Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Estructura de Datos

Semana 14

Estructuras de datos no lineales

01

Grafos – árboles de expansión
Algoritmos de Prim y Kruskal

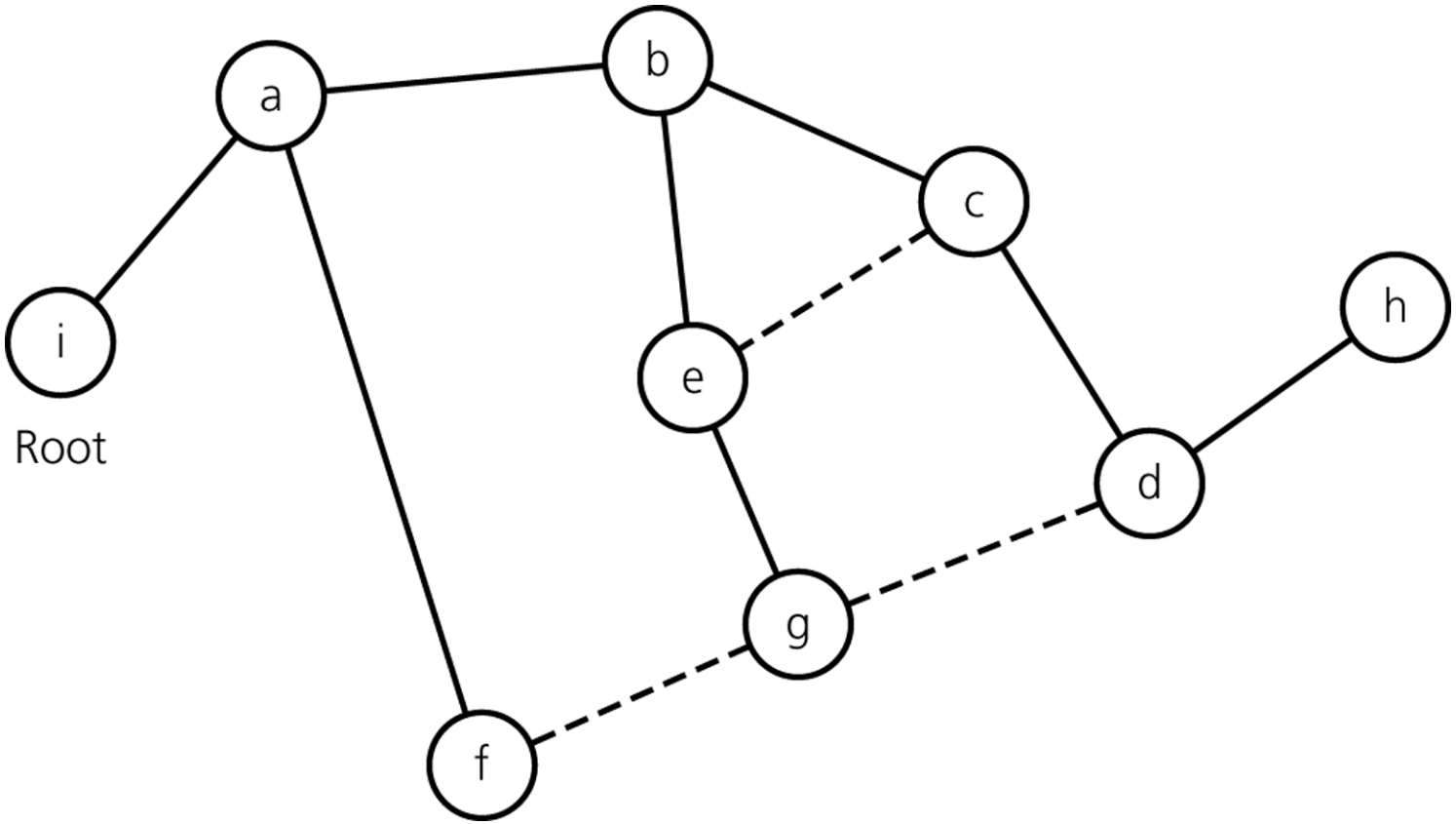
Agenda

- TAD Grafo Implementación Diccionario
- Recorridos
- Algoritmo de camino mínimo

Árboles de expansión

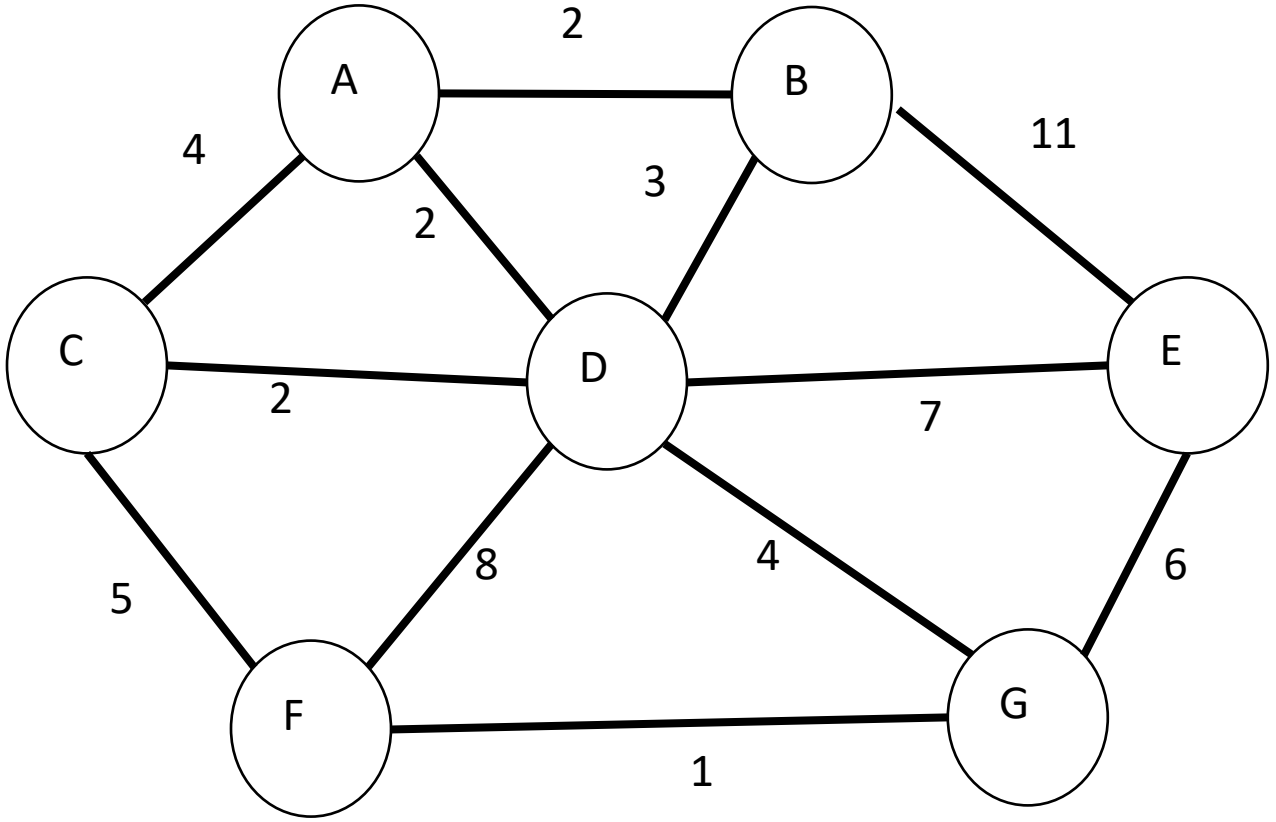
- Un árbol es un tipo especial de grafo no dirigido.
- Es decir, un **árbol** es un grafo no dirigido y conectado sin ciclos.
- Todos los árboles son grafos, no todos los grafos son árboles.
- Un árbol de expansion ***(spanning tree)*** de un grafo G conectado no dirigido es un subgrafo de G que contiene todos los vértices de G y suficientes aristas para formar un árbol.
- Puede haber varios árboles de expansión para un grafo determinado.
- Si tenemos un grafo no dirigido conectado con ciclos, eliminamos las aristas hasta que no haya ciclos para obtener un árbol de expansión.

Un árbol de expansión



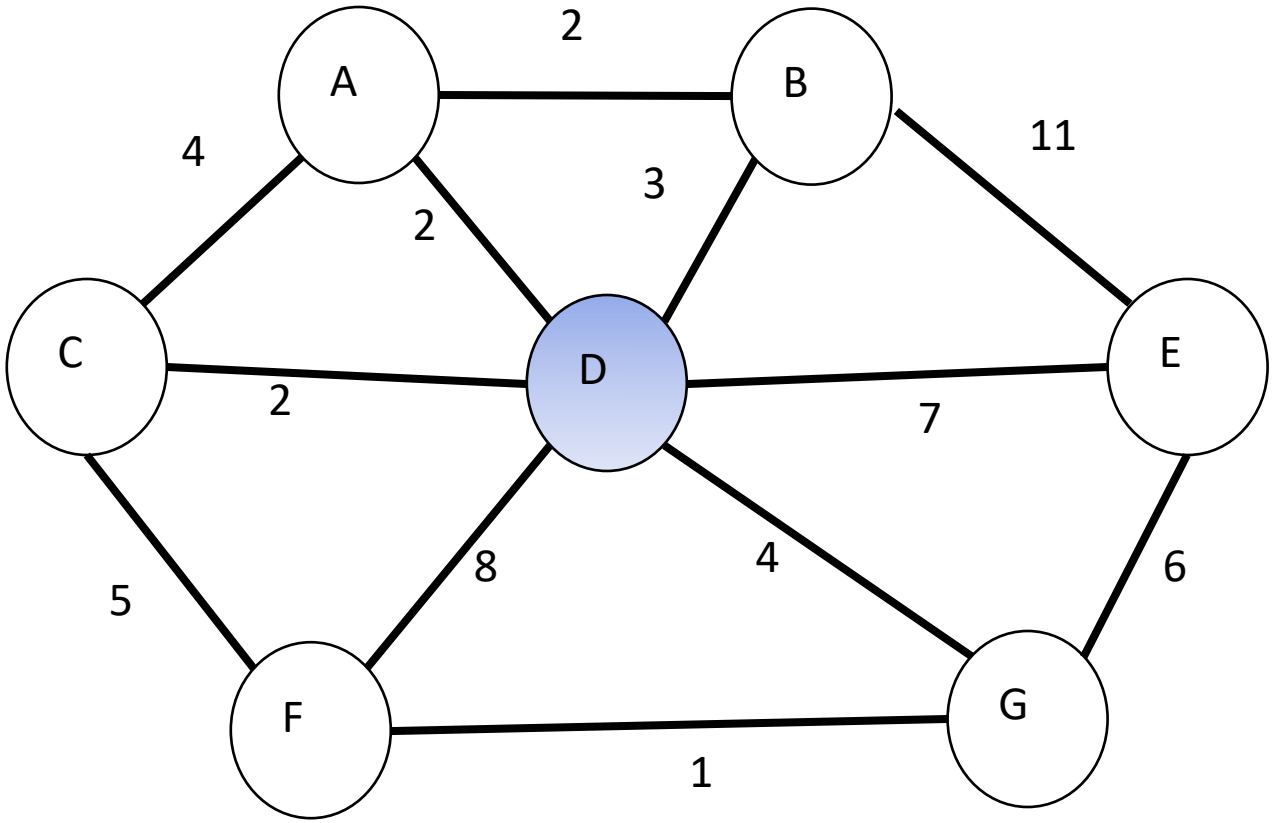
Elimine las líneas discontinuas para obtener un árbol de expansión

Árboles de expansión de coste mínimo



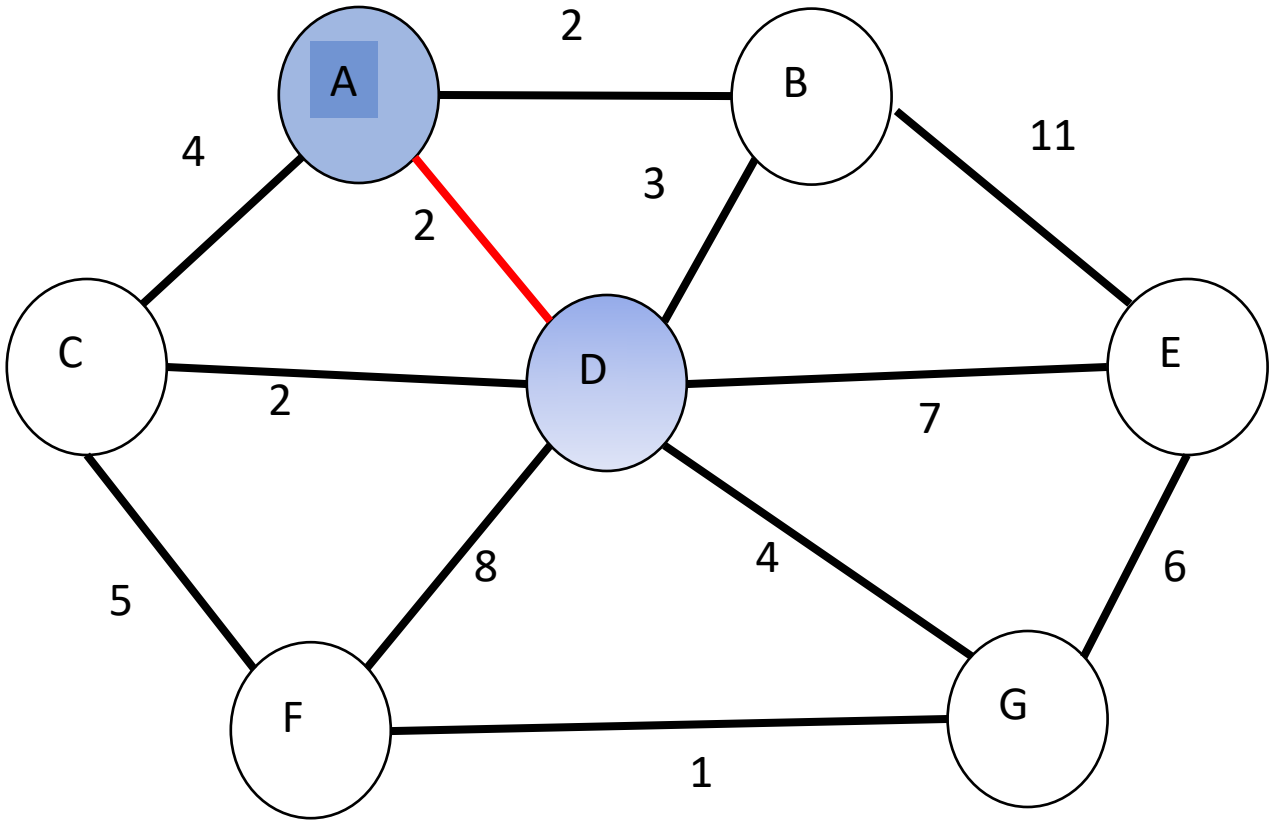
- **Problema:** conectar todos los nodos con el menor coste total.
- **Solución:** algoritmos clásicos de Prim y Kruskal.

Algoritmo de Prim



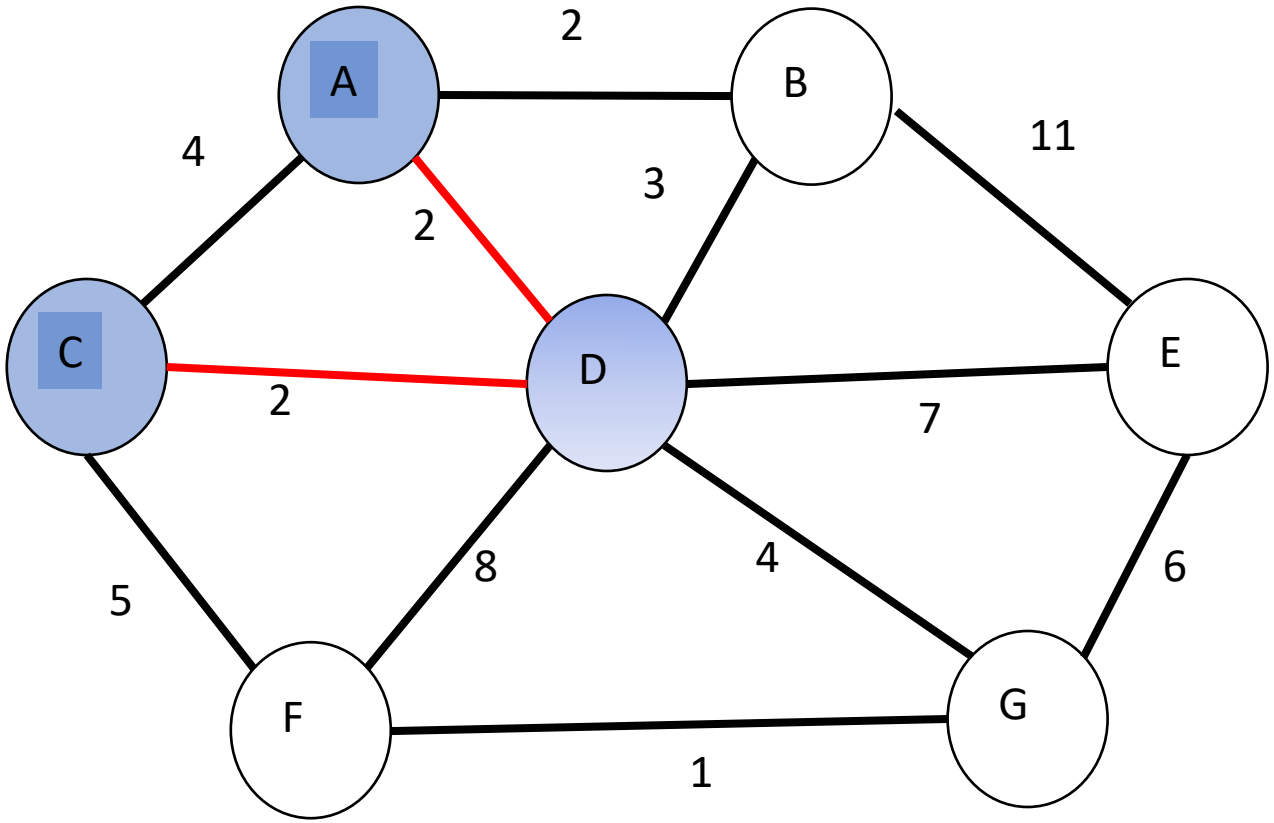
Elija D como raíz

Algoritmo de Prim



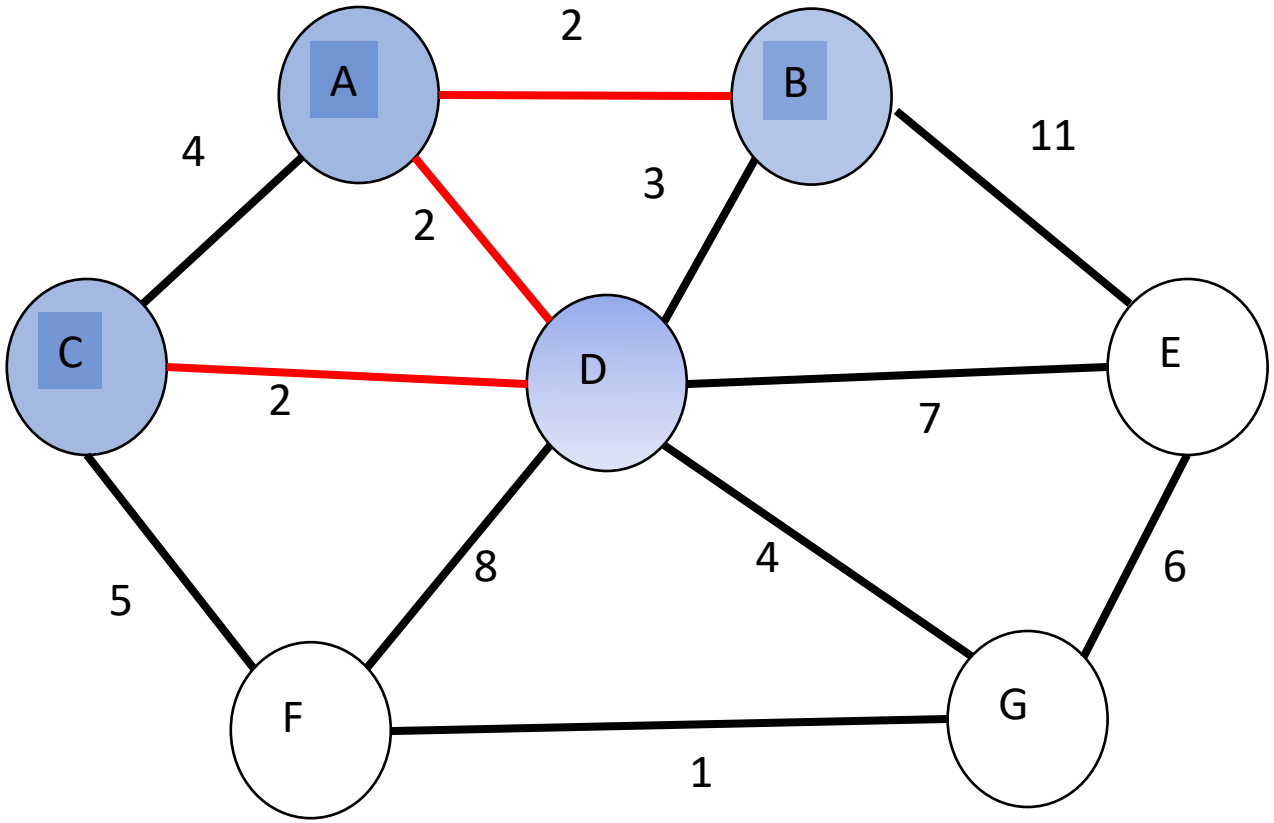
¿El arista de menor costo del árbol al vértice que no está en el árbol? 2 de D a A (o C)

Algoritmo de Prim



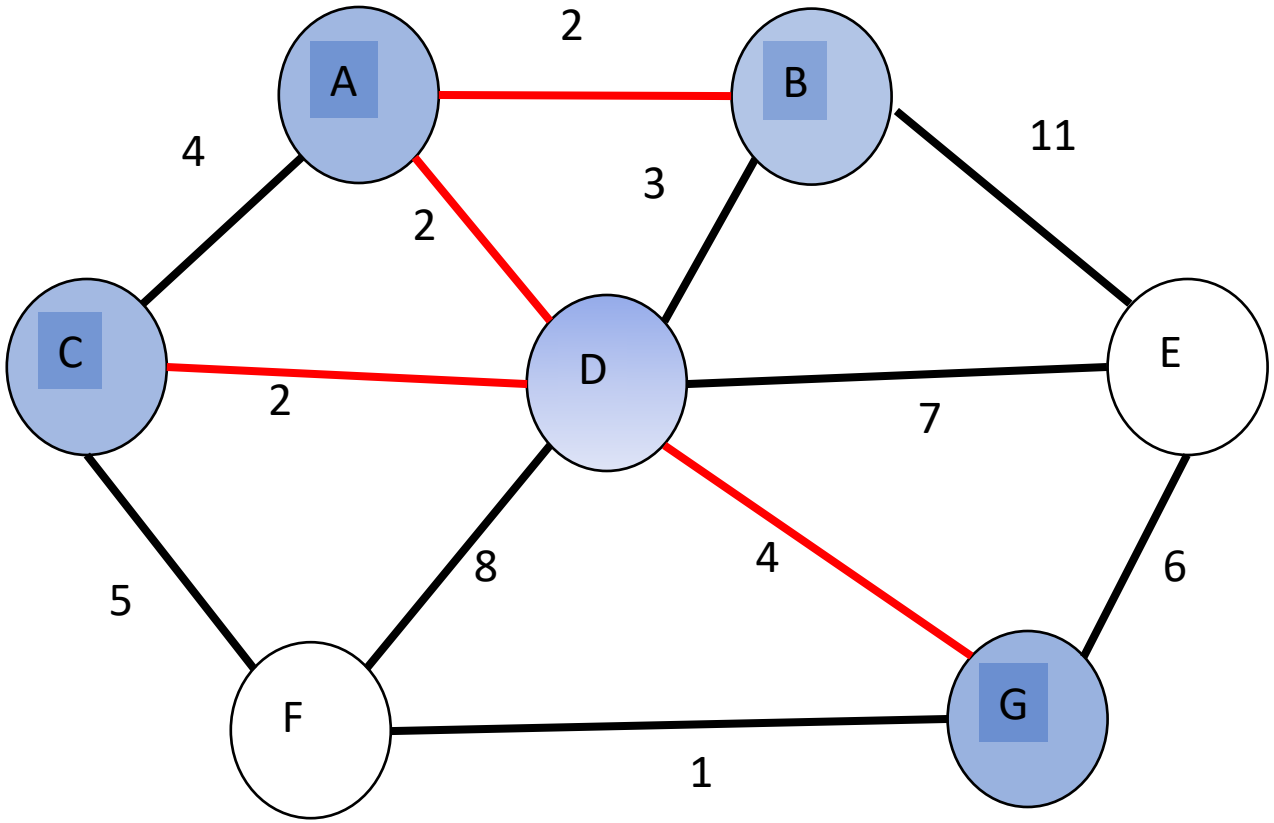
¿El arista de menor costo del árbol a un vértice que no está en el árbol? 2 de D a C (o de A a B)

Algoritmo de Prim



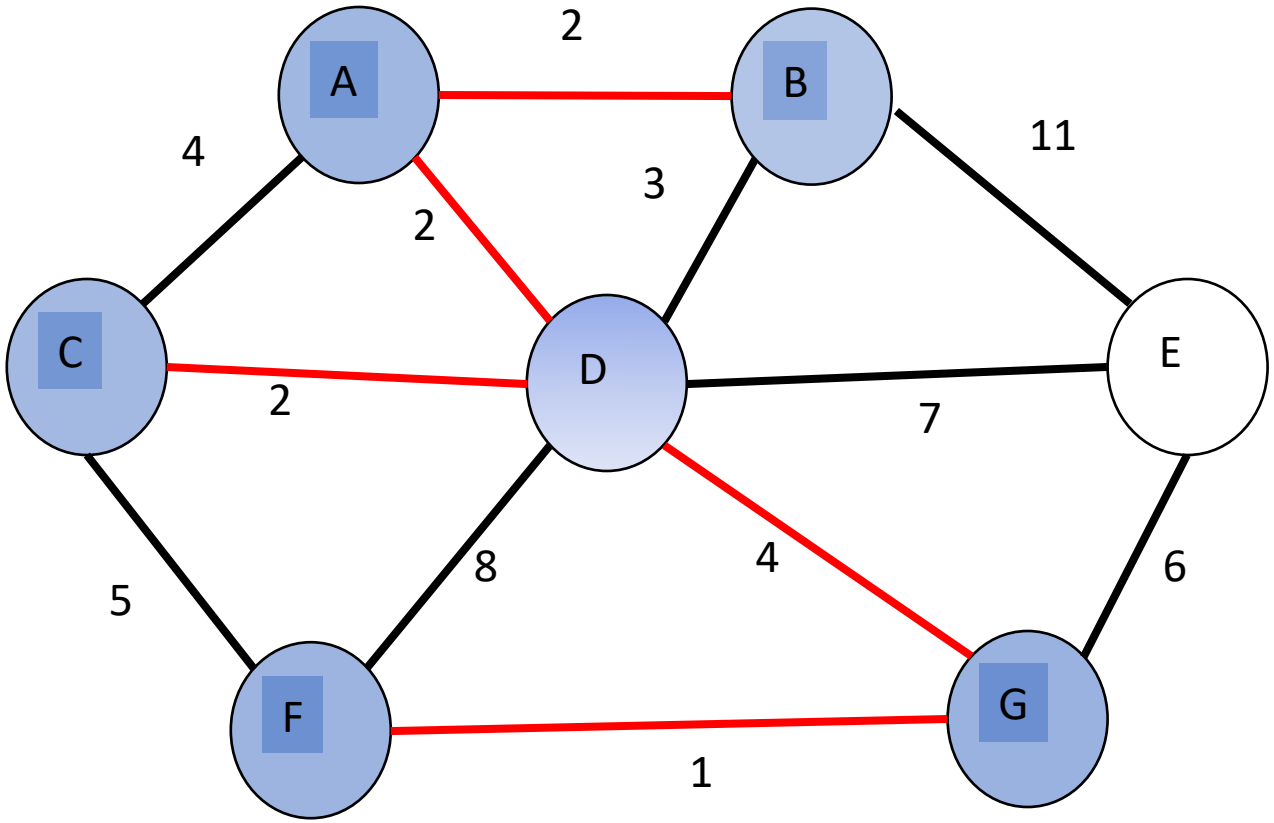
¿El arista de menor costo de árbol a un vértice que no está en el árbol? 2 de A a B

Algoritmo de Prim



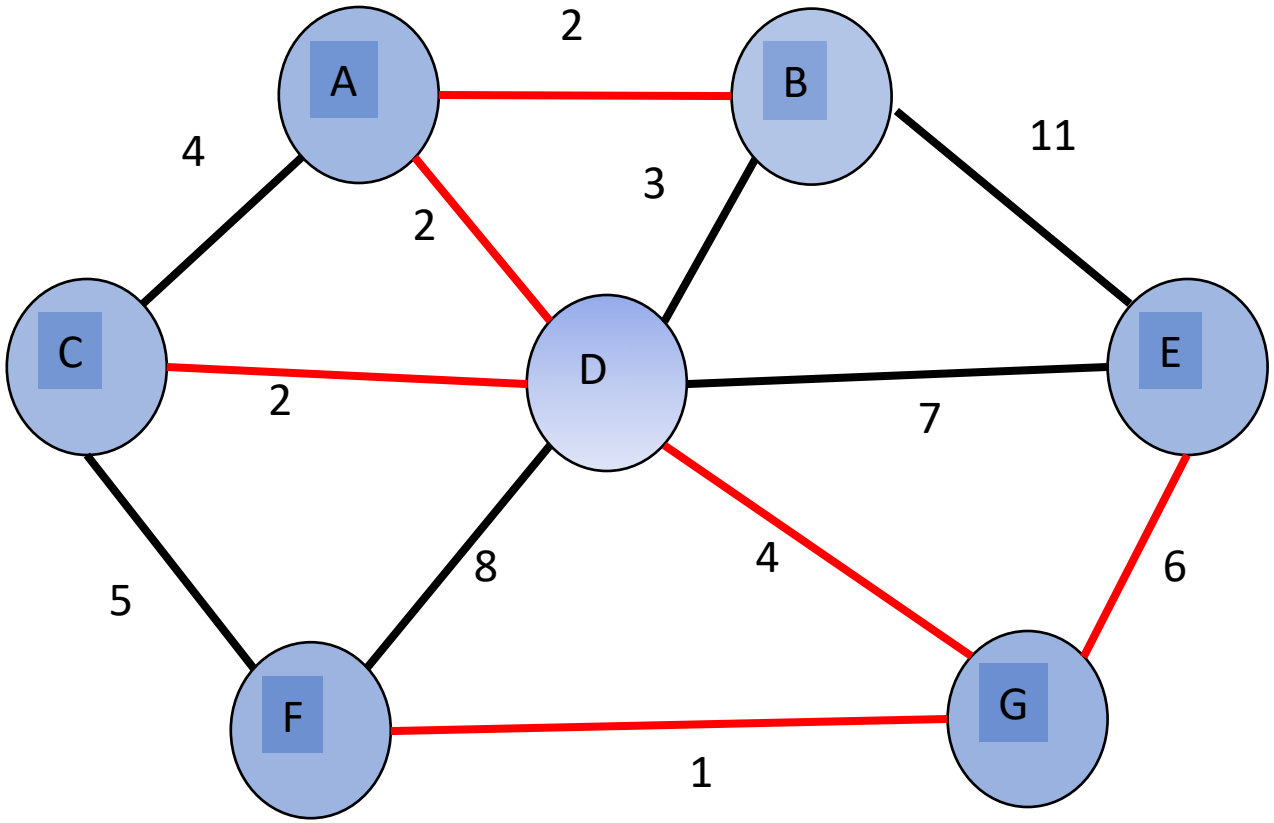
¿arista de menor costo del árbol a vértice que no está en el árbol? 5 de D a G

Algoritmo de Prim



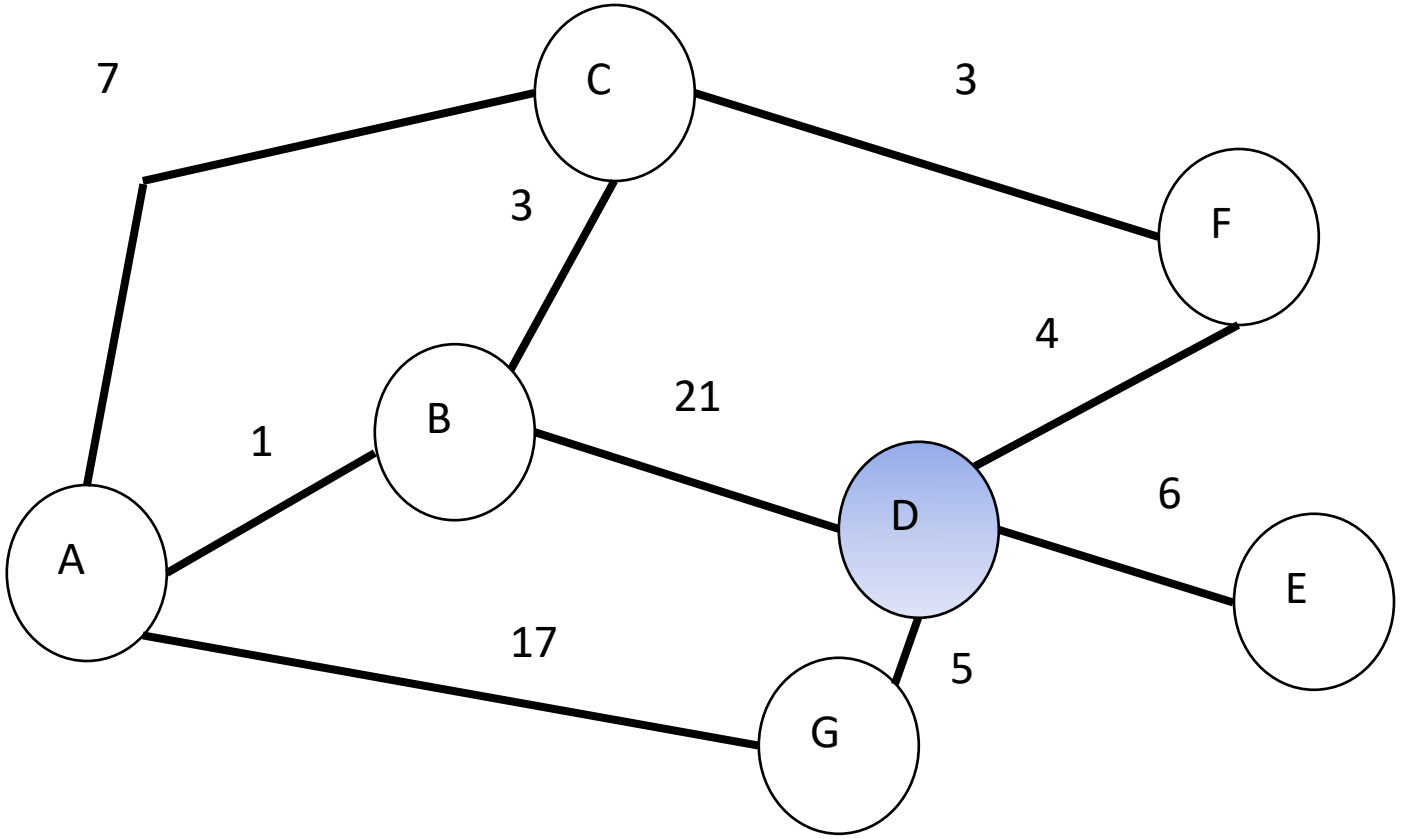
¿El arista de menor costo del árbol a un vértice que no está en el árbol? 1 de G a F

Algoritmo de Prim



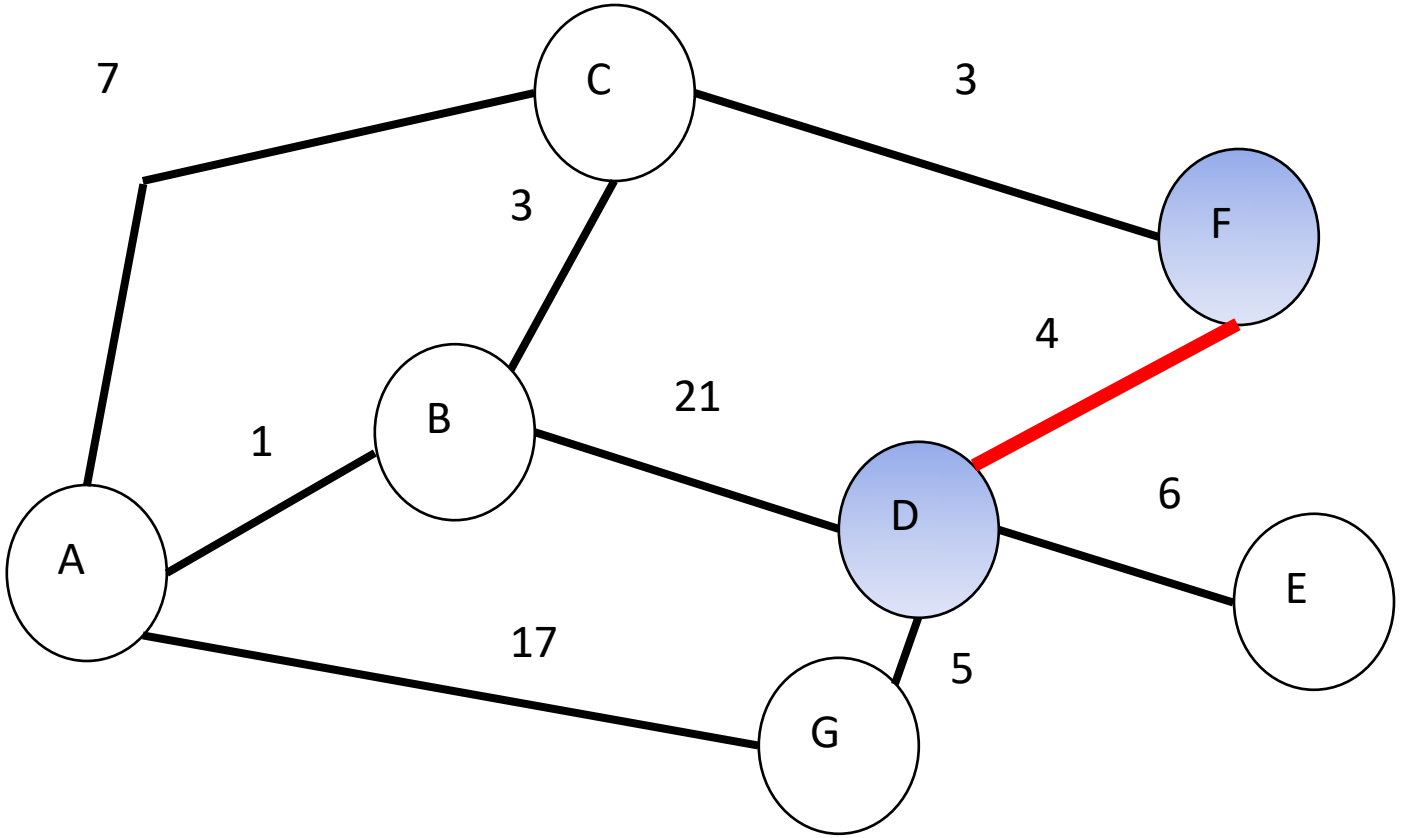
¿El arista de menor costo del árbol a un vértice que no está en el árbol? 6 de G a E

Algoritmo de Prim



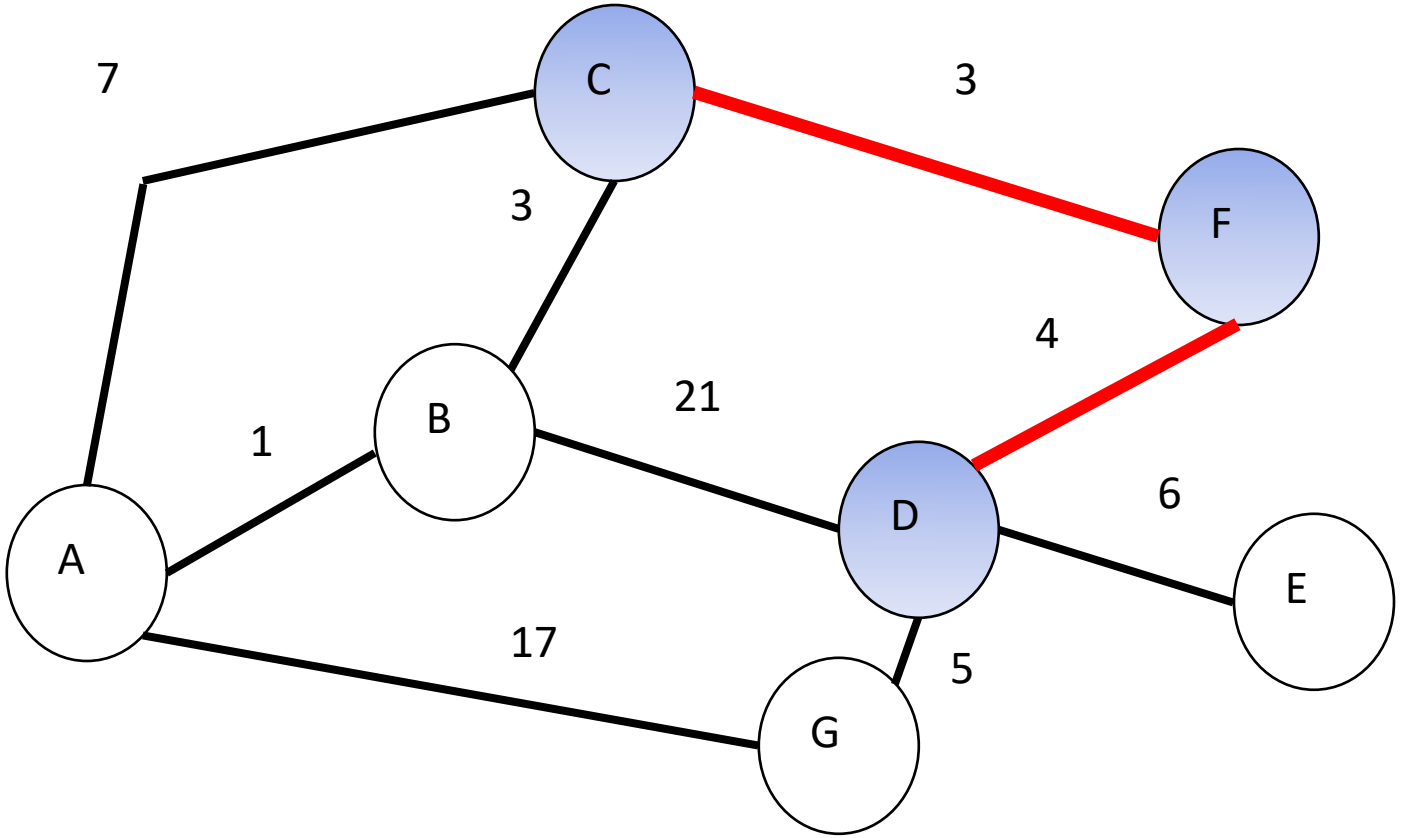
Elija D como raíz

Algoritmo de Prim



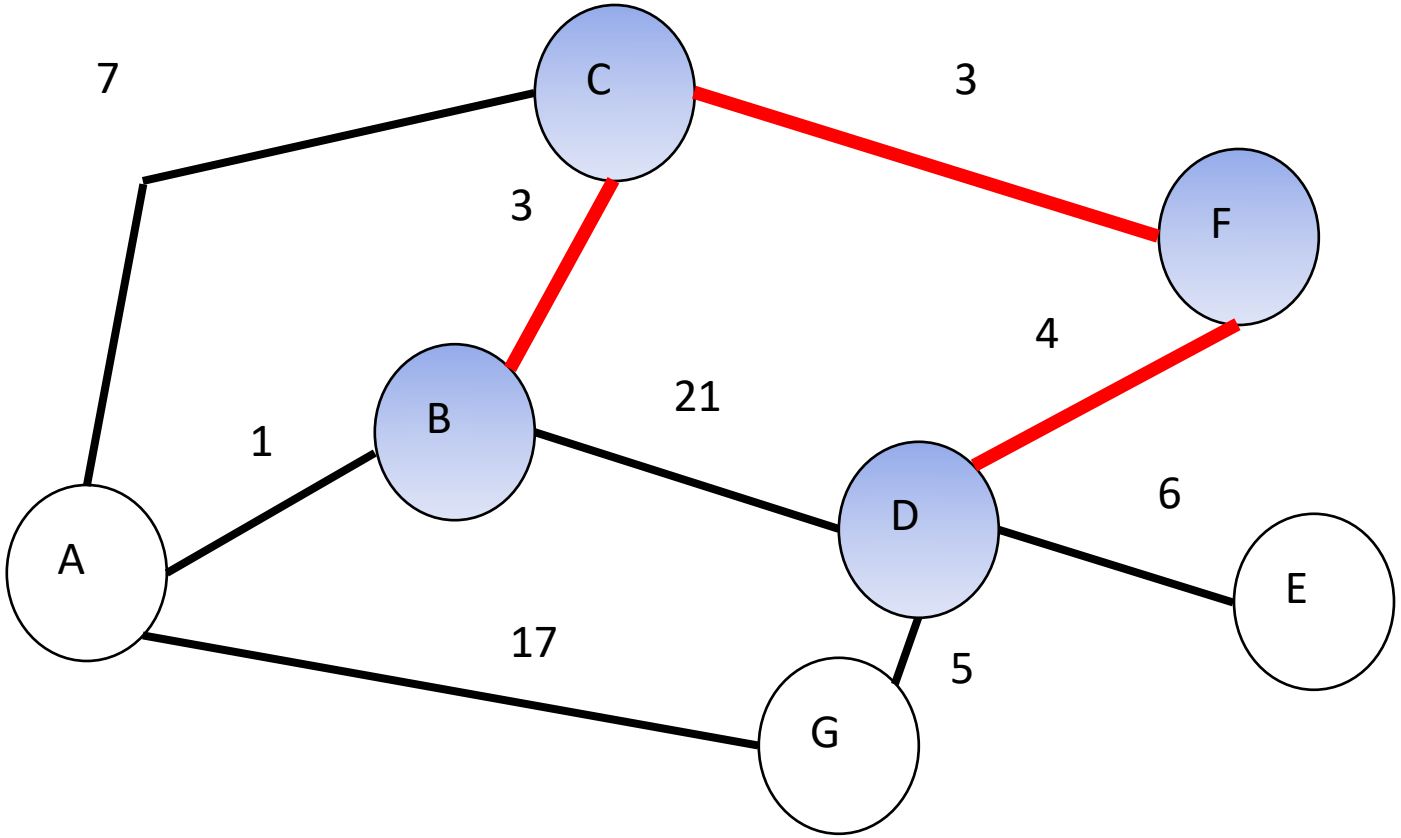
¿El arista de menor costo del árbol a un vértice que no está en el árbol? 4 de D a F

Algoritmo de Prim



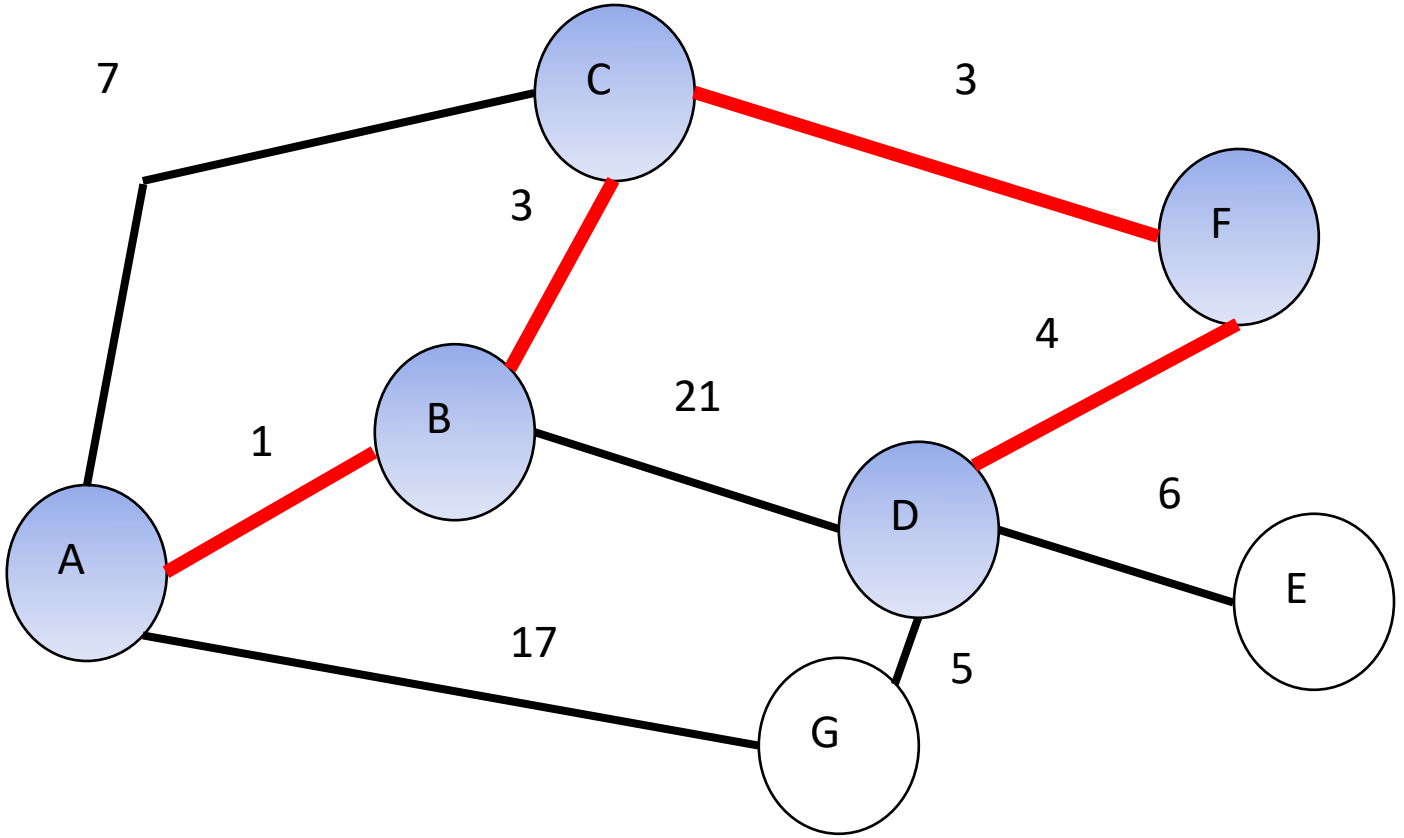
¿El arista de menor costo del árbol a un vértice que no está en el árbol? 3 de F a C

Algoritmo de Prim



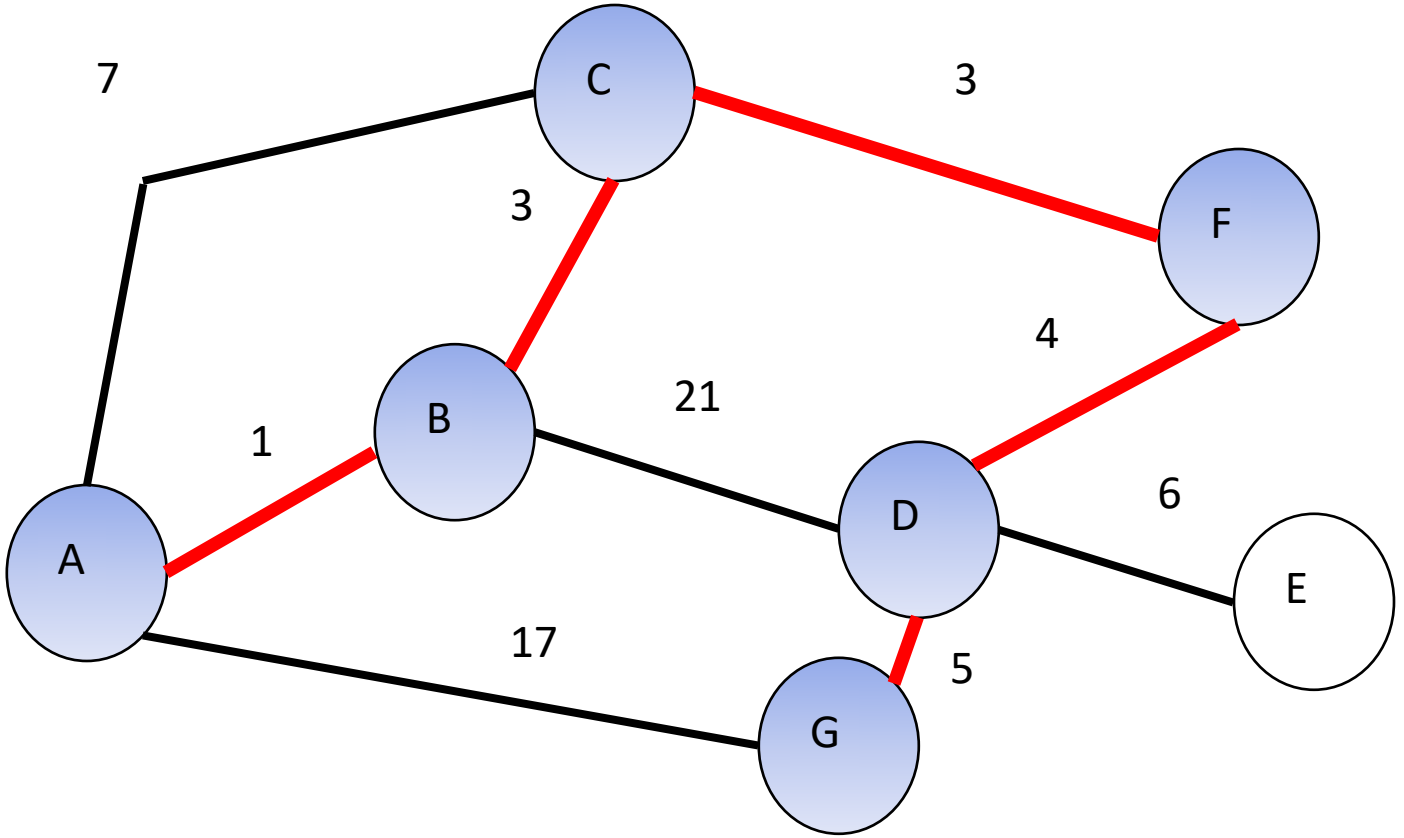
¿El arista de menor costo del árbol a un vértice que no está en el árbol? 3 de C a B

Algoritmo de Prim



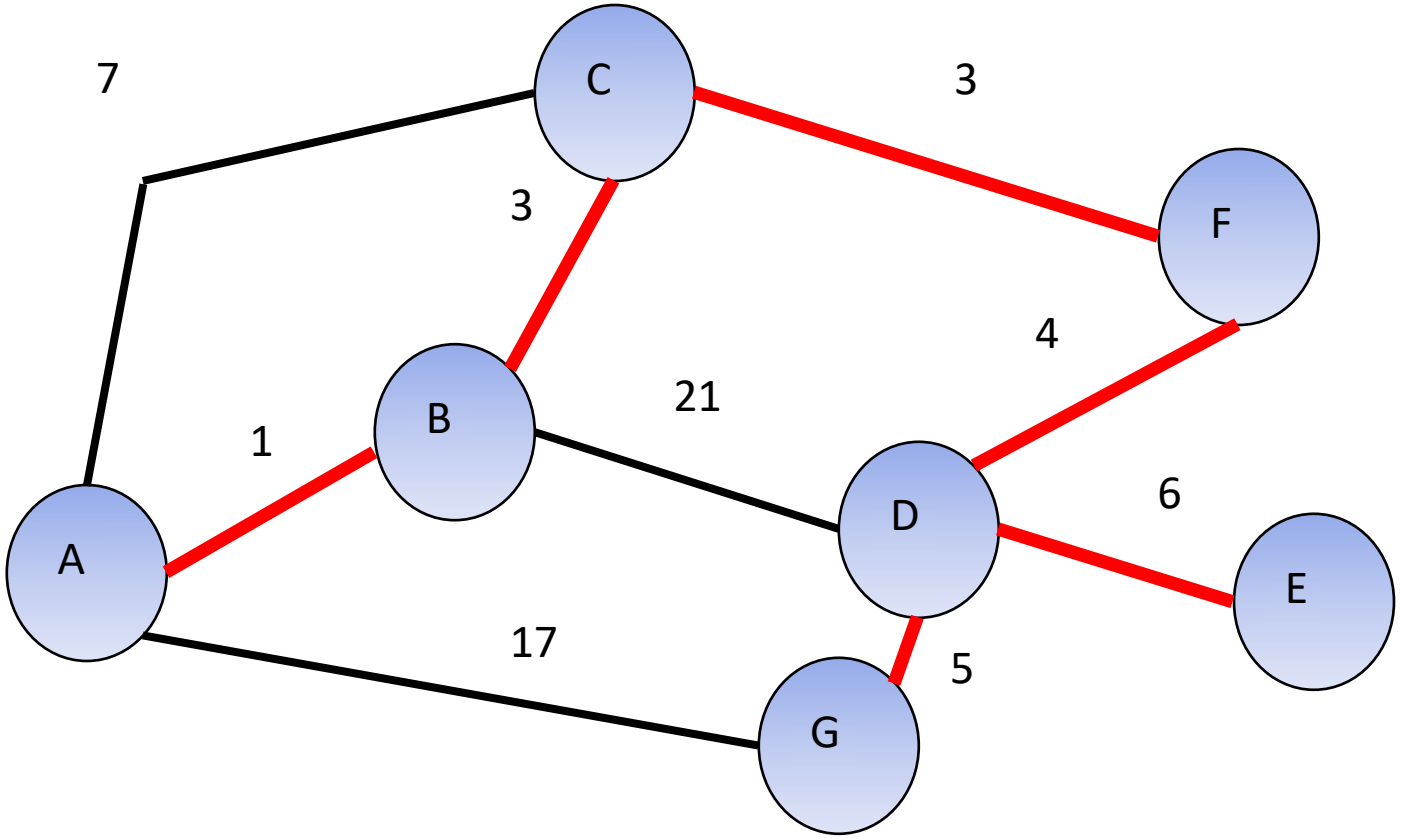
¿El arista de menor costo del árbol a un vértice que no está en el árbol? 1 de B a A

Algoritmo de Prim



¿El arista de menor costo del árbol a un vértice que no está en el árbol? 5 de D a G

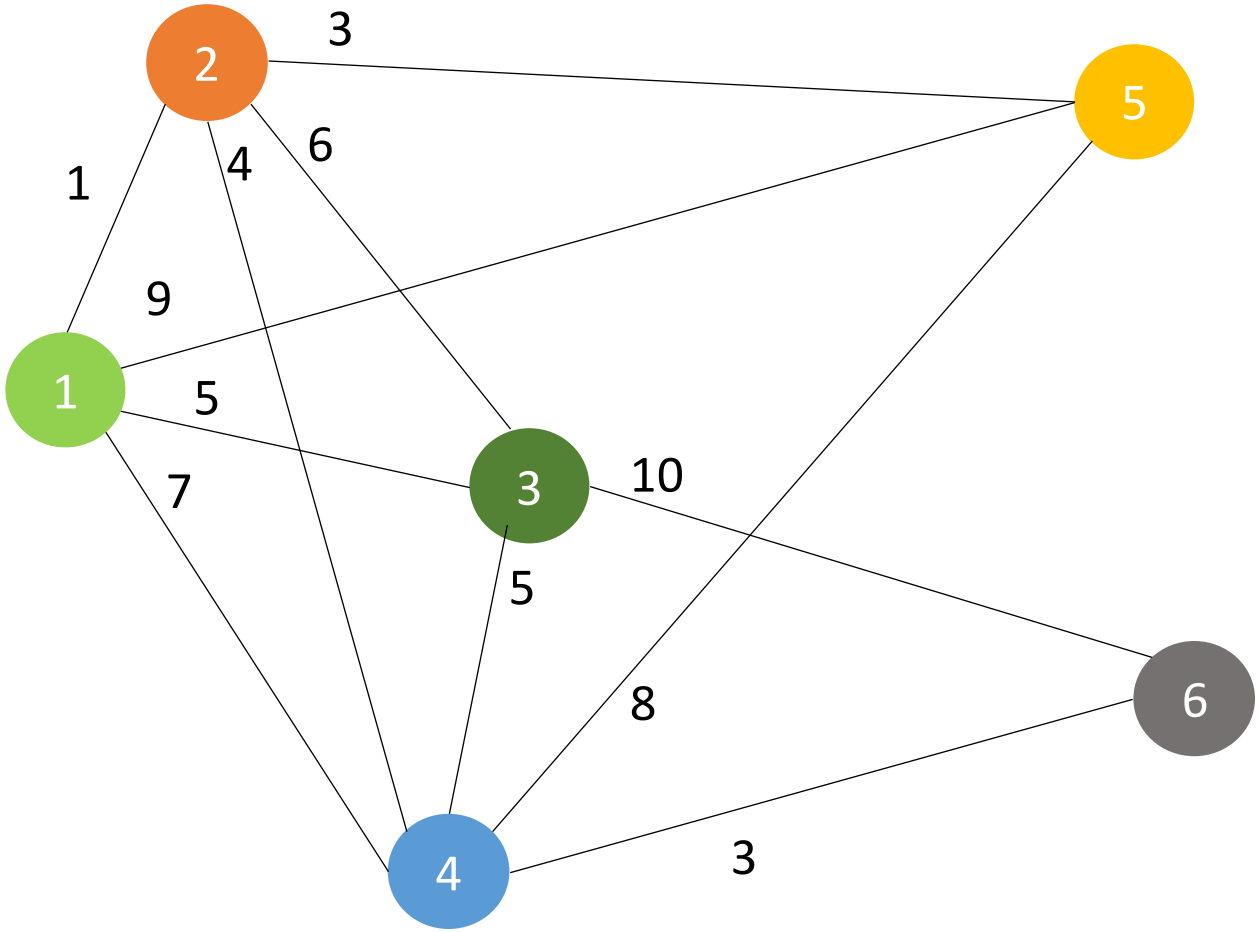
Algoritmo de Prim



¿El arista de menor costo del árbol a un vértice que no está en el árbol? 6 de D a E

Ejercicio - Problema del árbol de expansión mínima

Midwest TV Cable Company va a proporcionar servicio de cable a seis desarrollos habitacionales. La figura ilustra las posibles conexiones de TV a las cinco áreas, con los Km de cable anexadas a cada arco. El objetivo es determinar la red de cables más económica.



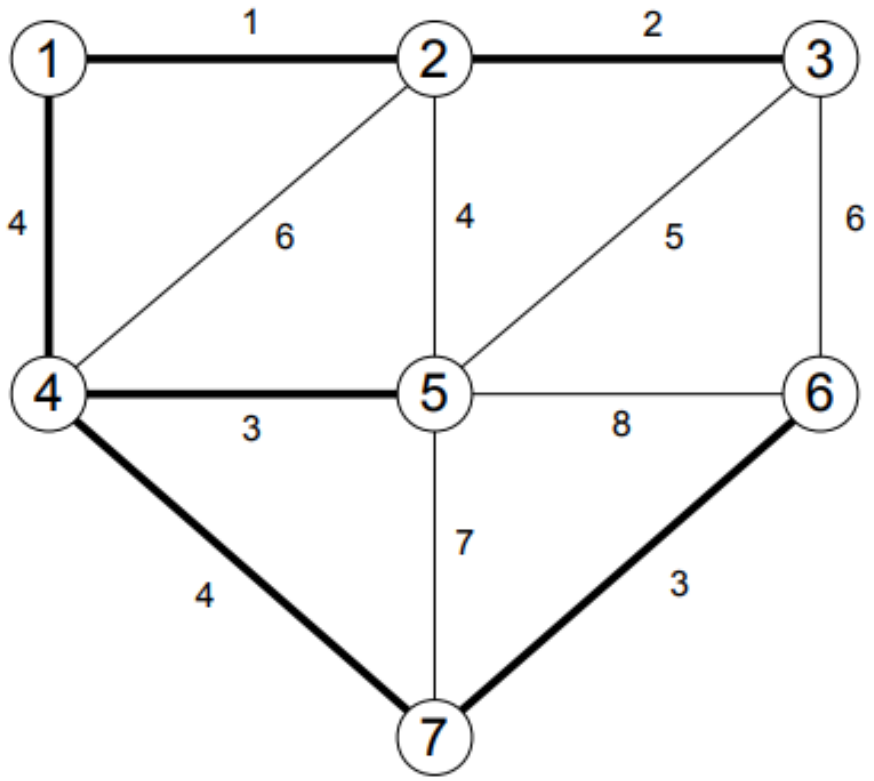
Algoritmo de Prim

Esquema:

1. Empezar en un vértice cualquiera **v**. El árbol consta inicialmente sólo del nodo **v**.
2. Del resto de vértices, buscar el que esté más próximo a **v** (es decir, con la arista (**v**, **w**) de coste mínimo). Añadir **w** y la arista (**v**, **w**) al árbol.
3. Buscar el vértice más próximo a cualquiera de estos dos. Añadir ese vértice y la arista al árbol de expansión.
4. Repetir sucesivamente hasta añadir los **n** vértices.

Algoritmo de Prim

El algoritmo de Prim construye el árbol de recubrimiento de coste mínimo añadiendo un nuevo vértice a cada paso, comenzando por un vértice arbitrario.



Paso	Arista elegida	B
Inicialización	–	{1}
1	{1, 2}	{1, 2}
2	{2, 3}	{1, 2, 3}
3	{1, 4}	{1, 2, 3, 4}
4	{4, 5}	{1, 2, 3, 4, 5}
5	{4, 7}	{1, 2, 3, 4, 5, 7}
6	{7, 6}	{1, 2, 3, 4, 5, 6, 7}

Algoritmo de Prim

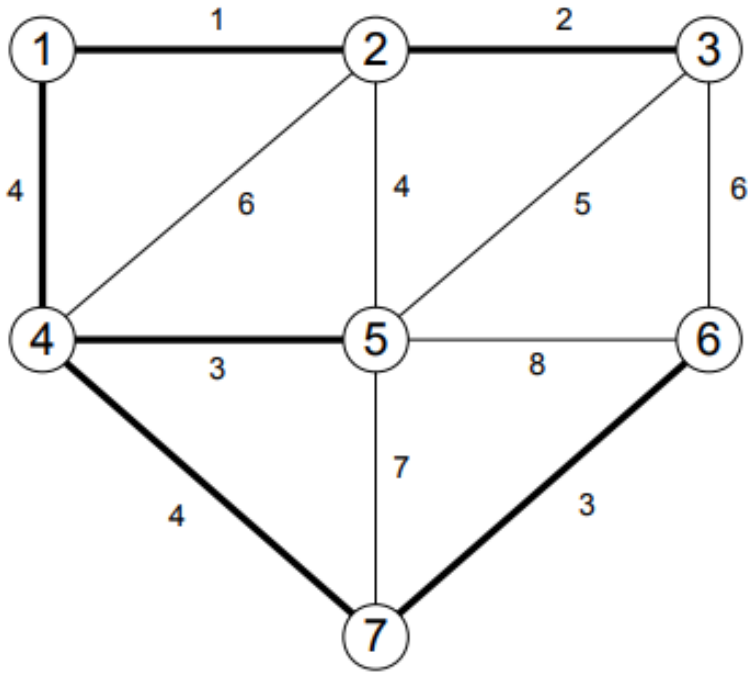
Implementación en lenguaje Python

```
# n: número de vértices del grafo
# c: diccionario de costes, indexado por aristas: tuplas (i,j)
# Se supone definida la función argmin()

def Prim(n,c):
    T = []                    # inicialización del árbol de recubrimiento
    B = [0]                   # B se inicializa con un vértice cualquiera
    while len(B) < n:
        e = argmin(B,n,c) # e: arista de menor coste t.q. e[0]∈B y e[1]∉B
        B.append(e[1])    # el vértice e[1] entra en B
        T.append(e)       # la arista e entra en la solución
    return T
```

Algoritmo de Kruskal

Para construir el árbol de recubrimiento de coste mínimo de un grafo, el algoritmo de Kruskal recorre las aristas en orden creciente de costes.



Paso	Arista considerada	Arista aceptada	Componentes conexas
Inicialización	–	–	{1}{2}{3}{4}{5}{6}{7}
1	{1, 2}	Sí	{1, 2}{3}{4}{5}{6}{7}
2	{2, 3}	Sí	{1, 2, 3}{4}{5}{6}{7}
3	{4, 5}	Sí	{1, 2, 3}{4, 5}{6}{7}
4	{6, 7}	Sí	{1, 2, 3}{4, 5}{6, 7}
5	{1, 4}	Sí	{1, 2, 3, 4, 5}{6, 7}
6	{2, 5}	No	{1, 2, 3, 4, 5}{6, 7}
7	{4, 7}	Sí	{1, 2, 3, 4, 5, 6, 7}

Algoritmo de Kruskal

Implementación en lenguaje Python

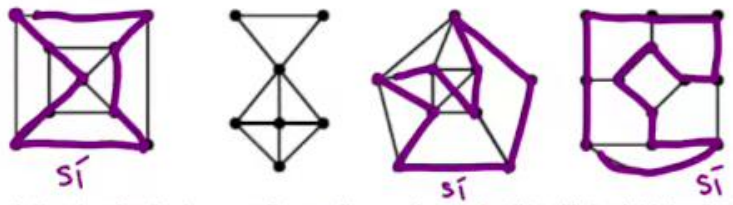
```
# n: número de vértices del grafo
# c: diccionario de costes, indexado por aristas: tuplas (i,j)
# Se suponen definidas las funciones argmin() y componente()

def Kruskal(n,c):
    T = []                          # inicialización del árbol de recubrimiento
    B = [[i] for i in range(n)]     # inicialmente: n componentes conectadas,
                                    # cada una con un vértice
    while len(T) < n-1:
        e = argmin(c)               # se extrae la arista de coste mínimo:
        c.pop(e)
        i = componente(B,e[0])      # componente donde está el vértice e[0]
        j = componente(B,e[1])      # componente donde está el vértice e[1]
        if i != j:
            B[i].extend(B[j])       # se fusionan las componentes
            B.pop(j)
            T.append(e)             # la arista e entra en la solución
    return T
```

Grafos eulerianos y hamiltonianos

https://www.youtube.com/watch?v=dtdnM0OaGEI

7. **[2 puntos]** Consideremos los siguientes grafos:



Indicar si son hamiltonianos o no. Para aquellos que sí lo sean, describir o dibujar el ciclo hamiltoniano correspondiente.

G hamiltoniano $\Leftrightarrow \exists$ ciclo hamiltoniano.

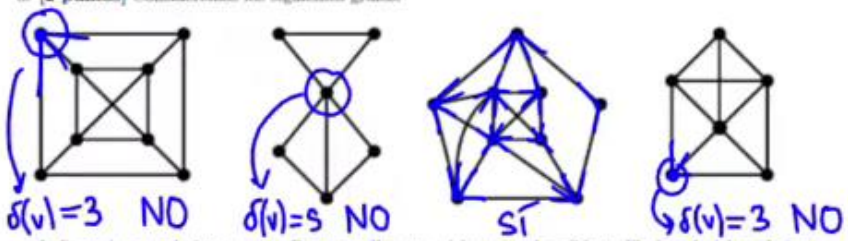
- Pasamos de un vértice al mismo sin repetir aristas
- Pasamos por todos vértices una única vez.

- DIRAC: Si $\delta(v) \geq \frac{|V|}{2}$ $\forall v \in V \Rightarrow$ G hamiltoniano.

- Si G hamiltoniano $\Rightarrow \omega(G-W) \leq |W| \quad \forall W \subset V(G)$

G NO hamiltoniano $\Leftarrow$ Si $\exists W \subset V(G) \mid \omega(G-W) > |W|$

6. **[2 puntos]** Consideremos los siguientes grafos:



Indicar si son eulerianos o no. Para aquellos que sí lo sean, describir o dibujar el ciclo euleriano correspondiente.

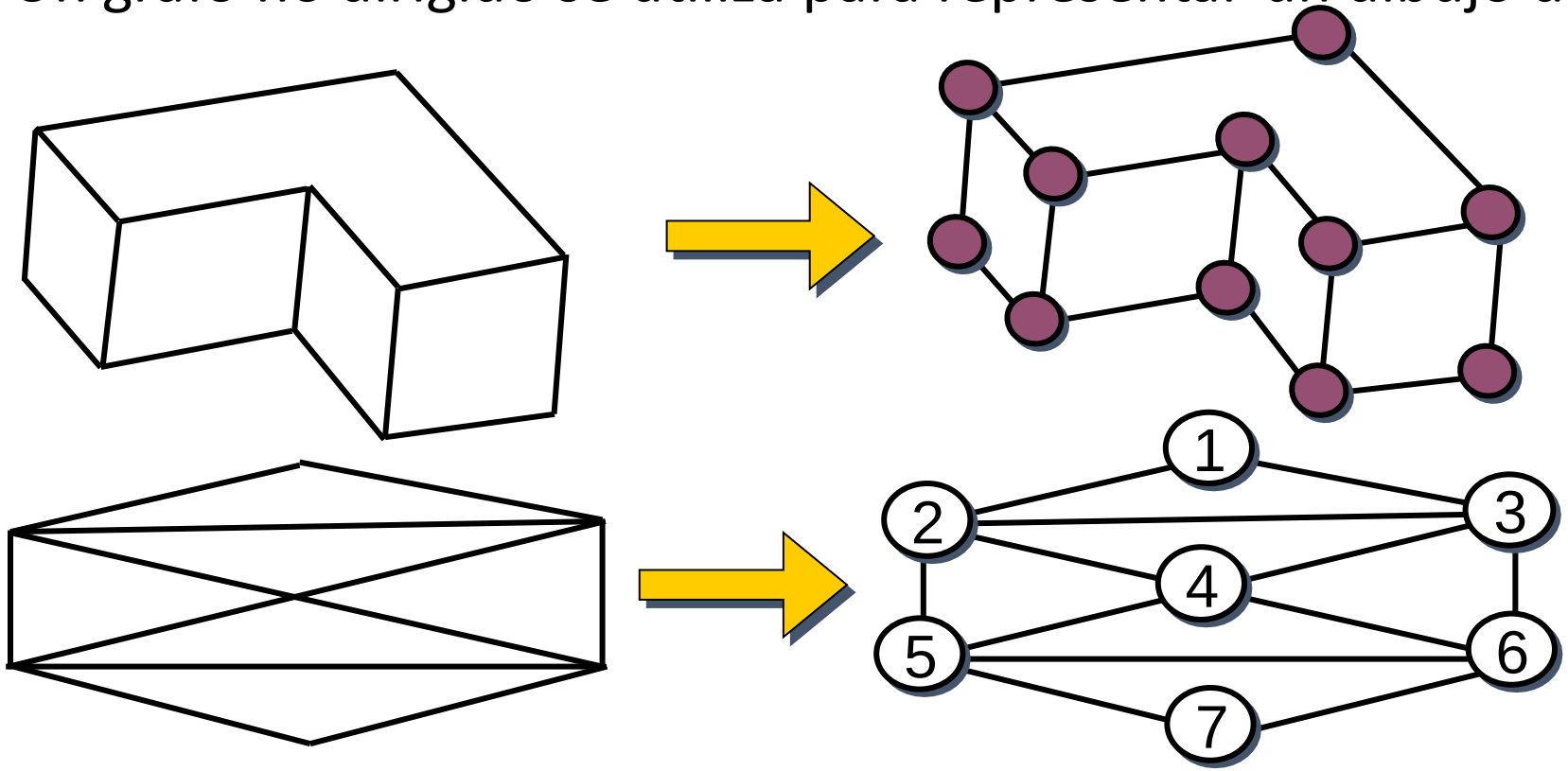
G euleriano $\Leftrightarrow \exists$ ciclo euleriano

Pasa por todas las aristas

- EULER: G euleriano $\Leftrightarrow \delta(v)$ par $\forall v \in V(G)$.

Caminos y circuitos de Euler.

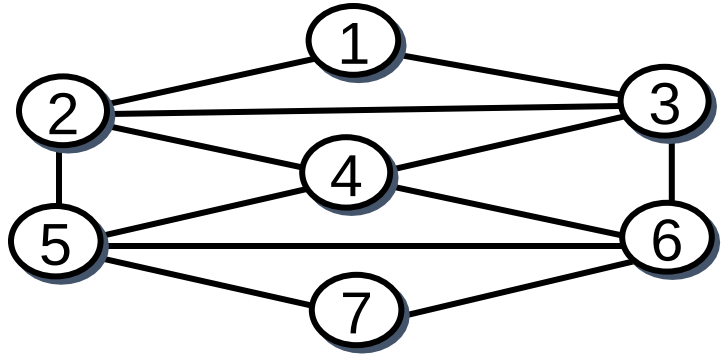
- **Aplicación:** Un grafo no dirigido se utiliza para representar un dibujo de líneas.



- **Pregunta:** ¿Es posible dibujar estas figuras con un bolígrafo, pintando cada línea una sola vez, sin levantar el bolígrafo y acabando donde se empezó?

Caminos y circuitos de Euler.

- El problema se **transforma** en un problema de grafos.
- **Circuito de Euler:** Es un ciclo (no necesariamente simple) que visita todas las aristas exactamente una vez.
- Si puede empezar y acabar en nodos distintos: **Camino de Euler**.



- **Condiciones necesarias y suficientes para que exista un circuito de Euler:**
  - El grafo debe ser conexo.
  - Todos los nodos deben tener grado par, ya que el camino entra y sale de los nodos.
- ¿Y para los caminos de Euler?

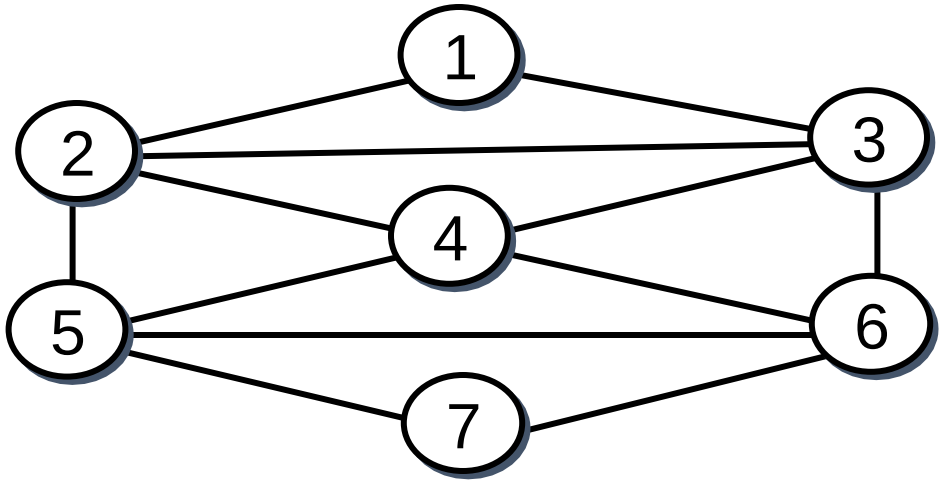
Caminos y circuitos de Euler.

- Si existe un circuito de Euler, ¿cómo calcularlo?
- **Algoritmo para encontrar un circuito de Euler en un grafo G, partiendo de un nodo v.**
  1. Buscar un ciclo cualquiera en **G** empezando por **v**.
  2. Si quedan aristas por visitar, seleccionar el primer nodo, **w**, del ciclo que tenga una arista sin visitar. Buscar otro ciclo partiendo de **w** que pase por aristas no visitadas.
  3. Unir el ciclo del paso 1 con el obtenido en el paso 2.
  4. Repetir sucesivamente los pasos 2 y 3 hasta que no queden aristas por visitar.

- ¿Cómo encontrar un ciclo en el grafo, que pase por aristas no visitadas (pasos 1 y 2)?

Caminos y circuitos de Euler.

- **Ejemplo:** Encontrar un circuito de Euler para el siguiente grafo.

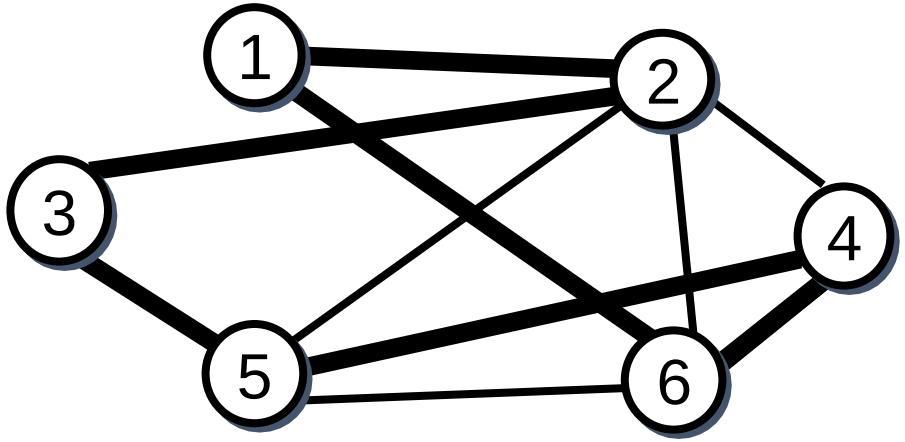


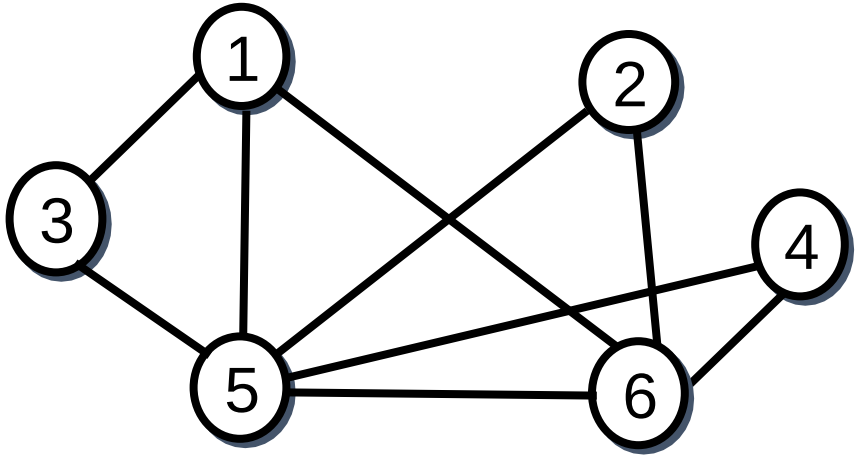
- ¿Cómo modificar el algoritmo para el caso del camino de Euler?

Otros problemas con grafos.

Problema del ciclo hamiltoniano

- **Definición:** Dado un grafo no dirigido **G**, un **ciclo de Hamilton (o hamiltoniano)** es un ciclo simple que visita todos los vértices. Es decir, pasa por todos los vértices exactamente una vez.
- **Problema del ciclo hamiltoniano.**
  Determinar si un grafo no dirigido dado tiene un ciclo hamiltoniano o no.





Otros problemas con grafos.

- Aunque el problema es muy parecido al del circuito de Euler, no se conoce ningún algoritmo eficiente para resolverlo, en tiempo polinomial.

- El problema del ciclo hamiltoniano pertenece a un conjunto de problemas de difícil solución, llamados **problemas NP-completos**.

- Las soluciones conocidas requieren básicamente “evaluar todas las posibilidades”, dando lugar a órdenes de complejidad exponenciales o factoriales.

- Otra alternativa es usar métodos **heurísticos**: soluciones aproximadas que pueden funcionar en algunos casos y en otros no.

Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América