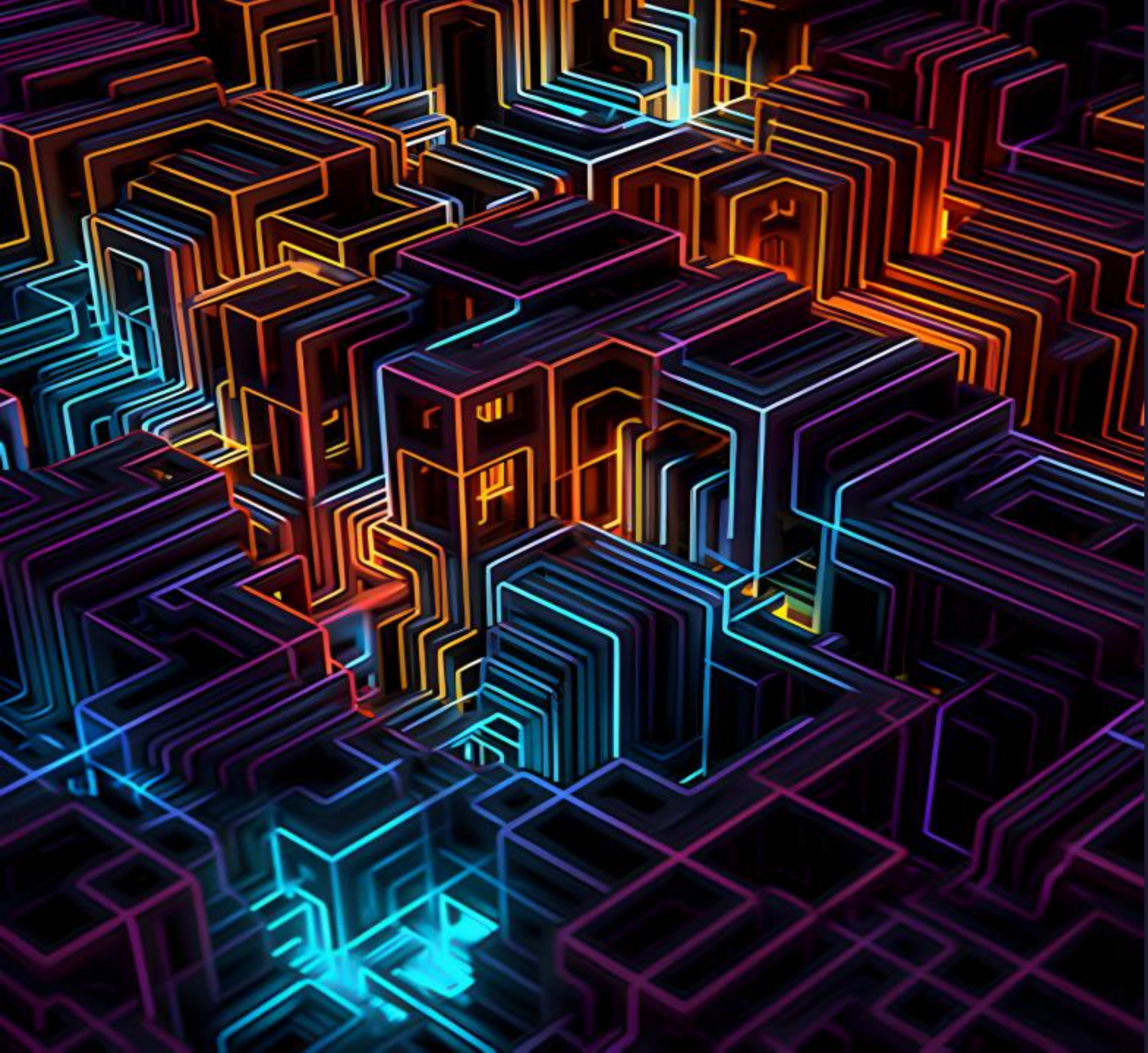


Algoritmos y problemas en grafos

Análisis y diseño de algoritmos avanzados

Dra. Valentina Narváez Terán

Tecnológico de Monterrey

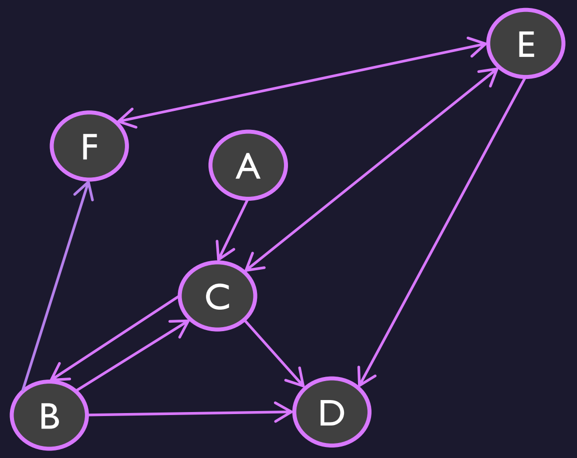
Grafos: repaso de algunas definiciones

Un grafo $G \; = \; V, E$ esta formado por

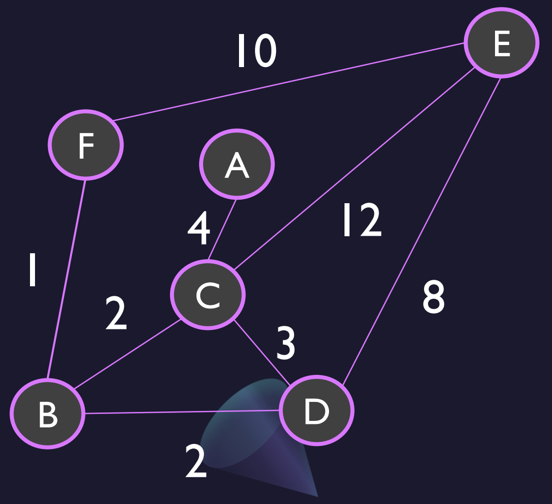
V : un conjunto de nodos

E : un conjunto de aristas que conectan a los nodos

Grafos dirigidos: las aristas tienen dirección. Las aristas (B,C) y (C,D) no son la misma

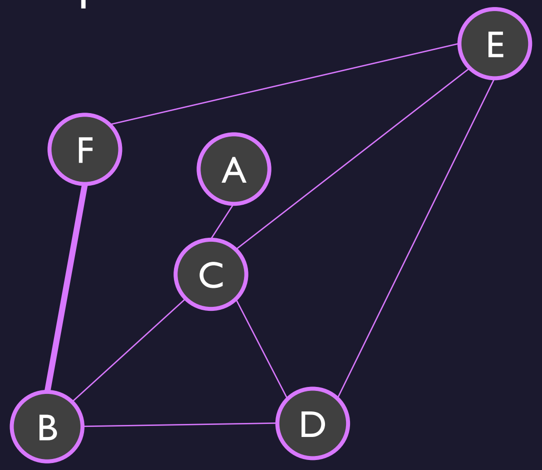


Grafos ponderados: las aristas tienen un peso o costo asociado



Grafo simple: no-dirigido y sin lazos (conexiones de un nodo con si mismo)

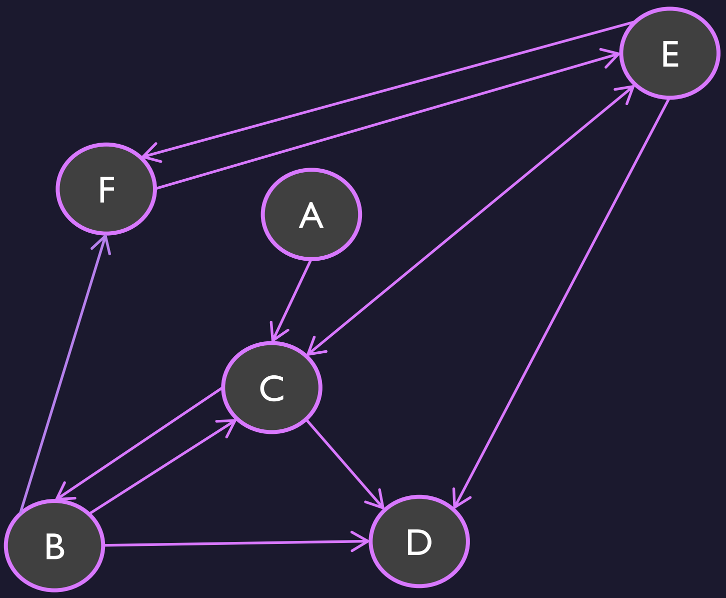
Puede ser ponderado o no



Grafos: representaciones

	A	B	C	D	E	F
A			1			
B			1	1		1
C		1		1		
D						
E				1		1
F					1	

Las principales formas de representar grafos son la matriz de adyacencia



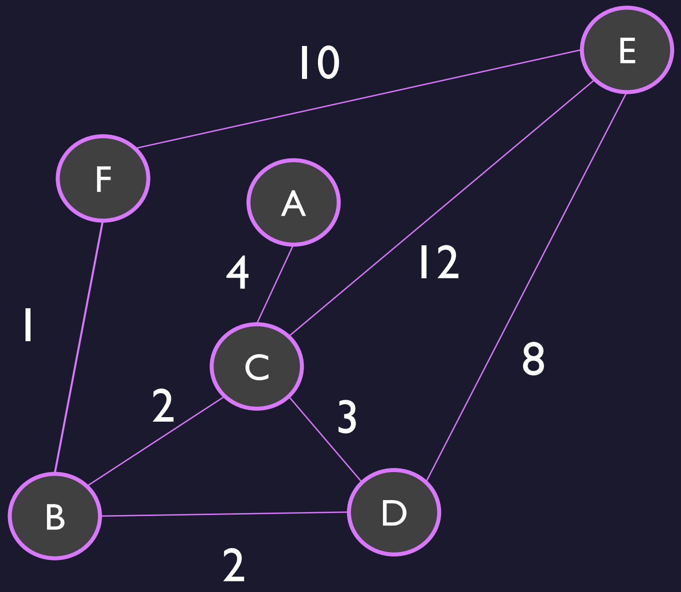
nodo	vecinos
A	$[C]$
B	$[C, D, F]$
C	$[B, D]$
D	$[\ ]$
E	$[D, F]$
F	$[E]$

O la lista de adyacencia (fácil de implementar con diccionarios de Python)

Grafos: representaciones

	A	B	C	D	E	F
A			4			
B			2	2		1
C	4	2		3	12	
D		2	3			
E			12			10
F		1			10	

Para grafos ponderados la matriz guarda pesos

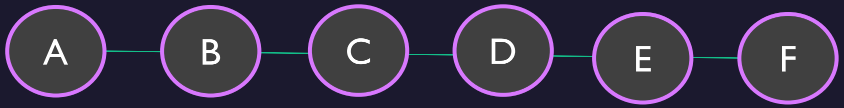


nodo	vecinos
A	$[\,[C,4]\,]$
B	$[\,[C,2],[D,2],[F,1]\,]$
C	$[\,[A,4],[B,2],[D,3],[E,12]\,]$
D	$[\ \ [B,2],[C,3],[E,8]]$
E	$[\,[C,12],[D,8],[F,10]\,]$
F	$[\,[F,1],[E,10]\,]$

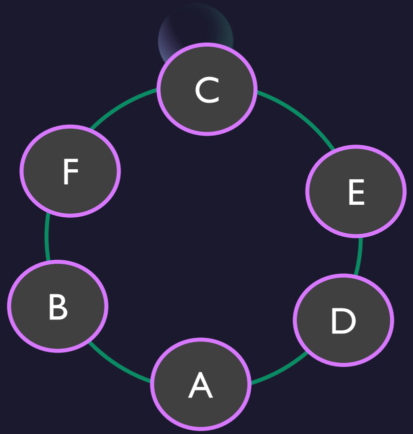
Y una lista de adyacencia en diccionario incluye el vecino y el peso de la arista

Grafos: topologías

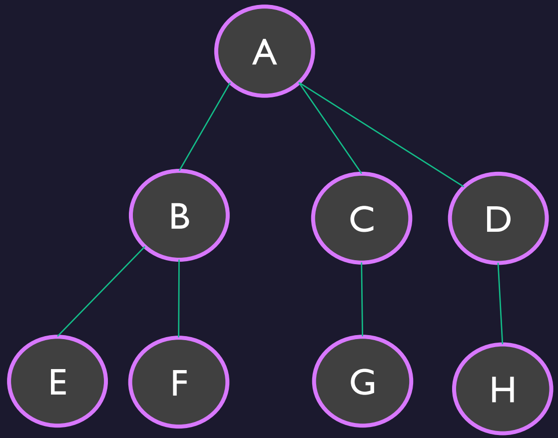
Existen distintos patrones de conexión



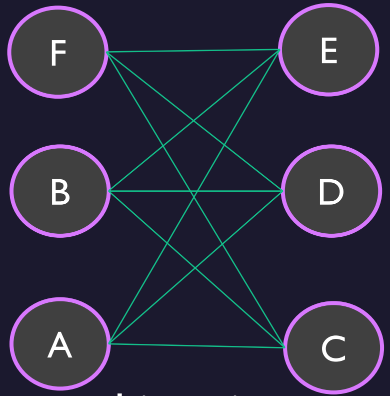
grafo camino (path graph, line graph)



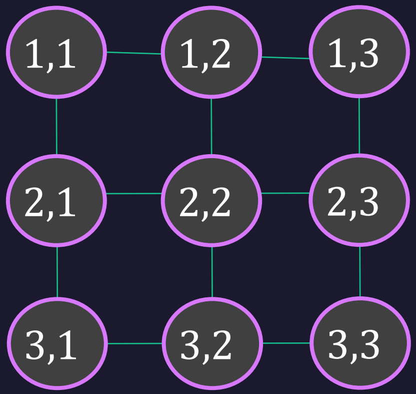
grafo ciclo (cycle graph)



grafo árbol (no tiene ciclos)



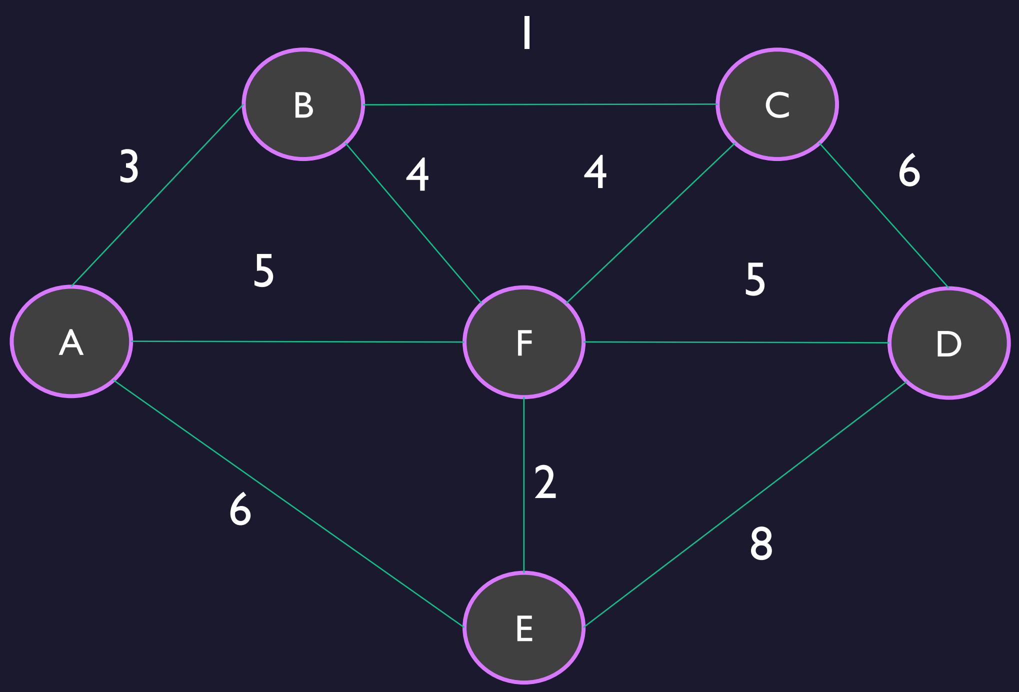
bipartita



malla (mesh graph)

Y muchos otros…

Revisitando el minumum spanning tree



Imagina que este grafo representa una serie de instalaciones. Queremos conectarlas a algún servicio (por ejemplo, fibra óptica)

Los pesos en las aristas representan el costo proyectado para conectar una instalación con otra.

No hay mucho presupuesto, así que debemos crear solo las conexiones mínimamente necesarias, de modo que todo quede conectado y gastemos lo menos posible…

El nodo A ya tiene el servicio, así que será el origen

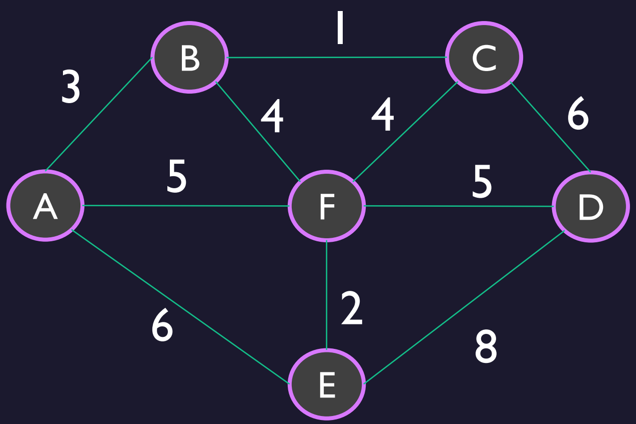
Revisitando el minumum spanning tree

El concepto aplicado en el ejemplo es el minimum spanning tree

Spanning tree: es un árbol conexo con los mismos vértices que un G, pero con solo un subconjunto de sus aristas

El spanning tree mínimo de un grafo ponderado tiene además el menor costo total al sumar todo su subconjunto de aristas

Algoritmo de Prim



	A	B	C	D	E	F
distancia	0					
procesado						
predecesor						

En este ejemplo, el nodo origen es A

$distancias$ = infinito para cada nodo, 0 para el nodo origen
$procesado \ = \ 0$ para cada nodo
$predecesor \ = \ -1$ para cada nodo

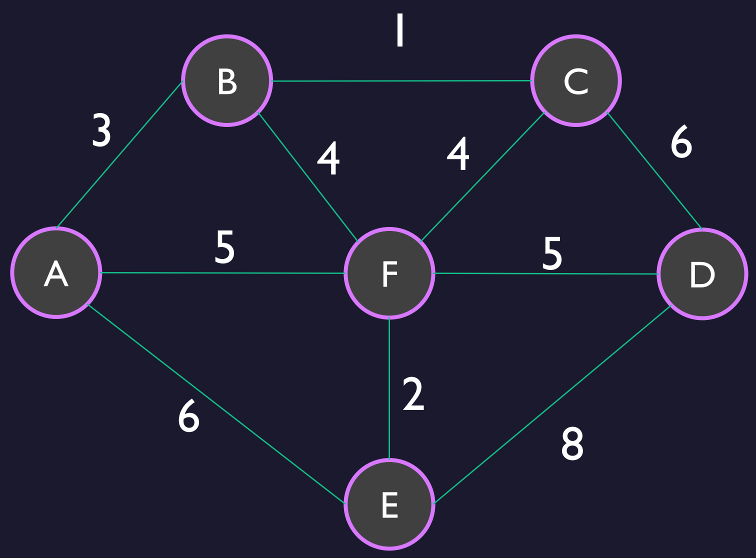
Agregar el nodo origen a una fila q con $distancias[origen]$ como prioridad

Para cada nodo
    u = sale nodo de la fila de prioridad q
    Marca u como procesado, es decir $procesado[u] \ = \ 1$

    Para cada j vecino de u:
        $nueva\_distancia$ = peso de la arista de u a j

        Si $nueva_{distancia} < \ distancia$
            $distancia[j] \ = \ nueva\_distancia$
            $predecesor[j] \ = \ u$
            Agregar j a la fila, con prioridad $distancia[j]$ si j no ha sido procesado

Algoritmo de Kruskal



Ordena las aristas por su costo

T = grafo vacío
$seleccionadas = 0$

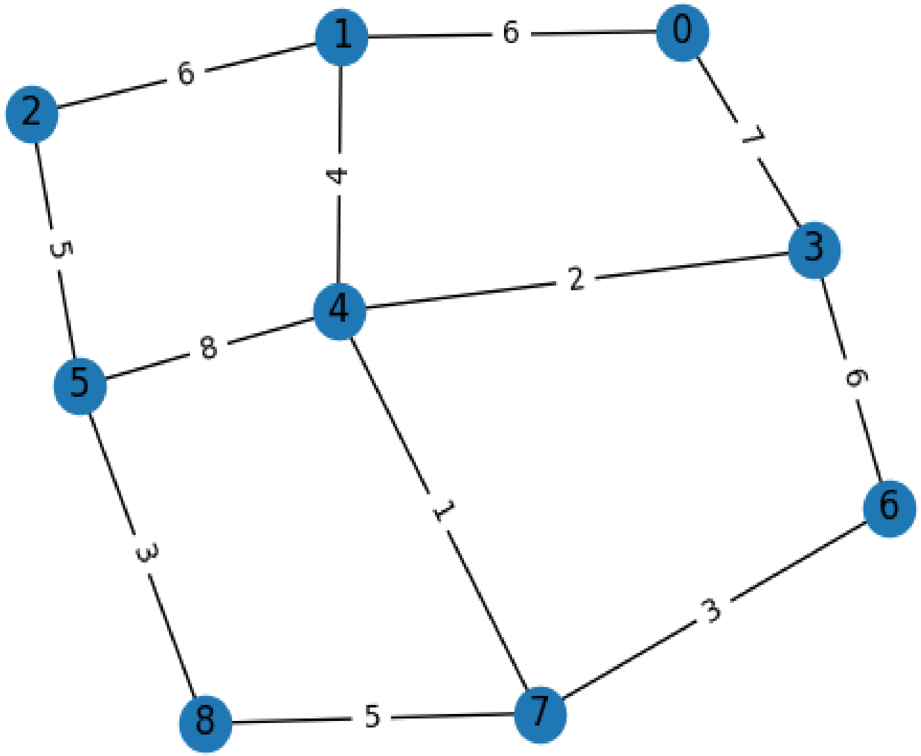
Mientras $seleccionadas < n - 1$
  $minima$ = la arista menos costosa disponible
  Agrega $minima$ a T
  Si T tiene un ciclo
    Remueve mínima de T
  Si no
    $seleccionadas++$

Otra alternativa para obtener el minimum spanning tree (lo vimos con los algoritmos greedy, clase 06)

¿Como saber si T tiene un ciclo? (clase 03)

Grafos: caminos mas cortos

El algoritmo de Dikjstra encuentra los caminos mas cortos de desde un nodo origen hacia los demás



$distancias$ = infinito para cada nodo, 0 para el nodo origen
$procesado = 0$ para cada nodo
$predecesor = -1$ para cada nodo

Agregar el nodo origen a una fila q con $distancias[origen]$ como prioridad

Para cada nodo
  u = sale nodo de la fila de prioridad q

  Marca u como procesado, es decir $procesado[u] = 1$
    Para cada j vecino de u:
      $nueva\_distancia$ = $distancia[u]$ + peso de la arista de u a j

      Si $nueva\_distancia < distancia[j]$
        $distancia[j] = nueva\_distancia$
        $predecesor[j] = u$
        Agregar j a la fila, con prioridad $distancia[j]$ si j no ha sido procesado

Ejercicios

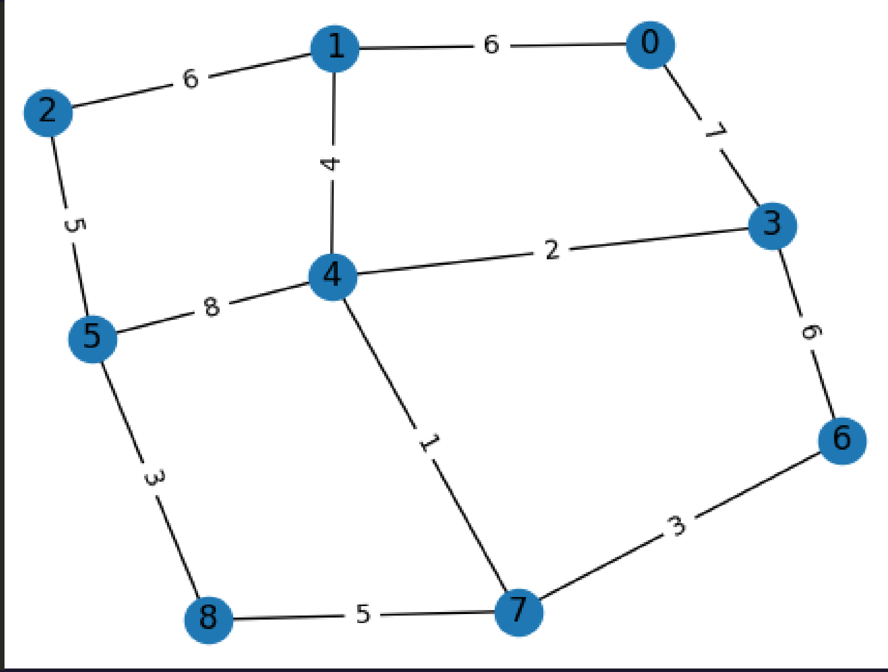
Puedes encontrar estos grafos en Drive (clase 14).
El primero es no dirigido, el segundo no

- Crea un programa de Python que pueda leerlos
  La primer línea indica su numero de nodos
  Las siguientes líneas son cada arista, formadas por dos nodos y un peso

- Aplica los algoritmo de Prim, Kruskal y Dijkstra para spanning tree mínimo y caminos mas cortos

- Guárdalo para entregarlo mas adelante

```
9
0 3 7
0 1 6
1 4 4
1 2 6
2 5 5
3 6 6
3 4 2
4 7 1
4 5 8
5 8 3
6 7 3
7 8 5
```




```
6
0 1 1
0 3 2
3 1 4
3 4 3
1 2 6
2 4 1
2 5 2
4 5 1
```