75.40 Algoritmos y Programación II Curso 4
Tda Grafo

Dr. Mariano Méndez[1]

[1]Facultad De Ingeniería. Universidad de Buenos Aires

7 de junio de 2020

1. Introducción

El primer artículo científico relativo a grafos fue escrito por el matemático suizo Leonhard Euler en 1736. Euler se basó en su artículo en el problema de los puentes de Königsberg. La ciudad de Kaliningrado, originalmente Königsberg, es famosa por sus siete puentes que unen ambas márgenes del río Pregel con dos de sus islas. Dos de los puentes unen la isla mayor con la margen oriental y otros dos con la margen occidental. La isla menor está conectada a cada margen por un puente y el séptimo puente une ambas islas. El problema planteaba lo siguiente: ¿es posible dar un paseo comenzando desde cualquiera de estas regiones, pasando por todos los puentes, recorriendo solo una vez cada uno y regresando al mismo punto de partida?

Abstrayendo este problema y planteándolo con la (entonces aún básica) teoría de grafos, Euler consigue demostrar que el grafo asociado al esquema de puentes de Königsberg no tiene solución, es decir, no es posible regresar al vértice de partida sin pasar por alguna arista dos veces.

Figura 1: El problema de los puentes de Königsberg

De hecho, Euler resuelve el problema más general: ¿qué condiciones debe satisfacer un grafo para garantizar que se puede regresar al vértice de partida sin pasar por la misma arista más de una vez? Si definimos como grado al número de líneas que se encuentran en un punto de un grafo, entonces la respuesta al problema es que los puentes de un pueblo se pueden atravesar exactamente una vez si, salvo a lo sumo dos, todos los puntos tienen un grado par.

dónde aparecen los grafos:

- redes sociales.
- Análisis de rutas.
- Recomendaciones.

1.1. Glosario

Un grafo es una colección de vértices (o nodos) y aristas (o arcos). En vértice es un objeto que entre otras propiedades tiene que tener un nombre. Una arista es una conexión entre dos vértices. Un *grafo* G es un par ordenado $G = (V, E)$ donde:

- V es un conjunto de vértices o nodos, y
- E es un conjunto de aristas o arcos, que relacionan estos nodos.

Normalmente V suele ser finito.

1.1.1. Grafo No Dirigido

Un grafo **no dirigido** o *grafo propiamente* dicho es un grafo G=(V,E) donde:

- $V \neq \emptyset$, un conjunto no vacío de objetos simples llamados vértices o nodos.
- $E \subseteq \{x \in \mathcal{P}(V) : |x| = 2\}$ es un conjunto de pares no ordenados de elementos de V.

Un par no ordenado es un conjunto de la forma $\{a, b\}$, de manera que $\{a, b\} = \{b, a\}$. Para los grafos, estos conjuntos pertenecen al conjunto potencia de V, denotado $\mathcal{P}(V)$, y son de cardinalidad 2.

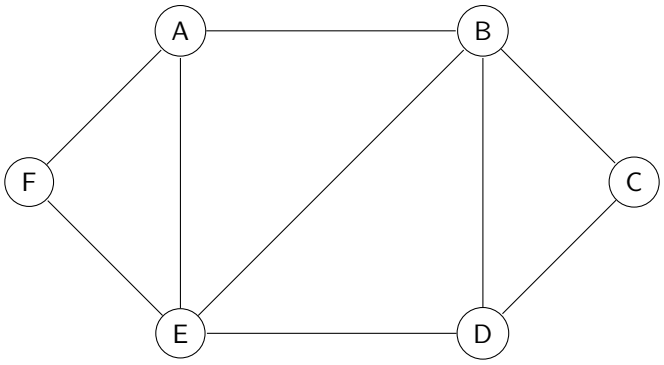


Figura 2: Grafo no Dirigido

1.1.2. Grafo Dirigido

Un grafo dirigido o digrafo es un grafo $G = (V, E)$ donde:

- $V \neq \emptyset$, un conjunto no vacío de objetos simples llamados vértices o nodos.
- $E \subseteq \{(a, b) \in V \times V : a \neq b\}\}$, es un conjunto de pares ordenados de elementos de V. Dada una arista(a, b), a es su nodo inicial y b su nodo final.

Ver Figura **??**
Por definición, los grafos dirigidos no contienen bucles.

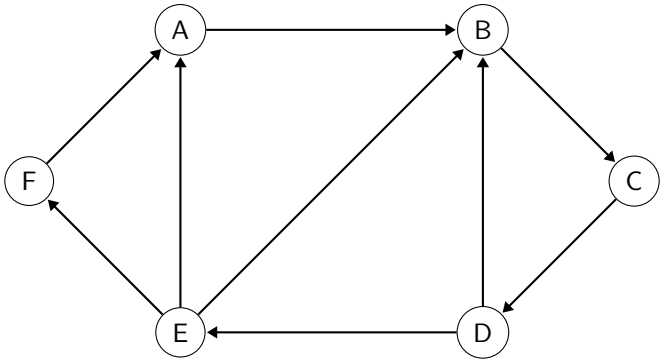


Figura 3: Grafo Dirigido

Un subgrafo es un subconjunto de aristas de un grafo y sus vértices asociados que constituyen un grafo.
Un **camino (path)** en un grafo es una secuencia de vértices en la cual cada vértice sucesivo (después del primero) es adyacente a su predecesor en el camino. En un **camino simple** los vértices y las aristas son distintos.
Un **ciclo** es un camino simple en el cual el primer y último vértice son el mismo.
Un grafo es **conexo** si existe un camino desde cada vértice hacia todos los demás vértices en el grafo.
Ademas las aristas pueden poseer lo que se denomina un peso:

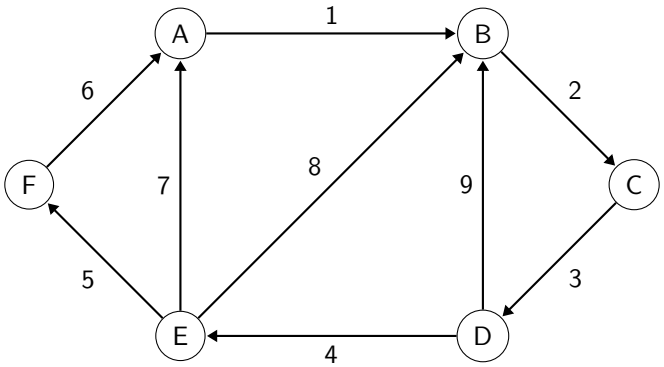


Figura 4: Grafo Dirigido con peso

Los vértices pueden ser clasificados como:

- directos con peso
- directos sin peso
- sin dirección con peso
- sin dirección sin peso

1.1.3. Grafos Densos y Dispersos

En matemática, un **grafo denso** es un grafo cuyo numero de aristas está muy cerca del valor máximo de aristas que este puede tener. Para un grafo simple no dirigido, un grafo denso con V vértices y E aristas, se define como:

$$D = \frac{2|E|}{|V|\left(|V|-1\right)}$$

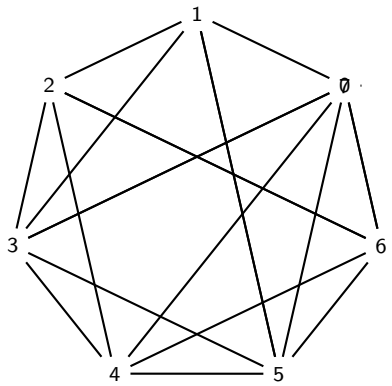


Figura 5: Grafo Denso

Por el contrario un **grafo disperso** es un grafo con pocas aristas :

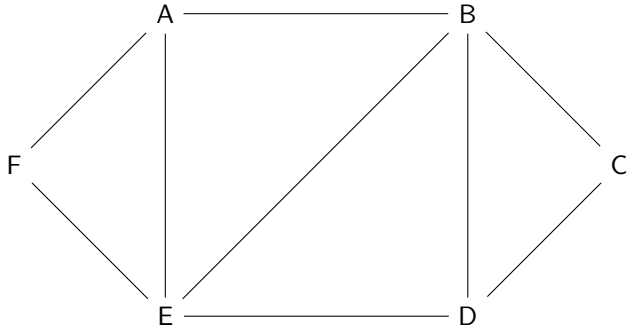


Figura 6: Grafo Disperso

2. El tda Grafo

Existen varias formas de implementar el tipo de dato abstracto grafo. A continuación se definirán las operaciones básicas del tda:

- **crear()**: esta operación crea la estructura interna del grafo y devuelve un puntero a un grafo
- **insertar_arista(grafo, arista)**: inserta dentro de la estructura del grafo la nueva arista.
- **eliminar_arista(grafo, arista)** : elimina dicha arista del grafo.
- **aristas(aristas[], grafo)**: devuelve un lista de todas las aristas del grafo.
- **destruir(grafo)**: destruye el tda grafo.

más operaciones:

- adyacentes(vértice, vértice)
- vértices(vértices[], arista)
- **copiar(grafo)**: devuelve un grafo que es la copia del pasado por parámetro.

Además de estas operaciones se define un tipo de dato arista y un tipo de dato vértice.

2.1. Representaciones

Existen varias formas de representar un grafo. En cada caso se evaluará el costo en espacio de la representación.

2.1.1. Matriz de Adyacencia

Es posible representar un grafo a partir de una matriz binaria en la cual cada elemento representa una relación de adyacencia entre dos vértices, es decir si existe una arista entre el vértice i y el vértice j entonces en m[i][j] se debe colocar en 1, de lo contrario va el valor 0. En el caso de que el grafo sea no dirigido que la matriz de adyacencia resultante es una matriz simétrica, ya que A[i][j] = A[j][i].

	A	B	C	D	E	F
A	0	1	0	1	0	0
B	1	0	1	1	1	0
C	1	1	0	0	1	0
D	0	1	0	0	1	0
E	0	1	1	1	0	1
F	0	0	0	0	1	0

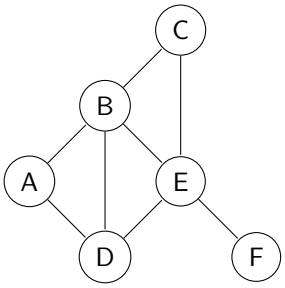


Figura 7: Grafo no Dirigido

Si el grafo fuera un grafo dirigido la matriz de adyacencia solo registra el vértice de inicio y al vértice final, A[i][j] = 1, si y solo si la arista parte del vértice i hacia el vértice j:

	A	B	C	D	E	F
A	0	1	0	1	0	0
B	0	0	1	1	1	0
C	0	0	0	0	1	0
D	0	1	0	0	1	0
E	0	1	1	1	0	1
F	0	0	0	0	0	0

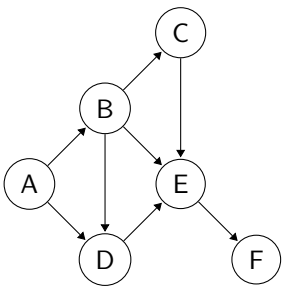


Figura 8: Grafo Dirigido

En el caso que Las aristas del grafo posean un peso :

	A	B	C	D	E	F
A	0	1	0	3	0	0
B	0	0	4	7	8	0
C	0	0	0	0	2	0
D	0	0	0	0	1	0
E	0	0	0	0	1	0
F	0	0	0	0	0	0

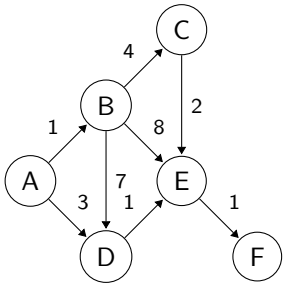


Figura 9

El costo en espacio de esta representación es $O(V^2)$
La representación de un grafo mediante una matriz de adyacencia tiene algunos problemas:

1. Respecto al espacio de memoria que utiliza es $O(N^2)$.
2. Si el grafo es un grafo denso o completo puede ser una buena representación.
3. si el grafo es un grafo disperso es una pésima representación.

2.1.2. Lista de Adyacencia

Otra forma de representar un grafo es mediante una lista de adyacencia. Conceptualmente es parecida a la matriz de adyacencia y consiste que cada arista posee una lista simplemente enlazada de a que vértices está unida. En un grafo no dirigido:

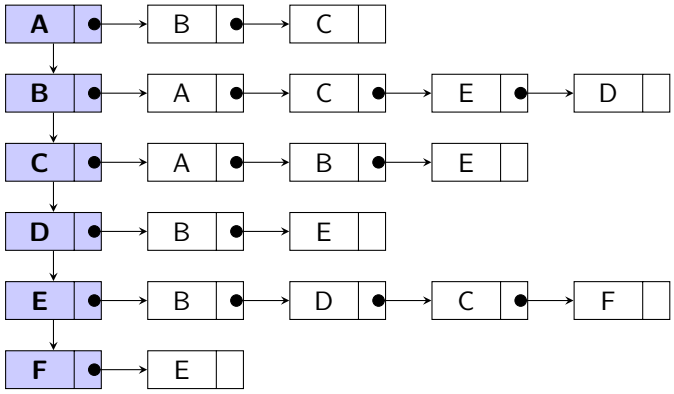


Figura 10: Lista de Adyacencia

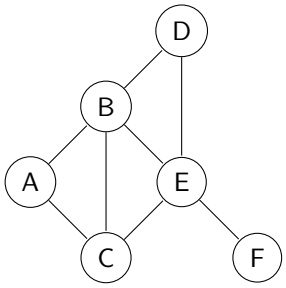


Figura 11

En un Grafo dirigido:

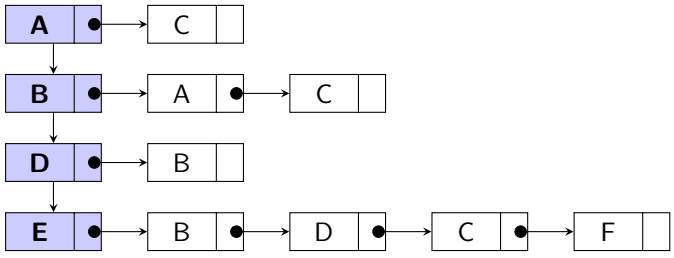


Figura 12: Lista de Adyacencia

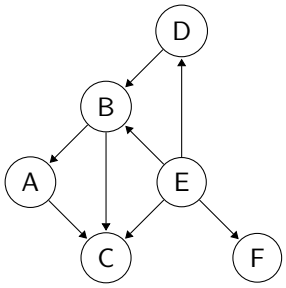


Figura 13

El costo en espacio de esta representación es $O(V + A)$

2.1.3. Matriz de Incidencia

Es otra forma para representar un grafo, se denomina matriz de incidencia. En la cual las columnas son los vértices y las aristas son las filas:

	A	B	C	D	E	F
a1	1	0	1	0	0	0
a2	0	0	1	0	1	0
a3	0	0	0	0	1	1
a4	0	0	0	1	1	0
a5	0	1	1	0	0	0
a6	0	1	0	0	1	0
a7	0	1	0	1	0	0
a8	1	1	0	0	0	0

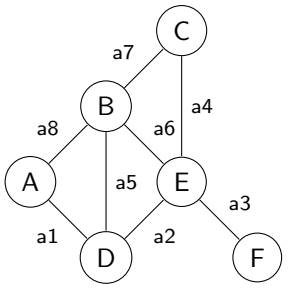


Figura 14

Si el grafo a representar es dirigido, se utiliza la siguiente convención, -1 se le asigna al vértice de salida y 1 al vértice de entrada:

	A	B	C	D	E	F
a1	-1	0	1	0	0	0
a2	0	0	1	0	-1	0
a3	0	0	0	0	-1	1
a4	0	0	0	1	-1	0
a5	0	-1	1	0	0	0
a6	0	1	0	0	-1	0
a7	0	-1	0	1	0	0
a8	1	-1	0	0	0	0

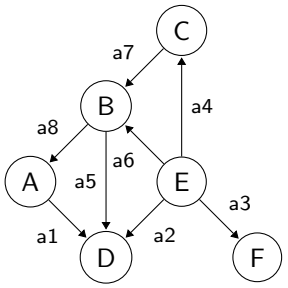


Figura 15

En el caso que el grafo fuera dirigido y con peso, en vez de utilizar el numero 1 se utiliza el numero del peso de cada arista.

El costo en espacio de esta representación es $O(V * A)$

2.1.4. Más definiciones

- **Ciclo**: recorrido de al menos dos aristas que empieza y termina en mismo lugar. Se pone la restricción de las aristas porque sino todo en un grafo no dirigido sería un ciclo.
- **Camino**: un recorrido a través del grafo.
- **Camino simple**: Es un camino, pero que pasa (a lo sumo) una vez por cada vértice
- **Componente conexa**: para grafos no dirigidos únicamente, conjunto de vértices donde existe un camino de un vértice a otro.
- **Grado de entrada**: cantidad de aristas que entran al vértice.
- **Grado de salida**: cantidad de aristas que salen del vértice. En grafo no dirigido, ambos grados son iguales y directamente se lo llama grado del vértice.

3. Algoritmos Clásicos

3.1. Recorridos

Cuando se habla de recorrido se entiende por el proceso de búsqueda o atravesar la estructura de datos de un grafo que involucra la visita de cada vértice/nodo en el grafo; el orden en el cual los vértices son visitados es la forma en que se clasifica el tipo de recorrido. Existen básicamente dos formas claves en que un grafo se recorre en **anchura** o en **profundidad**.

Cuando se realiza un recorrido en un grafo el algoritmos tiene 2 acciones bien definidas :

- **Visitar**: normalmente esta acción es marcar como visitado al nodo.
- **Explorar**: es la política que va a definir cómo se comportará el algoritmo, explorar los vecinos, explorar los hijos, etc.

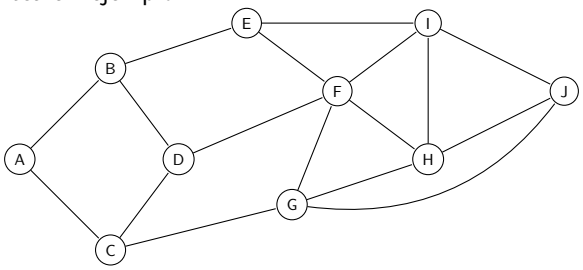
3.1.1. BFS

Este algoritmo **Breath First Search** encara el recorrido del grafo en anchura, es decir primero visita nodos hermanos / vecinos antes de visitar nodos hijos. Para su implementación utiliza una estructura de datos auxiliar que es una cola.

El Algoritmo El algoritmo de BFS consiste en un conjunto básico de pasos:

1. Agregar un nodo/vértice del grafo a la cola de nodos a ser visitados.
2. Visitar el primer nodo de la cola, y marcarlo como tal
3. Si el nodo tiene algún vecino, verificar si han sidos visitados o no.
4. Agregar a la colla todos aquellos vecinos que tienen que ser visitados a la cola.
5. Eliminar el nodo visitado de la cola

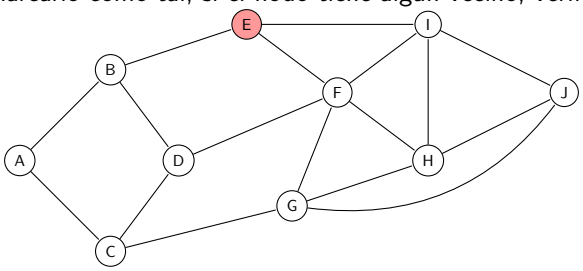
Se repiten estos pasos hasta que no quede ninguno nodo de la cola sin haber sido visitado. A continuación se plantea un ejemplo:



Cola

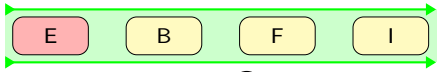
En el ejemplo se toma el vértice/nodo de partida en forma arbitraria, en este caso E:

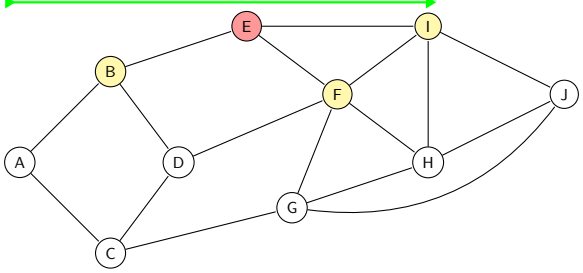
Paso 1 Se agrega el nodo/vértice E del grafo a la cola de nodos a ser visitados. Se visita el primer nodo de la cola, y marcarlo como tal, si el nodo tiene algún vecino, verificar si han sidos visitados o no.



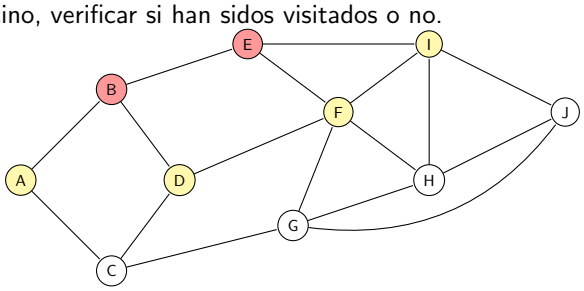
Se agregan a la cola todos aquellos vecinos que tienen que ser visitados, en este caso los nodos B,F,I. Eliminar el nodo visitado de la cola (color rojo).

Cola



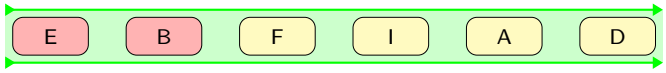


Paso 2 Se toma el siguiente nodo de la cola marcado como no visitado se marca como tal, si el nodo tiene algún vecino, verificar si han sidos visitados o no.

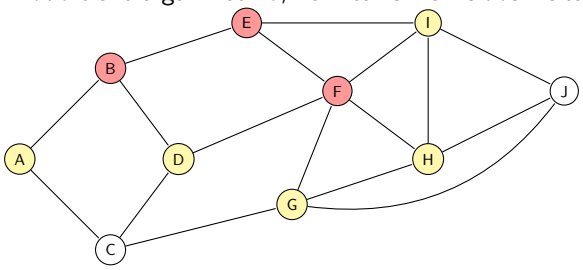


Se agregan a la cola todos aquellos vecinos que tienen que ser visitados, en este caso los nodos A,D. Eliminar el nodo visitado de la cola (color rojo)

Cola

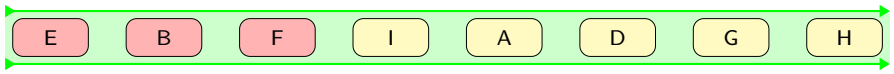


Paso 3 se toma el siguiente nodo de la cola marcado como no visitado se marca como tal, en este caso el nodo F, si el nodo tiene algún vecino, verificar si han sidos visitados o no.

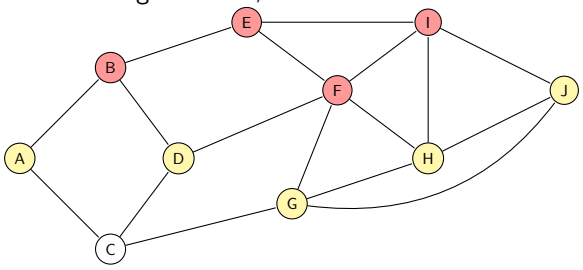


Se agregan a la cola todos aquellos vecinos que tienen que ser visitados, en este caso los nodos G,H. Eliminar el nodo visitado de la cola

Cola

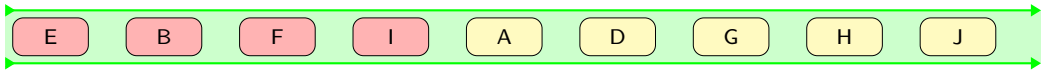


Paso 4 El nodo/vértice al inicio de la cola marcado como no visitado se marca como tal, en este caso el nodo I, si el nodo tiene algún vecino, verificar si han sidos visitados o no.

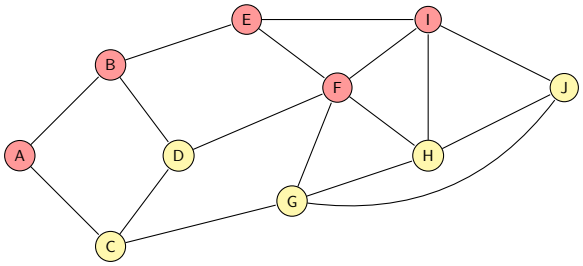


Se agregan a la cola todos aquellos vecinos que tienen que ser visitados, en este caso los nodos J. Eliminar el nodo visitado de la cola

Cola

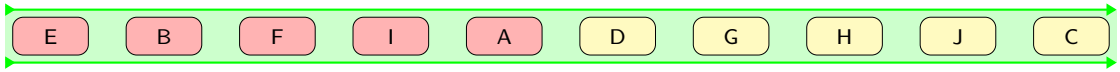


Paso 5 El nodo/vértice al inicio de la cola marcado como no visitado se marca como tal, en este caso el nodo A, si el nodo tiene algún vecino, verificar si han sidos visitados o no.

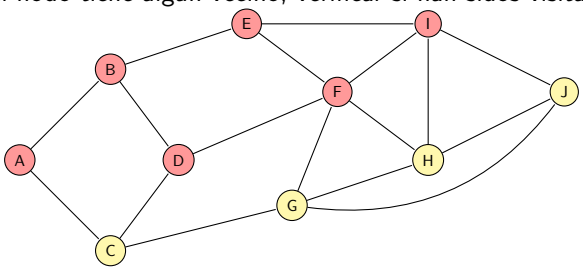


Se agregan a la cola todos aquellos vecinos que tienen que ser visitados, en este caso los nodos C. Eliminar el nodo visitado de la cola

Cola

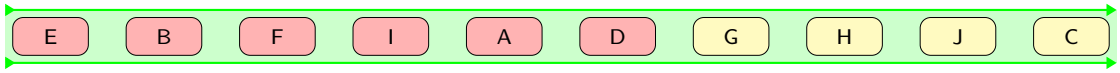


Paso 6 El nodo/vértice al inicio de la cola marcado como no visitado se marca como tal, en este caso el nodo D, si el nodo tiene algún vecino, verificar si han sidos visitados o no.

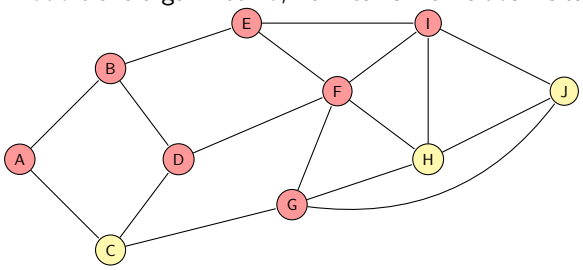


Se agregan a la cola todos aquellos vecinos que tienen que ser visitados, en este caso no hay mas nodos. Eliminar el nodo visitado de la cola

Cola

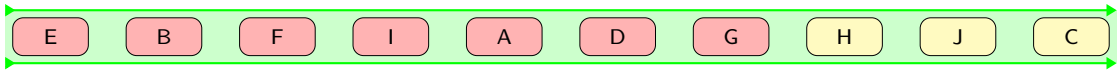


Paso 7 El nodo/vértice al inicio de la cola marcado como no visitado se marca como tal, en este caso el nodo G, si el nodo tiene algún vecino, verificar si han sidos visitados o no.

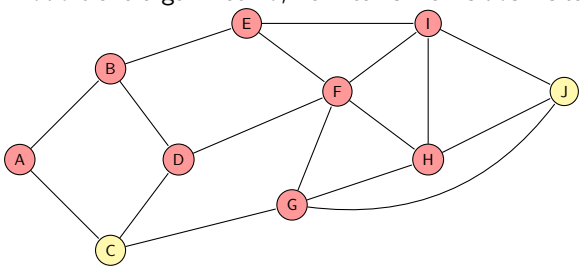


Se agregan a la cola todos aquellos vecinos que tienen que ser visitados, en este caso no hay mas nodos. Eliminar el nodo visitado de la cola

Cola

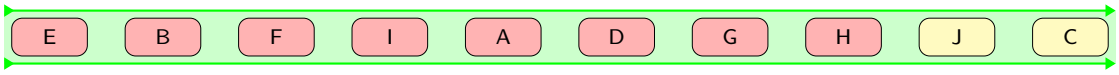


Paso 8 El nodo/vértice al inicio de la cola marcado como no visitado se marca como tal, en este caso el nodo H, si el nodo tiene algún vecino, verificar si han sidos visitados o no.

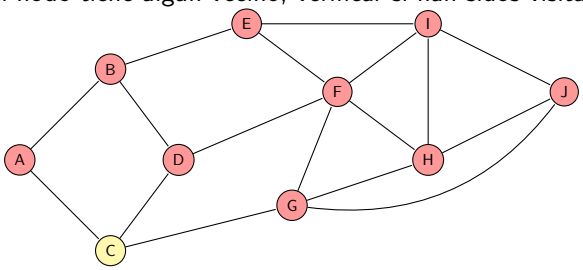


Se agregan a la cola todos aquellos vecinos que tienen que ser visitados,en este caso no hay mas nodos. Eliminar el nodo visitado de la cola

Cola

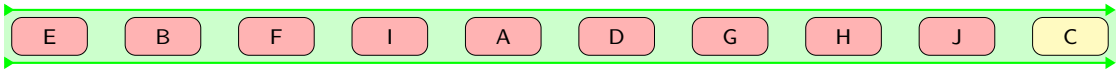


Paso 9 El nodo/vértice al inicio de la cola marcado como no visitado se marca como tal, en este caso el nodo J, si el nodo tiene algún vecino, verificar si han sidos visitados o no.

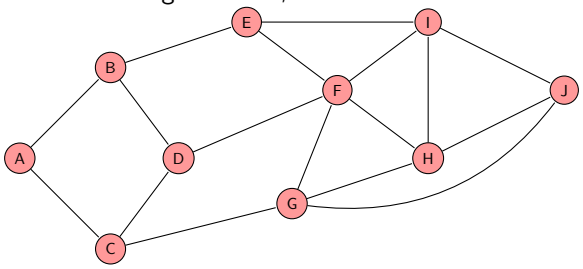


Se agregan a la cola todos aquellos vecinos que tienen que ser visitados, en este caso no hay mas nodos. Eliminar el nodo visitado de la cola

Cola

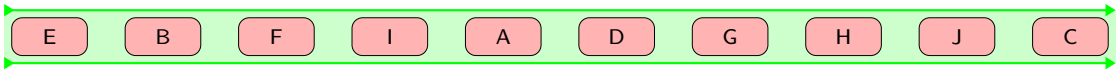


Paso 10 El nodo/vértice al inicio de la cola marcado como no visitado se marca como tal, en este caso el nodo C, si el nodo tiene algún vecino, verificar si han sidos visitados o no.



Se agregan a la cola todos aquellos vecinos que tienen que ser visitados, en este caso no hay mas nodos. Eliminar el nodo visitado de la cola

Cola



Ya en el paso 10 se tiene el grafo visitado en anchura.

3.1.2. DFS

Depth First Serach es otra forma de recorrer un grafo. En este caso el enfoque es completamente diferente al actuado en el algoritmo **Breath First Search** que encara el recorrido del grafo en anchura o de a un nivel a la vez. DFS justamente utiliza una estrategia opuesta. DFS permite determinar si dos nodos x e y poseen un camino entre ellos, la forma de hacerlo es mirando si por cada hijo del nodo inicial, en este caso x se puede llegar al nodo y. En forma recursiva, una y otra vez los mismos pasos, para determinar si existe dicho camino entre los nodos dados.

Mientras que el método BFS, va a recorrer un grafo un nivel de hijos a la vez, DFS va a recorrer un único camino hasta el final, de a un hijo por paso.

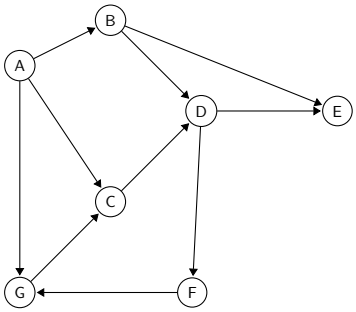
El método DFS es muy parecido a cuando se resuelve un laberinto, se va avanzando hasta que se llega a un callejón sin salida, se vuelve sobre los pasos hasta un punto en que se puede seguir avanzando. de esta forma si uno llega a la salida del laberinto, uno puede asegurar que hay un camino entre la entrada y la salida.

El Algoritmo: Iterativo Existen dos puntos a tener en cuenta:

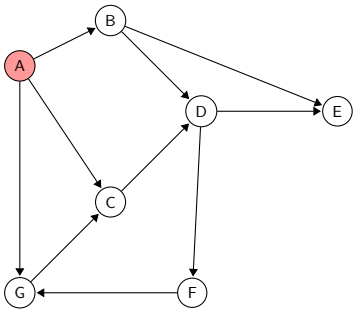
1. El nodo de partida puede ser seleccionado arbitrariamente.
2. Cuando un nodo se marca como visitado, no se vuelve a repetir.

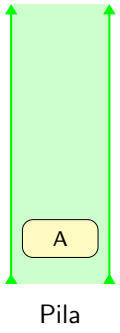
Cada vez que se alcanza un nuevo vértice:

1. Se agrega el vértice en el tope de la pila de vértices visitados
2. Se marca como visitado
3. Se chequea si dicho vértice tiene hijos:
   a) Si los tiene se verifican que no hayan sido visitados y se visitan
   b) si ha sido visitado se hace pop del nodo de la pila.



Paso 1 En concordancia con la descripción del algoritmo se seleccionará el nodo A, para comenzar con el recorrido DFS.

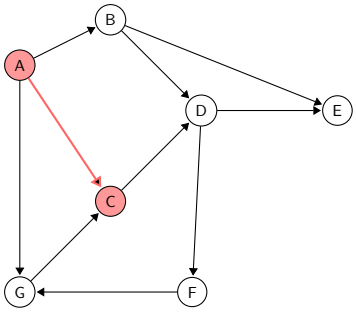


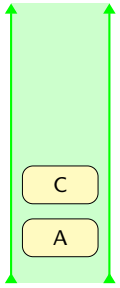


Pila

Visitados=[A]

Paso 2 Ahora se selecciona un hijo del nodo A, en este caso el vértice C, se lo pushea en la pila, si no ha sido visitado se lo marca como visitado.

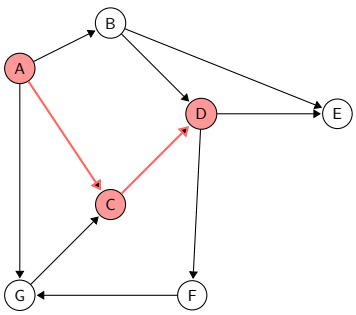


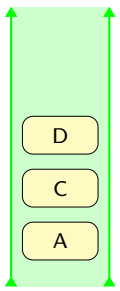


Pila

Visitados=[A,C]

Paso 3 Ahora se selecciona un hijo del nodo C, en este caso el vertice D, se lo pushea en la pila, si no ha sido visitado se lo marca como visitado.

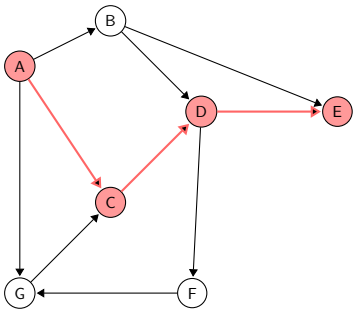


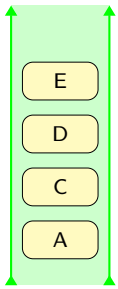


Pila

Visitados=[A,C,D]

Paso 4 Ahora se selecciona un hijo del nodo D, en este caso el vértice E, se lo pushear en la pila, si no ha sido visitado se lo marca como visitado.

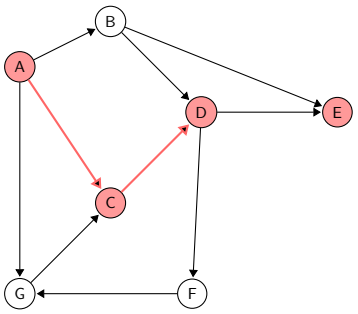


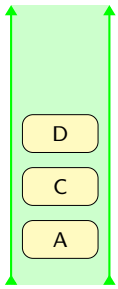


Pila

Visitados=[A,C,D,E]

Paso 5 Dado que el nodo visitado no posee más hijos, se elimina el tope de la pila ya que está marcado como visitado, en este caso el vértice E.

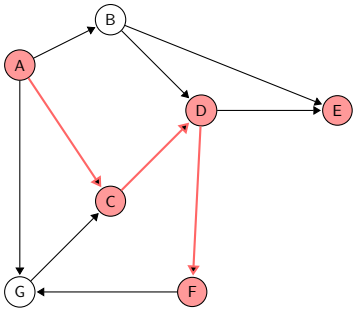


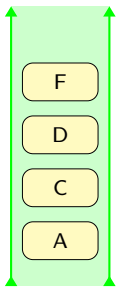


Pila

Visitados=[A,C,D,E]

Paso 6 Ahora se selecciona un hijo del nodo D no visitado, en este caso el vértice F, se lo pushear en la pila, si no ha sido visitado se lo marca como visitado.

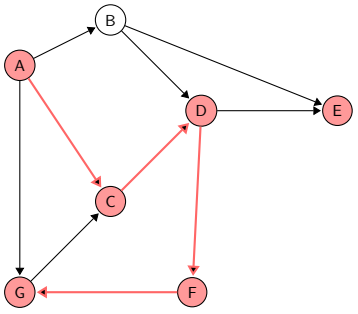


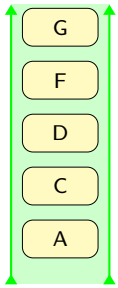


Pila

Visitados=[A,C,D,E,F]

Paso 7 Ahora se selecciona un hijo del nodo F no visitado, en este caso el vértice G, se lo pushear en la pila, si no ha sido visitado se lo marca como visitado.

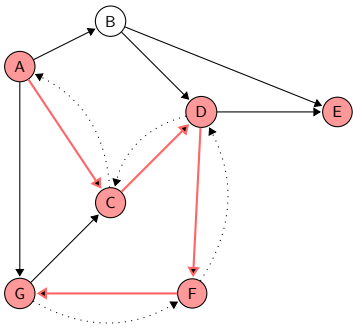


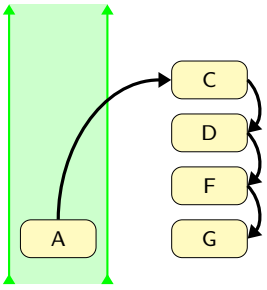


Pila

Visitados=[A,C,D,E,F,E]

Paso 8 Como el vértice G tiene su único hijo marcado como visitado, este no se agrega en la pila y el vértice G es eliminado de la misma al no poseer más hijos no visitados. lo mismo sucede con los vértices F,D,C.

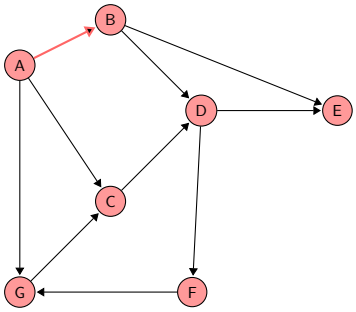


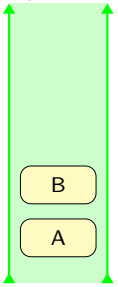


Pila

Visitados=[A,C,D,E,F,E]

Paso 9 Como el vertice G tiene su único hijo marcado como visitado, este no se agrega en la pila y el vértice G es eliminado de la misma al no poseer más hijos no visitados, lo mismo sucede con los vértices F,D,C. De tal forma que se vacía la pila hasta el primer elemento que posea un nodo hijo no visitado, en este caso ese nodo es el vértice A. Con lo cual se continua con el algoritmo, se introduce en la pila el nodo hijo del vértice A, el vértice B, se marca como visitado, en este caso el algoritmo termina ya que todos sus vértices están marcados como visitados





Pila

Visitados=[A,C,D,E,F,E,B]

3.2. Orden Topológico

La idea del orden topologico es la de procesar los vértices de un Grafo Acíclico Dirigido de forma tal que este es procesado antes que todos los vértices al que él apunta.

3.3. Dijkstra

El algoritmo de Dijkstra, también llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto, dado un **vértice origen**, hacia el resto de los vértices en un grafo que tiene pesos en cada arista. Su nombre alude a Edsger Dijkstra, científico de la computación de los Países Bajos que lo describió por primera vez en 1959.

La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen hasta el resto de los vértices que componen el grafo, el algoritmo se detiene. No funciona en grafos con aristas de peso negativo.

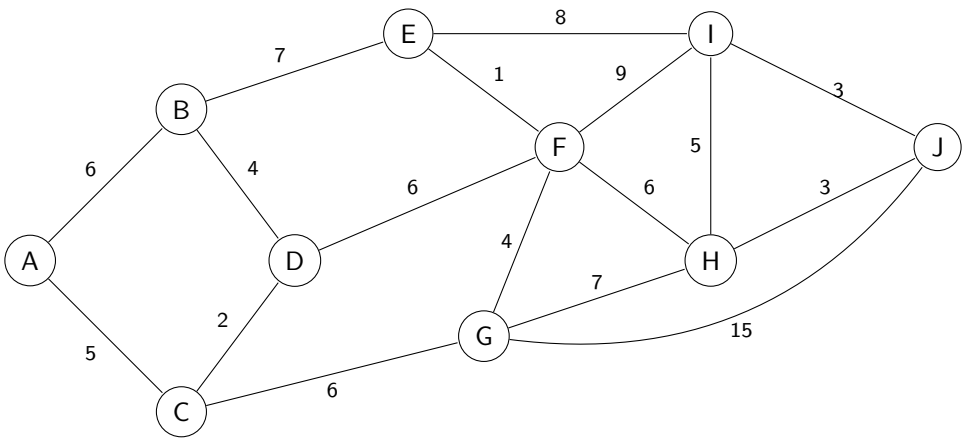


Figura 16

3.3.1. El Algoritmo

El algoritmo de Dijkstra toma un grafo y un nodo del mismo y calcula el camino mínimo de ese nodo a todos los demás nodos que componen el grafo. Para ello:

1. se elije el Vértice V sobre el cual se quiera aplicar el algoritmo,
2. se crean dos lista de nodos, una lista de nodos Visitados y otra listas de nodos NO Visitados, que contiene a todos los nodos del grafo.
3. se crea una tabla con 3 columnas, Vértice, Distancia mínima V y el nodo anterior por el cual se llego.
4. Se toma el Vértice V como vertice inicial y se calcula su distancia a sí mismo, que es 0.
5. se actualiza la tabla, en la cual todas las distancias de los demás vértices a V se marcan como infinito.

Una vez completado esto :

Vértice	Distancia	V. Anterior
A	0	-
B	∞	-
C	∞	-
D	∞	-
E	∞	-
F	∞	-
G	∞	-
H	∞	-
I	∞	-
J	∞	-

V=[]
NV=[A,B,C,D,E,F,G,H,I,J]

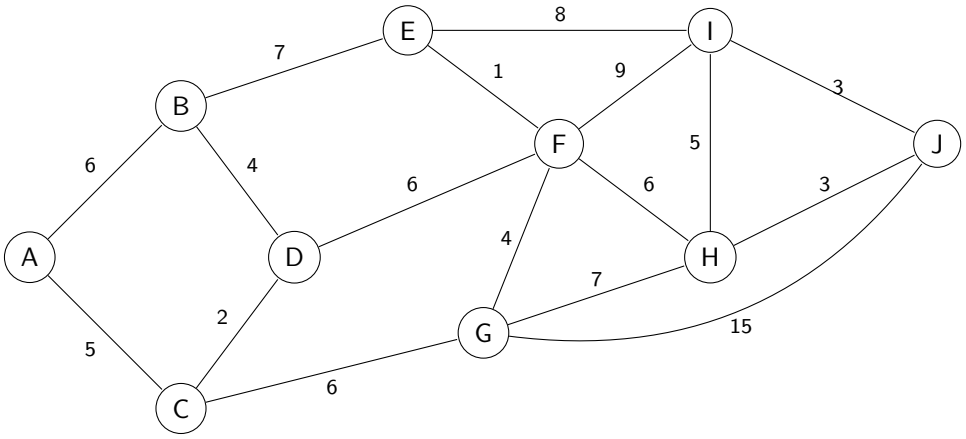
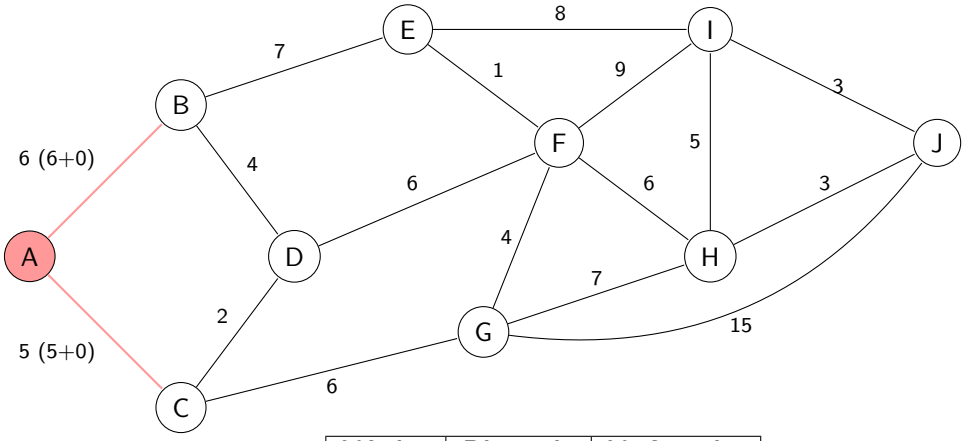


Figura 17

A continuación:

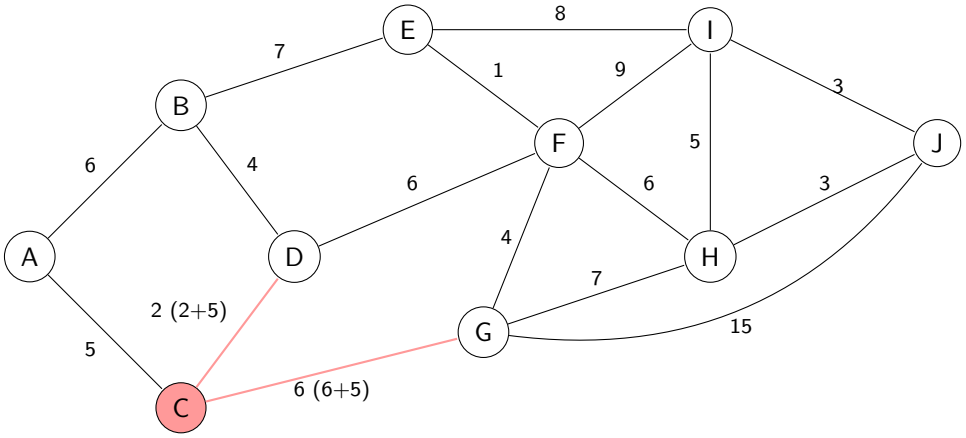
- Se visita el vértice NO VISITADO con menor distancia conocida desde el primer vértice, que es A, ya que la distancia de A a A es 0 y las demás infinito.
- Se calcula la distancia entre los vertices sumando los pesos de cada uno con la distancia de A.
- Si la distancia calculada de los vértices conocidos es menor a la que está en la tabla se actualiza y tambien los vértices desde donde se llegó.
- Se pasa el Vertice A a la lista de Vertices visitados.
- Se continua con el vértice no visitado con menor distancia que es C



Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	∞	-
E	∞	-
F	∞	-
G	∞	-
H	∞	-
I	∞	-
J	∞	-

V=[A]
NV=[B,C,D,E,F,G,H,I,J]

si la lista de no visitados está aun llena, se vuelve a repetir el proceso

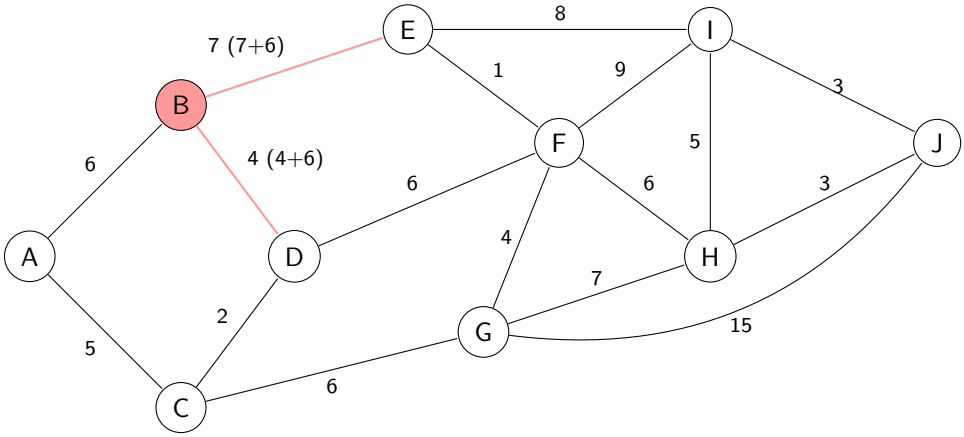


C->G =11 es menor que infinito se actualiza la distancia.
C->D =7 es menor que infinito se actualiza la distancia.

Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	7	C
E	∞	-
F	∞	-
G	11	C
H	∞	-
I	∞	-
J	∞	-

Se pasa C a la lista de Vertice no Visitados:
V=[A,C] NV=[B,D,E,F,G,H,I,J]

Se vuelve a repetir el proceso con el vertice no visitado de menor distancia, en este caso B y se vuelve a repetir el proceso

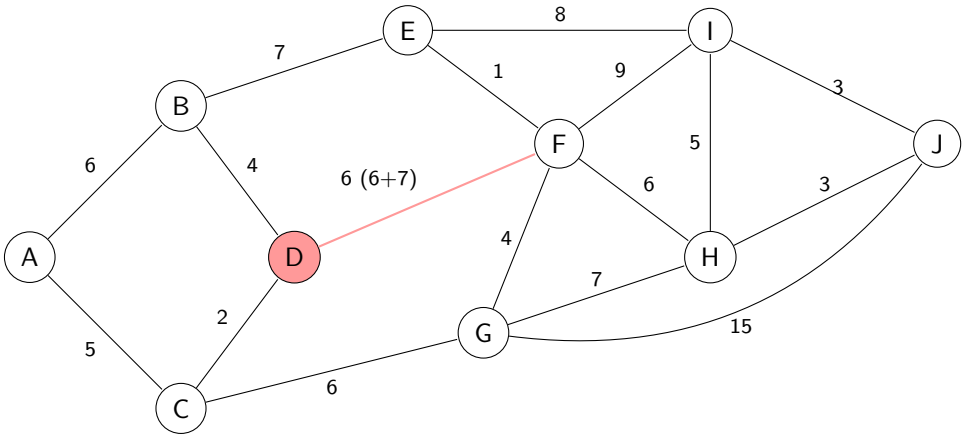


B->E =13 es menor que infinito se actualiza la distancia.
B->D =10 es MAYOR NO SE ACTUALIZA.

Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	7	C
E	13	B
F	∞	-
G	11	C
H	∞	-
I	∞	-
J	∞	-

Se pasa C a la lista de Vertice no Visitados:
V=[A,C,B] NV=[D,E,F,G,H,I,J]

Se vuelve a repetir el proceso con el vertice no visitado de menor distancia, en este caso D y se vuelve a repetir el proceso

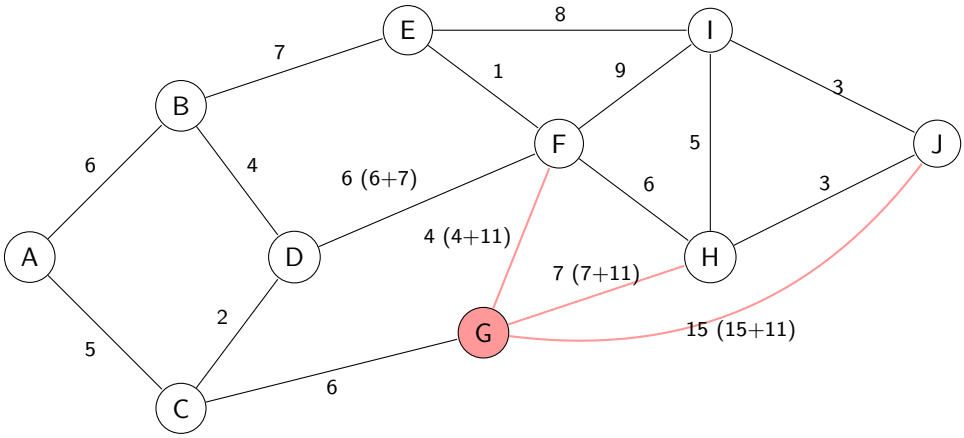


D->F =13 es menor que infinito se actualiza la distancia.

Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	7	C
E	13	B
F	13	D
G	11	C
H	∞	-
I	∞	-
J	∞	-

Se pasa C a la lista de Vertice no Visitados V=[A,C,B,D] NV=[E,F,G,H,I,J]

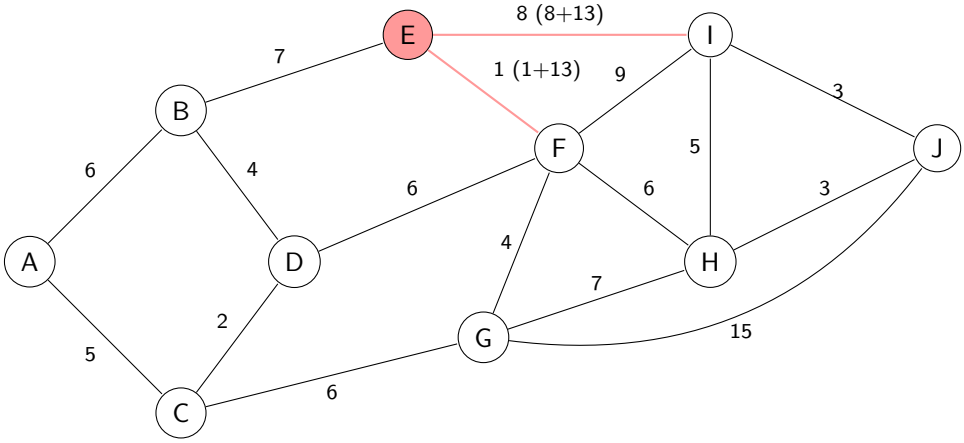
Se vuelve a repetir el proceso con el vertice no visitado de menor distancia, en este caso G y se vuelve a repetir el proceso



G->F =14 es MAYOR NO SE ACTUALIZA.
G->H =18 es menor que infinito se actualiza la distancia.
G->J =26 es menor que infinito se actualiza la distancia.

Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	7	C
E	13	B
F	13	D
G	11	C
H	18	G
I	∞	-
J	26	G

Se pasa C a la lista de Vértice no Visitados V=[A,C,B,D,G] NV=[E,F,H,I,J] Se vuelve a repetir el proceso con el vértice no visitado de menor distancia, en este caso E o F y se vuelve a repetir el proceso, en este caso se decide seguir con E



E->F =14 es MAYOR NO SE ACTUALIZA.
E->I =21 es menor que infinito se actualiza la distancia.

Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	7	C
E	13	B
F	13	D
G	11	C
H	18	G
I	21	E
J	26	G

Se pasa C a la lista de Vertice no Visitados V=[A,C,B,D,G,E] NV=[F,H,I,J] Se vuelve a repetir el proceso con el vertice no visitado de menor distancia, en este caso F y se vuelve a repetir el proceso

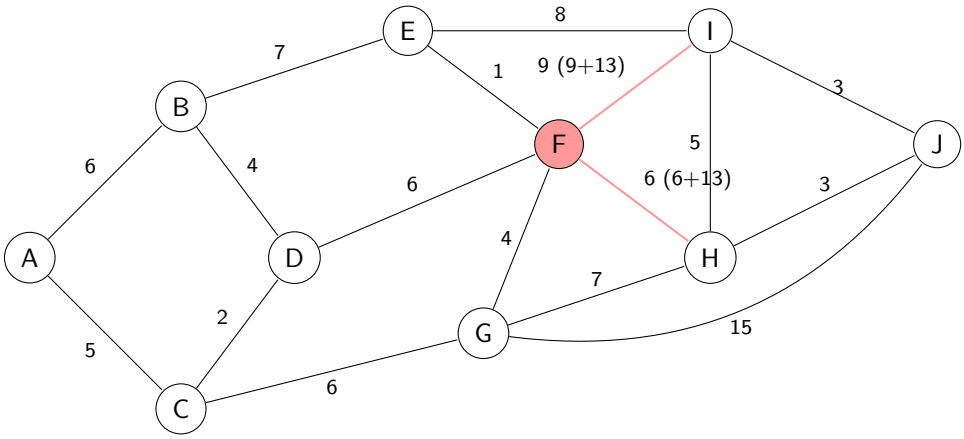


Figura 18

F->I =22 es MAYOR NO SE ACTUALIZA. F->H =19 es MAYOR NO SE ACTUALIZA.

Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	7	C
E	13	B
F	13	D
G	11	C
H	18	G
I	21	E
J	26	G

Se pasa C a la lista de Vertice no Visitados V=[A,C,B,D,G,E,F,] NV=[H,I,J]

Se vuelve a repetir el proceso con el vértice no visitado de menor distancia, en este caso H y se vuelve a repetir el proceso

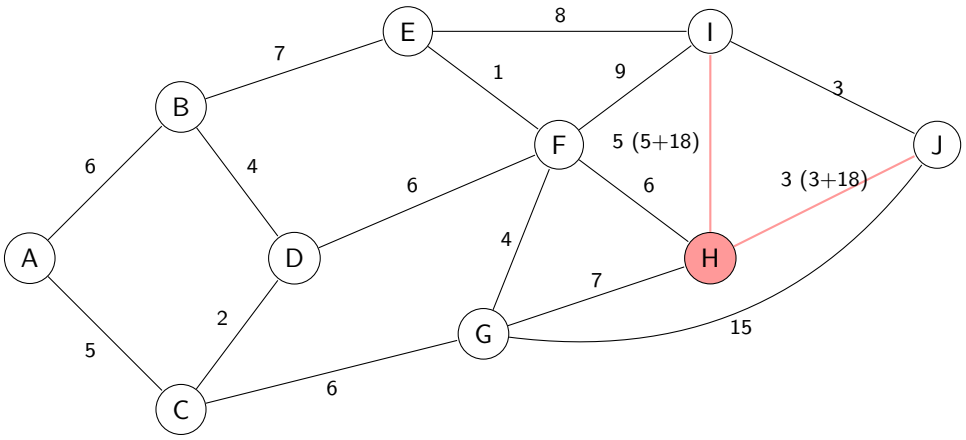


Figura 19

H->I = 23 es MAYOR NO SE ACTUALIZA. H->J = 21 es MAYOR NO SE ACTUALIZA.

Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	7	C
E	13	B
F	13	D
G	11	C
H	18	G
I	21	E
J	21	H

Se pasa C a la lista de Vertice no Visitados
V=[A,C,B,D,G,E,F,H]
NV=[I,J]

Se vuelve a repetir el proceso con el vértice no visitado de menor distancia, en este caso I o J y se vuelve a repetir el proceso, se toma el I

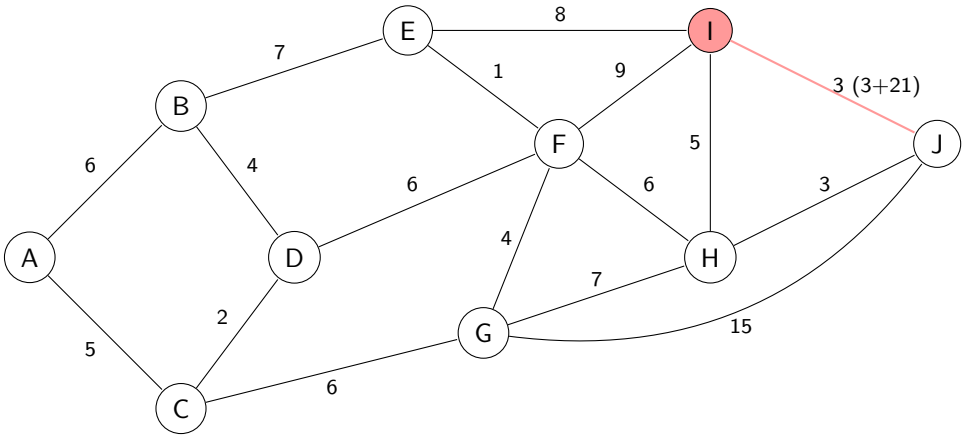


Figura 20

I->J = 23 es MAYOR NO SE ACTUALIZA.

Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	7	C
E	13	B
F	13	D
G	11	C
H	18	G
I	21	E
J	21	H

Se pasa C a la lista de Vertice no Visitados V=[A,C,B,D,G,E,F,I] NV=[J]

Se vuelve a repetir el proceso con el vértice no visitado de menor distancia, en este caso J y se vuelve a repetir el proceso.

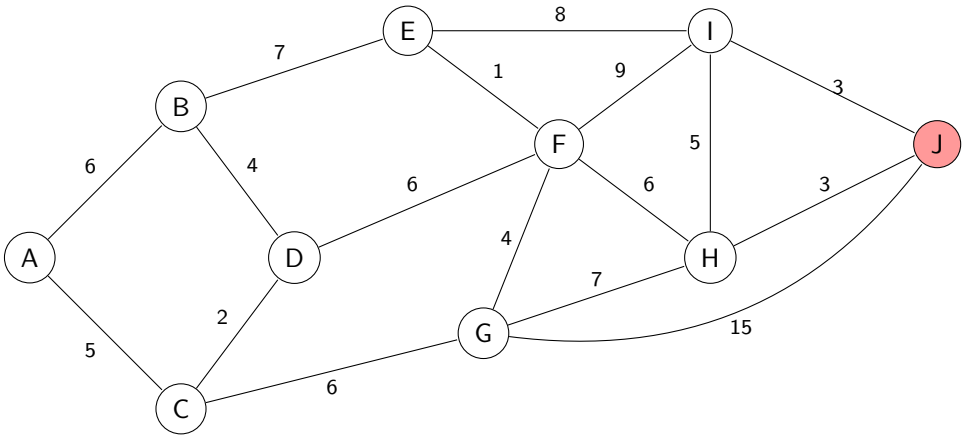


Figura 21

Desde el J no se va a ningun nodo por ende se pasa el nodo a la lista de visitados y se finaliza el algoritmo

Vértice	Distancia	V. Anterior
A	0	-
B	6	A
C	5	A
D	7	C
E	13	B
F	13	D
G	11	C
H	18	G
I	21	E
J	21	H

Se pasa C a la lista de Vertice no Visitados V=[A,C,B,D,G,E,F,I,J] NV=[]

una vez concluido el algoritmo se obtienen todos los caminos mínimos entre el Vertice A y todos los vertices del grafo, para hallarlos solo basta recorrea la tabla en forma inversa:

	Camino mínimo	
$A->B$	$A->B$	6
$A->C$	$A->C$	5
$A->D$	$A->C->D$	7
$A->E$	$A->B->E$	13
$A->F$	$A->C->D->F$	13
$A->G$	$A->C->G$	11
$A->H$	$A->C->G->H$	18
$A->I$	$A->B->E->I$	21
$A->J$	$A->C->G->H->J$	21

A continuación se mostrara el camino minimo entre el vertice A y el vertice J:

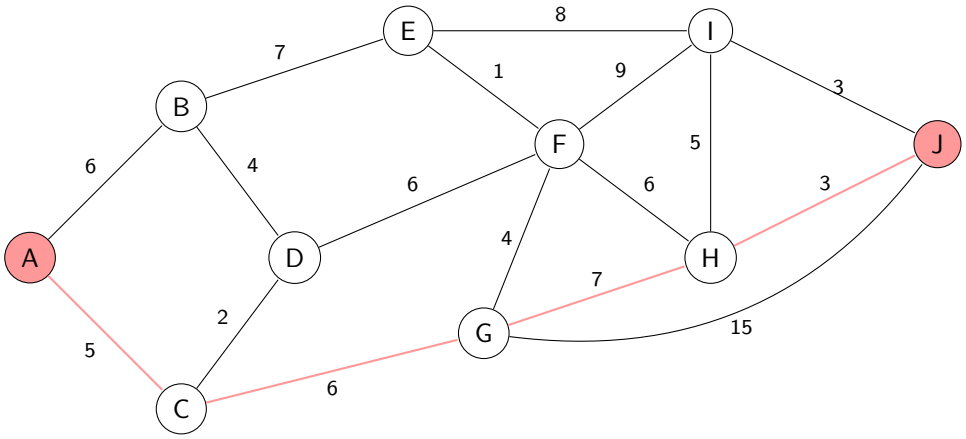


Figura 22: Camino mínimo entre el vértice A y el vértice J

3.4. Prim

El Algortimo de Prim es un algoritmo greedy que permite encontrar el mínimo spanning tree. Dado un grafo conexo y no dirigido, un árbol expansión o spanning tree de ese grafo es un subgrafo que tiene que ser un árbol y contener todos los vértices del grafo inicial. Cada arista tiene asignado un peso proporcional entre ellos, que es un número representativo de algún objeto, distancia, etc.; y se usa para asignar un peso total al árbol expansión mínimo computando la suma de todos los pesos de las aristas del árbol en cuestión. Un árbol expansión mínimo o un árbol expandido mínimo es un árbol expansión que pesa menos o igual que otros árboles de expansión. Todo grafo tiene un bosque de expansión mínimo.

3.4.1. El Algoritmo

El algoritmo es sencillo:

1. se consideran todos los nodos fuera del árbol.
2. se selecciona arbitrariamente un nodo del grafo
3. se agrega al arbol el nodo adyacente con minimo peso, se lo marca como visitado.
4. se repite 3 hasta que todos los nodos se hayan visitado.

A continuación se probará el algoritmo con el grafo de la figura comenzando por el nodo E

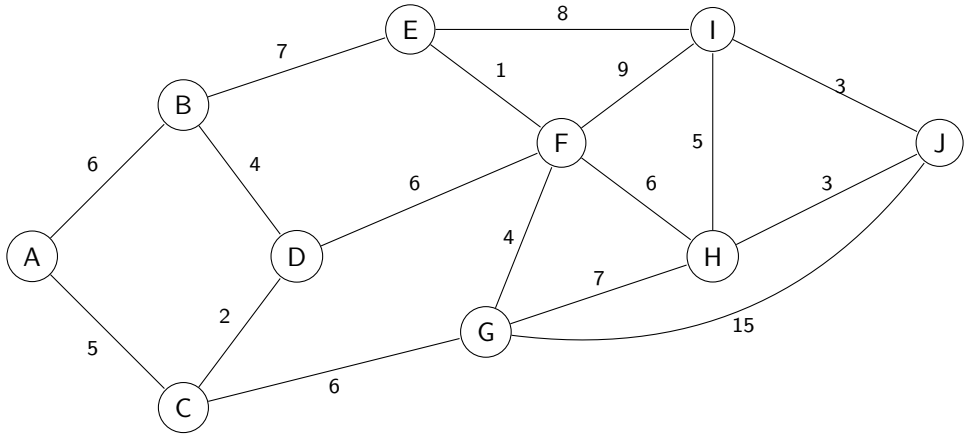


Figura 23

Paso 1

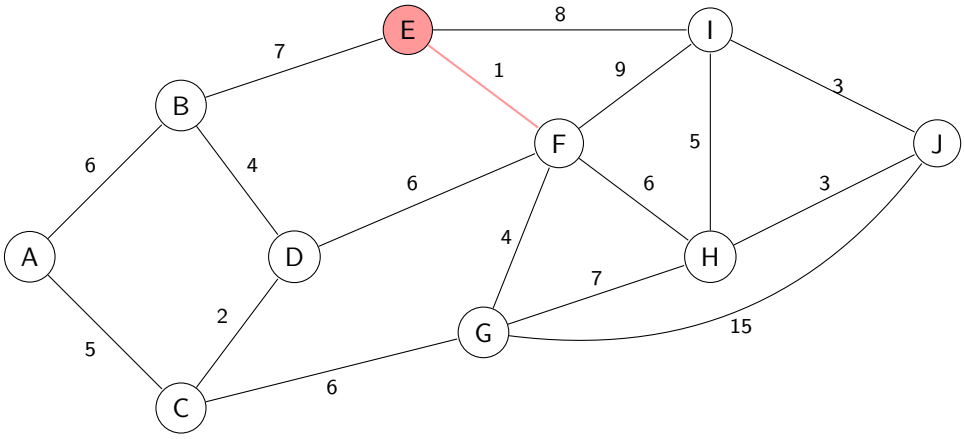


Figura 24

Se selecciona la arista con peso minimo que conecta a E con F, se marca a F como nodo visitado y perteneciente al arbol. Visitados=[E,F] NV=[A,B,C,D,G,H,I,J]

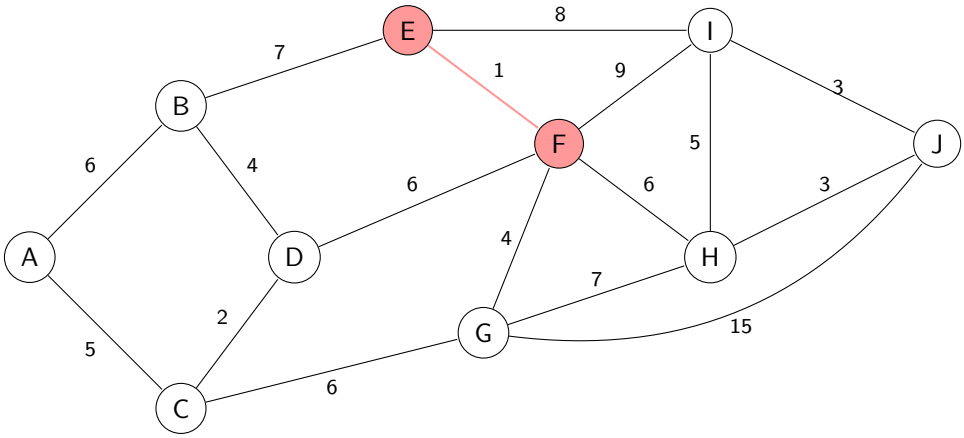


Figura 25

Paso 2 De los dos nodos del arbol se buscar el adyacente no visitado con menor peso. en este caso la arista que conecta al vertice G con peso 4.

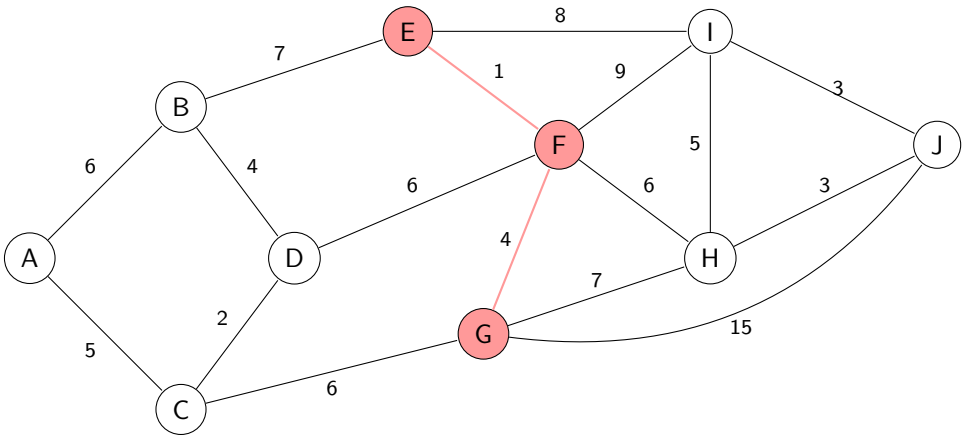


Figura 26

Visitados=[E,F,G] NV=[A,B,C,D,H,I,J]

Paso 3 De los dos nodos del árbol se buscar el adyacente no visitado con menor peso. En este caso existen tres posibles elecciones: todas con peso 6, las 2 aristas del Vertice F y la arista del vertice G. Arbitrariamente se selecciona la arista de peso 6 hacia el vertice D.

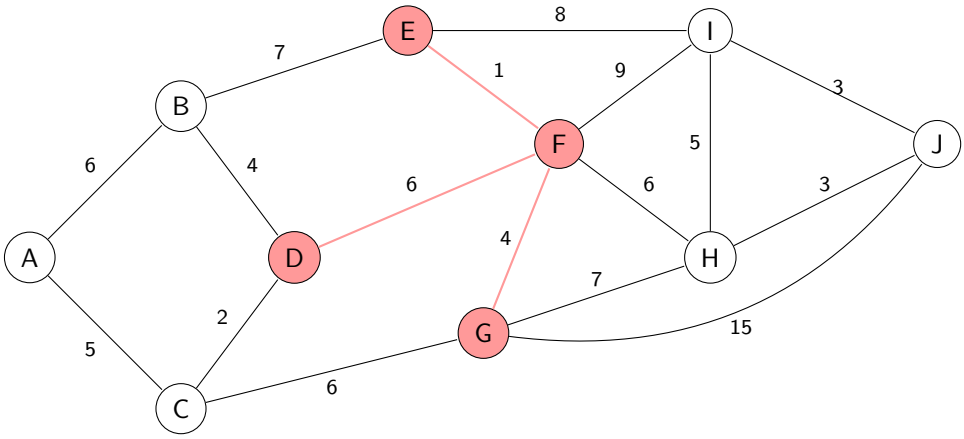


Figura 27

Visitados=[E,F,G,D] NV=[A,B,C,H,I,J]

Paso 4 De los dos nodos del árbol se buscar el adyacente no visitado con menor peso. En este caso seleccionamos la arista con peso 2 que conecta al nodo C.

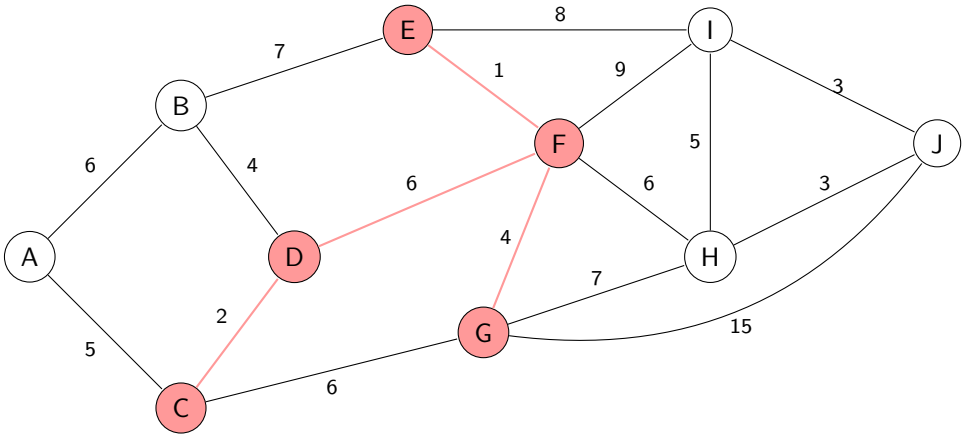


Figura 28

Visitados=[E,F,G,D,C] NV=[A,B,H,I,J]

Paso 5 De los dos nodos del árbol se buscar el adyacente no visitado con menor peso. En este caso se selecciona el nodo B con la arista con peso 4.

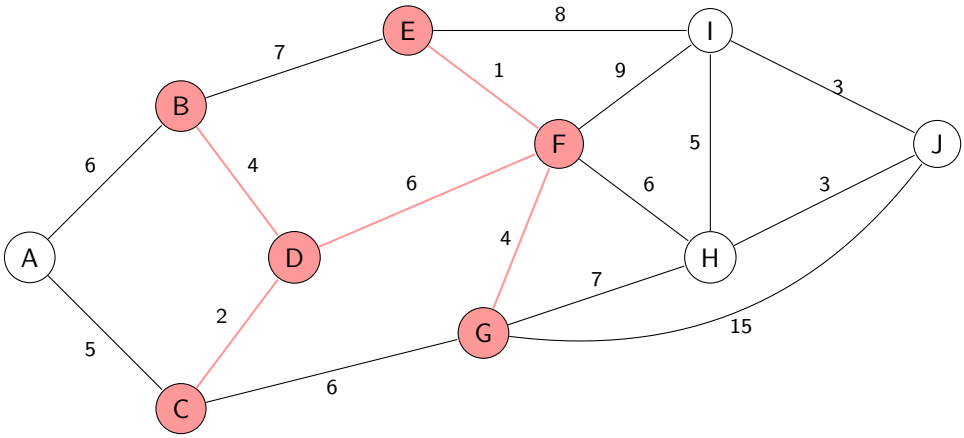


Figura 29

Visitados=[E,F,G,D,C,B] NV=[A,H,I,J]

Paso 6 De los dos nodos del árbol se buscar el adyacente no visitado con menor peso. En este caso se selecciona el nodo A con la arista con peso 5, que parte swl nodo C.

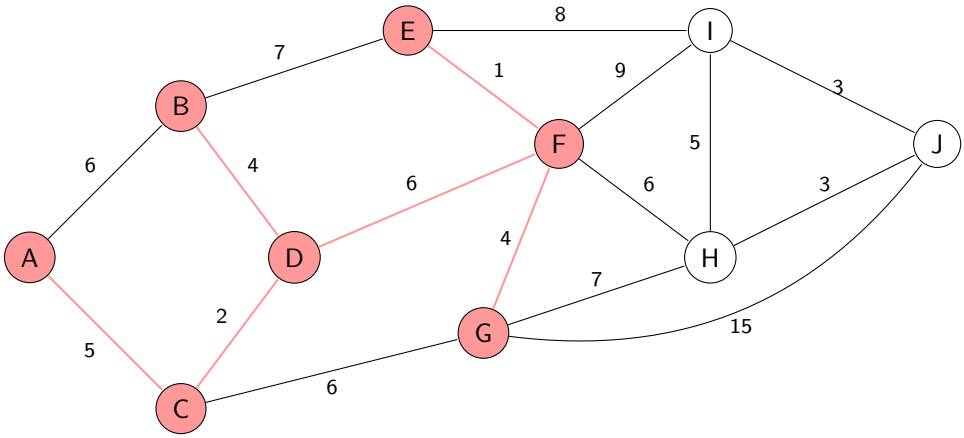


Figura 30

Visitados=[E,F,G,D,C,B,A] NV=[H,I,J]

Paso 7 De los dos nodos del árbol se buscar el adyacente no visitado con menor peso. En este caso se selecciona el nodo H con la arista con peso 6, que parte del nodo F.

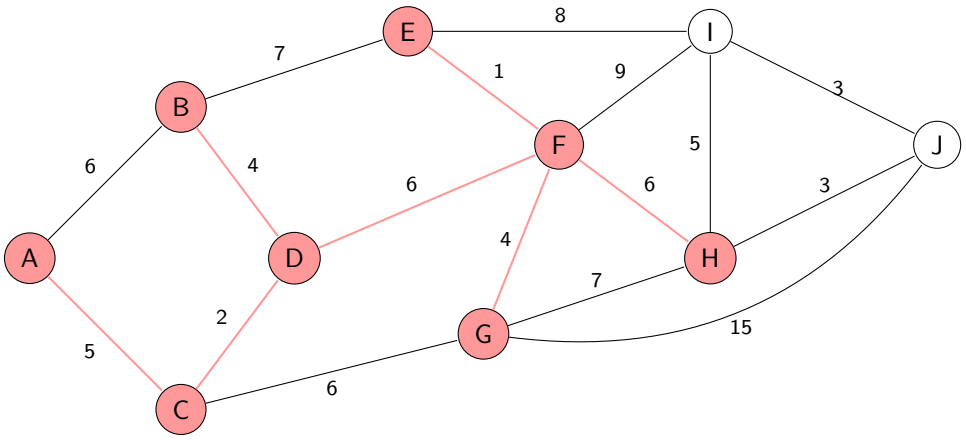


Figura 31

Visitados=[E,F,G,D,C,B,A,H] NV=[I,J]

Paso 8 De los dos nodos del árbol se buscar el adyacente no visitado con menor peso. En este caso se selecciona el nodo J con la arista con peso 3, que parte del nodo H.

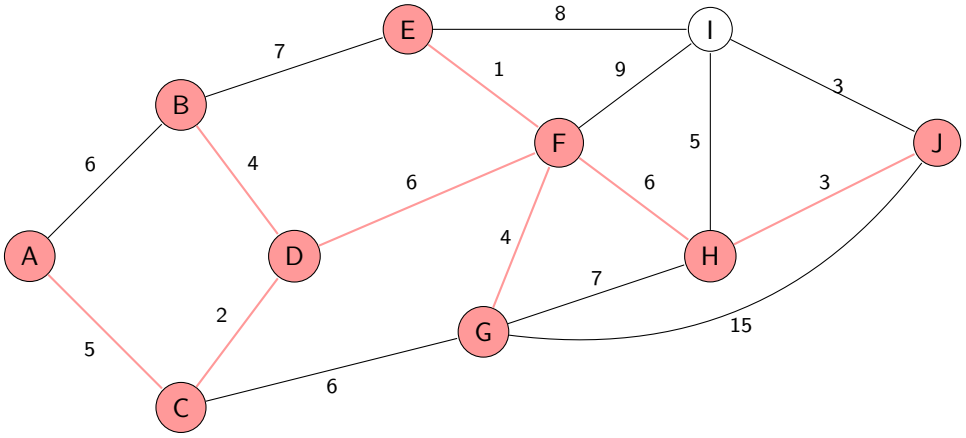


Figura 32

Visitados=[E,F,G,D,C,B,A,H,J] NV=[I]

Paso 9 De los dos nodos del árbol se buscar el adyacente no visitado con menor peso. En este caso se selecciona el nodo I con la arista con peso 3, que parte del nodo J.

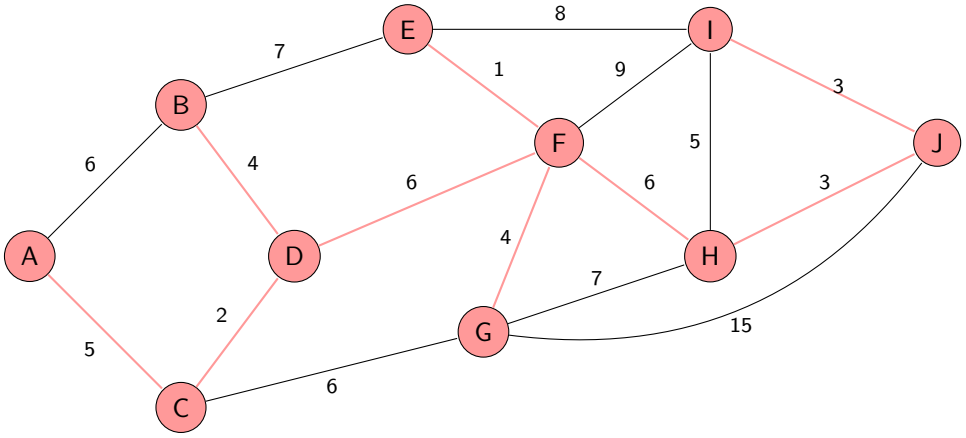


Figura 33

Visitados=[E,F,G,D,C,B,A,H,J,I] NV=[]

Todos los nodos han sido visitados y se ha obtenido el arbol de extensión o spanning tree mínimo del grafo partiendo del nodo E:

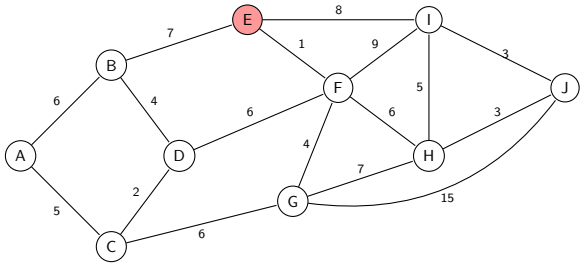


Figura 34

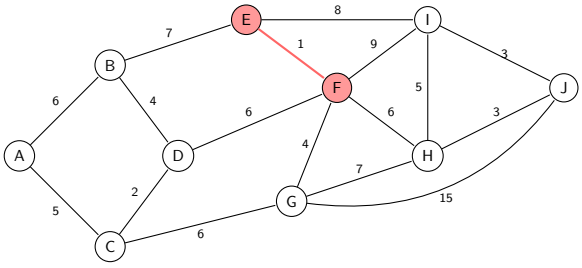


Figura 35

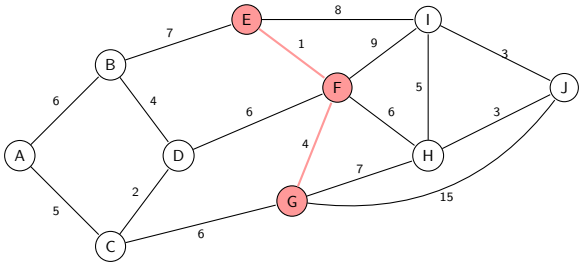


Figura 36

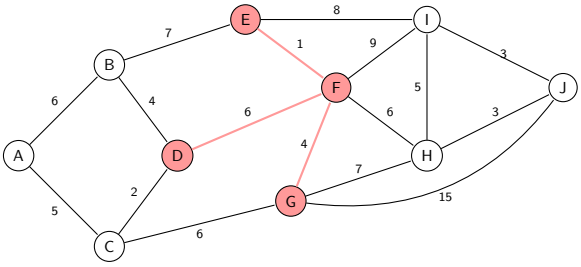


Figura 37

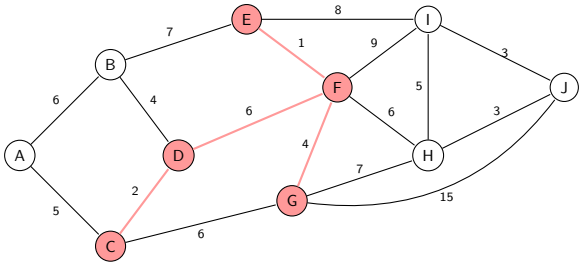


Figura 38

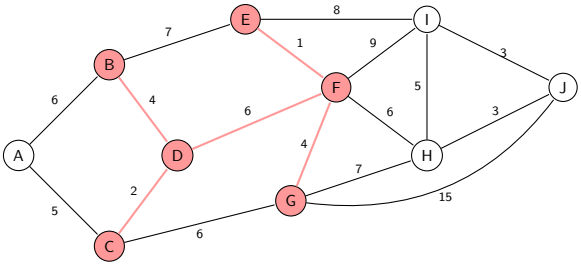


Figura 39

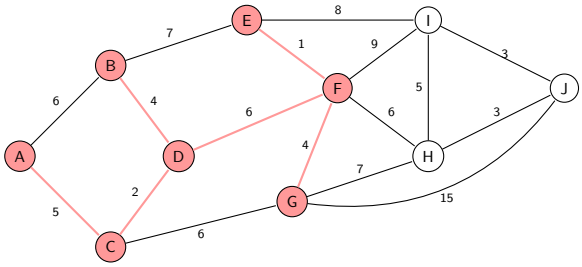


Figura 40

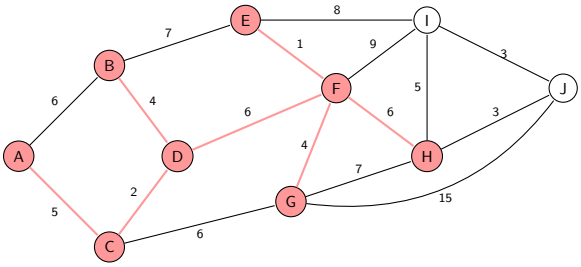


Figura 41

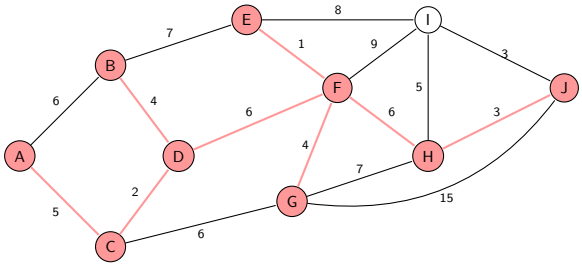


Figura 42

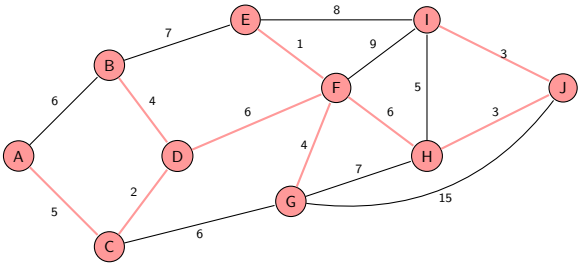


Figura 43

3.5. Kruskal

El algoritmo de Kruskal es un algoritmo de la teoría de grafos para encontrar un árbol de expansión mínimo en un grafo conexo y ponderado. Es decir, busca un subconjunto de aristas que, formando un árbol, incluyen todos los vértices y donde el valor de la suma de todas las aristas del árbol es el mínimo. Si el grafo no es conexo, entonces busca un bosque expandido mínimo (un árbol expandido mínimo para cada componente conexa). Este algoritmo toma su nombre de Joseph Kruskal, quien lo publicó por primera vez en 1956. Otros algoritmos que sirven para hallar el árbol de expansión mínima o árbol recubridor mínimo es el algoritmo de Prim, el algoritmo del borrador inverso y el algoritmo de Boruvka.

3.5.1. El Algoritmo

El algoritmo de Kruskal es un ejemplo de algoritmo greedy que funciona de la siguiente manera:

- se crea un bosque B (un conjunto de árboles), donde cada vértice del grafo es un árbol separado
- se crea un conjunto C que contenga a todas las aristas del grafo mientras C es no vacío
    - eliminar una arista de peso mínimo de C
    - si esa arista conecta dos árboles diferentes se añade al bosque, combinando los dos árboles en un solo árbol
    - en caso contrario, se desecha la arista

Al terminar el algoritmo, el bosque tiene un solo componente, el cual forma un árbol de expansión mínimo del grafo. En un árbol de expansión mínimo se cumple : la cantidad de aristas del árbol es la cantidad de nodos menos uno (1).

A continuación se probará el algoritmo con el grafo de la figura:

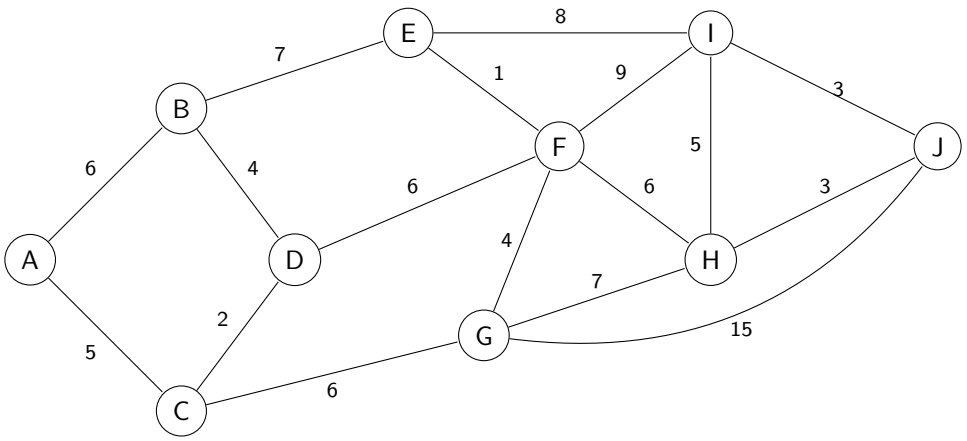


Figura 44

Paso 1 Se crea un bosque B (un conjunto de árboles), donde cada vértice del grafo es un árbol separado:

Bosque =[A,B,C,D,E,F,G,H,I,J,]

Se crea un conjunto C que contenga a todas las aristas del grafo:

Peso	1	2	3	3	4	4	5	5	6	6	6	6	7	7	8	9	15
Aristas	E-F	D-C	I-J	H-J	B-D	F-G	A-C	I-J	D-F	A-B	F-H	C-G	B-E	G-H	E-I	F-I	G-J

Se toma la arista con menor peso y se verifica si esta une dos vértices en arboles distintos, en este caso la arista que une E y F de peso 1

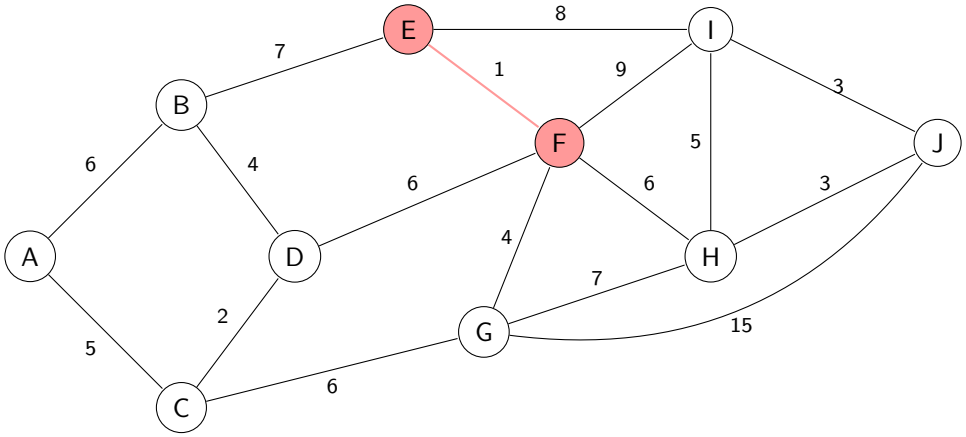


Figura 45

En este caso el se genera un nuevo árbol en el bosque en el cual se une el vértice E y el F [A,B,C,D,E-F,G,H,I,J,]

	✓																
Peso	1	2	3	3	4	4	5	5	6	6	6	6	7	7	8	9	15
Aristas	E-F	D-C	I-J	H-J	B-D	F-G	A-C	I-J	D-F	A-B	F-H	C-G	B-E	G-H	E-I	F-I	G-J

Paso 2 Se vuelve a tomar la arista con menor peso y se verifica si esta une dos vértices en arboles distintos, en este caso la arista debe tener peso 2 , y une los vértices C y D

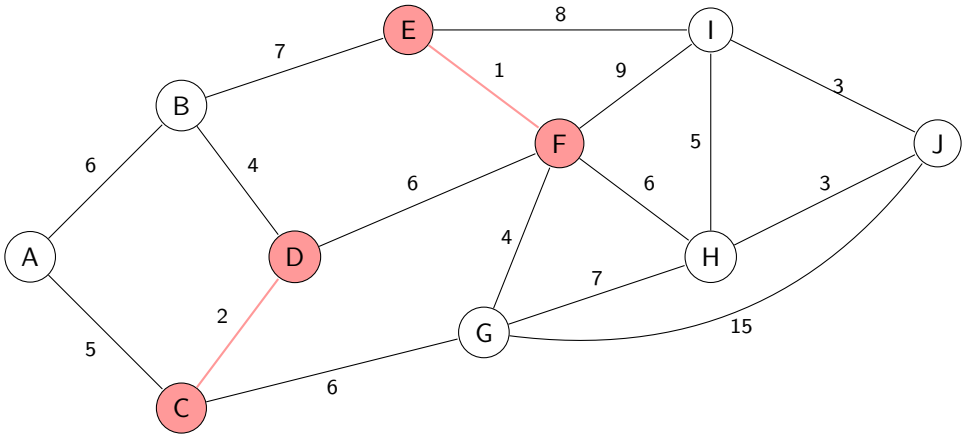


Figura 46

Dado que esta arista conecta dos árboles diferentes se añade al bosque, combinando los dos árboles en un solo árbol. En este caso el se genera un nuevo árbol en el bosque en el cual se une el vértice C y el D

[A,B,C-D,E-F,G,H,I,J]

	✓	✓															
Peso	1	2	3	3	4	4	5	5	6	6	6	6	7	7	8	9	15
Aristas	E-F	D-C	I-J	H-J	B-D	F-G	A-C	I-J	D-F	A-B	F-H	C-G	B-E	G-H	E-I	F-I	G-J

Paso 3 Se vuelve a tomar la arista con menor peso y se verifica si esta une dos vértices en arboles distintos, en este caso la arista debe tener peso 3 , y une los vértices I y J (al existir más de una la elección es arbitraria)

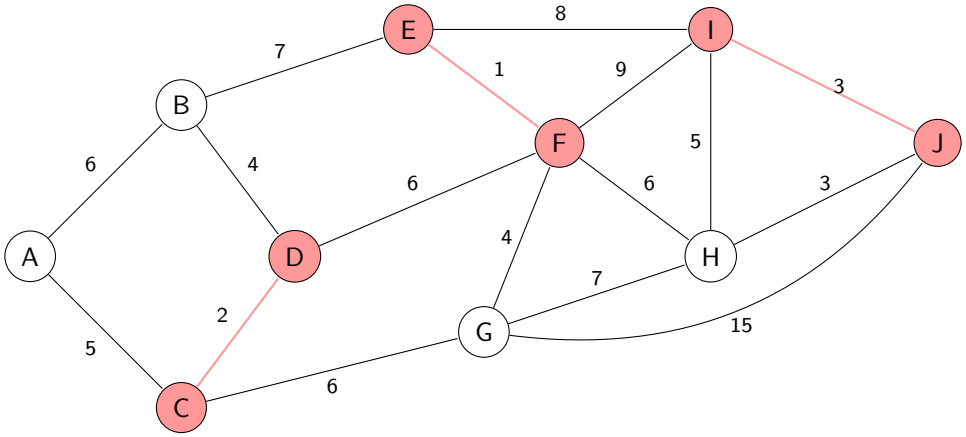


Figura 47

Dado que esta arista conecta dos árboles diferentes se añade al bosque, combinando los dos árboles en un solo árbol. En este caso el se genera un nuevo árbol en el bosque en el cual se une el vértice I y el J

[A,B,C-D,E-F,G,H,I-J]

	✓	✓	✓														
Peso	1	2	3	3	4	4	5	5	6	6	6	6	7	7	8	9	15
Aristas	E-F	D-C	I-J	H-J	B-D	F-G	A-C	I-J	D-F	A-B	F-H	C-G	B-E	G-H	E-I	F-I	G-J

Paso 4 Se vuelve a tomar la arista con menor peso y se verifica si esta une dos vértices en arboles distintos, en este caso la arista debe tener peso 3, y une los vértices J y H

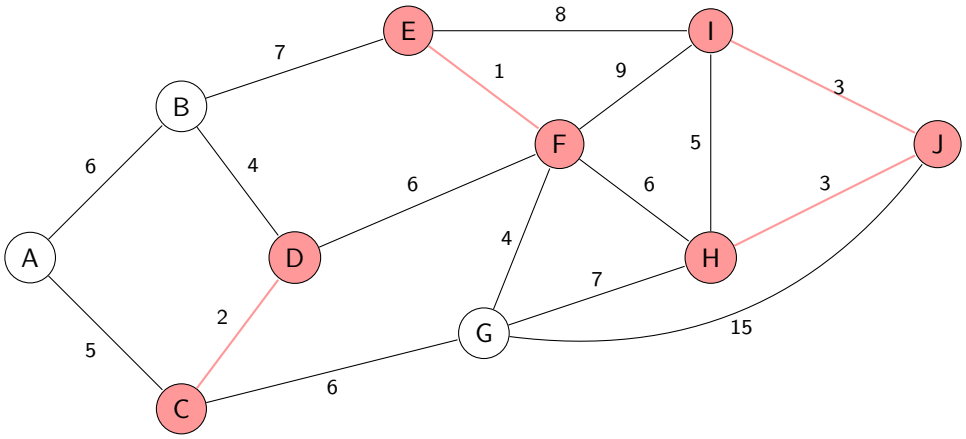


Figura 48

Dado que esta arista conecta dos árboles diferentes se añade al bosque, combinando los dos árboles en un solo árbol. En este caso el se genera un nuevo árbol en el bosque en el cual se une el vértice J y el H

[A,B,C-D,E-F,G,H-I-J]

	✓	✓	✓	✓													
Peso	1	2	3	3	4	4	5	5	6	6	6	6	7	7	8	9	15
Aristas	E-F	D-C	I-J	H-J	B-D	F-G	A-C	I-J	D-F	A-B	F-H	C-G	B-E	G-H	E-I	F-I	G-J

Paso 5 Se vuelve a tomar la arista con menor peso y se verifica si esta une dos vértices en arboles distintos, en este caso la arista debe tener peso 4 , y une los vértices B y D

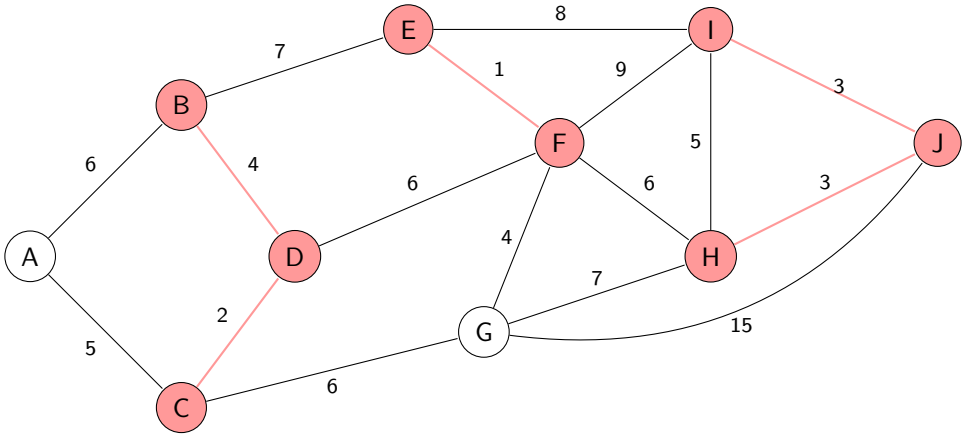


Figura 49

Dado que esta arista conecta dos árboles diferentes se añade al bosque, combinando los dos árboles en un solo árbol. En este caso el se genera un nuevo árbol en el bosque en el cual se une el vértice B y el D
[A,B-C-D,E-F,G,H-I-J]

	✓	✓	✓	✓	✓												
Peso	1	2	3	3	4	4	5	5	6	6	6	6	7	7	8	9	15
Aristas	E-F	D-C	I-J	H-J	B-D	F-G	A-C	I-J	D-F	A-B	F-H	C-G	B-E	G-H	E-I	F-I	G-J

Paso 6 Se vuelve a tomar la arista con menor peso y se verifica si esta une dos vértices en arboles distintos, en este caso la arista debe tener peso 4 , y une los vértices F y G

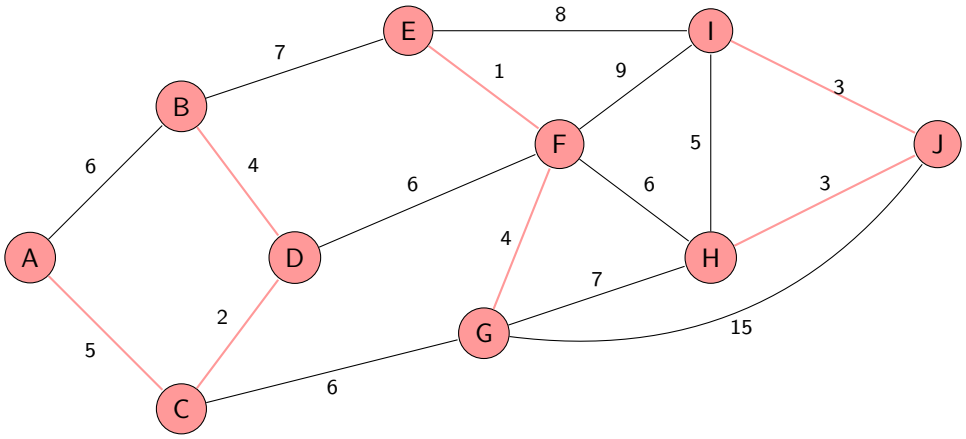


Figura 50

Dado que esta arista conecta dos árboles diferentes se añade al bosque, combinando los dos árboles en un solo árbol. En este caso el se genera un nuevo árbol en el bosque en el cual se une el vértice B y el D
[A,B-C-D,E-F-G,H-I-J]

	✓	✓	✓	✓	✓	✓											
Peso	1	2	3	3	4	4	5	5	6	6	6	6	7	7	8	9	15
Aristas	E-F	D-C	I-J	H-J	B-D	F-G	A-C	I-J	D-F	A-B	F-H	C-G	B-E	G-H	E-I	F-I	G-J

Paso 7 Se vuelve a tomar la arista con menor peso y se verifica si esta une dos vértices en arboles distintos, en este caso la arista debe tener peso 5 , y une los vértices A y C

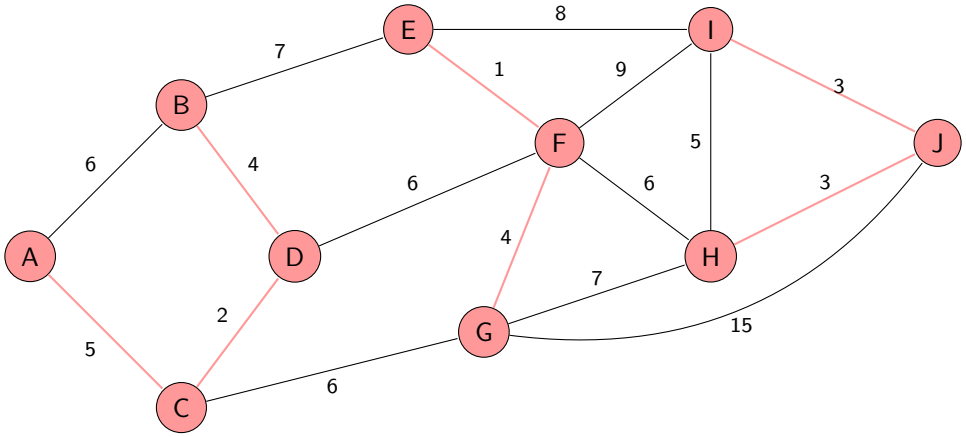


Figura 51

Dado que esta arista conecta dos árboles diferentes se añade al bosque, combinando los dos árboles en un solo árbol. En este caso el se genera un nuevo árbol en el bosque en el cual se une el vértice A y el C
[A-B-C-D,E-F-G,H-I-J]

	✓	✓	✓	✓	✓	✓	✓										
Peso	1	2	3	3	4	4	5	5	6	6	6	6	7	7	8	9	15
Aristas	E-F	D-C	I-J	H-J	B-D	F-G	A-C	I-J	D-F	A-B	F-H	C-G	B-E	G-H	E-I	F-I	G-J

Paso 8 Se vuelve a tomar la arista con menor peso y se verifica si esta une dos vértices en arboles distintos, en este caso la arista debe tener peso 5 , y une los vértices H y I

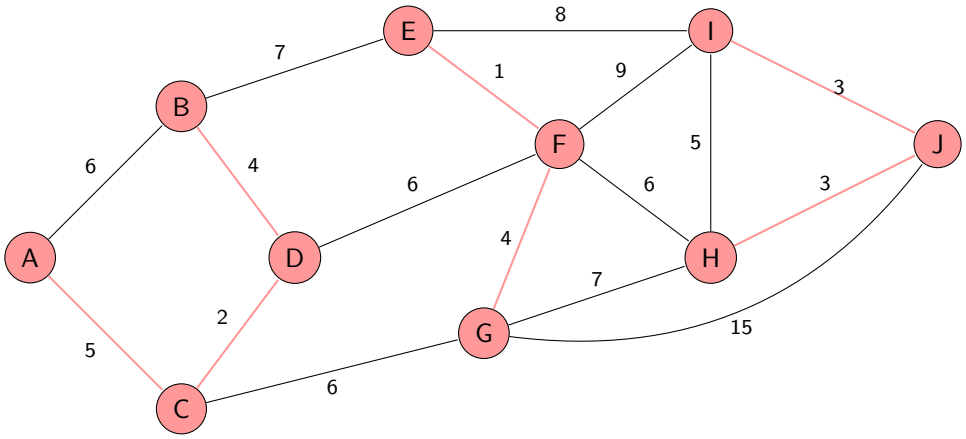


Figura 52

Dado que esta arista **no** conecta dos árboles diferentes, se pasa de largo y se desecha.
[A-B-C-D,E-F-G,H-I-J]

	✓	✓	✓	✓	✓	✓	✓	✗									
Peso	1	2	3	3	4	4	5	5	6	6	6	6	7	7	8	9	15
Aristas	E-F	D-C	I-J	H-J	B-D	F-G	A-C	I-J	D-F	A-B	F-H	C-G	B-E	G-H	E-I	F-I	G-J

Paso 9 Se vuelve a tomar la arista con menor peso y se verifica si esta une dos vértices en arboles distintos, en este caso la arista debe tener peso 6: existen cuatro posibles elecciones A-B, D-F, F-H o C-G. Se toma la Arista A-B

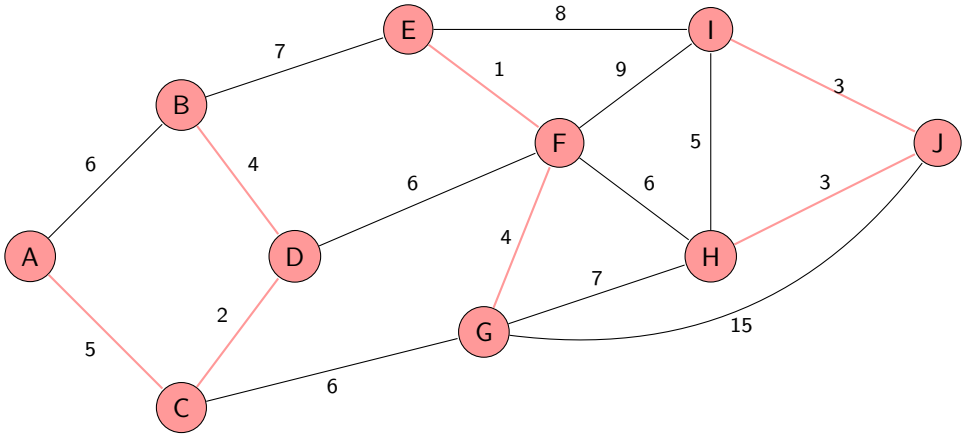


Figura 53

Dado que esta arista **no** conecta dos árboles diferentes, se pasa de largo y se desecha.
[A-B-C-D,E-F-G,H-I-J]

	✓	✓	✓	✓	✓	✓	✓	✓		✗							
Peso	1	2	3	3	4	4	5	5	6	6	6	6	7	7	8	9	15
Aristas	E-F	D-C	I-J	H-J	B-D	F-G	A-C	I-J	D-F	A-B	F-H	C-G	B-E	G-H	E-I	F-I	G-J

Paso 10 Se vuelve a tomar la arista con menor peso y se verifica si esta une dos vértices en arboles distintos, en este caso la arista debe tener peso 6: existen tres posibles elecciones D-F, F-H o C-G. Se toma la Arista D-F

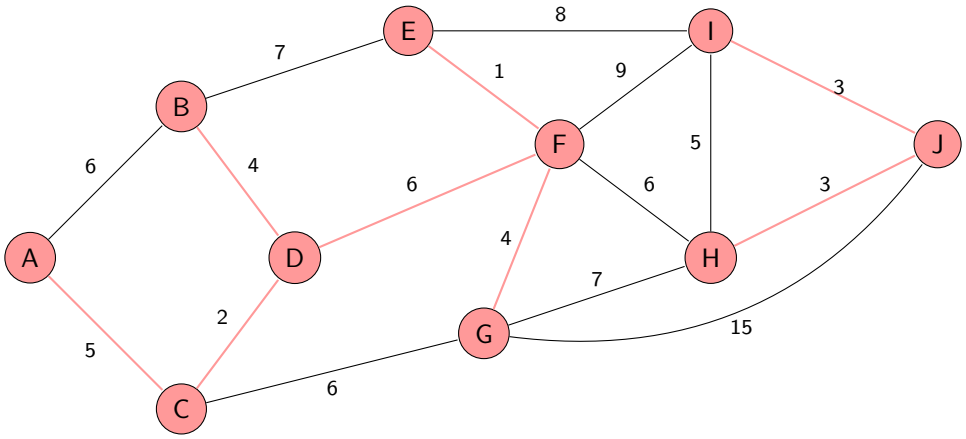


Figura 54

Dado que esta arista conecta dos árboles diferentes se añade al bosque, combinando los dos árboles en un solo árbol. En este caso el se genera un nuevo árbol en el bosque en el cual se une el vértice D y el F
[A-B-C-D-E-F-G,H-I-J]

	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗							
Peso	1	2	3	3	4	4	5	5	6	6	6	6	7	7	8	9	15
Aristas	E-F	D-C	I-J	H-J	B-D	F-G	A-C	I-J	D-F	A-B	F-H	C-G	B-E	G-H	E-I	F-I	G-J

Paso 11 Se vuelve a tomar la arista con menor peso y se verifica si esta une dos vértices en arboles distintos, en este caso la arista debe tener peso 6: existen dos posibles elecciones , F-H o C-G todas las aristas unirían arboles distintos, se debe elegir una , en etse caso F-H

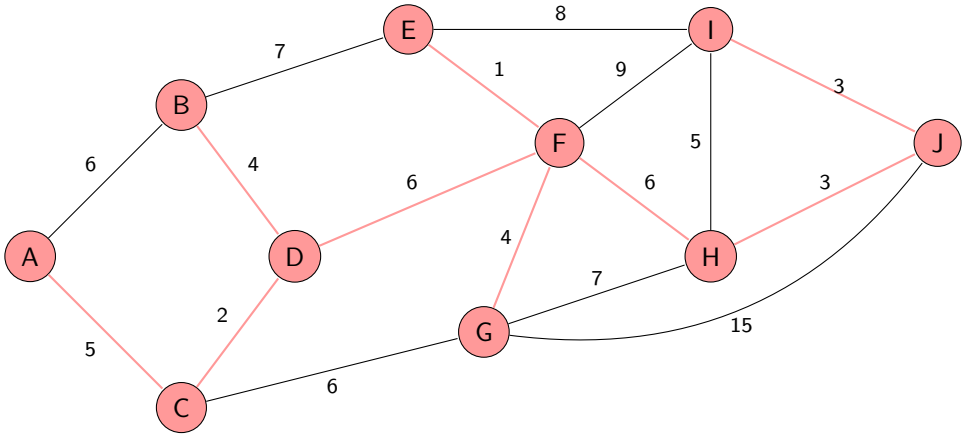


Figura 55

Dado que esta arista conecta dos árboles diferentes se añade al bosque, combinando los dos árboles en un solo árbol. En este caso el se genera un nuevo árbol en el bosque en el cual se une el vértice D y el F

[A-B-C-D-E-F-G-H-I-J]

	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓						
Peso	1	2	3	3	4	4	5	5	6	6	6	6	7	7	8	9	15
Aristas	E-F	D-C	I-J	H-J	B-D	F-G	A-C	I-J	D-F	A-B	F-H	C-G	B-E	G-H	E-I	F-I	G-J

Paso 12 Como en este caso sólo existe un bosque con un único árbol, el algoritmo termina, desechando todas las demás aristas, que no forman parte del **minimun spanning tree**.

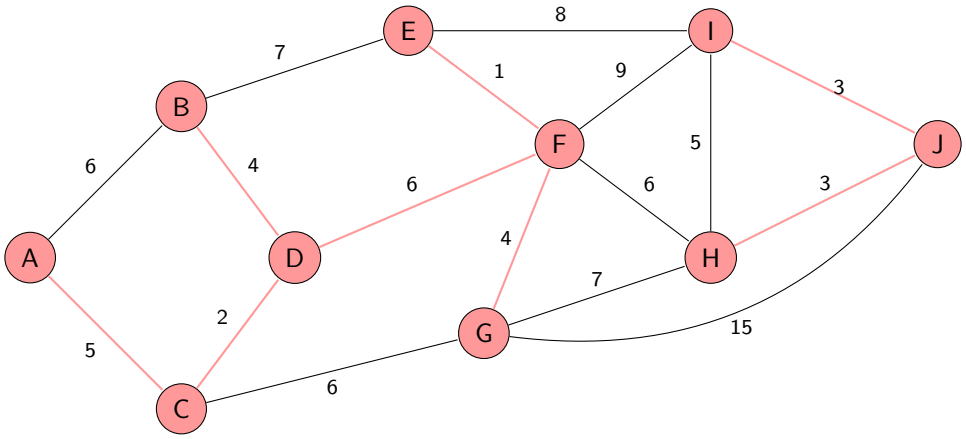


Figura 56

	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗
Peso	1	2	3	3	4	4	5	5	6	6	6	6	7	7	8	9	15
Aristas	E-F	D-C	I-J	H-J	B-D	F-G	A-C	I-J	D-F	A-B	F-H	C-G	B-E	G-H	E-I	F-I	G-J

Con lo cual queda un único árbol que es el spanning tree minimo:

[A-B-C-D-E-F-G-H-I-J]

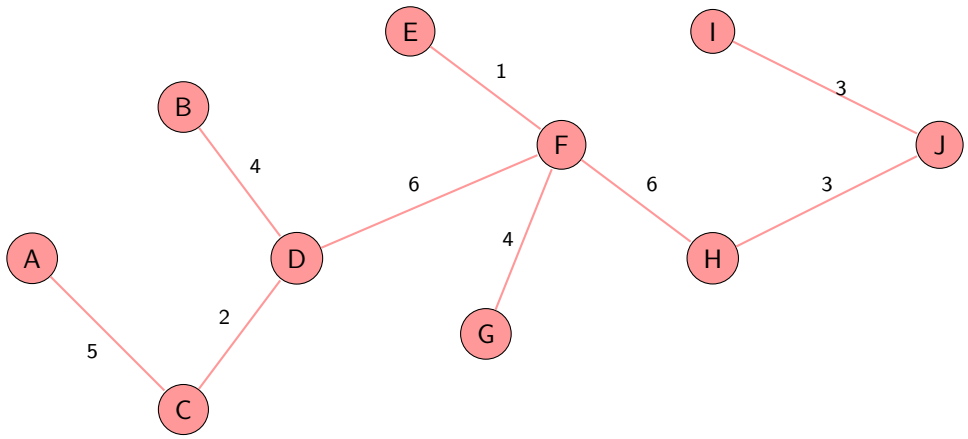


Figura 57

3.6. FloyD-Warshall

Este algoritmo encuentra la distancia mínima a todos los los conjuntos de pares de vértices, para ello calcula las distancia entre los pares de vertices pasando por un intermedio.

Referencias