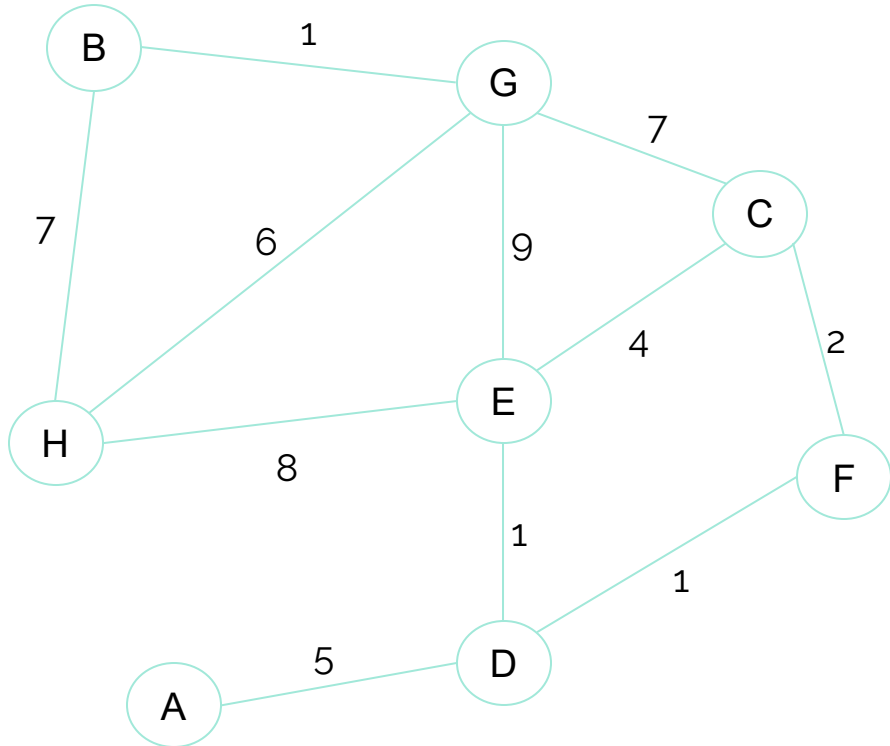


Dijkstra - Bellman Ford - MST~Kruskal

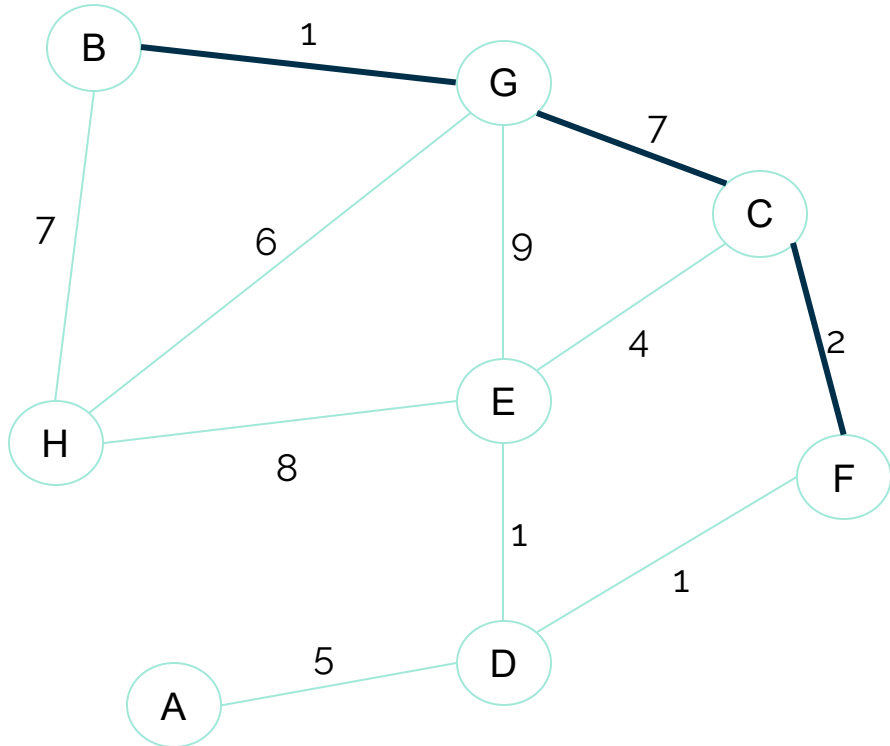
Joaquín Peralta - Elías Ayaach - Alex Infanta - Amelia Gonzalez - Paula Grune

El camino más corto



Definimos el camino más corto entre dos nodos como la menor suma de las aristas que los conectan

El camino más corto



Por ejemplo, el camino más corto entre B y F

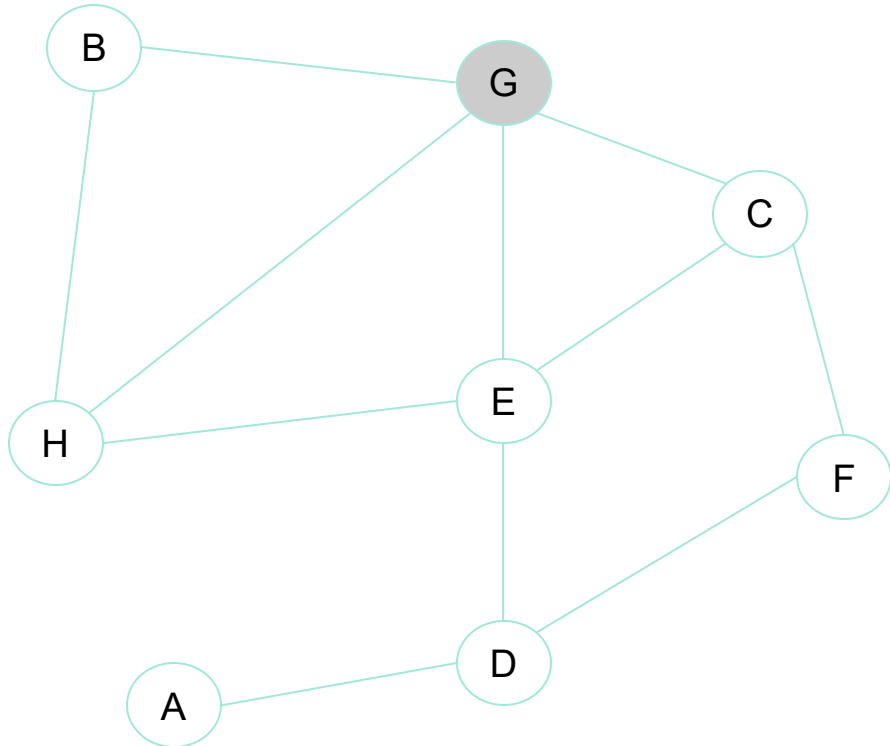
¿Cómo podemos encontrar este camino más corto?

BFS

Antes de ver cómo encontramos el camino más corto, debemos entender como funciona BFS:

- Es un algoritmo de **búsqueda en amplitud**
- Partiendo de un nodo del grafo, recorremos primero los nodos que están a una arista de distancia, luego a dos aristas de distancia, y así hasta llegar al último nodo
- Marcamos los nodos que ya visitamos con el fin de no revisar infinitamente
- Para esto **usamos una cola FIFO**, cada vez que encontramos un nodo nuevo lo agregamos al final de la cola y para revisar nuevos sacamos el primer nodo

BFS

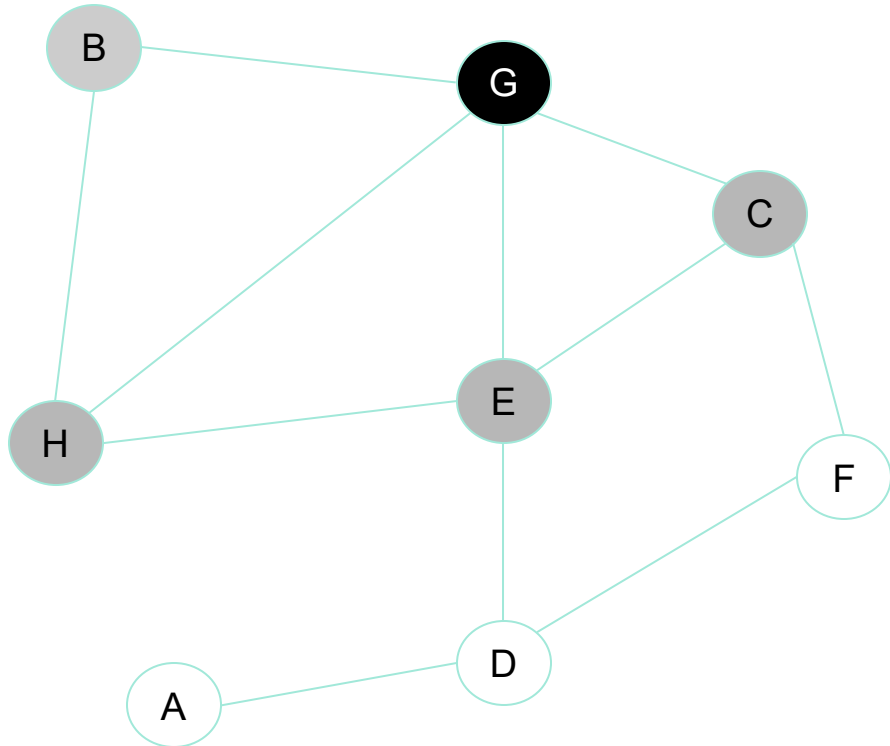


Partimos seleccionando un nodo (por ejemplo G)

Cada vez que agregamos un nodo a la cola lo marcamos de gris, cada vez que sacamos un nodo lo marcamos de negro

$Q = \{G\}$

BFS

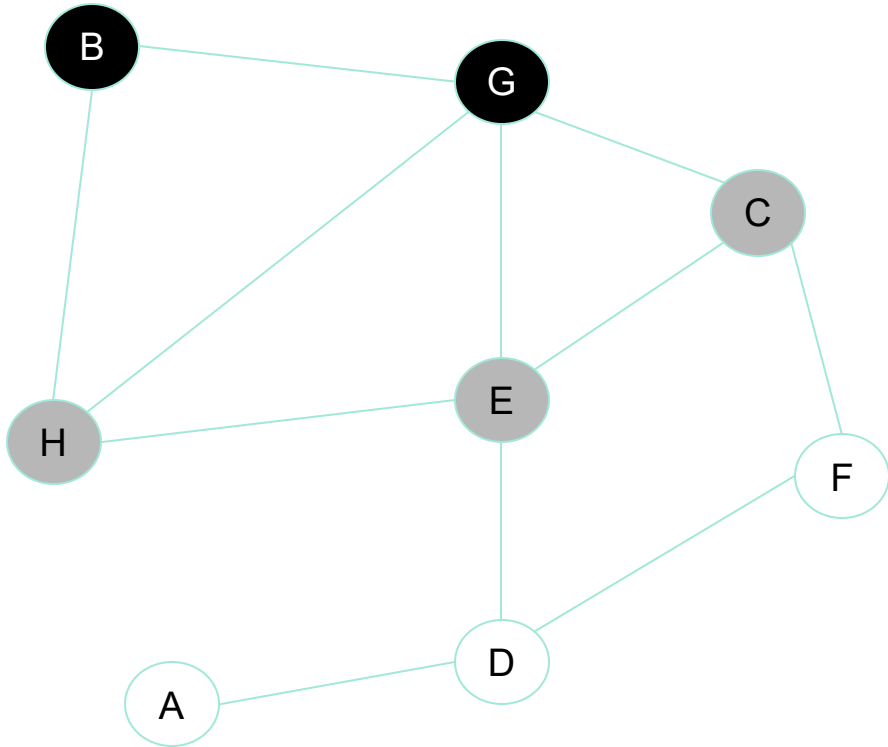


Si un nodo es gris o negro, no lo volvemos a agregar

Cada vez que agregamos un nodo a la cola lo marcamos de gris, cada vez que sacamos un nodo lo marcamos de negro

$Q = \{B, H, E, C\}$

BFS

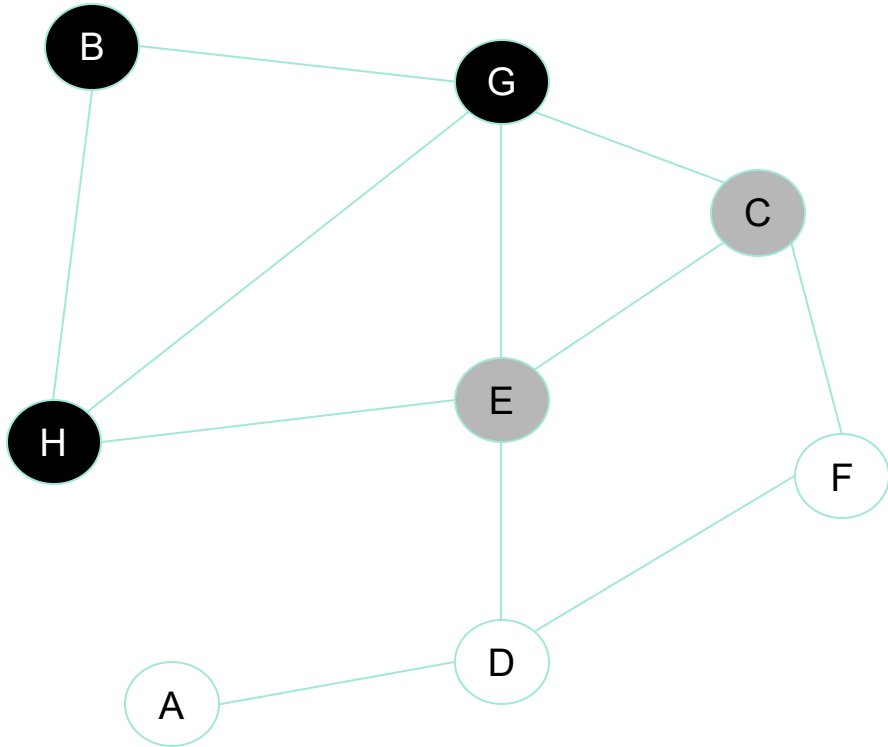


Si un nodo es gris o negro, no lo volvemos a agregar

Cada vez que agregamos un nodo a la cola lo marcamos de gris, cada vez que sacamos un nodo lo marcamos de negro

$Q = \{H, E, C\}$

BFS

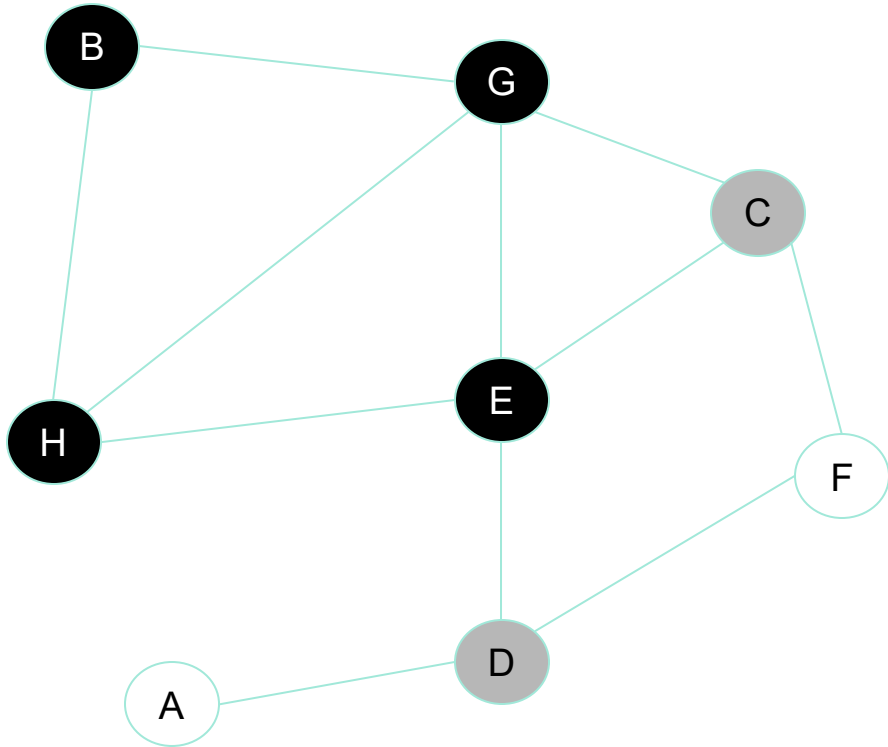


Si un nodo es gris o negro, no lo volvemos a agregar

Cada vez que agregamos un nodo a la cola lo marcamos de gris, cada vez que sacamos un nodo lo marcamos de negro

$Q = \{E, C\}$

BFS

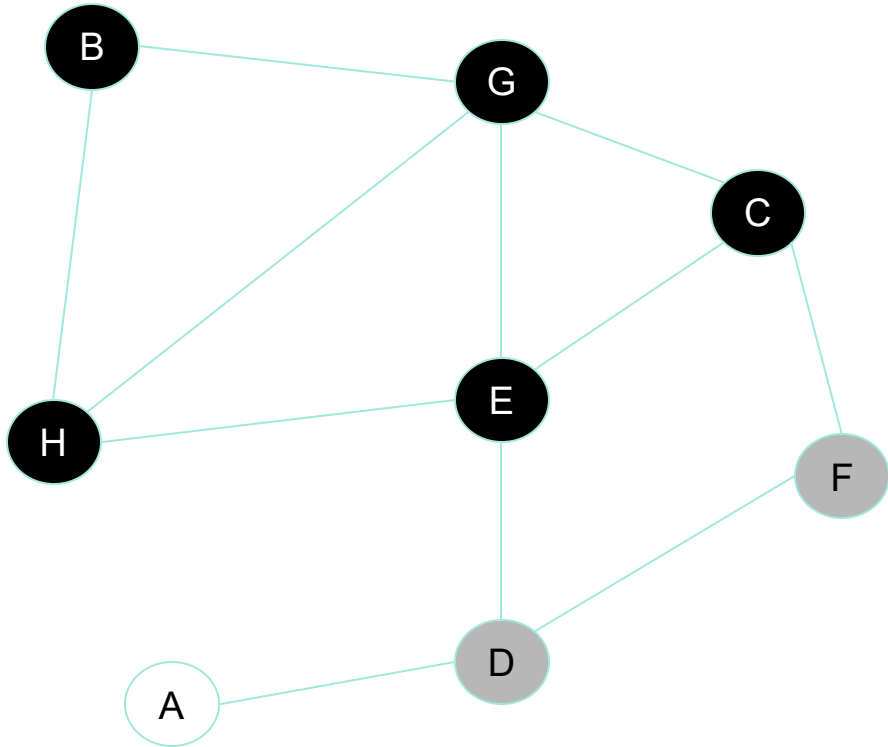


Si un nodo es gris o negro, no lo volvemos a agregar

Cada vez que agregamos un nodo a la cola lo marcamos de gris, cada vez que sacamos un nodo lo marcamos de negro

$Q = \{C, D\}$

BFS

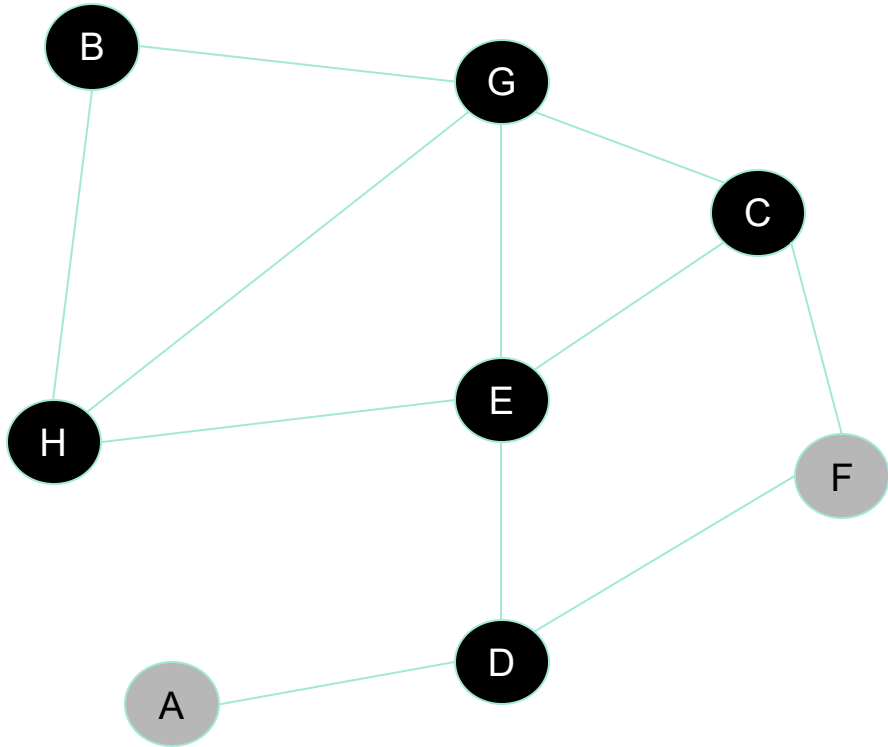


Si un nodo es gris o negro, no lo volvemos a agregar

Cada vez que agregamos un nodo a la cola lo marcamos de gris, cada vez que sacamos un nodo lo marcamos de negro

$Q = \{D, F\}$

BFS

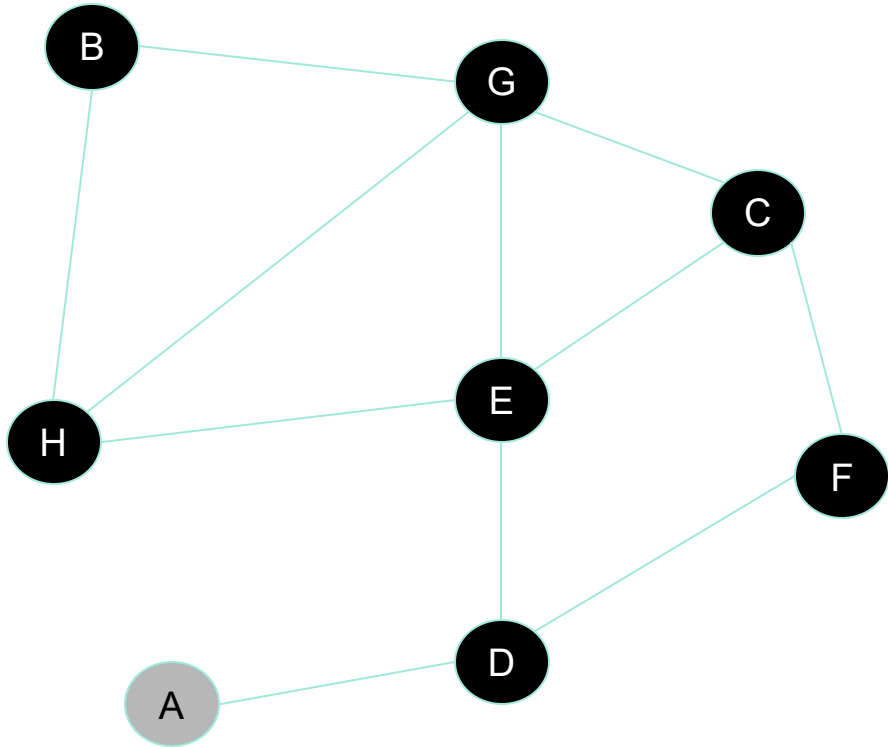


Si un nodo es gris o negro, no lo volvemos a agregar

Cada vez que agregamos un nodo a la cola lo marcamos de gris, cada vez que sacamos un nodo lo marcamos de negro

$Q = \{F, A\}$

BFS

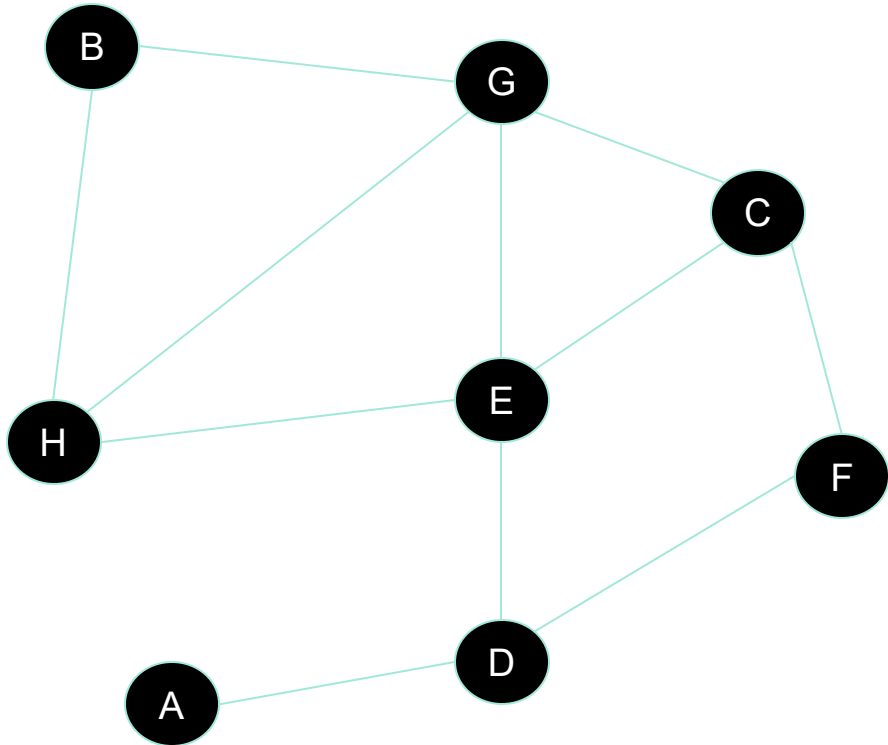


Si un nodo es gris o negro, no lo volvemos a agregar

Cada vez que agregamos un nodo a la cola lo marcamos de gris, cada vez que sacamos un nodo lo marcamos de negro

$Q = \{A\}$

BFS



Si un nodo es gris o negro, no lo volvemos a agregar

Cada vez que agregamos un nodo a la cola lo marcamos de gris, cada vez que sacamos un nodo lo marcamos de negro

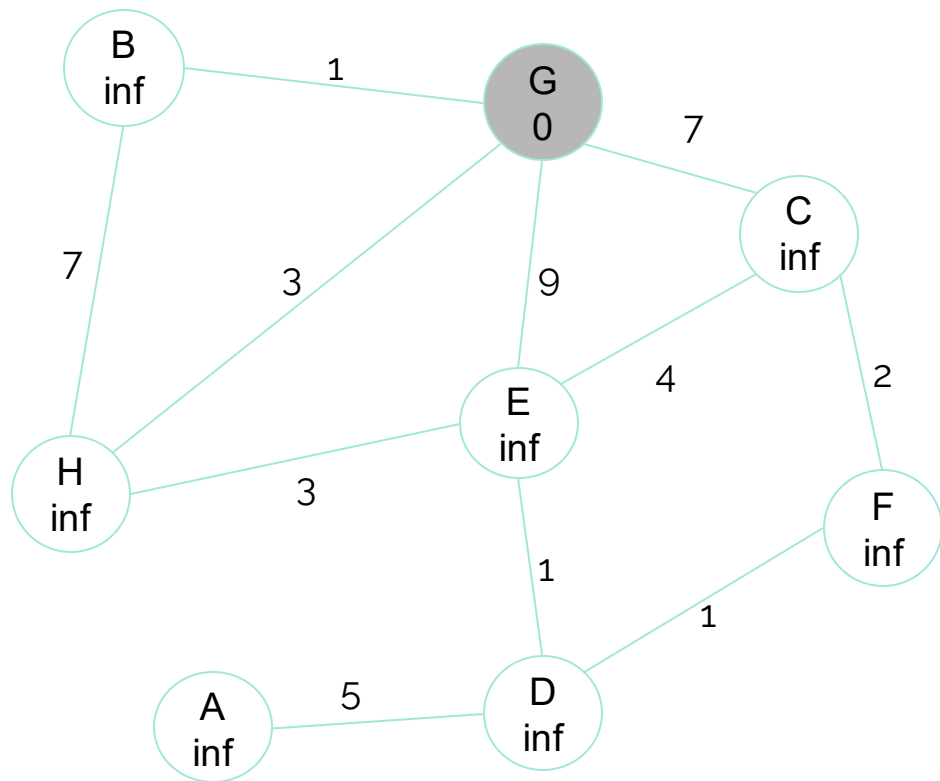
$Q = \{\}$

Una vez que la cola está vacía termina el algoritmo

¿Cómo podemos mejorar BFS para encontrar la ruta más corta?

- Cambiamos la cola por una cola de prioridad, que ordene las aristas según peso de menor a mayor, de tal forma que siempre revisemos primero posibles caminos más cortos
- Iniciamos todos los nodos con distancia infinita a nuestro nodo inicial, de tal forma que cada vez que lo visitamos revisamos si encontramos una distancia menor

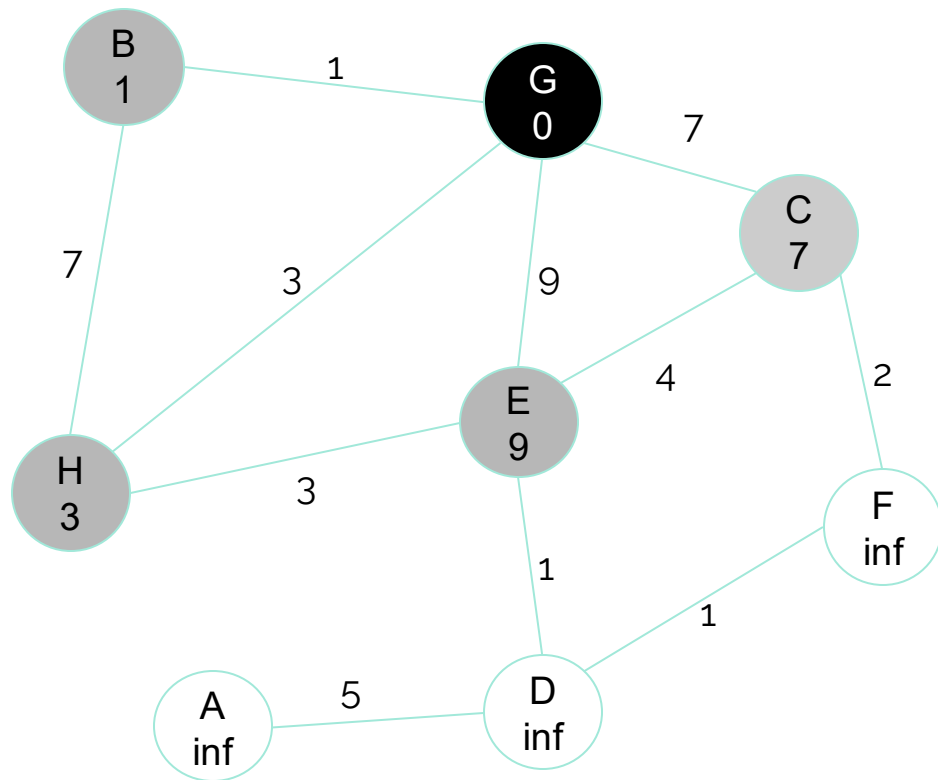
Dijkstra



Partimos desde el nodo G, similar a BFS
marcamos los nodos con los mismos
códigos de color

$PQ = \{(G, 0)\}$

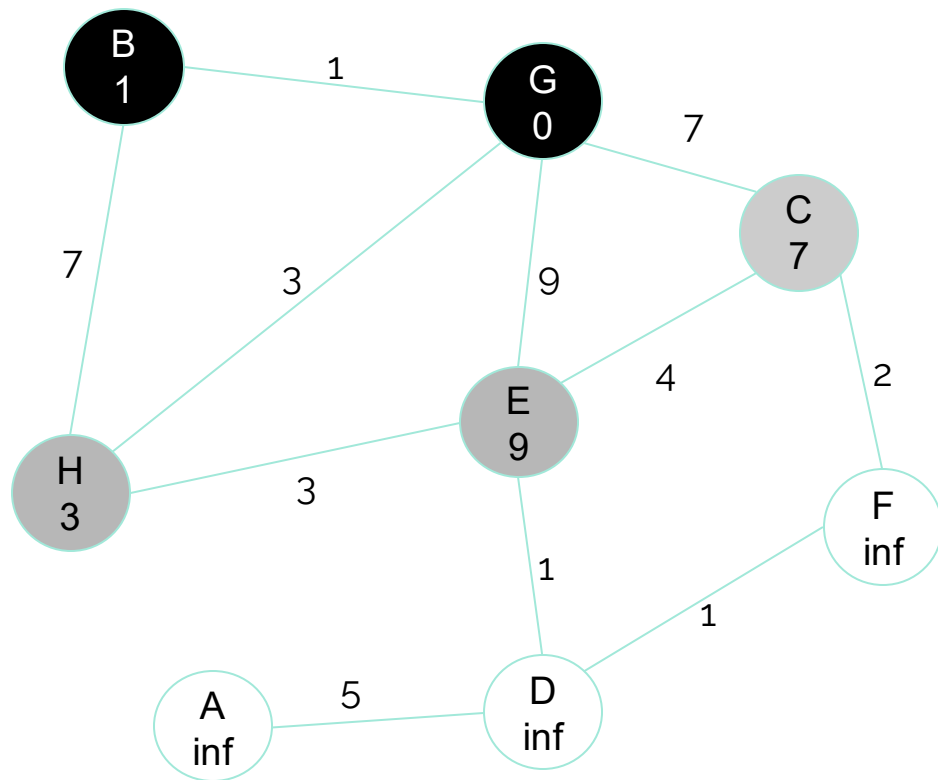
Dijkstra



Para calcular las distancias, sumamos el peso de la arista con la distancia que lleva acumulada el nodo

$PQ = \{(B, 1), (H, 3), (C, 7), (E, 9)\}$

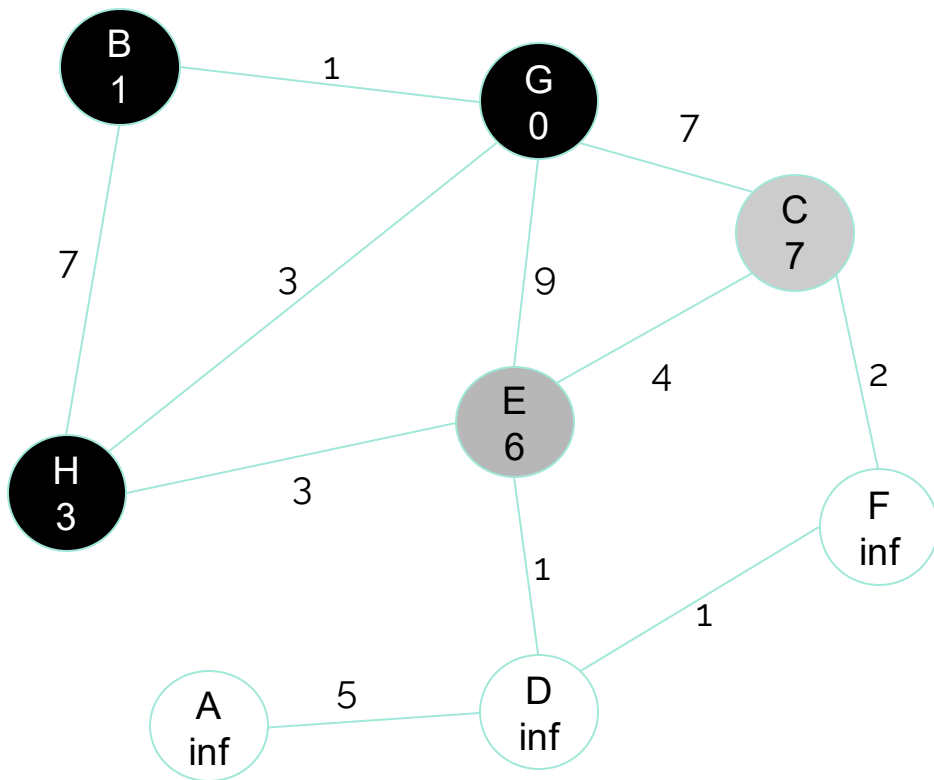
Dijkstra



Para este caso tenemos que la distancia de G a H pasando por B es mayor que la que teníamos

$PQ = \{(H, 3), (C, 7), (E, 9)\}$

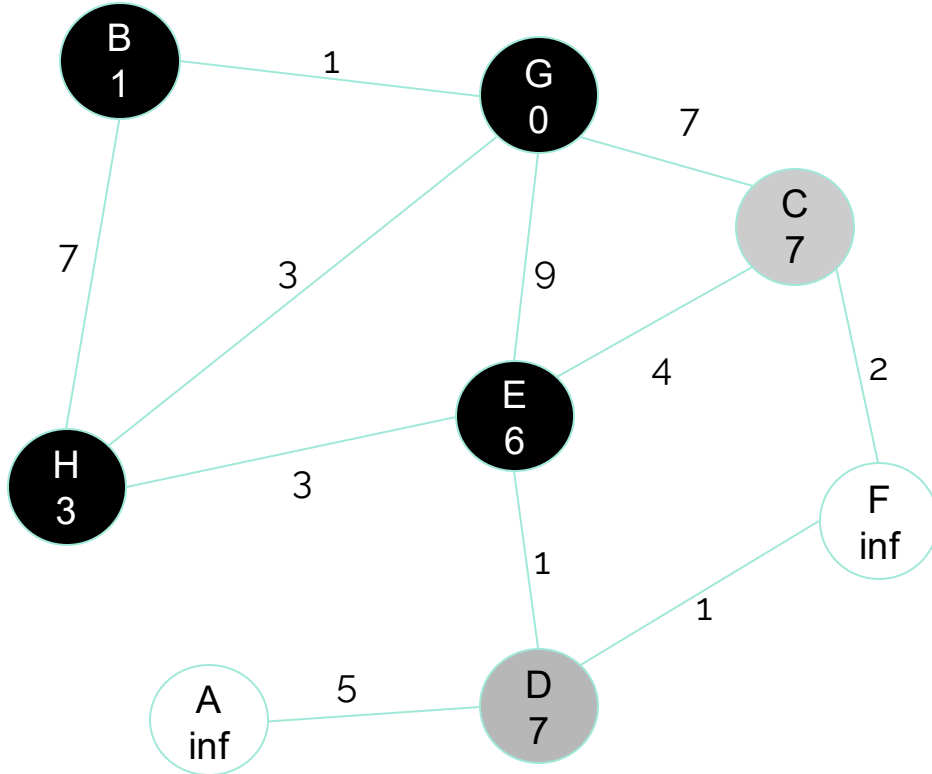
Dijkstra



Para este caso tenemos que la distancia de G a E pasando por H es menor que la que teníamos

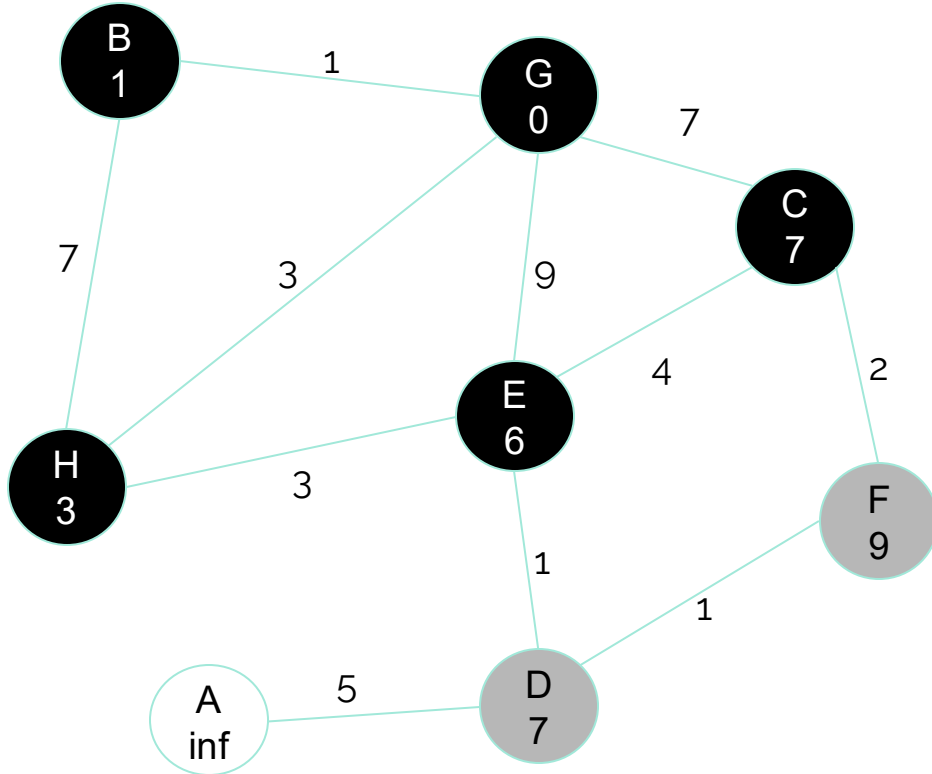
$PQ = \{(E, 6), (C, 7)\}$

Dijkstra



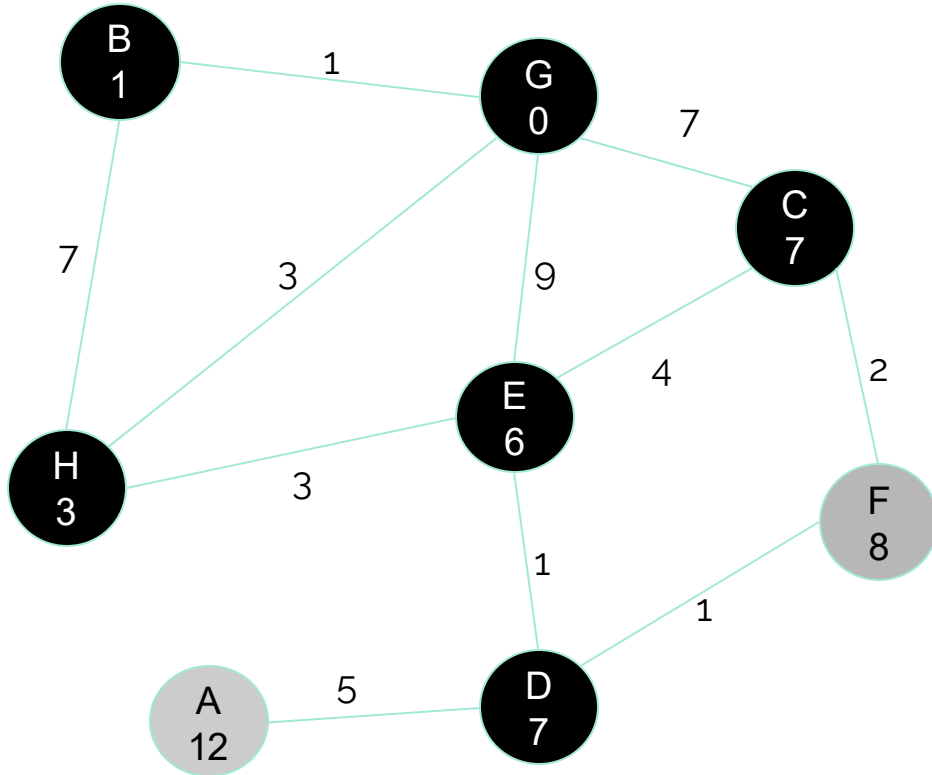
$PQ = \{(C, 7), (D, 7)\}$

Dijkstra



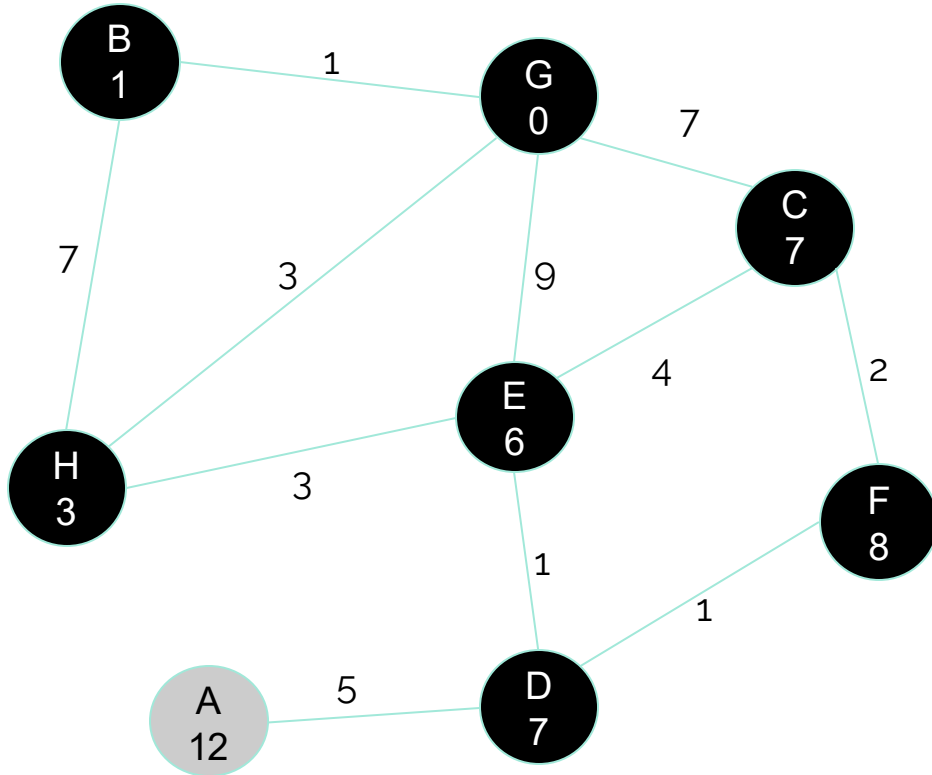
$PQ = \{(D, 7), (F, 9)\}$

Dijkstra



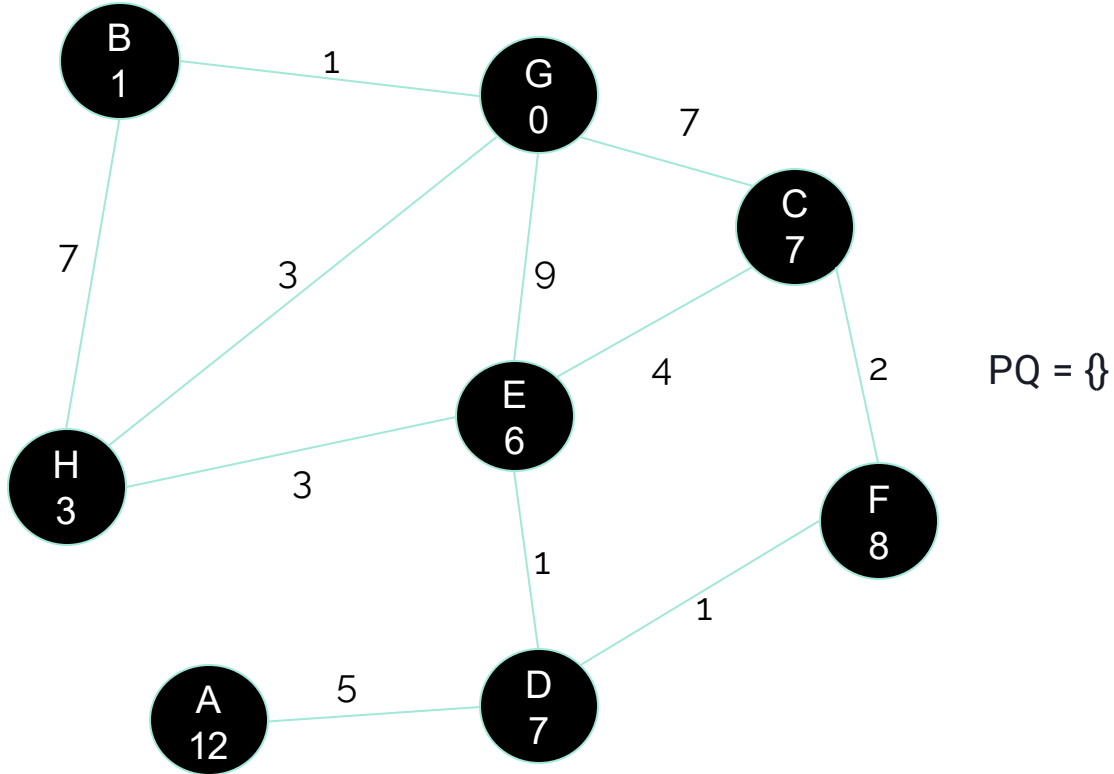
$PQ = \{(F, 8), (A, 12)\}$

Dijkstra



PQ = {(A, 12)}

Dijkstra



I3 2022-2 P4

- (b) [2 ptos.] El diámetro de un árbol no dirigido conexo T se define como el largo del camino más largo en T . Suponga que existe un único camino de largo máximo en T con extremos u y v . Si x es un nodo cualquiera, se puede demostrar que el nodo más lejano a x es u o v . Usando este resultado, proponga un algoritmo que determine el diámetro de un árbol T .

I3 2022-2 P4

- (b) [2 ptos.] El diámetro de un árbol no dirigido conexo T se define como el largo del camino más largo en T . Suponga que existe un único camino de largo máximo en T con extremos u y v . Si x es un nodo cualquiera, se puede demostrar que el nodo más lejano a x es u o v . Usando este resultado, proponga un algoritmo que determine el diámetro de un árbol T .

La propiedad descrita permite plantear el siguiente algoritmo

Diameter(V, E):

```
1   $x \leftarrow$  cualquier nodo de  $V$ 
2   $d \leftarrow \text{BFS}(x)$ 
3   $u \leftarrow$  nodo que tiene máximo  $d[\cdot]$ 
4   $d \leftarrow \text{BFS}(u)$ 
5   $D \leftarrow \max\{d[v] \mid v \in V\}$ 
6  return  $D$ 
```

donde se usa una versión modificada de **BFS** para que retorne el arreglo con distancias desde la fuente, así como el nodo asociado a cada distancia.

Bellman–Ford

A diferencia de los dos otros algoritmos vistos, Bellman-Ford permite resolver el problema de encontrar los caminos más cortos pero para grafos que permiten tener aristas con pesos negativos.

- Este algoritmo funciona con grafos dirigidos y aristas con pesos
- Detecta ciclos negativos

```

BellmanFord(s):
1   for  $u \in V$  :
2        $d[u] \leftarrow \infty$ ;  $\pi[u] \leftarrow \emptyset$ 
3    $d[s] \leftarrow 0$ 
4   for  $k = 1 \dots |V| - 1$  :
5       for  $(u, v) \in E$  :
6           if  $d[v] > d[u] + \text{cost}(u, v)$  :
7                $d[v] \leftarrow d[u] + \text{cost}(u, v)$ 
8                $\pi[v] \leftarrow u$ 

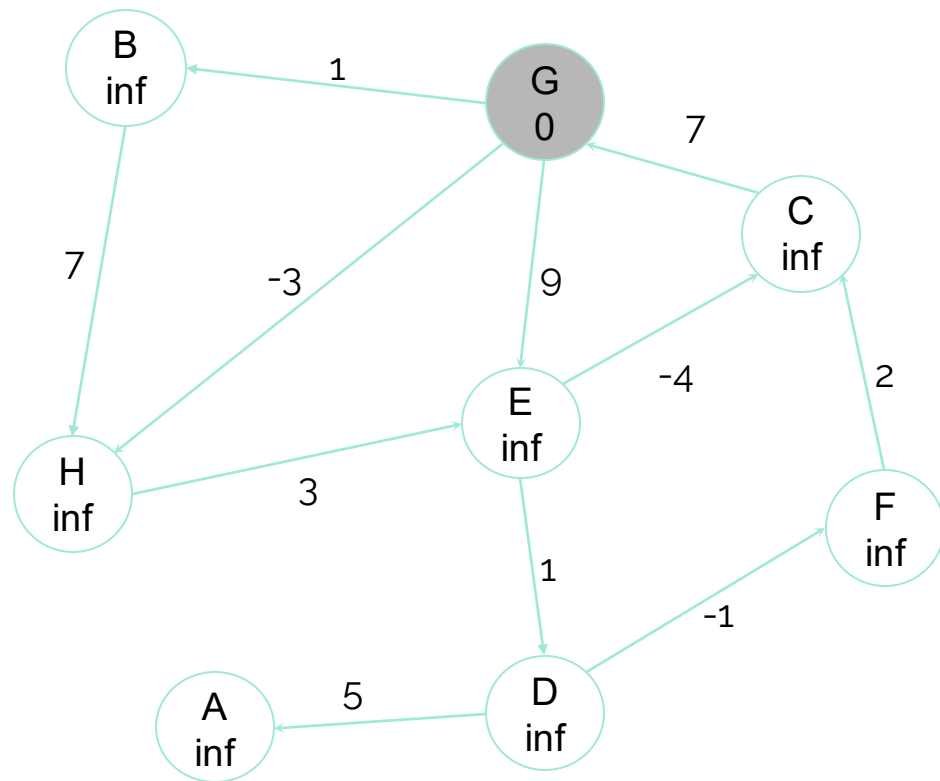
```

```

ValidBellmanFord(s):
1   for  $u \in V$  :
2        $d[u] \leftarrow \infty$ ;  $\pi[u] \leftarrow \emptyset$ 
3    $d[s] \leftarrow 0$ 
4   for  $k = 1 \dots |V| - 1$  :
5       for  $(u, v) \in E$  :
6           if  $d[v] > d[u] + \text{cost}(u, v)$  :
7                $d[v] \leftarrow d[u] + \text{cost}(u, v)$ 
8                $\pi[v] \leftarrow u$ 
9   for  $(u, v) \in E$  :
10      if  $d[v] > d[u] + \text{cost}(u, v)$  :
11          return false
12  return true

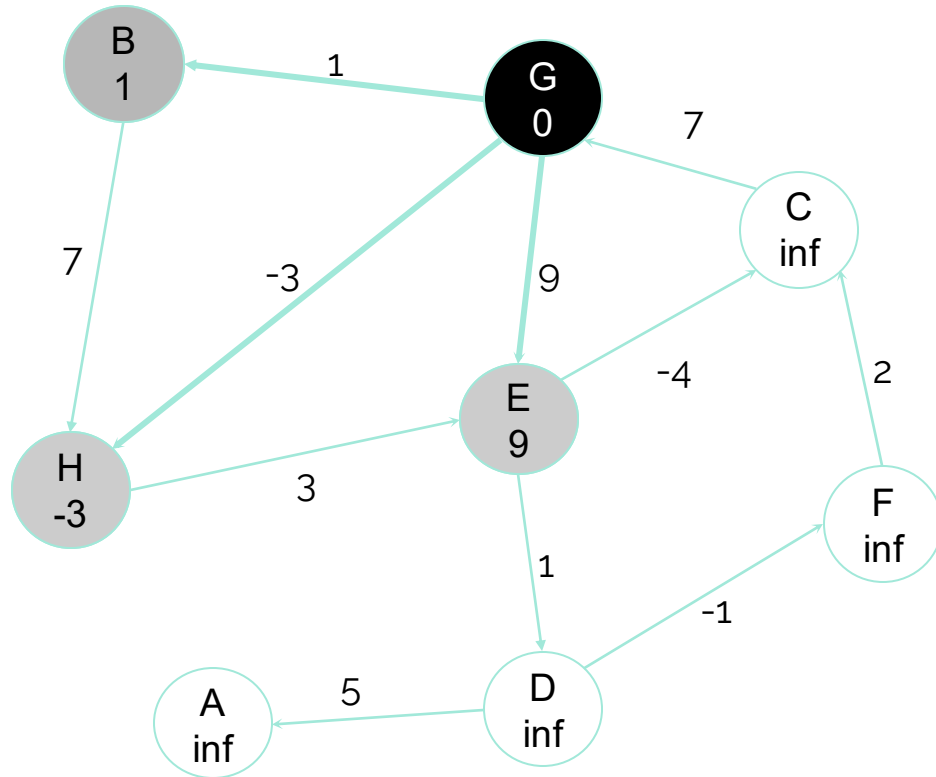
```

Bellman-Ford

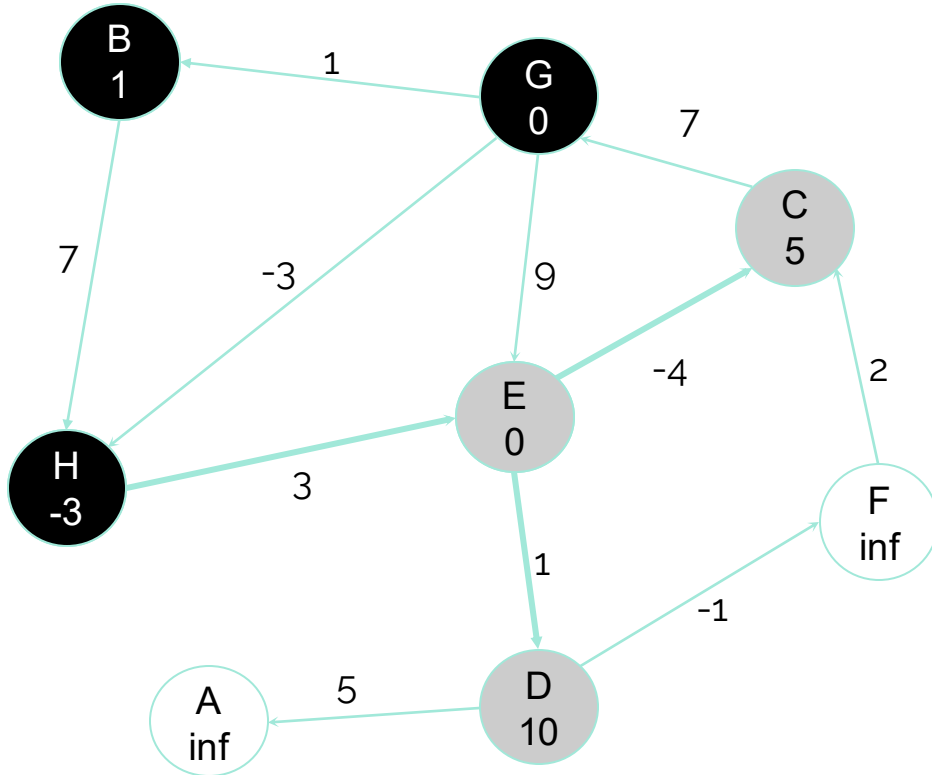


Notemos que ahora tenemos grafos dirigidos

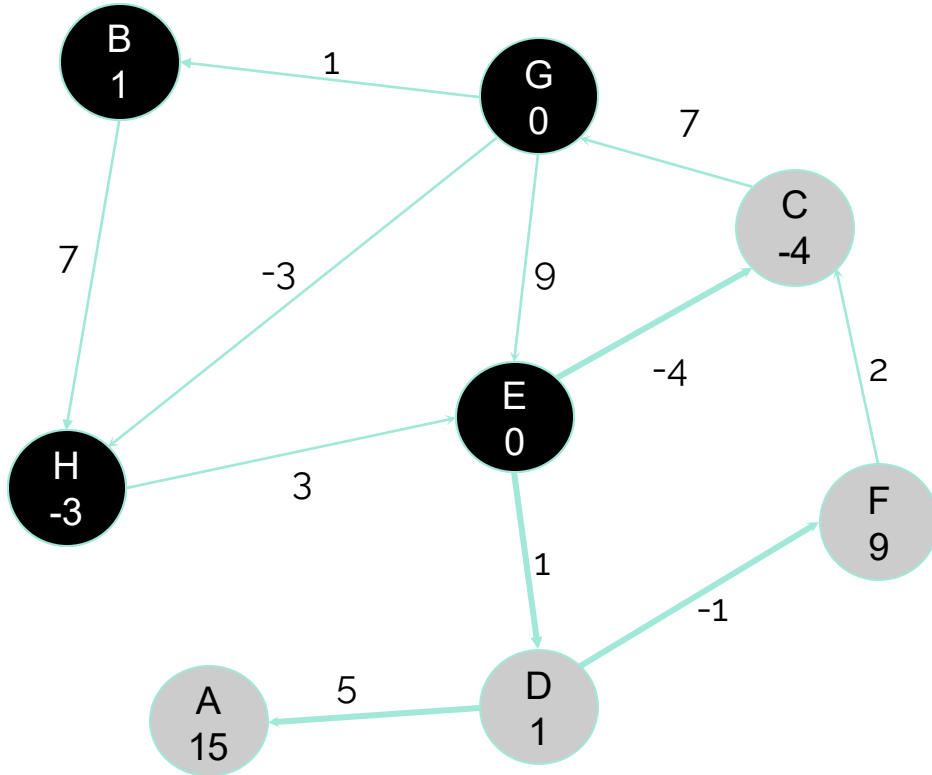
Bellman-Ford



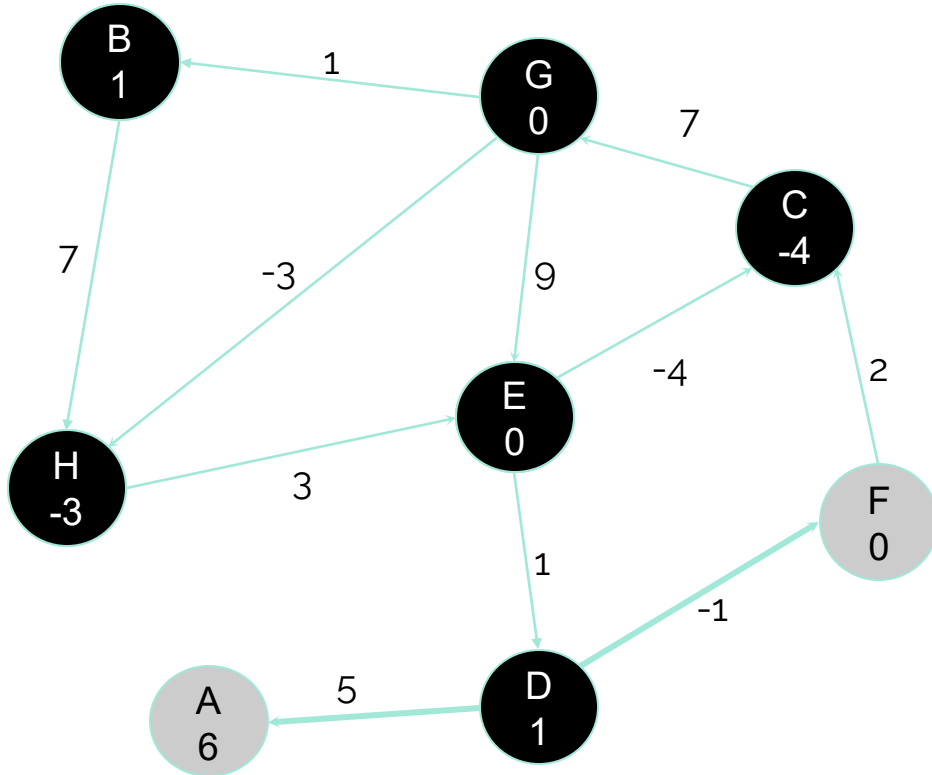
Bellman-Ford



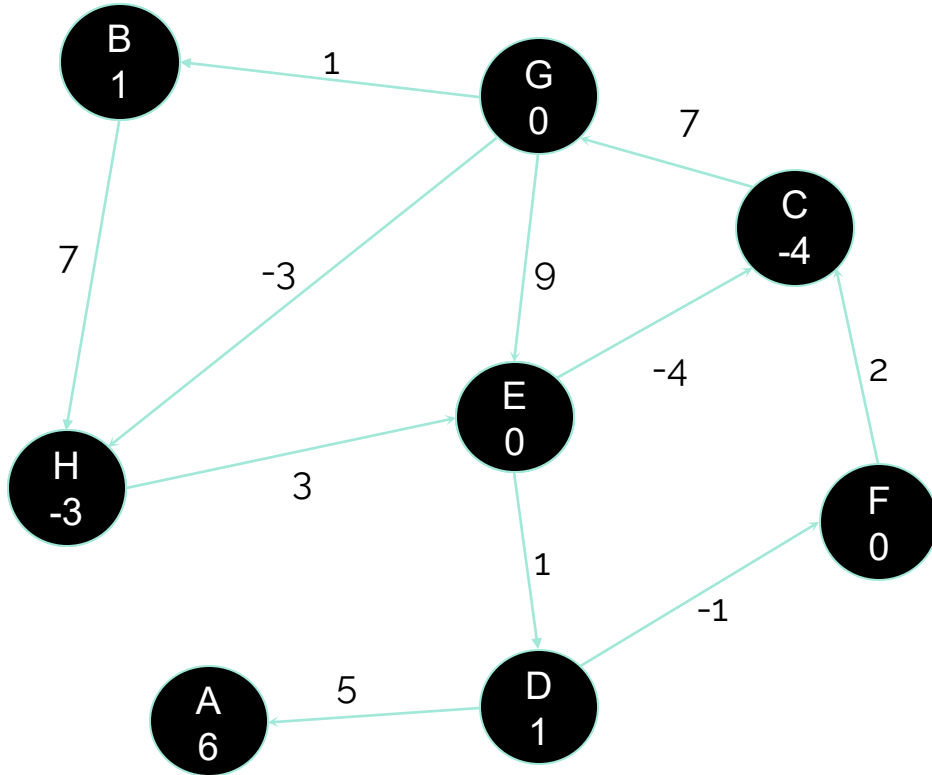
Bellman-Ford



Bellman-Ford

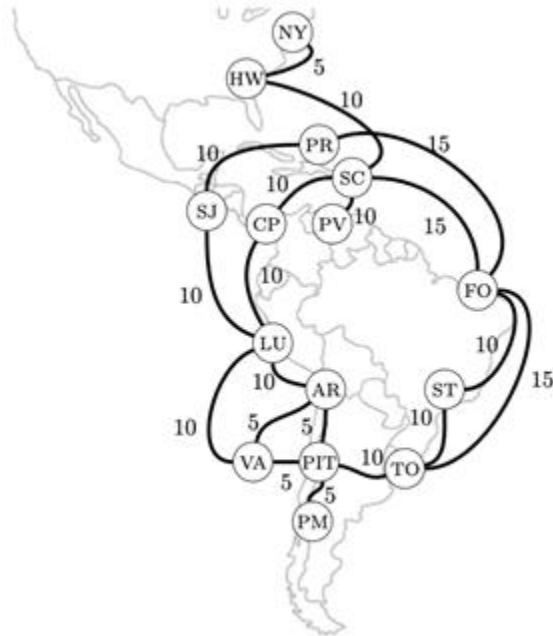


Bellman-Ford



I3 2023-2

La red de fibra óptica terrestre y submarina que conecta nuestros computadores puede verse como una grafo no dirigido. En este contexto, se pueden utilizar los protocolos OSPF y RIP para determinar las rutas a utilizar. Considere el siguiente fragmento de la red, donde se indican las latencias como costos en aristas.



NY	Brookhaven, NY, EEUU
HW	Hollywood, FL, EEUU
PR	San Juan, PR, EEUU
SC	St. Croix, Virgin Islands, EEUU
PV	Puerto Viejo, Venezuela
CP	Colón, Panamá
SJ	Puerto San José, Guatemala
FO	Fortaleza, Brazil
ST	Santos, Brazil
LU	Lurín, Perú
TO	Las Toninas, Argentina
AR	Arica, Chile
VA	Valparaíso, Chile
PM	Puerto Montt, Chile
PIT	Santiago, Chile

I3 2023-2

- (c) [2 ptos.] Considere que se quiere privilegiar el uso de las aristas (LU,CP) y (SJ,PR) cambiando su peso por -30 y -25 respectivamente. Determine si con este cambio es posible usar el algoritmo de Bellman-Ford para determinar la ruta más barata de Santiago a Lurín. Justifique su respuesta.



NY	Brookhaven, NY, EEUU
HW	Hollywood, FL, EEUU
PR	San Juan, PR, EEUU
SC	St. Croix, Virgin Islands, EEUU
PV	Puerto Viejo, Venezuela
CP	Colón, Panamá
SJ	Puerto San José, Guatemala
FO	Fortaleza, Brazil
ST	Santos, Brazil
LU	Lurín, Perú
TO	Las Toninas, Argentina
AR	Arica, Chile
VA	Valparaíso, Chile
PM	Puerto Montt, Chile
PIT	Santiago, Chile

I3 2023-2

- (c) [2 ptos.] Considere que se quiere privilegiar el uso de las aristas (LU,CP) y (SJ,PR) cambiando su peso por -30 y -25 respectivamente. Determine si con este cambio es posible usar el algoritmo de Bellman-Ford para determinar la ruta más barata de Santiago a Lurín. Justifique su respuesta.

Solución.

Al modificar las aristas indicadas, existe un ciclo de costo negativo dado por

LU, CP, SC, FO, PR, SJ, LU

que es alcanzable desde PIT. Luego, el algoritmo de Bellman-Ford no se puede utilizar para obtener un camino de costo mínimo desde PIT hasta LU.

Puntajes.

1.0 por indicar la existencia de ciclo de costo negativo.

1.0 por indicar que no se puede usar Bellman-Ford debido a ello.

I3 2023-1 P2 a)

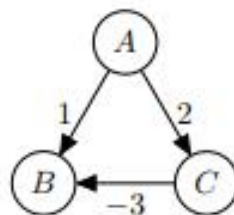
- (a) [3 ptos.] Demuestre que el algoritmo de Dijkstra no siempre es óptimo cuando el grafo dirigido sobre el que opera tiene costos negativos.

I3 2023-1 P2 a)

- (a) [3 ptos.] Demuestre que el algoritmo de Dijkstra no siempre es óptimo cuando el grafo dirigido sobre el que opera tiene costos negativos.

Solución.

Consideremos el grafo



El algoritmo de Dijkstra entrega como solución de A hasta B el camino A, B con costo 1, mientras que el óptimo real es el camino A, C, B con costo -1 .

I3 2023-1 P2 b)

(b) [3 ptos.] Sea G un grafo dirigido con costos.

(i) [2 ptos.] Proponga el pseudocódigo de un algoritmo que entregue una lista con los nodos de algún ciclo de costo negativo, en caso que G tenga tal ciclo. En caso contrario, retorna una lista vacía.

I3 2023-1 P2 b)

(b) [3 ptos.] Sea G un grafo dirigido con costos.

- (i) [2 ptos.] Proponga el pseudocódigo de un algoritmo que entregue una lista con los nodos de algún ciclo de costo negativo, en caso que G tenga tal ciclo. En caso contrario, retorna una lista vacía.

Solución.

Utilizando como estructura una cola FIFO implementada como lista ligada, los métodos son

NegativeCycle(s):

```
1  for  $u \in V$  :
2       $d[u] \leftarrow \infty$ ;  $\pi[u] \leftarrow \emptyset$ 
3   $d[s] \leftarrow 0$ 
4  for  $k = 1 \dots |V| - 1$  :
5      for  $(u, v) \in E$  :
6          if  $d[v] > d[u] + cost(u, v)$  :
7               $d[v] \leftarrow d[u] + cost(u, v)$ 
8               $\pi[v] \leftarrow u$ 
9  for  $(u, v) \in E$  :
10     if  $d[v] > d[u] + cost(u, v)$  :
11          $L \leftarrow$  lista con elemento  $v$ 
12          $current \leftarrow u$ 
13         while  $current \neq v$  :
14             agregar a  $L$  el nodo  $current$ 
15              $current \leftarrow \pi[current]$ 
16  return Lista vacía
```

Notemos que el algoritmo es esencialmente **ValidBellmanFord**, al cual se le cambia su retorno booleano. En caso que exista ciclo negativo (resultado **false** del algoritmo visto en clases), se construye la lista visitando ancestros del nodo que genera el ciclo. En caso contrario, se retorna lista vacía.

Ahora, este método solo indica ciclo negativo desde una fuente. Si existe un ciclo negativo, pero no es alcanzable desde un nodo específico, tenemos que asegurarnos de detectarlo. Para esto, se puede ejecutar este método desde todos los vértices. Es decir

NegativeCycleGlobal():

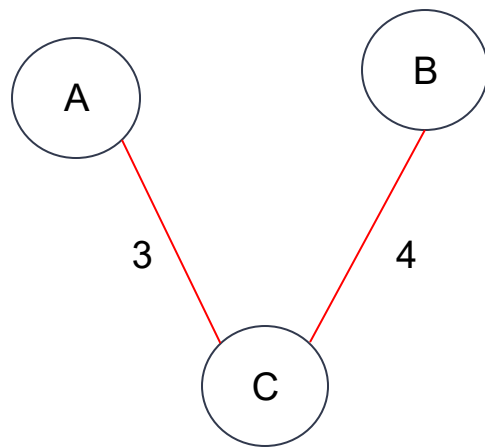
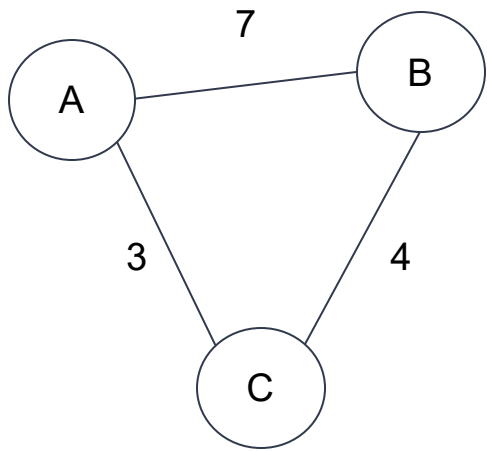
```
1  for  $u \in V$  :
2       $L \leftarrow$  NegativeCycle( $u$ )
3      if  $L \neq \emptyset$  :
4          return  $L$ 
5  return Lista vacía
```


MST (Minimum Spanning Tree)

Definición

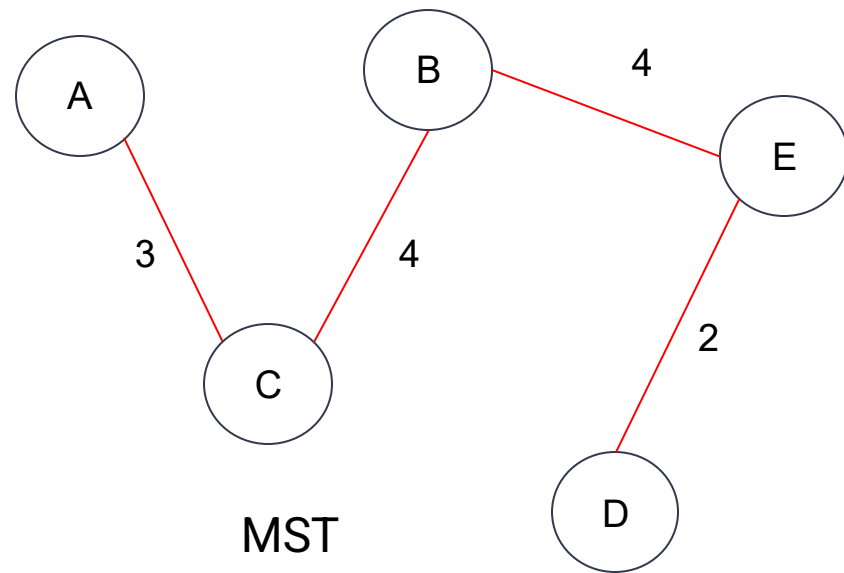
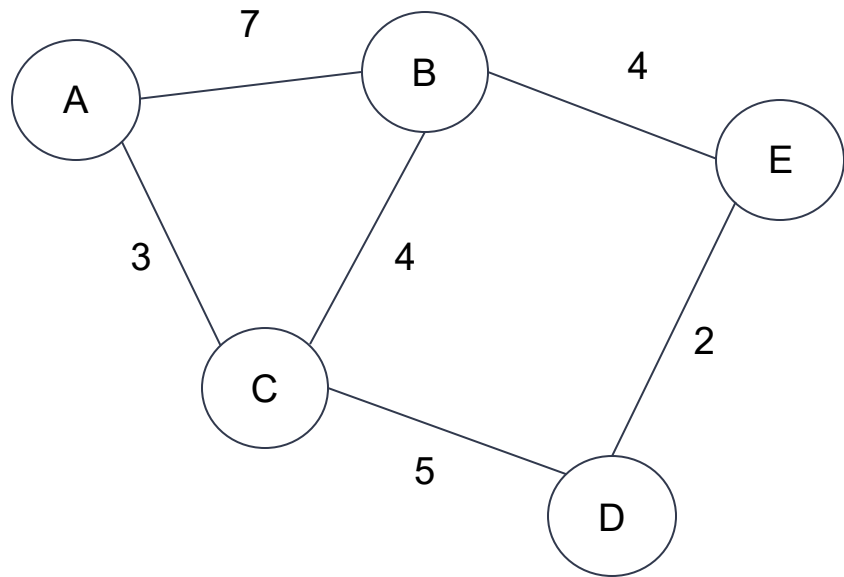
- Dado un grafo G no dirigido, un subgrafo T se dice MST de G si:
 - T es un árbol
 - $V(T) = V(G)$
 - No existe otro MST T' para G con menor costo total

MST ~ Ejemplo



MST
Costo = 7

MST ~ Ejemplo



MST
Costo = 13

MST (Minimum Spanning Tree)

- Con cada nodo que se agrega se vuelve más complicado encontrar “*visualmente*” el MST
- Necesitamos un algoritmo que resuelva ese problema por nosotros (Kruskal y PRIM)

KRUSKAL y Union Find

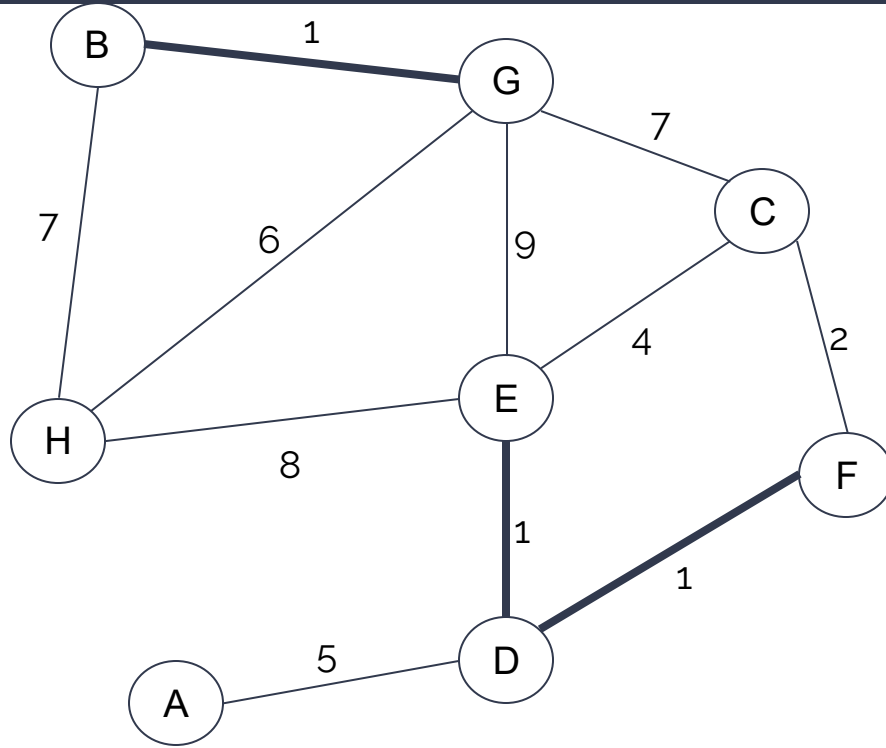
Idea base: Crear un bosque que converge en un único árbol

Sea $G = (V, E)$ grafo no dirigido iteramos sobre las aristas e en orden no decreciente de costo


1. Si e genera un ciclo al agregarla a T , la ignoramos
2. Si no genera ciclo, se agrega

¿Será necesario revisar **todas** las aristas de E ?

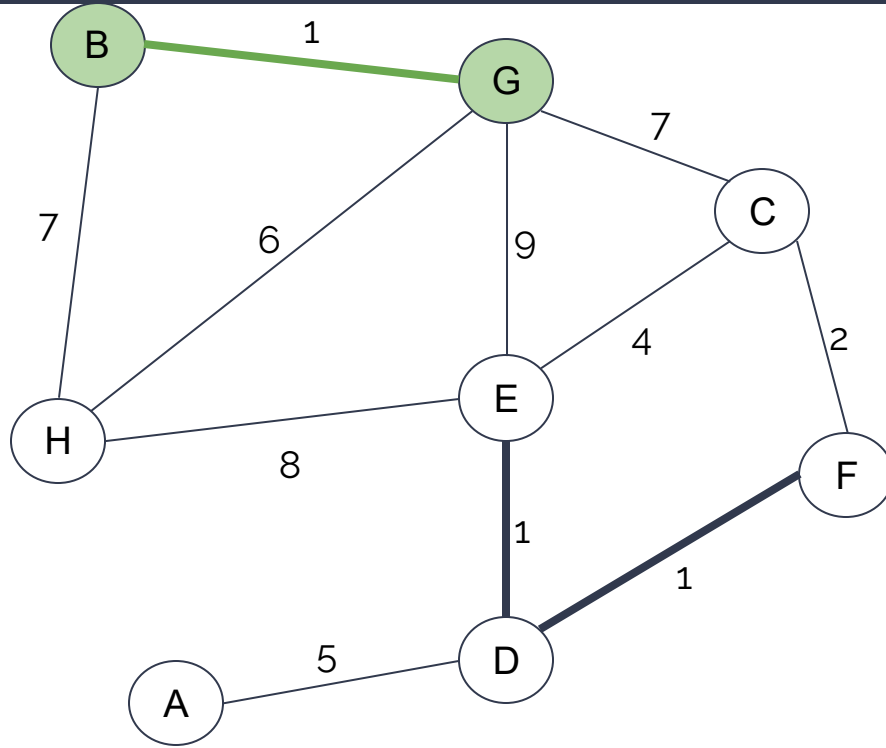
KRUSKAL y Union Find



 Aristas candidatas

 Aristas en MST

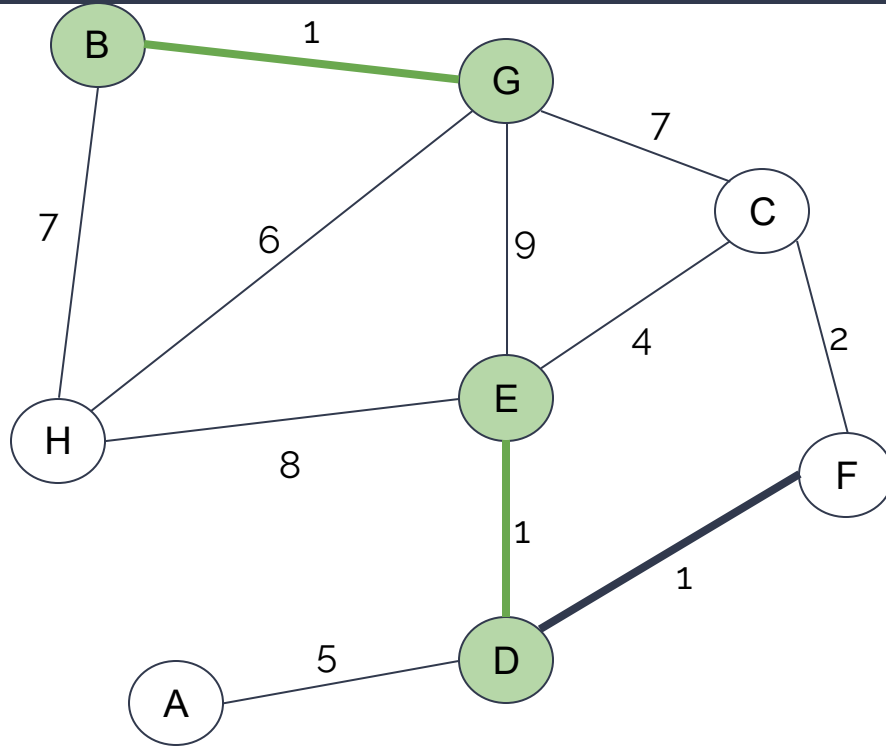
KRUSKAL y Union Find



— Aristas candidatas

— Aristas en MST

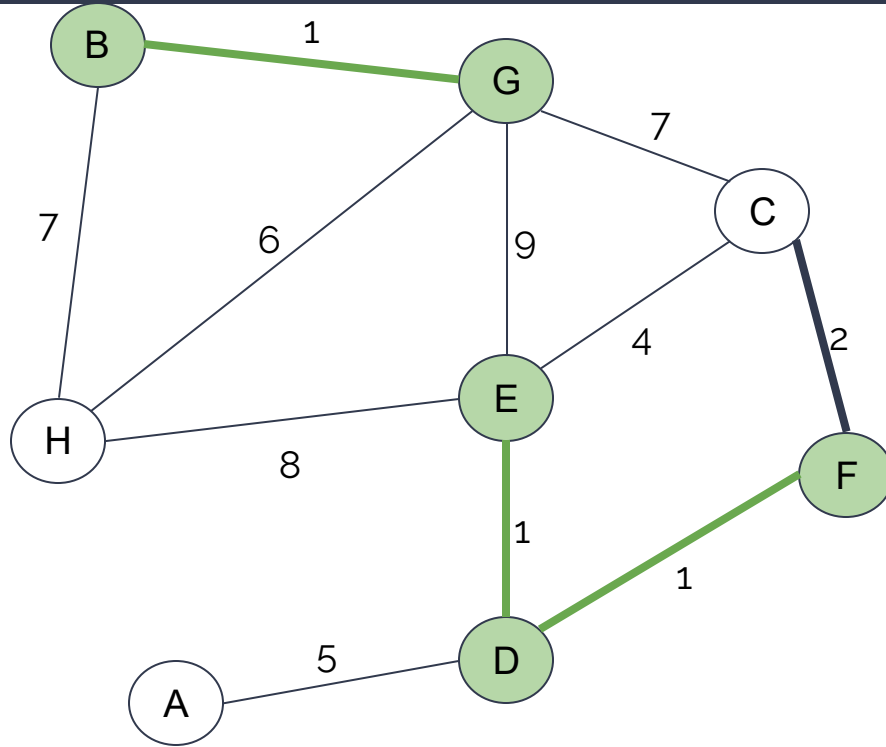
KRUSKAL y Union Find



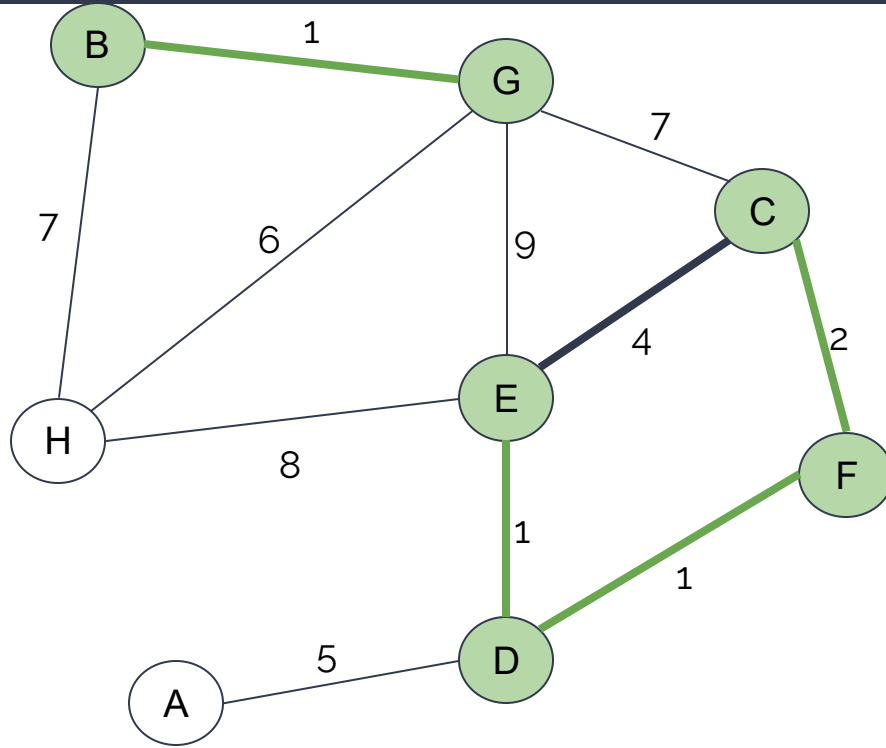
— Aristas candidatas

— Aristas en MST

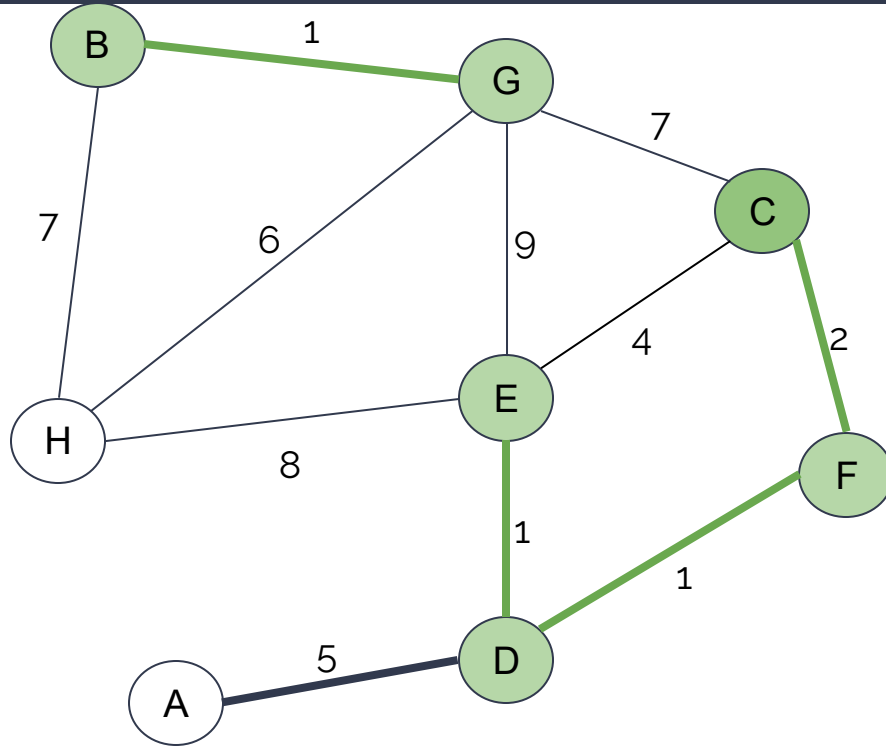
KRUSKAL y Union Find



KRUSKAL y Union Find



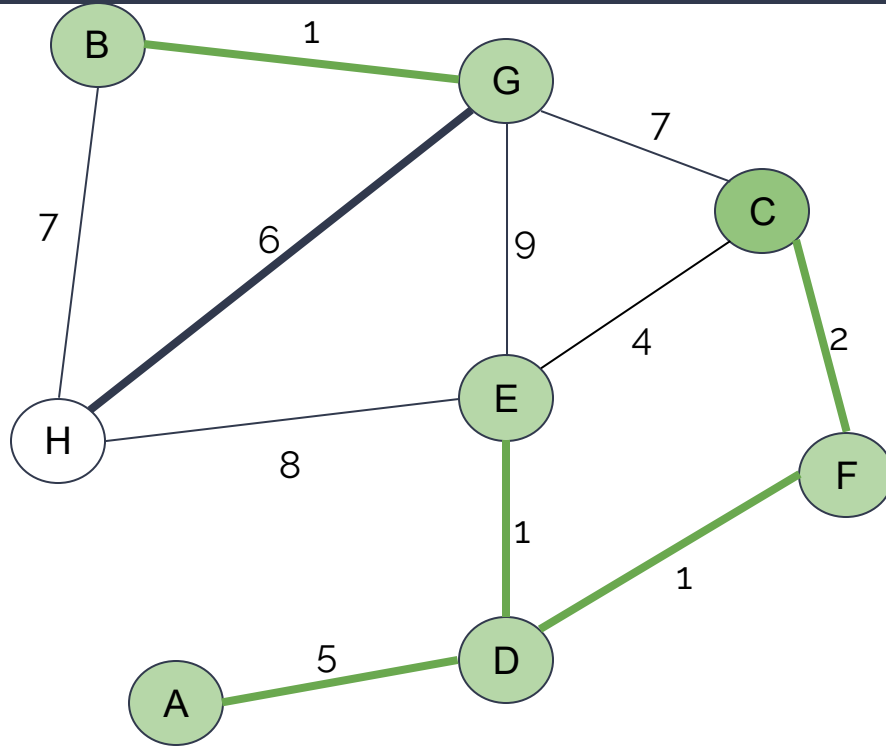
KRUSKAL y Union Find



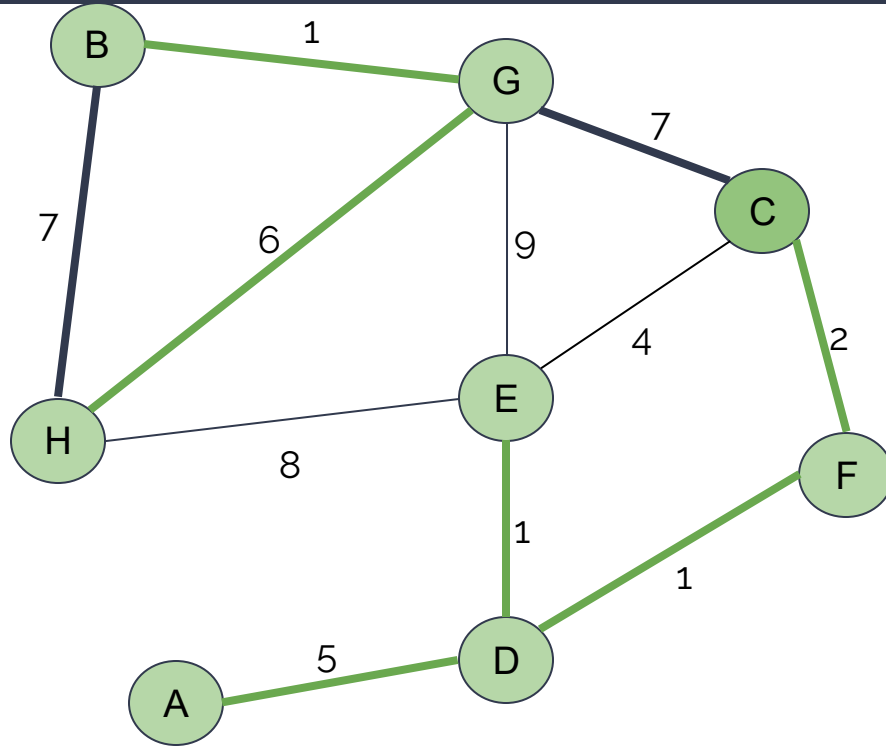
— Aristas candidatas

— Aristas en MST

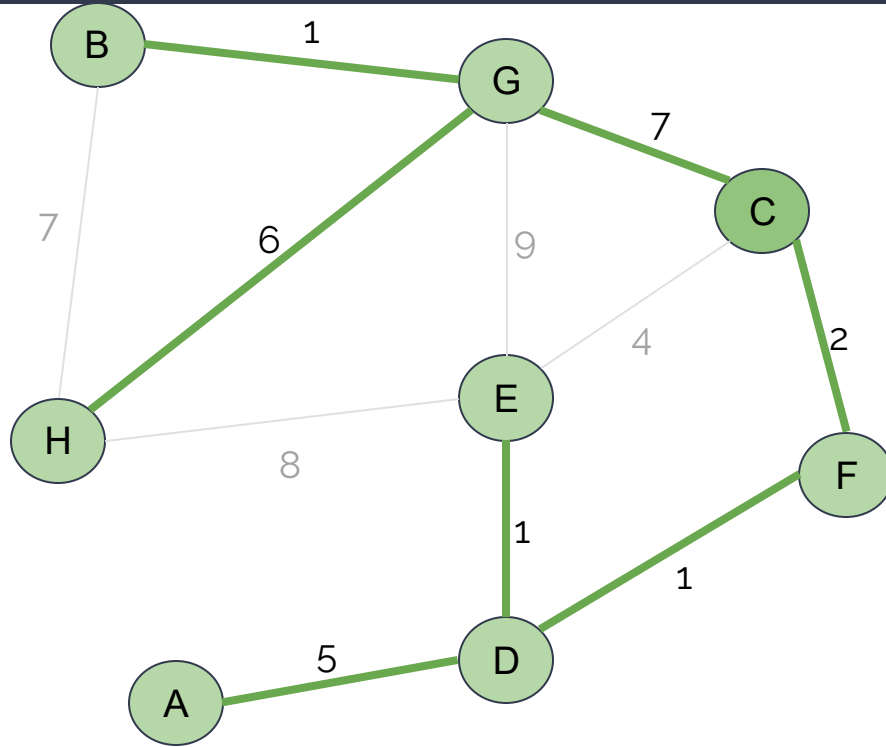
KRUSKAL y Union Find



KRUSKAL y Union Find



KRUSKAL y Union Find



 Aristas candidatas

 Aristas en MST

KRUSKAL y Union Find

¿Cómo detectamos si una arista produce o no un ciclo?

¡Conjuntos disjuntos!

1. Asignamos un set a cada nodo dentro del grafo
2. Cuando agregamos una arista unimos sus sets
3. Al revisar más aristas a futuro, sabemos que si dos nodos se encuentran en el mismo set, entonces ya existe un camino que los conecta, por lo que agregar esa arista nos generaría un ciclo

KRUSKAL

```
Kruskal( $G$ ):  
1    $E \leftarrow E$  ordenada por costo, de menor a mayor  
2   for  $v \in V$  :  
3       MakeSet( $v$ )  
4    $T \leftarrow$  lista vacía  
5   for  $(u, v) \in E$  :  
6       if Find( $u$ )  $\neq$  Find( $v$ ) :  
7            $T \leftarrow T \cup \{(u, v)\}$   
8           Union( $u, v$ )  
9   return  $T$ 
```

* Find(u) entrega el conjunto al que pertenece el nodo u

KRUSKAL y Union Find

```
kruskal(Grafo G)
  UnionFind T // se crea la estructura T
  E = G.edges
  sort(E) //Por el peso de menor a mayor
  for w,u,v in E
    if not T.same_set(u,v):
      T.join(u,v) //añadir arista u,v
```

$O(E \log E)$

I3 2023-1 P3

Una estrategia base para **asegurar cobertura de comunicaciones** es contar con **diferentes servicios de conexión**. Por ejemplo, el servicio de Tesorería (TGR) puede conectar sus oficinas y puntos de atención mediante:

Redes de fibra de proveedores, Redes de telefonía móvil, Red StarLink, Redes Satelitales, Radioaficionados y Banda Local.

Naturalmente, cada uno de ellos tienen diferentes costos, velocidades y latencias, y pueden estar disponibles o no entre diferentes oficinas y puntos de atención, o perderse al ocurrir una emergencia.



KRUSKAL y Union Find

a) ¿De qué forma puede TGR decidir **la mejor forma de conectar sus oficinas y puntos de atención** en función de los servicios de conexión disponibles en cada oficina o punto?

Indique la forma de representar el problema y las estrategias conocidas para determinar la solución.

Pista: Define primero como determinar la mejor conexión en cada caso.

KRUSKAL y Union Find

El problema se puede representar como un grafo no dirigido con costos en las aristas.

- Nodos: oficinas y puntos de atención
- Aristas: conexiones existentes entre nodos, con una arista por cada tipo de conexión disponible (puede haber más de una arista entre dos nodos).
- Costo de cada arista: función que relaciona costo, velocidad y latencia como un valor que describe el costo real de un tipo de conexión entre nodos.

La mejor forma de conectar, basándose en minimizar costo total, es determinar el MST de este grafo con algún algoritmo estudiado como Prim o Kruskal.

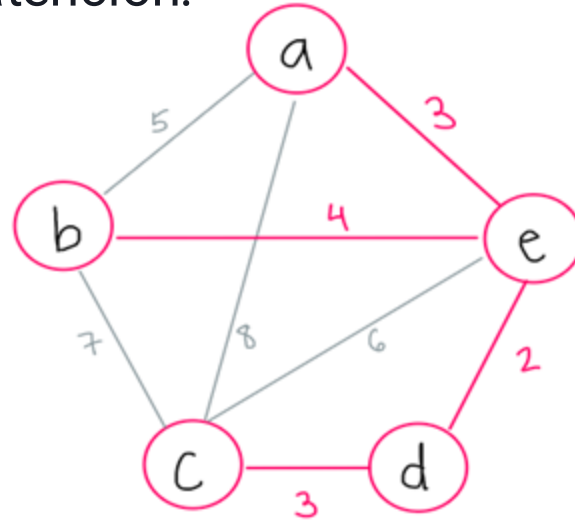
KRUSKAL y Union Find

b) Al ocurrir una emergencia, **TGR pierde la conectividad entre dos oficinas a través de uno de los enlaces que es parte de la mejor forma de conectar ya escogida**. Proponga una forma de recuperar la conectividad de la mejor forma disponible.

Justifique por qué es la mejor opción.

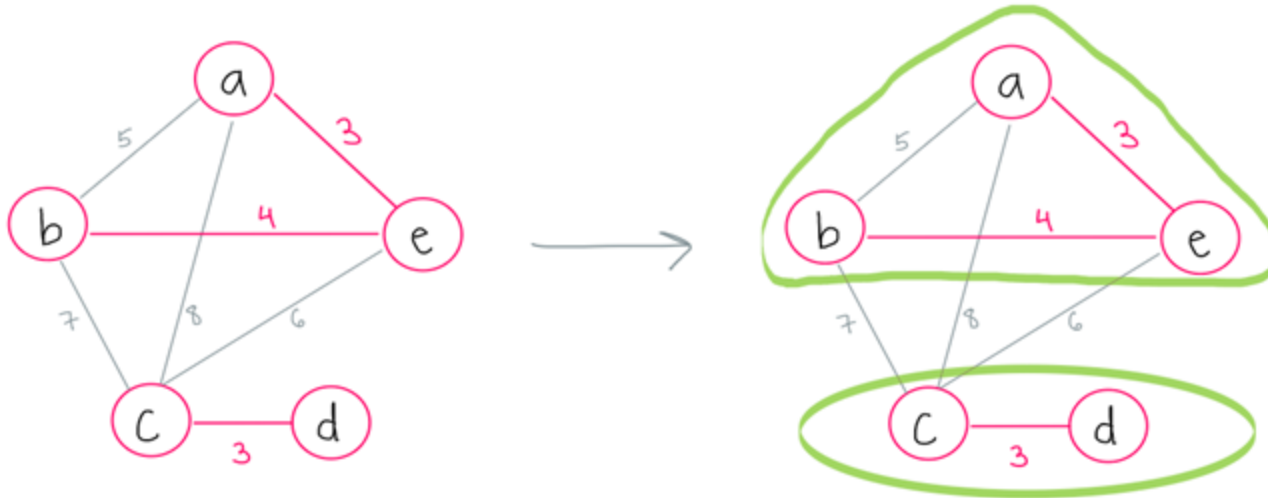
KRUSKAL y Union Find

Supongamos que este es el grafo que representa el MST de las oficinas y puntos de atención.



KRUSKAL y Union Find

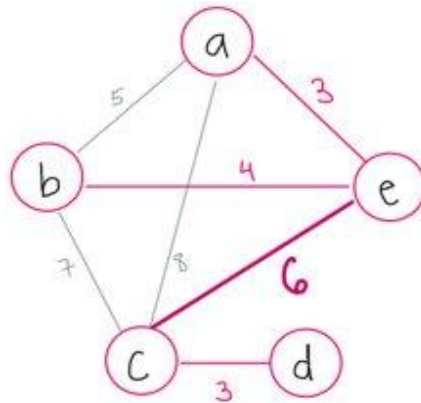
Al perder un enlace el MST del grafo se parte en dos MST independientes.



KRUSKAL y Union Find

Sabemos por la estrategia de construcción de los MST que ambos MST son óptimos para conectar los nodos que cubren.

Luego, si consideramos que lo que tenemos al perder la arista es un “corte” entre los MST (como en Kruskal) si determinamos la **siguiente arista de las disponibles que tiene el menor costo y no genera un ciclo** podemos recuperar el MST con las aristas disponibles.



KRUSKAL y Union Find

c) Si la emergencia **afecta simultáneamente a múltiples enlaces de la forma de conectar ya escogida**, ¿La estrategia que se propuso antes sigue siendo válida?

Argumente su respuesta.

KRUSKAL y Union Find

c) Si la emergencia **afecta simultáneamente a múltiples enlaces de la forma de conectar ya escogida**, ¿La estrategia que se propuso antes sigue siendo válida?

Sí, es posible generalizar la estrategia anterior cuando el MST del grafo pierde múltiples aristas que eran parte del MST. Podemos aplicar un enfoque similar para reconstruir el MST después de perder una arista.