

## Tema 3

# Introducción a la teoría de grafos

Seguiremos al pie de la letra el capítulo 6 del libro “Lógica para informáticos y otras herramientas matemáticas” del Dr. Jesús García Miranda.

### 3.0.1 temporización

- Primera semana: Estudio de las cinco primeras secciones. Ejercicios 1 al 15 de la relación.
- Segunda semana: Estudio de las secciones 6, 7, 8 y 9. Ejercicios 16 al 28 de la relación.
- Tercera semana: Estudio de las secciones 10 y 11. Ejercicios 29 al 43 de la relación.

Todo lo anterior con el añadido y las omisiones que se indican a continuación:

### 3.1 Generalidades sobre grafos

- Incluir las definiciones

#### Definiciones 3.1 (extremos, grafo simple)

1. Los extremos del lado  $e \in E$  son los vértices  $u, v \in V$  tales que  $\gamma(e) = \{u, v\}$
2. Un grafo es simple si no tiene lazos ni lados paralelos.

Algunos autores llaman *grafos* a los grafos simples y *multigrafos* o *pseudografos* a lo que nosotros hemos llamado grafos.

- Omitir la definición 58.
- Este año no consideraré esencial el **algoritmo de Dijkstra** para grafos ponderados. Este algoritmo no figura en los apuntes del Dr. Jesús García Miranda previos al libro, pero si en su libro.

### 3.2 Representación matricial de grafos.

En esta sección sólo estudiaremos la matriz de adyacencia.

- Omitir desde la definición de la matriz de incidencia (definición 65 del libro, definición 62 en los apuntes) hasta el final de la sección.

### 3.3 Isomorfismo de grafos

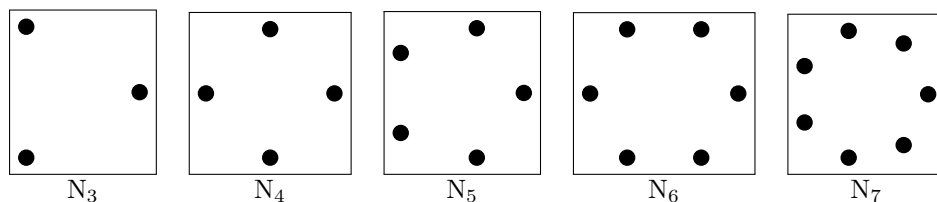
Sin cambios

### 3.4 Algunas familias de grafos.

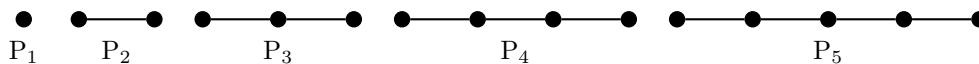
Sin cambios. Esta sección sólo aparece en el libro. Debéis de saber que algunos autores de teoría de grafos llaman  $P_n$  al grafo camino con  $n$  lados y otros al grafo camino con  $n$  vértices. La terminología de grafos puede cambiar de unos textos a otros. ¡¡¡Tened cuidado!!!

Veamos algunos ejemplos de grafos simples:

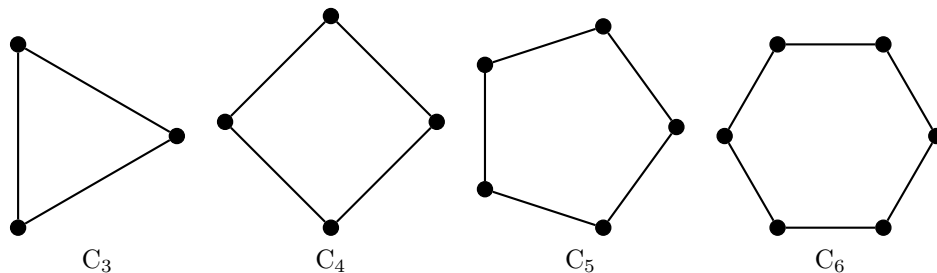
1. Grafo nulo o vacío con  $n$  vértices es aquel que no tiene lados.  $N_n$  o  $E_n$



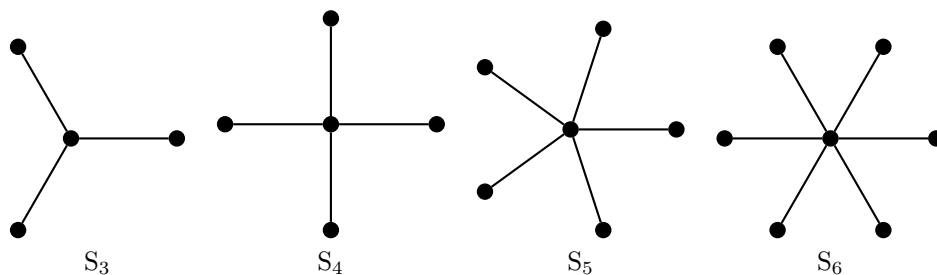
2. Grafo camino  $P_n$



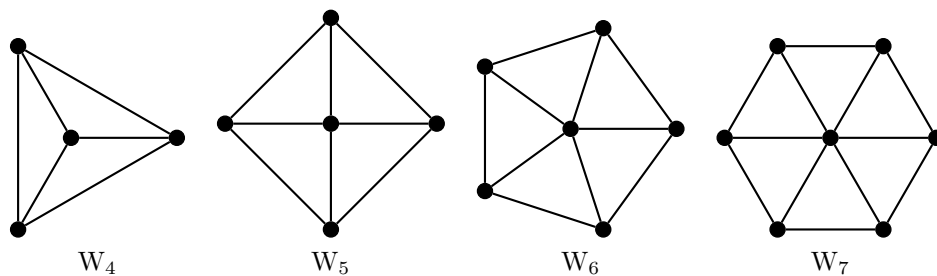
3. Grafo circular o poligonal  $C_n$ . Determinado por los vértices de un polígono regular. Los lados son los del polígono.



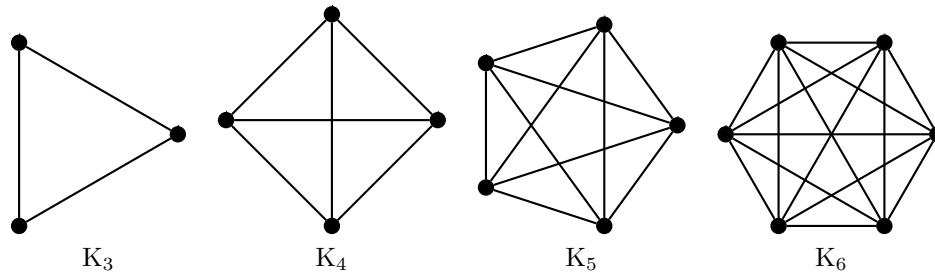
4. Grafo estrella  $S_n = K_{1,n}$ . Determinado por los vértices de un polígono regular y su centro. Los lados son los que unen el centro con cada uno de los vértices del polígono.



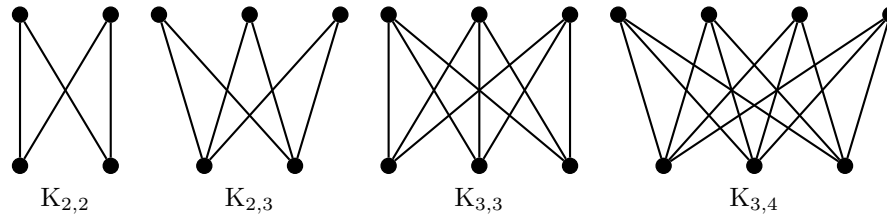
5. Grafo rueda  $W_n$ . Determinado por los vértices de un polígono regular y su centro. Los lados son los del polígono y los que unen el centro con cada uno de los vértices del polígono.



6. Grafo completo  $K_n$ . Determinado por  $n$  vértices. Los lados unen cada vértice con todos los demás.



7. Grafo bipartido completo  $K_{r,s}$ . Determinado por dos grupos de vértices, con  $r$  y  $s$  vértices respectivamente. Los lados unen cada vértice de un grupo con todos los demás vértices del otro grupo.



### 3.5 Sucesiones gráficas.

En esta sección nos apartamos ligeramente del libro en los algoritmos que utilizan el teorema de Havel-Hakimi. Pasamos a estudiarlos:

#### Algoritmo de demolición.

Decide si una sucesión finita es una sucesión gráfica generando un número finito de sucesiones finitas.

1. ¿Son todos los elementos de la sucesión 0? En caso afirmativo la sucesión es gráfica y hemos acabado.
2. ¿Existe un número en la sucesión que es mayor o igual que el número de elementos no nulos de la sucesión? En caso afirmativo la sucesión no es gráfica y hemos acabado.  
 item Elige un vértice que cumpla que no hay ninguno en la sucesión con mayor grado y márcalo(pivote). Si el grado marcado es  $r$ , elige  $r$  vértices que cumplan que entre los no elegidos no hay uno con mayor grado que alguno de los elegidos.
3. Genera una nueva sucesión poniendo a cero el marcado (pivote) y disminuyendo en una unidad los elegidos.
4. Vuelve a 1 con la nueva sucesión.

La diferencia con el libro es que no reordenamos y es más fácil reconstruir.

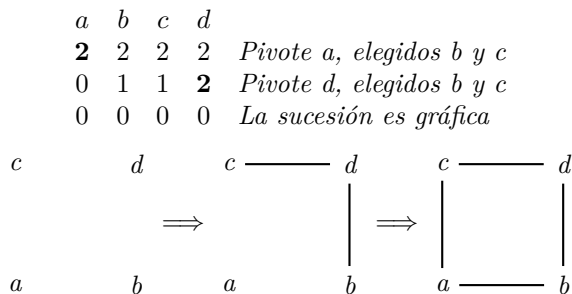
#### Algoritmo de reconstrucción paso a paso.

Si el algoritmo de demolición da como salida sucesión gráfica. Se construye un grafo para cada fila del algoritmo de demolición leyendo las filas de abajo hacia arriba.

1. Partimos del grafo nulo con  $n$  vértices que corresponde a la última fila de todos ceros.
2. Pasamos de una fila a la superior añadiendo al grafo de la fila  $r$  lados que conectan el vértice marcado en la fila superior con los elegidos (los que aumentan el grado en 1).

#### Ejemplos 3.1

1. Consideramos la sucesión 2, 2, 2, 2, que sabemos que es gráfica. Vamos a comprobarlo con el algoritmo.



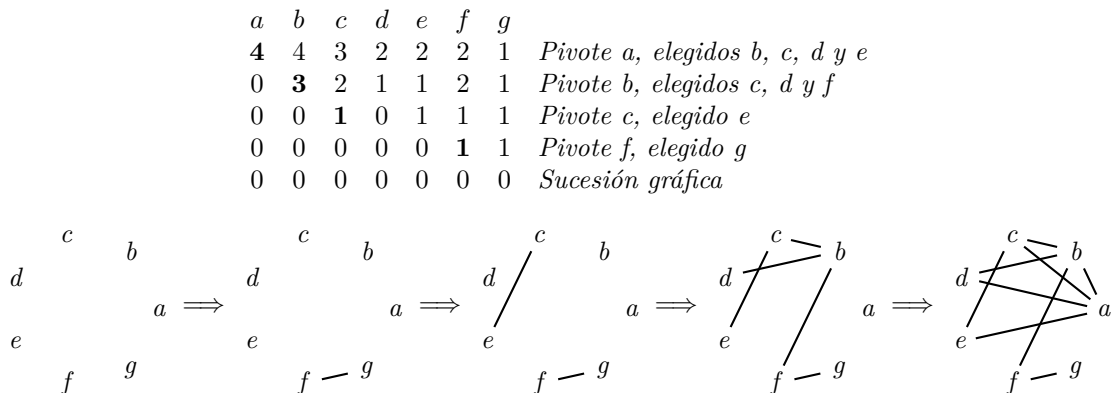
2. Tomamos ahora la sucesión 5, 4, 4, 2, 2, 1, que sabemos que no es gráfica. Vamos a comprobarlo.

$a$	$b$	$c$	$d$	$e$	$f$	
<b>5</b>	4	4	2	2	1	Pivote $a$ , elegidos $b, c, d, e$ y $f$
0	<b>3</b>	3	1	1	0	Pivote $b$ , elegidos $c, d$ y $e$
0	0	<b>2</b>	0	0	0	Pivote $c$ y no podemos elegir ningún vértice. La sucesión no es gráfica.

3. Sea ahora la sucesión 4, 3, 3, 2, 2, 1, que sabemos que no es gráfica pues la suma de sus términos es un número impar.

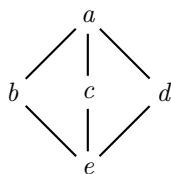
$a$	$b$	$c$	$d$	$e$	$f$	
<b>4</b>	3	3	2	2	1	Pivote $a$ , elegidos $b, c, d$ y $e$
0	<b>2</b>	2	1	1	1	Pivote $b$ , elegidos $c$ y $d$
0	0	<b>1</b>	0	1	1	Pivote $c$ , elegido $e$
0	0	0	0	0	<b>1</b>	Pivote $f$ , no podemos elegir. No es gráfica.

4. Consideramos la sucesión 4, 4, 3, 2, 2, 2, 1. Nos planteamos si es o no una sucesión gráfica.



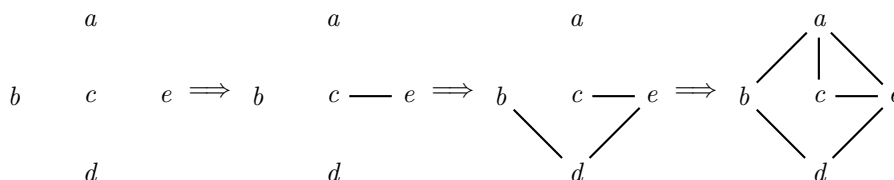
**Ejemplo 3.2** Este ejemplo es fundamental entenderlo y reflexionar sobre su significado.

Sea el grafo



con sucesión gráfica 3, 2, 2, 2, 3. Apliquemos los algoritmos de demolición y reconstrucción.

$a$	$b$	$c$	$d$	$e$	
<b>3</b>	2	2	2	3	Pivote $a$ , elegidos $b, c$ y $e$
0	1	1	<b>2</b>	2	Pivote $d$ , elegidos $b$ y $e$
0	0	<b>1</b>	0	1	Pivote $c$ , elegido $e$
0	0	0	0	0	Sucesión gráfica



Hemos cambiado la posición de algunos vértices para evitar cruces de lados. El grafo obtenido no es isomorfo al original de partida pero cumple que los grados de cada vértice coinciden con los del de partida.

El grafo obtenido es distinto porque el teorema de Havel-Hakimi demuestra que dado un grafo existe otro con la misma sucesión gráfica que cumple que los vértices de mayor grado son adyacentes entre sí.

Como en el grafo original  $a$  y  $e$  no son adyacentes en el reconstruido tienen que serlo y por tanto no puede salir el mismo. De aquí concluimos que la sucesión formada por los grados de los vértices no es un invariante. Hay grafos distintos que dan lugar a la misma sucesión gráfica y al reconstruir obtenemos uno con la propiedad adicional de que los vértices de mayor grado son adyacentes.

## 3.6 Grafos de Euler

Sin cambios.

## 3.7 Grafos de Hamilton

Sin cambios.

## 3.8 Grafos bipartidos

Sin cambios

## 3.9 Grafos planos

- Podéis omitir leer el ejemplo de los grafos de los poliedros regulares (ejemplo 6.9.2 del libro, ejemplo 6.8.2 de los apuntes).
- Omitir desde la definición de grafo dual (definición 82 del libro, definición 73 de los apuntes) hasta el final de la sección. No estudiaremos los grafos duales.

## 3.10 Coloración de grafos

- Leer toda la sección y añadir.

Por comodidad introducimos las potencias descendentes de la siguiente forma:

$$x^{\underline{n}} = x(x-1)\cdots(x-n+1)$$

con lo que

$$p(K_n, x) = x^{\underline{n}}$$

Veremos dos algoritmos de cálculo del polinomio cromático de un grafo simple. Ambos se basan en la misma fórmula.

Dado un grafo  $G$ , nos fijamos en un lado  $e$  que una los vértices  $u$  y  $v$ . Entonces el grafo  $G_e$  es el grafo con los mismos vértices que  $G$ , pero al que se le ha quitado el lado  $e$ , y el grafo  $G'_e$  es el grafo que resulta de identificar en  $G_e$  los vértices  $u$  y  $v$ .

**Algoritmo de la suma.**

La fórmula

$$p(G_e, x) = p(G, x) + p(G'_e, x)$$

nos indica que dado un grafo simple  $G_e$  hallar su polinomio cromático se reduce a sumar los polinomios cromáticos de los grafos  $G$  y  $G'_e$ .  $G$  tiene un lado más que  $G_e$  (lado que une dos vértices no adyacentes de  $G_e$ ) y  $G'_e$  tiene un vértice menos que  $G_e$ . En un número finito de pasos todos los grafos obtenidos serán completos que es la condición de parada.

**Algoritmo de la resta.**

La fórmula anterior podemos escribirla como

$$p(G, x) = p(G_e, x) - p(G'_e, x)$$

que nos permite reducir el cálculo del polinomio cromático de un grafo al cálculo de polinomios cromáticos de grafos más pequeños (con menos lados o con menos vértices). La condición de parada es ambigua. Paramos cuando obtengamos grafos de los que conozcamos sus polinomios cromáticos.

Veamos algunos ejemplos.

**Ejemplo 3.3** Para simplificar la notación, vamos a identificar el polinomio cromático de un grafo con el propio grafo.

1. Vamos a calcular el polinomio cromático de un ciclo de longitud 4.

**Algoritmo de la suma.**

$$C_4 = C_4 + C_3 + C_2 = p(K_4, x) + 2p(K_3, x) + p(K_2, x) = x^4 + 2x^3 + x^2$$

**Algoritmo de la resta.**

$$C_4 = P_4 - K_3 = p(P_4, x) - (K_3, x) = x(x-1)^3 - x(x-1)(x-2)$$

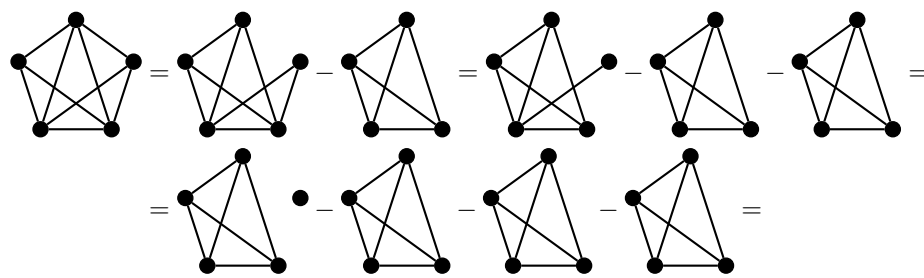
- En este ejemplo el algoritmo de la suma necesita dos pasos y tres descomposiciones en sumas, a cambio nos proporciona el polinomio en potencias descendentes y el número cromático  $\chi_G = 2$  que es el menor exponente descendente. La condición de parada es simple, llegar a grafos completos.
- El algoritmo de la suma necesita un sólo paso, el número cromático habría que calcularlo dando valores naturales al polinomio a partir del 1. Las condiciones de parada son llegar a grafos con polinomios cromáticos conocidos.

2. Vamos a calcular otro polinomio cromático.

**Algoritmo de la suma.**

$$K_5 = K_5 + K_4 = p(K_5, x) + p(K_4, x) = x^5 + x^4$$

Algoritmo de la resta.



$$\begin{aligned}
 &= p(K_4, x)p(K_1, x) - 3p(K_4, x) = x^4x - 3x^4 = x^4(x - 3) = x^4(x - 4 + 1) = \\
 &= x^4(x - 4) + x^4 = x^5 + x^4
 \end{aligned}$$

## 3.11 Árboles

Añadir a la sección del libro. Los árboles etiquetados son totalmente nuevos.

### 3.11.1 Caracterización de los árboles.

Sin cambios.

### 3.11.2 Árboles generadores.

**Definición 3.2** Un árbol generador (de expansión, abarcador) de un grafo conexo es un subgrafo que tiene todos los vértices del grafo y es un árbol.

**Lema 3.1** Sea  $G$  un grafo conexo que contiene un ciclo. Entonces, si quitamos uno de los lados del ciclo el grafo sigue siendo conexo.

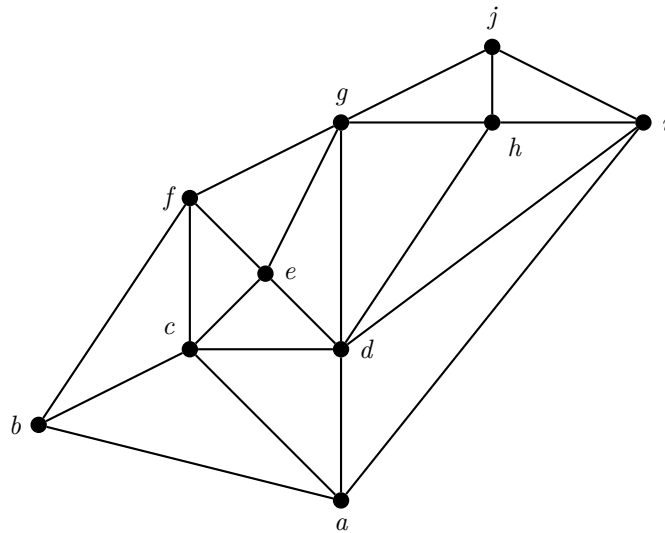
**Proposición 3.2** Todo grafo conexo tiene un árbol generador.

Dado un grafo conexo con  $n$  vértices y  $l$  lados hay dos estrategias para obtener un árbol generador:

**Building-up (constructiva).**

Se eligen  $n - 1$  lados de uno en uno de forma que cada uno no forme ciclos con los anteriormente elegidos.

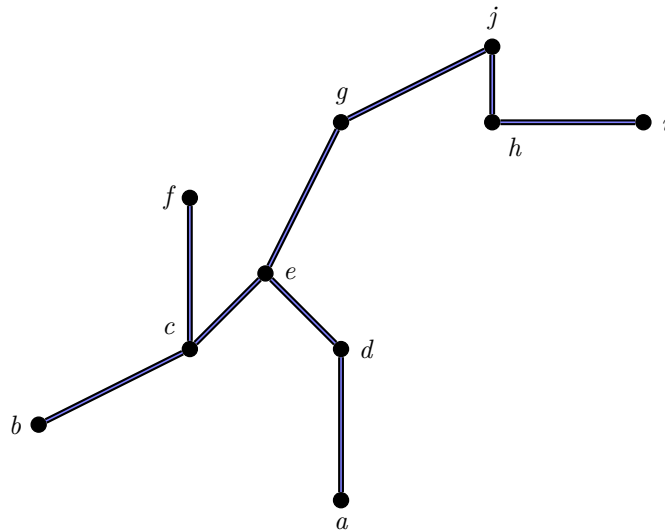
**Ejemplo 3.4** Ordeno los lados del grafo aleatoriamente y elijo los nueve (hay diez vértices) primeros lados de forma que no se formen ciclos.



*Sucesión arbitraria*

*de, cf, hj, ce, ef, bc, eg, hi, bf, dg, ad, gj, ij, fg, di, ai, ac, cd, dh, gh, ab*

***de, cf, hj, ce, ef, bc, eg, hi, bf, dg, ad, gj, ij, fg, di, ai, ac, cd, dh, gh, ab***

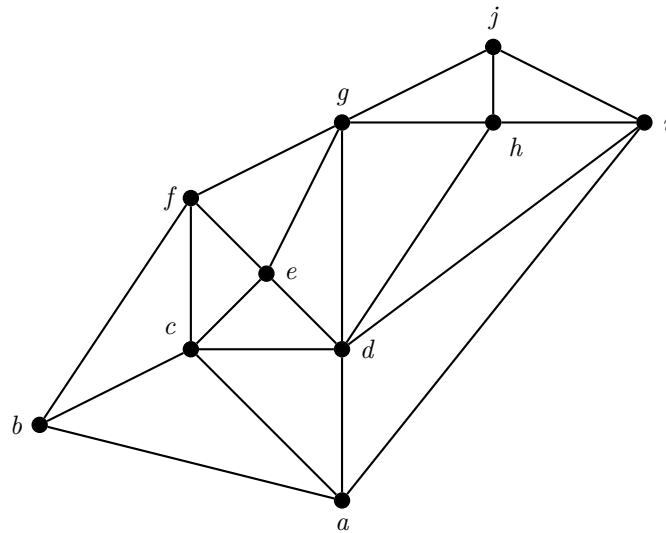


### Cutting-down (destructiva).

Se descartan  $l - (n - 1) = l - n + 1$  lados de uno en uno de forma que cada uno de los que se descartan rompa un ciclo (forme parte de un ciclo) del grafo que va quedando.

**Ejemplo 3.5** *Ordeno los lados del grafo aleatoriamente y elijo los doce (hay 21 lados y hay que dejar 9) primeros lados que rompen ciclos del grafo que va quedando o que no rompan la conexión del grafo que va quedando.*

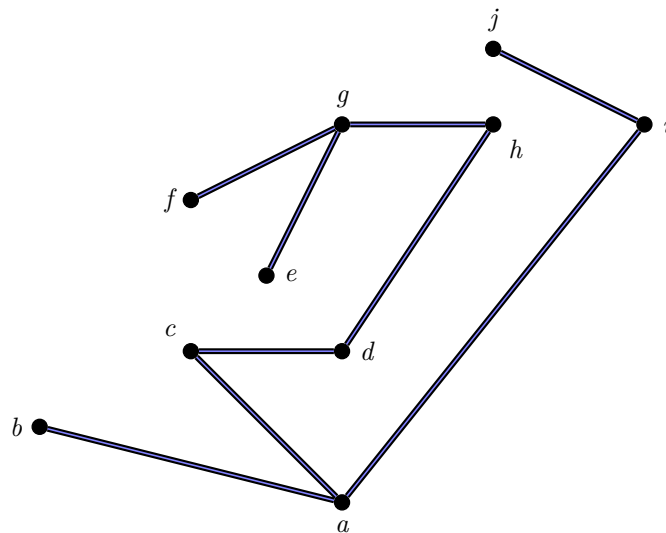




*Sucesión arbitraria*

*de, cf, hj, ce, ef, bc, eg, hi, bf, dg, ad, gj, ij, fg, di, ai, ac, cd, dh, gh, ab*

*de, cf, hj, ce, ef, bc, **eg**, hi, bf, dg, ad, gj, **ij**, **fg**, di, **ai**, **ac**, **cd**, **dh**, **gh**, **ab***



*Observa que es como aplicar la estrategia building-up a la sucesión reversa.*

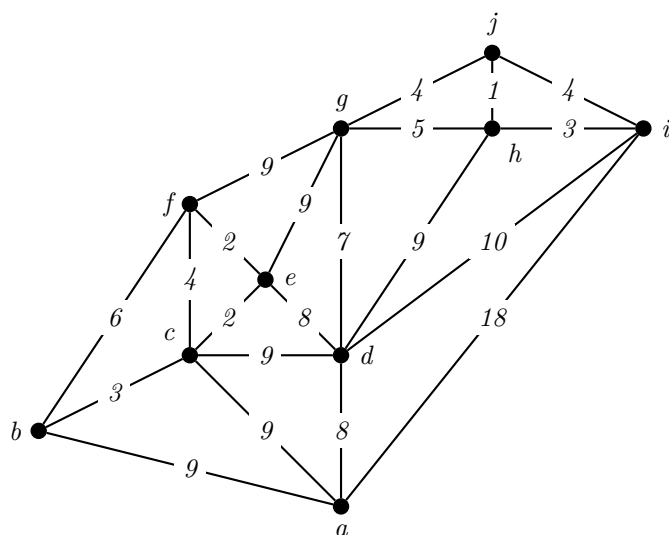
### Algoritmos de Kruskal y Prim.

Se utilizan para dado un grafo ponderado conexo encontrar un árbol generador de peso mínimo.

### Algoritmo de Kruskal (constructivo).

Aplicación de la estrategia building-up a una sucesión de aristas de pesos no decrecientes.

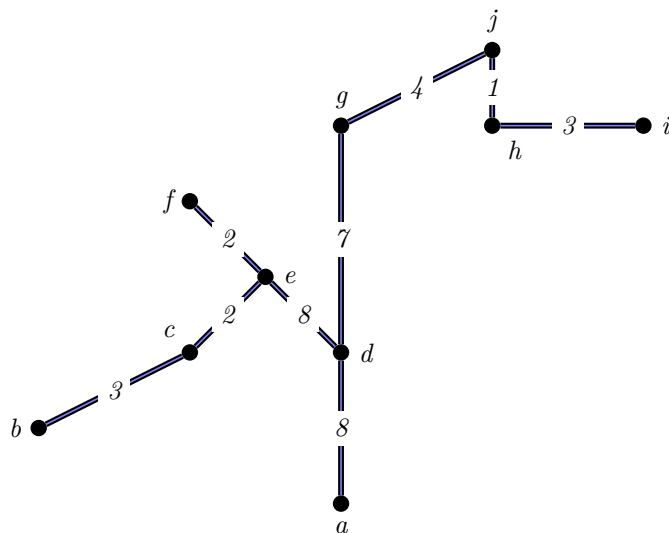
### Ejemplo 3.6



*Sucesión no decreciente*

*hj, ce, ef, bc, hi, cf, gj, ij, gh, bf, dg, ad, de, ab, ac, cd, dh, eg, fg, di, ai*

**hj, ce, ef, bc, hi, cf, gj, ij, gh, bf, dg, ad, de, ab, ac, cd, dh, eg, fg, di, ai**

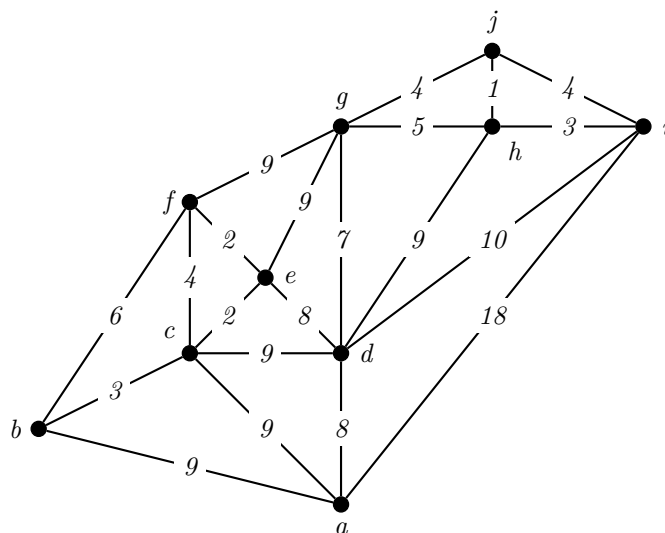


*El peso del árbol es 38.*

**Algoritmo de Kruskal (destructivo).**

Aplicación de la estrategia cutting-down a una sucesión de aristas de pesos no crecientes.

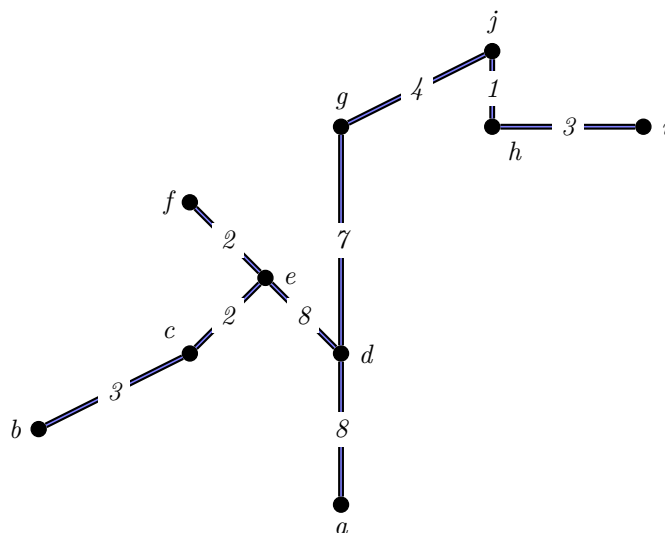
**Ejemplo 3.7**



*Sucesión no creciente de pesos*

*ai, bi, ab, ac, cd, dh, eg, fg, ad, de, dg, bf, gh, cf, gj, ij, bc, hi, ce, ef, hj*

*ai, bi, ab, ac, cd, dh, eg, fg, **ad, de, dg**, bf, gh, cf, **gj**, ij, **bc, hi, ce, ef, hj***



*El peso del árbol es 38.*

### Algoritmo de Prim.

Se trabaja con vértices y lados.

- Se parte de un vértice arbitrario  $v$  que pasa al conjunto de los elegidos  $T = \{v\}$  y  $E = \{\}$
- En cada paso se añade un vértice  $u$  a  $T$  y un lado  $e$  a  $E$  con las condiciones:
  1. que  $u$  sea un vértice que no esté en  $T$ .
  2. que  $u$  sea adyacente mediante el lado  $e$  a uno de  $T$ .
  3. que el lado  $e$  no forme ciclo con los lados de  $E$ .
  4. que  $e$  sea de peso mínimo entre los que cumplen las condiciones anteriores.
- El proceso acaba cuando se han elegido  $n - 1$  lados.

**Ejemplo 3.8** Tomemos el grafo anterior y partamos del vértice  $a$ .

1.	$\{a\}$	$\{\}$	
2.	$\{a, d\}$	$\{ad\}$	8
3.	$\{a, d, g\}$	$\{ad, dg\}$	15
4.	$\{a, d, g, j\}$	$\{ad, dg, gj\}$	19
5.	$\{a, d, g, j, h\}$	$\{ad, dg, gj, jh\}$	20
6.	$\{a, d, g, j, h, i\}$	$\{ad, dg, gj, jh, hi\}$	23
7.	$\{a, d, g, j, h, i, e\}$	$\{ad, dg, gj, jh, hi, de\}$	31
8.	$\{a, d, g, j, h, i, e, c\}$	$\{ad, dg, gj, jh, hi, de, ec\}$	33
9.	$\{a, d, g, j, h, i, e, c, f\}$	$\{ad, dg, gj, jh, hi, de, ec, ef\}$	35
10.	$\{a, d, g, j, h, i, e, c, f, b\}$	$\{ad, dg, gj, jh, hi, de, ec, ef, cb\}$	38

### 3.11.3 Árboles con raíz.

Se define árbol con raíz, hoja, hijos de un nodo, profundidad o nivel de un nodo, altura de un árbol, altura de un nodo y rango de un nodo.

#### Recorrido de árboles con raíz.

Para recorrer árboles  $n$ -arios se tienen en cuenta tres parámetros de un nodo.

- Profundidad o nivel de un nodo es su distancia al nodo raíz.
- Altura de un nodo es la altura o profundidad del subárbol que lo tiene como raíz.
- Rango de un nodo, depende de la representación del árbol y sirve para comparar nodos del mismo nivel. Un nodo tiene menor rango que otro del mismo nivel si está situado a la izquierda del otro. Podríamos darle un valor numérico numerando los nodos de un mismo nivel de izquierda a derecha.

#### Recorrido en Preorden.

Primero se recorre la raíz y después se recorren los subárboles hijos en preorden en orden creciente del rango de sus raíces.

#### Recorrido en Postorden.

Primero se recorren los subárboles hijos en postorder en orden creciente del rango de sus raíces y después se recorre la raíz.

#### Recorrido en Inorden.

Primero se recorre el subárbol hijo de menor rango en inorder después la raíz y por último se recorren en inorder el resto de subárboles hijos en orden creciente del rango de sus raíces. Algunos autores sólo consideran este recorrido para árboles binarios.

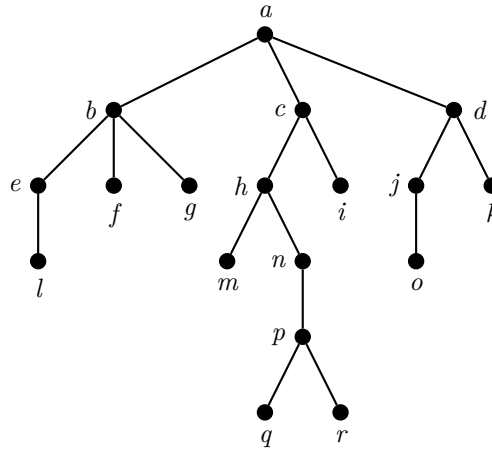
#### Recorrido Top-down.

Se recorren los nodos en orden creciente de profundidad o nivel y dentro de un nivel en orden creciente de rangos. Es el recorrido en anchura usual y es trivial en el sentido de que es el orden de lectura de izquierda a derecha y de arriba hacia abajo.

#### Recorrido Bottom-up.

Es el más complejo y difícil. Se recorren los nodos en orden creciente de altura, dentro de los de misma altura en orden creciente de profundidad y dentro de los de la misma altura y profundidad en orden creciente de rango.

**Ejemplo 3.9** Dado el grafo



Escribe la sucesión de sus nodos al recorrerlo en:

**Preorden:**  $a, b, e, l, f, g, c, h, m, n, p, q, r, i, d, j, o, k$

**Postorden:**  $l, e, f, g, b, m, q, r, p, n, h, i, c, o, j, k, d, a$

**Inorden:**  $l, e, b, f, g, a, m, h, q, p, r, n, c, i, o, j, d, k$

**Top-down:**  $a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r$

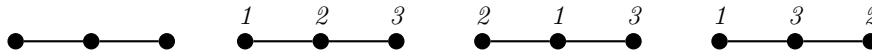
**Bottom-up:**  $\underbrace{f, g, i, k, l, m, o, q, r}, \underbrace{e, j, p}, \underbrace{b, d, n, h, c, a}$

### 3.11.4 Árboles etiquetados.

**Definiciones 3.3** (árbol etiquetado, isomorfismo)

1. Un árbol etiquetado es un árbol con  $n$  vértices en que los vértices tienen como etiquetas los números naturales  $\{1, 2, \dots, n\}$
2. Dos árboles etiquetados son isomorfos si tienen el mismo número de vértices y la aplicación identidad es un isomorfismo de grafos.

**Ejemplo 3.10**



Hay un sólo árbol con tres vértices, pero tres árboles etiquetados.

**Teorema 3.3** El número de árboles etiquetados con  $n$  vértices es  $n^{n-2}$ .

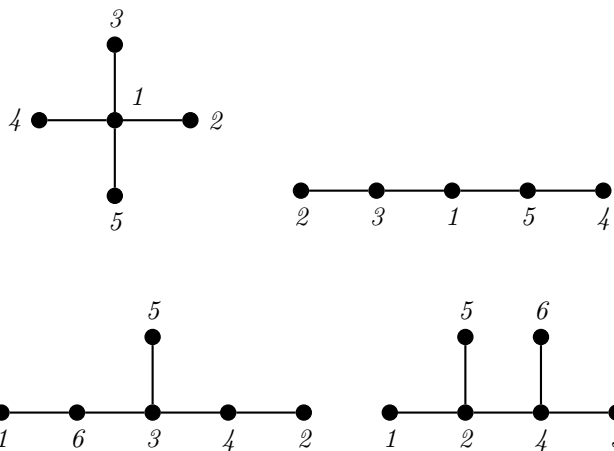
La demostración se puede hacer de varias formas, una de ellas es mediante los códigos de Prüfer.

**Código de Prüfer.**

Suponemos los árboles etiquetados con los números naturales de 1 a  $n$ . Llamamos código de Prüfer de un árbol etiquetado a la sucesión de longitud  $n - 2$  obtenida de la siguiente forma:

1. Se parte de la sucesión vacía (código vacío) y del árbol etiquetado  $T$ .
2. Si  $T$  tiene dos vértices se devuelve el código y fin.
3. Se determina la hoja con menor etiqueta del árbol  $T$  se añade al código la etiqueta del adyacente a la hoja seleccionada y  $T$  pasa a ser el árbol con la hoja seleccionada suprimida.
4. Vuelve a 2 con el nuevo código y el nuevo árbol.

**Ejemplo 3.11** Vamos a hallar los códigos de Prüfer de los árboles:



Aplicando el algoritmo obtenemos los códigos  $(1, 1, 1)$ ,  $(3, 1, 5)$ ,  $(6, 4, 3, 3)$  y  $(2, 4, 2, 4)$ .

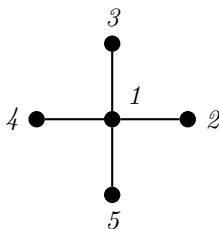
### Generación de un árbol etiquetado con un código dado.

Un código de Prüfer es una sucesión finita de  $n - 2$  números comprendidos entre 1 y  $n$ .

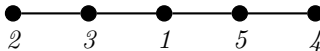
1. Se parte del código  $c$  de longitud  $n - 2$ , del conjunto  $V = \{1, 2, \dots, n\}$  y  $T$  el grafo vacío con  $n$  vértices etiquetado.
2. Si  $c$  es vacío y  $V$  tiene dos vértices se pone un lado entre los vértices de  $V$ , se devuelve  $T$  y se acaba.
3. Se determina el menor número de  $V$  que no está en  $c$  se añade a  $T$  un lado entre ese menor número y el primer elemento del código. Se quita de  $V$  el elemento seleccionado y de  $c$  su primer elemento.
4. Vuelve a 2 con el nuevo código, el nuevo conjunto  $V$  y el nuevo grafo  $T$ .

### Ejemplos 3.12

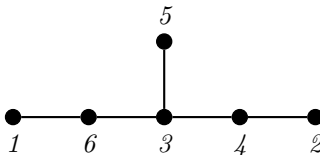
1. Vamos a hallar el árbol con código de Prüfer  $(1, 1, 1)$ . Tiene 5 vértices y 4 lados. Aplicando el algoritmo los lados son  $1-2$ ,  $1-3$ ,  $1-4$  y  $1-5$ , resultando



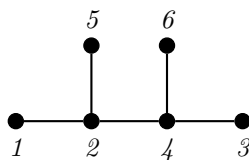
2. Vamos a hallar el árbol con código de Prüfer  $(3, 1, 5)$ . Tiene 5 vértices y 4 lados. Aplicando el algoritmo los lados son  $3-2$ ,  $1-3$ ,  $5-1$  y  $4-5$ , resultando



3. Vamos a hallar el árbol con código de Prüfer  $(6, 4, 3, 3)$ . Tiene 6 vértices y 5 lados. Aplicando el algoritmo los lados son  $6-1$ ,  $4-2$ ,  $3-4$ ,  $3-5$  y  $3-6$ , resultando



4. Vamos a hallar el árbol con código de Prüfer  $(2, 4, 2, 4)$ . Tiene 6 vértices y 5 lados. Aplicando el algoritmo los lados son  $2-1$ ,  $4-3$ ,  $2-5$ ,  $4-2$  y  $4-6$ , resultando



## Posibles preguntas para el examen final

- Ejercicio en que haya que utilizar el algoritmo de demolición y algoritmo de reconstrucción.
- Ejercicio de cálculo de un polinomio cromático.
- Ejercicio en que haya que utilizar alguno de los algoritmos de Kruskal o el algoritmo de Prim.
- Ejercicio de recorrido de un árbol.
- Ejercicio de a partir de un árbol etiquetado obtener el código de Prüfer y viceversa.