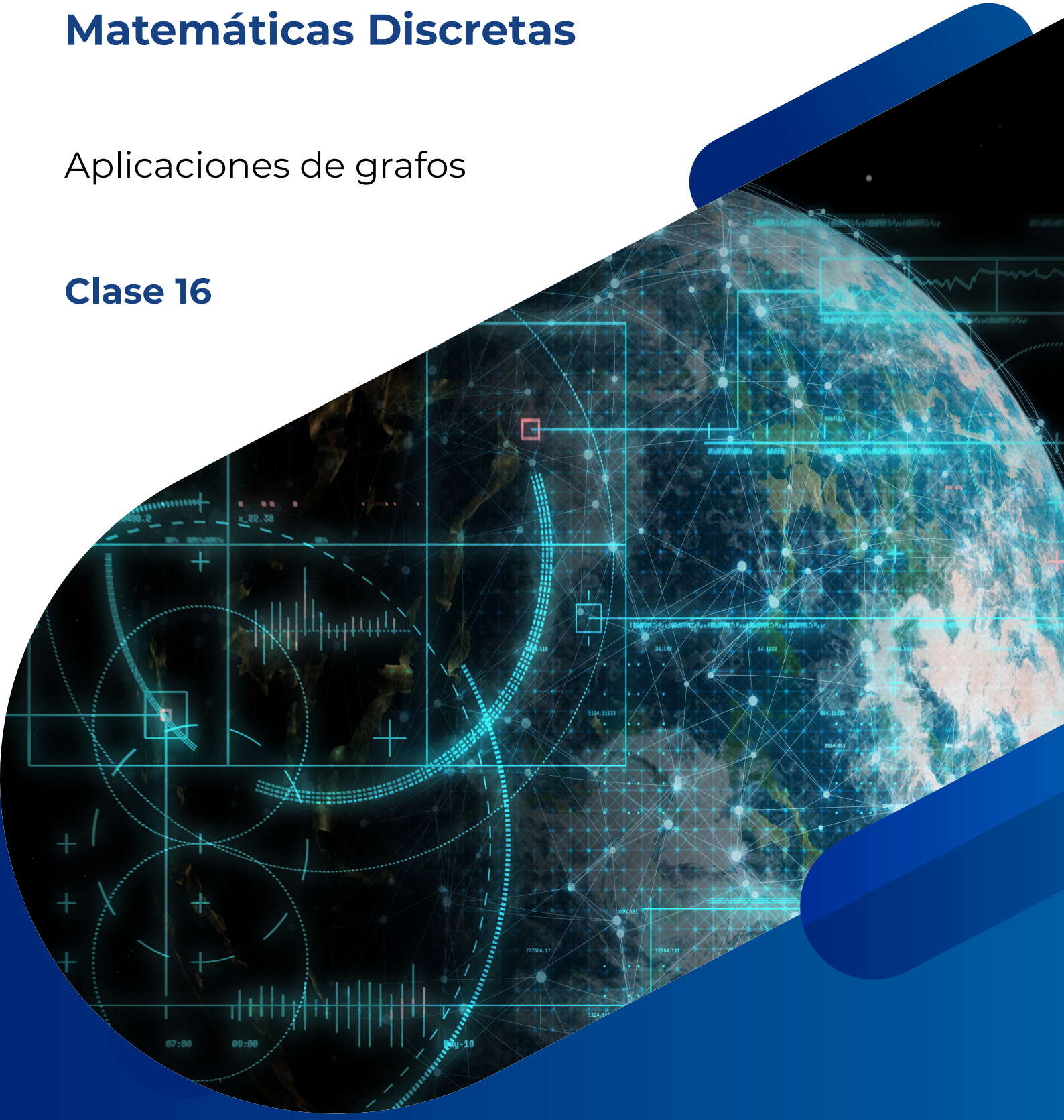
Matemáticas Discretas

Aplicaciones de grafos

Clase 16



La excelencia no se improvisa

Introducción de la clase

A lo largo de este curso hemos visto cómo los grafos nos permiten representar situaciones diversas: desde redes sociales hasta sistemas de transporte. En esta clase daremos un paso más y veremos cómo estas estructuras pueden ayudarnos a resolver problemas concretos que surgen en la vida real. ¿Es posible recorrer todas las calles de una ciudad sin repetir ninguna? ¿Cómo encontrar la mejor ruta para visitar varias ciudades? ¿Cuál es el camino más corto para llegar de un punto a otro en una red?

Nos enfocaremos en tres aplicaciones: los grafos Eulerianos y Hamiltonianos, que nos enseñan a recorrer aristas o vértices de manera óptima; los grafos ponderados, donde cada conexión tiene un costo y buscamos minimizarlo con estrategias como el camino más corto; y finalmente, los árboles, estructuras especiales que aparecen en jerarquías, clasificaciones y sistemas informáticos.

Clase 16: Aplicaciones de grafos

RDA 3: Aplicar los principios y métodos de la Matemática Discreta en una variedad de situaciones, en contextos tanto teóricos como prácticos.

16. Aplicaciones de grafos

16.1. Grafos Eulerianos y Hamiltonianos

En una clase anterior discutimos el famoso problema de los puentes de Königsberg: ¿es posible cruzar todos los puentes de la ciudad sin repetir ninguno y volver al punto de inicio? En esta sección formalizaremos las ideas detrás de ese problema, introduciendo dos tipos de recorridos estudiados en la teoría de grafos: los **Eulerianos** y los **Hamiltonianos**. Ambos conceptos tienen aplicaciones, desde la optimización de rutas en redes hasta el análisis de estructuras moleculares y de circuitos computacionales.

Para comenzar, recordemos algunas definiciones relacionadas con caminos en un grafo (Johnsonbaugh 2018):

- **Camino:** dado un grafo $G = (V, A)$ y dos vértices $a, b \in V$, un camino entre a y b es una secuencia

$$C = (v_0, e_1, v_1, e_2, v_2, \dots, e_{n-1}, v_{n-1}, e_n, v_n)$$

donde $v_0 = a$, $v_n = b$ y cada arista e_k conecta v_{k-1} con v_k.

- **Camino simple:** un camino en el que todos los vértices son distintos.
- **Recorrido:** un camino en el que todas las aristas son distintas.
- **Circuito:** un camino cerrado, es decir, que comienza y termina en el mismo vértice.
- **Ciclo:** un camino cerrado en el que todos los vértices son distintos, salvo el primero y el último, que coinciden.

Con estos conceptos, el problema de los puentes de Königsberg se puede reformular así: dado un grafo donde los vértices representan zonas de la ciudad y las aristas representan puentes, ¿existe un camino cerrado que recorra cada arista exactamente una vez? Esta reformulación nos permite pensar el problema en términos abstractos, y nos lleva a definir una clase especial de recorridos en grafos, conocidos como *circuitos eulerianos*.

DEFINICIÓN 1: Circuito euleriano, Grafo euleriano

Sea G un grafo. Un **circuito euleriano** es un circuito simple que contiene todas las aristas de G. Si G contiene un circuito euleriano, se dice que es un **grafo euleriano**.

Una pregunta natural es: ¿cómo podemos determinar si un grafo es o no euleriano? La siguiente caracterización proporciona una respuesta clara.

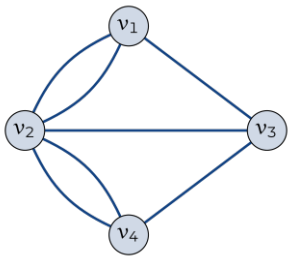
TEOREMA 1.

Sea G un grafo conexo. G es euleriano si y solo si todos sus vértices tienen grado par.

Ejemplo 1. *Consideremos el grafo que representa los puentes de Königsberg. Cada vértice representa una porción de tierra, y cada arista representa un puente.*

Figura 1

Grafo de los puentes de Königsberg



Nota: creación propia (Merino, A., 2025).

Analicemos el grado de cada vértice:

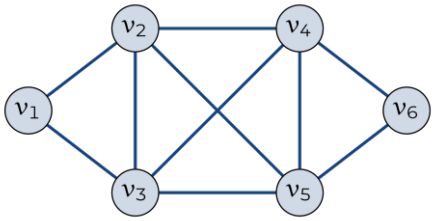
$$\text{grad}(v_1) = 3, \quad \text{grad}(v_2) = 5, \quad \text{grad}(v_3) = 3, \quad \text{grad}(v_4) = 3.$$

Todos los vértices tienen grado impar. Por lo tanto, no se cumple la condición del teorema anterior para la existencia de un recorrido euleriano. En consecuencia, no es posible recorrer todos los puentes una sola vez sin repetir y volver al punto de partida.

Ejemplo 2. *Consideremos el siguiente grafo, ¿es posible dibujarlo «sin levantar la mano»?*

Figura 2

Ejemplo de grafo euleriano



Nota: creación propia (Merino, A., 2025).

Para responder si eso es posible, analicemos los grados de cada vértice:

$$\text{grad}(v_1) = 2, \quad \text{grad}(v_2) = 4, \quad \text{grad}(v_3) = 4, \quad \text{grad}(v_4) = 4,$$

$$\text{grad}(v_5) = 4, \quad \text{grad}(v_6) = 2.$$

Todos los vértices tienen grado par. Por lo tanto, según el teorema de Euler, este grafo es euleriano: existe un recorrido cerrado que utiliza cada arista exactamente una vez. Esto significa que sí es posible dibujar toda la figura sin levantar el lápiz ni pasar dos veces por la misma línea, y además, regresar al punto de inicio.

Te invitamos a tomar lápiz y papel e intentar construir este recorrido. ¿En qué vértice comenzarías? ¿Puedes encontrar más de una forma de hacerlo?

En algunos casos, puede no ser posible regresar al punto de inicio, pero sí recorrer todas las aristas exactamente una vez. Esto motiva una definición más general: **recorrido euleriano** el cual es un recorrido que utiliza todas las aristas del grafo, aunque no necesariamente forma un circuito.

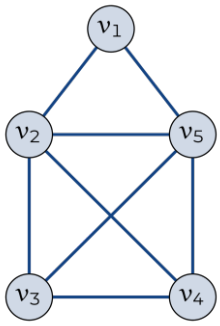
Cuando un grafo tiene exactamente dos vértices de grado impar, se garantiza la existencia de un recorrido euleriano que empieza en uno de ellos y termina en el otro.

TEOREMA 2.

Sea G un grafo conexo. Si G tiene exactamente dos vértices de grado impar, entonces existe un recorrido euleriano que inicia en uno de esos vértices y termina en el otro.

Ejemplo 3. *Consideremos el siguiente grafo, ¿es posible dibujarlo «sin levantar la mano»?*

Figura 3. *Ejemplo de grafo con recorrido euleriano*



Nota: creación propia (Merino, A., 2025).

Al analizar el grado de cada vértice, obtenemos:

$$\text{grad}(v_1) = 2, \quad \text{grad}(v_2) = 4, \quad \text{grad}(v_3) = 3, \quad \text{grad}(v_4) = 3, \quad \text{grad}(v_5) = 4.$$

Notamos que los vértices v_3 y v_4 tienen grado impar, mientras que los demás tienen grado par. Esto implica que el grafo no es euleriano, ya que no todos los vértices tienen grado par. Sin embargo, como tiene exactamente dos vértices de grado impar, el teorema anterior garantiza que existe un recorrido euleriano: se puede recorrer cada

arista exactamente una vez, comenzando en uno de los vértices de grado impar (v_3 o v_4) y terminando en el otro.

Por tanto, sí es posible dibujar toda la figura sin levantar el lápiz ni repetir líneas, siempre que comencemos en v_2 y terminemos en v_4, o viceversa. Te invitamos a intentarlo con lápiz y papel ¿Puedes encontrar el recorrido?

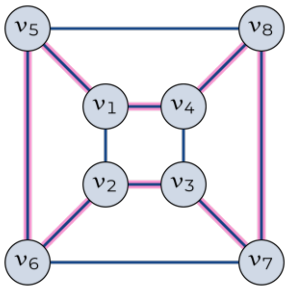
Ahora cambiemos el enfoque. En lugar de querer usar todas las aristas, imaginemos que queremos visitar todos los *vértices* de un grafo una única vez. Este tipo de problema da lugar a otra noción importante: los grafos hamiltonianos.

DEFINICIÓN 2: Circuito hamiltoniano, Grafo hamiltoniano

Sea G un grafo. Un **circuito hamiltoniano** es un camino cerrado simple que contiene todos los vértices de G. Si G contiene un circuito hamiltoniano, se dice que es un **grafo hamiltoniano**.

Ejemplo 4. *Observemos el siguiente grafo:*

Figura 4. *Ejemplo de grafo hamiltoniano*



Nota: creación propia (Merino, A., 2025).

El recorrido resaltado en color magenta corresponde a un circuito hamiltoniano, es decir, un camino cerrado que pasa una sola vez por cada vértice y regresa al punto de partida.

Este recorrido es útil como ejemplo del llamado Problema del agente viajero, en el que se desea visitar cada ciudad exactamente una vez minimizando el recorrido total. Este tipo de grafos se utiliza para planificar rutas óptimas, por ejemplo, en logística, distribución de productos, o recorridos de inspección automatizada por parte de robots.

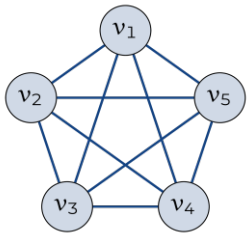
Determinar si un grafo es hamiltoniano es, en general, un problema complejo. Sin embargo, existen condiciones suficientes que permiten garantizar que un grafo es hamiltoniano. La siguiente es una de ellas:

TEOREMA 3.

Sea $G = (V, A)$ un grafo conexo de n vértices, con $n \geq 3$. Si para todo vértice $u \in V$ se cumple que $\text{grad}(u) \geq n/2$, entonces el grafo es hamiltoniano.

Ejemplo 5. *Considere el grafo completo K_5, en el cual cada vértice tiene grado 4. Como $5/2 = 2.5$, y todos los vértices tienen grado mayor a 2.5, se cumple la condición del teorema. Por tanto, K_5 es hamiltoniano.*

Figura 5. *Grafo K_5*



Nota: creación propia (Merino, A., 2025).

Es necesario comprender la diferencia entre los recorridos eulerianos y los hamiltonianos, ya que, aunque ambos buscan cubrir completamente el grafo, lo hacen de maneras distintas. Mientras uno se enfoca en recorrer todas las *conexiones* (aristas), el otro se centra en visitar todos los *puntos* (vértices) una única vez.

- Un **circuito euleriano** es un recorrido cerrado que utiliza cada arista del grafo exactamente una vez. Puede pasar más de una vez por los mismos vértices, siempre que no repita aristas.
- Un **circuito hamiltoniano** es un recorrido cerrado que pasa exactamente una vez por cada vértice del grafo. Las aristas pueden repetirse únicamente si es necesario para cerrar el ciclo.

Estos conceptos tienen múltiples aplicaciones prácticas:

- **Problema del cartero chino:** Un cartero debe recorrer todas las calles (aristas) de su zona sin repetir ninguna, regresando al punto de inicio. Este es un problema euleriano.
- **Ruta de entrega de paquetes:** Un repartidor debe visitar cada punto de entrega (vértice) una sola vez, minimizando el trayecto. Es un problema hamiltoniano.
- **Diseño de circuitos electrónicos:** El diseño eficiente de rutas de conexión puede modelarse mediante recorridos eulerianos.
- **Secuenciación del genoma (bioinformática):** La reconstrucción de secuencias de ADN se puede modelar con recorridos eulerianos sobre grafos de De Bruijn.
- **Optimización de recorridos en robótica:** En tareas donde un robot debe cubrir todas las áreas o estaciones sin repetir, los recorridos hamiltonianos resultan útiles.

Para profundizar en estos conceptos y explorar más ejemplos interactivos sobre recorridos y circuitos en grafos, puedes consultar el siguiente enlace.

- **Título del enlace:** Euler Trails and Circuits
- **Descripción del enlace:** Recurso interactivo que explica las condiciones para que un grafo tenga un recorrido o circuito euleriano.

Enlace: https://discrete.openmathbooks.org/dmoi4/sec_gt-paths.html

16.2. Grafos ponderados: el camino más corto

En muchas aplicaciones reales no basta con saber si hay un camino entre dos puntos; también queremos saber cuál es el más eficiente. Por ejemplo, un sistema de navegación no solo busca una ruta entre dos ciudades, sino la más corta, rápida o económica. Este tipo de problemas se modela con **grafos ponderados**, donde a cada arista se le asigna un peso que representa distancia, tiempo o costo, como lo vimos en una clase anterior.

La longitud de una trayectoria en un grafo ponderado es la suma de los pesos de las aristas que lo componen. Si $w(i,j)$ es el peso de la arista que conecta i y j, entonces la longitud del camino es la suma total de los $w(i,j)$ correspondientes.

Un problema común es encontrar la **ruta más corta** entre dos vértices. Para resolverlo, uno de los algoritmos más famosos fue propuesto por el informático holandés Edsger W. Dijkstra (1930–2002) (Epp 2020). Dijkstra fue pionero en concebir la programación como una ciencia formal.

A continuación, presentamos el algoritmo de Dijkstra para encontrar la longitud de la ruta más corta entre dos vértices a y z en un grafo ponderado conexo, donde todos los pesos son positivos.

Algoritmo 1

Algoritmo de Dijkstra para encontrar la ruta más corta

Entrada: Un grafo ponderado conexo $G = (V, E)$ con pesos positivos $w(i, j)$, vértices a (origen) y z (destino)
Salida : La distancia más corta $L(z)$ desde a hasta z

```
para cada v ∈ V hacer
    L(v) ← ∞ ;                    // Inicializar todas las etiquetas en infinito
L(a) ← 0 ;                        // La distancia desde a a sí mismo es cero
T ← V ;                           // Inicializar conjunto de vértices temporales
mientras z ∈ T hacer
    Seleccionar v ∈ T con L(v) mínimo ;
    T ← T \ {v} ;                                  // Marcar v como permanente
    para cada x ∈ T tal que (v, x) ∈ E hacer
        si L(v) + w(v, x) < L(x) entonces
            L(x) ← L(v) + w(v, x) ;    // Actualizar etiqueta si encontramos mejor
             ruta
devolver L(z)
```

Nota: adaptado de Johnsonbaugh (2018).

Una vez que el vértice z recibe una etiqueta permanente, el valor $L(z)$ corresponde a la longitud de la ruta más corta desde a hasta z.

Para ver cómo se implementa este algoritmo paso a paso y observar su funcionamiento visualmente, puedes consultar el siguiente enlace:

- **Título del enlace:** Cómo funciona el algoritmo de Dijkstra
- **Descripción del enlace:** Explicación visual de cómo funciona el algoritmo de Dijkstra para encontrar caminos más cortos en grafos ponderados.

Enlace: https://youtu.be/EFg3u_E6eHU?si=ovHdpOqIHQ0R184L

16.3. Introducción a árboles

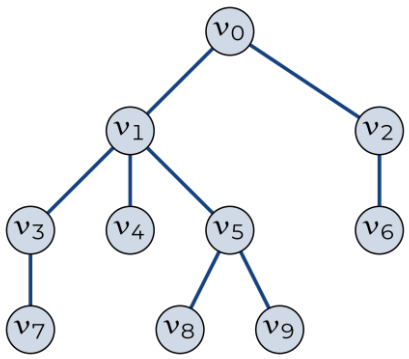
A lo largo de esta unidad hemos trabajado con grafos en general: estructuras que pueden tener ciclos, pesos, o distintas formas de conexión. En esta sección nos enfocaremos en un tipo especial de grafo que tiene una estructura jerárquica y es ampliamente utilizado tanto en teoría como en aplicaciones prácticas: los árboles.

DEFINICIÓN 3. Árbol

Sea T un grafo. Se dice que T es un **árbol** si es conexo y no posee ciclos (es decir, no contiene caminos cerrados).

Ejemplo 6. *Consideremos el siguiente grafo:*

Figura 6. *Ejemplo de árbol*



Nota: creación propia (Merino, A., 2025).

Este grafo es un árbol: es conexo y no tiene ciclos. Se puede recorrer desde cualquier vértice alcanzando todos los demás sin repetir aristas ni formar bucles. Este tipo de estructuras es común en jerarquías, estructuras organizacionales y sistemas de archivos.

A partir de esta representación, podemos introducir algunas definiciones útiles cuando trabajamos con árboles:

- **Raíz:** El vértice desde el cual se organiza el árbol. En este ejemplo, v_0 es la raíz.
- **Nivel:** La distancia (en número de aristas) desde la raíz hasta un vértice. Por ejemplo, v_0 está en el nivel 0, v_1 en el nivel 1, v_3 y v_5 en el nivel 2, y v_7 y v_8 en el nivel 3.

- **Padre:** Un vértice u es padre de v si hay una arista entre ellos y u está un nivel más arriba en la jerarquía. Por ejemplo, v_1 es padre de v_3.
- **Hijo:** Un vértice v es hijo de u si u es su padre. Por ejemplo, v_4 y v_5 son hijos de v_1.
- **Hoja:** Un vértice sin hijos. En este árbol, los vértices v_4, v_6, v_7, v_8 y v_9 son hojas.

Los árboles tienen varias propiedades importantes. Por ejemplo, entre cualquier par de vértices existe un único camino. Además, si un árbol tiene n vértices, entonces contiene exactamente $n - 1$ aristas.

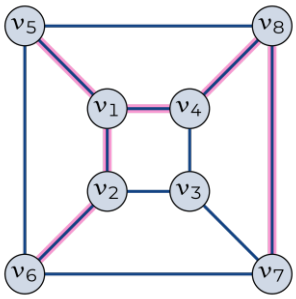
En muchos casos, nos interesa obtener un árbol a partir de un grafo más grande. Esto nos lleva a las siguientes definiciones:

DEFINICIÓN 4.

Sea G un grafo y T un subgrafo de G. Se dice que T es un **árbol de expansión** de G si T es un árbol y contiene todos los vértices de G.

Ejemplo 7. *Observemos el siguiente grafo:*

Figura 7. *Árbol de expansión resaltado en color*



Nota: creación propia (Merino, A., 2025).

En este grafo, las aristas coloreadas en magenta forman un árbol de expansión. Esto significa que dichas aristas: conectan todos los vértices del grafo original, no contienen ciclos, forman una estructura conectada mínima.

Este árbol de expansión es un subgrafo del grafo original, que conserva todos los vértices, pero solo parte de las aristas. Si asignáramos pesos a las aristas, podríamos

buscar el árbol de expansión que tenga la menor suma total de pesos (lo que se conoce como árbol de expansión mínima), pero en este caso nos limitamos a ilustrar una expansión posible sin considerar los pesos.

16.3.1. Generación de árboles de expansión

Para generar un árbol de expansión a partir de un grafo, podemos utilizar algoritmos de búsqueda sistemática. En esta sección exploraremos dos enfoques clásicos: la búsqueda en anchura (BFS) y la búsqueda en profundidad (DFS). Ambos algoritmos permiten recorrer todos los vértices del grafo a partir de un nodo inicial, registrando las aristas utilizadas para conectar vértices nuevos. Estas aristas forman un árbol de expansión del grafo.

Árbol de expansión con búsqueda en anchura (BFS):

El siguiente algoritmo recorre el grafo utilizando una cola y construye un árbol de expansión añadiendo una arista cada vez que se visita un nuevo vértice.

Algoritmo 2

Árbol de expansión usando BFS

Entrada: Un grafo conexo $G = (V, A)$ y un vértice inicial $v \in V$
Salida : Un árbol de expansión (M, T) con raíz en v

Función `ArbolExpansionBFS(G, v)`:
- $Q \leftarrow (v)$; `// Cola de vértices por visitar`
- $M \leftarrow \{v\}$; `// Conjunto de vértices visitados`
- $T \leftarrow \varnothing$; `// Aristas del árbol`
- **mientras** $Q \neq \varnothing$ **hacer**
  - $u \leftarrow$ `Ultimo(Q)` ; `// Siguiente vértice`
  - $Q \leftarrow$ `SinUltimo(Q)`;
  - **para cada** $w \in V[u]$ **hacer**
    - **si** $w \notin M$ **entonces**
      - $M \leftarrow M \cup \{w\}$;
      - $T \leftarrow T \cup \{\{u, w\}\}$;
      - $Q \leftarrow$ `Concatenar(w,Q)`;
- **devolver** (M, T)

Nota: creación propia (Merino, A., 2025).

Código 1. *Árbol de expansion usando BFS*

```
def arbol_expansion_bfs(G, v):
    Q = [v]
    M = {v}
```

```
T = set()
while Q != []:
    u = Q.pop()
    for w in G[u]:
        if w not in M:
            M.add(w)
            T.add((u, w))
            Q.insert(0, w)
return M, T
```

Nota: creación propia (Merino, A., 2025).

Árbol de expansión con búsqueda en profundidad (DFS):

La búsqueda en profundidad utiliza una pila y explora lo más profundo posible antes de retroceder. El siguiente algoritmo produce un árbol de expansión marcando las aristas a medida que descubre nuevos vértices.

Algoritmo 3

Árbol de expansión usando DFS

Entrada: Un grafo conexo $G = (V, A)$ y un vértice inicial $v \in V$
Salida : Un árbol de expansión (M, T) con raíz en v

Función `ArbolExpansionDFS(G, v)`**:**

```
Q ← (v);                                  // Pila de vértices por visitar
M ← {v};                                  // Vértices visitados
T ← ∅;                                    // Aristas del árbol
mientras Q ≠ ∅ hacer
    u ← Ultimo(Q);
    si V[u] ⊆ M entonces
        Q ← SinUltimo(Q);
    en otro caso
        para cada w ∈ V[u] hacer
            si w ∉ M entonces
                M ← M ∪ {w};
                T ← T ∪ {{u, w}};
                Q ← Concatenar(Q,w);
                salir;
devolver (M, T)
```

Nota: creación propia (Merino, A., 2025).

Código 2

Árbol de expansion usando DFS

```
def arbol_expansion_dfs(G, v):
    Q = [v]
    M = {v}
    T = set()
    while Q != []:
        u = Q[-1]
        if all(w in M for w in G[u]):
            Q.pop()
        else:
            for w in G[u]:
                if w not in M:
                    M.add(w)
                    T.add((u, w))
                    Q.append(w)
                    break
    return M, T
```

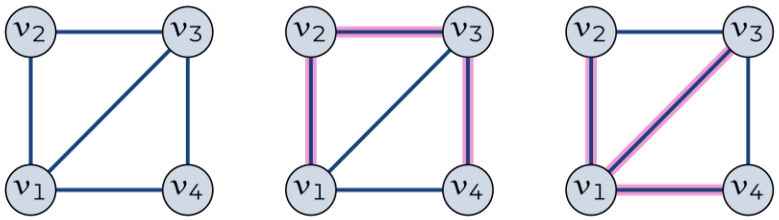
Nota: creación propia (Merino, A., 2025).

Ejemplo 8. *La siguiente figura muestra un grafo original (izquierda) y dos posibles árboles de expansión (centro y derecha) construidos a partir de él mediante diferentes estrategias de recorrido:*

- ***Izquierda:*** *grafo original.*
- ***Centro:*** *árbol de expansión generado por búsqueda en profundidad (DFS).*
- ***Derecha:*** *árbol de expansión generado por búsqueda en anchura (BFS).*

Figura 8

Árboles de expansión obtenidos por DFS y BFS



Nota: creación propia (Merino, A., 2025).

*En el panel central se muestra el árbol de expansión obtenido mediante **búsqueda en profundidad (DFS)**. Este método tiende a explorar caminos lo más profundo posible antes de retroceder, por lo que suele generar árboles más «largos».*

*En el panel derecho se presenta el árbol de expansión generado mediante **búsqueda en anchura (BFS)**, que explora primero todos los vecinos inmediatos del vértice inicial antes de avanzar. Esto suele producir árboles «más planos».*

Referencias

Epp, S. S. (2020). Discrete Mathematics with Applications (5.a ed.). Cengage Learning.

Johnsonbaugh, R. (2018). Discrete Mathematics (8.a ed.). Pearson Educación.

Términos

- **Camino:** Secuencia de vértices conectados por aristas, usada para ir de un punto a otro en un grafo.
- **Árbol:** Grafo conectado que no tiene ciclos.

Profundización

Recurso 1

- **Título del recurso:** Algoritmos sobre grafos en Python
- **Descripción del enlace:** Este recurso presenta una guía de Python para los principales algoritmos de grafos.
- **Enlace:** https://colab.research.google.com/github/andres-merino/MatematicasDiscretas-Virtual-05-N0279/blob/main/pages/clases/clase16/AlgoritmosGrafos.ipynb

PUCE Virtual

La excelencia no se improvisa

síguenos