pretende ser de utilidad para la docencia y el aprendizaje de la y otras disciplinas relacionadas como la ingeniería de organización industrial, la logística y el transporte, investigación operativa, el diseño de redes, etc. Grafos se puede usar perfectamente para el modelado y resolución de problemas reales de cierto tamaño y complejidad.

Existen diferentes esfuerzos por investigadores para contribuir al desarrollo de algoritmos para el coloreo de grafos, sin embargo, generalmente generan los algoritmos o programas, pero no hay sistemas en línea que permitan realizar la *k*-coloración, el sistema que se aproxima al trabajo propuesto es el sistema Graph Online que es desarrollado por una gran comunidad de programadores. Nuestra propuesta va encaminada con fines educativos y de investigación para que cualquier usuario de internet pueda probar el coloreo de grafos.

1.1 Objetivos Generales y Específicos del Proyecto

Diseñar un algoritmo y su implementación computacional para determinar la k-coloración de un grafo conexo. El sistema computacional deberá ser una aplicación web para que esté disponible a personas interesadas en el tema. El sistema debe determinar el número de colores necesarios para colorear del grafo, además que mostrara de forma gráfica dicho grafo ya coloreado.

Objetivos Específicos.

- Diseñar el algoritmo que permita determinar si un grafo conexo se puede colorear con k-colores.
- Diseñar el sistema en web para integrar el algoritmo k-coloreable propuesto.
- Implementar el algoritmo k-coloreable.
- Implementar el sistema web para la lectura del grafo, la interfaz con el algoritmo y dibujar el algoritmo coloreado.
- Realizar instancias de prueba para grafos de diferentes formas y tamaños.

1.3 Metodología.

Los pasos necesarios para el desarrollo del trabajo de tesis se describen a continuación:

- Análisis del problema de coloreo de grafos y revisar las diversas propuestas algorítmicas para el coloreo de grafos [7, 8, 9]. Se realizó un análisis de diversas propuestas algorítmicas con el fin de elegir el adecuado para los propósitos del problema planteado.
- Diseño del algoritmo propuesto k-coloreable de un grafo. Se Diseño el algoritmo de coloración de grafos y se probó con diferentes tipos de grafos.
- Diseño del sistema web [10, 11]. Se realizó el diseño del sistema para ser implementado, se consideraron las opciones que debe tener el sistema

incluyendo requerimientos funcionales (opciones del sistema) y no funcionales como el lenguaje de programación y las herramientas necesarias para crear el prototipo funcional.

- Implementación del algoritmo k-coloreable [12]. Se implemento el prototipo funcional sobre un servidor web y se realizaron las pruebas funcionales del sistema.

- Integración y pruebas de la aplicación web y el algoritmo k-coloreable. Se creo un conjunto de ejemplos funcionales donde el usuario puede seleccionar y realizar las pruebas necesarias.

Capítulo 2. Coloreo de grafos

Para el desarrollo del trabajo fue necesario revisar conceptos teóricos [14] que sirven como base para plantear el algoritmo de colorear.

2.1 Teoría de grafos

En el área de ciencias de la computación, así como en matemáticas, la teoría de grafos estudia se encarga del estudio de los denominados grafos. Un grafo es considerado un conjunto, no vacío, de elementos u objetos llamados vértices (o también nodos) y un conjunto de líneas de conexión entre los nodos o vértices, llamados aristas (o arcos) que pueden ser dirigidos o no dirigidos [2, 3, 4].

Típicamente, un grafo (Figura 2.1) se representa mediante una serie de puntos (los nodos) conectados por líneas (las aristas).

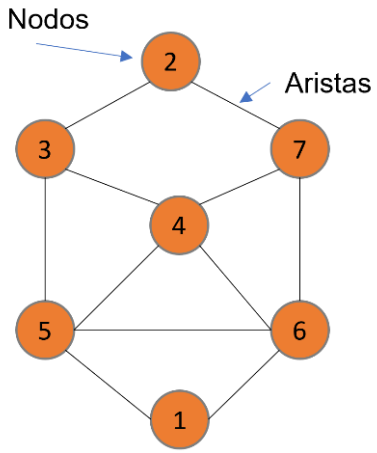

Figura 2.1. Grafo básico

Vértice

Los nodos o vértices de un grafo constituyen uno de los dos elementos principales que forman un grafo *G*. La unidad fundamental de la que están compuestos los grafos son los vértices.

En los grafos, los vértices son tratados también como unidades indivisibles y sin propiedades, aunque cada vértice puede contener cualquier tipo de información,

además pueden tener estructuras adicionales dependiendo del uso o aplicación del grafo al que representa la solución de un problema.

Arista

Las aristas también llamados arcos, junto con los vértices, forman los elementos principales de un grafo. Se definen como las uniones entre nodos o vértices. Normalmente las aristas representan relaciones entre dos o más vértices, además pueden representar orden, la vecindad o la herencia.

Las aristas además de unir dos vértices suelen tener una dirección establecida (dígrafos).

Existen algunos tipos de grafos para aplicaciones específicas donde las aristas llevan asociadas un número, un peso o un valor, que indica una información asociada a ambos vértices. Este valor puede indicar un coste o ser una representación del trabajo necesario para recorrer el camino de un vértice a otro vértice. En grafos dirigidos puede darse el caso en que el camino o trayectoria de *A* hacia *B* tenga un valor o coste diferente que la trayectoria de *B* hacia *A*.

Finalmente, la relación entre vértices y aristas no tiene por qué producirse, puede darse el caso de que los nodos o vértices sean aislados y no tengan ninguna relación con ningún otro nodo en el grafo.

Grado de un grafo

El grado de un grafo se define como el máximo grado de sus nodos, es decir, se obtiene el grado de cada nodo como el número de aristas que inciden en dicho nodo y se obtiene el nodo de mayor grado como el nodo del grafo.

Un grafo se puede representar de varias maneras en función de su complejidad y de la necesidad de utilizarlo en algoritmos y aplicaciones específicas. Aquí hay algunas de las representaciones más comunes de un grafo:

Representación de un grafo

Existen diversas formas de representar los datos de un grafo, es decir representar los nodos y las aristas. A continuación, exponemos algunas de las

representaciones comunes de grafos, y la elección de la representación depende de la aplicación y de los algoritmos que se van a utilizar. Cada una tiene sus ventajas y desventajas en términos de eficiencia y facilidad de uso en diferentes contextos.

1. Lista de adyacencia: En esta representación, se utiliza una lista (o matriz) para cada nodo del grafo, donde se enumeran todos los nodos a los que está conectado. En un grafo dirigido, cada lista de adyacencia contiene los nodos a los que se puede llegar desde el nodo correspondiente. En un grafo no dirigido, la relación es simétrica, por lo que, si el nodo A está en la lista de adyacencia de B, entonces B también está en la lista de adyacencia de A. Esta representación es eficiente para grafos dispersos y suele utilizarse en implementaciones de algoritmos.

2. Matriz de adyacencia: Esta representación es la más general y utilizada ya que se utiliza una matriz bidimensional donde las filas y las columnas representan nodos, y cada celda indica si hay una arista entre los nodos correspondientes. Si es un grafo no dirigido y no ponderado, la matriz suele ser simétrica y solo contiene valores binarios (1 si hay una arista, 0 si no). En un grafo dirigido, la matriz puede ser asimétrica y también puede contener pesos si el grafo es ponderado.

3. Lista de aristas: En esta representación, se almacenan todas las aristas del grafo en una lista. Cada arista se representa como un par ordenado de nodos (o una tupla en el caso de grafos ponderados). Esta representación es útil cuando se necesita acceder a todas las aristas del grafo.

4. Matriz de incidencia: Esta representación se utiliza principalmente para grafos dirigidos. Se crea una matriz donde las filas representan nodos y las columnas representan aristas. Cada celda indica si un nodo es el punto inicial (con valor -1), el punto final (con valor 1) o no está involucrado (con valor 0) en una arista particular.

2.2 Algoritmos de coloreo de grafos

Los algoritmos de coloreo de grafos se pueden clasificar en diferentes categorías según diversos criterios y enfoques [15], por ejemplo:

1. Basados en Ordenación de Vértices: Pueden ordenar los vértices según su grado, grado de saturación, o mediante otras heurísticas, tales como: Algoritmos que colorean los vértices en un orden específico, como el algoritmo de Welsh-Powell y el algoritmo de Matula, Marble y Isaacson (MMI).

2. Basados en Grado de Saturación: Algoritmos que utilizan el concepto de grado de saturación (DSATUR) para seleccionar el siguiente vértice a colorear, como el algoritmo de Brelaz.

3. Basados en Búsqueda Exhaustiva: Este tipo de algoritmos pueden encontrar la solución óptima, pero pueden ser computacionalmente muy costosos para grafos grandes generalmente se basan en backtracking que prueban todas las combinaciones posibles de colores para los vértices hasta encontrar una solución válida.

4. Basados en Heurísticas Evolutivas: Algoritmos inspirados en la evolución biológica, como algoritmos genéticos, que buscan mejorar las soluciones mediante operadores como selección, cruce y mutación.

5. Basados en Optimización Combinatoria: Algoritmos que utilizan técnicas de programación lineal, relajación y otros enfoques de optimización para encontrar soluciones aproximadas.

6. Basados en Descomposición: Algoritmos que descomponen el grafo en subgrafos más pequeños o estructuras particulares y colorean cada parte por separado.

7. Basados en Métodos de Aproximación: Algoritmos que ofrecen garantías de aproximación, es decir, encuentran soluciones que están dentro de un factor constante del número cromático óptimo.

8. Basados en Coloreo de Aristas: Algoritmos que se enfocan en colorear aristas en lugar de vértices, reduciendo así el problema de coloración de vértices a uno de aristas.

9. Basados en Heurísticas Adicionales: Algoritmos que utilizan heurísticas específicas para tipos particulares de grafos o estructuras, buscando explotar propiedades especiales.

Existen diversas propuestas en particular extenderemos un poco el estudio de los siguientes algoritmos heurísticos.

2.3 Algoritmo de coloración secuencial básico

El orden en que se colorean los vértices se decide antes de que se empiece a colorearlos. Dada una ordenación de los vértices del grafo, los algoritmos secuenciales asignan el mínimo color posible al siguiente vértice. Es decir, si queremos colorear *v*, teniendo ordenados numéricamente los colores, asignamos a *v* el color más pequeño que no aparece entre los asignados a los vecinos de *v* ya coloreados [4, 15].

El algoritmo de coloreo secuencial básico es uno de los métodos más simples para asignar colores a los vértices de un grafo de manera que no haya dos vértices adyacentes con el mismo color. Los pasos básicos del algoritmo son:

1. Ordenar los Vértices: Elige un orden en el que se colorearán los vértices del grafo. Este orden puede influir en el número de colores necesarios para colorear el grafo. Un enfoque común es ordenar los vértices en función de su grado (número de aristas conectadas al vértice).

2. Coloreo Secuencial: Comienza a colorear los vértices en el orden determinado. Para cada vértice, asigna el color más bajo posible que no esté siendo usado por ninguno de sus vecinos previamente coloreados. Es decir, asigna el color que no esté presente en los vértices adyacentes ya coloreados.

3. Revisión de Colores: A medida que colores los vértices, es posible que necesites revisar los colores de los vértices previamente coloreados para asegurarte de que no haya conflictos. Si al colorear un vértice te das cuenta de que no hay colores disponibles que cumplan con la restricción de no adyacencia, deberás agregar un nuevo color a la paleta de colores y asignárselo al vértice en cuestión.

4. Finalización: Continúa coloreando los vértices en el orden definido hasta que todos los vértices estén coloreados.

El resultado puede depender en gran medida del orden en que se eligen los vértices. En algunos casos, el número cromático obtenido puede ser mayor que el mínimo posible.

2.4 Algoritmo de coloración de Welsh y Powell

Es una variante del algoritmo de coloración secuencial básico, también conocida como "Primero el de mayor grado". Es debida a Welsh y Powell, y en este algoritmo, los vértices se ordenan inicialmente de acuerdo con sus grados. Es decir, ordenamos de forma que $d(v1) >= d(v2) >= ... >= d(v_n)$. Los pasos básicos del algoritmo. El algoritmo de coloreo de Welsh-Powell es un método heurístico simple y eficiente para colorear vértices de un grafo. Aunque no garantiza encontrar el número cromático mínimo, a menudo produce buenas soluciones en un tiempo razonable. El algoritmo [4, 15] se basa en el siguiente enfoque:

1. Ordenar los Vértices: Primero, se ordenan los vértices en orden decreciente según su grado (número de aristas conectadas). Esto significa que se comienza coloreando los vértices con mayor grado, lo que tiende a reducir la posibilidad de conflictos de colores.

2. Coloreo Secuencial: Luego, se procede a colorear los vértices en el orden determinado. Para cada vértice, se asigna el color más bajo posible que no esté siendo usado por ninguno de sus vecinos ya coloreados.

3. Revisión de Colores: Al igual que en el algoritmo de coloreo secuencial básico, es posible que sea necesario revisar y ajustar los colores de vértices previamente coloreados a medida que se continúa coloreando los vértices restantes.

4. Finalización: Se continúa coloreando los vértices en el orden definido hasta que todos los vértices estén coloreados.

El algoritmo de Welsh-Powell es efectivo para muchos grafos y tiende a producir resultados razonables en un tiempo corto. Sin embargo, es importante tener en cuenta que no garantiza encontrar el número cromático mínimo en todos los casos, ya que su efectividad depende del orden inicial de los vértices.

2.5 Algoritmo de coloración de Matula, Marble, Isaacson

Es una variante del algoritmo de coloración secuencial básico, también conocida como "El de menor grado el último". Se debe a Marble, Matula e Isaacson, y en este algoritmo, los vértices se ordenan en orden inverso. Primero se elige v_n como el vértice de menor grado, luego se elige v_{n-1} como el vértice de menor grado en $G-\{v_n\}$, y así se continúa recursivamente, examinando los vértices de menor grado y eliminándolos del grafo [4, 15]. Los pasos básicos del algoritmo son:

1.Ordenación de Vértices: El algoritmo comienza ordenando los vértices en orden creciente de sus grados (número de aristas adyacentes). Esto significa que se comienza coloreando los vértices con menor grado primero.

2.Selección de Vértices: Luego, se elige un vértice no coloreado con el grado más bajo posible. Esto significa que el algoritmo selecciona un vértice que tiene menos conexiones, lo que reduce las posibilidades de conflictos de colores.

3. Coloreo del Vértice Seleccionado: Una vez que se ha seleccionado un vértice, se le asigna el color más bajo posible que no esté siendo usado por ninguno de sus vecinos ya coloreados.

4. Revisión de Colores: Al igual que en otros algoritmos de coloreo, es posible que sea necesario revisar y ajustar los colores de vértices previamente coloreados a medida que se continúa coloreando los vértices.

5. Finalización: El proceso de selección, coloreo y revisión continúa hasta que todos los vértices estén coloreados.

El algoritmo MMI se basa en la idea de que comenzar por vértices con grados más bajos reduce la posibilidad de conflictos de colores, ya que estos vértices tienen menos vecinos. Si bien el algoritmo no garantiza encontrar el número cromático mínimo y su efectividad puede depender del grafo específico y del orden de vértices seleccionado, en muchos casos produce resultados aceptables en un tiempo razonable.

2.6 Algoritmo de coloración de Brelaz

En los algoritmos anteriores, la única información que se tiene de cada vértice es su grado. En el algoritmo de Brelaz, tendremos en cuenta también la suma de los grados de los vecinos de cada vértice y los colores ya asignados a dichos vecinos. Definimos el grado de color de un vértice v como el número de colores usados en los vecinos de *v*. El orden en que iremos coloreando vértices depende del grado y del grado de color [4, 15]. El algoritmo de coloración de Brelaz se basa en la siguiente idea principal:

1. Ordenación de Vértices: Se comienza ordenando los vértices en orden decreciente de sus grados (número de aristas adyacentes). Esto significa que se comienza coloreando los vértices con mayor grado primero.

2. Asignación de Grados de Saturación: Para cada vértice, se calcula su grado de saturación, que se define como el número de colores diferentes asignados a sus vecinos. Es decir, cuántos colores diferentes se están utilizando para colorear los vértices adyacentes. Luego, se coloca a cada vértice en una lista ordenada según su grado de saturación.

3. Selección y Coloreo del Vértice: Se selecciona el vértice de mayor grado de saturación de la lista y se le asigna el color más bajo posible que no esté siendo usado por sus vecinos ya coloreados.

4. Actualización de Grados de Saturación: Después de colorear un vértice, se actualizan los grados de saturación de sus vecinos, ya que se ha agregado un color nuevo al conjunto de colores utilizados. Luego, los vértices se reordenan en función de sus nuevos grados de saturación.

5. Repetición: Los pasos 3 y 4 se repiten hasta que todos los vértices estén coloreados.

El algoritmo de Brelaz combina la elección de vértices con mayor grado de saturación con la asignación cuidadosa de colores, lo que tiende a reducir la cantidad de colores utilizados y, por lo tanto, puede acercarse al número cromático óptimo.

2.7 Algoritmo de independencia MCI (máximo conjunto independiente)

Un conjunto *S* de vértices de un grafo *G* es independiente si no hay dos vértices de *S* que sean adyacentes en *G*. Un conjunto independiente *S* de vértices de un grafo *G* se llama un conjunto independiente maximal si *S* no es subconjunto propio de ningún conjunto independiente de vértices de *G*. El máximo cardinal de un conjunto independiente se llama número de independencia de *G* y se designa por *B(G).* Cada conjunto independiente de cardinal *B(G)* es maximal, pero el recíproco no necesariamente [4, 15].

Estas ideas sugieren algoritmos en los que, en un primer paso se localice un conjunto independiente de vértices de cardinal próximo a *B(G)*, se coloreen todos con el color 1, después se borre este conjunto de vértices y se repita el proceso en el grafo que resulte, continuando así hasta colorear todos los vértices [5, 6, 7].

Una forma de obtener los conjuntos independientes de un grafo es la siguiente:

Paso 1: Obtener el vértice con menor grado

Paso 2: Añadir el vértice al conjunto independiente actual

Paso 3: Eliminar del grafo el vértice seleccionado y sus vecinos.

Paso 4: Si el grafo no tiene ningún vértice, ir al Paso 5, si el grafo tiene vértices, volver al Paso 1.

Paso 5: Eliminar del grafo los vértices pertenecientes al conjunto independiente actual.

Paso 6: Si el grafo no tiene ningún vértice, se han obtenido todos los conjuntos independientes, ir al Paso 7. Si el grafo tiene vértices, inicializar el conjunto independiente actual y volver al Paso 1.

Paso 7: Fin del algoritmo.

2.8 El coloreo de grafos y los conjuntos independientes

El coloreo de grafos y los conjuntos independientes son conceptos fundamentales en teoría de grafos y están estrechamente relacionados. Ya que el coloreo de grafos se refiere a la asignación de colores a los vértices de un grafo de manera que no haya dos vértices adyacentes con el mismo color. El objetivo principal es encontrar el número mínimo de colores necesarios para colorear el grafo de manera válida, lo que se conoce como el número cromático del grafo. Y un conjunto independiente en un grafo es un conjunto de vértices en el que ningún par de vértices está conectado por una arista. En otras palabras, en un conjunto independiente, ningún vértice comparte una arista con otro vértice del mismo conjunto. El objetivo es encontrar el conjunto independiente más grande en un grafo dado, lo que se conoce como el número de independencia del grafo [13].

La relación entre el coloreo de grafos y los conjuntos independientes es inversa en cierto sentido, debido a que:

- En un grafo, el número cromático (el mínimo número de colores necesarios para colorear el grafo) está relacionado con el tamaño del conjunto independiente más grande. Si un grafo puede ser coloreado con *k* colores, entonces hay al menos un conjunto independiente de tamaño *k*, ya que cada vértice en un conjunto independiente debe tener un color diferente.

- Por otro lado, el complemento del conjunto independiente más grande (los vértices que no están en el conjunto independiente) formaría un conjunto que puede ser coloreado con un único color. Por lo tanto, el tamaño del conjunto independiente más grande está relacionado con el número cromático complementario.

- El número de independencia de un grafo (tamaño del conjunto independiente más grande) está relacionado con el número cromático complementario. El número cromático complementario es el número mínimo de colores necesarios para colorear el grafo de manera que los vértices en el conjunto independiente más grande tengan diferentes colores.

- Colorear el grafo de manera que los vértices en el conjunto independiente más grande tengan diferentes colores es equivalente a encontrar el número cromático complementario.

En muchos casos, resolver problemas de coloreo y conjuntos independientes en un grafo puede proporcionar información valiosa sobre su estructura y propiedades.

En este proyecto de tesis se utilizará una combinación de las principales propuestas algorítmicas para crear un sistema grafico en web que permita probar con cuantos colores se puede colorear un grafo.

2.9 La conjetura de los cuatro colores

Los vértices de cada gráfico plano se pueden colorear con cuatro o menos colores de tal manera que cada dos vértices adyacentes tengan un color diferente. Si bien la historia matemática bien podría recordar a Alfred Bray Kempe como el individuo que anunció que había resuelto el problema de los cuatro colores, luego escribió un artículo "probando" esto y luego se demostró que estaba equivocado, esta caracterización de lo que hizo es en sí misma incorrecta. e injusto. La técnica que desarrolló Kempe fue inteligente y luego demostró ser útil. Luego tuvo muchos logros en matemáticas. De hecho, lo que todos podemos aprender de Kempe es que debemos estar dispuestos a probar nuevas ideas y no tener miedo de cometer errores. Hay mucho que aprender de los errores de los demás y de los que cometemos nosotros mismos.

El problema de los cuatro colores en el siglo XX

En 1951 el matemático danés Gabriel Andrew Dirac afirmó lo siguiente: La coloración de gráficos abstractos es una generalización de la coloración de mapas, y el estudio de la coloración de gráficos abstractos... abre un nuevo capítulo en la parte combinatoria de las matemáticas [16].

Muchos matemáticos que intentaron resolver el problema de los cuatro colores durante el siglo XX utilizaron el enfoque de Kempe con una diferencia importante. En lugar de considerar el caso en el que el mapa *M* en cuestión contenía una región rodeada por un anillo de cinco regiones vecinas, quienes intentaron resolver el problema consideraron un gran número de posibles configuraciones de regiones, al menos una de las cuales *M* debe contener. Dado que *M* tiene el número mínimo de regiones entre esos mapas que no se pueden

colorear con cuatro colores, cada mapa que consta de regiones dentro y fuera del anillo de regiones que rodean cualquiera de estas configuraciones se puede colorear con cuatro colores. Si se pudiera demostrar que, por cada coloración de cada uno de estos mapas más pequeños con cuatro colores, también había una coloración con cuatro colores de las regiones de la configuración dentro del anillo, lo que resultaba en una coloración de todo el mapa con cuatro colores, entonces se obtendría una contradicción.

Sin embargo, este enfoque entrañaba muchas dificultades. Primero, había que demostrar que cualquiera que fuera el mapa *M*, debía contener una de las configuraciones consideradas, es decir, que estas configuraciones eran inevitables. En segundo lugar, aunque el mapa que consta de las regiones que se encuentran dentro y fuera del anillo que rodea cada configuración se podía colorear con cuatro colores, era necesario mostrar que, independientemente de cómo este mapa más pequeño se podía colorear con cuatro colores, había alguna manera de que las regiones en la configuración y podrían colorearse para dar como resultado una coloración de *M* con cuatro colores, obteniendo así una contradicción. Después de muchos años durante los cuales se intentó este enfoque, dos matemáticos finalmente lo lograron. En 1976 Kenneth Appel (1932-2013) y Wolfgang Haken(1928-) de la Universidad de Illinois anunciaron que habían logrado localizar 1936.

2.10 Número Cromático $\chi(G)$

El "Número Cromático" de un grafo, denotado como $\chi(G)$, se refiere al número mínimo de colores necesarios para realizar una coloración válida de los nodos (vértices) del grafo de tal manera que ningún par de nodos adyacentes (conectados por una arista) tenga el mismo color. En otras palabras, el número cromático es la cantidad mínima de colores que se deben asignar a los nodos de manera que se cumpla la restricción de que dos nodos adyacentes no pueden tener el mismo color [13, 15].

Algunos puntos clave relacionados con el número cromático incluyen:

1.Número Cromático Mínimo: El número cromático mínimo χ(G) es el valor más pequeño de colores que se requiere para realizar una coloración válida del grafo. Es un parámetro importante en teoría de grafos y se utiliza para describir la propiedad de coloreo de un grafo.

2. Coloreo Válido: Una "coloración válida" es una asignación de colores a los nodos del grafo de tal manera que ningún par de nodos adyacentes comparta el mismo color. Es decir, si dos nodos están conectados por una arista, deben tener colores diferentes.

3. Problema de Coloreo: El problema de encontrar el número cromático mínimo de un grafo se conoce como el "Problema de Coloreo de Grafos." Es un problema de optimización combinatoria y determinar el número cromático mínimo de un grafo puede ser un desafío computacionalmente difícil en grafos generales

4. Número de Independencia y Número de Estabilidad: El número cromático está relacionado con el número de independencia y el número de estabilidad de un grafo. El número de independencia α(G) es el tamaño del conjunto independiente máximo (conjunto de nodos en el que ningún par de nodos está conectado), y el número de estabilidad β(G) es el tamaño de la clique máxima (conjunto de nodos en el que cada par de nodos está conectado). Se cumple la siguiente relación: $\alpha(G) \leq \chi(G) \leq \beta(G)$.

El número cromático es un concepto importante en teoría de grafos y tiene aplicaciones en diversas áreas, como la planificación de horarios, la asignación de recursos, la asignación de colores en mapas, la asignación de frecuencias en redes de comunicación y la resolución de problemas de optimización. Aunque encontrar el número cromático mínimo en un grafo puede ser un problema NP-duro (difícil), en ciertos casos, existen algoritmos y heurísticas eficientes para encontrar soluciones aproximadas en tiempo razonable en muchos casos prácticos.

Capítulo 3. Algoritmo de coloración implementado

El algoritmo desarrollado e implementado es el algoritmo de Greedy es uno de los métodos más simples para colorear grafos. Funciona asignando colores a los vértices de un grafo de manera iterativa, asegurándose de que los vértices adyacentes no tengan el mismo color. Los pasos necesarios se describen a continuación:

1. Inicialización: Comienza con un conjunto vacío de colores y un vértice no coloreado.

2. Selecciona un vértice no coloreado: Puedes elegir los vértices en el orden que desees, por ejemplo, comenzando por el primer vértice que encuentres.

3. Asigna el color mínimo disponible: Recorre los vértices adyacentes al vértice seleccionado y verifica los colores asignados a esos vértices. Asigna el color mínimo que no esté siendo utilizado por los vértices adyacentes.

4. Marca el vértice como coloreado: Una vez que asignas un color al vértice, márcalo como coloreado y pasa al siguiente vértice no coloreado.

5. Repite los pasos 2 a 4 hasta que todos los vértices estén coloreados.

El algoritmo de coloreo Greedy no garantiza la asignación mínima de colores, y en algunos casos puede producir soluciones subóptimas y puede ser eficiente para grafos pequeños o con estructuras simples.

3.1 Pseudocódigo del algoritmo de coloreo

A continuación, se describe el pseudocódigo del algoritmo de Greedy que se utilizó para el coloreo del grafo.

Algoritmo Greedy

Entrada: Grafo *G* con n vértices y m aristas

Salida: Vértices con el color que le corresponde

1. Inicializar un diccionario o lista de colores vacía.

2. Para cada vértice *v* en *G*, haz:

   3. Inicializa un conjunto de colores usados por los vecinos de *v*.

   4. Para cada vecino *u* de *v*, haz:

   5. Si *u* está en el diccionario de colores, agrega el color de *u* al conjunto de colores usados.

   6. Elige el color más bajo que no está en el conjunto de colores usados y asigna ese color a *v*.

   7. Agrega *v* al diccionario o lista de colores con su color asignado.

8. Para cada vértice *v* en *G:*

   Imprimir "Vértice", *v*, "coloreado con color", colores[*v*]

3.2 Prueba de escritorio del algoritmo

A continuación, se describe el funcionamiento del algoritmo de coloreo propuesto en el desarrollo de este trabajo.

1.- Definir el orden de los colores a utilizar: (1) Rojo, (2) Azul, (3) Amarillo, (4) Verde y el grafo a colorear (ver Figura 3.1).

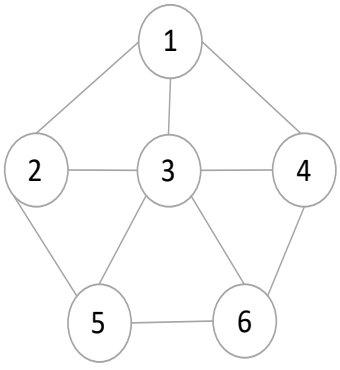

Figura 3.1. Grafo inicial

2.- Iniciar con cualquier vértice del grafo, en este ejemplo iniciaremos con el vértice marcado con la etiqueta 1. Asignamos el primer color disponible (Rojo), ver Figura 3.2.

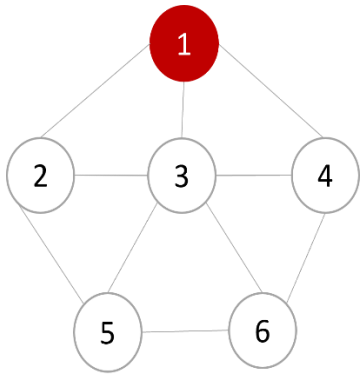

Figura 3.2. Coloreo del vértice inicial

3.- Seleccionar el segundo nodo del grafo (vértice 2), como es adyacente con el vértice 1, entonces no podemos usar el color actual (rojo), por lo que usamos el siguiente color en la lista de colores disponible (azul), por lo que la Figura 3.3. Muestra cómo queda el grafo coloreado al momento.

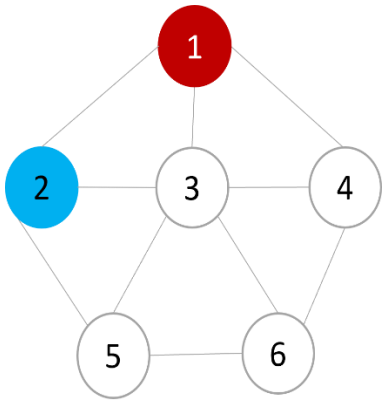

Figura 3.3. Grafo colorado con los dos primeros vértices

4.- Continua el mismo proceso, elegimos el vértice 3 y revisamos que es adyacente con los vértices con color asignado (rojo y azul) antes por lo que debemos usar un nuevo color de la lista de colores disponibles, en este caso usamos el amarillo como se muestra en la Figura 3.4.

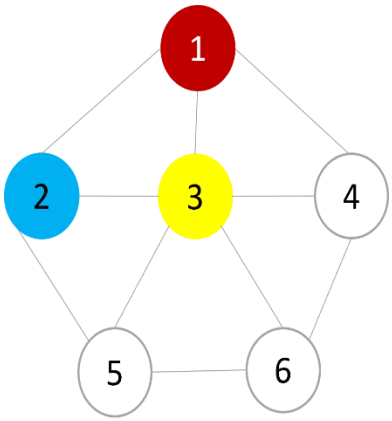

Figura 3.4. Grafo colorado con los tres primeros vértices

5.- El procedimiento de coloreo del grafo continua, ahora seleccionamos el vértice 4 y verificamos sus adyacencias considerando los colores utilizados anteriormente, comparamos versus el vértice 1 y como si es adyacente al 4 entonces no podemos usar el color rojo, pero al comparar el color azul vemos que el vértice 4 no es adyacente con el vértice de color azul (2), entonces si podemos usar el color azul (ver Figura 3.5).

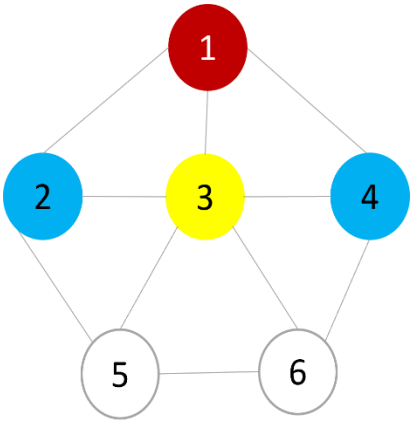

Figura 3.5. Grafo colorado con los cuatro primeros vértices

6.- El procedimiento se repite de forma iterativa eligiendo el siguiente vértice y comparándolo con la lista de colores utilizado para verificar si hay o no adyacencias. La Figura 3.6 representa el grafo donde se colorea el vértice 5 la

Figura 4.7 representa el grafo final coloreado, como podemos notar fue necesario utilizar el cuarto color (verde) debido a que había adyacencias con los otros tres colores (rojo, azul y amarillo) asignados previamente.

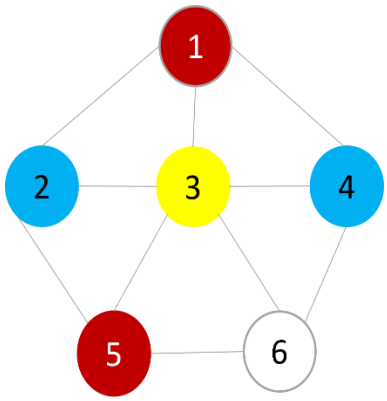

Figura 3.6. Coloreo del vértice 5

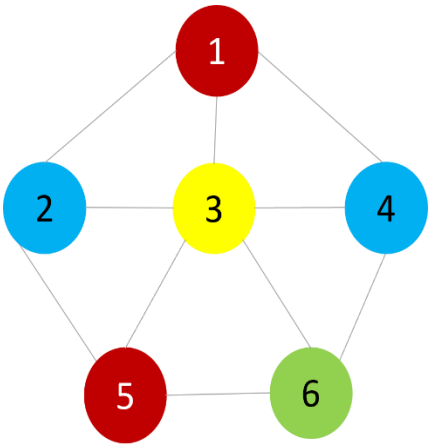

Figura 3.7. Grafo finalmente coloreado

3.3 Análisis del Algoritmo propuesto

La complejidad computacional del algoritmo de Greedy para la coloración de grafos puede variar según diversos factores, como el número de vértices y el

número de aristas (tamaño del grafo), la estructura del grafo (si el grafo es simple o menos simple) y la manera en que se implemente el algoritmo.

Peor Caso:

En el peor caso, el algoritmo de Greedy podría iterar a través de todos los vértices y probar todos los colores disponibles para cada uno. Esto podría llevar a una complejidad cuadrática en términos del número de vértices del grafo. En el caso de grafos densos, donde cada vértice está conectado con muchos otros, la coloración podría requerir muchos intentos antes de encontrar un color válido, lo que aumentaría la complejidad por ejemplo con grafos K12 (Figura 3.7).

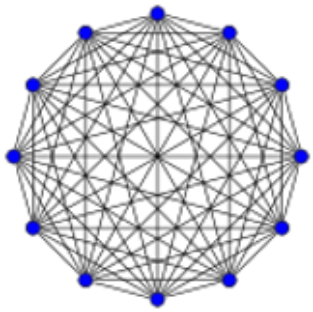

Figura 3.7. Grafo K12

Mejor Caso:

En el mejor caso, si el grafo es fácil de colorear y los vértices se pueden colorear rápidamente sin conflictos, la complejidad podría ser lineal en términos del número de vértices. Por ejemplo, grafos como la Figura 3.8.

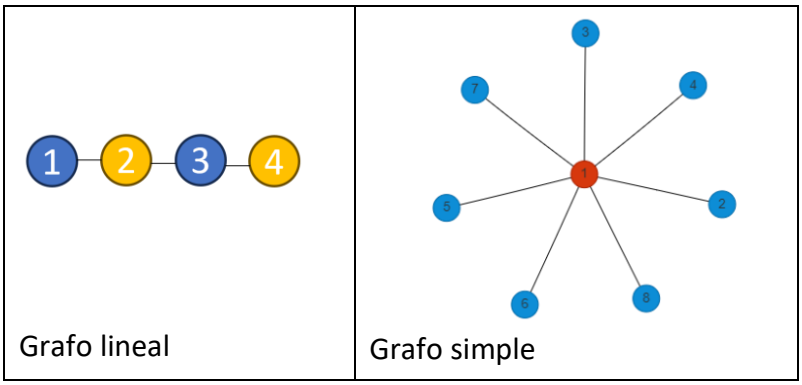

Figura 3.8. Grafos simples con dos colores

Factor de Ramificación:

La complejidad también podría depender del factor de ramificación del grafo, que se refiere a la máxima cantidad de vecinos que puede tener un vértice. Un factor de ramificación más bajo puede reducir la cantidad de colores disponibles que el algoritmo debe probar para cada vértice (ver Figura 3.8).

Complejidad Temporal:

En el peor de los casos, el algoritmo de Greedy puede requerir un tiempo cuadrático ($O(V^2)$) para colorear un grafo con V vértices. Esto puede ocurrir si cada vértice tiene un grado similar al número total de vértices en el grafo, lo que significa que tendrías que verificar todos los colores para cada vértice.

Sin embargo, en muchos casos prácticos, la complejidad temporal tiende a ser mejor que cuadrática. La eficacia del algoritmo de Greedy depende en gran medida de la estructura del grafo y de cómo se elijan los colores disponibles.

Complejidad Espacial:

La complejidad espacial del algoritmo de Greedy es generalmente lineal ($O(V)$) en función del número de vértices en el grafo. Esto se debe a que el algoritmo generalmente necesita almacenar información sobre los colores de los vértices y la disponibilidad de colores.

Es importante tener en cuenta que el algoritmo de Greedy es un algoritmo heurístico y no garantiza encontrar la solución óptima. Puede encontrar soluciones razonables en un tiempo razonable para muchos grafos, pero puede tener dificultades para lidiar con grafos muy complejos o específicos.

La complejidad computacional del algoritmo de Greedy puede variar desde lineal hasta cuadrática, dependiendo de varios factores, y puede ser influenciada por la estructura y características del grafo en cuestión como se mencionó antes.

Capítulo 4. Sistema de coloreo de grafos

En esta sección se presenta el sistema web que permite leer un archivo de entrada y generar el coloreo del grafo indicando el número de colores (número cromático) que fue utilizado para dicho coloreo.

4.1. Herramientas y Lenguaje de desarrollo

Para la implementación del algoritmo se utilizó un servidor Web, el lenguaje de programación PHP, HTML, CSS3 y la librería Vis.js que permite visualizar los grafos de forma gráfica. Además, se utilizó programación orientada a objetos como paradigma para el desarrollo del sistema.

Servidor Web

Un servidor web es un software o hardware que proporciona servicios de hosting y entrega de contenido en la World Wide Web (WWW) a través del protocolo HTTP (Hypertext Transfer Protocol). Los servidores web son esenciales para la distribución de sitios web y aplicaciones web a los usuarios finales que acceden a través de sus navegadores web. En nuestro trabajo se utilizó el servidor Apache HTTP Server que es uno de los servidores web más utilizados en el mundo. Es de código abierto y se ejecuta en una variedad de sistemas operativos, incluyendo Linux, Windows y macOS. Para este propósito se descargó e instalo WAMPSERVER (https://www.wampserver.com/en/).

Los servidores web pueden servir diversos tipos de contenido web, como páginas HTML estáticas, imágenes, archivos de estilo, scripts de servidor (como PHP, Python o Ruby), y más. También pueden gestionar la autenticación, la seguridad y otras funciones relacionadas con el servidor. Un servidor web es una parte esencial de la infraestructura de internet, ya que permite que los sitios web y las aplicaciones web sean accesibles para los usuarios en todo el mundo.

Lenguaje PHP

PHP (Hypertext Preprocessor) es un lenguaje de programación ampliamente utilizado en el desarrollo web. PHP desempeña un papel fundamental en el desarrollo web, permitiendo la creación de aplicaciones web dinámicas y accesibles. Su facilidad de aprendizaje, amplia adopción y eficiencia en el procesamiento web lo convierten en una herramienta importante para desarrolladores web de todo el mundo. Su importancia radica en varias áreas:

1. Desarrollo web dinámico: PHP es especialmente adecuado para la creación de sitios web dinámicos, donde el contenido se genera en tiempo real en función de las solicitudes del usuario. Puede conectarse a bases de datos, procesar formularios, y realizar diversas tareas que permiten la interacción del usuario con la aplicación.

2. Amplia adopción: PHP ha sido ampliamente adoptado en la comunidad de desarrollo web. Muchos sitios web populares, como Facebook y WordPress, utilizan PHP en su infraestructura. Esto significa que hay una gran cantidad de recursos disponibles en línea, como bibliotecas, tutoriales y comunidades de desarrolladores.

3. Facilidad de aprendizaje: PHP es conocido por ser relativamente fácil de aprender para aquellos que están comenzando en el desarrollo web. Su sintaxis es similar a otros lenguajes como C y JavaScript, lo que facilita la transición para muchos programadores.

4. Amplia compatibilidad: PHP se ejecuta en una amplia variedad de plataformas y sistemas operativos, incluyendo Linux, Windows, macOS y más. También es compatible con numerosas bases de datos, como MySQL y PostgreSQL.

5. Eficiencia en el procesamiento web: PHP fue diseñado específicamente para el procesamiento web, por lo que tiende a ser eficiente en esta tarea. Puede manejar grandes cantidades de solicitudes web sin un alto consumo de recursos del servidor.

HTML y CSS

HTML (HyperText Markup Language) y CSS (Cascading Style Sheets) son dos lenguajes fundamentales en el desarrollo web que trabajan juntos para crear y dar formato a páginas web. Cada uno desempeña un papel específico en la construcción de sitios web interactivos y visualmente atractivos.

Vis.js

Vis.js (https://visjs.org/) es una biblioteca JavaScript de código abierto que se utiliza para crear visualizaciones interactivas en la web. Vis.js es ampliamente utilizado en aplicaciones web que requieren visualizaciones de datos, como aplicaciones de análisis, herramientas de representación gráfica de redes sociales, sistemas de gestión de proyectos y mucho más. Algunas de las características principales son:

1. Visualización de datos: Vis.js permite representar datos complejos y relaciones de una manera visualmente efectiva. Puedes utilizarlo para crear gráficos, diagramas, redes, líneas de tiempo y otras visualizaciones.

2. Interactividad: Las visualizaciones creadas con Vis.js son interactivas, lo que significa que los usuarios pueden interactuar con los elementos visuales, hacer clic en nodos, arrastrar elementos, acercar y alejar, y más, según el tipo de visualización.

3. Personalización: La biblioteca ofrece una amplia gama de opciones de personalización, lo que te permite controlar la apariencia y el comportamiento de tus visualizaciones. Puedes establecer colores, tamaños, etiquetas y otros atributos de los elementos visuales.

4. Facilidad de uso: Vis.js proporciona una API sencilla de utilizar que facilita la creación de visualizaciones personalizadas. También incluye una documentación detallada y ejemplos que ayudan a los desarrolladores a comenzar rápidamente.

5. Licencia de código abierto: Vis.js se distribuye bajo una licencia de código abierto (MIT License), lo que significa que es gratuito para su uso en proyectos personales y comerciales.

4.2 Formato de los archivos de entrada

El sistema web de coloreo de grafos funciona mediante la lectura de un archivo de texto que representa el grafo a colorear. Por ejemplo, la Figura 6 muestra el grafo de Petersen, si deseamos colorear dicho grafo, primero debemos crear el archivo de texto que represente el número de vértices y las aristas correspondientes.

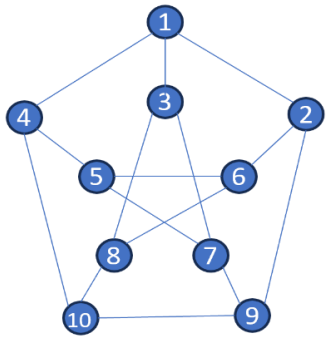

Figura 4.1. Grafo de Petersen

El archivo de texto (petersen.txt) que representa el grafo de la Figura 4.1 debe tener el formato donde en la primera línea se especifica el número total de vértices y de la segunda línea en adelante se especifican las adyacencias dejando un espacio entre el vértice inicial y el vértice final.

```
10      ← número de nodos
1 2     ← adyacente del vértice 1 con 2
1 3     ← adyacente del vértice 1 con 3
1 4     ← adyacente del vértice 1 con 4
2 6     ← adyacente del vértice 2 con 6
2 9
3 7
3 8
4 5
4 10
5 6
5 7
6 8
7 9
8 10
9 10    ← adyacente del vértice 9 con 10
```

4.3 Pantallas y pruebas del sistema

A continuación, se describe el funcionamiento del sistema web implementado sobre el servidor web (localhost). La Figura 4.2. representa l página inicial del sistema, se describe brevemente el problema de coloreo de grafos y se incluye un menú con las opciones:

Inicio: Dando clic en la imagen de Koloreo.

Kolorear: Permite cargar un archivo de texto para colorear el grafo.

Pruebas: Muestra una lista de grafos de ejemplo que al seleccionar alguno se colorea.

Autores: Describe los autores de este trabajo.

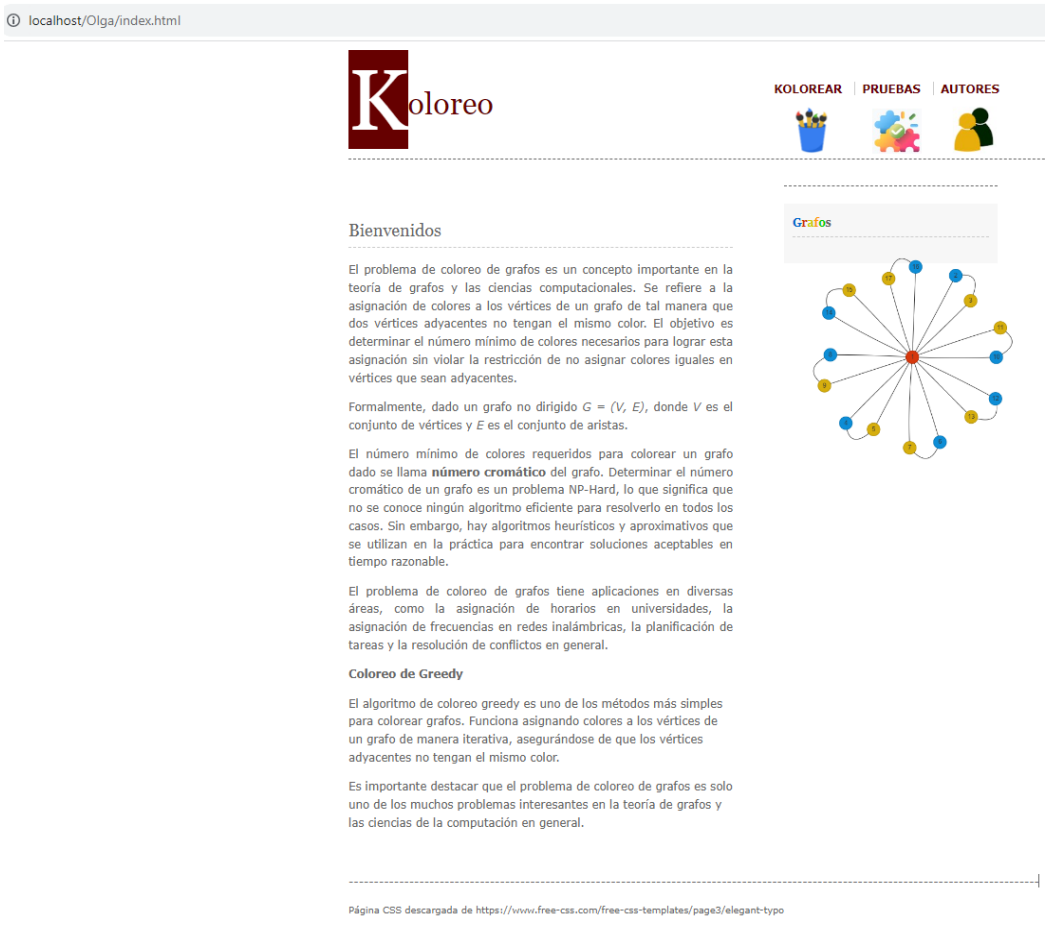

Figura 4.2. Página Index.html del sistema web de coloreo de grafos

Figura 4.3. Página para cargar el archivo a colorear

En la Figura 4.3 se indica la página que permite cargar el archivo de texto donde se tiene el archivo que representa el grafo, en la imagen se indica que se cargó el archivo petersen.txt, recordando que el archivo de texto tiene el número de vértices y las aristas correspondientes para dibujar el grafo. En la parte derecha del sistema web se ejemplifica la forma en que se debe crear el archivo de texto, esto por si el usuario es nuevo o si olvida el formato de los archivos de texto.

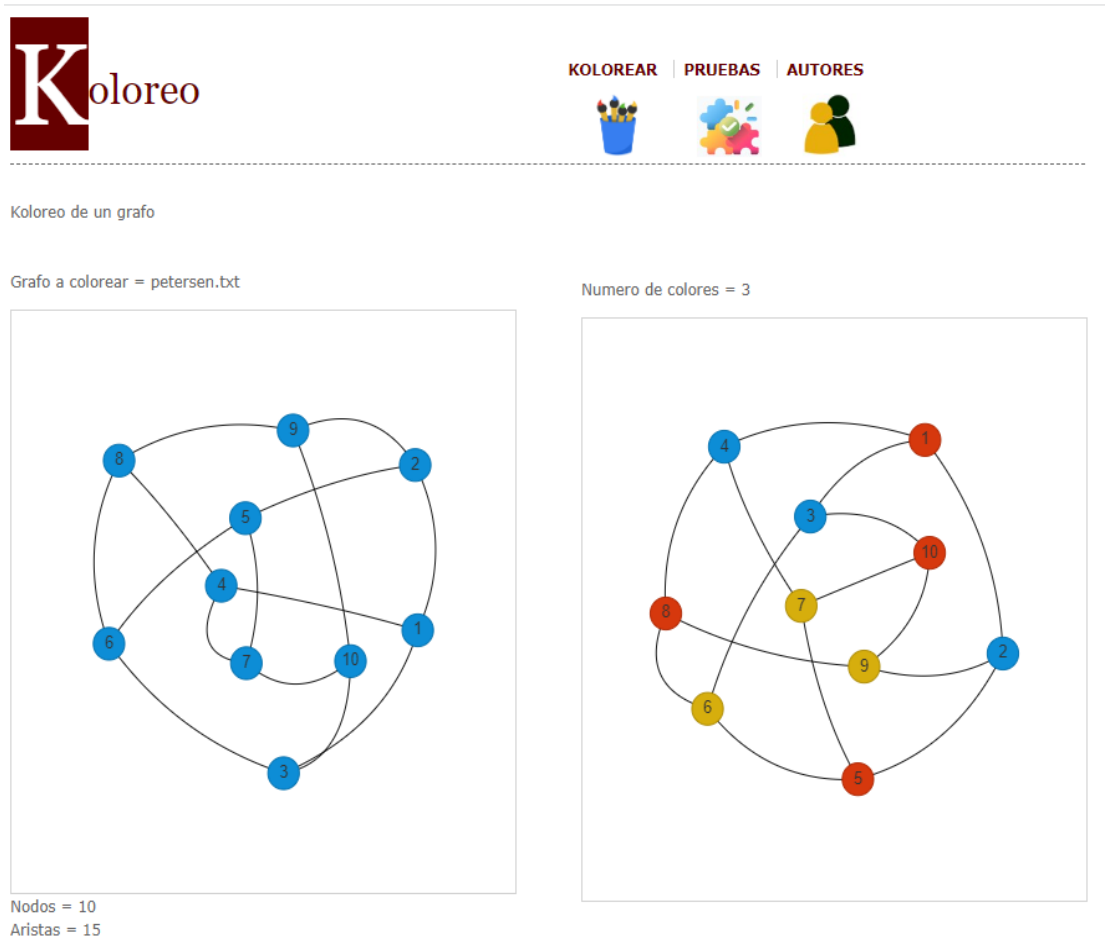

Figura 4.4. Página del coloreo de un grafo

La Figura 4.4 muestra la ejecución del sistema de coloreo de grafos del grafo de Petersen almacenado en el archivo peterse.txt. De lado izquierdo se muestra el grafo original con solo un color mientras que de lado derecho se muestra el grafo coloreado de acuerdo con la aplicación del algoritmo se determina que el número de colores = 3 (número cromático). De igual forma se indica el número de vértices y el número de aristas que tiene el grafo.

En todo momento está disponible el menú superior para poder cambiar de una opción a otra o incluso al dar clic en la K el sistema de dirige a la página principal del sistema.

4.4 Ejemplos de coloreo de grafos

En la opción “PRUEBAS” del sistema de k coloración se muestra una lista de diversos grafos que al seleccionar se muestra el grafo original y su coloración (ver Figura 4.5).

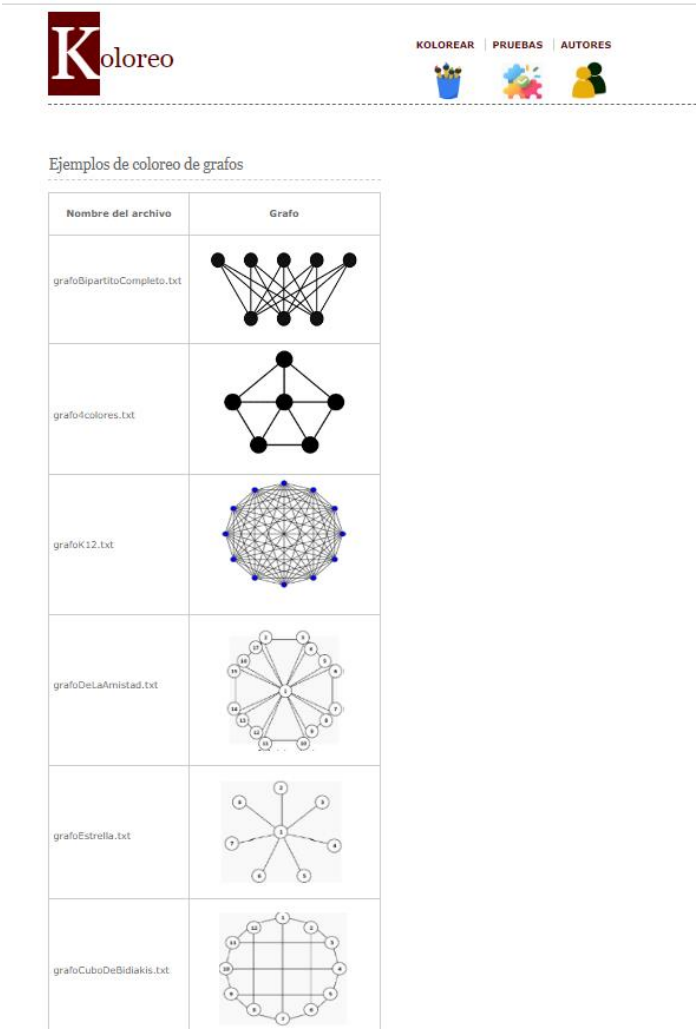

Figura 4.5. Grafos de prueba para colorear

Las Figuras 4.6, 4.7, 4.8 y 4.9 muestra la selección de algunos ejemplos del coloreo de grafo de la sección de “PRUEBAS”. En cada caso se muestra el grafo original en un color y el grafo coloreado con los colores que determina el algoritmo de coloración implementado.

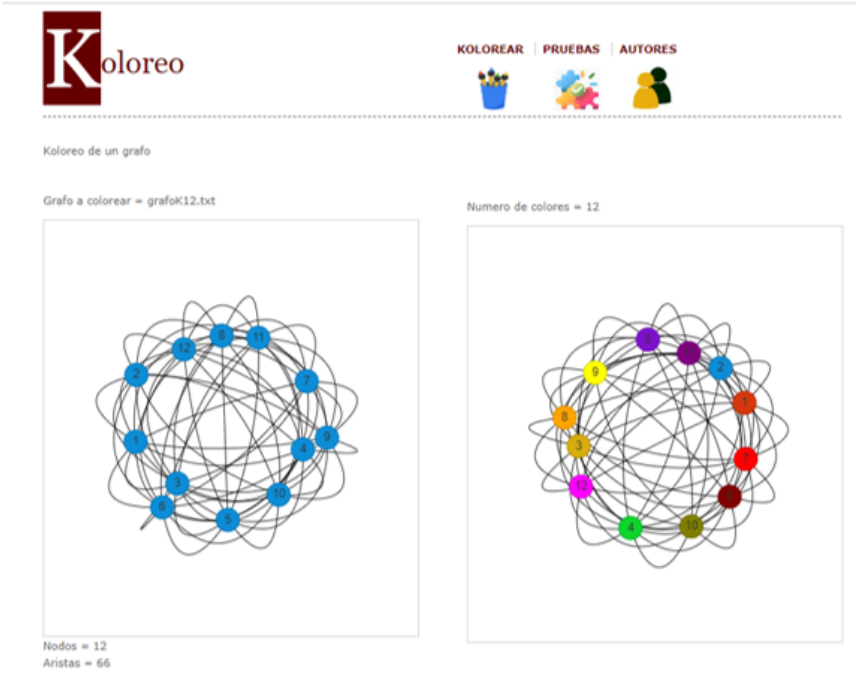

Figura 4.6. Grafo K12

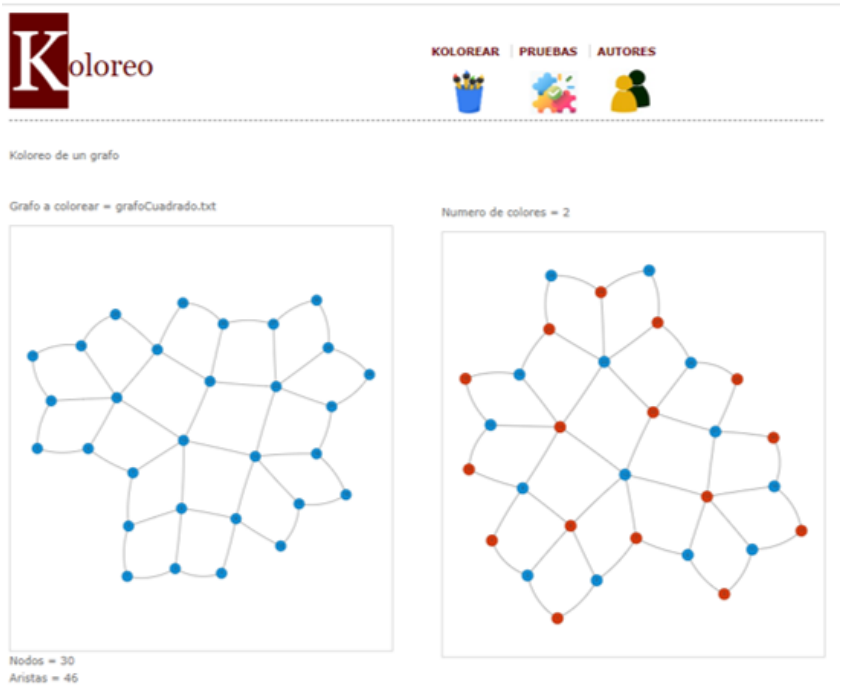

Figura 4.7. Grafo cuadrado

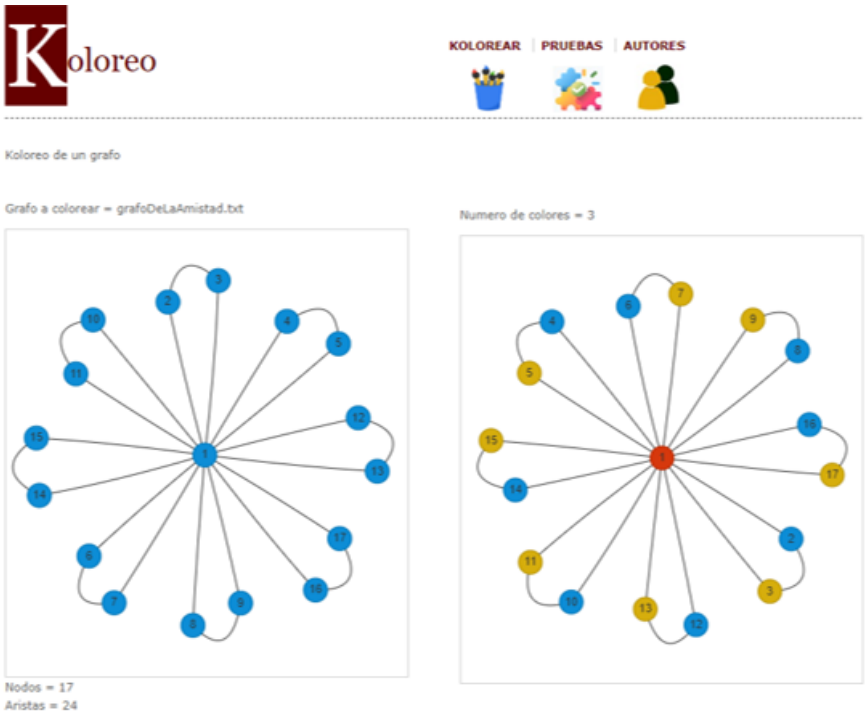

Figura 4.8. Grafo de la amistad

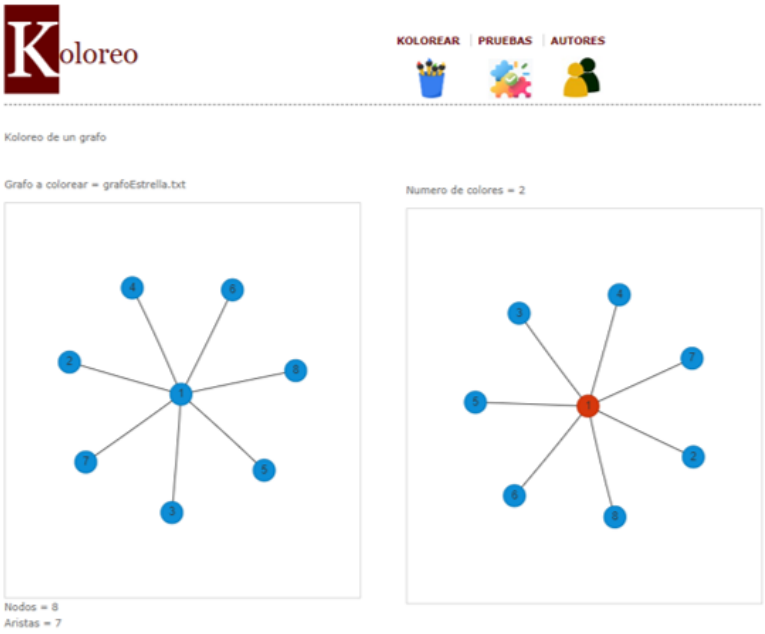

Figura 4.9. Grafo estrella

La Figura 4.10 muestra la pantalla del sistema al seleccionar la opción de "AUTORES".

Figura 4.10. Autores del trabajo

4.5 Estructuras de datos del sistema

En la implementación del sistema de coloreo se utilizó el lenguaje de programación PHP que permite de forma fácil el manejo de un entorno visual con HTML y con hojas de estilo CSS3. Además, se utilizó una clase denominada grafo como estructura de datos:

```
class Grafo {
// atributos
   public $NN; // número de nodos
   public $NA; // número de aristas
   public $NC; // número de colores
   public $NI; // nodos iniciales
   public $NF; // nodos finales
   public $COL; // color de nodos
   public $vertices = [];
```

```
// métodos

public function leeGrafo($arch)
public function colorear()
public function dibujo()

// Instancias
$grafo= new Grafo();
$grafo->leeGrafo($archivo_entrada);

// colores a seleccionar
for($i=1; $i<=$n; $i++)
{
  $m=count($resultadoColoreo[$i]);
  for ($j=0; $j<$m; $j++)
  {
     $c=trim($resultadoColoreo[$i][$j]);
     if ($i==1) $grafo2->COL[$c]='#D6380D'; // rojoInicial
     if ($i==2) $grafo2->COL[$c]='#0D8DD6'; //azul
     if ($i==3) $grafo2->COL[$c]='#D6AE0D'; //amarillo
     if ($i==4) $grafo2->COL[$c]='#0DD62B'; // verde
     if ($i==5) $grafo2->COL[$c]='#7E11CE'; // morado
     if ($i==6) $grafo2->COL[$c]='#800000'; // maroon
     if ($i==7) $grafo2->COL[$c]='#ff0000'; // red hard
     if ($i==8) $grafo2->COL[$c]='#ffA500'; // orange
     if ($i==9) $grafo2->COL[$c]='#ffff00'; // yellow
     if ($i==10) $grafo2->COL[$c]='#808000'; // olive
     if ($i==11) $grafo2->COL[$c]='#800080'; // purple
     if ($i==12) $grafo2->COL[$c]='#ff00ff'; // fuchsia
     if ($i==13) $grafo2->COL[$c]='#ffffff'; // white
     if ($i==14) $grafo2->COL[$c]='#00ff00'; // lime
     if ($i==15) $grafo2->COL[$c]='#008000'; // green
     if ($i==16) $grafo2->COL[$c]='#000080'; // navy
     if ($i==17) $grafo2->COL[$c]='#0000ff'; // blue
     if ($i==18) $grafo2->COL[$c]='#00ffff'; // aqua
     if ($i==19) $grafo2->COL[$c]='#008080'; // teal
     if ($i==20) $grafo2->COL[$c]='#000000'; // black
     if ($i==21) $grafo2->COL[$c]='#c0c0c0'; // silver
     if ($i==22) $grafo2->COL[$c]='#808080'; // gray
  }
}
```

En el código anterior se puede notar que el algoritmo puede asignar un máximo de 22 colores. En las pruebas realizadas se probó con grafos completos como los denominados grafos K, el mayor grafo K coloreado es el grafo K14, este tipo de grafo es complejo por el número de adyacencias que tiene.

Se generaron grafos aleatorios con 100 vértices (ver Figura 4.11 para su coloreo) y coloreo de grafos con 200 vértices (ver Figura 4.12), con un número variable de aristas.

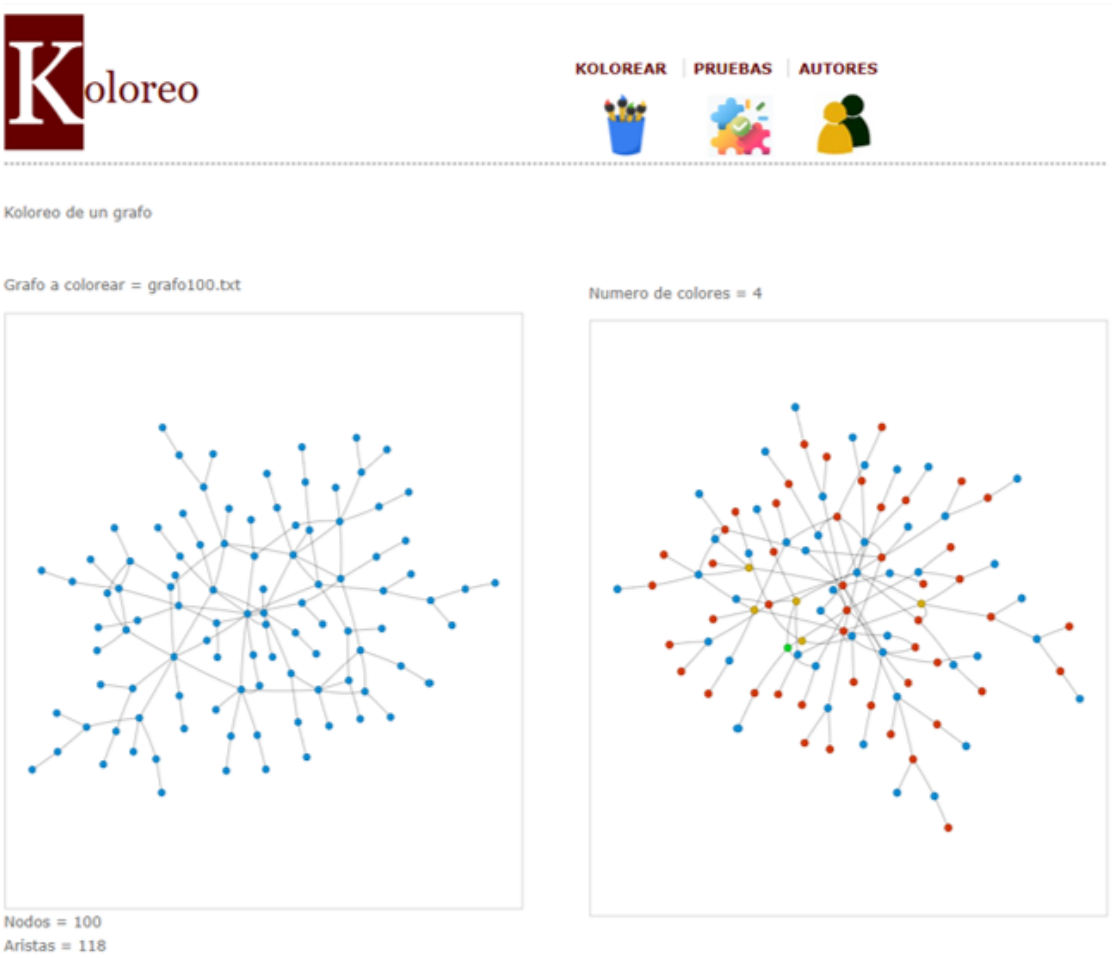

Figura 4.11. Grafo con 100 vértices y 118 aristas

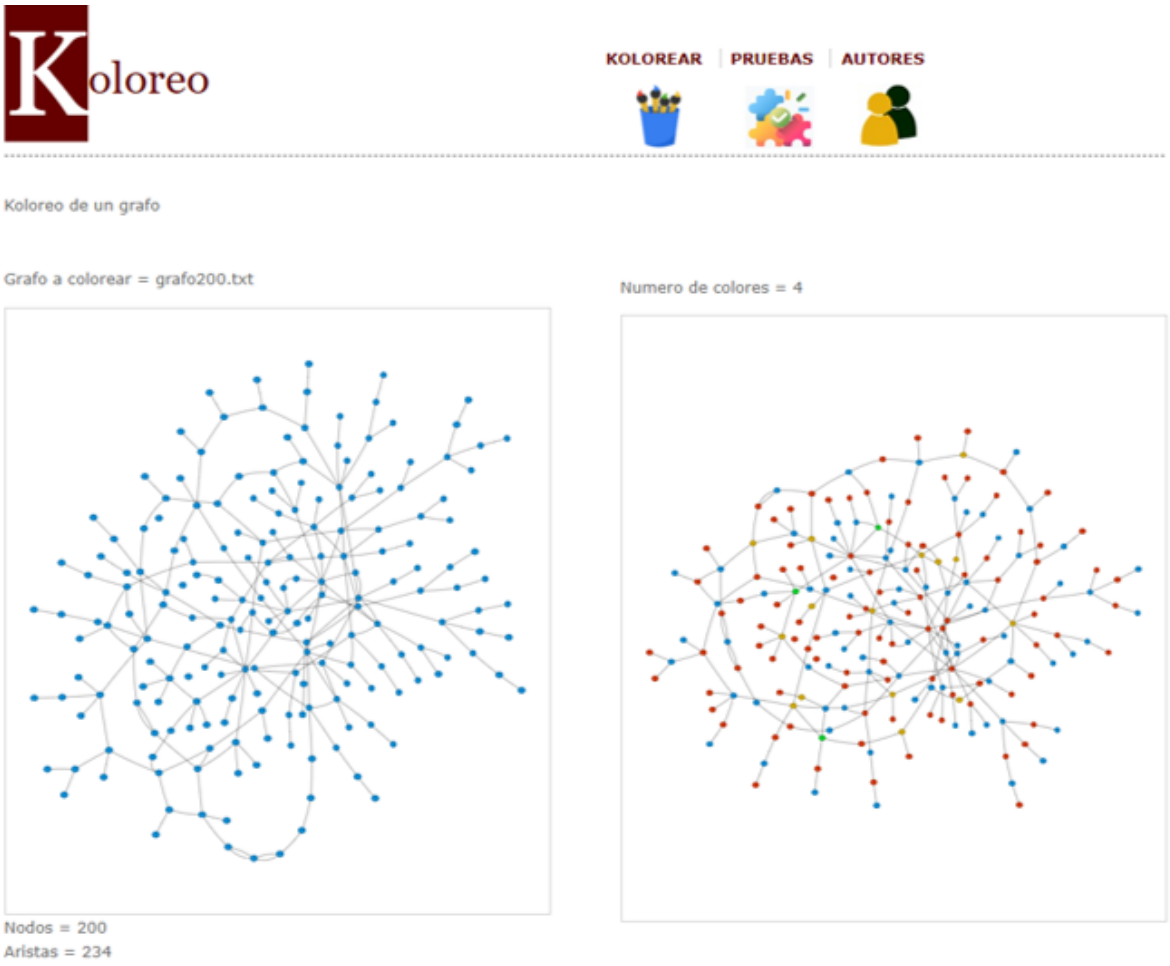

Figura 4.12 Grafo con 200 vértices y 234 aristas

Conclusiones

El coloreo de grafos es una herramienta matemática esencial que se aplica en una amplia variedad de campos para resolver problemas de asignación, planificación y optimización, ayudando a mejorar la eficiencia y la toma de decisiones en una amplia gama de aplicaciones prácticas. La teoría de grafos es una mezcla extraordinaria de historia, cultura y soluciones a problemas complejos desde el mundo de las matemáticas. Con esta teoría se busca representar de forma visual conjuntos de datos abstractos en formas de nodos o vértices y la unión o relaciones que estas pueden tener con otros nodos a través de aristas. Gracias a esta teoría se han podido lograr grandes avances en el análisis de amplios volúmenes de data. En la actualidad los grafos integran parte central de un número increíble de soluciones a problemas complejos. Son uno de los componentes fundamentales de la analítica social, relaciones entre personas y más recientemente se han utilizado sus propiedades para incrementar la eficiencia en la lucha contra el blanqueo de capitales y el fraude bancario.

El trabajo desarrollado apoya a los estudiantes de computación y áreas afines a entender el problema del coloreo de grafos con el fin de incentivar a poder aplicar la estrategia aquí presentada para resolver otros problemas computacionales. Las principales aportaciones del trabajo de tesis a las ciencias computacionales son:

- Un algoritmo para el k-coloreo de un grafo conexo
- Un sistema web que permita realizar pruebas para diversas instancias de grafos de forma gráfica.
- Un documento informativo de la Integración del sistema web con la implementación del programa de k-coloreo en un lenguaje de programación para web.

Perspectivas

Programar otras estrategias de algoritmos de coloreo de grafos, que se muestren de forma visual y que genere una tabla comparativa respecto a los tiempos de ejecución y al número cromático obtenido.

Implementar una función para determinar el tiempo de ejecución que tarda el algoritmo con el objetivo de obtener los limites del sistema.

Bibliografía

[1] Benjamin A., Chartrand G., Zhang P. (2015). The Fascinting World of Graph Theory, Princeton University Press. Oxford, USA.

[2] Greenhill C. (1999). The complexity of counting colourings and independent sets in sparse graphs and hypergraphs, Computational Complexity.

[3] Cartwright D., Harary F. (1968). On the coloring of signed graphs. Elemente der Mathematik 23: 85-89.

[4] https://www.makingcode.dev/2015/08/algoritmos-de-coloracion-de-grafos.html.

[5] Sisa J. (2002). Estructuras de Datos y Algoritmos con énfasis en programación orientada a objetos, Prentice Hall, Colombia.

[6] Trudeau R. J. (1994). Introduction to Graph Theory. www.doverpublications.com.

[7] Lee R. C. T., Tseng S. S., Chang R. C., Tsai Y.T. (2005). Introducción al diseño y análisis de algoritmos, Mc Graw Hill, México.

[8] López G., Jeder I., Vega A. (2009). Análisis y diseño de algoritmos, implementados en C y Pascal. Alfaomega. México.

[9] Papadimitriou, C. H. (1994). Computational complexity. Addison Wesley (1994).

[10] Gutiérrez A., Bravo G. (2005). PHP5, Alfaomega Ra-MA, México.

[11] Welling L., Thomson L. (2009). Desarrollo Web con PHP y MySQL Editorial Anaya. México.

[12] Sommerville I. (2005). ingeniería del software, Séptima edición, Pearson Educación. S. A. Madrid.

[13] Pons C., Rosenfeld R., y Smith C. (2017). Lógica para informáticos. Editorial de la Universidad de La Plata, Buenos Aires, Argentina.

[14] Tremblay J, P., Sorenson P. G. (2017). An introduction to data strcutures with aplplications, second edition, McGraw Hill.

[15] Lewis R. (2015). A guide to Graph Colouring: Algorithms and Applications. Springer. Berlin.

[16] Wilson R. (2004). Four Colors Suffice: How the Map Problem Was Solved, Princeton Science Library.