Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

Una propuesta algorítmica para aproximar el coloreo de Grafos

Tesis de Licenciatura en Ingeniería en Ciencias de la Computación

Presenta

Raymundo Pérez Gómez

Asesor

Dr. Guillermo De Ita Luna

Febrero 2014

AGRADECIMIENTOS

- Primero ante todo quiero agradecer a Dios por todo lo que me ha brindado y me sigue brindando, por ayudarme en todo momento, por acompañarme en mi trayecto académico, por guiarme por alguna opción cuando ya no encuentro más opciones, y por qué sé que siempre me estará guiando para llevar un buen rumbo por la vida, TE ALABO GRACIAS DIOS NUESTRO.

- Quiero agradecer a mis hermanos por apoyarme económicamente y moralmente, por estar al pendiente de mi crecimiento académico, en especial agradecer a mi madre por brindarme apoyo incondicional, que siempre me ayuda en todo momento, que no me deja solo, que ella está pendiente para cuando la necesite, las decisiones académicas tomadas mi madre me ha correspondido y apoyado tanto económicamente como moralmente, que siempre tiene un alimento para brindarme. TE AMO DEMASIADO MAMA.

- También quisiera agradecer a el personal de esta facultad, personal administrativo y personal académico por brindarme sus conocimientos, consejos y amistad, sin duda alguna esa amistad es para mí es un tesoro muy grande.

- Quisiera hacer un énfasis y agradecer a mis compañeros de clases pues con ellos compartí conocimientos, risas, y sobre todo buena amistad, no sin antes agradecer a mi novia por creer en mí y confiar en que yo terminaría de manera satisfactoria mi tesis, GRACIAS MI AMOR TE AMO.

- Por ultimo quiero pedir una disculpa a todos los mencionados si en algún momento de nuestro trato les haya mal correspondido, desde luego rectificar mi cariño y afecto hacia todos los mencionados.

Contenido

Tabla de Figuras 4

I) Introducción 5

1.1 Planteamiento del Problema 6

1.2 Estado del Arte 8

1.3 Relevancia del Problema 10

1.4 Objetivos del Proyecto. 12

II) Teoría de Grafos y Técnicas de Búsqueda. 13

2.1 Conceptos Preliminares de la Teoría de Grafos. 13

2.2 Recorrido en Amplitud 14

2.3 Recorrido a lo Profundo. 15

2.3.1 Árbol Generador y Ciclos básicos. 17

2.4 Ciclos Intersectados y Compuestos 18

2.5 Puntos de Articulación 20

III) Procedimientos básicos para el coloreo 28

3.1 Procedimiento para el 2 Coloreo 28

3.2Condiciones Necesarias para el 3 Coloreo 29

3.3 Grafos que no son 3-coloreables 30

3.4 Algoritmos básicos de coloración 32

3.4.1 Coloreo Secuencial o voraz 32

3.4.2 Algoritmo de coloración Welsh-Powell 33

IV) Análisis e Implementación de la Propuesta Algorítmica 34

4.1Definición y Obtención de Conjunto I_B 34

4.2Definición y Lemas para la Obtención del MIS K 35

4.3 Búsqueda del Número Cromático 37

4.4 Implementación del sistema 38

4.4.1 Descripción General del Sistema 38

4.4.2 Diagrama General del Sistema 39

V) Pruebas y Resultados 42

VI) Análisis de complejidad 51

VII) Conclusiones y Trabajo a Futuro 53

VIII) Bibliografía y Referencias 54

Tabla de Figuras

Fig. 1. Líneas del metro 5
Fig. 2. Programación de actividades 6
Fig. 3. Grafo no dirigido 6
Fig. 4. Grafo coloreado con 3 colores 7
Fig. 5. Puentes de Königsberg 8
Fig. 6. Circuitos 8
Fig. 7. Mapa de 4 colores 9
Fig. 8. Algoritmo para el recorrido en amplitud de un grafo 15
Fig. 9. Recorrido en amplitud de un grafo 15
Fig. 10. Algoritmo para el recorrido en profundidad 16
Fig. 11. Recorrido a profundidad de un grafo 17
Fig. 12. Grafo de Petersen con su respectivo árbol generador 18
Fig. 13. Ciclo básico de un grafo 18
Fig. 14. Ciclos intersectados e independientes 19
Fig. 15. a) ciclos básicos, b) ciclo compuesto 19
Fig. 16. Puntos de Articulación 21
Fig. 17. Búsqueda de Bloques y puntos de articulación 22
Fig. 18. a) Grafo 2-coloreable b) Grafo que no es 2-coloreable 28
Fig. 19. Grafo con dos ciclos impares 29
Fig. 20. Ciclo impar intersectado con un ciclo par 29
Fig. 21. Ejemplo de ciclos impares embebidos 30
Fig. 22. Coloración con algoritmo secuencial 32
Fig. 23. Coloración con el algoritmo Welsh-Powell 33
Fig. 24. Procedimiento de construcción de un MIS(G) 37
Fig. 25. Algoritmo búsqueda del númeroCromático(G) 38
Fig. 26. Grafo procesado y coloreado 39
Fig. 27. Nodos adyacentes marcados con una etiqueta 39
Fig. 28. Diagrama general de sistema 40
Fig. 29. Grafo y su representación con listas de adyacencia 40
Fig. 30. Conjunto I_B. 41
Fig. 31. Grafo completo sin coloreo 42
Fig. 32. Matriz de adyacencia 42
Fig. 33. Grafo quitando el conjunto I_B, $G_1=G-I_B$ 43
Fig. 34. Eliminación de nodo maximal 6 44
Fig. 35. Eliminación de nodo maximal 9 44
Fig. 36. Eliminación de nodo maximal 10 45
Fig. 37. Eliminación de nodo máximal 0 45
Fig. 38. Eliminación de nodo máximal 7 46
Fig. 39. Se regresan los nodos de la pila 46
Fig. 40. Eliminación de nodo maximal 8 47
Fig. 41. Regresando los nodos de la pila 48
Fig. 42. Grafo Coloreado 48
Fig. 43. Grafo de Grotzsch 49
Fig. 44. Grafo de Petersen 49
Fig. 45. Grafo de myciel3 50
Fig. 46. Desigualdad 51

I) Introducción

Un grafo es una representación matemática de mucho interés para su estudio, ya que a través de él, se pueden modelar diversas situaciones de la vida real, tales como líneas de autobuses o del metro (Fig. 1.), calendarización de tareas “scheduling” (Fig. 2), etc. Esto nos permite plantear y solucionar problemas relacionados a estos modelos utilizando la teoría de grafos.

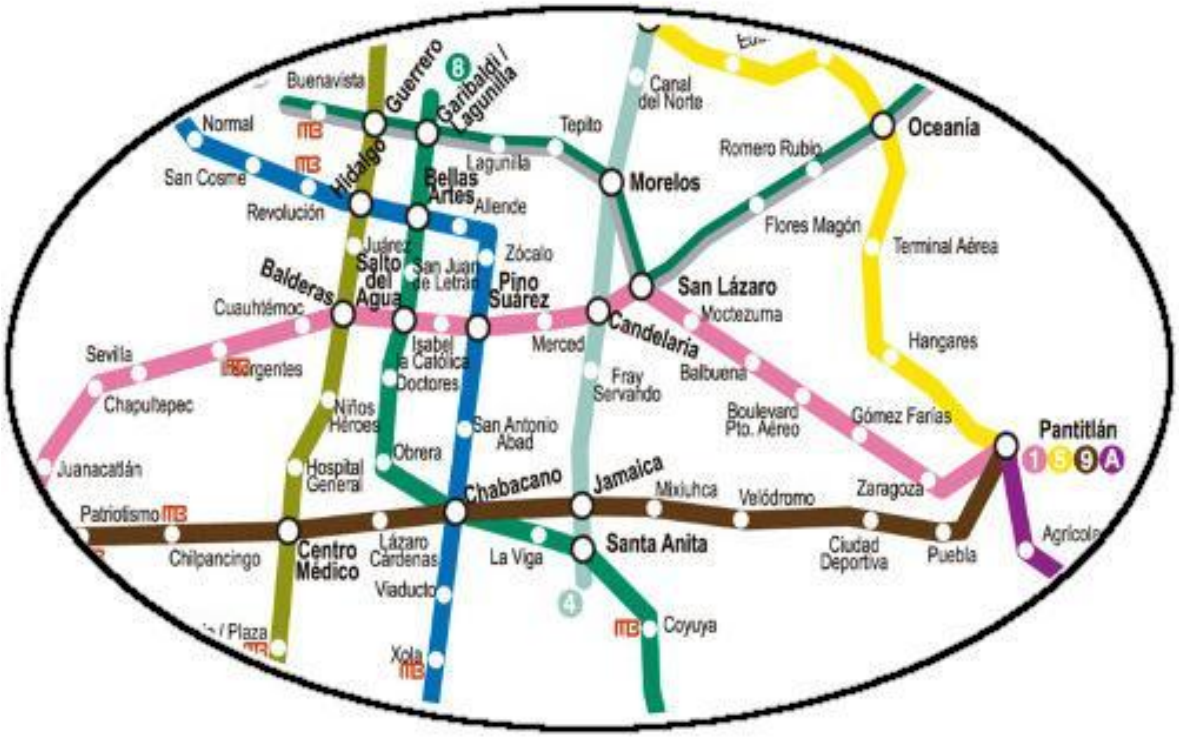


Fig. 1. Líneas del metro

Se describe a continuación un grafo de manera informal para efectos de introducir al lector al problema, sin embargo más adelante será detallado de manera formal.

Un grafo consiste en un conjunto de nodos o vértices unidos a través de aristas las cuales pueden o no llevar una dirección, esto implica una clasificación de los grafos en grafos dirigidos o no dirigidos, para nuestro caso de estudio trabajaremos exclusivamente con grafos no dirigidos (Fig. 3).

ACTIVIDAD	DEPENDENCIA	DURACIÓN
A	-	6
B	A	4
C	A	5
D	A	2
E	-	6
F	-	8
G	A,E	8

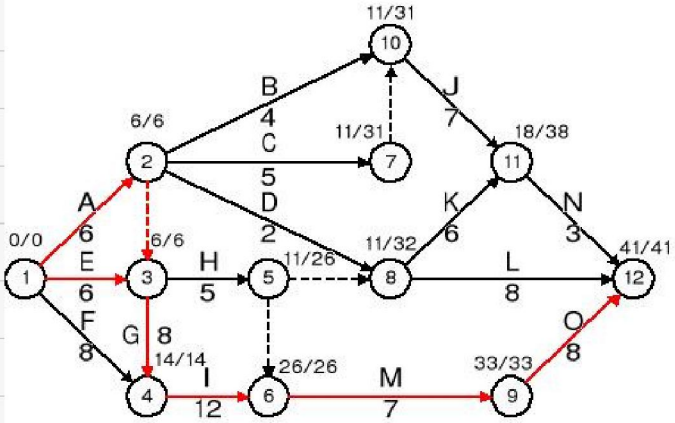


Fig. 2. Programación de actividades

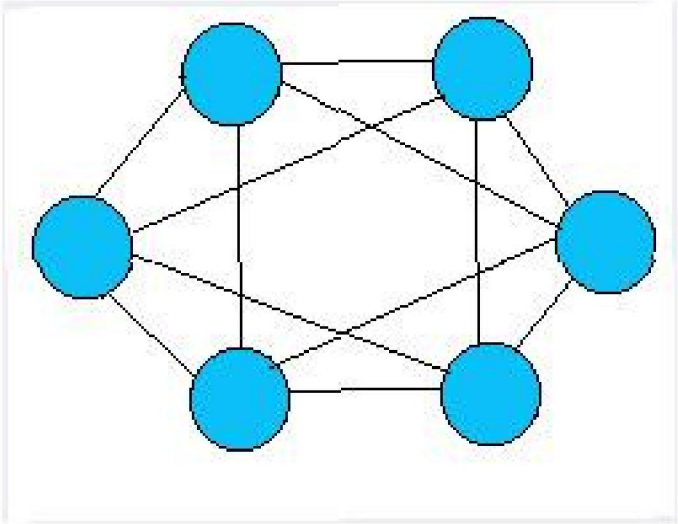


Fig. 3. Grafo no dirigido

1.1 Planteamiento del Problema

El problema del coloreo de grafos consiste en asignar o colorear los nodos de un grafo. La meta central en el coloreo de grafos consiste en colorear los nodos de forma que ningún par de nodos adyacentes (es decir dos nodos unidos por una arista) tengan el mismo color, a esto se le conoce como coloreo propio. El *k-coloreo* propio consiste en

colorear un grafo estrictamente con *k* colores sin asignar el mismo color a dos nodos adyacentes.

El problema sobre el cual se enfoca nuestra investigación e implementación es un algoritmo para poder colorear cualquier grafo no dirigido con la menor cantidad posible de colores. En la (Fig.4.) se muestra un grafo con un *3-coloreo* propio.

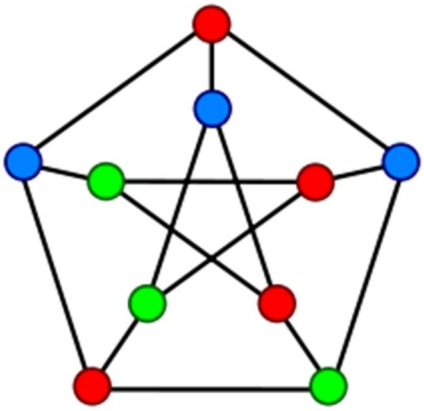


Fig. 4. Grafo coloreado con 3 colores

El número cromático de un grafo *G*, denotado como, $\chi(G)$ representa el número mínimo de colores con los cuales puede ser coloreado el grafo *G*.

El plantear el problema de coloreo de grafos es algo realmente sencillo, sin embargo su solución no lo es tanto, de hecho, determinar el número cromático $\chi(G)$ es uno de los problemas más difíciles de atacar, sobre todo por el tiempo computacional que se requiere para su solución.

A la fecha, determinar si $\chi(G) = 2$, es decir, que un grafo dado *G* puede ser coloreado propiamente solo con dos colores, es un problema que tiene una solución computacional que requiere tiempo de ejecución de orden polinomial, por lo tanto, se dice que tiene una solución computacionalmente eficiente. Sin embargo, al trabajar con grafos cuyo número cromático $\chi(G)$ es mayor o igual a 3, encontramos que hasta ahora, todas las propuestas para determinarlo no son eficientes, ya que el tiempo que requieren para ello es de complejidad exponencial.

Es por esto que se tiene el interés de dedicarse a realizar este trabajo de investigación y de aplicación, para atacar dicho problema que no tiene una solución eficientemente y que requiere de propuestas algorítmicas.

1.2 Estado del Arte

Leonhard Euler en 1736 trabajó sobre el problema de los puentes de Königsberg, el cual es considerado como uno de los primeros resultados de la teoría de grafos. Este problema consistía en encontrar un recorrido para cruzar a pie toda la ciudad, pasando solo una vez por cada uno de los puentes, y regresando al mismo punto de inicio [6]. También se considera uno de los primeros resultados topológicos en geometría (que no depende de ninguna medida). Este ejemplo ilustra la profunda relación entre la teoría de grafos y la topología (Fig.5.).

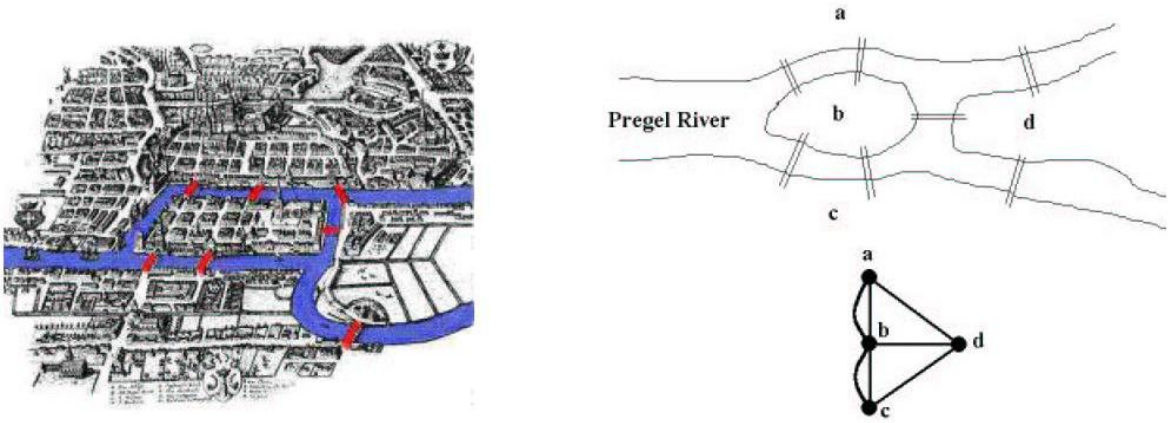


Fig. 5. Puentes de Königsberg

En 1845 Gustav Kirchhoff publicó sus leyes de los circuitos para calcular el voltaje y la corriente en los circuitos eléctricos [1] (Fig. 6.). En su planteamiento se hace un uso extensivo de la representación de circuitos eléctricos usando grafos.

Fig. 6. Circuitos

En 1852 Francis Guthrie plantea el problema de los cuatro colores [1], utilizando solo cuatro colores, colorear cualquier mapa de países de tal forma que dos países vecinos nunca tengan el mismo color. Este problema, que no fue resuelto hasta un siglo después por Kenneth Appel y Wolfgang Haken, puede ser considerado como el nacimiento de la teoría de coloreo de grafos. Al tratar de resolverlo, los matemáticos definieron términos y conceptos teóricos fundamentales de los grafos (Fig. 7).

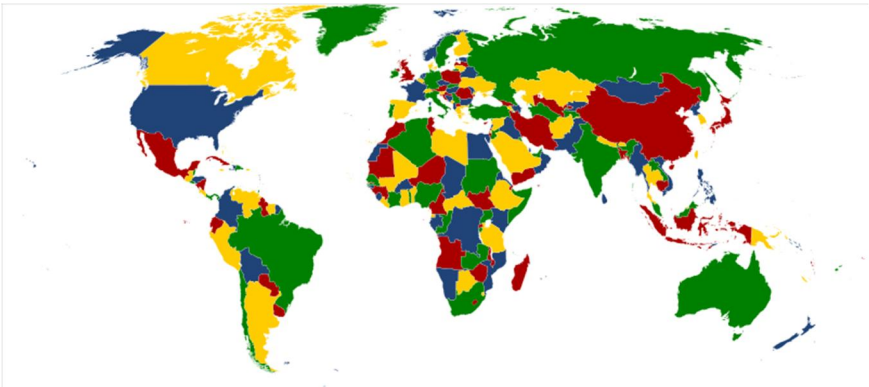


Fig. 7. Mapa de 4 colores

Determinar el número cromático $\chi(G)$ de un grafo G se presenta en una variedad de aplicaciones, y fue uno de los 22 problemas planteados como NP-completos en la lista de Karp [4]. La determinación de $\chi(G)$ es polinomialmente computable cuando $\chi(G) \leq 2$, pero cuando $\chi(G) \geq 3$, el problema se convierte en NP-completo, incluso para los grafos G con grado $\Delta(G) \geq 3$.

Posteriormente, se dedicó mucho esfuerzo en tratar de diseñar algoritmos de aproximación eficientes, es decir, dado un grafo *k-coloreable*, tratar de colorearlo de manera eficiente, y con l colores, $l \geq k$, y donde l es tan pequeño como sea posible. Y en esta misma línea, varias investigaciones se han realizado para determinar cuándo un grafo es 3-coloreable.

En el área de los algoritmos exactos para calcular el número cromático de un grafo, el primer algoritmo fue diseñado por Lawler con una complejidad en tiempo de $O(2.4423^n)$, los factores polinomiales son ignorados. Después de su presentación, el algoritmo de Lawler no fue mejorado durante 25 años [3].

Siguiendo la línea de algoritmos exactos y utilizando máximo conjuntos independientes para calcular el número cromático, Eppstein estableció un algoritmo con complejidad en tiempo de $O(2.4151^n)$ [3].

Posteriormente, Byskov proporciona un algoritmo con complejidad en tiempo de $O(2.24023^n)$ [4]. Todos los algoritmos utilizan una combinación de límites superiores basados en mejores aproximaciones del número de conjuntos independientes máximos de tamaño a lo más k, y algunos cambios en la manera de llenar la tabla de sub-soluciones en un enfoque de programación dinámica.

Las cotas superiores de tiempo más conocidas en la actualidad para el *k-coloreo*, son las siguientes: $O(1.3289^n)$ para $k = 3$. Para $k = 4$, $O(1.7504^n)$ el algoritmo Byskov, y para $k = 5$, $O(2.1020^n)$ del algoritmo de Byskov y Eppstein.

Aunque determinar el 3-coloreo de un grafo es un problema NP-completo, en la actualidad se han reconocido varios patrones topológicos en los grafos que permiten decidir la 3-coloreabilidad. Uno de los Primeros Resultados en esta línea, es el famoso

teorema de Grotzsch, que garantiza que cada grafo plano libre de triángulos es 3-coloreable. A partir de este trabajo pionero varias propuestas se han desarrollado a través de reconocer los patrones en el grafo para determinar la 3 coloración.

Por otro lado, para $k \leq 2$ el problema es solucionable polinomialmente, como es el problema general de coloreo para muchas clases de grafos tales como: grafos de intervalos, grafos acordes, grafos de comparación [2], el 3-coloreo de grafos libres de AT y, en general, para grafos perfectos [4]. En estos casos, las topologías especiales del grafo que se han encontrado, permiten identificar tanto a los tipos de grafos como también permiten diseñar algoritmos de tiempo polinomial para su coloreo.

1.3 Relevancia del Problema

Las aplicaciones del 3-coloreo de grafos son numerosas, a continuación se presentan solo algunos ejemplos.

Telefonía celular. El sistema de telefonía móvil está dispuesto a través de celdas, en este sistema se espera que los canales disponibles sean asignados de forma que cada celda tenga un grupo único de frecuencias y no haya colisiones entre células adyacentes. El coloreo de grafos aplicado en esta área puede ayudar a reducir el número de frecuencias cuando se diseña una red de telefonía para un área específica, esto por consecuencia nos conduce a un ahorro, al poder reutilizar frecuencias.

Programación de horarios. Este problema aplica para un sinfín de casos: asignar horarios de asesorías para alumnos, citas en dependencias gubernamentales, asignación de horarios para el uso de equipo por parte de trabajadores en empresas, asignación de tiempos para usar recursos o servicios por los usuarios de los mismos, etc. Cualquiera que sea el caso, la reducción a un problema de coloreo de grafos puede plantearse más o menos de la siguiente manera:

El grafo representa la forma en la que se asignan los horarios considerando que cada vértice de V es un recurso (computadora, profesor, salón, maquinaria) que puede asignarse, y las aristas indican los recursos que no pueden ser asignados al mismo tiempo. Por lo tanto los colores son los horarios que se asignarán a cada nodo considerando que no se puede asignar el mismo horario a dos recursos adyacentes.

Compiladores. Cuando un compilador calcula la serie de instrucciones para máquina abstracta, lo siguiente es la asignación de registros de la CPU a cada variable, para esto se toma en cuenta si se necesita cada una de las variables y cuando deja de utilizarse, esto nos permite optimizar la ejecución y acceder a memoria un menor número de veces posibles. Para reducir este problema a un coloreo de grafos, se genera una matriz de interferencias, la cual se traduce a un grafo de interferencias, donde cada vértice corresponde a una variable y las aristas unen variables que están vivas al mismo tiempo, es decir, variables que deben estar en registros distintos del CPU [16]. Por lo

tanto, el coloreo de tal grafo, dará como resultado la asignación del menor número de registros de la CPU a las variables.

Asignación de Vuelos. Si se quiere asignar vuelos a un determinado número de aviones donde un vuelo específico dura cierto intervalo de tiempo, debemos considerar que si dos vuelos se empalman no podemos asignarles el mismo avión. Representando el problema como un grafo, los vértices serían los vuelos, y las aristas unen vuelos que se empalman. Los colores serían los aviones que se asignan a los vuelos, obviamente al utilizar el menor número de aviones estamos reduciendo costos.

Como ya se habrá dado cuenta, al resolver el problema del coloreo de grafos, se abarca al mismo tiempo la solución de miles de problemas de la vida real que pueden ser modelados a través de un grafo. El coloreo de grafos es especialmente útil para situaciones de la vida real en las cuales se requiera optimizar recursos, es por ello que la relevancia de este problema es demasiada si consideramos que este tipo de situaciones son innumerables.

En la actualidad el estudio del coloreo de grafos se desarrolla de manera constante, esto se puede observar claramente en la cantidad de artículos publicados en diferentes congresos internacionales. Esta actividad también es evidente en varias universidades como Western Michigan University por poner solo un ejemplo, que cuentan en sus filas con académicos dedicados de lleno al estudio de grafos, y más específicamente al coloreo de los mismos.

1.4 Objetivos del Proyecto.

Desarrollar un sistema computacional que utilice un algoritmo eficiente aún y cuando sea con respuestas aproximadas para proponer soluciones al problema de coloreo, y así cualquier grafo no-dirigido que sea introducido en el sistema sea coloreado.

Objetivos Específicos:

Proponer y desarrollar un sistema que utilice el nuevo algoritmo propuesto, dicho sistema podrá recibir cualquier tipo de grafo no-dirigido y este deberá ser coloreado en base al algoritmo propuesto.

Automatizar un algoritmo de coloreo para poder obtener resultados en tiempo real, y así reducir el tiempo de coloreo.

Contribuir al área de ciencias de la computación en un problema que lleva años en ser estudiado, que no es fácil de resolver, y sobre todo, proporcionar un algoritmo eficiente para este problema.

II) Teoría de Grafos y Técnicas de Búsqueda.

La Teoría de Grafos es un área comprendida tanto en matemáticas como en ciencias de la computación. Estudia una estructura matemática especial llamada grafo, así como sus propiedades. Un grafo es un conjunto de nodos o vértices unidos por medio de aristas, estas pueden (o no) ser dirigidas. Generalmente se representa un grafo como puntos unidos por líneas [8].

2.1 Conceptos Preliminares de la Teoría de Grafos.

Sea $G = (V, E)$ un grafo no dirigido simple (es decir, finito, sin loops y sin aristas múltiples) con conjunto de vértices (o conjunto de nodos) V y conjunto de aristas E. $E(G)$ y $V(G)$ son usados para enfatizar los conjuntos de aristas y vértices de un grafo determinado G. Dos nodos v y w que pertenecen a V se llaman adyacentes si hay una arista $\{v, w\} \in E$, uniéndolos.

El vecindario de $x \in V$ es $N(x) = \{y \in V: \{x, y\} \in E\}$ y su vecindario cerrado es $N(x) \cup \{x\}$, que se denota por $N[x]$. Tómese en cuenta que x no esta en $N(x)$.

Se denota la cardinalidad de un conjunto A, por $|A|$. Dado un grafo $G = (V, E)$, el grado de un vértice $x \in V$, denotado por $\delta(x)$, es $|N(x)|$. El tamaño del vecindario de x, $\delta(N(x))$, es $\delta(N(x)) = \sum_{y \in N(x)} \delta(y)$. El grado máximo de G, o simplemente el grado de G es $\Delta(G) = max\{\delta(x): x \in V\}$, mientras que se denota el grado mínimo con $\delta min(G) = min\{\delta(x): x \in V\}$ y con $\delta(G) = (2 \cdot |E|) / |V|$ el grado promedio del grafo.

Dado un subconjunto de vértices $S \subseteq V$ el subgrafo de G denotado por $G|S$ tiene a S como conjunto de vértices y como conjunto de aristas a $E(G|S) = \{\{u, v\} \in E: u, v \in S\}$. A $G|S$ se le llama el subgrafo de G inducido por S. Escribimos $G\text{-}S$ para denotar el grafo $G|(V\text{-}S)$. El subgrafo inducido por $N(v)$ se denota como $H(v) = G|N(v)$ que tiene a $N(v)$ como el conjunto de nodos y todas las aristas entre ellos.

Un camino de un vértice v al vértice w en un grafo es una secuencia de aristas: $v_0, v_1, v_2, \ldots, v_{n-1}, v_n$. De modo que $v = v_0$, $v_n = w$, v_k es adyacente a v_{k+1} y la longitud de la ruta es n. Un camino simple es un camino en el que $v_0, v_1, \ldots, v_{n-1}, v_n$ son todos vértices distintos. Un ciclo es un camino no vacío tal que los vértices primero y el último son iguales, y un ciclo simple es un ciclo en el que ningún vértice se repite, a excepción de los vértices inicial y final.

Un k-ciclo es un ciclo de longitud k, es decir, un k-ciclo tiene k aristas. Un ciclo de longitud impar se llama un ciclo impar, mientras que un ciclo de longitud par se llama un ciclo par. Un grafo G es acíclico si no tiene ciclos.

Un grafo completo de n nodos tiene $n\cdot(n\text{-}1)/2$ aristas distintas, se denota Kn el grafo completo de n nodos. Un grafo G es un grafo regular si todos los vértices tienen el mismo grado, G es k-regular si es regular, de grado k.

Un componente conectado de G es un subgrafo máximo inducido de G [7], es decir, un subgrafo conexo que no es un subgrafo propio de cualquier otro subgrafo conexo de G. Note que en un componente conectado, para cada par de vértices x, y, hay un camino de x a y. Si un grafo acíclico también es conectado, entonces se llama un árbol libre.

Una coloración de un grafo $G = (V, E)$ es una asignación de colores a sus vértices. Un coloreo es propio si los vértices adyacentes siempre tienen colores diferentes. Un k-coloreo de G es una asignación de V al conjunto $\{1, 2,... ,k\}$ de k "colores".

El número cromático de G denotado por $\chi(G)$ es el valor mínimo k tal que G tiene un k-coloreo propio. Si $\chi(G) = k$, G se dice entonces que es k-cromático. Determinar el valor de $\chi(G)$ es computable polinomialmente cuando $\chi(G) \leq 2$, pero cuando $\chi(G) \geq 3$, el problema es NP-completo, incluso para grafos G con grado $\Delta(G) \geq 3$.

Sea $G = (V, E)$ un grafo, G es bipartito si V se puede dividir en dos subconjuntos U_1 y U_2, llamados conjuntos partitos, de manera que todas las aristas de G se unen a un vértice de U_1 con un vértice de U_2.

Dado un grafo $G = (V, E)$, $S \subseteq V$ es un conjunto independiente en G si para cada dos vértices v_1, v_2 en S, se cumple que $\{v_1, v_2\} \notin E$. Sea $I(G)$ el conjunto de todos los conjuntos independientes de G. Un conjunto independiente $S \in I(G)$ es maximal, si no es un subconjunto de cualquier conjunto independiente más grande y es máximo si tiene el tamaño más grande de todos los conjuntos independientes en $I(G)$.

Si $G = (V,E)$ es un grafo k-cromático, entonces es posible particionar V en k conjuntos independientes $V_1, V_2, ..., V_k$, llamados clases de color, pero no es posible particionar V en k-1 conjuntos independientes.

2.2 Recorrido en Amplitud.

El recorrido en amplitud comienza por un vértice $v \in V$, este se marca como visitado; posteriormente se almacenan los vértices vecinos que no estén visitados de v, es decir, $P \subset N(v)$. En caso de que exista una arista que nos conduzca a un nodo previamente visitado, entonces se etiqueta como arista de cruce. Después se selecciona el siguiente vértice que no este visitado y que tengamos almacenado; continuamos con este procedimiento mientras exista al menos un vértice sin visitar, en la (Fig.8.) se muestra el algoritmo para realizar dicho recorrido.

```
bfs( v )
Entrada: v ∈ V
Salida: T_G
 1: QUEUE C = ∅
 2: C.Inserta( v )
 3: mientras not( C.Vacia() ) hacer
 4:    v = C.saca()
 5:    V_TG = V_TG ∪ {v}
 6:    Visita v
 7:    para w ∈ N(v) hacer
 8:       si w no Visitado entonces
 9:          E_TG = E_TG ∪ {(v, w)}
10:          C.Inserta( w )
11:       si no
12:          E'_TG = E'_TG ∪ {(v, w)}
13:       fin si
14:    fin para
15: fin mientras
16: devolver
```

Fig. 8. Algoritmo para el recorrido en amplitud de un grafo

En la (Fig. 9) se observa el resultado del recorrido en amplitud de un grafo, en la parte izquierda de la figura se muestra el grafo *G* y en la parte derecha se observa el recorrido en amplitud de dicho grafo.

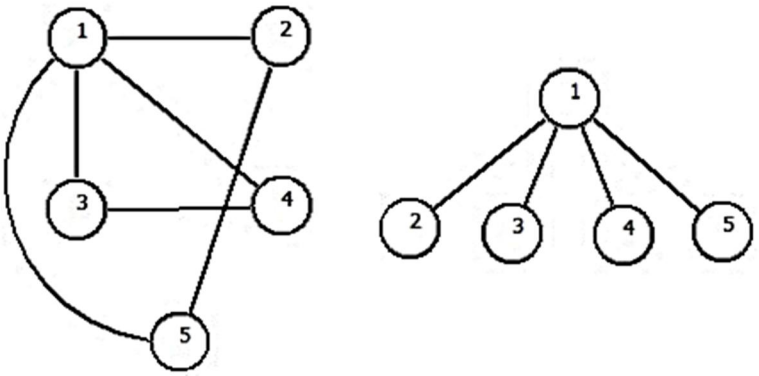


Fig. 9. Recorrido en amplitud de un grafo

2.3 Recorrido a lo Profundo.

Una idea simple de como considerar el recorrido a lo profundo, es que se comienza el recorrido en el vértice inicial (vértice con índice 1), el cual se marca como vértice activo.

Se continúa visitando nodos no visitados hasta que todos los vértices del grafo hayan sido visitados, en cada paso se avanza al vecino con el menor índice siempre que se pueda, pasando este a ser el vértice activo. Cuando todos los vecinos al vértice activo hayan sido ya visitados, se retrocede al vértice *X* desde el que se alcanzó el vértice activo y se prosigue siendo ahora *X* el vértice activo, el algoritmo queda como se muestra en la (Fig.10).

dfs(v)
Entrada: $v \in V$
Salida: T_G
1: Visita v
2: $V_{T_G} = V_{T_G} \cup \{v\}$
3: **para** $w \in N(v)$ **hacer**
4: **si** w no Visitado **entonces**
5: $E_{T_G} = E_{T_G} \cup \{(v, w)\}$
6: dfs(w)
7: **si no**
8: $E'_{T_G} = E'_{T_G} \cup \{(v, w)\}$
9: **fin si**
10: **fin para**
11: **devolver**

Fig. 10. Algoritmo para el recorrido en profundidad

Este procedimiento tiene una complejidad de tiempo de *O(m+n)* donde *n* y *m* son el número de nodos y el número de aristas de entrada del grafo *G*.

Tomemos como ejemplo el siguiente grafo de la (Fig.11) para mostrar como es el recorrido en profundidad:

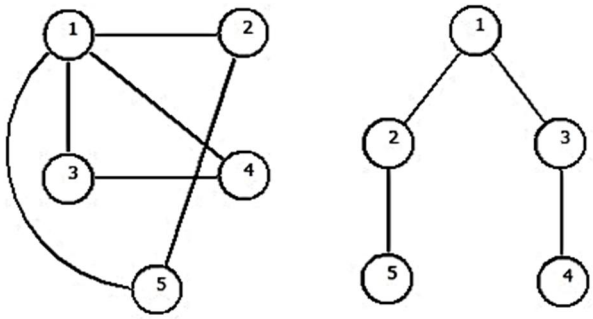


Fig. 11. Recorrido a profundidad de un grafo

Se comienza a partir del primer nodo y se recorre el grafo abarcando los nodos adyacentes de tal forma que se regrese a cada nodo padre solo después de haber visitado los nodos más profundos, quedando el árbol recorrido en el siguiente orden (Fig.11).

2.3.1 Árbol Generador y Ciclos básicos.

El resultado de la búsqueda a lo profundo genera un árbol que denotaremos con T_G. T_G se forma por los nodos y las aristas de árbol cuando se aplica el procedimiento dfs (búsqueda a lo profundo) sobre el grafo G. Dado un grafo no dirigido conexo G = (V, E) el aplicar una búsqueda en profundidad para recorrer G se produce T_G, donde V(T_G) = V(G). Las aristas de T_G se llaman aristas de árbol, mientras que las aristas E(G)-E(T_G) se llaman aristas de retroceso.

Sea T el árbol generador de un grafo conexo G= (V,E). Cualquier arista {v,w} que pertenece a $E(G - T)$ que se agregue a T formará un ciclo único que se denota como $C_{v,w}$. Donde v y w se conectan por un camino único que denotaremos como $P_{v,w}$. en T. De esta forma $P_{v,w}$+ {v,w} forman el ciclo $C_{v,w}$ el cual llamaremos ciclo básico de v a w , con respecto al árbol generador T. Entonces, sea $e \in E(G) - E(T_G)$ una arista de retroceso dada. La unión de la ruta en T_G entre los extremos finales de e con la misma arista e forma un ciclo simple, un ciclo de este tipo se llama ciclo básico (o fundamental) de G con respecto a T_G (Fig.12). Cada arista de retroceso tiene el máximo trayecto contenido en el ciclo básico del que forma parte. Llamamos nodo final de un ciclo, al nodo donde se encontró una arista de retroceso durante la búsqueda en profundidad.

A partir de los ciclos básicos del árbol T_G podemos generar cualquier ciclo que aparece en G. De hecho, cualquier ciclo que se pueda formar en G, es un ciclo que se forma al realizar uniones simétricas entre ciclos básicos.

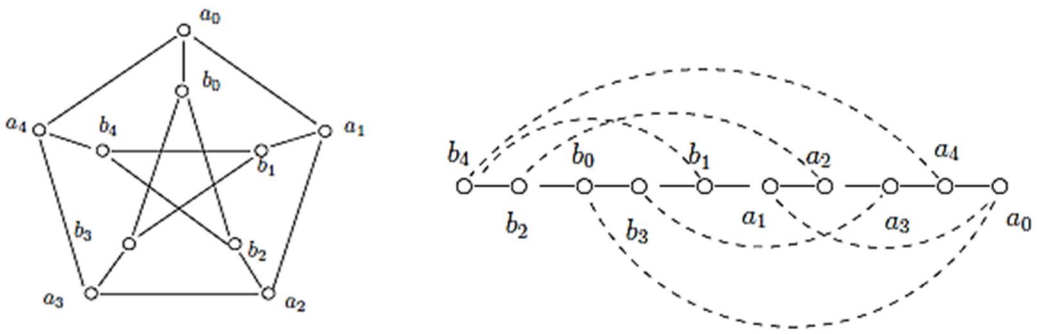


Fig. 12. Grafo de Petersen con su respectivo árbol generador

Las aristas de retroceso que sean encontradas durante la búsqueda a lo profundo darán como resultado un ciclo básico. (Fig. 13).

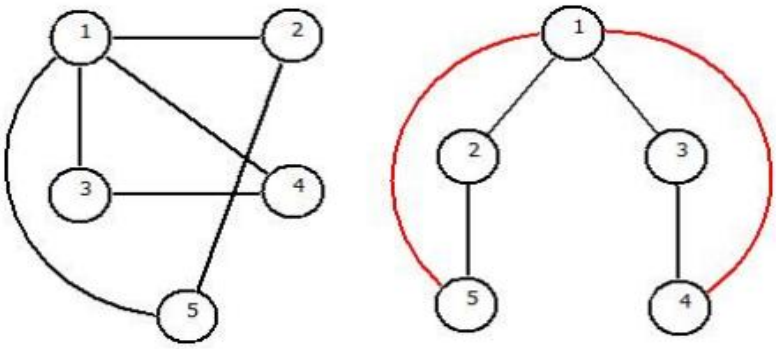


Fig. 13. Ciclo básico de un grafo

2.4 Ciclos Intersectados y Compuestos.

Sea $C = \{C_1, C_2, ..., C_k\}$ el conjunto de los ciclos básicos encontrados durante la búsqueda en profundidad en G. Dado un par de ciclos básicos C_i y C_j del grafo G, si C_i y C_j comparten al menos una arista se les llama ciclos intersectados, de lo contrario se les llama ciclos independientes.

En particular, si dos ciclos comparten una sola arista se llaman adyacentes. Se dice que un ciclo básico $C_i \in C$ es independiente si C_i y $C_j \in C$ son disjuntos para cada $j<>i$.

En la (Fig. 14.), en la parte izquierda se muestra un grafo G y en la parte derecha se muestra su recorrido en profundidad de G, las aristas de retroceso son marcadas en color rojo, por consiguiente existe 3 ciclos $C_1=\{v_1, v_2, v_5, v_1\}$, $C_2=\{v_1, v_2, v_6, v_1\}$, $C_3=\{v_1, v_3, v_4, v_1\}$

El ciclo C_1 y C_3, así como C_2 y C_3 son independientes, pero el ciclo C_1 y C_2 son ciclos intersectados.

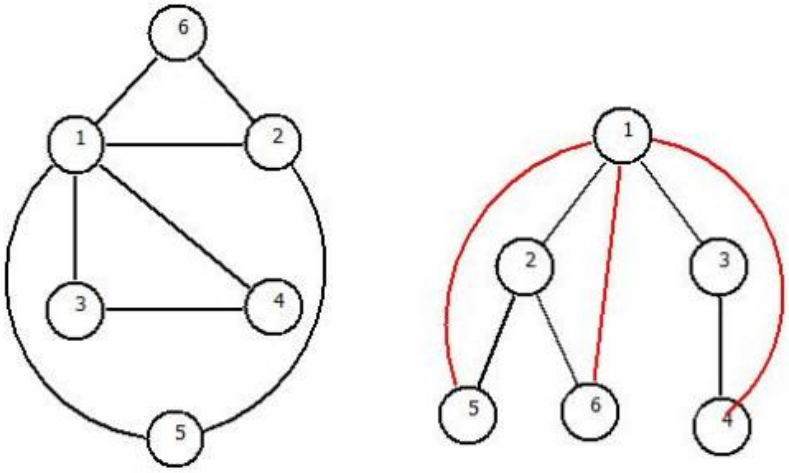


Fig. 14. Ciclos intersectados e independientes

La operación de diferencia simétrica entre dos ciclos intersectados: $C_i \Delta C_j$ forma un nuevo ciclo. Δ denota la operación diferencia simétrica entre el conjunto de aristas de ambos ciclos, que consiste en unir las aristas de ambos ciclos y restarle a esta unión las aristas comunes de ambos ciclos. Así, $C_i \Delta C_j = (E(C_i) \cup E(C_j)) - (E(C_i) \cap E(C_j))$.

Un ciclo compuesto se forman al aplicar el operador Δ entre cada par de ciclos básicos intersectados: $C_i, C_j \in C$. El resultado de aplicar la operación Δ construye un nuevo ciclo compuesto que contiene las aristas de C_i y C_j excepto las aristas que comparten (Fig.15). En el inciso (a) se muestran dos ciclos básicos C1 y C2 intersectados y en el inciso (b) se muestra el ciclo compuesto generado al aplicar el operador Δ a C1 y C2.

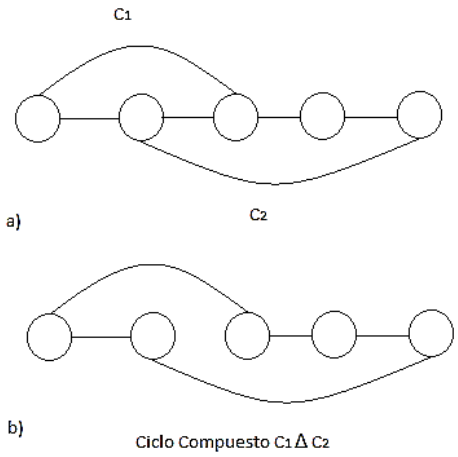


Fig. 15. a) ciclos básicos, b) ciclo compuesto

Adelantando un poco el tema del coloreo se sabe que cualquier grafo acíclico es 2-coloreable y por lo tanto, 3-coloreable. Además, si un grafo *G* tiene sólo ciclos de longitud par entonces *G* es 2-coloreable ya que alternar dos colores con respecto a los niveles del árbol T_G se construye un 2-coloreo propio.

Ciclos de longitud impar requieren por lo menos 3 colores para tener un coloreo propio. Por lo tanto, las estructuras de subgrafos que generan conflicto para el 3-coloreo, son los ciclos impares.

2.5 Puntos de Articulación

Es de suma importancia mencionar este tema ya que será usado para poder realizar nuestra propuesta de algoritmo en este proyecto.

Un punto de articulación de un grafo es un vértice *v* tal que cuando se elimina junto con todas las aristas incidentes sobre él, se divide un componente conexo en dos o más partes. Por ejemplo, los puntos de articulación de la (Fig.16) son los nodos *a* y *c*. Si se elimina *a*, el grafo, que es un componente conexo, se divide en dos triángulos: {*b,d,e*} y {*c,f,g*}; si se elimina *c* se divide en {*a,b,d,e*} y {*f,g*}, sin embargo al eliminar cualquier otro vértice del grafo de la Fig.16 (*a*), el componente conexo no se dividirá. Un grafo sin puntos de articulación se llama biconexo. La búsqueda en profundidad es muy útil para encontrar los componentes biconexos de un grafo [15].

El problema de encontrar los puntos de articulación es el más simple de muchos problemas importantes relacionados con la conectividad de grafos. Como ejemplo de las aplicaciones de los algoritmos de conectividad, se puede presentar una red de comunicaciones como un grafo en el que los vértices son lugares que hay que mantener comunicados entre sí. Un grafo tiene conectividad *k* si la eliminación de *k-1* vértices cualesquiera no lo desconecta. Por ejemplo, un grafo tiene conectividad dos o más si y solo si, no tiene puntos de articulación, es decir, si y solo si es biconexo.

Cuanto mayor sea su conectividad, tanto más fácil será que las comunicaciones del grafo sobrevivan al fallo de alguno de sus vértices, sea por fallo de las unidades de procesamiento colocadas en los vértices o sea por motivos externos.

Ahora se presenta un algoritmo simple de búsqueda en profundidad para encontrar todos los puntos de articulación de un grafo, y así probar por medio de la ausencia de puntos de articulación si el grafo es biconexo.

Se realiza una búsqueda en profundidad del grafo, calculando el *número_bp*[*v*] para todo vértice *v*. En esencia, el *número_bp* ordena los vértices como un recorrido en orden previo del árbol abarcador en profundidad.

Para cada vértice *v*, obtener bajo [*v*], como el *número_bp* más pequeño de *v* o de cualquier otro vértice *w* accesible desde *v*, siguiendo cero o más aristas de árbol hasta un descendiente *x* de *v* (*x* puede ser *v*) y después seguir una arista de retroceso (*x,w*). Se calcula *bajo*[*v*] para todos los vértices *v*, visitándolos en un recorrido en orden posterior. Cuando se procesa *v*, se ha calculado *bajo*[*y*] para todo hijo *y* de *v*. se toma *bajo*[*v*] como el mínimo del *número_bp*[*v*], *número_bp* [*z*] para cualquier vértice *z* para el cual haya una arista de *retroceso*(*v,z*), y *bajo*[*y*] para cualquier hijo *y* de *v*.

Ahora se encuentran los puntos de articulación como sigue.

La raíz es un punto de articulación si y sólo si tiene dos o más hijos. Puesto que no hay aristas cruzadas, la eliminación de la raíz debe desconectar los subárboles cuyas raíces se encuentren en sus hijos, como *a* desconecta {*b,d,e*} de {*c,f,g*} en la Fig.16.

Un vértice *v* distinto de la raíz es un punto de articulación si y solo si hay un hijo *w* de *v* tal que *bajo*[*w*] ≥ *número_bp*[*v*]. En este caso, *v* desconecta *w* y sus descendientes del resto del grafo. A la inversa, si *bajo*[*w*] <*número_bp*[*v*], debe haber un camino para descender desde *w* en el árbol y regresar hasta un antecesor propio de *v* (el vértice cuyo *número_bp* es *bajo*[*w*]) y, por tanto, la eliminación de *v* no desconecta *w* ni sus descendientes del resto del grafo [15].

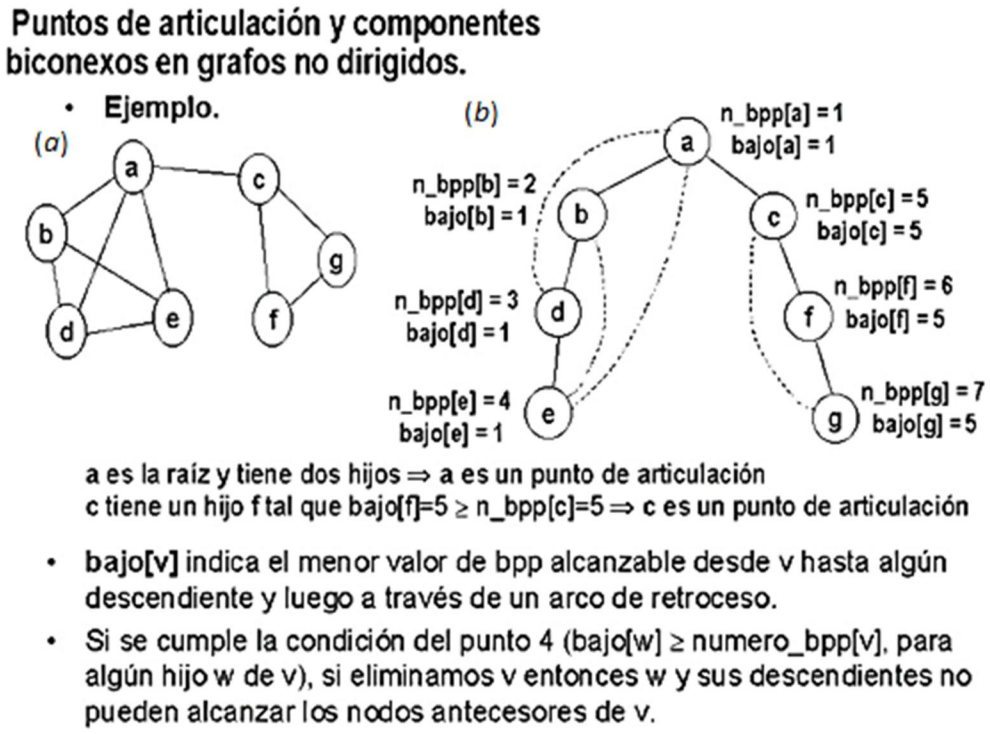


Fig. 16. Puntos de Articulación

En la Figura 16 se observa un ejemplo de puntos de articulación que después de obtener cada valor bajo, se verifica cada vértice. La raíz *a* es un punto de articulación porque tiene dos hijos. El vértice *c* es un punto de articulación *f* con *bajo*[*f*] ≥ *número_bp*[*c*]. Los otros vértices no son puntos de articulación.

Ahora mostramos otra forma similar de obtener los puntos de articulación y la cual fue utilizada para implementar nuestro proyecto, los puntos de articulación pueden detectarse al hacer el algoritmo de búsqueda en profundidad. Este algoritmo crea un árbol

con aristas de avance y aristas de retroceso. Un vértice *v* de este árbol es un punto de articulación si tiene cuando menos un subárbol sin conectar con cualquiera de sus predecesores por medio de una arista de retroceso; como es un árbol, es casi seguro que ninguno de los predecesores de *v* es alcanzable desde cualquiera de sus sucesores por un enlace de avance. Por ejemplo, el grafo de la Fig. 17a se transforma en un árbol de búsqueda primero en profundidad (Fig. 17c), y este árbol tiene cuatro puntos de articulación*, b, d, h,* e *i* debido a que no hay arista de retroceso desde cualquier nodo debajo de *d* a cualquier nodo arriba del mismo en el árbol, y no hay arista de retroceso desde cualquier vértice en el subárbol derecho de cualquier vértice arriba de *h*. Pero el vértice *g* no puede ser un punto de articulación porque su sucesor *h* está conectado a un vértice arriba de él. Los cuatro vértices dividen al grafo en cinco bloques indicados en la Fig. 17c por líneas punteadas [12].

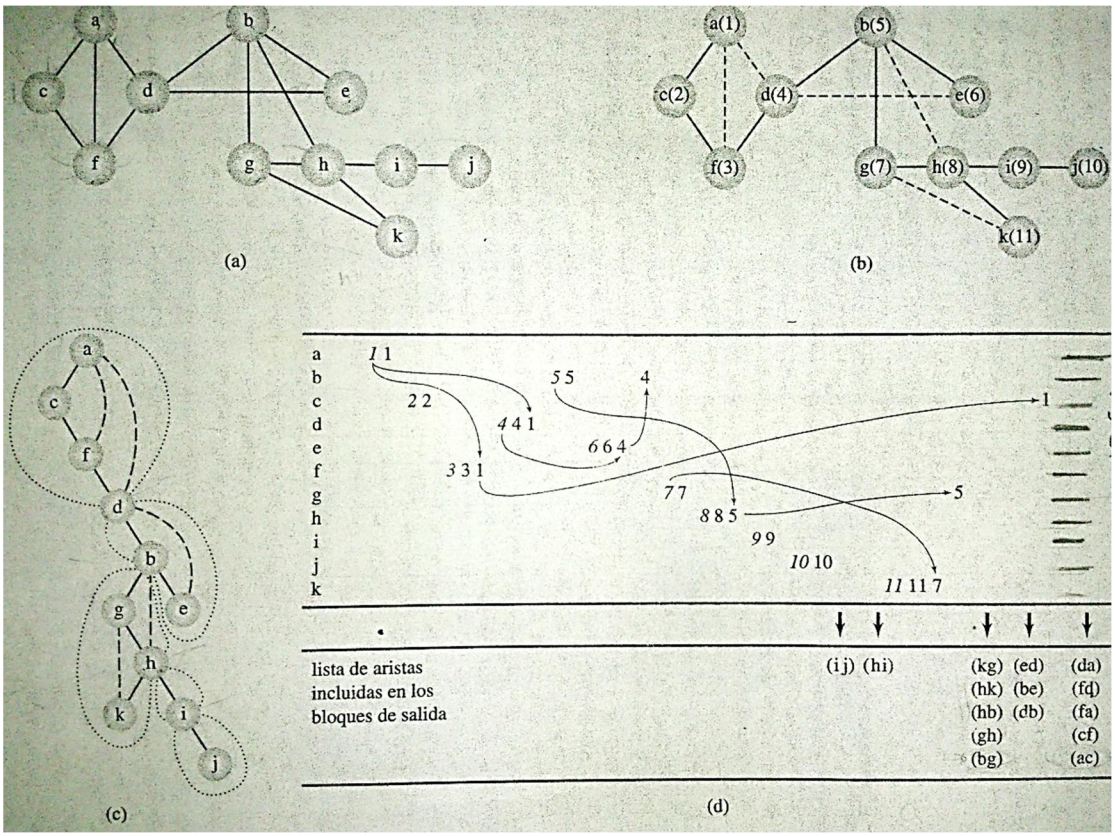


Fig. 17. Búsqueda de Bloques y puntos de articulación

Para encontrar puntos de articulación, se usa un parámetro *pred(v)* definido como *min(núm(v), núm(u1)…. núm(uk))*, donde $u_1 \ldots u_k$ son los vértices conectados por una arista de retroceso con un descendiente de *v* o con *v* mismo. Como entre mayor sea un predecesotr de *v*, su número será menor, elegir un número mínimo significa elegir el predecesor mayor de *v*. Para el árbol de la Fig. 17c, *pred(c)=pred(d)=1, pred(b)=4* y

pred(k)=7, a continuación se muestra el algoritmo, el cual utiliza una pila para almacenar todas las aristas procesadas.

```
BloqueBusquedaProfundidad(v)
        pred(v) = núm(v)=i++;
para todos los vertices u adyacenrtesav
                si arista (uv) no se ha procesado
                        mete(arista(uv))
                sinúm (u) es 0
                        BloqueBusquedaProfundidad(u)
                        sipred (u) ≥ núm (v)
                                e=sacar();
                                mientras e / arista(vu)
                                        mostrar e;
                                        e=sacar();
                                imprime e;
                        elsepred(v) = mín(pred(v), pred (u));
                else si u no es el padre de v
                        pred(v)=min(pred(v),núm(u));

bloqueBusqueda()
        para todos los vértices v
                núm(v)=0;
        i=1;
mientras exista un vértice v tal que núm(v)==0
        BloqueBusquedaProfundidad(v);
```

Un ejemplo de la ejecución de este algoritmo se muestra en la Fig. 17*d* según se aplicó al grafo de la Fig.17a. La tabla lista todos los cambios en *pred (v)* para todos los

vértices *v* procesados por el algoritmo, y las flechas muestran la fuente de los nuevos valores de *pred(v)*. Para cada vértice *v bloqueBusquedaProfundidad(v)* primero asigna dos numeros: *núm(v)*, mostrado en itálicas, y *pred(v)*, el cual puede cambiar durante la ejecución de *bloqueBusquedaProfundidad(v)*. Por ejemplo, *a* se procesa primero con *núm(a)* y *pred(a)* se establece en 1. *arista(ac)* se inserta en la pila como *núm(c)* es 0, el algoritmo se invoca para *c*. En este punto, *núm(c)* y *pred(c)* se establecen en 2. A continuación el algoritmo se invoca para *f*, un descendiente de *c*, así que *núm(f)* y *pref(f)* se establecen en 3 , y luego este se invoca para *a*, un descendiente de *f*. Debido a que *núm(a)* no es 0 y *a* no es el padre de *f*, *pred(f)* se establece en 1 = *min(pred(f),núm(a))= mín(3,1).*

Este algoritmo imprime las aristas en los bloques detectados y estas aristas se muestran en la Fig. 17d en el momento que se imprimieron después de extraerlos de la pila.

2.6 Clases de Complejidad: P y NP

Por conveniencia, la teoría de la complejidad está diseñada especialmente para aplicarse a problemas de decisión. Un problema de decisión PD [4] se plantea como una pregunta general, la cual acepta como respuesta sólo una de dos posibilidades, la respuesta 'SI' o la respuesta 'NO'. En forma abstracta, un problema de decisión, puede describirse como:

PD :<*D, S*>, donde: *D* - Dominio de valores posibles y $S \subseteq D$ - son las instancias que resuelven el problema. El planteamiento del *PD* es:

$$\forall x \in D\text{: } \mathrm{PD}(x) = \begin{cases} \text{Sí} & \text{si } x \in S \\ \text{No} & \text{en otro caso} \end{cases}$$

En estos términos, un *PD* puede usarse como medio para reconocer un *lenguaje S* que es un subconjunto de palabras de un alfabeto *D*. En la práctica, el *PD* intenta reconocer algún conjunto específico de objetos matemáticos, tales como: fórmulas lógicas verdaderas, gráficas que tienen una cierta propiedad, etc.,.... Pero esto requiere el codificar adecuadamente las estructuras subyacentes de los objetos para cualquier instancia del problema, de tal forma que finalmente, se pueda construir un predicado que aún sólo con respuestas 'SI' o 'NO' obtenga cualquier otra información que se desee.

El problema de decisión que abordamos en este trabajo de tesis, es el de coloración de un grafo.

Instancia: Una gráfica $G = (V,E)$, donde V es el conjunto de nodos o vértices y E es el conjunto de aristas entre los vértices. Una cota $k \in \mathbb{Z}+$, con $k \leq |V|$.

Pregunta: ¿ Será G k-coloreable ?, es decir, existirá una función $f: V \rightarrow \{1,2,...,k\}$ tal que $f(u) \neq f(v)$ siempre que $\{u,v\} \in E$?

Un algoritmo es un procedimiento general que trabaja paso a paso y en un número finito de estos, resuelve un problema. Un algoritmo resuelve un problema de decisión PD si puede aplicarse a cualquier instancia *I* del PD y garantiza que siempre produce una solución para esa instancia. Podemos pensar en un algoritmo como un programa de computadora o como la descripción de la función de transición de una máquina de Turing.

En general, nos interesa encontrar el algoritmo más eficiente que resuelve un problema dado. En el sentido más amplio, la noción de eficiencia tiene que ver con los varios recursos computacionales necesarios para ejecutar el algoritmo. Aunque el recurso dominante es el del tiempo, por tal, normalmente el algoritmo más eficiente que resuelve un problema PD es aquel que se ejecuta más rápidamente de entre todos los algoritmos que resuelven el mismo problema [4].

Los requerimientos de tiempo en la ejecución de un algoritmo se expresan en términos de una sola variable: la longitud o tamaño de las instancias *I* del problema, que refleja la cantidad de datos de entrada y la magnitud de cada uno de estos datos.

Dada una instancia *I* de un problema de decisión PD con m datos de entrada; (d_1, ..., d_m) se define la longitud de la instancia como: $long(I) = \sum_{i=1}^{m} long(d_i)$, donde *long*(di) es la longitud de cada uno de los datos de entrada.

Por ejemplo, si se tuviera una instancia *I* del problema del coloreo de un grafo, se requiere codificar el conjunto de nodos $V=\{u_1,\ldots,u_n\}$ y el conjunto de aristas $E=\{a_1,\ldots,a_m\}$ del grafo *G* de entrada. Se asume dada una instancia específica, que hay *n* nodos y *m* aristas, por lo que se requieren *m+n* etiquetas para diferenciar los nodos y aristas. Suponiendo que *long*(*x*) es el número de bytes que se requiere para codificar el dato *x* en el programa, se tendría que el tamaño de la instancia del problema es:

long(*I*) = $\sum_{i=1}^{n} long(u_i) + \sum_{j=1}^{m} long(a_j) + \sum_{j=1}^{n+m} long(etiquetas_j)$

La función de complejidad de tiempo de un algoritmo, expresa los requerimientos de tiempo que un determinado modelo de computación necesita al ejecutar el algoritmo para resolver cada posible instancia del problema. Con el fin de clasificar el esfuerzo computacional que se requiere al resolver los problemas de decisión, estos problemas se han clasificado en clases de complejidad. Una misma clase de complejidad contiene a todos los problemas que requieren funciones similares de tiempo para ser resueltos. Denotando a un algoritmo determinista como *AD* y a un algoritmo no-determinista como *AnD*, podemos definir las siguientes clases de complejidad:

DLOG = $\{PD \mid \exists\ AD$ que resuelve PD usando espacio logarítmico$\}$

P = $\{PD \mid \exists\ AD$ que resuelve PD en tiempo polinomial$\}$

NP $\{PD \mid \exists\ AnD$ que resuelve PD en tiempo polinomial$\}$

EXP = $\{PD \mid \exists\ AD$ que resuelve PD en tiempo exponencial$\}$

Las clases P y NP las podemos también identificar de la siguiente manera:

- P es la clase de problemas que poseen algoritmos deterministas de resolución que tardan tiempo polinomial.
- NP es la clase de problemas que tienen un algoritmo determinista de resolución que corre en tiempo exponencial, pero para los cuales también existe un algoritmo no determinista que corre en tiempo polinomial.

A los problemas de la clase P, se les reconoce como problemas tratables, puesto que para cualquiera de sus instancias, éstas se resuelven por algoritmos que corren en un tiempo acotado de cómputo. Se dice que un problema no ha sido bien resuelto hasta que es hallado un algoritmo determinista de tiempo polinomial que lo resuelve [4]. Se le asigna a un problema el término de intratable, si este se ha mostrado tan difícil que no se ha encontrado algoritmo determinista con complejidad polinomial en tiempo que lo resuelva, es decir, no se ha hallado algoritmo eficiente que lo resuelva. Entonces una primera clase de complejidad que contiene problemas intratables, es la clase NP.

Todo algoritmo cuya función de complejidad de tiempo, no pueda acotarse por una función polinomial, se dice que es un algoritmo de complejidad exponencial en tiempo, aún cuando debe notarse, que esta definición incluye ciertas funciones de complejidad de tiempo, tales como, $n * log(n)$, las cuales normalmente no son consideradas como funciones exponenciales.

El término intratable, refleja el punto de vista de que los algoritmos de tiempo exponencial no son considerados 'buenos' algoritmos y que generalmente tal clase de algoritmos refleja variaciones de búsquedas exhaustivas, mientras que algoritmos de tiempo polinomial generalmente son construidos sólo a través de explotar la estructuras internas e inherentes del problema.

Existen algoritmos de tiempo exponencial que han sido muy útiles en la práctica. La complejidad de tiempo de un algoritmo tal y como la hemos definido, es una medida para el peor de los casos, y el hecho de que un algoritmo tenga complejidad exponencial, significa que existe al menos una instancia del problema, que requiere mucho tiempo, aún y cuando muchas de las instancias del mismo problema requieran en la práctica de mucho menos tiempo que un valor exponencial para ser resueltos.

Aún y cuando ciertos algoritmos de complejidad exponencial para determinados problemas tienen un buen comportamiento en la práctica, esto no ha detenido el avance de las investigaciones por buscar algoritmos de tiempo polinomial que resuelvan los mismos problemas. Y de hecho, el gran éxito de tales algoritmos exponenciales, lleva a la sospecha de que hace falta capturar alguna propiedad crucial del problema cuyo refinamiento pueda llevarnos a mejores métodos. Un área de gran interés es sobre técnicas generales que permitan diseñar algoritmos deterministas de tiempo polinomial ya sea para los problemas intratables, o para variaciones de éstos.

La habilidad de algoritmos no deterministas, de poder revisar un número exponencial de posibilidades en tiempo polinomial, genera la sospecha, de que este tipo

de algoritmos son estrictamente más poderosos que los algoritmos deterministas de complejidad polinomial. Sin embargo, contra todos los esfuerzos realizados, no se ha podido demostrar esta especulación. La pregunta de si ¿*P* = *NP*?, es uno de los problemas fundamentales de estudio en la ciencia de la computación [4].

En lo que respecta al problema del coloreo de un grafo, es conocido que cuando un grafo es 2-coloreable, tal propiedad puede ser reconocida en un tiempo polinomial, esto significa que determinar si un grafo es o no 2-coloreable, es un problema de la clase de complejidad P. De hecho, una forma de revisar que un grafo *G* es 2-coloreable es revisando que *G* sea bipartito, y esto puede hacerse al verificar que todo ciclo que aparece en *G* sea de longitud par.

Al incrementar el número de colores de 2 a 3, el problema del 3-coloreo de un grafo es conocido como un problema que se encuentra en la clase de complejidad NP. Y en general, el problema del *k-coloreo* con $k \geq 3$ es un problema de la clase NP. Esto significa, que aún y con todo el esfuerzo que se ha dedicado al diseño de algoritmos que resuelven el problema del coloreo de grafos, estas propuestas algorítmicas no son eficientes, considerando como entrada grafos generales donde se requieran más de dos colores para ser coloreados.

En el capítulo siguiente presentaremos algunos de los métodos y condiciones utilizadas en los algoritmos que permiten la 3-coloración de un grafo de manera eficiente. Así como también presentaremos algunas topologías simples de grafos que determinan que 3 colores no son suficientes para el coloreo del grafo.

III) Procedimientos básicos para el coloreo

Se mostraran topologías de grafos que son 2 y 3 coloreables, además de mostrar ejemplos de grafos que no se pueden colorear con sólo 3 colores, concluido esta sección se mostrará nuestra propuesta algorítmica.

3.1 Procedimiento para el 2 Coloreo

La ventaja de la existencia de topologías de grafos 2-coloreables, es que pueden ser reconocidas en un tiempo polinomial en base al tamaño del grafo. Consideremos un grafo de entrada G y sea T_G su grafo formado del recorrido a lo profundo sobre G.

1. Si T_G no tiene ciclos básicos impares entonces G es 2-coloreable. Esta opción considera el caso de que T_G es un grafo bipartito. Dado que después de realizar la búsqueda a profundidad se puede alternar dos colores con respecto a los niveles resultantes de T_G, se construye un árbol 2-coloreable propio, esto se puede verificar solo si ningún color de v es igual a su adyacente por medio de su arista de retroceso, se muestra en la Fig.18a un grafo sin ningún ciclo impar, por lo tanto es 2-coloreable, en la Fig. 18b se muestra un grafo que tiene ciclos impares y que contiene aristas de retroceso que unen vértices con el mismo color, por lo tanto no es 2-coloreable.

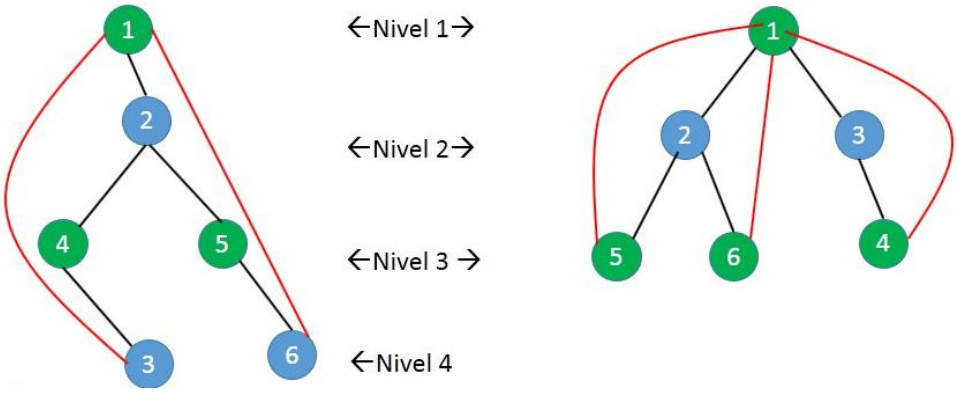


Fig. 18. a) Grafo 2-coloreable **b) Grafo que no es 2-coloreable**

3.2Condiciones Necesarias para el 3 Coloreo

Existen topologías de grafos que han sido identificadas por ser 2-coloreables o 3-coloreables

1. Si T_G tiene ciclos impares, pero todos ellos son independientes de otros ciclos impares entonces T_G es 3-coloreable. Podemos colorear T_G por niveles, pero usando el tercer color para pintar un nodo final de cada ciclo impar, (Fig. 19) muestra un grafo con dos ciclos impares que no comparten ninguna arista (ciclos independientes), se puede colorear con 3 colores como se muestra, asignando el tercer color al nodo final del ciclo.

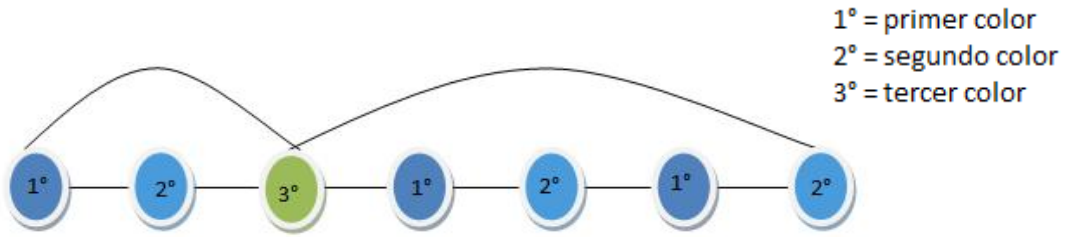


Fig. 19. Grafo con dos ciclos impares

2. Si un ciclo impar en *G* se cruza solo con ciclos pares o a lo mucho hay dos ciclos intersectados entonces T_G es 3-coloreable. Como el caso anterior, T_G es dos coloreado por niveles, y el tercer color se utiliza para colorear el nodo final que es común a cada par de ciclos intersectados impares, en (Fig.20) podemos ver un ejemplo en que se tiene un ciclo impar intersectado con un ciclo par de tal forma que se colorea con un tercer color el nodo final común y el resto es dos coloreado.

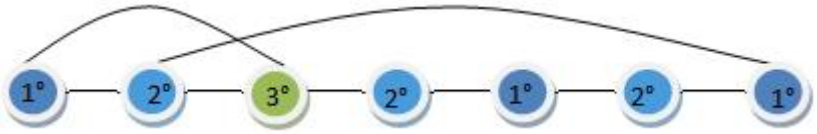


Fig. 20. Ciclo impar intersectado con un ciclo par

3. Si todo el conjunto de los ciclos impares intersectados de *G* puede ser dispuesto como ciclos planos embebidos uno dentro de otro, entonces *G* es 3-coloreable. Coloreamos cualquier conjunto de ciclos planos embebidos desde el más interno al ciclo externo (Fig. 21). Cuando empezamos a colorear un nuevo ciclo, se utiliza un color diferente a los nodos vecinos (un máximo de dos nodos vecinos), ya que consideramos que sólo los nodos en los ciclos internos ya han sido coloreados. En este caso, podemos considerar a T_G como un ejemplo de un grafo serial paralelo, el cual es conocido por ser coloreado en tiempo polinomial [4].

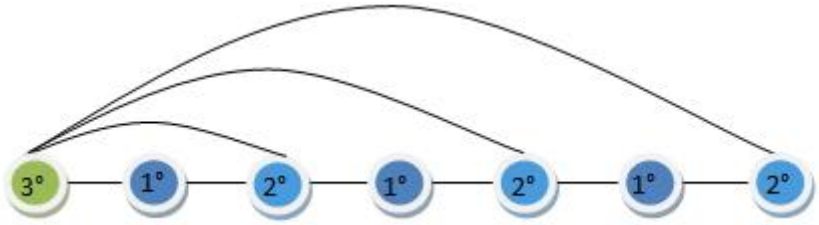


Fig. 21. Ejemplo de ciclos impares embebidos

3.3 Grafos que no son 3-coloreables

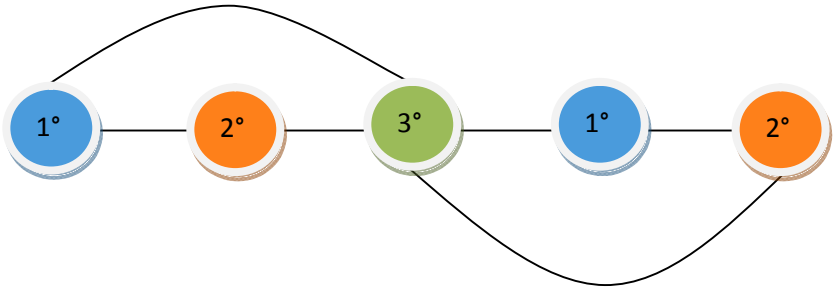
En el apartado anterior se presentaron instancias de grafos a partir de los cuales se puede considerar que un grafo es 3-coloreable.

En la sección anterior se abordó el siguiente caso: Si un ciclo impar en *G* se cruza solo con ciclos pares o a lo mucho con dos ciclos intersectados, entonces T_G es 3-coloreable.

Partimos de este punto para plantear la topología de los grafos que no son 3-coloreables.

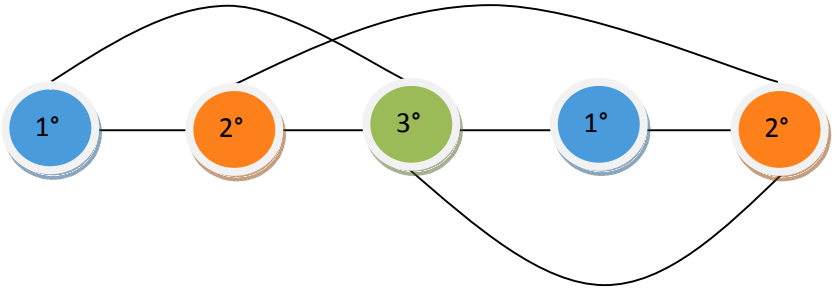
Si un ciclo par en *G* se cruza con dos ciclos impares intersectados de forma que compartan un nodo adyacente a sus extremos entonces el grafo no es 3-coloreable.

Ejemplo:



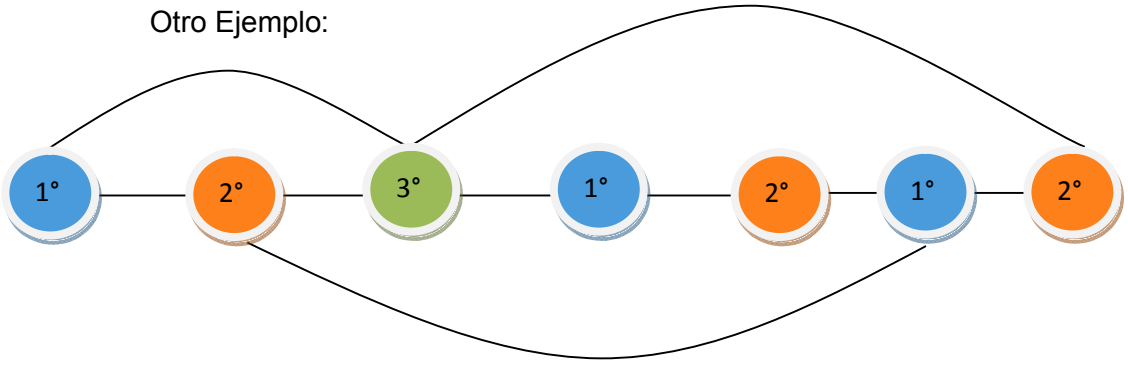
1° = primer color
2° = segundo color
3° = tercer color

El grafo anterior es 3-coloreable si cerramos un ciclo par cuyos extremos toquen el extremo de los impares, entonces el grafo deja de ser 3-coloreable, de la siguiente forma:

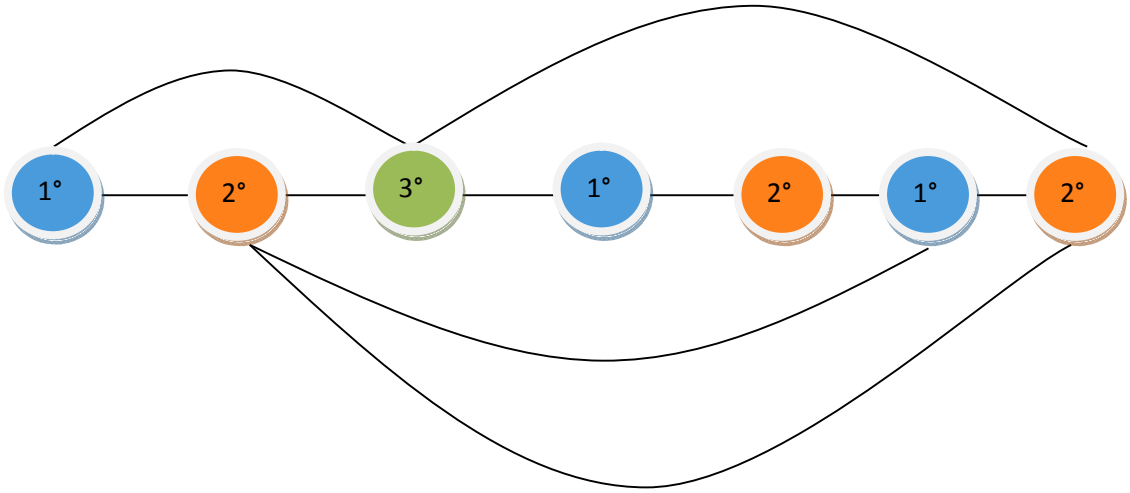


El nuevo ciclo añadido requiere que cualquiera de los dos nodos con el segundo color deban cambiar de color para evitar la repetición de color, sin embargo, ambos nodos se encuentran tocando tanto nodos del primer color como del tercer color. Por lo tanto el grafo no será 3-coloreable.

Otro Ejemplo:



Este grafo es 3-coloreable, al cerrar el ciclo par intersectado deja de serlo, de la siguiente forma:



Los nodos con el segundo color se hacen adyacentes y no pueden cambiar a alguno de los otros dos colores, ya que tocan al primer y tercer color, viéndonos obligados a usar un color extra para colorear el grafo de forma propia.

3.4 Algoritmos básicos de coloración

3.4.1 Coloreo Secuencial o voraz

Este algoritmo sigue una estrategia voraz, es decir comienza la coloración de los vértices según el orden de estos en la matriz de incidencia o de su lista de adyacencias del grafo. La coloración se realiza siguiendo los siguientes pasos [11].

1) Asignar un color al primer vértice de la entrada del algoritmo.

2) A continuación escogemos el siguiente vértice en la ordenación y le asignamos el menor color posible, es decir utilizar uno de los primeros colores, partiendo desde el primer color utilizado y que sea diferente respecto a sus adyacentes. Repetimos este proceso hasta que todos los vértices del grafo hayan sido coloreados. En cada paso se le asigna un color a un vértice y no se le vuelve a modificar a lo largo de la ejecución de ahí el carácter voraz del algoritmo [9].

Algoritmo:

1) Recibe un grafo,

2) Inicia Algoritmo Secuencial,

(3) Se tienen ordenados los vértices de 1 a n, y se selecciona el primero en la lista y se colorea,

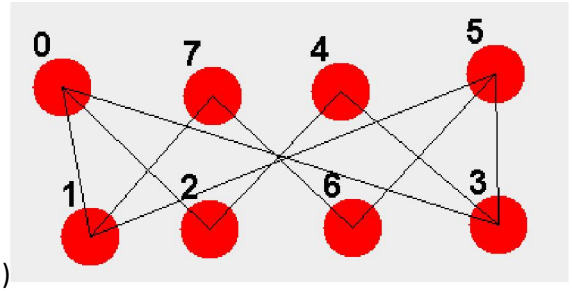
(4) Se toma el siguiente vértice y se colorea con en el menor número admisible; es decir, se verifican adyacencias,

(5) Se repite el paso (4) hasta que todos los vértices hayan sido coloreados.

(6) Se muestran los resultados.

La aplicación del algoritmo secuencial colorea el grafo de la Fig. 22 a) obteniendo como resultado la Fig.22 b) y por consecuencia los siguientes colores:

Amarillo = {0,4,5,7}, Azul = {1,2,3,6}.

a) 

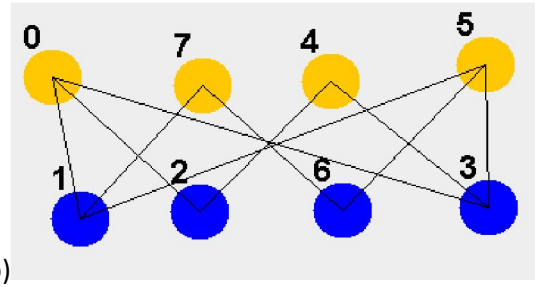
b) 

Fig. 22. Coloración con algoritmo secuencial

3.4.2 Algoritmo de coloración Welsh-Powell

La única diferencia respecto al algoritmo secuencial es el orden en el que se realiza la coloración de vértices. En este caso los vértices se ordenan de mayor a menor grado, es decir, en función del número de vértices adyacentes [10].

Primero se ordenan los vértices de mayor a menor grado. Después se colorea uno a uno con el menor número (color) cuidando que no tengan el mismo color los vértices adyacentes [10].

Algoritmo:

1) Abre un archivo, recibe el número de vértices, aristas y crea la matriz o lista de adyacencias,

2) Inicia el Algoritmo Welsh-Powell,

(3) Se ordenan los vértices de mayor a menor grado, se selecciona el primero en la lista y se colorea o se etiqueta con el 1,

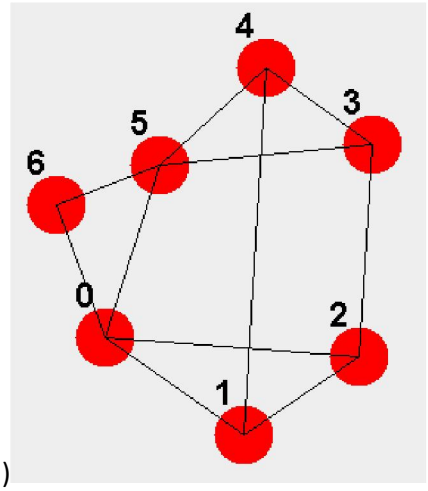
(4) Se toma el siguiente vértice y se colorea o etiqueta con en el menor número admisible; es decir, se verifican adyacencias,

(5) Se repite el paso (4) hasta que todos los vértices hayan sido coloreados.

(6) Se muestran los resultados.

El algoritmo podría colorear el grafo de la Fig. 23a obteniendo como resultado la Fig.23b) y por consecuencia los siguientes colores:

Amarillo = {0,3}, Azul = {1,5}, Cyan = {2,4,6}.

a) 

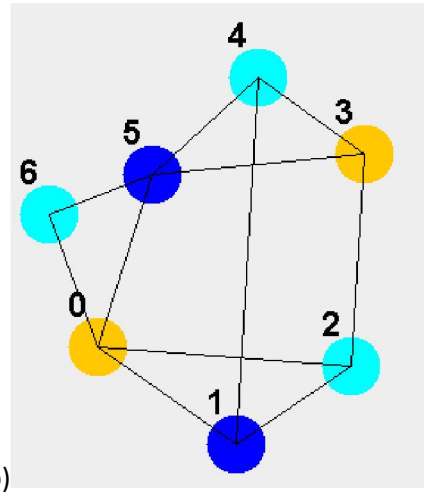
b) 

Fig. 23. Coloración con el algoritmo Welsh-Powell

IV) Análisis e Implementación de la Propuesta Algorítmica

Visto lo anterior tenemos lo necesario para poder analizar nuestra nueva propuesta algorítmica que consiste en construir clases óptimas para el coloreo un grafo, este algoritmo se compone de dos partes principales que es la identificación de un conjunto I_B y la búsqueda y obtención de un MIS K, lo que se describe a continuación.

4.1Definición y Obtención de Conjunto I_B

Dado un grafo conexo $G=(V,E)$, sea $n=|V|$, $m=|E|$ el número de nodos y aristas, respectivamente. Se realiza el recorrido en profundidad sobre G. Se conforma un conjunto I_B de nodos de G consistiendo de nodos que no forman parte de ningún ciclo impar para el grafo G. El cómputo de I_B puede realizarse en tiempo polinomial sobre el tamaño $(n+m)$ de G.

El conjunto I_B conforma un subgrafo bipartito de G mostrado por el siguiente lema.

Lema 3. Sea G un grafo conexo y sea $G|I_B \subseteq G$ el subgrafo inducido de G formado por nodos que no son parte de ciclos impares de G, por lo tanto $G|I_B$ es bipartito [14].

La prueba se basa en la conexión de los nodos en I_B, por ejemplo, el primer caso considera los nodos de I_B que forman un grafo conexo y el segundo considera el caso donde I_B se compone de grafos desconectados. En el primer caso, es bien conocido que un grafo conexo que no tiene ciclos impares es bipartito. Para el segundo caso, se muestra el siguiente lema.

Lema 4. Sea$G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$ son grafos bipartitos.

Si $E_1 \cap E_2 = 0$ y $|V_1 \cap V_2| < 2$ entonces $G = G_1 \cup G_2$ es bipartito.

Comprobación: sea C_i y C_j ciclos de G_1 y G_2 respectivamente , $C_i \Delta C_j = (E(C_i) \cup E(C_j)) - (E(C_i) \cap E(C_j)) = (E(C_i) \cup E(C_j))$, como $E_1 \cap E_2 = 0$ y entonces, ya no hay más ciclos en G que las existentes en G_1 y G_2. Como no hay más ciclos impares en G_1 y G_2 porque ambos subgrafos son bipartitos entonces no hay ciclos impares en G y G es bipartito [14].

I_B podría ser llamado el componente bipartito inicial de G porque estos nodos pueden ser coloreados al final del proceso de coloreo usando sólo los dos primeros colores básicos.

4.2Definición y Lemas para la Obtención del MIS *K*

Un conjunto independiente maximal de un grafo será denotado como un MIS del grafo. Un MIS representa un color de clase para el grafo, esta es una de las bases que utiliza nuestro algoritmo para poder obtener el número cromático del grafo.

Definición 1. Un vértice *v* en un grafo conectado *G* es llamado corte vértice o punto de articulación si $G \setminus v$ es un grafo desconectado.

Lema 5. Sea *G* un grafo conectado de tal manera que $|E(G)| \geq |V(G)| + 1$. Aquí no existe un conjunto de articulación $S \subset V(G)$ de tal manera que $|E(G \setminus S)| < |V(G \setminus S)|$.

Comprobación: Si $|E(G)| > |V(G)| + 1$ significa que hay ciclos en el grafo.

Construir un árbol extendido del grafo. Para cada ciclo básico *C*i en *G* donde e_i es la arista de retroceso que forma el ciclo, eliminar un nodo excepto los de e_i. Después de romper cada ciclo de *G* el grafo resultante tiene menos aristas que nodos por lo que es acíclico. El conjunto *S* es construido con los nodos eliminados en el grafo original.

Lema 6. Sea *G* un grafo conectado de tal manera que $|E(G)| \leq |V(G)|$, hay un MIS*K* de *G*, de tal manera que $\delta G(K) \geq |V(G)| - 1$.

Comprobación: si $|E(G)| < |V(G)|$, entonces *G* es un grafo acíclico que tiene una estructura de árbol. El nodo principal está en el nivel 0 del árbol mientras que sus hijos están en el nivel 1, sus nietos en el nivel 2, y así sucesivamente. *K* es formado por los nodos en niveles pares del árbol, incluyendo el nodo principal. Si $|E(G)| = |V(G)|$,entonces *G* tiene un ciclo simple. Se inicia *K* añadiendo al nodo v con el máximo grado en el ciclo simple, por lo tanto elimina $N[v]$. En este punto $|E(G)| < |V(G)|$, por lo tanto el método acíclico puede ser usado. Nótese que las aristas de *G* son cubiertas por al menos un nodo de *K*, entonces $\delta G(K) = \sum_{x \in K} \delta G(K) \geq |E(G)|$, que significa que $\delta G(K) \geq |V(G)|-1$

Lema 7. Sea *G* y *H* dos grafos conectados no intersectados, *K* un MIS de *G* de tal manera que $\delta_G(K) \geq |V(G)| - 1$. Supongamos que hay un vértice $x \in V(H)$ tal que $N[x] = V(H)$. Sea *S* un conjunto arbitrario en $V(G) \times V(H)$ tal que *S* conecta al menos un vértice $y \neq x \in N[x]$ a un vértice *w* en $N(K)$, por lo tanto se cumple para la unión $G \cup H$, que:

1. $K \cup \{x\}$ es un MIS de $G \cup H$, tal que $N_{G \cup H}(K) = N_G(K) \cup \{y\}$, *y* es un vértice tal que $N_{G \cup H}(y) = N_H(y) \cup \{w\}$ para algunos $w \in N(K)$.

2. $\delta_{G \cup H}(K \cup \{x\}) \geq |V(G \cup H)| - 1$.

Demostración

1 se cumple, dado que cada vértice de *H* está en la vecindad de *x*, así solo el vértice que puede contribuir a un MIS de $G \cup H$ es *x* mismo.

Claramente se cumple 2, ya que $y \neq x$ conecta a algún $w \in N(K)$, al realizar la operación de unión $G \cup H$, $N_{G \cup H}(K) = N_G(K) \cup \{y\}$ implica que $|N_{G \cup H}(K)| = |N_G(K)| + 1$, por tal

$\delta_{G\cup H}(K)= \delta_G(k)+1$, y como $G\cap H=0$ y $N[x]=H$ nosotros tenemos que $|V(G\cup H)|= |V(G)+V(H)|$ y $\delta_{G\cup H}(x)= \delta_H(X)=|V(H)|-1$, respectivamente.

$$\begin{aligned}\delta_{G\cup H}(K\cup\{x\}) &= \delta_{G\cup H}(K)+ \delta_{G\cup H}(x)\\ &= \delta_G(K)+1+|V(H)|-1\\ &\geq |V(G)|-1+1+|V(H)|-1\\ &= |V(G\cup H)|-1\end{aligned}$$

4.2.1Construcción del MIS *K*

Sea G un grafo conectado tal que $|E(G)| \geq |V(G)|+1$, el algoritmo que construye el MIS K consiste de los siguientes pasos:

1. Mientras $|E(G)| \geq |V(G)|$ elegir un nodo que no sea un punto de articulación $e \in G_i$, meter x a la pila y remover x del grafo G_i ,$G_{i+1}= G_i- \{x\}$. almacenar las aristas de x en el subgrafo G_i. Para acelerar el proceso se elige un nodo con el máximo grado en G_i y que no sea un punto de articulación. Al final se obtendrá un grafo G_j tal que $|E(G_j)| \leq |V(G_j)|$.
2. Construir un MIS de G_j (lema 6): un MIS inicial K_0 de G_j es construido tal que $\delta_{Gi}(K_0)\geq |V(G_j)|-1$.
3. Recuperar los nodos de la pila (lemma7): pop x de la pila, y recobrar sus respectivas aristas del subgrafo G_{j-1}. Si $N_{Gj-1}(x) \cap K_0=0$ entonces $K_0= K_0 \cup\{x\}$, repetir paso 3 hasta que la pila este vacía.
4. Regresar K_0.

En la (Fig. 24.) se muestra el algoritmo para construir el conjunto MIS*K*.

Entrada: Un grafo *G*, de tal manera que $|E(G)| \geq |V(G)| + 1$

Salida: Un MIS *K* de tal manera que $\delta G(K) \geq |V(G)| - 1$

Procedimiento:

1. *G1 = G*; reconocer puntos de articulación (*G*); /*identificación de puntos de articulación de *G* */

mientras ($|E(G_i)| > |V(G_i)|$){

Selecciona $x \in V(G_i)$, $\delta G_i(x)$ es máximal, tambien x no es un punto de articulación en G_i;

meter(x, pila);

meter(EGi(x), subgrafo);

$G_{i+1} = G_i - \{x\}$;

Reconocer puntos de articulación (G_i);

}

2. construye el inicial MIS *K* de *G*i de tal manera que $\delta G_i(K) \geq |V(G_i)| - 1$

3.*mientras(Stack contenga x)* {

sacar(x, Stack);

sacar(E(x), Subgrafo);

$G_{j-1} = G_j \cup E(x)$;

Si (x no es adyacente a algún nodo de *K)*

$K = K \cup \{x\}$;

}

regresa(K)

Fig. 24. Procedimiento de construcción de un MIS(G)

4.3 Búsqueda del Número Cromático

*E*n las secciones anteriores de este capítulo se describió como formar el conjunto I_B y un conjunto independiente maximal MIS *K,* en esta parte concluimos nuestra propuesta algorítmica al presentar el proceso general de colorear el grafo de entrada *G*.

A continuación se describe una estrategia general de nuestro método de búsqueda del número cromático (Seek_Chromatic_Number(*G*)).

Dado una entrada de un grafo $G=(V,E)$, con $n=|V|$, $m=|E|$, procedemos a aplicar una búsqueda en profundidad a *G*, un conjunto I_B que consiste de nodos que no forman parte de ningún ciclo impar de *G* se identifica y es apartado de *G*.

Se puede definir la estrategia general de la siguiente forma:

Primero: en cada iteración para la búsqueda de un MIS *K* se prueba si el grafo actual es dos coloreable.

Segundo: se forma un MIS *K* tal que $\delta G_i(K) \geq |V(G_i)| - 1$ y

Tercero: se colorean los nodos en K_i, es decir, obtenemos el color actual de K_i para colorear el MIS, los nodos y aristas de K_i son eliminados del grafo actual y el proceso se repite, al finalizar se hace la unión del conjunto I_B con el grafo actual G_i.En la (Fig. 25) se muestra el algoritmo Seek_Chromatic_Number(*G*) o búsqueda del número cromático(G).

Entrada: *G* un grafo no dirigido

Salida: un aproximado valor para $\chi(G)$

Procedimiento:

k = 0; /* inicia con la clase color *k* = 0 */

G= *dfs*(*G*); *IB* = {*u* ∈*V*(*G*) : *u no es parte de algún ciclo impar de G*};

G= *G* − *IB*; /*Considere sólo los nodos que forman ciclos impares o intersección */

si(*G* bipartito)

regresa("χ(G) is 2");

mientras(es bipartito(G)==falso){ /* mientras hay ciclos impares en *G* */

K= construir MIS(G);

G = *G* − *K*; *k* + +; /* actualizar para el siguiente MIS */

}

G = *G* ∪ *IB*; /* regresa el primer componente bipartito */

llama 2-colouring(*G*); /* en el final, el grafico restante es 2-colorable*/

regresa ("χ(G) is k + 2");

Fig. 25. Algoritmo búsqueda del númeroCromático(G)

4.4 Implementación del sistema

Se implementó el algoritmo anterior en un sistema de cómputo con interfaces orientadas al usuario, el cual permite introducir un grafo *G* de manera gráfica, el algoritmo procesa y da como resultado un coloreo de dicho grafo *G* introducido en el sistema.

4.4.1 Descripción General del Sistema

El sistema funciona obteniendo un grafo dentro de su lienzo de dibujo, este dibujo es obtenido luego de que el usuario dibuja un grafo de su interés sobre el lienzo, al obtener dicho grafo este pasa a ser procesado de tal manera que es convertido internamente en una estructura de datos, el usuario selecciona la opción del método de numero cromático, dicha opción invocará el método interno del programa Seek_Chromatic_Number(*G*) "BusquedaNumeroCromático" para que procese la estructura de datos, el grafo sea coloreado y se muestre en pantalla el resultado del coloreo (Fig. 26).

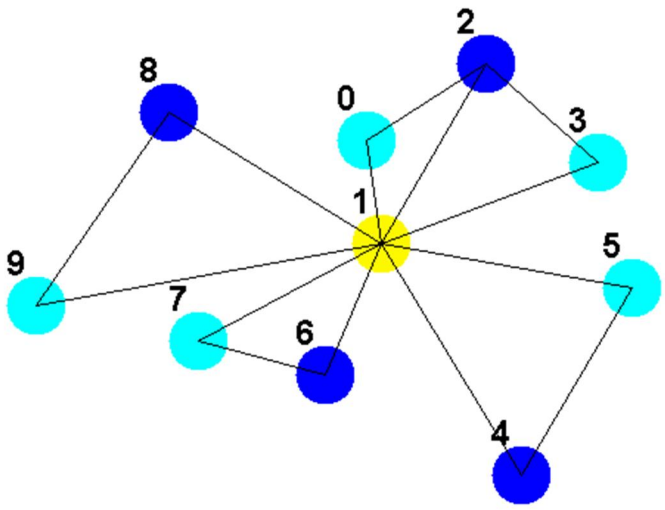


Fig. 26. Grafo procesado y coloreado

Es importante mencionar que dentro del lienzo de dibujo los nodos serán marcados con una etiqueta solo después de hacer una arista de adyacencia hacia otro nodo.

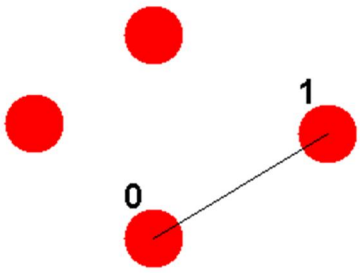


Fig. 27. Nodos adyacentes marcados con una etiqueta

4.4.2 Diagrama General del Sistema

El sistema esta propuesto por 3 principales módulos que funcionan en coordinación y de forma continua (Fig.28).

1) Dibujo y detección del Grafo

2) Construcción de conjunto I_B y K

3) Obtención de clases MIS K y Coloreo del Grafo

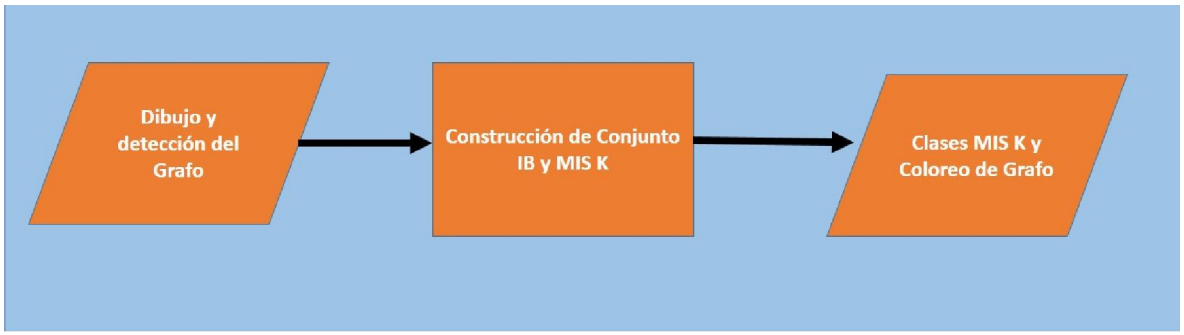


Fig. 28. Diagrama general de sistema

El módulo de Dibujo y detección del grafo se compone de dos partes principales, la primera es dibujar el grafo en el lienzo y dado este dibujo, obtener una estructura de datos, la cual será mandada al módulo 2.

El segundo módulo es el más interesante pues en él están tres cuartas partes del desarrollo, ya que en este módulo del sistema se realizan todos los algoritmos necesarios para poder obtener el coloreo, para profundizar más de este módulo, se utiliza la estructura de datos planteada por el anterior módulo, esta estructura de datos es un arreglo de listas ligadas es decir que el grafo está representado por listas de adyacencia [12], podemos ver la representación en la (Fig.29).

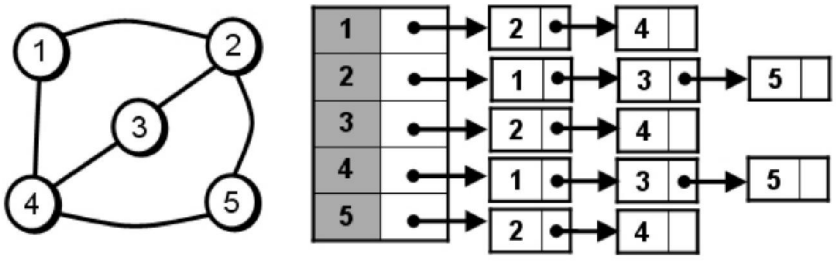


Fig. 29. Grafo y su representación con listas de adyacencia

Obtenido el grafo, el método realiza una búsqueda en profundidad, dicha búsqueda en profundidad es codificada de tal forma que también se obtengan las aristas de retroceso, estas aristas de retroceso nos servirá más adelante para verificar la existencia de ciclos en el grafo.

Ahora pasamos a obtener el Conjunto I_B, este conjunto se obtiene después de realizar la búsqueda en profundidad. Se obtendrán nodos que sean hojas o que puedan ser dos coloreables y se eliminan del grafo. En la (Fig. 30) se muestra un ejemplo del conjunto I_B a quitar de un grafo, este conjunto identificado es dos coloreable, I_B = {4,8,5,7,6}.

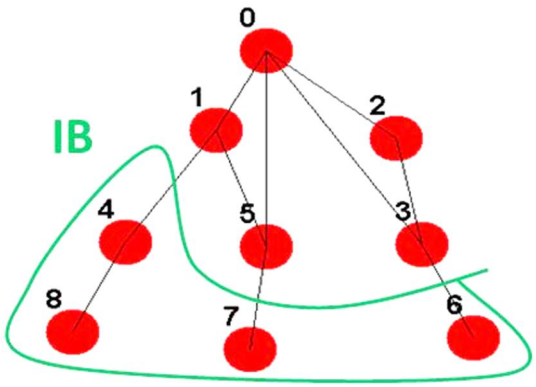


Fig. 30. Conjunto I_B.

Después de obtener el conjunto I_B pasamos al algoritmo de construcción de un MIS *K,* el cual para funcionar correctamente necesita identificar puntos de articulación, nodos con grado maximal y por último, identificar correctamente el MIS *K.* En la siguiente sección de ejemplos veremos más a fondo como se obtienen cada uno de ellos en un ejemplo completo.

El tercer módulo prácticamente presenta todo el resultado del desarrollo que se realiza en el módulo 2, en el módulo 3 podemos obtener las clases para los diferentes MIS *K,* así como también podemos ver el coloreo del grafo dibujado en el lienzo.

V) Pruebas y Resultados

En esta sección nos enfocaremos en desarrollar un solo ejemplo para mostrar todos los pasos internos que se desarrollan dentro del algoritmo para lograr el coloreo adecuado, lo que se desglosa en:

Obtener el conjunto I_B.

Obtener un MIS K.

Unión de conjunto I_B y MIS K.

A continuación se muestra un grafo a colorear y su respectiva representación en listas de adyacencia, con dicha lista se trabajará internamente el algoritmo Seek_Chromatic_Number(G), los vértices son representados en color rojo y sus adyacentes en color naranja[13].

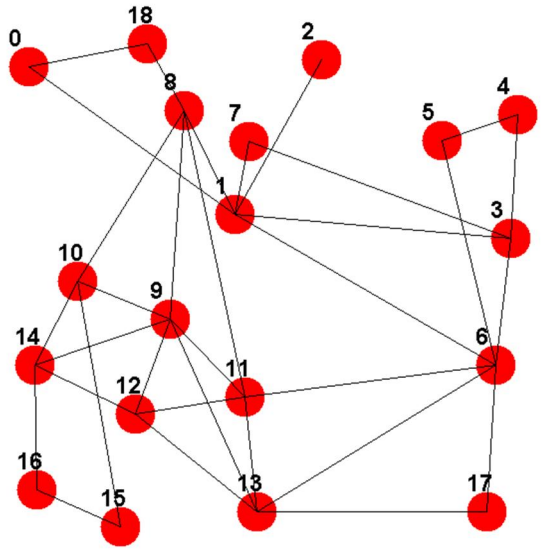


Fig. 31. Grafo completo sin coloreo

0	1	18				
1	0	2	3	7	8	6
2	1					
3	1	4	7	6		
4	3	5				
5	4	6				
6	5	13	17	1	3	11
7	3	1				
8	1	9	10	11	18	
9	8	10	11	12	14	13
10	9	8	14	15		
11	9	8	12	13	6	
12	11	9	13	14		
13	12	11	6	17	9	
14	10	12	9	16		
15	10	16				
16	15	14				
17	13	6				
18	8	0				

Fig. 32. Matriz de adyacencia

Paso 1) Identificar y quitar el conjunto I_B del grafo G.

Procedemos con el primer paso, para este ejemplo de grafo el conjunto I_B está compuesto solamente del nodo {2}, entonces quitamos del grafo G dicho conjunto. Quedando de la siguiente forma $G_1=G-I_B$, (Fig.33).

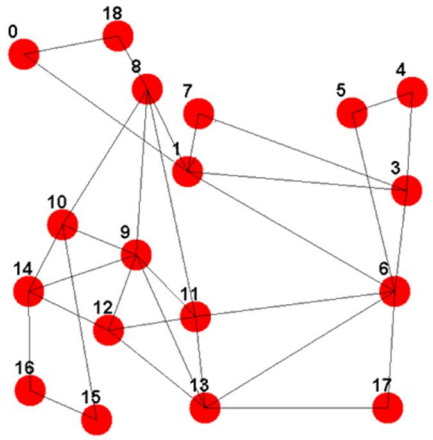


Fig. 33. Grafo quitando el conjunto I_B, $G_1=G-I_B$

Paso 2) Búsqueda y obtención de MIS K.

Ahora se tiene grafo G sin el conjunto I_B y se verifica si es 2 coloreo, como no es 2 coloreo pasamos a buscar un MIS K que cumpla la condición $\delta G_i(K) \geq |V(G_i)| - 1$, para esto se obtiene del grafo G todos los nodos con mayor grado, los cuales se eliminan del grafo y se agregan a una pila la cual denotaremos como “pilaMaximal”, estos nodos serán maximales y se itera hasta que la condición $E(G)>V(G)$ se cumpla, se buscaran nodos con grado que no sean puntos de articulación para así eliminarlos del grafo y meterlos a una pila.

Obtenemos el número de aristas y numero de vértices respectivamente obteniendo que: $E(G)=32$, $V(G)=18$.

Condición para obtener los nodo maximales hasta que la condición $E(G)>V(G)$ se cumpla, en cuyo caso pasamos a encontrar los puntos de articulación y obtener nodos maximales quedando de la siguiente forma:

Articulación = { }, en esta fase no hay puntos de articulación.

pilaMaximal = {6}.

Se procede a eliminar el nodo maximal 6 del grafo quedando como se ilustra en la (Fig.34), tener en cuenta que para seleccionar y eliminar cierto nodo maximal este no debe de ser un nodo de articulación, para el caso en el que varios nodos tengan el mismo grado se toma el nodo con la menor etiqueta.

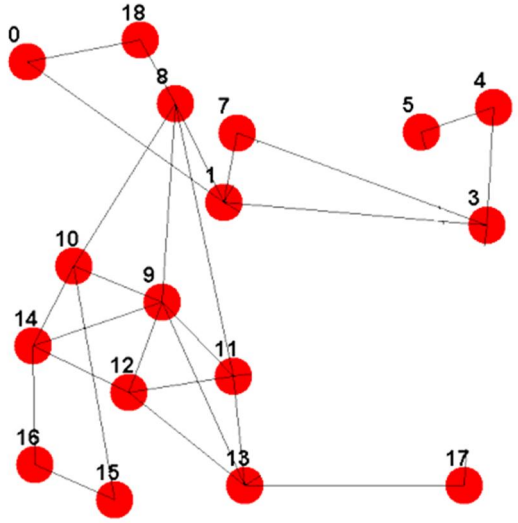


Fig. 34. Eliminación de nodo maximal 6

Quitado el anterior nodo maximal se obtiene el número de aristas y vértices respectivamente, obteniendo los resultados $E(G)=26$, $V(G)=17$.

Condición para obtener nodo maximal: $E(G)>V(G)$, como se cumple la condición pasamos a encontrar los puntos de articulación y obtener un nodo maximal quedando de la siguiente forma:

Articulación = {4,3,1,13,8}. En esta parte ya se aprecian nodos de articulación.

pilaMaximal = {6,9}, ahora eliminaremos el maximal 9 (Fig.35).

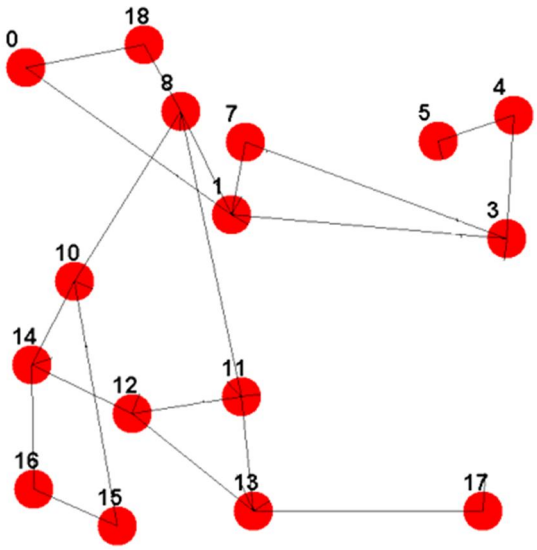


Fig. 35. Eliminación de nodo maximal 9

Dada la eliminación del anterior nodo maximal obtenemos el número de aristas y vértices respectivamente: $E(G)=20$ y $V(G)=16$ como la condición para obtener maximal se sigue cumpliendo obtenemos los siguientes resultados.

Articulación = {4,3,1,13,8}, se mantienen los mismos nodos de articulación.

pilaMaximal = {6,9,10}, ahora se procede a eliminar el numero 10(Fig.36).

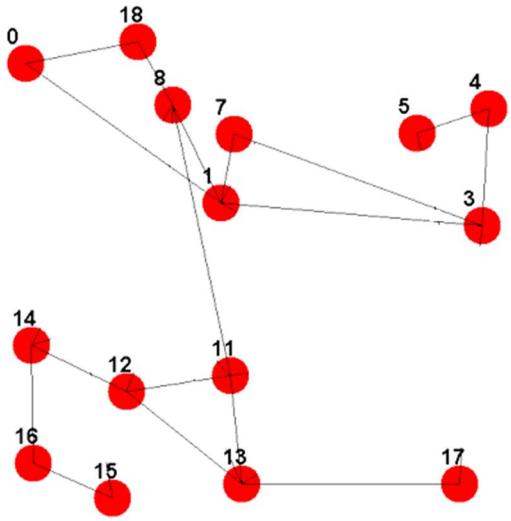


Fig. 36. Eliminación de nodo maximal 10

Verificando la condición para obtener maximal $E(G)>V(G)$ y obteniendo $E(G)=17$ y $V(G)=15$ procedemos a obtener los siguientes resultados.

Articulación = {4,3,1,13,8,16,14,12,11}, se generan nuevos nodos de articulación.

pilaMaximal = {6,9,10,0}, ahora se procede a eliminar el numero 0(Fig.37).

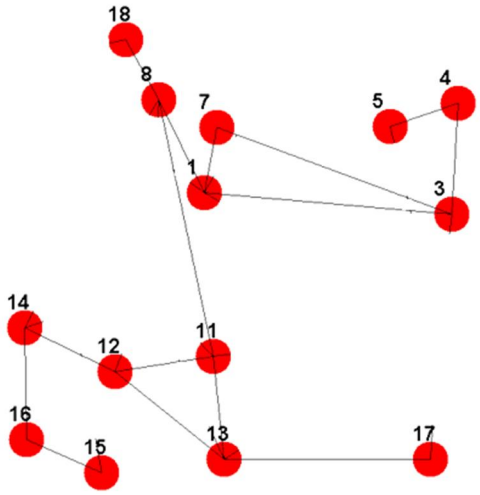


Fig. 37. Eliminación de nodo máximal 0

Teniendo $E(G)=15$ y $V(G)=14$ se verifica nuevamente la condición $E(G)>V(G)$ y se obtienen los siguientes resultados

Articulación = {4,3,1,13,8,16,14,12,11}.

pilaMaximal = {6,9,10,0,7}, ahora se procede a eliminar el número 7, (Fig.36).

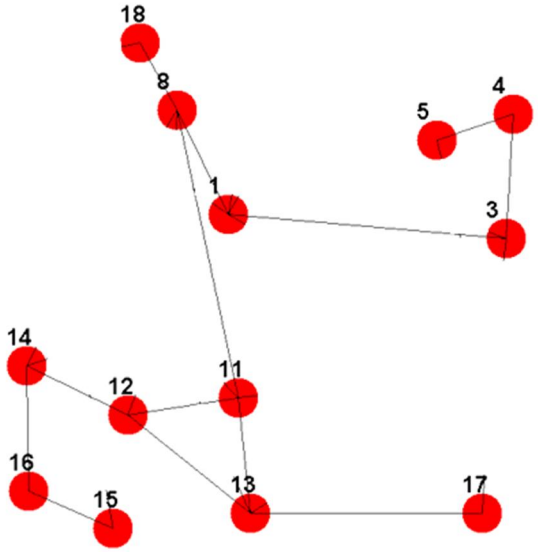


Fig. 38. Eliminación de nodo máximal 7

Teniendo $E(G)=13$ $V(G)=13$, en este punto ya no se cumple la condición $E(G)>V(G)$ y por lo tanto se pasa a formar el conjunto MIS K que debe de cumplir la condición $\delta G_i(K) \geq |V(G_i)| - 1$

Quedando así el primer conjunto MIS K_1={1,4,11,14,15,17,18}, entonces ahora eliminamos el conjunto MIS K_1 del grafo y regresamos los nodos maximales {6,9,10,0,7} que están en la pila, (Fig.39).

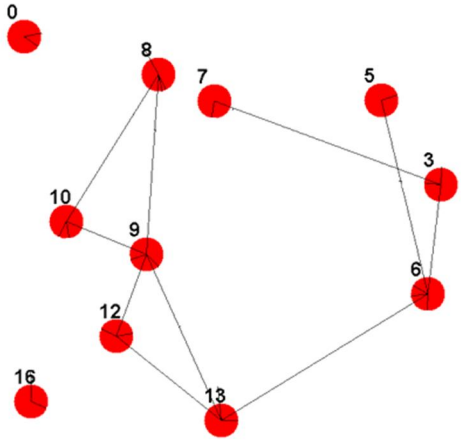


Fig. 39. Se regresan los nodos de la pila

Teniendo la primera clase de color se verifica si el grafo restante de la Fig. 39 es 2-coloreable, como no lo es, entonces empezamos a obtener de nuevo los nodos de grado maximal y que no sean puntos de articulación.

Ahora obtendremos los resultados de numero de aristas y vértices respectivamente, cabe señalar que los vértices {0,16} de la Figura (40),quedan con grado 0 por lo que no son tomados en cuenta para obtener un maximal ya que estos formarán parte del 2-coloreo, únicamente son tomados en cuenta los vértices que tienen al menos un grado, entonces continuando con el proceso tenemos $E(G)=10$ y $V(G)=9$, por tanto la condición para obtener maximal $E(G)>V(G)$ se cumple obteniendo lo siguiente:

Articulación = {6,9,13,3}.

pilaMaximal = {8}, ahora se procede a eliminar el número 8, (Fig.40).

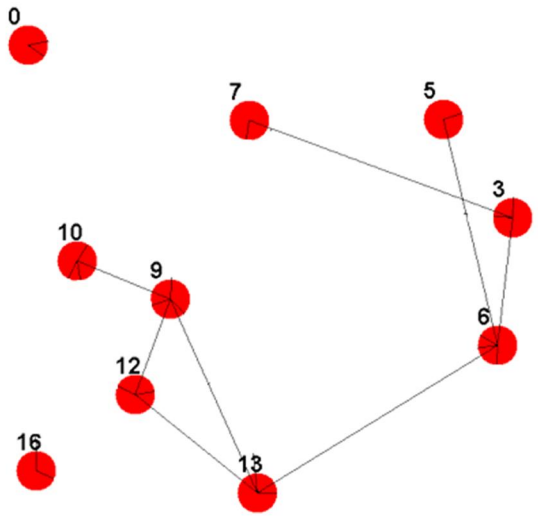


Fig. 40. Eliminación de nodo maximal 8

Verificamos la condición de máximal $E(G)>V(G)$ teniendo número de aristas y vértices respectivamente, $E(G)=8$, $V(G)=8$ por tanto ya no se cumple la condición $E(G)>V(G)$ y se pasa a formar el conjunto MIS K que debe de cumplir la condición $\delta G_i(K) \geq |V(G_i)| - 1$.

Quedando así el segundo conjunto MIS K_2={6,7,9}, entonces ahora eliminamos el conjunto MIS K_2 del grafo y regresamos los nodos maximales {8} que están en la pila de maximales, (Fig. 41).

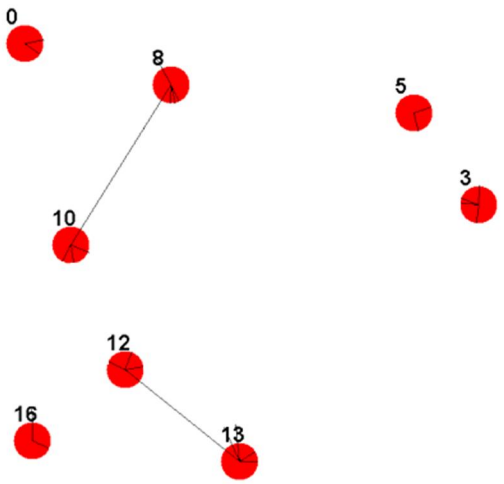


Fig. 41. Regresando los nodos de la pila

Por ultimo obtenemos el grafo de la figura (Fig.42), que ya es dos coloreable, por tanto, se colorea con 2 colores y se regresan los MIS *K* con su respectivo color, además se regresa el conjunto I_B al grafo final resultante.

Resultado final de la ejecucion del algoritmo Seek_Chromatic_Number(*G*)

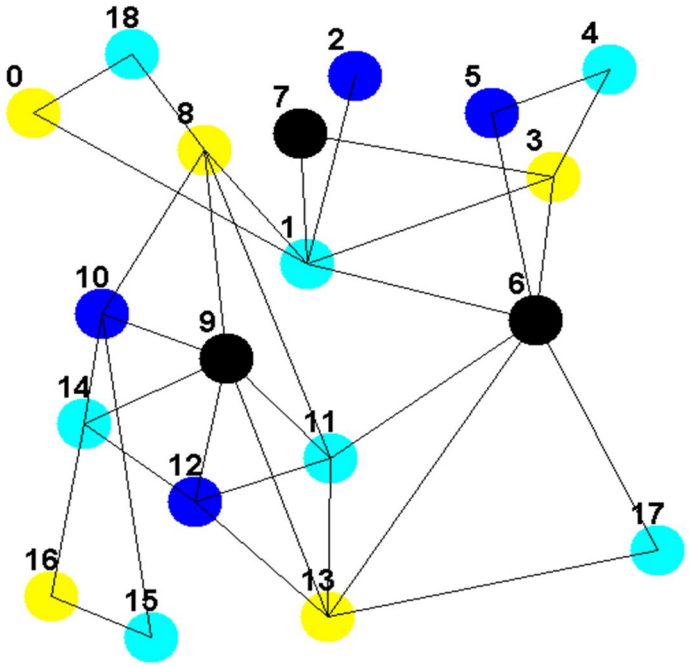


Fig. 42. Grafo Coloreado

Coloreo del grafo (Fig. 42) y MIS K quedando de la siguiente forma

MIS K_1 = {1,4,11,14,15,17,18} (color cyan) MIS K_2 = {6,7,9} (color negro)

MIS K_3 = {16,13,0,8,3} (color naranja) MIS K_4 = {10,2,12,5} (color azul)

Ahora mostramos algunos grafos que son difíciles de colorear y que con nuestra aplicación se están haciendo de manera eficiente, cabe señalar que al probar dichos algoritmos nos dio la cantidad precisa del k-coloreo de G.

Empezando por el grafo de Grotzsch, que se sabe es 4-coloreable.

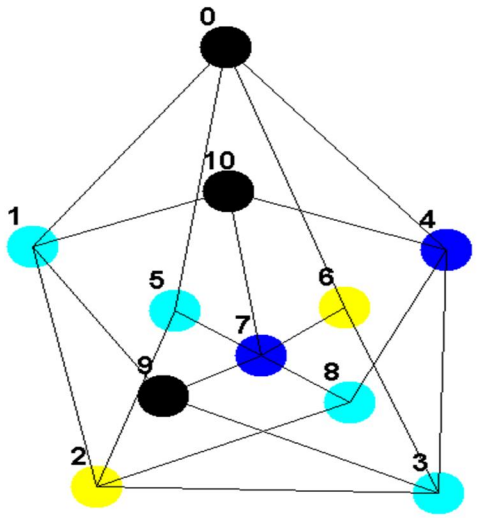


Fig. 43. Grafo de Grotzsch

Ejemplo de 4-coloreo para el grafo de Petersen.

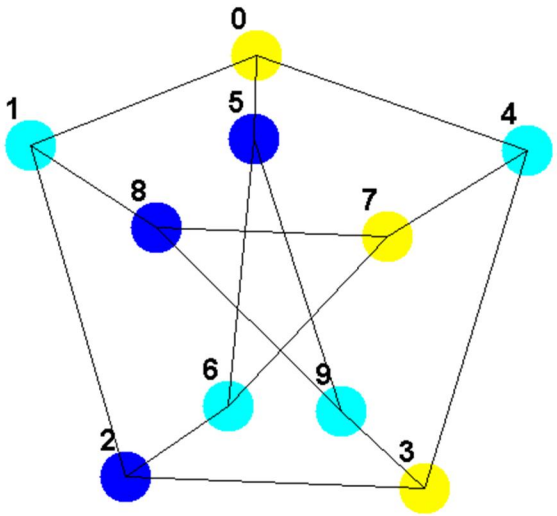


Fig. 44. Grafo de Petersen

Ejemplo de un 4-coloreo para el Grafo myciel3 del benchmark.

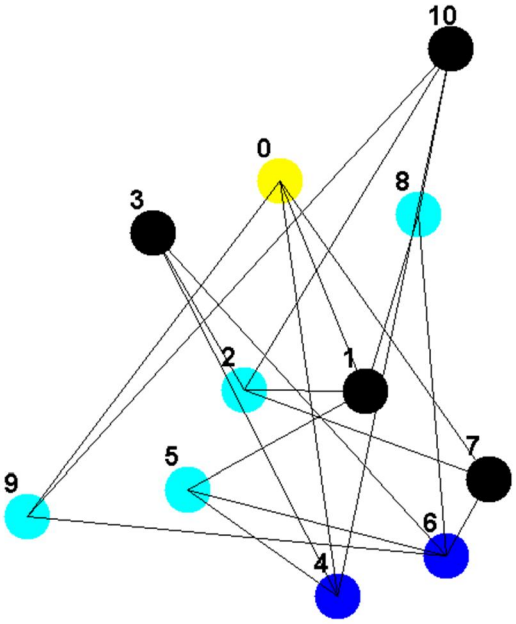


Fig. 45. Grafo de myciel3

VI) Análisis de complejidad

Dado inicialmente un grafo conectado G, sea $G_0=(G-I_B)$ la entrada del grafo sin el componente bipartito (I_B),$G_0=(V,E)$ con $n=|V|$ y $m=|E|$. Podemos asumir que $m=t\cdot n$, $t>1$, y que G_0 tiene ciclos intersectados impares, entonces $m>n$.

Sea T_i el MIS formado en la iteración i del ciclo del procedimiento de Seek_Chromatic_Number(G). Sea $G_{i+1}=(V_{i+1},E_{i+1})$ el subgrafo restante generado de G_i después de finalizar la iteración i, que es,$G_{i+1}=G_i-T_i$.

Sea $n_{i+1}=|V_{i+1}|$, $m_{i+1}=|E_{i+1}|$ y sea $\delta_i=\frac{2mi}{ni}$ el grado promedio de cada subgrafo G_i. En cada iteración, el número de nodos y aristas son actualizados como: $n_{i+1}=n_i-|T_i|$ y $m_{i+1}=m_i-|E_{Gi}(T_i)|$, donde en cada iteración los nodos en Ti (un MIS del actual grafo) son eliminados como también las aristas: $E_{Gi}(T_i)$.

En cada iteración el procedimiento “Construir_MIS” construye un MIS Ti del actual grafo Gi tal que ($\sum_{x\in Ti}\delta Gi(x)\geq(n_i-1)$), bajo los supuestos de que G_i es conectado y $m_i\geq n_i$.

En la primera iteración se mantiene $\sum_{x\in Ti}\delta G(x)\geq(n-1)$, en la segunda iteración $\sum_{x\in T2}\delta_{G-Ti}(x)\geq n_1-1$ que es equivalente a $\sum_{x\in T2}\delta_G(x)-|T_i|\geq(n-|T_1|)-1$ donde cada nodo en T_2 estaba originalmente adyacente a algún nodo en T_1, T_1 es el primer MIS de G y $n_1=n-|T_i|$. Asi$\sum_{x\in T2}\delta_G(x)\geq n-1$

El mismo análisis se mantiene para tres iteraciones a $\sum_{x\in T3}\delta_{G-(Ti\cup T2)}(x)\geq n_2-1$, que es equivalente a $\sum_{x\in T3}\delta_G(x)-(|T_1|+|T_2|)\geq(n-|T_1|-|T_2|)-1$, donde cada nodo en T_2 , $n_2=n-|T_1|-|T_2|$. Asi$\sum_{x\in T3}\delta_G(x)\geq n-1$.

El procedimiento para hasta que el grafo G_k es bipartito, Así en la iteración K, se cumple que: $\sum_{x\in Tk}\delta_G(x)\geq n-1$.

Adición de las partes izquierda y derecha de la desigualdad respectivamente (Fig.46)

$$\sum_{x\in\cup_{i=1}^{k}T_i}\delta_G(x)=\sum_{x\in V}\delta_G(x)=2m\geq k\cdot(n-1)$$

Fig. 46. Desigualdad

Puesto que t_i, i=1,…..,k es una partición de V y la suma del grado de los nodos de un grafo conectado es el doble de su número de aristas.

La ultima desigualdad establece un orden de $k \leq (2m)/(n\text{-}1)$ iteraciones para el procedimiento Seek_Chromatic_Number(G). Entonces, $[(2m)/(n\text{-}1)] + 1$ colores son suficientes (debido a los dos colores utilizados en la última iteración) para colorear al grafo inicial G, lo que nos da un valor de $\delta(1 + [1/(n-1)]) + 1$, o $[\delta \cdot \frac{n}{n-1}] +1$ colores,con $\delta=2m/n$ el grado promedio del grafo inicial G_0.

Así, si cada grafo G_{i+1} generado por nuestra heurística es conectado, entonces $\delta(G_0)$ es el grado promedio de la entrada del grafo G fuera de este primer componente bipartito.

Este último resultado además puede ser mejorado si se adiciona en la condición que G sea conectado en cada iteración, esto también mantiene que $m_i \geq n_i + 1$ y entonces un MIS T_i puede ser construido tal que $\sum x \in T_i \delta G_i (x) \geq n_i$, por lo tanto $\sum_{x \in} \cup^{k}_{i=1} T_i \ \delta_G(x) = \sum_{x \in} V\delta_G(x)=2m \geq k \cdot n$. Así el número de iteraciones del procedimiento Seek_Chromatic_Number(G) podría ser $k \leq (2m)/n=\delta$ y $[\delta(G_0)]+ 1$ colores podrían ser requeridos para colorear el grafo introducido G.

Nótese que el principal propósito de considerar $G0 = G\text{-}I_B$ al empezar el procedimiento iterativo Seek_Chromatic_Number(G) es reducir la posibilidad que G_i pueda ser un subgrafo desconectado. pero si durante una iteración del procedimiento G_i es desconectado, entonces $X(G_i)=max\{X(H_1)\ldots\ldots,X(H_t)\}$ donde $X(H_i)\ldots\ldots,X(H_t)$ son las diferentes componentes conectados de G_i y el número de colores para colorear G podría no ser el límite superior dado por $\delta(G_0)+2$.

Uno de los pasos de mayor consumo en tiempo dentro de “Construir_MIS” es reconocer puntos de articulación esta tarea se hace en tiempo $O(m+n)$ y asume que $m>n$, que son los casos cuando “Construir_MIS” es llamado, el total de tiempo para reconocer puntos de articulación es $O(m)=O(2m)$.

El número de iteraciones en el procedimiento “Construir_MIS” es de orden $O(n/3)$ por que $n/3$ nodos pueden ser removidos del grafo original para formar un grafo acíclico. Y la estructura del “mientras” es de orden n porque es el máximo valor para algún $|N(x)|$. Y el segundo paso (para construir el MIS inicial) requiere en el peor caso de tiempo $O(n)$. Entonces “Construir_MIS” tiene un tiempo de complejidad en el peor de los casos de $O(n^2)=(n/3) \cdot n$.

Uno de los pasos más costosos con respecto al tiempo de complejidad, del procedimiento Seek_Chromatic_Number(G) es la condición del “mientras” cuyo cuerpo tiene un tiempo de complejidad de $O(n^2)$, dado que este consiste en llamar a “Construccion_MIS”.

Y el número de iteraciones es proporcional a $\delta(G).\frac{n}{n-1}$, entonces en el peor de los casos el total tiempo del procedimiento podría ser $(n^2) \cdot (\frac{2m}{n}) \cdot (\frac{n}{n-1})=2\frac{(n \cdot n)(m)}{n-1} \approx (2 \cdot m \cdot n)$.

Entonces un límite superior del tiempo de complejidad de nuestro procedimiento sería $O(m \cdot n)$, que es un valor polinomial en el tamaño de la entrada del grafo G.

VII) Conclusiones y Trabajo a Futuro

El problema del coloreo de grafos es un problema que mantiene desafíos matemáticos y computacionales actuales. Aun cuando se tienen varios años en el análisis y desarrollo de métodos de solución, el problema del coloreo de vértices de un grafo, al igual que cualquier problema de la clase de complejidad NP-completo, siempre serán de gran importancia como objeto de estudio y mientras no se responda con precisión la pregunta ¿es P = NP?, se tendrá la necesidad de proponer nuevos algoritmos o mejorar algoritmos para resolver problemas de la clase NP.

En este proyecto de tesis, se presentó un nuevo algoritmo de tiempo polinomial para determinar el número cromático de un grafo G. Dada una entrada de un grafo conectado G lo primero que se busca y se obtiene es un componente bipartito denotado por I_B, por tanto $G_0=G-I_B$.

El principal propósito es construir los MISK que cumplan la condición: $\delta G_i(K) \geq |V(G_i)| - 1$, al finalizar de encontrar todos los MIS K se procede a hacer la unión $I_B \cup G$.

Dado el Algoritmo de Seek_Chromatic_Number(G) o búsqueda de numero Cromático presentado, se implementó una versión del mismo usando el lenguaje JAVA, lo que nos proporciona un sistema para poder introducir un grafo y obtener el resultado del coloreo de manera gráfica.

El resultado de la investigación y la implementación fue de gran motivación para continuar investigando y desarrollando, cabe resaltar que se demostró que aunque es difícil el problema del coloreo siendo este un problema NP-Completo, se pueden construir algoritmos y llevarlos a la implementación de un programa de cómputo para poder resolverlos.

Para trabajos a futuros se propone añadir esta implementación a un software que resuelva un problema real como lo es la calendarización de horarios, en otro caso se propone también llevar este algoritmo a un ambiente web y poder utilizarlo para seguir probando diferentes grafos, más aun para recibir peticiones y resolver algún problema real.

No sin antes continuar analizando el algoritmo para poder mejorarlo en algún aspecto interno para hacer más eficiente dicho programa de cómputo.

VIII) Bibliografía y Referencias

[1] Gary Chartrand, Ping Zhang, (2005), Introduction to Graph Theory, Mc Graw-Hill Int. Edition.

[2] Golumbic, M.C., (2004), Algorithmic Graph Theory and Perfect Graphs, 2nd edn. North Holland.

[3] Bodlaender H.L., Kratsch D., (2006), An exact algorithm for graph colouring with polynomial memory, Technical Report UU-CS-2006-015, www.cs.uu.nl.

[4] Javier Aurelio Martínez Castillo,3-Coloreo de grafos basado en ciclos, tesis.

[5] Guillermo De Ita, Javier A. Castillo, (October 2012), Recognizing 3-colorings cycle-patterns on graphs, Jour. Pattern Recognition Letters.

[6] Joan Gimbert Quintilla, Els grafs com a models matemàtics, http://web.udl.es/usuaris/p4088280/

[7] Fausto A. Toranzos,(1976), Introducción a la teoría de grafos, Introducción a la teoría de grafos, Washington: Secretaria General de la Organización de los Estados Americanos.

[8] Reinhard Diestel. Graph Theory, Fourth Edition, Springer.

[9] Nancy Maribel Arratia Martínez,Coloración de Grafos, http://yalma.fime.uanl.mx/~maribel/proyecto_progra.html

[10] Hernández Penalver, Matemática discreta (coloración), http://libroweb.alfaomega.com.mx/catalogo/matematicascomputacion/libreacceso/reflector/ovas_statics/cap7/

[11] Abraham Iniesto Díaz y Juan Carlos Delgado Núñez, Algoritmos de coloración, http://www.dma.fi.upm.es/gregorio/grafos/IAGraph/coloracion.html.

[12] Adam Drozdek, (2007), Estructuras de datos y algoritmos con Java, International Thomson Editores.

[13] Mark Allen Weiss, (2000), Estructuras de Datos en Java, Addison-Wesley, ISBN 9788478290352

[14] Guillermo De Ita, J. Raymundo Marcial Romero, J. A. Hernandez, A New Upper Bound for the Chromatic Number of a Graph,

[15] Alfred V. Aho, John E. Hopcroft ; Jeffrey D. Ullman, (1998), Estructura De Datos y Algoritmos (2 Edicion), Alhambra Mexicana, S.A., ISBN 9789684443457

[16] Julián Urbano, (2007), El Teorema de los 4 Colores Aplicado a los compiladores, http://www.caerolus.com/category/matematicas