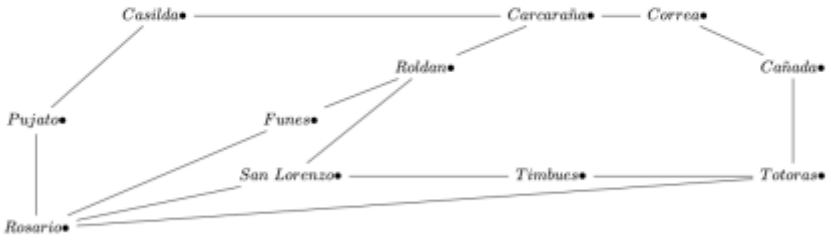
¿Qué es un Grafo?

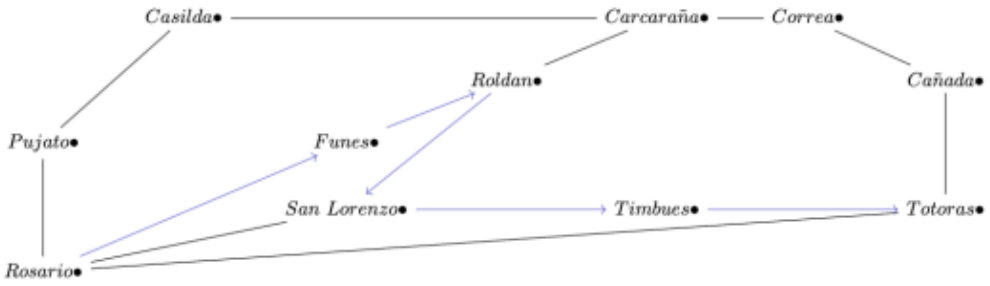
Definición. Un **grafo** (también llamado grafo no dirigido o grafo simple) G consiste en un conjunto de vértices (o nodos) y un conjunto E de aristas (o arcos) tal que cada arista $e \in E$ se asocia con un **par no ordenado** de vértices. Si la arista e está asociada con los vértices v y w, se escribe $e = (v, w)$ o $e = (w, v)$. Es importante notar que ambas son exactamente la misma arista.



El grafo de la Figura es un grafo simple: *Casilda* y *Pujato* son ejemplos de nodos, y el par no ordenado (*Casilda*, *Pujato*) es una arista del grafo.

¿Qué es un camino?

Definición. Sea un grafo $G = (V, E)$, un **camino** en dicho grafo es una sucesión de vértices $v_0, v_1, ..., v_n$, y aristas $l_0, l_1, ..., l_{n-1}$, tales que $l_i = (v_i, v_{i+1})$ para todo $i \in \{0, 1, ...n\}$. La **longitud** del camino se determina por la cantidad de aristas que recorre, en este caso, n.



En la Figura se muestra el camino formado por la sucesión de vértices *Rosario*, *Funes*, *Roldan*, *San Lorenzo*, *Timbues*, *Totoras*, y las aristas (*Rosario*, *Funes*), (*Funes*, *Roldan*), (*Funes*, *San Lorenzo*), (*San Lorenzo*, *Timb* (*Timbues*, *Totoras*). La longitud de dicho camino es 5.

¿Qué es un Grafo Dirigido?

Definición Un **grafo dirigido** G consiste en un conjunto V de vertices y un conjunto E de aristas tal que cada arista $e \in E$ se asocia con un **par ordenado** de vértices. Si la arista e esta asociada con los vertices u y w, se escribe $e = (v, w)$. Es importante considerar que en este contexto que las aristas (v, w) y (w, v) representan cosas distintas. El siguiente ejemplo muestra un grafo dirigido.

¿Qué es un Grafo Ponderado o Grafo con Pesos?

Definición Llamamos **grafo ponderado** o **grafo con pesos** a un grafo $G = (V, E)$ con una función $w : E \to \mathbb{R}$, es decir, un grafo donde a cada arista se le asigna un peso. Notamos $w(i, j)$ al peso de la arista que va de i a j, si esta existe.

Definición En un grafo con pesos, modificamos la definición de **longitud de un camino**. La longitud del camino en un grafo con pesos es la suma de los pesos de las aristas que recorre dicho camino.

Resumen de que es un Grafo

¿Qué son?

1. un conjunto de vértices (que representan cualquier cosa).
2. un conjunto de aristas (relaciones entre esos vértices.

Características

1. Dirigido o no dirigido.
2. Ponderado o sin ponderar.
3. Simple o completo.
4. Con bucles o sin bucles.

Cuando decimos "grafo" sin más, nos referimos a un grafo simple, no dirigido, sin pesos y sin bucles. Cualquier otra aclaración será dada.

¿Cómo hacemos un Grafo como un TAD?

`__init__()` Que crea un grafo vacío.

`add_node(x)` Que agrega un nodo al grafo.

`remove_node(x)` Que remueve un nodo del grafo.

`add_edge(x,y)` Que agrega la arista entre los nodos x e y al grafo.

`remove_edge(x, y)` Que remueve la arista entre x e y del grafo.

`are_adjacent(x, y)` Devuelve True si x e y son adyacentes, False en caso contrario.

`is_node(x)` Devuelve True si x es un nodo del grafo, False si no.

`get_nodes()` Devuelve una lista de nodos del grafos.

`get_adjacent(x)` Devuelve una lista de todos los nodos adyacentes al nodo x.

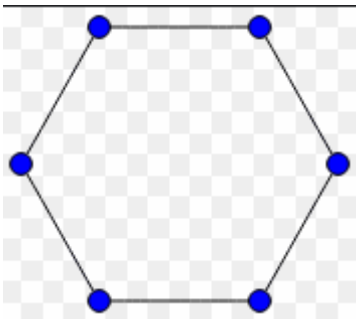
Diferencia con el TAD Grafo Ponderado

`add_edge(x,y, w)` Que agrega la arista entre los nodos x e y, con peso w, al grafo ponderado.

`are_adjacent(x, y)` Devuelve el peso de la arista (x,y) si x e y son adyacentes, imprime un error en caso contrario.

Otros tipos de Grafos

Ciclo Dado un grafo (dirigido o no dirigido) $G = (V, E)$ llamamos ciclo a un camino de longitud $>= 2$, que comienza y termina en el mismo vértice.



Definición Un **grafo conexo** es un grafo donde, dado cualesquiera dos vértices, siempre existe un camino que comienza en uno y termina en el otro. Todos los ejemplos que grafos que hemos visto hasta ahora son conexos. A continuacion, se da un ejemplo de un grafo **no conexo**.

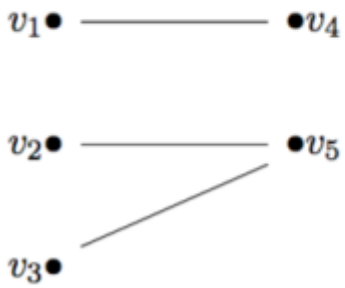
Intuitivamente, un grafo conexo es aquel "de una sola pieza". Mientras que los grafos no conexos parecen, a simple vista, estar formado por varias partes. A cada una de estas "partes" se la llama **componente conexa**. Se nota con $\kappa(G)$ a la cantidad de componentes conexas del grafo. En el ejemplo, $\kappa(G) = 3$.

Definición Sea $G = (V, E)$. Si $U \subseteq V$, el **subgrafo de G inducido por** U es el subgrafo cuyo conjunto de vértices es U y que contiene todas las aristas de G de la forma (v, w) donde $v, w \in U$.

Definición Un subgrafo G' de un grafo $G = (V, E)$ es un **subgrafo inducido** si existe un conjunto $U \subseteq V$ tal que G' es el subgrafo de G inducido por U.

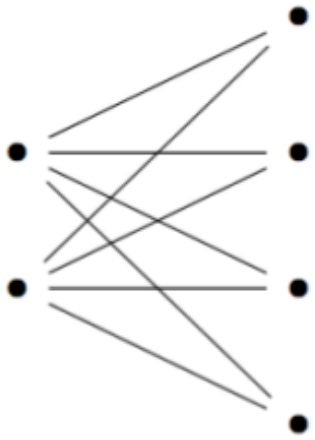
Definición El grafo completo de n vértices, notado como K_n, es el grafo simple con n vértices y una arista entre cada par de vértices distintos. A modo de ejemplo, se muestra el grafo K_4.

Por ejemplo, el grafo de la siguiente figura es bipartito con $V_1 = \{v_1, v_2, v_3\}$ y $V_2 = \{v_4, v_5\}$.



Definición El grafo bipartito completo de m y n, denotado $K_{m,n}$ vértices es el grafo simple donde el conjunto de vértices puede particionarse en V_1 con m vértices y V_2 con n vértices, y donde el conjunto de aristas consiste en todas las aristas de la forma (v_1, v_2) con $v_1 \in V_1$ y $v_2 \in V_2$.

Se muestra como ejemplo el grafo $K_{2,4}$.



Algoritmo de Dijkstra

Este algoritmo Dijkstra **encuentra la longitud de una ruta más corta** del vértice a al vértice z en un **grafo** $G = (V, E)$ **ponderado y conexo**. El peso de la arista (i, j) es $w(i, j) > 0$ (es decir, requerimos que los pesos sean siempre positivos).

Es importante verificar que estas condiciones se cumplen antes de aplicar el algoritmo.

Arboles como Grafos

Los **árboles** son un tipo especial de **grafo**. La definición que dimos en unidades anteriores se puede expresar ahora en términos de teoría de grafos.

Definición. Un **árbol** (libre) T es una grafo simple que satisface lo siguiente: si v y w son vértices en T existe un camino único de v a w.

Recordemos también que llamamos **árbol con raíz** a un árbol donde se ha designado un nodo especial, llamado raíz.

Ahora que sabemos teoría de grafos podemos, además, dar algunas propiedades adicionales de árboles.

Propiedad Un grafo conexo y sin ciclos es un árbol. En efecto, como el grafo es conexo entre dos vértices cualesquiera siempre hay un camino.

Propiedad Un grafo conexo con n vértices y $n-1$ aristas es necesariamente un árbol.

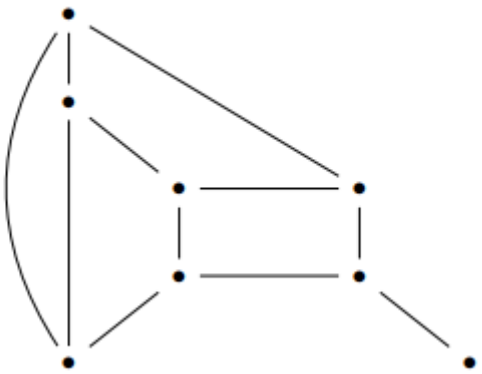
Propiedad Si un grafo con n vértices y $n-1$ aristas, no contiene ciclos entonces necesariamente es un árbol.

Árbol de Expansión

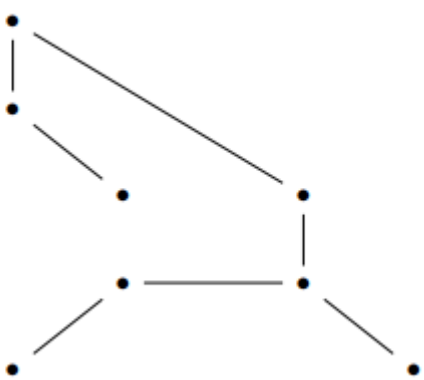
Se dice que árbol T es un **árbol de expansión** de un grafo G si:

1. T es un subgrafo de G.
2. T es un árbol.
3. T contiene todos los vértices de G.

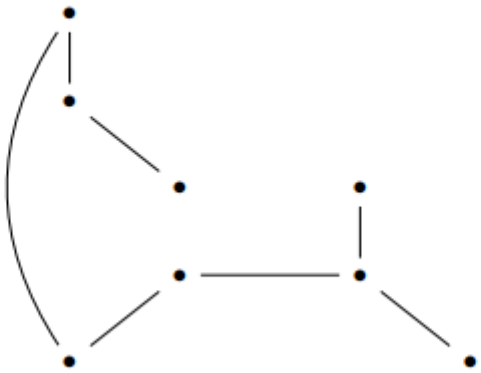
Por ejemplo, dado este grafo



Este es un árbol de expansión de ese grafo:



Nota El árbol de expansión de un grafo en general no es único, por ejemplo, acá mostramos otro árbol de expansión distinto del mismo grafo.



Un árbol de expansión para un grafo existe sí y sólo sí el grafo es conexo.

Algoritmo de Búsqueda en Profundidad

El algoritmo de búsqueda en profundidad, normalmente llamado DFS por sus siglas en inglés *Depth First Search* permite encontrar el árbol de expansión de un grafo conexo.
La estrategia consiste en partir de un vértice determinado v y a partir de allí, cuando se visita un nuevo vértice, explorar cada camino que salga de él. Un camino deja de explorarse cuando se llega a un vértice ya visitado. Si existen vértices no alcanzables, el recorrido queda incompleto; entonces, se debe "volver hacia atrás" hasta encontrar un vértice donde hayamos dejado un camino sin explorar, y repetir el proceso.

input: Un grafo G = (V, E)
output: Un árbol T = (V', E')
El algoritmo construye un árbol de expansión para el grafo G.
variables

```
T = Grafo() # Grafo sin vértices
P = Pila() # Una pila vacía
visitados = [] # Una lista de nodos visitados
elegimos origen (un vertice en V)
agregamos origen a S
mientras S no sea vacia hacer
        v = S.pop()
        visitados.append(v)
        para cada w adyacente a v en G hacer
                si w no esta en visitados entonces
                        visitados.append(w)
                        P.push(w)
fin mientras
```

Árbol de Expansión Mínima

Definición Sea G un grafo con pesos. Un árbol de expansión mínima de G es un árbol de expansión de G que tiene peso mínimo entre todos los posibles.

Nota El algoritmo DFS no asegura que el árbol encontrado sea de peso mínimo.

El **algoritmo de Prim** permite encontrar un **árbol de expansión mínimo** para un grafo con pesos conexo de vértices $v_1, v_2, ...v_n$.
El algoritmo comienza con un único nodo, elegido mediante algún criterio, e incrementa continuamente el tamaño de un árbol agregando sucesivamente vértices cuya distancia a los anteriores es mínima.
Esto significa que en cada paso, las aristas a considerar son aquellas que inciden en vértices que ya pertenecen al árbol.

El árbol está completamente construido cuando no quedan más vértices por agregar.
input: Un grafo G = (V, E)
output: Un árbol T = (V', E')
El algoritmo construye un árbol de expansión mínimo para el grafo G.
variables
T = Grafo() # El grafo vacio
U = [v] # v es el vértice inicial, elegido con algún criterio
mientras T != U **hacer**
    elegimos (u,v) la arista más barata / u esta en U y v esta en V - U.
    V' = V' + [v]
    E' = E' + (u,v)
fin mientras

El **algoritmo de Kruskal** es otro algoritmo que permite encontrar el **árbol de expansión mínimo** para un grafo con pesos conexo de vértices $v_1, v_2, ...v_n$.
Este algoritmo funciona eligiendo siempre la arista de menor costo que no forme un ciclo, aunque el grafo quede desconexo.
Comienza con un grafo que contiene los mismos nodos que el grafo inicial, pero ninguna de las aristas. Luego, va a agregando aristas del menor costo posible, de forma tal que no se formen ciclos. El árbol está completamente

construido cuando el grafo resultante es conexo

input: Un grafo G = (V, E)
output: Un árbol T = (V', E')
El algoritmo construye un árbol de expansión mínimo para el grafo G.
variables
T = Grafo(V) # El grafo con los mismos vértices, sin aristas.
mientras T no sea conexo **hacer**
    elegimos (u,v) la arista más barata / T no tiene ciclos
    E' = E' + (u,v)
fin mientras