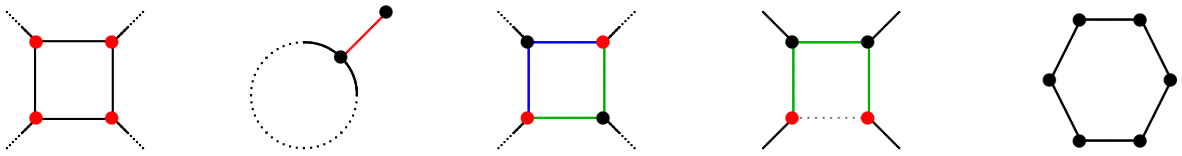
1.4 Grafos no dirigidos acíclicos: Árboles

Hasta ahora hemos buscado caminos y estudiado la conexión de un grafo, sin entrar en el estudio de los ciclos y la aciclicidad. En esta sección trataremos de ello, introduciéndonos brevemente en el estudio los grafos acíclicos no dirigidos.

Los resultados siguientes, aunque sencillos (casi "simplones") son básicos en el estudio y ponen de manifiesto la imbricación entre conexión y ciclos

- ⋆ Cualquier vértice que esté en un ciclo tiene al menos grado 2 (el recíproco no es cierto).
- ⋆ Si un vértice tiene grado 1, la arista incidente no forma parte de ningún ciclo.
- ⋆ Si dos vértices pertenecen a un mismo ciclo, están conectados al menos por dos caminos distintos.
- ⋆ Si eliminamos una arista de un ciclo, los vértices extremos siguen estando conectados.
- ⋆ Si un grafo consta únicamente de un ciclo tiene tantas aristas como vértices.



Bastan estas gráficas para considerar la validez de los resultados anteriores.

Aunque también sencillos los resultados siguientes son más elaborados:

Lema 30.- *Si $G=(V,A)$ es un grafo conexo y a es una de sus aristas, entonces:*
$(V, A-\{a\})$ es un grafo conexo $\iff$ a está en un ciclo.

Demostración:

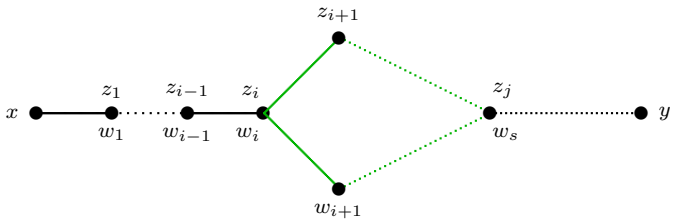
Si la arista a une los vértices x e y, y $(V, A-\{a\})$ es conexo hay un camino C uniendo estos dos vértices que no pasa por a, en consecuencia $C \cup a$ forma un ciclo en G.

Recíprocamente, si a está en un ciclo y la eliminamos, el grafo resultante sigue siendo conexo, pues hay otro camino que une los puntos extremos. ■

Lema 31.- *Si dos vértices de un grafo están conectados por dos caminos distintos el grafo tiene al menos un ciclo.*

Demostración:

Si $x\, z_1\, z_2 \cdots z_p\, y$ y $x\, w_1\, w_2 \cdots w_r\, y$ son caminos distintos, tienen al menos un vértice distinto. Si recorremos los dos caminos empezando en x, como tienen al menos un vértice distinto, habrá un vértice z_i (que puede ser x) a partir del cual se separan, luego la primera vez que vuelvan a coincidir en un vértice se forma el ciclo –y deben volver a juntarse pues ambos acaban en y, luego se juntan en y o en otro anterior–. (Ver la figura de la derecha donde aparece en verde el ciclo formado.)



■

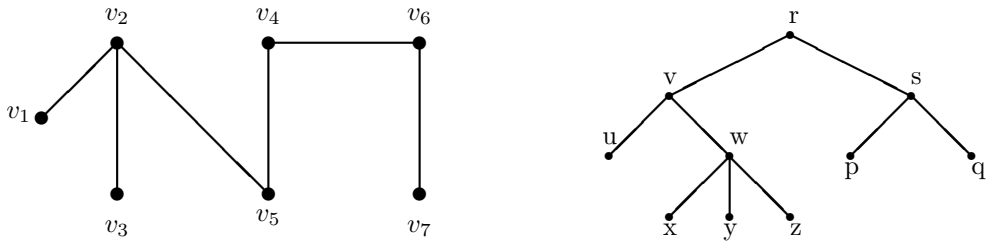
Los resultados que veremos aquí no son mucho más difíciles que los vistos hasta ahora, pero primero definamos con propiedad los elementos a tratar:

Definición 32.- *A un grafo no dirigido, acíclico y conexo se le llama **árbol**. Si no es conexo se denomina **bosque** y, cada componente conexa es un árbol.*

*En un árbol, a los vértices de grado uno se les llama **hojas**.*

En el estudio de los grafos acíclicos nos limitaremos a los conexos, los árboles, pues para los no conexos (bosques) basta aplicar los resultados a cada una de las componentes conexas.

Los árboles figuran entre los tipos más importantes de grafos y se presentan en varias aplicaciones: los árboles genealógicos, los organigramas o las redes de distribución son ejemplos de ellos. Pueden usarse para presentar, organizar o analizar redes eléctricas, relaciones mercantiles, estructuras de la programación, etc. Las dos representaciones siguientes ilustran su estructura:



donde se aprecia claramente la aciclicidad y, en el segundo, su imagen arbórea. Los vértices v_1, v_3 y v_7 son hojas del primer árbol.

Algunas de las buenas propiedades de los árboles son

- ⋆ Entre cada par de vértices hay un único camino que los une.
- ⋆ Un árbol tiene un número fijo de aristas.

Estos resultados se obtienen directamente de los siguientes que son definiciones equivalentes de árbol

Proposición 33.- *Las siguientes condiciones sobre un grafo $G = (V, A)$ son equivalentes:*

a) *$G = (V, A)$ es un árbol.*

b) *$G = (V, A)$ es conexo y, para cada $a \in A$, se tiene que $G_1 = (V, A - \{a\})$ no es conexo.*

c) *G es acíclico y para cada x, y con $\{x, y\} \notin A$, se tiene que $G_1 = (V, A \cup \{\{x, y\}\})$ no es acíclico.*

Demostración:

a) $\Rightarrow$ b) G es conexo por ser un árbol. Si al suprimir la arista $\{x, y\}$ el grafo $G_1 = (V, A - \{\{x, y\}\})$ sigue siendo conexo, es que hay un camino C (que no usa $\{x, y\}$) uniendo los vértices x e y, pero entonces $C \cup \{y, x\}$ es un ciclo en G en contra de que es acíclico.

b) $\Rightarrow$ c) G es conexo, y si tiene un ciclo, al suprimir una arista del ciclo el grafo resultante seguirá siendo conexo, en contra de la hipótesis, por lo que G es acíclico. Si al grafo conexo G le añadimos una arista $\{x, y\}$, con $x, y \in V$, se forma un ciclo entre esta arista y el camino que conecta los vértices x e y en el grafo G.

c) $\Rightarrow$ a) Si el grafo acíclico G no fuera un árbol, sería no conexo y, por tanto, tiene un par de vértices x e y en componentes conexas distintas y al añadir la arista $a = \{x, y\}$ resultará un grafo igualmente acíclico, en contra de la hipótesis. ■

La primera propiedad es ahora fácil de comprobar, pues dos vértices cualesquiera están unidos por un camino si y sólo si el grafo es conexo; y el camino es único si y sólo si es acíclico (cada par de vertices de un ciclo están unidos por dos caminos distintos). Luego *dos vértices cualesquiera están unidos por un único camino* (que también es una definición equivalente de árbol).

Para la segunda propiedad, hay que "leer" el significado de las definiciones equivalentes de la proposición anterior:

- ⋆ Un árbol tiene el mayor número posible de aristas manteniendo la aciclicidad (al adición de una arista crea un ciclo).

- ⋆ Un árbol tiene el menor número posible de aristas para mantener la conexión (la eliminación de cualquier arista desconecta el grafo).

que nos lleva a considerar que el número de aristas en un árbol es fijo. De hecho, un árbol de n vértices tiene exactamente $n-1$ aristas:

Lema 34.- *Un árbol finito con al menos una arista (o con más de un vértice) tiene al menos una hoja (de hecho al menos dos hojas).*

Demostración:

Elegimos un vértice cualquiera del grafo, x, y se construye a partir de él un camino hasta que no podamos seguir, entonces el vértice final del camino es una hoja: por ser acíclico de él no pueden salir otras aristas (distintas de la de llegada) a vértices anteriores del camino y si no se puede seguir tampoco salen aristas a otros vértices, luego la única arista incidente en él es la de llegada por lo que tiene grado 1.

(Si x tiene grado 1 es la segunda hoja y si tiene grado mayor prolongamos el camino anterior a partir de x siguiendo otra arista hasta que no podamos más, entonces el vértice final para esta parte también es una hoja –por la misma razón que antes–.)

■

Teorema 35.- *Sea $G=(V,A)$ un grafo con n vértices. Entonces G es un árbol si y solo si es conexo y tiene $n-1$ aristas.*

Demostración:

$\Rightarrow\rfloor$ Si G es un árbol con $n=1$ vértices, tiene 0 aristas; luego es cierto.

Sea G es un árbol con n vértices y m aristas. Si $n \geq 2$ tiene al menos una hoja, luego eliminando la hoja y la arista incidente nos queda un grafo con $n-1$ vértices y $m-1$ aristas que es también conexo y acíclico, luego es un árbol. Si $n-1 \geq 2$ tiene al menos una hoja, si eliminamos la hoja y la arista incidente nos vuelve a quedar un árbol de $n-2$ vértices y $m-2$ aristas. Repitiendo el proceso sucesivamente hasta obtener un árbol de $n-(n-1)$ vértices y $m-(n-1)$ aristas, es decir, un árbol de $n-(n-1)=1$ vértice y $m-(n-1)$ aristas. Pero si es un árbol con un vértice tiene que tener 0 aristas, luego $m-(n-1)=0$ de donde $m=n-1$.

$\Leftarrow\rfloor$ G es conexo por hipótesis, por lo que basta ver que es acíclico.

Si G tiene $n=1$ vértices y 0 aristas es acíclico.

Si $n \geq 2$ y tiene $n-1$ aristas tiene al menos una hoja (si $gr(v_i) \geq 2$ para todo i, se tendría que $2(n-1)=\sum_{i=1}^{n} gr(v_i) \geq \sum_{i=1}^{n} 2 = 2n$ lo que es absurdo), luego eliminando la hoja y la arista incidente en ella nos queda un grafo G_1 con $n-1$ vértices y $n-2$ aristas que también es conexo y con los mismos ciclos que en G (la arista eliminada no pertenecía a ningún ciclo).

Repetimos lo anterior para G_1: si $n-1 \geq 2$ y tiene $n-2$ aristas tiene al menos una hoja, luego eliminando la hoja y la arista incidente en ella nos queda un grafo G_2 con $n-2$ vértices y $n-3$ aristas que es conexo y con los mismos ciclos que en G_1 (que tiene los mismos que G).

Y sucesivamente, hasta llegar a un grafo G_{n-1} con 1 vértice y 0 aristas y con los mismos ciclos que el inicial. Luego como este es acíclico, el inicial también es acíclico.

■

Corolario 36.- *Un bosque con n vértices y k componentes conexas tiene $n-k$ aristas.*

Demostración:

Si es un bosque, cada componente conexa es un árbol, luego si una componente conexa tiene n_i vértices tiene n_i-1 aristas. Entonces el bosque tendrá $\sum_{i=1}^{k}(n_i-1)=\left(\sum_{i=1}^{k} n_i\right)-k=n-k$ aristas. ■

Corolario 37.- *Sea $G = (V, A)$ un grafo con n vértices. Entonces G es un árbol si y solo si es acíclico y tiene $n - 1$ aristas.*

Demostración:

$\Rightarrow\rfloor$ Si es un árbol, es acíclico y si tiene n vértices, tiene $n - 1$ aristas.

$\Leftarrow\rfloor$ G es acíclico luego es un bosque con k componentes conexas y $n - k$ aristas. Como tiene $n - 1$ aristas, debe ser $n - 1 = n - k$ de donde $k = 1$ y sólo tiene una componente conexa, por lo que es un árbol.

■

1.4.1 Árboles generadores

Definición 38.- *Llamaremos **peso** de un grafo (o digrafo) $G = (V, A)$ a una función real positiva sobre el conjunto de aristas (arcos) del grafo, es decir, a una función $\omega\colon A \to \mathbb{R}^+$.*

*El peso de cada arista lo denotaremos por $\omega(\{v_i, v_j\}) = \omega_{ij}$ y llamaremos **peso de una trayectoria** a la suma de los pesos de las aristas que la componen.*

En un grafo simple pesado, se llama **matriz de pesos** del grafo a la matriz $\Omega = (\omega_{ij})_{n\times n}$, donde pondremos $\omega_{ij} = \infty$ si no hay arista desde el vértice v_i al vértice v_j y $\omega_{ii} = 0$, ceros en la diagonal.

Nota: A efectos prácticos trabajaremos sobre grafos simples, pues si queremos minimizar (o maximizar) el peso de un recorrido cogeríamos siempre la arista de menor (o mayor) peso de las múltiples.

Definición 39.- *Llamaremos **árbol generador** (o **árbol de expansión**) de un grafo conexo a cualquier subgrafo con el mismo número de vértices que sea árbol (los subgrafos que mantienen todos los vértices suelen denominarse* subgrafos parciales*).*

*Un **árbol generador mínimo** de un grafo conexo pesado es un árbol generador con mínima suma de pesos.*

Es decir, un árbol generador representa la mínima estructura necesaria para mantener la conexión entre los vértices. Y así se usa en la práctica: si únicamente interesa mantener la conexión del grafo, puedo eliminar todas las aristas innecesarias con el consiguiente ahorro.

Proposición 40.- *Todo grafo conexo tiene al menos un árbol generador.*

Demostración:

Como el grafo es conexo, si eliminamos una arista de un ciclo del grafo obtenemos un subgrafo conexo (ver Lema 30) con al menos un ciclo menos; luego basta ir eliminando, sucesivamente, una arista cualquiera de cada ciclo que quede en el grafo. Como los subgrafos formados son conexos, cuando no queden ciclos también será acíclico y por tanto será el árbol generador buscado.

■

La busqueda de árboles generadores mínimos en un grafo se realiza mediante procesos algorítmicos específicos. Los más usuales son los dos Algoritmos de Kruskal y el Algoritmo de Prim.

1.4.1.1 Algoritmo de Kruskal (versión destructiva).

Comencemos con este algoritmo que reproduce el proceso descrito en la prueba de existencia de árbol generador (de ahí el apelativo *destructivo*): se trata de ir eliminando aristas del grafo hasta obtener un árbol generador mínimo. Para consiguir el objetivo, necesitamos tener en cuenta que:

- ⋆ Si G es un grafo conexo de n vértices y m aristas, hay que quitar $m-(n-1)$ aristas (para que nos queden las $n - 1$ aristas del árbol).
- ⋆ Hay que eliminar aristas pertenecientes a ciclos, es decir, cuya eliminación no desconecte el grafo.
- ⋆ Buscamos un árbol generador mínimo, luego hay que ir eliminando las aristas más pesadas.

Sea $G = (V, A)$ un grafo conexo de n vértices y Ω su matriz de pesos. El proceso de Kruskal admite varias versiones de algoritmos que lo describan y, por supuesto muchas más programaciones. La versión del algoritmo descrita a continuación refleja fielmente el proceso a realizar: usaremos un conjunto A con las aristas del grafo y un conjunto B que nos indique las aristas que faltan por chequear, y se trata de ir eliminando de A las aristas de mayor peso que no desconecten el grafo. El proceso se detiene cuando en A sólo quedan las $n-1$ aristas que forman el árbol:

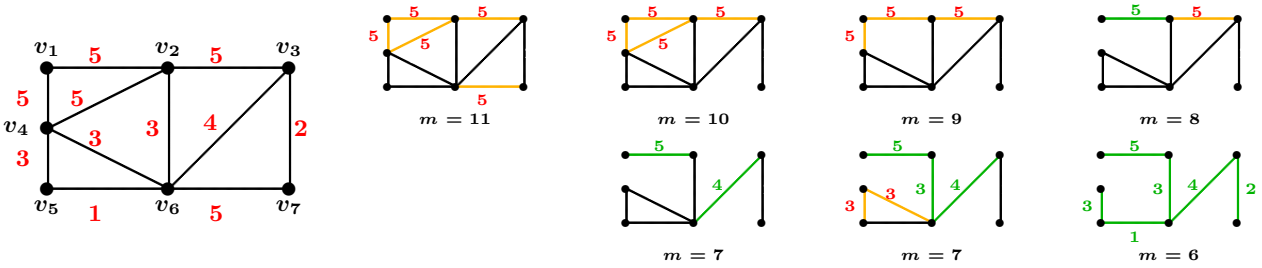
Algoritmo 3.- *(destructivo de Kruskal)*

```
inicio: n; Ω; A; B = A
mientras sea tamaño(A) > n − 1
      tomar a ∈ B con peso máximo
      hacer A = A − {a}
      si (V, A) no es conexo
         hacer A = A ∪ {a}
      fin
      hacer B = B − {a}
fin
```

Observaciones 41.-

- ⋆ Como en este algoritmo se busca la arista de peso máximo, en la matriz de pesos Ω deben cambiarse los valores ∞ por ceros.
- ⋆ Ya se ha comentado que este algoritmo admite mejores variantes: por ejemplo, si recorremos las aristas de A de manera ordenada puede evitarse el uso del conjunto B que sólo aparece como control de las aristas chequeadas.
- ⋆ La elección de una u otra arista de peso máximo da lugar a distintos árboles generadores mínimos (salvo que sólo haya uno).

Ejemplo 42.- Aplicamos el algoritmo "destructivo" de Kruskal al grafo de abajo. Tiene $n = 7$ vértices y $m = 11$ aristas, luego nos hemos de quedar con $7 - 1 = 6$ aristas (y quitar 5).



Se van eliminando sucesivamente las aristas de mayor peso que forman ciclos para no desconectar el grafo. No se eliminan las aristas que desconectan el grafo (en el dibujo que reproduce el proceso se cambian a color verde). Cuando queden $n-1$ aristas (hayamos quitado las $m-(n-1)$ aristas sobrantes) tendremos el árbol buscado.

1.4.1.2 Algoritmo de Kruskal (versión codiciosa o constructiva).

El algoritmo *codicioso* de Kruskal obtiene el árbol generador mínimo de manera contraria al anterior, no eliminando sino eligiendo aristas de entre las de menor peso hasta conseguir formar el árbol.

Si el *destructivo* se basa en mantener la conexión mientras se eliminan las aristas, éste lo hace manteniendo la aciclicidad mientras se añaden aristas (por el Corolario 37).

El *truco* del algoritmo (y ¡atención! es también el proceso a seguir en las implementaciones del mismo) es el siguiente: partimos de un grafo sin aristas con los vértices aislados (luego en componentes conexas distintas) y vamos incorporando aristas sin que se formen ciclos; así en cada momento del proceso tenemos un grafo acíclico y entonces sólo puedo añadir una arista *si une vértices pertenecientes a componentes conexas distintas*. En efecto, un grafo acíclico es un bosque y cada componente conexa suya es un árbol, luego añadir una arista que una vértices de la misma componente conexa es añadir

una arista a un árbol con lo que se formaría un ciclo (ver la Proposición 33 y comentarios posteriores).

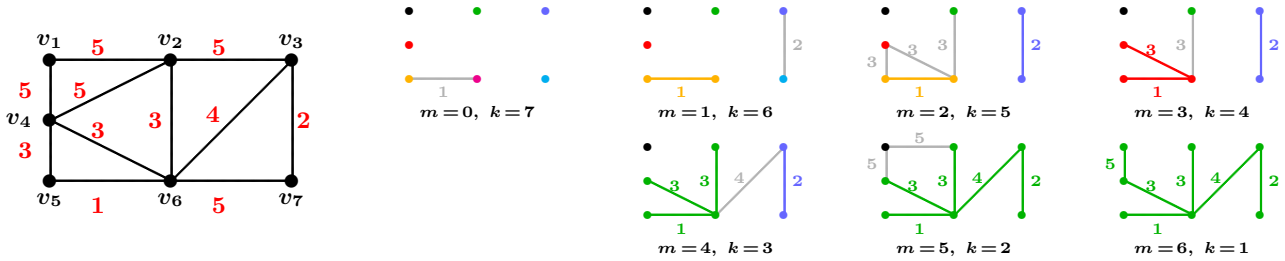
Sea $G = (V, A)$ un grafo conexo y Ω su matriz de pesos. La descripcion del algoritmo es similar a la del algoritmo destructivo, que requiere A y otro conjunto B, inicialmente vacío, donde vamos almacenando las aristas del árbol. El proceso se detiene cuando en B tengamos las $n-1$ aristas necesarias:

Algoritmo 4.- *(codicioso de Kruskal)*

```
inicio: n; Ω; A; B = ∅
mientras sea tamaño(B) < n − 1
      tomar a ∈ A con peso mínimo
      hacer B = B ∪ {a}
      si (V, B) no es acíclico
         hacer B = B − {a}
      fin
      hacer A = A − {a}
fin
```

El algoritmo, en seudocódigo, describe el proceso ideado por Kruscal. En la implementación práctica, el *truco* a que se hacía referencia anteriormente, se usa en la comprobación de la aciclicidad: si la nueva arista a añadida a B une vértices pertenecientes a distintas componentes conexas el grafo (V, B) será acíclico y no lo será si los vértices están en la misma componente conexa. En el ejemplo siguiente de la aplicación del algoritmo, se muestra en uso del *truco* señalando con colores distintos cada componente conexa.

Ejemplo 43.- Apliquemos el algoritmo de Kruskal sobre el grafo del ejemplo anterior. Usamos m para indicar las aristas elegidas y k para el número de componentes conexas que hay en cada paso (cada componente conexa de distinto color; cuando dos componentes conexas se unen con una arista forman una sola por lo que se redibujan juntas con uno de los dos colores):



1.4.1.3 Algoritmo de Prim.

En este algoritmo se construye al árbol generador mínimo a traves de una sucesión de árboles generadores mínimos. Se parte de un vértice inicial (árbol generador mínimo de 1 vértice) y se van añadiendo hojas hasta llegar a un árbol generador mínimo de n vértices (la adición de una hoja mantiene la conexión y la aciclicidad).

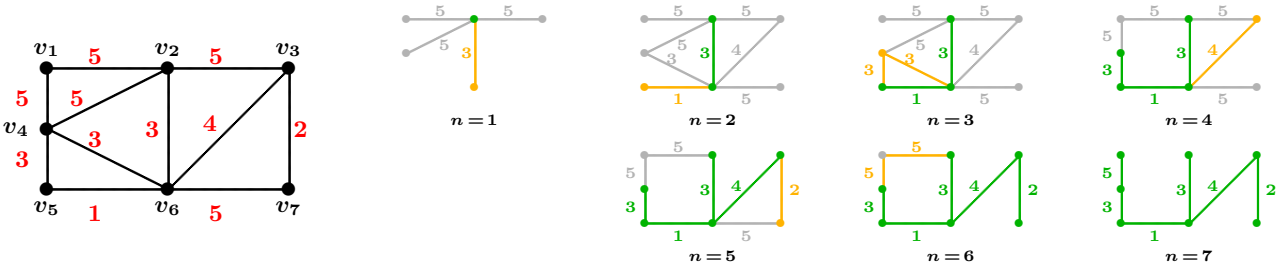
En cada paso del proceso se consideran las aristas que unen vértices del árbol actual con vértices aún no elegidos y se toma la de menor peso: el vértice no elegido de esa arista es la hoja buscada. El proceso acaba cuando incorporamos al árbol todos los vértices.

Sea $G = (V, A)$ un grafo conexo y Ω su matriz de pesos. Consideremos un conjunto B donde iremos almacenando las aristas elegidas en el proceso y otro conjunto W que nos indicará el conjunto de vértices para el que ya tenemos un árbol generador mínimo. El proceso se detiene cuando en B tengamos las $n-1$ aristas necesarias (o cuando en W tengamos todos los vértices de V):

Algoritmo 5.- *(de Prim)*

```
inicio: n; Ω; V; B = ∅; W = {v_p}
mientras sea W ≠ V
      tomar v ∈ V − W, tal que para algún w ∈ W, {v,w} tiene peso mínimo
      hacer W = W ∪ {v}
      hacer B = B ∪ {{v,w}}
fin
```

EJEMPLO 44.- Apliquemos el algoritmo de Prim sobre el grafo de los ejemplos anteriores, empezando en el vértice v_2 (aparece en verde el árbol generador mínimo que se va construyendo; en gris las aristas posibles, que unen vértices verdes con otros vértices; y en dorado las aristas de menor peso de entre las posibles):



Observaciones 45.- ⋆ La elección de un vértice u otro inicial es irrelevante en la aplicación del algoritmo de Prim.

⋆ El algoritmo de Prim (como el codicoso de Kruskal) elige $n-1$ aristas de las m del grafo para formar el árbol, mientras que el destructivo de Kruskal elige $m-(n-1)$ aristas de las m del grafo para eliminar. Por consiguiente, la elección de un algoritmo construtivo o destructivo depende del número de aristas.

⋆ La aplicación de cualquiera de los tres algoritmos a un grafo no pesado produce un árbol generador. Basta suponer cada arista de peso 1 o definir en el grafo una función peso cualquiera.

⋆ Si el grafo es no conexo, puede obtenerse un *bosque generador mínimo* aplicando el algoritmo a cada una de las componentes conexas.

⋆ Estos algoritmos (los tres) consiguen el árbol generador buscado si y sólo si el grafo es conexo. Si el grafo no es conexo, no solo no se consigue el objetivo, sino que dependiendo de su diseño puede crearse un bucle infinito.

De hecho, al ser estos algoritmos más eficientes que Warsall o Floyd, suelen usarse para comprobar si un grafo es o no conexo.

1.4.2 Ejercicios

4.1 Para cada uno de los grafos del Ejercicio 3.1 encontrar árboles generadores mínimos con cada uno de los tres algoritmos.

4.2 La tabla expresa, en decenas de kilómetros, las longitudes de varias carreteras conectando seis ciudades gallegas C, V, L, O, S y P. Representar mediante un grafo el mapa de carreteras.

	C					
C	–	S				
S	7	–	P			
P	–	6	–	V		
V	–	–	5	–	L	
L	9	10	10	–	–	O
O	–	–	–	11	11	–

¿Cuál de los dos algoritmos de Kruskal elegirías para obtener un árbol generador mínimo? ¿Por qué?

Usar ambos algortimos para comprobar que has acertado en la elección.

Introducir la matriz de pesos `W` del grafo y obtener a partir de ella la matriz de adyacencia `M`.

[i] Encontrar el número de aristas y de vértices del grafo. ¿Cuantas aristas hay que tomar para formar un árbol generador? ¿Cuántas aristas hay que eliminar del grafo para conseguirlo?

[ii] Obtener, a partir de `W`, la matriz `W1` de pesos del árbol generador mínimo obtenido arriba. Calcular su peso y encontrar las hojas del árbol.

[iii] Obtener las aristas con mayor peso en `W` y eliminarlas sucesivamente del mismo.

[iv] Comprobar, tras cada eliminación, si el grafo obtenido es conexo usando los algoritmos de Warsall o Floyd. Si no lo es colocar de nuevo la arista en el grafo.

[v] A la vista de `W`, ¿como podría cambiarse fácilmente por otra matriz que simplifique el proceso del apartado [iii]?

[vi] Construir una matriz `P` de 3 filas, que en la primera fila tenga los pesos de las aristas del grafo y en las otras dos filas los vértices extremos correspondientes a cada arista. Obtener la arista de mayor peso y eliminarla en `W`.

[vii] Reordenar `P` en una matriz `P_ord` de manera que los pesos de las aristas están ordenados de mayor a menor. Obtener la arista de mayor peso y eliminarla en `W`.

4.3 En el grafo usado en los ejemplos anteriores (42, 43 y 44) eliminar las aristas de peso mayor que 3. El grafo obtenido no es conexo, ¿verdad?

(a) ¿Cuántas componentes conexas tiene y cómo se reparten los vértices en ellas?

(b) De las aristas eliminadas, ¿cuáles unen vértices de la misma componente conexa y cuáles de distinta?

(c) Añadir una de las aristas que unen componentes conexas distintas y comprobar que su adición no crea nuevos ciclos en el grafo. ¿Cuántas componentes conexas hay ahora y cómo se reparten los vértices en ellas?

Construir `W`, la matriz de pesos del grafo, introduciendo únicamente los pesos de las aristas.

[i] Obtener a partir de ella la matriz `W1` resultante de eliminar las aristas indicadas.

[ii] Hallar la matriz `M1` de adyacencia del grafo obtenido y comprobar usando Warsall o Floyd que el grafo no es conexo.

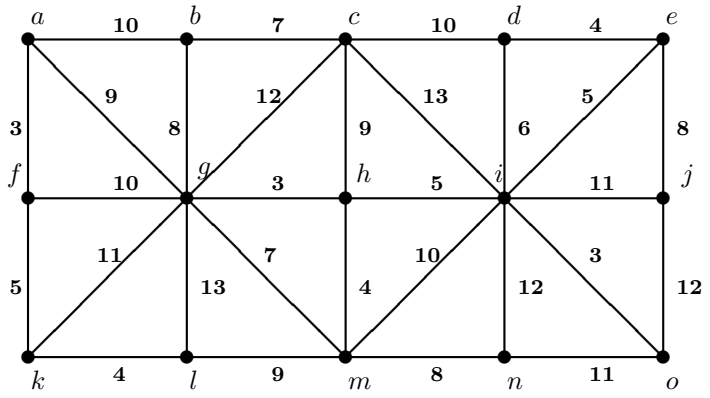
[iii] Observar la matriz resultante de comprobar la conexión: ¿cómo se refleja en ella que hay varias componentes conexas y los vértices que las forman?

[iv] Construir un vector `C` que refleje como se reparten los vértices en las distintas componentes conexas. Comprobar usando `C` cuales de las aristas eliminadas unen vértices de la misma componente conexa y cuales de distinta.

[v] Elegir una de las aristas que unen vértices de distintas componentes conexas y añadirla a `W1`, ¿cuál será ahora el vector `C`? Modifíquese a partir del vector `C` del apartado anterior.

4.4 Telefónica tendió demasiadas líneas entre un grupo de casas. En el grafo de la figura, los vértices son las casas y las aristas las líneas telefónicas. El peso de cada arista representa la longitud

de la línea. La compañía desea quitar las líneas sobrantes, de forma que dos casas cualesquiera sigan conectadas, y la longitud total del tendido sea la mínima posible. ¿Qué líneas hay que quitar? Dibújese el grafo resultante, indicando el algoritmo usado.



4.5 En el ejercicio anterior, encuentrese un árbol generador mínimo usando el algoritmo de Prim, comenzando desde el vértice h.

Considerar `W` la matriz de pesos del grafo.

[i] Obtener las aristas incidentes en h y elegir la de menor peso.

[ii] Tras el segundo paso de la aplicación del algoritmo de Prim se tienen los vértices h, g y m unidos mediante las aristas $\{g,h\}$ y $\{h,m\}$. Obtener una submatriz de `W` que muestre las aristas existentes entre estos tres vértices y los demás vértices.

[iii] De entre las aristas anteriores elegir la de menor peso ¿qué vértices une?

Prctica 2.-

1.- Construir una función `Quitar` que, a partir de la matriz de pesos de un grafo, obtenga un árbol generador mínimo con una implementación del algoritmo *destructivo* de Kruskal. Con dos salidas: en la primera el peso total del árbol obtenido, y en la segunda una matriz que lo represente (la matriz de adyacencia o la de pesos del árbol).

2.- Construir una función `Poner` análoga a la anterior pero que implemente uno de los dos algoritmos *constructivos* (Kruskal o Prim).

3.- Construir una función para obtener un árgol generador mínimo, tal que:

a) tenga como argumento de entrada la matriz de pesos de un grafo

b) compruebe si corresponde a un grafo conexo y que finalize si no lo es.

c) llame a la función `Quitar` o a la función `Poner`, según convenga a tenor del número de aristas del grafo.

d) devuelva primero, el peso del árbol generador mínimo y segundo, una matriz que lo represente.

Optativo: Implementar en una función el algoritmo constructivo **no** usado antes en la función `Poner` del punto 2, para utilizarla en la comprobación de la conexion de un grafo.