Algorítmica

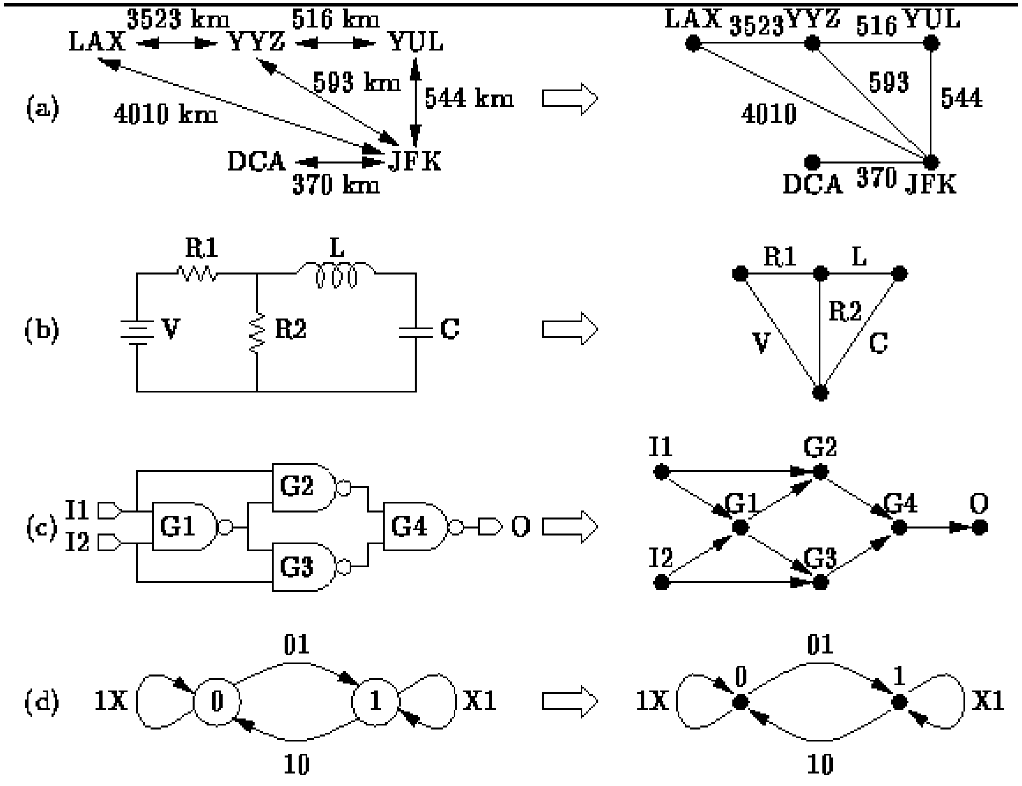
Algoritmos Greedy

Repaso de grafos

Algoritmos Greedy para Grafos

- *¿ Por qué hay que estudiar grafos?*
- *Podemos abstraer grafos de distintas situaciones físicas del mundo real para:*
  - *Resolver problemas de recorridos dando servicios eficientes a nuestros clientes o a los usuarios de los sistemas:*
  - *Por ejemplo el Problema del Viajante de Comercio*
  - *Diseñar Redes poco costosas de computadores de telefonía, etc.*
- *Son básicos en Inteligencia Artificial, pero también en Arquitectura y en otras muchas Ingenierías*
- *Desde luego en Robótica*

Algoritmos Greedy para Grafos



a) *Problema del viajante de comercio*

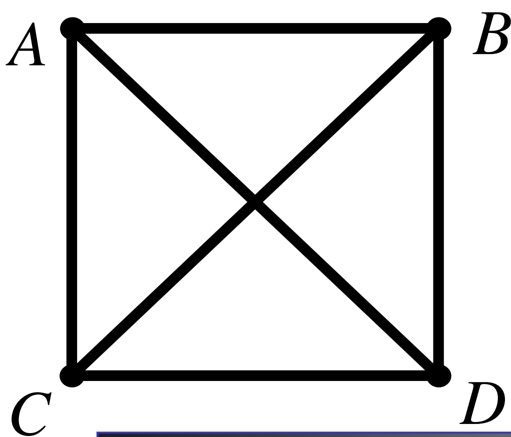
b) *Un circuito eléctrico: los puntos indican donde se conectan las componentes, que son las aristas*

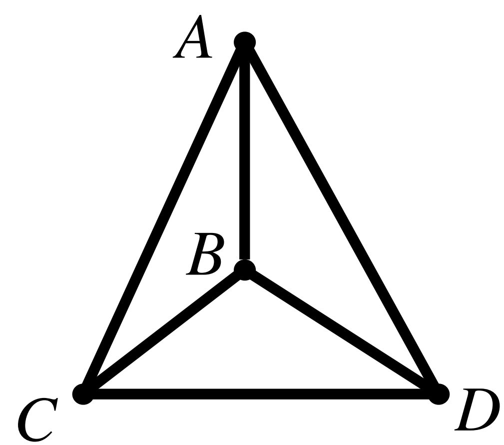
c) *Un circuito lógico: los nodos son puertas lógicas y los arcos marcan flujos*

d) *Una maquina de estado finito: los nodos son los estados y los arcos las transiciones posibles*

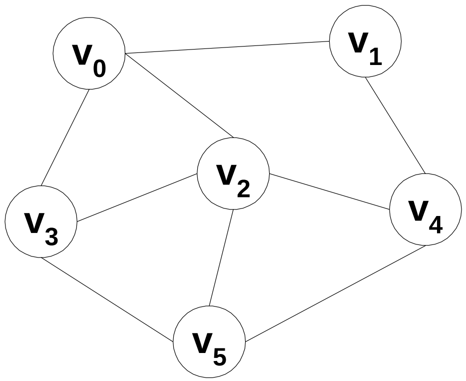
Nociones básicas de grafos

- *Un grafo se define con dos conjuntos:*
  - *Un conjunto de* **vertices** *(nodos), y*
  - *Un conjunto de* **aristas**
- *Cuando las aristas tienen origen y final (dirección), se habla de* **Grafos Dirigidos***, y en lugar de aristas tendremos arcos. Tambien hay* **Grafos Ponderados**

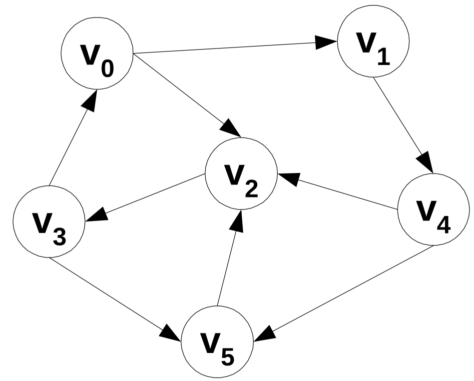




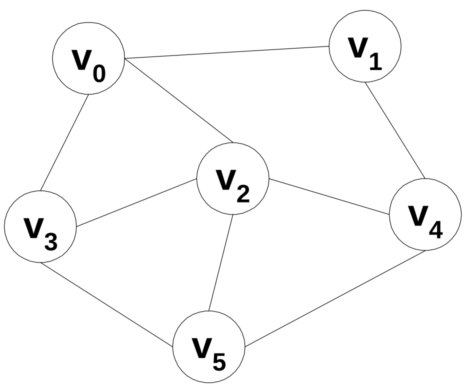
Ejemplos



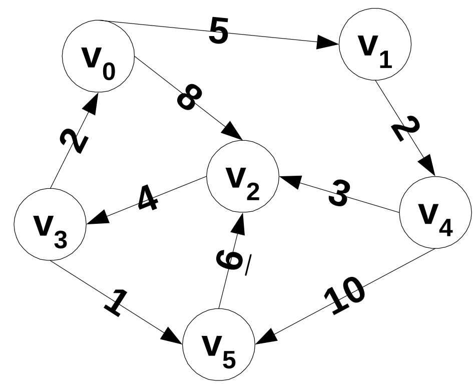
No Dirigido



Dirigido



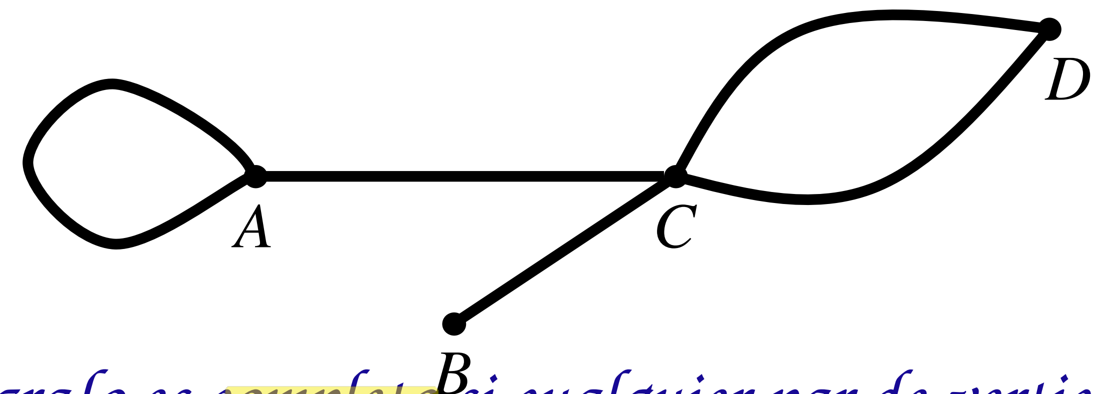
No Ponderado



Ponderado

Nociones básicas de grafos

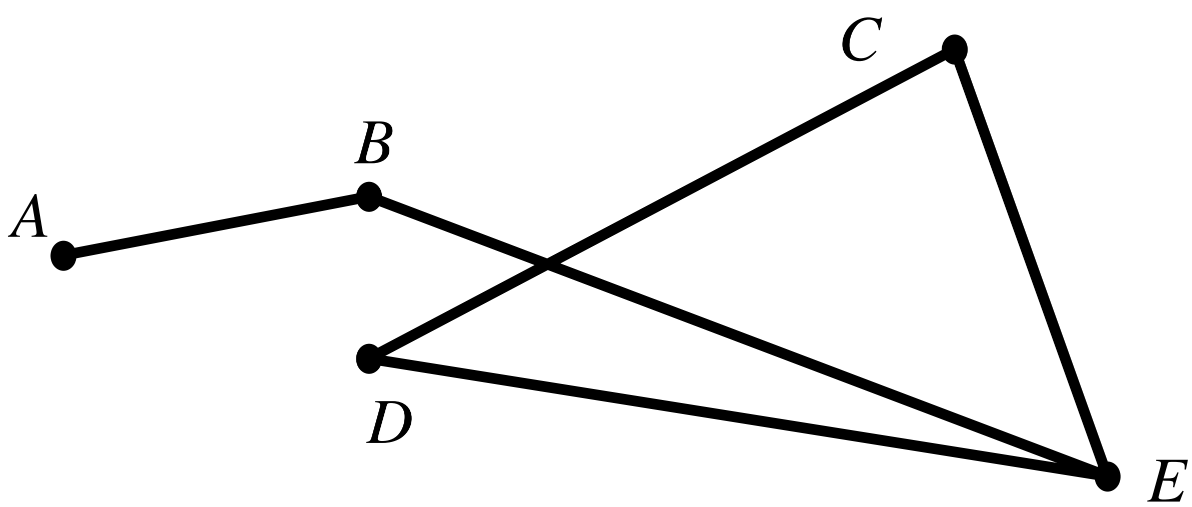
- *Podemos unir dos vertices varias veces, obteniendo multiples aristas*
- *Podemos unir un vertice a si mismo, para formar un lazo*



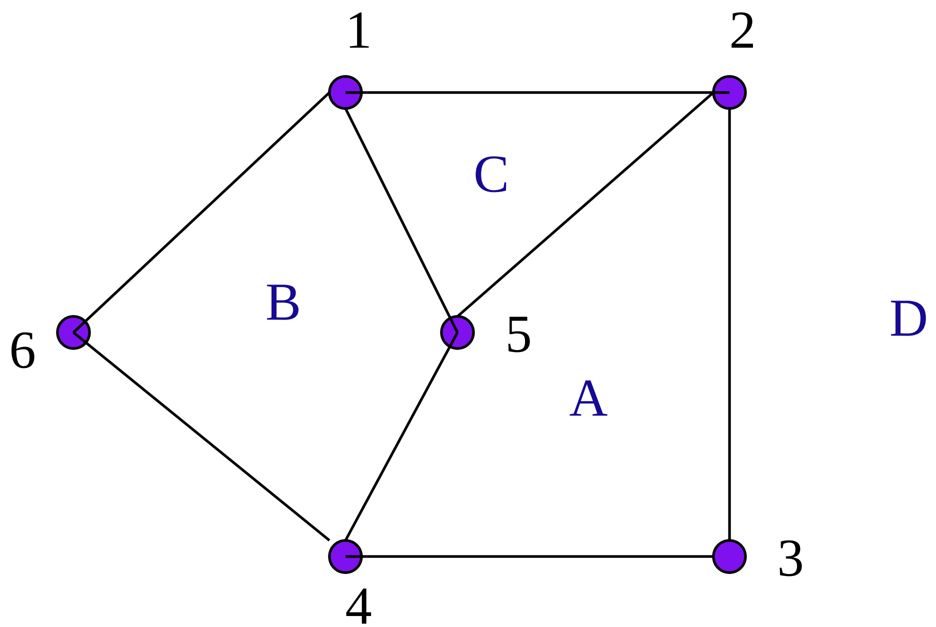
- *Un grafo es completo si cualquier par de vertices distintos esta unido por una arista*

Nociones básicas de grafos

- *Dos vertices son adyacentes si existe una arista que los une (B y E son adyacentes, pero B y D no)*
- *Dos aristas son adyacentes si comparten un vertice (AB y BE son adyacentes, pero las AB y CE no)*



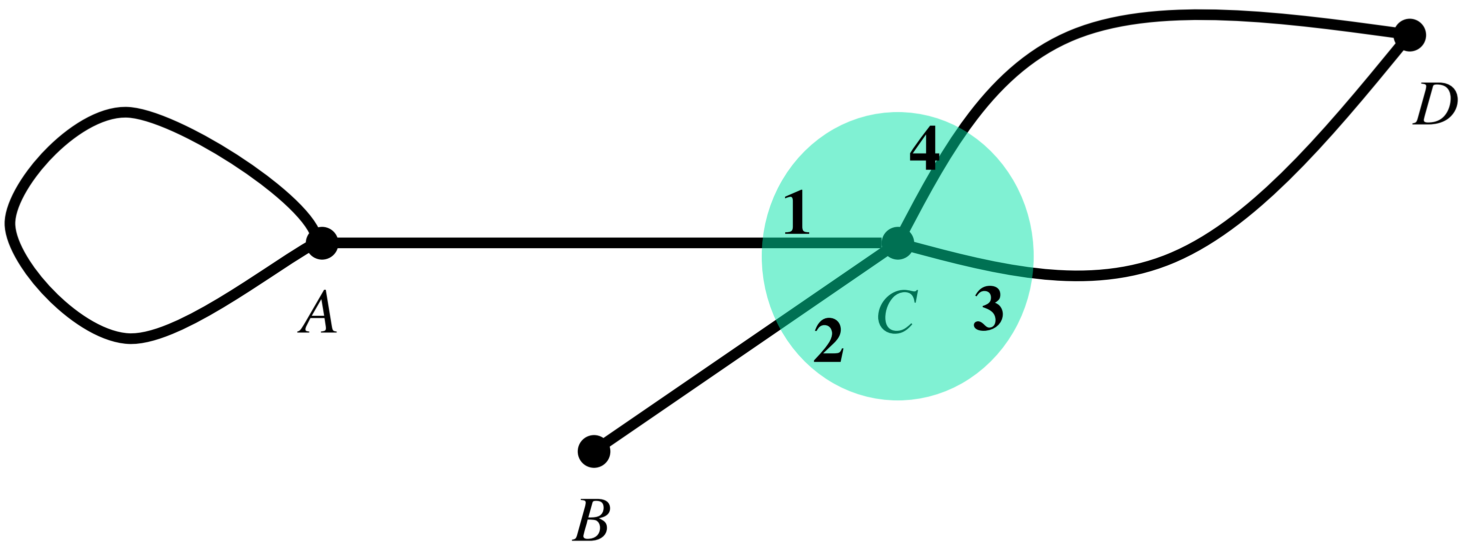
Nociones básicas de grafos



Un grafo se llama plano si puede pintarse en el plano (o en una esfera) sin que se crucen sus aristas.

Nociones básicas de grafos

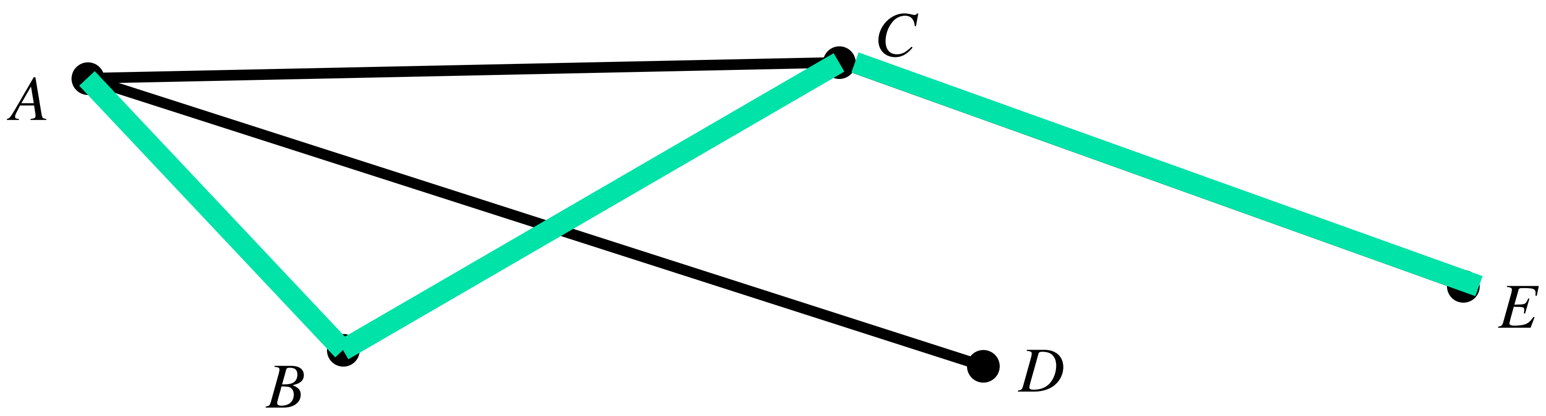
- El grado de un vertice es el numero de aristas que pasan por ese vertice.



Gra(C) = 4, Gra(A) = 3, Gra(B) = 1, Gra(D) = 2

Nociones básicas de grafos

- Un **camino** es una sucesion de aristas distintas adyacentes.
  - ¡No se permite repetir aristas en un camino!

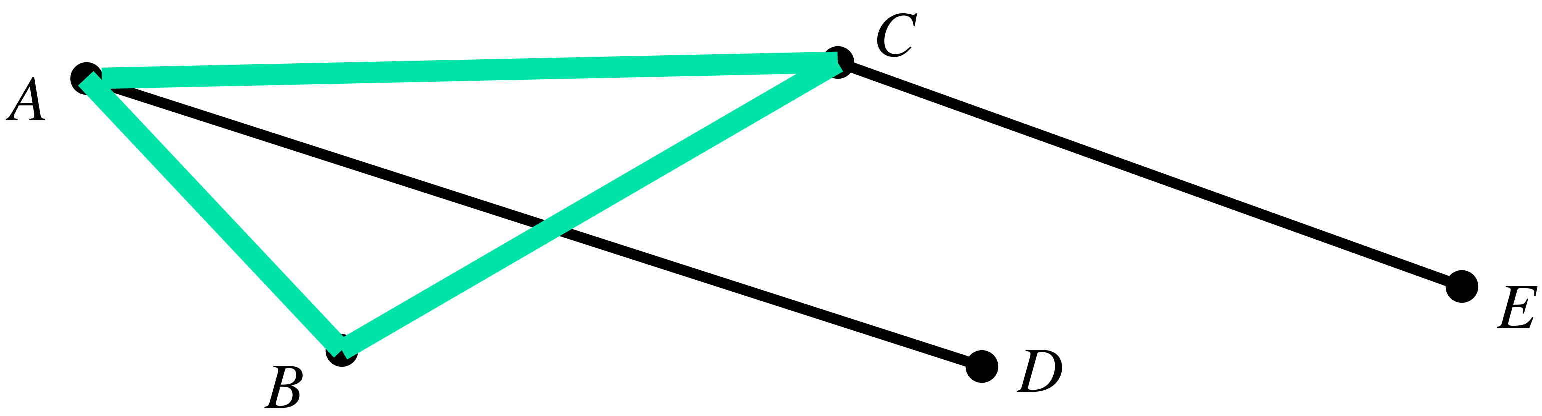


Las aristas AB, BC, y CE forman el camino ***A,B,C,E***

Las aristas AD, DA y AC no forman un camino (repeticion)

Nociones básicas de grafos

- *Un **circuito** es un camino que comienza y termina en el mismo vertice.*
  - *¡No se permite repetir aristas!*

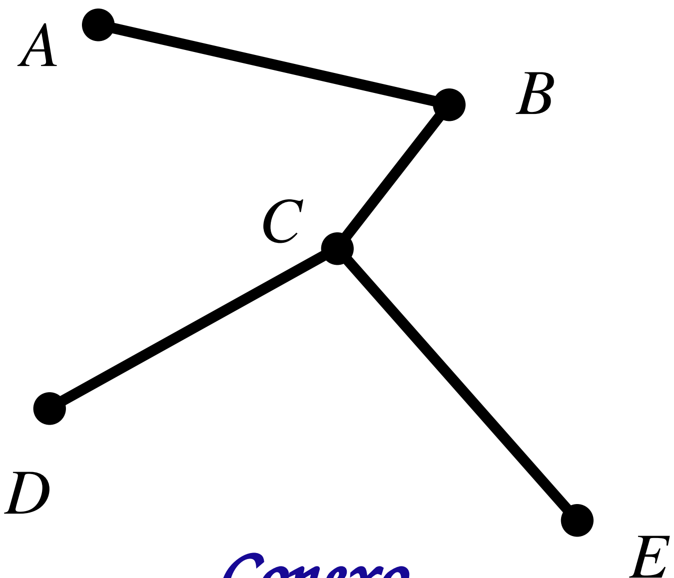


*AB, BC, and CA forman el circuito **A,B,C,A***

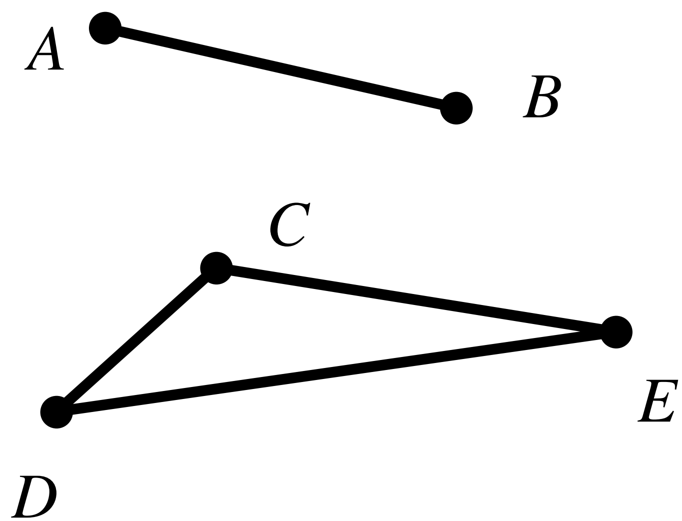
Las aristas BA y AD no forman un circuito

Nociones básicas de grafos

- *Un grafo se dice* ***conexo*** *si cualesquiera dos vertices pueden unirse por un camino. Si un grafo no es conexo, se llama* ***inconexo***



Conexo



Inconexo

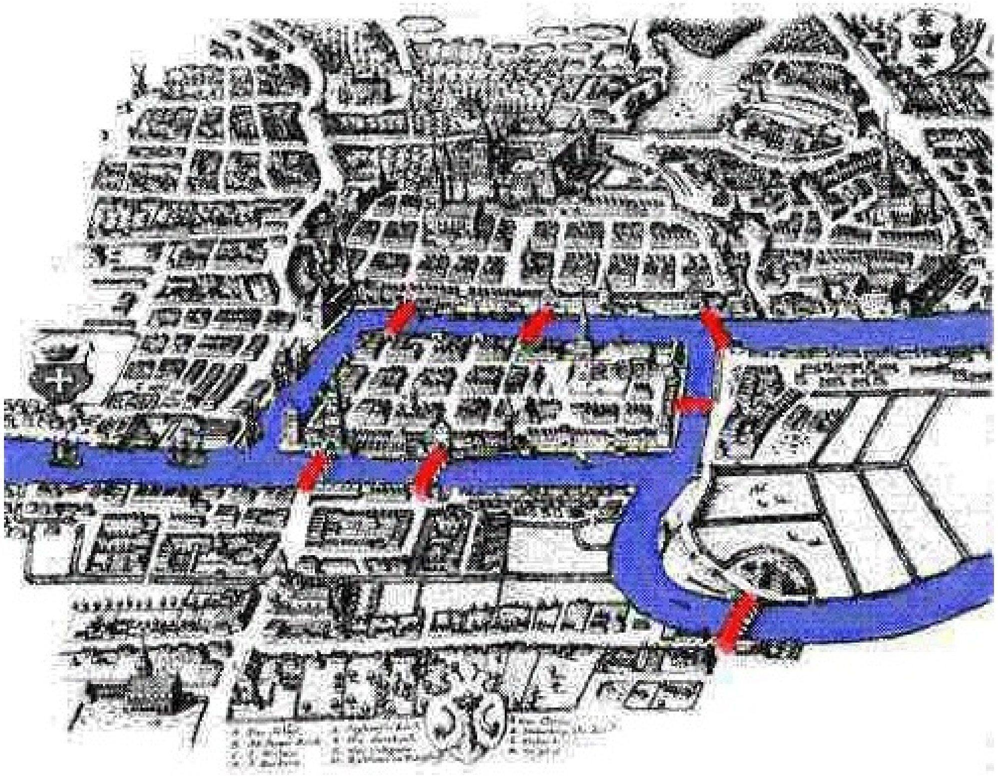
Nociones básicas de grafos

Leonhard Euler *(1707-1783) ha sido el matemático mas prolifico de todos los tiempos, con contribuciones importantes en Geometria, Calculo, Fisica, ... y Grafos.*

Aproximadamente la mitad de sus publicaciones las escribio despues de quedar ciego. Cuando perdió la vista comentó:

"Asi ahora me distraere menos."

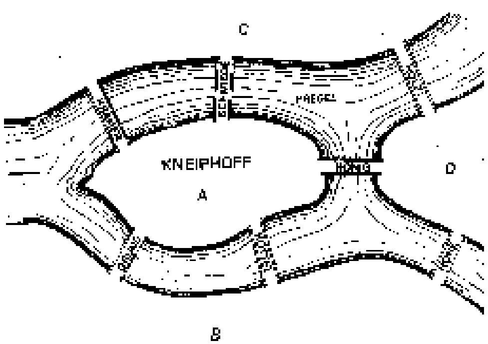
Nociones básicas de grafos

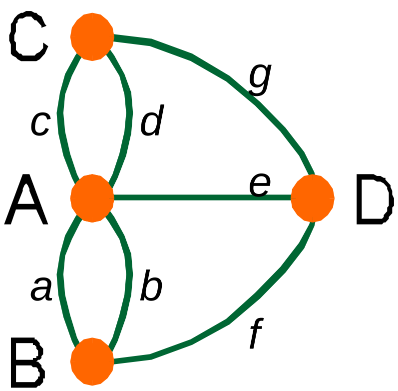


En la ciudad de ***Königsberg*** *en Austria, hay una isla llamada* ***"Kneiphoff"*** *bordeada por el rio* ***Pregel****. Hay* ***siete puentes*** *conectando las orillas. El problema es saber si una persona puede recorrer todos estos puentes, pasando por todos y cada uno de ellos solamente una vez, y volviendo a su punto de partida*

Nociones básicas de grafos

- *Cuando Euler llego a Königsberg, habia consenso en la imposibilidad de hacer aquel recorrido, pero nadie lo aseguraba con certeza*
- *Euler planteo el problema como uno de grafos:*
  - *Cada parte de tierra supondria un vertice, y*
  - *Cada puente representaria una arista*





Y en 1736 demostró la imposibilidad de dar un paseo como el que se quería

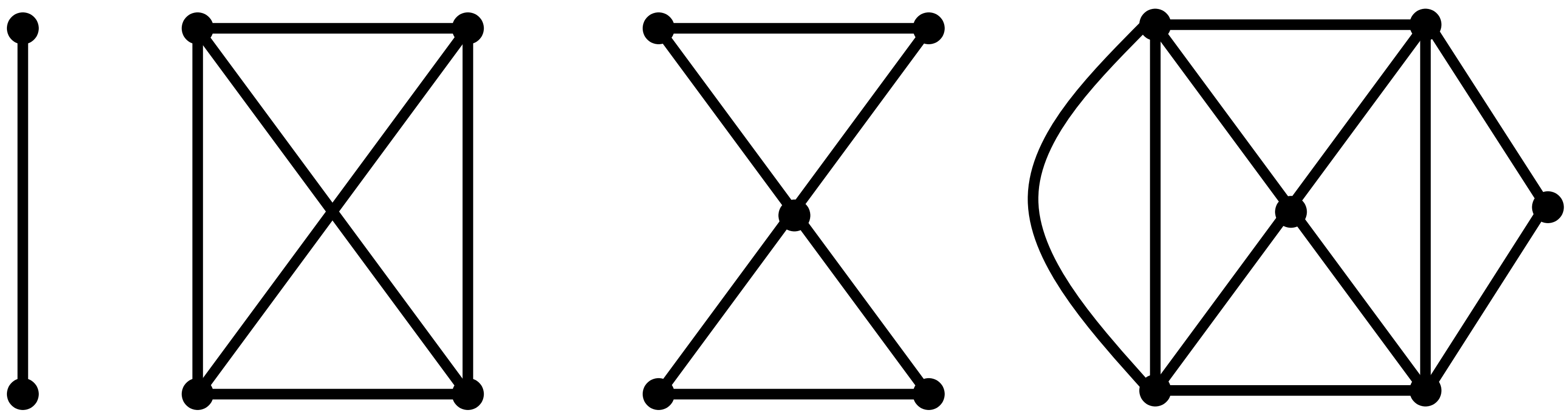
Nociones básicas de grafos

- *Un **Camino Euleriano** es un camino que pasa a traves de cada arista del grafo una y solo una vez*
- *Un **Circuito Euleriano** es un circuito que pasa por cada arista del grafo una y solo una vez*
- ***Teorema de Euler***
  - Si todos los vertices de un grafo son de grado impar, entonces no existen circuitos eulerianos.
  - Si un grafo es conexo y todos sus vertices son de grado par, existe al menos un circuito euleriano.

Nociones básicas de grafos

- Un **Circuito Hamiltoniano** es un circuito que pasa a traves de cada vertice una y solo una vez, y termina en el mismo vertice en el que comenzó.
- Los **Circuitos Hamiltonianos y los Circuitos Eulerianos** son conceptos distintos y separados: En un grafo podemos tener de unos, y no de otros.
- A diferencia de los Circuitos Eulerianos, no tenemos un resultado simple que nos diga si un grafo tiene o no Circuitos Hamiltonianos.

Ejemplos



¿Circuito Euleriano?

no no si si

¿Circuito Hamiltoniano?

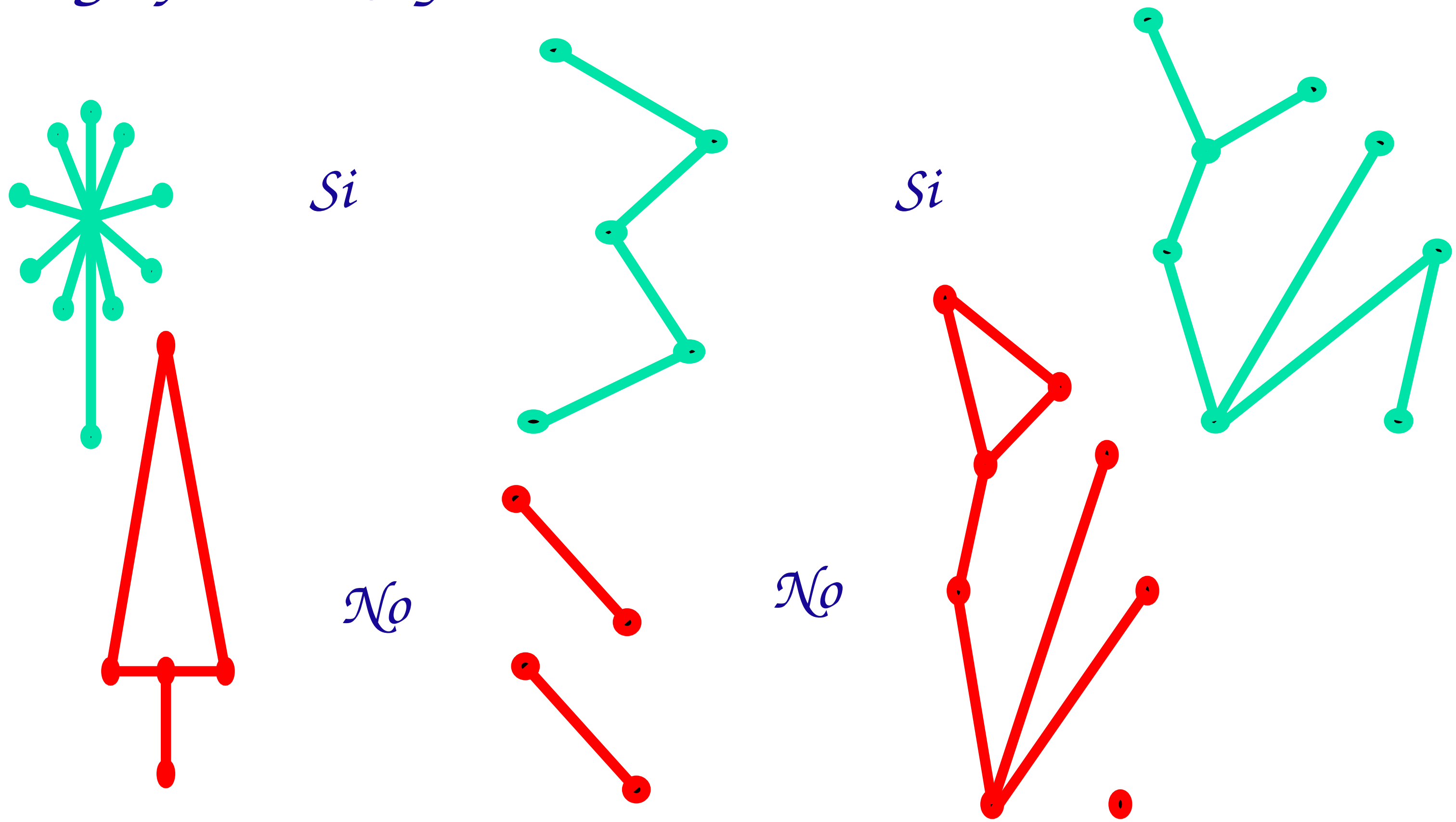
no si no si

Ejemplos

- *La determinacion de un circuito euleriano minimal es lo que se conoce con el nombre del Problema del Cartero Chino:*
  - *Encontrar el circuito de longitud minimal que recorre cada arista de un grafo al menos una vez.*
- *A la busqueda de un circuito hamiltoniano minimal recibe el nombre de Problema del Viajante de Comercio:*
  - *Hallar el circuito de longitud minimal que recorre todos los nodos de un grafo una y solo una vez, comenzando y terminando por el mismo vertice*

Nociones básicas de grafos

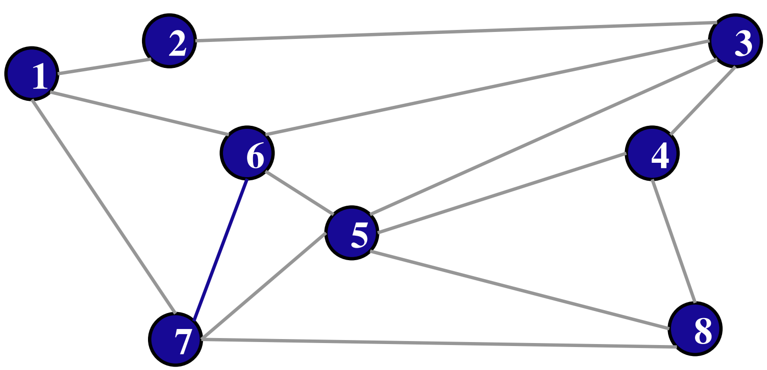
- *Un **arbol** es un grafo que no tiene ciclos.*
- *Un grafo conexo y sin circuitos, se llama un **arbol***



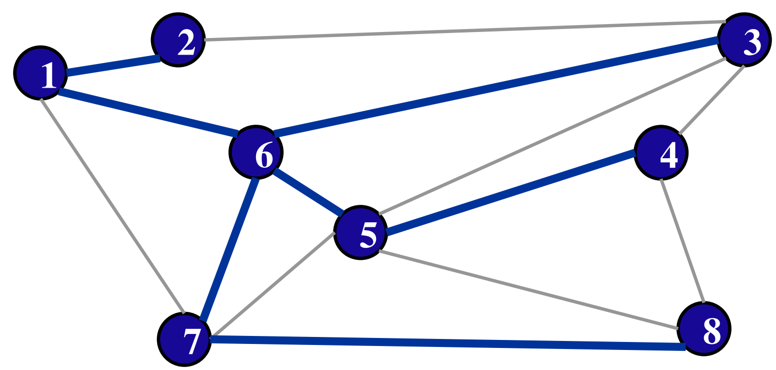
Árbol Generador de un Grafo

- Sea $T = (V, F)$ un subgrafo de $G = (V, E)$.
  - T es un arbol generador de G:
  - T es aciclico y conexo.
  - T es conexo y tiene $|V|$ - 1 arcos.
  - T es aciclico y tiene $|V|$ - 1 arcos.

$G = (V, E)$

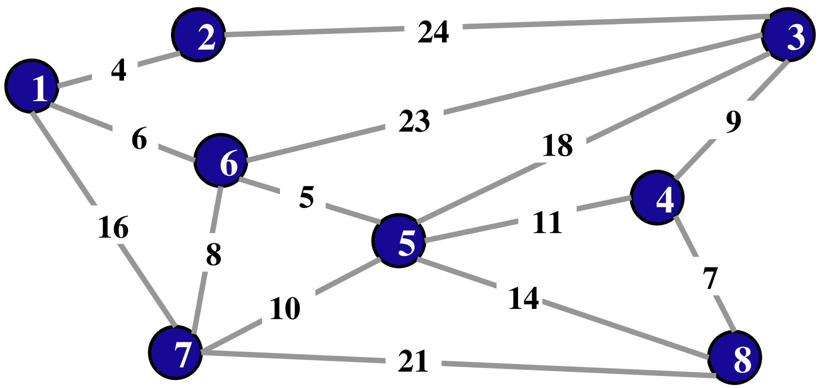


$T = (V, F)$

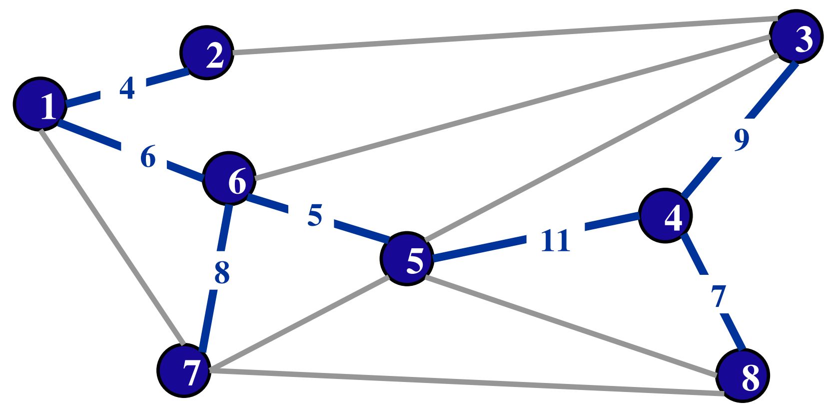


Arbol Generador Minimal

- *Dado un grafo conexo G con pesos en sus arcos c_e, un* ***Arbol Generador Minimal*** *es un arbol generador de G en el que la suma de los pesos de sus arcos es minima.*



G = (V, E)



T = (V, F) *l(T) = 50)*

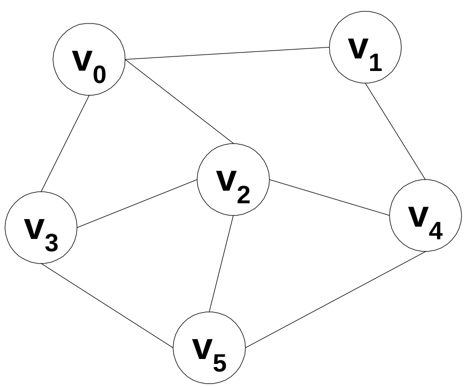
- ***Teorema de Cayley*** *(1889). Hay n^{n-2} arboles generadores de K_n (el grafo completo de n vertices)*
  - *Por tanto el empleo de la fuerza bruta para encontrar el AGM de un grafo no es un metodo recomendable*

La matriz de adyacencia

- *Si suponemos un grafo G = (X, E) con n vértices, entonces su matriz de adyacencia es:*

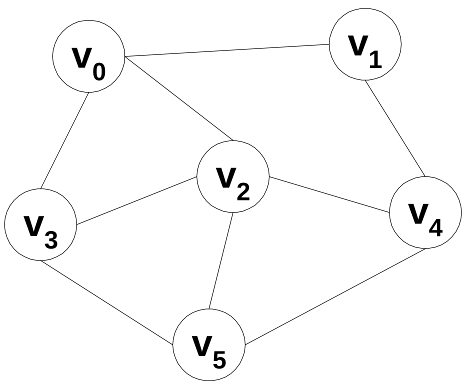
$$A_G(i,j)=\begin{cases}1\ldots\ldots si\left(x_i,x_j\right)\in E\\0\ldots\ldots si\left(x_i,x_j\right)\notin E\end{cases}$$

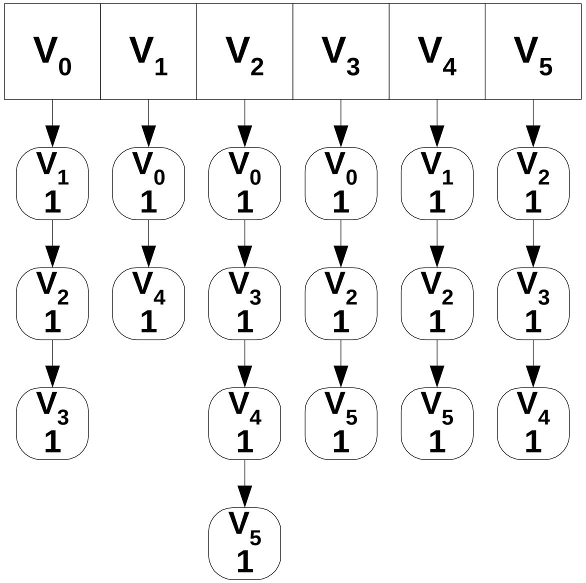
- *Cuando el grafo es ponderado, el valor que aparece en cada casilla es el peso de la arista correspondiente*

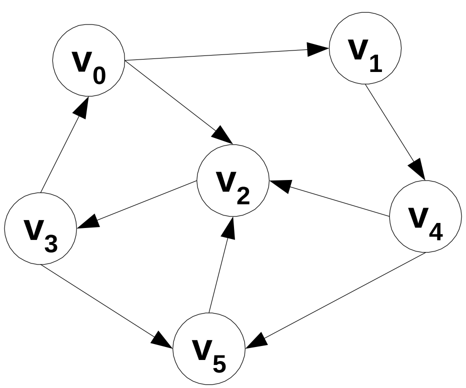


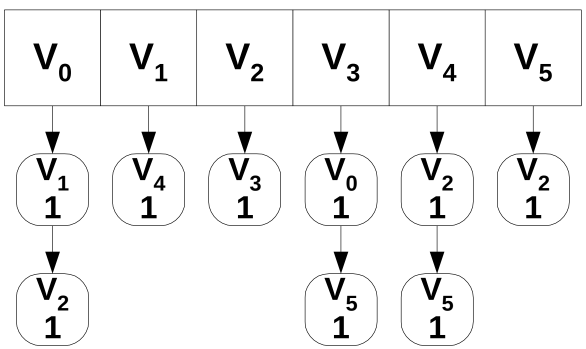
left → right	0	1	2	3	4	5
0	-	1	1	1	-	-
1	1	-	-	-	1	-
2	1	-	-	1	1	1
3	1	-	1	-	-	1
4	-	1	1	-	-	1
5	-	-	1	1	1	-

Representación por listas de adyacencia







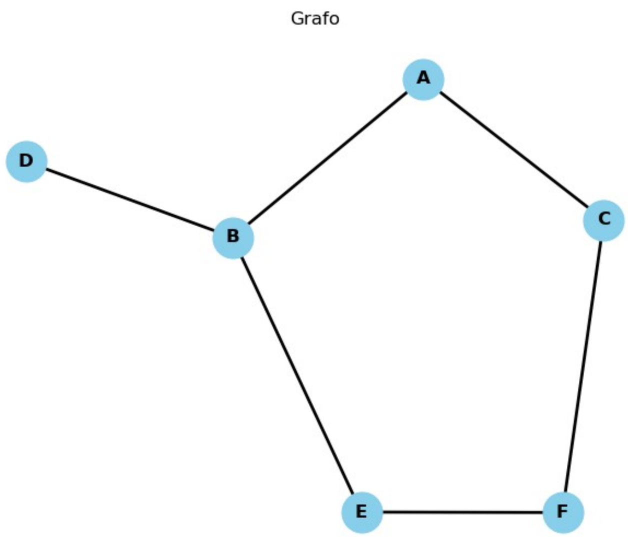


Matriz de Incidencia

- *Una matriz de incidencia es una representación de un grafo que muestra qué vértices están conectados por cada arista. Para un grafo no dirigido, la matriz de incidencia tendrá filas que representan los vértices y columnas que representan las aristas. Cada celda indicará si el vértice correspondiente está presente en la arista correspondiente.*
- *Si se trata de un grafo no dirigido, entonces la matriz es:*

$$B_G(i,j)=\begin{cases} 1.....si...x_i..esta..en..a_j \\ 0.....en..otro..caso \end{cases}$$

donde x_i es un nodo y a_j es una arista.



Matriz de Incidencia:

	AB	AC	BD	BE	CF	DE	EF
A	1	1	0	0	0	0	0
B	1	0	1	1	0	0	0
C	1	0	0	0	1	0	0
D	0	0	1	0	0	1	0
E	0	0	0	1	0	1	1
F	0	1	0	0	1	0	1

Matriz de Incidencia

-
- *Si se trata de un grafo dirigido, entonces la matriz es:*

$$B(i,j)=\begin{cases} +1\ldots\ldots si..x_i..es..inicial..en..a_j \\ 0\ldots\ldots en..otro..caso..(bucle) \\ -1\ldots\ldots si..x_i..es..final..en..a_j \end{cases}$$

donde x_i *es un nodo y* a_j *es un arco.*

Recorridos en grafos

- Los recorridos en grafos son algoritmos que visitan todos los nodos o aristas de un grafo, explorando su estructura.

  Son esenciales para explorar y comprender la estructura y las relaciones en un grafo.

- Aplicaciones
  - Búsqueda de Caminos o Ciclos: Los recorridos en grafos pueden utilizarse para encontrar caminos entre dos nodos o detectar ciclos en el grafo.
  - Componentes Conectados: Pueden utilizarse para determinar si el grafo está completamente conectado o dividido en componentes separados.
  - Ordenamiento Topológico: En el caso de grafos dirigidos acíclicos, un orden topólogico es aquel en el que si existe un camino de A hacia B en el grafo entonces A precede a B
  - Resolución de Problemas de Búsqueda: fundamentales en la resolución de problemas en inteligencia artificial y optimización combinatoria.

Recorridos en grafos

- ***Recorrido en Anchura (BFS - Breadth-First Search)***
  - *Explora todos los vecinos de un nodo antes de avanzar a los vecinos de sus vecinos.*
  - *Utiliza una cola para mantener los nodos que deben ser visitados.*

- ***Recorrido en Profundidad (DFS - Depth-First Search)***
  - *Explora un camino hasta que llega a un punto donde no hay más nodos por explorar.*
  - *Utiliza una pila o recursión para mantener los nodos que deben ser visitados.*

Recorridos en grafos

- ## Recorrido en Anchura (BFS - Breadth-First Search)

```
# Función para el recorrido en anchura (BFS)

 bfs(grafo  G, vertice ini, set<vertice> & visited)

    queue<vertice> queue(inicio)
    visited.add(inicio)

    while !queue.empty(){
        node = queue.pop()
         // ACCIÓN SOBRE CON EL NODO
        print(node)
        for neighbor in G.adjacent(node){
         // si neighbor no está en visited
        if (visited.find(neighbor)==visited.end()) {
                queue.append(neighbor)
                visited.add(neighbor)
            }
        }
    }
}
```

Recorridos en grafos

- ## Recorrido en Profundidad (BFS - Depth-First Search)

```
# Función para el recorrido en profundidad (DFS)

 void dfs(const grafo & G, vertice act,
          set<vertice> & visited){

    visited.add(act)
    // ACCIÓN SOBRE CON EL NODO
        print(ini)
    for neighbor in G.adjacent(act){
         // si neighbor no está en visited
         if (visited.find(neighbor)==visited.end() )
              dfs(G, neighbor, visited)
    }
}
```

Recorridos en grafos

- ***Recorrido en Grafos***
  - *el recorrido visita todos los vértices de una componente conexa*

```
# recorrer todo el grafo

 set<vertices> visitados;

 while (visitados.size()!=G.size()) // todos los vertices visitados
    node = seleccionar vertice no visitado;
    dfs( G, node, visitados)
```