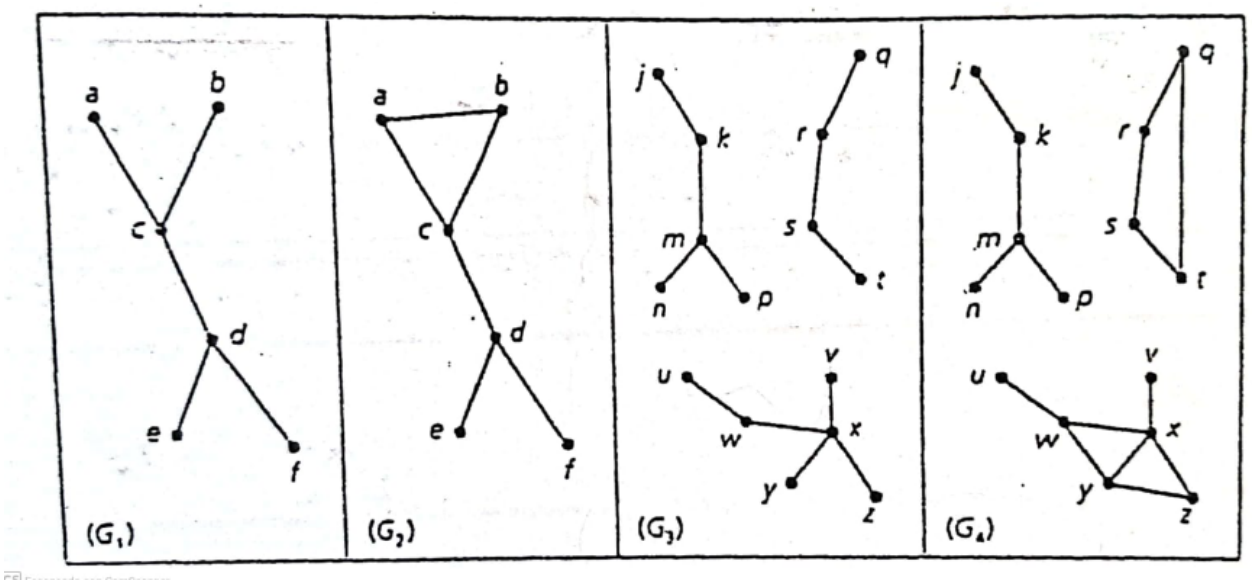
Algoritmos Sobre Grafos

Árboles

Sea $G = (V, E)$ un grafo sin lazos.

El grafo G es un *árbol* si G es conexo y no contiene ciclos.

Ejemplo:



- G_1 es un árbol.
- G_2 no es un árbol. ¿Por qué?
- G_3 no es un árbol. ¿Por qué?
- Cada componente conexa de G_3 es un árbol.....G_3 es un **bosque.**

Sea $G = (V, E)$ un grafo sin lazos, no conexo.

El grafo G es un *bosque* si cada componente conexa de G es un árbol.

OBJETIVO:

Queremos estudiar dos tipos de árboles y bosques: los árboles recubridores y los bosques recubridores.

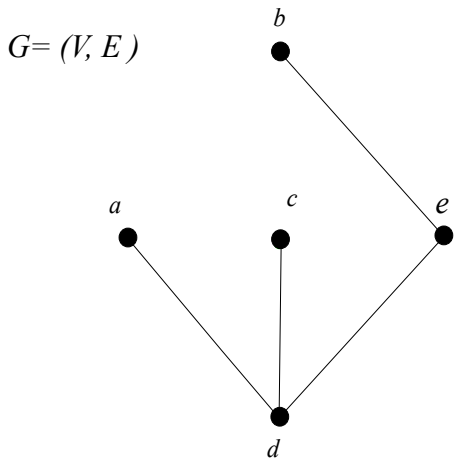
Pero antes necesitamos el concepto de subgrafo.

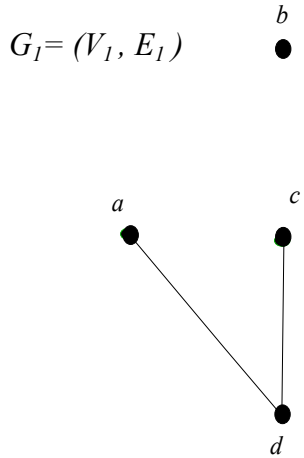
Concepto de subgrafos

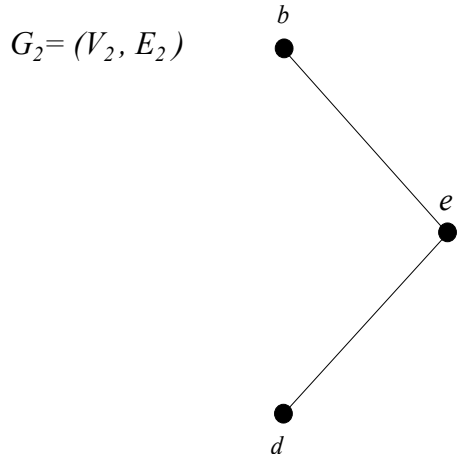
Si $G = (V, E)$ es un grafo (o digrafo), entonces $G_1 = (V_1, E_1)$ es un *subgrafo de* G si:

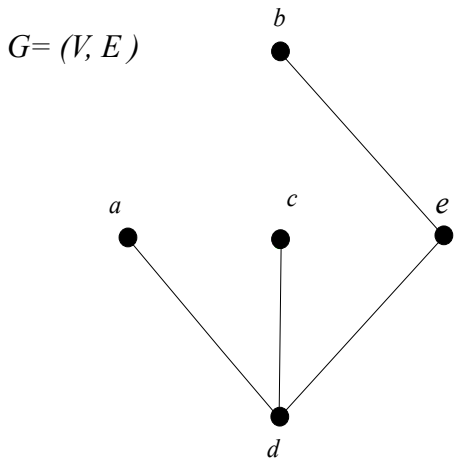
- $V_1 \subseteq V$, con $V_1 \neq \emptyset$ y
- $E_1 \subseteq E$, donde cada arista de E_1 es incidente con los vértices de V_1

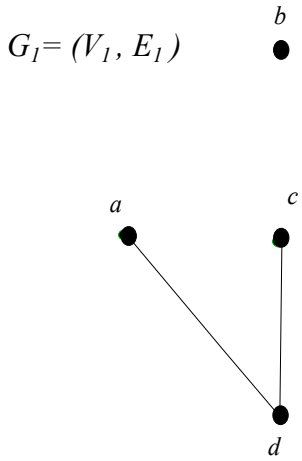
Ejemplos: La figura nos muestra dos subgrafos distintos G_1 y G_2 del mismo grafo G

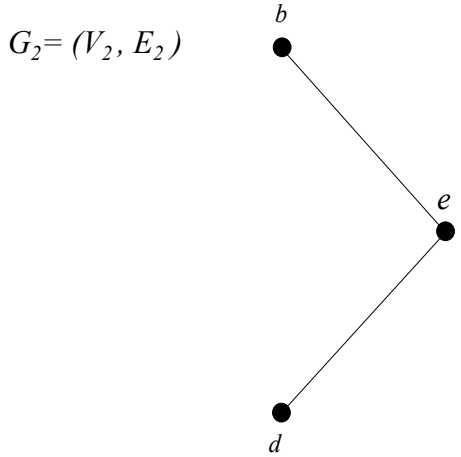












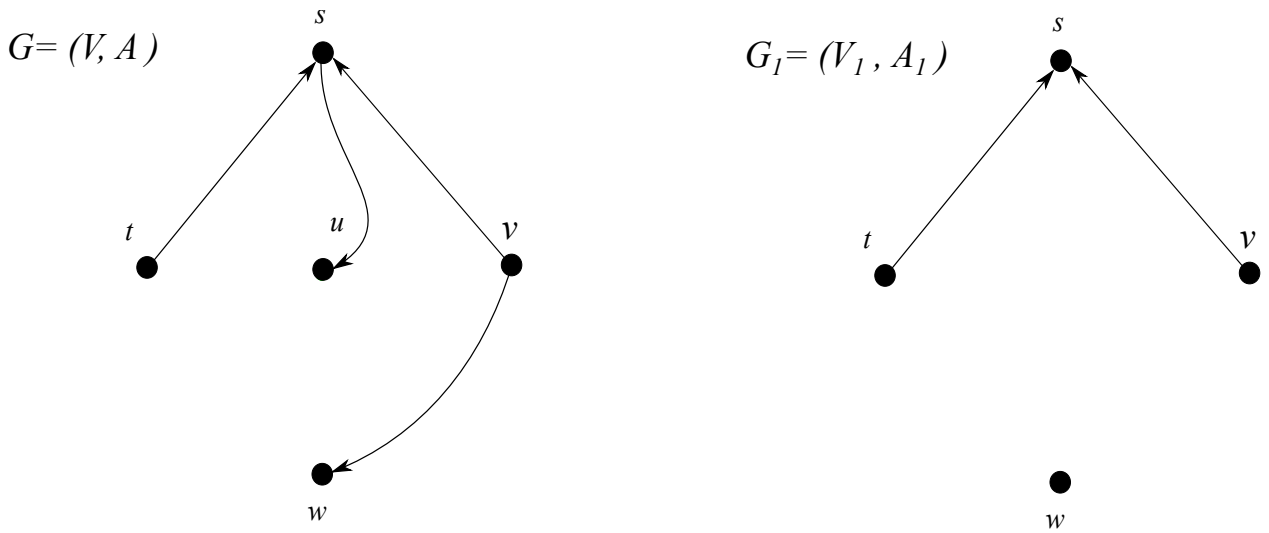
- $V_1 =$
- $E_1 =$

Observación: V_1 esta contenido en V pero es distinto de V.

- $V_2 =$
- $E_2 =$

Concepto de subgrafos

Ejemplos: La figura nos muestra el digrafo G y un subgrafo G_1.

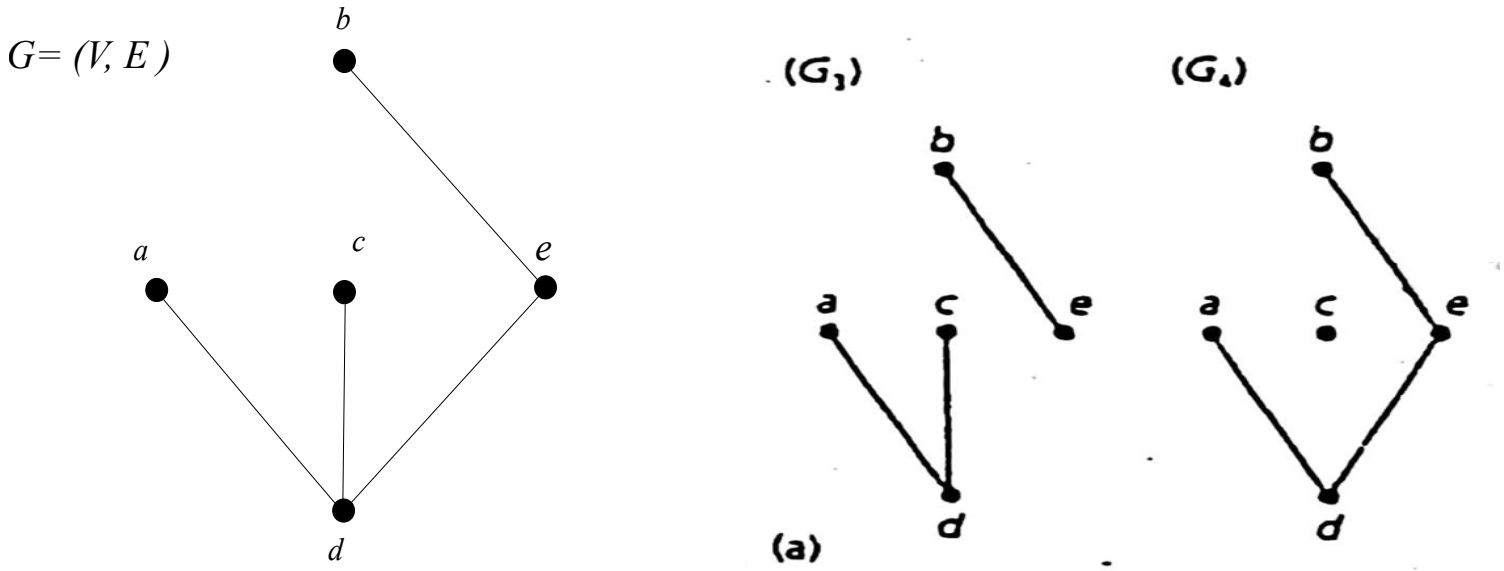


- $V_1 =$
- $A_1 =$

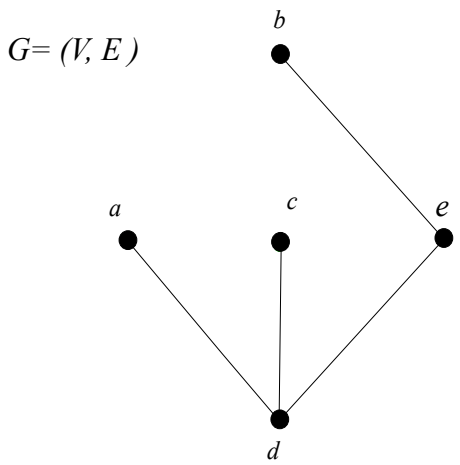
Tipo especial de subgrafo: Subgrafo recubridor

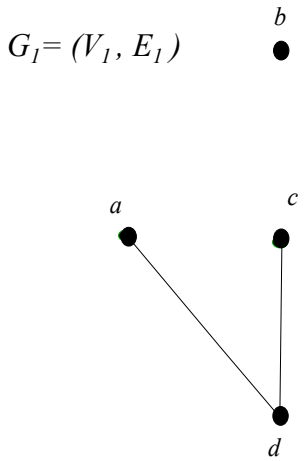
Dado un grafo $G = (V, E)$ (o digrafo), y $G_1 = (V_1, E_1)$ un subgrafo de G.
Si $V = V_1$, entonces G_1 es un *subgrafo recubridor de G*.

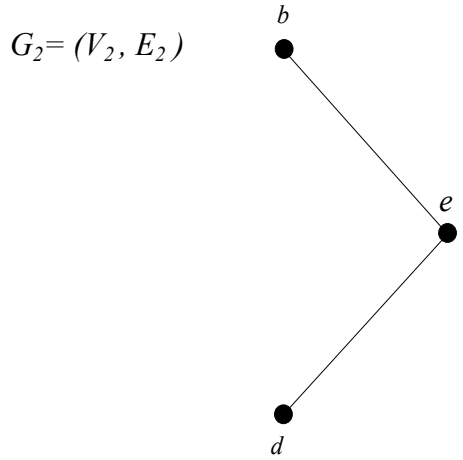
Ejemplo: Los subgragos G_3 y G_4 son subgrafos recubridores de G.



Tipo especial de subgrafo: Subgrafo recubridor







Ni G_1 ni G_2 son subgrafos recubridores de G. ¿ Por qué?

Árboles y bosques recubridores

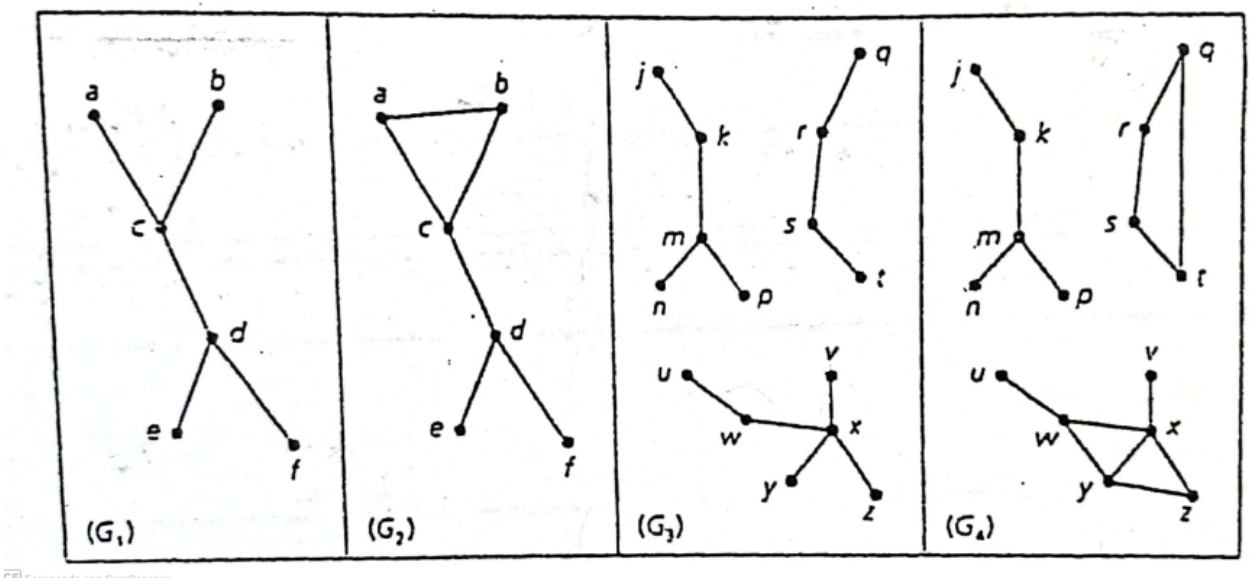
Un *árbol recubridor* de un grafo conexo G es un subgrafo recubridor de G que es un árbol.

Un *bosque recubridor* de un grafo G es un subgrafo recubridor de G que es un bosque.

Podemos pensar en un árbol recubridor como aquel que proporciona una conectividad minimal para el grafo, como un esqueleto minimal que une los vértices.

Árboles y bosques recubridores

Ejemplo:



- G_1 es un árbol recubridor de G_2. ¿ Por qué?
- G_3 es un bosque recubridor para el grafo G_4. ¿ Por qué?

Caracterización de los grafos árboles

Teorema:

Sea $G = (V, E)$ un grafo no dirigido y sin lazos. Las siguientes proposiciones son equivalentes:

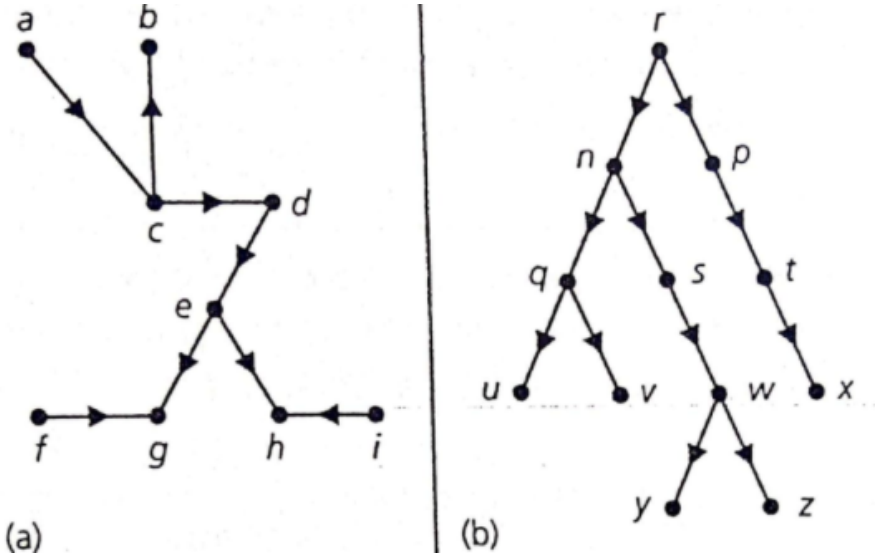
1. G es un árbol.
2. G es conexo, pero si se elimina cualquiera de sus aristas, G quedará desconectado en dos subgrafos que son árboles.
3. G no contiene ciclos y $|V| = |E| + 1$.
4. G es conexo y $|V| = |E| + 1$.
5. G no contiene ciclos y si $a, b \in V$ con $ab \notin E$, entonces el grafo que se obtiene de agregar la arista ab a G tiene precisamente un cilo.

Árboles con raíz

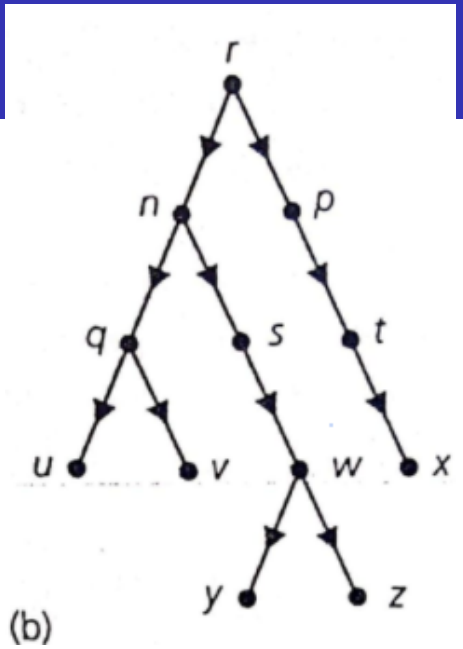
Si G es un grafo dirigido, entones G es *árbol dirigido* si el grafo no dirigido asociado (*grafo subyacente*) es un árbol.

Si G es un árbol dirigido, G es un *árbol con raíz* si existe un único vértice r en G, llamado *raíz* tal que el grado de entrada de r es nulo, $g_e(r) = 0$ y para todos los demás vértices v, el grado de entrada de v es $g_e(v) = 1$.

Ejemplo: La figura (a) es un árbol dirigido pero sin raíz, el árbol de la parte (b) tiene r como raíz.



Árboles con raíz



- El *nivel de un vértice* v es la longitud de un $rv-$ camino.

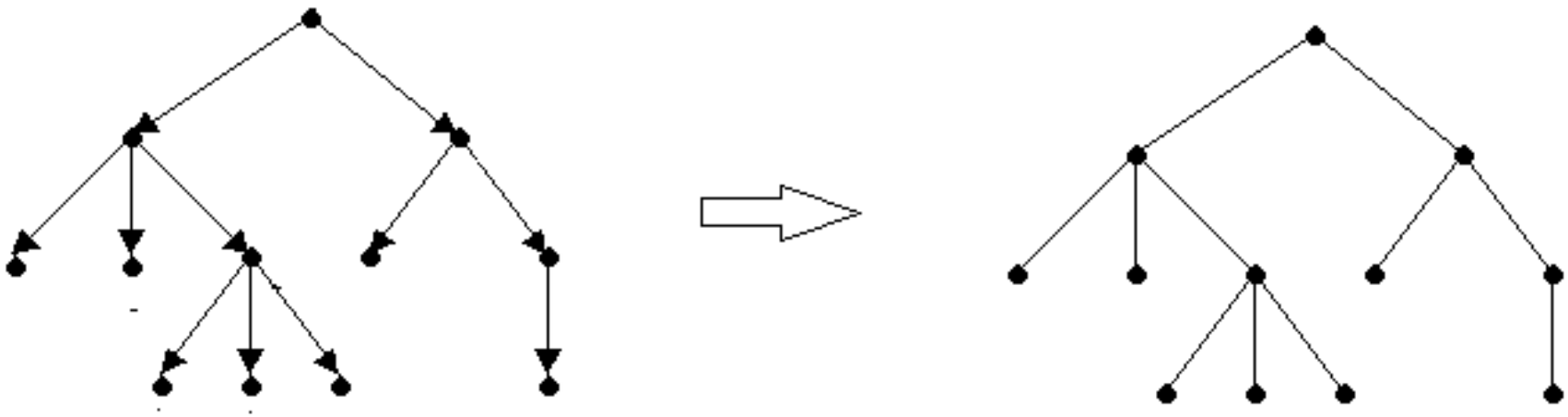
Ejemplo: El camino desde la raíz r hasta s es de longitud 2, por lo que s esta a nivel 2 del árbol.

- Decimos que s es un *hijo* de n y n es el *padre* de s.
- Dos vértices con un padre en común son *hermanos*.
- w, y, z son *descendientes* de s, n y r.
- s, n y r son *ascendientes* de w, y, z.

Observación:

En un árbol dirigido con raíz siempre las flechas van de arriba hacia abajo, por eso se lo grafica directamente sin indicar las flechas y respetando los diferentes niveles:

Ejemplo:



Búsqueda de un árbol recubridor en un grafo G

Dado $G = (V, E)$ un grafo no dirigido, conexo, queremos buscar un árbol recubridor T del grafo G. Para ello veremos dos algoritmos:

- Algoritmo de **búsqueda en profundidad** de un árbol recubrido T en G.
- Algoritmo de **búsqueda en anchura** de un árbol recubrido T en G.

Algoritmo de búsqueda en profundidad de un árbol recubridor T en G

- Sea $G = (V, E)$ conexo, sin lazos, no dirigido, $|V| = n$ y donde los vértices están ordenados $v_1, v_2, \cdots, v_n$.
- Buscaremos un árbol recubridor con raíz, donde la raíz será v_1.
- v será una variable, indica el vértice que se está analizando.

Algoritmo de búsqueda en profundidad

Paso 1: Se asigna v_1 a la variable v y se inicializa T como el árbol que consta solamente de este vértice. (El vértice v_1 será la raíz del árbol recubridor que se va a desarrollar.)

Paso 2: Seleccionamos el subíndice más pequeño i, $2 \leq i \leq n$, tal que $\{v, v_i\} \in E$ y v_i no ha sido visitado todavía.

Si no se encuentra tal subíndice, entonces se va al paso 3. En caso contrario, se hace lo siguiente: (1) Añadimos la arista $\{v, v_i\}$ al árbol T; (2) asignamos v_i a v; y (3) regresamos al paso 2.

Paso 3: Si $v = v_1$, el árbol T es el árbol recubridor (ordenado, con raíz) del orden dado.

Paso 4: Si $v \neq v_1$, retrocedemos desde v. Si u es el padre del vértice asignado a v en T, entonces asignamos u a v y regresamos al paso 2.

Ejemplo:

Aplicaremos ahora este algoritmo al grafo $G = (V, E)$ que se muestra en la figura 12.21(a). En este caso, el orden de los vértices es alfabético: $a, b, c, d, e, f, g, h, i, j$.

Asignamos primero el vértice a a la variable v e inicializamos T sólo con el vértice a (la raíz). En el paso 2, vemos que el vértice b es el primer vértice tal que $\{a, b\} \in E$ que no ha sido visitado todavía, por lo que agregamos la arista $\{a, b\}$ a T, asignamos b a v y regresamos al paso 2.

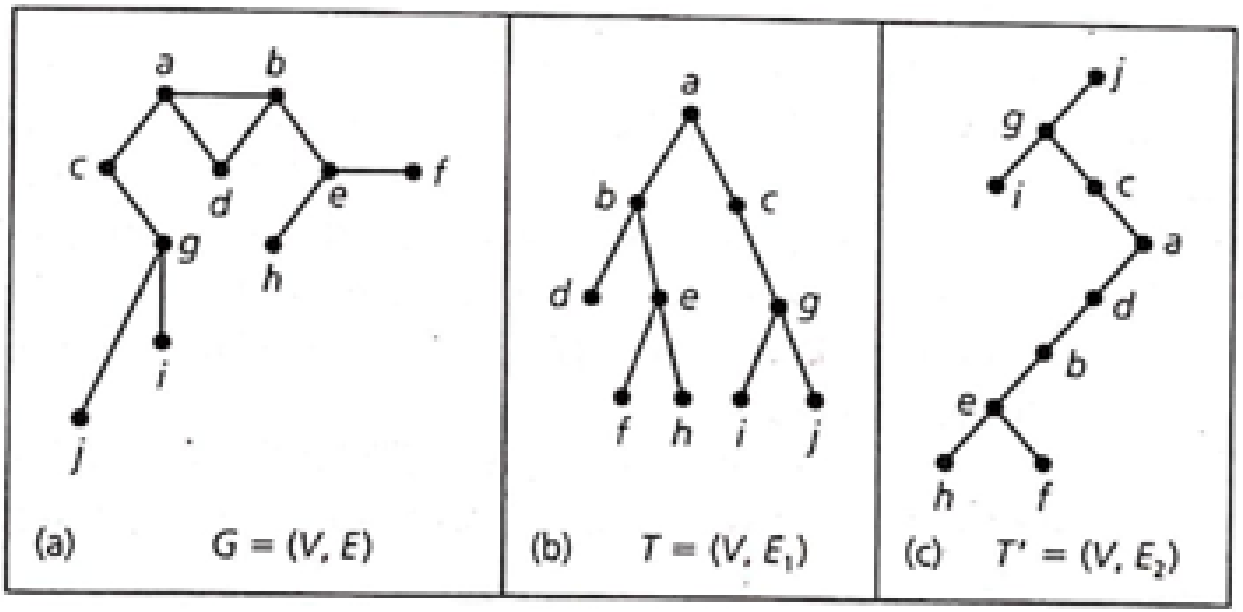
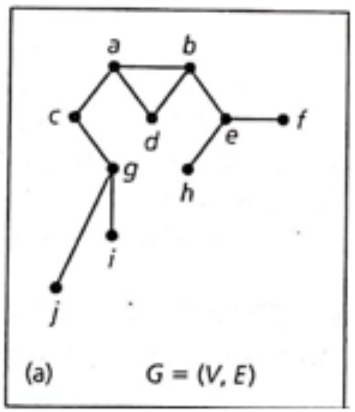
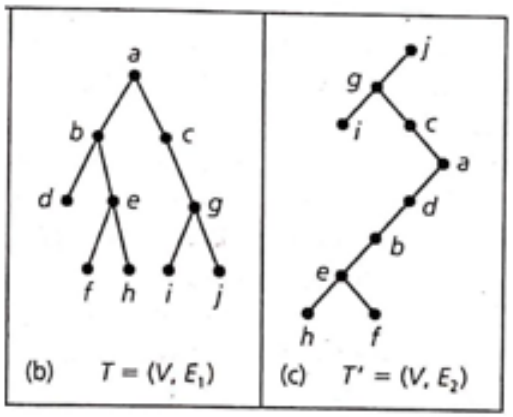


Figura 12.21

Ejemplo (continuación):





- Para $v = b$, vemos que el primer vértice (no visitado todavía) y que proporciona una arista al árbol recubridor es d. En consecuencia, agregamos la arista $\{b, d\}$ a T, asignamos d a v y volvemos al paso 2.
- Sin embargo, esta vez no existe un nuevo vértice que podamos obtener de d, puesto que los vértices a y b ya han sido visitados, por lo que vamos al paso 3. Pero en este caso, el valor de v es d, no a, así que vamos al paso 4. Retrocedemos desde d y asignamos el vértice b a v y después regresamos al paso 2. En este momento vemos que podemos añadir la arista $\{b, e\}$ a T.
- Seguimos con el proceso y añadimos a continuación las aristas $\{e, f\}$ y $\{e, h\}$. Pero ahora, como el vértice h está asignado a v, debemos retroceder de h a e a b a a. Cuando v tiene asignado el vértice a por segunda vez, se obtiene la nueva arista $\{a, c\}$. Después agregamos las aristas $\{c, g\}$, $\{g, i\}$ y $\{g, j\}$. En este momento, hemos visitado todos los vértices de G, por lo que retrocedemos de j a g a c a a. Con $v = a$ de nuevo, regresamos al paso 2 y de ahí al paso 3, donde termina el proceso.

- El árbol resultante $T = (V, E_1)$ aparece en la parte (b) de la figura 12.21. La parte (c) de la figura muestra el árbol T' que resulta del ordenamiento de los vértices: $j, i, h, g, f, e, d, c, b, a$.

OBSERVACIÓN: En el algoritmo, importa como llamamos a los vértices. Si cambiamos el orden/nombres de los vértices, vamos a encontrar otro árbol recubridor.

Algoritmo de búsqueda en anchura de un árbol recubridor T en G

- Sea $G = (V, E)$ conexo, sin lazos, no dirigido, $|V| = n$ y donde los vértices están ordenados $v_1, v_2, \cdots, v_n$.
- Aquí designamos un vértice como la raíz y recorremos todos los vértices adyacentes a la raíz (de ahí que la búsqueda es en anchura).

Algoritmo de búsqueda en anchura

Paso 1: Insertamos el vértice v_1 en la cola Q e inicializamos T como el árbol formado por este único vértice v_1 (la raíz de la versión final de T).

Paso 2: Eliminamos los vértices del frente de Q. Al eliminar un vértice v, consideramos v_i para cada $2 \leq i \leq n$. Si la arista $\{v, v_i\} \in E$ y v_i no ha sido visitado, agregamos la arista a T. Si examinamos *todos* los vértices que ya estaban en Q y no obtenemos aristas nuevas, el árbol T (generado hasta ese momento) es el árbol recubridor (ordenado con raíz) del orden dado.

Paso 3: Insertamos los vértices adyacentes a cada v (del paso 2) en el final de la cola Q, según el orden en que fueron visitados por primera vez. Después regresamos al paso 2.

Ejemplo:

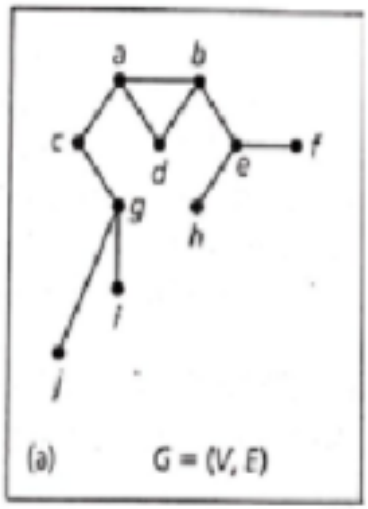


Figura 12.21

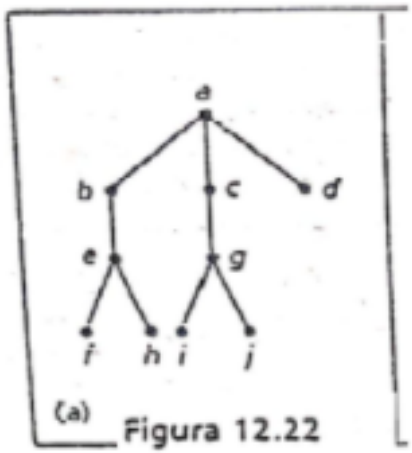


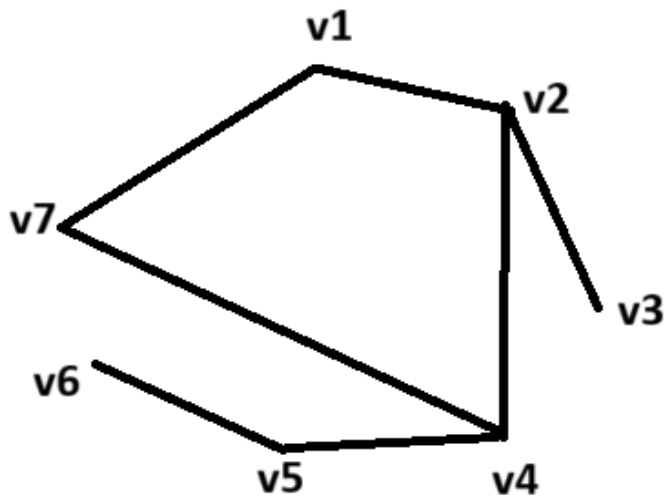
Figura 12.22

Utilizaremos el grafo de la figura 12.21(a) con el orden $a, b, c, d, e, f, g, h, i, j$ para ilustrar el uso del algoritmo de búsqueda en anchura.

- Partimos del vértice a. Insertamos a en Q e inicializamos T con este vértice (la raíz del árbol resultante).
- En el paso 2 eliminamos a de Q y visitamos los vértices adyacentes a él: b, c, d. (Estos vértices no han sido visitados previamente.) Esto permite añadir las aristas $\{a, b\}, \{a, c\}$ y $\{a, d\}$.
- En el paso 3, insertamos b, c, d (en este orden) en Q y regresamos al paso 2. Ahora eliminamos estos vértices de Q y visitamos los vértices adyacentes a ellos (no visitados antes) de acuerdo con el orden dado de los vértices de G. De aquí obtenemos los dos nuevos vértices e, g y en las aristas $\{b, e\}, \{c, g\}$ que agregamos a T.
- Después vamos al paso 3 e insertamos e, g en Q. Regresamos al paso 2, eliminamos cada uno de estos vértices de Q y encontramos, en orden, los nuevos vértices (no visitados previamente) f, h e i, j. Esto nos permite añadir las aristas $\{e, f\}, \{e, h\}$ y $\{g, i\}, \{g, j\}$ al árbol T.
- De nuevo regresamos al paso 3, donde insertamos los vértices f, h, i, j en Q. Pero ahora, cuando vamos al paso 2 y eliminamos los vértices de Q, no encontramos vértices nuevos (no visitados previamente). En consecuencia, el árbol T de la figura 12.22(a) es el árbol **recubridor en anchura de G, para el orden dado.**

Ejemplo:

Aplicar ambos algoritmos sobre G para buscar un recubridor.



Grafo ponderado y distancia entre dos vértices

En esta sección presentaremos técnicas que optimizan algunos resultados analizando digrafo que tienen un número real positivo asociada a cada arco.

Analizaremos en esta sección los problemas de determinar la distancia más corta entre un vértice dado v_0 y cada uno de los demás vértices de un digrafo conexo sin lazos...

Grafo ponderado

Sea $G = (V, A)$ un digrafo conexo sin lazos.

A cada arco $e = (a, b)$ de este digrafo le asociamos un número real positivo llamado *peso de e*, que denotaremos $p(e)$ o $p(a, b)$. Si $a, b \in V$ pero $(a, b) \notin A$, definimos $p(a, b) = \infty$.

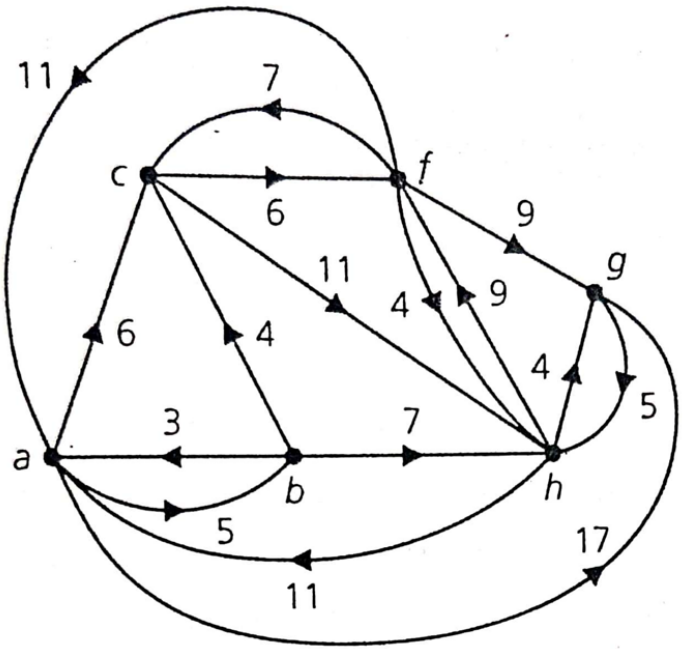
Cuando demos un digrafo G con las asignaciones de peso aquí descripas, diremos que el grafo es un *grafo ponderado*.

Representación del peso: Para cualqueir $e = (a, b) \in A$, $p(e)$ podría representar:

- la longitud de una ruta de a a b.
- el tiempo que tarda recorrer la ruta de a a b.
- el costo del viaje de a a b por esta ruta.

Ejemplo: En el siguiente grafo ponderado, se representan:

- Las ciudades, por medio de un vértice.
- Las rutas de viaje, por medio de un arco.
- El tiempo de un vuelo directo de la ciudad x a la cuidad y, por medio del peso en cada arista (x, y).



$p(c, f) =$

$p(f, c) =$

$p(f, a) =$

$p(a, f) =$

Observaciones:

* Hay situaciones en las que $p(x, y) \neq p(y, x)$, esto puede ser debido a la dirección del viento.
* Como (c, g) y $(g, c) \notin A$, resulta $p(c, g) = p(g, c) = \infty$.
* Tenemos $p(a, f) = \infty$ mientras que $p(f, a) = 11$.

Distancia entre dos vértices

Dado un digrafo ponderado $G = (V, A)$, para cada arco $e = (a, b) \in A$ interpretamos $p(e)$ como la longitud de una ruta directa de a a b.

Para cualesquiera dos vértices $a, b \in V$ con $a \neq b$, definimos:

- $d(a, b)$ =la distancia más corta de a a b, es decir, la longitud del camino más corto de a a b cuando existe un camino de a a b.
- $d(a, b) = \infty$, cuando no existe un camino de a a b.

Para cualquier $a \in V$, definimos $d(a, a) = 0$.

5.2 Algoritmo de Dijkstra: Introducción

Un problema muy recurrente es encontrar el camino más corto entre dos vértices dados (es decir, un camino que tiene la longitud mínima entre todas los posibles caminos entre esos dos vértices.)
Edsger W. Dijkstra (1930-2002) fue un científico de la computación holandés que ideó un algoritmo que resuelve el problema de la ruta más corta de forma óptima.

Edsger Wybe Dijkstra

Además, fue de los primeros en proponer la programación como una ciencia. Ganador del premio Turing en 1972. El premio Turing es "el Nobel de la programación". Poco después de su muerte recibió también el premio de la ACM (Association for Computing Machinery). El premio pasó a llamarse Premio Dijkstra el siguiente año en su honor.

Algoritmo de Dijkstra: Introducción

Este algoritmo encuentra la longitud de una ruta más corta del vértice v_0 al vértice v en un grafo $G = (V, E)$ ponderado y conexo.

El peso del arco (a, b) es $p(a, b) > 0$ (es decir, requerimos que los pesos sean siempre positivos).

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

El objetivo es encontrar la ruta de longitud mínima, el algoritmo Dijkstra funciona asignando **distancias** a los vértices, que siempre son relativas al vértice inicial v_0.

Objetivo:

Dado un digrafo ponderado, G=(V,A) donde si a, $b \in V$:

$$p(a,b) \in \mathbb{R}^+ \ \ si \ \ (a,b) \in A$$

$$p(a,b) = \infty \ \ si \ \ (a,b) \notin A$$

Fijemos ahora $v_0 \in V$, entonces para cualquier vértice $v \in V$ queremos:

1. $d(v_0, v)$= longitud del camino más corto de v_0 a v.
2. un camino simple dirigido de v_0 a v si $d(v_0, v)$ es finito.

Algoritmo de Dijkstra

Sea $G = (V, A)$ un digrafo ponderado y $v_o \in V$.

En el algoritmo de Dijkstra trabajaremos con dos subconjuntos de vértices:

- $S \subseteq V$ tal que $v_0 \in S$.

  S será el conjunto de vértices para los cuales ya sé $d(v_0, v)$ donde v son los vértices pertenecientes al conjunto S.

- $\bar{S} = V - S$ es el conjunto de vértices del grafo G que no están en S.

Y trabajaremos con etiquetas $(L(z), v)$ donde $L(z)$ nos dara información sobre la distancia y la segunda componente de la etiqueta es v, un vértice de S que hace $L(z)$ mínima.

Algoritmo de Dijkstra

INICIO:

- $S = \{v_0\}$.
- Etiqueto:
  - al vértice v_0 con la etiqueta $(0, -)$.
  - a los vértices $z \neq v_0$ con la etiqueta

$$\begin{cases} (L(z), v_o) \quad donde \quad L(z) = p(v_0, z) & si \quad p(v_o, z) < \infty \\ (\infty, -) & sino \end{cases}$$

- Si todos los vértices $\bar{S}$ tiene etiqueta $(\infty, -)$, **terminé**.

Sino:

1. Seleccionar un vértice v tal que $v \notin S$ y $L(v) = min\{L(z)/\ \ z \notin S\}$

   Notar que v es el más cercano a v_0

2. $S := S \cup \{v\}$.

3. Para todo vértice $z \notin S$, actualizo sus etiquetas

$$L(z) := min\{L(z), L(v) + p(v, z)\}$$

   Etiqueta: Para todo vértice $z \notin S$, $(L(z), v)$ donde $v \in S$ que hace $L(z)$ mínimo.

4. Si $S = V$ paro (**terminé**), **sino** voy al paso 1.

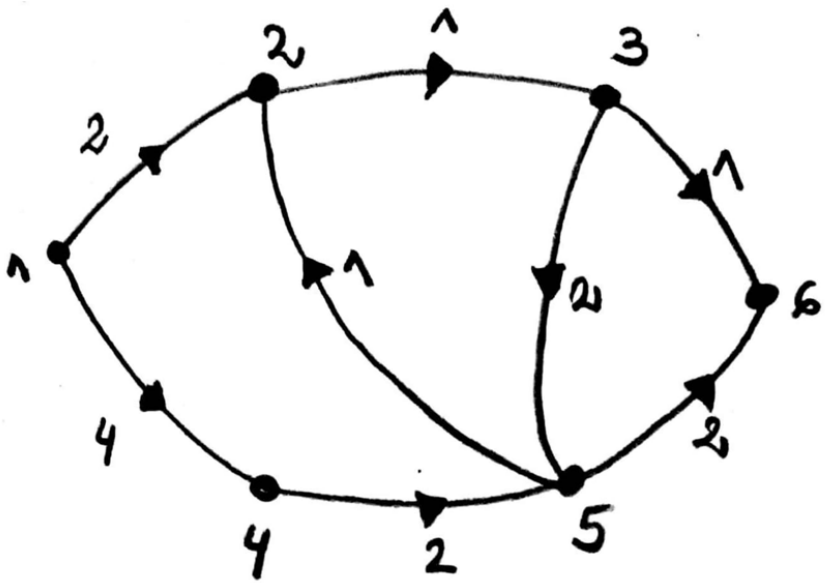
Observación

El algoritmo de Dijkstra se puede aplicar también para grafos (no dirigidos) considerando para todo par de vértices $a, b \in V$ tal que $ab \in E$ (sea una arista del grafo) que

$$p(a, b) = p(b, a).$$

Ejemplo 1: Algoritmo de Dijkstra

1) Considerar el siguiente digrafo y hallar el camino más corto del nodo 1 al nodo 6 y su distancia, aplicando el algoritmo de Dijkstra.



Ejemplo 1: (Continuación)

i	S	1	2	3	4	5	6	Elección de π
Inicio	$\pi_s = 1$	$(0,-)$	$(2,1)$	$(\infty,-)$	$(4,1)$	$(\infty,-)$	$(\infty,-)$	$2 = \pi$
$i=1$	1, 2			$(3,2)$	$(4,1)$	$(\infty,-)$	$(\infty,-)$	$\pi = 3$
$i=2$	1, 2, 3				$(4,1)$	$(5,3)$	$(4,3)$	$\pi = 6$
	1, 2, 3, 6.							

↳ como el $6 \in S$ y quiero $d(1,6)$ ¡Paro!

$i=1$ Actualizo etiqueta $\forall z \notin S$ $L(z) = \min\{L(z); L(\pi) + p(\pi, z)\}$

$\pi = 2$

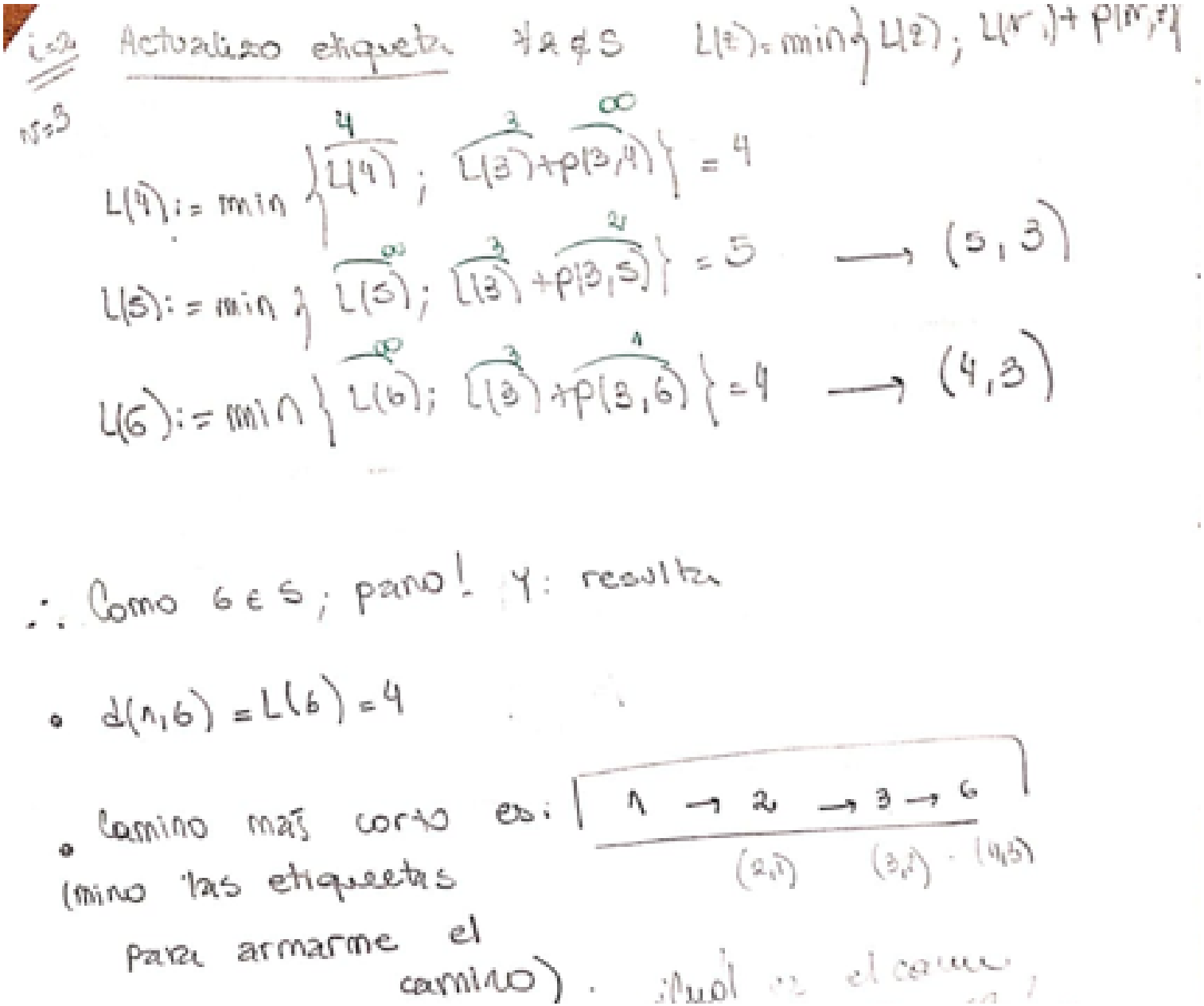
$L(3) = \min\{\overset{\infty}{L(3)}; \overset{2}{L(2)} + \overset{1}{p(2,3)}\} = 3 \longrightarrow (3,2)$

$L(4) := \min\{\overset{4}{L(4)}, \overset{2}{L(2)} + \overset{\infty}{p(2,4)}\} = 4$

$L(5) := \min\{\overset{\infty}{L(5)}, \overset{2}{L(2)} + \overset{\infty}{p(2,5)}\} = \infty$

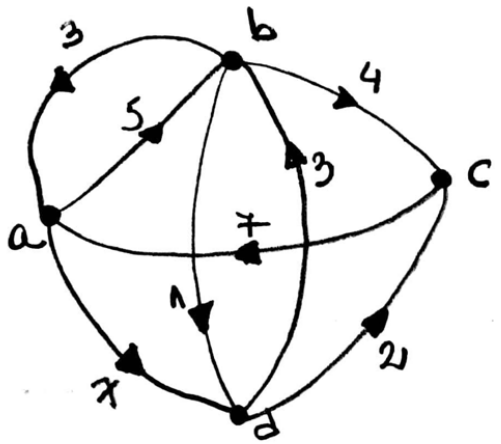
$L(6) := \min\{\overset{\infty}{L(6)}, \overset{2}{L(2)} + \overset{\infty}{p(6,5)}\} = \infty$

Ejemplo 1: (Continuación)

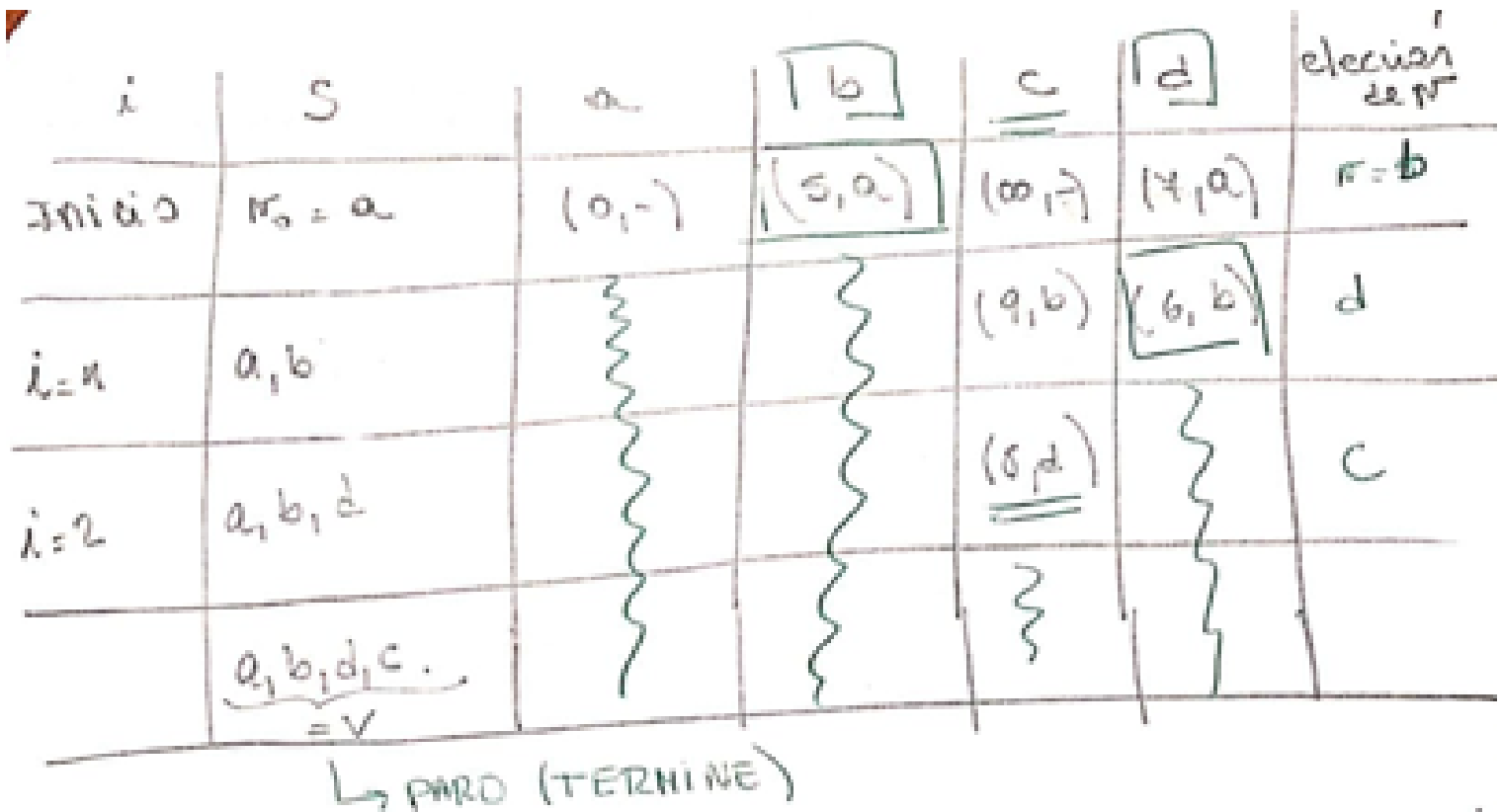


Ejemplo 2: Algoritmo de Dijkstra

2) Hallar la distancia y el camino más corto del nodo *a* a los demás nodos del siguiente digrafo.



Ejemplo 2: (Continuación)



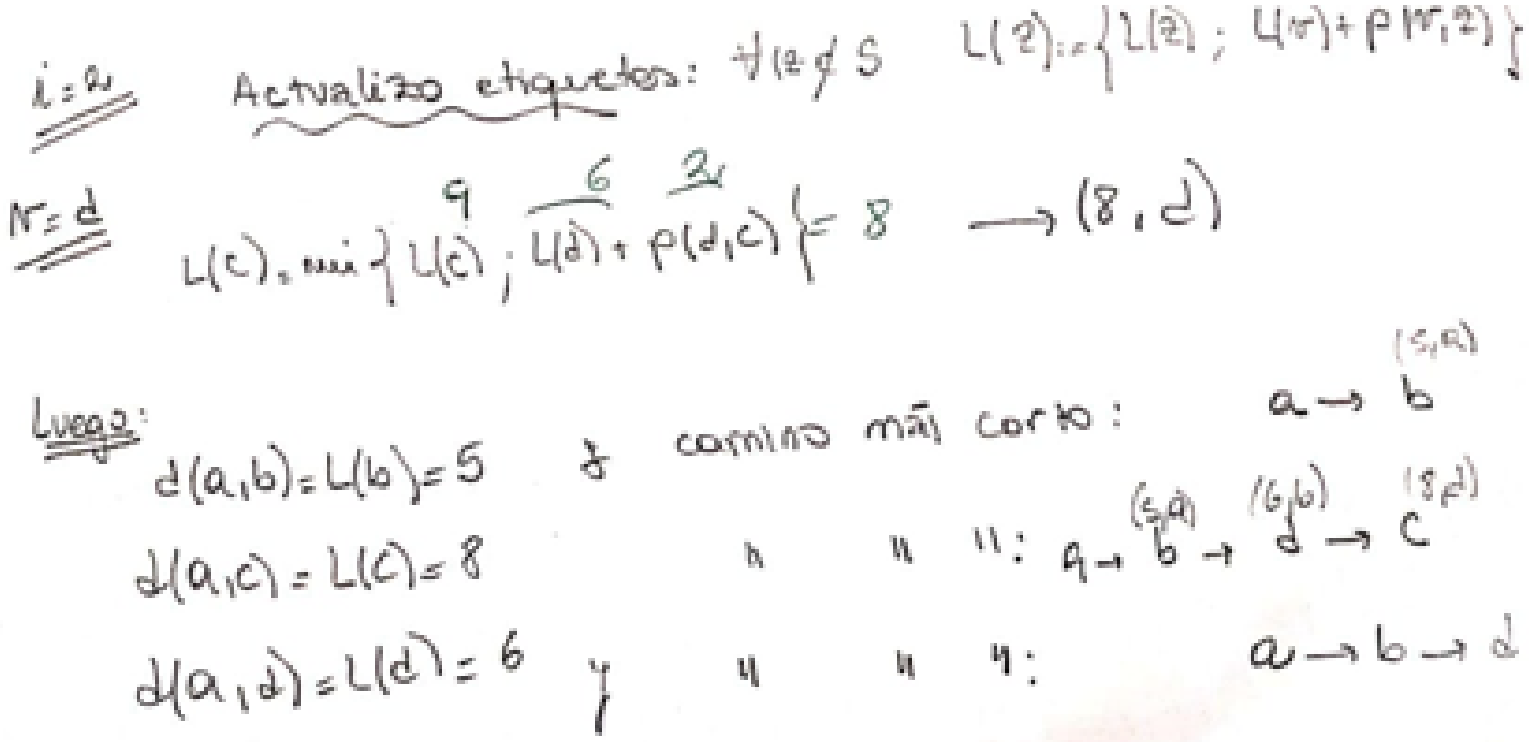
i=1 prob

Actualizo etiquetas: $\forall z \notin S \quad L(z) := \{L(z) ; L(v) + p(v, z)\}$

$L(c) := \min\{\overbrace{\infty}^{L(c)} ; \overbrace{5}^{L(b)} + \overbrace{4}^{p(b,c)}\} = 9 \rightarrow (9, b)$

$L(d) := \min\{\overbrace{7}^{L(d)}, \overbrace{5}^{L(b)} + \overbrace{1}^{p(b,d)}\} = 6 \longrightarrow (6, b)$

Ejemplo 2: (Continuación)

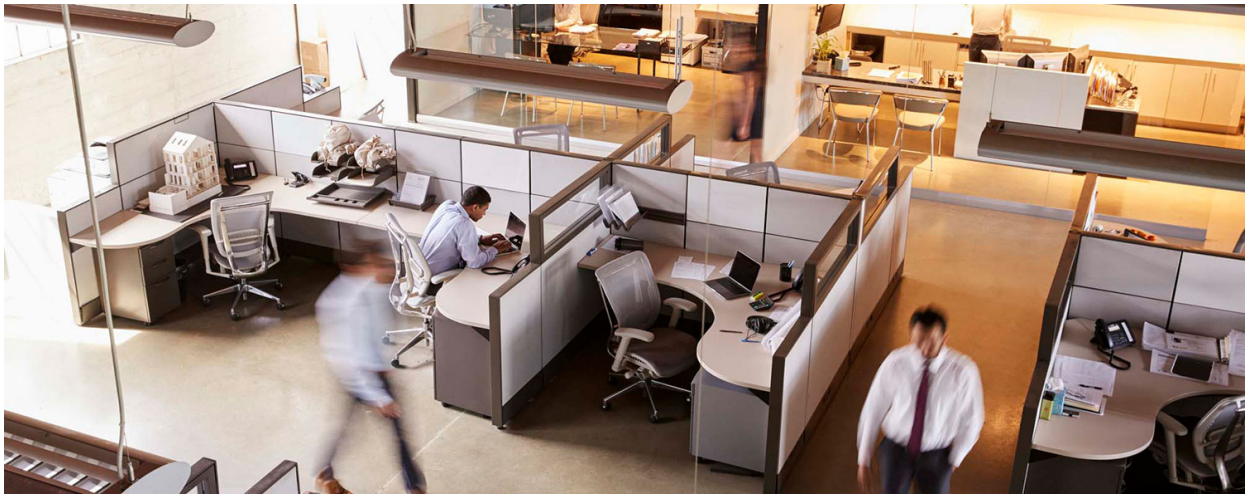


Otros problemas de optimización

Consideremos la siguiente situación real:

- Una empresa con siete oficinas, trata de contruir una red de cómputo, en las cuales existen líneas de transmisión para conectar ciertos pares de oficinas.

- Cada línea de transmisión tiene un costo previsto para la construcción.



OBJETIVO: Contruir una red minimizando el costo total de la construcción.

Otros problemas de optimización

Modelamos la situación de la siguiente manera:

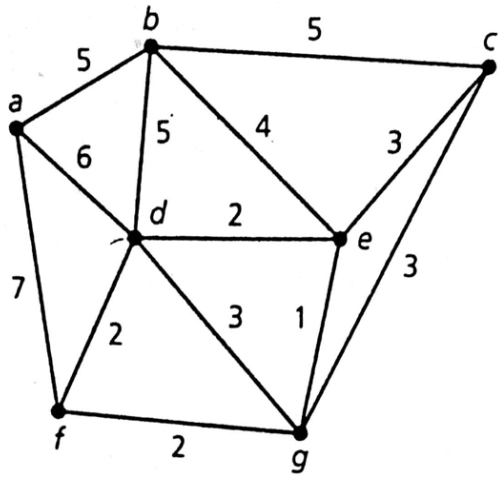
- Cada oficina se representa mediante un vértice en el grafo:

$$a, b, c, d, e, f, g.$$

- Por cada línea de transmisión que permite conectar dos oficinas trazamos una arista:

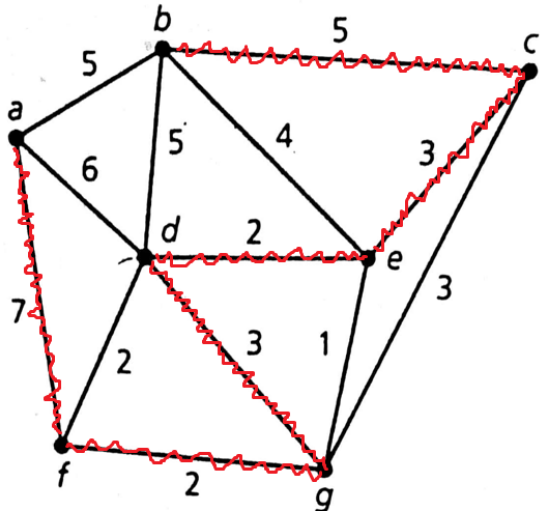
$$ab, ad, af, bc, be, \cdots$$

- A cada arista le asociamos un número real positivo $p(e)$ que indica el costo de la construcción de esa línea de transmisión particular.

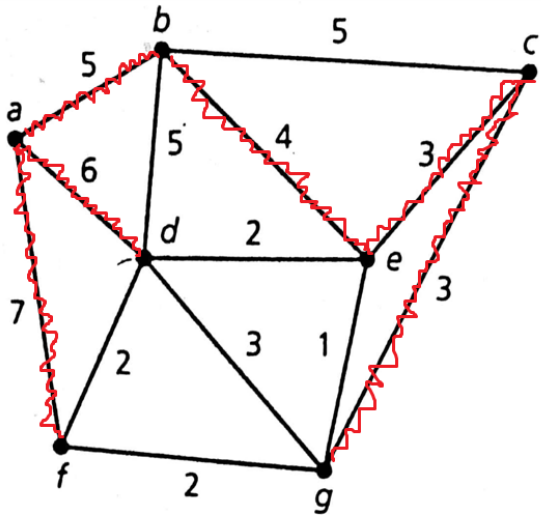


Diferentes formas de contruir una red entre las oficinas:

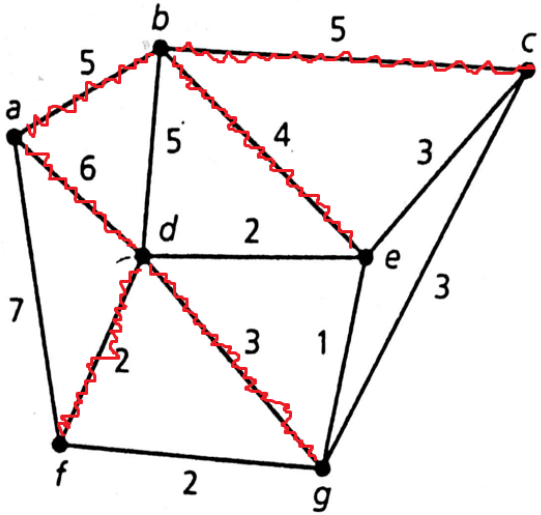
Forma 1:



Forma 2:



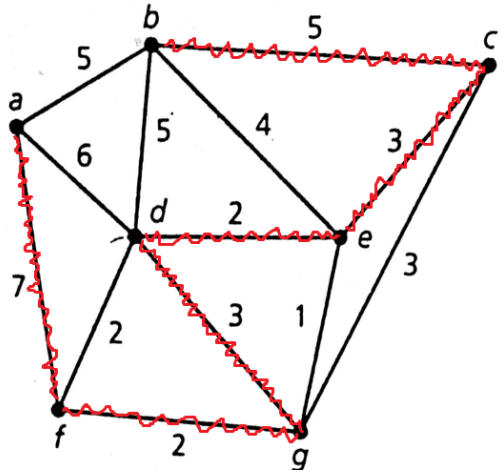
Forma 3:



¿Qué estructura buscamos? *Árbol recubridor T, de peso mínimo.*

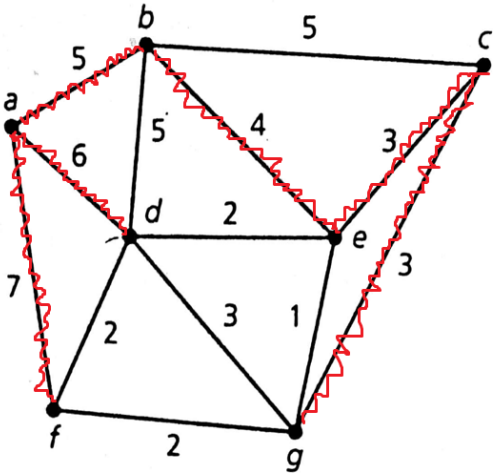
¿Qué peso tiene los árboles recubridores obtenidos?

- Forma 1:



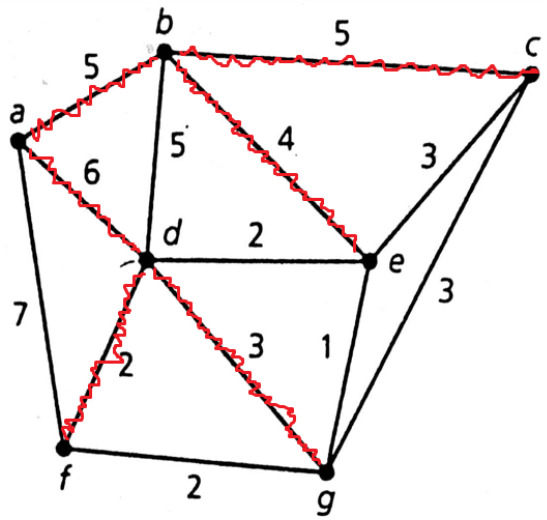
$$p(T_1) = 5 + 3 + 2 + 3 + 2 + 7 = 22.$$

- Forma 2:



$$p(T_2) = 3 + 3 + 4 + 5 + 6 + 7 = 28.$$

- Forma 3:



$$p(T_3) = 5 + 5 + 4 + 6 + 3 + 2 = 25.$$

Objetivo

Hallar un árbol recubridor T, tal que la suma de los pesos de las aristas de T sea mínima.

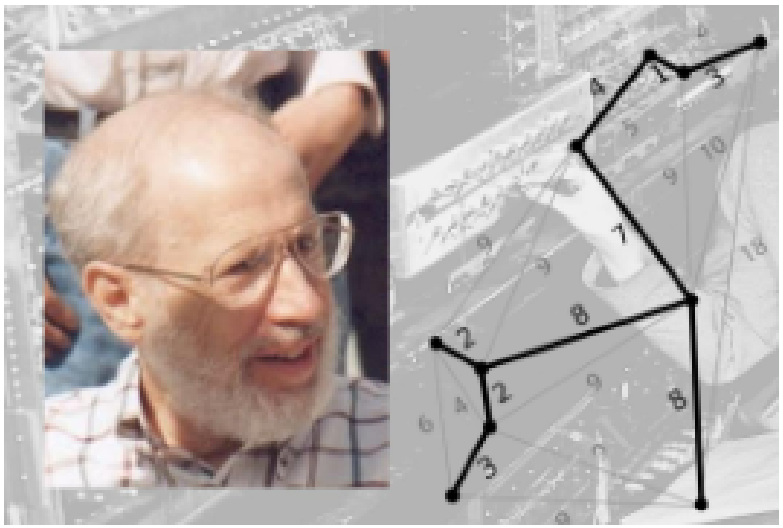
Problemas de optimización: Búsqueda de un árbol recubridor de peso mínimo.

La construcción de dicho árbol recubridor *óptimo* puede realizarse por medio de los algoritmos desarrollados por:

- Joseph Kruskal (1928-2010).
- Robert Prim (1921-2021).

Joseph Kruskal

- Joseph B. Kruskal (1928 -2010) fue un matemático y estadístico estadounidense.
- En informática, su trabajo más conocido es el algoritmo de Kruskal para calcular el árbol recubridor de mínimo peso en un grafo ponderado.



Algoritmo de Kruskal

Sea $G = (V, E)$ un grafo conexo sin lazo, ponderado. Donde:

$|V| = n$ y cada arista e tiene asignado un número real positivo $p(e)$.

El Algoritmo de Kruskal nos permitirá encontrar un árbol recubridor óptimo (minimo) para G. Es decir, nos permite hallar:

T árbol recubridor de G tal que $p(T) \leq p(T')$ para todo árbol recubridor T' de G.

Donde $p(T) = \sum_{e \in E(T)} p(e)$.

- Es un algoritmo goloso: en cada paso del proceso, se hace una elección óptima (de peso mínimo) de los datos disponibles.
- Lo que parece ser la mejor opción localmente (en cada iteración), es la mejor opción globalmente.
- El algoritmo goloso, nos llevará a una solución óptima.

Algoritmo de Kruskal

- **Paso 1:** Hacemos el contador $i = 1$ y seleccionamos una arista e_1 en G, tal que $p(e_1)$ sea lo menor posible.
- **Paso 2:** Para $1 \leq i \leq n-2$, si hemos seleccionado las aristas $e_1, e_2, \cdots, e_i$, entonces:

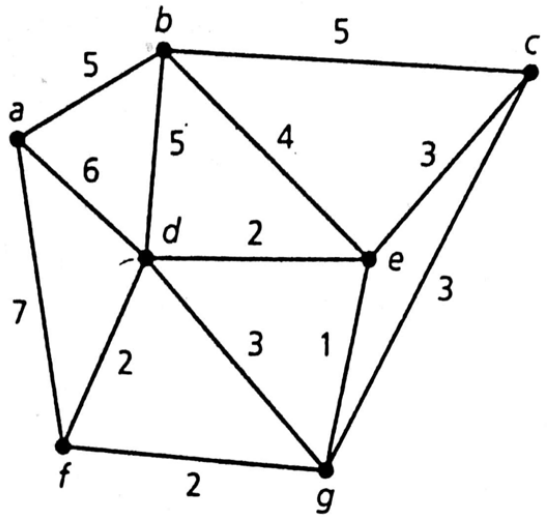
  Seleccionamos la arista e_{i+1} tal que $p(e_{i+1})$ sea de peso mínimo entre todas las aristas aún no elegidas que no forman ciclos con $e_1, e_2, \cdots, e_i$.

- **Paso 3:** Reemplazamos el contador i con $i + 1$.
  - Si $i = n - 1$, **terminamos**. El grafo determinado por $e_1, e_2, \cdots, e_{n-1}$ es un árbol recubridor de G (conexo con n vértices y $n - 1$ aristas).
  - Si $i < n - 1$, regresamos al **Paso 2**.

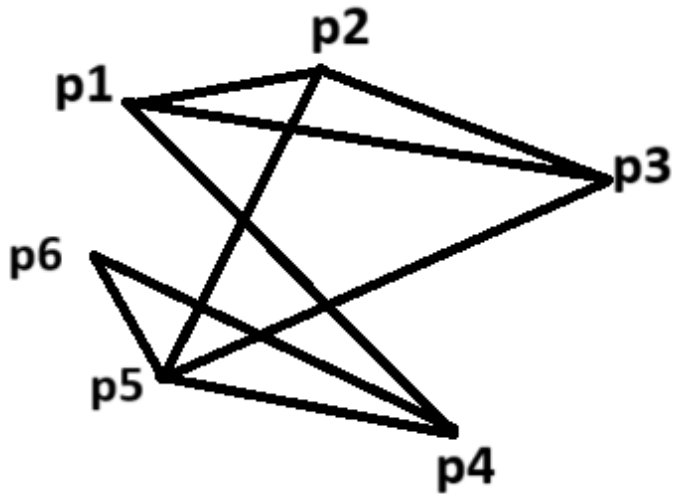
Teorema:

Sea G un grafo ponderado, conexo y sin lazos.
Cualquier árbol recubridor de G obtenido mediante el algoritmo de Kruskal es óptimo.

Ejemplo 1: Hallar un árbol recubridor de peso mínimo en el siguiente grafo ponderado, aplicando el algoritmo de Kruskal:



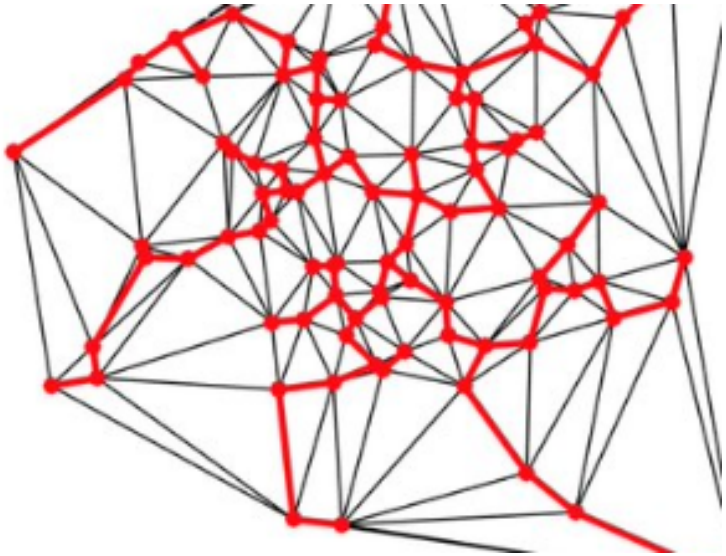
Ejemplo 2: Para hacer un control del funcionamiento de las bombas en 6 pozos petroleros: $p_1, \cdots, p_6$, una persona a cargo necesita visitar todos los pozos diariamente.
Se quiere optimizar el recorrido de mantenimiento preventivo de dicha persona a cargo. Si la conexión entre los diferentes pozos esta dada por la siguiente matriz de adyacencia:



y las distancias entre los diferentes pozos esta dado en metros por: $d(p_1, p_2) = 200,\ d(p_1, p_3) = 150,\ d(p_1, p_4) = 1000,\ d(p_2, p_3) = 500,\ d(p_2, p_5) = 450,\ d(p_3, p_5) = 940,\ d(p_4, p_5) = 240,\ d(p_4, p_6) = 650; d(p_5, p_6) = 700$, hallar un recorrido óptimo.

Robert Prim

- Robert C. Prim (1921- 2021) era un matemático y científico de la computación estadounidense.
- En 1941 se licenció en ingeniería eléctrica en la Universidad de Princeton. Más tarde, en 1949 recibe su doctorado en matemáticas en la misma universidad.
- Robert Prim fue compañero de Joseph Kruskal y desarrolló otro algoritmo (goloso) diferentes para encontrar los árboles recubridor de mínimos peso en un grafo ponderado.



Algoritmo de Prim

Robert Prim desarrolló una segunda técnica para la construcción de un árbol recubridor óptimo de un grafo ponderado G, conexo y sin lazos.

En este algoritmo, los vértices se dividen en dos conjuntos:

- **P:** vértices procesados.
- **N:** vértices por procesar.

Al principio, sólo hay un vértice en el conjunto P y los demás están en N.

Cada iteración del algoritmo incrementa en un vértice al conjunto P , mientras que el cardinal del conjunto N decrece en uno.

Algoritmo de Prim

Sea $G = (V, E)$ grafo ponderado, conexo y sin lazos. Donde $n = |V|$.

- **Paso 1:** Hacemos el contador $i = 1$ y colocamos un vértice arbitrario $v_1 \in V$ en el conjunto P. Definimos $N = V - \{v_1\}$ y $T = \emptyset$.
- **Paso 2:** Para $1 \leq i \leq n-2$, si tenemos $P = \{v_1, v_2, \cdots, v_i\}$, $T = \{e_1, e_2, \cdots, e_{i-1}\}$, y $N = V - P$ entonces:

  Seleccionamos la arista de peso mínimo de G que conecte un vértice v de P con un vértice v_{i+1} en N.

  Agregamos dicha arista a T, dicho vértice v_{i+1} en P y lo eliminamos de N.

$$P := P \cup \{v_{i+1}\}, \qquad N := N - \{v_{i+1}\}, \qquad T := T \cup \{vv_{i+1}\}$$

- **Paso 3:** Reemplazamos el contador i con $i + 1$.
  - Si $i = n$, **terminamos**. El grafo determinado por las aristas $e_1, e_2, \cdots, e_{n-1}$ es un árbol recubridor óptimo (conexo con n vértices y $n-1$ aristas).
  - Si $i < n$, regresamos al **Paso 2**.

Teorema:

Sea G un grafo ponderado, conexo y sin lazos.
Cualquier árbol recubridor de G obtenido mediante el algoritmo de Prim es óptimo.

Ejemplo 3: Hallar un árbol recubridor de peso mínimo en el siguiente grafo ponderado, aplicando el algoritmo de Prim: