UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Estructuras Discretas
Grupo 5

Ing. Orlando Zaldívar Zamorategui

Proyecto:
Tutorial: Teoría de Grafos.
Manipulación de Gráficas.

Equipo 2:
Maya Torres Bruno
Ordoñez Figueroa María Fernanda
Castañeda Luna Valeria
Barqueras Capultitla Elian Serge
Uriel Alejandro Vega Reyes
Ortiz Valles Joaquín Rafael

Fecha de asignación.	Fecha de elaboración:	Fecha de entrega:
22/Octubre/2024	22/Octubre/2024 - 4/Noviembre/2024	5/Noviembre/2024

Índice

Teoría de Grafos y Manipulación de Gráficas..... 3
1. Objetivo:..... 3
2. Introducción..... 4
3. Desarrollo..... 5
3.1 Introducción a la Teoría de Grafos y Estructuras Discretas..... 5
Teoría de Grafos..... 5
3.1.1 ¿Qué es un Grafo?..... 5
3.1.2 Contexto Histórico..... 5
3.1.3 Tipos de Grafos y Propiedades Clave..... 6
Estructuras Discretas..... 6
3.1.4 Conjuntos..... 6
3.1.5 Relaciones y Funciones..... 6
3.1.6 Álgebra Booleana..... 6
3.1.7 Principio de Inclusión-Exclusión..... 7
3.1.8 Aplicaciones de la Teoría de Grafos y Estructuras Discretas..... 7
3.2 Conceptos Básicos de Grafos..... 7
3.2.1 Teoría de Grafos..... 7
3.2.2 Aplicaciones..... 7
3.3 Vértices, aristas, grados. Tipos de grafos..... 8
3.3.1 Vértices y Aristas..... 8
3.3.2 Grado de un Vértice..... 9
3.3.3 Tipos de Grafos..... 9
3.4 Representaciones: Matriz de adyacencia y lista de adyacencia..... 13
3.4.1 Matriz de adyacencia..... 13
3.4.2 Lista de adyacencia..... 14
3.5 Recorridos en grafos..... 15
3.5.1. Algoritmos de búsqueda en anchura (BFS)..... 15
CONSTRUCCIÓN DEL ALGORITMO..... 15
3.5.2 Algoritmos de búsqueda en anchura (DFS)..... 15
CONSTRUCCIÓN DEL ALGORITMO..... 15
3.6 Isomorfismo y Propiedades de Grafos..... 16
3.6.1. Isomorfismo de Grafos..... 16
3.7 Software para manipulación de Grafos..... 17
3.7.1 Historia y Concepto del Software Libre(Contexto)..... 17
3.7.2 Aplicación del Software Libre en la Educación Matemática..... 17
3.7.3 Herramientas de Software Libre en Matemáticas Discretas..... 18
3.7.3.1 Maxima:..... 18
3.7.3.2 GeoGebra:..... 18
3.7.3.3 Dia:..... 18

3.7.4 Beneficios Pedagógicos del Software Libre en la Educación..... 18
4. Ejemplos..... 19
4.1 Lista de adyacencia y matriz de adyacencia:.....19
4.2 Otro ejemplo de lista y matriz de adyacencia:.....21
4.3 Ejemplo de Algoritmo de Búsqueda en Profundidad (DFS):.....23
4.4 Ejemplo de Algoritmo De Búsqueda En Anchura (BFS):.....25
4.5 Ejemplo de isomorfismo de grafos:.....28
5. Conclusión.....30
6. Bibliografía..... 31

Teoría de Grafos y Manipulación de Gráficas

1. Objetivo:

El objetivo de este proyecto es desarrollar un recurso completo e interactivo para el aprendizaje de la teoría de grafos, con el propósito de abordar la falta de material unificado que facilite tanto la comprensión teórica como la aplicación práctica de los conceptos.
Este recurso está diseñado para estudiantes de ingeniería, en especial aquellos interesados en áreas como informática, telecomunicaciones y ciencias computacionales, donde la teoría de grafos y sus algoritmos son esenciales.

El proyecto se estructura en torno a los siguientes componentes:

- Documentación Temática: Una cobertura completa de temas relevantes, que abarca desde fundamentos de grafos hasta conceptos avanzados como recorridos, algoritmos y propiedades específicas de los grafos.
- Software de Visualización y Manipulación: Herramientas interactivas que permiten a los estudiantes visualizar estructuras de grafos, explorar algoritmos de búsqueda y optimización, y analizar estructuras discretas en tiempo real.
- Ejemplos y Ejercicios Prácticos: Ejemplos resueltos y problemas prácticos que reflejan situaciones del mundo real, como redes de comunicación y optimización de rutas, que permiten a los estudiantes aplicar sus conocimientos.
- Cuestionarios y Evaluación: Cuestionarios interactivos diseñados para evaluar la comprensión de los temas y reforzar el aprendizaje, brindando retroalimentación y guía para fortalecer áreas de mejora.

Este tutorial tiene como fin capacitar a los estudiantes para abordar problemas de lógica y optimización mediante el uso de grafos, mejorando así sus habilidades en el análisis y resolución de problemas. Los requerimientos del sistema incluyen un navegador compatible, acceso a internet para actualizaciones y herramientas de visualización gráfica integradas. Al final de este tutorial, los estudiantes serán capaces de entender y aplicar de forma autónoma los conceptos fundamentales y avanzados de la teoría de grafos en contextos aplicados y académicos.

2. Introducción

La teoría y manipulación de gráficas es un campo que combina tanto fundamentos teóricos como aplicaciones prácticas, ofreciendo técnicas para modelar y resolver problemas complejos en diferentes disciplinas científicas y tecnológicas.

La Teoría de Gráficas es un área de las matemáticas y la informática que se dedica al estudio de estructuras discretas conocidas como gráficas o grafos. Los grafos son estructuras que permiten modelar relaciones entre objetos mediante nodos (o vértices) y conexiones (aristas). La manipulación de gráficas se refiere a las técnicas y herramientas utilizadas para crear, modificar, y analizar gráficas. Esto incluye desde agregar o eliminar vértices y aristas, hasta aplicar algoritmos específicos de búsqueda, como el recorrido en anchura (BFS) y en profundidad (DFS). Estas técnicas son cruciales para resolver problemas de optimización, como la búsqueda de caminos más cortos, la detección de ciclos y el análisis de conectividad.

Existen diversas herramientas de software que facilitan el trabajo con gráficas, estas permiten modelar gráficas complejas, aplicar algoritmos avanzados, y visualizar las estructuras para un análisis más profundo. Estas herramientas simplifican la implementación de algoritmos, el análisis de grandes conjuntos de datos y la resolución de problemas prácticos en redes y sistemas complejos.

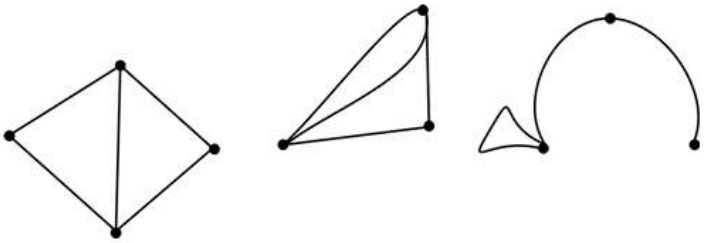
3. Desarrollo

3.1 Introducción a la Teoría de Grafos y Estructuras Discretas.

Teoría de Grafos

3.1.1 ¿Qué es un Grafo?

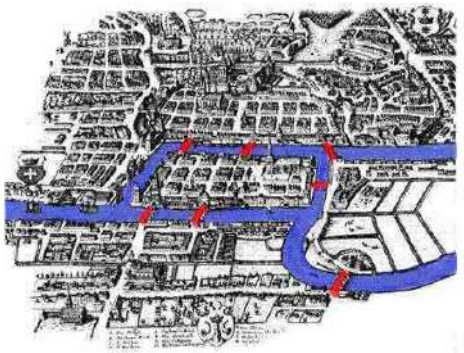
Un grafo es una estructura matemática compuesta por un conjunto de nodos (también conocidos como vértices) y aristas (o arcos) que los conectan. Las aristas establecen conexiones entre pares de nodos. Los grafos son herramientas útiles para modelar relaciones y estructuras que se pueden representar a través de estas conexiones.
Entre los ejemplos más comunes se encuentran las redes sociales, las redes informáticas y los circuitos electrónicos.



Cánovas Peña, J. S. (s. f.). Teoría de grafos. Departamento de Matemática Aplicada y Estadística. Figura 1.: Ejemplos de grafo simple, multigrafo y pseudografo, p. 3.

3.1.2 Contexto Histórico

La teoría de los grafos tiene sus raíces en el siglo XVIII gracias al matemático suizo Leonhard Euler. En 1736, Euler se enfrentó al famoso problema de los siete puentes de Königsberg, intentando averiguar si era posible cruzar cada puente de la ciudad una sola vez. Este problema llevó a la creación de los conceptos básicos de la teoría de grafos.



Con el tiempo, varios matemáticos hicieron aportes importantes a esta teoría:
En 1847, Gustav Kirchhoff utilizó la teoría de grafos para analizar circuitos eléctricos, formulando las leyes de Kirchhoff.
En 1852, Francis Guthrie planteó el problema de los cuatro colores, que fue resuelto casi un siglo después, ayudando a definir términos clave en la teoría.
En 1857, Arthur Cayley usó grafos para estudiar isómeros químicos.
El término "grafo" proviene del inglés "graphic notation", utilizado por primera vez por Edward Frankland y adoptado por Alexander Crum Brown en 1884. Desde entonces, la teoría ha evolucionado y se ha convertido en una herramienta esencial en campos como la informática y el análisis de redes sociales.

3.1.3 Tipos de Grafos y Propiedades Clave

El documento presente tiene la finalidad de dar a conocer las características de dichos grafos, dichas características se muestran en los diferentes tipos de grafos que existen, tales grafos serían: Dirigidos (Digrafos), No Dirigidos, Ponderados y No Ponderados.
A su vez los grafos tienen propiedades que son de suma importancia, ya sea para su manipulación y entendimiento, tales propiedades son las siguientes: Grado de un Nodo, Camino, Ciclo, Conectividad.

Estructuras Discretas

3.1.4 Conjuntos

Un conjunto es una agrupación de elementos. En el ámbito de las estructuras discretas, se analizan operaciones sobre conjuntos, como la unión, intersección y diferencia.

3.1.5 Relaciones y Funciones

Relaciones: Se establece una relación entre los elementos de dos conjuntos. Un ejemplo común es la relación de equivalencia.
Funciones: Son asignaciones que vinculan elementos de un conjunto con otro. Las funciones pueden ser inyectivas, suprayectivas o biyectivas.

3.1.6 Álgebra Booleana

El álgebra booleana es un sistema matemático que se utiliza para trabajar con valores de verdad (verdadero/falso). Es fundamental en el diseño de circuitos digitales y en la lógica de programación.

3.1.7 Principio de Inclusión-Exclusión

Este principio se aplica para determinar el tamaño de la unión de varios conjuntos y es crucial en combinatoria y teoría de probabilidades.

3.1.8 Aplicaciones de la Teoría de Grafos y Estructuras Discretas

Redes Informáticas: Se utilizan para modelar conexiones y flujos de datos.
Algoritmos de Optimización: Permiten encontrar el camino más corto o el árbol de expansión mínima, entre otros.
Bases de Datos: Ayudan a modelar las relaciones entre diferentes entidades.
Inteligencia Artificial: Se emplean para representar conocimiento y facilitar el razonamiento.

3.2 Conceptos Básicos de Grafos

3.2.1 Teoría de Grafos

La teoría de grafos es una disciplina que estudia las relaciones entre objetos mediante estructuras llamadas grafos, formadas por vértices y aristas. Los grafos se pueden representar de diversas maneras, siendo las más comunes la matriz de adyacencia y la lista de adyacencia, cada una con sus ventajas dependiendo del tipo de grafo analizado.

Las propiedades esenciales de los grafos incluyen su clasificación como bipartitos, planos, densos o dispersos, lo cual es crucial para comprender su estructura y comportamiento en distintos contextos.

Esta teoría abarca varios algoritmos clave que permiten realizar diversas operaciones y análisis. Entre ellos destacan la búsqueda en profundidad (DFS) y la búsqueda en anchura (BFS) para explorar los vértices de un grafo. Asimismo, el algoritmo de Dijkstra es esencial para encontrar el camino más corto en grafos ponderados, mientras que los algoritmos de Kruskal y Prim son vitales para determinar el Árbol de Expansión Mínima.

3.2.2 Aplicaciones

Redes de Transporte: Se utilizan para modelar y optimizar rutas.

Sistemas de Recomendación: Representan las relaciones entre usuarios y productos.

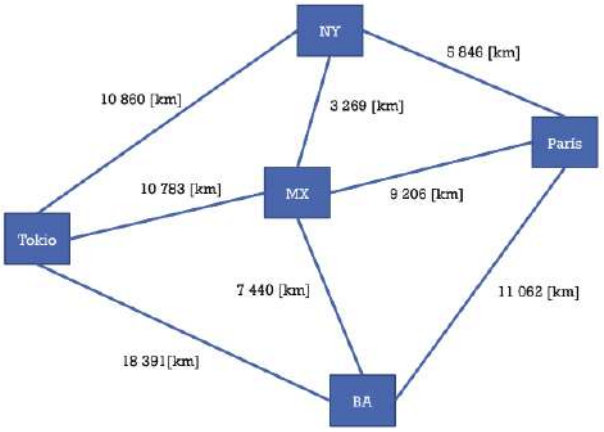
Biología Computacional: Facilitan el análisis de redes relacionadas con proteínas y genes.

Teoría de Redes Sociales: Se centran en el estudio de las interacciones y relaciones entre individuos.

3.3 Vértices, aristas, grados. Tipos de grafos.

3.3.1 Vértices y Aristas

En un grafo, los vértices (o nodos) son los elementos que representan las entidades u objetos de interés, mientras que las aristas (o arcos) son las conexiones entre esos vértices.



Solano, J. (2024). Algoritmos de grafos. Universidad Nacional Autónoma de México, Facultad de Ingeniería, Departamento de Computación. [Grafo de distancias entre ciudades]. Página 12.

Para entender a profundidad estos conceptos, es esencial tener en cuenta:

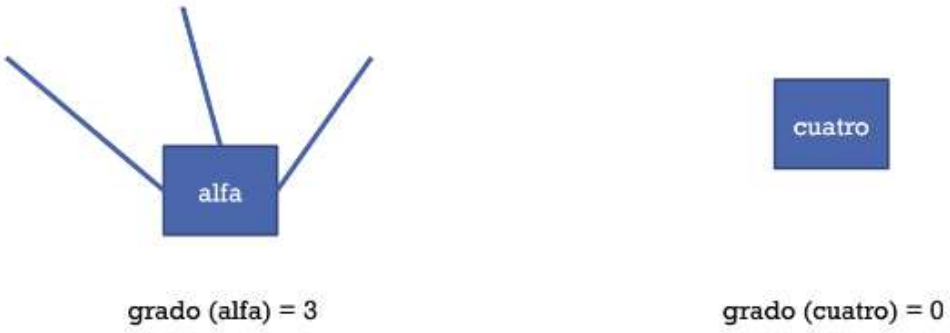
- **Vértices (V):** Son los elementos o puntos que forman un grafo. En un grafo denotado por *G*, el conjunto de vértices se expresa como *V(G)*. Cada vértice puede tener múltiples conexiones, reflejadas mediante aristas.
- **Aristas (A):** Son los enlaces entre los vértices. Se denotan como pares de vértices *(u,v)*. Estas conexiones pueden ser:
    - **No dirigidas:** Cuando la relación entre dos vértices no tiene una dirección específica.

    - **Dirigidas:** Cuando existe una dirección que va desde un vértice *u* hacia otro vértice *v*. Esto se denota como (u→v).

"Cada arista solo puede unir un par de nodos del conjunto V(G). Una arista que enlaza al nodo u con el nodo v se denota como: a=(u, v)" *(Solano, s.f., p. 14).*

3.3.2 Grado de un Vértice

El grado de un vértice es el número de aristas que inciden en él. Esta métrica es fundamental para estudiar la estructura de un grafo.



Solano, J. (2024). Algoritmos de grafos. Universidad Nacional Autónoma de México, Facultad de Ingeniería, Departamento de Computación. [Grado de dos vértices]. Página 18.

Los grados se dividen en:

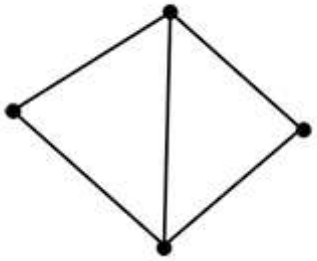
- **Grado de un vértice en grafos no dirigidos:** En un grafo no dirigido, el grado de un vértice v (denotado como grado(v)) es simplemente el número de aristas conectadas a él. Si un vértice tiene un grado igual a cero, se le denomina vértice aislado, ya que no tiene aristas incidentes.

- **Grado de entrada y grado de salida en grafos dirigidos:** En un grafo dirigido, se distinguen dos tipos de grados para un vértice v:
    - **Grado de entrada:** Es el número de aristas que llegan al vértice v.
    - **Grado de salida:** Es el número de aristas que salen del vértice v.

3.3.3 Tipos de Grafos

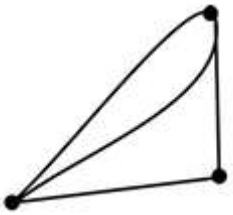
Los grafos se clasifican en diversos tipos según sus características y propiedades estructurales:

1. **Grafos simples**: Son aquellos que no tienen lazos ni aristas múltiples entre pares de vértices. Son los grafos más básicos.



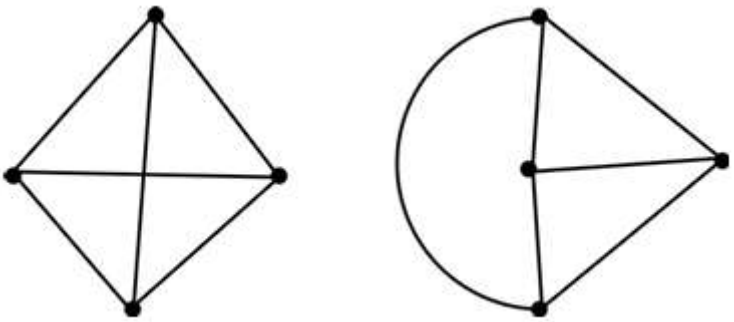
Cánovas Peña, J. S. (s. f.). Teoría de grafos. Departamento de Matemática Aplicada y Estadística. Figura 1.2: Ejemplos de grafo simple, multigrafo y pseudografo, p. 3.

2. **Multigrafos**: Permiten múltiples aristas entre un par de vértices y pueden tener lazos. Este tipo de grafos se utiliza para modelar situaciones en las que existen conexiones redundantes o múltiples relaciones entre elementos.



Cánovas Peña, J. S. (s. f.). Teoría de grafos. Departamento de Matemática Aplicada y Estadística. Figura 1.2: Ejemplos de grafo simple, multigrafo y pseudografo, p. 3.

3. **Grafos completos**: Son aquellos en los que cada par de vértices está conectado por una arista. En un grafo completo de n vértices, cada vértice se conecta con n−1 vértices más.

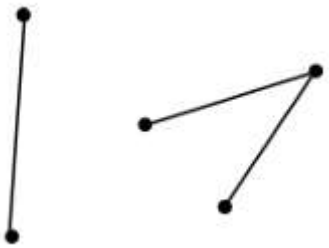


Cánovas Peña, J. S. (s. f.). Teoría de grafos. Departamento de Matemática Aplicada y Estadística. Figura 1.3: Ejemplos de grafo completo, p. 4.

4. **Grafos conexos**: Son aquellos en los que existe al menos un camino entre cualquier par de vértices. La conectividad en un grafo es clave para modelar redes de transporte, comunicación y otros sistemas interconectados.

Cánovas Peña, J. S. (s. f.). Teoría de grafos. Departamento de Matemática Aplicada y Estadística. Figura 1.5: Ejemplos de grafo conexo, p. 5.

5. **Grafo no conexo**: Un grafo no conexo es aquel en el que existen vértices que no están conectados entre sí a través de ningún camino. Es decir, no se puede ir de un vértice a otro mediante una secuencia de aristas. Un grafo no conexo puede descomponerse en componentes conexos, que son subgrafos conexos donde cada componente conexo no tiene conexiones con los demás componentes del grafo.

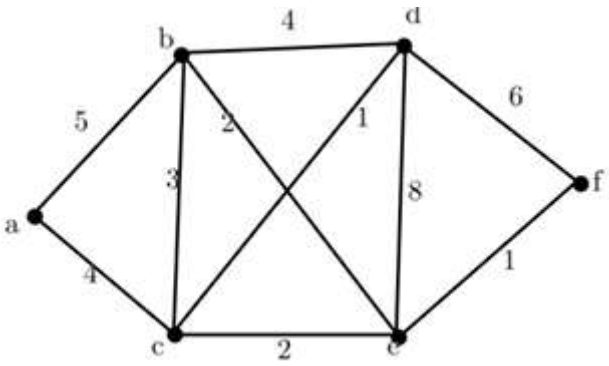


Cánovas Peña, J. S. (s. f.). Teoría de grafos. Departamento de Matemática Aplicada y Estadística. Figura 1.5: Ejemplos de grafo no conexo, p. 5

6. **Árboles**: Un árbol es un tipo especial de grafo conexo que no contiene ciclos y tiene un único camino entre cada par de vértices. Los árboles son fundamentales para representar estructuras jerárquicas y rutas únicas en diferentes contextos.

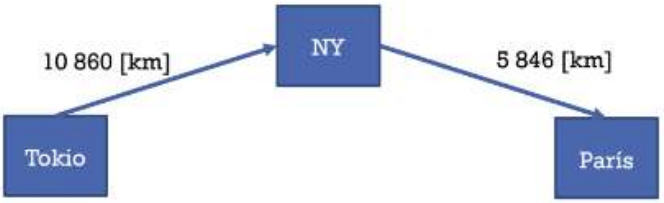
Cánovas Peña, J. S. (s. f.). Teoría de grafos. Departamento de Matemática Aplicada y Estadística. Figura 1.10: Ejemplos árboles, p. 10

7. **Grafos ponderados**: En este tipo de grafos, cada arista tiene un valor numérico asociado (peso), que puede representar una distancia, costo o alguna otra medida. Los grafos ponderados son útiles para resolver problemas de optimización como la búsqueda de rutas mínimas.



Cánovas Peña, J. S. (s. f.). Teoría de grafos. Departamento de Matemática Aplicada y Estadística. Figura 1.12: Ejemplo de grafo ponderado, p. 10.

8. **Grafo dirigido:** En la teoría de grafos, una gráfica dirigida o digráfica se caracteriza por la presencia de aristas que tienen una dirección específica. Como menciona Solano (s.f.), "una gráfica dirigida (o digráfica) G se caracteriza porque cada arista a tiene una dirección asignada y, por tanto, cada arista está asociada a un par ordenado (u, v) de vértices" (p. 31).



Solano, J. (2024). Algoritmos de grafos. Universidad Nacional Autónoma de México, Facultad de Ingeniería, Departamento de Computación. [Grafo dirigido]. Página 31..

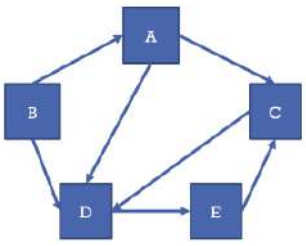
3.4 Representaciones: Matriz de adyacencia y lista de adyacencia.

En la teoría de grafos, existen varias formas de representar gráficamente las conexiones entre los vértices, y dos de las más comunes son la matriz de adyacencia y la lista de adyacencia. Estas estructuras son ampliamente utilizadas en informática y matemáticas para trabajar con grafos de manera eficiente en términos de espacio y velocidad, dependiendo del tipo de grafo y de las operaciones a realizar.

3.4.1 Matriz de adyacencia.

"Una matriz de adyacencia es una matriz booleana de orden n, donde n indica el número de vértices de G. Tanto los renglones como las columnas de la matriz representan los nodos de la gráfica y su contenido representa la existencia (1) o no (0) de arcos entre los nodos i y j" (Solano, s.f., p. 35).

A continuación, vemos el ejemplo de un grafo dirigido y su matriz de adyacencia.



Gráfica G (v, a)

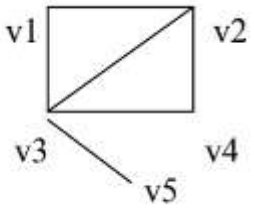
	A	B	C	D	E
A	0	0	1	1	0
B	1	0	0	1	0
C	0	0	0	1	0
D	0	0	0	0	1
E	0	0	1	0	0

Matriz de adyacencia

Solano, J. (2024). Algoritmos de grafos. Universidad Nacional Autónoma de México, Facultad de Ingeniería, Departamento de Computación. [Grafo dirigido y su matriz de adyacencia]. Página 36.

Cuando se trata de **grafos ponderados** en lugar de 1 el valor que tomará será el peso de la arista.
Si el grafo es **no dirigido** hay que asegurarse de que se marca con un 1 (o con el peso) tanto la entrada a[i][j] como la entrada a[j][i], puesto que se puede recorrer en ambos sentidos. En el caso de un grafo no dirigido, la matriz de adyacencia siempre es simétrica.

	v1	*v2*	*v3*	*v4*	*v5*
v1	0	1	1	0	0
v2	1	0	1	1	0
v3	1	1	0	1	1
v4	0	1	1	0	0
v5	0	0	1	0	0



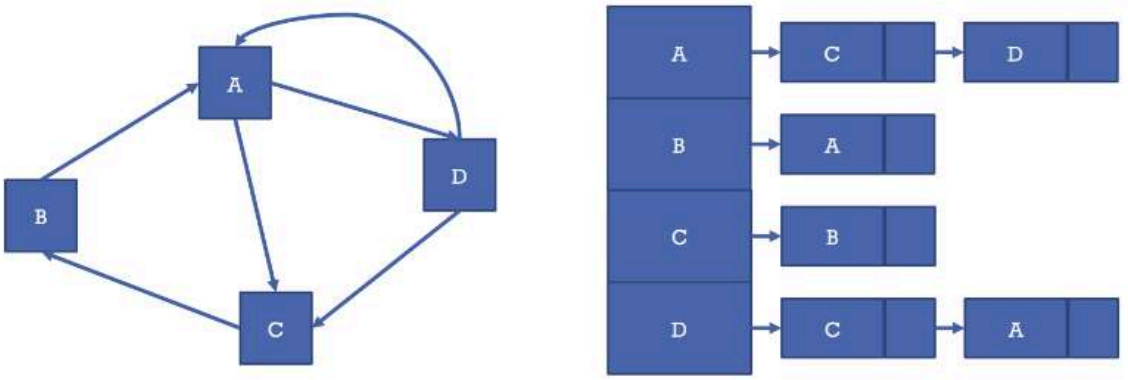
Gracias a la matriz de adyacencia, podemos conocer cuántos caminos existen de un vértice a otro:

Theorem 1.3.1 *Sean $G = (V, E)$ un grafo simple con $|V| = n$ y $\mathbf{A}$ su matriz de adyacencia. Dado $k \in \mathbb{N}$, sea $\mathbf{A}^k = (a_{ij}^{[k]})$. Entonces $a_{ij}^{[k]}$ es el número de caminos de longitud k que unen v_i con v_j para todo $i, j \in \{1, 2, ..., n\}$.*

Cánovas Peña, J. S. (s. f.). *Teoría de grafos*. Departamento de Matemática Aplicada y Estadística. Teorema 1.3.1: La matriz de adyacencia, p. 9.

3.4.2 Lista de adyacencia.

Una lista de adyacencia es una forma de representar grafos que se utiliza comúnmente en teoría de gráficas. En esta representación, cada vértice del grafo tiene una lista asociada que contiene todos los vértices a los que está conectado mediante aristas.



Solano, J. (2024). Algoritmos de grafos. Universidad Nacional Autónoma de México, Facultad de Ingeniería, Departamento de Computación. [Grafo dirigido y su lista de adyacencia]. Página 40.

En un grafo no dirigido, la lista de adyacencia de un vértice v incluye todos los vértices u tales que hay una arista entre v y u.

3.5 Recorridos en grafos

3.5.1. Algoritmos de búsqueda en anchura (BFS)

La idea básica de la búsqueda en anchura es desplegarse a tantos vértices como sea posible antes de penetrar en profundidad dentro de un árbol. Esto significa que visitaremos todos los vértices adyacentes a uno dado antes de cambiar de nivel. Es uno de los algoritmos fundamentales para explorar grafos y tiene aplicaciones en problemas de conectividad, búsqueda de caminos más cortos en grafos no ponderados, y análisis de redes.

CONSTRUCCIÓN DEL ALGORITMO

Dado un vértice inicial en una gráfica G=(V,E), el algoritmo BFS sigue los siguientes pasos:

INICIO
- Marca el nodo de inicio como visitado y este último se añade a la cola

RECORRIDO
- Mientras la cola no esté vacía, se desencola un vértice de la cola
- Para cada vértice recorrido, se comprueba si su vértice vecino ha sido visitado
- En caso de no ser visitado, se recorre, se marca como visitado y se añade a la cola

FIN
- El algoritmo termina cuando la cola está vacía, lo que significa que se han explorado todos los vértices alcanzables desde el vértice inicial.

3.5.2 Algoritmos de búsqueda en anchura (DFS)

La idea básica de búsqueda en profundidad es penetrar tan profundamente como sea posible dentro de un árbol antes de desplegarse a otros vértices. Esto se consigue al tomar el nuevo vértice adyacente al último de los posibles vértices anteriores. Este algoritmo es ampliamente utilizado en la teoría de grafos y tiene aplicaciones en la detección de ciclos, el análisis de componentes conexos, y en problemas de recorrido y conectividad en redes.

CONSTRUCCIÓN DEL ALGORITMO

Dado un vértice inicial en una gráfica G=(V,E), el algoritmo DFS sigue estos pasos:

INICIO
- Selecciona un vértice de inicio y se marca como visitado
- Se exploran los nodos vecinos del vértice de inicio

RECORRIDO

- Si algún nodo no ha sido visitado se marca como visitado y se realiza una llamada recursiva para el vértice vecino no visitado o bien se apila y se sigue el proceso iterativamente.
    - RETROCESO
        - Una vez que todos los vecinos de un vértice han sido visitados, el algoritmo "retrocede" al vértice anterior para explorar otros caminos no visitados.

FIN

- El proceso continúa hasta que todos los vértices alcanzables desde el vértice de inicio hayan sido visitados.

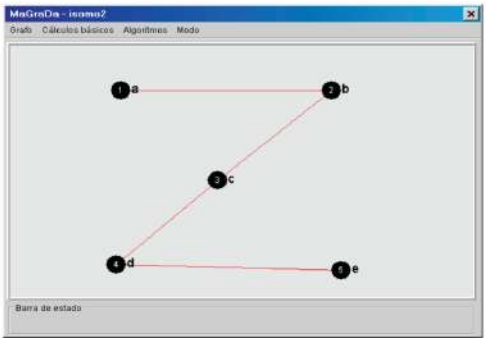
3.6 Isomorfismo y Propiedades de Grafos

3.6.1. Isomorfismo de Grafos

Para que dos grafos sean isomorfos, debe de existir una correspondencia uno a uno entre los nodos de los dos grafos y, además, conservan la adyacencia entre los nodos, así como la dirección de las aristas, si estas existen.

Para ser isomorfos dos grafos, deben de cumplir con:

1. Dos grafos isomorfos deben tener el mismo número de vértices, ya que hay una aplicación biyectiva entre los conjuntos de vértices de los grafos.
2. Dos grafos isomorfos deben tener el mismo número de aristas.
3. Los grados de los vértices en grafos simples isomorfos deben ser los mismos. Es decir, un vértice v de grado d en G1 se corresponde con un vértice f(v) de grado d en G2, ya que un vértice w en G1 es adyacente a v si y sólo si f(v) y f(w) son adyacentes en G2.



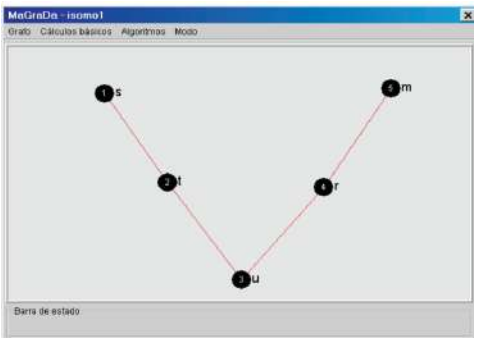


Foto 1

Foto 2

Caballero Palomino, M. Á., Migallón Gomis, V., & Penadés Martínez, J. (Año). Prácticas de matemática discreta con MaGraDa [Foto 1]. Universidad de Alicante. p. 54.

Caballero Palomino, M. Á., Migallón Gomis, V., & Penadés Martínez, J. (Año). Prácticas de matemática discreta con MaGraDa [Foto 2]. Universidad de Alicante. p. 54.

En general, cuando dos grafos son isomorfos pensamos en ellos como si fueran el mismo grafo. dos grafos no son isomorfos si no cuentan con el mismo número de vértices y aristas. Si efectivamente cuentan con el mismo número de vértices y aristas, tampoco se garantiza que sean isomorfos.

En ese sentido, verificar la correspondencia entre grados puede ser de utilidad. Sin embargo, incluso si coinciden en los grados de sus vértices, ello no implica que sean isomorfos. La única forma de probarlo es afirmar o refutar la existencia de la función biyectiva f: V(G) -> V(G*). Sea n = Card(V(G)) = Card(V(G*))(Emparejar los vértices de un grafo G y G*), Entonces, de acuerdo a lo anterior, se tendrían que examinar a lo más n!(n factorial) posibles funciones.

3.7 Software para manipulación de Grafos

3.7.1 Historia y Concepto del Software Libre(Contexto)

El **software libre** se define como aquel cuyo código fuente es accesible y puede ser libremente modificado, utilizado y distribuido por cualquier usuario. Este concepto surgió en la década de 1980, impulsado por Richard Stallman, quien fundó el proyecto GNU con la visión de crear software que promoviera la libertad de los usuarios para personalizar y distribuir programas según sus necesidades. Las cuatro libertades esenciales del software libre son: la libertad de ejecutar el programa con cualquier propósito, estudiar y modificar su funcionamiento, redistribuir copias y mejorar el programa en beneficio de la comunidad (Villalpando Becerra & Pantoja Rangel, 2016, p. 18).

A diferencia del software propietario, que restringe el acceso al código fuente y limita los derechos de uso, el software libre se asocia con un modelo colaborativo, en el que desarrolladores de todo el mundo contribuyen a su mejora continua. Esta filosofía fomenta la accesibilidad y la personalización del software, promoviendo la innovación en sectores como la educación matemática (Villalpando Becerra & Pantoja Rangel, 2016, p. 19).

3.7.2 Aplicación del Software Libre en la Educación Matemática

El software libre se ha convertido en una herramienta poderosa en la enseñanza de conceptos complejos en matemáticas discretas, como los grafos y estructuras algebraicas. En entornos educativos, permite que los estudiantes visualicen y manipulen elementos matemáticos de manera dinámica e interactiva, lo que es fundamental para entender estructuras abstractas y facilitar un aprendizaje significativo. Además, al eliminar barreras económicas, el software libre proporciona acceso a herramientas avanzadas, lo cual es especialmente valioso para instituciones con recursos limitados (Villalpando Becerra & Pantoja Rangel, 2016, p. 20).

3.7.3 Herramientas de Software Libre en Matemáticas Discretas

Entre las herramientas de software libre destacadas en el ámbito de las matemáticas discretas se encuentran:

3.7.3.1 Maxima:

Este sistema de álgebra computacional permite realizar cálculos simbólicos y numéricos, facilitando el análisis de estructuras complejas como las matrices de adyacencia en grafos. Maxima es ideal para el desarrollo de habilidades analíticas, promoviendo la comprensión de la relación entre representaciones gráficas y algebraicas (Villalpando Becerra & Pantoja Rangel, 2016, p. 21).

3.7.3.2 GeoGebra:

Conocido principalmente por su aplicación en geometría, GeoGebra también permite representar y manipular grafos de manera visual e interactiva. Los estudiantes pueden explorar propiedades estructurales de los grafos, analizar caminos y ciclos, y mejorar su comprensión a través de un aprendizaje práctico y visual (Villalpando Becerra & Pantoja Rangel, 2016, p. 22).

3.7.3.3 Dia:

Herramienta de diagramación que facilita la creación de representaciones visuales de nodos y aristas en grafos, permitiendo a los estudiantes analizar conceptos como conectividad y recorridos. Aunque no se dedica exclusivamente a la teoría de grafos, su flexibilidad la convierte en una herramienta eficaz para este tipo de representaciones (Villalpando Becerra & Pantoja Rangel, 2016, p. 23).

3.7.4 Beneficios Pedagógicos del Software Libre en la Educación

El uso del software libre en la educación matemática ofrece beneficios pedagógicos al promover un aprendizaje activo y colaborativo. A través de la manipulación directa de elementos matemáticos, los estudiantes experimentan con las propiedades de los grafos, fortaleciendo su comprensión y sus habilidades en resolución de problemas. Este enfoque favorece la construcción del conocimiento y, al ser de libre acceso, promueve una educación equitativa y accesible (Villalpando Becerra & Pantoja Rangel, 2016, p. 24).

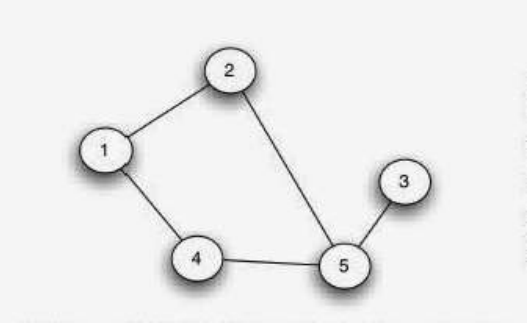
El software libre en matemáticas discretas constituye una herramienta valiosa que empodera tanto a estudiantes como a docentes, facilitando un aprendizaje interactivo y accesible. A través de herramientas como Máxima, GeoGebra y Día, los estudiantes

desarrollan competencias analíticas y tecnológicas esenciales en su formación matemática, contribuyendo a una educación inclusiva y de calidad (Villalpando Becerra & Pantoja Rangel, 2016, p. 25-26).

4. Ejemplos

4.1 Lista de adyacencia y matriz de adyacencia:

Vamos a ver el ejemplo de cómo obtener la matriz y lista de adyacencia del siguiente **grafo no dirigido**:



Como **primer paso**, debemos identificar los vértices y las aristas que los unen dentro del grafo. En este caso, observamos que:

- El vértice 1 está conectado a los vértices 2 y 4.
- El vértice 2 está conectado a los vértices 1 y 5.
- El vértice 3 está conectado al vértice 5.
- El vértice 4 está conectado a los vértices 1 y 5.
- El vértice 5 está conectado a los vértices 2, 3 y 4.

El siguiente paso es crear la lista de adyacencia. Para ello, simplemente mediante una lista ordenaremos a cada vértice con sus respectivos nodos vecinos:

Entonces, la **Lista de Adyacencia** es:

1: [2,4]
2: [1,5]
3: [5]
4: [1,5]
5: [2,3,4]

Ahora procedemos a crear la **matriz de adyacencia** de nuestro grafo.
Primero, tenemos que identificar la cantidad de nodos que hay, ya que si el grafo tiene n vértices, la matriz de adyacencia será de tamaño n×n.
En este caso usaremos una matriz 5×5 (ya que hay 5 vértices) y la inicializamos con valores en cero.

ixj	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

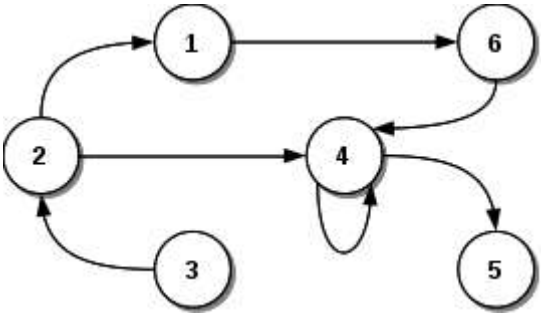
Finalmente, llenaremos la matriz de forma que cada posición (i, j) tendrá un 1 si hay una conexión entre el vértice i(filas) y el vértice j(columnas), y un 0 si no la hay.

ixj	1	2	3	4	5
1	0	1	0	1	0
2	1	0	0	0	1
3	0	0	0	0	1
4	1	0	0	0	1
5	0	1	1	1	0

Así finalizamos el proceso para obtener la matriz y lista de adyacencia del grafo.

4.2 Otro ejemplo de lista y matriz de adyacencia:

Ahora, haremos un ejemplo sobre cómo obtener la matriz y lista de adyacencia de un **grafo dirigido.**



El proceso será muy similar al anterior. Primero tenemos que observar cómo está constituido el grafo: en este caso tenemos los vértices numerados del 1 al 6 y las aristas están dirigidas entre ellos. Observemos las conexiones:

- El vértice 1 tiene una arista dirigida hacia el vértice 6.
- El vértice 2 tiene una arista dirigida hacia el vértice 1 y 4.
- El vértice 3 tiene una arista dirigida hacia el vértice 2.
- El vértice 4 tiene una arista dirigida hacia el vértice 5 y un lazo (una arista que regresa al mismo vértice.
- El vértice 5 no tiene aristas salientes.
- El vértice 6 tiene una arista dirigida hacia el vértice 4

En un **grafo dirigido**, la lista de adyacencia refleja las conexiones en una dirección específica. Esto significa que cada arista tiene un vértice de **origen** y un vértice de **destino**, y solo el vértice de origen incluirá al vértice de destino en su lista de adyacencia. Por lo tanto, en este caso la lista de adyacencia que obtenemos es:

1: [6]
2: [1,4]
3: [2]
4: [4,5]
5: []
6: [4]

Ahora, para elaborar la **matriz de adyacencia**, primero identificamos el número de vértices que este contiene, ya que la matriz de adyacencia será de tamaño nxn (siendo n la cantidad de vértices). Inicialmente, le damos valores de cero, en este caso, la matriz será de 6x6.

ixj	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

Posteriormente, comenzamos con el llenado de la matriz. Para ello, por cada arista dirigida del grafo, coloca un 1 en la posición (i,j) de la matriz si hay una arista dirigida desde el vértice i al vértice j. Si no existe una arista, deja un 0 en esa posición.

Si algún vértice tiene un lazo (una arista que apunta a sí mismo), coloca un 1 en la posición diagonal correspondiente, es decir, en (i,i).

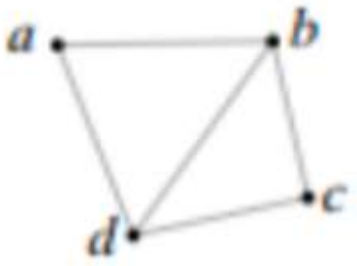
De esta forma obtenemos la siguiente matriz:

ixj	1	2	3	4	5	6
1	0	0	0	0	0	1
2	1	0	0	1	0	0
3	0	1	0	0	0	0
4	0	0	0	1	1	0
5	0	0	0	0	0	0
6	0	0	0	1	0	0

Así finalizamos el proceso de obtener la matriz y lista de adyacencia de un grafo dirigido y que incluía un lazo.

4.3 Ejemplo de Algoritmo de Búsqueda en Profundidad (DFS):

A continuación veremos un ejemplo de cómo se realiza la búsqueda a profundidad (DFS) con el siguiente grafo:



Paso 1: Iniciamos el recorrido en el nodo "a".

-Partimos en el nodo "a" y lo marcamos como visitado.

-Lista de visitados: a

-Desde aquí, tenemos dos opciones de vecinos a visitar: b y d. En DFS, seguimos el camino en profundidad, así que elijamos el primer vecino en orden alfabético, que en este caso es "b".

Paso 2: Moverse al nodo "b".

-Ahora estamos en "b" y lo marcamos como visitado.

-Lista de visitados: a → b

-En "b", tenemos tres vecinos: a, c, y d.

-Como "a" ya ha sido visitado, lo omitimos y nos movemos al siguiente vecino no visitado, que es "c".

Paso 3: Moverse al nodo "c".

-Llegamos a "c" y lo marcamos como visitado.

-Lista de visitados: a → b → c

-En "c", encontramos dos vecinos: b y d.

-Como "b" ya ha sido visitado, omitimos esa opción y nos movemos al siguiente vecino no visitado, que es "d".

Paso 4: Moverse al nodo "d".

-Llegamos a "d" y lo marcamos como visitado.

-Lista de visitados: a → b → c → d

-En "d", tenemos tres vecinos: a, b, y c.

-Sin embargo, todos estos vecinos ya han sido visitados. Por lo tanto, no hay más nodos que explorar desde "d".

Paso 5: Retroceso.

-Dado que ya no hay vecinos no visitados desde "d", retrocedemos al nodo anterior, que es "c".

-En "c", tampoco quedan vecinos no visitados, por lo que retrocedemos nuevamente a "b".

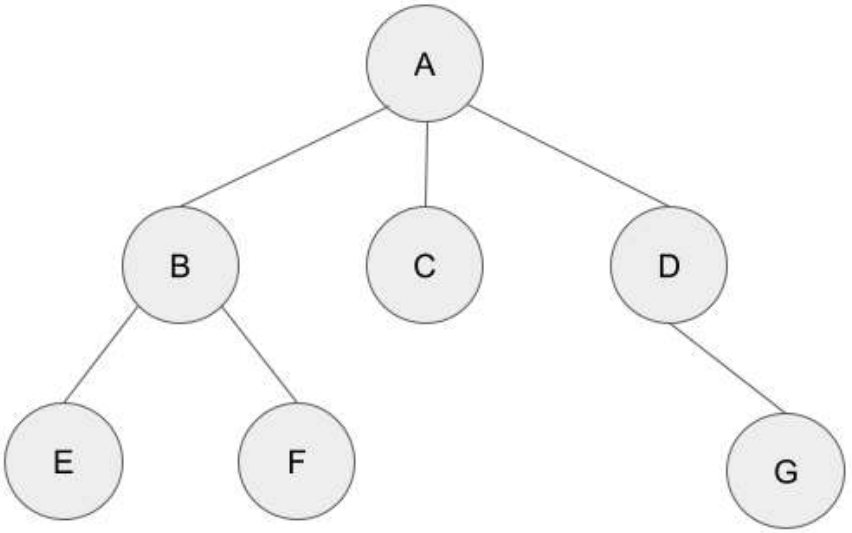
-En "b", tampoco hay vecinos no visitados, así que retrocedemos finalmente a "a".

En "a", todos los vecinos han sido explorados, lo que indica que el recorrido DFS está completo.

Finalmente, podemos concluir que el recorrido DFS encontró y visitó todos los nodos del grafo en el orden a → b → c → d , asegurándose de que cada rama ó “camino” se agotara completamente antes de pasar a otra.

4.4 Ejemplo de Algoritmo De Búsqueda En Anchura (BFS):

Usar el algoritmo BFS considerando el siguiente grafo no dirigido:



Inicialización:

Seleccionar un nodo inicial: Comenzaremos nuestra búsqueda desde el nodo A.
Estructuras necesarias:
-Una cola para almacenar los nodos que vamos a visitar.
-Un conjunto o lista para marcar los nodos visitados.

Proceso Manual del BFS:

Paso 1: Comenzar en el nodo A.
-Encolamos A y lo marcamos como visitado.

Estado actual:
Cola: [A]
Visitados: {A}

Paso 2: Desencolar A.
Desencolamos A y procesamos sus vecinos:
-Agregamos B, C, y D a la cola y los marcamos como visitados.

Estado actual:
Cola: [B, C, D]

Visitados: {A, B, C, D}

Paso 3: Desencolar B.
Desencolamos B y procesamos sus vecinos:
-A ya está visitado.
-Agregamos E y F a la cola y los marcamos como visitados.

Estado actual:
Cola: [C, D, E, F]
Visitados: {A, B, C, D, E, F}

Paso 4: Desencolar C.
Desencolamos C y procesamos sus vecinos:
-A ya está visitado, no se agrega nada más.

Estado actual:
Cola: [D, E, F]
Visitados: {A, B, C, D, E, F}

Paso 5: Desencolar D.
Desencolamos D y procesamos sus vecinos:
-A ya está visitado.
-Agregamos G a la cola y lo marcamos como visitado.

Estado actual:
Cola: [E, F, G]
Visitados: {A, B, C, D, E, F, G}

Paso 6: Desencolar E.
Desencolamos E y procesamos sus vecinos:
B ya está visitado.

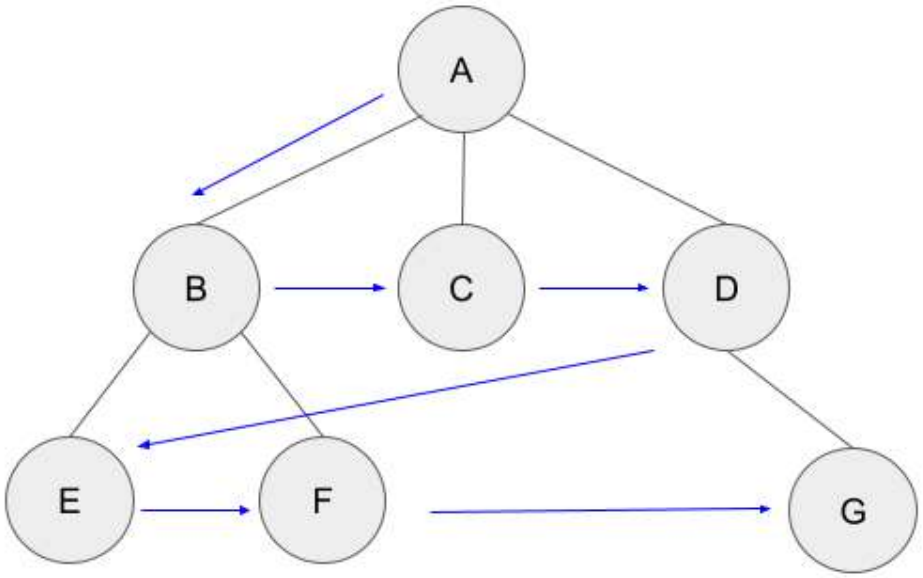
Estado actual:
Cola: [F, G]
Visitados: {A, B, C, D, E, F, G}

Paso 7: Desencolar F.
Desencolamos F y procesamos sus vecinos:
B ya está visitado.

Estado actual:
Cola: [G]
Visitados: {A, B, C, D, E, F, G}

Paso 8: Desencolar G.
Desencolamos G, que no tiene vecinos no visitados.

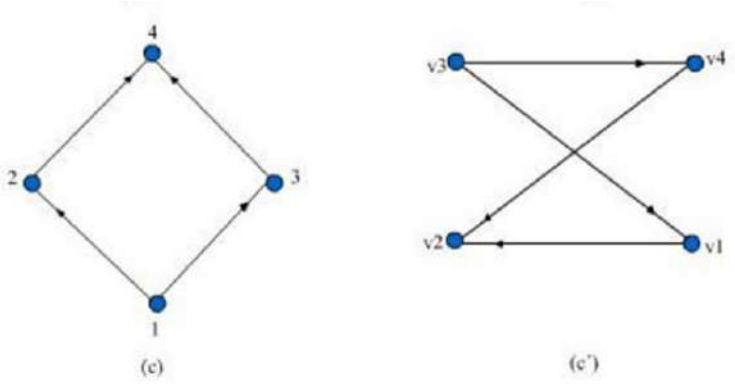
Resultado Final:
Al finalizar el proceso manual de BFS comenzando desde el nodo A:
Se han visitado todos los nodos en el siguiente orden:
A→B→C→D→E→F→G



Resumen del Proceso:

Comienza por marcar el nodo inicial como visitado y encolarlo.
Mientras la cola no esté vacía: Desencola un nodo.
Procesa todos sus vecinos no visitados (marcándolos como visitados y encolándolos).
Repite hasta que todos los nodos alcanzables hayan sido visitados.

4.5 Ejemplo de isomorfismo de grafos:



Para verificar si los dos grafos en la imagen son isomorfos, seguimos estos pasos:

Paso 1: Contar el número de vértices y aristas.
Observamos ambos grafos y verificamos que tengan el mismo número de vértices y aristas.

-Ambos grafos tienen 4 vértices
-Ambos grafos tienen 4 arista

Esto es un primer indicio de que podrían ser isomorfos, pero aun necesitamos revisar que también tienen la misma estructura.

Paso 2: Revisar los grados de los vértices.

Comprobamos que los grados (número de aristas que inciden en cada vértice) de los vértices correspondientes sean iguales.

En el grafo de la izquierda, el grado de cada vértice es:

-Vértice 1: grado 2
-Vértice 2: grado 2
-Vértice 3: grado 2
-Vértice 4: grado 2

En el grafo derecha, el grado de cada vértice es:

-Vértice v1: grado 2
-Vértice v2: grado 2

-Vértice v3: grado 2
-Vértice v4: grado 2

Paso 3: Comprobar la adyacencia entre los vértices.

Para que dos grafos sean isomorfos, debe existir una correspondencia entre los vértices de ambos grafos que conserve la adyacencia(conexiones).

Primero observemos las conexiones (adyacencias) en el primer grafo(el que está en la izquierda):

-Vértice 1 está conectado con los vértices 2 y 3.
-Vértice 2 está conectado con los vértices 1 y 4.
-Vértice 3 está conectado con los vértices 1 y 4.
-Vértice 4 está conectado con los vértices 2 y 3.

Ahora hay que buscar una correspondencia entre los vértices del primer grafo y el segundo que conserve la adyacencia:

-Vértice 1 en el primer grafo se corresponde con v1 en el segundo grafo.
-Vértice 2 en el primer grafo se corresponde con v2 en el segundo grafo.
-Vértice 3 en el primer grafo se corresponde con v3 en el segundo grafo.
-Vértice 4 en el primer grafo se corresponde con v4 en el segundo grafo.

Ahora se comprueba la adyacencia en el segundo grafo(el de la derecha) para esta correspondencia:

-v1 está conectado con v2 y v3, que corresponde a las conexiones de 1 en el primer grafo.
-v2 está conectado con v1 y v4, que corresponde a las conexiones de 2 en el primer grafo.
-v3 está conectado con v1 y v4, que corresponde a las conexiones de 3 en el primer grafo.
-v4 está conectado con v2 y v3, que corresponde a las conexiones de 4 en el primer grafo.

En conclusión, **es un isomorfismo de grafos.**

5. Conclusión

La teoría de gráficas es un campo fundamental en matemáticas y ciencias de la computación que permite modelar y analizar sistemas complejos mediante la representación de elementos y sus relaciones. A través de conceptos básicos como vértices, aristas y distintos tipos de grafos, se facilita el estudio de estructuras interconectadas en áreas tan diversas, desde ingeniería hasta redes de comunicación, biología, economía y logística.

La manipulación de gráficas, incluyendo técnicas como recorridos y algoritmos de optimización, permite resolver problemas prácticos, como la búsqueda de rutas óptimas o la detección de patrones en redes sociales. Además, las representaciones de grafos mediante matrices y listas de adyacencia ofrecen herramientas versátiles para analizar su estructura de manera eficiente según el tipo de las conexiones.

En conclusión, el dominio y conocimiento de la teoría de gráficas y sus métodos de manipulación es crucial para abordar problemas complejos en los que la interrelación entre elementos desempeña un rol fundamental.

6. Bibliografía

- Solano, J. (s.f.). *Algoritmos de grafos*. Universidad Nacional Autónoma de México, Facultad de Ingeniería, Departamento de Computación.

- Claverol, M., Simó, E., & Zaragoza, M. (s.f.). Teoría de grafos. Departamento de Matemática Aplicada IV, EPSEVG - UPC.

- Universidad Tecnológica Latinoamericana en Línea (UTEL). (s.f.). Teoría de gráficas. Recuperado de https://apps.utel.edu.mx/recursos/files/r161r/w24810w/teoria_de_graficas.pdf

- Caballero Palomino, M. Á., Migallón Gomis, V., & Penadés Martínez, J. (s.f.). Prácticas de matemática discreta con MaGraDa. Publicaciones de la Universidad de Alicante.

- Salas, S., & Godino, J. D. (2016). Potencial educativo de la aritmética mapuche en Chile. En A. M. Rosas Mendoza (Ed.), Avances en matemática educativa. Tecnología y matemáticas (pp. 72-84). Editorial Lectorum.

- Cánovas Peña, J. S. (s.f.). Teoría de grafos. Departamento de Matemática Aplicada y Estadística.

- Pérez, Aguila, Ricardo. Una introducción a las matemáticas discretas y teoría de grafos, El Cid Editor, 2013. ProQuest Ebook Central, https://ebookcentral.proquest.com/lib/bibliodgbsp/detail.action?docID=3213116

- Becerra, J. F. V., & Rangel, R. P. Enseñanza De Las Matemáticas Discretas Utilizando Software Libre. AVANCES EN MATEMÁTICA EDUCATIVA TECNOLOGÍA Y MATEMÁTICAS NO., 8.

- Redondo González, M. E. (2023). Matemática Discreta. Teoría de grafos y Aplicaciones a grupos (Bachelor's thesis).

- Kolman, B., Busby, R. C., & Ross, S. (1997). Estructuras de matemáticas discretas para la computación. Pearson Educación.